PROJECT 3
(UTPIDPWMOC)

Kunal Mukherjee

UNIVERSITY OF EVANSVILLE

STATEMENT OF THE PROJECT

The project is supposed to give the students expertise with PID controller, serial communication, USART, and LCD display. A PID controller is “Proportional Integral Derivative” controller. A PID controller gives an output depending on the error of the system as well as certain parameter such as predicting what error is going to come. For example, if the error is more than the output signal will be large and vice-versa. The LCD display used USART serial communication to communicate with the nucleo board.

This project first asks the user to map the position of the pivot arm somehow, here we use a potentiometer to measure the angle or the position of the drone motor. Then, the project uses a PWM signal to control the DC drone motor to increase or decrease thrust. The ultimate objective of the project is that the drone is supposed to hover at a target angle. The “error” in the angle, will be utilized by the PID controller to change the PWM signal, so that the drone can hover over the desired angle.

POSSIBLE UTILITY OF THE PROJECT

The physical project maybe of just practice utility, but the PID algorithm used can be readily used for other project requiring a PID. While doing this project I had to use my knowledge of ADC, PWM and PID controllers. So, I had practice. After this project, I think I should have made a drone as my senior project.

IDENTIFICATION OF SPECIFICATION

Design specification are as follows:

- Create a basic PID controller with externally adjustable coefficient
- Method for externally triggering a capture of the target position
- Demonstrate it can control a physical system like the 1-D helicopter
- Target position can be set in software
- Some method of display for the PID coefficients

IDENTIFICATION OF DESIGN ISSUES AND SOLUTION

The first design issue that was presented on how to use a 3.3V PWM signal from the nucleo board to control a DC drone motor. The DC motor needs at least 10 V and current up to 1 A. Therefore, to solve this issue, I used a H-bridge to drive the motor. H-bride increased the response time of the motor but, it does not affect the system drastically.

The second design issue was how to display coefficient. I used a 16x2 LCD display and used USART to communicate with it. The USART takes a noticeable amount of time, so I update the display only if the values change.

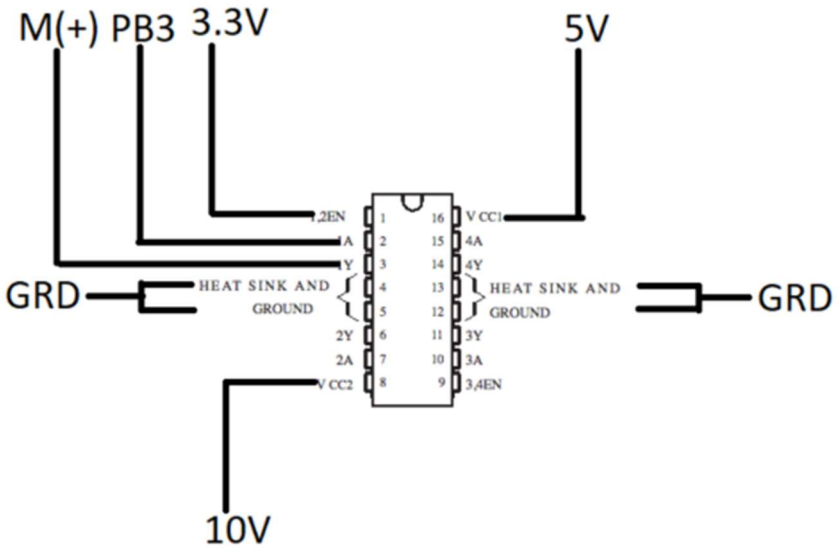
The third design issue of how to quantify position of the drone arm, but it was already solved by the professor as the arm was attached to a potentiometer. The voltage of the potentiometer gave a quantification of the location or the angle at which the arm should over.

The PID controller give an output value that corresponds to an ADC value. The fourth design issue was how to use that output value and convert it to a PWM signal that will change the speed of the propeller. It was solved by using clever mathematical tools. First, finding the percentage of the ADC value compared the max range of ADC value ($2^{12} = 4095$), then multiplying by the max time width of the PWM signal. For example, PWM = (Output / 4096) *

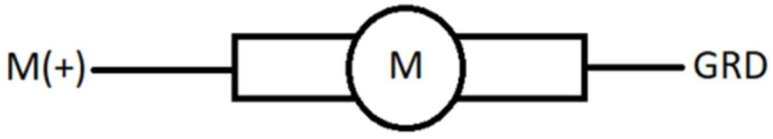
MAX_ARR_PWM Then, the output PWM signal will be of the same percentage of the max time period of the PWM, as the output ADC is of the max ADC value.

The final design issue was how to find the correct constant values. Dr. Mitchell suggested that to first turn all the constants to 0, except for the proportionality constant. Increase the proportionality constant slowly until, I get a sinusoidal motion on the drone. Once, I get the motion, then increasing the derivate constant, so that the arm comes to the position without overshoot or undershooting. That was the proper way to get the constant values.

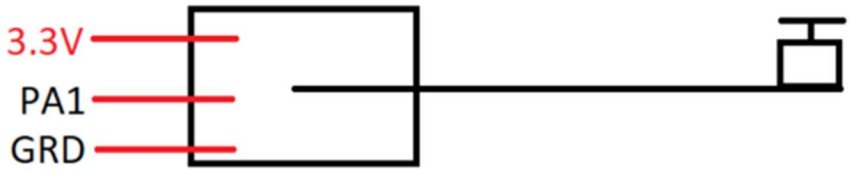
SCHEMATIC OF COMPONENTS EXTERNAL TO THE STM32F446 BOARD



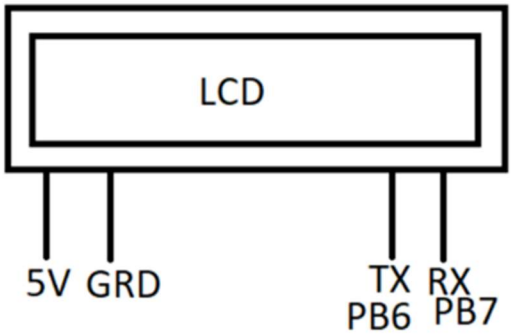
H-bridge Connection



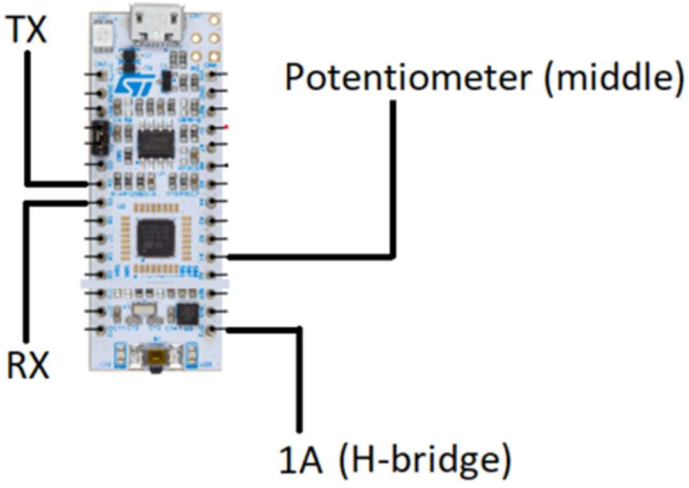
Motor Connection



Potentiometer Connection



LCD Connection



Nucleo Board Connection

INFORMATIONAL RESOURCES USED

- Youtube board showing PID tuning
  https://www.youtube.com/watch?v=0t3JL_zGEYY

```c
//Kunal Mukherjee;
//4/24/2019
//Project 3: UTPIDPWMOC

#include "stm32l432.h"
#include "string.h"
#include "stdint.h"
#include "stdio.h"
#include "stdlib.h"
#include "time.h"

//GPIO Initialize
void GPIO_Init(void);

//USART
void USART1_Initialize(void);

void USART1_Read(char *buffer, int bytes);

void USART1_Write(char *buffer, int bytes);

//USART functions
void USART1_ClearScreen(void);

void USART1_WriteValue(float Kp,float Ki,float Kd);

//ADC
void ADC_Init(void);

//PWM Timer
void PWM_Init(void);

#define MAX_BYTE_VAL 32
#define WAIT_TIME 500000

#define TARGET 1577
#define MAX_ADC_VALUE 4096
#define MAX_ARR_VALUE 4000

#define KP 1.25  //2.4//2.3//1.2
#define KI 0
#define KD -0.175 //0.123 //0.69 //0.5
#define T 10

#define DELAY 5000
#define DELAY2 100
#define DELAY3 5000

int main()

{
  int i,j;
  float Kp = 0, Kd = 0, Ki = 0;
  float derivative = 0,  integral = 0;
  int position = 0, target = 0, error = 0, error_old = 0;
  float output = 0, tmr2 = 0;
  int adcCaptureFlag = 0, kValueChangeFlag = 0;
  int Kselection;

  target = TARGET;
  Kp = KP;
  Ki = KI;
  Kd = KD;

  GPIO_Init();
  ADC_Init();
  PWM_Init();
  USART1_Initialize();

  USART1_WriteValue(Kp, Ki, Kd);

  while(!adcCaptureFlag)
```

```
  73      {
  74        if ((GPIOA_IDR & (1 << 8)) == 0)
  75        {
  76          ADC_CR |= (1 << 2); //start adc regular conversion
  77          while((ADC_ISR & (1 << 2)) == 0);//look at the EOC flag
  78          target = ADC_DR & 0xFFF;
  79
  80          adcCaptureFlag = 1;
  81        }
  82
  83        for (i = 0; i < DELAY3; i++);
  84      }
  85
  86      while(1)
  87      {
  88
  89        ADC_CR |= (1 << 2); //start adc regular conversion
  90        while((ADC_ISR & (1 << 2)) == 0);//look at the EOC flag
  91        position = ADC_DR & 0xFFF;
  92
  93        error_old = error;
  94        error = target - position;
  95        integral += error;
  96
  97        derivative = error - error_old;
  98
  99        output = output + (Kp * error) + (Ki * ( T * integral)) + ((Kd/T) * derivative);
 100        //output =  output + Kp * error + (Kd/T) * derivative;
 101
 102        tmr2 = ((float) ((float) output / (float) MAX_ADC_VALUE) * (float) MAX_ARR_VALUE);
 103
 104        //tmr2 = MAX_ARR_VALUE;
 105
 106        if (tmr2 < 0)
 107        {
 108          tmr2 = 0;
 109        }
 110        if (tmr2 > MAX_ARR_VALUE)
 111        {
 112          tmr2 = MAX_ARR_VALUE;
 113        }
 114
 115        TIM2_CCR2 = tmr2;
 116
 117        //if constant value is change write it to the screen
 118        if (kValueChangeFlag == 1)
 119        {
 120          USART1_WriteValue(Kp, Ki, Kd);
 121          kValueChangeFlag = 0;
 122        }
 123
 124        //use PA# and PA# to control which constant to select
 125        if ((GPIOA_IDR & (1 << 4)) == 0)
 126        {
 127          Kselection++;
 128
 129          if(Kselection > 2)
 130          {
 131            Kselection = 2;
 132          }
 133          for (i = 0; i < DELAY3; i++);
 134        }
 135        if ((GPIOA_IDR & (1 << 5)) == 0)
 136        {
 137          Kselection--;
 138
 139          if(Kselection < 0)
 140          {
 141            Kselection = 0;
 142          }
 143          for (i = 0; i < DELAY3; i++);
 144        }
```

```
145
146       //use PA# and PA# to increase or decr a const
147       if ((GPIOA_IDR & (1 << 3)) == 0)
148       {
149         if(Kselection == 0)
150         {
151           Kp += .10;
152         }
153         if(Kselection == 1)
154         {
155           Ki += .10;
156         }
157         if(Kselection == 2)
158         {
159           Kd += .10;
160         }
161
162         kValueChangeFlag = 1;
163
164         for (i = 0; i < DELAY3; i++);
165       }
166
167       if ((GPIOA_IDR & (1 << 7)) == 0)
168       {
169         if(Kselection == 0)
170         {
171           Kp -= .10;
172         }
173         if(Kselection == 1)
174         {
175           Ki -= .10;
176         }
177         if(Kselection == 2)
178         {
179           Kd -= .10;
180         }
181
182         kValueChangeFlag = 1;
183
184         for (i = 0; i < DELAY3; i++);
185       }
186
187
188       for (i =0; i < DELAY; i++)
189       {
190         for (j =0; j < DELAY2; j++);
191       }
192     }
193
194   }
195
196   void GPIO_Init(void)
197   {
198     RCC_AHB2ENR  |= (1 << 0);     //GPIOA clk enable
199     RCC_AHB2ENR  |= (1 << 1);     //GPIOB clock enable bit
200     RCC_APB2ENR |= 1 << 14;       // USART1 Enable
201
202     //enable pb3 for debug
203     //GPIOB_MODER &= ~(3 << (2 * 3));
204     //GPIOB_MODER |=  (1 << (2 * 3)); //PB3 Output
205
206
207     // enable GPIO pin and configure the TX pin and the Rx pin as:
208     // ALternate function, high speed, push-pull, pull-up
209     // USART1  PB6 = TX and PB.7 = RX
210     GPIOB_MODER   &= ~(0xF << (2*6)); // clr PB6 AND 7
211     GPIOB_MODER   |=  (0xA << (2*6)); // Altfunc PB6 AND 7
212
213     // Alternate function 7 = Usart1
214     // Appendix I shows all alternate functions
215     GPIOB_AFRL    |=   (0x77 << (4*6)); // set pB6 and 7 to AF 7
216     GPIOB_OSPEEDR |=   (0xF  << (2*6)); // HIGH SPEED ON PB6 AND 7
```

```
217     GPIOB_PUPDR   &= ~((0xF) << (2*6));
218     GPIOB_PUPDR   |=   (0x5  << (2*6)); // pull up on PB6 and 7
219     GPIOB_OTYPER  &=  ~(0x3  <<  6); //PB6 and 7 open drain
220
221     //PA8 ADC_CAPTURE
222     GPIOA_MODER &= ~(3 << (2 * 8));
223     GPIOA_PUPDR |=  (2 << (2 * 8));
224
225     GPIOA_MODER &= ~(3 << (2 * 4));
226     GPIOA_PUPDR |=  (2 << (2 * 4));
227
228     GPIOA_MODER &= ~(3 << (2 * 5));
229     GPIOA_PUPDR |=  (2 << (2 * 5));
230
231     GPIOA_MODER &= ~(3 << (2 * 3));
232     GPIOA_PUPDR |=  (2 << (2 * 3));
233
234     GPIOA_MODER &= ~(3 << (2 * 7));
235     GPIOA_PUPDR |=  (2 << (2 * 7));
236   }
237
238   void USART1_ClearScreen()
239   {
240     char pData[2] = "|-";
241
242     USART1_Write(pData,2);
243   }
244
245   void USART1_WriteValue(float Kp,float Ki,float Kd)
246   {
247     USART1_ClearScreen();
248
249     int i;
250
251     char pDataKp[5], pDataKi[5], pDataKd[5];
252
253     sprintf(pDataKp, "%.2f", Kp);
254     sprintf(pDataKi, "%.2f", Ki);
255     sprintf(pDataKd, "%.2f", Kd);
256
257     USART1_Write(pDataKp,5);
258     USART1_Write(pDataKi,5);
259     USART1_Write(pDataKd,5);
260   }
261
262   // USART1 initialize
263   void USART1_Initialize()
264   {
265     USART1_CR1 &= ~(1<<0); // DISABLE USART
266
267     // SET DATA LENGTH TO 8 BITS
268     // 00 = 8 DATA BITS, 01 = 9 DATA BITS ,10 = 7 DATA BITS
269     USART1_CR1 &= ~ (1 << 12); //M1
270     USART1_CR1 &= ~(1 << 28); // M0
271
272     // SELECT1 STOP BIT
273     // 00 = 1 STOPBIT 01 = 0.5 STOP BIT
274     // 10 = 2 STOPBITS 11 = 1.5 STOP BIT
275     USART1_CR2 &= ~(0x3 << 12);
276
277     // SET PARITY CONTROL AS NO PARITY
278     //0 = NO PARITY
279     // 1 = PARITY ENABLE (THEN PROGRAM PS BIT TO SELECT EVEN OR ODD PARITY)
280     USART1_CR1 &= ~(1 << 10);
281
282     // OVERSAMPLING BY 16
283     // 0 = OVERSAMPLING BY 16, 1 = OVERSAMPLING BY 8
284     USART1_CR1 &= ~(1 << 15);
285
286     // SET BAUD RATE TO 9600 USING APB FREQUENCY (80 MHZ)
287     // SEE EXAMPLE 1 IN SECTION 22.1.2
288     USART1_BRR = 0X1A1;
```

```
289
290     // ENABLE USART
291     USART1_CR1 |= (USART_CR1_TE | USART_CR1_RE); //  transmitter and reciever
292
293     //  ENABLE USART
294     USART1_CR1 |= 1<<0;
295
296     // VERIFY THAT USART IS READY FOR TRANSMISSION
297     // TEACK: TRANSMIT ENABLE ACKNOWLEDGE FLAG. HARDWARE SETS OR RESETS IT.
298     while ((USART1_ISR & USART_ISR_TEACK) == 0);
299
300     // VERIFY THAT USART IS READY FOR RECEPTION
301     //REACK : RECIEVE ENABLE ACKNOWLEDGE FLAG. HARDWARE SETS OR RESETS IT.
302     while((USART1_ISR & USART_ISR_REACK) == 0);
303   }
304
305   // USART1 READ FUNCTION
306   void USART1_Read(char *buffer, int bytes)
307   {
308     int i;
309
310     for(i = 0; i < bytes; i++)
311     {
312       // WAIT UNTIL HARDWARE SETS RXNE
313       while(!(USART1_ISR & USART_ISR_RXNE));
314       buffer[i] = USART1_RDR;
315     }
316   }
317
318   // USART1 WRITE FUNCTION
319   void USART1_Write(char *buffer, int bytes)
320   {
321     int i,j;
322
323     for(i = 0; i < bytes; i++)
324     {
325       // WRITING TO TDR CLEARS TXE FLAG
326       USART1_TDR = buffer[i] & 0XFF;
327
328       // WAIT UNTIL HARDWARE SETS TXE
329       while(!(USART1_ISR & USART_ISR_TXE));
330
331       //delay
332       for(j = 0;j<WAIT_TIME;j++);
333     }
334
335     // WAIT UNTIL TC BIT IS SET. TC IS SET BY HARDWARE AND CLEARED BY SOFTWARE
336     // TC:TRANSMISSION COMPLETE FLAG
337     while(!(USART1_ISR & USART_ISR_TC));
338
339     // WRITING 1 TO THE TCCF BIT IN ICR CLEARS THE TC BIT IN ISR
340     USART1_ISR &= ~(USART_ISR_TC);
341
342     // TCCF:TRANSMISSION COMPLETE CLEAR FLAG
343     USART1_ICR |= USART_ICR_TCCF;
344   }
345
346   void ADC_Init(void)
347   {
348     int i;
349
350     RCC_AHB2ENR |= (1 << 13); //set ADC clk
351
352     //mode default to analog
353     GPIOA_PUPDR &= ~(1  << 1); //PA1 to no pull or down
354
355     ADC_CR &= ~(1 << 29); //deep power mode cleared
356     ADC_CR |=  (1 << 28); //set voltage reg
357
358     for (i=0; i <10000; i++);//wait for .5us
359
360     ADC_CCR |= (1 << 22); //VREF ENAB
```

```
361     ADC_CCR |= (1 << 16); //HCLK/1 (Synchronous clock mode) enb
362
363     ADC_ISR |= (1 << 0); //ADC ready
364
365     ADC_CR |= (1 << 0); //ENB ADC
366
367     while ((ADC_ISR & (1 << 0)) == 0); //wait till ADC is ready
368
369     //ADC_SQR1 L=1 length of sequence is 1
370     ADC_SQR1 |= (6 << 6);//CH6
371
372     ADC_CFGR |= (1 << 16); //DISCEN: Discontinuous mode for regular channels
373   }
374
375   void PWM_Init(void)
376   {
377     RCC_AHB2ENR  |= (1 << 1); //set the GPIOB clk
378     RCC_APB1ENR1 |= (1 << 0); //TIM2 enb
379
380     GPIOB_MODER &= ~(3 << (2 * 3)); //clear the GPIOB mode bits
381     GPIOB_MODER |=  (2 << (2 * 3));  //set port b3 is alternate 10
382     GPIOB_AFRL  |=  (1 << (4 * 3));   //alt func 1, port pin 3,
383                                      //control bit are 4 bit wide
384
385     TIM2_CR1 |= (1 << 7); //ARPE: Auto-reload preload enable
386     TIM2_PSC = 0;         //PSC set to 0
387     TIM2_ARR = MAX_ARR_VALUE;     //4MHz/4000= 1000 Hz = 1 ms
388
389     TIM2_CCMR1 |= 0x6800; //Channel 2;
390                           //bit 11: OC2PE: Output compare 2 preload enable
391                           //0110: PWM mode 1 - In upcounting,
392                           //channel 1 is active as long as
393                            //TIMx_CNT<TIMx_CCR1else inactive.
394
395     TIM2_CCER |= (1 << 4); //CC2E: Capture/Compare 2 output enable.
396
397     TIM2_CCR2 |= 0;     //CCR2 is the value to be loaded in the actual
398                            //capture/compare 2 register (preload value).
399
400     TIM2_EGR |= (1 << 0); //UG: update event
401
402     TIM2_CR1 |= (1 << 0); //CEN: counter enabled
403
404   }
405
406
```

```
1   //Kunal Mukherjee
2   //4/19/2019
3
4   #define MASTER_READ 1
5   #define MASTER_WRITE 0
6   #define I2C_CR1_PE      (1 << 0)
7   #define I2C_CR1_TCIE    (1 << 6)
8   #define I2C_CR2_WRN     (1 << 10)
9   #define I2C_CR2_START   (1 << 13)
10  #define I2C_CR2_STOP    (1 << 14)
11  #define I2C_CR2_NACK    (1 << 15)
12  #define I2C_ISR_STOPF   (1 << 05)
13  #define I2C_ICR_STOPCF  (1 << 05)
14  #define I2C_ISR_BUSY    (1 << 15)
15  #define I2C_ISR_TXIS    (1 << 01)
16  #define I2C_ISR_TXE     (1 << 00)
17  #define I2C_ISR_TC      (1 << 06)
18  #define I2C_ISR_NACKF   (1 << 04)
19  #define I2C_ISR_RXNE    (1 << 02)
20  #define I2C_ISR_ADDCODE (127 << 17)
21  #define I2C_ISR_DIR     (1 << 16)
22  #define I2C_ISR_ADDR    (1 << 3)
23
24  #define USART_CR1_TE      (1 << 3)
25  #define USART_CR1_RE      (1 << 2)
26  #define USART_ISR_TEACK   (1 << 21)
27  #define USART_ISR_REACK   (1 << 22)
28  #define USART_ISR_RXNE    (1 << 5)
29  #define USART_ISR_TXE     (1 << 7)
30  #define USART_ISR_TC      (1 << 6)
31  #define USART_ICR_TCCF    (1 << 6)
32
33
34  //RCC starts at 0x4002 1000
35  #define RCC_AHB2ENR  (*((volatile unsigned long *) 0x4002104C)) //AHB2 peripheral clock enable register
36  #define RCC_APB1ENR1 (*((volatile unsigned long *) 0x40021058)) //APB1 peripheral clock enable register 1
37  #define RCC_APB2ENR  (*((volatile unsigned long *) 0x40021060))  //Peripheral Clock Enable Register
38
39  //GPIOA start 0x4800 0000
40  #define GPIOA_MODER   (*((volatile unsigned long *) 0x48000000)) //GPIO A Mode register
41  #define GPIOA_OTYPER  (*((volatile unsigned long *) 0x48000004)) //GPIO A Output type reg
42  #define GPIOA_OSPEEDR (*((volatile unsigned long *) 0x48000008)) //GPIO A Output speed register
43  #define GPIOA_PUPDR   (*((volatile unsigned long *) 0x4800000C)) //GPIO A Pudr register
44  #define GPIOA_AFRL    (*((volatile unsigned long *) 0x48000020)) //GPIO A Alternate func register low
45  #define GPIOA_AFRH    (*((volatile unsigned long *) 0x48000024)) //GPIO A Alternate func register high
46  #define GPIOA_ODR     (*((volatile unsigned long *) 0x48000014)) //GPIO A Output data reg
47  #define GPIOA_IDR     (*((volatile unsigned long *) 0x48000010)) //GPIO A Output data reg
48
49  //GPIOB start 0x4800 0400
50  #define GPIOB_MODER   (*((volatile unsigned long *) 0x48000400)) //GPIO B Mode register
51  #define GPIOB_PUPDR   (*((volatile unsigned long *) 0x4800040C)) //GPIO B Pudr register
52  #define GPIOB_BSRR    (*((volatile unsigned long *) 0x48000418)) //GPIO B Output Bit set/reset register
53  #define GPIOB_AFRL    (*((volatile unsigned long *) 0x48000420)) //GPIO B Alternate func register
54  #define GPIOB_ODR     (*((volatile unsigned long *) 0x48000414)) //GPIO B Output data reg
55  #define GPIOB_OTYPER  (*((volatile unsigned long *) 0x48000404)) //GPIO B Output type reg
56  #define GPIOB_OSPEEDR (*((volatile unsigned long *) 0x48000408)) // speed register
57
58  //ADC start 0x5004 0000
59  #define ADC_ISR     (*((volatile unsigned long *) 0x50040000)) //ADC interrupt and status register
60  #define ADC_IER     (*((volatile unsigned long *) 0x50040004)) //ADC interrupt enable register
61  #define ADC_CR      (*((volatile unsigned long *) 0x50040008)) //ADC control register
62  #define ADC_SQR1    (*((volatile unsigned long *) 0x50040030)) //ADC regular sequence register
63  #define ADC_DR      (*((volatile unsigned long *) 0x50040040)) //ADC data register
64  #define ADC_CCR     (*((volatile unsigned long *) 0x50040308)) //ADC common control register
65  #define ADC_CFGR    (*((volatile unsigned long *) 0x5004000C)) //ADC configuration register
66
67  //TIM1 start 0x4001 2C00
68  #define TIM1_CR1   (*((volatile unsigned long *) 0x40012C00)) //TIM1 control register
69  #define TIM1_EGR   (*((volatile unsigned long *) 0x40012C04)) //TIM1 event generation register
70  #define TIM1_CCMR1 (*((volatile unsigned long *) 0x40012C18)) //TIM1 capture/compare mode register
71  #define TIM1_CCMR2 (*((volatile unsigned long *) 0x40012C1C)) //TIM1 capture/compare mode register
72  #define TIM1_PSC   (*((volatile unsigned long *) 0x40012C28)) //TIM1 event generation register
```

```
  73  #define TIM1_ARR   (*((volatile unsigned long *) 0x40012C2C)) //TIM1 auto-reload register
  74  #define TIM1_CCR1  (*((volatile unsigned long *) 0x40012C34)) //TIM1 capture/compare register
  75  #define TIM1_CCR2  (*((volatile unsigned long *) 0x40012C38)) //TIM1 capture/compare register
  76  #define TIM1_CCR3  (*((volatile unsigned long *) 0x40012C3C)) //TIM1 capture/compare register
  77  #define TIM1_CCR4  (*((volatile unsigned long *) 0x40012C40)) //TIM1 capture/compare register
  78  #define TIM1_CCER  (*((volatile unsigned long *) 0x40012C20)) //TIM1 capture/compare enable register
  79  #define TIM1_DIER  (*((volatile unsigned long *) 0x40012C0C)) //TIM1 interrupt enable register
  80  #define TIM1_SR    (*((volatile unsigned long *) 0x40012C10)) //TIM1 status register
  81
  82  //TIM2 start 0x4000 0000
  83  #define TIM2_CR1   (*((volatile unsigned long *) 0x40000000)) //TIM2 control register
  84  #define TIM2_EGR   (*((volatile unsigned long *) 0x40000014)) //TIM2 event generation register
  85  #define TIM2_CCMR1 (*((volatile unsigned long *) 0x40000018)) //TIM2 capture/compare mode register
  86  #define TIM2_CCMR2 (*((volatile unsigned long *) 0x4000001C)) //TIM2 capture/compare mode register
  87  #define TIM2_PSC   (*((volatile unsigned long *) 0x40000028)) //TIM2 event generation register
  88  #define TIM2_ARR   (*((volatile unsigned long *) 0x4000002C)) //TIM2 auto-reload register
  89  #define TIM2_CCR1  (*((volatile unsigned long *) 0x40000034)) //TIM2 capture/compare register
  90  #define TIM2_CCR2  (*((volatile unsigned long *) 0x40000038)) //TIM2 capture/compare register
  91  #define TIM2_CCR3  (*((volatile unsigned long *) 0x4000003C)) //TIM2 capture/compare register
  92  #define TIM2_CCR4  (*((volatile unsigned long *) 0x40000040)) //TIM2 capture/compare register
  93  #define TIM2_CCER  (*((volatile unsigned long *) 0x40000020)) //TIM2 capture/compare enable register
  94  #define TIM2_DIER  (*((volatile unsigned long *) 0x4000000C)) //TIM2 interrupt enable register
  95  #define TIM2_SR    (*((volatile unsigned long *) 0x40000010)) //TIM2 status register
  96
  97  //I2C1 starts 0x4000 5400
  98  #define I2C1_CR1     (*((volatile unsigned long *) 0x40005400)) //I2C1 status register
  99  #define I2C1_CR2     (*((volatile unsigned long *) 0x40005404)) //I2C1 status register
 100  #define I2C1_TIMINGR (*((volatile unsigned long *) 0x40005410)) //I2C1 timing register
 101  #define I2C1_ISR     (*((volatile unsigned long *) 0x40005418)) //I2C1 interrupt and starts register
 102  #define I2C1_ICR     (*((volatile unsigned long *) 0x4000541C)) //I2C1 interrupt control register
 103  #define I2C1_TXDR    (*((volatile unsigned long *) 0x40005428)) //I2C1 tranfer data register
 104  #define I2C1_RXDR    (*((volatile unsigned long *) 0x40005424)) //I2C1 receive data register
 105  #define I2C1_OAR1    (*((volatile unsigned long *) 0x40005408)) //I2C1 own address 1 register
 106  #define I2C1_OAR2    (*((volatile unsigned long *) 0x4000540C)) //I2C1 own address 2 register
 107
 108  //I2C3 starts 0x4000 5C00
 109  #define I2C3_CR1     (*((volatile unsigned long *) 0x40005C00)) //I2C3 status register
 110  #define I2C3_CR2     (*((volatile unsigned long *) 0x40005C04)) //I2C3 status register
 111  #define I2C3_TIMINGR (*((volatile unsigned long *) 0x40005C10)) //I2C3 timing register
 112  #define I2C3_ISR     (*((volatile unsigned long *) 0x40005C18)) //I2C3 interrupt and starts register
 113  #define I2C3_ICR     (*((volatile unsigned long *) 0x40005C1C)) //I2C3 interrupt control register
 114  #define I2C3_TXDR    (*((volatile unsigned long *) 0x40005C28)) //I2C3 tranfer data register
 115  #define I2C3_RXDR    (*((volatile unsigned long *) 0x40005C24)) //I2C3 receive data register
 116  #define I2C3_OAR1    (*((volatile unsigned long *) 0x40005C08)) //I2C3 own address 1 register
 117  #define I2C3_OAR2    (*((volatile unsigned long *) 0x40005C0C)) //I2C3 own address 2 register
 118
 119  //NVIC 0xE000 E100 programmer maunal
 120  #define NVIC_ISER0   (*((volatile unsigned long *) 0xE000E100)) //Interrupt set enbale register 31-0
 121  #define NVIC_ISER1   (*((volatile unsigned long *) 0xE000E104)) //Interrupt set enbale register 63-32
 122  #define NVIC_ISER2   (*((volatile unsigned long *) 0xE000E108)) //Interrupt set enbale register 80-64
 123
 124  //SYSCFG 0x4001 0000
 125  #define SYSCFG_EXTICR1 (*((volatile unsigned long *) 0x40010008)) //SYSCFG
 126
 127  //EXTI  0x4001 0400
 128  #define EXTI_IMR1  (*((volatile unsigned long*) 0x40010400))  //EXTI_IMR1
 129  #define EXTI_RTSR1 (*((volatile unsigned long*) 0x40010408))  //EXTI_RTSR1
 130  #define EXTI_PR1   (*((volatile unsigned long*) 0x40010414))  //EXTI_PR1
 131
 132  //USART 0x4001 3800
 133  #define USART1_CR1 (*((volatile unsigned long *) 0x40013800))
 134  #define USART1_CR2 (*((volatile unsigned long *) 0x40013804))
 135  #define USART1_BRR (*((volatile unsigned long *) 0x4001380C))
 136  #define USART1_ISR (*((volatile unsigned long *) 0x4001381C))
 137  #define USART1_RDR (*((volatile unsigned long *) 0x40013824))
 138  #define USART1_TDR (*((volatile unsigned long *) 0x40013828))
 139  #define USART1_ICR (*((volatile unsigned long *)
      0x40013820))
 140
 141  //USART 0x4001 4C00
 142  #define UART4_CR1 (*((volatile unsigned long *) 0x40004C00))
 143  #define UART4_CR2 (*((volatile unsigned long *) 0x40004C04))
```

```
144   #define UART4_BRR (*((volatile unsigned long *) 0x40004C0C))
145   #define UART4_ISR (*((volatile unsigned long *) 0x40004C1C))
146   #define UART4_RDR (*((volatile unsigned long *) 0x40004C24))
147   #define UART4_TDR (*((volatile unsigned long *) 0x40004C28))
148   #define UART4_ICR (*((volatile unsigned long *) 0x40004C20))
149
150
151
152
153
```