Chapter 8

Hardware / Software Co-design on Prototype

Professor Ching-Lung Su
E-mail: kevinsu@yuntech.edu.tw
Http://soc.eecs.yuntech.edu.tw
NYUST/EL
Outline

8.1 Hardware / Software Co-design on Prototype
8.2 Test and Debug on Prototype
8.3 Introduction to GDB Server
8.4 Reference
8.1 Hardware / Software Co-design on Prototype

8.2 Test and Debug on Prototype
8.3 Introduction to GDB Server
8.4 Reference
8.1 Hardware / Software Co-design on Prototype

◆ Software and Hardware Partition
◆ Procedure Structure of the Software
◆ Systematic Structure of the Hardware
◆ Integrate Hardware and Software
8.1 Software and Hardware Partition

- Separate the partitions of Hardware and Software
  - Analysis the complexity of MPEG-2 software
  - Determine the cut of the hardware and software
  - Software partition:
    - Operation movements are dispersed in different function.
    - Operation amount is small
  - Hardware partition:
    - Operation movements are suitable for accelerating regularly with the hardware.
    - Operation amount is great
8.1 Software and Hardware Partition

- **Design Flow of SW & HW Partition**
  - Analysis the source code to decide the partitions of Hardware and Software.
  - Integrate Hardware and software.

```
Design Flow of SW & HW Partition

1. Reference source
2. Profiling analysis
3. HW and SW partition
4. Software Partition
5. Hardware Partition
6. Integrate HW and SW
7. Debug
```
Architecture of the System

Software

![Diagram of software components]

- CPU
- Storage Device
- Peripheral

Hardware

- FPGA
- I/O device

Bus interface
8.1 Software and Hardware Partition

**Architecture of the System**

Software: CPU

- Software: Application code
- OS kernel
- Software: Driver

Hardware: FPGA

- Hardware: The design of hardware IP

Bus interface:
Real signal of the bus

Bus interface
8.1 Software and Hardware Partition

**MPEG-2 Decoder Flow**

- Regard “MPEG-2 decoder” as the example.
8.1 Software and Hardware Partition

- MPEG-2 Decoder Profiling

- Test sequence: Foreman sequence on CIF (352x288)

![Diagram showing software and hardware partition]

```
Test sequence:
Foreman sequence on CIF (352x288)
```

```
Use hardware
```

```
IDCT
Motion Compensate
VLC
Store Frame
```

```
8.1 Software and Hardware Partition
```

```
P-10/115
教育部顧問室 PAL聯盟/系統雛型與軟硬體整合設計
```

```
第八章：系統雛型軟硬體設計實現
```

```
教育部顧問室 PAL聯盟/系統雛型與軟硬體整合設計
```
8.1 Software and Hardware Partition

Software and Hardware Partition

Diagram:

- $F_{n-1}$ (reference)
- $F_n$ (decoded frame)
- Motion Compensate
- IDCT
- Rescale
- Recorder
- Entropy decode
- Vectors and headers
- Coded Bistream

Hardware

Software
8.1 Procedure Structure of the Software

- MPEG-2 Decoder Flow

Diagram:

- $F_{n-1}$ (reference)
- $F_n$ (decoded frame)
- Motion Compensate
- IDCT
- Rescale
- Entropy decode
- Vectors and headers
- Recorder
- Coded Bitstream

Files:
- recon.c
- idct.c
- getblk.c
- getvlc.c
8.1 Procedure Structure of the Software

**MPEG-2 Decoder Flow**

- getpicture → getMBs
- Intra block
  - Inter block
- startcode
  - getpid
  - getMB2intra
- showbits
  - getbits
- getMB2inter
  - VLC decode
  - recon
  - recon_comp
  - Motion Compensation
  - idct
  - idctrow
  - Transform
  - idetcol
  - addblock
  - storeframe
8.1 Procedure Structure of the Software

MPEG-2 Decoder Source

Software

- Entropy decode
- Motion Compensation
- Block count == 6?
- Next Micro-Block

Hardware

- IDCT
- addblock
- YUV to RGB

NO

YES

One Frame End?
8.1 Procedure Structure of the Software

RGB 565 Format & System Memory Map

**RGB 565 format**

RGB 565 frame buffer format

```
R R R R R G G G G G G G G G B B B B B B
```

**System memory map**

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>IQ Data</td>
</tr>
<tr>
<td>0x0FF</td>
<td>MC Data</td>
</tr>
<tr>
<td>0x100</td>
<td>YUV Data</td>
</tr>
<tr>
<td>0x13F</td>
<td>RGB Data</td>
</tr>
<tr>
<td>0x140</td>
<td>Controller flag reg</td>
</tr>
<tr>
<td>0x17F</td>
<td></td>
</tr>
<tr>
<td>0x180</td>
<td></td>
</tr>
<tr>
<td>0x1FF</td>
<td></td>
</tr>
<tr>
<td>0x200</td>
<td></td>
</tr>
</tbody>
</table>


8.1 Procedure Structure of the Software

**HW Controller Flag Register (0x200)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| ... | T | M | S2 | S1 | S0 | C2 | C1 | C0 | B2 | B1 | B0 | A2 | A1 | A0 |

- **A2-A0**: IP run block number A, RGB IP Block count
- **B2-B0**: IP run block number B, MC IP Block count
- **C2-C0**: IP run block number C, IDCT IP Block count
- **S2-S0**: IP status
  - 1: IP ready, 0: IP work, not ready
  - S2:IDCT, S1:MC, S0:RGB
- **M**: SW send MC data status
  - 1: SW send to MC IP
  - the step of the value to enable MC IP work: 0 → 1 → 0
- **T**: MB Block Type
  - 0: Intra type, 1: Inter type
8.1 Systematic Structure of the Hardware

MPEG-2 HW System

- **Address decoder**
  - Controller flag addr.
  - IQ data addr.
  - MC data addr.
  - YUV addr.
  - RGBout data addr.

- **System controller**
  - (Controller flag reg)

- **IDCT**
- **MC Add**
- **RGB**

- **Data output decoder**
  - Data (residual + MC)
  - Data (RGB)
  - Controller flag reg.
8.1 Systematic Structure of the Hardware

- MPEG-2 HW System

![Diagram of MPEG-2 HW System]

- System controller (controller flag reg.)
- IDCT
  - IQ data in
  - IDCT data
  - SW MC data in
- MC Add
  - Run block num
  - Run block num
- RGB
  - RGB run
  - YUV data
  - RGB data to SW
  - YUV data to SW
8.1 Integrate Hardware and Software

Software and Hardware

Software (getpic.c)
- getmpg2intrablock
- getmpg2interblock

- reconstruct

- IQ Data

- MC Data

- Next Block (UVYYYY)

Hardware
- IDCT

- MC add

- YUV to RGB

- Next MB
8.1 Integrate Hardware and Software

- The Step of the Flow

Step 1. Software procedure:
- Get the type of the MB (Intra type or Inter type)
- Get IQ data and MC data
- Send IQ data and MC data into Hardware

Step 2.
- Check the HW Controller Flag Register (0x200)
- Check whether the bit11(S2) is high or not
- Judge the output (IDCT-Data) of IDCT-IP is ready or not.

Step 3.
- The output (IDCT-Data) of IDCT-IP is ready
- Set the value into bit12 and bit13 of HW Controller Flag Register (0x200)
- Bit12: the step of the value to enable MC-IP work: 0 → 1 → 0
- Bit13: set the type of the MB: 0-intra type, 1-inter type
8.1 Integrate Hardware and Software

■ The Step of the Flow

Step 4.

■ Check the HW Controller Flag Register (0x200)
■ Check whether the bit10(S1) is high or not
■ IF it is high, software read the YUV data from HW system

Step 5.

■ When decode the MB, the order is U→V→Y→Y→Y→Y
■ When decode Y block, check the bit9(RGB-IP status) and bit20(RGB block count) of the HW Controller Flag Register (0x200)
■ IF RGB-IP is ready, software read RGB data from Hardware

Step 6.

■ Send RGB data to Frame Buffer
■ Display the picture
8.2 Test and Debug on Prototype

8.1 Hardware / Software Co-design on Prototype
8.2 Test and Debug on Prototype
8.3 Introduction to GDB Server
8.4 Reference
8.2 Test and Debug on Prototype

- ARM Developer Suit
- Quartus II
- Logic Analyzer
8.2 Test and Debug on Prototype

- It is hard to debug on Embedded System, mainly divide three parts:
  - Software: ARM Developer Suit (ADS)
    - AXD Debugger
  - Hardware: Quartus II
  - Bus: Logic Analyzer
8.2 ARM Developer Suit

◆ About the ARM Developer Suite

- ADS consists of a suite of applications, together with supporting documentation and examples, that enable you to write and debug applications for the ARM family of RISC processors.
- You can use ADS to develop, build, and debug C, C++, or ARM assembly language programs.
8.2 ARM Developer Suit

◆ System Architecture of ADS

- ARM Assembly
- C/C++ Language
- Compiler
- Object File
- Object File
- Link
- C/C++ Library
- ADS Compiler
- File
- Binary
8.2 ARM Developer Suit

◆ Components of ADS

■ ADS Consists of The Following Major Components
  ● Command-line Development Tools
  ● GUI Development Tools
  ● Utilities
  ● Supporting Software
8.2 ARM Developer Suit

◆ Command-line Development Tools

- armcc
  - The ARM C compiler. The compiler is tested against the Plum Hall C Validation Suite for ANSI conformance. It compiles ANSI source into 32-bit ARM code.

- armcpp
  - This is the ARM C++ compiler. It compiles ISO C++ or EC++ source into 32-bit ARM code.
8.2 ARM Developer Suit

- **tcc**
  - The Thumb C compiler. The compiler is tested against the Plum Hall C Validation Suite for ANSI conformance. It compiles ANSI source into 16-bit Thumb code.

- **tcpp**
  - This is the Thumb C++ compiler. It compiles ISO C++ or EC++ source into 16-bit Thumb code.

- **armasm**
  - The ARM and Thumb assembler. This assembles both ARM assembly language and Thumb assembly language source.
8.2 ARM Developer Suit

■ armlink

- The ARM linker. This combines the contents of one or more object files with selected parts of one or more object libraries to produce an executable program. The ARM linker creates ELF executable images.

■ armsd

- The ARM and Thumb symbolic debugger. This enables source level debugging of programs. You can single-step through C or assembly language source, set breakpoints and watchpoints, and examine program variables or memory.
8.2 ARM Developer Suite

- Rogue Wave C++ library
  - The Rogue Wave library provides an implementation of the standard C++ library as defined in the ISO/IEC 14822:1998 International Standard for C++. For more information on the Rogue Wave library.

- support libraries
  - The ARM C libraries provide additional components to enable support for C++ and to compile code for different architectures and processors.
8.2 ARM Developer Suit

Components of ADS

- ADS Consists of The Following Major Components
  - Command-line Development Tools
  - GUI Development Tools
  - Utilities
  - Supporting Software
8.2 ARM Developer Suite

- GUI Development Tools
  - AXD
    - The ARM Debugger for Windows and UNIX. This provides a full Windows and UNIX environment for debugging your C, C++, and assembly language source.
  - CodeWarrior IDE
    - The project management tool for Windows. This automates the routine operations of managing source files and building your software development projects. The CodeWarrior IDE is not available for UNIX.
Components of ADS

ADS Consists of The Following Major Components

- Command-line Development Tools
- GUI Development Tools
- Utilities
- Supporting Software
8.2 ARM Developer Suit

- **Utilities**
  - **fromELF**
    - The ARM image conversion utility. This accepts ELF format input files and converts them to a variety of output formats, including plain binary, Motorola 32-bit S-record format, Intel Hex 32 format, and Verilog-like hex format. `fromELF` can also generate text information about the input image, such as code and data size.
  - **armprof**
    - The ARM profiler displays an execution profile of a simple program from a profile data file generated by an ARM debugger.
8.2 ARM Developer Suit

- armar
  - The ARM librarian enables sets of ELF format object files to be collected together and maintained in libraries. You can pass such a library to the linker in place of several ELF files.

- Flash downloader
  - Utility for downloading binary images to Flash memory on an ARM Integrator?board or an ARM Development board (PID7T).
8.2 ARM Developer Suit

◆ Components of ADS

- ADS Consists of The Following Major Components
  - Command-line Development Tools
  - GUI Development Tools
  - Utilities
  - Supporting Software
8.2 ARM Developer Suit

◆ Supporting Software

- The following support software is provided to enable you to debug your programs, either under simulation, or on ARM-based hardware:
  - ARMulator
    The ARM core simulator. This provides instruction accurate simulation of ARM processors, and enables ARM and Thumb executable programs to be run on non-native hardware. The ARMulator is integrated with the ARM debuggers.
8.2 ARM Developer Suit

- Supported Standards
  - **ar**
    - UNIX-style archive files are supported by armar.
  - **DWARF2**
    - DWARF2 debug tables are supported by the compilers, linker, and debuggers. The deprecated format DWARF1 is supported in the debuggers only.
  - **ANSI C**
    - The ARM and Thumb compilers accept ANSI C as input. The option `-strict` can be used to enforce strict ANSI compliance.
  - **C++**
    - The ARM and Thumb C++ compilers support a subset of the ISO C++ language.
8.2 ARM Developer Suit

- **EC++**
  - The ARM and Thumb C++ compilers support the Embedded C++ (EC++) informal standard that is a subset of C++.

- **ELF**
  - The ARM tools produce ELF format files. The FromELF utility can translate ELF files into other formats.

- **RDI**
  - All debug agents and targets within ADS support version 1.5.1 of the Remote Debug Interface (RDI). The debuggers support all the debug agents (for example ARMulator and Remote_A) that are released as part of ADS. They also support Multi-ICE.
8.2 ARM Developer Suit

- Debugger Concepts
  - Debugger
  - Debug Target
  - Debug Agent
  - Remote Debug Interface
  - Single-processor Hardware
  - Multi-processor Hardware
  - Contexts
  - Scope
8.2 ARM Developer Suit

◆ Debugger

- A debugger is software (AXD) that enables you to make use of a debug agent in order to examine and control the execution of software running on a debug target.
- ARM eXtended Debugger (AXD)
8.2 ARM Developer Suit

- Debugger Concepts
  - Debugger
  - Debug Target
  - Debug Agent
  - Remote Debug Interface
  - Single-processor Hardware
  - Multi-processor Hardware
  - Contexts
  - Scope
8.2 ARM Developer Suite

- **Debug Target**
  - The debugger issues instructions that can:
    - Load software into memory on the target
    - Start and stop execution of that software
    - Display the contents of memory, registers, and variables
    - Enable you to change stored values
  - The form of the target is immaterial to the debugger as long as the target obeys these instructions in exactly the same way as the final product.
8.2 ARM Developer Suit

- Debugger Concepts
  - Debugger
  - Debug Target
  - Debug Agent
  - Remote Debug Interface
  - Single-processor Hardware
  - Multi-processor Hardware
  - Contexts
  - Scope
8.2 ARM Developer Suit

◆ Debug Agent

- A debug agent performs the actions requested by the debugger, for example:
  - Setting Breakpoints
  - Reading from Memory
  - Writing to Memory

- The debug agent is not the program being debugged, or the debugger itself. Examples of debug agents include:
  - Multi-ICE
  - ARMulator
  - Angel
8.2 ARM Developer Suit

- Debugger Concepts
  - Debugger
  - Debug Target
  - Debug Agent
  - Remote Debug Interface
  - Single-processor Hardware
  - Multi-processor Hardware
  - Contexts
  - Scope
Remote Debug Interface

The Remote Debug Interface (RDI) is an ARM standard procedural interface between a debugger and the debug agent.
8.2 ARM Developer Suit

- RDI gives the debugger a uniform way to communicate with:
  - A debug agent running on the host
  - A debug monitor running on ARM-based hardware accessed through a communication link
  - A debug agent controlling an ARM processor through hardware debug support
8.2 ARM Developer Suit

- Debugger Concepts
  - Debugger
  - Debug Target
  - Debug Agent
  - Remote Debug Interface
  - Single-processor Hardware
  - Multi-processor Hardware
  - Contexts
  - Scope
8.2 ARM Developer Suit

- Single-processor Hardware
  - In many cases, the target has only a single processor. All ARM debuggers can operate successfully on single-processor targets.

- Multi-processor Hardware
  - There is a growing requirement for multi-processor hardware:
    - Certain processors might be dedicated to particular tasks
    - Parallel processing might be appropriate and beneficial
  - In these cases the debugger must allow you to examine and control the processes happening simultaneously in a number of processors.
8.2 ARM Developer Suit

- Debugger Concepts
  - Debugger
  - Debug Target
  - Debug Agent
  - Remote Debug Interface
  - Single-processor Hardware
  - Multi-processor Hardware
  - Contexts
  - Scope
Contexts

- Each processor in the target can have a process currently in execution. Each process uses values stored in variables, registers, and other memory locations. These values can change during the execution of the process.

- The context of a process describes its current state, as defined principally by the call stack that lists all the currently active calls. When a function is called, and again when control is returned, the context changes.

- Every process has its own context. When execution of a process stops, you can examine and change values in its current context.
8.2 ARM Developer Suit

◆ Debugger Concepts
  ■ Debugger
  ■ Debug Target
  ■ Debug Agent
  ■ Remote Debug Interface
  ■ Single-processor Hardware
  ■ Multi-processor Hardware
  ■ Contexts
  ■ Scope
8.2 ARM Developer Suit

◆ Scope

The scope of a variable is determined by the point within a program at which it is defined. Variables can have values that are relevant within:

- A specific class only (Class)
- A specific function only (Local)
- A specific file only (Static Global)
- The entire process (Global)
The Quartus II development software provides a complete design environment for system-on-a-programmable-chip (SOPC) design. Regardless of whether you use a personal computer or a UNIX or Linux workstation, the Quartus II software ensures easy design entry, fast processing, and straightforward device programming. The following sections describe the general capabilities.

- Quartus II Highlights
- Design Capabilities
- NativeLink Integration with other EDA Tools
8.2 Quartus II

◆ Quartus II Highlights

- The Quartus II software offers a rich graphical user interface complemented with an illustrated, easy-to-use online Help system. The complete Quartus II system comprises an integrated design environment that includes every step from design entry to device programming.

- The Quartus II Compiler lies at the heart of the system, providing powerful design processing that you can customize to achieve the best possible silicon implementation of your project.
8.2 Quartus II

◆ Design Capabilities

The Quartus II software is a fully integrated, architecture-independent package for designing logic with Altera programmable logic devices (PLDs), including APEX 20K, APEX 20KC, APEX 20KE, APEX II, Excalibur, Cyclone, Cyclone II, FLEX 6000, FLEX 10K, FLEX 10KA, FLEX 10KE, HardCopy II, HardCopy Stratix, MAX II, MAX 3000A, MAX 7000AE, MAX 7000B, MAX 7000S, Mercury, Stratix, Stratix II, and Stratix GX devices. The Quartus II software offers a full spectrum of logic design capabilities:

- Design Entry Using Schematics, Block Diagrams, AHDL, VHDL, and Verilog HDL
8.2 Quartus II

- Floorplan Editing
- LogicLock Incremental Design
- Powerful Logic Synthesis
- Functional and Timing Simulation
- Timing Analysis
- Embedded Logic Analysis with SignalTap II Logic Analyzer
- Software Source File Importing, Creation, and Linking to Produce Programming Files
- Combined Compilation and Software Projects
- Automatic Error Location
- Device Programming and Verification
8.2 Quartus II

- NativeLink Integration with other EDA Tools

The Quartus II software provides NativeLink integration with major design tools to provide seamless transfer of information between the Quartus II software and other EDA tools. This NativeLink integration allows the Quartus II software to easily identify the source of errors in the EDA tool's source files, enabling you to correct them quickly. In addition, the Quartus II software allows you to run many EDA tools automatically from within the Quartus II software, further enhancing its integration into your design flow.
The Quartus II software also reads standard EDIF netlist files, VHDL netlist files, and Verilog HDL netlist files, and generates VHDL and Verilog HDL netlist files, including VITAL-compliant files, for a convenient interface to other industry-standard EDA tools.
8.2 Quartus II

❖ Quartus II Design Flow

- Design Entry
- Synthesis
- Place & Route
- Simulation
- Timing Analysis
- Programming & Configuration
- Power Analysis
- Debugging
- Engineering Change Management
- Timing Closure
8.2 Quartus II

◆ Design Entry

- Using the Quartus II Block Editor, Text Editor, MegaWizard Plug-In Manager, and EDA design entry tools to create the design files in a project.

- The MegaWizard Plug-In Manager helps you to create design files for customized megafunctions.

- The Quartus II software also supports EDIF Input Files (.edf) or Verilog Quartus Mapping Files (.vqm) generated by EDA design entry and synthesis tools.
8.2 Quartus II

◆ Synthesis

- Using the Analysis & Synthesis module of the Compiler to analyze and synthesize design files and create the project database. Analysis & Synthesis performs logic synthesis to minimize the logic of the design, and performs technology mapping to implement the design logic using device resources such as logic elements, and then it generates a single project database integrating all the design files in a design.

- Analysis & Synthesis uses Quartus II Integrated Synthesis to synthesize your Verilog Design Files (.v) or VHDL Design Files (.vhd).
Place & Route

The Quartus II Fitter, which is also known as the PowerFit Fitter, performs place and route, also referred to as "fitting" in the Quartus II software. Using the database that has been created by Analysis & Synthesis, the Fitter matches the logic and timing requirements of the project with the available resources of a device. It assigns each logic function to the best logic cell location for routing and timing, and selects appropriate interconnection paths and pin assignments.
8.2 Quartus II

- Power Analysis
  - Using The PowerPlay Power Analyzer
    - The PowerPlay Power Analyzer supports thermal and power supply planning by providing thermal power-dissipation and power-consumption estimates for a design, based on data from a variety of sources that you provide. The power analyzer produces a power estimate for the device, analyzing relevant device, design, and environmental factors that can have an impact on power consumption.
8.2 Quartus II

◆ Debugging

■ Using The SignalTap II Logic Analyzer

The SignalTap II Logic Analyzer lets you capture signals from internal device nodes while the device is running at speed. The captured data is displayed as a waveform within the SignalTap II Logic Analyzer and can be saved as a SignalTap II File (.stp). You can also instantiate one or more SignalTap II Logic Analyzer Megafunctions into the design using the MegaWizard Plug-In Manager and later create an STP File from the megafunction instance(s).
8.2 Quartus II

◆ Engineering Change Management
  ■ Making Post-Compilation Changes Introduction
  - The Quartus II software provides tools that allow you to view the placement and routing details of a compiled design, and to make minor changes to the design without having to recompile it. These capabilities enable you to correct functional flaws or implement last-minute engineering change orders (ECOs) on the post-fitting netlist, without having to rerun synthesis or place and route operations.
8.2 Quartus II

◆ Simulation

- The Quartus II Simulator is a tool for testing and debugging the logical operation and internal timing of your design. Because the Simulator allows you to verify your project before it is actually committed to hardware, it can significantly shorten the time it takes to transform your initial design concept into working silicon.
8.2 Quartus II

- Timing Closure

  - Working with Assignments in the Floorplan Editor

    - Creating and assign nodes or entities to custom regions and to LogicLock regions in the Timing Closure floorplan. Also edit existing assignments to device resources such as pins, logic cells, rows, columns, regions, MegaLAB structures, and LABs. The Timing Closure floorplan automatically displays user assignments and fitter placements, and allows you to edit assignments.
8.2 Quartus II

◆ Timing Analysis

Quartus II Static Timing Analysis is a method of analyzing, debugging, and validating the performance of a design. The Quartus II Timing Analyzer measures the delay of every design path and verifies the performance and operation of the design in terms of slack.
8.2 Quartus II

Programming & Configuration

The Programmer allows you to program or configure all Altera devices supported by the Quartus II software with files generated by the Compiler. The Assembler module of the Quartus II Compiler generates programming files that the Programmer can use to program or configure a device with Altera programming hardware. You can also use a stand-alone version of the Programmer to program and configure devices.
8.2 Logic Analyzer

- Basic Concept of Logic Analyzer
  - What is LA?
  - Why need LA?
  - Function of LA
  - Architecture of LA
    - Probe
    - Synchronous & Asynchronous
    - Trigger state machine
    - Acquisition memory
8.2 Logic Analyzer

- **What is LA?**

<table>
<thead>
<tr>
<th>Voltage vs. Time</th>
<th>Oscilloscope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power vs. Frequency</td>
<td>Spectrum or FFT Analyzer</td>
</tr>
<tr>
<td>Frequency vs. Time</td>
<td>Vector Analyzer</td>
</tr>
<tr>
<td>Logic vs. Time</td>
<td>Logic Analyzer</td>
</tr>
</tbody>
</table>
8.2 Logic Analyzer

◆ Why need LA?
  ■ LA can solve the following problem:
    ● Chase the immediate data of microprocessor
    ● Observe the multiple logic information at the same time
    ● Acquisition the systematic trouble of intermission
    ● Chase the reason of the system crash
    ● Development of Embedded System
8.2 Logic Analyzer

◆ Function of LA
  ■ Retrieve (Asynchronous and Synchronous)
  ■ Store
  ■ Trigger and Qualification
  ■ Display (State and Waveform)
8.2 Logic Analyzer

Diagram of Logic Analyzer
8.2 Logic Analyzer

◆ Architecture of LA
  ■ Probe
  ■ Asynchronous and Synchronous
  ■ Trigger State Machine
  ■ Acquisition Memory
8.2 Logic Analyzer

- LA Probes

Critical Voltage

LA Display
8.2 Logic Analyzer

- Characteristic of Digital Probe
  - Multi-channel
  - Limited Dynamic Range
  - Probe with Critical Point
    - Fix Value (TTL/CMOS Logic)
    - Variable Value (+/- 5 ~ 10V)
  - Probe can change or reduce bandwidth
8.2 Logic Analyzer

- Architecture of LA
  - Probe
  - Asynchronous and Synchronous
  - Trigger State Machine
  - Acquisition Memory
8.2 Logic Analyzer

- Asynchronous ➔ Timing
  - Produce Sample Pulse from Logic Analyzer
  - The sooner the better

- Synchronous ➔ State
  - Produce Sample Pulse from System
  - Logic Analyzer: it is enough for the acceptable external pulse
8.2 Logic Analyzer

- **Timing Analysis**
  - Produce Asynchronous Pulse from Logic Analyzer
  - Fast – Offer Sufficient Resolution

LA Sample Cycle

8 GHz = 125ps clock
8.2 Logic Analyzer

Logic Analyzer Error

LA Sample Cycle

Data

LA Display
8.2 Logic Analyzer

- Resolution vs. Accuracy

![Diagram of Logic Analyzer with Data 1 and Data 2, showing 12 samples for Data 1 and 1 sample for Data 2, with delays labeled as Delay 1 and Delay 2.]

- Delay 1 ≈ Delay 2
8.2 Logic Analyzer

- Conventional vs. Transitional
  - Transitional Timing

```
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | - Conventional = 8 locations |
Data: 1 0 1 0 0 - Transitional = 4 locations |
```

LA Sample Cycle
8.2 Logic Analyzer

- Bus Speed vs. Process Speed
  - Process Speed ≠ Bus Speed

33 MHz Clock = 11 MHz Bus

68020

33 MHz Clock = 66 MHz Bus

R3000
8.2 Logic Analyzer

Setup / Hold Time Requirement
8.2 Logic Analyzer

- Earlier than Setup Time

![Logic Analyzer Diagram]

DATA → D → Q → OUTPUT

CLOCK → CK

DATA
OUTPUT
CLOCK

Setup Hold
8.2 Logic Analyzer

- Violation Setup Time

![Diagram of logic analyzer with setup and hold times](image)
8.2 Logic Analyzer

- Violation Hold Time
8.2 Logic Analyzer

◆ Later than Hold Time

![Diagram of Logic Analyzer](image-url)
8.2 Logic Analyzer

◆ Time & State Operation

- **Timing** is clocking internal to the logic on a clock Edge
  - Store at Fixed Intervals – Asynchronous
  - Precise time interval measurement for signal to signal edges or pulse/event widths
  - Capture Glitches

- **State** is clocking external from the circuit under test
  - Store data when valid – Synchronous
  - Use for Processor Buses & State Machines
  - Setup/Hold Violations
8.2 Logic Analyzer

- Architecture of LA
  - Probe
  - Asynchronous and Synchronous
  - Trigger State Machine
  - Acquisition Memory
8.2 Logic Analyzer

- Triggering Source
  - Events
    - Timer/Counters
  - States/Levels
    - Search Events
  - Actions
8.2 Logic Analyzer

- Triggering Resources
  - Events
    - Word Recognition
    - Sequence Recognition
    - External Event
    - Range Recognition
    - Counter
    - Timer
8.2 Logic Analyzer

- Actions
  - Stop/Start Storage
  - External Pulse
  - Pause Data Storage
  - Trigger
  - Go State X
  - Control Timer
  - Control Counter
8.2 Logic Analyzer

- Trigger State Machine
  - Control Counters/Timers
  - Go to State X
  - Trigger Other Modules
  - Trigger System
8.2 Logic Analyzer

- AND/OR Events
- If/Then/Else

Diagram:

- State 0
- IF (test 0) → Else -> Conditions → Actions
- IF (test 1) → IF (test 2) → Else → IF (test 3)
8.2 Logic Analyzer

- Architecture of LA
  - Probe
  - Asynchronous and Synchronous
  - Trigger State Machine
  - Acquisition Memory
8.2 Logic Analyzer

- Acquisition memory
  - The main materials storing
  - Deeper memory
    - Transitional Timing
    - Qualified Storage
8.2 Logic Analyzer

- Qualified Storage

I/O Program - Reads Data & Writes Data

- Initialize
- Idle Loop
- Data Read
- Process Data
- Wait for Device Ready
- Data Write

Store

Store
8.3 Introduction to GDB Server

9.1 Hardware / Software Co-design on Prototype
9.2 Test and Debug on Prototype
9.3 Introduction to GDB Server
9.4 Reference
8.3 Introduction to GDB Server

- GDB Commands
- How to install GDB
- Compiling for Debugging
8.3 Introduction to GDB Server

- The purpose of a debugger such as GDB is to allow you to see what is going on "inside" another program while it executes or what another program was doing at the moment it crashed.

- GDB can do these main kinds of things to help you catch bugs in the act:
  - Start your program, specifying anything that might affect its behavior.
  - Make your program stop on specified conditions.
  - Examine what has happened, when your program has stopped.
8.3 Introduction to GDB Server

▶ You can use GDB to debug programs written in C or C++, Modula-2 and Chill is partial.

▶ Debugging Pascal programs which use sets, file variables, or nested functions does not currently work. GDB does not support entering expressions, printing values, or similar features using Pascal syntax.

▶ GDB can be used to debug programs written in Fortran, although it does not yet support entering expressions, printing values, or similar features using Fortran syntax.
8.3 GDB Commands

- A GDB command is a single line of input. There is no limit on how long it can be. It starts with a command name, which is followed by arguments whose meaning depends on the command name.
- GDB command names may always be truncated if that abbreviation is unambiguous. Other possible command abbreviations are listed in the documentation for individual commands.
- You can always ask GDB itself for information on its commands, using the command `help`. 
8.3 GDB Commands

◆ In addition to help, you can use the GDB commands info and show to inquire about the state of your program, or the state of GDB itself.

- **info**
  - This command (abbreviated i) is for describing the state of your program.

- **show**
  - In contrast to info, **show** is for describing the state of GDB itself.
8.3 How to install GDB

◆ Compiler GDB Server for X86

■ Step 1.
  - # cd /root
  - # tar -zxf gdb-6.6.tar.gz

■ Step 2.
  - # cd /root/gdb-6.6
  - # ./configure --target=arm-linux \
    --prefix=/usr/local/arm/gdb

■ Step 3.
  - # make
  - # make install
8.3 How to install GDB

Cross Compiler GDB Server for ARM

- **Step 1.**
  - `cd /root/gdb-6.6/gdb/gdbserver`
  - `./configure --host=arm-linux`

- **Step 2.**
  - `export CC=/usr/local/arm/2.95.3/bin/arm-linux-gcc`
  - `make`

- **Step 3.**
  - `mkdir SDLlib`
  - `cp /usr/local/arm/2.95.3/arm-linux/lib/libthread_db-1.0.so SDLlib`
  - `cp /usr/local/arm/2.95.3/arm-linux/lib/libthread_db.so SDLlib`
  - `cp /usr/local/arm/2.95.3/arm-linux/lib/libthread_db.so.1 SDLlib`
8.3 Compiling for Debugging

- When you run a program under GDB, you must first generate debugging information when you compile it.
- You may start GDB with its arguments, if any, in an environment of your choice.
- If you are doing native debugging, you may redirect your program's input and output, debug an already running process, or kill a child process.
8.3 Compiling for Debugging

◆ In order to debug a program effectively, you need to generate debugging information when you compile it.

◆ This debugging information is stored in the object file; it describes the data type of each variable or function and the correspondence between source line numbers and addresses in the executable code.

◆ To request debugging information, specify the "-g" option when you run the compiler.
8.3 Compiling for Debugging

- The GNU C/C++ compiler, supports “-g” with or without “-O”, making it possible to debug optimized code.
- Many compilers are unable to handle the “-g” and “-O” options together. Using those compilers, you cannot generate optimized executables containing debugging information.
- When you debug a program compiled with “-g –O”, remember that the optimizer is rearranging your code.
8.3 Compiling for Debugging

- Older versions of the GNU C compiler permitted a variant option “-gg” for debugging information. GDB no longer supports this format; if your GNU C compiler has this option, do not use it.
- GDB knows about preprocessor macros and can show you their expansion.
- Most compilers do not include information about preprocessor macros in the debugging information if you specify the “-g” flag alone, because this information is rather large.
Reference

◆ 探矽工作室 胡繼陽、侯玉展、陳澤民，ARM韌體教戰手冊—使用Net-Start! Lite & ADS
◆ 范盛一著，ARM原理與嵌入式系統實作
◆ Tektronix，邏輯分析儀概念與基本原理
◆ ARM Developer Suit Online Book
◆ AXD and armd Debuggers Guide
◆ Quartus II User Guide
◆ GDB Online DOCS