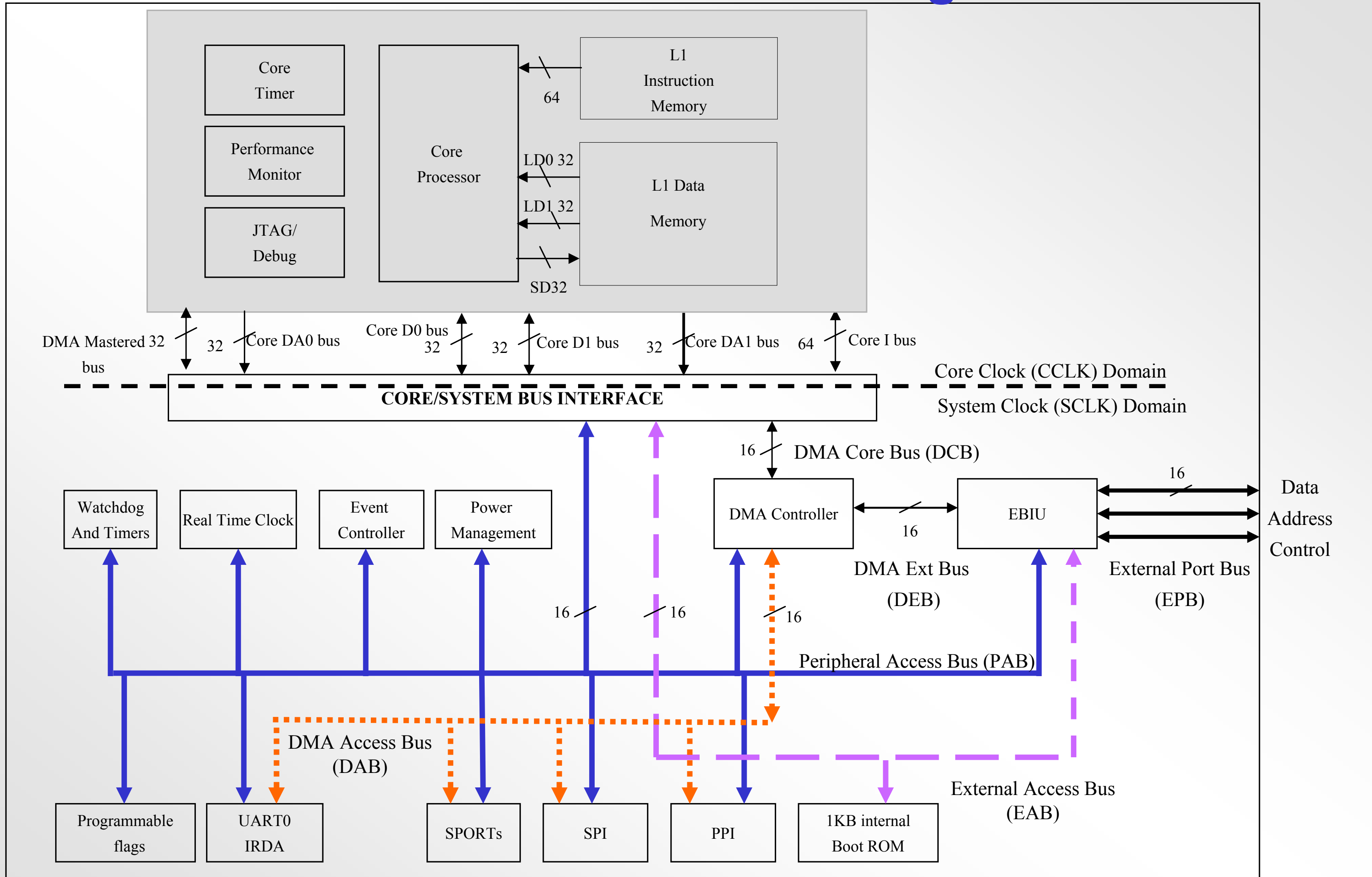
Section 7

Program Sequencer

KAZTEK ENGINEERING

ANALOG DEVICES

ADSP-BF533 Block Diagram



KAZTEK ENGINEERING

ANALOG DEVICES

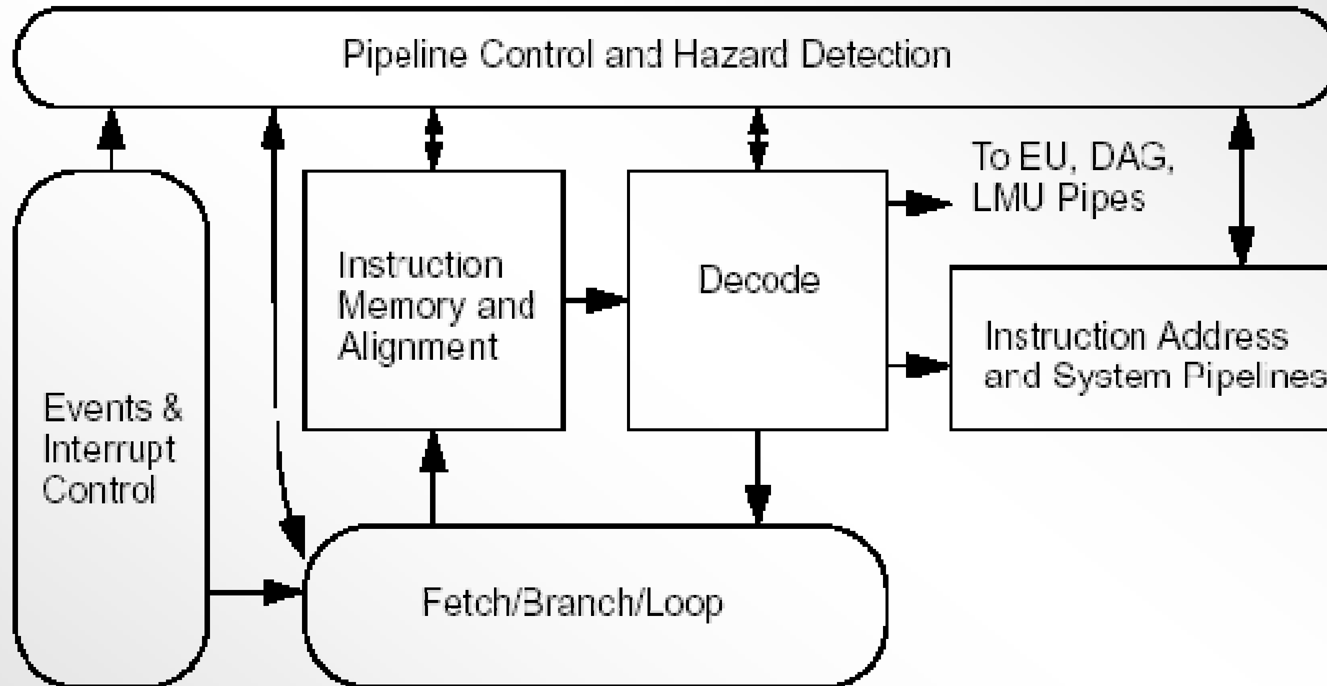
Program Sequencer Features

- **The Program Sequencer controls all program flow:**
  - **Maintains Loops, Subroutines, Jumps, Idle, Interrupts and Exceptions**
  - **Contains an 10-stage instruction pipeline**
  - **Includes Zero-Overhead Loop Registers**

KAZTEK ENGINEERING

ANALOG DEVICES

Program Sequencer



KAZTEK ENGINEERING

ANALOG DEVICES

Sequencer-Related Registers

Register Name	Description
SEQSTAT	Sequencer Status register
 RETX RETN RETI RETE RETS	Return Address registers: See "Events and Sequencing" on page 4-18. Exception Return NMI Return Interrupt Return Emulation Return Subroutine Return
 LC0, LC1 LT0, LT1 LB0, LB1	Zero-Overhead Loop registers: Loop Counters Loop Tops Loop Bottoms
FP, SP	Frame Pointer and Stack Pointer: See "Frame and Stack Pointers" on page 5-5
SYSCFG	System Configuration register
CYCLES, CYCLES2	Cycle Counters: See "CYCLES and CYCLES2 Registers" on page 19-25
PC	Program Counter

KAZTEK ENGINEERING

ANALOG DEVICES

Program Flow Instructions

Program Flow Instruction	*Instruction Function*
JUMP	Unconditional Branch
IF CC JUMP IF !CC JUMP	Conditional Branch
CALL	Subroutine call
RTS,RTI,RTX,RTN,RTE	Return from Flow interrupter
LSETUP	Set up Hardware Loop

- Jump (P5); /* indirect jump instruction */
- Jump (PC + P3); /* indirect jump with offset (PC-relative) */
- Call (P5); /* RETS register is loaded with address of instruction after call */
- Call (PC + P3); /* RETS register is loaded with address of instruction after call */
- IF CC Jump <label>; /* jump on condition cc=1 */
- Call <label>; /* OK within 24-bit offset from PC */

KAZTEK ENGINEERING

ANALOG DEVICES

Conditional Execution – CC Bit

- Condition Code Flag (CC bit) resolves
  - Conditional branch
    - e.g., IF !CC JUMP TO_END;
  - Conditional move
    - e.g., IF CC r0 = r1;
- Some ways to access CC to control program flow
  - Dreg value can be copied to CC, and vice-versa
  - Status flag can be copied into CC, and vice-versa
    - e.g., CC = AV1;
  - CC can be set to result of a Preg comparison
  - CC can be set to result of a Dreg comparison
    - e.g., CC = R3==R2;
  - BITTST instruction
- Refer to Chapter 4 in Workshop for more info on CC bit

ADSP-BF533 Execution Pipeline

- 10-stage super-pipeline
- The sequencer ensures that the pipeline is fully interlocked and that all the data hazards are hidden from the programmer
- If executing an instruction that requires data to be fetched, the pipeline will stall until that data is available

Instruction Pipeline

Pipeline Stage	Description
Instruction Fetch 1 (IF1)	Start instruction memory access.
Instruction Fetch 2 (IF2)	Intermediate memory pipeline.
Instruction Fetch 3 (IF3)	Finish L1 instruction memory access.
Instruction Decode (DEC)	Align instruction, start instruction decode, and access Pointer register file.
Address Calculation (AC)	Calculate data addresses and branch target address.
Execute 1 (EX1)	Start access of data memory.
Execute 2 (EX2)	Register file read.
Execute 3 (EX3)	Finish accesses of data memory and start execution of dual cycle instructions.
Execute 4 (EX4)	Execute single cycle instructions.
Write Back (WB)	Write states to Data and Pointer register files and process events.

ADSP-BF533 Execution Pipeline

	Inst Fetch1	Inst Fetch2	Inst Fetch3	Inst. Decode	Address Calc	Ex1	Ex2	Ex3	Ex4	WB
Inst Fetch1	Inst Fetch2	Inst Fetch3	Inst. Decode	Address Calc	Ex1	Ex2	Ex3	Ex4	WB	

TIME →

Pipeline Stage										
	IF1	**IF2**	**IF3**	**DC**	**AC**	**EX1**	**EX2**	**EX3**	**EX4**	**WB**
1	Insta	Inst9	Inst8	Inst7	Inst6	Inst5	Inst4	Inst3	Inst2	Inst1
2		Insta	Inst9	Inst8	Inst7	Inst6	Inst5	Inst4	Inst3	Inst2
3			Insta	Inst9	Inst8	Inst7	Inst6	Inst5	Inst4	Inst3
4				Insta	Inst9	Inst8	Inst7	Inst6	Inst5	Inst4
5					Insta	Inst9	Inst8	Inst7	Inst6	Inst5
6						Insta	Inst9	Inst8	Inst7	Inst6
7							Insta	Inst9	Inst8	Inst7
8								Insta	Inst9	Inst8
9									Insta	Inst9
10										Insta

Pipeline Events

- **Stall**
  - **A latency stall condition can occur when two instructions require extra cycles to complete, because they are close to each other in the assembly program. Other stalls can be memory or loop related. Stalls can be diagnosed with the Pipeline Viewer, and can be remedied with some rescheduling.**
- **Kill**
  - **Instructions after a branch are invalidated in the pipeline, because they will have entered the pipeline before the actual branch instruction gets serviced**
- **Multicycle Instruction**
  - **These instructions take more than one cycle to complete. These extra cycles cannot be avoided without removing the instruction that caused them.**
- **See EE-197 Appnote for a complete list of stalls and multicycle instructions.**

SSYNC and CSYNC instructions

- **SSYNC instruction synchronizes "the System", executing everything in the processor pipeline, and completing all pending reads and writes from peripherals.**
  - **Until SSYNC completes, no further instructions can enter the pipeline.**
- **CSYNC instruction synchronizes "the Core", executing everything in the processor pipeline**
  - **CSYNC is typically used after Core MMR writes to prevent imprecise behavior.**

Some Examples of Stall Conditions

- **Use of a Preg loaded in the previous instruction causes a 3-cycle stall**
  - `P0=[P1++];`
  - `R0=[P0];`
- **Use of a Preg which was transferred from Dreg in the previous instruction causes a 4-cycle stall.**
  - `P0=R0;`
  - `P1=P0+P2;`
- **Back-to-back multiplication where the result of first multiplication is used as an operand of the second multiplication causes 1-cycle stall**
  - `R0 = A1+=R1.L*R2.L;`
  - `R1 = A1+=R0.L*R2.L;`
- **Dual data fetch from the same Bank (A,B), 16KB half-bank (A16 matches), sub-bank (A13 and A12 match), and 32-bit polarity (A2 matches) takes 2 cycles**

  (e.g. I0 is address 0xFF80 1344, I1 is address 0xFF80 1994)

  R1 = R4.L * R5.H (IS) || R2 = [I0++] || [I1++] = R3;

KAZTEK ENGINEERING

ANALOG DEVICES

Avoiding Pipeline Stalls

- Most common numeric operations have no instruction latency
- Application note EE-197 available on avoiding stalls
  - Gives instruction combinations with associated stall info
- VDSP++ 3.5 Pipeline Viewer highlights Stall, Kill conditions

Pipeline Viewer

Cycle	Decode	Address	Execute0	Execute1	Execute2	Execute3	Writeback
2087	R1 =...	P0 =...	I0.H...	I0.L...	B0.H...	B0.L...	L0 =...
2088	S R1 =...	B	P0 =...	I0.H...	I0.L...	B0.H...	B0.L...
2089	S R1 =...	B	B	P0 =...	I0.H...	I0.L...	B0.H...
2090	S R1 =...	B	B	B	P0 =...	I0.H...	I0.L...
2091	S R1 =...	B	B	B	B	P0 =...	I0.H...
2092	JUMP...	R1 =...	B	B	B	B	P0 =...
2093	K	JUMP...	R1 =...	B	B	B	B
2094	K	K	JUMP...	R1 =...	B	B	B
2095	K	K	K	JUMP...	R1 =...	B	B
2096	K	K	K	K	JUMP...	R1 =...	B
2097				K	K	JUMP...	R1 =...
2098				K	K	K	JUMP...
2099				K	K	K	K
2100				I0 +...	K	K	K
2101				R1 =...	I0 +...	K	K
2102				IDLE ;	R1 =...	I0 +...	K

Details for stage Decode (cycle 2096)
Address: Invalid
Instruction: Invalid
Event 0:
Type: Kill
Cause: Mispredict, Interrupt, Refetch

KAZTEK ENGINEERING

ANALOG DEVICES

Change of Instruction Flow

- **When a change of flow happens, a new address is presented to the Instruction Memory Unit**
  - **There will be a minimum of four cycles before the new instructions appear in the decoder (except when utilizing the hardware loop buffers)**

- **When an instruction in a given pipeline stage is killed, all the instructions in stages above it will also be killed**

Unconditional Branches (JUMPS) in the Pipeline

- The Branch target address calculation takes place in the AC stage of the pipeline
- For all the unconditional branches, the Branch Target address is sent to the Fetch address bus at the beginning of the next cycle (EX1 stage of the branch instruction).
- The latency for all unconditional branches is 4 cycles

	1	2	3	4	5	6	7	8	9	10	11	12	13
IF1	I1	**Br**	I2	I3	I4	I5	**BT**						
IF2		I1	**Br**	I2	I3	I4	I5	**BT**					
IF3			I1	**Br**	I2	I3	I4	I5	**BT**				
DC				I1	**Br**	NOP	NOP	NOP	NOP	**BT**			
AC					I1	**Br**	NOP	NOP	NOP	NOP	**BT**		
EX1						I1	**Br**	NOP	NOP	NOP	NOP	**BT**	
EX2							I1	**Br**	NOP	NOP	NOP	NOP	**BT**
EX3								I1	**Br**	NOP	NOP	NOP	NOP
EX4									I1	**Br**	NOP	NOP	NOP
WB										I1	**Br**	NOP	NOP

I1: Instruction Before the Branch

I2: 1st Instruction After the Branch

I3: 2nd Instruction After the Branch

I4: 3rd Instruction After the Branch

I5: 4th Instruction After the Branch

Br: Branch Instruction

BT: Instruction at the Branch Target

KAZTEK ENGINEERING

ANALOG DEVICES

Conditional Branches (Jumps) in the Pipeline

- **Conditional Branches (Jumps) are executed based on the CC bit.**
- **A static prediction scheme (based on BP qualifier in instruction) is used to accelerate conditional branches**
  - Example: IF CC JUMP *user_label* (bp) ;
- **The branch is handled in the AC stage. In the EX4 stage, the sequencer compares the true CC bit to the predicted value.**
  - If mis-predicted, the branch is corrected and the correction address is put out in the WB stage of the branch instructions

Prediction	**Taken**		**Not taken**	
Outcome	Taken	Not taken	Taken	Not taken
Total Cycles to Execute	5 cycles	9 cycles	9 cycles	1 cycle

KAZTEK ENGINEERING

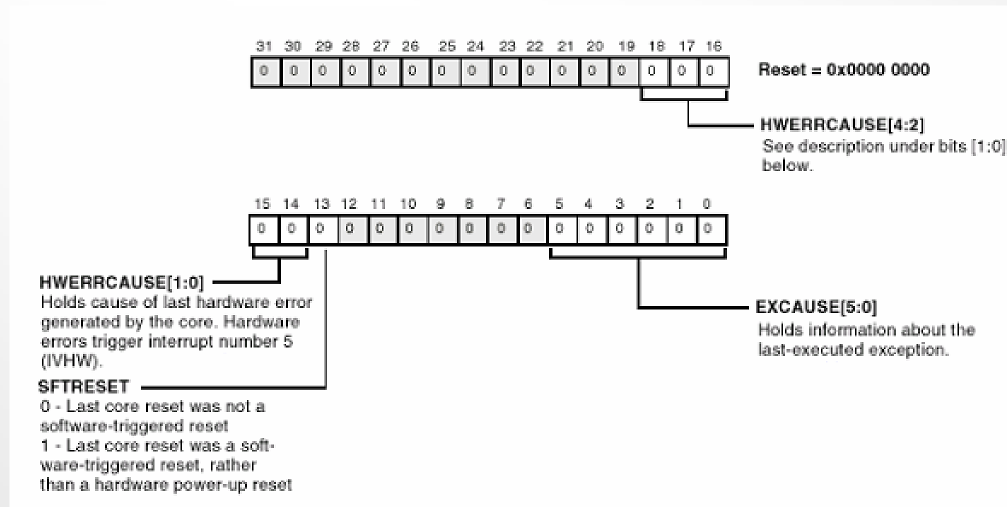
ANALOG DEVICES

Protection Model

- *User mode* protected instructions
  - RTI, RTX, RTN, RTE
  - CLI, STI
  - RAISE
  - IDLE
- *User mode* protected registers
  - RETI, RETX, RETN, RETE
  - SEQSTAT, SYSCFG
  - All Memory Mapped Registers

Sequencer Status Register (SEQSTAT)

- **SEQSTAT contains information about current Sequencer state and diagnostic information about the last event**

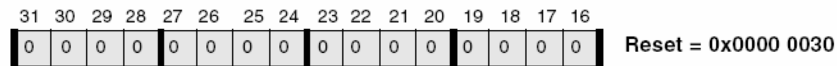


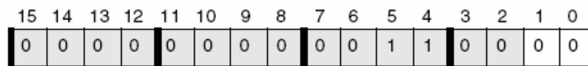
KAZTEK ENGINEERING

ANALOG DEVICES

BF533 System Configuration Register (SYSCFG)

- **SYSCFG controls the processor configuration.**





Must be set to 1

CCEN (Cycle Counter Enable)

0 - Disable 64-bit, free-running cycle counter

1 - Enable 64-bit, free-running cycle counter

SSSTEP (Supervisor Single Step)

When set, a Supervisor exception is taken after each instruction is executed. It applies only to User mode, or when processing interrupts in Supervisor mode. It is ignored if the core is processing an exception or higher priority event. If precise exception timing is required, CSYNC must be used after setting this bit.

KAZTEK ENGINEERING

ANALOG DEVICES

Hardware Loop Buffers

- The ADSP-BF533 DSP provides two sets of dedicated registers to support two zero-overhead nested loops
- One way to load these registers is by using the Loop Setup (LSETUP) instruction;

```
P5 = 0x20 ;
LSETUP ( lp_start, lp_end ) LC0 = P5 ;
lp_start:
R5 = R0 + R1 || R2 = [P2++] || R3 = [I1++] ;

lp_end:  R5 = R5 + R2 ;
```

Two sets of loop registers are used to manage two nested loops:

- LC[1:0] – the Loop Count registers
- LT[1:0] – the Loop Top address registers
- LB[1:0] – the Loop Bottom address registers

First/Last Address of the Loop	PC-Relative Offset Used to Compute the Loop Start Address	Effective Range of the Loop Start Instruction
Top / First	5-bit signed immediate; must be a multiple of 2.	0 to 30 bytes away from LSETUP instruction.
Bottom / Last	11-bit signed immediate; must be a multiple of 2.	0 to 2046 bytes away from LSETUP instruction (the defined loop can be 2046 bytes long).

When LCx = 0, the loop is disabled, and a single pass of the code executes.

- If the desired loop size exceeds the largest LSETUP size in the table above, LT[1:0], LB[1:0], LC[1:0] can be set manually
- If more than 2 nested loops are required, the stack must be used

KAZTEK ENGINEERING

ANALOG DEVICES

Hardware Loop Buffers

- **The two zero-overhead looping mechanisms each use a four-deep instruction “loop buffer” which acts like a cache**
- **The loop buffer instructions are the first four instruction of a loop**

```
lsetup(strt, end) lc0 = p4 >> 1;
strt:  a1 = r0.h * r1.l,  a0 = r0.l * r1.l (is)  || r0.l = w[i0++] || r2 = [i3++];
       a1 += r0.l * r1.h, a0 += r0.h * r1.h (is) || r0.h = w[i0--];
       a1 += r0.h * r2.l, a0 += r0.l * r2.l (is) || r0 = [i1++]    || r3 = [i3++];
       a1 += r0.h * r2.h, a0 += r0.l * r2.h (is) || r0.l = w[i1++];
       a1 += r0.l * r3.l, a0 += r0.h * r3.l (is) || r0.h = w[i1--] || r1 = [i3++];
       a1 += r0.h * r3.h, a0 += r0.l * r3.h (is) || r0 = [i2++];
       a1 += r0.h * r1.l, a0 += r0.l * r1.l (is) || r0.l = w[i2++] || r2 = [i3++];
       a1 += r0.l * r1.h, a0 += r0.h * r1.h (is) || r0.h = w[i2--] || r1 = [i3++];
       r6.h = (a1 += r0.h * r2.l), r6.l = (a0 += r0.l * r2.l) (is);
end: mnop || [p1++] = r6 || r0 = [i0++];
```

- **The loop buffer instructions get fetched the first time through the loop, and are immediately available on subsequent iterations**
- **The loop buffer is especially helpful if the program resides in external memory, because of the access latencies involved in that case**

KAZTEK ENGINEERING

ANALOG DEVICES

Loop-Related Stalls

- The ADSP-BF533 has two loop buffers that correspond to the two zero-overhead loop units. There are two situations to consider:
  - A 3 cycle stall is incurred if the LSETUP is not immediately followed by the loop top
  - If the first instruction of the loop is 64-bits, it must be 64-bit aligned or it will incur an additional 1 cycle stall

Event Controller

Events (Interrupts / Exceptions)

- **The Event Controller manages 5 types of Events:**
  - **Emulation (via SW or external pin)**
  - **Reset (via SW or external pin)**
  - **Non-Maskable Interrupt (NMI) - for events that require immediate processor attention (via SW or external pin)**
  - **Exception**
  - **Interrupts**
    - **Global Interrupt Enable**
    - **Hardware Error**
    - **Core Timer**
    - **9 General-Purpose Interrupts for servicing peripherals**

Interrupts vs. Exceptions

INTERRUPTS

- Hardware-generated
  - Asynchronous to program flow
  - Requested by a peripheral
- Software-generated
  - Synchronous to program flow
  - Generated by RAISE instruction
- All instructions preceding the interrupt in the pipeline are killed

EXCEPTIONS

- Service Exception
  - Return address is the address following the excepting instruction
  - Never re-executed
  - EXCPT instruction is in this category
- Error Condition Exception
  - Return address is the address of the excepting instruction
  - Excepting instruction will be re-executed

The ADSP-BF533 is always in Supervisor Mode while executing Event Handler software and can be in User Mode only while executing application tasks.

KAZTEK ENGINEERING

ANALOG DEVICES

Exception Causes

Priority	Exception	EXCAUSE
1	Unrecoverable Event	0x25
2	I-Fetch Multiple CPLB Hits	0x2D
3	I-Fetch Misaligned Access	0x2A
4	I-Fetch Protection Violation	0x2B
5	I-Fetch CPLB Miss	0x2C
6	I-Fetch Access Exception	0x29
7	Watchpoint Match	0x28
8	Undefined Instruction	0x21
9	Illegal Combination	0x22
10	Illegal use protected resource	0x2E
11	DAG0 Multiple CPLB Hits	0x27
12	DAG0 Misaligned Access	0x24
13	DAG0 Protection Violation	0x23
14	DAG0 CPLB Miss	0x26
15	DAG1 Multiple CPLB Hits	0x27
16	DAG1 Misaligned Access	0x24
17	DAG1 Protection Violation	0x23
18	DAG1 CPLB Miss	0x26
19	EXCPT instruction	m- field
20	Single Step	0x10
21	Trace Buffer	0x11

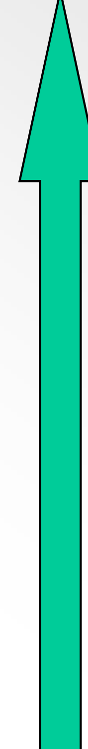
KAZTEK ENGINEERING

ANALOG DEVICES

Event Priorities

Event Source	IVG #	Core Event Name
Emulator	0	EMU
Reset	1	RST
Non Maskable Interrupt	2	NMI
Exceptions	3	EVSW
Reserved	4	Reserved
Hardware Error	5	IVHW
Core Timer	6	IVTMR
General Purpose 7	7	IVG7
General Purpose 8	8	IVG8
General Purpose 9	9	IVG9
General Purpose 10	10	IVG10
General Purpose 11	11	IVG11
General Purpose 12	12	IVG12
General Purpose 13	13	IVG13
General Purpose 14	14	IVG14
General Purpose 15	15	IVG15

Highest



Lowest

BF533 System and Core Interrupt Controllers

System Interrupt Source	IVG #[1]
PLL Wakeup interrupt	IVG7
DMA error (generic)	IVG7
PPI error interrupt	IVG7
SPORT0 error interrupt	IVG7
SPORT1 error interrupt	IVG7
SPI error interrupt	IVG7
UART error interrupt	IVG7
RTC interrupt	IVG8
DMA 0 interrupt (PPI)	IVG8
DMA 1 interrupt (SPORT0 RX)	IVG9
DMA 2 interrupt (SPORT0 TX)	IVG9
DMA 3 interrupt (SPORT1 RX)	IVG9
DMA 4 interrupt (SPORT1 TX)	IVG9
DMA 5 interrupt (SPI)	IVG10
DMA 6 interrupt (UART RX)	IVG10
DMA 7 interrupt (UART TX)	IVG10
Timer0 interrupt	IVG11
Timer1 interrupt	IVG11
Timer2 interrupt	IVG11
PF interrupt A	IVG12
PF interrupt B	IVG12
DMA 8/9 interrupt (MemDMA0)	IVG13
DMA 10/11 interrupt (MemDMA1)	IVG13
Watchdog Timer Interrupt	IVG13

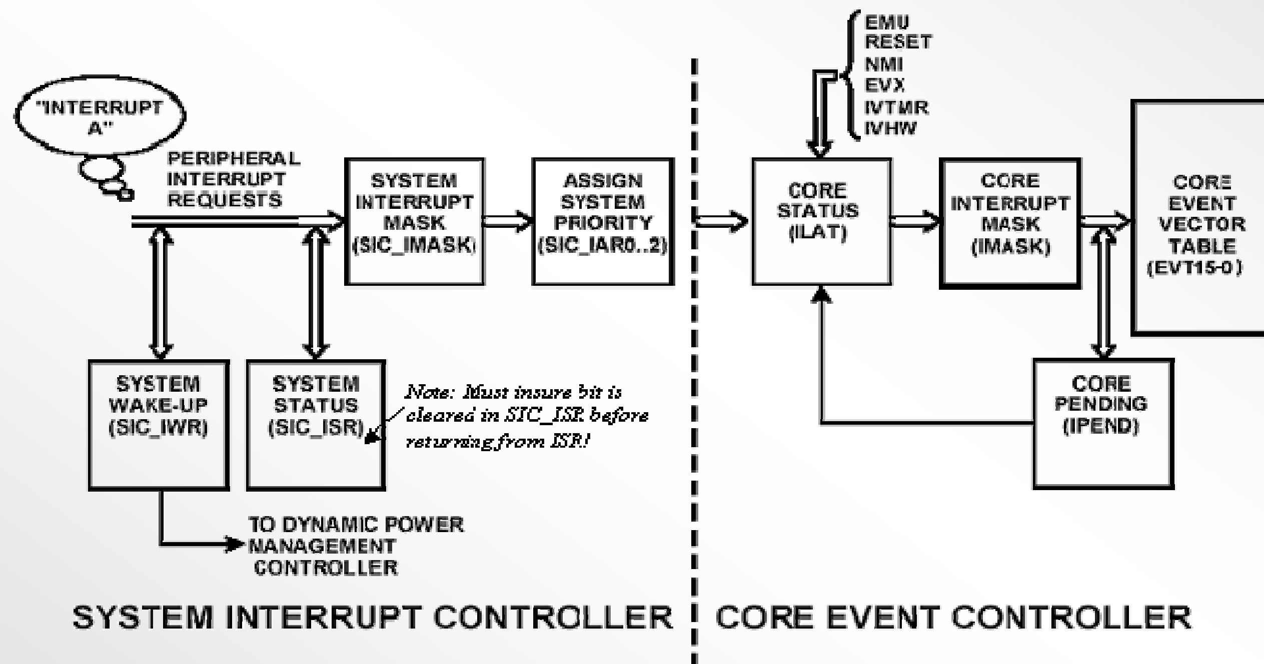
Event Source	IVG #	Core Event Name
Emulator	0	EMU
Reset	1	RST
Non Maskable Interrupt	2	NMI
Exceptions	3	EVSW
Reserved	4	-
Hardware Error	5	IVHW
Core Timer	6	IVTMR
General Purpose 7	7	IVG7
General Purpose 8	8	IVG8
General Purpose 9	9	IVG9
General Purpose 10	10	IVG10
General Purpose 11	11	IVG11
General Purpose 12	12	IVG12
General Purpose 13	13	IVG13
General Purpose 14	14	IVG14
General Purpose 15	15	IVG15

[1] Note: Default IVG configuration shown.

KAZTEK ENGINEERING

ANALOG DEVICES

Event Processing Flow

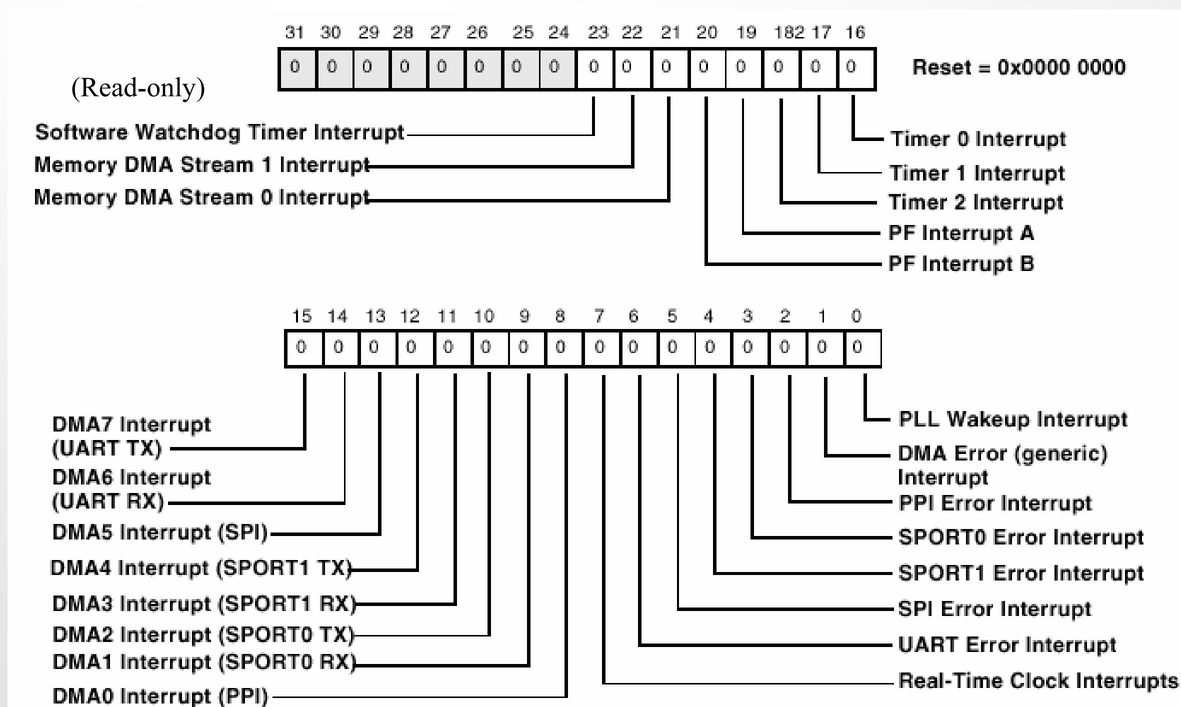


KAZTEK ENGINEERING

ANALOG DEVICES

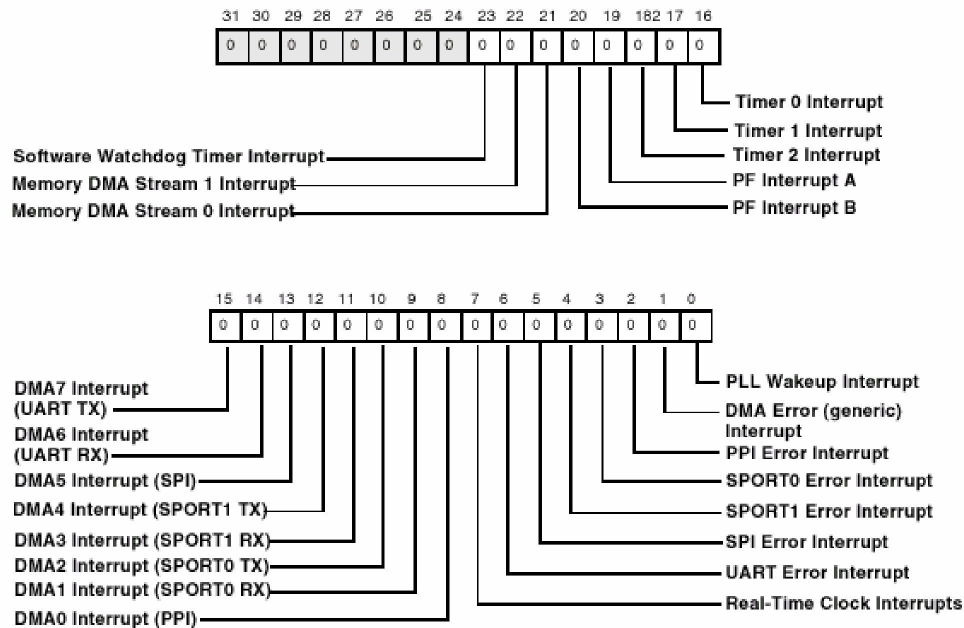
BF533 System Interrupt Status Register (SIC_ISR)

- SIC_ISR shows which peripheral interrupts are currently asserted
- Must insure the interrupt-generating mechanism that set the SIC_ISR bit is cleared before exiting the service routine, or the interrupt will be requested again!



System Interrupt Mask Register (SIC_IMASK)

- **Enable the peripheral to interrupt the core by setting the corresponding bit in SIC_IMASK**

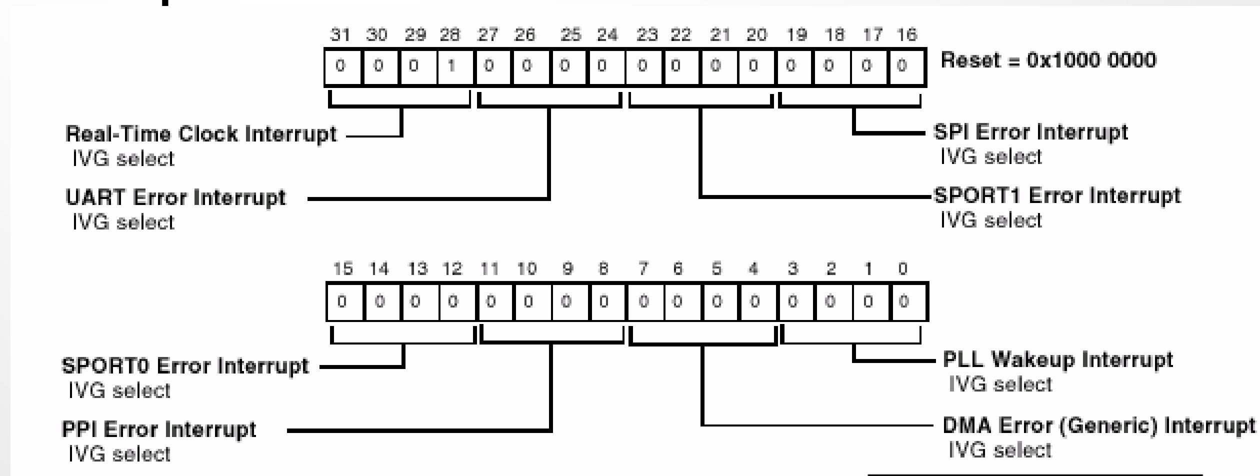


KAZTEK ENGINEERING

ANALOG DEVICES

System Interrupt Assignment Register 0 (SIC_IAR0)

- **The SIC_IARx registers map system interrupts to core IVG interrupt vectors.**

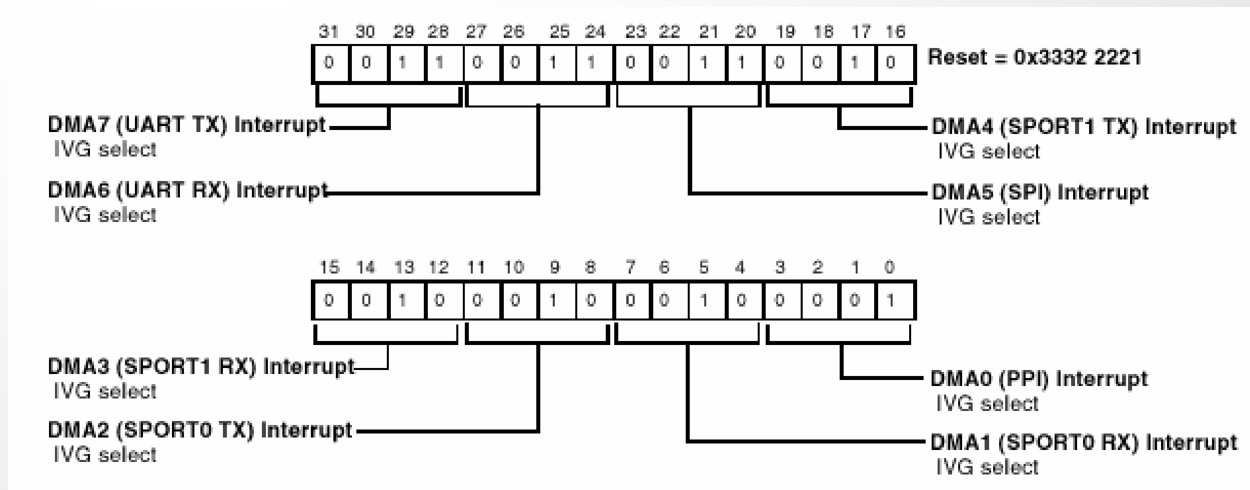


General-Purpose Interrupt	Value in SIC_IAR
IVG7	0
IVG8	1
IVG9	2
IVG10	3
IVG11	4
IVG12	5
IVG13	6
IVG14	7
IVG15	8

KAZTEK ENGINEERING

ANALOG DEVICES

System Interrupt Assignment Register 1 (SIC_IAR1)

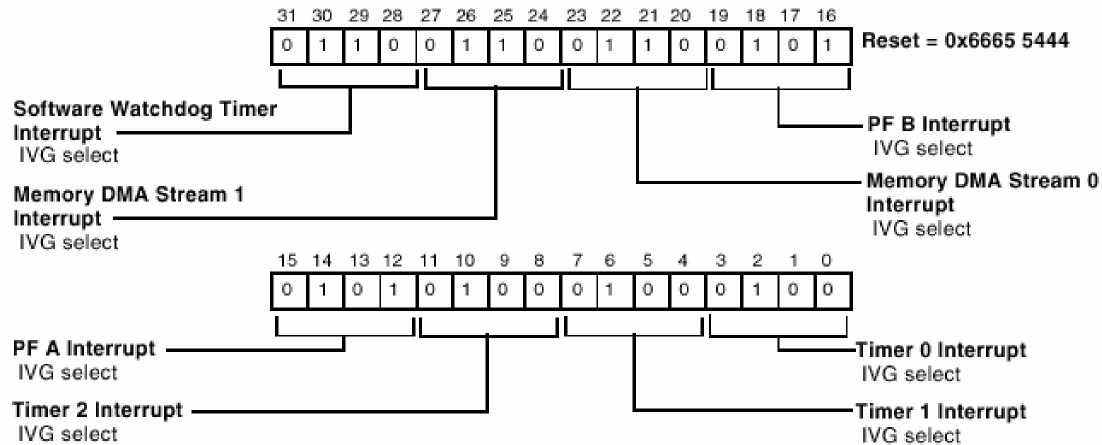


General-Purpose Interrupt	Value in SIC_IAR
IVG7	0
IVG8	1
IVG9	2
IVG10	3
IVG11	4
IVG12	5
IVG13	6
IVG14	7
IVG15	8

KAZTEK ENGINEERING

ANALOG DEVICES

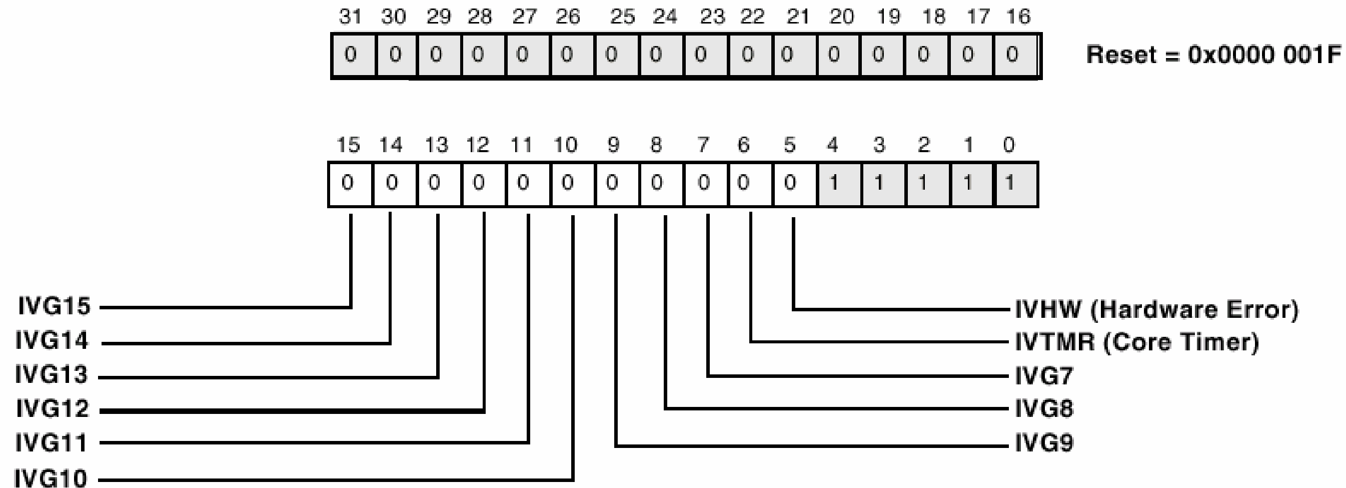
System Interrupt Assignment Register 2 (SIC_IAR2)



General-Purpose Interrupt	Value in SIC_IAR
IVG7	0
IVG8	1
IVG9	2
IVG10	3
IVG11	4
IVG12	5
IVG13	6
IVG14	7
IVG15	8

Core Interrupt Mask Register (IMASK)

- **Choose which interrupt to allow servicing of by setting that bit in IMASK**



KAZTEK ENGINEERING

ANALOG DEVICES

Non-interruptible code

- **Instruction CLI disables interrupts**
  - **copies current IMASK to a Dreg**
  - **clears IMASK**
- **Instruction STI restores IMASK**
- **Change to IMASK should be done with CLI**

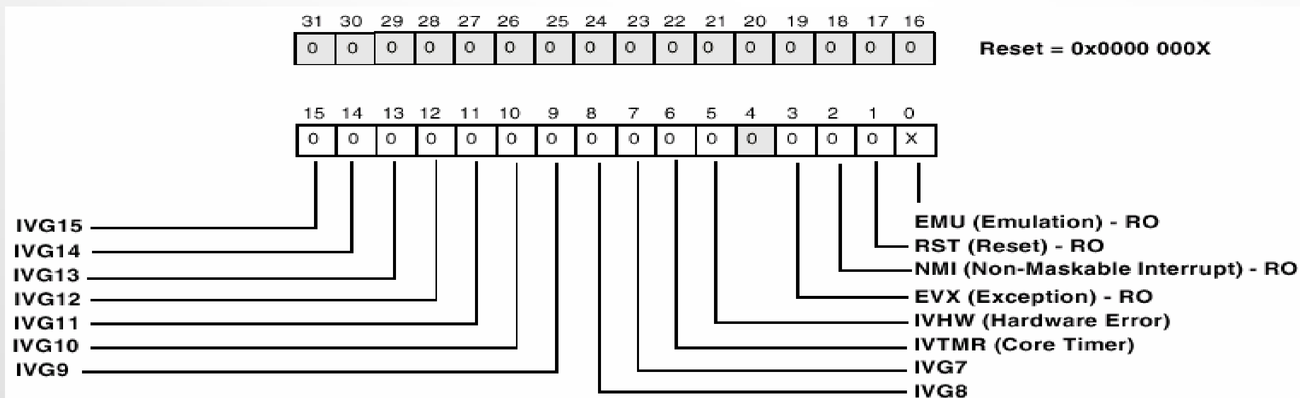
```
CLI R0;  //Save IMASK into R0 & clear all interrupt bits in IMASK
BITSET(R0,8); //Set bit 8 of R0
STI R0;       //Restore IMASK with change
```

KAZTEK ENGINEERING

ANALOG DEVICES

Core Interrupt Latch Register (ILAT)

- A set bit in ILAT indicates when the corresponding event has been latched
- The bit is cleared upon entry into the Interrupt Service Routine or by writing a "1" to ILAT[n] when IMASK[n] = 0. (n=5-15)
- RAISE n Instruction (n = 1, 2, 5-15)
  - Forces a bit to be set in ILAT. It 'raises' the priority of the execution
- EXCPT n Instruction (n= 0-15)
  - Forces an exception to occur : EVSW bit is set in ILAT and 'n' determines which exception routine to execute

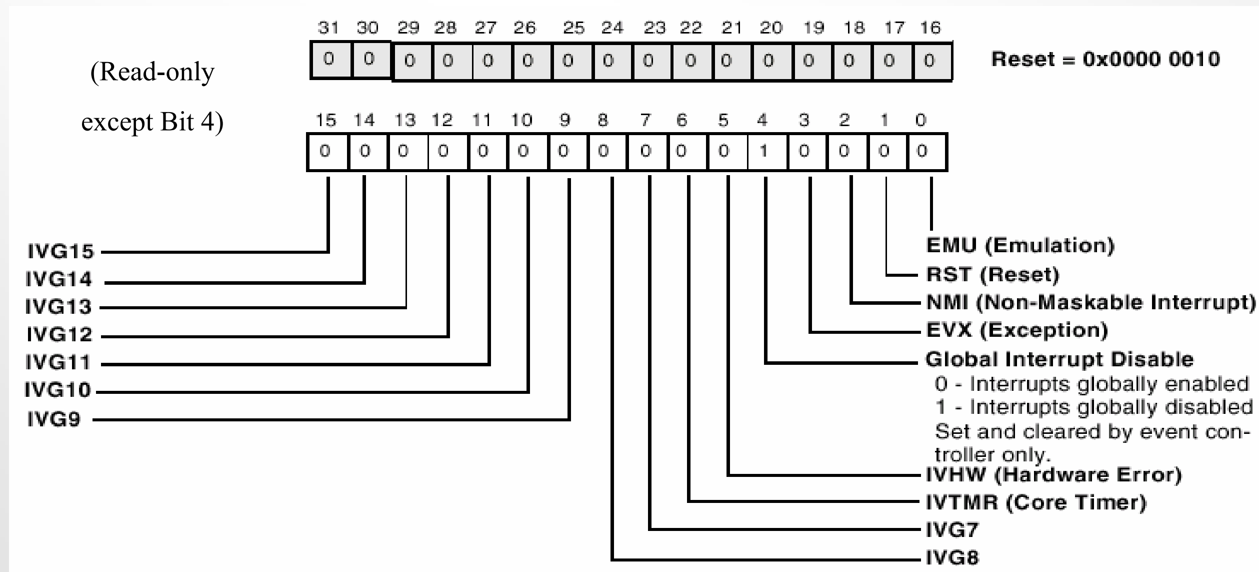


KAZTEK ENGINEERING

ANALOG DEVICES

Core Interrupt Pending Register (IPEND)

- IPEND tracks currently active or nested interrupts
- IPEND holds current status of all nested events.
- Rightmost bit in IPEND that is currently set indicates interrupt currently being serviced



KAZTEK ENGINEERING

ANALOG DEVICES

Event Vector Table (EVT)

- Memory-mapped space containing an entry for each event EVT0-EVT15, corresponding to EMU, RST, NMI, … IVG15
  - HW Table with 16 32-bit entries accessed as CORE MMRs
  - EVT0-EVT15 are undefined at Reset
  - Entries should be programmed in the Reset service routine with the corresponding Interrupt Service Routine vector
- Each entry holds starting address for that event
- When Event #n occurs, instruction fetch starts at address location stored in EVTn

Interrupt Service Routine

- Interrupt vector from the Event Vector Table is used as the next fetch address
- Return address is saved
  - RETI, RETX, RETN, RETE based on event
  - All interrupts are disabled until the return address (RETI) is pushed on the stack
- Processor operating mode is set to supervisor or emulation

KAZTEK ENGINEERING

ANALOG DEVICES

Nested Interrupts

- **To enable another higher priority interrupt to interrupt this interrupt RETI must be pushed on the stack.**
- **The state of the processor needs to be saved onto the Stack:**

```
ISR:
[--SP] = RETI;  // Interrupts enabled
[--SP] = ASTAT;
[--SP] = FP;
[--SP] = (Rx,Ax,Px,Ix);
…
(Rx,Ax,Px,Ix) = [SP++];
FP = [SP++];
ASTAT = [SP++];
RETI = [SP++];  // Interrupts disabled
CSYNC;          // Wait until RETI load takes effect, may want to use
                // SSYNC to confirm system writes have committed.
RTI;            // Interrupts enabled
```

KAZTEK ENGINEERING

ANALOG DEVICES

Non-nested Interrupts

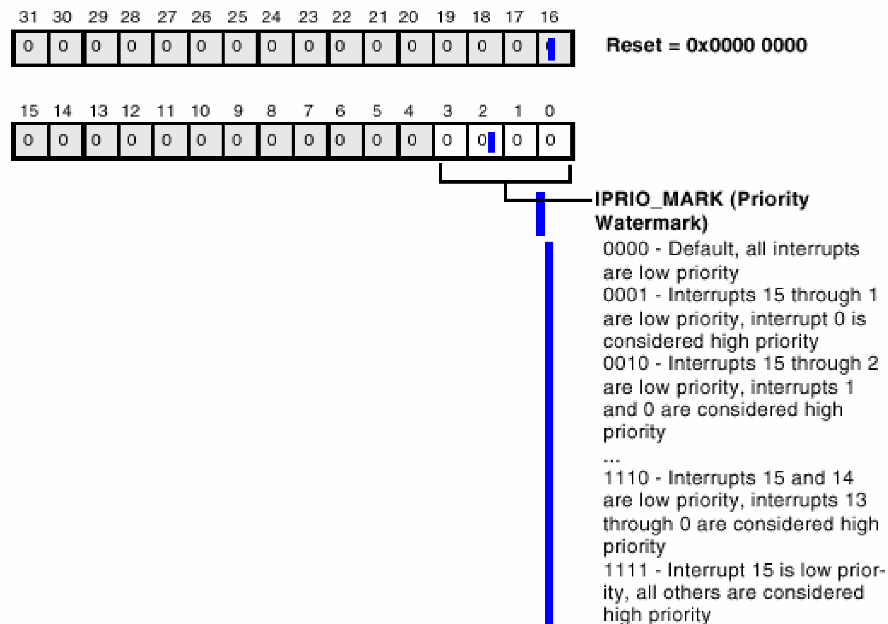
- RETI does not need to be saved on the stack
- All interrupts remain disabled in the ISR
- The state of the processor should be saved on the stack
- RTI is executed to return from interrupt
- Emulation, NMI and Exceptions are still accepted by the system

Deferring Exception Processing

- **Exceptions higher priority than interrupts**
- **If exception handlers are long routines, interrupts are disabled for long time**
- **To avoid this situation, the exception handler should be written to only identify the exception (EXCAUSE field in SEQSTAT register) and defer the exception processing to a low priority interrupt by using the RAISE n instruction**

Interrupt Priority Register

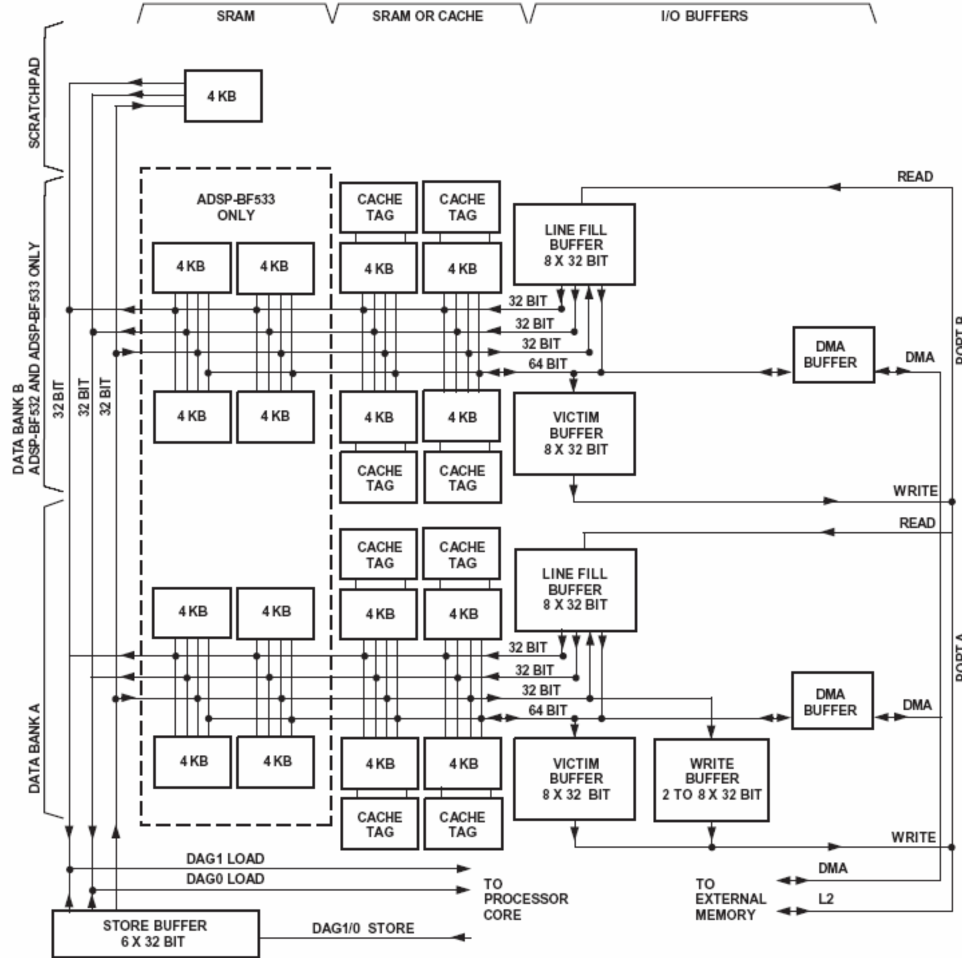
- **When code branches from a low-priority interrupt to a high-priority interrupt the write-buffer increase in size from 2 deep to 8 deep to off-load the store buffer.**
- **Frees path to L1 memory (IE: context saving to scratchpad)**
- **When code returns from a high-priority interrupt to a low-priority, the core will stall until the write-buffer size decreases back to 2 deep.**



KAZTEK ENGINEERING

ANALOG DEVICES

BF533 L1 Data Memory



Victim Buffers:
Victimized Write-Back Cached Data to external memory

Write Buffer:
Write-Through and Non-cached Data to external memory

KAZTEK ENGINEERING

ANALOG DEVICES

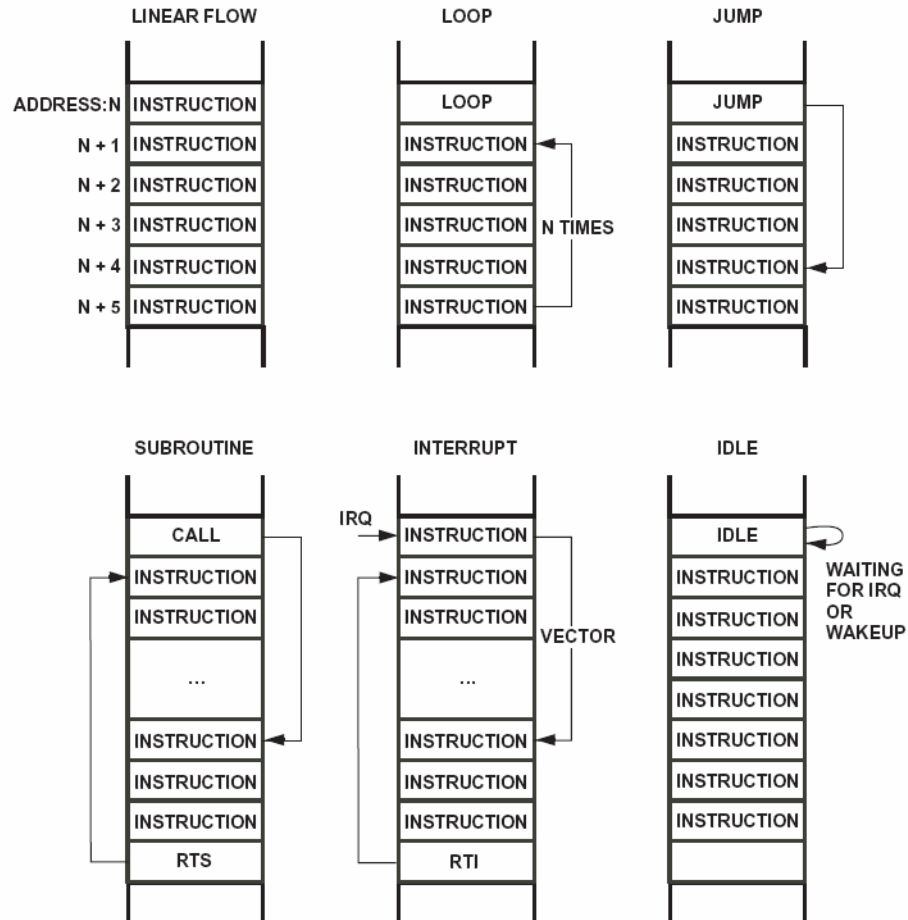
Reference Material

Sequencer

KAZTEK ENGINEERING

ANALOG DEVICES

Variations in Program Flow



KAZTEK ENGINEERING

ANALOG DEVICES

Multi-Cycle Instructions

- A 32-bit multiply operation is available
  - `r0 *= r1; // 3 cycles to execute`
- The Push Multiple and Pop Multiple instructions take n cycles to complete, where n is the number of registers pushed or popped, assuming L1 memory.
  - `[--SP] = (R7:0, P5:0); // 14 cycles to execute`
- Multi-cycle instructions will not execute faster through rescheduling.
- See EE-197 Appnote for a complete list of stalls and multicycle instructions

Event Processing Flow

Clear SIC_ISR

"Event A"

Awake from IDLE!

IVTMR
IVHW

Watchdog
MEMDMA1
MEMDMA0
PF B
PF A
Timer 2
Timer 1
Timer 0
DMA7 (UART TX)
DMA6 (UART RX)
DMA5 (SPI)
DMA4 (Sport1 TX)
DMA3 (Sport1 RX)
DMA2 (Sport0 TX)
DMA1 (Sport0 RX)
DMA0 (PPI)
RTC
.....
DMA Error
PLL Wakeup

1

SIC_ISR

SIC_IWR

SIC_IMASK

SIC_IARx

EVSW
NMI
RST
EMU

FFA0 1000	[- -sp] = P4;
FFA0 1002	[- -sp] = R4;
FFA0 1004	R4.l = 0x1000;
FFA0 1008	P4.H = hi(FLAG_C);
FFA0 100C	P4.L = lo(FLAG_C);
FFA0 1010	W[P4] = R4;
FFA0 1012	R4 = [sp++];
FFA0 1014	P4 = [sp++];
FFA0 1016	RTI;

Clear IPEND

Keep ILAT Cleared

1 ILAT

0

1 IPEND

R0

IMASK

CLI R0; // 0000

STI R0; // IMASK = R0;

IVG15 IVG14 IVG13 IVG12 IVG11 IVG10 IVG9 IVG8 IVG7 IVG6 IVG5 IVG3 IVG2 IVG1 IVG0

0xFFA0 1000

EVT

CORE PIPE LINE

Core Event Control Registers

Bit	Event Name	Description	ILAT=1 means	IPEND=1 means	IMASK=1 means
0	EMU	Emulation Event	Event latched	Event active	*<reserved>*
1	RST	Reset Event	Event latched	Event pending or active	*<reserved>*
2	NMI	Non-maskable Interrupt	Interrupt latched	Interrupt pending or active	*<reserved>*
3	EVX	Exception Event	Event latched	Event pending or active	*<reserved>*
4		Global Interrupt Disable	*<reserved>*	Interrupts globally disabled	*<reserved>*
5	IVHW	Hardware Error Interrupt	Interrupt latched	Interrupt pending or active	Interrupt enabled
6	IVTMR	Core Timer Interrupt	Interrupt latched	Interrupt pending or active	Interrupt enabled
7-15	IVG7-15	General Purpose Interrupts #7-#15	Respective interrupt latched	Respective interrupt pending or active	Respective interrupt enabled

Semaphores

Semaphores

- Semaphores provide a way of signaling between separate processes
  - A background task may be waiting for a semaphore that may be provided by an ISR before it can start. The semaphore could indicated the presence of a new buffer of data.
  - CoreA and CoreB could be sharing a buffer in memory, but only one can access at a time. A semaphore would be used to provide exclusive access by one core or the other.

- For a mutex (multual exclusion) to be effective, one must be able to check a semaphore to see if a resource is free and then set a bit to claim it before the other processor has a chance to claim it. In other words, the read-check-modify-write must be atomic.

Testset

Testset(preg)

Example

```
testset(p1);
```

- **This instruction reads the byte pointed to by preg, sets the MSB, and stores the byte back into memory. If the byte was originally zero, the CC bit is set. If the byte was originally nonzero, the CC bit is cleared.**
  - **Typically, a zero is used to indicated a free resource. If CC tests true, the resource is now claimed exclusively for the process. When done with the resource, the process must clear the semaphore.**
  - **If CC tests false, it indicates that the resource is being used. Typically, the process waits until the resource becomes free by spinning in a tight loop.**