TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

4 Development Support

This section serves as a guide to a more thorough understanding of the TPU architecture for the purpose of debugging microcode in a system environment. Development support encompasses operation of the scheduler, microengine, execution unit, RAM, timer channels, and development support registers. Figure 4-1, which shows a detailed block diagram of the full TPU, is divided into five main units: the scheduler, the microengine, the execution unit, the timer channels, and the host interface (including the registers and parameter RAM).

4.1 Scheduler

Every time function executed on a particular channel is composed of one or more states. A state is constructed of a specific number of microinstructions that are not interruptible when executed by the microengine. This microinstruction set is typically the code necessary to calculate the next phase of a waveform to be input or output from a given channel. The intent of every channel is to receive time for state execution (to be serviced). Since one microengine handles 16 time functions operating concurrently, the time function states must be executed serially. The task of the scheduler is to recognize and prioritize the channels needing service and to grant each channel state execution time. The time given to an individual state for execution or service is called a time slot. The duration of a time slot is determined by the number of microinstructions the state contains and, therefore, varies in length.

At any time, an arbitrary number of channels can require service by the microengine. To request service, a channel notifies the scheduler by issuing a service request. A service request, which is any occurrence that asserts the service request latch, has four origins:

1. Match Recognition Service Request
2. Transition Detect Service Request
3. Channel Linking Service Request
4. Host Service Request

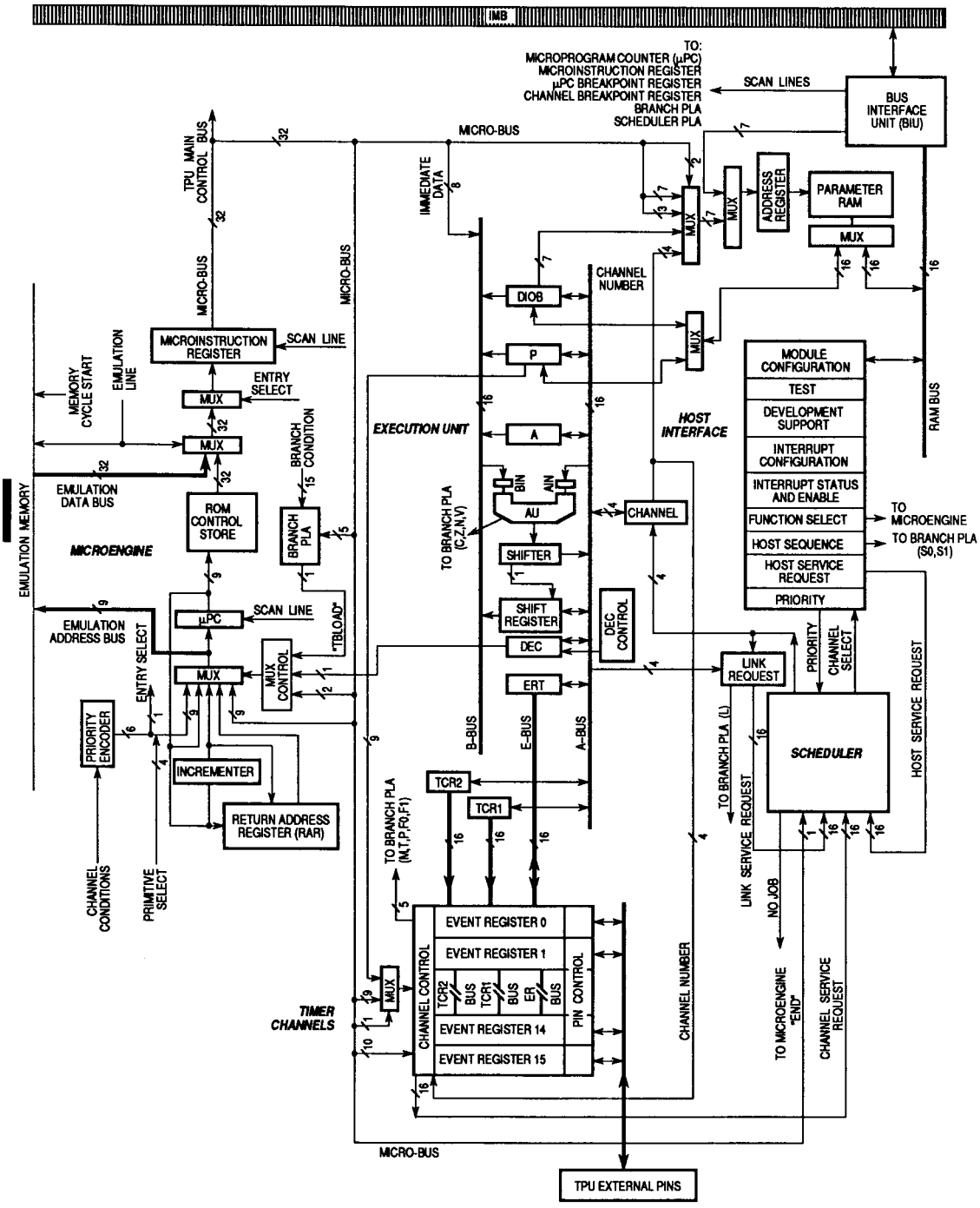


Figure 4-1. TPU Detailed Block Diagram

The scheduler then grants the channel a time slot. Once a time slot is granted, the service grant latch for that channel is asserted, which disables the service *request* latch. As a result, the channel may request new service, but is not serviced again until all other requesting channels have been serviced. The service *grant* latch then notifies the scheduler that the channel has been granted a time slot. Likewise, while this latch is asserted, the channel is not granted another time slot for new service.

In addition to organizing incoming requests, the scheduler must also ensure that no channel permanently blocks another channel from receiving a time slot. To meet such demand, scheduling necessitates a priority scheme.

4.1.1 Priority Scheme

For servicing, every channel is assigned one of three priority levels: high, middle, or low. Assignment, as specified in the system priority registers, is discussed in **2.2.6 Channel Priority Registers (CPR1, CPR2).** Priority level is determined based on the maximum latency desired for each channel. A channel having a time function that requires the most frequent or more immediate service should be allocated a high priority level. To execute service requests, the scheduler addresses two aspects of priority: 1) it recognizes that the time function of one channel may require data more frequently than the function of another channel, and 2) it recognizes that all channels need an equal opportunity to be serviced.

The TPU employs a primary and a secondary priority scheme. These two schemes ensure frequent servicing of high-demanding time functions and ensure a minimum time allocation to all channels requesting service, regardless of their priority level. The primary scheme prioritizes requesting channels that have different priority levels; the secondary scheme prioritizes requesting channels that have the same priority level. The relationship of these schemes is discussed in the following paragraphs.

Initially, a channel requests service and is granted a time slot by the scheduler. Both service request and service grant latches are asserted. If only high -level channels constantly receive service first because of their priority level, middle and low-level channels would only be serviced by default, i.e., if no high-level channels request service. To ensure that each priority level receives an opportunity for servicing, every time slot has a fixed priority level that the scheduler honors first. Divided into sets of seven, time slots are numbered from one to seven. Figure 4-2 illustrates the numbered time slots in sets of seven (fields **A** and B) and identifies their assigned priority level. The high level has more time slots than the middle and low levels. Out of every seven time slots available, four are assigned to honor high-level channels first, two are assigned to honor middle-level channels first, and one is assigned to honor low-level channels first. Service requests are assigned a time slot for execution. Only one request is serviced per time slot.

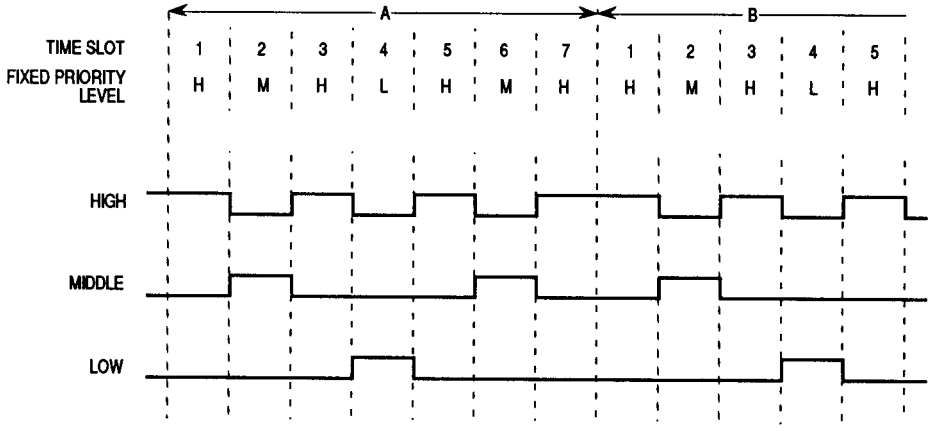


Figure 4-2. Priority Levels

4.1.1.1 Primary Scheme – Priority among Channels on Different Priority Levels

Although time priority is fixed, the servicing priority is not. The primary scheme acknowledges the priority level assigned to a-time slot, granting service first to a channel having the same priority. Referring again to Figure 4-2, time slot 1 has a high-level assignment; therefore, any high-level channel requesting service is recognized first. However, if no high-level channel requests service, the scheduler recognizes a requesting middle-level channel. If this level has no request, the scheduler continues to the low level. If no requests occur, the scheduler remains in the time slot waiting for any channel to request service. Granting service to a different-level channel is called priority passing. The order of passing, which always gives second priority to a high-level channel, is as follows:

Assigned Priority Level		Next Priority Level		Next Priority Level
High	→	Middle	→	Low
Middle	→	High	→	Low
Low	→	High	→	Middle

When priority is passed to another level, that level is serviced and the fixed priority-level sequence is resumed with the next time slot. In field **A** of Figure 4-3, no high-level service requests are present before time slot 7. Thus, time slots 1, 3, and 5**,** which are normally granted to the high-level channels, are passed to the next privileged level. Time slot 1 passes priority to a requesting middle-level channel; time slot 3 passes priority to another middle-level channel, but time slot 5 passes priority to a low-level channel since no middle level channel is requesting service.

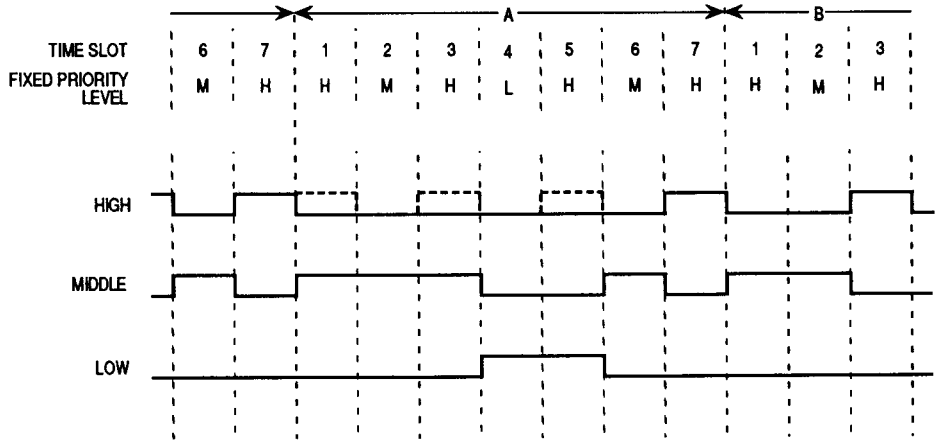


Figure 4-3. Priority Passing

A two-microcycle no operation (NOP) is introduced after channel service under the following condition: when the service request of a channel is recognized and the channel is the last one on a priority level whose service grant latch is negated. When that channel has been serviced, an NOP is executed. At all other times, service proceeds with the next time slot number without introduction of the NOP. This delay mechanism, necessary because of design timing constraints, allows the last channel serviced to be included in the next arbitration for new service on its assigned priority level. This mechanism is necessary for the secondary scheme.

4.1.1.2 Secondary Scheme – Priority among Channels on the Same Priority

Because channels can randomly request service, inevitably, channels having the same priority level will request service simultaneously. A secondary scheme prioritizes these requests. The scheduler services channels on each of the three priority levels, beginning with the lowest numbered channel on that level. It services all requesting same-level channels before clearing any of them for new service.

4.1.1.3 Correlation of Primary and Secondary Schemes

The overall priority scheme simultaneously incorporates both primary and, secondary schemes. Combining both schemes in the following example conveys their correlation.

1. Having its service request latch asserted, a single high-level channel requires service and is granted time slot 1, which has high-level priority. (Primary scheme) Once serviced, the channel's service grant latch is asserted. Next, both service latches, grant and request, are negated, and a two-microcycle NOP is executed.
2. The scheduler proceeds to time slot 2, which has middle-level priority; however, no middle-level channel is requesting service. Priority is then passed to the high level, but no high-level channel is requesting service; therefore, priority is passed again, and service is granted to the single requesting low-level channel. Once scheduled, this channel's service latches are negated, and a NOP is executed.
3. The scheduler resumes with the fixed-priority sequence on time slot 3; however, no channels are requesting service. An NOP is executed, and the scheduler remains at time slot 3 executing NOPs while waiting for requests.
4. Three high-level channels simultaneously request service. (Secondary scheme) The scheduler finds the lowest numbered high-level channel and assigns it to time slot 3, which has high-level priority. This channel's service grant latch is asserted; however, the two remaining high-level channels have asserted service request latches.
5. The scheduler continues to time slot 4, which has low-level priority, and allocates the slot to the lowest numbered low-level channel requesting service. (Primary scheme) The scheduler notes the still unserviced low-level channels and proceeds to time slot 5. (Secondary scheme resumes)
6. The next lowest numbered high-level channel is assigned to time slot 5, which has high-level priority. Noting the one remaining high-level channel, the scheduler continues to time slot 6.
7. However, time slot 6 has middle-level priority, and multiple middle-level channels are requesting service. (Primary scheme) The slot is allocated to
8. the lowest numbered mid-level channel. (Secondary scheme) The remaining mid-level channels are still unserviced, and the scheduler proceeds to time slot 7. (Secondary scheme resumes)
9. Having high-level priority, time slot 7 is allocated to the third high-level channel that initially requested service in time slot 4. This channel's service grant latch is asserted. The scheduler checks again. All service grant latches are asserted; therefore, all high-level channels have been allocated execution time. Under this condition, all service request and service grant latches of the high-level serviced channels are negated, and a NOP is executed.
10. The scheduler proceeds to time slot 1 again.

NOTE

Note that in numbers 6 and 7 multiple mid and low channels are requesting.

4.1.2 Time-Slot Latency

Latency is the amount of time between a service request and the beginning of service on that channel. The factors that affect latency are as follows:

- Number of active channels
- Number of channels on a priority level
- Number of available time slots on a priority level
- Number of microcycles required to execute a state of a time function
- TPU module clock frequency

Each time slot may require a different number of microcycles, depending on the state of a time function to be executed. This variation is graphically shown in Figure 4-4 where "microcycles" exemplify the number of microcycles executed for each state serviced.

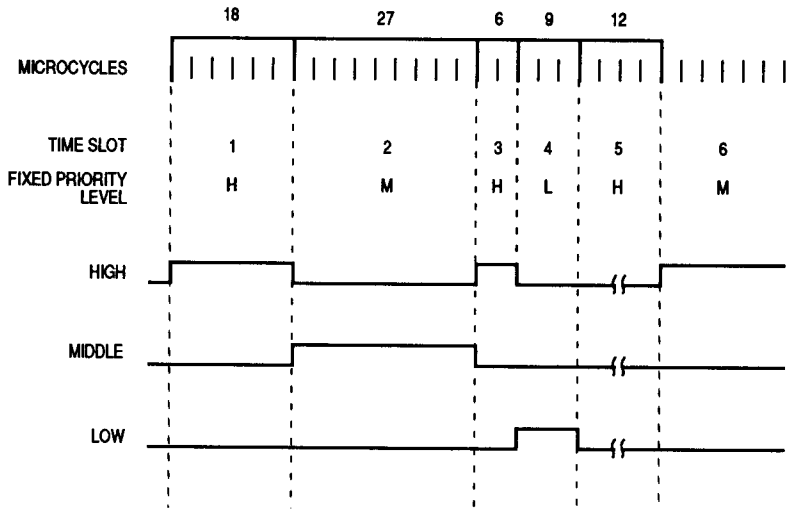


Figure 4-4. Time-Slot Variation

If the maximum time slot for each channel and the number of RAM accesses by the host CPU are known, the user can determine the maximum latency that will occur for each channel. This data and a method of calculating the worst-case latency are given in **APPENDIX D TIMING SUMMARY.**

4.2 Microengine

The microengine provides control over the execution unit and timer channels to synthesize time functions. Signals to the control points of the TPU are decoded from microinstructions.

In the discussion, the term microengine is restricted as an entity to the controlling elements within its physical location. The microengine actually functions through the interaction of the following elements: the control store, microprogram counter (μPC), branch programmable logic array (PLA), function select registers, return address register (RAR), decrementor (DEC), flag registers

in the arithmetic unit (AU), microinstruction register, microinstruction decode, and the emulation control.

The microengine can access the control store, a program memory array. This array contains the microinstructions necessary for the synthesis of time functions. Predefined time functions are in a ROM control store. Real-time emulation capability is provided by the MCU RAM module. An emulation mode allows up to **16** functions to be executed out of the RAM. Sequential access of the control store is provided by the μPC. The microengine can modify the sequential access based on a condition related to the channel receiving service.

The control store also contains entry points consisting of beginning addresses for the microcode sequences of the time functions as well as other data. An entry point is determined based on the channel function executing and the channel conditions. Branching within the microcode sequence is greatly reduced since the important channel conditions determine the entry point.

Any of the 16 time functions can be executed on a channel. The channel function select register (see **2.2.3 Channel Function Select Registers (CFSRO, CFSRI, CFSR2, CFSR3))** contains a field for each channel that specifies the function to be executed.

A state is constructed of a specific number of microinstructions that is not interruptible when executed by the microengine. This set of microinstructions is typically the code necessary to calculate the next phase of any single waveform to be input or output from a given channel. Prior to the execution of a state, a parameter necessary for the servicing of the channel is preloaded into the execution unit, thereby reducing the number of microinstructions required in each state.

The microengine provides several sequencing features. For example, one-level subroutining or repetitive execution of a single microinstruction for up to 17 times is possible. Another sequencing feature allows up to 16 microinstructions to be used as a subroutine, providing the capability of jumping (or subroutining) to a sequence of microinstructions, executing a programmable number of microinstructions in that sequence, and then returning to the original sequence. Sequence control is aided by a 4-bit decrementor located in the execution unit.

Immediate, memory direct, and memory indirect addressing modes are provided by the microengine. Both absolute and relative address calculation techniques may be used for the formation of the address. Absolute address calculation indicates that the operand is the address. Relative address calculation indicates that the operand field is arithmetically related to the channel number. Relative address calculation is very useful in designing reentrant microinstruction sequences that can be executed on any channel. TPU operands include parameter, link, channel, and decrement-count values.

Because of the restriction on access time of the control store due to emulation requirements, overlap fetching of the next microinstruction (pipelining) is incorporated to improve microengine performance by fetching the next microinstruction while the current microinstruction is being executed. Therefore, the next microinstruction, whose address is contained in the μPC, is ready for execution.

Programmable microinstruction flush control has also been incorporated to eliminate the microcoding restrictions associated with pipelining. The microinstruction following a branch or jump microinstruction may be executed either before the branch occurs or "flushed" to cause no action. Consequently, no microinstructions are wasted in performing an NOP following the execution of a branch microinstruction. If the microcode flow permits, the microinstruction

following a jump or branch microinstruction can be programmed for execution, regardless of the eventual outcome of the branch, using an otherwise wasted cycle.

4.2.1 Control Store

The control store, the program memory array of the microengine, is divided into two segments: the microcode and the entry points, as shown in Figure 4-5.

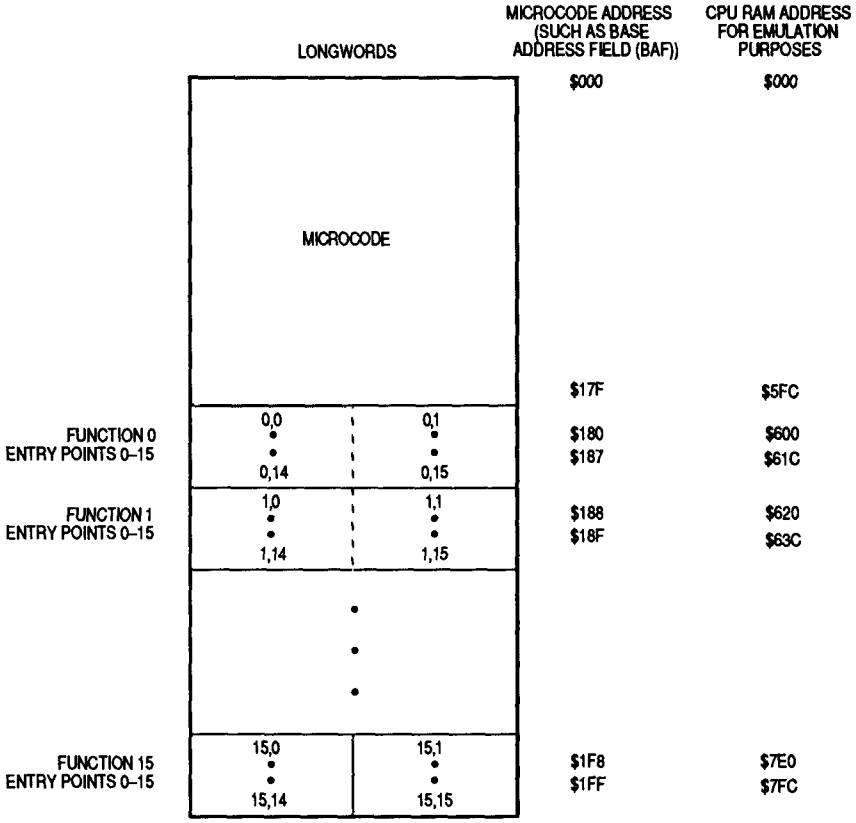


Figure 4-5. Control Store

4.2.1.1 Microcode Segment

This segment contains the microinstructions required to synthesize functions. Each microinstruction is 32 bits. Formats for the microinstructions are found in **APPENDIX B MICROINSTRUCTION FORMATS AND ENCODINGS**. Table 4-1 provides a quick reference that identifies all the fields and bits of the microinstructions.

Table 4-1. Microinstruction Bit/Field Quick Reference Guide

Bit/Field Mnemonic	Function	Format Number (Bits)				
		1	2	3	4	5
AID (6:0)	RAM Address/Immediate Field	8-2			8-2	
BAF (8:0)	Branch Address Field			24-16	24-16	
CJC	Conditional Jump Code			29-26		
BCF	Branch Conditional False			8		
BINV	AU B-Bus Invert Control	12	12			
CCL	AU Condition Latch Control	17				17

		Format Number (Bits)				
CCM	Channel Control MUX			2		
CIN	AU B-Bus Carry Control	13	13			
CIR	Channel Interrupt Request		2			2
DEC/END	Decrementor/End State Control	1-0	1-0		1-0	1-0
ERW	Event Register Write Control		29			
FLC	Flag Control		5-3	5-3	15-13	5-3
FLS	μPC Flush Control			25	25	
IOM	RAM Input/Output Mode Control	11-9			11-9	
LSL	Link Service Negation Latch Control		8		12	8
MRL	Match Recognition Negation Latch Control		17			
MTD	Match/Transition Detect Service Request Inhibit Control			1-0		
NMA	Next μPC Address Mode Control				27-26	
PAC	Pin Action Control		11-9	11-9		
PSC	Pin State Control		7-6	7-6		
RW	RAM Read-Write Control	29			28	
SHF	AU Shifter Control	20-19	20-19			20-19
SRC	Shift Register Control	18				18
T1ABS	T1 A-Bus Control	28-25	28-25			28-25
T1BBI	T1 B-Bus Immediate Data					16-9
T1BBS	T1 B-Bus Source Control	16-14	16-14			
T3ABD	T3 A-Bus Destination Control	24-21	24-21			24-21
TBS	Time Base Select Control			15-12		
TDL	Transition Detect Negation Latch Control		18			

4.2.1.2 Entry Point Segment

The entry points contain information about the state to be executed. The information includes a preload-parameter selection field, a preload-parameter destination field, a match enable field, and the beginning microcode address of the state. **A** total of 256 entry points exist, 16 points for each of the 16 possible time functions. Unused entry points may be used for microcode. Formatting of the entry points is given in **4.2.2 Entry Point Format.**

During the time-slot transition period, the μPC is initially updated with an entry point address formed by concatenating the function select field with an encoded version of the channel conditions (see **4.2.13 Time-Slot Transition Period).** This combination forms an address for accessing an entry point which, in turn, contains the address of the first microword of a state to be executed (see Figure 4-6). Encoding of the channel conditions is shown in **Table 3-1 Entry Points and Channel Conditions.** If multiple service request sources inclusive of a host service request are asserted when a channel receives a time slot, the host service request is granted higher priority and is serviced first. Another time slot is scheduled by the scheduler for the remaining service request sources.

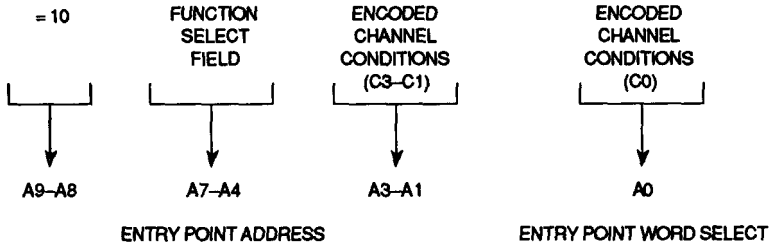


Figure 4-6. Entry Point Address Generation

4.2.2 Entry Point Format

Figure 4-7 illustrates the format content of the entry point segment.

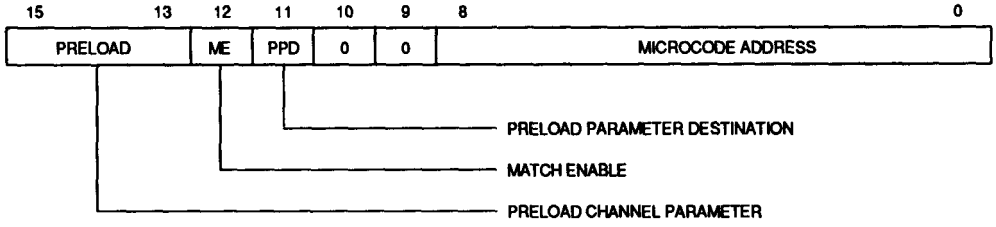


Figure 4-7. Entry Point Format

PRELOAD - Preload Parameter
This bit field indicates which channel parameter will be loaded into the execution unit prior to the execution of a state. If the parameter number is six or seven for those channels that only have parameters zero to five, the destination register (preload (P) or data input/output buffer (DIOB)) is loaded with $0000. Preloading occurs during the time-slot transition period (see **4.2.13 Time-Slot Transition Period).**

ME - Match Enable
ME specifies whether match events are enabled or disabled during execution of the state associated with the entry point. If they are disabled, a match event, which happens due to a value in the match register, occurs after channel service if the match register remains unchanged. For more details refer to **4.5.3.2 MATCH ENABLE LATCH (MEL).**
0 = Matches are disabled.
1 = Matches are enabled.

PPD - Preload Parameter Destination: DIOB or P Registers
PPD indicates the destination in the execution unit for the preloaded parameter. If bits 9 and 10 are 11, then no preload occurs.
0 = P register is destination for preloaded parameter.
1 = DIOB register is destination for preloaded parameter.

MICROCODE ADDRESS - Starting Microcode Address
This field contains the microcode address at which the state is to begin execution by loading this value into the μPC.
Xxxxxxxxx Starting microcode address ($000-$lFF)

4.2.3 Microprogram Counter (μPC)

The μPC contains the address used to access a microword from the control store for execution during the next microcycle. The address may be generated in one of five ways:

1. By the function select field (in the function select register) concatenated with an encoded version of the channel conditions for the channel to be serviced;

2. By a beginning microcode address from the control store;

3. By an incremented value of the μPC;

4. By an address field of a jump or branch microinstruction;

5. By the contents of the return address register.

As described in **4.2.1.2 ENTRY POINT SEGMENT,** the μPC is initially updated with the function select field during the time-slot transition period. Two microcycles later, the beginning address of the state, contained in the entry point, is loaded into the μPC and execution begins.

During execution of a state, the μPC is sequentially incremented unless the microcode calls for a change in sequence by a branch or use of a subroutine. A conditional branch, an unconditional jump to a new sequence, or a jump to a subroutine loads the μPC with the content of the branch address field (BAF). A return from subroutine (format 4) loads μPC with the content of the RAR.

When the content of BAF is loaded into μPC**,** flush control (FLS) is examined. FLS indicates whether the microinstruction following a jump or branch microinstruction is to be flushed. That is, it determines it a NOP is executed following a jump microinstruction or if the microinstruction following the branch microinstruction is executed. This feature provides for more efficient use of microcode space with consideration to pipelining, which is discussed in **4.2 MICROENGINE.**

4.2.4 Branch PLA

The decision-making capability of the microengine is provided through a conditional branch operation. As previously mentioned, the μPC is normally incremented. However, a branch may be desired based on some condition related to the channel being serviced. Conditions on the channel are referred to as branch conditions, and specification of one of these conditions is referred to as a branch condition code. The branch conditions are as follows:

- Match Recognition Latch
- Transition Detection Latch
- Link Service Latch
- Pin Status
- Host Sequence Bits (two)
- AU Flags: Carry, Negative, Overflow, and Zero

- Channel Flags (two)

Since the channels of the TPU are free running, some branch conditions are latched for input into the branch PLA to ensure proper servicing by the microengine. All branch conditions except pin status, AU flags, and channel flags are latched only during the time-slot transition period. The pin status is latched during the time-slot transition period and when the CHAN register is changed via microcode. Latching of the AU flags is controlled via microcode. The microcoder must latch the AU flags in the microcycle in which the calculation occurs since these flags change each microcycle. Lastly, the state of the channel flags currently executing are always input directly into the branch PLA. Therefore, the branch conditions latched into the branch PLA pertain only to the scheduled channel with one exception: when CHAN register is changed, the latches for pin state and the channel flags in the flag registers are updated for the new channel. Refer to **4.3.8.1 CHANNEL NUMBER REGISTER (CHAN)** for additional information.

Either the true or false state of a branch condition can be used for branching. As shown in the following table, this selection is specified by the branch condition set/clear field (BCF). When BCF = 1, the BAF of formats 3 and 4 microinstructions is loaded into the μPC if the branch condition defined by the conditional jump code (CJC) field is met. Otherwise, the μPC is incremented or loaded from one of the other sources. When BCF = 0, BAF is loaded into the μPC if the branch condition defined by CJC is not met. Otherwise, the μPC is incremented or loaded from one of the other sources.

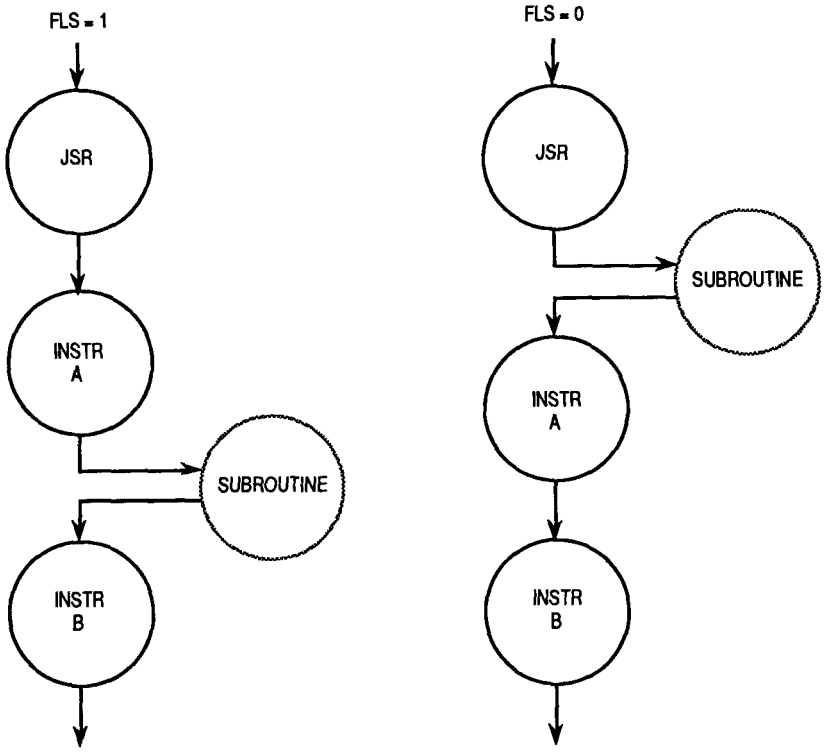
BCF	CJC	μPC Loaded With
0	Not Met	BAF
0	Met	μPC + 1
1	Not Met	μPC + 1
1	Met	BAF

4.2.5 Return Address Register (RAR)

RAR provides for one-level subroutining. When a subroutine jump is programmed with next-microword-address (NMA) mode control of format 4 microinstructions, a microinstruction address, modified according to FLS, is loaded into RAR.

When a subroutine return is executed or the decrementor decrements to one while in a special mode (see **4.3.8.2 DECREMENTOR (DEC)),** the μPC is loaded with the contents of RAR.

When a subroutine jump microinstruction is executed while FLS = 1, indicating that the next microinstruction is not to be flushed, RAR is loaded with μPC + 1. In this case, the first microinstruction executed upon return from the subroutine is the microinstruction following the unflushed (executed) microinstruction. When FLS = 0, indicating that the next microinstruction is to be flushed, RAR is loaded with μPC. Therefore, the first microinstruction executed upon return from the subroutine is the flushed microinstruction. The effect of the FLS bit on microinstruction flow is shown in the following illustration:



4.2.6 Decrementor

Refer to **4.3.8.2 DECREMENTOR (DEC)** for a description of the decrementor.

4.2.7 Flag Registers

Included within the microengine are two flag registers. Each contains one flag associated with each channel. Only the TPU may assert and negate these flags. The flag is addressed by the microcode flag-control (FLC) field, and the CHAN register. If CHAN is changed by microcode on a given microcycle, a flag-set or a flag-clear operation on the immediately following cycle will affect the flags of the previous channel addressed. Refer to **4.3.8.1 CHANNEL NUMBER REGISTER (CHAN)** for more information.

4.2.8 Microinstruction Register

The 32-bit register contains the microinstruction currently being executed.

4.2.9 Microinstruction Decode

The signals affecting the control points of the TPU are driven by this logic.

4.2.10 Emulation via RAM

Emulation for the TPU is provided with the use of the RAM module on the MCU. In emulation mode the RAM is accessed only by the TPU and contains state information, entry points, and microcode for up to 16 functions. Access to the RAM module through the intermodule bus (IMB) by a host is blocked. During normal operation, the RAM is

used for system purposes. A programmable switch is used to configure the RAM module in one of the two modes shown in Figure 4-8.

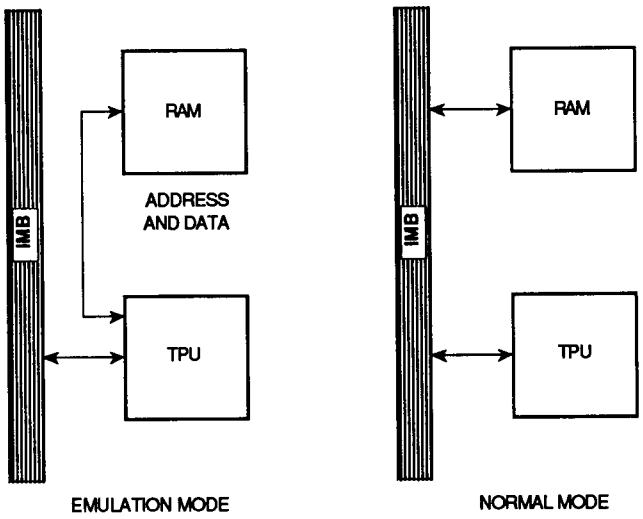


Figure 4-8. RAM Configuration

To utilize the emulation mode, the user must first load microcode into the RAM from the IMB through the RAM bus interface unit (BIU), and then assert the EMU bit in the TPU module configuration register. Assertion of EMU configures both modules for emulation mode, causing the MCU to function as the control store for the TPU. In emulation mode, only functions contained in RAM can be executed by the TPU. Conversely, in normal operation, only functions contained in ROM can be executed.

All operations and conditions that apply to the control store (ROM) also apply to the RAM when used for emulation. The memory map is the same as that shown in Figure 4-5; the timing is identical. The only difference is that the user can program the microcode and entry point segments of the RAM with functions.

4.2.11 Microinstruction Formats and Encodings

Due to the logistics of the emulation method, microinstruction size is limited to long word. Since the bit count for the microinstruction encodings is much greater, microinstruction formats that contain appropriate combinations of the encodings are necessary. Five microinstruction formats and their encodings are shown and defined in **APPENDIX B MICROINSTRUCTION FORMATS AND ENCODINGS.** Refer to Table 4-1 for a quick reference listing of the fields and bits.

4.2.12 Microengine Timing

The microengine uses four basic clocks for timing: T1, T2, T3, and T4. These clocks are derived from the system clock on the IMB. Each has a 25% duty cycle and a period twice the system clock period. The sequential occurrence of these four T states (T1-4) constitutes a microcycle. Two additional T states are derived from the system clocks: T2\ and T4\. The T2\ state occurs when the TPU loses RAM arbitration to a bus master (see **4.4 RAM OPERATIONS).** The T4\ state occurs due to a breakpoint or FREEZE condition (see **4.6 TPU MICROCODE DEVELOPMENT SUPPORT).**

A T2\ or T4\ state is defined as a wait state of one system clock period in which the T clocks continue to run, but the control points associated with the clocks are unaffected. That is, no operation occurs during this state. For illustration, T2\ and T4\ states are typical examples of the bus wait state. Both T2\ and T4\ states occur in multiples of two system clocks (see Figure 4-9) to keep the microengine synchronized with the free-running channels. Due to the pipelining of microinstructions, latches are required for T3 and T4 controls.

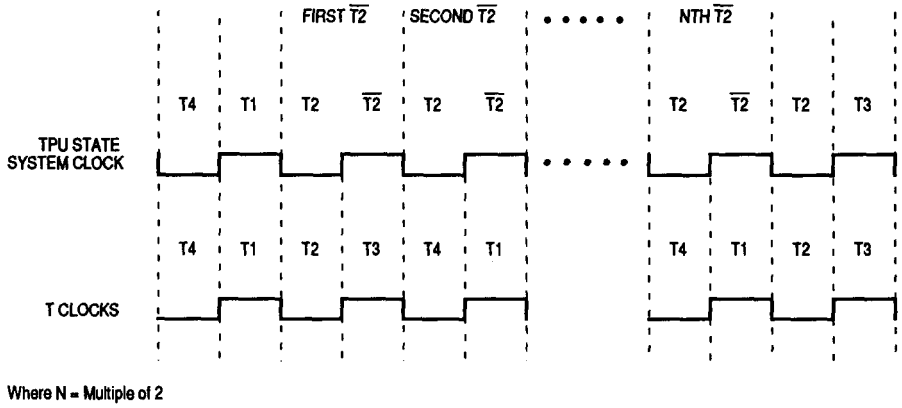


Figure 4-9. Wait States

Basic timing for some of the functional units is shown in Figure 4-10. Figures 4-11 and 4-12 illustrate the timing for these units in T2\ and T4\ states. Note that the timing shown is for registers where cross-hatched segments indicate register loads.

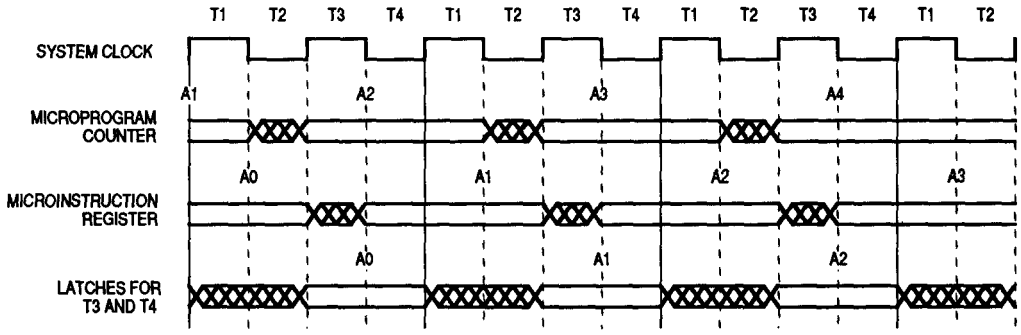


Figure 4-10. Microengine Timing

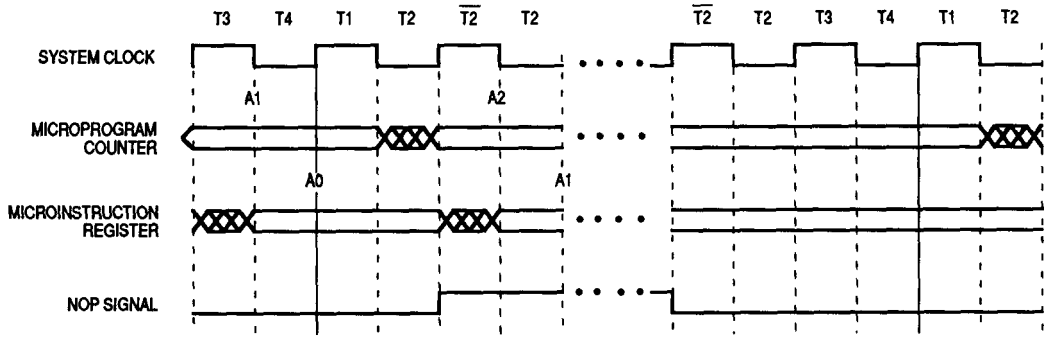


Figure 4-11. T2\ Timing

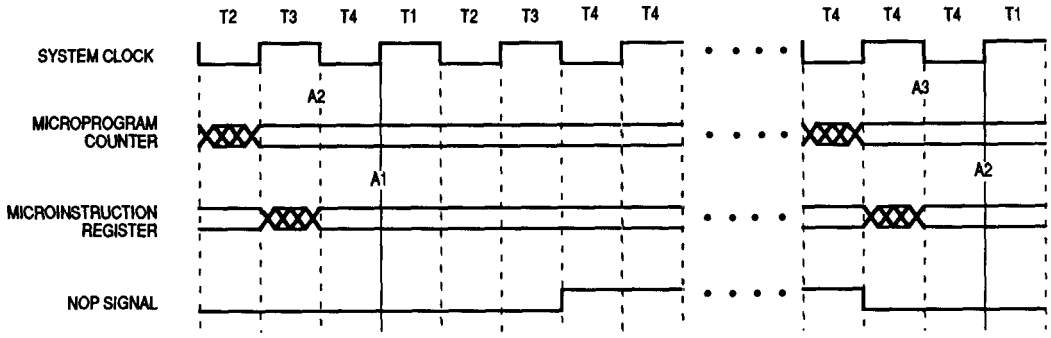


Figure 4-12. T4\ Timing

4.2.13 Time-Slot Transition Period

The time-slot transition period is a five-microcycle (10-clock) period between the servicing of channels. The primary tasks completed in this period are as follows:

- Update of the CHAN register with the number of the new channel to be serviced
- Programmable update of the match register of the previous channel with ERT (specified by last microinstruction executed)
- Update of the ERT with the capture register of the new channel
- Latching of the branch conditions of the new channel to be serviced
- Formation of the address of the entry point location
- Access of the entry point location
- Access of the first microinstruction of the state to be executed for the new channel
- Preload of a parameter

During the time-slot transition period, an NOP signal negates all signals from the decode; hence, no microinstructions are executed during the transition. Refer to Figure 4-13 for the timing of the time-slot transition period.

4.2.14 Microengine Data References

Table 4-2 describes the microengine tasks that require data references. This table describes the task, indicates the location of the operand, provides the data type and size, and identifies the task name.

The change channel, link, and set decrementor tasks are programmed with the T3ABD microinstruction field of formats 1, 2, and 5. Selection of the channel, link, or decrementor register as the T3 A-bus destination will result in the respective operation. However, other execution unit microinstruction fields are necessary for specifying the A-bus and B-bus sources, the arithmetic operation, etc. A parameter access is programmed with the microinstruction fields IOM and AID of certain microinstructions.

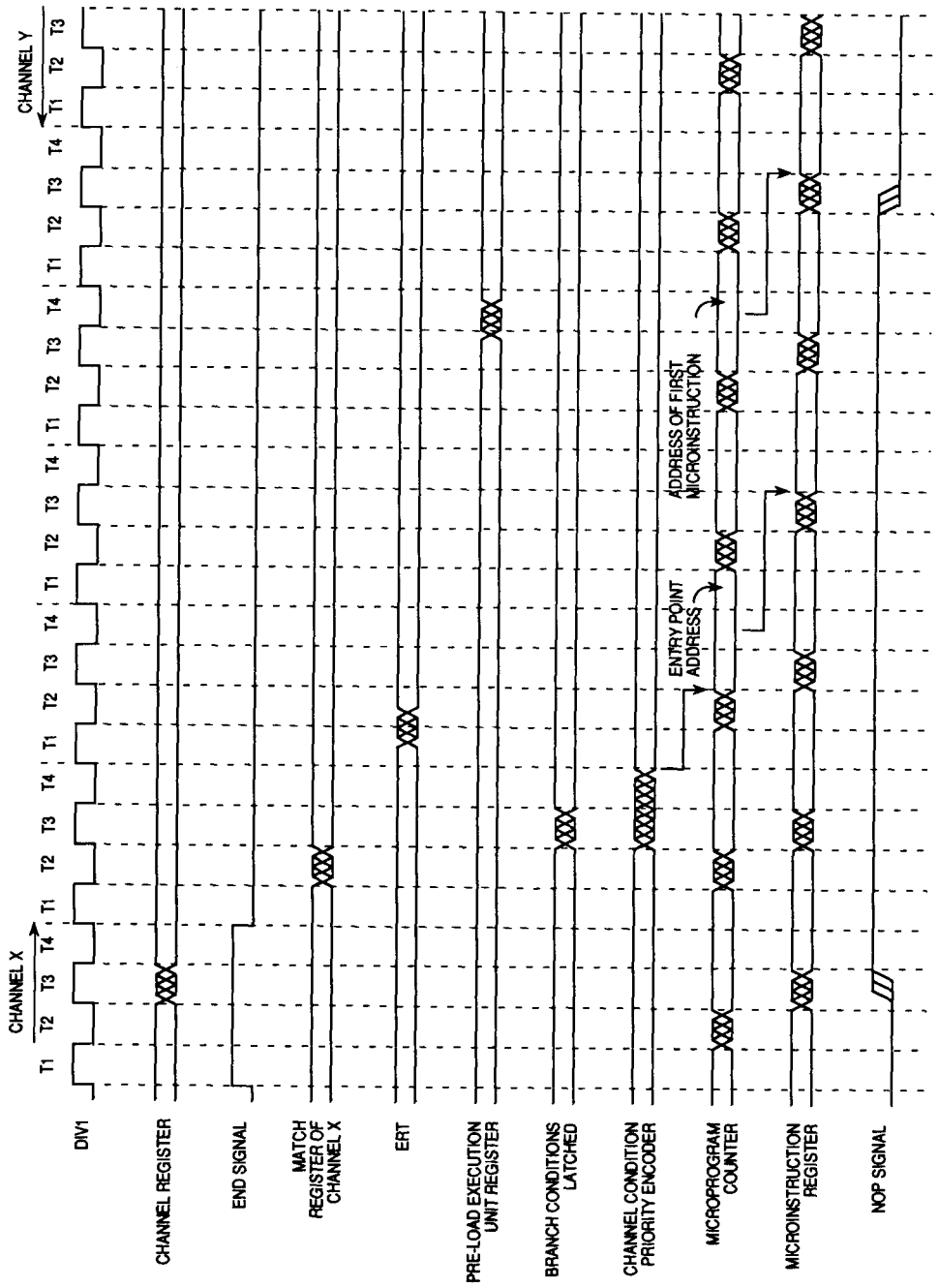


Figure 4-13. Time-Slot Transition Period Timing

Table 4-2. Microengine Data References

Tasks	Address Mode	Data Type	Data Size	Name
CHANGE CHANNEL:				
Change to Channel Indicated by the Contents of Immediate Field	Immediate	Channel Number	4	Immediate Channel Operation
Change to Channel Offset from the Current Channel. The Offset is Contained in Immediate Field	Immediate	Channel Offset	4	Immediate Channel Relative Operation
Change to Channel Indicated by the Contents of a Parameter Register	Memory Direct*	Channel Number	4	Direct Channel Operation
Change to Channel Offset from the Current Channel. The Offset is Contained in a Parameter Register	Memory Indirect*	Channel Offset	4	Direct Channel Relative Operation
LINK:				
Link to Channel Indicated by the Contents of Immediate Field	Immediate	Channel Number	4	Immediate Link Operation
Link to Channel Offset from the Current Channel. The Offset is Contained in Immediate Field	Immediate	Channel Offset	4	Immediate Link Relative Operation
Link to Channel Indicated by the Contents of a Parameter Register	Memory Direct*	Channel Number	4	Direct Link Operation
Link to Channel Offset from the Current Channel. The Offset is Contained in a Parameter Register	Memory Indirect*	Channel Offset	4	Direct Link Relative Operation
PARAMETER ACCESS:				
Access Parameter Indicated by the Address in Immediate Field	Memory Direct*	Parameter	16	Direct Parameter Access
Access Parameter Indicated by the Address Formed with the Concatenation of Immediate Field and the Channel Number	Memory Direct*	Parameter	16	Direct Channel Relative Access
Access Parameter Indicated by the Address Contained in a Parameter Register	Memory Indirect*	Parameter	16	Indirect Parameter Access
SET DECREMENTOR:				
Set Decrementor as Specified by the Contents of an Immediate Field	Immediate	Decrement Count	4	Immediate Decrementor Set
Set Decrementor as Specified by the Contents of a Parameter Register	Memory Direct*	Decrement Count	4	Direct Decrementor Set

*All tasks using the memory direct and indirect addressing modes, with the exception of memory direct access of parameters, require two microcycles for completion.

4.3 Execution Unit

The execution unit consists of a number of registers, functional units, and data paths. These elements and certain decision-making capabilities that exist within the microengine evaluate and control the resources associated with each channel to synthesize time functions. Resources accessed by the execution unit include the match and capture registers, test configuration registers (TCR1 and TCR2), channel number register, link logic, and parameter RAM. Refer to Figure 4-1 for a block diagram of the execution unit.

The basic timing unit for the execution unit is a microcycle (see **4.2.12 Microengine Timing).** In one microcycle the execution unit can:

- In T1, source one or two operands from various registers onto the two internal buses;
- Send the operands to the AU which, in T2, adds or subtracts them and generates one result;
- In T3, pass the result through the shifter unshifted or, with a 1-bit shift or rotate, send the shifter result back onto one of the internal buses and into a destination register;
- In T4, write the result into a register or into parameter RAM.

Two internal buses (A bus and B bus) within the execution unit transfer data between the registers and functional units. Depending upon the function, each of the registers and functional units can read and/or write one or both of the internal buses. In addition, some of the registers have access to buses, which access various resources outside of the execution unit, e.g., parameter RAM, link logic, etc. Both internal buses and most registers and functional units within the execution unit are 16 bits wide. The following registers and functional units are discussed in subsequent paragraphs.

- Arithmetic Unit (AU) - 16 bits
- Arithmetic Unit Shifter (AU shifter) - 16-bit shift and rotate
- Shift Register (SR) - 16 bits
- Preload Register (P) - 16 bits
- Data Input/Output Buffer (DIOB) - 16 bits
- Accumulator (A) - 16 bits
- Event Register Temporary Register (ERT) - 16 bits
- Decrementor (DEC) - 4 bits
- Channel Number Register (CHAN) - 4 bits
- Encoded Link Register (LINK) - 4 bits

The formats and encodings of the microinstructions used to accomplish these functions are listed in **APPENDIX B MICROINSTRUCTION FORMATS AND ENCODINGS.** The operations performed are summarized as follow:

- A-bus byte operand source may be P (either upper or lower byte), DEC, CHAN, or zero. Word operand source may be P, A, SR, DIOB, TCR1, TCR2, ERT, or zero.
- B-bus word operand source may be from P, A, SR, DIOB, or constants, $8000 or $0000.
- A-bus destination may be A, SR, ERT, DIOB, P (either byte or full word), LINK, CHAN, DEC, TCR1, TCR2, or no destination.

4.3.1 Arithmetic Unit (AU)

The arithmetic functions that the AU performs include add and subtract of 8- or 16-bit operands. The data to be operated upon is placed on the internal buses during T1. Two 16-bit latches, one connected to each of the two internal buses, retain the data for AU operation in T2. The result of the AU operation is placed on the A bus during T3 to be loaded into a destination register.

Constants, which may be generated for input to the AU, include one ($0001), zero ($0000), negative one ($FFFF), and $8000. The operations performed by the AU are as follows:

0	+ 0	Clear
0	+ BBus	Pass B
0	+ BBus + CIN	Increment B
0	+ BBus\	Ones complement of B
0	+ BBus\ + CIN	Negate B
ABus + 0	Pass A	
ABus + 0\ + CIN	Increment A	
ABus + 0\ ($FFFF)	Decrement A	
ABus + $8000	A + Max. parameter offset	
ABus + BBus	Add (see Note 2)	
ABus + BBus\ + CIN	Subtract	

NOTES:

1. BBus\ = Ones complement or inversion
2. If, and only if, the SR and the AU shifter are **both** enabled to shift and DEC is decrementing, will the B-bus input to the AU be the content of the B-bus or zero as determined by the least significant bit of SR = 1 or 0 respectively. This operation effects a multiply of the B-bus by the content of the SR.

Certain status flags from an AU operation may be utilized by the microengine for the purpose of branching. These flags include a zero result (Z), a negative result (N), an overflow result (V), and a carry out (C).

AU flags are calculated based on either both bytes of the result or the lower byte of the result, depending on whether a word or byte operation is executed.

- The C and V flags always reflect the result of the AU.
- Z and N always reflect the output of the AU shifter.

For example, if the AU result is shifted right in the shifter, C and V are calculated based on the AU result, and Z and N are calculated based on the right-shifted AU result output.

For AU result flag calculation, byte operations are distinguished as follows:

- The destination is a byte of the P register; or

- The sources are a byte of a register and an immediate value, with or without a destination specified, i.e., the encodings for TLABS are 0000, 0001, 0010, 0011, 0111, or 0100 with T1BBI specifying the appropriate immediate data.

Otherwise, the AU operation is considered to be word. Also, if the AU shifter is shifting (either right, left, or rotate) then the C calculation is considered to be a word operation.

Z and N flags are calculated as follows:

$$Z = S0 \bullet S1 \bullet S2 \ldots \bullet Sm$$
$$N = Sm$$

where:

S0, S1 ... Sm = AU-shifter output

m = AU-shifter output bit 15 for a word operation or AU-shifter output bit 7 for byte operation

Figure 4-14 shows C and V calculation for byte/word AU operations.

AU Shifter	AU Operation	Flag Calculation
No Shift	Add	C = Am • Bm | $\overline{Rm}$ • Am | Bm • $\overline{Rm}$ V = Am • Bm • $\overline{Rm}$ | $\overline{Am}$ • $\overline{Bm}$ • Rm
	Subtract	C = $\overline{Am}$ • Bm | Rm • $\overline{Am}$ | Bm • Rm V = Am • $\overline{Bm}$ • $\overline{Rm}$ | $\overline{Am}$ • Bm • Rm
Left Shift	Add	C = R15 V = Am • Bm • $\overline{Rm}$ | $\overline{Am}$ • $\overline{Bm}$ • Rm
	Subtract	C = R15 V = Am • $\overline{Bm}$ • $\overline{Rm}$ | $\overline{Am}$ • Bm • Rm
Right Shift	Add	C = R0 V = Am • Bm• $\overline{Rm}$ | $\overline{Am}$ • $\overline{Bm}$ • Rm
	Subtract	C = R0 V = Am • $\overline{Bm}$ • $\overline{Rm}$ | $\overline{Am}$ • Bm • Rm
Right Rotate	Add	C = R0 V = Am • Bm • $\overline{Rm}$ | $\overline{Am}$ • $\overline{Bm}$ • Rm
	Subtract	C = R0 V = Am • $\overline{Bm}$ • $\overline{Rm}$ | $\overline{Am}$ • Bm • Rm

Where:
- Am = A-bus bit 15 for a word operation or A-bus bit 7 for a byte operation
- Bm = B-bus bit 15 for a word operation or B-bus bit 7 for a byte operation
- Rm = AU output bit 15 for a word operation or AU output bit 7 for byte operation
- R15 = AU output bit 15
- R0 = AU output bit 0
- $\overline{Xm}$ = Inversion/complement

Figure 4-14 . Carry and Overflow Calculation for Word/Byte Operation

4.3.2 AU Shifter

The output of the AU reaches the A bus through a 16-bit shifter. This shifter allows for shifting or rotating of the output by one bit position before being gated onto the A bus. Normally, the shifter passes the output of the AU to the internal A bus without performing a shift or rotate operation. If a shifter option is selected, the procedure used for filling the vacant bit is described as follows:

- When a left shift is selected, a binary zero will be gated onto bit 0 of the A bus.
- When a right-shift is selected, then the least significant bit of the AU output will be available to the input of the SR bit 15 (SR15). If the SR is also shifting, then carryout from the AU will be gated onto bit 15 of the A bus. Otherwise, a binary zero will be gated onto bit 15 of the A bus.
- When a right rotate is selected, bit 0 of the AU output will be shifted onto bit 15 of the A bus.

The shifter, therefore, includes these functions:

- T3 No shift
- T3 Right shift AU carryout onto A15
- T3 Right rotate AU0 onto A15
- T3 Left shift Zero onto A0

4.3.3 Shift Register (SR)

SR is a 16-bit general-purpose register that also may shift right one bit position per microcycle. When SR is enabled for shifting and the AU shifter is also enabled to shift right, the least significant bit of the AU shifter output is shifted into SR15, effecting a 32-bit shift.

In the combined condition when the SR is shifting, the AU shifter is right shifting, and DEC is decrementing, the least significant bit of SR will control the B-bus input to the AU. If SR0 = 1, the contents of the internal buses are added. If SR0 = 0, the content of the A bus is passed, i.e., add zero to A bus.

The SR performs the following functions:

- T1 Read (output) to A0-A15.
- T1 Read (output) to B0-B15.
- T3 Write (input) from A bus.
- T3 Right shift; (no bus in/out)

  If AU shifter not shifting, SR15 = 0

  Else SR15 = least significant bit of AU shifter.

4.3.4 Preload Register (P)

P, a 16-bit general-purpose register, interfaces to the parameter RAM and to certain control latches of the channels as a data register with read and write capability. P may be preloaded during the time-slot transition period with one of the parameter RAM locations associated with the channel requesting service. The parameter, which is preloaded, is described in **4.2.2 Entry Point Format.**

The contents of P can be a source to either the A bus or B bus, and P can be a destination for the A bus. In byte operations, either byte of P can be a source or a destination to the lower byte of the A bus.

- T3 Write (input) from A7-A0.
- T3 Write (input) P7-P0 from A15-A0.

- T3 Write (input) P15-P8 from A7-A0.
- T1 Read (output) P7-P0 to A7-A0 (A15-A8: = 0).
- T1 Read (output) P15-P8 to A7-A0 (A15-A8: = 0).
- T1 Read (output) P15-P0 to B15-B0.
- T4 Read RAM to P15-P0.
- T4 Write P15-P0 to RAM.
- T2 Write P8-P0 to channel configuration control latches.

4.3.5 Data Input/Output Buffer Register (DIOB)

DIOB, a 16-bit general-purpose register, is also used by the execution unit to interface to the parameter RAM. It interfaces as both a data register, with read and write capability, and an address register. Bits 7-1 can be used to address the parameter RAM. In this usage, parameter registers can be read from the RAM into either P or DIOB and written to the RAM from P.

DIOB or P may be preloaded during the time-slot transition period with one of the parameter RAM locations associated with the channel requesting service. The parameter, which is preloaded, is described in **4.2.2 Entry Point Format.**

- T1 Read (output) to A15-A0.
- T1 Read (output) to B15-B0.
- T3 Write (input) from A15-A0.
- T4 Read RAM, write to DIOB.
- T1 Write RAM, read from DIOB (T1 of next microcycle).
- T2 DIOB7-DIOB1 used as RAM address.

4.3.6 Accumulator Register (A)

A, a general-purpose 16-bit register in the execution unit, has access to both internal buses. Its primary function is as an accumulator for storing the result of the AU operation.

- T3 Write (input) from A bus.
- T1 Read (output) to B bus.
- T1 Read (output) to A bus.

4.3.7 Event Register Temporary (ERT) Register

ERT contains a copy of the capture register for the channel currently being serviced. Before the servicing of a channel or when the channel to be serviced is changed via microcode, the content of the selected channel's capture register is automatically written into ERT. Via microcode, ERT can also be loaded with the match register contents of a channel, or ERT can load the match register of a channel. When a new channel number is written into CHAN via microcode to service a different channel, the contents of ERT represent the **old** channel for the next two microcycles and represent the **new** channel on the third microcycle following the change channel operation. Availability of various TPU attributes following a change of CHAN contents is presented in Table 4-3.

Table 4-3. Change of CHAN Register

Feature Used	Microcycle n	n + 1	n + 2	n + 3
Write Channel Register	Chan := xxxx	New	New	New
RAM Commands Using Chan	Old	New	New	New
Branch Using Pin State or Channel Flags	Old	Old	New	New
Branch on All Other Conditions	Old	Old	Old	Old
ERT Value	Old	Old	Old	New
Channel Commands: Neg_MRL, Neg_TDL, TBS, PAC, PSC	Old	New	New	New
Channel Command Write MER	Old	Do Not Write	New	New
Channel Command Read MER	Old	Old	Do Not Read	New
Channel Commands Set/Clear Channel Flags, etc.	Old	Old	New	New
Negate Link, Set PIR	Old	Old	Old	Old

- T3 Write (input) from A bus.
- T1 Read (output) to A bus.
- T2 Copy ERT into capture register.
- T2 Copy match register into ERT.
- T2 Copy capture register into ERT.

4.3.8 Registers Affecting Microengine

Three registers, CHAN, DEC, and LINK, allow the execution unit to alter the flow of the microengine.

4.3.8.1 Channel Number Register (CHAN)

CHANNEL NUMBER REGISTER (CHAN). During the time-slot transition period, this 4-bit register is initialized by the scheduler to the number of the channel receiving the time slot. CHAN can also be changed via microcode to initiate a sequence of events. One such event is the update of ERT with the capture register of the new channel selected. The new pin state and

contents of ERT are valid on the third microcycle following the channel change (see **4.2.4 Branch PLA** and Table 4-3).

CHAN is used by the microengine for addressing the parameter RAM and for relative linking, in addition to the expected use of selecting channel for service. it is accessed on bits 7-4 of the A bus and is initialized to $0 at reset.

- T3 Write (input) from A7-A4.
- T1 Read (output) to A7-A4.
- T1 Write (input) from scheduler.

4.3.8.2 Decrementor (DEC)

Consists of a 4-bit counter with control. DEC is accessed on bits 3-0 of the A bus and may be loaded with a value from $0 to $F; $1-$F represents values 1 to 15, respectively; $0 represents 16. When decrementing, DEC decrements once per microcycle.

When DEC is loaded with a value and started, it decrements to one, indicates this occurrence to the microengine for interpretation, and is then set to $F, which ensures that DEC is always preset to $F for the next usage. During the time-slot transition period and at reset, DEC is also initialized to $F.

DEC has two modes of operation. In the repeat mode when DEC is enabled to decrement, the μPC does not increment, which allows for the repeated execution of a microinstruction that results in shifting or some other repetitive operation. The number of times a microinstruction is executed in this mode equals the initial value of DEC + 1; therefore, values from $1 to $F repeat 2 to 16 times, and $0 repeats 17 times.

In the subroutine mode, the μPC continues to count normally while DEC decrements but, when DEC reaches one, the return address register contents are copied into μPC. This mode allows exiting from a subroutine after a predetermined number of microcycles have been executed. In this mode, DEC can be changed within the subroutine to effect an earlier or later exit. The number of microcycles for which microinstructions can be executed in this mode equals the initial value of DEC; therefore, values from $1 to $F execute 1 to 15 microcycles, and a value of $0 executes 16 cycles.

- T3 Write (input) A3-A0 to DEC3-DEC0.
- T1 Read (output) DEC3-DEC0 to B3-B0 (B15-B4 := 0).
- T4 Decrement.

4.3.8.3 Encoded Link Register (LINK)

LINK is a 4-bit register used for generating "links" by asserting the link-service-request bit of a channel. Writing an encoded channel number to LINK initiates a procedure in which the link.

service-request bit is asserted for the channel number written to LINK. This register is initialized to $0 at reset.

- T3 Write (input) from A7-A4.
- T1 Read (output) to A7-A4.
- T1 Read (output) to scheduler.

4.3.9 Miscellaneous

The execution unit can also use 8-bit immediate values supplied by the microinstruction. Immediate values are placed directly onto the B bus during T1. TCR1 and TCR2 are also readable by the execution unit for arithmetic computations with parameters and are writable for testing purposes. These Miscellaneous functions are as follows:

- T1 Immediate value from control store to B7-B0 (B15-B8 := 0).
- T1 Read TCR1 15-0 to A15-A0.
- T1 Read TCR2 15-0 to A15-A0.
- T3 Write A15-A0 to TCR1 15-0.
- T3 Write A15-A0 to TCR2 15-0.
- T3 Write A7-A4 to link logic.

4.4 RAM Operations

Transfers may be performed between the parameter RAM registers and the execution unit registers, P and DIOB, by using encoding in the RW, IOM, and AID fields of the microword. A load from RAM to P/DIOB is prefetched in one microcycle for execution unit operations in the next microcycle. Execution unit operations are completed in T3, allowing a result to be stored into RAM at T4 of the same microcycle.

Operations performed between the RAM and P or DIOB are as follows:

- Load (write) or store (read) P using RAM address formed by CHAN register concatenated with AID (2:0).
- Load (write) or store (read) P using RAM address from AID (6:0).
- Load (write) or store (read) P using RAM address from DIOB (7:1).
- Load (write) or store (read) DIOB using RAM address formed by CHAN concatenated with AID (2:0).

- Load (write) or store (read) DIOB using RAM address from AID (6:0).
- Load (write) or store (read) DIOB using RAM address from DIOB (7:1).

4.4.1 RAM Accesses

Access to the parameter RAM is shared between the TPU and IMB bus masters, such as the CPU. Since the parameter RAM may be accessed by only one source at a time, an arbitration scheme prevents simultaneous accesses but allows access to all requesters in turn. In general, long-word CPU accesses are coherent, and every successive pair of TPU accesses is also coherent. The following rules regulate RAM access:

- The TPU asserts the TPU RAM request latch during the T2 state.
- The TPU negates or maintains the request during the subsequent T2 states.
- The TPU arbitrates during either T2 or T4 when the RAM is not being accessed.
- The TPU is granted access during a T2 state.
- The IMB master requests access for B1-B4 cycle during the B1 state.
- The IMB master maintains request until the subsequent B4 state.
- The IMB master arbitrates during B1 or B3 when the RAM is not being accessed.
- The IMB master is granted access during a B3 state.
- The IMB master does not relinquish access until one data transfer (word or long word) is complete.
- While the TPU is waiting for access, it executes T2\ wait states. The TPU always waits a multiple of two wait states.
- While the IMB master is waiting for access it executes B3\ wait states.

The CPU has priority for the RAM when:

1. The TPU has completed a data transfer during the last access; or
2. The RAM was not being accessed during the last arbitration period; or
3. The CPU is arbitrating for the second word access of a long-word transfer.

The TPU has priority for the RAM when:

1. The CPU has completed a data transfer during the last access; or

2. The TPU is arbitrating for the second access of a data transfer in which no TPU access occurred in the microcycle prior to the first access of the current data transfer. (A data transfer is defined as word or long-word access.)

Many scenarios exist for accessing the parameter RAM, several of which are shown in Figures 2-5a and 2-5b. These figures illustrate word accesses by a host, such as the CPU, and word or long-word accesses by the TPU.

4.5 Channel Control Operations

The channel control hardware is configured by the microengine to determine the relationship between the activity at each channel pin and the two TCRS. Figure **4-15** illustrates a functional block diagram of the channel control hardware. Reference to this diagram will help clarify the interaction of the event register, control signals, the 1/0, and other channel features. Microinstruction formats and encodings are found in **APPENDIX B MICROINSTRUCTION FORMATS AND ENCODINGS.**

4.5.1 Channel Operation

The major features of a TPU channel are presented in the following list, which introduces some new terminology. Most new terms are discussed in detail on the following pages.

- The event register can be accessed once per microcycle.
- Assertion of the transition detect latch inhibits the assertion of the match recognition latch until the transition detect latch is negated.
- Service requests caused by the assertion of the match request latch or transition detect latch may be blocked, allowing input transition capture events to be received at a very high rate.
- The value of the event register temporary (copy of event register) is correlated to the state of the transition detect latch and/or the match request latch and to the pin state.
- All inputs must be valid for at least four clock periods after transition to be recognized as valid. (Inputs are debounced with synchronizers and digital filtering to reject inputs less than four clocks in duration.)
- The match request latch has an interlock mechanism requiring a write to the match register of the event register before additional match events can assert the match request latch.
- To minimize microcode when changing pending matches, assertion of the match request latch may be inhibited during service. This latch may be selectively enabled to allow assertion during the time slot by a bit in the entry point on a per-state basis.
- Update of the match event register and negation of the match request latch, transition detect latch, and the link-service-request sources are done coherently to prevent inadvertent matches.

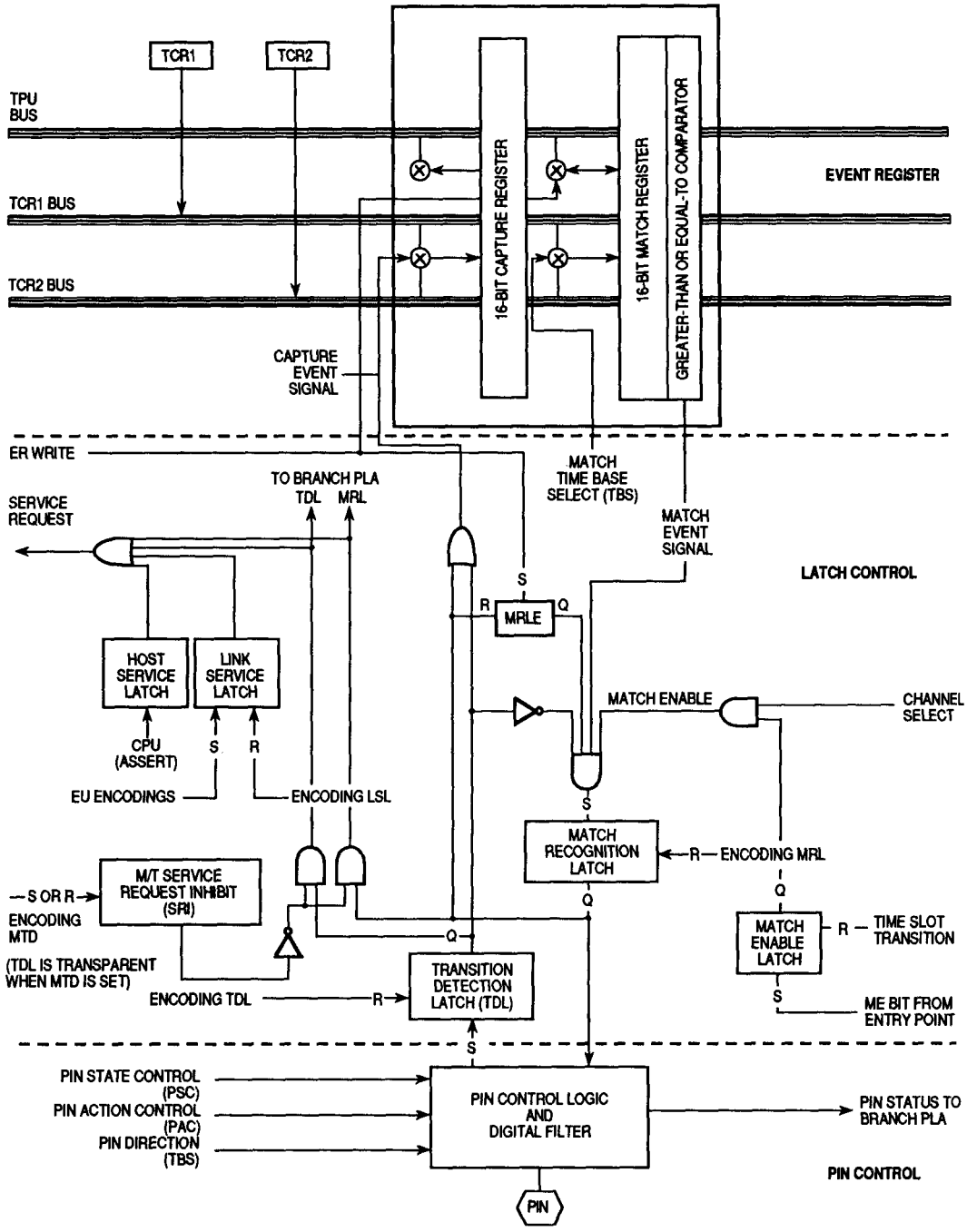


Figure 4-15. Channel Block Diagram

Each channel is composed of three logical sections: the event register, the latch control, and the pin control. They perform the following functions:

1. The event register is the hardware through which the microengine references the time associated with an event on a channel.

2. Latch control retains information concerning the occurrence of events on a channel. It also issues service requests for that channel to the scheduler. The microengine accesses the latch control during a time slot to determine the state of the channel, to control the occurrence of events, and to control the negation of certain latches within a channel.

3. Pin control is the hardware through which a capture event is interpreted from a specified pin action or a match event is translated into a specified pin action.

The microengine is synchronized with channel timing when the channel is accessed by the microengine. The microcycle consists of four ticks, T1-T4, or two TPU system clocks. The resolution period (equivalent to two microcycles) consists of four resolution states (RS1-RS4). Channel timing is shown in Figure 4-16.

Table 4-4 shows the reset state for various channel controls and latches. Refer to **Table 2-3 CHANNEL CONTROL Options** for pin state control and pin action control details.

Table 4-4. Reset State for Channel Controls and Latches

Channel Controls and Latches	Reset State
Match Recognition Latch (MRL)	Negated
Match Recognition Latch Enable (MRLE)	Negated Inhibits Assertion of MRL
Transition Detect Latch (TDL)	Negated
Match/Transition Service Request Inhibit (SRI) Latch	Asserted Inhibits MRL and TDL Service Requests, (But Not Assertion of MRL or TDL Latch)
Match Time Base	Use TCR1
Capture Time Base	Use TCR1
Pin Action Control (PAC)	No Transition Will Be Detected
Pin Directionality	Input

4.5.2 Event Register

Each channel has an ER, which is composed of one 16-bit greater-than-or equal comparator, one 16-bit compare/match register, and one 16-bit capture register. ERs are capable of performing two kinds of events: match and capture. The ER comparator is used to perform a greater-than-or-equal compare of a specified TCR against the contents of the match register to generate match events. The ER capture register is used to capture a specified TCR as a result of a specified input transition or a match event. The match register may be either read or written; whereas, the capture register may only be read. The CPU may not directly access an ER. CPU access of the information in ERs is provided through the parameter registers by the particular time function executing on the channel.

To provide for simultaneous match and capture on all channels, two TCR buses traverse all channels. In addition, a TPU ER bus runs through all channels to provide microengine access to the ER of each channel.

4.5.2.1 TCR Interface

All events are associated with a known time base. In the TPU, time is traced by referencing all events to either of two free-running TCRS. TCR1 is clocked from the output of a prescaler that

allows clocking rates from (system clock/4) to (system clock/256). TCR2 is clocked from the output of a prescaler that is clocked by an external source via the TCR2 pin. This clocking is synchronized to the RS1 resolution state after a rising edge at the TCR2 pin as shown in Figure 4-16. A minimum delay of 13 ticks (6.5 system clocks) to a maximum delay of 17 ticks (8.5 system clocks) is required from the time a rising-edge transition is detected at the TCR2 pin until the timer TCR2 is incremented, assuming the TCR2 prescaler is configured to divide by one. Alternatively, the TCR2 pin can act as a gating signal in which an internal clock drives the prescaler associated with TCR2 whenever the TCR2 pin is high. TCR1 and TCR2 are gated onto their appropriate bus during T2 of RS1. Because of the subtleties of bus timing, if a TCR is written in one microcycle and read in the next, the **old** value is read. Inputs to the TCR2 pin must be valid for at least four clock periods after transition to be recognized as valid. (Inputs are debounced with synchronizers and digital filtering to reject inputs less than four clocks in duration.)

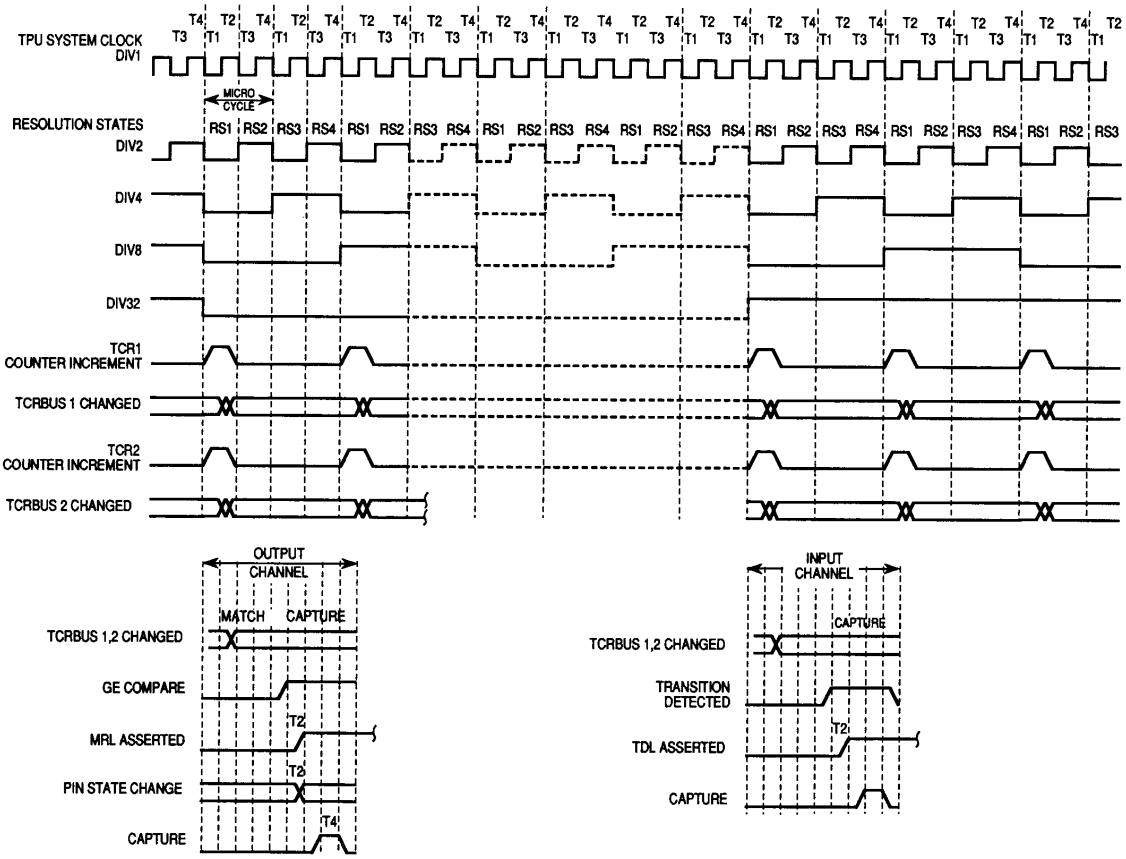


Figure 4-16. Channel Control Timing

4.5.2.2 TPU Interface

The capture register is read onto the TPU E bus and is automatically latched into the ERT register (execution unit) during the channel transition period whenever the CHAN register (execution unit) is modified. Access of an ER is qualified by the channel currently being serviced by the microengine (i.e., the channel value currently in CHAN). To access the ER of a channel other than the channel currently selected, the microcode must modify CHAN to the channel number of

the channel to be accessed. ERT is then automatically loaded with the capture register of the new channel. All microengine accesses of ER are limited to state T2 of the microcycle.

4.5.2.3 Match Event

A match event may only occur after a write to a channel match register. The match primitive is performed every resolution period by comparing the match register of a channel's ER against the value of the specified TCR. A match event is the condition in which the specified TCR increments to a value that is equal to or greater than that of a channel's match register, which asserts the match recognition latch (MRL). This condition also requires that the match enable latch (MEL) for the channel currently receiving service is enabled. **A** description of these latches is presented in **4.5.3 Latch Control.** Once a match event occurs, a new one cannot occur until after a subsequent write to the match register. The time base used for a match event is nonexclusive, i.e., either TCR1 or TCR2 can be the reference time base. A match is conditioned by resolution states RS2 and RS3 (see Figure 4-16).

If the sum of the match register and the two's complement of the match TCR result in bit 15 is one, then the current match TCR time is greater than or equal to the match register. When a value is written to a match register, within one resolution period, a greater-than-or-equal-to compare is made. If the time stored in the match register is less than the TCR, then no further action occurs until the TCR increments to the value contained in the match register. If the time stored in the match register is greater than or equal to the TCR-, then the channel immediately forces the pin state as specified by pin action control latches.

4.5.2.4 Capture Event

A capture event is the condition in which the value of a specified TCR is gated into the capture register. A capture may be initiated by two different occurrences: the assertion of MRL or a specified input transition at the pin (transition detect latch (TDL) asserted). A capture caused by a match occurs in resolution state RS4. A capture event caused by an input transition occurs in resolution state RS2 or RS4. Figure 4-16 depicts these two occurrences.

4.5.3 Latch Control

Latch control refers to those latches affecting the operation of the ER and/or the channel service requests (match and capture). These controls are the match recognition latch, match enable latch, match recognition latch enable, transition detect latch, and the match/transition service request inhibit latch.

4.5.3.1 Match Recognition Latch (MRL)

MRL indicates the occurrence of a greater-than-or-equal-to match event. Assertion of MRL initiates a service request to the scheduler unless the match/transition service-request-inhibit (SRI) latch is asserted. When MRL is asserted, the level specified by the pin action control latches is output to the associated channel pin during T2 of RS3, and a capture occurs.

Once MRL has been asserted and then negated, it cannot be asserted again until its associated match register has been written. Also, after reset, the match register must be written for MRL to be asserted.

Assertion of MRL is inhibited by assertion of TDL. This inhibit ensures that no subsequent match will initiate a capture until TDL is negated. Assertion of MRL is also inhibited by MEL.

4.5.3.2 Match Enable Latch (MEL)

MEL selectively enables assertion of MRL for the scheduled channel during service. When channel service occurs (as the result of either a link or a host request) while a match is pending, MEL selectively enables the assertion of the MRL. In many cases without MEL, significant microcode overhead would be required to prevent an output pin glitch as a result of MRL being asserted due to a previously scheduled match while trying to extend a pulse width. A bit in the entry point allows selective enabling of MRL for each state. (See **4.2.2 Entry Point Format.)**

4.5.3.3 Match Recognition Latch Enable (MRLE)

MRLE is negated when TDL is asserted. This is to ensure that data captured due to the specified transition is not overwritten, a condition which occurs when a match event is used as a timeout. Writing the match register to schedule the next match sets MRLE and enables matches.

4.5.3.4 Transition Detect Latch (TDL)

TDL indicates a specified transition occurrence on a channel whose associated pin is configured as an input. TDL initiates a capture event and a service request to the scheduler if SRI control is not asserted. Assertion of TDL can occur on any T2 of a microcycle.

This latch is negated during reset and may also be negated by the microengine. If the SRI bit is asserted, TDL will be negated and will not be asserted even if the specified transition occurs at the pin. However, the specified TCR is captured in the channel capture register.

4.5.3.5 Match/Transition Service-Request Inhibit (SRI) Latch

SRI blocks channel service requests due to the assertion of MRL and/or TDL. SRI does not affect recognition of link service requests or host service requests. It is asserted during reset and may be written by the microengine. Assertion of this control latch also negates MRL and TDL.

To unburden the microengine, SRI configures a channel "dumb" regarding the servicing of match and capture channel service requests. MRL can still be asserted, and the level specified by the pin action control latches will be output to the pin. For inputs, the TDL is transparent, and so the last specified transition always causes a capture event.

4.5.4 Pin Control and TPU Pins

Any TPU pin may be programmed to be either an output or an input via microcode. Pin control is provided by three fields of a channel parameter register: pin action control (PAC), pin state control (PSC), and time/base directionality control (TBS). These fields are detailed in **2.3.2 Channel Parameter Registers-Channel Configuration Control.**

4.5.4.1 Output Pin

The host CPU may affect the logic level of an output pin by implementing one of three actions:

1. Specify the logic level output to the pin when MRL asserts. PAC is programmed to output high, low, or toggle logic levels with assertion of the MRL.

2. Immediately force a logic level previously programmed into PAC.

3. Force the output logic level of the pin directly to high or low.

Data contained in a parameter may also be used to affect PAC, PSC, and TBS.

4.5.4.2 Input Pin

As an input pin, the microcode can directly control the effect of the transition edge. PAC can be programmed to cause TDL to be asserted when a rising and/or falling edge is detected.

The TCR2 pin and each channel configured as an input has an associated synchronizer followed by a digital filter connected to the pin that samples pin transitions. These filter out high and low pulse widths less than the period of two system clocks, preventing these transitions from being input to the transition detect logic. The synchronizer and digital filter are guaranteed to pass pulses that are greater than the period of four system clocks. A minimum delay of seven ticks (3.5 TPU system clocks) to a maximum delay of 11 ticks (5.5 TPU system clocks) is required from the time the specified transition occurs at a pin until assertion of TDL on that channel.

4.6 TPU Microcode Development Support

The TPU contains a number of hardware hooks that aid in the development of microcode. The following paragraphs describe the development support hooks supplied in the TPU. For a quick reference of the register bits and fields discussed on subsequent pages, refer to **Table 2-2 Bit/Field Quick Reference Guide.**

The TPU provides extensive support for a development mode of operation when the chip is in test mode (refer to standby RAM (with TPU emulation) descriptions in the MC68332 user's manual, document number MC68332UM/AD (formerly *MC68332 SIM User's Manual).* This support includes:

1) the capability to halt TPU microcode execution at the end of a microcycle as a result of one of several breakpoint sources, both internal and external to the TPU,
2) the capability for the bus master to examine and/or modify most internal resources available to the TPU microengine,
3) the capability to single step the TPU microengine,
4) the capability to change the μPC and microinstruction register, thus altering the flow of the microprogram, and
5) the capability to execute microcode from a source external to the TPU via a test scan path.

Microcode development support has two registers: the development support control register and the development support status register. The development support hooks are configured through the development support control register. This register controls the enabling of internal breakpoints, TPU response to the IMB FREEZE signal, μPC latching characteristics, TCR stopping and starting, etc., and indicates their status to the bus masters.

TPU resources available to the bus master while the TPU is in the halted state include the parameter RAM and all system registers normally available. In addition, the link, service grant latch, and decoded channel registers, as well as most registers in the microengine and execution unit, are available to the bus master.

TPU execution of microcode can be halted due to two types of breakpoint sources: internal and external. Internal breakpoints may be generated by comparing for the channel of service, the address of the μPC**,** or the service request sources on a channel. The external breakpoint sources are the IMB FREEZE signal and the HOT4 bit in the development support control register. The halted state is identical to the T4\ state described in **4.2.12 Microengine Timing.**

TPU microinstruction execution can be halted because of one or more breakpoints on the following: the μPC register, the CHAN register, and the four service request sources.

The TPU contains a nine-bit μPC breakpoint register and a 4-bit channel number breakpoint register. These registers halt the TPU whenever the μPC register and CHAN register, respectively, are equal to the contents of the breakpoint registers. The TPU can also be halted by any combination of the four service requests. Any or all of the internal breakpoint sources can be enabled via the development support control register.

Both breakpoint registers are loaded and read using the test mode serial scan paths in the TPU. By necessity, the TPU serial scan paths are only accessed in test mode by using logic in the test submodule. (Refer to test scan path descriptions in the MC68332 user's manual (formerly *MC68332 SIM User's Manual)* document number MC68332UM/AD). As a result, the breakpoint registers are not contained in the memory address map and are, therefore, not directly accessible to the CPU. The breakpoint registers are set to zero out of reset.

Whenever any type of breakpoint is taken within the TPU, the BKPT bit in the development support status register is asserted by the TPU. Assertion of the breakpoint bit asserts the IMB BRKPT signal. The TPU subsequently negates BRKPT when a breakpoint acknowledge bus cycle occurs or the FREEZE signal is asserted.

TPU microcode execution can be halted when the FREEZE signal or the HOT4 bit is asserted. If FREEZE is asserted, the user can halt TPU execution either at the end of the next microcycle, at the end of the current state, or ignore FREEZE and continue executing. Once the TPU is halted, it remains halted until FREEZE is negated. If the HOT4 bit is asserted, the TPU halts at the end of the current microcycle and remains halted until HOT4 is negated.

Once the TPU is halted as a result of an internal or external breakpoint, the user can single step the TPU on a microcycle basis. Single stepping causes the TPU to begin execution from the halted state, execute one microinstruction, and then halt again. This control enables the microcoder to trace the flow of a microprogram and examine the TPU's resources between the execution of every microinstruction.

Another use of the single-step feature is the execution of microinstructions from a test scan path. Using the test scan path into the microinstruction register, the user can load the microinstruction register while the TPU is in the halted state. After loading the microinstruction register, the user can then single step the TPU, which is useful for gaining access to the TPU resources in the execution unit.

After the TPU has halted due to an internal or external breakpoint, the user can alter the flow in a microprogram being executed by changing the μPC contents via a test scan path. Once the μPC has been changed, the user can single step the TPU or resume normal execution using the new contents of the μPC.

Once the TPU is halted, the bus master can access all TPU resources needed for microcode development in one of two ways: via a standard IMB bus cycle or via a test scan path. Accessing certain resources may require a combination of both methods. The CPU may access certain registers and functional units using the test serial scan paths. These registers and functional units are the μPC register, microinstruction register, the two breakpoint registers, the branch PLA, and the scheduler PLA.

Scanning into the microinstruction register allows the user to execute microcode from an external source. Using this feature, the CPU can gain access to the remaining resources in the TPU. Access includes all registers in the execution unit and the match and capture registers in any channel. All of these registers may be transferred to the parameter RAM.

4.6.1 Test Configuration Register

The TPU module has one test register to configure and control the module for test purposes. This register resides in supervisor data space. Access to this register is also qualified by the MCU being in test mode. When read from supervisor data space in nontest mode, the test register reads as $0000. In addition to using this register for testing the TPU, a test submodule, described in the MC68332 user's manual (document number MC68332UM/AD (formerly *MC68332 SIM User's Manual))* is extensively used in testing the TPU module.

TCR $YFFE02

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	INCAD	TCR1C	ACUTR1	ACUTR0	0	0	SOSEL2	SOSEL1	SOSEL0	SISEL2	SISEL1	SISEL0	TMM
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BITS 15-13, 8, 7 - Not Implemented

INCAD - Increment Address
When set, INCAD forces the μPC to increment once for each assertion of the ACUTL line. This feature is used to sequentially dump the microROM. This bit can be read or written in test mode only.

0 = Normal operation
1 = μPC increments

TCR1C - TCR1 Clock
TCR1C selects the clock source, internal or external, for TCR1 (see Figure 4-17). This bit can be read or written in test mode only.

0 = Normal operation (TCR1 clocked internally)
1 = TCR1 clocked externally via the TCR2 pin; TCR1 is now clocked in parallel with TCR2.

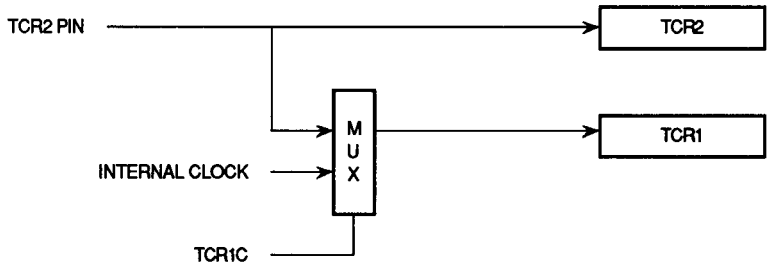


Figure 4-17. TCR1 Control Structure

ACUTR1, ACUTR0 - Activate Circuit Under Test Response 1, 0
The ACUTR bits determine the TPU module response to the test submodule asserting ACUT only. These bits can be read or written in test mode only.

ACUTR1	ACUTR0	TPU Response
0	0	None
0	1	Run one TPU microcycle. This setting is used to single step the TPU.
1	0	Assert scheduler end-of-time slot signal. This setting is used to test the HS PLA.
1	1	Reserved

SOSEL2-SOSEL0 - Scan-Out Select 2-0
These bits define the TPU output scan path to be connected to master shift register B (MSRB) via the SCANB line. MSRB is located in the test submodule and captures test responses from the TPU module. These bits can be read or written in test mode only.

SOSEL2	SOSEL1	SOSEL0	Output Scan Path
0	0	0	None
0	0	1	(μPC)
0	1	0	(Microinstruction)
0	1	1	(Branch PLA)
1	0	0	(μPC Breakpoint)
1	0	1	(Scheduler PLA)
1	1	0	(Channel Breakpoint)
1	1	1	Reserved

SISEL2-SISEL0 - Scan-in Select 2-0
The bits define the TPU input scan path to be connected to master shift register A (MSRA) via the SCANA line. Located in the test submodule, MSRA scans test stimulus from the test submodule to the TPU module. These bits can be read or written in test mode only.

SISEL2	SISEL1	SISEL0	Input Scan Path
0	0	0	None
0	0	1	(μPC)
0	1	0	(Microinstruction)
0	1	1	(Branch PLA)
1	0	0	(μPC Breakpoint)
1	0	1	(Scheduler PLA)
1	1	0	(Channel Breakpoint)
1	1	1	Reserved

TMW-Test Memory Map
TMM permits placing the TPU module at a fixed memory map location, which allows development of test patterns that are independent of the user memory map. This bit can be read or written in test mode only.

0 = Normal memory map (defined in the MG68332 users manual, formerly *MC68332 SIM User's Manual*)

1 = Module located at $YDF000-$YDFFFF. Figure 4-18 illustrates the test memory map of a 512-byte TPU. The TPU registers are mirrored in multiple locations in the range $YMDFxxx (Y = m111 where m reflects the state of the modmap bit in the module configuration register of the system integration module (Y = $7 or $F)).

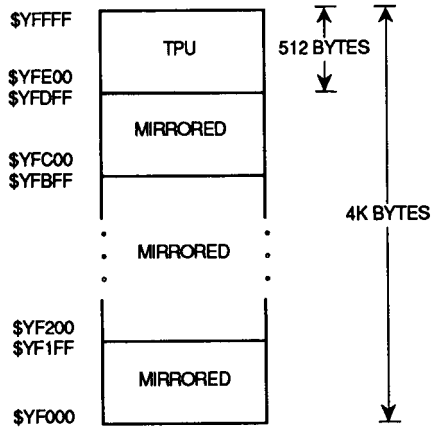


Figure 4-18. 512-Byte TPU Test Memory Map

4.6.2 Development Support Control Register

The development support control register controls the starting and stopping of the TPU microengine as a result of various internal and external events. Events causing the TPU to halt execution are described in more detail in **4.6 TPU MICROCODE DEVELOPMENT SUPPORT.**

This register resides in supervisor data space. Access to this register is also qualified by the MCU being in test mode (IMBTST asserted on IMB). When read from supervisor data space in nontest mode, this register reads as $0000 and returns a DTACK. During reset this register is initialized to $0000.

DEVELOPMENT SUPPORT CONTROL REGISTER $YFFE04

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HOT4	0	0	0	0	BLC	CLKS	FRZ0/FRZ1		CCL	BP	BC	BH	BL	BM	BT
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

HOT4 - Hang on T4
When asserted, HOT4 causes the TPU to enter a T4\ wait state when the subsequent T4 state occurs. Negating this bit releases the TPU from the wait on T4\ state.

0 = Exit wait on T4 state caused by assertion of HOT4

1 = Enter wait on T4 state

Bits 14-11 - Not Implemented

BLC - Branch Latch Control
This bit is intended for scanning in specific conditions in test mode, with the TPU halted. The effect of the BLC bit falls into three categories, according to the type of branch condition. Some conditions are latched only during time-slot transition, some conditions are latched both as a result of microcode and during time-slot transition, and some conditions are latched only as a result of microcode. Branch conditions latched during time-slot transition (first two cases) are also latched according to the following mechanisms:

1. Channel flags of the current channel are latched as specified by the FLC microinstruction field.

2. The pin state is latched whenever the channel register is changed via microcode (to reflect new channel's pin state).

3. If CCL = 1, MRL and TDL are latched whenever the channel register is changed via microcode.

The execution unit conditions (such as AU branch conditions) are latched only as specified by the by the CCL microinstruction field (latter or third case).

BLC specifies whether new branch condition information is to be latched (BLC = 0) or branch condition information scanned in during the halted state (BLC = 1) is to be retained, only for the first microcycle after exiting the halted state.

1 = Do not latch conditions into branch condition register prior to exiting the halted state or during the time-slot transition period.

0 = Latch conditions into branch condition register prior to exiting halted state.

When new branch information is scanned in during the halted state, BLC should be set to one, allowing the new scanned-in data to be used for the first microcycle after the halted state. If the TPU is then single-stepped through a sequence of microinstructions, the new scanned-in data remains indefinitely. If the TPU runs (not single stepping) after exiting the halted state, the new scanned-in data remains only for the first microcycle after the halted state. After the first microcycle, branch conditions are altered *only as described above.*

CLKS - Stop Clocks (to TCRs)
This bit controls whether or not the TCRs stop running when the TPU enters the halted state after asserting one of the following: a breakpoint, HOT4, or the IMB FREEZE signal. Unpredictable results can arise if this bit is changed during the wait state.

0 = Do not stop TCRs.
1 = Stop TCRs during the halted state.

FRZ1, FRZ-IMB FREEZE Response
The FRZ bits specify the TPU microengine response to the FREEZE signal. FRZ1 FRZ0

FRZ1	FRZ0	
0	0	Ignore FREEZE
0	1	Reserved
1	0	Freeze at end of current microcycle
1	1	Freeze at next time-slot boundary

CCL - Channel Conditions Latch
CCL controls the latching of channel conditions (MRL and TDL) when the CHAN register is written.

0 = Only the pin state condition of the new channel is latched as a result of the write CHAN register microinstruction.

1 = Pin state, MRL, and TDL conditions of the new channel are latched as a result of a write CHAN register microinstruction. Latching these conditions allows visibility of the channel state of other TPU channels without these channels requiring a time slot and service.

BP, BC, BH, BL, BM, and BT - Breakpoint Enable Bits
Bits 5-0 contain the various breakpoint enable bits for the TPU, specifying the conditions for a breakpoint. The breakpoint is enabled by setting the corresponding bit to one and is disabled by setting the bit to zero. These bits are defined as follows:

BP-Break if μPC equals μPC breakpoint register.

BC-Break if CHAN register equals channel breakpoint register at beginning of state or when CHAN is changed via microcode.

BH-Break if host service latch is asserted at beginning of state.

BL-Break if link service latch is asserted at beginning of state.

BM-Break if MRL is asserted at beginning of state.

BT-Break if TDL is asserted at beginning of state.

A breakpoint occurs if any or all of the enabled breakpoint sources are true. Breakpoints occur at the beginning of a state with these exceptions: μPC breakpoint can occur at any microcycle, and the channel breakpoint can occur at the beginning of the state or when CHAN is changed via microcode.

4.6.3 Development Support Status Register

The development support status register indicates to the CPU whether or not the TPU is in a halted state, and if so, indicates why. This register resides in supervisor data space. Access to this register is also qualified by the MCU being in test mode. When read from supervisor data space in nontest mode, this register reads as $0000 and returns a DTACK. The register is initialized to $0000 during reset.

DEVELOPMENT SUPPORT STATUS REGISTER $YFFE06

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	BKPT	PCBK	CHBK	SRBK	TPUF	0	0	0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits 15-8, 2-0 – Not Implemented

BKPT - Breakpoint Asserted Flag
If an internal breakpoint caused the TPU to enter the halted state, the TPU asserts the BKPT signal on the IMB and the BKPT flag. The TPU continues to assert BRKPT until it recognizes a breakpoint acknowledge cycle from a host, or until the FREEZE signal on the IMB is asserted. Upon recognition of the breakpoint acknowledge cycle or assertion of the FREEZE signal, the TPU ceases to assert BRKPT. The BKPT flag remains asserted until it is negated by a bus master. BKPT is negated by first reading the flag in the asserted state, followed by writing the complement state value. When negated, this flag causes the TPU to exit the halted state.

PCB - μPC Breakpoint Flag
PCBK is asserted if a breakpoint occurs due to a μPC register match with the μPC breakpoint register. PCBK is negated when the BKPT flag is negated.

CHBK - Channel Register Breakpoint Flag
CHBK is asserted if a breakpoint occurs due to a CHAN register match with the channel register breakpoint register. CHBK is negated when the BKPT flag is negated.

SRBK - Service Request Breakpoint Flag
SRBK is asserted if a breakpoint occurs due to any of the service request latches being asserted along with their corresponding enable flag in the development support control register. SRBK is negated when the BKPT flag is negated.

TPUF - TPU FREEZE Flag
TPUF is asserted whenever the TPU is in a halted state as a result of FREEZE being asserted. This flag is automatically negated when the TPU exits the hafted state due to FREEZE being negated.

4.6.4 Test Verification Registers

The test verification registers provide internal visibility into TPU operation. These registers consist of the link register, the service grant latch register, and the decoded channel number register. These registers are only accessible by the host CPU in test mode and while the TPU is halted.

4.6.4.1 Link Register

The link register used in generating a link service request on a channel contains a request bit for each channel.

LINK REGISTER $YFFE22

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register resides in supervisor data space. CPU access to this register is also qualified by the MCU being in test mode and halted (T4/T4\). When read from supervisor data space in nontest mode or when the TPU is not halted, this register reads as $0000 and returns a DTACK.

0 = Link bit not asserted
1 = Link bit asserted

4.6.4.2 Service Grant Latch Register

This register ensures that all channels requesting service on a particular priority level receive at least one service time before any channel that has been serviced and requests new service is serviced again. This register contains a service grant bit for each channel.

SERVICE GRANT LATCH REGISTER $YFFE24

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

When the corresponding service grant bit is asserted, service is granted to a channel. This register resides in supervisor data space. CPU access to this register is also qualified by the MCU being in test mode and halted (T4/T4\).

When read from supervisor data space in nontest mode or when the TPU is not in wait state, this register reads as $0000.

4.6.4.3 Decoded Channel Number Register

This read-only register indicates the channel currently receiving service and contains a bit for each corresponding channel.

DECODED CHANNEL NUMBER REGISTER $YFFE26

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

An asserted bit in this register indicates that the corresponding channel is being serviced by the TPU. Only one bit is asserted at any time. This register resides in supervisor data space. CPU access to this register is also qualified by the MCU being in test mode and halted (T4/T4\). When read from supervisor data space in nontest mode, when no channel is being serviced, or when the TPU is not halted, the register reads as $0000.

4.6.5 Test Scan Registers

4.6.5.1 Branch Condition Register (BCR)

The branch condition register reflects branch conditions that are latched from the channel being serviced by the microengine. (Refer to **4.2.4 Branch PLA** and **4.6.2 Development Support Control Register** for information on when each branch condition is latched.) This register can only be accessed via scan.

BCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V	N	C	Z	FLAG1	FLAG0	TDL	MRL	LSL	SEQ1	SEQ0	PIN	0	HSR1	HSR0	CH0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

V, N, C, Z - AU Flags
These are status flags from an AU operation indicating overflow, negative, carryout, and zero results.

Flag1 and Flag0 - Channel Flags
These flags indicate the state of channel flags.

TDL - Transition Detect Latch Flag
When asserted, TDL indicates a specified transition occurrence on a channel whose associated pin is configured as an input.

MRL - Match Recognition Latch Flag
When asserted, MRL indicates the occurrence of a greater-than-or-equal-to match event.

LSL - Link Service Latch
When asserted, LSL indicates the occurrence of a link.

SEQ1 and SEQ0 - Host Sequence Bits
These bits indicate the state of the host sequence bits.

Pin -
Indicates the state of the pin.

Bit 3 - Not Implemented

HSR1 and HSR0 - Host Service Request Bits
When asserted, these bits indicate the occurrence of a host service request.

Bit 0 - Branch Result
1 = Branch taken
0 = Branch not taken

In test mode, BCR can be scanned and branch results tested by single stepping the TPU. Conditions are scanned into BCR, a branch instruction is scanned into the microinstruction register (refer to BCC and BCF microinstruction encodings in Appendix B); the TPU is then single stepped, and updates CH0 (bit 0) with the result.

4.6.5.2 Hardware Scheduler Condition Register (HSCR)

HSCR contains status information on the hardware scheduler. This register can only be accessed via scan.

HSCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	NRH	NRM	NRL	CS2	CS1	CS0	NS2	NS1	NS0	SELH	SELM	SELL	0	0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits 15,14, 2, and 1 - Not Implemented

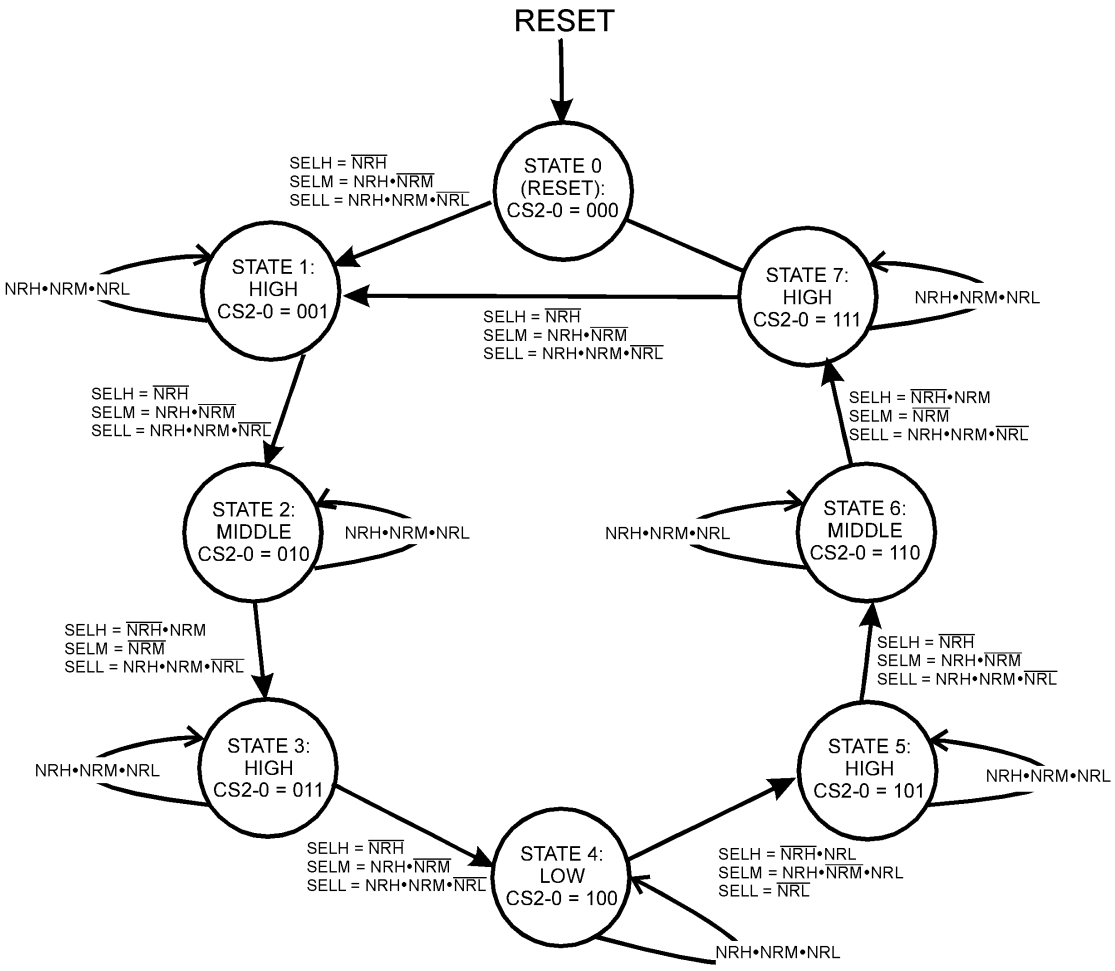
NRH, NRM, and NRL - No request on high, mid, or low priority

CS2, CS1, and CS0 - Current state of scheduler

NS2, NS1, and NS0 - Next state of scheduler

SELH, SELM, and SELL - Scheduler selects high, mid, or low for service

When in test mode, HSCR can be scanned and the scheduler tested by setting the test configuration register (TCR) and asserting ACUT line on the IMB. NRH, NRM, and NRL are scanned with request data; CS2, CS1, and CS0 are scanned with state data, and the schedule is exercised via ACUT line. The scheduler updates NS2, NS1, NS0, SELH, SELM, and SELL with next state and service data. (See Figure 4-19 Hardware Scheduler State Diagram.)



NOTE:
Inputs: NRH, NRM, NRL - No requests on high, medium, or low priority
CS2, CS1, CS0 - Current states of scheduler

Outputs: SELH, SELM, SELL - Selected priority high, medium, or low for service

Figure 4-19. Hardware Scheduler State Diagram

4.7 Using the TPU Development Support Features

The TPU development support features are explained by describing the registers and by detailing methods for setting up breakpoints, scanning into and out of the TPU registers, running the TPU in emulation mode, single stepping the TPU, and dumping the TPU control store. Addresses are interpreted as:

$$Y = m111 (\$7 \text{ or } \$F)$$

where m = state of modmap bit in system integration module.

The TPU registers used are as follows:

- TPU Module Configuration Register-$YFFE00

This register is discussed in **2.1.1 Module Configuration Registers**

- TPU Test Configuration Register-$YFFE02

SISEL/SOSEL values:
001 - μPC
010 - Microinstruction
011 - Branch PLA
100 - μPC Breakpoint
101 - Hardware Scheduler
110 - Channel Breakpoint

- TPU Development Support Control Register - $YFFE04
- TPU Development Support Status Register - $YFFE06

The test-submodule registers used are contained in the following list. These registers are also discussed in the MC68332 user's manual, document number MC68332UM/AD (formerly *MC68332 SIM User's Manual*).

- Test-Submodule Control Register-$YFFA38

- Test-Submodule Master Shift Register A-$YFFA30

  This 16-bit register is used for stimulus of the module under test (the TPU). It is written by the bus master with the value to be shifted into one of the TPU registers.

- Test-Submodule Master Shift Register B-$YFFA32

  This 16-bit register collects the response of the module under test (the TPU). It is read by the bus master and contains the value shifted out of one of the TPU registers.

- Test-Submodule Shift Count Register A/B-$YFFA34

  This 16-bit register controls the number of bits shifted in (A) or out (B) of the module under test (TPU). The upper byte is referred to as A; the lower byte is referred to as B.

- RAM Array Base Address and Status Register-$YFFB04

4.7.1 Setting Up Breakpoints

The TPU has three types of breakpoints: break on a service request match (transition detect latch, match recognition latch, link service latch, or host service latch asserted at the beginning of a phase), break on a channel register match (when the CHAN register is updated), and break on a μPC match. Each breakpoint type is described in the following paragraphs. The steps for setting up any of the service request breakpoints are as follows:

1. Enter test mode (these registers and bits can only be written in test mode) by asserting the ETM bit in the test-submodule control register.

2. Enter wait T4 states by asserting the HOT4 bit in the TPU development support control register.

3. Write to the TPU development support control register to enable the breakpoint. Breakpoints are enabled by setting the corresponding bit(s) to a one (1) and are disabled by clearing them. The bits have the following definitions:

   BT - Break if TDL is asserted at the beginning of a phase.

   BM - Break if MRL is asserted at the beginning of a phase.

   BL - Break if the link service latch is asserted at the beginning of a phase.

   BH - Break if the host service latch is asserted at the beginning of a phase.

4. Begin normal operation by requesting service using any of the normal methods and by exiting wait states (clear HOT4).

Now the TPU is configured for breakpoint on a certain type of service request. When the corresponding latch is asserted at the beginning of a phase, a breakpoint will be taken at the end of the time-slot transition period before the execution of the first microinstruction. At that time, the TPU will enter the halted state, and the TPU development support status register will reflect the breakpoint status. The BKPT and SRBK flags will be set. While in the halted state, other TPU registers may be scanned. Before exiting the hafted state, care must be taken to clear the conditions causing the breakpoint. The following steps should be used:

1 . Clear the latch that caused the breakpoint (TDL, MRL, link service latch, or host service latch). If this step is not done and breakpoints are still enabled upon exiting the halted state, another breakpoint will be taken.

2. If no more breakpoints are desired from this source, clear the corresponding bit in the TPU development support control register (BM, BT, BL, or BH.)

3. Read the TPU development support status register. The breakpoint flags may not be cleared until they have been read in the asserted state.

4. Clear the BKPT flag in the TPU development support status register. This step clears all other breakpoint flags that are set and causes the TPU to exit the halted state (unless HOT4 in the TPU development support control register is set). Normal execution of the TPU resumes at this point.

The steps for setting up breakpoints on a channel register match and a μPC match are similar:

1. Enter test mode (these registers and bits can only be written in test mode) by asserting ETM in the test-submodule control register.

2. Enter wait T4 states by asserting HOT4 in the TPU development support control register.

3. Write to the TPU development support control register to enable the breakpoint. Breakpoints are enabled by setting the corresponding bit(s) to one and are disabled by clearing them. These bits have the following definitions:

   BC - Break on a channel register match whenever the CHAN register is updated.

   BP - Break on a μPC match.

4. Scan the match value into the corresponding breakpoint register (μPC break-point register or channel breakpoint register) using the appropriate methods described.

5. Begin normal operation by requesting service using any of the normal methods and by exiting wait states (clear HOT4).

Now the TPU is configured for breakpoint on a match of one of these registers. When the match occurs, a breakpoint is taken. The timing of the breakpoint depends on whether or not other breakpoints are pending. If a μPC match is the only breakpoint pending, it occurs immediately (during the time phase execution.) If a channel match is the only breakpoint pending, it occurs at the end of the time-slot transition period before the execution of the first microinstruction (if CHAN register was updated during the time-slot transition period) or immediately (if CHAN register was updated via microcode). If either breakpoint is pending and a service request breakpoint is also pending, the breakpoint occurs at the end of the time-slot transition period before the execution of the first microinstruction. When the breakpoint is taken, the TPU enters the halted state, and the TPU development support status register reflects the breakpoint status. The BKPT and PCBK flags or the CHBK flag will be set. While in the halted state, other TPU registers may be scanned. Before exiting the halted state, care must be taken to clear the conditions causing the breakpoint. These steps outline the procedure to follow:

1. It may be necessary to update the latch that caused the breakpoint (μPC breakpoint register or channel number breakpoint register) if the match condition is still present. The latch also must be updated if another breakpoint is to be taken at another location or when changing to another channel. The latch is updated by scanning in a new value.

2. If no more breakpoints are desired from this source, clear the corresponding bit in the TPU development support control register (BC or BP).

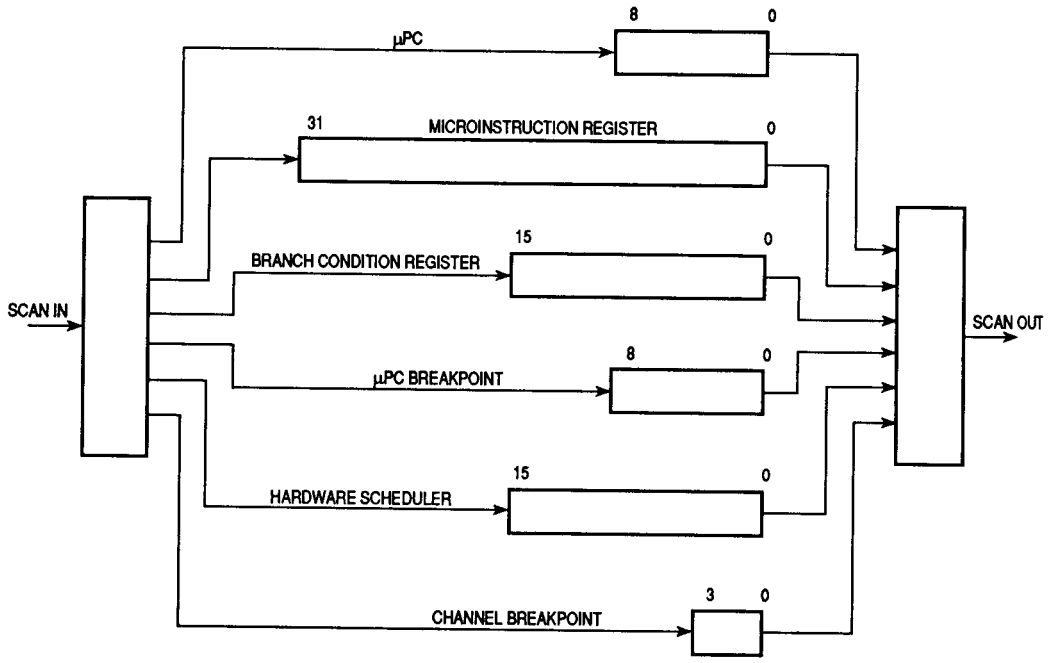
3. Read the TPU development support status register. The breakpoint flags may not be cleared until they have been read in the asserted state.

4. Clear the BKPT flag in the TPU development support status register. This step clears all other breakpoint flags that are set and causes the TPU to exit the halted state (unless HOT4 in the TPU development support control register is set). Normal execution of the TPU resumes at this point.

More than one breakpoint source can be enabled simultaneously. When multiple sources cause a breakpoint simultaneously, all corresponding flags in the TPU development support status register will be set. Care must be taken to clear the conditions causing the breakpoint unless another breakpoint from the same condition is desired.

4.7.2 Steps for Scanning Out the Value of a TPU Register

The following steps may be used to scan out the µPC, branch PLA, µPC breakpoint register, hardware scheduler PLA, and channel number breakpoint register. The action of scan-in and scan-out select is shown in the following illustration. (See **4.7.4 To Scan Out of the Microinstruction Register** for scanning out of the microinstruction register.)



1. Enter test mode (the following registers and bits can only be written in test mode) by asserting ETM in the test-submodule control register.

2. Reconfigure the IMB IRQ lines to test lines by asserting IMBTST in the test-submodule control register.

3. Enter wait T4 states by asserting HOT4 in the TPU development support control register.

4. Configure to run one microcycle in response to ACUTL line by Writing 01 to ACUTR in the TPU test configuration register.

5. Clear MUXSEL in the test-submodule control register.

6. Set scan-in select (SISEL = none) and scan-out select (SOSEL = register desired to scan out) by writing to the TPU test configuration, register.

7. Set shift-out count by writing the number of bits in the specified register to the test-submodule shift count register B.

8. Start shift by setting SSHOP in the test-submodule control register.

9. Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

10. Read the value in the test-submodule master shift register B and save. The value read may have to be right-adjusted if less than 16 bits were shifted.

The following steps restore the value into the selected TPU register.

11. Set scan-in select (SISEL) to the scanned-out register previously selected to restore its contents and scan-out select (SOSEL) to none by writing to the TPU test configuration register.

12. Set up shift-in count by writing the number of bits in the specified register to the test-submodule shift count register A .

13. Set scan-in value by writing the value that was read into the test submodule master shift register A.

14. Start shift by setting SSHOP in the test-submodule control register.

15. Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

16. Clear IMBTST in the test-submodule control register to reconfigure the IMB IRQ lines.

17. Exit wait T4 states by clearing HOT4 in the TPU development support control register.

4.7.3 To Scan into a TPU Register

Follow previous steps 1-5 and 11-17, writing the desired value into test submodule master shift register A at step 13.

4.7.4 To Scan Out of the Microinstruction Register

1. Enter test mode (these registers and bits can only be written in test mode) by asserting ETM in the test-submodule control register.

2. Reconfigure the IMB IRQ lines to test lines by asserting IMBTST in the test submodule control register.

3. Enter wait T4 states by asserting HOT4 in the TPU development support control register.

4. Configure to run one microcycle in response to ACUTL line by writing 01 to ACUTR in the TPU test configuration register.

5. Clear MUXSEL in the test-submodule control register.

6. Set scan-in select (SISEL = none) and scan-out select (SOSEL microinstruction) by writing to the TPU test configuration register.

7. Set shift-out count by writing $0010 to the test-submodule shift count register B (16 bits in lower word).

8. Start shift by setting SSHOP in the test-submodule control register.

9. Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

10. Read the lower word value in the test-submodule master shift register B and save.

The next steps restore the lower word value and read the upper word value:

11. Set scan-in select (SISEL = microinstruction) and scan-out select (SOSEL = none) by writing to the TPU test configuration register.

12. Set shift counts A and B by writing $1010 to the test-submodule shift count register (16 bits in each half of microinstruction).

13. Set scan-in value by writing the lower word value that was read into the test-submodule master shift register A.

14. Start shift by setting SSHOP in the test-submodule control register.

15. Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

16. Read the upper word value in the test-submodule master shift register B and save.

17. Combine the microinstruction upper and lower words.

The next steps restore the value of the upper word:

18. Set scan-in select (SISEL = microinstruction) and scan-out select (SOSEL = none) by writing to the TPU test configuration register.

19. Set shift-in count by writing $1000 to the test-submodule shift count register A (16 bits in upper word of microinstruction).

20. Set scan-in value by writing upper word value that was read into the test submodule master shift register A.

21. Start shift by setting SSHOP in the test-submodule control register.

22. Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

23. Clear IMBTST in the test-submodule control register to reconfigure the IMB IRQ lines.

24. Exit wait T4 states by clearing HOT4 in the TPU development support control register.

4.7.5 To Scan into the Microinstruction Register

1. Enter test mode (these registers and bits can only be written in test mode) by asserting ETM in the test-submodule control register.

2. Reconfigure the IMB IRQ lines to test lines by asserting IMBTST in the test submodule control register.

3. Enter wait T4 states by asserting HOT4 in the TPU development support control register.

4. Configure to run one microcycle in response to the ACUTL line by writing 01 to ACUTR in the TPU test configuration register.

5. Clear MUXSEL in the test-submodule control register.

6. Set scan-in select (SISEL = microinstruction) and scan-out select (SOSEL = none) by writing to the TPU test configuration register.

7. Set shift-in count by writing $1000 to the test-submodule shift count register A (16 bits in lower word of microinstruction).

8. Set scan-in value by writing the lower word value into the test-submodule master shift register A.

9. Start shift by setting SSHOP in the test-submodule control register.

10. Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

11. Set scan-in select (SISEL = microinstruction) and scan-out select (SOSEL = none) by writing to the TPU microinstruction register.

12. Set shift-in count by writing $1000 to the test-submodule shift count register A (16 bits in upper word of the microinstruction).

13. Set scan-in value by writing the upper word value into the test-submodule master shift register A.

14. Start shift by setting SSHOP in the test-submodule control register.

15. Reset IMBTST in the test-submodule control register to reconfigure the IMB IRQ lines.

16. Exit wait T4 states by clearing HOT4 in the TPU development support control register.

4.7.6 Running the TPU in Emulation Mode

1. Write the desired RAM base address, which specifies address bits A23 – A11 of the RAM array, to RAMBAR. The RAM array is enabled when bit 0 of RAMBAR is zero. RAMBAR can only be written one time out of reset.

2. Write microcode (including entry points) into the RAM module. Begin at the base address and continue as in the normal memory map for the TPU control store. The entire block does not need to be filled as long as the entry points for all functions to be run are in the correct locations.

3. Set the emulation mode bit (EMU) in the TPU module configuration register. This bit can only be written one time out of reset. Once set, the RAM cannot be accessed via the IMB.

4. Continue as if running any of the built-in functions:

    - Write to the channel priority registers to enable the desired channels.

    - Write to the function select registers to choose functions to run on each of the desired channels (only the microcode in RAM may be executed).

    - Write to the parameter RAM to initialize values needed by the functions.

    - Write to any other TPU registers necessary for the desired application (e.g., enable/disable interrupts).

    - Write to the channel priority register allowing operation to begin.

4.7.7 Dumping the Contents of the Control Store

1. Enter test mode (these registers and bits can only be written in test mode) by asserting ETM in the test-submodule control register.

2. Reconfigure the IMB IRQ lines to test lines by asserting IMBTST in test submodule control register.

3. Assert INCAD in the TPU test configuration register to increment μPC once for each microcycle, no jumps.

4. Enter wait T4 states by asserting HOT4 in the TPU development support control register.

5. Configure to run one microcycle in response to the ACUTL line by writing 01 to ACUTR in the TPU test configuration register.

6. Clear MUXSEL in the test-submodule control register.

7. Clear test registers in the SIM module that are not used (test-submodule master shift register B, reps counter, distributed register (DREG)).

8. Set scan-in select (SISEL = μPC) and scan-out select (SOSEL = none) by writing to the TPU test configuration register.

9. Set shift-in count by writing $0900 to the test-submodule shift count register A (nine bits in μPC).

10. Set scan-in value by setting the test-submodule master shift register A value to the start address of the dump (000).

11. Start shift by setting SSHOP in the test-submodule control register.

12. Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

13. Repeat for each instruction to be dumped:

    a) Set SISEL (0) and SOSEL (microinstruction) by writing to the TPU test configuration register.

    b) Run one microcycle by setting ACUT in the test-submodule control register.

    c) Set shift-out count by writing $0010 to the test-submodule shift count register B (16 bits in microinstruction low word.)

    d) Shift out low word of microinstruction; start shift by setting SSHOP in the test-submodule control register.

    e) Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

    f) Read the lower word value in the test-submodule master shift register B and save.

The next six steps are to scan out the second part of microinstruction and restore the first part:

    g) Set SISEL (microinstruction) and SOSEL (microinstruction) by writing to the TPU test configuration register.

    h) Set shift counts by writing $1010 to the test-submodule shift count register (16 bits each word).

    i) Set scan-in value by setting test-submodule master shift register A value to the lower word read.

    j) Start shift by setting SSHOP in the test-submodule control register.

    k) Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

    l) Read the upper word value in the test-submodule master shift count register B and save.

    m) Combine the upper and lower words to form microinstruction.

The next five steps restore the second word of microinstruction:

n) Set SISEL (microinstruction) and SOSEL (0) by writing to the TPU test configuration register.

o) Set shift-in count by writing $1000 to the test-submodule shift count register A (1 6 bits in upper microinstruction).

p) Set scan-in value by setting the test-submodule master shift register A value to upper word read.

q) Start shift by setting SSHOP in the test-submodule control register.

r) Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

14. Clear IMBTST in the test-submodule control register to reconfigure the IMB IRQ lines.

15. Clear INCAD in the TPU test configuration register.

4.7.8 Single Stepping the TPU

1. Enter test mode (these registers and bits can only be written in test mode) by asserting ETM in the test-submodule control register.

2. Reconfigure the IMB IRQ lines to test lines by asserting IMBTST in the test submodule control register.

3. Enter wait T4 states by asserting HOT4 in the TPU development support control register.

4. Configure to run one microcycle in response to ACUTL line by writing 01 to ACUTR in the TPU test configuration register.

5. Clear MUXSEL in the test-submodule control register.

6. Clear test registers in the SIM module that are not used (test-submodule master shift register B, reps counter, distributed register (DREG)).

7. Set scan-in select (SISEL = μPC) and scan-out select (SOSEL = none) by writing to the TPU test configuration register.

8. Set shift-in count by writing $0900 to test-submodule shift count register A (nine bits in μPC).

9. Set scan-in value by setting the test-submodule master shift register A value to the start address of the code to be executed.

10. Start shift by setting SSHOP in the test-submodule control register.

11. Wait for the test-submodule control register BUSY bit to be cleared, which indicates that shifting is finished.

12. Repeat for each instruction to be executed:

    a) Run one microcycle by setting ACUT in the test-submodule control register.

    b) Do whatever work or checking is desired in between microcycles (i.e., reading, writing, or scanning the TPU registers).

13. Clear IMBTST in the test-submodule control register to reconfigure the IMB IRQ lines.

14. Exit wait T4 states by clearing HOT4 in the TPU development support control register.

NOTE

Single stepping can be used with code running out of the control store or in emulation mode.

4.8 Example: A Coherency Solution

Coherent transfer of data in TPU operation is the access of data identical in age or logically related. More specifically, when written coherently, all data must be written before any is used. When read coherently, all data must be correctly related to one another. That is, all data must be updated or none must be updated when read. The condition in which a block of data is read, while some but not all has been updated, is not allowable.

4.8.1 Coherency Requirements for the TPU

Future time functions developed either by users or by Motorola may require coherent transfer of three or four parameters. This section provides a solution to such a need. For the predefined functions, a maximum of two parameters is required to be transferred coherently for proper operation. For example, the PWM function requires two coherent parameters: the period and high time. Coherent access of two adjacent parameter registers, with respect to logical address, is provided by the RAM arbitration scheme. The bus master accesses two adjacent parameters coherently by executing a long-word read or write. The TPU can also access two parameters coherently via microcode.

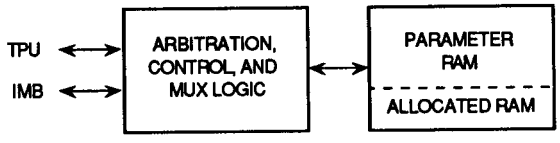


Figure 4-20. TPU RAM

4.8.2 Solution

Various solutions accomplish the coherency requirements. The solution chosen requires no specialized hardware. Coherency is ensured by the execution of a host control state. A predefined host control state for coherent reads and another for coherent writes can be written to support coherent transfers of any number of parameters. Refer to Figure 4-20, a block diagram of the TPU RAM, for aid in understanding the coherency solution. Figure 1-1 TPU Simplified Block Diagram demonstrates the integration of the RAM into the TPU. An overview and detailed discussion follow.

The TPU parameter RAM is dual access. The two users of the RAM are the TPU and a bus master of the IMB, normally the host CPU. If coherency of more than two parameters is required for the execution time functions, a portion of RAM is allocated for coherency, i.e., allocated RAM cannot be used for time function operands.

RAM allocated for coherency contains two types of registers when used for coherency: the coherent data registers and the coherent data control register.

Coherent data registers are used to interlock coherent parameters. Upon the execution of a host control state for coherent transfer, coherent data registers (some or all) are transferred to or from other parameter registers; this transfer is the essence of the solution. Coherency is ensured by the interlocking of parameters in the coherent data registers and by their transfer to and from other parameter registers by the TPU. The number of registers to be transferred depends upon the number of coherent parameters required for a particular coherent operation. The flow for coherent transfers is demonstrated in Figure 4-21.

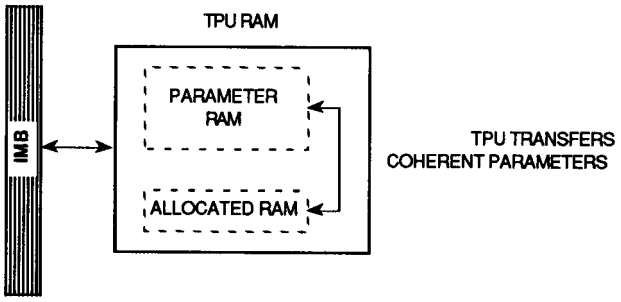


Figure 4-21. Coherent Data Flow

The transfer of the coherent data registers to and from the RAM is controlled via a coherent data control register, one of the parameter registers allocated for coherency. This register contains status and control bits, including a semaphore bit, which indicates the status of a shared resource in a system. Before a bus master can use the resource, it must first check the semaphore bit. If this bit is asserted, the resource is being used. If this bit is negated, the bus master may assert the bit and begin using the resource.

4.8.3 Parameter Registers Allocated for Coherency

As mentioned above, parameter registers can be allocated for coherency, if coherency is required. Among the registers are one coherent data control register and multiple coherent data registers. The number of parameters allocated as coherent data registers depends upon the maximum

number of coherent parameters required. For example, if, at most, three coherent parameters are required, three coherent data registers must be allocated.

When utilized for coherency, the parameter registers are part of the programmers model of the TPU and constitute a work space or cache to the bus master coherently accessing the TPU RAM.

4.8.3.1 Coherent Data Register

A coherent data register is a parameter register that may be allocated to interlock a coherent parameter. Coherent data registers are transferred to other parameter registers upon the execution of a coherent write state, and parameter registers are transferred to coherent data registers upon the execution of a coherent read state. Any parameter register may be allocated as a coherent data register.

4.8.3.2 Coherent Data Control Register

This register is a parameter register used for controlling the coherent transfers. It contains control bits that affect the host control states that perform the transfers: a semaphore bit and a parameter address bit field.

The semaphore bit is used to indicate the status of the portion of RAM allocated for coherency. If this bit is asserted, the allocated RAM is currently being used. If this bit is negated, the allocated RAM is not being used.

For the semaphore bit to provide status of the allocated RAM's usage, the following protocol for bus masters must be used. Before a bus master can use the allocated RAM, it must first check the semaphore bit. If the bit is asserted, the master must wait for it to be negated. If the bit is negated, the master may assert the bit and begin using the allocated RAM. The master must negate the bit after it is finished to relinquish the allocated RAM. A bus master may access unallocated parameter registers noncoherently regardless of the state of the semaphore bit. The bus master, which initiated the coherent transfer, must negate the semaphore bit upon completion of the coherent transfer.

Also included in the coherent data control register is a parameter address bit field. When a coherent transfer occurs, the parameter address bit field acts as a pointer for the coherent parameter registers.

4.8.3.3 Write Collision Protection

A coherent write state can also be used to provide write collision protection. Write collision is an overwrite by the TPU of a parameter written by a bus master (host CPU). A write collision can occur for the following condition: a state of a time function is executed in which a particular parameter is updated and a bus master is simultaneously updating the same parameter. The execution of a coherent write state eliminates the possibility of a write collision since the TPU microengine executes the write state. Parameters are generally updated by the TPU or a bus master, but not both.

4.8.4 Four-Parameter Coherency Microcode Example

```
PRIMITIVE : Coherent Read/Write
ACTION :    Coherent Read/Write
NOTES :     CDC refers to coherent data control register and CDRn
            refers to coherent data register.
```

```
PARAMETERS

%macro  CHCTL prm0.
%macro  CDC   #'FE'x.
%macro  CDR0  #'E8'x.
%macro  CDR1  #'EA'x.
%macro  CDR2  #'EC'x.
%macro  CDR3  #'EE'x.
```

```
PHASE :     Coherent read
PRELOAD PARAMETER :
ENTER WHEN :      issued host service request 01
ACTION :     execute a coherent read
```

```
%entry prim = 2, ram p <- prm0, target *, cond hsr1 = 1 hsr0 = 0.

%org #0.

Coherent-read
000 * ram p <-    @CDC.
001 * au acc := phi.
002 * au sr := #77.

Mode:
003   AE07FEFF    if SEQ1 = 0 then goto Start-trans.
004 * au dec := #08.
005 * au dec := #0D.


Start-trans:
006   Call Coh-trans, flsh.
```

```
End-of-Phase : for undefined entries
```

```
007 3FFFFFFE      !End_of_phase : end.
```

```
PHASE : Coherent write
PRELOAD PARAMETER :
ENTER WHEN : issued host service request 10
ACTION : execute a coherent write
```

```
%entry prim = 2, ram p <- prm0, target cond hsr1 = 1 hsr0 = 1.

Coherent write :
008 *     ram p <- @CDC.

009 *     au sr := phi.

00A *     goto Mode.

00B *     au acc := #77.


PROCEDURE : Coh_trans
CALLED BY : Coherent_read, Coherent_write
ACTION : coherent transfer of 3/4 parameters
PARAMETERS & REGISTERS :
If called by Coherent_read:
      sr - right-justified address for CDR3
      acc - right-justified address for LAST PARAMETER of coherent data
            block (P2 or P3)
If called   by Coherent_write:
      sr - right-justified address for LAST PARAMETER of coherent data
           block (P2 or P3)
      acc - right-justified address for CDR3



Coh-trans:
00C * au diob :=<< acc.
00D * ram p <- (diob); au diob :=<< sr.
00E * ram p -> (diob); au acc := acc
00F * goto Coh_trans; suben.
010 * au sr := sr - 1.


UNDEFINED ENTRIES - execute an end.

%entry prim = 2, ram p<-prm0, target End-of_phase,
cond hsr1 = 0 hsr0 = 1

%entry prim = 2, ram p<-prm0, target End - of_phase,
cond hsr1 = 0 hsr0 = 0 m/tsr = 1 lsl = 0.

%entry prim = 2, ram p<-prm0, target End-of_phase,
cond hsr1 = 0 hsr0 = 0 lsl = 1.


ENTRY TABLE

110  10001000
```

```
111  1001100E
112  10001000
113  10001000
114  10001000
115  10001000
116  10001000
117  10001000
```

ROM MAP

```
  0123456789ABCDEF
  ----------------
0 XXXXXXXXXXXXXXX
1 XXXXXXXXXXX•••••
2 ••••••••••••••••
3 ••••••••••••••••
4 ••••••••••••••••
5 ••••••••••••••••
6 ••••••••••••••••
7 ••••••••••••••••
8 ••••••••••••••••
9 ••••••••••••••••
A ••••••••••••••••
B ••••••••••••••••
C ••••••••••••••••
D ••••••••••••••••
E ••••••••••••••••
F ••••••••••••••••
```

ENTRY TABLE MAP

```
100 •• •• •• •• •• •• •• ••
108 •• •• •• •• •• •• •• ••
110 XX XX XX XX XX XX XX XX
118 •• •• •• •• •• •• •• ••
120 •• •• •• •• •• •• •• ••
128 •• •• •• •• •• •• •• ••
130 •• •• •• •• •• •• •• ••
138 •• •• •• •• •• •• •• ••
140 •• •• •• •• •• •• •• ••
148 •• •• •• •• •• •• •• ••
150 •• •• •• •• •• •• •• ••
158 •• •• •• •• •• •• •• ••
160 •• •• •• •• •• •• •• ••
168 •• •• •• •• •• •• •• ••
170 •• •• •• •• •• •• •• ••
178 •• •• •• •• •• •• •• ••
```

LABELS

Label	Address	(decimal)
coherent_read	01	1
coherent_write	0E	14
end_of_phase	00	0
read_3_param	07	7

```
write_3_param     14                20
```

Symbol Table

Symbol	Value
chctl	prm0
cdc	#’fe’x
cdr0	#’e8’x
cdr1	#’ea’x
cdr2	#’ec’x
cdr3	#’ee’x