

# ESE 323 Project Presentation

---

By Chris Nielsen



# Defining the project

In many competitive and showcase scenarios the legitimacy of player control/inputs as well as other input based performance is brought into question. My original goal was to create a bridge between the game layer and a discrete/serial based reporting system. This would require a microcontroller to decode and decipher the signals sent by old school controllers, and then both display in real time and report to a computer what is going on.



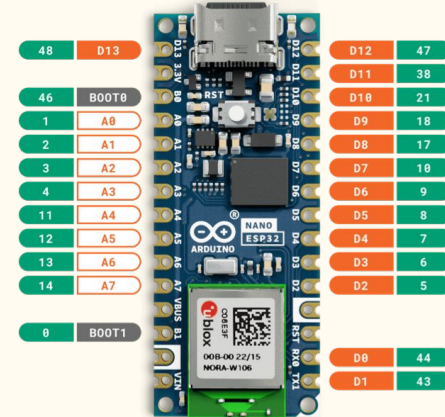
# MicroController Selected

I chose the ESP32 as recommendation from Professor Westerfeld.

Arduino's easy to use system programmer as well as interface made debugging and development easy.

**Nano / ESP32**

Pinout

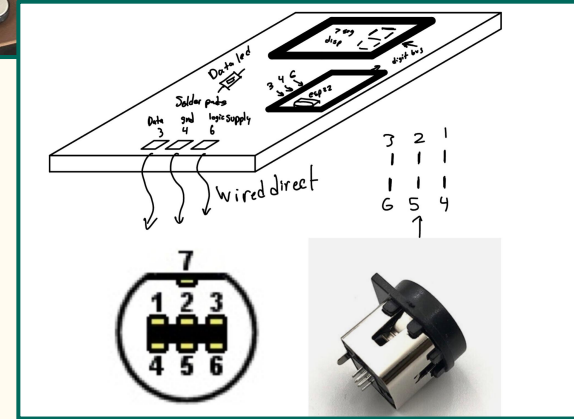
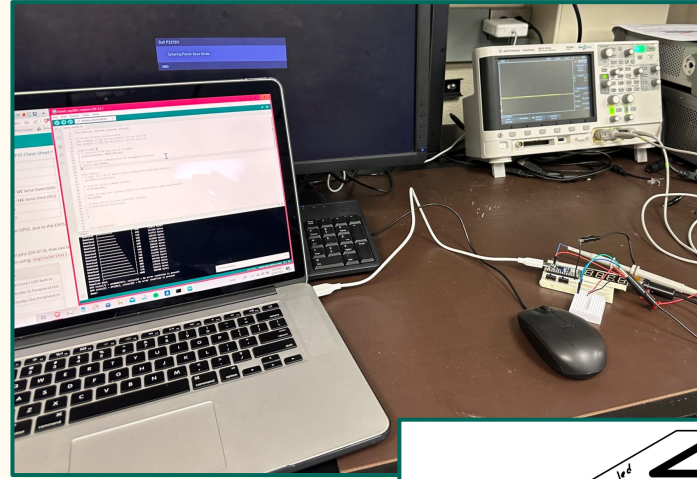


**i** ESP32 pin numbers

**i** Nano pin numbers

# Development (Hell?)

Development and research brought to my attention that the emulation and control of controllers made by game companies was not a reasonable task, and my device would have to be parasitic in its operations, only reading inputs when sensed across the bus.



# Basic Program Flow

For the arduino, i have it setup for a maximum of 4 different controller types can be sensed and used. Here, i have hardcoded handling the gamecube outputted controller data packet structure. It works by looking first for the command sent by the game console and then delaying until it sees the 64 bit controller state response. Then once all bits are received it parses them and can be use to communicate everything over serial, and also output which button has been pressed to the 7-Segment display using the multi usage ports on the arduino.

With an official controller attached, there is an interval of about 6.7ms between successive updates. Each update looks to last a little over 450us. The sequence starts with a 24-bit command from the console:

```
0100 0000 0000 0011 0000 0010
```

The last two bits of this sequence appear to be the 'rumble' control. The pattern is normally '10', but seems to change to '01' when the motor is running. After a rumble the pattern sometimes remains '00' for a while, but eventually reverts to '10'. It's not yet clear whether these are a simple enable and drive signal, or if some initialisation sequence is required as with the N64 controller.

This sequence is probably a command word from the console. Although I've not seen the other bits change in this word, I'm sure they have some function codes (for example, there may be commands to reset the controller, or query what kind of hardware is attached to the console).

After the 24-bit command word, there is a short delay (about 15us, the data line being high) before the controller responds. After this period, the controller responds with a string of bits that contain the state of all the buttons along with joystick position data. From examination with an oscilloscope, the sequence appears to be as follows:

```
000, Start, Y, X, B, A, 1, L, R, Z,  
DPadUp, DPadDown, DPadRight, DPadLeft,  
JoyX (8-bit signed), JoyY (8-bit signed),  
CX (8-bit signed), CY (8-bit signed),  
Left (8-bit), Right (8-bit)
```

Finally, the data line returns high again. Above, the L/R buttons are the end-stops on the L/R shoulder buttons. Note that between A and L there is a bit that always appears to be high. Also, I haven't seen the three leading zeros change. The buttons (and extra bits) make up a 16-bit field, the analogue controls make up a further 48-bits. Therefore, the GC sends a 24-bit command to the controller, and the control responds with a total of 64-bits. On the oscilloscope, the data bits look to be 5us, which gives a total of  $(64+24+3)*5us = 455us$ . This is assuming that the 'scope time-base is accurate.

Here is an example of what a complete data string looks like on the 'scope. In this figure, point A is the start of the 24-bit command word sent by the console, and point B marks the start of the 64-bit response from the controller. The quality of the image is quite poor, but it's actually possible to see the individual data bits.



The bit-pattern looks the same regardless of which controller port is used, so I guess that the control word doesn't contain a port identifier. If the console is reset, the first rumble bit seems to go low initially and then returns high after a few seconds. The same thing happens when you power on the console initially. In general, the communication with controllers seems to happen almost immediately when the console is turned on, so it seems likely that there is some dedicated interface here, rather than something that needs to be set up programmatically by the CPU.

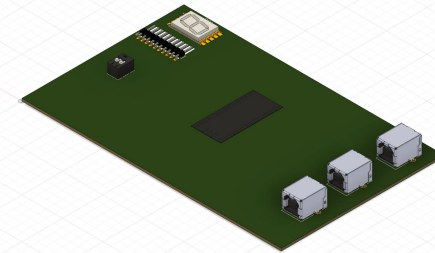
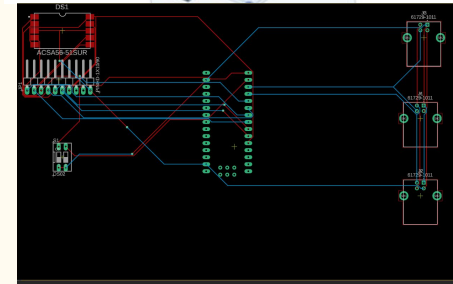
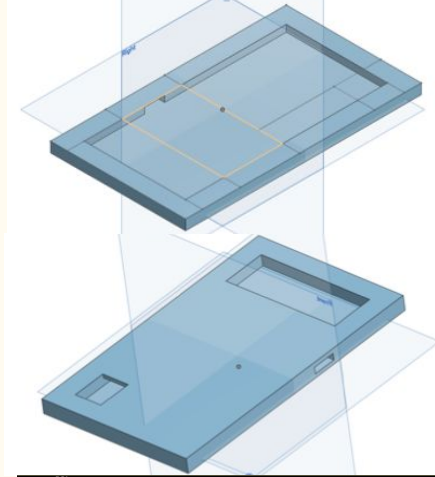
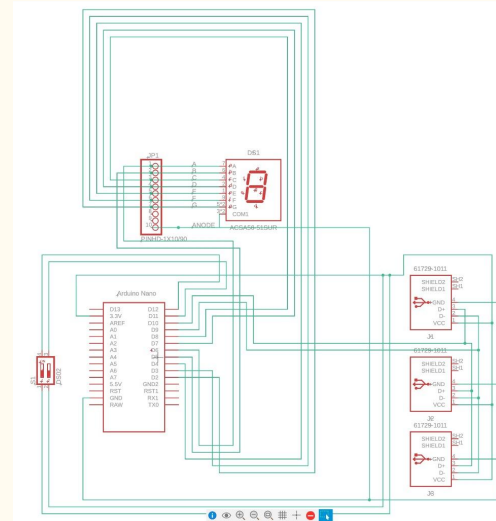
Finally, here is a close-up view of the individual data bits when transmitting binary 0100. This shows the presence of the short spikes in between every data bit, presumably where the transistor is switched off (internal bus tri-stated?) and the data line floats high again.



```
1 // Serial port connected to the data line (Pin 2 - Rx)
2 const int dataPin = 2;
3
4 void setup() {
5   // Setup the data pin as an OUTPUT
6   pinMode(dataPin, OUTPUT);
7
8   // Begin serial communication for debugging (optional)
9   Serial.begin(9600);
10
11 void loop() {
12   // Wait for a brief moment before sending the command
13   delayMicroseconds(15);
14
15   // Send the 24-bit command sequence
16   sendCommand();
17
18   // Wait for the controller response (adjust the delay based on your requirements)
19   delay(10);
20
21   void readCommand() {
22     // Define the 24-bit command sequence
23     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
24
25     // Send through each byte of the command
26     for (int i = 0; i < sizeof(command); i++) {
27       Serial.write(command[i]);
28     }
29
30     // Wait for the controller response (adjust the delay based on your requirements)
31     delay(10);
32
33     // Read the 64-bit response
34     for (int i = 0; i < 64; i++) {
35       bitRead(response, i);
36     }
37
38     // Convert the response to a string of bits
39     String responseStr = "";
40     for (int i = 0; i < 64; i++) {
41       responseStr += bitRead(response, i) ? "1" : "0";
42     }
43
44     // Print the response to the serial port
45     Serial.println(responseStr);
46
47     // Wait for the specified timing (adjust as needed)
48     delayMicroseconds(15);
49
50     // Set the data line high
51     digitalWrite(dataPin, HIGH);
52
53     // Wait for the specified timing (adjust as needed)
54     delayMicroseconds(15);
55
56   }
57
58   // Before the pin connected to the data line (Pin 2 - Rx)
59   const int dataPin = 2;
60
61   // Initialize the data pin as an OUTPUT
62   pinMode(dataPin, OUTPUT);
63
64 void loop() {
65   // Wait for a brief moment before sending the command
66   delayMicroseconds(15);
67
68   // Send the 24-bit command sequence
69   sendCommand();
70
71   // Wait for the controller response (adjust the delay based on your requirements)
72   delay(10);
73
74   void readCommand() {
75     // Define the 24-bit command sequence
76     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
77
78     // Send through each byte of the command
79     for (int i = 0; i < sizeof(command); i++) {
80       Serial.write(command[i]);
81     }
82
83     // Wait for the controller response (adjust the delay based on your requirements)
84     delay(10);
85
86     // Read the 64-bit response
87     for (int i = 0; i < 64; i++) {
88       bitRead(response, i);
89     }
90
91     // Convert the response to a string of bits
92     String responseStr = "";
93     for (int i = 0; i < 64; i++) {
94       responseStr += bitRead(response, i) ? "1" : "0";
95     }
96
97     // Print the response to the serial port
98     Serial.println(responseStr);
99
100    // Wait for the specified timing (adjust as needed)
101    delayMicroseconds(15);
102
103    // Set the data line high
104    digitalWrite(dataPin, HIGH);
105
106    // Wait for the specified timing (adjust as needed)
107    delayMicroseconds(15);
108
109  }
110
111  // Before the pin connected to the data line (Pin 2 - Rx)
112  const int dataPin = 2;
113
114  // Initialize the data pin as an OUTPUT
115  pinMode(dataPin, OUTPUT);
116
117 void loop() {
118   // Wait for a brief moment before sending the command
119   delayMicroseconds(15);
120
121   // Send the 24-bit command sequence
122   sendCommand();
123
124   // Wait for the controller response (adjust the delay based on your requirements)
125   delay(10);
126
127   void readCommand() {
128     // Define the 24-bit command sequence
129     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
130
131     // Send through each byte of the command
132     for (int i = 0; i < sizeof(command); i++) {
133       Serial.write(command[i]);
134     }
135
136     // Wait for the controller response (adjust the delay based on your requirements)
137     delay(10);
138
139     // Read the 64-bit response
140     for (int i = 0; i < 64; i++) {
141       bitRead(response, i);
142     }
143
144     // Convert the response to a string of bits
145     String responseStr = "";
146     for (int i = 0; i < 64; i++) {
147       responseStr += bitRead(response, i) ? "1" : "0";
148     }
149
150     // Print the response to the serial port
151     Serial.println(responseStr);
152
153     // Wait for the specified timing (adjust as needed)
154     delayMicroseconds(15);
155
156     // Set the data line high
157     digitalWrite(dataPin, HIGH);
158
159     // Wait for the specified timing (adjust as needed)
160     delayMicroseconds(15);
161
162   }
163
164   // Before the pin connected to the data line (Pin 2 - Rx)
165   const int dataPin = 2;
166
167   // Initialize the data pin as an OUTPUT
168   pinMode(dataPin, OUTPUT);
169
170 void loop() {
171   // Wait for a brief moment before sending the command
172   delayMicroseconds(15);
173
174   // Send the 24-bit command sequence
175   sendCommand();
176
177   // Wait for the controller response (adjust the delay based on your requirements)
178   delay(10);
179
180   void readCommand() {
181     // Define the 24-bit command sequence
182     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
183
184     // Send through each byte of the command
185     for (int i = 0; i < sizeof(command); i++) {
186       Serial.write(command[i]);
187     }
188
189     // Wait for the controller response (adjust the delay based on your requirements)
190     delay(10);
191
192     // Read the 64-bit response
193     for (int i = 0; i < 64; i++) {
194       bitRead(response, i);
195     }
196
197     // Convert the response to a string of bits
198     String responseStr = "";
199     for (int i = 0; i < 64; i++) {
200       responseStr += bitRead(response, i) ? "1" : "0";
201     }
202
203     // Print the response to the serial port
204     Serial.println(responseStr);
205
206     // Wait for the specified timing (adjust as needed)
207     delayMicroseconds(15);
208
209     // Set the data line high
210     digitalWrite(dataPin, HIGH);
211
212     // Wait for the specified timing (adjust as needed)
213     delayMicroseconds(15);
214
215   }
216
217   // Before the pin connected to the data line (Pin 2 - Rx)
218   const int dataPin = 2;
219
220   // Initialize the data pin as an OUTPUT
221   pinMode(dataPin, OUTPUT);
222
223 void loop() {
224   // Wait for a brief moment before sending the command
225   delayMicroseconds(15);
226
227   // Send the 24-bit command sequence
228   sendCommand();
229
230   // Wait for the controller response (adjust the delay based on your requirements)
231   delay(10);
232
233   void readCommand() {
234     // Define the 24-bit command sequence
235     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
236
237     // Send through each byte of the command
238     for (int i = 0; i < sizeof(command); i++) {
239       Serial.write(command[i]);
240     }
241
242     // Wait for the controller response (adjust the delay based on your requirements)
243     delay(10);
244
245     // Read the 64-bit response
246     for (int i = 0; i < 64; i++) {
247       bitRead(response, i);
248     }
249
250     // Convert the response to a string of bits
251     String responseStr = "";
252     for (int i = 0; i < 64; i++) {
253       responseStr += bitRead(response, i) ? "1" : "0";
254     }
255
256     // Print the response to the serial port
257     Serial.println(responseStr);
258
259     // Wait for the specified timing (adjust as needed)
260     delayMicroseconds(15);
261
262     // Set the data line high
263     digitalWrite(dataPin, HIGH);
264
265     // Wait for the specified timing (adjust as needed)
266     delayMicroseconds(15);
267
268   }
269
270   // Before the pin connected to the data line (Pin 2 - Rx)
271   const int dataPin = 2;
272
273   // Initialize the data pin as an OUTPUT
274   pinMode(dataPin, OUTPUT);
275
276 void loop() {
277   // Wait for a brief moment before sending the command
278   delayMicroseconds(15);
279
280   // Send the 24-bit command sequence
281   sendCommand();
282
283   // Wait for the controller response (adjust the delay based on your requirements)
284   delay(10);
285
286   void readCommand() {
287     // Define the 24-bit command sequence
288     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
289
290     // Send through each byte of the command
291     for (int i = 0; i < sizeof(command); i++) {
292       Serial.write(command[i]);
293     }
294
295     // Wait for the controller response (adjust the delay based on your requirements)
296     delay(10);
297
298     // Read the 64-bit response
299     for (int i = 0; i < 64; i++) {
300       bitRead(response, i);
301     }
302
303     // Convert the response to a string of bits
304     String responseStr = "";
305     for (int i = 0; i < 64; i++) {
306       responseStr += bitRead(response, i) ? "1" : "0";
307     }
308
309     // Print the response to the serial port
310     Serial.println(responseStr);
311
312     // Wait for the specified timing (adjust as needed)
313     delayMicroseconds(15);
314
315     // Set the data line high
316     digitalWrite(dataPin, HIGH);
317
318     // Wait for the specified timing (adjust as needed)
319     delayMicroseconds(15);
320
321   }
322
323   // Before the pin connected to the data line (Pin 2 - Rx)
324   const int dataPin = 2;
325
326   // Initialize the data pin as an OUTPUT
327   pinMode(dataPin, OUTPUT);
328
329 void loop() {
330   // Wait for a brief moment before sending the command
331   delayMicroseconds(15);
332
333   // Send the 24-bit command sequence
334   sendCommand();
335
336   // Wait for the controller response (adjust the delay based on your requirements)
337   delay(10);
338
339   void readCommand() {
340     // Define the 24-bit command sequence
341     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
342
343     // Send through each byte of the command
344     for (int i = 0; i < sizeof(command); i++) {
345       Serial.write(command[i]);
346     }
347
348     // Wait for the controller response (adjust the delay based on your requirements)
349     delay(10);
350
351     // Read the 64-bit response
352     for (int i = 0; i < 64; i++) {
353       bitRead(response, i);
354     }
355
356     // Convert the response to a string of bits
357     String responseStr = "";
358     for (int i = 0; i < 64; i++) {
359       responseStr += bitRead(response, i) ? "1" : "0";
360     }
361
362     // Print the response to the serial port
363     Serial.println(responseStr);
364
365     // Wait for the specified timing (adjust as needed)
366     delayMicroseconds(15);
367
368     // Set the data line high
369     digitalWrite(dataPin, HIGH);
370
371     // Wait for the specified timing (adjust as needed)
372     delayMicroseconds(15);
373
374   }
375
376   // Before the pin connected to the data line (Pin 2 - Rx)
377   const int dataPin = 2;
378
379   // Initialize the data pin as an OUTPUT
380   pinMode(dataPin, OUTPUT);
381
382 void loop() {
383   // Wait for a brief moment before sending the command
384   delayMicroseconds(15);
385
386   // Send the 24-bit command sequence
387   sendCommand();
388
389   // Wait for the controller response (adjust the delay based on your requirements)
390   delay(10);
391
392   void readCommand() {
393     // Define the 24-bit command sequence
394     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
395
396     // Send through each byte of the command
397     for (int i = 0; i < sizeof(command); i++) {
398       Serial.write(command[i]);
399     }
400
401     // Wait for the controller response (adjust the delay based on your requirements)
402     delay(10);
403
404     // Read the 64-bit response
405     for (int i = 0; i < 64; i++) {
406       bitRead(response, i);
407     }
408
409     // Convert the response to a string of bits
410     String responseStr = "";
411     for (int i = 0; i < 64; i++) {
412       responseStr += bitRead(response, i) ? "1" : "0";
413     }
414
415     // Print the response to the serial port
416     Serial.println(responseStr);
417
418     // Wait for the specified timing (adjust as needed)
419     delayMicroseconds(15);
420
421     // Set the data line high
422     digitalWrite(dataPin, HIGH);
423
424     // Wait for the specified timing (adjust as needed)
425     delayMicroseconds(15);
426
427   }
428
429   // Before the pin connected to the data line (Pin 2 - Rx)
430   const int dataPin = 2;
431
432   // Initialize the data pin as an OUTPUT
433   pinMode(dataPin, OUTPUT);
434
435 void loop() {
436   // Wait for a brief moment before sending the command
437   delayMicroseconds(15);
438
439   // Send the 24-bit command sequence
440   sendCommand();
441
442   // Wait for the controller response (adjust the delay based on your requirements)
443   delay(10);
444
445   void readCommand() {
446     // Define the 24-bit command sequence
447     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
448
449     // Send through each byte of the command
450     for (int i = 0; i < sizeof(command); i++) {
451       Serial.write(command[i]);
452     }
453
454     // Wait for the controller response (adjust the delay based on your requirements)
455     delay(10);
456
457     // Read the 64-bit response
458     for (int i = 0; i < 64; i++) {
459       bitRead(response, i);
460     }
461
462     // Convert the response to a string of bits
463     String responseStr = "";
464     for (int i = 0; i < 64; i++) {
465       responseStr += bitRead(response, i) ? "1" : "0";
466     }
467
468     // Print the response to the serial port
469     Serial.println(responseStr);
470
471     // Wait for the specified timing (adjust as needed)
472     delayMicroseconds(15);
473
474     // Set the data line high
475     digitalWrite(dataPin, HIGH);
476
477     // Wait for the specified timing (adjust as needed)
478     delayMicroseconds(15);
479
480   }
481
482   // Before the pin connected to the data line (Pin 2 - Rx)
483   const int dataPin = 2;
484
485   // Initialize the data pin as an OUTPUT
486   pinMode(dataPin, OUTPUT);
487
488 void loop() {
489   // Wait for a brief moment before sending the command
490   delayMicroseconds(15);
491
492   // Send the 24-bit command sequence
493   sendCommand();
494
495   // Wait for the controller response (adjust the delay based on your requirements)
496   delay(10);
497
498   void readCommand() {
499     // Define the 24-bit command sequence
500     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
501
502     // Send through each byte of the command
503     for (int i = 0; i < sizeof(command); i++) {
504       Serial.write(command[i]);
505     }
506
507     // Wait for the controller response (adjust the delay based on your requirements)
508     delay(10);
509
510     // Read the 64-bit response
511     for (int i = 0; i < 64; i++) {
512       bitRead(response, i);
513     }
514
515     // Convert the response to a string of bits
516     String responseStr = "";
517     for (int i = 0; i < 64; i++) {
518       responseStr += bitRead(response, i) ? "1" : "0";
519     }
520
521     // Print the response to the serial port
522     Serial.println(responseStr);
523
524     // Wait for the specified timing (adjust as needed)
525     delayMicroseconds(15);
526
527     // Set the data line high
528     digitalWrite(dataPin, HIGH);
529
530     // Wait for the specified timing (adjust as needed)
531     delayMicroseconds(15);
532
533   }
534
535   // Before the pin connected to the data line (Pin 2 - Rx)
536   const int dataPin = 2;
537
538   // Initialize the data pin as an OUTPUT
539   pinMode(dataPin, OUTPUT);
540
541 void loop() {
542   // Wait for a brief moment before sending the command
543   delayMicroseconds(15);
544
545   // Send the 24-bit command sequence
546   sendCommand();
547
548   // Wait for the controller response (adjust the delay based on your requirements)
549   delay(10);
550
551   void readCommand() {
552     // Define the 24-bit command sequence
553     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
554
555     // Send through each byte of the command
556     for (int i = 0; i < sizeof(command); i++) {
557       Serial.write(command[i]);
558     }
559
560     // Wait for the controller response (adjust the delay based on your requirements)
561     delay(10);
562
563     // Read the 64-bit response
564     for (int i = 0; i < 64; i++) {
565       bitRead(response, i);
566     }
567
568     // Convert the response to a string of bits
569     String responseStr = "";
570     for (int i = 0; i < 64; i++) {
571       responseStr += bitRead(response, i) ? "1" : "0";
572     }
573
574     // Print the response to the serial port
575     Serial.println(responseStr);
576
577     // Wait for the specified timing (adjust as needed)
578     delayMicroseconds(15);
579
580     // Set the data line high
581     digitalWrite(dataPin, HIGH);
582
583     // Wait for the specified timing (adjust as needed)
584     delayMicroseconds(15);
585
586   }
587
588   // Before the pin connected to the data line (Pin 2 - Rx)
589   const int dataPin = 2;
590
591   // Initialize the data pin as an OUTPUT
592   pinMode(dataPin, OUTPUT);
593
594 void loop() {
595   // Wait for a brief moment before sending the command
596   delayMicroseconds(15);
597
598   // Send the 24-bit command sequence
599   sendCommand();
600
601   // Wait for the controller response (adjust the delay based on your requirements)
602   delay(10);
603
604   void readCommand() {
605     // Define the 24-bit command sequence
606     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
607
608     // Send through each byte of the command
609     for (int i = 0; i < sizeof(command); i++) {
610       Serial.write(command[i]);
611     }
612
613     // Wait for the controller response (adjust the delay based on your requirements)
614     delay(10);
615
616     // Read the 64-bit response
617     for (int i = 0; i < 64; i++) {
618       bitRead(response, i);
619     }
620
621     // Convert the response to a string of bits
622     String responseStr = "";
623     for (int i = 0; i < 64; i++) {
624       responseStr += bitRead(response, i) ? "1" : "0";
625     }
626
627     // Print the response to the serial port
628     Serial.println(responseStr);
629
630     // Wait for the specified timing (adjust as needed)
631     delayMicroseconds(15);
632
633     // Set the data line high
634     digitalWrite(dataPin, HIGH);
635
636     // Wait for the specified timing (adjust as needed)
637     delayMicroseconds(15);
638
639   }
640
641   // Before the pin connected to the data line (Pin 2 - Rx)
642   const int dataPin = 2;
643
644   // Initialize the data pin as an OUTPUT
645   pinMode(dataPin, OUTPUT);
646
647 void loop() {
648   // Wait for a brief moment before sending the command
649   delayMicroseconds(15);
650
651   // Send the 24-bit command sequence
652   sendCommand();
653
654   // Wait for the controller response (adjust the delay based on your requirements)
655   delay(10);
656
657   void readCommand() {
658     // Define the 24-bit command sequence
659     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
660
661     // Send through each byte of the command
662     for (int i = 0; i < sizeof(command); i++) {
663       Serial.write(command[i]);
664     }
665
666     // Wait for the controller response (adjust the delay based on your requirements)
667     delay(10);
668
669     // Read the 64-bit response
670     for (int i = 0; i < 64; i++) {
671       bitRead(response, i);
672     }
673
674     // Convert the response to a string of bits
675     String responseStr = "";
676     for (int i = 0; i < 64; i++) {
677       responseStr += bitRead(response, i) ? "1" : "0";
678     }
679
680     // Print the response to the serial port
681     Serial.println(responseStr);
682
683     // Wait for the specified timing (adjust as needed)
684     delayMicroseconds(15);
685
686     // Set the data line high
687     digitalWrite(dataPin, HIGH);
688
689     // Wait for the specified timing (adjust as needed)
690     delayMicroseconds(15);
691
692   }
693
694   // Before the pin connected to the data line (Pin 2 - Rx)
695   const int dataPin = 2;
696
697   // Initialize the data pin as an OUTPUT
698   pinMode(dataPin, OUTPUT);
699
700 void loop() {
701   // Wait for a brief moment before sending the command
702   delayMicroseconds(15);
703
704   // Send the 24-bit command sequence
705   sendCommand();
706
707   // Wait for the controller response (adjust the delay based on your requirements)
708   delay(10);
709
710   void readCommand() {
711     // Define the 24-bit command sequence
712     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
713
714     // Send through each byte of the command
715     for (int i = 0; i < sizeof(command); i++) {
716       Serial.write(command[i]);
717     }
718
719     // Wait for the controller response (adjust the delay based on your requirements)
720     delay(10);
721
722     // Read the 64-bit response
723     for (int i = 0; i < 64; i++) {
724       bitRead(response, i);
725     }
726
727     // Convert the response to a string of bits
728     String responseStr = "";
729     for (int i = 0; i < 64; i++) {
730       responseStr += bitRead(response, i) ? "1" : "0";
731     }
732
733     // Print the response to the serial port
734     Serial.println(responseStr);
735
736     // Wait for the specified timing (adjust as needed)
737     delayMicroseconds(15);
738
739     // Set the data line high
740     digitalWrite(dataPin, HIGH);
741
742     // Wait for the specified timing (adjust as needed)
743     delayMicroseconds(15);
744
745   }
746
747   // Before the pin connected to the data line (Pin 2 - Rx)
748   const int dataPin = 2;
749
750   // Initialize the data pin as an OUTPUT
751   pinMode(dataPin, OUTPUT);
752
753 void loop() {
754   // Wait for a brief moment before sending the command
755   delayMicroseconds(15);
756
757   // Send the 24-bit command sequence
758   sendCommand();
759
760   // Wait for the controller response (adjust the delay based on your requirements)
761   delay(10);
762
763   void readCommand() {
764     // Define the 24-bit command sequence
765     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
766
767     // Send through each byte of the command
768     for (int i = 0; i < sizeof(command); i++) {
769       Serial.write(command[i]);
770     }
771
772     // Wait for the controller response (adjust the delay based on your requirements)
773     delay(10);
774
775     // Read the 64-bit response
776     for (int i = 0; i < 64; i++) {
777       bitRead(response, i);
778     }
779
780     // Convert the response to a string of bits
781     String responseStr = "";
782     for (int i = 0; i < 64; i++) {
783       responseStr += bitRead(response, i) ? "1" : "0";
784     }
785
786     // Print the response to the serial port
787     Serial.println(responseStr);
788
789     // Wait for the specified timing (adjust as needed)
790     delayMicroseconds(15);
791
792     // Set the data line high
793     digitalWrite(dataPin, HIGH);
794
795     // Wait for the specified timing (adjust as needed)
796     delayMicroseconds(15);
797
798   }
799
800   // Before the pin connected to the data line (Pin 2 - Rx)
801   const int dataPin = 2;
802
803   // Initialize the data pin as an OUTPUT
804   pinMode(dataPin, OUTPUT);
805
806 void loop() {
807   // Wait for a brief moment before sending the command
808   delayMicroseconds(15);
809
810   // Send the 24-bit command sequence
811   sendCommand();
812
813   // Wait for the controller response (adjust the delay based on your requirements)
814   delay(10);
815
816   void readCommand() {
817     // Define the 24-bit command sequence
818     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
819
820     // Send through each byte of the command
821     for (int i = 0; i < sizeof(command); i++) {
822       Serial.write(command[i]);
823     }
824
825     // Wait for the controller response (adjust the delay based on your requirements)
826     delay(10);
827
828     // Read the 64-bit response
829     for (int i = 0; i < 64; i++) {
830       bitRead(response, i);
831     }
832
833     // Convert the response to a string of bits
834     String responseStr = "";
835     for (int i = 0; i < 64; i++) {
836       responseStr += bitRead(response, i) ? "1" : "0";
837     }
838
839     // Print the response to the serial port
840     Serial.println(responseStr);
841
842     // Wait for the specified timing (adjust as needed)
843     delayMicroseconds(15);
844
845     // Set the data line high
846     digitalWrite(dataPin, HIGH);
847
848     // Wait for the specified timing (adjust as needed)
849     delayMicroseconds(15);
850
851   }
852
853   // Before the pin connected to the data line (Pin 2 - Rx)
854   const int dataPin = 2;
855
856   // Initialize the data pin as an OUTPUT
857   pinMode(dataPin, OUTPUT);
858
859 void loop() {
860   // Wait for a brief moment before sending the command
861   delayMicroseconds(15);
862
863   // Send the 24-bit command sequence
864   sendCommand();
865
866   // Wait for the controller response (adjust the delay based on your requirements)
867   delay(10);
868
869   void readCommand() {
870     // Define the 24-bit command sequence
871     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
872
873     // Send through each byte of the command
874     for (int i = 0; i < sizeof(command); i++) {
875       Serial.write(command[i]);
876     }
877
878     // Wait for the controller response (adjust the delay based on your requirements)
879     delay(10);
880
881     // Read the 64-bit response
882     for (int i = 0; i < 64; i++) {
883       bitRead(response, i);
884     }
885
886     // Convert the response to a string of bits
887     String responseStr = "";
888     for (int i = 0; i < 64; i++) {
889       responseStr += bitRead(response, i) ? "1" : "0";
890     }
891
892     // Print the response to the serial port
893     Serial.println(responseStr);
894
895     // Wait for the specified timing (adjust as needed)
896     delayMicroseconds(15);
897
898     // Set the data line high
899     digitalWrite(dataPin, HIGH);
900
901     // Wait for the specified timing (adjust as needed)
902     delayMicroseconds(15);
903
904   }
905
906   // Before the pin connected to the data line (Pin 2 - Rx)
907   const int dataPin = 2;
908
909   // Initialize the data pin as an OUTPUT
910   pinMode(dataPin, OUTPUT);
911
912 void loop() {
913   // Wait for a brief moment before sending the command
914   delayMicroseconds(15);
915
916   // Send the 24-bit command sequence
917   sendCommand();
918
919   // Wait for the controller response (adjust the delay based on your requirements)
920   delay(10);
921
922   void readCommand() {
923     // Define the 24-bit command sequence
924     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
925
926     // Send through each byte of the command
927     for (int i = 0; i < sizeof(command); i++) {
928       Serial.write(command[i]);
929     }
930
931     // Wait for the controller response (adjust the delay based on your requirements)
932     delay(10);
933
934     // Read the 64-bit response
935     for (int i = 0; i < 64; i++) {
936       bitRead(response, i);
937     }
938
939     // Convert the response to a string of bits
940     String responseStr = "";
941     for (int i = 0; i < 64; i++) {
942       responseStr += bitRead(response, i) ? "1" : "0";
943     }
944
945     // Print the response to the serial port
946     Serial.println(responseStr);
947
948     // Wait for the specified timing (adjust as needed)
949     delayMicroseconds(15);
950
951     // Set the data line high
952     digitalWrite(dataPin, HIGH);
953
954     // Wait for the specified timing (adjust as needed)
955     delayMicroseconds(15);
956
957   }
958
959   // Before the pin connected to the data line (Pin 2 - Rx)
960   const int dataPin = 2;
961
962   // Initialize the data pin as an OUTPUT
963   pinMode(dataPin, OUTPUT);
964
965 void loop() {
966   // Wait for a brief moment before sending the command
967   delayMicroseconds(15);
968
969   // Send the 24-bit command sequence
970   sendCommand();
971
972   // Wait for the controller response (adjust the delay based on your requirements)
973   delay(10);
974
975   void readCommand() {
976     // Define the 24-bit command sequence
977     byte command[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
978
979     // Send through each byte of the command
980     for (int i = 0; i < sizeof(command); i++) {
981       Serial.write(command[i]);
982     }
983
984     // Wait for the controller response (adjust the delay based on your requirements)
985     delay(10);
986
987     // Read the 64-bit response
988
```

The final design is as follows:

A series of data lines from some USB compatible headers, which can be connected to any kind of game controller using soldering and wires after the fact, a dip switch able to tell the arduino which kind of controllers to process (if ever further developed for), as well as the top case and bottom case. This includes slots for the LED display, a programmer cable, and leaves room for any game controller port header.



# Conclusion

This project had a lot of eureka moments, difficulties in development, hurdles in difficulty, expectation issues, etc.

Overall, I feel i know a lot more about the difficulties of taking a conceptual product on a breadboard, or even in a VHDL simulation environment, and producing it into an unchangeable and hard wired environment in which the system can run and live forever. I definitely will need the context of this project and this class to further my skill set in development/design of digital systems.