Theme Report 2: Reason

Diego Verdin
400504430
16 March 2025

Theme

The second theme in the course is reasoning. In the context of intelligent systems, this refers to the ability to process acquired data, analyze it, and make decisions based on logical operations. This is fundamental in intelligent systems where microcontrollers execute programmed instructions to respond to input conditions. Reasoning is important for developing good, adaptive and efficient control mechanisms which react to real world conditions.

Background

Reasoning in intelligent systems extends beyond observation and involves decision making. This requires mechanisms such as FSMs (Finite State Machines) and peripheral interfacing among others.

Lab 4 introduced PWM (pulse width modulation) control, demonstrating how varying duty cycles can regulate power output affecting devices like LEDs and motors. In Lab 5, peripheral interfacing was explored to observe how a program can reason based on the input on an outside signal such as a button or a keypad. Lastly in Lab 6, the first project deliverable, emphasized the importance of combining multiple subsystems to create a coherent solution that was able to reason based on outside input and convert that into a real world implementation.

Theme Exemplars

Exemplar 1: Project Deliverable 1 (Lab 6)

Project Deliverable 1 put together everything in this theme of the course, therefore it was chosen as the main exemplar since it showcases both reasoning to control an output signal like lab 4 did, reasoning to interpret an input like lab 5 did, and it puts both together to control an output based on the input.

Method:

- Wire 4 inputs, two on board buttons, two out of board buttons.
- Wire an output to a stepper motor.
- Create a function to accurately interpret the button presses.
- Create a function to control the output to the stepper motor.
- Create a function that implements the logic required depending on each button press.
- Validate functionality through real time testing and debugging.

Validation:

- The built circuit, shown below, was validated by making sure that every button performed its function properly. This was tested physically and by parts.
- First, test that buttons are being properly interpreted by just turning on an LED.
- Second, test that motor function works properly by calling it with arbitrary test inputs.
- Lastly, test that each button does what it was supposed to do in the context of the milestone.

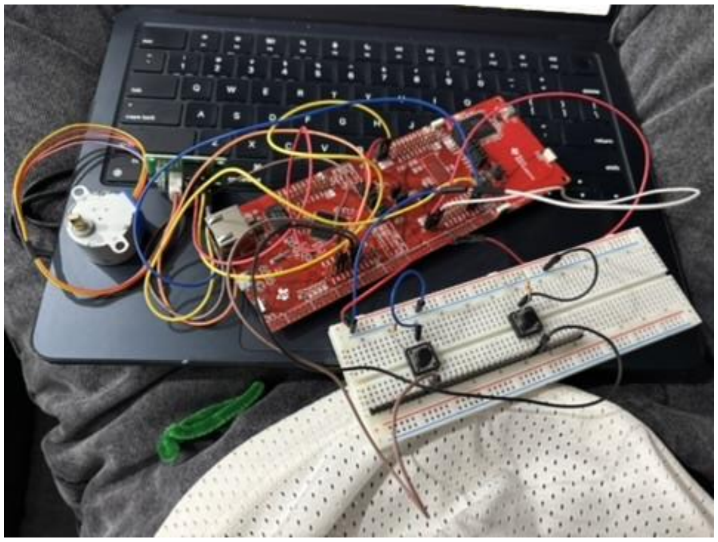


Figure 1 Built Circuit from Lab 6

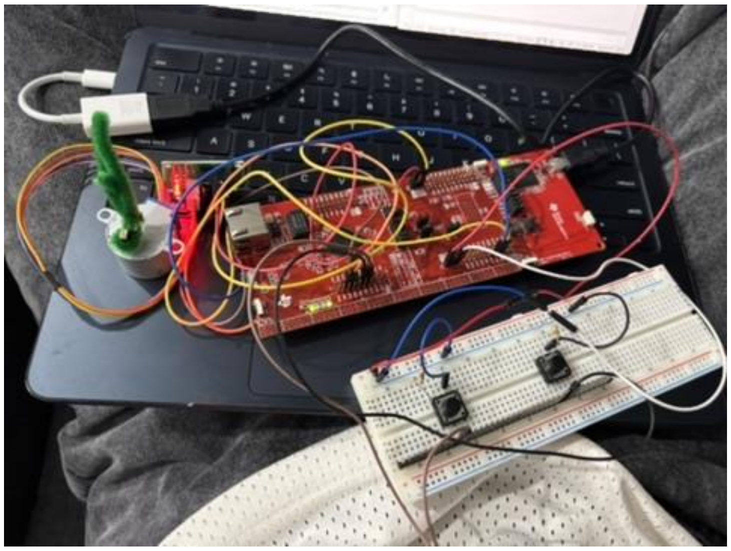


Figure 2 Built Circuit from lab 6 running

Debugging Exemplar

A significant debugging challenge during this set of labs came during Lab 5. The issue arose due to the motor not spinning causing an error on the desired operation.

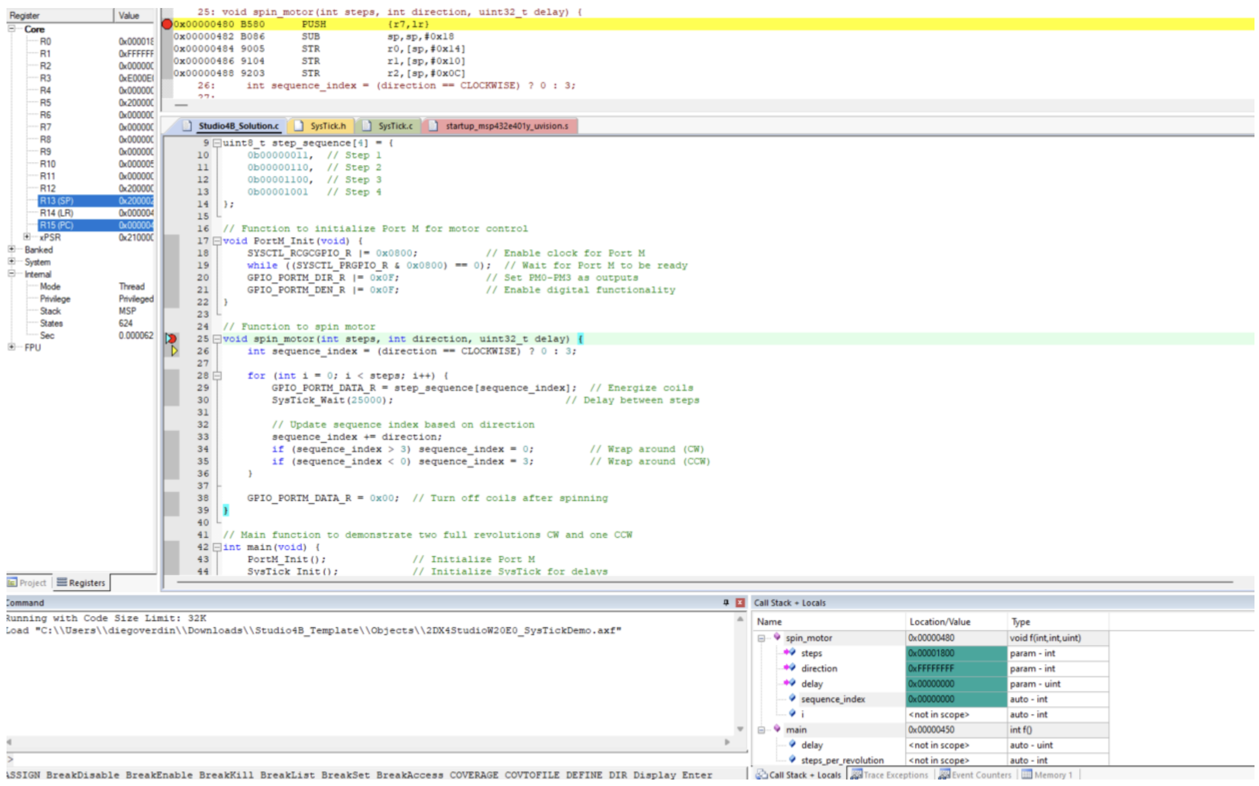
Debugging Method:

- Implanted a variable to hold the values of each of the specific pins that were outputting to the motor, this was in the form of an array.
- Used break point debugging in Keil MDK to monitor said variable's values and program flow, looking for where the code resulted in incorrect outputs.
- Introduced watchdog timers to reset the system
- Ran long duration tests.

Code with watchdog timers

```
#include <stdint.h>
#include "tm4c1294ncpdt.h"
#include "SysTick.h" // Include appropriate header for SysTick

#define CLOCKWISE 1
#define COUNTER_CLOCKWISE -1

// Full-step sequence patterns
uint8_t step_sequence[4] = {
   0b00000011,  // Step 1
   0b00000110,  // Step 2
   0b00001100,  // Step 3
   0b00001001   // Step 4
};

// Function to initialize Port M for motor control
void PortM_Init(void) {
   SYSCTL_RCGCGPIO_R |= 0x0800;           // Enable clock for Port M
   while ((SYSCTL_PRGPIO_R & 0x0800) == 0); // Wait for Port M to be ready
   GPIO_PORTM_DIR_R |= 0x0F;              // Set PM0-PM3 as outputs
   GPIO_PORTM_DEN_R |= 0x0F;              // Enable digital functionality
}

// Function to initialize Watchdog Timer
void WDT_Init(void) {
   SYSCTL_RCGCWD_R |= SYSCTL_RCGCWD_R0;   // Enable Watchdog Timer 0
   while (!(SYSCTL_PRWD_R & SYSCTL_PRWD_R0)); // Wait for WDT to be ready

   WATCHDOG0_LOAD_R = 16000000; // Set timeout (e.g., 1 sec assuming 16MHz
clock)
   WATCHDOG0_CTL_R = 0x03; // Enable reset mode and watchdog timer
}

// Function to reset Watchdog Timer
void WDT_Reset(void) {
   WATCHDOG0_ICR_R = 0xFFFFFFFF; // Clear timer count
}

// Function to spin motor with WDT monitoring
```

```
void spin_motor(int steps, int direction, uint32_t delay) {
    int sequence_index = (direction == CLOCKWISE) ? 0 : 3;

    for (int i = 0; i < steps; i++) {
        // Check if sequence_index is valid
        if (sequence_index < 0 || sequence_index > 3) {
            // Out-of-bounds condition detected, do not reset WDT (forces system reset)
            while (1); // Intentional infinite loop to trigger watchdog reset
        }

        // Reset watchdog timer since sequence_index is valid
        WDT_Reset();

        // Energize coils
        GPIO_PORTM_DATA_R = step_sequence[sequence_index];
        SysTick_Wait(25000); // Delay between steps

        // Update sequence index based on direction
        sequence_index += direction;
        if (sequence_index > 3) sequence_index = 0; // Wrap around (CW)
        if (sequence_index < 0) sequence_index = 3; // Wrap around (CCW)
    }

    GPIO_PORTM_DATA_R = 0x00; // Turn off coils after spinning
}

// Main function to demonstrate two full revolutions CW and one CCW
int main(void) {
    PortM_Init();            // Initialize Port M
    SysTick_Init();          // Initialize SysTick for delays
    WDT_Init();              // Initialize Watchdog Timer

    uint32_t delay = 0;      // Adjust delay as needed
    int steps_per_revolution = 2048;

    // Two full revolutions clockwise (2 * 2048 = 4096 steps)
    // spin_motor(2 * steps_per_revolution, CLOCKWISE, delay);

    // Brief pause between directions
    // SysTick_Wait10ms(50);

    // Three full revolutions counter-clockwise (3 * 2048 = 6144 steps)
    spin_motor(3 * steps_per_revolution, COUNTER_CLOCKWISE, delay);

    while (1);  // End of program
-   }
```



Register	Value
Core	
R0	0x00001ℇ
R1	0xFFFFFF
R2	0x00000C
R3	0xE000E(
R4	0x00000C
R5	0x20000C
R6	0x00000C
R7	0x00000C
R8	0x00000C
R9	0x00000C
R10	0x000005
R11	0x00000C
R12	0x20000C
R13 (SP)	0x200002
R14 (LR)	0x000004
R15 (PC)	0x000004
xPSR	0x21000C
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	624
Sec	0.000062
FPU	

```
    25: void spin_motor(int steps, int direction, uint32_t delay) {
0x00000480 B580      PUSH          {r7,lr}
0x00000482 B086      SUB           sp,sp,#0x18
0x00000484 9005      STR           r0,[sp,#0x14]
0x00000486 9104      STR           r1,[sp,#0x10]
0x00000488 9203      STR           r2,[sp,#0x0C]
    26:     int sequence_index = (direction == CLOCKWISE) ? 0 : 3;
```

Studio4B_Solution.c | SysTick.h | SysTick.c | startup_msp432e401y_uvision.s

```c
uint8_t step_sequence[4] = {
    0b00000011,  // Step 1
    0b00000110,  // Step 2
    0b00001100,  // Step 3
    0b00001001   // Step 4
};

// Function to initialize Port M for motor control
void PortM_Init(void) {
    SYSCTL_RCGCGPIO_R |= 0x0800;             // Enable clock for Port M
    while ((SYSCTL_PRGPIO_R & 0x0800) == 0);  // Wait for Port M to be ready
    GPIO_PORTM_DIR_R |= 0x0F;                // Set PM0-PM3 as outputs
    GPIO_PORTM_DEN_R |= 0x0F;                // Enable digital functionality
}

// Function to spin motor
void spin_motor(int steps, int direction, uint32_t delay) {
    int sequence_index = (direction == CLOCKWISE) ? 0 : 3;

    for (int i = 0; i < steps; i++) {
        GPIO_PORTM_DATA_R = step_sequence[sequence_index];  // Energize coils
        SysTick_Wait(25000);                               // Delay between steps

        // Update sequence index based on direction
        sequence_index += direction;
        if (sequence_index > 3) sequence_index = 0;         // Wrap around (CW)
        if (sequence_index < 0) sequence_index = 3;         // Wrap around (CCW)
    }

    GPIO_PORTM_DATA_R = 0x00;  // Turn off coils after spinning
}

// Main function to demonstrate two full revolutions CW and one CCW
int main(void) {
    PortM_Init();              // Initialize Port M
    SysTick_Init();            // Initialize SysTick for delays
```

```
Running with Code Size Limit: 32K
Load "C:\\Users\\diegoverdin\\Downloads\\Studio4B_Template\\Objects\\2DX4StudioW20E0_SysTickDemo.axf"

ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE DEFINE DIR Display Enter
```

Call Stack + Locals

Name	Location/Value	Type
spin_motor	0x00000480	void f(int,int,uint)
steps	0x00001800	param - int
direction	0xFFFFFFFF	param - int
delay	0x00000000	param - uint
sequence_index	0x00000000	auto - int
i	<not in scope>	auto - int
main	0x00000450	int f()
delay	<not in scope>	auto - uint
steps_per_revolution	<not in scope>	auto - int

Figure 3 Keil MDK debugger with breakpoints as well as a watch variable

Synthesis

The exemplar shows how reasoning enables intelligent systems to process input data to produce meaningful outputs. Lab 6 emphasized structured decision making by requiring the integration of various hardware and software components. The program that was written was able to reason when given a series of inputs, the reasoning then allowed it to produce outputs such as turning on LEDs and turning a motor. The debugging process shows the importance of methodical troubleshooting and being able to watch variables is important to make sure the logic with which the program reasons is sound.

Reflection

These labs allowed me to gain a deeper understanding of how intelligent systems reasons by processing inputs and making decisions. The practical challenges, particularly in debugging allowed me to reinforce the importance of structured and planned out problem solving. The ability to control peripherals and using their inputs in computer logic is essential for intelligent systems as well as for the design of my final project. These labs have allowed to become comfortable with these concepts and will be of great importance as I make more progress with the design of my final project.