DTV Broadcasting System
Chapter 6
Terminal System

- Digital Receiver
- Embedded Operating System
Digital Receiver

- STB Setup

- STB Back Panel
Digital Receiver—Video Interface (Analog)

- **Video: Analog Interface**

1. **1RCA (CVBS)**
   - All the signals are composite together. TV needs to separate the signal to R, G, B before display.
   - Poor quality.

2. **S-Video (Y/C)**
   - Separate luma (Y) and chroma (C) signals
   - TV still needs to separate signals from it
   - Quality also not good.

3. **3RCA (YCbCr/YPbPr)**
   - Better quality than S-video
   - Support for 480i, 480p, 720p, 1080i

4. **D-sub (or VGA, RGB)**
   - Used by computer display card
Digital Receiver – Video Interface (Analog)

- **Video: Analog Interface**

5. SCART (CVBS, S-Video)
   - Used in Europe
   - It can carry CVBS and S-Video (and analog audio)

6. D-Connector (YPbPr)
   - Japan format (EIA-J)
   - D1, D2, D3, D4, D5 (from 480i to 1080p)

<table>
<thead>
<tr>
<th>Name</th>
<th>Resolution</th>
<th>H. Freq.</th>
<th>V. Freq.</th>
<th>Scanning</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1 SDTV480I</td>
<td>720x480</td>
<td>15.75KHz</td>
<td>30Hz</td>
<td>Interlaced</td>
</tr>
<tr>
<td>D2 SDTV480P</td>
<td>720x480</td>
<td>31.5KHz</td>
<td>60Hz</td>
<td>Progressive</td>
</tr>
<tr>
<td>D3 HDTV1080I</td>
<td>1920x1080</td>
<td>33.75KHz</td>
<td>30Hz</td>
<td>Interlaced</td>
</tr>
<tr>
<td>D4 HDTV720P</td>
<td>1280x720</td>
<td>45KHz</td>
<td>60Hz</td>
<td>Progressive</td>
</tr>
<tr>
<td>D5 HDTV1080P</td>
<td>1920x1080</td>
<td>67.5KHz</td>
<td>60Hz</td>
<td>Progressive</td>
</tr>
</tbody>
</table>
Video: Digital Interface

1. DVI (Digital Visual Interface)
   - Developed by an industry consortium, the Digital Display Working Group (DDWG).
   - To maximize visual quality of digital display devices (Plasma, PLD, LCD, digital projector, etc.)
     • carrying uncompressed digital video data
     • No D/A and A/D is needed
     • Speed: single-link: 3.7Gbps, dual-link: 7.4Gbps
   - Three types
     • DVI-D (digital), DVI-I (digital & analog), DVI-A
     • Display cards usually use DVI-I, and monitors usually use DVI-D (monitor has VGA already)
   - Work with HDCP for copy protection
     • But need to pay for license
     • Not much monitors with DVI support HDCP so far.
Video: Digital Interface

2. **HDMI** (High-Definition Multimedia Interface)
   - HDMI: digital video/audio (for CE); DVI: video only (for PC)
   - **Digital video**: 480i, 480p, 720p, 1080i, 1080p (uncompressed)
   - **Digital audio**: compressed: Dolby, DTS
     - uncompressed: 8 channels, 192KHz, 24Bits
     - (Note: CD audio: 2 channels, 44.1KHz, 16bits)
   - Support HDCP (High-bandwidth Digital Content Protection)
Analog TV – Video Interface (Digital)

- **Video: Digital Interface**
  - Bandwidth Comparison
Digital Receiver—Audio Interface

• **Audio: Analog Interface**
  - Baseband audio
    - RED: Right channel
    - WHITE: Left channel
  - Sub-woofer

• **Audio: Digital Interface**
  - S/PDIF (Coaxial)
  - S/PDIF (Optical)
Digital Receiver

• Hardware
  - An example

Note: Tuner and demodulator are not put into DTV SoC for the flexibility to be used for Satellite, Cable, Terrestrial applications.
DTV SoC — An Example

- TS from Front-End
- External Memory
- HSDI
  - 1394 I/F
  - 1284 I/F
  - Ext DMA
- ECD
- TPP
- Traffic Controller
- Video Decode / Display
- NTSC/PAL Encoder
- OSD
- Audio Decode
- Y/C or RGB and Composite Video
- PCM Audio SPDIF Output
- Comm. Processor
- Extension Bus Interface
- MIPS 3201
- Data RAM
- ROM
- GPIO
- I2C Smart Card
- UART (three)
- IR
- MIPS 3201 Processor

11
DTV SoC – An Example
TPP: Transport Processor Block Diagram
• Data Transport Processor Features
  – Capable of processing three independent transport streams simultaneously
  – Maximum transport stream rate supported is 78Mbps
  – Two independent PCR recovery blocks
  – Support 64 entry PID table for parsing MPEG TS packets. PID table entries can be arbitrarily assigned to any of three transport streams.
  – Support 64/56-bit keys of DES/Multi2/DVB Descrambler for up to 64 PIDs.
  – PES packet extraction for up to 64 PID channels.
  – Support 64 section filter coeff, capable of filtering 16 bytes for each of 64 PID channels. Support bit mask and exclusive mask.
  – Support two independent record channels, and three playback channels.
  – Support 3DES/DES record scrambler and playback de-scrambler.
  – Support memory buffer management and DMA controller.
DTV SoC – An Example
Digital Video Decoding Block Diagram
DTV SoC – An Example

- Digital Video Decoding Functional Block Diagram
  - Capable of decoding all MPEG MP@HL video streams.
  - Capable of decoding two MPEG video simultaneously.
  - Include four custom RISC processors:
    - **TRISC**: TP Processor that performs multi-channel TS/PES/ES decode.
    - **VideoRISC**: Picture processor that performs picture layer of processing.
    - Two identical **RowRISC**: macroblock and block layers of processing.
    - Firmware for the four processors is executed from on-chip memory.
  - Support two video display feeders to have two independent video scalars. This configuration supports PIP and other advanced feature.
  - Multi-client SDRAM controller
DTV SoC – An Example

Video Display Functional Block Diagram
DTV SoC – An Example

- Video Display Functional Block Diagram
  - Support two video scalers: one HD and one SD Scaler.
  - The horizontal and vertical scale factors can be programmed independently to support such case as 1080i, 720p, 576p/i, 480p/i.
  - The HD/SD scaler can handle pan-scan vector for aspect ratio purpose
  - Support video display compositor that can combine up to
    - four graphic layers
    - a background color
    - two video layers
    - Also support color space conversion (YUV/RGB)
  - HD/SD CRT controller generate horizontal/vertical timing for CRT
  - VWIN provides playback engine to display video from SDRAM. It also support progressive conversion.
  - HD_DENC: for routing video data from different sources to different 6 DAC outputs (3 for component, 2 for s-video, 1 for composite).
Digital Receiver

- **DTV SoC**
- **OSD** (On Screen Display)
  - Multi-layer display
  - Alpha blending
Digital Receiver

- **Software Stack**
  - Basic Software Map
    - Application
    - Device drivers
    - CA System
    - OS
    - STB hardware
  - Software Map with Middle Layer
    - Application
    - Middle Layer
    - CA System
    - Device drivers
    - OS
    - STB hardware
Digital Receiver

- **Software Stack** — An example

![Diagram of Digital Receiver software stack](image-url)
DVT Application Software – **Single-task Architecture**

- **State machine**
  - Control state transition to achieve time sharing purpose

- **DTV software as an example**
  - Tuner state
  - Section filter state
    - SI/PSIP parsing state
    - Channel/Event DB building state
  - RCU/FP key processing state
    - User Interface state
    - channel navigation state
    - Menu setting state
  - OSD state
  - CA state
DVT Application Software – Multitask Architecture

- Channel/Event DB Building Task
- SI/PSIP Task
- Tuner task
- Section Filter Task
- User Interface Task
- OSD Task
- FP/RCU key Task
- Mutex/semaphore
- Ch/Event DB
- Message Queue
- Timer

- CA related task
  - ECM/EMM task
  - CA task
  - Smart card task

- Middleware related task
  - •••
  - •••
  - •••
  - •••
Chapter 6
Terminal System

• Digital Receiver

• Embedded Operating System
  – RTLinux
  – uCLinux
  – eCos
  – VxWorks
RTOS

• Pure Real-time OS
  – Especially designed for real-time requirements
  – Completely real-time compliant
  – Often usable for simple architecture
  – **Advantage**: no or little overhead (computing power, memory, etc.)
  – **Disadvantage**: limited functionality
  – Example: VxWorks, QNX, Nucleus, eCos, pSoS, OSE, Lyra

• OS Real-time Extensions
  – Extension of an OS by real-time components
  – Cooperation between RT- and non-RT parts
  – **Advantage**: rich functionality
  – **Disadvantage**: No general real-time ability
    Computing and memory resources
  – Example: RT-Linux, Solaris, Windows NT.
RTOS – Embedded Linux

• What is embedded Linux?
  – When someone talks about embedding Linux, they mean porting Linux kernel to run on a particular CPU and board for a device.
  – There are many companies that sell embedded Linux solutions, which usually include a ported Linux kernel (for particular CPU) with cross-development tools, and sometimes with real-time extensions.
  – For the most part, the APIs and kernel code-base are the same for embedded Linux as desktop Linux.

• Why embedded Linux?
  – No runtime royalties 19.2%
  – Source code is available (and free) 20.2%
  – It’s not from Microsoft 8.7%
  – Linux has excellent networking support 16.2%
  – More drivers and tools available 9.2%
  – Lots of programmers familiar with Linux 7.7%
  – Linux is more robust and reliable 17.0%
  – Others (many CPU architectures support, easy to configure, etc.) 1.5%
RTLinux
RTLinux

• **What is RT-Linux?**
  – RT-Linux is an operating system, in which a small real-time kernel co-exists with standard Linux kernel
  – The real-time kernel sits between *standard Linux kernel* and the *h/w*.
  – The standard Linux kernel sees this real-time layer as actual h/w
  – The real-time kernel *intercepts all hardware interrupts*.  
    • Only for those RTLinux-related interrupts, the appropriate ISR is run.
    • All other interrupts are held and passed to the standard Linux kernel as software interrupts when the standard Linux kernel runs.
  – The real-time kernel assigns the *lowest priority* to the *standard Linux kernel*. Thus the realtime tasks will be executed in real-time
  – user can create realtime tasks and achieve correct timing for them by deciding on scheduling algorithms, priorities, execution freq, etc.
  – Realtime tasks are *privileged* (that is, they have direct access to hardware), and they do *NOT use virtual memory.*
RTLinux

- **RTLinux Architecture**

  Standard Linux is executed in the background

  ![Diagram showing RTLinux architecture](image)
RTLinux

• Process Scheduler
  – RT-Linux contains a dynamic scheduler
  – RT-Linux has many kinds of real-time schedulers
    • EDF
    • Rate-monotonic

• Real-time FIFOs
  – They are used to pass information between real-time process and ordinary Linux process.
  – They are designed to never block the real-time tasks.
  – Like realtime tasks, they are never page out. This eliminates the problem of unpredictable delay due to paging.
Linux v.s. RTLinux

- **Linux Non-real-time Features**
  - Linux scheduling algorithms are not designed for real-time tasks
    - Provide good *average* performance or throughput
  - Unpredictable delay
    - Uninterruptible system calls, the use of interrupt disabling, virtual memory support (context switch may take hundreds of microseconds).
  - Linux Timer resolution is coarse, 10ms
  - Linux Kernel is Non-preemptible.

- **RTLinux Real-time Features**
  - Support real-time scheduling
  - Predictable delay (by its small size and limited operations)
  - Finer time resolution
  - Preemptible kernel
  - No virtual memory support
uCLinux
uCLinux

• What is uCLinux?
  – Stands for “Micro-Control-Linux” (pronounced “you-see-linux“)
  – Developed and maintained by Lineo Corp.
  – uCLinux features
    • Open source
    • Linux-based
    • Targeted for embedded systems, not for real-time systems
    • Small footprint (compared with standard Linux)
    • No MMU
    • Internet-ready
    • Excellent file system support
    • Flat file format
    • Execute-in-place (XIP) support
uCLinux

• **Open source OS**
  – under GNU General Public License (GPL).
  – the source can be freely downloaded from [www.uClinux.org](http://www.uClinux.org).

• **Linux-based**
  – derived from Linux 2.x. Kernel releases include 2.0, 2.4, 2.6
  – still retaining the main advantages of standard Linux: stability, superior network capability, excellent file system support, standard and common Linux APIs support.

• **Targeted for embedded systems, not real-time systems**
  – Intended for MMU-less microcontrollers.
  – Use non-preemptive kernel, same as standard Linux.

• **Small footprint**
  – most of uCLinux kernel, services and application library have been rewritten, and heavily optimized to slim-down the code base.
    • uCkernel < 512 KB
    • uCkernel + tools < 900 KB
    • Instead of using standard Linux library glibc, it provides uClibc
uCLinux

- **No MMU**
  - Reduce power, memory, and cost overhead.
  - Low overhead of context switch
    - Since all processes share a single address space, no need to flush or invalidate virtual cache after context switch.
    - Reduce cache miss at first access after context switch.
  - Side-effects
    - **no memory protection.**
      - Any application or kernel may corrupt any part of system.
    - **static code space**
      - Code area in physical memory must be contiguous and cannot be expanded at runtime as there may be other processes beside it.
    - **missing memory due to fragmentation**
      - If we do memory de-fragmentation (i.e., to move memory) and without VM to make it transparent to programs (appear to be at the same address), the programs will easily get crashed.
    - **multitasking can be tricky**
uCLinux

• **Internet-ready**
  – full TCP/IP stacks, and numerous other network protocols. (Same as Linux.)

• **Excellent file system support**
  – NFS, ext2, ROMfs, JFFS, MS-DOS, and FAT16/32
    • ROMfs: It is a small-sized, read-only file system, directly running on FLASH/ROM (save RAM space). It is widely used in uCLinux.

• **Flat file format**
  – all standard executable formats are not supported, because of no VM.
  – uCLinux uses “Flat” format, which is a condensed executable format.
    • stores only executable code and data, along with relocations needed to load the executable into memory.

• **Execute-in-place (XIP) support**
  – For kernel to boot, or applications to run directly on Flash/ROM. (Standard Linux booting is also supported.)
    • Currently, only the m68k and ARM toolchains provide the support for it.
    • XIP must run with ROMfs.
    • Shared libraries under uClinux must be compiled as flat XIP. Without XIP, shared libraries result in a full copy of the library for each application using it.
eCos
What is eCos?

- Stands for "Embedded Configurable Operating System"
- Developed and maintained by RedHat
- eCos features:
  - Not Linux-based
  - Minimum footprint: about 50KB
  - Configurable
  - Portable
  - Under eCos license
  - Real-time support
  - General embedded application support
eCos

• Configuration system
  – this enables eCos developers to create their own application-specific operating system and makes eCos suitable for a wide range of embedded uses.
  – Configuration also ensures that the resource footprint of eCos is minimized as all unnecessary functionality and features can be removed. The minimum footprint is about 50KB.
  – The configuration system also presents eCos as a component architecture. This provides a standardized mechanism for component suppliers to extend the functionality of eCos.
    • allows applications to be built from a wide set of optional configurable components provided from a variety of sources including the standard eCos release, commercial third party developers, and open source contributors.
eCos

- **Portable**
  - eCos has been *ported* to a variety of architectures and platforms
    - ARM, Hitachi H8300, Intel x86, MIPS, Matsushita AM3x, Motorola 68k, PowerPC, SuperH, SPARC and NEC V8xx.
    - Also include many of the popular variants of these architectures and evaluation boards.
  - The eCos kernel, libraries and runtime components are layered on the *Hardware Abstraction Layer (HAL)*, and thus will run on any target once the relevant *HAL* and *device drivers* have been ported to the target's processor architecture and board.

- **eCos license (very similar to GPL)**
  - Royalty-free: you can deploy your product using the standard eCos release without incurring any royalty charges.
  - No up-front license charges for the eCos runtime source code and associated tools.
eCos

- **Real-time Support**
  - full preemptability, minimal interrupt latencies, and all the necessary synchronization primitives, scheduling policies, and interrupt handling mechanisms needed for real-time applications.

- **General embedded application support**
  - including device drivers, memory management, exception handling, C, math libraries, etc.
  - includes all the tools necessary to develop embedded app.
    - including eCos software configuration and build tools,
    - support GNU based compilers, assemblers, linkers, debuggers (GDB), and simulators.
eCos

- eCos Architecture

Diagram showing the eCos architecture with layers and components such as Libraries, Compatibility, Kernel, Networking Stack, File System, RedBoot ROM Monitor, Hardware Abstraction Layer (HAL), and Device Drivers.
## eCos v.s. Linux

<table>
<thead>
<tr>
<th></th>
<th>Linux</th>
<th>eCos</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMU</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>Memory map</td>
<td>Virtual &amp; Physical</td>
<td>Single address</td>
</tr>
<tr>
<td>License</td>
<td>GPL</td>
<td>eCos license</td>
</tr>
<tr>
<td>ISO C library</td>
<td>O</td>
<td>Separate Math lib</td>
</tr>
<tr>
<td>POSIX</td>
<td>Not fully support</td>
<td>Not fully support</td>
</tr>
<tr>
<td>Real-time</td>
<td>X</td>
<td>▲</td>
</tr>
<tr>
<td>Device driver</td>
<td>In the Kernel</td>
<td>Separate from Kernel</td>
</tr>
<tr>
<td>Minimal footprint</td>
<td>About 2MB</td>
<td>About 50KB</td>
</tr>
</tbody>
</table>
POSIX

What’s POSIX?
- Posix is an acronym for Portable Operating System Interface.
- Open software architecture – interoperability & portability
- Developed by the Portable Applications Standards Committee (PASC) of the IEEE Computer Society.
- The POSIX standard, IEEE Std 1003.1-1990, is an international standard that defines common interfaces as a basis for open systems.
- IEEE POSIX 1003 is a group of standards, each of which addresses a specific area of system technology. The POSIX standards describe functions of an operating system interface. Systems that implement the standard functions are said to be POSIX-compliant.
- The reason for implementing POSIX functions on a system is to increase software portability and minimize porting costs.
  - Applications developed on any POSIX-compliant system using POSIX functions can be ported to other POSIX-compliant systems with little or no modification.
<table>
<thead>
<tr>
<th>1003.1a</th>
<th>OS Definition</th>
<th>Basic OS interfaces: single process, multi process, signals, file system, device I/O, pipes, FIFO...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003.1b</td>
<td>Real-time Extensions</td>
<td>Functions needed for real-time systems: real-time signals, priority scheduling, timers, asynchronous/synchronized I/O, message passing, semaphores</td>
</tr>
<tr>
<td>1003.1c</td>
<td>Threads</td>
<td>Functions to support multiple threads within a process: thread control, thread attributes, priority scheduling, mutexes, mutex priority inheritance, mutex priority ceiling, and condition variables</td>
</tr>
<tr>
<td>1003.1d</td>
<td>Additional Real-time Extensions</td>
<td>Additional interfaces: I/O advisory information, timeouts on blocking functions, device control, and interrupt control</td>
</tr>
<tr>
<td>1003.1j</td>
<td>Advanced Real-time Extensions</td>
<td>More real-time functions</td>
</tr>
<tr>
<td>1003.21</td>
<td>Distributed Real-time</td>
<td>Functions to support real-time distributed communication: buffer management, priorities, message labels, and implementation protocols</td>
</tr>
<tr>
<td>1003.2h</td>
<td>High Availability</td>
<td>Services for Reliable, Available, and Serviceable (SRASS): includes support for: logging, core dump control, shutdown/reboot, and reconfiguration</td>
</tr>
</tbody>
</table>