XAPP1321 (v1.0) October 23, 2017

Fast Calibration and Daisy Chaining Functions in DDR4 Memory Interfaces

Authors: John Schmitz, Parth Joshi, and Santosh P

Summary

This application demonstrates how to achieve a much faster DDR4 calibration time (ten-times faster) and how to preserve the content in the DDR4 memory during partial or full reconfiguration to enable daisy chaining functions in the Xilinx® UltraScale™ and UltraScale+™ devices. As we push parallel interfaces to faster and faster data rates our calibration schemes and time have increased exponentially. This increased calibration time means that you have to waste valuable time waiting for the memory to be ready to operate. Daisy chaining functions come into play when there is a desire to load a bit file into the FPGA and memory content into DDR4 memory where it can be stored until the next bit file is ready to use that data. This application note shows you how to quickly load calibration data into the DDR4 Memory Controller and how to keep the content in the memory valid even if the FPGA is reconfigured. This application note also provides considerations for system-level implementation for the preservation of external DRAM content during reconfiguration.

Download the reference design files for this application note from the Xilinx® website. For detailed information about the design files, see Reference Design.

Fast Training and Preservation of Content

The high bit rates of DDR4 (up to 2,667 Mb/s in UltraScale+ devices) have necessitated the use of complex training algorithms to optimally tune the interface for maximum performance. This can take longer than desired for some applications. In many applications, however, an initial full training can be done followed by a fast mini-training each time the interface is reset or restarted, which results in a much lower startup time.

Preservation of the DDR4 memory content is useful for many applications. These range from a desire to achieve maximum power savings with the FPGA completely powered down to applications where the FPGA is reconfigured but the DDR4 memory data must be preserved to store intermediate values to be used by a subsequent new function. The latter is frequently referred to as *daisy chaining functions*.

Both of these use cases make use of the ability of the Xilinx DDR4 memory interface training algorithm to save its values and its ability to use the saved value to decrease the time needed for training. See *UltraScale Architecture-Based FPGAs Memory IP v1.4* (PG150) [Ref 1] for detailed information on this save/restore process. The training and calibration data can be saved either into a static region inside the FPGA (if using partial reconfiguration) or stored externally off the FPGA (for quick startup or a full reconfiguration). The process puts the training data into known register locations which can be extracted by the user design. This data can be pushed back into the same registers and a flag can be set to instruct the training algorithm to skip the full training process. The training automatically makes some adjustments because the voltage and temperature might have varied since the initial full training was completed. These adjustments are much faster than the initial training process.

The DRAM is put into the self-refresh mode via a command sequence from the memory controller. Minimal interaction with the host controller is needed when in this state. Clocks can be turned off and most command and address pins are ignored. This is a lower power operating mode which preserves the contents of the DRAM. When the controller decides it wants to access the DRAM contents, it goes through another sequence to return the DRAM to normal operating mode. Typically the self-refresh mode is used to save power when no access is needed to the DRAM.

The self-refresh feature of the DRAM opens up some additional applications when used with an FPGA. In particular this can allow the use of full and partial reconfiguration of the FPGA without losing the contents of the DRAM, which can be critical in some applications. With full FPGA reconfiguration the need for the self-refresh support is clear because the soft logic-based memory controller cannot continue to function during this reconfiguration process. Less obvious but quite important is the need for support for partial reconfiguration. The solution of simply putting the memory controller in a static region seems straightforward, but this can create routing challenges inside the device that are difficult to surmount. Thus there is a strong need to be able to leave the memory controller in a region that will be reconfigured, yet be able to preserve the DRAM contents. Information on partial reconfiguration for Xilinx FPGAs can be found in the *Vivado® Design Suite User Guide: Partial Reconfiguration* (UG909) [Ref 2].

DDR4 DRAM Component Self-Refresh Mode

The DDR4 DRAM self-refresh feature is documented in the JEDEC specification JESD79-4B and on the memory vendors' data sheets. To summarize, the SRE command causes entry into the self-refresh mode, and the SRX command is used to exit self-refresh. The CK, ODT, command, and address are all *don't cares* after the DRAM enters into the self-refresh mode. CKE must be held Low. See Figure 146 in the JEDEC JES79-4B specification. RESET_n must also be held High.

DDR4 RDIMM Self-Refresh Mode

The process of putting the DDR4 RDIMM into self-refresh is similar to the method for DDR4 DRAM, however, it has different requirements for use with full or partial reconfiguration. The register clock driver (RCD) on the RDIMM requires that it be put into the clock stop power down mode prior to allowing the clock input pins to float. Additionally, the RCD automatically drives the CKE pin Low to the DDR4 DRAM components when in this mode. The clock stop power down mode is achieved by driving both legs of the clock (ck_t and ck_c) Low after entering the self-refresh mode. The RCD chip has internal pull-down resistors on ck_t and ck_c, but these inputs must be driven Low prior to allowing them to float. In this case the FPGA option of PUDC (pull-up during configuration) should not be used as it might override the weak pull-down of the RCD and cause the RCD chip to exit the clock stop power down mode. See the JEDEC DDR4 RCD01 and DDR4 RCD02 specifications for more information on the clock stop power down mode.

Board-Level Considerations

Handling of External Signals to DRAM

The critical external signals to the DDR4 DRAM are CKE and RESET_n. The RDIMM requires special handling of the ck_t and ck_c signals as well. Treatment of these signals depends on the configuration of the DRAM (component, DIMM) and the reconfiguration mode (full, partial). There is no single method of handling these signals that is best for all systems. This application note describes several possibilities with the pros and cons of each to allow you to choose what is best for your particular design.

RESET_n

RESET_n must be held Low during power-up, and is held Low by the controller at the start of the initialization process and is then driven High. RESET_n must remain High after this to ensure that the DRAM does not get reset which would cause data corruption. During a full reconfiguration of the FPGA, and a partial reconfiguration where the I/O bank(s) of the controller are located, the FPGA pins float (although PUDC can be used to have a light pull-up). See the *Configuration Pin Definitions* table in the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 3] for more information. Typically a 4.7 kΩ resistor is used to pull down the RESET_n pin which ensures that the DRAM is held in reset during power-up. Unfortunately, this pull-down resistor causes the DRAM to be reset if the FPGA is reconfigured unless specific handling is used. RESET_n is a CMOS signal that is asynchronous to the DRAM clock.

Partial Reconfiguration

The easiest solution for this case is to route the RESET_n signal from the DDR4 memory controller core to the static region and then out to a pin. The static region contains the logic that handshakes with the memory controller and knows when self-refresh has been entered. In this case, the static logic can contain an OR gate between the memory controller's RESET_n output and the OBUF on the FPGA that drives out to the RESET_n of the controller. Care should be taken that this override signal is set to `0` during the initial configuration and operation of the FPGA. Subsequently, this override signal can be set after the self-refresh state is entered. This forces the RESET_n output to be High and not interfere with the DRAM self-refresh operation while the reconfigure occurs. This technique is illustrated in Figure 1.

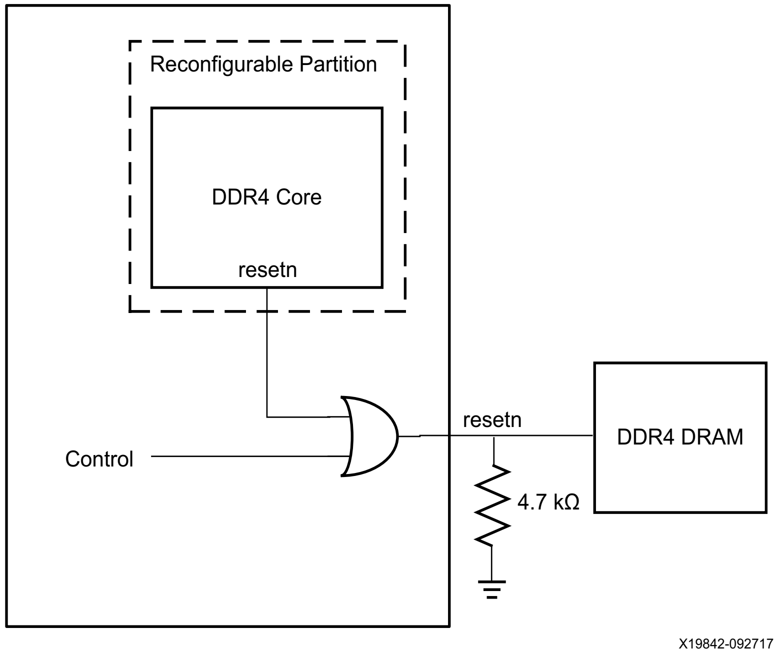


Figure 1: **Partial Reconfiguration Solution**

If the RESET_n signal from the DDR4 controller core cannot be routed to the static region, external logic is needed to force the RESET_n High during the self-refresh state. This system logic is responsible for activating this reset override signal when the controller core indicates self-refresh is activated, and is also responsible for deactivating the reset override when self-refresh is exited. The system logic driving the reset override can be either external from or internal to the FPGA, as shown in Figure 2.

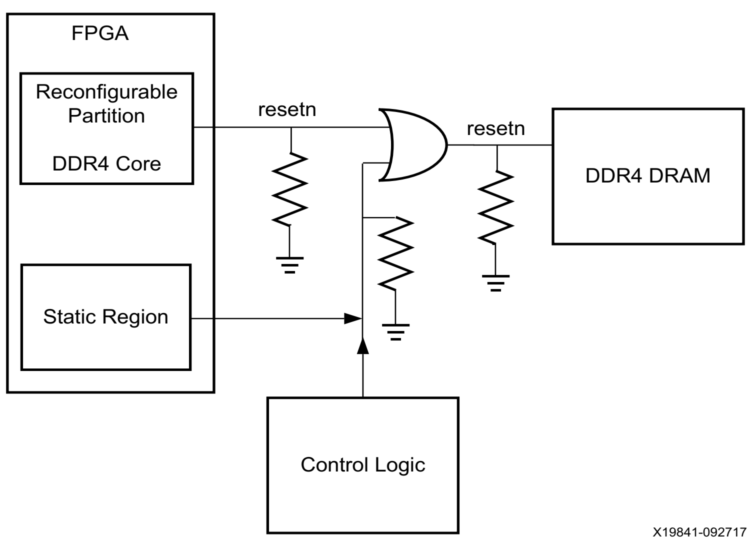


Figure 2: **Partial Reconfiguration Solution with External Logic**

Full Reconfiguration

In the full reconfiguration case, the RESET_n override must occur externally to the FPGA. The system logic is responsible for activating this reset override signal when the controller core indicates self-refresh is activated, and the system logic is also responsible for deactivating the reset override when self-refresh is exited, as shown in Figure 3.

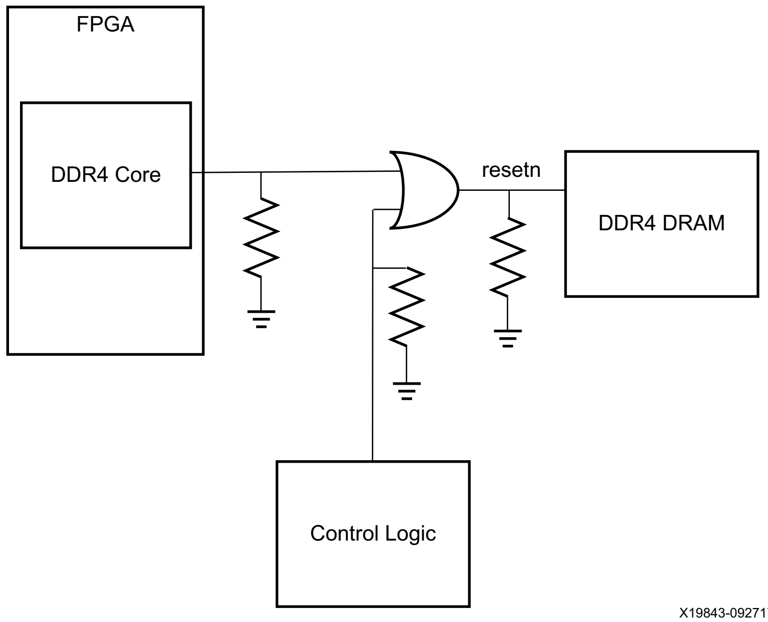


Figure 3: **Full Reconfiguration Solution**

CKE for DDR4 Component and Unbuffered UDIMM and SODIMM

The CKE signal is more challenging than the RESET_n because it is a high-speed command signal to the DRAM that must meet setup and hold times relative to the clock. Additionally, it normally has a relatively low impedance termination (such as 40Ω or 50Ω) to the V_{TT} supply (0.6V in the DDR4 case). The driver from the controller is normally configured for SSTL12 signaling. The CKE signal, along with the other command, control, and address signals is compared against the reference pin by the DDR4 DRAM component V_{REFCA}. V_{REFCA} is set to the midpoint of the V_{DD} supply which is also 0.6V. Figure 4 shows the typical CKE topology.

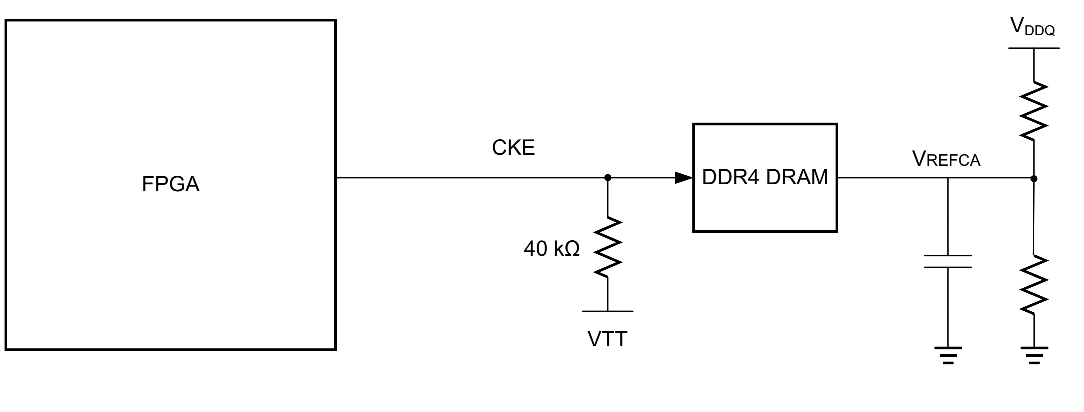


Figure 4: **Typical CKE Topology**

The strong termination to V_{TT} is challenging because this is equal to the V_{REFCA} which means that the CKE signal is very close to its logic threshold when the FPGA output is disabled (high-Z). A small amount of noise in the system causes CKE to been seen as a High-level signal, which can cause the DRAM to exit self-refresh inadvertently and thus cause data corruption. This is an unacceptable situation in the default configuration for use with reconfiguration of the FPGA.

Unlike the RESET_n solution where a static region can be used, CKE has a high-speed timing requirement that cannot be satisfied by placing it in a bank that is static and not reconfigured. The timing skew is too high between banks for this to be acceptable. There are multiple solutions to the CKE issue depending on the system components and design trade-offs available. Note that CKE does not need special treatment for RDIMM cases (see ck_t and ck_c for RDIMM for more information).

Control via Active CKE Pull-Down

One solution is to pull down CKE directly. This technique uses a transistor and resistor to pull down the CKE signal after self-refresh is entered. Because this directly affects the CKE signal quality, care should be taken to ensure it has the least possible signal integrity impact. CKE operates as an SDR signal operating at the same frequency as the memory clock. Figure 5 shows a sample implementation.

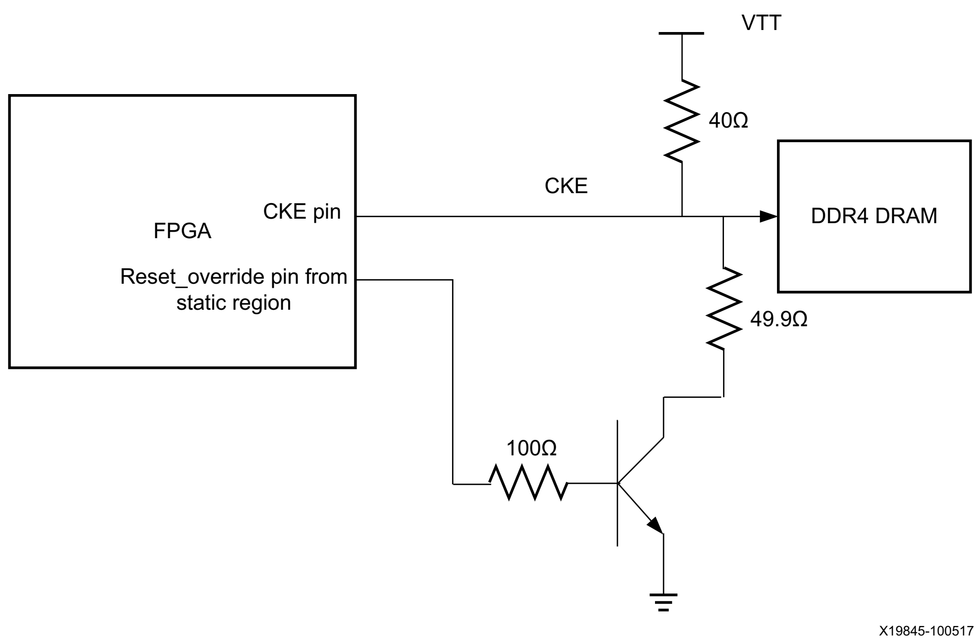


Figure 5: **Control via Active CKE Pull-Down**

This technique is best used in component systems where the resistor and transistor can be placed very close to the 40Ω termination to V_{TT}. Simulations must be used to confirm the suitability of this technique for the specific board topology. Unbuffered DIMMs and SODIMMs have a longer stub on the DIMM card so this technique is not recommended with their use.

Control via the V_{TT} Power Supply

The strong termination to V_{TT} is what pulls the CKE signal to the incorrect voltage region when the FPGA outputs are disabled (High-Z) during reconfiguration. If the V_{TT} supply is lowered or turned off during self-refresh, the CKE signal could stay in the correct low voltage during this time. Many V_{TT} regulators do have an enable pin and can be used for this purpose. An example is shown in Figure 6. This example uses the TPS51200 regulator from Texas Instruments. Pin 7, the "EN" pin is the enable.

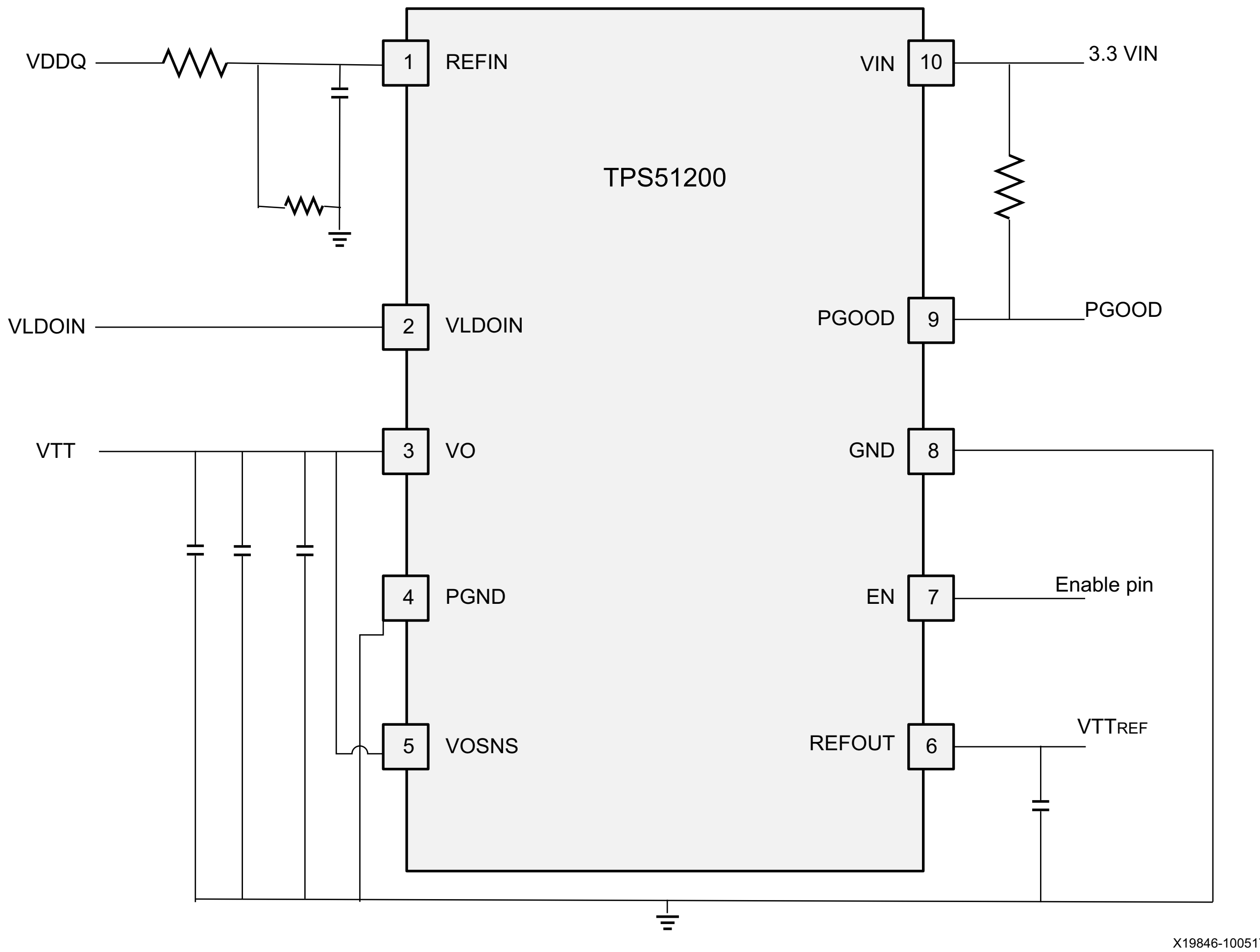


Figure 6: **Control via the V_{TT} Power Supply**

You must take into account the time that it takes the regulator to shut down after the self-refresh is entered and before the FPGA is reconfigured. You must ensure that the V_{TT} regulator has restored V_{TT} prior to the release of the reset to the DRAM controller core when reconfiguration is complete. You must ensure that the regulator turns on and off without significant under or overshoot. The output of the regulator must be held Low when it is disabled (can be accomplished through the use of an external pull-down resistor). The Texas Instruments TPS51200 controls the turn on and turn off ramps and pulls down the output when disabled.

You must also ensure that the V_{TT} supply and the V_{REFCA} are *not* supplied by the same regulator. This is because CKE (and other command/address/control signals) are referenced against the

V_{REFCA} signal. In this case, the V_{REFCA} signal must be supplied by some other source such as a resistive divider. A typical resistive divider is shown in Figure 7.

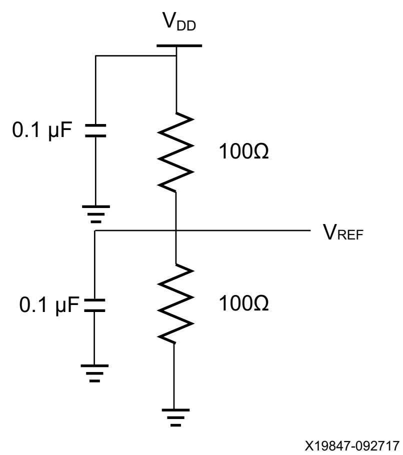


Figure 7: **Typical V_{REF} Resistive Divider**

ck_t and ck_c for RDIMM

In RDIMM applications there is a requirement for the RCD chip on the RDIMM to enter the clock *stop power down* mode prior to allowing the clock pins to the RCD to float as will happen during the reconfiguration process. The DDR4 memory controller IP must ensure that both ck_t and ck_c are driven Low after entering self-refresh. In addition, the memory controller IP must restart the clock properly after reconfiguration has exited.

Your FPGA design must not set the PUDC (*pull-up during configure*) because this will interfere with the weak pull-downs inside the RCD device that are needed to keep the clock pins Low during reconfiguration.

Vivado Design Tools Flow with Memory Interface Generator IP Design

The next few sections describe the process to enter self-refresh, save the calibration data, exit self-refresh mode, and restore the calibration data. The partial reconfiguration flow and its advantages are also explained in detail.

Xilinx Programmable Logic DDR4 Controller Self-Refresh Support

The Xilinx PL DDR4 controller supports the self-refresh feature in conjunction with *calibration save and restore*. This is a multi-step process that not only puts the DRAM in self-refresh mode, but saves the calibration data so a quick exit from self-refresh into normal operation is possible. The process for entering self-refresh is as follows:

1. The user design halts traffic to the memory controller.

2. The user design issues a self-refresh request to the memory controller on the app_sref_req port.
3. The controller acknowledges the save request after flushing out pending DRAM transactions and causing the DRAM to enter the self-refresh mode with the app_sref_ack signal.
4. The user design copies the calibration data from the controller to either a static region (for partial reconfiguration) or off the FPGA (for full reconfiguration).

When in self-refresh, the CKE of the DRAM must be held Low, and the RESET_n signal must be maintained High. These are the critical external signals of interest for this application (discussed under Board-Level Considerations).

At this point in the sequence a full or partial reconfiguration can occur, and as long as the CKE and RESET_n signals are not disturbed at the DRAM, the DRAM stays in self-refresh mode and data integrity is maintained. To exit self-refresh mode and resume operation, the following sequence is necessary:

1. Within 50 memory controller clock cycles after the user interface reset signal is deasserted, the mem_init_skip and app_restore_en signals must be asserted and stay asserted until the fast calibration cycle completes (indicated by the init_calib_complete signal). Signal mem_init_skip directs the calibration process to skip the usual DRAM initialization process, and signal app_restore_en informs the controller that it will use the stored data instead of running a full (and destructive to the contents of the DRAM) calibration process.
2. The user design copies the stored calibration data from either the static partition or from off-chip memory to the controller.
3. The user design indicates the copy of the calibration data is complete by asserting signal app_restore_complete.
4. The controller exits self-refresh of the DRAM and uses the retrieved calibration data to perform a fast calibration process.
5. Completion of the fast calibration process is indicated by assertion of signal init_calib_complete.
6. The user design deasserts signals mem_init_skip and app_restore_en.
7. The user design can now send transactions to the memory controller as desired with the previous data preserved.

More information on the Xilinx implementation of the self-refresh in the controller can be found in the DDR3/4 Core Architecture section of *UltraScale Architecture-Based FPGAs Memory IP v1.4* (PG150) [Ref 1].

Save-Restore Feature to Save Calibration Time

The save-restore feature is very useful to dramatically shorten the training time and does not require the use of the partial reconfiguration flow. This feature is supported through the standard memory interface generator (MIG) IP example design and is enabled by selecting the save-restore feature which generates the top-level ports. Refer to the Save Restore section of *UltraScale Architecture-Based FPGAs Memory IP v1.4* (PG150) for port descriptions. The calibration data can be stored in block RAM in the same FPGA or outside of the FPGA in onboard memory. The choice depends on the application. This application note provides an example of storing calibration data in the block RAM on the same FPGA. Refer to the associated reference design. The reference design has a state machine in the `example_top` module which drives the save-restore sequence. For save-restore features the state machine can be run on the same clock as the DDR4 interface. In the reference design it is running off another clock from the static region because the design is used for both save-restore and self-refresh features. There are no hardware considerations required for this feature. Refer to the Save-Restore section of *UltraScale Architecture-Based FPGAs Memory IP v1.4* (PG150) [Ref 1] for more information.

Memory initialization is done during the save-restore process. The main advantage of this feature is to save calibration time as opposed to going through full calibration. It restores previously calibrated data back to Xilinx System Debugger (XSDB) block RAM and only does the DQS gate tracking stage of calibration. The disadvantage is the additional storage required and some added logic.

The TCL script provided with the reference design demonstrates the steps in the save-restore feature. The reference design with this application note is tested in hardware using Vivado Design Suite version 2016.4. For detailed steps refer to the associated `readme` file.

Partial Reconfiguration

The advantages of the DDR4 self-refresh feature with partial reconfiguration:

- Reduces calibration time
  - Save-restore in conjunction with self-refresh features reduce calibration time to ~50 ms
- Saves power
  - Self-refresh commonly is used to save DRAM power when unused
  - Self-refresh along with partial reconfiguration of the FPGA reduces both DRAM and FPGA power use
- Preserves data integrity
- Provides partial reconfiguration flexibility to route the additional logic using the partial reconfiguration flow

Partial Reconfiguration Design Generation

A block diagram of partial reconfiguration design generation is shown in Figure 8.

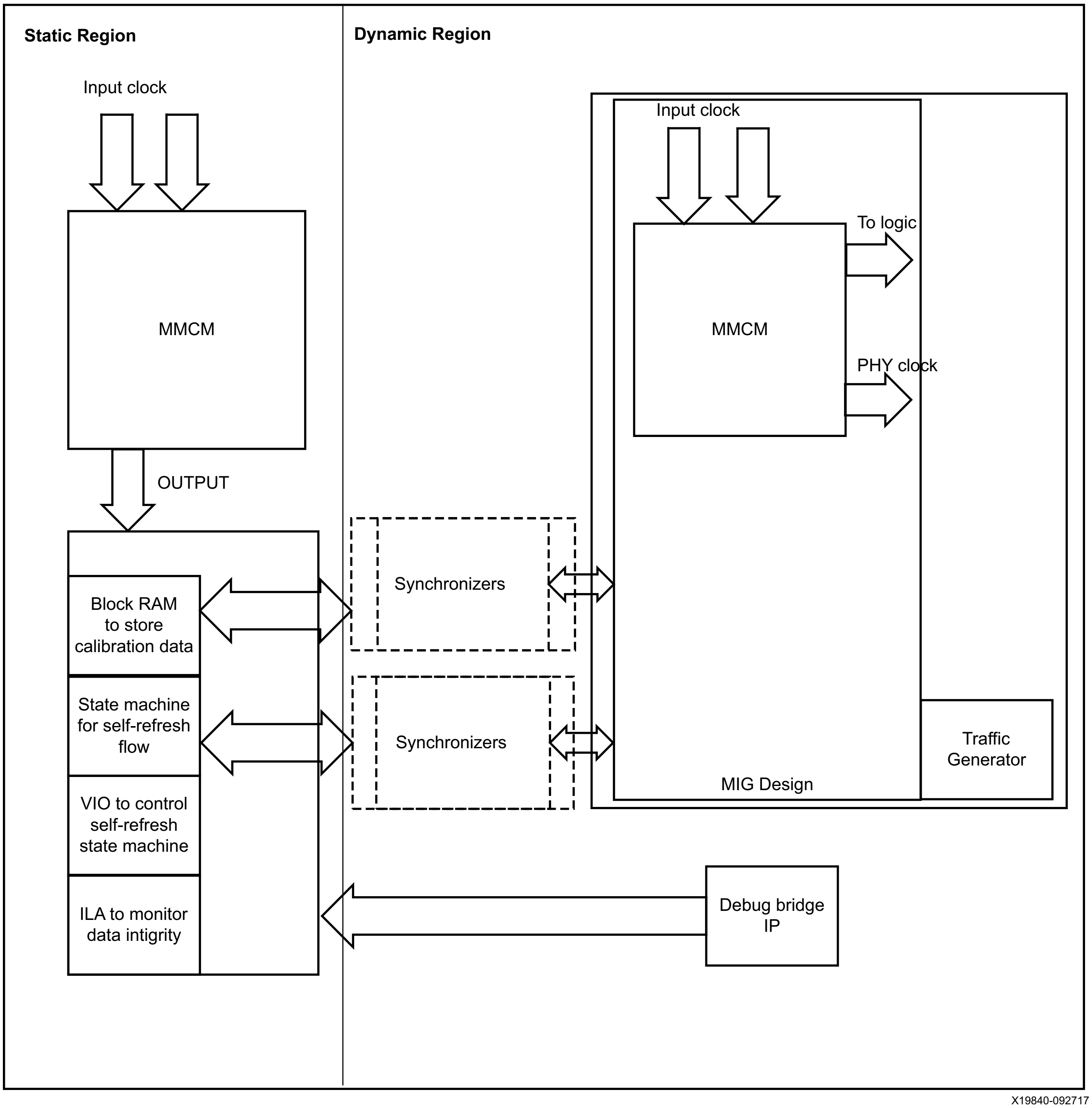


Figure 8: **Partial Reconfiguration Design Generation Block Diagram**

As shown in the block diagram, there are two regions in the design. The static region has the logic which is preserved during FPGA reprogramming. The dynamic region can be reprogrammed during FPGA configuration. Based on the application and logic requirement, both the dynamic and static region can be floorplanned. This can be done using p-blocks. In this design the dynamic region has p-blocks and the rest of the device is used as a static region.

Note: The Vivado Design Suite provides the ability to hierarchically divide the design into smaller, more manageable physical blocks (p-blocks). P-blocks can include logic modules and primitive logic from anywhere in the design.

Partial Reconfiguration Flow Software Considerations

Partial reconfiguration flow software considerations are as follows:

- Clocks cannot be driven out from the dynamic region
- Clocks can be driven from the static region to the dynamic region
- Partial reconfiguration flow implementation with MIG DDR4 IP requires Vivado Design Suite 2016.3 or later in batch mode, or Vivado Design Suite v2017.1 or later in project mode.

MIG DDR4 IP Generation Requirements

The following steps are required to generate a MIG IP design:

1. The self-refresh feature must be enabled in the MIG GUI when creating a MIG design, as shown in Figure 9.

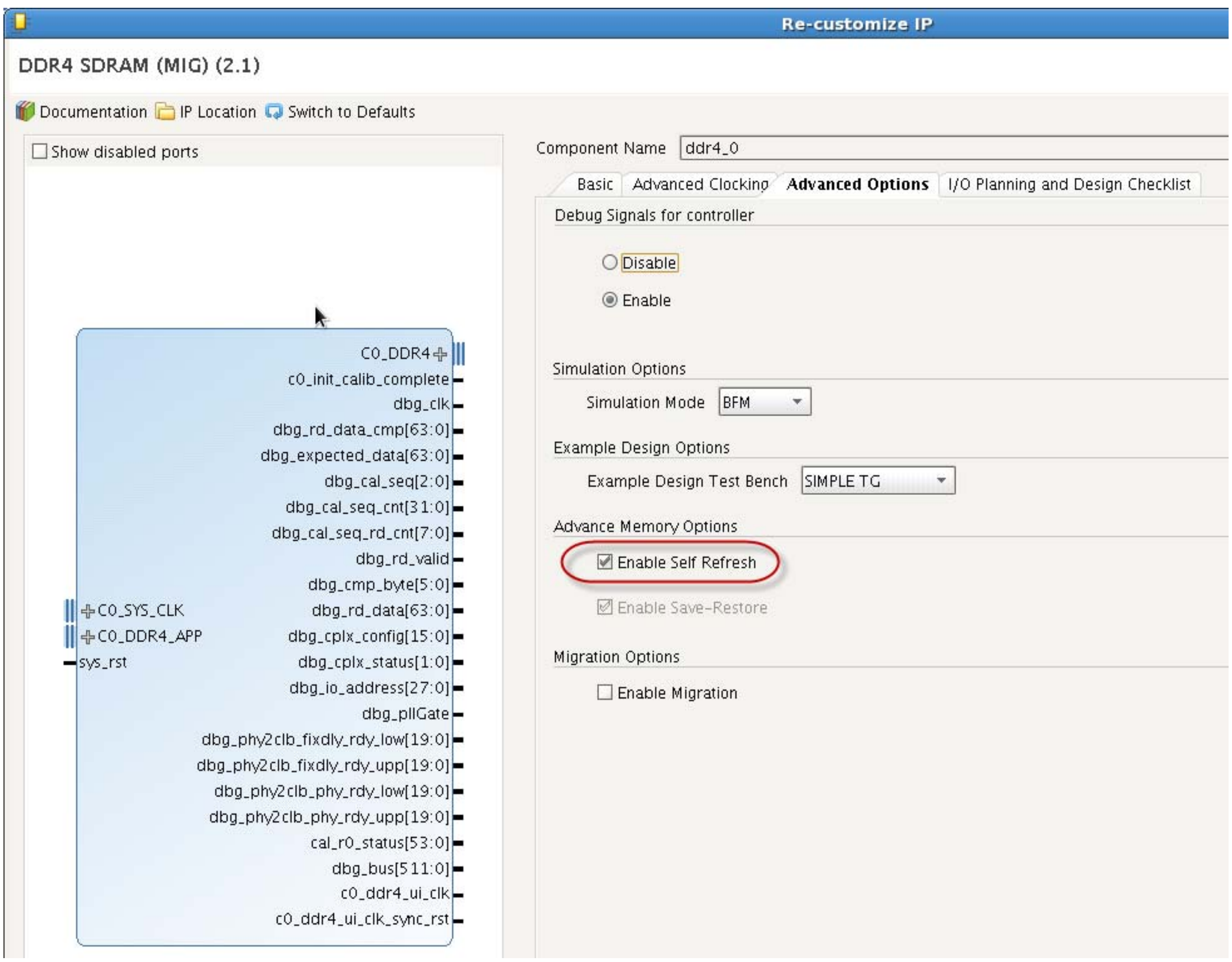


Figure 9: **MIG GUI**

2. The MIG IP needs to synthesized as shown in Figure 10.

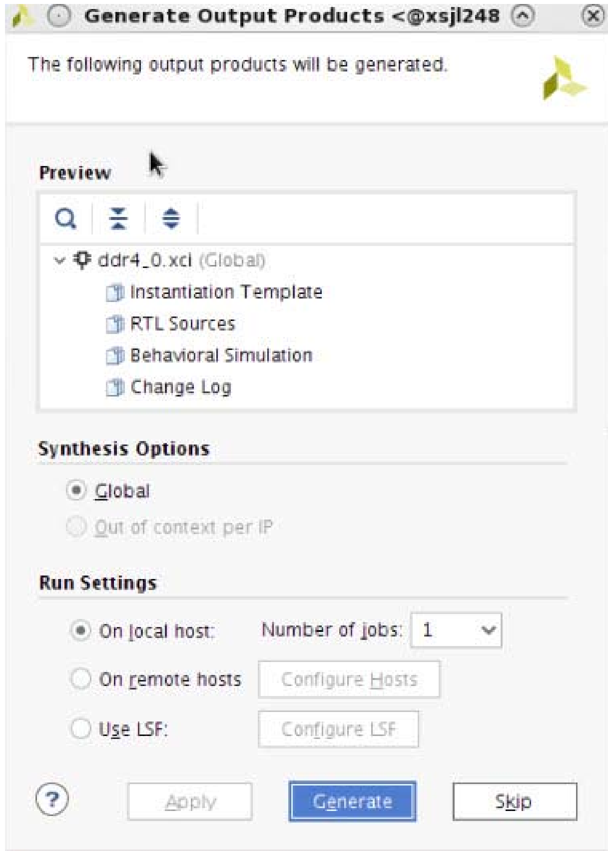


Figure 10: **Synthesis Options**

3. Debug bridge IP instantiation is not required for Vivado Design Suite 2017.1 or later (refer to the provided design in project mode for Vivado Design Suite 2017.1). However you must instantiate debug_bridge IP in the dynamic region for Vivado Design Suite 2016.3 and 2016.4:
    a. Bring all of the ports to the top-level module in the static region.
    b. Connect the static region clock with the clk port of the IP.
4. Refer to the provided reference design for port connections. The Vivado design suite automatically makes all other required connections.

Design Architecture

The reference design demonstrates how these techniques could be used in an actual system. The example design shows how the data can be preserved in DRAM through the self-refresh process while partial reconfiguration occurs with the memory controller. The reference design has been tested in hardware using Vivado Design Suite 2017.1 in project mode and 2017.2 in non-project mode. The example design performs the following sequence of events:

1. Releases reset to the memory controller and lets it calibrate normally
2. Writes a pattern to memory

3. Causes the memory controller to enter self-refresh
4. Stores the data from calibration
5. Pauses for some time
6. Reconfigures the memory controller region
7. Restores the saved calibration values to the memory controller
8. Causes the memory controller to exit self-refresh and use the saved calibration values to re-enable access to the DRAM
9. Reads back from the DRAM to verify that the data written originally is preserved
10. Indicates success or failure of the data verification

The reference design top-level module wraps and controls the lower level modules, and runs the sequence of events. Additionally, it creates and stores data through the memory controller, and checks the data when received from the controller after the completion of the reconfigure and the exit of self-refresh. The example top module is located in the static region and contains these modules:

- MMCM – generates the clock for the static region This clock needs to be running at the same frequency as the XSDB block RAM clock in the MIG IP.
- Block RAM – stores calibration data from the dynamic region's XSDB block RAM.
- State machine – controls the self-refresh entry/exit cycle.
- VIO – controls self-refresh state machine and traffic generator controls.
    - The Virtual Input/Output (VIO) core is a customizable core that can both monitor and drive internal FPGA signals in real time.
- ILA – monitors for data compare errors and calibration complete signals.
    - The customizable Integrated Logic Analyzer (ILA) IP core is a logic analyzer core that can be used to monitor the internal signals of a design.
- Performance counter – provides time to restore after the self-refresh exit.
- Internal Configuration Access Port (ICAP) instance ICAPE2 – monitors the partial reconfiguration process.

The dynamic region contains these modules:

- Synchronizer – drives signals between the static and dynamic regions to prevent clock-domain-crossing (CDC) issues.
- MIG DDR4 IP design (out of context per IP (OOC) synthesized option) – provides solutions for interfacing with DDR4 SDRAM memory types.
- Traffic generator (from the MIG IP example design; traffic generator inputs controlled through the VIO). Traffic generator is instantiated in the example design (`example_top.sv`) to drive the memory design through the application interface. Refer to *UltraScale Architecture-Based FPGAs Memory IP v1.4* (PG150) [Ref 1] for more information about the traffic generator and its capabilities.

- Debug bridge IP (in tandem with field updates and partial reconfiguration solution) – user selectable mode From_BSCAN_to_Debug adds a debug bridge instance in each reconfigurable module which would connect to debug cores such as ILA, VIO, Memory IP, and JTAG2AXI.

The reference design has a state machine in the `example_top` module (see reference design) that drives the self-refresh feature. During the self-refresh process, memory initialization is bypassed. This is done to preserve data integrity when exiting from self-refresh and normal operation to DRAM resumes. The main advantage of this feature is to save power and calibration time. It restores previously calibrated data back to XSDB block RAM and only performs the DQS gate tracking stage of calibration. It also preserves data integrity so data written to memory before entering the self-refresh cycle remains as it is after exiting from the self-refresh cycle.

Non-project Mode Flow

The Partial Reconfiguration flow with the MIG IP design is currently supported with non-project mode in Vivado Design Suite 2016.3 or later. The MIG IP needs to be either global or OOC-synthesized. Refer to the associated reference design for the detailed directory structure.

The project files with `*.prj` file extension contain all of the XCI, Verilog, and System Verilog files from both the static and dynamic regions. The *black box* Verilog file `*_bb.v` contains all of the ports from the top-level dynamic region module. This module is instantiated in the top-level static module.

If changes are desired or there is a need to add new modules to this design, the project `*.prj` files need to be updated. If there are new ports added in the dynamic region, the *black box* Verilog file `*_bb.v` needs to be updated.

To run the full implementation, locate the `design.tcl` file and run this command:

```
vivado -mode batch -source design.tcl
```

This initiates the complete flow and generates three bit files along with the associated clear bit files:

- Bit file – used with full design static and reconfigurable modules
- Partial clear bit file – prepares the reconfigurable region for the next reconfigurable modules (required only for UltraScale family devices, not required for UltraScale+ family devices)
- Partial bit file – used with the reconfigurable module

The TCL script provided with the reference design performs the steps to validate the DDR4 self-refresh entry/exit cycle. For detailed steps refer to the reference design `readme` file.

Project Mode Flow

The Partial Reconfiguration flow with the MIG IP design is supported with project mode in Vivado Design Suite 2017.1 or later. The MIG IP must be synthesized as Global through Vivado

Design Suite 2017.2. OOC synthesis for IP within Reconfigurable Modules is supported in Vivado Design Suite 2017.3 and later. Refer to the associated reference design for the detailed directory structure.

Self-Refresh Entry Cycle

The following steps initiate the self-refresh cycle:

1. Request signal app_sr_Req is asserted.
2. Acknowledge signal app_sr_ack is asserted.
3. After data from XSDB block RAM is saved into static region block RAM, signal xsdb_bram_save_complete transitions from Low to High, as shown in Figure 11.

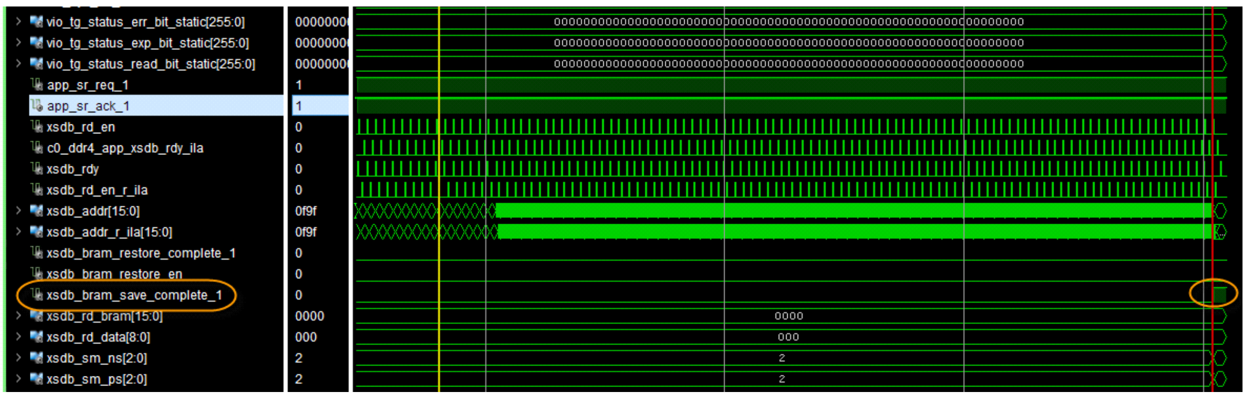


Figure 11: **Signal xsdb_bram_save_complete Transition**

Self-Refresh Exit Cycle

The following steps initiate the self-refresh exit cycle:

1. Memory initialization must be avoided to preserve data integrity and exit the self-refresh cycle. Therefor signal hw_init_skip_en must be asserted.
2. Assert signal xsdb_bram_restore_en within 50 general interconnect cycles after the user interface reset (ui_clk_sync_rst) is deasserted in the self-refresh exit cycle. It should stay asserted until the calibration completes. The memory interface signals reset_n and cke are expected to be at 1 and 0 values respectively at this time.
3. After data from block RAM in the static region is restored back to XSDB block RAM in the dynamic region signal xsdb_bram_restore_complete transitions from Low to High, as shown in Figure 12.

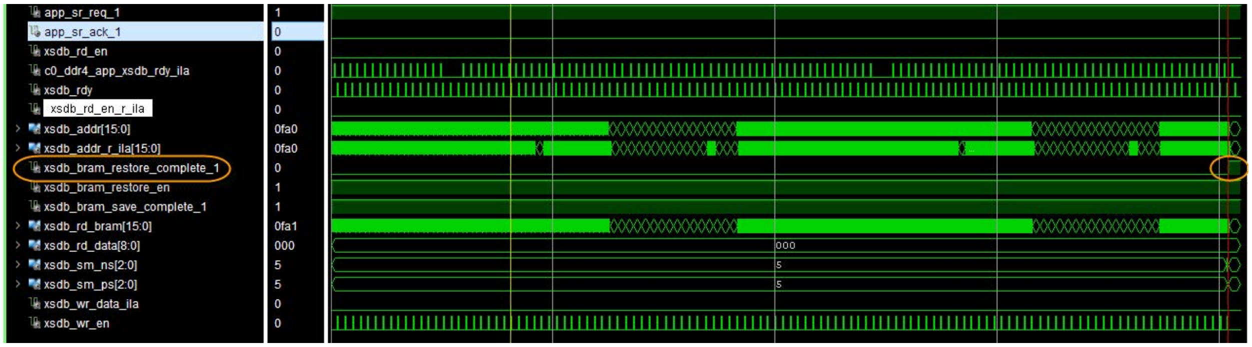
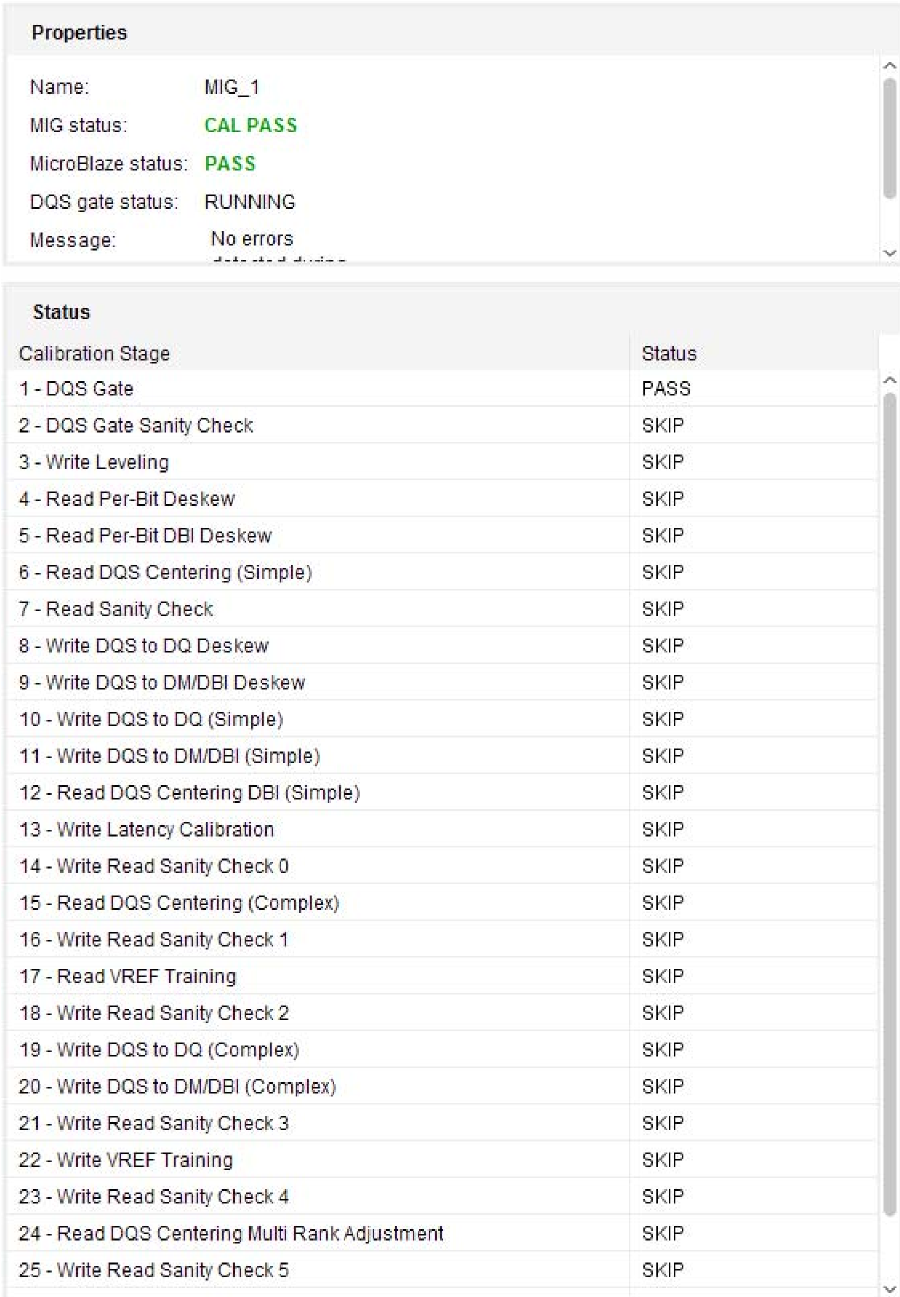


Figure 12: **Signal xsdb_bram_restore_complete Transition**

Figure 13 shows the calibration stage status GUI for the MIG. Note that after the self-refresh exit cycle the MIG DDR4 interface did not go through full calibration. It skipped all the calibration stages except DQS_GATE tracking. The GUI indicates that the calibration passed.



Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	SKIP
3 - Write Leveling	SKIP
4 - Read Per-Bit Deskew	SKIP
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	SKIP
7 - Read Sanity Check	SKIP
8 - Write DQS to DQ Deskew	SKIP
9 - Write DQS to DM/DBI Deskew	SKIP
10 - Write DQS to DQ (Simple)	SKIP
11 - Write DQS to DM/DBI (Simple)	SKIP
12 - Read DQS Centering DBI (Simple)	SKIP
13 - Write Latency Calibration	SKIP
14 - Write Read Sanity Check 0	SKIP
15 - Read DQS Centering (Complex)	SKIP
16 - Write Read Sanity Check 1	SKIP
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	SKIP
20 - Write DQS to DM/DBI (Complex)	SKIP
21 - Write Read Sanity Check 3	SKIP
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi Rank Adjustment	SKIP
25 - Write Read Sanity Check 5	SKIP

Figure 13: **Calibration Stage Status GUI**

Conclusion

This application note demonstrates how to use the save-restore feature of the DDR4 IP to greatly reduce training (calibration) time. The application note also describes how to utilize the DDR4 self-refresh feature to allow the use of partial or full reconfiguration of the FPGA for various needs and how to handle specific DDR4 DRAM signals in order to use these features.

Reference Design

Download the reference design files for this application note from the Xilinx website. Table 1 shows the reference design matrix.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer names	Xilinx
Target devices	UltraScale and UltraScale+ devices
Source code provided	Yes
Source code format	Verilog
Design uses code and IP from existing Xilinx application note and reference designs or third party	MIG DDR4 IP, Clock wizard, ILA, VIO, Debug Bridge IP from Vivado Design Suite
Simulation	
Functional simulation performed	No
Timing simulation performed	No
Test bench used for functional and timing simulations	No
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	No
Implementation	
Synthesis software tools/versions used	N/A
Implementation software tools/versions used	N/A
Static timing analysis performed	No
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	Xilinx internal test board

References

This application note uses the following references:

1. *UltraScale Architecture-Based FPGAs Memory IP v1.4* (PG150)
2. *Vivado Design Suite User Guide Partial Reconfiguration* (UG909)
3. *UltraScale Architecture Configuration User Guide* (UG570)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/23/2017	1.0	Initial Xilinx release.

Please Read: Important Legal Notices