Embedded Programming in C/C++:
Lesson-1: Programming Elements and Programming in C

Programming

- An essential part of any embedded system design

Programming in Assembly or HLL

- Processor and memory-sensitive instructions: Program codes may be written in assembly
- Most of codes: Written in a high level language (HLL), ‘C’, ‘C++’ or Java

1. Assembly Language Programming

Advantage of Assembly Language Programming

- Assembly codes sensitive to the processor, memory, ports and devices hardware
- Gives a precise control of the processor internal devices
- Enables full use of processor specific features in its instruction set and its addressing modes

Advantage of Assembly Language Programming

- Machine codes are *compact*, processor and memory sensitive
- System needs a smaller memory.
- Memory needed does not depend on the programmer data type selection and rule-declarations
- Not the compiler specific and library-functions specific

Advantage of Assembly Language Programming

- Device driver codes may need only a few assembly instructions.
- Bottom-up-design approach

2. Advantage of using high level language (HLL) for Programming

Short Development Cycle

- Code reusability— A function or routine can be repeatedly used in a program
- Standard library functions— For examples, the mathematical functions and delay ( ), wait ( ), sleep ( ) functions
- Use of the modular building blocks

Short Development Cycle– Bottom-up design

- Sub-modules are designed first for specific and distinct set of actions, then the modules and finally integration into complete design.
- First code the basic functional modules and then build a bigger module and then integrate into the final system

Short Development Cycle– Top-down Design:

- First design of main program (blue-print), then its modules and finally the sub-modules are designed for specific and distinct set of actions.
- Top-down design Most favoured program design approach

Use of Data Type and Declarations

- Examples, *char, int, unsigned short, long, float, double, boolean.*
- Each data type provides an abstraction of the (i) methods to use, manipulate and represent, and (ii) set of permissible operations.

Use of Type Checking

- Type checking during compilation makes the program less prone to errors.
- Example– type checking on a char data type variable (a character) does not permit subtraction, multiplication and division.

Use of Control Structures, loops and Conditions

- Control Structures and loops
- Examples— *while*, *do-while*, *break* and *for*
- Conditional Statements examples
- *if*, *if- else*, *else - if* and *switch - case*)
- Makes tasks simple for the program-flow design

Use of Data Structures

- Data structure- A way of organizing large amounts of data.
- A data elements' collection
- Data element in a structure identified and accessed with the help of a few pointers and/or indices and/or functions.

Standard Data structure

- Queue
- Stack
- Array – one dimensional as a vector
- Multidimensional
- List
- Tree

Use of Objects

- Objects bind the data fields and methods to manipulate those fields
- Objects reusability
- Provide inheritance, method overloading, overriding and interfacing
- Many other features for ease in programming

3. Advantage of using C for Programming

C

- Procedure oriented language (No objects)
- Provision of inserting the assembly language codes in between (called in-line assembly) to obtain a direct hardware control.

Procedure oriented language

- A large program in ‘C’ splits into the declarations for variables, functions and data structure, simpler functional blocks and statements.

In-line assembly codes of C functions

- Processor and memory sensitive part of the program within the in-line assembly, and the complex part in the HLL codes.
- Example function outportb (q, p)
- Example– *Mov al, p; out q, al*

4. C Program Elements

Preprocessor include Directive

- Header, configuration and other available source files are made the part of an embedded system program source file by this directive

Examples of Preprocessor include Directives

```
# include "VxWorks.h" /*  Include
    VxWorks functions*/
# include "semLib.h"  /* Include
    Semaphore functions Library */
# include "taskLib.h" /* Include
    multitasking functions Library */
```

Examples of Preprocessor include Directives

include "*sysLib.c*" /* Include system library for system functions */

include "*netDrvConfig*.txt" /* Include a text file that provides the 'Network Driver Configuration'. */

include "*prctlHandlers.c*" /* Include file for the codes for handling and actions as per the protocols used for driving streams to the network. */

Preprocessor Directive for the definitions

- Global Variables — *# define volatile boolean IntrEnable*
- Constants — *# define false 0*
- Strings— *# define welcomemsg* "Welcome To ABC Telecom"

Preprocessor Macros

- Macro - A named collection of codes that is defined in a program as preprocessor directive.
- Differs from a function in the sense that once a macro is defined by a name, the compiler puts the corresponding codes at the macro at every place where that macro-name appears.

Difference between Macro and Function

- The codes for a function compiled once only
- On calling that function, the processor has to save the context, and on return restore the context.
- Macros are used for short codes only.

Difference between Macro and Function

- When a function call is used instead of macro, the overheads (context saving and return) will take a time, $T_{overheads}$ that is the same order of magnitude as the time, T_{exec} for execution of short codes within a function.
- Use the function when the $T_{overheads} << T_{exec}$ and macro when $T_{overheads} \sim= \text{or} > T_{exec}$.

Use of Modifiers

- auto
- unsigned
- static
- const
- register

Use of Modifiers

- interrupt
- extern
- volatile
- volatile static

Use of infinite loops

- Infinite loops- Never desired in usual programming. Why? The program will never end and never exit or proceed further to the codes after the loop.
- Infinite loop is a feature in embedded system programming!

Example:

A telephone is never switching off. The system software in the telephone has to be always in a waiting loop that finds the ring on the line. An exit from the loop will make the system hardware redundant.

```
# define false 0
# define true 1
void main (void) {
/*  Call RTOS run  here */
rtos.run ( );
/* Infinite while loops follows in each
   task. So never there is return from the
   RTOS. */
}
```

```
void task1 (....)  {
/* Declarations */.
while (true) {
/* Run Codes that repeatedly execute */
/* Run Codes that execute on an event*/
if (flag1) {....;}; flag1 =0;
/* Codes that execute for message to the
   kernel */
message1 ( ); } }
```

Use of typedef

- Example─ A compiler version may not process the declaration as an unsigned byte
- The 'unsigned character' can then be used as a data type.
- Declared as follows: typedef unsigned character portAdata
- Used as follows: #define Pbyte portAdata 0xF1

Use of Pointers

- Pointers are powerful tools when used correctly and according to certain basic principles.

Example

 # define COM ((struct sio near*) 0x2F8);

This statement with a single master stroke assigns the addresses to all 8 variables

Byte at the sio Addresses

0x2F8: Byte at RBR/THR /DLATCH-LByte

0x2F9: Byte at DLATCH-HByte

0x2FA: Byte at IER; 0x2FB: Byte at LCR;

0x2FC: Byte at MCR;

0x2FD: Byte at LSR; 0x2FE: Byte at MSR

0x2FF: Byte Dummy Character

Example

Free the memory spaces allotted to a data structure.

#define NULL (void) 0x0000*

- Now statement *& COM ((struct sio near*) = NULL;*

assigns the COM to Null and make free the memory between 0x2F8 and 0x2FF for other uses.

Data structure

- Example– structure *sio*
- Eight characters– Seven for the bytes in BR/THR/DLATCHLByte, IER, IIR, LCR, MCR, LSR, MSR registers of serial line device and one dummy variable

Example of Data structure declaration

- **Assume structured variable COM at the addresses beginning 0x2F8.**

 # define COM ((struct sio near*) 0x2F8)

- **COM is at 8 addresses 0x2F8-0x2FF and is a structure consisting of 8 character variables structure for the COM port 2 in the UART serial line device at an IBM PC.**

Example of Data structure declaration

 # define COM1 ((struct sio near*) 0x3F8);
It will give another structured variable COM1 at addresses beginning 0x3F8 using the data structure declared earlier as *sio*

Use of functions

(i) Passing the Values (elements):

The values are copied into the arguments of the functions. When the function is executed in this way, it does not change a variable's value at the function, which calls new function.

(ii) Passing the References—

When an argument value to a function passes through a pointer, the called function can change this value. On returning from this function, the new value may be available in the calling program or another function called by this function.

Use of Reentrant Function

- Reentrant function- A function usable by the several tasks and routines synchronously (at the same time). This is because all the values of its argument are retrievable from the stack.

Three conditions for a function called as *reentrant function*

1. All the arguments pass the values and none of the argument is a pointer (address) whenever a calling function calls that function.

2. When an operation is not atomic, that function should not operate on any variable, which is declared outside the function or which an interrupt service routine uses or which is a global variable but passed by reference and not passed by value as an argument into the function. [The value of such a variable or variables, which is not local, does not save on the stack when there is call to another program.]

3. That function does not call any other function that is not itself Reentrant.

Summary

We learnt

- An embedded system programming in the assembly language- only when a precise control of the processors internal devices and full use of processor specific features in its instruction set and addressing modes needed.

- Program in a high-level language features- A short development cycle for a complex system and portability to the system hardware modifications.
- It easily makes larger program development feasible.
- Top-down design approach
- C or C++ or JAVA mainly used for embedded system codes development

We learnt

- C language supports in-line assembly (fragments of codes in assembly) gives the benefits of assembly as well as high-level language

We learnt

- Important programming elements in C- preprocessor directives, library functions, header files, source and configuration files, data type declarations, macros, data structures, infinite loops, reentrant functions,

End of Lesson 1 of Chapter 5