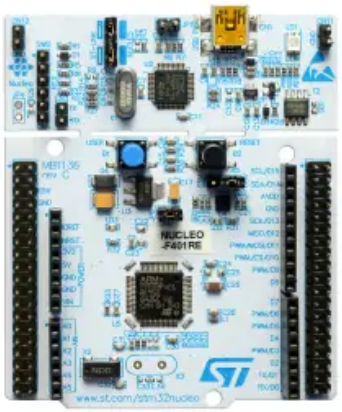
ECE 0202: EMBEDDED PROCESSORS AND INTERFACING

ISAAC SHAKER, JOSH MILANI, ANTHONY SPADAFORE

PROJECT SCOPE AND CONTEXT:

The purpose of this project is to mimic the functionality of an elevator using Assembly Language and an STM32-Nucleo microcontroller. The requirements of this design are as follows:

- **Door Motor**: rotation direction represents up/down motion of the elevator (counterclockwise = up, clockwise = down)
- **Lift Motor**: rotation direction represents opening/closing of the elevator doors (counterclockwise = opening, clockwise = closing)
- **Keypad**: allows users to register a destination level of the elevator
- **Call Buttons**: allows users to call the elevator to their floor
- **Emergency Button**: stops the elevator at the nearest floor
- **Admin Button**: pulls the elevator to the first floor and resets the sequence
- **LEDs**: LED lights up if the elevator is destined to stop at that floor
- **7-Segment Display**: indicates the current level of the elevator
- **Tera Term**: displays the current status and operations of the elevator



OUR APPROACH:

Our system relies on three primary values, which we labeled destination (*dest*), current (*curr*), and *direction*. These values act like global variables, and they are stored in registers seven, eight, and nine, respectively. The relationship between *curr*, *dest*, and *direction* is well defined in the "ELEVATOR ALGORITHM PSEUDOCODE" section below.

- *dest:* a bit vector that stores the floors that clients would like to visit (LSB = floor 1, MSB = floor 4). For example, if *dest* = 0010, then the elevator must visit the second floor. If *dest* = 1100, the elevator must visit the third and fourth floor.
- *curr:* another bit vector that stores the current location of the elevator. For instance, if *curr* = 1000, then the elevator is currently on the fourth floor. If *curr* = 0100, the elevator is on the third floor.
- *direction*: a two bit vector that stores the state of the elevator. There are three possible states: rest = 00, up = 01, and down = 10. The "SYSTEM DIAGRAM" section below includes a finite-state machine that specifies the traversal between these states.

Our algorithm constantly compares the *dest* and *curr* vector in order to determine the direction of the elevator. The destination vector is continuously changing due to keypad and button entries, which update the locations that the elevator must visit. To handle this dynamic variable, we took advantage of the SysTick interrupt.

The SysTick interrupt handles all of our inputs (keypad, emergency, admin, and call buttons). We have determined that utilizing polling would be most effective for our design. The world record for clicks per second is 16, or, in other words, around 63 milliseconds per click. Therefore, in our design, an interrupt is generated every 60 milliseconds, and the input registers are scanned for changes. If any inputs are detected, the system is changed accordingly. By setting our interrupt reload value to 4,799,999, we can identify most button clicks without disrupting the operation of the elevator.

CONSTRAINTS:

One constraint that we ran into was the number of GPIO pins available for use. We found out that PA2, PA3, PA5, PA13, PA14, PB3, and PC13 were already used by the board. This, along with the limited number of pins to begin with, put a constraint on the number of floors, buttons, and LEDs our elevator could potentially have.

Another constraint was the amount of space that we had to work with. We had several components to fit onto our breadboard including two motors, two motor controllers, a keypad, five buttons, 4 LEDs, a seven-segment display, and the microcontroller. This is a lot of components for one breadboard, limiting the ultimate size of our elevator.

Furthermore, we were also constrained by the memory of the microcontroller. Because of this, we had to write our code efficiently by reusing registers and making functions when necessary. While our code is not perfect, this constraint definitely motivated us to make our algorithm as efficient as possible.

SUCCESSES AND FAILURES:

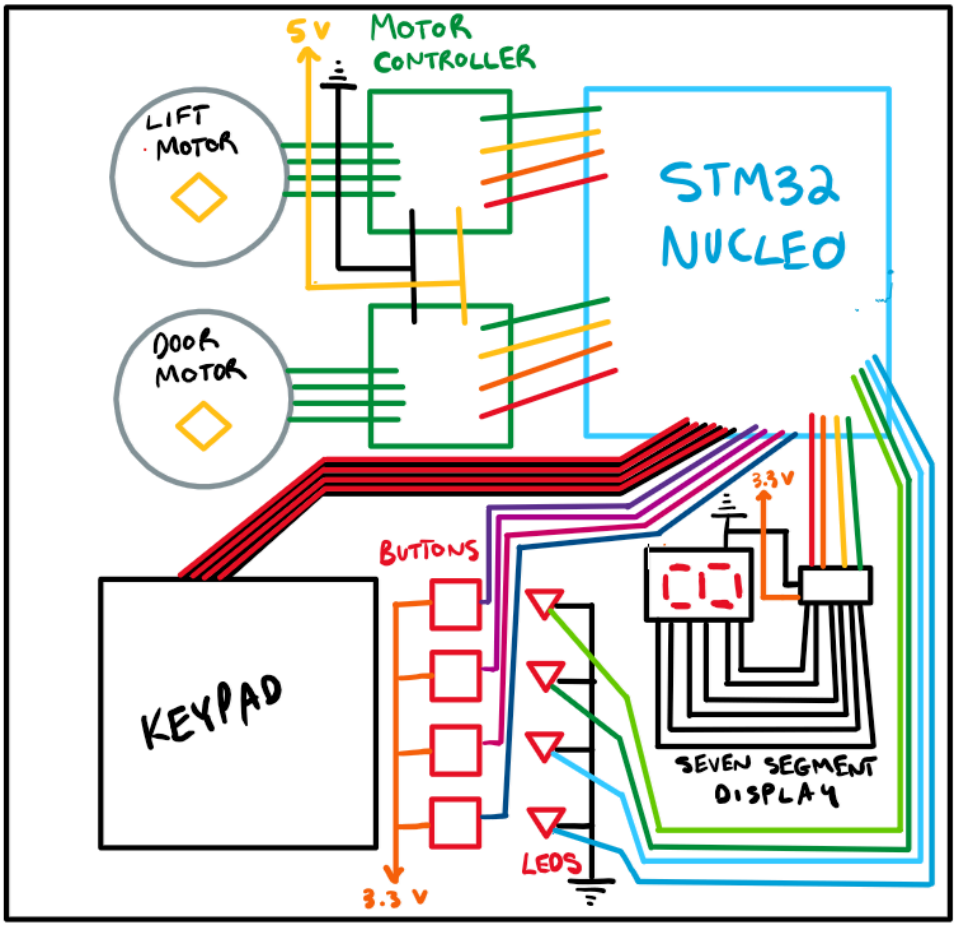
When we initially set up our SysTick interrupt, the program would branch to the included file, "stm32l476xx_constants.s," which had a SysTick_Handler process with an endless loop inside of it. Our program would enter this endless loop even though we had defined a SysTick_Handler of our own in the "main.s" file. In order to resolve this issue, we created a separate file called "systick.s," where we defined our SysTick_Handler and made sure to use the EXPORT directive so that our main file would recognize the function and override the one defined in the "stm32l476xx_constants.s" file.

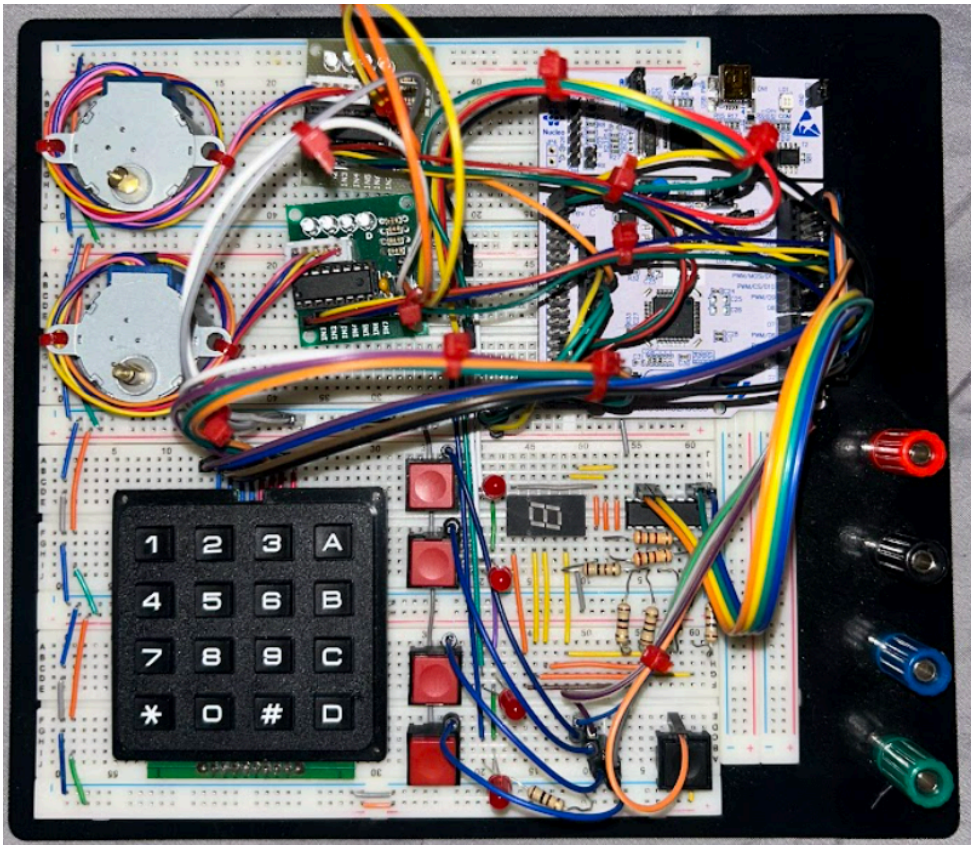
To continue, at first, our program did not display any text to Tera Term. We fixed this problem by writing "BL System_Clock_Init" and "BL UART2_Init" at the beginning of our code. While the program did display to Tera Term following the addition of these commands, our motors no longer worked. The whole program seemed to be sped up so fast that the motors could not operate properly. This was due to changes in the clock frequency after calling System_Clock_Init. To counteract this, we increased the value used in our delay function, and, afterwards, both our motors and Tera Term worked.

Ultimately, we ran into a problem with our admin button. When we pressed the admin button, our elevator acted as it was supposed to considering that all destinations were cleared and the elevator made its way to the first floor. Although, in the time that it took the elevator to reach the first floor, user's could still press the other buttons, causing the elevator to stop on its way down. Therefore, if the elevator was on the fourth floor and the admin button was pressed, another button press on the second floor would cause the elevator to stop on that floor before reaching the bottom. To prevent any user inputs during the admin process, we utilized register six, which we named *admin*. If the admin button was pressed, *admin* would be set to 1. In the SysTick_Handler, we check if this value is set, and if it is, we immediately branch out of the handler to prevent any further detection of user inputs. Once the elevator reaches the bottom floor, *admin* is reset to 0, and the SysTick_Handler can resume scanning for inputs every 60 milliseconds.

The three problems above were not the only obstacles of this project, but they were definitely the most significant ones. On top of these, there were a number of endless loop cases and errors, but that comes with any project. We found the microcontroller's built-in debugger to be very useful in stepping through our program to see where issues might have arisen. At the end of the day, we are very happy with the results of our efforts.

ELEVATOR SCHEMATIC AND CIRCUIT:

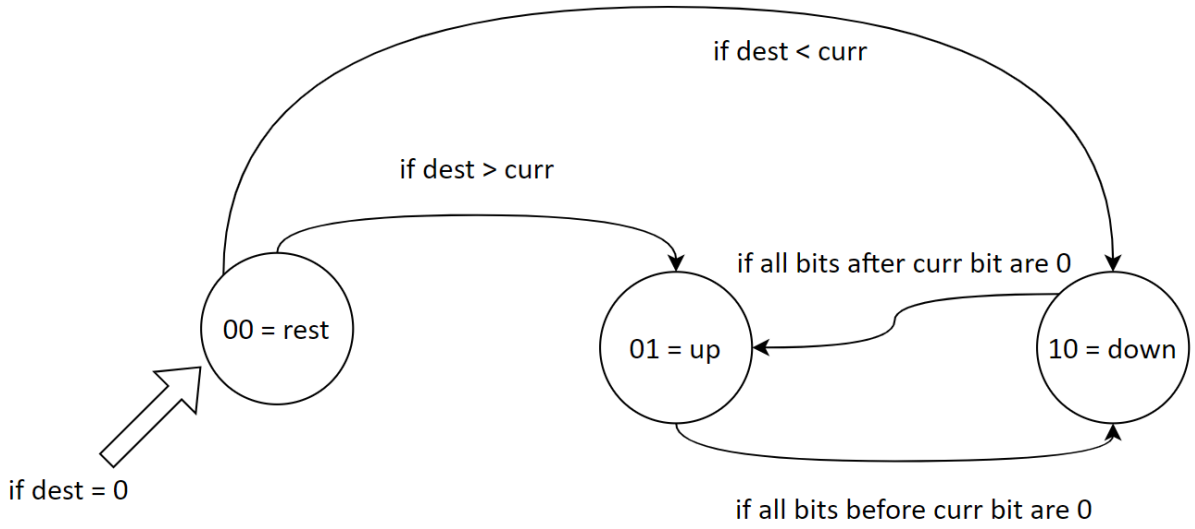




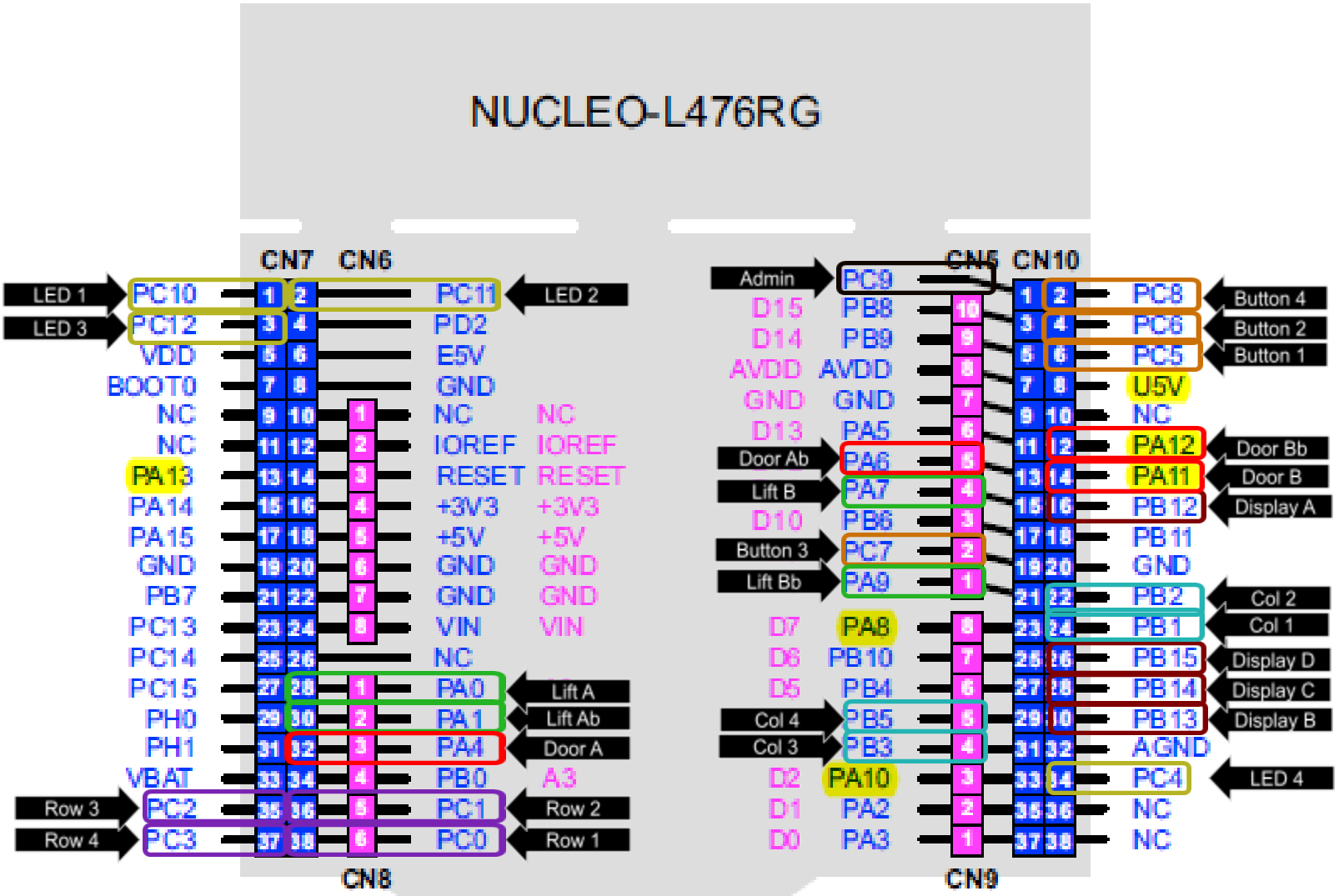
ELEVATOR ALGORITHM PSEUDOCODE:

```
while(true)
    if(dest == 0)
        direction = rest
    else if(dest != 0)                  //find direction
        if(direction == rest)
            if(dest > curr)
                direction = up
            else if(dest < curr)
                direction = down
            else if(dest == curr)
                Display(r0 str*, r1 bytes)
                Door_Clockwise()
                Delay(r0 delay)
                Door_Counterclockwise()
                BIC dest, curr
                continue                //loop

        if(direction == up)             //going up
            if(dest < curr)             //no more upward
                direction = down
            else
                Lift_Clockwise()
                LSL curr, #1

                if(curr && dest == curr)
                    Display(r0 str*, r1 bytes)
                    Door_Clockwise()
                    Delay(r0 delay)
                    Door_Counterclockwise()

                Display(r0 str*, r1 bytes)
                BIC dest, curr
                continue                //loop

        else if(direction == down)      //going down
            if(destReversed <= curr)    //no more downward
                direction = up
            else
                Lift_Counterclockwise()
                LSR curr, #1

                if(curr && dest == curr)
                    Display(r0 str*, r1 bytes)
                    Door_Clockwise()
                    Delay(r0 delay)
                    Door_Counterclockwise()

                Display(r0 str*, r1 bytes)
                BIC dest, curr
                continue                //loop
```

SYSTEM DIAGRAM:



HARDWARE SETUP:



RED = Door Motor **CYAN = Keypad Columns** **ORANGE = Call Buttons**
GREEN = Lift Motor **PURPLE = Keypad Rows** **BLACK = Admin Button**
YELLOW = LEDs **BURGUNDY = 7-Segment Display**