# CSE 379 Lab 5: Interrupts

Daniel Jin (djin3)
Motaz Khalifa (motazkha)

---

## 1) Purpose of the Program

### a) Objective

The objective of this program is to record and update the amount of times that the keyboard and switches are pressed. Each switch out of the four switches has its own counter. Pressing “q” (lowercase) on the keyboard will quit the program. This program utilizes interrupts to receive button or key presses and update the counters in PuTTy in real time.

### b) Debugging Steps

To make sure we are reading inputs from the keyboard and switches properly, we put breakpoints at the beginning of the handlers. We want to make sure that the program can correctly find the number of the switch that was pressed, the flag is correctly changed when “q” is pressed, and the counters are retaining their values every interrupt. The memory addresses we monitored while debugging was 0x400073FC (Port D’s data register) and the address that our flag was stored in. The other debugging strategy that we ran was the standard breakpoint and stepping over.

### c) How To Run The Program

Open PuTTy using the same process as previous labs. In Code Composure Studios, click Debug → Resume. You will see — in PuTTy — the counters initialized to zero. You will find that if you press any of the switches, the corresponding switch counter displayed in PuTTy will update itself in real time. Similarly, the same will happen if you press the keyboard; there are some keys with special functions that when pressed, will not update the keyboard’s counter. Such keys include SHIFT, CTRL, and ALT.

### d) Outside References

The Tiva™ TM4C123GH6PM Microcontroller Data Sheet was referenced for information on the registers necessary for this lab and the purpose of each bit within these registers.

## 2) Division of Work

Daniel wrote *interrupt_init* and its subroutines (*uart_interrupt_init* and *gpio_interrupt_init*). Motaz wrote *lab_5* and the majority of both *UART0_Handler* and *Switches_Handler* (the interrupt clears at the bottom of these subroutines were written by Daniel). *print_update* was written by Motaz. *uart_init*, *output_string, initiate_push_btns, output_newline, div_and_mod,* and *convert_to_ASCII_updated* were taken from previous labs' library files; some of these subroutines were slightly modified.

## 3) Logic

### a) Summary of Main Program Flowchart

All registers/pins necessary for this lab, the flag, and the counters are initialized. The program goes into an infinite loop, while constantly checking the flag. If the flag is 1, the program quits. Otherwise, the program keeps looping until an interrupt brings the program to a handler. If the program goes to the UART handler, the keyboard counter is increased. If the program goes to the Switches handler, the program finds which switch was pressed, and increases the corresponding and appropriate switch counter. After a counter is increased, the interrupt is promptly cleared and the program returns to the infinite loop.

### b) Summary of Subroutine Flowcharts

*i) lab_5*

This subroutines initializes everything that needs to be initialized, then prints out the display (instructions and counters) to PuTTy. Then, this subroutine goes into an infinite loop where it checks the flag in every iteration. The program exits the infinite loop either to a handler or to the end of the program if the flag is a 1.

*ii) interrupt_init*

*interrupt_init* calls two subroutines: *uart_interrupt_init* and *gpio_interrupt_init.*

*uart_interrupt_init* initializes UART0's NVIC and UART Interrupt Mask Register.

*gpio_interrupt_init* initializes Port D in the GPIO NVIC, GPIOIS, GPIOIBE, GPIOIEV, and GPIOIM.

*iii) UART0_Handler*

This handler reads the character pressed on the keyboard. If the character is lowercase "q" then this subroutine changes the flag to a 1. If the character is anything else, the keyboard counter is incremented, then the string of updated counters is outputted to PuTTy. The UART interrupt is then cleared, and the handler returns to the interrupted instruction.

*iv) Switches_Handler*

This handler checks to see which switch is pressed, and increments the corresponding counter. The string of updated counters is then outputted to PuTTy. The GPIO interrupt is then cleared, and the handler returns to the interrupted instruction.

*v) convert_to_ASCII_updated*

This subroutine takes in a number and converts it to its corresponding ASCII value and outputs it. The purpose of this subroutine is to output a decimal value for the user.

*vi) print_update*

This subroutine prints a carriage return to PuTTy, then prints out all the counters with spaces in between, effectively overwriting the previous values.

## c) Registers and Addresses

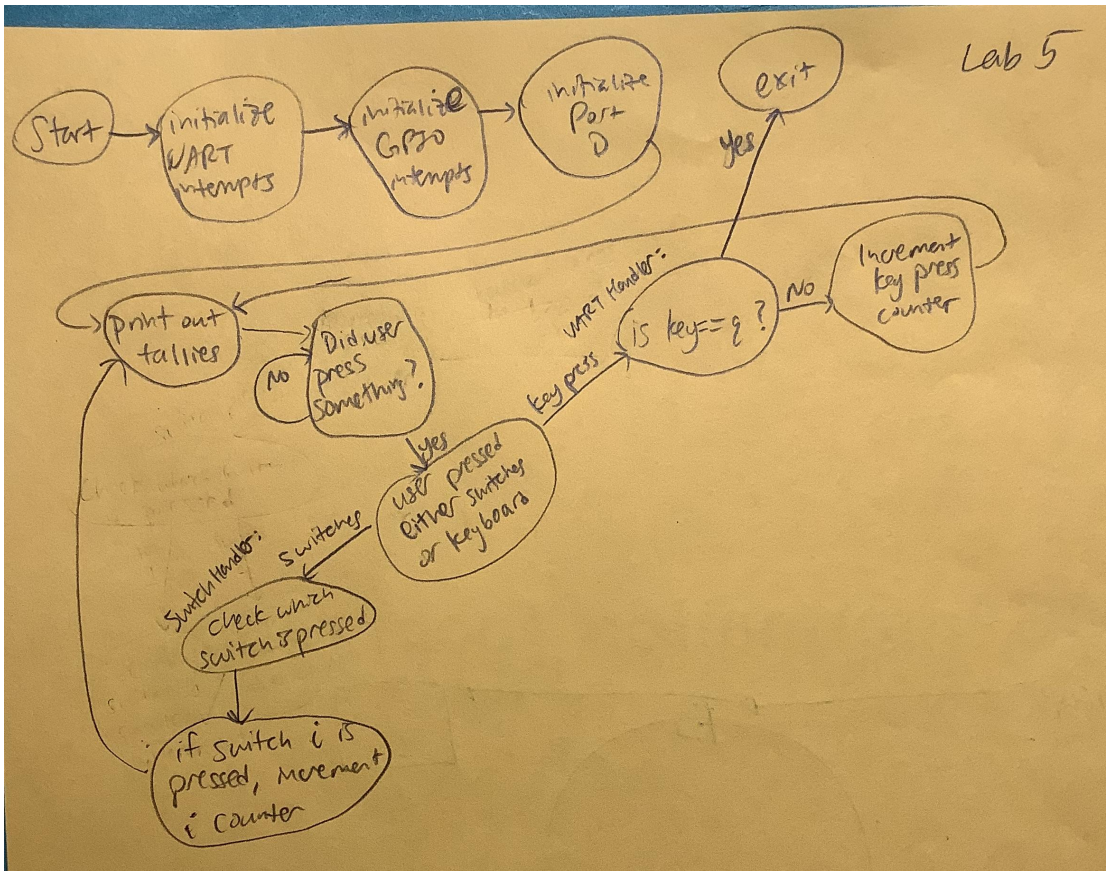
The necessary registers needed for the initialization of UART interrupts are the UART NVIC (0xE000E100) and the UART Interrupt Mask Register (UARTIM; offset 0x038). The RTIM bit in UARTIM (bit 6) and bit 5 in the NVIC enables the interrupt. UART Interrupt Clear (UARTICR; offset 0x044) is the interrupt clear register. The RXIC (bit 4) is needed to clear the corresponding interrupt. The base address for UART0 is 0x4000C000.

The necessary registers needed for the switch/GPIO interrupt initialization are the GPIO NVIC (0xE000E100), GPIO Interrupt Sense (GPIOIS; offset 0x404), GPIO Interrupt Both Edges (GPIOIBE; offset 0x408), GPIO Interrupt Event (GPIOIEV; offset 0x40C) and GPIO Interrupt Mask (GPIOIMl offset 0x410). Writing a 0 to GPIOIS sets edge sensitivity, a 0 to GPIOIBE lets GPIOIEV set either falling or rising edge (0 or 1 respectively) and a 1 to GPIOIM enables triggering. GPIO Interrupt Clear (GPIOICR; offset 0x41C) is the GPIO interrcupt clear register; writing a 1 to its pins will clear interrupts in GPIOIS and GPIOIM. The base address of GPIO Port D is 0x40007000.

# 4) Flowcharts

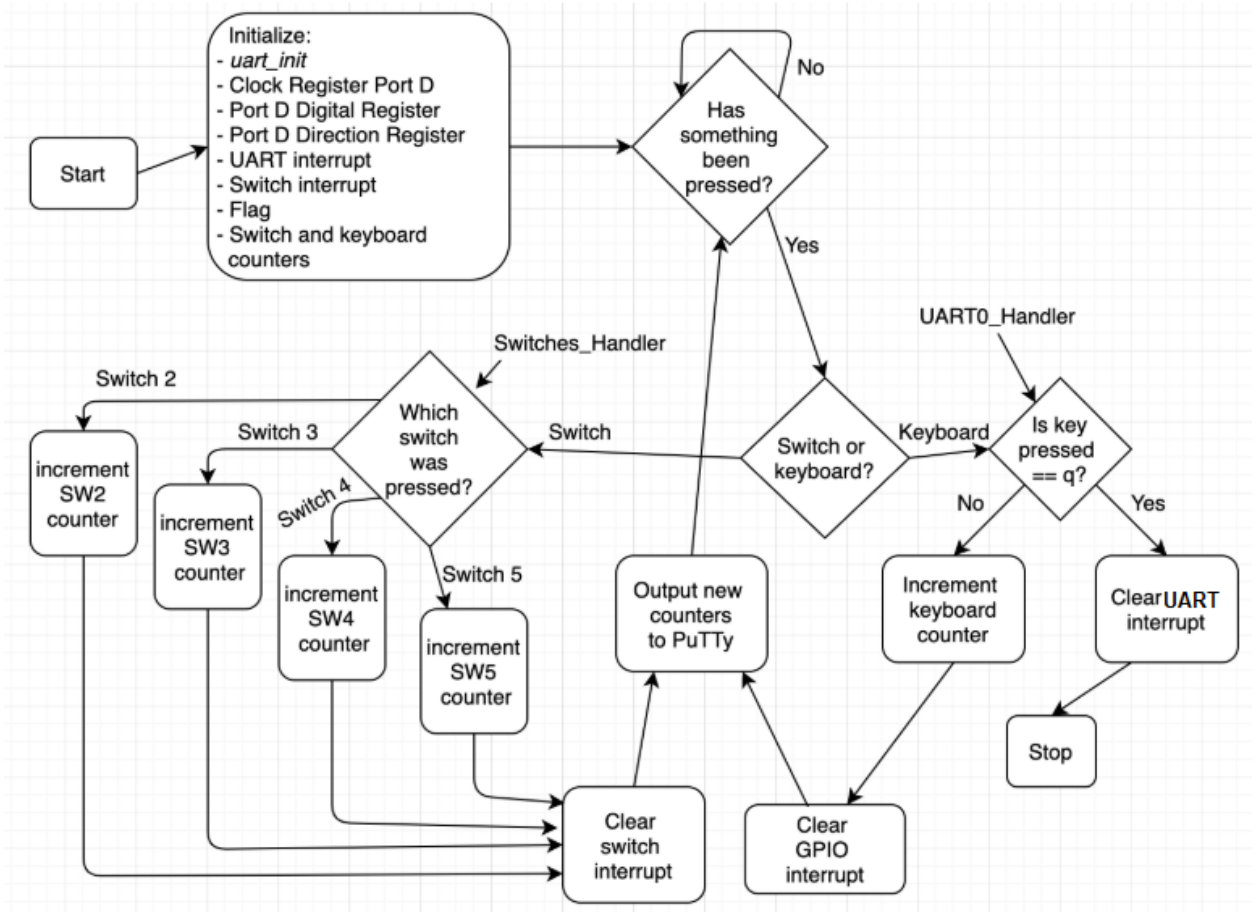
All flowcharts are generated by draw.io.

## a) Rough Draft



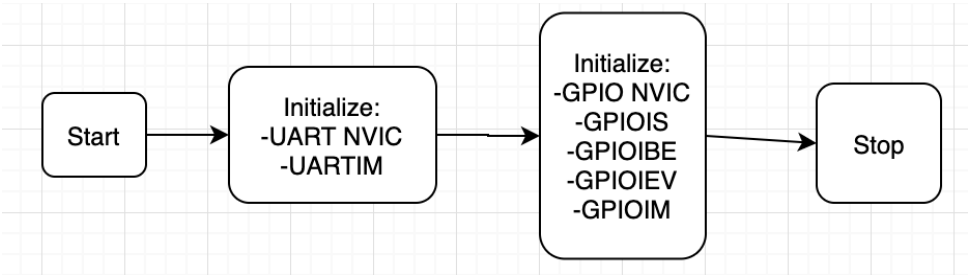
## b) Main Program Flowchart

*lab_5:*

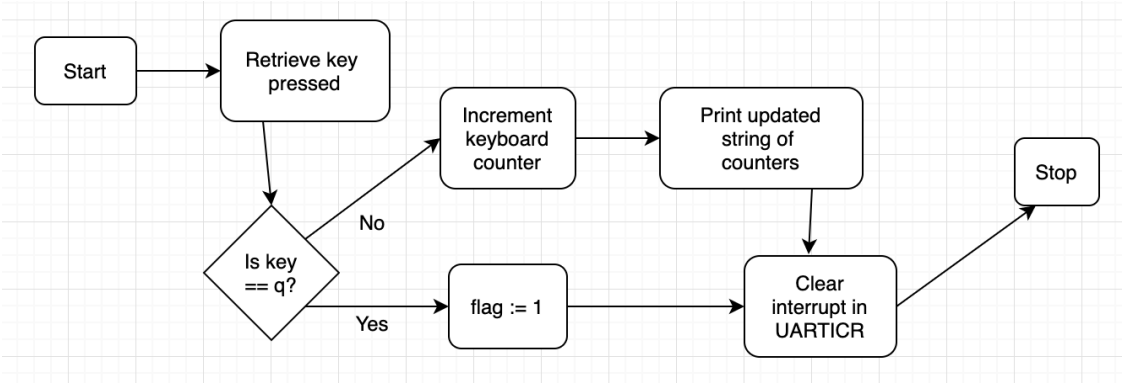


## c) Subroutine Flowcharts

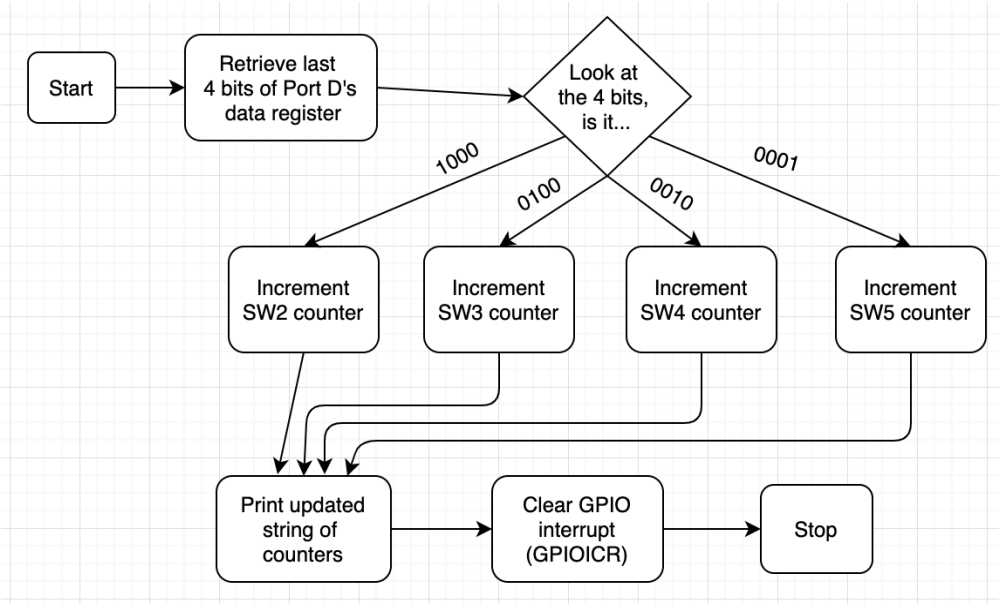
*i) interrupt_init*



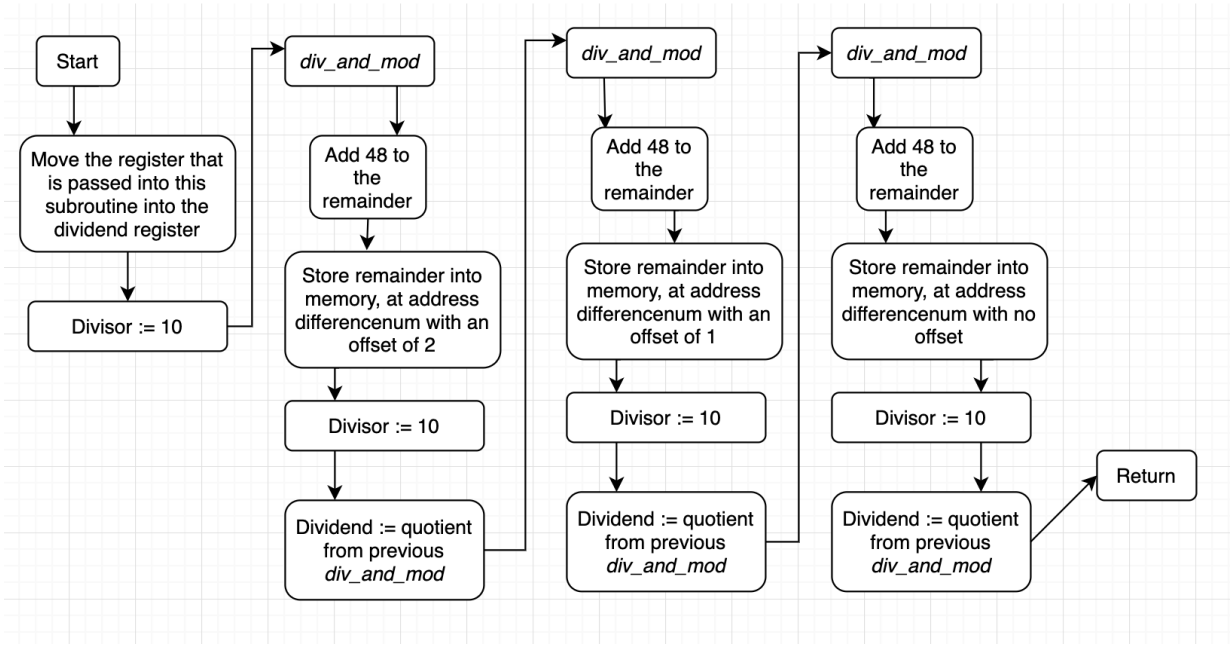

*ii) UART0_Handler*




*iii) Switches_Handler*



*iv) convert_to_ASCII_updated*




*v) print_update*