Wireless LED Display Board

Connor Monahan  
Washington University in St. Louis  
January 10, 2021

# Introduction and Goals

Visual displays come in a variety of form factors, from single indicator lights to the largest jumbotron video displays. Any of these can be used for a variety of applications. Messages or images can be displayed given a high enough pixel density. While it is possible to build a display using an off-the-shelf LCD display module or a prebuilt array, I endeavored to make one with an array of multicolor LEDs, for a low-resolution reminiscent of old games. The device would then be useful as a decoration or as a component of a costume.

For this project, I desired to build a system that would control the output of an array of LEDs based on a wireless control mechanism. It should be able to display at least 5 different visuals/messages selectable wirelessly by buttons with no more than 50ms of delay. Some of the related goals were to have a form factor of around 19x19cm, like that of a tablet. I additionally wanted to be able to run it on a rechargeable battery for at least an hour so it could be used in mobile applications. My resulting LED display board achieves these goals with wide margins, having the capacity for hundreds of visuals and up to 10 hours of run time.

Table of Contents

[Introduction and Goals 1](#_Toc60955257)

[Specifications 2](#_Toc60955258)

[Wireless transmission specification 2](#_Toc60955259)

[Display specification 3](#_Toc60955260)

[System specification 3](#_Toc60955261)

[Hardware Design 4](#_Toc60955262)

[Software Design 10](#_Toc60955263)

[Wireless interface 10](#_Toc60955264)

[LED interface 11](#_Toc60955265)

[Overall state diagram 12](#_Toc60955266)

[System integration 14](#_Toc60955267)

[Testing and Results 14](#_Toc60955268)

[Conclusion 15](#_Toc60955269)

# Specifications

As outlined in my project proposal, the specifications created before any integration testing included the following:

* The remote shall have two primary buttons: up and down
* On a button press, the remote shall transmit a wireless message
* The wireless message shall contain a 1-byte address representing the intended receiver and a 1-byte identifier representing the button pressed
* The receiver shall check for incoming messages with no more than 50ms of delay
* Upon receiving a valid message, the display controller shall cycle forward or backward through visuals
* The controller shall periodically update display based on current state

## Wireless transmission specification

* Transmitter and receiver will communicate on a 1MHz channel centered at 2.476GHz at a bit rate of 1Mbps with normal power output.
* Data link format will use the “Enhanced ShockBurst” protocol as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1-byte Preamble | 5-byte Address | 9-bit Control | 32-byte Payload | 2-byte CRC16 |

where Address is {0x30, 0x30, 0x30, 0x30, 0x31} and other details are defined in the protocol specification[[1]](#footnote-1).

* Application data format (Payload) is defined as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1-byte address | 6 bits undefined | 1 bit: down button status | 1 bit: up button status | 30 bytes undefined |

where button status means 1 when button is pressed, 0 otherwise. Address shall be unique for each pair of remote and display (0x01 used in the demo).

* Remote shall constantly monitor button status while idle and transmit immediately as soon as either button is pressed.
* There shall be a debounce/cooldown time of 100ms after each button press during which further messages shall not be sent.
* Display shall monitor wireless interface for messages at least once per 50ms and perform the appropriate action if a valid (address matches) message is received where only one button is indicated as being pressed.

## Display specification

* The display microcontroller shall control the output of the LEDs using the one-wire WS2812B-V5 protocol[[2]](#footnote-2), defined as

|  |  |  |  |
| --- | --- | --- | --- |
| Blanking time | 24 bits GRB color code for LED 1 | … | 24 bits GRB color code for LED n |

where the blanking time is a logic 0 pulse for at least 280µs. A bit 0 is transmitted with a logic 1 pulse of 220ns-380ns followed by a logic 0 pulse of 580ns-1µs. A bit 1 is transmitted with a logic 1 pulse of 580ns-1µs followed by a logic 0 pulse of 580ns-1µs. Bits may have idle logic 0 output for <280µs between them. The 24 bits color code is repeated for all LEDs connected in sequence, transmitted MSB first.

* The output of all LEDs shall be updated when the state describing the currently selected visual/image changes according to the receipt of a message by the wireless interface

## System specification

The components of the subsystem, on the display and the remote, communicate using an interface similar to the following:

LEDs

Display(

byte data[])

Transponder

Set-Config()

Byte[] Read(cmd, length)

Write(cmd, data[], length)

Controller

Int cur\_visual

Check-Message()

Update-State()

Set-Output(vis)

Remote

Read-Buttons()

Send-Message()

# Hardware Design

For component selection, I opted to prefer parts I had experience using to decrease the probability of required rework, but that also met the requirements. I also opted for parts that I had at home, or were available locally, or were available at a preferred supplier. The final list of components used in the design are:

* Microcontroller: ATSAMC21J18A-AUT
* Transponder: NRF24L01+ on a package with integrated antenna and other passives
* Display battery: PKCELL 3.7V Lithium-Polymer battery at 2500mAh / 10Wh
* Smart LED IC: WS2812B-V5
* LDO 3.3V Regulator: TC1186-3.3V
* Power MOSFET: AO3401A
* Resistors/capacitors: miscellaneous Hong Kong brands in 0603 package
* Remote: Arduino

The chosen LED ICs require 3.7V-5.3V at VDD to operate correctly, with logic 1 input between 2.7V-VCC+0.7V. The chosen microcontroller requires 2.7V-5.5V at VCC to operate correctly. The chosen wireless transceiver requires 1.9-3.6V at VDD to operate correctly but accepts logic 1 up to 5.25V if VDD is limited to 3.3V. To satisfy the variety of voltage requirements in this system, I chose to use a primary voltage supply between 3.7V-4.1V to power the microcontroller and the LEDs, regulated down to a secondary supply of 3.3V to power the wireless transceiver. This was chosen as the power draw at 3.7V will be significantly higher (>220mA due to phantom draw from LEDs) than that on 3.3V (<15mA even during communication), such that the 3.3V power could be generated with an ultra-low dropout linear regulator and the 3.7V+ supply could come directly from a lightweight high-capacity rechargeable Lithium-Polymer battery.

The chosen LED IC was no accident: it is commonly sold at local hobby shops in the form factor of an LED strip with dozens/hundreds in sequence, as their custom protocol works like a shift register. The wireless transceiver is also commonly sold at hobby shops on an 8-pin form factor breakout board with on-board antenna and all required RF passives pre-assembled. Same thing with the chosen microcontroller, as the related SAMD21 series can be found on breakout boards at hobby shops. This allowed all preliminary unit testing to occur with off-the-shelf components on breadboard.

Remote

Display Controller

CPU

Buttons

Transponder

CPU

Transponder

Battery

LEDs

Fig 1. Block diagram of primary hardware components

After separately validating both my ability to control a single LED strip with this CPU as well as my ability to talk wirelessly between my remote prototype (Arduino with NRF shield) and this CPU with a NRF module, I began to work on the design for the combined package. The first design to consider was the power supplies for all devices.



In this diagram, power from the battery around 3.7V nominal is supplied at P1. It then connects to a small optional reverse polarity protection circuit I designed using power P-FETS to support up to 12A, more than enough to drive the board at full power. 3.3V is then generated from this supply using the ultra-LDO regulator with appropriate load capacitors. This can support up to 150mA while needing less than a 0.4V drop, which is more than enough to power the transceiver.



The 3.7V supply powers the CPU on both its analog and digital domains, with appropriate bypass capacitors. To program the CPU, a 10-pin JTAG header was necessary, which was compatible with Atmel-ICE JTAG programmers I had lying around. The 6 independent serial communication interfaces provided by the microcontroller were connected to 5 separate LED strips through a resistor to provide power isolation and to the wireless transceiver through one SPI interface with other necessary control signals.



Each individual LED IC is connected to 3.7V power and ground. The LED array is divided up into 5 independent circuits, to isolate failures. Each of these has 40 ICs in sequence except for the last one which has 60 ICs in sequence, which is far less than the maximum that can be supported of 1024 at 30Hz refresh rate.

While the original goal and specification of the project was to build up these circuits on breadboard to demonstrate the functionality, I decided to order a printed circuit board assembly based on these designs in the off chance it worked as my design would be useful for future personal projects. Due to its success as demonstrated in the results, I will submit and describe the layout as well.



This layout is a 4-layer PCB with internal 3.7V and GND planes designed according to the capabilities of a Shenzhen-based circuit board factory. My design placed the 220 LEDs in a rectangular pattern as close as possible across the surface of the board spaced uniformly in the horizontal and vertical directions. The CPU and other components were placed along the periphery with connectors on the other side. Traces were sized appropriately for ampacity and minimum requirements. Two boards were ordered with full SMT assembly service, as this was somehow cheaper than even ordering the LEDs from a U.S. supplier and otherwise made this possible.

A close - up of a circuit board

Description automatically generated with medium confidence

Figure 5. Final PCBA (a bit smaller than actual size of ~7.5” wide)

In addition to this hardware assembly for the display, I also assembled a remote for testing the wireless interface. The remote was constructed on breadboard using an Arduino as the microcontroller, with a wireless interface connected over SPI in the same fashion as shown above, with an additional two SPST buttons connected to general purpose I/O pins on the CPU with appropriate pull-up resistors. The hardware assembly for the remote can be seen in the final testing video provided later.

# Software Design

Since the targeted CPU for this project is a 32-bit ARM processor, we have a full compiler toolchain available. Therefore, the software design for this project will be primarily in C with some inline assembly for critical sections.

The main interfaces for the display microcontroller are the wireless transceiver which communicates over SPI, and the LED strips which I originally hoped could interact over SPI but had to resort to a custom solution due to its strict timing requirements. The SAMC21 series of microcontrollers provides a generalized hardware serial communication interface, known as SERCOM, which can be used to control SPI devices and other serial devices, as well as a hardware general purpose I/O interface, known as PORT, which are both accessible from C code as registers.

## Wireless interface

To enable the wireless interface, I first had to set the pin configuration for the 6 SPI and control signals in the PORT interface.

*// set up MOSI/MISO/SCK*

PORT\_REGS->GROUP[1].PORT\_PMUX[16 / 2] =

PORT\_PMUX\_PMUXE(MUX\_PB16C\_SERCOM5\_PAD0) | *// MOSI*

PORT\_PMUX\_PMUXO(MUX\_PB17C\_SERCOM5\_PAD1); *// MISO*

PORT\_REGS->GROUP[0].PORT\_PMUX[21 / 2] = PORT\_PMUX\_PMUXO(MUX\_PA21C\_SERCOM5\_PAD3); *// SCK*

PORT\_REGS->GROUP[1].PORT\_PINCFG[16] = PORT\_PINCFG\_PMUXEN(1);

PORT\_REGS->GROUP[1].PORT\_PINCFG[17] = PORT\_PINCFG\_PMUXEN(1);

PORT\_REGS->GROUP[0].PORT\_PINCFG[21] = PORT\_PINCFG\_PMUXEN(1);

*// set up CE/CSN*

PORT\_REGS->GROUP[NRF\_CE\_CSN\_GROUP].PORT\_DIRSET = NRF\_CE\_PORT | NRF\_CSN\_PORT;

*// pull CSN high to disable SPI*

PORT\_REGS->GROUP[NRF\_CE\_CSN\_GROUP].PORT\_OUTSET = NRF\_CSN\_PORT;

*// pull CE low to disable transmitter/receiver*

PORT\_REGS->GROUP[NRF\_CE\_CSN\_GROUP].PORT\_OUTCLR = NRF\_CE\_PORT;

To set up the SPI interface, I first had to enable a clock signal to go to the peripheral, then enable CPU interrupts for the peripheral, then reset it, change the config, and re-enable it.

*// clocks*

MCLK\_REGS->MCLK\_APBCMASK |= MCLK\_APBCMASK\_SERCOM5(1);

GCLK\_REGS->GCLK\_PCHCTRL[25] = GCLK\_PCHCTRL\_CHEN(1) | GCLK\_PCHCTRL\_GEN\_GCLK0;

NVIC\_EnableIRQ(SERCOM5\_IRQn);

*// reset it*

NRF\_SERCOM\_REGS->SPIM.SERCOM\_CTRLA = SERCOM\_SPIM\_CTRLA\_ENABLE(0);

**while** (NRF\_SERCOM\_REGS->SPIM.SERCOM\_SYNCBUSY){}

*// set configuration*

NRF\_SERCOM\_REGS->SPIM.SERCOM\_CTRLA = NRF\_CTRLA;

NRF\_SERCOM\_REGS->SPIM.SERCOM\_CTRLB = NRF\_CTRLB;

NRF\_SERCOM\_REGS->SPIM.SERCOM\_BAUD = NRF\_BAUD;

*// enable*

NRF\_SERCOM\_REGS->SPIM.SERCOM\_CTRLA |= SERCOM\_SPIM\_CTRLA\_ENABLE(1);

**while** (NRF\_SERCOM\_REGS->SPIM.SERCOM\_SYNCBUSY){}

I could then send and receive individual bytes using busy-wait I/O.

*// Wait until we can write*

**while** (!(NRF\_SERCOM\_REGS->SPIM.SERCOM\_INTFLAG & SERCOM\_SPIM\_INTFLAG\_DRE\_Msk)){}

*// Write*

NRF\_SERCOM\_REGS->SPIM.SERCOM\_DATA = out;

*// Wait until done receiving*

**while** (!(NRF\_SERCOM\_REGS->SPIM.SERCOM\_INTFLAG & SERCOM\_SPIM\_INTFLAG\_RXC\_Msk)){}

*// Read*

**return** NRF\_SERCOM\_REGS->SPIM.SERCOM\_DATA;

The wireless transceiver chip provides a set of commands described in its datasheet that can be sent over SPI to control the device, in either read or write mode depending on the command. First, the chip select pin would be set low. Then, the command number and optional argument would be sent as one byte. Then, a known number of bytes would be written or read from the SPI depending on the command. Finally, the chip select pin would be raised high and kept high for a minimum duration of time.

PORT\_REGS->GROUP[NRF\_CE\_CSN\_GROUP].PORT\_OUTCLR = NRF\_CSN\_PORT;

uint8\_t status = spi\_xfer(command);

**for** (**int** i = 0; i < length; ++i)

{

output[i] = spi\_xfer(input[i]);

}

PORT\_REGS->GROUP[NRF\_CE\_CSN\_GROUP].PORT\_OUTSET = NRF\_CSN\_PORT;

*// CSN must go high for a certain minimum duration between commands*

**asm**("nop\r\nnop\r\nnop\r\nnop\r\nnop\r\n");

With a couple of defined functions in my [drv\_nrf24l01.h](https://github.com/cmastudios/LEDDisplayBoard/blob/master/Software/RinaBitBang.X/drv_nrf24l01.h) and [drv\_nrf24l01.c](https://github.com/cmastudios/LEDDisplayBoard/blob/master/Software/RinaBitBang.X/drv_nrf24l01.c), the device could be initialized as follows.

spi\_res = nrf\_read\_op(NRF\_CMD\_NOP, 0, **NULL**);

spi\_res = nrf\_write\_reg(NRF\_REG\_RF\_CH, 76); *// 2.476GHz*

spi\_res = nrf\_write\_reg(NRF\_REG\_SETUP\_RETR, (0b0101<<4)|(0b1111)); *// delay 1500uS 15 tries*

spi\_res = nrf\_write\_reg(NRF\_REG\_RF\_SETUP, 0); *// 1Mbps at 0dBm*

*// reset current status*

spi\_res = nrf\_write\_reg(NRF\_REG\_STATUS,

NRF\_REG\_STATUS\_RX\_DR | NRF\_REG\_STATUS\_TX\_DS | NRF\_REG\_STATUS\_MAX\_RT);

*// flush buffers*

spi\_res = nrf\_read\_op(NRF\_CMD\_FLUSH\_TX, 0, **NULL**);

spi\_res = nrf\_read\_op(NRF\_CMD\_FLUSH\_RX, 0, **NULL**);

*// turn the thing on, prime it for receiver mode*

spi\_res = nrf\_write\_reg(NRF\_REG\_CONFIG, NRF\_REG\_CONFIG\_EN\_CRC | NRF\_REG\_CONFIG\_CRCO\_CRC16 |

NRF\_REG\_CONFIG\_PRIM\_RX | NRF\_REG\_CONFIG\_PWR\_UP);

We can now read from the device periodically.

**static** uint8\_t rx\_data[32] = {0};

spi\_res = nrf\_read\_op(NRF\_CMD\_NOP, 0, **NULL**);

**if** (spi\_res & NRF\_REG\_STATUS\_RX\_DR)

{

uint8\_t width;

spi\_res = nrf\_read\_op(NRF\_CMD\_R\_RX\_PL\_WID, 1, &width);

width = (width > 32) ? 32 : width;

spi\_res = nrf\_read\_op(NRF\_CMD\_R\_RX\_PAYLOAD, width, rx\_data);

}

The remote can transmit messages using a similar interface.

## LED interface

As detailed in the specification above, the LED control interface uses a custom protocol with strict timing requirements. To meet the requirements, bit-banging is used dependent on the 16MHz clock of the CPU, known single-cycle instruction timing, and the SysTick timer.

To initialize the output signals for the LED strips, we simply use the PORT module similarly to that done for the wireless interface, setting each pin as an output.

PORT\_REGS->GROUP[LED\_1\_GROUP].PORT\_DIRSET = LED\_1\_PORT;

As described in the specification, to start displaying a visual on the LEDs we must first drive the signal low for the blanking period. This is accomplished using the GPIO interface, the known clock speed of the CPU, and the SysTick timer built into the CPU.

SysTick->CTRL = SysTick\_CTRL\_CLKSOURCE\_Msk | SysTick\_CTRL\_ENABLE\_Msk;

**static** **inline** **void** delay\_cycles(**const** uint32\_t n){

SysTick->LOAD = n;

SysTick->VAL = 0;

**while** (!(SysTick->CTRL & SysTick\_CTRL\_COUNTFLAG\_Msk)){}

}

#define CLOCK 16000000

#define delay\_us(x) delay\_cycles(CLOCK/1000000\*(x))

PORT\_REGS->GROUP[LED\_1\_GROUP].PORT\_OUTCLR = LED\_1\_PORT;

delay\_us(350);

ws\_transmit(LED\_0, image->eyes, 40 \* 3);

Since the CPU we’re using has hundreds of kilobytes of flash space, I calculated that hundreds of visuals can be stored uncompressed in program memory. Therefore, each is stored as a GRB byte array in the program and accessed through the normal memory interface.

**struct** image {

uint8\_t upper[20 \* 8 \* 3];

uint8\_t lower[10 \* 6 \* 3];

};

**extern** **const** **struct** image image\_face, image\_cse467, image\_washu, image\_error\_battery, …;

To display the visual, we simply need to send every byte over the WS2812B interface, and thus every bit. As discussed in the specifications, a bit 0 is sent with a high pulse given specific timing and a low pulse given specific timing. Given the CPU is running at 16MHz single cycle, we can observe that a high time of 4 instructions and a low time of 15 instructions meets the requirements and can be achieved by adding extra nop instructions.

PORT\_REGS->GROUP[LED\_1\_GROUP].PORT\_OUTSET = LED\_1\_PORT;

**asm**("nop\r\nnop\r\nnop\r\n");

PORT\_REGS->GROUP[LED\_1\_GROUP].PORT\_OUTCLR = LED\_1\_PORT;

**asm**("nop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\n");

A bit 1 can be sent similarly with a high and low time of 10 instructions.

PORT\_REGS->GROUP[LED\_1\_GROUP].PORT\_OUTSET = LED\_1\_PORT;

**asm**("nop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\n");

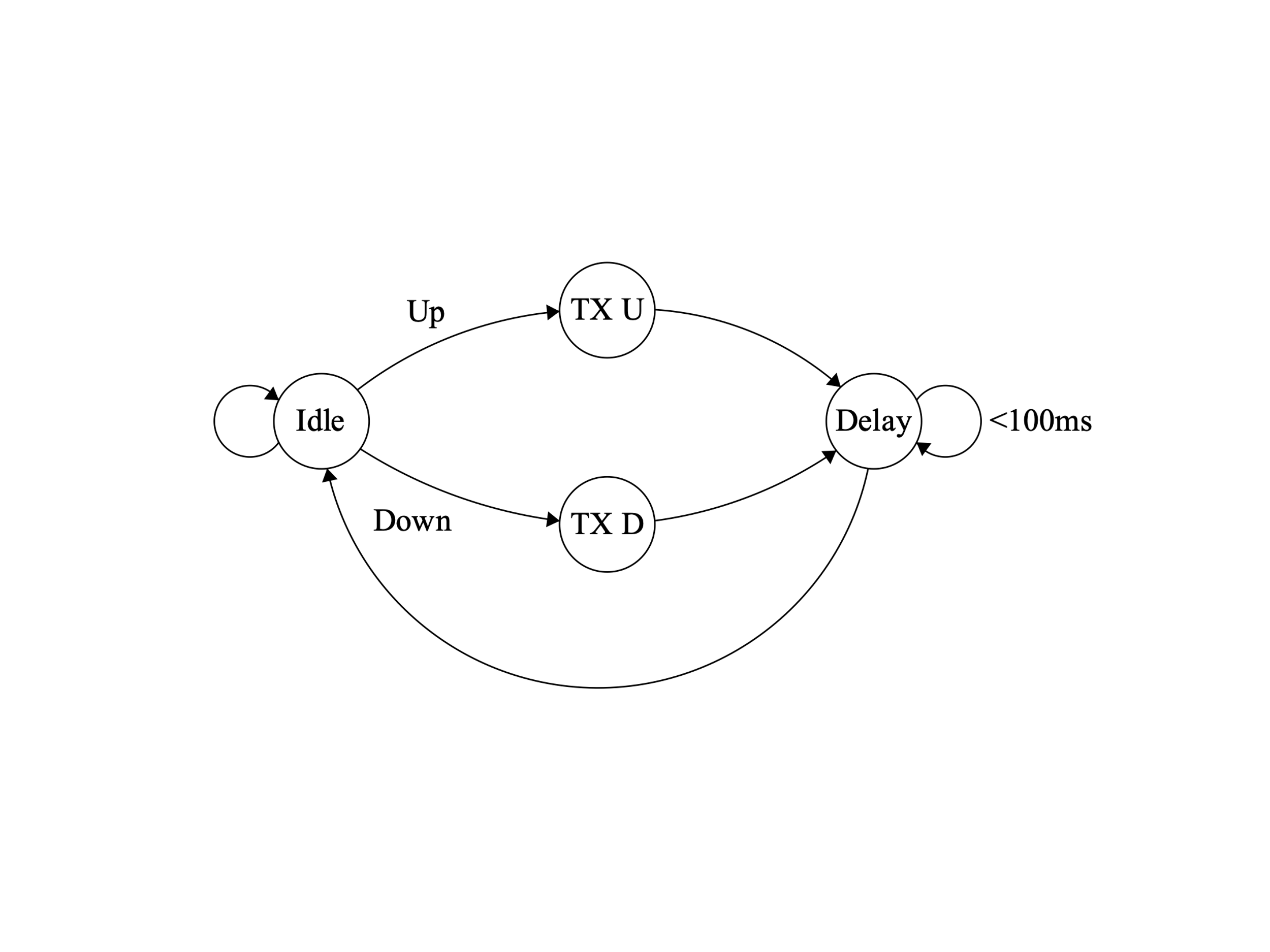
PORT\_REGS->GROUP[LED\_1\_GROUP].PORT\_OUTCLR = LED\_1\_PORT;

**asm**("nop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\nnop\r\n");

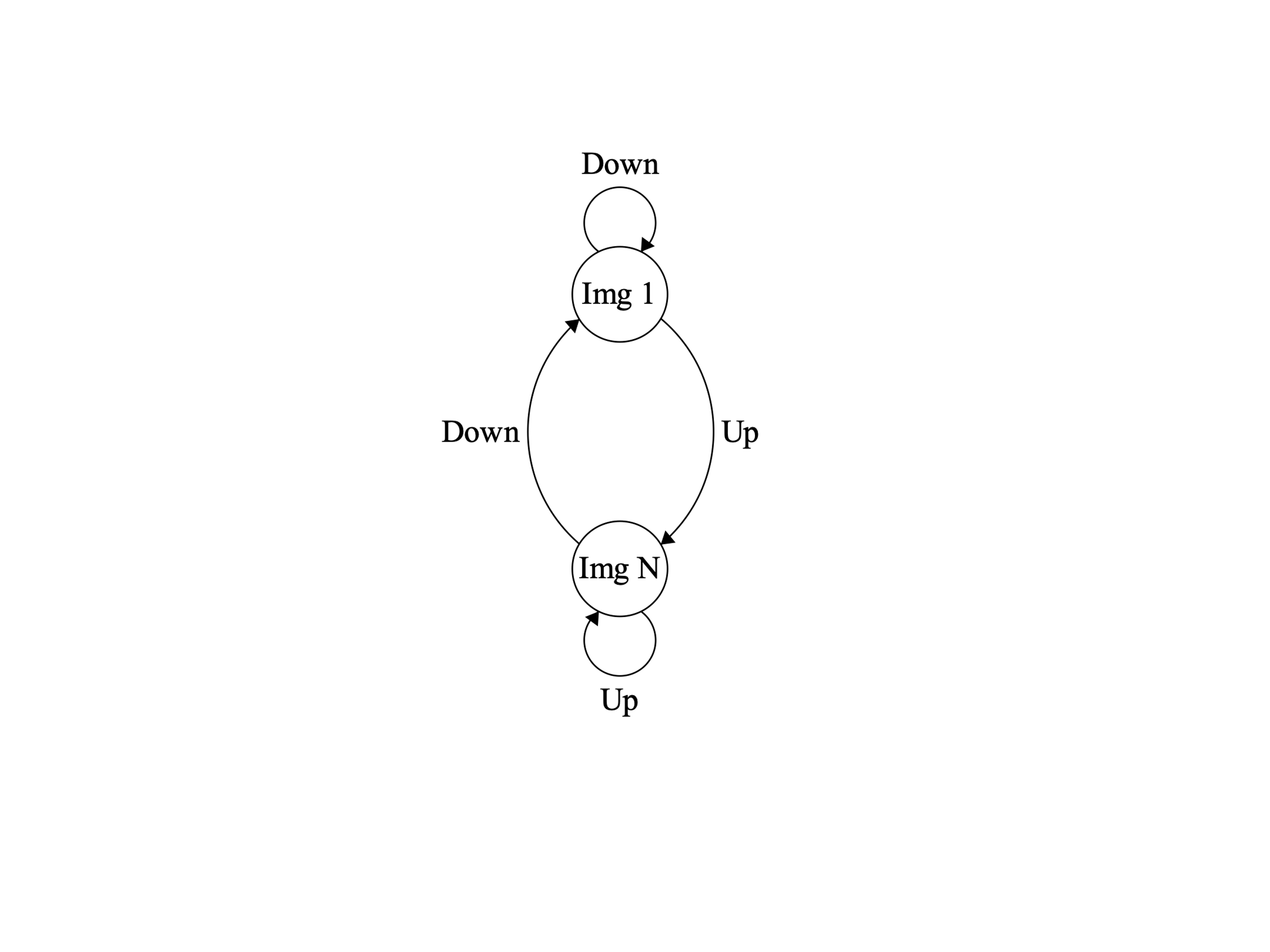
This process is then repeated for every bit and byte in the visual.

## Overall state diagram

Remote: transitions to respective transmission state on a button press, then waits 100ms before checking for further interactions



Display: transitions between displayed visuals when an Up or Down message is received. Doesn’t transition up after reaching the last image and doesn’t transition down after reaching the first image. LEDs are updated upon transition to a new state. Demo has N=5 images.



…

Full source code for the software in this project is available at <https://github.com/cmastudios/LEDDisplayBoard/tree/master/Software/RinaBitBang.X>

Software was compiled with the latest version of the GCC compiler toolchain for ARM using MPLAB X IDE and linked based on memory definition files for this microcontroller.

# System integration

At this stage, the separate components of the system included software for the display, software for the remote, hardware assembly for the display, hardware assembly for the remote, and a Lithium-polymer battery. To install the software on the microcontroller for the display, an Atmel-ICE JTAG programmer was connected between the microcontroller and a workstation computer running MPLAB X IDE, which was then used to download the compiled application. To install the software on the remote, the Arduino IDE was used.

# Testing and Results

Before integration, I performed a series of electrical tests on the display PCBA itself. I tested the resistance between the various power circuits and ground to determine if there were any obvious short circuits, finding none. I then performed continuity tests between all of the LED data input pins and the microcontroller output pins, finding they were all connected and that there were no shorts. I tested a few LEDs to verify power and ground were connected, which was the case. I then installed the through-hole connector components and then performed visual inspection and connectivity tests for shorts.

I first powered up the display board with no software installed, to spot any remaining electrical issues. With all LEDs off, the wireless interface disabled, and the CPU in sleep mode, the board drew 127mA of current at 4.1V. This is similar to the expected idle current draw of the LED ICs of 0.6mA per LED or 132mA for all. Unfortunately, I did not realize that such a large idle power draw should be expected as I must have missed that detail in the datasheet, which is disappointing as that left no way to put the board in a low power state without removing the battery. After software is installed, the first visual was displayed, which increased the power draw to 224mA at 5V. This meets the requirement of at least an hour of runtime, as this corresponds to around 10 hours of operation with our 2500mA battery.

Integration testing for this project primarily involved supplying appropriate power to the display and the remote, interacting with the up and down buttons on the remote for an appropriate period of time, and then observing the output on the LEDs on the display. The results of this testing are visible in my Demonstration Video at <https://youtu.be/-yaSNob3mKQ> and will be summarized below. Briefly, the display board powered up and displayed the first visual, a happy face, after the battery was connected. The remote was then powered up by connecting to the workstation. Pressing the down/previous button had no effect as we were at the first image already. Pressing the up/next button caused the display to advance to the next image and display it immediately on the LEDs, as expected. This continued to work until reaching the last image, a warning sign, where pressing next ceased to have an effect, as expected. Pressing the previous button caused the display to redisplay the last image, repeating this until reaching the first image again.

# Conclusion

The successful results of this project definitely included a lot of luck that everything worked out especially with the first try of the PCBA order. However, the development of the project was not as straightforward as the format above allowed me to present. In hardware design, I don’t touch on the three times I significantly modified the board design to reduce risk such as swapping out the CPU for an exact chip I previously used, changing the power networks to use the same voltage at the CPU and the LEDs, and adding extra pads in case I needed to add a resistor by slicing a trace. In software design, I briefly noted how I had to give up on using the hardware SPI to implement the custom WS2812B protocol as it didn’t meet timing requirements when used with more than 5 LEDs. On the other hand, I was surprised how well my NRF wireless protocol implementation worked without major effort, only major issues in testing being some C code typos and incorrect pin numbers. One major issue discovered in testing was related to component failures. I ordered assembly for two PCBs from the supplier, and the first PCBA I was using for testing was eventually found to have at least 6 failed LED ICs. The second PCBA used had no such failures, and so I had thus spent many unnecessary hours debugging the first board. I believe that the possible failure modes for these LED ICs may include ESD/physical damage in shipment or overtemperature damage I may have caused by soldering on nearby components.

While this project met the requirements and goals outlined at the beginning, given more time I would have liked to create more to make this a finished product. This future work would include making a more permanent design for the remote, such as by ordering a PCBA with a small CPU, a standard primary battery, the wireless interface, and up/down buttons. This would further necessitate some active power management in software to keep it running on the same battery for a long period of time. Given a further increased scope, plastic enclosure designs could also be created for the remote similar to a small TV remote and for the display similar to a portable tablet, with diffuse plastic in front of the LEDs to produce a continuous image.

All in all, this project allowed me to gain further experience in hardware and software design while working on something related to my interests. Thanks for this exciting opportunity!

1. nRF24L01+ Preliminary Product Specification (<https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf> ) [↑](#footnote-ref-1)
2. WS2812B-V5 Intelligent control LED integrated light source (<http://www.peace-corp.co.jp/data/WS2812B-V5_V1.0_EN.pdf> ) [↑](#footnote-ref-2)