High-Performance dsPIC33A Core with Floating-Point Unit, High-Speed ADCs and High-Resolution PWM

dsPIC33AK128MC106 Family

Operating Conditions

- 3.0V to 3.6V: -40°C to +85°C, DC to 200 MHz
- 3.0V to 3.6V: -40°C to +125°C, DC to 200 MHz
- 3.0V to 3.6V: -40°C to +150°C, DC to 200 MHz

High-Performance dsPIC33A DSP/CISC CPU

- 32-bit Comprehensive Instruction Set for Optimized Speed and Program Code Size:
  - 16-bit dsPIC33 core compatible
  - Non-paged linear Data/Flash 24-bit addressing space
  - 16-bit/32-bit instructions for optimized code size and performance
- 32-bit Wide Data Paths
- Single and Double Precision Floating-Point Unit (FPU) Coprocessor
- 2 Kbyte Instruction Cache
- Sixteen 32-bit Working Registers
- Dual 72-bit Accumulators Supporting 32-bit and 16-bit Fixed-Point DSP Operations
- Eight Level Deep Working Register Contexts
- Eight Level Deep Accumulator Register Contexts
- Eight Level Deep Floating Point Register Contexts

Memory Features

- Up to 128 Kbytes of Program Flash Memory:
  - 10,000 erase/write cycle endurance
  - 20 years minimum data retention
  - Self-programmable under software control
  - Programmable code protection
  - Flash Error Correcting Code (ECC)
  - Programmable OTP regions
  - Entire Flash OTP by ICSP™ write inhibit
  - 64 x128-bit OTP area
- Up to 16 Kbytes of RAM Memory:
  - 6-channel hardware Direct Memory Access (DMA) module
  - RAM Error Correcting Code (ECC)
  - RAM Memory Built-In Self-Test (MBIST)

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Controller Features

- High-Current Sink/Source Capable I/Os
- Programmable Weak Pull-Up and Pull-Down Resistors
- Programmable Open-Drain Outputs
- Edge or Level Change Notification Interrupt on I/O pins
- Peripheral Pin Select (PPS) Remappable Pins to Reduce Board Layout Complexity
- Multiple Interrupt Vectors with Individual Programmable Priority
- Five External Interrupt Pins
- Selectable Oscillator Options Including:
  - 8 MHz, 1% at 0°C-85°C Internal Fast RC (FRC) oscillator
  - 8 MHz, 2% Internal Backup Fast RC (BFRC) oscillator with 32 kHz divided output
  - High-speed crystal resonator oscillator or external clock
- Two 1.6 GHz PLLs for Peripheral which can be clocked from the FRC or a Crystal Oscillator
- Reference Clock Output (REFO)
- Low-Power Modes (Sleep and Idle)
- Power-On Reset and Brown-Out Reset

High-Speed PWM

- Four PWM Generators (Four Pairs, Eight Outputs)
- Up to 2.5 ns PWM Resolution
- Dead Time for Rising and Falling Edges
- Dead-Time Compensation Supports Lower Speed Operation
- PWM Support for:
  - BLDC, PMSM, ACIM, SRM and Stepper Motors
- Fault and Current Limit Inputs
- Flexible Trigger Configuration for ADC Triggering

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Two High-Speed Analog-to-Digital Converters

- 12-bit Resolution
- Up to 40 Msps Conversion Rate
- Up to 22 Analog Input Pins
- 20 Settings Channels. Each Channel:
  - Supports Discrete Configuration
  - Can be assigned to any analog input (I/O pin or internal signal)
  - Can be set to a different sampling time
  - Can be configured as single-ended or differential
  - Conversion result can be formatted as unsigned or signed
  - Conversion result can be left-aligned (fraction format)
  - Has a separate 32-bit conversion result register
- Supports Four Sampling modes:
  - Oversampling of multiple samples
  - Integration of multiple samples
  - Window (multiple samples accumulated when the gate signal is active)
  - Single Conversion
  - All channels have a digital comparator to detect when the conversion result is less than, greater than, in bounds or out of bounds for the configurable thresholds
  - Three channels support second result accumulator to implement second order filters
- Band Gap Reference and Temperature Sensor Diode Inputs

Other Analog Features

- Three 5 nS Analog Comparators with 12-bit Pulse Density Modulation DACs:
  - Input multiplexing
  - Slope compensation
  - One DAC output buffer
- Three Rail-to-Rail 100 MHz Operational Amplifiers with:
  - 100 V/µS slew rate
  - 1 mV offset (typical)
  - User calibration of input offset voltage
- Four 10 µA Constant Sources + Four Programmable Sources

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Peripheral Features

- Three 4-Wire SPI Modules:
  - 4-byte FIFO
  - Variable data width
  - I^2S mode
- Two I^2C modules:
  - Independent Host and Client Logic
  - Supports 100 kHz, 400 kHz and 1 MHz Bus Specifications
  - 7-bit and 10-bit Device Addresses
  - Supports IPMI Standard, SMBus and PMBus
- Three Protocol UARTs with 8-Character RX/TX FIFOs and Automated Handling Support for:
  - LIN 2.2
  - Digital Multiplex 512 (DMX)
  - Smart Card (ISO 7816)
  - IrDA®
- Two Single-Edge Nibble Transmission (SENT) Modules
- One Dedicated 32-bit Timer/Counter
- Four Single Output Capture/Compare/PWM/Timer (SCCP) Modules:
  - Flexible configuration as PWM, input capture, output compare or timers
  - Two 16-bit timers or one 32-bit timer in each module
  - Single PWM output pin
- One Quadrature Encoder Interface (QEI):
  - Four inputs: Phase A, Phase B, Home, Index
- Four Configurable Logic Cells (CLC) with Internal Connections to Select Peripherals and PPS
- Bidirectional Serial Synchronous (BiSS) Encoder Interface with up to Four Client Encoders Support
- Peripheral Trigger Generator (PTG):
  - 10 input trigger sources from other peripheral modules
  - 5 output triggers to other peripheral modules
  - 4 individual interrupt request signals
  - CPU independent state machine-based instruction sequencer

Security Module

- Secure Boot
- Secure Debug
- Immutable Root of Trust (IRT)
- Code Protect
- ICSP Program/Erase Disable (Entire Flash OTP by ICSP Write Inhibit)
- Firmware IP Protection
- Flash Write Protection

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Safety Features

- Windowed Watchdog Timer (WDT)
- Deadman Timer (DMT)
- Four I/O Integrity Monitors (IOIM)
- Fail-Safe Clock Monitor (FSCM) with Automatic Switchover to Backup Clock Source with:
  - Programmable over-frequency/under-frequency thresholds
- Flash Error Correcting Code (NVM ECC)
- RAM Error Correcting Code (RAM ECC)
- RAM Memory Built-In Self-Test (MBIST)
- 32-bit Cyclic Redundancy Check (CRC) Module
- Entire Flash OTP by ICSP™ Write Inhibit
- Capless Internal Voltage Regulator
- Virtual PPS Pins for Redundancy and Monitoring
- Temperature Sensor Diode

Functional Safety

Functional Safety Readiness – ISO 26262/IEC 61508/IEC 60730

To learn about the Functional Safety Readiness of this device family and various Functional Safety standards an application can target using this device family, visit www.microchip.com/dsPIC33-Functional-Safety

Qualification

AEC-Q100 REV H:
- Grade 1: -40°C to +125°C
- Grade 0: -40°C to +150°C Planned

Programming and Debug Interfaces

- Three Programming and Debugging Interfaces:
  - Two-wire ICSP™ interface with non-intrusive access and real-time data exchange with application
- Five Program Addresses and Five Full-Featured Breakpoints
- IEEE Standard 1149.2 Compatible (JTAG) Boundary Scan

Targeted Applications

- Power Factor Correction (PFC):
  - Interleaved PFC
  - Critical Conduction PFC
  - Bridgeless PFC
- DC/DC Converters:
  - Buck, Boost, Forward, Flyback, Push-Pull
  - Half/Full-Bridge
  - Phase-Shift Full-Bridge

© 2023-2024 Microchip Technology Inc. and its subsidiaries

  - Resonant Converters
- DC/AC:
  - Half/Full-Bridge Inverter
  - Resonant Inverter
- Motor Control:
  - BLDC
  - PMSM
  - SR
  - ACIM
- Advanced Sensor Interfacing
- High-Performance Embedded Control
- Safety-Critical Designs
- Digital Lighting

© 2023-2024 Microchip Technology Inc. and its subsidiaries

dsPIC33AK128MC106 Product Family

The dsPIC33A family names, pin counts, memory sizes and peripheral availability of each device are listed in Table 1, and their pinout diagrams are included as well.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Table 1. dsPIC33AK128MC106 Family

Product	Pins	Program Memory (Kbytes)	Data Memory (Kbytes)	General Purpose I/O/PPS	High-Resolution PWM (Generator Pairs)	Two 12-bit ADCs (External Analog Inputs)	Remappable Peripherals								Op Amplifiers	Comparators	12-bit DACs	I2C	32-bit CRC	DMA (Channels)	Packages
							Dedicated 32-bit Timers	UART	BISS	SCCP(1)	CLC	SPI/I2S	SENT	QEI							
dsPIC33AK32MC102	28	32	8	19/19	4*2	11	1	3	1	4	4	3	2	1	2	3	3	2	1	6	SSOP/VQFN
dsPIC33AK32MC103	36	32	8	27/27	4*2	15	1	3	1	4	4	3	2	1	3	3	3	2	1	6	VQFN
dsPIC33AK32MC105	48	32	8	35/35	4*2	18	1	3	1	4	4	3	2	1	3	3	3	2	1	6	VQFN/TQFP
dsPIC33AK32MC106	64	32	8	49/49	4*2	22	1	3	1	4	4	3	2	1	3	3	3	2	1	6	VQFN/TQFP
dsPIC33AK64MC102	28	64	16	19/19	4*2	11	1	3	1	4	4	3	2	1	2	3	3	2	1	6	SSOP/VQFN
dsPIC33AK64MC103	36	64	16	27/27	4*2	15	1	3	1	4	4	3	2	1	3	3	3	2	1	6	VQFN
dsPIC33AK64MC105	48	64	16	35/35	4*2	18	1	3	1	4	4	3	2	1	3	3	3	2	1	6	VQFN/TQFP
dsPIC33AK64MC106	64	64	16	49/49	4*2	22	1	3	1	4	4	3	2	1	3	3	3	2	1	6	VQFN/TQFP
dsPIC33AK128MC102	28	128	16	19/19	4*2	11	1	3	1	4	4	3	2	1	2	3	3	2	1	6	SSOP/VQFN
dsPIC33AK128MC103	36	128	16	27/27	4*2	15	1	3	1	4	4	3	2	1	3	3	3	2	1	6	VQFN
dsPIC33AK128MC105	48	128	16	35/35	4*2	18	1	3	1	4	4	3	2	1	3	3	3	2	1	6	VQFN/TQFP
dsPIC33AK128MC106	64	128	16	49/49	4*2	22	1	3	1	4	4	3	2	1	3	3	3	2	1	6	VQFN/TQFP

Note:

1. SCCP can be configured as a PWM with one output, input capture, output compare, 2 x 16-bit timers or 1 x 32-bit timer.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Pin Diagrams

Figure 1. 28-Pin SSOP

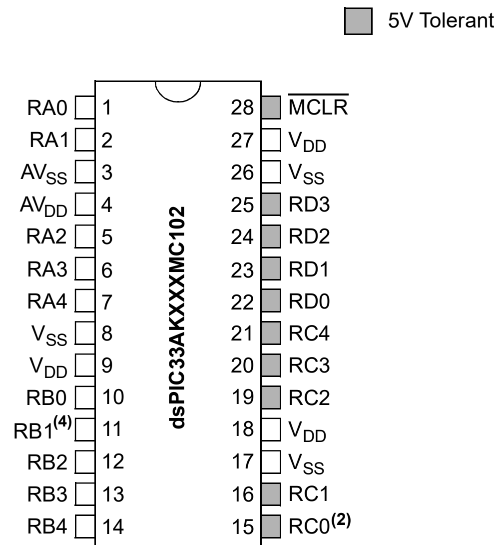


Table 2. 28-Pin SSOP Complete Pin Function Descriptions[1,3]

Pin	Function	Pin	Function
1	PGD2/AD2AN6/CMP3C/ISRC2/IBIAS2/**RP1**/SDA2/IOMF2/RA0	15	OSCO/CLKO/**RP33**/IOMF5/RC0[2]
2	PGC2/DACOUT1/AD1AN7/AD2AN3/CMP1D/CMP2D/ CMP3D/**RP2**/SCL2/RA1	16	OSCI/CLKI/**RP34**/IOMF6/RC1
3	AV_{SS}	17	V_{SS}
4	AV_{DD}	18	V_{DD}
5	OA1OUT/AD1AN0/CMP1A/**RP3**/RA2	19	PGC3/**RP35**/PWM4H/RC2
6	OA1IN-/AD1ANN1/AD2AN0/**RP4**/RA3	20	PGD3/**RP36**/PWM3H/IOMD0/RC3
7	OA1IN+/AD1AN1/CMP1B/**RP5**/RA4	21	**RP37**/PWM3L/IOMD1/RC4
8	V_{SS}	22	**RP49**/PWM2H/IOMD2/RD0
9	V_{DD}	23	TCK/**RP50**/PWM2L/IOMD3/RD1
10	OA2OUT/AD2AN1/CMP2A/**RP17**/INT0/RB0	24	TDO/**RP51**/PWM1H/IOMD4/RD2
11	TMS/OA2IN-/AD1AN4/AD2ANN1/**RP18**/RB1[4]	25	TDI/**RP52**/PWM1L/IOMD5/RD3
12	OA2IN+/AD2AN4/CMP2B/**RP19**/RB2	26	V_{SS}
13	PGD1/AD1AN5/CMP1C/ISRC0/IBIAS0/**RP20**/SDA1/RB3	27	V_{DD}
14	PGC1/AN2AN5/CMP2C/ISRC1/IBIAS1/**RP21**/SCL1/RB4	28	$\overline{MCLR}$

Note:

1. **RPn** represents remappable peripheral functions.
2. This pin has 8x drive strength.
3. Unless otherwise stated, pins are 4x drive strength. Refer to Electrical Specifications for current drive strength details.
4. A pull-up resistor is connected to this pin when device is erased (JTAG enabled) and during programming.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Pin Diagrams

Figure 2. 28-Pin VQFN

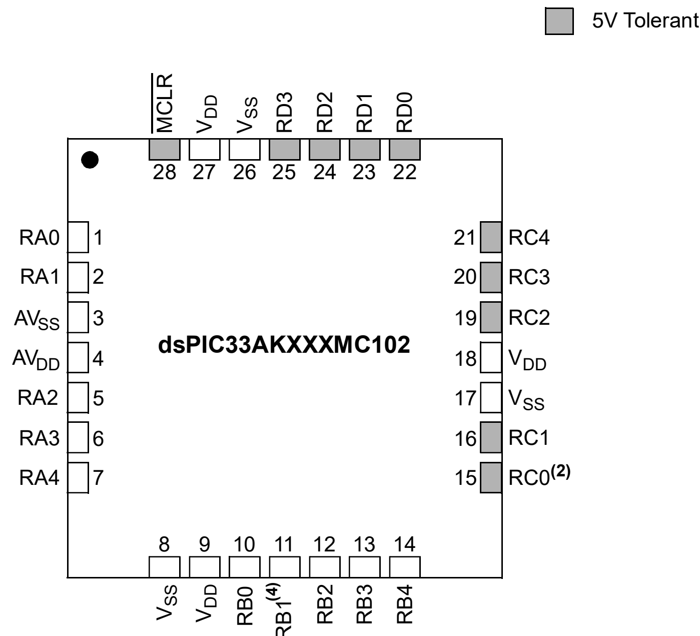


Table 3. 28-Pin VQFN Complete Pin Function Descriptions(1,3)

Pin	Function	Pin	Function
1	PGD2/AD2AN6/CMP3C/ISRC2/IBIAS2/**RP1**/SDA2/IOMF2/RA0	15	OSCO/CLKO/**RP33**/IOMF5/RC0(2)
2	PGC2/DACOUT1/AD1AN7/AD2AN3/CMP1D/CMP2D/ CMP3D/**RP2**/SCL2/RA1	16	OSCI/CLKI/**RP34**/IOMF6/RC1
3	AV_{SS}	17	V_{SS}
4	AV_{DD}	18	V_{DD}
5	OA1OUT/AD1AN0/CMP1A/**RP3**/RA2	19	PGC3/**RP35**/PWM4H/RC2
6	OA1IN-/AD1ANN1/AD2AN0/**RP4**/RA3	20	PGD3/**RP36**/PWM3H/IOMD0/RC3
7	OA1IN+/AD1AN1/CMP1B/**RP5**/RA4	21	**RP37**/PWM3L/IOMD1/RC4
8	V_{SS}	22	**RP49**/PWM2H/IOMD2/RD0
9	V_{DD}	23	TCK/**RP50**/PWM2L/IOMD3/RD1
10	OA2OUT/AD2AN1/CMP2A/**RP17**/INT0/RB0	24	TDO/**RP51**/PWM1H/IOMD4/RD2
11	TMS/OA2IN-/AD1AN4/AD2ANN1/**RP18**/RB1(4)	25	TDI/**RP52**/PWM1L/IOMD5/RD3
12	OA2IN+/AD2AN4/CMP2B/**RP19**/RB2	26	V_{SS}
13	PGD1/AD1AN5/CMP1C/ISRC0/IBIAS0/**RP20**/SDA1/RB3	27	V_{DD}
14	PGC1/AD2AN5/CMP2C/ISRC1/IBIAS1/**RP21**/SCL1/RB4	28	$\overline{\text{MCLR}}$

Note:

1. **RPn** represents remappable peripheral functions.
2. This pin has 8x drive strength.
3. Unless otherwise stated, pins are 4x drive strength. Refer to Electrical Specifications for current drive strength details.
4. A pull-up resistor is connected to this pin when device is erased (JTAG enabled) and during programming.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Pin Diagrams

Figure 3. 36-Pin VQFN

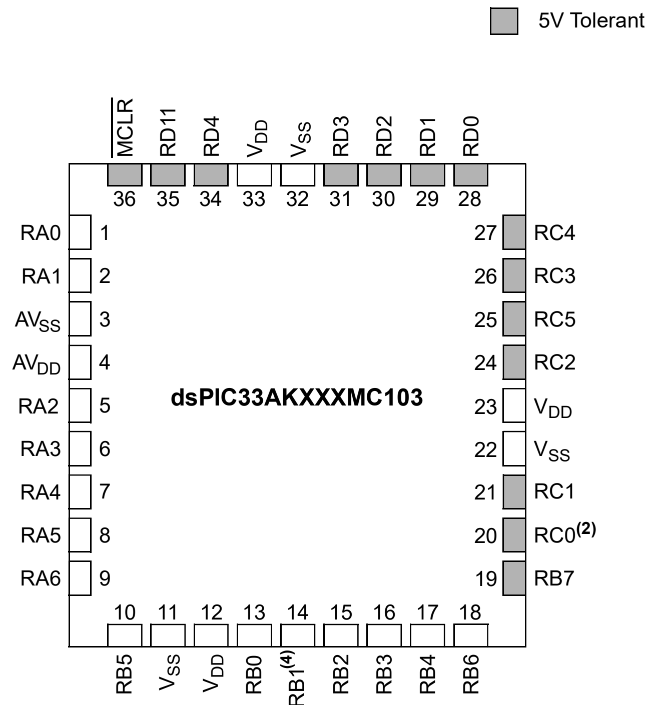


Table 4. 36-Pin VQFN Complete Pin Function Descriptions(1,3)

Pin	Function	Pin	Function
1	PGD2/AD2AN6/CMP3C/ISRC2/IBIAS2/**RP1**/SDA2/IOMF2/RA0	19	AD2ANN2/AD2AN8/**RP24**/IOMF0/RB7
2	PGC2/DACOUT1/AD1AN7/AD2AN3/CMP1D/CMP2D/ CMP3D/**RP2**/SCL2/RA1	20	OSCO/CLKO/**RP33**/IOMF5/RC0**(2)**
3	AV_{SS}	21	OSCI/CLKI/**RP34**/IOMF6/RC1
4	AV_{DD}	22	V_{SS}
5	OA1OUT/AD1AN0/CMP1A/**RP3**/RA2	23	V_{DD}
6	OA1IN-/AD1ANN1/AD2AN0/**RP4**/RA3	24	PGC3/**RP35**/PWM4H/RC2
7	OA1IN+/AD1AN1/CMP1B/**RP5**/RA4	25	**RP38**/PWM4L/RC5
8	OA3OUT/AD1AN3/CMP3A/**RP6**/RA5	26	PGD3/**RP36**/PWM3H/IOMD0/RC3
9	OA3IN-/AD1AN2/**RP7**/RA6	27	**RP37**/PWM3L/IOMD1/RC4
10	OA3IN+/AD2AN2/CMP3B/**RP22**/RB5	28	**RP49**/PWM2H/IOMD2/RD0
11	V_{SS}	29	TCK/**RP50**/PWM2L/IOMD3/RD1
12	V_{DD}	30	TDO/**RP51**/PWM1H/IOMD4/RD2
13	OA2OUT/AD2AN1/CMP2A/**RP17**/INT0/RB0	31	TDI/**RP52**/PWM1L/IOMD5/RD3
14	TMS/OA2IN-/AD1AN4/AD2ANN1/**RP18**/RB1**(4)**	32	V_{SS}
15	OA2IN+/AD2AN4/CMP2B/**RP19**/RB2	33	V_{DD}
16	PGD1/AD1AN5/CMP1C/ISRC0/IBIAS0/**RP20**/SDA1/RB3	34	**RP53**/PCI22/RD4
17	PGC1/AD2AN5/CMP2C/ISRC1/IBIAS1/**RP21**/SCL1/RB4	35	**RP60**/RD11
18	AD1ANN2/AD1AN8/**RP23**/RB6	36	$\overline{MCLR}$

Notes:

1. **RPn** represents remappable peripheral functions.
2. This pin has 8x drive strength.
3. Unless otherwise stated, pins are 4x drive strength. Refer to Electrical Specifications for current drive strength details.
4. A pull-up resistor is connected to this pin when device is erased (JTAG enabled) and during programming.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Pin Diagrams

Figure 4. 48-Pin VQFN, TQFP

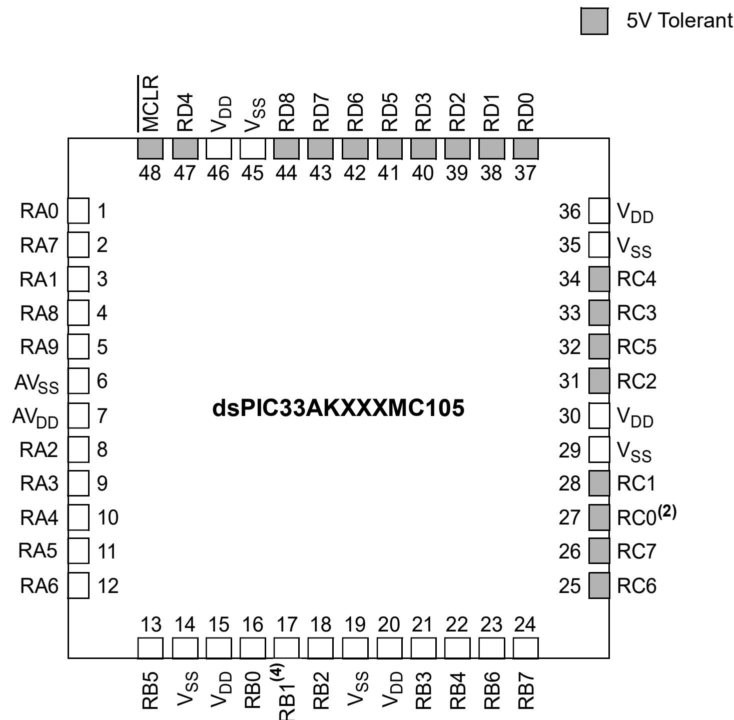


Table 5. 48-Pin VQFN, TQFP Complete Pin Function Descriptions(1,3)

Pin	Function	Pin	Function
1	PGD2/AD2AN6/CMP3C/ISRC2/IBIAS2/**RP1**/SDA2/IOMF2/RA0	25	**RP39**/RC6
2	AD1AN6/**RP8**/IOMF1/RA7	26	**RP40**/RC7
3	PGC2/DACOUT1/AD1AN7/AD2AN3/CMP1D/CMP2D/ CMP3D/**RP2**/SCL2/RA1	27	OSCO/CLKO/**RP33**/IOMF5/RC0(2)
4	AD2AN9/ISRC3/IBIAS3/**RP9**/RA8	28	OSCI/CLKI/**RP34**/IOMF6/RC1
5	AD1ANN3/AD1AN9/**RP10**/RA9	29	V_{SS}
6	AV_{SS}	30	V_{DD}
7	AV_{DD}	31	PGC3/**RP35**/PWM4H/RC2
8	OA1OUT/AD1AN0/CMP1A/**RP3**/RA2	32	**RP38**/PWM4L/RC5
9	OA1IN-/AD1ANN1/AD2AN0/**RP4**/RA3	33	PGD3/**RP36**/PWM3H/IOMD0/RC3
10	OA1IN+/AD1AN1/CMP1B/**RP5**/RA4	34	**RP37**/PWM3L/IOMD1/RC4
11	OA3OUT/AD1AN3/CMP3A/**RP6**/RA5	35	V_{SS}
12	OA3IN-/AD1AN2/**RP7**/RA6	36	V_{DD}
13	OA3IN+/AD2AN2/CMP3B/**RP22**/RB5	37	**RP49**/PWM2H/IOMD2/RD0
14	V_{SS}	38	TCK/**RP50**/PWM2L/IOMD3/RD1
15	V_{DD}	39	TDO/**RP51**/PWM1H/IOMD4/RD2
16	OA2OUT/AD2AN1/CMP2A/**RP17**/INT0/RB0	40	TDI/**RP52**/PWM1L/IOMD5/RD3
17	TMS/OA2IN-/AD1AN4/AD2ANN1/**RP18**/RB1(4)	41	**RP54**/ASCL1/RD5
18	OA2IN+/AD2AN4/CMP2B/**RP19**/RB2	42	**RP55**/ASDA1/RD6
19	V_{SS}	43	**RP56**/ASCL2/IOMD7/IOMF4/RD7
20	V_{DD}	44	**RP57**/ASDA2/IOMD6/IOMF3/RD8
21	PGD1/AN1P5/CMP1C/ISRC0/IBIAS0/**RP20**/SDA1/RB3	45	V_{SS}
22	PGC1/AD2AN5/CMP2C/ISRC1/IBIAS1/**RP21**/SCL1/RB4	46	V_{DD}
23	AD1ANN2/AD1AN8/**RP23**/RB6	47	**RP53**/PCI22/RD4
24	AD2ANN2/AD2AN8/**RP24**/IOMF0/RB7	48	$\overline{MCLR}$

© 2023-2024 Microchip Technology Inc. and its subsidiaries

...........continued

Pin	Function	Pin	Function

Note:

1. **RPn** represents remappable peripheral functions.
2. This pin has 8x drive strength.
3. Unless otherwise stated, pins are 4x drive strength. Refer to Electrical Specifications for current drive strength details.
4. A pull-up resistor is connected to this pin when device is erased (JTAG enabled) and during programming.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Pin Diagrams

Figure 5. 64-Pin VQFN, TQFP

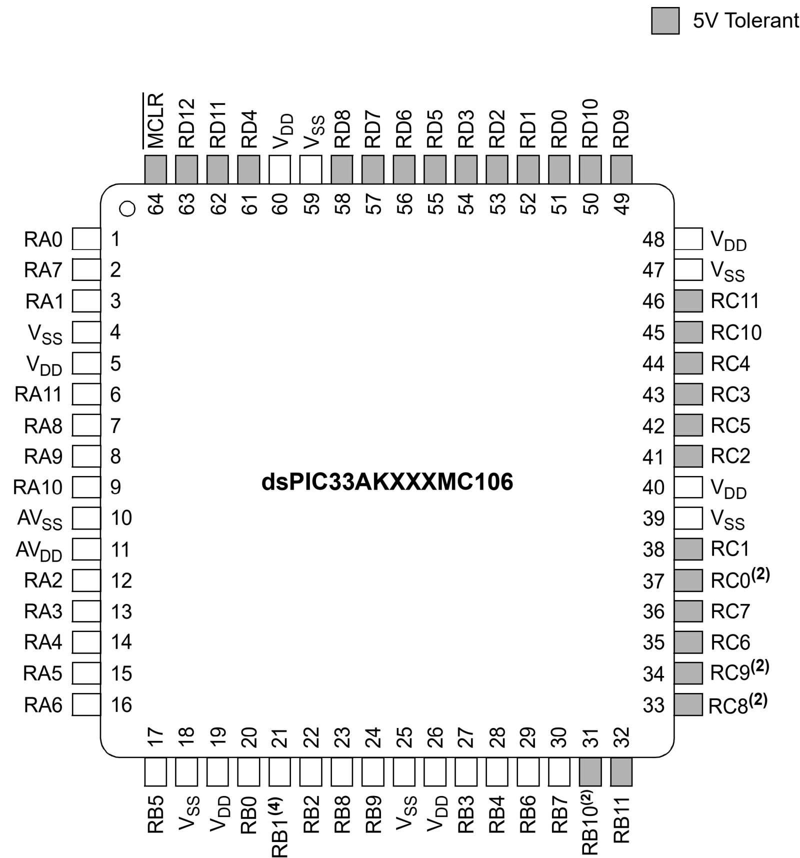


Table 6. 64-Pin VQFN, TQFP Complete Pin Function Descriptions(1,3)

Pin	Function	Pin	Function
1	PGD2/AD2AN6/CMP3C/ISRC2/IBIAS2/**RP1**/SDA2/IOMF2/RA0	33	**RP41**/IOMD11/IOMF11/PCI20/RC8(2)
2	AD1AN6/**RP8**/IOMF1/RA7	34	**RP42**/IOMD10/SDO2/IOMF10/PCI19/RC9(2)
3	PGC2/DACOUT1/AD1AN7/AD2AN3/CMP1D/CMP2D/ CMP3D/**RP2**/SCL2/RA1	35	**RP39**/RC6
4	V_{SS}	36	**RP40**/RC7
5	V_{DD}	37	OSCO/CLKO/**RP33**/IOMF5/RC0(2)
6	AD1AN10/**RP12**/RA11	38	OSCI/CLKI/**RP34**/IOMF6/RC1
7	AD2AN9/ISRC3/IBIAS3/**RP9**/RA8	39	V_{SS}
8	AD1ANN3/AD1AN9/**RP10**/RA9	40	V_{DD}
9	AD2ANN3/AD2AN7/**RP11**/RA10	41	PGC3/**RP35**/PWM4H/RC2
10	AV_{SS}	42	**RP38**/PWM4L/RC5
11	AV_{DD}	43	PGD3/**RP36**/PWM3H/IOMD0/RC3
12	OA1OUT/AD1AN0/CMP1A/**RP3**/RA2	44	**RP37**/PWM3L/IOMD1/RC4
13	OA1IN-/AD1ANN1/AD2AN0/**RP4**/RA3	45	**RP43**/IOMD9/IOMF9/RC10
14	OA1IN+/AD1AN1/CMP1B/**RP5**/RA4	46	**RP44**/IOMD8/IOMF8/RC11
15	OA3OUT/AD1AN3/CMP3A/**RP6**/RA5	47	V_{SS}
16	OA3IN-/AD1AN2/**RP7**/RA6	48	V_{DD}
17	OA3IN+/AD2AN2/CMP3B/**RP22**/RB5	49	**RP58**/IOMF7/RD9
18	V_{SS}	50	**RP59**/RD10
19	V_{DD}	51	**RP49**/PWM2H/IOMD2/RD0
20	OA2OUT/AD2AN1/CMP2A/**RP17**/INT0/RB0	52	TCK/**RP50**/PWM2L/IOMD3/RD1

© 2023-2024 Microchip Technology Inc. and its subsidiaries

...........continued

Pin	Function	Pin	Function
21	TMS/OA2IN-/AD1AN4/AD2ANN1/**RP18**/RB1(4)	53	TDO/**RP51**/PWM1H/IOMD4/RD2
22	OA2IN+/AD2AN4/CMP2B/**RP19**/RB2	54	TDI/**RP52**/PWM1L/IOMD5/RD3
23	AD1AN11/**RP25**/RB8	55	**RP54**/ASCL1/RD5
24	AD2AN10/**RP26**/RB9	56	**RP55**/ASDA1/RD6
25	V_{SS}	57	**RP56**/ASCL2/IOMD7/IOMF4/RD7
26	V_{DD}	58	**RP57**/ASDA2/IOMD6/IOMF3/RD8
27	PGD1/AD1AN5/CMP1C/ISRC0/IBIAS0/**RP20**/SDA1/RB3	59	V_{SS}
28	PGC1/AD2AN5/CMP2C/ISRC1/IBIAS1/**RP21**/SCL1/RB4	60	V_{DD}
29	AD1ANN2/AD1AN8/**RP23**/RB6	61	**RP53**/PCI22/RD4
30	AD2ANN2/AD2AN8/**RP24**/IOMF0/RB7	62	**RP60**/RD11
31	**RP27**/SCK2/RB10(2)	63	**RP61**/PCI21/RD12
32	**RP28**/SDI2/RB11	64	$\overline{MCLR}$

Note:

1. **RPn** represents remappable peripheral functions.
2. This pin has 8x drive strength.
3. Unless otherwise stated, pins are 4x drive strength. Refer to Electrical Specifications for current drive strength details.
4. A pull-up resistor is connected to this pin when device is erased (JTAG enabled) and during programming.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Pinout I/O Descriptions

Table 7. Pinout I/O Descriptions

Pin Name[1]	Pin Type	Buffer Type	PPS	Description
AN1P0 - AN1P11 AN1N1 - AN1N3 AN2P0 - AN2P10 AN2N1 - AN2N3	I I I I	Analog Analog Analog Analog	No No No No	ADC1 positive input channels ADC1 negative input channels ADC2 positive input channels ADC2 negative input channels
ADTRG31	I	ST	Yes	ADC Trigger Input 31
CLKI CLKO	I O	ST/CMOS —	No No	External Clock (EC) source input. Always associated with OSCI pin function. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. Optionally functions as CLKO in RC and EC modes. Always associated with OSCO pin function.
OSCI OSCO	I I/O	ST/CMOS —	No No	Oscillator crystal input. ST buffer when configured in RC mode; CMOS otherwise. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. Optionally functions as CLKO in RC and EC modes.
REFCLKI	I	ST	Yes	Reference clock input
REFCLKO	O	—	Yes	Reference clock output
INT0 INT1 INT2 INT3 INT4	I I I I I	ST ST ST ST ST	No Yes Yes Yes Yes	External Interrupt 0 External Interrupt 1 External Interrupt 2 External Interrupt 3 External Interrupt 4
IOCA[4:0] IOCB[15:0] IOCC[15:0] IOCD[15:0] IOCE[15:0] IOCF[15:0]	I I I I I I	ST ST ST ST ST ST	No No No No No No	Interrupt-on-Change input for PORTA Interrupt-on-Change input for PORTB Interrupt-on-Change input for PORTC Interrupt-on-Change input for PORTD Interrupt-on-Change input for PORTE Interrupt-on-Change input for PORTF
IOMD[n:0] IOMF[n:0]	O I	ST ST	Yes Yes	I/O Monitor Reference I/O Monitor Feedback
QEIA1 QEIB1 QEINDX1 QEIHOM1 QEICMP	I I I I O	ST ST ST ST —	Yes Yes Yes Yes Yes	QEI Input A1 QEI Input B1 QEI Index 1 input QEI Home 1 input QEI comparator output
RA0-RA4	I/O	ST	No	PORTA is a bidirectional I/O port

Legend: CMOS = CMOS compatible input or output; TTL = TTL input buffer; Analog = Analog input; P = Power; ST = Schmitt Trigger input with CMOS levels; O = Output; I = Input;
PPS = Peripheral Pin Select

Notes:

1. Not all pins are available in all package variants. See the Pin Diagrams section for pin availability.
2. These pins are remappable as well as dedicated.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

...........continued

Pin Name[1]	Pin Type	Buffer Type	PPS	Description
RB0-RB15	I/O	ST	No	PORTB is a bidirectional I/O port
RC0-RC15	I/O	ST	No	PORTC is a bidirectional I/O port
RD0-RD15	I/O	ST	No	PORTD is a bidirectional I/O port
RE0-RE15	I/O	ST	No	PORTE is a bidirectional I/O port
RF0-RF15	I/O	ST	No	PORTF is a bidirectional I/O port
T1CK	I	ST	Yes	Timer1 external clock input
$\overline{\text{U1CTS}}$	I	ST	Yes	UART1 Clear-to-Send
$\overline{\text{U1RTS}}$	O	—	Yes	UART1 Request-to-Send
U1RX	I	ST	Yes	UART1 receive
U1TX	O	—	Yes	UART1 transmit
$\overline{\text{U1DSR}}$	I	ST	Yes	UART1 Data-Set-Ready
U1DTR	O	—	Yes	UART1 Data-Terminal-Ready
$\overline{\text{U2CTS}}$	I	ST	Yes	UART2 Clear-to-Send
$\overline{\text{U2RTS}}$	O	—	Yes	UART2 Request-to-Send
U2RX	I	ST	Yes	UART2 receive
U2TX	O	—	Yes	UART2 transmit
$\overline{\text{U2DSR}}$	I	ST	Yes	UART2 Data-Set-Ready
U2DTR	O	—	Yes	UART2 Data-Terminal-Ready
$\overline{\text{U3CTS}}$	I	ST	Yes	UART3 Clear-to-Send
$\overline{\text{U3RTS}}$	O	—	Yes	UART3 Request-to-Send
U3RX	I	ST	Yes	UART3 receive
U3TX	O	—	Yes	UART3 transmit
$\overline{\text{U3DSR}}$	I	ST	Yes	UART3 Data-Set-Ready
U3DTR	O	—	Yes	UART3 Data-Terminal-Ready
SENT1	I	ST	Yes	SENT1 input
SENT2	I	ST	Yes	SENT2 input
SENT1OUT	O	—	Yes	SENT1 output
SENT2OUT	O	—	Yes	SENT2 output
PTGTRG24	O	—	Yes	PTG Trigger Output 24
PTGTRG25	O	—	Yes	PTG Trigger Output 25
TCKI1-TCKI9	I	ST	Yes	SCCP Timer Inputs 1 through 9
ICM1-ICM9	I	ST	Yes	SCCP Capture Inputs 1 through 9
OCFA-OCFD	I	ST	Yes	SCCP Fault Inputs A through D
OCM1-OCM9	O	—	Yes	SCCP Compare Outputs 1 through 9
SCK1	I/O	ST	Yes	Synchronous serial clock I/O for SPI1
SDI1	I	ST	Yes	SPI1 data in
SDO1	O	—	Yes	SPI1 data out
$\overline{\text{SS1}}$	I/O	ST	Yes	SPI1 Client synchronization or frame pulse I/O

Legend: CMOS = CMOS compatible input or output; TTL = TTL input buffer; Analog = Analog input; P = Power; ST = Schmitt Trigger input with CMOS levels; O = Output; I = Input;
PPS = Peripheral Pin Select

Notes:

1. Not all pins are available in all package variants. See the Pin Diagrams section for pin availability.
2. These pins are remappable as well as dedicated.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

...........continued

Pin Name[1]	Pin Type	Buffer Type	PPS	Description
SCK2	I/O	ST	Yes	Synchronous serial clock I/O for SPI2
SDI2	I	ST	Yes	SPI2 data in
SDO2	O	—	Yes	SPI2 data out
$\overline{SS2}$	I/O	ST	Yes	SPI2 Client synchronization or frame pulse I/O
SCK3	I/O	ST	Yes	Synchronous serial clock I/O for SPI3
SDI3	I	ST	Yes	SPI3 data in
SDO3	O	—	Yes	SPI3 data out
$\overline{SS3}$	I/O	ST	Yes	SPI3 Client synchronization or frame pulse I/O
SCL1	I/O	ST	No	Synchronous serial clock I/O for I2C1
SDA1	I/O	ST	No	Synchronous serial data I/O for I2C1
ASCL1	I/O	ST	No	Alternate synchronous serial clock I/O for I2C1
ASDA1	I/O	ST	No	Alternate synchronous serial data I/O for I2C1
SCL2	I/O	ST	No	Synchronous serial clock I/O for I2C2
SDA2	I/O	ST	No	Synchronous serial data I/O for I2C2
ASCL2	I/O	ST	No	Alternate synchronous serial clock I/O for I2C2
ASDA2	I/O	ST	No	Alternate synchronous serial data I/O for I2C2
BISS1SL	I	ST	Yes	BiSS1 Return Input
BISS1GS	I	ST	Yes	BiSS1 Get Sense
BISS1MO	O	ST	Yes	BiSS1 Output
BISS1MA	O	ST	Yes	BiSS1 Clock
TMS	I	ST	No	JTAG Test mode select pin
TCK	I	ST	No	JTAG test clock input pin
TDI	I	ST	No	JTAG test data input pin
TDO	O	—	No	JTAG test data output pin
PCI8-PCI18	I	ST	Yes	PWM Inputs 8 through 18
PCI19-PCI22	I	ST	No	PWM Inputs 19 through 22
PWMEA-PWMEF	O	—	Yes	PWM Event Outputs A through F
PWM1L-PWM4L[2]	O	—	Yes	PWM Low Outputs 1 through 4
PWM1H-PWM4H[2]	O	—	Yes	PWM High Outputs 1 through 4
CLCINA-CLCIND	I	ST	Yes	CLC Inputs A through D
CLC1OUT-CLC8OUT	O	—	Yes	CLC Outputs 1 through 8
CMP1	O	—	Yes	Comparator 1 output
CMP1A-CMP3A	I	Analog	No	Comparator Channels 1A through 3A inputs
CMP1B-CMP3B	I	Analog	No	Comparator Channels 1B through 3B inputs
CMP1C-CMP3C	I	Analog		Comparator Channels 1C through 3C inputs
CMP1D-CMP3D	I	Analog	No	Comparator Channels 1D through 3D inputs
			No	
DACOUT1	O	—	No	DAC1 output voltage

Legend: CMOS = CMOS compatible input or output; TTL = TTL input buffer; Analog = Analog input; P = Power; ST = Schmitt Trigger input with CMOS levels; O = Output; I = Input;
PPS = Peripheral Pin Select

Notes:

1. Not all pins are available in all package variants. See the Pin Diagrams section for pin availability.
2. These pins are remappable as well as dedicated.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

...........continued

Pin Name[1]	Pin Type	Buffer Type	PPS	Description
IBIAS3, IBIAS2, IBIAS1, IBIAS0/ISRC3, ISRC2, ISRC1, ISRC0	O	Analog	No	Constant-Current Outputs 0 through 3
OA1IN+	I	—	No	Op Amp 1+ input
OA1IN-	I	—	No	Op Amp 1- input
OA1OUT	O	—	No	Op Amp 1 output
OA2IN+	I	—	No	Op Amp 2+ input
OA2IN-	I	—	No	Op Amp 2- input
OA2OUT	O	—	No	Op Amp 2 output
OA3IN+	I	—	No	Op Amp 3+ input
OA3IN-	I	—	No	Op Amp 3- input
OA3OUT	O	—	No	Op Amp 3 output
PGD1	I/O	ST	No	Data I/O pin for Programming/ Debugging Communication Channel 1
PGC1	I	ST	No	Clock input pin for Programming/ Debugging Communication Channel 1
PGD2	I/O	ST	No	Data I/O pin for Programming/ Debugging Communication Channel 2
PGC2	I	ST	No	Clock input pin for Programming/ Debugging Communication Channel 2
PGD3	I/O	ST	No	Data I/O pin for Programming/ Debugging Communication Channel 3
PGC3	I	ST	No	Clock input pin for Programming/ Debugging Communication Channel 3
$\overline{MCLR}$	I/P	ST	No	Master Clear (Reset) input. This pin is an active-low Reset to the device.
AV_{DD}	P	P	No	Positive supply for analog modules. This pin must be connected at all times.
AV_{SS}	P	P	No	Ground reference for analog modules. This pin must be connected at all times.
V_{DD}	P	—	No	Positive supply for peripheral logic and I/O pins
V_{SS}	P	—	No	Ground reference for logic and I/O pins

Legend: CMOS = CMOS compatible input or output; TTL = TTL input buffer; Analog = Analog input; P = Power; ST = Schmitt Trigger input with CMOS levels; O = Output; I = Input;
PPS = Peripheral Pin Select

Notes:

1. Not all pins are available in all package variants. See the Pin Diagrams section for pin availability.
2. These pins are remappable as well as dedicated.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

To Our Valued Customers

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via E-mail at docerrors@microchip.com. We welcome your feedback.

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Website at:

www.microchip.com/

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., DS30000000A is version A of document DS30000000).

Errata

An errata sheet, describing minor operational differences from the data sheet and recommended workarounds, may exist for current devices. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Website; www.microchip.com/
- Your local Microchip sales office (see last page)

When contacting a sales office, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Customer Notification System

Register on our website at www.microchip.com/ to receive the most current information on all of our products.

Terminology Cross Reference

Table 8 provides updated terminology for deprecated naming conventions. Register and bit names remain unchanged, however, descriptions and usage guidance may have been updated.

Table 8. Terminology Cross References

Use Case	Deprecated Term	New Term
CPU	Master	Initiator
DMA	Master	Initiator
I^2C	Master	Host
	Slave	Client
SPI	Master	Host
	Slave	Client
UART, LIN Mode	Master	Commander
	Slave	Responder
PWM	Master	Host
	Slave	Client

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Table of Contents

dsPIC33AK128MC106 Product Family.......7

Pin Diagrams.......9

Pinout I/O Descriptions.......16

Terminology Cross Reference.......20

1. Device Overview.......27

2. Guidelines for Getting Started with Digital Signal Controllers.......28
   - 2.1. Basic Connection Requirements.......28
   - 2.2. Decoupling Capacitors.......28
   - 2.3. Master Clear ($\overline{MCLR}$) Pin.......29
   - 2.4. ICSP Pins.......30
   - 2.5. External Oscillator Pins.......30
   - 2.6. External Oscillator Layout Guidance.......30
   - 2.7. Oscillator Value Conditions on Device Start-up.......31
   - 2.8. Unused I/Os.......31
   - 2.9. Bulk Capacitors.......31

3. CPU.......32
   - 3.1. Architectural Overview.......32
   - 3.2. CPU Register Descriptions.......34
   - 3.3. CPU Operation.......56
   - 3.4. Prefetch Buffer Unit (PBU).......82
   - 3.5. Performance Monitor Unit (PMU).......93
   - 3.6. Floating-Point Unit (FPU) Coprocessor.......103

4. Memory Organization.......160
   - 4.1. Device-Specific Information.......160
   - 4.2. Architectural Overview.......163
   - 4.3. Register Summary.......166
   - 4.4. BMX Operation.......181

5. Data Memory.......186
   - 5.1. Device-Specific Information.......186
   - 5.2. Architectural Overview.......186
   - 5.3. Register Summary.......188
   - 5.4. Operation.......208

6. Flash Program Memory.......215
   - 6.1. Device-Specific Information.......215
   - 6.2. Register Summary.......217
   - 6.3. Operation.......245
   - 6.4. Application Example.......250

7. Configuration Bits.......251
   - 7.1. Configuration Register Summary.......253

© 2023-2024 Microchip Technology Inc. and its subsidiaries

7.2. Device Calibration and Identification....270

8. Security Module....273
   8.1. Architectural Overview....273
   8.2. Security Module Register Summary....276
   8.3. Flash Memory Map....287
   8.4. Device Locking....290
   8.5. Flash Protection Regions....294
   8.6. Peripheral Access Controller (PAC)....296

9. Resets....304
   9.1. Architectural Overview....304
   9.2. Register Summary....305
   9.3. Operation....307
   9.4. Application Examples....309
   9.5. Effects of Reset....310

10. Interrupt Controller....312
   10.1. Device-Specific Information....312
   10.2. Architectural Overview....317
   10.3. Interrupt Vector Table....318
   10.4. Register Summary....321
   10.5. Operation....460
   10.6. Interrupt Control and Status Registers....461
   10.7. Priority....462
   10.8. Interrupt Sequence....463
   10.9. Non-Maskable Traps....465
   10.10. Interrupt Operations....467

11. I/O Ports with Edge Detect....471
   11.1. Device-Specific Information....471
   11.2. Architectural Overview....478
   11.3. Register Summary....482
   11.4. Operation....543
   11.5. Applications....553
   11.6. Interrupts....554
   11.7. Operation in Power-Saving Modes....554
   11.8. Effects of Various Resets....555

12. Oscillator and Clocking Module....556
   12.1. Device-Specific Information....556
   12.2. Architectural Overview....557
   12.3. Register Summary....560
   12.4. Operation....604

13. Direct Memory Access (DMA) Controller....629
   13.1. Device-Specific Information....629
   13.2. Architectural Overview....631
   13.3. DMA Register Summary....633
   13.4. Operation....655

© 2023-2024 Microchip Technology Inc. and its subsidiaries

13.5. Application Examples....677
13.6. DMA Interrupts....679
13.7. Operation During Sleep and Idle Modes....682

14. PWM....683
14.1. Device-Specific Information....683
14.2. Architectural Overview....685
14.3. Register Summary....689
14.4. Operation....751
14.5. Application Examples....799
14.6. Interrupts....817
14.7. Operation in Power-Saving Modes....818

15. High-Speed, 12-Bit Low Latency ADC....819
15.1. Device-Specific Information....819
15.2. Architectural Overview....823
15.3. Register Summary....825
15.4. ADC Operation....864
15.5. Application Examples....872
15.6. Effects of Reset....879

16. High-Speed Analog Comparator with Slope Compensation DAC....880
16.1. Device-Specific Information....880
16.2. Architectural Overview....882
16.3. DAC Register Summary....884
16.4. Operation....894
16.5. Application Examples....900

17. Quadrature Encoder Interface (QEI)....906
17.1. Device-Specific Information....906
17.2. Architectural Overview....906
17.3. QEI Register Summary....910
17.4. Operation....928
17.5. Application Example....936
17.6. Interrupts....937
17.7. QEI Operation in Power-Saving Modes....937

18. Universal Asynchronous Receiver Transmitter (UART)....938
18.1. Device-Specific Information....938
18.2. Architectural Overview....939
18.3. UART Register Summary....941
18.4. Operation....962
18.5. Application Examples....990
18.6. Interrupts....992
18.7. Power-Saving Modes....993

19. Serial Peripheral Interface (SPI)....994
19.1. Device-Specific Information....994
19.2. Architectural Overview....994
19.3. Register Summary....1000

© 2023-2024 Microchip Technology Inc. and its subsidiaries

19.4. Operation........ 1014
19.5. Interrupts........ 1048
19.6. Operation in Power-Saving and Debug Modes........ 1049

20. Inter-Integrated Circuit (I^2C)........ 1051
20.1. Device-Specific Information........ 1051
20.2. Architectural Overview........ 1051
20.3. I2C System Overview........ 1054
20.4. Register Summary........ 1056
20.5. Operation........ 1085
20.6. Application Examples........ 1133
20.7. Interrupts........ 1142
20.8. Operation in Power-Saving Modes........ 1144

21. Single-Edge Nibble Transmission (SENT)........ 1145
21.1. Device-Specific Information........ 1145
21.2. Architectural Overview........ 1145
21.3. Register Summary........ 1148
21.4. Operation........ 1156
21.5. Application Examples........ 1167
21.6. Interrupts........ 1170
21.7. Operation in Power-Saving Modes........ 1170
21.8. Effects of a Reset........ 1170

22. Bidirectional Serial Synchronous (BiSS) Module........ 1171
22.1. Device-Specific Information........ 1171
22.2. Architectural Overview........ 1171
22.3. Register Summary........ 1178
22.4. Operations........ 1202
22.5. Application Examples........ 1210
22.6. Interrupts........ 1214
22.7. Power Saving Modes........ 1214
22.8. Terminology........ 1215

23. Timer1........ 1216
23.1. Device-Specific Information........ 1216
23.2. Architectural Overview........ 1216
23.3. Register Summary........ 1218
23.4. Operation........ 1222
23.5. Interrupts........ 1232
23.6. Operation in Power-Saving Modes........ 1233
23.7. Effects of Various Resets........ 1233

24. Single-Output Capture/Compare/PWM/Timer Modules (SCCP)........ 1235
24.1. Device-Specific Information........ 1235
24.2. Architectural Overview........ 1235
24.3. Register Summary........ 1238
24.4. Operation........ 1254
24.5. Operation During Sleep and Idle Modes........ 1291
24.6. Effects of a Reset........ 1292

© 2023-2024 Microchip Technology Inc. and its subsidiaries

25. Configurable Logic Cell (CLC)........1293
 25.1. Device-Specific Information........1293
 25.2. Architecture........1294
 25.3. CLC Control Registers........1298
 25.4. Operation........1305
 25.5. CLC Application Example........1309
 25.6. CLC Interrupts........1310
 25.7. Operation in Power-Saving Modes........1311

26. Peripheral Trigger Generator (PTG)........1312
 26.1. Device-Specific Information........1312
 26.2. Architectural Overview........1313
 26.3. PTG Register Summary........1321
 26.4. Operation........1336
 26.5. Application Examples........1349
 26.6. Interrupts........1360
 26.7. Power-Saving Modes........1360

27. 32-Bit Programmable Cyclic Redundancy Check (CRC) Generator........1362
 27.1. Architectural Overview........1362
 27.2. Register Summary........1364
 27.3. CRC Operation........1369
 27.4. Application Examples........1376
 27.5. CRC Operation in Power Saving Modes........1380

28. Current Bias Generator (CBG)........1381
 28.1. Device-Specific Information........1381
 28.2. Architectural Overview........1381
 28.3. Current Bias Generator Control Register........1383
 28.4. Operation........1384
 28.5. Application Examples........1386
 28.6. Interrupts........1388
 28.7. Operating in Power-Saving Modes........1388
 28.8. Effects of a Reset........1388

29. Operational Amplifier........1389
 29.1. Device-Specific Information........1389
 29.2. Architectural Overview........1389
 29.3. Op Amp Register Summary........1390
 29.4. Operations........1394
 29.5. Op Amp Application Examples........1396

30. Watchdog Timer (WDT)........1397
 30.1. Device-Specific Information........1397
 30.2. Architectural Overview........1397
 30.3. Register Summary........1399
 30.4. Watchdog Timer Operation........1401
 30.5. Watchdog Timer Reset........1404
 30.6. Operation of Watchdog Timer in Sleep/Idle Modes........1404
 30.7. WDT Generic Trap........1405

© 2023-2024 Microchip Technology Inc. and its subsidiaries

30.8. WDT Sample Configuration.....1405

31. Deadman Timer (DMT).....1408
    31.1. Architectural Overview.....1408
    31.2. Deadman Timer Register Summary.....1410
    31.3. DMT Operation.....1419

32. Power-Saving Modes.....1423
    32.1. Architectural Overview.....1423
    32.2. Power-Saving Control Register Summary.....1424
    32.3. Power-Saving Operations.....1433

33. JTAG Interface.....1438

34. In-Circuit Debugger.....1439

35. Instruction Set Summary.....1440

36. Development Support.....1451

37. Electrical Characteristics.....1452
    37.1. DC Characteristics.....1452
    37.2. AC Characteristics and Timing Parameters.....1463

38. Packaging Information.....1489
    38.1. Package Marking Information.....1489
    38.2. Package Details.....1491

39. Revision History.....1512

Microchip Information.....1518
    The Microchip Website.....1518
    Product Change Notification Service.....1518
    Customer Support.....1518
    Product Identification System.....1519
    Microchip Devices Code Protection Feature.....1520
    Legal Notice.....1520
    Trademarks.....1520
    Quality Management System.....1521
    Worldwide Sales and Service.....1522

© 2023-2024 Microchip Technology Inc. and its subsidiaries

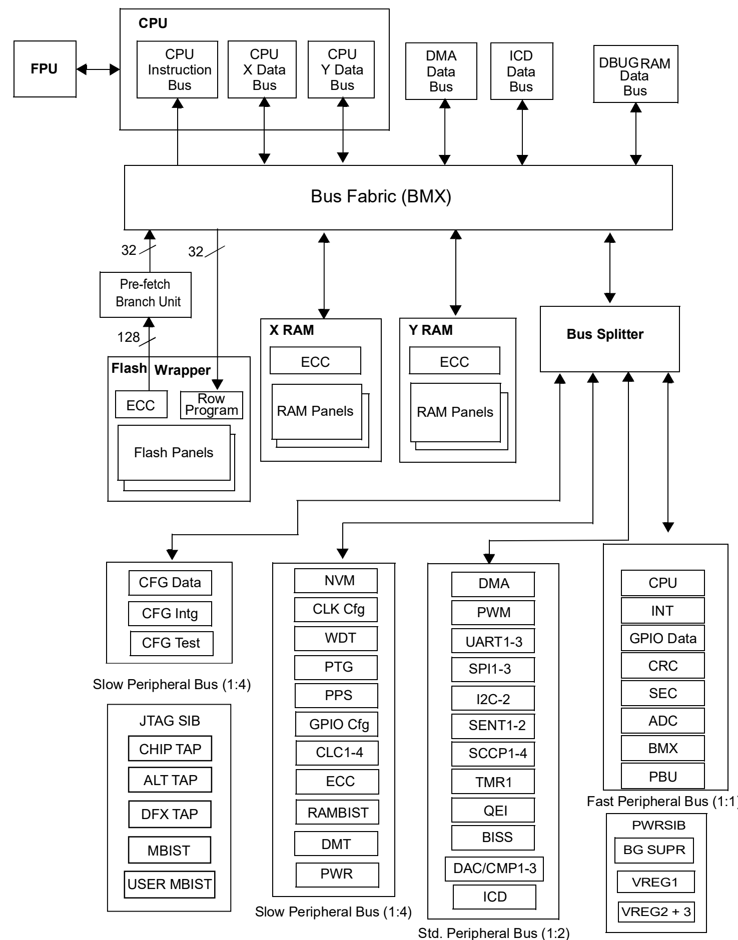
1. Device Overview

This document contains device-specific information for the dsPIC33AK128MC106 Digital Signal Controller (DSC) family of devices.

The dsPIC33AK128MC106 devices support a high-performance architecture with the Digital Signal Processor (DSP) and a Single and Double Precision Floating Point Unit (FPU). The dsPIC33AK128MC106 family of devices operate with an internal core supplied with a 1.1V using a low-voltage regulator.

Figure 1-1 shows a general block diagram of the core and peripheral modules of the dsPIC33AK128MC106 family.

Figure 1-1. dsPIC33AK128MC106 Family Block Diagram



Notes:

1. Not all I/O pins or features are implemented on all device pinout configurations. See Pinout I/O Descriptions for specific implementations by pin count.
2. Some peripheral I/Os are only accessible through Peripheral Pin Select (PPS).

© 2023-2024 Microchip Technology Inc. and its subsidiaries

2. Guidelines for Getting Started with Digital Signal Controllers

2.1 Basic Connection Requirements

Getting started with the dsPIC33AK128MC106 family devices requires attention to a minimal set of device pin connections before proceeding with development. The following pins must always be connected:

- All V_{DD} and V_{SS} power supply pins must be properly biased with required voltages (see 37. Electrical Characteristics)
- All AV_{DD} and AV_{SS} analog supply pins must be properly biased regardless of which analog modules or components of the dsPIC33A device are used (see 37. Electrical Characteristics)
- $\overline{\text{MCLR}}$ pin is connected with V_{DD} and V_{SS} based on circuit or application needs
- PGCx/PGDx pins used for In-Circuit Serial Programming™ (ICSP™) and debugging purposes (see 2.4. ICSP Pins)
- OSCI and OSCO pins when an external oscillator source is used (see 2.5. External Oscillator Pins)

2.2 Decoupling Capacitors

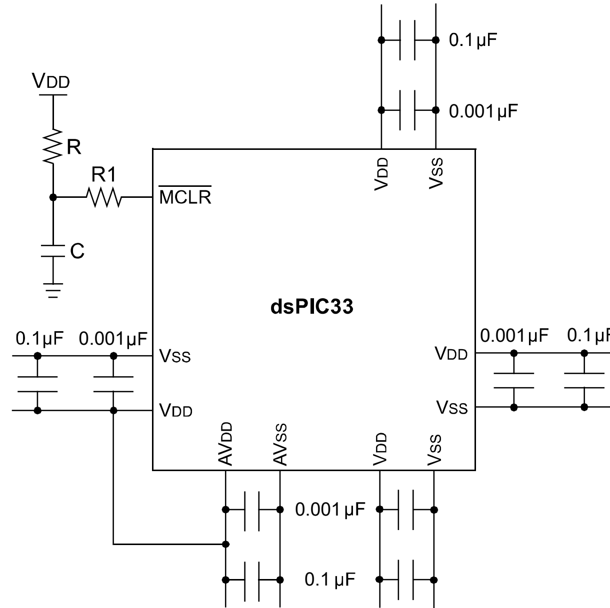
The use of decoupling capacitors on every pair of power supply pins, such as V_{DD}, V_{SS}, AV_{DD} and AV_{SS}, is required.

Consider the following criteria when using decoupling capacitors:

- **Value and type of capacitor:** Recommendation of 0.1 µF (100 nF), in parallel with a 1000 pF (1 nF), 10-20V. These capacitors should be low-ESR and have a resonance frequency in the range of 20 MHz and higher. Ceramic capacitors are recommended.
- **Placement on the Printed Circuit Board:** The decoupling capacitors should be placed as close to the pins as possible. It is recommended to place the capacitors on the same side of the board as the device. If space is constricted, the capacitor can be placed on another layer on the PCB using a via; however, ensure that the trace length from the pin to the capacitor is within one-quarter inch (6 mm) in length.
- **Handling high-frequency noise:** If the board is experiencing high-frequency noise above tens of MHz, add an additional ceramic-type capacitor in parallel to the decoupling capacitors. The value can be in the range of 0.01 µF to 0.001 µF. Place this capacitor next to the primary decoupling capacitors. In high-speed circuit designs, consider implementing a decade set of capacitances as close to the power and ground pins as possible. For example, 0.1 µF in parallel with 0.01 µF and 0.001 µF.
- **Maximizing performance:** On the board layout from the power supply circuit, run the power and return traces to the decoupling capacitors first and then to the device pins. This ensures that the decoupling capacitors are first in the power chain. Equally important is to keep the trace length between the capacitor and the power pins to a minimum, thereby reducing PCB track inductance.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 2-1. Recommended Minimum Connection



2.3 Master Clear (MCLR) Pin

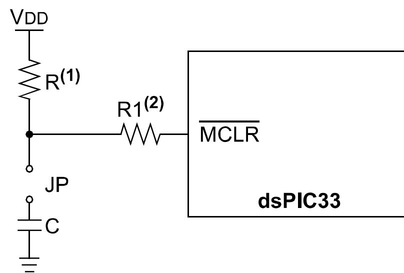
The MCLR pin provides two specific device functions:

- Device Reset
- Device Programming and Debugging

During device programming and debugging, the resistance and capacitance that can be added to the pin must be considered. Device programmers and debuggers drive the MCLR pin. Consequently, specific voltage levels (V_{IH} and V_{IL}) and fast signal transitions must not be adversely affected. Ensure that the MCLR pin V_{IH} and V_{IL} voltage specifications are met.

For example, Figure 2-2 shows the MCLR pin connections with general circuit components used, such as resistor R, series resistor R1 and capacitor C, and their placement. It is recommended to place these passive components with one-quarter inch (6mm) from the MCLR pin.

Figure 2-2. Example of MCLR Pin Connections



© 2023-2024 Microchip Technology Inc. and its subsidiaries

Notes:

1. R ≤ 10 kΩ is recommended. A suggested starting value is 10 kΩ. Ensure that the $\overline{MCLR}$ pin V_{IH} and V_{IL} specifications are met.
2. R1 ≤ 470Ω will limit any current flowing into $\overline{MCLR}$ from the external capacitor, C, in the event of $\overline{MCLR}$ pin breakdown due to Electrostatic Discharge (ESD) or Electrical Overstress (EOS). Ensure that the $\overline{MCLR}$ pin V_{IH} and V_{IL} specifications are met.
3. C ≤ 1 µF may be recommended. However, values of C should be based on reset timings required for any application. Make sure to isolate C from the $\overline{MCLR}$ pin during programming and debugging operations.

2.4 ICSP Pins

The PGCx and PGDx pins are used for programming and debugging purposes. It is recommended to keep the trace length between the ICSP connector and the ICSP pins on the device as short as possible. If the ICSP connector is expected to experience an ESD event, a series resistor is recommended, with the value in the range of a few tens of Ohms, not to exceed 100 Ohms.

Pull-up resistors, series diodes and capacitors on the PGCx and PGDx pins are not recommended as they will interfere with the programmer/debugger communications to the device. If such discrete components are an application requirement, they should be removed from the circuit during programming and debugging. Alternatively, refer to the AC/DC characteristics and timing requirements information in the respective device Flash programming specification for information on capacitive loading limits and pin Voltage Input High (V_{IH}) and Voltage Input Low (V_{IL}) requirements.

2.5 External Oscillator Pins

When the Primary Oscillator (POSC) circuit is used to connect a crystal oscillator, special care and consideration is needed to ensure proper operation. The POSC circuit should be tested across the environmental conditions in which the end product is intended to be used. The load capacitors specified in the crystal oscillator data sheet can be used as a starting point, however, the parasitic capacitance from the PCB traces can affect the circuit, and the values may need to be altered to ensure proper start-up and operation. Excessive trace length and other physical interaction can lead to poor signal quality. Poorly tuned oscillator circuits can have reduced amplitude, incorrect frequency (runt pulses), distorted waveforms and long start-up times that may result in unpredictable application behavior, such as instruction misexecution, illegal opcode fetch, etc. Ensure that the crystal oscillator circuit is at full amplitude and the correct frequency before the system begins to execute code. In planning the application's routing and I/O assignments, ensure that adjacent port pins, and other signals in close proximity to the oscillator, do not have high frequencies, short rise and fall times, and other similar noise. For further information on the Primary Oscillator, see 12.4.3. Primary Oscillator (POSC).

2.6 External Oscillator Layout Guidance

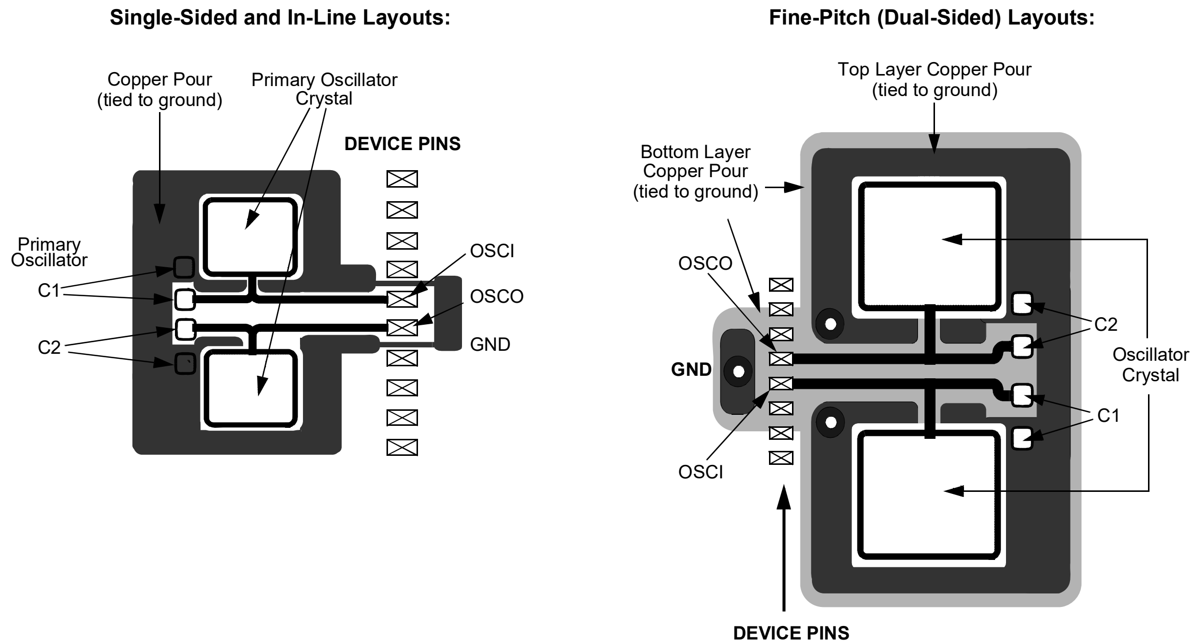
Use best practices during PCB layout to ensure robust start-up and operation. The oscillator circuit should be placed on the same side of the board as the device. Also, place the oscillator circuit close to the respective oscillator pins, not exceeding one-half inch (12 mm) distance between them. The load capacitors should be placed next to the oscillator itself, on the same side of the board. Use a grounded copper pour around the oscillator circuit to isolate them from surrounding circuits. The grounded copper pour should be routed directly to the MCU ground. Do not run any signal traces or power traces inside the ground pour. If using a two-sided board, avoid any traces on the other side of the board where the crystal is placed. Suggested layouts are shown in Figure 2-3. With fine-pitch packages, it is not always possible to completely surround the pins and components. A suitable solution is to tie the broken guard sections to a mirrored ground layer. In all cases, the guard trace(s) must be returned to ground.

For additional information and design guidance on oscillator circuits, please refer to these Microchip Application Notes, available at the Microchip website (www.microchip.com):

© 2023-2024 Microchip Technology Inc. and its subsidiaries

- *AN943, "Practical PICmicro® Oscillator Analysis and Design"*
- *AN949, "Making Your Oscillator Work"*
- *AN1798, "Crystal Selection for Low-Power Secondary Oscillator"*

Figure 2-3. Suggested Placement of the Oscillator Circuit



2.7 Oscillator Value Conditions on Device Start-up

If the PLL of the target device is enabled and configured for the device start-up oscillator, the maximum oscillator source frequency must be limited to a certain frequency (see 12.4.2. Phase-Locked Loop (PLL)) to comply with device PLL Start-up conditions. This means that if the external oscillator frequency is outside this range, the application must start up in the FRC mode first. The default PLL settings after a POR with an oscillator frequency outside this range will violate the device operating speed.

Once the device powers up, the application firmware can initialize the PLL SFRs, CLKDIV and PLLFBD, to a suitable value, and then perform a clock switch to the Oscillator + PLL clock source. Note that clock switching must be enabled in the device Configuration Word.

2.8 Unused I/Os

Unused I/O pins should be configured as outputs and driven to a Logic Low state. Alternatively, connect a resistor (1k-10k ohm) between V_{SS} and unused pins, and drive the output to a logic low.

2.9 Bulk Capacitors

On boards with power traces running longer than six inches in length, it is suggested to use a bulk capacitor for integrated circuits, including DSCs, to supply a local power source. The value of the bulk capacitor should be determined based on the trace resistance that connects the power supply source to the device and the maximum current drawn by the device in the application. In other words, select the bulk capacitor so that it meets the acceptable voltage sag at the device. Typical values range from 4.7 µF to 47 µF.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3. CPU

The dsPIC33AK128MC106 family has a Fixed-Point fractional DSP engine supporting the Central Processing Unit (CPU). The CPU processes instructions out of program memory and utilizes system RAM to perform tasks and calculations. The CPU is interfaced to memory and peripherals through the bus matrix. The CPU supports coprocessors, including the Floating Point Unit (FPU) for mathematical computation.

CPU key features:

- 32-bit working registers
- Unified memory map
- 5-stage instruction pipeline
- Conditional branching with speculative execution
- Instruction pre-fetch cache
- Mathematical support
- Low overhead loop support

3.1 Architectural Overview

The dsPIC33A CPU has 32-bit (data) modified, Harvard architecture with a 5-stage instruction pipeline, single phase clock design, with 32-bit instructions.

The CPU has a 32-bit instruction word with a variable length opcode field. The CPU also supports some instructions that are only available in 16-bit format. The Program Counter (PC) is 24 bits wide to access a 16MB (24-bit address) unified linear address map.

The CPU supports up to eight addressing modes. A 5-stage fully interlocked instruction pipeline with reduced branch latency and hardware mitigated pipeline hazard stalls helps maintain throughput and provides predictable execution. Most instructions execute in a single-cycle effective execution rate, with the exception of instructions that change the program flow. A hardware program loop construct is supported by the overhead free `REPEAT` instruction, which is interruptible at any point. For loops greater than one instruction, the DBT (Decrement Test and Branch) instruction may be used to reduce loop overhead.

The CPU supports High Performance Math Support with a tightly coupled 16/32-bit Integer and a Fixed-Point fractional DSP engine with a 72-bit shifter, saturation and rounding support. There is an optional common issue Single and Double Precision Floating Point Unit (FPU) coprocessor with an independent load-store execution pipeline.

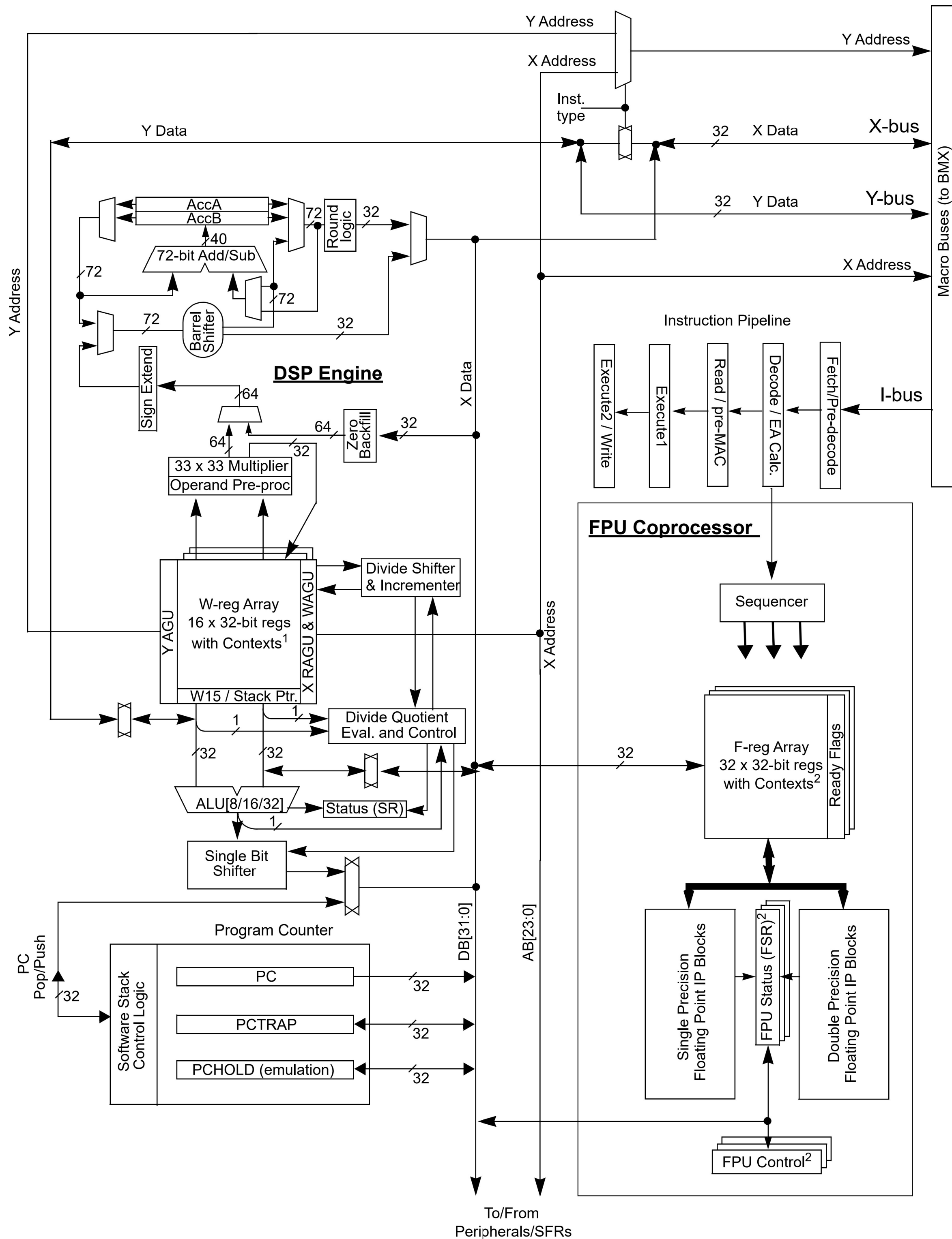
CPU Supports closely coupled coprocessor macros with the following features:

- Decode and issue from the CPU pipeline into independent coprocessor pipeline(s)
- Pipeline hazards detected and mitigated in both the CPU and coprocessor(s)
- Dedicated data move and conditional coprocessor status branch instructions
- Coprocessor interrupt support

Figure 3-1 illustrates the dsPIC33A CPU block diagram.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-1. dsPIC33A Core Conceptual Block Diagram with FPU Coprocessor



© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2 CPU Register Descriptions

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	PC	31:24								
		23:16	PC[23:16]							
		15:8	PC[15:8]							
		7:0	PC[7:0]							
0x04	SPLIM	31:24								
		23:16	SPLIM[23:16]							
		15:8	SPLIM[15:8]							
		7:0	SPLIM[7:0]							
0x08	RCOUNT	31:24	RCOUNT[31:24]							
		23:16	RCOUNT[23:16]							
		15:8	RCOUNT[15:8]							
		7:0	RCOUNT[7:0]							
0x0C	DISIIPL	31:24								
		23:16								
		15:8								
		7:0						DISIIPL[2:0]		
0x10	CORCON	31:24								
		23:16								
		15:8				US				
		7:0	SATA	SATB	SATDW	ACCSAT			RND	IF
0x14	MODCON	31:24								
		23:16								
		15:8	XMODEN	YMODEN						
		7:0	YWM[3:0]				XWM[3:0]			
0x18	XMODSRT	31:24								
		23:16	XMODSRT[23:16]							
		15:8	XMODSRT[15:8]							
		7:0	XMODSRT[7:0]							
0x1C	XMODEND	31:24								
		23:16	XMODEND[23:16]							
		15:8	XMODEND[15:8]							
		7:0	XMODEND[7:0]							
0x20	YMODSRT	31:24								
		23:16	YMODSRT[23:16]							
		15:8	YMODSRT[15:8]							
		7:0	YMODSRT[7:0]							
0x24	YMODEND	31:24								
		23:16	YMODEND[23:16]							
		15:8	YMODEND[15:8]							
		7:0	YMODEND[7:0]							
0x28	XBREV	31:24								
		23:16								
		15:8		XBREV[14:8]						
		7:0	XBREV[7:0]							
0x2C	PCTRAP	31:24								
		23:16		PCTRAP[22:16]						
		15:8	PCTRAP[15:8]							
		7:0	PCTRAP[7:0]							
0x30	FEX	31:24	FEX[31:24]							
		23:16	FEX[23:16]							
		15:8	FEX[15:8]							
		7:0	FEX[7:0]							
0x34	FEX2	31:24	FEX2[31:24]							
		23:16	FEX2[23:16]							
		15:8	FEX2[15:8]							
		7:0	FEX2[7:0]							

© 2023-2024 Microchip Technology Inc. and its subsidiaries

...........continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x38	PCHOLD	31:24								
		23:16	PCHOLD[23:16]							
		15:8	PCHOLD[15:8]							
		7:0	PCHOLD[7:0]							
0x3C	VFA	31:24								
		23:16	VFA[23:16]							
		15:8	VFA[15:8]							
		7:0	VFA[7:0]							
0x40 ... 0x1E0F	Reserved									
0x1E10	HPCCON	31:24								
		23:16								
		15:8	ON		CLR					
		7:0								
0x1E10	HPCSEL0	31:24				SELECT[3][4:0]				
		23:16				SELECT[2][4:0]				
		15:8				SELECT[1][4:0]				
		7:0				SELECT[0][4:0]				
0x1E14	HPCSEL1	31:24				SELECT[7][4:0]				
		23:16				SELECT[6][4:0]				
		15:8				SELECT[5][4:0]				
		7:0				SELECT[4][4:0]				
0x1E18 ... 0x1E1F	Reserved									
0x1E20	HPCCNTL0	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E24	HPCCNTH0	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E28	HPCCNTL1	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E2C	HPCCNTH1	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E30	HPCCNTL2	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E34	HPCCNTH2	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E38	HPCCNTL3	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E3C	HPCCNTH3	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							

© 2023-2024 Microchip Technology Inc. and its subsidiaries

...........continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x1E40	HPCCNTL4	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E44	HPCCNTH4	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E48	HPCCNTL5	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E4C	HPCCNTH5	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E50	HPCCNTL6	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E54	HPCCNTH6	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E58	HPCCNTL7	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E5C	HPCCNTH7	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E60	CHECON	31:24								
		23:16								ISBBUF
		15:8	ON				CHEINV	CHECOH		
		7:0								FLTINJ
0x1E64	CHESTAT	31:24								
		23:16								
		15:8								
		7:0						TPE	RD	PAR
0x1E68	CHEFLTINJ	31:24								
		23:16								
		15:8								
		7:0	FLTPTR[7:0]							

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.1 CPU Program Counter Register

Name: PC
Offset: 0x000

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	PC[23:16]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	PC[15:8]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	PC[7:0]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bits 23:0 – PC[23:0] Program Counter bits

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.2 Stack Pointer Limit Value Register

Name: SPLIM
Offset: 0x004

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	SPLIM[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	SPLIM[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	SPLIM[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 23:0 – SPLIM[23:0] Stack Limit Address bits

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.3 REPEAT Loop Counter Register

Name: RCOUNT
Offset: 0x008

Bit	31	30	29	28	27	26	25	24
	RCOUNT[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	RCOUNT[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	RCOUNT[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

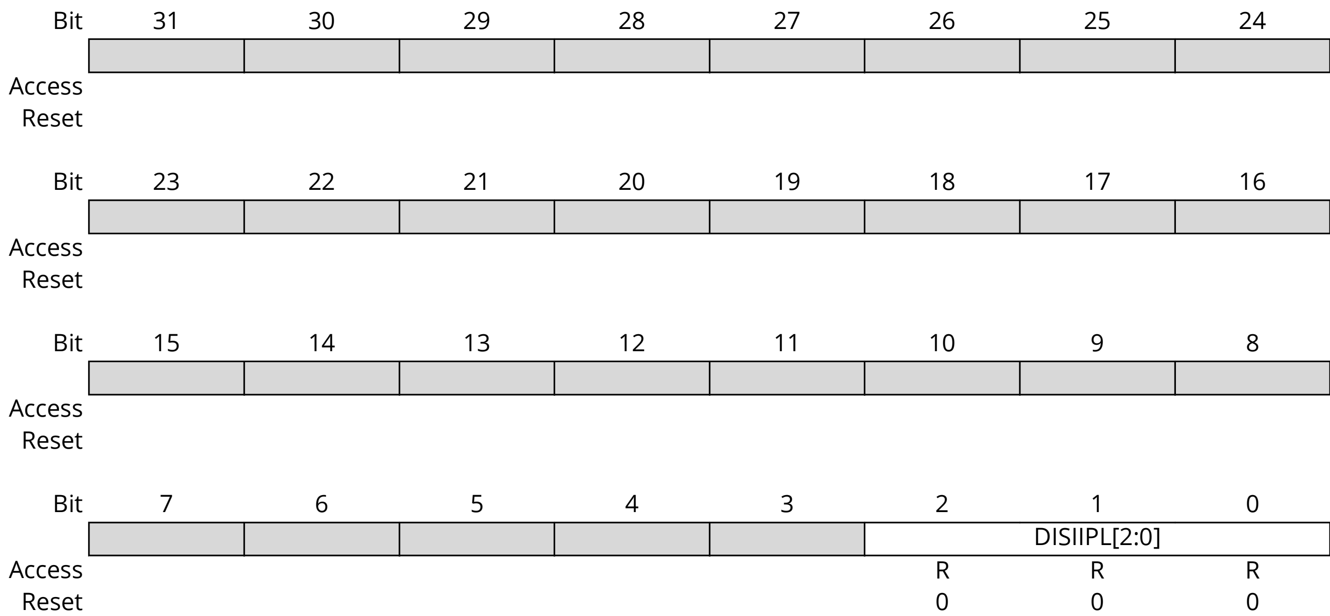
Bit	7	6	5	4	3	2	1	0
	RCOUNT[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – RCOUNT[31:0] Current Loop Counter Value for `REPEAT` Instruction

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.4 DISIIPL(W) Instruction Current IPL Threshold

Name: DISIIPL
Offset: 0x00C



Bits 2:0 – DISIIPL[2:0] DISIIPL(W) current IPL threshold value

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.5 Core Mode Control Register(1)

Name: CORCON
Offset: 0x010

Note:
1. The Core Control register (CORCON) has bits that control the operation of the DSP multiplier hardware. The IPL3 bit is concatenated with the IPL[2:0] bits (SR[7:5]) to form the CPU Interrupt Priority Level.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
				US				
Access				R/W				
Reset				0				

Bit	7	6	5	4	3	2	1	0
	SATA	SATB	SATDW	ACCSAT			RND	IF
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

Bit 12 – US Unsigned or Signed Multiplier Mode Select bit

Value	Description
1	Unsigned mode enabled for DSP ops
0	Signed mode enabled for DSP ops

Bit 7 – SATA AccA Saturation Enable bit

Value	Description
1	Accumulator A saturation enabled
0	Accumulator A saturation disabled

Bit 6 – SATB AccB Saturation Enable bit

Value	Description
1	Accumulator B saturation enabled
0	Accumulator B saturation disabled

Bit 5 – SATDW Data Space Write from DSP Engine Saturation Enable bit

Value	Description
1	Data Space write saturation enabled
0	Data Space write saturation disabled

Bit 4 – ACCSAT Accumulator Saturation Mode Select bit

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Value	Description
1	9.63 saturation (super saturation)
0	1.63 saturation (normal saturation)

Bit 1 – RND Rounding Mode Select bit

Value	Description
1	Biased (conventional) rounding enabled
0	Unbiased (convergent) rounding enabled

Bit 0 – IF Integer or Fractional Multiplier Mode Select bit

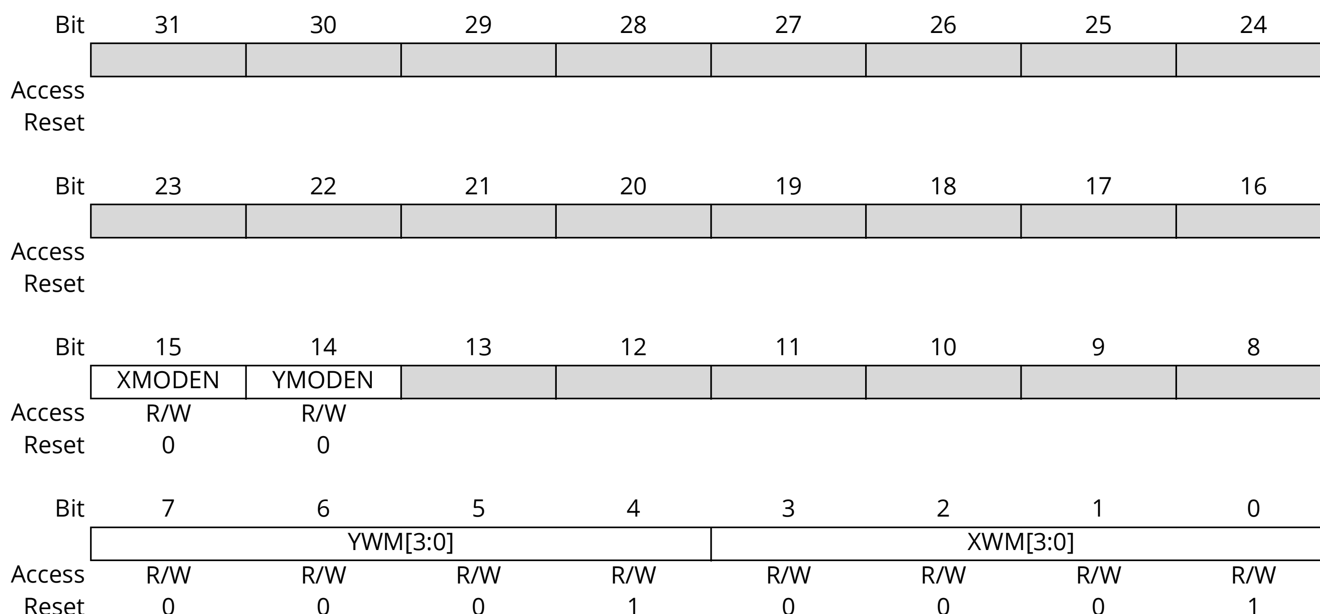
Value	Description
1	Integer mode is enabled for DSP multiply
0	Fractional mode is enabled for DSP multiply

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.6 Modulo Addressing Control Register[1]

Name: MODCON
Offset: 0x0014

Note:
1. The MODCON register enables and configures Modulo Addressing (circular buffers).



Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	XMODEN	YMODEN						
Access	R/W	R/W						
Reset	0	0						

Bit	7	6	5	4	3	2	1	0
	YWM[3:0]				XWM[3:0]			
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	1	0	0	0	1

Bit 15 – XMODEN X RAGU & X WAGU Modulus Addressing Enable bit

Value	Description
1	X AGU Modulus Addressing enabled
0	X AGU Modulus Addressing disabled

Bit 14 – YMODEN Y AGU Modulus Addressing Enable bit

Value	Description
1	Y AGU Modulus Addressing enabled
0	Y AGU Modulus Addressing disabled

Bits 7:4 – YWM[3:0] Y AGU W Register Select for Modulo Addressing bit

Value	Description
1111	Modulo Addressing disabled (W15 does not support Modulo Addressing)
1110	W14 selected for Modulo Addressing
...	
0000	W0 selected for Modulo Addressing

Bits 3:0 – XWM[3:0] X RAGU & X WAGU W Register Select for Modulo Addressing bit

Value	Description
1111	Modulo Addressing disabled (W15 does not support Modulo Addressing)
1110	W14 selected for Modulo Addressing
...	
0000	W0 selected for Modulo Addressing

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.7 X AGU Modulo Addressing Start Register

Name: XMODSRT
Offset: 0x0018

Note:
1. The XMODSRT and XMODEND registers hold the start and end addresses for modulo (circular) buffers implemented in the X data memory address space.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	XMODSRT[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	XMODSRT[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	XMODSRT[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 23:0 – XMODSRT[23:0] X RAGU & X WAGU Modulo Addressing Start Address bits[1]

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.8 X AGU Modulo Addressing End Register[1]

Name: XMODEND
Offset: 0x001C

Note:

1. The XMODSRT and XMODEND registers hold the start and end addresses for modulo (circular) buffers implemented in the X data memory address space.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	XMODEND[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	XMODEND[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	XMODEND[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 23:0 – XMODEND[23:0] X RAGU & X WAGU Modulo Addressing End Address bits

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.9 Y AGU Modulo Addressing Start Address Register[1]

Name: YMODSRT
Offset: 0x0020

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	YMODSRT[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	YMODSRT[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	YMODSRT[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 23:0 – YMODSRT[23:0] Y RAGU Modulo Addressing Start Address bits

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.10 Y AGU Modulo Addressing End Register[1]

Name: YMODEND
Offset: 0x0024

Note:
1. The YMODSRT and YMODEND registers hold the start and end addresses for modulo (circular) buffers implemented in the Y data memory address space.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	YMODEND[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	YMODEND[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	YMODEND[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

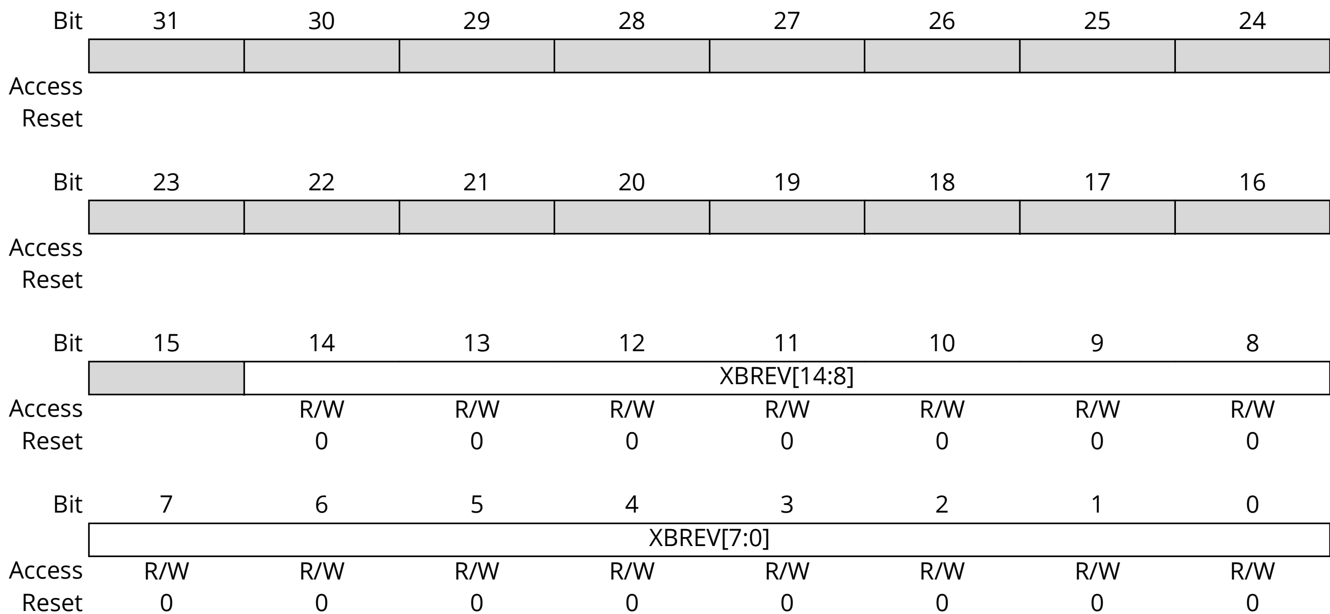
Bits 23:0 – YMODEND[23:0] Y RAGU Modulo Addressing End Address bits

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.11 X AGU Bit Reversal Addressing Control Register[1]

Name: XBREV
Offset: 0x0028

Note:
1. The XBREV register sets the buffer size used for Bit-Reversed Addressing.



Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
		XBREV[14:8]						
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	XBREV[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 14:0 – XBREV[14:0] X AGU Bit Reversed Modifier bits

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.12 Captured PC Address at Time of Trap Register

Name: PCTRAP
Offset: 0x002C

Notes:
1. PCTRAP[0] always reads as `0`.
2. If the current IPL is greater or equal to 8, the PC address will not be captured.
3. Hardware update is blocked after the first PCTRAP update occurs, preventing newer traps from overwriting the source address of older ones. Update can be re-enabled by user attempting to write 24’h000000 to PCTRAP (write will not occur, preserving PCTRAP contents).

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
		PCTRAP[22:16]						
Access		R	R	R	R	R	R	R
Reset		0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	PCTRAP[15:8]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	PCTRAP[7:0]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bits 22:0 – PCTRAP[22:0] Captured PC Address at time of trap exception[1,2,3]

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.13 Force Execution Instruction Register 1(1)

Name: FEX
Offset: 0x0030

Bit	31	30	29	28	27	26	25	24
	FEX[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	FEX[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	FEX[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	FEX[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	1

Bits 31:0 – FEX[31:0] For 2 word operations, FEX contains the first instruction to be executed using the UFEX instruction.

FEX is only visible as a R/W register in Debug mode. In all other operating modes, it is read-only of all 0's.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.14 Force Execution Instruction Register 2[1]

Name: FEX2
Offset: 0x0034

Bit	31	30	29	28	27	26	25	24
	FEX2[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	FEX2[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	FEX2[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	FEX2[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	1

Bits 31:0 – FEX2[31:0] For 2 word operations, FEX contains the second instruction to be executed using the UFEX instruction.

FEX is only visible as a R/W register in Debug mode. In all other operating modes, it is read-only of all 0's.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.15 Debug Hold PC Register

Name: PCHOLD
Offset: 0x0038

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	PCHOLD[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	PCHOLD[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	PCHOLD[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	1

Bits 23:0 – PCHOLD[23:0] Debug Hold PC register bits
PCHOLD is only visible as a R/W register in Debug mode. In all other operating modes, it is read-only of all 0's.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.16 Vector Fail Address Register

Name: VFA
Offset: 0x003C

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	VFA[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	VFA[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	VFA[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	1

Bits 23:0 – VFA[23:0] Vector Fail Address Register bits

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.2.17 CPU STATUS Register[1]

Name: SR

Note:
1. The CPU STATUS register is not memory mapped.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	VF					CTX[2:0]		
Access	R					R	R	R
Reset	0					0	0	0

Bit	15	14	13	12	11	10	9	8
	OA	OB	SA	SB	OAB	SAB		IPL3
Access	R/W	R/W	R/W	R/W	R	R/C		R/C
Reset	0	0	0	0	0	0		0

Bit	7	6	5	4	3	2	1	0
	IPL[2:0]			RA	N	OV	Z	C
Access	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 23 – VF Vector (Fetch) Fail Status bit

Value	Description
1	Indicates to the Bus Error handler that the source of the bus error is a vector fetch. The vector data read will be substituted with the contents of the Vector Fail Address (VFA) SFR.
0	Indicates to the Bus Error handler that the source of the bus error is not a vector fetch.

Bits 18:16 – CTX[2:0] Current (W register) Context Identifier bits

Value	Description
111	Context 7 is currently in use
110	Context 6 is currently in use
101	Context 5 is currently in use
100	Context 4 is currently in use
011	Context 3 is currently in use
010	Context 2 is currently in use
001	Context 1 is currently in use
000	Context 0 is currently in use

Bit 15 – OA Accumulator A Fractional Overflow Status bit

Value	Description
1	Accumulator A fractional overflow has occurred (its contents can no longer be represented as a 1.31 fractional value)
0	Accumulator A not overflowed

Bit 14 – OB Accumulator B Fractional Overflow Status bit

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Value	Description
1	Accumulator B fractional overflow has occurred (its contents can no longer be represented as a 1.31 fractional value)
0	Accumulator B not overflowed

Bit 13 – SA Accumulator A Saturation/Sign Overflow 'Sticky' Status bit

Value	Description
1	Accumulator A is saturated, or has been saturated at some time, or has overflowed into bit 71 (if saturation is disabled)
0	Accumulator A is not saturated or has not overflowed into bit 71 (if saturation is disabled)

Bit 12 – SB Accumulator B Saturation/Sign Overflow 'Sticky' Status bit

Value	Description
1	Accumulator B is saturated, or has been saturated at some time, or has overflowed into bit 71 (if saturation is disabled)
0	Accumulator B is not saturated or has not overflowed into bit 71 (if saturation is disabled)

Bit 11 – OAB OA || OB Combined Accumulator Fractional Overflow Status bit

Value	Description
1	Accumulators A or B fractional overflow has occurred (one or both of their contents can no longer be represented as a 1.31 fractional value)
0	Neither Accumulators A nor B have overflowed

Bit 10 – SAB SA || SB Combined Accumulator 'Sticky' Status bit

Value	Description
1	Accumulators A or B are saturated, or have been saturated at some time, or have overflowed into bit 71 (if saturation is disabled)
0	Neither Accumulator A nor B are saturated or have overflowed into bit 71 (if saturation is disabled)

Bit 8 – IPL3 MS-bit of CPU Priority Level Nibble bit

Value	Description
1	CPU Priority ≥ 8 (trap exception underway)
0	CPU Priority < 8 (no trap exception underway)

Bits 7:5 – IPL[2:0] CPU Interrupt Priority Level status bits

User Mode: This bit is R/C-0 (read only if Supervisor Mode supported) and will reset to 1'b0.
Supervisor Mode: This bit is R/C-0 (CPU will reset into Supervisor Mode).

Value	Description
111	All interrupts disabled
110	Level 7 interrupts enabled
101	Level 6 and 7 interrupts enabled
100	Level 5 through 7 interrupts enabled
011	Level 4 through 7 interrupts enabled
010	Level 3 through 7 interrupts enabled
001	Level 2 through 7 interrupts enabled
000	Level 1 through 7 interrupts enabled

Bit 4 – RA REPEAT Loop Active bit

Value	Description
1	REPEAT loop in progress
0	REPEAT loop not in progress

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Bit 3 – N ALU Negative bit

Bit 2 – OV ALU Overflow bit

Bit 1 – Z ALU 'Sticky' Zero bit

Value	Description
1	An operation which effects the Z bit has set it at some time in the past
0	The most recent operation which effects the Z bit has cleared it (i.e. a non-zero result)

Bit 0 – C ALU Carry/$\overline{\text{Borrow}}$ bit
SR[31:0] is stacked during exception processing, preserving context.

3.3 CPU Operation

3.3.1 Instruction Set

The dsPIC33A instruction set has two classes of instructions: MCU instructions and DSP instructions. These two classes are seamlessly integrated into the architecture and execute from a single execution unit. The instruction supports integer, fixed point and floating-point math operation.

3.3.2 Data Space Addressing

The Data Space is split into two blocks as X and Y data memory. Each memory block has its own independent Address Generation Unit (AGU). The MCU class of instructions operates solely through the X memory AGU, which accesses the entire memory map as one linear data space. Certain DSP instructions operate through the X and Y AGUs to support dual operand reads, which splits the data address space into two parts.

In dsPIC33A devices, overhead-free circular buffers (Modulo Addressing mode) are supported in both X and Y address spaces. The Modulo Addressing removes the software boundary checking overhead for DSP algorithms. The X AGU Circular Addressing can be used with any of the MCU class of instructions. The X AGU also supports the Bit-Reversed Addressing mode to greatly simplify input or output data reordering for radix-2 FFT algorithms.

3.3.3 Addressing Modes

The CPU supports up to eight addressing modes as shown in Table 3-1

Table 3-1. MCU Instruction Addressing Mode Definitions

Function (Source, ppp)	Function (Destination, qqq)	Description
EA = [Ws + Wb]	EA = [Wd + Wb]	Indirect with (signed) Register Offset
EA = SR	EA = SR	Status Register direct
EA = [Ws+=1]	EA = [Wd+=1]	Register indirect pre-incremented
EA = [Ws-=1]	EA = [Wd-=1]	Register indirect pre-decremented
EA = [Ws]+= 1	EA = [Wd]+= 1	Register indirect post-incremented
EA = [Ws]-= 1	EA = [Wd]-= 1	Register indirect post-decremented
EA = [Ws]	EA = [Wd]	Register indirect
EA = Ws	EA = Wd	Register direct

Each instruction is associated with a predefined addressing mode group, depending upon its functional requirements. For most instructions, the dsPIC33A CPU can execute all of the following functions in a single instruction cycle:

- Data memory read
- Working register (data) read
- Data memory write

© 2023-2024 Microchip Technology Inc. and its subsidiaries

- Program (instruction) memory read

As a result, three-operand instructions can be supported, allowing A + B = C operations to be executed in a single cycle.

3.3.4 Programmer's Model

The programmer's model for the dsPIC33A CPU is shown in Figure 3-2. All registers in the programmer's model are memory-mapped and can be manipulated directly by instructions. Table 3-2 provides a description of each register in the programmer's model.

In addition to the registers contained in the programmer's model, the dsPIC33A devices contain control registers for Modulo Addressing, Bit-Reversed Addressing and Interrupts. These registers are described in subsequent sections of this document.

All registers associated with the programmer's model are shown in Figure 3-2.

Table 3-2. Programmer's Model Register Descriptions

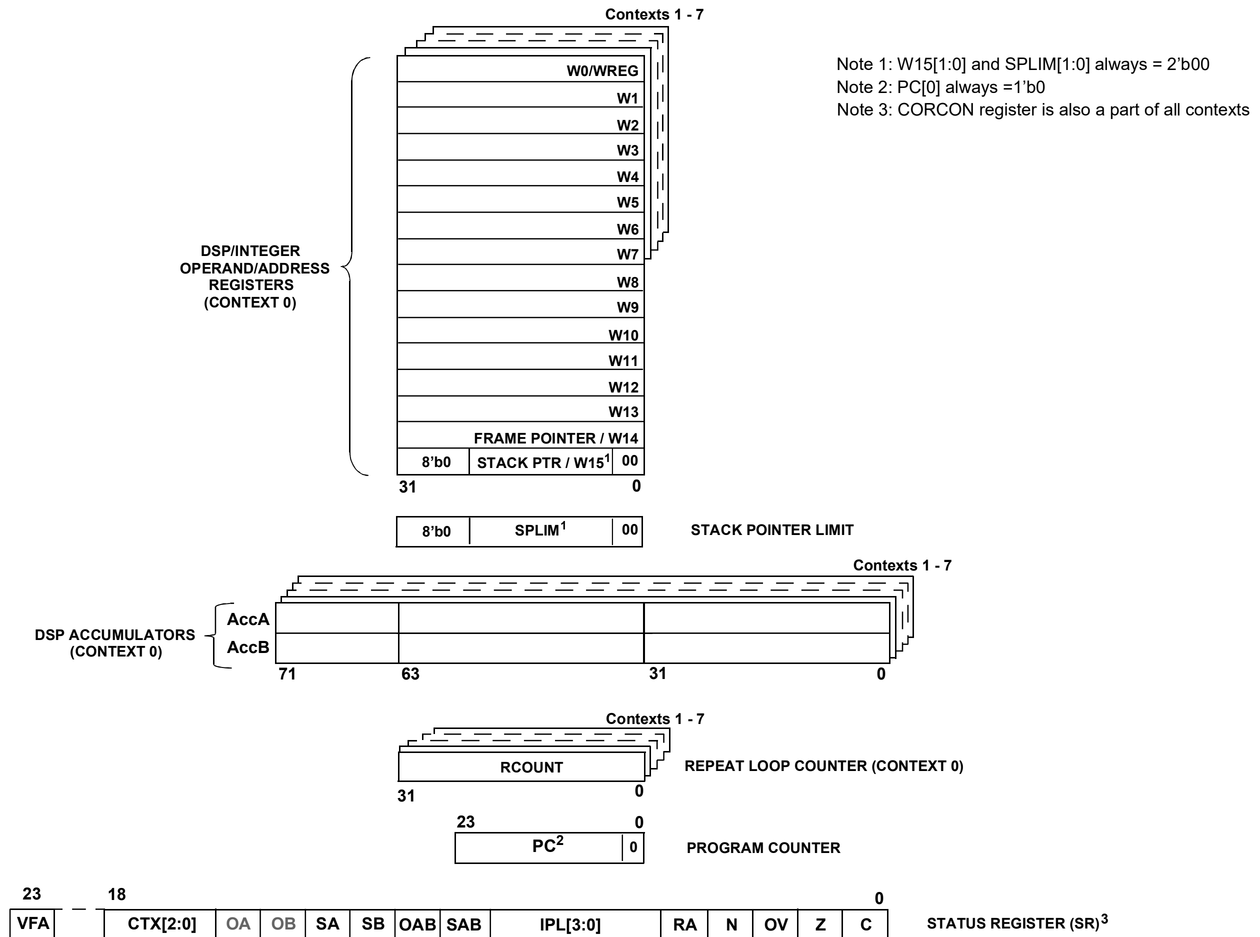
Register(s) Name	Description
W0 through W15[(1)]	Working Register Array (Default Context)
W0 through W7[(1,2)]	Working Register Array (Alternate Context 1-7)
ACCA,ACCB[(1)]	72-bit DSP Accumulators (Context 0-7)
PC	24-bit Program Counter
SR[(1)]	ALU and DSP Engine Status Register
SPLIM	Stack Pointer Limit Value Register
RCOUNT	`32-bit REPEAT` Loop Count Register (Context 0-7)
CORCON	DSP Engine Configuration

Notes:

1. W0 through W15, ACCx and SR are not mapped to memory.
2. W0 through W7 are part of Alternate W register sets.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-2. dsPIC33A CPU Programmer's Model



3.3.5 DSP Engine and Instructions

The DSP engine features:

- A high-speed, 33-bit by 33-bit multiplier
- A 72-bit ALU
- Two 72-bit saturating accumulators
- A 72-bit bidirectional barrel shifter, capable of shifting a 40-bit value up to 32 bits right, or up to 32 bits left, in a single cycle

The DSP instructions operate seamlessly with all other instructions and are designed for optimal real-time performance. The `MAC` instruction, and other associated instructions, can concurrently fetch two data operands from memory while multiplying two W registers. This requires that the data space be split for these instructions and linear for all others.

3.3.6 Exception Processing

The dsPIC33A devices have a vectored exception scheme, with up to eight possible sources of non-maskable traps and up to 246 possible interrupt sources. Each interrupt source can be assigned to one of seven priority levels.

In addition, each of the Alternate W register contexts can be associated with its own Interrupt Priority Level (IPL) for exception handling. See 3.3.9. Alternate Working Register Arrays for more information.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.3.7 CPU Register Descriptions

3.3.7.1 SR: CPU STATUS Register

The dsPIC33A CPU has a 32-bit STATUS Register (SR). A detailed description of the CPU SR is shown in 3.2.17. SR.

SR contains:

- All ALU Operation Status flags
- The CPU Interrupt Priority Level Status bits, IPL[3:0]
- The `REPEAT` Loop Active Status bit, RA (SR[4])
- The DSP Adder/Subtracter Status bits

The SR bits are readable/writable with the following exceptions:

- The RA bit (SR[4]) is read-only
- The OA, OB (SR[15:14]), OAB (SR[11]), SA, SB (SR[13:12]) and SAB (SR[10]) bits are readable and writable; however, once set, they remain set until cleared by the user application, regardless of the results from any subsequent DSP operations.
  Note: Clearing the SAB bit also clears both the SA and SB bits. Similarly, clearing the OAB bit also clears both the OA and OB bits. A description of the STATUS Register bits affected by each instruction is provided in the *"dsPIC33A Programmer's Reference Manual"*.
- The CTX bit (SR[18:16]) is read-only; it reflects which W register context is currently in use by the CPU
- The VF bit (SR[23]) is read-only

3.3.7.2 CORCON: Core Control Register

The Core Control register (CORCON) has bits that control the operation of the DSP multiplier.

3.3.8 Working Register Array

The Working (W) registers can function as data, address or address offset registers. The function of a W register is determined by the addressing mode of the instruction that accesses it.

The dsPIC33A instruction set can be divided into two instruction types: Register instructions and File register instructions.

3.3.8.1 Register Instructions

Register instructions can use each W register as a data value or an address offset value. Example 3-1 shows register instructions.

Example 3-1. Register Instructions

```
MOV.w    W0, W1                ; move contents of W0 to W1
MOV.w    W0, [W1]              ; move W0 to address contained in W1
ADD.w    W0, [W4], W5          ; add contents of W0 to contents pointed
                             ; to by W4. Place result in W5.
```

3.3.8.2 File Register Instructions

File register instructions operate on a specific memory address contained in the instruction opcode and register, W0. W0 is a special Working register used in File register instructions.

The File register address space is determined by the maximum address range of the file instructions, which is either 64 KB (if a W-reg operand is required) or 1 MB (if no W-reg operand is required), and encompasses the user RAM area and Special Function Registers (SFRs) within DS.

Example 3-2 shows File register instructions.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Example 3-2. File Register Instructions

```
ADD.w 0x4500, Wn                ; (0x4500)+w0 -> 0x4500
ADD.w 0x4500, w0, Wn                      ; (0x4500)+w0 -> 0x4500
ADD.w 0x4500, w4, Wn                      ; (0x4500)+w4 -> 0x4500
```

3.3.8.3 W Register Memory Mapping

The W registers are not memory-mapped, and thus, it is not possible to access a W register in a File register instruction. This helps in eliminating data hazards.

3.3.8.4 W Registers and Byte Mode Instructions

Byte instructions that target the W register array affect only the Least Significant Byte (LSB) of the target register. Since the Working registers are memory-mapped, the LSB and the Most Significant Byte (MSB) can be manipulated through byte-wide data memory space accesses.

3.3.9 Alternate Working Register Arrays

Alternate Working register arrays are a subset of the Working registers (W0 through W7). Depending on the specific device, up to seven Alternate Working register arrays may be implemented. Each set implements registers W0 through W7, AccA, AccB, RCOUNT, and DSP related CORCON control bits (US, SATA, SATB, SATDW, ACCSAT, RND, IF).

The Alternate W registers are not memory-mapped to data memory space just like the default W array.

All W register arrays are persistent; that is to say, the contents of the default and Alternate W registers do not change whenever the CPU switches to another set. This saves time by reducing the amount of saving and restoring of register contents, making this very useful for time-critical applications.

Each Alternate W array is inherently assigned to a respective IPL (e.g., IPL4 is assigned to Context 4) and Interrupt Service Routine (ISR) in the application code. The Current Context Identifier (CTX[2:0]) status field is located within the Status Register (SR). Each context is associated with a specific Interrupt Priority Level (IPLV). The context is exited during execution of RETFIE instruction of the interrupt ISR.

During an exception processing, the (CTX[2:0]) status field located within the Status Register (SR) is stacked. The stacked SR.CTX[2:0] represents the CPU register context in use at the time of the exception. The value is updated whenever the register context is changed, either through automatic interrupt-based hardware switching, or as the result of a context change brought about by the execution of a CTXTSWP{W} instruction.

Depending on the device, different context Working register behavior can be observed with nested interrupts.

Consider the example, as shown in Figure 3-3, where there are nested interrupts. In this case, the system is configured as follows:

- Timer1 interrupt with an Interrupt Priority Level (IPL) of 1. The Alternate Working Register Set 1 (CTX1) has an IPL of 1.
- ADCAN1 interrupt with an IPL of 4. The Alternate Working Register Set 4 (CTX4) has an IPL of 4.
- PWM1 interrupt with an IPL of 5, The Alternate Working Register Set 5 (CTX5) has an IPL of 5.

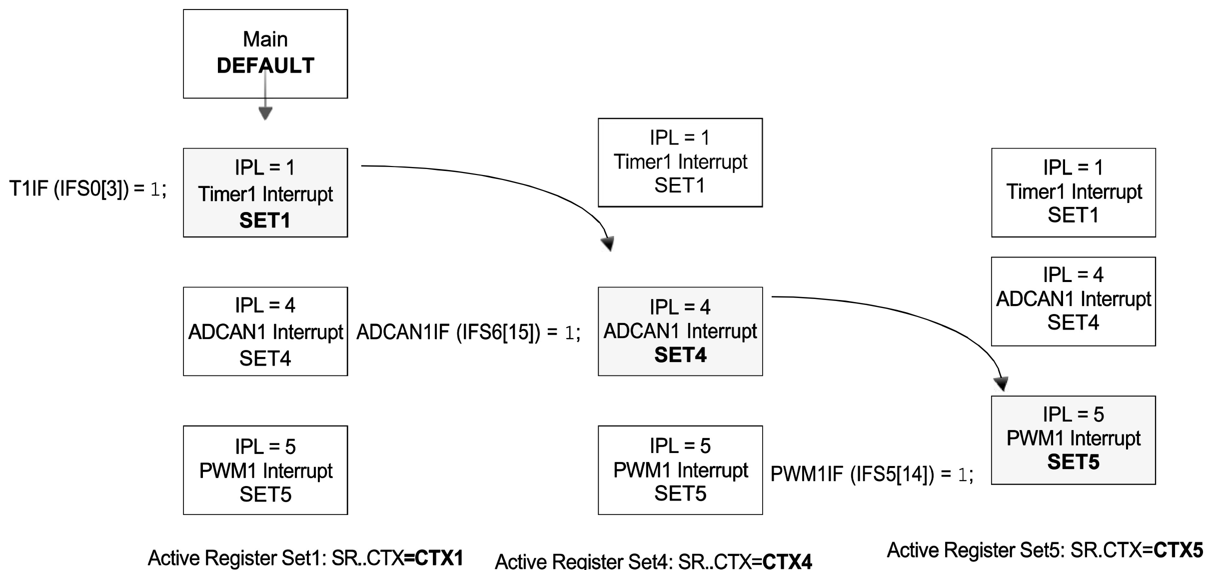
The application begins in the main function. At some point in time, the Timer1 interrupt flag is set and the program jumps to the Timer1 ISR. The register set switches from the default Working register set 0 to the Alternate Working register set 1, CTX1. At some point during the Timer1 ISR, the ADCAN1 conversion completes, and its interrupt flag is set. Because it has a higher IPL, the program jumps to the ADCAN1 ISR. The register set switches from the set 1, CTX1 Alternate Working register set to the Alternate Working register set 4, CTX4. At some point during the ADCAN1 ISR, the PWM1

© 2023-2024 Microchip Technology Inc. and its subsidiaries

interrupt flag is set. Because the PWM1 IPL is higher than the ADCAN1 IPL, the program jumps to the PWM1 ISR and remains in the Alternate Working register set 5 CTX5.

Once the PWM ISR execution is completed, the program jumps back to the ADCAN1 ISR using CTX4. Similarly, after the execution of the ADCAN1 ISR, the program jumps back to the Timer1 ISR using CTX1. Exceptions above IPL7 (i.e., traps) will execute in whatever register context the CPU was in prior to the trap event.

Figure 3-3. Nested Interrupt Context Flow for dsPIC33A Devices



3.3.9.1 Alternate Working Register Set

Alternatively, before enabling interrupts associated with a particular context, the application may manually switch to it by executing the `CTXTSWP` instruction. `CTXTSWP` does not affect the CPU IPL; it is used to support software context switching for either context initialization, run-time usage of contexts within procedure calls or the like, thus operating independently from the interrupt system.

3.3.10 Software Stack Pointer

The W15 register serves as a dedicated Software Stack Pointer (SSP) and is automatically modified by exception processing, subroutine calls and returns; however, W15 can be referenced by any instruction in the same manner as all other W registers. This simplifies reading, writing and manipulating the Stack Pointer (for example, creating stack frames).
Note: To protect against misaligned stack accesses, W15[1:0] is fixed to '`00`' by the hardware.

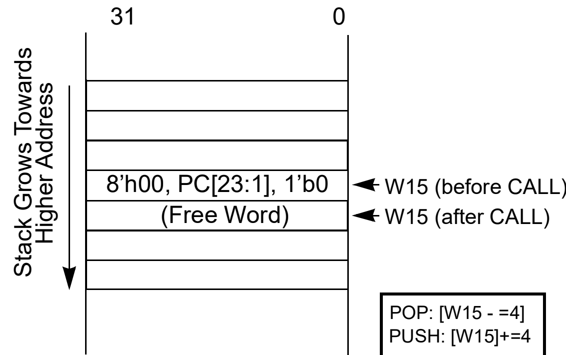
W15 is initialized to 0x4000 during all Resets. This address ensures that the Software Stack Pointer points to valid RAM in all dsPIC33A devices and permits stack availability for non- maskable trap exceptions. These can occur before the SSP is initialized by the user software. Reprogramming the SSP to any location within data space is possible during initialization.

The Software Stack Pointer always points to the first available free word in the data space (RAM) and fills the software stack, working from lower toward higher addresses. Figure 3-4 illustrates how it pre-decrements for a stack pop (read) and post-increments for a stack push (writes).

When the PC is pushed onto the stack, PC[23:0] are pushed onto the first available stack word, as shown in Figure 3-4.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-4. Stack Operation for a `CALL` Instruction



3.3.10.1 Software Stack Examples

The software stack is manipulated using the `PUSH` and `POP` instructions. The `PUSH` and `POP` instructions are the equivalent of a `MOV` instruction with W15 as the Destination Pointer. For example, the contents of W0 can be pushed onto the stack by:

```
PUSH W0
```

This syntax is equivalent to:

```
MOV.L W0,[W15++]
```

The contents of the Top-of-Stack (TOS) can be returned to W0 by:

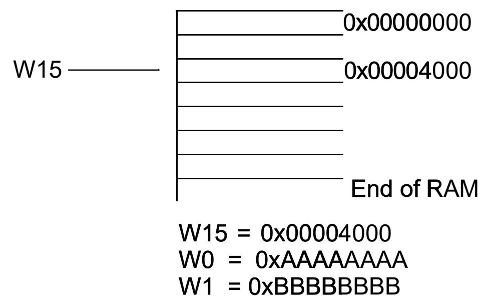
```
POP W0
```

This syntax is equivalent to:

```
MOV.L [--W15],W0
```

Figure 3-5 through Figure 3-8 illustrate examples of how the software stack is used. Figure 3-5 illustrates the software stack at device initialization. W15 has been initialized to 0x00004000. This example assumes the values, 0xAAAAAAAA and 0xBBBBBBBB, have been written to W0 and W1, respectively. In Figure 3-6, the stack is pushed for the first time and the value contained in W0 is copied to the stack. W15 is automatically updated to point to the next available stack location (0x00004004). In Figure 3-7, the contents of W1 are pushed onto the stack. Figure 3-8 illustrates how the stack is popped and the Top-of-Stack value (previously pushed from W1) is written to W3.

Figure 3-5. Stack Pointer at Device Reset



© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-6. Stack Pointer After the First `PUSH` Instruction

	Address
	0x000000
0xAAAAAAAA	0x004000
	0x004004 ← W15
	End of RAM

W15 = 0x00004004
W0 = 0xAAAAAAAA
W1 = BBBBBBBB

```
PUSH W0
```

Figure 3-7. Stack Pointer After the Second `PUSH` Instruction

	Address
	0x000000
0xAAAAAAAA	0x004000
0xBBBBBBBB	0x004004
	0x004008 ← W15
	End of RAM

W15 = 0x00004008
W0 = 0xAAAAAAAA
W1 = 0xBBBBBBBB

```
PUSH W1
```

Figure 3-8. Stack Pointer After a `POP` Instruction

	Address
	0x000000
0xAAAAAAAA	0x004000
0xBBBBBBBB	0x004004 ← W15
	End of RAM

W15 = 0x00004004
0xBBBBBBBB→W3

```
POP W3
```

3.3.10.2 W14 Software Stack Frame Pointer

A frame is a user-defined section of memory in the stack that is used by a single function. The Working register, W14, can be used as a Stack Frame Pointer with the `LNK` (link) and `ULNK` (unlink) instructions. W14 can be used in a normal Working register by instructions when it is not used as a Frame Pointer.

3.3.10.3 Stack Pointer Overflow

The Stack Pointer Limit (SPLIM) register specifies the size of the stack buffer. SPLIM is a 32-bit register, but SPLIM[1:0] is fixed to '`00`' because all stack operations must be long word-aligned.

The stack overflow check is not enabled until a Long word write to SPLIM occurs. After this, it can only be disabled by a device Reset. All Effective Addresses (EAs), generated using W15 as a source or destination, are compared against the value in SPLIM. If Effective Addresses (EAs) exceed the contents of the SPLIM register, and a `PUSH` operation is performed, a stack error trap occurs on a subsequent `PUSH` operation. For example, if it is desirable to cause a stack error trap when the stack grows beyond address 0x5000 in RAM, initialize the SPLIM with the value 0x4FFC.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Note: A stack error trap can be caused by any instruction that uses the contents of the W15 register to generate an Effective Address (EA). Therefore, if the contents of W15 are greater than the contents of the SPLIM register by a value of four, and a `CALL` instruction is executed or if an interrupt occurs, a stack error trap is generated.

If stack overflow checking is enabled, a stack error trap also occurs if the W15 Effective Address calculation wraps over the end of data space.

A pre/post inc/dec operation is performed on W15 that results in EA[1:0] != 2'b00 (i.e., not long word aligned). This will detect byte and word pre/post inc/dec operations that are otherwise considered aligned but would result in a misaligned Stack Pointer.

Note: A write to the SPLIM should not be followed by an indirect read operation using W15.

3.3.10.4 Stack Pointer Underflow

The stack is initialized to 0x4000 during a Reset. A stack error trap is initiated if the Stack Pointer address is less than 0x4000.

Note: Locations in data space between 0x0000 and 0x3FFF are, in general, reserved for core and peripheral Special Function Registers (SFRs).

3.3.11 Arithmetic Logic Unit (ALU)

The dsPIC33A ALU is 32 bits wide and is capable of addition, subtraction, single bit shifts and logic operations. Unless otherwise mentioned, arithmetic operations are 2's complement in nature. Depending on the operation, the ALU can affect the values of the following bits in the STATUS Register:

- Carry (C)
- Zero (Z)
- Negative (N)
- Overflow (OV)

The ALU can perform 8/16-bit or 32-bit operations, depending on the mode of the instruction that is used. Data for the ALU operation can come from the W register array or data memory depending on the addressing mode of the instruction. Likewise, output data from the ALU can be written to the W register array or a data memory location.

Note:

1. Byte operations use the 16-bit ALU and can produce results in excess of eight bits. However, to maintain backward compatibility with PIC® MCU devices, the ALU result from all byte operations is written back as a byte (i.e., the MSB is not modified) and the STATUS Register is updated based only upon the state of the LSB of the result.

3.3.11.1 Byte to Word Conversion

The dsPIC33A CPU has two instructions that are helpful when mixing 8-bit and 16-bit ALU operations.

The Sign-Extend (`SE`) instruction takes a byte value in a W register or data memory and creates a sign-extended word value that is stored in a W register.

The Zero-Extend (`ZE`) instruction clears the 8 MSbs of a word value in a W register or data memory and places the result in a destination W register.

3.3.12 DSP Engine

The DSP engine is a block of hardware that is fed data from the W register array, but contains its own specialized result registers. The DSP engine is driven from the same instruction decoder that directs the MCU ALU. In addition, all operand Extended Addresses (EAs) are generated in the W register array. Concurrent operation with MCU instruction flow is not possible, though both the MCU ALU and DSP engine resources can be shared by all instructions in the instruction set.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

The DSP engine consists of the following components:

- High-speed, 33-bit by 33-bit multiplier
- Barrel shifter
- 72-bit adder/subtractor
- Two target Accumulator registers
- Rounding logic with selectable modes
- Saturation logic with selectable modes

Data input to the DSP engine is derived from one of the following sources:

- Directly from the W array for dual source operand DSP instructions. Data values fetched via the X and Y memory data buses.
- From the X memory data bus for all other DSP instructions.

Data output from the DSP engine is written to one of the following destinations:

- The target accumulator, as defined by the DSP instruction being executed.
- The X memory data bus to any location in the data memory address space.

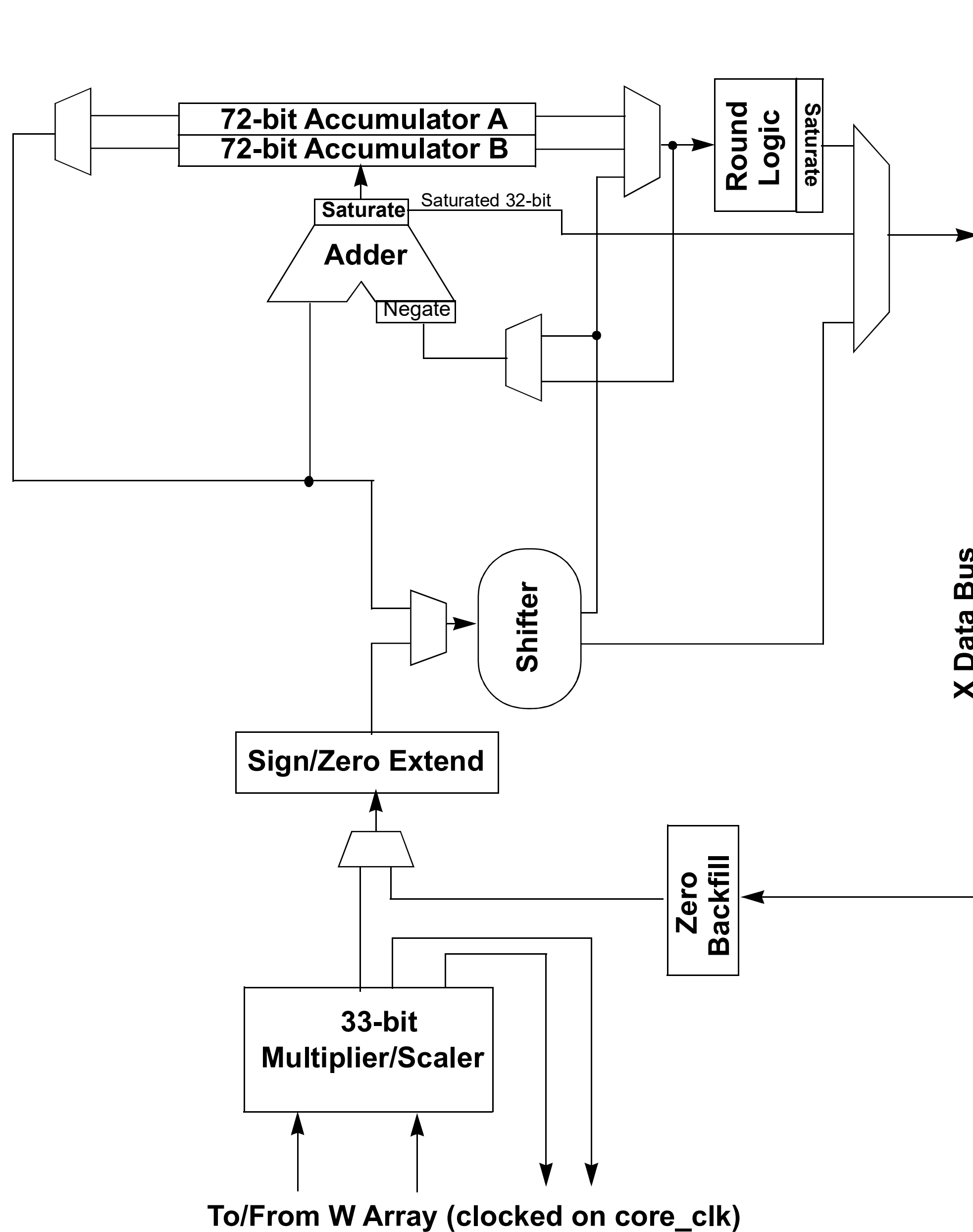
The DSP engine can perform inherent accumulator-to-accumulator operations that require no additional data.

The MCU shift and multiply instructions use the DSP engine hardware to obtain their results. The X memory data bus is used for data reads and writes in these operations.

Figure 3-9 illustrates a block diagram of the DSP engine.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-9. DSP Engine Block Diagram



3.3.12.1 Data Accumulators

Two 72-bit data accumulators, ACCA and ACCB, are the Result registers for the DSP instructions listed in 3.3.12.2.1. DSP Multiply Instructions. Each accumulator is not memory-mapped and is referred as these three registers, where 'x' denotes the particular accumulator:

- ACCxL: ACCx[31:0]
- ACCxH: ACCx[63:32]
- ACCxU: ACCx[71:64]

For fractional operations that use the accumulators, the radix point is located to the right of bit 31. The range of fractional values that can be stored in each accumulator is -256 to +(256 - 2**-63).

© 2023-2024 Microchip Technology Inc. and its subsidiaries

For integer operations that use the accumulators, the radix point is located to the right of bit 0. The range of integer values that can be stored in each accumulator is -0x80_00000000_00000000 to 0x7F_FFFF_FFFF_FFFF_FFFF.

3.3.12.2 Multiplier

The dsPIC33A devices feature a 33-bit-by-33-bit multiplier shared by both the MCU ALU and the DSP engine. The multiplier is capable of signed, unsigned or mixed-sign operation and supports either 9.31 fractional (Q.31) or 64-bit integer results.

The multiplier takes in 32-bit input data and converts the data to 33 bits. Signed operands to the multiplier are sign-extended. Unsigned input operands are zero-extended. The internal 33-bit representation of data in the multiplier allows correct execution of mixed-sign and unsigned 32-bit by 32-bit multiplication operations.

The representation of data in hardware for Integer and Fractional Multiplier modes is as follows:

- Integer data is inherently represented as a signed two's complement value, where the Most Significant bit (MSb) is defined as a Sign bit. Generally speaking, the range of an N-bit two's complement integer is $-2^{(N-1)}$ to $2^{(N-1)}-1$.
- Fractional data is represented as a two's complement fraction, where the MSb is defined as a Sign bit and the radix point is implied to lie just after the Sign bit (Q.X format). The range of an N-bit two's complement fraction with this implied radix point is -1.0 to $(1 – 2^{(1-N)})$.

The range of data in both Integer and Fractional modes is listed in Table 3-3. Figure 3-10 and Figure 3-11 illustrate how the multiplier hardware interprets data in Integer and Fractional modes.

The Integer or Fractional Multiplier Mode Select (IF) bit (CORCON[0]) determines integer/ fractional operation for the instructions listed in Table 3-4. The IF bit does not affect MCU multiply instructions listed in Table 3-5, which are always integer operations. The multiplier scales the result one bit to the left for fractional operation. The LSb of the result is always cleared. The multiplier defaults to Fractional mode for DSP operations at a device Reset.

Table 3-3. dsPIC33A Data Ranges

Register Size	Integer Range	Fraction Range	Fraction Resolution
16-Bit	-32768 to 32767	-1.0 to $(1.0 – 2^{-15})$ (Q1.15 Format)	3.052×10^{-5}
32-Bit	-2,147,483,648to 2,147,483,647	-1.0 to $(1.0 – 2^{-31})$ (Q1.31 Format)	4.657×10^{-10}
64-Bit	-9.223372037e18 to 9.223372037e18	-1.0 to $(1.0 – 2^{-63})$ (Q.1.63 Format)	1.08420×10^{-19}
72-Bit	-2.361183241e21 to 2.361183241e21	-256.0 to $(256.0 – 2^{-63})$ (Q.9.63 Format with 8 Guard bits)	1.08420×10^{-19}

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-10. Integer and Fractional Representation of 0x40000001

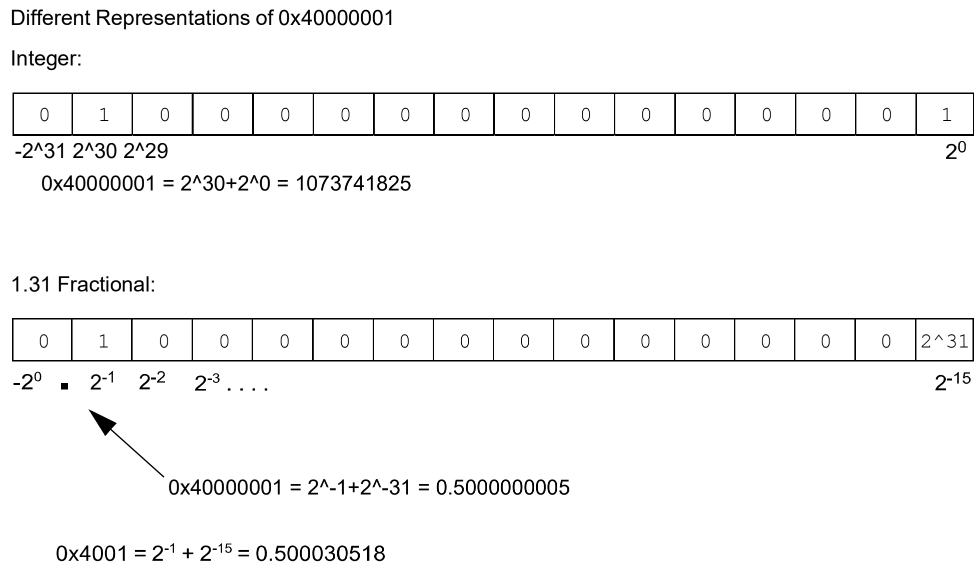
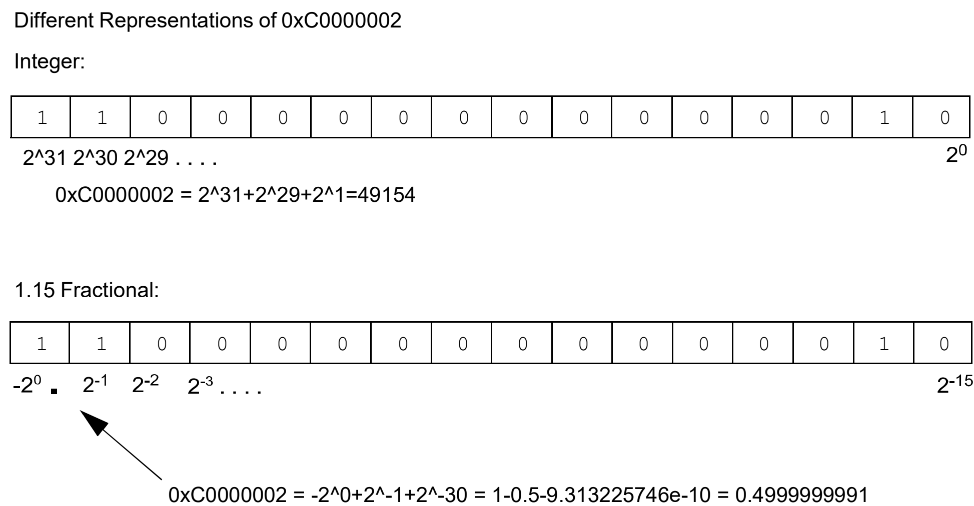


Figure 3-11. Integer and Fractional Representation of 0xC0000002



3.3.12.2.1 DSP Multiply Instructions

The DSP instructions that use the multiplier are summarized in Table 3-4.

Table 3-4. DSP Instructions that Use the Multiplier

DSP Instruction[1]	Description	Algebraic Equivalent
`MAC`	Multiply and Add to Accumulator or Square and Add to Accumulator	$a = a + b * c$ $a = a + b2$
`MSC`	Multiply and Subtract from Accumulator	$a = a - b * c$
`MPY`	Multiply	$a = b * c$

Note:

1. DSP instructions using the multiplier can operate in signed or unsigned Fractional (1.15/1.31) or Integer modes.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

...........continued

DSP Instruction[1]	Description	Algebraic Equivalent
MPYN	Multiply and Negate Result	$a = -b * c$
SQR	Square to Accumulator	$a = b ^ 2$
SQRAC	Square and Accumulate	$a = a + (b ^ 2)$
ED	Partial Euclidean Distance	$a = (b – c)^2$
EDAC	Add Partial Euclidean Distance to the Accumulator	$a = a + (b – c)^2$

Note:
1. DSP instructions using the multiplier can operate in signed or unsigned Fractional (1.15/1.31) or Integer modes.

The DSP Multiplier Unsigned/Signed Control (US) bits (CORCON[12]) determine whether the DSP multiply instructions are signed (default) or unsigned. The US bits do not influence the MCU multiply instructions, which have specific instructions for signed or unsigned operation. If the USx bits are set to '01', the input operands for instructions shown in Table 3-4 are considered as unsigned values, which are always zero-extended into the 33rd bit of the multiplier value. If the USx bits are set to '00', the operands are sign-extended.

If the USx bits (CORCON[13:12]) are set to '10', the operands for the instructions listed above are considered as unsigned values. The result is zero-extended prior to any operation with the accumulator (which will always effectively be signed).

3.3.12.2.2 MCU Multiply Instructions

The same multiplier supports the MCU multiply instructions, which include integer, 32-bit signed, unsigned and mixed-sign multiplies, as shown below. All multiplications performed by the MUL instruction produce integer results. The MUL instruction can be directed to use byte or word- sized operands. Byte input operands produce a 16-bit result and word input operands produce either a 16-bit result or a 32-bit result, either to the specified register(s) in the W array or to an accumulator. Word input operands produce a 32-bit result and word input operands produce either a 32-bit result or a 64-bit result, either to the specified register(s) in the W array or to an accumulator.

Table 3-5. MCU Instructions that Utilize the Multiplier

MCU Instruction[1]	Description
MUL/MUL.UU	Multiply two unsigned integers and generate 64-bit results.
MUL.SS	Multiply two signed integers and generate 64-bit results.
MUL.SU/MUL.US	Multiply a signed integer with an unsigned integer and generate 64-bit results.
MULD.UU	Multiply two unsigned integers and generate 72-bit results.
MULD.SS	Multiply two signed integers and generate 72-bit results.
MULD.SU/ MULD.US	Multiply a signed integer with an unsigned integer and generate a 72-bit result.
MULW.UU	Multiply two unsigned integers and generate 32-bit results.
MULW.SS	Multiply two signed integers and generate 32-bit results.
MULW.SU/MULW.US	Multiply a signed integer with an unsigned integer and generate a 32-bit result.
MULB	Multiply two unsigned 8-bit integers and generate 16-bit results.
MULW	Multiply two unsigned 16-bit integers and generate 32-bit results.
MULL	Multiply two unsigned 32-bit integers and generate 32-bit results.

Note:
1. MCU instructions using the multiplier operate only in Integer mode.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.3.12.3 Data Accumulator Adder/Subtractor

The data accumulators have a 72-bit adder/subtractor with automatic sign extension or zero extension logic for the multiplier result. It can select one of two accumulators (A or B) as its pre-accumulation source and post-accumulation destination. For the `ADD` (accumulator) and `LAC` instructions, the data to be accumulated or loaded can optionally be scaled via the barrel shifter prior to accumulation.

The 72-bit adder/subtractor can optionally negate one of its operand inputs to change the sign of the result (without changing the operands). The negate is used during multiply and subtract (`MSC`) or multiply and negate (`MPYN`) operations.

The 72-bit adder/subtractor has an additional saturation block that controls accumulator data saturation, if enabled.

3.3.12.3.1 Accumulator Status Bits

Six STATUS Register bits that support saturation and overflow are located in the CPU STATUS Register (SR) and are listed in Table 3-6.

Table 3-6. Accumulator Overflow and Saturation Status Bits

Status Bit (SR Location)	Description
OA ([15])	Accumulator A overflowed into guard bits (ACCA[71:63])
OB ([14])	Accumulator B overflowed into guard bits (ACCB[71:63])
SA ([13])	ACCA saturated (bit 63 overflow and saturation) or ACCA overflowed into guard bits and saturated (bit 71 overflow and saturation)
SB ([12])	ACCB saturated (bit 63 overflow and saturation) or ACCB overflowed into guard bits and saturated (bit 71 overflow and saturation)
OAB ([11])	OA logically ORed with OB, clearing OAB clears both OA and OB
SAB ([10])	SA logically ORed with SB, clearing SAB clears both SA and SB

The OA and OB bits are modified each time data passes through the accumulator add/subtract logic. When set, they indicate that the most recent operation has overflowed into the accumulator guard bits (ACCx[71:64]). This type of overflow is not catastrophic; the guard bits preserve the accumulator data. The OAB Status bit is the logically OR value of OA and OB.

The OA and OB bits, when set, can optionally generate an arithmetic error trap. The trap is enabled by setting the corresponding Overflow Trap Flag Enable bit (OVATE or OVBTE) in Interrupt Control Register 4 (INTCON4[10:9]) in the interrupt controller. The trap event allows the user to take immediate corrective action, if desired.

The SA and SB bits can be set each time data passes through the accumulator saturation logic. Once set, these bits remain set until cleared by the user application. The SAB Status bit indicates the logical OR value of SA and SB. When set, these bits indicate that the accumulator has overflowed its maximum range (bit 63 for 64-bit saturation or bit 71 for 72-bit saturation) and are saturated (if saturation is enabled).

When saturation is not enabled, the SA and SB bits indicate that a catastrophic overflow has occurred (the sign of the accumulator has been destroyed). If the Catastrophic Overflow Trap Enable (COVTE) bit (INTCON4[8]) is set, SA and SB bits will generate an arithmetic error trap when saturation is disabled. The SA and SB bits can be set in software, enabling efficient context state switching.

3.3.12.3.2 Saturation And Overflow Modes

The dsPIC33A CPU supports three Saturation and Overflow modes.

- **Accumulator 71-Bit Saturation**
  In this mode, the saturation logic loads the maximally positive 9.63 value (0x7F_FFFF_FFFF_FFFF_FFFF) or maximally negative 9.63 value (0x80_0000_0000_0000_0000) into

© 2023-2024 Microchip Technology Inc. and its subsidiaries

the target accumulator. The SA or SB bit is set and remains set until cleared by the user application. This Saturation mode is useful for extending the dynamic range of the accumulator.

To configure for this mode of saturation, set the Accumulator Saturation Mode Select (ACCSAT) bit (CORCON[4]). Additionally, set the ACCA Saturation Enable (SATA) bit (CORCON[7] and/or the ACCB Saturation Enable (SATB) bit (CORCON[6]) to enable accumulator saturation.

- **Accumulator 63-Bit Saturation**
  In this mode, the saturation logic loads the maximally positive 1.63 value (0x00_7FFF_FFFF_FFFF_FFFF) or maximally negative 1.63 value (0xFF_8000_0000_0000_0000) into the target accumulator. The SA or SB bit is set and remains set until cleared by the user. When this Saturation mode is in effect, the guard bits, 64 through 71, are not used except for sign extension of the accumulator value. Consequently, the OA, OB or OAB bits in SR are never set.

  To configure for this mode of overflow and saturation, the ACCSAT (CORCON[4]) bit must be cleared. Additionally, the SATA (CORCON[7]) and/or SATB (CORCON[6]) bits must be set to enable accumulator saturation.
- **Accumulator Catastrophic Overflow**
  If the SATA (CORCON[7]) and/or SATB (CORCON[6]) bits are not set, then no saturation operation is performed on the accumulator, and the accumulator is allowed to overflow all the way up to bit 71 (destroying its sign). If the Catastrophic Overflow Trap Enable (COVTE) bit (INTCON4[8] in the interrupt controller) is set, a catastrophic overflow initiates an arithmetic error trap.

Accumulator saturation and overflow detection can only result from the execution of a DSP instruction that modifies one of the two accumulators via the 72-bit DSP ALU. Saturation and overflow detection do not take place when the accumulators are accessed via the MCU class of instructions. Furthermore, the Accumulator Status bits shown in Table 3-6 are not modified. However, the MCU Status bits (Z, N, C, OV, DC) will be modified, depending on the MCU instruction that accesses the accumulator.

3.3.12.3.3 Data Space Write Saturation

In addition to adder/subtractor saturation, writes to data space can be saturated without affecting the contents of the source accumulator. This feature allows data to be limited, while not sacrificing the dynamic range of the accumulator during intermediate calculation stages. Data space write saturation is enabled by setting the data space write from the DSP Engine Saturation Enable (SATDW) Control bit (CORCON[5]). Data space write saturation is enabled by default at a device Reset.

The data space write saturation feature works with the `SAC` and `SACR` instructions. The value held in the accumulator is never modified when these instructions are executed. The hardware takes the following steps to obtain the saturated write result:

1. The read data is scaled based upon the arithmetic shift value specified in the instruction.
2. The scaled data is rounded (`SACR` only).
3. For Word mode instruction, scaled/rounded value is saturated to a 16-bit result based on the value of the guard bits. For data values greater than 0x007FFF, the data written to memory is saturated to the maximum positive 1.15 value, 0x7FFF. For input data less than 0xFF8000, data written to memory is saturated to the maximum negative 1.15 value, 0x8000. Similarly, the data written to memory is saturated to maximum positive/negative 1.31 value for Long Word mode operation.

3.3.12.3.4 Accumulator Write Back

The `MAC` and `MSC` instructions can optionally write a rounded version of the accumulator that is not the target of the current operation into data space memory. The write is performed across the X-bus into the combined X and Y address space. This accumulator write-back feature is beneficial in certain algorithms, such as FFT and LMS filters.

Two addressing modes are supported by the accumulator write-back hardware:

© 2023-2024 Microchip Technology Inc. and its subsidiaries

- W0, W1, W2, W3 or W13, Register Direct: The rounded contents of the non-target accumulator are written into the destination register as a 1.15 (Word mode) or 1.31 (Long Word mode) fractional result.
- [W13++] or [W15++], Register Indirect with Post-Increment: The rounded contents of the non-target accumulator are written into the address pointed to by W13 or W15 as a 1.15 (Word mode) or 1.31 (Long Word mode) fraction. W13 or W15 is then incremented by 2/4 depending on selected Word/Long Word mode. [W15++] is equivalent to a push onto the system stack.

3.3.12.4 Round Logic

The round logic can perform a conventional (biased) or convergent (unbiased) round function during an accumulator write (store). The Round mode is determined by the state of the Rounding Mode Select (RND) bit (CORCON[1]). It generates a 16-bit 1.15 or 32-bit 1.31 data value, which is passed to the data space write saturation logic. If rounding is not indicated by the instruction, a truncated 1.15 or 1.31 data value is stored.

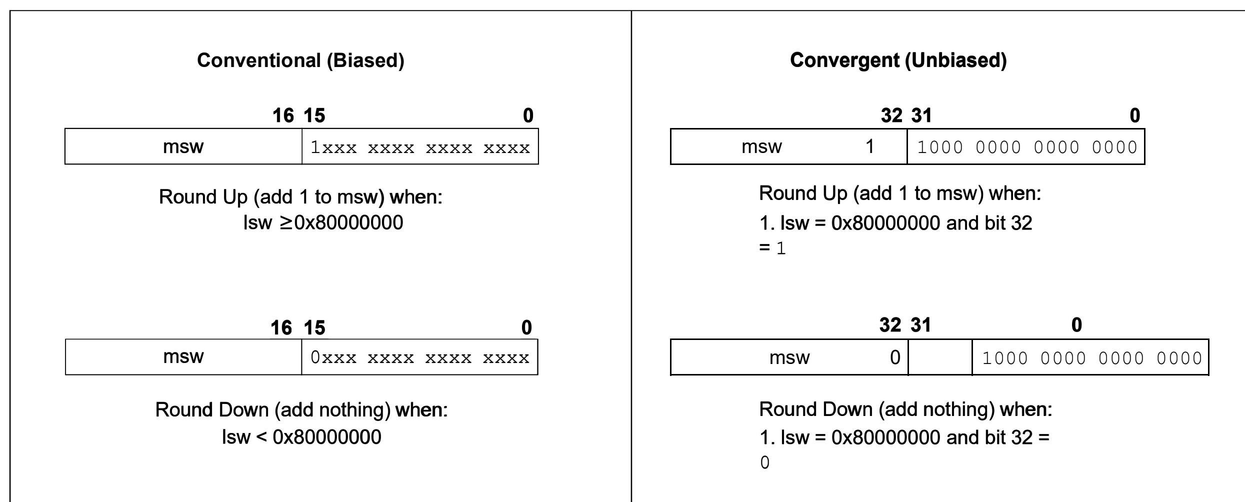
The two Rounding modes are shown in Figure 3-12. Conventional rounding takes bit 31 of the accumulator, zero-extends it and adds it to the most significant word (msw), excluding the guard or overflow bits (bits 32 through 63). If the least significant word (lsw) of the accumulator is between 0x80000000 and 0xFFFFFFFF (0x80000000 included), the msw is incremented. If the lsw of the accumulator is between 0x0000 and 0x7FFFFFFF, the msw remains unchanged. A consequence of this algorithm is that over a succession of random rounding operations, the value tends to be biased slightly positive.

Convergent (or unbiased) rounding operates in the same manner as conventional rounding except when the lsw equals 0x80000000. If this is the case, the LSb of the msw (bit 16 of the accumulator) is examined. If it is '1', the msw is incremented. If it is '0', the msw is not modified. Assuming that bit 16 is effectively random in nature, this scheme removes any rounding bias that may accumulate.

The SAC and SACR instructions store either a truncated (SAC) or rounded (SACR) version of the contents of the target accumulator to data memory via the X-bus (subject to data saturation).

For the MAC class of instructions, the accumulator write-back data path is always subject to rounding. An overflow that occurs as a consequence of a rounding operation will also be subject to saturation.

Figure 3-12. Conventional and Convergent Rounding Modes



3.3.12.5 Barrel Shifter

The barrel shifter can perform up to a 32-bit arithmetic right shift, or up to a 32-bit left shift, in a single cycle. DSP or MCU instructions can use the barrel shifter for multibit shifts.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

The shifter requires a signed binary value to determine both the magnitude (number of bits) and direction of the shift operation:

- A positive value shifts the operand right
- A negative value shifts the operand left
- A value of '0' does not modify the operand

The barrel shifter is 72 bits wide to accommodate the width of the accumulators. A 72-bit output result is provided for DSP shift operations, and a 32-bit result is provided for MCU shift operations.

Table 3-7 provides a summary of instructions that use the barrel shifter.

Table 3-7. Instructions that Use the DSP Engine Barrel Shifter

Instruction	Description
ASR	Arithmetic multibit right shift of data memory location
LSR	Logical multibit right shift of data memory location
SL	Multibit shift left of data memory location
SAC	Store DSP accumulator with optional shift
SFTAC	Shift DSP accumulator

3.3.12.6 DSP Engine Mode Selection

These operational characteristics of the DSP engine, discussed in previous sections, can be selected through the CPU Core Configuration register (CORCON):

- Fractional or integer multiply operation
- Conventional or convergent rounding
- Automatic saturation on/off for ACCA
- Automatic saturation on/off for ACCB
- Automatic saturation on/off for writes to data memory
- Accumulator Saturation mode selection

3.3.12.7 DSP Engine Trap Events

Arithmetic error traps that can be generated for handling exceptions in the DSP engine are selected through the Interrupt Control Register 4 (INTCON4). These are:

- Trap on ACCA overflow enable using OVATE (INTCON4[21])
- Trap on ACCB overflow enable using OVBTE (INTCON4[20])
- Trap on catastrophic ACCA and/or ACCB overflow enable using COVTE (INTCON4[19]). Occurrence of the traps is indicated by these error status bits:
  - OVAERR (INTCON4[5])
  - OVBERR (INTCON4[4])
  - COVAERR (INTCON4[3])
  - COVBERR (INTCON4[2])

  An arithmetic error trap is also generated when the user application attempts to shift a value beyond the maximum allowable range (±32 bits) using the SFTAC instruction. This trap source cannot be disabled and is indicated by the Shift Accumulator Error Status (SFTACERR) bit (INTCON4[1] in the interrupt controller). The instruction will execute, but the results of the shift are not written to the target accumulator.

3.3.13 Divide Support

The dsPIC33A CPU supports the following types of division operations:

© 2023-2024 Microchip Technology Inc. and its subsidiaries

- DIVF: 16/16 signed fractional divide
- DIVF: 32/16 signed fractional divide
- DIVF.L: 32/32 signed fractional divide
- DIV.SL: 32/32 signed divide
- DIV.UL: 32/32 unsigned divide
- DIV.S: 32/16 signed divide
- DIV.U: 32/16 unsigned divide
- DIV.S: 16/16 signed divide
- DIV.U: 16/16 unsigned divide

The quotient for all divide instructions can placed in any Working register, Wm. The remainder is placed in W(m+1). The 32/16-bit divisor can be located in any W register. A 32/16-bit dividend can be located in any W register. The integer 16/16 divide instructions will either zero or sign extend the least significant dividend word into the most significant dividend word during the first iteration to create a 32-bit dividend.

All 16-bit/16-bit and 32-bit/16-bit divide instructions are iterative operations and must be executed six times within a REPEAT loop. All 32-bit/32-bit divide instructions are iterative operations and must be executed ten times within a REPEAT loop.

The developer is responsible for programming the REPEAT instruction. A complete divide operation takes seven or eleven instruction cycles to execute.

The divide flow is interruptible, just like any other REPEAT loop. All data is restored into the respective data registers after each iteration of the loop, so the user application is responsible for saving the appropriate W registers in the ISR. Although they are important to the divide hardware, the intermediate values in the W registers have no meaning to the user application. The divide instructions must be executed seven or eleven times in a REPEAT loop to produce a meaningful result.

A divide-by-zero error generates a math error trap. This condition is indicated by the Arithmetic Error Status (DIV0ERR) bit (INTCON4[0] in the interrupt controller).

3.3.14 Instruction Flow Types

Most instructions in the dsPIC33A architecture occupy a single word of program memory and execute in a single cycle. However, some instructions take two or more instruction cycles to execute. Consequently, there are seven different types of instruction flow in the dsPIC® DSC architecture.

3.3.15 Loop Constructs

The dsPIC33A CPU supports two REPEAT constructs to provide unconditional automatic program loop control. The REPEAT instruction implements a single instruction program loop. REPEAT instructions use control bits within the CPU STATUS Register (SR) to temporarily modify CPU operation.

3.3.15.1 REPEAT Loop Construct

The REPEAT instruction causes the instruction that follows it to be repeated a specified number of times. A literal value contained in the instruction, or a value in one of the W registers, can be used to specify the REPEAT count value. The W register option enables the loop count to be a software variable.

An instruction in a REPEAT loop is executed at least once. The number of iterations for a REPEAT loop is the 20-bit literal value + 1 or Wn + 1. The syntax for the two forms is shown in 3.3.15.1. REPEAT Loop Construct.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Example 3-3. `REPEAT` Loop Construct

```
;    Using a literal value as a counter
REPEAT #lit20    ; RCOUNT <-- lit20
(Valid target Instruction)
;
;    Using a W register as a counter
REPEAT Wn    ; RCOUNT <-- Wn
(Valid target Instruction)
```

3.3.15.1.1 REPEAT Operation

The loop count for `REPEAT` operations is held in the 32-bit Repeat Loop Counter register (RCOUNT), which is memory-mapped. RCOUNT is initialized by the `REPEAT` instruction. The `REPEAT` instruction sets the `REPEAT` Loop Active (RA) Status bit (SR[4]) to '`1`' if the RCOUNT is a non-zero value.

RA is a read-only bit and cannot be modified through software. For `REPEAT` loop count values greater than '`0`', the Program Counter is not incremented. Furthermore, Program Counter increments are inhibited until RCOUNT = `0`.

For a loop count value equal to '`0`', `REPEAT` has the effect of a `NOP` and the RA (SR[4]) bit is not set. The `REPEAT` loop is essentially disabled before it begins, allowing the target instruction to execute only once while pre-fetching the subsequent instruction (i.e., normal execution flow).

Note: The instruction immediately following the `REPEAT` instruction (i.e., the target instruction) is always executed at least one time and it is always executed one time more than the value specified in the 20-bit literal or the W register operand.

3.3.15.1.2 Interrupting a REPEAT Loop

A `REPEAT` instruction loop can be interrupted at any time.

The state of the RA bit is preserved on the stack during exception processing to enable the user application to execute further `REPEAT` loops from within any number of nested interrupts. After SR is stacked, the RA Status bit is cleared to restore normal execution flow within the ISR.

Note: If a `REPEAT` loop has been interrupted, and an ISR is being processed, the user application must stack the Repeat Count register (RCOUNT) before it executes another `REPEAT` instruction within an ISR.
If a `REPEAT` instruction is used within an ISR, the user application must unstack the RCOUNT register before it executes the `RETFIE` instruction.

Returning into a `REPEAT` loop from an ISR using the `RETFIE` instruction requires no special handling. RETFIE pops the PC and that becomes the address of the next instruction to be fetched in its F-stage. The RETFIE instruction is "padded" with FNOPs (2) so the target instruction of the RETFIE PFC can execute as normal.

Early Termination of a REPEAT Loop

An interrupted `REPEAT` loop can be terminated earlier than normal in the ISR by clearing the RCOUNT register in software.

3.3.15.1.3 Restrictions on the REPEAT Instruction

Any instruction can immediately follow a `REPEAT` except for the following:

- Program Flow Control instructions (any branch, compare and skip, subroutine calls, returns, etc.)
- Another `REPEAT` or `DTB` instruction
- `DISICTL`, `ULNK`, `LNK`, `PWRSAV` or `RESET` instruction
  - `MOV.D` instruction
    Note: Some instructions and/or Instruction Addressing modes can be executed within a `REPEAT` loop, but it might not make sense to repeat all instructions.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.3.16 Data Space Address Generation Units (AGUs)

dsPIC33AK128MC106 family devices contain three independent address generator units. The X RAGU and X WAGU support byte (.b), word (.w) and long (.l) word sized data space reads and writes, respectively, for MCU instructions, and word or long word reads and writes for DSP instructions. The Y AGU supports word and long word sized data reads for the DSP MAC-class of instructions only. The AGUs are each capable of supporting two types of data addressing:

- Linear Addressing
- Modulo (circular) Addressing

In addition, the X WAGU can support Bit-Reversed Addressing.

Linear and Modulo Data Addressing modes can be applied to any address within the unified address space. Although Bit- Reversed Addressing will work with any EA calculation, by definition it is only applicable to data space.

Data space memory is organized as 32-bit words; all Effective Addresses (EAs) point to bytes. Instructions can thus access any byte or aligned word (data words at an even byte address) or aligned long word (data words at an even 32-bit word address).

Misaligned accesses are not supported, and if attempted they will initiate an address error trap. The least significant 2 bits of the EA is used to determine the byte or upper/lower 16-bit word access. EA[0] will always be 1’b0 for word accesses, and EA[1:0] will always be 2’b00 for long word accesses.

SFRs and RAM support byte, word, and double word read or write operations.

When executing instructions that require just one source operand to be fetched from (and one result to be written back to) data space, the X RAGU and X WAGU are used to calculate the EAs of the source and destination, respectively. The AGUs can generate an address to point to anywhere in the 16 Mbyte address space. They support all MCU addressing modes and Modulo Addressing for low overhead circular buffers. The X WAGU also supports Bit-Reversed Addressing to facilitate FFT data reorganization.

When executing instructions which require two source operands to be concurrently fetched (i.e. the MAC class of DSP instructions), both the X RAGU and Y AGU are used simultaneously.

The dsPIC33AK128MC106 device family contains an X AGU and a Y AGU for generating data memory addresses. Both X and Y AGUs can generate any EA within the available data memory range. However, EAs that are outside of the physical memory provided return all zeros for data reads and writes to those locations and therefore have no effect. Furthermore, an address error trap will be generated. For more information on address error traps, refer to 10. Interrupt Controller.

3.3.16.1 Address Generation Units and DSP Class Instructions

The Y AGU and Y memory data path are used in concert with the X RAGU by the DSP class of instructions to provide two concurrent data read paths. For example, the MAC instruction can simultaneously fetch two operands to be used in the next multiplication.

DSP class of instructions may use any W-reg (except W15) for either X or Y address space accesses, unlike previous dsPIC devices. Any data write performed by a DSP class instruction takes place in the combined X and Y data space and the write occurs across the X-bus. Consequently, the write can be to any address regardless of where the EA is directed.

The Y AGU only supports Post-Modification Addressing modes associated with the DSP class of instructions. The Y AGU also supports Modulo Addressing for automated circular buffers. All other (MCU) class instructions can access the Y data address space through the X AGU when it is regarded as part of the composite linear space.

3.3.16.2 Data Alignment

The ISA supports long word (32-bit), word (16-bit) and byte (8-bit) sized operations. Data is aligned in data memory and registers as long words, but all data space EAs resolve to bytes. Data word and byte reads will read the complete 32-bit word that contains the word or byte, using the LSbs of any

© 2023-2024 Microchip Technology Inc. and its subsidiaries

EA to determine which word or byte to select within the CPU. The selected word or byte is placed onto the lsw or byte of the X data path (no byte accesses are possible from the Y data path as the MAC-class of instruction can only fetch words or long words). That is, data memory and registers are organized as four parallel byte-wide entities with a shared (long word) address decode but separate write lines. Data byte writes will only write to the corresponding side of the array or register which matches the byte address.

Note: Byte reads will always read the entire word, so mechanisms to clear or set peripheral status bits when read (e.g. quick flag clearing mechanisms) are not allowed.

As a consequence of this byte addressability, all EA calculations must be scaled to step through long word aligned memory. For example, the core must recognize that post modified register indirect addressing mode, [Ws]+=1, will result in a value of Ws+1 for byte operations, Ws+2 for word operations, and Ws+4 for long word operations.

Misaligned word or long word accesses are not supported. For word accesses, the LSb of the EA must be 1’b0. For long word accesses, the least significant 2 bits of the EA must be 2’b00. Therefore, care must be taken when mixing operations of different data widths or translating from 16-bit dsPIC code. Should a misaligned read or write be attempted, an address error trap will be forced. If the fault occurs during a read access, the read will be allowed to complete. If the fault occurs during a write access, the write will also be allowed to complete (inhibiting the write would have been possible but inconsistent with other situations where an errant write could not be inhibited). In both cases, the address error trap will be asserted. The next instruction (already pre-fetched and underway) will be executed while the exception is arbitrated and acknowledged. When this instruction completes, the trap will then be taken, allowing the system and/or user to examine the machine state subsequent to execution of the address fault.

Note: Byte and word ALU operations can produce results in excess of a byte or a word. However, to maintain 16-bit dsPIC backwards code compatibility, the ALU result destination write from all operations maintains the same width as that of the source operands (i.e. MSbs of the destination are not modified) and the SR is updated based only upon the state of the result data.

A sign extend (SE) instruction is provided to allow users to translate 8-bit to16-bit, and 16-bit to 32-bit signed values. Alternatively, for unsigned data, users can clear the MS portion of any W register through executing a byte or word zero extend (ZE).

Note: Care must be taken when mixing byte and word size instructions/operands.

Although most instructions are capable of operating on long word, word or byte data sizes, it should be noted that the DSP and some other instructions operate on long word or word sized data only.

Figure 3-13. Data Alignment

31	23	15	7 0	Address
Byte 3	Byte2	Byte1	Byte 0	24’h00_0000
Byte 7	Byte6	Byte5	Byte 4	24’h00_0004
Byte 11	Byte10	Byte9	Byte 8	24’h00_0008

3.3.17 MAC Instructions

The dual source operand DSP instructions (`ED`, `EDAC`, `MAC`, `MPY`, `MPYN`, `SQR`, `SQRAC`, `MSC`, `SQRSC` and `SQRN`), also referred to as `MAC` instructions, use a simplified set of addressing modes to allow the user application to effectively manipulate the Data Pointers through register indirect tables.

These instructions support various addressing modes for X and Y data bus, where W-registers accessing these data buses may be any W-reg (except W15) for either X or Y address space accesses. Pre or post modification values are scaled based upon instruction operand width. The MAC-class instruction also supports the ability to write the contents of the accumulator that is not being used

© 2023-2024 Microchip Technology Inc. and its subsidiaries

as the instruction result destination to a memory or W-register as defined by the instruction with a restricted set of addressing modes. This is referred to as the Accumulator Write Back (AWB).

Note:
AWB is only intended for use when the DSP engine is operating in fractional data mode. It can only write the MS portion of the target accumulator fractional value.

MAC-class instructions are no longer tied to operand reads of X and Y address space. Operands may both be sourced from X-space, resulting in reading the operand data sequentially rather than concurrently. This will add an additional RAM data fetch delay (typically one cycle) to all such instructions.

3.3.18 Modulo Addressing

Modulo Addressing mode is a method of providing an automated means to support circular data buffers using hardware. The objective is to remove the need for software to perform data address boundary checks when executing tightly looped code, as is typical in many DSP algorithms.

Modulo Addressing can operate in either Data or Program Space (since the Data Pointer mechanism is essentially the same for both). One circular buffer can be supported in each of the X (which also provides the pointers into Program Space) and Y Data Spaces. Modulo Addressing can operate on any W Register Pointer. However, it is not advisable to use W14 or W15 for Modulo Addressing since these two registers are used as the Stack Frame Pointer and Stack Pointer, respectively.

In general, any particular circular buffer can be configured to operate in only one direction, as there are certain restrictions on the buffer start address (for incrementing buffers) or end address (for decrementing buffers) based upon the direction of the buffer.

The only exception to the usage restrictions is for buffers that have a power-of-two length. As these buffers satisfy the start and end address criteria, they can operate in a Bidirectional mode (that is, address boundary checks are performed on both the lower and upper address boundaries).

3.3.18.1 Start and End Address

The Modulo Addressing scheme requires that a starting and ending address be specified and loaded into the 24-bit Modulo Buffer Address registers: XMODSRT, XMODEND, YMODSRT and YMODEND.

Note: Y space Modulo Addressing EA calculations assume word-sized data (LSb of every EA is always clear).

The length of a circular buffer is not directly specified. It is determined by the difference between the corresponding start and end addresses. The maximum possible length of the circular buffer is 32K words (64 Kbytes).

3.3.18.2 W Address Register Selection

The Modulo and Bit-Reversed Addressing Control register, MODCON[15:0], contains enable flags, as well as a W register field to specify the W Address registers. The XWM and YWM fields select the registers that operate with Modulo Addressing:

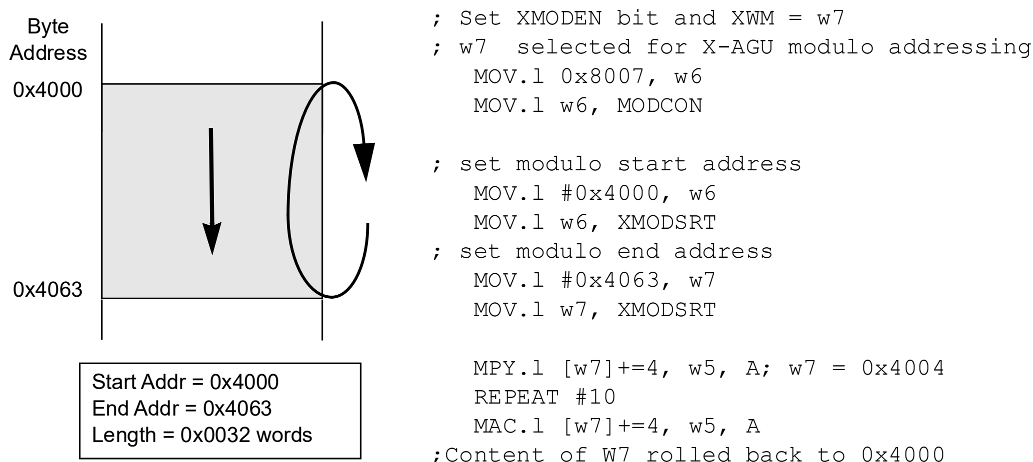
- If XWM = `1111`, X RAGU and X WAGU Modulo Addressing is disabled
- If YWM = `1111`, Y AGU Modulo Addressing is disabled

The X Address Space Pointer W (XWM) register, to which Modulo Addressing is to be applied, is stored in MODCON[3:0]. Modulo Addressing is enabled for X Data Space when XWM is set to any value other than '`1111`' and the XMODEN bit is set (MODCON[15]).

The Y Address Space Pointer W (YWM) register, to which Modulo Addressing is to be applied, is stored in MODCON[7:4]. Modulo Addressing is enabled for Y Data Space when YWM is set to any value other than '`1111`' and the YMODEN bit is set (MODCON[14]).

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-14. Modulo Addressing Operation Example



3.3.18.3 Bit-Reversed Addressing

Bit-Reversed Addressing mode is intended to simplify data reordering for radix-2 FFT algorithms. It is supported by the X AGU for data writes only.

The modifier, which can be a constant value or register contents, is regarded as having its bit order reversed. The address source and destination are kept in normal order. Thus, the only operand requiring reversal is the modifier.

3.3.18.3.1 Bit-Reversed Addressing Implementation

Bit-Reversed Addressing can only be enabled through the use of the movr.(w/l) instruction. This type of addressing is effective when used with pre-modified or post-modified destination addressing. The destination Bit-Reversed Addressing modifier is sourced from XBREV.XB[14:0].

If the length of a bit-reversed buffer is $M = 2^N$ bytes, the last ‘N’ bits of the data buffer start address must be zeros.

The XB[14:0] bits are the Bit-Reversed Addressing modifier, or ‘pivot point’, which is typically a constant. In the case of an FFT computation, its value is equal to half of the FFT data buffer size.

Note: All bit-reversed EA calculations assume either word-size (where the least significant bit of every Effective Address is always clear) or long word-size (where the two least significant bits of the Effective Address are always clear), based on the operation data width selected. The XB value is scaled accordingly to generate compatible (byte) addresses.

Bit-Reversed Addressing is only possible when using the MOVR instruction, and it can target a 16-bit or 32-bit sized data. MOVR instruction supports Register Indirect with Pre-Increment or Post-Increment Addressing and 16/32 bit-sized data writes. When Bit-Reversed Addressing is active, the W Address Pointer is always added to the address modifier (XB) and the offset associated with the Register Indirect Addressing mode is ignored. In addition, the LSb of each 16-bit address and the LS 2-bits of each 32-bit address, will always be zero for both source and destination EAs. The MOVR instruction also supports “in-place” data re-ordering (where only one data buffer is used for both source and destination), source and destination indirect addressing may use the same register

Note: Modulo Addressing and Bit-Reversed Addressing can be enabled simultaneously using the same W register, but the Bit-Reversed Addressing operation will always take precedence for data writes when enabled.

If Bit-Reversed Addressing has already been enabled by setting the BREN (XBREV[15]) bit, a write to the XBREV register should not be immediately followed by an indirect read operation using the W register that has been designated as the Bit-Reversed Pointer.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-15. Bit-Reversed Addressing Example

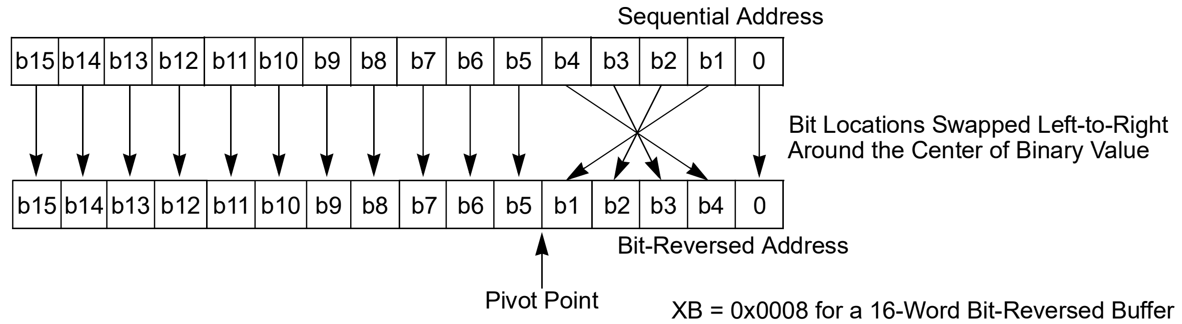


Table 3-8. Bit-Reversed Addressing Sequence (16-Entry)

Normal Address					Bit-Reversed Address				
A3	**A2**	**A1**	**A0**	**Decimal**	**A3**	**A2**	**A1**	**A0**	**Decimal**
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	8
0	0	1	0	2	0	1	0	0	4
0	0	1	1	3	1	1	0	0	12
0	1	0	0	4	0	0	1	0	2
0	1	0	1	5	1	0	1	0	10
0	1	1	0	6	0	1	1	0	6
0	1	1	1	7	1	1	1	0	14
1	0	0	0	8	0	0	0	1	1
1	0	0	1	9	1	0	0	1	9
1	0	1	0	10	0	1	0	1	5
1	0	1	1	11	1	1	0	1	13
1	1	0	0	12	0	0	1	1	3
1	1	0	1	13	1	0	1	1	11
1	1	1	0	14	0	1	1	1	7
1	1	1	1	15	1	1	1	1	15

Example 3-4. 32-Bit Data, Two Buffer Bit-Reversed Data Reordering Example

```
; Two buffer (input and output) bit reversed data re-order subroutine for 32-
bit (real)
; data values
;
; W0: Temp
; W1: Data table size N (long words)
; W8: Input data table pointer (natural order) initialized to start of table
; W9: Output data table pointer (bit reversed) initialized to start of table

        push.l w0
        mov.sl #_XBREV, w0
        lsr.l w1, #1, [w0] ; XBREV = N/2

        sub.l #1, w1
        repeat w1
        movr.l [w8++], [w9++] ; Move data from input to output buffer, then
                              ; bump natural order and bit reversed pointers
        pop.l w0
        return
```


© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.3.19 Address Register Dependencies

The dsPIC33A architecture supports a data space read (source) and a data space write (destination) for most MCU class instructions. The EA calculation by the AGU, and subsequent data space read or write, each take one instruction cycle to complete. This timing causes the data space read and write operations for each instruction to overlap.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.3.20 Multiplier

Using the high-speed, 33-bit x 33-bit multiplier, the ALU supports an unsigned, signed or mixed-sign operation in several MCU Multiplication modes:

- 32-bit x 32-bit signed
- 32-bit x 32-bit unsigned
- 32-bit signed x 5-bit (literal) unsigned
- 32-bit signed x 32-bit unsigned
- 32-bit unsigned x 5-bit (literal) unsigned
- 32-bit unsigned x 32-bit signed
- 16-bit unsigned x 16-bit unsigned

3.4 Prefetch Buffer Unit (PBU)

The Prefetch Buffer Unit (PBU) in the dsPIC33A core devices accelerates the interface between the dsPIC33A program Flash memory and the CPU instruction bus. The PBU can predictively prefetch the next sequential address and cache fetched program data that are the target of a CPU instruction fetch.

PBU in dsPIC33A core devices supports the following functions:

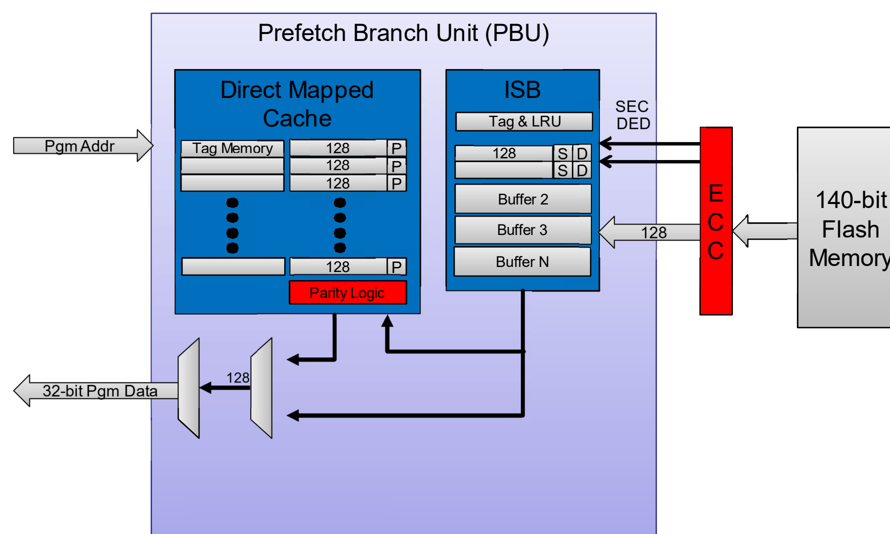
1. PBU accelerates the execution of linear program code flow.
2. As cache accelerates the execution of non-linear program flow changes (branches).

The PBU in the dsPIC33A core devices have the following features:

- Provides interface between Program Flash Memory (PFM) and CPU instruction bus
- Instruction Stream Buffers for prefetching and caching of linear PFM instruction flows
- Instruction Cache for caching of most frequently hit target instructions
- Provides parity checks on program data stored in the Instruction Cache to ensure data integrity

The PBU block diagram in Figure 3-16 shows data paths to and from the PBU in the dsPIC33A environment. The PBU provides data when the CPU fetches program data from Flash memory. It may provide program data from an internal buffer, or it may fetch program data from Flash if the requested program data is not available. Flash fetch operations are therefore accelerated when data are sourced from internal PBU buffers.

Figure 3-16. PBU Block Diagram



© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.4.1 Architectural Overview

The PBU is a direct mapped 128-line cache that helps in providing faster program data fetches to the CPU from Flash memory. The PBU provides program data from an internal instruction buffer (ISB), but if it is not available in the internal buffer, the PBU may fetch program data from Flash. Flash fetch operations are therefore accelerated when data are sourced from internal PBU buffers.

The PBU provides an interface between the Program Flash Memory (PFM) and the CPU instruction bus and have the following components associated for operation:

- Instruction Stream Buffer (ISB) - Also termed as the Prefetch Unit (PFU), is available for prefetching and caching of linear PFM instruction flows. ISB is the component that buffers program data words from the program memory. The ISB consists of one or more buffers of a fixed depth. Each buffer holds one or more lines of data fetches from Flash memory. The data held in each buffer represents a linear code flow. These are termed as internal PBU buffers.
- Instruction Cache (IC) - Also known as Branch Target Instruction Cache (BTIC), is used for the caching of target instructions that are most frequently hit. The Instruction Cache refers to the cache memory and associated control logic that form the cache. A cache consists of N lines, directly mapped or through M-way associative. The PBU supports a direct mapped 128-line cache. The required width for the cache is 129-bits. The PBU Cache has two operating modes: IC mode and BTIC mode.
- Integrity Checking Logic - Provides parity checks on program data stored in the Instruction Cache to ensure data integrity. This logic provides parity checking and fault injection on the contents of RAM associated with the Instruction Cache.

The PBU assumes Flash data width and Flash access speed are sufficient to allow linear program execution at the desired speed using only the ISB. The ISB serves as the prefetch buffer and allows the next line of Flash to be fetched as instructions from the current line are executed.

The Instruction Cache becomes useful when there are frequent program flow changes in the source code. A program flow change will result in extra clock cycles because the current Flash fetch must be allowed to complete and then a new fetch must be initiated at the new location. If the desired program data is available in the Instruction Cache, the data may be sourced immediately without waiting for the ISB to complete a new fetch from Flash. However, PBU uses a larger, direct-mapped Instruction Cache and has little control and status interface available to the user as its operations are transparent.

The PBU does not provide data or caching for initiators other than the CPU instruction bus. Data access by the CPU data bus and other bus initiators is accomplished via a dedicated read buffer in the NVM wrapper.

The ISB has multiple buffers also called slices. The ISB Slices help increase performance with CALL/ RETURN and other flow changes in the code that return back to the previous code stream.

The ISB is two levels deep in the dsPIC33A PBU. For the first generation of dsPIC33A devices, Flash access time is fast enough to support linear code execution with the given program data word width. Therefore, only one level of prefetch buffer is required. The CPU can execute from the first level, while the next fetch occurs into the second level.

In the case where the code to be executed has a linear flow, no further caching of data would be necessary. However, program flow changes insert latency into the code flow. A prior Flash fetch must be completed and discarded. Then, a new Flash fetch must be started in the new flow. This process can add a variable amount of clock cycles to the execution time, depending on when the flow change occurred relative to the prefetch that was in progress.

When Flash access time is fast enough to support continuous linear program flow, full instruction caching is not required. The cache could be configured as a BTIC, for which only the targets of program flow changes are cached. This mode of cache increases the effective cache size because all program data words do not have to be cached. However, the BTIC operating mode places more

© 2023-2024 Microchip Technology Inc. and its subsidiaries

burden on the internal data buses of the PBU. Program data must be transferred from the cache memory to the ISB when a flow change occurs so that a prefetch of the following data words can take place.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.4.2 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x1E60	CHECON	31:24								
		23:16								ISBBUF
		15:8	ON				CHEINV	CHECOH		
		7:0								FLTINJ
0x1E64	CHESTAT	31:24								
		23:16								
		15:8								
		7:0						TPE	RD	PAR
0x1E68	CHEFLTINJ	31:24								
		23:16								
		15:8								
		7:0	FLTPTR[7:0]							

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.4.2.1 Cache Control Register

Name: CHECON
Offset: 0x1E60

Notes:
1. After being set, this bit will be cleared by hardware after the cache and ISB invalidations are completed. Any automatic invalidation will also result in this bit being cleared.
2. This setting is useful when programming non-program data into Flash (emulated EEPROM).

Legend: R = Readable bit; S = Settable bit; Hardware Clearable bit

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
								ISBBUF
Access								R/W
Reset								1

Bit	15	14	13	12	11	10	9	8
	ON				CHEINV	CHECOH		
Access	R/W				R/S/HC	R/W		
Reset	1				1	1		

Bit	7	6	5	4	3	2	1	0
								FLTINJ
Access								R/S/HC
Reset								1

Bit 16 – ISBBUF ISB Buffer Selection bit

Value	Description
1	When On = 0, ISB buffer 1 will be used for prefetch
0	When On = 0, ISB buffer 0 will be used for prefetch

Bit 15 – ON Cache ON bit

Value	Description
1	Cache and all ISB slices are enabled
0	All cache lines and ISB buffers except for the first buffer slice are invalidated. ISB operates with one buffer slice, two deep buffer (basic prefetch mode)

Bit 11 – CHEINV Manual Invalidate Control bit(1)

Value	Description
1	Force invalidation of all cache and ISB lines
0	Invalidation of Instruction Cache and ISBs occurs according to CHECOH bit

Bit 10 – CHECOH Cache Coherency Control bit(2)

Value	Description
1	Invalidate cache upon a Flash programming event
0	Do not invalidate cache on a Flash programming event

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Bit 0 – FLTINJ Fault Inject Control bit

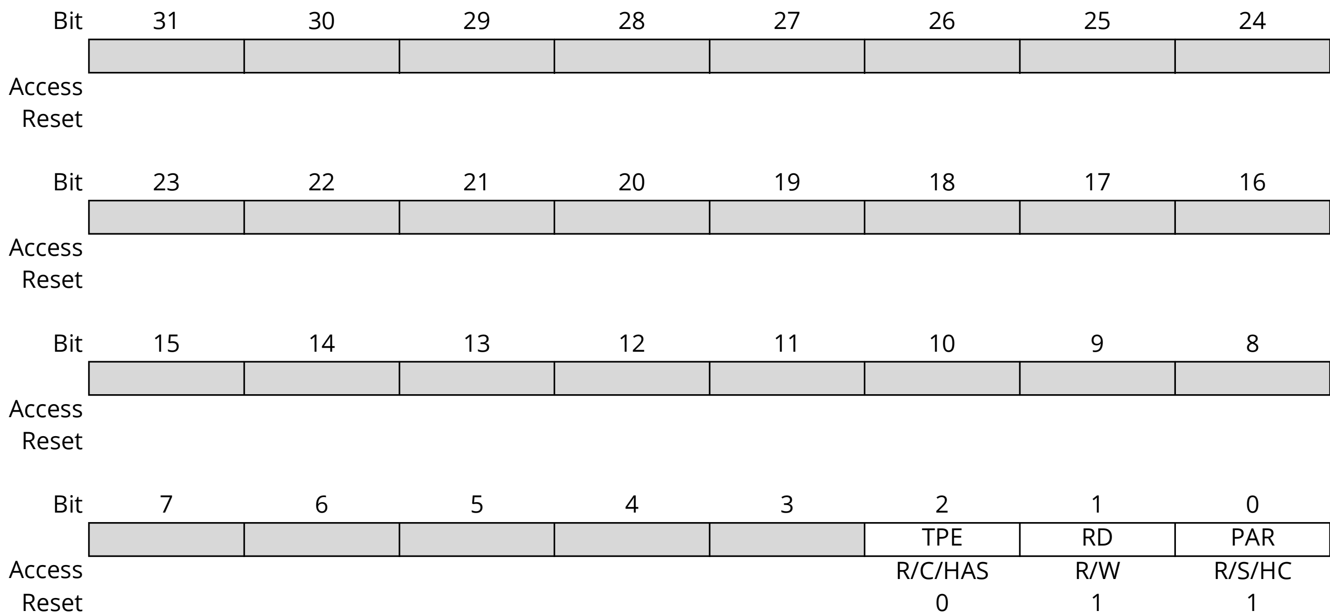
Value	Description
1	Parity fault injection enabled for one-time event; cache line will be invalidated and flushed when access occurs and upbs_event[1] will be asserted to indicate an integrity error to the system.
0	Parity fault injection disabled

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.4.2.2 Cache Status Register

Name: CHESTAT
Offset: 0x1E64

Legend: R = Readable bit; S = Settable bit; Hardware Clearable bit



Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
						TPE	RD	PAR
Access						R/C/HAS	R/W	R/S/HC
Reset						0	1	1

Bit 2 – TPE Read Error Status bit

Value	Description
1	A read error event has occurred; the parity error is latched and a MISS is generated
0	No TAG memory read error event has occurred

Bit 1 – RD Read Error Status bit

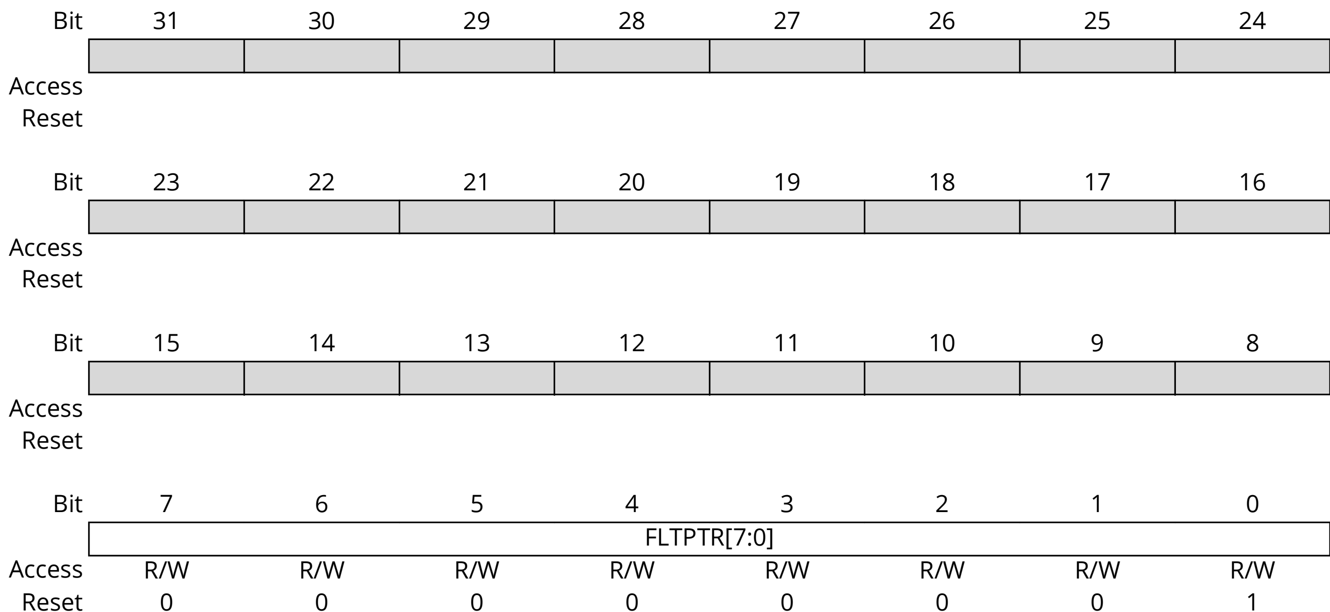
Value	Description
1	A read error event has occurred; the CPU has fetched a word from the ISB with a security error
0	No read error event has occurred

Bit 0 – PAR Cache Parity Error Status bit

Value	Description
1	A parity error event has occurred; the CPU has fetched a word from the cache with a parity error
0	No parity error event has occurred

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.4.2.3 Cache Fault Injection Register



Bits 7:0 – FLTPTR[7:0] Fault Injection Pointer bits

Value	Description
255-129	No effect
128	Cache Data Line Parity bit
127	Bit 127 of cache data line
...	
1	Bit 1 of cache data line
0	Bit 0 of cache data line

3.4.3 Operation

The PBU registers only have control to enable or disable certain PBU functions. Parameters such as ISB depth, ISB number of buffers, cache associativity, etc., are all fixed.

The CHECON.ON bit is reset to '1' by default. This provides the best CPU performance for both linear code and program flow changes. The ON bit can be cleared in software to disable most caching functions and make the PBU behave as a basic 2-deep prefetch buffer. This results in lower code performance due to longer program flow changes but still gives deterministic execution behavior, and thus the program flow changes to longer execution time but takes a constant number of cycles.

3.4.3.1 Cache/ISB Manual Invalidation

Manual invalidation of the Instruction Cache and ISBs is used to force cache coherency when the user knows that the cache and Flash contents may not match. It occurs under the following conditions:

- CHECON.CHEINV Control bit: When set by software, this bit will invalidate both the Instruction Cache and all ISBs. This bit will clear automatically by hardware after the cache and ISB memory have been invalidated.
  Note: CHECON.CHEINV is also cleared should an automatic invalidation occur after the bit has been set.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

- CHECON.ON control bit: The Instruction Cache memory and ISB buffers are invalidated when the CHECON.ON bit is cleared. This will be the case out of Reset. Execution continues using only a single default ISB slice. Setting CHECON.ON has no effect with respect to cache/ISB invalidation as it is already invalidated and the active ISB will contain valid data from the current instruction flow.

3.4.3.2 Cache/ISB Automatic Invalidation

Automatic invalidation of the Instruction Cache and ISBs is used to ensure cache coherency when the device knows that the cache and Flash contents may not match. It occurs under the following conditions:

- Flash write operation: Automatic invalidation only occurs if the Cache Coherency Control bit (CHECON.CHECOH) is set (default) and Flash is programmed/erased. This is only applicable for writing to the active panel in dual-panel devices.
- Parity error: Refer to 3.4.3.3. PBU Data Error Handling for further details. Only the accessed cache line of the ISB buffer is affected, and the remainder of the cache memory does not need to be invalidated.

3.4.3.2.1 Cache Invalidation when Writing to Flash

Whenever the Flash is written, the user has the option to automatically invalidate the instruction Cache and ISBs using the CHECON.CHEOH control bit. If instruction data are being written to Flash, invalidation ensures Flash memory contents remain synchronized with the Cache and ISB contents.

Note: To fully ensure correct operation, it is recommended that the final instruction that initiates Flash programming be followed by 4 NOP instructions to flush the instruction pipeline. This ensures coherency since the remaining instructions in the CPU pipeline will take no action.

3.4.3.3 PBU Data Error Handling

The PBU handles error correction in two ways. First, any data errors that originate from the program memory are tracked. Secondly, internal PBU data errors that may occur while program data are stored in the PBU Cache RAM are monitored.

3.4.3.3.1 Program Memory Data Errors

Error status is captured from the program memory and buffered along with the fetched program data word in the ISB. Consequently, each line in the ISB has 129 bits: 128 bits of data and 1 bit for error status. The error status bit indicates the data read from the program memory are unusable and incorrect. The program memory data can be bad for multiple reasons, including an uncorrectable ECC error and a security violation that would suppress the data. In any case, the data are not a valid CPU instruction and should not be executed by the CPU.

The PBU does not generate any kind of event, trap or interrupt when bad data are fetched from the program memory. This is because the ISB may speculatively fetch data that would never be executed by the CPU. Secondly, the CPU may speculatively fetch instructions from the PBU during conditional branches that may never get executed.

A bus error signal is passed with the program data to the CPU for instructions fetched from the PBU. If the program data are invalid with the bus error signal asserted, then the CPU can suspend execution in the pipeline and cause a trap event to occur.

3.4.3.3.2 Cached Data Error

The second method of PBU error handling occurs when the cache has detected a parity error on a cached line of program word data. When valid program data is cached for later consumption, then the error status bit is stripped, and the program data word is stored in the cache memory. A single even parity bit is calculated and stored along with the data. This parity bit is used to protect the system from data corruption that could occur in the cache RAM.

A maskable interrupt event is generated by the PBU when a parity error is detected on a cache line. In this case, the cache will invalidate the line with the parity error and the program data must be re-fetched from the program memory. Other than the interrupt event, the only other effect that can

© 2023-2024 Microchip Technology Inc. and its subsidiaries

be observed during a cache parity error is additional execution latency caused by Flash program fetch. No address associated with the parity error is captured.

3.4.3.3.3 Corrected Program Memory Data Errors

The program memory error correction logic may correct a data error in the program data supplied to the PBU. These corrections are not reported to the PBU which is usually done for the uncorrectable errors. Specifically, a single-bit corrected error (SEC event) is not reported to the PBU.

The program memory is responsible for tracking and reporting the corrected event. These actions serve as a warning to the system software that integrity of the NVM data may be failing.

3.4.3.4 Cache Fault Injection

A single bit error can be injected on any of the data bits of the cache line or the associated parity bit. The error injection is performed by XORing the data read from the cache line with a '1'. Since the PBU can cache program data from a variety of address locations depending on the program flow, it is impractical to perform error injection for a particular program memory fetch address.

The PBU error injection, when enabled with the FLTINJ bit, will cause a one-time error injection the next time the cache memory is accessed by the CPU. The CHESTAT.PAR bit will indicate when the error injection has been performed. At this time, the PBU will also signal that an integrity error has occurred by creating an interrupt event. The user will not be able to determine which line of the cache buffer caused the event and fetch address.

A write to the FLTPTR register while FLTINJ = 1 will have the effect of re-arming the fault injection. This will help facilitate a software test routine that cycles through an error injection on each bit.

3.4.3.5 Non-Cached Events

Certain fetches from the NVM are not cached. These include:

- Interrupt Vector fetches
- Fetches of debug executive code
- Fetches of invalid program memory data

All these types of fetches are not cached to avoid cache thrashing. Thrashing occurs when other useful data are evicted from the cache and replaced with less useful or invalid data. Inhibiting caching during the above fetches is expected to improve the overall efficacy of the cache, resulting in more cache hits at run-time.

Interrupt Vector fetches are a special type of non-cached event. Specifically, only one program word is fetched from the NVM when the CPU indicates a vector fetch and the ISB is bypassed. When an interrupt occurs, the interrupt vector address is fetched from the vector table, then the instruction at the interrupt vector address is fetched. There is no need for an ISB to perform a prefetch and fetch the program word after the one that contains the interrupt vector address. This would be wasteful and produce extra latency in the servicing of the interrupt event.

The PBU monitors whether the CPU is executing user code or debug executive code. Instructions fetched from the debug executive code are not cached. This avoids additional indeterministic behavior when code execution transitions from the debug executive code back to user mission-mode code.

In addition, the BMX supports execution from RAM, and a RAM based Interrupt Vector Table (IVT). Program or vector fetches from RAM are also non-cached events. However, this capability introduces the possibility of both a vector and its associated handler routine being in either NVM or RAM. Whenever the IVT and/or an exception handler (interrupt or trap) is located within RAM, this is treated as a special case by the PBU to maintain efficient operation.

3.4.3.6 PBU Performance Monitoring

Each word of data requested on the CPU instruction bus will be sourced either from the ISB or the Instruction Cache and not the external NVM. This ensures that each fetch of program data can be completed in minimal time, which maximizes application performance.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Program data that are not already present in the ISB or cache must be fetched from the NVM, which takes additional cycles and decreases overall application performance. Once program data in a particular NVM program word has been consumed by the CPU, it is stored in the Instruction Cache for later use. The exceptions to this are program data with uncorrectable errors, security violations, or debugger executive program data. Once stored, the program data will be available for later reuse in the Instruction Cache until those contents are erased and replaced with another program data word.

3.4.3.6.1 Cache Busy Cycles

The program word is cached on the cycle following the fetch from NVM. During this time, the IC will be busy because of the write to cache memory. If the CPU requests program data during this cycle, the PBU will check the contents of the ISB for an address tag match. In most cases, the data may be sourced from the ISB while the IC is busy. This results in an ISB hit and no extra cycle penalty is incurred. When the IC is busy and no ISB hit occurs, then an extra cycle of latency will be inserted while the PBU waits for the IC write cycle to complete. Then, the IC address tags are checked for an IC hit.

3.4.3.6.2 PBU Performance Event Outputs

The PBU has event outputs that can be connected to external performance counters at the device level for characterization of the PBU performance. These events can be counted over a period of application execution and compared with the total number of executed instructions and/or the total number of elapsed clock cycles to get a measurement of the PBU efficacy. The performance event signals available from the PBU include:

- Instruction Cache “hit” event
- Instruction Stream Buffer “hit” event
- PBU “hit” event
- Instruction Cache “busy” event

The IC hit event indicates when a particular instruction was fetched from the cache memory. The ISB hit event indicates when a particular instruction was fetched from the ISB. This generally happens on the second fetch from a program word while that word is written to the cache memory.

The PBU hit event is of most interest for PBU performance analysis. This event signal is the logical OR of the IC hit and the ISB hit events and indicates that the PBU was able to source the requested data without initiating a new NVM fetch.

The IC busy event is used to count the number of extra cycles that were inserted when the ISB could not source the requested data and a Wait state was necessary to determine if the data were available in the IC. An IC busy event is expected to be infrequent and would occur during program flow changes.

3.4.3.6.3 Factors Affecting PBU Efficacy

PBU efficacy is not a constant value. For a given code segment such as function call, the efficacy of the PBU will be very much dependent on these factors:

- The code that was executed prior to a given code segment
- The size of the code
- The specific location of this code in memory
- Flow changes that occur during the execution of a specific code segment

Different performance results are possible when a specific segment of code is executed in one context vs. another context. The prior code executed will determine what code data is present in the cache memory. The prior code may have evicted all program data associated with the segment of interest. However, if the segment of interest is repetitively executed, then there is a strong possibility that program data associated with this segment will remain in the cache memory without eviction.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

In general, a small segment of code which is repetitively executed will produce the best PBU performance results. This is because the code size is small enough to fit within the cache memory and the repetitive nature of the code will maximize the reuse of the cache contents with a minimum of evictions. The absolute location of a code segment within memory will impact the PBU performance. This is closely related to how the code is compiled, optimized, and linked during the software development process.

Two different program data words in a segment of code could have the same address tags. If these program data words are executed often, then numerous cache evictions and NVM fetches will result during code execution. A larger cache memory and/or increased cache associativity can both help this issue. A larger cache memory increases the number of available address tags, while increased associativity increases the number of location options where a specific program data word could be stored. The more flow changes that occur in each segment of code, the higher the possibility that PBU performance will be reduced.

3.4.3.6.4 Implications of Variable NVM Wait States

The NVM Wait states are currently fixed at three, supporting a 4-cycle NVM read access time, and the nature of the PBU/NVM data access handshake is not sensitive to NVM access time. However, variable access times could be advantageous:

1. Devices are designed to target maximum frequency, and the NVM read access time is based upon this requirement. But not all applications will require full-speed operation and/or may be willing to trade-off speed for lower power consumption. Consequently, it may be desirable to allow the user to select fewer (or no) NVM read access Wait state when operating at lower frequencies. This will improve the IC/ISB miss latency and decrease the effective CPI (clocks per instruction) metric, improving overall device execution efficacy.
2. Slower Flash panels will consume less power so future devices may support different speed NVM. Zero Wait state linear code execution directly from Flash would, of course, no longer be possible but would rely on the ISB and IC implementations.

3.5 Performance Monitor Unit (PMU)

The performance monitor provides a method to analyze code efficiency, and allows software routines that incur processor stalls to be identified and optimized. In the dsPIC33A family of devices, the architecture does not have a fixed relationship between the CPU clock speed in MHz and the throughput of the CPU in MIPS (Million Instructions per Second). The throughput of the CPU is dependent on extra cycles incurred from the following:

- CPU pipeline data dependency
- Branches or program flow changes
- Cache misses
- Slow memory or SFR accesses
- Arbitration between bus masters
- A bus that is slower than the CPU

The performance monitor counts the events that cause extra cycles to be inserted into the program flow and the number of elapsed clock cycles. Using this information, the cycles-per-instruction (CPI) can be calculated and the reasons for poor code efficiency can be determined. The CPI value is the number of elapsed clock cycles divided by the number of opcodes that were executed. The stall cycle types listed above will increase the CPI.

The performance monitor uses a set of event signals from the CPU to determine stalls. The module features eight independent 64-bit counters to capture the number of events.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.5.1 Device-Specific Information

Table 3-9. Performance Monitor Summary

Number of counters	Peripheral Bus Speed	Clock Source
8	Standard	Standard Speed Peripheral Clock

Table 3-10. Counter Event Source Selection

SELECT n [4:0]	Event source	Note
18	Fetch stage PBU miss	This event indicates that the requested program data could not be sourced from either the cache memory or the ISB. Therefore, a new fetch from program memory with additional execution cycles was required to obtain the data.
17	Fetch stage PBU hit	This event indicates that the requested program data was sourced from either the cache memory or the ISB. Therefore, no additional execution cycles were required to fetch the instruction.
16	Fetch stage cache busy	Indicates a cycle during which time the cache was busy transferring data from the instruction stream buffer (ISB) to the cache memory.
15	Fetch stage program memory vector fetch	Indicates that the CPU is fetching an interrupt vector and is aligned with a Program Flow Change event. This event can be used to count interrupt events.
14	Fetch stage program memory program flow change	Indicates that a change in program flow has occurred. This could be due to a CALL, RETURN, RETFIE, conditional or unconditional branch, or interrupt event.
13	Fetch stage read stall	Indicates an extra cycle is needed to fetch a program word from memory. This could be caused by a cache miss or an arbitration conflict when fetching program words and data from the same memory.
12	Fetch stage interrupt latency count enable	Indicates the number of cycles due to interrupt latency.
11	Address stage stall	Indicates that CPU pipeline was stalled in the Address stage for any reason, possibly because the instruction is being discarded.
10	Address stage read stall	Indicates that an instruction could not continue because of extra latency reading a RAM or SFR location.
9	Address stage FPU read stall	Indicates that CPU execution is presently stalled because the CPU cannot read from a FPU register. This has occurred because the FPU is currently busy updating the register data.
8	Address stage FPU instruction stall	Indicates that execution in the FPU coprocessor is currently stalled due to a register data dependency.
7	Address stage hazard	Indicates an extra execution cycle caused by a data dependency upon an earlier instruction in the CPU pipeline, which could not be forwarded.
6	Read stage branch mispredict	Indicates an extra execution cycle caused by mispredicted program flow changes.
5	Read stage conditional branch	Indicates the occurrence of a conditional branch instruction. The count of conditional branch instructions can be compared to the number of branch mispredictions in order to determine the effectiveness of the CPU branch prediction logic.
4	Write stage stall	Indicates that an instruction could not continue because of extra latency writing to RAM or SFRs.
3	Write stage FPU stall	Indicates that CPU execution is presently stalled because the CPU cannot write to the FPU registers. This has occurred because the FPU is currently busy working on the existing register data.
2	CPU instruction completed	Indicates that an instruction in the CPU pipeline has completed.
1	CPU cycle elapsed (reference)	This event count provides the total number of CPU clock cycles elapsed.
0	None	

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.5.2 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x1E10	HPCCON	31:24								
		23:16								
		15:8	ON		CLR					
		7:0								
0x1E10	HPCSEL0	31:24				SELECT[3][4:0]				
		23:16				SELECT[2][4:0]				
		15:8				SELECT[1][4:0]				
		7:0				SELECT[0][4:0]				
0x1E14	HPCSEL1	31:24				SELECT[7][4:0]				
		23:16				SELECT[6][4:0]				
		15:8				SELECT[5][4:0]				
		7:0				SELECT[4][4:0]				
0x1E18 ... 0x1E1F	Reserved									
0x1E20	HPCCNTL0	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E24	HPCCNTH0	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E28	HPCCNTL1	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E2C	HPCCNTH1	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E30	HPCCNTL2	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E34	HPCCNTH2	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E38	HPCCNTL3	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E3C	HPCCNTH3	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E40	HPCCNTL4	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E44	HPCCNTH4	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E48	HPCCNTL5	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							

© 2023-2024 Microchip Technology Inc. and its subsidiaries

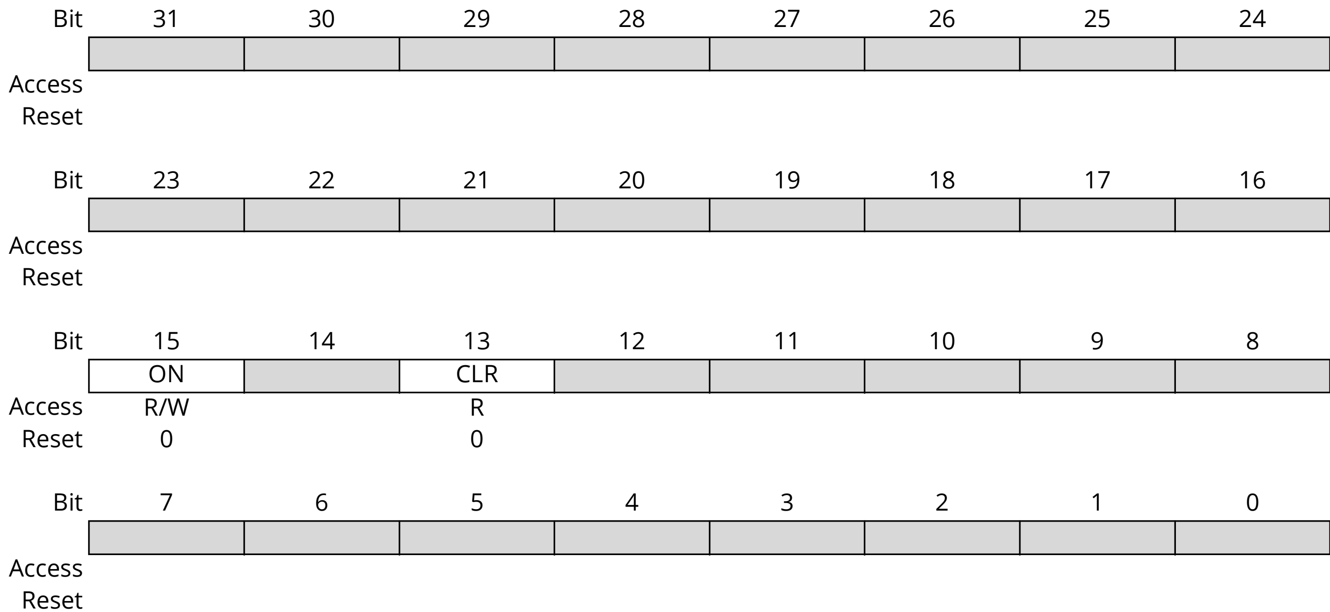
...........continued

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x1E4C	HPCCNTH5	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E50	HPCCNTL6	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E54	HPCCNTH6	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							
0x1E58	HPCCNTL7	31:24	HPCCNT[31:24]							
		23:16	HPCCNT[23:16]							
		15:8	HPCCNT[15:8]							
		7:0	HPCCNT[7:0]							
0x1E5C	HPCCNTH7	31:24	HPCCNT[63:56]							
		23:16	HPCCNT[55:48]							
		15:8	HPCCNT[47:40]							
		7:0	HPCCNT[39:32]							

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.5.2.1 HPCCON Register

Name: HPCCON
Offset: 0x1E10



Bit 15 – ON On Control bit

Value	Description
1	Module is enabled and counters increment on event signals
0	Module is disabled and counters do not event on event signals. Counter values may be read.

Bit 13 – CLR Clear Control bit

A write of a '1' to this location will cause the event counters to clear. This bit may be set at any time whether the PMU is in the Enabled state or the Disabled state. This bit location always reads as '0'.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.5.2.2 HPCSEL0 Register

Name: HPCSEL0
Offset: 0x1E10

Bit	31	30	29	28	27	26	25	24
				SELECT[3][4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
				SELECT[2][4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
				SELECT[1][4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
				SELECT[0][4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bits 28:24 – SELECT[3][4:0] Counter #3 Event Source Selection bits
These control bits determine which event is counted by the associated counter.
See Table 3-10 for assignments.

Value	Description
11111-00001	Selects the event to be monitored
00000	No event selected (1'b0), counter is disabled

Bits 20:16 – SELECT[2][4:0] Counter #2 Event Source Selection bits
These control bits determine which event is counted by the associated counter.
See Table 3-10 for assignments.

Value	Description
11111-00001	Selects the event to be monitored
00000	No event selected (1'b0), counter is disabled

Bits 12:8 – SELECT[1][4:0] Counter #1 Event Source Selection bits
These control bits determine which event is counted by the associated counter.
See Table 3-10 for assignments.

Value	Description
11111-00001	Selects the event to be monitored
00000	No event selected (1'b0), counter is disabled

Bits 4:0 – SELECT[0][4:0] Counter #0 Event Source Selection bits
These control bits determine which event is counted by the associated counter.
See Table 3-10 for assignments.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Value	Description
11111-00001	Selects the event to be monitored
00000	No event selected (1’b0), counter is disabled

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.5.2.3 HPCSEL1 Register

Name: HPCSEL1
Offset: 0x1E14

Bit	31	30	29	28	27	26	25	24
				SELECT[7][4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
				SELECT[6][4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
				SELECT[5][4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
				SELECT[4][4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bits 28:24 – SELECT[7][4:0] Counter #7 Event Source Selection bits
These control bits determine which event is counted by the associated counter.

Value	Description
11111-00001	Selects the event to be monitored
00000	No event selected (1’b0), counter is disabled

Bits 20:16 – SELECT[6][4:0] Counter #6 Event Source Selection bits
These control bits determine which event is counted by the associated counter.

Value	Description
11111-00001	Selects the event to be monitored
00000	No event selected (1’b0), counter is disabled

Bits 12:8 – SELECT[5][4:0] Counter #5 Event Source Selection bits
These control bits determine which event is counted by the associated counter.

Value	Description
11111-00001	Selects the event to be monitored
00000	No event selected (1’b0), counter is disabled

Bits 4:0 – SELECT[4][4:0] Counter #4 Event Source Selection bits
These control bits determine which event is counted by the associated counter.

Value	Description
11111-00001	Selects the event to be monitored
00000	No event selected (1’b0), counter is disabled

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.5.2.4 HPCCNTLx Register

Name: HPCCNTLx
Offset: 0x1E20, 0x1E28, 0x1E30, 0x1E38, 0x1E40, 0x1E48, 0x1E50, 0x1E58

Bit	31	30	29	28	27	26	25	24
	HPCCNT[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	HPCCNT[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	HPCCNT[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	HPCCNT[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – HPCCNT[31:0] Event Counter bits

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.5.2.5 HPCCNTHx Register

Name: HPCCNTHx
Offset: 0x1E24, 0x1E2C, 0x1E34, 0x1E3C, 0x1E44, 0x1E4C, 0x1E54, 0x1E5C

Bit	31	30	29	28	27	26	25	24
	HPCCNT[63:56]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	23	22	21	20	19	18	17	16
	HPCCNT[55:48]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8
	HPCCNT[47:40]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	7	6	5	4	3	2	1	0
	HPCCNT[39:32]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – HPCCNT[63:32] Event Counter bits

3.5.3 Operation

The performance monitor operates on the basis of comparing the counter values to the number of CPU cycles. To capture the number of CPU cycles as a reference, one of the available counters is used.

For example, counter 0 can be used for the reference count, and the remaining counters can be used to monitor the available events.

3.5.3.1 Event Selection

Each counter has an associated control to select one of the event sources. The SELECTn[4:0] bits in HPSEL0 and HPSEL1 select one of the signals that are listed in Table 3-10. The CPU cycle elapsed event is the reference and is incremented on each CPU cycle. The CPU instruction completed event indicates that the CPU pipeline has completed. Comparing instructions completed to cycles elapsed yields the CIP value. The ideal value is one. The remaining stall, branch or hazard events can be used to determine where stalls occur and what part of the code to optimize.

3.5.3.2 Counters

Each 64-bit counter is split across a pair of 32-bit registers, HPCCNTLx and HPCCNTHx. The registers are read- only and do not have provisions for saturation or roll over events. It is up to user software to halt the module before saturation occurs. The counters can be reset with the CLR bit (HPCCON[13]). The counters are started and stopped using the ON bit (HPCCON[15]). The count values should only be read when ON = ‘0’.

3.5.3.3 Debugging

Provisions have been made to support the performance monitor in Debug mode. By default, the module is halted in Debug mode to avoid counting cycles associated with the debug executive.

3.5.3.4 Operation in power saving modes

The Performance Monitor module does not operate in Sleep or Idle modes.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6 Floating-Point Unit (FPU) Coprocessor

The dsPIC33A FPU Coprocessor includes hardware implementations of the most common floating-point operations for both Single Precision (32-bit) and Double Precision (64-bit) data formats. It is intended to significantly accelerate C compiler floating point operations when compared to executing software library equivalents and is designed to be compliant with the IEEE 754-2008/2019 floating point standards. It also includes additional non-IEEE compliant features which may be enabled to handle subnormal values and improve performance.

3.6.1 Features

- Comprehensive IEEE 754-2008/2019 compliant instruction set
  - Supports both Single and Double Precision operations for most instructions
  - Supports all required rounding modes
- Closely coupled to dsPIC33A CPU core
  - Instructions issued from CPU core as part of application instruction stream
  - Independent instruction pipeline and hazard management
- 32 x 32-bit data registers (F-regs)
  - May be used to hold 32-bit Single Precision or 64-bit Double Precision values
  - Base plus 7 partial FPU register contexts
- Optional subnormal handling for improved performance
  - Subnormal result “Flush-To-Zero” (FTZ) mode
  - Subnormal operand “Subnormals-Are-Zero” (SAZ) mode
- Comprehensive exception implementation and reporting structure
  - IEEE 754-2019 compliant exception implementation
  - Additional exceptions supported for Huge Integer results and Subnormal operands
- Debug features supported:
  - Exception address capture register (FEAR)
  - Exception break signaling
  - NaN propagation

3.6.2 Architectural Overview

The FPU macro relies on the associated dsPIC33A CPU for all instruction fetches, most decoding, and for all operand movement to and from the system memory. The FPU contains no local memory other than its own register set. Being coupled to the CPU, data size nomenclature is common to both CPU and FPU wherein a word is 16-bits wide, a long word is 32-bits wide and a double word is 64-bits wide.

FPU instructions are part of the CPU Instruction Set Architecture and are executed as part of the CPU code image. FPU instructions are therefore executed as a part of the normal execution flow. There are no restrictions with regards to when FPU instructions may appear within the instruction flow.

The CPU can issue, and the FPU can accept, no more than one instruction per clock cycle. However, once issued, the CPU and FPU use independent pipelines to execute the instruction. Consequently, there can be multiple instructions in the process of being executed in both pipelines at any one time. The FPU pipeline will stall the CPU when it is unable to accept any more instructions. The FPU pipeline is also sensitive to speculative instruction control from the CPU (i.e., such that not all issued FPU instructions will be committed). This allows FPU instructions to be located within speculative execution slots that follow conditional branches.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

After successful issue of an FPU instruction, the CPU continues as if executing a single cycle FNOP instruction, and the FPU instruction execution continues within the FPU. Therefore, as some FPU instructions require several cycles to complete, subsequent CPU (and/or FPU) instructions can be fetched, issued and executed (dependencies aside) while the FPU operation progresses. Only when the CPU encounters a hazard with the FPU will it be stalled until the hazard is resolved.

Data and structural hazards are detected and mitigated in both the CPU and FPU and can result in operational stalls which will extend the execution time and increase the effective Cycles Per Instruction of both CPU and FPU instructions.

Note: Refer to the dsPIC33A Programmer's Reference Manual for the syntax of all FAND, FIOR, FMUL, etc, instructions.

3.6.2.1 Instruction Pipeline Overview

The pipeline stages consist of Read (RD), Execute (X[n]) and Write-Back (WB), differentiated from the equivalent CPU pipeline stages through the use of different nomenclature. The RD-stage is a single cycle operation (unless stalled). The WB-stage is always a single cycle operation. However, the execute stage will consist of as many cycles as deemed necessary for the selected instruction functional block. Most basic functions are single cycle execute operations, though more complex functions (e.g., divide) can be many cycles.

Each instruction that is issued to the FPU must be completed (or killed if speculative) in the order issued. That is, Out of Order (OoO) execution is not supported. However, as the execution time of the FPU instructions can vary considerably, in-order execution requires logic to tag each instruction as it is committed for execution, then track its progress as it flows through the instruction pipeline. Subsequent instructions will therefore be stalled until such time that earlier ones have progressed to allow for sequential, in-order execution.

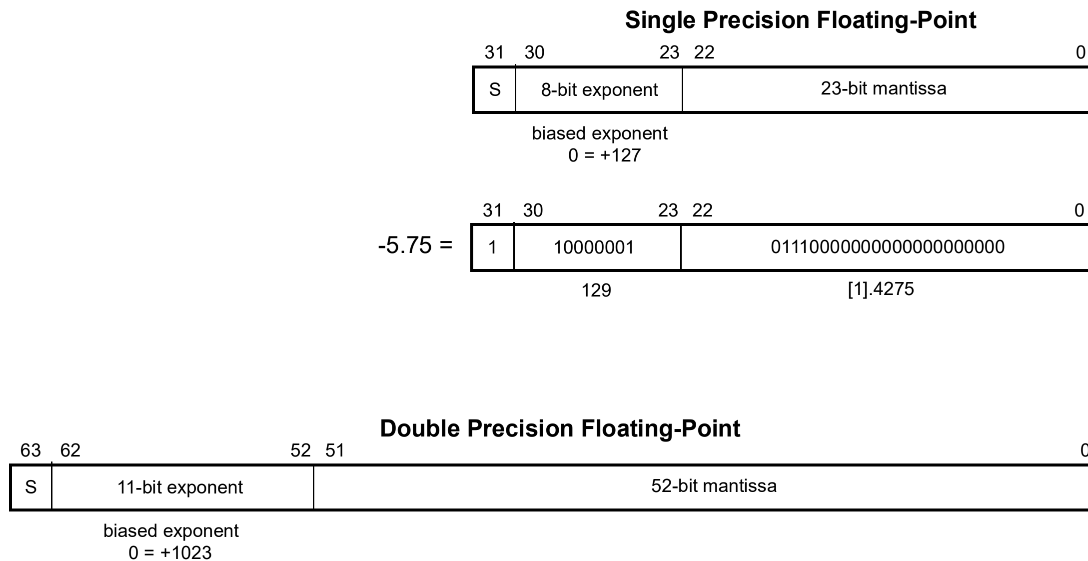
3.6.2.2 Introduction to Floating Point

The IEEE standard for Floating-Point Arithmetic (IEEE 754-2008) specifies the floating-point data formats which are comprised of a Sign bit, an exponent value and a (fractional) mantissa value. The dsPIC33A Floating-Point Unit (FPU) supports both Single Precision (32-bit, SP) and Double Precision (64-bit, DP) operations for most (though not all) instructions. To avoid the need for another Sign bit in the exponent, the IEEE floating-point format exponent is biased by 127 (SP) or 1023 (DP). Consequently, for any datum, the required IEEE exponent value = datum exponent + bias.

In addition, the '1' to the left of the most significant bit of the mantissa is implied for all numbers except subnormal numbers and is consequently referred to as the leading bit convention "hidden bit." The mantissa is therefore a fractional value with an implied integer value of [1].

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-17. IEEE Floating-Point Data Formats and Single Precision Example



An IEEE floating-point number can therefore be represented as:

$(-1)^S \times [1].(m_{(base2)}) \times 2^{(e-bias)}$

where:

- 'S' indicates the sign of the number (same values as a signed integer value)
- 'e' represents the exponent value
- 'm' represents the fractional mantissa value
- 'bias' is 127 (SP) or 1023 (DP)

For example, -5.75 = $-(1.4275 \times 2^2)$. In IEEE SP format this would be represented as:

(-1)1 x [1].4275 x 2(129-127)

or (as shown in Figure 3-17):

S = 1, exponent = 129_{10}, mantissa = [1].427510 or:

0xC0B8 0000

3.6.2.2.1 IEEE 754-2008 Compliance

This module is compliant with the IEEE 754-2008 Standard for Floating-Point Arithmetic for data formats, supported signaling and quiet branch predicates, exception status flags, and exception status behavior.

Note that the IEEE 754-2008 minNum(x,y) and maxNum(x,y) definitions are supported only through the largely compatible IEEE 754-2019 minimumNum(x,y) and maximumNum(x,y) operations via the FMINNUM and FMAXNUM instructions. The functional differences related to how +0 and -0 are considered:

- IEEE 754-2008 minNum(x,y) / maxNum(x,y): Operand values +0 and -0 are regarded as equivalent. The (implementation dependent) result could therefore be either +0 or -0.
- IEEE 754-2019 minimumNum(x,y) / maximumNum(x,y): Operand values +0 and -0 are not regarded as equivalent such that -0 compares to less than +0. The result will therefore be the correct sign of 0 based on the selected operation.

Features Beyond IEEE 754 Requirements

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Exception Address Capture Register

The Floating-Point Exception origination address capture register (FEAR) captures the address of the instruction that generates a floating-point macro exception, provided the associated exception mask bit is clear. If the exception is masked, nothing is captured.

This register is intended for use during system debug, though the FEAR register is read/write in both Mission and Debug modes.

Huge Integer Exception

This exception is signaled whenever a Float-to-Integer conversion operation (FF2DI and FF2LI) results in an integer value that is larger than the destination register can represent. It is not defined within any IEEE 754 specification apart from a reference to setting the Invalid exception should an integer value exceed the destination size unless this "cannot otherwise be indicated."

3.6.2.2.2 IEEE 754-2019 Compliance

Minimum and Maximum Functions

The FPU module supports all minimum and maximum operations defined in the IEEE 754-2019 standard. The IEEE 754-2008 minNum(x,y) and maxNum(x,y) operations are not directly supported.

- FMINNUM, FMAXNUM: IEEE 754-2019 minimumNumber(x,y)/maximumNumber(x,y) functions. When one of the input operands is a NaN and the other input is a floating-point number (that is not a NaN), the instructions will return the floating-point number. If both input operands are a NaN, the instructions will return a qNaN.
- FMIN, FMAX: IEEE 754-2019 minimum(x,y)/maximum(x,y) functions. When one (or both) of the input operands is a NaN, the instructions will return a qNaN.

Refer to the truth table shown in Table 3-11 for a definition of how NaN operands are handled.

For all minimum and maximum operations, any finite operand value will compare as less than +infinity, or greater than -infinity. Operand value of -0 compares to less than +0.

Table 3-11. FMINNUM/FMAXNUM/FMIN/FMAX Operation

Op	Source Operands		Invalid Exception Mask	Result Fd	FSR.INVAL	Invalid Exception Taken?
	Fb	Fs				
FMINNUM FMAXNUM FMIN FMAX	FPN1	FPN2	Don't care	FPN1 or FPN2[1,2,3,4]	0	No
	qNaN1	qNaN2	Don't care	qNaN1 or qNaN2 [5]	0	No
	sNaN	qNaN	1	qNaN (Fs) [6]	1	No
			0			Yes
	qNaN	sNaN	1	qNaN (Fb) [6]	1	No
			0			Yes
	sNaN1	sNaN2	1	Quieted sNaN1 or sNaN2 [5]	1	No
			0			Yes
FMINNUM FMAXNUM	FPN1	qNaN	Don't care	FPN1	0	No
	qNaN	FPN2	Don't care	FPN2	0	No
	FPN1	sNaN	1	FPN1	1	No
			0			Yes
	sNaN	FPN2	1	FPN2	1	No
			0			Yes

© 2023-2024 Microchip Technology Inc. and its subsidiaries

...........continued

Op	Source Operands		Invalid Exception Mask	Result Fd	FSR.INVAL	Invalid Exception Taken?
	Fb	Fs				
FMIN FMAX	FPN1	qNaN	Don’t care	qNaN (Fs)	0	No
	qNaN	FPN2	Don’t care	qNaN (Fb)	0	No
	FPN1	sNaN	1	Quieted sNaN (Fs)	1	No
			0			Yes
	sNaN	FPN2	1	Quieted sNaN (Fb)	1	No
			0			Yes

Notes:
1. FPN1 and FPN2 are floating-point numbers that are not a NaN (i.e., normal, zero, infinity or sub-normal).
2. Result determined by FMINNUM/FMIN or FMAXNUM/FMAX operation.
3. Operand value of -0 compares to less than +0.
4. If Fb = Fs (and of the same sign, including infinities), result (Fd) will be loaded with Fb.
5. NaN with largest significand will be passed to result (Fd), quieted if an sNaN.
6. qNaN values have priority over sNaN values (see Table 3-13).

Clamping (Limit) Functions

Although not specified in any IEEE 754 standard, the ISA supports a clamping (or limit) instruction (FFLIM) intended for use where an input operand needs to be constrained between an upper and lower limit. It serves a similar purpose to the integer equivalent FLIM instruction and is essentially a concurrent execution of FMIN and FMAX operations with a common operand. Refer to the truth table shown in Table 3-12 for a definition of how NaN operands are handled.

Any finite operand value will compare as less than +infinity or greater than -infinity. Operand value of -0 compares to less than +0.

For FFLIM operations, when both upper and lower limits are either both qNaN or both sNaN values, a NaN significand comparison (to select result NaN source) is not required, and the Fb NaN will be the default source for the result. This differs from how coincident NaN values are treated in general.

Furthermore, a NaN input value (Fd) will cause the limit values to be ignored and will become the source for the result. That is, checks between input NaNs and limit values (NaNs or otherwise) are also not required.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Table 3-12. FFLIM Operation

Fb(3) (Lower Limit)	Fs(3) (Upper Limit)	Fd (Input Value)	Invalid Exception Mask	Fd (Result)	FSR.INVAL	Exception Taken?
FPNL	FPNU	FPN	Don't care	FPNL or FPNU or FPN(2)or Distinguished qNaN(3)	0	No
FPNL	qNaN_U	FPN	Don't care	qNaN_U	0	No
FPNL	sNaN_U	FPN	1	Quieted sNaN_U	1	No
			0			Yes
qNaN_L	FPNU	FPN	Don't care	qNaN_L	0	No
sNaN_L	FPNU	FPN	1	Quieted sNaN_L	1	No
			0			Yes
sNaN_L	sNaN_U	FPN	1	Quieted sNaN_L(5)	1	No
			0			Yes
sNaN_L	qNaN_U	FPN	1	qNaN_U(6)	1	No
			0			Yes
qNaN_L	sNaN_U	FPN	1	qNaN_L(6)	1	No
			0			Yes
qNaN_L	qNaN_U	FPN	Don't care	qNaN_L(5)	0	No
Don't care	Don't care	sNaN	1	Quieted sNaN(7)	1	No
			0			Yes
Don't care	Don't care	qNaN	Don't care	qNaN	0	No

Notes:

1. FPNL and FPNU are floating-point numbers that are not a NaN.
2. Result determined by FFLIM operation.
3. If Fs is less than Fb (and neither Fs nor Fb are NaN values), the result will be the distinguished qNaN, and the Invalid exception will be signaled.
4. FFLIM operation based on IEEE 754-2019 minimum(x,y) and maximum(x,y) operation definitions.
5. Unlike FMIN/FMAX operations, no magnitude comparison of limit NaN values is required. Default result will always be sourced from Fb.
6. qNaN values have priority over sNaN values (see Table 3-13).
7. Unlike FMIN/FMAX operations, no comparison of limit and input (Fd) values is required. Default result will always be sourced from Fd.

NaN Propagation

The FPU macro supports NaN (payload) propagation to facilitate code debugging. After the CPU issues an instruction to the FPU, the source operands are examined and a NaN value detected, compared, and then propagated. Two operand instructions propagate NaN values as shown in Table 3-13.

The FMAC instruction is a special case with respect to NaN propagation as it consists of essentially three operands consisting of the two source operands (for the multiply) and a prior FMAC result value (i.e, the intermediate used for the accumulate function). The source operands are examined as usual but in conjunction with the selected intermediate result, and any NaN values detected are propagated as defined by Table 3-13.

FFLIM is also a three-operand instruction, though it is ultimately either a two-operand maximum or minimum operation based on the value of the source operand. NaN values detected are propagated as defined by Table 3-12.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Table 3-13. NaN Propagation Priority

Source Operands		Result	Condition	Notes
Fb	Fs			
FPN	sNaN	Quieted sNaN	—	INVAL signaled
FPN	qNaN	qNaN	—	-
sNaN	FPN	Quieted sNaN	—	INVAL signaled
qNaN	FPN	qNaN	—	-
qNaN1	qNaN2	qNaN1	qNaN1 ≥qNaN2	-
		qNaN2	qNaN2 > qNaN1	-
sNaN	qNaN	qNaN	—	INVAL signaled
qNaN	sNaN	qNaN	—	INVAL signaled
sNaN1	sNaN2	Quieted sNaN1	sNaN1 ≥ sNaN2	INVAL signaled
		Quieted sNaN2	sNaN2 > sNaN1	

NaN Propagation Rules

For instructions that generate a result, special propagation rules apply when one or both source operands are NaN values, such that sNaNs can be successfully used as “tracer” values.

When both source operands are NaNs, qNaNs take priority over sNaNs. The appropriate NaN values will be selected as the operation default result as shown in Table 3-13. In the absence of any NaN source operands, any other floating-point numbers will be processed by the FPU module to generate the result.

Note: Source sNaN values will always generate an Invalid exception, but the corresponding quieted sNaN may not always be the operation result.

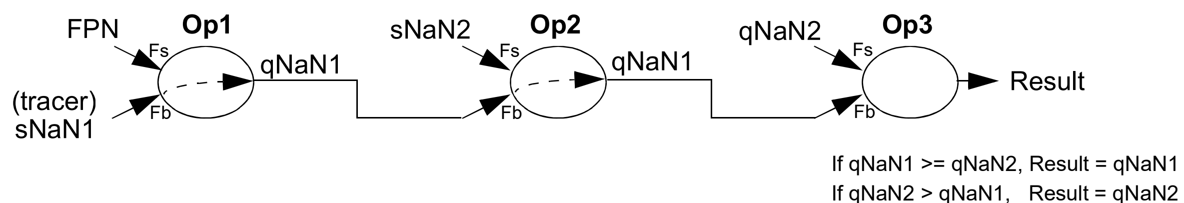
This magnitude comparison is based on the magnitude of the significand associated with each of these values (the sign is ignored). It is straightforward to implement because:

- The MSb of a sNaN significant is 0 (with any non-zero value in the remaining bits).
- The MSb of a qNaN significant is 1 (with any value in the remaining bits).

An example tracer sNaN propagation is shown in Figure 3-18. When an FPU operation (Op1) executes with a sNaN and a normal floating-point number, the sNaN will be quieted and propagate as the result. In Figure 3-18, this is sNaN1 (the initial tracer) being propagated as qNaN1. Should a subsequent operation (Op2) execute with qNaN1 and, for example, a later sNaN tracer (sNaN2), operand qNaN1 will have priority, thereby maintaining propagation of the original tracer payload. However, should that qNaN1 value then be presented to another FPU operation (Op3) together with another qNaN, the qNaN result could be either of the source qNaNs, depending upon the magnitude of their respective significands.

However, if the significand of the initial sNaN1 tracer is large enough, it will ultimately be able to continue to propagate past all subsequent NaNs and be available to view at the end of the code block, thereby allowing it to be traced back to its source.

Figure 3-18. Tracker sNaN Operand Propagation Example



© 2023-2024 Microchip Technology Inc. and its subsidiaries

Table 3-14. FMAC NaN Propagation Priority

Multiply Source Operands		Add Source Operands		FMAC Result (Fd)	Notes
Fb or Fs	Fs or Fb	Intermediate Result	Accumulator Source (Fd)		
FP Multiply					
		FP Add			
FPN	FPN	FPN	FPN	FPN	
			qNaN	qNaN	
			sNaN	Quieted sNaN	INVAL signaled
		Distinguished qNaN(3)	FPN	Distinguished qNaN	INVAL signaled
			qNaN1	Distinguished qNaN or qNaN1(2)	INVAL signaled
			sNaN	Distinguished qNaN or Quieted sNaN(2)	INVAL signaled
FPN	sNaN1	Quieted sNaN1	FPN	Quieted sNaN1	INVAL signaled
			qNaN	Quieted sNaN1 or qNaN(2)	INVAL signaled
			sNaN2	Quieted sNaN1(2)	INVAL signaled
FPN	qNaN1	qNaN1	FPN	qNaN1	
			qNaN2	qNaN1 or qNaN22	
			sNaN	qNaN1	INVAL signaled
qNaN1	qNaN2	qNaN1 or qNaN2(2)	FPN	qNaN1 or qNaN2(2)	
			qNaN3	qNaN1 or qNaN2 or qNaN3(2)	
			sNaN	qNaN1 or qNaN2(2)	INVAL signaled
sNaN1	sNaN2	Quieted (sNaN1 or sNaN2)(2)	FPN	Quieted (sNaN1 or sNaN2)(2)	INVAL signaled
			qNaN	Quieted (sNaN1 or sNaN2)(2) or qNaN	INVAL signaled
			sNaN3	Quieted (sNaN1 or sNaN2)(2)	INVAL signaled

Notes:

1. FPN is a floating-point number that is not a NaN.
2. Using significand magnitude comparisons as defined in Table 3-13.
3. Distinguished qNaN intermediate result will arise when operands are 0 and Inf (any sign).

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6.3 Zero, Infinity, Not a Number (NaN) and Subnormal Values

The IEEE 754-2008/2019 standards reserve data encoding to represent special values, as shown in Figure 3-19.

Zero is conveyed when both exponent and mantissa are all 0's. Zero is a signed value (for some operations) as determined by the Sign bit. Infinity is conveyed by an exponent value of all 1's with an all 0's mantissa. Infinity is a signed value as determined by the Sign bit.

A Signaling NaN is conveyed by an exponent value of all 1's with the MSb of the mantissa set to 0 (remaining mantissa bits may be set to any value). The Quiet Nan (qNaN, see 3.6.3.1. Not a Number (NaN)) is conveyed by an exponent value of all 1's with the MSb of the mantissa set to 1 (remaining mantissa bits may be set to any value). NaN values are not signed, so the Sign bit may be any state.

A Subnormal value (see 3.6.3.2. Subnormal Number) is conveyed by an exponent of all 0's and any non-zero mantissa value. Subnormals are signed values as determined by the Sign bit.

3.6.3.1 Not a Number (NaN)

The Signaling NaN (sNaN) and Quiet NaN (qNaN) are specific data codes that indicate certain situations. In all cases, an exponent value of all 1's with a non-zero mantissa signifies a NaN (an exponent value of all 1's with an all 0's mantissa is used to convey Infinity).

qNaNs may be generated as the result of an invalid operation, such as taking the square root of a negative floating-point number. A qNaN will propagate through subsequent floating-point operations. Operations that will generate an Invalid exception for each instruction are documented in Table 3-18.

sNaNs are reserved input operands which, under default exception handling, will signal an Invalid exception when encountered. This may be used to indicate uninitialized variables, or as debug aids, but they are never generated by arithmetic computations or comparisons. Whenever the source operand of operation is an sNaN, the result will be a qNaN.

Both sNaNs and qNaNs can store "Payloads" in the mantissa bit field. The payload must not affect the MSB of the mantissa. The payload can be used as a debugging aid in tracing through complex arithmetic calculations.

3.6.3.1.1 qNaN and sNaN Propagation

The IEEE 754-2008/2019 standards indicates that source qNaNs should be propagated, including any associated payload. The FPU module does not propagate any source qNaNs, but instead generates fixed distinguished qNaN results.

In keeping with other device floating-point implementations, this module will propagate qNaN and sNaN values where possible. Refer to NaN Propagation for further detail.

For instructions where a source operand qNaN is not available, a distinguished qNaN value as shown below will be provided as the result whenever those instructions suffer a computational error.

- Single Precision: Distinguished qNaN = 0x7FC0_0001
- Double Precision: Distinguished qNaN = 0x7FF8_0000_0000_0001

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-19. Floating-Point Encodings

Single Precision

31	30 ... 23	22 ... 0	
S	Not all Ones Not all Zeros	Any Value	= ±Normal
S	00000000	00000000000000000000000	= ±0
S	00000000	Not All Zeros	= ±Subnormal
S	11111111	00000000000000000000000	= ±∞
x	11111111	0 ... Not All Zeros	= sNaN
x	11111111	1 ... Any Value	= qNaN

Note: sNaN and qNaN values may be of either sign.

Double Precision

63	62 ... 52	51 ... 0	
S	Not all Ones Not all Zeros	Any Value	= ±Normal
S	00000000000	0000000000000000000000000000000000000000000000000000	= ±0
S	00000000000	Not All Zeros	= ±Subnormal
S	11111111111	0000000000000000000000000000000000000000000000000000	= ±∞
x	11111111111	0 ..... Not All Zeros	= sNaN
x	11111111111	1 .... Any Value	= qNaN

Note: sNaN and qNaN values may be of either sign.

3.6.3.1.2 NaN Operands with Float-to-Integer Conversion

The FF2DI and FF2LI are the float-to-integer instructions. These instructions can output Huge Integer in lieu of Invalid when the source is a value that would convert to an integer outside the range of the result format under the applicable rounding attribute. This output is implemented as a new exception, Huge Integer (FSR.HUGI).

The IEEE 754-2008/2019 standard calls for Invalid to be signaled if this situation cannot otherwise be indicated. The FSR.HUGI exception is considered an implementation of “otherwise indicated,” making the FF2DI and FF2LI instructions compliant.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

The module drives status output Invalid (and does not drive Huge Integer) when the source is ±NaN, or ±∞ per the IEEE 754-2008/2019 standards. Note that Invalid is also driven for a qNaN input

3.6.3.2 Subnormal Number

A subnormal number (historically also referred to as a denormal number) is a *non-zero* floating-point number with a magnitude of less than that of the smallest normal number representable in the given format. The benefit of subnormal numbers is that they allow for gradual underflow when a result is very small (when compared to that without subnormal numbers). The IEEE 754 standard represents subnormal numbers as a special case.

Using Single Precision data format as an example, the smallest normal numbers around 0 are greater than $+2^{-126}$ or less than -2^{-126}, which occur when the floating-point number exponent is 1 (bearing in mind that the 8-bit exponent is defined with a bias of +127) and the mantissa is all 0's.

The exponent value of 0 is reserved for subnormal numbers. However, the IEEE 754 standard treats subnormal numbers as a special case where the hidden mantissa bit becomes 0 and the exponent bias is changed (by +1) to compensate, such that the datum exponent becomes -126. This allows the subnormal range to surround 0 and be between a little greater than -2^{-126} to a little less than -2^{-126}. That is:

$$-2^{-126} < subnormal < +2^{-126}$$

The minimum exponent value is referred to as Emin, and is -126 for Single Precision and -1023 for Double Precision formats. A subnormal number would therefore be represented as:

$$(-1)^{S} \times [0].m_{base2} \times 2^{Emin}$$

where:

-'*S*' indicates the sign of the number (same values as a signed integer value)

-'*m*' represents the fractional mantissa value

For example, the largest SP positive subnormal number will be when all mantissa bits are all set (0x007F_FFFF), and the smallest number will be when all mantissa bits are all clear (0x0000_0000), which is 0.0.

3.6.3.2.1 Subnormal Number Handling

Should any floating-point calculation generate a subnormal result, the FSR.UDF will be set; if it is not already set, the sticky status FSR.UDFS will also be set. In addition, if any instruction is presented with a subnormal operand value, FSR.SUBO will be set. If it is not already set, the sticky status FSR.SUBOS will also be set.

Subnormal Override Functions

Although not IEEE 754 compliant, subnormal operands and/or results may be overridden to improve the performance of some applications that do not require subnormal number precision. Use of the subnormal override function:

- Avoids the consequences of processing or having to deal with subnormal datum.
- Handles result underflows when a result is subnormal, negating the need to handle an underflow exception.

The subnormal override functions consist of two parts, one to flush subnormal input operands to zero (referred to as Subnormals-Are-Zeros, or SAZ mode), and the other to remove subnormal results (referred to as Flush-To-Zero, or FTZ mode).

Note: Subnormal override modes are not applicable to FCPS/FCPQ (no result to override), `FAND`, `FIOR`, `FTST`, `FMOV`, `FMOVC` or any CPU to/from FPU data move instruction.

Subnormals-Are-Zero (SAZ)

Subnormals-Are-Zero (SAZ) mode is enabled when FCR.SAZ is set and will ensure that any subnormal operand input to a Functional Block is replaced with a 0 value of the same sign as

© 2023-2024 Microchip Technology Inc. and its subsidiaries

the subnormal value it is replacing. This avoids the consequences of processing or having to deal with subnormal datum. This operation applies to all floating-point instructions except: `FMOV`, `FMOVC`, `FAND`, `FIOR`, `FTST`, `FLI2F` and `FDI2F`.

Note: SAZ mode is applied to `FABS` and `FNEG` instructions to ensure result consistency with that of an equivalent sequence of FPU arithmetic instructions.

Note: Does not apply to FPU to CPU or CPU to FPU move instructions.

3.6.3.2.2 Flush-To-Zero (FTZ)

Flush-To-Zero (FTZ) mode is enabled when both FCR.FTZ and FCR.UDFM are set. If the underflow exception is unmasked (FCR.UDFM = `0`), then the FCR.FTZ bit will have no effect. Should a floating-point operation generate an infinitely precise result that is less than the smallest possible subnormal number, then the Functional Block will round this to a result of 0 with the same sign as the subnormal value. This will occur irrespective of whether FTZ mode is enabled or not. Both Underflow (FSR.UDF) and Inexact (FSR.INX) will be signaled (if not already set, sticky status FSR.UDFS and FSR.INXS will also be set). Should a floating-point result be a subnormal number (that the Functional Block has not rounded up to the smallest magnitude normal number), and FTZ mode is enabled, the result will be replaced with 0 of the same sign as the subnormal value it is replacing. Again, both Underflow (FSR.UDF) and Inexact (FSR.INX) will be signaled (if not already set, sticky status FSR.UDFS and FSR.INXS will also be set), though the Underflow exception has to be masked (in order to enabled FTZ mode), so no interrupt will be issued. Forcing the result to 0 allows the user to ignore underflows (though at the expense of some accuracy).

The FCR.FTZ bit is only examined during the WB-stage of an instruction such that it may be modified as late as the cycle before the instruction enters the WB-stage. For example, the following code sequence will only apply the FTZ function to the `FSUB` instruction:

```
 * Assume FCR.FTZ=0 && FCR.UDFM=1 at entry
FADD.s F0, F1,F2                  ;add without FTZ
FIOR #0x0400, FCR                 ; set FCR.FTZ
FSUB.s F3, F4,F5                  ;sub with FTZ
FAND #0xFBFF, FCR                 ;clear FCR.FTZ
FADD.s F2,F6,F7                   ;add without FTZ
```

3.6.3.2.3 Subnormal Operand Exception

Should any affected instruction execute using a subnormal operand, and SAZ mode is disabled, the Subnormal Operand (FSR.SUBO) exception will be signaled. This provides a mechanism to indicate the use of a subnormal value without requiring the operand be tested (FTST).

Should SAZ mode be enabled and a subnormal operand is encountered (and changed to a 0 value), SUBO will not be signaled.

SAZ mode may be enabled irrespective of whether the SUBO exception is masked or not (though when enabled will never signal SUBO).

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6.4 Floating-Point Data Register (F0-F31)

Name: Fn

Bit	31	30	29	28	27	26	25	24
	Fn[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	Fn[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	Fn[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	Fn[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 31:0 – Fn[31:0] Floating-Point Data Register bits

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6.5 Floating-Point Control Register

Name: FCR

Note:

1. Floating-Point Exception Mask bits, FCR [6:0]: Each Exception Mask bit corresponds to an Exception Status flag in the FSR. The Mask bit must be clear to allow the exception event to generate an interrupt to the CPU. The Underflow Mask bit (FCR.UDFM) is also used as part of the Flush-to-Zero (FTZ) mode enable as discussed in 3.6.3.2.2. Flush-To-Zero (FTZ).

   Floating-point rounding mode control, FCR [9:8]: These bits define the global IEEE 754 compatible rounding mode used by the FPU instruction. 3.6.8.9.3. Rounding Modes.

   Floating-point subnormal override mode control, FCR [11:10]: These bits enable the Subnormals-Are-Zero (SAZ) and Flush-to-Zero (FTZ) subnormal override modes supported by the FPU.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
					SAZ	FTZ	RND [1:0]	
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
		SUBOM	HUGIM	INXM	UDFM	OVFM	DIV0M	INVALM
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit 11 – SAZ Subnormals Are Zero Operand Mode bit

Value	Description
1	Subnormals Are Zero mode is enabled
0	Subnormals Are Zero mode is disabled

Bit 10 – FTZ Flush To Zero Result Mode bit

Value	Description
1	Flush To Zero mode is enabled
0	Flush To Zero mode is disabled

Bits 9:8 – RND [1:0] FPU Rounding Mode bit

Value	Description
11	IEEE Round to Negative Infinity (floor)
10	IEEE Round to Positive Infinity (ceiling)
01	IEEE Round to Zero (truncate)
00	IEEE Round to Nearest (even)

Bit 6 – SUBOM Subnormal Operand Exception Mask bit

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Value	Description
1	Subnormal exception is masked
0	Subnormal exception is not masked

Bit 5 – HUGIM Huge Integer Exception Mask bit

Value	Description
1	Huge Integer exception is masked
0	Huge Integer exception is not masked

Bit 4 – INXM Inexact Exception Mask bit

Value	Description
1	Inexact exception is masked
0	Inexact exception is not masked

Bit 3 – UDFM Underflow Exception Mask bit

Value	Description
1	Underflow exception is masked
0	Underflow exception is not masked

Bit 2 – OVFM Overflow Exception Mask bit

Value	Description
1	Overflow exception is masked
0	Overflow exception is not masked

Bit 1 – DIV0M Divide By Zero Exception Mask bit

Value	Description
1	Divide By Zero exception is masked
0	Divide By Zero exception is not masked

Bit 0 – INVALM Invalid Exception Mask bit

Value	Description
1	Invalid exception is masked
0	Invalid exception is not masked

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6.6 Floating-Point Status Register

Name: FSR

Note: Dynamic floating-point exception status, FSR [6:0]: Dynamic status bits are updated based on the results from each instruction Functional Block and will be updated after execution of each instruction.

Sticky floating-point exception status, FSR [14:8]: Sticky status bits can be set based on the results from each instruction Functional Block but cannot be cleared by hardware (other than at device Reset), and therefore represent a history of status since the last time the sticky bits were cleared. The FSR bits can be cleared through software.

Floating point compare status, FSR [19:16]: Status generated by executing a floating-point compare (FCPQ/FCPS) instruction. Used individually or combined to generate the floating-point branch conditions used by the CPU CBRAn instructions.

Floating-point test status, FSR [27:24]: Floating-point datum characteristic status generated by executing the floating-point test (FTST) instruction.

Bit	31	30	29	28	27	26	25	24
				SUB	INF	FN	FZ	FNAN
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
					GT	LT	EQ	UN
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit	15	14	13	12	11	10	9	8
		SUBOS	HUGIS	INXS	UDFS	OVFS	DIV0S	INVALS
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
		SUBO	HUGI	INX	UDF	OVF	DIV0	INVAL
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit 28 – SUB (FTST) Subnormal Status bit

Value	Description
1	Operand is subnormal
0	Operand result is not subnormal

Bit 27 – INF (FTST) Infinite Status bit

Value	Description
1	Operand is infinite
0	Operand is not infinite

Bit 26 – FN (FTST) Negative Status bit

Value	Description
1	Operand is negative
0	Operand is not negative

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Bit 25 – FZ (FTST) Zero Status bit

Value	Description
1	Operand is zero
0	Operand is not zero

Bit 24 – FNAN (FTST) Not a Number Status bit

Value	Description
1	Operand is a NaN (qNaN or sNaN) value
0	Operand is not a NaN value

Bit 19 – GT (FCPS/FCPQ) Greater Than Status bit

Value	Description
1	Minuend is greater than the subtrahend (Fb > Fs)
0	Minuend is not greater than the subtrahend (Fb ≤ Fs)

Bit 18 – LT (FCPS/FCPQ) Less Than Status bit

Value	Description
1	Minuend is less than the subtrahend (Fb < Fs)
0	Minuend is not less than the subtrahend (Fb ≥ Fs)

Bit 17 – EQ (FCPS/FCPQ) Equal Status bit

Value	Description
1	Minuend is equal to the subtrahend (Fb = Fs)
0	Minuend is not equal to the subtrahend (Fb != Fs)

Bit 16 – UN (FCPS/FCPQ) Unordered Status bit

Value	Description
1	Either or both operands are NaN values
0	Neither operands are NaN values

Bit 14 – SUBOS Sticky Subnormal Operand Exception Flag bit

Value	Description
1	Subnormal Operand exception has just occurred, or at some time in the past
0	Subnormal Operand exception has not occurred

Bit 13 – HUGIS Sticky Huge Integer Exception Flag bit

Value	Description
1	Huge Integer exception has just occurred, or at some time in the past
0	Huge Integer exception has not occurred

Bit 12 – INXS Sticky Inexact Exception Flag bit

Value	Description
1	Inexact exception has just occurred, or at some time in the past
0	Inexact exception has not occurred

Bit 11 – UDFS Sticky Underflow Exception Flag bit

Value	Description
1	Underflow exception has just occurred, or at some time in the past
0	Underflow exception has not occurred

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Bit 10 – OVFS Sticky Overflow Exception Flag bit

Value	Description
1	Overflow exception has just occurred, or at some time in the past
0	Overflow exception has not occurred

Bit 9 – DIV0S Sticky Divide by Zero Exception Flag bit

Value	Description
1	Divide by Zero exception has just occurred, or at some time in the past
0	Divide by Zero exception has not occurred

Bit 8 – INVALS Sticky Invalid Exception Flag bit

Value	Description
1	Invalid exception has just occurred, or at some time in the past
0	Invalid exception has not occurred

Bit 6 – SUBO Subnormal Operand Exception Flag bit

Value	Description
1	Subnormal Operand exception has occurred
0	Subnormal Operand exception has not occurred

Bit 5 – HUGI Huge Integer Exception Flag bit

Value	Description
1	Huge Integer exception has occurred
0	Huge Integer exception has not occurred

Bit 4 – INX Inexact Exception Flag bit

Value	Description
1	Inexact exception has occurred
0	Inexact exception has not occurred

Bit 3 – UDF Underflow Exception Flag bit

Value	Description
1	Underflow exception has occurred
0	Underflow exception has not occurred

Bit 2 – OVF Overflow Exception Flag bit

Value	Description
1	Overflow exception has occurred
0	Overflow exception has not occurred

Bit 1 – DIV0 Divide by Zero Exception Flag bit

Value	Description
1	Divide by Zero exception has occurred
0	Divide by Zero exception has not occurred

Bit 0 – INVAL Invalid Exception Flag bit

Value	Description
1	Invalid exception has occurred
0	Invalid exception has not occurred

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6.7 Floating-Point Exception Address Capture Register

Name: FEAR

Note:
1. FEAR [1] always set to 1'b0.

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
	FEAR[22:15]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	FEAR[14:7]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	FEAR[6:0]							EACE
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 23:1 – FEAR[22:0] Floating-Point Instruction Exception Address Capture Register bits[(1)]

Bit 0 – EACE Exception Address Capture Enable bit

Value	Description
1	FEAR register address capture enabled
0	FEAR register address capture disabled (and FEAR [23:0] may contain a captured address)

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6.8 FPU Module Operation

Figure 3-20. Module Block Diagram

3.6.8.1 Floating-Point Unit Registers

The dsPIC Floating-Point Unit (FPU) provides a large set of working registers (F-regs):

- 32 x 32-bit (Single Precision, F0 ... F31) or
- 16 x 64-bit (Double Precision, F0, F2 ... F28, F30) or
- A mix of the two sizes aligned as shown in Figure 3-21.

In addition to the F-regs, status (FSR) and control (FCR) registers are also supported as shown in Figure 3-21:

- FSR (FPU Status Register, 32-bit): Holds the status of retired floating-point instructions:
  – FSR [6:0]: Instruction “most-recent” exception status
  – FSR [14:8]: Instruction "sticky" exception status
  – FSR [19:16]: FCPS/FCPQ instruction status
  – FSR [28:24]: FTST instruction status
- FCR (FPU Control Register, 16-bit):
  – FCR [6:0]: Exception mask control
  – FCR [9:8]: Rounding mode control
  – FCR [10]: Subnormal result “Flush-to-Zero” (FTZ) control
  – FCR [11]: Subnormal operand “Subnormals-are-Zero” (SAZ) control
- FEAR: (FPU Exception Address Capture Register, 24-bit): Holds the address of the first instruction encountered that causes an exception. All subsequent instructions in the FPU pipeline that

© 2023-2024 Microchip Technology Inc. and its subsidiaries

subsequently retire will not affect the FEAR, even if they too generate exceptions. The FEAR is intended for use during debug of the floating-point software.

Note: The FSR msws and lsws may be read/written independently of each other by some instructions.

Note: Although inconsistent with device interrupts, where interrupt controls are referred to as enables (where logic 1 represents enabled), it is more conventional (and in keeping with the IEEE-754 specification) that the FPU exception controls be referred to as masks (where logic 1 represents masked). These bits are all set at Reset, masking exceptions by default.

3.6.8.1.1 FPU Register Access

Data may be moved in and out of any FPU register, from OR to W-regs or DS memory, by using dedicated coprocessor register move instructions that execute from within the integer pipeline (refer to `MOVCRW`, `MOVWCR`, `MOVLCR`, `LDWLOCR`, `STWLOCR`, `PUSHCR` and `POPCR` CPU instructions as described in 3.6.8. FPU Module Operation).

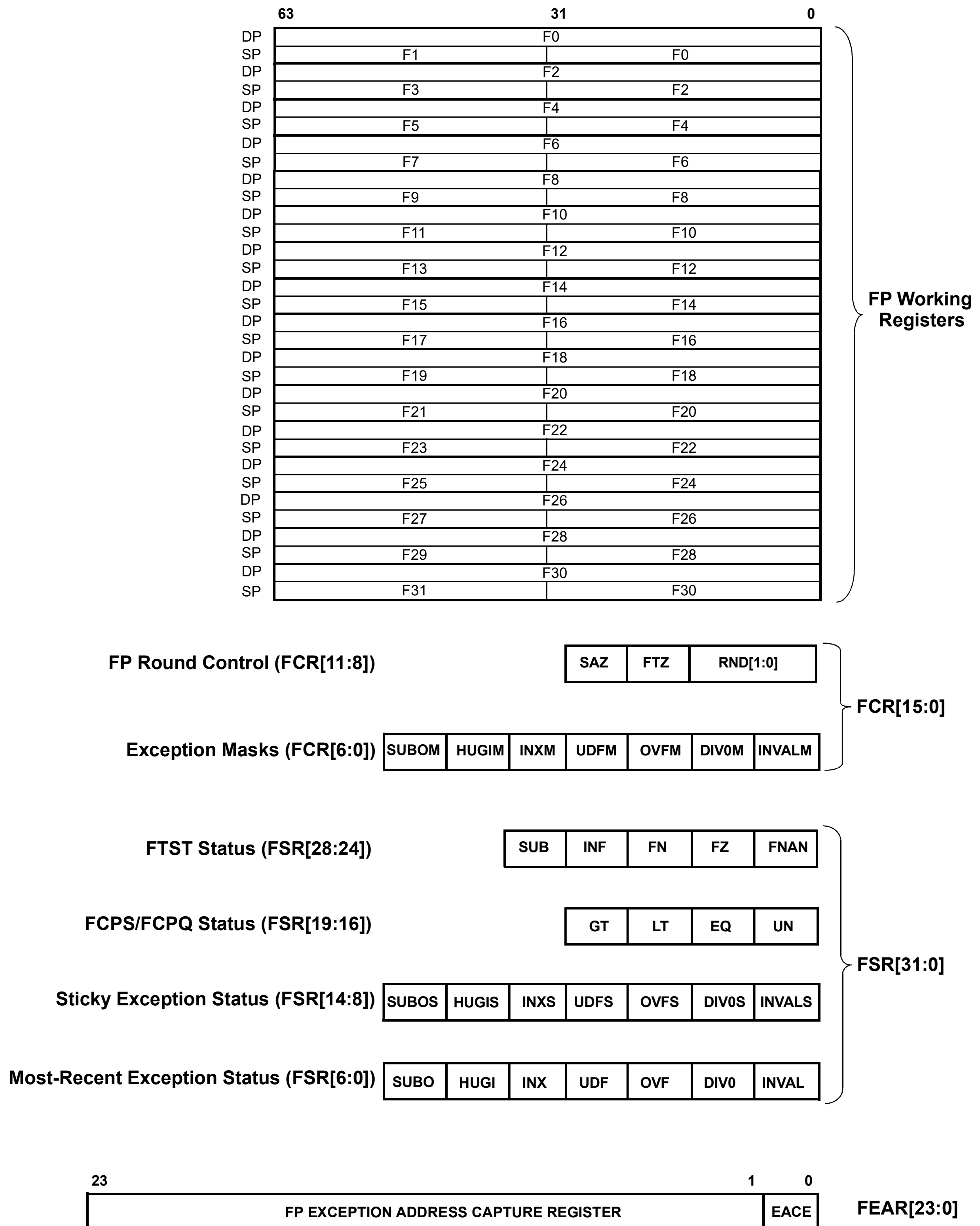
All data is moved as 32-bit entities, so Double Precision data moves will require the execution of two instructions (64-bit data moves are not supported in this device).

In addition, the FPU supports `FAND` and `FIOR` instructions that can logically AND or OR a literal value with the FSR (lsw only, exception status), FCR or FEAR (lsw only).

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6.8.2 FPU Programmer's Model

Figure 3-21. FPU Programmer's Model



© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6.8.3 FPU Register Set

The FPU Programmer's Model of registers is shown in Figure 2-1 and is comprised of floating-point operand registers (F-regs), a floating-point control register (FCR), a floating-point status register (FSR), and a floating-point exception address capture register (FEAR). None of the registers are memory mapped and must be read or written by the CPU using the coprocessor move instructions (`MOVCRW`, `MOVWCR`, `PUSHCR`, `POPCR`, `LDWLOCR`, `STWLOCR`, and `MOVLCR`). The FCR, FSR and FEAR registers may also be subjected to a literal AND or OR operation by the `FAND` and `FIOR` instructions respectively.

3.6.8.3.1 Floating-Point Operand Registers (F-Regs)

To differentiate from the CPU working W-regs, the FPU operand/result data working registers are referred to as F-regs. The FPU supports up to 32 Single Precision values, or up to 16 Double Precision values. Aligned pairs of the F-regs registers (e.g., F1:F0 values may be used to provide data storage for Double Precision values. Single and Double Precision values may be mixed within the register file. Other than data movement in and out of the FPU, all instructions are register-to-register operations within the FPU register set.

The F-regs are not memory-mapped and can only be accessed by the CPU using specific instructions as discussed in 3.6.8.3.3. CPU Access of FPU Registers.

The 32 x 32-bit F-reg array, together with additional register contexts, is implemented as a register file. FPU instructions can have 1, 2 or 3 operands (read sources) and 0 or 1 result destination, and most also update status in the FSR. Registers may be used individually for Single Precision data values or coupled as odd; even pairs (only) should be used to support Double Precision data values (e.g., F1:F0).

Source registers are bound to an instruction when the instruction is issued and are not writable by the CPU until the instruction is committed. At this point, they are clocked into operand registers that drive the target functional block and can therefore be subsequently written. Note that a bound source register may be read at any time.

Destination registers (F-regs and FSR) are bound to an instruction when the instruction is committed and are not accessible by the CPU until the instruction has retired.

Floating-Point Control Register (FCR)

The FCR is comprised of the following bit fields as defined in 3.6.5. FCR.

Floating-Point Exception Mask bits, FCR [6:0]: Each Exception Mask bit corresponds to an Exception Status flag in the FSR. The Mask bit must be clear to allow the exception event to generate an interrupt to the CPU. The Underflow Mask bit (FCR.UDFM) is also used as part of the Flush-to-Zero (FTZ) mode enable as discussed in 3.6.3.2.2. Flush-To-Zero (FTZ).

Floating-Point Rounding mode control, FCR [9:8]: These bits define the global IEEE 754 compatible rounding mode used by the FPU instruction. See 3.6.8.9.3. Rounding Modes.

Floating-Point Subnormal Override mode control, FCR [11:10]: These bits enable the Subnormals-Are-Zero (SAZ) and Flush-to-Zero (FTZ) subnormal override modes supported by the FPU.

3.6.8.3.2 Floating-Point Unit Register Contexts

To speed up real time control systems and other time critical applications, the dsPIC FPU supports multiple register contexts that are tied to Interrupt Priority Levels.

The FPU includes a set of hardware register contexts. Each context includes the FSR, FCR and four register pairs (i.e., F0 through F7). All other F-regs and FEAR are not included and must be saved and restored through software.

The number of supported register contexts matches that of the CPU and is fixed at seven, which represents one context per CPU Interrupt Priority Level (IPL). Should the CPU change context, then the FPU will follow suit, and all subsequent instructions issued to the FPU will execute within that (new) context. However, all FPU instructions issued in a prior context will be allowed to continue to execute and retire within that context, irrespective of the context change within the CPU. Similarly,

© 2023-2024 Microchip Technology Inc. and its subsidiaries

any data dependencies that occur within the context of the instruction underway will remain within that context.

As the FSR is part of the register context, exceptions are context specific. Should the FPU change register context, any FPU exceptions generated as a result of the execution of FPU instructions already issued from the prior context will remain pending until the FPU returns to that original context.

Hazard detection is also context based such that each instruction operand and result register is tagged with its own context. Hazards can therefore only exist within the same register context.

This concept extends to the FSR and FCR which have an independent representation within each register context. Consequently, the CPU will not stall (assuming no FSR and/or FCR hazard exists within the current context) if it were to access the FSR or FCR while the FPU continued to execute instructions issued from within a different context. These instructions would have access to their own version of the FSR and FCR.

3.6.8.3.3 CPU Access of FPU Registers

The following CPU instructions are provided specifically to support data movement into and out of the coprocessors. The assembler uses the register declarations to direct encoding of the FPU as the target coprocessor within each instruction op code:

- `MOVCRW`: Move any FPU register to a W-reg or DS memory (using indirect addressing).
- `LDWLOCR`: Move the contents of DS memory (read using register+literal offset addressing (`[Ws + Slit14]`)) to any FPU F-reg register.
- `STWLOCR`: Move any FPU F-reg to DS memory (read using register+literal offset addressing (`[Wd+Slit14]`)).
- `PUSHCR`: 16-bit short instruction dedicated to moving any FPU register onto the system stack.
- `MOVWCR`: Move a W-reg or DS memory value (using indirect addressing) to any FPU register.
- `POPCR`: 16-bit short instruction dedicated to moving a value from the system stack to any FPU register.
- `MOVLCR`: Move a 32-bit literal value to any FPU register.

Note: These instructions are referred as `mov.l`, `push.l` or `pop.l`. Please refer to the dsPIC33A Programmer's Reference Manual for the correct syntax of these instructions.

3.6.8.3.4 Intra-FPU Register Moves and Logical Operations

In addition to CPU to/from FPU data movement, the FPU supports instructions that execute within its own pipeline that perform register to register moves or logical operations:

- `FMOV`: Copy any F-reg or F-reg pair into another F-reg or F-reg pair.
- `FMOVC`: Move one of 32 Single or Double Precision constant values into an F-reg or F-reg pair.
- `FAND`: Logically AND a 16-bit literal value (lit16) with the lsw of the FPU FSR, FCR or FEAR.
- `FIOR`: Logically OR a 16-bit literal value (lit16) with the lsw of the FPU FSR, FCR or FEAR.

Note: To allow subsequent instruction to immediately utilize `FAND` and `FIOR` changes to FCR.RND[1:0] and FCR.SAZ control bits without stalls, these bits are manipulated and updated in the first pipeline stage (RD-stage). However, the remaining FCR bits are not written back until the end of the instruction as usual. Consequently, should the CPU need to read the FCR immediately after modification, it will be stalled by the FPU until the `FAND` or `FIOR` instruction has retired.

3.6.8.4 Data Hazard Management

Read-After-Write (RAW) data hazards can arise due to:

- Data dependencies between FPU instructions

© 2023-2024 Microchip Technology Inc. and its subsidiaries

- As the result of a register move from an FPU register to the CPU when an FPU instruction underway has not yet completed its result write (to the same register)

Write-After-Read (WAR) data hazards within the FPU pipeline alone are not possible because the pipeline ensures that instruction reads always precede subsequent instruction writes. However, a WAR hazard can arise when the CPU pipeline writes to an FPU register that has yet to be read by a previously issued but stalled FPU instruction.

Write-After-Write (WAW) data hazards are possible should the CPU attempt to write to an FPU register that is also the target of a prior FPU instruction which has not yet completed its result write.

All hazards are detected within the FPU or CPU (or both) and will be mitigated either through data forwarding or pipeline stalls. Refer to 3.6.8.7. FPU Hazards for further details.

3.6.8.5 FPU and CPU Exceptions

Issued FPU instructions that become committed (accepted by the Execute stage) are always atomic with respect to CPU exceptions. No CPU exception (other than a Reset event) can force the FPU to abandon an instruction that is already underway.

CPU exceptions will result in a register context switch in both the CPU and FPU. Furthermore, FPU exceptions are always context specific. That is, any FPU exception occurring after a context switch will remain pending until the FPU returns to the prior context.

FPU exceptions can only be taken and handled when unmasked (referred to as alternate exception handling). The FPU will return the calculated result of each operation and signal any exception via an interrupt to the CPU.

If FPU exceptions are masked, the FPU will return a default result for each operation that generates an exception as defined in Table 3-15. The exception will be signaled by setting the corresponding bit(s) in the FSR, but no interrupt will be issued to the CPU. This is intended to allow code to execute unhindered by exception handling at the time of execution. If required, exception status may be examined at a later time and appropriate action taken.

3.6.8.5.1 Huge Integer and Subnormal Exceptions

In addition to the IEEE 754-2008/2019 compliant exception support, this macro also offers two additional exceptions and associated masks that some users may find useful.

- Huge Integer: FSR.HUGI
  Exception signaled whenever a Float-to-Integer conversion operation (FF2DI and FF2LI) results in an integer value that is larger than the destination register can represent.
- Subnormal Operand: FSR.SUBO
  Exception signaled whenever an operand of an affected instruction is a subnormal value and Subnormals-Are-Zeros (SAZ) mode is disabled (FCR.SAZ = 0). This is the only exception that can be triggered by an operand source condition (all others are related to result conditions).

Table 3-15. Default Exception Results

Exception	FSR Bit Name	Default Result
Invalid	INVAL[2]	Distinguished qNaN or quieted sNaN or Largest integer result (for FF2DI/FF2LI only)
Divide By Zero	DIV0	Correctly signed Infinity[3]

Notes:

1. Under default exception handling, UDF is only set (along with INX) if the result is an inexact underflow. Applies irrespective of whether FTZ mode is enabled or not.
2. FCPS and FCPQ do not generate a result other than an FSR update. However, INVAL will be set by FCPS if either or both operands are a qNaN or sNaN, or by FCPQ if either or both operands are an sNaN.
3. 0/0 is a special case (where both the dividend and divisor are not finite) which will return the distinguished qNaN as the result. INVAL will be set but DIV0 will not.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

...........continued

Exception	FSR Bit Name	Default Result		
Overflow	OVF	Rounding Mode	Nearest (Even)	Infinity with sign of exact result
			Zero	Most positive finite number with sign of exact result
			+Infinity	Positive overflow: +Infinity Negative overflow: Most negative finite number
			-Infinity	Positive overflow: Most positive finite number Negative overflow: -Infinity
Underflow	UDF[1]	FCR.FTZ = `0`: Rounded subnormal result		
		FCR.FTZ = `1`; Zero with sign of exact result		
Inexact	INX	Rounded (inexact) result		
Huge Integer	HUGI	Largest integer value with sign of input operand		
Subnormal Operator	SUBO	N/A (input operand exception)		

Notes:

1. Under default exception handling, UDF is only set (along with INX) if the result is an inexact underflow. Applies irrespective of whether FTZ mode is enabled or not.
2. FCPS and FCPQ do not generate a result other than an FSR update. However, INVAL will be set by FCPS if either or both operands are a qNaN or sNaN, or by FCPQ if either or both operands are an sNaN.
3. 0/0 is a special case (where both the dividend and divisor are not finite) which will return the distinguished qNaN as the result. INVAL will be set but DIV0 will not.

3.6.8.6 CPU to FPU Interface

The CPU can issue instructions to a coprocessor (FPU), and directly read and write FPU registers. However, coprocessors otherwise operate independently of the CPU instruction pipeline, executing their instructions within their own pipeline hardware.

An FPU can only receive, send and process data that is funneled through (and under the direction of) the CPU. No CPU addressing capability is shared with an FPU. Consequently, an FPU can only support register direct addressing for all instruction source or destination addressing modes that target a FPU register. Data flow to and from each FPU is controlled using dedicated move instructions that execute within the CPU. Because the CPU and FPU pipelines execute independently, data related hazards that may arise when moving data between the CPU and an FPU are mitigated using a simple request/grant bus which will stall the CPU as needed.

The CPU supports speculative execution of instructions that immediately follow a conditional branch. These could be FPU instructions, so a mechanism exists to allow the CPU to cleanly kill these instructions should the branch prediction prove incorrect.

In case an FPU SFR read is killed, all FPU SFR (e.g., status and control registers) are defined such that a read of any SFR is not destructive within itself. This will avoid the possibility of a killed SFR read affecting the state of the FPU.

3.6.8.6.1 FPU Pipeline Operation

The CPU decodes all coprocessor instructions during the F-stage. The source and destination coprocessor registers are extracted from the opcode and supplied to the coprocessor, along with a corresponding instruction select and control signals such that no instruction decode is necessary within the coprocessor.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

The FPU pipeline stages consist of Read (RD), Execute (X[n]) and Write-Back (WB) stages. The Read and Write-Back stages consist of a single register and are common to all instructions. The Execute stage consists of as many stages as required to execute the specific instruction (i.e., X [0], X[1]..... X[n]) but at least X [0].

The CPU pipeline F-stage and A-stage fetch can issue FPU instructions respectively as shown in Figure 3-23. The pipeline can suffer both structural and data hazards, as discussed later in 3.6.8.7.1. FPU Structural Hazards and FPU Functional Block Unavailable, respectively, which can result in CPU and FPU pipeline stalls, as shown in the corresponding diagrams.

One instruction may be issued into the RD-stage, where it will remain for one cycle (hazards aside) until dispatched into the X [0] stage. The number of cycles each instruction remains within the execute phase varies depending upon the operation. In order to avoid stalling the pipeline for the duration of any long instruction, up to four instructions may be dispatched into X[0] and executed concurrently (structural hazards aside).

Instructions retire in the same order in which they are issued. As a consequence of being able to execute multiple instructions with varying execution times, the pipeline Instruction/Hazard Tracker logic is designed to ensure that in-order retirement is maintained.

All instructions with an execution latency of four cycles or less are implemented such that the execution stages are fully pipelined. Consequently, assuming no data dependencies (hazards) arise, these instructions can be repeatedly issued at a rate of one per cycle (and receive their results at a rate of one per cycle after an initial execution latency), without incurring a structural hazard stall.

For instructions where the execution latency exceeds four cycles (FDIV and FSQRT), the FPU pipeline will fill the instruction and then stall subsequent instructions (due to a structural hazard) until the required execution resource becomes available.

- FDIV: Floating-point divide is implemented as an iterative operation such that the input data cannot be pipelined until all iterations have completed and the result is passed onto the adjustment stage within the Functional Block. For example, should the CPU issue two sequential FDIV instructions, the second FDIV instruction will stall in the RD-stage until the first FDIV enters the final execution cycle, at which point the second FDIV may be dispatched into execute stage to commence execution.
- FSQRT: Floating-point square root requires 10 (Single Precision) or 13 (Double Precision) cycles to execute. The hazard tracker can handle up to four issued instructions, so an FSQRT followed by up to three sequential FPU instructions (including FSQRT) may be executing at any one time. The CPU may issue one more instruction, but it will remain in the RD-stage until the oldest FSQRT instruction underway enters the WB-stage, six cycles later, and subsequently retires. At this point, one slot within the hazard tracker is now available for use, and the pending FPU instruction will be committed for execution. Another FSQRT instruction will retire in the next cycle, opening another hazard tracker slot for another issued FSQRT instruction, and so forth, until the hazard tracker is full again and the pipeline must wait a further six cycles for the initial FSQRT to retire. For FSQRT alone, the best case block repeat rate is therefore one per cycle for the initial 4 FSQRT instructions issued, with a subsequent four FSQRT instructions to be issued after six (Single Precision) or nine (Double Precision) cycles have passed. This supports an average execution time of (4+6)/4 or 2.5 cycles/instruction (Single Precision) or (4+9)/4 or 3.25 cycles/instruction (Double Precision).

FPU Read Stage

The FPU pipeline RD-stage receives instructions issued by the CPU. The CPU issues FPU instructions from the A-stage into the FPU RD stage which consists of a single register, such that only one FPU instruction can be held at any one time. The instruction is committed when it is dispatched to X[0], where it will start execution. X[0] holds the instruction such that the CPU is free to issue another instruction into the RD-stage.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

The RD-stage is also subject to hazard checks and can therefore be stalled. Should a RAW hazard be detected with a prior instruction that is already executing within the FPU pipeline, the hazard will be detected in the RD-stage which will then be stalled until such time that the hazard is resolved.

Should the CPU subsequently attempt to issue additional FPU instructions, the RD-stage will not be able to accept them so will also stall the CPU until such time that the RAW hazard has been resolved. From the CPU perspective, this scenario is viewed as a structural hazard.

The RD-stage will also stall the CPU under the following conditions:

1. Whenever the number of instructions (default value is four) are in their execute X[n] stages, an instruction is pending in the RD-stage, and the CPU is attempting to issue a further instruction. In this situation, the Instruction/Hazard Tracker is full so the FPU cannot dispatch another instruction from the RD-stage into X [0] until one of the instructions currently executing passes into the WB-stage (refer to 3.6.8.7.1. FPU Structural Hazards). Assuming the default value is four, this can occur when the pipeline is executing instructions that take longer than four cycles to execute, and additional FPU instructions are issued while the long instruction is still executing (i.e., not yet in the WB-stage). The longer instruction(s) execute and retire at a rate which is slower than the rate at which the Instruction/Hazard Tracker can be filled, resulting in the CPU being stalled.
2. Whenever the CPU attempts to issue more than two FDIV instructions while a previously issued and dispatched FDIV instruction is still executing (i.e., not yet in the WB-stage). FDIV is a special case where no more than one instance can be executed within the pipeline at any one time. Consequently, executing another FDIV while a prior instance is still executing will cause this second FDIV to be issued but held pending in the RD-stage (i.e., CPU will not stall). But attempting to issue a third FDIV instruction while the pending (second) instance has not yet been dispatched to X [0], will result in a CPU (issue) stall. The RD-stage also includes special logic to support the FAND and FIOR operations (refer to FAND and FIOR Instructions).

FPU Execute Stage

Each instruction may consist of one or more execute stages depending upon the functional block targeted by the operation. When the instruction enters the X [0]-stage, it is registered such that the RD-stage is free to receive another instruction issued by the CPU.

All instructions (other than FDIV) are pipelined through as many X[n] stages as deemed necessary to meet the timing requirements.

The pipeline stages will be added such that the propagation delay of each is as balanced as possible, and that sequential issue of the same instruction may be fully pipelined (i.e., instructions using the same Functional Block may be sequentially issued without incurring a structural hazard in the execute stage).

FPU Write-Back Stage

The WB-stage captures each Single Precision or Double Precision result as they exit the execute stage in dedicated registers. FPU instruction execution time is variable, but only one instruction is permitted to be in the WB-stage at any one time. If more than one instruction has completed execution and is in a position to retire, the pipeline will determine which instruction to retire to maintain instruction execution order and eliminate any WAW hazards. The instruction will then complete the write-back in one cycle during the WB-stage before being retired. The Instruction/Hazard Tracker logic will ensure instructions enter the WB-stage in the same order as they were issued (refer to 3.6.8.7.3. Instruction/Hazard Tracker).

Prior to writing the result, if FTZ mode is enabled (see 3.6.3.2.2. Flush-To-Zero (FTZ)), the result is modified accordingly if subnormal. This final value is also passed onto the RAW hazard mitigation forwarding logic (see Internal RAW Hazards).

© 2023-2024 Microchip Technology Inc. and its subsidiaries

FAND and FIOR Instructions

The FAND and FIOR instructions operate with a 16-bit literal, and can only target the FCR, FSR and FEAR. They are considered a special case as they are executed using custom blocks that are implemented within the RD-stage for some FCR bits and the WB-stage for everything else.

To allow subsequent instruction to immediately use FAND and FIOR changes to FCR.RND [1:0] and FCR.SAZ control bits without (RAW hazard) stalls, these bits are modified during the RD-stage, then updated at the end of the RD-stage such that they are available for immediate use by any subsequent instruction.

The remaining FCR bits and all FSR and FEAR bits are read, modified and written back during the WB-stage. Reading the FSR late (i.e., in the WB-stage rather than the RD-stage) avoids a potential RAW hazard arising between a prior instruction FSR update and a subsequent FAND or FIOR FSR operation.

3.6.8.7 FPU Hazards

The coprocessor interface can suffer from structural and data dependencies as described in the following sections. RAW, WAR and WAW data hazards are possible; RAR hazards are not.

3.6.8.7.1 FPU Structural Hazards

When a requested FPU resource is unavailable, a structural hazard will be detected. This may result in the coprocessor stalling the CPU until the hazard is resolved.

Hazards that arise from actions within the FPU are referred to as “internal” hazards. Those that arise due to actions between the CPU and FPU are referred to as “external” hazards. Depending upon how the CPU/FPU pipeline is viewed (separate or conjoined), some of these hazards may be viewed as either structural (i.e., a resource is unavailable) or data related.

FPU Pipeline Full or Busy

When the CPU attempts to issue an instruction to the coprocessor and it is unable to accept it because the pipeline is full or busy, an external structural hazard will result, and the coprocessor will stall the CPU until such time that the instruction can be accepted.

When an issued instruction is stalled in the FPU RD-stage due to a RAW hazard with a prior currently executing instruction, the FPU pipeline is considered busy such that further FPU instructions cannot be accepted. Consequently, should the CPU attempt to issue any additional FPU instructions while the RD-stage is stalled, the FPU will stall the CPU until such time that the hazard resolves, resulting in an external structural hazard as shown in Figure 3-24.

The pipeline is considered full when the Instruction/Hazard Tracker FIFO is full, which occurs when the number of instructions (default value is four) are active within it, including the one waiting in the RD-stage for dispatch into X[0]. The pipeline will remain full until the oldest instruction enters the WB-stage. Should the CPU attempt to issue another FPU instruction, the FPU will stall the CPU until such time that the Instruction/Hazard Tracker FIFO is no longer full.

FPU Functional Block Unavailable

If the FPU pipeline is not full, and the FPU attempts to dispatch an instruction from the RD-stage that uses a functional block that is already in use by a prior instruction, an internal structural hazard will result, and the RD-stage will be stalled until such time that the functional block is no longer in use. If the CPU attempts to issue another FPU instruction before this occurs, the FPU will then stall the CPU until the hazard resolves.

This scenario can arise as a result of in-order retirement where instructions that target the same functional block will be stalled in the pipeline waiting for slower, older instructions to complete execution. An example is shown in Figure 3-3 where a slow instruction (FSIN) is followed by multiple instructions that target the same MISC_SP functional block. The first FMOV will stall in X [0] waiting for the FSIN to retire, resulting in an internal structural hazard. The subsequently issued FMOV will issue but will be stalled in the RD-stage because it cannot progress into X [0] until the first FMOV is able to move into the WB-stage, another internal structural hazard. As the RD-stage is now stalled,

© 2023-2024 Microchip Technology Inc. and its subsidiaries

should the CPU attempt to issue any additional FPU `FMOV` or `FMOVC` instructions (which share the same functional block), the FPU will stall the CPU until such time that the pipeline can advance again, causing an external structural hazard.

This scenario will always arise for sequential issue of the multi-cycle iterative FDIV instruction (all other instructions can be pipelined) as shown in Figure 3-4.

FPU Register Unavailable to Read

When the CPU attempts to read a register that is bound to an issued FPU instruction, an external structural hazard will result and the coprocessor will not be able to read until such time that the register becomes available, creating a read stall for the CPU.

FPU Register Unavailable to Write

When the CPU attempts to write to a register that is bound to an issued FPU instruction, the co-processor will not be able to write until such time that the register becomes available, creating a write stall for the CPU (see FPU WAW Hazards).

3.6.8.7.2 FPU Data Hazards

The coprocessor interface can suffer from data dependencies leading to RAW, WAR and WAW data hazards (RAR hazards are not possible). Unlike the CPU integer pipeline, a coprocessor hazard does not necessarily prevent the pipeline from progressing for other coprocessor instructions, unless subject to other hazards.

Internal RAW Hazards

An internal RAW (Read-After-Write) data dependency hazard will occur when the result of an FPU instruction is not available at the time it is selected as the source operand (F-reg) of a subsequent FPU instruction. The affected instruction will be stalled in the RD-stage until such time that the hazard is resolved.

In order to mitigate the hazards, the coprocessor includes data forwarding paths between the FPU execution result data output and the coprocessor RD-stage (as shown in Figure 3-8). This path will forward the write data value should the write and read instructions target a common register. Forwarding as soon as the result data is available (i.e., prior to the FPU register write) will help mitigate the impact of the hazard.

External RAW Hazards

An external RAW (Read-After-Write) data dependency hazard will occur should the contents of an FPU register be unavailable at the time it is read by the CPU because the register is bound to a previously issued FPU instruction. The coprocessor will detect the hazard and read will be stalled until such time that the register becomes available (i.e., after the result has be written), creating a read stall for the CPU.

In addition, an external RAW hazard will occur if:

- A CPU write to a coprocessor register is immediately followed by the CPU issuing a coprocessor instruction that uses the same register as an operand source.

  or

- A CPU write to a coprocessor register is immediately followed by a CPU read of the same register.

In both of these CPU RAW hazard scenarios, the CPU is responsible for detecting the hazard and inserting the necessary stall cycle for the coprocessor to resolve the hazard. Hazard detection is the same for both scenarios.

In order to resolve these hazards, the coprocessor includes data forwarding paths between the CPU W-stage and both the coprocessor RD-stage (as shown in Figure 3-5) and the CPU read data output (as shown in Figure 3-6). These paths will forward the write data value should the write and read instructions target a common register.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Note: When the CPU attempts to write to an F-reg, it is possible that an instruction in the RD-stage is using the same register as an operand source, and it is stalled as the result of an internal RAW hazard. Forwarding the new CPU write data into this register would then be incorrect because the RD-stage instruction was issued prior to the CPU write instruction. Consequently, should the instruction in the RD-stage have been there for more than one cycle (i.e., be stalled), the FPU will disable the forward path and allow the stall mechanism to recognize the hazard as usual. This will prevent the CPU write from completing until such time that the instruction in the RD-stage has been dispatched to start execution.

CPU write data forwarding to the coprocessor RD-stage allows the CPU to issue a coprocessor instruction earlier than would be possible if the CPU coprocessor write had to complete. CPU write data forwarding to the CPU read data path together with a CPU stall cycle (detected and inserted by the CPU) resolves the (unlikely) hazard that arises when a CPU write is followed immediately by a CPU read of the same coprocessor register. The converse scenario where a CPU read of an FPU register into a W-reg is immediately followed by a CPU write of the same W-reg to another F-reg is shown in Figure 3-7.

FPU WAR Hazards

WAR (Write-After-Read) anti-dependency hazard can occur should the pipeline allow read and write execution to be out of (instruction sequence) order. That is, a WAR hazard will arise whenever an instruction writes to a register before the same register is read by a *prior* instruction. That is, the read and write occur out of execution order resulting in the (older) read instruction ultimately using the (later) write data which would be incorrect.

Under normal sequential execution conditions, a WAR hazard should never arise because the read of all older instructions always precedes the writes of later ones. However, a WAR hazard can arise within the coprocessor pipeline should a slow instruction (e.g., `FPU FSIN`) have a result data dependency (RAW hazard) with a later instruction, and that later instruction is followed by a `MOVWCR` or `POPCR` instruction that targets the same register as the dependency. This is because the dependency will force an FPU pipeline stall until the result data is available and the RAW hazard is resolved, but the `MOVWCR` or `POPCR` move instructions (which do not execute using the FPU pipeline) will not be stalled. Consequently, it is possible that the write from the `MOVWCR` or `POPCR` instruction would occur prior to the stalled instruction continuing execution (after the RAW hazard). The write would then be overwritten by the FPU pipeline and therefore lost. This scenario is detected as a WAR hazard and prevented from happening by stalling the most recent write, such that the write order remains correct. CPU to FPU move instructions that do not target the register involved in the stall will still execute as normal (i.e., without stalling).

The coprocessor must therefore detect the possibility of such a hazard and force in-order execution of all dependent instructions by stalling the most recent CPU write instruction in the W-stage until after the prior read is completed.

An example WAR hazard and its resolution is shown in Figure 3-10. A RAW hazard between the `FSIN.s` and `FMOV.s` instructions will stall `FMOV.s` to resolve the hazard (stall cycles shown in green), but this will also set the pipeline up for a possible WAR hazard because the subsequent `MOVWCR` instruction is not prevented from continuing execution.

As is the case in this example, should the `MOVWCR` instruction destination be the same F-reg as that used by the `FMOV.s` as a source, the `MOVWCR` must be prevented from writing until the prior (`FSIN.s`) has been able to forward the write data to the `MOV.s` RD-stage. The `FSIN.s` and `MOVWCR` enter their respective write stages together, and the FPU prioritizes the CPU write, maintaining correct write ordering. This results in a one cycle stall of `MOVWCR` instruction to resolve the hazard.

FPU WAW Hazards

The WAW (Write-After-Write) hazard is a further consequence of allowing instructions to continue execution while others are stalled or taking longer to execute. As is the case for WAR hazards, instruction writes can end up out of order, leaving an incorrect (stale) value in a destination register.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

WAW output dependency is possible because once the coprocessor instruction is issued, the CPU and coprocessor pipelines operate independently. A multicycle coprocessor instruction may therefore complete after one or more CPU instructions that were subsequently issued (i.e., out of order). A WAW hazard will exist when the CPU instruction is ready to retire before the coprocessor instruction retires and either:

- The same register is a destination for both the coprocessor instruction and the CPU instruction that follows it.

  or

- The CPU instruction write targets a coprocessor register that is being used by the prior coprocessor instruction.

In both cases, the resource cannot be shared.

An internal WAW hazard can arise between successive FPU instructions that have differing execution time. However, each issued instruction is tracked by pushing its associated functional unit ID into a FIFO, which is emptied in the same order as it is filled when instructions move results from their functional units into the WB-stage. Should an expected (from the FIFO) functional unit result not be ready, this knowledge is used in the write stage to complete the destination write in the correct sequence, stalling those instructions that arrive out of order, thereby eliminating the WAW hazard.

If the CPU and FPU pipelines are viewed as conjoined, a WAW hazard is also possible should the CPU attempt to write a value to the same register as that also targeted by a previously issued FPU instruction whose write has not yet completed. However, access to the write target(s) of an instruction is inhibited as soon as the instruction is committed (see 3.6.8.5. FPU and CPU Exceptions). Consequently, any attempt by the CPU to write to an FPU register that is already bound to a prior FPU instruction being executed will result in the write grant failure (and the CPU write stalling).

An example WAW hazard and its resolution is shown in Figure 3-11 for an FPU instruction that requires three iterations of the execute stage to complete. This results in a two cycle write stall within the CPU instruction pipeline.

Note: For the purpose of WAW hazard detection, the FSR is considered as a single entity.

Note: The FSR is bound to all FPU instructions except for `FMOVC` and `FMOV` (these ops do not update the FSR), and FAND and FIOR unless they will modify the FSR. The FEAR is bound to all FPU instructions except for `FMOVC`, `FMOV` and `FTST`. It is also not bound to FAND or FIOR unless it will be modified by them. Note that this applies irrespective of whether FEAR is enabled or not (i.e., FEAR.EACE is a “don’t care” with respect to FEAR hazard detection).

3.6.8.7.3 Instruction/Hazard Tracker

The Instruction/Hazard Tracker is a mechanism whereby hazard related information required while an instruction is progressing through the execute stages is captured in a FIFO for each issued instruction when that instruction is committed and enters the FPU pipeline X [0]-stage. The FIFO depth (Default is four) defines how many instructions may be sequentially dispatched into the execution stage before it is regarded as full.

Each FIFO entry includes the following information which is used during the X-stages:

1. Entry valid flag.
2. Flags to indicate which Functional Block (function and operation precision) is targeted.
3. Operand source register identification and valid flag such that RAW hazards may be identified as the instruction progresses.
4. Flags to support Single Precision and Double Precision NaN propagation logic.
5. Flag to indicate if instruction is FDIV or FSUB (where the operand order is reversed).
6. Flag to indicate if instruction is FMAC (special case for NaN propagation).

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Each FIFO entry requires a 'valid' bit which is clear whenever the entry is empty or after it has been used in the WB-stage. This bit will inhibit any associated hazard detection after an instruction has retired.

Operation precision partially identifies the selected Functional Block but also directs the hazard logic. Single Precision operations need only check for hazards involving single F-regs whereas Double Precision must check F-reg pairs for hazards.

Operand register identification and valid flags log which F-regs are used for operands (not all instructions require all three source operands) for hazard tracking. In addition, each FIFO entry includes the following information (also detected in the RD-stage) which is used during the instruction WB-stage:

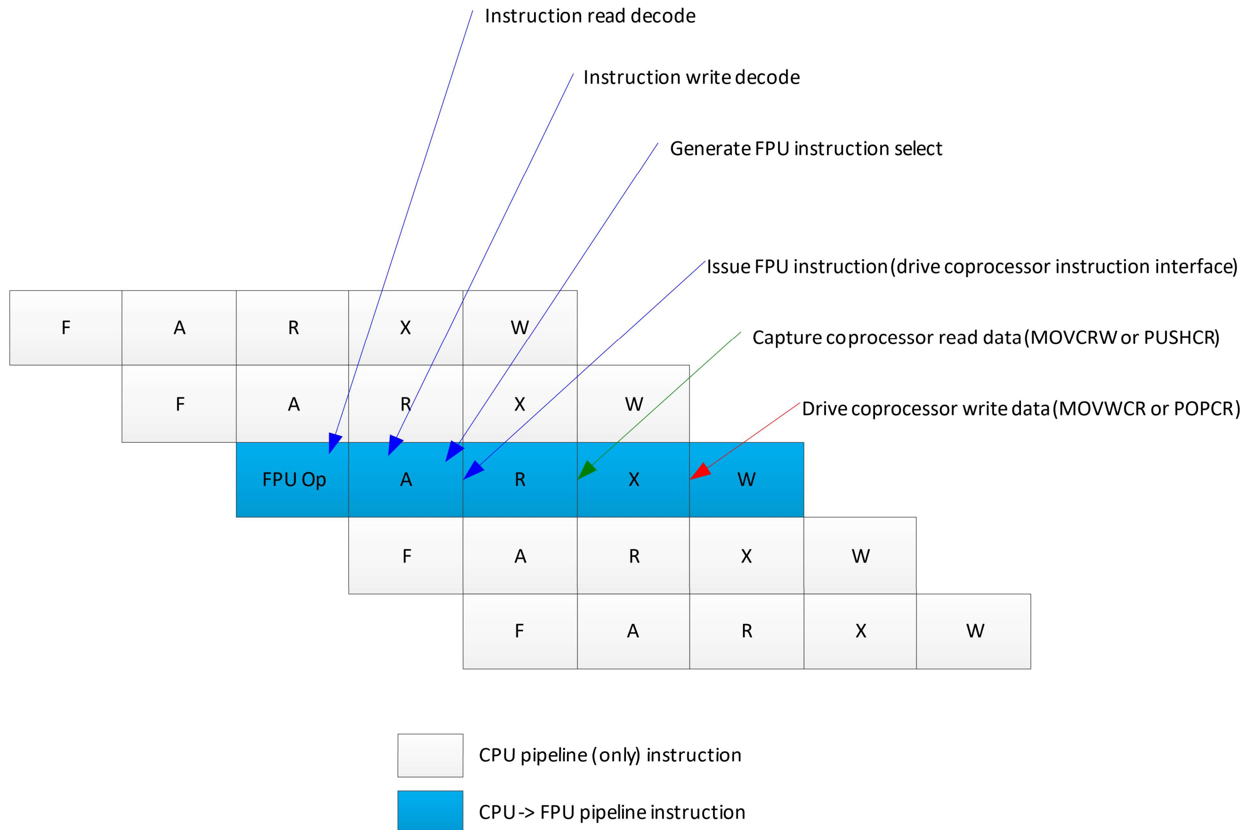
1. Flags to indicate if any result is to be written to an F-reg and whether the FSR is to be updated.
2. Result destination register (and context) identification (defined as DP targets). Additional flags select the active registers (i.e., DP F-reg pair or one of two SP F-reg destinations).
3. Flag to indicate if instruction permits FTZ override of result.
4. CPU A-stage instruction address to capture in the FEAR (if enabled) should the instruction generate an exception.
5. Flags to indicate if the instruction is a FAND or IOR, and the associated FSR/FCR/FEAR target register select bits.
6. The presence of a subnormal operand (when SAZ mode is disabled) is captured and used to signal the subnormal exception (i.e., at the same time as any other exceptions the instruction may generate).

3.6.8.7.4 CPU Write Stalls

Whenever the CPU encounters a write stall, the entire integer pipeline is stalled (because the CPU only supports in-order execution). No subsequent instruction is permitted to move into the W-stage to retire until the write stall is resolved. Different Pipeline stages are explained in 3.6.8.6.1. FPU Pipeline Operation.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

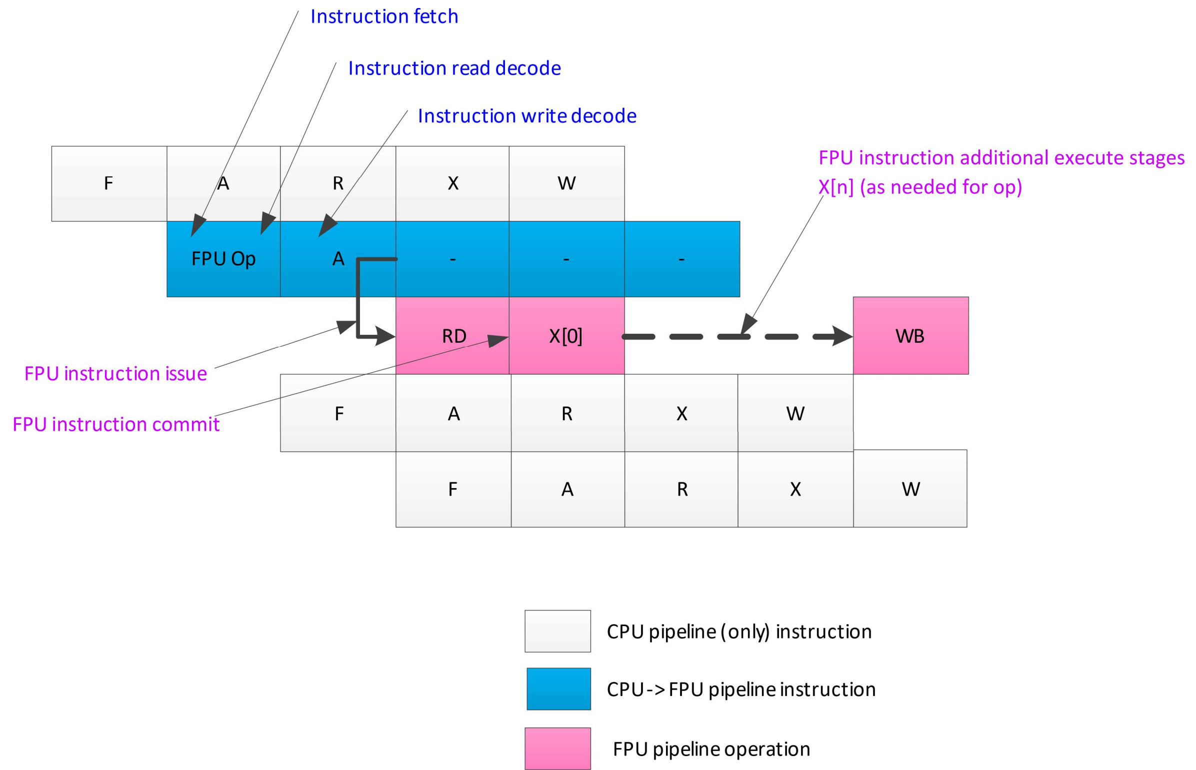
Figure 3-22. CPU Pipeline Coprocessor Interface Flow



Legend: F = Fetch, A = Address decode, R = Read, X = Execute, W = Write back

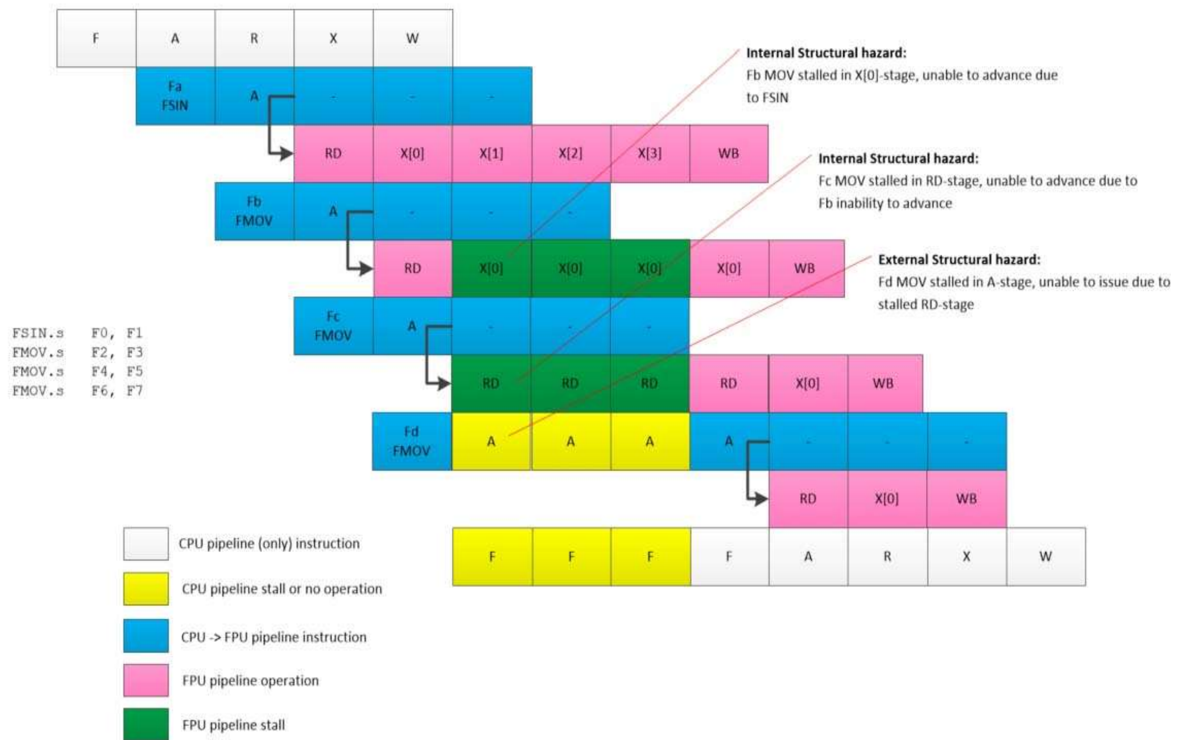
© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-23. CPU Pipeline Coprocessor Issue Flow



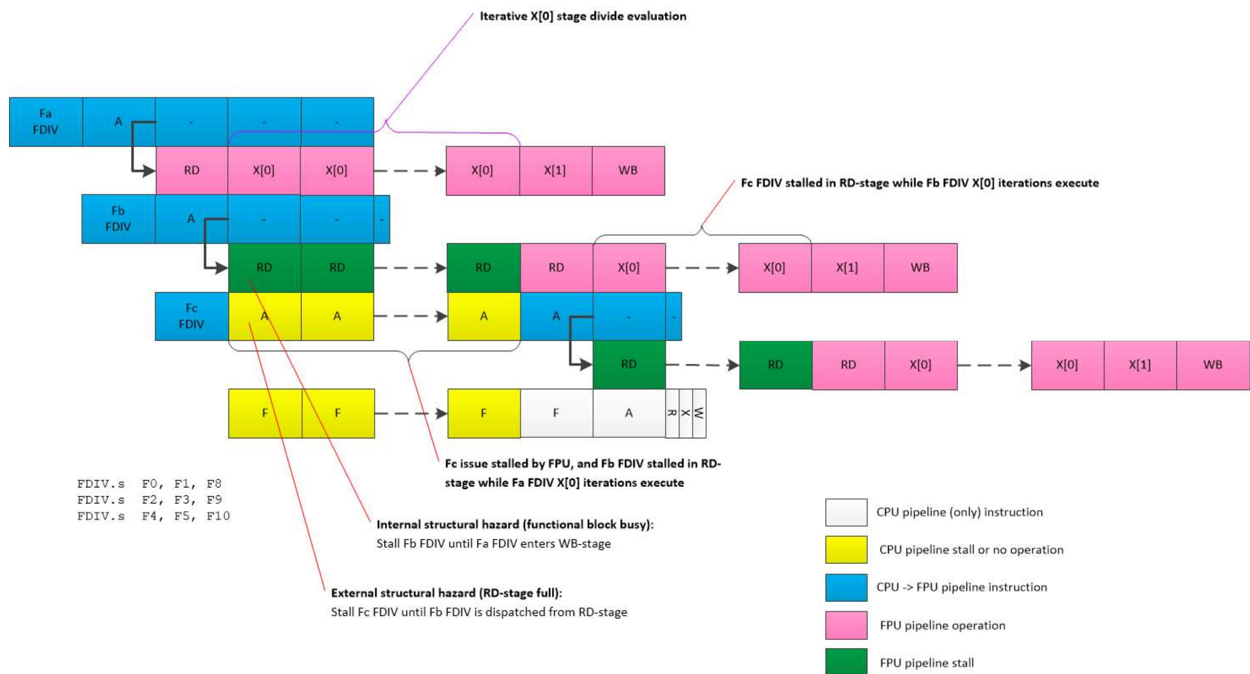
© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-24. Pipeline and Functional Block Busy Internal/External Structural Hazards



Legend: F = Fetch, A = Address decode, R = Read, X = Execute, W = Write back

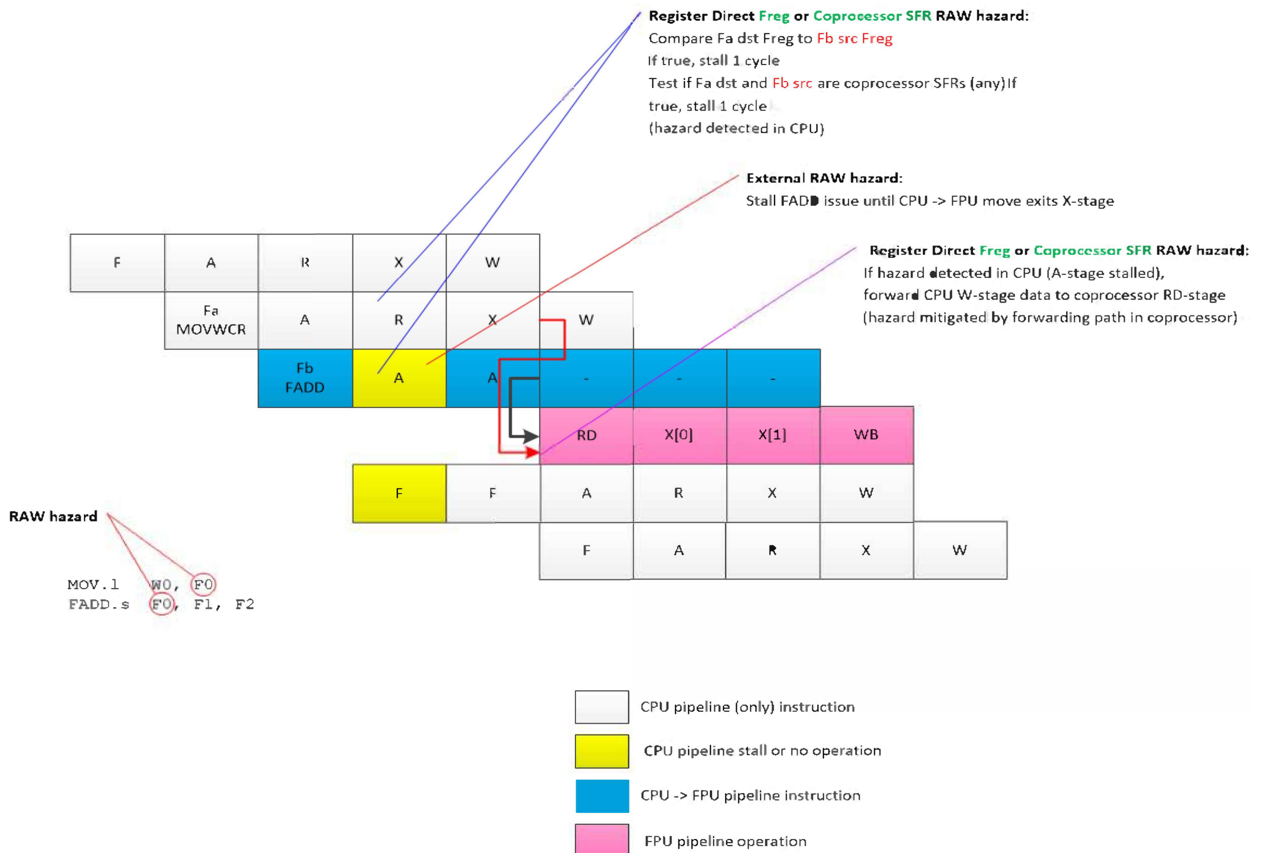
Figure 3-25. FDIV Pipeline and Functional Block Busy Internal/External Structural Hazards



Legend: F = Fetch, A = Address decode, R = Read, X = Execute, W = Write back

© 2023-2024 Microchip Technology Inc. and its subsidiaries

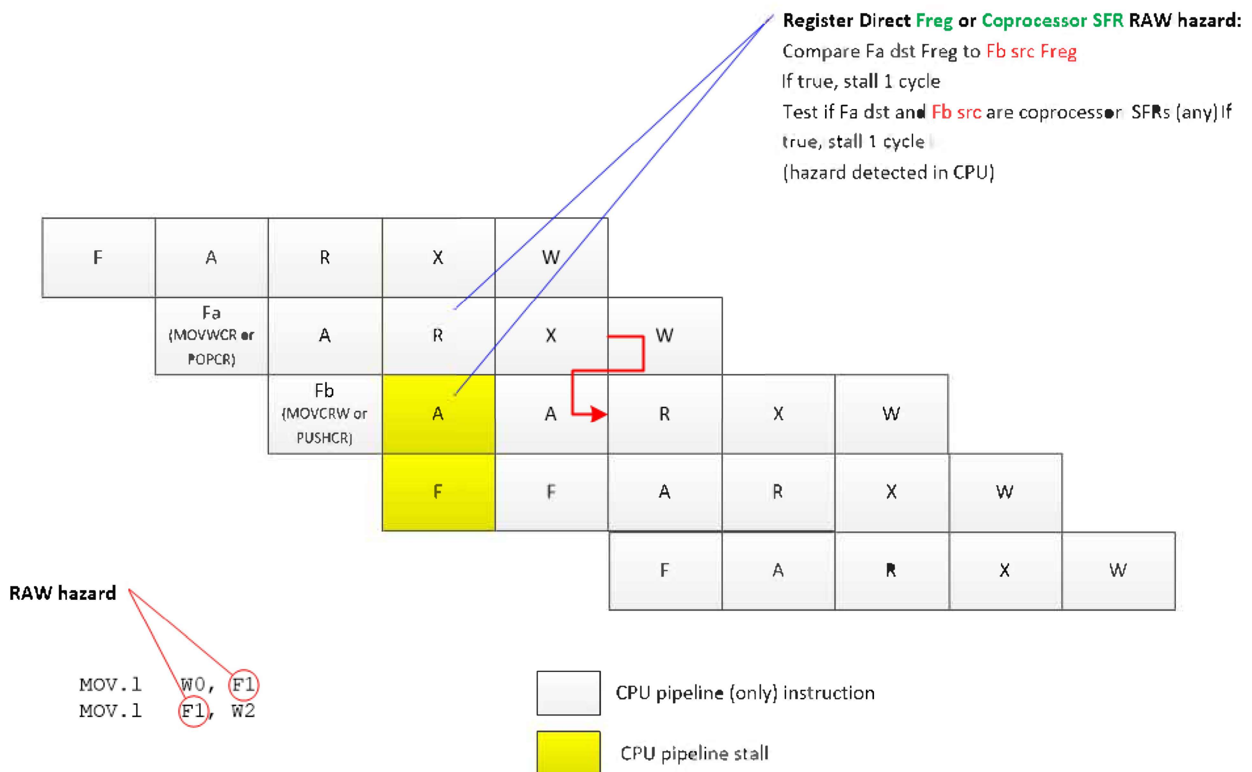
Figure 3-26. External RAW Hazard (CPU Write Data to FPU Read Forwarding)



Legend: F = Fetch, A = Address decode, R = Read, X = Execute, W = Write back

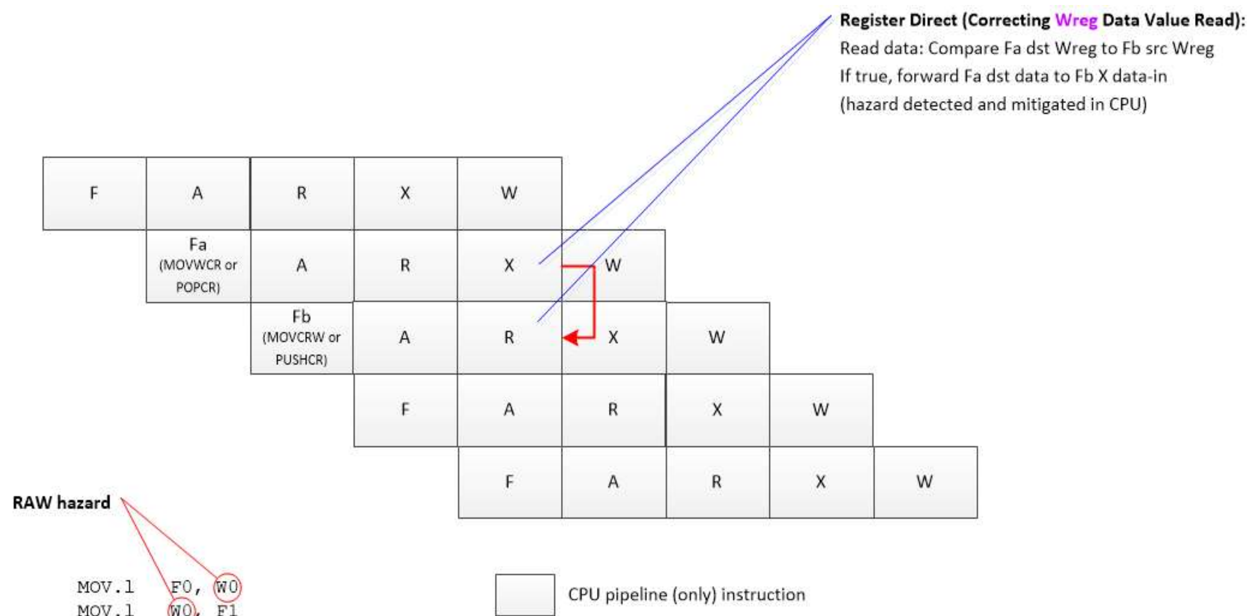
© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-27. External Raw Hazard (CPU F-REG Write Data to CPU F-REG Read Forwarding)



Legend: F = Fetch, A = Address decode, R = Read, X = Execute, W = Write back

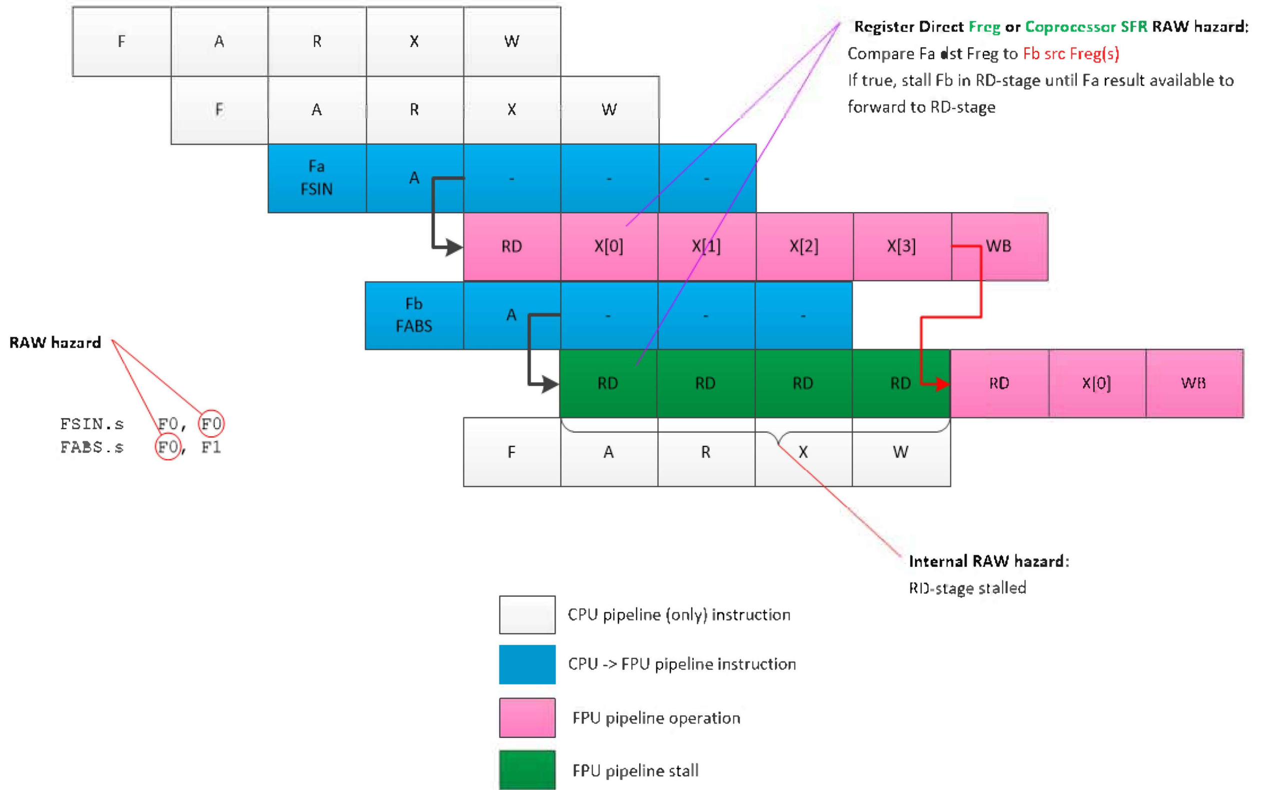
Figure 3-28. External Raw Hazard (CPU W-REG Write Data to CPU W-REG Read Forwarding)



Legend: F = Fetch, A = Address decode, R = Read, X = Execute, W = Write back

© 2023-2024 Microchip Technology Inc. and its subsidiaries

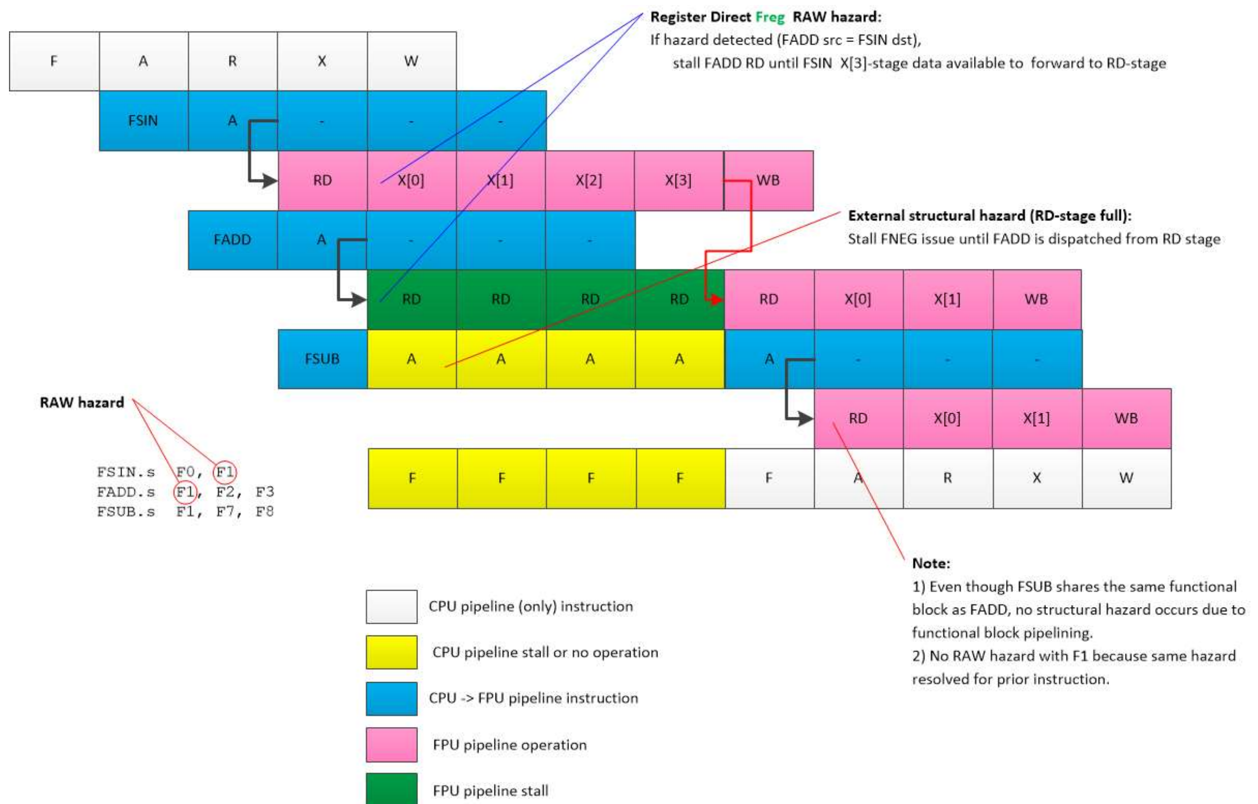
Figure 3-29. Internal Raw Hazard (FPU Write Data to FPU Read Forwarding)



Legend: F = Fetch, A = Address decode, R = Read, X = Execute, W = Write back

© 2023-2024 Microchip Technology Inc. and its subsidiaries

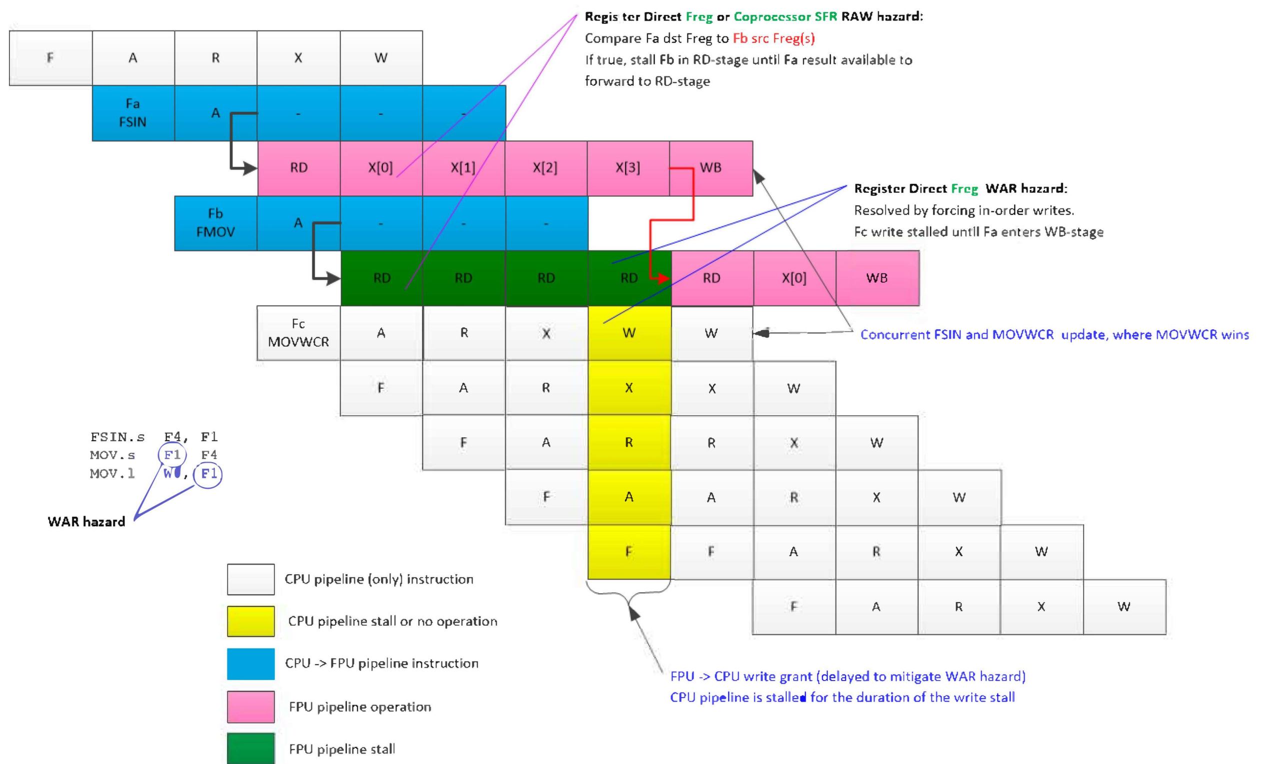
Figure 3-30. Internal Raw Hazard, External and Internal Structural Hazards



Legend: F = Fetch, A = Address decode, R = Read, X = Execute, W = Write back

© 2023-2024 Microchip Technology Inc. and its subsidiaries

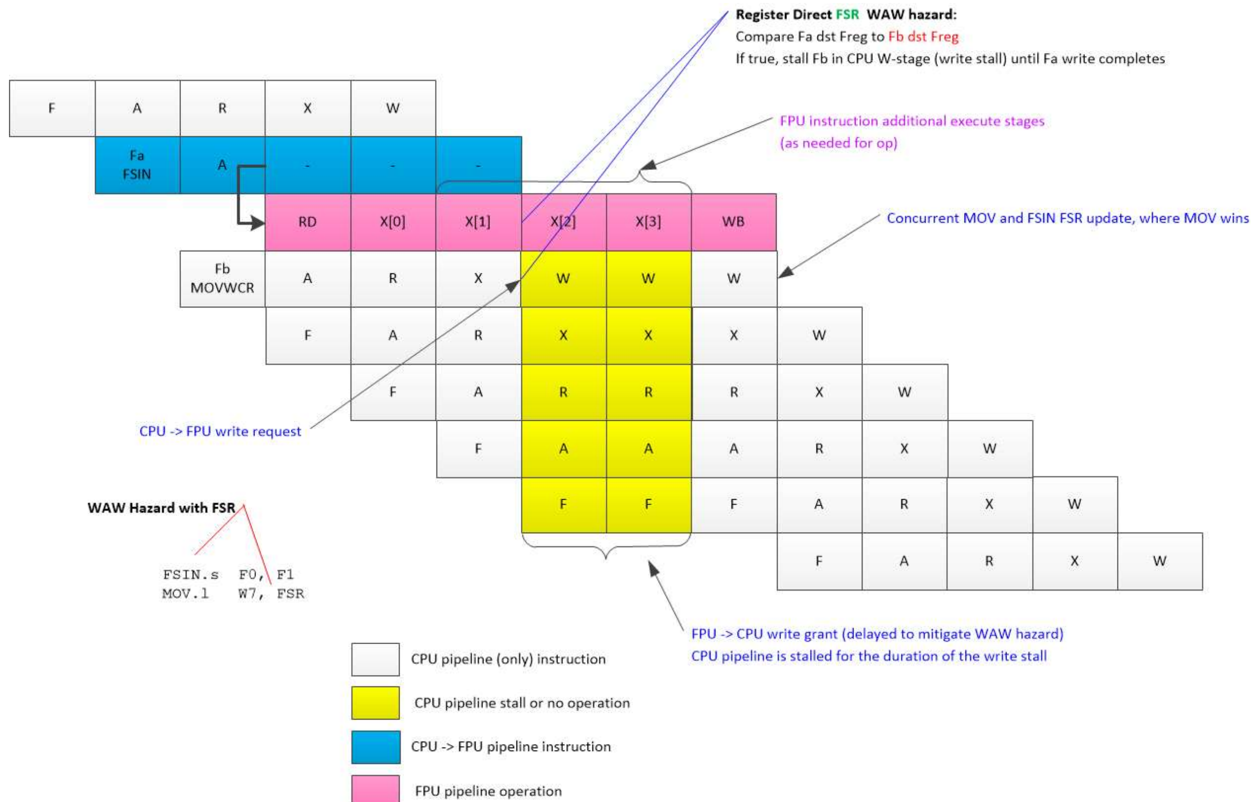
Figure 3-31. FPU WAR Hazard



Legend: F = Fetch, A = Address decode, R = Read, X = Execute, W = Write back

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Figure 3-32. CPU/FPU WAW Hazard



Legend: F = Fetch, A = Address decode, R = Read, X = Execute, W = Write back

3.6.8.8 Operand Pre-Processing

Floating-point operands are subject to examination during the RD-stage in order to implement NaN propagation and the subnormal value override function. This is necessary to apply rules that determine the outcome in the presence of one or more NaN input values and evaluate operands for special conditions.

3.6.8.8.1 NaN Propagation Operand Detection

For instructions that generate a result, special propagation rules apply when one or both source operands are NaN values, such that sNaNs can be successfully used as “tracer” values. Should a NaN be deemed as propagated, then it will replace the operation result.

With reference to NaN Propagation, all instructions will examine the operands for NaN values during the RD-stage:

- Two operand instructions:
  If one or both operands are NaN values, the RD-stage will apply a propagation priority as shown in Table 3-13.
- Three operand FMAC instruction:
  The source operands are examined in the RD-stage as usual but in conjunction with the selected intermediate result, and any NaN values detected are propagated as shown in Table 3-14.
- Three operand FFLIM instruction:
  If one or both limit operands are NaN values, the RD-stage will apply a propagation priority as shown in Table 3-12. If the FFLIM input value is a NaN, the limit values are ignored and the input NaN value is propagated (quieted if an sNaN).

© 2023-2024 Microchip Technology Inc. and its subsidiaries

In all cases, if a NaN is to be propagated, the corresponding NaN value is entered as the operand value in the Instruction/Hazard Tracker FIFO entry for that instruction. The instruction FIFO entry also sets a flag to indicate that NaN propagation is enabled.

3.6.8.8.2 Subnormals Operands

The FPU supports a subnormal operand override mode, Subnormals-Are-Zero (SAZ), the functionality of which is defined in 3.6.3.2.3. Subnormal Operand Exception. Subnormals- Are-Zero (SAZ) mode is enabled when FCR.SAZ is set.

Should a subnormal operand be detected when SAZ mode is disabled, the subnormal exception will be signaled by setting FSR.SUBO (and FSR.SUBOS if not already set) during the WB-stage (i.e., at the same time as when all other exceptions are signaled). If SAZ mode is enabled, the subnormal exception will not be signaled.

Note: SAZ mode is not applicable to `FAND`, `FIOR`, `FMOV`, `FMOVC` or any CPU to/from FPU data move instruction, none of which can modify any FPU status. In addition, SAZ mode is ignored by `FTST` such that a subnormal operand will always be recognized as such by the instruction, irrespective of the state of FCR.SAZ. However, SAZ mode can influence FF2LI/FF2DI operands. In these cases, subnormal or zero operands will write the same result (integer value of 0). But if the operand is subnormal and SAZ mode disabled, a subnormal exception will also be signaled. Conversely, if the operand is subnormal and SAZ mode enabled, a subnormal exception will not be signaled.

3.6.8.9 Result Post-Processing

Floating-point results are subject to examination during the WB-stage to implement the subnormal result override Flush-To-Zero (FTZ) mode and NaN propagation results.

3.6.8.9.1 Subnormal Results

The FPU supports a subnormal result override mode, Flush-To-Zero (FTZ), the functionality of which is defined in 3.6.3.2.2. Flush-To-Zero (FTZ). Flush-To-Zero (FTZ) is enabled when both FCR.FTZ and FCR.UDFM are set. Should the underflow exception be unmasked (FCR.UDFM = 0), then the FCR.FTZ bit will have no effect.

This mode is implemented within the WB-stage such that results written to the destination register (and those forwarded) will be adjusted accordingly if FTZ mode is enabled. The FCR.FTZ bit is only examined during the WB-stage of an instruction such that it may be modified as late as the cycle before the instruction enters the WB-stage.

Note: FTZ mode is not applicable to `FAND`, `FIOR`, `FMOV`, `FMOVC` or any CPU to/from FPU data move instruction, none of which can modify any FPU status. It is also not applicable to FTST because the FSR is the only possible destination for this operation. In addition, FTZ mode will have no effect on FF2LI/FF2DI and FLI2F/FDI2F instruction results because - FF2LI/FF2DI results are integers and - FLI2F/FDI2F destination data may be 0 but never subnormal.

3.6.8.9.2 NaN Propagation Result Write

NaN operand values are detected in the RD-stage, prioritized, and then passed (via an Instruction/ Hazard Tracker FIFO entry) to the instruction WB-stage. A valid NaN propagation will cause the operation result from the Execute stage to be ignored, and the propagated NaN value to be written into the result destination instead, as discussed in NaN Propagation.

3.6.8.9.3 Rounding Modes

The rounding mode for each instruction Functional Block is defined by the value written into FCR.RND [1:0] as defined in 3.6.5. FCR. The FPU treats the rounding mode input as an operand supplied from the RD-stage when the instruction is dispatched into the Execute stage.

Note: Rounding Modes are not applicable to `FAND`, `FIOR`, `FCPQ`, `FCPS`, `FTST`, `FABS`, `FNEG`, `FFLIM`, `FMAX`, `FMIN`, `FMAXNUM`, `FMINNUM`, `FMOV`, `FMOVC` or any CPU to/from FPU data move instruction.

There is a 3-bit rounding mode input (rnd [2:0]) to support up to eight different rounding modes for all FPU conversion operations. Setting rnd [2] = 1 and mapping rnd [1:0] to FCR [9:8] will allow user selection of the IEEE 754 compliant modes as defined in 3.6.5. FCR.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

The integer/floating-point conversion instructions (`FDI2F`, `FLI2F`, `FF2DI`, `FF2LI`) may either specify the rounding mode within the instruction syntax or default to that defined in FCR.RND [1:0]. CPU will issue one of these instructions and the FPU will use it to determine the Functional Block Rounding mode as shown in Table 3-16.

Table 3-16. FPU Conversion OP Rounding Modes Control

Rounding Mode Bits in Opcode[2:0]	Functional Block Rounding Mode
`111`	IEEE Round to Negative Infinity (floor)
`110`	IEEE Round to Positive Infinity (ceiling)
`101`	IEEE Round to Zero (truncate)
`100`	IEEE Round to Nearest (even)
`0xx`	Global mode (defined by FCR.RND[1:0])

3.6.8.10 Floating Point Status

The FPU generates four types of status:

- Exception condition “most-recent” status from most instructions (see Table 3-19). These bits are located within FSR [6:0]:INX, HUGI, OVF, UDF, DIV0, INVAL, SUBO.
- Exception condition "sticky" status from most instructions (see Table 3-19). These bits are located within FSR [14:8]: INXS, HUGIS, OVFS, UDFS, DIV0S, INVALS, SUBOS.
- Value ordering relations status to indicate the result of the FCPS/FCPQ compare instructions. These bits are located within FSR [19:16]:GT, LT, EQ, UN.
- Operand characteristic status from the FTST datum inspection/classify instruction. These bits are located within FSR [28:24]: SUB, INF, FZ, FN, FNAN.

Operand comparisons are likely to be used frequently, so the compare status bits generated by the FCPS/FCPQ instructions are supported with CPU conditional branch instructions. All other status must be read into the CPU (using the MOVCRW instruction) or pushed onto the stack (using PUSHCR) and then acted upon as necessary.

Note: Irrespective of whether an exception is masked or not, writing a logic 1 to an exception status flag using any instruction that can write 1’s to the FSR will not result in any associated exception being taken.

3.6.8.10.1 Compare Status and Predicates

IEEE 754-2008/2019 standards specify Quiet and Signaling Compare Predicates (equations) as shown in Table 3-17. A “signaling” predicate signals (i.e., attempts to generate an exception) when a Quiet NaN or Signaling NaN (qNaN or sNaN) operand is detected.

A “quiet” predicate will not signal when a qNaN operand is detected.

An sNaN will always signal an exception when detected as an operand for all instructions except those that do not generate any exceptions (`FMOV`, `FMOVC`, `FABS`, `FNEG` and `FTST`).

The FPU coprocessor macro implements Signaling and Quiet predicates by supporting two floating-point compare options, one signaling (FCPS), one quiet FCPQ), and a set of floating-point branch operations that test for the required predicates. Each compare instruction will set one of the four mutually exclusive ordering relations (GT, LT, EQ, UN status bits) located in the FSR to indicate the result of the comparison.

- FCPS (signaling compare):
  - qNaN or sNaN: If either or both operands are a qNaN or sNaN value, the compare is considered unordered which will cause the FSR.UN bit to be set. In addition, the FSR.INVAL bit will be set, causing the CPU to be signaled via the Invalid exception (assuming that the exception is not masked).
- FCPQ (quiet compare):

© 2023-2024 Microchip Technology Inc. and its subsidiaries

- qNaN: If one or more operands contain a qNaN value, the compare is considered unordered which will cause the FSR.UN bit to be set. A qNaN will not set the FSR.INVAL bit, so no signaling will occur.
- sNaN: If either or both operands are a sNaN value, the compare is considered unordered which will cause the FSR.UN bit to be set. In addition, the FSR.INVAL bit will be set, causing the CPU to be signaled via the Invalid exception (assuming that the exception is not masked).

The compare operation subtracts Fs (subtrahend) from Fb (minuend). The EQ, GT and LT status bits are set as follows:

- If the minuend is equal to the subtrahend (Fb = Fs) the EQ status bit is set.
- If the minuend is greater than the subtrahend (Fb > Fs) the GT status bit is set.
- If the minuend is less than the subtrahend (Fb < Fs) the LT status bit is set.

In addition, the UN status bit is set if one or both operands is a NaN. If this is the case, no other compare status is set (i.e., UN and EQ, GT, LT are mutually exclusive).

Note: The FCPS/FCPQ instructions consider -0 and +0 as equivalent.

Note: Comparing a value to itself should produce an equivalence result. However, UN has precedence over EQ such that, should two values be identical but both NaN, the UN bit will be set but the EQ bit will be cleared.

FPU Status Conditional Branches

The CPU has the ability to conditionally branch off various status bits generated within the coprocessor. In the case of the FPU, an internal status register (FSR) is supported which is updated at the end of each floating-point operation.

The FPU FSR is comprised of instruction exception status and FCPS/FCPQ/FTST instruction status. Conditional branching is supported within the CPU for the FCPS/FCPQ compare instructions only.

The CPU ISA includes a set of generic coprocessor conditional branch instructions, CBRA0 through CBRA15, each of which can operate with any instantiated coprocessor and branch based upon the state of a corresponding bit within a vector supplied by each coprocessor. In the case of the FPU, CBRA0 through CBRA13 are used, each represented as an FBRA instruction with its corresponding assembler attribute, for the FCPS/FCPQ instruction status branch conditions. The FCPS/FCPQ status is held in FSR [19:16] and indicates the comparison result. CBRA[n] timing is the same as any other CPU conditional branch, such that the condition is examined at the end of the CBRA[n] R-stage. If the condition is true, the branch is taken. If the condition is not true, the branch is not taken and sequential execution continues.

As is the case for all conditional branches, the instruction(s) immediately following the branch are speculatively executed, and they will either be part of the taken or the not taken path, based on the direction of the branch. These instructions are permitted to be floating-point operations. This requires that the FPU accommodate the possibility that these instructions could ultimately be killed due to a branch mispredict.

Note that the FPU will not return the result of FBRA instruction until any FCPS/FCPQ instruction already underway in the coprocessor pipeline has retired. The CPU will consequently stall until such time that the msw of the FSR is available to be read (though these are fast operations, so stalls should be minimal). In effect, a CPU conditional branch instruction operation will synchronize the integer and floating-point pipelines with respect to FPU FCPS/FCPQ status.

The LS 3-bits of the branch opcode concatenated with the sub-opcode bit (such that the sub-opcode bit becomes the LSb of this value), may be used by the CPU decoder as a bit pointer into the 16-bit branch status test value to select the corresponding branch predicate result. The branch then decides if the outcome is true (taken) or false (not taken) based on the state of the selected bit (where true is when the bit is set, false when clear).

© 2023-2024 Microchip Technology Inc. and its subsidiaries

Note: FCPS/FCPQ and FTST instructions update two different portions of the FSR. Consequently, execution of an FTST instruction (which also updates the FSR) will not inhibit the CPU CBRAn instructions from using the branch status generated from the FSR ordering relation bits.

The FTST instruction will test the operand and update the SUB, INF, FN, FZ, FNAN status bits. No exceptions will be generated by this instruction. Due to the relative infrequent use of this instruction, dedicated conditional branches are not supported by the CPU to test these status bits. The user must read the FSR and then act upon the bits of instruction using existing CPU instructions.

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6.3.10.2 Operand Characteristic (Test) Status

Table 3-17. Floating-Point Conditional Branches and Associated Predicates

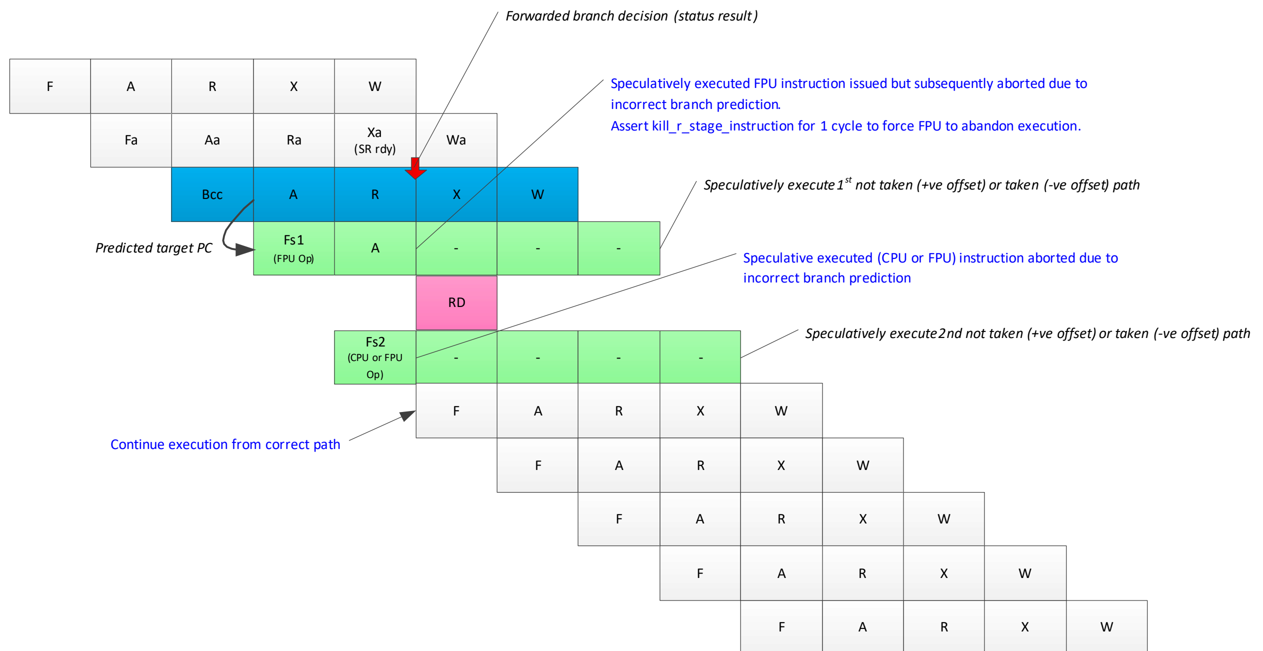
Assembler FBRA Attribute	Design Mnemonic CBRA Mapping	Predicates					Assembler FBRA Attribute	Design Mnemonic CBRA Mapping	Negated Predicates				
		Ordering Relation				Definition(1) (Alternative Definition)			Ordering Relation				Definition(2) (Alternative Definition)
		>	<	=	?				>	<	=	?	
EQ	CBRA0	F	F	T	F	Equal	UNE	CBRA1	T	T	F	T	Unordered or Greater Than or Less Than (Unordered or Not Equal)
NE	CBRA2	T	T	F	F	Greater Than or Less Than (Not Equal)	UEQ(3)	CBRA3	F	F	T	T	Unordered or Equal
GT	CBRA4	T	F	F	F	Greater Than	ULE	CBRA5	F	T	T	T	Unordered or Less Than or Equal
GE	CBRA6	T	F	T	F	Greater Than or Equal	ULT	CBRA7	F	T	F	T	Unordered or Less Than
LT	CBRA8	F	T	F	F	Less Than	UGE	CBRA9	T	F	T	T	Unordered or Greater Than or Equal
LE	CBRA10	F	T	T	F	Less Than or Equal	UGT	CBRA11	T	F	F	T	Unordered or Greater Than
OR	CBRA12	T	T	T	F	Ordered	UN	CBRA13	F	F	F	T	Unordered

© 2023-2024 Microchip Technology Inc. and its subsidiaries

3.6.8.10.3 FPU Instruction Kill

As is the case for all instructions executed within conditional branch speculative slots, floating-point instructions will be killed if a branch mispredict occurs. The CPU will recognize a mispredict prior to the end of the conditional branch R-stage (i.e., when the prior instruction status is available to forward). If the instruction in the first speculative slot is an FPU instruction, it will be issued to the FPU, but the CPU will assert a signal to kill the instruction for one cycle, forcing the FPU to subsequently abandon execution prior to it being committed. If the instruction in the second speculative slot is an FPU instruction, it will be abandoned prior to being issued to the FPU.

Figure 3-33. FPU Instruction Speculative Execution



© 2023-2024 Microchip Technology Inc. and its subsidiaries