



SiFive VIC_E24 User Guide

© SiFive, Inc.

SiFive VIC_E24 User Guide

Proprietary Notice

Copyright © 2019, SiFive Inc. All rights reserved.

Information in this document is provided “as is,” with all faults.

SiFive expressly disclaims all warranties, representations, and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose and non-infringement.

SiFive does not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

SiFive reserves the right to make changes without further notice to any products herein.

Contents

1	Introduction	3
1.1	About this Document	3
1.2	About this Release	3
2	Deliverables	4
2.1	Folder Structure	4
3	Memories	6
3.1	RAM Instances	6
4	VIC_E24 Interfaces	8
4.1	Clock & Reset	8
4.1.1	Real Time Clock (rtc_toggle)	8
4.2	Ports	9
4.2.1	System Port	9
4.3	VIC_E24 Interrupt Interfaces	9
4.3.1	Machine External Interrupts	9
4.3.2	Local External Interrupts	9
4.4	Debug Output Signals	10
4.5	JTAG Debug Interface Pinout	10
5	VIC_E24 Error Handling	11
5.1	2 Series Error Handling	11
6	TileLink to AHB Bridge (TL2AHB)	12
6.1	Introduction	12
6.2	Compliance	12
6.3	Block Diagram	13
6.4	TL2AHB Interface	14

	2
6.5	Functional Description 14
6.5.1	Atomic Memory Operations (AMO) 14
6.5.2	Bursts 14
6.5.3	TL2AHB System Integration 15
7	Debug Interface 18
7.1	JTAG TAPC State Machine 19
7.2	Resetting JTAG Logic..... 19
7.3	JTAG Clocking..... 20
7.4	JTAG Standard Instructions 20
7.5	JTAG Debug Commands 20
7.6	Using Debug Outputs 20
8	Implementation 21
8.1	Top Level 21
8.2	Clocking..... 21
8.2.1	Clocking Guidelines..... 22
8.3	Retiming 22
8.4	Gate Level Simulation 22
9	Simulation Testbench 24
9.1	Included Test Bench..... 24
9.1.1	Executing the Testbench 24
9.2	Testbench Output..... 25
9.2.1	Testbench Output - Trace 25
9.2.2	Testbench Output - Waves 25
9.2.3	Adding Tests To The Included Testbench 26
9.3	SiFive Insight..... 26
9.3.1	Viewing SiFive Insight Signals 27
9.3.2	Enabling SiFive Insight Outside of the SiFive Testbench 27

Chapter 1

Introduction

1.1 About this Document

This document describes the VIC_E24 production deliverables. To learn more about the functionality of the VIC_E24 please read the VIC_E24 Manual.

1.2 About this Release

This is the Production release of the VIC_E24.

Chapter 2

Deliverables

This chapter describes the contents of the deliverables for the VIC_E24 .

2.1 Folder Structure

The folder structure of the delivery is as follows:

arty_a7_100t-sifive

Contains the designs FPGA bitstream.

freedom-e-sdk

Software SDK for the design including Freedom Metal BSP and applications.

bsp

Freedom Metal BSP for the RTL testbench and, where applicable, the FPGA bitstream. Note that Freedom Metal BSPs also include a design's Device Tree file (DTS).

freedom-devicetree-tools

Tools used to generate Freedom Metal BSPs from DTS files. Can be used to re-generate BSPs in the case of a hand-edited DTS file.

freedom-metal

Source code for the Freedom Metal library.

scripts

Helper scripts used by the main makefile.

software

One folder for each included Freedom Metal application. Each application includes a pre-built hex file in its *release* directory which can be run directly on the testbench without needed to re-compile from source. Note: In some cases there will not be a pre-built hex file for every application. This will be the case when a particular application is not expected to run correctly on the selected core configuration. Possible reasons for this are:

- Executable image does not fit in configured boot memory
- Application data does not fit in configured data memory
- Multicore application excluded for single-core configuration

Makefile

Top-level SDK makefile which can be used to re-build all included examples from source. `Readme.md`; `Readme` file describing how to use the SDK's top-level makefile.

info

Files which describe the design.

mems.conf

Configuration file which describes the memory instances of the design.

modules_to_be_retimed.txt

Contains the list of modules which need to be retimed.

sifive_insight.yml

Contains a `.yml` description of the SiFive Insight signals included in the design. See Section 9.3 for more details on SiFive Insight.

rtl

The `VIC_E24` RTL.

memories

A single verilog file containing all memories in the design.

testbench

Includes all the modules in the synthesizable testbench, the test driver, and extracted simulation constructs (assertions) that are bound to locations in the DUT.

design

The `VIC_E24` itself. Includes the top-level module `E2_CoreIPSubsystem` and all sub-modules.

sifive_insight

Contains all the System Verilog files defining and binding the SiFive Insight signals to modules in the design.

.F files

Manifest files for the associated folder. A complete list of files to be synthesized as part of the design can be found in `design.F`.

Makefile

Used to execute the test bench described in Chapter 9.

Chapter 3

Memories

This chapter describes the memories used in the VIC_E24 design.

3.1 RAM Instances

The Core Complex RAM instances consist of synchronous single-ported SRAMs. These are contained within wrapper modules that expose a standardized generic interface for the VIC_E24. For each of the modules specified in Table 1, the implementer is required to provide a module definition that instantiates the SRAM macros and connects the macro-specific pins to the interface described in Table 2.

Behavioral models of the RAMS are provided as part of the deliverable in the file: `verilog/memories/CoreIPSubsystem*`.

The VIC_E24 RAM instances are delivered as is and are not configurable. It is, however, possible to possible construct the memory instances from multiple smaller instances.

Module Name	Depth	Address Width (N_{addr})	Data Width (N_{data})	Write Mask Granularity (N_{part})	Description
<code>instr_mem_ext</code>	4096	12	32	32	ICache Data Array
<code>sysram0_ext</code>	8192	13	32	8	sys-sram-0
<code>sysram0_ext</code>	8192	13	32	8	sys-sram-1
<code>tag_mem_ext</code>	512	9	18	18	ICache Tag Array

Table 1: SRAM Modules and Configuration

Name	Direction	Width	Description
RW0_clk	Input	1	Memory clock
RW0_en	Input	1	Active-high signal indicating that the memory is being access. This may be used for clock gating.
RW0_addr	Input	N_{addr}	Address of access.
RW0_rdata	Output	N_{data}	Read data
RW0_wmode	Input	1	Active-high signal indicating that the access is a write operation.
RW0_wdata	Input	N_{data}	Write data.
RW0_wmask	Input	N_{data} / N_{part}	Active-high write mask. Each bit controls whether or not the corresponding N_{part} -bit subword is written. This is present only in memories that require mask write functionality.

Table 2: SRAM Signals

Chapter 4

VIC_E24 Interfaces

This chapter describes the primary interfaces to the VIC_E24.

4.1 Clock & Reset

The `clock`, `rtc_toggle`, `reset`, and `reset_vector_0` inputs are described in Table 3.

The relationship between the clock input frequencies are as follows:

$$\text{clock} > (2 * \text{rtc_toggle})$$

Name	Direction	Width	Description
<code>clock</code>	Input	1	The core pipeline and peripheral clock.
<code>rtc_toggle</code>	Input	1	The Real Time Clock input. Must run at strictly less than half the rate of <code>clock</code> .
<code>reset</code>	Input	1	Synchronous reset signal. Active high. Must be asserted for 16 cycles of <code>clock</code> and synchronously de-asserted.
<code>reset_vector_0</code>	Input	32	Reset Vector Address. Implementations MUST set this signal to a valid address.

Table 3: Clock and Reset Interfaces

4.1.1 Real Time Clock (`rtc_toggle`)

As defined in the RISC-V privileged specification, RISC-V implementations must expose a real-time counter via the `mtime` register. In the VIC_E24 the `rtc_toggle` input is used as the real-time counter. `rtc_toggle` must run at strictly less than half the frequency of `clock`. Furthermore, for RISC-V compliance, the frequency of `rtc_toggle` must remain constant, and software must be made aware of this frequency.

4.2 Ports

This section will describe all of the Ports in the VIC_E24.

Name	Base Address	Top	Protocol	Description
sys_port_ahb_0	0x8000_0000	0xFFFF_FFFF	AHB	32-bit data width. Synchronous to c1ock

Table 4: VIC_E24 Platform Bus Interfaces

4.2.1 System Port

The VIC_E24 has one System Port, which is typically used to access higher-bandwidth peripheral devices in off-core-complex address space. Errors that propagate to the processor via the Peripheral Port have the effects described in Chapter 5. The System Port is described in Table 4.

The VIC_E24 System Port pass through a TileLink to AHB Bridge (TL2AHB) which is described in Chapter 6.

4.3 VIC_E24 Interrupt Interfaces

This chapter describes all of the interrupt signals in the VIC_E24.

4.3.1 Machine External Interrupts

Name	Direction	Width	Description
meip_X	Input	1	Machine external interrupt signal exposed at the top level which can be used to integrate the VIC_E24 with an external Interrupt Controller.

Table 5: Machine External Interrupt Interface

4.3.2 Local External Interrupts

Local interrupts are interrupts which can be connected to peripheral sources and connect to the CLIC to signal directly to an individual hart. Please see the VIC_E24 Manual for a detailed description of local interrupts.

Name	Direction	Width	Description
local_interrupts_0	Input	127	Interrupts from peripheral sources destined to core 0. These are level-based interrupt signals connected to the CLIC and must be synchronous with c1ock

Table 6: Local External Interrupt Interface

4.4 Debug Output Signals

Signals which are outputs from the debug module are shown in Table 7.

Name	Direction	Width	Description
debug_ndreset	Output	1	This signal is a reset signal driven by the debug logic of the chip. It can be used to reset parts of the SoC or the entire chip. It should NOT be wired into logic which feeds back into the debug_systemjtag_reset signal for this block. This signal may be left unconnected.
debug_dmactive	Output	1	This signal, 0 at reset, indicates that debug logic is active. This may be used to prevent power gating of debug logic, etc. It may be left unconnected.

Table 7: External Debug Logic Control Pins

4.5 JTAG Debug Interface Pinout

SiFive uses the industry-standard JTAG interface which includes the four standard signals, TCK, TMS, TDI, and TDO. A test logic reset signal must also be driven on the debug_systemjtag_reset input. This reset is synchronized internally to the design. The test logic reset must be pulsed before the core reset is deasserted.

Name	Direction	Width	Description
debug_systemjtag_TCK	Input	1	JTAG Test Clock
debug_systemjtag_TMS	Input	1	JTAG Test Mode Select
debug_systemjtag_TDI	Input	1	JTAG Test Data Input
debug_systemjtag_TDO_data	Output	1	JTAG Test Data Output
debug_systemjtag_TDO_driven	Output	1	JTAG Test Data Output Enable
debug_systemjtag_reset	Input	1	Active-high Reset
debug_systemjtag_mfr_id	Input	11	The SoC Manufacturer ID which will be reported by the JTAG IDCODE instruction.

Table 8: SiFive standard JTAG interface for off-chip external TAPC

Chapter 5

VIC_E24 Error Handling

This chapter describes how the VIC_E24 handles errors from its memories and interfaces.

Errors can be introduced to the core via ECC errors or error responses returned on the various port interfaces. For port interfaces that are not natively TileLink, their error responses are translated into TileLink errors as described in the respective bridge chapters. The core's behavior is purely determined by the type of TileLink or ECC error that it receives. The behavior is also dependent on the type of core.

5.1 2 Series Error Handling

For 2-series cores, on the various interfaces to the VIC_E24:

- TileLink denied/corrupt on instruction fetch causes a precise exception
- TileLink denied/corrupt on data access is ignored by the core.

Note that 2-series TIMs are TileLink devices within the Core Complex boundary, so errors in those structures present TileLink errors to the core the same as devices outside the boundary.

Chapter 6

TileLink to AHB Bridge (TL2AHB)

6.1 Introduction

SiFive's TileLink to AHB Bridge (TL2AHB) can be used to connect SiFive Core Complex IP to AMBA 3 AHB Protocol v1.0 based systems. SiFive Core Complex IP natively uses TileLink for all system communication external to the Core Complex. The TL2AHB bridge translates TileLink transactions to AMBA 3 AHB Protocol v1.0.

6.2 Compliance

- The SiFive TL2AHB is fully compliant with AMBA 3 AHB Protocol v1.0 and this document should be read in conjunction with the AMBA 3 AHB Protocol v1.0 Protocol Specification.
- The SiFive TL2AHB is fully compatible with SiFive Core Complex IP. Some properties of the TL2AHB are specific to a given Core Complex implementation. This document should be read in conjunction with the Core Complex IP Manual.

6.3 Block Diagram

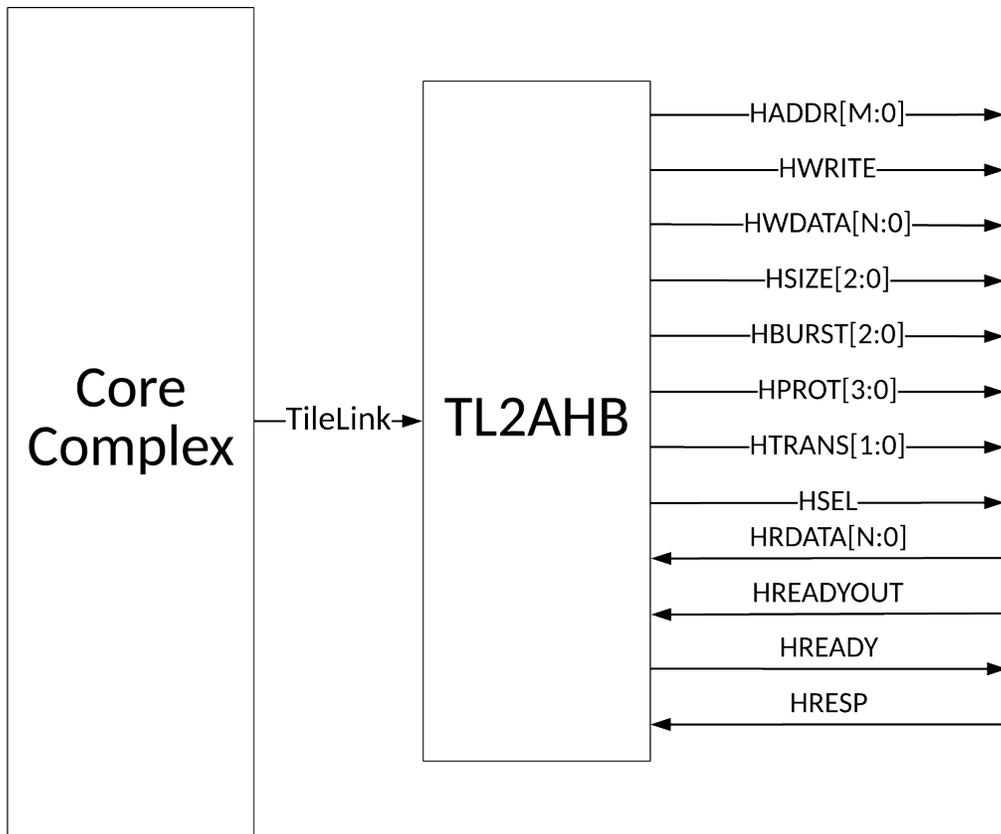


Figure 1: TL2AHB Block Diagram

6.4 TL2AHB Interface

Name	Direction	Width	Description
HADDR	Out	[M:0]	Transfer address where M is the minimum width necessary for the address range assigned to TL2AHB in a given Core Complex implementation.
HWRITE	Out	1	When HIGH this signal indicates a write transfer. When LOW this signal indicates a read transfer.
HWDATA	Out	[N:0]	Write transfer data where N is dependent on a given Core Complex implementation.
HSIZE	Out	[2:0]	The size of the transfer.
HBURST	Out	[2:0]	Burst type indicator.
HPROT	Out	[3:0]	Protection control signals.
HTRANS	Out	[1:0]	Transfer type of current transfer.
HRDATA	In	[N:0]	Read transfer data where N is dependent on a given Core Complex implementation.
HREADYOUT	In	1	Used by a slave to indicate when its transfer has finished. See Section 6.5.3 for more details.
HREADY	Out	1	Used to indicate to the master and all slaves that the previous transfer has finished. See Section 6.5.3 for more details.
HRESP	In	1	Response signal used to indicate transfer status: OKAY or ERROR.
HSEL	Out	1	Indicates that the current transfer is intended for the selected slave. See Section 6.5.3 for more details.

Table 9: AHB Interface

6.5 Functional Description

This section will describe the functional behavior of the TL2AHB in more detail.

6.5.1 Atomic Memory Operations (AMO)

- TileLink AMOs are translated to Read-Modify-Write operations and therefore no longer atomic.
- Because AMOs issued through the TL2AHB can not guarantee atomicity, they should not be issued through the TL2AHB in a multi-master system.

6.5.2 Bursts

- The TL2AHB will fragment a TileLink burst transaction to the largest supported AHB burst size
- Request are always aligned to the burst size.

- Only fixed size incrementing or single beat burst are issued, specifically:
 - SINGLE
 - INCR4
 - INCR8
 - INCR16
- Multi-beat narrow burst are never issued.

HADDR

- For a given Core Complex implementation, the width of HADDR is the minimum width necessary for the the address range of the TileLink bus it is connected to.

HMASTLOCK

- HMASTLOCK is always tied to 0.

HPROT

- HPROT is tied to 0x3: Non-cacheable, Non-bufferable, privileged, data access.

HSEL

- A single HSEL bit is implemented and used to indicate when the port is active.
- An external arbiter and decoder is required to select more than one slave device. See Section 6.5.3.

HRESP

- When HRESP indicates a transfer error, the signal is translated into the TileLink response **d_error**. OKAY = LOW.

6.5.3 TL2AHB System Integration

The TL2AHB is implemented in a way which allows for some flexibility when integrating into a larger system. It is possible to:

- Directly connect a single slave to the TL2AHB described in Section 6.5.3.1.
- Connect to a Decoder/Multiplexor which allows for multiple slave connections described in Section 6.5.3.2.

TL2AHB Direct Slave Connection

- In this use case, the TL2AHB signals map directly onto the Slave's interface. This is depicted in Figure 2.

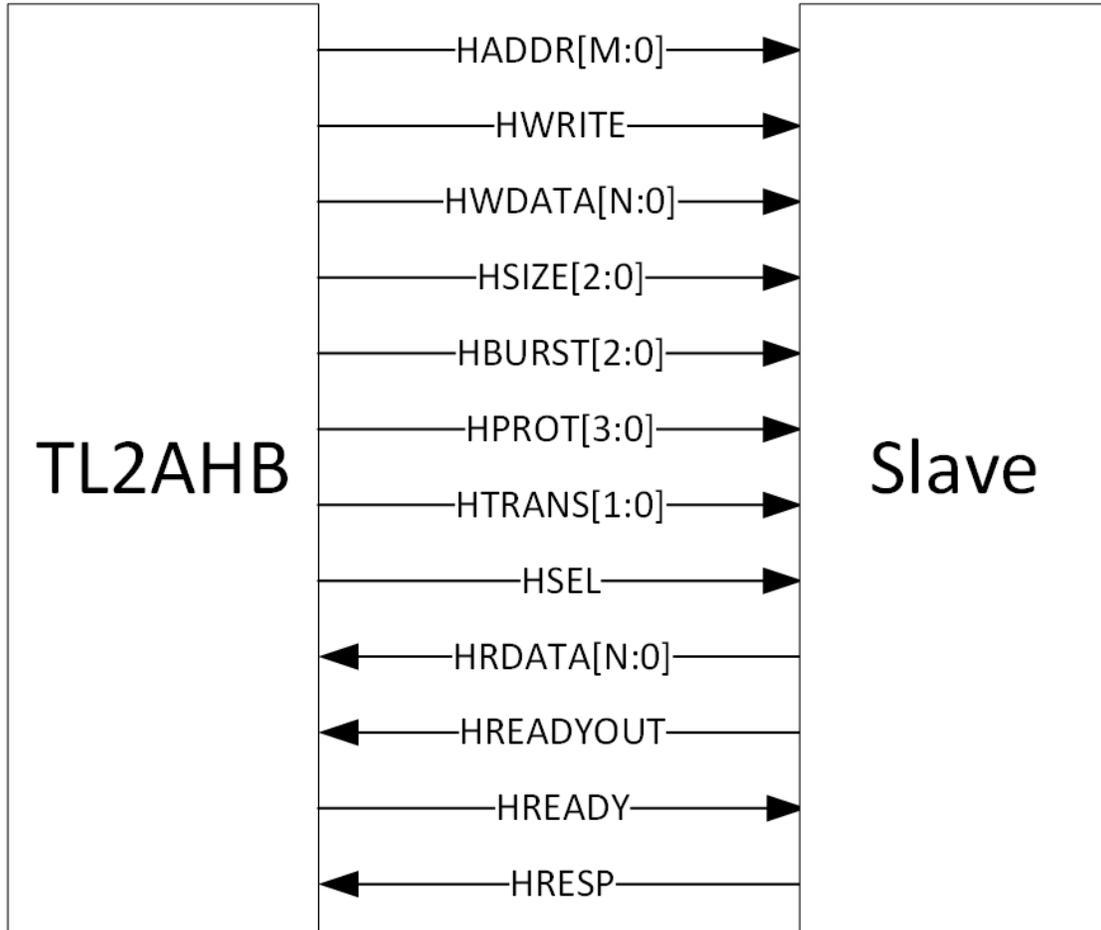


Figure 2: TL2AHB connected directly to a slave

TL2AHB Decoder/Multiplexor Connection

- The HREADY signal from the decoder should be connected to the TL2AHB's HREADYOUT signal.
- HSEL and HREADY on the TL2AHB should be left floating.

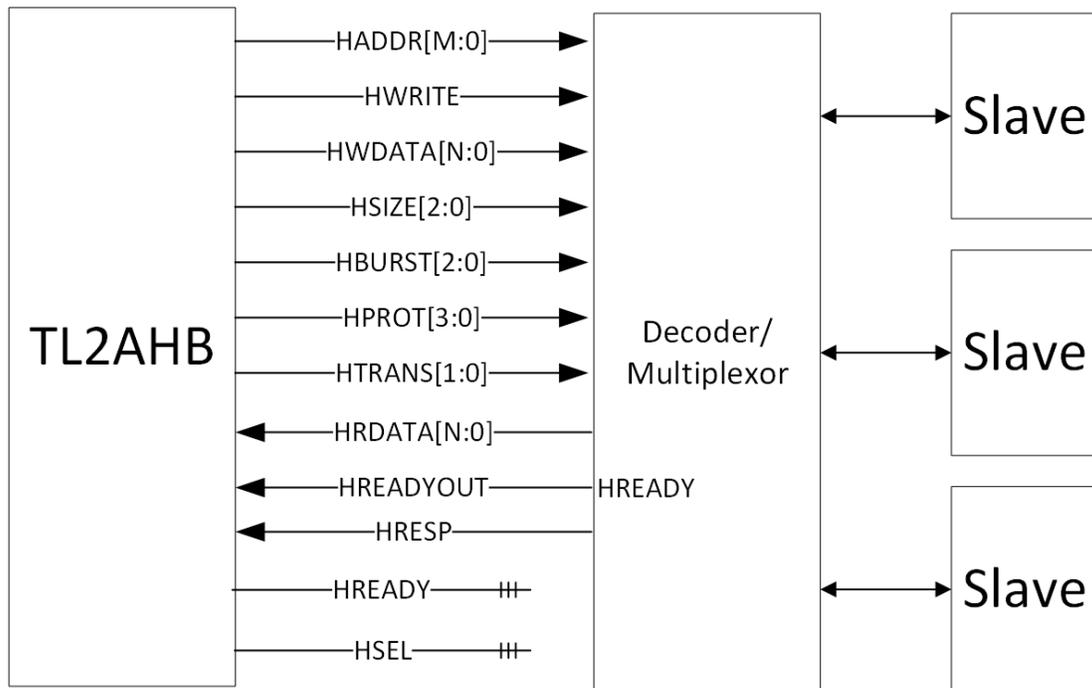


Figure 3: TL2AHB connected directly to a slave

Chapter 7

Debug Interface

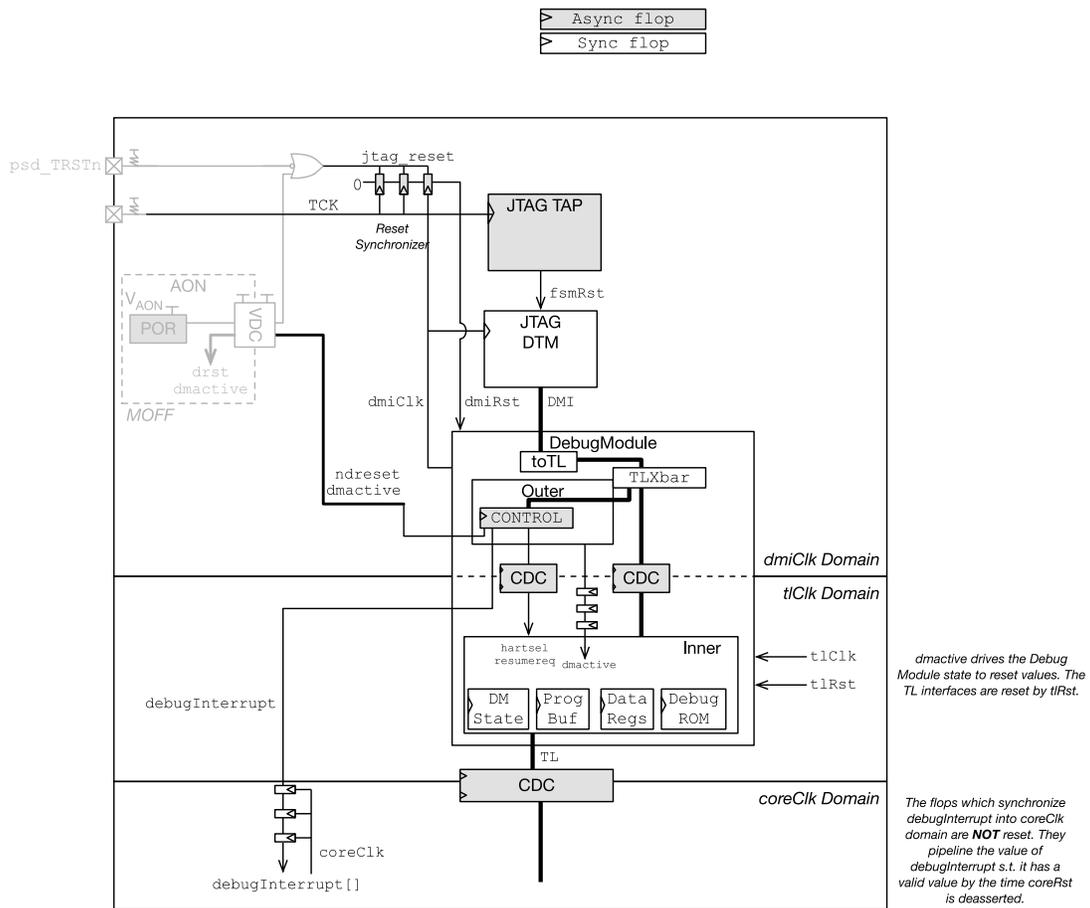


Figure 4: Debug Transport Module and Debug Module for HW Debug.

The SiFive VIC_E24 includes the JTAG debug transport module (DTM) described in *The RISC-V Debug Specification, Version 0.13*. This enables a single external industry-standard 1149.1 JTAG interface to test and debug the system. The JTAG interface can be directly con-

nected off-chip in a single-chip microcontroller or can be an embedded JTAG controller for a RISC-V Core IP designed to be included in a larger SoC.

The DTM and debug module are depicted in Figure 4.

On-chip JTAG connections must be driven (no pullups), with a normal 2-state driver for TDO under the expectation that on-chip mux logic will be used to select between alternate on-chip JTAG controllers' TDO outputs. TDO logic changes on the falling edge of TCK.

7.1 JTAG TAPC State Machine

The JTAG controller includes the standard TAPC state machine shown in Figure 5. 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.

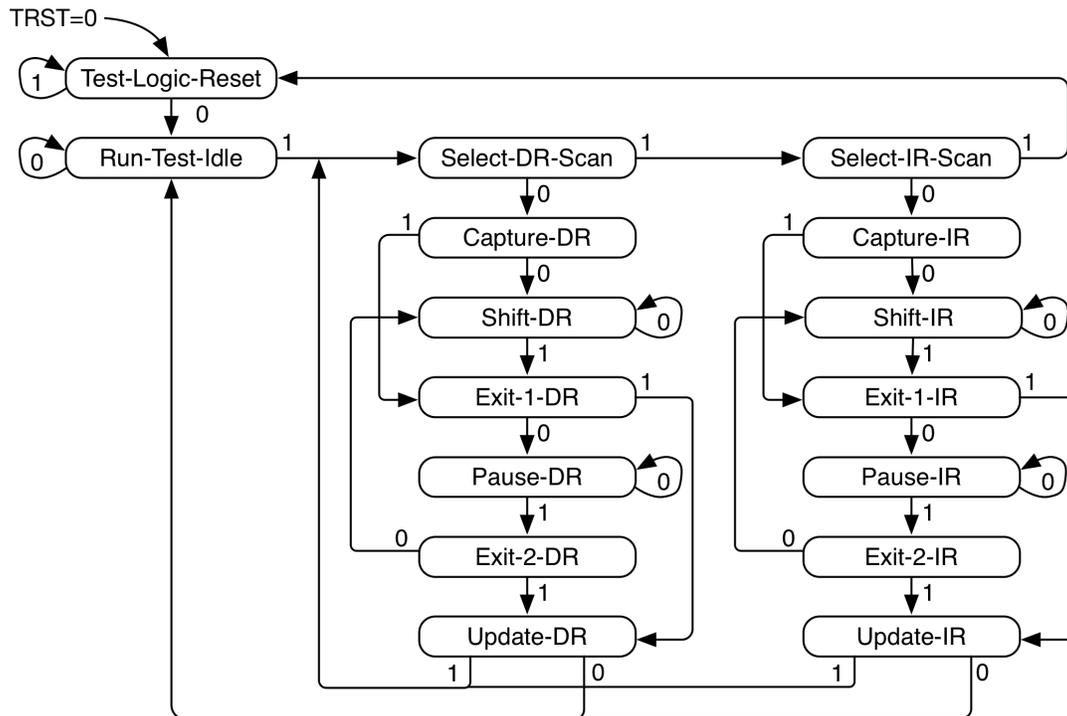


Figure 5: JTAG TAPC state machine.

7.2 Resetting JTAG Logic

The JTAG logic must be asynchronously reset by asserting `jtag_reset` before `coreReset` is deasserted.

Asserting `jtag_reset` resets both the JTAG DTM and debug module test logic. Because parts of the debug logic require synchronous reset, the `jtag_reset` signal is synchronized inside the `VIC_E24`.

During operation, the JTAG DTM logic can also be reset without `jtag_reset` by issuing 5 `jtag_TCK` clock ticks with `jtag_TMS` asserted. This action resets only the JTAG DTM, not the debug module.

7.3 JTAG Clocking

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

7.4 JTAG Standard Instructions

The JTAG DTM implements the `BYPASS` and `IDCODE` instructions.

The Manufacturer ID field of `IDCODE` is provided by the RISC-V Core IP integrator, on the `jtag_mfr_id` input.

7.5 JTAG Debug Commands

The JTAG `DEBUG` instruction gives access to the SiFive debug module by connecting the debug scan register between `jtag_TDI` and `jtag_TDO`.

The debug scan register includes a 2-bit opcode field, a 7-bit debug module address field, and a 32-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, Version 0.13*.

7.6 Using Debug Outputs

The debug module logic in SiFive Systems drives two output signals: `ndreset` and `dmactive`. These signals can be used in integration. It is suggested that the `ndreset` signal contribute to the system reset. It must be synchronized before it contributes back to the RISC-V Core IP's overall reset signal. This signal must not contribute to the `jtag_reset` signal. The `dmactive` signal can be used, for example, to prevent clock or power gating of the debug module logic while debugging is in progress.

Chapter 8

Implementation

This chapter will describe the steps necessary to synthesize VIC_E24.

8.1 Top Level

The top level verilog module is defined in the file:
`verilog/design/E2_CoreIPSubsystem.v`

All top level interfaces are described in the the VIC_E24 Chapter 4.

8.2 Clocking

The VIC_E24 has the following main clock inputs: `clock`, `rtc_toggle`. `clock` is used to clock the bus, PLIC and debug interfaces. `rtc_toggle` is exposed via the architecturally defined real time counter exposed in the `mtime` CSR.

The relationship between the clock input frequencies are as follows:

$$\mathit{clock} > (2 \times \mathit{rtc_toggle})$$

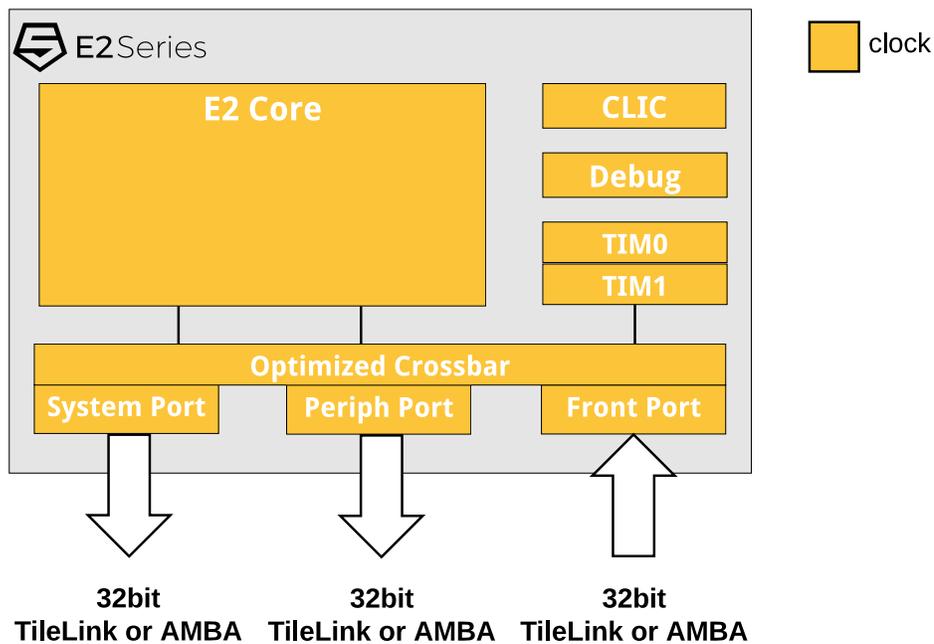


Figure 6: VIC_E24 Clock Diagram.

8.2.1 Clocking Guidelines

`rtc_toggle` is used by software for time keeping and therefore not necessary to have a high frequency. Generally, `rtc_toggle` is connected to either a Real Time Clock (e.g. 32.768 kHz) or to the base clock input frequency of the platform. The only restrictions on `rtc_toggle` are that it must be strictly less than half the frequency of `clock` and that it must remain constant. Furthermore, for RISC-V compliance, software must be made aware of this frequency via a header file or similar.

8.3 Retiming

A list of all modules (if any) which require retiming is included in the design deliverables. The list can be found in the `info/retiming_modules.txt` file.

8.4 Gate Level Simulation

To avoid X propagation, all state such as flip-flops and SRAMs must be initialized at the beginning of simulation, as well as after each power-up event if conducting power-aware simulation. Additionally, ensure that the SRAM models never produce X values by modifying them as necessary.

For Synopsys Design Compiler and IC Compiler, the command `all_registers -output_pins` can be used to enumerate all state elements in the design. The UCLI command `force`

-deposit <inst> <value> can be used in VCS simulation to force the output pins to 0 or 1. VCS also requires a PLI table file to enable wn capability (debug access) on these instances.

Note that if the standard cell logic library contains flip-flops with inverted outputs (i.e., Q and QN), those pins must be initialized to opposite values.

Chapter 9

Simulation Testbench

9.1 Included Test Bench

The VIC_E24 includes an example test simulation environment which is designed to work with Synopsys VCS version K-2015.09-SP2 or higher, Cadence Xcelium version 17.05 or higher, and Verilator version 4.014 or higher . Several tests are included as part of the testbench to verify functionality and can be run using the included Makefile. The testbench tests and their sources are located in the included Freedom E SDK environment, specifically the `freedom-e-sdk/software` folder. The Freedom E SDK environment contains all of the necessary source code, Makefiles, and utilities necessary to recompile the included tests as well as the ability to easily extend the testbench with additional tests. Freedom E SDK requires a suitable toolchain which can be downloaded from: <https://www.sifive.com/boards>

For more information on the Freedom E SDK environment, please read the readme file located in the `freedom-e-sdk` directory.

9.1.1 Executing the Testbench

GNU Make is used to build the RTL into a simulator and run the included binary test files. The Make targets are described below:

- `clean` - Cleans the build
- `all` - Runs all tests on all simulators
- `all-verilator` - Runs all tests using verilator
- `all-vcs` - Runs all tests using vcs
- `all-xrun` - Runs all tests using xcelium
- `all-waves` - Runs all tests while dumping waveforms in VPD format from all simulators
- `all-verilator-waves` - Runs all tests using verilator while dumping waveforms in VPD format
- `all-vcs-waves` - Runs all tests using vcs while dumping waveforms in VPD format
- `all-xrun-waves` - Runs all tests using xcelium while dumping waveforms in VPD format

- test.out - Runs the test called test
- test.vpd - Runs the test called test and produces a waveform in VPD format

Executing the command:

```
make all-waves
```

will run all of the tests in the tests folder and produce the resulting <test_name>.out and <test_name>.vpd files which can then be analyzed for detailed execution information.

9.2 Testbench Output

The testbench is capable of producing two types of output files: .out and .vpd. .out files contain a trace of all instructions executed by the processor. .vpd files contain VPD waveforms of the design which can be viewed with a waveform viewer such as Synopsys DVE.

9.2.1 Testbench Output - Trace

The test bench will produce output files with the filename <test_name>.out. The output files contain a cycle-by-cycle dump of a core's write-back stage. An example output is provided below:

Format:

```
core id: cycle [valid] pc=[address] Written[register=value][valid]
Read[register=value] Read[register=value]
```

Example:

```
C0: 483 [1] pc=[00000002138] W[r 3=000000007fff7fff][1] R[r 1=000000007fffffff] R[r
2=ffffffffffff8000]
C0: 484 [1] pc=[0000000213c] W[r29=000000007fff8000][1] R[r31=ffffffff80007ffe]
R[r31=0000000000000005]
C0: 485 [0] pc=[00000002140] W[r 0=0000000000000000][0] R[r 0=0000000000000000] R[r
0=0000000000000000]
```

The first [1] at cycle 483, core 0, shows that there's a valid instruction at PC 0x2138 in the writeback stage. The second [1] tells us that the register file is writing r3 with the corresponding value 0x7fff7fff. When the add instruction was in the decode stage, the pipeline had read r1 and r2 with the corresponding values next to it. Similarly at cycle 484, there's a valid instruction at PC 0x213c in the writeback stage. At cycle 485, there isn't a valid instruction in the writeback stage, perhaps, because of a instruction cache miss at PC 0x2140.

9.2.2 Testbench Output - Waves

When running the included testbench with the all-waves or <test_name>.vpd targets, the testbench will create VPD formatted waveforms which can be viewed with a waveform viewer such as Synopsys DVE. The waveform along with the trace log can be helpful when debugging tests run on the testbench.

9.2.3 Adding Tests To The Included Testbench

The simplest way to add new tests to the testbench is to start from one of the included tests. The return-pass test contains an empty main function which returns 0 (pass value) and is suitable for starting a new test. The example below demonstrates how to build a new test starting from return-pass.

- In the *freedom-e-sdk/software* directory, make a copy of the *return-pass* folder and name the copied folder to the name of your test. For this example we will use \$TEST.
- In the \$TEST directory, edit the makefile variable *PROGRAM* to match the name \$TEST.
- In the \$TEST directory, change the filename of *return-pass.c* to *\$TEST.c*
- Use the Freedom E SDK makefile to build the test targeting the *RTL* BSP and the *release* Configuration. The name of your BSP can be found in the *freedom-e-sdk/bsp* directory.

```
make TARGET=deliverable-name-rtl PROGRAM=$TEST CONFIGURATION=release software
```

- In the base directory of the deliverable, it is now possible to run the new test using the testbench makefile.

```
make $TEST.out
```

For more information on using Freedom E SDK and its environment, please read the readme file located in the *freedom-e-sdk* directory.

9.3 SiFive Insight

VIC_E24 is enabled with SiFive Insight technology which provides deep visibility into the design while at the same time being easily accessible. SiFive Insight is a verilog module that contains a curated list of signals chosen by the designers and presented in an intuitive hierarchy with descriptive names. SiFive Insight is primarily meant to be used during simulation waveform debugging and allows for a deep understanding of what is happening inside the SiFive deliverable without knowing details of the design.

Note that some signals in SiFive Insight are pseudo-signals which represent several signals with logic applied to them in order to present a more useful higher-level function. For example, the *Instruction Commit* signal in a design may be the logical combination of several signals. SiFive Insight also manages the grouping of signals to improve readability. An example of this would be how SiFive Insight presents the *mstatus* CSR. SiFive Insight presents *mstatus* such that each field in the CSR is grouped together, improving readability directly from the waveform viewer.

A complete list of SiFive Insight signals, along with descriptions, can be found in the *info/sifive_insight.yml* file located in the deliverables.

9.3.1 Viewing SiFive Insight Signals

Follow the instructions in Section 9.1.1 to execute the testbench and generate the resulting VPD wave files. This can be done with either the `make all-waves` or `make <test_name>.vpd` Make targets.

Once the test completes, open the VPD wave file with a waveform viewer such as Synopsys DVE. The SiFive Insight module can be found under the Verilog module hierarchy `TestDriver/testHarness/system/SiFive_Insight` or by simply searching for `SiFive_Insight`.

From here it is possible to add the SiFive Insight signals to the waveform viewer.

9.3.2 Enabling SiFive Insight Outside of the SiFive Testbench

To enable SiFive Insight in testbenches other than the one provided with the SiFive deliverables, simply include the SiFive Insight Verilog files with the testbench and design under test during the compilation of the simulator. The SiFive Insight Verilog files are located in the `verilog/sifive_insight/` directory. Waveform dumps will automatically include the SiFive Insight signals.