



ST. ANNE'S COLLEGE OF ENGINEERING AND TECHNOLOGY

Approved by AICTE, New Delhi. Affiliated to Anna University, Chennai

Accredited by NAAC

ANGUCHETTPALAYAM, PANRUTI – 607 106.

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

LAB MANUAL

EE3018 - EMBEDDED PROCESSORS

VI SEMESTER

Program

```
#include "stm32f4xx.h"
#include <stdio.h>
#include "stm32f4xx_rcc.h"

void LED_init();
/* delay function */
void delay(long int a)
{
    long int b,c;
    for(b=0;b<a;b++)
        for(c=0;c<a;c++);
}

/*main function starts here */
void main()
{
    LED_init(); /* LED initialization */
    while(1)
    {
        GPIOD->ODR =0x00000F00; /* leds turns ON */
        delay(3000); /* delay */
        GPIOD->ODR =0x00000000; /* leds turns OFF */
        delay(3000); /* delay */
    }
}

void LED_init()
{
    RCC->AHB1ENR = 1 <<3; /*enable the clock to port D */
    GPIOD->MODER = 0x55555555; /* select all pins as output mode */
    GPIOD->OTYPER =0x00000000; /* push-pull type */
    GPIOD->OSPEEDR =0xAAAAAAAA; /* 50MHz speed */
    GPIOD->PUPDR =0x00000000; /* no pull-up no pull down*/
}
```

Ex. No: 1

Flashing of LED

Aim

To develop a 'C' program to make the LED blink (including delay routine) and upon change in the delay program the speed of the blink should vary.

Apparatus & Software Required

1. VSK-SCM4 Development board.
2. IAR IDE software.
3. Flash Loader Demonstrator.

Theory

- ✓ LEDs are based on the semiconductor diode.
- ✓ When the diode is forward biased (switched on), electrons are able to recombine with holes and energy is released in the form of light. This effect is called electroluminescence.
- ✓ The color of the light is determined by the energy gap of the semiconductor

Procedure

- ✓ Set the current date and time in the PC
- ✓ Open the **IAR Embedded work bench** from the desktop

1. Create the project:

- ✓ Select the **project** from the Menu bar
- ✓ Select **create new project** from the project menu bar.
- ✓ Click **empty project**
- ✓ Click **ok**
- ✓ Create a new folder in the 'C' drive and give the name of the folder.
- ✓ Save the project in the created folder
- ✓ Then click **ok**
- ✓ Copy **INC** folder and supporting file **STM_lib.a** from main folder and paste it in to created folder

2. Create the file:

- ✓ Select the **file** from the menu bar.
- ✓ Click **new** and the select **file**
- ✓ Type the program
- ✓ Save the program with the extension of (.c).

3. Add the Source file & Supporting file:

- ✓ Right click on project name
- ✓ Select **Add**
- ✓ Select file name.c
- ✓ Again, right click on project name
- ✓ Then click **add files** to select a STM.lib.a (library files)

4. Setting Options:

- ✓ Enable created project
- ✓ Click **project** from menu bar to select the **options**
- ✓ Click **general options** to select **Device**
- ✓ Select ST→STM32F407→ STM32F407VG.
- ✓ Click **C\C++ compiler** and select the **preprocessor** from menu and then type **\$PROJ_DIR\$\INC.**
- ✓ Click **output converter** to enable the **generate** and select **intel extended.**
- ✓ Click **linker** and enable **override default.**
- ✓ Click **debugger** and select **ST- LINK** and click **download** from menu and enable **user flash loader.**

- ✓ Click **ST-LINK** and enable **SWD**.
- ✓ Finally click OK.

5. Compile the program:

- ✓ Click **project** and select **rebuild all**.
- ✓ Click **project** and select download and click **download active application**.

6. Output verification:

- ✓ Open the **STM32ST link utility** from the desktop.
- ✓ Click **target** from menu bar and select **connect**
- ✓ Again then click **target** and to select **program and verify**.
- ✓ Click **debug** and to select the **.hex file** from the **Exe** folder
- ✓ LED is blinking.

Result:

The C-Language program to make the LED blink was developed and output was verified and upon change in the delay program the speed variation was verified.

Program

```
#include "stm32f4xx.h"

#include <stdio.h>

#include "stm32f4xx_gpio.h"

#include "stm32f4xx_rcc.h"

#include "stm32f4xx_adc.h"

int ConvertedValue = 0;          /* Converted value read from ADC1 */

void adc_configure()

{

    /* Clock configuration */

    RCC->APB2ENR |= 1<<10;      /*The ADC3 is connected the APB2 peripheral bus thus we will
                                   use its clock source */
    RCC->AHB1ENR |= 1<<0;      /* Clock for the ADC port!! Do not forget about this one ;) */

    /* Analog pin configuration */

    GPIOA->MODER |= 0x0000000F;  /* analog mode PA1,PA2 */

    GPIOA->OSPEEDR = 0xFFFFFFFF;

    GPIOA->PUPDR = 0x00000000;

    /* ADC configuration */

    ADC3->CR1 = 0x00000000;      /* scan mode disable,12-bit resolution. */
    ADC3->CR2 = 0x00000002;      /* data right alignment,continuous conversion mode. */
    ADC3->SQR1 = 0x00000000;     /* single mode conversion */
    ADC3->CR2 |= 0x00000001;     /* ADC enable */
    ADC3->SMPR2 = 0x00000030;    /*ADC3 channel-1 with 144 cycles */
    ADC3->SQR3 = 0x00000001;     /* rank1 to ADC3 channel-1 */

}
```

Aim

To develop a C-Language program for reading an on-chip ADC, convert into decimal and to display it in PC and to generate a square wave depending on the ADC reading. The ADC input is connected to any analog sensor/ on board potentiometer.

Apparatus & Software Required

1. VSK-SCM4 Development board.
2. IAR IDE software.
3. Flash Loader Demonstrator.

Theory

- ✓ Three 12-bit analog-to-digital converters are embedded and each ADC shares up to 16 external channels, performing conversions in the single-shot or scan mode.
- ✓ In scan mode, automatic conversion is performed on a selected group of analog inputs.
- ✓ Additional logic functions embedded in the ADC interface allow:
 - Simultaneous sample and hold
 - Interleaved sample and hold

```

int adc_convert()
{
    ADC_SoftwareStartConv(ADC3);                /* Start the conversion */
    while(!ADC_GetFlagStatus(ADC3, ADC_FLAG_EOC));    /* Processing the conversion */
    return ADC_GetConversionValue(ADC3);            /* Return the converted data */
}

void USART2_config()
{
    RCC->AHB1ENR |= 1 << 0;                /* clock to portA */ RCC-
    >APB1ENR |= 1 <<17;                /* clock to USART2 */
    GPIOA->MODER |= 0x000000A0;            /* alternate function mode(PA2,PA3) */
    GPIOA->AFR[0] |= 0x00007700;            /* USART2 AF */
    USART2->BRR = 0x16D;                /*115200 baud rate */
    USART2->CR3 = 0x0000;
    USART2->CR2 = 0x000; USART2-
    >CR1 = 0x200C;
}

int main(void)
{
    USART2_config();
    adc_configure();                /*Start configuration */
    while(1)
    {
        ConvertedValue = adc_convert();    /*Read the ADC converted value */
        printf("\n ADC value => %d",ConvertedValue); /*print the ADC value */
    }
}

```

Procedure

- ✓ Set the current date and time in the PC
- ✓ Open the **IAR Embedded work bench** from the desktop

1. Create the project:

- ✓ Select the **project** from the Menu bar
- ✓ Select **create new project** from the project menu bar.
- ✓ Click **empty project**
- ✓ Click **ok**
- ✓ Create a new folder in the 'C' drive and give the name of the folder.
- ✓ Save the project in the created folder
- ✓ Then click **ok**
- ✓ Copy **INC** folder and supporting file **STM_lib.a** from main folder and paste it in to created folder

2. Create the file:

- ✓ Select the **file** from the menu bar.
- ✓ Click **new** and the select **file**
- ✓ Type the program
- ✓ Save the program with the extension of (.c).

3. Add the Source file & Supporting file:

- ✓ Right click on project name
- ✓ Select **Add**
- ✓ Select file name.c
- ✓ Again, right click on project name
- ✓ Then click **add files** to select a STM.lib.a (library files)

4. Setting Options:

- ✓ Enable created project
- ✓ Click **project** from menu bar to select the **options**
- ✓ Click **general options** to select **Device**
- ✓ Select ST→STM32F407→ STM32F407VG.
- ✓ Click **C\C++ compiler** and select the **preprocessor** from menu and then type **\$PROJ_DIR\$\INC.**
- ✓ Click **output converter** to enable the **generate** and select **intel extended**.

```
int putchar(int data)
{
    USART2->DR = (data & 0x01FF);

    /* Loop until the end of transmission */
    while((USART2->SR & 0x40) == 0)
    {
        return data;
    }
}
```

- ✓ Click **linker** and enable **override default**.
- ✓ Click **debugger** and select **ST- LINK** and click **download** from menu and enable **user flash loader**.
- ✓ Click **ST-LINK** and enable **SWD**.
- ✓ Finally click OK.

5. Compile the program:

- ✓ Click **project** and select **rebuild all**.
- ✓ Click **project** and select download and click **download active application**.

6. Output verification:

- ✓ Open the **STM32ST link utility** from the desktop.
- ✓ Click **target** from menu bar and select **connect**
- ✓ Again then click **target** and to select **program and verify**.
- ✓ Click **debug** and to select the **.hex file** from the **Exe** folder
- ✓ Digital output is displayed in personal computer.

RESULT:

Thus a C-Language program was developed for reading an on-chip ADC to convert into decimal and to display it in PC and to generate a square wave depending on the ADC value

Program

```
#include "stm32f4xx.h"
#include "stm32f4xx_dac.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_tim.h"
#include "stm32f4xx_syscfg.h"
static void TIM6_Config(void)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

    /* TIM6 Periph clock enable */ RCC_APB1Periph_Clock
    Cmd(RCC_APB1Periph_TIM6, ENABLE);

    /* Time base configuration */ TIM_TimeBaseStructInit(&TIM_TimeBase
    Structure); TIM_TimeBaseStructure.TIM_Period = 0xFF; TIM_Time
    BaseStructure.TIM_Prescaler = 0; TIM_TimeBase
    Structure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM6, &TIM_TimeBaseStructure);

    /* TIM6 TRGO selection */
    TIM_SelectOutputTrigger(TIM6, TIM_TRGOSource_Update);

    /* TIM6 enable counter */
    TIM_Cmd(TIM6, ENABLE);
}
static void DAC_Ch2_TriangleConfig(void)
{
    DAC_InitTypeDef DAC_InitStructure;
```

Ex. No: 3**Interfacing DAC****Aim**

To develop a C-Language program for reading an on-chip DAC, convert into analog and to display it in CRO.

Apparatus & Software Required

1. VSK-SCM4 Development board.
2. IAR IDE software.
3. Flash Loader Demonstrator.
4. CRO

Theory:

- ✓ The two 12-bit buffered DAC channels can be used to convert two digital signals into two analog voltage signal outputs.
- ✓ Eight DAC trigger inputs are used in the device.
- ✓ The DAC channels are triggered through the timer update outputs that are also connected to different DMA streams.
- ✓ This dual digital Interface supports the following features:
 - Two DAC converters: one for each output channel
 - 8-bit or 12-bit monotonic output
 - Left or right data alignment in 12-bit mode
 - Synchronized update capability
 - Noise-wave generation
 - Triangular-wave generation
 - Dual DAC channel independent or simultaneous conversions
 - DMA capability for each channel
 - External triggers for conversion
 - Input voltage reference VREF+

```

DAC_WaveGeneration_Triangle; DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude
= DAC_TriangleAmplitude_1023; DAC_InitStructure.DAC_OutputBuffer
= DAC_OutputBuffer_Enable; DAC_Init(DAC_Channel_1, &DAC_InitStructure);
/* Enable DAC Channel2 */
DAC_Cmd(DAC_Channel_1, ENABLE);
/* Set DAC channel2 DHR12RD register */
DAC_SetChannel2Data(DAC_Align_12b_R, 0x100);
}
int main(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    /* DMA1 clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);
/* GPIOA clock enable (to be used with DAC) */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    /* DAC Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
    /* DAC channel 1 & 2 (DAC_OUT1 = PA.4)(DAC_OUT2 = PA.5) configuration */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN; //DAC pin A5 as analog mode

    GPIO_InitStructure.GPIO_PuPd    =    GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    TIM6_Config();
while (1)
    {
        DAC_Ch2_TriangleConfig();
    }
}

```

Procedure

- ✓ Set the current date and time in the PC
- ✓ Open the **IAR Embedded work bench** from the desktop

1. Create the project:

- ✓ Select the **project** from the Menu bar
- ✓ Select **create new project** from the project menu bar.
- ✓ Click **empty project**
- ✓ Click **ok**
- ✓ Create a new folder in the 'C' drive and give the name of the folder.
- ✓ Save the project in the created folder
- ✓ Then click **ok**
- ✓ Copy **INC** folder and supporting file **STM_lib.a** from main folder and paste it in to created folder

2. Create the file:

- ✓ Select the **file** from the menu bar.
- ✓ Click **new** and the select **file**
- ✓ Type the program
- ✓ Save the program with the extension of (.c).

3. Add the Source file & Supporting file:

- ✓ Right click on project name
- ✓ Select **Add**
- ✓ Select file name.c
- ✓ Again, right click on project name
- ✓ Then click **add files** to select a STM.lib.a (library files)

4. Setting Options:

- ✓ Enable created project
- ✓ Click **project** from menu bar to select the **options**
- ✓ Click **general options** to select **Device**
- ✓ Select ST→STM32F407→ STM32F407VG.
- ✓ Click **C\C++ compiler** and select the **preprocessor** from menu and then type **\$PROJ_DIR\$\INC.**
- ✓ Click **output converter** to enable the **generate** and select **intel extended**.
- ✓ Click **linker** and enable **override default**.

- ✓ Click **debugger** and select **ST- LINK** and click **download** from menu and enable **user flash loader**.
- ✓ Click **ST-LINK** and enable **SWD**.
- ✓ Finally click OK.

5. Compile the program:

- ✓ Click **project** and select **rebuild all**.
- ✓ Click **project** and select download and click **download active application**.

6. Output verification:

- ✓ Open the **STM32ST link utility** from the desktop.
- ✓ Click **target** from menu bar and select **connect**
- ✓ Again then click **target** and to select **program and verify**.
- ✓ Click **debug** and to select the **.hex file** from the **Exe** folder
- ✓ Use CRO or DSO to view the DAC output by connecting the +ve terminal to pin and the –ve terminal to GND.
- ✓ Observe the waveform and plot the graph

RESULT:

Thus a C-Language program was developed for reading an on-chip ADC to convert into decimal and to display it in PC and to generate a square wave depending on the ADC value

Program

```
/* Includes -----*/

#include "stm32f4xx.h"

#include <stdio.h>

uint16_t PrescalerValue = 0;

static int duty_cycle = 1 ;

int duty;

/* Private function prototypes -----*/

void TIM10_Config(void); void
TIM11_Config(void); void
TIM3_Config(void); void delay(int
delay);

int switch_read();

void mx_pinout_config(void);

void USART2_config();

/* main functions -----*/

int main(void)

{

    USART2_config();

    TIM3_Config(); while(1)
```

Ex.No: 4

PWM Generator

Aim

To write a C program to generate a PWM and to vary the brightness of the LED depending on the duty cycle.

Apparatus & Software Required

1. VSK-SCM4 Development board.
2. IAR IDE software.
3. Flash Loader Demonstrator. .
4. CRO
5. LED board Setup

Theory

- ✓ The PWM is based on the standard timer block and inherits all of its features.
- ✓ The timer is designed to count cycles of the peripheral clock (PCLK) and optionally generate interrupts or perform other actions when specified timer values occur, based on seven match registers.
- ✓ The PWM function is also based on match register events.

```

{
    duty = switch_read();                /* receive switch value */

    /* for debug purpose */
    // printf("\n duty =%d",duty);

    /* Compute the prescaler value */

    PrescalerValue = (uint16_t) ((SystemCoreClock /2) / 21000000) - 1;    /* 21MHZ timer/counter
clock */
    TIM3->ARR =0x348;                    /* 25khz */ TIM3-
    >PSC =PrescalerValue;
    TIM3->CR1 = 0<<8|0<<9;
    TIM3->CCMR1 = 1<<4|1<<5|1<<6;
    TIM3->CCER |= 1<<0|1<<1;            /*enable capture/compare2*/
    TIM3->CCR1 = duty;                  /* apply duty cycle */
    TIM3->CCMR1 = 1<<3;                 /* enable channel1 preload register */
    TIM3->CCMR1 = 1<<12|1<<13|1<<14;
    TIM3->CCER |= 1<<4|1<<5;           /* enable capture/compare2 */
    TIM3->CCR2 = duty;                  /* apply duty cycle */
    TIM3->CCMR1 = 1<<11;               /* enable channel2 preload register */
    /* TIM10 enable counter */
    TIM3->CR1 =1<<0;
}
}

void TIM3_Config(void)
{
    RCC->APB1ENR |=1<<1;    RCC-
    >AHB1ENR |=1<<2;

    GPIOC->MODER |=0x0000A000;
    GPIOC->AFR[0]=0x22000000;}

```

Procedure

- ✓ Set the current date and time in the PC
- ✓ Open the **IAR Embedded work bench** from the desktop

1. Create the project:

- ✓ Select the **project** from the Menu bar
- ✓ Select **create new project** from the project menu bar.
- ✓ Click **empty project**
- ✓ Click **ok**
- ✓ Create a new folder in the 'C' drive and give the name of the folder.
- ✓ Save the project in the created folder
- ✓ Then click **ok**
- ✓ Copy **INC** folder and supporting file **STM_lib.a** from main folder and paste it in to created folder

2. Create the file:

- ✓ Select the **file** from the menu bar.
- ✓ Click **new** and the select **file**
- ✓ Type the program
- ✓ Save the program with the extension of (.c).

3. Add the Source file & Supporting file:

- ✓ Right click on project name
- ✓ Select **Add**
- ✓ Select file name.c
- ✓ Again, right click on project name
- ✓ Then click **add files** to select a STM.lib.a (library files)

```

int switch_read()
{
    int val;
    mx_pinout_config();
    val = ((GPIOE->IDR)&0x14);
    if(val == 0x04)
    {
        if(duty_cycle <= 0)
        {
            duty_cycle =1;
        }
        duty_cycle--;
        delay(250);
        /* variable decrement */
    }
else if(val == 0x10)
    {
        duty_cycle++;
        delay(250);
        /* variable increment */
        if(duty >= 1500)
        {
            duty =1500;
        }
    }
    return(duty_cycle);
}

int putchar(int data)

```

4. Setting Options:

- ✓ Enable created project
- ✓ Click **project** from menu bar to select the **options**
- ✓ Click **general options** to select **Device**
- ✓ Select ST→STM32F407→ STM32F407VG.
- ✓ Click **C\C++ compiler** and select the **preprocessor** from menu and then type **\$PROJ_DIR\$\INC**.
- ✓ Click **output converter** to enable the **generate** and select **intel extended**.
- ✓ Click **linker** and enable **override default**.
- ✓ Click **debugger** and select **ST- LINK** and click **download** from menu and enable **user flash loader**.
- ✓ Click **ST-LINK** and enable **SWD**.
- ✓ Finally click OK.

5. Compile the program:

- ✓ Click **project** and select **rebuild all**.
- ✓ Click **project** and select download and click **download active application**.

6. Output verification:

- ✓ Open the **STM32ST link utility** from the desktop.
- ✓ Click **target** from menu bar and select **connect**
- ✓ Again then click **target** and to select **program and verify**.
- ✓ Click **debug** and to select the **.hex file** from the **Exe** folder
- ✓ Connect the +ve terminal of the LED board to the connector and –ve terminal to GND.
- ✓ On varying the duty cycle of the PWM, the intensity of the LEDs will vary.
- ✓ To increase the duty cycle press SW2 and to decrease, press SW5.

```

{
    USART2->DR = (data & 0x01FF);

    /* Loop until the end of transmission */
    while((USART2->SR & 0x40) == 0)
    {
        return data;
    }
}

void mx_pinout_config(void)
{
    /*Enable or disable the AHB1 peripheral clock */

    RCC->AHB1ENR |= 1<<3;                /* enable the clock to portD */ RCC-
    >AHB1ENR |= 1<<4;                /*enable clock to portE */

    GPIOE->MODER |= 0x00000000;          /*set portE as i/p mode, other pins are input mode */
    GPIOE->OTYPER = 0x00000000;
    GPIOE->PUPDR  =  0x00000000;  GPIOE-
    >OSPEEDR = 0xAAAAAAAA;
}

void USART2_config()
{
    RCC->AHB1ENR |= 1 << 0;            /* clock to PORTA */ RCC-
    >APB1ENR |= 1 <<17;                /* clock to USART2 */ GPIOA-
    >MODER |= 0x000000A0;              /* AF mode */

    GPIOA->AFR[0] |= 0x00007700;        /* USART2 AF */
    USART2->BRR = 0x16D;                /* 115200 baud rate */
    USART2->CR3 = 0x0000;
    USART2->CR2 = 0x000; USART2-
    >CR1 = 0x200C;}

```

```
void delay(int delay)
{
    int d1,d2; for(d1=0;d1<delay;d1++)
        for(d2=0;d2<delay;d2++);
}
```

Result

The C program is written to vary the intensity of the LED depending on the duty cycle of the PWM has been developed and is verified.

Program

```
#include "stm32f4xx.h"

#include "stm32f4xx_i2c.h"

#include "stm32f4xx_gpio.h"

#include "stm32f4xx_rcc.h"

#include "stm32f4xx_usart.h"

#include "misc.h"

#include "stdio.h"

//uint8_t GK[64] = "ABCD";

#define SLAVE_ADDRESS1 0xD0      // the slave address for write

#define SLAVE_ADDRESS2 0xD1      // the slave address for read

/* prototype functions */

void I2C_stop(I2C_TypeDef* I2Cx);

void I2C_write(I2C_TypeDef* I2Cx, uint8_t data);

void I2C_start(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction);

void init_I2C1();

void Delay(int t);

void USART2_config();

int BCD_to_DECIMAL(int bcd);

int RTC_read(I2C_TypeDef* I2Cx,uint8_t address);

int main(void)

{
    USART2_config();
    printf("\n welcome to RTC");

    init_I2C1();

    int gk_sec,gk_min,gk_hrr;

    I2C_start(I2C1, SLAVE_ADDRESS1, I2C_Direction_Transmitter);
```

Ex.No .5 RTC interfacing with ARM using Serial communication programming

Aim

To develop a C-Language program for reading the RTC, convert into decimal and to display it.

Apparatus & Software Required

1. VSK-SCM4 Development board.
2. IAR IDE software.
3. Flash Loader Demonstrator.

Theory

- ✓ The real-time clock (RTC) is an independent BCD timer/counter.
- ✓ Dedicated registers contain the second, minute, hour (in 12/24 hour), week day, date, month, year, in BCD (binary coded decimal) format.
- ✓ Correction for 28, 29 (leap year), 30, and 31 day of the month are performed automatically.
- ✓ The RTC provides a programmable alarm and programmable periodic interrupts with wakeup from Stop and Standby modes.
- ✓ The sub- seconds value is also available in binary format.
- ✓ It is clocked by a 32.768 kHz external crystal, resonator or oscillator, the internal low- power RC oscillator or the high-speed external clock divided by 128.
- ✓ The internal low- speed RC has a typical frequency of 32 kHz.
- ✓ The RTC can be calibrated using an external 512 Hz output to compensate for any natural quartz deviation.

```

I2C_write(I2C1,0x00);    /* start a transmission in Master transmitter mode */
I2C_write(I2C1,0x00);    /*send RTC second address */
I2C_write(I2C1,0x00);    /* send RTC second data */
I2C_write(I2C1,0x08);    /* send RTC min data */
I2C_stop(I2C1);          /* send RTC hour data */

Delay(500);

while(1)
{
    gk_sec = RTC_read(I2C1,0x00);    /*read RTC second */ gk_min
    = RTC_read(I2C1,0x01);    /*read RTC minute */ gk_hrr =
    RTC_read(I2C1,0x02);    /*read RTC hour */

    printf("\n\t RTC_TIME => %d:%d:%d",gk_hrr,gk_min,gk_sec);

}

}

void Delay(int t)
{
    int y,u;  for(y=0;y<t;y++)
    for(u=0;u<t;u++);
}

int RTC_read(I2C_TypeDef* I2Cx,uint8_t address)
{
    int r_value,gj;

    I2C_start(I2Cx, SLAVE_ADDRESS1, I2C_Direction_Transmitter);

    /* start a transmission in Master transmitter mode */

    I2C_write(I2Cx,address);    /* send EEPROM MSB address */

    Delay(200);

```

Procedure

- ✓ Set the current date and time in the PC
- ✓ Open the **IAR Embedded work bench** from the desktop

1. Create the project:

- ✓ Select the **project** from the Menu bar
- ✓ Select **create new project** from the project menu bar.
- ✓ Click **empty project**
- ✓ Click **ok**
- ✓ Create a new folder in the 'C' drive and give the name of the folder.
- ✓ Save the project in the created folder
- ✓ Then click **ok**
- ✓ Copy **INC** folder and supporting file **STM_lib.a** from main folder and paste it in to created folder

2. Create the file:

- ✓ Select the **file** from the menu bar.
- ✓ Click **new** and the select **file**
- ✓ Type the program
- ✓ Save the program with the extension of (.c).

3. Add the Source file & Supporting file:

- ✓ Right click on project name
- ✓ Select **Add**
- ✓ Select file name.c
- ✓ Again, right click on project name
- ✓ Then click **add files** to select a STM.lib.a (library files)

```

    I2C_stop(I2Cx);

    Delay(200);

    I2C_start(I2Cx, SLAVE_ADDRESS2, I2C_Direction_Receiver);

    /* start a transmission in Master transmitter mode */

    Delay(200);

    gj = I2C_ReceiveData(I2Cx);

    r_value = BCD_to_DECIMAL(gj);

    I2C_stop(I2Cx);

    Delay(3700);

    return(r_value);
}

int BCD_to_DECIMAL(int bcd)
{
    int gk_MSB,gk_LSB,decimal; gk_MSB
    = (bcd&0xF0)>>4; gk_LSB =
    (bcd&0x0F);

    decimal =(gk_MSB*10)+gk_LSB;

    return(decimal);
}

void USART2_config()
{
    RCC->AHB1ENR |= 1 << 0; RCC-
    >APB1ENR |= 1 <<17;

    GPIOA->MODER |= 0x000000A0; GPIOA-
    >AFR[0] |= 0x00007700;

```

4. Setting Options:

- ✓ Enable created project
- ✓ Click **project** from menu bar to select the **options**
- ✓ Click **general options** to select **Device**
- ✓ Select ST→STM32F407→ STM32F407VG.
- ✓ Click **C\C++ compiler** and select the **preprocessor** from menu and then type **\$PROJ_DIR\$\INC**.
- ✓ Click **output converter** to enable the **generate** and select **intel extended**.

- ✓ Click **linker** and enable **override default**.
- ✓ Click **debugger** and select **ST- LINK** and click **download** from menu and enable **user flash loader**.
- ✓ Click **ST-LINK** and enable **SWD**.
- ✓ Finally click OK.

5. Compile the program:

- ✓ Click **project** and select **rebuild all**.
- ✓ Click **project** and select download and click **download active application**.

6. Output verification:

- ✓ Open the **STM32ST link utility** from the desktop.
- ✓ Click **target** from menu bar and select **connect**
- ✓ Again then click **target** and to select **program and verify**.
- ✓ Click **debug** and to select the **.hex file** from the **Exe** folder
- ✓ View the result in WINX talk.
- ✓ The time will be executed in PC.
- ✓ To change the time i.e. hours and minutes press the switches SW5 and SW2 respectively.

```
USART2->BRR = 0x16D;//115200 baud rate
```

```
USART2->CR3 = 0x0000; USART2-
```

```
>CR2 = 0x000; USART2->CR1 =
```

```
0x200C;
```

```
}
```

```
int putchar(int data)
```

```
{
```

```
USART2->DR = (data & 0x01FF);
```

```
/* Loop until the end of transmission */
```

```
while((USART2->SR & 0x40) == 0)
```

```
{}
```

```
return data;
```

```
}
```

```
void init_I2C1()
```

```
{
```

```
I2C_InitTypeDef I2C_InitStruct;
```

```
/* setup SCL and SDA pins
```

```
* You can connect I2C1 to two different
```

```
* pairs of pins:
```

```
* 1. SCL on PB6 and SDA on PB9
```

```
*/
```

```
RCC->AHB1ENR |= 1<<1; //CLK to port B RCC-
```

```
>APB1ENR |= 1<<21; // CLK to I2C1
```

```
GPIOB->MODER = 0x00082000;
```

```
/* PB6 & PB9 =>i2c pins (SDA,SCL) */
```

```
GPIOB->AFR[0] = 0x04000000;
```

```
GPIOB->AFR[1] = 0x00000040;
```

```

GPIOB->OTYPER = 0xFFFFFFFF;                /*open drain */

GPIOB->OSPEEDR =0xAAAAAAAA;                  /* 50 MHZ */ GPIOB-
>PUPDR = 0x55555555;                        /* pull up */

/* configure I2C1 */
I2C_InitStruct.I2C_ClockSpeed = 100000;      /* 100kHz */
I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;      /* I2C mode */
I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2; /* 50% duty cycle --> standard*/
I2C_InitStruct.I2C_OwnAddress1 = 0x00;        /* own address, not relevant in master mode*/
I2C_InitStruct.I2C_Ack = I2C_Ack_Disable;

/* disable acknowledge when reading (can be changed later on) */
I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;

/* set address length to 7 bit addresses */

I2C_Init(I2C1, &I2C_InitStruct);             /* init I2C1 */


/* enable I2C1 */ I2C_Cmd(I2C1,
ENABLE);

}

void I2C_start(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction)

{

/* wait until I2C1 is not busy anymore */

```

```

while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));

    /* Send I2C1 START condition */

I2C_GenerateSTART(I2Cx, ENABLE);

    /* wait for I2C1 EV5 --> Slave has acknowledged start condition */
while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

    /* Send slave Address for write */

I2C_Send7bitAddress(I2Cx, address, direction);

    /* wait for I2C1 EV6, check if
        * either Slave has acknowledged Master transmitter or
        * Master receiver mode, depending on the transmission
        * direction */

if(direction == I2C_Direction_Transmitter)
{
    while(!I2C_CheckEvent(I2Cx,
        I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
}

else if(direction == I2C_Direction_Receiver)
{
    while(!I2C_CheckEvent(I2Cx,
        I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
}
}

```

```

void I2C_write(I2C_TypeDef* I2Cx, uint8_t data)
{
    I2C_SendData(I2Cx, data);

    /* wait for I2C1 EV8_2 --> byte has been transmitted */
    while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
}

uint8_t I2C_read_ack(I2C_TypeDef* I2Cx)
{
    /* enable acknowledge of received data */
    I2C_AcknowledgeConfig(I2Cx, ENABLE);

    // wait until one byte has been received
    while( !I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED) );

    // read data from I2C data register and return data byte uint8_t
    data = I2C_ReceiveData(I2Cx);

    return data;
}

uint8_t I2C_read_nack(I2C_TypeDef* I2Cx)
{
    // disable acknowledge of received data

    // nack also generates stop condition after last byte received

    // see reference manual for more info

```

```

I2C_AcknowledgeConfig(I2Cx,DISABLE);

I2C_GenerateSTOP(I2Cx, ENABLE);

// wait until one byte has been received
while( !I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED) );

    // read data from I2C data register and return data byte uint8_t

data = I2C_ReceiveData(I2Cx);

return data;
}
void I2C_stop(I2C_TypeDef* I2Cx)
{
    // Send I2C1 STOP Condition
    I2C_GenerateSTOP(I2Cx, ENABLE);
}

```

Result

Thus the C-Language program for reading RTC and displaying it in PC was written & output is verified with running the RTC from a default/specified time.

Program

```
#include "stm32f4xx.h"

#include "stm32f4xx_rcc.h"

#include "stm32f4xx_gpio.h" static int

i=0;

void anticlock_wise(); void
clock_wise(); void delay()
{
    for(int i=0;i<0xFF;i++)
        for(int j=0;j<0xFF;j++);
}

void mx_pinout_config(void)
{
    RCC->AHB1ENR |= 1<<1;

    GPIOB->MODER = 0x55555555; GPIOB-
    >OTYPER =0x00000000; GPIOB->PUPDR
    = 0x00000000; GPIOB->OSPEEDR =
    0xFFFFFFFF;
}
```

Ex.No.6**Stepper Motor****Aim**

To write C Programs for running stepper motor either in clock-wise or counter-clock- wise and the direction of the rotation of the stepper motor depends on the variation in the temperature sensor.

Apparatus & Software Required

1. VSK-SCM4 Development board.
2. IAR IDE software.
3. Flash Loader Demonstrator. .
4. Stepper motor

Theory

- ✓ Stepper motors, effectively have multiple "toothed" electromagnets arranged around a central metal gear.
- ✓ To make the motor shaft turn, first one electromagnet is given power, which makes the gear's teeth magnetically attracted to the electromagnet's teeth.
- ✓ When the gear's teeth are thus aligned to the first electromagnet, they are slightly offset from the next electromagnet.
- ✓ So when the next electromagnet is turned on and the first will turn off, the gear rotates slightly to align with the next one and from there the process is repeated.
- ✓ Each of those slight rotations is called a "step."
- ✓ In that way, the motor can be turned to a précised angle.
- ✓ There are two basic arrangements for the electromagnetic coils: bipolar and unipolar.

```

int main(void)
{
    mx_pinout_config();

    while (1)
    {
        anticlock_wise(); delay();

        clock_wise(); delay();

    }
}

void anticlock_wise()
{
    for(i=0;i<52;i++)
    {
        GPIOB->ODR = 0xA000;

        delay(); delay(); delay();delay(); delay(); GPIOB-
        >ODR = 0x6000;

        delay(); delay(); delay();delay(); delay(); GPIOB-
        >ODR = 0x5000;

        delay(); delay(); delay();delay(); delay(); GPIOB-
        >ODR = 0x9000;

        delay(); delay(); delay();delay(); delay();

    }
}

void clock_wise()

```

Procedure

- ✓ Set the current date and time in the PC
- ✓ Open the **IAR Embedded work bench** from the desktop

1. Create the project:

- ✓ Select the **project** from the Menu bar
- ✓ Select **create new project** from the project menu bar.
- ✓ Click **empty project**
- ✓ Click **ok**
- ✓ Create a new folder in the 'C' drive and give the name of the folder.
- ✓ Save the project in the created folder
- ✓ Then click **ok**
- ✓ Copy **INC** folder and supporting file **STM_lib.a** from main folder and paste it in to created folder

2. Create the file:

- ✓ Select the **file** from the menu bar.
- ✓ Click **new** and the select **file**
- ✓ Type the program
- ✓ Save the program with the extension of (.c).

3. Add the Source file & Supporting file:

- ✓ Right click on project name
- ✓ Select **Add**
- ✓ Select file name.c
- ✓ Again, right click on project name
- ✓ Then click **add files** to select a STM.lib.a (library files)

4. Setting Options:

- ✓ Enable created project
- ✓ Click **project** from menu bar to select the **options**
- ✓ Click **general options** to select **Device**
- ✓ Select ST→STM32F407→ STM32F407VG.
- ✓ Click **C\C++ compiler** and select the **preprocessor** from menu and then type **\$PROJ_DIR\$\INC.**

{

```
for(i=0;i<52;i++)  
{  
    GPIOB->ODR = 0x9000;  
    delay(); delay(); delay();delay(); delay(); GPIOB-  
        >ODR = 0x5000;  
    delay(); delay(); delay();delay(); delay(); GPIOB-  
        >ODR = 0x6000;  
    delay(); delay(); delay();delay(); delay(); GPIOB-  
        >ODR = 0xA000;  
    delay(); delay(); delay();delay(); delay();  
}  
}
```

- ✓ Click **output converter** to enable the **generate** and select **intel extended**.
- ✓ Click **linker** and enable **override default**.
- ✓ Click **debugger** and select **ST- LINK** and click **download** from menu and enable **user flash loader**.
- ✓ Click **ST-LINK** and enable **SWD**.
- ✓ Finally click OK.

5. Compile the program:

- ✓ Click **project** and select **rebuild all**.
- ✓ Click **project** and select download and click **download active application**.

6. Output verification:

- ✓ Open the **STM32ST link utility** from the desktop.
- ✓ Click **target** from menu bar and select **connect**
- ✓ Again then click **target** and to select **program and verify**.
- ✓ Click **debug** and to select the **.hex file** from the **Exe** folder
- ✓ The stepper motor will start rotating.

Result

The C-Language program to Stepper Motor is developed and is verified.

Program

```
#include "stm32f4xx.h"

#include <stdio.h>

void USART2_config(); void
send_data(char data); char gk[8] =
"VSK-SCM4"; void delay(int j)
{
    int g,k; for(g=0;g<j;g++)
        for(k=0;k<j;k++);
}

void main()
{
    int i; USART2_config();
    while(1)
    {
        for(i=0;gk[i] != '\0';i++)
        {
            send_data(gk[i]);
        } send_data(0x0d);
        send_data(0x0A);
    }
}

void USART2_config()
{
    RCC->AHB1ENR |= 1 << 0; RCC-
    >APB1ENR |= 1 <<17;
```

Ex.No.7**Implementing Zigbee Protocol with ARM****Aim**

To transmit the data via Xbee and view the output data in the PC.

Apparatus & Software Required

1. VSK – STM4 syllabus board.
2. IAR IDE software.
3. Flash Loader Demonstrator.
4. Xbee modules 2 Nos.

Theory

- ✓ The X Bee/X Bee-PRO ZNet 2.5 (formerly known as Series 2 and Series 2 PRO) RF Modules were directed to operate within the Zig Bee protocol.
- ✓ The modules provide reliable delivery of data between remote devices.
- ✓ Zigbee is the communication protocol like wifi and Bluetooth.
- ✓ Xbee is the module using Zigbee protocol

Some of its features are:

- Zigbee is targeted at radio-frequency (RF) applications
- Low data rate, long battery life, and secure networking
- Transmission range is between 10 and 75 meters (33~246 feet)
- Using local addressing, simple networks of more than 65,000 nodes can be configured, with reduced address overhead
- The radios use direct-sequence spread spectrum coding, which is managed by the digital stream into the modulator.
- To ensure reliable data transmission.
- Binary phase shift keying (BPSK) in the 868/915 MHz
- Offset quadrature phase shift keying (O-QPSK) at 2.4 GHz

```
GPIOA->MODER |= 0x000000A0; GPIOA-  
>AFR[0] |= 0x00007700;
```

```
USART2->BRR = 0x1117; USART2-  
>CR3 = 0x0000; USART2->CR2 =  
0x000; USART2->CR1 = 0x200C;
```

```
}  
void send_data(char data)  
{  
    USART2->DR = (data & 0x01FF);  
    /* Loop until the end of transmission */  
    while((USART2->SR & 0x40) == 0)  
    {}  
}
```

```
int putchar(int data)  
{  
    USART2->DR = (data & 0x01FF);  
    /* Loop until the end of transmission */  
    while((USART2->SR & 0x40) == 0)  
    {}  
    return data;  
}
```

Procedure

- ✓ Set the current date and time in the PC
- ✓ Open the **IAR Embedded work bench** from the desktop

1. Create the project:

- ✓ Select the **project** from the Menu bar
- ✓ Select **create new project** from the project menu bar.
- ✓ Click **empty project**
- ✓ Click **ok**
- ✓ Create a new folder in the 'C' drive and give the name of the folder.
- ✓ Save the project in the created folder
- ✓ Then click **ok**
- ✓ Copy **INC** folder and supporting file **STM_lib.a** from main folder and paste it in to created folder

2. Create the file:

- ✓ Select the **file** from the menu bar.
- ✓ Click **new** and the select **file**
- ✓ Type the program
- ✓ Save the program with the extension of (.c).

3. Add the Source file & Supporting file:

- ✓ Right click on project name
- ✓ Select **Add**
- ✓ Select file name.c
- ✓ Again, right click on project name
- ✓ Then click **add files** to select a STM.lib.a (library files)

4. Setting Options:

- ✓ Enable created project
- ✓ Click **project** from menu bar to select the **options**
- ✓ Click **general options** to select **Device**
- ✓ Select ST→STM32F407→ STM32F407VG.
- ✓ Click **C\C++ compiler** and select the **preprocessor** from menu and then type **\$PROJ_DIR\$\INC.**
- ✓ Click **output converter** to enable the **generate** and select **intel extended.**
- ✓ Click **linker** and enable **override default.**
- ✓ Click **debugger** and select **ST- LINK** and click **download** from menu and enable **user flash loader.**

- ✓ Click **ST-LINK** and enable **SWD**.
- ✓ Finally click OK.

5. Compile the program:

- ✓ Click **project** and select **rebuild all**.
- ✓ Click **project** and select download and click **download active application**.

6. Output verification:

- ✓ Open the **STM32ST link utility** from the desktop.
- ✓ Click **target** from menu bar and select **connect**
- ✓ Again then click **target** and to select **program and verify**.
- ✓ Click **debug** and to select the **.hex file** from the **Exe** folder
- ✓ Connect 1 Xbee module with VSK-STM4 board and the other module to the PC.
- ✓ Open Win X Talk with a baud rate of 115200.
- ✓ Now change the Mode-Selection-Switch to **EXEC** position and reset the board.
- ✓ The data that is transmitted from the controller can be viewed in PC.

Result

The C-Language program to implement Zigbee Protocol with ARM is developed and is verified.

Program:

```
#include "stm32f4xx.h"

#define BUZZER_ON ((GPIOC->ODR = 0x01))
#define BUZZER_OFF ((GPIOC->ODR = 0x00))

void buzzer_init()
{
    RCC->AHB1ENR |= 1 << 2; //enable clock to PORT-C
    GPIOC->MODER = 0x00000001; //PC0 as o/p mode
    GPIOC->OTYPER = 0x00000000; //push-pull type
    GPIOC->PUPDR = 0x00000000; //no pull-up,pull-down
    GPIOC->OSPEEDR = 0xAAAAAAAA; //speed is 50MHZ
}

void delay(long int delay)
{
    long int d1,d2;
    for(d1=delay;d1>0;d1--)
        for(d2=delay;d2>0;d2--);
}

void main()
{
    buzzer_init();
    while(1)
    {
        BUZZER_ON;
        delay(4000);
        BUZZER_OFF;
        delay(4000);
    }
}
```

Aim

To develop a 'C' program to make the buzzer on and upon change in the delay program the sound of the buzzer should vary.

Apparatus & Software Required

1. VSK-SCM4 Development board.
2. IAR IDE software.
3. Flash Loader Demonstrator.

Procedure

- ✓ Set the current date and time in the PC
- ✓ Open the **IAR Embedded work bench** from the desktop

1. Create the project:

- ✓ Select the **project** from the Menu bar
- ✓ Select **create new project** from the project menu bar.
- ✓ Click **empty project**
- ✓ Click **ok**
- ✓ Create a new folder in the 'C' drive and give the name of the folder.
- ✓ Save the project in the created folder
- ✓ Then click **ok**
- ✓ Copy **INC** folder and supporting file **STM_lib.a** from main folder and paste it in to created folder

2. Create the file:

- ✓ Select the **file** from the menu bar.
- ✓ Click **new** and the select **file**
- ✓ Type the program
- ✓ Save the program with the extension of (.c).

3. Add the Source file & Supporting file:

- ✓ Right click on project name
- ✓ Select **Add**
- ✓ Select file name.c
- ✓ Again, right click on project name
- ✓ Then click **add files** to select a STM.lib.a (library files)

4. Setting Options:

- ✓ Enable created project
- ✓ Click **project** from menu bar to select the **options**
- ✓ Click **general options** to select **Device**
- ✓ Select ST→STM32F407→ STM32F407VG.
- ✓ Click **C\C++ compiler** and select the **preprocessor** from menu and then type **\$PROJ_DIR\$\INC.**
- ✓ Click **output converter** to enable the **generate** and select **intel extended.**
- ✓ Click **linker** and enable **override default.**
- ✓ Click **debugger** and select **ST- LINK** and click **download** from menu and enable **user flash loader.**
- ✓ Click **ST-LINK** and enable **SWD.**
- ✓ Finally click OK.

5. Compile the program:

- ✓ Click **project** and select **rebuild all.**
- ✓ Click **project** and select download and click **download active application.**

6. Output verification:

- ✓ Open the **STM32ST link utility** from the desktop.
- ✓ Click **target** from menu bar and select **connect**
- ✓ Again then click **target** and to select **program and verify.**
- ✓ Click **debug** and to select the **.hex file** from the **Exe** folder
- ✓ The sound comes from buzzer.

Result:

The C-Language program to make the buzzer on was developed and output was verified and upon change in the delay program the sound variation was verified.