

ECEN 5803 - Mastering Embedded System Architecture

Homework set 2

Swati Kadivar

Theory and Analysis:

1.

a.) Which version would you pick to use and why?

I would NOT pick any of the versions because B is not giving exact string copy functionality. While loop in B case will never stop. Also none of them are checking destination string size whether it will be able to accommodate source or not.

b.) Do both versions provide the intended function?

No. Version A is correct implementation of string copy functionality. Version B also does copy but there is no terminating condition provided. While loop will run infinitely.

c.) Which is most efficient?

Version B will be most efficient comparatively if implemented properly with terminating conditions.

d.) Why use the const keyword in the source declaration?

To prevent accidental modification to the source while copying.

e.) Can you write a better version of the string copy function? If so, show it here:

```
int mystrcpy(char str1[], const char str2[]){  
    int len1;  
    int len2;
```


```

for(len1 = 0; str1[len1] !='\0'; len1++);
for(len2 = 0; str2[len2] !='\0'; len2++);

if(len1 < len2){
    printf("destination string size is not enough to accomodate source string\n");
    return -1;
}
while(*str2 != '\0'){
    *str1++=*str2++;
}
}

```

2. Describe the meaning and use of the C keyword volatile. Give example code using this keyword.

When a variable declared with volatile keyword  then it is an instruction to the compiler to not apply any optimizations on that variable because this variable can change outside of the code in ways that cannot be determined by the compiler.

```
#include <stdio.h>
```

```
Void main(void)
```

```

{
    const int a = 10;
    int *p = (int*) &a;
    printf("Initial value of a: %d \n", a);
    *p = 100;
}

```

```

    printf("Modified value of a: %d \n", a);
    return 0;
}

```

Output:

Initial value of a: 10

Modified value of a: 100

This example when compiled with `-o volatile` option, compiler will not try to optimize that variable as anyone outside of code can modify that variable. Here, we won't be able to see the compiler level output, but this concept can be used when many sensors are connected to the system and lower-level drivers are not supposed to modify sensor data output and just send it to higher-level code.

3.

```

    //r2 = 0;
    MOVS R2, #0

    //while (r1 != 0) {
LOOP    CMP R1,#0
        BEQ loopend
        //if ((r1 & 1) != 0) {
        MOVS R3, #1
        ANDS R3,R1,R3
        MOV R1,R3
        CMP R1,#0
        BEQ ifexit
        //r2 += r0;

```

```

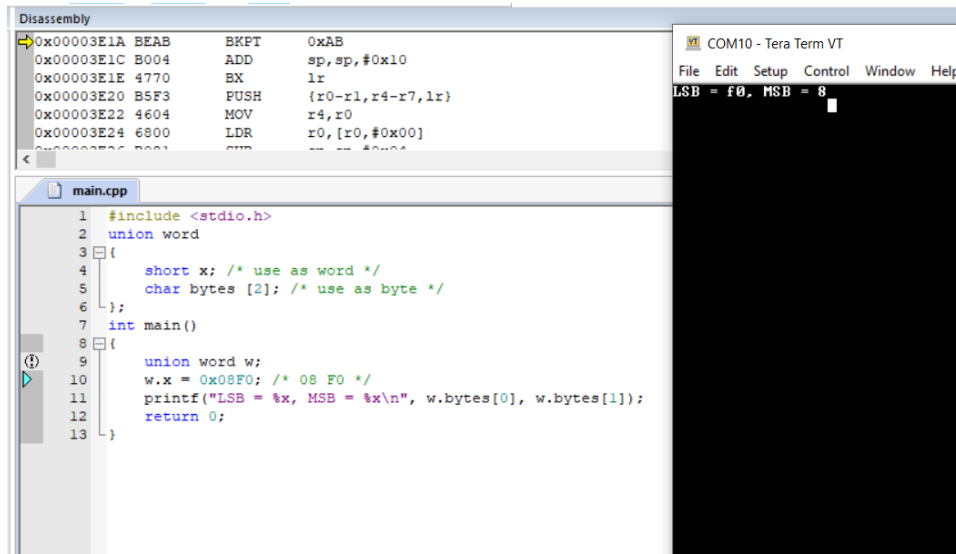
        ADD R2,R2,R0
    //}
ifexit
    //r0 <<= 1;
    LSLS R0,R0,#1
    //r1 >>= 1;
    LSRS R1,R1,#1
//}
loopend
    BX LR

```

4. For the below ARM assembly code, trace the values as it executes that will be placed into the registers R4, R5, and R6.

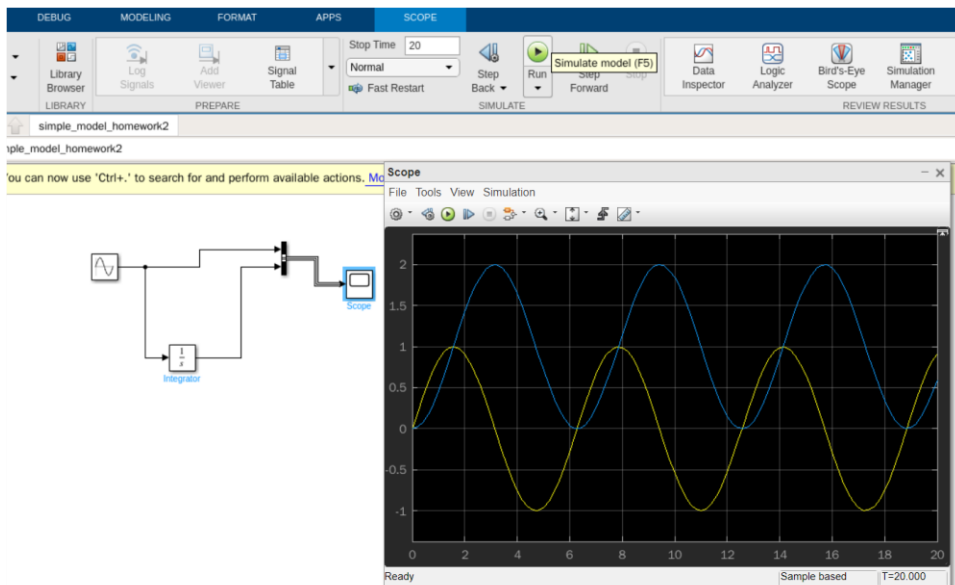
	R4	R5	R6	R7
1	11	7	3	7
2	18	11	2	11
3	29	18	1	18
4	47	29	0	29

5. Embedded C. Evaluate the following code and determine what will print when using an ARM Cortex-M0+ processor:



This code explains about the use of an union in Embedded C. How two variables refer to the same memory and so it modes data depending on how the user is requesting it. Here, we fed data in x which is 2 bytes short int but we are reading data through char array so referring to the same memory location is returing same data but in form of two array elements instead of one int variable.

6. Complete the simple model Simulink simulation in the Simulink startup guide, sl_gs.pdf chapter 2. Show both the finished block diagram and scope screenshots.



7. Select the best processor to meet the listed requirements. Limit your search to processors from Atmel, Freescale, and Microchip. Use the processor selection spreadsheet you have been given, or any online processor selection tools. Document the steps you took to come to your conclusions.

Requirement for an Electronic Cigarette

- 1. Very lowest cost**
- 2. Low power, < 1 mW average.**
- 3. 3v operation**
- 4. Internal oscillator**
- 5. Package must be smaller area than 25 mm squared.**
- 6. Code must be secure, not possible to read out to prevent reverse engineering.**
- 7. ≥ 4 kB SRAM, 8 kB FLASH on chip**
- 8. UART and I2C port for BLE and pressure sensor.**
- 9. 4 PWM outputs**
- 10. MicroUSB OTG or Device port**

Manufacturer : Atmel,

Part Number: ATSAM51G18A,

Family: SAMD,

Core: Cortex-M4F

8. Select the best processor to meet the listed requirements. Limit your search to processors from NXP (not including Freescale), ST, and Texas Instruments (TI). Use the processor selection spreadsheet you have been given, or any online processor selection tools. Document the steps you took to come to your conclusions.

Requirement for an USB Audio Concentrator

- 1. Cost under \$10**
- 2. USB 2.0 Device Interface**
- 3. 4 channels I2S**
- 4. DSP processing requires 100 DMIPS per I2S channel**
- 5. SPI port to Serial FLASH**
- 6. ≥ 40 kB SRAM on chip, can be in cache.**
- 7. > 100 kB FLASH on chip**
- 8. OS must support USB Audio class**

Manufacturer: ST

Part Number: STM32F072CB

Family: STM32

Core: ARM M0

9.10 Identify and briefly discuss the steps that comprise the Spiral model?

The model takes a risk-oriented view of the development life cycle. Each spiral addresses the major risks that have been identified. After all the risks have been addressed, the Spiral model terminates.

- Determine objectives, alternatives, and constraints:

First step is to determine objectives so that we will be aware of our requirements and constraints, be aware about alternatives in case of dead end in development or if such inter-dependencies arise.

- Identify and resolve risks:

Each spiral addresses the major risks that have been identified. After deciding on objectives, this is more of analytical step which asks us to identify and avoid/resolve as many risks possible at first.

- Evaluate alternatives

After resolving identifiable risks, if there are any dependencies then consider alternatives and evaluate them to approach next step on time.

- Develop deliverables—verify that they are correct:

Plan and develop deliverables so that they can be delivered on time and also validation and verification of deliverables is necessary to make sure they are correct.

- Plan the next iteration:

Log all the issues faced and all the learnings from them in the current spiral and plan the next iteration based on those learnings.

- Commit to an approach for the next iteration:

Execute the planned approach for next spiral based on previous spiral learnings.

9.17 For Whom is the requirements document written?

- What is to be tested?
- The testing order within each type of test.
- Assumptions made.
- Algorithms that may be used.

All above mentioned things are mentioned in the requirements document and it is written for the engineers who will test the design while keeping certain things in mind. So they must know what to test, what is the order of the testing, assumptions made in the design to keep in mind and what all algorithms can be used to test the system.

9.19) A Version control system is generally used to manage code revisions on most embedded designs. Should the requirements and design specifications be managed in similar manner? Explain your thinking?

Yes. We should maintain design specifications in version control manner. As per new updates on the products, requirements also get updated. But at any given point of time, at different places, multiple versions of same products are being used, even

though latest products are present in the market, there may be many customers who are using older version of the same product. So, for older products, we should maintain older requirements and design specification documents. For example, latest version of iPhone13 is launched in the market but still there will be many people who are using iPhone12 or iPhone11. So it is necessary to maintain iPhone12 and iPhone11's requirements and design documents.

Problem 6.5.

What is relocatable code?

Relocatable-Code is position-independent code in multiple manners. This type of code can be loaded anywhere in memory, but usually has been relocated or fixed up before it is executable. In fact, some architectures that use this type of code embed things like "reloc" sections for fixing up the relocatable parts of the code. The downside of this type of code is that once it is relocated and fixed up, it almost becomes absolute in nature and fixed at its address.

The major advantage of relocatable code is that it allows code to be easily broken down into sections. Each section can be loaded anywhere in memory to fit its requirements and then during relocation, any code that references another section can be fixed-up with a relocation table and thus the sections can be tied together nicely. The code itself is usually relative (as with the x86 architecture), but it need not be, as anything that might be out of range can be assembled as a relocatable instruction such that it consists of an offset added to its load address. It also means that limitations imposed by relative-addressing are no-longer an issue.

Ref:

<https://stackoverflow.com/questions/22889719/what-is-relocatable-and-absolute-machine-code>

Problem 6.11.

What is boot code, and what is its purpose?

Boot code is a set of "instructions" that are run by a computer when it is starting up. The boot code helps the computer prepare the system for loading and running an operating system, but boot code itself is usually not operating system specific. Boot

code may run from a hard drive or even directly from a computer CPU (if integrated into the silicon). Without boot code, a computer would likely not boot up correctly, if at all.

Ref: <https://www.computerhope.com/jargon/b/bootcode.htm>

Problem 6.21.

What is the purpose of the const qualifier?

Const qualifier is used to tell the compiler and to the whole program that this variable is not allowed to be modified throughout the execution. The qualifier const can be applied to the declaration of any variable to specify that its value will not be changed (Which depends upon where const variables are stored, we may change the value of const variable by using pointer). The result is implementation-defined if an attempt is made to change a const.

Ref: <https://www.geeksforgeeks.org/const-qualifier-in-c/>

Problem 6.24.

Within an embedded C program, what does the term scope mean?

Scope means till when variable is visible. For example, if you declare a variable inside some function then that variable's scope will be till the end of the function but within the function also if you declare some variable inside a loop or some if statement then that variable won't be accessible outside the loop/if. There are three type of scopes : **global, file, local**. Local variables are only visible within the block in which they have been declared and defined. Variables in a local scope are not initialized upon declaration unless special provisions are made. The visibility of file scope variables is limited to the file in which they are declared. Global variables, as their name suggests, are visible from the point of declaration until the end of the program. They are visible to every function within the program, all comprising C source files. Such visibility means that any function within the program can access and potentially change these variables. Problems arise because functions may change a value that others depend on.

Problem 6.4.

What is a symbol table? What information is stored in a symbol table and what is its purpose?

The compiler translates each of the two C modules into the assembly language for the target machine. During that process, the names of all variables are identified and an entry for each is made in a table called symbol table the symbol table. If the compiler is able to identify an address within the module for each variable, that information is associated with the variable in the symbol table. If it cannot, and it hasn't been told that the variable is defined in some other module (via the extern directive), it cites the variable as being undefined.

7.5: Following declaration and definition, a variable is stored somewhere in memory. How can one find out where the variable is stored?

Based on the type of a variable declared, there are six type of storage classes and variable will be stored in different parts of the memory accordingly.

- Auto

It specifies that the variable so designated is to be stored on the stack, that it will be local to the function using it, and that the compiler will automatically destroy it when the enclosing scope is exited.

- Extern

The extern storage class gives the means to declare and define a variable in one implementation file (or standard or custom library) and then to use it in another.

- Static

A static variable persists across multiple invocations of the containing function. The memory for such a variable is allocated in the static memory pool and initialized once (to 0 by default) when the program starts, and yet it lives as long as the program containing it does.

- Register

Registers are the fastest type of hardware storage and are the closest to the CPU of all the memory in the microprocessor.

7.63: Identify and briefly describe the steps necessary to handle an interrupt

CPU context:

At any point in time, the values of all the registers in the CPU defines the context of the CPU. Another name used for CPU context is CPU state.

Saving context:

The exception/interrupt handler uses the same CPU as the currently executing process. When entering the exception/interrupt handler, the values in all CPU registers to be used by the exception/interrupt handler must be saved to memory. The saved register values can later restored before resuming execution of the process.

Determine the cause

The handler may have been invoked for a number of reasons. The handler thus needs to determine the cause of the exception or interrupt. Information about what caused the exception or interrupt can be stored in dedicated registers or at predefined addresses in memory.

Handle the exception/interrupt

Next, the exception or interrupt needs to be serviced. For instance, if it was a keyboard interrupt, then the key code of the keypress is obtained and stored somewhere or some other appropriate action is taken. If it was an arithmetic overflow exception, an error message may be printed or the program may be terminated.

Select a process to resume

The exception/interrupt have now been handled and the kernel. The kernel may choose to resume the same process that was executing prior to handling the exception/interrupt or resume execution of any other process currently in memory.

Restoring context

The context of the CPU can now be restored for the chosen process by reading and restoring all register values from memory.

Resume

The process selected to be resumed must be resumed at the same point it was stopped. The address of this instruction was saved by the machine when the interrupt occurred, so it is simply a matter of getting this address and make the CPU continue to execute at this address.

Ref: <http://www.it.uu.se/education/course/homepage/os/vt18/module-1/exception-and-interrupt-handling/>

7.40: Why do interrupt service routines typically accept no arguments and have no return value?

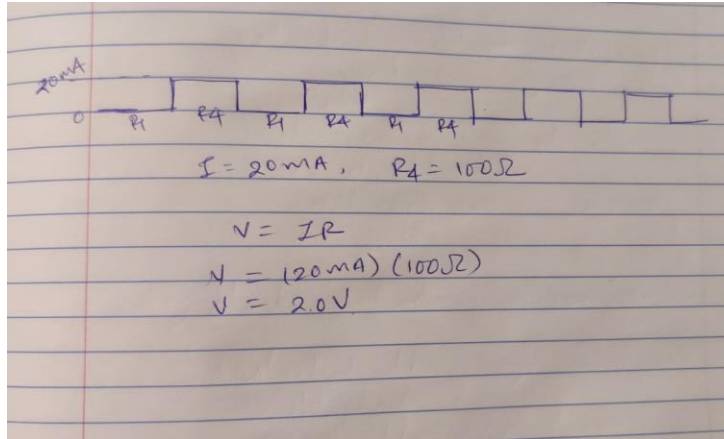
The ISR gets called by the CPU in the middle of the running code to respond to some higher priority tasks. The CPU pushes all of the current program's execution information (PC, LR etc) and then executes ISR. So there really is no calling code for the ISR and so it does not have anybody to pass arguments or to pass on the results in return.

Ref: <https://www.embeddedrelated.com/showthread/comp.arch.embedded/9113-1.php>

Problem 2.18: A junior engineer has proposed the design in Figure P2.60 for a portion of a circuit she is working on. She has said the design works perfectly, but has asked you to review it anyway. She gives you the information shown in the figure from the vendor's data sheet.

a. Using the information from the data sheets, complete the timing diagram in Figure P2.61 for the identified signals based on the conditions shown.

Below is the timing diagram of Fig 2.61 operation. Book assumes that Left circuit drives output low and right-side circuit is driving output high. So accordingly, I reviewed the circuit and draw below timing diagram. As per given in the datasheet, 20mA of current is passing through R4. Circuit makes rout to ground via R1 and R4. Current through R4 is 20mA and if we take $R4 = 100\Omega$ then there will be 2V of voltage drop across R4.



b. Based on your analysis, what can you tell her about the design?

Based on above analysis, I can say that her design has two redundant registers R_2 , R_3 as current makes path to the ground through R_1 and R_4 . So she can remove those two registers. Also assumption is made (left circuit is driving output low and right one high) to understand the behavior of the circuit.