```
1   //Kunal Mukherjee;
2   //3/26/2019
3   //Project 2: HI2C
4
5   #include "stm32l432.h"
6   #include "string.h"
7   #include "stdint.h"
8
9   void GPIO_Init(void);
10
11  //I2C1 function definition
12  void I2C1_Initialize(void);
13
14  void I2C1_Start(unsigned long SlaveAddress,
15                  char size,
16                  char direction);
17
18  void I2C1_Stop(void);
19
20  void I2C1_WaitLineIdle(void);
21
22  int I2C1_SendData(char SlaveAddress,
23                    char size,
24                    char * pData);
25  int I2C1_ReceiveData(char SlaveAddress,
26                       char size,
27                       char * pData);
28
29  //I2C3 function definition
30  void I2C3_Initialize(unsigned long slaveAddress);
31
32  void I2C3_Start(unsigned long SlaveAddress,
33                  char size,
34                  char direction);
35
36  void I2C3_Stop(void);
37
38  void I2C3_WaitLineIdle(void);
39
40  int I2C3_SendData(char SlaveAddress,
41                    char size,
42                    char * pData);
43
44  int I2C3_ReceiveData(char SlaveAddress,
45                       char size,
46                       char * pData);
47
48  //LED Blink
49  void LEDBlink (int number);
50
51  int main()
52  {
53    char Data_Receive[8];
54    char Data_Send[8];
55    char num = 1;
56
57    unsigned long slaveAddress = 0x53;
58    int i;
59
60    Data_Send[0] = 0x03;
61    GPIO_Init();
62    I2C1_Initialize();
63    I2C3_Initialize(slaveAddress);
64
65    //setup the global interrupt
66    NVIC_ISER2 |= (1 << 8); //I2C3_EV event interrupt 72,73
67    NVIC_ISER2 |= (1 << 9); //I2C3_ER error interrupt
68
69    while(1)
70    {
71      I2C1_ReceiveData(slaveAddress, num, Data_Send);
72      //or
```

```
73        //I2C1_SendData(slaveAddress, num, Data_Send);
74
75        //delay
76        for (i = 0; i < 1000; i++);
77      }
78
79      return 0;
80    }
81
82    void I2C3_EV_IRQHandler()
83    {
84      //variables to hold the receive address and the direction
85      char direction, ReceiveAddress;
86      char data[6];
87      //the data the slave sends out if the Master wants to receive
88      data[0] = 0x05;
89
90      //the index of the array that contains the data
91      int num = 1;
92
93      //receive data register not empty
94      if ((I2C3_ISR & I2C_ISR_RXNE) == I2C_ISR_RXNE) //master write, slave read
95      {
96        I2C3_ReceiveData(ReceiveAddress, num, data);
97      }
98
99      //check to see if slave was called
100     if ((I2C3_ISR & I2C_ISR_ADDR) == I2C_ISR_ADDR)
101     {
102       ReceiveAddress = (I2C3_ISR & I2C_ISR_ADDCODE) >> 17;
103       direction = (I2C3_ISR & I2C_ISR_DIR) >> 16;
104
105       I2C3_SendData(ReceiveAddress, num, data);
106
107     }
108
109
110     //disable the ADDR flag
111     I2C3_ICR |= (1 << 3);
112     return;
113   }
114
115   void GPIO_Init(void)
116   {
117     RCC_AHB2ENR  |= (1 << 0);    //GPIOA clk enable
118     RCC_AHB2ENR  |= (1 << 1);    //GPIOB clock enable bit
119
120     //set up PA9 as alternate func I2C1_SCL
121     //set up PA10 as alternate func I2C1_SDA
122
123     GPIOA_MODER   &= ~(3 << (2 * 9)); //clear the GPIOA mode bits
124     GPIOA_MODER   |=  (2 << (2 * 9)); //set port PA9 is alternate 10
125
126     GPIOA_MODER   &= ~(3 << (2 * 10)); //clear the GPIOA mode bits
127     GPIOA_MODER   |=  (2 << (2 * 10)); //set port PA10 is alternate 10
128
129     GPIOA_AFRH    |=  (4 << (4 * 1)); //alt func 1, port pin 1,
130                                       //control bit are 4 bit wide
131
132     GPIOA_AFRH    |=  (4 << (4 * 2)); //alt func 1, port pin 1,
133                                       //control bit are 4 bit wide
134
135     GPIOA_OTYPER |= (1 << 9);  //pa9 as open-drain
136     GPIOA_OTYPER |= (1 << 10); //pa10 as open-drain
137
138     GPIOB_MODER &= ~(3 << (2 * 3));
139     GPIOB_MODER |=  (1 << (2 * 3)); //PB3 Output
140
141     //set up PA7 as alternate func I2C3_SCL
142     //set up PB4 as alternate func I2C3_SDA
143
144     GPIOA_MODER   &= ~(3 << (2 * 7)); //clear the GPIOA mode bits
```

```
145     GPIOA_MODER   |=  (2 << (2 * 7)); //set port PA7 is alternate 10
146
147     GPIOB_MODER   &= ~(3 << (2 * 4)); //clear the GPIOB mode bits
148     GPIOB_MODER   |=  (2 << (2 * 4)); //set port PB4 is alternate 10
149
150     GPIOA_AFRL    &=  ~((unsigned long)(15 << (4 * 7)));
151     GPIOA_AFRL    |=   (4  << (4 * 7)); //alt func 1, port pin 1,
152                                         //control bit are 4 bit wide
153
154     GPIOB_AFRL    &=  ~(15 << (4 * 4));
155     GPIOB_AFRL    |=   (4  << (4 * 4)); //alt func 1, port pin 1,
156                                         //control bit are 4 bit wide
157
158     GPIOA_OTYPER |= (1 << 7);  //pa7 as open-drain
159     GPIOB_OTYPER |= (1 << 4);  //pb4 as open-drain
160   }
161
162   void I2C1_Initialize(void)
163   {
164     RCC_APB1ENR1 |= 1 << 21; //I2C1 clock enable
165     I2C1_CR1 &= ~I2C_CR1_PE;
166
167     I2C1_TIMINGR |= 0 << 28; //Fpresc = Fi2c/(0+1) = 4 MHz
168     I2C1_TIMINGR |= 5 << 20; //SCLDEL = 5 -> 1.5 us
169     I2C1_TIMINGR |= 5 << 16; //SDADEL = 5 -> 1.5 us
170     I2C1_TIMINGR |= 19 << 8; //SCLH = 19 -> Thigh = 5 us
171     I2C1_TIMINGR |= 19; //SCLL = 19 -> Tlow = 5 us
172
173     I2C1_CR1 |= (1 << 01); //transmit interrupt
174     I2C1_CR1 |= (1 << 02); //receive interrupt
175     I2C1_CR1 |= (1 << 03); //address match interrupt
176     I2C1_CR1 |= (1 << 05); //STOP detection interrupt
177     I2C1_CR1 |= (1 << 06); //transmit complete interrupt
178
179     I2C1_CR1 |= I2C_CR1_PE; //peripheral enbale
180   }
181
182   void I2C1_Start(unsigned long SlaveAddress,
183                   char size,
184                   char direction)
185   {
186     unsigned long temp = I2C1_CR2;
187
188       temp &= 0xFC009800; //clear SADD, NBYTES, RELOAD, AUTOEND, RD_WRN, START, STOP
189
190     if(direction == MASTER_READ)
191     {
192       temp |= I2C_CR2_WRN; //Transfer direction (master mode)
193     }
194     else
195     {
196         temp &= ~(unsigned long)I2C_CR2_WRN; //0: Master requests a write transfer
197     }
198
199     temp |= SlaveAddress << 1; //SADD[7:1]: Slave address bit 7:1 (master mode)
200     temp |= (unsigned long) size << 16; //NBYTES[7:0] num of bytes
201     temp |= I2C_CR2_START; //Bit 13 START: Start generation
202                            //cleared by software by writing ‘1’
203                            //to the ADDRCF bit in the I2C_ICR register
204
205     I2C1_CR2 = temp;
206   }
207
208   void I2C1_Stop(void)
209   {
210     //generate the STOP bit after the current byte has been transferred
211     I2C1_CR2 |= I2C_CR2_STOP;
212
213     //wait while stopf flag is reset
214     while ((I2C1_ISR & I2C_ISR_STOPF) == 0);
215
216     I2C1_ICR |= I2C_ICR_STOPCF; //write 1 to clear STOPF flag
```

```
217   }
218
219   void I2C3_Initialize(unsigned long slaveAddress)
220   {
221     RCC_APB1ENR1 |= 1 << 23; //I2C3 clock enable
222     I2C3_CR1 &= ~I2C_CR1_PE;
223
224     //enabling the interrupt
225     I2C3_CR1 |= (1 << 01); //transmit interrupt
226     I2C3_CR1 |= (1 << 02); //receive interrupt
227     I2C3_CR1 |= (1 << 03); //address match interrupt
228     I2C3_CR1 |= (1 << 05); //STOP detection interrupt
229     I2C3_CR1 |= (1 << 06); //transmit complete interrupt
230     I2C3_CR1 |= (1 << 16); //slave byte control
231     I2C3_CR1 |= (1 << 17); //clock stret disable
232
233     I2C3_OAR1 |= (slaveAddress << 1); //ERROR [7:1] own address
234
235     I2C3_OAR1 |= (1 << 15); //own slave address enabled
236                             //receive slave address ACKed
237
238     I2C3_CR2 |= I2C_CR2_NACK;
239
240     I2C3_CR1 |= I2C_CR1_PE; //peripheral enbale
241   }
242
243   void I2C1_WaitLineIdle(void)
244   {
245     while ((I2C1_ISR & I2C_ISR_BUSY) == I2C_ISR_BUSY); //if busy wait
246   }
247
248   int I2C1_SendData(char SlaveAddress,
249                     char size,
250                     char * pData)
251   {
252     int i;
253     if (size <= 0 || pData == NULL) return -1;
254
255     //wait until the line is idle
256     I2C1_WaitLineIdle();
257
258     I2C1_Start(SlaveAddress, size, MASTER_WRITE);
259
260     for (i = 0; i < (int)size; i++)
261     {
262       //wait for Transmit register empty to be empty
263       while((I2C1_ISR & I2C_ISR_TXIS) == 0);
264       //TXIS is cleared by writing to the TXDR reg
265       I2C1_TXDR = pData[i];
266     }
267
268     //wait intil TC transfer complete flag is set
269     while ((I2C1_ISR & I2C_ISR_TC) == 0 &&
270             (I2C1_ISR & I2C_ISR_NACKF) == 0);
271
272     if (( I2C1_ISR & I2C_ISR_NACKF) != 0)
273       return -1;
274
275     I2C1_Stop();
276
277     return 0;
278   }
279
280   int I2C1_ReceiveData(char SlaveAddress,
281                        char size,
282                        char * pData)
283   {
284     int i;
285     if (size <= 0 || pData == NULL) return -1;
286
287     //wait until the line is idle
288     I2C1_WaitLineIdle();
```

```
289
290     I2C1_Start(SlaveAddress, size, MASTER_READ);
291
292     for(i = 0; i < size; i++)
293     {
294       //wait until receive data register not empty flag set
295       while((I2C1_ISR & I2C_ISR_RXNE) == 0);
296       pData[i] = I2C1_RXDR;
297     }
298
299     //blink LED
300     LEDBlink(pData[0]);
301
302     //wait until TCR flag is set
303     while ((I2C1_ISR & I2C_ISR_TC) == 0);
304
305     I2C1_Stop();
306
307     return 0;
308   }
309
310   /*void I2C3_Start(unsigned long SlaveAddress,
311                   char size,
312                   char direction)
313   {
314     unsigned long temp = I2C3_CR2;
315
316       temp &= 0xFC009800; //clear SADD, NBYTES, RELOAD, AUTOEND, RD_WRN, START, STOP
317
318     if(direction == MASTER_READ)
319     {
320       temp |= I2C_CR2_WRN;
321     }
322     else
323     {
324         temp &= ~(unsigned long)I2C_CR2_WRN;
325     }
326
327     temp |= SlaveAddress << 1;
328     temp |= (unsigned long) size << 16;
329     temp |= I2C_CR2_START;
330
331     I2C3_CR2 = temp;
332   }*/
333
334   /*void I2C3_Stop(void)
335   {
336     //generate the STOP bit after the current byte has been transferred
337     I2C3_CR2 |= I2C_CR2_STOP;
338
339     //wait while stopf flag is reset
340     //while ((I2C3_ISR & I2C_ISR_STOPF) == 0);
341
342     I2C3_ICR |= I2C_ICR_STOPCF; //write 1 to clear STOPF flag
343   }*/
344
345   /*void I2C3_WaitLineIdle(void)
346   {
347     while ((I2C3_ISR & I2C_ISR_BUSY) == I2C_ISR_BUSY); //if busy wait
348   }*/
349
350
351   int I2C3_SendData(char SlaveAddress,
352                     char size,
353                     char * pData)
354   {
355     int i;
356     if (size <= 0 || pData == NULL) return -1;
357
358     for (i = 0; i < (int)size; i++)
359     {
360
```

```
361       while((I2C3_ISR & I2C_ISR_TXIS) == 0);
362       //TXIS is cleared by writing to the TXDR reg
363       I2C3_TXDR = pData[i];
364     }
365
366     return 0;
367   }
368
369   int I2C3_ReceiveData(char SlaveAddress,
370                        char size,
371                        char * pData)
372   {
373     int i;
374     if (size <= 0 || pData == NULL) return -1;
375
376     for(i = 0; i < size; i++)
377     {
378       pData[i] = I2C3_RXDR;
379     }
380
381     //blink LED
382     LEDBlink(pData[0]);
383
384     return 0;
385   }
386
387   void LEDBlink (int number)
388   {
389     int i, j, k;
390
391     for (i = 0; i < number; i++)
392     {
393       GPIOB_ODR |= (1 << 3);
394       for (j = 0; j < 10000; j++){for (k = 0; k < 20; k++);}; //delay
395       GPIOB_ODR &= ~(1 << 3);
396       for (j = 0; j < 10000; j++){for (k = 0; k < 20; k++);}; //delay
397     }
398   }
```