The pin assignments for the 96-pin Euro-DIN NuBus accessory card connectors in the NuBus interface are shown in Table 2-17.

Table 2-17 NuBus pin assignments

Pin	Name	Pin	Name	Pin	Name
a1	–12V	b1	–12V	c1	/RESET*
a2	SB0	b2	GND	c2	GND
a3	/SPV	b3	GND	c3	+5V
a4	/SP	b4	+5V	c4	+5V
a5	/TM1	b5	+5V	c5	/TM0
a6	/AD1	b6	+5V	c6	/AD0
a7	/AD3	b7	+5V	c7	/AD2
a8	/AD5	b8	/TM02	c8	/AD4
a9	/AD7	b9	/CM0	c9	/AD6
a10	/AD9	b10	/CM1	c10	/AD8
a11	/AD11	b11	/CM2	c11	/AD10
a12	/AD13	b12	GND	c12	/AD12
a13	/AD15	b13	GND	c13	/AD14
a14	/AD17	b14	GND	c14	/AD16
a15	/AD19	b15	GND	c15	/AD18
a16	/AD21	b16	GND	c16	/AD20
a17	/AD23	b17	GND	c17	/AD22
a18	/AD25	b18	GND	c18	/AD24
a19	/AD27	b19	GND	c19	/AD26
a20	/AD29	b20	GND	c20	/AD28
a21	/AD31	b21	GND	c21	/AD30
a22	GND	b22	GND	c22	GND
a23	GND	b23	GND	c23	/PFW
a24	/ARB1	b24	/CLK2X	c24	/ARB0
a25	/ARB3	b25	STDBYPWR†	c25	/ARB2
a26	/GA1	b26	/CLK2XEN	c26	/GA0
a27	/GA3	b27	/CBUSY	c27	/GA2
a28	/ACK	b28	+5V	c28	/START

continued

Table 2-17 NuBus pin assignments (continued)

Pin	Name	Pin	Name	Pin	Name
a29	+5V	b29	+5V	c29	+5V
a30	/RQST	b30	GND	c30	+5V
a31	/NMRQx	b31	GND	c31	GND
a32	+12V	b32	+12V	c32	/CLK

* A slash before a signal name indicates that it is in the low state when active.
† Trickle +5 V supply.

The power available and maximum capacitance loading for each expansion card are shown in Table 2-18.

Table 2-18 Power budget for each slot card

Voltage (V)	Maximum current (A)	Maximum capacitance (µF)
+5	2.0	1513
+12	0.175	536
–12	0.15	698

Digital Audio/Video Expansion Connector

In the Macintosh Quadra 840AV, a digital audio/video (DAV) expansion connector is mounted on the main circuit board in line with NuBus slot address $C (the slot nearest the center of the computer), to let an accessory card access sound and video data directly. In the Macintosh Centris 660AV, the DAV connector is mounted on the optional NuBus adapter card (shown in Figure 2-13). Both models can accept a short NuBus accessory card that accesses the DAV connector; the Macintosh Quadra 840AV can also accept a long card.

Figure 2-14 illustrates the lower-right portion of a standard short or long NuBus card that has a connector added to plug into the DAV connector. It shows the mechanical relation between the DAV connector and the normal NuBus connector, with dimensions given in inches.

The DAV connector provides access to the system's 4:2:2 unscaled YUV video input signal and to the digital audio signal input for the Singer codec. One use for this feature is to provide a hardware audio or video compression capability on an accessory card, which could write out compressed data to NuBus. The DAV connector is a 40-pin type, model KEL 8801-40-170L. Table 2-19 gives its pin assignments.

Figure 2-14 DAV connection on a NuBus card

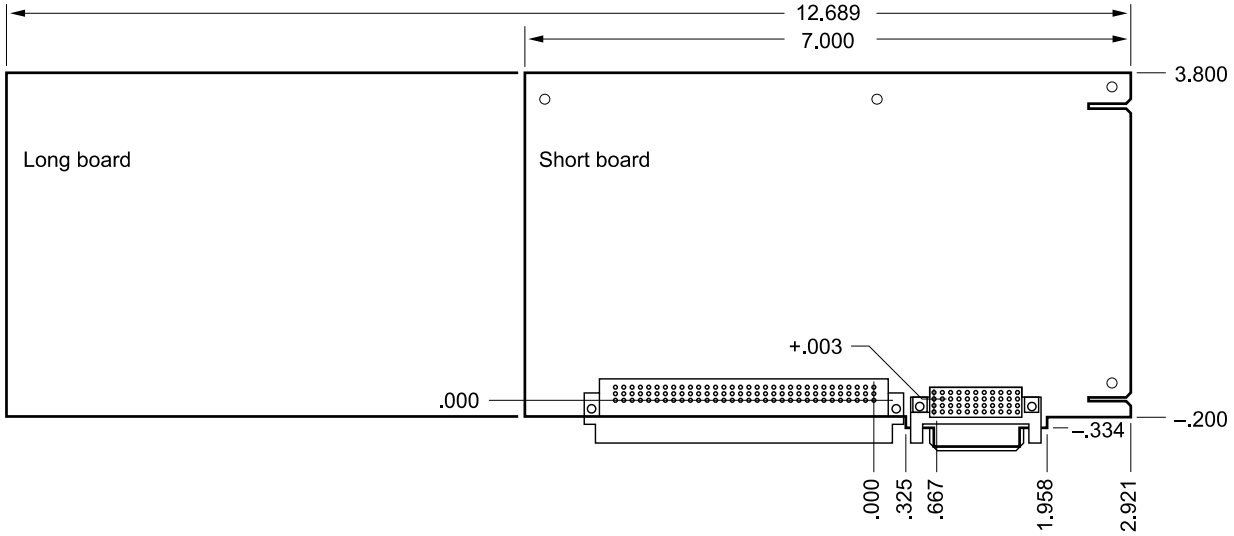


Table 2-19 DAV connector pin assignments

Pin	Signal	Pin	Signal	Pin	Signal
1	Y bit 7	15	Y bit 0	29	UV bit 1
2	LLClk	16	Ground	30	NC (reserved)
3	Y bit 6	17	UV bit 7	31	UV bit 0
4	Ground	18	FEI~	32	Ground
5	Y bit 5	19	UV bit 6	33	SingerSync
6	VS	20	Ground	34	Ground
7	Y bit 4	21	UV bit 5	35	SingerSerOut
8	Ground	22	iicSDA	36	SingerBitClk
9	Y bit 3	23	UV bit 4	37	SingerSerIn
10	HRef	24	Ground	38	Ground
11	Y bit 2	25	UV bit 3	39	Ground
12	Ground	26	iicSCL	40	SingerMClk
13	Y bit 1	27	UV bit 2		
14	vdcCRef	28	Ground		

DAV Sound Interface

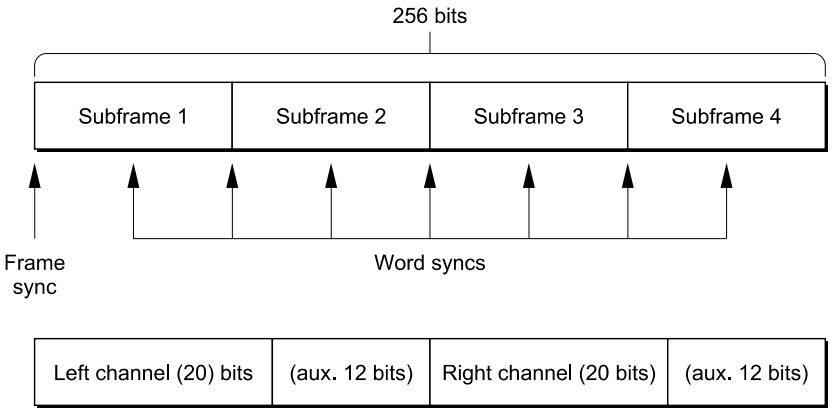
The Singer sound codec uses time-division multiplexing to transfer multiple audio channels between the DAV connector, the Singer chip, and the PSC for DMA transfers to and from RAM memory. The sound signals that appear at the DAV connector are listed in Table 2-20. These signals have a minimum setup time of 10 ns and a minimum hold time of 8 ns; they can tolerate a maximum load of 20 pF.

Table 2-20 DAV connector sound signals

Signal	Description
singerMClk	24.576 MHz master clock
singerBitClk	Bit clock that clocks serial data on singerSerOut and singerSerIn; 256 times the sample rate; also used to clock singerSync
singerSync	Signal that marks the beginning of a frame and a word
singerSerOut	Sound output from PSC to DAV connector
singerSerIn	Sound input from DAV connector to PSC

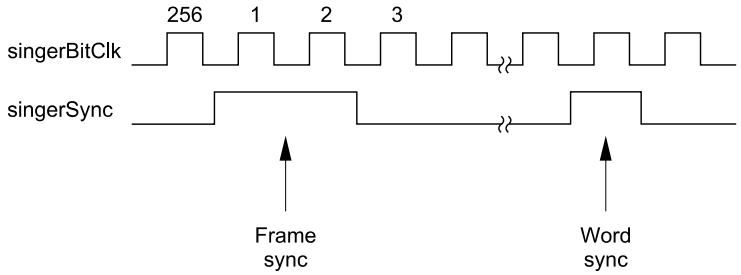
The Singer codec transfers data in 256-bit frames, each of which contains four subframes of 64 bits each. Each subframe carries two 32-bit audio samples, one left and one right. Each sample contains 20 data bits and 12 auxiliary bits. Subframe 1 is reserved for the Macintosh system sound I/O; the other subframes are available for applications and accessory cards to use. The Singer frame structure is shown in Figure 2-15

Figure 2-15 Singer sound frame



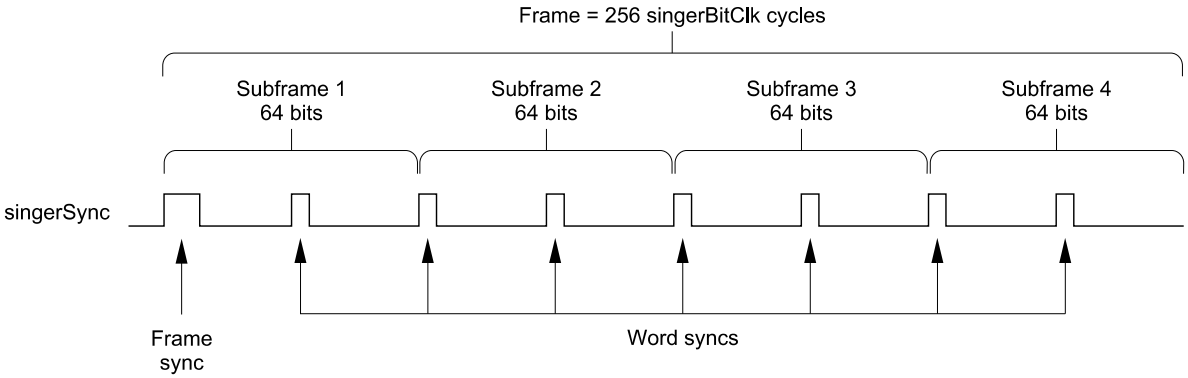
The signals singerSync, singerSerOut, and singerSerIn are clocked by the singerBitClk signal. The falling edge of the clock is used to clock the signals, and the rising edge is used to sample them. As shown in Figure 2-16, a frame sync is marked by a pulse two singerBitClk cycles wide; a word sync is marked by a pulse one singerSync cycle wide.

Figure 2-16 Sound frame and word synchronization



The singerSync synchronization signals for each subframe are shown in Figure 2-17.

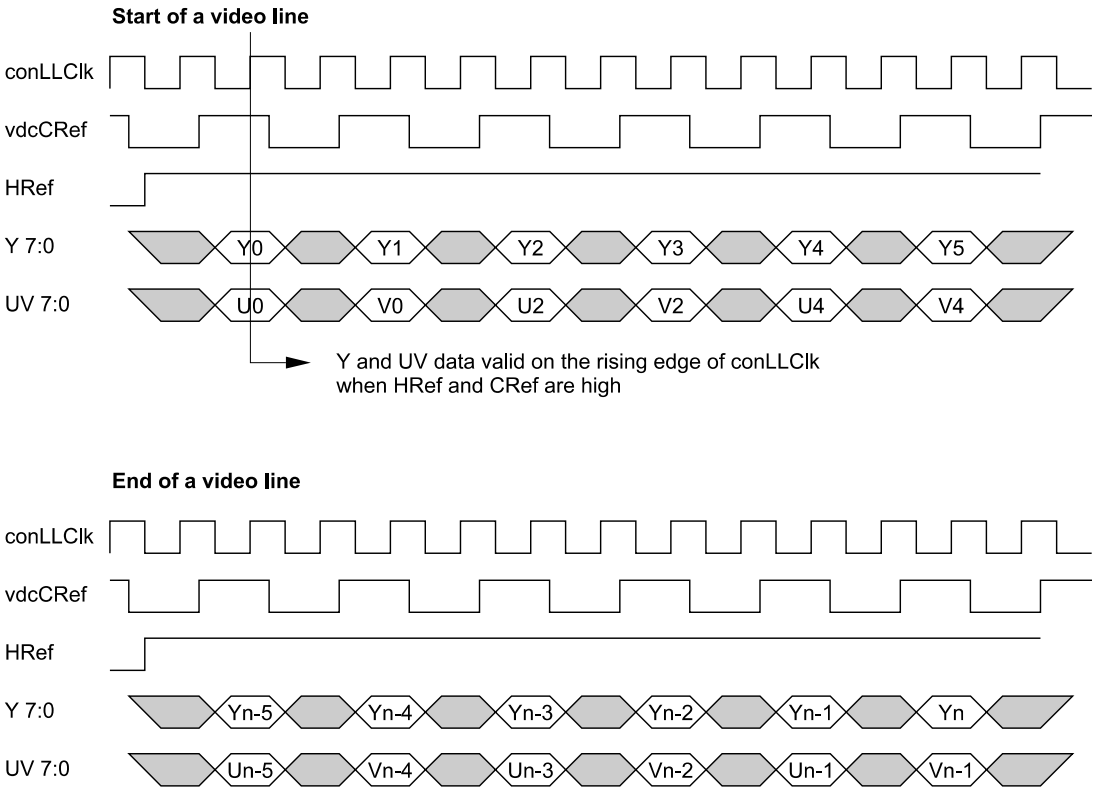
Figure 2-17 Sound subframe synchronization



DAV Video Interface

At the DAV connector, the digital video signal data format conforms to CCIR Specification 601 and is compatible with most video compression chips. In the DAV interface, video lines are defined by the HRef signal; it goes high during the image transmission and low during the blanking interval. The DAV video signal timing relations are shown in Figure 2-18.

Figure 2-18 DAV video timing



Processor-Direct Cards for the Macintosh Centris 660AV

The Macintosh Centris 660AV (but not the Macintosh Quadra 840AV) can accept an accessory card that plugs directly into the main circuit board instead of into the adapter card shown in Figure 2-13. An accessory card plugged into the main circuit board can gain access to the processor as well as to the DAV bus. The resulting processor-direct slot (PDS) capability is similar to that of the Macintosh Centris 610 computer, described in the *Macintosh Centris 610 Developer Note*.

The Macintosh Centris 610 computer uses an AMP type 650231-5 connector for PDS cards; the Macintosh Centris 660AV uses an AMP type 650231-3 connector. Because the corresponding pins are aligned, it is possible to design PDS cards that work on both models.

The Macintosh Centris 660AV PDS Connector

The pin assignments for the Macintosh Centris 660AV PDS connector are given in Table 2-21. Pin numbers preceded by an asterisk have signals that are different from those in the Macintosh Centris 610. Pin numbers preceded by a minus sign are not used by the Macintosh Centris 660AV.

Table 2-21 Macintosh Centris 660AV PDS connector pin assignments

Pin number	Signal name	Pin number	Signal name
1	GND	26	+5 V
2	A(1)	27	D(19)
3	A(3)	28	D(17)
4	A(4)	29	GND
5	A(6)	30	D(14)
6	A(7)	31	D(13)
7	A(9)	32	D(11)
8	A(11)	33	D(9)
9	A(13)	34	D(8)
10	A(15)	35	D(6)
11	GND	36	D(4)
12	A(18)	37	+5 V
13	A(19)	38	D(1)
14	A(21)	39	GND
15	A(23)	40	SIZE(1)
16	A(24)	41	RW
17	A(26)	–42	/TIP.CPU*
18	A(29)	*43	GND†
19	A(31)	44	/TEA
20	D(31)	*45	/NC
21	D(29)	*46	GND
22	D(27)	47	/TRST
23	D(25)	–48	/CI.OUT
24	D(24)	49	GND
25	D(22)	*50	NC

continued

Table 2-21 Macintosh Centris 660AV PDS connector pin assignments (continued)

Pin number	Signal name	Pin number	Signal name
–51	/BR.40SLOT	82	A(17)
52	/BB	83	+5 V
53	/LOCK	84	A(20)
54	/MEM.RESET	85	A(22)
55	/CPURESETOUT	86	GND
56	+5 V	87	A(25)
*57	040INPROGRESS	88	A(27)
58	/NMRQ(6)	89	A(28)
59	GND	90	A(30)
60	/IPL(0)	91	D(30)
61	/IPL(1)	92	D(28)
62	/IPL(2)	93	D(26)
63	–12V	94	GND
64	GND	95	D(23)
*65	NC	96	D(21)
*66	NC	97	D(20)
*67	/NMRQ(5)	98	D(18)
*68	/NMRQ(4)	99	D(16)
*69	/040LOCKE	100	D(15)
70	+5 V	101	+5 V
71	AUX.CPUCLK	102	D(12)
72	A(0)	103	D(10)
73	A(2)	104	GND
74	+5 V	105	D(7)
75	A(5)	106	D(5)
76	GND	107	D(3)
77	A(8)	108	D(2)
78	A(10)	109	D(0)
79	A(12)	110	SIZE(0)
80	A(14)	111	+5 V
81	A(16)	*112	+5 V

continued

Table 2-21 Macintosh Centris 660AV PDS connector pin assignments (continued)

Pin number	Signal name	Pin number	Signal name
113	/TA	127	/SYS.RESET
114	GND	–128	TM(0)
115	/TS	–129	TM(1)
*116	GND	–130	TM(2)
*117	+5 V	131	+5 V
*118	+5 V	*132	NC
–119	/BG.40SLOT	133	+12V
120	/BG.CPU	134	GND
121	+5 V	135	BS.CLK
122	TT(0)	136	/BS.MODE
123	TT(1)	*137	MUNI/RQ
124	GND	*138	NC
–125	TLN(0)	139	reserved
–126	TLN(1)	140	+5 V

* A slash before a signal name indicates that it is in the low state when active.
† GND on pin 43 identifies the Macintosh Centris 660AV; on the Macintosh Centris 610 pin 43 is not connected.

Most of the signals listed in Table 2-21 are connected directly to the computer's processor. Table 3-22 lists the PDS signals that are connected to the computer's processor but that should not be connected to a processor on a PDS card. Table 3-23 lists the PDS signals that are not directly connected to the computer's processor.

Table 3-22 Restricted microprocessor signals on the PDS connector

Signal name	Direction	Function
/IPL(0–2)	I*	Interrupt priority lines from the PSC; not to be used as wire-OR lines; can be monitored by a PDS card
/TIP.CPU	I	From the MC68040 on the main circuit board; not connected to any other part of the computer

*I indicates input to the PDS card.

Observe the following additional cautions when designing PDS cards for the Macintosh Centris 660AV:

Table 3-23 Nonmicroprocessor signals on the PDS connector

Signal name	Direction	Function
/SYS.RESET	I/O*	Enables PDS to drive system reset signal; used only for testing
AUX.CPU.CLK	I	Buffered version of main processor's bus clock (BClk)
/BG.CPU	O	Bus grant for main processor
/BG.40SLOT	I	Bus grant for PDS card
/BR.40SLOT	O	Bus request for PDS card
/MEM.RESET	I	Fast reset generated for memory controller IC
/MI.SLOT	O	Memory inhibit from PDS card to memory controller IC
/NMRQ(6)	O	NuBus slot $E interrupt; also connected to NuBus slot $E

* I indicates input to the PDS card; O indicates output from the PDS card.

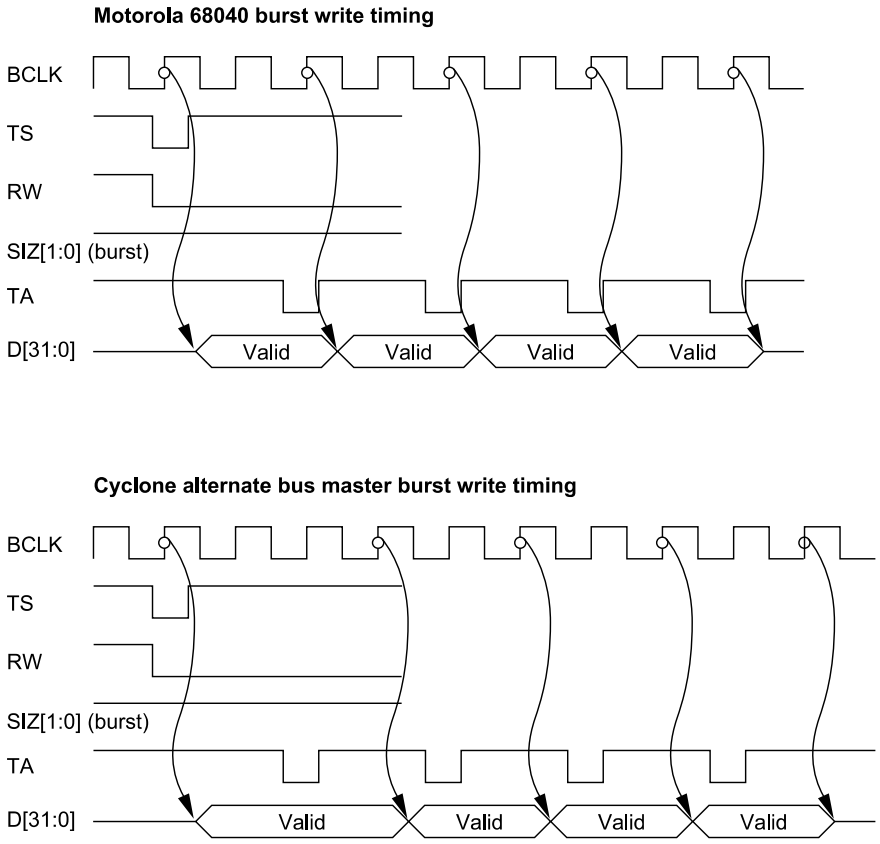
- Most signals on the PDS connector are connected directly to the main processor with no buffers. Therefore, the PDS card must present capacitive loads of not more than 40 pF on the address, data, and clock lines and not more than 20 pF on the control lines.
- The AUX.CPUCLK line (pin 71) is terminated with a series resistor. To reduce reflections on this line, all loads on the card should be lumped.
- /DLE (pin 45) is not connected because DLE-type read actions are not supported by some Macintosh Centris 660AV bus masters.
- The Macintosh Centris 660AV does not support snooping; pins 46 and 116 are grounded.
- /BR.CPU (pin 50) is not connected because the Macintosh Centris 660AV system bus arbiter always grants bus control to the microprocessor when there are no other high-priority bus requests.
- The Macintosh Centris 660AV does not support /BG.40SLOT (pin 119) and /BR.40SLOT (pin 51) because it does not support using an accessory card as a bus master in addition to the existing bus masters (the processor, the DSP, the PSC, and the MUNI).
- /TBI (pin 112) is connected to +5 V because the TBI signal is not allowed by some bus masters.
- /PDS.SLOT.E.EN (pin 132) is not connected because the MUNI is programmed to decode or ignore individual slots.

The 040INPROGRESS signal (pin 57) is high when the main processor is the bus master. When this signal is low, a different bus master can use the alternate burst write timing protocol described in the next section.

Processor Bus Burst Write Timing

The Macintosh Centris 660AV computer's processor bus supports two different timing protocols for burst write actions. When pin 57 of the PDS connector is high, the main processor is bus master and burst write actions must use the timing shown in the top half of Figure 2-19. When pin 57 of the PDS connector is low, an alternate bus master may perform burst write actions using the timing shown in the bottom half of Figure 2-19.

Figure 2-19 Burst write timing



RAM Expansion Cards

The user can expand RAM capacity by inserting 72-pin SIMM cards in RAM expansion slots. Table 3-24 shows the RAM SIMM pin assignments.

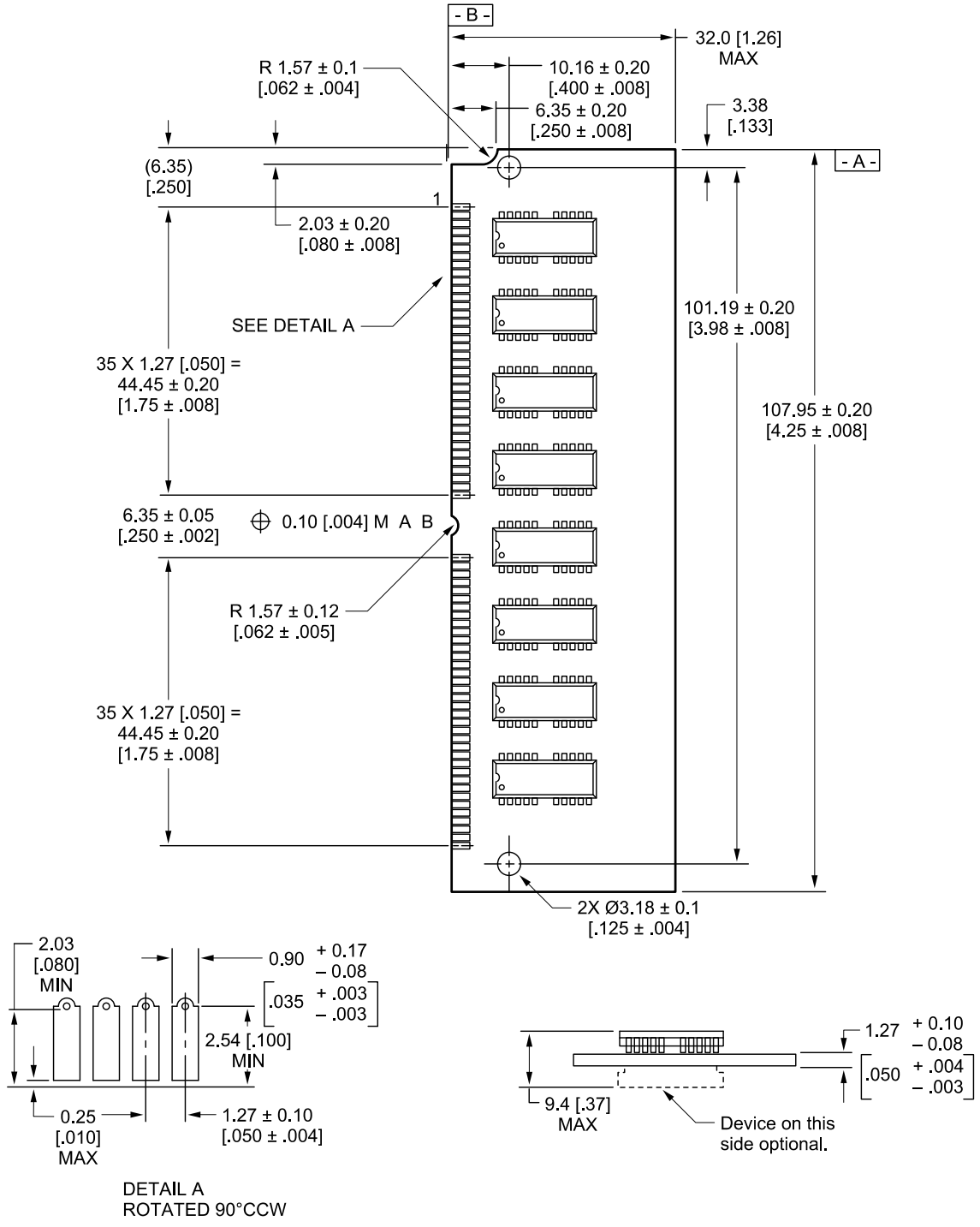
Table 3-24 RAM SIMM pin assignments

Pin	Name	Pin	Name	Pin	Name
1	GND	25	DQ22	49	DQ8
2	DQ0	26	DQ7	50	DQ24
3	DQ16	27	DQ23	51	DQ9
4	DQ1	28	A7	52	DQ25
5	DQ17	29	NC	53	DQ10
6	DQ2	30	+5V	54	DQ26
7	DQ18	31	A8	55	DQ11
8	DQ3	32	A9	56	DQ27
9	DQ19	33	/RAS3*	57	DQ12
10	+5V	34	/RAS2	58	DQ28
11	NC	35	Reserved	59	+5V
12	A0	36	Reserved	60	DQ29
13	A1	37	Reserved	61	DQ13
14	A2	38	Reserved	62	DQ30
15	A3	39	GND	63	DQ14
16	A4	40	/CAS0	64	DQ31
17	A5	41	/CAS2	65	DQ15
18	A6	42	/CAS3	66	NC
19	NC	43	/CAS1	67	Reserved
20	DQ4	44	/RAS0	68	Reserved
21	DQ20	45	/RAS1	69	Reserved
22	DQ5	46	NC	70	Reserved
23	DQ21	47	WE	71	Reserved
24	DQ6	48	NC	72	GND

* A slash before a signal name indicates that it is in the low state when active.

Figure 2-20 shows the mechanical dimensions of SIMM modules for expanding RAM. Dimensions are given in millimeters, with inch equivalents in brackets.

Figure 2-20 RAM SIMM mechanical dimensions



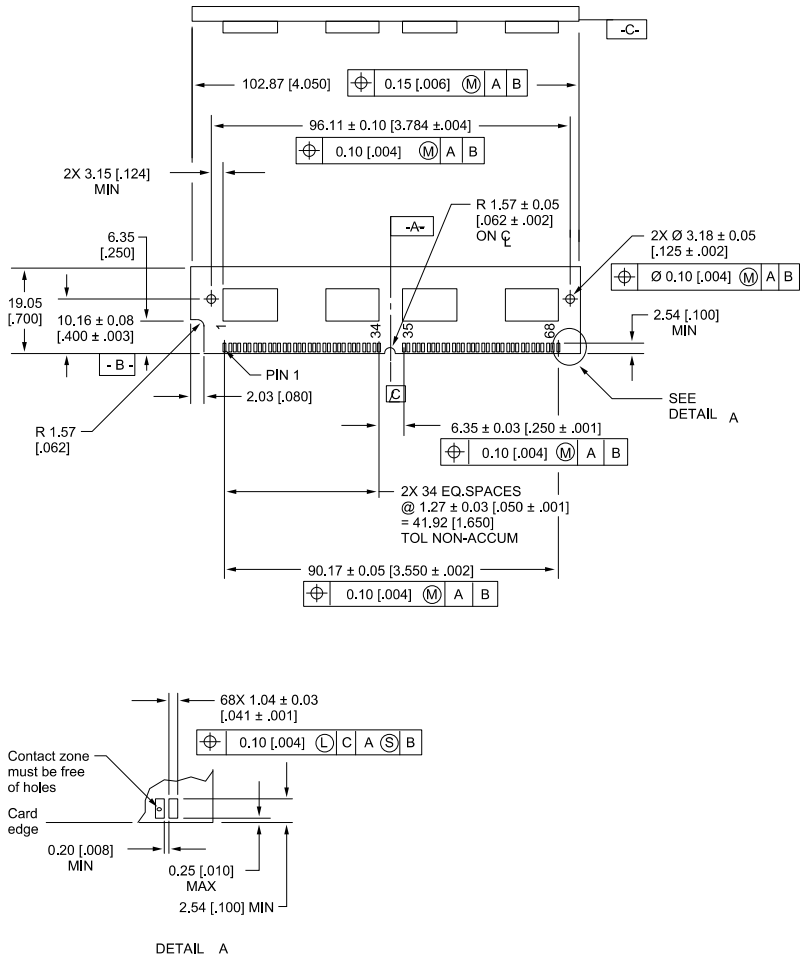
Because of signal loading limits, there may not be more than eight chips per bank of RAM; composite SIMM cards cannot be used.

The Macintosh Quadra 840AV also accepts SIMM cards of a different configuration to expand its VRAM, as shown in the next section.

VRAM Expansion Cards

The Macintosh Quadra 840AV lets the user expand VRAM capacity by inserting 68-pin SIMM cards in its two VRAM expansion slots. Figure 2-21 shows the mechanical dimensions of SIMM modules for expanding VRAM, which are different from the RAM cards discussed in the previous section. Dimensions are given in millimeters, with inch equivalents in brackets.

Figure 2-21 VRAM SIMM mechanical dimensions



VRAM SIMM pin assignments are shown in Table 3-25.

Table 3-25 VRAM SIMM pin assignments

Pin	Name	Pin	Name	Pin	Name
1	+5V	25	DQ5	49	A7
2	DSF	26	SDQ7	50	A8
3	SDQ0	27	SDQ6	51	NC
4	SDQ1	28	NC	52	+5V
5	/DT-OE0*	29	+5V	53	GND
6	DQ0	30	DQ7	54	GND
7	DQ1	31	DQ6	55	SDQ12
8	SDQ3	32	/CAS0	56	SDQ13
9	SDQ2	33	A4	57	NC
10	/WE0	34	A5	58	DQ12
11	/RAS	35	GND	59	DQ13
12	/SE0	36	SC	60	SDQ15
13	DQ3	37	SDQ8	61	SDQ14
14	DQ2	38	SDQ9	62	NC
15	A0	39	/DT-OE1	63	NC
16	A1	40	DQ8	64	DQ15
17	A2	41	DQ9	65	DQ14
18	A3	42	SDQ11	66	/CAS1
19	GND	43	SDQ10	67	GND
20	GND	44	/WE1	68	GND
21	SDQ4	45	/SE1		
22	SDQ5	46	DQ11		
23	NC	47	DQ10		
24	DQ4	48	A6		

* A slash before a signal name indicates that it is in the low state when active.

VRAM access times in numbers of clock cycles are shown in Table 3-26. The Random columns in Table 3-26 show the times required for random accesses; the Second columns show the times required for immediately succeeding accesses.

Table 3-26 VRAM access times

	Macintosh Quadra 840AV		**Macintosh Centris 660AV**	
Access type	Random	Second	Random	Second
Single write	5	7	3	4
Burst write	5-3-3-3	7-3-3-3	3-2-2-2	4-2-2-2
Single read	7	7	4	4
Burst read	7-3-3-3	7-3-3-3	4-2-2-2	4-2-2-2
VDC write	19	20	19	20

Real-Time Data Processing

This part of the *Macintosh Quadra 840AV and Macintosh Centris 660AV Developer Note* covers the software technology of the Macintosh Quadra 840AV and Macintosh Centris 660AV digital signal processing facilities. It contains three chapters:

- Chapter 3, "Introduction to Real-Time Data Processing," describes the software architecture of the real-time data processing facility in the Macintosh Quadra 840AV and Macintosh Centris 660AV. This facility consists of an AT&T DSP3210 chip that performs data-processing operations for applications that contain digital signal processor (DSP) code.
- Chapter 4, "Real Time Manager," describes a new part of the Macintosh system software that supplies all the services an application requires to use the DSP, including loading and running DSP code and performing DSP memory management.
- Chapter 5, "DSP Operating System," covers the DSP operating system, contained in the DSP chip. It provides the services every DSP program needs to work with the Macintosh Operating System.

CHAPTER 3

Introduction to Real-Time Data Processing

This chapter describes the new real-time data processing software architecture for the Macintosh Quadra 840AV and Macintosh Centris 660AV computers, including the functional specifications, features, programming interface, capabilities, and performance. For hardware information about these computers' DSP implementation, see Chapter 2, "Hardware Details."

For the novice in digital signal processing, this chapter begins with an overview of the AT&T DSP3210 digital signal processor and the architecture of real-time data processing. It provides the basics for understanding the rest of the chapter, which provides a more complete discussion of all the concepts and fuller architectural details.

The serious programmer of real-time data processing should read this entire chapter. You must understand several concepts introduced in the section "Real-Time Processing Architecture" to handle real-time programming and data flow properly.

Other parts of this book supplement this chapter. Chapter 4, "Real Time Manager," provides information to the Macintosh programmer and can be skipped by the DSP programmer. Chapter 5, "DSP Operating System," provides information to the DSP programmer and can be skipped by the Macintosh programmer. However, for a complete understanding of the interrelationships and dependencies between the two types of programming anyone doing system debugging or integration should read both chapters. For information about installing and debugging DSP programs in the Macintosh Quadra 840AV and Macintosh Centris 660AV, see Appendix A, "DSP d Commands for MacsBug," Appendix B, "BugLite User's Guide," and Appendix C, "Snoopy User's Guide."

Introduction to Digital Signal Processors

Real-time data processing requires a hardware and software architecture for integrating digital signal processing technology into the Macintosh Quadra 840AV and Macintosh Centris 660AV computers. The architecture supports the computer's digital signal processor as a coprocessor that has its own operating system but is capable of accessing the same data memory as the main processor.

Concepts of Digital Signal Processing

Digital signal processing is the manipulation and conversion of digitized data. Digitized data are digital representations of analog signals, which may represent sounds, images, speech, or other analog forms. To correctly process these signals it is necessary to know at what rate they were converted (the sample rate) and the format of the digital bits used to represent the original data. With this information the signal can be manipulated by a conventional program using the digitized data as its input. The result can then be stored on disk or converted back into an analog signal.

All such processing accomplished by a computer is called digital signal processing. The digital signal processor supports the math routines required in a special chip designed specifically for signal processing applications. The multiply/accumulate operation is the basic ingredient of signal processing programs. The digital signal processor is designed to perform this operation very rapidly.

The equivalent of digital signal processing in the analog domain is accomplished using electronic components, such as inductors, capacitors, resistors, and transistors. The advantage of doing the processing in the digital domain is that the functions can be very precise, reliable, elaborate, and software-configurable. It is difficult and costly to achieve these same goals in the analog domain.

Real-Time Processing Capability

The Macintosh Quadra 840AV and Macintosh Centris 660AV computers' real-time capability uses a multi-tasking coprocessor to give high-performance processing of sound, communications, speech, and images (both graphic and video) while utilizing the system's low-cost dynamic random-access memory (DRAM) for primary storage of data and code. The standard hardware is the AT&T DSP3210 and the audio and telephone input/output (I/O) ports. The software is a custom operating system designed to perform isochronous (*real-time*) and asynchronous (*timeshare*) algorithms. The operating system is based on a *team processing* approach where the work of the total system is carefully separated and delegated between the main processor and the digital signal processor.

This approach has the benefit of

- greatly reducing implementation and hardware costs
- simplifying and speeding up interprocessor communications and data sharing or data streaming
- allowing flexible dynamic load sharing between the main processor and the DSP on selected algorithms
- maximizing the potential to meet future needs for higher performance and multiple coprocessors
- increasing the range of possible application functions the DSP can provide

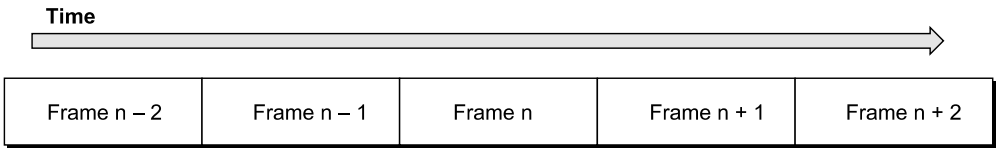
The DSP software architecture supports *dual threaded* processing streams. **Real-time processing** uses interrupt-level isochronous algorithms with guaranteed processing bandwidth to execute real-time functions requiring precisely timed signal generation or inputs such as sound and communications. (Guaranteed processing bandwidth is defined in the next section.) **Timeshare processing** uses asynchronous algorithms that employ the excess DSP bandwidth for functions not requiring time-correlated processing, such as still image decompression or scientific computing.

Additionally, the architecture supports the implementation of NuBus cards to make configurations of multiple DSPs possible.

Real-Time Processing Architecture

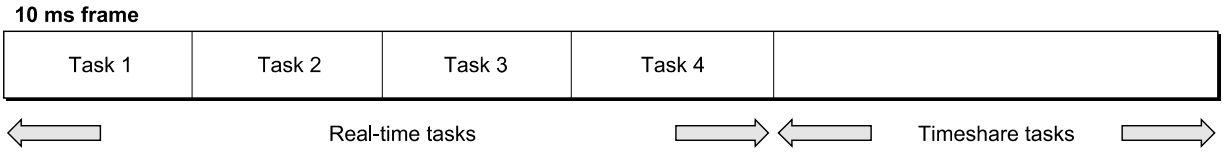
Program execution on the DSP is divided into segments of time called **frames**, typically 10 ms in length, as diagrammed in Figure 3-1. During each frame an attempt is made to run all of the code that is installed on the DSP. **Tasks** are blocks of DSP code that are grouped together by the programmer to perform a specific function.

Figure 3-1 Frames



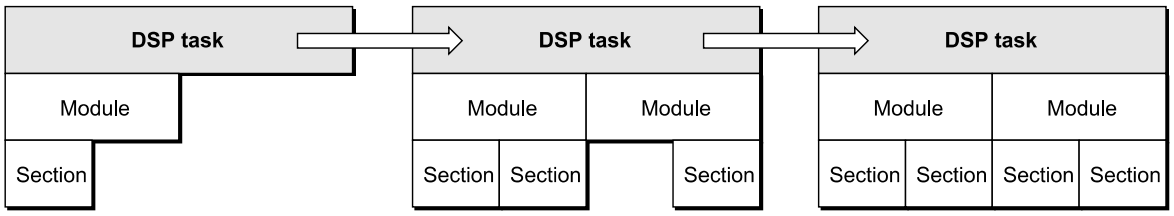
There are two types of tasks: real-time and timeshare. During each frame all of the real-time tasks are executed and then any remaining time in the frame is used for executing timeshare tasks, as diagrammed in Figure 3-2. Real-time tasks are useful for sound, modem, and video processing where there is a fixed amount of data that must be processed during each frame; if more processing time were available it would not be used. However, timeshare tasks use as much processing power as they can get each frame. Image decompression is an example of a timeshare task, since it should decompress the image as fast as possible. This means that when a faster version of the DSP3210 is available timeshare tasks run faster but real-time tasks continue to process the same amount of data.

Figure 3-2 Real-time and timeshare tasks



Each task is assembled out of **modules**, which are the functions that the DSP programmer creates, and each module is composed of **sections**. This relationship is shown in Figure 3-3.

Figure 3-3 Task list



To understand the need for sections, it is necessary to understand how the memory system of the DSP works. To keep hardware costs down, the DSP uses the same DRAM as the main processor. Because the DSP can access memory at a much higher rate than the RAM can provide, and must also compete with the main processor for RAM access, some type of caching on the DSP is needed. The DSP does not have a hardware cache like that in the 68040 main processor. It has a small amount of memory on the DSP chip that is accessed in the same way as main RAM. It is called on-chip memory, in contrast to main memory, which is off-chip. The lack of DSP hardware caching means that caching must be managed by the DSP program and the DSP operating system. This is called **visible caching** as opposed to the transparent operation of most main processor caches.

To accomplish visible caching, the DSP programmer must mark which sections of the code are loaded in on-chip memory before execution and which sections are saved off-chip after execution. Visible caching operates in one of two modes. In **AutoCache** mode, loading and saving are controlled by the DSP operating system; there is only one set of sections on-chip during the execution of a module. In **DemandCache** mode, loading and saving are controlled by the DSP program, so sections can be moved on and off-chip during the execution of the module. Caching modes are discussed in more detail in “Visible Caching” and “Execution Models,” later in this chapter.

To make modules slightly more general, a mechanism is provided for a single module to work at different frame rates and sample rates. This is done by making sections individually scalable. The DSP programmer has the option of saying which sections are scalable and the possible sizes of the scalable sections. For example, if a reverberation module works with both 24k Hz and 48k Hz sound at a 10 millisecond frame rate it would have an input and an output section, both of which would be scalable to either 240 or 480 samples per frame. When the Macintosh program loads the module from disk, it specifies the module scale of operation.

To ensure that all of the real-time tasks are executed during each frame, the DSP programmer must specify an upper bound for the execution time of the module. If there is enough processing power on the DSP, the task that contains this module will be installed and executed. As long as every module’s estimate is correct, the DSP will execute frames evenly. However, if a module’s estimate is not its upper bound, the DSP

could take more time to execute the real-time tasks than is available in a given frame. When this frame overrun occurs the DSP operating system will find the module that specified its incorrect upper bound, remove the task that contains this module from the execution stream, and then resume execution. This procedure is called **guaranteed processing bandwidth (GPB).**

Since a task is made up of modules which typically share data, optimization is provided to keep the data on-chip between modules, instead of saving it off-chip in one module and then loading it back on-chip for the next module. This is accomplished by connecting sections from one module to another, letting the DSP operating system decide if data saving and loading is required. Data that must be shared between tasks, such as the sound going to the speaker, is passed between tasks in **intertask buffers (ITBs).** The only logical difference between ITBs and connected sections is that the sections are in different tasks for ITBs and in the same task for connected sections. Both ITBs and connected sections are managed by the Macintosh programmer, as described in "Data Buffering," later in this chapter.

Software Model

The software model for real-time data processing in the Macintosh Quadra 840AV and Macintosh Centris 660AV computers consists of three distinct pieces:

- The host toolbox is the **Real Time Manager.** The Real Time Manager runs on the main processor and is written in C for portability.
- The DSP Driver contains both main processor code and DSP code components. All hardware-dependent functions are included in the drivers. They are written in the 68000 and DSP assembly languages for efficiency.
- The DSP toolbox is called the **DSP operating system.** The DSP operating system runs on the DSP, and is written in DSP assembly language for efficiency.

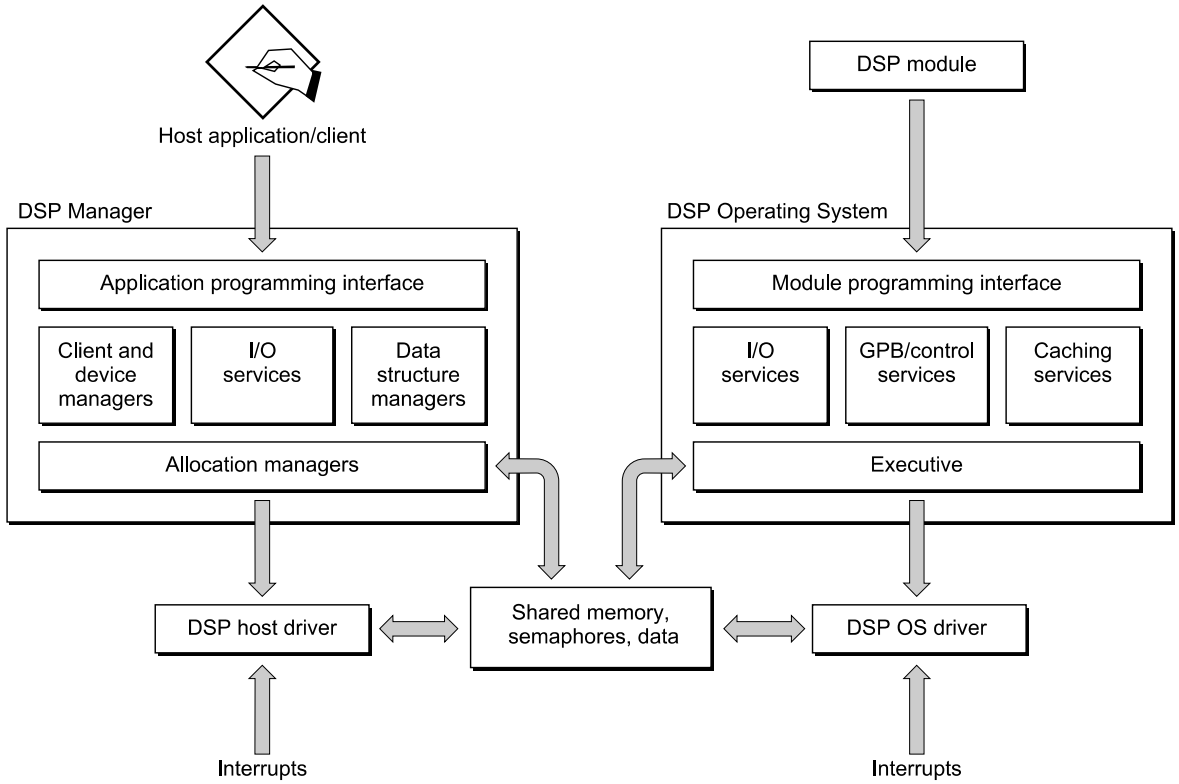
Almost all routines in the Real Time Manager are reentrant and callable from interrupt level. This is necessary, since communications between the DSP and main processor often take the form of interrupt messages.

A major component of the model is a shared block of memory. This memory consists of local memory as well as main memory. The local memory is either in system DRAM or in optional card memory. It is through data structures and semaphores in this shared memory that the main processor and DSP toolboxes communicate. A more complete diagram of the software model is shown in Figure 3-4.

Dual Programming Model

Figure 3-4 shows the dual programming interface for real-time data processing: the application programming interface (API) in the Real Time Manager, and the module programming interface (MPI) in the DSP operating system. These two interfaces are completely separate, and designed to be used by different programmers. It is not necessary for a programmer to be both a Macintosh programmer and a DSP programmer.

Figure 3-4 Real-time data processing organization



It is usually better to have two programmers involved when programming an application that requires DSP modules. This is because the two types of programming are very different, and very specialized. The two programmers communicate with each other by creating a *DSP Module Specification* document. This document provides a vehicle for transferring all the information necessary to ensure a correct interface between the main processor program and each DSP module. For more information about the data this document should contain see "DSP Program Information for the Macintosh Programmer," in Chapter 5.

Real Time Manager

The Real Time Manager uses the standard trap interface to call the Macintosh Toolbox. The set of calls accessible to an application are labeled as the application programming interface layer in Figure 3-4.

Three major functions of the Real Time Manager support I/O services, client and device management, and data structure management. These functions make calls on the Real Time Manager's allocation routines at the lowest level.

The allocation layer is responsible for DSP cache and local memory allocation, for GPB allocation, and for I/O resource allocation.

DSP Operating System

The DSP operating system also has an interface layer. This layer works in a similar fashion to the Real Time Manager: a trap mechanism is used to make calls on the DSP operating system from the DSP module.

The DSP operating system also provides services to the DSP module: I/O services, including FIFO management, GPB and control services, and caching operations on the DSP. The underlying function of the DSP operating system is contained in an executive layer, which is responsible for managing task-sequencing and frame-handling functions.

DSP Driver

The DSP Driver has two distinct components. One works exactly like a standard Macintosh driver, and is written in 68000 code. The other component performs a similar function for the DSP operating system. It contains all DSP code that is hardware-dependent, as well as booting and restart code These two components are stored together as one driver. The DSP driver also controls the I/O drivers for any serial or parallel I/O ports included as part of the DSP system. These resources are accessed using the Real Time Manager services.

Other Software Components

Additional system software that supports real-time data processing includes:

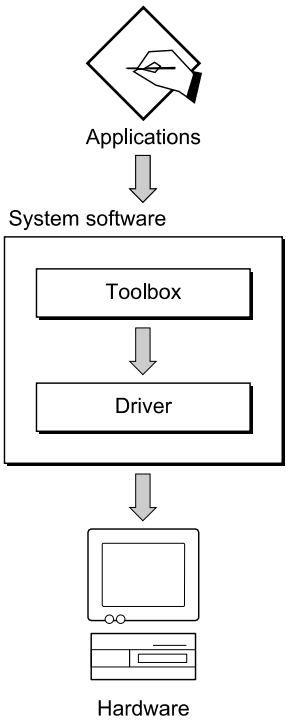
- A *sound driver* provides the interface between the Macintosh Sound Manager and the Real Time Manager by means of a set of *standard sound* modules, including sound input and output, compression, filtering, sample rate conversion, and mixing.
- A *telecom driver* provides the interface between the telecommunications Manager/ Communications Toolbox and the Real Time Manager, including a set of standard telecom modules, plus modem, fax, and speech.
- *Development tools* include a DSP C compiler, assembler, libraries, linker, resource generator, and include-files with macros and definitions.
- *Debugging and test tools* include a graphical module installer, DSP code debugger, and MacsBug extensions.

The purpose of the various toolbox drivers is to provide access to the capability of the DSP at the highest possible toolbox level. This allows applications that are not written for the DSP to use it automatically when it is available. Even with this level of toolbox support, it is clear that many applications will work better by directly accessing the DSP using the DSP API. Such applications provide significantly more functionality or speed when a DSP is available. However, an application that uses the DSP API either cannot run on a platform without the DSP, or must provide alternative main processor programming if a DSP is not available.

Software Layers

The basic Macintosh software model has four primary conceptual layers: the application layer on top, the toolbox layer, the driver layer, and finally the hardware layer. The separation of system software into toolbox and driver layers allows the separation of hardware dependencies from the major system functions, and makes revisions in the hardware easier to support. If this model is followed correctly, major changes in the hardware can be made without breaking applications. For this reason, Apple encourages developers to access functions at the highest possible toolbox layer, even if they could be more efficient writing directly to the hardware. This separation allows Apple to improve the hardware base without disrupting the application base. A diagram of the four-layer Macintosh model is shown in Figure 3-5.

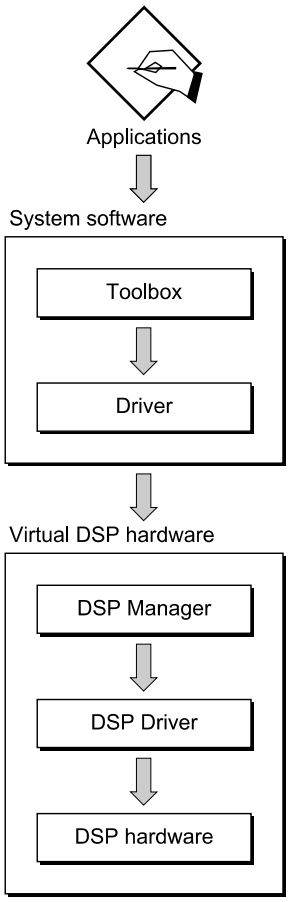
Figure 3-5 Four-layer Macintosh model



As shown in Figure 3-5, an application that accesses the Real Time Manager is hardware-dependent. This means the application would require that a DSP coprocessor be present in the system in order for it to operate. This is true even though the Real Time Manager is hardware-independent. The emphasis here is on implementation. The Real Time Manager assumes that there is a DSP available, otherwise there is no reason for the manager to be installed. Additionally, it provides the necessary isolation from the specific implementation details. By accessing a higher toolbox layer the application also becomes DSP-independent and will operate across multiple Macintosh platforms.

If the original Macintosh model is combined with the DSP model, the DSP software and hardware must be viewed as *virtual hardware*. This concept is illustrated in Figure 3-6.

Figure 3-6 Six-layer model

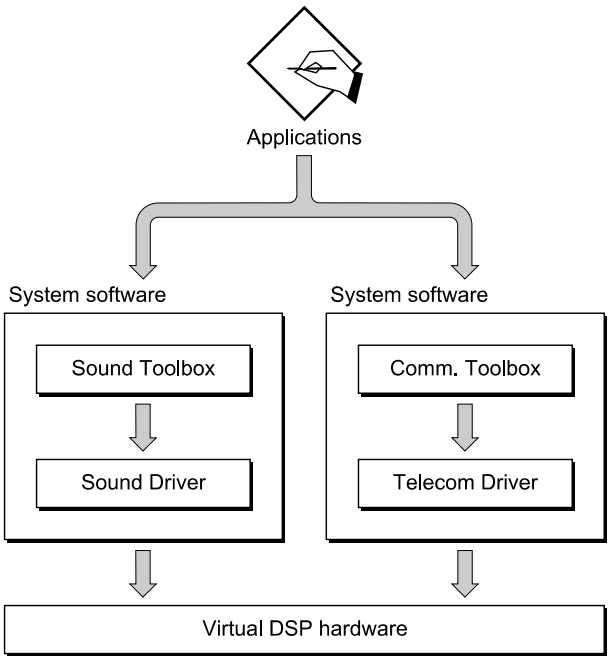


The model shown in Figure 3-6 is used for the DSP software. Notice that the driver layer is specific for the virtual hardware. If the DSP is available, this layer must be able to install tasks in the task list and must deal with any specific characteristics of this machine that may affect its operation. If there are no such characteristics, then the driver is not dependent on the machine implementation, but only on the availability of the DSP. In either case, the driver is specific for the virtual hardware.

Figure 3-7 shows two sample toolbox/driver combinations for the Real Time Manager.

In the case of sound, there are no hardware-specific features that the Sound Driver needs to deal with. Hence only a single-layer driver is needed. The driver is capable of working with the DSP in any supported configuration, and does not need to deal with specific implementation details. This results in a six-layer model.

Figure 3-7 Example of toolbox and driver layers

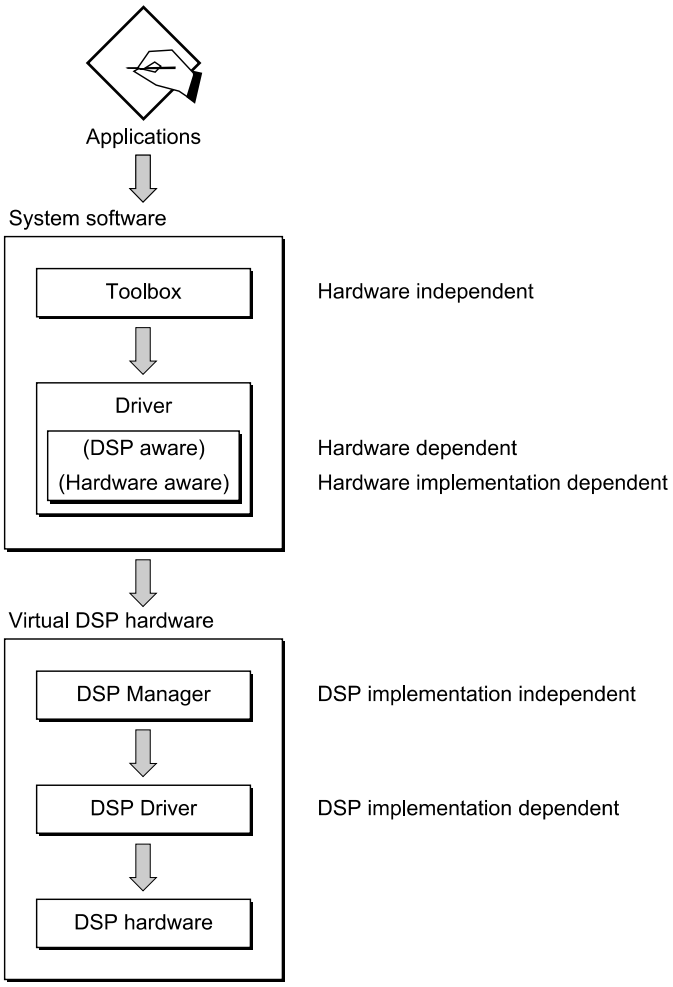


For the communications case, the Telecom Driver deals specifically with the way that the DSP I/O subsystem is connected to the telephone line. Thus, specific bit input and output (BIO) pins on the DSP perform functions that the Telecom Driver uses. The driver takes control of these functions if the appropriate external hardware is present on the telecom port. This makes the Telecom Driver hardware-specific relative to the telecom subsystem. It is also hardware-dependent on the DSP virtual hardware.

To the extent that the same configurations are used for all CPUs and cards, the Telecom Driver becomes universal, and seemingly hardware-independent. However, different arrangements of telecom subsystems for different implementations of the DSP will require a different telecom driver. Notice that a different telecom driver must be supplied for a NuBus card and for a CPU, even if the configuration is identical. This is because the CPU Driver can recognize a specific CPU but cannot recognize a specific NuBus card. If the wiring of the I/O subsystem is identical in both cases, then the only change to the driver is the hardware recognition code.

To facilitate this, the driver layer should be divided into two separate parts: the DSP-handling layer on top that uses Real Time Manager routines, and the hardware-specific layer on the bottom that deals with specific hardware wiring. This allows simple modification of the driver to support different hardware platforms. This arrangement is shown in Figure 3-8.

Figure 3-8 Seven-layer real-time model



The addition of this seventh "H/W driver" layer is only necessary if the driver requires specific access to I/O subsystems.

DSP-Aware Applications

A DSP-aware application can be designed to operate in two different ways:

- to recognize and use the DSP if it is there, for enhanced performance of specific application functions
- to require the DSP and not run at all if no DSP is available

There are many interesting applications in both categories. It is important to realize that the Real Time Manager's implementation independence makes it possible to write a DSP-aware application that will run, without change, on different DSP implementations,

assuming the same (instruction set compatible) DSP is used. Such an application can make direct calls to the Real Time Manager for service. Different instruction sets can be supported by the appropriate processing modules.

It is important to note that if a desired function is available from a high-level toolbox then the DSP connection will be made automatically, providing enhanced performance without the application being written for the DSP. A good example of this is any application that plays sound. If it calls the Sound Manager then the processing will be handed over to the DSP. However, if a sound application needs more service than the Sound Manager provides then the application should directly access the Real Time Manager. Depending on the application, either of these DSP-aware models could be used.

Software Architecture

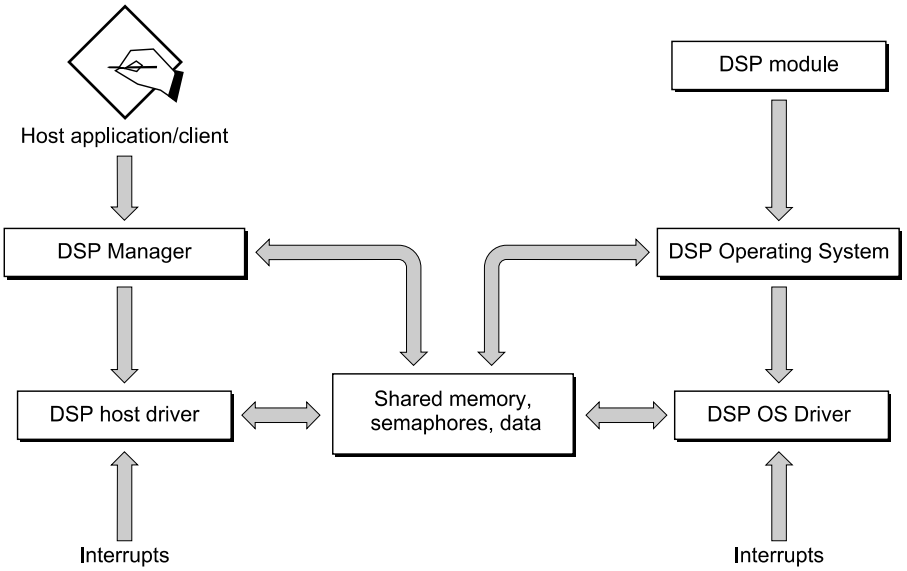
The real-time data processing software is based on a data flow model. It is important for a real-time signal processing system to accept and process incoming samples at the average rate that they are being produced by the input process. It is equally important for it to create outgoing samples at the average rate that they are being consumed by the output process.

By buffering the samples, it is possible to process groups of samples at a time rather than single samples at a time. This approach is called **frame-based processing.** During each frame the application loads the required program code, variables, and input data into a high-speed cache on the DSP. The program code is executed from this cache, and the resulting output data is dumped from the cache back into off-chip memory. Alternately, the input data may already be in the cache from a previous operation, and the output data may be kept in the cache if it is needed for following operations.

The operating software for real-time data processing works on a team processing basis. In particular, careful attention has been paid to the division of labor between the main processor and the DSP. The goal is to maximize the processing throughput of the DSP while minimizing the processing requirements and bus loading of the main processor. The operating software consists of a part of the Macintosh toolbox (the Real Time Manager and its driver) and a DSP control program (the DSP operating system and its driver). A block diagram of this concept is shown in Figure 3-9.

These two programs interact with one another through shared memory, interrupt processing, and semaphores. The Real Time Manager supports application software on the main processor, while the DSP operating system supports DSP program modules on the DSP. Thus, there are two completely separate application program interfaces in real-time data processing: one for the main processor program and one for the DSP program.

Figure 3-9 Real-time software organization

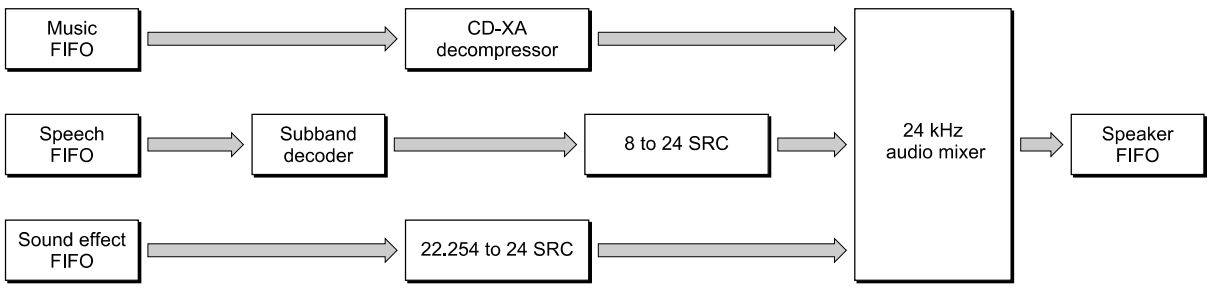


In most applications the DSP will need to run several different code modules or algorithms in sequence to process blocks of data. For example, five different DSP modules are required for a sound player to mix the following three channels of sound:

- compressed music requiring data decompression
- compressed speech requiring a subband decoder and an 8-to-24 kHz sample rate converter
- sound effects requiring a 22.2545-to-24 kHz sample rate converter

Each module must be cached and executed in the proper order to accomplish the desired results. See Figure 3-10 for a diagram of the data flow in this process.

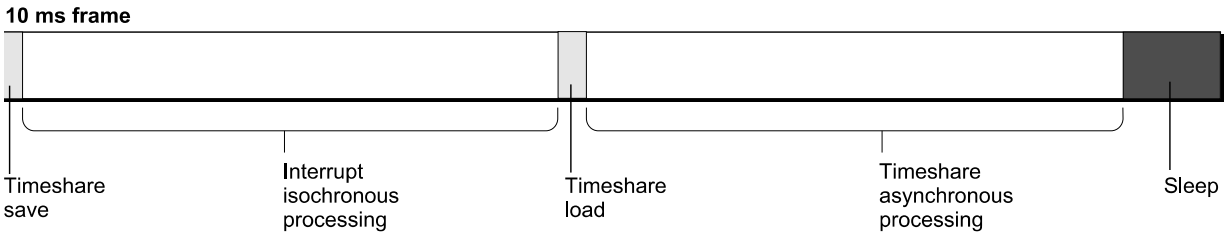
Figure 3-10 Sound player example data flow



Frame Organization

Figure 3-11 shows the processing divisions that occur during a frame. Each frame begins with the frame interrupt. If a timeshare task is running, its context is saved in external memory. Then the list of real-time tasks is parsed and each of the active tasks are executed in sequence. When all real-time tasks are completed, the timeshare processing is resumed. If there was a task being executed when the frame interrupt occurred, it is reloaded; otherwise, the list of timeshare tasks is checked. The next active task is located using a round-robin scheduling algorithm. This selected task is then loaded and executed. Processing continues until the next frame interrupt or until all timeshare tasks are completed or become inactive.

Figure 3-11 Frame-based processing

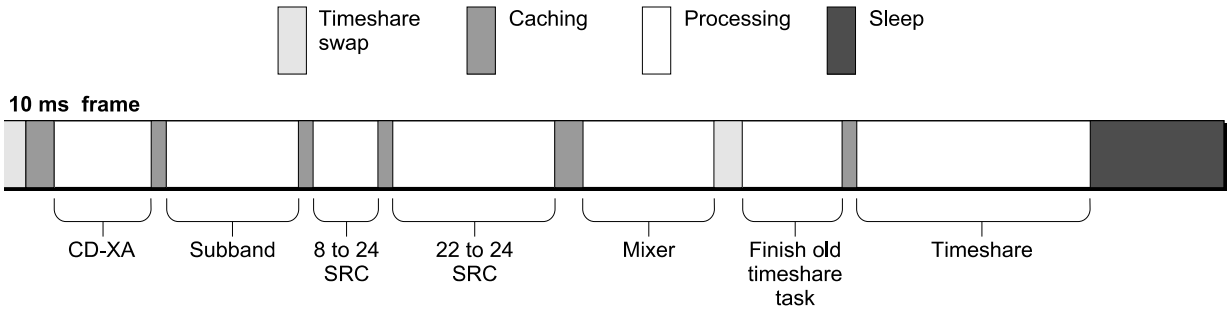


If there are no active timeshare tasks to be done, the DSP goes into **sleep mode** (shuts itself down), using the wait-for-interrupt instruction. The DSP will then be brought back online automatically at the next frame interrupt. This provides automatic power control for portable computers based on the DSP's processing load. If no DSP tasks are active, the Real Time Manager will go even further and shut down all DSP-related circuits, including the timer, serial ports, and other related hardware.

Note
During a frame all real-time tasks are executed once and only once. Timeshare tasks use cooperative multitasking, similar to Macintosh applications, and are executed in sequence until all timeshare tasks become inactive or the end of the frame is reached. ◆

Using the sound player example given earlier, a detailed diagram of a frame is shown in Figure 3-12. This figure is not to scale and shows only general content.

Figure 3-12 Multiple code module processing

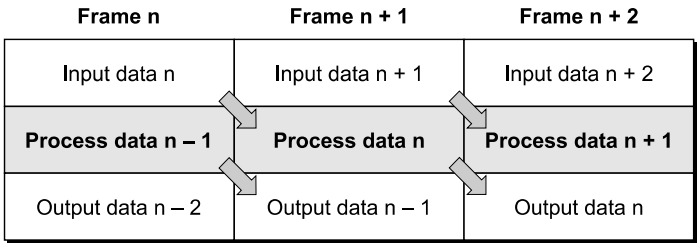


As Figure 3-12 shows, the five required real-time processing modules are run in sequence. The timeshare algorithm that was running when the frame started is reloaded and completed. One more timeshare algorithm is run, and since no more timeshare algorithms are active in this example, the DSP goes to sleep and waits for the next frame interrupt.

Frame Size Selection

Frame-based processing requires some latency in the data flow. In particular, the input port must collect a full frame's worth of samples before the DSP can process them. Likewise, the DSP must generate a full frame's worth of samples before the output port can start transmitting them. This requires a latency of two frames between input and output data. Figure 3-13 illustrates this basic concept.

Figure 3-13 Process data flow



There are four factors that influence the selection of the time interval of the frame. They are:

- *Size of buffer.* This is proportional to the frame time interval. The longer the frame, the more cache memory is needed for each buffer.
- *Overhead reduction.* This is inversely proportional to the frame time interval. The shorter the frame, the greater percentage of DSP processing time is used in overhead. For example, if the frame represents 240 samples then the overhead is 1/240 of the algorithm on a sample-by-sample basis. Algorithm caching is needed only once for every 240 samples or 0.42% compared to processing a single sample at a time.

- *Granularity of access.* During a frame the processing sequence cannot be interrupted. Changes in process configurations must happen on frame boundaries.
- *Input/output latency for important algorithms.* The longer the frame the higher the latency between input to output data streams.

Buffer size and overhead reduction pull in opposite directions. Granularity of access is dependent on the application; sound synthesis with MIDI is probably the most demanding potential application, putting the lower limit at approximately 2 to 4 ms per frame. Input/output latency sets the upper limit on the frame time. The most demanding known algorithm for latency is the V.32 data protocol, which sets an upper limit of 13 ms per frame.

The default frame time for the Macintosh Quadra 840AV and Macintosh Centris 660AV is 10 ms. This is a convenient value for the following reasons:

- many common sample rates have an integer number of samples in 10 ms
- the buffers are small enough to have several in the cache at the same time (only 240 samples for 24 kHz)
- a decimal-based frame time is easier to work with
- a 10 ms frame time reduces the DSP operating system overhead

The software architecture of the Macintosh Quadra 840AV and Macintosh Centris 660AV is flexible and supports multiple frame rates up to four. The standard alternate frame rate is 5 ms. In the Macintosh Quadra 840AV and Macintosh Centris 660AV implementation, the frame rate can be changed only when no programs are using the DSP.

Visible Caching

The basic assumption for visible caching is that there is not enough high-speed cache to hold all of the code the DSP must execute each frame. This difficulty is overcome without increasing hardware costs by caching each algorithm (module) from external memory into high-speed cache when it is needed. Because most algorithms for the DSP consist of some set-up code and a compact set of loops that take up most of the processing time, this method of visible caching results in only a small fraction of the total main processor bus bandwidth being used by the DSP.

▲ **WARNING**
If you are writing a system extension that uses real-time processing, be aware there is only a limited amount of memory available because the system heap is not expandable. You will need to include a 'zsys' resource in your system extension to enlarge the system heap before the system extensions run. The amount of memory needed may be more than required by your system extension because some of the memory may be used by LocalTalk, EtherTalk, TokenTalk, and A/ROSE. ▲

The visible caching approach works for many signal processing algorithms. The assumption is that only a small processing loop is needed with a small amount of data per frame, resulting in a fairly short caching time overhead. The loop is run many times per sample and takes considerable processing time. For audio and telecommunication

algorithms, the ratio of processing instruction cycles to caching cycles is often in the 40:1 range. Hence, the caching overhead cost in processing power is in the 10 percent range. This is a fairly low impact considering the cost savings from eliminating fast SRAM and its related support circuitry. However, this processing model does not work well for applications where these assumptions do not hold.

The signal processing algorithms, variables, and data tables are all stored in locked contiguous memory blocks (called **sections**) and are loaded into cache memory either automatically or by calls to the DSP operating system's visible caching routines. With this approach the DSP programmer has complete control of the caching process, unlike most hardware caches that are invisible to the programmer and to the executing program.

Code can also be executed directly from external memory. This is useful for small code blocks, such as set-up and control code, or blocks that contain only single instruction loops that are cached automatically on the DSP chip. It also allows very large code blocks to be run by the DSP, although the execution speed will be substantially lower.

Assuming support for DRAM page mode is provided in the hardware, the caching function (block move) is likely to be three times more efficient than single accesses. Single external accesses are used when executing from external memory or when fetching or updating data in external memory. Even for fairly short control and set-up code blocks it is often faster to cache them before execution. The break-even point can be calculated based on the cache speed, single access speed, and block move speed of any given implementation, and is often as low as 25 instructions. For information about DRAM timing, see "Access Timing," in Chapter 2.

Under normal circumstances, the DSP should demand only a low percentage of the CPU bus bandwidth. This allows graphics and other main processor functions to proceed as rapidly as possible. However, there are DSP applications that take a significant amount of the CPU bus time, in which case the main processor runs slower. But since much of the work is being done on the DSP, the total system runs faster than a computer without a DSP.

As explained in "Real-Time Processing Architecture," earlier in this chapter, here are two visible caching execution models that are supported by the DSP operating system: AutoCache and DemandCache. With AutoCache the programmer specifies which code and data blocks are to be loaded and saved. The DSP operating system performs all load and save functions automatically. In DemandCache the programmer explicitly moves code and data blocks on and off-chip whenever needed by making the appropriate calls to the DSP operating system from the module. Both models have advantages and disadvantages.

The AutoCache model provides a simple easy-to-use method of visible caching for small DSP algorithms (for example, sample rate converters, compressors and expanders, filters, and others). Whenever possible, the AutoCache model should be used, for simplicity of operation and programming.

In the DemandCache model, caching is explicitly handled by the DSP programmer. In the simple case, the programmer provides a single main program and one or more cacheable functions. A cacheable function is made up of one or more code blocks. The

main program resides in external memory and calls the DSP operating system to cache functions on-chip and run them. The programmer can thus select functions in any order and can repeat functions as needed, at the cost of increased program size and complexity.

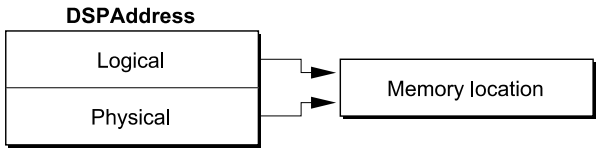
DemandCache is used for algorithms that must select different signal processing functions depending on conditions or commands. A good example of such an algorithm is a multimode modem program. The actual data processing program selected depends on the kind of modem on the other end of the telephone line. The required program would be cached explicitly by the main program.

Another way to build complex functions is by combining multiple simple modules and using the skip function. This is described in "Grouped Modules," later in this chapter.

DSP and Main Processor Addressing

Real-time data processing is designed for systems that include a memory management unit (MMU). However, the DSP3210 does not use an MMU to translate logical addresses to physical addresses. As a result, the main processor uses logical addresses for all of its memory accesses while the DSP uses physical addresses. Addresses that are used by both the Macintosh and DSP operating systems are stored in `DSPAddress` structures that contain both the logical and physical form of the address. A diagram of the structure is shown in Figure 3-14.

Figure 3-14 `DSPAddress` structure



Note
The Real Time Manager is responsible for setting up and maintaining these `DSPAddress` data structures. Since the DSP uses locked–down memory, this approach allows the DSP to operate in a virtual memory (VM) system without actually having an MMU. The local memory addresses are translated from logical to physical form by the Real Time Manager before the DSP chip uses them. ◆

All blocks of memory indicated by a `DSPAddress` data structure are by definition locked contiguous and non-cacheable. They are locked contiguous so that the DSP does not have to worry about scatter/gather operations when using a `DSPAddress` data structure. The blocks are locked non-cacheable to eliminate conflicts that would occur when the DSP modifies a memory location that the main processor had cached.

The `DSPAddress` is a general type. There are also specific types, including `DSPFIFOAddress`, `DSPTaskAddress`, `DSPModuleAddress`, and `DSPSectionAddress`. Each has the same data structure as a `DSPAddress` but points to a specific structure.

Containers

Each memory location that a given section may occupy is called a **container.** For example, if a section can be cached on-chip from an off-chip location it has two containers—one in main memory and one in the DSP's on-chip memory. Containers are fully discussed in "Sections Defined," later in this chapter. The DSP operating system keeps track of the active container by means of data structure called a **section table.**

Primary and Secondary Pointers

Each section has a primary and a secondary pointer. There are two possible values for these pointers, depending on whether the section uses one container or two containers. You must be careful when examining or using these pointers when DSP code is running because in DemandCache the DSP operating system can change the sections from one-container to two-container when caching sections on-chip, and from two-container to one-container when moving sections off-chip. The pointers are summarized in Table 3-1, where X and Y are pointers to sections.

Table 3-1 Primary and secondary pointers

Primary	Secondary	Where applicable
X	`nil`	One-container section
X	*Y*	Two-container section
X	*X*	Not applicable
`nil`	*X*	Not applicable
`nil`	`nil`	Not applicable

The pointer to the section in the exception vector table is always the same as the primary pointer. This invariant is maintained by the DSP operating system during both AutoCache and DemandCache.

One-Container Sections

Sections that have only one container have a primary address and a `nil` secondary address. The primary address can point either on-chip or off-chip. Whenever the section data is accessed by the DSP, the primary address is used.

Two-Container Sections

Sections that have two containers are slightly more complicated. There are valid addresses in both the primary and secondary pointers. The primary pointer is where the DSP user code will access the section. The secondary pointer is where the DSP operating system will load the section from and save it to. Both the primary and the secondary address may point on-chip or off-chip.

On-Chip and Off-Chip Addressing

Initially the application will want to find out if the addresses discussed in the previous section point to locations that are on-chip or off-chip. The following rules apply:

- The application can tell if the address points on-chip by looking at the physical and logical components of `DSPAddress`. If the logical value is `nil` and the physical value is not `nil`, the address points to on-chip memory.
- Pointers to off-chip memory can be recognized because the logical and physical pointers are both not `nil`.
- It is not valid to have a logical address without a physical address.
- If the logical and physical components of `DSPAddress` are both `nil`, the pointer is `nil`.

These rules are summarized in Table 3-2, where X and Y are addresses.

Table 3-2 On-chip and off-chip addresses

Physical address	Logical address	Where located
X	`nil`	On-chip
X	*Y*	Off-chip
X	*X*	Off-chip
`nil`	*X*	Not valid
`nil`	`nil`	Not valid

Guaranteed Processing Bandwidth

A system of measuring and controlling execution time guarantees that real-time tasks will execute completely every frame. This system is called guaranteed processing bandwidth (GPB).

GPB is measured in processor instruction cycles. For example, with 10 ms frames, 166,666 cycles are available for a 60 ns instruction cycle and 125,000 cycles for an 80 ns instruction cycle. Therefore, if a processor is running 60 ns instruction cycles instead of 80 ns instruction cycles, more instruction cycles are available for a given frame time.

Each code module is assigned a GPB number during development by the DSP programmer. This number is called the *GPB estimate*. It is an estimate because certain portions of the processing time depend on bus latency and other factors that are not the same for different machines or implementations.

When the DSP program tries to install a task in the real-time task list, its estimated GPB requirement is compared with the remaining GPB available (calculated by subtracting the GPB values for real-time tasks already installed from the total available GPB). If there is enough time available, the new real-time task is installed. Otherwise, an error message is sent back to the application attempting to do the installation.

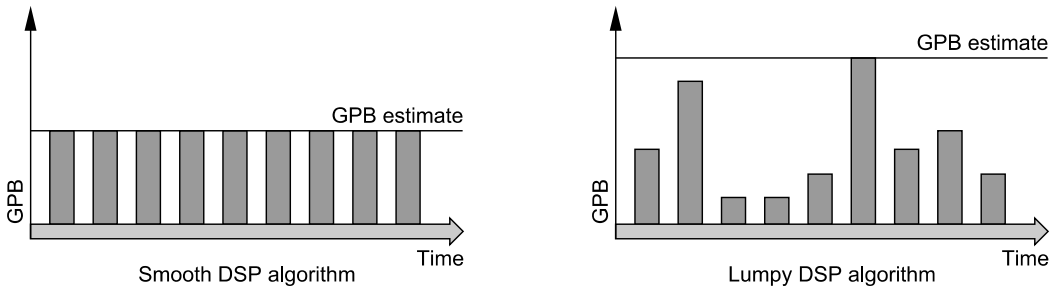
Each time a real-time task runs, the DSP operating system calculates the *GPB actual value* for the task. This actual value is used for future calculations in determining if additional real-time tasks can be installed. Also, this revised GPB actual value can be used to update the modules value in the DSP Prefs file to improve the GPB estimate for the current target machine. In this way, the estimate becomes adapted to faster or slower hardware implementations.

Smooth and Lumpy Algorithms

The simple model described above works well for smooth DSP algorithms. A **smooth algorithm** is one that always takes the same or almost the same time to execute every frame. The "almost" comes from variations outside the control of the algorithm, including I/O time handled by the DSP operating system, and bus overhead, which may vary depending on other bus traffic. There can also be minor variations within the algorithm, but these must be kept to a small percentage if the model is to work correctly.

The other type of DSP algorithm is called a **lumpy algorithm.** In this case, the algorithm uses various levels of processing for each frame This may depend on the data being processed, the status of the function it is controlling, or other variables. A diagram comparing the two types of algorithms is shown in Figure 3-15.

Figure 3-15 Smooth and lumpy DSP algorithms



As you can see from the diagram, the GPB estimate for the smooth algorithm is also the GPB actually used on a regular basis. On the other hand, the GPB estimate for the lumpy algorithm must indicate the maximum level of processing required. To guarantee DSP processing availability, the maximum level of processing must always be used in GPB calculations. Thus there is often additional timeshare processing available when a lumpy algorithm is running. The DSP programmer must indicate whether each module is using a smooth or lumpy algorithm.

Calculating GPB

For real-time algorithms, the actual GPB is recalculated by the DSP operating system every frame. If the new GPB actual value is larger than the stored GPB actual value from previous frames, the new value is stored. This is called the *peak detection* algorithm. It is designed to maintain the actual maximum GPB used, including any bus or I/O variations. The GPB actual value starts off at zero when the real-time task is installed.