SC2107 Lab3 Assignment Sheet (to be submitted to NTULearn before next lab)

Name: \_\_\_Ganesh Rudra Prasadh\_\_\_ Lab Group: SCE1 Date: 25/Oct/2023\_\_\_

1. Section 6. Exception handling in ARM processor is handle at three levels (Global, NVIC, and Peripheral). Which are the two registers that configure the exception handling at a global level? Explain whether we need to configure these registers in our lab exercise.

Answer: At the global level, there are two registers that configure the exception handling. The PRIMASK (Priority Mask) toggles the global interrupt main switch via it’s LSB value. The BasePRI (Base Priority) register defines the minimum priority for interrupt handling. In this lab, it’s not required to configure these values since the default values work for this lab exercise.

1. Section 6.2. The bump switch used in the lab is shown below. Pin 1 and 3 of the bump switch are connected to the MSP432. Draw the internal circuit of the bump switch and describe how the MSP432 GPIO can be used to detect that the switch is closed?

|  |  |
| --- | --- |
| Answer: |  |

Above is the diagram of the internal circuit of the bump switch. Since it consists of an internal pullup resistor, it reads logic 1 when the switch is open. Consequently, it reads logic 0 when the bump switch is closed. By enabling interrupts on the pin, we can send an interrupt during the falling edge to check if there’s a change in the value from 1 to 0. Through this, the MSP432 GPIO can be used to detect that the switch is closed.

1. Section 6.3. Write down the GPIO configuration used for pins connected to the Bump switches IF they are connected to Pin6.0 to P6.5.

Answer: P6->SEL0 &= ~(0x3F); // initialise as GPIO

P6->SEL1 &= ~(0x3F); // initialise as GPIO

P6->DIR &= ~(0x3F); // initialise the 6 pins as input

P6->REN |= 0x3F; // enable pull up/down

P6->OUT |= 0x3F; // set as pullup

P6->IE |= 0x3F; // Enable interrupt

P6->IFG &= ~(0x3F); Clear the IFG – no pending interrupts

P6->IES |= 0x3F; // Set as high to low transition

1. Section 6.3. What is the frequency of the clock source of systick timer? Explain how systick timer is configured to interrupt the system at 1000Hz frequency. Illustrate with detail calculations and APIs used.

Answer: In the code, we call the function “Clock\_Init48MHz()” which implies that the frequency of the clock source of the systick timer is 48 MHz.

To interrupt the system at 1 KHz frequency, the reload register must have a value of 48000000 Hz / 1000 Hz = 48000.

Calling the function Systick\_Init(48000, 1) loads 48000 into the reload register. Since 48000 / 48000000 Hz = 0.001 Hz-1, which is basically seconds, it will take the Systick timer 0.001s or 1ms to count down to 0 from the reload value.

Since an interrupt is called everytime the timer reaches 0, an interrupt will be called every 1 millisecond which is 1 / 1ms = 1000 Hz which matches the requirements of the question.

1. Section 6.3. What is the advantage the method of reading Reflectance sensor (in Lab3 section 6.3) has compared to the method used in Lab2?

Answer: In lab 2, we call the function ‘Clock\_Delay1us’ function which uses a spin loop to implement a delay before sampling the capacitor discharge. This is not the best implementation since during the delay time, the CPU doesn’t do anything. In lab 3, the delay is tracked using systick interrupts instead. This frees and doesn’t hog the CPU during the delay time, thereby more efficiently using the CPU resources than in lab 2.

1. Section 6.4. Reference to PWM\_Init34() in PWM.c, what is the timer base clock used to increment the counters in Timer\_A0? Show the details of how this base clock of Timer\_A0 is derived, starting from processor clock. Note that SMCLK=12Mhz.

Answer: The timer base clock used to increment the counters in Timer\_A0 was the SMCLK (which has a frequency of 12MHz as given in the question). Although 1 divider is sufficient, from PWM\_Init34(), two clock dividers are used to step down the frequency.

From the line “TIMER\_A0-> CTL = 0x02F0” which in binary is 0b0000\_0010\_1111\_0000. Bits 7 and 6 is used for the TA0CTL register ID and provide the ratio for the first divider, which in this case are both 1. This means that the input frequency is divided by 8.

In the line “TIMER\_A0 -> EX0 = 0x0000” which is just all 0s in binary, bits 0, 1 and 2 are used for TAIDEX in the TA0EX0 register which gives the second clock divider ratio. Since the value of the three bits are 0, the input frequency is divided by 1 (no change). This leads to the output frequency being equal to 12 MHz / 8 / 1= 1.5 MHz.

1. Section 6.5. What is the PWM frequency generated to the motor? illustrate with detail working.

Answer: From the previous question, the base clock frequency after the dividers is 1.5 MHz. This means that it would take 1 / 1500000 Hz = 0.000000667 s or 0.667 μs to count once. However, since the timer is configured to up/down mode, it will need to count from 0h to CCR0 and back to 0h.

It is given that CCR0 = 7500, therefore the time period of the signal will be 2 x 7500 x 0.667 μs = 0.01s. Note that we multiply by 2 since it needs to count both up and down to complete one cycle. The PWM frequency generated to the motor is equal to 1 / 0.01s = 100 Hz.

1. Section 6.5. Is interrupt mechanism used in the PWM generation via Timers?

Answer: No interrupts are used. The PWM generation via timers (which is in toggle/reset mode) is outputted via the CCR2 timer output. The generation will occur when the runtime timer count is equal to the CCR2 register value. This process does not involve an interrupt mechanism.

1. Section 6.5. What is the IRQ number you need to reference to if Timer\_A2 instead of Timer A1 is used in Lab3\_TimerCompare\_Motor project? What is the corresponding Exception number?

Answer: To reference to Timer\_A2 instead of Timer\_A1, we need to reference to IRQ number 12. Its corresponding exception number is 28.