ELEC458-Embedded System

PROJECT #1-RemoteKeylessSystem

Group ID - Group-9
Ahmet SERDAR- 151024063
Ömer KONAN - 171024085

1.Introduction

a) Project Goal :

- Assembly programming fundamentals
- Understanding of stack operation
- Register usage awareness and gimmicks
- Subroutine usage for efficiency
- Familiarity of GNU toolchain
- Learn to use a debugger and pinpoint problems
- Read/write from/to GPIO pins
- Read/understand a small subset of morse code (numbers)

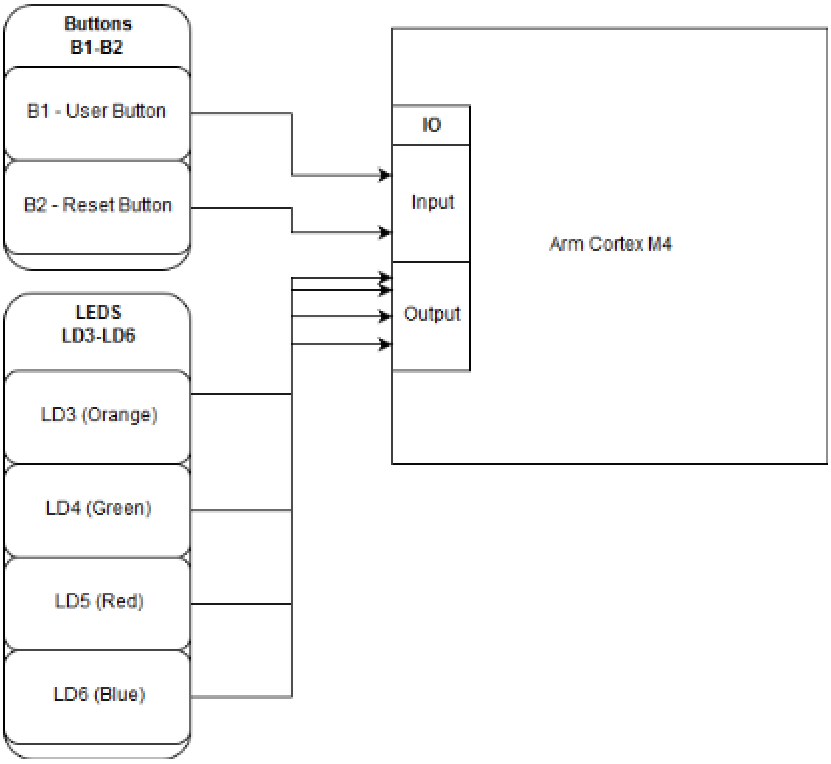
b) Our goals for the Project :

- Learn how to use a debugger and pinpoint problems
- Understanding of assembly operation (push,pop opreations, link register, branch)
- Read/write from/to GPIO pins
- Read/understand Manchester Encoding
- Produce algorithm suitable for project purpose
- Understanding encryption

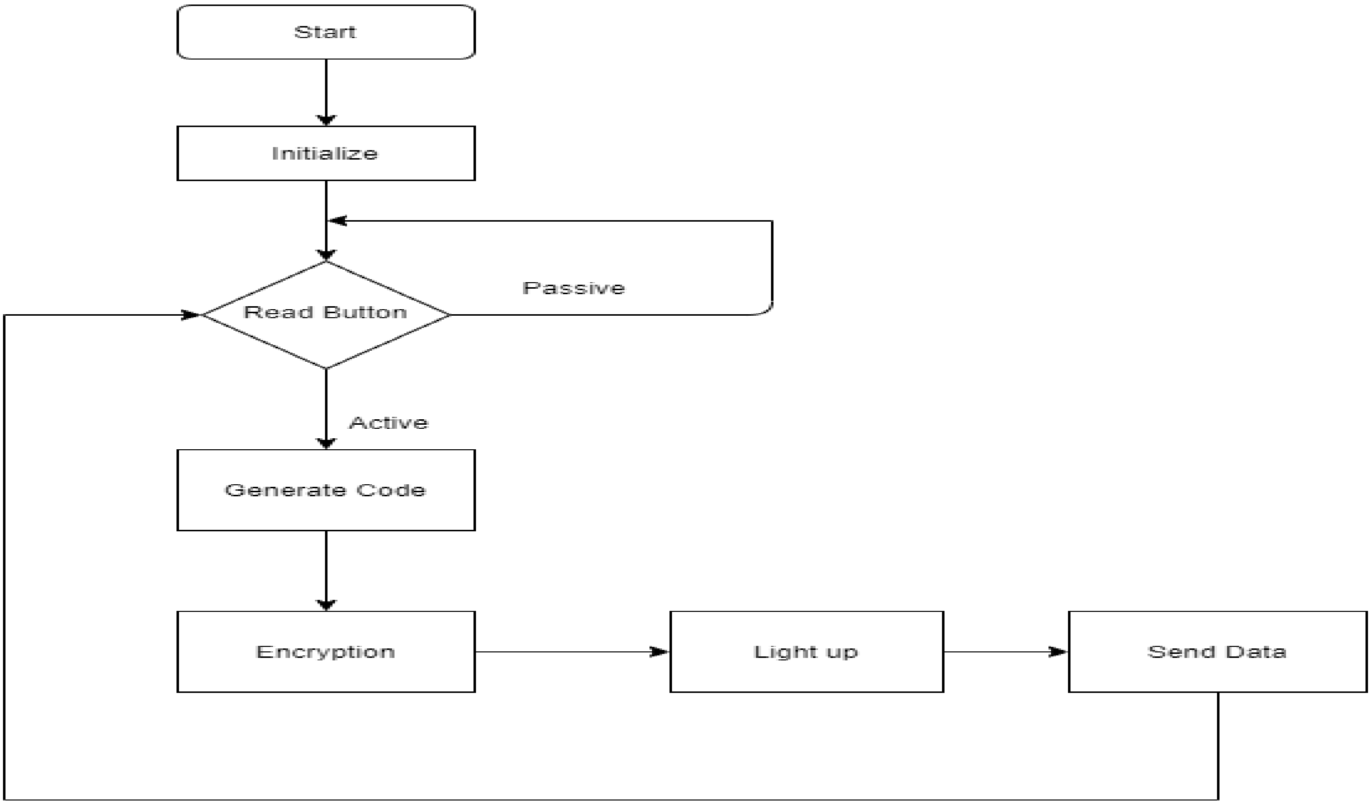
c) Load distribution:

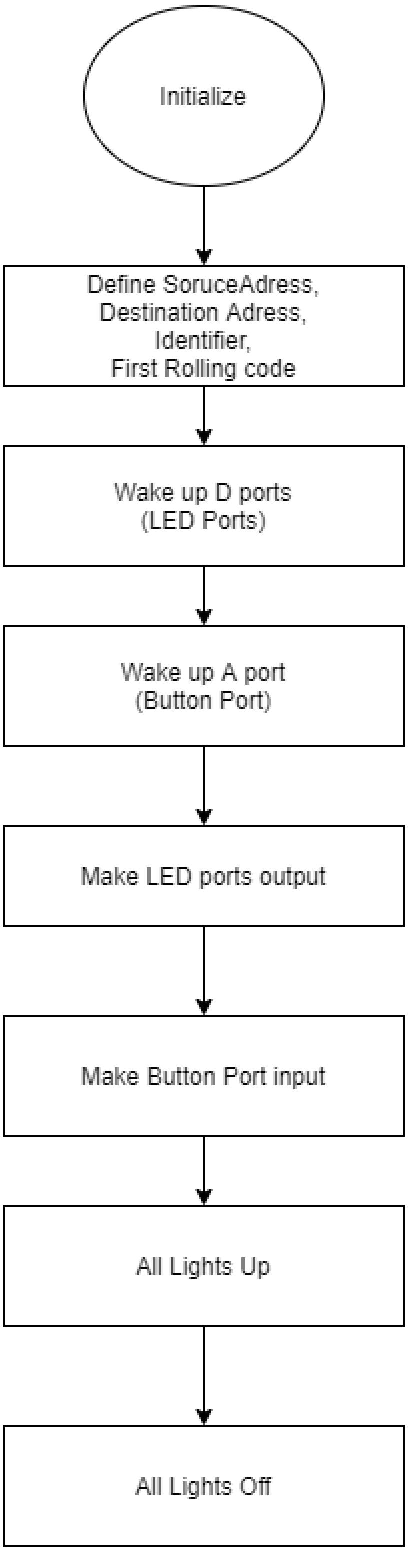
- **Block Diagram and Flowchart :**Ömer Konan
- **Testing :** Ahmet Serdar, Ömer Konan
- **Preparation of report :** Ahmet Serdar, Ömer Konan
- **Coding :**
  - **Main function :** Ahmet Serdar, Ömer Konan
  - **Polling to button:** Ahmet Serdar, Ömer Konan
  - **Led Lightining with Rolling Code:** Ömer Konan
  - **Delay Functions :** Ahmet Serdar
  - **Manchester Encoding:** Ahmet Serdar, Ömer Konan
  - **Combining codes :** Ahmet Serdar, Ömer Konan
  - **Encryption:** Ahmet Serdar, Ömer Konan
  - **Activating Leds and Button :** Ömer Konan

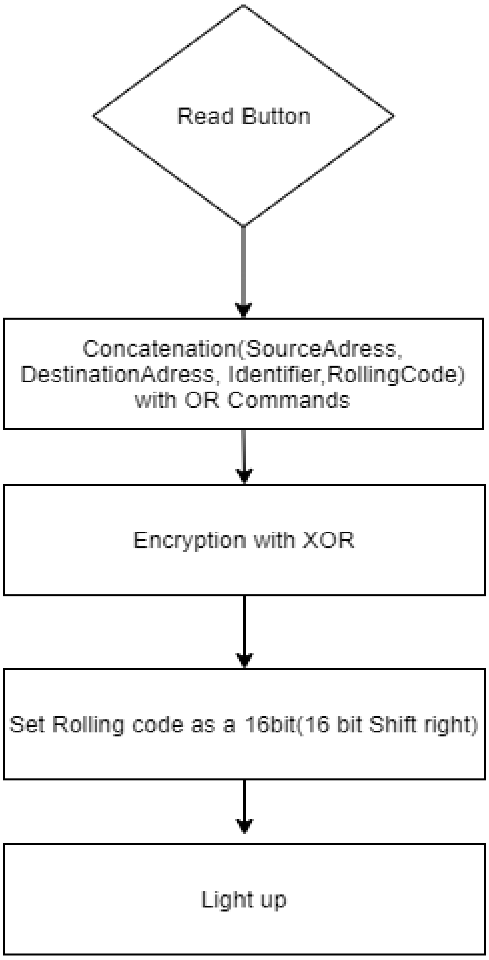
2. Block Diagram

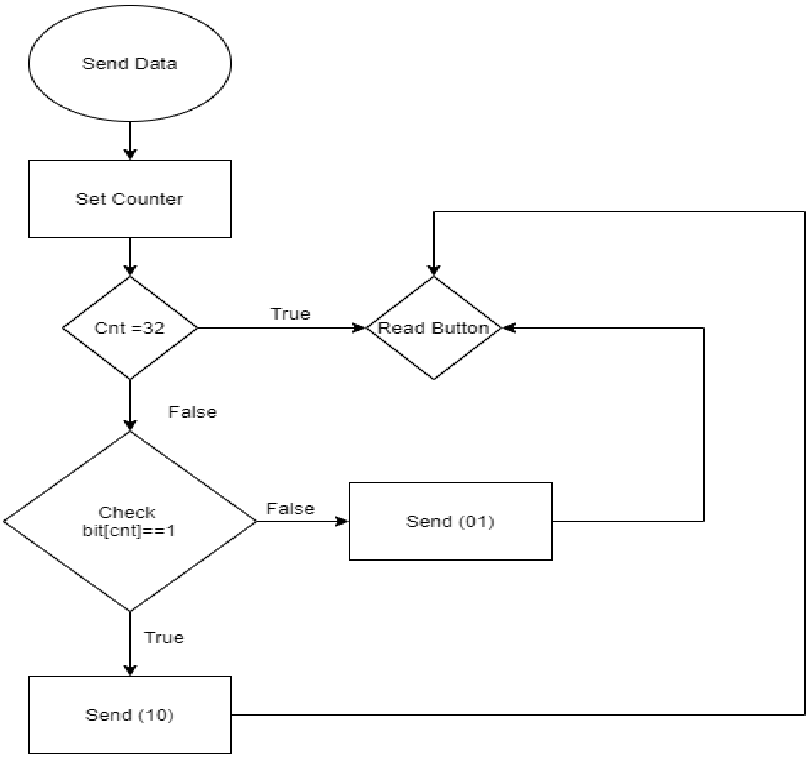


3. Flowchart




Initialize
Define SoruceAdress,
Destination Adress,
Identifier,
First Rolling code
Wake up D ports
(LED Ports)
Wake up A port
(Button Port)
Make LED ports output
Make Button Port input
All Lights Up
All Lights Off


Read Button
Concatenation(SourceAdress,
DestinationAdress, Identifier,RollingCode)
with OR Commands
Encryption with XOR
Set Rolling code as a 16bit(16 bit Shift right)
Light up


Send Data
Set Counter
Cnt =32
True
Read Button
False
Check
bit[cnt]==1
False
Send (01)
True
Send (10)

4. Design Overview

In this Project we tried to implement basic Remote Keyless System with ARM microproccessor. Firstly,we create an algorithm for whole system. Second step was to create a flowchart for the Project. After that we start to write assembly code. According to Project description it doesn’t need stack operation that is pop and push. During testing , we encountered with some problems which are mismatch registers, button control, delay duration and light up respect to Rolling code.

5. Conclusion

In this Project, we learned register addresses which are used Port Mode selection,using gpio pin,how to find fault with debugger,how to use pinpoint operation and than we learned usage of link register for connection of functions, how we activate leds and button.

6. Grade sheet

GEBZE TECHNICAL UNIVERSITY
DEPARTMENT OF ELECTRONIC ENGINEERING

ELEC 458 - Embedded System Design

Project 1

PROJECT NAME:-Remote Keyless System

STUDENT NAMES : Ahmet Serdar
: Ömer Konan

Technical objectives

Customer Satisfaction Objectives

7. Appendix

Assembly Code:

```
// STM32F4 Discovery - Assembly template
// Turns on an LED attached to GPIOD Pin 12
// We need to enable the clock for GPIOD and set up pin 12 as output.

// Start with enabling thumb 32 mode since Cortex-M4 do not work with arm mode
// Unified syntax is used to enable good of the both words...

// Make sure to run arm-none-eabi-objdump.exe -d prj1.elf to check if
// the assembler used proper instructions. (Like ADDS)
//f5 debug run
.thumb
.syntax unified
//.arch armv7e-m
// Constants
.equ         DELAY,         5000// Use at least 70000 due to see light
.equ         LIGHT_UP,      4000000
.equ   src_adrs,       0x00090000    // Source Address
.equ   dst_adrs,       0x0000A000      // Destination Address
.equ   idt,            0x00000001    // Identifier
.equ   roll_code,      0x30    // Rolling code init value
.equ   ENCRYPTION_KEY, 0x94949494  // 171024085 + 151024163 = 322048148  322048148
% 256 = 148  in hexadecimal form = 94
.equ   CNT,                     0
// Register Addresses
// You can find the base addresses for all peripherals from Memory Map section
// RM0090 on page 64. Then the offsets can be found on their relevant sections.

// RCC   base address is 0x40023800
//   AHB1ENR register offset is 0x30
.equ     RCC_AHB1ENR,   0x40023830 // RCC AHB1 peripheral clock reg (p age 180)

// GPIOD base address is 0x40020C00
//   MODER register offset is 0x00
//   ODR   register offset is 0x14
.equ     GPIOD_MODER,   0x40020C00 // GPIOD port mode register (page 281)
.equ     GPIOA_MODER,   0x40020000
.equ     GPIOD_ODR,     0x40020C14 // GPIOD output data register (page 283)
.equ     GPIOA_IDR,     0x40020010
// Start of text section
.section .text

       .long    __StackTop                 // Top of the stack. from linker script
       .long    _start +1                  // reset location, +1 for thumb mode


_start:
//PORT CLOCK SETTING
       ldr r6, = RCC_AHB1ENR
       ldr r5, [r6]
       //orr r5, 0x00000009   //
       orr r5, 0x00000009
       str r5, [r6]
```

```
//MODER SETTING
       ldr r6, = GPIOD_MODER//output
       ldr r5, [r6]
       and r5, 0x00FFFFFF
       orr r5, 0x55000000
       str r5, [r6]


       ldr r6, = GPIOA_MODER//input
       ldr r5, [r6]
       and r5, 0xFFFFFFFC
       str r5, [r6]


       ldr r6,= GPIOD_ODR @ Load GPIOD output data register
       ldr r5, [r6]
       orr r5, 0xF000
       str r5, [r6]

       ldr r10 , =LIGHT_UP
       bl delay

       ldr r5, [r6]
       and r5, 0x0000
       str r5, [r6]


       ldr r0, =roll_code
       b PressButton

delay:
       //
       subs r10,#1//1cycle
       cmp r10,#1//1cycle
       bne delay//2cycle
       bx lr// =totally 4 cycle,16mhz/4=4mhz, 4millioncycle=1second,400.000
cycle=100msec, 64bit send out for 100msec,100/64msec,6250cycle for 1 bit


PressButton://POLLING



       @ Button press check
       ldr r8, = GPIOA_IDR                 @ GPIOA_IDR adress copy to r6
       ldr r9, [r8]                        @ GPIOA_IDR data copy to r5
       and r9, 0x1                         @ Clear bits 15:1 for reading button
       press
       cmp r9, #1                          @ r5 == 1 => Button press | r5 == 0 =>
       Button not press
       beq    Frame_Calc
       b PressButton



Frame_Calc://calculate initial values
```

```
      ldr r4, = CNT
      and r0, r0, #255   @ mod 255 for rolling code
      lsl r0, #24              @ left shift for rolling code
      orr r1, r0               @ Add Rolling code r1's 31-24 bits
      orr r1, src_adrs   @ add src_adrs r1's 23-16 bits
      orr r1, dst_adrs   @ add destination address r1's 15-8 bits
      orr r1, idt              @ add identifier r1's 0-7 bits
      eor r1, ENCRYPTION_KEY
      b Light_up


Light_up:

      ldr r5, = GPIOD_ODR @ Load GPIOD output data register
      mov r2, r0               @ Copy rolling code to r2
      lsr r2, r2, #16
      str r2, [r5]
      b Send_Data

Send_Data:
      cmp r4, #32
      beq Finish_Transmission
      and r7, r1, #0x00000001
      lsr r1, r1, #1
      cmp r7, #0x1
      beq Hıgh_to_Low
      b Low_to_Hıgh

Hıgh_to_Low:
      add r4, #1
      ldr r6, = GPIOD_ODR @ Load GPIOD output data register
      ldr r5, [r6]
      and r5,0xF000
      orr r5,0x0001
      str r5,[r6]
      ldr r10 , =DELAY
      bl delay
      and r5,0xF000
      str r5,[r6]
      ldr r10 , =DELAY
      bl delay

      b Send_Data


Low_to_Hıgh:
      add r4, #1
      ldr r6, = GPIOD_ODR @ Load GPIOD output data register
      ldr r5, [r6]
      and r5,0xF000
      str r5,[r6]
      ldr r10 , =DELAY
      bl delay
      and r5,0xF000
      orr r5,0x0001
      str r5, [r6]
      ldr r10 , =DELAY
      bl delay
      b Send_Data
```

```
Finish_Transmission:
       lsr r0, r0, #24
       add r0, 0x1
       b PressButton
```