Theme Report 2 - Reason

COMPENG 2DX3
Dr. Yaser Haddara

March 16th, 2025
Edison He (400449140)

Theme

The theme of this report is the ability of a system to reason and make decisions based on data it receives. In labs 4 to 6, the microcontroller gathers inputs from its various sensors and makes decisions based on its logic. Reasoning is crucial in a variety of applications such as autonomous vehicles. Tesla's self-driving cars analyze data from cameras and sensors to make decisions in real time, to move, break, or turn. Labs 4 to 6 explore how information is processed to make specific output decisions. This ability to reason is foundational in systems where dynamic decisions are made to increase efficiency and where intelligent choices are needed.

Background

For autonomous systems to be effective, they must be able to observe their surroundings. For autonomous systems, like self-driving cars, to function safely and effectively, real-time data gathered from their environment is essential. These systems use a variety of sensors, cameras, and radar to continuously monitor their surroundings. This enables them to observe important information including traffic, road conditions, people, and obstacles. The system can then make the correct decisions in real time, which optimizes guarantee safety and flexibility. A Tesla can automatically collect information to determine the vehicle's location on the road, change speed, and avoid obstructions without the need for human intervention.

Reasoning is found in lab 4 and 6 from how the microcontroller processes inputs and makes decisions based on timing, control signals, and system feedback. In Lab 4, the microcontroller is tasked with interpreting data (such as r the stepper motor’s control signals) and then making logical decisions to generate specific outputs. It applies specific binary sequences to the motor's control pins in the correct order to achieve precise rotation. Additionally, it must wait a specific delay between steps to control the motor’s speed. Lab 6 involves the microcontroller making decisions based on button inputs to control the stepper motor and LEDs. When a button is pressed, the system must determine whether to start/stop the motor, reverse its direction, adjust the LED blink, or return it to the home position. It then updates the LEDs to reflect these changes. This decision-making process is key to embedded systems, where inputs are processed in real-time to control outputs effectively.

Theme Exemplars

For this report, Labs 4 and 6 will be used as theme examples. Lab 4 serves as a good introductory lab on the reasoning of the stepper motor and PWM concerts by the microprocessor. This topic is further expanded on in Lab 6 where button inputs are introduced to build on the stepper motor reasoning

Lab 4

Lab 4 included 2 milestones which used reasoning with timings of duty cycles, and switching phases with delays. The objective of the milestone was to create a varying duty cycle

and verify the change with an increase and decrease in the brightness of the LED. The microprocessor would reason based on variables to increase and decrease the duty cycle, which was outputted to a LED, creating the flashing effect. The varying brightnesses can be seen in Figure 1 below

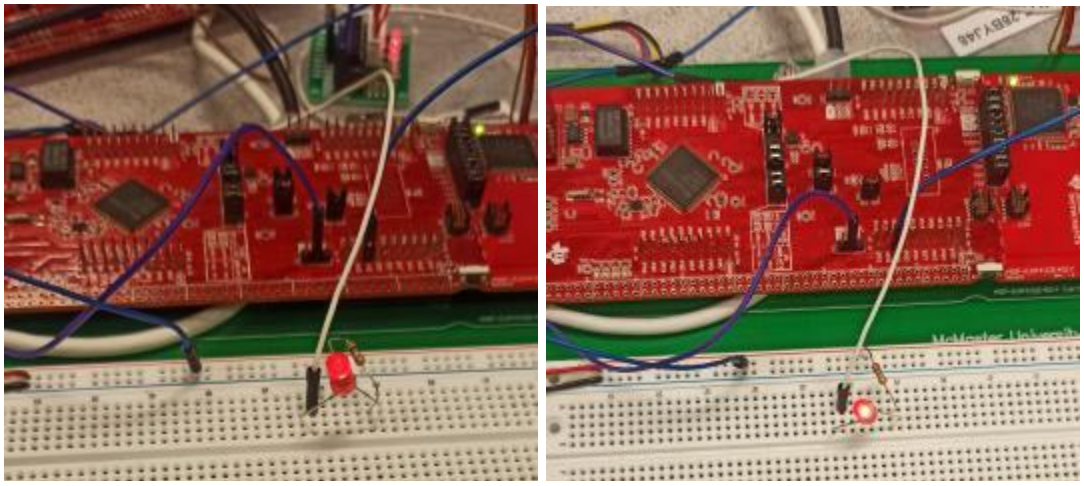
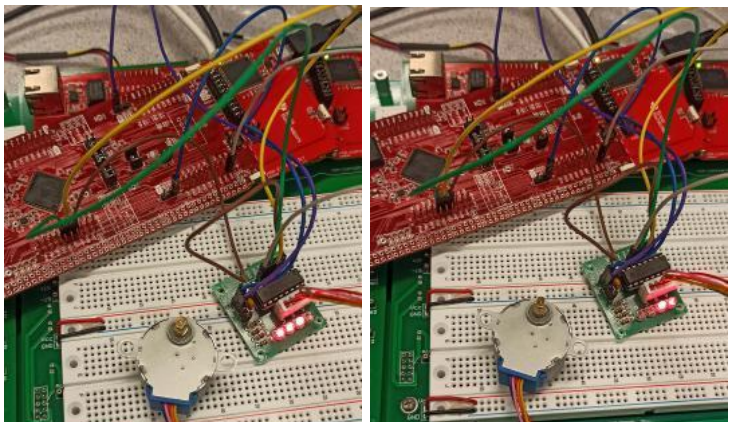


Figure 1: Varying LED Brightness

Milestone 2 further applied this concept to a stepper motor. The stepper motor is more complex than a DC motor to control. It involves poles which must be turned on and off to create magnetic attraction, which are sent from port M to the driver board which rotates the stepper motor. An image of the circuit can be seen in Figure 2. In order to rotate the motor different "phases" must be completed in sequential order. After 2048 incremental steps, the motor completes a full rotation. In this case, the microprocessor dynamically reasons based on the current state to move onto the next state. The alternating LEDs are a visual representation and verification of the state changes. In addition, depending on the observed data, the microprocessor must reason and output phases in specific order to rotate in both directions.



```
void spin(uint32_t direction, uint32_t s
  uint32_t delay = 1;          // Does yo

  if(direction == 0){ //CCW
    for(int i=0; i< step; i++){       //
      GPIO_PORTM_DATA_R = 0b00001001;
      SysTick_Wait10ms(delay);       // W
      GPIO_PORTM_DATA_R = 0b00001100;
      SysTick_Wait10ms(delay);
      GPIO_PORTM_DATA_R = 0b00000110;
      SysTick_Wait10ms(delay);
      GPIO_PORTM_DATA_R = 0b00000011;
      SysTick_Wait10ms(delay);
  }
}
  if(direction == 1){ //CW
    for(int i=0; i< step; i++){       //
      GPIO_PORTM_DATA_R = 0b00000011;
      SysTick_Wait10ms(delay);       // W
      GPIO_PORTM_DATA_R = 0b00000110;
      SysTick_Wait10ms(delay);
      GPIO_PORTM_DATA_R = 0b00001100;
      SysTick_Wait10ms(delay);
      GPIO_PORTM_DATA_R = 0b00001001;
      SysTick_Wait10ms(delay);
```

Figure 2: Stepper Motor Circuits and Code

Lab 6

This lab is an early integration of project 1, demonstrating basic functionality of a system using a microcontroller. It uses a stepper motor on port H, push buttons, and LED outputs for status. Push button 0 acts as a start and stop button and button 1 switches direction. These buttons are found on the microprocessor itself on port J. On the other hand, button 3 acts as a toggle for LED blinks per 11.25/45 degrees and button 4 returns the motor to home. These push buttons are active low push buttons built on a breadboard, which is sent to port M.

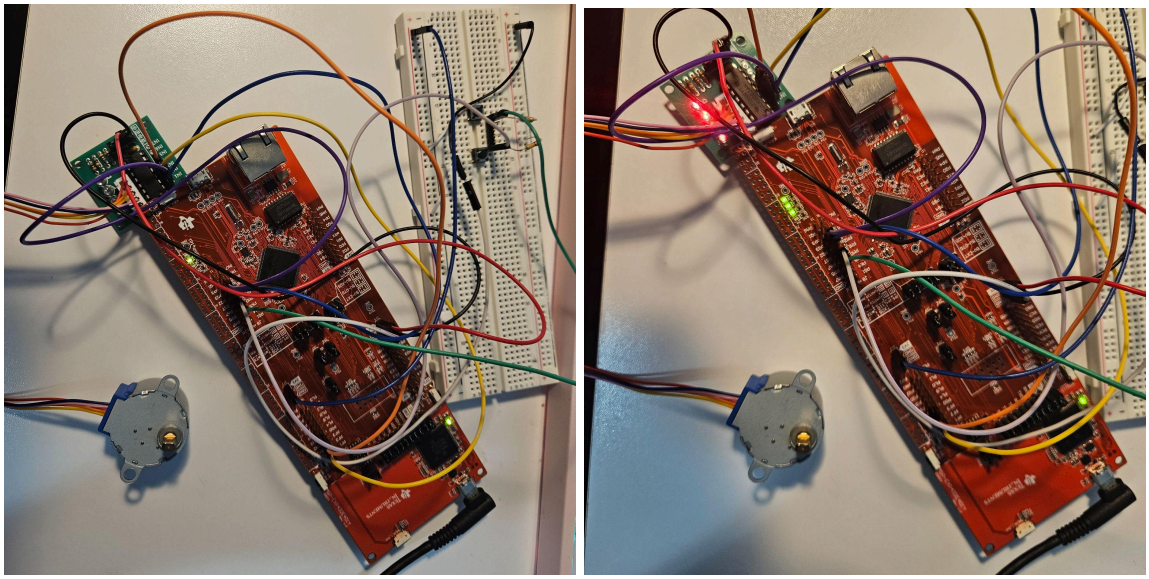


Figure 3: Idle and Motor Movement

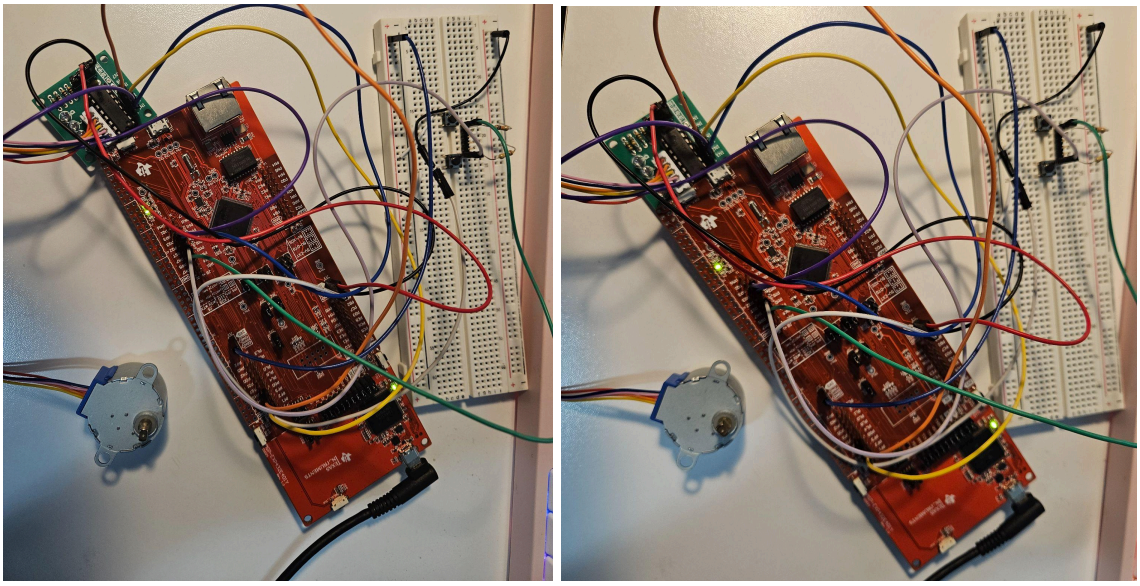


Figure 4: Direction and Angle Blink Toggle

To ensure the microprocessor responds to all inputs, polling was introduced to the code, seen in Figure 5 below. The numerous if statements within the while loop continuously reads inputs. If at any time it detects an input, the microprocessor detects which button was pressed and outputs the correct instruction.

```
119    while(1) {
120        if(GPIO_PORTJ_DATA_R == 0x02 && direction == -1) {
121          direction = 1;
122          GPIO_PORTN_DATA_R|= 0x01;
123        while(GPIO_PORTJ_DATA_R == 0x02);
124        }
125        if(GPIO_PORTJ_DATA_R == 0x02 && direction == 1) {
126          direction = -1;
127          GPIO_PORTN_DATA_R &= ~0x01;
128          while(GPIO_PORTJ_DATA_R == 0x02);
129        }
130        if(GPIO_PORTJ_DATA_R == 0x01) {
131            while(GPIO_PORTJ_DATA_R == 0x01) {}
132            GPIO_PORTN_DATA_R |= 0x02;
133            state = 1;
134        }
135
136        while(state == 1) {
137            if(GPIO_PORTJ_DATA_R == 0x01) {
138                while(GPIO_PORTJ_DATA_R) {}
139                GPIO_PORTN_DATA_R |= 0x02;
140                state = 0;
141                break;
142            }
143            if(GPIO_PORTJ_DATA_R == 0x02 && direction == -1) {
144            direction = 1;
145            GPIO_PORTN_DATA_R|= 0x01;
146            while(GPIO_PORTJ_DATA_R == 0x02);
147            }
148            if(GPIO_PORTJ_DATA_R == 0x02 && direction == 1) {
149              direction = -1;
150              GPIO_PORTN_DATA_R &= ~0x01;
151              while(GPIO_PORTJ_DATA_R == 0x02);
152          }
153           if(GPIO_PORTM_DATA_R == 0x01 && angle == 1) {
154          GPIO_PORTF_DATA_R |= 0x10;
155          angle = 0;
156          step = 16;
157          while(GPIO_PORTM_DATA_R == 0x01);
158          }
159          if(GPIO_PORTM_DATA_R == 0x01 && angle == 0) {
160              GPIO_PORTF_DATA_R &= ~0x10;
161              angle = 1;
162              step = 64;
163              while(GPIO_PORTM_DATA_R == 0x01);
164          }
165          position = (position+spin(direction))%512;
166          if(position < 0)
167          {
168            position = 512;
169          }
170          if(GPIO_PORTM_DATA_R == 0x02) {
171            returnHome(position);
172            position = 0;
173            state = 0;
174          }
175          if(position%step == 0)
176          {
177            GPIO_PORTF_DATA_R |= 0x01;
178          }
179          if(position%step != 0)
180          {
181            GPIO_PORTF_DATA_R &= ~0x01;
182          }
```

Figure 5: Polling

Debugging

LED debugging strategies were seen primarily in Lab 6. To ensure that button presses are being read in addition to state changes, LEDs were used. For example, while first implementing the push buttons, a button debounce was not implemented. On the LED side, the LED needed to toggle every button press. However, the LEDs were flashing on for a second before turning off. This meant that there was a software issue and that states were not being transitioned to. Adding the Debounce fixed this issue. Another case of debugging with LEDs was used on the driverboard of the stepper motor. When setting up for the demonstration, the reverse direction was faulty. Checking the onboard LEDs revealed that the button press was correctly being registered. Checking the driver board LED revealed that PH0 was not connected. An example can be seen below in Figure 6, where in both pictures, the onboard LEDs are working, but with each rotation, the top driver board LED does not turn on.

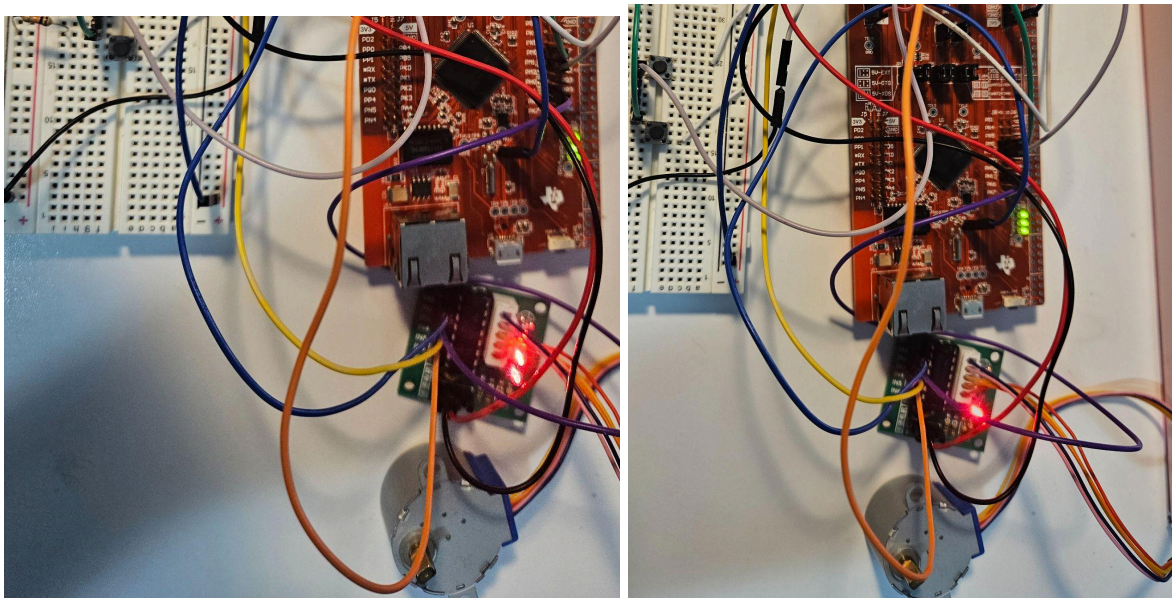


Figure 6: LED Debugging

Synthesis

Labs 4 and 6 effectively illustrate the theme of reasoning and adaptation in systems as the microprocessor alters its outputs based on inputs and internal states. Lab 4 is an introductory step by demonstrating how the microprocessor adjusts the duty cycle to control LED brightness which creates a flashing effect. This concept is expanded in Milestone 2 where the microprocessor transitions between various phases to rotate based on state and direction. Lab 6 further expands on this concept by integrating multiple components. This includes push buttons, LEDs, and a stepper motor. It requires the microprocessor to monitor button presses at all times and respond accordingly. Through polling, the system continuously reads inputs, ensuring low latency and high responsiveness. Debugging with LEDs further reinforces the importance of observation, as seen when addressing debounce issues and verifying motor driver connections. This concept mirrors adaptive cruise control in modern vehicles, where the system continuously observes the distance to the car ahead and adjusts acceleration or braking to maintain safe spacing. This demonstrates real-time reasoning and response to environmental changes.

Reflection

Labs 4 and 6 provided valuable insight into how microprocessors reason and adapt based on inputs and states.. While initially, it seemed straightforward for a microcontroller to read inputs and produce outputs, the process requires precise control over timing, signal sequencing, and state transitions. This was evident in Lab 4, where the microcontroller had to determine the correct phase sequence to drive the stepper motor efficiently.

Another realization was the necessity of continuous monitoring in systems. Lab 6 covered this concept with push buttons, LEDs, and a stepper motor, requiring the microprocessor

to actively poll inputs and with minimal latency. This mimics real-world applications, such as autonomous vehicles and adaptive cruise control, where real-time data processing is critical for safety and functionality.

Additionally, debugging with LEDs in Lab 6 highlighted the importance of a system to observe. Before implementing a debounce, the LED behaved inconsistently, revealing a problem with state transitions. Similarly, checking the driver board LEDs helped diagnose the faulty motor direction. This reveals that minor hardware issues can severely disrupt the system's logic. I learned that reasoning in engineering is not just about making decisions but also about verifying those decisions through observation and vigorous testing.

Overall, these labs deepened my understanding of reasoning in embedded systems, showcasing how microcontrollers process inputs, reason, and adapt to different conditions. They emphasize decision-making, dynamic monitoring, and debugging to enable efficient and reliable systems.