UNIVERSITY OF WATERLOO

ECE750 Real-time Embedded System

Lab #3

Name: Shaocong Ren

Student number: 20478300

E-mail: s7ren@uwaterloo.ca

Q1. Explain how your polling implementation detects rising edges on a GPIO pin.

A1. In the infinite while loop I read the state of GPIO Port B Pin 0 and record this value in the variable **curFlag**. At the end of the loop assign this value to variable **preFlag** which is the history value in next loop. While in the body of the loop, if **curFlag** equals 1 and **preFlag** equals 0, then a rising edge of GPIO Port B Pin 0 is detected. Following is the code of while loop:

```
while(1){
    curFlag = (GPIOB->IDR & (0x1<<0)); // read the current value of PortB Pin0
    if(curFlag == 1 && preFlag == 0){// condition to evaluate rising edge
        //These two clauses are to generate rising edges.
        //set bit to 1
        GPIOA->ODR|=(0x1<<1);
        //set bit to 0
        GPIOA->ODR&=~(0x1<<1);
    }
    preFlag = curFlag; //record history value to be used in next loop
}
```

Q2. Compare the observed latency (time) and variance of polling vs. interrupts. Explain the results.

A2:

	Polling	Interrupts
Mean	0.967857 μs	1.023810 μs
Median	0.964286 μs	1.000000 μs
Mode	0.952381 μs	1.000000 μs
Variance	0.001603 μs	0.010771 μs

After the rising edge of PORTB Pin0 is detected, the program will execute other reading I/O, variables assignment and function jumping, so in practice there is usually latency between detection and response. From the value of Mean, Median and Mode of the above chart, it can be seen that general latency of polling is lower than that in interrupts.

Specifically, in this polling program, there is no other interrupt to preempt the CPU, and the polling loop is the only one job to do, such that the time within a loop is short. In the interrupt program, otherwise, after EXIT Line0 detects the rising edge, the program will jump to the ISR function, which needs to push program pointer into the stack. The operation of push and pop of stack wastes much time. This is the

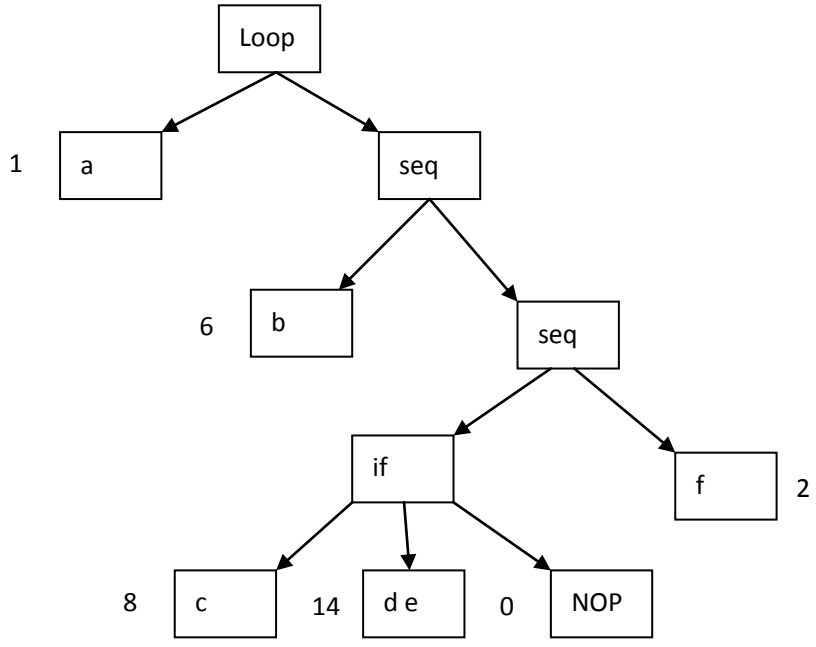
reason why the latency of interrupt is a little longer than polling.

Q3. What is the worst case execution time (in terms of number of assembly instructions) between a rising edge arriving on the GPIO pin and it being echoed back by your polling loop implementation? Show how you arrived at your result (the compiled assembly is in main.lst).

A3:

The worst case execution time between a rising edge arriving on the GPIO pin and it being echoed back in this program means the whole execution time within the while loop. The precondition of this problem is that the rising edge in GPIOB Pin 0 is coming in current polling loop. Detecting previous state of GPIOB Pin 0 is 0 and current state is 1 is just the method. In this way, we can assert that the number of instructions of the while loop is the WCET which equals 31. The figure shows the parse tree of the while loop, which can shows the WCET clearly.

```
a     while(1){
b         curFlag = GPIOB->IDR & (0x1<<0);
c         if((curFlag)    & (0 == preFlag)){
d             GPIOA->ODR|=(0x1<<1);
e             GPIOA->ODR&=~(0x1<<1);
          }
f         preFlag = curFlag;
    }
```



WCET = 1 + 6 + 8 + 14 + 2 = 31

```
    while(1){
        curFlag = GPIOB->IDR & (0x1<<0);
800020a:    f44f 6380       mov.w   r3, #1024 ; 0x400
800020e:    f2c4 0302       movt    r3, #16386      ; 0x4002
8000212:    691b            ldr     r3, [r3, #16]
8000214:    b2db            uxtb    r3, r3
8000216:    f003 0301       and.w   r3, r3, #1
800021a:    71bb            strb    r3, [r7, #6]
        if((curFlag)    & (0 == preFlag)){
800021c:    79ba            ldrb    r2, [r7, #6]
800021e:    79fb            ldrb    r3, [r7, #7]
8000220:    2b00            cmp     r3, #0
8000222:    bf14            ite     ne
8000224:    2300            movne   r3, #0
8000226:    2301            moveq   r3, #1
8000228:    4013            ands    r3, r2
800022a:    2b00            cmp     r3, #0
800022c:    d017            beq.n   800025e <main+0x6e>

            //Following two clauses are to generate rising edges.
            //set bit i to 1
            GPIOA->ODR|=(0x1<<1);
800022e:    f04f 0300       mov.w   r3, #0
8000232:    f2c4 0302       movt    r3, #16386      ; 0x4002
8000236:    f04f 0200       mov.w   r2, #0
800023a:    f2c4 0202       movt    r2, #16386      ; 0x4002
800023e:    6952            ldr     r2, [r2, #20]
8000240:    f042 0202       orr.w   r2, r2, #2
8000244:    615a            str     r2, [r3, #20]
            //set bit i to 0
            GPIOA->ODR&=~(0x1<<1);
8000246:    f04f 0300       mov.w   r3, #0
800024a:    f2c4 0302       movt    r3, #16386      ; 0x4002
800024e:    f04f 0200       mov.w   r2, #0
8000252:    f2c4 0202       movt    r2, #16386      ; 0x4002
8000256:    6952            ldr     r2, [r2, #20]
8000258:    f022 0202       bic.w   r2, r2, #2
800025c:    615a            str     r2, [r3, #20]
        }
        preFlag = curFlag;//(GPIOB->IDR & (0x1<<0));
```

```
800025e:    79bb        ldrb  r3, [r7, #6]
8000260:    71fb        strb  r3, [r7, #7]
    }
8000262:    e7d2        b.n   800020a <main+0x1a>
```