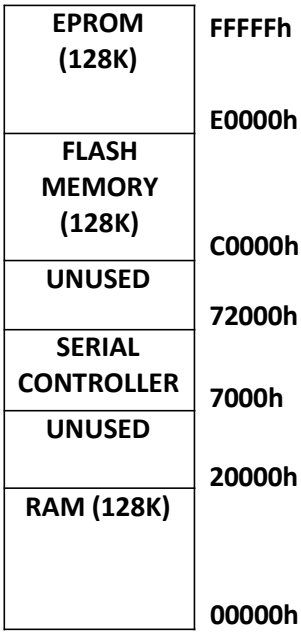
Embedded System

Part-2

7.3 MEMORY MAP

- A Memory Map is the processor's "address book." It shows what these devices look like to the processor. The memory map contains one entry for each of the memories and peripherals that are accessible from the processor's memory space.
- All processors store their programs and data in memory.
- These chips are located in the processor's memory space, and the processor communicates with them by way of two sets of electrical wires called the address bus and the data bus. To read or write a particular location in memory, the processor first writes the desired address onto the address bus. The data is then transferred over the data bus.
- A memory map is a table that shows the name and address range of each memory device and peripheral that is located in the memory space.
- Organize the table such that the lowest address is at the bottom and the highest address is at the top. Each time a new device is added, add it to the memory map, place it in its approximate location in memory and label the starting and ending addresses, in hexadecimal. After inserting all of the devices into the memory map, be sure to label any unused memory regions as such.
- The block diagram of the Printer sharing device shown above contains three devices attached to the address and data buses. These devices are the RAM and ROM and a Serial Controller.
- Let us assume that the RAM is located at the bottom of memory and extends upward for the first 128 KB of the memory space.
- The ROM is located at the top of memory and extends downward for 256 KB. But considering the ROM contains two ROMs-an EPROM and a Flash memory device-each of size 128 KB.
- The third device, the Serial Controller, is a memory-mapped peripheral whose registers are accessible between the addresses say 70000h and 72000h.

- The diagram below shows the memory map for the printer sharing device.



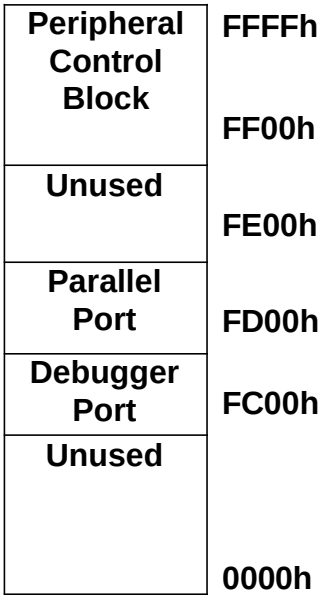
•For every embedded system, a header file should be created that describes these important features and provides an abstract interface to the hardware. It allows the programmer to refer to the various devices on the board by name, rather than by address.

- The part of the header file below describes the memory map

```
#define RAM_BASE      (void *) 0x00000000
#define SC_BASE       (void  *) 0x70000000
#define SC_INTACK     (void  *) 0x70001000
#define FLASH_BASE    (void  *) 0xC0000000
#define EPROM_BASE    (void *) 0xE0000000
```

7.4 I/O MAP

- The I/O map contains one entry for each of the peripheral.
- An I/O map has to be created if a separate I/O space is present. It is done by repeating the steps performed to create memory map.
- To create an I/O map, simply create a table of peripheral names and address ranges, organized in such a way that the lowest addresses are at the bottom.
- The diagram below shows the I/O map for the printer sharing device



•It includes three devices: the peripheral control block (PCB), parallel port, and debugger port. The PCB is a set of registers within the processor that are used to control the on-chip peripherals. The chips that control the parallel port and debugger port reside outside of the processor. These ports are used to communicate with the printer and a host-based debugger, respectively.

- The part of the header file below describes the I/O map

```
#define SVIEW_BASE 0xFC00
#define PIO_BASE 0xFD00
#define PCB_BASE 0xFF00
```

7.5 INTERRUPT MAP

•There are two techniques which can be used by the processor to communicate with memories or peripheral devices. These are:

a. Polling: In this technique the processor polls the device (asks question) repeatedly at regular intervals to check if the device has completed the given task or has any new task to execute.

b. Interrupt:

- An interrupt is a signal sent from a peripheral to the processor. A peripheral may send an interrupt signal to a processor when it has some job to perform which requires the processors intervention.

•Upon receiving an interrupt signal the Processor does the job by issuing certain commands and waits for another interrupt to signal the completion of the job.

•While the processor is waiting for the interrupt to arrive, it is free to continue working on other things.

•When a fresh interrupt signal is received, the processor temporarily sets aside its current work and executes a small piece of software called the interrupt service routine (ISR). When the ISR completes, the processor returns to the work that was interrupted.

•The programmer must write the ISR himself and enable it so that it will be executed when the relevant interrupt occurs.

- **Interrupt Map**
- Embedded systems usually have only a handful of interrupts. Associated with each of these are an interrupt pin which is present on the outside of the processor chip and an ISR.

•In order for the processor to execute the correct ISR, a mapping must exist between interrupt pins and ISRs. This mapping usually takes the form of an interrupt vector table.

•The vector table is usually just an array of pointers to functions, located at some known memory address. The processor uses the interrupt type (a unique number associated with each interrupt pin) as its index into this array. The value stored at that location in the vector table is usually just the address of the ISR to be executed.

•An Interrupt Map is a step taken in this process. The Interrupt Map is a table that contains a list of interrupt types and the devices to which they refer.

•The diagram below shows the Interrupt map for the printer sharing device

Interrupt Type	Generating Device
8	Timer/Counter #0
17	Serial Controller
18	Timer/Counter #1
19	Timer/Counter #2
20	Serial Port Receive
21	Serial Port Transmit

- Once the I/O map is created the header file should be appended with the following information:

```
#define SCC_INT 17                    /*Serial Controller*/

#define TIMER0_INT 8                  /* On-Chip Timer/Counters*/
#define TIMER1_INT 18
#define TIMER2_INT 19

#define RX_INT 20                     /* On-Chip Serial Ports */
#define TX_INT 21
```

7.6 REVIEW QUESTIONS

1. Explain the Components involved in a printer sharing device
2. Explain Memory Map for a printer sharing device
3. Explain I/O Map for a printer sharing device
4. Explain interrupt Map for a printer sharing device

7.7 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

8

EMBEDDED SYSTEMS: MEMORY

Unit Structure

1. OBJECTIVES

After reading this chapter you will understand:

- ✓ Different types of memory available
- ✓ Types of RAM
- ✓ Types of ROM
- ✓ Types of Hybrid Memory

8.1 INTRODUCTION

There are different types of memories available to be used in computers as well as embedded system.

This chapter guides the reader through the different types of memories that are available and can be used and tries to explain their differences in simple words.

8.2 TYPES OF MEMORY

- There are three main types of memories, they are

a) RAM (Random Access Memory)

- It is read write memory.
- Data at any memory location can be read or written.
- It is volatile memory, i.e. retains the contents as long as electricity is supplied.
- Data access to RAM is very fast

b) ROM (Read Only Memory)

- It is read only memory.
- Data at any memory location can be only read.
- It is non-volatile memory, i.e. the contents are retained even after electricity is switched off and available after it is switched on.
- Data access to ROM is slow compared to RAM

c) HYBRID

- It is combination of RAM as well as ROM
- It has certain features of RAM and some of ROM
- Like RAM the contents to hybrid memory can be read and written
- Like ROM the contents of hybrid memory are non volatile

- The following figure gives a classification of different types of memory

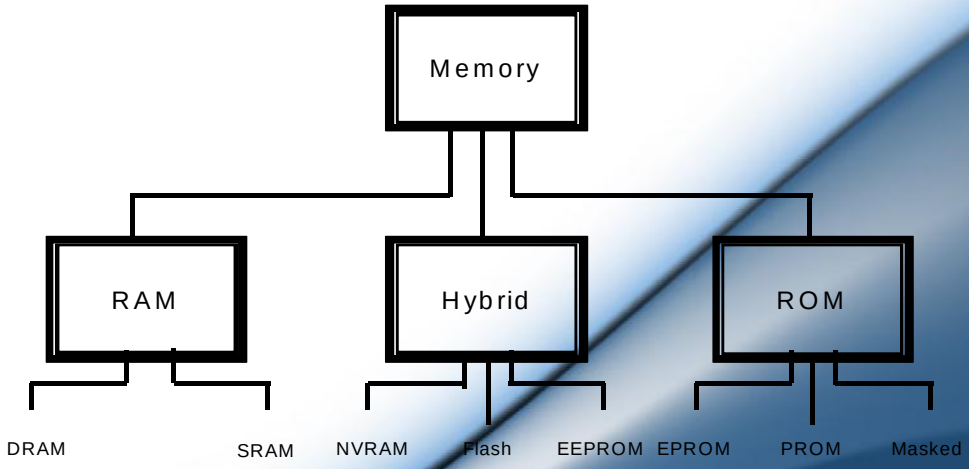


Figure: Types of Memory

8.3 TYPES OF RAM

- There are 2 important memory device in the RAM family.
  a) SRAM (Static RAM)
  b) DRAM (Dynamic RAM)

8.3.1 SRAM (Static RAM)

c) It retains the content as long as the power is applied to the chip.
d) If the power is turned off then its contents will be lost forever.

8.3.2 DRAM (Dynamic RAM)

a) DRAM has extremely short Data lifetime(usually less than a quarter of second). This is true even when power is applied constantly.
b) A DRAM controller is used to make DRAM behave more like SRAM.
c) The DRAM controller periodically refreshes the data stored in the DRAM. By refreshing the data several times a second, the DRAM controller keeps the contents of memory alive for a long time.

4. TYPES OF ROM

There are three types of ROM described as follows:

1. Masked ROM

a. These are hardwired memory devices found on system.
b. It contains pre-programmed set of instruction and data and it cannot be modified or appended in any way. (it is just like an Audio CD that contains songs pre-written on it and does not allow to write any other data)
c. The main advantage of masked ROM is low cost of production**.**

2. PROM (PROGRAMMABLE ROM)

a) This memory device comes in an un-programmed state i.e. at the time of purchased it is in an un-programmed state and it allows the user to write his/her own program or code into this ROM.
b) In the un-programmed state the data is entirely made up of 1's.
c) PROMs are also known as one-time-programmable (OTP) device because any data can be written on it only once. If the data on the chip has some error and needs to be modified this memory chip has to be discarded and the modified data has to be written to another new PROM.

3. EPROM (ERASABLE-AND-PROGRAMABLE ROM)

a) It is same as PROM and is programmed in same manner as a PROM.

b) It can be erased and reprogrammed repeatedly as the name suggests.

c) The erase operation in case of an EPROM is performed by exposing the chip to a source of ultraviolet light.

d) The reprogramming ability makes EPROM as essential part of software development and testing process.

5. TYPES OF HYBRID MEMORY

There are three types of Hybrid memory devices:

1. EEPROMs

a. EEPROMs stand for Electrically Erasable and Programmable ROM.

b. It is same as EPROM, but the erase operation is performed electrically.

c. Any byte in EEPROM can be erased and rewritten as desired

8.5.2 Flash

a. Flash memory is the most recent advancement in memory technology.

b. Flash memory devices are high density, low cost, nonvolatile, fast (to read, but not to write), and electrically reprogrammable.

c. Flash is much more popular than EEPROM and is rapidly displacing many of the ROM devices.

d. Flash devices can be erased only one sector at a time, not byte by byte.

8.5.3 NVRAM

a. NVRAM is usually just a SRAM with battery backup.

b. When power is turned on, the NVRAM operates just like any other SRAM but when power is off, the NVRAM draws enough electrical power from the battery to retain its content.

c. NVRAM is fairly common in embedded systems.

d. It is more expensive than SRAM.

6. DIRECT MEMORY ACCESS (DMA)

- DMA is a technique for transferring blocks of data directly between two hardware devices.

- In the absence of DMA the processor must read the data from one device and write it to the other one byte or word at a time.
- DMA Absence Disadvantage: If the amount of data to be transferred is large or frequency of transfer is high the rest of the software might never get a chance to run.
- DMA Presence Advantage: The DMA Controller performs entire transfer with little help from the Processor.
- Working of DMA
    - The Processor provides the DMA Controller with source and destination address & total number of bytes of the block of data which needs transfer.
    - After copying each byte each address is incremented & remaining bytes are reduced by one.
    - When number of bytes reaches zeros the block transfer ends & DMA Controller sends an Interrupt to Processor.

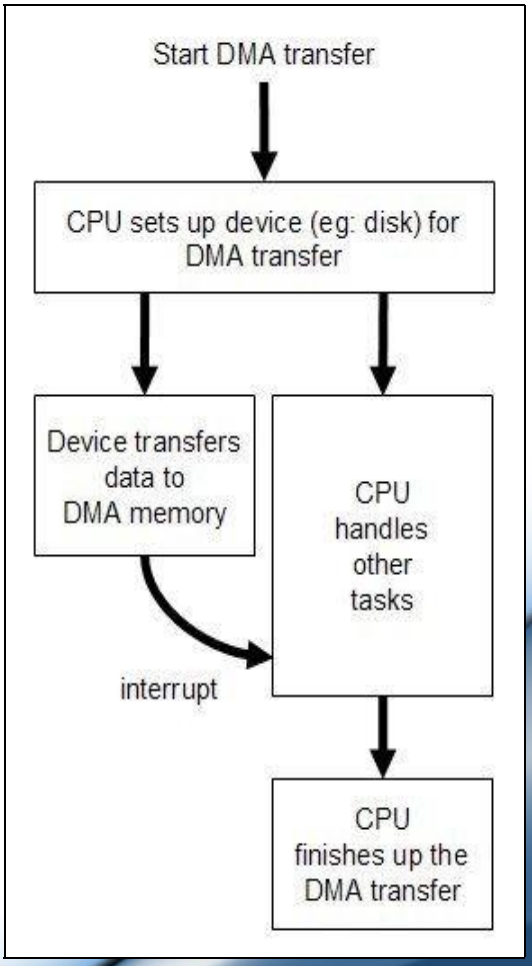


Figure: Direct Memory Access

7. REVIEW QUESTIONS

1. What are the different types of Memory?
2. What are the different types of RAM?
3. What are the different types of ROM?
4. What are the different types of Hybrid Memory?

8. REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

9

EMBEDDED SYSTEMS: MEMORY TESTING

Unit Structure

9.0 OBJECTIVES

After reading this chapter you will be able to understand:

- ✓ What is memory testing?
- ✓ What are the common memory related problems?
- ✓ What are the different types of test to detect memory related problems and a general idea about the working of these tests

1. INTRODUCTION

The previous chapter dealt with the different types of memory. This chapter will focus on the concept of testing memory devices, its purpose and different methods available.

2. MEMORY TESTING AND ITS PURPOSE

- The purpose of a memory test is to confirm that each storage location in a memory device is working.
- Memory Testing is performed when prototype hardware is ready and the designer needs to verify that address and data lines are correctly wired and memory chips are working properly.
- Basic idea implement in testing can be understood by this simple task:

- Write some set of Data values to each Address in Memory and Read it back to verify.
- Ex. If number '50' is stored at a particular Address it is expected to be there unless rewritten or erased.
- If all values are verified by reading back then Memory device passes the test.

- Only through careful selection of data values can make sure passing result to be meaningful.
- Difficulties involved in memory testing:
  - It can be difficult to detect all memory problems with a simple test.
  - Many Embedded Systems include Memory Tests only to detect catastrophic memory failures which might not even notice memory chips removal.

3. COMMON MEMORY PROBLEMS

- Memory Problems rarely occur with the chip itself, but due to a variety of post production tests to check quality this possibility is ruled out.

- Catastrophic Failure is a memory problem that occurs due to physical and electrical damage, it is uncommon and easily detectable.

- A common source of memory problems is associated with the circuit board. Typical circuit board problems are:
  1. Circuit board wiring between Processor & Memory device.
  2. Missing Memory chip.
  3. Improperly inserted Memory chip.

1. Circuit board wiring between Processor & Memory device.

- These are usually caused by,
  - i. An error in design
  - ii. An error in production of the board
  - iii. Any damage after manufacture

- Wires that connect the memory are:- i.
  - ii. Address line :- select the memory location
  - iii. Data line :- transfer the data
- Control line :- read or write operation

Two wiring problems are shown below

1. Connected to another wire on the board

- May be caused by a bit of solder splash

2. Not connected to anything

- Caused by broken trace

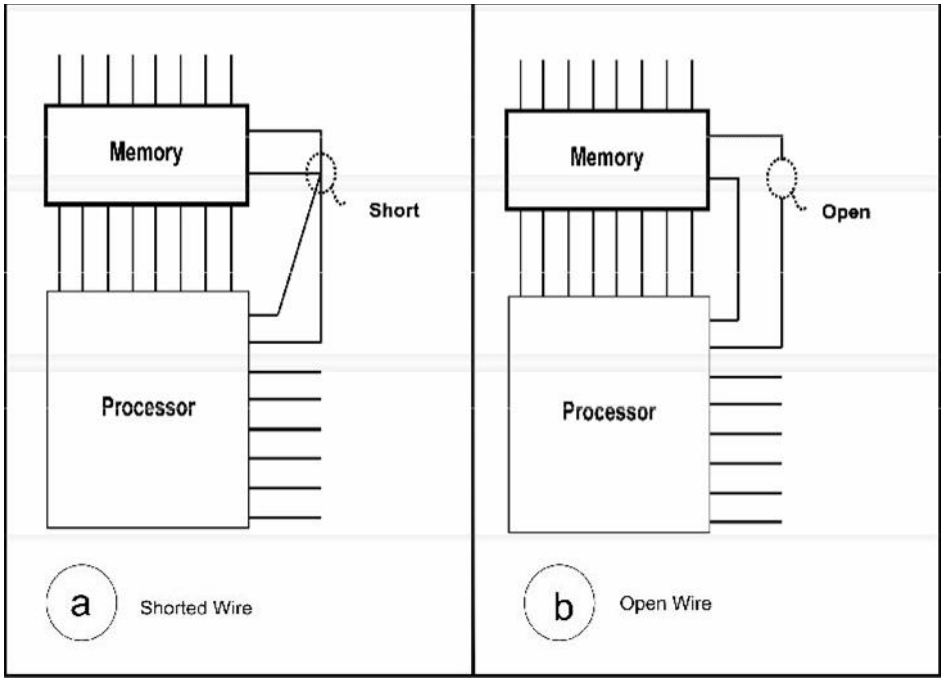


Figure: a. wiring problems: two wires shorted
b. wiring problems: one wire open

- When **Address line** has a wiring problem
  - memory locations overlap
  - i.e. memory device to see an address different from the one selected by the processor.
  - Problem is with a **data line**
  - several data bits "stuck together"
  - i.e. two or more bits always contains same value

- When the problem is with a **Data line**
  - several data bits "stuck together"
  - i.e. two or more bits always contains same value
  - When Control lines is shorted or open

- When **Control lines** is shorted or open
  - The operation of many control lines is specific to the processor or memory architecture.
  - the memory will probably not work at all.

2. Missing Memory chip.

- A missing memory chip is clearly a problem that should be detected
- Unfortunately, because of the capacitive nature of unconnected electrical wires, some memory tests will not detect.

- For e.g. suppose you decided to use the following test algorithm
  - ✓ write the value 1 to the first location in memory, verify the value by reading it back
  - ✓ write 2 to the second location, verify the value
  - ✓ write 3 to the third location, verify, etc.
- Because each read occurs immediately after the corresponding write, it is possible that the data read back represents nothing more than the **voltage remaining** on the data bus from the previous write.
- If the data is read back too quickly, it will appear that the data has been correctly stored in memory-even though there is no memory chip at the other end of the bus!
- To detect a missing memory chip the previous algorithm for test must be altered.
- For example,
  - ✓ write the value 1 to the first location,
  - ✓ 2 to the second location,
  - ✓ And 3 to the third location,
- Then verify the data at the first location, the second location, etc. If the data values are unique (as they are in the test just described), the missing chip will be detected

3. Improperly inserted Memory chip.

- Caused by pins on the memory chip
  - Will either not be connected to the socket at all
  - Will be connected at the wrong place
- Symptoms :-
  - System behaves same as though there is a wiring problem or a missing chip.
- How to detect :-
  - Detected by any test

9.4 A STRATEGY FOR MEMORY TESTING

- For memory testing the strategy adopted should be effective and efficient. Ideally there should be multiple small tests instead of one large test.
- It would be best to have three individual memory tests:
  1. **A data bus test:** Checks electrical wiring problems

2. **An address bus test:** Checks improperly inserted chips
3. **A device test:** Checks to detect missing chips and catastrophic failures and problems with the control bus wiring

- These tests have to be executed in a proper order which is: data bus test first, followed by the address bus test, and then the device test. That's because the address bus test assumes a working data bus, and the device test results are meaningless unless both the address and data buses are known to be good.

1. Data Bus Test

- It is used to check data bus wiring.
- In this test we need to confirm that the received data is same as the data sent by processor

- **Implementation:**
  - Here we write all possible data values and verify that the memory device stores each one successfully.
  - In short to test the bus one bit at a time.

- **Walking 1's test**
  - This test is used to independently test every bit.
  - A single data bit is set to 1 and "walked" through the entire data word.
  - If the data bus is working properly, the function will return 0.
  - Otherwise it will return the data value for which the test failed.
  - Because we are testing only the data bus at this point, all of the data values can be written to the same address. Any address within the memory device will do

00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000

Figure: Consecutive data values for walking 1's test

9.4.2 Address Bus Test

- Address bus problems lead to overlapping memory locations.
- In the Address Bus test we need to confirm that each of the address pins can be set to 0 and 1 without affecting any of the others.
- The smallest set of address that will cover all possible combinations is the set of “power of two” addresses.
- After writing one of the addresses, we must check none of the others has been overwritten.

9.4.3 Device Test

- It is used to test if the memory device is working properly. It is necessary to test the integrity of the memory device itself.
- The thing to test is that every bit in the device is capable of holding both 0 and 1.
- For a thorough and complete device test every memory location has to be visited twice.
- A simple test implemented is the **Increment test** as shown in the table below
  - The first column represents the memory location
  - The second column represents the data that is written at the memory location indicated in column 1 in incremental fashion.
  - The third column represents the data of column 2 in inverted format.

000h	00000001	11111110
001h	00000010	11111101
002h	00000011	11111100
003h	00000100	11111011
… … …	… … …	… … …
… … …	… … …	… … …
… … …	… … …	… … …
0FEh	11111111	00000000
0FFh	00000000	11111111

Figure: Data Bus Test – Increment Test

- During the first pass the data in column 1 is verified and during second pass the data in column 2 is verified.

9.5 REVIEW QUESTIONS

1. What is Memory Testing? Why is it required?
2. What are common memory problems in embedded system?
3. Describe a test strategy for performing memory testing on embedded system. Is there a specific order to perform these tests? if yes, why?
4. Describe the different types of memory testing techniques available.

9.6 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

10

EMBEDDED SYSTEMS: PERIPHERALS

Chapter Structure

10.0 OBJECTIVES

After reading this chapter you will learn:

- ✓ Concept of testing non –volatile memory devices using Checksum and CRC
- ✓ Control and Status Registers
- ✓ Device Driver
- ✓ Watch Dog Timer

1. INTRODUCTION

This chapter initially continues the part of memory testing from last chapter. Here testing of Non Volatile memory devices is studied.

Then we study how peripheral devices are incorporated in Embedded System. Control and Status Registers, Device Drivers and Watch Dog Timers are explained in the subsequent sections.

2. TESTING NON VOLATILE (ROM AND HYBRID) MEMORY DEVICES

- The testing techniques described previously cannot help to test ROM and hybrid devices since ROM devices cannot be written at all, and hybrid devices usually contain data or programs that cannot be overwritten.

- However ROM or hybrid memory device face the same problems as missing memory chip, improperly inserted memory chip, damaged memory chip or wiring problem with the memory chip.

- Two Techniques Checksums and CRC can be used to test non volatile memory devices.

- **Checksum**
  - Checksums basically deals with the question whether the data stored in a memory device is valid or not?
  - To do this the checksum of the data in the memory device is computed and stored along with the data. The moment when we have to confirm the validity of the data, we just have to recalculate the checksum and compare it with previous checksum. If the two checksums match, the data is assumed to be valid.
  - The simplest checksum algorithm is to add up all the data bytes discarding carries.
  - A Checksum is usually stored at some fixed location in memory. This makes it easy to compute and store the check sum for the very first time and later on to compare the recomputed checksum with the original one.
  - Disadvantage: A simple sum-of-data checksum cannot detect many of the most common data errors.

- **CRC – Cyclic Redundancy Check**
  - A Cyclic Redundancy Check is a specific checksum algorithm designed to detect the most common data errors.
  - CRC's are frequently used in Embedded Applications that requires the storage or transmission of large blocks of data.
  - The CRC works as follows:
    - The message is composed of a long string of 0's and 1's
    - A division operation occurs between the message at numerator and the generator polynomial at denominator. The generator polynomial is a fixed smaller length binary string.
    - The remainder of the division operation is the CRC Checksum

3. CONTROL AND STATUS REGISTERS

- Control and status registers are the basic interface between and embedded processor and peripheral device.
- These registers are a part of peripheral hardware and their location size and individual meanings are feature of the peripheral.
- For example, The registers vary from device to device: example the registers within a serial controller are very different from those in a timer.
- Depending upon the design of the processor and target board , peripheral devices are located either in the processor's memory space or within the I/O space.
- It is common for Embedded Systems to include some peripherals of each type. These are called Memory-Mapped and I/O-mapped peripherals.
- Of the two types, memory-mapped peripherals are generally easier to work with and are increasingly popular.
- Memory-mapped control and status registers can be used just like ordinary variables.

4. DEVICE DRIVER

- The goal of designing a device driver is to hide the hardware completely.
- Attempts to hide the hardware completely are difficult.
- For example all Flash memory devices share the concept of sectors. An erase operation can be performed only on an entire sector. Once erased individual bites or words can be rewritten.
- Device drivers for embedded systems are quite different from the workstation counter parts. In modern computers workstation device drivers are most often concerned with satisfying the requirement of the operating system.
- There are three benefits of good device driver:

i. Modularization, it makes the structure of the overall software is easier to understand.

ii. There exists only one module that interacts directly with the peripheral's registers making communication easier.

iii. Software changes that result from hardware changes are localized to the device driver.

- **Components of a Device Driver**

A device driver can be implemented (as components) in the following steps:

1. **A data structure that overlays the memory-mapped control and status registers of the device:**
   - This basic step involves creating a C style structure that is actually a map of the registers present in the device. These registers can be found out by referring to the data sheet for the device.
   - A table is created which maps the control register to their relative offsets.
   - An example is shown below for a timer counter data structure.

```
struct TimerCounter
                {
unsigned short count; // Current Count, offset 0x00
unsigned short maxCountA;// Maximum Count, offset 0x02
unsigned short _reserved;  // Unused Space, offset 0x04
unsigned short control; // Control Bits, offset 0x06
                };
```

   - To make the bits within the control register easier to read and write individually, we define the following bitmasks:

```
#define TIMER_ENABLE 0xC000 // Enable the timer.
#define TIMER_DISABLE 0x4000 // Disable the timer.
#define TIMER_INTERRUPT 0x2000 // Enable timer interrupts.
#define TIMER_MAXCOUNT 0x0020 // Timer complete?
#define TIMER_PERIODIC 0x0001 // Periodic timer?
```

2. **A set of variables to track the current state of the hardware and device driver:** It involves listing out the required variables needed to keep track of the state of the hardware and device driver

3. **Initialize the hardware:** Once the variables to be used are known the next step in device driver programming is to initialize the hardware. Next functions can be written to control the device.

4. **A set of routines that provide an API for users of the device driver**
   This involves writing different functions that will implement the various tasks listed to be performed by the device.

5. **Interrupt service routines**
   Once the required functions and routines are coded the thing remaining to be done is to identify and write routines for servicing the interrupts.

5. WATCHDOG TIMER

- It is hardware equipment.
- It is special purpose hardware that protects the system from software hangs.
- Watchdog timer always counts down from some large number to zero
- This process takes a few seconds to reset, in the meantime, it is possible for embedded software to “kick” the watchdog timer, to reset its counter to the original large number.
- If the timer expires i.e. counter reaches zero, the watchdog timer will assume that the system has entered a state of software hang, then resets the embedded processor and restarts the software
- It is a common way to recover from unexpected software hangs
- The figure below diagrammatically represents the working of the watchdog timer

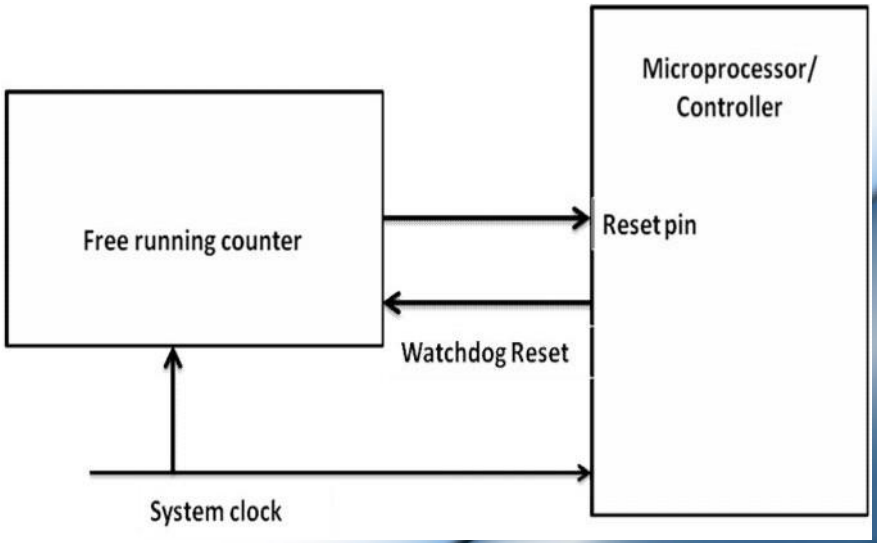


Figure: Watchdog Timer

10.6 REVIEW QUESTIONS

1. Explain testing for non-volatile memory devices
2. Write short note on Control and status registers
3. What is a device driver?
4. What are the components of a device driver?
5. Write short note on Watch Dog Timer

10.7 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

11

EMBEDDED OPERATING SYSTEM

Chapter Structure

1. OBJECTIVES

After reading this chapter you will learn:

- ✓ The basics of embedded Operating system with respect to
  1. Tasks
  2. Task States
- ✓ Scheduler with respect to:
  1. Scheduling Points
  2. Ready List
  3. Idle task
- ✓ Concept of Context Switch and Task Synchronization
- ✓ Real Time Characteristic of embedded operating system.

2. INTRODUCTION

This chapter introduces the readers to the embedded operating system. Any operating system has a set of programs which are implemented through a set of tasks.

Every embedded system may not require an operating system. The requirement and complexity on an operating system depends on the functionality to be implemented by the embedded system.

2. BASICS

1. Tasks

- Task is a piece of code or program that is separate from another task and can be executed independently of the other tasks.

- In embedded systems, the operating system has to deal with a limited number of tasks depending on the functionality to be implemented in the embedded system.

- Multiple tasks are not executed at the same time instead they are executed in pseudo parallel i.e. the tasks execute in turns as the use the processor.

- From a multitasking point of view, executing multiple tasks is like a single book being read by multiple people, at a time only one person can read it and then take turns to read it. Different bookmarks may be used to help a reader identify where to resume reading next time.

- An Operating System decides which task to execute in case there are multiple tasks to be executed. The operating system maintains information about every task and information about the state of each task.

- The information about a task is recorded in a data structure called the ***task context.*** When a task is executing, it uses the processor and the registers available for all sorts of processing. When a task leaves the processor for another task to execute before it has finished its own, it should resume at a later time from where it stopped and not from the first instruction. This requires the information about the task with respect to the registers of the processor to be stored somewhere. This information is recorded in the task context.

- A C++ version of a Task that holds all information needed by operating system is as follows:

```
class Task
{
        public:
                Task(void (*function)(), Priority p, int
                stackSize);

                TaskId id;
                Context context;
                TaskState state;
                Priority priority;
```

```
        int * pStack;
        Task * pNext;

        void (*entryPoint)();

    private:
    static TaskId nextId;
};
```

2. Task States

- In an operation system there are always multiple tasks. At a time only one task can be executed. This means that there are other tasks which are waiting their turn to be executed.

- Depending upon execution or not a task may be classified into the following three states:

  - **Running state** - Only one task can actually be using the processor at a given time that task is said to be the “running” task and its state is “running state”. No other task can be in that same state at the same time

  - **Ready state** - Tasks that are are not currently using the processor but are ready to run are in the “ready” state. There may be a queue of tasks in the ready state.

  - **Waiting state** - Tasks that are neither in running nor ready state but that are waiting for some event external to themselves to occur before the can go for

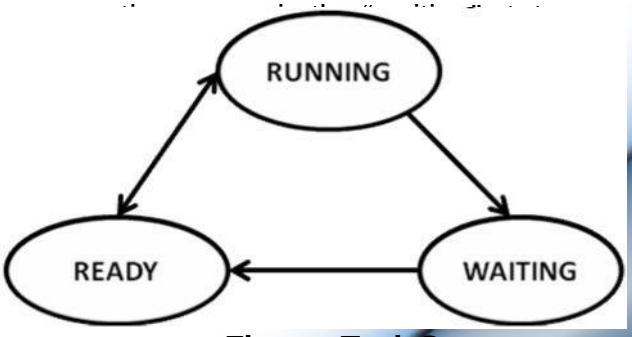


Figure: Task States

- A transition of state between the ready and running state occurs whenever the operating system selects a new task to run.

- The task that was previously in running state becomes ready and the new task is promoted to running state.

- A task will leave running state only if it needs to wait for some event external to itself to occur before continuing.

- A task's state can be defined as follows:
  enum TaskState { Ready, Running, Waiting };

3. SCHEDULER

- The heart and soul of any operating system is its scheduler.
- This is the piece of the operating system that decides which of the ready tasks has the right to use the processor at a given time.
- It simple checks to see if the running task is the highest priority ready task.
- Some of the more common scheduling algorithms:

1. First-in-first-out

- First-in-first-out (FIFO) scheduling describes an operating system which is not a multitasking operating system.

- Each task runs until it is finished, and only after that is the next task started on a first come first served basis.

2. Shortest job first

- Shortest job first scheduling uses algorithms that will select always select a task that will require the least amount of processor time to complete.

3. Round robin.

- Round robin scheduling uses algorithms that allow every task to execute for a fixed amount to time.

- A running task is interrupted an put to a waiting state if its execution time expires.

11.3.1 Scheduling Points

- The scheduling points are the set of operating system events that result in an invocation of the scheduler.

- There are three such events: **task creation** and **task deletion.** During each of these events a method is called to select the next task to be run.

- A third scheduling point called the **clock tick** is a periodic event that is triggered by a timer interrupt. When a timer expires, all of the tasks that are waiting for it to complete are changed from the waiting state to the ready state.

11.3.2 Ready List

- The scheduler uses a data structure called the **ready list** to track the tasks that are in the ready state.
- The ready list is implemented as an ordinary linked list, ordered by priority.
- So the head of this list is always the highest priority task that is ready to run.

11.3.3 Idle task

- If there are no tasks in the ready state when the scheduler is called, the idle task will be executed.
- The idle task looks the same in every operating system.
- The idle task is always considered to be in the ready state.

4. CONTEXT SWITCH

- The actual process of changing from one task to another is called **Context Switch**.
- Since **contexts are processor-specific**, so is the code that implements the context switches, hence, it must always be **written in assembly language**.

5. TASK SYNCHRONIZATION

- All the tasks in the multitasking operating systems work together to solve a larger problem and to synchronize their activities, they occasionally communicate with one another.
- For example, in the printer sharing device the printer task doesn't have any work to do until new data is supplied to it by one of the computer tasks.
- So the printer and the computer tasks must communicate with one another to coordinate their access to common data buffers.
- One way to do this is **to use a data structure called a mutex**.
- Mutexes are mechanisms provided by many operating systems **to assist with task synchronization**.

- A mutex is **a multitasking-aware binary flag**. It is because the processes of setting and clearing the binary flag are **atomic** (i.e. these operations cannot be interrupted).

- When this binary flag is set, the shared data buffer is assumed to be in use by one of the tasks. All other tasks must wait until that flag is cleared before reading or writing any of the data within that buffer.

- The atomicity of the mutex set and clear operations is enforced by the operating system, which disables interrupts before reading or modifying the state of the binary flag.

6. REAL TIME CHARACTERISTIC

An Operating system is called "Real-Time Operating System" (RTOS) only if it has following characteristics:

i. Deterministic

- An OS is said to be deterministic if the worst case execution time of each of the system calls is calculable.

- The data sheet of an OS should publish the real-time behavior of its RTOS provides average, minimum and maximum number of clock cycles required by each system call.

ii. Interrupt Latency

- Interrupt Latency is the total length of time from an interrupt signal's arrival at the processor to the start of the associated interrupt service routine.

iii. Context Switch

- Context Switch is important because it represents overhead across your entire system.

7. SELECTION PROCESS

The process of selecting the best commercial operating system that best fits the needs of one's project depends on various factors.

- Commercial operating systems form a continuum of functionality, performance and price.

- Operating Systems that offer only a basic scheduler and a few other system calls are inexpensive and come with the source code that one can modify and do not require payment of royalties.

- While on the other hand operating systems that include a lot of useful functionality beyond just the scheduler are quite expensive and royalties due on every copy shipped in ROM and they might also make a stronger guarantees about real-time performance.

Two important points to be considered while selecting an operating system :-

- Put your processor, real time performance and budgetary requirements first.
- Contact all of the vendors of the remaining operating systems for more detailed technical information.

11.8 REVIEW QUESTIONS

1. Explain the embedded Operating system with respect to
   i) Tasks
   ii) Task States
2. Explain Scheduler with respect to:
   i) Scheduling Points
   ii) Ready List
   iii) Idle task
3. Write a short note on Context Switch and Task Synchronization
4. Explain the Real Time Characteristic of embedded operating system.

11.9 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

12

EMBEDDED SYSTEMS: INTEGRATED DEVELOPMENT ENVIRONMENT

Chapter Structure

1. OBJECTIVES

After reading this chapter you will understand:

- Embedded IDE
- Types of file involved
- Disassembler/ Decomipiler
- Simulator
- Firmware Debugging and Emulator

1. INTRODUCTION

This chapter explains the IDE used for embedded systems. It then explains the different types of files that are generated on cross compilation. Then it gives an account of utility tools like Disassembler/ Decomipiler, Simulator and then FirmWare Debugging.

2. EMBEDDED IDE

- Integrated Development Environment with respect to embedded system IDE stands for an Integrated Environment for developing and debugging the target processor specific embedded software.

- IDE is a software package which contains:
  1. Text Editor(Source Code Editor)

2. Cross Compiler(For Cross platform development and complier for the same platform development)
3. Linker and debugger.

- Some IDEs may provide an interface to an emulator or device programmer.
- IDEs are used in embedded firmware development.

➢ IDEs may be of two types:

1. **Command Line Base**
   - Turbo C++ IDE is an example for a generic IDE with a Command Line Interface.
2. **GUI Base**
   - Microsoft Visual Studio is an example of GUI base IDE.
   - Others examples are NetBeans, Eclipse.

3. TYPES OF FILE GENERATED ON CROSS COMPILATION

Following are some of the files generated upon cross compilation:

1. List file .lst
2. Hex file .hex
3. Preprocessor output file
4. Map file .map
5. Obj file .obj

1. List File(.lst):-

- Listing file is generated during the cross-compilation process.
- It contains an information about the cross compilation process like cross compiler details, formatted source text('C' code), assembly code generated from the source file, symbol tables, errors and warnings detected during the cross-compilation process.
- The list file contain the following sections:

1. Page Header

- It indicates the compiler version name, source file name, Date, Page No.
- Example: C51 COMPILER V8.02 SAMPLE 05/23/2006 11:12:58 PAGE 1

2.Command Line

- It represents the entire command line that was used for invoking the compiler.
- C51 COMPILER V8.02, COMPILATION OF MODULE SAMPLE OBJECT MODULE PLACED IN sample.obj

- COMPILER INVOKED BY: C:\Keil\C51\BIN\C51.EXE sample.c BROWSE DEBUG OBJECTTEXTEND CODE LISTINCLUDE SYMBOLS

3.Source Code

- It contains source code along with line numbers
- Line level Source

```
1       //Sample.c for printing Hello World!
2       //Written by xyz
3       #include<stdio.h>
1       //Body part starts
2
3
4
5
6       //Body part end
4       void main()
5       {
6       printf("Hello World");
7       }
8       //Header part ends
```

4. Assembly listing

- It contains the asembly code generated by compiler for even given 'C' code.
- ASSEMBLY LISTING OF GENERATED OBJECT CODE;
- FUNCTION main(BEGIN)

```
    ;SOURCE LINE #5
    ;SOURCE LINE #6
        0000  7BFF              MOV  R3,#0FFH
        0002  7A00        R     MOV
          R2,#HIGH?SC_0
```

5. Symbol listing

- It contains symbolic information about the various symbols present in the cross compiled source file.
- Eg: NAME, TYPE, SFR, SIZE.
-

6. Module Information

- The module information provides the size of initialized and un-initialized memory areas defined by the source file.

```
Module Information          Static     Overlayable
Code Size                   9          ------------
Constant size               14         ------------
Bit size                    -------    ------------
END OF MODULE INFORMATION
```

7. Warnings and Errors

- Warnings and Errors section of list file records the errors encountered or any statement that may create issues in application(Warnings), during cross compilation.

- ie:- C51 COMPILATION COMPLETE, 0WARNING(S), 0 ERROR(S).

2. Preprocessor Output File

- It contains preprocessor output for preprocessor instructions used in the source file.

- This file is used for verifying the operation of Macros and preprocessor directive.

3. Object File(.OBJ File)

- Cross-compiling each source module converts the Embedded C/Assembly instructions and other directives present in the module to an object(.OBJ file)

4. Map File(.MAP)

- Also called as Linker List file. Map file contains information about the link/locate process and is composed of a number of sections described below:

I. Page Header

Each MAP file contains a header which indicates the linker version number, date, time and page number.

II. Command Line

Represents the entire command line that was used for invoking the linker.

III. CPU Details

It contains details about the target CPU and its memory model which includes information on internal data memory, external data memory, paged data memory, etc.

IV. Input Modules

It includes the names of all the object files, library files and other files that are included in the linking process.

V. Memory Map

It lists the starting address, length, relocation type and name of each segment in the program

VI. Symbol Table

It contains the name, value and type for all symbols from different input modules.

VII. Inter Module Cross Reference

It includes the section name, memory type and module names in which it is defined and all modules where it is accessed.

Ex.

NAME……..USAGE……..

MODULE NAMES

?CCCASE……CODE;……?C?CCASE PRINTF

?C?CLDOPTR……CODE;……..?C?CLDOPTR PRINTF

?C?CSTPTR……CODE;……..?C?CSTPTR PRINTF

VIII. Program Size

It contains the size of various memory areas, constants and code space for the entire application Ex. Program Size: data=80.1 xdata=0 code 2000

IX. Warnings and Errors

It contains the warnings and errors that are generated while linking a program. It is used in debugging link errors

5. HEX FILE (.hex file)

1. It is a binary executable file created from the source code.
2. The file created by linker/locater is converted into processor understandable binary code.
3. The tool used for converting and object file into a hex file is known as object to Hex converter.
4. Hex file have specific format and it varies for different processor and controller. Two commonly used hex file format are:
   A. Intel Hex
   B. Motorola Hex.
5. Both Intel and Motorola hex file format represent data in the form of ASCII codes.

4. DISASSEMBLER/ DECOMIPILER

- A **Disassembler/ Decomipiler** is a reverse engineering tool.
- Reverse Engineering is used in embedded system to find out the secret behind the working of a proprietary product.
- A **DISASSEMBLER** is a utility program which converts machine codes into target processor specific assembly code/instruction.
- The process of converting machine codes to assembly code is called **disassembling.**
- A **DECOMIPILER** is a utility program for translating machine codes into corresponding high level language instruction.
- A decompiler performs the reverse operation of a compiler/cross-compiler.

5. SIMULATOR

- Simulators are used for embedded firmware debugging.
- Simulator simulates the target hardware, while the code execution can be inspected.
- Simulators have the following characteristics which make them very much favorable:
  - ✓ Purely software based
  - ✓ No need of target system (hardware)
  - ✓ Support only for basic operations
  - ✓ Cannot Support or lack real time behavior

- Advantages

1. Simple and straight forward.

- Simulators are a software utility with assumptions about the underlying hardware. So it only requires concentrating on debugging of the code, hence straight forward.

2. No Hardware

- Simulators are purely software oriented.
- The IDE simulates the target CPU. The user needs to know only about the target specific details like memory map of various devices.
- Since no hardware is required the code can be written and tested even before the hardware prototype is ready thus saving development time

3. Simulation options

- Simulators provide various simulation options like I/O peripherals or CRO or Logic analyzers.

- Simulators I/O support can be used to edit values for I/O registers.

4. Simulation of abnormal conditions

- Using simulator the code can be tested for any desired value.
- This helps to study the code behavior in abnormal conditions without actually testing it on the hardware.

- **Disadvantages**
  1. **Lack of real time behavior**
     - A simulator assumes the ideal condition for code execution.
     - Hence the developer may not be able to debug the code under all possible combinations of input.
     - The results obtained in simulation may deviate from actual results on target hardware.

  2. **Lack of real timeliness**
     - The I/O condition in hardware is unpredictable. So the output of simulation is usually under ideal condition and hence lacks timeliness.

6. FIRMWARE DEBUGGING

- Debugging in embedded application is the process of diagnosing the firmware execution, monitoring the target processor's registers and memory while the firmware is running and checking the signals on various buses of hardware.
- Debugging is classified into Hardware Debugging and Firmware Debugging.
- Hardware Debugging deals with debugging the various aspects of hardware involved in the embedded system.
- The various tools used for hard ware debugging are Multimeter, CRO, Logic Analyzers and Function Generators.
- Firmware Debugging involves inspecting the code, its execution flow, changes to different registers on code execution.
- It is done to find out the bugs or errors in code which produces unexpected behavior in the system.
- There is a wide variety of firmware debugging techniques available that have advanced from basic to advanced.
- Some to the tools used are Simulators and Emulators.

- **Emulators**
  - The terms simulators and emulators are very confusing but their basic functionality is the same i.e. to debug the code. There is a difference in which this is achieved by both the tools.

  - A simulator is a utility program that duplicates the target CPU and simulates the features and instructions supported by target CPU whereas an Emulator is a self contained hardware device which emulates the target CPU.

  - The Emulator hardware contains the necessary emulation logic and is connected to the debugging application that runs on the host PC.

  - The Simulator *'simulates'* while the Emulator *'emulates'*

12.7 REVIEW QUESTIONS

1. Write a Short note on Embedded IDE
2. What is Cross- Compilation? List the files that are generated upon cross compilation
3. Explain the contents of .MAP file.
4. Explain the contents of .LST file.
5. Write short notes on :
   I. .OBJ File
   II. .HEX File
   III. Preprocessor Output File

12.8 REFERENCES & FURTHER READING

Introduction to Embedded systems – Shibu K. V

❖❖❖❖

13

EMBEDDED DEVELOPMENT LIFE CYCLE

Chapter Structure

1. OBJECTIVES

After Reading this chapter you will understand

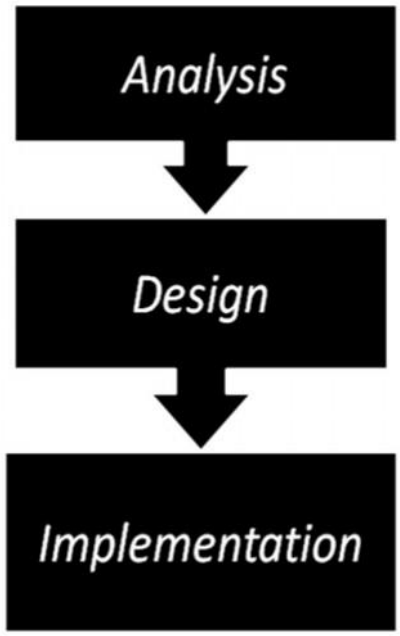
- ✓ The Embedded Development Life Cycle
- ✓ Phases Involved in the EDLC

2. INTRODUCTION

Just like the SDLC used in Software Development, there is EDLC used in Embedded product development. This chapter explains what is the EDLC, its objectives, the phases that are involved in the EDLC.

13.2 EMBEDDED PRODUCT DEVELOPMENT LIFE CYCLE (EDLC)

- ➢ EDLC is Embedded Product Development Life Cycle
- ➢ It is an Analysis – Design – Implementation based problem solving approach for embedded systems development.
- ➢ There are three phases to Product development:



- Analysis involves understanding what product needs to be developed
- Design involves what approach to be used to build the product
- Implementation is developing the product by realizing the design.

1. Need for EDLC

- EDLC is essential for understanding the scope and complexity of the work involved in embedded systems development
- It can be used in any developing any embedded product
- EDLC defines the interaction and activities among various groups of a product development phase.
  Example:-project management, system design

2. Objectives of EDLC

- The ultimate aim of any embedded product in a commercial production setup is to produce Marginal benefit
- Marginal is usually expressed in terms of Return On Investment
- The investment for product development includes initial investment, manpower, infrastructure investment etc.
- EDLC has three primary objectives are:

i. Ensure that high quality products are delivered to user

- Quality in any product development is Return On Investment achieved by the product
- The expenses incurred for developing the product the product are:-

- Initial investment
- Developer recruiting
- Training
- Infrastructure requirement related

ii. **Risk minimization defect prevention in product development through project management**

- In which required for product development 'loose' or 'tight' project management
- 'project management is essential for ' predictability co-ordination and risk minimization
- Resource allocation is critical and it is having a direct impact on investment
- Example:- Microsoft @ Project Tool

iii. **Maximize the productivity**

- Productivity is a measure of efficiency as well as Return On Investment
- This productivity measurement is based on total manpower efficiency
- Productivity in which when product is increased then investment is fall down
- Saving manpower

13.3 DIFFERENT PHASES OF EDLC

The following figure depicts the different phases in EDLC:

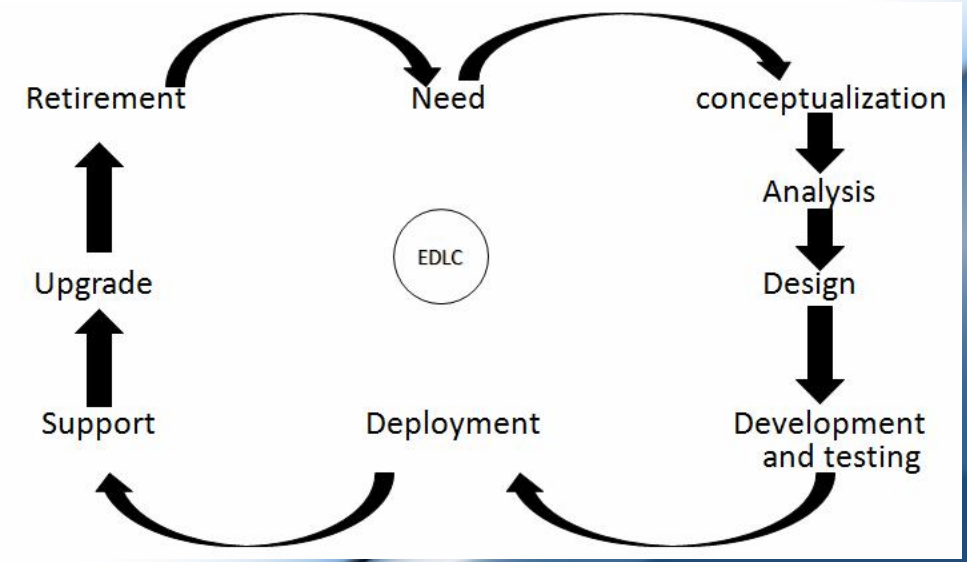


Figure : Phases of EDLC

1. **Need**
   - The need may come from an individual or from the public or from a company.
   - 'Need' should be articulated to initiate the Development Life Cycle; a 'Concept Proposal' is prepared which is reviewed by the senior management for approval.

   - **Need can be visualized in any one of the following three needs:**
     1. New or Custom Product Development.
     2. Product Re-engineering.
     3. Product Maintenance.

2. **Conceptualization**
   - Defines the scope of concept, performs cost benefit analysis and feasibility study and prepare project management and risk management plans.

   - **The following activities performed during this phase:**

     1. **Feasibility Study :** Examine the need and suggest possible solutions.
     2. **Cost Benefit Analysis (CBA):** Revealing and assessing the total development cost and profit expected from the product.
     3. **Product Scope:** Deals with the activities involved in the product to be made.
     4. **Planning Activities:** Requires various plans to be developed first before development like Resource Planning & Risk management Plans.

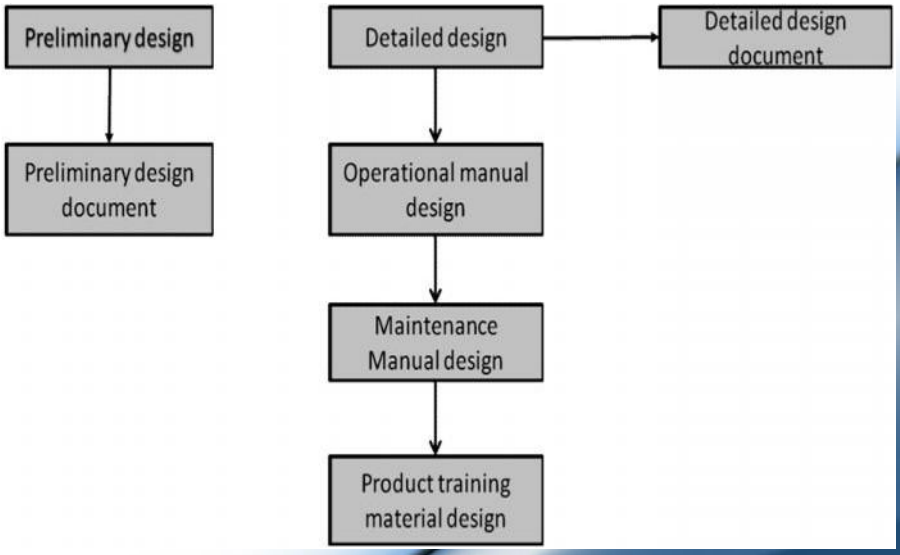
3. **Analysis**
   - The product is defined in detail with respect to the inputs, processes, outputs, and interfaces at a functional level.

   - **The various activities performed during this phase..**
     - **Analysis and Documentations:** This activity consolidates the business needs of the product under development.
     - **Requirements that need to be addressed..**
       - ❑ Functional Capabilities like performance
       - ❑ Operational and non-operational quality attribute
       - ❑ Product external interface requirements
       - ❑ Data requirements
       - ❑ User manuals
       - ❑ Operational requirements
       - ❑ Maintenance requirements
       - ❑ General assumptions

- **Defining Test Plan and Procedures**: The various type of testing performed in a product development are:
  - ❑ **Unit testing –** Testing Individual modules
  - ❑ **Integration testing –** Testing a group of modules for required functionality
  - ❑ **System testing-** Testing functional aspects or functional requirements of the product after integration
  - ❑ **User acceptance testing-** Testing the product to meet the end user requirements.

4. Design

- The design phase identifies application environment and creates an overall architecture for the product.
- It starts with the Preliminary Design. It establishes the top level architecture for the product. On completion it resembles a 'black box' that defines only the inputs and outputs. The final product is called Preliminary Design Document (PDD).
- Once the PDD is accepted by the End User the next task is to create the 'Detailed Design'.
- It encompasses the Operations manual design, Maintenance Manual Design and Product Training material Design and is together called the 'Detailed Design Document'.



5. Development and Testing

- Development phase transforms the design into a realizable product.
- The detailed specification generated during the design phase is translated into hardware and firmware.

- The Testing phase can be divided into independent testing of firmware and hardware that is:
  - ❑ Unit testing
  - ❑ Integration testing
  - ❑ System testing
  - ❑ User acceptance testing

6. Deployment

- Deployment is the process of launching the first fully functional model of the product in the market.
- It is also known as First Customer Shipping (FCS).
- **Tasks performed during this phase are:**
  - **Notification of Product Deployment:** Tasks performed here include:
    - ❑ Deployment schedule
    - ❑ Brief description about the product
    - ❑ Targeted end user
    - ❑ Extra features supported
    - ❑ Product support information
  - **Execution of training plan**

    Proper training should be given to the end user top get them acquainted with the new product.
  - **Product installation**

    Install the product as per the installation document to ensure that it is fully functional.
  - **Product post Implementation Review**

    After the product launch, a post implementation review is done to test the success of the product.

7. Support

- The support phase deals with the operational and maintenance of the product in the production environment.
- Bugs in the product may be observed and reported.
- The support phase ensures that the product meets the user needs and it continues functioning in the production environment.
- Activities involved under support are
  - ❑ **Setting up of a dedicated support wing**: Involves providing 24 x 7 supports for the product after it is launched.

- **Identify Bugs and Areas of Improvement:** Identify bugs and take measures to eliminate them.

8. Upgrades

- Deals with the development of upgrades (new versions) for the product which is already present in the market.
- Product upgrade results as an output of major bug fixes.
- During the upgrade phase the system is subject to design modification to fix the major bugs reported.

9. Retirement/Disposal

- The retirement/disposal of the product is a gradual process.
- This phase is the final phase in a product development life cycle where the product is declared as discontinued from the market.
- The disposal of a product is essential due to the following reasons
  - Rapid technology advancement
  - Increased user needs

13.4 ELDC APPROACHES

Following are some of the different types of approaches that can be used to model embedded products.

1. Waterfall or Linear Model
2. Iterative/ Incremental or Fountain Model
3. Prototyping Model
4. Spiral Model

13.5 REVIEW QUESTIONS

1. What is EDLC? Why is it needed? What are its objectives?
2. Draw an neat labeled diagram of the phases of the EDLC and explain any two phases in detail.

13.6 REFERENCES & FURTHER READING

Introduction to Embedded systems – Shibu K. V

14

EDLC MODELS

Chapter Structure

1. Objectives
2. Introduction
3. Waterfall or Linear Model
4. Iterative/ Incremental or Fountain Model
5. Prototyping Model
6. Spiral Model
7. Review Questions
8. References & Further Reading

1. OBJECTIVES

After reading this chapter you will understand:

- ✓ Some EDLC Models like:
  - ➢ Waterfall or Linear Model
  - ➢ Iterative/ Incremental or Fountain Model
  - ➢ Prototyping Model
  - ➢ Spiral Model

2. INTRODUCTION

The previous chapters introduced the readers to what is meant by EDLC. This chapter is meant to explain the various models available under the EDLC.

3. WATERFALL MODEL

- ➢ Linear or waterfall model is the one adopted in most of the olden systems.
- ➢ In this approach each phase of EDLC (Embedded Development Product Lifecycle) is executed in sequence.
- ➢ It establishes analysis and design with highly structured development phases.
- ➢ The execution flow is unidirectional.

- The output of one phase serves as the input of the next phase
- All activities involved in each phase are well planned so that what should be done in the next phase and how it can be done.
- The feedback of each phase is available only after they are executed.
- It implements extensive review systems To ensure the process flow is going in the right direction.
- One significant feature of this model is that even if you identify bugs in the current design the development process proceeds with the design.
- The fixes for the bug are postponed till the support phase.
- Advantages
    - Product development is rich in terms of:
        - Documentation
        - Easy project management
        - Good control over cost & Schedule
- Drawbacks
    - It assumes all the analysis can be done without doing any design or implementation
    - The risk analysis is performed only once.
    - The working product is available only at the end of the development phase
- Bug fixes and correction are performed only at the maintenance/support phase of the life cycle.

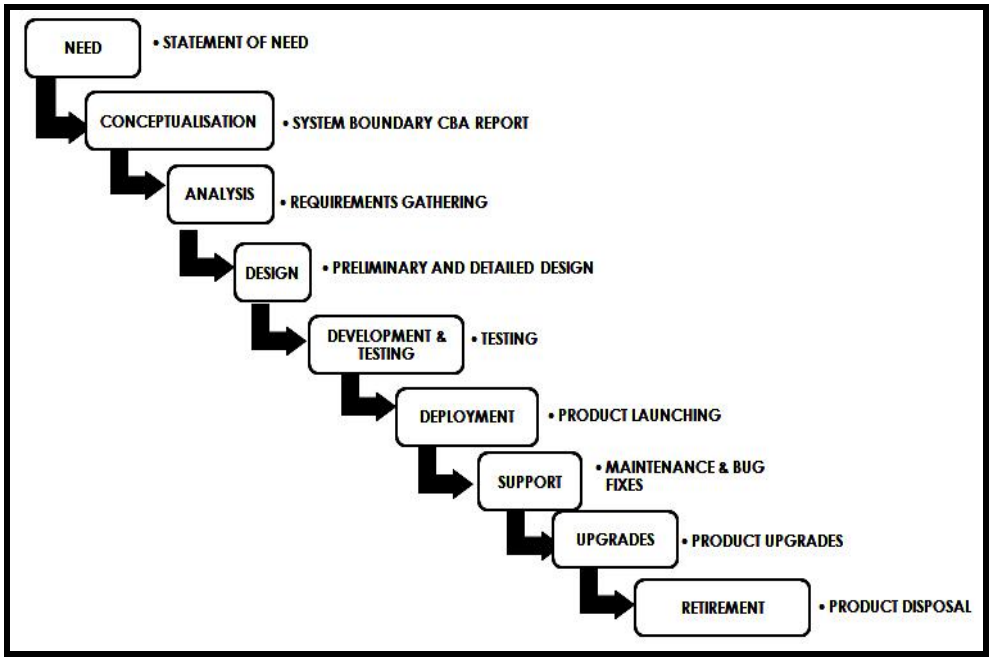


Figure: Waterfall Model

14.3 ITERATIVE/ INCREMENTAL OR FOUNTAIN MODEL

- Iterative and Incremental development is at the heart of a cyclic software development process developed in response to the weaknesses of the waterfall model.
- The iterative model is the repetitive process in which the Waterfall model is repeated over and over to correct the ambiguities observed in each iteration.

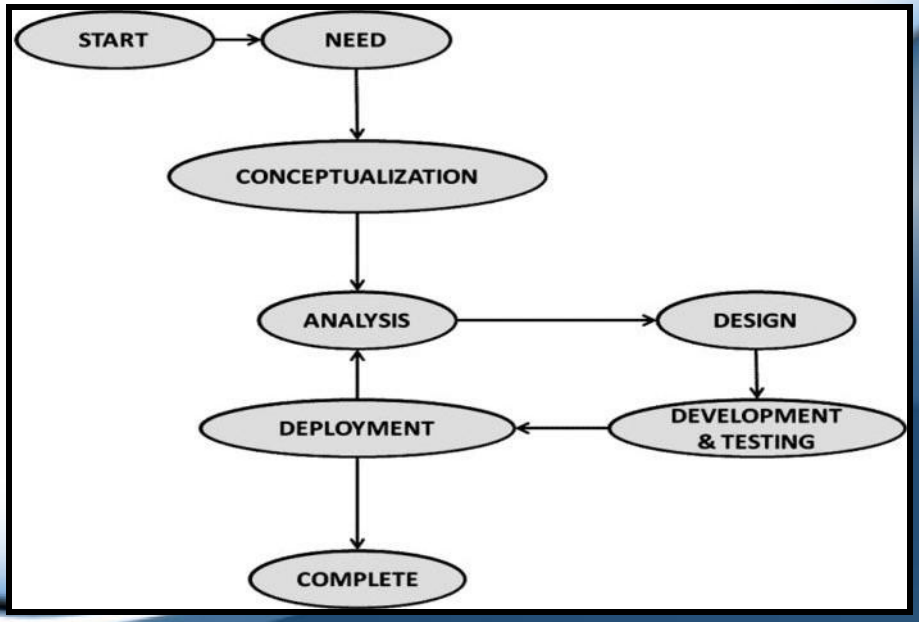


Figure: Iterative model

- The above figure illustrates the repetitive nature of the Iterative model.
- The core set of functions for each group is identified in the first cycle, it is then built, deployed and release. This release is called as the first release.
- Bug fixes and modification for first cycle carried out in second cycle.
- Process is repeated until all functionalities are implemented meeting the requirements.

- **Advantages**
  - Good development cycle feedback at each function/feature implementation
  - Data can be used as reference for similar product development in future.
  - More responsive to changing user needs.
  - Provides working product model with at least minimum features at the first cycle.
  - Minimized Risk
  - Project management and testing is much simpler compared to linear model.
  - Product development can be stopped at any stage with a bare minimum working product.

- **Disadvantages**
  - Extensive review requirement each cycle.
  - Impact on operations due to new releases.
  - Training requirement for each new deployment at the end of each development cycle.
  - Structured and well documented interface definition across modules to accommodate changes

4. PROTOTYPING MODEL

- It is similar to iterative model and the product is developed in multiple cycles.
- The only difference is that, Prototyping model produces a refined prototype of the product at the end of each cycle

Thank You