NAMES:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ECE-C304 Lab – Assignment # 2

(2A,2B,2C due 4th Week, 2D,2E,2F,2G due 5th Week)

The lab is based on the handout supplied

*Note: Always use the part no CY8C29466-24PXI*

Lab 2A-Posted Interrupts

2A-1 What does this code do?

**This code will increment Port 1 every second. The LEDs will light up to represent the bits 0-4.**

2A-2 Do the LEDs operate as you predicted in 2A-1?

**Yes**

2A-3 With the sleep timer parameter at 8 Hz what should be the effect on the LEDs?

**Port 1 will increment every 1/8th of a second.**

2A\_4 Do the observations confirm your prediction? **Yes**

2A-5 Identify the register and the bits which determine the sleep timer period.

**Register:**  OSC\_CR0 **Bits:** 3 & 4

2A-6 Change the sleep timer parameter back to 1 Hz in the GUI but add code to change it to 8 Hz at

run time using information from 2A-5. Confirm you get the same LED behavior as in 2A-4

**(submit code)**

#include <m8c.h> // part specific constants and macros

#include "PSoCAPI.h" // PSoC API definitions for all User Modules

extern unsigned char bShadow = 0;

void main(void)

{

OSC\_CR0 ^= 0x10;

PRT1DR = 0;

while(1){

while((INT\_CLR0 & 0x40) == 0);//wait til set

INT\_CLR0 = INT\_CLR0 & ~0x40; //clear it

bShadow++;

PRT1DR = bShadow;

}

}

2A-7 Does the sleep timer frequency depend on the CPU clock? To determine this, change the

CPU clock to 750 KHz in the code (do not use GUI). Find the registers and bits to achieve this.

Does this change alter the LED behavior?

**No**

2A-8 Convert the entire code of 2A-1 to Assembly language and show that it works. **(submit code)**

include "m8c.inc" ; part specific constants and macros

include "memory.inc" ; Constants & macros for SMM/LMM and Compiler

include "PSoCAPI.inc" ; PSoC API definitions for all User Modules

export bShadow

area MyArea(RAM) ;Create storage location for variables

bShadow: BLK 1

area text(ROM,REL)

export \_main

\_main:

MOV [bShadow], 0

MOV reg[PRT1DR], 0

loop:

TST reg[INT\_CLR0],40h

JNZ interruptLoop

JMP loop

interruptLoop:

MOV A,[bShadow]

INC A

MOV [bShadow],A

MOV reg[PRT1DR],A

MOV A,reg[INT\_CLR0]

AND A,~40h

MOV reg[INT\_CLR0],A

JMP loop

.terminate:

jmp .terminate

Initials\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Time\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab 2B- Pending Interrupts

2B-1 Why might you want to use the sleep mode instead of polling for the posted interrupt?

**So that you are using less energy in between when you actually need the chip to do something.**

2B-2 Why might you want to poll instead of putting the system into sleep mode?

**You might want to have the chip doing something and periodically execute something else.**

2B-3 Do this exercise without using the built-in routines or the GUI. You have to find the

appropriate registers and bits to set the drive mode for the pins , to put the system

in sleep mode, the sleep timer frequency etc. and write code for these **(submit code)**

#include <m8c.h> // part specific constants and macros

#include "PSoCAPI.h" // PSoC API definitions for all User Modules

extern unsigned char bShadow = 0;

void main(void){

PRT1DR = 0; //Zeroes out Port 1

INT\_MSK0 |= 0x40;//allows sleep mode to be put into sleep

//Sets the sleep timer frequency to 8Hz

OSC\_CR0 &= ~0x18;

OSC\_CR0 |= 0x10;

//These 3 ports set the drive modes of the pins\

//DM[210] = 001 sets the drive mode to strong.

PRT1DM0 = 0x0F; //DM0[0-3] = 1

PRT1DM1 = 0x00; //DM1[0-3] = 0

PRT1DM2 = 0x00; //DM1[0-3] = 0

while(1){

CPU\_SCR0 |= 0x08; //Puts the chip into sleep mode

INT\_CLR0 = INT\_CLR0 & ~0x40; //clear it

bShadow++;

PRT1DR = bShadow;

}

}

Initials\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Time\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab 2C – Global Interrupts

2C-1 How much code space is available at this vector location?

**4 bytes**

2C-2 Suppose more space is needed, what can you do?

**You can use the ljmp assembly code and jump to a label where the rest of your code is and put “reti” at the end of that code.**

2C-3 Why is it not necessary to clear the posted interrupt in this case?

**Because the ISR handles that automatically.**

2C-4 What does the Interrupt Service Routine (ISR) do?

**It is a block of code that normal code execution is diverted to when the chip receives a hardware interrupt. Many interrupt sources may exist each with their own code block and priority. At the end of each ISR the chip returns to executing the main code from where it left off.**

//----------------------------------------------------------------------------

// C main line

//----------------------------------------------------------------------------

#include <m8c.h> // part specific constants and macros

#include "PSoCAPI.h" // PSoC API definitions for all User Modules

extern unsigned char bShadow = 0;

void main(void){

PRT1DR = 0;

INT\_MSK0 |= 0x40;

M8C\_EnableGInt;

while(1){

M8C\_Sleep;

bShadow++;

PRT1DR = bShadow;

}

}

Initials\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Time\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab 2D – Assembly Language ISRs

2D-1 How many bytes are needed for this instruction?

**3 Bytes**

2D-2 Suppose ljmp in location 64h is changed to lcall, what other changes are required?

Is it advisable to do so? Why or why not?

**If you use lcall you will have to change it so that after your lcall command you have a reti statement to reset the sleep timer and a ret statement after your subfunction. Lcall takes up more memory to execute because it has to keep the subfunction in the stack in order to be able to return to where it was. It is better to use ljmp in places where you don’t have the stack memory to hold the function while you use it.**

**Main.c**

#include <m8c.h> // part specific constants and macros

#include "PSoCAPI.h" // PSoC API definitions for all User Modules

extern unsigned char bShadow = 0;

void main(void){

PRT1DR = 0;

INT\_MSK0 |= 0x40;

M8C\_EnableGInt;

while(1){

}

**SleepTimerRoutine.asm**

include "m8c.inc"

export SleepTimerISR

SleepTimerISR:

push A

inc [\_bShadow]

mov A,[\_bShadow]

mov reg[PRT1DR],A

pop A

reti

Initials\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Time\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab 2E – C Language ISRs

2E- 1 Why can’t the function accept parameters or return a result?

It can’t accept any parameters or return a result because it can never be called normally in the program. It can only be called through interrupts.

#include <m8c.h> // part specific constants and macros

#include "PSoCAPI.h" // PSoC API definitions for all User Modules

extern unsigned char bShadow = 0;

void main(void){

PRT1DR = 0;

INT\_MSK0 |= 0x40;

M8C\_EnableGInt;

while(1){

}

}

#pragma interrupt\_handler SleepTimerISR

void SleepTimerISR(void){

M8C\_Sleep;

bShadow++;

PRT1DR = bShadow;

}

Initials\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Time\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab 2F – Software Interrupts

2F-1 In previous labs the posted interrupt was cleared without adding code to set ENSWINT to

Zero. Why did this work?

**The processor was posting the sleep timer automatically on its own count. By setting ENSWINT, we set the sleep timer in-between the processor’s own count.**

2F-2 How should this program work now?

**The program should be looping through much faster. Because every time it exists the sleep timer function, it should set it again in the while loop.**

2F-3 Does the actual operation agree with your prediction?

**Yes**

2F-4 What is the purpose of software interrupts?

**The purpose is to enter the interrupt function even when it normally shouldn’t be executed.**

#include <m8c.h> // part specific constants and macros

#include "PSoCAPI.h" // PSoC API definitions for all User Modules

extern unsigned char bShadow = 0;

void main(void){

PRT1DR = 0;

INT\_MSK0 |= 0x40;

INT\_MSK3 |= 0x80;

OSC\_CR0 &= ~0x18;

OSC\_CR0 |= 0x10;

M8C\_EnableGInt;

while(1){

INT\_CLR0 |= 0x40;

}

}

#pragma interrupt\_handler SleepTimerISR

void SleepTimerISR(void){

bShadow++;

PRT1DR = bShadow;

}

Initials\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Time\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab 2G – Creating very small ISRs

2G-1 How many bytes does this handler take?

2G-2 How should the output behave?

2G-3 Does actual operation agree with your prediction?

Initials\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Time\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_