Name – Harshpreet Kaur Nagra

Student ID – 2410994798

SIT315 – Programming Paradigms

M1_T1P-C-D Sense-Think-Act System

REFLECTION REPORT – BRIGHTNESS CONTROL INTERRUPT SYSTEM

I designed an entire interrupt-driven embedded system utilizing an Arduino Uno as the basis of the project. The system implements the Sense-Think-Act model and uses both Pin Change Interrupts (PCI) and a Timer interrupt. The hardware consists of three push button input sensors connected to digital pins 8, 9 & 10, and one LED connected to digital pin 5 that supports PWM operation.

The three push buttons are used as digital input sensors. I configured them in INPUT_PULLUP mode so they will normally be HIGH and will go LOW when pressed. Instead of polling for each button on each loop iteration, I am using Pin Change Interrupts (PCINT0_vect) to determine which button has changed state, which will call the associated interrupt service routine (ISR). In the ISR, my only action is to set the respective flag variable to keep the ISR short and safe. The main loop will check the value of the respective flag and perform any button logic activities.

Each button has a different operation. Button 1 will increase the LED brightness level, Button 2 will decrease the LED brightness and Button 3 will reset the LED brightness to zero. The LED brightness level will be controlled using the analogWrite() function for pin D5 to provide a PWM output signal. The LED brightness level implementation utilizes conditional logic and proper state management utilizing a brightness variable.

I set up an interrupt using Timer1 in CTC mode to fire every second, allowing for one system status update message to print out on the Arduino Serial Monitor, detailing the current brightness level. An interrupt from Timer1 allowed me to separate the event-driven logic (button interrupts), and time-driven logic (timer interrupts).

A challenge I had with this project was learning to determine what pin has changed when using pin change interrupts. Since D8-D13 all use the same interrupt group (PORTB), I had to keep track of the previous and current state of PORTB in order to determine which button I pressed. I also made sure that inside my timer interrupt I'm not using delay() or Serial printing, to ensure I'm following safe practices.

Overall, I felt this project helped me learn how to use interrupts to handle multiple concurrent events as a real-time embedded system. The finished product demonstrates PCI functionality, Timer Interrupts, State Management, and Clean Modular Design.