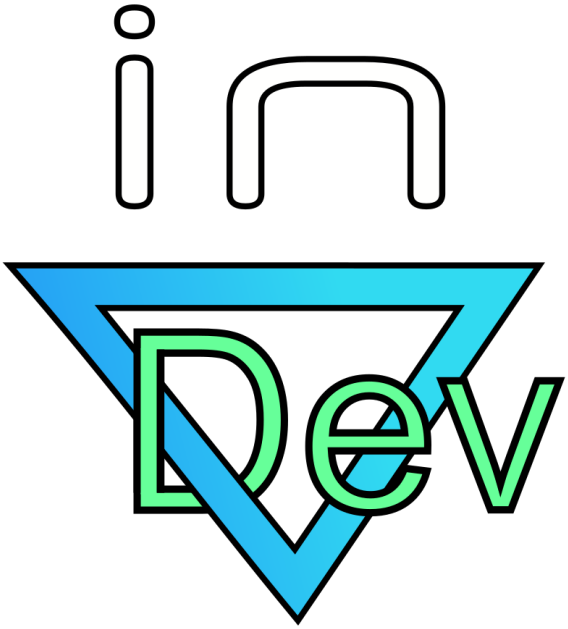
May 2020 Development Debrief

for Infinity v1.0



Project Start Date: 9/15/2019
Project Manager: Chris Stone
Project Status: In Development

Table of Contents

Abstract

While developing another recent project that required a robust system for serial communication, it was determined that USB would be required to interface a device with modern PCs, but that USB is a poor fit for most small CPUs, including the one that the project in question is based upon. As most developers tend to translate from USB to a proprietary TTL interface, the same route was taken for this project. However, unlike typical development for proprietary interfaces, several measures were taken to make this new protocol both accessible and profitable to the general public.

This paper discusses the research, theory, design, manufacturing, documentation, assembly, interfacing, and beta testing of *Infinity*, a new versatile, context-driven serial protocol for use with small CPUs. The process began with a general investigation into modern hardware requirements and then segwayed into the design of the Infinity Portal, the world's first translator from USB Type C to TTL logic, and the library and desktop software required to interface it with the new protocol. From there, an example project was designed and manufactured which would implement the Infinity Protocol as a peripheral to observe and test basic Infinity data exchange with a PC. A digital oscilloscope was used to debug both the peripheral and PC software until it could perform the intended operations.

Finally, a hardware test was performed to evaluate the peripheral's immunity to electrical noise, and the results were synthesized with a software test, performed to quantify the tolerance of the system to bitrate variance and assess the integrity of the protocol over large data transfers. The hypothesis agreed with the results, as all data was transferred without a single error while the transmission bitrate discrepancy fell within the recommended bounds.

1. Introduction

The world around us is filled with serial data, or data which is transferred as a *series* of bits.[1] Such data is transferred in packets between devices, with a variety of hardware interfaces using optical, electrical, and electromagnetic signals to represent the data in transmission.[2] In a recent project, there arose a need to communicate with a device based around a small CPU, the ATtiny1617.[3] The device, similar to its popular cousin, the ATmega328P,[4] has several hardware interfaces to convert data stored in RAM[5] to binary[6] electrical signals, which can then be decoded and loaded into the memory of another processor. However, to communicate with modern PCs, a USB interface will be required.[7] The Universal Serial Bus[8] (USB) system is differential,[9] and the required hardware is complex and difficult to implement on CPUs like those in the ATtiny and Atmega series.[10] Thus, for these devices, a translator between USB and some other protocol designed for the speed and hardware of smaller CPUs must be used to communicate with modern PCs.[11] Most companies choose to develop a proprietary protocol for interfacing between their devices and software, and where external compatibility is required, they try to make most important features work over the SPI, I2C, or CAN protocols.[12] [13] [14] [15] These protocols are rarely a first choice, as implementing them drives up development cost and hardware complexity.[16]

For the specific needs of the project in question, all the aforementioned protocols are poor fits due to overhead,[17] synchronization,[18] or the requirement of several wires to transmit data. Asynchronous data transfer[19] has the advantage of fewer wires and less hardware limitations, as transfer speed is not dependent on what external clock speeds a device can generate. Thus, an asynchronous protocol, tailored to the specific needs of the project must be developed. Such a protocol should feature standardized basic functionality that may be

implemented on demand by the application programmer, but also allow the device to be set into a configuration where data can be interpreted in a pre-defined context. This removes all overhead once the device has been set up. As not to create yet another proprietary protocol that remains hidden from the general public, this protocol could be documented and released to the public with small developers in mind, who will likely also benefit from its framework, allowing development to begin immediately, where default functionality like descriptors[20] can be implemented later when the more critical routines are nailed down.

Releasing the protocol to the public will require desktop software to abstract the driver library[21] that interfaces the USB translator, documentation of the protocol and its specifications, and an example project to demonstrate and test the protocol's working capability. It will also be beneficial to make a website where all the protocol's documents and software can be collected and served.

This paper debriefs the current development state of Infinity, a new versatile, context-driven serial protocol which gives power and flexibility back to developers while providing a standard framework for device-to-device communication. The paper's discussion ranges from the theory behind the protocol to the design and documentation of the various hardware and software tools required for its implementation. Finally, it outlines the tests were performed to quantify the electrical reliability of the hardware, and the extensive data transfers that were conducted under the protocol to verify the integrity of the system when faced with reasonable variations in the transmission bitrate between devices. If the bitrate variation between the transmitter and receiver remains within ±4%, a reliable hardware connection is established, and no unreasonable electrical noise is imposed upon the system, then even very large data transmissions will occur with little or no error.

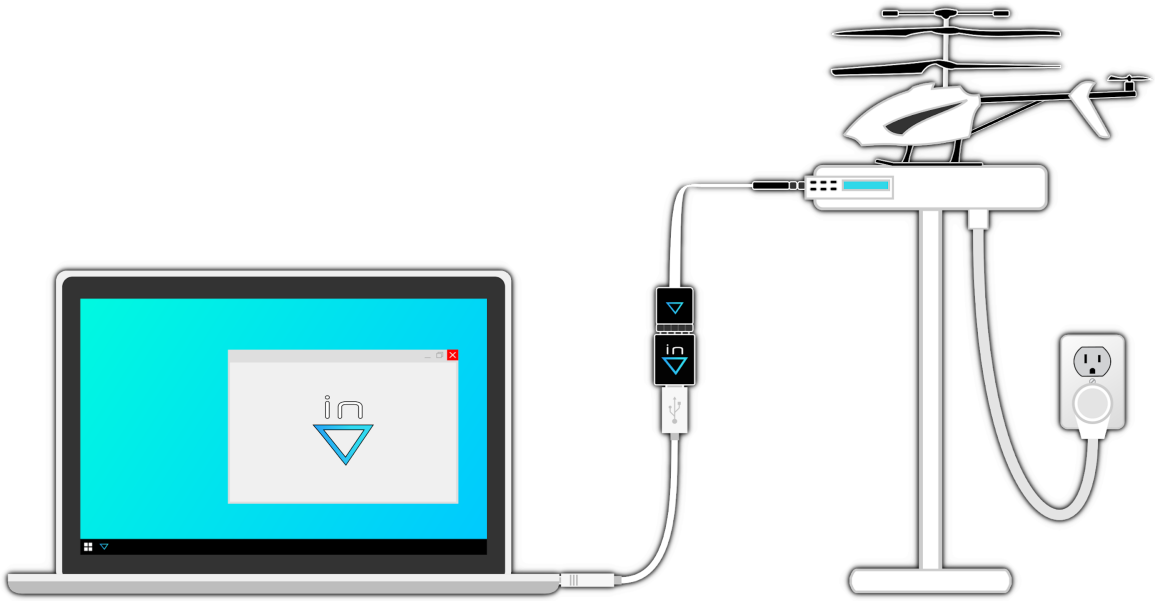
2. The Need for a New Protocol

USB has virtually eclipsed its competitors in PC communication. Both PCs and devices designed for them tend to feature only USB hardware, with RS232[22] and parallel[23] ports being phased out of newer computer models.[24] As a result, devices that are neither designed for, nor could benefit from USB must use a translator between the complicated, high-speed, differential protocol and a protocol natively supported by the device. Because most devices have custom communication requirements, companies tend to develop their own protocols to translate to USB and communicate between their devices. Other published alternatives include the Inter-Integrated Circuit protocol (I2C), developed by Philips Electronics, the Serial Peripheral Interface (SPI), and the Computer Area Network (CAN) protocol. Each of these has its disadvantages, however, such as limited transfer speeds, peculiar hardware requirements, and mandatory synchronization.

Asynchronous data transfer, on the other hand, has the advantage of fewer wires and independence of clocks, given that transmission and evaluation frequencies are within a certain margin. Changing clock speeds mid-transmission is also easily achieved without a requirement for synchronization. When it comes to asynchronous serial protocols, however, the Recommended Standard[25] series of protocols is the only prominent system, which requires support for higher voltages than most small devices can easily produce without extensive external hardware. These standards, as previously mentioned, have been phased out of most modern computer systems for this reason. As a result, generic USB to Transistor-Transistor Logic (TTL)[26] convertors are designed for converting USB data to asynchronous 3.3v-peak logic signals for use with small CPUs. Any defined data exchange system (protocol) development is left to the developer of the final device being interfaced. There is clearly a need for a new protocol for asynchronous communication between such devices.

3. The Hardware Design

Figure 3.1
An abstract view of a typical Infinity setup



3.1 The Infinity Portal

Figure 3.1.1
Converting USB Type-C to Infinity via the Infinity Portal

The Infinity Portal bridges the gap between USB Type C and infinity devices. This allows for easy connection to a PC. After selecting the FT230X[27] from Future Technology Devices International[28] (FTDI) as the CPU for the translator board, supporting components were selected

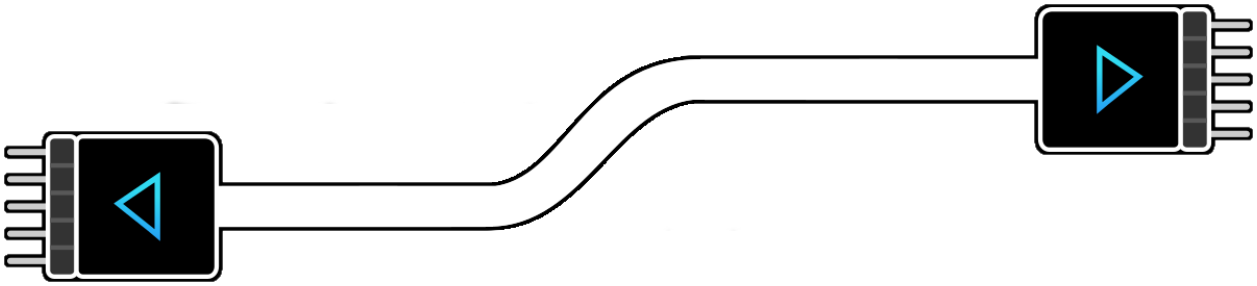
and identified, a Bill of Materials[29] (BOM) was compiled, and a Printed Circuit Board (PCB)[30] hardware layout was developed. Altium Circuitstudio[31] was used to develop and convert the schematics[32] and Gerber[33] files for the project. After the development was complete, the circuit board files were sent to the JIALICHUANG Electronic Technology Development Co., Ltd.[34] of Shenzhen China for production, and orders for all BOM components were placed online at the Shenzhen LCSC Electronics[35] marketplace.

Upon arrival, it was evident that the manufacturer had made a fatal error during circuit board production, making several holes on the board the wrong shape and size. The circuit boards were manually modified, drilling out the correct area to make room for the components. A modified board was selected, paste solder[36] was administered to the soldering surface, the components were secured in place, and the board was heated to 300° C using a reflow[37] gun. The board was allowed to cool and inspections of the mechanical connection were made with an Optical Inspection Microscope[38] (OIM). Poor or unformed mechanical connections were touched up manually with a TS100 digital soldering iron[39] using a custom bit. The Infinity receptacle was glued to the board, soldered in place manually, and the entire board was cleaned in isopropyl alcohol.

The new board was plugged into a PC via a USB Type C cable, and the portal showed up in the device manager. The descriptors were customized using the FT_Prog desktop application[40] from FTDI, and the circuit board was encased in electrical tape for reduced vulnerability to damage from static electricity.

3.2 Cables and Signals

Figure 3.2.1
The 5-Pin Infinity Cable



A suitable cable was developed for connecting Infinity devices. A plug and receptacle system with rotational and connectional symmetry was developed using machined male and female rounded pin headers[41] from Boom Precision Electronics[42] (BOOMELE). The electric signal lines were assigned to the pins and their corresponding receptacles as outlined in the table below.

Table 3.2.1
Signals on the Infinity Bus

Pin	Signal	Description
Outer Pins	**TX**	The transmit line, from the perspective of the receptacle.
Inner Pins	**RX**	The receive line, from the perspective of the receptacle.
Center Pin	**GND**	The common reference potential for the data signals.

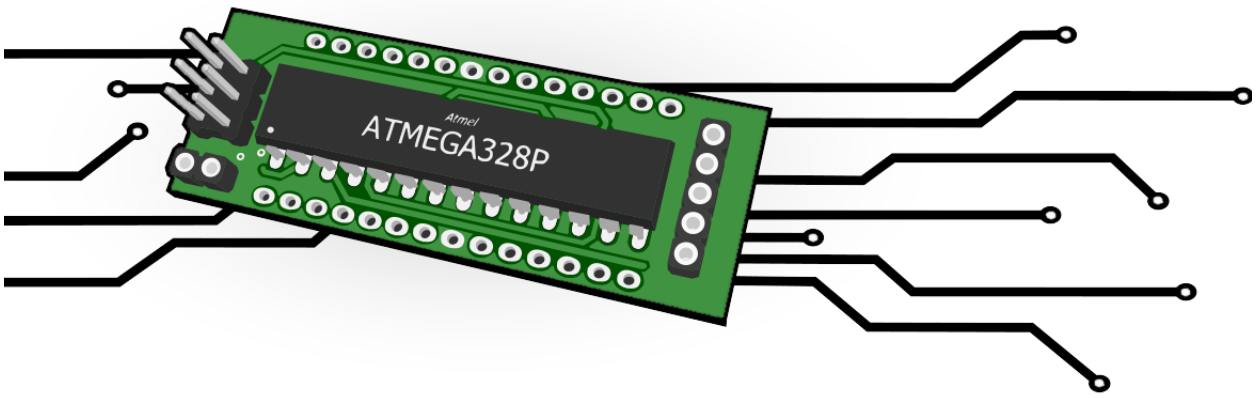
Table 3.2.2
Infinity Bus absolute ratings

Variable	Description	Unit	Min	Max
D_L	**Data Logic Low Signal**	**V**	**GND**	**GND + 0.5**
D_H	**Data Logic High Signal**	**V**	**GND + 2.5**	**GND + 5.5**
C_D	**Capacitance of Data Lines**	**pF**	**-**	**50**

Data signal ranges, as outlined in **Table 3.2.2**, can vary up to 5.5v, although it is recommended that they peak closer to 5v. Because most CPUs consider anything 2v or greater to be a logical 1 and anything 0.8v or less to be a logical zero (with a liberal hysteresis[43] margin in between), Infinity standards merely require all logical 1 signals be 2.5v or greater, and all logical zero signals be 0.5v or less. If a logical unit intended for use with Infinity cannot meet these ratings and requirements, additional hardware must be used to do so. Line capacitance must be kept at a minimum for fast data transfer.

3.3 The Example Project

Figure 3.3.1
The Example Project



The project that inspired Infinity's development is too complicated to benefit an exposition of the protocol's capabilities. Thus, the Infinity Example Project was developed, based on the ATmega328P, a popular hobbyist-friendly CPU which comes in large Dual Inline Packages[44] (DIP). The controller features a Universal Asynchronous Receiver and Transmitter[45] (UART), a digital hardware peripheral that bridges the gap between clocked shift registers[46] and regular registers in the CPU's data space.[47] Like the Infinity Portal, the example project was

designed in CircuitStudio and orders were placed at JIALICHUANG and LCSC. However, due to complications in China from the recent outbreak of COVID-19, the order was heavily delayed.

During the order's extended production, software and assembly instructions were developed for the example project along with the new Hiovita Supporting Files Licence[48] (HSFL). The license outlines the use conditions for supporting files which apply to developers who download supporting files like those that make up the Infinity Example Project from the Hiovita Official Downloads[49] page. This was another step in making the protocol practical and available for other developers around the world.

Upon arrival, the example project components were manually soldered to the circuit board, as the design was framed with the consideration that some developers may not have reflow equipment on hand. After assembly, the board was cleaned with isopropyl alcohol and the power cable was soldered together and plugged in. The board was plugged into a breadboard using pin headers, and the optional LEDs[50] were added to the setup. The blue LED turned on on plug in. An Atmel AVR ICE [51] was then plugged into a PC and attached to the ISP[52] socket on the device. Atmel Studio[53] was used to load the example Infinity peripheral firmware onto the ATmega328P, and the red LED turned on upon completion. The assembled example project was then unplugged and set aside until the hardware test.

4. The Infinity Protocol

The contextual nature of the Infinity protocol comes from a simple framework. All data transmission is evaluated within a *context*, or a set of predefined rules for that data's interpretation. Because the rules are predefined by the software engineer, there is no need for overhead; the controller is free to change between data evaluation contexts throughout the

exchange of data, and where more complicated or time-sensitive routines must be performed, multiple contexts can work together to achieve an end result.

4.1 Device Types

Infinity Beta 1.0 recognises several device types based on the roles they play in data transfers. This list may be updated in the future. Note that all devices must be self-powered.

Table 4.1.1
The various Infinity devices and their functions

Device	Function
Controller	Controllers initiate and maintain data transmission. Controllers can use one or more peripherals in helping accomplish a task, such as reading the ambient temperature, stimulating a system, or interacting with users and the environment. Systems will typically have one controller and many peripherals. A controller may enter **peripheral mode** to communicate with other controllers.
Peripheral	Peripherals aid controllers in accomplishing a task. They only send data on demand, and have several contexts which controllers can use to set up an ideal data exchange scheme.
Portal	A portal is a controller strictly meant for connecting Infinity peripherals to a PC or other USB-enabled device. It cannot enter into a peripheral mode, thus all controllers must do so to use it.
Interpreter	An interpreter is a device that mediates several Infinity controllers, peripherals, and portals where limited cables are required, or multiple controllers need to frequently access the same peripherals. Interpreters can mediate devices in frequent use or establish dedicated channels between devices. The interpreter will undergo greater development in later versions of Infinity.

4.2 Device Status

Peripherals must keep controllers aware of their current state. After each instruction sent to a peripheral is evaluated, it must determine whether to stay in its current state or change to another state, and notify the controller of the change. Peripherals do so by sending status codes. Controllers are responsible for reading a peripheral's status codes and executing fallback routines if the status is not what is required.

Table 4.2.1
Supported device status codes and their descriptions

Code	Name	Description
0x0	**RDY**	The peripheral is ready to enter a new context.
0x1	**ERR**	The peripheral has encountered an error. The controller should restart communication.
0x2	**UNIMP**	The requested context is unimplemented; the peripheral has returned to the RDY state.
0x3	**LCKD**	The peripheral is locked, and must be unlocked before sending data or commands..
0x4	**STRM**	The peripheral has entered a context that expects an indefinite amount of data.The context can only be left through the use of the **stream escape sequence**.
0x5	**BLK**	The peripheral is in a context that expects a definite amount of data. The peripheral will enter the RDY state once all the data has been sent.
0x6	**UNKNOWN**	This code is used to signal the end of the status codes. Any code a controller receives of equal or greater value will be considered invalid. Similarly to the ERR code, the controller should restart communication. Peripherals should never send a code of equal or greater value intentionally.

4.3 Device Contexts

Contexts will always reside at a specific address between 0x0 and 0xFF. When a peripheral is in the RDY state, a controller can select the context by sending the context address to the peripheral. The first 16 addresses are reserved for Infinity default features, which will be further developed in the future. All developer-defined stream contexts should be placed directly after the Infinity contexts. All developer-defined block contexts should begin directly after the stream contexts. When a controller requests an address that has no context routine associated with it, the peripheral should return an UNIMP code and return to the RDY state.

Table 4.3.1
Infinity context arrangement by address, where **$STRM_{end}$** and **BLK_{end}** are the addresses of the last stream and block context respectively

Address	Contexts	Description
0x00- 0x0F	**Infinity Contexts**	Reserved space for future default features. When requested, UNIMP should be sent.
0x10 - $STRM_{end}$	**Stream Contexts**	The stream section. When requested, STRM should be sent.
($STRM_{end}$ + 0x01) - BLK_{end}	**Block Contexts**	The block section. When requested, BLK should be sent.
(BLK_{end} + 0x01) - 0xFF	**Unused Space**	Any leftover address spaces. When requested, UNIMP should be sent.

4.4 Locked Devices

The **locked context** is a special stream context that has no address. The context can be left by sending the **unlock sequence**. When the sequence is received, the peripheral should send the RDY status code and prepare to enter the context the controller selects. If anything other than the escape code is sent, the peripheral should send the LCKD status code. If the escape code is received, but followed by something other than the unlock command, it should send the ERR status code, but remain in the locked context and continue looking for another escape code, as all devices that encounter errors must return to the locked context. Each peripheral must enter the locked context on restart and boot.

4.5 Stream Commands and the Escape Sequence

In stream contexts, any byte can be used as data, with no reserved characters. However, this creates a need for a method of leaving a stream, as unlike block contexts, stream contexts have no definite point after which data transmission ends: they can be active for a microsecond or a month. The **escape code** can be used to evaluate the next byte as a command. The code, hexadecimal **0x55**, was carefully chosen because of its binary equivalent representation. When sending this code, the data signal will resemble the longest perfect square wave possible. This pattern, when followed by a valid command, is almost impossible for pseudo-random electrical noise to produce, and prevents peripherals from being unlocked on plug in. The two commands currently supported are the **unlock command** and the **leave stream command**. Sending these commands after an escape code can unlock a peripheral or leave a stream context respectively. The peripheral should send a RDY status code upon successful command execution. Peripherals that encounter unsuccessful command execution

or an invalid command code after an escape character should return the ERR status code and enter the locked context.

Table 4.5.1
Infinity codes and commands

Code	Name	Function
0x55	**Escape Code**	Indicates to a peripheral that the next byte is a command.
0x00	**Unlock Command**	Instructs a peripheral to leave the locked context.
0x01	**Leave Stream Command**	Instructs a peripheral to leave a stream context.

Because it is likely that controllers will need to send a byte with the same value as the escape code to be evaluated in the current stream context, an escape character can be escaped by sending another escape character before it. When two escape characters in a row are encountered by a peripheral, the first one should be discarded, and the second passed on to the current stream context. Controllers must never send a byte equal to the escape character to a stream context without escaping it. Peripherals do not pass commands to a master, so data returned from a peripheral will never be escaped. Escape codes and commands have no function in block contexts.

4.6 Hardware Notes

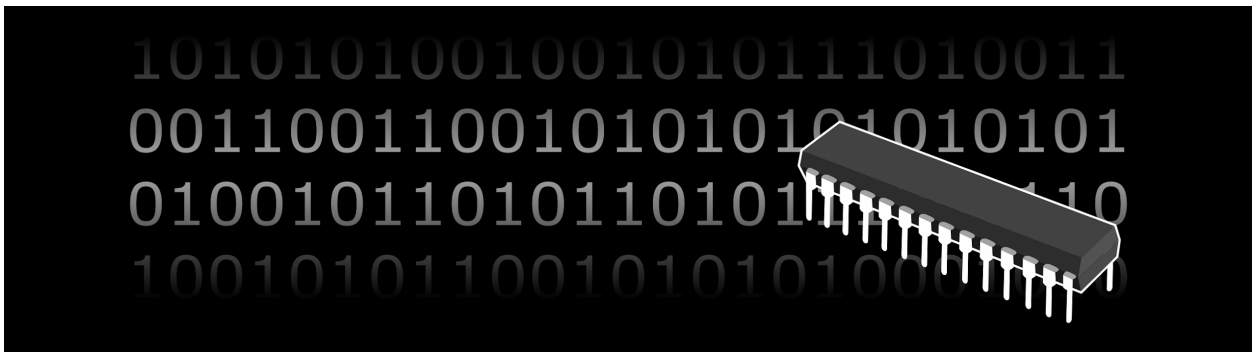
In addition to the considerations already outlined in the Cables and Signals section of the Hardware Design, it should be noted that all transmit and receive signals must be active high. The protocol does not feature a parity, preferring software verification of data integrity for critical exchanges, and thus any parity UART function must be turned off. After a low start bit, eight data bits must be sent, followed by one stop bit from a peripheral and two stop bits from a controller. Because a peripheral can only send bytes of data when requested by the controller, the controller’s extra stop bit allows the peripheral’s accumulation to reset, preventing error accumulation due to slight imbalances in transmission bitrates between devices.

5. The Software Design

Once the hardware was complete, several software packages had to be developed. Custom firmware was required for the example project, a static Windows™ library[54] for the Infinity Portal, and a Windows™ desktop application[55] for end-user use. Automation, efficiency, and professionalism were central focuses of the development process.

5.1 The Device Firmware

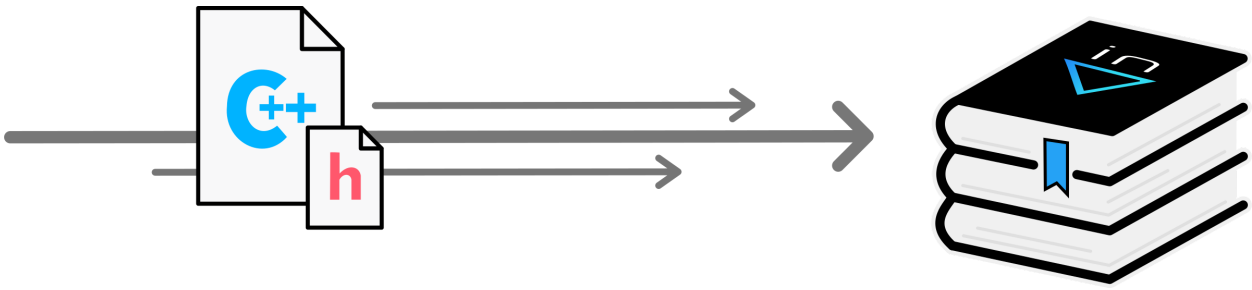
Figure 5.1.1
The 28-pin ATmega328P



Custom device firmware was developed for the ATmega328P in AVR assembly[56]. The program utilizes the registers of the device to set interrupts and handlers that interface the built-in UART hardware to act as an Infinity peripheral.

5.2 The Static Library

Figure 5.2.1
A graphical symbolization of a static library packaging



Static libraries[57] are collections of compiled code meant for inclusion in future programming projects. In an effort to document and support Infinity, a static c++ library was developed to abstract Infinity for other developers to use. The static library was later posted on the Infinity Website and used to build the Infinity Portal App.

5.3 The Portal App

Figure 5.3.1
The interface of the Infinity Portal App

The Portal App was developed in Visual Studio Community 2019[58] using a mixture of C[59] and C++.[60] The static library was abstracted and routines were written to account for portal, peripheral, and user error. A visual interface was developed through the Ultralight™ GPU[61] framework using the Hypertext Markup Language (HTML)[62], Cascading Style Sheets[63] (CSS), and Javascript[64] in Visual Studio Code[65]. Graphics were developed for the app using the GNU Image Manipulation Program (GIMP)[66] and Inkscape[67] for raster[68] and vector[69] graphics manipulation respectively.

The app is currently console-based, and the console can be opened by pressing the console icon. In the console, commands for connecting portals, communicating with devices and automatically testing the setup can be entered. **Table 5.3.1** lists the current commands supported by the Portal App, their parameters, and parameter requirements. Feedback about command execution will be provided after the execution, if any, completes. Possible feedback messages types are shown in **Table 5.3.2**. Note that all commands are executed by typing **portal** followed by the command and each parameter, separated by spaces.

Table 5.3.1
Portal App commands and parameters

Command	Parameters	Function and Requirements
clear	-	Clears the console.
list	-	Lists all portals connected to the PC with their ID and port. Excludes portals in use by other apps.
open	**portal_id** *	Opens the portal with the entered ID for communication and resets any devices connected to it. The portal can no longer be used by other apps once opened. **portal_id** must be a valid ID of an available portal connected to the PC.
close	**portal_id** *	Closes a portal opened earlier, making it available for use by other apps. **portal_id** must be a valid ID of an open portal.
unlock	**portal_id** *	Sends the unlock command to the device through the portal. The command will fail and reset the device if it is not in the locked context. **portal_id** must be a valid ID of an open portal.
changecontext	**portal_id** *, **context_code** *	Selects a new context on the device through the portal. The command will fail if the device is not in the ready state. **portal_id** must be a valid ID of an open portal. **context_code** must be a valid context address for the connected device.
leavestream	**portal_id** *	Leaves a stream context on the device through the portal by sending it the leave stream command. The command will fail if the device is not in a stream context. **portal_id** must be a valid ID of an open portal.

* required parameter

Command	Parameters	Function and Requirements
setspeed	**portal _id** *, **bitrate** *	Currently for testing purposes only. Sets the bitrate of the portal to the provided value. The command will fail if the device through the portal is not in the ready state. Note that all values entered will be floored to the nearest supported bitrate on the Infinity Portal. **portal_id** must be a valid ID of an open portal. **bitrate** must be an integer between 5000 and 3000000
test	**portal_id** *, **test_count** *	Currently for testing purposes only. Tests an Infinity Example Project through the provided portal using a predefined testing routine. The command will fail if the device is not in ready mode, or any single test encounters a testing error. Each test will attempt to send 256 bytes to each of the test contexts implemented in the example project software, and verify how many bytes were returned correctly. Successful testing will continue until the specified number of tests have completed, and testing results will appear in the console. The device will be returned to the ready state after the testing completes. **portal_id** must be a valid ID of an open portal. **test_count** must be a positive integer.
sendblock	**portal_id** *, **data** *	Sends the user-entered data through the portal. The command will fail if the connected device is not in a data context. A max of 256 characters can be sent at a time, and all non-standard characters will be removed before sending. Note that the device does not need to be in a block context to send block data. If the device is in a stream context, all escape characters in the block will be doubled and escaped, and the resulting block will be sent to the device. **portal_id** must be a valid ID of an open portal. **data** must be a string of one to 256 numbers, letters, or basic symbols.
colors	-	Lists all the colors in **Table 5.3.2** and what they signify.

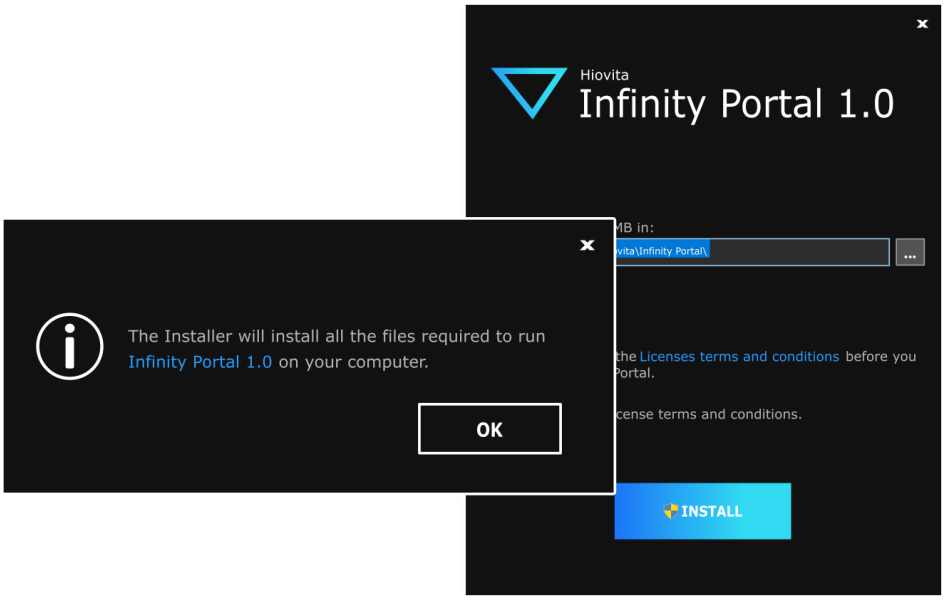
* required parameter

Table 5.3.2
Portal App message colors and their significance

Color Code	Signification	Description
#EEEEEE	Information	General information related to a command's execution.
#8066FF	Critical Information	Important information that has been highlighted.
#66FF99	Success	Informs the user of successful command execution.
#FFBF67	Warning	Warns the user of unexpected or unusual events or parameters during execution.
#FF7366	Error	Informs the user that the execution has failed and why.

5.4 The App Installer

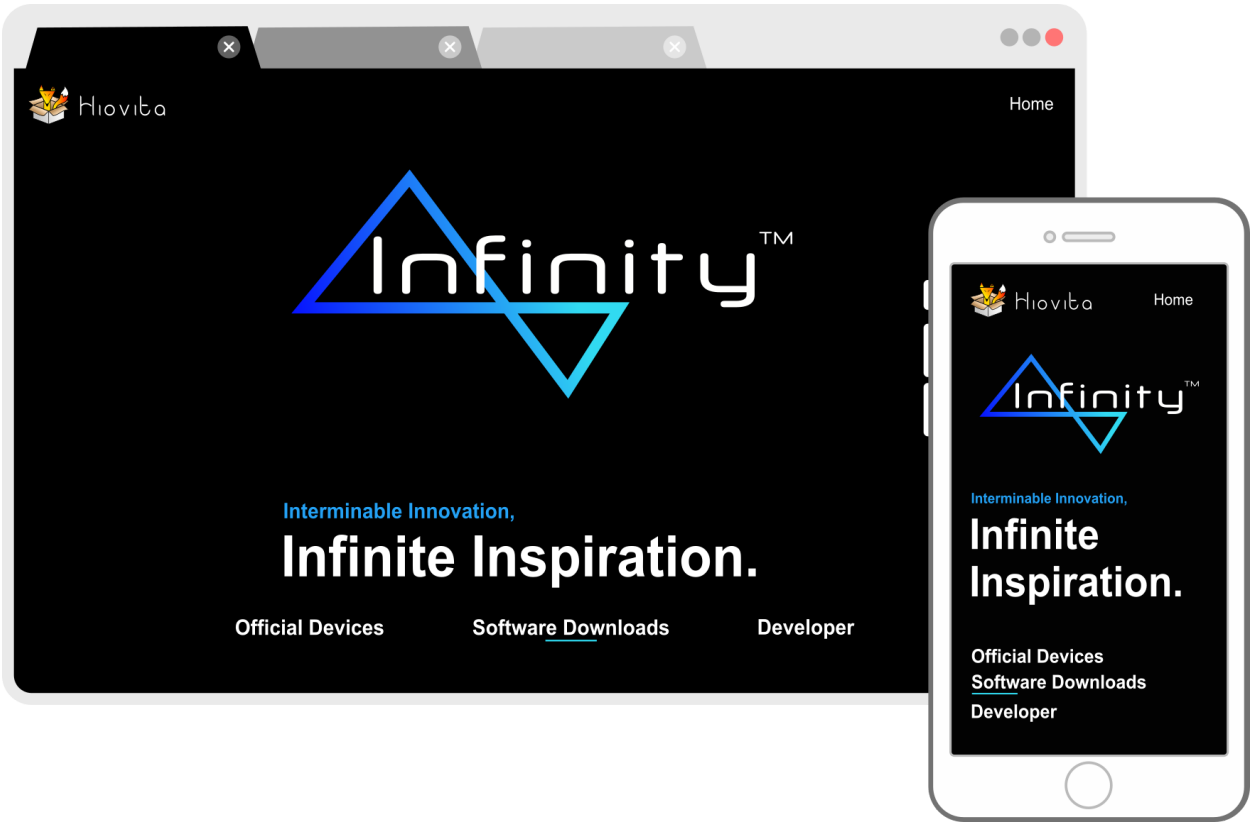
Figure 5.4.1
Interface components of the Infinity Portal App Installer



Installing an app on Windows 10 is more challenging than ever. Countless hardware and software requirements must be met for the program to function correctly, and all files must be placed in the correct location. To allow the average person to use the Portal App, an App Installer was designed and built to automatically collect and install all the required files.

6. The Infinity Website

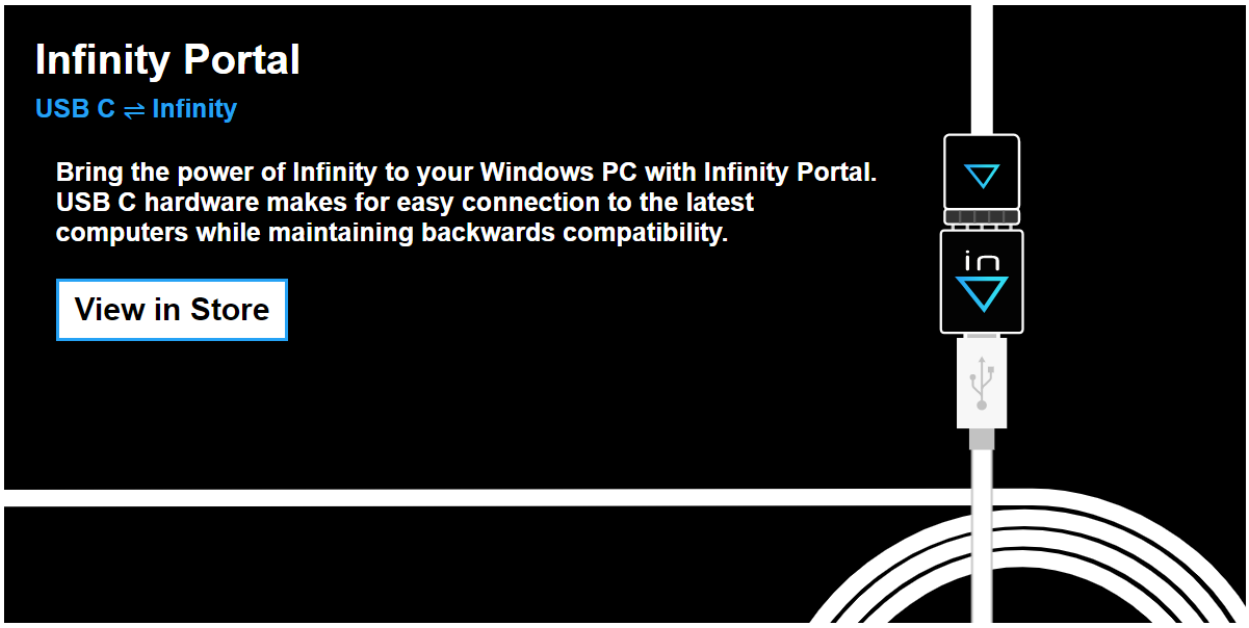
Figure 6.1
The desktop and mobile version of the Infinity Website



A website was developed to document items related to the Infinity Protocol. Files, icons, and markup was custom made for the site, which was programmed from scratch without the aid of development tools. Located at https://hiovita.net/infinity, the website contains several sections which provide resources to developers and end-users.

6.1 Official Devices

Figure 6.1.1
A block for the Infinity Portal which appears on the Official Devices section



The Official Devices section serves as a digital home for information about the Infinity Portal and any other official Infinity-compatible devices. It features a link to the Hiovita Store[70] where future pre-assembled devices can be purchased.

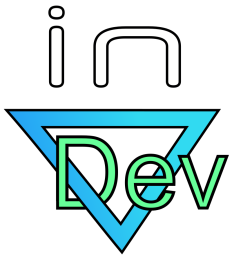
6.2 The Downloads Section

Figure 6.2.1
The Hiovita Downloads banner

In an effort to make the Infinity Portal App more accessible, a downloads section for the app was added to the site. The link will take users to the Hiovita Official Downloads[71] page, where they can download an app Installer for Windows 10.

6.3 The Developer Program

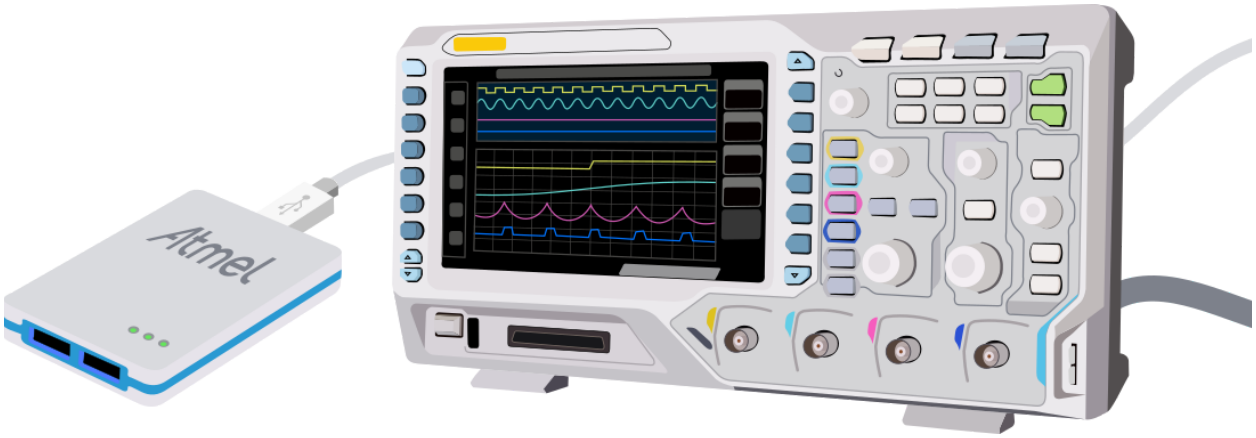
Figure 6.3.1
A logo for the Infinity Developer Program



A developer program was drafted to aid new designers looking to implement Infinity. In addition to the Static Library, Example Project and Implementation Specifications, the Infinity Website will eventually feature a developer identification program which provides cost-effective cloud services and drivers.

7. The Hardware Test

Figure 7.1
A digital oscilloscope and Atmel ICE programmer, used throughout the experiments



Although not the central focus of the system evaluation, the hardware test will help set up the software test, and verify the presence of the reliable hardware connection required for successful software testing. The hardware test will also assess the resistance of the system to being unlocked by random electrical noise on plug-in.

7.1 Setup

1. Obtain an assembled Infinity Example Project, or order and assemble one using the link on the Developer section of the Infinity Website.

2. Power the example project using the power cable. If the ATmega328P CPU has not been preprogrammed with the device firmware, follow the programming instructions in the example project's package to install the firmware on the device using a specialty programming tool. If the optional LEDs were added to the setup, the blue and the red LED should turn on after the device firmware is run.

3. Connect the Infinity Portal to the example project using the Infinity Cable

4. Now plug the Infinity Portal into the PC via a USB Type C cable. Wait a few seconds for the PC to recognise it and install the drivers.

5. Download the latest version of the Infinity Portal App Installer from the Infinity Website.

6. Run the installer, accept the license agreement, and open the app.

7. Click on the console icon to open the console.

8. Type **portal list** in the console to see a list of available portals. The one that was just plugged in should appear.

9. Power up a digital oscilloscope and pull the Infinity Cable out of the example project's receptacle just enough to expose the pins. Connect a probe to the center pin; this is the receive line for the device. Connect another probe to one of the outermost pins. This is the transmit line of the device.

7.2 Procedure

1. Using the oscilloscope, observe the data lines. Determine whether there is an unreasonable amount of electrical noise on either signal.

2. Observe the voltage level on the transmit line. It should be no less than 4.9v and no greater than 5.5v.

3. Do the same for the receive line. Its voltage should be between 2.9v and 3.3v.

4. Now unplug the Infinity Cable from the Portal and plug it back in several times. Observe the oscilloscope as each insertion and removal creates large amounts of electrical noise. Some of the noise spikes will inevitably trigger the device to begin reading the line for a new data frame. Upon completing the evaluation, the device software will send a message that the device is locked if the data read is anything other than the unlock sequence. This should look like a quick low pulse on the transmit line, followed by a high section twice as long, and then a long low section. If the optional LEDs were added, the red LED can be observed instead to determine if the device is still locked.

5. Continue plugging the cable in and out until it has been inserted and removed 1000 times. If the device unlocks, record it, unplug the device, and plug it back in to lock it again.

6. Now turn back to the Infinity Portal App. Type **portal open** followed by the id of the portal that was listed earlier. The app should open the portal for communication.

7. Type **portal unlock** followed by the id of the portal to prepare the device for software testing. The red LED should turn off and the green LED should turn on if the optional LEDs were added to the setup.

8. Make a statement about the setup's resistance to hardware noise on plug-in.

8. The Software Test

The digital abstraction was developed to distill an infinitely complex electrical world into two distinct values - 1 and 0. This makes it possible for a software test to come back with perfect accuracy. USB, for example, is designed to never have a hardware error if possible. The Infinity Protocol's success rests on its ability to facilitate error-free data exchange. Thus, the software test sends a large amount of data from a PC at various bitrates and reports errors where data was lost or returned incorrectly from the testing device to determine the system's ability to transmit data without errors.

8.1 Hypothesis

If the bitrate variation between the transmitter and receiver remains within ±4%, a reliable hardware connection is established, and no unreasonable electrical noise is imposed upon the system, then even very large data transmissions will occur with little or no error.

8.2 Setup

1. Type **portal test** followed by the id of the portal and at least 5 for the number of tests. As data is being transferred, zoom in on the data with the oscilloscope and select **frequency** from the horizontal menu on both channels.

2. Switch the oscilloscope to **single mode**, freezing the display. Record the frequency of the device's transmit line and double it to get the transmission bitrate, as bits are transmitted on both the high and low edges of the wave. The value should be around 9600 bits/sec.

3. Given that the temperature, humidity, and the voltage powering the device remain constant, the device transmission bitrate drift should be negligible. Use the previously-recorded transmission bitrate as a reference and develop a list of various rates within ±4.5% of this value.
4. Now calculate and record a bitrate 8% greater and 8% less than the reference. These values will be used as a control.

5. Type **portal clear** to clear the console.

8.3 Procedure

1. Type **portal setspeed** in the Portal App followed by the id of the portal and the calculated bitrate that was 8% greater than the reference in bits/sec.

2. Type **portal test** again, followed by the id of the portal and 1000 for the number of tests.

3. Testing could continue for several minutes. If the testing completes successfully, record the number of bytes sent and the number of errors for each of the three sub-tests. Record any length errors.

4. Repeat steps 1-3, this time using the calculated speed that was 8% *less* than the reference.

5. This completes the control testing. Type **portal clear** where necessary to clean up the console after tests.

6. Now repeat the test for each of the rates on the list of rates within ±4.5% of the reference. The green and yellow LEDs should turn on and off throughout the test if the optional LEDs were added to the setup. Record all the results in a table.

7. Synthesize the results of the software test with the hardware test, and use them to evaluate the reliability of both the setup and the protocol.

8.4 Results

Table 8.4.1
The software test results

Type	Bitrate	Variance	Bytes Sent	Errors				
				Echo	Add	Subtract	Length	Total
Control	10195	+8%	N/A	N/A	N/A	N/A	N/A	N/A
Control	8685	-8%	N/A	N/A	N/A	N/A	N/A	N/A
Test	9440	0%	768,000	0	0	0	0	0
Test	9534	+1%	768,000	0	0	0	0	0
Test	9346	-1%	768,000	0	0	0	0	0
Test	9629	+2%	768,000	0	0	0	0	0
Test	9251	-2%	768,000	0	0	0	0	0
Test	9723	+3%	768,000	0	0	0	0	0
Test	9157	-3%	768,000	0	0	0	0	0
Test	9817	+4%	768,000	0	0	0	0	0
Test	9062	-4%	768,000	0	0	0	0	0
Test	9864	+4.5%	768,000	0	0	0	0	0
Test	9015	-4.5%	384,000	210	237	146	0	593

9. Discussion

The hardware test was a success. The voltage on each signal line was within the required range and the signals were free of excessive idle noise. While plugging the Infinity Cable in and out 1000 times, and generating significant electrical noise in the process, the red LED remained on throughout the test, as the device could not be unlocked by electrical noise. Upon typing the unlock command within the Portal App, however, the device was immediately unlocked and ready for data transfer to begin. These results provided a solid basis for the software test by verifying the existence of a reliable hardware connection and the absence of excess electrical noise.

During the software test, the Portal App threw errors when trying to test the control bitrates with a ±8% variance from the device bitrate. The App verifies each data transfer with the device multiple times during the test and refuses to continue sending data when repeated verification errors occur. The control test results clearly set the precedent that data transfer well outside the 4% bitrate margin set by the hypothesis is error-prone and unreliable, as well as confirming the app security's effectiveness against sending corrupt data.

Although the positive 4.5% test came back without error, the negative 4.5% test revealed several transmission errors, and had to be broken up into 10 parts of 50 tests with cool-down periods for the CPU in between. This was necessary because the 1000-test block refused to complete in its entirety and return any results when a testing error was encountered in the block. As more tests were run, the number of errors gradually rose, until tests began to fail. After 500 tests were reached, the battery level of the PC had drained down, and it was plugged in to continue the testing. At this point, even single tests could not be completed without testing errors, and no further results could be collected. It is likely that the PC's USB voltage rose slightly upon plug-in and this change carried into the USB block that the testing device was connected to. According to the ATmega328P datasheet, the frequency of the internal oscillator has a positive correlation with the supply voltage and temperature.[72] This both fits with the results of the experiment and qualifies the assumption that the device transmission bitrate will not change significantly throughout the experiment, as the errors occurred on the negative-going threshold of the Infinity Portal bitrate and the errors increased with each test as the CPU heated up. Near the thresholds, even a tenth of a percent variation in the bitrate from clock drift can corrupt the data transmission.

The tests within the 4% margin, however, all came back without any errors, and the hypothesis was verified to be correct. Given that the controller in an Infinity data exchange has a

clock with at least 1% accuracy (the Infinity Portal in this case), the peripheral (such as an ATmega328P) can have a clock accuracy that drifts up to 3% away from its nominal frequency without posing a threat to the integrity of transferred data. This allows peripherals to utilize cheaper CPUs and cut down on external electronic components: critical requirements for simple and compact devices.

According to the datasheet for the ATmega328P, similar to all CPUs of its class, the frequency of the internal oscillator will not vary outside of 3% of its nominal value when the recommended voltage is applied.[73] Newer CPUs with silicon oscillators have even better ratings, and in the case of an inadequate oscillator rating, external quartz crystal oscillators[74] and oscillator calibration[75] can be used to bring the frequency variance within an acceptable range for data transmission.

Combined with the results from the hardware test, the software test results lay the foundation for further Infinity Protocol development by confirming the hardware capabilities of the Infinity Example Project and the Infinity Portal. From here, further development of default Infinity features, desktop automation, and example software can proceed with confidence that the central requirements of the protocol have been met.

10. Conclusion

Infinity development is ready to begin its next stage. With experimental results verifying the reliable hardware and software capabilities of both the protocol and the example setup within a wide margin of bitrate error, desktop automation, developer registration, aesthetic upgrades and further development of Infinity's default features can begin. This will serve both the project that inspired the protocol's development and the projects of other developers in need of an asynchronous protocol for small CPUs.

10.1 Project State

The Infinity Portal proved itself to be an invaluable hardware accessory for converting between the latest USB Type C-enabled devices and the new Infinity Protocol throughout the testing. It is unlikely that it will need any further hardware development. Because it was designed for use with any serial protocol, not just Infinity, even those who have their own communication scheme can interface it as a generic device through FTDI's virtual COM port drivers, widening its marketable audience.

Through the Infinity Website, developers around the world can view and download resources to get the most out of Infinity for their next small CPU project. The plan for a reasonably-priced developer program will make it possible for small start-ups to gain a professional presence in the electronics industry without paying outrageous fees for basic features. This gives Infinity the potential to break the cycle of reserved, proprietary asynchronous TTL protocols.

With the Infinity Portal App, it has never been easier to test an Infinity connection from a PC. Useful functions for unlocking devices, setting the transmission speed and context, sending data, and automated testing make working with Infinity devices a breeze, and enabled the efficient collection of the critical data that was used to quantify Infinity's capabilities.

With context-driven data exchange, developers get maximum efficiency and an array of default features while retaining the freedom to choose which features to implement and how. This opens the door for standardization, but also allows developers to begin working on the most critical features of their communication scheme first.

10.2 Further Development

The Infinity default feature set is the next priority for development. Systems for device naming, changing speeds mid-transmission, and error recovery will need to be implemented to take full advantage of Infinity's contextual nature. The Infinity Portal circuit board will also need to be cast in resin for a sleek and watertight finish. As development continues, a new plug and receptacle system for the Infinity Cable must be selected as well, as the current temporary system is unnecessarily bulky.

The developer program is also in need of additional resources and a system of developer registration. Said system will need to be programmed dynamically in PHP. In addition, a database must be constructed to store the information. This will come with a new set of graphics and interface requirements as well.

Furthermore, the Infinity Portal App is in need of automation and a graphical user interface. Although the console was adequate for testing and early development, automatic detection of Portals and devices connected to them will reduce the time it takes to develop new features significantly, while presenting end users with a responsive way to interact with the Infinity network.

Future versions of Infinity could also contain provisions for other features such as power transfer over the data lines, allowing devices that do not use large amounts of power to power and charge themselves from the receive line. Research will be performed into this technology to determine if it can be used to cut down on cable cost, weight, and length. It may even be possible to develop a superspeed version of the protocol that runs on differential hardware as an alternative to USB where faster speeds are required. Regardless, with the precedent set by the development and testing results previously discussed, Infinity has the potential to redefine the future of versatile, context-driven serial communication for small CPUs.

11. Bibliography

1. *Serial data*. (2020, May 31). Www.Digi.Com. https://www.digi.com/resources/documentation/Digidocs/90001541/reference/r_serial_data.htm?TocPath=Serial%20communication%7C_____2

2. Miettinen, A. (2016, October 17). *Future is bright for data transmission*. Www.Oulu.Fi. https://www.oulu.fi/university/node/42774

3. *ATTINY1617 - 8-bit Microcontrollers*. (2019). Www.Microchip.Com. https://www.microchip.com/wwwproducts/en/ATTINY1617

4. *ATmega328P - 8-bit AVR Microcontrollers*. (2019). Microchip.Com. https://www.microchip.com/wwwproducts/en/ATmega328p

5. Ibrahim, D. (2010). *Random Access Memory - an overview | ScienceDirect Topics*. Www.Sciencedirect.Com. https://www.sciencedirect.com/topics/engineering/random-access-memory

6. *Introducing binary - Revision 1 - GCSE Computer Science - BBC Bitesize*. (2019). BBC Bitesize. https://www.bbc.co.uk/bitesize/guides/zwsbwmn/revision/1

7. Cunningham, A. (2014, August 18). *How USB became the undefeated king of connectors*. Wired UK. https://www.wired.co.uk/article/usb-history

8. *Front Page | USB-IF*. (n.d.). Www.Usb.Org. Retrieved March 25, 2020, from https://www.usb.org/

9. Sattel, S. (2017, June 23). *What Is Differential Signaling?* | *EAGLE* | *Blog*. Eagle Blog. https://www.autodesk.com/products/eagle/blog/what-is-differential-signaling/

10. Miller, W. (2015, March 25). *USB Connectivity for MCUs: Which is Right for Your Next Design?* | *DigiKey*. Www.Digikey.Com. https://www.digikey.com/en/articles/techzone/2015/mar/usb-connectivity-for-mcus-which-is-right-for-your-next-design

11. Knowledgebase - Premium USB. (2020). Www.Premiumusb.Com. https://www.premiumusb.com/adapters

12. Johnson-Davies, avid. (2019, September 30). *Technoblogy - UPDI Programmer Stick*. Www.Technoblogy.Com. http://www.technoblogy.com/show?2OJT

13. Dhaker, P. (2018). *Introduction to SPI Interface* | *Analog Devices*. Www.Analog.Com. https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html#

14. I2C Info – I2C Bus, Interface and Protocol. (2019). I2C Info – I2C Bus, Interface and Protocol. https://i2c.info/

15. Controller Area Network (CAN) Overview - National Instruments. (2019, March 5). Www.Ni.Com. https://www.ni.com/en-us/innovations/white-papers/06/controller-area-network--can--overview.html

16. SPI vs I2C Protocols - Pros and Cons. (2019, February 4). Arrow.Com. https://www.arrow.com/en/research-and-events/articles/spi-vs-i2c-protocols-pros-and-cons

17. Boosten, M. (1998, March 11). *Transmission overhead and optimal packet size*. Hsi.Web.Cern.Ch. http://hsi.web.cern.ch/HSI/dshs/publications/wotug21/dsnic/html/node9.html

18. Pfile, R., Chien, S., & Koskie, S. (2017). *Synchronous vs. Asynchronous*. Et.Engr.Iupui.Edu. http://et.engr.iupui.edu//~skoskie/ECE362/lecture_notes/LNB25_html/text12.html

19. Austerlitz, H. (2003). *Asynchronous Method - an overview | ScienceDirect Topics*. Www.Sciencedirect.Com. https://www.sciencedirect.com/topics/engineering/asynchronous-method

20. Peacock, C. (2018, April 12). *USB in a NutShell - Chapter 5 - USB Descriptors*. Www.Beyondlogic.Org. https://www.beyondlogic.org/usbnutshell/usb5.shtml

21. Lattimer, B. (2019, January 7). *Overview of developing Windows client drivers for*

22. The RS-232 protocol. (2019, April 17). Https://Www.Omega.Com/En-Us/. https://www.omega.com/en-us/resources/rs232-serial-communication

22. Centronics. (1976). centronics :: 37400040F Model 306 Technical Manual Mar76. In *Internet Archive*.

23. USB devices - Windows drivers. Docs.Microsoft.Com. https://docs.microsoft.com/en-us/windows-hardware/drivers/usbcon/usb-driver-development-guide

https://archive.org/details/bitsavers_centronicschnicalManualMar76_13986426/mode/2up

24. Lundqvist, T. (2016, May 6). *The void left by the parallel port*. Medium. https://medium.com/@tltx/the-void-left-by-the-parallel-port-51eb6c919e8a

25. EIA Technical Standards. (2019). Www.Ecianow.Org. https://www.ecianow.org/eia-technical-standards

26. Bies, L. (2019, December). USB to RS232 converters. USB to RS232 Converters. https://www.lammertbies.nl/comm/info/rs-232-usb

27. Future Technologies. (2018). FT230X USB Bridge | UART. Www.Ftdichip.Com. https://www.ftdichip.com/Products/ICs/FT230X.html

28. Future Technologies. (2017). FTDI Chip Home Page. Www.Ftdichip.Com. https://www.ftdichip.com/

29. Arena. (2019, May 23). What is a Bill of Materials (BOM). Arena Solutions. https://www.arenasolutions.com/resources/articles/creating-bill-of-materials/

30. Printed Circuits. (2020). What is a Printed Circuit Board (PCB)? Printed Circuits LLC. https://www.printedcircuits.com/what-is-a-pcb/

31. Altium Limited. (2020). Professional PCB design tool | CircuitStudio. Www.Altium.Com. https://www.altium.com/circuitstudio/

32. Build EC. (2020, April 17). Electronic Schematics – What You Need To Know. Build Electronic Circuits. https://www.build-electronic-circuits.com/electronic-schematics/

33. Ucamco. (2020). Official Gerber Format Website. Ucamco. https://www.ucamco.com/en/gerber

34. JIALICHUANG Electronic Technology Development Co., Ltd. (2017). Get Started - JLCPCB: Help & Support. Support.Jlcpcb.Com. https://support.jlcpcb.com/

35. LCSC. (2017). Welcome to LCSC. Lcsc.Com. https://lcsc.com/about.html#/about/company

36. Cooks On Gold. (2019, April 1). What Is Solder Paste? And How Do You Use It? The Bench. https://www.cooksongold.com/blog/equipment-technique-focus/what-is-solder-paste-and-how-do-you-use-it

37. SMT Process. (2015). Reflow Soldering Process. SURFACE MOUNT PROCESS. http://www.surfacemountprocess.com/reflow-soldering-process.html

38. ByVision. (2013, March 14). The role of optical inspection in today's demanding electronics industry. Vision Engineering. https://www.visioneng.us/resources/articles/the-role-of-optical-inspection-in-today s-demanding-electronics-industry/

39. Hackaday. (2017, July 24). Review: TS100 Soldering Iron. Hackaday. https://hackaday.com/2017/07/24/review-ts100-soldering-iron/

40. Future Technologies. (2020). FTDI Utilities. Www.Ftdichip.Com. https://www.ftdichip.com/Support/Utilities.htm#FT_PROG

41. LCSC. (2019b). BOOMELE(Boom Precision Elec) | BOOMELE(Boom Precision Elec) C9742 | Pin Header & Female Header | LCSC. Lcsc.Com.

https://lcsc.com/product-detail/Pin-Header-Female-Header_Boom-Precision-Elec-2-54mm-1-40P-round-Headers-Pins_C9742.html

42. LCSC. (2019a). BOOMELE(Boom Precision Elec) | LCSC Electronics. Lcsc.Com. https://lcsc.com/brand-detail/86.html

43. Cornell Lassp. (1994, June 30). What's Hysteresis? Www.Lassp.Cornell.Edu. http://www.lassp.cornell.edu/sethna/hysteresis/WhatIsHysteresis.html

44. Hackaday. (2018, November 8). The Dual In-Line Package And How It Got That Way. Hackaday. https://hackaday.com/2018/11/08/the-dual-in-line-package-and-how-it-got-that-way/

45. WhatIS. (2011). What is UART (Universal Asynchronous Receiver/Transmitter)? WhatIs.Com. https://whatis.techtarget.com/definition/UART-Universal-Asynchronous-Receiver-Transmitter

46. Basic Electronics Tutorials (BET). (2013, August 30). Shift Register - Parallel and Serial Shift Register. Basic Electronics Tutorials. https://www.electronics-tutorials.ws/sequential/seq_5.html

47. Atmel. (2020c). AVR Memory - Developer Help. Microchipdeveloper.Com. https://microchipdeveloper.com/8avr:memory

48. Stone, C. (2020c). Hiovita Supporting Files Licence. https://hiovita.net/licenses/HSFL.pdf

49. Stone, C. (2020b, February 5). Hiovita Downloads. Hiovita.Net. https://hiovita.net/downloads/?file=portal-setup-1.0.exe

50. LedsMagazine. (2019). What is an LED? | LEDs Magazine. Ledsmagazine.Com. https://www.ledsmagazine.com/leds-ssl-design/materials/article/16701292/what-is-an-led

51. Atmel. (2020b). Atmel-ICE. Www.Microchip.Com. https://www.microchip.com/DevelopmentTools/ProductDetails/ATATMEL-ICE

52. Atmel. (2016c). AVR910: In-System Programming. http://ww1.microchip.com/downloads/en/appnotes/atmel-0943-in-system-programming_applicationnote_avr910.pdf

53. Atmel. (2020a). Atmel Studio IDE - Developer Help. Microchipdeveloper.Com. https://microchipdeveloper.com/atstudio:start

54. TLDP. (2003, April 11). Static Libraries. Www.Tldp.Org. https://www.tldp.org/HOWTO/Program-Library-HOWTO/static-libraries.html

55. Microsoft. (2019b, April 11). Build desktop apps for Windows PCs. Docs.Microsoft.Com. https://docs.microsoft.com/en-us/windows/apps/desktop/

56. Atmel. (2016b). AVR Microcontrollers AVR Instruction Set Manual OTHER Instruction Set Nomenclature. http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf

57. Microsoft. (2019c, April 25). Walkthrough: Creating and Using a Static Library (C++). Microsoft.Com.

https://docs.microsoft.com/en-us/cpp/build/walkthrough-creating-and-using-a-static-library-cpp?view=vs-2019

58. Microsoft. (2019a). Visual Studio 2019 | Download for free. Visual Studio. https://visualstudio.microsoft.com/vs/

59. TutorialsPoint. (2019). C Language Overview. Www.Tutorialspoint.Com. https://www.tutorialspoint.com/cprogramming/c_overview.htm

60. CPP Network. (2020). cplusplus.com - The C++ Resources Network. Www.Cplusplus.Com. https://www.cplusplus.com/

61. Ultralight LLC. (2020). Ultralight - Pure-GPU HTML UI Engine for Desktop and Games. Ultralig.ht. https://ultralig.ht/

62. W3Schools. (2018). HTML Tutorial. W3schools.Com. https://www.w3schools.com/html/

63. W3Schools. (2019a). CSS Tutorial. W3schools.Com. https://www.w3schools.com/css/

64. W3Schools. (2019b). JavaScript Tutorial. W3schools.Com. https://www.w3schools.com/Js/

65. Microsoft. (2016, April 14). Visual Studio Code. Visualstudio.Com. https://code.visualstudio.com/

66. GIMP Foundation. (2019). GIMP. GIMP. https://www.gimp.org/

67. Inkscape Website Developers. (2019). Draw Freely | Inkscape. Inkscape.Org. https://inkscape.org/

68. Weinreb, B. (2013, February 12). What Are Raster Graphics? Definition, Terms, and File Extensions. Learn.G2.Com. https://learn.g2.com/raster-graphics

69. Adobe. (2020). What is vector art - vector art for beginners | Adobe. Www.Adobe.Com. https://www.adobe.com/creativecloud/illustration/discover/vector-art.html

70. Stone, C. (2020d, February 5). Shop Online. Hiovita.Net. https://hiovita.net/store

71. Stone, C. (2020a, February 5). Downloads. Hiovita.Net. https://hiovita.net/downloads/?file=portal-setup-1.0.exe

72. Atmel. (2015a). ATmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash DATASHEET Features. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

73. Atmel. (2015b). ATmega328P Datasheet (p. 274). http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

74. Atmel. (2013, August 25). Quartz Crystal Oscillator and Quartz Crystals. Basic Electronics Tutorials. https://www.electronics-tutorials.ws/oscillator/crystal.html

75. Atmel. (2016a). AVR053: Internal RC Oscillator Calibration for tinyAVR and megaAVR Devices. http://ww1.microchip.com/downloads/en/appnotes/atmel-2555-internal-rc-oscillator-calibration-for-tinyavr-and-megaavr-devices_applicationnote_avr053.pdf

