Theme Report 1: Observe

COMEPNG 2DX3: Microprocessor Systems Project

Sunday, February 16th, 2025

Joey McIntyre (400520473)

Theme

The observe theme when it comes to intelligent systems is the system's ability to gather and perceive information from its environment using digital and analog inputs. This is a central theme of intelligent systems as it is fundamental to the decision-making process. Without observation, a system can not understand or react to its surrounding environment [1].

For example, receiving a digital input through the press of a button or flick of a switch and processing sequential inputs using a Finite State Machine (FSM) are both standard observation methods. These concepts were covered in Lab 1 (Digital Signals), Lab 2 (Finite State Machines), and Lab 3 (Analog Signals), which were largely focused on digital input processing, interpreting state-based logic, and analog signal acquisition.

Background

Observation is the fundamental capability of intelligent systems to perceive and acquire data from their environment using digital and analog inputs. Without the ability to observe, systems would be unable to operate under unpredictable real-world conditions. They must be able to recognize changes in the environment and make instructed decisions. Observation is critical to acquire real-time data, which allows microcontrollers to process inputs and react accordingly. Ineffective observation would take away an intelligent system's ability to respond to changes in its environment, detect errors, and process data for decision-making.

In intelligent electronic systems, observation occurs through sensors, buttons, levers, cameras and many other input devices. There are numerous examples of technologies that rely on accurate and efficient real-time observations. This includes autonomous cars, which rely on sensors and cameras to observe their surroundings and react to the ever-changing traffic. Another example would be smart home systems, which use motion and temperature sensors to make houses more comfortable and safer. A final example of intelligent systems that use observation in the real world is smart assistants such as Google Nest and Amazon Alexa, which accept and process voice commands and respond in a human-like fashion while performing the requested tasks [2]. All these examples of revolutionary technologies would not be possible without observation.

This course uses the MSP432E401Y microcontroller, which observes inputs through digital and analog interfaces. The GPIO pins detect digital inputs through on/off signals, such as the press of a button or flick of a switch. Finite State Machines allow structured observation by tracking sequential inputs and reacting with the correct output. Analog to digital converters take real-world signals and convert them into a format our microcontroller can process. Labs 1-3 focus on implementing these methods to help us understand how embedded systems observe and interpret data before making decisions.

Theme Exemplars

One example that demonstrates how intelligent systems observe data was Lab 1 Milestone 2. In this lab, we were tasked with configuring Port M of the General-Purpose Input/Output (GPIO to detect the press of a button and control an LED. This demonstrated the observation of a digital input on an elementary level. Digital inputs are one of the fundamental ways in which an embedded system observes its environment. These binary signals can be HIGH (1) or LOW (0).

The method by which this milestone was completed started with configuring our ports. We configured the GPIO Port M, Bit 0 (PM0), to be an input that detects the press of a button. To do this, we had to enable the clock for the GPIO peripheral and set the pin as an input in the Data Direction Register (DIR). Next, we used a pull-up resistor to stabilize the HIGH signal when the button is unpressed. This ensures that when the button is unpressed, the input reads HIGH (1), and when it is pressed, the input reads LOW (0).

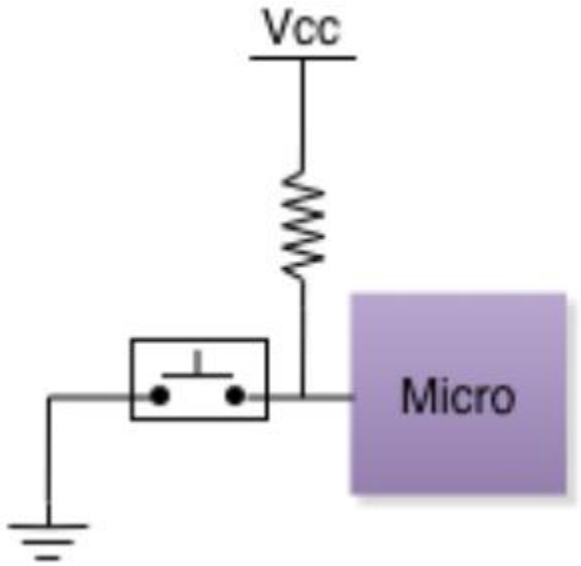

Figure 1: Correct Pull-Up Resistor Wiring (Figure 3.2 in Lab Manual)

The next step was writing the assembly code to read the GPIO MP0 input register continuously. If the LOW state was detected, indicating the button was pressed, we turned on an LED connected to port N. The LED was turned off if the button was not pressed, meaning the register was at a HIGH state.

To validate the results of this milestone, we demonstrated the successful toggling of the LED. It turned on when the button was pressed and off when it was released. We also used the Kiel built-in debugger to view the register and verify that when the button was pressed, the PM0 bit read a value of 0, and when it was unpressed, the PMO bit read a value of 1. Lab 1 involved processing digital inputs, which allowed us to learn how microcontrollers observe binary signals. This is an essential function of intelligent systems.

Another example of how intelligent systems observe data came in Lab 2 Milestone 2. As discussed in the previous example, digital inputs provide a simple HIGH or LOW observation; however, more complex systems need to be able to track a sequence of inputs. In Lab 2, Finite State Machines (FSM) were used to observe how intelligent systems accept sequential inputs.

The first step of this lab was to design a combinational digital lock. This was accomplished by configuring PM0-PM2 as digital inputs representing each digit of the three-bit code. PM3 was configured as a load button to enter the input. We then coded the microcontroller to monitor PM0-PM2 continuously. LED D1 being on represented that an incorrect code was entered, but if the correct code (100) is entered, the LED D2 is turned on, and D1 is turned off. The next step involved using a FSM to implement a sequential lock. Instead of evaluating the entire combination simultaneously, the FSM observes and processes the inputs sequentially.

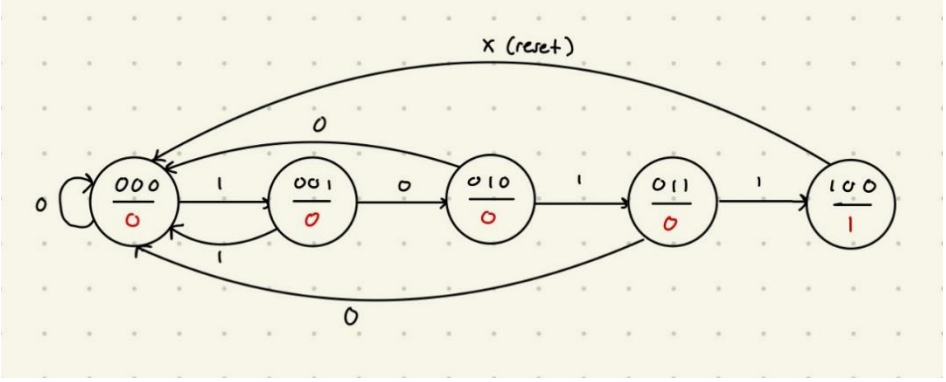

Figure 2: FSM Diagram for the 4-digit Sequential Lock

As demonstrated in Figure 2, the FSM updates each time the load button is pressed. Using assembly code, each time the load button was pressed, the state of the binary value button was checked and noted. LED D1 was set on if the incorrect code was entered, and LED D2 was turned on and D1 off if the correct sequence was entered at any point.

Current State	Input	Next State	Output
0 0 0	0	0 0 0	0
0 0 0	1	0 0 1	0
0 0 1	0	0 1 0	0
0 0 1	1	0 0 0	0
0 1 0	0	0 0 0	0
0 1 0	1	0 1 1	0
0 1 1	0	0 0 0	0
0 1 1	1	1 0 0	0
1 0 0	X	0 0 0	1

Figure 3: State Table of the Sequential Lock

Figure 3 demonstrates the desired functionality of the sequential lock. The output is LOW whenever the incorrect code is entered and is only changed to HIGH when the correct sequence of inputs was entered.

LEDs were used to observe the FSM transitions as the binary sequence was entered to validate our results. This was helpful in determining where the mistakes were happening if we weren't seeing the desired result. The Keil debugger tool was also used to observe the state register values and confirm that the sequence was appropriately tracked. The final validation was the FSM correctly unlocking only when the correct input sequence was entered, which was checked by TAs. By implementing FSM-based observation in Lab 2, we learned how intelligent systems observe input sequences, allowing for complex decision-making beyond processing simple digital inputs like in Lab 1.

Debugging Exemplar

An example of when we needed to use a debugging technique was in Lab 1. Initially, the GPIO digital input from PM0 did not turn on the LED. It remained off regardless of whether the button was pressed or not. To figure out what the problem was, we used the built-in Keil debugger. More specifically, we set a breakpoint in our code in the loop that read the GPIO input. We then used the debugger register view to step through the code and examine the GPIO data register. We observed that PM0 always read LOW, even when the button was not pressed. This confirms the issue was caused by misreading the input. To fix this bug, I checked the circuit and realized that the pull-up resistor was implemented incorrectly, as the wrong example was used from Figure 3.2 in the lab manual [3]. After correcting this mistake, the LED turned on when the button was pressed, as intended. We used the same debugging steps to ensure this bug was fixed entirely and confirmed that PM0 was reading as HIGH when unpressed and LOW when pressed. The Keil debugger is a great tool; a good understanding of how it works will certainly be essential to success in this course.

Synthesis

For intelligent systems to comprehend and respond to their surroundings, the observe theme (which we thoroughly examined in these first three labs) is the essential first step. We can gain a better understanding of how embedded systems process the data they observe by examining the two topic exemplars previously mentioned from Labs 1 and 2.

Some key takeaways from the exemplars include that digital inputs are the basis for all observations. Lab 1 demonstrated how our microcontroller captures basic digital signals (the binary state of a button) and how configuring the GPIO is crucial in ensuring accurate data collection. Another key takeaway is how finite state machines observe structured, sequential inputs. Lab 2 covered this concept by observing how the microcontroller observes sequences of inputs over time. Instead of responding to a signal immediately, the system processed and stored the inputs until the correct code was entered and the LED turned on. The Labs covering the

observe theme also stressed the importance of debugging to ensure reliable observations. The Keil debugger was used to detect, isolate and solve issues in all three labs. When trying to understand how an intelligent system observes data, it is crucial to be able to check the register values in the microprocessor at different points and understand what they should be and how the system should react. By combining digital input sensing, sequential logic tracking, and debugging techniques, intelligent systems can consistently observe their environment and determine what action to perform.

Reflection

This sequence of labs focusing on the observation theme of intelligent systems significantly improved my understanding of data observation in our microcontroller. Each lab helped me understand a different aspect of the observe theme. Lab 1 taught me how microcontrollers observe simple binary inputs, and Lab 2 demonstrated structural sequential observation using finite state machines, and it deepened my understanding of how intelligent systems track inputs over time. On top of this, the Keil debugger helped me visualize GPIO data using the register view, which made debugging issues a much easier task, all while improving my understanding of how to effectively use the microcontroller. A strong knowledge of the observe theme is going to be essential for success in this course because future labs and final projects will cover two new themes that heavily rely on this understanding. By mastering structured observation and debugging techniques, I am better prepared to work on the more complex intelligent systems that will be covered in this course.

Reference

[1] “What Are Intelligent Systems | Computer Science & Engineering,” *University of Nevada, Reno*. https://www.unr.edu/cse/undergraduates/prospective-students/what-are-intelligent-systems

[2] “Intelligent Systems: What are they, how do they work and why are they so important,” *www.algotive.ai*. https://www.algotive.ai/blog/intelligent-systems-what-are-they-how-do-they-work-and-why-are-they-so-important

[3] S. Athar, T. Doyle, Y. Haddara “Computer Engineering 2DX3 | 2024-2025 Laboratory Manual” *McMaster University, Hamilton.*