**SYSTICK TIMER**

--------------------------------------------------------------------------------

#include "stm32f412Zx.h"

#include <stdio.h>

void delay(int dd);

int main() {

RCC->AHB1ENR |= 0x02;

GPIOB->MODER |= 0x10004000;

GPIOB->ODR = 0x4000;

SysTick -> LOAD = 16000000;

SysTick -> VAL = 0;

SysTick -> CTRL = 0x5;

int c = 0;

while (1) {

if (SysTick -> CTRL & 0x00010000){

c++;

if (c % 10 == 0){

GPIOB -> ODR ^= 0x4080;

}

}

}

}

----------------------------------------------------------------------------

**ADC**

#include "stm32f412Zx.h"

int adcvalue; // Defining variable to watch ADC values

int main() {

RCC->APB2ENR |= 0x100 ; // Enable clock for ADC1

RCC->AHB1ENR |= 0x1 ; // Enable clock for GPIOA- PA0 is internally connected to IN 0 )

ADC1->CR2 |= 0x2 ; // Enable continuous conversion mode

ADC1->CR2 |= 0x400 ; // EOC after each conversion

ADC1->CR2 |= 0x1 ; // ADON =1 enable ADC1

ADC1->SQR3 |= 0 ; // Conversion in regular sequence

GPIOA->MODER |= 0x3 ; // Analog mode for PA0

GPIOB->MODER |= 0x4000;

GPIOB->ODR=0x0;

while ( 1 ) {

ADC1->SR = 0 ; // Clear the status register

ADC1->CR2 |= (0x40000000) ; // Starting conversion by SWSTART

while (ADC1->SR & (0x2)) { // Check until conversion completes

adcvalue = ADC1->DR; // Update ADC values to variable defined

if ((double)adcvalue\*3.3/4095 < 1.78) { // If voltage drops below 1.78V

GPIOB->ODR = 0x80;

}

else {

GPIOB->ODR = 0x0; // Turn off LED

}

}

}

}

-------------------------------------------------------------------------------------------------------------

**USART**

#include <stm32f412zx.h>

#include <stdio.h>

int main(){

RCC->AHB1ENR |= 0x10100; // Enable clock for GPIOC & GPIOE

GPIOC->MODER |= 0x2; // Output mode

GPIOC->ODR |= 0x1; // Set the output register to 1

// Set timer using TIM2 --> 10us --> 10^5 Hz = 16 MHz/(0+1)(159+1)

// This timer will note the time at the start of process

USART1->CR1 |= 0x4;

USART1->CR2 |= 0x0;

USART1->CR1 |= (1<<15);

USART1->CR2 = 0x0;

TIM2->ARR = 0;

TIM2->PSC = 159;

TIM2->CNT = 0;

TIM2->CR1 |= 0x11; // start counting

GPIOE->ODR = 0x010; // turn Port E to Output to show LED

int t1;

while(1){

if(TIM2->SR & 1){

TIM2->SR = 0;

t1 = TIM2->CCR1;

break;

}

}

// Set timer using TIM3 --> 10us --> 10^5 Hz = 16 MHz/(0+1)(159+1)

// This timer will note the time at the end of process

int t2;

RCC->APB1ENR = 0x01;

TIM3->PSC = 1599;

TIM3->ARR = 0;

TIM3->CNT = 0;

while(1){

if (TIM3->SR & 1){

TIM3->SR = 0;

break;

}

}

**POTENTIOMETER ADC**

#include <stm32f412zx.h>

#include <stdio.h>

// POTENTIOMETER EXPERIMENT - Use of ADC

// variable to temporarily store and read the adc readings

void delay(int dd){

for(int i=0; i<dd; i++){

for(int j=0; j<dd; j++){

}

}

}

int main(){

//int adc\_value;

RCC->APB2ENR |= 0x100 ; // Enable clock for ADC1

RCC->AHB1ENR |= 0x1 ; // Enable clock for GPIOA- PA0 is internally connected to IN 0 )

RCC->AHB1ENR |=0x2;

ADC1->CR2 |= 0x2 ; // Enable continuous conversion mode

ADC1->CR2 |= 0x400 ; // EOC after each conversion

ADC1->CR2 |= 0x1 ; // ADON =1 enable ADC1

ADC1->SQR3 |= 0 ; // Conversion in regular sequence

GPIOA->MODER |= 0x3 ; // Analog mode for PA0

GPIOB->MODER |= 0x4000;

GPIOB->ODR=0x0;

int x = 0 ;

while(1){

x++;

ADC1->SR = 0; // Status register - cleared

ADC1->CR2 |= 0x40000000; // Control register -> 30th bit SWSTART = 1 --> Conversion takes place

if (ADC1->SR & 0x02){

float adc\_value = ADC1->DR; // store the value from Data Register to the variable.

x=0;

// Quantising the reading

float reading = 0;

reading = adc\_value\*3.3/4095; // 12 bit register -> 4096

if(reading < 1.78){

GPIOB->ODR = 0x0080; // Output reg for LED 7

delay(100);

}

else{

GPIOB->ODR = 0x0000;

}

}

}

}

--------------------------------------------------------------------------------------------------

**ULTRASONIC SENSOR**

#include "stm32f4xx.h"

#include "uart.h"

#include <stdio.h>

#define ARM\_MATH\_CM4

void GPIO\_Init(void);

void TIM2\_us\_Delay(uint32\_t delay); //TIM2 for generating 10us pulse for trig pin

void GPIO\_Init(){

//Configuring PA5 for generating pulse sent to trig pin

RCC->AHB1ENR |= 1; //Enable GPIOA clock

GPIOA->MODER |= 1<<10; //Set the PA5 pin to output mode

//Configuring output from echo pin to be sent to the board (PA6 pin)

GPIOA->MODER &= ~(0x00003000); //Set PA6 to input mode

GPIOA->ODR = 0x00000000; // Setting trig pin to low to initialize the module

}

void TIM2\_us\_Delay(uint32\_t delay){

}

int main(){

UART3Config();

UART3\_SendString ("hello world12345\r\n");

GPIO\_Init();

int time\_diff;

int pulsewidth;

int toggle = 0;

int val1; // store initial value of count register

int val2; // store final value of count register

while(1){

GPIOA->ODR |= 1<<7; // keep output pin high --> sound emitted

RCC->APB1ENR |=1; //Start the clock for the timer peripheral

// delay = 10 us --> f = 1/(10^-5) = 10^5 Hz = 0.1MHz freq = 16 MHz PSC = 0 --> PSC+1 = 1

// 0.1 = 16/(ARR+1) --> ARR = 159, PSC = 0

TIM2->ARR = 159;

TIM2->PSC = 0;

TIM2->CNT = 0;

TIM2->CR1 |= 1; //Start the Timer

while(!(TIM2->SR & TIM\_SR\_UIF)){ //Polling the update interrupt flag

if(!(TIM2->CNT & TIM2->ARR)){

TIM2->CNT += 1;

}

}

TIM2->SR &= ~(0x0001); //Reset the update interrupt flag

if((GPIOA->IDR == 1<<6) && (toggle==0)){

toggle = 1;

val1 = TIM2->CNT;

}

if((GPIOA->IDR == 0) && (toggle==1)){

toggle = 0;

val2 = TIM2->CNT;

}

time\_diff = val2 - val1;

pulsewidth = time\_diff/58;

printf("%d\n",pulsewidth);

// fprintf("%d\n",pulsewidth);

}

}

----------------------------------------------------------------------------------------------------------------------------------

**7 SEGMENT DISPLAY**

#include <stm32f412zx.h>

void delay(int dd){

for(int i=0;i<dd;++i){

for(int j=0;j<dd;++j){

}

}

}

int main(){

RCC->AHB1ENR |= 1; // enable clock

GPIOA->MODER = 0x555555555; // configuring 7 segments i.e. 7 GPIOA registers to output mode 01 => 01/0101/0101/0101/0000

GPIOA->ODR = 0xFFF; // ODR = 1 1 1 1 / 1 1 1 1 / 1 1 1 0

// PA8 PA7 PA6 PA5 PA4 PA3 PA2 PA1 PA0

// 0 -- ABCDEF -- PA1, PA2, PA3, PA4, PA5, PA6 --> 0 0111 1110 --> 0x07E

// 1 -- B,C = 1 --> PA2, PA3 --> 0/0000/1100 --> 0X0C

// 2 -- ABGED --> PA1, PA2, PA4, PA5, PA7 --> 0 1011 0110 --> 0x0B6

// 3 -- ABGCD --> PA1, PA2, PA3, PA4, PA7 --> 0 1001 1110 --> 0x09E

// 4 -- FGBC --> PA2 PA3, PA6, PA7 --> 0 1100 1100 -> 0x0CC

// 5 -- AFGCD --> PA1 PA3 PA4 PA6 PA7 --> 0 1101 1010 --> 0X0DA

// 6 -- AFGECD --> PA1 PA3 PA4 PA5 PA6 PA7 --> 0 1111 1010 --> 0x0FA

// 7 -- ABC --> PA1 PA2 PA3 --> 0 0000 1110 -> 0X00E

// 8 -- ABCDEFG -->

// 9 -- ABCDFG --> 0 1101 1110 --> oxoDE

while(1){

GPIOA->ODR = 0x07E;

delay(1000);

GPIOA->ODR = 0x00C;

delay(1000);

GPIOA->ODR = 0x0B6;

delay(1000);

GPIOA->ODR = 0x09E;

delay(1000);

GPIOA->ODR = 0x0CC;

delay(1000);

GPIOA->ODR = 0x0DA;

delay(1000);

GPIOA->ODR = 0x0FA;

delay(1000);

GPIOA->ODR = 0x00E;

delay(1000);

GPIOA->ODR = 0XFFF;

delay(1000);

GPIOA->ODR = 0x0DE;

delay(1000);

}

return 0;

}

------------------------------------------------------------------------------------------------------

**INTERRUPT**

#include <stm32f412zx.h>

#include <stdio.h>

void delay(uint32\_t delay){

RCC->APB1ENR |=1; // Start the clock for the timer peripheral

TIM2->PSC = 100;

TIM2->ARR = (160000\*delay)-1; // delay in microsecond 160000-1 = 159999

TIM2->CNT = 0;

TIM2->CR1 |= 1; //Start the Timer

while(!(TIM2->SR & TIM\_SR\_UIF)){} //Polling the update interrupt flag

TIM2->SR &= ~(0x0001); //Reset the update interrupt flag

}

void EXTI15\_10\_IRQHandler(void) {

GPIOB->ODR = 0x00;

delay(1);

GPIOB->ODR = 0x80;

delay(1);

GPIOB->ODR = 0x0;

delay(1);

GPIOB->ODR = 0x80;

delay(1);

GPIOB->ODR = 0x0;

delay(1);

GPIOB->ODR = 0x80;

EXTI->PR = 0x2000;

}

int main(){

\_\_disable\_irq( );

// Interrupt Request Handler

RCC->AHB1ENR |= 0x02 ;

RCC->AHB1ENR |= 0x4 ;

RCC->APB2ENR |= 0X4000 ;

GPIOB->MODER |= 0x00004000 ;

SYSCFG->EXTICR[3] = 0X0020 ;

EXTI->IMR |= 0x2000 ;

EXTI->RTSR |= 0x2000 ;

// Once handling done, enable interrupt requests again

NVIC\_EnableIRQ (EXTI15\_10\_IRQn) ;

\_\_enable\_irq( ) ;

while(1) { }

}