Nirbhay Sharma (B19CSE114)

Digital Systems Lab - 10

Task-1

```
#include "stm32f4xx.h"

main(void)
{
  // enabline clocks
    RCC->AHB1ENR = 0X10; // ENABLING CLOCK FOR PORT E
    RCC->APB2ENR = 0X01; // ENABLINE TIM1

    // configuring pin (PE8)
    GPIOE->MODER = 0X20000;
    GPIOE->PUPDR = 0X00;
    GPIOE->AFR[1] = 0X01;

    // configuring timer
    TIM1->CR1 = 0X01;
    TIM1->PSC = 79;
    TIM1->ARR = 999;
    TIM1->CCMR1 = 0X68;
    TIM1->CR1 |= (1 << 7);
    TIM1->BDTR = (1 << 15);
    TIM1->CCER = (1 << 2);

    TIM1->CR1 |= 0X01;
    while(1){
        TIM1->CCR1 = 500;
    }
}
```

code-explanation

- first enable GPIOE and TIM1 using RCC$\rightarrow$AHB1ENR and RCC$\rightarrow$APB2ENR respectively
- configure pin 8 of GPIOE to alternate function mode (10) so getting 0x20000
- configure pullup pull down register also
- now set GPIOE$\rightarrow$AFRH 0th pin to 0001 (AF1) mode for TIM1 and code wise we write GPIOE $\rightarrow$AFR[1] = 0X01
- now setting psc and arr value using formula

$$f_{clkout} = \frac{f_{clkin}}{(psc+1)(arr+1)}$$

$$100HZ = \frac{8 \times 10^6 HZ}{(psc+1)(arr+1)}$$

$$(psc+1)(arr+1) = 8 \times 10^4$$

$$(psc+1)(arr+1) = 80000$$

so choosing the appropriate value for psc and arr and I choose psc = 79 and arr = 999

- now setting up cr1 and bdtr (setting MOE) registers and ccer regsiters
- now run a while loop and makde ccr1 = 500 since we want 50% duty cycle

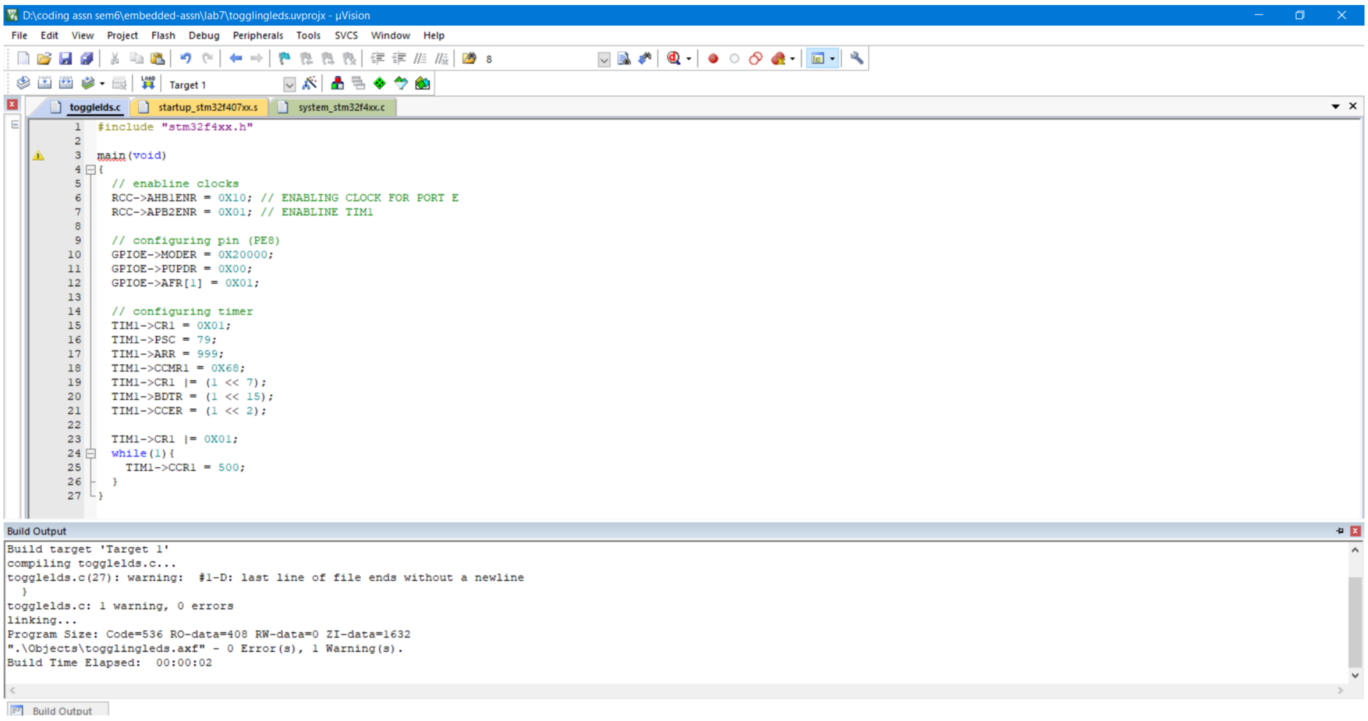
$$DutyCycle = \frac{CCR}{ARR+1}$$

$$0.5 = \frac{CCR}{(999+1)}$$

$$0.5 = \frac{CCR}{1000}$$

$$CCR = 500$$

build-output


D:\coding assn sem6\embedded-assn\lab7\togglingleds.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
togglelds.c
startup_stm32f407xx.s
system_stm32f4xx.c
Build Output

```
#include "stm32f4xx.h"

main(void)
{
  // enabline clocks
  RCC->AHB1ENR = 0X10; // ENABLING CLOCK FOR PORT E
  RCC->APB2ENR = 0X01; // ENABLINE TIM1

  // configuring pin (PE8)
  GPIOE->MODER = 0X20000;
  GPIOE->PUPDR = 0X00;
  GPIOE->AFR[1] = 0X01;

  // configuring timer
  TIM1->CR1 = 0X01;
  TIM1->PSC = 79;
  TIM1->ARR = 999;
  TIM1->CCMR1 = 0X68;
  TIM1->CR1 |= (1 << 7);
  TIM1->BDTR = (1 << 15);
  TIM1->CCER = (1 << 2);

  TIM1->CR1 |= 0X01;
  while(1){
    TIM1->CCR1 = 500;
  }
}
```

```
Build target 'Target 1'
compiling togglelds.c...
togglelds.c(27): warning:  #1-D: last line of file ends without a newline
  }
togglelds.c: 1 warning, 0 errors
linking...
Program Size: Code=536 RO-data=408 RW-data=0 ZI-data=1632
".\Objects\togglingleds.axf" - 0 Error(s), 1 Warning(s).
Build Time Elapsed:  00:00:02
```

Task-2

```
#include "stm32f4xx.h"

void delay(int dd){
    for (int i = 0;i<dd;i++){
        for (int j = 0;j<300000;j++){
            ;
        }
    }
}
```

```
main(void)
{
  // enabline clocks
    RCC->AHB1ENR = 0X10; // ENABLING CLOCK FOR PORT E
    RCC->APB2ENR = 0X01; // ENABLINE TIM1

    // configuring pin (PE8)
    GPIOE->MODER = 0X20000;
    GPIOE->PUPDR = 0X00;
    GPIOE->AFR[1] = 0X01;

    // configuring timer
    TIM1->CR1 = 0X01;
    TIM1->PSC = 79;
    TIM1->ARR = 999;
    TIM1->CCMR1 = 0X68;
    TIM1->CR1 |= (1 << 7);
    TIM1->BDTR = (1 << 15);
    TIM1->CCER = (1 << 2);

    TIM1->CR1 |= 0X01;

    int tim_values_terms = 7;
    int tim_values[] = {200, 300, 500, 400, 900, 800, 600};
    int i = 0;
    while (1) {
        TIM1->CCR1 = tim_values[i];
        i = (i + 1) % tim_values_terms;
        delay(1200);
    }
}
```

code-explanation

- first enable GPIOE and TIM1 using RCC⟶AHB1ENR and RCC⟶APB2ENR respectively
- configure pin 8 of GPIOE to alternate function mode (10) so getting 0x20000
- configure pullup pull down register also
- now set GPIOE⟶AFRH 0th pin to 0001 (AF1) mode for TIM1 and code wise we write GPIOE ⟶AFR[1] = 0X01
- now setting psc and arr value using formula

$$f_{clkout} = \frac{f_{clkin}}{(psc+1)(arr+1)}$$

$$100HZ = \frac{8 \times 10^6 HZ}{(psc+1)(arr+1)}$$

$$(psc+1)(arr+1) = 8 \times 10^4$$

$$(psc+1)(arr+1) = 80000$$

so choosing the appropriate value for psc and arr and I choose psc = 79 and arr = 999

- now setting up cr1 and bdtr (setting MOE) registers and ccer regsiters
- now run a while loop and change ccr1 from an array in order to get PWM output of various duty cycles, so suppose we want x dutycycle

$$DutyCycle = \frac{CCR}{ARR + 1}$$

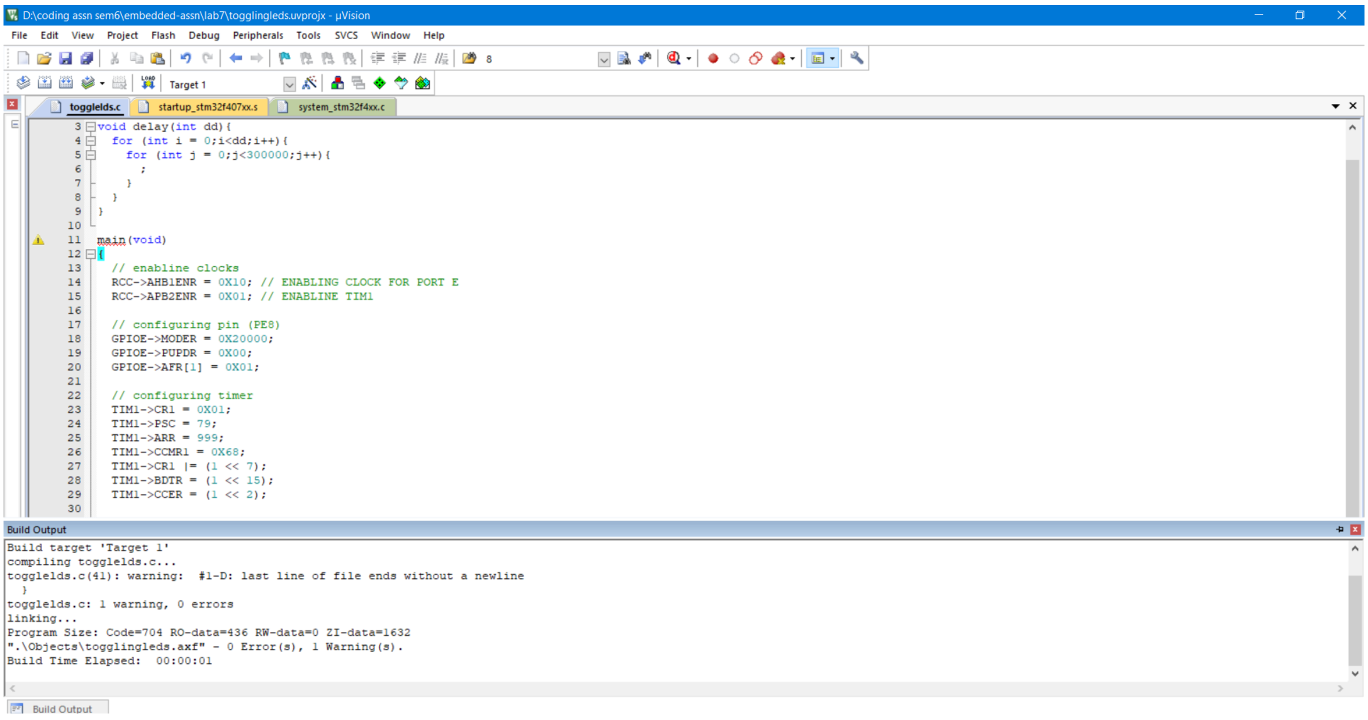
$$\frac{x}{100} = \frac{CCR}{(999 + 1)}$$

$$\frac{x}{100} = \frac{CCR}{1000}$$

$$CCR = 10x$$

- now changing various x we get various values for duty cycle and we can put them in an array and change it in while loop

build-output



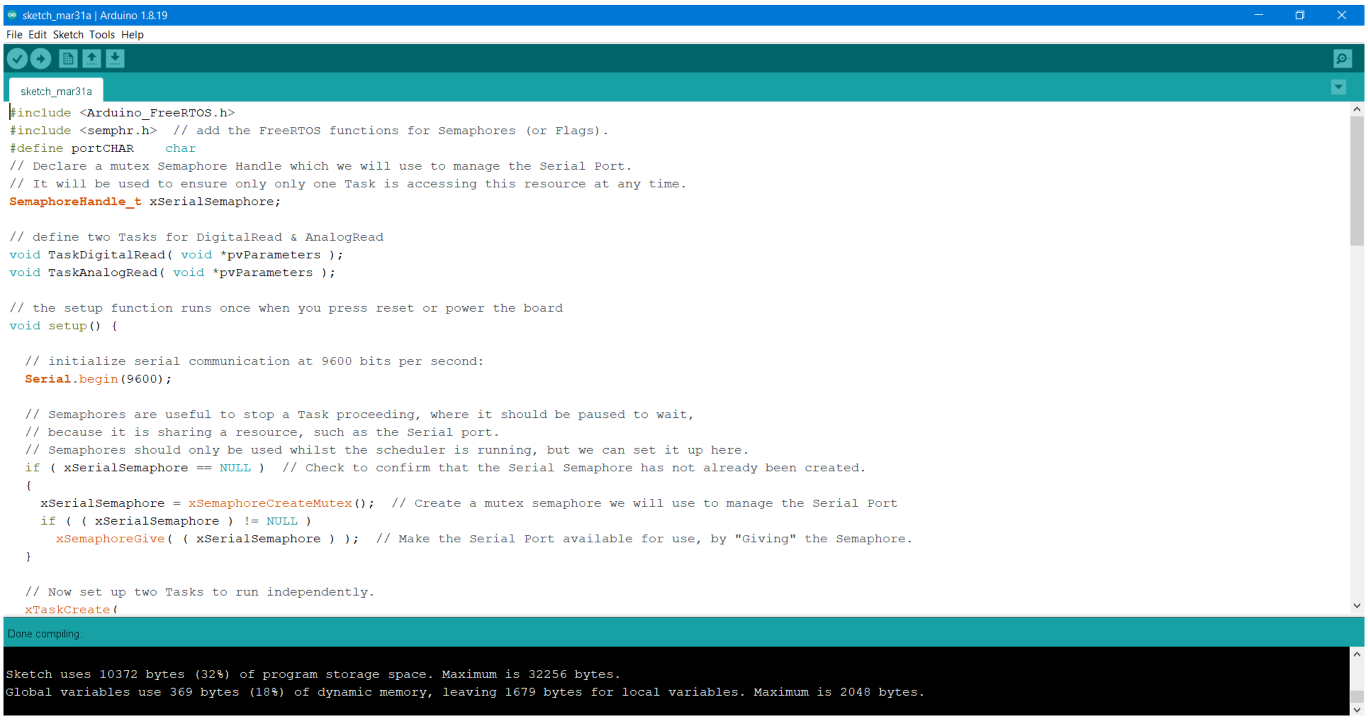
D:\coding assn sem6\embedded-assn\lab7\togglingleds.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
togglelds.c
startup_stm32f407xx.s
system_stm32f4xx.c
Build Output

```
void delay(int dd){
  for (int i = 0;i<dd;i++){
    for (int j = 0;j<300000;j++){
      ;
    }
  }
}

main(void)
{
  // enabline clocks
  RCC->AHB1ENR = 0X10; // ENABLING CLOCK FOR PORT E
  RCC->APB2ENR = 0X01; // ENABLINE TIM1

  // configuring pin (PE8)
  GPIOE->MODER = 0X20000;
  GPIOE->PUPDR = 0X00;
  GPIOE->AFR[1] = 0X01;

  // configuring timer
  TIM1->CR1 = 0X01;
  TIM1->PSC = 79;
  TIM1->ARR = 999;
  TIM1->CCMR1 = 0X68;
  TIM1->CR1 |= (1 << 7);
  TIM1->BDTR = (1 << 15);
  TIM1->CCER = (1 << 2);

```

```
Build target 'Target 1'
compiling togglelds.c...
togglelds.c(41): warning:  #1-D: last line of file ends without a newline
  }
togglelds.c: 1 warning, 0 errors
linking...
Program Size: Code=704 RO-data=436 RW-data=0 ZI-data=1632
".\Objects\togglingleds.axf" - 0 Error(s), 1 Warning(s).
Build Time Elapsed:  00:00:01
```

Task-3

part-1



```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>  // add the FreeRTOS functions for Semaphores (or Flags).
#define portCHAR    char
// Declare a mutex Semaphore Handle which we will use to manage the Serial Port.
// It will be used to ensure only only one Task is accessing this resource at any time.
SemaphoreHandle_t xSerialSemaphore;

// define two Tasks for DigitalRead & AnalogRead
void TaskDigitalRead( void *pvParameters );
void TaskAnalogRead( void *pvParameters );

// the setup function runs once when you press reset or power the board
void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  // Semaphores are useful to stop a Task proceeding, where it should be paused to wait,
  // because it is sharing a resource, such as the Serial port.
  // Semaphores should only be used whilst the scheduler is running, but we can set it up here.
  if ( xSerialSemaphore == NULL )  // Check to confirm that the Serial Semaphore has not already been created.
  {
    xSerialSemaphore = xSemaphoreCreateMutex();  // Create a mutex semaphore we will use to manage the Serial Port
    if ( ( xSerialSemaphore ) != NULL )
      xSemaphoreGive( ( xSerialSemaphore ) );  // Make the Serial Port available for use, by "Giving" the Semaphore.
  }

  // Now set up two Tasks to run independently.
  xTaskCreate(
```

Done compiling.

```
Sketch uses 10372 bytes (32%) of program storage space. Maximum is 32256 bytes.
Global variables use 369 bytes (18%) of dynamic memory, leaving 1679 bytes for local variables. Maximum is 2048 bytes.
```

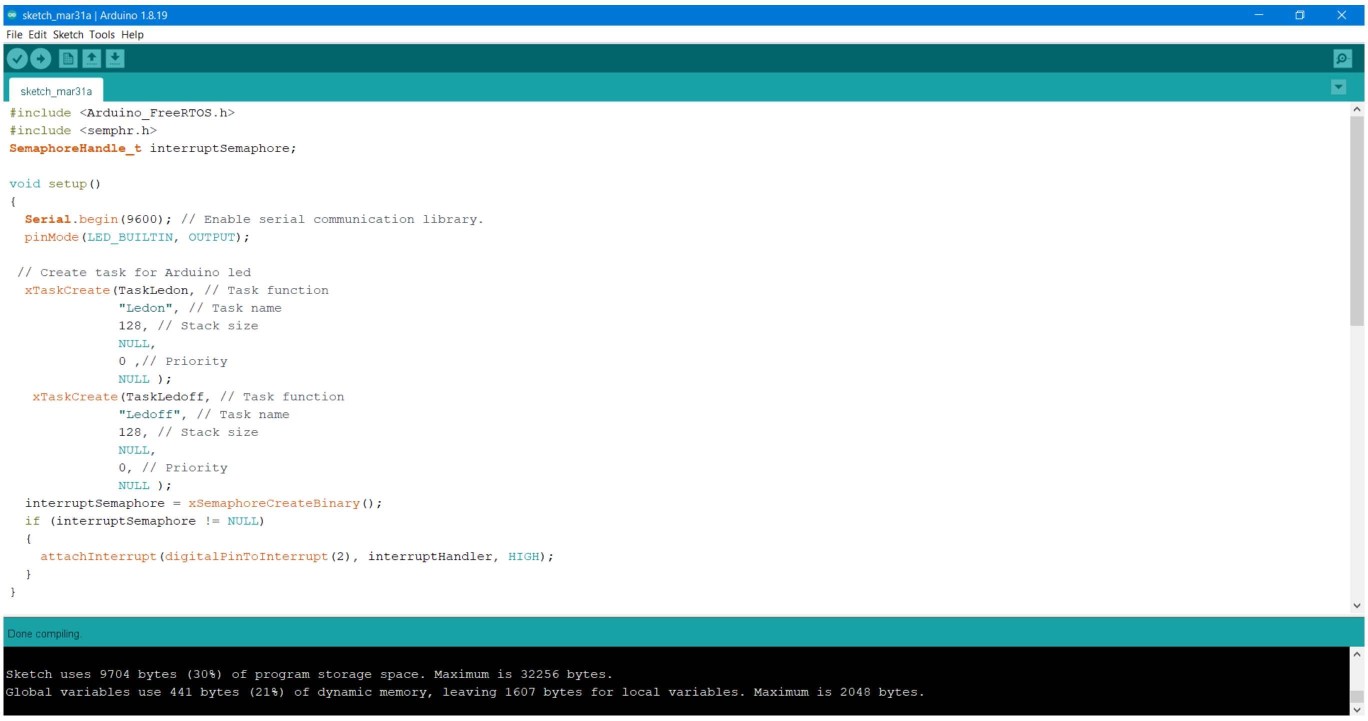
- In this code there are two tasks, TaskDigitalRead, TaskAnalogRead
- so the semaphore here is trying to maintain synchronization between digital read and analog read
- the segment of code given below

  ```
  if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
  {
    Serial.println(buttonState);

    xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial
  Port for others.
  }
  ```

  from above code we can see that each task first try to take semaphore (i.e. decrease it's value) if it is available the task is free to do it's task and once it's task completes it give the semaphore back (i.e. increment its value)

- and hence synchronization is created using a binary semaphore here

part-2



```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
SemaphoreHandle_t interruptSemaphore;

void setup()
{
  Serial.begin(9600); // Enable serial communication library.
  pinMode(LED_BUILTIN, OUTPUT);

 // Create task for Arduino led
  xTaskCreate(TaskLedon, // Task function
              "Ledon", // Task name
              128, // Stack size
              NULL,
              0 ,// Priority
              NULL );
   xTaskCreate(TaskLedoff, // Task function
              "Ledoff", // Task name
              128, // Stack size
              NULL,
              0, // Priority
              NULL );
  interruptSemaphore = xSemaphoreCreateBinary();
  if (interruptSemaphore != NULL)
  {
    attachInterrupt(digitalPinToInterrupt(2), interruptHandler, HIGH);
  }
}
```

Done compiling.

```
Sketch uses 9704 bytes (30%) of program storage space. Maximum is 32256 bytes.
Global variables use 441 bytes (21%) of dynamic memory, leaving 1607 bytes for local variables. Maximum is 2048 bytes.
```

- just like previous part, this part also has two tasks, Ledon, Ledoff
- so here also we are using binary semaphore to maintain synchronization between ledon and off operations, which means that no two process should make led on and off at the same time
- here it is using interrupts to give the semaphore (increment it) and every process once started, take the semaphore (decrement it) and was given back using interrupts which is handled by interrupt handler

---