Embedded Systems

UNIT-VI

Simulator: Simulators simulate the target hardware and the firmware execution can be inspected using simulators. This is a software tool used for simulating the various conditions for checking the functionality of the application firmware.

The features of simulator based debugging are listed below.

1. Purely software based
2. Doesn't require a real target system
3. Very primitive [lack of featured I/O supported]
4. Lack of Real-time behaviour

✓ **Advantages of Simulator Based Debugging**

Simulator based debugging techniques are simple and straightforward. The major advantages are

* No need for original target board
* Simulate I/O peripherals
* Simulates Abnormal conditions

No need for original target board:

This is purely software oriented. IDE's software support simulates the CPU of the target board. User only needs to know about the memory map of various devices within the target board and the firmware should be written on the basis of it. Since the real hardware is not required, firmware development can start well in advance immediately after the device interface and memory maps are finalized. This saves the development time.

Scanned with CamScanner

Simulate I/O peripherals: simulator provide option to simulate various I/O peripherals. In simulator's I/O support you can edit the values for I/O registers and can be used as the input/output value in the firmware execution there it eliminates the need for connecting I/O devices for debugging the firmware.

Simulates Abnormal conditions: with simulator's simulation support you can input any desired value for any parameter during debugging the firmware and can observe the control flow of firmware. It really helps the developer in simulating abnormal environment for firmware and helps the firmware developer to study the behaviour of the firmware under abnormal input condition

Limitations of Simulator based debugging

Even simulation based firmware debugging technique is very helpful in embedded applications they possess certain limitations and we can't fully rely upon the simulator-based firmware debugging. Some of the limitations are

* Deviation from Real Behaviour
* Lack of real timeliness

Deviation from Real Behaviour: simulation-based firmware debugging is always carried out in a development environment where the developer may not be able to debug the firmware under all possible combinations of input. Under certain operating conditions we may get some particular result and it need not be the same when the firmware runs in a production environment.

Scanned with CamScanner

Lack of timeliness: The major limitation of simulator based debugging is that it is not real-time in behaviour. The debugging is developer driven and it is no way capable of creating a real time behaviour. Moreover in a real application the I/O condition may be varying or unpredicta-ble.

✓ Emulators and Debuggers

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 from various buses of the embedded hardware. Debugging process in embedded application is broadly classified into two, namely, hardware debugging and firmware debugging.

Hardware debugging deals with the monit-oring of various bus signals and checking the status lines of the target hardware.

Firmware debugging deals with examining the firmware execution, execution flow, chan-ges to various CPU registers and status register on execution of the firmware to ensure that the firmware is running as per the design.

The most primitive type of debugging are

* Incremental EEPROM Burning Technique
* Inline Breakpoint Based Firmware Debugging
* Monitor Program Based Firmware Debugging
* In circuit Emulator (ICE) Based Firmware Debugging
* On Chip Firmware Debugging (OCD)

Scanned with CamScanner

Incremental EEPROM Burning Technique:

This is the most primitive type of firmware debugging technique where code is separated into different functional code units. Instead of burning the entire code into the EEPROM chip at once, the code is burned in incremental order, where the code corresponding to all functionalities are separately coded, cross-compiled and burned into the chip one by one.

If the first functionality is found working perfectly on the target board with the corresponding code burned into the EEPROM, go for burning the code corresponding to the next functionality and check whether it is working. Repeat this process till all functionalities are covered.

If the code corresponding to any functionality is found not giving the expected result, fix it by modifying the code and then only go for adding the next functionality for burning into the EEPROM. After you found all functionalities working properly, combine the entire source for all functionalities together, re-compile and burn the code for the total system functioning.

Obviously it is a time-consuming process. But remember it is a one time process and once you test the firmware in an incremental mode you can go for mass production.

Inline Breakpoint Based Firmware Debugging:

Inline breakpoint based debugging is another primitive method of firmware debugging. Within the firmware where you want to ensure that firmware execution is reaching up to a specific point, insert an inline debug code immediately after the point. Cross-compile the source code with the debug codes embedded within it. Burn the corresponding hex file into EEPROM.

Scanned with CamScanner

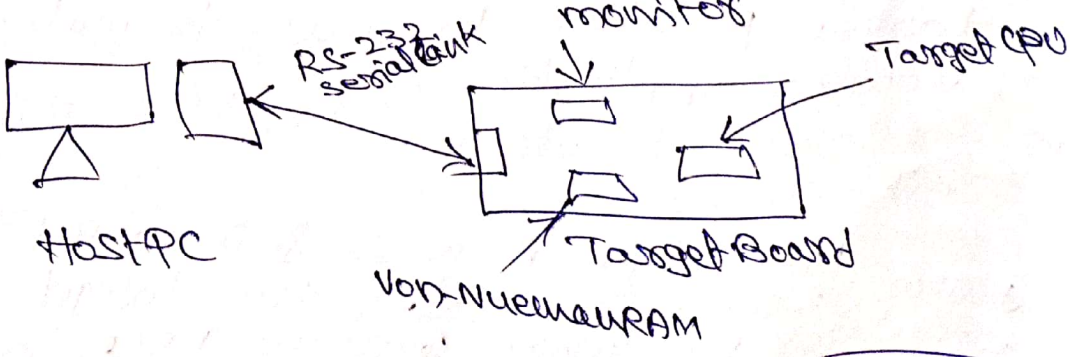
Embedded Systems

UNIT - VI (3)

Monitor Program Based Firmware Debugging:

This is a invasive method for firmware debugging. The monitor program which acts as a supervisor is developed. This program controls the downloading of usercode into the code memory, inspects and modified registers/memory locations; allows single stepping of source code. The monitor program implements the debug functions as per a pre-defined command set from the debug application interface. The monitor program always listens to the serial port of the target device and according to the command received from the serial interface it performs command specific actions like firmware downloading, memory inspection/modification, firmware single stepping and sends the debug information back to the main debug program running on the development PC. The first step in any monitor program development is determining a set of commands for performing various operations like firmware downloading, memory/register inspection/modification, single stepping. These may be received through any of the external interface of the target processor.

The entire code stuff handling the command reception and corresponding action implementation is known as the "monitor program".



Scanned with CamScanner

The code memory containing the program is known as the 'monitor ROM'.

- The monitor program contains the following of minimal features.

① Command set interface to establish communication with the debugging application

② Firmware download option to code memory

③ Examine and modify processor registers and working memory (RAM)

④ Single step program execution.

⑤ Set breakpoints in the firmware execution

⑥ Send debug information to debug application running on host machine.

- The major drawback of monitor based debugging system are:

① The entire memory map is converted into a von-Neumann model and is shared b/w the monitor ROM, monitor program data memory, monitor program trace buffer, user written firmware and external user memory. The memory shrink may be the result.

② The communication link b/w the debug application running on development pc and monitor program residing in the target system is achieved through a serial link and usually the controller's on-chip UART is used for establishing this link. Hence one serial port of the target processor becomes dedicated for the monitor application.

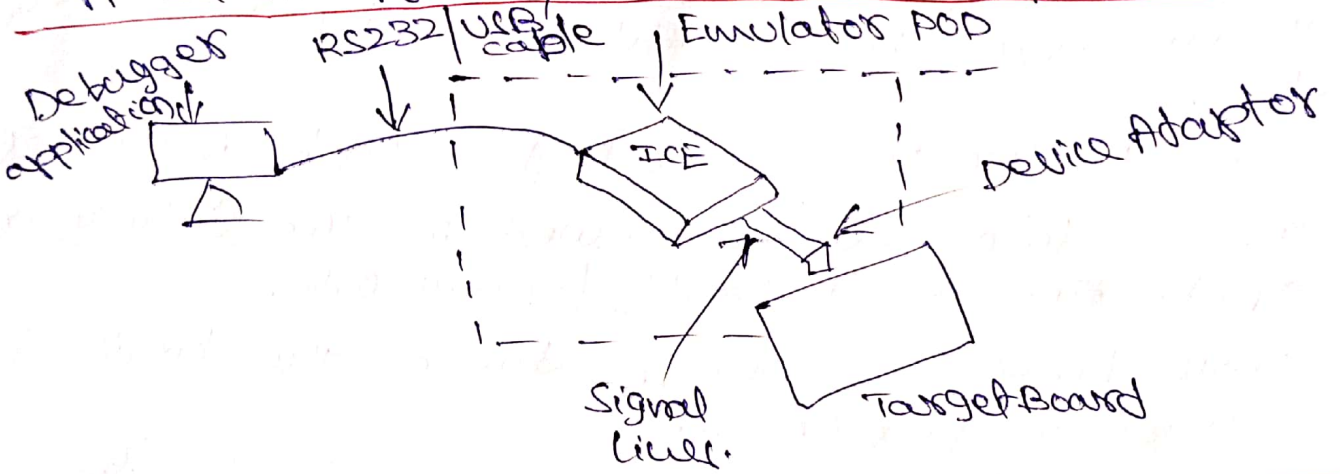
- **In Circuit Emulator (ICE) Based firmware Debugging:**

Emulator is a self-contained hardware device which emulates the target CPU. The emulator hardware contains necessary emulation

Scanned with CamScanner

logic and it is hooked to debugging application running on the development PC on one end and connects to the target board through some interface on the other end.

In older days emulators were defined as special hardware devices used for emulating the functionality of a processor/controller and performing various debug operations like halt firmware execution, set breakpoints, get or set internal RAM/CPU registers, etc. A hardware emulator is controlled by a debugger application running on the development PC.



In circuit Emulator (ICE) Based Target Debugging

The emulator POD forms the heart of any emulator System and it contains the following functional units.

* Emulation Device
* Emulation Memory
* Emulator Control logic
* Device Adaptor

Emulation Device: is a replica of the target CPU which receives signals from the target board through a device adaptor connected to the target board and performs the execution of firmware

PTO (4)

Scanned with CamScanner

of the firmware under the control of debug from the debug application. The emulation device be either a standard chip same as the target processor or a programmable Logic Device (PLD) -figured to function as the target CPU.

Emulation Memory: It is the RAM incorporated in the Emulator device. It acts as a replacement to the target board's EEPROM where the code is supposed to be downloaded after each firmware modification. Here the original EEPROM memory is emulated by the RAM of emulator.

Emulation memory also acts as a trace buffer in debugging.

The common features of trace buffer memory and trace buffer data viewing are listed below.

* Trace buffer records each bus cycle in frames
* Trace data can be viewed in the debugger application as Assembly / source code.
* Trace buffering can be done on the basis of a Trace trigger (Event)
* Trace buffer can also record signals from target board other than CPU signals
* Trace data is very useful information in firmware debugging.

Emulator Control Logic: This is the logic circuit used for implementing complex hardware break points, trace buffer tigger detection, trace buffer control. These can also implement logic analyser functions in advanced emulator devices.

Scanned with CamScanner

Embedded Systems

UNIT - VI (5)

Device Adaptor: This acts as an interface b/w the target board and emulator pod. Device adaptors are normally pin-to-pin compatible sockets which can be inserted/plugged into the target board for routing the various signals from the pins assigned for the target processor. The device adaptor is usually connected to the emulator POD using ribbon cable.

✓ **On Chip Firmware Debugging (OCD)**

Advances in semiconductor technology has brought out new dimensions to the target firmware debugging. Today almost all processors/controllers incorporate built in debug modules called on chip Debug (OCD) support. This technique supporting fast and efficient firmware debugging. An on chip Debugger can be enabled by setting the OCD enable bit.

✓ **Embedded product Development Life Cycle (EDLC)**

This an 'Analysis - Design - Implementation' based standard problem solving approach for Embedded Product Development.

EDLC defines the interation and activities among various groups of a product development sector including project management, System Design and development (hardware, firmware and enclosure design and development), System testing, release management and quality assurance.

✓ **The primary objectives of EDLC**

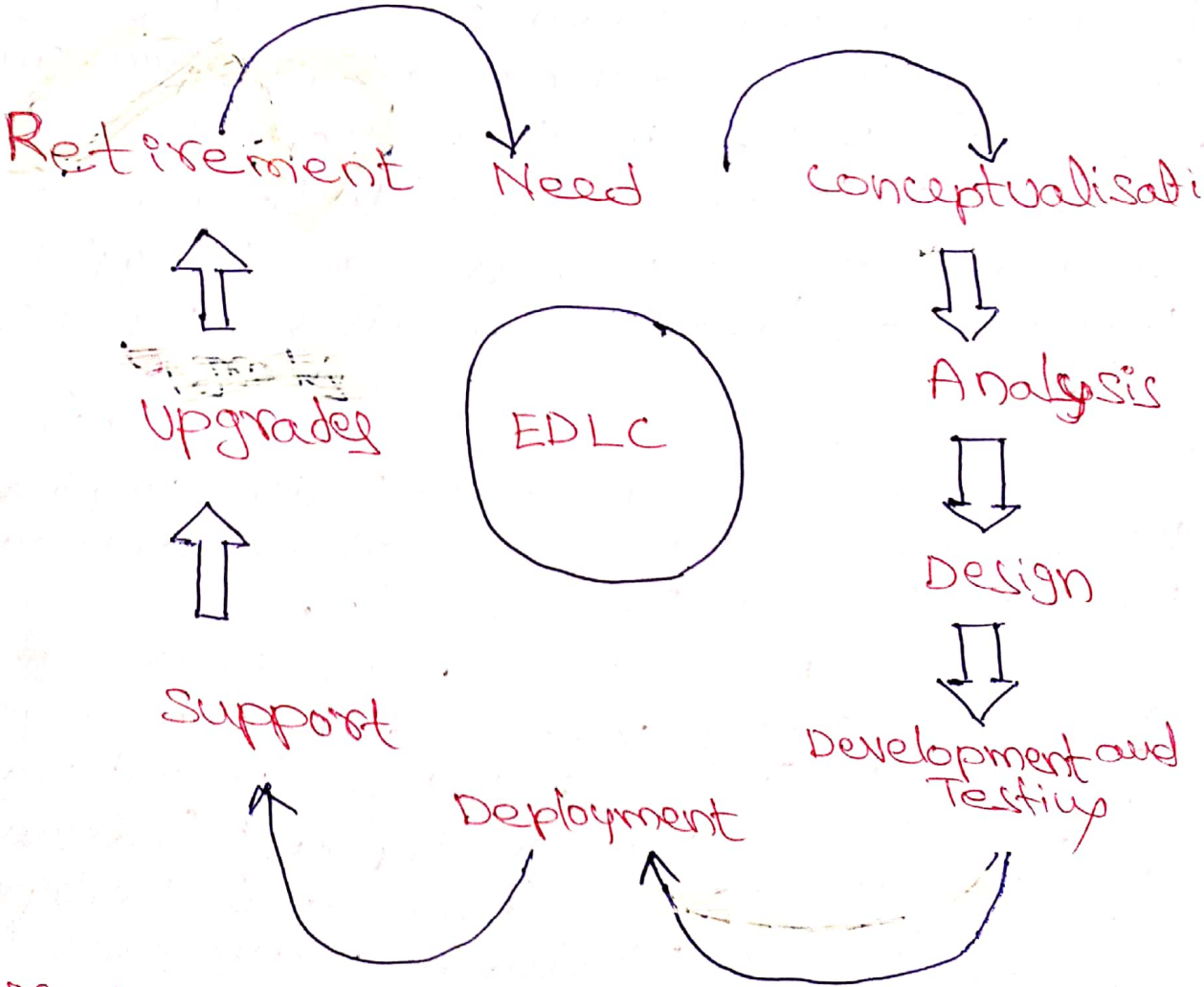
* Ensure that high quality products are delivered to the end user.

Scanned with CamScanner

* Risk minimisation and defect prevention in development through project management.
* Maximise the productivity.

✓ Different phases of EDLC

The life cycle of a product development is commonly referred to as models. The model defines the various phases involved in the life cycle. The classic Embedded Product Life Cycle Model contains the phases: 'Need', 'Conceptualisation', 'Analysis', 'Design', 'Development and Testing', 'Deployment', 'Support', 'upgrades' and 'Retirement / Disposal'.



Classic Embedded Product Development Life Cycle Model.

Scanned with CamScanner

Need: The need may come from an individual or from the public or from a company. Based the need for the product, a 'Statement of Need' or 'concep proposal' is prepared. And it must be approved by the concerned people. Once the proposal gets approval, it goes to a product development team. The product deve-lopment need can be visualised in any one of the following three needs.

* New or custom Product Development
* Product Re-engineering
* Product maintenance.

Conceptualisation: This is the 'product Developme-nt' phase. and it begins immediately after a Concept proposal is formally approved. This Phase defines the scope of the concept, performs cost benefit analysis and feasibility study and prepares project management and risk manage-ment plans. This phase performs the impor-tant 'Analysis and study activities'.

* Feasibility study
* Cost Benefit Analysis
* product scope
* planning Activities

Analysis: Requirements Analysis phase starts imm-ediately after the documents submitted during the 'conceptualisation' phase is approved by the client/sponsor of the project. Requirement analysis is performed to develop a detailed functio-al model of the product under consideration.

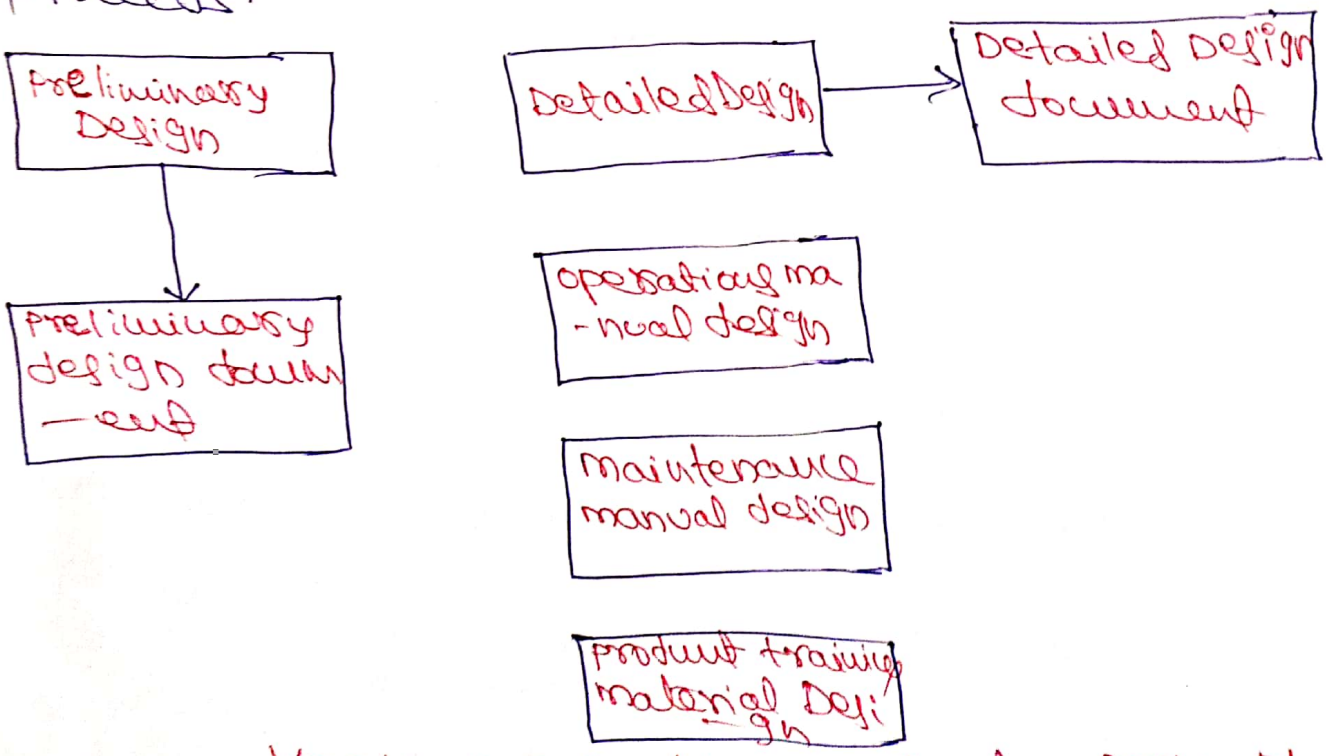
PTO (6)

Scanned with CamScanner

During this phase, a product is defined with respect to the inputs, processes, outputs, interfaces at a functional level. This phase emphasis on determining 'what' functions must be performed by the product' rather than 'how' to perform those functions. The following activities are performed during this phase.

* Analysis and Documentation
* Interface Definition and Documentation.
* Defining test plan and procedures

Design: This phase deals with the entire design of the product taking the requirements into consideration and it focuses on 'how' the required functionalities can be delivered to the product. The design phase identifies the application environment and creates an overall architecture for the product. The design always starts with 'preliminary' Design / High Level Design'. Further the detailed design proceeds.



Various Activities involved in Design Phase

Scanned with CamScanner

UNIT - 6 (7)

Development and Testing: This phase transforms the design into a realizable product. During development phase, the installation and setting up of various development tools is performed and the product hardware and firmware is developed using different tools and associated production setup. The activities can be partitioned into embedded hardware development, embedded firmware development and product enclosure development.

The testing phase can be divided into independent testing of firmware and hardware, testing of the product after integrating the firmware and hardware, testing of the whole system on a functionality and non-functionality basis and testing of the product against all acceptance criteria mentioned by the client/end user for each functionality.

Deployment: This is the process of launching the first fully functional model of the product in the market. (or) handing over the fully functional initial model to an end user/client. It is known as 'First Customer Shipping'. The deployment phase is initiated after the system is tested and accepted by the end user. The important tasks performed during this phase are.

* Notification of product Deployment
* Execution of Training plan
* Product Installation
* Product Post-Implementation Review.

Scanned with CamScanner

Support: This deals with the operations and maintenance of the product in a production environment. During this phase all aspects of operations and maintenance of the product are covered and the product is scrutinised to ensure that it meets the requirements put forward by the end user. Support should be provided to the end user to fix the bugs in the product. The various activities are

* Set up a Dedicated Support Wing
* Identify Bugs and Areas of Improvement.

Upgrades: The upgrade phase of product development 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 or feature enhancement requirements from the end user. During this phase the system is subject to design modification to fix the major bugs reported or to incorporate the new feature addition requirements aroused during the support phase. Here the upgradation can be for both hardware and firmware.

Retirement / Disposal: The disposal/retirement of a product is a gradual process. This is the last phase of product development life cycle where the product is declared as obsolete and discontinued from the market. This may be, because of

① Rapid technology advancement
② Increased user needs.

Scanned with CamScanner

Embedded Systems

UNIT-VI (8)

Trends in the Embedded Industry

There are continuous upgrades in various areas of embedded system. So that the trends in the embedded Industry are continuous process. Those are

* Processor trends in embedded system
* Embedded OS trends
* Development Language trends
* Open standards, Framework and Alliances.

Integration and Testing of Embedded Hardware and Firmware

Integration of Hardware & Firmware:

Integration of hardware and firmware deals with the embedding of firmware into the target hardware board. It is the process of 'Embedding Intelligence' to the product. The embedded processor/controllers used in the target board may or may not have built in code memory. For non-operating system based embedded product, if the processor/controller does not support built in code memory or the size of the firmware is exceeding the memory size supported by the target processor/controller, an external dedicated EPROM/FLASH memory chip is used for holding the firmware.

The firmware embedding into the target board can be done using various ways.

Scanned with CamScanner

* Out-of-circuit Programming
* In-System Programming
* In-Application Programming
* Use of factory programmed chip.

Scanned with CamScanner

Embedded Systems

UNIT - 6 (9)

Introduction to ARM family of processors.

The ARM architecture has evolved a lot from its first version ARM1 to the latest ARM11 processor core. ARM1, ARM2 .... ARM11 are the product families from ARM since its introduction to the market.

ARM is a RISC processor and it supports multiple levels of pipelines for instruction execution. It contains 37, 32-bit registers. Out of the 37 registers 30 are general purpose registers and 16 general purpose registers [r0 to r15] are available in the user mode of operation. Register r15 is program counter. The current Program Status Register (CPSR) holds execution status of the processor, processor operation mode, interrupt enable bit status, etc.

ARM supports the following operating modes.

Usermode: It is the main execution mode for user applications. It enables the protection and isolation of operating system from user applications.

Fast Interrupt Processing (FIQ) mode: The processor enters this mode when high priority interrupt is raised.

Normal Interrupt Processing (IRQ) mode: The processor enters this mode when a normal priority interrupt is raised.

Supervisor/Software Interrupt mode: The processor enters this mode on reset and when a software interrupt instruction is executed.

PTO(9)

Scanned with CamScanner

Abort Mode: Enters this mode when a memory access violation occurs.

Undefined Instruction mode: Enters this mode when the processor tries to execute an undefined instruction.

System mode: This mode is used for running operating system tasks. It uses the same register as the User mode.

The Instruction Set Architecture (ISA) of ARM supports three different types of Instruction sets, namely.

ARM Instruction Set: Here all instructions are 32-bit wide and are word aligned. Since all instructions are word aligned, one single fetch reads four 8-bit memory locations.

Thumb Instruction set: Thumb is the 16-bit subset for the 32-bit ARM instructions. These instructions can be considered of a 16-bit compressed form of the original 32-bit ARM instructions.

Jazelle Instruction Set: This is the hardware implementation of the Java Virtual Machine (JVM) for executing java byte codes.

The ARM Instruction set can be broadly classified into:

* Data processing Instructions
* Program control Instructions
* Multiplication Instructions
* Barrel Shift operations
* Branching Instruction
* Co-Processor Specific Instructions

Scanned with CamScanner