SiFive FE310-G000 Manual
v2p3

SiFive FE310-G000 Manual

Release Information

Version	Date	Changes
v2p3	October 11, 2017	Core Complex branding
v2p2	September 28, 2017	Clarify PLIC, PMU, RTC, WDT reset values. Add “empty” bit to UART rxdata register map. General Formatting.
v2p1	September 21, 2017	Correct the location of the config string pointer
v2p0	September 15, 2017	Fold in relevant E31 Core Complex and E300 Platform information
1.0.3	July 24, 2017	Correct DWAKEUP_N and AON_PMU_OUT_0 pin assignments
1.0.2	June 12, 2017	Clarify that QFN48 is the 6x6 Standard format
1.0.1	December 20, 2016	Add QFN48 Package Pinout, add Configuration String, rename chip to FE310-G000
1.0	November 29, 2016	HiFive1 release

Contents

Chapter 1

Introduction

The FE310-G000 is the first Freedom E300 SoC, and forms the basis of the HiFive1 development board for the Freedom E300 family. The FE310-G000 is built around the E31 Core Complex instantiated in the Freedom E300 platform and fabricated in the TSMC CL018G 180nm process. This manual describes the specific configuration for the FE310-G000.

The SiFive FE310-G000 is compatible with all applicable RISC-V standards, and this document should be read together with the official RISC-V user-level, privileged, and external debug architecture specifications.

1.1 FE310-G000 Overview

Figure 1.1 shows the overall block diagram of the FE310-G000 which contains an E31 Core Complex, a selection of flexible I/O peripherals, a dedicated off-chip Quad-SPI flash controller with execute-in-place support, 8 KiB of in-circuit programmable OTP memory, 8 KiB of mask ROM, clock generation, and an always-on (AON) block including a programmable power-management unit (PMU).

A feature summary table can be found in Table 1.1.

1.2 RISC-V Core

The FE310-G000 includes a SiFive E31 RISC-V core, which is a high-performance single-issue in-order execution pipeline, with a peak sustainable execution rate of one instruction per clock cycle. The RISC-V core supports Machine mode only as well as the standard Multiply, Atomic, and Compressed RISC-V extensions (RV32IMAC).

The RISC-V core is described in more detail in Chapter 8.

1.3 On Chip Memory System

The FE310-G000 memory system has Data Tightly Integrated Memory subsystem (DTIM) optimized for high performance. The data subsystem has a DTIM size of 16 KiB. The instruction subsystem consists of a 16 KiB 2-way instruction cache.

The system mask ROM is 8 KiB in size and contains simple boot code. The system ROM also holds the platform configuration string and debug ROM routines.

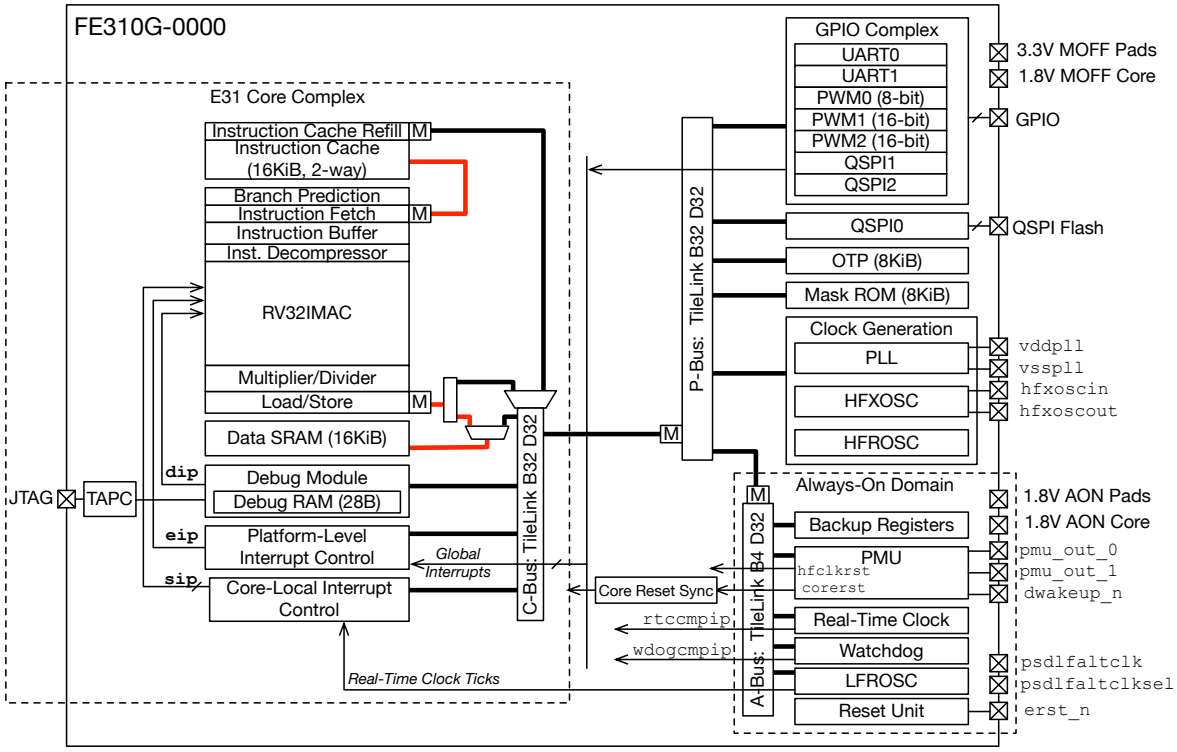

Figure 1.1: FE310-G000 top-level block diagram.

The on-chip memory system is described in more detail in the RISC-V CPU core Chapter 8, the OTP Chapter 16, and in the Memory Map in Chapter 3.

1.4 Interrupts

The FE310-G000 includes a RISC-V standard platform-level interrupt controller (PLIC), which supports 51 global interrupts with 7 priority levels. This Core Complex also provides the standard RISC-V machine-mode timer and software interrupts via the Core Local Interruptor (CLINT).

Interrupts are described in Chapter 9, the PLIC in Chapter 10, and the CLINT in Chapter 11.

1.5 Always-On (AON) Block

The AON block contains the reset logic for the chip, an on-chip low-frequency oscillator, a watchdog timer, connections for an off-chip low-frequency oscillator, the real-time clock, a programmable power-management unit, and 16×32-bit backup registers that retain state while the rest of the chip is in a low-power mode.

The AON can be instructed to put the system to sleep. The AON can be programmed to exit sleep mode on a real-time clock interrupt or when the external digital wakeup pin, `dwakeup_n`, is pulled low. The `dwakeup_n` input supports wired-OR connections of multiple wakeup sources.

The Always-On block is described in Chapter 12.

FE310-G000 Feature Summary Table		
Feature	**Description**	**Available in QFN48**
RISC-V Core	1x E31 RISC-V Core with machine mode only, 16 KiB 2-way instruction cache, and a 16 KiB data tightly integrated memory (DTIM).	✓
Interrupts	Software and timer interrupts, 51 peripheral interrupts connected to the PLIC with 7 levels of priority.	✓
UART 0	Universal Asynchronous/Synchronous Transmitters for serial communication.	✓
UART 1	Universal Asynchronous/Synchronous Transmitters for serial communication.	
QSPI 0	Serial Peripheral Interface. QSPI 0 has Execute in Place mode enabled by default and 1 chip select signal.	✓
QSPI 1	Serial Peripheral Interface. QSPI 1 has 4 chip select signals.	✓ (3 CS lines) (2 DQ lines)
QSPI 2	Serial Peripheral Interface. QSPI 2 has 1 chip select signal.	
PWM 0	8-bit Pulse-width modulator with 4 comparators	✓
PWM 1	16-bit PWM with 4 comparators	✓
PWM 2	16-bit PWM with 4 comparators	✓
GPIO	32 GPIO pins with SPI, UART, PWM pin aliases	✓ (19 pins)
Always On Domain	Supports low-power operation and wakeup	✓

Table 1.1: FE310-G000 Feature Summary Table. Note that the FE310-G000 contains features in silicon which are available to software, but are not connected to pads on the package.

1.6 GPIO Complex

The GPIO complex manages the connection of digital I/O pads to digital peripherals, including SPI, UART, and PWM controllers, as well as for regular programmed I/O operations. FE310-G000 has two additional QSPI controllers in the GPIO block, one with four chip selects and one with one. FE310-G000 also has two UARTs. FE310-G000 has three PWM controllers, two with 16-bit precision and one with 8-bit precision.

In the QFN48-pin package, not all GPIO pins are exported to package pads, thus not all peripherals and pads are available, though their control and status registers are accessible by software. Refer to Table 1.1 for more information.

The GPIO complex is described in more detail in Chapter 17.

1.7 Quad-SPI Flash

A dedicated quad-SPI (QSPI) flash interface is provided to hold code and data for the system. The QSPI interface supports burst reads of 32 bytes over TileLink to accelerate instruction cache refills. The QSPI can be programmed to support eXecute-In-Place (XIP) modes to reduce SPI command overhead on instruction cache refills. The QSPI interface also supports single-word data reads over the primary TileLink interface, as well as programming operations using memory-mapped control registers.

The QSPI flash interface is described in more detail in Chapter 18.

1.8 Hardware Serial Peripheral Interface

Hardware serial peripheral interfaces (SPI) are available and provide a means for serial communication between the FE310-G000 and off-chip devices.

Chapter 18 describes the hardware SPI controllers in more detail.

1.9 Universal Asynchronous Receiver/Transmitter

Multiple universal asynchronous receiver/transmitter (UARTs) are available and provide a means for serial communication between the FE310-G000 and off-chip devices.

The UART peripherals are described in Chapter 19.

1.10 Pulse Width Modulation

The pulse width modulation (PWM) peripheral can generate multiple types of waveforms on GPIO output pins, and can also be used to generate several forms of internal timer interrupt.

The PWM peripherals are described in Chapter 20.

1.11 Debug Support

The debug module is accessed over JTAG, and has support for two programmable hardware breakpoints. The debug RAM has 28 bytes of storage.

A four-wire 1149.1 JTAG connection is used to connect the external debugger to the internal debug module.

Debug support is described in detail in Chapter 21 and the debug interface is described in Chapter 22.

Chapter 2

Terminology

CLINT	Core Local Interruptor. Generates per-hart software interrupts and timer interrupts.
Hart	HARdware Thread
DTIM	Data Tightly Integrated Memory
ITIM	Instruction Tightly Integrated Memory
JTAG	Joint Test Action Group
LIM	Loosely Integrated Memory. Used to describe memory space delivered in a SiFive Core Complex but not tightly integrated to a CPU core.
PMP	Physical Memory Protection
PLIC	Platform-Level Interrupt Controller. The global interrupt controller in a RISC-V system.
TileLink	A free and open interconnect standard originally developed at UC Berkeley.
RO	Used to describe a Read Only register field.
RW	Used to describe a Read/Write register field.
WO	Used to describe a Write Only registers field.
WARL	Write-Any Read-Legal field. A register field that can be written with any value, but returns only supported values when read.
WIRI	Writes-Ignored, Reads-Ignore field. A read-only register field reserved for future use. Writes to the field are ignored, and reads should ignore the value returned.
WLRL	Write-Legal, Read-Legal field. A register field that should only be written with legal values and that only returns legal value if last written with a legal value.
WPRI	Writes-Preserve Reads-Ignore field. A register field that may contain unknown information. Reads should ignore the value returned, but writes to the whole register should preserve the original value.

Chapter 3

Memory Map

The memory map of the FE310-G000 is shown in Table 3.1.

FE310-G000 Memory Map				
Base	**Top**	**Attr.**	**Description**	**Notes**
0x0000_0000	0x0000_00FF		*Reserved*	Debug Address Space
0x0000_0100	0x0000_0FFF	RWXC	Debug	
0x0000_1000	0x1000_1FFF	RXC	Mask ROM	On-Chip Non-Volatile Memory
0x0000_2000	0x0001_FFFF		*Reserved*	
0x0002_0000	0x0002_1FFF	RXC	OTP 8KiB	
0x0002_2000	0x01FF_FFFF		*Reserved*	
0x0200_0000	0x0200_FFFF	RW	CLINT	On-Chip Peripherals
0x0201_0000	0x0BFF_FFFF		*Reserved*	
0x0C00_0000	0x0FFF_FFFF	RW	PLIC	
0x1000_0000	0x1000_7FFF	RW	Always-On (AON)	
0x1000_8000	0x1000_FFFF	RW	PRCI	
0x1001_0000	0x1001_0FFF	RW	OTP Control	
0x1001_1000	0x1001_1FFF		*Reserved*	
0x1001_2000	0x1001_2FFF	RW	GPIO 0	
0x1001_3000	0x1001_3FFF	RW	UART 0	
0x1001_4000	0x1001_4FFF	RW	QSPI0 Control	
0x1001_5000	0x1001_5FFF	RW	PWM 0	
0x1001_6000	0x1002_2FFF		*Reserved*	
0x1002_3000	0x1002_3FFF	RW	UART 1	
0x1002_4000	0x1002_4FFF	RW	QSPI 1	
0x1002_5000	0x1002_5FFF	RW	PWM 1	
0x1002_6000	0x1003_3FFF		*Reserved*	
0x1003_4000	0x1003_4FFF	RW	QSPI 2	
0x1003_5000	0x1003_5FFF	RW	PWM 2	
0x1003_6000	0x1FFF_FFFF	RW	*Reserved*	
0x2000_0000	0x3FFF_FFFF	RXC	QSPI 0 XIP (512 MiB)	Off-Chip Non-Volatile Memory
0x4000_0000	0x7FFF_FFFF		*Reserved*	
0x8000_0000	0x8000_3FFF	RWXC	Data Tightly Integrated Memory (DTIM) 16 KiB)	On-Chip Volatile Memory
0x8000_4000	0xFFFF_FFFF		*Reserved*	

Table 3.1: FE310-G000 Memory Map.
Memory Attributes: **R** - Read **W** - Write **X** - Execute **C** - Cacheable

Chapter 4

Power Modes

This chapter describes the different power modes available on the FE310-G000. The FE310-G000 supports three power modes: Run, Wait, and Sleep.

4.1 Run Mode

Run mode corresponds to regular execution where the processor is running. Power consumption can be adjusted by varying the clock frequency of the processor and peripheral bus, and by enabling or disabling individual peripheral blocks. The processor exits run mode by executing a "Wait for Interrupt" (WFI) instruction.

4.2 Wait Mode

When the processor executes a WFI instruction it enters Wait mode, which halts instruction execution and gates the clocks driving the processor pipeline. All state is preserved in the system. The processor will resume in Run mode when there is a local interrupt pending or when the PLIC sends an interrupt notification. The processor may also exit wait mode for other events, and software must check system status when exiting wait mode to determine the correct course of action.

4.3 Sleep Mode

Sleep mode is entered by writing to a memory-mapped register `pmusleep` in the power-management unit (PMU). The `pmusleep` register is protected by the `pmukey` register which must be written with a defined value before writing to `pmusleep`.

The PMU will then execute a power-down sequence to turn off power to the processor and main pads. All volatile state in the system is lost except for state held in the AON domain. The main output pads will be left floating.

Sleep mode is exited when an enabled wakeup event occurs, whereupon the PMU will initiate a wakeup sequence. The wakeup sequence turns on the core and pad power supplies while asserting reset on the clocks, core and pads. After the power supplies stabilize, the clock reset is deasserted to allow the clocks to stabilize. Once the clocks are stable, the pad and processor resets are deasserted, and the processor begins running from the reset vector.

Software must reinitialize the core and can interrogate the PMU `pmucause` register to determine the cause of reset, and can recover pre-sleep state from the backup registers. The processor

always initially runs from the HFROSC at the default setting, and must reconfigure clocks to run from an alternate clock source (HFXOSC or PLL) or at a different setting on the HFROSC.

Because the FE310-G000 has no internal power regulator, the PMU's control of the power supplies is through chip outputs, `pmu_out_0` and `pmu_out_1`. The system integrator can use these outputs to enable and disable the power supplies connected to the FE310-G000.

Chapter 5

Clock Generation

The FE310-G000 supports many alternative clock-generation schemes to match application needs. This chapter describes the structure of the clock generation system. The various clock configuration registers live either in the AON block (Chapter 12) or the PRCI block (Chapter 5.2).

5.1 Clock Generation Overview

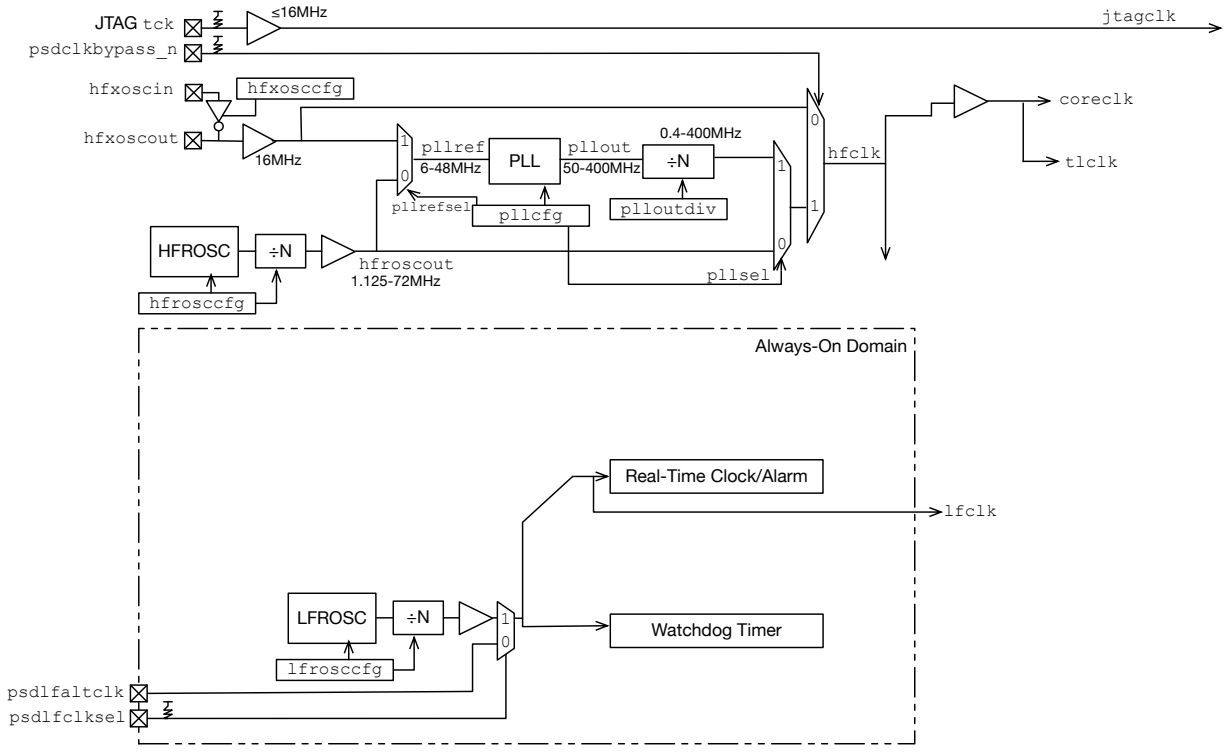

Figure 5.1: FE310-G000 clock generation scheme.

Figure 5.1 shows an overview of the FE310-G000 clock generation scheme. Most digital clocks on the chip are divided down from a central high-frequency clock `hfclk` produced from either the

PLL or an on-chip trimmable oscillator. The PLL can be driven from either the on-chip oscillator or an off-chip crystal oscillator. In systems without a PLL, the off-chip oscillator can drive the high-frequency clock directly.

For the FE310-G000, the TileLink bus clock (`tlclk`) is fixed to be the same as the processor core clock (`coreclk`).

The Always-On block includes a real-time clock circuit that is driven from one of the low-frequency clock sources: an off-chip oscillator (LFOSC) or an an on-chip low-frequency oscillator (LFROSC).

5.2 PRCI Address Space Usage

PRCI (Power, Reset, Clock, Interrupt) is an umbrella term for platform non-AON memory-mapped control and status registers controlling component power states, resets, clock selection, and low-level interrupts, hence the name. The AON block contains registers with similar functions, but only for the AON block units.

Table 5.1 shows the memory map for the PRCI on the FE310-G000.

FE310-G000 PRCI Memory Map				
Offset	**Width**	**Attr.**	**Description**	**Notes**
0x0000	4B	RW	`hfrosccfg`	See Section 5.3
0x0004	4B	RW	`hfxosccfg`	See Section 5.4
0x0008	4B	RW	`pllcfg`	See Section 5.5
0x000c	4B	RW	`plloutdiv`	See Section 5.6

Table 5.1: FE310-G000 PRCI Memory Map.

5.3 Internal Trimmable Programmable 72 MHz Oscillator (HFROSC)

An internal trimmable high-frequency ring oscillator (HFROSC) is used to provide the default clock after reset, and can be used to allow operation without an external high-frequency crystal or the PLL.

The oscillator is controlled by the `hfrosccfg` register, which is memory-mapped in the PRCI address space, and whose format is shown in Figure 5.2.

HF Ring Oscillator Configuration Register (`hfrosccfg`)				
Register Offset		0x000		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	hfroscdiv	RW	0x4	HFROSC divider
[15:6]	*Reserved*	RW	X	Reserved
[20:16]	hfrosctrim	RW	0x10	HFROSC trim value
[29:21]	*Reserved*	RW	X	Reserved
30	hfroscen	RW	0x1	HFROSC Enable
31	hfroscrdy	RO	X	HFROSC Ready

Table 5.2: HF Ring Oscillator Configuration Register

The frequency can be adjusted in software using a 5-bit trim value in the `hfrosctrim`. The trim value (from 0–31) adjusts which tap of the variable delay chain is fed back to the start of the ring. A

value of 0 corresponds to the longest chain and slowest frequency, while higher values correspond to shorter chains and therefore higher frequencies.

The HFROSC oscillator output frequency can be divided by an integer between 1 and 64 giving a frequency range of 1.125 MHz–72 MHz assuming the trim value is set to give a 72 MHz output. The value of the divider is given in the `hfroscdiv` field, where the divide ratio is one greater than the binary value held in the field (i.e., `hfroscdiv`=0 indicates divide by 1, `hfroscdiv`=1 indicates divide by 2, etc.). The value of the divider can be changed at any time.

The HFROSC is the default clock source used for the system core at reset. After a reset, the `hfrosctrim` value is reset to 16, the middle of the adjustable range, and the divider is reset to $\div 5$ (`hfroscdiv`=4), which gives a nominal 13.8 MHz ($\pm$50%) output frequency.

The value of `hfrosctrim` that most closely achieves an 72 MHz clock output at nominal conditions (1.8 V at 25 C) is determined by manufacturing-time calibration and is stored in on-chip OTP storage. Upon reset, software in the processor boot sequence can write the calibrated value into the `hfrosctrim` field, but the value can be altered at any time during operation including when the processor is running from HFROSC.

To save power, the HFROSC can be disabled by clearing `hfroscen`. The processor must be running from a different clock source (the PLL, external crystal, or external clock) before disabling HFROSC. HFROSC can be explicitly renabled by setting `hfroscen`. HFROSC will be automatically re-enabled at every reset.

The status bit `hfroscrdy` indicates if the oscillator is operational and ready for use as a clock source.

5.4 External 16 MHz Crystal Oscillator (HFXOSC)

An external high-frequency 16 MHz crystal oscillator can be used to provide a precise clock source. The crystal oscillator should have a capacitive load of $\leq$12 pF and an ESR $\leq$80 Ω.

When used to drive the PLL, the 16 MHz crystal oscillator output frequency must be divided by two in the first-stage divider of the PLL (i.e., $R = 2$) to provide an 8 MHz reference clock to the VCO.

The input pad of the HFXOSC can also be used to supply an external clock source, in which case, the output pad should be left unconnected.

The HFXOSC input can be used to generate `hfclk` directly if the PLL is set to bypass.

The HFXOSC is controlled via the memory-mapped `hfxosccfg` register.

HF Crystal Oscillator Configuration Register (`hfxosccfg`)				
Register Offset		`0x004`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[29:0]	*Reserved*	RW	X	Reserved
30	hfxoscen	RW	0x1	HFXOSC Enable
31	hfxoscrdy	RO	X	HFXOSC Ready

Table 5.3: HF Crystal Oscillator Configuration Register

The `hfxoscen` bit turns on the crystal driver and is set on wakeup reset, but can be cleared to turn off the crystal driver and reduce power consumption. The `hfxoscrdy` bit indicates if the crystal oscillator output is ready for use.

The hfxoscen bit must also be turned on to use the HFXOSC input pad to connect an external clock source.

5.5 Internal High-Frequency PLL (HFPLL)

The PLL generates a high-frequency clock by multiplying a mid-frequency reference source clock, either the HFROSC or the HFXOSC. The input frequency to the PLL can be in the range 6–48 MHz. The PLL can generate output clock frequencies in the range 48–384 MHz.

The PLL is controlled by a memory-mapped read-write pllcfg register in the PRCI address space. The format of pllcfg is shown in Figure 5.4.

PLL Configuration Register (pllcfg)				
Register Offset		0x008		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[2:0]	pllr	RW	0x1	PLL R input divider value
3	*Reserved*	RW	X	Reserved
[9:4]	pllf	RW	0x1f	PLL F multiplier value
[11:10]	pllq	RW	0x3	PLL Q output divider value
[15:12]	*Reserved*	RW	X	Reserved
16	pllsel	RW	0x0	PLL select
17	pllrefsel	RW	0x1	PLL reference select
18	pllbypass	RW	0x1	PLL bypass
[30:19]	*Reserved*	RW	X	Reserved
31	plllock	RO	X	PLL lock indicator

Table 5.4: PLL Configuration Register

Figure 5.2 shows how the PLL output frequency is set using a combination of three read-write fields: pllr[2:0], pllf[2:0], pllq[1:0]. The frequency constraints must be observed between each stage for correct operation.

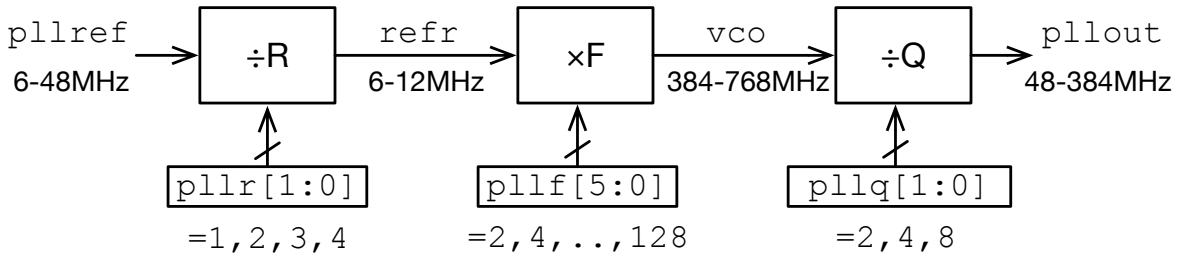

Figure 5.2: Controlling the FE310-G000 PLL output frequency.

The pllr[1:0] field encodes the reference clock divide ratio as a 2-bit binary value, where the value is one less than the divide ratio (i.e., 00=1, 11=4). The frequency of the output of the reference divider (refr) must lie between 6–12 MHz.

The `pllf[5:0]` field encodes the PLL VCO multiply ratio as a 6-bit binary value, N, signifying a divide ratio of $2 \times (N + 1)$ (i.e., `000000`=2, `111111`=128). The frequency of the VCO output (`vco`) must lie between 384–768 MHz. Table 5.5 summarizes the valid settings of the multiply ratio.

Valid PLL Multiply Ratios				
refr (MHz)	**Legal pllf multiplier**		**vco frequency (MHz)**	
	Min	Max	Min	Max
6	64	128	384	768
8	48	96	384	768
10	39	76	390	760
12	32	64	384	768

Table 5.5: Valid PLL multiply ratios. The multiplier setting in the table is given as the actual multiply ratio; the binary value stored in `pllf` field should be $(M/2) - 1$ for a multiply ratio M.

The `pllq[1:0]` field encodes the PLL output divide ratio as follow, `01`=2, `10`=4, `11`=8. The value `00` is not supported. The final output of the PLL must have a frequency that lies between 48–384 MHz.

The one-bit read-write `pllbypass` field in the `pllcfg` register turns off the PLL when written with a 1 and then `pllout` is driven directly by the clock indicated by `pllrefsel`. The other PLL registers can be configured when `pllbypass` is set. The agent that writes `pllcfg` should be running from a different clock source before disabling the PLL. The PLL is also disabled with `pllbypass`=1 after a wakeup reset.

The `pllsel` bit must be set to drive the final `hfclk` with the PLL output, bypassed or otherwise. When `pllsel` is clear, the `hfroscclk` directly drives `hfclk`. The `pllsel` bit is clear on wakeup reset.

The `pllcfg` register is reset to: bypass and power off the PLL `pllbypass`=1; input driven from external HFXOSC oscillator `pllrefsel`=1; PLL not driving system clock `pllsel`=0; and the PLL ratios are set to R=2, F=64, and Q=8 (`pllr`=01, `pllf`=011111, `pllq`=11).

The PLL provides a lock signal which is set when the PLL has achieved lock, and which can be read from the most-significant bit of the `pllcfg` register. The PLL requires up to 100 μs to regain lock once enabled, and the lock signal will not necessarily be stable during this initial lock period so should only be interrogated after this period. The PLL may not achieve lock and the lock signal might not remain asserted if there is excessive jitter in the source clock.

The PLL requires dedicated 1.8 V power supply pads with a supply filter on the circuit board. The supply filter should be a 100 Ω resistor in series with the board 1.8 V supply decoupled with a 100 nF capacitor across the VDDPLL/VSSPLL supply pins. The VSSPLL pin should not be connected to board VSS.

5.6 PLL Output Divider

The `plloutdiv` register controls a clock divider that divides the output of the PLL.

If the `plloutdivby1` bit is set, the PLL output clock is passed through undivided. If `plloutdivby1` is clear, the value N in `plloutdiv` sets the clock-divide ratio to $2 \times (N + 1)$ (between 2–128). The output divider expands the PLL output frequency range to 0.375–384 MHz.

PLL Output Divider Register (plloutdiv)				
Register Offset		0x00C		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	plloutdiv	RW	0	PLL output divider
[7:6]	*Reserved*	RW	X	Reserved
8	plloutdivby1	RW	0x1	PLL output divider bypass
[31:9]	*Reserved*	RW	X	Reserved

Table 5.6: PLL Output Divider Register

The plloutdivby1 register is reset to divide-by-1 (plloutdivby1=1)..

5.7 Internal Programmable Low-Frequency Ring Oscillator (LFROSC)

A second programmable ring oscillator (LFROSC) is used to provide an internal low-frequency ≈32 kHz clock source. The LFROSC can generate frequencies in the range 1.5–230 kHz (±45%).

The lfrosccfg register lives in the AON block as shown in Table 12.1.

At power-on reset, the LFROSC resets to selecting the middle tap (lfrosctrim=16) and ÷5 (lfroscdiv=4), resulting in an output frequency of ≈30 kHz.

The LFROSC can be calibrated in software using a more accurate high-frequency clock source.

LF Ring Oscillator Configuration Register (lfrosccfg)				
Register Offset		0x070		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	lfroscdiv	RW	0x4	LFROSC divider
[15:6]	*Reserved*	RW	X	Reserved
[20:16]	lfrosctrim	RW	0x10	LFROSC trim value
[29:21]	*Reserved*	RW	X	Reserved
30	lfroscen	RW	0x1	LFROSC enable
31	lfroscrdy	RO	X	LFROSC ready

Table 5.7: LF Ring Oscillator Configuration Register

5.8 Alternate Low-Frequency Clock (LFALTCLK)

An external low-frequency clock can be driven on the psdlfaltclk pad, when the psdlfaltclksel is tied low. This mux selection can only be controlled by external pads, it is not controllable by software.

5.9 FE310-G000 Clock Summary

Table 5.8 summarizes the major clocks on the FE310-G000 and their initial reset conditions. At external reset, the AON domain lfclk is clocked by either the LFROSC or psdlfaltclk, as selected by psdlfaltclksel. At wakeup reset, the MOFF domain hfclk is clocked by the HFROSC.

FE310-G000 Clock Sources					
Name	**Reset Source**	**Frequency**			**Notes**
		Reset	**Min**	**Max**	
AON Domain					
LFROSC	`lfroscrst`	32 kHz	1.5 kHz	230 kHz	±45%
`psdlfaltclk`	-	-	0 kHz	500 kHz	When selected by `psdlfaltclksel`
MOFF Domain					
HFROSC	`hfclkrst`	13.8 MHz	0.77 MHz	118 MHz	±45%
HFXOSC crystal HFXOSC input	`hfclkrst`	ON	10 MHz 0 MHz	20 MHz 20 MHz	16 MHz on HiFive External clock source
PLL	`hfclkrst`	OFF	0.375 MHz	384 MHz	
JTAG TCK	`trst_n`	OFF	0 MHz	16 MHz	

Table 5.8: Summary of clock sources on FE310-G000

Chapter 6

Boot Process

This chapter describes the operation of FE310-G000 during the boot process.

6.1 Non-volatile Code Options

There are four possible sources of non-volatile memory from which code can be initially fetched on a FE310-G000 system: Gate ROM, Mask ROM, OTP, and off-chip SPI flash.

6.1.1 Gate ROM (GROM)

The debug ROM is built from gate ROM and contains code for the debug interrupt handler that jumps to debug RAM, as well as code to wait for a debug interrupt.

The default value of `mtvec`, the trap vector base address, is set to `0x0`. Fetches from address `0x0` are hardwired to return 0, which is an illegal instruction, causing another trap, hence causing the processor to spin in a trap loop on any fetch to address 0.

The trap loop is used to hold the processor when waiting for the debugger to download code to be executed. The debugger can issue a debug interrupt, which causes the processor to jump to the debug interrupt handler in debug ROM, which in turn jumps to the code written to the debug RAM. The debug RAM code can be used to bootstrap download of further code.

6.1.2 Mask ROM (MROM)

MROM is fixed at design time, and is located on the peripheral bus on FE310-G000 but instructions fetched from MROM are cached by the E31 core's I-cache. The MROM contains an instruction at address `0x1000` which jumps to the OTP start address at `0x2_0000`.

6.1.3 One-Time Programmable (OTP) Memory

The OTP is located on the peripheral bus, with both a control register interface to program the OTP, and a memory read port interface to fetch words from the OTP. Instruction fetches from the OTP memory read port are cached in the E31 core's instruction cache.

The OTP needs to be programmed before use and can only be programmed by code running on the E31 core. The OTP bits contain all 0s prior to programming.

6.1.4 Quad SPI Flash Controller (QSPI)

The dedicated QSPI flash controller connects to external SPI flash devices that are used for execute-in-place code. SPI flash is not available in certain scenarios such as package testing or board designs not using SPI flash (e.g., just using on-chip OTP).

Off-chip SPI devices can vary in number of supported I/O bits (1, 2, or 4). SPI flash bits contain all 1s prior to programming.

6.2 Reset and Trap Vectors

FE310-G000 fetches the first instruction out of reset from `0x1000`. The instruction stored there jumps straight to OTP at `0x2_0000`, and will either enter a trap loop if the OTP is not programmed, or start running the OTP code.

Chapter 7

Configuration String

The initial version of the FE310-G000 has a configuration string of:

```
/cs-v1/;
/{
  model = \"SiFive,FE310G-0000-Z0\";
  compatible = \"sifive,fe300\";
  /include/ 0x20004;
};
```

This string is included in the Mask ROM, and its address is a pointer stored in the MaskROM at 0x100C.

Chapter 8

E31 RISC-V Core

This chapter describes the 32-bit E31 RISC-V processor core used in the FE310-G000. The E31 processor core comprises an instruction memory system, an instruction fetch unit, an execution pipeline, a data memory system, and support for global, software, and timer interrupts.

The E31 feature set is summarized in Table 8.1.

E31 Feature Set	
Feature	**Description**
ISA	RV32IMAC.
Instruction Cache	16 KiB 2-way instruction cache.
Data Tightly Integrated Memory	16 KiB DTIM.
Modes	The E31 supports the following modes: Machine Mode.

Table 8.1: E31 Feature Set

8.1 Instruction Memory System

The instruction memory system consists of a dedicated 16 KiB 2-way set-associative instruction cache. The access latency of all blocks in the instruction memory system is one clock cycle. The instruction cache is not kept coherent with the rest of the platform memory system. Writes to instruction memory must be synchronized with the instruction fetch stream by executing a FENCE.I instruction.

The instruction cache has a line size of 32 B and a cache line fill will trigger a burst access. The core will cache instructions from executable addresses. Please see the E31 Memory Map in Chapter 3 for a description of executable address regions which are denoted by the attribute X.

Trying to execute an instruction from a non-executable address will result in a synchronous trap.

8.2 Instruction Fetch Unit

The E31 instruction fetch unit contains branch prediction hardware to improve performance of the processor core. The branch predictor comprises a 40-entry branch target buffer (BTB) which predicts the target of taken branches, a 128-entry branch history table (BHT), which predicts the

direction of conditional branches, and a 2-entry return-address stack (RAS) which predicts the target of procedure returns. The branch predictor has a one-cycle latency, so that correctly predicted control-flow instructions result in no penalty. Mispredicted control-flow instructions incur a three-cycle penalty.

The E31 implements the standard Compressed (C) extension to the RISC-V architecture which allows for 16-bit RISC-V instructions.

8.3 Execution Pipeline

The E31 execution unit is a single-issue, in-order pipeline. The pipeline comprises five stages: instruction fetch, instruction decode and register fetch, execute, data memory access, and register writeback.

The pipeline has a peak execution rate of one instruction per clock cycle, and is fully bypassed so that most instructions have a one-cycle result latency. There are several exceptions:

- LW has a two-cycle result latency, assuming a cache hit.
- LH, LHU, LB, and LBU have a three-cycle result latency, assuming a cache hit.
- CSR reads have a three-cycle result latency.
- MUL, MULH, MULHU, and MULHSU have a 5-cycle result latency.
- DIV, DIVU, REM, and REMU have between a 2-cycle and 33-cycle result latency, depending on the operand values.

The pipeline only interlocks on read-after-write and write-after-write hazards, so instructions may be scheduled to avoid stalls.

The E31 implements the standard Multiply (M) extension to the RISC-V architecture for integer multiplication and division. The E31 has a 8-bit per cycle hardware multiply and a 1-bit per cycle hardware divide.

Branch and jump instructions transfer control from the memory access pipeline stage. Correctly-predicted branches and jumps incur no penalty, whereas mispredicted branches and jumps incur a three-cycle penalty.

Most CSR writes result in a pipeline flush with a five-cycle penalty.

8.4 Data Memory System

The E31 data memory system has a tightly integrated 16 KiB data memory (DTIM). The access latency is two clock cycles for full words and three clock cycles for smaller quantities. Misaligned accesses are not supported in hardware and result in a trap to allow software emulation.

Stores are pipelined and commit on cycles where the data memory system is otherwise idle. Loads to addresses currently in the store pipeline result in a five-cycle penalty.

8.5 Atomic Memory Operations

The E31 core supports the RISC-V standard Atomic (A) extension on the DTIM and the peripheral memory region. Atomic memory operations to regions that do not support them generate an access exception precisely at the core.

The load-reserved and store-conditional instructions are only supported on cached regions, hence generate an access exception on DTIM and other uncached memory regions.

See The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1 [1] for more information on the instructions added by this extension.

Chapter 9

Interrupts

This chapter describes how interrupt concepts in the RISC-V architecture apply to the FE310-G000. The definitive resource for information about the RISC-V interrupt architecture is The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10 [2].

9.1 Interrupt Concepts

The FE310-G000 has support for the following interrupts: local (including software and timer), and global.

Local interrupts are signaled directly to an individual hart with a dedicated interrupt value. This allows for reduced interrupt latency as there is no arbitration required to determine which hart will service a given request, nor additional memory accesses required to determine the cause of the interrupt. Software and timer interrupts are local interrupts generated by the Core Local Interruptor (CLINT). The FE310-G000 contains no other local interrupt sources.

Global interrupts by contrast, are routed through a Platform-Level Interrupt Controller (PLIC), which can direct interrupts to any hart in the system via the external interrupt. Decoupling global interrupts from the hart(s) allows the design of the PLIC to be tailored to the platform, permitting a broad range of attributes like the number of interrupts and the prioritization and routing schemes.

This chapter describes the E31 interrupt architecture. Chapter 10 describes the global interrupt architecture and the PLIC design. Chapter 11 describes the Core Local Interruptor.

The FE310-G000 interrupt architecture is depicted in Figure 9.1.

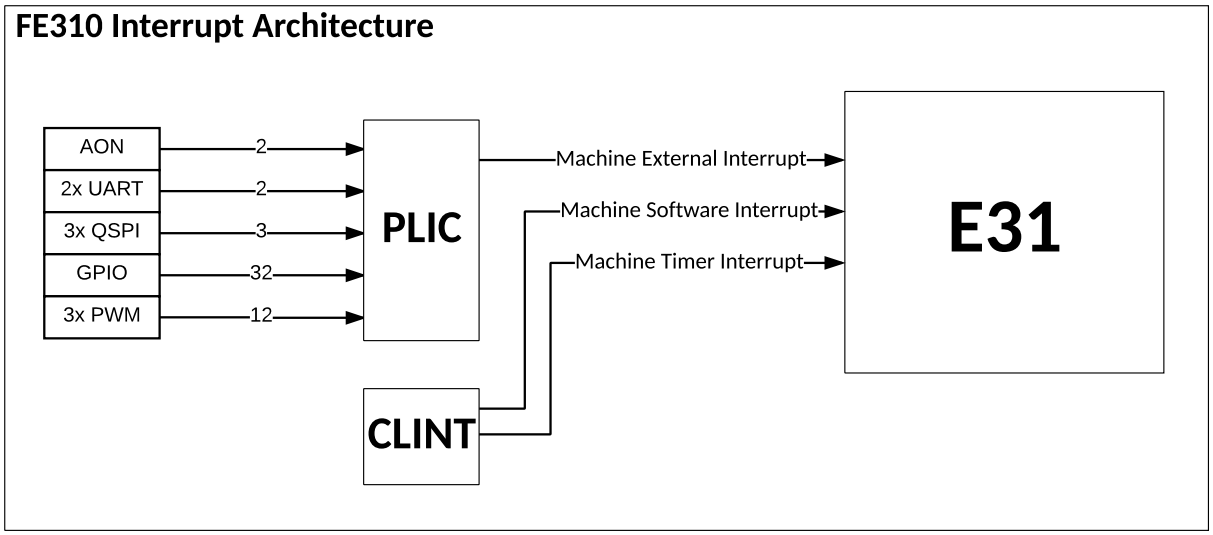

Figure 9.1: E31 Interrupt Architecture Block Diagram.

9.2 Interrupt Entry and Exit

When a RISC-V hart takes an interrupt the following will occur:

- The value of `mstatus.MIE` is copied into `mstatus.MPIE`, then `mstatus.MIE` is cleared, effectively disabling interrupts.
- The current `pc` is copied into the `mepc` register, and then `pc` is set to the value of `mtvec`. In the case where vectored interrupts are enabled, `pc` is set to `mtvec.BASE` + 4×exception code.
- The privilege mode prior to the interrupt is encoded in `mstatus.MPP`.

At this point control is handed over to software in the interrupt handler with interrupts disabled. Interrupts can be re-enabled by explicitly setting `mstatus.MIE`, or by executing an MRET instruction to exit the handler. When an MRET instruction is executed, the following will occur:

- The privilege mode is set to the value encoded in `mstatus.MPP`.
- The value of `mstatus.MPIE` is copied into `mstatus.MIE`.
- The `pc` is set to the value of `mepc`.

At this point control is handed over to software.

The Control and Status Registers involved in handling RISC-V interrupts are described in Section 9.3.

9.3 Interrupt Control Status Registers

The SiFive E31 specific implementation of interrupt CSRs is described below. For a complete description of RISC-V interrupt behavior and how to access CSRs, please consult The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10 [2].

9.3.1 Machine Status Register (`mstatus`)

The `mstatus` register keeps track of and controls the hart's current operating state including whether or not interrupts are enabled. A summary of the `mstatus` fields related to interrupts in the E31 is provided in Table 9.1; note that this is not a complete description of `mstatus` as it contains fields unrelated to interrupts. For the full description of `mstatus` please consult the The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10 [2].

Machine Status Register			
CSR	`mstatus`		
Bits	**Field Name**	**Attr.**	**Description**
[2:0]	*Reserved*	WPRI	
3	MIE	RW	Machine Interrupt Enable
[6:4]	*Reserved*	WPRI	
7	MPIE	RW	Machine Previous Interrupt Enable
[10:8]	*Reserved*	WPRI	
[12:11]	MPP	RW	Machine Previous Privilege Mode

Table 9.1: E31 `mstatus` register (partial)

Interrupts are enabled by setting the MIE bit in `mstatus` and by enabling the desired individual interrupt in the `mie` register described in Section 9.3.2.

9.3.2 Machine Interrupt Enable Register (`mie`)

Individual interrupts are enabled by setting the appropriate bit in the `mie` register. The E31 `mie` register is described in Table 9.2.

Machine Interrupt Enable Register			
CSR	`mie`		
Bits	**Field Name**	**Attr.**	**Description**
[2:0]	*Reserved*	WIRI	
3	MSIE	RW	Machine Software Interrupt Enable
[6:4]	*Reserved*	WIRI	
7	MTIE	RW	Machine Timer Interrupt Enable
[10:8]	*Reserved*	WIRI	
11	MEIE	RW	Machine External Interrupt Enable
[15:12]	*Reserved*	WPRI	

Table 9.2: E31 `mie` register

9.3.3 Machine Interrupt Pending (mip)

The machine interrupt pending (mip) register indicates which interrupts are currently pending. The E31 mip register is described in Table 9.3.

Machine Interrupt Pending Register			
CSR	mip		
Bits	**Field Name**	**Attr.**	**Description**
[2:0]	*Reserved*	WPRI	
3	MSIP	RO	Machine Software Interrupt Pending
[6:4]	*Reserved*	WPRI	
7	MTIP	RO	Machine Timer Interrupt Pending
[10:8]	*Reserved*	WPRI	
11	MEIP	RO	Machine External Interrupt Pending
[15:12]	*Reserved*	WPRI	

Table 9.3: E31 mip register

9.3.4 Machine Cause Register (mcause)

When a trap is taken in machine mode, mcause is written with a code indicating the event that caused the trap. When the event that caused the trap is an interrupt, the most-significant bit of mcause is set to 1, and the least-significant bits indicate the interrupt number, using the same encoding as the bit positions in mip. For example, a Machine Timer Interrupt causes mcause to be set to 0x8000_0007. mcause is also used to indicate the cause of synchronous exceptions, in which case the most-significant bit of mcause is set to 0. Refer to Table 9.5 for a list of synchronous exception codes.

Machine Cause Register			
CSR	mcause		
Bits	**Field Name**	**Attr.**	**Description**
[30:0]	Exception Code	WLRL	A code identifying the last exception.
31	Interrupt	WARL	1 if the trap was caused by an interrupt; 0 otherwise.

Table 9.4: E31 mcause register

Interrupt Exception Codes		
Interrupt	**Exception Code**	**Description**
1	0–2	*Reserved*
1	3	Machine software interrupt
1	4–6	*Reserved*
1	7	Machine timer interrupt
1	8–10	*Reserved*
1	11	Machine external interrupt
1	12–15	*Reserved*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8–10	*Reserved*
0	11	Environment call from M-mode
0	12–31	*Reserved*

Table 9.5: E31 `mcause` Exception Codes

9.3.5 Machine Trap Vector (`mtvec`)

All interrupts trap to a single address defined in the `mtvec` register. It is up to the interrupt handler to read `mcause` and react accordingly.

Machine Trap Vector Register			
CSR	`mtvec`		
Bits	**Field Name**	**Attr.**	**Description**
[31:2]	BASE[31:2]	WARL	Interrupt Vector Base Address. Note, BASE[1:0] is not present in this register and is implicitly 0.

Table 9.6: E31 `mtvec` register

See Table 9.5 for the E31 interrupt exception code values.

9.4 Interrupt Priorities

Individual priorities of global interrupts are determined by the PLIC, as discussed in Chapter 10.

E31 interrupts are prioritized as follows, in decreasing order of priority:

- Machine external interrupts
- Machine software interrupts
- Machine timer interrupts

9.5 Interrupt Latency

Interrupt latency for the FE310-G000, as counted by the numbers of cycles it takes from signaling of the interrupt to the hart to the first instruction fetch of the handler, is 4 cycles.

Global interrupts routed through the PLIC incur additional latency of 3 cycles.

This is a best case cycle count and assumes the handler is cached or located in ITIM. It does not take into account additional latency from a peripheral source.

Additionally, the hart will not abandon a Divide instruction in flight. This means if an interrupt handler tries to use a register which is the destination register of a divide instruction, the pipeline will stall until the divide is complete.

Chapter 10

Platform-Level Interrupt Controller (PLIC)

This chapter describes the operation of the platform-level interrupt controller (PLIC) on the SiFive E31. The PLIC complies with The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10 [2].

10.1 Memory Map

The memory map for the SiFive E31 PLIC control registers is shown in Table 10.1. The PLIC memory map has been designed to only require naturally aligned 32-bit memory accesses.

PLIC Register Map				
Address	**Width**	**Attr.**	**Description**	**Notes**
0x0C00_0000			*Reserved*	
0x0C00_0004	4B	RW	source 1 priority	
0x0C00_0008	4B	RW	source 2 priority	See Section 10.3 for more information
. . .				
0x0C00_00CC	4B	RW	source 51 priority	
0x0C00_00D0			*Reserved*	
. . .				
0x0C00_0FFF				
0x0C00_1000	4B	RO	Start of pending array	See Section 10.4 for more information
. . .				
0x0C00_1004	4B	RO	Last word of pending array	
0x0C00_1008			*Reserved*	
. . .				
0x0C00_1FFF				
0x0C00_2000	4B	RW	Start Hart 0 M-Mode interrupt enables	See Section 10.5 for more information
0x0C00_2004	4B	RW	End Hart 0 M-Mode interrupt enables	
0x0C00_2020			*Reserved*	
. . .				
0x0C1F_FFFF				
0x0C20_0000	4B	RW	Hart 0 M-Mode priority threshold	See Section 10.6 for more information
0x0C20_0004	4B	RW	Hart 0 M-Mode claim/complete	See Section 10.7 for more information
0x0C20_0008			*Reserved*	
. . .				
0x0FFF_FFFF				

Table 10.1: SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are supported.

10.2 Interrupt Sources

The mapping of FE310-G000 interrupt sources to their corresponding ID's are provided in Table 10.2.

PLIC IRQ Mapping		
IRQ	**Peripheral**	**Description**
0	*No Interrupt*	
1 2	Watchdog RTC	AON peripheral interrupts See AON Chapter 12 for more information.
3 4	UART0 UART1	UART peripheral interrupts See UART Chapter 19 for more information.
5 6 7	QSPI0 QSPI1 QSPI2	QSPI peripheral interrupts See QSPI Chapter 18 for more information.
8 . . . 39	GPIO 0 GPIO 31	GPIO peripheral interrupts See GPIO Chapter 17 for more information.
40 . . . 43 44 . . . 47 48 . . . 51	PWM0CMP0 PWM0CMP3 PWM1CMP0 PWM1CMP3 PWM2CMP0 PWM2CMP3	PWM peripheral interrupts See PWM Chapter 20 for more information.

Table 10.2: FE310-G000 Interrupt Sources

10.3 Interrupt Priorities

Each PLIC interrupt source can be assigned a priority by writing to its 32-bit memory-mapped `priority` register. The E31 supports 7 levels of priority. A priority value of 0 is reserved to mean "never interrupt" and effectively dissables the interrupt. Priority 1 is the lowest active priority, and priority 7 is the highest. Ties between global interrupts of the same priority are broken by the Interrupt ID; interrupts with the lowest ID have the highest effective priority. Please see Table 10.3 for the detailed register description.

PLIC Interrupt Priority Register (`priority`)				
Base Address		`0x0C00_0000 + 4×Interrupt ID`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[2:0]	Priority	WARL	X	Sets the priority for a given global interrupt.
31:3]	*Reserved*	WIRI	X	

Table 10.3: PLIC Interrupt Priority Registers

10.4 Interrupt Pending Bits

The current status of the interrupt source pending bits in the PLIC core can be read from the pending array, organized as 2 words of 32 bits. The pending bit for interrupt ID N is stored in bit $(N \bmod 32)$ of word $(N/32)$. As such, the E31 has 2 interrupt pending registers. Bit 0 of word 0, which represents the non-existent interrupt source 0, is hardwired to zero.

A pending bit in the PLIC core can be cleared by setting the associated enable bit then performing a claim as as described in Section 10.7.

PLIC Interrupt Pending Register 1 (pending1)				
Base Address		0x0C00_1000		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 0 Pending	RO	0	Non-existent global interrupt 0 is hardwired to zero
1	Interrupt 1 Pending	RO	0	Pending bit for global interrupt 1
2	Interrupt 2 Pending	RO	0	Pending bit for global interrupt 2
...				
31	Interrupt 31 Pending	RO	0	Pending bit for global interrupt 31

Table 10.4: PLIC Interrupt Pending Register 1

PLIC Interrupt Pending Register 2 (pending2)				
Base Address		0x0C00_1004		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
1	Interrupt 32 Pending	RO	0	Pending bit for global interrupt 32
...				
19	Interrupt 51 Pending	RO	0	Pending bit for global interrupt plicinputs
[31:20]	*Reserved*	WIRI	X	

Table 10.5: PLIC Interrupt Pending Register 2

10.5 Interrupt Enables

Each global interrupt can be enabled by setting the corresponding bit in the `enables` register. The `enables` registers are accessed as a contiguous array of 2×32-bit words, packed the same way as the `pending` bits. Bit 0 of enable word 0 represents the non-existent interrupt ID 0 and is hardwired to 0.

Only 32-bit word accesses are supported by the `enables` array in SiFive RV32 systems.

PLIC Interrupt Enable Register 1 (enable1)				
Base Address		0x0C00_2000		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 0 Enable	RW	X	Non-existent global interrupt 0 is hardwired to zero
1	Interrupt 1 Enaable	RW	X	Enable bit for global interrupt 1
2	Interrupt 2 Enaable	RW	X	Enable bit for global interrupt 2
...				
31	Interrupt 31 Enaable	RW	X	Enable bit for global interrupt 31

Table 10.6: PLIC Interrupt Enable Register 1

PLIC Interrupt Enable Register 2 (enable2)				
Base Address		0x0C00_2004		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 32 Enable	RW	X	Enable bit for global interrupt 32
...				
19	Interrupt 51 Enable	RW	X	Enable bit for global interrupt 51
[31:20]	*Reserved*	WIRI	X	

Table 10.7: PLIC Interrupt Enable Register 2

10.6 Priority Thresholds

The E31 supports setting of a interrupt priority threshold via the `threshold` register. The `threshold` is a **WARL** field, where the E31 supports a maximum threshold of 7.

The E31 will mask all PLIC interrupts of a priority less than or equal to `threshold`. For example, a `threshold` value of zero permits all interrupts with non-zero priority, whereas a value of 7 masks all interrupts.

PLIC Interrupt Priority Threshold Register (threshold)				
Base Address		0x0C20_0000		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[2:0]	Threshold	RW	X	Sets the priority threshold
[31:3]	*Reserved*	WIRI	X	

Table 10.8: PLIC Interrupt Threshold Registers

10.7 Interrupt Claim Process

The E31 can perform an interrupt claim by reading the `claim/complete` register (Table 10.9), which returns the ID of the highest priority pending interrupt or zero if there is no pending interrupt. A successful claim will also atomically clear the corresponding pending bit on the interrupt source.

The E31 can perform a claim at any time, even if the MEIP bit in the `mip` (Section 9.3.3) register is not set.

The claim operation is not affected by the setting of the priority threshold register.

10.8 Interrupt Completion

The E31 signals it has completed executing an interrupt handler by writing the interrupt ID it received from the claim to the `claim/complete` register (Table 10.9). The PLIC does not check whether the completion ID is the same as the last claim ID for that target. If the completion ID does not match an interrupt source that is currently enabled for the target, the completion is silently ignored.

PLIC Claim/Complete Register (`claim`)				
Base Address		`0x0C20_0004`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[31:0]	Interrupt Claim	RW	X	A read of zero indicates that no interrupts are pending. A non-zero read contains the id of the highest pending interrupt. A write to this register signals completion of the interrupt id written

Table 10.9: PLIC Interrupt Claim/Complete Register

Chapter 11

Core Local Interruptor (CLINT)

The CLINT block holds memory-mapped control and status registers associated with software and timer interrupts. The E31 CLINT complies with The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10 [2].

11.1 E31 CLINT Address Map

Table 11.1 shows the memory map for CLINT on SiFive E31.

CLINT Register Map				
Address	**Width**	**Attr.**	**Description**	**Notes**
0x0200_0000	4B	RW	`msip` for hart 0	MSIP Registers
0x0200_0004 ... 0x0200_3FFF			*Reserved*	
0x0200_4000	8B	RW	`mtimecmp` for hart 0	Timer compare register
0x0200_4008 ... 0x0200_BFF7			*Reserved*	
0x0200_BFF8	8B	RO	`mtime`	Timer register
0x0200_C000 ... 0x0200_FFFF			*Reserved*	

Table 11.1: SiFive E31 CLINT Memory Map.

11.2 MSIP Registers

Machine-mode software interrupts are generated by writing to the memory-mapped control register `msip`. The `msip` register is a 32-bit wide **WARL** register, where the LSB is reflected in the `msip` bit of the `mip` register. Other bits in the `msip` registers are hardwired to zero. On reset, the `msip` registers are cleared to zero.

Software interrupts are most useful for interprocessor communication in multi-hart systems, as harts may write each other's `msip` bits to effect interprocessor interrupts.

11.3 Timer Registers

`mtime` is a 64-bit read-write register that contains the number of cycles counted from the `lfclk` signal shown described in Chapter 12. A timer interrupt is pending whenever `mtime` is greater than or equal to the value in the `mtimecmp` register. The timer interrupt is reflected in the `mtip` bit of the `mip` register described in Chapter 9.

On reset, `mtime` is cleared to zero. The `mtimecmp` registers are not reset.

Chapter 12

Always-On (AON) Domain

The FE310-G000 supports an always-on (AON) domain that includes a real-time counter, a watchdog timer, backup registers, and reset and power-management circuitry for the rest of the system. Figure 12.1 shows an overview of the AON block.

12.1 AON Power Source

The AON domain is continuously powered from an off-chip power source, either a regulated power supply or a battery.

12.2 AON Clocking

The AON domain is clocked by the low-frequency clock, `lfclk`. The core domain's Tilelink peripheral bus uses the high-frequency `tlclk`. A HF-LF power-clock-domain crossing (VCDC) bridges between the two power and clock domains.

12.3 AON Reset Unit

An AON reset is the widest reset on the FE310-G000 and resets all state.

An AON reset can be triggered by an external active-low reset pin (`erst_n`), or expiration of the watchdog timer (`wdogrst`).

These sources provide a short initial reset pulse `frst`, which is extended by a reset stretcher to provide the LFROSC reset signal `lfroscrst` and a longer stretched internal reset, `srst`.

The `lfroscrst` signal is used to initialize the ring oscillator in the LFROSC. This oscillator provides `lfclk`, which is used to clock the AON. `lfclk` is also used as the clock input to `mtime` in the CLINT.

The `srst` strobe is passed to a reset synchronizer clocked by `lfclk` to generate `aonrst`, an asychronous-onset/synchronous-release reset signal used to reset most of the AON block.

The "mostly off" (MOFF) resets `coreclkrst` and `corerst` are generated by the Power Management Unit (PMU) state machine after `aonrst` is deasserted.

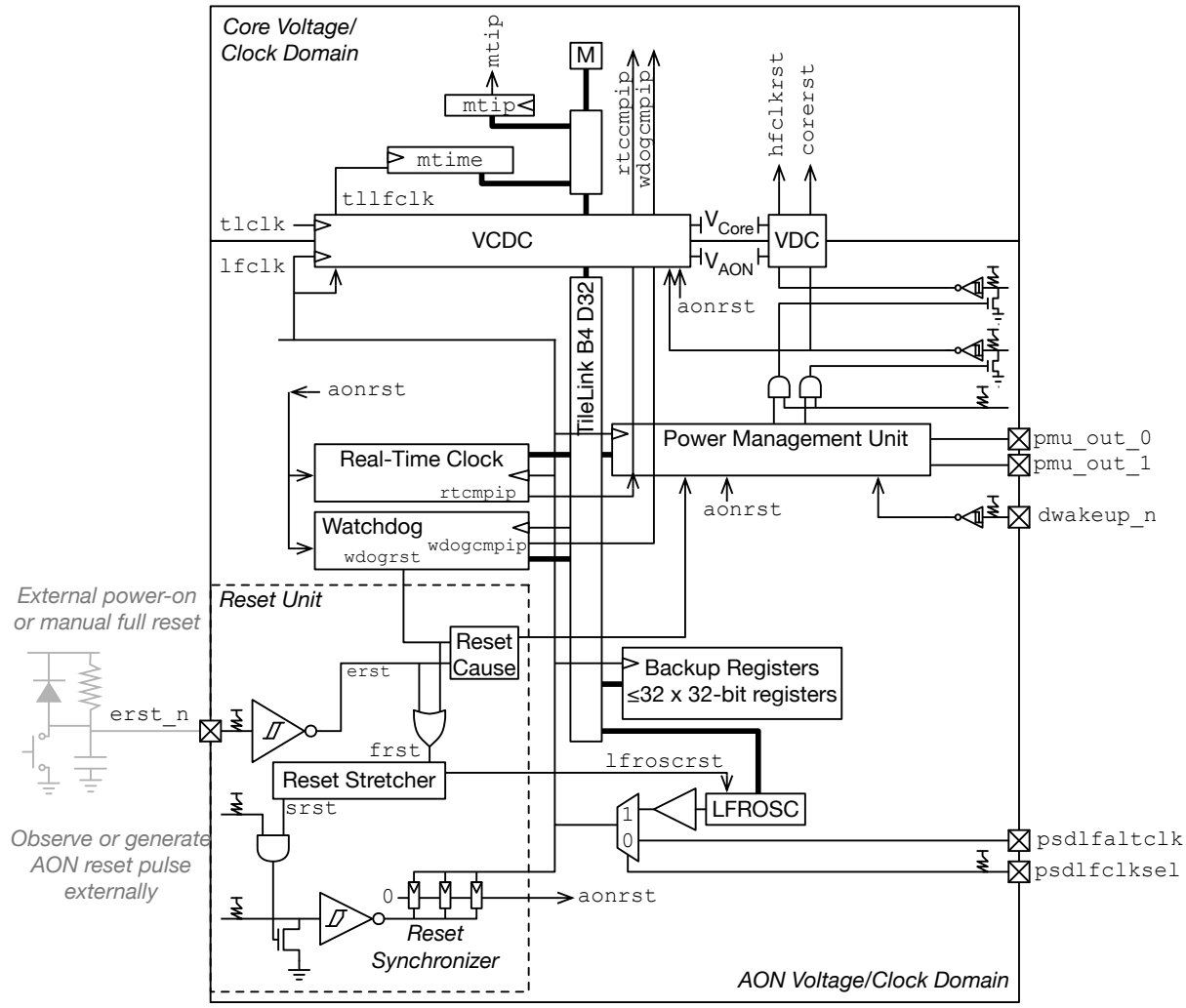

Figure 12.1: FE310-G000 Always-On Domain.

12.3.1 External Reset Circuit

The FE310-G000 can be reset by pulling down on the external reset pin (`erst_n`), which has a weak pullup. An external power-on reset circuit consisting of a resistor and capacitor can be provided to generate a sufficiently long pulse to allow supply voltage to rise and then initiate the reset stretcher.

The external reset circuit can include a diode as shown to quickly discharge the capacitor after the supply is removed to rearm the external power-on reset circuit.

A manual reset button can be connected in parallel with the capacitor.

12.3.2 Reset Cause

The cause of an AON reset is latched in the Reset Unit and can be read from the `pmucause` register in the PMU, as described in Chapter 13.

12.4 Watchdog Timer (WDT)

The watchdog timer can be used to provide a watchdog reset function, or a periodic timer interrupt. The watchdog is described in detail in Chapter 14.

12.5 Real-Time Clock (RTC)

The real-time clock maintains time for the system and can also be used to generate interrupts for timed wakeup from sleep-mode or timer interrupts during normal operation. The Real-Time Clock is described in detail in Chapter 15.

12.6 Backup Registers

The backup registers provide a place to store critical data during sleep. The FE310-G000 has 16×32-bit backup registers.

12.7 Power-Management Unit (PMU)

The power-management unit (PMU) sequences the system power supplies and reset signals when transitioning into and out of sleep mode. The PMU also monitors AON signals for wakeup conditions. The PMU is described in detail in Chapter 13.

12.8 AON Memory Map

Table 12.1 shows the memory map of the AON block.

AON Memory Map				
Address	**Width**	**Attr.**	**Description**	**Notes**
0x1000_0000	4B	RW	`wdogcfg`	
0x1000_0004	4B		*Reserved*	
0x1000_0008	4B	RW	`wdogcount`	
0x1000_000C	4B		*Reserved*	
0x1000_0010	4B	RW	`wdogs`	Watchdog Timer Registers.
0x1000_0014	4B		*Reserved*	See Chapter 14
0x1000_0018	4B	RW	`wdogfeed`	
0x1000_001C	4B	RW	`wdogkey`	
0x1000_0020	4B	RW	`wdogcmp`	
...			*Reserved*	
0x1000_0040	4B	RW	`rtccfg`	
0x1000_0044	4B		*Reserved*	
0x1000_0048	4B	RW	`rtclo`	
0x1000_004C	4B	RW	`rtchi`	
0x1000_0050	4B	RW	`rtcs`	Real-Time Clock Registers.
0x1000_0054	4B		*Reserved*	See Chapter 15
0x1000_0058	4B		*Reserved*	
0x1000_005C	4B		*Reserved*	
0x1000_0060	4B	RW	`rtccmp`	
...			*Reserved*	
0x1000_0070	4B	RW	`lfrosccfg`	AON Clock Configuration.
...			*Reserved*	See Section 5.7
...			*Reserved*	
0x1000_0080	4B	RW	`backup0`	Backup Registers. See
0x1000_0084	4B	RW	`backup1`	Section 12.6
...				
0x1000_00FC	4B	RW	`backup31`	
0x1000_0100	64B	RW	PMU wakeup program memory	
0x1000_0120	64B	RW	PMU sleep program memory	
0x1000_0140	4B	RW	`pmuie`	Power Management Unit.
0x1000_0144	4B	RW	`pmucause`	See Chapter 13
0x1000_0148	4B	RW	`pmusleep`	
0x1000_014C	4B	RW	`pmukey`	

Table 12.1: The FE310-G000 AON Memory Map.

Chapter 13

Power-Management Unit (PMU)

The FE310-G000 power-management unit (PMU) is implemented within the AON domain and sequences the system's power supplies and reset signals during power-on reset and when transitioning the "mostly off" (MOFF) block into and out of sleep mode.

13.1 PMU Overview

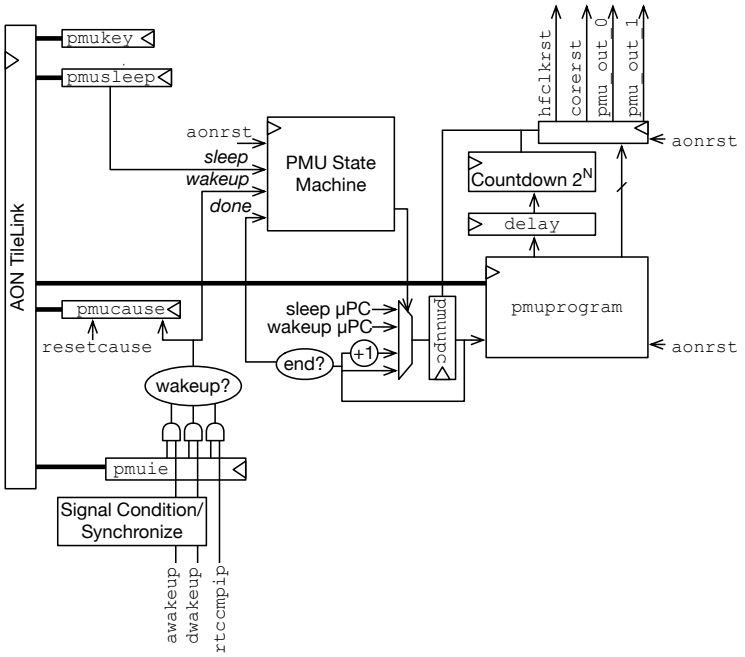

Figure 13.1: FE310-G000 Power-Management Unit.

The PMU is a synchronous unit clocked by the `lfclk` in the AON domain. The PMU handles reset, wakeup, and sleep actions initiated by power-on reset, wakeup events, and sleep requests. When the MOFF block is powered off, the PMU monitors AON signals to initiate the wakeup sequence. When the MOFF block is powered on, the PMU awaits sleep requests from the MOFF block, which initiate the sleep sequence. The PMU is based around a simple programmable microcode

sequencer that steps through short programs to sequence output signals that control the power supplies and reset signals to the clocks, core, and pads in the system.

13.2 Memory Map

The memory map for the PMU is shown in Table 13.1. The memory map has been designed to only require naturally aligned 32-bit memory accesses.

PMU Register Offsets				
Offset	**Width**	**Attr.**	**Description**	**Notes**
0x100	4B	RW	pmuwakeupi0	Wakeup program instruction 0
0x104	4B	RW	pmuwakeupi1	Wakeup program instruction 1
0x108	4B	RW	pmuwakeupi2	Wakeup program instruction 2
0x10c	4B	RW	pmuwakeupi3	Wakeup program instruction 3
0x110	4B	RW	pmuwakeupi4	Wakeup program instruction 4
0x114	4B	RW	pmuwakeupi5	Wakeup program instruction 5
0x118	4B	RW	pmuwakeupi6	Wakeup program instruction 6
0x11c	4B	RW	pmuwakeupi7	Wakeup program instruction 7
0x120	4B	RW	pmusleepi0	Sleep program instruction 0
0x124	4B	RW	pmusleepi1	Sleep program instruction 1
0x128	4B	RW	pmusleepi2	Sleep program instruction 2
0x12c	4B	RW	pmusleepi3	Sleep program instruction 3
0x130	4B	RW	pmusleepi4	Sleep program instruction 4
0x134	4B	RW	pmusleepi5	Sleep program instruction 5
0x138	4B	RW	pmusleepi6	Sleep program instruction 6
0x13c	4B	RW	pmusleepi7	Sleep program instruction 7
0x140	4B	RW	pmuie	PMU interrupt enables
0x144	4B	RW	pmucause	PMU wakeup cause
0x148	4B	RW	pmusleep	Initiate sleep sequence
0x14c	4B	RW	pmukey	PMU key register

Table 13.1: PMU register offsets within the AON memory map. Only naturally aligned 32-bit memory accesses are supported.

13.3 PMU Key Register (pmukey)

The pmukey register has one bit of state. To prevent spurious sleep or PMU program modification, all writes to PMU registers must be preceded by an unlock operation to the pmukey register location, which sets pmukey. The value 0x51F15E must be written to the pmukey register address to set the state bit before any write access to any other PMU register. The state bit is reset at AON reset, and after any write to a PMU register.

PMU registers may be read without setting pmukey.

13.4 PMU Program

The PMU is implemented as a programmable sequencer to support customization and tuning of the wakeup and sleep sequences. A wakeup or sleep program comprises eight instructions. An instruction consists of a delay, encoded as a binary order of magnitude, and a new value for all

of the PMU output signals to assume after that delay. The PMU instruction format is shown in Figure 13.2. For example, the instruction `0x108` delays for 2^8 clock cycles, then raises `hfclkrst` and lowers all other output signals.

The PMU output signals are registered and only toggle on PMU instruction boundaries. The output registers are all asynchronously set to 1 by `aonrst`.

Figure 13.2: PMU instruction format.

At power-on reset, the PMU program memories are reset to conservative defaults. Table 13.2 shows the default wakeup program, and Table 13.3 shows the default sleep program.

Default PMU Wakeup Program		
Program Instruction	**Value**	**Meaning**
0	0x1f0	Assert all resets and enable all power supplies
1	0x0f8	Idle 2^8 cycles, then deassert `hfclkrst`
2	0x030	Deassert `corerst` and `padrst`
3-7	0x030	*Repeats*

Table 13.2: Default PMU wakeup program.

Default PMU Sleep Program		
Program Instruction	**Value**	**Meaning**
0	0x0f0	Assert `corerst`
1	0x1f0	Assert `hfclkrst`
2	0x1d0	Deassert `pmu_out_1`
3	0x1c0	Deassert `pmu_out_0`
4-7	0x1c0	*Repeats*

Table 13.3: Default PMU sleep program.

13.5 Initiate Sleep Sequence Register (`pmusleep`)

Writing any value to the `pmusleep` register initiates the sleep sequence stored in the sleep program memory. The MOFF block will sleep until an event enabled in the `pmuie` register occurs.

13.6 Wakeup Signal Conditioning

The PMU can be woken by the external `dwakeup` signal, which is preconditioned by the signal conditioning block.

The dwakeup signal has a fixed deglitch circuit that requires the dwakeup signal remain asserted for two AON clock edges before being accepted. The conditioning circuit also resynchronizes the dwakeup signal to the AON lfclk.

13.7 PMU Interrupt Enables (pmuie) and Wakeup Cause (pmucause)

The pmuie register indicates which events can wake the MOFF block from sleep. The dwakeup bit indicates that a logic 0 on the dwakeup_n pin can rouse MOFF. The rtc bit indicates that the RTC comparator can rouse MOFF.

PMU Interrupt Enable Register (pmuie)				
Register Offset		0x140		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	*Reserved*	RO	1	
1	rtc	RW	X	RTC Comparator active
2	dwakeup	RW	X	dwakeup_n pin active.
3-31	*Reserved*	RO	0x0	

Table 13.4: PMU Interrupt Enable Register

Following a wakeup, the pmucause register indicates which event caused the wakeup. The value in the wakeupcause field corresponds to the bit position of the event in pmuie, e.g., a value of 2 indicates dwakeup. The value 0 indicates a wakeup from reset. These causes are shown in Table 13.6

In the event of a wakeup from reset, the resetcause field indicates which reset source triggered the wakeup. Table 13.7 lists the values the resetcause field may take. The value in resetcause persists until the next reset.

PMU Wakeup Cause Register (pmucause)				
Register Offset		0x144		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[1:0]	wakeupcause	RO	0x0	Wakeup cause, See Table 13.6
[7:2]	*Reserved*	RO	0x0	
[9:8]	resetcause	RO	*Varies*	Reset cause, See Table 13.7
[31:10]	*Reserved*	RO	0x0	

Table 13.5: PMU Wakeup Cause Register

Wakeup Cause Values	
Index	**Meaning**
0	Reset
1	RTC Wakup (rtc)
2	Digitial input wakeup (dwakeup)

Table 13.6: Wakeup cause values.

Reset Cause Values	
Index	**Meaning**
1	External reset
2	Watchdog timer reset

Table 13.7: Reset cause values.

Chapter 14

E300 Watchdog Timer (WDT)

The watchdog timer (WDT) is used to cause a full power-on reset if either hardware or software errors cause the system to malfunction. The WDT can also be used as a programmable periodic interrupt source if the watchdog functionality is not required. The WDT is implemented as an upcounter in the Always-On domain that must be reset at regular intervals before the count reaches a preset threshold, else it will trigger a full power-on reset. To prevent errant code from resetting the counter, the WDT registers can only be updated by presenting a WDT key sequence.

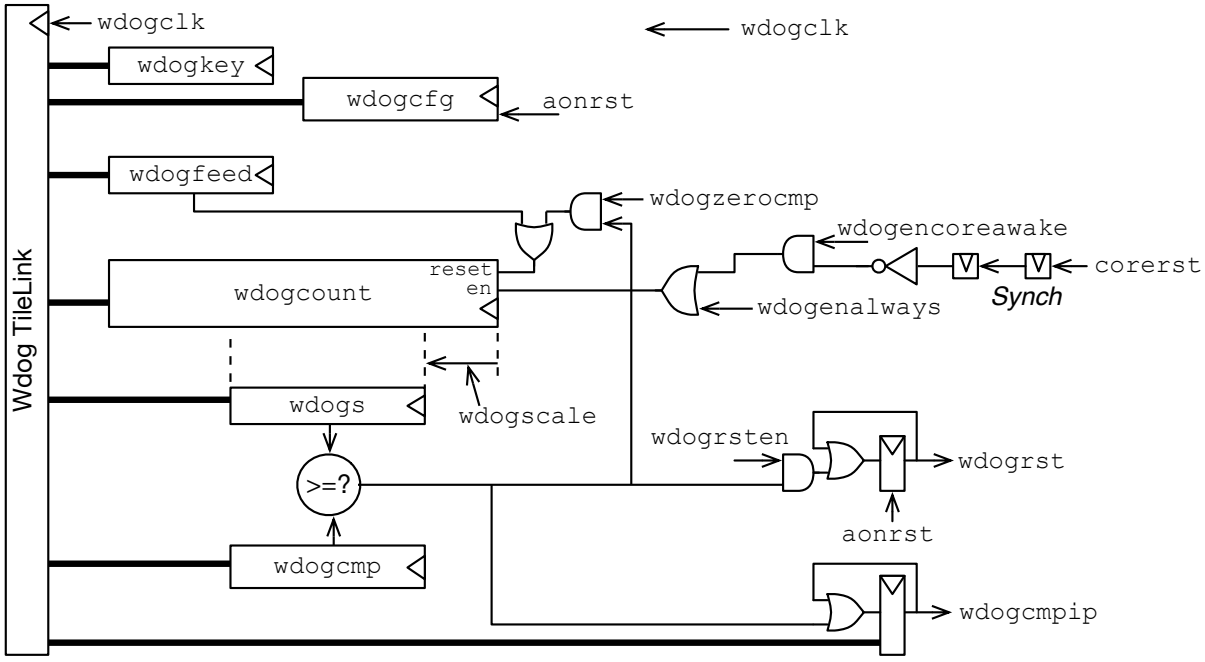

Figure 14.1: E300 Watchdog Timer.

14.1 Watchdog Count Register (`wdogcount`)

The WDT is based around a 31-bit counter held in `wdogcount[30:0]`. The counter can be read or written over the TileLink bus. Bit 31 of `wdogcount` returns a zero when read.

The counter is incremented at a maximum rate determined by the watchdog clock selection. Each cycle, the counter can be conditionally incremented depending on the existence of certain conditions, including always incrementing or incrementing only when the processor is not asleep.

The counter can also be reset to zero depending on certain conditions, such as a successful write to `wdogfeed` or the counter matching the compare value.

14.2 Watchdog Clock Selection

The WDT unit clock, `wdogclk`, is either driven from LFXOSC or LFRCOSC and runs at approximately 32 kHz.

14.3 Watchdog Configuration Register `wdogcfg`

Watchdog Configuration Register (`wdogcfg`)				
Register Offset		`0x000`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[3:0]	wdogscale	RW	X	Watchdog counter scale
[7:4]	*Reserved*	RO	0x0	
8	wdogrsten	RW	0	Watchdog full reset enable
9	wdogzerocmp	RW	X	Watchdog zero on comparator
[11:10]	*Reserved*	RO	0x0	
12	wdogenalways	RW	0	Watchdog enable couter always
13	wdogencoreawake	RW	0	Watchdog counter only when awake
[27:14]	*Reserved*	RO	0x0	
28	wdogcmpip	RW	X	Watchdog interrupt pending
[31:29]	*Reserved*	RO	0x0	

Table 14.1: Watchdog Configuration Register

The `wdogen*` bits control the conditions under which the watchdog counter `wdogcount` is incremented. The `wdogenalways` bit if set means the watchdog counter always increments. The `wdogencoreawake` bit if set means the watchdog counter increments if the processor core is not asleep. The WDT uses the `corerst` signal from the wakeup sequencer to know when the core is sleeping. The counter increments by one each cycle only if any of the enabled conditions are true. The `wdogen*` bits are reset on AON reset.

The 4-bit `wdogscale` field scales the watchdog counter value before feeding it to the comparator. The value in `wdogscale` is the bit position within the `wdogcount` register of the start of a 16-bit `wdogs` field. A value of 0 in `wdogscale` indicates no scaling, and `wdogs` would then be equal to the low 16 bits of `wdogcount`. The maximum value of 15 in `wdogscale` corresponds to dividing the clock rate by 2^{15}, so for an input clock of 32.768 kHz, the LSB of `wdogs` will increment once per second.

The value of `wdogs` is memory-mapped and can be read as a single 16-bit value over the AON TileLink bus.

The `wdogzerocmp` bit, if set, causes the watchdog counter `wdogcount` to be automatically reset to zero one cycle after the `wdogs` counter value matches or exceeds the compare value in `wdogcmp`.

```
li t0, 0x51F15E  # Obtain key.
sw t0, wdogkey   # Unlock kennel.
li t0, 0xD09F00D # Get some food.
sw t0, wdogfeed  # Feed the watchdog.
```

Figure 14.2: Sequence to reinitialize watchdog.

This feature can be used to implement periodic counter interrupts, where the period is independent of interrupt service time.

The wdogrsten bit controls whether the comparator output can set the wdogrst bit and hence cause a full reset.

The wdogcmpip interrupt pending bit can be read or written.

14.4 Watchdog Compare Register (wdogcmp)

Watchdog Compare Register (wdogcmp)				
Register Offset		0x020		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	wdogcmp	RW	X	Watchdog compare value
[31:16]	*Reserved*	RO	0x0	

Table 14.2: Watchdog Compare Register

The compare register is a 16-bit value against which the current wdogs value is compared every cycle. The output of the comparator is asserted whenever the value of wdogs is greater than or equal to wdogcmp.

14.5 Watchdog Key Register (wdogkey)

The wdogkey register has one bit of state. To prevent spurious reset of the WDT, all writes to wdogcfg, wdogfeed, wdogcount, wdogs, wdogcmp and wdogcmpip must be preceded by an unlock operation to the wdogkey register location, which sets wdogkey. The value 0x51F15E must be written to the wdogkey register address to set the state bit before any write access to any other watchdog register. The state bit is reset at AON reset, and after any write to a watchdog register.

Watchdog registers may be read without setting wdogkey.

14.6 Watchdog Feed Address (wdogfeed)

After a successful key unlock, the watchdog can be fed using a write of the value 0xD09F00D to the wdogfeed address, which will reset the wdogcount register to zero. The full watchdog feed sequence is shown in Figure 14.2.

Note there is no state associated with the wdogfeed address. Reads of this address return 0.

14.7 Watchdog Configuration

The WDT provides watchdog intervals of up to over 18 hours (≈65,535 seconds).

14.8 Watchdog Resets

If the watchdog is not fed before the `wdogcount` register exceeds the compare register zero while the WDT is enabled, a reset pulse is sent to the reset circuitry, and the chip will go through a complete power-on sequence.

The WDT will be initalized after a full reset, with the `wdogrsten` and `wdogen*` bits cleared.

14.9 Watchdog Interrupts (`wdogcmpip`)

The WDT can be configured to provide periodic counter interrupts by disabling watchdog resets (`wdogrsten`=0) and enabling auto-zeroing of the count register when the comparator fires (`wdogzerocmp`=1).

The sticky single-bit `wdogcmpip` register captures the comparator output and holds it to provide an interrupt pending signal. The `wdogcmpip` register resides in bit 28 of the `wdogcfg` register, and can be read and written over TileLink to clear down the interrupt.

Chapter 15

Real-Time Clock (RTC)

The real-time clock (RT) is located in the always-on domain, and is clocked by a selectable low-frequency clock source. For best accuracy, the RTC should be driven by an external 32.768 kHz watch crystal oscillator (LFXOSC), but to reduce cost, can be driven by a factory-trimmed on-chip oscillator.

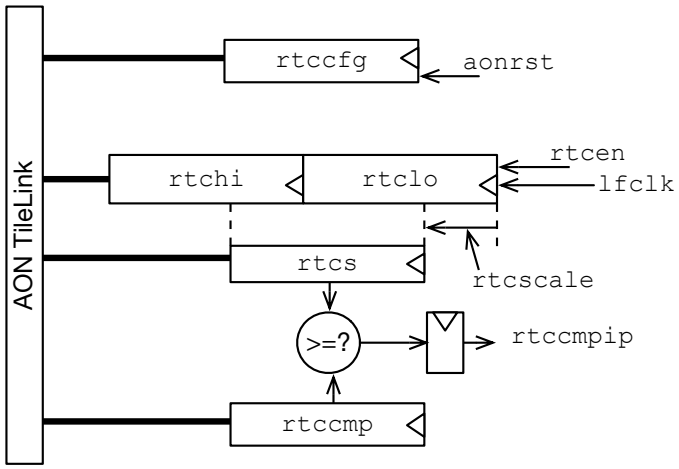

Figure 15.1: Real-Time Clock.

15.1 RTC Count Registers `rtchi`/`rtclo`

The real-time counter is based around the `rtchi`/`rtclo` register pair, which increment at the low-frequency clock rate when the RTC is enabled. The `rtclo` register holds the low 32 bits of the RTC, while `rtchi` holds the upper 16 bits of the RTC value. The total $\geq$48-bit counter width ensures there will no counter rollover for over 270 years assuming a 32.768 kHz low-frequency real-time clock source. The counter registers can be read or written over the TileLink bus.

15.2 RTC Configuration Register `rtccfg`

The `rtcenalways` bit controls whether the RTC is enabled, and is reset on AON reset.

RTC Counter Register High (rtchi)				
Register Offset		0x04C		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	rtchi	RW	X	RTC counter register, high bits
[31:16]	*Reserved*	RO	0x0	

Table 15.1: RTC Counter Register High

RTC Counter Register Low (rtclo)				
Register Offset		0x048		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[31:0]	rtclo	RW	X	RTC counter register, low bits

Table 15.2: RTC Counter Register Low

RTC Configuration Register (rtccfg)				
Register Offset		0x040		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[3:0]	rtcscale	RW	X	RTC clock rate scale
[11:4]	*Reserved*	RO	0x0	
12	rtcenalways	RW	0x0	RTC counter enable
[27:13]	*Reserved*	RO	0x0	
28	rtcmpip	RO	X	RTC comparator interrupt pending
[31:29]	*Reserved*	RO	0x0	

Table 15.3: RTC Configuration Register

The 4-bit rtcscale field scales the real-time counter value before feeding to the real-time interrupt comparator. The value in rtcscale is the bit position within the rtclo/rtchi register pair of the start of a 32-bit field rtcs. A value of 0 in rtcscale indicates no scaling, and rtcs would then be equal to rtclo. The maximum value of 15 in rtcscale corresponds to dividing the clock rate by 2^{15}, so for an input clock of 32.768 kHz, the LSB of rtcs will increment once per second. The value of rtcs is memory-mapped and can be read as a single 32-bit register over the AON TileLink bus.

The rtccmpip interrupt pending bit is read-only.

15.3 RTC Compare Register rtccmp

The rtccmp register holds a 32-bit value that is compared against rtcs, the scaled real-time clock. If rtcs is greater than or equal to rtccmp, the rtccmpip interrupt pending bit is set. The rtccmpip bit can be cleared down by writing a value to rtccmp that is greater than rtcs.

RTC Counter Compare Register (rtccmp)				
Register Offset		0x060		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[31:0]	rtccmp	RW	X	RTC counter comparison value

Table 15.4: RTC Counter Compare Register

Chapter 16

One-Time Programmable Memory (OTP) Peripheral

This chapter describes the operation of the One-Time Programmable Memory (OTP) Controller.

Device configuration and power-supply control is principally under software control. The controller is reset to a state that allows memory-mapped reads, under the assumption that the controller's clock rate is between 1 MHz and 37 MHz. `vrren` is asserted during synchronous reset; it is safe to read from OTP immediately after reset if reset is asserted for at least 150 μs while the controller's clock is running.

Programmed-I/O reads and writes are sequenced entirely by software.

16.1 Memory Map

The memory map for the OTP control registers is shown in Table 16.1. The control-register memory map has been designed to only require naturally aligned 32-bit memory accesses. The OTP controller also contains a read sequencer, which exposes the OTP's contents as a read/execute-only memory-mapped device.

16.2 Programmed-I/O lock register (`otp_lock`)

The `otp_lock` register supports synchronization between the read sequencer and the programmed-I/O interface. When the lock is clear, memory-mapped reads may proceed. When the lock is set, memory-mapped reads do not access the OTP device, and instead return `0` immediately.

The `otp_lock` should be acquired before writing to any other control register. Software can attempt to acquire the lock by storing `1` to `otp_lock`. If a memory-mapped read is in progress, the lock will not be acquired, and will retain the value `0`. Software can check if the lock was successfully acquired by loading `otp_lock` and checking that it has the value `1`.

After a programmed-I/O sequence, software should restore the previous value of any control registers that were modified, then store `0` to `otp_lock`.

Figure 16.1 shows the synchronization code sequence.

OTP Register Offsets		
Offset	**Name**	**Description**
0x00	otp_lock	Programmed-I/O lock register
0x04	otp_ck	OTP device clock signal
0x08	otp_oe	OTP device output-enable signal
0x0c	otp_sel	OTP device chip-select signal
0x10	otp_we	OTP device write-enable signal
0x14	otp_mr	OTP device mode register
0x18	otp_mrr	OTP read-voltage regulator control
0x1c	otp_mpp	OTP write-voltage charge pump control
0x20	otp_vrren	OTP read-voltage enable
0x24	otp_vppen	OTP write-voltage enable
0x28	otp_a	OTP device address
0x2c	otp_d	OTP device data input
0x30	otp_q	OTP device data output
0x34	otp_rsctrl	OTP read sequencer control

Table 16.1: SiFive OTP Register Offsets. Only naturally aligned 32-bit memory accesses are supported.

```
      la t0, otp_lock
      li t1, 1
loop: sw t1, (t0)
      lw t2, (t0)
      beqz t2, loop
      #
      # Programmed I/O sequence goes here.
      #
      sw x0, (t0)
```

Figure 16.1: Sequence to acquire and release `otp_lock`.

16.3 Programmed-I/O Sequencing

The programmed-I/O interface exposes the OTP device's and power-supply's control signals directly to software. Software is responsible for respecting these signals' setup and hold times.

The OTP device requires that data be programmed one bit at a time and that the result be re-read and retried according to a specific protocol.

See the OTP device and power supply data sheets for timing constraints, control signal descriptions, and the programming algorithm.

16.4 Read sequencer control register (`otp_rsctrl`)

The read sequence consists of an address-setup phase, a read-pulse phase, and a read-access phase. The duration of these phases, in terms of controller clock cycles, is set by a programmable clock divider. The divider is controlled by the `otp_rsctrl` register, the layout of which is shown in Table 16.2

The number of clock cycles in each phase is given by 2^{scale}, and the width of each phase may be optionally scaled by 3. That is, the number of controller clock cycles in the address-setup phase is given by the expression $2^{scale}\,(1 + 2t_{AS})$; the number of clock cycles in the read-pulse phase is given by $2^{scale}\,(1 + 2t_{RP})$; and the read-access phase is $2^{scale}\,(1 + 2t_{RACC})$ cycles long. The reset value of `scale` is 1.

Software should acquire the `otp_lock` prior to modifying `otp_rsctrl`.

Read Sequencer Control Register (`otp_rsctrl`)				
Register Offset		`0x34`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[2:0]	scale	RW	0x1	OTP timescale
3	t_AS	RW	0x0	Address setup time
4	t_RP	RW	X	Read pulse time
5	t_RACC	RW	X	Read access time
[31:6]	*Reserved*	RW	X	Reserved

Table 16.2: Read Sequencer Control Register

16.5 OTP Programming Warnings

Warning: Improper use of the One Time Programmable (OTP) memory may result in a nonfunctional device and/or unreliable operation.

- OTP Memory must be programmed following the procedure outlined below *exactly*.
- OTP Memory is designed to be programmed or accessed only while the system clock is running between 1MHz and 37MHz.
- OTP Memory must be programmed **only** while the power supply voltages remain within specification.

16.6 OTP Programming Procedure

1. LOCK the otp:
   (a) Writing 0x1 to `otp_lock`
   (b) **Check that 0x1 is read back from** `otp_lock`.
   (c) Repeat this step until 0x1 is read successfully.
2. SET the programming voltages by writing the following values:

   ```
   otp_mrr=0x4
   otp_mpp=0x0
   otp_vppen=0x0
   ```

3. WAIT 20us for the programming voltages to stabilize
4. ADDRESS the memory by setting `otp_a`
5. WRITE **one bit at a time**:
   (a) set **only** the bit you want to write high in `otp_d`
   (b) Bring `otp_ck` HIGH for 50us
   (c) Bring `otp_ck` LOW.

   Note that this means **only** one bit of `otp_d` should be high at any time.
6. VERIFY the written bits setting `otp_mrr=0x9` for read margin.
7. SOAK any verification failures by repeating steps 2-5 using 400us pulses.
8. REVERIFY the rewritten bits setting `otp_mrr=0xF`. Steps `7,8` may be repeated up to 10 times before failing the part.
9. UNLOCK the otp by writing `0x0` to `otp_lock`.

Chapter 17

General Purpose Input/Output Controller (GPIO)

This chapter describes the operation of the General Purpose Input/Output Controller (GPIO) on the FE310-G000 . The GPIO controller is a peripheral device mapped in the internal memory map. It is responsible for low-level configuration of the actual GPIO pads on the device (direction, pull up-enable, and drive value), as well as selecting between various sources of the controls for these signals. The GPIO controller allows seperate configuration of each of N GPIO bits. Figure 17.1 shows the control structure for each pin.

Atomic operations such as toggles are natively possible with the RISC-V 'A' extension.

17.1 Memory Map

The memory map for the GPIO control registers is shown in Table 17.1. The GPIO memory map has been designed to only require naturally aligned 32-bit memory accesses.

17.2 Input / Output Values

The same `port` register can be configured on a bitwise fashion to represent either inputs or outputs, as set by the `direction` register. Writing to the `port` register will update the bits regardless of the tristate value. Reading the `port` register will return the written value. Reading the `value` register will return the actual value of the pin.

In other words, on a read:

```
value = (input & direction) | (output & ~direction)
```

17.3 Interrupts

A single interrupt bit can be generated for each GPIO bit. The interrupt can be driven by rising or falling edges, or by level values, and each can be enabled individually.

Inputs are synchronized before being sampled by the interrupt logic, so the input pulse width must be long enough to be detected by the synchronization logic.

To enable an interrupt, set the corresponding bit in the `rise_ie` and/or `fall_ie` to 1. If the corresponding bit in `rise_ip` or `fall_ip` is set, an interrupt pin will be raised.

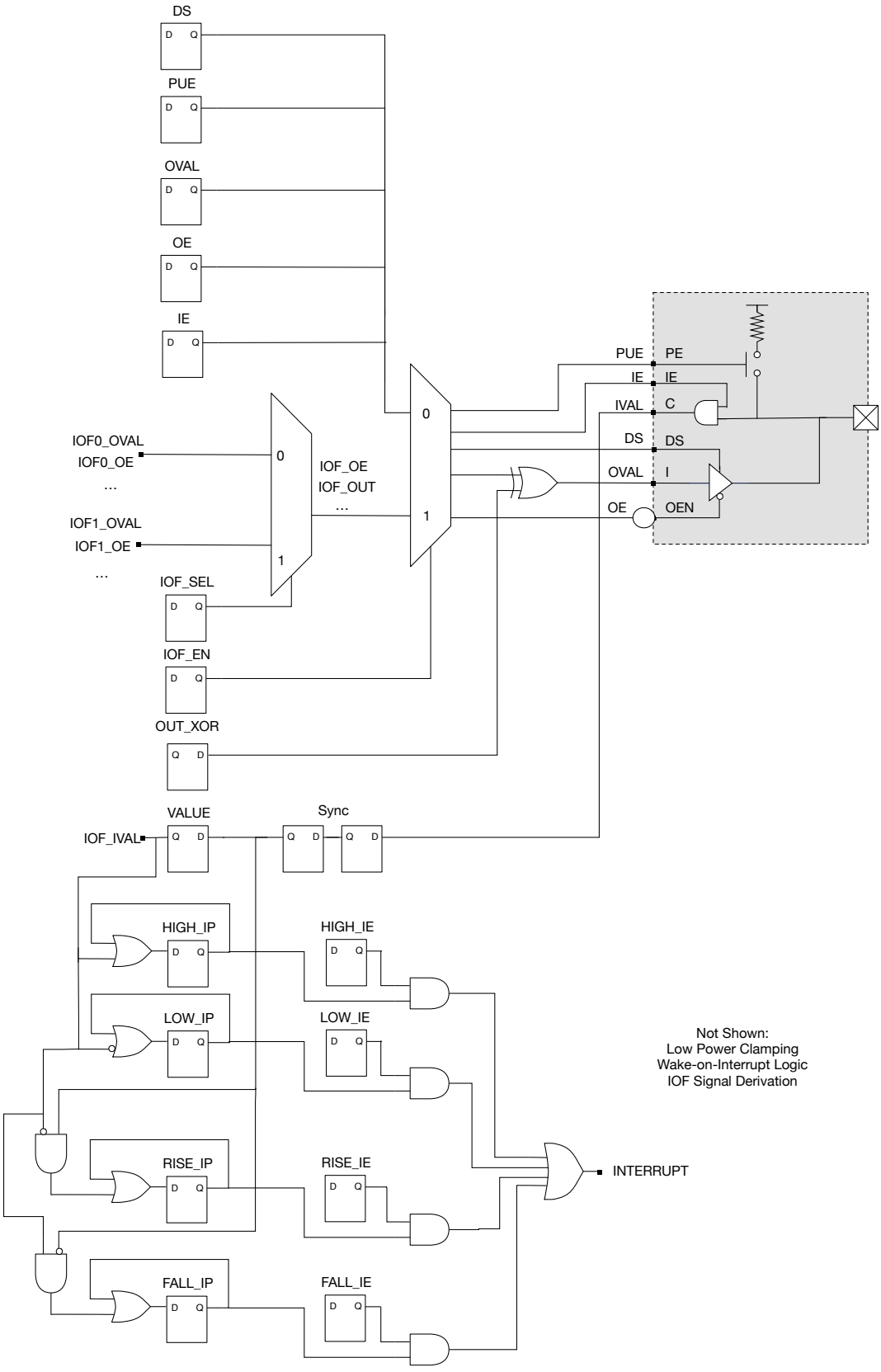

Figure 17.1: Structure of a single GPIO Pin with Control Registers. This structure is repeated for each pin.

GPIO Peripheral Offset Registers		
Offset	**Name**	**Description**
0x000	value	pin value
0x004	input_en	* pin input enable
0x008	output_en	* pin output enable
0x00C	port	output port value
0x010	pue	* internal pull-up enable
0x014	ds	Pin Drive Strength
0x018	rise_ie	rise interrupt enable
0x01C	rise_ip	rise interrupt pending
0x020	fall_ie	fall interrupt enable
0x024	fall_ip	fall interrupt pending
0x028	high_ie	high interrupt enable
0x02C	high_ip	high interrupt pending
0x030	low_ie	low interrupt enable
0x034	low_ip	low interrupt pending
0x038	iof_en	* HW I/O Function enable
0x03C	iof_sel	HW I/O Function select
0x040	out_xor	Output XOR (invert)

Table 17.1: GPIO Peripheral Register Offsets. Only naturally aligned 32-bit memory accesses are supported. Registers marked with an * are asynchronously reset to 0. All other registers are synchronously reset to 0.

Once the interrupt is pending, it will remain set until a 1 is written to the `*_ip` register at that bit.

The interrupt pins may be routed to the PLIC, or directly to local interrupts.

17.4 Internal Pull-Ups

When configured as inputs, each pin has an internal pull-up which can be enabled by software. At reset, all pins are set as inputs and pull-ups are disabled.

17.5 Drive Strength

When configured as output, each pin has a SW-controllable Drive Strength.

17.6 Output Inversion

When configured as an output (either SW or IOF controlled), the SW-writable `out_xor` register is combined with the output to invert it.

17.7 HW I/O Functions (IOF)

Each GPIO pin can implement up to 2 HW-Driven functions (IOF) enabled with the `iof_en` register. Which IOF is used is selected with the `iof_sel` register.

When a pin is set to perform an IOF, it is possible that the software registers `port`, `output_en`, `pullup`, `ds`, `input_en` may not be used to control the pin directly. Rather, the pins may be controlled by hardware driving the IOF. Which functionalities are controlled by the IOF and which are

controlled by the software registers are fixed in the hardware on a per-IOF basis. Those that are not controlled by the hardware continue to be controlled by the software registers.

If there is no IOFx for a pin configured with IOFx, the pin reverts to full software control.

Chapter 18

Serial Peripheral Interface (SPI)

This chapter describes the operation of the SiFive Serial Peripheral Interface (SPI) controller.

18.1 SPI Overview

The SPI controller supports master-only operation over the single-lane, dual-lane, and quad-lane protocols. The baseline controller provides a FIFO-based interface for performing programmed I/O. Software initiates a transfer by enqueuing a frame in the transmit FIFO; when the transfer completes, the slave response is placed in the receive FIFO.

In addition, the dedicated SPI0 controller implements a SPI flash read sequencer, which exposes the external SPI flash contents as a read/execute-only memory-mapped device. The SPI0 controller is reset to a state which allows memory-mapped reads, under the assumption that the input clock rate is less than 100 MHz and the external SPI flash device supports the common Winbond/Numonyx serial read (`0x03`) command. Sequential accesses are automatically combined into one long read command for higher performance.

The `fctrl` register controls switching between the memory-mapped and programmed-I/O modes. While in programmed-I/O mode, memory-mapped reads do not access the external SPI flash device and instead return `0` immediately. Hardware interlocks ensure that the current transfer completes before mode transitions and control register updates take effect.

18.2 Memory Map

The memory map for the SPI control registers is shown in Table 18.1. The SPI memory map has been designed to only require naturally aligned 32-bit memory accesses.

18.3 Serial Clock Divisor Register (`sckdiv`)

The `sckdiv` register specifies the divisor used for generating the serial clock (SCK). The relationship between the input clock and SCK is given by the following formula:

$$f_{\mathsf{sck}} = \frac{f_{\mathsf{in}}}{2(\mathsf{div}+1)}$$

The input clock is the bus clock `tlclk`. The reset value of the `div` field is 0x003.

SPI Register Offsets		
Offset	**Name**	**Description**
0x000	sckdiv	Serial clock divisor
0x004	sckmode	Serial clock mode
0x010	csid	Chip select ID
0x014	csdef	Chip select default
0x018	csmode	Chip select mode
0x028	delay0	Delay control 0
0x02C	delay1	Delay control 1
0x040	fmt	Frame format
0x048	txdata	Tx FIFO data
0x04C	rxdata	Rx FIFO data
0x050	txmark	Tx FIFO watermark
0x054	rxmark	Rx FIFO watermark
0x060	fctrl*	SPI flash interface control
0x064	ffmt*	SPI flash instruction format
0x070	ie	SPI interrupt enable
0x074	ip	SPI interrupt pending

Table 18.1: Register offsets within the SPI memory map. Registers marked * are present only on controllers with the direct-map flash interface, i.e., SPI0.

Serial Clock Divisor Register (sckdiv)				
Register Offset		0x000		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[11:0]	scale	RW	0x003	Divisor for serial clock
[31:12]	*Reserved*	RW	X	

Table 18.2: Serial Clock Divisor Register

18.4 Serial Clock Mode Register (sckmode)

The sckmode register defines the serial clock polarity and phase. Tables 18.4 and 18.5 describe the behavior of the pol and pha fields, respectively. The reset value of sckmode is 0.

Serial Clock Mode Register (sckmode)				
Register Offset		0x004		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	pha	RW	0x0	Serial clock phase
1	pol	RW	0x0	Serial clock polarity
[31:2]	*Reserved*	RW	X	

Table 18.3: Serial Clock Mode Register

Serial Clock Polarity	
Value	**Description**
0	Inactive state of SCK is logical 0
1	Inactive state of SCK is logical 1

Table 18.4: Serial clock polarity.

Serial Clock Phase	
Value	**Description**
0	Data is sampled on the leading edge of SCK and shifted on the trailing edge of SCK
1	Data is shifted on the leading edge of SCK and sampled on the trailing edge of SCK

Table 18.5: Serial clock phase.

18.5 Chip Select ID Register (csid)

The csid register encodes the index of the CS pin to be toggled by hardware chip select control. The reset value is 0.

Chip Select ID Register (csid)				
Register Offset		0x010		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[31:0]	csid	RW	0x0000_0000	Chip Select ID

Table 18.6: Chip Select ID Register

18.6 Chip Select Default Register (csdef)

The csdef register specifies the inactive state (polarity) of the CS pins. The reset value is high for all implemented CS pins.

Chip Select Default Register (csdef)				
Register Offset		0x014		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[31:0]	csdef	RW	0x0000_0001*	Chip Select Default Value

Table 18.7: Chip Select Default Register

18.7 Chip Select Mode Register (csmode)

The csmode register defines the hardware chip select behavior as described in Table 18.9. The reset value is 0 (AUTO). In HOLD mode, the CS pin is de-asserted only when one of the following conditions occur:

- A different value is written to csmode or csid.
- A write to csdef changes the state of the selected pin.
- Direct-mapped flash mode is enabled.

Chip Select Mode Register (csmode)				
Register Offset		0x018		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[1:0]	mode	RW	X	Chip select mode
[31:2]	*Reserved*	RW	X	

Table 18.8: Chip Select Mode Register

Chip Select Modes		
Value	**Name**	**Description**
0	AUTO	Assert/de-assert CS at the beginning/end of each frame
2	HOLD	Keep CS continuously asserted after the initial frame
3	OFF	Disable hardware control of the CS pin

Table 18.9: Chip select modes.

18.8 Delay Control Registers (`delay0` and `delay1`)

The `delay0` and `delay1` registers allow for the insertion of arbitrary delays specified in units of one SCK period.

The `cssck` field specifies the delay between the assertion of CS and the first leading edge of SCK. When `sckmode.pha` $= 0$, an additional half-period delay is implicit. The reset value is `0x01`.

The `sckcs` field specifies the delay between the last trailing edge of SCK and the de-assertion of CS. When `sckmode.pha` $= 1$, an additional half-period delay is implicit. The reset value is `0x01`.

The `intercs` field specifies the minimum CS inactive time between de-assertion and assertion. The reset value is `0x01`.

The `interxfr` field specifies the delay between two consecutive frames without de-asserting CS. This is applicable only when `sckmode` is HOLD or OFF. The reset value is `0x00`.

Delay Control Register 0 (`delay0`)				
Register Offset		`0x028`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	cssck	RW	0x01	CS to SCK Delay
[15:8]	*Reserved*	RW	X	
[23:16]	sckcs	RW	0x01	SCK to CS Delay
[31:24]	*Reserved*	RW	X	

Table 18.10: Delay Control Register 0

Delay Control Register 1 (`delay1`)				
Register Offset		`0x02C`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	intercs	RW	0x01	Minimum CS inactive time
[15:8]	*Reserved*	RW	X	
[23:16]	interxfr	RW	0x00	Maximum interframe delay
[31:24]	*Reserved*	RW	X	

Table 18.11: Delay Control Register 1

18.9 Frame Format Register (`fmt`)

The `fmt` register defines the frame format for transfers initiated through the programmed-I/O (FIFO) interface. Tables 18.13, 18.14, and 18.15 describe the `proto`, `endian`, and `dir` fields, respectively. The `len` field defines the number of bits per frame, where the allowed range is 0 to 8 inclusive.

The reset value is `0x0008_0008`, corresponding to `proto` = single, `dir` = Rx, `endian` = MSB, and `len` $= 8$ for spi0. The reset value is `0x0008_0000`, corresponding to `proto` = single, `dir` = Rx, `endian` = MSB, and `len` $= 8$ for spi1 and spi2.

Frame Format Register (`fmt`)				
Register Offset		`0x040`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[1:0]	proto	RW	0x0	SPI Protocol
2	endian	RW	0x0	SPI endinanness
3	dir	RW	0x1*	SPI I/O Direction
[15:4]	*Reserved*	RW	X	
[19:16]	len	RW	0x8	Number of bits per frame
[31:20]	*Reserved*	RW	X	

Table 18.12: Frame Format Register

SPI Protocol		
Value	**Description**	**Data Pins**
`0`	Single	DQ0 (MOSI), DQ1 (MISO)
`1`	Dual	DQ0, DQ1
`2`	Quad	DQ0, DQ1, DQ2, DQ3

Table 18.13: SPI protocol. Unused DQ pins are tri-stated.

SPI Endianness	
Value	**Description**
`0`	Transmit most-significant bit (MSB) first
`1`	Transmit least-significant bit (LSB) first

Table 18.14: SPI endianness.

SPI I/O Direction	
Value	**Description**
`0`	Rx: For dual and quad protocols, the DQ pins are tri-stated. For the single protocol, the DQ0 pin is driven with the transmit data as normal.
`1`	Tx: The receive FIFO is not populated.

Table 18.15: SPI I/O direction.

18.10 Transmit Data Register (`txdata`)

Writing to the `txdata` register loads the transmit FIFO with the value contained in the `data` field. For `fmt.len` < 8, values should be left-aligned when `fmt.endian` = MSB and right-aligned when `fmt.endian` = LSB.

The `full` flag indicates whether the transmit FIFO is ready to accept new entries; when set, writes to `txdata` are ignored. The `data` field returns `0x00` when read.

Transmit Data Register (txdata)				
Register Offset		0x048		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	data	RW	0x00	Transmit data
[30:8]	*Reserved*	RW	X	
31	full	RO	X	FIFO full flag

Table 18.16: Transmit Data Register

18.11 Receive Data Register (rxdata)

Reading the rxdata register dequeues a frame from the receive FIFO. For fmt.len < 8, values are left-aligned when fmt.endian = MSB and right-aligned when fmt.endian = LSB.

The empty flag indicates whether the receive FIFO contains new entries to be read; when set, the data field does not contain a valid frame. Writes to rxdata are ignored.

Receive Data Register (rxdata)				
Register Offset		0x04C		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	data	RO	X	Received data
[30:8]	*Reserved*	RO	X	
31	empty	RO	X	FIFO empty flag

Table 18.17: Receive Data Register

18.12 Transmit Watermark Register (txmark)

The txmark register specifies the threshold at which the Tx FIFO watermark interrupt triggers. The reset value is 1 for spi0, and 0 for spi1 and spi2.

Transmit Watermark Register (txmark)				
Register Offset		0x050		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[2:0]	txmark	RW	0x1*	Transmit watermark
[31:3]	*Reserved*	RW	X	

Table 18.18: Transmit Watermark Register

18.13 Receive Watermark Register (rxmark)

The rxmark register specifies the threshold at which the Rx FIFO watermark interrupt triggers. The reset value is 0.

18.14 SPI Interrupt Registers (ie and ip)

The ie register controls which SPI interrupts are enabled, and ip is a read-only register indicating the pending interrupt conditions. ie is reset to zero.

Receive Watermark Register (rxmark)				
Register Offset		0x054		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[2:0]	rxmark	RW	0x0	Receive watermark
[31:3]	*Reserved*	RW	X	

Table 18.19: Receive Watermark Register

The txwm condition becomes raised when the number of entries in the transmit FIFO is strictly less than the count specified by the txmark register. The pending bit is cleared when sufficient entries have been enqueued to exceed the watermark.

The rxwm condition becomes raised when the number of entries in the receive FIFO is strictly greater than the count specified by the rxmark register. The pending bit is cleared when sufficient entries have been dequeued to fall below the watermark.

SPI Interrupt Enable Register (ie)				
Register Offset		0x070		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	txwm	RO	0x0	Transmit watermark enable
1	rxwm	RO	0x0	Receive watermark enable
[31:2]	*Reserved*	RW	X	

Table 18.20: SPI Watermark Interrupt Enable Register

SPI Watermark Interrupt Pending Register (ip)				
Register Offset		0x074		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	txwm	RO	X	Transmit watermark pending
1	rxwm	RO	X	Receive watermark pending
[31:2]	*Reserved*	RW	X	

Table 18.21: SPI Watermark Interrupt Pending Register

18.15 SPI Flash Interface Control Register (fctrl)

When the en bit of the fctrl register is set, the controller enters SPI flash mode. Accesses to the direct-mapped memory region causes the controller to automatically sequence SPI flash reads in hardware. The reset value is 0x1.

SPI Flash Interface Control Register (fctrl)				
Register Offset		0x060		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	en	RW	0x1	SPI Flash Mode Select
[31:1]	*Reserved*	RW	X	

Table 18.22: SPI Flash Interface Control Register

18.16 SPI Flash Instruction Format Register (ffmt)

The ffmt register defines the format of the SPI flash read instruction issued by the controller when the direct-mapped memory region is accessed while in SPI flash mode.

An instruction consists of a command byte followed by a variable number of address bytes, dummy cycles (padding), and data bytes. Table 18.23 describes the function and reset value of each field.

SPI Flash Instruction Format Register (ffmt)				
Register Offset		0x064		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	cmd_en	RW	0x1	Enable sending of command
[3:1]	addr_len	RW	0x3	Number of address bytes(0 to 4)
[7:4]	pad_cnt	RW	0x0	Number of dummy cycles
[9:8]	cmd_proto	RW	0x0	Protocol for transmitting command
[11:10]	addr_proto	RW	0x0	Protocol for transmitting address and padding
[13:12]	data_proto	RW	0x0	Protocol for receiving data bytes
[15:14]	*Reserved*	RW	X	
[23:16]	cmd_code	RW	0x03	Value of command byte
[31:24]	pad_code	RW	0x00	First 8 bits to transmit during dummy cycles

Table 18.23: SPI Flash Instruction Format Register.

Chapter 19

Universal Asynchronous Receiver/Transmitter (UART)

This chapter describes the operation of the SiFive Universal Asynchronous Receiver/Transmitter (UART).

19.1 UART Overview

The UART peripheral supports the following features:

- 8-N-1 and 8-N-2 formats: 8 data bits, no parity bit, 1 start bit, 1 or 2 stop bits
- 8-entry transmit and receive FIFO buffers with programmable watermark interrupts
- 16× Rx oversampling with 2/3 majority voting per bit

The UART peripheral does not support hardware flow control or other modem control signals, or synchronous serial data tranfesrs.

19.2 Memory Map

The memory map for the UART control registers is shown in Table 19.1. The UART memory map has been designed to only require naturally aligned 32-bit memory accesses.

UART Register Offsets		
Offset	**Name**	**Description**
0x000	txdata	Transmit data register
0x004	rxdata	Receive data register
0x008	txctrl	Transmit control register
0x00C	rxctrl	Receive control register
0x010	ie	UART interrupt enable
0x014	ip	UART Interrupt pending
0x018	div	Baud rate divisor

Table 19.1: Register offsets within UART memory map.

19.3 Transmit Data Register (txdata)

Writing to the txdata register enqueues the character contained in the data field to the transmit FIFO if the FIFO is able to accept new entries. Reading from txdata returns the current value of the full flag and zero in the data field. The full flag indicates whether the transmit FIFO is able to accept new entries; when set, writes to data are ignored. A RISC-V amoswap instruction can be used to both read the full status and attempt to enqueue data, with a non-zero return value indicating the character was not accepted.

Transmit Data Register (txdata)				
Register Offset		0x000		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	data	RW	X	Transmit data
[30:8]	*Reserved*	RW	X	
31	full	RW	X	Transmit FIFO full

Table 19.2: Transmit Data Register

19.4 Receive Data Register (rxdata)

Reading the rxdata register dequeues a character from the receive FIFO, and returns the value in the data field. The empty flag indicates if the receive FIFO was empty; when set, the data field does not contain a valid character. Writes to rxdata are ignored.

Receive Data Register (rxdata)				
Register Offset		0x004		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	data	RO	X	Received data
[30:8]	*Reserved*	RW	X	
31	empty	RO	X	Receive FIFO empty

Table 19.3: Receive Data Register

19.5 Transmit Control Register (txctrl)

The read-write txctrl register controls the operation of the transmit channel. The txen bit controls whether the Tx channel is active. When cleared, transmission of Tx FIFO contents is suppressed, and the txd pin is driven high.

The nstop field specifies the number of stop bits: 0 for one stop bit and 1 for two stop bits.

The txcnt field specifies the threshold at which the Tx FIFO watermark interrupt triggers.

The txctrl register is reset to 0.

19.6 Receive Control Register (rxctrl)

The read-write rxctrl register controls the operation of the receive channel. The rxen bit controls whether the Rx channel is active. When cleared, the state of the rxd pin is ignored, and no characters will be enqueued into the Rx FIFO.

The rxcnt field specifies the threshold at which the Rx FIFO watermark interrupt triggers.

Transmit Control Register (txctrl)				
Register Offset		0x008		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	txen	RW	0x0	Transmit enable
1	nstop	RW	0x0	Number of stop bits
[15:2]	*Reserved*	RW	X	
[18:16]	txcnt	RW	0x0	Transmit watermark level
[31:19]	*Reserved*	RW	X	

Table 19.4: Transmit Control Register

The rxctrl register is reset to 0.

Receive Control Register (rxctrl)				
Register Offset		0x00C		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	rxen	RW	0x0	Receive enable
[15:1]	*Reserved*	RW	X	
[18:16]	rxcnt	RW	0x0	Receive watermark level
[31:19]	*Reserved*	RW	X	

Table 19.5: Receive Control Register

19.7 Interrupt Registers (ip and ie)

The ip register is a read-only register indicating the pending interrupt conditions, and the read-write ie register controls which UART interrupts are enabled. ie is reset to 0.

The txwm condition becomes raised when the number of entries in the transmit FIFO is strictly less than the count specified by the txcnt field of the txctrl register. The pending bit is cleared when sufficient entries have been enqueued to exceed the watermark.

The rxwm condition becomes raised when the number of entries in the receive FIFO is strictly greater than the count specified by the rxcnt field of the rxctrl register. The pending bit is cleared when sufficient entries have been dequeued to fall below the watermark.

UART Interrupt Enable Register (ie)				
Register Offset		0x010		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	txwm	RW	0x0	Transmit watermark interrupt enable
1	rxwm	RW	0x0	Receive watermark interrupt enable
[31:2]	*Reserved*	RW	X	

Table 19.6: UART Interrupt Enable Register

19.8 Baud Rate Divisor Register (div)

The read-write div register specifies the divisor used by baud rate generation for both Tx and Rx channels. The relationship between the input clock and baud rate is given by the following formula:

UART Interrupt Pending Register (ip)				
Register Offset		0x014		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	txwm	RO	X	Transmit watermark interrupt pending
1	rxwm	RO	X	Receive watermark interrupt pending
[31:2]	*Reserved*	RO	X	

Table 19.7: UART Interrupt Pending Register

$$f_{\text{baud}} = \frac{f_{\text{in}}}{\text{div} + 1}$$

Baud Rate Divisor Register (div)				
Register Offset		0x018		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	div	RW	0xFFFF	Baud rate divisor
[31:16]	*Reserved*	RW	X	

Table 19.8: Baud Rate Divisor Register

The input clock is the bus clock tlclk. Table 19.9 shows divisors for some common core clock rates and commonly used baud rates. Note the table shows the divide ratios, which are one greater than the value stored in the div register.

Common Baud Rates				
tlclk (MHz)	**Target Baud (Hz)**	**Divisor**	**Actual Baud (Hz)**	**Error (%)**
2	31250	64	31250	0
2	115200	17	117647	2.12
16	31250	512	31250	0
16	115200	139	115108	0.08
16	250000	64	250000	0
200	31250	6400	31250	0
200	115200	1736	115207	0.0064
200	250000	800	250000	0
200	1843200	109	1834862	0.45
384	31250	12288	31250	0
384	115200	3333	115212	0.01
384	250000	1536	250000	0
384	1843200	208	1846154	0.16

Table 19.9: Common baud rates (MIDI=31250, DMX=250000) and required divide values to achieve them with given bus clock frequencies. The divide values are one greater than the value stored in the div register.

The receive channel is sampled at 16× the baud rate, and a majority vote over 3 neighboring bits is used to determine the received value. For this reason, the divisor must be ≥16 for a receive channel.

Chapter 20

Pulse-Width Modulation (PWM)

This chapter describes the operation of the Pulse-Width Modulation peripheral (PWM).

20.1 PWM Overview

Figure 20.1 shows an overview of the PWM peripheral. The default configuration described here has four independent PWM comparators (pwmcmp0–pwmcmp3), but custom configurations with ncmp comparators are available on request. The PWM block can generate multiple types of waveform on GPIO output pins (pwm$X$gpio), and can also be used to generate several forms of internal timer interrupt. The comparator results are captured in the pwmcmp$X$ip flops and then fed to the PLIC as potential interrupt sources. The pwmcmp$X$ip outputs are further processed by an output ganging stage before being fed to the GPIOs.

The PWM unit can be provided in different comparator precisions up to 16 bits, with the version described here having the full 16 bits. To support clock scaling, the pwmcount register is 15 bits wider than the comparator precision, cmpwidth.

20.2 PWM Memory Map

The memory map for the PWM peripheral is shown in Table 20.1.

20.3 PWM Count Register (pwmcount)

The PWM unit is based around a counter held in pwmcount. The counter can be read or written over the TileLink bus. The pwmcount register is $(15 + cmpwidth)$ bits wide. For example, for cmpwidth of 16 bits, the counter is held in pwmcount[30:0], and bit 31 of pwmcount returns a zero when read.

When used for PWM generation, the counter is normally incremented at a fixed rate then reset to zero at the end of every PWM cycle. The PWM counter is either reset when the scaled counter pwms reaches the value in pwmcmp0, or is simply allowed to wraparound to zero.

The counter can also be used in one-shot mode, where it disables counting after the first reset.

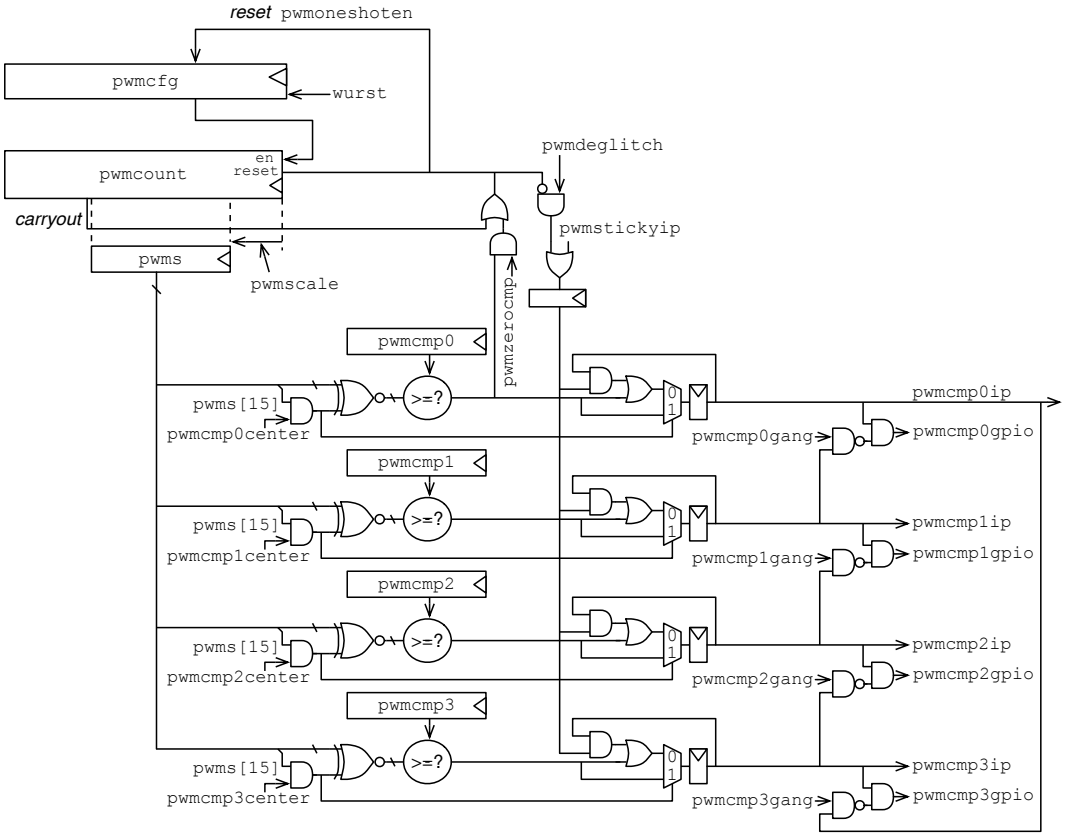

Figure 20.1: PWM Peripheral.

20.4 PWM Configuration Register (pwmcfg)

The pwmcfg register contains various control and status information regarding the PWM peripheral, as shown in Table 20.2.

The pwmen* bits control the conditions under which the PWM counter pwmcount is incremented. The counter increments by one each cycle only if any of the enabled conditions are true.

If the pwmenalways bit is set, the PWM counter increments continuously. When pwmenoneshot is set, the counter can increment but pwmenoneshot is reset to zero once the counter resets, disabling further counting (unless pwmenalways is set). The pwmenoneshot bit provides a way for software to generate a single PWM cycle then stop. Software can set the pwnenoneshot again at any time to replay the one-shot waveform. The pwmen* bits are reset at wakeup reset, which disables the PWM counter and saves power.

The 4-bit pwmscale field scales the PWM counter value before feeding it to the PWM comparators. The value in pwmscale is the bit position within the pwmcount register of the start of a cmpwidth-bit pwms field. A value of 0 in pwmscale indicates no scaling, and pwms would then be equal to the low cmpwidth bits of pwmcount. The maximum value of 15 in pwmscale corresponds to dividing the clock rate by 2^{15}, so for an input bus clock of 16 MHz, the LSB of pwms will increment at 488.3 Hz.

PWM Register Offsets	
Offset	**Description**
0x00	pwmcfg
0x04	*Reserved*
0x08	pwmcount
0x0C	*Reserved*
0x10	pwms
0x14	*Reserved*
0x18	*Reserved*
0x1C	*Reserved*
0x20	pwmcmp0
0x24	pwmcmp1
0x28	pwmcmp2
0x2C	pwmcmp3

Table 20.1: SiFive PWM memory map, offsets relative to PWM peripheral base address.

The value of pwms is memory-mapped and can be read as a single cmpwidth-bit value over the TileLink bus.

The pwmzerocmp bit, if set, causes the PWM counter pwmcount to be automatically reset to zero one cycle after the pwms counter value matches the compare value in pwmcmp0. This is normally used to set the period of the PWM cycle. This feature can also be used to implement periodic counter interrupts, where the period is independent of interrupt service time.

20.5 PWM Compare Registers (pwmcmp0–pwmcmp3)

The primary use of the ncmp PWM compare registers is to define the edges of the PWM waveforms within the PWM cycle.

Each compare register is a cmpwdith-bit value against which the current pwms value is compared every cycle. The output of each comparator is high whenever the value of pwms is greater than or equal to the corresponding pwmcmp$X$.

If the pwmzerocomp bit is set, when pwms reaches or exceeds pwmcmp0, pwmcount is cleared to zero and the current PWM cycle is completed. Otherwise, the counter is allowed to wrap around.

20.6 Deglitch and Sticky circuitry

To avoid glitches in the PWM waveforms when changing pwmcmp$X$ register values, the pwmdeglitch bit in pwmcfg can be set to capture any high output of a PWM comparator in a sticky bit (pwmcmp$X$ip for comparator X) and prevent the output falling again within the same PWM cycle. The pwmcmp$X$ip bits are only allowed to change at the start of the next PWM cycle.

Note the pwmcmp0ip *bit will only be high for one cycle when* pwmdeglitch *and* pwmzerocmp *are set where* pwmcmp0 *is used to define the PWM cycle, but can be used as a regular PWM edge otherwise.*

If pwmdeglitch is set, but pwmzerocmp is clear, the deglitch circuit is still operational but is now triggered when pwms contains all 1s and will cause a carry out of the high bit of the pwms incrementer just before the counter wraps to zero.

PWM Configuration Register (pwmcfg)				
Register Offset		0x00		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[3:0]	pwmscale	RW	X	PWM Counter scale
[7:4]	*Reserved*	RO	0x0	
8	pwmsticky	RW	X	PWM Sticky - disallow clearing pwmcmpXip bits
9	pwmzerocmp	RW	X	PWM Zero - counter resets to zero after match
10	pwmdeglitch	RW	X	PWM Deglitch - latch pwmcmpXip within same cycle
11	*Reserved*	RO	0x0	
12	pwmenalways	RW	0	PWM enable always - run continuously
13	pwmenoneshot	RW	0	PWM enable one shot - run one cycle
[15:14]	*Reserved*	RO	0x0	
16	pwmcmp0center	RW	X	PWM0 Compare Center
17	pwmcmp1center	RW	X	PWM1 Compare Center
18	pwmcmp2center	RW	X	PWM2 Compare Center
19	pwmcmp3center	RW	X	PWM3 Compare Center
[23:0]	*Reserved*	RO	0x0	
24	pwmcmp0gang	RW	X	PWM0/PWM1 Compare Gang
25	pwmcmp1gang	RW	X	PWM1/PWM2 Compare Gang
26	pwmcmp2gang	RW	X	PWM2/PWM3 Compare Gang
27	pwmcmp3gang	RW	X	PWM3/PWM0 Compare Gang
28	pwmcmp0ip	RW	X	PWM0 Interrupt Pending
29	pwmcmp1ip	RW	X	PWM1 Interrupt Pending
30	pwmcmp2ip	RW	X	PWM2 Interrupt Pending
31	pwmcmp3ip	RW	X	PWM3 Interrupt Pending

Table 20.2: PWM Configuration Register

PWM0 Compare Register (pwmcmp0)				
Register Offset		0x20		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	pwmcmp0	RW	X	PWM 0 Compare Value
[31:16]	*Reserved*	RO	0x0	

Table 20.3: PWM0 Compare Register. This diagram assumes a `cmpwidth` of 16. The actual width each register is `cmpwidth`.

The `pwmsticky` bit will disallow the `pwmcmp`X`ip` registers from clearing if they're already set, and is used to ensure interrupts are seen from the `pwmcmp`X`ip` bits.

20.7 Generating Left- or Right-Aligned PWM Waveforms

Figure 20.2 shows the generation of various base PWM waveforms. The Figure illustrates that if `pwmcmp0` is set to less than the maximum count value (6 in this case), it is possible to generate both 100% (`pwmcmp`$X = 0$) and 0% (`pwmcmp`$X >$ `pwmcmp0`) right-aligned duty cycles using the other

PWM1 Compare Register (pwmcmp1)				
Register Offset		0x24		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	pwmcmp0	RW	X	PWM1 Compare Value
[31:16]	*Reserved*	RO	0x0	

Table 20.4: PWM1 Compare Register. This diagram assumes a cmpwidth of 16. The actual width each register is cmpwidth.

PWM2 Compare Register (pwmcmp2)				
Register Offset		0x28		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	pwmcmp0	RW	X	PWM2 Compare Value
[31:16]	*Reserved*	RO	0x0	

Table 20.5: PWM2 Compare Register. This diagram assumes a cmpwidth of 16. The actual width each register is cmpwidth.

PWM3 Compare Register (pwmcmp3)				
Register Offset		0x2C		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	pwmcmp0	RW	X	PWM3 Compare Value
[31:16]	*Reserved*	RO	0x0	

Table 20.6: PWM3 Compare Register. This diagram assumes a cmpwidth of 16. The actual width each register is cmpwidth.

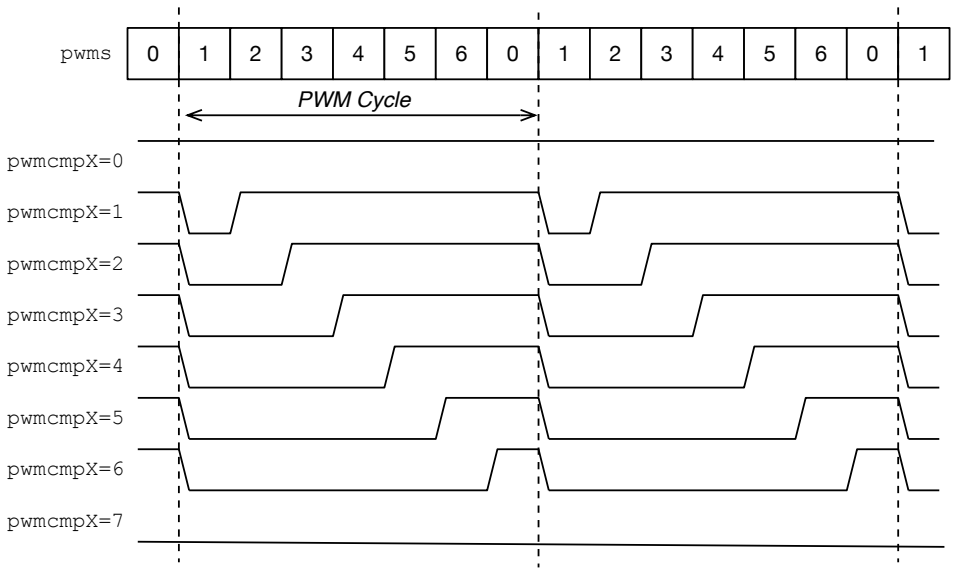

Figure 20.2: Basic right-aligned PWM waveforms. All possible base waveforms are shown for a 7-clock PWM cycle (pwmcmp0=6). The waveforms show the single cycle delay caused by registering the comparator outputs in the pwmcmp$X$ip bits. The signals can be inverted at the GPIOs to generate left-aligned waveforms.

comparators. The pwmcmp$X$ip bits are routed to the GPIO pads, where they can be optionally and individually inverted thereby creating left-aligned PWM waveforms (high at beginning of cycle).

pwmcmpXcenter Inversion	
pwms	**pwmscenter**
000	000
001	001
010	010
011	011
100	011
101	010
110	001
111	000

Figure 20.3: Illustration of how count value is inverted before presentation to comparator when pwmcmp$X$center is selected, using a 3-bit pwms value.

20.8 Generating Center-Aligned (Phase-Correct) PWM Waveforms

The simple PWM waveforms above shift the phase of the waveform along with the duty cycle. A per-comparator pwmcmp$X$center bit in pwmcfg allows a single PWM comparator to generate a center-aligned symmetric duty-cycle as shown in Figure 20.4 The pwmcmp$X$center bit changes the comparator to compare with the bitwise inverted pwms value whenever the MSB of pwms is high.

This technique provides symmetric PWM waveforms but only when the PWM cycle is at the largest supported size. At a 16 MHz bus clock rate with 16-bit precision, this limits the fastest PWM cycle to 244 Hz, or 62.5 kHz with 8-bit precision. Higher bus clock rates allow proportionally faster PWM cycles using the single comparator center-aligned waveforms. This technique also reduces the effective width resolution by a factor of 2.

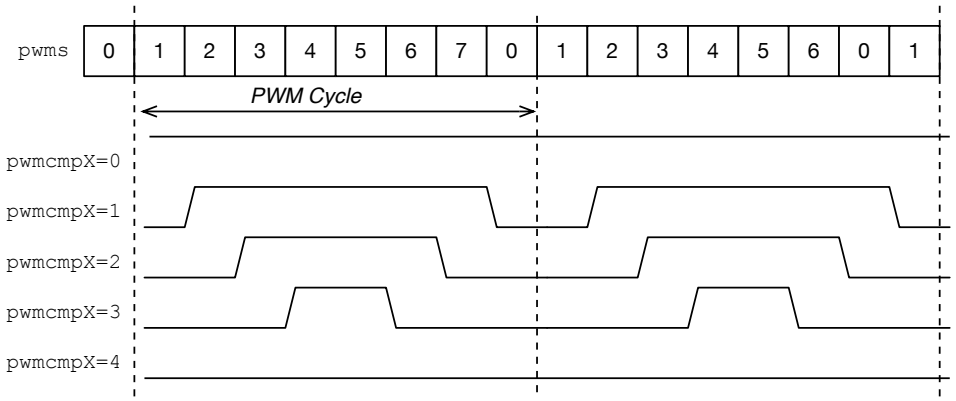

Figure 20.4: Center-aligned PWM waveforms generated from one comparator. All possible waveforms are shown for a 3-bit PWM precision. The signals can be inverted at the GPIOs to generate opposite-phase waveforms.

When a comparator is operating in center mode, the deglitch circuit allows one 0-1 transition during the first half of the cycle, and one 1-0 transition on the second half of the cycle.

20.9 Generating Arbitrary PWM Waveforms using Ganging

A comparator can be ganged together with its next-highest-numbered neighbor to generate arbitrary PWM pulses. When the pwmcmp$X$gang bit is set, comparator X fires and raises its pwm$X$gpio signal. When comparator $X + 1$ (or pwmcmp0 for pwmcmp3) fires, the pwm$X$gpio output is reset to zero.

20.10 Generating One-shot Waveforms

The PWM peripheral can be used to generate precisely timed one-shot pulses by first initializing the other parts of pwmcfg then writing a 1 to the pwmenoneshot bit. The counter will run for one PWM cycle, then once a reset condition occurs, the pwmenoneshot bit is reset in hardware to prevent a second cycle.

20.11 PWM Interrupts

The PWM can be configured to provide periodic counter interrupts by enabling auto-zeroing of the count register when a comparator 0 fires (pwmzerocmp=1). The pwmsticky bit should also be set to ensure interrupts are not forgotten while waiting to run a handler.

The interrupt pending bits pwmcmp$X$ip can be cleared down using writes to the pwmcfg register.

The PWM peripheral can also be used as a regular timer with no counter reset (pwmzerocmp=0), where the comparators are now used to provide timer interrupts.

Chapter 21

Debug

This chapter describes the operation of the FE310-G000 debug hardware, which follows the RISC-V debug spec, v11 [3]. Interactive debug and hardware breakpoints are supported.

21.1 Debug CSRs

This section describes the per-hart trace and debug registers (TDRs), which are mapped into the CSR space as follows:

CSR Name	Description	Allowed Access Modes
`tselect`	Trace and debug register select	D, M
`tdata1`	First field of selected TDR	D, M
`tdata2`	Second field of selected TDR	D, M
`tdata3`	Third field of selected TDR	D, M
`dcsr`	Debug control and status register	D
`dpc`	Debug PC	D
`dscratch`	Debug scratch register	D

The `dcsr`, `dpc`, and `dscratch` registers are only accessible in debug mode, while the `tselect` and `tdata1–3` registers are accessible from either debug mode or machine mode.

21.1.1 Trace and Debug Register Select (`tselect`)

To support a large and variable number of TDRs for tracing and breakpoints, they are accessed through one level of indirection where the `tselect` register selects which bank of three `tdata1–3` registers are accessed via the other three addresses.

The `tselect` register has the format shown below:

The `index` field is a **WARL** field that will not hold indices of unimplemented TDRs. Even if `index` can hold a TDR index, it does not guarantee the TDR exists. The `type` field of `tdata1` must be inspected to determine whether the TDR exists.

Trace and Debug Select Register			
CSR	tselect		
Bits	**Field Name**	**Attr.**	**Description**
[31:0]	index	WARL	Selection index of trace and debug registers

Table 21.1: E31 tselect CSR.

21.1.2 Test and Debug Data Registers (tdata1–3)

The tdata1–3 registers are XLEN-bit read/write registers selected from a larger underlying bank of TDR registers by the tselect register.

Trace and Debug Data Register 1			
CSR	tdata1		
Bits	**Field Name**	**Attr.**	**Description**
[27:0]	TDR-Specific Data		
[31:28]	type	RO	Type of the trace & debug register selected by tselect

Table 21.2: E31 tdata1 CSR.

Trace and Debug Data Registers 2 and 3			
CSR	tdata2/3		
Bits	**Field Name**	**Attr.**	**Description**
[31:0]	type		TDR-Specific Data

Table 21.3: E31 tdata2/3 CSRs.

The high nibble of tdata1 contains a 4-bit type code that is used to identify the type of TDR selected by tselect. The currently defined types are shown below:

type	Description
0	No such TDR register
1	Reserved
2	Address/Data Match Trigger
$\geq$3	Reserved

The dmode bit selects between debug mode (dmode=1) and machine mode (dmode=1) views of the registers, where only debug mode code can access the debug mode view of the TDRs. Any attempt to read/write the tdata1–3 registers in machine mode when dmode=1 raises an illegal instruction exception.

21.1.3 Debug Control and Status Register dcsr

This register gives information about debug capabilities and status. Its detailed functionality is described in the RISC-V Debug Specification 0p11.

21.1.4 Debug PC dpc

When entering Debug Mode, the current PC is copied here. When leaving debug mode, execution resumes at this PC.

21.1.5 Debug Scratch `dscratch`

This register is generally reserved for use by Debug ROM in order to save registers needed by the code in Debug ROM. The debugger may use it as described in the RISC-V Debug Specification 0p11.

21.2 Breakpoints

The FE310-G000 supports 2 hardware breakpoint registers, which can be flexibly shared between debug mode and machine mode.

When a breakpoint register is selected with `tselect`, the other CSRs access the following information for the selected breakpoint:

TDR CSRs when used as Breakpoints		
CSR Name	**Breakpoint Alias**	**Description**
`tselect`	`tselect`	Breakpoint selection index
`tdata1`	`mcontrol`	Breakpoint Match control
`tdata2`	`maddress`	Breakpoint Match address
`tdata3`	N/A	*Reserved*

21.2.1 Breakpoint Match Control Register `mcontrol`

Each breakpoint control register is a read/write register laid out as follows:

Breakpoint Control Register (`mcontrol`)				
Register Offset		`CSR`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	R	WARL	X	Address match on LOAD
1	W	WARL	X	Address match on STORE
2	X	WARL	X	Address match on Instruction FETCH
3	U	WARL	X	Address match on User Mode
4	S	WARL	X	Address match on Supervisor Mode
5	H	WARL	X	Address match on Hypervisor Mode
6	M	WARL	X	Address match on Machine Mode
[10:7]	match	WARL	X	Breakpoint Address Match type
11	chain	WARL	0	Chain adjacent conditions.
[17:12]	action	WARL	0	Breakpoint action to take. 0 or 1.
18	timing	WARL	0	Timing of the breakpoint. Always 0.
19	select	WARL	0	Perform match on address or data. Always 0.
20	*Reserved*	WPRI	X	Reserved
[26:21]	maskmax	RO	4	Largest supported NAPOT range
27	dmode	RW	0	Debug-Only access mode
[31:28]	type	RO	2	Address/Data match type, always 2

Table 21.4: Test and Debug Data Register 3

The `type` field is a four-bit read-only field holding the value 2 to indicate this is a breakpoint containing address match logic.

The `bpaction` field is an eight-bit read-write **WARL** field that specifies the available actions when the address match is successful. The value 0 generates a breakpoint exception. The value 1 enters debug mode. Other actions are not implemented.

The R/W/X bits are individual **WARL** fields and if set, indicate an address match should only be successful for loads/stores/instruction fetches respectively, and all combinations of implemented bits must be supported.

The M/H/S/U bits are individual **WARL** fields and if set, indicate that an address match should only be successful in the machine/hypervisor/supervisor/user modes respectively, and all combinations of implemented bits must be supported.

The `match` field is a 4-bit read-write **WARL** field that encodes the type of address range for breakpoint address matching. Three different `match` settings are currently supported: exact, NAPOT, and arbitrary range. A single breakpoint register supports both exact address matches and matches with address ranges that are naturally aligned powers-of-two (NAPOT) in size. Breakpoint registers can be paired to specify arbitrary exact ranges, with the lower-numbered breakpoint register giving the byte address at the bottom of the range and the higher-numbered breakpoint register giving the address one byte above the breakpoint range, and using the `chain` bit to indicate both must match for the action to be taken.

NAPOT ranges make use of low-order bits of the associated breakpoint address register to encode the size of the range as follows:

NAPOT Size Encoding	
maddress	**Match type and size**
`a...aaaaaa`	Exact 1 byte
`a...aaaaa0`	2-byte NAPOT range
`a...aaaa01`	4-byte NAPOT range
`a...aaa011`	8-byte NAPOT range
`a...aa0111`	16-byte NAPOT range
`a...a01111`	32-byte NAPOT range
`...`	...
`a01...1111`	2^{31}-byte NAPOT range

The `maskmax` field is a 6-bit read-only field that specifies the largest supported NAPOT range. The value is the logarithm base 2 of the number of bytes in the largest supported NAPOT range. A value of 0 indicates that only exact address matches are supported (one byte range). A value of 31 corresponds to the maximum NAPOT range, which is 2^{31} bytes in size. The largest range is encoded in `maddress` with the 30 least-signicant bits set to 1, bit 30 set to 0, and bit 31 holding the only address bit considered in the address comparison.

The unary encoding of NAPOT ranges was chosen to reduce the hardware cost of storing and generating the corresponding address mask value.

To provide breakpoints on an exact range, two neighboring breakpoints can be combined with the `chain` bit. The first breakpoint can be set to match on an address using `action` of 2 (greater than or equal). The second breakpoint can be set to match on address using `action of 3` (less than). Setting then `chain` bit on the first breakpoint will then cause it prevent the second breakpoint from firing unless they both match.

21.2.2 Breakpoint Match Address Register (maddress)

Each breakpoint match address register is an XLEN-bit read/write register used to hold significant address bits for address matching, and also the unary-encoded address masking information for NAPOT ranges.

21.2.3 Breakpoint Execution

Breakpoint traps are taken precisely. Implementations that emulate misaligned accesses in software will generate a breakpoint trap when either half of the emulated access falls within the address range. Implementations that support misaligned accesses in hardware must trap if any byte of an access falls within the matching range.

Debug-mode breakpoint traps jump to the debug trap vector without altering machine-mode registers.

Machine-mode breakpoint traps jump to the exception vector with "Breakpoint" set in the mcause register, and with badaddr holding the instruction or data address that caused the trap.

21.2.4 Sharing breakpoints between debug and machine mode

When debug mode uses a breakpoint register, it is no longer visible to machine-mode (i.e., the tdrtype will be 0). Usually, the debugger will grab the breakpoints it needs before entering machine mode, so machine mode will operate with the remaining breakpoint registers.

21.3 Debug Memory Map

This section describes the debug module's memory map when accessed via the regular system interconnect. The debug module is only accessible to debug code running in debug mode on a hart (or via a debug transport module).

21.3.1 Component Signal Registers (0x100–0x1FF)

The 8-bit address space from 0x100–0x1FF is used to access per-component single-bit registers. This region only supports 32-bit writes.

On a 32-bit write to this region, the 32-bit data value selects a component, bits 7–3 of the address select one out of 32 per-component single-bit registers, and bit 2 is the value to be written to that single-bit register, as shown below.

Component Signal Address Register (csra)				
Register Offset				
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[1:0]	00	RO	0x0	
2	value	RW	X	Value to be written
[7:3]	register	RW	X	Register to be written
[31:8]	0x000001	RW	0x000001	

Table 21.5: Component Signal Address Register

Component Signal Data Register (csrd)				
Register Offset				
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[31:0]	component	RW	X	Component select

Table 21.6: Component Signal Data Register

This addressing scheme was adopted so that RISC-V debug ROM routines can signal that a hart has stopped using a single store instruction to an absolute address (offset from register x0) and one free data register, which holds the hart ID.

The set of valid component identifiers is defined by each implementation.

There are only two per-component registers specified so far, the debug interrupt signal (register 0) and the halt notification register (register 1), resulting in the following four possible write actions.

Possible Write Actions	
Address Written	**Action**
0x100	Clear debug interrupt signal going to component
0x104	Set debug interrupt signal going to component
0x108	Clear halt notification from component
0x10C	Set halt notification from component

21.3.2 Debug RAM (0x400–0x43f)

SiFive systems provide at least the minimal required amount of Debug RAM, which is 28 bytes for an RV32 system and 64 bytes for an RV64 system.

21.3.3 Debug ROM (0x800–0xFFF)

This ROM region holds the debug routines on SiFive systems. The actual total size may vary between implementations.

Chapter 22

Debug Interface

The SiFive FE310-G000 includes the JTAG Debug Transport Module described in the RISC-V Debug Specification v0p11. This enables a single external industry-standard 1149.1 JTAG interface to test and debug the system. The JTAG interface is directly connected to input pins.

22.1 JTAG TAPC State Machine

The JTAG controller includes the standard TAPC state machine shown in Figure 22.1.

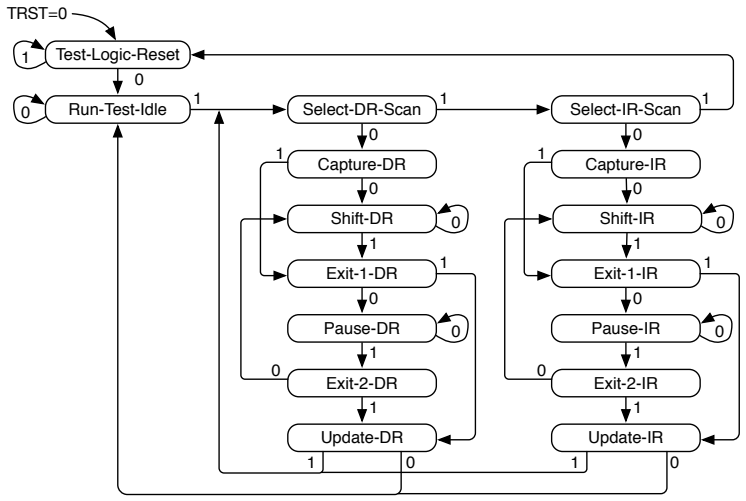

Figure 22.1: JTAG TAPC state machine. The state machine is clocked with TCK. All transitions are labelled with the value on TMS, except for the arc showing asynchronous reset when TRST=0.

22.1.1 JTAG Clocking

The JTAG logic always operates in its own clock domain clocked by TCK. The JTAG logic is fully static and has no minimum clock frequency. The maximum TCK frequency is part-specific.

22.1.2 JTAG Standard Instructions

The JTAG DTM implements the BYPASS and IDCODE instructions. On the FE310-G000 the IDCODE is set to `0x10E31913`.

22.2 JTAG Debug Commands

The JTAG DEBUG instruction gives access to the SiFive debug module by connecting the debug scan register inbetween TDI and TDO.

The debug scan register includes a 2-bit opcode field, a 5-bit debug module address field, and a 34-bit data field to allow various memory-mapped read/write operations to be specified with a single scan of the debug scan register.

These are described in the RISC-V Debug Specification v0p11.

Chapter 23

References

Visit the SiFive forums for support and answers to frequently asked questions: `http://forums.sifive.com`.

[1] A. Waterman and K. Asanović, Eds., *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.2*, May 2017. [Online]. Available: https://riscv.org/specifications/

[2] ——, *The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.10*, May 2017. [Online]. Available: https://riscv.org/specifications/

[3] T. Newsome, Ed., *RISC-V External Debug Support 0.11*, November 2016. [Online]. Available: https://www.sifive.com/documentation/risc-v/risc-v-external-debug-support-0-11/