SC2107 Lab1 Assignment Sheet (to be submitted to NTULearn before next lab) (I’m using CCS 20.2)

Name: Jonathan Low Seng Kian Lab Group: TCMA Date: 29/8/2025

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

P1->SEL0 &= ~0xA0

1. Section 7.4. Write C statement(s) to extract bit 4 and 3 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.

x = x>>3;

x &= 0x03;

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

We need to declare the P1IN register with the volatile keyword because its value can change independently of the program flow, for example, due to hardware events on the Port 1 GPIO pins. Declaring it volatile tells the compiler not to optimize or cache its value, ensuring that every read accesses the current hardware state. Without volatile, the compiler might assume the value doesn't change and reuse a previous value, leading to incorrect behavior.

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?

SDIV is used because the divisor (n - 1058) could conceptually be negative if n were below 1058. In the given ADC range (2552–16383), the divisor is always positive, so UDIV would also work. Using SDIV is safer and ensures correct results even if the value of n goes below 1058 for future cases, future proofing the code.

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?

LR stores the return address, the address of the instruction immediately after the BL in the calling routine, so that sub routine knows how to go back to the main routine/calling routine. BX LR.

1. Section 7.6. If a function has 4 input parameters, which registers does the calling routine used to pass these parameters to the function according to AAPCS?

R0-R3

1. Section 7.6. What data content is loaded into R1 by the instruction “ldr r1, [pc, #0x2e4]”? Please give a qualitative description of the content loaded to R1 and not the value you see in CCS register window.

R1 has this content of address memory 0x40004C0A. The instruction ldr r1, [pc, #0x2e4] loads into R1 the address of the Port 1 SEL0 register (P1->SEL0). This allows the program to access the register in memory so that it can write 0x00 to it, configuring all Port 1 pins as GPIO

A screenshot of a computer

AI-generated content may be incorrect.

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.

The MAIN section of on-chip Flash memory has a total size of 256 KB, of which approximately 255.75 KB is unused and available (based off Lab1\_InputOutput.out) and available for future code development. You would extract this information from the “MEMORY CONFIGURATION” table, specifically the row for MAIN.

A screenshot of a computer program

AI-generated content may be incorrect.

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

Program code is allocated to the .text section by default. The largest contributor to code size in this project is InputOutput.obj (812 bytes), followed closely by system\_msp432p401r.obj (812 bytes).

A computer screen with white text

AI-generated content may be incorrect.

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?

The starting address of Port2\_Init() from the map file is 0x0000010F. They are not the same, the Disassembly Window shows 0x0000043A as the starting address of Port2\_Init(), which is the actual instruction address in memory. It differs from the map file symbol address (0x0000010F) due to linker relocation and memory alignment. The map file shows the symbol address, while the disassembly window shows the actual aligned instruction address in memory, this is because ARM instructions must be halfword (2-byte) aligned, and the linker may pad the section so that the instructions start at the nearest aligned address

A screen shot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.