SC2107 Lab1 Assignment Sheet (to be submitted to NTULearn before next lab)

Name: \_\_\_Ganesh Rudra Prasadh\_\_\_ Lab Group: SCE1 Date: 07/Sep/2023\_\_\_

1. Section 7.4. Write one C statement to set bit 7 and 5 of P1SEL0 register, keeping the rest of the bits in the register unchanged.

Answer: P1->SEL0 |= 0xA0; // 0b1010\_0000 sets the two bits (assuming it begins at bit0)

1. Section 7.4. Write C statement(s) to extract bit 6 and 5 of variable ‘x’ and right align these two bits. Masked off all other bits in variable ‘x’. e.g. if ‘x’ has a value 1101 0111b initially, it should have a value of 0000 0010b after executing the C statement.

Answer: x = (x & 0x60) >> 5; // First extract bit 6 and 5, and then shift it 5 bits to the right to right align it. Note that even 0b01100000 also works.

1. Section 7.4. Why do we need to declare the P1IN register, the register that contain the status of the processor Port1 GPIO input pin logic with a ‘volatile’ keyword qualifier?

Answer: A volatile keyword is required as if it’s not present, then the compiler might assume that the code will not be used, and either put it in a temporary register or completely remove it from the program. This occurs due to the code optimization feature of the C compiler, which even at the lowest level would assess the P1IN register and deem that it does nothing useful and remove it. Adding a volatile keyword informs the compiler that the value might change outside the context of the program, and is therefore not optimized.

1. Section 7.5. Why do we use SDIV instead of UDIV when calculating the Distance D? Or does it really matter whether SDIV or UDIV is used for this case?

Answer: SDIV stands for Signed division and UDIV stands for unsigned division. If the value produced by the ADC (n) is between 0 and 1058 (which is well within the range of output of a 14 bit ADC – 2^14 - 1), then while performing n – 1058, the value in the denominator would be negative. An SDIV is to account for this since a UDIV will cause an overflow and will produce a wrong answer. However, in this specific case, as seen in the graph given in the lab manual, the ADC does not produce a value before 1058, so both UDIV and SDIV work here, though it is good practice to just use SDIV.

1. Section 7.5. What is saved into the LR register when the calling routine calls “BL Convert”? What command is used to return from the sub-routine to the calling routine?

Answer: The address of the next instruction that is to be executed in the calling routine function (after the BL subroutine function is called) is stored in the LR register. We can use the BX LR command at the end of the subroutine to branch back to the calling routine, which will then copy whatever is in the LR to the Program Counter. This allows the program to continue off from right after the subroutine call.

1. Section 7.5. If a function has 4 input parameters, which registers does the calling routine used to pass these parameters to the function according to AAPCS?

Answer: The calling routine passes the parameters using registers R0 – R3 without needing to store and restore in the stack.

1. Section 7.6. What data content is loaded into R1 by the instruction “ldr r1, [pc, #0x2e4]”? Just the expression will do, need not give the exact value since the offset in your code may be different.

Answer: This loads the value of the next instruction to be executed (PC + 4) + offset of #0x2e4 into register R1.

1. Section 7.7. The Memory Section “MAIN” correspond to the On-Chip Flash Memory in MSP432. How much on-chip flash memory is available for future code development? Cut and paste the screen shot of the relevant content in the map file and highlight where you extract your answer from. Hint: Check the map file.

A screenshot of a computer

Description automatically generated

*Figure 1: Screen shot of map file depicting used and unused memory*

Answer: Above is a screenshot from the map file. The main file has a length of 0x00040000 which corresponds to 256 Kilobytes of on-chip flash memory. Since 0x7fc has been used up, there is still 0x0003f804 (approximately 254 KB) bytes of memory remaining which can be used for future codes.

1. Section 7.7. Which software section are code allocated to by default? Which file consumes the largest code size in this project? Hint: check the map file.

Answer: By default, the code is always allocated to the ‘.text’ software section. The size of this software section is 0x000006f0 bytes. Hence the code size of the project is the same (about 1.73 KB). The file – ‘system\_msp432p401r.obj’ is 0x0000032c bytes (812 bytes) and consumes the largest code size in the project.

1. Section 7.7. From the map file, what is the starting address of Port2\_Init()? Compare with the address you see in the Disassembly Window, are they the same? If not, why?

A computer screen with text and numbers

Description automatically generated

*Figure 2: Screenshot of Disassembly Window*

A screenshot of a computer

Description automatically generated

*Figure 3: Screenshot of another part of map file*

Answer: The address in the disassembly window and map file are different, with the disassembly window showing the starting address of Port2\_Init() as 0x0000043a and the map file showing 0x0000043b. This is because they Port2\_Init() is a separated function in the embedded C file, causing the entry point to differ in the map file and disassembly window. Therefore it is not surprising to see a different value in both the files.