



California State University, Channel Islands (CSUCI)
Department of Computer Science

COMP-462: Embedded Systems
Lab Report
Fall 2019

Lab Number: Lab 1

Lab Topic:

Design and Simulation of an Odd Parity Bit Checker Circuit

Student Name: Marc Julian Jamerlan
Student ID: 001956436
Student Major: B.S. Computer Science

I. Objectives

This lab aims to give students more experience with using Keil μ Vision and coding with assembly logic by designing code that calculates the parity of a 3-bit binary string. The goal is to be able to simulate a circuit that uses switches to signify bits and an LED that turns on or off depending on how many switches have been flipped.

II. Introduction

In computing, the parity of a binary string is determined by the number of “on” bits (in this case, the number of 1s in data) the string contains and whether the sum of these bits is odd or even, giving the string odd parity or even parity, respectively. For this lab, we implemented code in Keil μ Vision that simulated a microcontroller circuit acting as an odd-bit detection system. The system uses three switches in negative logic (if the switch is pressed, 0 is returned) as inputs that represent a three-bit binary string and an LED in positive logic (LED is turned on at 1 and off at 0) to represent the parity bit. If the system detects an odd number of 1s (two switches or no switches pressed), the LED is turned on. Conversely, the LED is turned off if there are an even number of 1s or no 1s (one switch or all switches pressed).

Input 1	Input 2	Input 3	Output	# of 1s (Odd/Even)	LED Status (ON/OFF)
0	0	0	0	Even	OFF
0	0	1	1	Odd	ON
0	1	0	1	Odd	ON
0	1	1	0	Even	OFF
1	0	0	1	Odd	ON
1	0	1	0	Even	OFF
1	1	0	0	Even	OFF
1	1	1	1	Odd	ON

Figure 1: Truth table of an odd parity detection system for a three-bit string

For this lab we used an assembly code skeleton provided by the University of Texas’ embedded systems course, with our modifications to the code having been the implementation of the actual parity checking loop as described in the Procedure section of the report.

III. Procedure

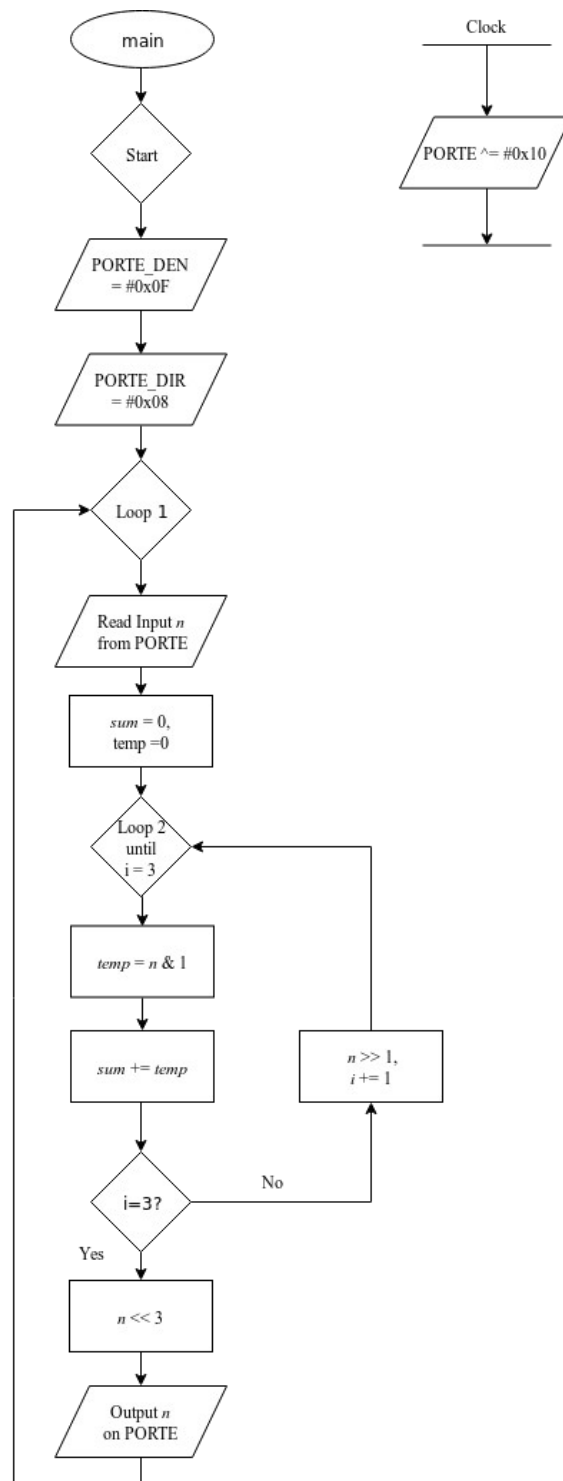


Figure 2: Flowchart of parity calculation

1. Designed a flowchart and pseudocode for parity calculation.
2. Used the code skeleton provided in Lab1_EE319K and implemented the algorithm in Figure 2.
3. Algorithm flow summary: Periodically take input from Port E and calculate parity of the first three bits. Then, send back to Port E as output. Repeat for the entire operation of the system.
4. After the code was implemented, the circuit was simulated using code provided in the project from Lab1_EE319K in Keil μ Vision.

IV. Problems

Aside from the challenge of designing an algorithm for checking the parity of a bit string, the only other problem arose from attempting to send output to GPIO_PORTE_DIR_R rather than through GPIO_PORTE_DATA_R. The logic of the algorithm was sound, but this lab taught me to make sure that I had initialized to the correct ports and to the correct addresses.

V. Results

Pseudocode:

Start

 Initialize Port E clock to start

 Wait for clock to start

 Initialize PE0-3 to digital I/O

 Initialize PE0-2 as input, PE3 as output

loop

 Read input from Port E

 sum, temp = 0

 while(i < 3)

 temp = input AND 1

 sum += input

 input >> 1

 i += 1

 input << 3

 Send input to Port E

 goto loop

Assembly code:

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

```
THUMB
```

```
AREA DATA, ALIGN=2
```

```
;global variables go here
```

```
ALIGN
```

```
AREA |.text|, CODE, READONLY, ALIGN=2
```

```
EXPORT Start
```

```
Start
```

```
;PORT INITIALIZATION
```

```
LDR R1, =SYSCTL_RCGCGPIO_R
```

```
LDR R0, [R1]
```

```
ORR R0, R0, #0x10 ; turns on clock at port e
```

```
STR R0, [R1]
```

```
NOP
```

```
NOP
```

```
LDR R1, =GPIO_PORTE_DEN_R
```

```
MOV R0, #0x0F ; digital I/O
```

```
STR R0, [R1]
```

```
LDR R1, =GPIO_PORTE_DIR_R
```

```
MOV R0, #0x08 ; PE3 output, all other pins input
```

```
STR R0, [R1]
```

```
loop
```

```
;PARITY CALCULATION
```

```
LDR R0, =GPIO_PORTE_DATA_R
```

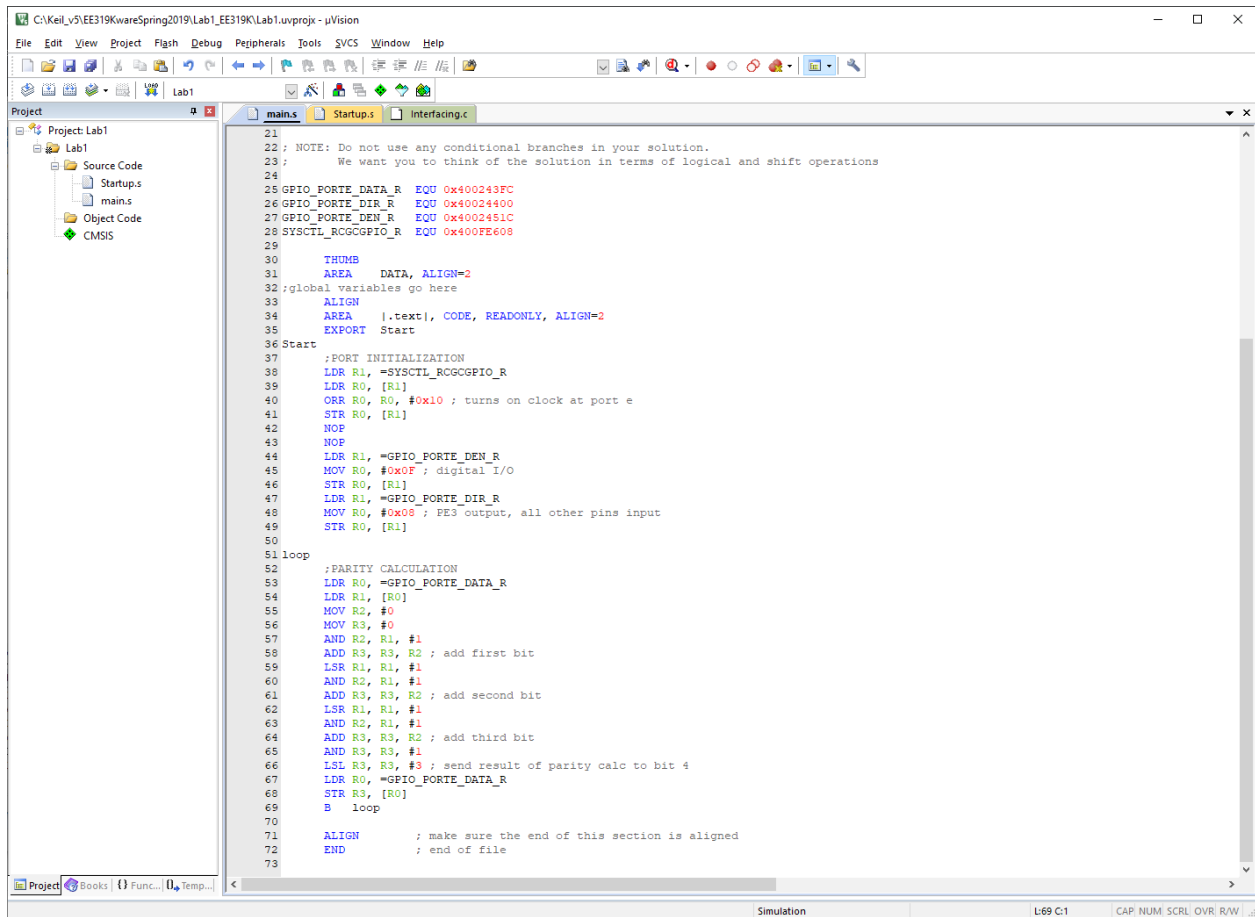
```

LDR R1, [R0]
MOV R2, #0
MOV R3, #0
AND R2, R1, #1
ADD R3, R3, R2 ; add first bit
LSR R1, R1, #1
AND R2, R1, #1
ADD R3, R3, R2 ; add second bit
LSR R1, R1, #1
AND R2, R1, #1
ADD R3, R3, R2 ; add third bit
AND R3, R3, #1
LSL R3, R3, #3 ; send result of parity calc to bit 4
LDR R0, =GPIO_PORTE_DATA_R
STR R3, [R0]
B    loop

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

```

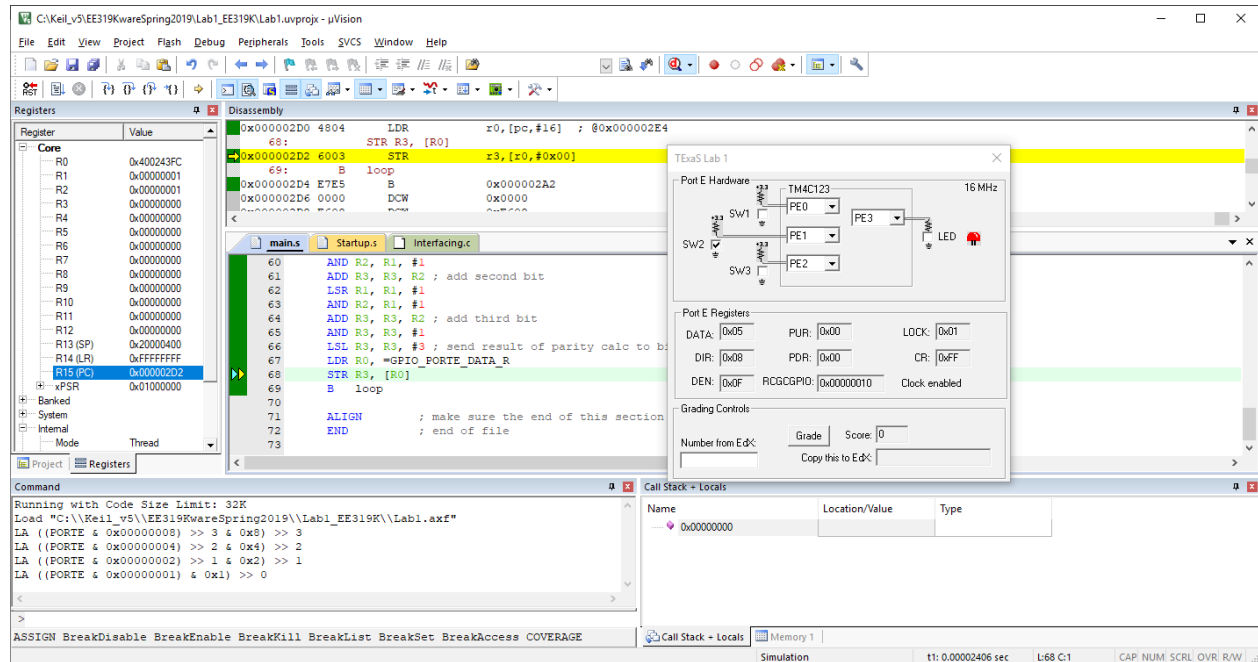
Screenshot of code:



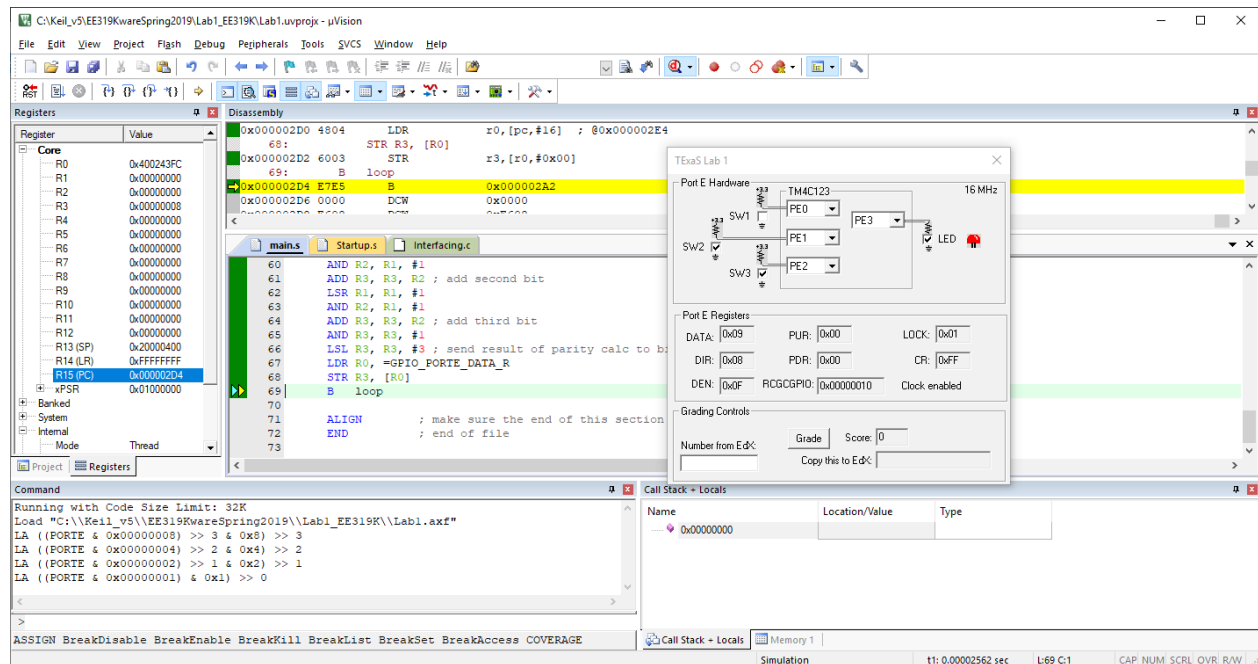
```
21
22 ; NOTE: Do not use any conditional branches in your solution.
23 ; We want you to think of the solution in terms of logical and shift operations
24
25 GPIO_FORTE_DATA_R EQU 0x400243FC
26 GPIO_FORTE_DIR_R EQU 0x40024400
27 GPIO_FORTE_DEN_R EQU 0x4002451C
28 SYSTCL_RCGCGPIO_R EQU 0x400FE608
29
30 THUMB
31 AREA DATA, ALIGN=2
32 ;global variables go here
33 ALIGN
34 AREA |.text|, CODE, READONLY, ALIGN=2
35 EXPORT Start
36 Start
37 ;PORT INITIALIZATION
38 LDR R1, =SYSTCL_RCGCGPIO_R
39 LDR R0, [R1]
40 ORR R0, R0, #0x10 ; turns on clock at port e
41 STR R0, [R1]
42 NOP
43 NOP
44 LDR R1, =GPIO_FORTE_DEN_R
45 MOV R0, #0x0F ; digital I/O
46 STR R0, [R1]
47 LDR R1, =GPIO_FORTE_DIR_R
48 MOV R0, #0x08 ; PE3 output, all other pins input
49 STR R0, [R1]
50
51 loop
52 ;PARITY CALCULATION
53 LDR R0, =GPIO_FORTE_DATA_R
54 LDR R1, [R0]
55 MOV R2, #0
56 MOV R3, #0
57 AND R2, R1, #1
58 ADD R3, R3, R2 ; add first bit
59 LSR R1, R1, #1
60 AND R2, R1, #1
61 ADD R3, R3, R2 ; add second bit
62 LSR R1, R1, #1
63 AND R2, R1, #1
64 ADD R3, R3, R2 ; add third bit
65 AND R3, R3, #1
66 LSL R3, R3, #3 ; send result of parity calc to bit 4
67 LDR R0, =GPIO_FORTE_DATA_R
68 STR R3, [R0]
69 B loop
70
71 ALIGN ; make sure the end of this section is aligned
72 END ; end of file
73
```

Screenshots of results of simulation:

OFF



ON



Final result was a system able to correctly take three inputs, calculate parity, and output an LED signal.