Microprocessor Lab1 Report

Kerim Turak

March 2021

Introduction

To complete laboratory assignment I follow this steps;

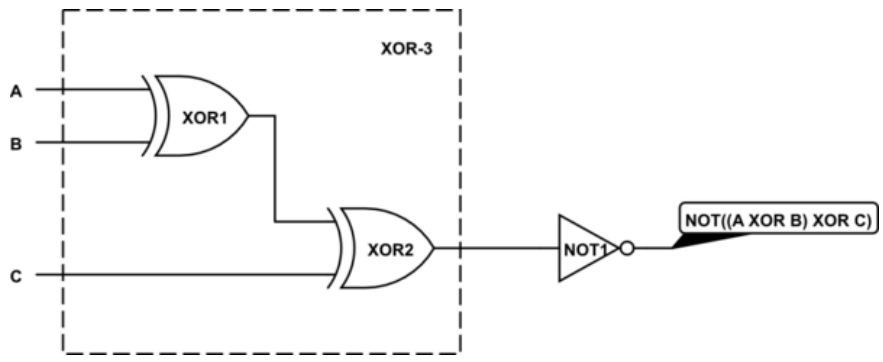
- Building logic diagram of parity checking
- Initializing necessary code blocks
- Writing main code blocks which perform the logical part of parity checking

a-) Building logic diagram of parity checking

First, to understand how to manage and solve this algorithm, I explored how I could build this system using the or-and-xor-lsl-lsr logic tools. I found a 3-bit two-stage xor gate logic diagram as the answer.

3-bit message			Odd parity bit generator (P)
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(Figure 1.0) 3-bit XOR



(Figure 1.1) 3-bit XOR Diagram

The system Works fine for even parity check, and I thought if I could get the reverse output of this logical process, I would find the answer for odd parity check. I think the four 3-bit conditions made up all the other conditions for the inputs, and this logic could apply to 8-bit (the register part I used) register data as well. Then I drew these four cases as follows.

1 1 1	110	100	000

(Figure 1.2) possible inputs

I decide to make some combination using logic gates for find the result. I shift through the left side all possible inputs variable by 1 bit and apply xor themselves as following.

1 1 1	110	100	000	→A
1 1 1	101	001	000	↘B
000	010	101	000	↗C = A xor B

(Figure 1.3) A XOR L-Shifted A

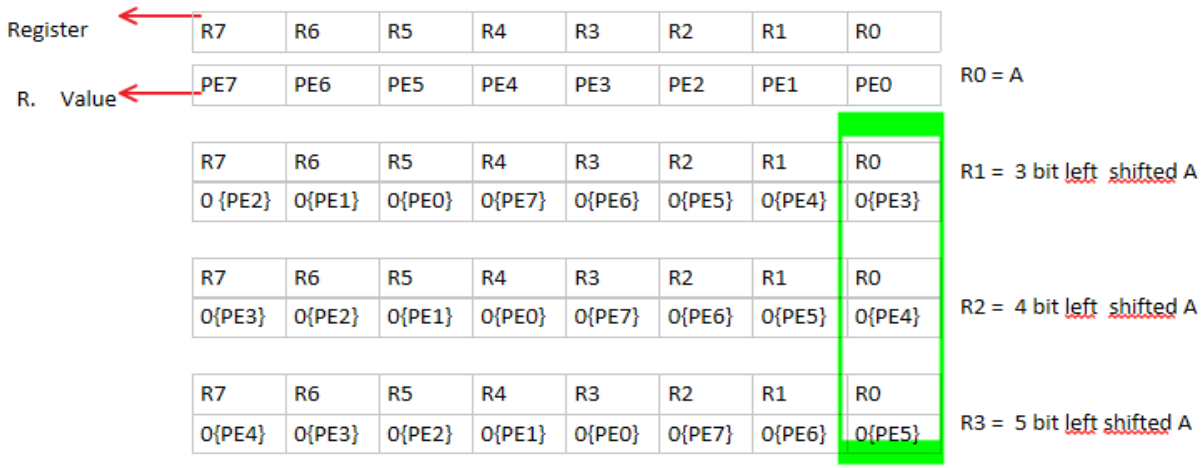
Output was not the desired result and I wanted to apply second one xor operation with the result of the previous operation and the inputs 1 bit shifted to the right.(figure1.2)

After the second xor gate I find even parity as following.

1 1 1	110	100	000	→A
1 1 1	101	001	000	↘B
				xor
000	011	101	000	↗C = A xor B
1 1 1	011	010	000	↗D
				xor
111	000	111	000	

(Figure 1.3) Second XOR Result

Actually we deal with converting row inputs to column ordered input to reduce all xor gates result to one bit by making this operations then we will shift this one bit result to register that holds value of PE2.End of the this processors if we apply same operation to 8-bit register then we get the result. Of course, we can find the result by shifting in different directions, only right or left shifts. The important thing is that PE3, PE4 and PE 5 are placed one on the other in the columns and the result of the process is shifted to PE2 at the end. This process showing as following;



(Figure 1.4) Operations

After this process we achieve desired colunm which is colored green. This mean we convert the ordered pin row datas to ordered column datas to make xor operation and return 1 bit to pin PE2. After three stage xor (for one direction shift) we need to transfer opeeration result to R2 (in my code it can be different) register according to this image.

B-) Initializing necessary code blocks

In this part, necessary port and unit addresses are arranged.

```
GPIO_PORTE_DATA_R  EQU 0x400243FC
GPIO_PORTE_DIR_R   EQU 0x40024400
GPIO_PORTE_DEN_R   EQU 0x4002451C
SYSCTL_RCGCGPIO_R  EQU 0x400FE608

GPIO_PORTE_AMSEL_R EQU 0x40024528;analog
GPIO_PORTE_PCTL_R  EQU 0x4002452C;general purpose
GPIO_PORTE_AFSEL_R EQU 0x40024420;alternative func.

GPIO_PE2           EQU 0x40024010 ; LED output
GPIO_PE3           EQU 0x40024020 ; SW1 input
GPIO_PE4           EQU 0x40024040 ; SW2 input
GPIO_PE5           EQU 0x40024080 ; SW3 input
```

- Data register address arranged
- Direction register address arrenged
- Port E enable
- Clock address is assigned to variable
- Analog pin = disable
- GPIO is arrenged
- Alternate function = disable
- Address of the Pin of the Port E get ready

And we need to code block thats should be read inputs data again and again and it should be return the result. According to my research in keil web site about subroutine or subroutine's function make this. To make this purpose I create a subroutine named **subroutine**

```
Subroutine
     ;code to run once that initializes PE4,PE2,PE1,PE0
     ; Enable ClOCK FOR PORT E
    LDR R1 , =SYSCTL_RCGCGPIO_R
    LDR R0 , [R1]
    ORR R0 ,  #0x10;0001.0000
    STR R0 , [R1]

    NOP         ; No operation for 3 cycles
    NOP         ; No operation for 3 cycles
    NOP         ; No operation for 3 cycles
```

End of the subroutine block.

```
Start
     BL  Subroutine
     LDR R5,=GPIO_PE2  ; Load address for PE2 into R5
     LDR R6,=GPIO_PE3  ; Load address for PE3 into R6
     LDR R7,=GPIO_PE4 ; Load address for PE4 into R7
     LDR R8,=GPIO_PE5 ; Load address for PE5 into R8
     MOV R4,#0         ; Move 0001 into R4
```

And end of the this subroutine block I add **Start** operant with **BL subroutine** because I want to make Pins assignment in last step. This means if we assign the pin values without doing the necessary operations in the previous lines, the code will not work because we have to enable the required Ports, such as clock and data.

3-)Writing main code blocks which perform the logical part of parity checking

In this part, I perform the logical part of the coding.

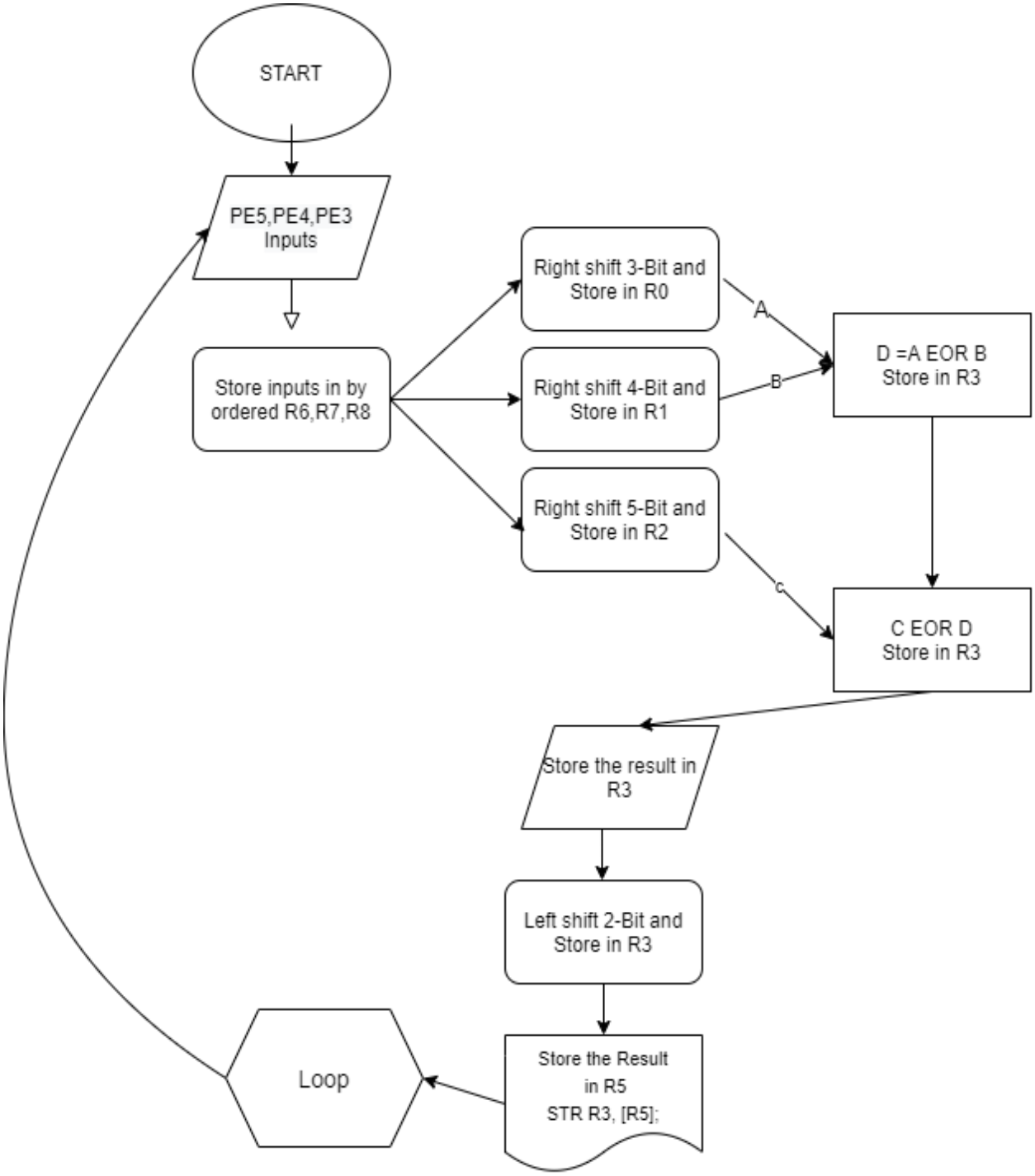
```
loop
    ;code that runs over and over
      LDR R0,[R6]       ; Load value from PE3 to R0
      MOV R0,R0,LSR #3  ; Right shift R0 by 3
      LDR R1,[R7]      ; Load value from PE4 to R1
      MOV R1,R1,LSR #4  ; Right shift R1 by 4
      LDR R2,[R8]      ; Load value from PE5 to R2
      MOV R2,R2,LSR #5  ; Right shift R2 by 5
      EOR R3,R0,R1      ; R0 XOR R1 -> R3
      EOR R3,R3,R2      ; R2 XOR R3 -> R3
      EOR R3,R3,R4      ; R3 XOR 1  -> R3
      MOV R3,R3,LSL #2  ; Left shift R3 by 2
      STR R3,[R5]       ; Store R3 in PE2 address
      B   loop


    ALIGN        ; make sure the end of this section is aligned
    END          ; end of file
```

Step by step

- Take value PE3 which stored in R6 then assign to R0
- Shift right R0 by 3
- Take value PE4 which stored in R7 then assign to R1
- Shift right R1 by 4
- Take value PE5 which stored in R8 then assign to R2
- Shift right R2 by 5
- R0 EOR R1>R3
- R2 EOR R1>R3
- R3 EOR 1>R3
- Left shift R3 by 2 to transfer the result to PE2’s register
- Store the value in R5
- Then loop

FLOWCHART


START
PE5,PE4,PE3
Inputs
Store inputs in by
ordered R6,R7,R8
Right shift 3-Bit and
Store in R0
Right shift 4-Bit and
Store in R1
Right shift 5-Bit and
Store in R2
A
B
C
D =A EOR B
Store in R3
C EOR D
Store in R3
Store the result in
R3
Left shift 2-Bit and
Store in R3
Store the Result
in R5
STR R3, [R5];
Loop

Pseudocode

assign required addresses to variables such as clock, data, pin addresses

Subroutine

-1*1-

-Enable clock for port E, disable analog func. ,enable the digital port……

Start (Code starts to run here)

First, go to the lines -1*1- where the necessary activations are made for the port E and make port E completely ready.

-2*2-

-Assign variables holding pin addresses to registers

Loop

-3*3-

- Shift the register holding the inputs right by 3 bits and assign it to a different register

-Shift the register holding the inputs right by 4 bits and assign it to a different register

-Shift the register holding the inputs right by 5 bits and assign it to a different register

-apply xor to first shifted value and second shifted value

-apply xor to first result value and third shifted value and return the result

To change odd to even or even to odd two changes can be modified

1) Changing the code

```
Start
    BL  Subroutine
    LDR R5,=GPIO_PE2
    LDR R6,=GPIO_PE3
    LDR R7,=GPIO_PE4
    LDR R8,=GPIO_PE5
    MOV R4,#1
```

With

```
Start
    BL  Subroutine
    LDR R5,=GPIO_PE2
    LDR R6,=GPIO_PE3
    LDR R7,=GPIO_PE4
    LDR R8,=GPIO_PE5
    MOV R4,#0
```

2)Adding extra eor gate to end point of code

```
loop
    ;code that runs ove
      LDR R0,[R6]
      MOV R0,R0,LSR #3
      LDR R1,[R7]
      MOV R1,R1,LSR #4
      LDR R2,[R8]
      MOV R2,R2,LSR #5
      EOR R3,R0,R1
      EOR R3,R3,R2
      EOR R3,R3,R4
      MOV R3,R3,LSL #2
      STR R3,[R5]
      B   loop
```

With

```
loop
    ;code that runs ove
      LDR R0,[R6]
      MOV R0,R0,LSR #3
      LDR R1,[R7]
      MOV R1,R1,LSR #4
      LDR R2,[R8]
      MOV R2,R2,LSR #5
      EOR R3,R0,R1
      EOR R3,R3,R2
      EOR R3,R3,R4
      MOV R3,R3,LSL #2
      EOR R3,R3,0x04
      STR R3,[R5]
      B   loop
```

MY FULL CODE

```
GPIO_PORTE_DATA_R  EQU 0x400243FC
GPIO_PORTE_DIR_R   EQU 0x40024400
GPIO_PORTE_DEN_R   EQU 0x4002451C
SYSCTL_RCGCGPIO_R  EQU 0x400FE608
GPIO_PORTE_AMSEL_R EQU 0x40024528;analog
GPIO_PORTE_PCTL_R  EQU 0x4002452C;general purpose
GPIO_PORTE_AFSEL_R EQU 0x40024420;alternative func.
GPIO_PE2          EQU 0x40024010 ; LED output
GPIO_PE3          EQU 0x40024020 ; SW1 input
GPIO_PE4          EQU 0x40024040 ; SW2 input
GPIO_PE5          EQU 0x40024080 ; SW3 input
      THUMB
      AREA    DATA, ALIGN=2
;global variables go here
     ALIGN
     AREA    |.text|, CODE, READONLY, ALIGN=2
     EXPORT  Start
Subroutine
    ;code to run once that initializes PE4,PE2,PE1,PE0
              ; Enable ClOCK FOR PORT E
              LDR R1 , =SYSCTL_RCGCGPIO_R
              LDR R0 , [R1]
              ORR R0 ,  #0x10;0001.0000
              STR R0 , [R1]
              NOP        ; No operation for 3 cycles
              NOP                              ; No operation for 3 cycles
              NOP                              ; No operation for 3 cycles
              ;Disable analog functionality
              LDR R1, =GPIO_PORTE_AMSEL_R
              LDR R0, [R1]
              BIC R0, R0, #0xFF; clear bit
              STR R0, [R1]
              ;Configure as GPIO
              LDR R1, =GPIO_PORTE_PCTL_R
              LDR R0, [R1]
              MOV R2, #0xFFFF
              BIC R0, R0, R2
              STR R0, [R1]
              ;Direction register
```

```
        LDR R1, =GPIO_PORTE_DIR_R
        LDR R0 , [R1]
        ORR R0 , #0x04 ;0000_0100 PE4 PE3 PE2 PE1 PE0 1=OUTPUT 0=INPUT
        STR R0 , [R1]
        ;Disable alternate functions
        LDR R1, =GPIO_PORTE_AFSEL_R
        LDR R0 , [R1]
        BIC R0 , #0xFF
        STR R0 , [R1]
        ;Enable the digital port
        LDR R1, =GPIO_PORTE_DEN_R
        LDR R0 , [R1]
        ORR R0, R0, #0x3C
        STR R0 , [R1]
        BX  LR
Start
        BL  Subroutine
    LDR R5,=GPIO_PE2  ; Load address for PE2 into R5
    LDR R6,=GPIO_PE3  ; Load address for PE3 into R6
    LDR R7,=GPIO_PE4 ; Load address for PE4 into R7
    LDR R8,=GPIO_PE5 ; Load address for PE5 into R8
    MOV R4,#0         ; Move 0000 into R4
loop
    ;code that runs over and over
      LDR R0,[R6]       ; Load value from PE3 to R0
      MOV R0,R0,LSR #3  ; Right shift R0 by 3
      LDR R1,[R7]       ; Load value from PE4 to R1
      MOV R1,R1,LSR #4  ; Right shift R1 by 4
      LDR R2,[R8]       ; Load value from PE5 to R2
      MOV R2,R2,LSR #5  ; Right shift R2 by 5
      EOR R3,R0,R1      ; R0 XOR R1 -> R3
      EOR R3,R3,R2      ; R2 XOR R3 -> R3
      EOR R3,R3,R4      ; R3 XOR 1  -> R3
      MOV R3,R3,LSL #2  ; Left shift R3 by 2
      STR R3,[R5]       ; Store R3 in PE2 address
      B   loop

    ALIGN        ; make sure the end of this section is aligned
    END          ; end of file
```