

Class : 2025-1 Embedded System Design—BEU

Team Number : 11

Student ID : 12244853, 12244858

Name : Elton Satiyev, Ikram Mammadov

Submission Date : 2025-04-17



HOMEWORK1.

Problem

1.

Implement HW and SW of provided scenario

Polling sequence toggles 3 LED by interval of 1sec.

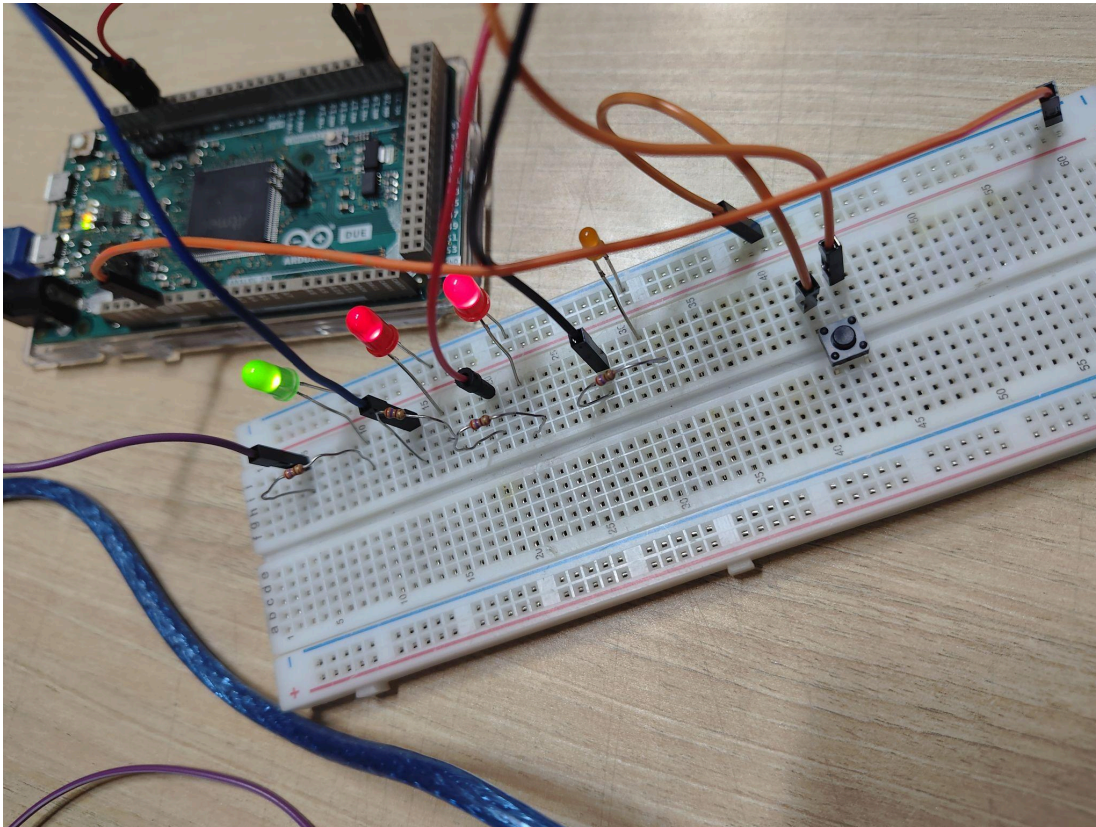
If you push button, the ISR called, then 3 LED stops toggling, another LED turns on for 5sec and ISR ends.

During ISR, print each second value to serial monitor.

You can use Timer/Counter interrupt inside GPIO interrupt

Hint : Use global parameter for purpose of increasing 'count' in Timer/Counter ISR

Circuit



Code

```
50
51 // Team 11: Homework 1
52 #include <DueTimer.h>
53 #define LD0 13
54 #define LD1 12
55 #define LD2 11
56 #define LD_ISR 20
57 #define SW 21
58 volatile bool inISR = false;
59 volatile int count = 0;
60 void setup() {
61     pinMode(LD0, OUTPUT);
62     pinMode(LD1, OUTPUT);
63     pinMode(LD2, OUTPUT);
64     pinMode(LD_ISR, OUTPUT);
65
66     // Button
67     pinMode(SW, INPUT_PULLUP);
68
69     Serial.begin(9600);
70     attachInterrupt(SW, buttonISR, FALLING);
71 }
72 void loop() {
73     if (!inISR) {
74         digitalWrite(LD0, HIGH);
75         digitalWrite(LD1, HIGH);
76         digitalWrite(LD2, HIGH);
77         delay(1000);
78
79         digitalWrite(LD0, LOW);
80         digitalWrite(LD1, LOW);
81         digitalWrite(LD2, LOW);
82         delay(1000);
83     }
84 }
85 void buttonISR() {
86     inISR = true;
87     count = 0;
88     Timer3.attachInterrupt(timerISR).start(1000000);
89     delayMicroseconds(100000);
90 }
91
92 void timerISR() {
93     count++;
94     Serial.print("ISR Second: ");
95     Serial.println(count);
96     digitalWrite(LD_ISR, HIGH);
97
98     if (count >= 5) {
99         Timer3.stop(); // Stop Timer after 5 seconds
100         digitalWrite(LD_ISR, LOW);
101         inISR = false;
102     }
103 }
```

Result

 week6_hw1.mp4

HW1_DISCUSSION

Problem Explanation

The goal of this homework was to implement both hardware and software logic where the system performs periodic LED toggling using polling and can interrupt this sequence with a button press. Upon interrupt, the normal LED toggling should pause, and a separate ISR LED must light up for 5 seconds while the elapsed seconds are printed to the serial monitor.

Solution Approach

- **Polling Mechanism:** In the main `loop()`, three LEDs (LD0, LD1, LD2) are toggled every second using delay-based polling. This forms the regular operation mode.
- **Interrupt Handling:** A GPIO interrupt is attached to the push button (SW), which triggers on the FALLING edge. When pressed, it activates an ISR via the `buttonISR()` function.
- **Nested Timer ISR:** Within the button ISR, a hardware timer (`Timer3`) is used to generate 1-second interval interrupts via `timerISR()`. This separates time-tracking logic from GPIO interrupt logic for better modularity.
- **LED and Serial Behavior in ISR Mode:** During the ISR phase, a different LED (LD_ISR) is turned on and the `count` variable is incremented each second. The serial monitor displays this second-by-second status. After 5 seconds, the ISR ends, LD_ISR is turned off, and the system resumes its regular toggling mode.

Key Points

- **Interrupt Nesting:** The combination of a GPIO-triggered interrupt with a timer-based ISR is a notable design decision. It ensures time-accurate behavior during the 5-second pause.
- **Serial Communication:** Serial output inside the timer ISR provides real-time feedback and verification of the elapsed time.

Conclusion

This implementation demonstrates the seamless integration of polling and interrupt-driven behavior using GPIO and hardware timers. The system reacts to user interaction efficiently and maintains accurate timing during the ISR phase.

HOMEWORK2.

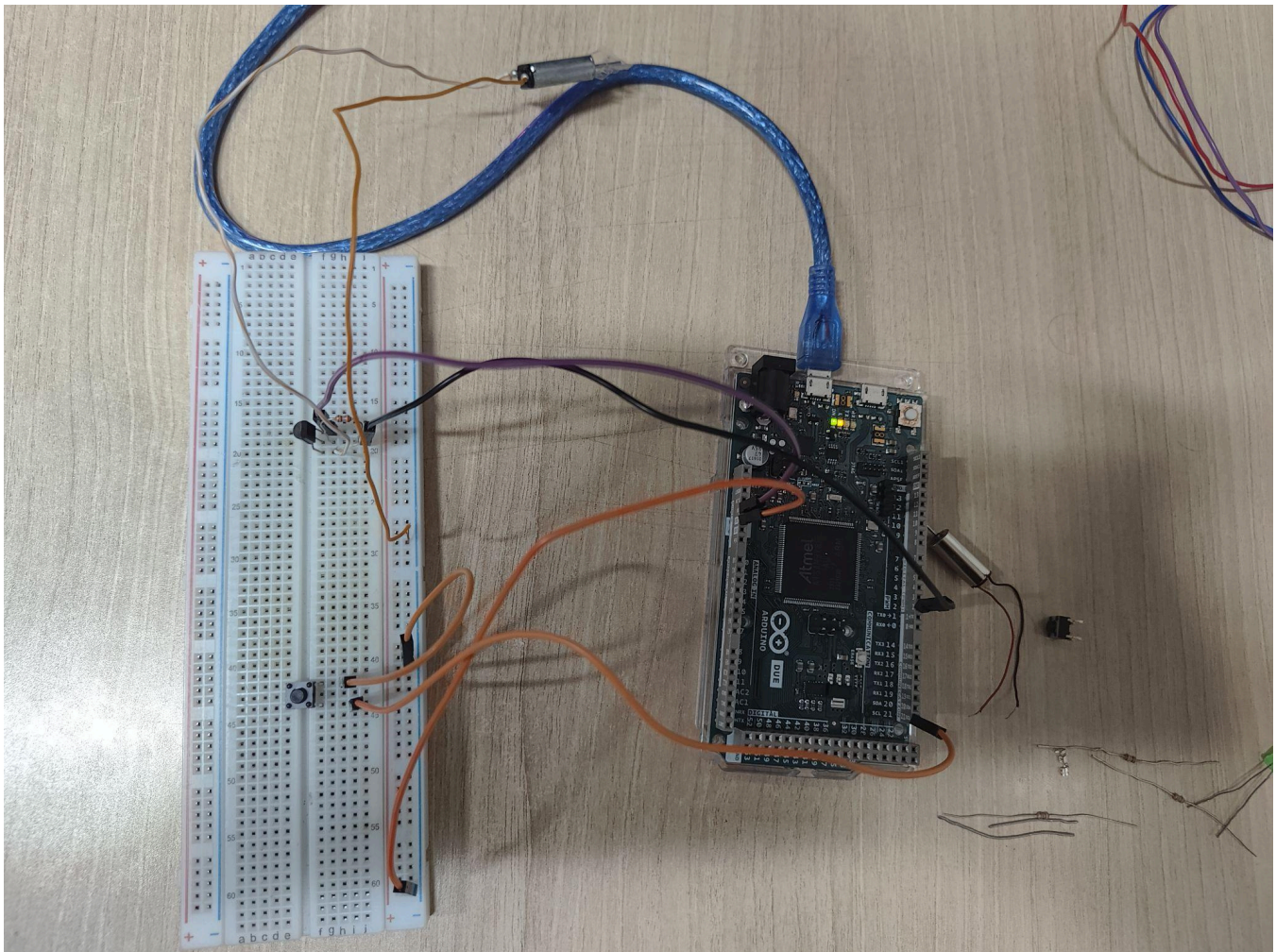
Problem

2.

Using Push button & GPIO interrupt, by pushed number of button (1, 2, 3 times), change DC motor's pwm duty(rotating speed).

Hint : Use DAC (analogWrite()) from week 5 practice


Circuit



Code

```
104 // Team 11: Homework 2
105 #define SW 21
106 #define MOTOR_PWM 2
107 volatile int pressCount = 0;
108 volatile unsigned long lastInterruptTime = 0;
109 void setup() {
110     pinMode(MOTOR_PWM, OUTPUT);
111     pinMode(SW, INPUT_PULLUP);
112
113     attachInterrupt(digitalPinToInterrupt(SW), SW_ISR, FALLING);
114 }
115
116 void loop() {
117     // all handled in ISR
118 }
119
120 void SW_ISR() {
121     unsigned long currentTime = millis();
122     // Debounce: Only trigger if at least 200ms passed since last press
123     if (currentTime - lastInterruptTime > 200) {
124         pressCount++;
125         if (pressCount > 3) {
126             pressCount = 0;
127         }
128
129         switch (pressCount) {
130             case 1:
131                 analogWrite(MOTOR_PWM, 64); // ~25% duty
132                 break;
133             case 2:
134                 analogWrite(MOTOR_PWM, 128); // ~50%
135                 break;
136             case 3:
137                 analogWrite(MOTOR_PWM, 255); // ~100%
138                 break;
139             case 0:
140                 analogWrite(MOTOR_PWM, 0);
141                 break;
142         }
143
144         lastInterruptTime = currentTime;
145     }
146 }
```

Result

 week6_hw2.mp4

HW2_DISCUSSION

Problem Explanation

This homework required designing an Arduino program that changes the speed (PWM duty cycle) of a DC motor based on the number of times a button is pressed. The task involved using GPIO interrupts to detect button presses and incrementally adjusting the motor's speed, simulating a simple speed control interface. Additionally, it was important to implement debounce logic to avoid false triggers caused by switch chattering.

Solution Approach

- **Interrupt-Based Input Handling** – An external GPIO interrupt was attached to a push button using `attachInterrupt()` to detect each button press in real-time without polling.
- **PWM Control Logic** – Each button press increased a `pressCount` value, cycling between 0 and 3, with each step representing an increase in motor speed (0%, ~25%, ~50%, and 100% duty cycle using `analogWrite()`).
- **Debounce with Millis** – Instead of using `delayMicroseconds()` (which blocks execution and may cause issues in ISR), debounce was handled using a timestamp (`millis()`), ensuring at least 200ms passed between valid presses.

Key Points

- **Efficient Interrupt Handling** – The ISR is short and optimized, avoiding delays, and only handles logical operations and PWM updates.

Conclusion

This implementation demonstrates a basic yet practical example of using interrupts and PWM for user-interactive embedded systems. By handling input through GPIO interrupts and adjusting motor speed with clean non-blocking debounce logic, the system achieves responsive and stable motor control.