CoreSDR v4.3

Handbook

Microsemi
a Microchip company

Revision History

Date	Revision	Change
May 2020	V4	Fourth release

Table of Contents

Preface

About this Document

This handbook provides details about Microsemi® CoreSDR and how to use it.

Intended Audience

Microsemi FPGA designers using Libero® System-on-Chip (SoC) or Libero Integrated Design Environment (IDE).

Introduction

Overview

CoreSDR provides a high-performance interface to single-data-rate (SDR) synchronous dynamic random access memory (SDRAM) devices. CoreSDR accepts read and write commands using the simple local bus interface and translates these requests to the command sequences required by SDRAM devices. CoreSDR also performs all initialization and refresh functions.

CoreSDR uses bank management techniques to monitor the status of each SDRAM bank. Banks are only opened or closed when necessary, minimizing access delays. Up to four banks can be managed at one time. Access cascading is also supported, allowing read or write requests to be chained together. This results in no delay between requests, enabling up to 100% memory throughput for sequential accesses.

CoreSDR is provided with configurable memory settings (row bits and column bits) and timing parameters (CAS latency, t_{RAS}, t_{RC}, t_{RFC}, t_{RCD}, t_{RP}, t_{MRD}, t_{RRD}, t_{REFC}, t_{WR}). The memory settings are configured through the SmartDesign GUI. The timing parameters are configured by setting the CoreSDR inputs. This ensures compatibility with virtually any SDRAM configuration.

The top-level interface diagram is shown in Figure 1.

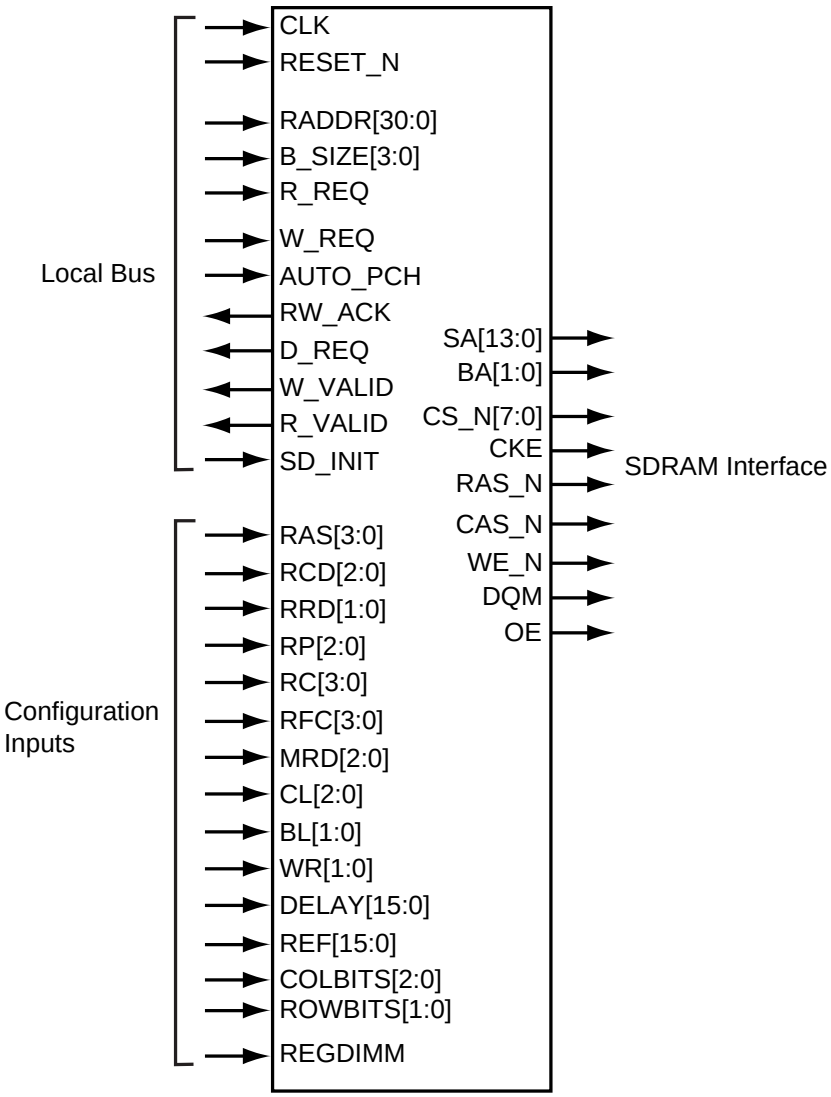

Figure 1 I/O Signal Diagram

Key Features

- High performance, single data rate controller for standard SDRAM chips and dual in-line memory modules (DIMMs)
- Synchronous interface, fully pipelined internal architecture
- Supports up to 1,024 MB of memory
- Bank management logic monitors status of up to 8 SDRAM banks.

Core Versions

This handbook applies to CoreSDR v4.3. The release notes provided with the core list known discrepancies between this handbook and the core release associated with the release notes.

Supported Families

- Axcelerator®
- RTAX™-S
- ProASIC®3
- ProASIC®3E
- ProASIC®3L
- ProASICPLUS®
- Fusion®
- SmartFusion®
- SmartFusion®2
- IGLOO®
- IGLOO®e
- IGLOO®2
- IGLOOPLUS®
- RTG4™
- PolarFire
- PolarFire SoC

Device Utilization and Performance

CoreSDR has been implemented in several Microsemi device families. A summary of the implementation data is listed in Table 1.

Table 1 CoreSDR Device Utilization and Performance

Family	Logical Elements		Utilization		Performance
	Sequential	Combinatorial	Device	Total	
ProASIC3	492	858	A3P250-2	21%	133 MHz
ProASIC3E					
Fusion	492	858	AFS250-2	21%	133 MHz
SmartFusion2	462	1147	M2S050T-FG484	1.43%	178.3 MHz
IGLOO2	462	1147	M2GL050T-FG484	1.43%	178.3 MHz
RTG4	473	1125	RTG4150-CG1657M	0.53%	141.2 MHz
PolarFire	462	1093	MPF500T-1FCG1152E	0.17%	242.3 MHz
PolarFire SoC	462	1093	MPFS250T_ES	0.31 %	242.3 MHz

Note: All data was obtained using a default system configuration with the local bus tied to internal user logic and the controller configuration inputs hard-coded. CoreSDR requires 62 FPGA I/O pins when interfaced to a 32-bit SDRAM memory. All performance data was obtained under commercial (COM) conditions. RTG4 performance data was obtained under military (MIL) conditions.

Functional Block Description

CoreSDR consists of the following primary blocks, as shown in Figure 2:

1. Control and Timing Block – Main controller logic
2. Initialization Control – Performs initialization sequence after RESET_N is deactivated or SD_INIT is pulsed.
3. Address Generation – Puts out address, bank address, and chip select signals on SDRAM interface.
4. Bank Management – Keeps track of last opened row and bank to minimize command overhead.
5. Refresh Control – Performs automatic refresh commands to maintain data integrity.

For CoreSDR, the datapath is external to the core.

Block Diagram

The tristate buffer shown in Figure 2 resides in the I/O and its output enable is controlled by the core.

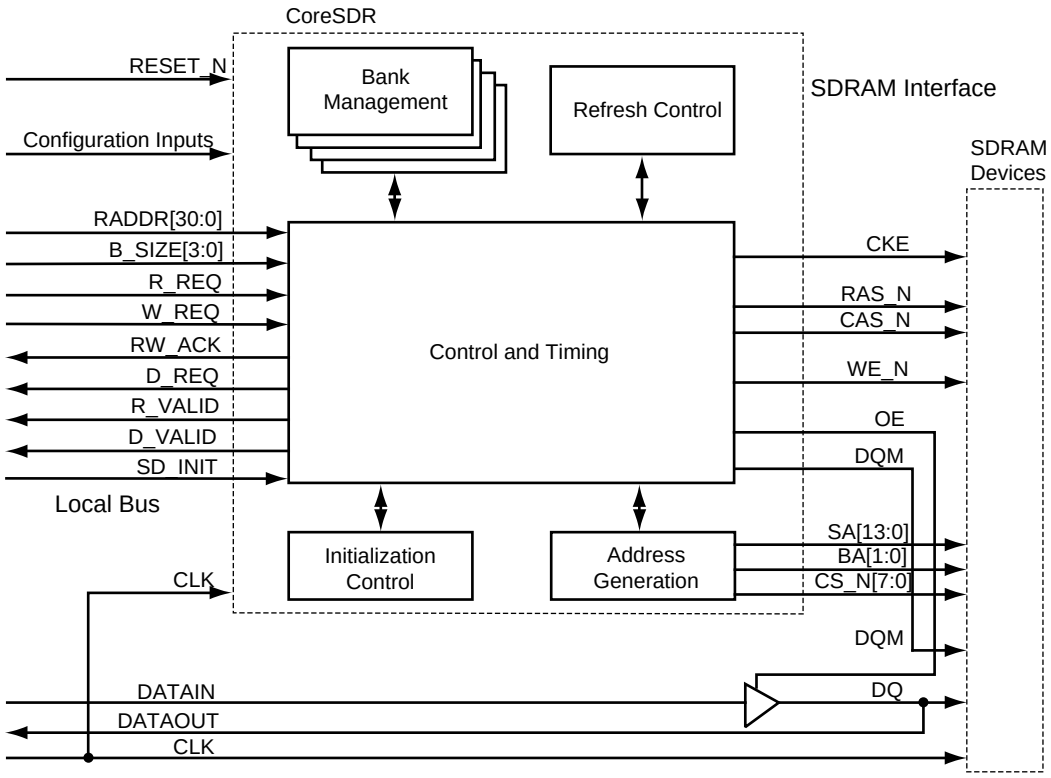

Figure 2 . CoreSDR Block Diagram

Address Mapping

The mapping of the RADDR bus at the local bus interface to the chip select, row, column, and bank addresses is shown in Figure 3. The exact bit positions of the mapping will vary depending on the rowbits and colbits configuration port settings. The column bits, bank bits, row bits, and chip select are mapped from the least significant bits of RADDR. By mapping the bank bits from this location, long accesses to contiguous address space are more likely to take place without the need for a precharge.

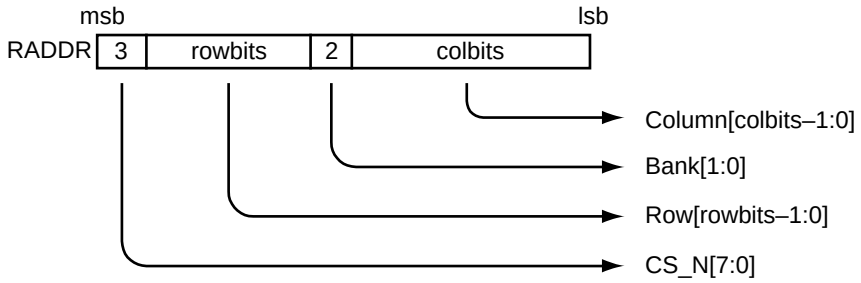

Figure 3 . Mapping RADDR

SDRAM Overview

The synchronous interface and fully pipelined internal architecture of SDRAM allow extremely fast data rates if used efficiently. SDRAM is organized in banks of memory addressed by row and column. The number of row and column address bits depends on the size and configuration of the memory.

SDRAM is controlled by bus commands that are formed using combinations of the RAS_N, CAS_N, and WE_N signals. For instance, on a clock cycle where all three signals are HIGH, the associated command is a *no operation* (NOP). A NOP is also indicated when the chip select is not asserted. The standard SDRAM bus commands are shown in Table 2.

Table 2 SDRAM Bus Commands

Command	RAS_N	CAS_N	WE_N
NOP	H	H	H
Active	L	H	H
Read	H	L	H
Write	H	L	L
Burst Terminate	H	H	L
Precharge	L	H	L
Auto-Refresh	L	L	H
Load Mode Register	L	L	L

SDRAM devices are typically divided into four banks. These banks must be opened before a range of addresses can be written to or read from. The row and bank to be opened are registered coincident with the *active* command. When a new row on a bank is accessed for a read or a write, it may be necessary to first close the bank and then reopen it to the new row. The *precharge* command closes a bank. Opening and closing banks costs memory bandwidth, so CoreSDR has been designed to monitor and manage the status of the four banks simultaneously. This enables the controller to intelligently open and close banks only when necessary.

When the *read* or *write* command is issued, the initial column address is presented to the SDRAM devices. In the case of SDR SDRAM, the initial data is presented concurrent with the *write* command. For the *read* command, the initial data appears on the data bus 1–4 clock cycles later. This is known as CAS latency and is due to the time required to physically read the internal DRAM and register the data on the bus. The CAS latency depends on the speed grade of the SDRAM and the frequency of the memory clock. In general, the faster the clock, the more cycles of CAS latency required. After the initial *read* or *write* command, sequential reads and writes will continue until the burst length is reached or a *burst terminate* command is issued. SDRAM devices support a burst length of up to eight data cycles. CoreSDR is capable of cascading bursts to maximize SDRAM bandwidth.

SDRAM devices require periodic refresh operations to maintain the integrity of the stored data. CoreSDR automatically issues the *auto-refresh* command periodically. No user intervention is required.

The *load mode register* command is used to configure the SDRAM operation. This register stores the CAS latency, burst length, burst type, and write burst mode. CoreSDR supports a *sequential burst type* and *programmed-length write burst mode*. The SDR controller only writes to the base mode register. Consult the SDRAM device specification for additional details on these registers.

In SDRAM, each bank is an organized block of rows and columns. A data width of 4, 8, or 16 is written to or read from the SDR SDRAM by providing the bank, row, and column addresses. To reduce pin count, SDRAM row and column addresses are multiplexed on the same pins. Table 3 lists the number of rows, columns, banks, and chip selects required for various standard discrete SDR SDRAM devices. The x4, x8, and x16 stand for data width. The user needs to check the SDRAM specification and set the row, column, and bank bits. CoreSDR will support any of these devices.

Table 3 Standard SDR SD RAM Device Configuration

Chip Size	Configuration	Rows	Columns	Banks
64 Mb	16M×4	12	10	4
64 Mb	8M×8	12	9	4
64 Mb	4M×16	12	8	4
64 Mb	2M×32	11	8	4
128 Mb	32M×4	12	11	4
128 Mb	16M×8	12	10	4
128 Mb	8M×16	12	9	4
128 Mb	4M×32	12	8	4
256 Mb	64M×4	13	11	4
256 Mb	32M×8	13	10	4
256 Mb	16M×16	13	9	4
512 Mb	128M×4	13	12	4
512 Mb	64M×8	13	11	4
512 Mb	32M×16	13	10	4
1,024 Mb	256M×4	14	12	4
1,024 Mb	128M×8	14	11	4
1,024 Mb	64M×16	14	10	4

SDRAM is typically available in dual in-line memory modules (DIMMs), small outline DIMMs (SO-DIMMs), and discrete chips. The number of row and column bits for a DIMM or SO-DIMM configuration can be found by determining the configuration of the discrete chips used on the module. This information is available in the module datasheet.

Operation

Initialization

After RESET_N is deasserted or SD_INIT is pulsed, CoreSDR performs the following sequence:

1. NOP command is issued for 200 µs (period controlled by the delay port parameter).
2. *Precharge-all* command
3. Eight *auto-refresh* commands
4. *Load mode register* command – This causes the SDRAM mode register to be loaded with the proper burst length (bl) and CAS latency (cl) values.

CoreSDR initialization timing is shown in Figure 4.

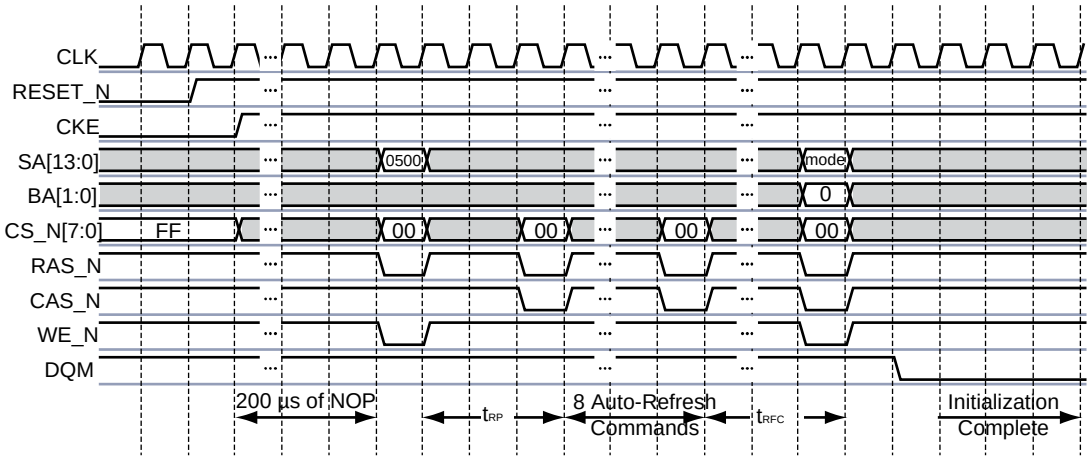

Figure 4 . SDR Initialization Sequence

Auto-Refresh

SDRAM devices require periodic *auto-refresh* commands to maintain data integrity. CoreSDR will automatically issue periodic *auto-refresh* commands to the SDRAM device(s) without user intervention. The refresh period configuration port (ref) specifies the period between refreshes, in clock cycles. Figure 5 shows an example of two refresh commands. The first refresh sequence occurs when one or more banks have been left open as a result of a *read without precharge or write without precharge* operation. All open banks are closed using the *precharge-all* command (RAS_N, WE_N asserted with sa[10] and sa[8]) prior to the refresh command. In Figure 5, a refresh occurs again after the refresh period has elapsed, as determined using the ref configuration port. The refresh will never interrupt a read or write in the middle of a data burst. However, if the controller determines that the refresh period has elapsed at a point concurrent with or prior to a read or write request, the request may be held off (RW_ACK will not get asserted) until after the refresh has been performed.

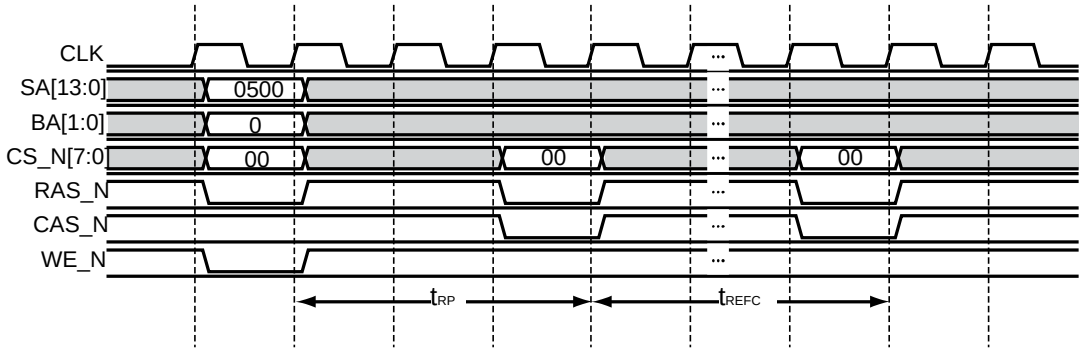

Figure 5 . Refresh Timings

Bank Management

CoreSDR incorporates bank management techniques to minimize command overhead. For each bank, the controller records the last opened row and whether the bank has been closed or not. When a local bus interface read or write request occurs, CoreSDR checks to determine if the requested bank is already opened and whether the request is for the same row as the one the bank is already opened with. If the bank is already opened with the requested row, CoreSDR performs the function immediately. If the bank is opened to a different row, the controller closes the bank (using the *precharge* command) and reopens the bank (using the *active* command) to the requested row. If the bank is already closed, the controller opens the bank to the requested row (using the *active* command).

Requests to the controller can be issued as *read with auto-precharge, write with auto-precharge, read without auto-precharge,* and *write without auto-precharge*. Commands are issued with auto-precharge if the AUTO_PCH signal is set concurrent with the read request (R_REQ) or write request (W_REQ) signals. After a read with auto-precharge or write with auto-precharge, the accessed bank is automatically closed internally by the SDRAM device(s). After a read without auto-precharge or write without auto-precharge, the accessed bank is left open until closing is required. Closing will occur whenever a request is issued to a row different from the row a bank is already open to, or during the next refresh sequence. The refresh sequence will close all the banks (using the *precharge-all* command) if all banks are not already closed.

The default configuration of the controller tracks the status of four banks at a time. This means that an access to row *a* on bank *a* on chip select a is treated differently from an access to row *a* on bank *a* on chip select *b.* Therefore, a close and open sequence is performed when switching between these rows.

CoreSDR

Generics

Customers can define the generics listed in Table 4 as required in the source code.

Table 4 CoreSDR Generics

Generic	Default Setting	Valid Values	Description
SDRAM_CHIPS	8	1 to 8	Number of chip selects
SDRAM_COLBITS	12	8 to 12	Maximum number of SDRAM column bits
SDRAM_ROWBITS	14	11 to 14	Maximum number of SDRAM row bits
SDRAM_CHIPBITS	3	1 to 3	Number of encoded chip select bits
SDRAM_BANKSTATMODULES	4	4 and 8	Number of bank status modules used (refer to Bank Management for additional information)
FAMILY	16	11 12 14 15 16 17 18 19 20 21 22 23 24 25 26 27	Must be set to match the supported FPGA family. 11 – Axcelerator 12 – RTAX-S 14 – ProASIC^{PLUS} 15 – ProASIC3 16 – ProASIC3E 17 – Fusion 18 – SmartFusion 19 – SmartFusion2 20 – IGLOO 21 – IGLOOe 22 – ProASIC3L 23 – IGLOO^{PLUS} 24 – IGLOO2 25 – RTG4 26 – PolarFire 27 – PolarFire SoC

Controller Configuration Ports

CoreSDR is configured using runtime-programmable configuration ports. These ports can be tied off by the user to fixed values or programmed at runtime. These values should not change after RESET_N is deasserted. Table 5 lists the configuration ports.

Table 5 SDRAM Controller Parameters

Parameter	Port Bits	Valid Values	Description
RAS	4	1–10	SDRAM active to precharge (t_{RAS}), specified in clock cycles
RCD	3	2–5	SDRAM active to read or write delay (t_{RCD}), specified in clock cycles
RRD	2	2–3	SDRAM active bank *a* to active bank *b* (t_{RRD}), specified in clock cycles
RP	3	1–4	SDRAM *precharge* command period (t_{RP}), specified in clock cycles
RC	4	3–12	SDRAM *active* to *active /auto-refresh* command period (t_{RC}), specified in clock cycles
RFC	4	2–14	*Auto-refresh* to *active/auto-refresh* command period (t_{RFC}), specified in clock cycles
MRD	3	1–7	SDRAM *load mode register* command to *active* or *refresh* command (t_{MRD}), specified in clock cycles
CL	3	1-4	SDRAM CAS latency, specified in clock cycles
BL	2	0–3	SDRAM maximum burst length (encoded). Values are decoded as follows: 0: 1 transfer/burst 1: 2 transfers/burst 2: 4 transfers/burst 3: 8 transfers/burst (valid for SDR only)
WR	2	1–3	SDRAM write recovery time (t_{WR})
DELAY	16	10–65,535 ns	Delay after a reset event that the controller waits before initializing the SDRAM, specified in clock cycles. Per JEDEC standards, SDR devices require this delay to be a minimum of 200 µs.
REF	16	10–65,535 ns	Period between *auto-refresh* commands issued by the controller, specified in clock cycles. REF = auto refresh interval / t_{CK}, where t_{CK} is the clock cycle in ns.
COLBITS	3	3–7	Number of bits in the column address (encoded). Values are decoded as follows: 3: 8 column bits 4: 9 column bits 5: 10 column bits 6: 11 column bits 7: 12 column bits
ROWBITS	2	0–3	Number of bits in the row address (encoded). Values are decoded as follows: 0: 11 row bits 1: 12 row bits 2: 13 row bits

Parameter	Port Bits	Valid Values	Description
			3: 14 row bits
REGDIMM	1	0–1	Set when using registered/buffered DIMMs. Causes adjustment in local bus interface timing to synchronize with SDRAM command timing delayed by register/buffer on DIMM.

For example:

If RCD is 20 ns in the specification and the clock is 133 MHz, then the number of clocks equals to 20/7.5=3. The user needs to hardcode the RCD ports to the binary value "011" in the top level since the value is "3".

If RCD is 20 ns in the specification and the clock is 100MHz, then the number of clocks equals to 20/10=2. If it's divisible without a remainder, then set it to the next integer, which is 3 in the above example, so that there will be little margin. The user needs to hardcode the rcd ports to the binary value "011" in the top level since the value is "3".

Example settings for the timing-related parameters are shown in Table 6. These settings are based on the speed grade of the SDRAM devices and the desired operating frequency. Consult the datasheet for the SDRAM device you are using for the specific timing values of that device.

Table 6 Example Controller Parameter Values for CoreSDR

Parameter	100 MHz (10 ns period)[1]		133 MHz (7.5 ns period)[2]	
	Specification	Value	Specification	Value
RAS	44.0 ns	5	37.0 ns	6
RCD	20.0 ns	3	15.0 ns	3
RRD	15.0 ns	2	14.0 ns	2
RP	20.0 ns	3	15.0 ns	3
RC	66.0 ns	7	60.0 ns	8
RFC	66.0 ns	7	66.0 ns	9
MRD	2 clks	2	2 clks	2
CL	–	2	–	2
WR	15.0 ns	2	14.0 ns	2
DELAY	200 µs	20,000	200 µs	26,667
REF	7.8125 µs	781	7.8125 µs	1,041

Notes:

1. *Values based on Micron MT48LC32M8A2-75*
2. *Values based on Micron MT48LC32M8A2-7E*

Interface Description

The port signals for CoreSDR are defined in Table 7 below, Table 8, and Table 1. The port signals are also illustrated in Figure 1. All signals are designated either input (input-only) or output (output-only).

Local Bus Signals

The user interface to CoreSDR is referred to as the local bus interface. The local bus signals are shown in Table 7.

Table 7 Local Bus Signals

Signal	Name	I/O	Description
CLK	Clock	Input	System clock. All local bus signals are synchronous to this clock.
RESET_N	Reset	Input	System reset Note: In the RTG4 device, RESET_N generates synchronous resets that propagate throughout the core.
RADDR[30:0]	Memory Address	Input	Local bus address
B_SIZE[3:0]	Burst Size	Input	Local bus burst length. Valid values are 1 through BL, where BL is the programmed burst length. (Refer to Table 5 for discussion of the BL parameter.)
R_REQ	Read Request	Input	Local bus read request
W_REQ	Write Request	Input	Local bus write request
AUTO_PCH	Auto-Precharge Request	Input	When asserted in conjunction with R_REQ or W_REQ, causes command to be issued as *read with auto-precharge* or *write with auto-precharge*, respectively.
RW_ACK	Read/Write Acknowledge	Output	Acknowledgement of read or write request
D_REQ	Data Request	Output	Requests data on the local bus write data bus (datain) during a write transaction. Asserts one clock cycle prior to when data is required.
W_VALID	Write Data Valid	Output	Frames the active data being written to SDRAM. Mimics D_REQ, except that it is delayed by one clock cycle.
R_VALID	Read Data Valid	Output	This signal is typically not used and is retained for legacy compatibility.
SD_INIT	Initialization Strobe	Input	Indicates that the data on the local bus read data bus (dataout) is valid during a read cycle.

Notes:

1. *All control signals are active high except RESET_N.*
2. *All local bus signals are synchronous to clk.*

SDR SDRAM Interface Signals

The external interface to SDRAM devices is referred to as the SDRAM interface. The SDRAM interface signals are shown in Table 8.

Table 8 SDR SDRAM Interface Signals

Signal	Name	I/O	Description
SA[13:0]	Address Bus	Output	Sampled during the active, precharge, read, and write commands. This bus also provides the mode register value during the load mode register command.
BA[1:0]	Bank Address	Output	Sampled during active, precharge, read, and write commands to determine which bank command is to be applied to.
CS_N[7:0]	Chip Selects	Output	SDRAM chip selects
CKE	Clock Enable	Output	SDRAM clock enable. Held LOW during reset to ensure SDRAM dq and dqs outputs are in the high-impedance state.
RAS_N	Row Address Strobe	Output	SDRAM command input
CAS_N	Column Address Strobe	Output	SDRAM command input
WE_N	Write Enable	Output	SDRAM command input
DQM	Data Mask	Output	SDRAM data mask asserted by controller during SDRAM initialization and during burst terminate. User may sum with user data mask bits.
OE	Output Enable	Output	Tristate control for DQ data

Timing Diagrams

SDRAM Writes

The user requests writes at the local bus interface by asserting the W_REQ signal and driving the starting address and burst size on RADDR and B_SIZE, respectively. The AUTO_PCH may also be asserted with W_REQ to cause the write to be issued as a *write with auto-precharge*.

The rules for write requests at the local bus interface are as follows:

1. Once W_REQ is asserted, it must remain asserted until RW_ACK is asserted by CoreSDR. After RW_ACK is asserted by CoreSDR, W_REQ may remain asserted to request a follow-on write transaction. The W_REQ signal may remain asserted over any number of RW_ACK pulses to generate any number of cascaded write bursts. The only time W_REQ may be deasserted is during the clock cycle immediately following the RW_ACK pulse from CoreSDR.
2. The signals RADDR, B_SIZE, and AUTO_PCH must maintain static values from the point when W_REQ becomes asserted until CoreSDR asserts RW_ACK. The RADDR, B_SIZE, and AUTO_PCH signals may only change values in the clock cycle immediately following the RW_ACK pulse from CoreSDR or when W_REQ is deasserted.
3. The W_REQ signal may not be asserted while the read request signal (R_REQ) is asserted.
4. The data request signal (D_REQ) will assert one clock prior to when the user must present data at the datain bus.
5. The timing relationship between an initial W_REQ assertion and an RW_ACK assertion, or between RW_ACK pulses as a result of multiple cascaded writes, will vary depending on the status of the banks being accessed, configuration port settings, refresh status, and initialization status. The user logic should not rely on any fixed timing relationship between W_REQ and RW_ACK.

Example CoreSDR Write Sequence

Figure 6 shows an example CoreSDR write sequence. In this sequence, two writes are requested, both to the same bank and row. The first write request is for a burst size of eight; the second is for a burst size of five. The write request signal (W_REQ) is first asserted with the starting address (RADDR) and the burst size (B_SIZE). As a result of this request, CoreSDR asserts the row address (sa), bank address (ba), and chip select (CS_N) with the *active* command to open the bank to the requested row. Next, CoreSDR requests data from the user at the local bus interface using the D_REQ signal and acknowledges the write request by asserting RW_ACK. At the next clock, CoreSDR begins writing the data to the SDRAM devices. Eight data cycles are transferred.

The local bus interface write request (W_REQ) remains asserted after RW_ACK is asserted by CoreSDR to request an additional write burst. After the RW_ACK pulse, the local bus interface changes the address and changes the burst size to five. As soon as the eight write cycles from the previous request are processed, the next write command is issued by CoreSDR, and the five data cycles for the new burst begin. In the case of this example, the second write was to the same bank and row as the first write. If the second write had been to a different bank or row, CoreSDR would have issued *precharge* and/or *activate* commands prior to the *write* command.

Since the burst size (B_SIZE) of the second write request is less than the programmed SDR SDRAM burst length, the burst must be terminated using the *burst terminate* command. CoreSDR does this automatically, asserting WE_N five clock cycles after the *write* command was issued. The dqm signal is also asserted during the *burst terminate* command to comply with JEDEC specifications.

As demonstrated in Figure 6, the output enable signal (OE) goes active one clock cycle before the data is actually required at the dq outputs. This is to accommodate long clock to out delays that may exist in PLD or ASIC tristate drivers. The OE signal stays asserted until the last data element is written.

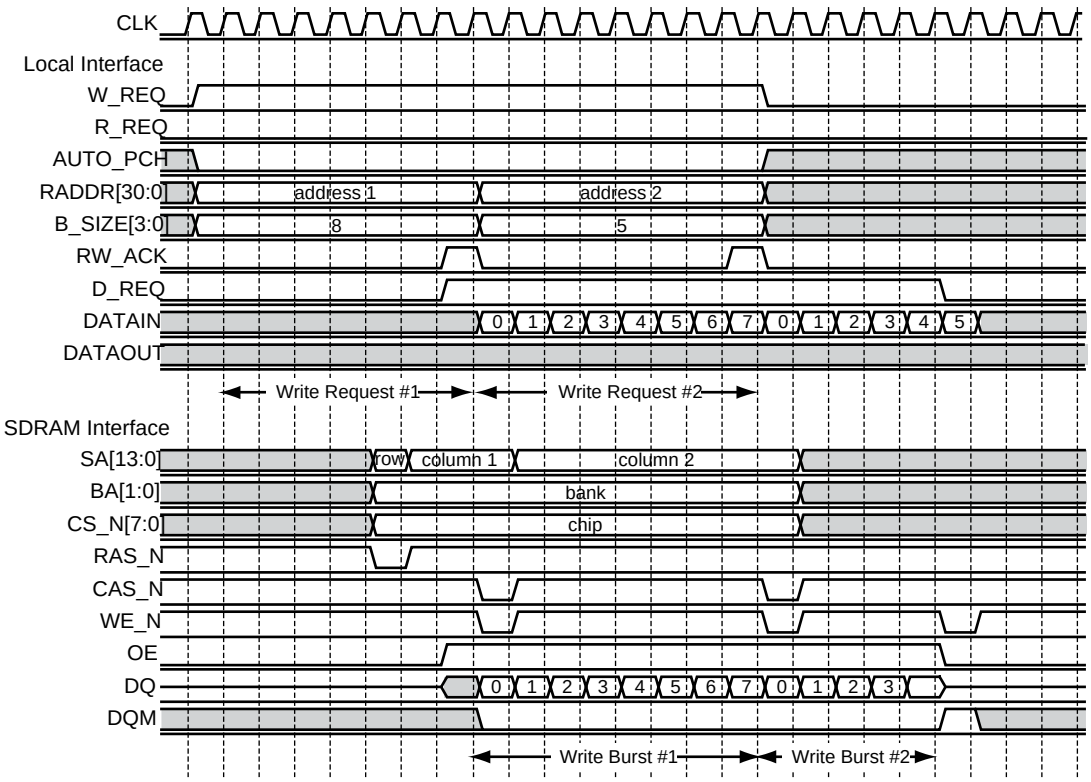

Figure 6 . CoreSDR Burst Write

Note: For the case shown, RCD = 3, BL = 2, and REGDIMM = 0

SDRAM Reads

The user requests reads at the local interface by asserting the R_REQ signal and driving the starting address and burst size on RADDR and B_SIZE, respectively. The AUTO_PCH may also be asserted with R_REQ to cause the read to be issued as a *read with auto-precharge.*

The rules for read requests at the local interface are as follows:

1. Once R_REQ is asserted, it must remain asserted until RW_ACK is asserted by CoreSDR. After RW_ACK is asserted by CoreSDR, R_REQ may remain asserted to request a follow-on read transaction. R_REQ may remain asserted over any number of RW_ACK pulses to generate any number of cascaded read bursts. The only time R_REQ may be deasserted is during the clock cycle immediately following the RW_ACK pulse from CoreSDR.
2. The signals RADDR, B_SIZE, and AUTO_PCH must maintain static values from the point when R_REQ is asserted until CoreSDR asserts RW_ACK. RADDR, B_SIZE, and AUTO_PCH may only change values in the clock cycle immediately following the RW_ACK pulse from CoreSDR, or when R_REQ is deasserted.
3. The R_REQ signal may not be asserted while the write request signal (W_REQ) is asserted.
4. The read data valid signal (R_VALID) will assert when valid data is available at the dataout bus.
5. The timing relationship between an initial R_REQ assertion and an RW_ACK assertion, or between RW_ACK pulses as a result of multiple cascaded reads, will vary depending on the status of the banks being accessed, configuration port settings, refresh status, and initialization status. The user logic should not rely on any fixed timing relationship between R_REQ and RW_ACK.

Example CoreSDR Read Sequence

Figure 7 shows an example CoreSDR read sequence. In this sequence, two reads are requested, both to the same bank and row. The first read request is for a burst size of eight, the second is for a burst size of five. The read request signal (R_REQ) is first asserted with the starting address (RADDR) and the burst size (B_SIZE). As a result of this first request, CoreSDR asserts the row address (SA), bank address (BA), and chip select (CS_N) with the *active* command to open the bank to the requested row. Next, CoreSDR acknowledges the read request by asserting RW_ACK. Since the CAS latency is set at two in this case, read data appears at dataout two clock cycles after CoreSDR issues the *read* command. CoreSDR asserts the R_VALID signal to indicate valid data at the dataout bus. Eight data cycles are transferred.

The local bus interface read request (R_REQ) remains asserted after RW_ACK is asserted by CoreSDR to request an additional read burst. After the RW_ACK pulse, the local bus interface changes the address and changes the burst size to five. CoreSDR issues the next read command to the SDRAM devices as soon as is possible to avoid interruption in the data flow. In the case of this example, the second read was to the same bank and row as the first read. If the second read had been to a different bank or row, CoreSDR would have issued *precharge* and/or *activate* commands prior to the *read* command. Since the burst size (B_SIZE) of the second read request is less than the programmed SDR SDRAM burst length, the burst must be terminated using the *burst terminate* command. CoreSDR does this automatically, asserting WE_N five clock cycles after the read command was issued.

CoreSDR never asserts the dqm signal while read data is transacting. User logic must never assert DQM while read data is transacting, as this will cause the SDRAM devices to drive high-impedance instead of valid data.

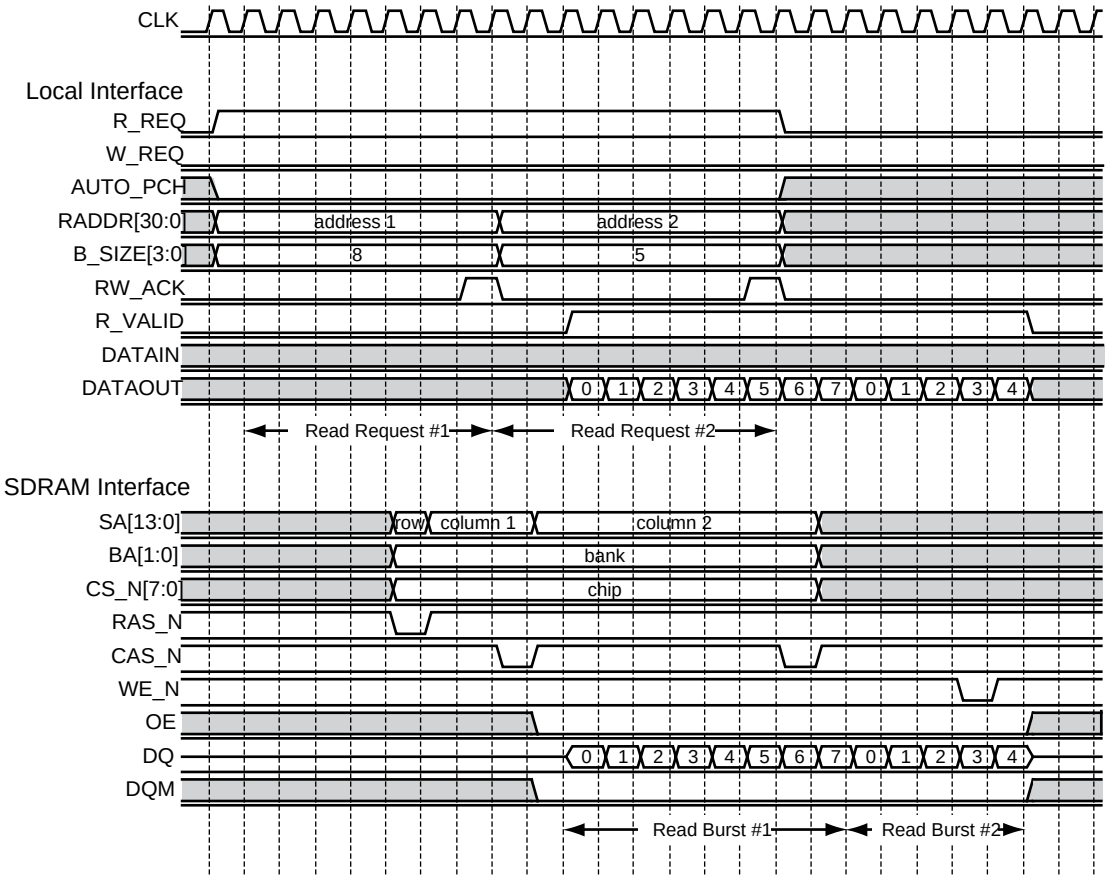

Figure 7 . CoreSDR Burst Read

Note: For the case shown, BL = 3, RCD = 3, WR = 2, RP = 3, CL = 2, and REGDIMM = 0.

Auto-Precharge

Read commands can be issued to the SDRAM devices as *read with auto-precharge* or *read without auto-precharge*. Likewise, *write* commands can be issued to the SDRAM devices as *write with auto-precharge* or *write without auto-precharge*. If the auto-precharge option is used, the SDRAM device will automatically close (precharge) banks being read from or written to at the end of the transaction. Any subsequent reads or writes to these banks will not require an explicit *precharge* command from CoreSDR. The user selects whether *read* or *write* commands are issued with auto-precharge using the AUTO_PCH signal. If AUTO_PCH is asserted along with W_REQ or R_REQ, the command will be issued to the SDRAM with auto-precharge. The auto-precharge option is useful in situations where the requested read or write addresses tend to be random. With random address sequences, banks are seldom left open with the exact row required by a subsequent request. If the auto-precharge was not used for the previous access to a bank, subsequent transactions to that bank first require the bank to be closed (precharged), causing a delay in the transaction. If auto-precharge was used for the previous access, the bank is already closed and ready to be opened to the desired row. Figure 8 shows example transactions using the auto-precharge feature for CoreSDR. In the example, two write requests are issued, the first using auto-precharge and the second not using auto-precharge. Both requests are to the same bank, but may be to different rows. CoreSDR issues the first command as a *write with auto-precharge* by driving bit sa[10] HIGH during the *write* command. CoreSDR issues the second command as a *write without auto-precharge* by driving bit sa[10] LOW during the *write* command. Since the first and second requests are to the same bank, CoreSDR must wait before reopening the bank to meet the SDRAM t_{WR} and t_{RP} requirements. If the first and second requests were to different banks, the second request would follow the first request such that there would be no interruption in data flow.

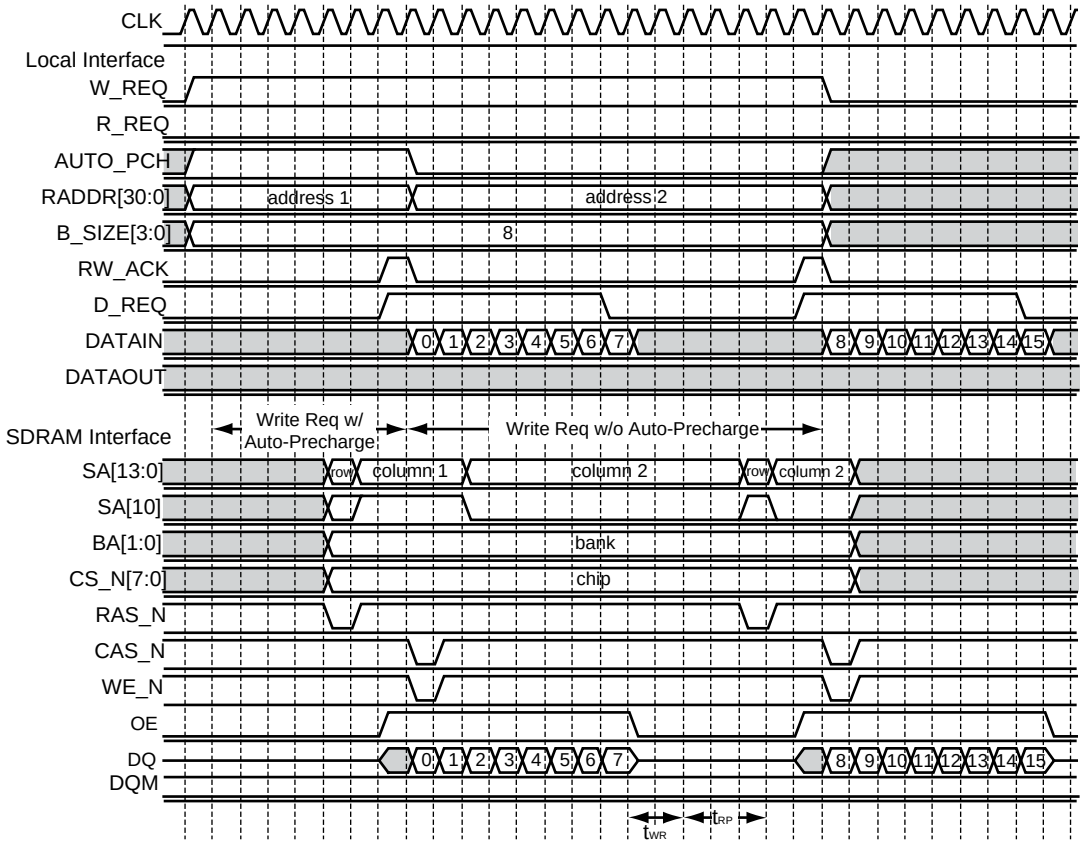

Figure 8 . CoreSDR Burst Writes with Auto-Precharge Option

Note: For the case shown, BL = 3, RCD = 3, WR = 2, RP = 3, CL = 2, and REGDIMM = 0.

Tool Flows

Licenses

This IP core is available for free with the Libero software.

Smart Design

The core can be configured using the configuration GUI within SmartDesign, as shown in Figure 9, Figure 10, and Figure 11.

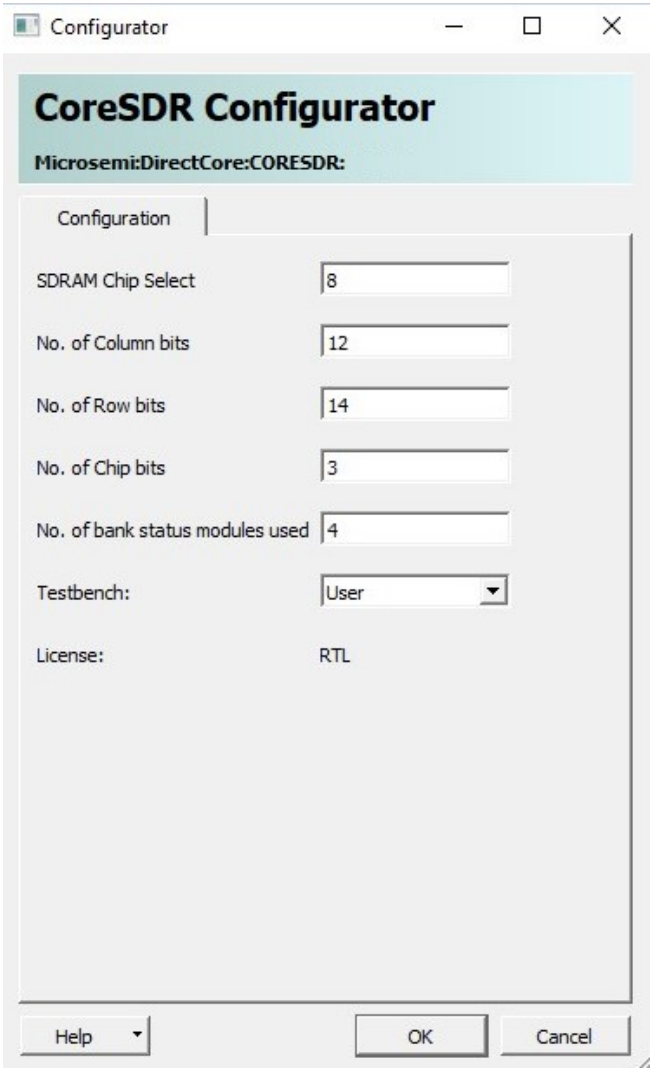

Figure 9 . CoreSDR Configuration within SmartDesign with User Testbench

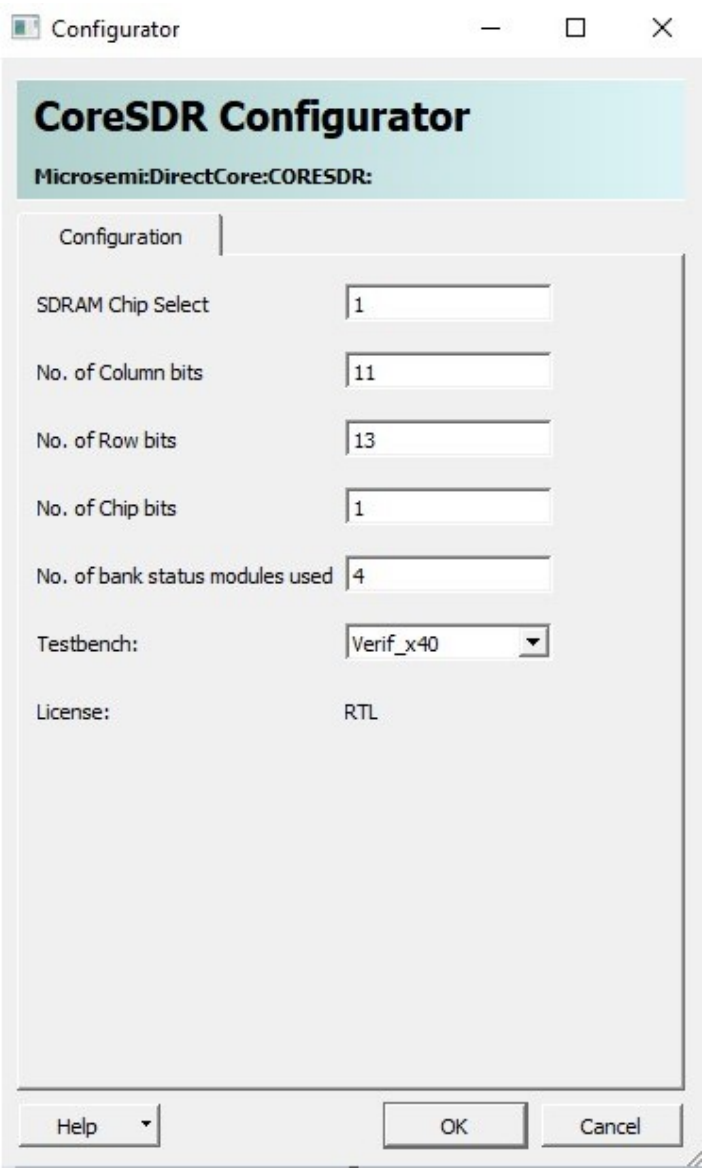

Figure 10 . CoreSDR Configuration within SmartDesign and Verif_x40 Testbench

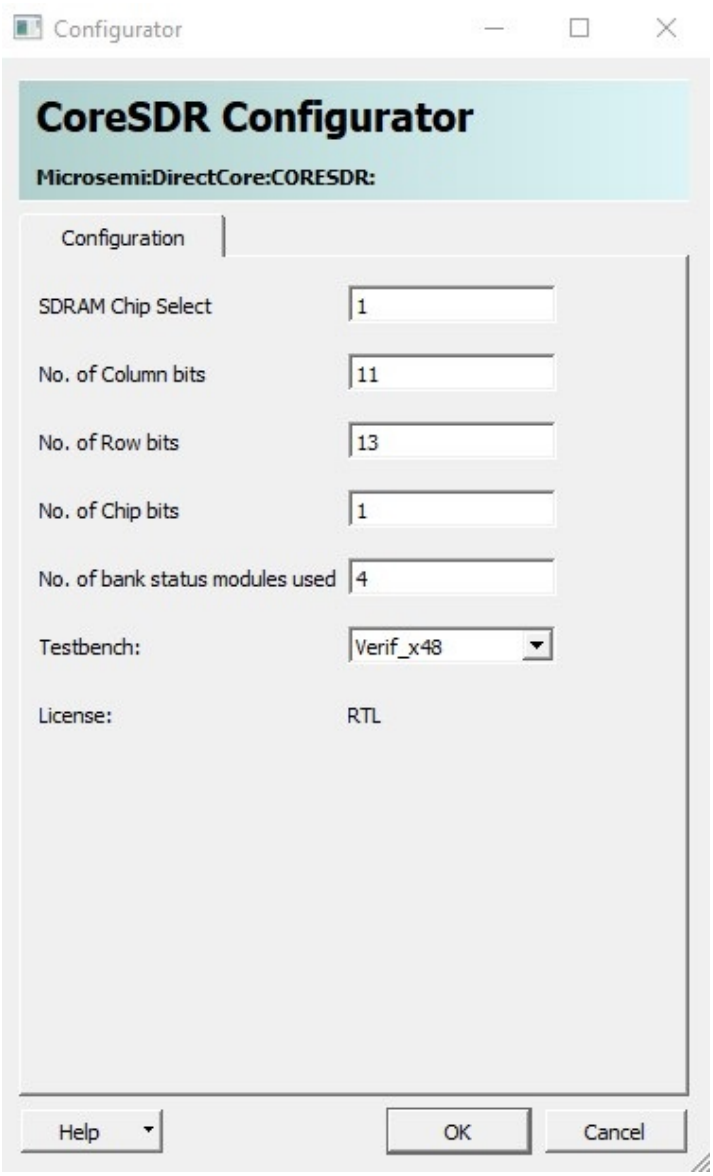

Figure 11 . CoreSDR Configuration within SmartDesign with Verif_x48 Testbench

Simulation Flows

To run simulations, select the user testbench or Verif testbench within the SmartDesign CoreSDR configuration GUI, right-click, and select **Generate Design** (Figure 2).

When SmartDesign generates the design files, it will install the appropriate testbench files. To run the simulation, set the design root to the CoreSDR instantiation in the Libero design hierarchy pane, and click the **Simulation** icon in the Libero Design Flow window. This will invoke Model*Sim*® and automatically run the simulation.

Synthesis in Libero

Set the design root appropriately and click the Synthesis icon in the Libero. The synthesis window appears, displaying the Synplicity® project. Set Synplicity to use the Verilog 2001 standard if Verilog is being used. To perform synthesis, click the **Run** icon.

Place-and-Route in Libero

Having set the design route appropriately and run Synthesis, click the **Layout** icon in Libero to invoke Designer. CoreSDR requires no special place-and-route settings.

Testbench Operation

Four testbenches are provided with CoreSDR:

- Verilog User Testbench
- VHDL User Testbench
- Verilog Verif Testbench
- VHDL Verif Testbench

Testbench Description

Included with RTL releases of CoreSDR are user testbench and Verilog/VHDL Verif testbench that gives an example of CoreSDR usage. A simplified block diagram of the testbench is shown in Figure 10. By default, the Verilog version, *tb_user.v*, instantiates a Micron 256 Mbit SDRAM model (*MT48LC16M16A2.v*, 4M×16 × 4 banks). The VHDL version, *tb_user.vhd*, instantiates a Micron 64 Mbit SDRAM model (*MT48LC4M16A2.vhd*, 1M×16 × 4 banks). The Verilog/VHDL version, tb_user_X40.v instantiates a Micron 512 Mbit SDRAM model (16Meg x 8x 4 banks) with five instances. The Verilog/VHDL Verif version, tb_user_X48.v instantiates a Micron 512 Mbit SDRAM model (16Meg x 8 x 4 banks) with 6 instances. The testbench instantiates the DUT (design under test), which is the CoreSDR macro, the SDRAM model, as well as the test vector modules that provide stimuli sources for the DUT. A procedural testbench controls each module and applies the sequential stimuli to the DUT

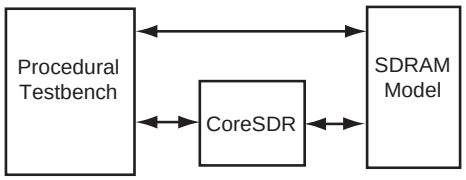

Figure 12 . CoreSDR Testbench

Verilog User Testbench

The Verilog user testbench is provided as a reference and can be modified to suit your needs. The source code for the Verilog user testbench is provided to ease the process of integrating the CoreSDR macro into your design and verifying its functionality.

VHDL User Testbench

The VHDL user testbench is provided as a reference and can be modified to suit your needs. The source code for the VHDL testbench is provided to ease the process of integrating the CoreSDR macro into your design and verifying its functionality.

Verilog Verif Testbench

The Verilog Verif testbench is provided as a reference and can be modified to suit your needs. The source code for the Verilog Verif testbench is provided to ease the process of integrating the CoreSDR macro into your design and verifying its functionality.

VHDL Verif Testbench

This testbench provides support through verilog models and VHDL DUT as mixed mode simulation. This testbench is provided as a reference and can be modified to suit as per your needs.

Microsemi Headquarters
One Enterprise, Aliso Viejo, CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
email: sales.support@microsemi.com
www.microsemi.com

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services.

Learn more at www.microsemi.com.