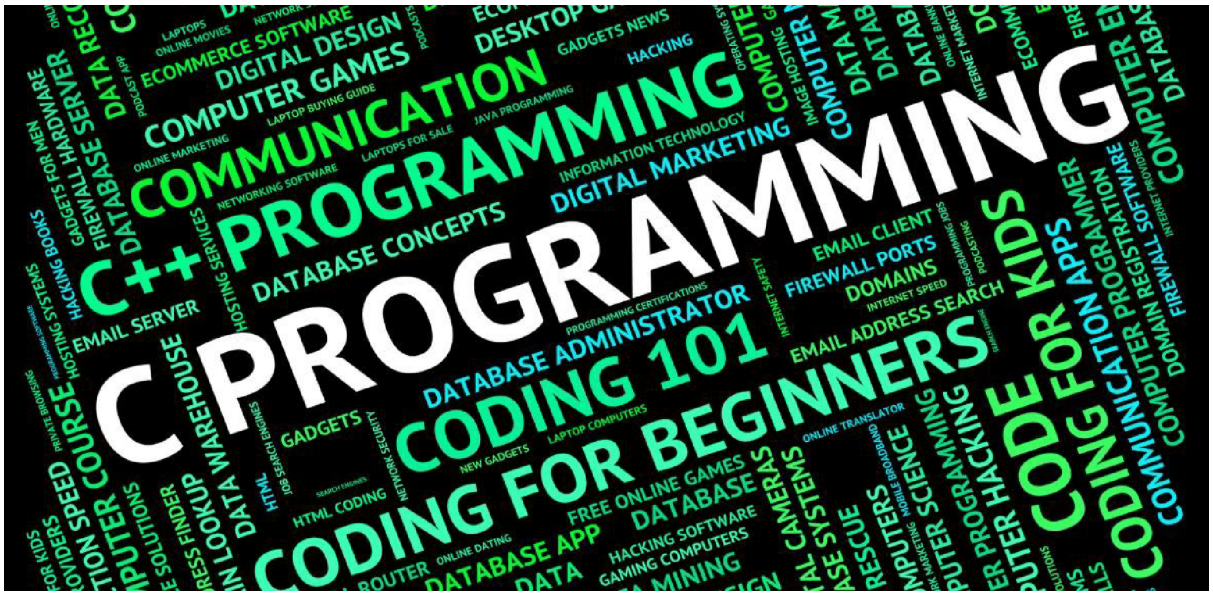

EEE3096S

Practical 1

Arend Jacques du Preez and Georgene De Wet

1. Introduction

This report details the application of C programming (which was comprehensively covered in Embedded Systems I) to programme an STM32 microcontroller. The practical specifically involves programming and implementing timers, LEDs, and pushbuttons on the microcontroller. The objective was to create a sequence of LED patterns with adjustable delays controlled via pushbuttons.

2. Project Objectives

The main objectives of this practical were:

1. To implement a sequence of LED patterns that change every second.
2. To modify the timer delay based on the following inputs of the pushbutton:
- 0.5s when Button 0 (PA0) is pressed.
- 2s when Button 1 (PA1) is pressed.
- 1swhen Button 2 (PA2) is pressed.
3. To reset the LED pattern sequence back to the first pattern when Button 3 (PA3) is pressed.
4. To properly implement the use of Github throughout the development process, thus allowing for a collaborative operation.

3. Implementation

3.1 LED Pattern Sequence

The LED patterns were defined as a 2D array, with each row representing a different pattern. The “currentPatternvariable” was used to track which LED pattern is currently being displayed. Using TIM16 a timer interrupt was set up, which triggers every second to update the LED pattern. The “SetLED” function was responsible for writing the pattern currently being displayed to the LEDs.

Here is a snippet of the pattern 2D array definitions from the main.c file:

```
uint8_t pattern[9][8] = {
        {0,0,0,0,0,0,0,0},
        {1,0,0,0,0,0,0,0},
        {0,1,0,0,0,0,0,0},
        {0,0,1,0,0,0,0,0},
        {1,0,0,1,0,0,0,0},
        {0,1,0,0,1,0,0,0},
        {1,0,1,0,0,1,0,0},
        {1,1,0,1,0,0,1,0},
        {1,1,1,0,1,0,0,1}
        };
```

3.2 Timer Configuration

The TIM16 timer was configured to generate an interrupt every second. Leading to a dynamically changing ARR (Auto Reload Register) based on the pushbutton inputs, this modifying the timer interval.

Here is a snippet of the delay definitions for the timer from the main.c file:

```
// Define delays
#define DELAY_0_5S  500
#define DELAY_1S    1000
#define DELAY_2S    2000
volatile uint32_t currentDelay = DELAY_1S;
```

3.3 Pushbutton Input Handling

The main loop of the program continuously checks the state of the pushbuttons.
The delay is adjusted by modifying the TIM16 ARR value, depending on which button is pressed.

Here is a snippet of the loop function from the main.c file:

```
if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_RESET) {
 __HAL_TIM_SET_AUTORELOAD(&htim16, (1000/2)-1);  // 0.5s delay
} else if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1) == GPIO_PIN_RESET) {
__HAL_TIM_SET_AUTORELOAD(&htim16, (2000)-1);  // 2s delay
} else if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_2) == GPIO_PIN_RESET) {
 __HAL_TIM_SET_AUTORELOAD(&htim16, (1000)-1);  // 1s delay
} else if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_3) == GPIO_PIN_RESET) {
currentPattern = 1;  // Reset to first pattern
SetLED(pattern[currentPattern]);
HAL_Delay(10);  // Debounce delay
}
```

3.4 Timer Interrupt Service Routine

The TIM16 interrupt service routine (ISR) handles the updating of the LED pattern. Each time the timer overflows, the ISR increments the currentPattern variable and updates the LEDs.

Here is a snippet of the ISR from the main.c file:

```
void TIM16_IRQHandler(void) {
HAL_TIM_IRQHandler(&htim16);
currentPattern = (currentPattern + 1) % 9;  // Cycle through patterns
 SetLED(pattern[currentPattern]);
}
```

4. Challenges Encountered

One of the challenges we encountered was pushbuttons debouncing inputs. If the buttons were pushed rapidly this lead to (sometimes) missed inputs. A small delay "(HAL_Delay(10))" was thus added to mitigate this issue.
Another issue was ensuring that the pattern changes occurred precisely at the specified intervals, which was addressed by carefully managing the timer interrupt and using the ARR register to control timing.

5. Conclusion

The practical successfully implemented the use of timers and interrupts in an embedded system (STM32 Microcontroller) via the use of C programming. The LED patterns were displayed correctly in demo, and the timing could be adjusted via the pushbuttons. This practical enhanced understanding of the STM32 microcontroller's capabilities, particularly in handling real-time events through interrupts.

6. GitHub Repository

The complete code for this practical, including the main.c file, is available in the following GitHub repository:
https://github.com/georgene01/EmbeddedSystems_Lab1_TimerProject_Group36