PROJECT 2 (MK 1)

Kunal Mukherjee

UNIVERSITY OF EVANSVILLE

STATEMENT OF THE PROJECT

The project was assigned to give us experience with I2C transmission protocol. I2C is one of the widely used protocol for sensors and inter-device communication. Therefore, having a strong grasp of the I2C protocol was necessary, and this project where we had to make I2C1 configured as master had to communicate with I2C3 which was configured as a slave.

POSSIBLE UTILITY OF THE PROJECT

The possible utility of the project is in sensor communication and game development. For EE 380, on the sensors is using I2C protocol to communicate. By having completed the code for this project, the I2C master and slave code can be easily modified to be used with the sensor for EE 380.

IDENTIFICATION OF SPECIFICATION

The specification was that the user should be able to give any slave address. Then by using the slave address, the user should be able to request the slave to blink a led some number of times that is specified by the user. Also, the master should be able to request a value from the slave, after which it will blink an led corresponding the value sent by the slave.

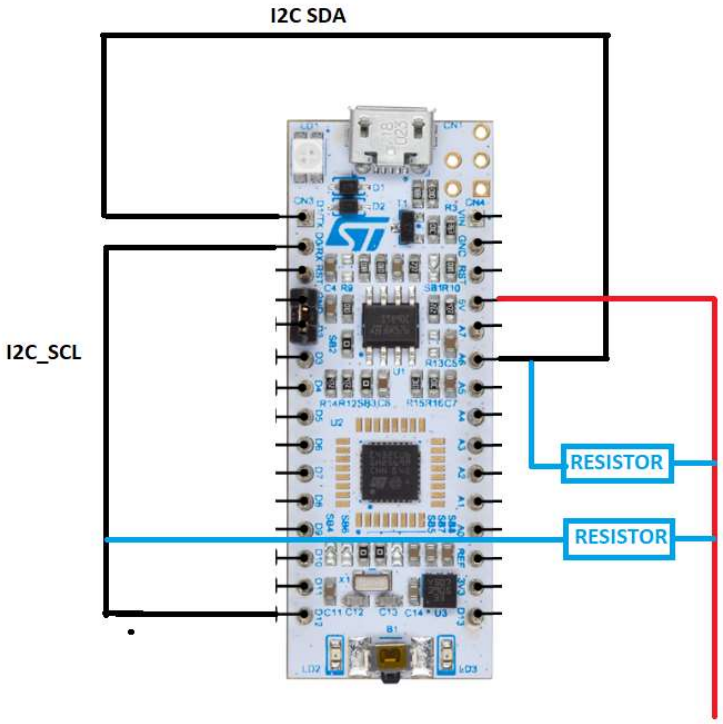
IDENTIFICATION OF DESIGN ISSUES AND SOLUTION

One of the biggest design issues that I faced was, using the correct interrupt. The slave had to look at his own interrupt to determine if it can receive data or can send data, but, the interrupt routine has to look at the TC (transfer complete flag) to see, if the transfer has been completed. The second issue was to determine which interrupt to enable, because if we enabled

all the interrupts, then we won't be able to leave the interrupt routine. We will be continuously stuck in the routine.

Another, place where I struggled was to notice that the slave I2C never gets control of the bus, so the idle, wait, start, stop function is useless. As, the slave cannot start or stop a transmission but rather send a acknowledge or a no acknowledge flag. In the beginning, I did not understand this concept, and I tried giving the slave control, and I my code would not work. But, after I understood the problem, I was able to work around it.

SCHEMATIC OF COMPONENTS EXTERNAL TO THE STM32F446 BOARD



INFORMATIONAL RESOURCES USED

- https://www.youtube.com/watch?v=Wdac8FnZvQA [58. How to: I2C Circuit and Initialization - ARM STM32 Microcontroller Tutorial]
- https://www.youtube.com/watch?v=XvakWfRHrCY&list=PL6PplMTH29SHgRPDufZhfMRoFwRAIrzOp&index=57 [59. How to Use I2C to Read a Device's Register Part 1 - ARM STM32 Microcontroller Tutorial]
- Embedded System Book
- https://i2c.info/ [I2C Info – I2C Bus, Interface and Protocol]

```
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   }
```

```
1  //Kunal Mukherjee
2  //3/15/2019
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 //RCC starts at 0x4002 1000
25 #define RCC_AHB2ENR  (*((volatile unsigned long *) 0x4002104C)) //AHB2 peripheral clock enable register
26 #define RCC_APB1ENR1 (*((volatile unsigned long *) 0x40021058)) //APB1 peripheral clock enable register 1
27
28 //GPIOA start 0x4800 0000
29 #define GPIOA_MODER   (*((volatile unsigned long *) 0x48000000)) //GPIO A Mode register
30 #define GPIOA_OTYPER  (*((volatile unsigned long *) 0x48000004)) //GPIO A Output type reg
31 #define GPIOA_OSPEEDR (*((volatile unsigned long *) 0x48000008)) //GPIO A Output speed register
32 #define GPIOA_PUPDR   (*((volatile unsigned long *) 0x4800000C)) //GPIO A Pudr register
33 #define GPIOA_AFRL    (*((volatile unsigned long *) 0x48000020)) //GPIO A Alternate func register low
34 #define GPIOA_AFRH    (*((volatile unsigned long *) 0x48000024)) //GPIO A Alternate func register high
35 #define GPIOA_ODR     (*((volatile unsigned long *) 0x48000014)) //GPIO A Output data reg
36
37 //GPIOB start 0x4800 0400
38 #define GPIOB_MODER  (*((volatile unsigned long *) 0x48000400)) //GPIO B Mode register
39 #define GPIOB_PUPDR  (*((volatile unsigned long *) 0x4800040C)) //GPIO B Pudr register
40 #define GPIOB_BSRR   (*((volatile unsigned long *) 0x48000418)) //GPIO B Output Bit set/reset register
41 #define GPIOB_AFRL   (*((volatile unsigned long *) 0x48000420)) //GPIO B Alternate func register
42 #define GPIOB_ODR    (*((volatile unsigned long *) 0x48000414)) //GPIO B Output data reg
43 #define GPIOB_OTYPER (*((volatile unsigned long *) 0x48000404)) //GPIO B Output type reg
44
45 //ADC start 0x5004 0000
46 #define ADC_ISR     (*((volatile unsigned long *) 0x50040000)) //ADC interrupt and status register
47 #define ADC_IER     (*((volatile unsigned long *) 0x50040004)) //ADC interrupt enable register
48 #define ADC_CR      (*((volatile unsigned long *) 0x50040008)) //ADC control register
49 #define ADC_SQR1    (*((volatile unsigned long *) 0x50040030)) //ADC regular sequence register
50 #define ADC_DR      (*((volatile unsigned long *) 0x50040040)) //ADC data register
51 #define ADC_CCR     (*((volatile unsigned long *) 0x50040308)) //ADC common control register
52 #define ADC_CFGR    (*((volatile unsigned long *) 0x5004000C)) //ADC configuration register
53
54 //TIM2 start 0x4000 0000
55 #define TIM2_CR1   (*((volatile unsigned long *) 0x40000000)) //TIM2 control register
56 #define TIM2_EGR   (*((volatile unsigned long *) 0x40000014)) //TIM2 event generation register
57 #define TIM2_CCMR1 (*((volatile unsigned long *) 0x40000018)) //TIM2 capture/compare mode register
58 #define TIM2_CCMR2 (*((volatile unsigned long *) 0x4000001C)) //TIM2 capture/compare mode register
59 #define TIM2_PSC   (*((volatile unsigned long *) 0x40000028)) //TIM2 event generation register
60 #define TIM2_ARR   (*((volatile unsigned long *) 0x4000002C)) //TIM2 auto-reload register
61 #define TIM2_CCR1  (*((volatile unsigned long *) 0x40000034)) //TIM2 capture/compare register
62 #define TIM2_CCR2  (*((volatile unsigned long *) 0x40000038)) //TIM2 capture/compare register
63 #define TIM2_CCR3  (*((volatile unsigned long *) 0x4000003C)) //TIM2 capture/compare register
64 #define TIM2_CCR4  (*((volatile unsigned long *) 0x40000040)) //TIM2 capture/compare register
65 #define TIM2_CCER  (*((volatile unsigned long *) 0x40000020)) //TIM2 capture/compare enable register
66 #define TIM2_DIER  (*((volatile unsigned long *) 0x4000000C)) //TIM2 interrupt enable register
67 #define TIM2_SR    (*((volatile unsigned long *) 0x40000010)) //TIM2 status register
68
69 //I2C1 starts 0x4000 5400
70 #define I2C1_CR1     (*((volatile unsigned long *) 0x40005400)) //I2C1 status register
71 #define I2C1_CR2     (*((volatile unsigned long *) 0x40005404)) //I2C1 status register
72 #define I2C1_TIMINGR (*((volatile unsigned long *) 0x40005410)) //I2C1 timing register
```

```
73  #define I2C1_ISR     (*((volatile unsigned long *) 0x40005418)) //I2C1 interrupt and starts register
74  #define I2C1_ICR     (*((volatile unsigned long *) 0x4000541C)) //I2C1 interrupt control register
75  #define I2C1_TXDR    (*((volatile unsigned long *) 0x40005428)) //I2C1 tranfer data register
76  #define I2C1_RXDR    (*((volatile unsigned long *) 0x40005424)) //I2C1 receive data register
77  #define I2C1_OAR1    (*((volatile unsigned long *) 0x40005408)) //I2C1 own address 1 register
78  #define I2C1_OAR2    (*((volatile unsigned long *) 0x4000540C)) //I2C1 own address 2 register
79
80  //I2C3 starts 0x4000 5C00
81  #define I2C3_CR1     (*((volatile unsigned long *) 0x40005C00)) //I2C3 status register
82  #define I2C3_CR2     (*((volatile unsigned long *) 0x40005C04)) //I2C3 status register
83  #define I2C3_TIMINGR (*((volatile unsigned long *) 0x40005C10)) //I2C3 timing register
84  #define I2C3_ISR     (*((volatile unsigned long *) 0x40005C18)) //I2C3 interrupt and starts register
85  #define I2C3_ICR     (*((volatile unsigned long *) 0x40005C1C)) //I2C3 interrupt control register
86  #define I2C3_TXDR    (*((volatile unsigned long *) 0x40005C28)) //I2C3 tranfer data register
87  #define I2C3_RXDR    (*((volatile unsigned long *) 0x40005C24)) //I2C3 receive data register
88  #define I2C3_OAR1    (*((volatile unsigned long *) 0x40005C08)) //I2C3 own address 1 register
89  #define I2C3_OAR2    (*((volatile unsigned long *) 0x40005C0C)) //I2C3 own address 2 register
90
91  //NVIC 0xE000 E100 programmer maunal
92  #define NVIC_ISER0   (*((volatile unsigned long *) 0xE000E100)) //Interrupt set enbale register 31-0
93  #define NVIC_ISER1   (*((volatile unsigned long *) 0xE000E104)) //Interrupt set enbale register 63-32
94  #define NVIC_ISER2   (*((volatile unsigned long *) 0xE000E108)) //Interrupt set enbale register 80-64
95
96  //SYSCFG 0x4001 0000
97  #define SYSCFG_EXTICR1    (*((volatile unsigned long *) 0x40010008)) //SYSCFG
98
99  //EXTI  0x4001 0400
100 #define EXTI_IMR1         (*((volatile unsigned long*) 0x40010400)) //EXTI_IMR1
101 #define EXTI_RTSR1        (*((volatile unsigned long*) 0x40010408)) //EXTI_RTSR1
102 #define EXTI_PR1          (*((volatile unsigned long*) 0x40010414)) //EXTI_PR1
103
104
105
106
107
```