```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
```