Chapter – 2

2. Features of an Embedded System

Embedded systems products have been effectively used not only in our day to day used products but these are also used as wholly or partially unavoidable components in many high end uses like military, scientific research, telecommunication etc. Its size may confined from hand held cell phones to components of nuclear missile. Irrespective of its size it necessarily consist of some hardware and software designed to work on hardware. So Embedded Systems are called Product of Hardware and Software Codesign. Features of different hardware and software units of embedded systems are explained in the following sections.

2.1 Hardware features of standalone embedded systems

Standalone embedded system includes different types of processors, power supply unit, clock, reset circuit, memories which are considered to be most essential hardware components of standalone embedded systems [1]. A brief discussion on important features of these components are given below.

2.1.1 Different types of processors used

Processor: A processor is the heart of the embedded system. It is responsible for execution of instruction and controlling flow of data to and from processor. Designer should have proper knowledge regarding efficiency of different types of processor and based on which one should select the appropriate processor as per requirement.

Different types of processors available can be categorized into four broad categories:
(1) General purpose processor (GPP)
(2) Application specific System processor (ASSP)
(3) Multiprocessor system
(4) GPP core or ASIP core.

A GPP has the usage advantages over other processors because of:
a) Having predefined known instruction set resulting fast system development.
b) Board and I/O Interfaces designed for GPP can be used for different system changing the software.
c) Ready availability of computer facilities in high level language along with compiler and debugger, resulting in fast development of a new system.
1. General Purpose Processor (GPP) may be any one of Microprocessor, Microcontroller, Embedded processor, Digital Signal Processor (DSP), and Media processor [2].

Microprocessor is a single VLSI chip that has a CPU with caches, floating point processing arithmetic unit, pipelining and super scaling units. Later units may present for faster processing. RAM is externally connected to CPU.

Microcontroller is also a single chip VLSI unit with limited computational capability keeping all functional units/components inside the chip.

Embedded processor may be microprocessor or microcontroller when design specially to achieve capabilities of fast context switching resulting lower latency, atomic ALU operation with no shared data problem RISC core for fast and precise calculations. ARM family processors, Intel i960 etc belong to this class.

DSP as a GPP is a single chip VLSI having computational capabilities of a microprocessor and a Multiply and Accumulate (MAC) unit(s). DSP is an essential unit of an embedded system with very large instruction word (VLWI) processing capabilities. It process very efficiently Single Instruction Multiple Data (SIMD), Discrete Cosine Transformation (DCT) and Inverse Discrete Cosine Transformation (IDCT). DCT and IDCT are most useful for algorithms for signal analysing, coding, filtering noise cancellation, echo-elimination etc.

2. Application Specific System Processor (ASSP)

ASSP is dedicated for faster processing and useful for applications like realtime video processing which incorporates lots of processing before transmitting. It may also include some features of RTOS. ASSP provides hardwired solution for most of its time consuming tasks. For example ASSP chip i2chip (http://www.i2chip.com) has TCP, UDP, IP, ARP and Ethernet 10/100 MAC (Media Access Control) hardwired logic included into it. In practice, an ASSP is used as an additional processing unit for running the application specific tasks in place of processing using embedded software.

3. Multiprocessor System

As Embedded algorithm has to work within strict deadline, sometimes it may not be possible to carryout the same with a single processor. In a real time video processing number of MAC operations required may be more than possible from one DSP unit. In such a case an embedded system may go for two or more processors. Similar requirement may be needed in modern cell phones which has to perform number of tasks. Multiprocessors are used when a single processor doesn't meet the need of the different tasks that have to be
performed concurrently. The operations of all processors are synchronized to obtain an optimum performance.

4. GPP or ASIP core

GPP core or ASIP core is integrated into either an Application Specific Integrated Circuit (ASIC) or a VLSI or an FPGA (Field programming Gate Array) core integrated with processor units. Lately a new innovation in this area is System On Chip (SOC). A SOC may be embedded with multiple processors, memories, multiple standard source solutions called IP (Intellectual Property) core and other logic and analog units. It may have also a network protocol embedded on it. It can embed DSP applications and FPGA core.

For a number of applications GPP core may not be a suitable solution. For various security application, smart card, video game, mobile Internet, Gbps transceiver, Gbps LAN, missile system needs a special processing unit on a VLSI design circuit to function as a processor. These units are called Application Specific Instruction Processor (ASIP). Sometime for an application both configurable processor (FPGA or ASIP) and non-configurable processor (DSP or microprocessor or microcontroller) might be needed on a chip. Generally this type of applications are very important in some killer applications (application which is useful to millions of people) such as HDTV, cell-phone etc.

2.1.2 Power supply unit.

Generally Embedded system has its own Power supply unit. Four range of voltage (i) 5.0V + 0.25V (ii) 3.3V + 0.3V (iii) 2.0V + 0.2V (iv) 1.5V + 0.2V are use for operation of different units. Additionally 12V + 0.2V supply is needed for a flash or EEPROM and RS232 Serial Interfaces. Supply of voltage to the chip depends on number of pins provided in the chip which is generally in pair supply and ground. A processor may have more than two pins of Vdd Vss which are responsible for distribution of power and reduction of inferences in all the sections. Supply should separately power the (a) External I/O driving port (b) timers and (c) clock and reset circuits. Clock and reset circuit should be specially designed to be free from radio frequency inference [3].

Embedded systems which don’t have a power source at their own are either connected to an external power supply or use charge pump for necessary power supply. Example of first type may be Network Interface Card (NIC) and Graphics accelerator which donot have their own power supply are connected to PC power-supply line. In the second type charge pump brings power from a non-supply line. It consist of a diode in the series followed by a charging capacitor. The diode gets forward bias input from an external signal, say
RTS(Request To Send) signal in case a mouse used with the computer. The charge pump inside the mouse store charge in inactive state and dissipate power when the mouse is used.

An embedded system has to perform tasks continuously from power-up to power-off and may even be kept on continuously. Real Time Systems(RTOS) use Wait and Stop instructions and disabling certain units when not needed. This indeed is very important for saving power during program execution. Performing tasks at reduce clock rate is also a way to control power dissipation. Performance of software analysis during design phase can include power dissipation considerations also. A good design must optimize the conflicting needs of low power dissipation and fast efficient program execution.

2.1.3 Clock Oscillator

The function of this oscillator circuit is to provide an accurate and stable periodic clock signal to a processor. The processor needs a clock oscillator as clock controls the various clocking requirements of CPU. The clocking requirements are the system timers and CPU machine cycles. The machine cycle includes (i) Fetching code and data from memory and decoding and execution and (ii) Transferring results to memory. The clock controls the time for executing an instruction. The clock circuit uses either a crystal (External to the processor) or a ceramic resonator (internally associated with the processor) or an external IC attached to the processor. (a) The crystal resonator gives the highest stability in frequency with temperature and drift in the circuit. The crystal in association with an appropriate resistance in parallel and a pair of series capacitance at both pins. The crystal is kept as near as feasible to the two pins of the processor. (b) The internal ceramic generator, if available in a processor, saves the use of the external crystal and gives a reasonable though not very high frequency. (c) The external IC based clock oscillator has a significantly higher power dissipation compared to the internal processor resonator. It provides a higher driving capability, which might be needed when various embedded circuits of embedded systems concurrently driven for e.g. in multiprocessor based systems [4].

2.1.4 Real time clock or timer units

A timer is suitably configured as system clock sometime referred as RTC (Real Time Clock). RTC is used by scheduler for real time programming. A hardware timer is a counter that is incremented at a fixed rate when the system clock pulses. There are several different types of timers available. A timer/counter can perform several different tasks. The CPU uses the timer to keep track of time accurately. The timer can generate a stream of pulses or a single pulse at different frequencies. It can be used to start and stop tasks at desired times. A
COP (computer operating properly) or watchdog timer checks for runaway code execution. The hardware implementation of watchdog timers varies considerably between different processors. In general watchdog timers must be turned on once within the first few cycles after reset and then reset periodically with software. Some watchdog timers can programmed for different time-out delays. The reset sequence is sometimes as simple as a specialized instruction or as complex as sending a sequence of bytes to a port. Watchdog timers either reset the processor or execute an interrupt when they time out [5].

More than one timers using the RTC may be needed for various timing and counting need. There may be hardware and software implementations of timers. At least one hardware timer device is must in a system which is used as system clock. The hardware timer gets the input from a clock out signal from processor and activates the system clock as per the number ticks preset at the hardware timer. Number of hardware timers present are generally limited.

A software timer is a software that executes and increases or decreases a count variable (count value) or an interrupt on a timer output or on a real time clock interrupt. A software timer can also generate interrupt on overflow of count value or the final value of count variable. Software timers are used as virtual timing devices. There are number of control bits and time out status flags in each timer device. A timer device when given count inputs, in place of clock pulses performs as a counting device.

2.1.5 Interrupt Handlers
A system possesses a number of devices and the system processor has to control and handle the requirements of devices by running appropriate Interrupt Service Routine(ISR) for each. An interrupt handling mechanism must exist in each system to handle interrupt from various processes in the system. An interrupt is an event that suspends regular program operation while the event is serviced by another program. Interrupts increase the response speed to external events. Different microcontrollers have different interrupt sources which can include external, timer and serial port interrupts. When an interrupt is received the current operation is suspended, the interrupt is identified and the controller jumps (vectors) to an interrupt service routine. There are two sources of interrupt: hardware and software. Hardware interrupts include a signal to a pin, timer overflow, and serial port interrupts. Software interrupts are commands given by the programmer. There are two different interrupt types: maskable and non-maskable. A maskable interrupt can be disabled and enabled while non-maskable interrupts can not be disabled and are therefore always enabled. Most 8 bit microcontrollers use vectored arbitration interrupts. Vectored arbitration means that when a specific interrupt occurs the interrupt handler automatically branches to an address associated
with that interrupt. The servicing of interrupts in general is dictated by the status of the GIE (Global Interrupt Enable). GIE is cleared when an interrupt occurs and all interrupts are delayed until it is set.

### 2.1.6 Reset circuit and Watchdog timer

Reset instruction start execution from starting address otherwise execution start from this address when it is powered up. The reset circuit activates for a fixed period (a few clock cycles) and then deactivates to let the program proceed from a default beginning address. On deactivation of the reset that succeed the processor activation, a program executes from start up address. Reset can be activated either by external reset circuit that activates on power up or by software instruction or by a programmed timer known as watchdog timer.

Watchdog timer is a timing device that resets the system after a predefined timeout. This time is usually configured and the watchdog timer is activated within the first few clock cycles after power up. It has many applications. In many embedded systems reset by a watchdog timer is very essential because it helps in rescuing the system from program hangs. On restart program can function normally [6].

### 2.1.7 Memories

Embedded system makes use of different types of memories based on their features. These can be viewed with following chart. These may be briefly explained based on their functionality [7].

1. Internal RAM used for registers, temporary data and stack.
2. Internal ROM/PROM/EPROM for application program.
3. External RAM for temporary data and stack.
4. Internal cache available in case of some microcontroller or microprocessor.
5. EEPROM of flash memory for saving the results.
6. External ROM or PROM for embedding software used in non-microcontroller based systems.
7. RAM memory buffers at ports. Caches for superscalar microprocessors.

![Various Forms of System Memory](image)

**Figure 2.1: Various forms of system memory**
Different types of memory devices in varying sizes are available for use as per requirement. These are (a) Masked ROM or EPROM of flash which stores the embedded software (ROM image). Masked ROM is for bulked manufacturing. (2) EPROM or EEPROM is used for testing and design stages. (3) EEPROM (5V form) is used to store the results during the system program run time. It is erased byte by byte and written during the system run. It is useful to store modifiable bytes for example run time system status, time and date. Flash is very useful when a processed image or voice is to be stored or a data set or system configuration data is to be stored which can be upgraded as and when required. In a flash new images after compressing and processing can be stored and the old one is erased from a sector in a single instruction cycle. In boot block flash a OPT sector is reserved to store once only at the time of first boot. It stores boot program and initial data or permanent system configuration data. This OTP sector can be used to store ROM image. (4) RAM is mostly used in SRAM form in a system. Advanced system uses RAM in the form of a DRAM, EDO RAM, SDRAM, or RDRAM (5) Parameterised distributed RAM is used when I/O devices and subunits require a memory buffer. (6) Subunits like MAC which operates at fast speed uses separate blocks of RAM.

2.1.8 Input / Output units and buses

The system gets input from physical devices such as keypads/boards, sensors, transducer circuits etc. It gets the values by read operations at the port address. The system has output ports through which it sends output bytes to the real world. It sends the values to output by a write operation at the port address. In case of some devices a port may be used as both input as well as output port. One example is mobile phone which sends as well as receives signals. There are two types of I/O ports (i) Parallel port and (ii) Serial port. In a serial port, system gets a serial stream of bits at an input and sends the signal as bits through a modem. A serial port facilitates long distance communications and interconnections. A serial port may be serial URAT, a serial synchronous port or serial interfacing port. A system may get inputs from multiple channels or may have to send multiple output channels. A demultiplexer takes input from various channels and transfers the input to a selected channel. A multiplexer takes output from the system and sends it to another system. A system might have to be connected to a number of other devices and systems. For networking system there are different types of buses eg, I2C, CAN, USB, ISA, EISA and PCI.
2.1.9 DAC/ADC

For automatic control and signal processing applications, a system must provide necessary interfacing circuit and software for Digital to Analog Conversion (DAC) unit and Analog to Digital Conversion (ADC) unit. A DAC operation is done with the help of a combination of PWM (Pulse Width Modulation) unit in the microcontroller and External Integrator chip. ADC operations are needed in systems for voice processing, Instrumentation, Data acquisition systems and automatic control.

2.2 Special features of a Network enabled embedded system

Two ways of connecting embedded systems are using CAN (Controller Area Network) and Ethernet [8]. CAN is a network for Industrial applications suitable for noisy conditions and electrically severe environments. Ethernet makes it possible to connect embedded systems with computers through switch, routers or gateways.

2.2.1 Networking through CAN

CAN was originally developed in Europe by Bosch for automobiles. Automobile industry had faced increase complexity in engine management in late 70s and 80s due to induction of ABS breaking, active suspension, electronics suspension, automated lighting, central locking etc. in most of the automobiles. As each of these cannot work in isolation instead works in a integrated way after considerable amount of information is exchanged [9]. Conventional method of point to point wiring leads to complexity and proved to be inadequate in terms of cost, reliability, weight because sometime it may have several kilometres of wiring. Thus CAN is introduced implementing real time communication up to 1 Mbps, over a two wire serial network. CAN network uses wired AND logic, with a maximum bus length of 1000 meters and a bus length of 40 meters at a maximum data rate of a twisted pair. CAN specifies only physical and Data link layer of ISO-OSI model, leaving higher layer to specific implementation. CAN network is now expanded and found in industrial automations, ships, trains and control systems. It is internationally standardized under ISO11898 and ISO1159-2.
CAN has multiple physical layers. These physical layers classify certain aspects of the CAN network, such as electrical levels, signalling schemes, cable impedance, maximum baud rates and more. The most common and widely used physical layers are described below:

**High-Speed CAN**

High-speed CAN networks are implemented with two wires and allow communication at transfer rates up to 1 Mbps. Typical high-speed CAN devices include anti-lock brake systems, engine control modules and emissions systems.

**Low-Speed/Fault-Tolerant CAN Hardware**

Low-speed fault-tolerant CAN networks are also implemented with two wires. CAN communicate with devices at rates up to 125 kbps and offer transceivers with fault-tolerant capabilities. Typical low-speed fault-tolerant devices in an automobile include comfort devices. Wires that have to pass through the door of a vehicle are low-speed fault-tolerant in light of the stress that is inherent to opening and closing a door. Also, in situations where an advanced level of security is desired, such as with brake lights, low-speed fault-tolerant CAN offers a solution.
Single-Wire CAN Hardware

Single-wire CAN interfaces can communicate with devices at rates up to 33.3 kbps (88.3 kbps in high-speed mode). Typical single-wire devices within an automobile do not require high performance. Common applications include comfort devices such as seat and mirror adjusters.

Software-Selectable CAN Hardware

With National Instruments CAN hardware products, it is possible to configure the software-selectable CAN interfaces to use any of the onboard transceivers (high-speed, low-speed fault-tolerant, or single-wire CAN). Multiple-transceiver hardware offers the perfect solution for applications that require a combination of communications standards. With software-selectable CAN hardware, choosing own external CAN transceiver.

CAN Terminology

CAN devices send data across the CAN network in packets called frames in the following format.

CAN Frame -- an entire CAN transmission: arbitration ID, data bytes, acknowledge bit, etc. Frames also are referred to as messages.

<table>
<thead>
<tr>
<th>SOF</th>
<th>11-BIT ARBITRATION ID</th>
<th>R</th>
<th>18-BIT ARBITRATION ID</th>
<th>R</th>
<th>0</th>
<th>DLC</th>
<th>0...8 BYTES DATA</th>
<th>CRC</th>
<th>ACK</th>
<th>EOF</th>
</tr>
</thead>
</table>

Figure 2.3: CAN Frame

SOF (start-of-frame) bit -- indicates the beginning of a message with a dominant (logic 0) bit

Arbitration ID -- Identifies the message and its priority. Frames come in two formats: standard, which uses an 11-bit arbitration ID, and extended, which uses a 29-bit arbitration ID.

IDE (identifier extension) bit -- allows differentiation between standard and extended frames

RTR (remote transmission request) Differentiate a remote frame from a data frame. A dominant (logic 0) indicates a data frame and (logic 1) RTR bit a remote frame.

DLC (data length code) -- indicates the number of bytes the data field contains
Data Field — contains 0-8 bytes of data

CRC (cyclic redundancy check) — 15-bit CRC used for error detection.

ACK (Acknowledgement) slot — After correctly receiving the message, CAN controller sends an ACK bit at the end of the message. The transmitting node checks for the presence of the ACK bit on the bus and reattempts transmission if no acknowledge is detected.

CAN Signal — an individual piece of data contained within the CAN frame data field usually referred as channels. Because the data field can contain up to 8 bytes of data, a single CAN frame can contain 0 to 64 individual binary signals. For each signal, following data is stored in databases:

- Channel name
- Location (start bit) and size (number of bits) of the channel within a given message
- Byte order (Intel/Motorola)
- Data type (signed, unsigned, and IEEE float)
- Scaling and units string
- Range
- Default value
- Comment

CAN database files may contain frame and signal definitions for an entire vehicle. Every network has its own unique database file. Additionally, these database files are vendor-specific and usually confidential. By using a database file for many frames on the CAN network, can automatically convert the frame information directly to a real world value. This simplifies application development.

**How CAN Communication Works**

CAN is a peer-to-peer network [10]. This means that there is no master that controls when individual nodes have access to read and write data on the CAN bus. Each CAN node having its local mechanism for sensing and control as shown in CAN distributed system. When a CAN node is ready to transmit data, it does a check to see if the bus is busy, and then simply writes a CAN frame onto the network. The CAN frames that are transmitted do not contain addresses of either the transmitting node or any of the intended receiving node(s). Instead, an arbitration ID that is unique throughout the network labels the frame. All nodes on
the CAN network receive the CAN frame, and, depending on the arbitration ID of that transmitted frame, each CAN node on the network decides whether to accept the frame. If multiple nodes try to transmit a message onto the CAN bus at the same time, the node with the highest priority (lowest arbitration ID) automatically gets bus access. Lower-priority nodes must wait until the bus becomes available before trying to transmit again. In this way, CAN networks is implemented to ensure deterministic communication among CAN nodes.

Many processors which has to be used in electrically noisy industrial application include CAN module. Some processor also include CAN interface module instead of using CAN which provides most of the functionality of CAN. Additional support required in such a case is a CAN driver.

2.2.2 Networking through Ethernet

Ethernet network can be made available to embedded systems by two ways one by adding Ethernet controller to embedded systems and other by attaching embedded circuit board to a node of Ethernet network through serial or USB connectivity.

Considering Ethernet controller as an example cirrus logic produces a single chip Ethernet controller known as the CS8900A. This chip allows to attach a 10 Mbps Ethernet interface to embedded system. The block diagram in Figure 2.4 shows a CS8900A implementation.

The CS8900A is connected to its 10BASE-T port through an isolation transformer. The transformer must have a winding ratio of 1:1 for receiver and a winding ratio n of 1:1.41 for the transmitter provided that CS8900A is using a 5 V supply. The CSA8900A can also directly drive LEDs, indicating Ethernet link status and bus network activity. An external 20 MHz crystal provides timing for CS8900A. The crystal is connected across the XTAL1 and XTAL2 pins and each pin is bypassed to ground using 33pF capacitors. This supports 16 bit ISA bus architecture. The ISA may easily be adapted to work with a range of non ISA processors. So it can be implemented with variety of computers as shown in figure 2.4

The other way of connection is very simple as actual networking will be formed by interconnection of different computers with which embedded system is connected by serial port, (RS 232) or USB. All the information of embedded system can be maintained in database files of the computer in addition to designing user friendly front end for monitoring and controlling of the system. Connecting computer shall be a node of the LAN/WAN and can be
configured either as embedded server or database server or both depending on the requirement. LAN/WAN will have its normal architecture connected through switches and/or Routers.

Figure 2.4: Crystal connections for the CS8900A

Networking is done through 8 pin RJ-45 connector connecting UTP cables such as CAT5 which follows following colour scheme for easy identification as shown in the table 2.1.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal name</th>
<th>Purpose</th>
<th>Wire color</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TD+</td>
<td>Transmit Data</td>
<td>White/Orange</td>
</tr>
<tr>
<td>2</td>
<td>TD+</td>
<td>Transmit Data</td>
<td>Orange</td>
</tr>
<tr>
<td>3</td>
<td>RD+</td>
<td>Received Data</td>
<td>White/Green</td>
</tr>
<tr>
<td>4</td>
<td>NC</td>
<td>No Connection</td>
<td>Blue</td>
</tr>
<tr>
<td>5</td>
<td>NC</td>
<td>No Connection</td>
<td>White/Blue</td>
</tr>
<tr>
<td>6</td>
<td>RD-</td>
<td>Received Data</td>
<td>Green</td>
</tr>
<tr>
<td>7</td>
<td>NC</td>
<td>No Connection</td>
<td>White/Brown</td>
</tr>
<tr>
<td>8</td>
<td>NC</td>
<td>No Connection</td>
<td>Brown</td>
</tr>
</tbody>
</table>

Table 2.1: RJ-45 Connector Signal
2.2.3 CAN versus Ethernet

CAN and Ethernet implementation in networked embedded systems depends on its requirement based on application. In an industrial automation or in an automobile assembly TCP/IP network configuration is not required, at the same time due to its size peer to peer nature makes CAN suitable for the same. In some other implementations where huge data storage and security, user friendly front end and remote operation with user intervention is required, obviously Ethernet based network will fulfil the requirement.

2.3. Software features

The software is the most important aspect of the embedded system, hardware perform the task as per software instruction. It is actually the brain of the system. An Embedded system processor and the system need software that is specific to a given application of that system. The processor of the system processes instructions coded and data. In the final stage these are placed in the memory (ROM) for all the tasks that have to be executed. Assembly as well as high level language like C, C++, Java etc are used for software development.

Challenging in designing and implementing embedded software comes from reliability, performance and cost. Reliability expectation brings greater responsibility to eliminate bugs and fault tolerant as many embedded system has to run 24 hours a day, a week and 365 days in a year. Sometime rebooting is not possible, so good programming and thorough testing is must for embedded software development. Performance issue may come from different considerations, such as proper multitasking and scheduling any considerably effect the performance. At the same time systems using sensors depends on how accurately sensor value is converted into real world value. Input/Output device may effect speed, complexity and cost. For better productivity sometime it may be needed to program directly in assembly in place of high level language. Embedded consumer products as produced in large so it is possible to keep in minimal production cost and no modification is performed once it start produced.

2.3.1 Creation of ROM image

In the final stage processed codes and instructions are placed in ROM which is called creation of ROM image. All executions of tasks are carried out from there. A brief description of creation of ROM image in assembly and High level language is described below.
There are different stages in converting an assembly language program into machine
implementable software file and then finally obtaining ROM image file. These steps are
explained with the following figure 2.5.

In the assembling step assembler translate assembly software into machine codes.
Next in linking phase linker links no of codes with other assembled codes. There are
certain codes having certain beginning address. Linking produces the final binary file by
linking all these. The linked file in a computer is commonly known as .exe file. In the
third phase reallocation of codes is done by placing it in physical memory by a program
called loader. Loader find out appropriate position in RAM that is ready to run. Finally
in locating phase ROM image is permanently placed in actually available address of ROM.
In embedded system since there is only one program so designer has to define the
available address to load and create files for permanent location. The locator locates the
I/O task and hardware device driver codes at unchanged address as port address of these
are fixed. In the last phase device programmer takes the ROM image and is burnt in to
the PROM or EPROM.

![Diagram of the process of converting assembly language program into ROM image](image)

Figure 2.5: Process of converting assembly language program into ROM image

In the conversion process of a high level language like C to ROM image file first
compiler generates the object codes. As per processor instruction compiler assemble the
codes and then code optimization is carried out by code optimizer [11]. Optimization is
carried out before linking. After compilation linker links codes including various standard
codes like printf, scanf and device driver codes. After linking subsequent steps for
creating ROM image is same as explained for assembly language.
Figure 2.6: Process of converting C Program into ROM image

A comparative view of build and load process of desktop and embedded application can be depicted with following figures 2.7 and 2.8

Figure 2.7: The build and load process for desktop application program.

Figure 2.8: The build and load process for embedded application program.
2.3.2 Software for embedded system device driver, multiple tasks, RTOS

There may be a number of physical devices attached with embedded systems. Device driver is the program needed to drive these devices. A driver uses hardware status flag and control register. It controls three functions (a) Initializing by placing appropriate bits at the control register. (b) Calling Interrupt service routine (ISR) for setting status flag (c) Resetting the status flag after interrupt service. Device driver coding is made using operating system functions such that underlying hardware is hidden. Device management software module provides codes for detecting the presence of devices. In designing the software for this category two types of devices are considered—Physical and Virtual. Physical devices include Keyboard, Printers, display matrix etc. Virtual device could be a file which may be used for reading and writing the stream of bytes. Operating system has modules for insertion of both device driver and device management module.

Sometime embedded systems have to control multiple devices for scheduling of multiple functions (task). To implement this embedded system must have a multitasking operating system above application level which is generally a Real Time Operating System (RTOS). In multitasking OS each process (task) has different memory allocation of its own and task has one or more than one procedures for a specific job [12]. A task may share memory (data) with other task. Processor may process different task separately or concurrently. An OS or RTOS has a kernel which is responsible for scheduling the transition of task from ready state to running state. Kernel may select a task for processing based on its priority value out of many ready state tasks. Calling ISR kernel may temporarily halt a running task and allow another task to run and resume the same after completion of new task.

An embedded system in multitasking environment always need not require an RTOS. An RTOS is required in a multitasking environment when real time constraints becomes must (i.e. task has to be completed in defined deadline). An RTOS main functions includes Real time task scheduling, Interrupt latency control, Time allocation and de-allocation to attain efficiency, predictable timing behaviour, priority management and time slicing of process management. RTOS may be of two types hard and soft real time. Hard real time strictly adhere task schedule whereas in soft real time precedence and sequence of task is defined.
2.3.3 Tools for designing embedded software

Different software tools for assembly language programming, high level language programming, RTOS, debugging and integrated tools can be summarized as given below.

**Editor:** It enables users to write codes for high level as well as assembly language in computer. Different features like addition, deletion, copy, insertion are made available for easy writing. It saves the content in a file with user defined or default extension. User can make necessary modification of saved files as and when required.

**Compiler:** It takes the input of whole high level source code and converts it to machine readable object code. It may include functions, library routines etc for compilation.

**Interpreter:** It converts high level codes to machine readable form line by line. Like compiler it may also include functions, library routines etc for conversion.

**Assembler:** It is used for conversion of assembly language programs to executable binary files. It creates the list file which has address, source code and hexadecimal object codes. It is processor specific.

**Cross assembler:** Cross assembler assembles the assembly code of target processor as assembly code of the processor of the PC used in the system development. Later it provides the object codes for the target processor. These will be the final codes used for the developed system.

**Simulator:** It is the program which can simulate all the functions of an embedded system circuit including additional memory and peripherals. It is independent of a particular target system.

**RTOS:** Explained in above chapter.

**Stethoscope:** This program is used to keep track of dynamic change in program variables and parameters. It can demonstrate the sequences of multiple processes, tasks, threads that execute and keeps entire time history.

**Trace scope:** It traces the change in module according to time. Accordingly list of actions to be initiated at desired time is also prepared.

**Integrated Development Environment (IDE):** Total software and hardware environment consist of simulator, compiler, assembler, cross assembler, logic analyzer EPROM/EEPROM, application codes, burners defines the integrated development environment of the system.
Locator: Locator program uses cross-assembler output and a memory allocation map and provides locator program output.

2.3.4 Use of Model in Embedded software design

In any software development process use of models make the process of software development easier as well as structured. This remains valid for embedded software development process also. In complex embedded software design which generally involves in multiprocessor systems, sometime it may not be possible to write directly the RTOS or codes for application programming. Use of different models in such a case may be easier for efficient software development. Different models used for this purpose are

(i) Finite state model: FSM are used for program models and analysis of real time programs. Real time system software implementation becomes greatly simplified when using the FSM programming tasks are reduced to the following steps. (a)Coding for each transition function and output function. (b) Knowing the time periods taken by the process at each state transition function and between the states, when programming for real time. FSM is appropriate for one process at a time for sequential flows from one state to the next state and for control flow of the program.

(ii) Petri net model: Petri net based modelling is one of the powerful tools for real time embedded systems. It is also a graphical modelling tool. Though FSM and Petri net both are very useful in coding ISR, FSM is used when there is a finite no of states and Petri nets can be used when no of states is not defined. FSM can be considered as a special case or sub class of Petri nets. Real time system software implementations are greatly simplified by use of Petri nets. When any of the several states can lead to any other state, it is impossible to know which one will definitely lead to which programming task are reduced. It perform the same by (a) Coding for each action at every transition (b) Knowing the time taken by the actions and processes at each node-transition function and the time between each fired transition.

(iii) Control Data Flow graph: A CDFG is basically a DFG where some control steps are inserted. The software designer programs and predetermined these steps. A CDFG graphically represents the conditions. It also represent the effect of events among the processes and shows which processes are activated on each specific event. Here a variable value changing above a limit or below a limit or falling within a range is like an event that activates a certain process. Software implementation becomes simplified when using the specification of the conditions and decision nodes in the CDFG s that represent the
controlled decision at the node and the program path (DFGs) that are traversed consequently from the nodes after the decision.

(iv) **Activity diagram based UML model**: Activity diagrams represent the business and operational workflows of a system. An Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state. UML (Unified Modelling Language) is an object oriented language used when (a) re-usability of defined object or set of objects that are common within a program or between the many applications. (b) When there is a need for abstraction and (c) When defining objects by inheritance, new objects can be created. There is a data encapsulation within an object. This model is found to be very much effective in object oriented software design.

Multiprocessor system computation and their firing instances can be modelled. Modelling simplifies the programming, scheduling and synchronising of processes. For multiprocessor system, the following additional models are required: (i) Synchronous Data Flow (SDF) graph (ii) Timed Petri-nets and Extended Predicate / Transition Net (iii) Multi Thread Graph (MTG) System. Multiprocessor system can be tightly or loosely coupled to the memory. It can also be coupled by mesh, ring, torrid or tree. There are different methods like SIMD, MIMD and VLW for scheduling and execution of instructions.

(i) **Synchronous Data Flow (SDF) graph**: SDFG model is like DFG but also models the delays as well as the number of inputs and outputs. When there is only one token at the input and one token at the output, an SDFG is a homogeneous SDFG (HSDFG).

(ii) **Timed Petri-nets and Extended Predicate / Transition Net**: Timed Petri-nets are Petri-nets with defined times for firing a transition and for computation times of the fired transition. Timed Extended Predicate / Transition Net are the timed Petri-nets with time distributed by probabilistic functions. These two types of nets model a multiprocessor system program flow.

(iii) **Multi Thread Graph (MTG) System**: MTG model of a program has two layers. One is CDFG and other timed Petri-nets for task level and processor level concurrent processing respectively. MTG provides optimal static and dynamic scheduling at the RTOS level.
2.3.5 Software Engineering practices in Embedded software design

Use of different software engineering tools or approaches is very much crucial in embedded software development [13]. An Embedded software generally differs from general computer application in (i) Different component of varied functionalities are found in embedded system software. (ii) Different component may have different response time still with the constraint of accomplishing the desired task in strict deadline. So optimum speed must be achieved by each component (iii) All software component must use memory optimally (iv) Software complexity must be controlled as well as thoroughly tested and debugged. Therefore in the process of software analysis, design, coding, debugging, Testing and validating use of appropriate software engineering tool is a must.

Activities during a software development are divided into three phases. Definition and Analysis of system requirements, development of designs, code and support. A software development process can be represented by a cycle called lifecycle. From the last stage, cycle can restart iteratively from first stage till verified and validated software is completed [14]. Refinement is done in successive iterations. There are several software-development process models, which can be grouped into following seven groups (1) Linear sequential lifecycle model (2) RAD (Rapid Development) model. (3) Incremental development Model (4) Concurrent Development Model (5) Component Based Model (6) Fourth Generation tool based development Model and (7) Object Oriented Development Model [15].

In the first phase of analysis of requirements specifications are obtained for data elements and structures, algorithms, behaviour, data contents and process specifications. This is followed by design phase. Design can be assumed to be consisted of four layers. Architecture design, data design, Interface design and component level design. Continuous refinement in design by effective communication between designers and implementers is necessary. In testing phase as embedded software need to be of highest quality and therefore it needs systematic approach in testing and it should be followed by debugging means finding the reason for fault. Testing approaches are white-box testing, black-box testing, specific requirement testing, comparison testing and testing by orthogonal data inputs. Test verifies that software developed after processing through the lifecycle stages is as per required and agreed specifications [16]. Validation of required specification is done at the last stage before delivering to customer. Then it will have to go through successive refinement model. Observed error source is traced during debugging by the use of breakpoints, testing macros, test vectors, simulators and laboratory tools. Figure 2.9 depicts all these activities in embedded software development process.
2.4 Software implemented in a Networked embedded system

Different adapters can be used to develop and support CAN based applications. All these adapters are software compatible with each other, so software developed for one adapter works with the other adapters also.

Networking in other case can be implemented based on Client Server Technology in contrast to peer to peer of CAN. Client Server communication is based on TCP/IP used popularly in LAN as well as WEB communications.

2.4.1 Implementing client server concept

As already explained, in CAN physical and Data ink Layers are Standardized by ISO. Its physical layer is like RS2332 and supports many different higher layer software protocols (such as CAN Open, DeviceNet and J1939). Higher layer / Application layer is totally application specific and different manufacturers develop it as per requirement of the
application. CAN is also a high performance serial line that can go up to 1M baud and is widely used in the automotive, medical and automation industries. CAN adapters and software tools want to communicate over network and is based on mechanism provided by Transmission Control Protocol (TCP). These TCP connections are established and closed as and when required between client and server. To implement the same TCP software on the server computer execute a process called as passive open [17]. This indicate that the server computer expects one or more clients to request it for establishing TCP connection for which it(server) waits endlessly. This is why it is called passive open. Passive open allows to establish connection on request but not sending in outgoing TCP connection. In contrast client always initiate TCP connection by sending request to server which is said to be active open. The process is explained in figure 2.10. A client process initiates a TCP connection by performing an active OPEN, sending a SYN message to a server. A server process using TCP prepares for an incoming connection request by performing a passive OPEN. Both devices create for each TCP session a data structure used to hold important data related to the connection, called a transmission control block (TCB).

The normal process of establishing a connection between a TCP client and server involves three steps: the client sends a SYN message; the server sends a message that combines an ACK for the client’s SYN and contains the server’s SYN; and then the client sends an ACK for the server’s SYN. This is called the TCP three-way handshake. Following diagram illustrates how a conventional connection is established between a client and server, showing the three messages sent during the process and how each device transits from the CLOSED state through intermediate states until the session is ESTABLISHED.
The first consequence of this is that each connection must be uniquely identified. This is done by using the pair of socket identifiers corresponding to the two endpoints of the connection, where a socket is simply the combination of the IP address and the port number of each process. This means a socket pair contains four pieces of information: source address, source port, destination address and destination port. Thus, TCP connections are sometimes said to be described by this addressing quadruple.

After establishment of connection client can perform the desired operations on server like accessing file etc. It can be explained as start of data transmission as given in figure 2.11.

A TCP connection is non-persistent. In simple terms this means after the connection establishment of Client server as per client request, server oblige. After sending response server closes the connection. If client wants to make connection again it has to go for a new
connection establishment. The session doesn’t persist beyond the lifetime of one request and one response, hence it is non-persistent. There are situations when non-persistent connections are not desirable. Say on Internet one page may contain text, image, audio and video. These content of same webpage may scattered in different servers or in different files of the same server. Client in such a situation may have to go for multiple TCP connections for obtaining the whole web page. This is shown in the figure 2.12.

![Multiple TCP connections](image)

**Figure 2.12 : Multiple TCP connections**

To overcome this problem some protocol say new version of HTTP allows persistent connection. Here after serving the request server doesn’t close the connection. Instead it keeps it open enabling client to reuse the same connection.

### 2.4.2 Implementing web enable features

Making network operated system web enabled the requirement is to induct a Web server for providing web services. A web server is a program running on a server computer. Additionally it consist of web sites containing a number of web pages. A web server constantly and passively waits for a request from browser program running on client. After receiving such a request it locates the page and sends it back to the client. To implement this, every website has a server process that listens to TCP connection request coming from different client. After sending response server releases the connection. This request response model is governed by a protocol called HTTP(Hyper Text Transfer Protocol). HTTP software on client prepares the request for a webpage, on the server side it interprets such a request and prepares response to sent back to client. Thus HTTP is must for both client and server.

A web browser works as a client during this interaction. Using this program user’s request for a web page is stored on a web server. The Web server locate this web page
which may be in a different server say a database server and sends back to the client computer. The Web browser interprets the web page written in HTML and displays it on the client's computer. This interaction is depicted in figure 4.7.
References


