



UHASSELT

KNOWLEDGE IN ACTION



Maastricht University

Doctoral dissertation submitted to obtain the degree of
Doctor of Sciences: Information Technology, to be defended by

Mannu Lambrichts

DOCTORAL DISSERTATION

Democratizing Prototyping of
Interactive Devices: A Unified
Plug-and-Play Platform for
Interconnecting and Driving
Heterogeneous Electronic
Components



UHASSELT

EDM

Promoter: Prof. Dr Raf Ramakers | UHasselt

D/2024/2451/67

Co-promoters: Prof. Dr Kris Luyten | UHasselt
Prof. Dr Steve Hodges | Lancaster University

ACKNOWLEDGEMENTS

I am deeply grateful to everyone who has supported and guided me throughout my PhD journey. This thesis would not have been possible without the contributions, encouragement, and support of many individuals and institutions who have helped shape both my research and my personal growth over these years.

First and foremost, I want to express my sincerest gratitude to my promotor, Prof. Dr. Raf Ramakers. Thank you for giving me the opportunity to pursue a PhD and for being a constant source of inspiration and guidance. From sparking my interest in Human-Computer Interaction (HCI) research during my master's studies to allowing me the freedom to explore my own research interests, your mentorship has been invaluable. I truly appreciate the trust you placed in me, the thoughtful advice you provided, and your patience in trying to make sense of my often unorganized ideas and thoughts. I am also grateful for the numerous opportunities you offered, from attending conferences to introducing me to key figures in the field. Your dedication to my growth, especially during our late-night work sessions before paper deadlines, is a testament to your commitment to my success. I couldn't have asked for a more supportive and understanding supervisor.

I am also deeply thankful to my co-promotor, Prof. Dr. Kris Luyten, for his insightful advice and unwavering support throughout my PhD. Your guidance has been instrumental in shaping my research, and I am grateful for the time you invested in helping me. Your method of guiding people truly reflects a deep commitment to understanding their individual needs and fostering their growth.

My heartfelt thanks go to Prof. Dr. Steve Hodges, who joined as my second co-promotor midway through my PhD. While your guidance in developing my electronic prototypes was incredibly helpful, I have learned much more from you in various aspects of research and innovation. Your advice, insights, and the connections you facilitated within the broader research community have greatly enriched my experience and contributed to my growth as a researcher. I am truly grateful for all the knowledge and opportunities you have shared with me.

I would also like to thank Prof. Dr. Rong-Hao Liang for being a member of my PhD committee and for offering valuable feedback throughout my research journey. Your insights have been incredibly helpful in refining my work, and I am truly grateful for

your thoughtful input and encouragement.

I am also deeply appreciative of the members of my PhD jury, Prof. Dr. Céline Coutrix and Prof. Dr. Aluna Everitt, for their time and thoughtful feedback on this dissertation. Your constructive comments have significantly improved my work, and I am grateful for the effort you put into reviewing my research, as well as your commitment to my PhD defense.

Throughout my PhD, I have had the privilege of working with many inspiring researchers and colleagues at the EDM. Room 0.02, in particular, will always hold a special place in my heart. I am especially thankful to Dr. Tom Veuskens, who began his PhD around the same time as I did. Tom, our conversations, whether about research, the ups and downs of the PhD journey, or life in general, have been a constant source of support and camaraderie. I am also grateful to Dr. Danny Leen for all our discussions and his valuable input on my—often very chaotic—ideas. I would also like to thank him for helping to improve the design of the figures used in my papers. I also want to thank Dries Cardinaels, for bringing fresh insights and perspectives to our discussions. Your enthusiasm and new ideas have been a breath of fresh air.

Beyond room 0.02, I want to thank Joris Herbots, Jeroen Ceyssens, and Dr. Bram Vandeuren for all the insightful conversations and support, both in research and on a personal level. Each of you has played a significant role in helping me navigate the challenges of completing a PhD, and I am deeply grateful for your friendship and advice. A special thanks goes to Joris for accompanying me through the last ten years, from our bachelor's and master's studies to the PhD journey. With your help and companionship, every step of this journey has been more manageable and memorable.

I am incredibly grateful for the wonderful work environment I experienced during my PhD. It has been a privilege to collaborate with so many people from diverse research areas who are always willing to help and work together. The openness and collaborative spirit within EDM have made this journey not just productive but also enjoyable. I cherish the friendships and memories we have created along the way.

On a professional note, I would like to acknowledge Dr. James Devine for his guidance in understanding the Jacdac protocol and Lorraine Underwood and Prof. Dr. Joe Finney for their efforts in conducting a preliminary user study on the CircuitGlue concept. Dr. Sven Coppers provided significant assistance with the statistics for the toolkit classification survey, and Prof. Dr. Davy Vanacken offered valuable feedback on papers, even when asked at the last minute. I am also grateful to all the individuals I met at conferences who shared their ideas and helped me broaden my horizons.

On a personal note, I cannot express enough gratitude to my partner, Delphine Tweepeninx. Delphine, you have been my rock throughout this entire journey, always believing in me and pushing me to do my best, even during the toughest times. Your unwavering support and love have been my guiding light, helping me through every

struggle and every long hour spent working. I am so grateful for your encouragement, your patience, and for always standing by my side. Thank you for everything, Delphine.

I would also like to thank my parents, my brother Willem Lambrichts, and my entire family for their constant belief in me and their endless support. Your encouragement has been a source of strength and motivation, and I am deeply grateful for all the love and support you have given me.

Finally, a special thanks to my friends Jorrit Gerets and Maarten Vangeneugden for their advice and feedback on the many challenges I faced during this journey. Your friendship and support have been invaluable, and I am thankful for the many conversations that helped me stay on track.

To everyone who has been a part of this journey, thank you from the bottom of my heart. This thesis is a testament to your support, encouragement, and belief in me. I couldn't have done it without you.

ABSTRACT

In the evolving landscape of electronics prototyping, a critical challenge has emerged: bridging the divide between the expectations of creators and the practical capabilities of the available prototyping tools. This dissertation seeks to tackle this challenge, aiming to harmonize the varied facets of the prototyping ecosystem. This approach goes beyond merely cataloging available tools; it seeks to find common ground—a unified understanding that encompasses the wide array of prototyping methodologies, toolkit functionalities, and user experiences.

My endeavor to find common ground begins with mapping the current electronics prototyping ecosystem. This involves a deep dive into the plethora of toolkits available to individuals, examining not only their technical specifications but also their design philosophies, intended user groups, and practical applications. The aim is to distill this rich tapestry of information into a cohesive framework that simplifies the decision-making process for users, enabling them to select tools that best align with their project goals and skill levels.

Further, recognizing the diversity of the prototyping community, this dissertation explores the varied needs and strategies of users. From hobbyists working on DIY projects to professional engineers developing complex systems, each user brings unique perspectives and requirements. By conducting an online survey, I gather firsthand insights into the challenges users face, the tools they prefer, and the strategies they employ to realize their visions. This comprehensive understanding is crucial for developing tailored solutions that address unmet needs and enhance the efficiency of prototyping practices.

The process of finding common ground also involves the development of integrated hardware-software solutions to streamline prototyping workflows. The challenges of compatibility and the technical complexities of integrating disparate components are significant hurdles for many individuals. By introducing innovative solutions that facilitate seamless interaction between software applications and hardware devices, I aim to simplify the prototyping process, making it more accessible and enjoyable for creators of all backgrounds.

In summary, “Finding Common Ground” is not just about cataloging tools or identifying user preferences; it’s about weaving together the diverse threads of the electronics

prototyping world into a coherent narrative. By achieving this, I hope to pave the way for future innovations in electronics toolkit offerings, making prototyping more accessible, efficient, and fulfilling for individuals everywhere.

LIST OF SCIENTIFIC CONTRIBUTIONS

This dissertation builds upon a series of academic publications and presentations that have been recognized within the Human-Computer Interaction (HCI) community. These works, which have appeared in scientific journals and been presented at international conferences, collectively contribute to the ongoing discourse in the HCI field.

This dissertation is an extension of the following academic publications in scientific journals and presented at international conferences in the field of Human-Computer Interaction (HCI):

1. **Mannu Lambrichts**, Raf Ramakers, Steve Hodges, Sven Coppers, and James Devine. 2021. *A Survey and Taxonomy of Electronics Toolkits for Interactive and Ubiquitous Device Prototyping*. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 5, 2, Article 70 (June 2021), 24 pages. (Best Presentation Award)
2. **Mannu Lambrichts**, Raf Ramakers, Steve Hodges, James Devine, Lorraine Underwood, and Joe Finney. 2023. *CircuitGlue: A Software Configurable Converter for Interconnecting Multiple Heterogeneous Electronic Components*. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 7, 2, Article 63 (June 2023), 30 pages.
3. **Mannu Lambrichts**, Raf Ramakers, Steve Hodges. 2024. *LogicGlue: Hardware-Independent Embedded Programming Through Platform-Independent Drivers*. Proc. ACM Hum.-Comput. Interact., EICS. [Manuscript submitted for publication]

In addition to the main contributions listed in the previous section, I have contributed to a number of publications that have not been directly incorporated into this dissertation:

1. **Mannu Lambrichts**, Jose Maria Tijerina, and Raf Ramakers. 2020. *SoftMod: A Soft Modular Plug-and-Play Kit for Prototyping Electronic Systems*. In Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '20). Association for Computing Machinery, New York, NY, USA, 287–298. (as the result of my master's thesis)
2. **Mannu Lambrichts**, Jose Maria Tijerina, Tom De Weyer, and Raf Ramakers. 2020. *DIY Fabrication of High-Performance Multi-Layered Flexible PCBs*. In Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '20). Association for Computing Machinery, New York, NY, USA,

565–571. (as the result of my master’s thesis)

3. Andrea Bianchi, Steve Hodges, David J. Cuartielles, Hyunjoo Oh, **Mannu Lambrichts**, and Anne Roudaut. 2023. *Beyond Prototyping Boards: Future Paradigms for Electronics Toolkits*. In Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems (CHI EA ’23). Association for Computing Machinery, New York, NY, USA, Article 333, 1–6.

I also demonstrated some of the work presented in this dissertation:

1. **Mannu Lambrichts**, Raf Ramakers, Steve Hodges. 2022. *Using Electronic Toolkits: Learning and Combining their Properties*. Aachen Maker Meetup, remote.
2. **Mannu Lambrichts**, Raf Ramakers, Steve Hodges. 2022. *Using Electronic Toolkits: Learning and Combining their Properties*. Sketching in Hardware, Dublin, Ireland.
3. **Mannu Lambrichts**, Raf Ramakers, Steve Hodges. 2023. *Plug-and-Play Electronics by CircuitGlue*. Poster at the 2023 Flanders Make Scientific Conference.

CONTENTS

Acknowledgements	i
Abstract	vi
List of Scientific Contributions	viii
Contents	x
List of Figures	xiv
List of Tables	xx
Acronyms	xxiii
1 Introduction	2
1.1 Introduction to Physical Computing	2
1.2 Interaction Modalities in Physical Computing	3
1.3 Exploring the Diversity of Physical Computing Platforms	4
1.3.1 Assembling Hardware: A Tangible Foundation	4
1.3.2 Developing Software: Bringing Hardware to Life	5
1.3.3 Bridging Hardware and Software: The Role of Hardware and Software Glue	6
1.4 Research Questions	8
1.5 Research Goals	9
1.6 Contributions	10
1.7 Dissertation Outline	12
2 Established Approaches to Electronics Prototyping	14
2.1 Introduction	14
2.2 Types of Prototyping	15
2.3 Bridging Between Types	21
2.4 Taking a look into the Future	23
3 Finding Common Ground	24
3.1 Introduction	25

3.2	Identifying and Reviewing the Literature	26
3.2.1	What is an Electronics Prototyping Toolkit?	26
3.2.2	Corpus of Products and Publications	27
3.2.3	Characteristics	27
3.2.4	Data Points and Clusters	27
3.3	Electronic Prototyping Platform Taxonomy	28
3.3.1	Nature and Application	28
3.3.2	Assembly of Prototypes	29
3.3.3	Deploying and Configuring	31
3.3.4	Availability and Adoption	31
3.4	Analyzing the characteristics	32
3.5	Understanding How Electronics Toolkits are Used	36
3.5.1	Our Respondents and Their Prototyping Experience	37
3.5.2	Use of Prototyping Platforms	38
3.5.3	Important Characteristics of Prototyping Platforms	39
3.5.4	Experiences of Type 1 Prototyping and Scaling Up to Multiple Copies	40
3.6	Discussion	42
3.7	Summary	45
4	Plug-and-Play Hardware Through CircuitGlue	46
4.1	Introduction	47
4.2	Walkthrough	49
4.3	Related Work	52
4.3.1	Modules for Electronics Prototyping	53
4.3.2	Tools to Ease Breadboarding and Development	53
4.3.3	Reprogrammable Integrated Circuits	54
4.4	Design Rationale	55
4.4.1	Early Feedback on the CircuitGlue Concept	55
4.4.2	Design Decisions	56
4.4.3	Jacdac as Bus Protocol	57
4.5	Supporting New Modules	57
4.6	CircuitGlue Hardware Design	60
4.6.1	System-on-Chip	60
4.6.2	Regulating Power	60
4.6.3	Changing the Assignment of a Programmable Header Pin	62
4.6.4	Circuit Board Design and Manufacturing	65
4.7	CircuitGlue Software Architecture	66
4.7.1	CircuitGlue Firmware	66
4.7.2	CircuitGlue Configuration Tool	66
4.7.3	Circuit Diagram Generator	67
4.8	Prototyping Styles and Benefits	68

4.8.1	Understanding, Testing and Comparing Modules	68
4.8.2	Rapid Prototyping with Heterogeneous Modules	68
4.8.3	Facilitate Breadboarding	70
4.8.4	Use Third Party Modules with Jacdac Ecosystem	70
4.8.5	Advanced Use	71
4.9	Technical Benchmark	72
4.10	Preliminary User Evaluation	74
4.10.1	Participants	75
4.10.2	Procedure	75
4.10.3	Results	76
4.11	Incorporating User Feedback	78
4.11.1	Modular PCB Design	78
4.11.2	Power Management	79
4.11.3	Changing Pin Assignments	80
4.11.4	Enhanced Visual Feedback	83
4.11.5	CircuitGlue Configuration Tool	84
4.12	Discussion and Future Work	84
4.13	Summary	86
5	Plug-and-Play Software Drivers Through LogicGlue	88
5.1	Introduction	89
5.2	LogicGlue	92
5.2.1	Writing Application Logic	92
5.2.2	Writing Driver Specifications	94
5.3	Related Work	96
5.3.1	Software Abstraction	96
5.3.2	Standardized Communication Interfaces	98
5.3.3	Intermediate Representation Layers	99
5.4	LogicGlue Driver Specification	100
5.4.1	Function Definitions	100
5.4.2	Numeric Instructions	102
5.4.3	List Instructions	104
5.4.4	Branching Instructions	106
5.4.5	Advanced Instructions	107
5.5	LogicGlue Interpreter	109
5.5.1	LogicGlue High-Level Programming Library	109
5.5.2	Converting Data Formats	110
5.6	Supporting LogicGlue on a new Platform	112
5.7	LogicGlue Benchmark	113
5.8	Discussion and Future Work	114
5.9	Summary	117

6	On-going Research into Unified Plug-and-Play Programming	118
6.1	Introduction	119
6.2	UniGlue: Bridging Hardware and Software for Unified Prototyping . . .	121
6.2.1	Shared Resource Bus	121
6.2.2	Communication Bus	122
6.2.3	UniGlue Interface	125
6.2.4	UniGlue Extension Shield	127
6.3	Walkthrough	129
6.4	Discussion	131
6.4.1	Plug-and-Play	131
6.4.2	Moving from TYPE 2 to TYPE 3	132
6.5	Limitations and Future Work	133
6.6	Conclusion	135
7	Discussion and Future Work	136
7.1	Addressing the Research Goals	136
7.2	The Role of Ecosystems in Physical Computing	137
7.2.1	Defining and Understanding Ecosystems	137
7.2.2	Challenges and Barriers in Ecosystems	138
7.2.3	The Future of Ecosystems	139
7.3	The Importance of Future User Evaluations	140
7.4	Future Directions	141
7.4.1	Responsive Application Code	141
7.4.2	Configuration through Hardware and Software	144
7.4.3	Modular Hardware Design	144
7.4.4	Moving from Prototype to Product	145
7.4.5	Enhancing Collaboration and Community Engagement	146
7.4.6	Addressing Environmental and Sustainability Concerns	147
8	Conclusion	148
Appendices		
A	Toolkit Classification	172
A.1	Holistic Characteristics	172
B	LogicGlue Driver Specification	174
B.1	Instructions	174
B.2	Numerics Subsystem	178
B.3	Lists Subsystem	184
C	LogicGlue Interpreter	187
C.1	Platform-Specific Functions	187

LIST OF FIGURES

1.1	This figure illustrates how the research questions and goals relate to the contributions made in this dissertation.	11
2.1	Summary of the five prototyping types we defined. Referred platforms are Arduino [Arduino, 2022], Teensy [Teensy, 2021], Raspberry Pi [Pi, 2022b], micro:bit [microbit, 2022], Circuit Playground Express [Adafruit, 2024], Jacdac [Devine, 2022] and .NET Gadgeteer [Hodges, 2013].	15
2.2	Illustration of a set of discrete components, considered TYPE 1 prototyping.	16
2.3	Illustration of a set of microcontroller development boards, considered TYPE 2a prototyping.	17
2.4	Illustration of a set of breakout boards, considered TYPE 2b prototyping.	18
2.5	Illustration of an integrated development board, considered TYPE 2c prototyping. This figure shows the micro:bit [microbit, 2022] platform.	19
2.6	Illustration of an integrated modular system, considered TYPE 3 prototyping. This figure shows the Jacdac [Devine, 2022] platform.	20
2.7	Overview of all modules within the SoftMod ecosystem.	21
3.1	Categories and labels concerning the nature and application of the platforms.	29
3.2	Categories and labels concerning the assembly of individual elements into a working prototype.	29
3.3	Categories and labels relating to deploying and configuring electronic prototypes built with the platforms in our survey. Note that the labels in two categories are not mutually exclusive.	31
3.4	Categories and labels relating broadly to the availability and use of the electronic prototyping platforms in-scope for our survey.	32
3.5	A visualization of the first part of our dataset.	34
3.6	A visualization of the second part of our dataset. The four clusters, representing more than a single toolkit, have a gray background.	35

3.7	Prototyping platforms ranked according to four holistic characteristics, with ‘better’ on the right. Not all platforms can be named due to space constraints, but the shading indicates the distribution of all 56 platforms from this study across each characteristic. Note that rankings are all relative to the dataset. A larger version of this image can be found in Appendix A.1.	36
3.8	Comparing the ranking of the importance of different characteristics between electronics engineers and respondents with a different background. We determined significance using Mann-Whitney U tests (* $p < 0.05$, ** $p < 0.001$).	39
3.9	The number of copies of electronic prototypes made by electronics engineers and respondents from other disciplines.	41
4.1	CircuitGlue is a novel electronic prototyping board that allows a wide variety of off-the-shelf electronic components and modules to be connected to a software configurable header (at right). After configuration and connection, modules work instantly and are compatible with each other independent of the voltage levels, interface types, communication protocols, and pinouts they use.	47
4.2	Configuring the CircuitGlue board to drive the temperature sensor by (1) connecting the CircuitGlue board to the computer and micro:bit, (2) configuring CircuitGlue by selecting the temperature module in the configuration tool, and (3) plugging the temperature sensor module into the programmable header.	49
4.3	Web-based CircuitGlue configuration tool.	50
4.4	Visualizing the reading of the temperature sensor module in the Jacdac dashboard.	51
4.5	Writing the application logic on the micro:bit using MakeCode building blocks.	51
4.6	A circuit diagram generated to facilitate building a custom circuit using the DC motor, temperature sensor, and PIR motion sensor.	52
4.7	Comparing the ultrasonic distance sensor and PIR motion sensor side-to-side in the Jacdac dashboard. The Jacdac dashboard visualizes the distance sensor (left) using a numeric value, while the motion sensor (right) is represented by a graphical illustration indicating whether motion is detected or not.	53
4.8	Example of the conceptual renderings used in the interviews to gather early feedback on the CircuitGlue concept.	55
4.9	Example of the translation code written to support the DHT11 temperature sensor module.	58
4.10	Adding a new module to the database using the configuration tool. . . .	59

4.11	The design of the CircuitGlue board with (1) a System-on-Chip (SoC) controlling and monitoring the board, (2) voltage regulation and power delivery, and (3) hardware components for switching the assignment of programmable pins.	60
4.12	Block diagram with all regulators responsible for providing the three voltage levels used by the CircuitGlue board.	61
4.13	The required setting in the digital potentiometer based on the requested output voltage and selected top resistor.	62
4.14	Block diagram of all components required for changing the assignments of the programmable header pins of the CircuitGlue board.	62
4.15	Circuit for connecting the programmable header pin to a digital high-speed GPIO pin on the nRF52840 SoC.	63
4.16	Circuit for connecting the programmable header pin to an analog pin on the nRF52840 SoC.	64
4.17	Circuit for connecting the programmable header pin to ground.	65
4.18	Circuit for connecting the programmable header pin to the programmable voltage level.	65
4.19	Comparing two different types of accelerometers in the Jacdac dashboard by using two connected CircuitGlue boards.	69
4.20	Building a prototype using multiple CircuitGlue boards.	70
4.21	Building a prototype using a single CircuitGlue board in combination with the <i>circuit diagram generator</i> to facilitate building the breadboard circuits.	71
4.22	CircuitGlue used to extend the Jacdac ecosystem with new modules. . .	71
4.23	Results of the technical evaluation of CircuitGlue.	74
4.24	Breadboard diagram demonstrating how participants of the user evaluation should interconnect all electronic modules.	76
4.25	Overview of the updated CircuitGlue board.	78
4.26	The updated design of the modular CircuitGlue system with (1) a controller board containing a System-on-Chip (SoC) for controlling and monitoring the CircuitGlue system, (2) a power board for voltage regulation and USB-C Power Delivery, and (3) a set of logic boards responsible for switching the assignment the programmable header pins.	79
4.27	Block diagram of the power board illustrating all voltage regulators responsible for providing the voltage levels used by the CircuitGlue logic boards.	80
4.28	Block diagram of the logic board, which is responsible for changing the assignments of four programmable header pins.	81
4.29	Block diagram illustrating the circuit for selecting the output voltage in each logic board.	81
4.30	Circuit for connecting the programmable header pin to ground, power, or output a PWM signal.	82

4.31	Circuit for connecting the programmable header pin to a digital high-speed GPIO pin on the nRF52840 SoC.	82
4.32	Circuit for connecting the programmable header pin to an analog pin on the STM MCU.	83
4.33	Redesign of the CircuitGlue interface, with (a) an active DC motor, (b) a configured RGB LED module, and (c) a second DC motor being configured.	84
5.1	Overview of LogicGlue. (a) The novel driver specification of LogicGlue encodes the behavior of drivers in bytecode to ensure platform independence and compatibility across various microcontrollers and programming languages. (b) The LogicGlue interpreter is responsible for processing the bytecode driver specifications and executing platform-specific commands. (c) The LogicGlue programming library is designed to facilitate interaction with electronic components through the interpreter.	91
5.2	a) The traditional code that is required to interact with the TMP36 analog temperature sensor. b) The traditional code that is required to interact with the MCP9808 digital temperature sensor.	93
5.3	a) Preamble for including the LogicGlue driver specification for the TMP36 analog temperature sensor. b) Preamble for including the LogicGlue driver specification for the MCP9808 digital temperature sensor. c) Application logic interacting with either temperature sensor using temperatures in Fahrenheit.	94
5.4	LogicGlue's graphical interface for creating drivers using the driver specification.	95
5.5	LogicGlue driver specification for the MCP9808 temperature sensor.	96
5.6	Driver specification for the HC-SR04 ultrasonic distance sensor. a) shows the bytecode definition, b) the boot function, and c) contains all function definitions.	101
5.7	Defining a property for setting the gain of the TCS34725 color sensor.	103
5.8	Defining static properties for getting the size of the SSD1306 OLED display.	103
5.9	Illustration of the numeric subsystem within the LogicGlue driver specification, demonstrating various numeric operations.	104
5.10	Illustration of the list subsystem within the LogicGlue driver specification, demonstrating the instructions for sending the pixel buffer to the SSD1306 OLED display.	105
5.11	Example of the <i>FOREACH_BYTE</i> instruction.	106
5.12	Example of the LogicGlue driver specification, demonstrating the scope of variables.	107
5.13	Example of advanced instructions of the LogicGlue driver specification, demonstrating how an if-elif-else test can be created using <i>GOTO</i> instructions and labels.	108

5.14	Example of advanced instructions of the LogicGlue driver specification, demonstrating how a for-loop can be created using <i>GOTO</i> instructions and labels.	108
5.15	Example of the application logic for interacting with an ultrasonic distance sensor and RGB LED.	109
5.16	Example of the application logic for interacting with the SSD1306 OLED display.	111
5.17	Subset of the graph with data formats and their converters. This figure shows data formats for color representations.	112
5.18	A conceptual illustration of an extension shield equipped with a memory chip allows for the embedding of a component's driver, ensuring automatic recognition and configuration by LogicGlue upon connection.	117
6.1	Components of the UniGlue system, with a) the UniGlue controller board, b) the UniGlue logic boards, c) the splitter separating the programmable header of the logic board, d) 4-pin UniGlue extension shield, e) 8-pin UniGlue extension shield, and f) the user's microcontroller.	122
6.2	Example of the UniGlue setup, with a) the UniGlue controller board, b) the UniGlue logic board, and c) an OLED display connected to the UniGlue extension shield.	123
6.3	Block illustration of the RS485 communication bus in the UniGlue system used for back-end data exchange between all boards.	124
6.4	Timing sequence of the UniGlue control line to manage bus access. (a) Two devices simultaneously attempt to access the bus by pulling the control line low. (b) Device 1, with a shorter pulse duration, checks the control line and finds it still low, indicating that another device has taken control. (c) Device 2, having a longer pulse duration, checks the control line and finds it high, confirming that it now has control of the bus. (d) Device 2 completes data transmission and releases the control line by returning it to a high state, making the bus available for other devices.	125
6.5	The UniGlue interface for displaying and interacting with connected electronic components. The logic board on the bottom uses a splitter board to divide the programmable header into two.	126
6.6	Popup showing an interactive example for the plugged-in DHT-22 temperature and humidity sensor. In this example, users can specify the desired output format for the temperature, and copy the example code directly into their application.	127
6.7	Popup for selecting a component (a) or uploading a driver file (b).	128
6.8	Example where component (a) is detected automatically and component (b) is manually configured.	128
6.9	Image of the UniGlue interface for specifying component specifications for a new driver.	129

6.10	UniGlue extension shield with 4 pins (a) or 8 pins (b), depending on the number of pins of the electronic component.	130
6.11	The UniGlue extension shield uses offset pin holes to provide a press-fit connection for components as an alternative to soldering.	133
6.12	Example illustration of a playground for an ultrasonic distance sensor allowing users to map numeric distance readings to percentages or colors.	135
7.1	Example of a modular PCB with replaceable motor drivers. Photo by BIGTREETECH (SKR3 EZ control board).	145
A.1	Prototyping platforms ranked according to four holistic characteristics, with ‘better’ on the right. Not all platforms can be named due to space constraints, but the shading indicates the distribution of all 56 platforms from this study across each characteristic. Note that rankings are all relative to the dataset.	173
C.1	Header file detailing the platform-specific functions needed to be implemented when porting LogicGlue to a new platform.	187

LIST OF TABLES

3.1 The four more holistic (and somewhat subjective) characteristics we evaluated (left) and the set of objective characteristics upon which they are based (right). 33

5.1 Execution times for interacting with electronic components using component-specific libraries versus LogicGlue. 114

ACRONYMS

API	Application Programming Interface. (p. 7)
CPU	Central Processing Unit. (p. 55)
DAC	Digital-to-Analog Converter. (p. 85)
DFU	Device Firmware Upgrade. (p. 66, 67)
DIY	Do-It-Yourself. (p. vi, 92)
ESC	Electronic Speed Controller. (p. 73)
FPAA	Field-Programmable Analog Array. (p. 54)
FPGA	Field-Programmable Gate Array. (p. 6, 54, 60)
GPIO	General Purpose Input/Output. (p. xvi, xvii, 62, 63, 75, 76, 79, 80, 82, 100, 102, 103, 110, 112, 114)
HAT	Hardware Attached on Top. (p. 22)
HATs	Hardware Attached on Top. (p. 98)
HCI	Human-Computer Interaction. (p. i, viii, 2, 3, 26, 54)
IC	Integrated Circuit. (p. 53, 81, 85, 144)
IDE	Integrated Development Environment. (p. 97, 137)
IoT	Internet of Things. (p. 97–99, 134)
IRL	Intermediate Representation Layer. (p. 99)
MCU	Microcontroller Unit. (p. 17, 18)
PCB	Printed Circuit Board. (p. xix, 6, 15–17, 32, 40, 41, 43, 53, 65, 78, 144–146)
PCI	Peripheral Component Interconnect. (p. 122)
PD	Power Delivery. (p. 60)
PIO	Programmable Input/Output. (p. 55)
PPI	Programmable Peripheral Interconnect. (p. 55)
PSoC	programmable system-on-chip. (p. 54, 60)

PWM	Pulse-Width Modulation. (p. <i>xvi, 72–75, 80–82</i>)
RISC	Reduced Instruction Set Computing. (p. <i>100</i>)
RTOS	Real-Time Operating Systems. (p. <i>97</i>)
SMD	Surface-Mounted Device. (p. <i>40, 41</i>)
SoC	System-on-Chip. (p. <i>xvi, xvii, 60–64, 66, 71, 79, 82, 83</i>)
SWD	Serial Wire Debug. (p. <i>66</i>)
SWS	Single Wire Serial. (p. <i>57</i>)
TUI	Tangible User Interface. (p. <i>3, 4</i>)

INTRODUCTION

1.1 Introduction to Physical Computing

In the domain of Human-Computer Interaction (HCI), Physical Computing refers to the process of making ubiquitous devices and represents a transformative approach to our engagement with technology. The field of Physical Computing is dedicated to creating systems that bridge the digital and physical world, enabling interactive environments that can sense, process, and actively respond to human input and environmental conditions. By integrating software behavior with responsive hardware, physical computing allows for the creation of systems that not only compute but also have the ability to interact with the real world in meaningful ways.

Compared to general-purpose computing systems, ubiquitous computing focuses on dedicated devices designed for specific functions rather than broad, versatile computing tasks. This specialization allows these devices to be seamlessly embedded into our environments, facilitating contextually relevant and intuitive interactions, like physical inputs and gestures, rather than through traditional interfaces like keyboards and screens. For example, a smart device attached to a refrigerator can monitor its contents, providing timely reminders or suggestions based on what is inside ¹. Similarly, Amazon Dash buttons ² simplify the reordering process for frequently used products by embedding this specific functionality within the user's physical environment. This approach to computing—where devices are purpose-built and integrated into everyday settings—enhances user experience by making technology more intuitive and immersive.

Over the years, the advancement of physical computing introduced a remarkable range of prototyping tools and platforms aimed at making physical computing more accessible to a broader audience. This democratization has considerably reduced the barriers to creating interactive and responsive systems. For example, the introduction of development platforms, such as Arduino [Arduino, 2022] and micro:bit [microbit, 2022], have made it easier for hobbyists, educators, and innovators to design and implement

¹ <https://www.samsung.com/us/explore/family-hub-refrigerator/overview/>

² <https://developer.amazon.com/virtual-dash-button-service>

interactive systems without the need for extensive programming or electronics expertise. They have opened the door to a wide range of applications, from wearable technology that monitors and responds to bodily functions [Seneviratne, 2017] to interactive art installations that change based on audience participation [Guljajeva, 2022], and smart home devices that adjust based on environmental conditions or user preferences [Garg, 2022].

However, the increasing diversity in these tools and platforms has also introduced significant challenges. The variety in technical specifications—including processor speed, memory, and supported input/output modalities—is as wide as differences in design, features, and target audiences. This diversity significantly impacts the user experience during prototyping, with each system offering different strengths and weaknesses depending on its intended application. For novices and experts alike, navigating this landscape of prototyping tools and platforms is challenging, as documentation is often sparse or hard to comprehend and thus requires sufficient technical knowledge. Furthermore, users must consider compatibility issues between hardware and software components, which may involve varying communication protocols, pin configurations, and power requirements. These challenges are particularly daunting for newcomers to the field.

1.2 Interaction Modalities in Physical Computing

Tangible User Interfaces (TUIs) represent a significant advancement in bridging the digital and physical worlds, especially in the context of HCI. TUIs allow users to interact with digital systems through physical objects, making abstract digital concepts more concrete and accessible. This modality aligns with the goals of physical computing, which seeks to create interactive systems that respond to real-world inputs. TUIs provide a tangible form for building prototypes, enabling users to manipulate objects directly, thereby offering a more hands-on approach than purely software-based tools.

However, despite their benefits, TUIs present several challenges in the context of physical prototyping. One of the main challenges is the inherent limitation in what can be built using TUIs. While they are excellent for demonstrating concepts and providing interactive feedback, the scope of what can be prototyped is often constrained by the predefined functionalities of the tangible objects themselves. Unlike software-based tools that can be easily updated or reprogrammed to handle new tasks, TUIs are often bound by their physical form, which can limit their flexibility and adaptability in rapidly evolving prototyping scenarios.

Another significant challenge with TUIs is the difficulty of integrating various tangible interfaces into a cohesive system. As with many electronics prototyping tools, TUIs often operate within their ecosystems, designed to work seamlessly with specific hardware and software components. Integrating TUIs from different systems or combining them

with other prototyping tools can be complex and requires a deep understanding of both the hardware and software involved. This fragmentation creates barriers for users, especially those who are not experts in electronics or programming, making it difficult to achieve a unified prototyping workflow that leverages multiple TUIs.

Furthermore, the integration of TUIs with other interaction modalities, such as sensor-driven environments or gesture recognition systems, can introduce additional technical complexities. These systems often require custom solutions for communication protocols, power management, and data synchronization, adding layers of difficulty to the prototyping process. As previously discussed, these challenges are exaggerated by the diversity of available tools and platforms, each with its specifications and compatibility issues, further complicating the integration of TUIs into a seamless prototyping ecosystem.

1.3 Exploring the Diversity of Physical Computing Platforms

The domain of physical computing has rapidly evolved, democratizing the process of technology creation and experimentation. This evolution has welcomed a diverse group of enthusiasts, ranging from hobbyists and educators to professional engineers, each bringing their unique perspectives and skills to the world of electronics prototyping. At the core of this transformation is an expansive range of hardware and software tools, each with distinct features, capabilities, and intended applications. This wide spectrum of prototyping resources is designed with a fundamental goal to cater to the varied needs of users with different levels of expertise and project ambitions.

This section underscores two fundamental challenges in physical prototyping: hardware assembly and software development. By highlighting the distinctions between hardware and software diversities within physical computing, this section explores how these variations empower creators and introduce prototyping difficulties.

1.3.1 Assembling Hardware: A Tangible Foundation

Physical computing involves the process of selecting and connecting various electronic components to build a functioning system. This process is key to creating devices that interact with the physical world. Physical prototyping involves a combination of microcontrollers, sensors, actuators, and other electronic components that work together to execute specific tasks or actions based on programming logic. With the increasing popularity of physical computing, hardware assembly has become more and more accessible to a wider audience. While traditional tools, like breadboards and jumper wires, require users to connect components manually, they require a solid understanding of electronics. This manual process can be daunting for beginners who must navigate the complexities of circuit design and component functionality [Mellis, 2016].

Breakout boards offer a solution to these challenges. They are designed to make small

or complex components easier to use by embedding all required components and extending their connections to a larger, more manageable board. This simplification allows users to focus on the essentials of their project without getting bogged down by intricate wiring details. Examples include the Adafruit BNO055 [Adafruit, 2022] and ESP8266 [Espressif, 2022a] breakout boards, incorporating an accelerometer and WiFi chip. However, breakout boards often come from different manufacturers with varying specifications—such as voltage and interface types—posing compatibility issues. Additional converter boards like the SparkFun Buck-Boost Converter³ and Adafruit FT232H breakout board⁴ help overcome these obstacles, yet selecting and properly integrating these components still requires electronics knowledge.

Further democratizing prototyping, integrated modular platforms like .NET Gadgeteer [Hodges, 2013], littleBits [Bdeir, 2009], and Lego Mindstorms [Lego, 2022] have been developed. These systems consist of a set of modules specifically designed to work together and allow users to easily connect modules without delving into technical datasheets or sourcing additional parts. While these platforms simplify electronics for newcomers, they may limit the customization and flexibility sought by experienced users or specific projects, highlighting a balance between ease of use and project customization.

1.3.2 Developing Software: Bringing Hardware to Life

The software development aspect of embedded systems is fundamental for assigning behavior to hardware assemblies. Typically, software development involves integrating several external elements like low-level drivers and high-level libraries. Low-level drivers are key for the hardware-software interaction and handle the communication between hardware components at a binary and electrical level. This includes managing hardware-specific registers and communication protocols. For instance, the driver for the SSD1306 OLED display controls the hardware registers over the I2C protocol to update pixels. These drivers ensure that users are not confronted with hardware-specific registers that often have to be assigned in a specific order and the detailed timings and intricacies of communication protocols. High-level libraries offer additional layers of abstraction on top of the driver. By offering simpler interfaces, the complexity of hardware interaction is significantly reduced, making it more accessible and reducing the learning curve. For example, to display text on the SSD1306 display, developers can utilize the features that convert each character into the pixel representation required by the display.

While drivers and libraries significantly lower the threshold for programming electronic systems, they are often entangled and available as a single software solution for interacting with a component or set of components. For instance, software solutions,

³ <https://www.sparkfun.com/products/15208>

⁴ <https://www.adafruit.com/product/2264>

such as the DHT sensor library⁵ developed for the DHT11 temperature sensor⁶, are not compatible with the DS18B20 temperature sensor⁷ as these components use different data reading methods and communication protocols despite offering similar functionalities. Similarly, the Adafruit GFX graphics library⁸ is primarily designed for displays using SPI. Driving I2C-based displays, such as the SSD1306, presents significant challenges. Moreover, many libraries are designed with a specific microcontroller or platform in mind, like Arduino [Arduino, 2022], making them incompatible with other platforms such as micro:bit [microbit, 2022].

The tight coupling of drivers and libraries presents significant challenges for developers who must find the most suitable software solution from a vast set of similar yet often incompatible sets of available solutions. Equally important is the burden that comes with maintaining these software solutions. Engineers need to ensure that new hardware components and microcontrollers are compatible with a wide variety of existing software solutions to guarantee compatibility with existing offerings.

Jacdac [Devine, 2022] partially addresses this problem, as it offers a communication protocol that enables developers to interface with electronic components via services. These services effectively split application logic from hardware but introduce additional translation steps to map functions of hardware components onto corresponding generic services and translate communication messages to the Jacdac protocol. This translation often results in the loss of features unique to individual components, which aren't captured by the generic services and may also introduce latencies.

1.3.3 Bridging Hardware and Software: The Role of Hardware and Software Glue

Navigating the complexities of physical computing, the concepts of *hardware glue* and *software glue* play pivotal roles in bridging gaps between components and functionalities, enabling seamless integration and interaction within projects. These concepts, while distinct, work in tandem to facilitate the creation of complex systems from disparate parts. Here, we introduce and explore the intricacies of hardware and software glue, shedding light on their significance in electronics prototyping.

Hardware glue refers to the physical components and circuitry used to connect various electronic modules and peripherals in a system. This includes everything from simple wires and solder bridges to voltage and protocol converters and more complex programmable devices like Field-Programmable Gate Arrays (FPGAs) or custom Printed Circuit Boards (PCBs) designed to interface different modules. The primary function of hardware glue is to ensure that components, which may not have been originally designed to work together, can communicate and function as part of a cohesive system.

⁵ <https://www.arduino.cc/reference/en/libraries/dht-sensor-library/>

⁶ <https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf>

⁷ <https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf>

⁸ <https://learn.adafruit.com/adafruit-gfx-graphics-library/overview>

It addresses challenges such as incompatible voltage levels, differing communication protocols, and physical connection mismatches.

Software glue, on the other hand, comprises the digital layers—drivers, Application Programming Interfaces (APIs), middleware, and custom scripts—that facilitate effective communication and data exchange between disparate software entities or between hardware components and the software controlling them. It is the invisible thread that weaves through the elements of a physical computing system, enabling components to communicate with each other and the user’s application. Software glue abstracts the complexities of direct hardware manipulation, providing developers with a more intuitive means of defining behavior and interactions within their systems.

The integration of hardware and software glue is crucial for lowering the barriers to developing electronics projects and products. Together, they enable developers to create systems that are greater than the sum of their parts, combining off-the-shelf components and custom solutions into fully functional prototypes or end-user products. In essence, hardware and software glue collectively form the backbone of modern prototyping and embedded systems design, allowing for rapid development, testing, and deployment of innovative electronic solutions. They address the inherent complexities of working with a diverse range of components and technologies, making technology creation more accessible and versatile for developers of all skill levels.

While hardware and software glue are essential tools in physical computing, their use introduces a potential efficiency trade-off. These trade-offs stem from the balance between ease of integration and system efficiency. On one hand, hardware and software glues abstract away the complexity of interfacing diverse components and technologies, significantly reducing development time and making prototyping more convenient. This abstraction enables developers to focus on design and functionality rather than the intricacies of hardware-software communication.

However, the abstraction layers introduced by hardware and software glue can lead to increased computational overhead. For example, software glue that abstracts driver functionality may need additional processing to translate high-level commands into low-level hardware actions, consuming more CPU cycles and potentially increasing response times. Similarly, hardware glue designed to standardize component connectivity might limit the direct control over hardware specifics, possibly affecting the precision and efficiency of the system’s operation.

In essence, while software and hardware democratize electronics prototyping by lowering technical barriers, they may also impose constraints on system performance and efficiency. Developers must, therefore, weigh the benefits of rapid development and ease of use against the potential for decreased system optimization and increased resource consumption.

1.4 Research Questions

As the landscape of electronics prototyping evolves, it is crucial to understand the unique challenges faced by users and how these challenges impact the usefulness and accessibility of prototyping tools. This dissertation aims to explore several key aspects of electronics prototyping to enhance our understanding and development of more inclusive and versatile prototyping solutions. The research questions guiding this dissertation are as follows:

Q1 What are the primary challenges faced by users of varying expertise levels when selecting and integrating hardware and software components for physical computing projects?

This question aims to uncover the specific difficulties encountered by different user groups, ranging from beginners to advanced users, in the electronics prototyping process. Understanding these challenges is crucial for developing tools and methodologies that are accessible and useful for a broad audience.

Q2 How do the concepts of hardware glue and software glue contribute to lowering the barriers to physical computing, and what are the trade-offs in terms of system performance and efficiency?

By exploring the roles of hardware and software glue—components that facilitate the integration of disparate systems—this question seeks to evaluate their effectiveness in simplifying the prototyping process. It also examines the potential downsides, such as decreased system performance or increased complexity, associated with their use.

Q3 What strategies can be developed to address compatibility issues between hardware and software components in physical computing, particularly concerning communication protocols, pin configurations, and software drivers?

This question focuses on finding solutions to the fragmentation and compatibility issues that often arise when integrating various components and tools in a prototyping environment. The aim is to identify or develop strategies that can streamline the integration process, making it more straightforward for users to combine multiple elements within their projects.

1.5 Research Goals

This dissertation aims to enhance the electronics prototyping workflow by looking into the concepts of hardware and software glue, making it more accessible for a wide range of users with different experience levels, both in hardware design and software development. This work is driven by several specific research goals, each contributing to the broader vision of democratizing technology creation through improved tools, frameworks, and knowledge.

Building on the research questions, the goals of this dissertation are to address the identified challenges in electronics prototyping and propose solutions that study and enhance the ease-of-use, efficiency, and versatility of prototyping tools and methodologies. The key research goals are as follows:

G1 Mapping and Understanding the Electronics Prototyping Domain

This goal involves an extensive analysis of the current spectrum of electronics prototyping toolkits, documenting their functionalities, advantages, and limitations. The aim is to establish a comprehensive framework that systematically categorizes these tools, making it straightforward for users of varying expertise to navigate through the applications and focus areas of existing toolkits, thereby streamlining the prototyping process to better meet a wide range of user needs. Furthermore, this goal is targeted at uncovering the diverse experiences and preferences among prototyping tool users by exploring the various challenges encountered, tools preferred, and strategies employed by users to materialize their ideas. Insights gained here will inform the creation of tailored solutions designed to address the wide range of user needs and to streamline the prototyping process. This goal directly addresses research question **Q1**.

G2 Bridging Hardware Compatibility and Integration

Aligning with questions **Q2** and **Q3**, this goal is dedicated to developing solutions that address the compatibility challenges and technical complexities involved in integrating various hardware components within electronics prototyping workflows. By introducing innovative approaches to hardware integration, this research seeks to simplify the process of combining disparate hardware elements, enhancing the overall prototyping efficiency.

G3 Bridging Software Interactions for Prototyping

Complementing the hardware-focused goal in questions **Q2** and **Q3**, this objective concentrates on creating and implementing solutions that simplify software interactions within the prototyping process. It tackles the hurdles associated with software compatibility, aiming to develop integrated software solutions that facilitate seamless communication and interaction between various applications and hardware devices.

1.6 Contributions

This dissertation presents advancements in electronics prototyping, making electronics prototyping more accessible for novice users. It delves into the vast range of existing prototyping tools, identifies crucial gaps, and introduces solutions to overcome these challenges.

The key contributions are outlined as follows. Figure 1.1 summarizes how the contributions link to the different research questions and goals.

1. **Development of a Comprehensive Taxonomy for Electronics Prototyping Toolkits [Lambrichts, 2021]:** We provide a structured categorization of electronics prototyping toolkits, distinguishing them by design, features, and intended user demographics. The taxonomy facilitates a clearer understanding of the prototyping landscape, guiding users and developers in selecting the most appropriate tools for their specific needs. This contribution addresses goal **G1**.
2. **Analysis of User Priorities and Prototyping Strategies [Lambrichts, 2021]:** Through a detailed analysis, this work uncovers the preferences and approaches users of different backgrounds have towards electronics prototyping toolkits. This analysis not only highlights the diverse needs and challenges users face but also provides a foundation for developing more user-centered prototyping tools and resources. This contribution addresses goal **G1**.
3. **Hardware Compatibility through CircuitGlue [Lambrichts, 2023]:** CircuitGlue is an electronic converter board that facilitates how heterogeneous components are interconnected. By enabling software programmable pin assignments, protocol translations, and voltage conversions, CircuitGlue helps creators integrate diverse off-the-shelf components into their projects, thereby simplifying the hardware assembly process. This capability lowers the barrier to entry for those new to electronics, as it reduces the complexity traditionally associated with hardware prototyping. This contribution addresses goal **G2**.
4. **Software Compatibility through LogicGlue [manuscript submitted]:** LogicGlue addresses the complexities of driver compatibility and programming in embedded systems. It introduces a framework for creating platform-independent drivers, removing the dependency on specific microcontrollers or programming languages. LogicGlue streamlines the development process, ensuring seamless interfacing with electronic components and preserving the full range of their functionalities. This contribution addresses goal **G3**.

	G1. Mapping and Understanding	G2. Bridging Hardware Compatibility	G3. Bridging Software Interactions
Q1. What challenges do different users face in integrating hardware and software?	Chapter 2 + 3: Taxonomy and Survey		
Q2. How do hardware and software glue simplify integration, and what are the trade-offs?		Chapter 4: CircuitGlue	Chapter 5: LogicGlue
Q3. What strategies solve compatibility issues in physical computing?		Chapter 4: CircuitGlue	Chapter 5: LogicGlue

Figure 1.1: This figure illustrates how the research questions and goals relate to the contributions made in this dissertation.

In evaluating the contributions of this dissertation, it is important to consider established frameworks that assess the quality and effectiveness of electronics toolkits. Myers’ concepts [Myers, 2000] of “high ceiling,” “low threshold,” “large walls,” and “path of least resistance” provide a valuable lens for evaluating how toolkits can cater to a wide range of users and project complexities. A high ceiling allows advanced users to create complex projects, while a low threshold ensures that beginners can easily start using the toolkit. Large walls represent the toolkit’s capacity to support a broad spectrum of features, and the path of least resistance highlights intuitive design that helps users achieve their goals efficiently.

Additionally, Olsen’s framework [Olsen, 2007] emphasizes expressiveness, ease of learning, and support for creativity as critical aspects for evaluating toolkits. These criteria focus on the toolkit’s ability to support innovative and diverse projects, facilitate user learning, and encourage creative exploration.

1.7 Dissertation Outline

This dissertation is structured into a series of chapters dedicated to exploring different aspects of electronics prototyping. This structure starts from evaluating current prototyping tools and methodologies to introducing novel solutions to addressing the identified gaps. Here is a detailed overview of the focus and contributions of each chapter:

CHAPTER 2: Established Approaches to Electronics Prototyping

This chapter explores established approaches to electronics prototyping, examining their benefits, limitations, and the potential for hybrid solutions that leverage the strengths of each type. By understanding the spectrum of prototyping methodologies, users can better navigate the choices available to them, fostering innovation and creativity in their projects.

CHAPTER 3: Finding Common Ground

This chapter delves into the vast range of toolkits available for prototyping interactive and ubiquitous electronic devices. Despite the easy availability of their technical specifications, these toolkits exhibit a wide variety of designs, features, and user focus, each with unique strengths and limitations. Through a newly developed taxonomy, this chapter thoroughly analyzes these systems beyond their technical aspects, incorporating findings from an online survey to shed light on user preferences and prototyping approaches. In addition, this chapter details the results of an online survey evaluating the taxonomy's real-world relevance. This provides insights into the electronics prototyping practices across a diverse audience, including hobbyists, educators, and professionals. This survey seeks to understand user preferences, tool usage patterns, and prototyping challenges, thereby validating the taxonomy and identifying prevalent trends and gaps in electronics prototyping.

CHAPTER 4: Plug-and-Play Hardware Through CircuitGlue

Responding to gaps identified in the previous chapter, CircuitGlue is introduced as an electronic converter board designed to simplify the interconnection of diverse electronic components. It features an eight-pin software programmable header, allowing for flexible configurations in software to support a variety of connections. This chapter showcases CircuitGlue's role in streamlining hardware assembly and explores new prototyping workflows it unlocks. A preliminary user study further examines its usability, particularly for individuals new to electronics, offering valuable insights into its practical applications and user experience.

CHAPTER 5: Plug-and-Play Software Drivers Through LogicGlue

LogicGlue emerges as a solution to the challenges of compatibility and programming intricacy in embedded system development. It introduces a platform-independent driver specification format, enabling the creation of versatile drivers that are not tied to any specific microcontroller or programming language. Supported by a visual programming interface and a comprehensive interpreter, LogicGlue facilitates straightforward interfacing with electronic components, ensuring the preservation of hardware functionality without the trade-offs typically associated with existing solutions. This chapter evaluates how LogicGlue facilitates software development for individuals with various backgrounds by ensuring software adaptability across different components and platforms.

CHAPTER 6: On-going Research into Unified Plug-and-Play Programming

Building on the foundations laid by CircuitGlue and LogicGlue, this chapter investigates their combined potential to create a plug-and-play ecosystem for electronic prototyping. It highlights how their integration enables developers to seamlessly incorporate various electronic components into their projects, removing barriers caused by compatibility and programming complexities. This unified approach further democratizes electronics prototyping, allowing users of all backgrounds to participate.

CHAPTER 7: Discussion and Future Work

An overarching discussion ties together the findings and implications from each chapter, examining the collective impact of our classification, CircuitGlue, and LogicGlue, within the broader context of electronic prototyping toolkits. This chapter reflects on the broader implications of this work for the field of embedded systems, suggesting pathways for future exploration and development in making prototyping more accessible.

CHAPTER 8: Conclusion

The final chapter synthesizes the key insights and contributions of the dissertation, highlighting the solutions presented to simplify the electronics prototyping process.

ESTABLISHED APPROACHES TO ELECTRONICS PROTOTYPING

Motivation

To lay the groundwork for this dissertation, this chapter presents a literature review to analyze general programming styles used in electronics prototyping. This chapter discusses the key findings of this literature review and introduces five distinct prototyping types that are determined by the tools and platforms used during prototyping (**G1**). These findings serve as the foundation for a comprehensive prototyping framework.

2.1 Introduction

Since the introduction of physical prototyping, engineers have sought ways to accelerate the design, testing, and assembly of interactive electronic devices. As described in the previous chapter, these tools and platforms have subsequently transitioned to a broader audience of individuals, trying to democratize the field of physical computing. Blikstein [Blikstein, 2015] offers a historical analysis of 30 years of developments in electronics prototyping toolkits and identifies three levels of abstraction. This chapter builds on Blikstein’s categorization to establish a revised and expanded framework in response to the ever-changing landscape of physical computing. To avoid confusion, we refer to ‘Types’ of electronics prototyping rather than Blikstein’s ‘Levels’.

This chapter is based on the conference proceedings paper “A Survey and Taxonomy of Electronics Toolkits for Interactive and Ubiquitous Device Prototyping”, which was published in the “Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies” [Lambrichts, 2021]. The paper was presented virtually at the IMWUT conference in 2021 and received a Best Presentation Award.

2.2 Types of Prototyping

In the following sections, we introduce five distinct prototyping types that are based on the unique characteristics of electronics prototyping toolkits. Figure 2.1 provides a summary of these types.

	TYPE 1 Discrete Components	TYPE 2a Integrated Microcontroller Boards	TYPE 2b Breakout Boards and Modules	TYPE 2c Integrated Development Boards	TYPE 3 Integrated Modular Systems
Electronics Components	Discrete components: resistors, capacitors, transistors, ...	Microcontroller development boards: Arduino, Teensy, Raspberry Pi, ...	Specialized modules: sensors, actuators, WiFi modules, ...	Boards with built-in sensors or actuators: micro:bit, Circuit Playground Express, ...	Complete sets designed to work together: Jacdac, .NET Gadgeteer, ...
Flexibility	High Full control over design and components	Moderate Functionality expandable with additional hardware components	Moderate Specialized functionalities, requires integration with other types	Low Built-in functionalities are very limited, but expandable with additional components	Very Low Limited to availability of ecosystem functionalities
Ease of Use	Low Requires in-depth electronics knowledge	Moderate Simplifies microcontroller use but needs additional components for full functionality	Moderate Simplifies hardware integration but often requires solving compatibility challenges	High Ready to interact with the physical world out of the box	Very High Fool-proof assembly and operation
Prototyping Speed	Slow Manual assembly and testing, error-prone	Moderate Requires interconnecting external components	Moderate Integration often requires adding conversion components	Very Fast Immediate start with no initial assembly	Fast Expedited prototyping by plug-and-play experience
Knowledge Required	High Significant electronics knowledge needed	Moderate Basic microcontroller and electronics knowledge required	Moderate Requires understanding of specific components and protocols	Low Minimal electronics knowledge needed, but moderate when connecting additional components	Very Low Designed for beginners with no electronics background
Targeted User Groups	Advanced users	Hobbyists, educators, and users needing convenient prototyping solutions	Hobbyists, educators, and users needing functionalities , users expanding Type 2a or 2c	Beginners and educators looking for efficient entry	Beginners, educators, and users prioritizing ease of use over customization

Figure 2.1: Summary of the five prototyping types we defined. Referred platforms are Arduino [Arduino, 2022], Teensy [Teensy, 2021], Raspberry Pi [Pi, 2022b], micro:bit [microbit, 2022], Circuit Playground Express [Adafruit, 2024], Jacdac [Devine, 2022] and .NET Gadgeteer [Hodges, 2013].

TYPE 1: Prototyping with Discrete Components

Blikstein’s ‘Level 1’ electronics is based on discrete components—the same components that are used in the mass production of electronic devices. In this thesis, we refer to this approach to prototyping as **TYPE 1**. The key to using production-ready electronic components for prototyping is making the desired electrical connections without needing to solder them to a custom-designed Printed Circuit Board (PCB). A common approach

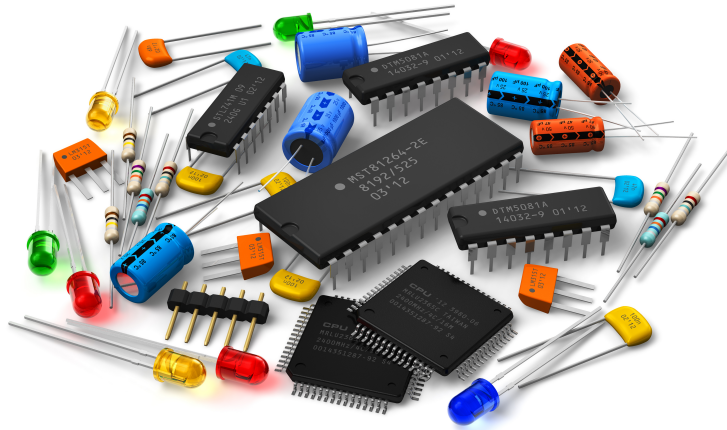


Figure 2.2: Illustration of a set of discrete components, considered *TYPE 1* prototyping.

to achieve this is to push the leads of components into a solderless breadboard.

Mellis et al. [Mellis, 2013] advocate for *TYPE 1* prototyping since it offers form-factor freedom and is closest to current engineering practices. With a solderless breadboard, it's also quick and easy to make changes as a design iterates. However, although it is relatively easy to physically plug and unplug components, making sure that the connections are exactly as planned can be fiddly and error-prone, and the resulting “rat’s nest” of circuitry is often susceptible to damage.

In an effort to facilitate *TYPE 1* electronics prototyping, product designers and researchers have built various tools. These either provide novel ways for connecting discrete components together, for example, Snap Circuits [Circuits, 2021], or they help novice engineers detect and diagnose problems [Drew, 2016]. In recent years, there is also an increased interest in using novel fabrication technologies to interconnect discrete electronic components. Researchers have explored novel approaches ranging from conductive pens [Hodges, 2014] and inkjet printing conductive traces on paper [Kawahara, 2013] to techniques for making soft [Hamdan, 2018] and stretchable [Nagels, 2018] circuits.

Despite the innovations listed above, two significant drawbacks remain when using *TYPE 1* prototyping. Firstly, if multiple copies of a circuit are needed, the entire artifact must be built from scratch each time, which can be time-consuming. Secondly, and often more significantly, prototyping with discrete components requires significant knowledge of electronics, slowing down the process of creating and iterating a prototype.

TYPE 2: Embedded Hardware Boards

Moving beyond the use of individual components for prototyping, many prototyping practices involve leveraging pre-assembled Printed Circuit Boards (PCBs), including specialized modules and development boards. Blikstein [Blikstein, 2015] categorizes this

approach as ‘Level 2’ prototyping, characterized by the use of integrated microcontroller development boards. These are PCBs centered around a specific Microcontroller Unit (MCU), equipped with additional components that facilitate operation and provide easy access to essential functionalities. In this thesis, we delve deeper into the nuances of **TYPE 2** prototyping, distinguishing it into three distinct subcategories to provide a more detailed exploration of the landscape.

TYPE 2a: Integrated Microcontroller Development Boards

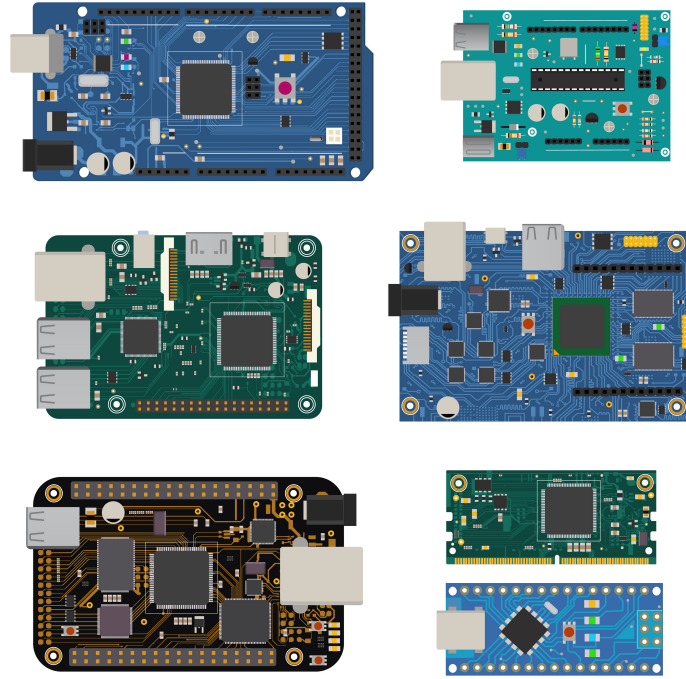


Figure 2.3: Illustration of a set of microcontroller development boards, considered **TYPE 2a** prototyping.

The first subdivision of **TYPE 2** prototyping, labeled as **TYPE 2a**, focuses on the utilization of microcontroller development boards (Figure 2.3). Example **TYPE 2a** products include the Arduino Uno [Arduino, 2022; Banzi, 2008], Raspberry Pi [Pi, 2022b], the STM32 discovery [STMicroelectronics, 2021], Teensy [Teensy, 2021], and the TI Launchpad [Launchpad, 2021]. These integrate components for power delivery, programming, communications, and basic user interaction via regulators, pin headers, USB ports, push buttons, and LEDs. **TYPE 2a** also encompasses more featured microcontroller development boards such as the Raspberry Pi [Pi, 2022b] and BeagleBone [BeagleBone, 2021]. These devices embed networking, mass storage, and additional hardware features like USB host and HDMI support and are often referred to as single-board computers as they function as stand-alone devices.

TYPE 2a prototyping is well-established in the electronics profession because, compared to the **TYPE 1** approach, it simplifies experimentation with microcontrollers. However,

for many prototype electronic devices, the microcontroller is only half of the story: any given design typically requires integration with components that are not present on the ready-made microcontroller development board. This shortcoming can, of course, be addressed by combining **TYPE 2a** with **TYPE 1**—wiring the microcontroller development board to a solderless breadboard with the necessary additional components.

TYPE 2b: Breakout Boards and Wireless Modules

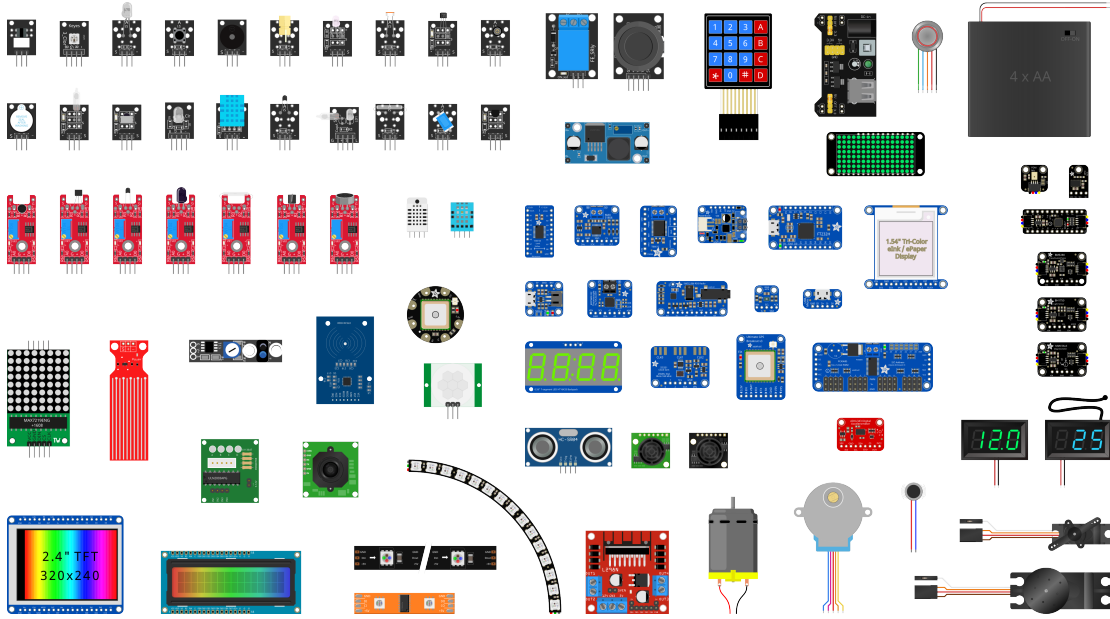


Figure 2.4: Illustration of a set of breakout boards, considered **TYPE 2b** prototyping.

Another approach that has become popular in the past decade or two is the use of breakout boards and wireless modules. These are similar to microcontroller development boards but are typically smaller, and rather than featuring an MCU, they contain a particular chip such as an accelerometer, regulator, or WiFi radio along with enough components to support its operation. Blikstein [Blikstein, 2015] does not label these explicitly, but we believe they form an interesting category of their own, which we refer to as **TYPE 2b** electronics prototyping tools (Figure 2.4).

Specific examples of **TYPE 2b** products include Adafruit’s Bosch BNO055 sensor breakout board [Adafruit, 2022], which embeds sensor fusion and an I2C interface, and ESP8266/ESP32 WiFi modules [Espressif, 2021]. Several groups of **TYPE 2b** modules are worth a particular mention: Seeed’s Grove system [Grove, 2021], MikroElektronika’s Click boards [Mikroe, 2021]; Digilent’s Pmod modules [Pmod, 2021]; SparkFun’s Qwiic modules [SparkFun, 2021a] and Adafruit’s STEMMA and STEMMA QT [Adafruit, 2021d]. These are all ecosystems of modules that are somewhat compatible with each other, making it relatively easy to interface several of them to a given microcontroller or to move between modules with different functionalities.

As with **TYPE 2a** components, a critical aspect of **TYPE 2b** components is that they are

not sufficient to build a prototype; you need to combine them with **TYPE 2a** and/or **TYPE 1** electronics. Although both **TYPE 2a** and **TYPE 2b** products may support common protocols like I2C, SPI, and UART, they are typically manufactured by different companies, and their operating voltage, physical connections, and/or communications speeds and protocols are not necessarily compatible. Therefore, selecting and interfacing components appropriately still requires a good understanding of electronics.

In summary, the main advantage of **TYPE 2** electronics prototyping over **TYPE 1** is speed and robustness—because fewer connections have to be specified and hand-made. The biggest disadvantage is reduced flexibility.

TYPE 2c: Integrated Development Boards

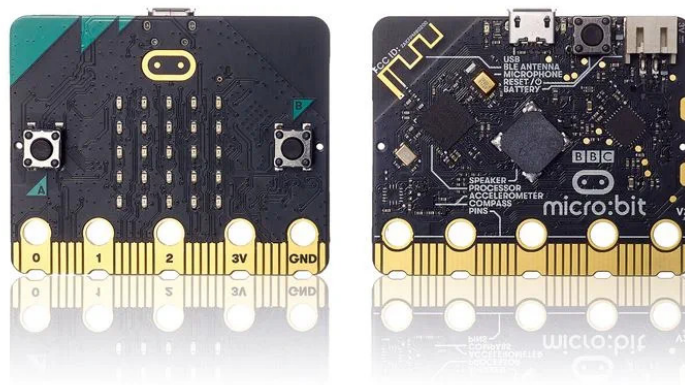


Figure 2.5: Illustration of an integrated development board, considered **TYPE 2c** prototyping. This figure shows the micro:bit [microbit, 2022] platform.

TYPE 2c platforms, as illustrated by devices like the micro:bit [microbit, 2022; Austin, 2020] and Circuit Playground Express [Adafruit, 2024], stand out for their all-in-one design that incorporates sensors and actuators directly onto the development board (Figure 2.5). This approach provides users with a device that is ready to interact with the physical world right out of the box, eliminating the need for initial hardware assembly. These boards serve as an accessible and straightforward gateway into the world of electronics prototyping, making them especially suitable for beginners and educational purposes.

However, while **TYPE 2c** boards facilitate a rapid start in prototyping with their built-

in features, they are typically very limited in the range of functionalities they offer. When projects demand more specialized capabilities not provided by the onboard features, these boards encounter limitations similar to those faced by **TYPE 2a** systems in integrating additional functionalities.

Overall, **TYPE 2c** boards strike a balance between ease of use and the potential for project expansion. They offer a practical solution for individuals and educators looking for an efficient entry point into electronics prototyping, with the flexibility to scale projects as needed.

TYPE 3: Integrated Modular Systems

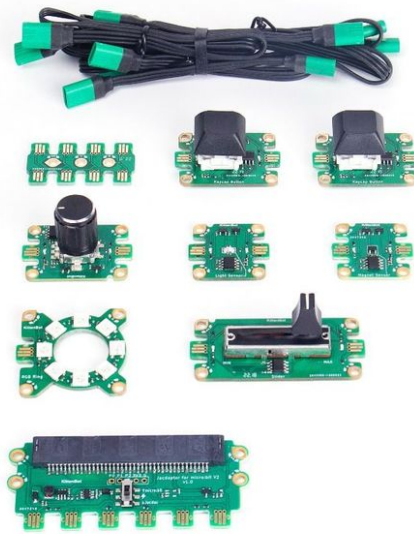


Figure 2.6: Illustration of an integrated modular system, considered **TYPE 3** prototyping. This figure shows the Jacdac [Devine, 2022] platform.

Consistent with Blikstein’s ‘Level 3’, our definition of **TYPE 3** electronics prototyping covers toolkits that consist of a complete set of modules specifically designed to work together without needing any other components (Figure 2.6). They often comprise a processing module combined with various input and output (I/O) modules. Many **TYPE 3** platforms offer keyed connectors with poka-yoke constraints or use modules that communicate wirelessly to make the assembly fool-proof. Popular examples include .NET Gadgeteer [Hodges, 2013], Phidgets [Greenberg, 2001], littleBits [Bdeir, 2009], Lego Mindstorms [Lego, 2022] and SAM Labs [Labs, 2021].

TYPE 3 platforms simplify and expedite electronics prototyping compared to **TYPE 2** because there is no need to find third-party compatible components and work out how to connect them. Blikstein [Blikstein, 2015] noticed that **TYPE 2** electronics expose more details of embedded electronic components compared to **TYPE 3**. For these reasons, **TYPE 3** platforms are frequently used in educational and leisure settings as they allow users to get up to speed and experience success without knowing many details of the

underlying electronics. However, this also means an important factor when selecting a **TYPE 3** platform is the particular set of modules it supports; adding discrete components or building custom modules is often hard.

Unlike **TYPE 2c** platforms, which integrate sensors and actuators directly onto the board for immediate use, **TYPE 3**'s approach involves selecting and assembling distinct modules to create a complete prototype. Although this introduces an extra step in the assembly process, **TYPE 3** platforms provide a versatile and modular way to prototype, allowing for the addition of functionalities as needed. This modular approach offers a balance between ease of use and the flexibility to tailor the prototype to specific project requirements.

A notable example of a **TYPE 3** platform is SoftMod, which is the result of my master thesis and was introduced in previous research [Lambrichts, 2020]. SoftMod is a flexible and modular electronics kit characterized by its soft, magnetic modules (Figure 2.7). By tracking the topology of connected modules, SoftMod offers simple, immediate plug-and-play functionality alongside more complex, user-defined behavior. SoftMod supports constructing two-dimensional and three-dimensional electronic structures, making it an inviting platform for exploration. The inherent plug-and-play nature fosters experimentation, while its support for advanced behavior customization and the ability to create a variety of soft and flexible shapes offers a ground for innovative interface designs, including wearables and interactive fabrics.

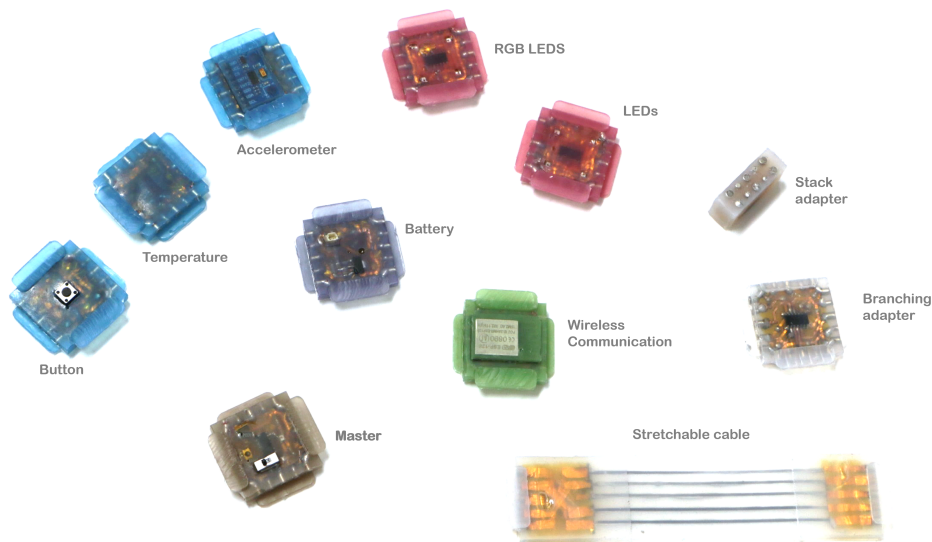


Figure 2.7: Overview of all modules within the SoftMod ecosystem.

2.3 Bridging Between Types

The progression from **TYPE 1** to **TYPE 3** prototyping represents a line from highly flexible and customizable to more user-friendly but less adaptable systems. **TYPE 1** prototyping

stands out for its use of discrete components, offering the most flexibility for projects with unique or complex requirements. However, this method's reliance on significant electronics knowledge and circuit assembly can be daunting for beginners, presenting obstacles due to its complexity and high potential for errors. Moving to **TYPE 2**, which includes microcontroller boards (**TYPE 2a/c**) and breakout boards (**TYPE 2b**), the process of integrating complex functionalities into projects becomes somewhat simplified. Despite this, challenges such as voltage mismatches and communication protocol disparities persist, highlighting ongoing compatibility difficulties. **TYPE 2c** platforms attempt to streamline the prototyping experience by integrating sensors and actuators directly onto the board, providing an immediately usable system. However, this convenience may come at the cost of reduced customization, as predefined functionalities are often limited. When the need arises to incorporate external components, **TYPE 2c** platforms revert to **TYPE 2a**'s broader but more complex integration process. Transitioning to **TYPE 3** platforms, the focus shifts towards maximizing ease of use through a modular, plug-and-play design. While this approach significantly lowers the barrier to entry, it inherently restricts flexibility and customization by confining users within closed ecosystems, challenging the integration of external or custom components.

Bridging between prototyping types **TYPE 1** and **TYPE 2** is a common practice that enables the utilization of the strengths of each approach. For instance, combining **TYPE 1**'s discrete components on a solderless breadboard with a **TYPE 2a/c** microcontroller development board allows for a flexible and powerful prototyping solution. This integration seamlessly merges the customizability and granularity of **TYPE 1** with the computational power and ease of use provided by **TYPE 2a/c** systems. Similarly, integrating **TYPE 2b** components with **TYPE 2a/c** microcontroller development boards is straightforward and very popular in the maker community, as breakout boards offer a convenient alternative to discrete components. Moreover, when **TYPE 2a** platforms like Arduino [Arduino, 2022], Raspberry Pi [Pi, 2022b], and BeagleBone [BeagleBone, 2021] are used with shields, HATs, and capes (respectively), they effectively move into **TYPE 3**. The same is true for some microcontroller development boards that incorporate sockets for .NET Gadgeteer [Hodges, 2013], or Click modules [Mikroe, 2021]. This compatibility underscores the inherent design of these platforms to be adaptable and flexible, accommodating a wide range of project requirements without the need for extensive modifications or complex interfacing solutions.

Challenges become more prominent when attempting to bridge **TYPE 2** and **TYPE 3** prototyping. The closed ecosystem characteristic of **TYPE 3** platforms, designed for simplicity and ease of use, often comes at the expense of flexibility and openness. This design philosophy can significantly restrict the ability to incorporate components from outside the ecosystem, thereby limiting the scope for customization and innovation within projects. Such constraints necessitate creative solutions and sometimes complex workarounds to integrate non-native modules or to extend the functionality beyond what is readily provided by the **TYPE 3** system.

2.4 Taking a look into the Future

The future of electronics prototyping is arriving at an exciting juncture, with potential advancements that could significantly enhance how creators approach their projects. The development of new tools and platforms explicitly aimed at facilitating seamless transitions and integrations across different prototyping styles is a promising direction. Such innovations could further democratize the field of electronics design, enabling individuals—regardless of their technical background or expertise—to harness the strengths of discrete components, integrated microcontroller boards, breakout modules, and modular systems cohesively and intuitively.

Imagine a future where creators can effortlessly mix and match components from **TYPE 1**'s vast array of discrete elements with **TYPE 2**'s microcontroller capabilities and **TYPE 3**'s modular convenience. Tools that simplify the identification of components and their specifications could automate much of the integration process, allowing for unprecedented cross-compatibility. This would expedite the prototyping phase and enhance the creative process by removing technical hurdles that may hinder innovation. Moreover, the emergence of these bridging tools could foster a collaborative environment where knowledge and resources are shared freely across communities. Educational platforms could leverage these advancements to provide more comprehensive and hands-on learning experiences, blending theory with practical application and enabling students to explore the full spectrum of electronics prototyping without being confined to a single approach.

As the boundaries between different prototyping styles become increasingly blurred, we might also witness the rise of hybrid platforms combining each type's best features. These platforms could offer the flexibility and control of working with discrete components, the rapid prototyping capabilities of microcontroller boards and breakout modules, and the user-friendly, plug-and-play experience of integrated modular systems. Such convergence could lead to a new era of prototyping where the focus shifts from managing compatibility issues to pushing the limits of what can be created.

In this evolving landscape, the role of the community—makers, educators, professionals, and hobbyists—will be essential. Feedback from these diverse groups will guide the development of tools that are not only technically robust but also aligned with the users' needs and aspirations. As we move forward, the synergy between technological innovation and community engagement will undoubtedly fuel rapid growth and transformation in electronics prototyping, opening up new avenues for exploration and discovery.

FINDING COMMON GROUND

Motivation

As the field of electronics prototyping continues to evolve, bridging the gap between the expectations of creators and the capabilities of current prototyping tools becomes increasingly important. This dissertation addresses these challenges through a set of defined research questions and goals that aim to find common ground in the prototyping landscape, understand the diverse needs of users, and motivate the development of integrated solutions that enhance prototyping workflows.

This chapter aims to explore the primary challenges faced by users during electronics prototyping (**Q1**) by conducting a comprehensive literature survey of the current set of prototyping platforms (**G1**). It introduces a taxonomy that encapsulates a broader set of characteristics essential for a holistic understanding of the prototyping process and its outcomes. Unlike existing comparisons focusing mainly on technical specifications, our approach acknowledges the importance of user-based characteristics for toolkit selection, particularly for individuals aiming to build deployable devices or prototypes that can be replicated reliably.

Despite establishing a taxonomy for electronics toolkits that categorizes available resources based on design, features, and target user groups, it became evident that further exploration was needed to gauge how these characteristics align with real-world applications and user expectations (**Q1**). To answer this question, this chapter presents the results of an online survey conducted to unravel the preferences and experiences of both non-professional makers and professional electronics engineers with prototyping electronics. The aim was to uncover any unmet needs or overlooked characteristics that could inform the development of integrated software-hardware solutions and address the challenges of ensuring broad usability and accessibility in prototyping tools.

This chapter is based on the conference proceedings paper “A Survey and Taxonomy of Electronics Toolkits for Interactive and Ubiquitous Device Prototyping”, which was published in the “Proceedings of the ACM on Interactive, Mobile, Wearable and

Ubiquitous Technologies” [Lambrichts, 2021]. The paper was presented virtually at the IMWUT conference in 2021 and received a Best Presentation Award.

3.1 Introduction

In this chapter, we present a literature survey and product summary of the broad gamut of electronics prototyping toolkits beyond the previously reported K-12 systems and develop a taxonomy that goes beyond technical features and specifications. In this sense, our work differs from existing online comparisons of commercial toolkits such as [MakeMagazine, 2021] because our broader set of characteristics provides a more holistic view of the prototyping process and the resulting artifact. For practitioners who must build a device that can be deployed reliably or want a prototype that can be replicated many times, including need-based characteristics is an important aid to toolkit selection.

Given the plethora of electronics prototyping solutions that exist, it was not feasible for us to examine them all. We have tried to provide a complete review of systems reported in the research literature, but our coverage of commercial systems is not comprehensive—for example, we have not included every microcontroller development board or every variant of Arduino [Arduino, 2022]. Instead, we have tried to include representative examples of each. In total, we have labeled 56 electronics toolkits. Our taxonomy is, however, applicable beyond the specifics of this set, and we believe it will also be valuable for discussing and comparing future generations of electronics toolkits.

We also report on the results of a formative online survey with 122 respondents, which indicates the relevance of the characteristics in our taxonomy and offers new insights into current prototyping practices, including the need to scale to multiple copies of a prototype. This data reveals some needs currently unaddressed by the established electronics toolkit offerings that could form the basis for future research.

The main contributions of this work are three-fold: Firstly, we report extensively on the literature relating to electronics prototyping toolkits, and we include representative commercial examples. Secondly, we present a novel taxonomy for classifying and comparing these toolkits in new ways that we think provide a useful perspective to others in the research community who are using electronics toolkits and/or developing new ones. We use this taxonomy to label the toolkits we have reviewed, and the resulting dataset is available online for ease of consumption. Thirdly, we report on the results of an online survey that indicates the relevance of the characteristics in our taxonomy to the practitioners we surveyed and highlights some common practices for building individual prototypes and scaling up to larger numbers. Collectively, we hope these contributions will provide a common ground for discussions between researchers in the field and will highlight opportunities for future research into toolkits that support the development of ubiquitous computing and interactive devices.

3.2 Identifying and Reviewing the Literature

Having provided a broad overview of the different approaches to electronics prototyping, in this section we describe our methodology for reviewing commercial products and academic projects in the electronics toolkit space. Data and insights from this literature review informed the taxonomy presented in the next section.

3.2.1 What is an Electronics Prototyping Toolkit?

Various research communities have contributed to electronics prototyping platforms, and these contributions have spanned many different research areas, including Human-Computer Interaction (HCI), UbiComp, electronics engineering, and mechatronics. A wealth of contributions have also come from industry, with examples of electronics toolkits from multinational corporations like Lego [Lego, 2022] and smaller startups such as SAM Labs [Labs, 2021]. With so many toolkits from a variety of stakeholders, we were keen to objectively define which prototyping toolkits should be considered in-scope or out-of-scope.

On this basis, we define electronics prototyping toolkits as having one or more higher-level electronic modules composed of discrete electronic components that empower users to prototype a wide variety of functional artifacts with control over both hardware and software. This definition is similar to the more general definition of toolkits in HCI reported in [Ledo, 2018], but we explicitly exclude the following types of projects and products:

1. The large set of discrete electronic components (TYPE 1) as they are not part of a coherent toolkit.
2. Projects that only present a novel engineering workflow without a new physical toolkit, such as the wide variety of workflows for connecting discrete components (TYPE 1) e.g. Instant Inkjet Circuits [Kawahara, 2013], iSkin [Weigel, 2015], PaperPulse [Ramakers, 2015], The ToastBoard [Drew, 2016], Scanalog [Strasnick, 2017], VirtualComponent [Kim, 2019], Makers' Marks [Savage, 2015] and Heimdall [Karchemsky, 2019].
3. Purely mechanical toolkits for making physical artifacts without embedded electronic functionality, such as basic Lego bricks, StrutModeling [Leen, 2017] and ProtoPiper [Agrawal, 2015].
4. Purely software electronic prototyping tools such as Fritzing [Knörrig, 2009; Fritzing, 2021], Autodesk EAGLE [Autodesk, 2021], ConductAR [Narumi, 2015], IFTTT [Mi, 2017] and Gumstix Geppetto [Geppetto, 2021].
5. "Assembly kits" according to Eisenberg et al.'s [Eisenberg, 2002] "specificity classification". These are interactive products, such as Sifteo cubes [Merrill, 2012], Sphero [Sphero, 2021], PlayPiper [Piper, 2021], and Cubetto [Cubetto, 2021] that require assembly or programming but only allow for making a single or a handful

of pre-determined artifacts. This naturally includes products primarily deployed as-is, such as the Niko Home Control [Niko, 2021] and Loxone [Loxone, 2021] home automation systems.

3.2.2 Corpus of Products and Publications

To create a representative corpus of products and research prototypes of electronics prototyping platforms, we conducted a systematic search on Google Search, the ACM Digital Library, and IEEE Xplore similar to the methodology of Grosse-Puppenthal et al. [Grosse-Puppenthal, 2017]. Our search terms included all possible combinations of *hardware*, *prototyping*, *construction*, *physical*, *robotic* and *electronic* with each of *platform*, *toolkit*, *prototyping* and *kit*. We then examined the referenced literature of the articles in these search results. Additionally, we browsed through popular online magazines, including Make Magazine [MakeMagazine, 2021] and retailers and manufacturers of electronic platforms, such as Seeed Studio [Studio, 2021], Sparkfun [SparkFun, 2021b] and Adafruit [Adafruit, 2021b]. For every product and article, we verified whether it was in or out-of-scope using the exclusion criteria listed above. This resulted in a representative set of 56 unique electronics toolkits.

3.2.3 Characteristics

An initial set of labels was developed by me following an open-coding process [Charmaz, 2014; Ledo, 2018], and this was complemented by deductive codes from existing literature on toolkits [Blikstein, 2015]. Collectively, we reflected on the set of labels and formulated more precise definitions. Next, we grouped the refined labels into objective characteristics that aligned with the hardware aspects of electronics toolkits that form the focus of this chapter. After reaching a consensus on characteristics and the associated labels, we did another pass over the dataset to ensure the labels were correctly applied. This procedure resulted in 13 distinct characteristics.

As our taxonomy focuses on hardware-related aspects of electronics toolkits, the final set of characteristics does not cover the programming and software debugging aspects of toolkits in detail. These largely depend on the availability, compatibility, and characteristics of software libraries and are therefore out-of-scope for this work. Likewise, our taxonomy mainly focuses on the user experience offered by electronics toolkits and is therefore complementary to existing comparisons of technical specifications such as the online overview of development boards by Make Magazine [MakeMagazine, 2021].

3.2.4 Data Points and Clusters

While labeling electronics toolkits, we identified four clusters of toolkits with very similar characteristics: (1) *generic breakout boards*, such as breakout boards for the Bosch BNO055 IMU sensor [Adafruit, 2022] and the MPR121 capacitive touch sensor [Adafruit, 2021a], (2) *programmable low-cost WiFi modules*, such as those based on the ESP32 [Espressif,

2022b] and the ESP8266 [Espressif, 2022a], (3) *silicon vendor development boards*, such as the TI Launchpad [Launchpad, 2021] and the nRF52840 DK [Semiconductor, 2021], and (4) *FPGA development boards*, such as the Arduino MKR Vidor [Arduino, 2021] and the Alchitry FPGA development boards [Alchitry, 2021]. Despite listing only two examples for each of these clusters, tens if not hundreds of similar tools exist. Although they have a wide range of technical specifications, we discovered a remarkable similarity in terms of the characteristics used for evaluation in this work. Therefore, instead of labeling each of these tools individually, we classified them into four clusters in our taxonomy. Indeed, the characteristics of each of these four clusters are very similar, and arguably, they could be further collapsed, but we didn't want to over-condense the representation of so many different toolkits. Similarly, some of the named toolkits have many variants; examples include a huge range of Arduino boards [Arduino, 2022] and many variants of Raspberry Pi, including the recently launched Pi Pico [Pi, 2022b]. Entering each of these as separate entries into our dataset was not feasible, so we have again consolidated them into a single entry.

After this clustering process and accounting for our exclusion criteria, our taxonomy includes 56 unique electronics toolkits as they exist at the time of writing. We believe that our taxonomy is more valuable than these individual data points as it presents a new perspective for discussing and comparing the current and future generations of electronics toolkits.

3.3 Electronic Prototyping Platform Taxonomy

This section presents all 13 characteristics that encapsulate the nature of each platform, the target audience, how widely adopted they are, and how prototypes are assembled, deployed, and used.

3.3.1 Nature and Application

We start by considering the **type of electronics** supported by each platform. As shown in Figure 3.1, 66% of platforms are **TYPE 3**, 8% are **TYPE 2b**, and the rest are **TYPE 2a** (using the definitions from Chapter 2). As a reminder, the large number of **TYPE 1** discrete electronic component solutions are out of scope as per the exclusion criterion 1.

Similar to the “domain specificity” dimension of Eisenberg et al. [Eisenberg, 2002], we examined each platform to determine if it was optimized for a particular **electronic sub-domain**. For example, mBot [mBot, 2021] and BITalino [Silva, 2014; Bitalino, 2021] target *robotic vehicles* and *biomedical sensing* respectively. Platforms that have no specific focus, including Arduino, Lego Mindstorms [Lego, 2022], and littleBits [Bdeir, 2009] are labeled *not applicable*. Resnick and Silverman [Resnick, 2005] suggest that prototyping toolkits used in a K-12 education context should have “wide walls”, meaning that there should be no particular focus on a sub-domain because generic platforms allow children to expand their interests and passions. Outside of the classroom, however, this does not

	Specific Toolkits				Generic Breakout boards	Low-cost WiFi modules	Silicon vendor MCU dev. boards	FPGA dev. boards
Type of electronics	Type 2a (30%)	Type 2b (10%)	Type 2c (5%)	Type 3 (65%)	Type 2b	Type 2a	Type 2a Type 3	Type 2a
Electronic sub-domain	Wearables and textiles (15%) Robotic vehicles and drones (3%) AI/ML (2%)	Home automation (3%) Musical instruments (2%) n/a (70%)	Interactive paper (3%) Biomedical sensing (2%)		n/a	n/a	n/a	n/a
Promoted with user groups (multi-value)	K-12 Education (50%)	Makers (77%)	Electronic engineers (35%)		Makers Electronic engineers	Makers Electronic engineers	Makers Electronic engineers	Makers Electronic engineers

Figure 3.1: Categories and labels concerning the nature and application of the platforms.

necessarily hold true. In many cases—especially when considering makers—specialized toolkits are beneficial as they allow for faster prototyping in a particular sub-domain.

In addition to particular sub-domains, some electronics platforms are **promoted with user groups**. In particular, we saw toolkits that were advertised for *education*, *makers*, and *electronics engineers*, which we recorded accordingly. We noticed that this oftentimes is not the same group for whom the toolkit was originally designed. For example, the Arduino [Arduino, 2022] originally targeted designers but has had a lot of impact on education. Similarly, the Raspberry Pi [Pi, 2022b] was designed for education but also offers a compute module [Pi, 2021] very-much targeted at electronics engineers.

3.3.2 Assembly of Prototypes

The way(s) in which the modules in a prototyping toolkit are put together affects important qualities such as the speed of assembly and ease of modification of a prototype, its looks, and its durability. As shown in Figure 3.2 and described in this section, we identified three characteristics that consistently affect these ‘assembly qualities’.

	Specific Toolkits			Generic Breakout boards	Low-cost WiFi modules	Silicon vendor MCU dev. boards	FPGA dev. boards
Type of connection	Individual conductors (45%) Wireless (5%)	Multi-wire cables (27%)	Direct module-to-module (38%)	Individual conductors	Individual conductors	Individual conductors	Individual conductors
Connection mechanism (multi-value)	Friction fit (67%) Crocodile clips (8%) Screws (2%)	Magnetic (8%) Adhesive (3%) Wireless (5%)	Locking (7%) Thread (3%)	Friction fit	Friction fit	Friction fit	Friction fit
Connection topology	Star (45%)	Hybrid (25%)	Bus (30%)	Star	Star	Star	Star

Figure 3.2: Categories and labels concerning the assembly of individual elements into a working prototype.

We identified four major **type of connections** used in prototyping platforms. 38% of toolkits require *individual conductors* to be manually connected, e.g., wires between an Arduino and a breadboard or WiFi module, copper tape for interconnecting Cir-

cuit Stickers [Hodges, 2014], or conductive thread with the Lilypad [Buechley, 2008]. Around a third of all toolkits use *multi-wire cables*, such as the ribbon cables used in .NET Gadgeteer [Hodges, 2013]. 36% of prototyping toolkits use a *direct module-to-module* approach that physically interconnects modules without wires, such as Arduino shields [Adafruit, 2021c] that stack and littleBits [Bdeir, 2009] that clip together. Unfortunately, this often results in prototypes that are increasingly tall or long. Finally, 2% of toolkits do not require physical connections as they communicate *wirelessly*, such as SAM Labs [Labs, 2021].

Multi-wire cables and *direct module-to-module* approaches help with interconnection as users can see and feel how modules can be combined without composing a detailed schematic diagram. Blikstein [Blikstein, 2015] refers to these affordances as “tangibility mappings”. Tangibility mappings can “raise the ceiling” as they encourage exploring all possibilities. These approaches also encourage “a path of least resistance” [Myers, 2000] as errors are prevented. Some connectors are even entirely fool-proof by embedding a mechanical or magnetic poka-yoke constraint, such as littleBits [Bdeir, 2009].

The next characteristic further details the specifics of the **connection mechanism**, which has implications for the durability of the assembly, the tools that are required during construction, and the reusability of components. The majority of toolkits embed *friction fit* connectors, such as .NET Gadgeteer’s ribbon cables [Hodges, 2013] and Arduino’s headers [Arduino, 2022]. 8% of toolkits support modules that interconnect mechanically using a *locking* mechanism, for example a locking JST connector [Lego, 2022; mBot, 2021] or snaps [Buechley, 2005]. 6% of toolkits use *crocodile clips*. To interconnect modules more easily, 10% of toolkits support *magnetic* connectors. Magnetic connectors simply snap together but can also accidentally disconnect when quite a light force is applied. A few toolkits offer modules with strong fixations, such as *screws* (2%) or *stitches* using *thread* (4%). While these connections require using external tools, they also allow for the construction of a greater range of pieces and materials [Eisenberg, 2002]. Finally, in 2% of toolkits, namely Circuit Stickers [Hodges, 2014], the modules interconnect using an adhesive. Similar to stitches, adhesives make components harder to re-use.

The last characteristic related to the assembly of modules details the **connection topology**. The majority of toolkits (56%) require direct connections to the processing board in a *star* topology. In contrast, 32% of toolkits employ a *bus* topology, allowing modules to be daisy chained (e.g., Foxels [Perteneder, 2020] and Cubelets [Cubelets, 2021; Schweikardt, 2006]). Some toolkits (10%) also support a *hybrid* star/bus, allowing daisy chaining for some modules, while others need a direct connection to the processing board (e.g., .NET Gadgeteer [Hodges, 2013] and Pmod [Pmod, 2021]). Some bus configurations support automatic detection of the topology of an assembly, a feature that further bridges the gap between computational and physical construction kits, according to Eisenberg et al. [Eisenberg, 2002].

3.3.3 Deploying and Configuring

Figure 3.3 shows the three categories that relate to deploying and configuring electronics prototyping toolkits. Typically, **TYPE 2b** breakout boards do not require or support programming since they are operated through **TYPE 2a/c** components, so we added a *not applicable* label to all three categories.

	Specific Toolkits			Generic Breakout boards	Low-cost WiFi modules	Silicon vendor MCU dev. boards	FPGA dev. boards
Programming style	Physical (20%)	Software (75%)	n/a (10%)	n/a	Software configuration	Software configuration	Software configuration
Dependencies for programming (multi-value)	Fully self-contained (23%)	Connected wirelessly (12%)	Tethered to computer (73%)	n/a	Tethered to computer	Tethered to computer	Tethered to computer
	n/a (8%)						
Dependencies during deployment	Fully self-contained (80%)	Connected wirelessly (3%)	Tethered to computer (8%)	n/a	Fully self-contained	Fully self-contained	Fully self-contained
	n/a (8%)						

Figure 3.3: Categories and labels relating to deploying and configuring electronic prototypes built with the platforms in our survey. Note that the labels in two categories are not mutually exclusive.

The first characteristic **programming style** classifies all toolkits into two categories. One label groups all toolkits that are programmed using a *physical configuration* of modules in space (22%). Examples include MakerWear [Kazemitabaar, 2017], ReWear [Kazemitabaar, 2016], and littleBits [Bdeir, 2009] that allow specifying behavior by physically composing sensor and modifier modules. A second label groups all toolkits for which the behavior is specified in a *software configuration* (74%). Examples include the micro:bit [microbit, 2022], which is programmed using visual building blocks (i.e., Microsoft MakeCode [Devine, 2018]) or Python.

The two last labeling categories document external computing **dependencies for programming** and **dependencies during deployment**. Eisenberg et al. [Eisenberg, 2002] argue that communication between prototyping toolkits and desktop machines allows the toolkit to leverage the computational power of the desktop computer and the internet. While almost all modern electronics toolkits support communications with external computing devices, some different architectures are available. For example, both Cubelets [Cubelets, 2021; Schweikardt, 2006] and SAM Labs [Labs, 2021] are programmed wirelessly, but SAM Labs also requires a wireless connection to a desktop or tablet computer at all times to operate the final prototype.

3.3.4 Availability and Adoption

Figure 3.4 shows four characteristics of electronic prototyping platforms relating broadly to their availability and use. These are described in more detail in this section.

The category **existing use** documents whether a platform has been used for *only one-off prototyping*, to make *multiple copies*, or if it has been embedded and shipped in *commercial products*. We define *multiple copies* as five or more exact copies built with the same

	Specific Toolkits			Generic Breakout boards	Low-cost WiFi modules	Silicon vendor MCU dev. boards	FPGA dev. boards
Existing use	In commercial products (7%)	Multiple copies (7%)	Only used in one-offs (86%)	Multiple copies	In commercial products	Only used in one-offs	Only used in one-offs
Commercially available	Yes (67%)	No longer (2%)	Never (31%)	Yes	Yes	Yes	Yes
Third party use	Yes (86%)	No (17%)		Yes	Yes	Yes	Yes
Open source	Fully (45%)	Partial (42%)	Closed (13%)	Fully	Fully	Fully	Fully

Figure 3.4: Categories and labels relating broadly to the availability and use of the electronic prototyping platforms in-scope for our survey.

prototyping platform. Multiple exact copies are often desired for long-term experiments with multiple setups [Scott, 2011]. Going further, sometimes *commercial products* ship using a prototyping platform, such as the Deltamaker 3D printer [DeltaMaker, 2021] that embeds a Raspberry Pi, and the TriggerTrap SLR Camera trigger device [Kamps, 2021] that embeds an Arduino board. This suggests the platform is highly robust and competitively priced compared to a custom PCB. Interestingly, as far as we can tell, 84% of platforms (excluding the generic categories) have only been used to make one-offs.

The next category indicates if the platform was or still is **commercially available**. While many electronics toolkits are commercially available, several toolkits are *no longer* available (e.g., .NET Gadgeteer [Hodges, 2013]) or only available as research prototypes (e.g., Foxels [Perteneder, 2020] and ESLOV [ESLOV, 2021]).

The following category, **third party use**, records if the platform was used only by the team that created it (*first party*) or also by *third parties*. Naturally, all toolkits that have been commercialized (60%) have also been used by many, but we were pleasantly surprised to find that an additional 19% of toolkits – about half of the platforms that have never been sold as products – have been used by third parties. Examples include the use of d.tools [Hartmann, 2006] by third parties in workshops.

The final category in Figure 3.4 labels whether the engineering details of the platform are **open source**, either *fully* (36%), *partially* (48%, e.g., via well-documented specifications or an open source schematic but closed PCB layout), or completely *closed* (16%, more common for commercial products).

3.4 Analyzing the characteristics

Figures 3.5 and 3.6 depict most of the 13 characteristics we have captured across all 56 toolkits we reviewed. To make this information easier to access, our dataset and an interactive tool for exploring it are publicly available at <http://etclassification.com>. This allows researchers and practitioners to explore the dataset by sorting, filtering, and color-coding features and to analyze our labels and characteristics in more detail. In light of the evolving nature of many electronics toolkits, we intentionally used the GitHub

version control platform to host our dataset and interactive tool. Via GitHub ‘pull requests’, practitioners and researchers can update the dataset when existing toolkits evolve, and new generations of prototyping toolkits become available.

Besides the objective characteristics covered in our taxonomy, there are important holistic attributes of electronic prototyping toolkits, such as ease of use and the level of expertise required. These attributes, which often map directly to user needs, are somewhat subjective, which makes them harder to assess. We have developed an approach for comparing different toolkits by building on the objective characteristics of our taxonomy. In particular, we have evaluated all 56 toolkits across four more holistic characteristics by assigning weights to the labels of our objective characteristics.

The first two of these more holistic characteristics we estimated are the level of electronics expertise required and the level of programming expertise required. These characteristics inevitably vary between toolkits, but the variation has not, to our knowledge, been quantified in the literature. The third holistic characteristic evaluates the ease of construction of a prototype, independently of electronics and programming expertise. This relates to how fiddly and time-consuming the construction process is and also encapsulates the time required to make multiple copies of the same prototype. Motivated by the work of Hodges et al. [Hodges, 2020; Hodges, 2019a; Khurana, 2020], the fourth and final more holistic characteristic that we evaluate is the ease of moving from a prototype to a product. This considers the complexity of the pathway from the prototype to a more integrated, robust, compact, and cost-effective design that can be used for long-term deployments, for more extensive evaluation, or even as a low-volume product.

Expertise in electronics:	Type of connection Promoted with user groups Connection topology
Expertise in programming:	Programming style Promoted with user groups
Ease of construction:	Type of connection Programming style Connection mechanism Connection topology
Ease of moving from prototype to product:	Programming style Existing use Dependency during deployment Open source

Table 3.1: *The four more holistic (and somewhat subjective) characteristics we evaluated (left) and the set of objective characteristics upon which they are based (right).*

Table 3.1 gives an overview of the objective characteristics in our taxonomy that contribute to each of the aforementioned holistic characteristics. For example, we postulate that it is more convenient to construct a prototype with a toolkit that (1) does not require connecting individual wires, (2) can be programmed by physically interconnecting

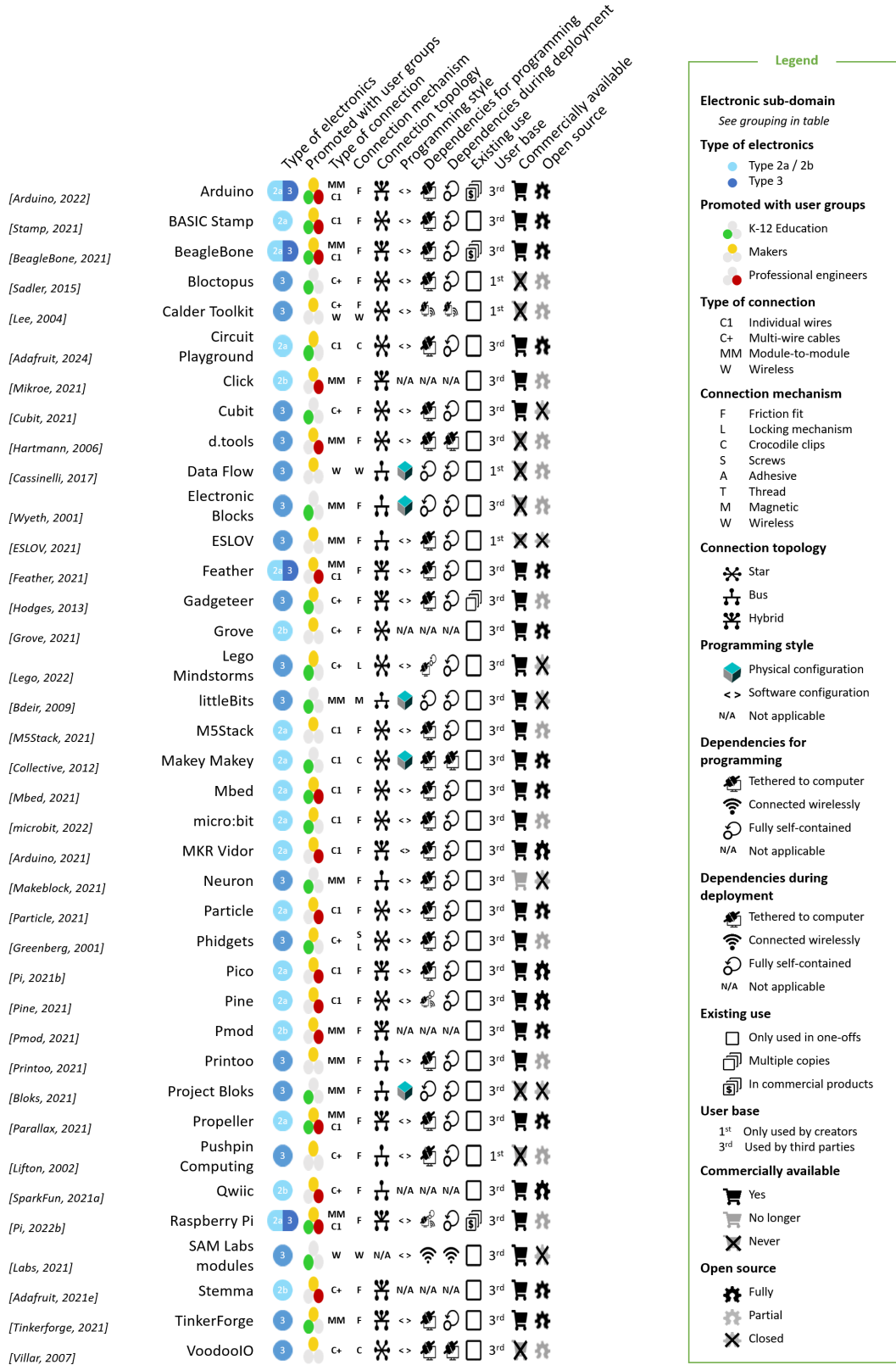


Figure 3.5: A visualization of the first part of our dataset.

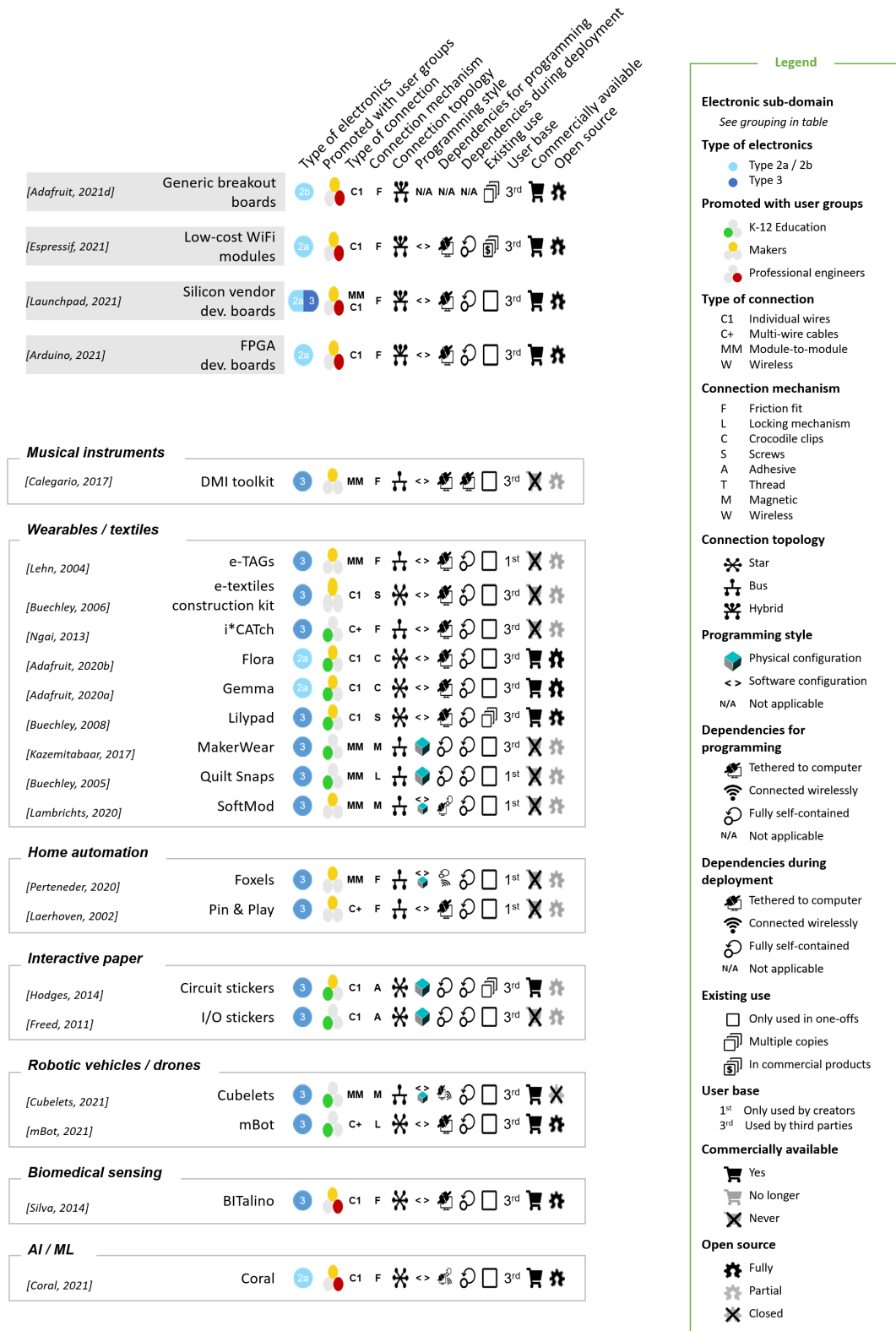


Figure 3.6: A visualization of the second part of our dataset. The four clusters, representing more than a single toolkit, have a gray background.

blocks, (3) does not require additional tools for connecting modules, such as adhesives or stitches, and (4) can be connected in a bus topology. In our calculation, all relevant characteristics contribute equally, and the weights are therefore distributed evenly across all labels within an objective characteristic. However, these weights can be adjusted according to personal preference or specific requirements in order to identify the platforms that are most suitable for specific prototyping settings. To allow others to adjust the weights in this way, we have made the holistic characteristics configurable as part of our interactive dataset^{3,4}.

Figure 3.7 illustrates how the electronics toolkits compare based on our holistic characteristics, using our default weights. Although the ranking obviously changes if the weights change, we are not aware of a more objective metric for assessing and comparing holistic characteristics of toolkits. Due to space constraints, only the five highest and lowest-ranking prototyping platforms for each of the four holistic characteristics are named in the figure. However, the histograms in the background of the figure visualize the distribution of all 56 electronics toolkits according to each of the more holistic characteristics.

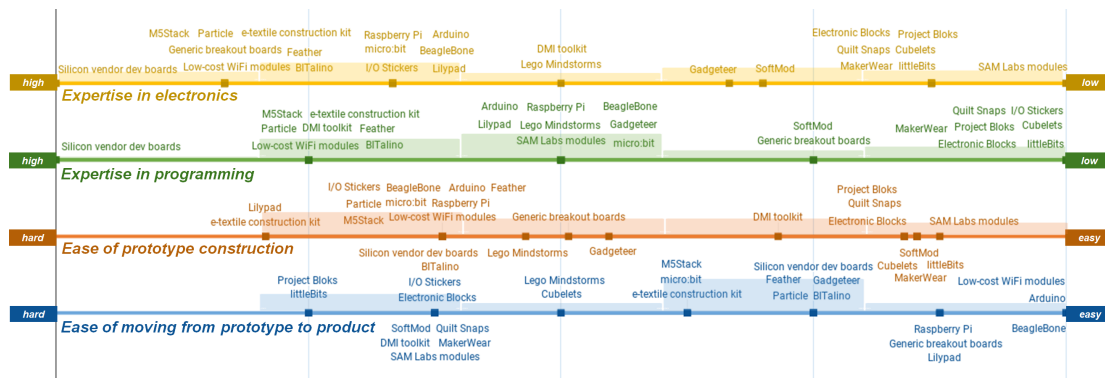


Figure 3.7: Prototyping platforms ranked according to four holistic characteristics, with ‘better’ on the right. Not all platforms can be named due to space constraints, but the shading indicates the distribution of all 56 platforms from this study across each characteristic. Note that rankings are all relative to the dataset. A larger version of this image can be found in Appendix A.1.

An interesting observation from Figure 3.7 is that platforms that require expertise in electronics often also require expertise in programming; similarly, platforms with low expertise requirements for electronics are often paired with low expertise in programming. There does not appear to be any obvious trade-off between these two holistic characteristics. The reverse is true of the other two holistic characteristics: very broadly, we can see from Figure 3.7 that for many toolkits supporting easy prototype construction, it’s hard to make a transition from prototype to product, and vice-versa.

3.5 Understanding How Electronics Toolkits are Used

Having developed a taxonomy for electronics toolkits, we wanted to get a deeper understanding of how users value the different characteristics in practice and uncover

any characteristics we may have missed. During the course of our research, we observed that there are hundreds of electronics toolkits targeted at professional electronics engineers and made available through numerous companies—but these are largely homogeneous, resulting in just a few categories of toolkits in our dataset. In contrast, a wide range of toolkits, many of which originate in academic research [Kazemitabaar, 2017; Cubelets, 2021; Hodges, 2013; Buechley, 2008; Hodges, 2014], are promoted for makers and for educators. Indeed, there seems to be a strong focus on the design and use of toolkits for education within the research community [Blikstein, 2013a; Eisenberg, 2002; Blikstein, 2015; Blikstein, 2013b; Resnick, 2005]. With such a strong focus on education, we were curious to evaluate if the needs of makers and electronics engineers are addressed by the current electronics toolkit offerings.

To this end, we conducted an online survey to get an understanding of preferences and experiences with prototyping electronics by non-professional (i.e., makers) and professional electronics engineers. The online survey was aimed at people who have built at least one electronic device prototype and consisted of both multiple-choice and free-form responses for a total of 75 questions. By formulating simple questions and grouping them in matrix Likert scales, we estimated filling in the survey would take 15 minutes on average. We employed a snowball sampling approach to reach as many electronics practitioners and experts as possible. We initially distributed the survey via email, social media, and community-specific platforms. The study ran for one week in April 2020, and 122 people participated (21% response rate based on unique page views). The time to complete the survey was 5-57 minutes (median = 13.5 minutes).

3.5.1 Our Respondents and Their Prototyping Experience

84% of our respondents identified as male, 13% as female, and 3% self-described as ‘other’ or did not disclose gender. 16% of the participants were aged 18-24; 41% aged 25-34; 20% aged 35-44; 15% aged 45-54; 4% aged 55-64; and 2% aged 65-74. 2% of our respondents did not disclose their age. 46% of respondents were located in Europe, 32% in North America, and 20% in Asia, with 2% not disclosing a geographic region.

We also asked respondents about their backgrounds: 40% self-identified as an electrical or electronics engineer (hereafter abbreviated to electronics engineer), 18% as a mechanical or mechatronics engineer, 15% as an engineer with a different specialization, 17% as a product or industrial designer and 59% as a computer scientist or programmer. In addition, 69% self-identified as a researcher, 31% as a student (above K-12 education), 55% as a maker/DIY builder, 17% as a hobby programmer, and 1% as retired. None of the preceding options were mutually exclusive—respondents were free to select all they felt applied. 3% of our respondents selected none.

We did an initial analysis of the survey data, looking for differences based on the background of our 122 respondents. We found the most insightful way to pivot our dataset was based on whether respondents self-identified as having a background

in electrical and electronics engineering or not. Put another way, the electrical and electronics engineers we surveyed reported different priorities and attitudes regarding electronics prototyping compared to the respondents who don't have the same technical grounding in electronics. For this reason, we split our respondents into two mutually exclusive groups of 49 electronics engineers vs. 73 respondents with backgrounds in other disciplines, and we have used that split for all the analyses presented hereafter.

Following the demographics questions, respondents were asked how frequently they build electronics prototypes in various device categories. They report having built electronic devices several times (meaning more than once or twice) for the following categories: learning and fun (70%), wearable electronics (45%), home automation systems (43%), robotic systems excluding wheeled robots (42%), games and toys (33%), office workplace devices (32%), wheeled robots (25%), interactive textiles (25%), biomedical sensing (24%), flying vehicles (12%), and in-vehicle devices for cars (8%).

When splitting the data based on formal engineering expertise, we noticed electronics engineers build significantly more electronic devices across all categories compared to respondents from other disciplines (Mann-Whitney $U=200966.5$, $p<0.001$, $Z=-5.61$).

3.5.2 Use of Prototyping Platforms

We presented participants with a list of the commercially available electronics toolkits covered in our detailed literature review and asked them to specify in a range how many they'd heard of, how many they had experimented with once or twice, and how many they used often. Participants were aware of the existence of many different platforms, with 97% of respondents having heard of more than 4 and 75% of more than 7. In terms of the hands-on experience of our respondents, 33% had experimented with 7 or more platforms, 65% had experimented with 4 or more, and 98% had experimented with at least 2 platforms. This indicates that our snowball sampling approach had successfully solicited respondents with meaningful experience with prototyping toolkits.

Our data also shows that 8% of respondents use 2 or 3 platforms often, 28% use 4-6 platforms often, and 6% use 7 or more different platforms often. This is interesting because it shows that the respondents who are regularly building prototypes tend to leverage 4-6 different toolkits—they don't appear to get comfortable with just one or two toolkits and rely on only these. When comparing electronics engineers to all other respondents, we noticed that 42% often use more than 4 platforms compared to only 19% of the respondents from other disciplines (The Fisher's exact test: $N = 116$, $p < 0.05$, odds ratio = 3.0).

We were also curious why participants did not use a wider set of platforms, and we addressed this through a multiple-choice (multiple-answer) question. 70% of the participants simply responded they were happy with the platforms they already use, with the majority of these (59%) indicating that it was not clear what benefits a new platform would bring.

In terms of the friction associated with adopting another platform, (30%) of our respondents indicated “I do not want to learn another platform”, (27%) indicated “The platforms are not well established and might be deprecated in the future”, (25%) checked “I do not want to pay for another platform”, (26%) indicated “The community support is not good enough”, (24%) checked “Documentation or examples are too limited”, for (21%) “The platform seems too limited”, (19%) indicated “The platform seems too complicated in use”, and (15%) said that “Too little detailed information is available about the platform”. The low response rate on these latter options suggests that despite the general trend of making electronics platforms simpler to learn and use, these factors are not necessarily sufficient for switching to another prototyping toolkit.

3.5.3 Important Characteristics of Prototyping Platforms

In a final series of questions on prototyping platforms, we asked participants to rate 26 characteristics based on how important they are when selecting a prototyping platform. These characteristics include the objective and more holistic characteristics listed earlier in this chapter, reformulated to make them easier to understand where appropriate. We also added nine new characteristics in order to highlight potential gaps in our previous analysis. Figure 3.8 shows the importance of each characteristic for both electronics engineers and respondents from other disciplines. The Likert scale answers for all characteristics were: “always unimportant”, “usually unimportant”, “usually important”, and “always important”.

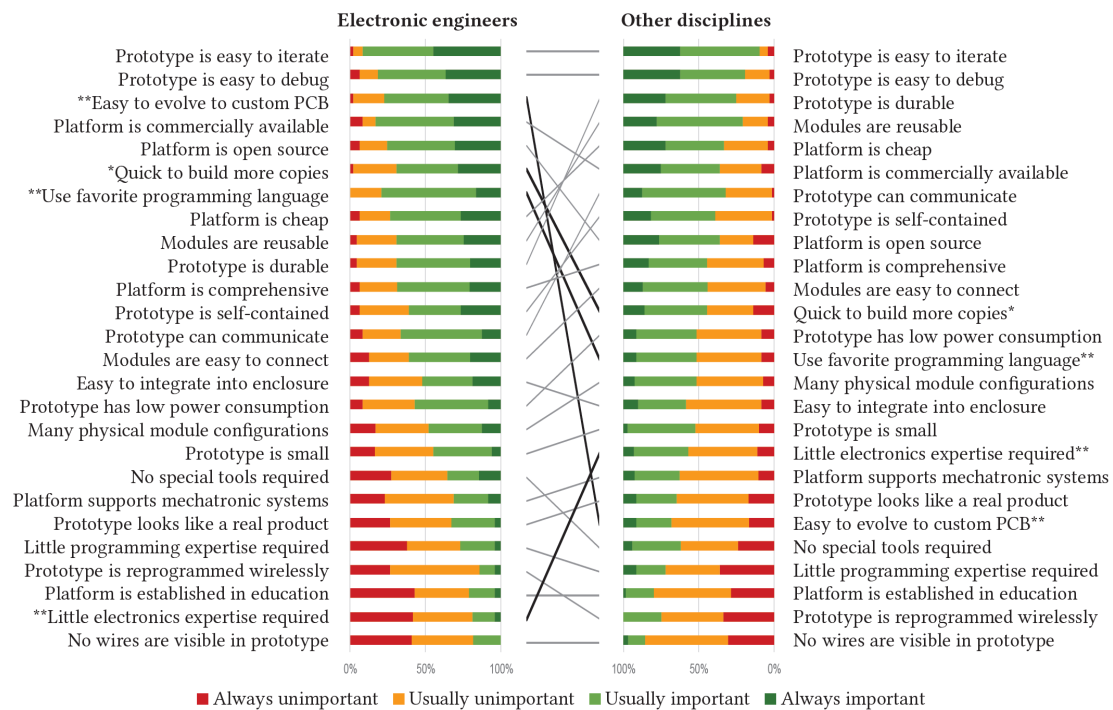


Figure 3.8: Comparing the ranking of the importance of different characteristics between electronics engineers and respondents with a different background. We determined significance using Mann-Whitney U tests (* $p < 0.05$, ** $p < 0.001$).

When analyzing the results, a Mann-Whitney U test showed that the characteristic “easy to evolve to a custom PCB” is significantly more important to electronics engineers than other respondents ($U = 836.5$, $Z = -5.16$, $p < 0.001$, $r = 0.47$). A significant difference was also found for the characteristic “Quick to build more copies” (Mann-Whitney $U = 1351$, $Z = -2.31$, $p < 0.05$, $r = 0.21$). These results suggest that electronics engineers find it more important to replicate prototypes, a process that often involves designing a custom PCB.

We also found a significant difference between electronics engineers and other disciplines for the characteristic “Use favorite programming language” (Mann-Whitney $U = 1164.5$, $Z = -3.46$, $p < 0.001$, $r = 0.31$), with other disciplines being less concerned about the choice of programming language. Conversely, and quite understandably, respondents from other disciplines were keen to see “Little electronics expertise required” whereas the electronics engineers were almost ambivalent about this (Mann-Whitney $U = 2393.5$, $Z = 3.79$, $p < 0.001$, $r = 0.35$).

The characteristics in our survey included three technical characteristics not covered in our taxonomy: support for wireless communication, requirements for low power consumption, and the support offered for debugging. The first two of these, frequently reported in datasheets and online reviews [MakeMagazine, 2021], were not rated as important selection criteria by our participants. However, debugging is very important, according to our respondents. Unfortunately, debugging is challenging to characterize and, as explained in Section 3.3.3, is largely affected by software aspects of a toolkit. Evaluating this across different electronics toolkits remains an opportunity for future research (see Section 3.6). An open question allowed respondents to suggest additional important characteristics. Four valued compatibility with specific operating system(s), and two prioritized their current prototyping platforms because they were on hand for immediate use.

3.5.4 Experiences of Type 1 Prototyping and Scaling Up to Multiple Copies

We also asked participants questions about their experience prototyping with **TYPE 1** electronics, i.e., with discrete electronic components. The majority (64%) reported using solderless breadboards often, followed by soldering TH components (48%), soldering components to a custom PCB (43%), soldering Surface-Mounted Device (SMD) components (36%), using pre-built modules e.g. WiFi and battery charging modules (36%), soldering components to strip boards (37%), designing custom PCBs for SMD components (33%), using pre-built breakout boards for SMD components (22%), designing custom PCBs for TH components (21%). When using Fisher’s exact test on the number of people who use these tools often, we detect significant differences between electronics engineers and other respondents regarding designing custom PCBs for SMD components (31 vs. 8, $N = 121$, $p < 0.001$, odds ratio = 13.4), designing custom PCBs for TH components (18 vs. 7, $N = 120$, $p < 0.001$, odds ratio = 5.5), soldering components to custom circuit boards (33 vs. 18, $N = 121$, $p < 0.001$, odds ratio = 6.1), soldering

components to strip boards (25 vs. 19, $N = 121$, $p < 0.01$, odds ratio = 2.9), and soldering SMD components (33 vs. 10, $N = 121$, $p < 0.001$, odds ratio = 12.4).

We also wanted to know to what extent those who prototype electronic devices start with one of the toolkits listed in this chapter with a view to transitioning to a custom PCB later in the process. We learned that 53% of all respondents often start the process with **TYPE 2a/c** development boards, 44% often start with solderless breadboards, and 30% often start with **TYPE 2b** breakout boards. Of those using **TYPE 2b** toolkits, one-half (15% of respondents) have used a system of modules such as Grove [Grove, 2021], Pmod [Pmod, 2021], or Click [Mikroe, 2021]. Note that these responses are not mutually exclusive, which is consistent with our earlier observation that toolkits of Types 1 and 2 are often used in conjunction with each other. Only 10% of our respondents often started the prototyping process with a **TYPE 3** toolkit such as .NET Gadgeteer [Hodges, 2013] or littleBits [Bdeir, 2009] with the intent to transition to a custom PCB.

Next, we asked participants if they ever made multiple copies of a prototype and how many. We were surprised that 94% of electronics engineers and 73% of other disciplines reported making multiple copies; we thought it would be fewer. Our chi-squared test showed that this difference is significant ($\chi^2(6, N = 122) = 21.59, p < 0.01, \phi=0.42$). Figure 3.9 shows the distribution of the maximum number of copies for both groups. While the majority of respondents who are not electronics engineers have made fewer than 10 copies, 48% of them have made 10 or more. Across all respondents, 24% had made more than 100 copies, 9% more than 1000, and 5% more than 10,000. Again, these numbers were higher than we expected.

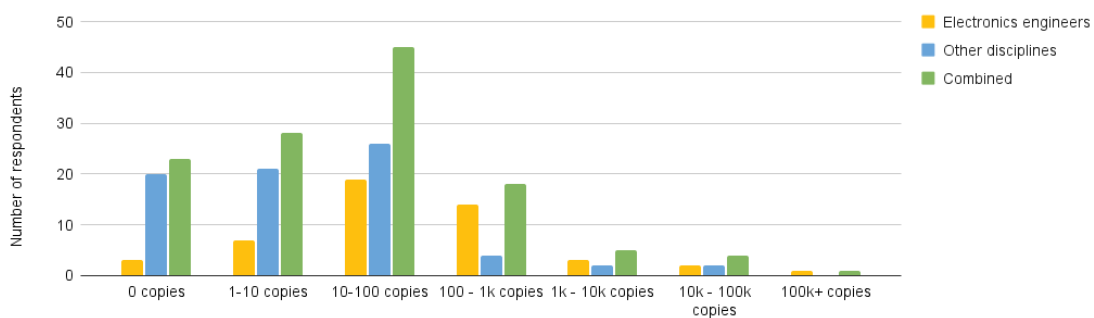


Figure 3.9: The number of copies of electronic prototypes made by electronics engineers and respondents from other disciplines.

Finally, we asked participants how frequently they use different approaches to transition from a one-off prototype to multiple copies. 39% of respondents reported they often make copies using a custom PCB; there was a strong difference between electronics engineers and respondents in other disciplines (69% vs. 16%, the Fisher’s exact test: $N = 116$, $p = < 0.001$, odds ratio = 11.2). In contrast, only 12% of respondents often make copies using the same prototyping platform as used for the one-off prototype, and only 4% switch electronics toolkits to facilitate multiple copies.

3.6 Discussion

For the participants in our survey and the current generation of electronics toolkits they use, it appears that electronics and programming expertise requirements are not a significant barrier—either for those with an electronics engineering background or those from other disciplines. This contrasts with a theme we see across many toolkits, namely a drive to lower the programming and electronics skills required for device prototyping. This trend may be driven by the use of electronics toolkits by children, for which researchers suggest making the technology “as simple as possible – and maybe even simpler” [Resnick, 2005]. Only a small portion of participants (19%) reported they would not use some of the toolkits in our taxonomy because they seem to complicated to use. This observation is also supported by the following free-form comment left by a participant at the end of our survey: *“Current modular kits make electronic design easier, but the fundamentals are missed out on, and users that use these platforms struggle [with] debugging (i.e., death by over-abstraction)”*. Having said this, our snowball sampling approach may have solicited respondents with quite some experience in prototyping toolkits; as mentioned above, 65% of them had experience with at least 4 different toolkits. Given this potential skew, we believe it’s important to continue lowering the bar for electronics development to stimulate interest in STEM and to empower more individuals and communities to engage in electronics prototyping. However, for users familiar with prototyping toolkits, the level of expertise required in electronics or programming appears to be sufficiently low already, and this group of users might value other features more.

Our respondents did reveal several qualities of prototyping platforms that both electronics engineers and users from other disciplines find important. The most highly rated characteristics across all respondents were ease of iteration and ease of debugging, attributes that naturally relate to our holistic characteristic of “speed of construction”. It seems that a future toolkit that speeds up the prototyping process without constraining the artifacts that can be built (as many of the commercially available **TYPE 3** toolkits currently do) would appeal broadly and could add much value. We would certainly ask participants more about this in a future survey.

Another characteristic that was ranked relatively high by all our respondents was low cost. This is corroborated by the free-form comment from one of our participants: *“They [the toolkits] are often ludicrously expensive to accommodate unneeded features, or feel too much like an end product in quality.”* As researchers, when we develop new toolkits, we tend to focus on adding features in service of adding value, and product designers may be tempted to improve the form, fit, and finish of toolkits, but this user feedback is a timely reminder that constraining cost can be as valuable as adding functionality.

We were somewhat surprised that the vast majority of our respondents (81%), independent of their background, have engaged in making multiple copies of prototypes.

Over 50% have made over ten copies, and over 20% have made over 100 copies of a prototype, again indicating a bias in our snowball sampling towards more experienced makers. This appears to reflect a significant group of people who are moving beyond demonstrating the feasibility of their ideas via a lab-bound prototype to a deployment stage, frequently involving tens or hundreds of devices, if not more. This transition is typically supported by moving from a toolkit-based prototype to custom-designed PCBs, a process that, in turn, benefits from the availability of open-source design information. However, 92% of our non-electronics engineering respondents do not often design custom PCBs. This observation is also supported by comments of respondents, such as: *“I like to get into PCB design – but often find it daunting”* and *“When designing PCBs, I found it hard to understand differences and packaging sizes of components and its implications on my design”*.

The holistic characteristic of “Ease of moving from prototype to product” that we introduced in Section 3.4 should be a useful indicator of the ease of moving to a custom PCB, but we would like to evaluate this more rigorously and formalize the selection of weights assigned to the objective characteristics that underpin the calculation. It may be possible to personalize recommendations for prototyping platforms based on the preferences, expertise, and needs of individual users. Our previous analysis also indicated that toolkits that more readily support a transition from prototype to product may be liable to complicate the prototyping process, something that warrants careful consideration. Finally, if more objective characteristics are made available in future versions of the taxonomy, additional holistic characteristics, such as ease of debugging, could be included.

One respondent highlighted a potentially fruitful research direction: *“A prototyping toolkit that makes this process [transitioning from prototypes to PCB] easier may make a huge difference”*. Although systems like Scanalog [Strasnick, 2017] help with component selection and circuit design, to our knowledge no existing electronics toolkits have been designed with the transition from a prototype to a more integrated PCB solution in mind. However, the value of doing so has recently been highlighted in the literature [Hodges, 2020; Hodges, 2019a]. In addition to the PCB design process, other barriers to making multiple copies have been reported in the literature [Khurana, 2020], and one of these was also raised by our respondents: *“it [prototyping] has gotten so much easier over the years... part identification/procurement for small volume runs that is affordable is the biggest hassle”*, pointing to another area of future research.

Our study results show that respondents use electronics toolkits for prototyping a variety of interactive and ubiquitous computing devices, such as systems for experimentation and fun, wearable electronics, and home automation solutions. This is consistent with some of the specific domains targeted by electronics toolkits and reported in our literature survey (see Figures 3.5 and 3.6).

In general, we believe there is a symbiotic relationship between electronics toolkit

availability and adoption in that when more tools are available, more ideas are explored, and novel potential is revealed, driving further demand. In the coming years, we imagine there will be a growing set of electronics toolkits to support AI and machine learning applications, complementing Coral [Coral, 2021]. Having said this, our respondents did not report much interest in textile interfaces or biomedical sensing.

The current version of our taxonomy, consisting of 13 objective characteristics and their corresponding labels, mainly focuses on hardware aspects of toolkits. During the course of this work, it has become clear that it could be extended to include aspects of programming such as development platform compatibility, the availability of software libraries, and support for debugging. In terms of the latter, we are aware of several novel debugging and inspection techniques and tools for electronic prototyping, such as BiFrost [McGrath, 2017], WiFrost [McGrath, 2018] and Scanalog [Strasnick, 2017], but more research is needed to characterize what makes platforms easy to debug.

On reflection, it could also be useful to include the more standard technical specifications that have been reported elsewhere, such as operating voltage requirements, processor speed, and memory, so that all pertinent information is available in a consolidated form. This would likely involve breaking out our four ‘generic’ categories (e.g., ‘Low-cost WiFi modules’) and several other aggregated categories (e.g., ‘Arduino’) into specific products. 59% of our survey participants reported that they were not clear about the benefits offered by prototyping platforms other than the ones they already use, but such a central repository of characteristics could help with this.

In developing the taxonomy presented in this chapter, we aimed for a generalized classification of prototyping tools and platforms. While this provides a broad overview, it can sometimes oversimplify the diverse experiences these tools offer. For example, using an Arduino with breakout boards versus shields results in notably different experiences: the former requires manual wiring and hands-on assembly, while the latter allows for easier connections and often comes with tailored software libraries. These differences affect both the assembly process and the software development approach. Additionally, variations in software programming styles due to different assembly methods can lead to varied user experiences. A platform like SoftMod, which supports both hardware assembly and software programming, can result in distinct interaction models, complicating their categorization within a generalized taxonomy. Thus, while our taxonomy captures broad trends, it may miss specific nuances of user experiences tied to different tools and methods. Future work should aim to refine this framework to better represent the complexities of hardware assembly and software development in electronics prototyping.

Finally, we must acknowledge that while our survey involved a good number of respondents, our snowball sampling approach may have skewed towards more experienced users of prototyping toolkits that are mostly male (84%) and located in the Western world (78%). This might have led to biases in our data that we are keen to address in

future surveys. Indeed, with electronics prototyping becoming increasingly accessible to a global population with diverse backgrounds, experiences, and needs, more studies are needed to address the needs of specific user groups. We, therefore, see our work as a starting point that we hope other researchers can build upon.

3.7 Summary

In conclusion, we hope that the analysis of 56 electronics prototyping toolkits presented in this chapter provides valuable insights for researchers and practitioners within the community. This chapter directly addresses the research goal of understanding the diverse needs of users (**G1**) by complementing existing surveys with the development of 13 objective characteristics that go beyond the technical specifications typically reported. These characteristics were designed to capture the holistic user experience, reflecting the more subjective yet essential aspects of toolkit usability, which are crucial for evaluating how well these tools meet the practical demands of various user groups.

Additionally, this chapter contributes to answering the research questions related to identifying the challenges users face when selecting and integrating hardware and software components (**Q1**). By introducing the concept of holistic characteristics, we provide a framework that more naturally represents typical user needs, enabling direct comparisons between toolkits. This approach not only enriches the existing body of knowledge but also offers a practical tool for users and researchers to better assess and select the appropriate prototyping platforms for their specific needs.

We encourage readers to explore our dataset via <http://etclassification.com> and GitHub, where they can delve deeper into the findings and apply them to their own work. The results of the survey of 122 electronics toolkit users presented in this chapter further corroborate and complement our first-hand analysis, aligning with the research goal of capturing diverse user needs and preferences.

By highlighting the strengths and weaknesses of existing toolkits, this chapter has identified future research directions that align with the goal of bridging the gap between hardware and software compatibility (**G2**, **G3**). We hope that the work presented in this chapter will inspire further research outside the scope of this dissertation in the development of electronics toolkits that cater to both novices and experienced users, supporting the broader aim of making prototyping more accessible and scalable. Ultimately, we envision a future where prototyping toolkits evolve to meet the needs of a diverse user base, enabling the creation of innovative interactive and ubiquitous computing solutions.

PLUG-AND-PLAY HARDWARE THROUGH CIRCUITGLUE

Motivation

Drawing from the insights of the online survey discussed in Chapter 3, it became apparent that hobbyists and professional engineers are reluctant to adopt **TYPE 3** prototyping kits due to their closed ecosystems and the difficulties in incorporating new or customized modules. This feedback pointed to a need for solutions that offer the ease of use found in closed-ecosystem kits with the flexibility offered by open systems. To bridge this gap, we developed CircuitGlue as a novel *hardware glue*, enabling the straightforward integration of diverse **TYPE 2** hardware components (**G2**) without constraining users into a closed ecosystem. Its software-configurable header facilitates hardware assembly by automatically changing the function of each pin, allowing users to effortlessly customize connections to suit a wide array of third-party components. As CircuitGlue is positioned between a microcontroller and off-the-shelf hardware component, CircuitGlue offers the same flexibility as **TYPE 2** breakout boards and modules while maintaining the same ease of use as **TYPE 3** modular systems.

Furthermore, this chapter also contributes the result of a formative and preliminary user study, providing an answer to research questions **Q1** and **Q2**. The formative study involved in-depth discussions with an educator, a hobbyist, and an electronics engineer, shedding light on their respective journeys through the prototyping landscape. The preliminary user study compares CircuitGlue to traditional prototyping approaches, further reinforcing the necessity for new prototyping tools.

This chapter is based on the conference proceedings paper “CircuitGlue: A Software Configurable Converter for Interconnecting Multiple Heterogeneous Electronic Components”, which was published in the “Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies” [Lambrichts, 2023]. The paper was presented at the IMWUT conference in 2023 in Cancún, Mexico.

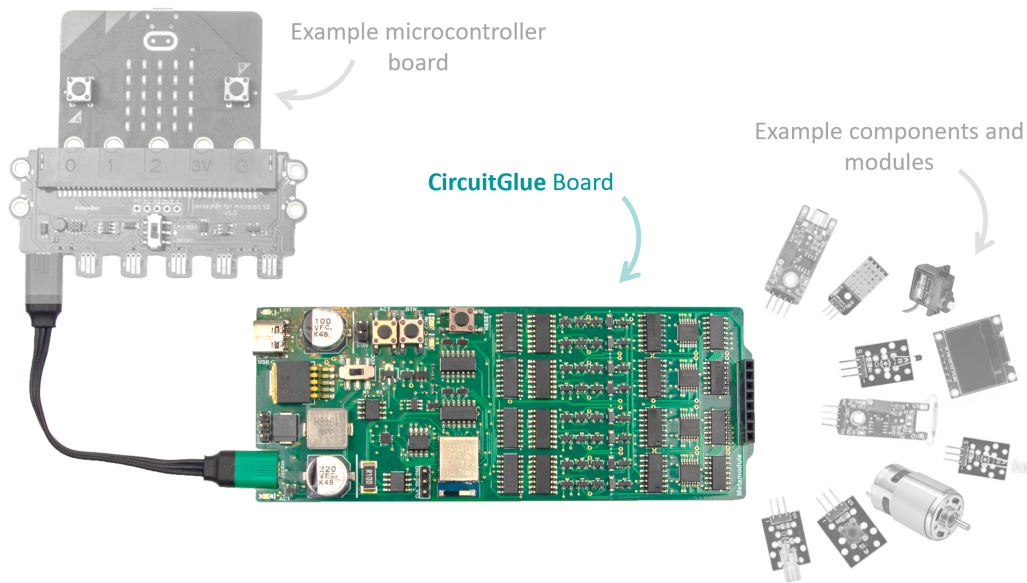


Figure 4.1: *CircuitGlue is a novel electronic prototyping board that allows a wide variety of off-the-shelf electronic components and modules to be connected to a software configurable header (at right). After configuration and connection, modules work instantly and are compatible with each other independent of the voltage levels, interface types, communication protocols, and pinouts they use.*

4.1 Introduction

With the growing availability and popularity of physical computing, people with increasingly diverse backgrounds are prototyping electronic circuits. As discussed in Chapter 2, **TYPE 1** electronics, like breadboards and jumper wires, are among the most common tools for this, but this electronic prototyping style requires significant electronic expertise as all components are wired individually. For some components that are only available in small package sizes or need custom circuitry, **TYPE 2** electronics have been created, such as the Adafruit BNO055 [Adafruit, 2022] and ESP8266 [Espressif, 2022a] breakout boards. Although this style partially eases prototyping because only the most essential connections are exposed, these modules are typically manufactured by different parties, and their operating voltages, interface types, communication speeds, protocols, and physical connections are often incompatible. Commercially available power and protocol conversion modules, such as the Sparkfun Buck-Boost Converter¹ and Adafruit FT232H² breakout board can help in this process, but selecting and interfacing between all components appropriately still requires a good understanding of electronics. Although breakout boards and development boards ease the design and creation of custom prototypes, selecting and interconnecting these components is often difficult for novices [Mellis, 2016]. To empower more people—especially those with non-electronics backgrounds—to prototype sensor systems, various integrated modular

¹ <https://www.sparkfun.com/products/15208>

² <https://www.adafruit.com/product/2264>

platforms have been developed. These **TYPE 3** electronics consist of a set of modules specifically designed to plug together without the need to study technical specifications in datasheets or to acquire any third-party components. Popular examples include .NET Gadgeteer [Hodges, 2013], littleBits [Bdeir, 2009], and LEGO Mindstorms [Lego, 2022]. However, the online survey in Chapter 3 revealed that hobby makers and engineers are often non-inclined to use these **TYPE 3** prototyping kits as they do not want to lock themselves into ecosystems [Lambrichts, 2021], and it can be hard for individuals to extend them with new modules.

To combine the versatility and extensibility of **TYPE 2** electronics and the ease of use of **TYPE 3** prototyping kits, we present CircuitGlue. CircuitGlue enables novices to prototype interactive systems quickly and easily without being locked into a particular ecosystem. CircuitGlue is an electronic prototyping board (Figure 4.1) that exposes eight “programmable” header pins that directly interface with a wide variety of third-party components and modules. Each of the eight header pins can be programmed to either connect to ground, output a specific voltage, or support an analog or digital reading or signal generation. Once the CircuitGlue board is configured, modules are immediately operational for testing or integration in a prototype—they are powered, and data and sensor readings are available on a digital bus. CircuitGlue also implements the Jacdac protocol stack [Devine, 2022], which means a standardized, abstracted representation of the component is presented, which allows the component to be integrated with code using any of the Jacdac-supported programming paradigms, including Microsoft MakeCode. Afterward, our complementary software environment further assists the user by showing them how to connect the module directly to a development board. In this way, the CircuitGlue board is ‘freed up’ and can then be reused to interface with another module or component. This approach helps novices in electronics to assemble advanced prototypes in iterations.

The core contribution of this work is CircuitGlue, a novel intelligent software-configurable prototyping board that facilitates interconnecting and testing various heterogeneous electronic components and modules. More specifically, we contribute:

- The CircuitGlue board, exposing eight programmable header pins that are configurable in software to drive a wide variety of electronic components. We benchmark our board’s design via a technical evaluation.
- A software architecture that makes the functionality of electronic modules and components available on the Jacdac communications bus. This makes many commercially available electronic modules compatible with each other and the Jacdac ecosystem.
- A demonstration of how CircuitGlue facilitates and enriches electronic prototyping workflows and helps with testing components and building electronic prototypes.
- A preliminary user evaluation reporting on the utility of CircuitGlue for novices in electronics.

4.2 Walkthrough

This walkthrough demonstrates how Sam, a novice prototyping enthusiast, uses CircuitGlue to prototype a smart desktop fan. The fan is powered by a DC motor and consists of a temperature sensor and a presence sensor to automatically power the fan when the temperature is too high, and presence is detected. Prototyping this interactive system with a microcontroller-based development board, such as the BBC micro:bit, would typically require inspecting the datasheet of all three components, finding and ordering an additional motor driver board as well as a DC-DC voltage converter, and figuring out the correct wiring of all these components as shown in Figure 4.6. However, Sam is uncertain about the exact workings of all these components and oftentimes does not understand the myriad of characteristics provided in datasheets. Therefore, Sam uses the CircuitGlue platform in the prototyping scenario below, allowing him to interconnect the heterogeneous set of electronic components needed in this project.

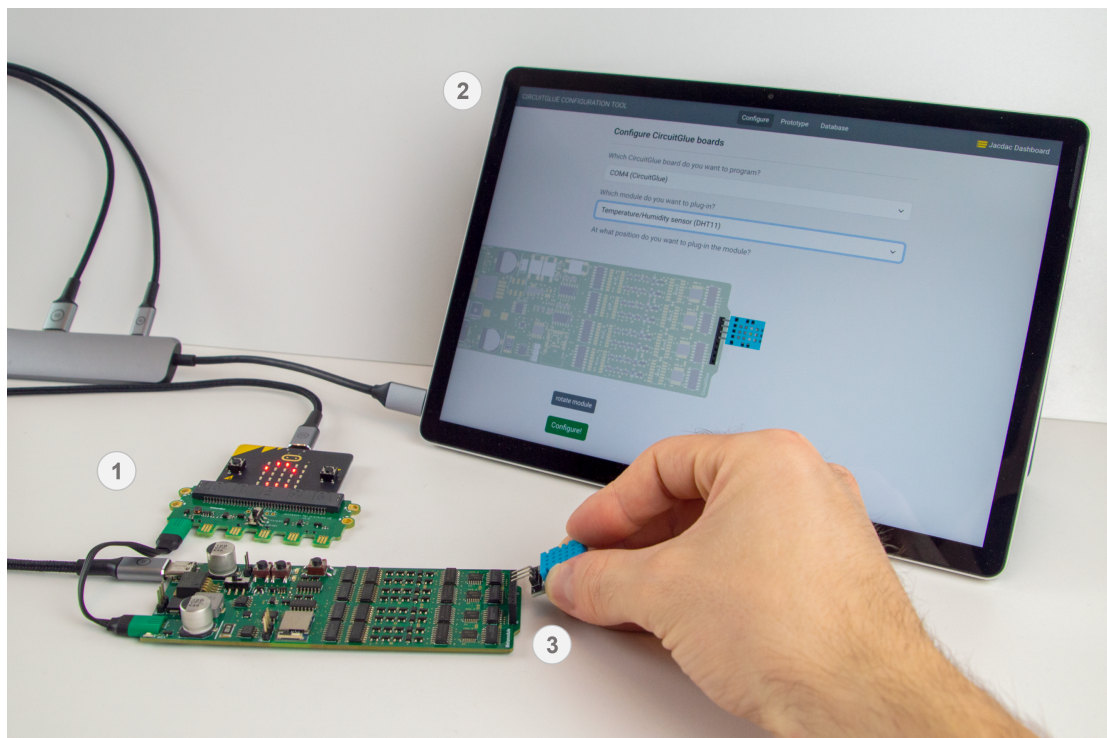


Figure 4.2: Configuring the CircuitGlue board to drive the temperature sensor by (1) connecting the CircuitGlue board to the computer and micro:bit, (2) configuring CircuitGlue by selecting the temperature module in the configuration tool, and (3) plugging the temperature sensor module into the programmable header.

Sam starts with the temperature sensor module. Instead of looking up detailed temperature sensor characteristics in its datasheet, Sam connects a CircuitGlue board to his computer with a USB cable and to the micro:bit with a Jacdac cable and a micro:bit Jacdac adapter (Figure 4.2 - 1). Next, he opens the CircuitGlue configuration tool in a browser and selects the temperature sensor (DHT11) from the list of components currently implemented in CircuitGlue. A visual representation of the CircuitGlue board

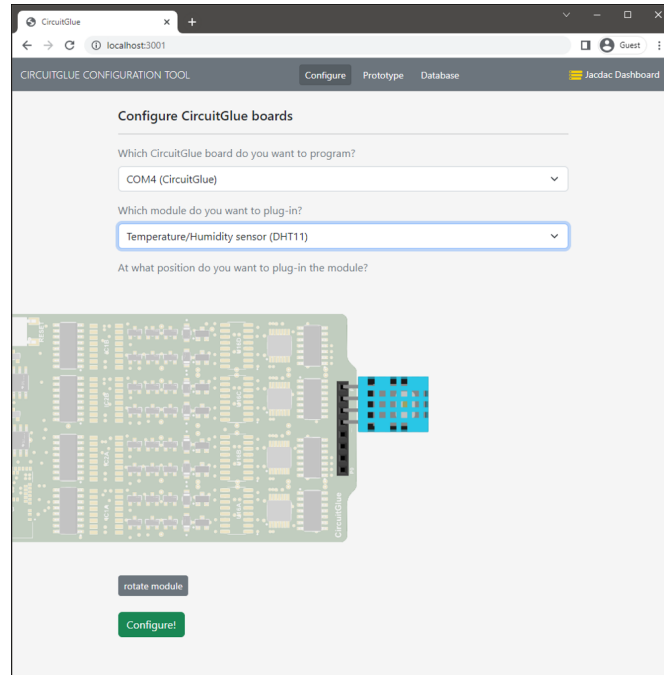


Figure 4.3: Web-based CircuitGlue configuration tool.

and connected temperature sensor is shown (Figure 4.2 - 2 and Figure 4.3). After the configuration tool has automatically configured the CircuitGlue board, Sam takes the temperature sensor and plugs it into the programmable header matching the position shown on-screen in the configuration tool (Figure 4.2 - 3). When browsing to the Jacdac dashboard³, he immediately sees a “digital twin” of the temperature sensor with live sensor readings (Figure 4.4). Sam now writes application logic for the temperature sensor by programming the micro:bit using, for example, Microsoft MakeCode⁴. MakeCode is one of several programming solutions that support the Jacdac communication protocol, and as such, all components available on the digital Jacdac bus are available as blocks in MakeCode (Figure 4.5).

Next, Sam connects a second CircuitGlue board to his computer and the same micro:bit and uses the CircuitGlue configuration tool to select the DC motor (Brushed - 12V DC). As he doesn’t need to add a motor driver module and external 12V power supply, the DC motor is instantly operational and visible via the Jacdac dashboard after plugging the motor directly into the CircuitGlue board. Like the temperature sensor—the DC motor is now also available as a block in MakeCode. Sam then programs the micro:bit to power the DC motor when the temperature reading exceeds 25° C.

When Sam is happy with his CircuitGlue-based prototype, he can use the *circuit diagram generator* in the CircuitGlue configuration tool to help him wire the temperature sensor and DC motor directly to the micro:bit, freeing up the CircuitGlue boards. As shown in Figure 4.6, this feature uses the knowledge of the characteristics of all components in

³ <https://microsoft.github.io/jacdac-docs/dashboard>

⁴ <https://makecode.microbit.org/>

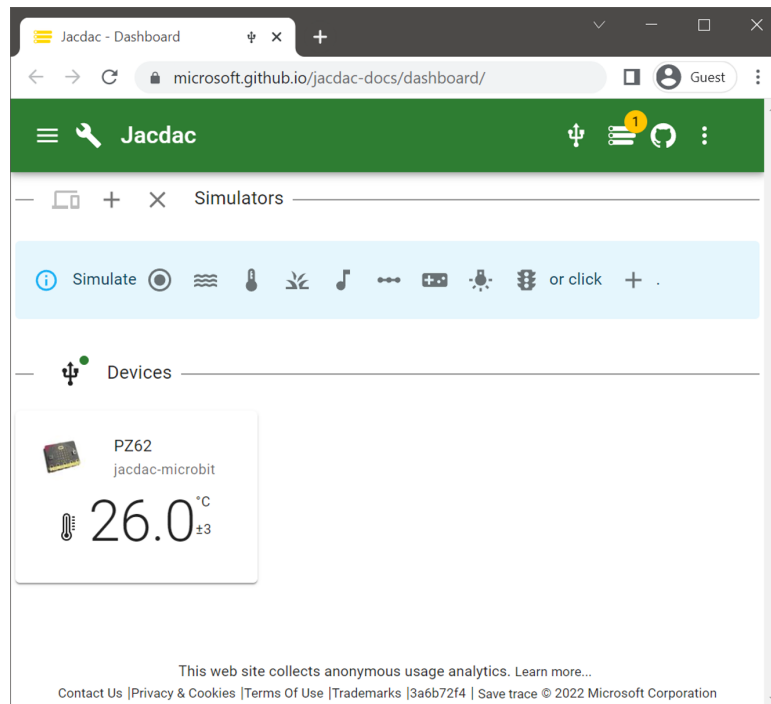


Figure 4.4: Visualizing the reading of the temperature sensor module in the Jacdac dashboard.

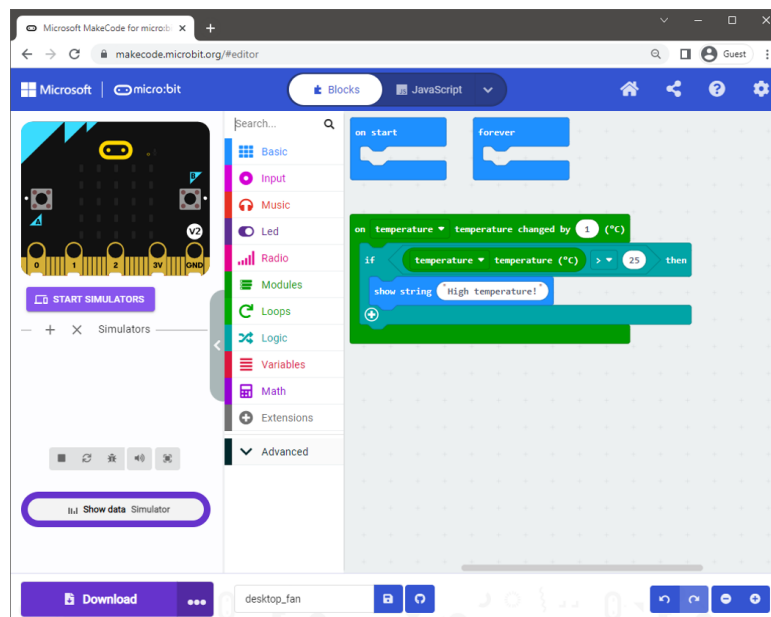


Figure 4.5: Writing the application logic on the micro:bit using MakeCode building blocks.

CircuitGlue to render a custom circuit diagram. Sam uses this diagram to connect the temperature sensor and the DC motor (using an L298N DC motor driver and 12V/3A power supply) to the micro:bit. Even though both sensors are now directly connected to the micro:bit without CircuitGlue boards, CircuitGlue ensures they are still available on the Jacdac bus as blocks in MakeCode. As such, Sam's prototype is still operational using the same application logic after taking out the CircuitGlue boards.

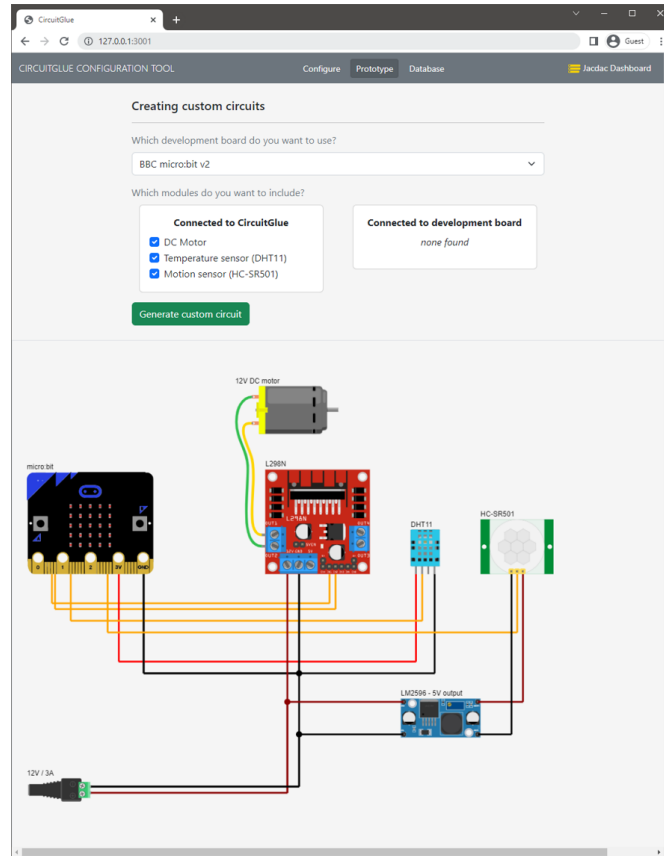


Figure 4.6: A circuit diagram generated to facilitate building a custom circuit using the DC motor, temperature sensor, and PIR motion sensor.

Sam is uncertain whether to use an ultrasonic distance sensor or a PIR motion sensor to detect presence. To compare the behavior of both sensors, Sam configures one CircuitGlue board to interface with the ultrasonic distance sensor (HC-SR04) and the other for the PIR motion sensor (HC-SR501). After plugging in both sensors, readings of both of them are visible side-by-side via the Jacdac dashboard (Figure 4.7). Sam experiments a bit and decides the PIR motion sensor is more effective for detecting presence in a room. He updates the application logic on the micro:bit to only power the DC motor when presence is detected, and the temperature exceeds 25° C. When satisfied with the prototyped system, Sam can use the *circuit diagram generator* again to receive instructions for connecting the PIR motion sensor directly to the micro:bit, making the CircuitGlue boards available for his next project.

4.3 Related Work

This work draws from and builds upon prior work on modules for electronic prototyping [Lambrichts, 2021], tools to ease breadboarding and development [Hodges, 2012], and reprogrammable integrated circuits.

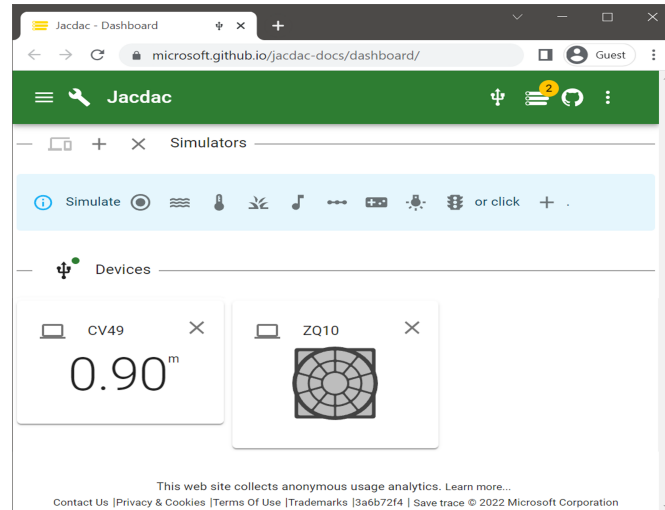


Figure 4.7: Comparing the ultrasonic distance sensor and PIR motion sensor side-to-side in the Jacdac dashboard. The Jacdac dashboard visualizes the distance sensor (left) using a numeric value, while the motion sensor (right) is represented by a graphical illustration indicating whether motion is detected or not.

4.3.1 Modules for Electronics Prototyping

Since the birth of the electronics industry, engineers have naturally sought ways to accelerate the prototyping process of electronic devices. Instead of working with individual electronic components only, many prototyping practices nowadays involve extending or interconnecting ready-made PCBs, such as breakout boards (e.g. for the BNO055 [Adafruit, 2022] and ESP8266 [Espressif, 2022a]), and development boards (including the BBC micro:bit [microbit, 2022], Arduino [Arduino, 2022], and the Raspberry Pi [Pi, 2022b]). Prototyping with such boards is usually more convenient than working directly with the ICs but still requires looking up technical details in datasheets and the use of additional voltage and protocol conversion modules because these boards are typically manufactured by different companies.

Integrated modular systems, on the other hand, offer complete sets of modules specifically designed to work together without needing any other components. Examples include .NET Gadgeteer [Hodges, 2013], SoftMod [Lambrichts, 2020], Phidgets [Greenberg, 2001], and SAM Labs [Labs, 2021]. However, adding modules or components not compatible with the integrated modular system is usually hard—their technical specifications may not be exposed, and even if they are, additional voltage and protocol converters are often necessary. In many ways, CircuitGlue brings the plug-and-play benefits of integrated modular systems to breakout boards by offering a converter that ensures compatibility between heterogeneous electronic components.

4.3.2 Tools to Ease Breadboarding and Development

As breadboards are one of the most popular tools for circuit prototyping, researchers frequently present tools to facilitate designing, building, and testing electronic pro-

totypes on breadboards. Prototyping boards such as ToastBoard [Drew, 2016] and CurrentViz [Wu, 2017] enable circuit inspection by continuously measuring and visualizing voltage and current levels throughout a breadboard circuit. In addition to visualizing the state of a breadboard, SchemaBoard [Kim, 2020b] instructs users how to connect components on a breadboard using integrated LEDs, and HeyTeddy [Kim, 2020a] guides users using text or voice conversations. Trigger-Action-Circuits [Anderson, 2017] facilitates breadboarding by generating all the necessary circuitry, firmware, and assembly instructions based on simple behavioral descriptions. In contrast, Circuito.io [Circuito, 2022] automatically generates breadboard connection diagrams based on a set of input/output modules and a microcontroller. Similar to these existing techniques, CircuitGlue facilitates building breadboard prototypes by demonstrating how a component can be connected to a development board using common off-the-shelf conversion modules instead of a CircuitGlue board.

Another important issue of breadboarding is the tangling of wires, which makes circuits fragile and error-prone. CircuitStack [Wang, 2016] contributes a new breadboard design that addresses this issue by interconnecting components via a printed circuit board instead of using jumper wires. Rather than using physical jumper wires, VirtualWire [Lee, 2021] takes a different approach and allows rows on a breadboard to be connected in software. Instead of requiring all components to be present on a breadboard, Proximo [Wu, 2019] injects software-generated signals to replace any missing components. Contributing to this line of research, CircuitGlue also avoids wired connections by allowing modules to connect directly to the CircuitGlue board.

Finally, prototypers frequently rely on a range of test and measurement equipment during the development process. Obvious examples include a huge variety of widely available power supplies, multimeters and oscilloscopes. Of particular note are low cost PC accessories such as the Bus Pirate serial communications monitor⁵, Saleae logic analyzers⁶ and Digilent’s Analog Discovery unit⁷.

4.3.3 Reprogrammable Integrated Circuits

Alphonsus et al. [Alphonsus, 2016] offer an extensive overview of various types of reprogrammable integrated circuits and their applications, including Field-Programmable Gate Arrays (FPGAs) [Romano, 2022] and programmable system-on-chips (PSoCs) [Infineon, 2022]. Over the past few years, reprogrammable integrated circuits have become more common in HCI. Scanalog [Strasnick, 2017], for example, uses a Field-Programmable Analog Array (FPAA) [Anadigm, 2022] to facilitate interactive design and debugging of analog circuits by using direct manipulation. A special type of reprogrammable integrated circuit is the crosspoint switch, which has been used in several interactive systems lately. VirtualWire [Lee, 2021], for example, uses a crosspoint switch

⁵ http://dangerousprototypes.com/docs/Bus_Pirate

⁶ <https://www.saleae.com>

⁷ <https://digilent.com/reference/test-and-measurement/analog-discovery/start>

to allow users to interconnect different rows on a breadboard in software. Similarly, VirtualComponent [Kim, 2019] uses a crosspoint switch to connect and disconnect specific component banks on a printed circuit board. CircuitGlue goes beyond software configurable connections between header pins [Lee, 2021] and peripherals [Infineon, 2022] by offering programmable voltage power delivery and conversion of digital protocols.

Several microcontrollers also embed features of reprogrammable integrated circuits. The nRF52 series [Semiconductor, 2022a], for example, implements a Programmable Peripheral Interconnect (PPI) [Semiconductor, 2022b] that allows dynamic pin mapping, configuration, and allocation of resources and enables peripherals to communicate autonomously independently of the CPU. Similarly, the RP2040 processor of the Raspberry Pi Pico [Pi, 2022a] includes a form of software programmable digital hardware called Programmable Input/Output (PIO) [Pi, 2022c]. While these approaches are very versatile, they can only be used to route low-current signals—unlike the programmable header pins of CircuitGlue, they are not suitable for powering external electronics.

4.4 Design Rationale

To guide the design of our CircuitGlue and gather early feedback on the prototyping styles that are interesting to potential user groups, we conducted informal interviews using conceptual renderings of the CircuitGlue board (Figure 4.8).

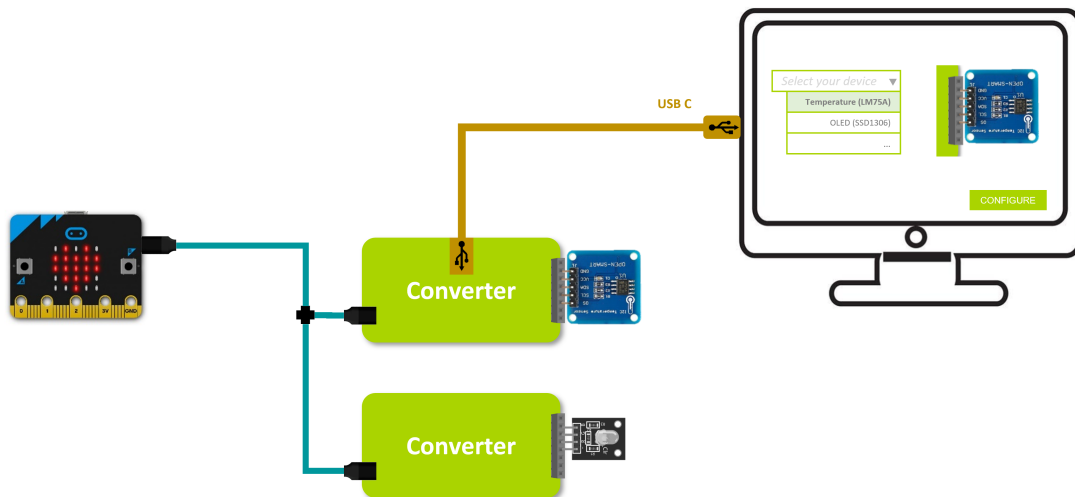


Figure 4.8: Example of the conceptual renderings used in the interviews to gather early feedback on the CircuitGlue concept.

4.4.1 Early Feedback on the CircuitGlue Concept

We conducted informal interviews with three potential users: a teacher, a maker, and an electronics engineer, all recruited from our network. Given the limited number of interviews, the results reported in this section simply offer initial feedback on CircuitGlue to gauge perceptions of the concept and do not necessarily reflect interest from the

broader community. The interviews were conducted online and lasted 30 minutes.

To correctly interpret and contextualize participants' comments, we first invited them to fill out a short questionnaire asking about their previous prototyping experiences. The teacher reported using integrated modular systems, such as .NET Gadgeteer [Hodges, 2013] frequently, whereas the maker and electronics engineer only used breakout boards or off-the-shelf electronic components.

At the start of the interview, we introduced the CircuitGlue concept as a black box for interfacing with any type of electronic component, including modules. For the teacher, this already had *"massive potential if I could hook up stuff without having to worry about the polarity"*, and the maker also saw potential in CircuitGlue for rapid prototyping. Using ten conceptual renderings, we then introduced various potential functionalities and prototyping styles (covered in detail in Section 4.8). After explaining the prototyping style, we asked participants how they valued the scenario and when it could be useful in their prototyping practices.

Both the maker and the teacher were enthusiastic about using CircuitGlue to quickly test components without needing to create complex circuits (Section 4.8.2). While the maker saw a lot of potential in using CircuitGlue to work with more complex components, such as motors and valves, the teacher's initial response was that s/he would still prefer using standard motor shields in the classroom as these shields are very cheap and familiar to him/her. The opportunity to automatically generate a custom circuit diagram, based on the knowledge of electronic components built into CircuitGlue, was especially appealing to the maker who liked *"ending up with a custom circuit without needing to do the thinking"*. Finally, we presented a scenario in which CircuitGlue was used in a similar way to a development board (Section 4.8.5). The maker, in particular, saw potential in writing custom software while still using the programmable voltage and logic level conversions provided by CircuitGlue. In contrast, the teacher appreciated the compatibility of CircuitGlue with micro:bit as s/he already used this platform in the classroom.

Throughout the interview, the electronics engineer did not find many scenarios appealing as s/he was used to looking in datasheets and did not trust libraries developed by third parties as *"they can potentially blow up modules."* The engineer, however, thought CircuitGlue would be great in educational settings.

4.4.2 Design Decisions

The CircuitGlue board exposes eight programmable header pins, of which the pin assignments are programmed in software; each can be configured to either output digital signals, read analog or digital signals, connect to ground, or deliver power, with the latter being programmable in steps of 0.1V. Our board supports digital communication signals via the programmable header pins at both 3.3V and 5V and supports analog readings up to 12V. The board is powered and can be programmed via an onboard USB-C connector. In situations where a module's pinout does not fit our single row eight

header pin configuration, such as modules that expose two rows, a small breadboard or adapter board can be used to connect to CircuitGlue. The CircuitGlue board supports components and modules that operate between 1.8V and 12V and supplies up to 3A. To allow for easy interconnection between a CircuitGlue board and Jaccad compatible devices, such as the micro:bit, our board embeds the Jaccad edge connector⁸. In addition to USB-C, the CircuitGlue board can also be programmed using Jaccad.

4.4.3 Jaccad as Bus Protocol

To allow a wide variety of electronic components and modules to intercommunicate, CircuitGlue needs to translate analog readings and digital signals, such as I2C and SPI, to a common bus protocol. This is realized using a short snippet of translation code for every electronic component, as described in Section 4.5.

CircuitGlue implements the Jaccad [Devine, 2022] communication protocol, which is built on top of Single Wire Serial (SWS), as the common bus protocol. By translating all analog and digital signals into Jaccad packets, every module or component connected to the CircuitGlue board can be recognized as a new device on the Jaccad bus. This enables seamless intercommunication between heterogeneous electronic components and modules, as demonstrated in our Walkthrough (Section 4.2).

CircuitGlue could have also used a different existing protocol, such as I2C, SPI, or a CAN bus, but supporting Jaccad on CircuitGlue further facilitates working with electronics. Firstly, Jaccad announces the components and their functionality as soon as they are connected. This allows instant interaction and experimentation with components and modules via the Jaccad dashboard³. Secondly, Jaccad introduces a software abstraction via its “services” layer. Each Jaccad service exposes a basic function, such as a button or accelerometer. This abstraction makes it easy to interface with any given type of component in exactly the same way independently of the specific hardware. For example, the ADXL345 is an accelerometer that makes readings available over I2C or SPI, while the MMA7361 accelerometer outputs analog voltages. By using the Jaccad accelerometer service, acceleration readings for both sensors use exactly the same methods. As such, one can substitute a component with a similar one from a different supplier without making changes to application logic.

4.5 Supporting New Modules

As demonstrated in the Walkthrough (Section 4.2), the CircuitGlue configuration tool allows users to simply select the electronic module they want from a list of supported modules. The CircuitGlue configuration tool then uploads the specifications for all eight programmable header pins to the CircuitGlue board and flashes the driver, referred to as “translation code”, to correctly interface with the connected module and expose its

⁸ <https://microsoft.github.io/jaccad-docs/ddk/design/electro-mechanical>

functionality on the Jaccac bus.

To add support for additional modules to CircuitGlue, the new module's pin configuration must be specified, and a translation code snippet has to be written. Writing the translation code requires implementing the respective Jaccac service layer⁹, such as the accelerometer service for an ADXL345 accelerometer. The translation code must call the initialization function of the appropriate Jaccac service and pass function pointers to custom-written `initialization()`, `update()`, and `read()` functions. In these three functions, custom code can be written, or calls can be made to an existing library, such as an Arduino library. The example in Figure 4.9 shows translation code for supporting the Jaccac temperature service using a DHT11 temperature sensor. More advanced components sometimes require additional functions to be implemented, as specified in the Jaccac documentation¹⁰. When a module offers multiple functionalities, multiple services should be implemented.

```
1  ~ #include "Arduino.h"
2  ~ #include "DHT.h"
3
4
5  DHT dht(1, DHT11);
6
7  uint32_t last_sample = 0;
8  uint32_t last_sample_interval = 0;
9
10 ~ void initializer(void) {
11     // start the temperature sensor
12     dht.begin();
13 }
14
15 ~ void updater(void) {
16     // sample the temperature every two seconds
17     if (jd_should_sample_delay(&last_sample_interval, 2000))
18         last_sample = JD_FLOAT_TO_I22_10(dht.readTemperature());
19 }
20
21 ~ uint32_t get_temperature(void) {
22     // returns the last sample
23     return last_sample;
24 }
25
26
27 ~ const env_sensor_api_t custom_temperature_api = {
28     .init = initializer,
29     .process = updater,
30     .get_reading = get_temperature,
31 };
32
33 ~ void setup(void) {
34     // initialize the temperature service providing our custom api
35     temperature_init(&custom_temperature_api);
36 }
37
```

Figure 4.9: Example of the translation code written to support the DHT11 temperature sensor module.

⁹ <https://microsoft.github.io/jaccac-docs/services/>

¹⁰ <https://microsoft.github.io/jaccac-docs/ddk/services/#implementing-service-firmware>

Although writing translation code requires more technical expertise than simply using CircuitGlue, leveraging the Jacdac programming paradigm and potentially re-using Arduino driver code makes it relatively straightforward. We also note that it's a one-time effort—we envision users sharing these configurations and drivers via community platforms in the future. Translation code is platform agnostic; the same code can be used to control a module via the Jacdac protocol even if a CircuitGlue board is not used. This is essential to ensure prototypes remain functional when modules are connected to a development board with custom circuitry with the help of the CircuitGlue *circuit diagram generator* feature.

In addition to writing translation code to expose a new module on the Jacdac bus, its pinout must be specified. The CircuitGlue configuration tool streamlines the process of supporting new modules by offering an interface for specifying the pinout and required voltages of a new module (Figure 4.10). The configuration tool also embeds additional intelligence that encodes and checks various heuristics relating to common protocols. For example, when only one of the two pins required for the I2C protocol is specified, the interface provides a warning.

The screenshot shows the 'CIRCUITGLUE CONFIGURATION TOOL' interface with tabs for 'Configure', 'Prototype', and 'Database'. The 'Database' tab is active, showing a form titled 'Adding new modules'. The form includes the following fields and options:

- Generic name:** Text input with placeholder 'e.g. Temperature sensor'.
- Module type:** Text input with placeholder 'e.g. DHT11'.
- Picture:** A button labeled 'Choose File' and a status 'No file chosen'.
- Operating voltage:** Text input with value '3,3' and a unit dropdown set to 'V'.
- Signal voltage:** Dropdown menu with value '3.3V'.
- Module pinout:** A series of dropdown menus for pins 0 through 7.
 - Pin 0:** Ground
 - Pin 1:** Voltage
 - Pin 2:** Digital (GPIO or PWM). Below this are sub-fields: 'Name' (digital_2), 'Controller' (Microcontroller), and 'Input range' (0 - 3.3V).
 - Pin 3:** Analog input. Below this are sub-fields: 'Name' (analog_3) and 'Input range' (0 - 3.3V).
 - Pin 4:** Communication protocol. Below this are sub-fields: 'Protocol' (SCK) and 'Pin' (SCK).
 - Pin 5:** Not connected
 - Pin 6:** Not connected
 - Pin 7:** Not connected

Figure 4.10: Adding a new module to the database using the configuration tool.

4.6 CircuitGlue Hardware Design

To structure a discussion of the hardware design of the CircuitGlue board, we split it into three sections (Figure 4.11): (1) a System-on-Chip (SoC) controlling and monitoring the board, (2) voltage regulation and power delivery, and (3) hardware components for configuring each of the eight programmable header pins.

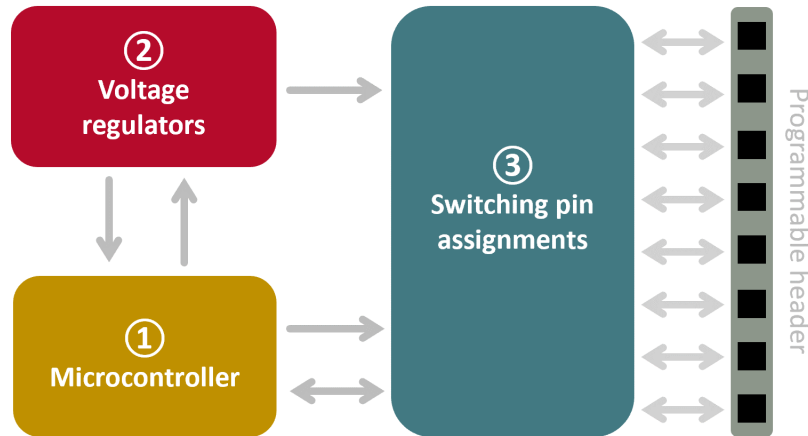


Figure 4.11: The design of the CircuitGlue board with (1) a System-on-Chip (SoC) controlling and monitoring the board, (2) voltage regulation and power delivery, and (3) hardware components for switching the assignment of programmable pins.

4.6.1 System-on-Chip

We decided to use a microcontroller SoC to control the CircuitGlue board instead of configurable integrated logic chips, such as programmable system-on-chip (PSoC) and Field-Programmable Gate Array (FPGA), as many existing software libraries for prototyping with electronics, including Jaccadac, are available for microcontrollers. The CircuitGlue board is built around the nRF52840; this SoC allows the dynamic assignment of peripherals to pins and, therefore, does not need additional hardware components, such as crosspoint switches, to re-map the functionality of CircuitGlue’s programmable header pins. To reduce the number of electronic components on the board, we decided to use Raytac’s nRF52840 module. This module embeds the nRF52840 SoC and complementary components, such as capacitors and an antenna.

4.6.2 Regulating Power

The CircuitGlue board is powered through the onboard USB-C connector using Power Delivery (PD) and accepts voltages between 5V and 15V. A USB-C PD controller on the CircuitGlue board allows voltage negotiations with USB-C PD compatible power supplies and is configured to always deliver the highest available voltage (with a maximum of 15V) to the CircuitGlue board. When Power Delivery is unavailable, the

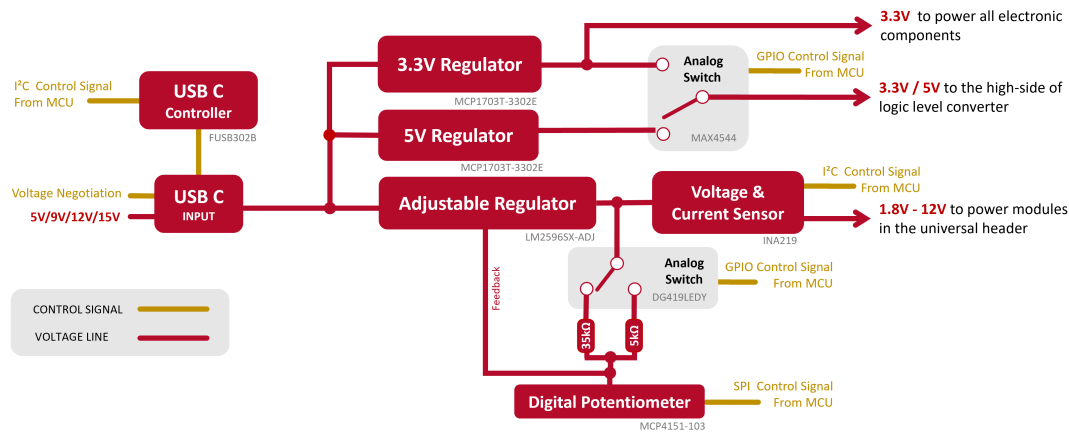


Figure 4.12: Block diagram with all regulators responsible for providing the three voltage levels used by the CircuitGlue board.

CircuitGlue board can be powered either from the 5V USB connection or by connecting an external power supply to the board. Using the input voltage, the CircuitGlue board internally regulates three voltage levels for its operation:

- **Board voltage:** A voltage of 3.3V to power all electronic components on the CircuitGlue board. As shown in Figure 4.12, this voltage is supplied by a low-dropout 3.3V regulator.
- **Signal voltage:** The voltage used by the logic level converters (described in Section 4.6.3) to translate digital communication signals from the board voltage to the voltage required for communicating with the electronic module plugged into the programmable header. As most electronic modules require either 3.3V or 5V communication signals, the signal voltage can toggle between these two voltages using a digital switch, as shown in Figure 4.12.
- **Programmable voltage:** The voltage used to power modules or components plugged into the programmable header. To supply a large selection of electronic modules, this voltage line is programmable from 1.8 to 12V in steps of 0.1V via an adjustable voltage regulator controlled by a 10k Ω digital potentiometer driven by the nRF52840 SoC. This can deliver a current up to 3A. See Figure 4.12 for details. While the digital potentiometer has 256 steps, achieving the required voltage range of 1.8V to 12V in steps of 0.1V is impossible due to the logarithmic scale in the output of the voltage divider. Therefore, we dynamically switch between two ranges using an analog switch. The graph shown in Figure 4.13 shows that swapping between our two ranges (by switching between a resistance of 35k Ω and 5k Ω) offers the full voltage range with the desired accuracy. As shown in Figure 4.12, a voltage and current sensor is added to measure the actual voltage and current usage of a module connected to the programmable header. This information later verifies the programmed voltage and provides basic protection against shorts and faults (Section 4.7).

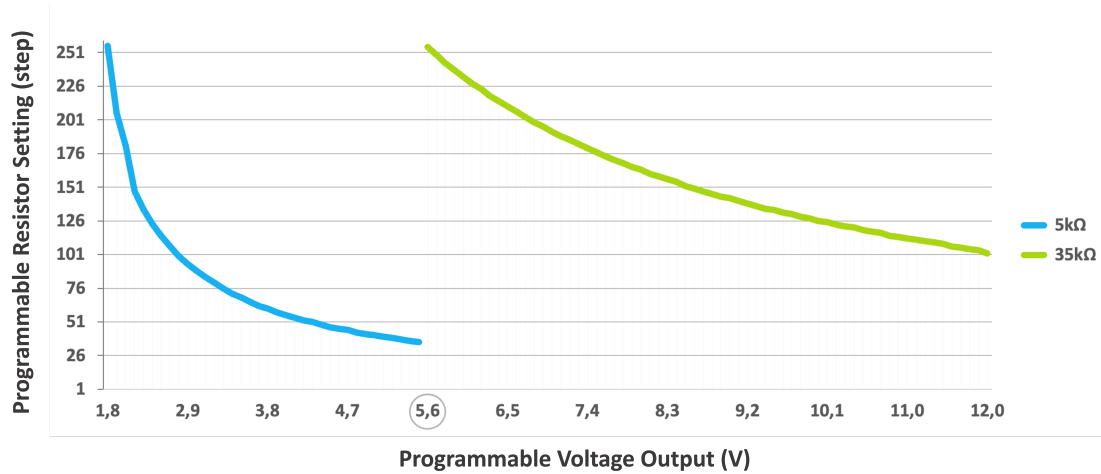


Figure 4.13: The required setting in the digital potentiometer based on the requested output voltage and selected top resistor.

4.6.3 Changing the Assignment of a Programmable Header Pin

As conceptualized in Figure 4.11, the CircuitGlue SoC controls everything necessary to correctly assign the eight programmable header pins on the CircuitGlue board. Figure 4.14 shows this in more detail. The dashed blocks in this figure represent the electronic circuit for setting the assignment of exactly one programmable header pin; eight such identical blocks are present on the CircuitGlue board. Each of these embeds the circuitry to either configure the programmable header pin (1) as a General Purpose Input/Output (GPIO) pin, (2) as an analog input pin, (3) as a connection to ground, or (4) as a programmable voltage supply.

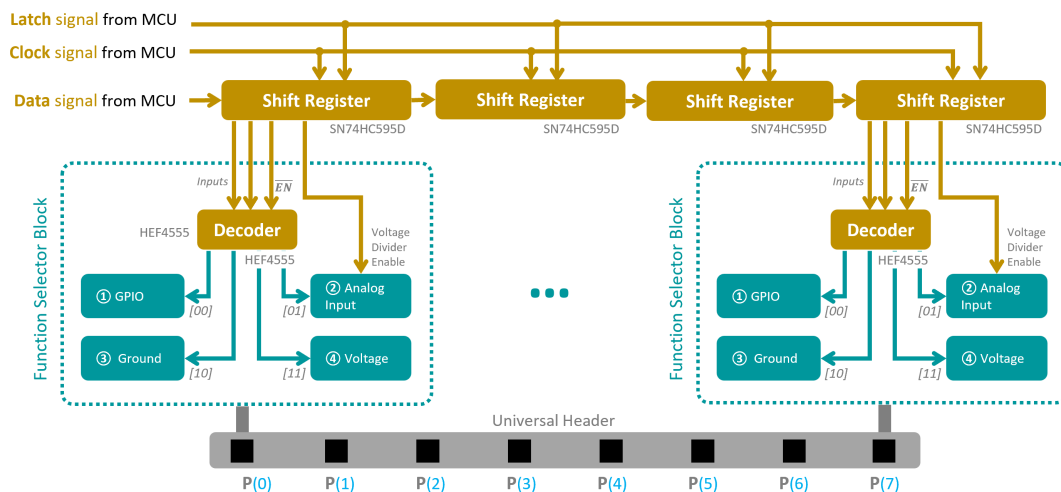


Figure 4.14: Block diagram of all components required for changing the assignments of the programmable header pins of the CircuitGlue board.

Shift registers connected in series control all of the eight identical dashed blocks and are driven by the nRF52840 SoC. For each dashed block, two shift register output pins

activate the appropriate circuit in a block by using a 1-of-4 decoder to ensure only one of the four circuits is active at any time. As shown in Figure 4.14, a third shift register output is used to disable the decoder when the programmable header pin needs to be in a high impedance state. As we will further explain below, a fourth shift register output is used to activate a voltage divider, which is part of the circuit for configuring the programmable header pin as an analog input pin.

Programmable Header Pin as GPIO

As shown in Figure 4.15, an analog switch controlled by the decoder connects each of the programmable header pins of the CircuitGlue board to a high-speed GPIO pin on the nRF52840 SoC. While all components on the CircuitGlue board work with 3.3V signals, a bidirectional logic level converter allows 5V signaling, as dictated by the plugged-in module (as discussed in Section 4.6.2).

The logic level converter supports both push-pull and open-drain applications and can achieve speeds up to 24Mbps and 2Mbps, respectively, which is sufficient for most popular digital communication protocols. Two methods are built-in to protect the board when users accidentally apply a higher voltage on one of the programmable header pins. First, a comparator and AND gate instantly overwrite the signal from the decoder to disconnect the analog switch when the voltage applied to the programmable header pin exceeds the signal voltage. A non-inverting summing amplifier slightly increases the signal voltage to prevent undesired cut-offs of the logic signal. Second, a clamping diode, shown in Figure 4.15, further prevents voltages higher than 3.3V from passing to the nRF52840 SoC.

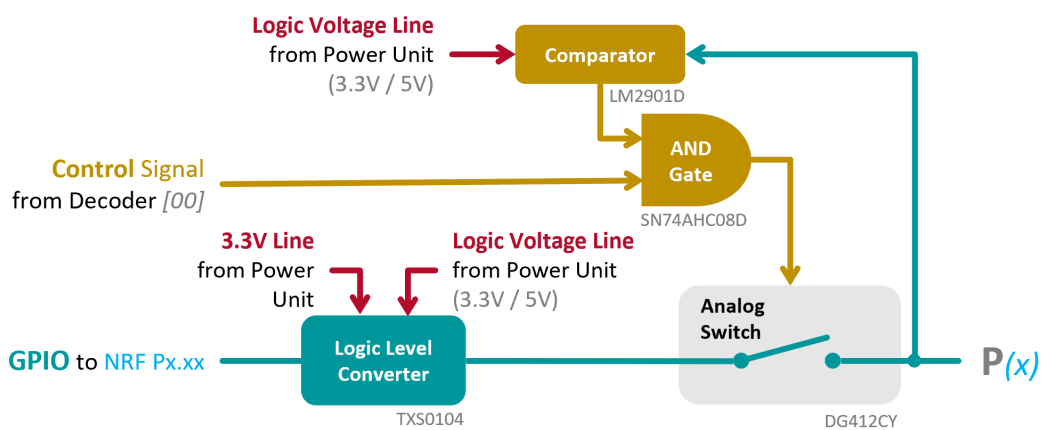


Figure 4.15: Circuit for connecting the programmable header pin to a digital high-speed GPIO pin on the nRF52840 SoC.

Programmable Header Pin as Analog Input

As shown in Figure 4.16, an analog switch controlled by the decoder connects the programmable header pin of the CircuitGlue board to an analog input pin of the nRF52840 SoC. The nRF52840 embeds a 12-bit analog-to-digital converter and supports voltages up to 3.3V. Analog input readings up to 12V are supported at the expense of the accuracy using a voltage divider controlled by the nRF52840 SoC, as shown in Figure 4.16. This allows CircuitGlue to make resistance measurements to support resistive sensors such as photoresistors. A comparator and OR gate automatically turn on the voltage divider when the voltage supplied on the programmable header pin exceeds 3.3V, and a non-inverting summing amplifier circuit allows small voltage peaks to surpass the threshold. Finally, a feedback signal informs the nRF52840 SoC when the voltage divider is activated so the actual voltage on the programmable header pin can be calculated.

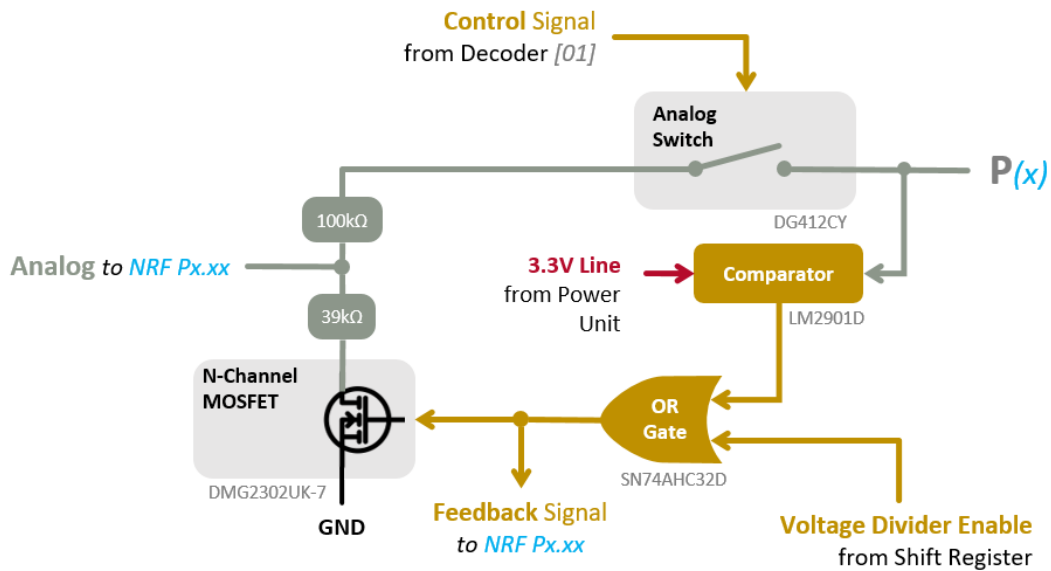


Figure 4.16: Circuit for connecting the programmable header pin to an analog pin on the nRF52840 SoC.

Programmable Header Pin as Ground

As shown in Figure 4.17, an N-channel MOSFET controlled by the decoder connects a programmable header pin to ground. While using an analog switch would entirely disconnect a line, MOSFETs are more suitable in this part of the circuit as they support higher currents and are available in smaller package sizes. CircuitGlue uses MOSFETs that support up to 2.8A of continuous current.

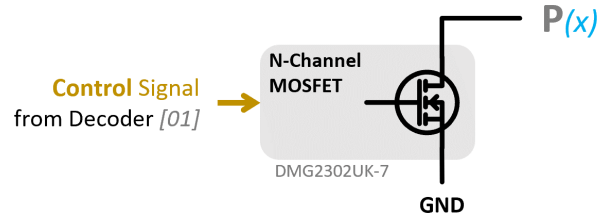


Figure 4.17: Circuit for connecting the programmable header pin to ground.

Programmable Header Pin as Power

Similar to the connection to ground, the P-channel MOSFET in Figure 4.18 is used for connecting a programmable header pin to the programmable voltage line (Figure 4.12). A signal translator consisting of an N-channel MOSFET translates the signal from the decoder as a P-channel MOSFET requires the control voltage to be in the same range as the source voltage (1.8V-12V). A diode prevents reverse current through the P-channel MOSFET, avoiding issues when the programmable header pin is used for digital or analog signals of a voltage higher than the programmable voltage.

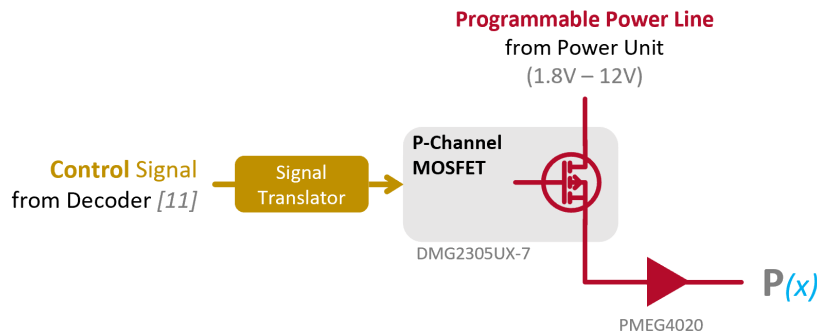


Figure 4.18: Circuit for connecting the programmable header pin to the programmable voltage level.

4.6.4 Circuit Board Design and Manufacturing

To realize the CircuitGlue board, we combined all building blocks on a single, four-layer PCB measuring 125 by 50mm. The final iteration uses 229 electronic components on both sides and for this version of our prototype, components were manually placed and soldered using a reflow oven. We carefully chose commodity components wherever possible to minimize cost and maximize supply chain flexibility. When manufacturing ten prototypes, the cost of the CircuitGlue PCB is around \$8 (from JLCPCB¹¹) plus \$49 for all components (from OctoPart¹²). This drops to around \$0.50 per PCB and \$26 for component quantities of 1k pieces. Given the predominance of commodity components, the component cost will likely be significantly less on the Chinese market.

¹¹<https://jlcpcb.com>

¹²<https://octopart.com>

4.7 CircuitGlue Software Architecture

4.7.1 CircuitGlue Firmware

The CircuitGlue firmware runs on the nRF52840 SoC and drives all functionalities of the board. If an internal error occurs, the CircuitGlue board disconnects its programmable header pins to avoid damaging any electronics. It furthermore notifies the user by blinking two LEDs on the CircuitGlue board. Such errors include internal components that are not responding as expected, or when the measured voltage or current is outside the required range due to a short circuit or component failure.

The CircuitGlue firmware is written in the C programming language, as it offers easy access to all peripherals and registers of the nRF52840 SoC. We built on top of several existing software libraries: the standalone nrfx drivers¹³ and libraries provided by the nRF5 SDK¹⁴ ease setting up peripherals, such as timers and communication protocols; and the existing platform-agnostic implementation of the Jaccac protocol¹⁵ and is extended with a platform-specific implementation of peripherals for the nRF52 family (e.g., half-duplex RS232, I2C, SPI, GPIOs, ...) that we wrote ourselves. The CircuitGlue firmware is available on GitHub¹⁶.

The firmware is compiled using the arm-none-eabi compiler¹⁷. To flash the compiled hex file, we use Device Firmware Upgrade (DFU) over USB. To generate a DFU package from the compiled hex file, we use the “nrfutil pkg generate” tool and flash the DFU package using “nrfutil dfu USB-serial”. To enable DFU on the nRF52840 SoC, we flashed the precompiled bootloader, which is provided by Nordic Semiconductor. Alternatively, the hex file can be flashed using the “nrfjprog” command or by connecting an external programmer to the Serial Wire Debug (SWD) pins on the CircuitGlue board.

4.7.2 CircuitGlue Configuration Tool

The CircuitGlue configuration tool is implemented as a web interface. The configuration tool uses Jaccac¹⁸ over either USB CDC or WebUSB¹⁹ to communicate with connected CircuitGlue boards. A JSON database contains all technical specifications, such as required voltages and pinouts, and the UF2 firmware file for each supported module. When users configure a module, the configuration tool loads its specifications from the database and sends it to the CircuitGlue board. After that, the firmware containing the translation code is flashed to the CircuitGlue board.

Before sending the specifications and firmware, the CircuitGlue configuration tool

¹³<https://github.com/NordicSemiconductor/nrfx>

¹⁴<https://www.nordicsemi.com/Products/Development-software/nrf5-sdk>

¹⁵<https://github.com/microsoft/jaccac-c>

¹⁶<https://github.com/MannuLambrichts/CircuitGlue>

¹⁷<https://developer.arm.com/downloads/-/gnu-rm>

¹⁸<https://github.com/microsoft/jaccac-ts>

¹⁹<https://wicg.github.io/webusb/>

automatically enables the CircuitGlue DFU bootloader on the CircuitGlue board. This bootloader allows for convenient firmware updates over Jaccac and disables all programmable pins to protect the connected module from previous configurations.

After flashing, the firmware sets the signal and programmable voltage (Section 4.6.2) and waits until the programmable voltage reaches its configured setting. Next, the firmware programs each pin assignment in the shift registers using four bits per programmable pin, as described in Section 4.6.3. After activating all shift registers, the function of each programmable header pin is assigned. The firmware then initializes all necessary peripherals for the digital communication protocols. Finally, the firmware instantiates the translation code necessary for translating the module's functionality to Jaccac services, which makes the module available on the Jaccac bus.

4.7.3 Circuit Diagram Generator

As demonstrated in the Walkthrough (Section 4.2), once a module is working as desired, the CircuitGlue configuration tool offers a *circuit diagram generator* feature to assist in connecting it directly to a development board. To support this feature, we developed a pin mapping algorithm. For every pin on every module, our algorithm first assigns a compatible pin on the development board without considering specific voltages. For the final pin assignment, the algorithm aims to maximize the number of modules that can be connected, similar to the routing strategies of PaperPulse [Ramakers, 2015]. For example, the SCL and SDA pins used for I2C will only be used for digital signals when all other digital pins are already in use. The algorithm then compares the voltages available on the development board with the voltages required by the modules and adds logic-level converters and an external power supply when needed. This power supply is selected based on the highest voltage needed, and DC-DC converters are additionally added to step down to other voltage levels.

Some electronic components, such as a DC motor, require additional driver circuits. To support this, we extended our JSON database, detailing the pinout and operating voltages for all modules, with an optional field that details the required driver module needed when converting from CircuitGlue to a custom circuit diagram. The *circuit diagram generator* uses this field to initiate additional driver components, such as the L298N DC motor driver.

In order to reuse application logic after converting to a custom circuit diagram, modules still need to be operational over Jaccac, even though they are directly connected via digital, analog, I2C, or SPI peripherals. To realize this, the micro:bit development board runs a custom version of the CircuitGlue firmware, which previously ran on the CircuitGlue Board, and the CircuitGlue configuration tool activates the same translation code on the micro:bit.

4.8 Prototyping Styles and Benefits

The unique features of CircuitGlue facilitate prototyping with electronics and thus enable several novel prototyping styles. Below, we discuss these novel opportunities.

4.8.1 Understanding, Testing and Comparing Modules

It is often challenging for novices in electronics to select appropriate electronic components when prototyping an interactive system [Świerczyński, 2014]. Example decisions include whether to use an accelerometer or gyroscope, ultrasonic distance sensor or infrared distance sensor or what type of temperature sensor is more suitable. To make an informed decision, novices traditionally consult online articles, books, or datasheets; this can be time-consuming, especially as many such resources are not written for novices [Świerczyński, 2014]. A complementary approach that may be useful in some cases is to simulate certain electronic components or sub-circuits, although this is rarely a substitute for experimenting with and verifying the operation of real components, especially in interactive systems where physical behavior, such as measuring acceleration or movement, often impacts the overall user experience. Using CircuitGlue, electronic components are instantly operational, and their basic operations can be tested immediately via the Jacdac dashboard—which renders real-time digital twins of all components on the Jacdac bus—without writing application logic. As such, novices build an intuitive understanding of components and can decide between components by observing differences in output (Figure 4.19). Testing components with CircuitGlue can, therefore, complement existing resources for understanding electronics because these practical tests confirm basic knowledge or help fill in knowledge gaps.

The Jacdac dashboard is primarily designed to support exploration and fault-finding; a self-contained interactive system requires custom application logic to define the desired behavior based on its various components. This application logic can be written to run on a PC, for example, in Python, .NET or a web app, or on an embedded microcontroller; in all cases, apps are typically simple compositions of the Jacdac services provided by the relevant modules, simplifying application development. This alleviates many compatibility issues and reduces the complexity of the testing process. We refer the reader to Jacdac [Devine, 2022] for more details.

4.8.2 Rapid Prototyping with Heterogeneous Modules

In addition to quickly getting a single module up and running, multiple CircuitGlue boards are easily interconnected using Jacdac. As such, interactive systems consisting of components working with different protocols and voltages (Figure 4.20) are realized in a few minutes. Small electronic modules, requiring only a few pins and the same operating voltage, can even connect to a single CircuitGlue board. In many ways, this prototyping style combines the ease of use of **TYPE 3** integrated modular systems with the versatility and flexibility of **TYPE 2** breakout boards and modules.

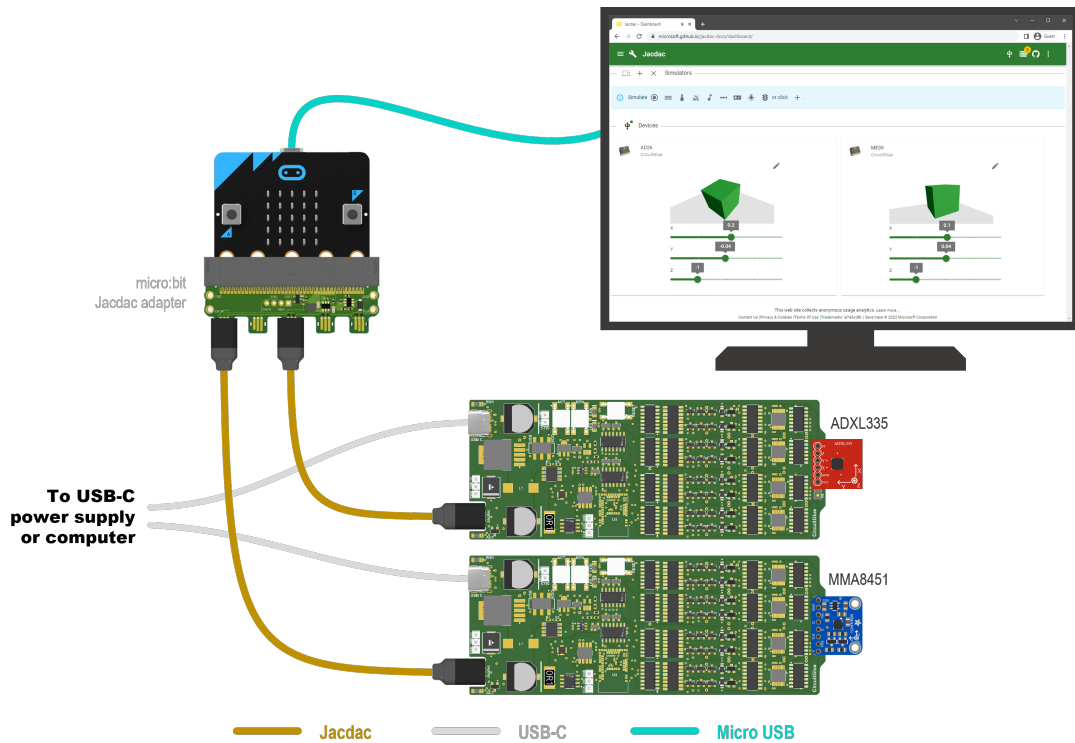


Figure 4.19: Comparing two different types of accelerometers in the Jacdac dashboard by using two connected CircuitGlue boards.

The *circuit diagram generator*, covered in the next section, helps with reusing CircuitGlue boards within a prototyping process. Application logic, defining the behavior of all components in the sensor system, is easy to write on Jacdac-compatible development boards, such as the micro:bit [microbit, 2022], Raspberry Pi [Pi, 2022b], or ESP32 [Espressif, 2022b]. When using MakeCode, which offers seamless support for Jacdac, all components on the Jacdac bus are available as blocks in the programming environment. When using other programming languages, such as .NET, Python, or JavaScript/TypeScript, modules become available by instantiating the Jacdac client service corresponding to the module. A client service provides the interface for interacting with the Jacdac service used by a module.

While it's theoretically possible to use the *circuit diagram generator* to design and implement electronic prototypes without using CircuitGlue as an intermediate evaluation step, this approach is slower, more fiddly, and may, therefore, not be suitable for those who are less experienced. By using CircuitGlue to test and verify individual components, users can gain a practical understanding of how different components work together and ensure compatibility between them before integrating them into the prototype. This not only saves time and reduces the risk of errors, but we believe it will also help users build an intuitive understanding of electronics, which would be beneficial for future projects.

Even users who only prototype with electronics occasionally often accumulate quite a

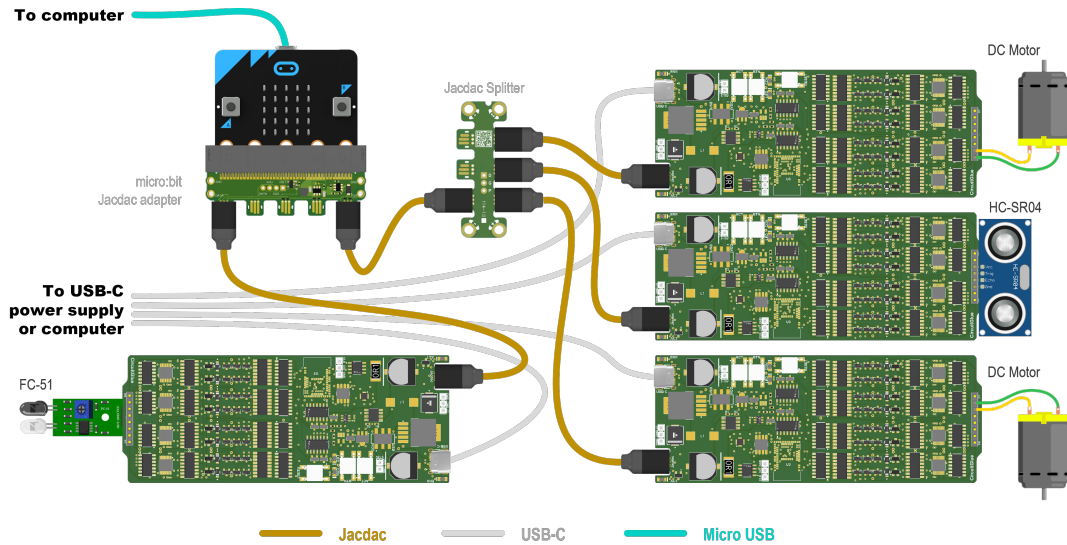


Figure 4.20: Building a prototype using multiple CircuitGlue boards.

few electronic components and modules; when creating a new prototype, it is often more convenient to buy new ones because existing ones might not be compatible with each other or have become deprecated, and thus harder to get operational again. CircuitGlue significantly helps with getting such components operational and ensures compatibility with different generations of components. Therefore, we believe CircuitGlue can help reduce the ecological footprint of electronics prototyping.

4.8.3 Facilitate Breadboarding

When a component is powered by a CircuitGlue board, the *circuit diagram generator* offers a visual guide on connecting all components on the Jacdac bus directly to a development board without using CircuitGlue boards (Figure 4.21). During this process, CircuitGlue ensures all components are still available on the Jacdac bus and compatible with the application logic, even when they are connected via analog pins or alternative digital protocols, such as I2C or SPI. This allows for a gradual transition from CircuitGlue boards to custom breadboard designs and is especially useful when running out of CircuitGlue boards or when starting with breadboard circuit prototyping. We see this as a top-down style to prototyping electronics as components are first operational using CircuitGlue before revealing lower-level wiring details to turn it into a more traditional breadboard prototype.

4.8.4 Use Third Party Modules with Jacdac Ecosystem

Instead of prototyping an entire system with CircuitGlue, users of the Jacdac modular system²⁰ can occasionally turn to CircuitGlue to make third party modules compatible

²⁰<https://microsoft.github.io/jacdac-docs/devices/kittenbot/jacdacstarterkitwithjacdaptv2v10/>

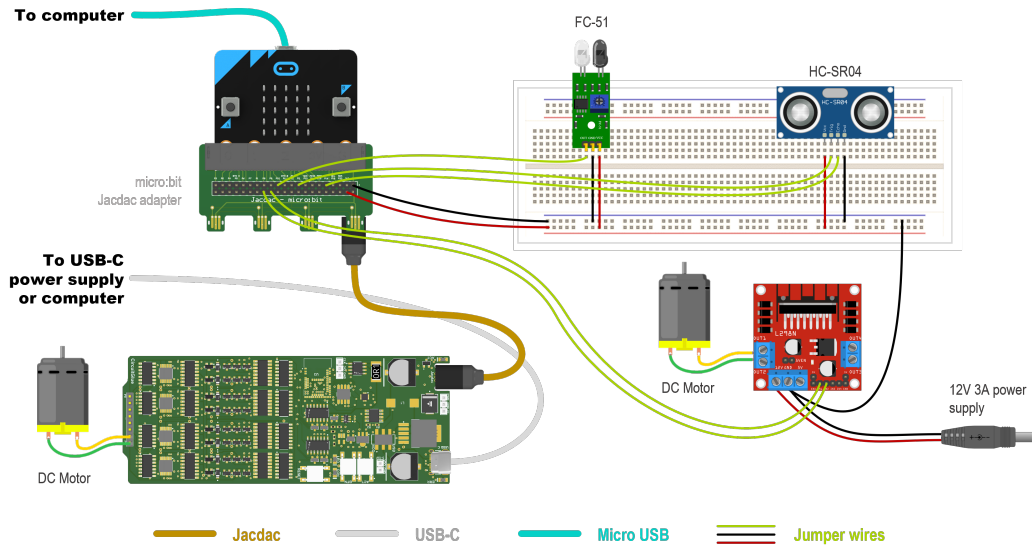


Figure 4.21: Building a prototype using a single CircuitGlue board in combination with the circuit diagram generator to facilitate building the breadboard circuits.

with this ecosystem. Figure 4.22 demonstrates such a setup in which a Jacdac button, RGB LED, and temperature sensor interconnect with the SSD1306 display module, currently not part of the Jacdac eco-system, via a CircuitGlue board.

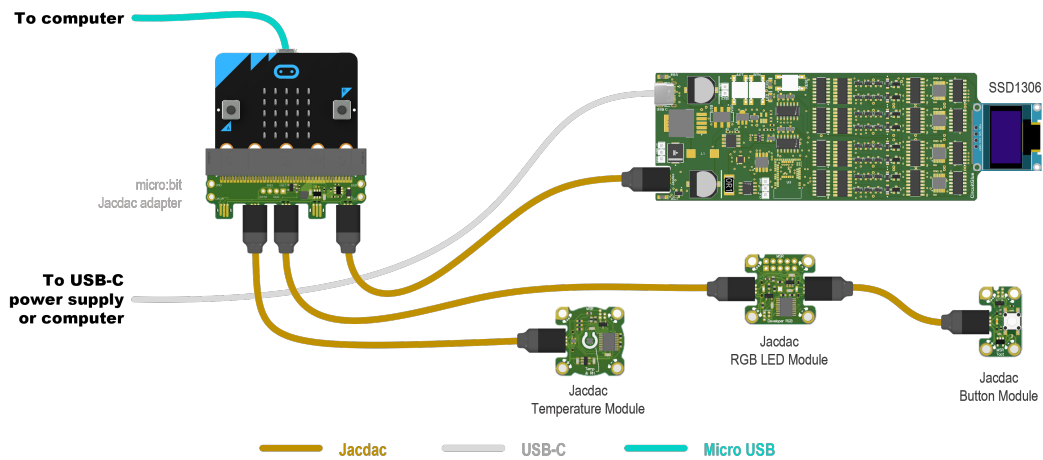


Figure 4.22: CircuitGlue used to extend the Jacdac ecosystem with new modules.

4.8.5 Advanced Use

Users who are more knowledgeable about electronics and embedded programming can use only parts of the CircuitGlue system in their prototyping practices. For example, instead of using a separate development board, such as the micro:bit, for running application logic, one of the CircuitGlue boards can run application logic on its SoC. When using a CircuitGlue board as such, users have to manually modify the CircuitGlue firmware to integrate their application logic and compile and flash it to the CircuitGlue

board. Future research can investigate how to make such features also available to novices.

To facilitate writing application logic directly into the CircuitGlue firmware, we provide a basic template using Arduino’s “setup” and “loop” constructs. In the future, we plan on adding support for the Arduino programming language to allow the use of Arduino libraries, similar to PlatformIO²¹. To enable or disable features provided by the CircuitGlue firmware, such as programmable voltages or translation to Jcdac, users can modify a header file containing compile-time configuration options.

4.9 Technical Benchmark

In this section, we report on CircuitGlue’s throughput, latency, analog reading accuracy, output voltage accuracy, and PWM signal characteristics to help the reader understand the capabilities and characteristics of CircuitGlue and its impact when used in a circuit design. All characteristics, except throughput and PWM characteristics, were benchmarked using the Saleae Logic Pro 8 logic analyzer and averaged over ten trials for consistency. The circuit design for each of the programmable header pins is identical, so the reported characteristics below are the same for every pin. We present six separate requirements; if a module or component fits all these requirements, it is compatible with CircuitGlue.

1. **Voltage range:** CircuitGlue supports a programmable voltage range of 1.8 to 12V with 0.1V resolution. The breakout board or component should operate within this voltage range to be compatible with CircuitGlue. We benchmarked the programmable power supply’s output voltage when the programmable pin is configured as Power. Figure 4.23D compares the requested voltages with the voltage measured at the programmable header pins. The results demonstrate that CircuitGlue can accurately output a requested voltage within a few millivolts. As the voltage is programmable from 1.8 to 12V in steps of 0.1V (Section 4.6.2), CircuitGlue is compatible with most common components and modules.
2. **Current rating:** CircuitGlue supports currents up to 3 amps. Breakout boards or components should require, at most, this much current to be compatible with CircuitGlue.
3. **Digital communication interfaces:** Similar to other popular development boards such as Arduino [Arduino, 2022] and micro:bit [microbit, 2022], CircuitGlue supports common digital communication protocols such as I2C, SPI, and UART, making it easy to interface with a wide variety of breakout boards and modules. The microcontroller and digital programmable header pin are connected via a logic-level converter and an analog switch. The parasitic capacitance of the switch is up to 35pF when ‘closed’, but more significantly, data throughput will be limited by the maximum operating speed of the TXS0104 logic level converter, reported in

²¹<https://platformio.org/>

the datasheet as 24Mbps when used as push-pull and 2Mbps when open-drain (see also Section 4.6.3).

To verify the latency introduced by CircuitGlue in practice, we benchmarked the delay between pulling a programmable header pin high and reading the high voltage at the microcontroller. The test results are shown in Figure 4.23A. As the latency is only a handful of nanoseconds, CircuitGlue will not noticeably impact the overall performance of most prototypes.

The DC input and output impedance and leakage current during digital communications are dominated by the characteristics of the CircuitGlue microcontroller.

4. **Analog communication interfaces:** CircuitGlue supports modules that communicate over analog signals of up to 12V. We measured the accuracy of analog readings by configuring an analog channel of our logic analyzer and connected it to a CircuitGlue programmable header pin configured as an analog input. We then applied a voltage to this pin and compared measurements from the logic analyzer with those measured by the CircuitGlue microcontroller. We conducted these measurements twice, once with our voltage divider enabled (allowing input between 0 and 12V) and once with it disabled (allowing input between 0 and 3.3V). Results are shown in Figure 4.23B-C. We note that readings differ only by a few millivolts.

In addition to analog voltages, CircuitGlue also supports modules that use a variable resistance as output, such as a temperature or light sensor. As described in Section 4.6.3, CircuitGlue has an internal pull-down resistor that can be used to measure the resistance of a sensor on the module.

As with digital communications, the DC input and output impedance and leakage current during analog communications are dominated by the characteristics of the CircuitGlue microcontroller.

5. **Pulse width modulation:** CircuitGlue can generate a Pulse-Width Modulation (PWM) signal in two ways: through a microcontroller-generated PWM signal or by rapidly switching between the Power and Ground functions of a CircuitGlue programmable pin. The best method to use depends on the requirements of the component or module being driven. Microcontroller-generated PWM signals are generally preferred for their flexibility and precision, while Power and Ground switching is used when non-standard voltages or higher currents are required. For example, an Electronic Speed Controller (ESC) module will work well with a microcontroller-generated PWM signal to set the speed of a brushless motor, while RGB LEDs and DC motors use the Power and Ground functions as they require different voltages and/or currents.

When a microcontroller-generated PWM signal is used, the frequency and duty cycle are directly controlled by the nRF52840 microcontroller. CircuitGlue supports PWM frequencies ranging up to 16MHz, and the duty cycle can be adjusted with a resolution of 8 bits. As the logic level converter (Section 4.6.3) can change its state in less than 10ns (5ns \pm 3ns), it doesn't limit the frequency of the PWM signal. On the

other hand, if the Power and Ground functions are used to control the PWM signal, the frequency is limited by the characteristics of the electronic components used to switch Ground and Power (Section 4.6.3 and Section 4.6.3). According to the datasheets of the relevant MOSFETs, shift register, and decoder ICs (Section 4.6.3), switching from Ground to Power takes around 40ns, while switching from Power to Ground takes 120nS. To provide adequate time for charging and discharging, we limit the PWM frequency to 1MHz.

6. **Physical interface:** As described in Section 4.4.2, CircuitGlue uses a physical interface consisting of a single row of eight header pins that most common breakout boards can directly plug into. However, there may be situations where a module's pinout does not match this configuration, such as modules that expose two rows of pins or have a different connector style. In such cases, an adapter board or a small breadboard can be used to connect the module to CircuitGlue.

(A) GPIO latency			
logic level	latency	SD (n=10)	unit
3.3V	5,2	2,71	ns
5V	5	2,86	ns

(B) Analog accuracy (0-12V)			
input	measured	SD (n=10)	unit
2,964	2,975	0,009	V
4,989	5,037	0,008	V
12,26	12,375	0,016	V

(C) Analog accuracy (0-3.3V)			
input	measured	SD (n=10)	unit
1,789	1,781	0,005	V
2,959	2,969	0,003	V

(D) VCC accuracy			
requested	output	SD (n=10)	unit
2,000	2,002	0,009	V
3,000	2,995	0,014	V
4,000	3,996	0,009	V
5,000	4,999	0,020	V
6,000	5,995	0,022	V
7,000	7,014	0,030	V
8,000	7,996	0,011	V
9,000	8,998	0,014	V
10,000	10,004	0,030	V
11,000	10,998	0,025	V
12,000	11,992	0,013	V

Figure 4.23: Results of the technical evaluation of CircuitGlue.

4.10 Preliminary User Evaluation

CircuitGlue combines the versatility and extensibility of **TYPE 2** breakout boards with the ease of use of **TYPE 3** integrated modular systems. To collect users' feedback and better understand the utility of CircuitGlue for electronic novices when building interactive prototypes, we conducted a preliminary user evaluation. This user evaluation aimed to better understand the differences when prototyping an electronic system using CircuitGlue versus a traditional approach, in which **TYPE 2** breakout boards are wired to a development board using breadboards and jumper wires.

4.10.1 Participants

We recruited six participants from our research institution, all aged between 25 and 36. Before recruitment, we assessed candidates' experience level by asking them to elaborate on previous projects they had built. As CircuitGlue is specifically designed for novices, we recruited five participants who self-claimed having a rough to basic understanding of electronics prototyping and one participant (P5) with no prior experience building electronic circuits. Participants with a basic knowledge of electronics had only participated in small projects using cheap and common components such as LEDs, buttons, and occasionally breakout boards. The study started with a short interview to further assess the participants' background and familiarity with prototyping workflows.

4.10.2 Procedure

The study design consists of two conditions, which we refer to as *CIRCUITGLUE* and *BREADBOARD*. In both conditions, participants were asked to prototype the smart desktop fan prototype, introduced in the Walkthrough (Section 4.2). Both conditions used the BBC micro:bit as development board, a temperature sensor breakout board (KY-015), a PIR motion sensor breakout board (HC-SR501), and a 12V PC fan consisting of a DC motor and built-in protection circuit. The temperature sensor and PIR motion sensor communicated using digital GPIO signals and required 3.3V and 5V, respectively, while the PC fan required a 12V PWM signal. In the *CIRCUITGLUE* condition, participants had access to three CircuitGlue boards with Jacdac cables for interconnecting them. Additionally, we handed participants a one-page printed guide displaying all three modules and their component names to ease identification. In the *BREADBOARD* condition, participants had access to breadboards, jumper wires, a DC-DC converter module (LM2596), and a DC motor driver (TB6612). Additionally, we gave participants access to a printout of the breadboard circuit diagram (Figure 4.24) for wiring the smart desktop fan. The study followed a within-subjects design where the order of conditions was alternated between participants.

Within the focus of this study, we concentrated on comparing the mechanics of building an interactive prototype through CircuitGlue versus a standard breadboard with additional conversion modules and associated wiring. We, therefore, considered the writing of application logic and the creation of translation code for the modules in both conditions to be out of scope. We believe this is the basis of a fair evaluation, because we are focused on the electronics aspects of prototyping, not the ease of writing code—which would be similar when using microcontroller boards like micro:bit and Arduino without the help of CircuitGlue

We preconfigured the BBC micro:bit with the necessary application logic written in MakeCode⁴ and ensured all three modules were supported in both the *CIRCUITGLUE* and *BREADBOARD* conditions. The application logic for the *CIRCUITGLUE* condition uses Jacdac blocks to interact with the high-level Jacdac services announced by the

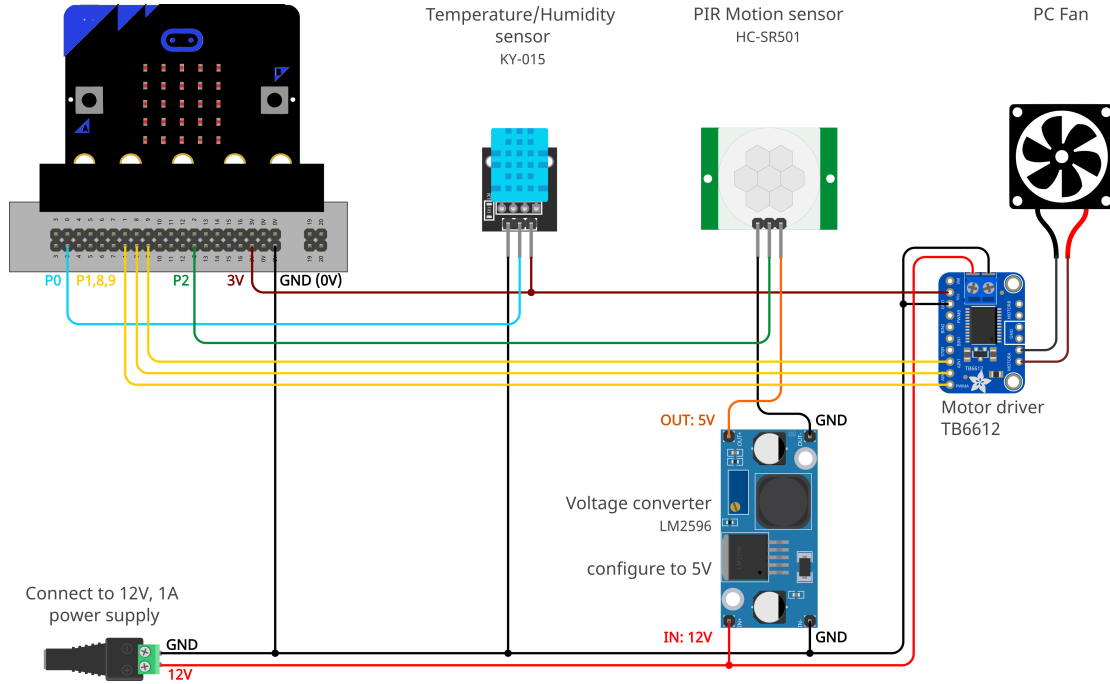


Figure 4.24: Breadboard diagram demonstrating how participants of the user evaluation should interconnect all electronic modules.

temperature sensor, PIR motion sensor, and PC fan. In the *BREADBOARD* condition, the application logic uses default GPIO functions to interact with the PIR motion sensor and DC motor driver. In addition, an external library is used to drive the temperature sensor.

For both conditions, participants received an introduction during which we demonstrated how to connect the PIR motion sensor, which they then had to replicate. We asked participants to think out loud throughout the study and describe their process. We concluded the evaluation with an interview asking participants about their experience with both methods. On average, the user evaluation lasted for 1 hour.

4.10.3 Results

While the results presented in this section are still very preliminary, they already show the usability of CircuitGlue for novices when building electronic prototypes.

On average, participants took 15 minutes ($SD=2.6$) to complete the breadboard prototype and 5.2 minutes ($SD=1.1$) when using CircuitGlue. In the *BREADBOARD* condition, participants expressed being concerned about damaging components by accidentally making faulty connections. All participants described their breadboard prototypes as a mess of wiring and did not feel confident about powering the prototype on before we verified it with them. Indeed, we noticed two participants made a mistake while wiring; after they thought their prototype was finished, we asked some follow-up questions about the connection wires to guide them to their mistake and self-correct it. We did not include this additional time in the measurements. In the *CIRCUITGLUE* condition, P1,

P2, P3, and P5 told us they were not very confident that the board had been configured correctly by the CircuitGlue software, and would have been reassured if they could see details of the current configuration. This contrasts with P4 and P6, who were under the impression that CircuitGlue would automatically verify important parameters and thereby prevent faults if they made a mistake during configuration.

During the post-study interview, we asked participants for each condition how comfortable and confident they felt when connecting modules. All participants reported being more comfortable and confident in the *CIRCUITGLUE* condition. All participants, however, had initial concerns about the size and additional cost that comes with CircuitGlue, especially for simple prototypes. When we explained that only a single CircuitGlue board is strictly necessary because CircuitGlue’s Diagram Generator can be used after each component is tested and operational (Section 4.8.3), all participants agreed that this prototyping style would be practical.

As we gave participants a printout of the exact circuit diagram in the *BREADBOARD* condition to accommodate for their limited electronics expertise, this condition is actually easier than it would likely be in practice. We therefore asked participants if they think they would be able to build the prototype in the *BREADBOARD* condition also without the diagram. All participants thought this would be feasible given enough time and access to online resources, although P4, P5, and P6 added that they would only consider starting such a project if absolutely needed.

We finally asked participants which method they would prefer and which they would potentially adopt when prototyping in the future. All participants preferred CircuitGlue to quickly test different electronic modules before starting to prototype. While P4 and P5 would prefer CircuitGlue for any prototype, P1, P2, P3, and P6 reported that this would depend on the specifics of the prototype, such as the complexity of modules, application area, and whether the purpose of the prototype is experimentation or building a specific device. P6 would only use CircuitGlue when boards are at hand. In addition, P6 also questioned if both methods could be used together and saw potential in a hybrid approach to offload complex components to CircuitGlue while still wiring simple components manually, similar to prototyping styles covered in Section 4.8. P5 especially liked that CircuitGlue allows for only concentrating on I/O components and does not require additional components for conversions.

We did not notice any effect on the order of conditions. Although both conditions instructed participants to build the same prototype, the process used for interconnecting components is different and thus requires different knowledge. While the *BREADBOARD* condition required users to interconnect modules using individual jumper wires according to a diagram, the *CIRCUITGLUE* condition involved plugging in components on a female header and selecting items from a drop-down menu.

4.11 Incorporating User Feedback

Based on the results of the preliminary user evaluation, informal conversations with potential users, and our experiences during the development and testing of the CircuitGlue board, we identified several areas for improvement. As a result, we created a new version of the CircuitGlue board. In addition to enhancing the overall prototyping experience, we focused on improving the board's debugging capabilities, allowing us to push the limits of our design without the need to solder a new board whenever something failed. Ultimately, the redesign enhances usability, safety, and functionality, making the board suitable for both novice and experienced users. Figure 4.25 illustrates the updated CircuitGlue board.

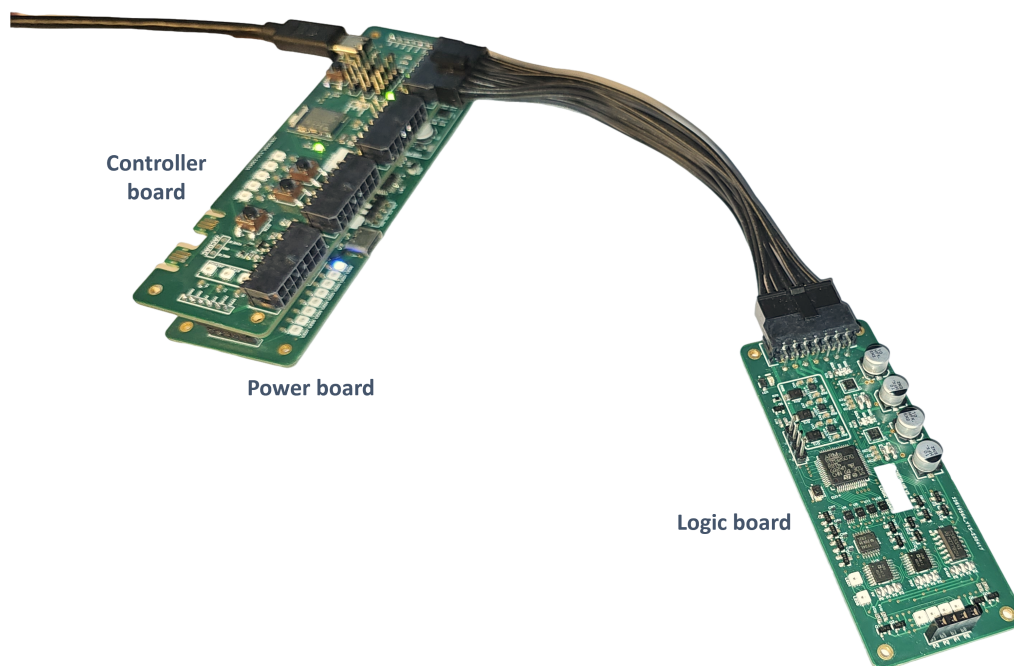


Figure 4.25: Overview of the updated CircuitGlue board.

The following sections summarize the key changes made to the CircuitGlue board. Although the board has undergone significant updates, all core functionalities and prototyping styles remain the same as in the original version.

4.11.1 Modular PCB Design

The CircuitGlue board has been transformed into a modular PCB design, creating separate PCB boards for power management, control, and handling pin function adjustments (logic). This modularization, illustrated in Figure 4.26, facilitates debugging by isolating key sections, enhancing the troubleshooting process during the iterative design phases. While each CircuitGlue system consists of one controller and one power management board, up to four logic boards can be connected to the controller

board. Given each logic board exposes four programmable header pins, the number of programmable header pins is increased from 8 to 16. As a result, CircuitGlue can drive multiple components simultaneously which reduces the overall footprint of a prototype, or drive a single component with a large number of pins, such as parallel displays²².

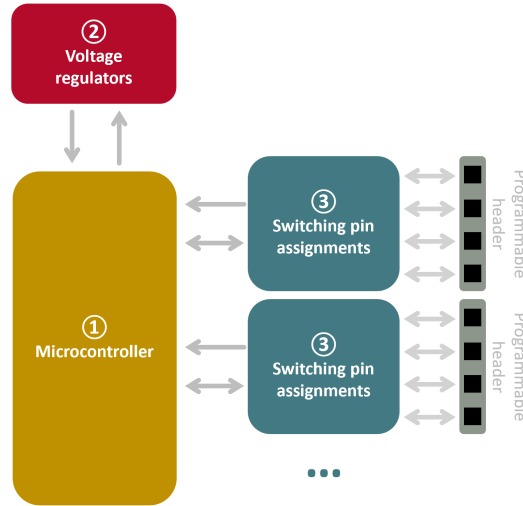


Figure 4.26: The updated design of the modular CircuitGlue system with (1) a controller board containing a System-on-Chip (SoC) for controlling and monitoring the CircuitGlue system, (2) a power board for voltage regulation and USB-C Power Delivery, and (3) a set of logic boards responsible for switching the assignment the programmable header pins.

Each modular CircuitGlue board is equipped with its own microcontroller and communicates via the I2C protocol with the controller, which retains the nRF52840 microcontroller from the initial design as SoC. The power management board utilizes the STM32G071 microcontroller²³, which integrates a USB Type-C Power Delivery controller for negotiating voltages with USB-C power supplies. The logic board is managed by the STM32G070 microcontroller²⁴, which has sufficient GPIO and analog pins for monitoring the status of all its hardware components.

4.11.2 Power Management

While the original CircuitGlue board already included over-voltage and over-current protection sensors, the effectiveness of these protections was primarily dependent on software response time. In our experience, many components can tolerate voltage and current spikes long enough for the System-on-Chip (SoC) to react. However, adding hardware components that respond more quickly, such as programmable fuses and back EMF protection diodes, would further safeguard the board and its connected modules.

As such, the power management system has been significantly enhanced with the

²²<https://www.digikey.com/en/products/detail/futaba-corporation-of-america/ELF1101AA/14669494>

²³<https://www.st.com/en/microcontrollers-microprocessors/stm32g071kb.html>

²⁴<https://www.st.com/en/microcontrollers-microprocessors/stm32g070rb.html>

addition of eFuses, advanced voltage regulators, and crowbar circuits. These components ensure efficient power handling and provide robust protection against electrical faults such as voltage spikes or short circuits. These improvements not only safeguard the hardware but also create a safer learning environment, encouraging users to explore and innovate with reduced risk of failure. Figure 4.27 provides a high-level overview of the improved power management system. In comparison to the original design, the new version of CircuitGlue features two USB-C ports: one data port dedicated to all data communications with the PC accepting 5V and one power port that supports USB-C Power Delivery (PD) and accepts up to 16V. Using this approach, CircuitGlue can be powered through a standard USB-A port on the computer, where the power port provides an optional input in case voltages higher than 5V are required. Additionally, the board can also be powered through the Jacdac port and provide power to the Jacdac bus itself. The CircuitGlue power board automatically switches between USB or Jacdac as input using a two-to-one power multiplexer with priority input set to USB, meaning the board will always choose the USB power source in case both USB and Jacdac are provided. This wide variety of power options ensures the CircuitGlue board can be used in various scenarios and conditions.

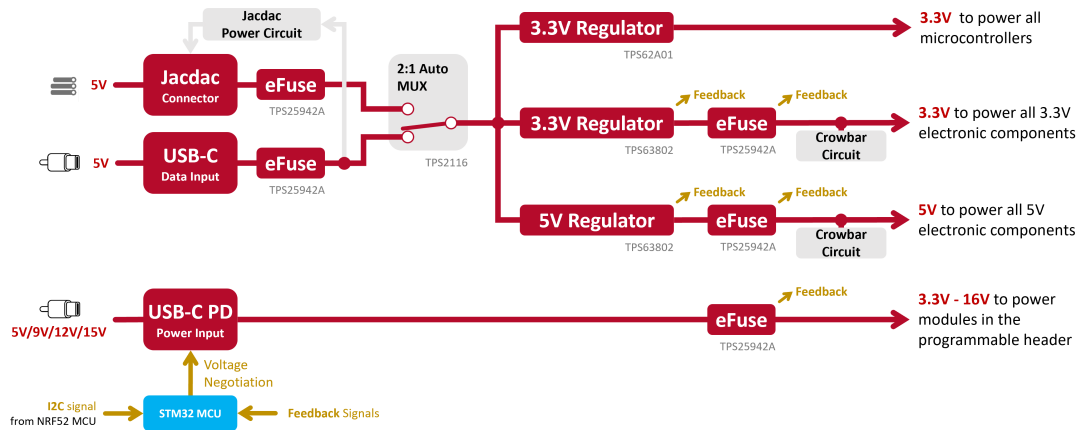


Figure 4.27: Block diagram of the power board illustrating all voltage regulators responsible for providing the voltage levels used by the CircuitGlue logic boards.

4.11.3 Changing Pin Assignments

As shown in Figure 4.26, the CircuitGlue controller board manages the power board and logic boards necessary to correctly assign the functions of the programmable header pins. Figure 4.28 shows a more detailed overview of the logic board, which is responsible for switching the assignments of the programmable header pins. The dashed blocks in this figure represent the electronic circuit for setting the assignment of exactly one programmable header pin; four such identical blocks are present on a single logic board. Each of these blocks embeds the circuitry to either configure the programmable header pin (1) as a General Purpose Input/Output (GPIO) pin, (2) as an analog input pin, (3) as a connection to ground, a programmable voltage supply, or output a PWM

signal. Each logic board can select the programmable voltage to be either 3.3V, 5V, or an adjustable voltage that is shared over all logic boards. The selection circuit is illustrated in Figure 4.29.

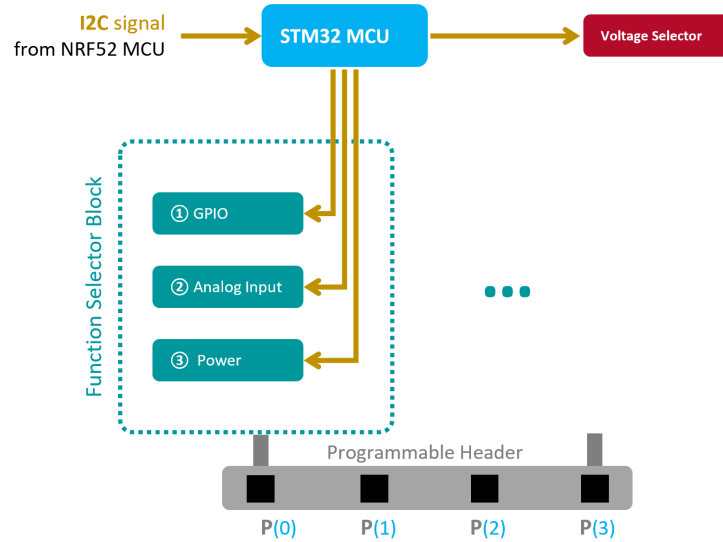


Figure 4.28: Block diagram of the logic board, which is responsible for changing the assignments of four programmable header pins.

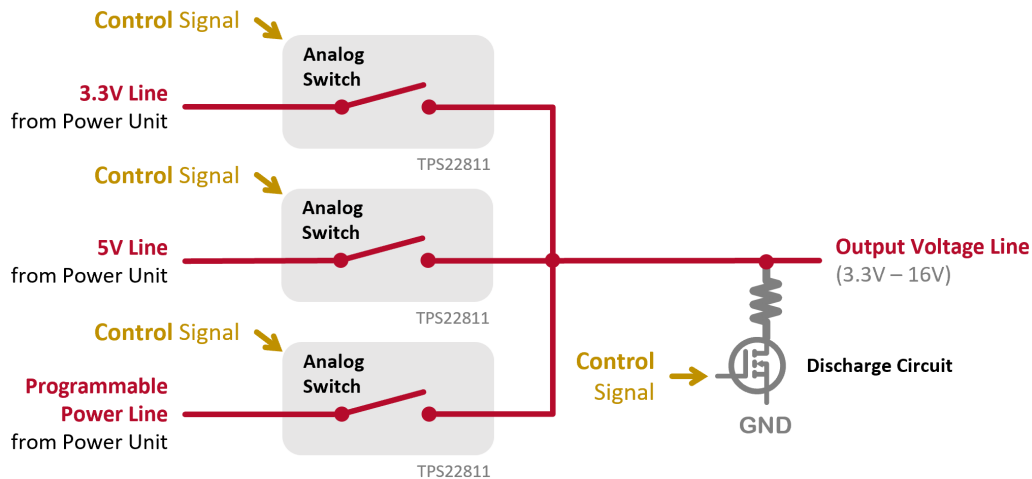


Figure 4.29: Block diagram illustrating the circuit for selecting the output voltage in each logic board.

To create a safer and more controlled environment for experimentation, the redesigned CircuitGlue board integrates high-end components with advanced monitoring features. This upgrade provides finer control over voltage and current, which is essential for monitoring, error detection, and educational purposes to explain the workings of plugged-in components. The basic MOSFET setups have been replaced with an integrated H-bridge IC that offers more sophisticated control options, including built-in Pulse-Width Modulation (PWM) capabilities. This IC also features monitoring functions

for current and voltage, helping to prevent overcurrent and over-voltage situations that could damage components. Figure 4.30 shows the updated circuit for connecting a programmable header pin to either ground, power, or output a PWM signal.

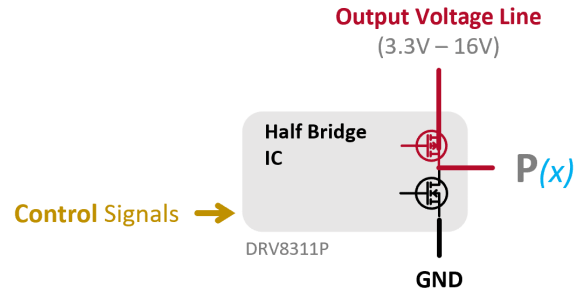


Figure 4.30: Circuit for connecting the programmable header pin to ground, power, or output a PWM signal.

The circuit designed to protect the voltage on the GPIO lines, which originally used a comparator and an AND gate to automatically disconnect the GPIO line when a higher voltage was detected (Section 4.6.3), has been replaced with an integrated analog switch. This new switch can handle voltages up to 16V with a supply voltage as low as 3.3V, and automatically disconnects when voltages higher than the supply voltage are detected. While the original version of CircuitGlue differentiates between signals based on their function, such as analog or digital, the new design differentiates based on the voltage level. As a result, protection circuits such as clamping diodes can be more tailored, and the circuit design becomes less complex. Using a multiplexer, the logic board automatically routes the signals through the correct paths, such as logic level converters. Figure 4.31 shows the design of the circuit handling digital signals up to 5V, and analog signals up to 3.3V.

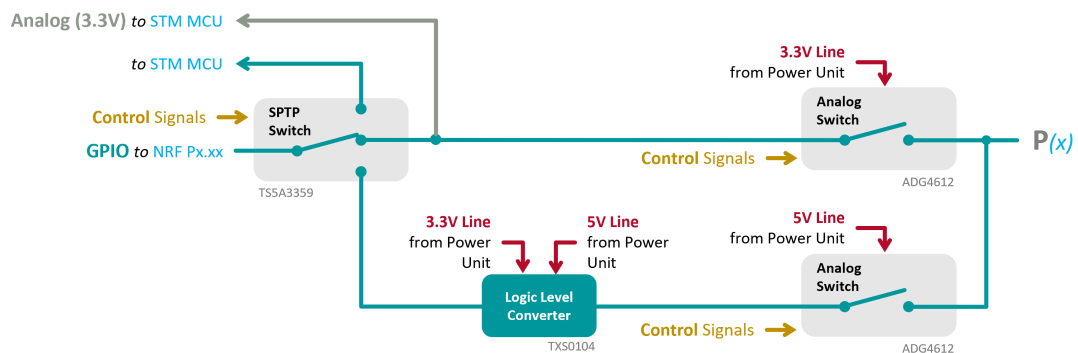


Figure 4.31: Circuit for connecting the programmable header pin to a digital high-speed GPIO pin on the nRF52840 SoC.

Lastly, to handle analog voltages higher than 3.3V, the logic boards uses a voltage follower circuit and voltage divider to translate the input voltage to the appropriate

levels. In comparison the the original version of CircuitGlue, analog signals are read by the STM microcontroller on the logic board rather than the SoC in the controller board. As a result, the new design of CircuitGlue is not limited by the number of analog pins of the SoC in the controller board. Figure 4.32 illustrates the circuit for the high-level analog signals.

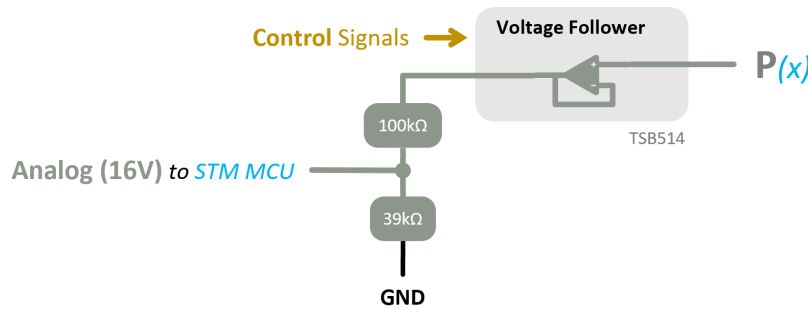


Figure 4.32: Circuit for connecting the programmable header pin to an analog pin on the STM MCU.

By incorporating these advanced features, CircuitGlue offers a more robust and versatile platform that supports innovation while emphasizing user safety and education. The monitoring and protection features provide immediate feedback and a deeper understanding of electrical principles, promoting a top-down learning approach where users can observe component behavior before delving into more complex technical details.

4.11.4 Enhanced Visual Feedback

Modules and components are instantly operational with CircuitGlue and do not require users to understand all the characteristics or workings first. Although our approach initially hides aspects of the electronics in the ‘black box’ formed by the CircuitGlue board, we believe it offers opportunities for a new top-down learning experience, in which students first build prototypes they are highly interested in and later discover and learn about lower level details by conversion to a breadboard circuit (Section 4.8.3).

However, during the preliminary user evaluation, it became apparent that some users were confused about the status of the CircuitGlue board and desired more transparency in the board’s operation. In response, the redesigned version of CircuitGlue incorporates RGB LEDs to provide enhanced visual feedback. These LEDs give instant visual cues about the board’s operational state, making it easier for users to understand and monitor what is happening with their setup at a glance.

While users may not immediately understand the meaning of different LED colors, the presence of visual indicators alone helps in acknowledging activity and functionality. This feature is especially beneficial for beginners, who often find comfort in visual feedback that confirms their actions have had an effect. This can make the learning process more intuitive and reduce the need for complex diagnostic tools, thereby lowering the barrier to entry for electronics prototyping.

4.11.5 CircuitGlue Configuration Tool

Alongside the hardware enhancements, the CircuitGlue interface has been upgraded to provide a more detailed visual representation of the board's status. This improved interface allows users to monitor the operational state of CircuitGlue more effectively, facilitating a clearer understanding of ongoing processes and aiding in the identification of potential issues. As shown in Figure 4.33, the updated interface displays a setup where one DC motor is connected and managed through the Jacdac motor service (a), while an RGB LED module is configured but not yet managed (b). Additionally, it shows the configuration of a second DC motor, which currently has an error icon due to incompatibility with the existing power settings (c).

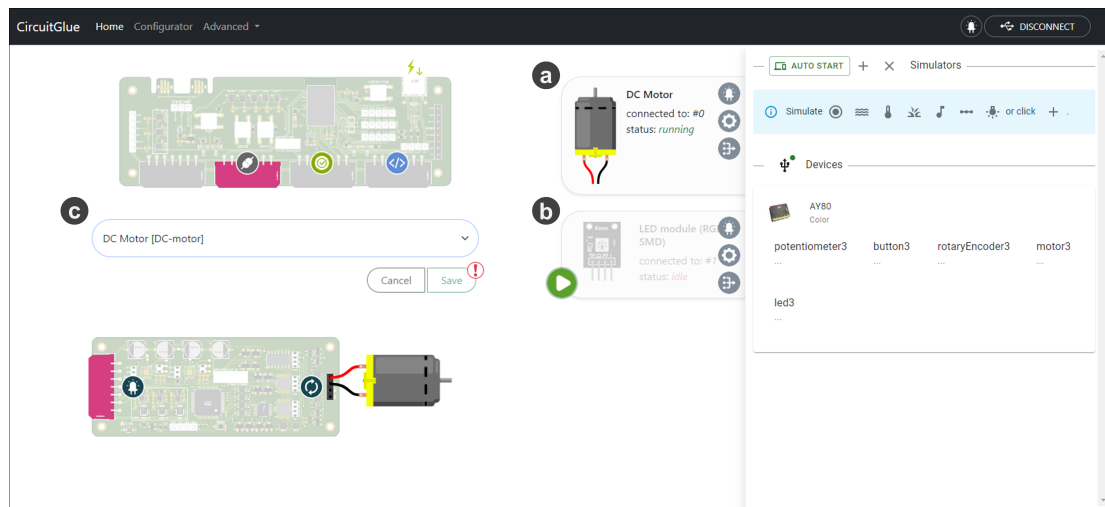


Figure 4.33: Redesign of the CircuitGlue interface, with (a) an active DC motor, (b) a configured RGB LED module, and (c) a second DC motor being configured.

4.12 Discussion and Future Work

Throughout this chapter, we have highlighted how CircuitGlue combines the user-friendly nature of integrated modular systems, commonly known as **TYPE 3** electronics [Lambrichts, 2021], with the adaptability and flexibility found in breakout boards and modules, referred to as **TYPE 2** electronics [Lambrichts, 2021]. While components and modules require configuration before use, we foresee the development of an online repository of reusable configurations that would simplify this process. Unlike conventional modular electronics toolkits that often restrict users to specific ecosystems, CircuitGlue is designed to be versatile, allowing the integration of modules from any supplier, including those outside the CircuitGlue or Jacdac ecosystems. This versatility addresses a significant limitation identified in the literature, as it enables users to incorporate a wider range of components into their projects. Furthermore, the circuit diagram generator (Section 4.8.3) allows users to transition smoothly from using CircuitGlue boards to standard breadboard prototyping, supporting iterative development by reducing initial barriers [Shneiderman, 2006] without limiting what can be built and

without increasing the ultimate cost and complexity of the final implementation.

Our preliminary evaluations have shown that CircuitGlue greatly simplifies the prototyping process for beginners, making it nearly three times faster and significantly reducing errors. This ease of use is especially beneficial for those who want to quickly build prototypes without becoming overwhelmed by the technicalities of circuit design. CircuitGlue's bus architecture supports the creation of large-scale prototypes involving multiple modules, although such projects would require several boards. However, even with just a single CircuitGlue board, users can build extensive prototypes by iteratively replacing CircuitGlue interfaces with custom breadboard circuits once each component or module is tested and operational (Section 4.8.2).

Both our experience with CircuitGlue and its technical evaluations indicate strong compatibility with a broad range of existing modules and suggest it will remain compatible with future developments. Despite this, there are opportunities for further enhancements, such as increasing voltage support up to 24V for driving larger inductive loads²⁵. Introducing a constant current source could also provide more control over the current on universal header pins, which would be advantageous for operating components like LEDs without resistors and for using industrial protocols that rely on a "4 to 20mA current loop." Additionally, integrating a Digital-to-Analog Converter (DAC) could allow for precise output voltages, useful for applications such as audio processing. Future versions of the CircuitGlue board could also be made more compact by placing components on both sides of the PCB and utilizing smaller IC packages where feasible.

A key feature of CircuitGlue is its circuit diagram generator, which facilitates a seamless transition from guided modular prototyping to more advanced hardware assembly using breadboards and jumper wires. While the current algorithm efficiently creates wiring diagrams that ensure component compatibility, there is potential to enhance its flexibility and optimize the placement of components in the diagram. Future improvements could include developing a more advanced algorithm that automatically combines necessary conversion components, such as level shifters and voltage regulators, based on the specific needs of the components being used. This would allow the generator to dynamically determine the optimal number and types of converters required, improving power efficiency and communication compatibility. For example, when generating the diagram for two DC motors, CircuitGlue will add two separate motor drivers, even if a single motor driver can drive two motors at the same time.

Moreover, CircuitGlue has significant potential to expand its educational capabilities by leveraging its extensive knowledge of the plugged-in electronic component. Future work could provide users with detailed explanations of key concepts such as voltage levels, logic levels, and communication protocols. When generating a circuit diagram, the software could display contextual information about each component's voltage requirements, the function of any added conversion components, and the protocols used

²⁵<https://www.adafruit.com/product/5141>

for communication. This information could be presented through interactive tutorials or tooltips, helping users develop a deeper understanding of the components and their interactions.

To further enhance the educational experience, CircuitGlue could introduce structured learning modules that progressively cover more advanced topics, such as signal integrity, power distribution, and the fundamentals of analog and digital electronics. By gradually decreasing automation and encouraging users to make more decisions about component selection and wiring, CircuitGlue could foster a more hands-on learning environment that promotes exploration and experimentation. Adding display elements like an OLED or LCD screen next to the header pins could provide real-time feedback on the current configuration, enhancing both usability and learning. Additionally, exploring the development of a wireless, battery-powered CircuitGlue board with the ability to expose connected modules over a wireless Jaccadac implementation could offer valuable insights into the behavior of sensors and actuators in real-world conditions, outside of controlled environments.

Future research should also focus on evaluating CircuitGlue over an extended period of time to understand its long-term benefits and explore how its features can be optimized for a broader user base. Empirical studies should examine the platform's effectiveness in enhancing the users' experience in electronics prototyping and its impact on learning outcomes. Moreover, a detailed technical evaluation of the CircuitGlue board, including assessments of analog crosstalk, slew rate, and characteristic impedances across different frequencies, would offer valuable insights into its performance and areas for enhancement. This comprehensive analysis can help identify potential limitations and inform further refinements.

4.13 Summary

This chapter introduced CircuitGlue, an electronic converter that automates the process of interconnecting heterogeneous electronic components and modules by ensuring voltage levels, interface types, communication protocols, and pinouts are compatible. Each pin in the programmable header of the CircuitGlue board can support either power, analog signals, or digital communication. We demonstrated several new prototyping styles created by CircuitGlue and evaluated their usability by conducting a preliminary user study with six participants. Their responses indicate that CircuitGlue significantly speeds up building electronic prototypes and is likely to accrue the greatest benefits in educational settings and for makers who quickly want to prototype something. While a more extensive study over a longer period is necessary to fully understand its impact, these initial findings indicate that CircuitGlue is a valid solution for addressing hardware compatibility issues (Q3). Furthermore, our technical evaluation demonstrates that there are minimal trade-offs for typical prototyping tasks, effectively addressing Q2.

The development and preliminary evaluation of CircuitGlue, as presented in this chapter, highlight its significant potential in advancing electronics prototyping. CircuitGlue's design addresses the compatibility challenges faced when integrating various hardware components, thereby fulfilling the second research goal (**G2**). It simplifies hardware assembly by enabling software-configurable pin assignments, protocol translations, and voltage conversions. These features make CircuitGlue particularly valuable for beginners and those looking to expedite their prototyping process without being constrained by the technical complexities typically associated with electronics design.

PLUG-AND-PLAY SOFTWARE DRIVERS THROUGH LOGICGLUE

Motivation

Building on the development of CircuitGlue discussed in Chapter 4, we identified that while hardware integration challenges were addressed, similar issues persisted in the software domain. This realization led to the creation of LogicGlue, a novel *software glue* designed to extend the principles of seamless integration into software. LogicGlue simplifies the electronics prototyping process by enabling the development of platform-independent drivers and hardware-independent application logic, effectively addressing software compatibility challenges (G3).

LogicGlue specifically addresses research question Q3, which explores strategies for overcoming compatibility issues between hardware and software components in physical computing. Traditional prototyping tools often face limitations due to the tight coupling of drivers and hardware, making it difficult for developers to adapt their projects to different platforms or integrate various components without extensive modifications. By introducing a universal driver specification, LogicGlue ensures that software written for one hardware setup can be effortlessly ported to another, promoting flexibility and reducing the need for redundant coding efforts.

Although this chapter does not include a formal user study to evaluate the usability of LogicGlue for various user groups, its design is inherently grounded in the practical needs observed during the development and deployment of CircuitGlue. The challenges faced in hardware integration directly informed the features of LogicGlue, ensuring that it effectively meets the requirements of diverse users working with heterogeneous hardware components. Furthermore, this chapter demonstrates that LogicGlue supports advanced programming structures with only minimal performance latency, addressing research question Q2. By prioritizing ease of use and maintaining hardware-specific functionalities, LogicGlue contributes to democratizing physical prototyping and lowering the barriers to electronics prototyping.

This chapter is based on the conference proceedings paper, “LogicGlue: Hardware-Independent Embedded Programming Through Platform-Independent Drivers”, which is submitted for the “ACM SIGCHI Symposium on Engineering Interactive Computing Systems”. If accepted, the paper will be presented at the EICS conference in 2025 in June, Germany.

5.1 Introduction

Over the past decade, advancements in microcontrollers, sensors, and actuators have integrated embedded systems into many aspects of daily life. This integration has democratized prototyping, enabling students [microbit, 2022], hobbyists [Arduino, 2022], and programmers to create electronic projects. A key factor in this democratization is the extensibility of microcontrollers with various third-party components and the availability of software libraries, such as the popular Arduino libraries [Arduino, 2024a], which facilitate programming.

Software for embedded systems typically includes three essential elements: low-level drivers, high-level programming libraries, and application logic. Low-level drivers manage direct communication between electronic components and the microcontroller, handling hardware-specific registers and protocols. For example, the SSD1306 OLED display¹ driver controls hardware registers via I2C or SPI protocols to update pixels and adjust settings like brightness. These drivers simplify the complex details of communication protocols and hardware registers, enabling programmers to focus on higher-level tasks instead of dealing with intricate timings and specific sequences required for hardware operations. High-level programming libraries provide an additional layer of abstraction, simplifying hardware interaction and making components more accessible to programmers. For instance, the Adafruit_SSD1306 library² allows developers to display text on the SSD1306 display without dealing with pixel-level operations.

While low-level drivers and high-level programming libraries significantly lower the barrier for programming embedded systems, they often come as tightly coupled packages specific to certain components and platforms, which limits compatibility and flexibility significantly. For example, the DHT sensor library³, developed for the DHT11 temperature sensor, is not compatible with, for example, the DS18B20 temperature sensor. Despite offering similar functionalities, such as reading the temperature in Celsius or Fahrenheit, the library for the DHT11 temperature sensor embeds low-level driver functions that are specific to the DHT11 sensor, such as data reading methods and communication protocols. Moreover, many libraries are designed for specific platforms, like Arduino [Arduino, 2022], making them incompatible with others such

¹ <https://www.arduino.cc/reference/en/libraries/ssd1306/>

² https://github.com/adafruit/Adafruit_SSD1306

³ <https://www.arduino.cc/reference/en/libraries/dht-sensor-library/>

as Raspberry Pi [Pi, 2022b] or micro:bit [microbit, 2022]. This tight coupling presents significant challenges for developers in finding the appropriate library for a component and ensuring compatibility with their development platform.

Some generic libraries, like the Adafruit GFX Graphics Library⁴, offer broader compatibility with third-party components by providing a range of abstractions over various low-level drivers. However, these libraries often cannot leverage each component's unique features and extending them to support new functionalities usually requires extensive knowledge of their architecture. For example, the Adafruit GFX library is primarily designed for displays communicating over SPI protocols, making it challenging to drive displays over I2C, such as the SSD1306.

High-level communication protocols like Jaccad [Devine, 2022] and CoAP [OASIS, 2024a] offer an alternative solution by abstracting low-level drivers into standardized interfaces. Jaccad, for instance, uses services to separate application logic from low-level drivers, increasing compatibility and significantly lowering the barrier for embedded programming. However, since electronic components cannot communicate directly with these high-level protocols, an additional microcontroller is required for each electronic component to handle the conversion, introducing latency and potential loss of unique component features that are not captured by the protocol.

In this chapter, we introduce LogicGlue, a novel software stack that allows for writing platform-independent drivers and, as such, allows for writing hardware-independent application logic. To realize this, LogicGlue consists of a novel driver specification (Figure 5.1a) for expressing the behavior of electronic components. The LogicGlue driver specification defines all functionalities and characteristics of electronic components, ranging from specific communication protocols to the formatting of data and their units. As these definitions are specified in bytecode, the LogicGlue driver specifications can be processed by any microcontroller and any programming language, making them platform-independent. As such, users are not restricted to a single microcontroller or platform.

The LogicGlue interpreter (Figure 5.1b) operates on the microcontroller and translates the instructions from the driver specification into platform-specific commands. Additionally, the high-level LogicGlue programming library (Figure 5.1c) facilitates interfacing with electronic components via the interpreter. Unlike high-level communication protocols, like Jaccad [Devine, 2022], LogicGlue does not require the translation of features of electronic components to services nor the conversion of communication messages to a single protocol. Instead, the driver commands of LogicGlue ensure all components' features remain available, and interfacing is done via the native signals supported by the electronic components, avoiding additional overhead.

⁴ <https://learn.adafruit.com/adafruit-gfx-graphics-library/overview>

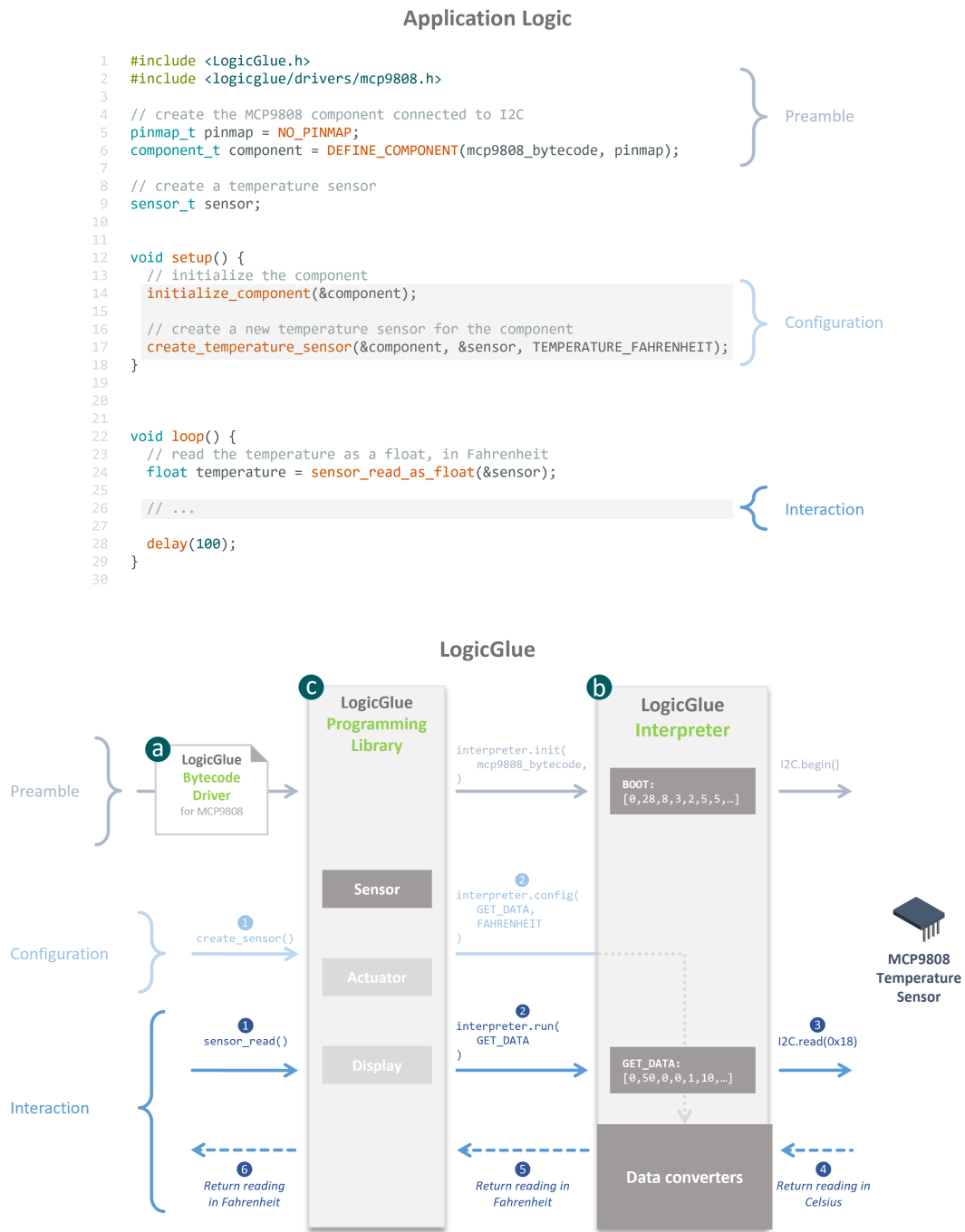


Figure 5.1: Overview of LogicGlue. (a) The novel driver specification of LogicGlue encodes the behavior of drivers in bytecode to ensure platform independence and compatibility across various microcontrollers and programming languages. (b) The LogicGlue interpreter is responsible for processing the bytecode driver specifications and executing platform-specific commands. (c) The LogicGlue programming library is designed to facilitate interaction with electronic components through the interpreter.

LogicGlue empowers developers to write hardware-independent application logic. Swapping, for example, a temperature sensor working over the I2C protocol, with one outputting analog voltage readings, does not require rewriting application logic as LogicGlue automatically handles the required data conversions. Furthermore, our buildup ensures that microcontrollers or development platforms, that implement the LogicGlue interpreter and the LogicGlue programming library, are instantly compatible with all electronic components for which LogicGlue driver specifications are available. Likewise, electronic components for which a LogicGlue driver specification is available are compatible with any microcontroller that supports LogicGlue.

In summary, we contribute:

1. A LogicGlue driver specification format, allowing the creation of platform-independent drivers for electronic components through bytecode commands.
2. An accompanying visual block-based programming interface to facilitate writing LogicGlue driver specifications in bytecode.
3. A LogicGlue interpreter for executing driver specifications.
4. A simple high-level programming library to interface with electronic components via the interpreter.

5.2 LogicGlue

LogicGlue consists of several components as shown in Figure 5.1. (a) The *LogicGlue Driver Specification*, which enables platform-independent instructions for specifying the workings and characteristics of electronic components. These driver specifications are composed in a visual block-based programming interface and are then stored in a bytecode format. This bytecode encodes an instruction set that is platform-independent and thus cannot be executed directly by the hardware. (b) The *LogicGlue Interpreter* processes and executes the bytecode instruction set on a specific microcontroller platform, such as Arduino or micro:bit. (c) The *LogicGlue Programming Library* offers high-level functions for interacting with the electronic components through the LogicGlue interpreter. This library offers an interface for interacting with components, allowing developers to write application logic without focussing on specific hardware considerations, such as protocols or data formats. This architecture enables swapping electronic components with minimal to no rewriting of the application logic, as we will demonstrate in the walkthrough below.

5.2.1 Writing Application Logic

To explain the procedure for writing the application logic, we will provide an example from the perspective of a DIY enthusiast, Alex, who integrates a temperature sensor into a prototype with the help of LogicGlue. Considering the wide variety of temperature sensors available, such as analog, digital, and infrared temperature sensors, Alex wants to experiment with different models to find the most suitable one for his project.

Traditionally, this would be a lengthy and complex process, as each component might use a different protocol and require different signal processing methods. In the best-case scenario, a software library is available for each component, and Alex only needs to rewrite the application logic three times to work with three libraries. For example, the library of the MCP9808 digital temperature sensor⁵ embeds features for initializing and using the I2C protocol, while the library for an analog temperature sensor handles all analog-to-digital conversions. If no library is available that is compatible with the development platform used, an additional driver has to be written first based on the specifications in the datasheet. Figure 5.2 shows the major differences in code for interacting with the MCP9808 digital temperature sensor over I2C, using the Adafruit library, and the TMP36 analog temperature sensor using analog readings, on the Arduino platform. Besides differences in protocols between the sensors, there are also major differences in output readings, as one temperature sensor outputs temperatures in Celsius while the other outputs voltages. Additional processing is thus needed.

a TMP36 temperature sensor (analog)

```

1 #define TMP36pin A0
2
3 void setup() {
4 }
5
6 void loop() {
7   // read the temperature
8   int sensorValue = analogRead(TMP36pin);
9
10  // convert the reading into Fahrenheit
11  float voltage_mV = ((sensorValue * 5.0) / 1023.0 - 0.5) * 1000;
12  float temperature_C = voltage_mV / 10.0;
13  float temperature_F = (temperature_C * 1.8) + 32;
14
15  // ...
16
17  delay(100);
18 }
```

b MCP9808 temperature sensor (digital)

```

1 #include <Wire.h>
2 #include "Adafruit_MCP9808.h"
3
4 Adafruit_MCP9808 tempsensor = Adafruit_MCP9808();
5
6 void setup() {
7   // initialize the sensor and setup I2C
8   if (!tempsensor.begin(0x18)) {
9     return;
10  }
11 }
12
13 void loop() {
14   // read the temperature
15   float temperature_C = tempsensor.readTempC();
16
17   // convert the reading into Fahrenheit
18   float temperature_F = (temperature_C * 1.8) + 32;
19
20   // ...
21
22   delay(100);
23 }
```

Figure 5.2: a) The traditional code that is required to interact with the TMP36 analog temperature sensor. b) The traditional code that is required to interact with the MCP9808 digital temperature sensor.

Using LogicGlue, Alex does not need to write multiple versions of the application logic to test different temperature sensors. Instead, he can include the respective driver specification file and test each temperature sensor using the same application logic code. Figure 5.3-c shows the code to interact with the MCP9808 digital temperature sensor, working over I2C, and the TMP36 analog temperature sensor, using analog readings, through LogicGlue. As shown in Figure 5.3a-b, only the preamble changes when swapping the two very different temperature sensors. These changes include the driver specification file, the parameter passed in the constructor, and the references to the microcontroller pins in which the component is plugged. Despite the difference in output readings (Celsius versus voltages), Alex specifies in both code snippets that he prefers temperature readings in Fahrenheit (line 20). The LogicGlue interpreter automatically converts the components' readings into the format preferred by Alex.

⁵ <https://www.arduino.cc/reference/en/libraries/mcp9808/>

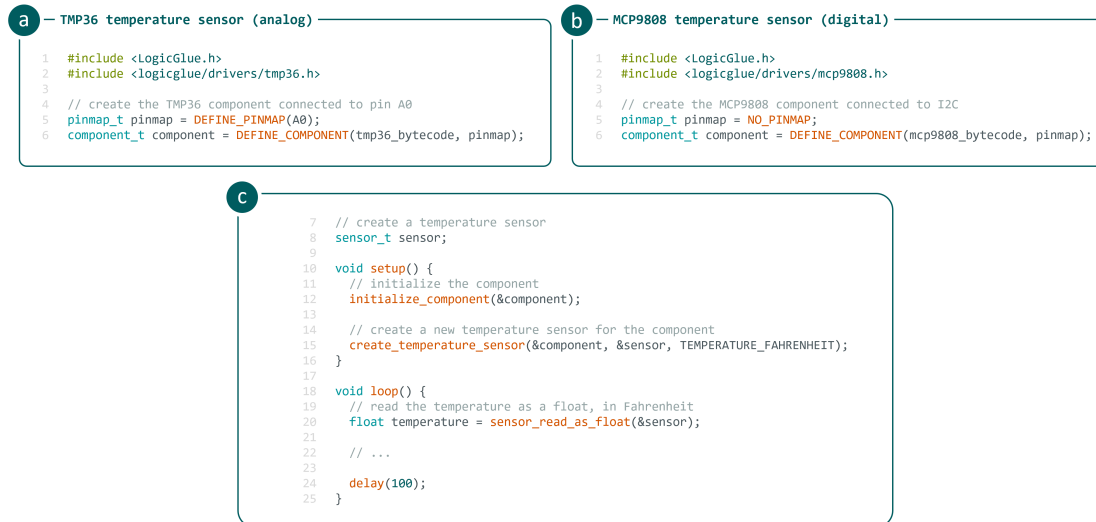


Figure 5.3: a) Preamble for including the LogicGlue driver specification for the TMP36 analog temperature sensor. b) Preamble for including the LogicGlue driver specification for the MCP9808 digital temperature sensor. c) Application logic interacting with either temperature sensor using temperatures in Fahrenheit.

5.2.2 Writing Driver Specifications

LogicGlue driver specifications include all functionalities for driving an electronic component. To streamline writing driver specifications, we developed a block-based specification interface based on Blockly⁶. As shown in Figure 5.4, users compose commands by selecting command blocks representing various functionalities. When the specifications are complete, the interface stores this graphical representation of the driver specifications as a bytecode sequence in a header file. This file is then included in the application logic as demonstrated in Section 5.2.1. To correctly compose driver specifications, a good understanding of the components' datasheet is required. However, this is a one-time effort, best done by component manufacturers or experienced engineers, and then shared with all customers or users.

The following example illustrates composing the driver specifications for the MCP9808 temperature sensor⁷. In the *init* procedure, we initialize the component's communication protocol, which, in this case, is the I2C protocol. As shown in Figure 5.5, we insert a *configure* block (line 1), select I2C as the protocol, and set the frequency to 400kHz as specified in the sensor's datasheet. The datasheet further details that this component is available via address 0x18 on the I2C bus, and uses 16-bit registers (line 2). A common practice for I2C devices then involves verifying the component's presence by reading out the manufacturer ID register (0x06) and comparing it to the ID detailed in the datasheet (0x54) using an assert (line 3). The last block in the start procedure involves setting the sensor's default configuration, using a write command, to ensure the sensor uses its default settings at startup. As shown in line 4, we set the data of the configuration

⁶ <https://developers.google.com/blockly>

⁷ Datasheet: <https://ww1.microchip.com/downloads/en/DeviceDoc/25095A.pdf>



Figure 5.4: LogicGlue's graphical interface for creating drivers using the driver specification.

register (0x01) to its default value (0) as specified in the sensor's datasheet.

The next step involves defining the features supported by the MCP9808 temperature sensor, such as temperature readings, resolution updates, and sleep and wake-up functionalities. Each functionality is created using a *functionality* block. For specifying the temperature reading functionality, we select *GetData* as the function category. This field defines that this is the primary functionality of the component, compared to, for example, the *resolution* feature. Within the definitions of this block, we specify the return parameter type and instructions for reading the temperature. The return type is set using a *output* block (line 5), which characterizes the return type. As specified in the datasheet, the MCP9808 temperature sensor returns temperature readings in Celsius. Alternatively, this block can support other function categories, such as setting data, triggering specific features like sending the pixel buffer to a display, or reading internal parameters like the internal temperature of an electronic component.

To read the temperature (line 6), we follow the instructions specified in the sensor's datasheet. Using a *read* block, we first read the temperature register (0x5) from the MCP9808 temperature sensor and store the data in a temporary variable. As the data is in two's complement format, we follow the calculations in the datasheet to convert the bits to a decimal temperature. First, we split the 16-bit value into an upper (line 7) and lower (line 8) byte and clear the flag bits in the upper byte. As the positive and negative temperature data are computed differently, we use an *if* block (lines 9-12) to check if the sign bit (0x10) that indicates a negative value is set (line 9). If this bit is set, the if-test evaluates to true, and lines 10-11 are executed. Line 10 resets the sign bit, while line 11

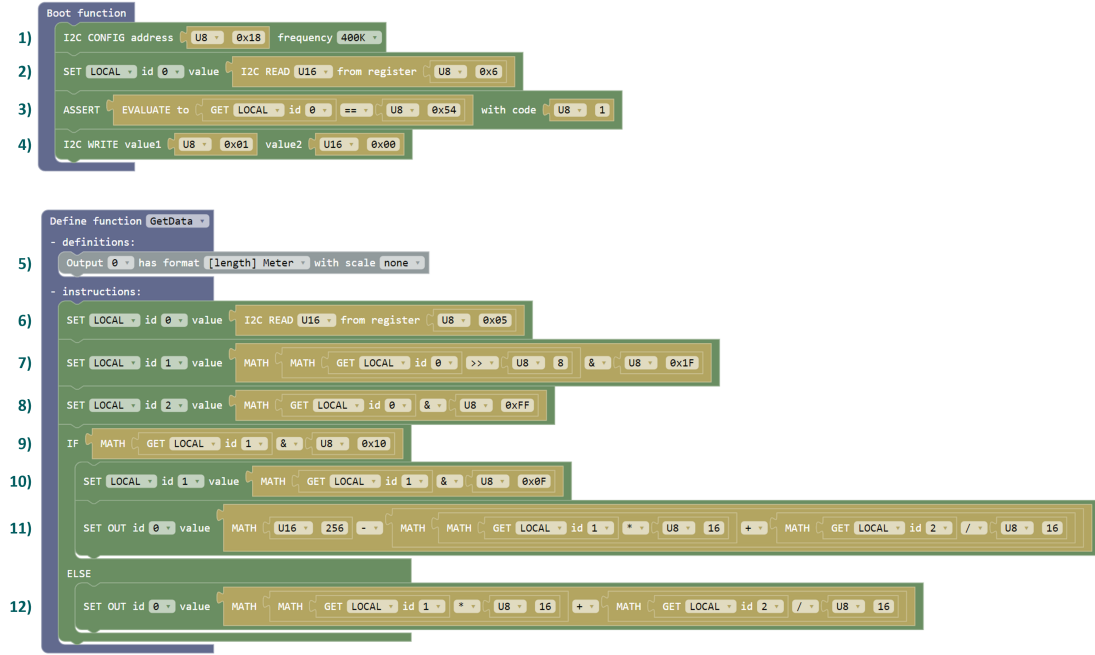


Figure 5.5: LogicGlue driver specification for the MCP9808 temperature sensor.

executes the mathematical operations specified in the datasheet to calculate a negative temperature in Celsius and stores the result in the output variable. If the bit indicating a negative value is not set, line 12 is executed, which calculates and stores a positive temperature in Celsius.

Once all functionalities are created, the LogicGlue interface automatically stores this driver specification as a bytecode sequence in a header file.

5.3 Related Work

This section discusses the contributions that have influenced the development of LogicGlue, focusing on advancements in software abstraction, standardized communication interfaces, and intermediate representation layers. Detailed descriptions of all instructions and their semantics are provided in Appendix B.1.

5.3.1 Software Abstraction

Over the years, embedded systems have been significantly shaped by the advent of software platforms and frameworks aimed at abstracting hardware complexities. Notable platforms such as PlatformIO [PlatformIO, 2024], Mbed OS [Mbed, 2024], and Zephyr [Zephyr, 2024b] have been instrumental in offering operating system-like functionalities to microcontrollers. These platforms mask the intricacies of hardware, allowing developers to concentrate on application logic. The role of Arduino [Arduino, 2022] in democratizing electronics through an easy-to-use hardware and software platform has further made embedded programming accessible to a broad audience.

The evolution has also been marked by the emergence of Real-Time Operating Systems (RTOS) like FreeRTOS [FreeRTOS, 2024], which provide concise, scalable, and flexible software management for embedded devices. Similarly, TinyOS [Levis, 2005] has played a pivotal role in promoting the development of networked sensor systems, highlighting the importance of specialized platforms in the advancement of Internet of Things (IoT) and embedded applications. Adding to this, ROS [ROS, 2024] offers a comprehensive set of software libraries and tools that assist users in building robot applications.

Challenges persist in integrating external hardware components despite these advancements. Zephyr’s implementation of Device Drivers [Zephyr, 2024a] illustrates a framework for hardware management, yet adapting these drivers for new components underscores the necessity for a deep understanding of hardware-software interaction. While Zephyr’s Device Drivers share a concept similar to the LogicGlue driver specification, LogicGlue focuses on interacting with external hardware, while Zephyr is designed to offer a standardized approach for setting up hardware peripherals. Furthermore, LogicGlue offers a convenient block-based interface for creating the drivers.

In parallel, frameworks like Codal [Devine, 2018] and Arduino [Arduino, 2022] have significantly eased microcontroller programming, streamlining direct hardware interaction. However, they often fall short in addressing the complexities of integrating external components, a task that still poses considerable challenges. Libraries such as the Adafruit GFX Graphics library [Burgess, 2024] aim to bridge this gap by abstracting hardware communication, yet these libraries often cannot leverage each component’s unique features, and integrating them in the application logic requires a good understanding of the functionalities offered by the library.

Integrated Development Environments (IDEs) have made significant inroads in addressing these challenges. The Arduino IDE [Arduino, 2024b], DeviceScript [Microsoft, 2024a], and Microsoft MakeCode [MakeCode, 2024] integrate tools that facilitate the discovery, selection, and integration of software libraries, enhancing the efficiency of the development workflow. These IDEs, alongside advanced environments like Visual Studio Code [Microsoft, 2024b] with its rich features for embedded development, play a pivotal role in lowering the entry barriers to embedded programming.

In general, while existing software platforms and frameworks have significantly simplified hardware interaction, they often come with a standardized abstraction layer that doesn’t account for the unique functionalities of individual components or the varied needs of developers. LogicGlue introduces a novel approach by offering platform-independent driver specification alongside a versatile programming library, ensuring developers can write hardware-independent application logic. This eliminates the limitations posed by the tight coupling of drivers and libraries in conventional systems, allowing for seamless hardware changes without extensive code adjustments.

5.3.2 Standardized Communication Interfaces

The concept of integrated modular systems has seen substantial development, with ecosystems like Jacdac [Devine, 2022], Modular-Things [Read, 2023], and .NET Gadgeteer [Hodges, 2013] leading the way in standardized communication interfaces. These systems offer a range of compatible components that communicate through standardized protocols, enabling easy system assembly and expansion. The Raspberry Pi platform [Pi, 2022b], with its extensive ecosystem of hardware add-ons and Hardware Attached on Top (HATs), exemplifies the power of modular design in promoting system scalability and interoperability.

SoftMod [Lambrichts, 2020], a concept that emphasizes configuring component behavior through their physical arrangement, represents a novel approach in this domain. This strategy facilitates a tangible and intuitive method for modifying system functionalities, showcasing the potential for physical configuration to impact software behavior directly. Further contributions to this field include the Intel Edison [Edison, 2024] and PMod [Pmod, 2021] platforms, which were designed to foster innovation in IoT and embedded projects through modular components and standardized interfaces. Similarly, the BeagleBone [BeagleBone, 2021] series offers an open-source platform that encourages experimentation and development with its cape plug-in boards, underlining the versatility of modular design in embedded systems.

Interoperability in embedded systems necessitates interacting over diverse communication protocols like SPI, I2C, and UART. While these protocols are frequently used to interact with electronic components, using them requires significant knowledge in embedded development [Lambrichts, 2021]. High-level protocols such as Jacdac [Devine, 2022] provide a simplified method for device communication, wrapping low-level communication protocols like I2C and SPI into a high-level communication standard. In practice, they introduce an ecosystem of modules that all share the same communication interface to streamline system assembly. In addition, high-level protocols tailored for IoT applications, like MQTT [OASIS, 2024b] and CoAP [OASIS, 2024a], have become prominent, offering lightweight solutions for resource-constrained environments. These protocols exemplify advancements in ensuring devices from various manufacturers can communicate seamlessly, fostering a more cohesive ecosystem. Here, frameworks like TinyOS [Levis, 2005] and Static TypeScript [Ball, 2019] have made contributions by focusing on networked systems and providing platforms for building complex, interconnected devices. However, compared to low-level communication protocols like SPI and I2C, these approaches often introduce additional translation steps to make electronic components compatible with the communication interface. In particular, each feature of an electronic component needs to be able to be exposed through the standardized interface, potentially leading to the loss of unique features and latency issues.

LogicGlue stands out by allowing for direct interaction with hardware components

through their native protocols without resorting to standardized interfaces. This direct approach ensures that the unique features of each component are preserved, offering developers a more efficient and feature-rich integration experience. Furthermore, the platform-independent nature of LogicGlue driver specifications ensures that any microcontroller or development platform implementing the LogicGlue interpreter and programming library becomes instantly compatible with all supported electronic components. Developers can switch between components with different communication protocols or functionalities without rewriting application logic, significantly reducing the complexity and improving the adaptability of embedded systems development. This approach not only streamlines development but also enhances the potential for creative hardware solutions by simplifying the integration of diverse components.

5.3.3 Intermediate Representation Layers

The adoption of high-level programming languages and abstraction layers has significantly transformed software development practices. Platforms like Node-RED [Node-RED, 2024], which provides a visual programming environment for IoT applications, illustrate the effectiveness of abstracting complex code into more accessible formats. Similarly, the introduction of TypeScript [Ball, 2019] has offered developers a powerful tool for building large-scale applications by providing types and high-level syntax that compile down to JavaScript, suitable for web and server environments.

Intermediate Representation Layers (IRLs) are key in bridging high-level programming constructs with the lower-level code required by microcontrollers and embedded devices. For instance, LLVM [LLVM, 2024] provides a wide range of tools and libraries that support converting high-level language code into machine code, facilitating cross-platform application development. This concept is crucial in understanding how abstract code structures can be effectively translated into executable commands that run on hardware devices.

Traditional software development practices for embedded systems often rely on converting the full application logic into an IRL, focusing primarily on programming the microcontrollers themselves. This approach is instrumental in bridging the gap between high-level programming constructs and the lower-level executable code required by microcontrollers, as seen in platforms leveraging LLVM [LLVM, 2024] or similar technologies. The primary objective here is to streamline the development process for the microcontroller's software, ensuring that high-level abstractions are effectively translated into machine-level commands that the hardware can execute.

LogicGlue, while embracing a conceptually similar use of IRLs, diverges in its application and objectives. Rather than focusing on the microcontroller's entire application logic, LogicGlue facilitates the interactions with hardware components. Its driver specification outlines the commands for interfacing with hardware components. This specificity ensures that developers can engage with the unique functionalities of each component

directly, without the intermediary step of translating general-purpose application logic into hardware-specific commands.

5.4 LogicGlue Driver Specification

The LogicGlue driver specification provides a comprehensive framework to characterize all functionalities of an electronic component. These functionalities range from initialization procedures to read and write actions, as well as fine-tuning the component's settings. The LogicGlue driver specification is based on the Reduced Instruction Set Computing (RISC) architecture and is Turing complete, indicating its capability to execute any computable function given sufficient resources. LogicGlue supports comprehensive data management, conditional operations, and program flow control.

Each instruction in the LogicGlue driver specification is represented by a predefined numeric code, organized within an enumeration (enum). Using an enum allows each instruction to be given a descriptive name, which enhances code readability and maintainability. While the driver specification itself is physically stored as a traditional array of bytes, this section presents all instructions using their enum names, along with indentation and color coding, to improve clarity and comprehension. Additionally, we have included a side-by-side visualization of the block-based representation in the LogicGlue interface to further aid understanding.

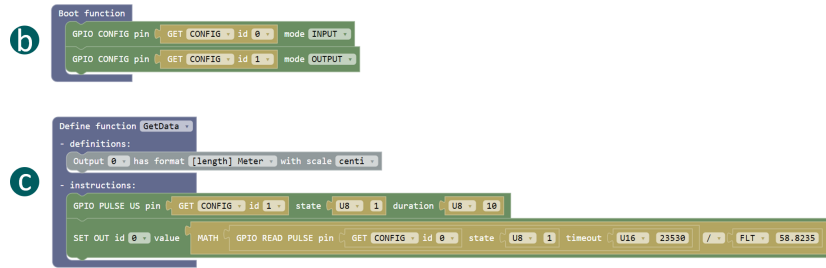
5.4.1 Function Definitions

The LogicGlue driver specification comprises several functions that represent the available features of an electronic component. This section explains how these functions are defined within the LogicGlue driver specification, using the driver for the HC-SR04 ultrasonic distance sensor as an example (Figure 5.6).

The specification begins with the bytecode definition (Figure 5.6a), specifying the required microcontroller resources for the environment in which the driver will run, including communication protocols and memory allocation requirements. These details ensure that the electronic component is compatible with the intended microcontroller or platform, which is crucial for correct operation within a given hardware setup. For example, if a sensor requires an I2C communication protocol and specific GPIO pins, these requirements are explicitly stated to prevent issues related to incompatibility. When using the LogicGlue visual interface, the bytecode definition is automatically determined and added when storing the driver specification header file.

Following the bytecode definition, the boot function (Figure 5.6b) provides instructions necessary for configuring both the microcontroller and the electronic component. This includes configuring communication protocols, initializing GPIO pins, and setting configuration registers. For instance, configuring an I2C temperature sensor would involve setting the I2C address and preparing the necessary registers to read temperature

LogicGlue Visual Block-based Interface



LogicGlue Driver Specification

```

// custom defines
#define PIN_ECHO          CONFIG(CFG_0)
#define PIN_TRIGGER       CONFIG(CFG_1)

// HC-SR04 Ultrasonic distance sensor driver
const uint8_t sr04_bytecode[] PROGMEM = {
  // bytecode definition
  B_VERSION(1), B_REQUIRES(REQ_GPIO), B_NUM_STACK(0), B_NUM_LABELS(0),
  B_NUM_VARS(0), B_NUM_LOCALS(0), B_NUM_CONFIGS(2), B_NUM_LISTS(0),
  B_NUM_PARAMS(0), B_NUM_FUNCTIONS(1), B_NUM_PROPERTIES(0),

  // boot section
  OP_BOOT,
  HW_GPIO_CONFIG, PIN_ECHO, U8(GPIO_MODE_INPUT),
  HW_GPIO_CONFIG, PIN_TRIGGER, U8(GPIO_MODE_OUTPUT),
  OP_EXIT,

  // function for reading the distance in centimeters
  OP_FUNCTION, FUNC_GET_DATA, PARAMETERS(1),
  OP_DEFINE_OUTPUT_FORMAT, 1, FORMAT_METRIC, SCALE_CENTI,

  // send trigger pulse
  HW_GPIO_PULSE_US, PIN_TRIGGER, U8(1), U8(10),
  // read echo pulse and calculate distance
  SET_ARG(VAR_0, 0),
  MATH_DIV,
  HW_GPIO_PULSE_READ_TIMEOUT, PIN_ECHO, U8(1), U16(23530),
  FLT(58.8235f),

  OP_EXIT,
};

```

Figure 5.6: Driver specification for the HC-SR04 ultrasonic distance sensor. a) shows the bytecode definition, b) the boot function, and c) contains all function definitions.

data. These steps ensure the electronic component is correctly initialized and ready for operation. The boot section is automatically executed.

The next section of the LogicGlue driver specification defines the functions of the component (Figure 5.6c). Each function is described by its name and the format of its input and output parameters, followed by a series of instructions that detail the steps required to perform the function. The function name indicates the type of functionality, such as *light sensing*, *temperature reading*, or *display output*. Parameters are characterized by the format name and a scale, which allows for adjusting the data value according to predefined standards such as metric scales. These details are crucial for LogicGlue’s automated data conversion (section 5.5.2) and, ultimately, allow for swapping between electronic components without rewriting application logic. For instance, if Fahrenheit

is used in the application logic, LogicGlue offers readings in this unit regardless of whether the temperature sensor returns readings as Fahrenheit, Celsius, or Kelvin.

Like function calls in programming, LogicGlue supports multiple alternative implementations of similar functionalities that are differentiated by their input or output parameters. This is useful when, for example, defining driver instructions for the MPU-6050 accelerometer⁸ which has built-in support to output quaternions and Euler angles. To determine the best driver function, LogicGlue prioritizes functions based on the number of required data converters to match the data format used in the application logic.

In addition to functions, the driver specification also details the properties of the electronic component, such as display size, gain, and sensor integration time. Properties are managed similarly to functions but do not require data format conversions for input and output values. Instead, properties are defined by a name and the range of values they accept, which can be a predefined set (e.g., sensor gain) or numeric values (e.g., display size). LogicGlue automatically verifies the compatibility of a given value with a property. For example, when setting the gain of a sensor, LogicGlue supports various generic gain settings of 1x, 2x, 4x, 8x, and so on. However, the TCS34725 color sensor⁹ specifically only supports gain settings of 1x, 4x, 16x, or 64x. By defining the property with a set of accepted values, as illustrated in Figure 5.7, LogicGlue ensures that only compatible gain settings are accepted, maintaining consistent behavior when swapping between different electronic components and showing an error in case incompatible values are used.

Lastly, properties can be defined as static when they return a constant value, providing a straightforward way to access static information about the component. For example, Figure 5.8 shows the definition of the static properties for getting the display size of the SSD1306 OLED display, which will never change and thus can be defined as static.

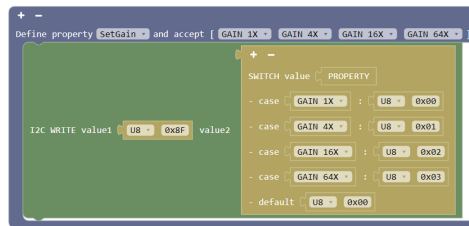
5.4.2 Numeric Instructions

LogicGlue features a versatile subsystem for handling numeric instructions. A numeric instruction is an instruction that evaluates to a numeric value, such as an integer or floating-point number. These instructions can represent constant values, the results of mathematical operations like addition and subtraction, or values received from hardware peripherals like GPIO pins or communication protocols. For each numeric instruction, LogicGlue keeps track of its data type and supports unsigned integers (U8, U16, U32), signed integers (I8, I16, I32), and numbers in both floating (FLT) and fixed (FIX) point formats. Detailed descriptions of all numeric values and their specific semantics are included in Appendix B.2.

⁸ <https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/>

⁹ <https://ams-osram.com/products/sensors/ambient-light-color-spectral-proximity-sensors/ams-tcs34725-color-sensor>

LogicGlue Visual Block-based Interface

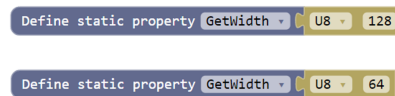


LogicGlue Driver Specification

```
// property for setting the gain of the TCS34725 color sensor
OP_PROPERTY, PROP_SET_GAIN, ACCEPTS(GAIN_1X, GAIN_4X, GAIN_16X, GAIN_64X),
  HW_I2C_WRITE_2,
    U8(TCS34725_COMMAND_BIT | TCS34725_CONTROL),
    NUM_SWITCH, 4, PROP(),
      U8(GAIN_1X), U8(TCS34725_GAIN_1X),
      U8(GAIN_4X), U8(TCS34725_GAIN_4X),
      U8(GAIN_16X), U8(TCS34725_GAIN_16X),
      U8(GAIN_64X), U8(TCS34725_GAIN_64X),
      U8(TCS34725_GAIN_1X),
    OP_EXIT,
```

Figure 5.7: Defining a property for setting the gain of the TCS34725 color sensor.

LogicGlue Visual Block-based Interface



LogicGlue Driver Specification

```
// property for getting the width of the SSD1306 display
OP_PROPERTY_CONST, PROP_GET_WIDTH, U8(128),

// property for getting the height of the SSD1306 display
OP_PROPERTY_CONST, PROP_GET_HEIGHT, U8(64),
```

Figure 5.8: Defining static properties for getting the size of the SSD1306 OLED display.

A key feature of the numeric subsystem is its ability to use numeric instructions as inputs for other numeric instructions, enabling the nesting of operations. This allows for the creation of complex numerical expressions and operations. For example, Figure 5.9 illustrates various numeric operations: a) the sum of two integers, b) multiple nested mathematical operations, c) the state of a GPIO pin, and d) an inline if-test.

It is important to note that since the LogicGlue driver specification is stored as an array of unsigned bytes, signed numbers must be cast, and large integers or floating-point numbers must be split across multiple bytes. LogicGlue provides convenient macros to automatically handle these conversions.

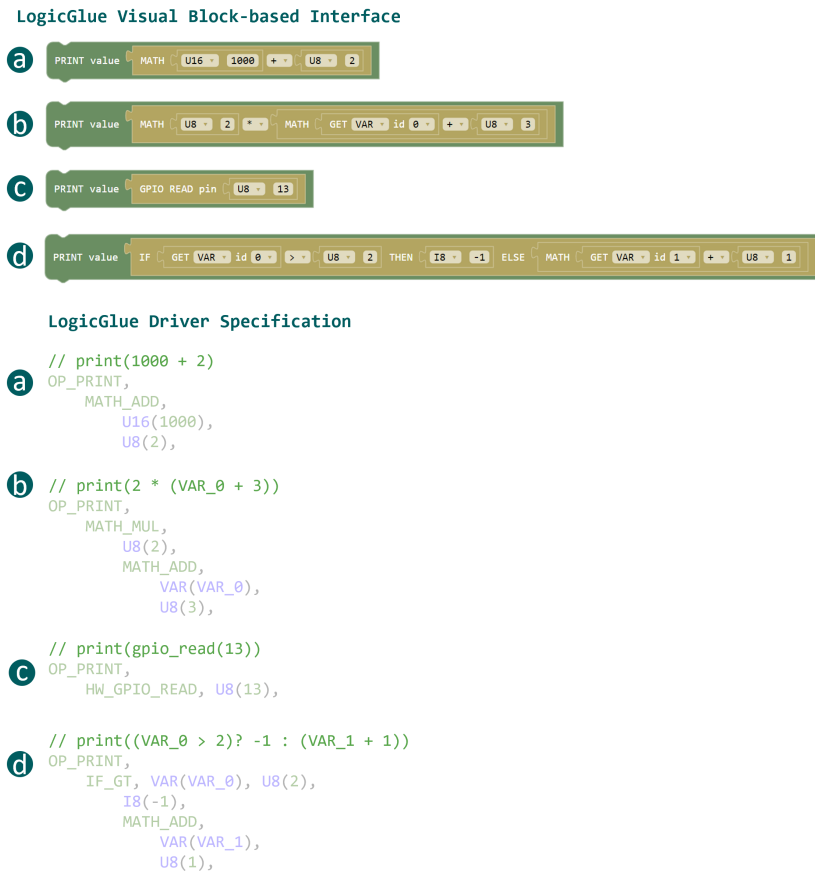


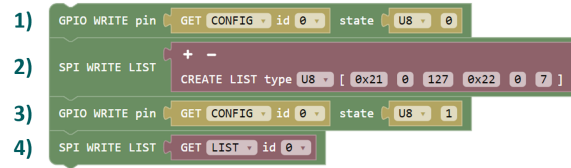
Figure 5.9: Illustration of the numeric subsystem within the LogicGlue driver specification, demonstrating various numeric operations.

5.4.3 List Instructions

List instructions in LogicGlue function similarly to numeric instructions, providing a flexible way to handle arrays of data. Like numeric instructions, LogicGlue maintains the data type of each list to ensure type consistency. Lists are commonly used as data buffers in drivers for displays or as a convenient method for sending multiple bytes over a communication protocol. For example, Figure 5.10 shows an example of the instructions for sending the pixel buffer to the SSD1306 OLED display using a list for the data commands (line 2) and a list for the pixel buffer (line 4). Detailed descriptions of all list items and their specific semantics are included in Appendix B.3.

LogicGlue supports various list types, including integer lists, floating-point lists, and binary arrays. Binary arrays are a special type of array where 8 bits are grouped and stored as a regular byte but can be individually addressed. LogicGlue supports two variants of binary arrays: one where the 8 bits in the x-direction are combined into one byte, and another where the 8 bits in the y-direction are combined. Binary arrays are typically used for single-color displays like the SSD1306 and dot-matrix displays, where each pixel can either be on or off.

LogicGlue Visual Block-based Interface



LogicGlue Driver Specification

```
// enable command mode, prepare data transfer
1) HW_GPIO_WRITE, DC_PIN, U8(PIN_LOW),
2) HW_SPI_WRITE_LIST, LIST_U8(6),
   0x21, // set column address
   0,    // start at 0
   127,  // end at 127
   0x22, // set page address
   0,    // start at 0
   7,    // end at 7

// enable data mode, transfer data
3) HW_GPIO_WRITE, DC_PIN, U8(PIN_HIGH),
4) HW_SPI_WRITE_LIST, LIST(LIST_0),
```

Figure 5.10: Illustration of the list subsystem within the LogicGlue driver specification, demonstrating the instructions for sending the pixel buffer to the SSD1306 OLED display.

When developing a LogicGlue driver for displays like the SSD1306 OLED, it is necessary to define functions that set the color of pixels based on specified x and y coordinates. Typically, this involves updating the pixel color by writing a new value to the internal pixel buffer. While straightforward, this method becomes inefficient when updating a large number of pixels in a loop because each iteration requires evaluating the input values and executing update instructions, leading to significant computational overhead.

To address this inefficiency, LogicGlue introduces specialized function types that execute predefined actions more effectively. For example, the `OP_DEFINE_FUNCTION_TYPE` instruction allows developers to define optimized driver functions, such as `SET_LIST_LOOP`, which streamlines the process of updating multiple values in a list. This function type enables developers to specify a range of indices to be updated at once and to provide either a static value or a callback function that determines the new value for each pixel. By employing an optimized loop within the LogicGlue interpreter, the `SET_LIST_LOOP` function significantly reduces the computational load by minimizing the repetitive evaluation of values. This efficient approach ensures rapid updates while maintaining the capability to automatically convert values as necessary, enhancing the overall performance and responsiveness of the display updates. As demonstrated in Section 5.7, updating the values for the SSD1306 OLED display is as efficient as traditional approaches.

5.4.4 Branching Instructions

LogicGlue offers a range of instructions that facilitate complex control flows, similar to traditional programming languages. For instance, the *IF*, *IF_ELSE*, and *IF_ELIF_ELSE* instructions allow for conditional branching, akin to their counterparts in conventional programming. The *LOOP* instruction requires start, end, and increment values, executing the subsequent instruction multiple times based on these parameters. Additionally, the *FOREACH* and *FOREACH_BYTE* instructions iterate over the items in a list, using each list item or byte, respectively. While *FOREACH* returns a single item, *FOREACH_BYTE* is particularly useful for binary arrays or lists storing 16- or 32-bit values, as it returns each byte separately, regardless of the list type. For example, in Figure 5.11, the driver for the dot-matrix display uses a binary array as a pixel buffer and requires the row address to be sent before each data byte. Using the *FOREACH_BYTE* instruction, this operation becomes more efficient as it allows the driver to iterate through each byte of the binary array, sending the row address and the corresponding data byte in a streamlined manner.

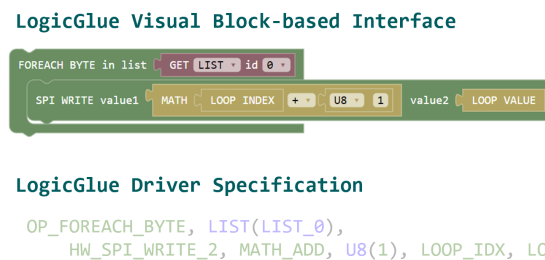


Figure 5.11: Example of the *FOREACH_BYTE* instruction.

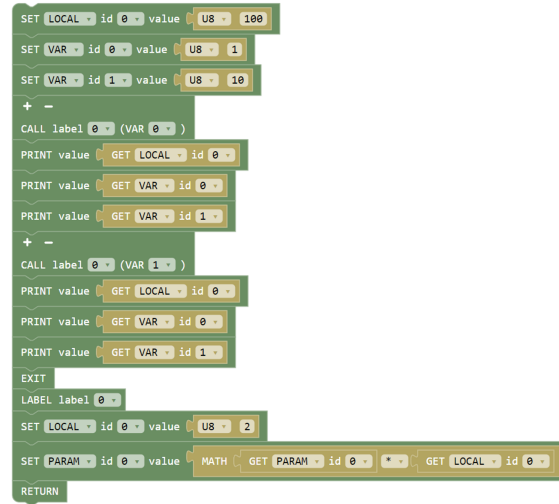
LogicGlue supports various methods for branching the program flow. Labels act as designated jump points within the LogicGlue driver specification, with the *CALL* instruction leveraging these labels to dynamically direct the program counter. The *CALL* instruction functions similarly to traditional programming functions, supporting arguments and creating a separate environment for the function that is called. The *RETURN* instruction reverts the flow of execution back to its original position before the jump, effectively managing the program's execution stack.

In addition to control flow, LogicGlue driver specification supports various types of data storage and manipulation, distinguishing between global and local variable scopes. Global variables are persistent and accessible throughout the entire driver specification, while local variables are temporary and only exist within specific functions, managed via a stack mechanism. This stack-based approach allows for the dynamic allocation and deallocation of local variables as functions are called and returned.

When functions are invoked, arguments are passed by reference, linking them to global variables. This method allows functions to alter different variables by updating the references passed as arguments, enabling reusable functions to operate on varying data

without redundancy. Figure 5.12 illustrates the different scopes of variables within the LogicGlue driver specification.

LogicGlue Visual Block-based Interface



LogicGlue Driver Specification

```
// Create local variable TMP_0, and global variables VAR_0 and VAR_1
SET_LOCAL(TMP_0), U8(100),
SET_VAR(VAR_0), U8(1),
SET_VAR(VAR_1), U8(10),

// Call function with 1 parameter, where parameter #0 = VAR_0
OP_CALL_ARGS, LABEL_0, 1, VAR_0,

OP_PRINT, LOCAL(TMP_0),      // print(100)  --> local context, variable unchanged
OP_PRINT, VAR(VAR_0),        // print(2)   --> VAR_0 is updated by the function
OP_PRINT, VAR(VAR_1),        // print(10)  --> VAR_1 is unchanged

// Call function with 1 parameter, where parameter #0 = VAR_1
OP_CALL_ARGS, LABEL_0, 1, VAR_1,

OP_PRINT, LOCAL(TMP_0),      // print(100)  --> local context, variable unchanged
OP_PRINT, VAR(VAR_0),        // print(2)   --> VAR_0 is unchanged
OP_PRINT, VAR(VAR_1),        // print(20)  --> VAR_1 is updated by the function
OP_EXIT,

//===== Functions =====//
OP_LABEL, LABEL_0,
// Create local variable TMP_0, only accessible in this function
SET_LOCAL(TMP_0), U8(2),

// Multiply parameter #0 by local variable TMP_0
SET_PARAM(0),
MATH_MUL,
PARAM(0),
LOCAL(TMP_0),

OP_RETURN,
```

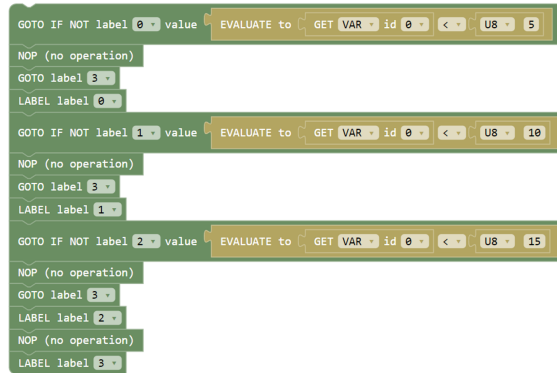
Figure 5.12: Example of the LogicGlue driver specification, demonstrating the scope of variables.

5.4.5 Advanced Instructions

In addition to using labels as jump points for function calls, labels can also be utilized with the *GOTO* instruction, which moves the program counter while maintaining the current context. Beyond the basic *GOTO* instruction, LogicGlue driver specification includes conditional instructions such as *GOTO_IF* and *GOTO_IF_NOT*, which perform jumps only when specified conditions are met. These instructions enable the creation

of more advanced behaviors, such as complex looping mechanisms. Figure 5.13 and Figure 5.14 demonstrate how an if-elif-else test and a for-loop, respectively, can be constructed using a series of labels and *GOTO* instructions.

LogicGlue Visual Block-based Interface

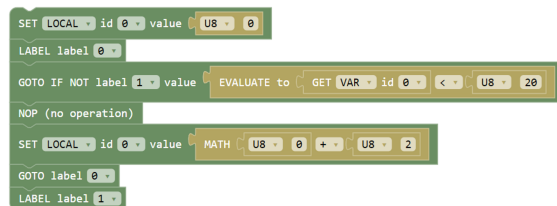


LogicGlue Driver Specification

```
// Example of an if-elif-else test that checks if VAR_ is less than 5, 10 or 15.
OP_GOTO_IF_NOT, LABEL_0, EVAL_LT, VAR(VAR_0), U8(5),
// instructions when VAR_0 is less than 5
OP_GOTO, LABEL_3,
OP_LABEL, LABEL_0,
OP_GOTO_IF_NOT, LABEL_1, EVAL_LT, VAR(VAR_0), U8(10),
// instructions when VAR_0 is less than 10
OP_GOTO, LABEL_3,
OP_LABEL, LABEL_1,
OP_GOTO_IF_NOT, LABEL_2, EVAL_LT, VAR(VAR_0), U8(15),
// instructions when VAR_0 is less than 15
OP_GOTO, LABEL_3,
OP_LABEL, LABEL_2,
// instructions for other conditions)
OP_LABEL, LABEL_3,
```

Figure 5.13: Example of advanced instructions of the LogicGlue driver specification, demonstrating how an if-elif-else test can be created using *GOTO* instructions and labels.

LogicGlue Visual Block-based Interface



LogicGlue Driver Specification

```
// Example of a for-loop, looping from 0 to 20 with increments of 2
SET_LOCAL(VAR_0), U8(0),
OP_LABEL, LABEL_0,
OP_GOTO_IF_NOT, LABEL_1, EVAL_LT, VAR(VAR_0), U8(20),
// instructions inside loop, with VAR_0 the current index
SET_LOCAL(VAR_0), MATH_ADD, VAR(VAR_0), U8(2),
OP_GOTO, LABEL_0,
OP_LABEL, LABEL_1,
```

Figure 5.14: Example of advanced instructions of the LogicGlue driver specification, demonstrating how a for-loop can be created using *GOTO* instructions and labels.

5.5 LogicGlue Interpreter

The LogicGlue interpreter runs on the user's microcontroller and translates the instructions in the driver specification into platform-specific commands. The LogicGlue high-level programming library complements the interpreter and offers an interface for developers to interact with electronic components via the interpreter. Unlike high-level communication standards like Jacdac [Devine, 2022], LogicGlue maintains the native signals and features of the components, avoiding the need for translation to a common protocol and the associated overhead.

5.5.1 LogicGlue High-Level Programming Library

The LogicGlue high-level programming library simplifies interaction with various electronic components by providing functions that serve as wrappers around interpreter calls. These functions handle the initialization, configuration, and operation of components, offering default values for parameters to streamline the process. For instance, Figure 5.15 illustrates a traditional example of how the driver specification is used in a typical application that changes the color of an RGB LED based on the measured distance from an ultrasonic distance sensor. This example highlights the standard use of LogicGlue for most components, demonstrating its effectiveness in managing component interactions efficiently.

```

1  #include <LogicGlue.h>
2  #include <logicglue/drivers/hc-sr04.h>
3  #include <logicglue/drivers/ky-016.h>
4
5  // create the HC-SR04 component connected to pin D8 and D9
6  pinmap_t dist_pinmap = DEFINE_PINMAP(D8, D9);
7  component_t dist_component = DEFINE_COMPONENT(sr04_bytecode, dist_pinmap);
8
9  // create the KY-016 component connected to pin D4, D5 and D6
10 pinmap_t led_pinmap = DEFINE_PINMAP(D4, D5, D6);
11 component_t led_component = DEFINE_COMPONENT(ky016_bytecode, led_pinmap);
12
13 // create a distance sensor and led actuator
14 sensor_t dist_sensor;
15 actuator_t led_actuator;
16
17 void setup() {
18     // initialize the components
19     initialize_component(&dist_component);
20     initialize_component(&led_component);
21
22     // create a new distance sensor for the component
23     create_distance_sensor(&dist_component, &dist_sensor, LENGTH_METRIC, SCALE_CENTI);
24     // create a new led actuator for the component
25     create_led_actuator(&led_component, &led_actuator, COLOR_HSL);
26 }
27
28 void loop() {
29     // get the distance from the sensor and normalize it between 10cm and 110cm
30     float distance = sensor_sample_average(&dist_sensor, 5);
31     distance = min(1.0f, max(0, (distance - 10.0f)) / 100.0f);
32
33     // calculate the hue based on the distance, use color range red (0°) -> blue (240°)
34     uint16_t hue = round(distance * 240.0f);
35     actuator_write_color_hsl(&led_actuator, hue, 100, 20);
36
37     delay(100);
38 }

```

Figure 5.15: Example of the application logic for interacting with an ultrasonic distance sensor and RGB LED.

Initializing components with LogicGlue involves loading the bytecode driver and setting configuration parameters, such as which GPIO pins are connected. This process creates a special environment for the driver to run, managing all the necessary resources, variables, and data buffers for the drivers to work properly. This environment also keeps track of variable states and configurations, ensuring that components operate consistently. This is particularly useful for complex tasks that require multiple steps and need to keep intermediate results. Additionally, the environment can buffer sensor readings, which is helpful for sensors with limited sampling rates, like DHT temperature sensors. In Figure 5.15, lines 19 and 20 show how the distance sensor and LED are initialized.

After initializing the component, LogicGlue provides specific functions to interact with different types of components. The *create_sensor* function sets up sensors so that data can be read in the right format. The *create_actuator* function sets up actuators, allowing data to be sent to them correctly. For displays or other buffer-based components, special functions are available to fill pixel buffers with data. These functions make it easier to work with components by simplifying complex interactions and providing default settings.

In Figure 5.15, lines 23 and 25 demonstrate how to create a distance sensor that measures in centimeters and an LED actuator that accepts colors in HSL format. Using LogicGlue’s high-level functions, interacting with these components becomes straightforward. For example, line 30 shows how the distance sensor is sampled five times to get an average reading. Line 35 shows how the LED color is set based on the calculated HSL hue value from the distance measurement.

Figure 5.16 illustrates an example of application logic for interacting with the SSD1306 display using the optimized LogicGlue functions to write a set of colors to the internal display buffers. In this example, a callback function determines the color of the pixels in the top-left rectangle of the display, while a constant color is applied to the bottom-right. It is important to note that this exact same code can also be used for other types of displays, such as a dot-matrix display, with no modifications needed beyond the preamble (lines 1-6), as demonstrated in Figure 5.3. This underscores the flexibility of LogicGlue in managing various components with minimal to no changes to the application logic.

5.5.2 Converting Data Formats

When using traditional software libraries and drivers to interact with electronic components, developers need to adhere to the data formats outlined by the components. In comparison, LogicGlue allows developers to select their preferred data format for each component in the application logic, independent of the characteristics of the electronic component. For example, in the walkthrough detailed in Section 5.2, temperature readings are specified in Fahrenheit (Figure 5.3, line 15), while the MCP9808 and TMP36 temperature sensors provide readings in Celsius and relative voltages, respectively.


```

1  #include <LogicGlue.h>
2  #include <logicglue/drivers/ssd1306.h>
3
4  // create the SSD1306 component connected to pin SPI and pin D11, D12 and D13
5  pinmap_t pinmap = DEFINE_PINMAP(11, 12, 13);
6  component_t component = DEFINE_COMPONENT(ssd1306_bytecode, pinmap);
7
8  display_t display_buffer;
9  display_t display;
10
11 // callback function that determines the color based on the provided coordinates
12 void calculate_color(uint16_t x, uint16_t y) {
13     // all pixels with an odd x-coordinate will be lit
14     display_buffer_set_color_binary(&display_buffer, x % 2);
15 }
16
17 void setup() {
18     // initialize the component
19     initialize_component(&component);
20
21     // create a display buffer and display object for sending the buffer to the display
22     create_display_buffer(&component, &display_buffer, COLOR_BINARY);
23     create_display(&component, &display);
24
25     // read the width and height of the display
26     uint8_t width = display_get_width(&component);
27     uint8_t height = display_get_height(&component);
28
29
30     // clear all pixels in the display
31     display_buffer_clear(&display_buffer);
32
33     // draw a rectangle in the top-left corner using a variable color
34     display_buffer_fill_rectangle_dynamic(&display_buffer, 0, 0, width/2, height/2, calculate_color);
35
36     // set the color for all next pixel updates (similar to selecting a brush in paint)
37     display_buffer_set_color_binary(&display_buffer, 1);
38     // draw a rectangle in the bottom-right corner, using the set color
39     display_buffer_fill_rectangle(&display_buffer, width/2, height/2, width/2, height/2);
40
41     // write the pixel buffer to the display
42     display_update(&display);
43 }

```

Figure 5.16: Example of the application logic for interacting with the SSD1306 OLED display.

To facilitate seamless data conversions, the LogicGlue interpreter automatically applies a series of built-in data converter functions. These functions, all written in bytecode, convert between the data format provided by the application logic and the data format specified in the driver specification. Rather than adopting the impractical approach of including a separate converter function for every possible data format—which would be infeasible given the memory constraints on prototyping platforms—LogicGlue leverages two complementary approaches to reduce the number of required data converters:

1. Converting Scalable Formats:

Scalable data formats use metric prefixes like centi-, milli-, and kilo- to adjust measurement units. Instead of separate converters for each unit pair, LogicGlue uses a generic scale converter that calculates the conversion factor based on these prefixes. This factor is derived from the difference in metric scale steps, each representing a power of 10. For example, converting from milli- to deci- involves moving two steps to the right on the scale, resulting in a conversion factor of $10^2 = 100$. For imperial measurements, LogicGlue uses predefined ratios relative to a base unit (foot). The conversion factor is found by dividing the 'from' ratio by the 'to' ratio. For example, converting inches to miles uses the ratios $\frac{1}{12}$ and 5280, respectively, giving a factor of $\frac{1}{12}/5280 = 0.00001578$. When converting between

metric and imperial formats, LogicGlue first converts to the base units before applying the appropriate conversion.

2. Converter Chaining:

LogicGlue automatically chains a set of converters in case no single converter is available. For example, if there are converters available for HSL colors to RGB colors and RGB colors to CMYK colors, LogicGlue automatically chains these to convert HSL colors to CMYK colors. Finding a compatible converter chain is managed by representing all data formats as a graph, where each node represents a specific data format, and each edge represents a converter that can transform data from one format to another. To find the most efficient conversion path between two data formats, LogicGlue employs a breadth-first search (BFS) algorithm. Figure 5.17 shows an example of a graph of data formats and available converters.

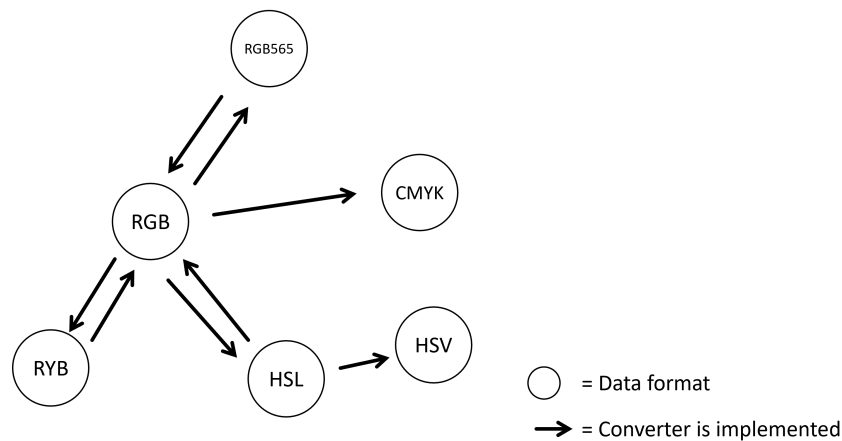


Figure 5.17: Subset of the graph with data formats and their converters. This figure shows data formats for color representations.

As LogicGlue uses bytecode to represent drivers, the compiler cannot automatically determine which data converters are necessary. Consequently, all available converters are included in the microcontroller’s embedded code by default, resulting in significant memory overhead. To optimize memory usage, LogicGlue employs C preprocessor definitions in the driver header files to selectively enable only the relevant converters during the compilation process, ensuring that unnecessary converters are excluded from the final code.

5.6 Supporting LogicGlue on a new Platform

The implementation of the LogicGlue interpreter consists of two parts: (a) An implementation, in C, for parsing the platform-independent driver specifications (bytecodes) and converting data formats. (b) A platform-specific implementation for communication protocols, GPIO pin access, and memory allocation. This dual architecture allows for

convenient porting of the LogicGlue interpreter to different microcontrollers. Porting LogicGlue to a platform that uses the C programming language only requires implementing the platform-specific functions, which are detailed in Appendix C.1. Since most microcontroller platforms support C and C++ programming, LogicGlue can be easily implemented on a broad variety of microcontrollers. We already have support for the Arduino and nRF52 platforms.

Supporting Logicglue is more complex for platforms that run on programming languages that do not build on the C language, such as CircuitPython or DeviceScript. Applications written in these languages are compiled into custom binaries and interpreted on the microcontroller. In these situations, developers must make a one-time effort to fully reimplement both the LogicGlue library and the Logicglue interpreter in this programming language. Alternatively, LogicGlue could be integrated into the runtime or the language’s SDK that executes the custom binaries, as these are typically written in C or C++.

5.7 LogicGlue Benchmark

The introduction of software abstraction layers typically introduces overhead. In LogicGlue, however, these abstraction layers do not significantly impact the performance as LogicGlue enables communication between the application logic and electronic components using the standard protocols and communication signals supported by the component. This is different from high-level communication protocols like Jaccadac [Devine, 2022], which requires the translation of every communication signal. In this section, we benchmark the performance of LogicGlue and demonstrate the LogicGlue software stack, including the interpreter and programming libraries, does not significantly impact the performance of interacting with electronic components compared to using component-specific libraries.

To benchmark our system, we measured the execution times for interacting with electronic components using component-specific libraries (baseline condition) and using LogicGlue. We primarily focus on the primary functions of reading and writing data to and from electronic components. Benchmarking initialization procedures is not considered as this typically is a short one-time process and thus has limited impact on the performance. Although LogicGlue embeds many data converters, we did not use this automated conversion of data formats to ensure a fair performance comparison with the baseline condition, as component-specific libraries do not support such features.

To benchmark the performance of both basic and advanced interactions, we selected three different sensors: an RGB LED controlled via three PWM pins (basic), a DHT22 temperature sensor (intermediate), and an SSD1306 display operating over SPI (advanced). Our tests were conducted on an Arduino Mega microcontroller running the Arduino platform. In the baseline condition, we interact with the RGB LED using Arduino’s

“AnalogWrite” function. For the DHT22 temperature sensor, we use the Adafruit DHT sensor library¹⁰, and for the SSD1306 display, we use the Adafruit SSD1306 library¹¹.

For a precise measurement of execution times, we connected a logic analyzer (Saleae Logic Pro 8) to an additional GPIO pin and pulled it to VCC and ground at, respectively, the start and end of the communication. The logic analyzer measures the time this GPIO pin is high, which represents the duration of the communication. This measurement technique is common as it allows for precise and reliable measurements of execution times on a microcontroller. Each test was repeated 100 times.

Electronic Component	Component-Specific Library	LogicGlue	Relative Difference
RGB LED	0.032ms	0.380ms	+1087.50%
DHT22 Temperature Sensor	4.256ms	4.208ms	-1.13%
SSD1306 Display	5.444ms	8.604ms	58.05%

Table 5.1: Execution times for interacting with electronic components using component-specific libraries versus LogicGlue.

The results are summarized in Table 5.1 and show the median execution time for the baseline and LogicGlue condition. All execution times are within ± 0.002 milliseconds, underscoring the consistency of the measurements in both conditions. Although the execution time for triggering the RGB LED has increased significantly with LogicGlue, it remains under 1 millisecond, which is still very fast and negligible in most practical applications. For context, the refresh rate of a standard 60fps display is about 16 milliseconds per frame, making the slight increase in LED triggering time negligible. For the DHT22 temperature sensor and the SSD1306 display, the difference in execution time with LogicGlue is minimal. The temperature sensor shows a slight decrease in execution time, and while the display update time has increased, the overall performance remains efficient. This demonstrates that LogicGlue introduces very little overhead, maintaining high performance and efficiency across different types of electronic components.

5.8 Discussion and Future Work

LogicGlue’s platform-independent drivers allow for writing hardware-independent application logic. This significantly simplifies integrating electronic components, as one does not need to consider technical characteristics, such as protocols and registers. As a result, LogicGlue also facilitates transitions between different microcontrollers and electronic components, which fosters experimentation and iterative development. As such, developers can easily swap components without extensive modifications to the application logic. However, while LogicGlue automatically handles all data format

¹⁰<https://github.com/adafruit/DHT-sensor-library>

¹¹https://github.com/adafruit/Adafruit_SSD1306

conversions, developers must still understand the functionalities and limitations of both the original and replacement components. For instance, swapping a high-precision I2C temperature sensor with a low-precision analog sensor will result in correct data format conversions but may impact the workings of the application due to the inherent differences in precision.

Display components present another example where careful consideration is needed. Swapping an RGB display with another of a different size requires that the visual interface scales correctly. Replacing an RGB display with a single-color display necessitates adjustments to accommodate the lack of color, which LogicGlue converts but does not adapt in terms of design. For example, while LogicGlue automatically converts the RGB color to a binary color, it can not adjust the interface to accommodate the lack of color. Similarly, transitioning from a high-resolution OLED display to a low-resolution dot-matrix display will require application-level modifications to handle the reduced resolution appropriately. Likewise, replacing, for example, a stepper motor with a DC motor involves recognizing the differences in control mechanisms and precision. A stepper motor offers precise position control, which is essential for applications like 3D printing, whereas a DC motor provides continuous rotation but lacks the same level of positional accuracy, making it suitable for applications like driving an RC car or a fan.

While software abstraction layers, such as the one offered in LogicGlue, typically introduce additional latency, LogicGlue enables the application logic to interact with electronic components using their native signals and protocols. As demonstrated in the benchmark (Section 5.7), this ensures the performance of our approach is similar to using the component-specific libraries. For the majority of real-time processing applications, the minor increase in latency, as reported in the benchmark, is neglectable. However, in scenarios in which components need to operate in sync, such as some robotic applications, the minor delay introduced by processing bytecodes might be problematic, and engineers might want to swap back to native code. Alternative approaches, such as the Jacdac, operate through an ecosystem of compatible modules that use the Jacdac protocol via standardized services. To provide this compatibility, each module in the Jacdac ecosystem contains a microcontroller that translates signals, specific to the component on that module, to the Jacdac protocol. This approach requires an additional microcontroller for every component, and the translation introduces overhead. In addition, the asynchronous nature of the Jacdac data bus can lead to additional delays and the loss of component-specific functionalities that are not covered by the Jacdac protocol.

While we benchmarked the performance of LogicGlue, our approach also introduces additional memory consumption to store the bytecode, the interpreter, and programming libraries. Each of these, in turn, introduces additional runtime memory usage. Although highly dependent on compiler settings and optimizations, we measured that LogicGlue requires around 35kb of flash memory when compiled for Arduino and 55kb for

the nRF52840 microcontroller. While memory availability on microcontrollers and development boards increases every year, the additional memory consumption of LogicGlue can be an issue for embedded systems with scarce memory resources. To mitigate this issue, future work can look into strategies for optimizing memory usage by refining the bytecode and interpreter. For instance, simplifying the data format converters within LogicGlue could reduce their memory demands. Alternatively, considering the feasibility of offloading certain processing tasks to external devices like a connected computer or leveraging cloud computing resources could help conserve the microcontroller's memory.

Logicglue eases the work of component manufacturers and engineers of development platforms as electronic components for which LogicGlue driver specifications are written, are instantly compatible with all development platforms that support LogicGlue. Likewise, new development platforms that implement LogicGlue are instantly compatible with the wide variety of electronic components that are on the market today. While LogicGlue requires a one-time effort for a component manufacturer or engineer to write LogicGlue driver specifications for new electronic components, artificial intelligence could be used in the future to automatically generate driver specifications from a component's datasheet.

LogicGlue's driver specifications are stored in bytecode format. They are thus very compact and can be stored in the cloud or on a developer's computer. As these driver specifications include all information to interface with the component, we believe it also makes sense to store this bytecode in the future on a memory chip located on every component. As this would require component manufacturers to follow a new standard, a similar, more practical implementation is the use of a shield that extends any electronic component with a memory chip, as shown in Figure 5.18. This memory chip could communicate with the LogicGlue Interpreter over I2C or 1-wire, making it possible to automatically recognize plugged-in components and load their bytecode driver specifications. We also envision the memory chip hosting additional information, such as the pinout and operating voltage. This enables new opportunities to assist the wiring process. To further ease or avoid the component wiring, we see opportunities for synergies between our research efforts on lowering the barrier for embedded programming and state-of-the-art solutions to avoid or facilitate component wiring, such as CircuitGlue [Lambrichts, 2023] or VirtualWire [Lee, 2021]. These synergies could lead to novel solutions in which any electronic component becomes plug-and-play, similar to the use of USB to simplify device connectivity.

Lastly, although this chapter does not include a formal user study, it is significant to acknowledge the importance of validating the usability and effectiveness of LogicGlue from an end-user perspective. The block-based interface, for example, is designed to make driver creation more accessible to users without deep knowledge of specific microcontroller architectures. However, the actual impact of this interface on user

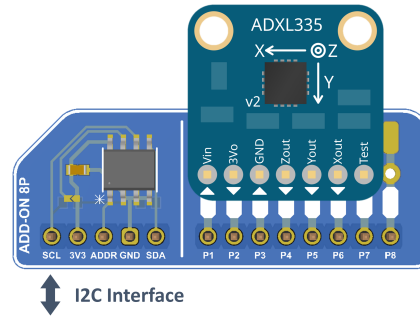


Figure 5.18: A conceptual illustration of an extension shield equipped with a memory chip allows for the embedding of a component’s driver, ensuring automatic recognition and configuration by LogicGlue upon connection.

performance, error rates, and learning curves has not been empirically validated. Future research should include user studies that focus on these aspects, collecting data on how both novice users and professional developers interact with LogicGlue. Such studies could involve tasks that measure the time needed to develop drivers and the efficiency of the created drivers.

5.9 Summary

In this chapter, we introduce LogicGlue, a novel framework that streamlines electronics prototyping by creating platform-agnostic drivers for hardware components and enabling the development of hardware-independent application logic. At the heart of LogicGlue lies a custom driver specification format and interpreter, which enables the definition of a component’s functionalities, regardless of platform types or programming languages. Furthermore, we provide a visual block-based programming interface that simplifies writing these specifications, making it more accessible to a broader range of individuals.

LogicGlue addresses the challenges outlined in research question **Q3** by providing a universal driver specification that mitigates compatibility issues between different software environments and hardware platforms. This solution aligns with research goal **G3** by promoting easier software integration and ensuring that the full functionality of hardware components remains accessible across various platforms. The technical evaluation answers research question **Q2** by showing that LogicGlue introduces minimal latency.

While this chapter does not include a formal user study to evaluate LogicGlue’s usability, its design was directly informed by the practical challenges identified during the development of CircuitGlue. LogicGlue exemplifies our commitment to making electronics prototyping more efficient and user-friendly, ultimately contributing to the democratization of technology creation by providing tools that are accessible to users of all skill levels.

ON-GOING RESEARCH INTO UNIFIED PLUG-AND-PLAY PROGRAMMING

Motivation

Building on the developments of CircuitGlue and LogicGlue, discussed in Chapters 4 and 5, we identified a unique opportunity to further enhance the electronics prototyping process by integrating the advantages of both hardware and software solutions. CircuitGlue addresses the challenges of hardware compatibility by offering a flexible platform for connecting heterogeneous electronic components. LogicGlue, on the other hand, extends these principles into the software domain, enabling the development of platform-independent drivers and facilitating hardware-independent application logic. However, while each of these solutions effectively addresses its respective domain, the potential for a unified approach that seamlessly integrates both hardware and software aspects remains unexplored.

This realization led to the ongoing research into a unified plug-and-play platform, which we call UniGlue. The motivation behind UniGlue is to create a comprehensive system that leverages the strengths and learnings from both CircuitGlue and LogicGlue to provide an even more streamlined and accessible prototyping experience. By combining hardware flexibility with software adaptability, UniGlue aims to simplify the prototyping process, making it easier for users of all skill levels to develop, test, and iterate their electronic projects.

UniGlue addresses the holistic needs of electronics prototyping by ensuring that both the hardware and software components work together seamlessly. This unified approach aligns with research question Q3, which seeks to explore strategies for overcoming compatibility issues between hardware and software components. UniGlue is designed to eliminate the boundaries that typically separate hardware integration from software development, thereby enhancing the overall user experience and reducing the complexity of managing disparate systems.

UniGlue addresses research goals **G2** and **G3** by bridging the gaps between hardware and software interactions in a cohesive manner. By integrating the flexible hardware connectivity of CircuitGlue with the platform-independent driver capabilities of LogicGlue, UniGlue enables a more versatile prototyping environment that is not limited by specific hardware configurations or software platforms. This integration is crucial for creating a truly plug-and-play experience, where users can focus on innovation and creativity rather than technical constraints.

While the core of the technical development of UniGlue has been completed, the concept has not been evaluated by users. This chapter outlines the technical work that has been done and presents it as a framework for ongoing research. The current progress represents the initial steps toward achieving a fully integrated platform, and future work will involve refining the technical implementation based on user feedback and conducting comprehensive evaluations to validate its usefulness. This ongoing research into unified plug-and-play programming sets the stage for a future where electronics prototyping is more accessible, efficient, and inclusive, ultimately contributing to the democratization of technology creation.

6.1 Introduction

The field of electronics prototyping is continually evolving, driven by the need to make the design and creation of electronic devices more accessible to a broader audience. As new technologies and components emerge, there is a growing demand for tools that simplify the integration of various hardware and software elements, serving as the *glue* that binds these systems together, especially for those new to electronics. This chapter addresses these challenges by introducing a unified approach to prototyping that builds on existing solutions, providing a seamless way to connect both hardware and software components.

In chapters 4 and 5, we discussed CircuitGlue and LogicGlue, two systems developed to address specific issues in the prototyping process. CircuitGlue was designed to simplify hardware integration, allowing users to easily connect different electronic components without worrying about compatibility issues such as voltage levels or communication protocols. LogicGlue, on the other hand, focused on the software side, offering a platform-independent framework for driver development to make it easier to write code that works across different microcontrollers.

Building on the strengths of these two systems, this chapter introduces UniGlue, a platform designed to combine the hardware integration capabilities of CircuitGlue with the software compatibility features of LogicGlue. UniGlue aims to create a seamless plug-and-play experience that simplifies both the physical assembly of electronic components and the software development process. By doing so, it seeks to lower the barriers to entry for novices, who often face significant challenges when working with unfamiliar

hardware and software.

The development of UniGlue is motivated by the recognition that many users, particularly novices, struggle with the complexities inherent in configuring hardware and writing software that works across different components and systems. By offering a solution that bridges the gap between hardware assembly and software development, UniGlue aims to lower the barriers to entry in electronics prototyping, making it more accessible to a broader audience.

UniGlue’s approach is centered on creating an open and flexible platform that can support a wide range of electronic components and microcontrollers. Unlike existing systems that often restrict users to specific ecosystems, UniGlue is designed to be adaptable, allowing for the integration of new electronic components as they emerge. This flexibility is key to ensuring that the platform remains relevant and useful in a rapidly evolving technological landscape. Furthermore, UniGlue allows direct interaction with components using their native communication standards, such as digital and analog GPIO pins, or protocols like I2C and SPI, reducing latency and preserving all unique features of an electronic component. This stands in contrast to existing solutions like Jacdac [Devine, 2022] and ROS [ROS, 2024], which translate components into a universal communication standard, potentially losing component-specific functionalities.

Furthermore, UniGlue offers a true plug-and-play experience with the addition of an extension shield that permanently attaches to an electronic component. This extension shield, conceptualized in Chapter 5, contains the technical specifications, image, and driver in LogicGlue bytecode format. Once plugged in, the extension shield enables automatic identification and configuration of the electronic component in both hardware and software. This ensures direct and seamless integration, enhancing efficiency, reducing development time, and making the prototyping process more accessible to a broader audience. The UniGlue interface shows plugged-in components, their information, and code examples demonstrating how to interact with the component. In addition, the interface allows users to manually select and configure components akin to the interface used by CircuitGlue in scenarios where an extension shield is not available or wanted.

While UniGlue represents a significant step forward in simplifying the prototyping process, it is still under development, with ongoing efforts to refine its features and expand its compatibility with additional components. Future work will include evaluations to better understand its impact on novice users and to explore further enhancements that could make the platform even more accessible and effective. This chapter delves into the design and implementation of UniGlue, exploring how it builds on the foundations laid by CircuitGlue and LogicGlue. It discusses the potential applications of UniGlue and outlines the future directions for this ongoing research, with the goal of further democratizing electronics prototyping.

6.2 UniGlue: Bridging Hardware and Software for Unified Prototyping

The complexity of electronics prototyping often poses significant challenges, especially when it comes to integrating diverse components and ensuring compatibility between hardware and software. UniGlue aims to address these challenges, building on the strengths of both CircuitGlue and LogicGlue to provide a unified, plug-and-play prototyping experience.

By leveraging the platform-independent drivers introduced by LogicGlue, UniGlue provides a unified interface that simplifies the complexities of interfacing with various components. The hardware design of UniGlue is built upon the redesigned version of CircuitGlue, which was discussed in Section 4.11, and inherits all functionalities, such as the custom diagram generator.

UniGlue consists of two types of boards: a controller board and a logic board. The controller board (Figure 6.1a) is responsible for managing the overall system, while a logic board (Figure 6.1b) handles the assignment of the programmable header pins. Each logic board is equipped with eight programmable header pins, which can be divided into two sets of four using a splitter board (Figure 6.1c). This modular design allows for greater flexibility in configuring different components. Compared to CircuitGlue, each logic board in UniGlue includes its own voltage regulators for the 3.3V and 5V lines, along with two programmable voltage regulators—one for each set of four programmable pins. This design allows for more precise voltage control and eliminates voltage restrictions when driving multiple components with different voltages. Additionally, multiple logic boards can be chained together to drive multiple components simultaneously. The number of logic boards that can be chained is only constrained by electrical limitations, such as voltage degradation of signal noise.

Visual feedback through a display and RGB LEDs on the logic board improves user interaction and troubleshooting. These visual indicators provide real-time information on the status of each component and the overall system, making it easier to identify and resolve issues. This intuitive feedback mechanism enhances the user experience, making the prototyping process more straightforward and user-friendly. Figure 6.2 shows an example setup of UniGlue, where an OLED display is being controlled.

6.2.1 Shared Resource Bus

Similar to CircuitGlue, UniGlue acts as an intermediary between electronic components and the user's microcontroller, facilitating seamless communication and integration. Unlike CircuitGlue, which translates all component interactions into a universal protocol, UniGlue allows the user's application to interact with components using their native protocols. This flexibility is enabled by the combination of the driver specification format from LogicGlue and the innovative use of a shared resource bus—a concept

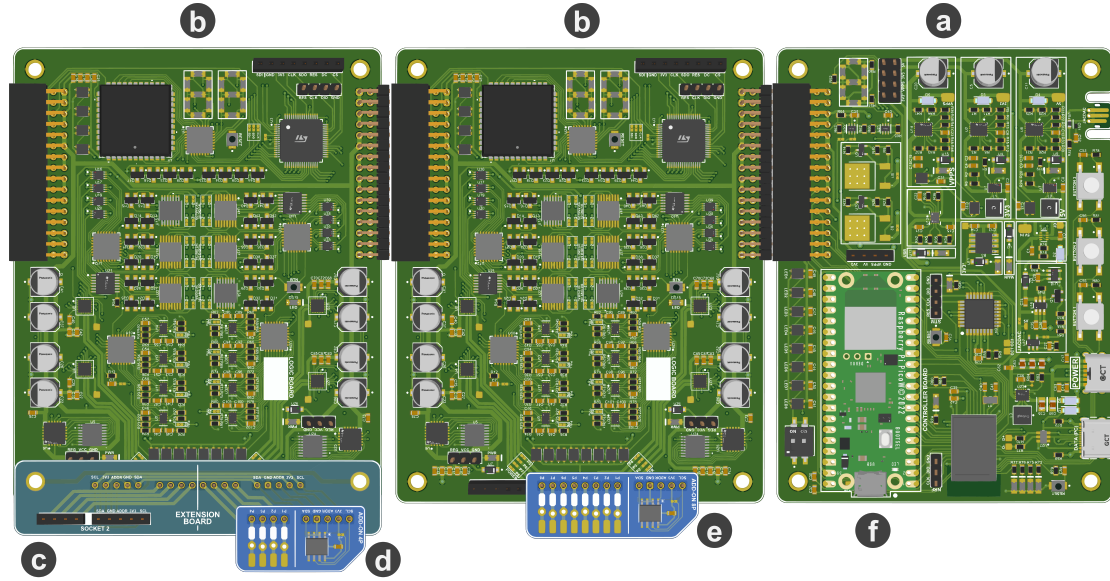


Figure 6.1: Components of the UniGlue system, with a) the UniGlue controller board, b) the UniGlue logic boards, c) the splitter separating the programmable header of the logic board, d) 4-pin UniGlue extension shield, e) 8-pin UniGlue extension shield, and f) the user's microcontroller.

inspired by the Peripheral Component Interconnect (PCI) lines in traditional computers. The shared resource bus consists of multiple parallel data lines that can be dynamically assigned based on the communication requirements of the connected components. This dynamic assignment allows the bus to adapt to various protocols and data streams, ensuring that the full range of the microcontroller's features can be leveraged. As a result, UniGlue can efficiently accommodate various communication protocols, such as I2C, SPI, UART, and others, depending on the project's needs. The current version of UniGlue includes 16 bus lines, and each logic board makes use of a crosspoint switch to connect programmable header pins to one or more bus lines.

6.2.2 Communication Bus

The UniGlue communication bus links all logic boards, controller board, UniGlue interface and user's microcontroller and enables the interchange of data. In comparison to CircuitGlue, which uses the I2C protocol for back-end communication between the controller and logic boards, UniGlue employs the RS485 standard with a custom multi-master communication protocol. While I2C is a common choice for interfacing with sensors and actuators due to its simplicity and effectiveness for small data transfers, it is less suitable for transmitting large amounts of data. In our experience, when using I2C for transmitting extensive data, such as bytecode drivers and component specifications, we encountered frequent issues with devices "hanging" the I2C bus. Moreover, I2C traditionally operates with a single-master setup, where one central device controls the communication by polling each connected device for updates. This setup requires complex management, especially as the number of connected devices

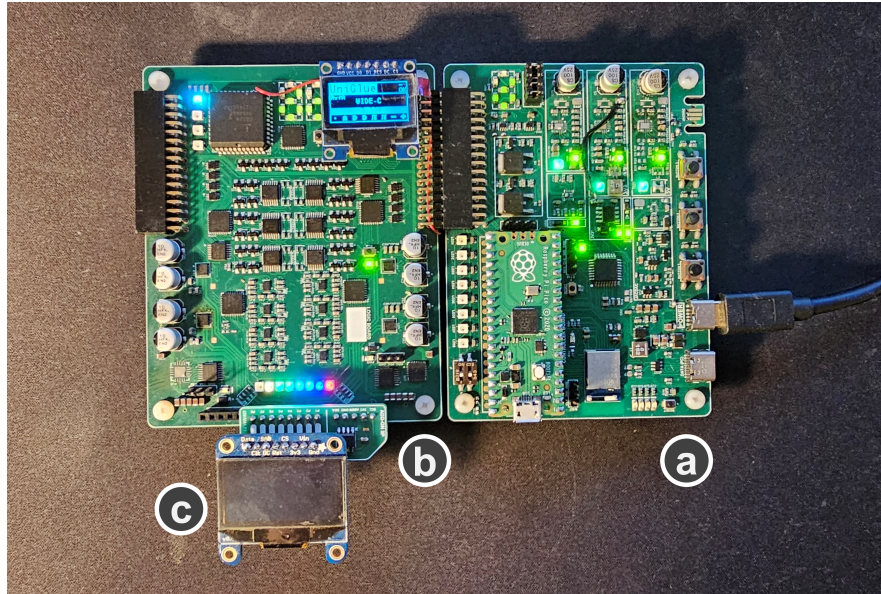


Figure 6.2: Example of the UniGlue setup, with a) the UniGlue controller board, b) the UniGlue logic board, and c) an OLED display connected to the UniGlue extension shield.

increases, which can lead to significant synchronization challenges and inefficiencies.

UniGlue addresses these limitations by implementing a custom multi-master communication protocol over the RS485 standard, illustrated in Figure 6.3. This protocol allows multiple devices to communicate on the same bus more effectively. It functions like a notification system, where each device sends an update whenever a significant event occurs, such as a component being plugged in or a logic board being connected. Each device in the UniGlue system transmits data using a predefined set of event codes, indicating the type of data that is transmitted. These codes enable devices to listen only for relevant messages, reducing unnecessary communication. For example, the LogicGlue interface is programmed to respond to events indicating that a new component has been connected, while other logic boards can ignore these messages. The controller board bridges the RS485 protocol with the WebUSB connection of the UniGlue Interface, allowing data to be visualized. In addition to sending notifications, the communication bus is also used to log messages from each device, providing a valuable tool for debugging.

The UniGlue system incorporates a custom method for managing bus access to prevent simultaneous data transmission by multiple devices, which could lead to data collisions. Alongside the standard TX (transmit) and RX (receive) lines used for data communication, UniGlue introduces a third control line specifically for managing access to the bus.

Figure 6.4 demonstrates how this control line operates when two devices attempt to transmit data simultaneously. Before initiating data transmission, each device checks the state of the control line. If the line is high, it indicates that the bus is free, and the device can take control by pulling the line low. However, if multiple devices check the

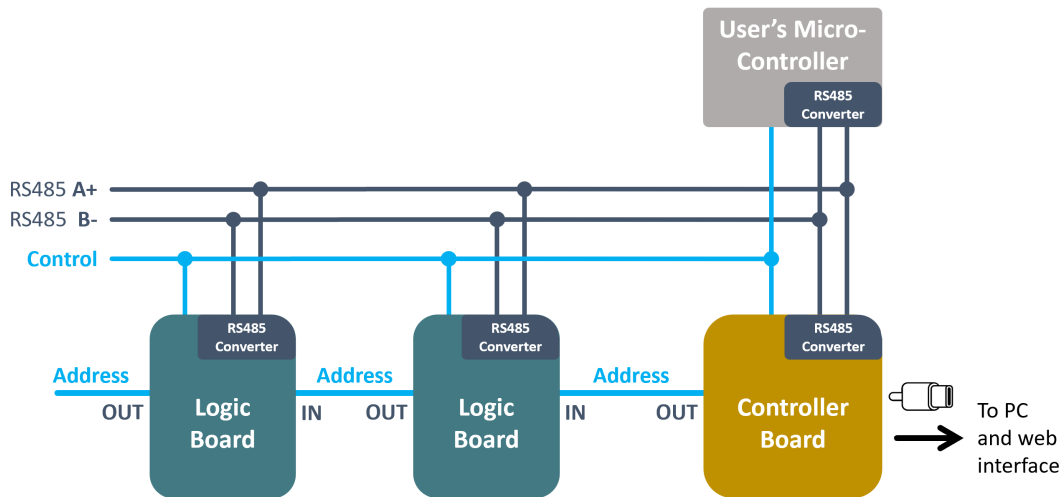


Figure 6.3: Block illustration of the RS485 communication bus in the UniGlue system used for back-end data exchange between all boards.

control line at exactly the same time, they all see that it is high, and all attempt to take control simultaneously, causing a data collision.

To prevent such collisions, UniGlue implements a unique timing mechanism. Each device pulls the control line low for a predefined duration, which is calculated using the device's unique address and thus varies slightly between devices. After this initial action, the device checks the state of the control line again. If the line is still low, it indicates that another device has taken control, and the device will wait before attempting to transmit again. If the line is high, the device knows it has exclusive control and can proceed with data transmission.

For instance, in Figure 6.4(a), two devices check the control line simultaneously and both pull it low, believing the bus is available. However, as illustrated in Figure 6.4(b), device 1 has a shorter pulse duration. When device 1 checks the control line again and finds it still low, it recognizes that another device (device 2) has taken control. Device 1 then waits for the next opportunity to transmit. Conversely, in Figure 6.4(c), device 2 checks the line after its pulse duration and finds the line high, confirming that it now has control. Device 2 then keeps the line low while transmitting data and releases the control line after completing the transmission, as shown in Figure 6.4(d).

This staggered timing mechanism effectively prevents data collisions by ensuring that only one device gains control of the bus at a time, enhancing the reliability and efficiency of the UniGlue back-end communication system.

Additionally, UniGlue automates the assignment of device addresses through a dynamic addressing mechanism that avoids the need for hard-coded addresses. This is achieved using a fourth line that connects devices in a daisy-chained topology. When a new device is connected, it signals its presence by pulling the upstream pin low, indicating

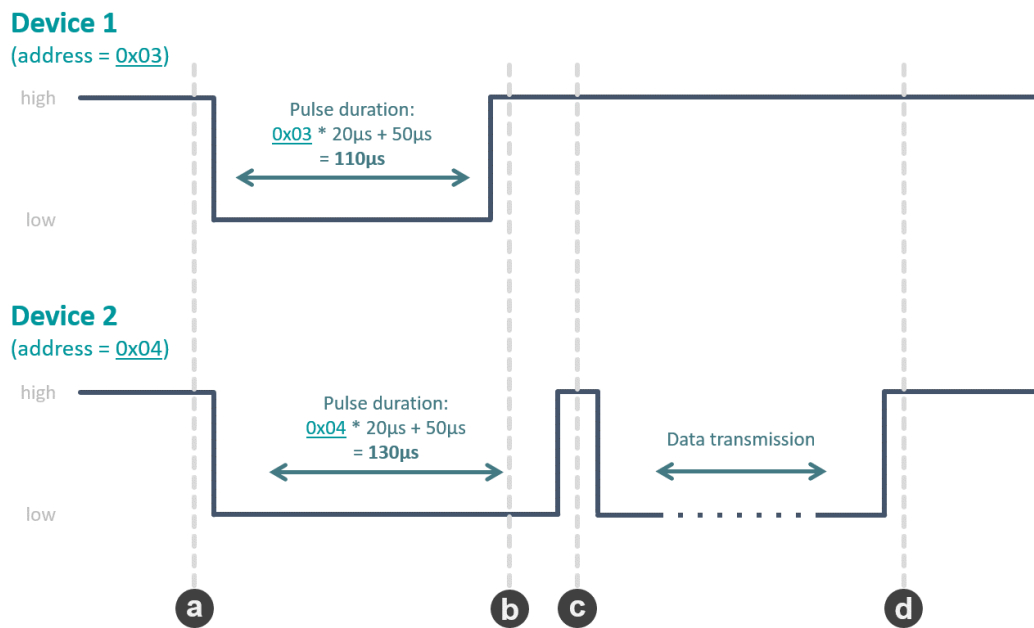


Figure 6.4: Timing sequence of the UniGlue control line to manage bus access. (a) Two devices simultaneously attempt to access the bus by pulling the control line low. (b) Device 1, with a shorter pulse duration, checks the control line and finds it still low, indicating that another device has taken control. (c) Device 2, having a longer pulse duration, checks the control line and finds it high, confirming that it now has control of the bus. (d) Device 2 completes data transmission and releases the control line by returning it to a high state, making the bus available for other devices.

readiness to receive an address. The upstream device, if it has a valid address itself, will then broadcast a message with its own address plus one. The new device monitors the upstream pin, and if it remains low after receiving the message, it accepts the new address. If the pin goes high, indicating the message is for another device, the new device waits and repeats the process with the next address notification.

This combination of a robust communication protocol and dynamic addressing allows UniGlue to handle large data transmissions and coordinate multiple devices more efficiently than systems relying on the traditional I2C protocol.

6.2.3 UniGlue Interface

The UniGlue interface (Figure 6.5) offers a comprehensive overview of all connected components, displaying their status and facilitating easy interaction. When a component is connected, the interface automatically detects it, visually represents the connection, and assigns a unique name for easy reference in application logic. Users can quickly browse the functionalities provided by each component and access customizable examples that demonstrate how to use specific features (Figure 6.6). For advanced users, the interface includes a Developer Mode that allows them to reprogram an extension shield with a new driver, offering greater flexibility for customization.

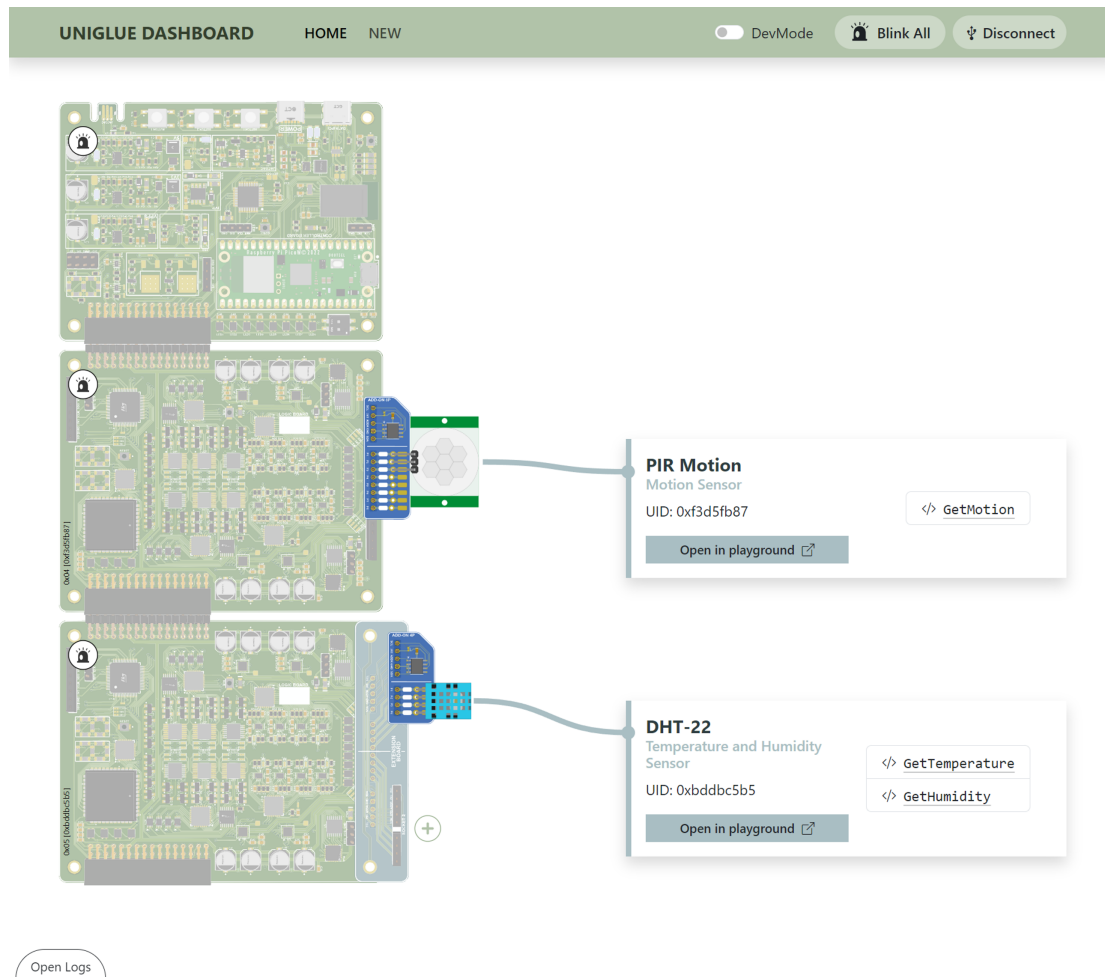


Figure 6.5: The UniGlue interface for displaying and interacting with connected electronic components. The logic board on the bottom uses a splitter board to divide the programmable header into two.

UniGlue supports three methods for configuring components, providing flexibility to suit different user preferences and needs.

- **Automatic Configuration with Extension Shields:**

When components are equipped with an extension shield, they are automatically detected and configured by UniGlue without any user interaction. The memory shield stores the component's specifications, image, and driver, allowing for offline use. This method enables a full plug-and-play prototyping experience.

- **Manual Configuration via the Interface:**

Users also have the option to manually specify a component in the UniGlue interface, similar to the approach used in CircuitGlue. Within the interface, users can browse through the online database of available components (Figure 6.7a).

- **Custom Driver Upload:**

For scenarios that require testing multiple driver variants or using custom drivers, users can upload a driver file directly into the interface (Figure 6.7b). This feature supports experimentation and customization, allowing users to tailor the system to their specific needs.

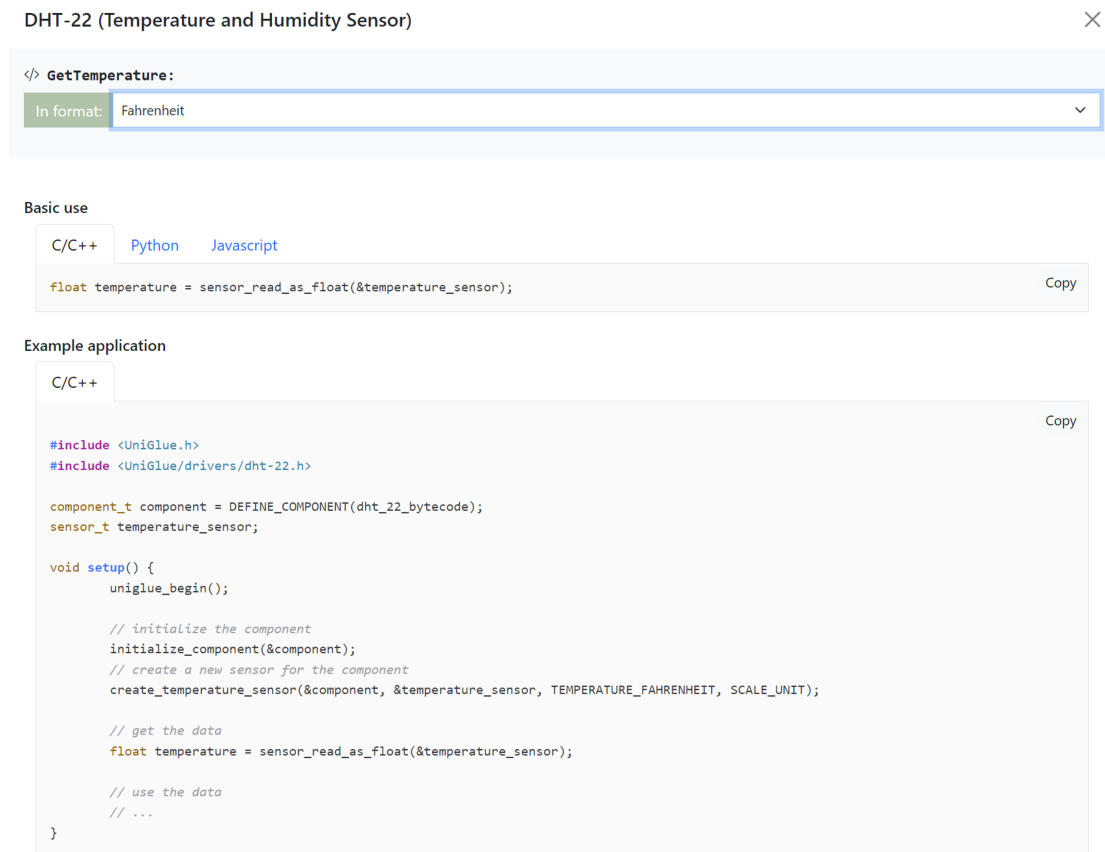


Figure 6.6: Popup showing an interactive example for the plugged-in DHT-22 temperature and humidity sensor. In this example, users can specify the desired output format for the temperature, and copy the example code directly into their application.

Figure 6.8 illustrates a scenario where a logic board has two connected components: (a) is detected automatically, and (b) is manually configured.

Additionally, the UniGlue interface provides a convenient method for creating new drivers. It includes a series of input fields for entering all relevant data, and utilizes the LogicGlue Blockly Interface to help users create driver code. Once the driver is complete, users can upload it to the online database or save it locally as a file for future use. Figure 6.9 shows this interface.

6.2.4 UniGlue Extension Shield

To streamline the integration of electronic components, a UniGlue extension shield (Figure 6.1d and e) can be permanently attached to an electronic component. This shield features a memory chip that stores the technical specifications such as pinout and voltage, driver specifications in LogicGlue bytecode format, and an SVG representation of the component. The SVG image is compressed using Brotli encoding¹ to reduce its size on the memory chip.

¹ <https://brotli.org/>

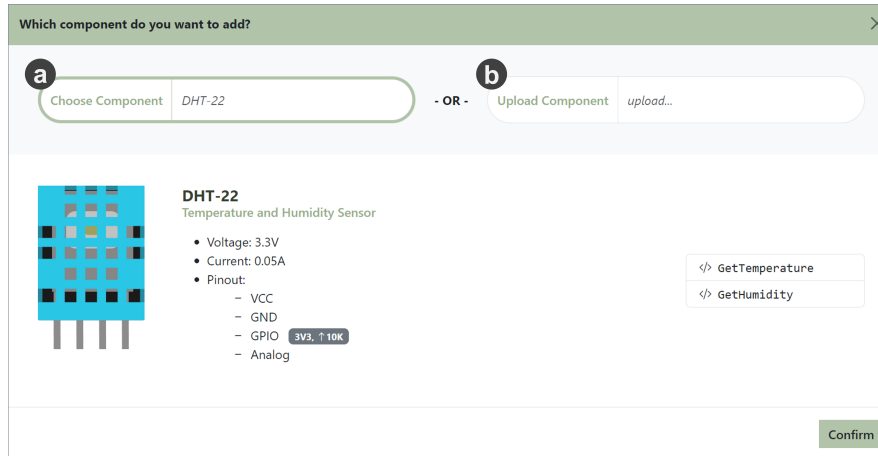


Figure 6.7: Popup for selecting a component (a) or uploading a driver file (b).

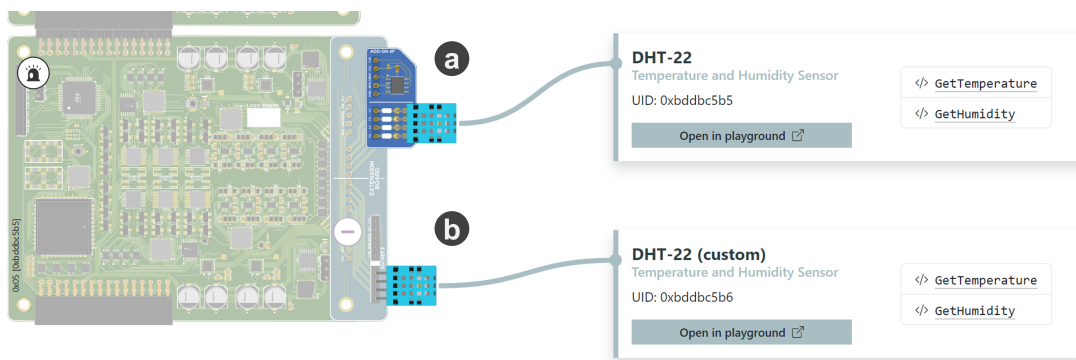
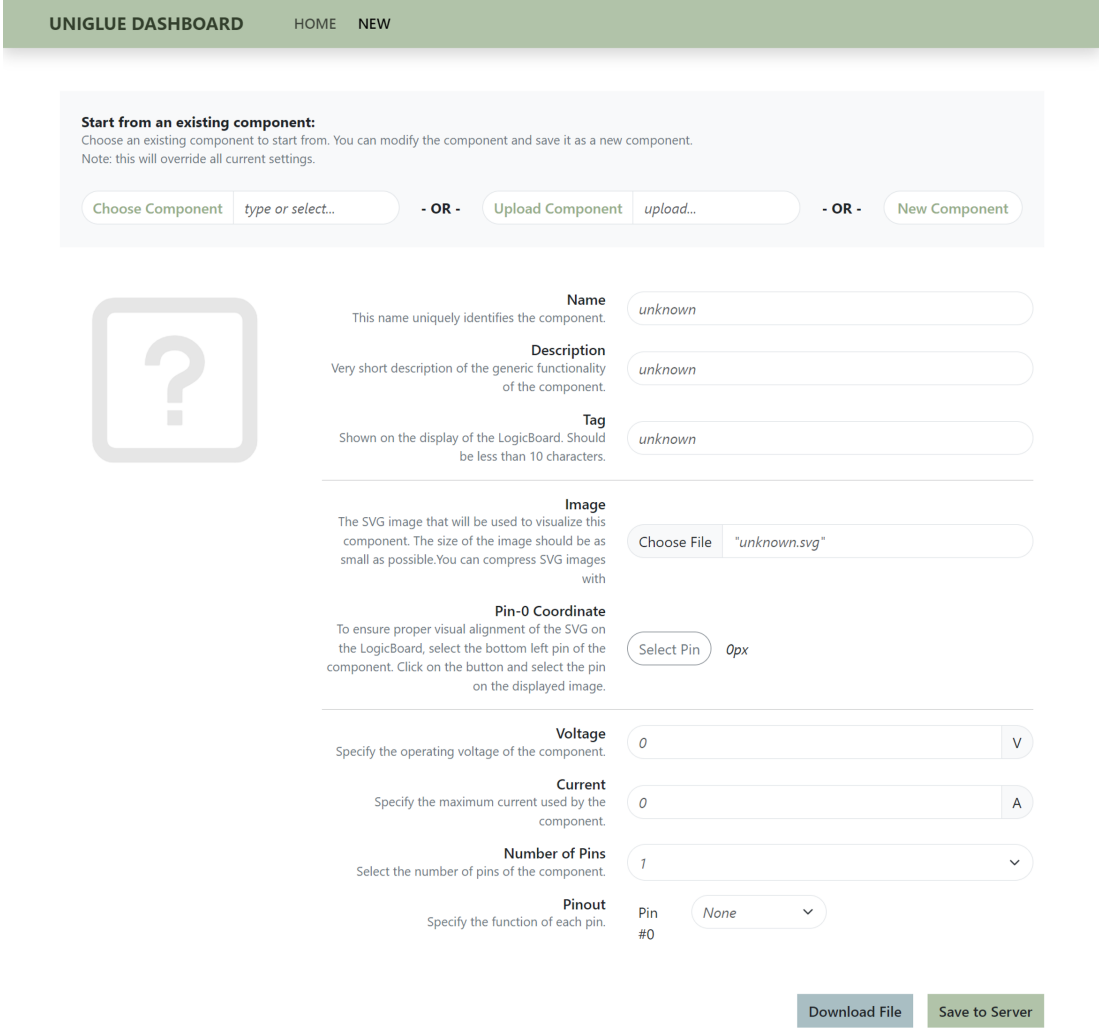


Figure 6.8: Example where component (a) is detected automatically and component (b) is manually configured.

When a component with an extension shield is connected, it is immediately detected, and the programmable header is automatically configured to match the component's specifications. This automated process eliminates the need for manual setup, greatly simplifying the user experience and ensuring seamless functionality. The extension shield communicates with the UniGlue logic board using I2C and contains five pins beside the pins of the attached electronic component. These pins include two for power, two for I2C communication, and one to determine the location of the extension shield on the programmable header in case a splitter board is used.

The UniGlue extension shield guarantees that components are always correctly configured and ready for use, significantly reducing the potential for errors and saving time during the setup process. This level of automation is particularly useful for users who may not have extensive technical knowledge, as it removes the complexity of manual configuration. By handling the technical details, the extension shield allows users to focus on their project goals and innovation without being bogged down by the intricacies of component integration.

Two versions of the UniGlue extension shield are available: one with 4 pins (Figure 6.10a) and one with 8 pins (Figure 6.10b). The 4-pin version is designed for simpler components



UNIGLUE DASHBOARD HOME NEW

Start from an existing component:
 Choose an existing component to start from. You can modify the component and save it as a new component.
 Note: this will override all current settings.

Choose Component - OR - Upload Component - OR - New Component

Name
 This name uniquely identifies the component.

Description
 Very short description of the generic functionality of the component.

Tag
 Shown on the display of the LogicBoard. Should be less than 10 characters.

Image
 The SVG image that will be used to visualize this component. The size of the image should be as small as possible. You can compress SVG images with

Pin-0 Coordinate
 To ensure proper visual alignment of the SVG on the LogicBoard, select the bottom left pin of the component. Click on the button and select the pin on the displayed image.

Voltage
 Specify the operating voltage of the component. V

Current
 Specify the maximum current used by the component. A

Number of Pins
 Select the number of pins of the component.

Pinout
 Specify the function of each pin. Pin #0

Figure 6.9: Image of the UniGlue interface for specifying component specifications for a new driver.

that do not require many connections, while the 8-pin version accommodates more complex components. When using multiple 4-pin components, a splitter (Figure 6.1c) can be used to divide the programmable header into two sets of 4 pins each.

6.3 Walkthrough

This section illustrates how UniGlue facilitates the prototyping workflow with an example where Alex, an electronics hobbyist, uses UniGlue to prototype a smart home system. This system includes a Raspberry Pi Pico microcontroller, a temperature sensor, a proximity sensor, and an RGB LED strip, all working together to control the lighting based on environmental conditions.

Alex starts by connecting the UniGlue controller to his computer using a USB cable and opening the UniGlue software interface. He then plugs the microcontroller into the socket on the UniGlue controller and uploads the UniGlue software stack to the Raspberry Pi Pico.

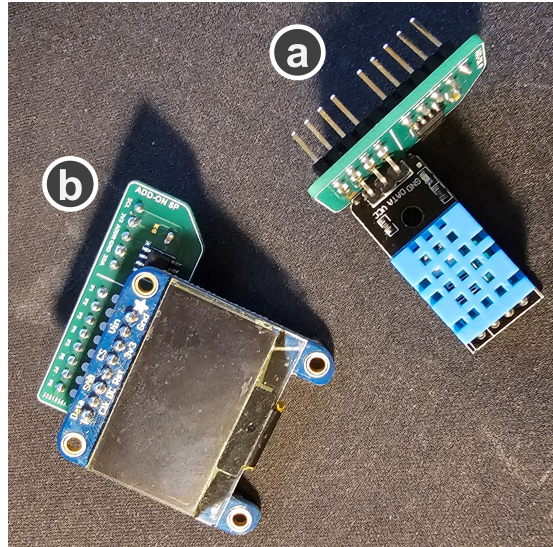


Figure 6.10: UniGlue extension shield with 4 pins (a) or 8 pins (b), depending on the number of pins of the electronic component.

Next, Alex connects a UniGlue logic board to the controller, chooses a temperature sensor, and plugs it into the programmable header on the UniGlue logic board. Since the temperature sensor is equipped with a UniGlue extension shield, Alex does not need to know the component's type or specifications. Once plugged in, UniGlue automatically identifies the sensor as the MCP9808 temperature sensor and configures the programmable header to supply 3.3V and communicate using the I2C protocol. The UniGlue interface shows that the MCP9808 is connected and provides a temperature reading, demonstrating that the component is operational.

Alex then adds another UniGlue logic board to the controller, chooses an infrared proximity sensor, and plugs it into the programmable header. UniGlue automatically detects the extension shield of the proximity sensor and configures the programmable header accordingly. In the UniGlue interface, Alex verifies that the proximity data is available.

For the RGB LED strip, Alex connects a third UniGlue logic board. Since the RGB LED strip lacks an extension shield, Alex uses the UniGlue interface to manually select the RGB LED strip from a drop-down menu and follows the displayed connection diagram to connect the LED strip to the programmable header. Once connected, Alex downloads the platform-independent driver for the LED strip and uploads it through the interface. UniGlue then configures the programmable header to output a PWM signal to control the LED strip's red, green, and blue channels.

Using the LogicGlue programming library, Alex writes his application logic. This library provides easy-to-use functions for interacting with the sensors and the LED strip. Alex writes a simple script to read temperature and proximity data from the sensors and change the color of the RGB LED strip based on these readings. Once Alex uploads his code to the Raspberry Pi, his prototype starts running immediately. However, Alex

notices that the proximity sensor's range is too limited for his application. He quickly swaps the proximity sensor with a PIR motion sensor containing an extension shield, which is automatically recognized and configured by UniGlue. Without changing his application logic, the prototype continues to function, and Alex immediately sees that the PIR motion sensor offers a wider range.

This example demonstrates how UniGlue simplifies the prototyping process by allowing users to effortlessly integrate and configure a wide range of electronic components. Using software-configurable headers and platform-independent drivers, UniGlue provides a flexible and user-friendly environment for developing sophisticated electronic systems. With minimal electronics knowledge, Alex can create and expand a smart home system with ease.

6.4 Discussion

The development of UniGlue marks a significant advancement in the effort to simplify electronics prototyping and make it more accessible to a diverse range of users. By combining the functionalities of CircuitGlue and LogicGlue, UniGlue effectively addresses many of the challenges that have traditionally complicated the prototyping process, particularly for novices. The integration of hardware and software components into a single platform reduces the technical barriers that often prevent new users from engaging fully with electronics prototyping. This unified approach allows users to focus on creative exploration and innovation rather than being bogged down by the complexities of configuring hardware connections or developing compatible software.

6.4.1 Plug-and-Play

A central concept underlying UniGlue is the idea of plug-and-play. This approach was chosen to streamline the prototyping process, making it as intuitive and user-friendly as possible. Plug-and-play systems are designed to automatically recognize and configure new devices or components as soon as they are connected. This eliminates the need for users to manually configure hardware settings or write complex code to establish compatibility between different components, significantly lowering the barriers to entry for electronics prototyping.

The plug-and-play concept was particularly appealing for UniGlue for several reasons. First, it aligns with the goal of democratizing electronics prototyping by making it accessible to users with varying levels of technical expertise. For beginners, the ability to connect components and immediately see them working without extensive setup is incredibly empowering. It allows them to quickly grasp the basics of electronics and gain confidence in their ability to build and experiment with different systems. For more experienced users, plug-and-play can accelerate the development process, enabling rapid prototyping and iteration, though the primary focus of UniGlue is on easing the entry for novices and educational settings.

Another key reason for choosing the plug-and-play approach is its potential to enhance learning and experimentation. By reducing the complexity of the setup process, users can spend more time understanding how different components work together and less time troubleshooting technical issues. This is particularly important in educational environments, where the goal is to foster a deeper understanding of electronics principles and encourage hands-on experimentation. The plug-and-play model supports this by allowing students to focus on learning through doing, rather than getting stuck on configuration challenges.

6.4.2 Moving from TYPE 2 to TYPE 3

While not strictly required for standard operation, the introduction of the UniGlue extension shield significantly enhances plug-and-play functionality by automating all configurations. However, despite its ease of use, a significant challenge lies in the potential need for a (permanent) modification to **TYPE 2** components to make them “UniGlue-compatible”. These modifications could increase the circuit board footprint and reduce compatibility with other **TYPE 2** prototyping platforms. To mitigate this, the UniGlue extension shield has been designed to minimize its impact. It maintains breadboard compatibility by placing all pins in a single row with a consistent offset, ensuring ease of use without significantly altering the component’s form factor. In addition, the extension shield does not alter the component’s pin functions but simply acts as a pass-through.

Despite these efforts to preserve compatibility, the need for any modification raises important questions about the balance between enhancing user experience through improved plug-and-play capabilities and maintaining openness and flexibility within the prototyping ecosystem. Transforming open hardware from **TYPE 2** prototyping into a “UniGlue-compatible” form for **TYPE 3** prototyping could be seen as a double-edged sword in the context of democratizing hardware prototyping. On the one hand, UniGlue’s plug-and-play system simplifies the prototyping process, making it more accessible to users who may not have the technical skills required to modify hardware manually or write intricate software drivers. This aligns with the movement to democratize hardware prototyping by lowering barriers to entry and enabling more people to engage in electronics development.

On the other hand, the requirement to modify **TYPE 2** components for UniGlue compatibility could be perceived as a move away from the principles of openness and flexibility that underpin the maker and open hardware communities. By potentially locking users into an ecosystem, there is a risk that UniGlue could limit the freedom of users to mix and match components from different platforms or to reuse components in diverse projects. This could inadvertently create a divide between users who can afford to adopt a new system and those who prefer or need to stick with more traditional, open prototyping platforms.

The decision to make hardware modifications for UniGlue compatibility brings about several risks that could push the project in a direction contrary to its intended goals. One of the primary risks is the alienation of the maker community, which highly values openness and the ability to modify and customize hardware. If UniGlue is perceived as a closed or restrictive system, it may deter these users, who are often the most enthusiastic advocates of new prototyping technologies. Additionally, requiring specific modifications to components could increase time and financial costs and reduce the availability of compatible parts, creating further barriers to adoption.

To mitigate these risks and ensure that UniGlue remains aligned with the principles of democratizing hardware prototyping, several strategies can be considered. First, UniGlue seeks to maintain maximum compatibility with existing **TYPE 2** components without requiring permanent modifications. This could be achieved by developing adapters or intermediate modules that enable seamless integration without altering the components themselves. For example, the current version of the UniGlue extension shield employs offset holes to create a press-fit connection for components as an alternative to soldering, a concept initially introduced by SparkFun ². This can be seen in Figure 6.11. Second, actively engaging with the maker community to gather feedback and insights will ensure that the platform evolves to meet the needs of its users and aligns with their values. By fostering open dialogue and inviting collaboration, UniGlue can build trust and support among its potential user base.

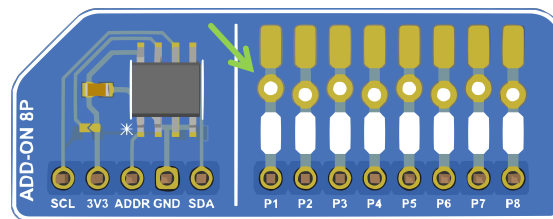


Figure 6.11: The UniGlue extension shield uses offset pin holes to provide a press-fit connection for components as an alternative to soldering.

6.5 Limitations and Future Work

While UniGlue offers significant benefits in terms of ease-of-use and flexibility, there still remain some limitations. The current iteration of the platform has yet to undergo extensive user testing and technical evaluations, which are essential for understanding how well UniGlue meets the needs of its intended audience and for identifying areas where further improvements are needed. Additionally, while UniGlue simplifies the prototyping process for many users, it may not fully cater to those who require more advanced control over their hardware and software configurations. Future developments should consider the needs of these users and explore ways to offer more customization options without compromising the platform’s accessibility.

² <https://www.sparkfun.com/tutorials/114>

A critical area for future development involves conducting comprehensive user evaluations to gather feedback on the platform's usability and effectiveness. This feedback will be invaluable for validating and identifying additional features that could enhance the learning experience. Furthermore, technical assessments will be conducted to expand compatibility with a broader range of components and microcontrollers, ensuring that UniGlue remains adaptable and relevant in a rapidly evolving technological landscape.

Looking ahead, several promising areas for future research and development would further enhance UniGlue's capabilities. One key direction is the development of an interactive playground—an interface that visualizes data from connected components and allows real-time interaction within the hardware, similar to the Jacdac dashboard ³. This feature would enable users to observe and understand the behavior of different electronic components more intuitively by providing immediate feedback on how they function. By making the process more engaging and less intimidating, this interactive approach could significantly enhance the platform's educational value, offering users a deeper insight into how various sensors, actuators, and modules interact within a system. For example, Figure 6.12 illustrates a potential playground for an ultrasonic distance sensor, where users can dynamically map distance measurements to corresponding colors or percentages. Once users are satisfied with their configuration, UniGlue could then automatically generate a custom driver for the ultrasonic distance sensor, translating distance readings into colors rather than standard numerical measurements. As a result, users can more easily experiment with different sensor behaviors and outputs without needing extensive programming knowledge. For example, the color output of the distance sensor can be directly used to set the color of an RGB LED, minimizing programming. This streamlined process not only accelerates the prototyping phase but also empowers users to explore creative applications and innovative uses of electronic components. By simplifying complex interactions and automating driver creation, UniGlue could help bridge the gap between beginners and more advanced users, fostering a more inclusive environment for learning and innovation in electronics prototyping.

Additionally, there is significant potential for UniGlue to be integrated into educational settings, such as classrooms, workshops, and maker spaces. Future research will explore how UniGlue can be used to teach fundamental concepts of electronics and programming, leveraging its intuitive interfaces and interactive playground to create a more engaging and informative learning experience. By providing an approachable entry point into electronics, UniGlue has the potential to inspire early interest in STEM fields and encourage more people to pursue careers in technology.

Finally, exploring the integration of UniGlue with emerging technologies, such as the Internet of Things (IoT) and machine learning systems, represents another exciting avenue for future research. By expanding its capabilities and applications, UniGlue could

³ <https://microsoft.github.io/jacdac-docs/dashboard/>

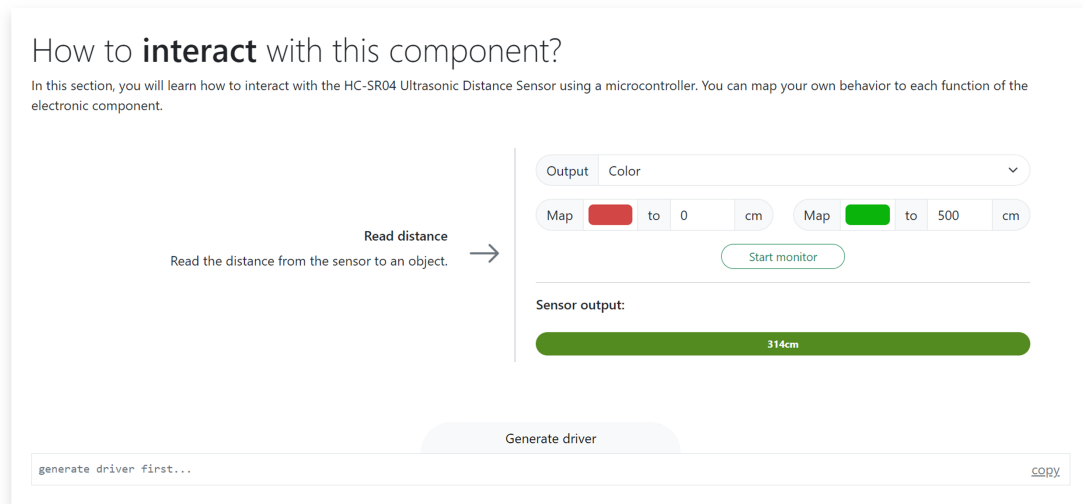


Figure 6.12: Example illustration of a playground for an ultrasonic distance sensor allowing users to map numeric distance readings to percentages or colors.

support more advanced prototyping projects, broadening its appeal and usefulness to a wider range of users.

6.6 Conclusion

This chapter presents UniGlue, an ongoing advancement in the effort to democratize electronics prototyping, aligning with research goals **G2** and **G3**. By integrating the strengths of CircuitGlue and LogicGlue into a unified platform, UniGlue offers a user-friendly solution for both hardware and software integration. The plug-and-play approach, central to UniGlue’s design, enhances usability and accessibility, making electronics prototyping more approachable for novices and educational environments. However, careful consideration must be given to the potential risks and challenges associated with requiring hardware modifications for compatibility. By maintaining openness and fostering community engagement, UniGlue can continue to evolve as a valuable tool for learning and innovation in the field of electronics prototyping. In addition, further research is required to explore the usability of LogicGlue in an empirical user study and technical evaluation.

DISCUSSION AND FUTURE WORK

This chapter provides an overarching discussion of the presented work and highlights future work opportunities.

7.1 Addressing the Research Goals

This dissertation addresses the research goals defined in Section 1.5 aimed at enhancing the electronics prototyping workflow. Each goal contributes to the broader vision of democratizing technology creation through improved tools, frameworks, and knowledge. Below is a detailed overview of how the dissertation meets these goals.

G1: Mapping and Understanding the Electronics Prototyping Domain

The first goal, mapping and understanding the electronics prototyping domain, is addressed in Chapters 2 and 3 through an extensive analysis of current prototyping toolkits. This involves a thorough documentation of their functionalities, advantages, and limitations. By establishing a comprehensive framework, the dissertation systematically categorizes these tools, helping users of different expertise levels navigate the diverse range of applications and focus areas of existing toolkits. The research goes beyond technical specifications by exploring the varied experiences and preferences of users, from hobbyists to professional engineers. This exploration includes examining the challenges users face, the tools they prefer, and the strategies they employ to realize their projects. Through an online survey, firsthand insights are gathered, providing a deep understanding of user needs and behaviors. These insights inform the development of tailored solutions that address unmet needs, enhance the efficiency of prototyping practices, and streamline the decision-making process for selecting appropriate tools.

G2: Bridging Hardware Compatibility and Integration

The second goal is focused on addressing compatibility challenges and the technical complexities involved in integrating various hardware components within electronics prototyping workflows. Innovative approaches to hardware integration are developed

to simplify the process of combining disparate hardware elements, enhancing overall prototyping efficiency. This is exemplified by the introduction of CircuitGlue in Chapter 4, an electronic converter board that facilitates the integration of diverse components by enabling software-programmable pin assignments, protocol translations, and voltage conversions.

G3: Bridging Software Interactions for Prototyping

Complementing the hardware-focused goal, the third objective aims to simplify software interactions within the prototyping process. This involves developing integrated software solutions that facilitate seamless communication and interaction between various applications and hardware devices. LogicGlue is introduced in Chapter 5 as a software framework that decouples the intricate dependencies between platforms, software libraries, drivers, and hardware components. By enabling the creation of platform-independent drivers and applications, LogicGlue enhances the flexibility of prototyping while preserving the full functionality of hardware components.

7.2 The Role of Ecosystems in Physical Computing

7.2.1 Defining and Understanding Ecosystems

In physical computing and electronic prototyping, the term “ecosystem” refers to a structured environment composed of interconnected hardware, software, standards, community practices, and knowledge bases that collectively support the development and deployment of electronic projects. These ecosystems naturally emerge when developing tools to facilitate and democratize electronics prototyping. They provide a comprehensive framework where components and tools are designed to work seamlessly together, thus simplifying the development process and enhancing user experience. Examples of well-known ecosystems include Arduino [Arduino, 2022], Raspberry Pi [Pi, 2022b], and Lego Mindstorms [Lego, 2022], each offering a cohesive set of resources that enable users to create a wide range of projects with relative ease.

An ecosystem in electronic prototyping includes several elements: hardware components like microcontrollers, sensors, actuators, and modular boards designed for interoperability; software tools such as Integrated Development Environments (IDEs), libraries, and drivers that facilitate programming and hardware interaction; standards and protocols that ensure compatibility among components; and community and documentation resources like user communities, forums, tutorials, and official guides that support learning and troubleshooting. These components work together to form a cohesive environment that simplifies the development process and enhances user experience.

Ecosystems significantly influence the prototyping process by providing a structured framework that streamlines development. Well-defined ecosystems reduce compatibility issues, making it easier for users to integrate different components. For instance,

platforms like Arduino and Raspberry Pi have extensive libraries and community support, allowing users to quickly prototype their ideas without delving into low-level hardware details. Standards within an ecosystem ensure that components and tools can be easily combined, reducing the learning curve for new users and accelerating the prototyping process. Ecosystems with robust community support and comprehensive documentation empower users to overcome challenges and innovate more effectively, with online forums, tutorials, and example projects providing valuable resources for troubleshooting and inspiration.

7.2.2 Challenges and Barriers in Ecosystems

While ecosystems provide numerous benefits, they also introduce constraints that often limit flexibility and innovation. Ecosystems, while designed for simplicity, often restrict the integration of external or custom components, posing a barrier for advanced users seeking to push the boundaries of what can be built. Heavy reliance on proprietary tools and standards can lock users into specific ecosystems, making it difficult to transition to other platforms or integrate diverse components. These barriers can stifle creativity and limit the scope of projects that users can undertake within a given ecosystem.

To support a wide range of users, from novices to experts, it is crucial to balance ease of use with the need for flexibility. Achieving this balance involves designing ecosystems with adaptable boundaries that can accommodate various user needs and preferences. Tools like CircuitGlue and LogicGlue exemplify this approach by offering programmable headers and platform-independent drivers, which facilitate the integration of diverse components. By promoting open standards and protocols within ecosystems, compatibility with external components and tools is enhanced, fostering innovation and enabling users to leverage a broader range of technologies. Designing ecosystems with the user in mind means providing intuitive interfaces, comprehensive documentation, and robust community support. These elements lower barriers to entry and foster a more inclusive environment, encouraging a diverse variety of users to engage in electronics prototyping and contribute to the ecosystem.

Several ecosystems exemplify the principles of flexibility and structure in electronic prototyping. The Arduino ecosystem is renowned for its user-friendly approach to electronics prototyping. With a wide range of compatible shields, sensors, and actuators, Arduino provides a versatile platform for both beginners and experts. The extensive community support and wealth of online resources further enhance its appeal. Similarly, Raspberry Pi offers a powerful single-board computer with flexible platform capabilities for various applications, from education to industrial automation. Its compatibility with numerous peripherals and extensive software support makes it a popular choice for prototyping complex systems.

CircuitGlue, for example, requires a software configuration that includes the pinout, voltages, and translation code into a universal protocol. This software configuration is

specific to CircuitGlue and thus requires expertise in both the architecture of CircuitGlue, the used universal protocol, and the electronic component itself. This specificity can be seen as a boundary, as users must have a detailed understanding of these elements to use CircuitGlue effectively. However, once configured, CircuitGlue enables seamless integration of various components, facilitating rapid prototyping and reducing development time.

LogicGlue defines its boundaries by the availability of its platform-independent drivers. These drivers abstract the underlying hardware details, allowing the same driver to be used across different platforms. While the vision is for hardware manufacturers to eventually create these drivers, users often need to develop them themselves using LogicGlue's block-based interface. This interface requires only an understanding of the electronic component, making it accessible for users without deep knowledge of specific microcontroller architectures. As these drivers are platform-independent, community support can significantly speed up development, as a single driver can work on all platforms, fostering a collaborative and supportive ecosystem.

UniGlue offers the same ease of use as CircuitGlue but combines it with the flexibility of LogicGlue's boundaries. UniGlue simplifies hardware and software integration by storing driver information on memory chips embedded in the hardware components. This approach allows for automatic recognition and configuration of components, making it as easy to use as CircuitGlue while maintaining the flexibility and adaptability of LogicGlue. This dual advantage makes UniGlue an ideal tool for both novice users and experienced developers, supporting a wide range of prototyping needs.

7.2.3 The Future of Ecosystems

The future of ecosystems in physical computing and electronic prototyping lies in enhancing both flexibility and inclusivity. To achieve this, it is crucial to develop tools and standards that facilitate interoperability between different ecosystems. This will enable users to integrate a wider range of components and technologies, fostering a more versatile and dynamic prototyping environment. The ability to seamlessly combine diverse elements from various ecosystems will not only expand the potential applications of electronic prototyping but also encourage innovation by removing compatibility barriers.

In conclusion, ecosystems are integral to the landscape of physical computing and electronic prototyping, providing the structure and support necessary for innovation. By balancing structured environments with flexible boundaries, ecosystems can cater to a diverse range of users and applications. Tools like UniGlue, which emphasize adaptability and ease of use, demonstrate the potential for ecosystems to democratize technology and foster a vibrant, inclusive community of creators. As ecosystems continue to evolve, their ability to support interoperability, adaptive learning, and sustainability will be key to their success and impact on the future of electronics prototyping.

7.3 The Importance of Future User Evaluations

As the development of tools like CircuitGlue and LogicGlue illustrates, simplifying the electronics prototyping process has great potential to broaden participation and innovation in this field. However, to truly understand the impact and utility of these tools, it is essential to conduct comprehensive user evaluations that extend beyond initial development feedback. Such evaluations are not just a routine step but a crucial component in validating the tools' effectiveness and ensuring they meet the evolving needs of users.

User evaluations provide a critical lens through which we can view the practical application of CircuitGlue and LogicGlue in varied environments. While the technical capabilities of these tools have been established, understanding how they are utilized in real-world scenarios is equally important. Evaluations should focus on how users from different backgrounds—whether hobbyists, educators, or professional engineers—navigate the tools, what challenges they encounter, and what aspects of the tools they find most valuable. For instance, an educator might value ease of use and integration with classroom activities, while a professional engineer might prioritize robustness and flexibility.

To gain these insights, a multi-faceted approach to user evaluations is necessary. Rather than relying solely on one-off surveys or brief testing periods, future research should aim to include longitudinal studies that track users over extended periods. This approach will reveal not just initial impressions but also how user experiences evolve over time as they become more familiar with the tools and integrate them into their workflows. Such studies can uncover hidden challenges and unexpected benefits that may not be immediately apparent, providing a more nuanced understanding of how these tools impact daily practices.

Additionally, in-depth user evaluations can explore specific contexts in which these tools are deployed. For example, examining the use of CircuitGlue and LogicGlue in educational settings could provide insights into how these tools can be adapted to support learning objectives and improve pedagogical outcomes. Are students more engaged when using these tools? Do they help bridge gaps in understanding basic electronics concepts? By answering these questions, evaluations can guide the development of features tailored to educational needs, enhancing the tools' value in this domain.

On the other hand, evaluating these tools in professional settings can help identify features that need enhancement or additional functionality to meet industry standards. For instance, understanding how engineers use CircuitGlue and LogicGlue in rapid prototyping or product development can highlight areas where the tools excel and where they might require further refinement to handle more complex tasks or integrate with existing professional-grade equipment.

Furthermore, user evaluations can provide insights into the tools' long-term sustainability and adaptability. By observing how users' needs change and how they continue to use (or abandon) the tools over time, we can identify trends that inform future iterations. This ongoing dialogue with users ensures that the tools evolve alongside technological advancements and user expectations, maintaining their relevance and utility in a fast-paced field.

To maximize the effectiveness of these evaluations, a collaborative approach involving various stakeholders is essential. Partnering with educational institutions, maker spaces, and industry professionals allows for a diverse range of perspectives, which is crucial for developing tools that are versatile and broadly applicable. These collaborations also facilitate access to different user groups, ensuring that the evaluations capture a wide spectrum of experiences and use cases.

In summary, while CircuitGlue and LogicGlue represent significant advancements in electronics prototyping, their true potential can only be realized through comprehensive user evaluations. These evaluations are not merely a procedural formality but a fundamental part of the iterative design process. By engaging deeply with users, understanding their needs, and adapting to their feedback, we can ensure that these tools continue to empower a diverse range of users, from beginners to experts, in their creative and technical endeavors.

7.4 Future Directions

Looking ahead, several areas offer potential for further advancements in the field of electronic prototyping, based on the research presented in this dissertation.

7.4.1 Responsive Application Code

In section 5.8, we briefly discussed the concept of swapping electronic components with minimal to no changes to the application code. This focus on replacing electronic components highlights the advantages provided by LogicGlue's platform-independent drivers, which abstract the technical details of individual components. This allows developers to swap components seamlessly without rewriting application logic. However, in practice, replacing components introduces a broader set of considerations beyond software compatibility, and these can affect the overall functionality, performance, and system design. As we look to the future, it is important to explore how these challenges can be addressed in a more comprehensive manner, drawing parallels with the evolution of responsive design in web development.

In the world of web development, responsive design has become a critical concept, allowing websites to adapt dynamically to different screen sizes, resolutions, and device capabilities. This flexibility ensures that users receive an optimized experience regardless of the device they are using, from mobile phones to large desktop monitors.

A similar concept could be applied to electronics prototyping: creating “responsive” application code that is adaptable and robust enough to accommodate changes in hardware components without sacrificing functionality or performance.

In electronics prototyping, the ability to seamlessly swap out components while maintaining application functionality is crucial as new hardware becomes available or as project requirements evolve. However, much like a responsive website that adjusts its layout based on screen size, a robust prototyping system should adapt its behavior based on the capabilities of the newly introduced components. This means not only ensuring compatibility at the software level but also dynamically adjusting performance parameters, energy consumption, and operational precision based on the characteristics of the new hardware.

For example, if a high-precision I2C temperature sensor is replaced with a less accurate analog sensor, the application should adapt its data handling processes accordingly, perhaps by adjusting sampling rates or implementing data smoothing algorithms to compensate for the lower fidelity. This requires the application code to be inherently flexible, capable of recognizing the specifications of new components and altering its behavior dynamically—much like a responsive website resizes its elements for smaller displays. Similarly, when replacing an RGB OLED display with a monochrome one, the system should automatically adjust how it renders graphical content, ensuring legibility and usability even on a less capable display. Another example would be replacing a GPS module with a lower-precision Wi-Fi-based location module. In this case, the system would need to adapt by reducing its reliance on fine-grained geolocation data, perhaps switching to an approximate location mode and adjusting features such as route mapping or distance calculations to ensure the system remains functional, albeit with less precision.

One of the key challenges in developing such a responsive system for hardware is ensuring that the code can not only recognize the new components but also understand their unique constraints and capabilities. LogicGlue, by abstracting the communication layer between components and application logic, takes a significant step toward this goal. It enables components to be swapped without requiring developers to manually adjust for communication protocols, pin mappings, or driver specifications.

However, there is more to be considered. Just as responsive web design involves more than simply resizing content, responsive prototyping systems must address a range of hardware-specific variations, such as:

- **Precision and Resolution:**

When replacing components like sensors or actuators, the resolution or precision may change. This is particularly relevant for sensors, where high-resolution data might be required for certain applications. The software should adapt by modifying how data is processed or by providing additional feedback to the user if the new component’s precision does not meet the required threshold.

- **Timing and Latency:**

Different components may introduce variations in data retrieval or response times. For example, replacing a fast SPI-based sensor with an I2C-based sensor may increase communication latency. The system should adapt its timing constraints or adjust the frequency of updates to ensure that real-time requirements are still met.

- **Power Consumption:**

Some components may consume more power than their predecessors, requiring adjustments in power management strategies. Responsive application code could dynamically adjust the duty cycle of components, manage sleep modes more effectively, or alert the user when a more energy-efficient alternative should be considered.

- **Physical Footprint and Mounting:**

Just as websites need to adjust for different screen real estate, electronics prototypes must consider the physical size and form factor of components. A larger sensor or motor might not fit within the original design constraints, requiring adjustments to the physical layout or housing.

The ultimate goal is to build applications that are hardware-agnostic, capable of interfacing with a wide range of components without compromising performance or user experience. This requires an ecosystem where hardware abstraction is coupled with intelligent, adaptive software that can adjust application behavior in response to the hardware changes detected.

A possible extension of this concept would involve the development of standardized profiles for various types of components, similar to how CSS media queries handle different display types in web development. These profiles would define the core characteristics of each component type, enabling the application to adjust its behavior based on real-time detection of the component's capabilities. For example, a motor profile might include parameters for torque, speed, and power consumption, and the application could adjust its control algorithms to optimize performance based on the available motor.

By developing these standardized profiles and incorporating dynamic detection and response mechanisms, the prototyping system would become more resilient to hardware changes. It could not only ensure compatibility but also provide meaningful adaptations that maintain the integrity of the application even when the underlying hardware shifts.

As electronics prototyping becomes more modular and adaptable, the concept of hardware-responsive application code could revolutionize the field, much like responsive design transformed web development. This shift would allow developers to build more resilient systems capable of evolving alongside hardware advancements, ensuring longevity and flexibility in their designs. Platforms like LogicGlue and CircuitGlue are paving the way by abstracting hardware complexities, but future developments will

need to incorporate dynamic adaptability, enabling a truly hardware-agnostic approach to electronics prototyping.

In conclusion, while LogicGlue and CircuitGlue significantly reduce the complexity of replacing components by abstracting low-level details, there is still a need to build adaptive, responsive application code that can dynamically adjust to hardware changes. By drawing on concepts from responsive web design, the future of electronics prototyping could evolve toward systems that not only integrate components seamlessly but also optimize their behavior based on the unique characteristics of the hardware being used. This approach would enhance the flexibility, scalability, and resilience of prototypes, ensuring that they can evolve with technology while maintaining their core functionality and performance.

7.4.2 Configuration through Hardware and Software

The evolution of the UniGlue platform opens new avenues for exploring the dynamic interplay between hardware and software in electronics prototyping. UniGlue's flexibility allows for seamless integration and configuration of components, paving the way for further innovation in prototype design and functionality. One promising direction is direct hardware configuration, complementing the existing software-driven approach. This method, which involves connecting components like a button and an LED directly in hardware without software intervention, echoes traditional electronics principles exemplified by Integrated Circuits (ICs) such as the 555 timer. Exploring direct hardware configurations within the UniGlue ecosystem could lead to more intuitive prototyping solutions with rapid response times and lower power consumption.

UniGlue's advanced hardware design enables control over both hardware and software elements, allowing prototypes to adapt dynamically based on software commands. This is particularly useful for creating adaptive systems, such as light sensors that modify their sensitivity or operational parameters in response to environmental changes. These adjustments, managed through LogicGlue's driver instructions, are executed by altering the hardware setup via CircuitGlue's capabilities. Future developments could integrate concepts from the VirtualComponent [Kim, 2019] framework, which allows for the digital insertion of electronic components into a system. This integration would enhance UniGlue's capacity for dynamic adjustments, enabling deeper interaction between the physical and digital aspects of prototyping.

7.4.3 Modular Hardware Design

Exploring modular PCBs presents an innovative pathway in advancing electronics prototyping platforms. This concept envisions a prototyping environment where individual sections of a PCB can be swapped or upgraded to align with evolving project requirements or incorporate new functionalities without replacing the entire circuit board. The UniGlue platform, with its adaptability, is ideally positioned to embrace

these hardware modifications seamlessly, eliminating the need for cumbersome manual adjustments.

For example, consider motor driver modules in 3D printer controller boards. As illustrated in Figure 7.1, these modules easily slot into the main board's sockets, allowing straightforward customization or replacement without needing a new controller board. Traditionally, such systems rely on a uniform pin layout across all driver modules to ensure compatibility. However, UniGlue offers a more versatile solution capable of dynamically recognizing and adjusting to the specific pinout of any connected component.

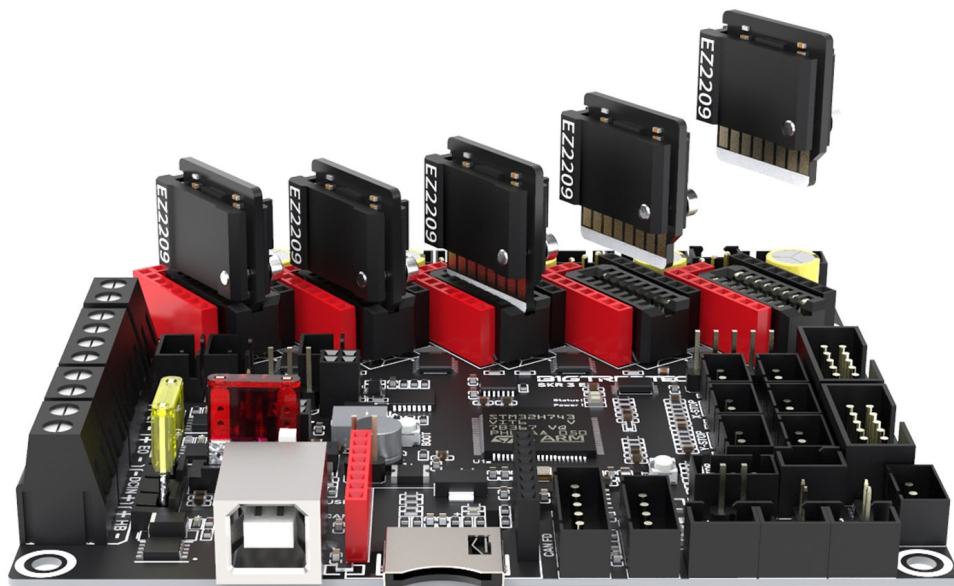


Figure 7.1: Example of a modular PCB with replaceable motor drivers. Photo by BIGTREETECH (SKR3 EZ control board).

This modular approach enhances the customization potential for users and significantly extends the lifecycle and utility of prototyping boards. By facilitating easy upgrades and adjustments, UniGlue empowers users to experiment and innovate with greater freedom. This shift towards modular PCB design represents a significant step forward in making electronics prototyping more accessible, flexible, and sustainable.

7.4.4 Moving from Prototype to Product

The transition from prototype to product, particularly in electronics, has garnered increased focus in recent years, with a trend towards isotyping—integrating prototype functionalities into final, market-ready products [Hodges, 2019b]. While advancements like CircuitGlue and LogicGlue have contributed to this process, they predominantly revolve around producing an enhanced fixed version of the initial prototype. To align

with the evolving trend of isotyping, future research should explore more flexible approaches for creating embedded systems.

One approach is integrating all components onto a single PCB, we refer to as *flattening*. This move shifts away from modular prototyping towards a more unified and product-ready design. There are several considerations based on the level of integration:

- **Reusing existing components** by designing sockets for them on the final PCB maintains modularity while achieving a more finished product form factor.
- **Directly incorporating the design of individual components** into the PCB layout reduces physical bulk and streamlines design, though it requires access to original component schematics.
- **Optimizing the entire system** design to eliminate redundancy, leading to miniaturization and removal of unnecessary components.

Each integration strategy brings its own considerations and challenges, such as component availability in the rapidly evolving electronics market. Designing for flexibility ensures that products can adapt to future changes without requiring a complete redesign. Additionally, transitioning from modular prototypes to single PCB designs raises questions about repairability, upgradability, and customization. Future research should balance the benefits of integrated design with practical considerations of maintenance and upgrades over the product lifecycle.

7.4.5 Enhancing Collaboration and Community Engagement

Collaboration and community engagement are essential for advancing physical computing and fostering innovation. The proposed framework should support collaborative development and knowledge sharing among users. This could involve creating online platforms where users can share projects, exchange ideas, and collaborate on new developments.

CircuitGlue and LogicGlue can significantly enhance collaboration and community engagement. CircuitGlue's universal connectivity and LogicGlue's platform-independent drivers make it easier for users to share and replicate projects. Online repositories where users can upload and download driver specifications, configurations, and project files would facilitate collaboration and community learning.

Engaging with diverse communities can bring new perspectives and insights into developing prototyping tools. Involving users from different backgrounds can help identify unique challenges and opportunities for making prototyping more inclusive and accessible. Future research should explore ways to enhance community engagement, ensuring that the benefits of physical computing are widely shared. Educational outreach programs and partnerships with schools and community organizations can help disseminate these tools to a broader audience, fostering a diverse and inclusive community of makers and innovators.

7.4.6 Addressing Environmental and Sustainability Concerns

As physical computing grows, it is increasingly important to consider the environmental and sustainability implications of prototyping activities. Electronic waste is a significant concern, and the proliferation of prototyping tools and components can contribute to this issue. Future work should explore ways to make prototyping more sustainable, such as developing reusable and recyclable components, minimizing hazardous materials, and promoting responsible disposal practices.

One crucial aspect of sustainability is the reuse of components. Encouraging component reuse reduces waste and promotes cost-effective practices. CircuitGlue and LogicGlue inherently support component reuse through their modular and reconfigurable nature. Designing components with disassembly and reuse in mind can significantly extend their lifecycle and reduce environmental impact. Implementing modular designs where parts can be upgraded or replaced without discarding the entire device further enhances sustainability.

Educational programs and community initiatives can also promote component reuse. Workshops and tutorials that teach individuals how to salvage and repurpose components from old or broken devices foster a culture of sustainability. Online platforms for exchanging or donating unused components help reduce waste and make prototyping more accessible.

Furthermore, the energy consumption of electronic devices is a critical factor in their environmental impact. Research should focus on developing energy-efficient components and systems and exploring renewable energy sources for powering interactive devices. By prioritizing sustainability, the proposed framework can contribute to more environmentally friendly practices in physical computing.

CONCLUSION

This dissertation explores electronic prototyping workflows, identifying key challenges and proposing innovative solutions to advance the field. Through an extensive literature review and user study, we developed a detailed framework for evaluating electronic prototyping platforms, with a focus on usability rather than just technical specifications. The study revealed a significant gap between user expectations and the actual performance of existing platforms, emphasizing the need for solutions that better align with user needs.

Based on these findings, we introduced CircuitGlue and LogicGlue, each designed to address specific aspects of the prototyping process. CircuitGlue simplifies the physical assembly of hardware components, making it easier for users to build and modify their projects. LogicGlue, on the other hand, streamlines software integration by enabling the creation of drivers that function across various platforms, thus reducing the complexity involved in programming and component interaction.

In addition to these contributions, we have also explored the concept of UniGlue as an ongoing project that combines the strengths of CircuitGlue and LogicGlue. UniGlue aims to further simplify the prototyping experience by enabling plug-and-play functionality with off-the-shelf components. A key feature of this concept is its extension shield, which includes an embedded memory chip that stores all necessary component information. When a component is connected, UniGlue automatically configures the software driver, hardware pinout, and voltage settings, removing common obstacles in hardware and software integration. While UniGlue is still in the development phase, its potential to enhance user experience and accessibility in electronics prototyping is promising.

In summary, this dissertation presents a new approach to electronic prototyping that prioritizes ease of use and inclusivity, seamlessly integrating hardware and software processes. While CircuitGlue and LogicGlue mark significant steps toward achieving these goals, UniGlue represents an exciting avenue for future work.

BIBLIOGRAPHY

Academic Sources

- [Agrawal, 2015] Harshit Agrawal, Udayan Umapathi, Robert Kovacs, Johannes Frohnhofen, Hsiang-Ting Chen, Stefanie Mueller, and Patrick Baudisch. “Protopiper: Physically Sketching Room-Sized Objects at Actual Scale”. In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. UIST ’15. Charlotte, NC, USA: Association for Computing Machinery, Nov. 5, 2015, pp. 427–436. ISBN: 978-1-4503-3779-3. DOI: 10.1145/2807442.2807505. URL: <https://doi.org/10.1145/2807442.2807505> (visited on 05/05/2020).
- [Alphonsus, 2016] Ephrem Ryan Alphonsus and Mohammad Omar Abdullah. “A review on the applications of programmable logic controllers (PLCs)”. In: *Renewable and Sustainable Energy Reviews* 60 (2016), pp. 1185–1205. ISSN: 1364-0321. DOI: <https://doi.org/10.1016/j.rser.2016.01.025>. URL: <https://www.sciencedirect.com/science/article/pii/S1364032116000551>.
- [Anderson, 2017] Fraser Anderson, Tovi Grossman, and George Fitzmaurice. “Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry”. In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST ’17. Québec City, QC, Canada: Association for Computing Machinery, 2017, pp. 331–342. ISBN: 9781450349819. DOI: 10.1145/3126594.3126637. URL: <https://doi.org/10.1145/3126594.3126637>.
- [Austin, 2020] Jonny Austin, Howard Baker, Thomas Ball, James Devine, Joe Finney, Peli De Halleux, Steve Hodges, Michał Moskal, and Gareth Stockdale. “The BBC micro:bit: from the U.K. to the world”. In: *Communications of the ACM* 63.3 (Feb. 24, 2020), pp. 62–69. ISSN: 0001-0782. DOI: 10.1145/3368856. URL: <https://doi.org/10.1145/3368856> (visited on 05/06/2020).
- [Ball, 2019] Thomas Ball, Peli de Halleux, and Michał Moskal. “Static TypeScript: an implementation of a static compiler for the TypeScript language”. In: *Proceedings of the 16th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes*. MPLR 2019. Athens, Greece: Association for Computing Machinery, 2019,

- pp. 105–116. ISBN: 9781450369770. DOI: 10.1145/3357390.3361032. URL: <https://doi.org/10.1145/3357390.3361032>.
- [Banzi, 2008] Massimo Banzi. *Getting Started with Arduino*. Ill. Sebastopol, CA: Make Books - Imprint of: O'Reilly Media, 2008. ISBN: 978-0-596-15551-3.
- [Bdeir, 2009] Ayah Bdeir. “Electronics as Material: LittleBits”. In: *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*. TEI '09. Cambridge, United Kingdom: Association for Computing Machinery, 2009, pp. 397–400. ISBN: 9781605584935. DOI: 10.1145/1517664.1517743. URL: <https://doi.org/10.1145/1517664.1517743>.
- [Blikstein, 2013a] Paulo Blikstein. “Digital fabrication and ‘making’ in education: The democratization of invention”. In: *FabLabs: Of machines, makers and inventors* 4.1 (2013), pp. 1–21.
- [Blikstein, 2013b] Paulo Blikstein. “Gears of our childhood: constructionist toolkits, robotics, and physical computing, past and future”. In: *Proceedings of the 12th international conference on interaction design and children*. 2013, pp. 173–182.
- [Blikstein, 2015] Paulo Blikstein et al. “Computationally Enhanced Toolkits for Children: Historical Review and a Framework for Future Design.” In: *Found. Trends Hum. Comput. Interact.* 9.1 (2015), pp. 1–68.
- [Buechley, 2005] L. Buechley, N. Elumeze, C. Dodson, and M. Eisenberg. “Quilt Snaps: A fabric based computational construction kit”. In: *IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE'05)*. New York, NY, USA: IEEE Institute of Electrical and Electronics Engineers, Dec. 28, 2005, 3 pp. ISBN: 978-0-7695-2385-9. DOI: 10.1109/WMTE.2005.55.
- [Buechley, 2008] Leah Buechley, Mike Eisenberg, Jaime Catchen, and Ali Crockett. “The LilyPad Arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: Association for Computing Machinery, Apr. 6, 2008, pp. 423–432. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357123. URL: <https://doi.org/10.1145/1357054.1357123> (visited on 05/05/2020).
- [Charmaz, 2014] Kathy Charmaz. *Constructing grounded theory*. sage, 2014.
- [Devine, 2018] James Devine, Joe Finney, Peli de Halleux, Michał Moskal, Thomas Ball, and Steve Hodges. “MakeCode and CODAL: intuitive and efficient embedded systems programming for education”. In: *SIGPLAN Not.* 53.6 (June 2018), pp. 19–30. ISSN: 0362-1340. DOI: 10.1145/3299710.3211335. URL: <https://doi.org/10.1145/3299710.3211335>.
- [Devine, 2022] James Devine, Michał Moskal, Peli de Halleux, Thomas Ball, Steve Hodges, Gabriele D’Amone, David Gakure, Joe Finney, Lorraine Underwood, Kobi Hartley, Paul Kos, and Matt Oppenheim. “Plug-and-play Physical Computing with Jacdac”. In: *Proc. ACM Interact.*

- Mob. Wearable Ubiquitous Technol.* 6.3 (Sept. 2022). DOI: 10.1145/3550317. URL: <https://doi.org/10.1145/3550317>.
- [Drew, 2016] Daniel Drew, Julie L. Newcomb, William McGrath, Filip Maksimovic, David Mellis, and Björn Hartmann. "The Toastboard: Ubiquitous Instrumentation and Automated Checking of Breadboarded Circuits". In: *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. UIST '16. Tokyo, Japan: Association for Computing Machinery, 2016, pp. 677–686. ISBN: 9781450341899. DOI: 10.1145/2984511.2984566. URL: <https://doi.org/10.1145/2984511.2984566>.
- [Eisenberg, 2002] Michael Eisenberg, Ann Eisenberg, Mark Gross, Khomkrit Kaowthumrong, Nathaniel Lee, and Will Lovett. "Computationally-enhanced construction kits for children: Prototype and principles". In: *Proceedings of the Fifth International Conference of the Learning Sciences*. 2002, pp. 23–26.
- [Garg, 2022] Radhika Garg and Hua Cui. "Social Contexts, Agency, and Conflicts: Exploring Critical Aspects of Design for Future Smart Home Technologies". In: *ACM Trans. Comput.-Hum. Interact.* 29.2 (Jan. 2022). ISSN: 1073-0516. DOI: 10.1145/3485058. URL: <https://doi.org/10.1145/3485058>.
- [Greenberg, 2001] Saul Greenberg and Chester Fitchett. "Phidgets: easy development of physical interfaces through physical widgets". In: *Proceedings of the 14th annual ACM symposium on User interface software and technology*. UIST '01. Orlando, Florida: Association for Computing Machinery, Nov. 11, 2001, pp. 209–218. ISBN: 978-1-58113-438-4. DOI: 10.1145/502348.502388. URL: <https://doi.org/10.1145/502348.502388> (visited on 05/05/2020).
- [Grosse-Puppendahl, 2017] Tobias Grosse-Puppendahl, Christian Holz, Gabe Cohn, Raphael Wimmer, Oskar Bechtold, Steve Hodges, Matthew S Reynolds, and Joshua R Smith. "Finding common ground: A survey of capacitive sensing in human-computer interaction". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2017, pp. 3293–3315.
- [Guljajeva, 2022] Varvara Guljajeva and Mar Canet Sola. "Dream Painter: An Interactive Art Installation Bridging Audience Interaction, Robotics, and Creative AI". In: *Proceedings of the 30th ACM International Conference on Multimedia*. MM '22. <conf-loc>, <city>Lisboa</city>, <country>Portugal</country>, </conf-loc>. Association for Computing Machinery, 2022, pp. 7235–7236. ISBN: 9781450392037. DOI: 10.1145/3503161.3549976. URL: <https://doi.org/10.1145/3503161.3549976>.
- [Hamdan, 2018] Nur Al-huda Hamdan, Simon Voelker, and Jan Borchers. "Sketch&Stitch: Interactive Embroidery for E-textiles". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal QC, Canada: Association for Computing Machinery, Apr. 19, 2018, pp. 1–13. ISBN: 978-1-4503-5620-6.

- doi: 10.1145/3173574.3173656. URL: <https://doi.org/10.1145/3173574.3173656> (visited on 05/05/2020).
- [Hartmann, 2006] Björn Hartmann, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. "Reflective physical prototyping through integrated design, test, and analysis". In: *Proceedings of the 19th annual ACM symposium on User interface software and technology - UIST '06*. the 19th annual ACM symposium. Montreux, Switzerland: ACM Press, 2006, p. 299. ISBN: 978-1-59593-313-3. DOI: 10.1145/1166253.1166300. URL: <http://dl.acm.org/citation.cfm?doid=1166253.1166300> (visited on 05/06/2020).
- [Hodges, 2020] Steve Hodges. "Democratizing the Production of Interactive Hardware". In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 5–6. ISBN: 9781450375146. URL: <https://doi.org/10.1145/3379337.3422877>.
- [Hodges, 2019a] Steve Hodges and Nicholas Chen. "Long Tail Hardware: Turning Device Concepts Into Viable Low Volume Products". In: *IEEE Pervasive Computing* 18 (Oct. 1, 2019), pp. 51–59. doi: 10.1109/MPRV.2019.2947966.
- [Hodges, 2019b] Steve Hodges and Nicholas Chen. "Long Tail Hardware: Turning Device Concepts Into Viable Low Volume Products". In: *IEEE Pervasive Computing* 18.4 (Dec. 2019), pp. 51–59. URL: <https://www.microsoft.com/en-us/research/publication/long-tail-hardware-turning-device-concepts-into-viable-low-volume-products/>.
- [Hodges, 2013] Steve Hodges, James Scott, Sue Sentance, Colin Miller, Nicolas Villar, Scarlet Schwiderski-Grosche, Kerry Hammil, and Steven Johnston. ".NET Gadeteer: A New Platform for K-12 Computer Science Education". In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. SIGCSE '13. Denver, Colorado, USA: Association for Computing Machinery, 2013, pp. 391–396. ISBN: 9781450318686. DOI: 10.1145/2445196.2445315. URL: <https://doi.org/10.1145/2445196.2445315>.
- [Hodges, 2014] Steve Hodges, Nicolas Villar, Nicholas Chen, Tushar Chugh, Jie Qi, Diana Nowacka, and Yoshihiro Kawahara. "Circuit stickers: peel-and-stick construction of interactive electronic prototypes". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '14. Toronto, Ontario, Canada: Association for Computing Machinery, Apr. 26, 2014, pp. 1743–1746. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557150. URL: <https://doi.org/10.1145/2556288.2557150> (visited on 05/05/2020).
- [Hodges, 2012] Steve Hodges, Nicolas Villar, James Scott, and Albrecht Schmidt. "A New Era for Ubicomp Development". In: *IEEE Pervasive Computing* 11.1 (2012), pp. 5–9. doi: 10.1109/MPRV.2012.1.
- [Karchemsky, 2019] Mitchell Karchemsky, J.D. Zamfirescu-Pereira, Kuan-Ju Wu, François Guimbretière, and Bjoern Hartmann. "Heimdall: A

- Remotely Controlled Inspection Workbench For Debugging Microcontroller Projects". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*. the 2019 CHI Conference. Glasgow, Scotland Uk: ACM Press, 2019, pp. 1–12. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300728. URL: <http://dl.acm.org/citation.cfm?doid=3290605.3300728> (visited on 05/06/2020).
- [Kawahara, 2013] Yoshihiro Kawahara, Steve Hodges, Benjamin S. Cook, Cheng Zhang, and Gregory D. Abowd. "Instant inkjet circuits: lab-based inkjet printing to support rapid prototyping of UbiComp devices". In: *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. UbiComp '13. Zurich, Switzerland: Association for Computing Machinery, Sept. 8, 2013, pp. 363–372. ISBN: 978-1-4503-1770-2. DOI: 10.1145/2493432.2493486. URL: <https://doi.org/10.1145/2493432.2493486> (visited on 05/05/2020).
- [Kazemitabaar, 2016] Majeed Kazemitabaar, Liang He, Katie Wang, Chloe Aloimonos, Tony Cheng, and Jon E. Froehlich. "ReWear: Early Explorations of a Modular Wearable Construction Kit for Young Children". In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '16. San Jose, California, USA: Association for Computing Machinery, May 7, 2016, pp. 2072–2080. ISBN: 978-1-4503-4082-3. DOI: 10.1145/2851581.2892525. URL: <https://doi.org/10.1145/2851581.2892525> (visited on 05/06/2020).
- [Kazemitabaar, 2017] Majeed Kazemitabaar, Jason McPeak, Alexander Jiao, Liang He, Thomas Outing, and Jon E. Froehlich. "MakerWear: A Tangible Approach to Interactive Wearable Creation for Children". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. Denver, Colorado, USA: Association for Computing Machinery, May 2, 2017, pp. 133–145. ISBN: 978-1-4503-4655-9. DOI: 10.1145/3025453.3025887. URL: <https://doi.org/10.1145/3025453.3025887> (visited on 05/05/2020).
- [Khurana, 2020] Rushil Khurana and Steve Hodges. "Beyond the Prototype: Understanding the Challenge of Scaling Hardware Device Production". In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI '20. Honolulu, HI, USA: Association for Computing Machinery, Apr. 21, 2020, pp. 1–11. ISBN: 978-1-4503-6708-0. DOI: 10.1145/3313831.3376761. URL: <https://doi.org/10.1145/3313831.3376761> (visited on 05/05/2020).
- [Kim, 2020a] Yoonji Kim, Youngkyung Choi, Daye Kang, Minkyong Lee, Tek-Jin Nam, and Andrea Bianchi. "HeyTeddy: Conversational Test-Driven Development for Physical Computing". In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3.4 (Sept. 2020). DOI: 10.1145/3369838. URL: <https://doi.org/10.1145/3369838>.
- [Kim, 2019] Yoonji Kim, Youngkyung Choi, Hyein Lee, Geehyuk Lee, and Andrea Bianchi. "VirtualComponent: A Mixed-Reality Tool for

- Designing and Tuning Breadboarded Circuits". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. Glasgow, Scotland Uk: Association for Computing Machinery, 2019, pp. 1–13. ISBN: 9781450359702. DOI: 10.1145/3290605.3300407. URL: <https://doi.org/10.1145/3290605.3300407>.
- [Kim, 2020b] Yoonji Kim, Hyein Lee, Ramkrishna Prasad, Seungwoo Je, Youngkyung Choi, Daniel Ashbrook, Ian Oakley, and Andrea Bianchi. "SchemaBoard: Supporting Correct Assembly of Schematic Circuits Using Dynamic In-Situ Visualization". In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 987–998. ISBN: 9781450375146. URL: <https://doi.org/10.1145/3379337.3415887>.
- [Knörig, 2009] André Knörig, Reto Wettach, and Jonathan Cohen. "Fritzing: a tool for advancing electronic prototyping for designers". In: *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction - TEI '09*. the 3rd International Conference. Cambridge, United Kingdom: ACM Press, 2009, p. 351. ISBN: 978-1-60558-493-5. DOI: 10.1145/1517664.1517735. URL: <http://portal.acm.org/citation.cfm?doid=1517664.1517735> (visited on 05/06/2020).
- [Lambrichts, 2021] Mannu Lambrichts, Raf Ramakers, Steve Hodges, Sven Coppers, and James Devine. "A Survey and Taxonomy of Electronics Toolkits for Interactive and Ubiquitous Device Prototyping". In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5.2 (June 2021). DOI: 10.1145/3463523. URL: <https://doi.org/10.1145/3463523>.
- [Lambrichts, 2023] Mannu Lambrichts, Raf Ramakers, Steve Hodges, James Devine, Lorraine Underwood, and Joe Finney. "CircuitGlue: A Software Configurable Converter for Interconnecting Multiple Heterogeneous Electronic Components". In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7.2 (June 2023). DOI: 10.1145/3596265. URL: <https://doi.org/10.1145/3596265>.
- [Lambrichts, 2020] Mannu Lambrichts, Jose Maria Tijerina, and Raf Ramakers. "Soft-Mod: A Soft Modular Plug-and-Play Kit for Prototyping Electronic Systems". In: *Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction*. TEI '20. Sydney NSW, Australia: Association for Computing Machinery, Feb. 9, 2020, pp. 287–298. ISBN: 978-1-4503-6107-1. DOI: 10.1145/3374920.3374950. URL: <https://doi.org/10.1145/3374920.3374950> (visited on 05/05/2020).
- [Ledo, 2018] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. "Evaluation strategies for HCI toolkit research". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018, pp. 1–17.
- [Lee, 2021] Woojin Lee, Ramkrishna Prasad, Seungwoo Je, Yoonji Kim, Ian Oakley, Daniel Ashbrook, and Andrea Bianchi. "VirtualWire: Supporting Rapid Prototyping with Instant Reconfigurations of Wires in Breadboarded Circuits". In: *Proceedings of the Fifteenth Interna-*

- tional Conference on Tangible, Embedded, and Embodied Interaction*. TEI '21. Salzburg, Austria: Association for Computing Machinery, 2021, pp. 1–12. ISBN: 9781450382137. DOI: 10.1145/3430524.3440623. URL: <https://doi.org/10.1145/3430524.3440623>.
- [Leen, 2017] Danny Leen, Raf Ramakers, and Kris Luyten. “StrutModeling: A Low-Fidelity Construction Kit to Iteratively Model, Test, and Adapt 3D Objects”. In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST '17. Québec City, QC, Canada: Association for Computing Machinery, Oct. 20, 2017, pp. 471–479. ISBN: 978-1-4503-4981-9. DOI: 10.1145/3126594.3126643. URL: <https://doi.org/10.1145/3126594.3126643> (visited on 05/05/2020).
- [Levis, 2005] Philip Levis, Samuel Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. “TinyOS: An Operating System for Sensor Networks”. In: vol. 00. New York: Springer-Verlag, Jan. 2005, pp. 115–148. ISBN: 978-3-540-23867-6. DOI: 10.1007/3-540-27139-2_7.
- [McGrath, 2017] Will McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis, and Björn Hartmann. “BifröSt: Visualizing and Checking Behavior of Embedded Systems across Hardware and Software”. In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST '17. Québec City, QC, Canada: Association for Computing Machinery, 2017, pp. 299–310. ISBN: 9781450349819. DOI: 10.1145/3126594.3126658. URL: <https://doi.org/10.1145/3126594.3126658>.
- [McGrath, 2018] William McGrath, Jeremy Warner, Mitchell Karchemsky, Andrew Head, Daniel Drew, and Bjoern Hartmann. “WiFröSt: Bridging the Information Gap for Debugging of Networked Embedded Systems”. In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. UIST '18. Berlin, Germany: Association for Computing Machinery, 2018, pp. 447–455. ISBN: 9781450359481. DOI: 10.1145/3242587.3242668. URL: <https://doi.org/10.1145/3242587.3242668>.
- [Mellis, 2013] David A Mellis, Sam Jacoby, Leah Buechley, Hannah Perner-Wilson, and Jie Qi. “Microcontrollers as material: crafting circuits with paper, conductive ink, electronic components, and an “untookit””. In: *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*. 2013, pp. 83–90.
- [Mellis, 2016] David A. Mellis, Leah Buechley, Mitchel Resnick, and Björn Hartmann. “Engaging Amateurs in the Design, Fabrication, and Assembly of Electronic Devices”. In: *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*. DIS '16. Brisbane, QLD, Australia: Association for Computing Machinery, 2016, pp. 1270–1281. ISBN: 9781450340311. DOI: 10.1145/2901790.2901833. URL: <https://doi.org/10.1145/2901790.2901833>.

- [Merrill, 2012] David Merrill, Emily Sun, and Jeevan Kalanithi. "Sifteo cubes". In: *CHI '12 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '12. Austin, Texas, USA: Association for Computing Machinery, May 5, 2012, pp. 1015–1018. ISBN: 978-1-4503-1016-1. DOI: 10.1145/2212776.2212374. URL: <https://doi.org/10.1145/2212776.2212374> (visited on 05/05/2020).
- [Mi, 2017] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. "An empirical characterization of IFTTT: ecosystem, usage, and performance". In: *Proceedings of the 2017 Internet Measurement Conference*. IMC '17: Internet Measurement Conference. London United Kingdom: ACM, Nov. 2017, pp. 398–404. ISBN: 978-1-4503-5118-8. DOI: 10.1145/3131365.3131369. URL: <https://dl.acm.org/doi/10.1145/3131365.3131369> (visited on 05/06/2020).
- [Myers, 2000] Brad Myers, Scott E. Hudson, and Randy Pausch. "Past, present, and future of user interface software tools". In: *ACM Trans. Comput.-Hum. Interact.* 7.1 (Mar. 2000), pp. 3–28. ISSN: 1073-0516. DOI: 10.1145/344949.344959. URL: <https://doi.org/10.1145/344949.344959>.
- [Nagels, 2018] Steven Nagels, Raf Ramakers, Kris Luyten, and Wim Deferme. "Silicone Devices: A Scalable DIY Approach for Fabricating Self-Contained Multi-Layered Soft Circuits using Microfluidics". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal QC, Canada: Association for Computing Machinery, Apr. 19, 2018, pp. 1–13. ISBN: 978-1-4503-5620-6. DOI: 10.1145/3173574.3173762. URL: <https://doi.org/10.1145/3173574.3173762> (visited on 05/05/2020).
- [Narumi, 2015] Koya Narumi, Steve Hodges, and Yoshihiro Kawahara. "ConductAR: an augmented reality based tool for iterative design of conductive ink circuits". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: Association for Computing Machinery, Sept. 7, 2015, pp. 791–800. ISBN: 978-1-4503-3574-4. DOI: 10.1145/2750858.2804267. URL: <https://doi.org/10.1145/2750858.2804267> (visited on 05/06/2020).
- [Olsen, 2007] Dan R. Olsen. "Evaluating user interface systems research". In: *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*. UIST '07. Newport, Rhode Island, USA: Association for Computing Machinery, 2007, pp. 251–258. ISBN: 9781595936790. DOI: 10.1145/1294211.1294256. URL: <https://doi.org/10.1145/1294211.1294256>.
- [Perteneder, 2020] Florian Perteneder, Kathrin Probst, Joanne Leong, Sebastian Gassler, Christian Rendl, Patrick Parzer, Katharina Fluch, Sophie Gahleitner, Sean Follmer, Hideki Koike, and Michael Haller. "Foxels: Build Your Own Smart Furniture". In: *Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction*. TEI '20. Sydney NSW, Australia: Association for Computing Machinery, Feb. 9, 2020, pp. 111–122.

- ISBN: 978-1-4503-6107-1. DOI: 10.1145/3374920.3374935. URL: <https://doi.org/10.1145/3374920.3374935> (visited on 05/05/2020).
- [Ramakers, 2015] Raf Ramakers, Kashyap Todi, and Kris Luyten. "PaperPulse: An Integrated Approach to Fabricating Interactive Paper". In: *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '15. Seoul, Republic of Korea: Association for Computing Machinery, 2015, pp. 267–270. ISBN: 9781450331463. DOI: 10.1145/2702613.2725430. URL: <https://doi.org/10.1145/2702613.2725430>.
- [Read, 2023] Jake Robert Read, Leo Mcelroy, Quentin Bolsee, B Smith, and Neil Gershenfeld. "Modular-Things: Plug-and-Play with Virtualized Hardware". In: *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI EA '23. Hamburg, Germany: Association for Computing Machinery, 2023. ISBN: 9781450394222. DOI: 10.1145/3544549.3585642. URL: <https://doi.org/10.1145/3544549.3585642>.
- [Resnick, 2005] Mitchel Resnick and Brian Silverman. "Some reflections on designing construction kits for kids". In: *Proceedings of the 2005 conference on Interaction design and children*. 2005, pp. 117–122.
- [Savage, 2015] Valkyrie Savage, Sean Follmer, Jingyi Li, and Björn Hartmann. "Makers' Marks: Physical Markup for Designing and Fabricating Functional Objects". In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. UIST '15. Charlotte, NC, USA: Association for Computing Machinery, Nov. 5, 2015, pp. 103–108. ISBN: 978-1-4503-3779-3. DOI: 10.1145/2807442.2807508. URL: <https://doi.org/10.1145/2807442.2807508> (visited on 05/05/2020).
- [Schweikardt, 2006] Eric Schweikardt and Mark D. Gross. "roBlocks: a robotic construction kit for mathematics and science education". In: *Proceedings of the 8th international conference on Multimodal interfaces*. ICMI '06. Banff, Alberta, Canada: Association for Computing Machinery, Nov. 2, 2006, pp. 72–75. ISBN: 978-1-59593-541-0. DOI: 10.1145/1180995.1181010. URL: <https://doi.org/10.1145/1180995.1181010> (visited on 05/06/2020).
- [Scott, 2011] James Scott, A.J. Bernheim Brush, John Krumm, Brian Meyers, Michael Hazas, Stephen Hodges, and Nicolas Villar. "PreHeat: controlling home heating using occupancy prediction". In: *Proceedings of the 13th international conference on Ubiquitous computing*. UbiComp '11. Beijing, China: Association for Computing Machinery, Sept. 17, 2011, pp. 281–290. ISBN: 978-1-4503-0630-0. DOI: 10.1145/2030112.2030151. URL: <https://doi.org/10.1145/2030112.2030151> (visited on 05/06/2020).
- [Seneviratne, 2017] Suranga Seneviratne, Yining Hu, Tham Nguyen, Guohao Lan, Sara Khalifa, Kanchana Thilakarathna, Mahbub Hassan, and Aruna Seneviratne. "A Survey of Wearable Devices and Challenges". In: *IEEE Communications Surveys and Tutorials* 19.4 (July 2017),

- pp. 2573–2620. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2731979.
- [Shneiderman, 2006] Ben Shneiderman, Gerhard Fischer, Mary Czerwinski, Mitch Resnick, Brad Myers, Linda Candy, Ernest Edmonds, Mike Eisenberg, Elisa Giaccardi, Tom Hewett, Pamela Jennings, Bill Kules, Kumiyo Nakakoji, Jay Nunamaker, Randy Pausch, Ted Selker, Elisabeth Sylvan, and Michael Terry. “Creativity Support Tools: Report From a U.S. National Science Foundation Sponsored Workshop”. In: *International Journal of Human–Computer Interaction* 20.2 (2006), pp. 61–77. DOI: 10.1207/s15327590ijhc2002_1. eprint: https://doi.org/10.1207/s15327590ijhc2002_1. URL: https://doi.org/10.1207/s15327590ijhc2002_1.
- [Silva, 2014] Hugo Placido da Silva, Ana Fred, and Raul Martins. “Biosignals for Everyone”. In: *IEEE Pervasive Computing* 13.4 (Oct. 2014), pp. 64–71. ISSN: 1536-1268. DOI: 10.1109/MPRV.2014.61. URL: <http://ieeexplore.ieee.org/document/6926682/> (visited on 05/06/2020).
- [Strasnick, 2017] Evan Strasnick, Maneesh Agrawala, and Sean Follmer. “Scanalog: Interactive Design and Debugging of Analog Circuits with Programmable Hardware”. In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST ’17. Québec City, QC, Canada: Association for Computing Machinery, 2017, pp. 321–330. ISBN: 9781450349819. DOI: 10.1145/3126594.3126618. URL: <https://doi.org/10.1145/3126594.3126618>.
- [Świerczyński, 2014] R. Świerczyński, K. Urbański, and A. Wymysłowski. “Methodology for supporting electronic system prototyping through semiautomatic component selection”. In: *2014 15th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE)*. New York, NY, USA: Institute of Electrical and Electronics Engineers, 2014, pp. 1–4. DOI: 10.1109/EuroSimE.2014.6813792.
- [Wang, 2016] Chiuang Wang, Hsuan-Ming Yeh, Bryan Wang, Te-Yen Wu, Hsin-Ruey Tsai, Rong-Hao Liang, Yi-Ping Hung, and Mike Y. Chen. “CircuitStack: Supporting Rapid Prototyping and Evolution of Electronic Circuits”. In: *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. UIST ’16. Tokyo, Japan: Association for Computing Machinery, 2016, pp. 687–695. ISBN: 9781450341899. DOI: 10.1145/2984511.2984527. URL: <https://doi.org/10.1145/2984511.2984527>.
- [Weigel, 2015] Martin Weigel, Tong Lu, Gilles Bailly, Antti Oulasvirta, Carmel Majidi, and Jürgen Steimle. “iSkin: Flexible, Stretchable and Visually Customizable On-Body Touch Sensors for Mobile Computing”. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI ’15. Seoul, Republic of Korea: Association for Computing Machinery, Apr. 18, 2015, pp. 2991–3000. ISBN: 978-1-4503-3145-6. DOI: 10.1145/2702123.2702391. URL: <https://doi.org/10.1145/2702123.2702391> (visited on 05/05/2020).

- [Wu, 2019] Te-Yen Wu, Jun Gong, Teddy Seyed, and Xing-Dong Yang. "Proxino: Enabling Prototyping of Virtual Circuits with Physical Proxies". In: *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. UIST '19. New Orleans, LA, USA: Association for Computing Machinery, 2019, pp. 121–132. ISBN: 9781450368162. DOI: 10.1145/3332165.3347938. URL: <https://doi.org/10.1145/3332165.3347938>.
- [Wu, 2017] Te-Yen Wu, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Jun-You Liu, Yu-Chih Lin, and Mike Y. Chen. "CurrentViz: Sensing and Visualizing Electric Current Flows of Breadboarded Circuits". In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST '17. Québec City, QC, Canada: Association for Computing Machinery, 2017, pp. 343–349. ISBN: 9781450349819. DOI: 10.1145/3126594.3126646. URL: <https://doi.org/10.1145/3126594.3126646>.

Online Sources

- [Adafruit, 2021a] Adafruit. *Adafruit 12-Key Capacitive Touch Sensor Breakout - MPR121*. 2021. URL: <https://www.adafruit.com/product/1982>.
- [Adafruit, 2021b] Adafruit. *Adafruit Industries, Unique & fun DIY electronics and kits*. 2021. URL: <https://www.adafruit.com/>.
- [Adafruit, 2021c] Adafruit. *Arduino Playground: Arduino-Compatible Hardware*. 2021. URL: <https://playground.arduino.cc/Main/SimilarBoards>.
- [Adafruit, 2021d] Adafruit. *What is STEMMA?* 2021. URL: <https://learn.adafruit.com/introducing-adafruit-stemma-qt>.
- [Adafruit, 2022] Adafruit. *Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055*. 2022. URL: <https://www.adafruit.com/product/2472>.
- [Adafruit, 2024] Adafruit. *Circuit Playground Express*. 2024. URL: <https://www.adafruit.com/product/3333>.
- [Alchitry, 2021] Alchitry. *Alchitry homepage*. 2021. URL: <https://alchitry.com/>.
- [Anadigm, 2022] Anadigm. *Anadigm FPAA*. 2022. URL: <https://www.anadigm.com/fpaa.asp>.
- [Arduino, 2021] Arduino. *Arduino MKR Vidor 4000 product page*. 2021. URL: <https://store.arduino.cc/arduino-mkr-vidor-4000>.
- [Arduino, 2022] Arduino. *Arduino - Home*. 2022. URL: <https://www.arduino.cc/>.
- [Arduino, 2024a] Arduino. *Arduino - Libraries*. 2024. URL: <https://www.arduino.cc/reference/en/libraries/>.
- [Arduino, 2024b] Arduino. *Arduino IDE*. 2024. URL: <https://www.arduino.cc/en/software>.
- [Autodesk, 2021] Autodesk. *Autodesk Eagle: PCB design made easy for every engineer*. 2021. URL: <https://www.autodesk.com/products/eagle>.
- [BeagleBone, 2021] BeagleBone. *BeagleBoard - community supported open hardware computers for making*. 2021. URL: <https://beagleboard.org/>.
- [Bitalino, 2021] Bitolino. *BITalino - Biomedical Equipment | Low-Cost Toolkit*. 2021. URL: <https://bitalino.com/>.

- [Burgess, 2024] Phillip Burgess. *Adafruit GFX Graphics Library*. 2024. URL: <https://learn.adafruit.com/adafruit-gfx-graphics-library/overview>.
- [Circuito, 2022] Circuito. *Design Your Circuit with Circuito.io*. 2022. URL: <https://www.circuito.io/>.
- [Circuits, 2021] Snap Circuits. *Educational STEM Toys: Snap Circuits*. 2021. URL: <https://www.elenco.com/>.
- [Coral, 2021] Coral. *Coral homepage*. 2021. URL: <https://coral.ai/>.
- [Cubelets, 2021] Cubelets. *Modular Robotics Cubelets robot blocks*. 2021. URL: <https://www.modrobotics.com/>.
- [Cubetto, 2021] Cubetto. *Meet Cubetto - Primo Toys Cubetto: A toy robot teaching kids code & computer programming*. 2021. URL: <https://www.primotoys.com/>.
- [DeltaMaker, 2021] DeltaMaker. *DeltaMaker: An Elegant 3D Printer*. 2021. URL: <https://www.deltamaker.com/>.
- [Edison, 2024] Intel Edison. *Intel Edison Compute Module*. 2024. URL: <https://ark.intel.com/content/www/us/en/ark/products/84572/intel-edison-compute-module-iot.html>.
- [ESLOV, 2021] ESLOV. *ESLOV IoT Invention Kit (Canceled)*. 2021. URL: <https://www.kickstarter.com/projects/iot-invention-kit/eslov-iot-invention-kit>.
- [Espressif, 2021] Espressif. *Espressif offers a wide range of fully-certified Wi-Fi & Bluetooth modules powered by our own advanced SoCs*. 2021. URL: <https://www.espressif.com/en/products/modules>.
- [Espressif, 2022a] Espressif. *A cost-effective and highly integrated Wi-Fi MCU for IoT applications*. 2022. URL: <https://www.espressif.com/en/products/socs/esp8266>.
- [Espressif, 2022b] Espressif. *ESP32*. 2022. URL: <https://www.espressif.com/en/products/socs/esp32>.
- [FreeRTOS, 2024] FreeRTOS. *FreeRTOS: Real-time operating system for microcontrollers*. 2024. URL: <https://www.freertos.org/index.html>.
- [Fritzing, 2021] Fritzing. *Fritzing*. 2021. URL: <http://fritzing.org/>.
- [Geppetto, 2021] Geppetto. *Welcome to Geppetto*. 2021. URL: <https://geppetto.gumstix.com/>.
- [Grove, 2021] Grove. *Grove Beginner Kit for Arduino(EOL) - Seeed Wiki*. 2021. URL: https://wiki.seeedstudio.com/Grove_Beginner_Kit_for_Arduino/.
- [Infineon, 2022] Infineon. *32-bit PSoC™ Arm® Cortex® Microcontroller*. 2022. URL: <https://www.infineon.com/cms/en/product/microcontroller/32-bit-psoc-arm-cortex-microcontroller/>.
- [Kamps, 2021] H. J. Kamps. *Hardware is Hard: Getting a Kickstarter project out the door*. 2021. URL: <https://medium.com/triggertrap-playbook/hardware-is-hard-getting-a-kickstarter-project-shipped-59c9596bdd7f>.
- [Labs, 2021] SAM Labs. *SAM Labs homepage*. 2021. URL: <https://samlabs.com/us/>.

[Launchpad, 2021]	TI Launchpad. <i>Hardware Kits & Boards Design Resources</i> . 2021. URL: http://www.ti.com/design-resources/embedded-development/hardware-kits-boards.html .
[Lego, 2022]	Lego. <i>Lego Mindstorms EV3</i> . 2022. URL: https://www.lego.com/en-us/product/lego-mindstorms-ev3-31313 .
[LLVM, 2024]	LLVM. <i>The LLVM Compiler Infrastructure</i> . 2024. URL: https://llvm.org/ .
[Loxone, 2021]	Loxone. <i>Loxone Smart Home & Commercial Projects Create Automation</i> . 2021. URL: https://www.loxone.com/enus/ .
[MakeCode, 2024]	MakeCode. <i>Microsoft MakeCode for micro:bit</i> . 2024. URL: https://makecode.microbit.org/ .
[MakeMagazine, 2021]	MakeMagazine. <i>Makers' Guide to Boards</i> . 2021. URL: https://makezine.com/comparison/boards/ .
[Mbed, 2024]	Mbed. <i>Mbed OS</i> . 2024. URL: https://os.mbed.com/mbed-os/ .
[mBot, 2021]	mBot. <i>Makeblock mBot Entry-level Educational Robot Kit</i> . 2021. URL: https://www.makeblock.com/mbot .
[microbit, 2022]	BBC micro:bit. <i>Micro:bit Educational Foundation</i> . 2022. URL: https://microbit.org/ .
[Microsoft, 2024a]	Microsoft. <i>DeviceScript TypeScript for Tiny IoT Devices</i> . 2024. URL: https://microsoft.github.io/devicescript/ .
[Microsoft, 2024b]	Microsoft. <i>Visual Studio Code</i> . 2024. URL: https://code.visualstudio.com/ .
[Mikroe, 2021]	Mikroe. <i>Mikroelektronika Click Boards</i> . 2021. URL: http://www.mikroe.com/click .
[Niko, 2021]	Niko. <i>Niko Home Control</i> . 2021. URL: https://www.niko.eu/en/products/niko-home-control .
[Node-RED, 2024]	Node-RED. <i>Node-RED - Low-code programming for event-driven applications</i> . 2024. URL: https://nodered.org/ .
[OASIS, 2024a]	OASIS. <i>CoAP - Constrained Application Protocol</i> . 2024. URL: https://coap.space/ .
[OASIS, 2024b]	OASIS. <i>MQTT</i> . 2024. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html .
[Pi, 2021]	Raspberry Pi. <i>Compute Module 3+</i> . 2021. URL: https://www.raspberrypi.org/products/compute-module-3-plus/ .
[Pi, 2022a]	Raspberry Pi. <i>RP2040</i> . 2022. URL: https://www.raspberrypi.com/products/rp2040/ .
[Pi, 2022b]	Raspberry Pi. <i>Teach, Learn, and Make with Raspberry Pi</i> . 2022. URL: https://www.raspberrypi.org/ .
[Pi, 2022c]	Raspberry Pi. <i>What is PIO?</i> 2022. URL: https://www.raspberrypi.com/news/what-is-pio/ .
[Piper, 2021]	Piper. <i>Piper Computer Kit</i> . 2021. URL: https://www.playpiper.com/ .
[PlatformIO, 2024]	PlatformIO. <i>PlatformIO</i> . 2024. URL: https://platformio.org/ .
[Pmod, 2021]	Pmod. <i>Digilent Pmod Modules and Connectors - Interface with Development Boards</i> . 2021. URL: https://store.digilentinc.com/pmod-modules-connectors/ .

- [Romano, 2022] David Romano. *A Brief History of FPGA*. 2022. URL: <https://makezine.com/2019/10/11/a-brief-history-of-fpga/>.
- [ROS, 2024] ROS. *ROS - Robot Operating System*. 2024. URL: <https://www.ros.org/>.
- [Semiconductor, 2021] Nordic Semiconductor. *Bluetooth Low Energy, Bluetooth mesh, NFC, Thread and Zigbee development kit for the nRF52840 SoC*. 2021. URL: <https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52840-DK>.
- [Semiconductor, 2022a] Nordic Semiconductor. *nRF52 Series*. 2022. URL: https://infocenter.nordicsemi.com/index.jsp?topic=%5C%2Fstruct_nrf52%5C%2Fstruct%5C%2Fnrf52.html.
- [Semiconductor, 2022b] Nordic Semiconductor. *PPI — Programmable peripheral interconnect*. 2022. URL: <https://infocenter.nordicsemi.com/index.jsp?topic=%5C%2Fcom.nordic.infocenter.nrf52832.ps.v1.1%5C%2Fppi.html>.
- [SparkFun, 2021a] SparkFun. *Prototyping with I2C has never been easier*. 2021. URL: <https://www.sparkfun.com/qwiic>.
- [SparkFun, 2021b] SparkFun. *SparkFun Electronics*. 2021. URL: <https://www.sparkfun.com/>.
- [Sphero, 2021] Sphero. *STEM Kits & Robotics for Kids Inspire STEM Education with Sphero*. 2021. URL: <https://sphero.com/>.
- [STMicroelectronics, 2021] STMicroelectronics. *STMicroelectronics STM32 Discovery Kits*. 2021. URL: <https://www.st.com/en/evaluation-tools/stm32-discovery-kits.html>.
- [Studio, 2021] Seeed Studio. *Seeed Studio - Seeed Studio*. 2021. URL: <https://www.seeedstudio.com/>.
- [Teensy, 2021] Teensy. *Teensy USB Development Board*. 2021. URL: <https://www.pjrc.com/teensy/>.
- [Zephyr, 2024a] Zephyr. *Device Driver Model*. 2024. URL: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/zephyr/kernel/drivers/index.html.
- [Zephyr, 2024b] Zephyr. *Zephyr*. 2024. URL: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/zephyr/index.html.

APPENDICES

TOOLKIT CLASSIFICATION

A.1 Holistic Characteristics

LOGICGLUE DRIVER SPECIFICATION

B.1 Instructions

Instruction	Description	Parameters
OP_NOP	No operation	No parameters
OP_BOOT	Marks the start of the boot section	No parameters
OP_FUNCTION	Define a function with arguments and properties	name:8, args:8, props:8
OP_PROPERTY	Define a property for the current function	name:8, n:8, accepts:8[n]
OP_PROPERTY_CONST	Define a constant property for the current function	name:8, val:num
OP_DEFINE_INPUT_FORMAT	Define the format and scale of input parameters	num:8, (format:8, scale:8)[n]
OP_DEFINE_OUTPUT_FORMAT	Define the format and scale of output parameters	num:8, (format:8, scale:8)[n]
OP_DEFINE_FUNCTION_TYPE	Define the type of the function, defaults to the default function	type:8
OP_SET_VAR	Save value in a variable	id:8, val:num
OP_SET_VAR_INITIAL	Save value in a variable only if the variable is not set	id:8, val:num
OP_SET_ARG	Save value in argument at specified index	id:8, idx:8, val:num
OP_SET_PROP	Save value in the current property	val:num
OP_SET_LOCAL	Save value in a local variable, valid for the current context	id:8, val:num
OP_SET_PARAM	Save value in a parameter referencing a variable outside the current context	id:8, val:num

OP_SET_LIST	Save list in a list variable	id:8, lst:list
OP_LIST_CREATE_1D	Create a 1D list for specified number of items	id:8, type:8(list_e), n:num
OP_LIST_CREATE_2D	Create a 2D list for specified number of items in x and y directions	id:8, type:8(list_e), n:num, m:num
OP_LIST_SET_1D	Save value in the 1D list at specified index	id:8, idx:num, val:num
OP_LIST_SET_2D	Save value in the 2D list at specified index	id:8, idx:num, idy:num, val:num
OP_LIST_FILL	Fill the list with the specified value	id:8, val:num
OP_PRINT	Print the value	val:num
OP_PRINT_LIST	Print the list	lst:list
OP_LABEL	Mark a label that can be jumped to	label:8
OP_GOTO	Go to a label	label:8
OP_GOTO_IF	Go to a label if the value is truthy	label:8, val:num
OP_GOTO_IF_NOT	Go to a label if the value is falsy	label:8, val:num
OP_CALL	Go to a label and store the current index on the stack	label:8
OP_CALL_ARGS	Go to a label, set the given list of parameters, and store the current index on the stack	label:8, n:8, idx[n]
OP_CALL_INTERNAL	Go to an internal function referenced by the label	internal:8
OP_CALL_INTERNAL_ARGS	Go to an internal function referenced by the label, and set the given list of parameters	internal:8, n:8, idx[n]
OP_RETURN	Return from the current call	No parameters
OP_BLOCK	Execute a block of instructions	n:8, instruction[n]
OP_BREAK	Break out of the current block or return	No parameters
OP_BREAK_IF	Break out of the current block or return if the value is truthy	val:num
OP_IF	Execute the instruction if the value is truthy	val:num, instruction
OP_IF_ELSE	Execute the first instruction if the value is truthy, otherwise execute the second instruction	val:num, instruction, instruction

OP_IF_ELIF_ELSE	Execute the instruction if the value is truthy, if no match execute the default instruction	n:8, (val:num, instruction)[n], instruction
OP_SWITCH	Switch the value over cases and execute the corresponding instruction, if no match execute the default instruction	n:8, switch:num, (case:num, instruction)[n], instruction
OP_LOOP	Create a loop and execute the instruction	start:num, end:num, incr:num, instruction
OP_FOREACH	For each item in the list, execute the instruction	lst:list, instruction
OP_FOREACH_BYTE	For each byte in the list, execute the instruction	lst:list, instruction
OP_EXIT	Exit the program	No parameters
OP_EXIT_CODE	Exit the program with a given code	code:num
OP_ASSERT	Assert the value is truthy, if not exit the program	val:num
OP_ASSERT_CODE	Assert the value is truthy, if not exit the program with a given code	val:num, code:num
HW_DELAY_MS	Delay for specified milliseconds	duration:num
HW_DELAY_US	Delay for specified microseconds	duration:num
HW_GPIO_CONFIG	Configure the mode of the pin	pin:num, mode:num
HW_GPIO_WRITE	Set the state of the pin	pin:num, state:num
HW_GPIO_PULSE_MS	Pulse the pin to state for specified milliseconds	pin:num, state:num, duration:num
HW_GPIO_PULSE_MS_n	Pulse the pin to state for specified milliseconds, multiple times with interval	pin:num, state:num, duration:num, interval:num, n:num
HW_GPIO_PULSE_US	Pulse the pin to state for specified microseconds	pin:num, state:num, duration:num

HW_GPIO_PULSE_US_n	Pulse the pin to state for specified microseconds, multiple times with interval	pin:num, state:num, du- ration:num, inter- val:num, n:num
HW_GPIOTE_CONFIG	Configure interrupts for pin on edge, when edge triggers, execute label	pin:num, edge:num, la- bel:8
HW_ADC_CONFIG	Configure ADC on specified pin	pin:num
HW_PWM_CONFIG	Configure PWM on specified pin with period	pin:num, pe- riod:num
HW_PWM_WRITE	Set PWM duty cycle on specified pin	pin:num, duty:num
HW_I2C_CONFIG	Configure I2C with address and frequency	addr:num, freq:8
HW_I2C_WRITE	Write value to I2C	val:num
HW_I2C_WRITE_2	Write two values to I2C	val:num, val:num
HW_I2C_WRITE_LIST	Write list to I2C	lst:list
HW_SPI_CONFIG	Configure SPI with CS, frequency, mode, and order	cs:num, freq:8, mode:8, order:8
HW_SPI_WRITE	Write value to SPI	val:num
HW_SPI_WRITE_2	Write two values to SPI	val:num, val:num
HW_SPI_WRITE_LIST	Write list to SPI	lst:list
HW_UART_CONFIG	Configure UART with baud rate, and mode	baudrate:8, mode:8
HW_UART_WRITE	Write value to UART	val:num
HW_UART_WRITE_2	Write two values to UART	val:num, val:num
HW_UART_WRITE_LIST	Write list to UART	lst:list
HW_DTH_CONFIG	Configure DTH on specified pin	pin:num
HW_WS2812_CONFIG	Configure WS2812 on specified pin	pin:num
HW_WS2812_WRITE	Write value to WS2812	val:num
HW_WS2812_WRITE_LIST	Write list to WS2812	lst:list

B.2 Numerics Subsystem

Instruction	Description	Parameters
_U8	Define an 8-bit unsigned integer constant	val:8
_U16	Define a 16-bit unsigned integer constant	val:16
_U32	Define a 32-bit unsigned integer constant	val:32
_I8	Define an 8-bit signed integer constant	val:8
_I16	Define a 16-bit signed integer constant	val:16
_I32	Define a 32-bit signed integer constant	val:32
_FLT	Define a 32-bit floating-point constant	val:32
_FLT_HEX	Define a 32-bit floating-point constant in hexadecimal format	val:32
_FIX	Define a fixed-point constant	val:32, frac:8
_CONFIG	Get configuration value	id:8
_ARG	Get the value at specified index from the argument	id:8, idx:num
_PROP	Get the value of the current property	No parameters
_VAR	Get variable value	id:8
_VAR_LOCAL	Get local variable value	id:8
_VAR_PARAM	Get parameter value	id:8
LIST_GET_1D	Get the value at specified index from the 1D list	id:8, idx:num
LIST_GET_2D	Get the value at specified index from the 2D list	id:8, idx:num, idy:num
LIST_SIZE	Get the number of items in the list	id:8
CAST	Cast the numeric value to the given type	type:8(num_e), val:num
MATH_ADD	Add two numeric values	val_a:num, val_b:num
MATH_SUB	Subtract second numeric value from the first	val_a:num, val_b:num
MATH_MUL	Multiply two numeric values	val_a:num, val_b:num

MATH_DIV	Divide the first numeric value by the second	val_a:num, val_b:num
MATH_MOD	Get the remainder of division of the first numeric value by the second	val_a:num, val_b:num
MATH_POW	Raise the first numeric value to the power of the second	val_a:num, val_b:num
MATH_SQRT	Get the square root of the numeric value	val:num
MATH_AND	Perform bitwise AND operation on two numeric values	val_a:num, val_b:num
MATH_OR	Perform bitwise OR operation on two numeric values	val_a:num, val_b:num
MATH_XOR	Perform bitwise XOR operation on two numeric values	val_a:num, val_b:num
MATH_NOT	Perform bitwise NOT operation on the numeric value	val:num
MATH_SHL	Shift the first numeric value left by the number of bits specified by the second	val_a:num, val_b:num
MATH_SHR	Shift the first numeric value right by the number of bits specified by the second	val_a:num, val_b:num
MATH_MIN	Get the minimum of two numeric values	val_a:num, val_b:num
MATH_MAX	Get the maximum of two numeric values	val_a:num, val_b:num
MATH_CLAMP	Clamp the numeric value within the range specified by low and high values	val:num, low:num, high:num
MATH_ABS	Get the absolute value of the numeric value	val:num
MATH_CEIL	Get the ceiling value of the numeric value	val:num
MATH_FLOOR	Get the floor value of the numeric value	val:num
MATH_ROUND	Round the numeric value	val:num
MATH_SIN	Get the sine of the numeric value	val:num
MATH_COS	Get the cosine of the numeric value	val:num
MATH_TAN	Get the tangent of the numeric value	val:num
MATH_ASIN	Get the arcsine of the numeric value	val:num

MATH_ACOS	Get the arccosine of the numeric value	val:num
MATH_ATAN	Get the arctangent of the numeric value	val:num
MATH_ATAN2	Get the arctangent of the quotient of the two numeric values	val:num
BITS_BIT	Get the value of the nth bit of the numeric value	val:num, n:num
BITS_MASK_NEG	Create a mask with background 1 and the nth bit 0	type:8(num_e), mask:num
BITS_MASK_POS	Create a mask with background 0 and the nth bit 1	type:8(num_e), mask:num
IF_EQ	Evaluate to val_if if val_a == val_b, otherwise evaluate to val_else	val_a:num, val_b:num, val_if:num, val_else:num
IF_NE	Evaluate to val_if if val_a != val_b, otherwise evaluate to val_else	val_a:num, val_b:num, val_if:num, val_else:num
IF_GT	Evaluate to val_if if val_a > val_b, otherwise evaluate to val_else	val_a:num, val_b:num, val_if:num, val_else:num
IF_GE	Evaluate to val_if if val_a >= val_b, otherwise evaluate to val_else	val_a:num, val_b:num, val_if:num, val_else:num
IF_LT	Evaluate to val_if if val_a < val_b, otherwise evaluate to val_else	val_a:num, val_b:num, val_if:num, val_else:num
IF_LE	Evaluate to val_if if val_a <= val_b, otherwise evaluate to val_else	val_a:num, val_b:num, val_if:num, val_else:num
BOOL_AND	Evaluate to the boolean val_a AND val_b	val_a:num, val_b:num
BOOL_OR	Evaluate to the boolean val_a OR val_b	val_a:num, val_b:num
BOOL_NOT	Evaluate to the boolean NOT val	val:num
EVAL_EQ	Evaluate to 1 if val_a == val_b, otherwise 0	val_a:num, val_b:num

EVAL_NE	Evaluate to 1 if val_a != val_b, otherwise 0	val_a:num, val_b:num
EVAL_GT	Evaluate to 1 if val_a > val_b, otherwise 0	val_a:num, val_b:num
EVAL_GE	Evaluate to 1 if val_a >= val_b, otherwise 0	val_a:num, val_b:num
EVAL_LT	Evaluate to 1 if val_a < val_b, otherwise 0	val_a:num, val_b:num
EVAL_LE	Evaluate to 1 if val_a <= val_b, otherwise 0	val_a:num, val_b:num
NUM_SWITCH	Switch the value over cases and evaluate to val if switch == case, if no match, evaluate to def	n:8, switch:num, (case:num, val:num)[n], def:num
NUM_LOOP_IDX_0	Get the current loop index 0	No parameters
NUM_LOOP_IDX_1	Get the current loop index 1	No parameters
HW_MILLIS	Returns the number of milliseconds since the board was powered up	No parameters
HW_MICROS	Returns the number of microseconds since the board was powered up	No parameters
HW_GPIO_READ	Read the value of a GPIO pin	pin:num
HW_GPIO_PULSE_READ	Get the duration of a pulse on the pin	pin:num, state:num
HW_GPIO_PULSE_READ_T	Get the duration of a pulse on the pin or timeout	pin:num, state:num, time- out:num
HW_ADC_READ	Read the value of an ADC pin	pin:num
HW_I2C_READ_U8	Read an 8-bit unsigned integer from the I2C device	No parameters
HW_I2C_READ_U16	Read a 16-bit unsigned integer from the I2C device	No parameters
HW_I2C_READ_U32	Read a 32-bit unsigned integer from the I2C device	No parameters
HW_I2C_READ_I8	Read an 8-bit signed integer from the I2C device	No parameters
HW_I2C_READ_I16	Read a 16-bit signed integer from the I2C device	No parameters
HW_I2C_READ_I32	Read a 32-bit signed integer from the I2C device	No parameters
HW_I2C_READ_FLT	Read a 32-bit floating point number from the I2C device	No parameters

HW_I2C_WRITE_READ_U8	Write a numeric to the I2C device, then read an 8-bit unsigned integer	val:num
HW_I2C_WRITE_READ_U16	Write a numeric to the I2C device, then read a 16-bit unsigned integer	val:num
HW_I2C_WRITE_READ_U32	Write a numeric to the I2C device, then read a 32-bit unsigned integer	val:num
HW_I2C_WRITE_READ_I8	Write a numeric to the I2C device, then read an 8-bit signed integer	val:num
HW_I2C_WRITE_READ_I16	Write a numeric to the I2C device, then read a 16-bit signed integer	val:num
HW_I2C_WRITE_READ_I32	Write a numeric to the I2C device, then read a 32-bit signed integer	val:num
HW_I2C_WRITE_READ_FLT	Write a numeric to the I2C device, then read a 32-bit floating point number	val:num
HW_SPI_READ_U8	Read an 8-bit unsigned integer from the SPI device	No parameters
HW_SPI_READ_U16	Read a 16-bit unsigned integer from the SPI device	No parameters
HW_SPI_READ_U32	Read a 32-bit unsigned integer from the SPI device	No parameters
HW_SPI_READ_I8	Read an 8-bit signed integer from the SPI device	No parameters
HW_SPI_READ_I16	Read a 16-bit signed integer from the SPI device	No parameters
HW_SPI_READ_I32	Read a 32-bit signed integer from the SPI device	No parameters
HW_SPI_READ_FLT	Read a 32-bit floating point number from the SPI device	No parameters
HW_SPI_WRITE_READ_U8	Write a numeric to the SPI device, then read an 8-bit unsigned integer	val:num
HW_SPI_WRITE_READ_U16	Write a numeric to the SPI device, then read a 16-bit unsigned integer	val:num
HW_SPI_WRITE_READ_U32	Write a numeric to the SPI device, then read a 32-bit unsigned integer	val:num

HW_SPI_WRITE_READ_I8	Write a numeric to the SPI device, then read an 8-bit signed integer	val:num
HW_SPI_WRITE_READ_I16	Write a numeric to the SPI device, then read a 16-bit signed integer	val:num
HW_SPI_WRITE_READ_I32	Write a numeric to the SPI device, then read a 32-bit signed integer	val:num
HW_SPI_WRITE_READ_FLT	Write a numeric to the SPI device, then read a 32-bit floating point number	val:num
HW_UART_READ_U8	Read an 8-bit unsigned integer from the UART device	No parameters
HW_UART_READ_U16	Read a 16-bit unsigned integer from the UART device	No parameters
HW_UART_READ_U32	Read a 32-bit unsigned integer from the UART device	No parameters
HW_UART_READ_I8	Read an 8-bit signed integer from the UART device	No parameters
HW_UART_READ_I16	Read a 16-bit signed integer from the UART device	No parameters
HW_UART_READ_I32	Read a 32-bit signed integer from the UART device	No parameters
HW_UART_READ_FLT	Read a 32-bit floating point number from the UART device	No parameters

B.3 Lists Subsystem

Instruction	Description	Parameters
<code>_LIST_U8</code>	Create a list of 8-bit unsigned integers	size:8
<code>_LIST_U16</code>	Create a list of 16-bit unsigned integers	size:8
<code>_LIST_U32</code>	Create a list of 32-bit unsigned integers	size:8
<code>_LIST_I8</code>	Create a list of 8-bit signed integers	size:8
<code>_LIST_I16</code>	Create a list of 16-bit signed integers	size:8
<code>_LIST_I32</code>	Create a list of 32-bit signed integers	size:8
<code>_LIST_FLT</code>	Create a list of 32-bit floating point numbers	size:8
<code>_LIST</code>	Get list with specified id	id:8
<code>_LIST_PARAM</code>	Get list parameter with specified id	id:8
<code>HW_I2C_READ_LIST</code>	Read a list from the I2C device	No parameters
<code>HW_I2C_WRITE_READ_LIST</code>	Write a list to the I2C device, then read a list	No parameters
<code>HW_SPI_READ_LIST</code>	Read a list from the SPI device	No parameters
<code>HW_SPI_WRITE_READ_LIST</code>	Write a list to the SPI device, then read a list	No parameters
<code>HW_UART_READ_LIST</code>	Read a list from the UART device	No parameters
<code>HW_UART_WRITE_READ_LIST</code>	Write a list to the UART device, then read a list	No parameters
<code>HW_DTH_READ_LIST</code>	Read a list from the DHT sensor with specified variant and pin	variant:8, pin:num

LOGICGLUE INTERPRETER

C.1 Platform-Specific Functions

```

//=== GPIO ===//
status_e target_gpio_config(const uint8_t pin, const gpio_mode_e mode);
status_e target_gpio_write(const uint8_t pin, const uint8_t state);
status_e target_gpio_read(const uint8_t pin, uint8_t* state);
status_e target_gpio_read_pulse(const uint8_t pin, const uint8_t state, uint32_t* pulse);
status_e target_gpio_read_pulse_timeout(const uint8_t pin, const uint8_t state, const uint32_t timeout, uint32_t* pulse);
//=== GPIOTE ===//
status_e target_gpiote_config(const uint8_t pin, const gpio_edge_e edge, const gpio_irq_f callback, const uint8_t idx);
//=== PWM ===//
status_e target_pwm_config(const uint8_t pin, const uint16_t period);
status_e target_pwm_write(const uint8_t pin, const uint16_t duty);
//=== ADC ===//
status_e target_adc_config(const uint8_t pin);
status_e target_adc_read(const uint8_t pin, uint16_t* value);
//=== I2C ===//
status_e target_i2c_config(const i2c_freq_e freq, const uint8_t mtu);
status_e target_i2c_write(const uint8_t address, uint8_t* buffer, uint16_t len);
status_e target_i2c_read(const uint8_t address, uint8_t* buffer, uint16_t len);
status_e target_i2c_write_read(const uint8_t address, uint8_t* write_buffer, uint16_t write_len, uint8_t* read_buffer, uint16_t read_len);
//=== SPI ===//
status_e target_spi_config(const spi_freq_e freq, const spi_mode_e mode, const spi_order_e order, const uint8_t mtu);
status_e target_spi_write(const uint8_t cs, uint8_t* buffer, uint16_t len);
status_e target_spi_read(const uint8_t cs, uint8_t* buffer, uint16_t len);
status_e target_spi_write_read(const uint8_t cs, uint8_t* write_buffer, uint16_t write_len, uint8_t* read_buffer, uint16_t read_len);
//=== UART ===//
status_e target_uart_config(const uart_baud_e baud, const uart_mode_e mode, const uint8_t mtu);
status_e target_uart_write(const uint8_t id, uint8_t* buffer, uint16_t len);
status_e target_uart_read(const uint8_t id, uint8_t* buffer, uint16_t len);
status_e target_uart_write_read(const uint8_t id, uint8_t* write_buffer, uint16_t write_len, uint8_t* read_buffer, uint16_t read_len);
//=== DHT ===//
status_e target_dht_config(const uint8_t pin);
status_e target_dht_read(const dht_variant_e variant, const uint8_t pin, uint8_t* buffer);

//=== TIME ===//
void target_sleep_ms(uint32_t ms);
void target_sleep_us(uint32_t us);
uint32_t target_millis();
uint64_t target_micros();
//=== MEMORY ===//
void* target_malloc(uint16_t size);
void* target_calloc(uint16_t num, uint16_t size);
void* target_realloc(void* ptr, uint16_t size);
void target_free(void* ptr);
int target_get_free_memory(void);
uint16_t target_get_memory_counter(void);
//=== INTERRUPTS ===//
void target_disable_interrupts(void);
void target_enable_interrupts(void);
uint8_t target_in_interrupt(void);

```

Figure C.1: Header file detailing the platform-specific functions needed to be implemented when porting LogicGlue to a new platform.

