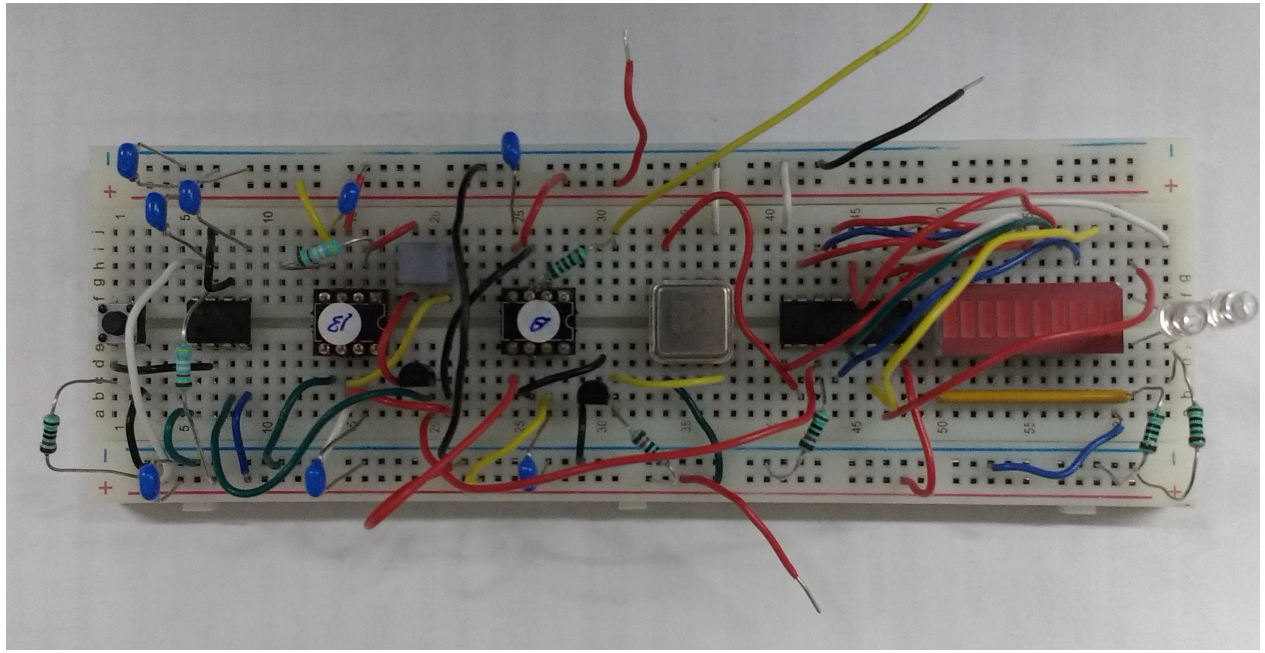
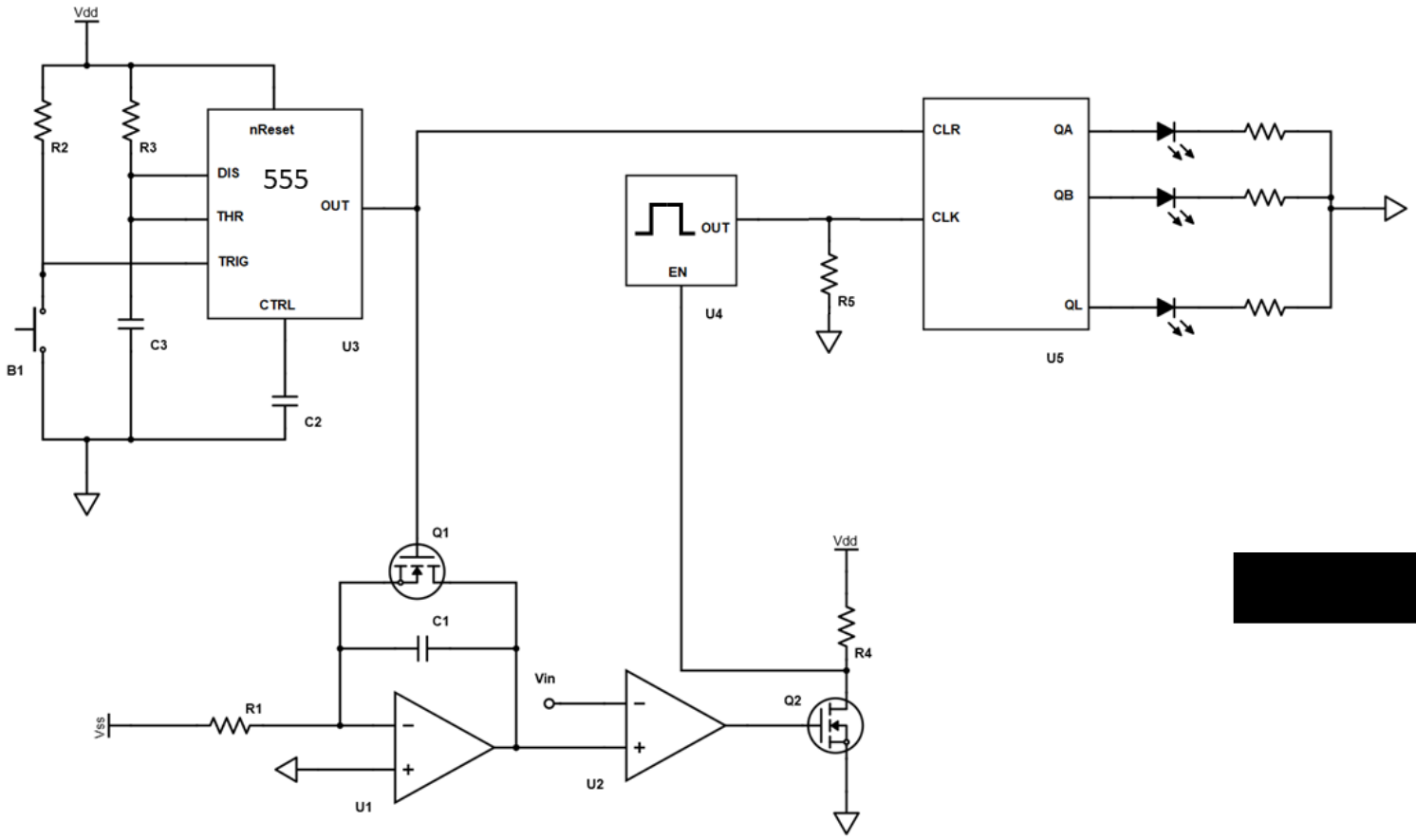
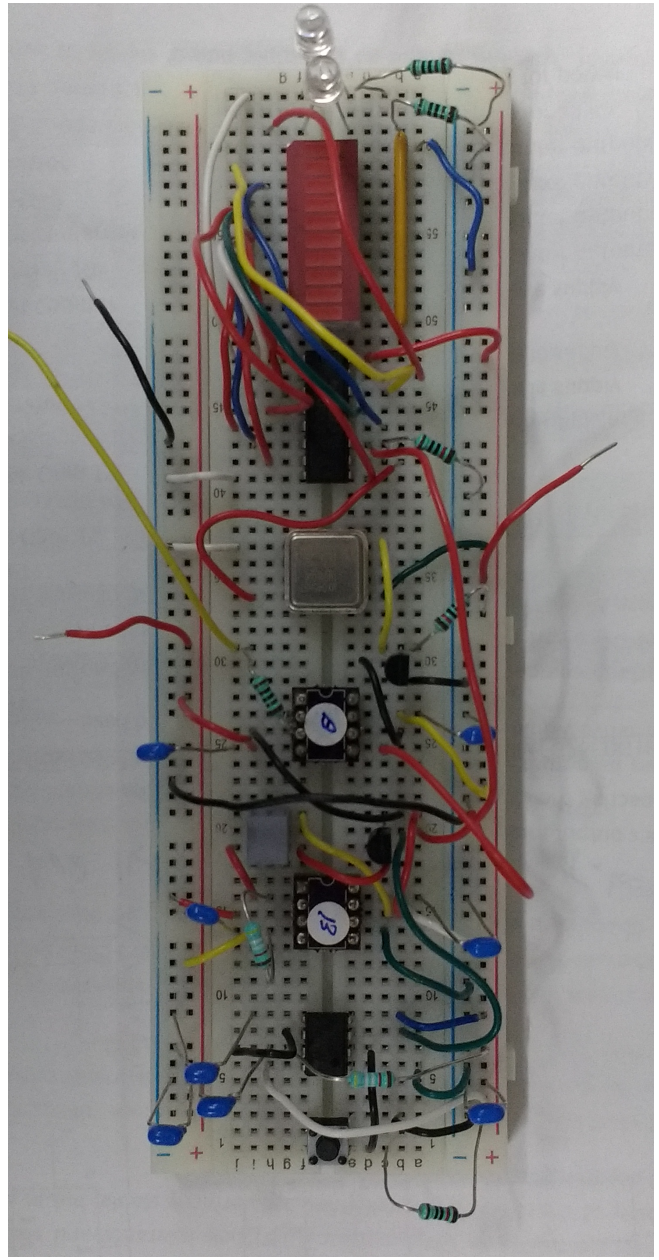


```c
//B. A. Bryce 2016
//GPIO STM32l0 CMSIS example

#include "stm32l0xx.h"

// Macros
#define NOP() __asm volatile ( "NOP" )

// Prototypes
void nopDelay(unsigned long delay);
unsigned short sendSPI1(unsigned short outDat);

int main(void)
{
    //****************
    //setup clocks and hardware
    //default clock is 8 MHz internal

    //Clocks:
    //enable clock for GPIOC
    RCC->CR |= RCC_CR_HSION;
    RCC->APB1ENR |= RCC_APB1ENR_PWREN;
    RCC->IOPENR |= RCC_IOPENR_GPIOCEN | RCC_IOPENR_GPIOBEN;
    RCC->CFGR |= RCC_CFGR_SW_HSI;

```

Why doesn’t it work?

Finding and solving bugs in complex systems





A common route to bugs: I know what to do. I will do it. It does not work.

For instance take a schematic and assemble the entire thing and then at the end check if it works. You probably did this when you put together your robot's PCB.

This is marginally okay even if you know it should work it's a very bad idea if you don't

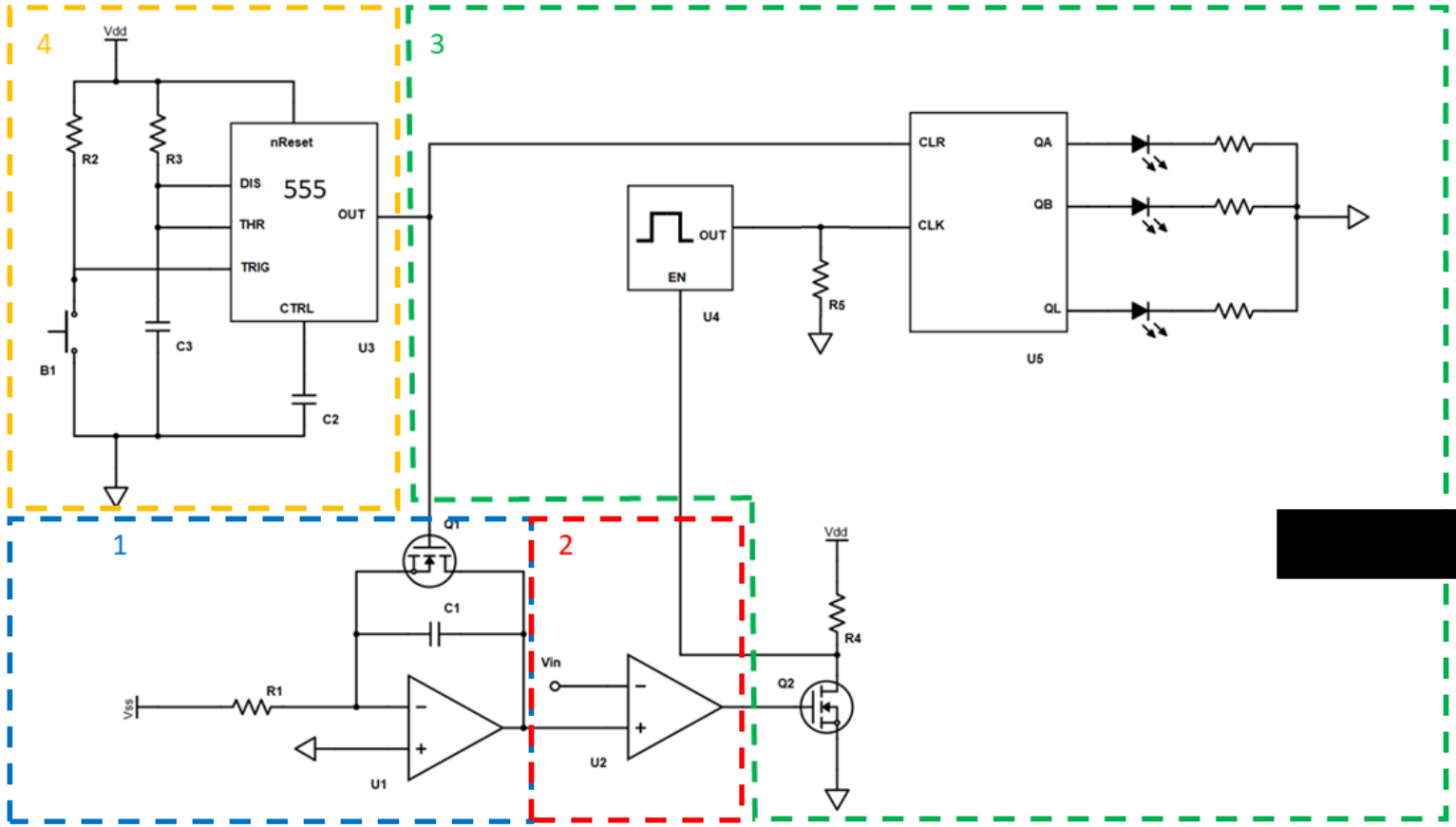
A large part of good engineering is about modules and modularity. It is best if you create a system from a set of sub-systems that have clear interfaces and functions.

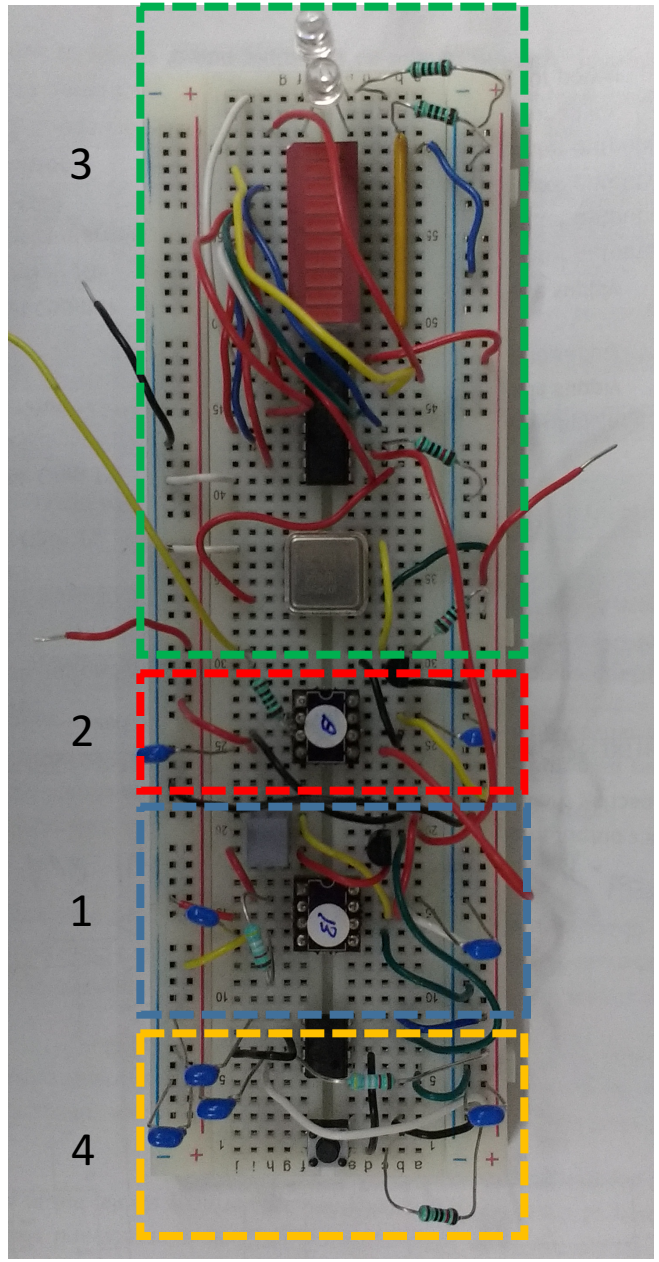
This applies to software and hardware!

This has numerous advantages. The biggest two of which are: reusability and robustness.

Reusability means you can combine building blocks together again to solve new problems without redoing work.

Robustness comes from meeting the interface specification. If a vendor discontinues a part, you might have to redesign a module but not the entire system so long as your new module meets to the interface specification.





Modules should be *testable* and have a clear purpose.

The granularity of your modules can be anything but it is wise to partition your hardware or code into modules of a reasonable size.

Takeaways:

- Before you start break down the problem into modules
- Consider how you will test these modules
- Test the modules as you build them

My module is broken. What now?

Even with the best planning errors will be made and you will have modules that do not work to the interface specification.

With good planning these modules will be simple enough to debug.

Harder bugs occur when your *interface* is broken. If your interface specification overlooks a case you will have edge case bugs and these can be harder to find. If both modules pass tests individually it is likely the interface that has the problem.

I am thinking of an object in the room, what is it?

This game normally called 20 questions in the US contains the core of how to debug a system.

You ask big questions first that eliminate large numbers of things and then smaller questions later to get to the exact source of the error. Even with binary answers you can quickly resolve a very open ended unknown.

How do we ask a question in an engineered system?

We make a measurement of a signal. That signal may be asserted by software or it might be made on real time hardware with an instrument.

To understand if the answer indicates a bug we have to have a model of the expected behavior of the system.

Common electronic hardware questions to ask (tool to ask with):

- Is point A connected to point B? (multimeter continuity)
- Is the device powered properly? (multimeter volts), consulting datasheet
- Is there a short on the board? (current draw from power supply or multimeter amps)
- Does the signal look right? (Oscilloscope or other analysis tool).

Our sources of information (feedback) are instruments in the lab. This is of course just a partial list.

The key idea is an expected value and a test of that value using feedback to locate the inconsistency. It may be that you misunderstood the datasheet or it may be that your part is broken only when you find the inconsistency can you think deeper about which it is.

Common electronic software questions to ask (tool to ask with):

- Did my code run to here? (LED, print statement, “debugger”)
- Did my code produce the expected output? (LED, Test function, print statement)
- Does my software take the correct path through functions? (print to logs, debugger)
- How long did my code take to run? (Profiler, tick/toc/timer)

Different kinds of software have different debugging available. When you are building a “real-time” system pausing the execution of the software can change the result or make the code not work at all. This requires log type debugging approaches. If you are writing software for a system that does not depend on an external real-time stream of information you can pause the execution with a debugger.

Is it a hardware bug or a software bug?

With systems like we have in E11 it can sometimes be hard to tell if the bug is in hardware or software as they interact.

Test your software with simulated hardware inputs.

Test your hardware with tools in the lab as needed if it is a hardware bug.

Let’s do an example…

```
#define LED1 13
#define LED2 10
#define LED3 5
#define LED4 2

unsigned char thePattern = 1;
unsigned char forwardBackward = 1;

void setup()
{
 Serial.begin(9600);
 pinMode(LED1, OUTPUT);
 pinMode(LED2, OUTPUT);
 pinMode(LED3, OUTPUT);
 pinMode(LED4, OUTPUT);
}

void loop()
{
 displayPattern(thePattern);         //display the pattern
 delay(1000);                        //let the user see the pattern
 thePattern = nextPattern(thePattern, forwardBackward); //go to the next pattern
 forwardBackward = !forwardBackward;

}

//displays the pattern on the LEDs
//function: 2 has only LED 2 on while 3 has LED 1 and 2 on
void displayPattern(unsigned char pattern)
{
 if(pattern >= 1)digitalWrite(LED1, HIGH);
 else digitalWrite(LED1, LOW);
 if(pattern >= 2)digitalWrite(LED2, HIGH);
 else digitalWrite(LED2, LOW);
 if(pattern >= 4)digitalWrite(LED3, HIGH);
 else digitalWrite(LED3, LOW);
 if(pattern >= 8)digitalWrite(LED4, HIGH);
 else digitalWrite(LED4, LOW);
}


//If forward return 2 greater, otherwise 1 less
unsigned char nextPattern(unsigned char currentPattern, unsigned char forward)
{
 if(forward == 1) return currentPattern + 2;
 else return currentPattern - 1;
}
```

Debug expected order for nextPattern:

1

3

2

4

3

5

4

6

```
//Fixed display pattern
void displayPattern(unsigned  char pattern)
{
 if(pattern & 0b1)digitalWrite(LED1,  HIGH);
 else digitalWrite(LED1, LOW);
 if(pattern & 0b10)digitalWrite(LED2,  HIGH);
 else digitalWrite(LED2, LOW);
 if(pattern & 0b100)digitalWrite(LED3,  HIGH);
 else digitalWrite(LED3, LOW);
 if(pattern & 0b1000)digitalWrite(LED4,  HIGH);
 else digitalWrite(LED4, LOW);
}
```