



HyperRAM Controller Core User Guide

UG-CORE-HYPERRAM-v3.0

April 2024

www.efinixinc.com



Contents

Introduction.....	3
Features.....	3
Device Support.....	3
Resource Utilization and Performance.....	4
Release Notes.....	4
Functional Description.....	5
Ports.....	7
AXI User Interface.....	12
AXI Addressing.....	12
Unaligned Address Access.....	12
AXI Read and Write Operation.....	13
Native User Interface.....	14
PLL Auto Calibration Flow.....	15
PLL Manual Calibration Flow.....	17
HyperRAM Controller Operation.....	17
Interface Designer Settings.....	20
PLL Auto Calibration and PLL Manual Calibration.....	20
Soft Logic Calibration.....	22
IP Manager.....	23
Customizing the HyperRAM Controller.....	24
HyperRAM Controller Example Design.....	26
Virtual I/O Debugger Settings.....	28
HyperRAM Controller Testbench.....	28
Revision History.....	29

Introduction

HyperRAM is a memory device that uses the HyperBus protocol. The HyperRAM Controller has two width options, x8 (13 I/O pins) and x16 (22 I/O pins). This flexibility allows designers to reduce the number of traces needed on the printed circuit board and thus is ideal for scalable solutions especially in automotive, industrial, and IoT applications. The HyperRAM Controller core interfaces Titanium FPGAs with HyperRAM memories.

Use the IP Manager to select IP, customize it, and generate files. The HyperRAM Controller core has an interactive wizard to help you set parameters. The wizard also has options to create a testbench and/or example design targeting an Efinix® development board.



Note: The HyperRAM Controller is only meant to communicate with single discrete HyperRAM chip.

Features

- PLL auto calibration, PLL manual calibration, and soft logic calibration
- x8 and x16 RAM bit widths
- Supports double-data rates of up to 800 MBps for x16 width configuration
- Supports up to 256 Mb HyperRAM
- Linear and wrap burst transfer
- AXI3 half-duplex or native interface to core
- 32, 64, 128, and 256 bit data width
- Includes Verilog HDL RTL and simulation testbench
- Includes example designs targeting the Titanium Ti60 F225 Development Board and Titanium Ti60 F100S3F2 FPGA

Device Support

Table 1: HyperRAM Controller Core Device Support

FPGA Family	Supported Device
Trion	All ⁽¹⁾
Titanium	All

Efinix® have verified HyperRAM Controller core for the Winbond (part no: W958D6NWS) HyperRAM with Titanium FPGAs.

⁽¹⁾ Soft Logic Calibration mode only.

Resource Utilization and Performance



Note: The resources and performance values provided are based on some of the supported FPGAs. These values are just guidance and can change depending on the device resource utilization, design congestion, and user design.

Table 2: Titanium Resource Utilization and Performance

FPGA	Calibration Mode	Logic Elements (Logic, Adders, Flipflops, etc.)	Memory Blocks	DSP Blocks	Efinity® Version ⁽²⁾
Ti60 F225 C4	AXI PLL Auto	2,033/60,800 (3.3%)	25/256 (9.7%)	0/160 (0%)	2023.2
	Native PLL Auto	1,730/60,800 (2.8%)	22/256 (8.5%)	0/160 (0%)	

Release Notes

You can refer to the IP Core Release Notes for more information about the IP core changes. The IP Core Release Notes is available in the [Efinity Downloads](#) page under each Efinity software release version.



Note: You must be logged in to the Support Portal to view the IP Core Release Notes.

⁽²⁾ Using Verilog HDL.

Functional Description

The HyperRAM Controller core supports three calibration modes:

- *PLL auto calibration*—Utilizes the Titanium PLL's dynamic shifting to calibrate incoming data from the HyperRAM.
- *PLL manual calibration*—Allows you to manually calibrate the PLL phase settings used to capture the incoming data without using the auto calibration logic.
- *Soft logic calibration*—Uses extra logic to calibrate incoming data from the HyperRAM.



Note: The PLL calibration mode HyperRAM Controller I/O's path is not impacted by design recompilations, therefore Efinix recommends that you use PLL auto calibration for Titanium FPGAs.

Figure 1: HyperRAM Controller PLL Calibration Block Diagram

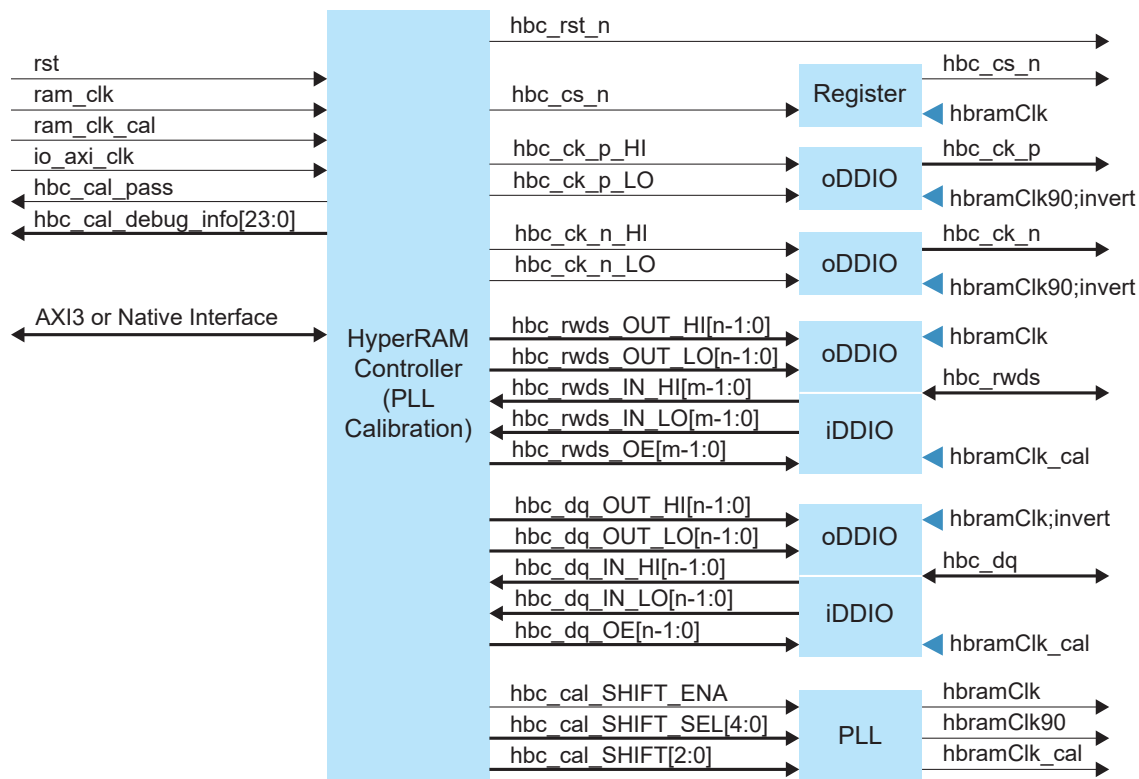


Figure 2: HyperRAM Controller PLL Manual Calibration Block Diagram

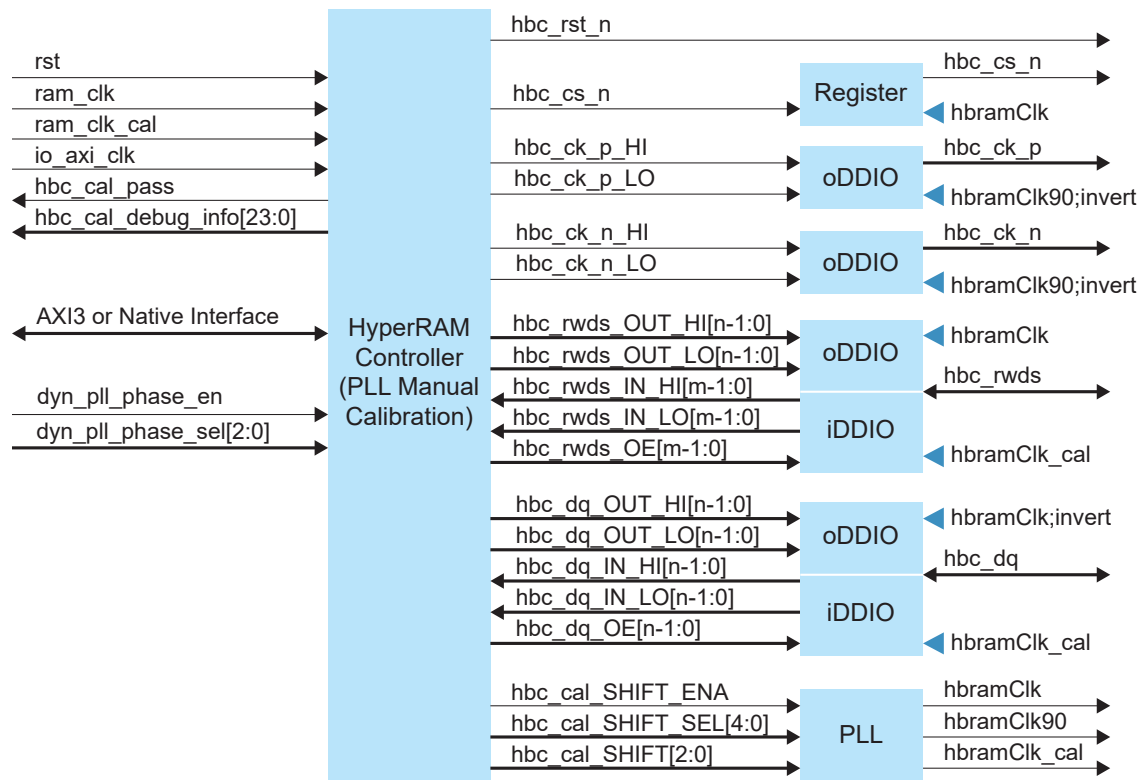
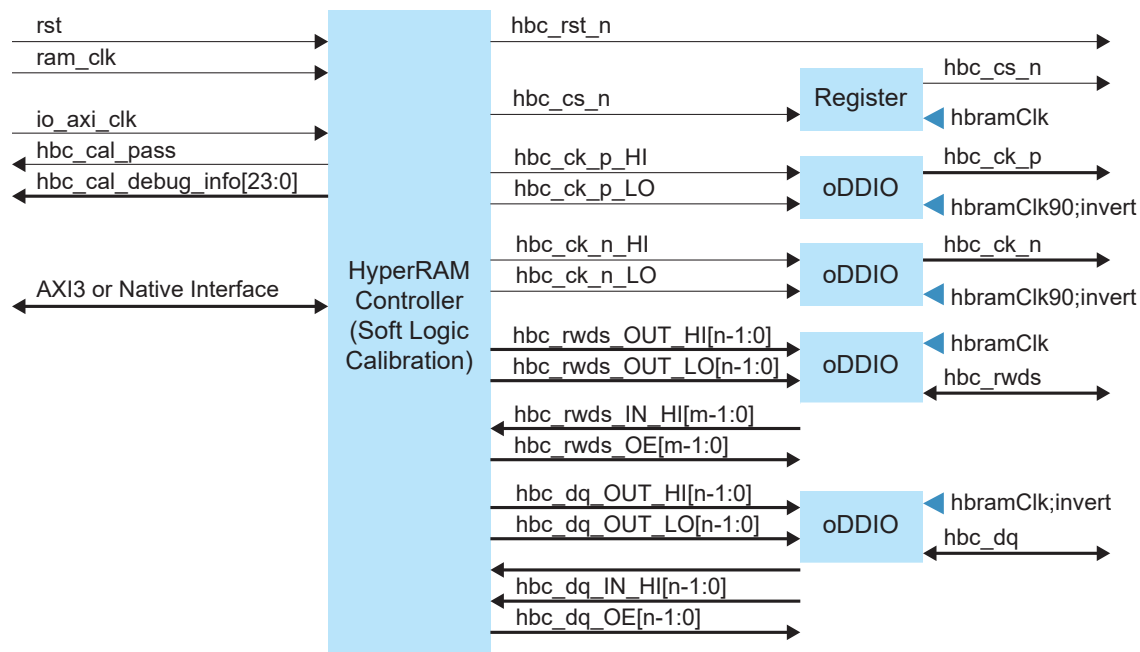


Figure 3: HyperRAM Controller Soft Logic Calibration Block Diagram



Ports

Table 3: HyperRAM Controller Ports

Port	Direction	Description										
hbc_cs_n	Input	Bus transactions are initiated with a logic high-to-low transition. Bus transactions are terminated with a logic low-to-high transition. The master device has a separate CS# signal for each slave.										
rst	Input	Core asynchronous reset, active high.										
ram_clk	Input	RAM operating clock.										
ram_clk_cal		RAM calibration clock. Used for PLL auto and manual calibration only.										
hbc_rst_n	Output	When logic low, the slave device self-initializes and returns to the standby state. RWDS and DQ signals are placed into the high-Z state when the RESET# signal is low.										
hbc_cal_pass	Output	Indicates calibration is passing.										
hbc_cal_debug_info [23:0]	Output	<table><tr><th>Bit</th><th>Description</th></tr><tr><td>15:8</td><td>Delay steps that received matched calibration data. One-hot encoded. Bit 0: Delay step 0 received matched data Bit 1: Delay step 1 received matched data Bit 2: Delay step 2 received matched data Bit 3: Delay step 3 received matched data Bit 4: Delay step 4 received matched data Bit 5: Delay step 5 received matched data Bit 6: Delay step 6 received matched data Bit 7: Delay step 7 received matched data</td></tr><tr><td>7:5</td><td>Locked down PLL phase shifted step</td></tr><tr><td>1</td><td>Calibration pass</td></tr><tr><td>0</td><td>Calibration done</td></tr></table> Bit [23:16] are reserved.	Bit	Description	15:8	Delay steps that received matched calibration data. One-hot encoded. Bit 0: Delay step 0 received matched data Bit 1: Delay step 1 received matched data Bit 2: Delay step 2 received matched data Bit 3: Delay step 3 received matched data Bit 4: Delay step 4 received matched data Bit 5: Delay step 5 received matched data Bit 6: Delay step 6 received matched data Bit 7: Delay step 7 received matched data	7:5	Locked down PLL phase shifted step	1	Calibration pass	0	Calibration done
Bit	Description											
15:8	Delay steps that received matched calibration data. One-hot encoded. Bit 0: Delay step 0 received matched data Bit 1: Delay step 1 received matched data Bit 2: Delay step 2 received matched data Bit 3: Delay step 3 received matched data Bit 4: Delay step 4 received matched data Bit 5: Delay step 5 received matched data Bit 6: Delay step 6 received matched data Bit 7: Delay step 7 received matched data											
7:5	Locked down PLL phase shifted step											
1	Calibration pass											
0	Calibration done											
hbc_ck_p_HI	Output	Differential Clock: Command, address, and data information is output with respect to the crossing of the CK and CK# signals. Single Ended Clock: CK# is not used, only a single ended CK is used. The clock is not required to be free-running.										
hbc_ck_p_LO	Output											
hbc_ck_n_HI	Output											
hbc_ck_n_LO	Output											
hbc_dq_OUT_HI [n-1:0]	Output	DQ output ports for command, address, and data. <i>n</i> = 8 (x8 mode), 16 (x16 mode)										
hbc_dq_OUT_LO [n-1:0]	Output											
hbc_dq_OE [n-1:0]	Output	DQ output enable port. <i>n</i> = 8 (x8 mode), 16 (x16 mode)										

Port	Direction	Description
hbc_dq_IN_HI [$n-1:0$]	Input	DQ input ports for data. $n = 8$ (x8 mode), 16 (x16 mode)
hbc_dq_IN_LO [$n-1:0$]	Input	DQ input ports for data. $n = 8$ (x8 mode), 16 (x16 mode)
hbc_rwds_OUT_HI [$m-1:0$]	Output	RWDS output ports for data mask during write operation. $m = 1$ (x8 mode), 2 (x16 mode)
hbc_rwds_OUT_LO [$m-1:0$]	Output	
hbc_rwds_IN_HI [$m-1:0$]	Input	RWDS input ports for latency indication, also center-aligned reference strobe for read data. $m = 1$ (x8 mode), 2 (x16 mode)
hbc_rwds_IN_LO [$m-1:0$]	Input	RWDS input ports for latency indication, also center-aligned reference strobe for read data. $m = 1$ (x8 mode), 2 (x16 mode) Applicable in PLL auto and manual calibration only.
hbc_rwds_OE [$m-1:0$]	Output	RWDS output enable port. $m = 1$ (x8 mode), 2 (x16 mode)
hbc_cal_SHIFT_ENA	Output	Enables PLL dynamic shifting. Applicable in PLL auto and manual calibration only.
hbc_cal_SHIFT_SEL[4:0]	Output	Selects PLL output. Applicable in PLL auto and manual calibration only.
hbc_cal_SHIFT[2:0]	Output	Delay steps value. Applicable in PLL auto and manual calibration only.
dyn_pll_phase_en	Input	Enable PLL phase adjustment. Used for PLL manual calibration only. 1'b0: Enable 1'b1: Disable
dyn_pll_phase_sel [2:0]	Input	PLL phase adjustment. 8 steps of the PLL dynamic phase shift settings. Used for PLL manual calibration only.

Table 4: HyperRAM Controller AXI Ports

Port	Direction	Description
io_axi_clk	Input	AXI interface operating frequency.
io_arw_valid	Input	Indicates that the channel is signaling a valid write/read address and control information.
io_arw_ready	Output	Indicates that the controller is ready to accept an address and associated control signals.
io_arw_payload_addr [31:0]	Input	The write address gives the address of the first transfer in a write/read burst transaction.
io_arw_payload_id [7:0]	Input	Identification tag for the write/read address group of signals.
io_arw_payload_len [7:0]	Input	Burst length. Indicates the exact number of transfers in a burst. Determines the number of data transfers associated with the address. Effective burst length = io_arw_payload_len + 1
io_arw_payload_size [2:0]	Input	Burst size. Indicates the size of each transfer in the burst. 3'b000: 1 3'b001: 2 3'b010: 4 3'b011: 8 3'b100: 16 3'b101: 32 3'b110: 64 3'b111: 128
io_arw_payload_burst [1:0]	Input	Burst type. The burst type and the size information, determines how the address for each transfer within the burst is calculated. The controller does not support fixed burst, only linear burst. 2'b00, b01: Linear burst 2'b10: Wrap burst
io_arw_payload_lock	Input	Reserved.
io_arw_payload_write	Input	Indicates the channel is accepting a write or read transfer. 1'b0: Read 1'b1: Write
io_w_payload_id [7:0]	Input	ID tag of the write data transfer.
io_w_valid	Input	Write valid. Indicates that valid write data and strobes are available.
io_w_ready	Output	Write ready. Indicates that the slave can accept the write data.
io_w_payload_data [n-1:0]	Input	Write data. $n = \text{AXI_DBW}$
io_w_payload_strb [n-1:0]	Input	Write strobes. Indicates which byte lanes hold valid data. There is one write strobe bit for each eight bits of the write data bus. $n = \text{AXI_DBW}/8$
io_w_payload_last	Input	Write last. Indicates the last transfer in a write burst.
io_b_valid	Input	Write response valid. Indicates that the channel is signaling a valid write response.
io_b_ready	Output	Response ready. Indicates that the master can accept a write response.
io_b_payload_id [7:0]	Output	Response ID tag. ID tag of the write response.

Port	Direction	Description
io_r_valid	Output	Read address valid. Indicates that the channel is signaling valid read address and control information.
io_r_ready	Input	Read ready. Indicates that the master can accept the read data and response information.
io_r_payload_data	Output	Read data.
io_r_payload_id [7:0]	Output	Read ID tag. Identification tag for the read data group of signals generated by the controller.
io_r_payload_resp [1:0]	Output	Read response. Indicates the status of the read transfer. This controller only responds 'b00 or OKAY.
io_r_payload_last	Output	Read last. Indicates the last transfer in a read burst.

Table 5: HyperRAM Controller Native Ports

Port	Direction	Description
native_ram_rdwrr	Input	HyperRAM write/read control. 1'b0 : Target for write 1'b1 : Target for read
native_ram_en	Input	Initiate a single pulse to trigger controller write/read to HyperRAM. For write operation, you must ensure that the data stored in the write buffer fulfills the configured burst length requirement before pulsing the signal. After pulsing the signal, monitor the native_ctrl_idle signal transit into low state to indicate the controller has started to process the request.
native_ram_burst_len [11:0]	Input	HyperRAM transaction burst length. You can dynamically change the burst length to maximize the transfer efficiency while still fulfilling the memory CS# maximum low time, 4 μ s (typical). The burst length number must not exceed the CS# maximum low time which is governed by the HyperRAM specification. $\text{FIFO Depth Size} = ((\text{DQ Width} * 2) / \text{Data width}) * \text{native_ram_burst_len}$ For example: 512 burst length with x16 RAM at 200 MHz will have approximately 2.6 μ s CS# low time.
native_ram_address [31:0]	Input	HyperRAM write/read address. The address width depends on the memory density.
native_wr_en	Input	Buffer write enable. Assert this signal when you want to place the write data into the write buffer. You can fill-in the write buffer at any time if the write buffer is not full indicated by the native_wr_buf_ready signal.
native_wr_data [n-1:0]	Input	Write data is placed in the write buffer before the native_ram_en signal is pulsed. $n = \text{AXI_DBW}$
native_wr_datamask [n/8-1:0]	Input	Write data mask. Set all to logic low if you do not use the write data mask. Otherwise, drive the signal per data byte (8 bits) granularity. $n = \text{AXI_DBW}$
native_rd_data[n-1:0]	Output	Read data width. $n = \text{AXI_DBW}$
native_rd_valid	Output	Read valid width. Logic high indicates that the returning data is valid.

Port	Direction	Description
native_ctrl_idle	Output	Logic high indicates that the controller is in idle mode. After you issue the native_ram_en signal, this signal will take multiple cycles to respond due to the clock domain crossing operation. Ensure that the signal is deasserted after issuing a write or read operation.
native_wr_buf_ready	Output	Write buffer availability. Logic high indicates that the write buffer is full and the write buffer ignores any incoming write data.

AXI User Interface

AXI Addressing

The AXI interface uses byte addressing, where each address holds 8-bit data. However, for HyperRAM, the addressing depends on the RAM bit width. For x8 mode, each address holds 16-bit data and for x16 mode, each address holds 32-bit data. The HyperRAM Controller core converts the AXI address to match the HyperRAM address.

Table 6: AXI Address Conversion Ratio

Mode	Divider Ratio
X16	4
X8	2

Unaligned Address Access

The AXI interface does not support unaligned address transfers. Instead, the AXI data width determines the address access.

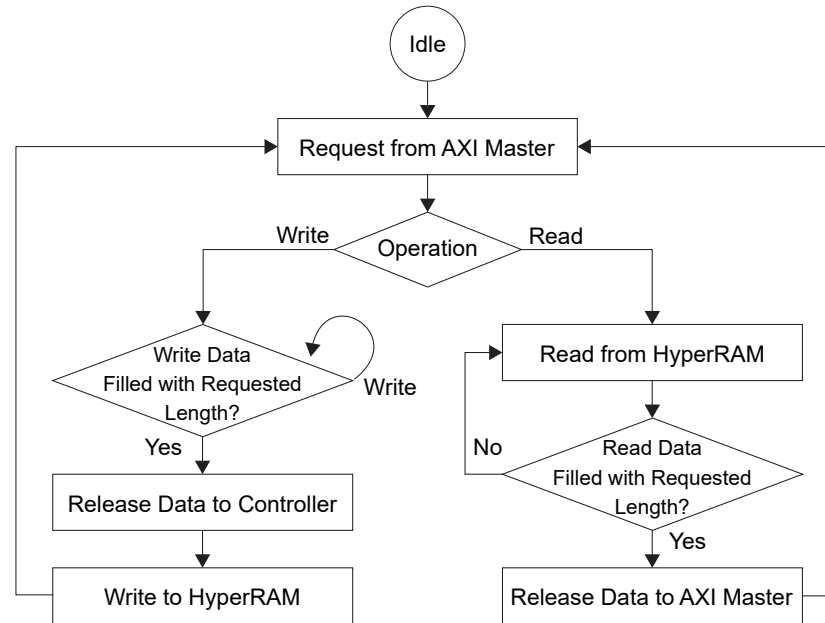
AXI Bit Width	Base Address Multiplier
256	32
128	16
64	8
32	4

For example, if you request h000012F8 on a 128-bits data width, the HyperRAM Controller core aligns the address to the nearest base address, which is h000012F0.

AXI Read and Write Operation

The following flow diagram explains the AXI read and write operation in the HyperRAM Controller core.

Figure 4: AXI3 Operation Flow Diagram



Native User Interface

The native user interface allows you to access memory in a more direct way, for example, you can issue an actual RAM address, controlling write or read operations, set RAM burst transaction length, or perform write mask as per the HyperRAM supported features. Asynchronous data FIFO is implemented between the user logic and the controller to facilitate the clock crossing between two different domains.

The native user interface uses word addressing which is according to the width of the HyperRAM, and not the native data width. For example, on an x16DQ HyperRAM, each address holds 32-bit data. Subsequently, on a x8DQ HyperRAM, each address holds 16-bit data.

This also applies to burst length as well. For example, on a x16DQ HyperRAM, one burst length is fixed to 32-bit data, a burst length of 10 will transfer a total of $10 \times 32\text{-bit} = 320$ bits of data to the HyperRAM. On a x8DQ HyperRAM, one burst length is fixed to 16-bit data.

For write operations, you need to set the write buffer depth as per the targeted RAM write length. Since the write operation is continuous, you need to fill-up the data buffer before issue a write trigger. You can store the data in the buffer at any time if the buffer is not full. For example, if the targeted write length (`native_ram_burst_len`) is 256-bit:

- If the native data width is 32-bit, the write buffer depth must be at least 256
- If the native data width is 64-bit, the write buffer depth must be at least 128

You calculate the write FIFO depth size with the following formula:

Write FIFO Depth Size = $((\text{DQ Width} \times 2) / \text{Data width}) \times \text{native_ram_burst_len}$



Note: A single write or read operation should not exceed the 4 us time memory CS# maximum low time. See [Table 5: HyperRAM Controller Native Ports](#) on page 10 for `native_ram_burst_len` detailed description.

For read operations, the data returns to the user logic side with `native_rd_valid` set high whenever the data is available.

In both write and read operations, you should always monitor the controller status signal, `native_ctrl_idle`. This signal should go high after you issue a transaction to ensure the controller has carry out the request.

PLL Auto Calibration Flow

Upon power-up, the AXI wrapper block `io_arw_ready` is asserted high to avoid taking any requests from the AXI master before calibration completes. The calibration master module sends the default setting of configuration register 0 and configuration register 1 to the HyperRAM and follows by writing 512 bytes of data on address `'h00000000`. The calibration slave module starts reading data from the HyperRAM continuously and adjusts the delay by providing different dynamically shifted clock (incrementing from 0 to 7, on recommended PLL settings each step is 45° (degree) delayed). This value changes the:

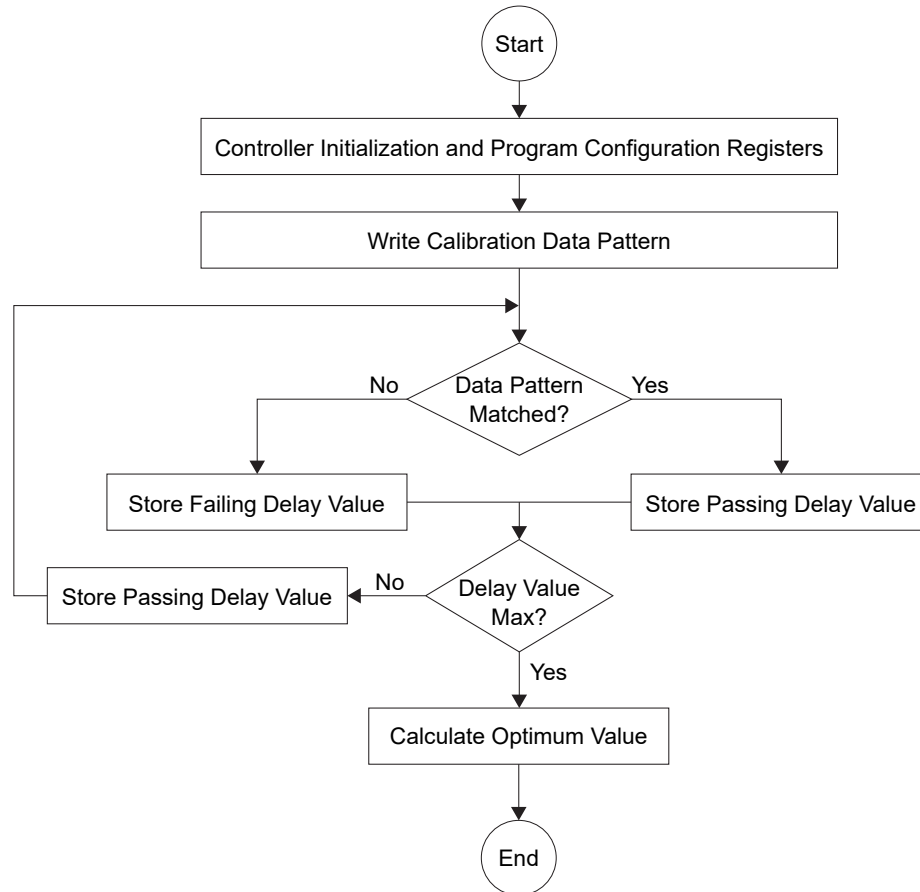
- *hbramClk_cal phase (PLL auto calibration)*—The `hbramClk_cal` is used to register the input of data and read/write data strobe (RWDS) in double-data rate I/O (DDIO) mode.
- *Mux delay module (soft logic calibration)*—The mux delay module is a chain of mux to emulate the delay path to DQ and RWDS.

Once the calibration slave module receives matched data from the HyperRAM, it stores the delay value, you can refer the delay values from `hbc_cal_debug_info[15:8]`. It keeps testing by incrementing the delay value and eventually chooses the optimum value to lock on. The HyperRAM Controller core asserts `hbc_cal_pass` high once to exit calibration and assert `io_arw_ready` signal.

If there is no matching data from the HyperRAM after 8 (eight) delays, calibration is failed and `hbc_cal_pass` remains 0 with `hbc_cal_done` becomes 1.

The HyperRAM Controller Core performs a one-off initial calibration. It is recommended to perform periodic re-calibration by resetting the HyperRAM Controller Core to maintain the optimal settings on the interface between the controller and the memory, or experiencing errors or unexpected behavior.

Figure 5: Calibration Flow Diagram



PLL Manual Calibration Flow

In PLL Manual calibration mode, you need to manually set the PLL phase shift by changing the `dyn_pll_phase_sel [2:0]`. You can leverage on the PLL Auto calibration mode to search for the optimum settings before applying the settings in your design. Refer to the [HyperRAM Controller Example Design](#) on page 26. However, the fixed settings you use in the PLL Manual calibration mode may not be the optimum setting when there are variations in voltage and temperature.

HyperRAM Controller Operation

The following waveforms describes timing sequence of signals between the HyperRAM Controller and interface connections during write and read operations.

Hyperbus Interface

Figure 6: Hyperbus Interface Write Operation Waveform

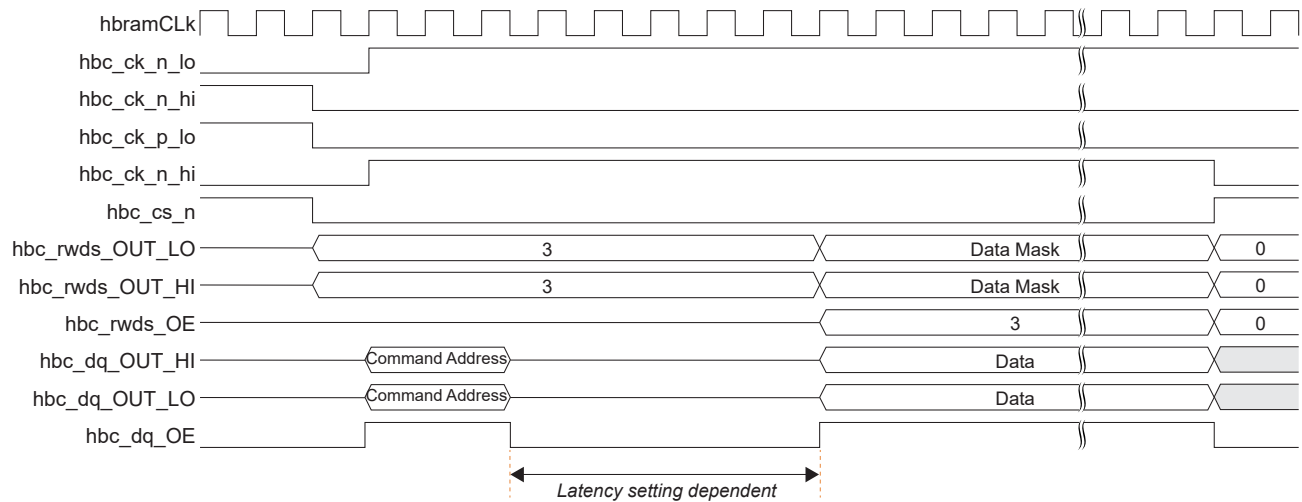
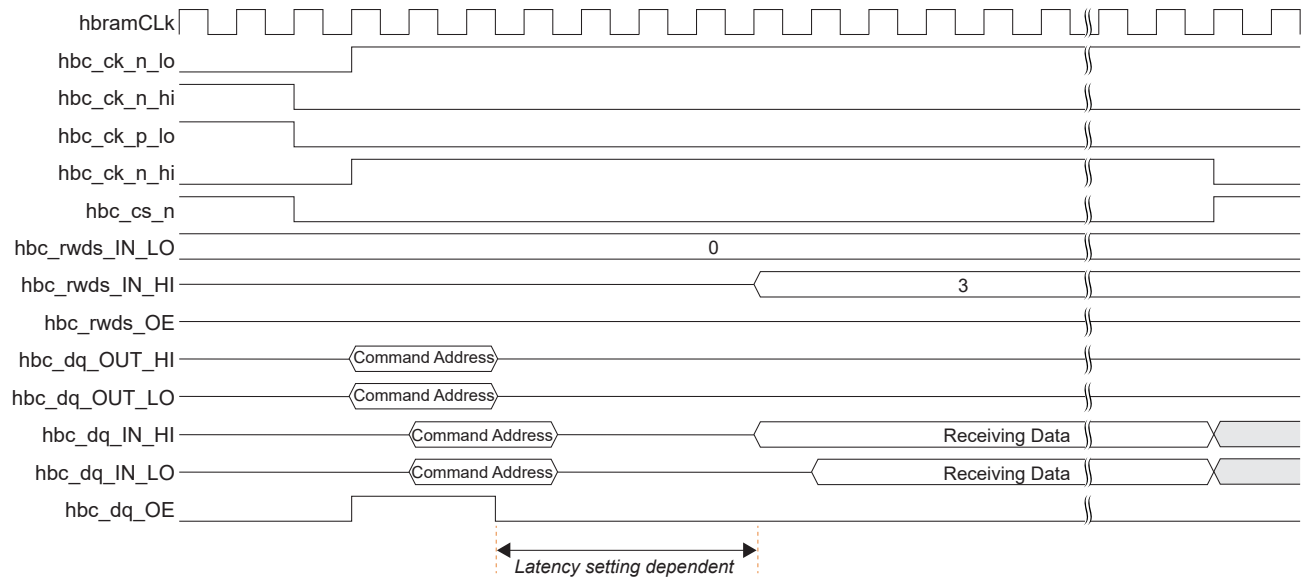


Figure 7: Hyperbus Interface Read Operation Waveform

Native Interface



Note: Native interface mode only supports linear mode operation and requires the `native_ram_address[31]` to always be set to high.

Figure 8: Native Interface Write Operation Waveform

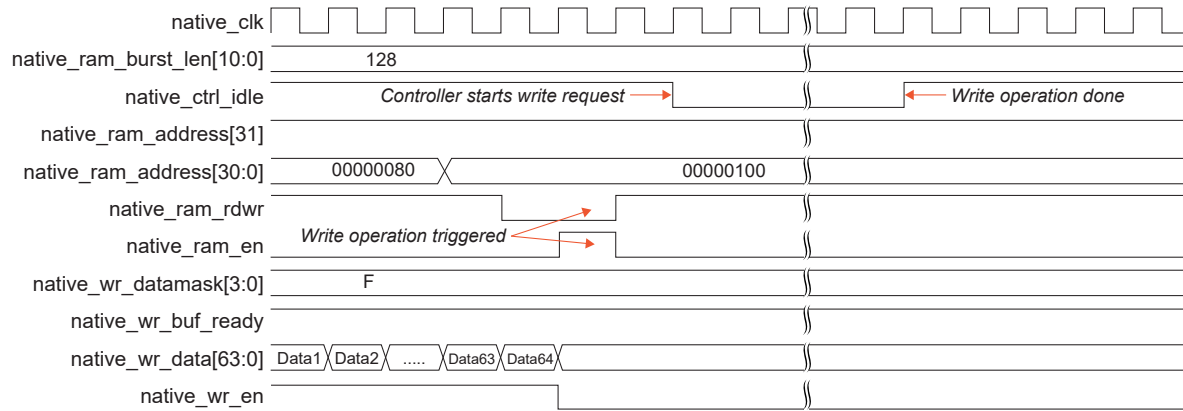
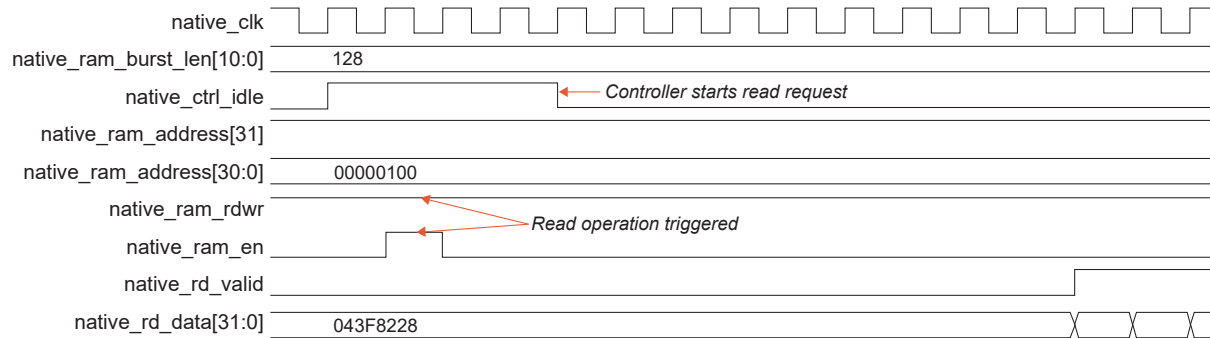


Figure 9: Native Interface Read Operation Waveform



Interface Designer Settings

When using the HyperRAM Controller with external memories, you need to create GPIO blocks and PLL output clocks in the Efinity® Interface Designer with the settings shown in following tables.

The Ti60 F100S3F2 FPGA has an embedded HyperRAM memory, so you do not need to create any GPIO to connect to it. Instead, you add a HyperRAM block to your interface design and then connect your RTL design to the block's pins. You also need to use a PLL for calibration as described in [PLL Auto Calibration and PLL Manual Calibration](#) on page 20. Efinix provides an example design targeting this FPGA as described in [HyperRAM Controller Example Design](#) on page 26, which you can use to get started.

PLL Auto Calibration and PLL Manual Calibration

Table 7: Interface Designer Settings for GPIO

Block/Bus	Settings					
	Mode	Register Option	Double Data I/O Option	Pull Option	Clock	Inverted Clock
hbc_rst_n	output	none	-	-	-	-
hbc_cs_n	output	register	-	-	hbramClk	-
hbc_ck_p_HI	output	register	normal	-	hbramClk90	Yes
hbc_ck_p_LO	output	register	normal	-	hbramClk90	Yes
hbc_ck_n_HI	output	register	normal	-	hbramClk90	Yes
hbc_ck_n_LO	output	register	normal	-	hbramClk90	Yes
hbc_rwds_OUT_HI [1:0]	output	register	normal	-	hbramClk	-
hbc_rwds_OUT_LO [1:0]	output	register	normal	-	hbramClk	-
hbc_rwds_OE [1:0]	output	register	-	-	hbramClk	-
hbc_rwds_IN_HI [1:0]	input	register	resync	weak pulldown	hbramClk_Cal	-
hbc_rwds_IN_LO [1:0]	input	register	resync	weak pulldown	hbramClk_Cal	-
hbc_dq_OUT_HI [15:0]	output	register	normal	-	hbramClk	Yes
hbc_dq_OUT_LO [15:0]	output	register	normal	-	hbramClk	Yes
hbc_dq_OE [15:0]	output	register	-	-	hbramClk	Yes
hbc_dq_IN_HI [15:0]	input	register	resync	weak pulldown	hbramClk_Cal	-
hbc_dq_IN_LO [15:0]	input	register	resync	weak pulldown	hbramClk_Cal	-

You must ensure that the step for the dynamic phase shift has 45 degree phase coverage. The formula to calculate a single phase step coverage is given by:

$$\text{Single phase step coverage} = (0.5 \times \text{Post Divider (O)} \times \text{Final Clock Out}) / F_{VCO} \times 360$$



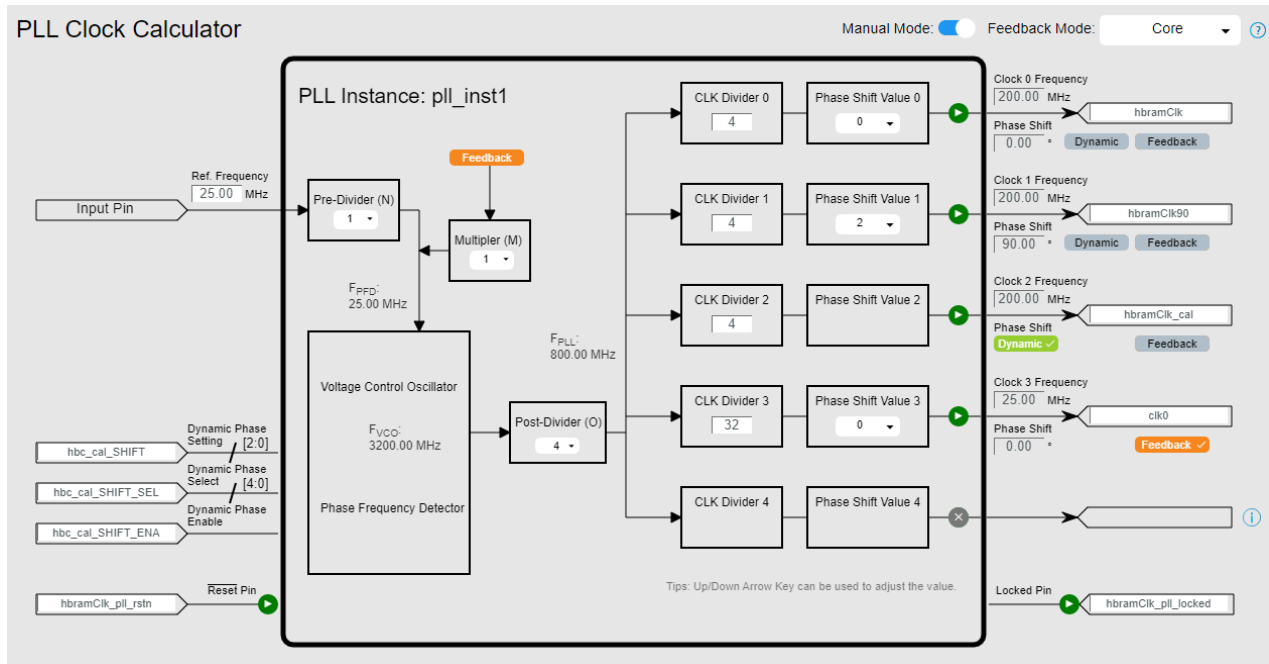
Note: Efinix recommends that you set the **CLK Divider** to 4 to get a 45° per phase shift for the hbramClk, hbramClk90, and hbramClk_cal clocks.

Table 8: PLL Settings for Various Clock Out Frequency

These settings are based on 25 MHz PLL refclk.

Target Clock Out Frequency	F _{VCO}	Post-Divider (O) Setting
200	3200	4
150	4800	8
125	4000	8
100	3200	8
50	3200	16

Figure 10: PLL Settings Example for 200 MHz Configuration



Soft Logic Calibration

Table 9: Interface Designer Settings for GPIO

Block/Bus	Settings					
	Mode	Register Option	Double Data I/O Option	Pull Option	Clock	Inverted Clock
hbc_rst_n	output	none	-	-	-	-
hbc_cs_n	output	register	-	-	hbramClk	-
hbc_ck_p_HI	output	register	normal	-	hbramClk90	Yes
hbc_ck_p_LO	output	register	normal	-	hbramClk90	Yes
hbc_ck_n_HI	output	register	normal	-	hbramClk90	Yes
hbc_ck_n_LO	output	register	normal	-	hbramClk90	Yes
hbc_rwds_OUT_HI [1:0]	output	register	normal	-	hbramClk	-
hbc_rwds_OUT_LO [1:0]	output	register	normal	-	hbramClk	-
hbc_rwds_OE [1:0]	output	register	-	-	hbramClk	-
hbc_rwds_IN_HI [1:0]	input	none	-	weak pulldown	-	-
hbc_dq_OUT_HI [15:0]	output	register	normal	-	hbramClk	Yes
hbc_dq_OUT_LO [15:0]	output	register	normal	-	hbramClk	Yes
hbc_dq_OE [15:0]	output	register	-	-	hbramClk	Yes
hbc_dq_IN_HI [15:0]	input	none	-	weak pulldown	-	-

Table 10: Interface Designer Settings for PLL

Option	Output Clock 0	Output Clock 1	Output Clock 2	Output Clock 3
Instance Name	hbramClk	hbramClk90	-	clk0
Clock Frequency	200 MHz	200 MHz	-	100 MHz
Phase Shift	0	90	-	0
Feedback Mode	-	-	-	✓

IP Manager

The Efinity® IP Manager is an interactive wizard that helps you customize and generate Efinix® IP cores. The IP Manager performs validation checks on the parameters you set to ensure that your selections are valid. When you generate the IP core, you can optionally generate an example design targeting an Efinix development board and/or a testbench. This wizard is helpful in situations in which you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core for different projects.



Note: Not all Efinix IP cores include an example design or a testbench.

Generating the HyperRAM Controller Core with the IP Manager

The following steps explain how to customize an IP core with the IP Configuration wizard.

1. Open the IP Catalog.
2. Choose **Memory Controllers > HyperRAM Controller** core and click **Next**. The **IP Configuration** wizard opens.
3. Enter the module name in the **Module Name** box.



Note: You cannot generate the core without a module name.

4. Customize the IP core using the options shown in the wizard. For detailed information on the options, refer to the *Customizing the HyperRAM Controller* section.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an Efinix® development board and/or testbench. These options are turned on by default.
6. (Optional) In the **Summary** tab, review your selections.
7. Click **Generate** to generate the IP core and other selected deliverables.
8. In the **Review configuration generation** dialog box, click **Generate**. The Console in the **Summary** tab shows the generation status.



Note: You can disable the **Review configuration generation** dialog box by turning off the **Show Confirmation Box** option in the wizard.

9. When generation finishes, the wizard displays the **Generation Success** dialog box. Click **OK** to close the wizard.

The wizard adds the IP to your project and displays it under **IP** in the Project pane.

Generated Files


The IP Manager generates these files and directories:

- **<module name>_define.vh**—Contains the customized parameters.
- **<module name>_tmpl.v**—Verilog HDL instantiation template.
- **<module name>_tmpl.vhd**—VHDL instantiation template.
- **<module name>.v**—IP source code.
- **settings.json**—Configuration file.
- **<kit name>_devkit**—Has generated RTL, example design, and Efinity® project targeting a specific development board.
- **Testbench**—Contains generated RTL and testbench files.

Customizing the HyperRAM Controller

The core has parameters so you can customize its function. You set the parameters in the General tab of the core's IP Configuration window.

Table 11: HyperRAM Controller Core Parameters (Memory Tab)

Parameter	Options	Description
Memory Operating Frequency	50, 100, 125, 133, 150, 166, 200	RAM operating frequency in MHz. Default: 200  Note: To enable the operating frequency of 250 MHz, contact the nearest FAEs.
Memory Data Width	8, 16	RAM bit width. Default: 16
Memory Size	32, 64, 128, 256	RAM size in Mb. Default: 256
Calibration Mode	PLL Auto Calibration PLL Manual Calibration Soft Logic Calibration	HyperRAM calibration mode. Default: PLL Auto Calibration
PLL Output Select	PLL Output 0, PLL Output 1, PLL Output 2, PLL Output 3, PLL Output 4	Select the PLL output clock in PLL auto and manual calibration mode. Default: PLL Output 2
Double Data Rate Input Mode	RESYNC	Indicate the DDIO register mode. Default: RESYNC
User Interface	AXI Native	Select the user interface. Default AXI
AXI3 Data Width	128, 64, 32	AXI data width in bit. Applicable to AXI3 interface. Default: 128
AXI3 AWR Channel Words	16, 32, 64, 128, 256, 512, 1024	AXI AWR channel FIFO depth. Applicable to AXI3 interface. Default: 16
AXI3 R Channel Words	16, 32, 64, 128, 256, 512, 1024	AXI read channel FIFO depth. Applicable to AXI3 interface. Efinix recommends that you to set this parameter to at least double the size of the maximum AXI burst length. For example, if burst length is 256 in the data transaction (io_arw_payload_len[7:0] == 8'h255), set the AXI3 R Channel Words to at least 512. Default: 256
AXI3 W Channel Words	16, 32, 64, 128, 256, 512, 1024	AXI write channel FIFO depth. Applicable to AXI3 interface. Default: 256
Native Data Width	128, 64, 32	Native data width in bit. Applicable to native interface. Default: 32

Parameter	Options	Description
Write Buffer Width	16, 32, 64, 128, 256, 512, 1024	Write buffer depth in bit. Applicable to native interface. ⁽³⁾ Default: 256
Read Buffer Width	16, 32, 64, 128, 256, 512, 1024	Read buffer depth in bit. Applicable to native interface. Default: 256

Table 12: HyperRAM Controller Core Parameters (Memory Register Tab)

Parameter	Options	Description
Initial Latency Count	3 Clocks, 4 Clocks, 5 Clocks, 6 Clocks, 7 Clocks	Define initial latency count. Default: 7 Clocks
Output Drive Strength	19 Ohms, 22 Ohms, 27 Ohms, 34 Ohms, 46 Ohms, 67 Ohms, 115 Ohms	Define output drive strength. Default: 34 Ohms
Hybrid Burst Enable	Hybrid, Legacy	Enable the hybrid burst. Default: Legacy
Wrap Burst Length	16 Bytes, 32 Bytes, 64 Bytes, 128 Bytes	Define wrap burst length. Default: 32 Bytes This setting must be the same as the AXI burst transaction size in wrap mode.
Clock Type	Single-Ended	Define the master clock type. Default: Single-Ended
Partial Array Refresh	None, Full Array, Bottom 1/2 Array, Bottom 1/4 Array, Bottom 1/8 Array, Top 1/2 Array, Top 1/4 Array, Top 1/8 Array	Define the partial array refresh. Default: Full Array

⁽³⁾ See [Native User Interface](#) on page 14 for more information about setting the buffer depth.

HyperRAM Controller Example Design

You can choose to generate the example design when generating the core in the IP Manager Configuration window. Compile the example design project and download the **.hex** or **.bit** file to your board. To generate example design, the **Optional Signals** option must be enabled.



Important: Efinix tested the example design generated with the default parameter options only.

Efinix provides the following four example designs:

Table 13: HyperRAM Controller Example Designs

All example designs uses the PLL Auto calibration mode. The same design can be used to test out the PLL Manual calibration mode via **Virtual I/O Debugger Settings** on page 28. However, the PLL auto calibration logic is still present in the design.

Example Design	Target	Calibration Mode
Ti60F225_devkit_axi_pll_auto	Titanium Ti60 F225 Development Board	PLL Auto
Ti60F225_devkit_native_pll_auto	Titanium Ti60 F225 Development Board	PLL Auto
Ti60F100_SiP_axi_pll_auto	Ti60 F100S3F2 FPGA	PLL Auto
Ti60F100_SiP_native_pll_auto	Ti60 F100S3F2 FPGA	PLL Auto

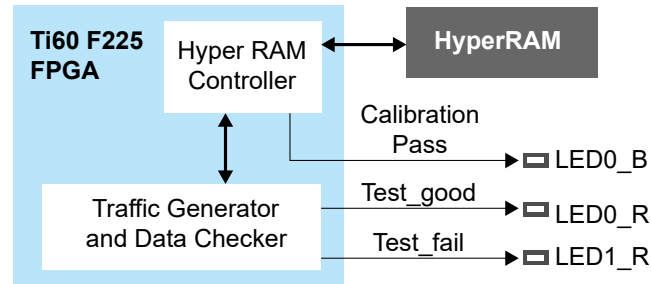
Table 14: Example Design Project Files

File	Description
top.v	Example design top-level wrapper.
ed_encrypt.v	Parameterized of the encrypted HyperRAM Controller file. This file is used by default in the generated example design.
<user_given_ip_name>.v	Generated encrypted HyperRAM Controller file based on user configuration in Efinity IP Manager. Comment-out the EFX_IPM switch in the top.v (line 16) to compile the example design. The example design is meant to target the default IP Manager settings for both AXI and native interface.
efx_ed_hyper_ram_axi_tc.v	Traffic generator and checker for AXI user interface.
efx_ed_hyper_ram_native_tc.v	Traffic generator and checker for native user interface.
efx_crc32.v	CRC32 module used by the traffic generator and checker.
debug_top.v	Verilog file for EFX debug module.
efx_clk_monitor.v	Verilog file for clock estimator module.
debug_profile.json	Virtual I/O Debugger core file. Load this file in the EfinityVirtual I/O Debugger to customize the example design. See Virtual I/O Debugger Settings on page 28.

Titanium Ti60 F225 Development Board

The design performs the memory continuous write and read check at 200 MHz HBRAM Clock. When you run the example design, you should expect the LED0_B to light up indicating that the calibration is passed and LED0_R keeps blinking to indicate that the memory operation is performed correctly. If there is read mismatch, the LED0_R stops blinking while LED1_R lights up to indicate a failed test.

Figure 11: Example Design Block Diagram

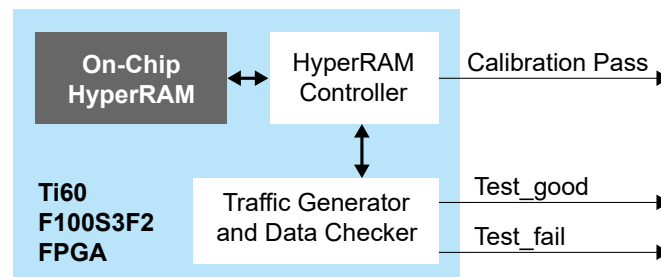


Ti60 F100S3F2 FPGA

This example performs the same function as the one for the Titanium Ti60 F225 Development Board and works in any board using Ti60 F100S3F2 FPGA. The design has the following features:

1. Uses the internal HyperRAM memory inside the package F100S3F2 FPGA.
2. Uses the internal oscillator (intosc) clock to generate the user clock and hyperram clock.
3. You can always monitor the test status via the Efinity® Debugger Virtual I/O feature that is embedded in the example design.

Figure 12: Ti60 F100S3F2 Example Design Block Diagram



Virtual I/O Debugger Settings

The example design includes Efinity Virtual I/O Debugger core for customizing and monitoring the design. The following table describes the Virtual I/O sources and descriptions.



Learn more: Refer to the Debug Perspective: Virtual I/O section of the [Efinity Software User Guide](#) for more information.

Table 15: Virtual I/O Sources and Probes

Name	Width	Radix	Value	Description
s0_Mode	1	Bin	0	Set to 1'b0 to use PLL Auto Calibration mode (Default). Set to 1'b1 to use PLL Manual Calibration mode.
s1_ShiftCtrl_en	1	Bin	0	Set to 1'b1 to enable PLL dynamic phase shift control.
s2_PLL_Shift	3	Bin	000	Total of 8 steps of the PLL dynamic phase shift settings.
p0_ShiftCtrl_en	1	Bin	0	1'b1 indicates that the PLL dynamic phase shift control is enabled.
p1_PLL_Shift	6	Bin	000011	Indicates the following settings: Bit [5:3]: Reserved, default 000. Bit [2:0]: Auto calibrated PLL dynamic phase shift settings
p2_Status	3	Bin	101	Indicates the test status: Bit 2: test_pass (Value toggles when test is passed) Bit 1: test_fail (1'b1 indicates failed test) Bit 0: Calibration pass (1'b1 indicates calibration is passed)
p3_Freq	29	Dec	241850102	Estimated hbram clock frequency. The accuracy is based on the internal oscillator frequency which can be up to $\pm 15\%$ of the desired frequency.
p4_Window_Lock	11	Bin	00001111011	Bit [11:3]: Step Pass Indicator, Bit 0 represent step 0 and up to Bit 7 which represent step 7 Bit [2:0]: Step Locked Value

HyperRAM Controller Testbench

You can choose to generate the testbench when generating the core in the IP Manager Configuration window. To generate testbench, the **Optional Signals** option must be enabled.



Note: You must include all **.v** files generated in the **/testbench** directory in your simulation.



Important: Efinix tested the testbench generated with the default parameter options only.

Efinix provides a simulation script for you to run the testbench quickly using the Modelsim software. To run the Modelsim testbench script, run `vsim -do modelsim.do` in a terminal application. You must have Modelsim installed on your computer to use this script.

The testbench simulates the example design with fixed latency mode.

Revision History

Table 16: Revision History

Date	Version	Description
April 2024	3.0	<p>Removed Data and RWDS calibration in Features. (DOC-1827)</p> <p>Updated Resource Utilization and Performance.</p> <p>Updated figures HyperRAM Controller PLL Calibration Block Diagram, HyperRAM Controller PLL Manual Calibration Block Diagram, and HyperRAM Controller Soft Logic Calibration Block Diagram (change to hbc_cal_debug_info[23:0]).</p> <p>Added Bit [23:16] are reserved in table HyperRAM Controller Ports (hbc_cal_debug_info[15:0]), HyperRAM Controller Core Parameters (Memory Tab), and Virtual I/O Sources and Probes.</p> <p>Updated topic PLL Auto Calibration Flow and figure Calibration Flow Diagram, topic PLL Manual Calibration Flow.</p> <p>Changed sub-topic AXI Interface to Hyperbus Interface, figure name of AXI Interface Write Operation Waveform and AXI Interface Read Operation Waveform to Hyperbus Write Operation Waveform and Hyperbus Read Operation Waveform. (DOC-1825)</p>
March 2024	2.9	<p>Updated table HyperRAM Controller Native Ports. (DOC-1749)</p> <p>Added new paragraph in Native User Interface topic.</p> <p>Updated figure Native Interface Write Operation Waveform.</p>
December 2023	2.8	Removed Hybrid Sleep Enable parameter. (DOC-1600)
November 2023	2.7	Updated supported data rate. (DOC-1535)
April 2023	2.6	<p>Added note about the HyperRAM verified in hardware. (DOC-1216)</p> <p>Corrected hbc_rwds_OUT_HI and hbc_rwds_OUT_LO signal widths. (DOC-1236)</p>
March 2023	2.5	Updated description for Wrap Burst Length parameter. (DOC-1187)
February 2023	2.4	<p>Added note about the resource and performance values in the resource and utilization table are for guidance only.</p> <p>Corrected native_wr_data and native_rd_data ports widths.</p>
September 2022	2.3	<p>Added note about only one HyperRAM Controller can be instantiated for a single device. (DOC-909)</p> <p>Updated native port descriptions, added FIFO depth size calculation, and added native mode operation waveform. (DOC-846)</p>
March 2022	2.2	Corrected supported bit width to 16x and maximum double data-rate to 1000 Mbps. (DOC-748)
January 2022	2.1	<p>Added read and write operation waveforms.</p> <p>Updated resource utilization table. (DOC-700)</p>
December 2021	2.0	<p>Added support for PLL manual calibration mode.</p> <p>Added support for native user interface.</p> <p>Updated parameters supported in IP Manager.</p> <p>Updated HyperRAM Controller ports.</p> <p>Updated example design.</p>
October 2021	1.1	<p>Added note to state that the f_{MAX} in Resource Utilization and Performance, and Example Design Implementation tables were based on default parameter settings.</p> <p>Updated design example target board to production Titanium</p>