ELEC 458 – EMBEDDED SYSTEMS

PROJECT 2 - REMOTE KEYLESS SYSTEM

AZİZ CAN AKKAYA 171024005
BERK SARI 141024068

Contents

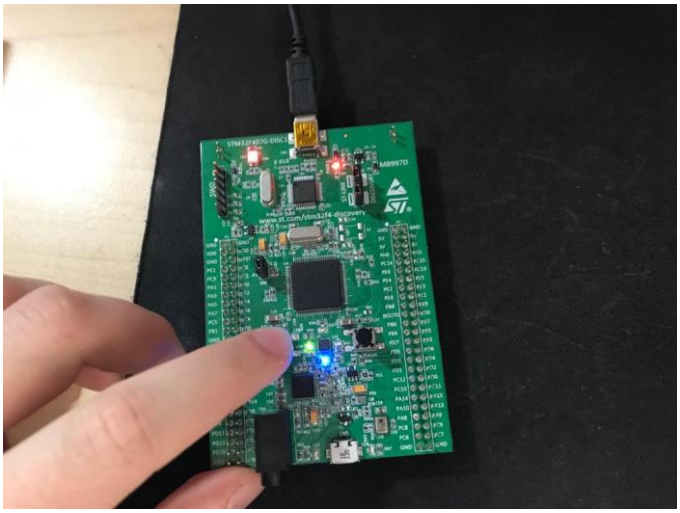
1 Intorduction

1.1 Briefing

Document Version : 1.4

Document Publised Date : 12/5/2020

Project Version : v2.4_c



1.2 Objective

This project is developed on a STM32F407VG Discovery Kit. The ARM Cortex-M4 architecture gained functionality with “main.c” file developed with Embedded C language. The aim of the Project is to understand and gain a brief knowledge about ARM Cortex-M4 and it’s functionality along side with system’s supported features.

1.2.1 Project Tasks

Aziz Can AKKAYA	-Algorithm Design with Interrupts -AES Encryption/Decryption -Data Frame Transmission (UART) -IWDG
Berk SARI	-Calculating Data Frame -AES Encryption/Decryption -Algorithm Design with Interrupts -Flash Memory -IWDG -Button Identification -Power Mode Configuration

2 Technical Aspects

2.1 Materials

This project needed a last longing battery, so we decided to use STM32F407VG ARM Cortex-M4 microcontroller. It has fast execution and respond with a low possibility of data hazard errors.

As for the software we decide to use STM32Cude IDE 1.3.0. The reason behind is rising popularity of the IDE and it has same components as Keil (the most popular IDE currently) but with a different and easier approach to UI.

2.2 Method

The system supports 3 different button press type; single press, double press and long press. Each time button is pressed, an interrupt will be generated according to the button type. Upon button press, a unique Rolling code number will be generated. Because system saves Rolling code in flash memory, reset or restart does not effect rolling code mechanism. The generated Rolling code will be packed with destination adress, source adress and identifier then encrypted with a key using AES. After that the encrypted data will be send through transmitter. On receiver side upon transmission detection an interrupt will be generated and this interrupt will activate receiver. When all the encrypted data collected by receiver, decrypteion will start with same key using AES. After data is decrypted on receiver side, rolling code from decrypted data on receiver side and rolling code from transmitter side will be compared by the system. If two rolling codes matche, user's command will be read from decrypted data frame. LEDs on the board will light up according to how user pressed the button(single, short or long press).

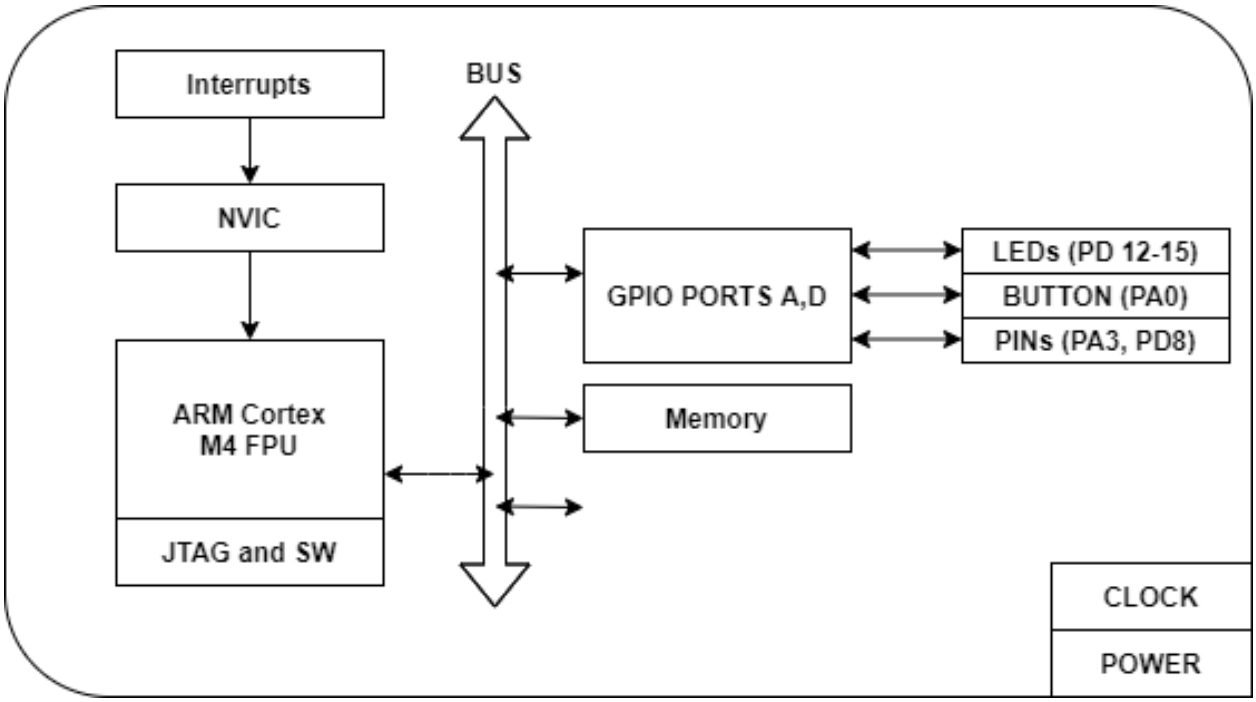
This system will work on STM32F407 Discovery 1 and coded with Embedded C. Lastly in the system, independent watch dog is used and will be in sleep mode to save power when there is no external interrupt from the button. The board requires 5V input and it's provided from USB Mini Type B port.

- The system will light up all the LEDs for a second at startup and when reset happens.
- The system will proceed into sleep mode if there is no external interrupt from the button. Whenever user gives input, system wokes up.
- The System guarded by IWDG (Internal Watchdog Timer) to prevent system lock ups and interrupt based timer is used to measure the time for how long button is pressed.
- Every 0.8 second, interrupt based SystemTick controls if there is any input from user.
- Before Rolling code calculation, previous Rolling code read from flash memory and then flash memory updated with new Rolling code.
- Data Frame will be produced by rolling code, destination adress, source adress and identifier.
- Data transmission is done with 8N1 UART protocol. BAUD rate is hardcoded and cant be changed. At the same time multiple transmission can be done.

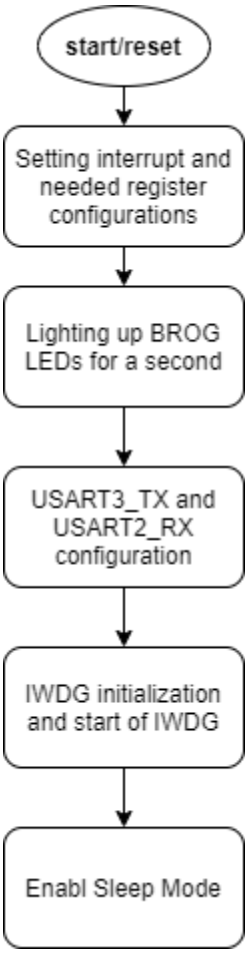
- After reciver got the encrypted Data, it will be decoded and analayzed. If requirements is met and then user's command will be shown on the BROG LEDs.
- If the received data is corrupted in a way and can't be used, the system will blink red LED really fast for half a second and then IWDG (Internal Watchdog Timer) will be triggered to recover the whole system.

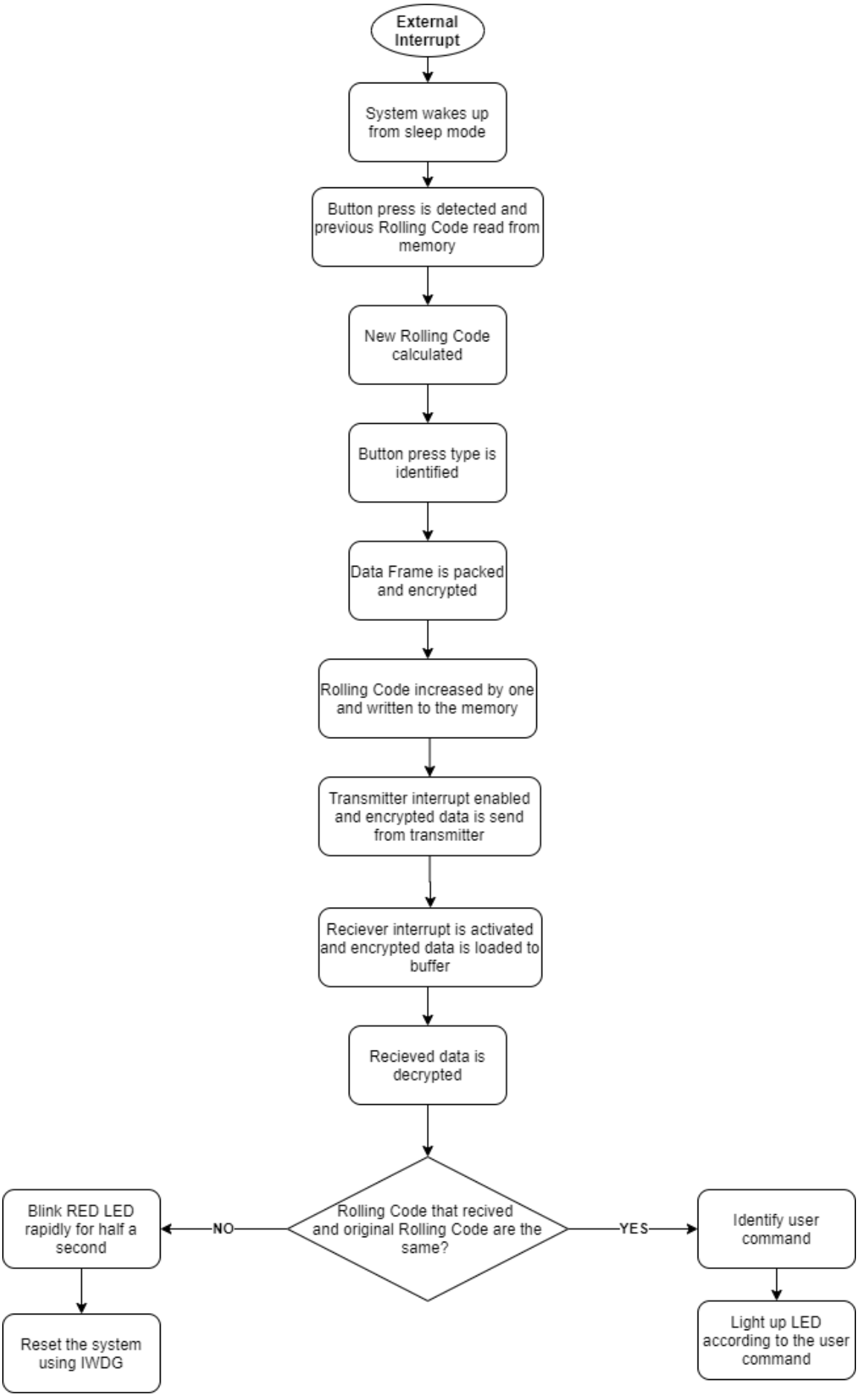
2.3 Diagrams

2.3.1 Hardware Block Diagram



2.3.2 Software Block Diagram




External Interrupt
System wakes up from sleep mode
Button press is detected and previous Rolling Code read from memory
New Rolling Code calculated
Button press type is identified
Data Frame is packed and encrypted
Rolling Code increased by one and written to the memory
Transmitter interrupt enabled and encrypted data is send from transmitter
Reciever interrupt is activated and encrypted data is loaded to buffer
Recieved data is decrypted
Rolling Code that recived and original Rolling Code are the same?
NO
YES
Blink RED LED rapidly for half a second
Reset the system using IWDG
Identify user command
Light up LED according to the user command

3 Output

DataFbuf_tx	volatile uint8_t [16]	0x20000060 <DataFbuf_tx>
DataFbuf_tx[0]	volatile uint8_t	0xd3 (Hex)
DataFbuf_tx[1]	volatile uint8_t	0x1a (Hex)
DataFbuf_tx[2]	volatile uint8_t	0xb6 (Hex)
DataFbuf_tx[3]	volatile uint8_t	0xbb (Hex)
DataFbuf_tx[4]	volatile uint8_t	0xce (Hex)
DataFbuf_tx[5]	volatile uint8_t	0x27 (Hex)
DataFbuf_tx[6]	volatile uint8_t	0x29 (Hex)
DataFbuf_tx[7]	volatile uint8_t	0xcf (Hex)
DataFbuf_tx[8]	volatile uint8_t	0xef (Hex)
DataFbuf_tx[9]	volatile uint8_t	0x56 (Hex)
DataFbuf_tx[10]	volatile uint8_t	0x73 (Hex)
DataFbuf_tx[11]	volatile uint8_t	0x25 (Hex)
DataFbuf_tx[12]	volatile uint8_t	0xf9 (Hex)
DataFbuf_tx[13]	volatile uint8_t	0xdb (Hex)
DataFbuf_tx[14]	volatile uint8_t	0x72 (Hex)
DataFbuf_tx[15]	volatile uint8_t	0x9 (Hex)
DataFbuf_rx	volatile uint8_t [16]	0x2000007c <DataFbuf_rx>
DataFbuf_rx[0]	volatile uint8_t	0xd3 (Hex)
DataFbuf_rx[1]	volatile uint8_t	0x1a (Hex)
DataFbuf_rx[2]	volatile uint8_t	0xb6 (Hex)
DataFbuf_rx[3]	volatile uint8_t	0xbb (Hex)
DataFbuf_rx[4]	volatile uint8_t	0xce (Hex)
DataFbuf_rx[5]	volatile uint8_t	0x27 (Hex)
DataFbuf_rx[6]	volatile uint8_t	0x29 (Hex)
DataFbuf_rx[7]	volatile uint8_t	0xcf (Hex)
DataFbuf_rx[8]	volatile uint8_t	0xef (Hex)
DataFbuf_rx[9]	volatile uint8_t	0x56 (Hex)
DataFbuf_rx[10]	volatile uint8_t	0x73 (Hex)
DataFbuf_rx[11]	volatile uint8_t	0x25 (Hex)
DataFbuf_rx[12]	volatile uint8_t	0xf9 (Hex)
DataFbuf_rx[13]	volatile uint8_t	0xdb (Hex)
DataFbuf_rx[14]	volatile uint8_t	0x72 (Hex)
DataFbuf_rx[15]	volatile uint8_t	0x9 (Hex)

DataFrame_rx	volatile uint32_t	0xa7054401 (Hex)
DataFrame	volatile uint32_t	0xa7054401 (Hex)
Rcode	volatile uint32_t	0xa7 (Hex)
Rcode_rx	volatile uint32_t	0xa7 (Hex)
identifier_rx	volatile uint8_t	0x1 (Hex)

4 Conclusion

4.1 Design Overview

This Project divided in to multiple parts; Rolling Code, Encryption-Decryption, USART Communication, Button Functions and Memory. First step was doing research about the parts and learning how they work. After enough data was collected, main logic and algorithms were created simultaneously.

In this Project, the most challenging parts were USART configuration, transmission and using interrupts for main logic. Lack of reading techniques for datasheets made our objects really hard to reach. Moreover, we could not decide wheter the USART transmission is working or not. Because we could not able to see data between transmitter and receiver. Lack off uart monitoring module, it was hard to track the transmission output. On the other hand, we had to think differently to implement interrupt and its' handlers to our logic.

4.2 Discussion

While working on this Project, we learned;

- How to define and use interrupts and its' handlers.
- The use of ARM C Compiler Debugger, reading data driectly from registers and memory. Crucial usage of breakpoints and expressions.
- Improved our knowledge on C and Embedded C programming.
- How to use and add libraries to the project file.
- Configuration and understanding power mods.
- Usage and knowledge of USART.
- Resetting system with IWDG to prevent lockup.
- Memory manipulation for further usage.
- How to read product's datasheet more efficiently and use its' features effectively.

This Project took around 3 and a half weeks to complete in total. While working on this project, because of our inexperience on Embedded C programming and lack of skills for understanding datasheet, we ran a lot of software logic errors .

In this Project, there is one major problem that we could not able to understand and solve. When we sent "long press" command from transmitter, the reciever could not be able to fetch the data frame correctly. It constantly recieves data from transmitter and stucks in a loop. We could not be able to locate the source of this error because we do not have a UART monitoring module and we could not analyze the signal between transmitter and reciever efficently.

5 Appendix

```
6    #include "stm32f4xx.h"
7    #include "system_stm32f4xx.h"
8    #include "string.h"
9
10   #define CBC 1
11   #define ECB 1
12   #define CTR 1
13   #include "aes.h"
14
15   /************************************************
16   * function declarations
17   ************************************************/
18   void init_systick(uint32_t s, uint8_t cen);
19   int main(void);
20   void delay_ms(volatile uint32_t);
21   void unlock_flash();
22   void lock_flash();
23   void erase_flash_sector3();
24   void write_flash(uint32_t addr, uint32_t datai);
25   void read_flash(uint32_t addr);
26
27   void Decryption_rx();
28   void CommandLEDs();
29
30   /************************************************
31   * variables
32   ************************************************/
33   volatile uint32_t tDelay;
34   volatile uint8_t x = 0x00;
35   volatile uint8_t y = 0x00;
36   volatile uint32_t mem_datao;
37   volatile uint8_t ButAct = 0;
38   volatile uint8_t identifier;
39   volatile uint8_t identifier_rx;
40   volatile uint32_t DataTemplate;
41   volatile uint32_t DataFrame = 0;
42   volatile uint32_t DataFrame_rx;
43   volatile uint32_t RC_in = 0x49;   //73
44   volatile uint32_t RC_mod = 0x100; //256
45   volatile uint32_t Rcode = 0;
46   volatile uint32_t Rcode_rx = 0;
47
48   uint8_t key[] = { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
     0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c };
49   uint8_t iv[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
     0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f };
50   uint8_t in[16];
51
52   #define KEY1 0x45670123
53   #define KEY2 0xCDEF89AB
54   #define KEYADDR 0x0800C000
55
56   volatile int tx_complete = 0;
57   volatile int rx_complete = 0;
58   volatile int bufpos_tx = 0;
59   volatile int bufpos_rx = 0;
60   volatile uint8_t DataFbuf_tx[16];
```

```
61  volatile uint8_t DataFbuf_rx[16];
62  volatile uint8_t c = 0;
63  volatile uint8_t count_tx = 0;
64
65  /***********************************************
66  * timer 2 interrupt handler
67  ***********************************************/
68  uint8_t TIM2_IRQHandler(void)
69  {
70      TIM2->SR = (uint16_t)(~(1 << 0));
71
72      if(ButAct == 1){
73          x++; }       //40 ticks per sec
74   }
75
76  /***********************************************
77  * external interrupt handler
78  ***********************************************/
79  void EXTI0_IRQHandler(void)
80  {
81    if (EXTI->PR & (1 << 0)){
82          ButAct ^= 0x1;
83          for(uint32_t j=0; j<500000; j++);
84
85          if (ButAct == 1){
86                y++;
87
88                read_flash(KEYADDR);
89                unlock_flash();
90                RC_in = mem_datao;
91                lock_flash();
92
93                Rcode = RC_in % 0x100;
94                //GPIOD->ODR ^= (uint16_t)(1 << 14);
95
96          }
97
98          EXTI->PR = (1 << 0);
99    }
100 }
101
102 /***********************************************
103 * SysTick interrupt handler
104 ***********************************************/
105 void SysTick_Handler(void)
106 {
107
108   IWDG->KR |= 0xaaaa;// Resetting IWDG timer
109
110   __disable_irq();
111
112   if(ButAct == 0){
113               if((x >= 1) & (y >= 2) & (x < 40)){          //DOUBLE PRESS
114                     ButAct = 0;
115                 x = 0;
116                 y = 0;
117                 ButAct = 0;
118               //GPIOD->ODR ^= (uint16_t)(1 << 13);
119
```

```
120                 RC_in++;
121                 identifier = 0x02;                                      //DOOR LOCK
122                 DataFrame = ((Rcode << 24) | (DataTemplate << 0) | (identifier
    << 0));
123                 identifier = 0;
124 
125                 // DataFrame Encryption
126                 uint8_t x0 = DataFrame & 0xFF;
127                 uint8_t x1 = (DataFrame >> 8) & 0xFF;
128                 uint8_t x2 = (DataFrame >> 16) & 0xFF;
129                 uint8_t x3 = (DataFrame >> 24) & 0xFF;
130                 in[0] = x0;
131                 in[1] = x1;
132                 in[2] = x2;
133                 in[3] = x3;
134                 for (int i = 4; i < 16; i++){
135                       in[i] = 0x00;
136                 }
137                 struct AES_ctx ctx;
138 //                      AES_init_ctx_iv(&ctx, key, iv);
139 //                      AES_CBC_encrypt_buffer(&ctx, in, 16);
140                 AES_init_ctx(&ctx, key);
141                 AES_ECB_encrypt(&ctx, in);
142 
143                       unlock_flash();
     //MEMORY WRITE
144                       erase_flash_sector3();
145                       write_flash(KEYADDR, RC_in);
146                       lock_flash();
147 
148                       //Enable TX interrupt
149                       USART3->CR1 |= (1 << 7);
150                 }
151 
152                 else if((x >= 1) & (x < 40) & (y < 2)){             //SINGLE
    PRESS
153                 x = 0;
154                 y = 0;
155                 ButAct = 0;
156                 //GPIOD->ODR ^= (uint16_t)(1 << 12);
157 
158                 RC_in++;
159                 identifier = 0x01   ;                                   //DOOR
    UNLCOK
160                 DataFrame = ((Rcode << 24) | (DataTemplate << 0) | (identifier
    << 0));
161                 identifier = 0;
162 
163                 // DataFrame Encryption
164                 uint8_t x0 = DataFrame & 0xFF;
165                 uint8_t x1 = (DataFrame >> 8) & 0xFF;
166                 uint8_t x2 = (DataFrame >> 16) & 0xFF;
167                 uint8_t x3 = (DataFrame >> 24) & 0xFF;
168                 in[0] = x0;
169                 in[1] = x1;
170                 in[2] = x2;
171                 in[3] = x3;
172                 for (int i = 4; i < 16; i++){
173                       in[i] = 0x00;
```

```
174                 }
175                 struct AES_ctx ctx;
176   //                      AES_init_ctx_iv(&ctx, key, iv);
177   //                      AES_CBC_encrypt_buffer(&ctx, in, 16);
178                 AES_init_ctx(&ctx, key);
179                 AES_ECB_encrypt(&ctx, in);
180
181                       unlock_flash();
      //MEMORY WRITE
182                       erase_flash_sector3();
183                       write_flash(KEYADDR, RC_in);
184                       lock_flash();
185
186                       //Enable TX interrupt
187                       USART3->CR1 |= (1 << 7); //tx interrupt
188           }
189
190                 else if(x >= 40){
      //LONG PRESS 2S
191                       x = 0;
192                       y = 0;
193                       ButAct = 0;
194                 //GPIOD->ODR ^= (uint16_t)(1 << 15);
195
196                 RC_in++;
197                 identifier = 0x10;                                    //REMOTE
    START
198                 DataFrame = ((Rcode << 24) | (DataTemplate << 0) | (identifier
    << 0));
199                 identifier = 0;
200
201                 // DataFrame Encryption
202                 uint8_t x0 = DataFrame & 0xFF;
203                 uint8_t x1 = (DataFrame >> 8) & 0xFF;
204                 uint8_t x2 = (DataFrame >> 16) & 0xFF;
205                 uint8_t x3 = (DataFrame >> 24) & 0xFF;
206                 in[0] = x0;
207                 in[1] = x1;
208                 in[2] = x2;
209                 in[3] = x3;
210                 for (int i = 4; i < 16; i++){
211                       in[i] = 0x00;
212                 }
213                 struct AES_ctx ctx;
214   //                      AES_init_ctx_iv(&ctx, key, iv);
215   //                      AES_CBC_encrypt_buffer(&ctx, in, 16);
216                 AES_init_ctx(&ctx, key);
217                 AES_ECB_encrypt(&ctx, in);
218
219                       unlock_flash();
      //MEMORY WRITE
220                       erase_flash_sector3();
221                       write_flash(KEYADDR, RC_in);
222                       lock_flash();
223
224                       //Enable TX interrupt
225                       USART3->CR1 |= (1 << 7); //tx interrupt
226                 }
227   }
```

```
228   __enable_irq();
229  }
230
231  /*************************************************
232  * UART2 interrupt handler -- (RX)
233  *************************************************/
234
235  void USART2_IRQHandler(void)
236  {
237    if (USART2->SR & (1 << 5)){
238        if (bufpos_rx < 16 ) {
239          DataFbuf_rx[bufpos_rx] = USART2->DR;
240          bufpos_rx++;
241        }
242        if (bufpos_rx == 16 ){
243          //USART2->CR1 &= (uint32_t)~(1 << 5);
244          bufpos_rx = 0;
245          rx_complete = 1;
246          Decryption_rx();
247        }
248    }
249  }
250
251  /*************************************************
252  * UART3 interrupt handler -- (TX)
253  *************************************************/
254
255  void USART3_IRQHandler(void)
256  {
257    for(int i = 0; i < 16; i++ ){
258          DataFbuf_tx[i] = in[i];
259    }
260
261    if (USART3->SR & (1 << 7)) {
262            // clear interrupt
263            USART3->SR &= (uint32_t)~(1 << 7);
264
265
266                if (bufpos_tx == 16) {
267                      // buffer is flushed out, disable tx interrupt
268                      tx_complete = 1;
269                      bufpos_tx = 0;
270                      USART3->CR1 &= (uint32_t)~(1 << 7);
271                }
272                else {
273                      USART3->DR = DataFbuf_tx[bufpos_tx];
274                      //delay_ms(200); //0.02s
275                      bufpos_tx++;
276                      tx_complete = 0;
277                }
278    }
279  }
280
281  /*************************************************
282  * initialize SysTick
283  *************************************************/
284  void init_systick(uint32_t s, uint8_t cen)
285  {
286      // Clear CTRL register
```

```
287     SysTick->CTRL = 0x00000;
288     // Main clock source is running with HSI by default which is at 8 Mhz.
289     // SysTick clock source can be set with CTRL register (Bit 2)
290     // 0: Processor clock/8 (AHB/8)
291     // 1: Processor clock (AHB)
292     SysTick->CTRL |= (0 << 2);
293     // Enable callback (bit 1)
294     SysTick->CTRL |= ((uint32_t)cen << 1);
295     // Load the value
296     SysTick->LOAD = s;
297     // Set the current value to 0
298     SysTick->VAL = 0;
299     // Enable SysTick (bit 0)
300     SysTick->CTRL |= (1 << 0);
301 }
302
303 /*********************************************
304 * MEMORY
305 *********************************************/
306
307 void unlock_flash(){
308   FLASH -> KEYR = KEY1;
309   FLASH -> KEYR = KEY2;
310 }
311
312 void lock_flash() {
313     FLASH->CR |= FLASH_CR_LOCK; // bit 31
314 }
315
316 void erase_flash_sector3() {
317     const uint32_t sec = 3;
318     __disable_irq();
319     while(FLASH->SR & FLASH_SR_BSY); // check if busy
320     FLASH->CR |= FLASH_CR_SER;
321     FLASH->CR |= (sec << 3); // SNB bit 3:6
322     FLASH->CR |= FLASH_CR_STRT; // start
323     while(FLASH->SR & FLASH_SR_BSY); // check if busy
324     __enable_irq();
325 }
326
327 void write_flash(uint32_t addr, uint32_t datai){
328     while(FLASH->SR & FLASH_SR_BSY); // check if busy
329     FLASH->CR |= FLASH_CR_PG;
330     FLASH->CR &= ~(0x3U << 8); // clear PSIZE bit 8:9
331     FLASH->CR |= (0x2 << 8);   // program PSIZE
332     *(volatile uint32_t*)addr = datai;
333     while(FLASH->SR & FLASH_SR_BSY); // check if busy
334 }
335
336 void read_flash(uint32_t addr){
337     while(FLASH->SR & FLASH_SR_BSY); // check if busy
338     FLASH->CR |= FLASH_CR_PG;
339     FLASH->CR &= ~(0x3U << 8); // clear PSIZE bit 8:9
340     FLASH->CR |= (0x2 << 8);   // program PSIZE
341     volatile uint32_t *bridge  = addr;
342     mem_datao= *bridge;
343     while(FLASH->SR & FLASH_SR_BSY); // check if busy
344 }
345
```

```
346 /**********************************************
347 * main code starts from here
348 **********************************************/
349 int main(void)
350 {
351
352  RCC->AHB1ENR |= 0x0000000B;
353     // enable SYSCFG clock (APB2ENR: bit 14) | enable TIM2 clock (bit0)
354     RCC->APB2ENR |= (1 << 14);                        //for ext interrupt
355     RCC->APB1ENR |= ((1 << 18) | (1 << 0));    // USART3 & timer
356     RCC->APB1ENR |= (1 << 17);                        //USART2
357
358     GPIOD->MODER &= 0x00FFFFFF;         //LEDs
359     GPIOD->MODER |= 0x55000000;
360
361     // set pin modes as alternate mode 7 (pins 2 and 3)
362     GPIOA->MODER &= 0xFFFFFF0F; // clear old values
363     GPIOA->MODER |= 0x000000A0; // Set pin 2/3 to alternate func. mode (0b10)
364     GPIOD->MODER |= (2 << 16); // gpiod alt func
365     // set pin modes as high speed
366     GPIOA->OSPEEDR |= 0x000000A0; // Set pin 2/3 to high speed mode (0b10)
367     GPIOD->OSPEEDR |= (2 << 16);
368     // choose AF7 for USART2 in Alternate Function registers
369     GPIOD->AFR[1] |= (0x7 << 0); // for pin PD8 for USART3 TX
370     GPIOA->AFR[0] |= (0x7 << 12); // for pin PA3 for USART2 RX
371
372
373     uint32_t source_adr = 0x44;         //68
374     uint32_t dest_adr = 0x05;           //5
375     DataTemplate = ((dest_adr << 16) | (source_adr << 8));
376
377
378     // Light up LEDs for 1 sec
379     GPIOD->ODR &= 0x0000;
380     GPIOD->ODR |= 0xF000;
381     delay_ms(1000000);
382     GPIOD->ODR &= 0x0000;
383
384     // * EXT INTERRUPT * //
385     SYSCFG->EXTICR[0] |= 0x00000000;
386     // Choose either rising edge trigger (RTSR) or falling edge trigger
    (FTSR)
387     EXTI->RTSR |= 0x00001;   // Enable rising edge trigger on EXTI0
388     EXTI->FTSR |= 0x00001;
389     // Mask the used external interrupt numbers.
390     EXTI->IMR |= 0x00001;    // Mask EXTI0
391     // Set Priority for each interrupt request
392     NVIC->IP[EXTI0_IRQn] = 0x10; // Priority level 1
393     // enable EXT0 IRQ from NVIC
394     NVIC_EnableIRQ(EXTI0_IRQn);
395
396
397     // * SysTick * //
398     /* set system clock to 168 Mhz */
399     set_sysclk_to_168();
400     // configure SysTick to interrupt every 21k ticks
401     // when SysClk is configured to 168MHz,
402     // SysTick will be running at 168Mhz/8 = 21Mhz speed
403     // passing 21000 here will give us 1ms ticks
```

```
404     // enable callback
405     init_systick(16777215, 1);
406     //10500000 => 0.5sn, 16777215 => max 0.8s
407
408
409     // * TIMER * //
410     // Timer clock runs at ABP1 * 2
411     //   since ABP1 is set to /4 of fCLK
412     //   thus 168M/4 * 2 = 84Mhz
413     // set prescaler to 83999
414     //   it will increment counter every prescalar cycles
415     // fCK_PSC / (PSC[15:0] + 1)
416     // 84 Mhz / 8399 + 1 = 10 khz timer clock speed
417     TIM2->PSC = 8399;
418     // Set the auto-reload value to 10000
419     //   which should give 1 second timer interrupts
420     TIM2->ARR = 500; // 0.05s
421     // Update Interrupt Enable
422     TIM2->DIER |= (1 << 0);
423     // enable TIM2 IRQ from NVIC
424     NVIC_EnableIRQ(TIM2_IRQn);
425     // Enable Timer 2 module (CEN, bit0)
426     TIM2->CR1 |= (1 << 0);
427
428
429     // * USART3 TX * //
430     // usart3 tx enable, RE bit 2
431     USART3->CR1 |= (1 << 3);
432     USART3->BRR |= (22 << 4);
433     USART3->BRR |= 13;
434     // usar3 word length M, bit 12
435     USART3->CR1 |= (0 << 12); // 0 - 1,8,n
436     // usart3 parity control, bit 9
437     USART3->CR1 |= (0 << 10); // 0 - no parity
438     //usart3 number of stop bits
439     USART3->CR2 |= (0 << 12);  // 0 - 1 stop bit
440     // enable usart3 - UE, bit 13
441     USART3->CR1 |= (1 << 13);
442     NVIC_EnableIRQ(USART3_IRQn);
443
444
445     // * USART2 RX * //
446     //usart3 rx enable, TE bit 3
447     USART2->CR1 |= (1 << 2);
448     //baud rate initialization
449     USART2->BRR |= (22 << 4);
450     USART2->BRR |= 13;
451     //usart3 word length M, bit 12
452     USART2->CR1 |= (0 << 12); // 0 - 1,8,n
453     //usart3 parity control, bit 9
454     USART2->CR1 |= (0 << 10); // 0 - no parity
455     //usart3 number of stop bits
456     USART2->CR2 |= (0 << 12);  // 0 - 1 stop bit
457     //enable usart2 - UE, bit 13
458     USART2->CR1 |= (1 << 13);
459     //enable rx interrupt
460     USART2->CR1 |= (1 << 5);
461     NVIC_EnableIRQ(USART2_IRQn);
462
```

```
463
464     // * IWDG * //
465     //Enable iwdg
466     IWDG->KR |= 0xcccc;
467     //remove iwdg register protection
468     IWDG->KR |= 0X5555;
469     //Arrange prescaler
470     IWDG->PR |= (3 << 0); // writing "011", enables divider 32 & 4096ms
471     //reload iwdg counter
472     IWDG->RLR = 0xfff; // load to max value (4096ms)
473     IWDG->KR |= 0xaaaa; //refresh the counter & disable wwdg
474
475
476     // * RESET RollingCode * //
477 //    unlock_flash();
478 //    erase_flash_sector3();
479 //    write_flash(KEYADDR, RC_in);
480 //    lock_flash();
481
482     // * SLEEP MODE * //
483     __enable_irq();
484     SCB->SCR |= (1 << 1); //Sleep on exit
485     __WFI();
486
487 }
488
489 void delay_ms(volatile uint32_t s)
490 {
491     tDelay = s;
492     while(tDelay != 0){
493         tDelay--;
494     }
495 }
496
497 void Decryption_rx()
498 {
499   struct AES_ctx ctx;
500   //AES_init_ctx_iv(&ctx, key, iv);
501   //AES_CBC_decrypt_buffer(&ctx, DataFbuf_rx, 16);
502   AES_init_ctx(&ctx, key);
503   AES_ECB_decrypt(&ctx, DataFbuf_rx);
504
505   for(int z=0; z<4; z++){
506         DataFrame_rx |= (DataFbuf_rx[z] << (8*z));
507   }
508
509   CommandLEDs();
510 }
511
512 void CommandLEDs(){
513
514   identifier_rx = (DataFrame_rx & 0xFF);
515   Rcode_rx = ((DataFrame_rx >> 24) & 0xFF);
516   //Rcode_rx = 5;
517
518   if(Rcode_rx == Rcode){
519         // identify the command
520         if(identifier_rx == 1){
521               GPIOD->ODR ^= (uint16_t)(1 << 12);     //GREEN
```

```
522                 identifier_rx = 0;
523                 DataFrame_rx = 0;
524                 //Rcode_rx = 0;
525           }
526           else if(identifier_rx == 2){
527                 GPIOD->ODR ^= (uint16_t)(1 << 13);     //ORANGE
528                 identifier_rx = 0;
529                 DataFrame_rx = 0;
530                 //Rcode_rx = 0;
531           }
532           else if(identifier_rx >= 16){
533                 GPIOD->ODR ^= (uint16_t)(1 << 15);     //BLUE
534                 identifier_rx = 0;
535                 DataFrame_rx = 0;
536                 //Rcode_rx = 0;
537           }
538     }
539     else{
540           //Error occured during transmission, iwdg reset
541           for(int s=0; s<20; s++){
542                 GPIOD->ODR ^= (uint16_t)(1 << 14);     //red
543                 delay_ms(250000);
544           }
545           identifier_rx = 0;
546           DataFrame_rx = 0;
547           //Rcode_rx = 0;
548           IWDG->KR |= 0X5555;
549           IWDG->RLR = 0x01;
550     }
551     //Rcode_rx = 0;
552   }
553
554
```

6 References

- STM32F407 Reference Manual
- ARMv7-M Architecture Reference Manual
- UM1472 User Manual
- PM0214 Programming Manual
- Cortex-M4 Technical Reference Manual
- RM0090 Reference Manual
- Cortex-M4 Generic User Guide
- https://www.st.com/content/ccc/resource/training/technical/product_training/d5/97/97/97/ef/b9/48/26/STM32L4_WDG_TIMERS_IWDG.pdf/files/STM32L4_WDG_TIMERS_IWDG.pdf/jcr:content/translations/en.STM32L4_WDG_TIMERS_IWDG.pdf