DAVID® System Specification

Version 2.5

www.radisys.com

World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision A
November 2001

Copyright and publication information

This manual reflects version 2.5 of DAVID.
Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

November 2001

Table of Contents

Chapter 1: Architecture

The Microware DAVID system architecture is based on the modular and flexible Microware OS-9 real-time operating system, making it easy to adapt to a variety of digital devices with varying levels of connectivity and interactivity.

This chapter contains information about the DAVID software architecture:

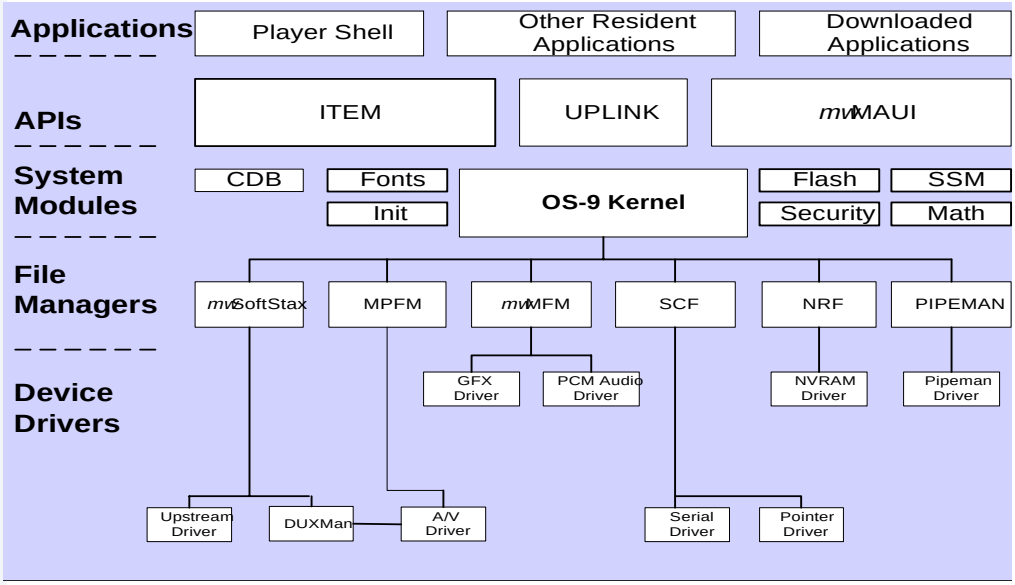
- **System Architecture**
- **Configuration Description Block**
- **DAVID Start-up Procedures**
- **Application Start-up Environment**

RadiSys.
MICROWARE SOFTWARE

System Architecture

Because DAVID is modular and flexible, your system can be configured to your specific application. By selecting only the modules your application uses, the memory footprint can be minimized.

Figure 1-1 DAVID System Architecture



As shown in **Figure 1-1**, there are five levels to the DAVID Software Model. The lowest level is the Device Drivers. This level contains all hardware-dependent software. This is the software that manufacturers must customize to match their particular hardware design.

The second level is the File Managers that perform functions for a class of devices in a hardware-independent way. For example, MPFM handles the queuing and synchronization of multiple requests for audio or video decoding, without talking directly to the decoders themselves.

The third level is the Kernel or System Layer. This level contains the core operating system services and forms the boundary between system state and user state. Along with the kernel are a series of trap handlers that manage hardware and data modules that customize the system for a particular hardware design.

The fourth level is the API (Application Program Interface) layer. The API layer contains linked or shared libraries that do not have any hardware dependencies but use the resources provided by the lower system layers. In DAVID, MAUI, ITEM, and UpLink are shared libraries, meaning that the application links to them an run time.

The final level is the Application layer. The application layer contains the highest level applications or processes that execute in the system. Some applications may be resident in the STB. Others are downloaded into the STB from the network.

System Components

Each file manager or API is responsible for managing one of the hardware components described in the previous section. Some of these components are ROM-resident in the STB, while others are linked in applications. The DAVID system components are modular, meaning that if a component or API is not needed for a particular application, it does not need to be included with the application. This gives the programmer more control over the finished size of the application.

Table 1-1 System Components

Type	Module	Purpose
File Manager	SoftStax (SPF)	Stacked Protocol File Manager. Manages network communications, including high speed isochronous data channels and lower speed control/data channels.
File Manager	MPFM	Motion Picture File Manager. Manages the decompression and display of MPEG encoded full-motion video and audio.
File Manager	MFM	Multimedia File Manager. Manages the graphics display.
File Manager	SCF	Serial Character File manager. Manages pointer devices (game pads and infrared remote controls) and serial communications.
File Manager	NRF	Non-volatile RAM File Manager. Manages a file system for user preferences stored in the optional non-volatile RAM.
File Manager	Pipeman	Pipe File Manager. Manages named and unnamed pipes for interprocess communication.

Table 1-1 System Components (continued)

Type	Module	Purpose
System API	ITEM	Integrated Telecommunications Environment for Multimedia. A high-level library that provides an abstraction layer for network services such as channel management, download management, and (if required) session management.
System API	UpLink	A high level library that provides a service for accessing application assets (graphics, audio, data) from an interactive network or cyclic broadcast service.
System API	MAUI	Multimedia Application User Interface. A high-level library for managing the display of graphics, messaging, and user input.

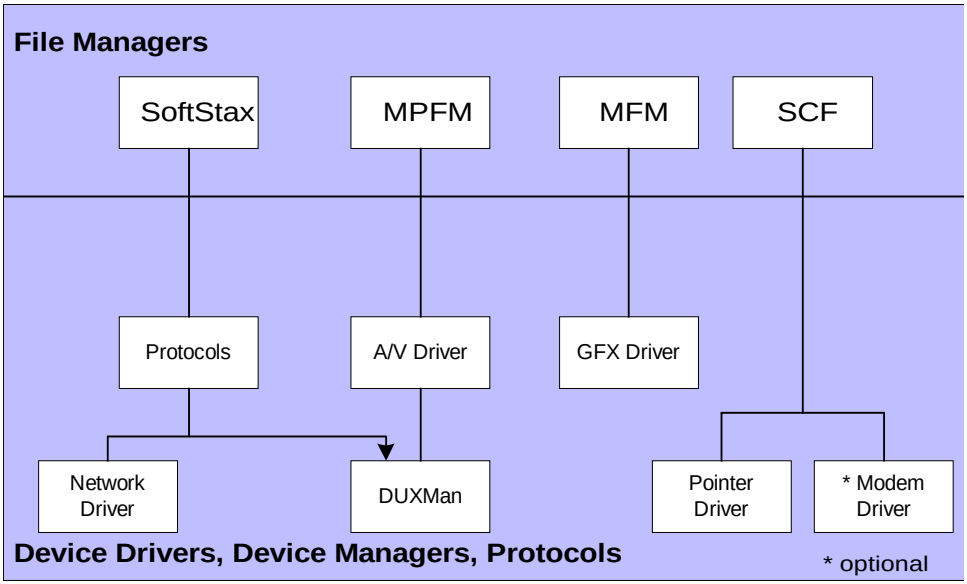
The modules listed above must be resident in memory after initialization. In addition, the C Shared Library (CSL) Trap must be in memory. FLASH is optional. After initialization/boot-up, OS-9 (v2.0 or later) must be resident in memory.

Additional I/O systems may be added to the DAVID system as manufacturer options.

Device Driver Level

The lowest level of the software in DAVID comprises device drivers, device managers, and protocols. The relationship of these software components to the software components on the File Manager level is shown in **Figure 1-2**.

Figure 1-2 Device Driver Level



Configuration Description Block

To access the I/O system in a DAVID environment, applications must know the name of the device which they are attempting to access. However, these names may differ from STB to STB. Additionally, one STB may have a single device (such as an MPEG decoder) while another STB may have two.

The DAVID system provides for this through the Configuration Description Block (CDB). The CDB is one or more data modules that contain a textual description of the STB. It contains a series of termcap-like entries for each device in the system. Each entry describes the device type, device name, and device parameters. The first line of the example shown below describes the system device (device type 0). This device, named `sys`, contains the following parameters: CP (CPU Motorola 68340), OS (operating system OS-9), RV (operating system revision 3.0), DV (DAVID system version 2.1), SR (System RAM of 2 MB), GR (Graphics RAM of 512 KB in memory color 128) and GR (Graphics RAM of 512 KB in memory color 129).

```
0:sys:CP=”68340”:OS=”OS9”:RV=”3.0”:DV=”2.1”:
SR#2048,1:GR#512,128:GR#512,129:
3:/gfx:AI=”MAUI”:
4:/nvr:
5:/rem/genrem:
9:/pipe:
20:/term:
20:/t1:
90:/mv:
91:/ma:
113:/sp0/lapb/x25:
114:/r0:HD:
```

A system may contain more than one device of a given type (such as two device types 20), but each device must have a unique name. When an application needs to open a device, it asks the CDB API for the name of the device with the specified type code. If more information is needed, the associated parameter strings for that device are returned as well.

For More Information

For more information on the CDB, see ***Using MAUI***.

DAVID Start-up Procedures

There are several distinct steps to DAVID system initialization. These steps are categorized as follows:

1. Manufacturer-dependent (bootstrap) routines executed prior to the kernel.
2. Kernel routines executed during coldstart.
3. Initial process routines.

Bootstrap Routines

The first step in the initialization process is to execute the bootstrap routines. These routines may vary widely from one system to the next, depending on the complexity of the design. In some cases, the operating system is stored locally in ROM or FLASH, in other cases, the bootstrap downloads the operating system from the network. In general, the bootstrap routines follow this procedure:

1. Reset and initialize system hardware.
2. Perform the Power On Self Test (POST) routines.
3. Locate (and initialize, if necessary) system ROM and RAM.
4. Locate and execute the kernel.

The first three steps are the same, regardless of where the kernel is. The last step varies as described in the following paragraphs.

Even if the bootstrap expects to download an operating system, it typically first looks to see if one exists in ROM or RAM. If it does find one locally, it compares the local version number with one available on the network for downloading. When the latest version is loaded, the bootstrap goes on to the next step.

In other cases, the bootstrap finds and executes a reduced portion of the operating system in ROM and executes a program to do this comparison and download. This configuration acts to simplify the bootstrap procedure.

Finally, it is possible that the bootstrap may encounter multiple copies of a kernel. This happens when a version stored in ROM is replaced by one contained in FLASH. In this case, the bootstrap executes the kernel with the highest revision number.

Kernel Coldstart Routines

After the bootstrap routines are completed, the kernel performs the following functions:

1. Builds the Free RAM list.
2. Locates modules in ROM and FLASH and builds the module directory.
3. Initializes internal system global data structures and interrupt vectors.
4. Links to the `init` module.
5. Starts the system clock.
6. Executes kernel customization (`os9p2`) modules (if any).
7. Opens system I/O paths.
8. Locates and executes the initial system process (usually the Player Shell).

These routines ensure the system comes up in a known and predictable state each time it is reset. Again, the kernel may encounter multiple copies of a single module; one in ROM and one or more in FLASH. In each case, the kernel chooses the one with the highest revision number for insertion into the module directory.

The amount of time spent booting to this point is highly dependent on the size of the search area, the number of modules found and the amount of RAM initialized. In building the free memory list, the kernel searches through the possible areas identified by the `init` module. When it finds RAM at a given address, the address of the memory block is added to the list. This process may be accelerated by keeping the search block size as large as possible.

Initial Process Routines

After initializing the system, the kernel looks in the `init` module to identify the initial process, the first application to run in the STB. The job of the first application varies widely from system to system, dependent on the network architecture and the requirements of the system operator.

In a DAVID STB, the initial process is typically known as the Player Shell.

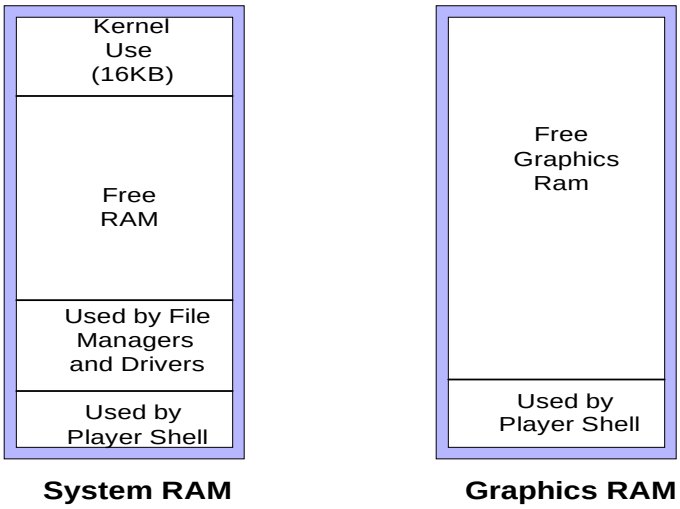
For More Information

For more information on bootstrap and kernel routines, see the ***OS-9 Technical Manual***. For more information on the Player Shell, see ***DAVID Utilities and Applications.***

Application Start-up Environment

The application in a DAVID environment must have a guaranteed set of system resources available to it in order to function properly. This applies primarily to system and graphics RAM, which might be used by the operating system or various background processes running in the system. System resources are shown in **Table 1-3**.

Figure 1-3 Typical Memory Map



The kernel uses a small amount of system memory at the lowest address available for its system data structures. This area includes the interrupt vectors, module directory, path table, and process table. Because the first application executed is the Player Shell, its data area is allocated from the highest addresses of free system RAM. The actual amount of RAM allocated is dependent upon the number of features in the Player Shell.

An additional amount of RAM is used by the file managers and drivers for their data structures. As described in the previous section, the Player Shell initializes all of the system devices at start-up. This causes all global storage allocations to be concentrated together.

This leaves a large, contiguous block of free memory available for applications to use. Each time an application is finished executing, the kernel frees up the memory, preparing the system for the next application.

For More Information

For a more complete discussion of memory management, refer to the ***OS-9 Technical Manual***.

Application Naming Conventions

All modules in OS-9 have a name. When modules are loaded or executed, they are placed in a “module directory”. To prevent name clashes in this directory, we highly recommend that modules included in ROM (including the entire DAVID OS) do NOT use the prefix `dvd_`.

On the other hand, any application modules downloaded to the system should start with the prefix `dvd_`.

Application Signalling

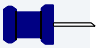
When an application is running, the Player Shell continues to run in “idle mode”. Certain events cause the Player Shell to become active and send signals to applications that are running. These events include certain signals received from the primary input device and Emergency Broadcast System (EBS) messages. Your application must be aware of these events and capable of handling signals from the Player Shell.

If the Player Shell receives an EBS message, it broadcasts a signal to applications that it forked. This allows applications to temporarily suspend their activities while the message is being displayed.

Additionally, some buttons (Power, VDT, VIP) cause the current application to abort. Again, the Player Shell signals the application to inform it of the impending abort to allow it to save its status before exiting.

Three signals are reserved for these warnings:

- SIG_TERMINATE$18
- SIG_SUSPEND$19
- SIG_RESUME$87



Note

Because signals SIG_SUSPEND and SIG_TERMINATE are below $20, they are considered fatal signals. That is, they terminate any blocking I/O operations when received. They do not affect the operation of the asynchronous calls to MPFM and SoftStax.

SIG_TERMINATE

When the user requests the termination of the current application, the SIG_TERMINATE signal is broadcast to certain processes in the system. This signals an application that it has one second to clean up and send any final messages to the server before being terminated by the Player Shell.

Note

When the user presses the VDT, VIP, or POWER buttons while an application is running, it is possible for the Player Shell to present the user with a prompt asking the user to verify the action (typically by pressing the button again). In this case, the Player Shell first broadcasts SIG_SUSPEND and then, if the action was verified, broadcasts SIG_TERMINATE.

SIG_SUSPEND

The `SIG_SUSPEND` signal is broadcast to all processes in the system one second before the Player Shell needs to use the graphics display. This allows applications time to halt any animations. It is not required that all applications immediately cease, but they should be aware that some or all of the graphics overlay is taken over for the display of a system message.

SIG_RESUME

The `SIG_RESUME` signal is broadcast to certain processes in the system immediately after the Player Shell has restored the previous graphics environment. The Player Shell will not destroy any graphics contexts in processes using the DAVID/MAUI interface to the graphics overlay functions, but applications may have to refresh (update) the display.

Additionally, it is recommended to flush any pointer messages still in the queue upon resuming.

Remote Control Key Use

The MAUI API allows multiple processes to share access to the remote control. In other words, some applications, such as the Player Shell, may wish to reserve some specific key presses on the remote, hiding them from other applications.

Specifically, there are several keys that the Player Shell always reserves:

- Power on/off (stand-by)
- Channel up
- Channel down

The action that the Player Shell takes is dependent upon its current operating state.

For More Information

DAVID Utilities and Applications contains detailed information about Player Shell operating states.

Chapter 2: Markets

Two markets for the DAVID product line include:

- **Interactive Television Market**
- **Digital Broadcast Market**

The minimum functional configuration of a DAVID system is called the base case. A base-case model is described in the section entitled:

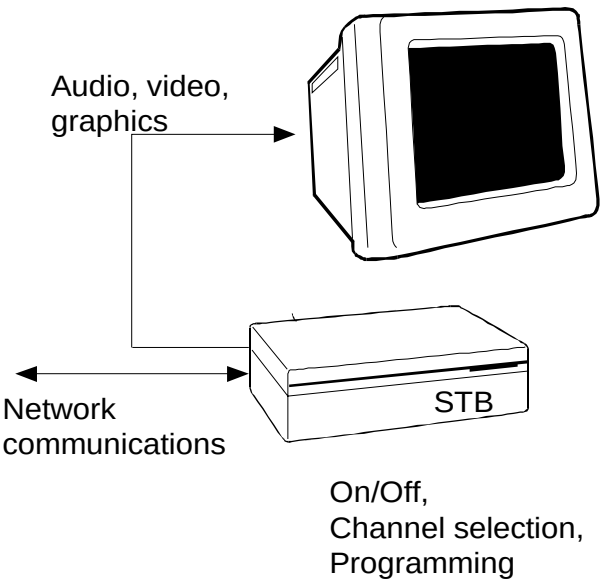
- **Base-Case System**

Interactive Television Market

DAVID services the Interactive Television (ITV) market. A DAVID Set Top Box (STB) connects between a digital or analog/digital network and a television set or computer system. The STB may be housed in a separate chassis that interconnects the TV and the network, or may be contained on a card installed inside a television chassis or computer system.

The STB controls the audio, video, and graphics delivered to the television. The STB also provides for user inputs via at least one primary input device for channel selection and program control. A typical DAVID configuration is shown in **Figure 2-1**.

Figure 2-1 Typical DAVID System Configuration



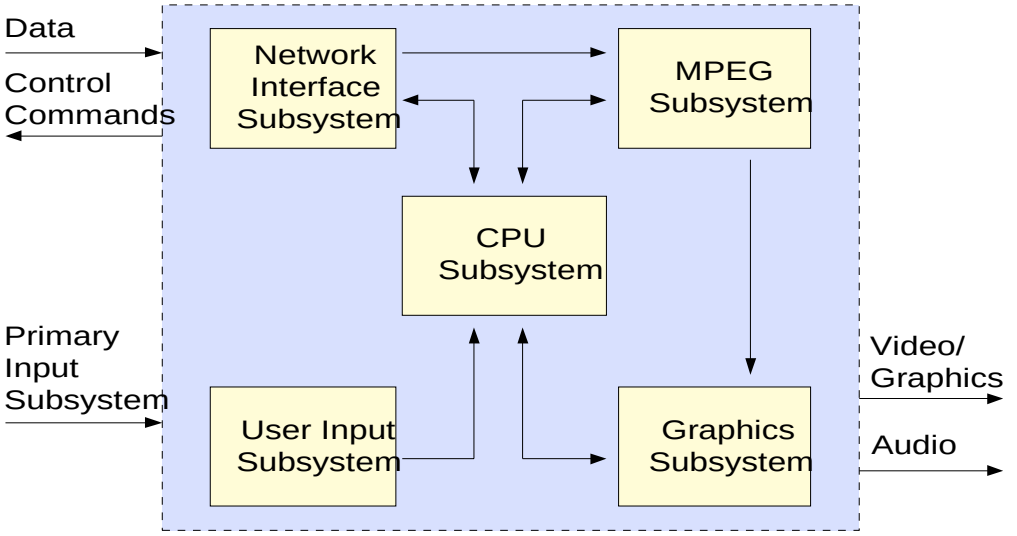
A base-case DAVID STB comprises the following five subsystems:

- CPU subsystem
- Network interface subsystem
- MPEG subsystem

- User input subsystem
- Graphics subsystem

The relationship of the five subsystems in the base-case DAVID STB is shown in **Table 2-2**.

Figure 2-2 DAVID Set Top Box



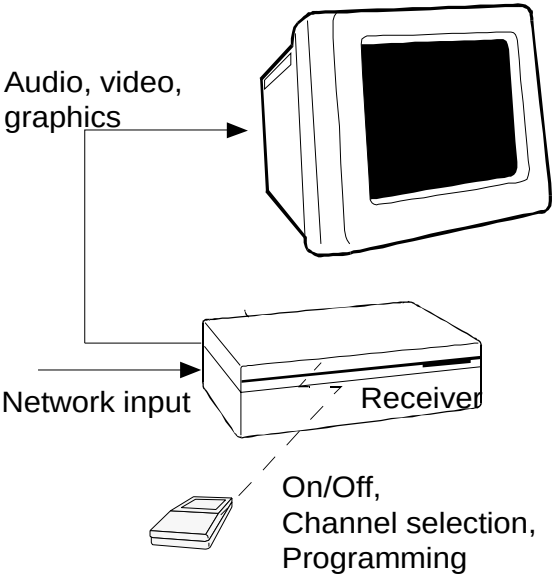
Digital Broadcast Market

DAVID services the Digital Broadcast market.

A DAVID receiver connects between a digital or analog/digital network and a television set or computer system. The receiver may be housed in a separate chassis that interconnects the TV and the network, or it may be contained on a card installed inside a television chassis or computer system.

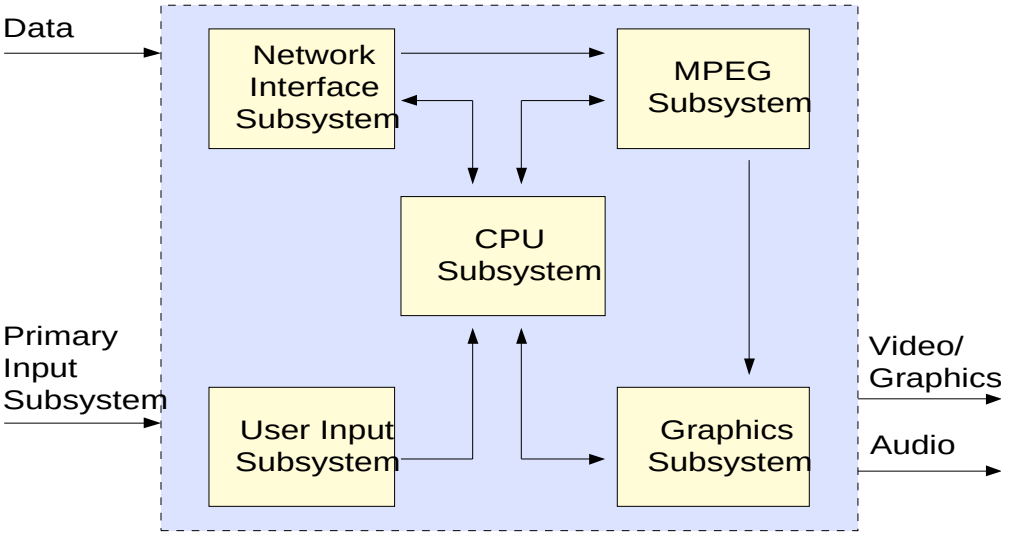
The receiver controls the audio, video, and graphics delivered to the television. The receiver also provides for user inputs via at least one primary input device for channel selection and programming. A typical DAVID configuration is shown in **Figure 2-3**.

Figure 2-3 Typical DAVID System Configuration for Broadcasting



The relationship of the five subsystems in the base-case DAVID receiver is shown in **Figure 2-4**.

Figure 2-4 DAVID Digital Broadcast Receiver



Base-Case System

A base-case DAVID system provides five high-level functions represented by five functional subsystems. In this specification, we make no attempt to distinguish between hardware and software implementations of a subsystem. As long as the minimum functional and performance requirements are met, manufacturers have considerable leeway in implementation options.

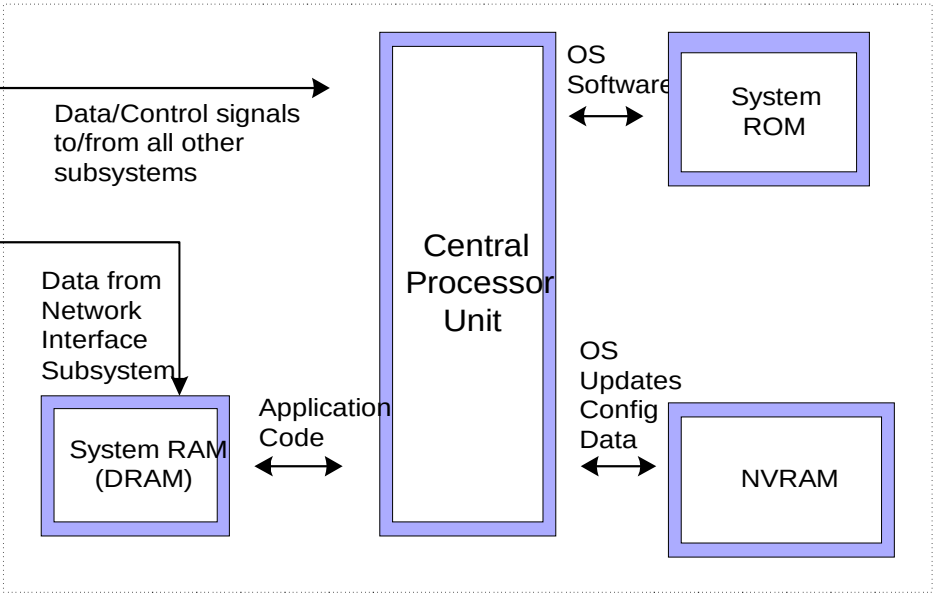
The five subsystems which make up the DAVID base-case include:

- **Core CPU Subsystem**
- **Network Interface Subsystem**
- **MPEG Subsystem**
- **User Input Subsystem**
- **Graphics Subsystem**

Core CPU Subsystem

The CPU subsystem controls the operation of the other four subsystems and contains the DAVID software and operating system. The CPU subsystem comprises the processor, system clock, system RAM, system ROM, and (in an interactive system) non-volatile RAM (NVRAM). **Figure 2-5** shows a simple block diagram of the CPU subsystem.

Figure 2-5 CPU Subsystem



The CPU subsystem includes:

- Central Processor Unit (CPU)
- System ROM
- System RAM (DRAM)
- Non-volatile RAM

The requirements for each of these elements are described in the following paragraphs.

Central Processor Unit (CPU)

DAVID supports the IBM/Motorola PowerPC™ processor family.

System ROM

The System ROM stores the operating system software. This may be implemented in one of two ways:

- In the first case, the ROM holds only a boot loader to download an operating system from the network. The operating system is stored in either RAM or FLASH depending upon the system configuration.

  The advantage of FLASH is its permanence. The download only has to happen once, and if the system needs to reboot, it can do so more quickly the second (or subsequent) time. The advantage of RAM is that it may be cheaper to extend the RAM portion of the STB than to supply FLASH for the operating system. The system code may also execute more quickly from RAM than FLASH, again depending upon the system configuration as well as the number and size of applications embedded in the system.
- In the second case, the entire operating system is stored in ROM initially, while updated modules are stored in FLASH.

  The actual amount of ROM needed is dependent on the choice of CPU and other physical components.

System RAM

Because the operating system uses very little memory for its own use, most system RAM is available for applications to use. The challenge of STB design is to balance the desires of application developers with the economics of consumer electronics. The recommended minimum STB configuration is 2MB of system RAM.

Non-volatile RAM (NVRAM)

Non-volatile RAM stores personalized configuration data such as favorite channel maps and PIN numbers for parental access. The STB should supply at least 4K of NVRAM, that can hold its data without power being connected to the STB for at least one year. The actual amount of NVRAM needed depends on size of the resident application (Player Shell).

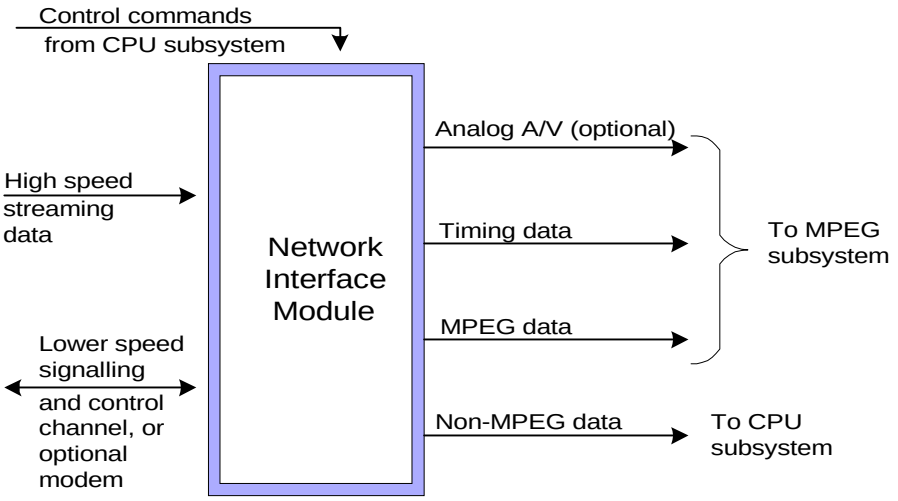
Network Interface Subsystem

The network interface subsystem receives data and control messages from a digital telephone-style interface, an analog/digital broadcast system, or a digital broadcast system. The network interface subsystem routes packets of data to the MPEG and CPU subsystems. The CPU subsystem sends control commands to the network subsystem to control its operation.

The network interface subsystem provides a high-speed data interface for the delivery of analog and digital data to the STB. The network interface subsystem divides MPEG data and non-MPEG data into separate signals. Analog A/V and MPEG data are sent to the MPEG subsystem. Non-MPEG data is sent to system RAM in the CPU subsystem. These operations and

the actual routing of data are controlled by commands sent from the CPU subsystem. **Figure 2-6** shows a block diagram of the network interface subsystem.

Figure 2-6 Network Interface Subsystem



There are several possible configurations for the network interface subsystem depending on the type of network serving the STB. In general, networks can be divided into three categories:

- **Digital Point-to-Point**
- **Analog/Digital Hybrid**
- **Digital Broadcast**

The differences between these categories are explained in the following sections.

Network Architectures

1. Digital Point-to-Point

   Digital point-to-point networks have the unique feature that an STB only receives that data intended for its own use. This network is like a telephone; you do not have to filter out other people's conversations from your own. Examples of this architecture are networks based on ADSL or ISDN technology. This is the simplest network interface to implement, and also the lowest bandwidth; typically 1.5 - 6 Mb/s.

   This architecture is useful in that it allows the data to travel over standard, twisted-pair, copper telephone wire. The STB itself is simpler as well, since it doesn't need to include hardware or software for tuning or conditional access. The STB receives only those signals it has requested and has permission to access.

2. Analog/Digital Hybrid

   Analog/digital hybrid networks require a more complex interface. This type of network is closer to a cable TV system such as CATV, MMDS, and DSS systems. The STB is receiving several channels of video (or other data) simultaneously. A tuner selects one particular analog channel from that broadcast signal. Some of the channels contain data being broadcast to anyone who wants to use it. Other channels deliver a specific program to a specific household. Because everyone in the neighborhood is getting the same set of channels, these channels are specially encrypted to prevent unauthorized access.

   Some of these channels contain traditional analog TV services, just as those used for cable TV. Other channels may carry digital broadcast video services. This complicates the network interface considerably, since it now must contain an analog tuner, a demodulator (for digital services), a conditional-access decryption module, and other circuits required for properly reconstructing the digital data stream.

3. Digital Broadcast

   The fully digital broadcast network is much like the hybrid network, except that all of the channels are digital. This means that “tuning” is not based on selecting a frequency, as in cable television (CATV), but instead on a program ID in the data stream.

   This type of network is called Switched Digital Video (SDV) and it typically delivers Asynchronous Transfer Mode (ATM) cells to the STB. The network interface, in this case, is responsible for the segmentation and re-assembly of cells. Data may still be encrypted to prevent unauthorized “signal theft”, but the tuner and demodulators used in analog systems are no longer necessary.

   The primary distinguishing feature of this architecture is that it typically brings fiber optic cabling to within 100 meters of the STB, in some cases all the way to the home.

Network Interface Components

1. The Data Channel

   Each of the three network architectures carries a high-speed, streaming data channel. The data channel carries virtually all of the application data from the network or servers to the STB. This data stream is formatted as an MPEG-2 (IEC/ISO 13818-1) Transport Stream. This means that the network module output is typically an MPEG-2 Transport Stream (MPTS). MPEG streams are routed directly to the MPEG subsystem.

   Another common configuration is a program containing only application private data; neither audio or video. In this case, the data is routed not to the MPEG decoders, but to system RAM for processing by the application.

   There are no inherent bandwidth requirements for the data channel, except those imposed by the contents of the stream. Movies or video encoded according to the MPEG-2 specification, require a bandwidth greater than 3 Mb/s. Typical networks deliver about 6 Mb/s per program.

2. The Control Channel

   The control channel is another one of the items the three network interfaces (point-to-point, hybrid broadcast, and digital broadcast) have in common. This channel is often a low speed signalling and control channel.

   The lower speed control channel is used for signalling or for carrying commands, requests, events, or small data packets between the STB and the various network components, including the server.

   This channel typically uses a packet-based protocol such as X.25 or UDP/IP. These protocols form the basis for STB addressing.

   There are relatively few constraints placed on the control channel hardware, as most of the issues related to reliable transport and addressing can be handled by software. Again, there are no inherent bandwidth requirements for the control channel, but it is recommended that it provide at least 9600 baud performance.

MPEG Subsystem

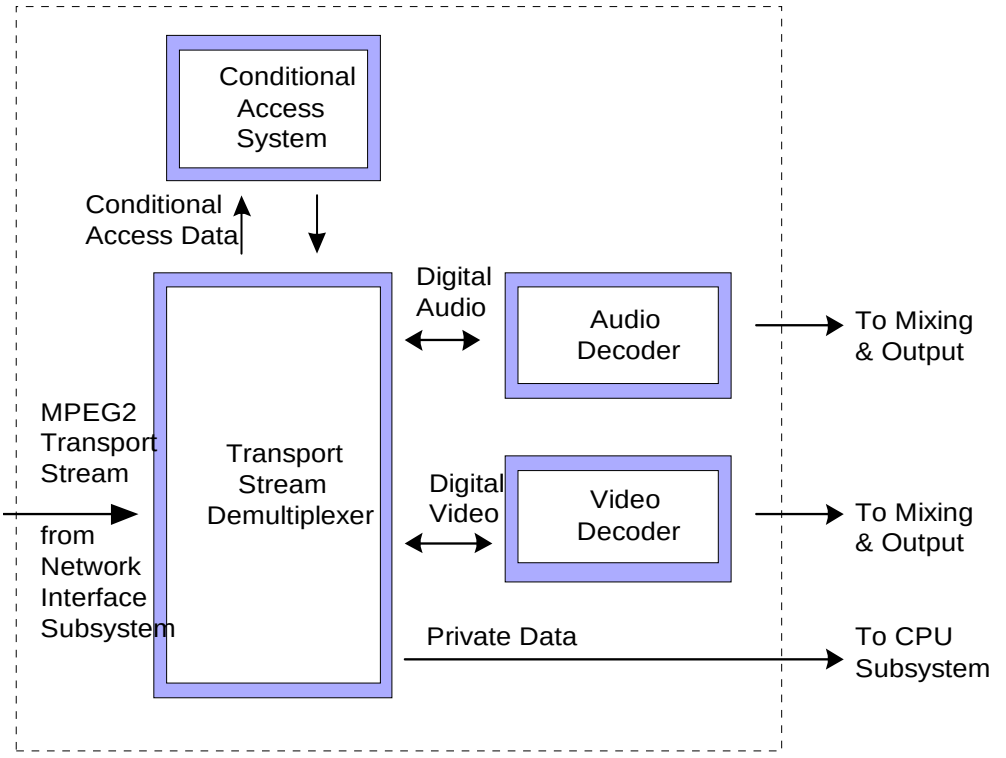
The MPEG subsystem contains MPEG audio and video decoders and the audio attenuation control. The MPEG decoders are capable of decoding MPEG-2 data streams. MPEG-1 data can be decoded, but must be delivered in an MPEG-2 transport stream. MPEG audio includes Musicam Layers 1 and 2, as well as Dolby AC-3.

The MPEG subsystem is composed of four parts:

- **Transport Stream Demultiplexer**
- **Conditional Access System**
- **MPEG Audio Decoder**
- **MPEG Video Decoder**

DAVID requires that these components be capable of decoding MPEG-2 transport streams. **Figure 2-7** shows a simple block diagram of the MPEG subsystem.

Figure 2-7 MPEG Subsystem



Transport Stream Demultiplexer

Data from the network is delivered to the MPEG subsystem through the Transport Stream Demultiplexer (TSD). The TSD routes the incoming stream to its appropriate destination based on instructions from the application.

If the incoming stream is encrypted or scrambled, the TSD first routes encrypted packets to a conditional access system for decrypting. Subsequently, it reroutes the decrypted packets to either the audio decoder, video decoder, or to system RAM.

Conditional Access System

The Conditional Access System (CAS) is optional in a DAVID system as not all network operators send encrypted data streams to the STB. If the streams are encrypted, though, this subsystem is responsible for decrypting that data which the user has authorization to view.

This specification does not place restrictions or requirements on the physical implementation of the CAS, but it is commonly implemented using smart cards. Other types of security micro-controllers may be built into the base unit.

MPEG Audio Decoder

The MPEG audio decoder is responsible for decoding MPEG Layer 1 and Layer 2 (Musicam) audio samples encoded at 32, 44.1 and 48 kHz. It must handle all modes and bit rates specified in the MPEG-2 standard. Additionally, it must comply with recommendations for MPEG-2 audio decoders including Dolby AC-3.

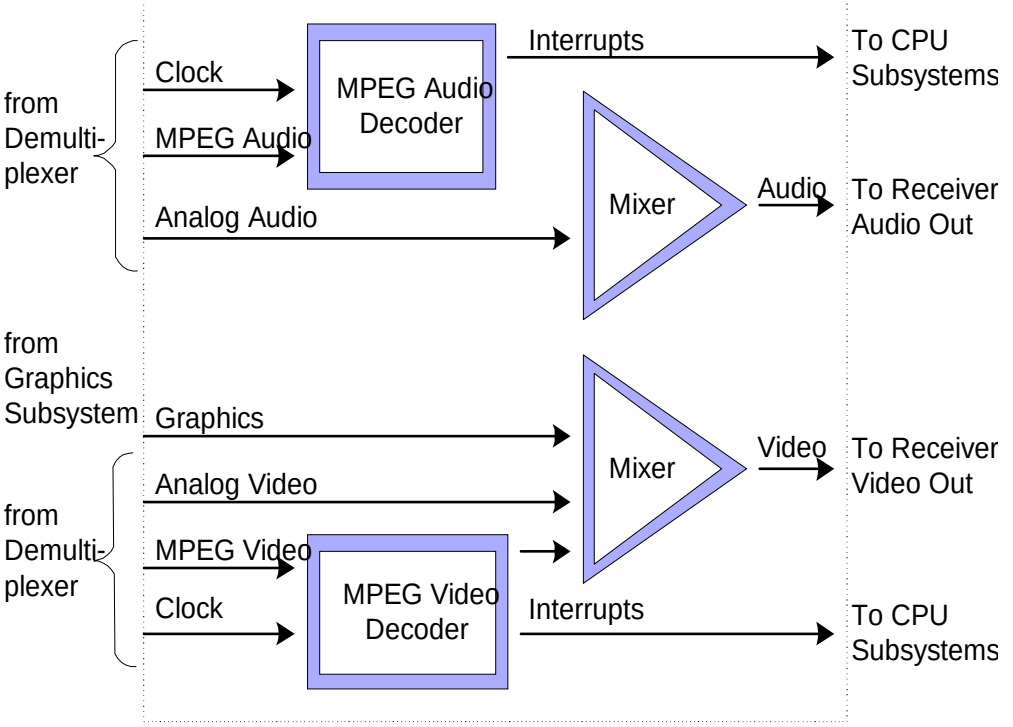
The audio hardware must:

- Generate an interrupt when the decoder is ready for data
- Indicate buffer overflow/underflow conditions to the processor
- Provide an interface for software attenuation control of at least 32 discrete steps of attenuation with smooth transition

Only one audio decoder is required, although additional audio decoders could be present to allow for concurrent decoding of multiple streams.

If a second audio decoder is provided, a mechanism must be provided for controlling attenuation independently for each decoder. The output of the decoders are then summed for output. This mixing may be done either before or after conversion of the audio to analog **Figure 2-8** provides a block diagram showing the MPEG audio and video decoders and mixers.

Figure 2-8 MPEG A/V Decoding and Mixing



MPEG Video Decoder

The MPEG video decoder must be able to decode MPEG-1 and MPEG-2 (Main Level Main Profile) video. MPEG-1 video is typically encoded at a lower resolution than MPEG-2 video. The decoder must assure that either picture is displayed as a full screen image on the TV or monitor. Picture sizes smaller than full-screen are not required to be supported by DAVID STBs.

Table 2-1 MPEG Image Resolutions

	NTSC	PAL
MPEG-1 (SIF)	352x240	384x288
MPEG-2 (CCIR601)	720x488	768x576

MPEG RAM must be sufficient to satisfy the requirements of the MPEG chip manufacturer.

The MPEG video decoder must:

- Accept data at bit rates up to 15 Mb/s
- Provide interrupts on key events, such as encountering the beginning or end of a stream, or when each I-frame is decoded on error or when video stream parameters change

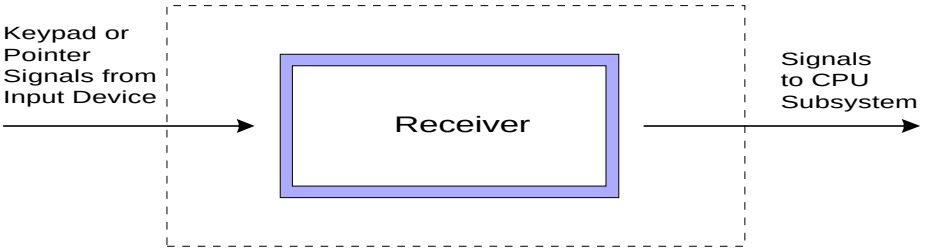
The video processor is responsible for decoding only one video stream at a given time, although it must be able to switch to a new stream for decoding on demand.

User Input Subsystem

The user input subsystem receives signals from the user input device(s) and routes the signals to the CPU. Typical user input devices are an infrared remote control and buttons on the STB. Extended remotes could include tethered game controllers or even keyboards.

A DAVID STB, by default, has one user input device. The input device may be a pointing device such as a remote control or joystick, or a keyed device such as a gamepad or keyboard.

Figure 2-9 User Input Subsystem



Primary Input Device

The simplest input device, such as a remote control, may have only a few keys or buttons. However, this number may vary widely from one product to another. The minimum set includes ten numeric keys (0 through 9) channel up/down keys, four directional “arrow” keys, a “select” key, and a power on/off key.

A base-case remote must be able to generate at least 10 messages per second.

Front Panel Buttons

In addition to the primary input device, some STBs include an additional set of buttons on the front panel of the STB. These buttons are not required by the DAVID architecture. If they are provided, they must be seen by the application as if they were standard remote control key presses.

Optional Buttons

Optional or network operator-dependent buttons on the remote control or the front panel of the STB are VIP, VDT, MENU, and TV/ITV.

Graphics Subsystem

The graphics subsystem enables the STB to display text and graphics as overlays on the analog or digital video. This subsystem is responsible for composing the image data and outputting it to the NTSC/PAL video encoder.

The DAVID system architecture requires the ability to display text and/or images over the full motion video plane (analog and digital). There are many options for the implementation of these graphics display planes.

Image Resolution

Because the MPEG video plane operates at CCIR601 resolution (720x480), it is recommended that the graphics operate at this resolution as well. There are, however, two caveats to this recommendation. First, there is often a requirement in graphics applications for **square** pixels. This implies a resolution of 640x480 (4:3 ratio). Secondly, there are also requirements for a more memory-efficient coding resolution, such as 360x240 or 320x240.

Because these are very valid requirements, the DAVID architecture is quite flexible in this regard. In general, DAVID requires that the system be capable of operating in a high resolution mode (minimum 640x480). Lower resolution modes are optional.

The higher resolution mode offers the advantages of quality. The lower resolution offers images that are faster to download and take up less room in memory. The lower memory footprint means that more images may be stored locally in RAM, which in turn, increases the performance or responsiveness of the application.

Image Coding Methods

Various standards bodies, such as DAVIC, are addressing the issues of minimum image quality and coding formats. Currently, DAVIC mandates support of 4-bit/pixel (Color Look Up Table) CLUT images, where the CLUT references 24-bit RGB values. Similarly, DAVID requires the STB to support at least 4-bit/pixel (16-color), CLUT images. Use of greater color depth image formats is optional.

Transparency Support

In order to properly display text and graphics over video, applications need a mechanism for specifying which areas of the display are **opaque** (show graphics) and which are **transparent** (show the video). The transparent and opaque areas must be definable based on either a color key or a region. The hardware must be able to support opaque and transparent modes as applications may use either.

In addition to strict transparency, there is often a need to define a level of **translucency** on a pixel basis. This translucency is most often used to anti-alias text or diagonal lines against the video. The translucency factor creates the impression of a blending of colors such that the edges of the graphics are smoother.

The hardware must support a minimum of 8 levels of translucency.

Chapter 3: Packages and Components

The following table identifies the components included with DAVID systems.

Table 3-1 Component Applicability to Markets

Component	DAVID	DAVID for Broadcast Only
OS-9	Required	Required
MPFM	Required	Required
DUXMan	Hardware Dependent	Hardware Dependent
MAUI Profile 1	Required	
MAUI Profile 2		Required
MAUI Profile 3	Optional	
SPF	Required	Required
LAN Communications Pak	Required	
NRF	Optional	
FLASH	Hardware Dependent	Hardware Dependent

Table 3-1 Component Applicability to Markets

Component	DAVID	DAVID for Broadcast Only
SPI	Hardware Dependent	Hardware Dependent
UpLink (Client Side)	Optional	Optional
Sample Applications/Utilities	To be used as example code	To be used as example code
UpLink Mini-Server[a]	Optional	

a. The UpLink Mini-Server is a separate product which must be purchased separately.

Chapter 4: Component Descriptions

DAVID is a very modular and components can be added as needed for your specific application. The components included in DAVID are:

- **OS-9**
- **MPFM**
- **DUXMan**
- **MAUI**
- **SoftStax**
- **LAN Communications Pak**
- **NRF**
- **FLASH**
- **SPI**
- **UpLink (Client Side)**
- **UpLink Mini-Server (Windows NT)**

In this chapter we have also included a section entitled:

- **Sample Applications and Utilities**

OS-9

OS-9 is a powerful and versatile operating system that can help you fully use your system's capabilities. OS-9 offers a wide selection of functions because it was designed to serve the needs of a broad audience. Whether you are a casual user or a professional programmer, you will find many useful features in OS-9.

OS-9 is designed to provide a friendly software interface for personal computers, educational systems, and the professional programmer. The OS-9 package includes utility programs.

MPFM

The Motion Picture File Manager (MPFM) is a software extension to OS-9. Applications use MPFM to manipulate MPEG audio and video decoders. This software extension is implemented using the OS-9 file manager and driver structures.

The Motion Picture File Manager provides a set of functions that enables your application to play MPEG-1 and MPEG-2 movies. MPEG movies are delivered as audio/video synchronized data streams across a network. MPFM also provides the means to play MPEG audio files or video files that are not synchronized.

MPEG-1 and MPEG-2 files require different bandwidths. An MPEG-1 stream is delivered at the typical CD bandwidth of 1.4 Mb/second. MPEG-2 is typically delivered at bandwidths from 3 Mb/second and above. MPEG-1 decoders are limited to playing only MPEG-1 data. MPEG-2 decoders can handle both MPEG-1 or MPEG-2 data, however, MPEG-1 elementary streams must be wrapped with MPEG-2 system layer in order to be played in a system that employs a demultiplexing chip.

DUXMan

The Demultiplexer Manager (DUXMan) is a device manager — a special device driver having no associated file manager. A device manager performs functions on behalf of other drivers in the system.

DUXMan manages the Transport Demultiplexer Integrated Circuit (TDIC) in DAVID hardware devices.

DUXMan functions are called by other device drivers and file managers such as the Stacked Protocol File manager (SPF) and the Motion Picture File Manager (MPFM). DUXMan functions control:

- MPEG-2 transport stream demultiplexing
- MPEG stream system layer parsing
- Program Specific Information (PSI) extraction
- Delivery of audio/video/private data

DUXMan is not directly called by an application program.

MAUI

The Multimedia Application User Interface (MAUI) is an Application Programming Interface (API) for the Digital Audio/Video Interactive Decoder (DAVID) environment. MAUI provides an extensive set of low-level graphical and communications services that can be used in interactive television decoders connected to telephone, cable, and wireless networks.

MAUI features include:

- Graphics processor independence to facilitate application portability.
- A high speed drawing engine that is modular (for flexibility) and compact (for embedded application environments).
- Windowing, clipping, and inking support.
- Sound processor independence to facilitate application portability.
- Support for authoring system tools for building applications and their associated data structures. By using MAUI, developers can avoid having to modify their runtime engines for each server type, and as a result, decrease development time.

MAUI Profiles

Different devices require different memory footprints. To address that, MAUI is shipped in three different profiles:

- **Complete MAUI Profile Including Windowing**
- **MAUI Profile for DAVID**
- **MAUI Profile for DAVID Broadcast Only**

There are two fundamental differences between the various profiles. The first is the memory footprint of the trap handler module that is installed in ROM or FLASH on the device. This is adjusted by moving code from the trap module into the compile-time library. The second difference is to totally remove certain functionality.

Complete MAUI Profile Including Windowing

This profile contains all of MAUI in the trap handler.

MAUI Profile for DAVID

This profile contains all of MAUI with the exception of Windowing in the trap handler.

MAUI Profile for DAVID Broadcast Only

This profile contains only a few libraries in the trap handler. Most of the remaining APIs are available in the compile-time library.

In addition, the only mixing modes available are `BLT_MIX_REPLACE`, `BLT_MIX_SXD`, and `BLT_MIX_RWT`.

Table 4-1 MAUI Profile Summary

	Complete MAUI		DAVID		DAVID Broadcast Only	
API	**Trap**	**Lib**	**Trap**	**Lib**	**Trap**	**Lib**
MAUI System	•			•		•
Shaded Memory	•		•		•	
CDB	•		•			•
Graphics Device	•		•		•	
Bit-BLT	•		•			•
Drawing	•		•			•

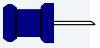
Table 4-1 MAUI Profile Summary

	Complete MAUI		DAVID		DAVID Broadcast Only	
API	**Trap**	**Lib**	**Trap**	**Lib**	**Trap**	**Lib**
Text	•		•			•
Animation	•		•			•
Messaging	•		•			•
Input	•		•			
Windowing	•			•		

SoftStax

SoftStax consists of an application programming interface (ITEM), the network infrastructure (SPF), a template driver (`spproto`) that can be used to create your own protocol drivers, a loopback driver (`sploop`) which emulates connection-oriented or connectionless networks for application and protocol testing, source and objects for a variety of HDLC controllers and Serial Communications Controllers (SCCs) for integrated microprocessors, and example applications that use the system in a variety of ways. This package forms the basis of the networking environment. For development, you need a minimum of the SoftStax (SPF) Base Pak and FasTrak. For testing, you need the OS-9 objects available in the OS-9 Development Kit for the desired target processor.

In addition to SPF itself, there is also the SPF Real-Time Network Driver (RTND). The RTND is responsible for gathering data from DUXMan for use by other APIs regarding channel management, conditional access, and EPGs.



Note

SoftStax was formerly entitled SPF Base Pak.

LAN Communications Pak

The LAN Communications Pak provides a small footprint, limited-functionality Internet package that enables small embedded devices to communicate in a network environment. The User Datagram Protocol (UDP), Transmission Control Protocol (TCP), and Internet Protocol (IP) transport protocols are supported in this package.

NRF

NRF (Non-volatile RAM File Manager) manages a random access file system stored on non-volatile RAM (NVRAM). NVRAM is a Random Access Memory (RAM) device that holds its data without primary power because it is battery-backed. NRF is available as a Dual Ported Input/ Output (DPIO) subsystem for OS-9.

NVRAM is a limited and precious resource for DAVID systems because they usually only have about 4-8k of NVRAM. NRF is specifically customized to use as little of this memory as possible for file system overhead by providing a flat file system using a link list of files and by using smaller and more compact file descriptors than what is found in the Random Block File Manager (RBF). Fragmentation in an NRF file system is zero.

NRF has a very important feature. If a system crash occurs during a file update, only the file being updated is lost. The rest of the file system remains intact.

The Application Programming Interface (API) for NRF is RBF compatible, allowing for easy portability between RBF disk applications and NRF. The following RBF utilities work with NRF files: `dir`, `list`, `copy`, `attr`, `del`, `nfree`, and `dump`.

NRF, like all OS-9 I/O systems, has four primary software blocks:

- Application Interface Library.
- File Manager
- Device Driver
- Device Descriptor

Applications make calls through the interface library, which are passed by the kernel down to the file manager. The file manager handles the call and if necessary, calls the driver. Both the file manager and the driver use the device descriptor during system initialization to determine the specific configuration details of the hardware.

Since NRF is RBF compatible at the application layer, the Application Interface Library for NRF is `os_lib.l,` a standard Ultra C library.

FLASH

FLASH is a type of non-volatile memory that can be erased, typically in sectors, and reprogrammed.

The advantage of FLASH is its permanence. A download only has to happen once, and if the system needs to reboot, it can do so much more quickly the second (or subsequent) time.

SPI

SPI is a low-level driver used to communicate with devices on an SPI bus. SPI is unknown to application programs. It is only accessed by other drivers that are written to control devices on an SPI bus, it is only of concern to the OEM when porting DAVID to their consumer device.

UpLink (Client Side)

UpLink is a command protocol application programming interface (API) that resides between DAVID set top box applications and the network. UpLink is used to establish network communications, request data from a server, control server data flow, and manage communications. UpLink provides a common interface layer that hides server-specific details from the application. In this way, applications can be written that are independent of the various servers or networks supporting them. Applications can communicate predictably and transparently with any network or server structure.

UpLink Mini-Server (Windows NT)

The UpLink Mini-Server for Windows NT™ is used to pump MPEG streams to a DAVID device as well as get and store private data from a DAVID device. This is accomplished through the UpLink interface on the device. (See **UpLink (Client Side)** on page 57.)

Sample Applications and Utilities

The sample applications and utilities listed below are described in more detail in the ***DAVID Utilities and Applications*** manual.

Player Shell

The player shell is the first process to execute in a DAVID system. It is supplied with the operating system software to perform basic system initialization, channel tuning, and downloading interactive applications from the network.

systrap Module Verification

The `systrap` functions are contained in a system state trap handler. The functions allocate memory for modules, and link and unlink modules.

fptest Utility

The `fptest` utility tests the front panel functionality of the Hellcat set-top box.

fwrite Utility

The `fwrite` utility is similar to the `pmod` utility except that it is usually used to write images into flash rather than single modules.

The `fwrite` utility must be loaded into RAM before it can be used. The Hellcat development system must be booted using the `<lr>` option.

pmod Utility

pmod makes individual module or multiple module changes to flash on a Hellcat development system. To use pmod the Hellcat must be booted using the <lr> option.

syscfg Utility

The syscfg utility is used to change system parameters associated with the the sysgo module as it is used with the Hellcat set-top box (STB).

The sysgo module for this version of DAVID is derived from the sysgo example provided in the ***OS-9 Technical Manual***.

xdmod Utility

Xdmod is a bi-directional command used to pack a file in a data module or remove the contents of a data module to a file. You must specify a file name and a module name.

Index

A

B

C

D

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From:_____

Company:_____

Phone:_____

Fax:_____Email:_____

Product Name: DAVID

Description of Problem:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Host Platform_____

Target Platform_____

RadiSys.

MICROWARE SOFTWARE