XAPP743 (v1.0.1) October 28, 2013

Eye Scan with MicroBlaze Processor MCS

Author: Mike Jenkins and David Mahashin

Summary

This application note describes code that executes on an internal MicroBlaze™ processor, implementing the algorithm to measure a statistical eye (bit error ratio (BER) versus time and voltage offset) at the post-equalization data sampling point within the receiver of a 7 series FPGA transceiver. Point-by-point measured data is stored in block RAM to be burst read by an external host PC.

Introduction to Eye Scan

As line rates and channel attenuation increase, receiver equalizers are more often enabled to overcome channel attenuation. This poses a challenge to system bring-up because the quality of the line cannot be determined by measuring the far-end eye opening at the receiver pins. At high line rates, the received eye measured on the printed circuit board can appear to be completely closed even though the internal eye after the receiver equalizer is open.

The RX Eye Scan in GTH, GTX, and GTP transceivers of 7 series FPGAs provides a mechanism to measure and visualize the receiver eye margin after the equalizer. Additional use modes enable several other methods to determine and diagnose the effects of equalization settings.

All Eye Scan functionality is based on comparison between the Data Sample in the nominal center of the eye and the Offset Sample captured by identical circuitry at a programmable horizontal and vertical offset from the nominal center of the eye (see the left plot in Figure 1). A bit error is defined as a miscompare between these two samples, and the BER at a particular horizontal and vertical offset is the ratio of the number of bit errors to the total number of such data-to-offset-sample comparisons. Calculating BER at each point of an array of horizontal and vertical offsets provides the data for a statistical eye. (See the right plot in Figure 1, where the color is mapped to $\log_{10}$(BER).)

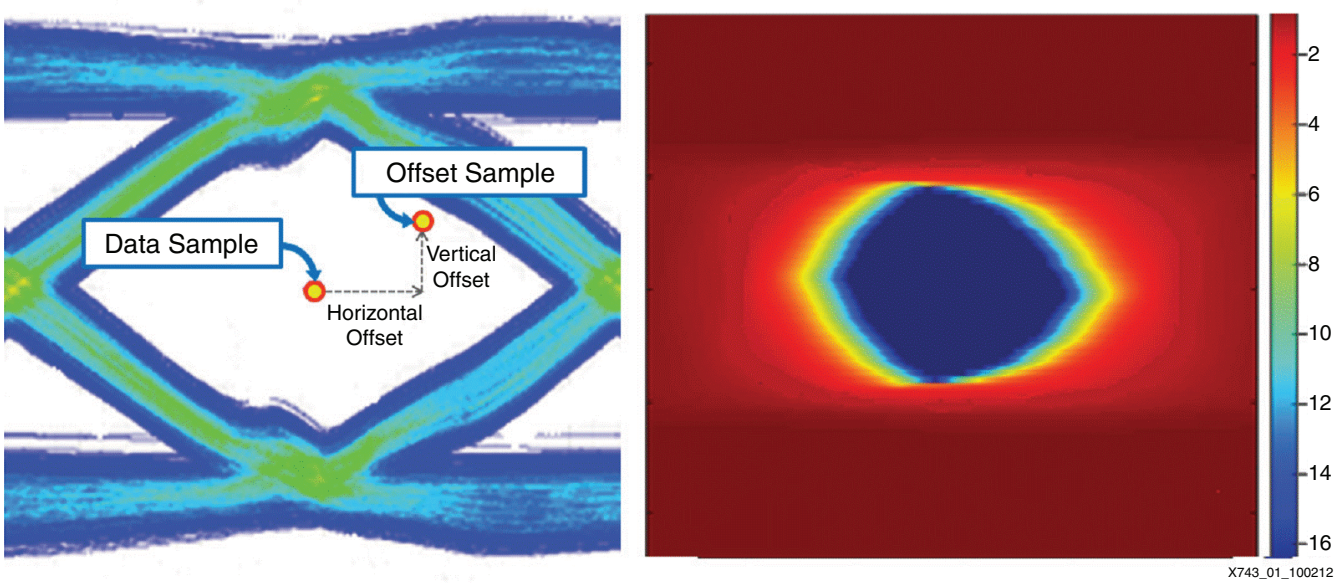


Figure 1: **Sample and Data Sample to Calculate BER as a Function of Offset—The Statistical Eye**

See the "RX Margin Analysis" section of *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) for more information as well as a detailed description of the attributes and ports associated with accumulating the above described error and sample counts.

Feature Description

The circuitry implemented in silicon in each GTX/GTH transceiver is the minimum functionality needed to support error and sample accumulation at any line rate with no impact on received data. Higher level algorithm logic must therefore be provided in some programmable fashion.

Initially, these algorithms were developed in external software, providing maximum flexibility and speed of code development. The cost of this approach, however, was the time to read or write each attribute through a serial interface. Because hundreds or thousands of such R/W operations are needed for a statistical eye, this became the dominant part of total algorithm execution time for BER acquisition below about 10-9. Inability to compile the code also limited the extensibility of this approach.

The implementation documented here represents a further evolution of the Programmable Imperative as it relates to the statistical eye algorithm. Because the algorithm has been reasonably well developed, translation into MATLAB® code was straightforward and provided a mechanism for automatically generating compilable C code. Further, this code is now written to be reentrant so one copy can be used to simultaneously acquire statistical eyes on multiple transceivers. Lastly, running the compiled code on a processor on chip greatly reduces the attribute R/W component of the total execution time.

There are many advantages to implementing Eye Scan on a MicroBlaze processor compared to designing a state machine in FPGA logic:

- Software-based implementation of Eye Scan allows faster turnaround time and flexibility of continual improvements:
  - Higher level of abstraction and enhanced code readability.
  - Faster compilation (compiles in seconds vs. minutes or hours).
  - Extensive debug support on Xilinx Software Development Kit (SDK).
- Capability of running Eye Scans on multiple channels simultaneously. Easy to extend to even more channels in the future.

- C code that can be compiled for the statistical eye algorithm and reused in other implementations.

Hardware Design

The centerpiece of the hardware design is the Micro Controller System (MCS) core of the MicroBlaze processor. Refer to *LogiCORE IP MicroBlaze Micro Controller System* (DS865) for more information on MCS. The MCS accesses the Eye Scan circuitry in the 7 series FPGA GTX transceiver through its dynamic reconfiguration port (DRP). A block RAM is used to temporarily store error and sample count data before they can be uploaded to a host computer. Because both DRP and block RAM share the same MCS address space, a simple state machine is implemented to decode the address and properly direct read and write commands to either block RAM or DRP.

The MCS Eye Scan design is built on top of a 4-lane XAUI example design generated by 7 Series FPGA Transceiver Wizard version 2.2. The design consists of four main modules, as shown in Figure 2:

- MicroBlaze MCS (`microblaze_mcs.v`)
- 7 Series GTX Transceivers (`gtx7_init.v`)
- DRP/block RAM Access Control (`drp_bridge.v`)
- Data Storage block RAM (`block_ram_2048x8.v`)

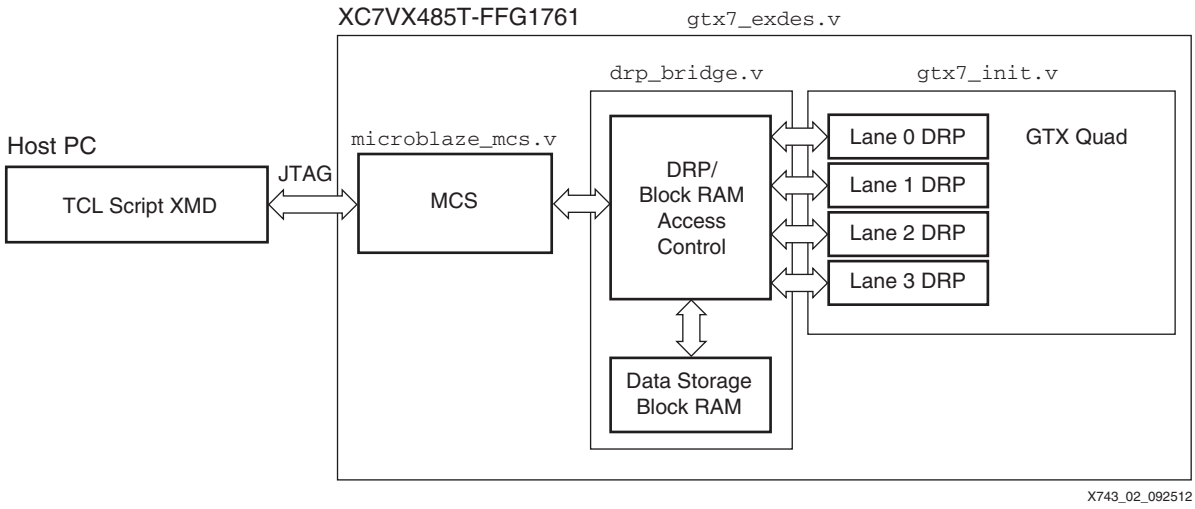


Figure 2: **Top-Level Block Diagram**

Notes relevant to Figure 2:

1. Non-relevant Wizard example design modules are not shown.
2. XMD is Xilinx Microprocessor Debugger.

7 Series GTX Transceivers

Using the 7 Series FPGAs Transceivers Wizard, four full-duplex GTX transceivers are generated with XAUI protocol template settings. The following modifications are done to the Wizard example design:

- MCS and other logic is inserted (Figure 2)
- The GTX transceiver's PMA_RSV2[5] attribute is set to 1 to enable Eye Scan circuitry

Figure 3 through Figure 9 show features selected in the Wizard.

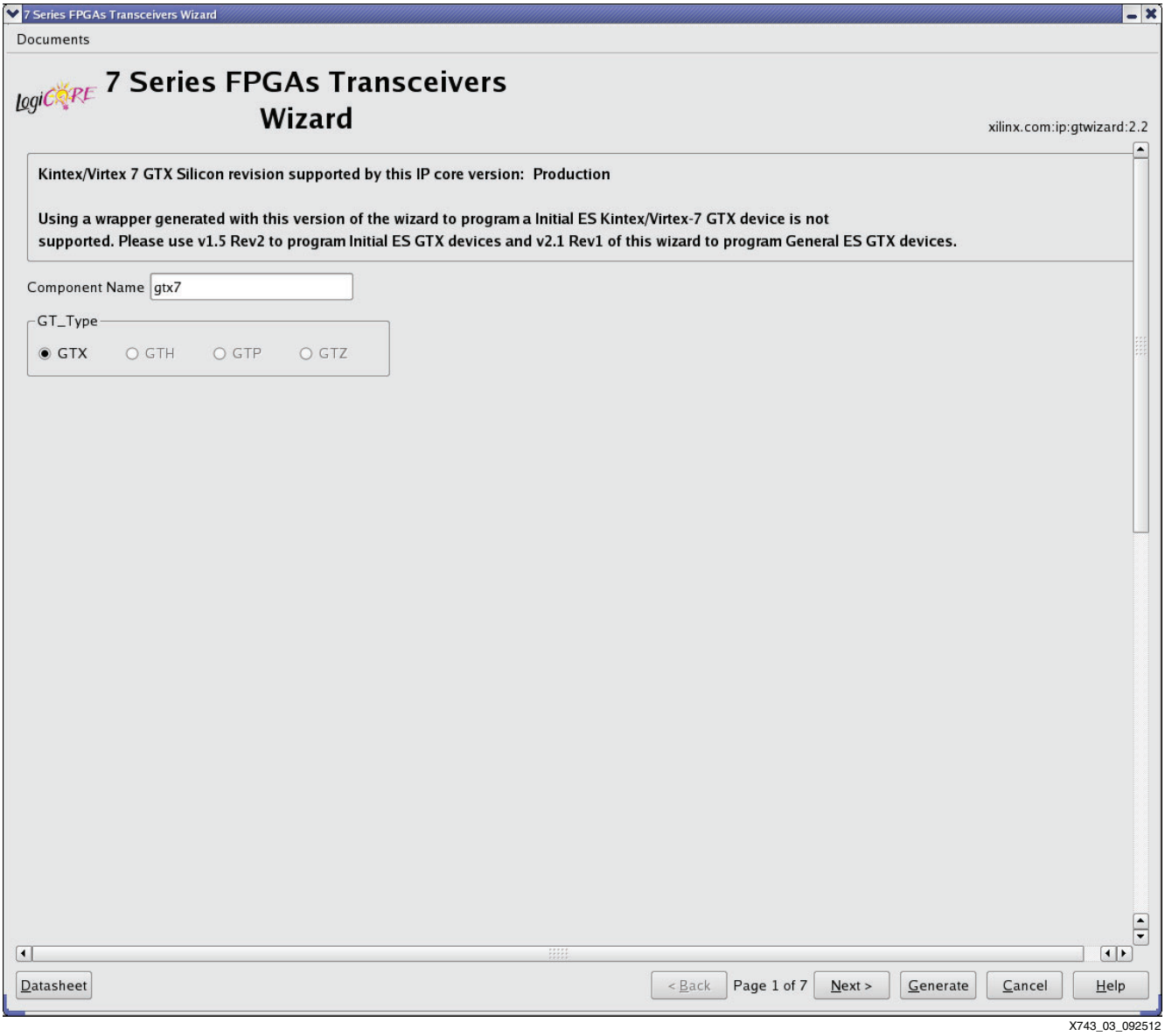


Figure 3: **7 Series FPGA Transceivers Wizard (1)**

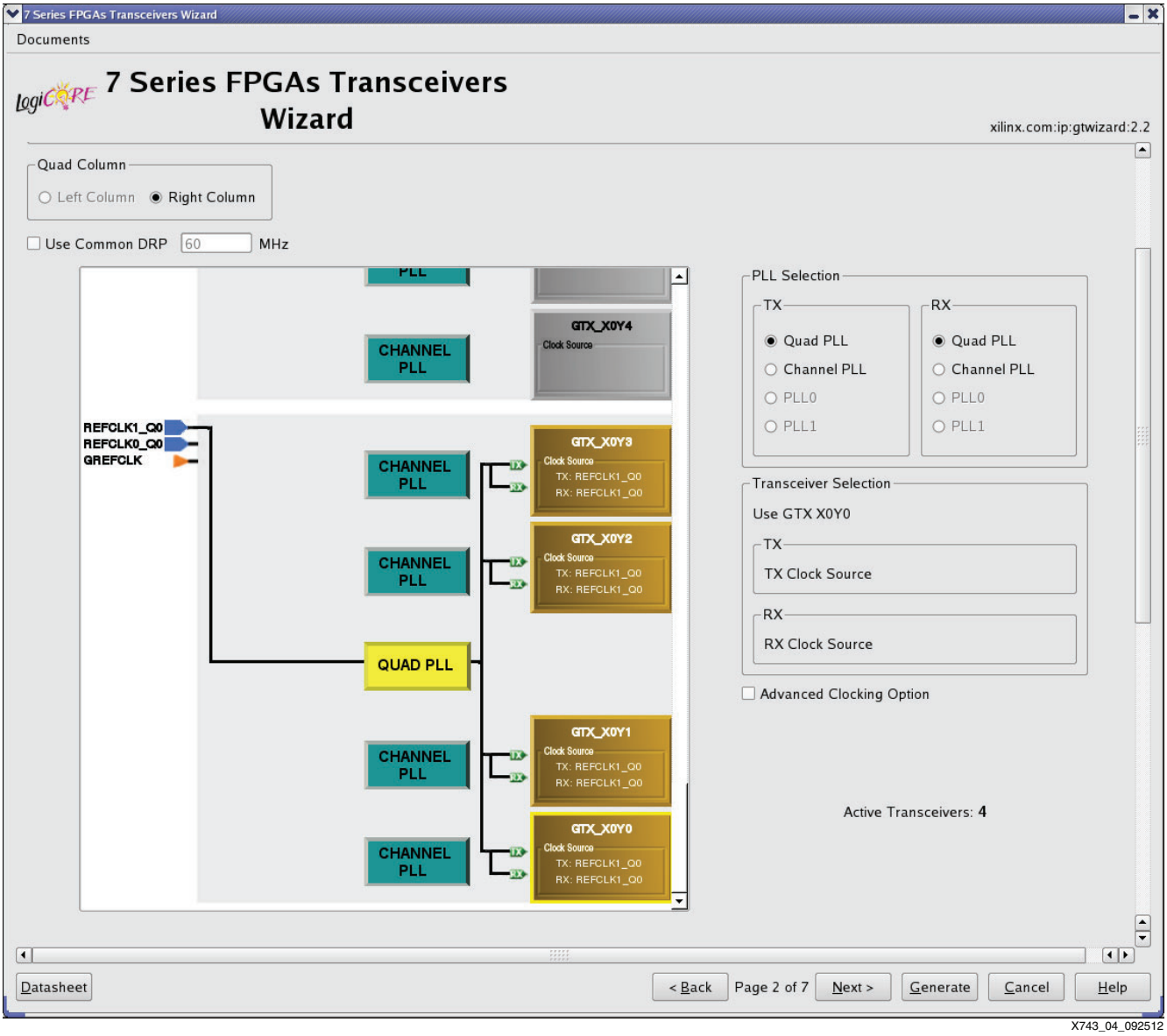


Figure 4: **7 Series FPGA Transceivers Wizard (2)**

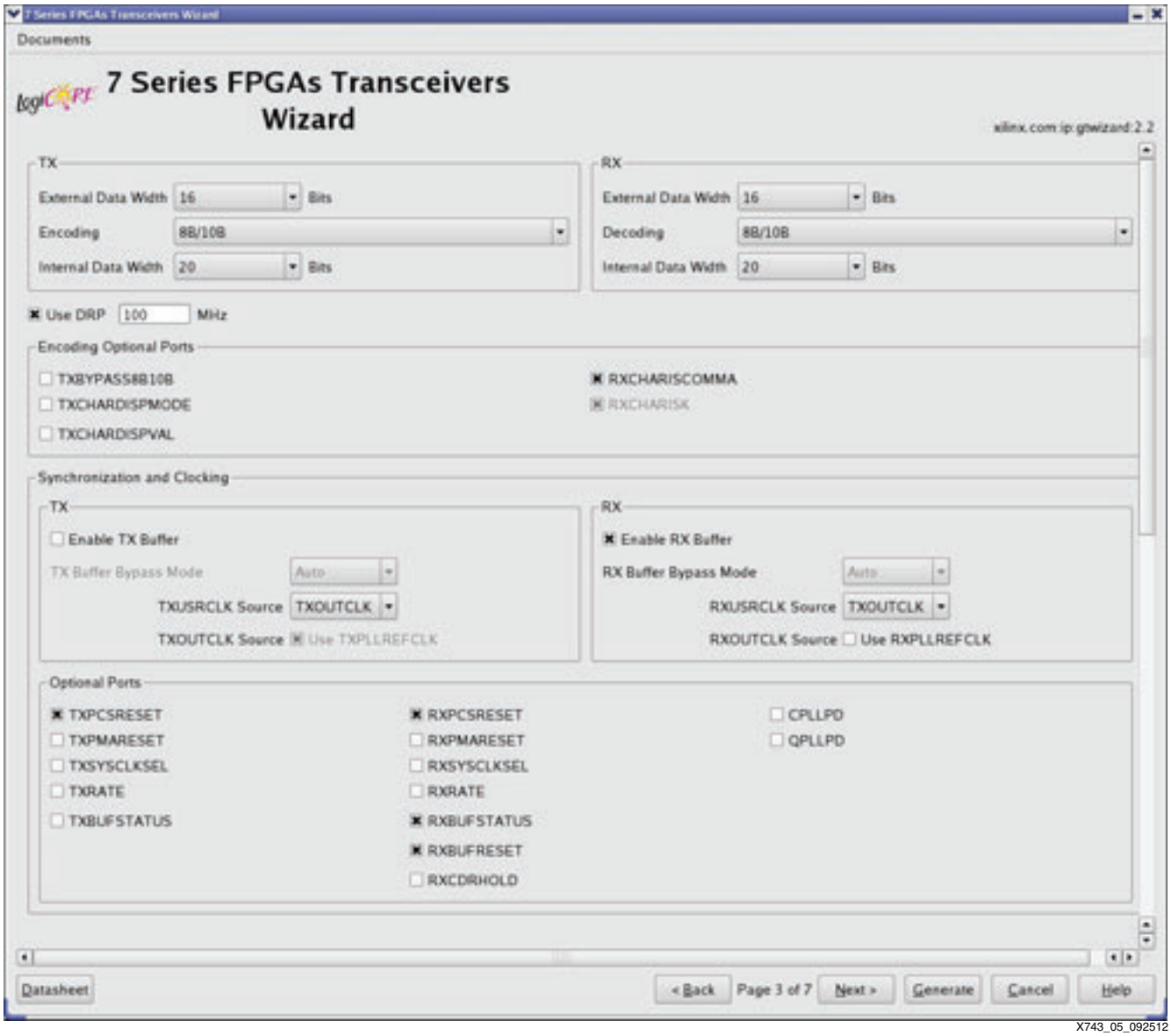


Figure 5: **7 Series FPGA Transceivers Wizard (3)**

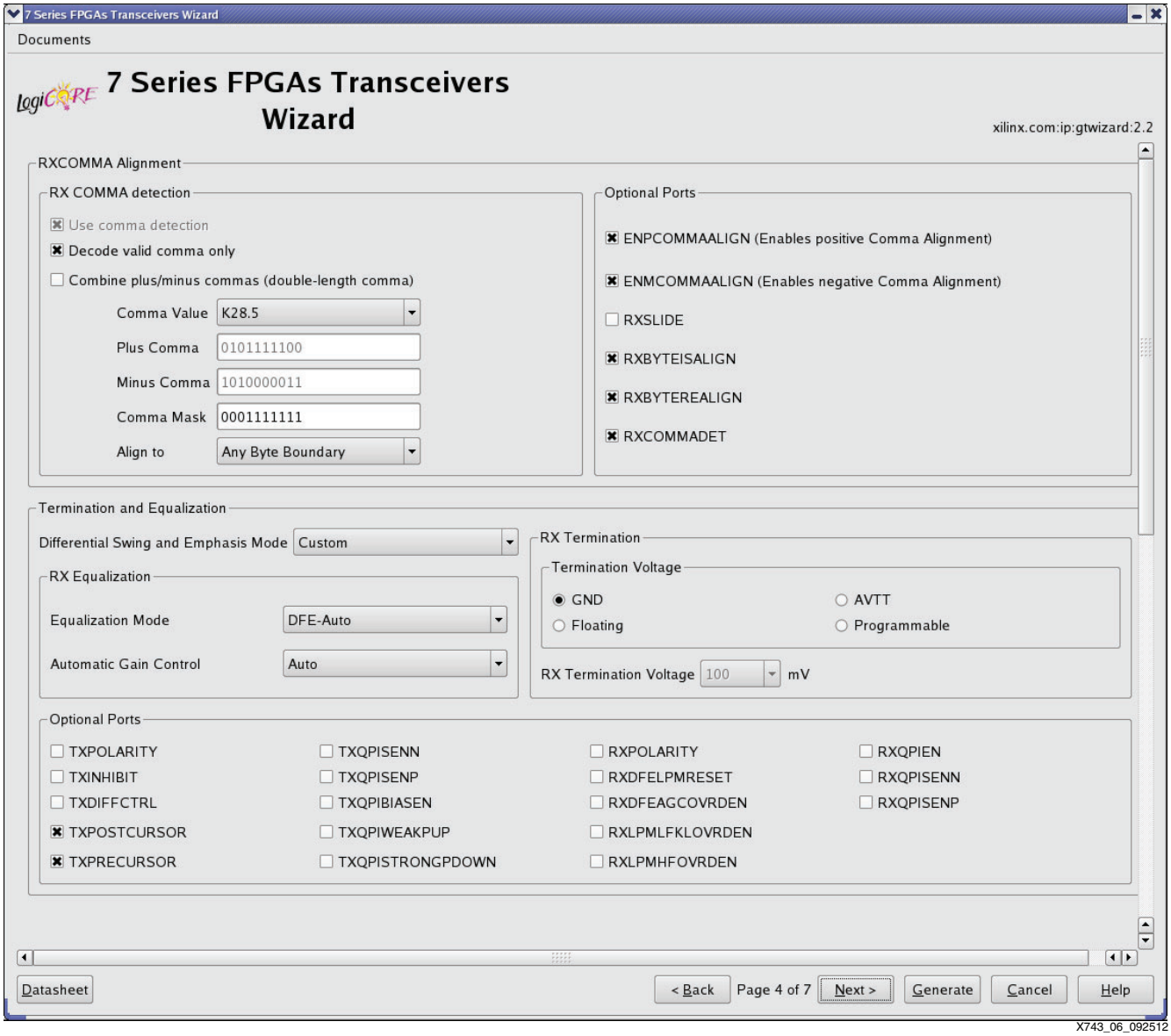


Figure 6: **7 Series FPGA Transceivers Wizard (4)**

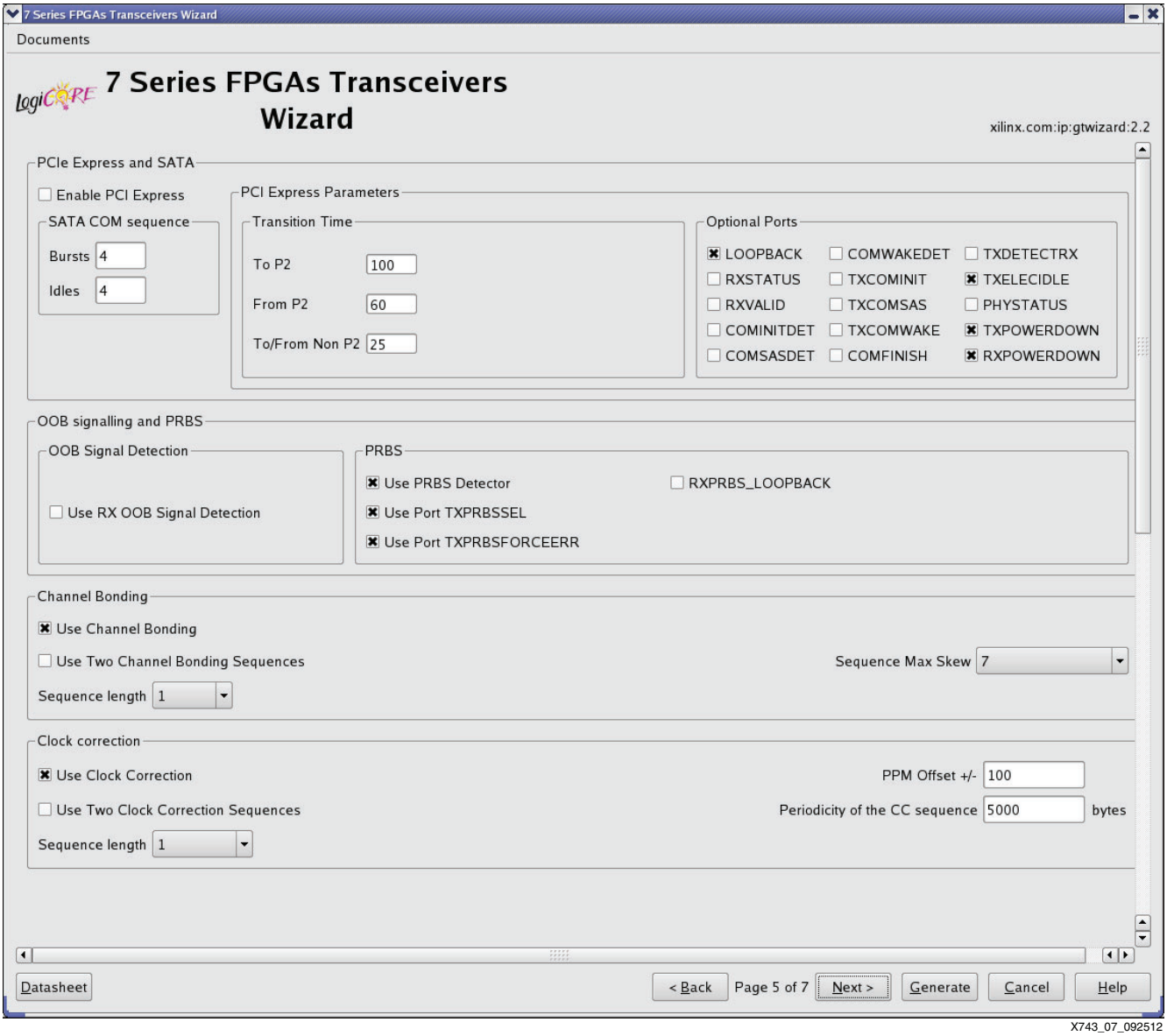


Figure 7: **7 Series FPGA Transceivers Wizard (5)**

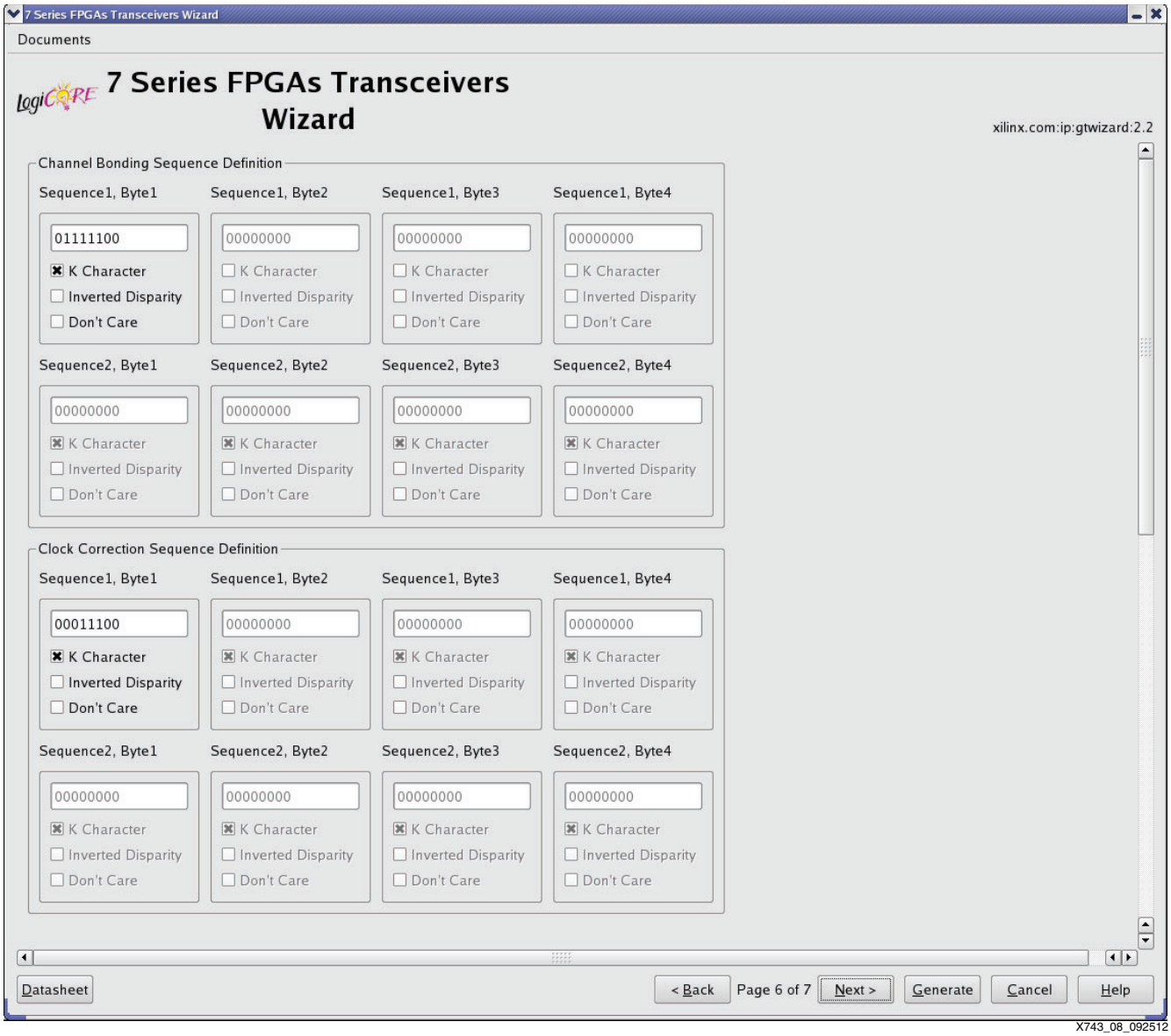


Figure 8: **7 Series FPGA Transceivers Wizard (6)**

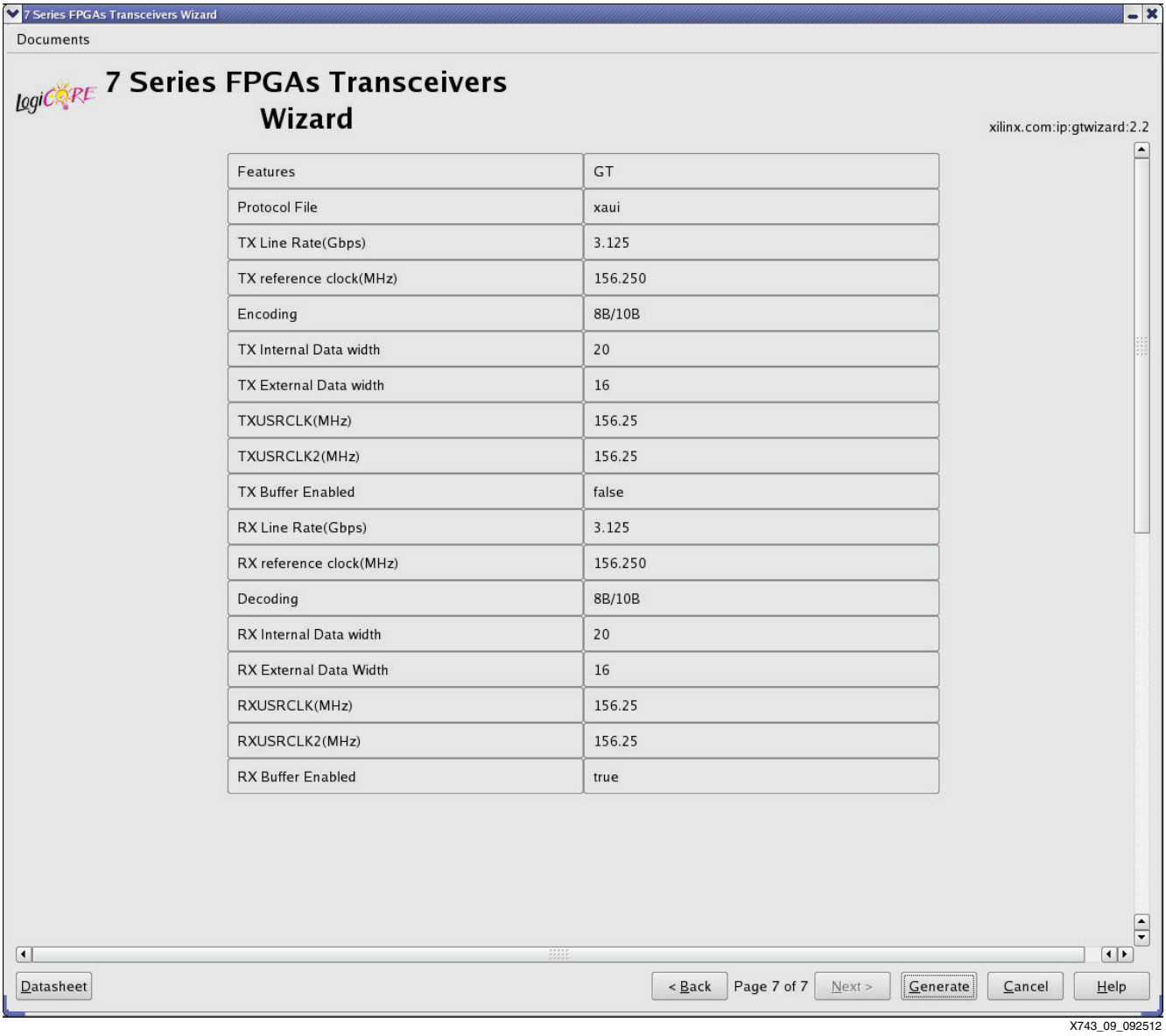


Figure 9: **7 Series FPGA Transceivers Wizard (7)**

MCS

The MCS is generated using the CORE Generator™ tool with I/O bus enabled. The I/O bus is used to access GTX DRP registers as well as data storage block RAM. A maximum memory size of 64 KB is selected to give flexibility in designing the software. The Enable Debug Support and Enable MicroBlaze Trace Bus options are checked to enable debugging capabilities.

Figure 10 through Figure 16 show the MCS features selected

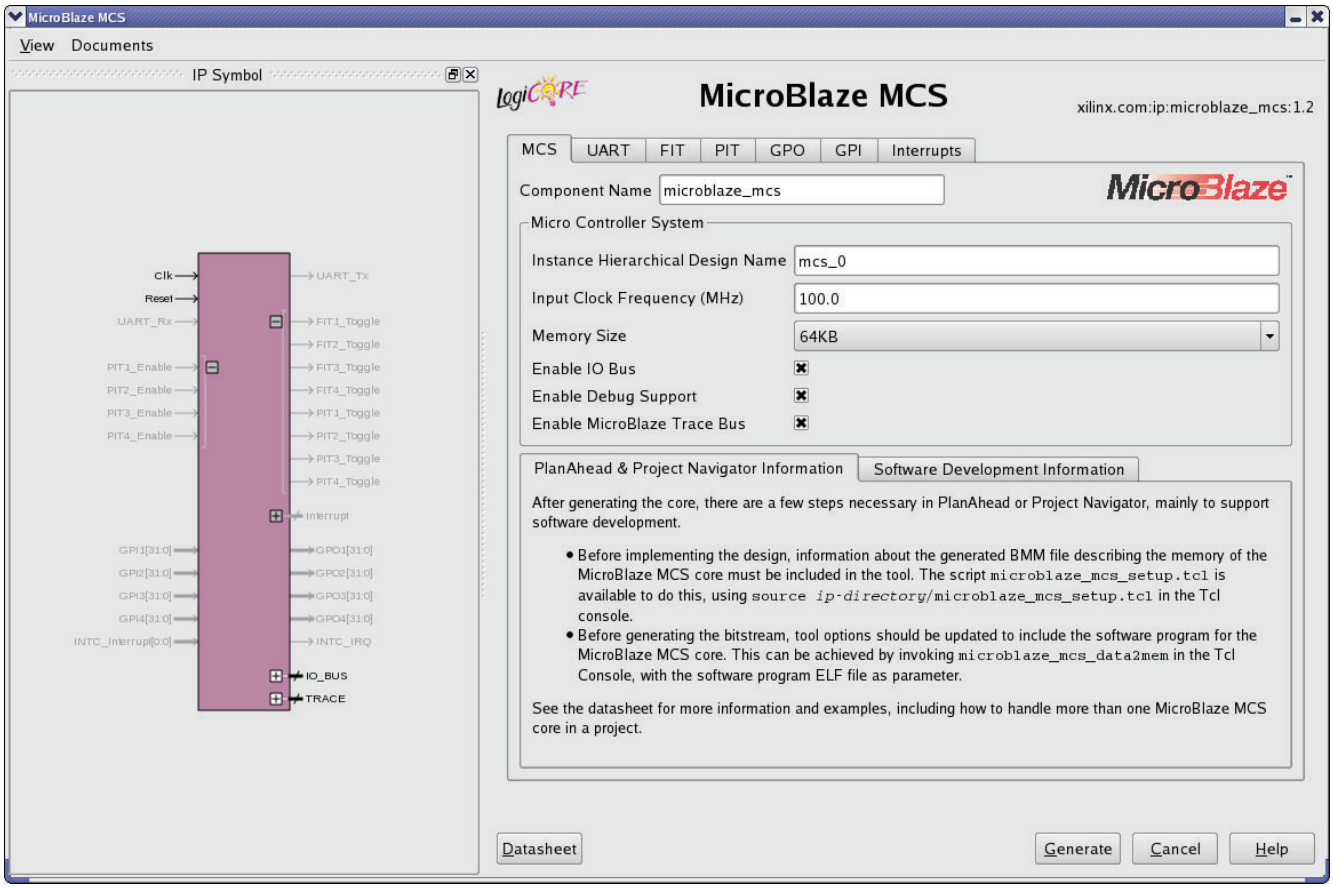


Figure 10: **MCS Settings**

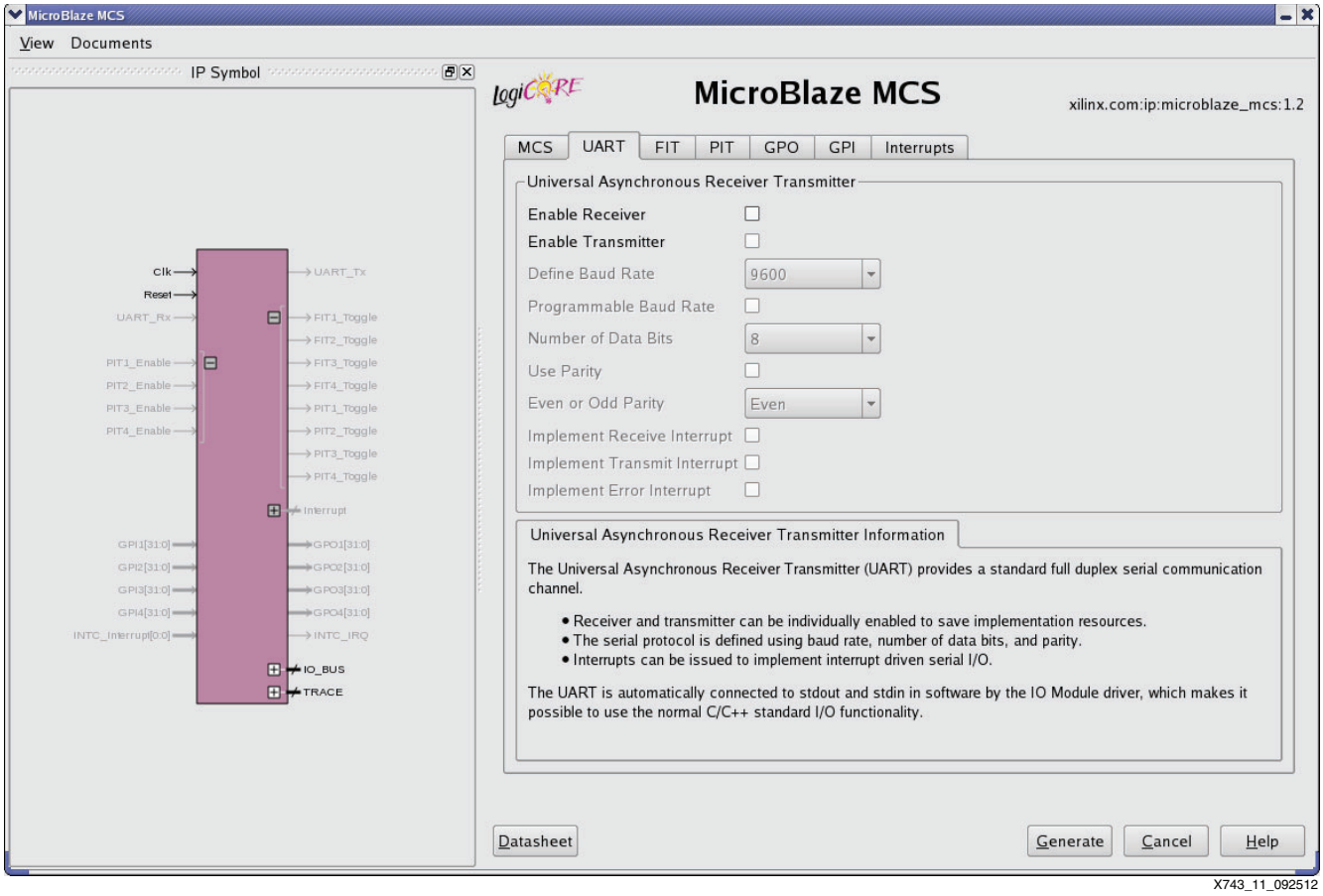


Figure 11: **MCS UART Settings**

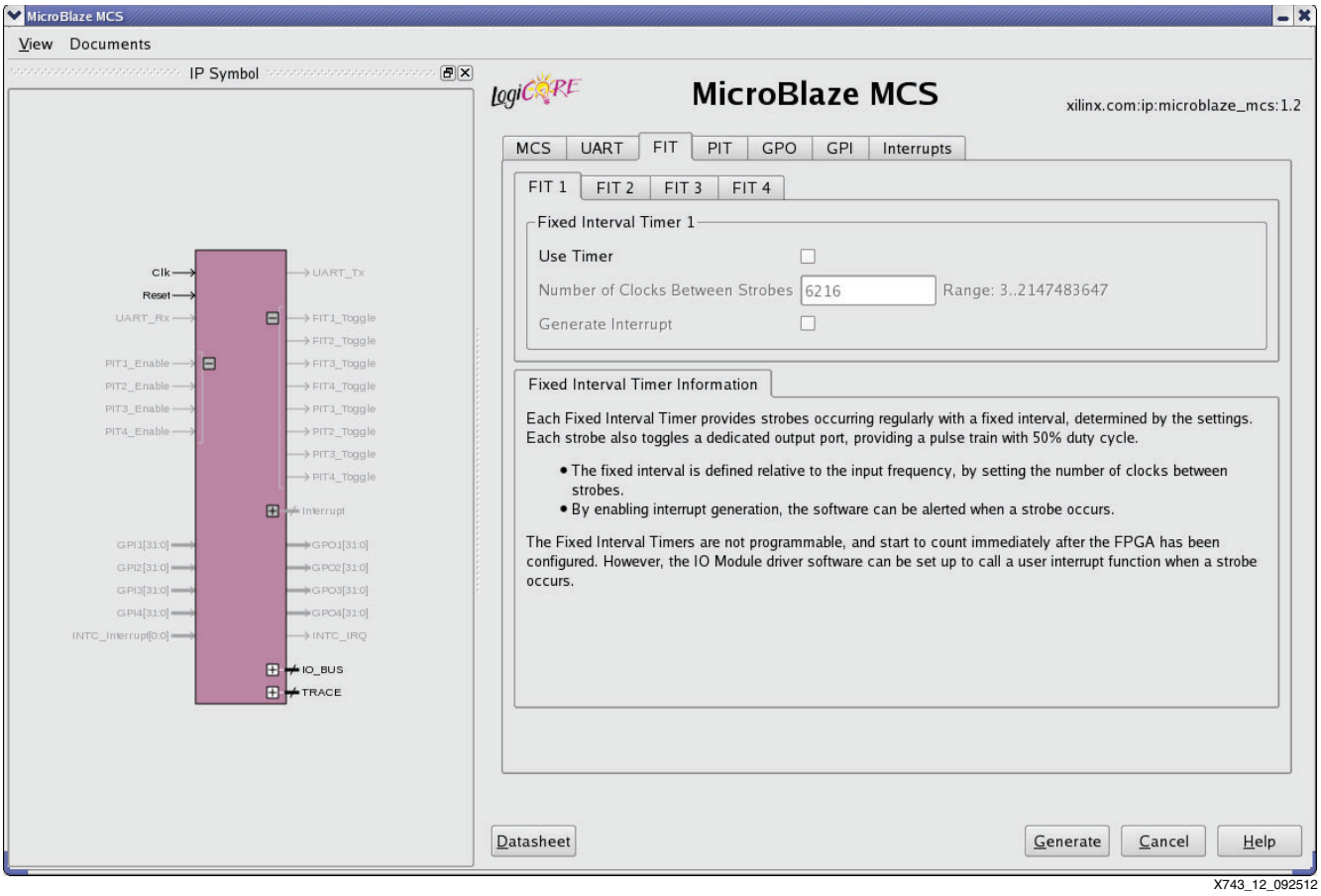


Figure 12: **MCS FIT Settings**

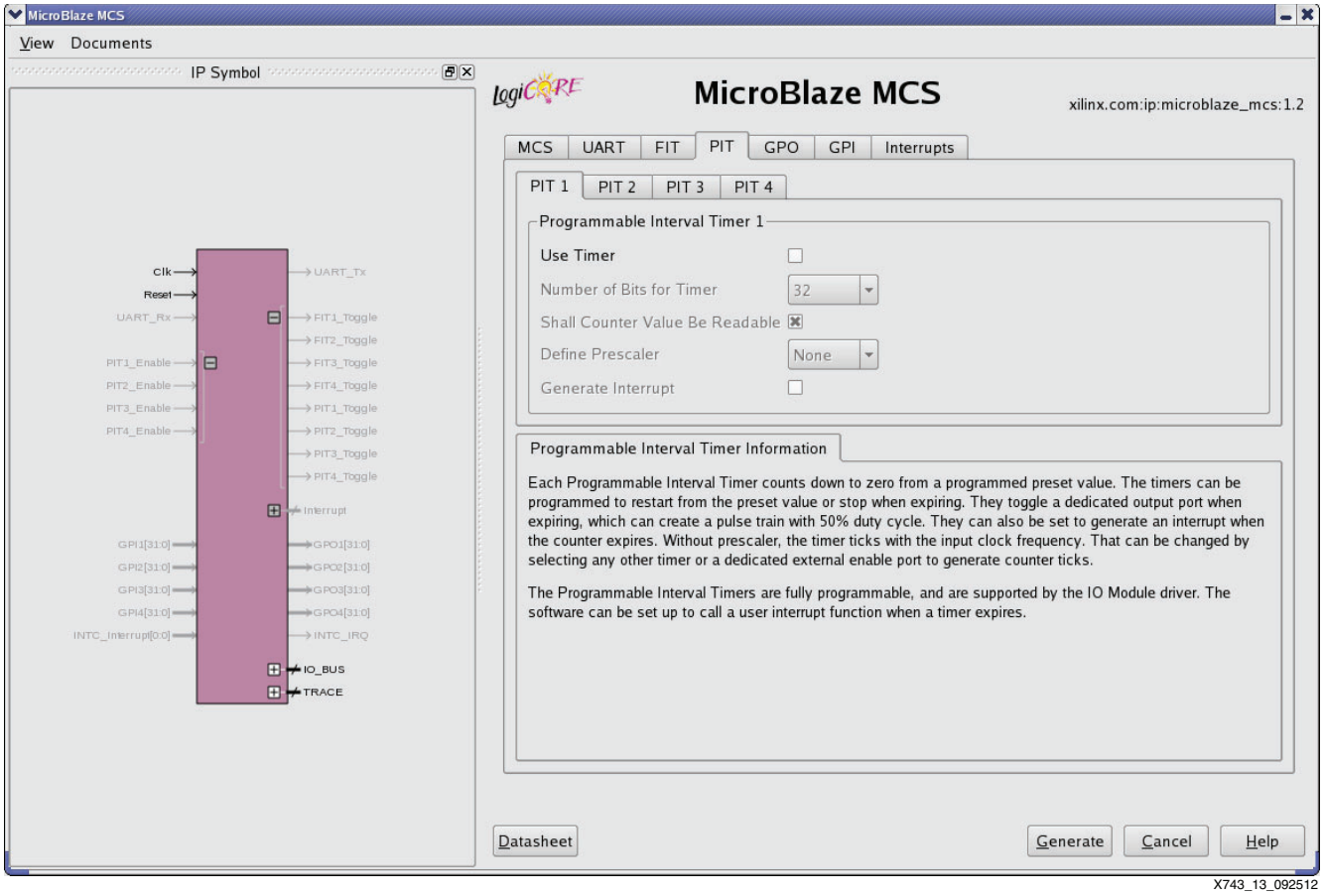


Figure 13: **MCS PIT Settings**

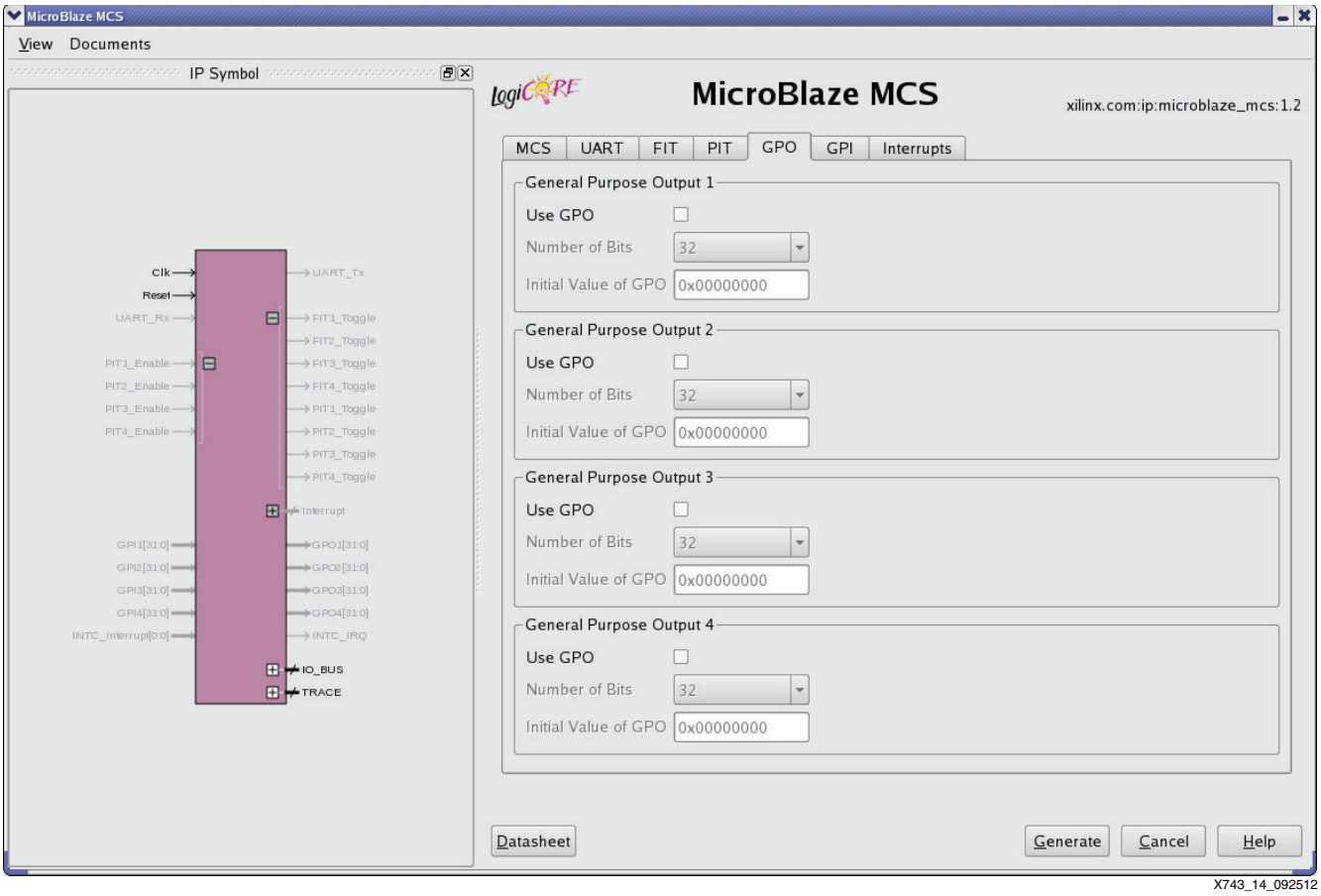


Figure 14: **MCS GPO Settings**

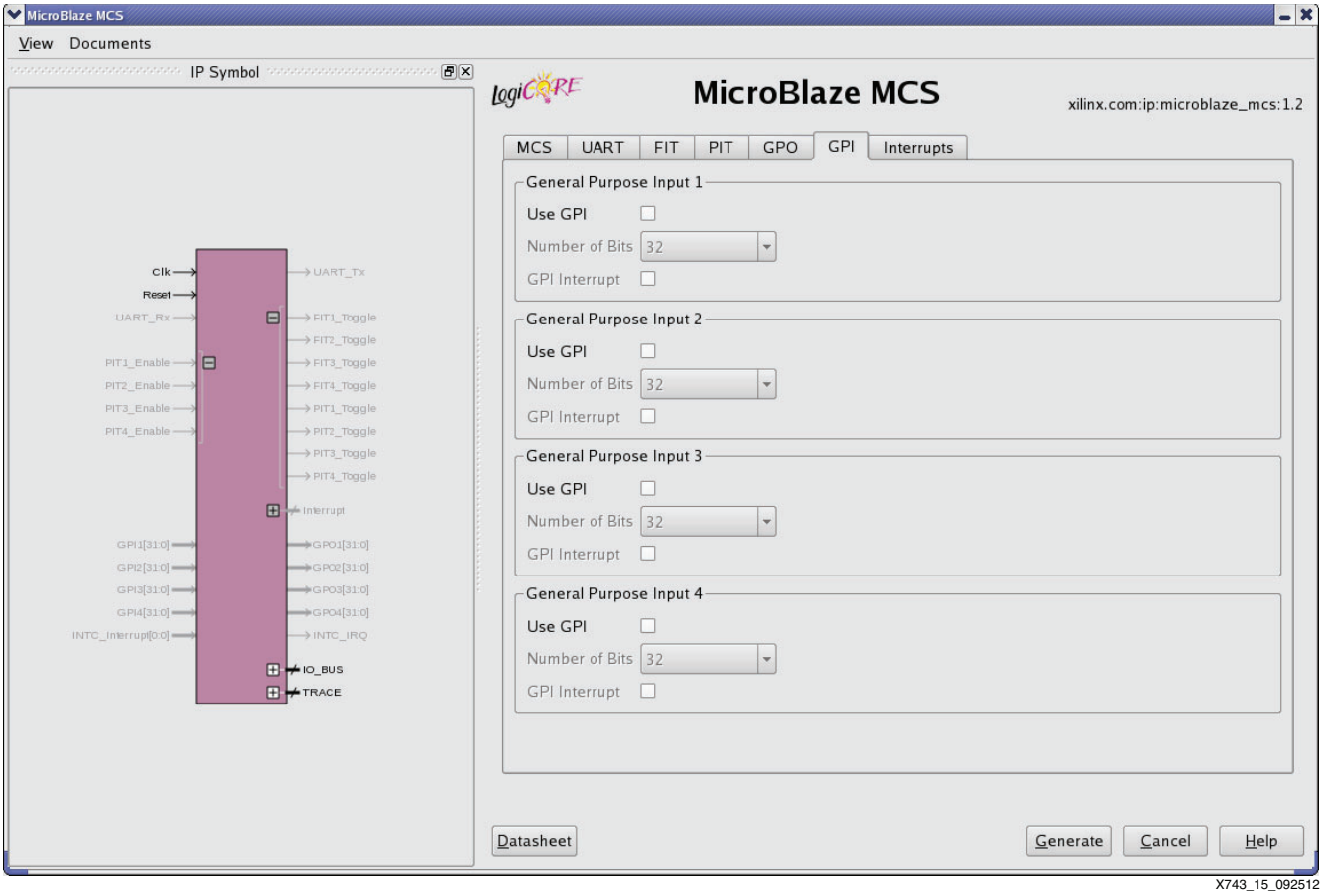


Figure 15: **MCS GPI Settings**

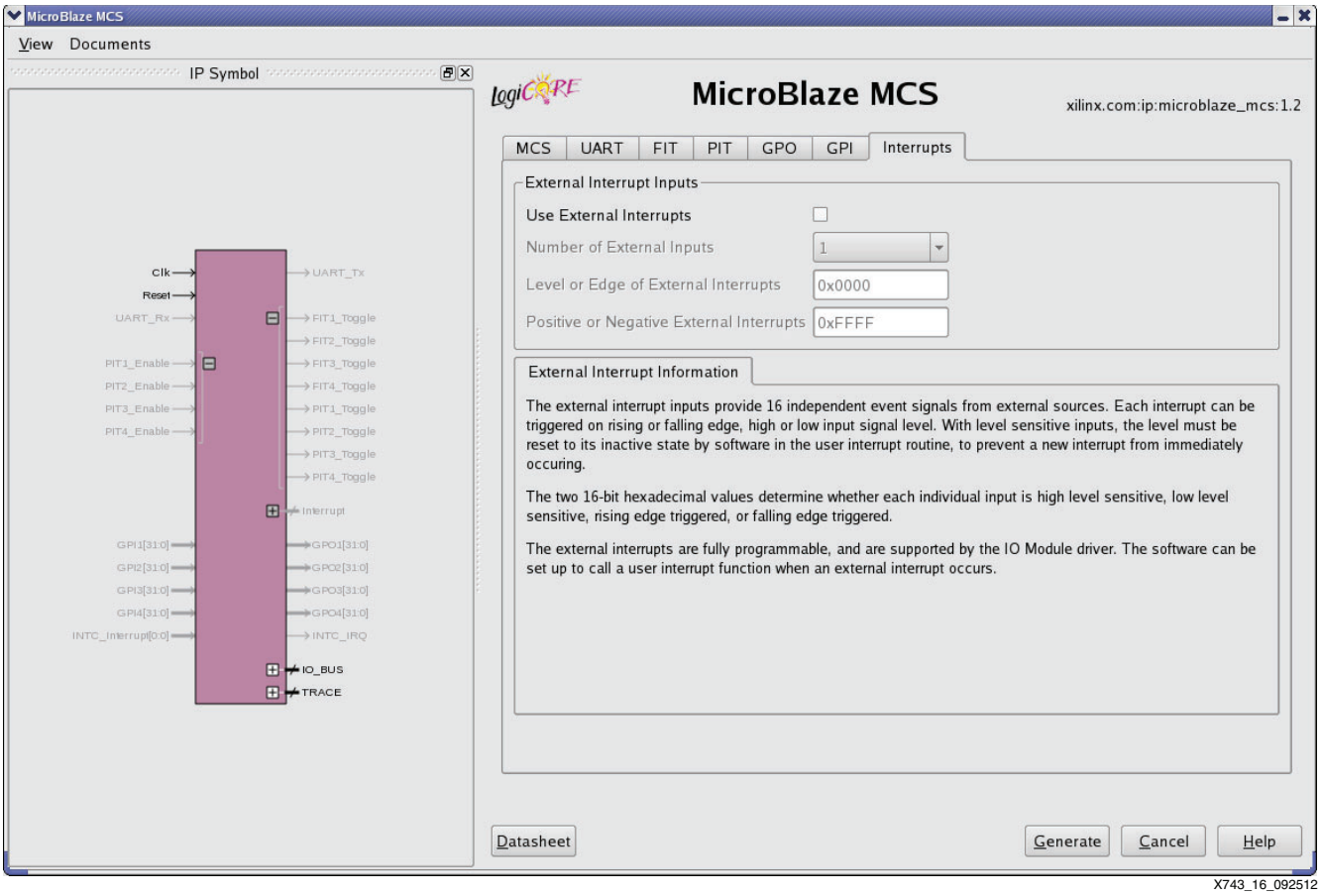


Figure 16: **MCS Interrupt Settings**

DRP/Block RAM Access Controller

The drp_bridge module implements a state machine to decode the MCS I/O bus address (Table 1), and properly directs read/write commands to either data storage block RAM or one of the four transceiver's DRP ports.

Table 1: **MCS Address Mapping for GTX DRP and Data Storage Block RAM**

Item	MCS I/O Bus Address Range (Hex)	Name	Description
1	0xC000_2000-0xC000_27FF	Lane 0 DRP	[13] = Set to 1 to select DRP, 0 to select data storage block RAM [12:11] = Lane number [10:2] = DRP address [1:0] = Ignored
2	0xC000_2800-0xC000_2FFF	Lane 1 DRP	
3	0xC000_3000-0xC000_37FF	Lane 2 DRP	
4	0xC000_3800-0xC000_3FFF	Lane 3 DRP	
5	0xC000_4000-0xC000_47FF	Lane 0 data storage block RAM	Eye Scan data, control, and status
6	0xC000_4800-0xC000_4FFF	Lane 1 data storage block RAM	
7	0xC000_5000-0xC000_57FF	Lane 2 data storage block RAM	
8	0xC000_5800-0xC000_5FFF	Lane 3 data storage block RAM	

Data Storage Block RAM

Four byte-wide, 2,048 deep memory blocks are instantiated to store Eye Scan data, control, and status information. The block RAM is accessed solely by the MCS I/O bus.

The width of 8 bits is selected to match the byte-based address space of the MCS I/O bus. Selection of memory depth is based on the trade-off between area and Eye Scan speed. Because MCS stalls the Eye Scan when the block RAM is full, a larger block RAM would decrease scan time at the expense of memory usage.

Figure 17 to Figure 22 show the block RAM settings selected.

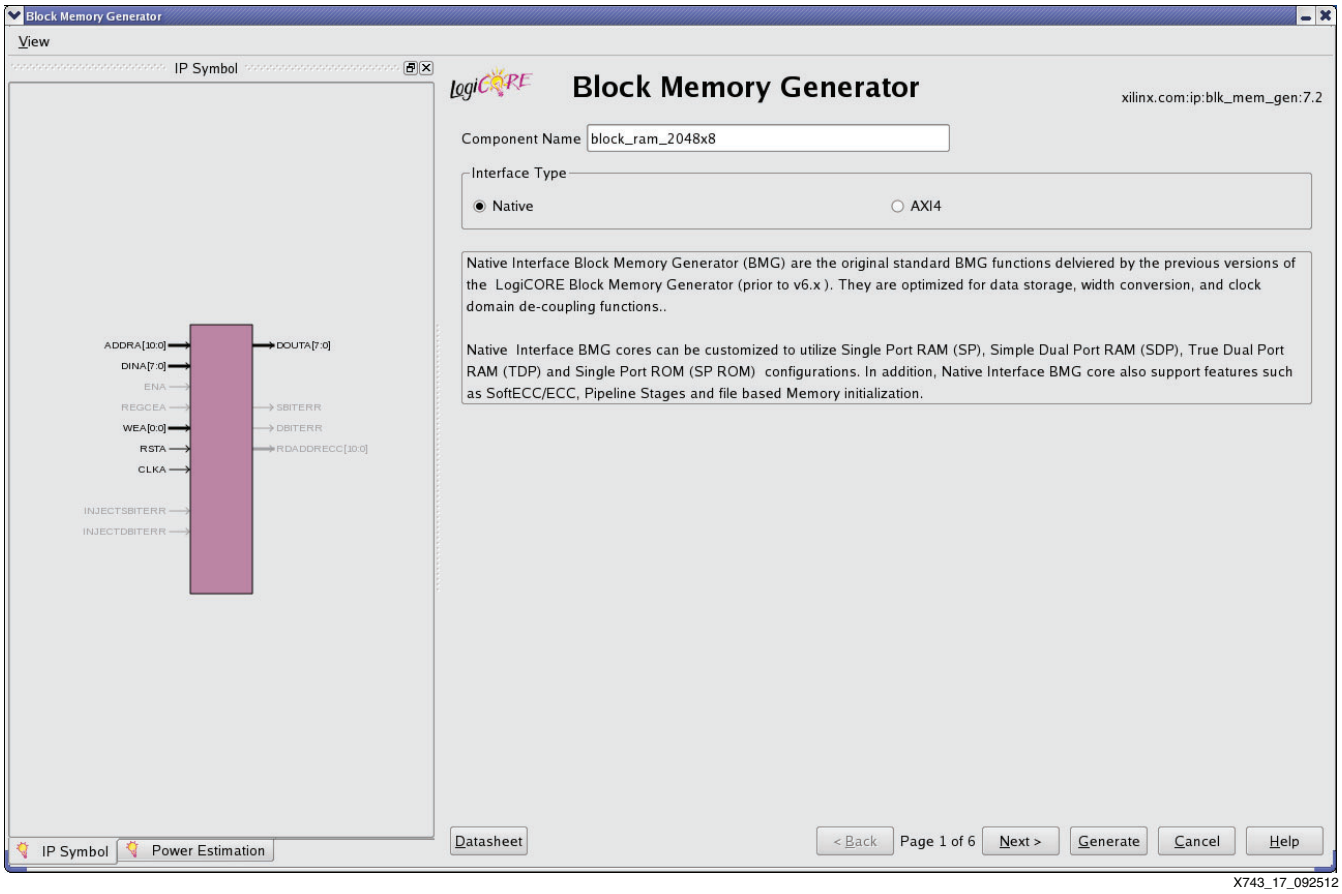


Figure 17: **Block Memory Generator (1)**

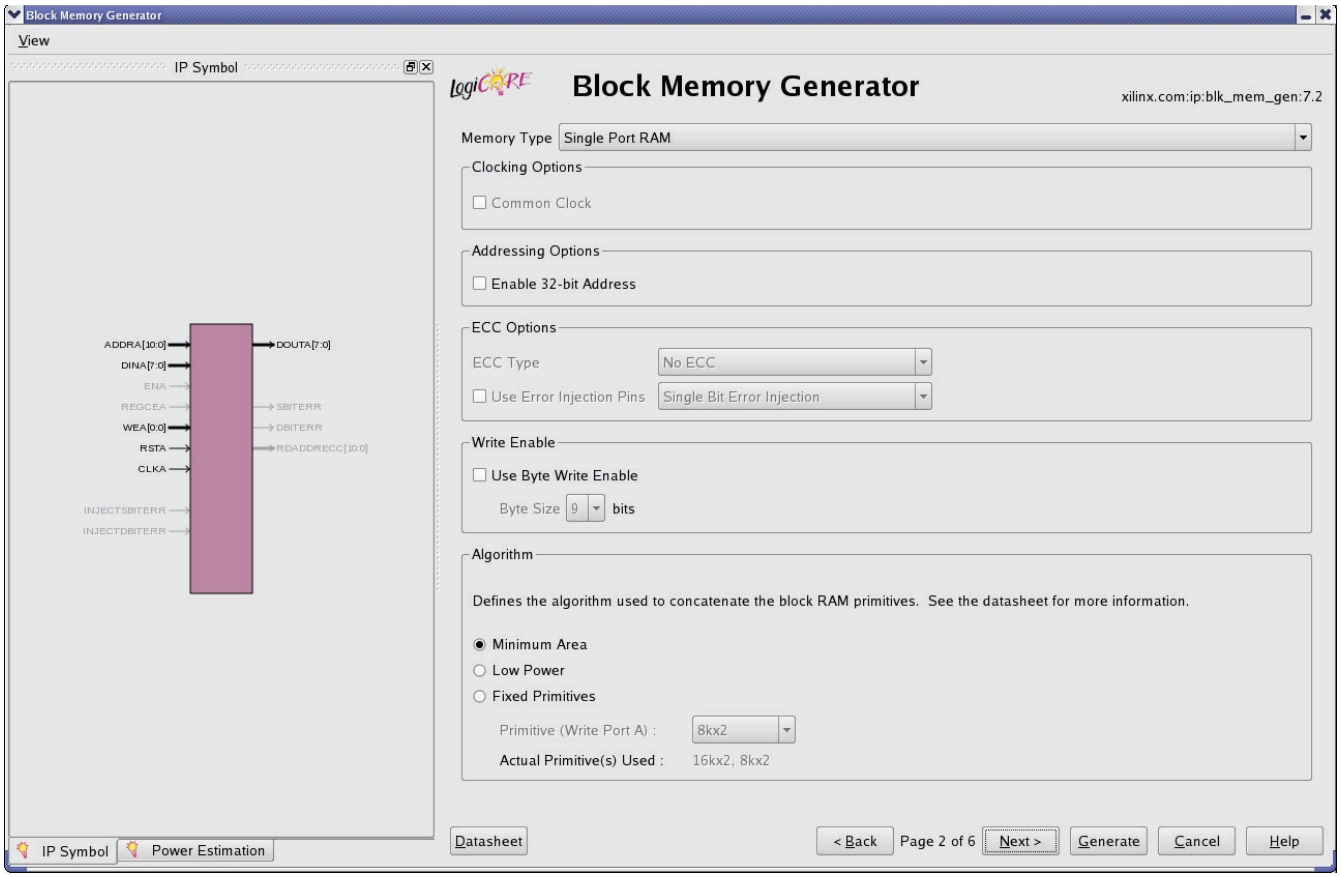


Figure 18: **Block Memory Generator (2)**

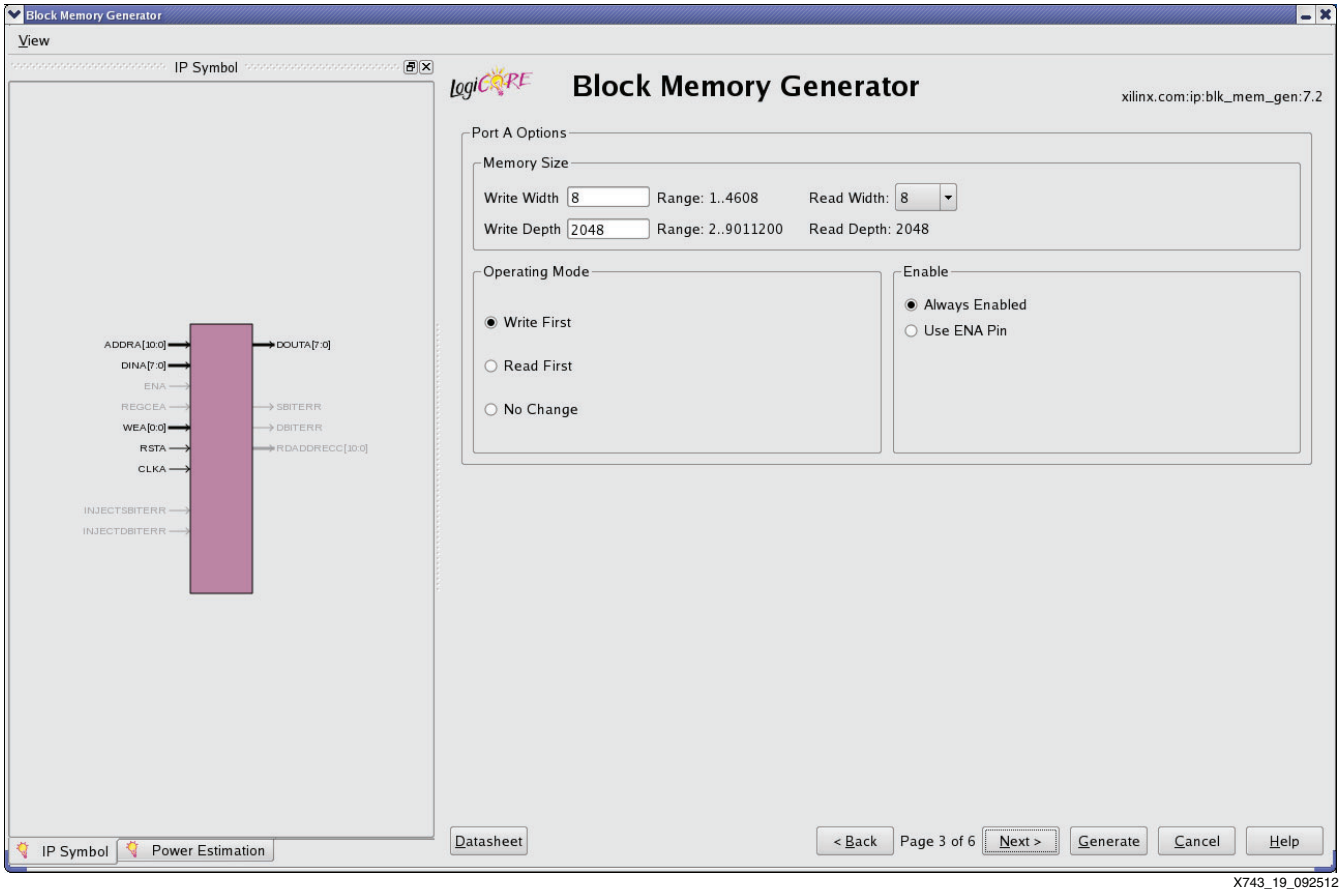


Figure 19: **Block Memory Generator (3)**

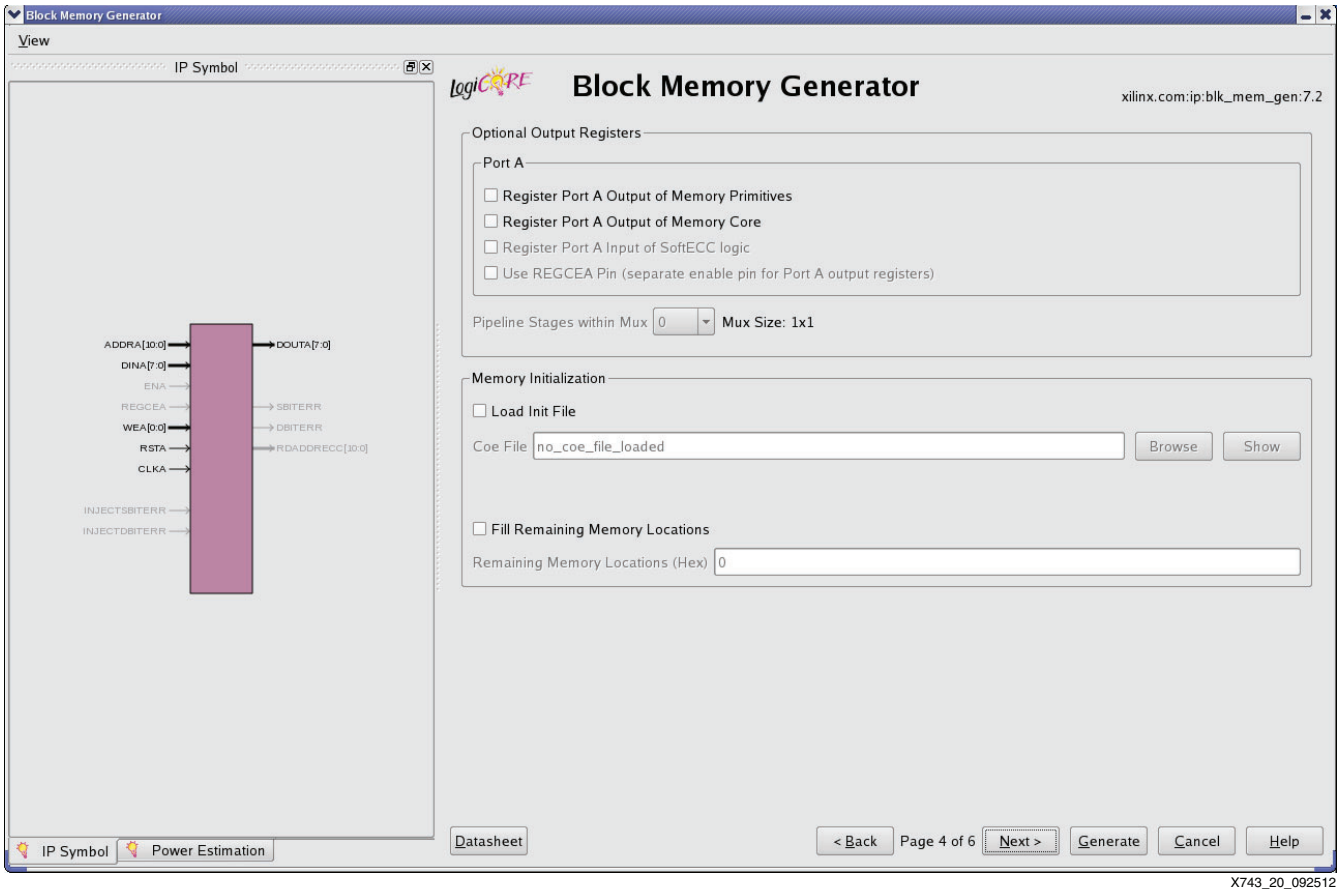


Figure 20: **Block Memory Generator (4)**

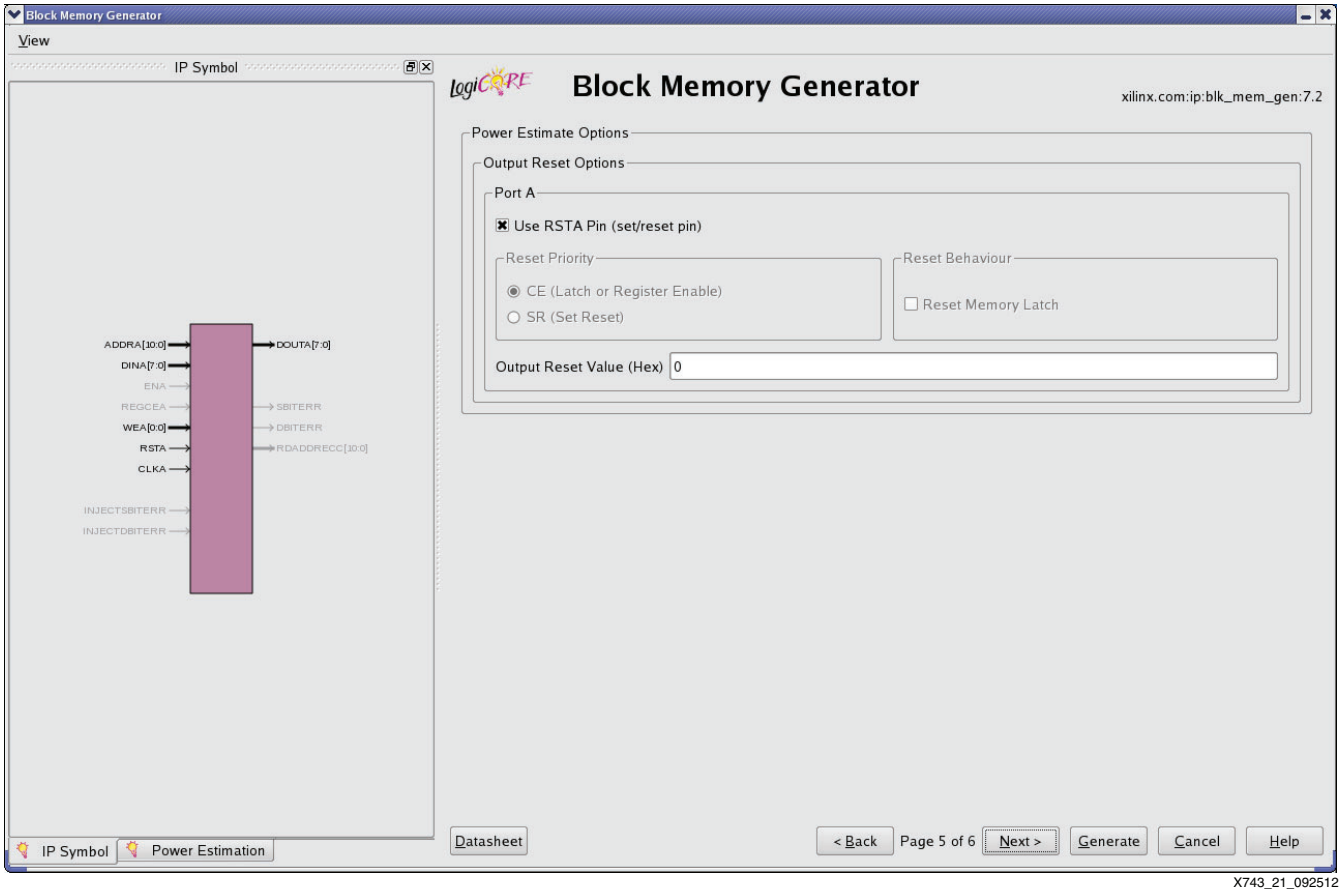


Figure 21: **Block Memory Generator (5)**

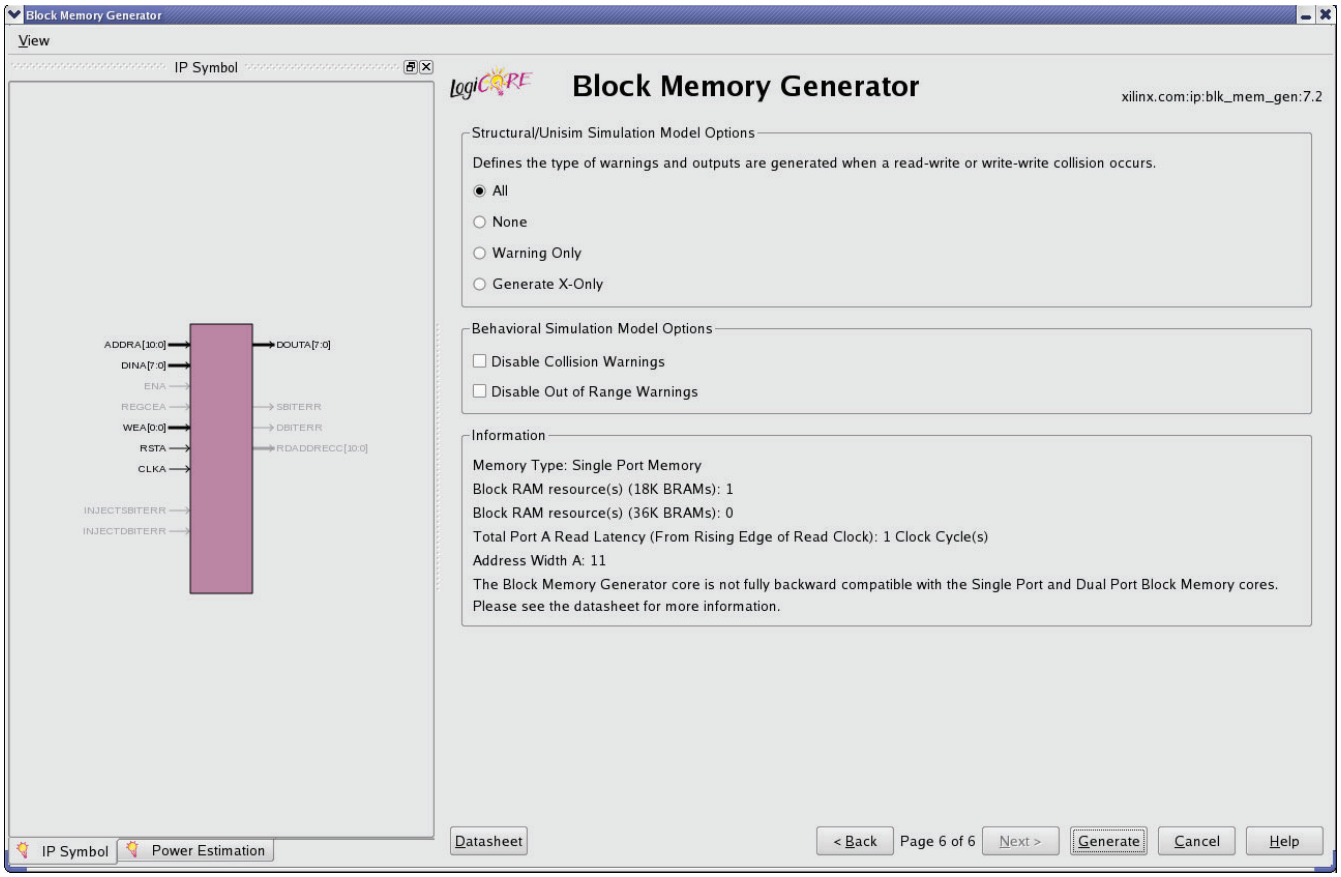


Figure 22: **Block Memory Generator (6)**

ICON Core JTAG Boundary Scan

An ICON core is included in the Wizard example design to enable the ChipScope™ tool Virtual Input/Output (VIO) and Integrated Logic Analyzer (ILA) features. To avoid collision between ICON and MCS boundary scan chains, it is necessary to regenerate the ICON core to use the “USER2” chain as shown in Figure 23.

By default, the Wizard's GT example design instantiates 6 control ports for its 5 VIO cores and 1 ILA core.

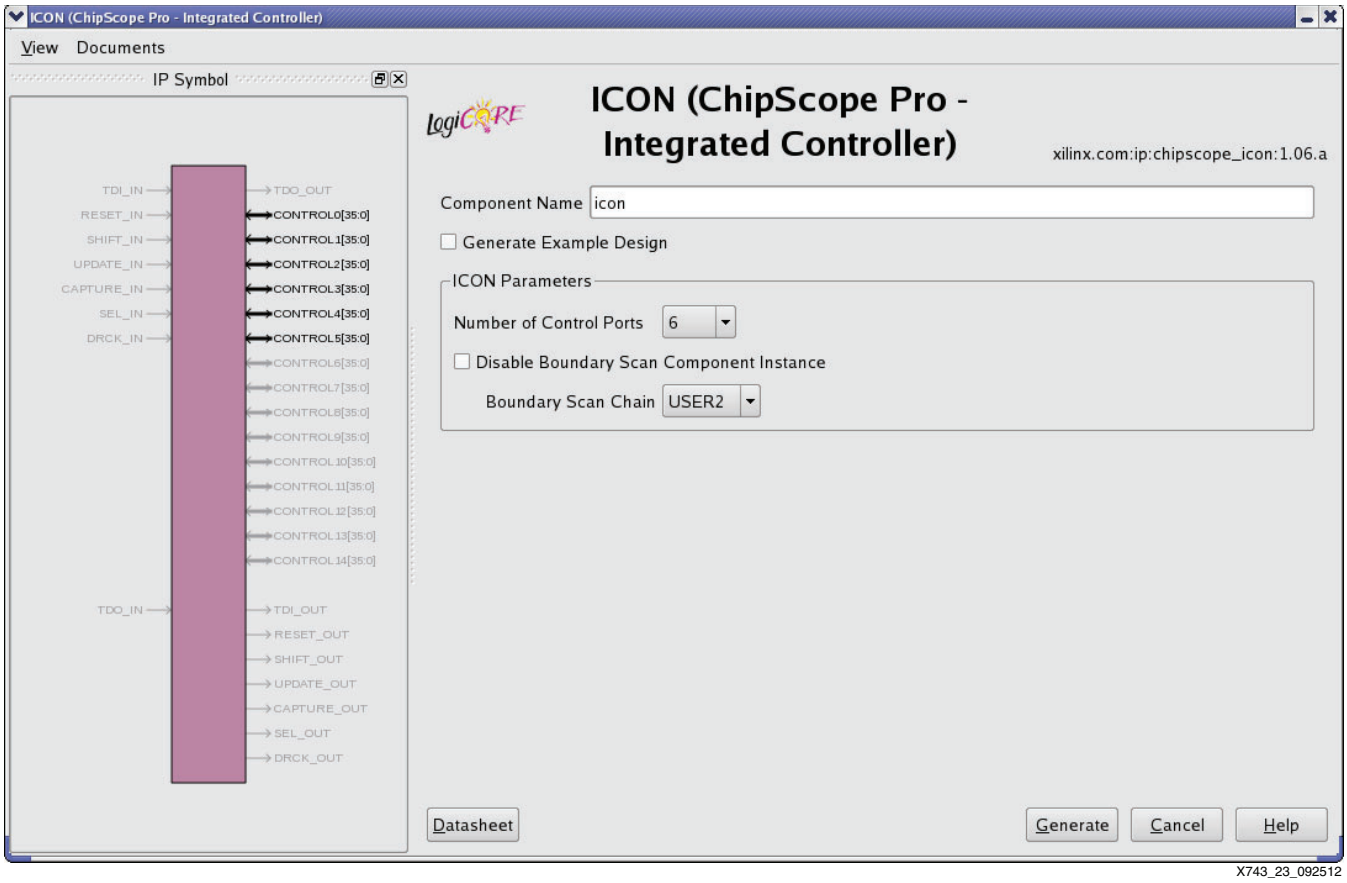


Figure 23: **Wizard ICON Core Boundary Scan Chain Setting**

Software Design

This section describes software design for performing Eye Scan. There are three sets of software included in the reference design:

- MATLAB code used to prototype the statistical eye algorithm
- MicroBlaze processor C code that includes the statistical eye algorithm, and code for managing data flow between the host computer and MCS
- TCL scripts for initiating Eye Scan from a host computer and for post-processing the data into a format for display on the Integrated Bit Error Ratio Test (IBERT) Plot Viewer

The following sections give detailed explanations of the software design. Source code discussed in these sections is included in the design files.

Statistical Eye Algorithm

Overview

Code that encapsulates the algorithm to acquire data needed to display a statistical eye for one or more transceiver lanes is described in this section. This code is represented by the block labeled es_simple_eye_acq in Figure 24.

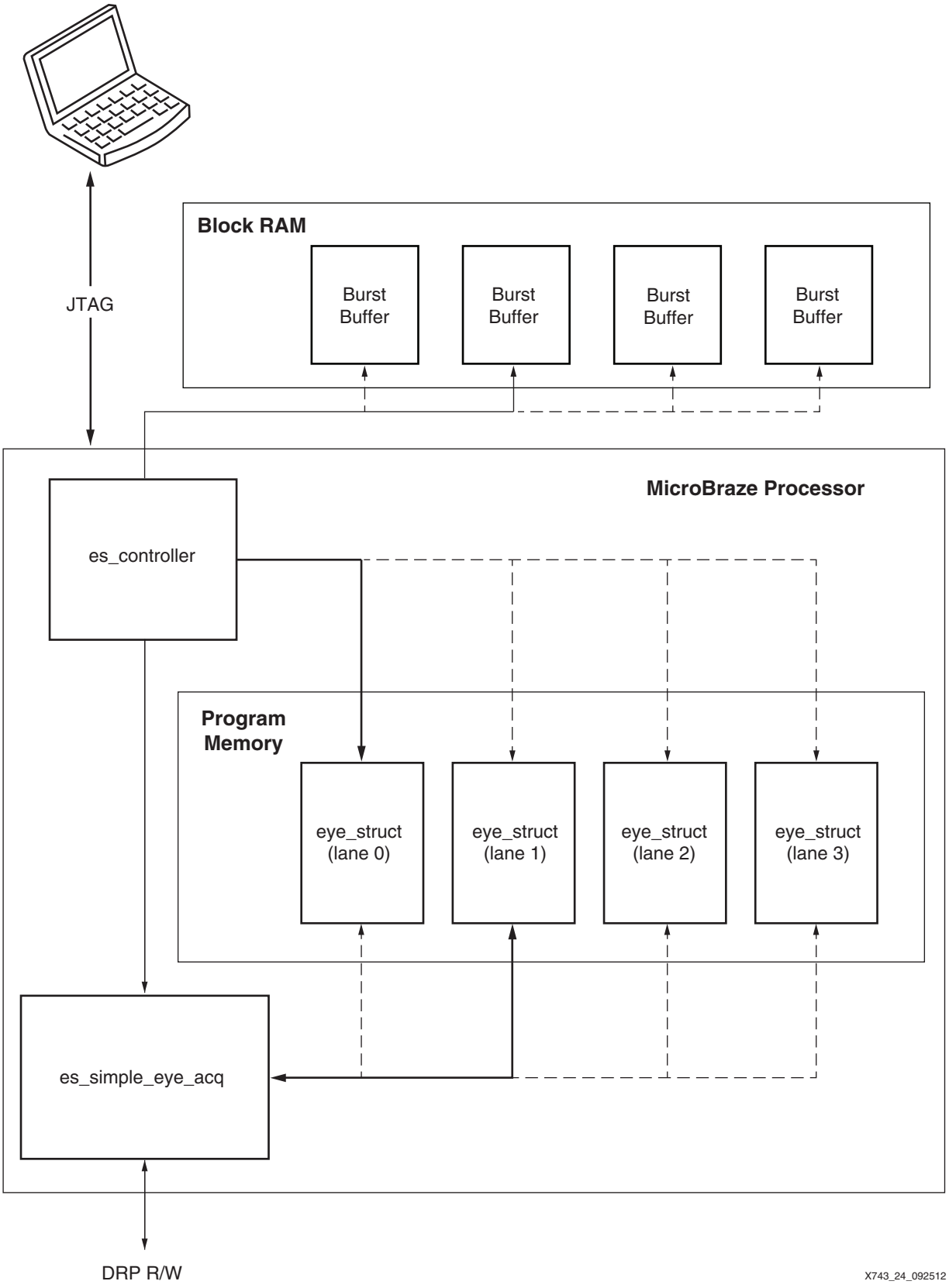


Figure 24: **Statistical Eye Algorithm**

The state for each of the lanes is maintained in a separate data structure in program memory (eye_struct in Figure 24). The controller code (es_controller) calls the algorithm code (es_simple_eye_acq), referencing the data structure for each of the MGT lanes in rotation. Because the majority of time in the algorithm is spent accumulating the error and sample counts needed to calculate BER for each pixel in the display, and because this accumulation is performed autonomously by hardware within each lane, data acquisition for multiple lanes proceeds virtually simultaneously.

The controller code, in addition to repetitively calling the algorithm code, polls the data_ready bit in each lane's data structure. When this is asserted by the algorithm, the controller copies the relevant data for one pixel of the statistical eye display to a buffer in a predefined region in block RAM. When the buffer is full, this data is burst-read by an external computer.

Statistical Eye Algorithm States

The states below describe the Statistical Eye algorithm, which is implemented in software and runs once to display each eye. These state names should not be confused with the states of the Eye Scan state machine that is implemented in hardware in the transceiver (as documented in *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)) and which is invoked by the Statistical Eye algorithm to measure each pixel in the statistical eye display.

- WAIT: If called in this state, the algorithm returns with no changes to the data structure.
- RESET: When called in this state, the algorithm initializes data structure members (horz_offset, vert_offset, and ut_sign) to begin a statistical eye acquisition. This state also writes the appropriate value to the ES_PRESCALE attribute. (Required settings for the attributes ES_EYE_SCAN_EN, ES_ERRDET_EN, ES_QUAL_MASK, and ES_SDATA_MASK are written by the controller code. See the RX Margin Analysis section of *7 Series FPGAs GTX/GTH Transceivers User Guide* for more information on the Eye Scan attributes.) When complete, RESET changes to the SETUP state.
- SETUP: This state increments data structure members horz_offset, vert_offset, and ut_sign to set up acquisitions at each pixel in the display, writing these values to the corresponding attributes. This state also starts error and sample count accumulation by asserting ES_CONTROL[0], changing the state to COUNT, and returning (permitting other lanes to be processed while these counts accumulate).
- COUNT: If called in this state, the algorithm returns unless the error and sample count accumulation has completed. If accumulation has completed, es_error_count, es_sample_count, and ES_PRESCALE attributes are read, and the values stored in the data structure and the data_ready flag are asserted. Depending on the relative values of the error and sample counts, the value of ES_PRESCALE can be adjusted to accommodate the very wide dynamic range of BER measurements. Finally, the state is changed to SETUP and the algorithm returns.

Detailed Code Description for es_simple_eye_acq

This code was developed in MATLAB, then automatically converted to C code and compiled. The functional code description in Table 2 references the MATLAB code in the design files.

Table 2: **eye_struct Data Structure**

eye_struct Member	Type	I/O	Description
state	uint16 (1x1)	I/O	Holds the present state of the statistical eye algorithm for this lane.
error_count	uint16 (1x1)	O	Accumulated error count (0 to 65535).
sample_count	uint16 (1x1)	O	Accumulated sample count (0 to 65535), scaled by ES_PRESCALE and bus width.

Table 2: **eye_struct Data Structure *(Cont'd)***

eye_struct Member	Type	I/O	Description
data_ready	logical (1x1)	I/O	Flag for controller to copy error_count, sample_count, prescale, ut_sign, vert_offset, and horz_offset to block RAM buffer.
lpm_mode	logical (1x1)	I	Input asserting whether RX is in decision feedback equalization (DFE) or longest prefix match (LPM) mode.
horz_step_size	int16 (1x1)	I	Multiple of unit counts incremented horizontally between pixels (e.g., set to 4 yields 0, ±4, ±8, ...)
vert_step_size	int16 (1x1)	I	Multiple of unit counts incremented vertically between pixels (e.g., set to 8 yields 0, ±8, ±16, …)
prescale	uint8 (1x1)	I/O	Value used to scale sample count. Actual bits sampled equals es_sample_count × bus_width × $2^{(1+prescale)}$.
max_prescale	uint8 (1x1)	I	Maximum allowed value of prescale. Determines the maximum bits sampled, and therefore, minimum measurable BER.
max_horz_offset	int16 (1x1)	I	+0.5 UI range. (32 at full rate, 64 at half rate, 128 at quarter rate, 256 at octal rate, and 512 at hex rate.)
horz_offset	int16 (1x1)	I/O	Horizontal offset counts for current pixel measurement.
vert_offset	int16 (1x1)	I/O	Vertical offset counts for current pixel measurement.
ut_sign	int16 (1x1)	I/O	Value of ut_sign for current pixel measurement. (For DFE, error and sample accumulation at 0 and 1 is required.)
lane_name	char (1x6)	I	'Lane_0', 'Lane_1', 'Lane_2' or 'Lane_3'
block_name	char (1x7)	I	'MGT_115', etc.
mode_name	char (1x9)	I	'hardware' or 'simulator' (Currently unused. Maintained for backward compatibility.)

Gear Shifting

Measurement of the BER across a statistical eye requires a considerable dynamic range. The number of bits sampled in one measurement is up to:

es_sample_count × bus_width × $2^{(1+ES\_PRESCALE)}$, where bus_width is 16, 20, 32, or 40.

Therefore, for a 32-bit bus width, a full sample_count of 65535 and ES_PRESCALE set to the maximum (31), $9.0 \times 10^{+15}$ bits can be sampled, enabling BER to be measured with high confidence down to 10^{-15}.

However, at that maximum ES_PRESCALE setting, BER for the outside region of the eye (BER > ~10^{-6}) could not be measured because the heavy prescaling would result in es_error_count saturating at 65535 and stopping the measurement while es_sample_count was still equal to 0, resulting in the calculation of an infinite BER. Also, many of these measurements would take a long acquisition time only to produce this meaningless result.

The inescapable conclusion is that ES_PRESCALE must be adjusted dynamically to measure statistical eyes with deep BER. (Without these dynamic adjustments, setting ES_PRESCALE to 0 would enable BER measurements down to about 10^{-6}, and setting ES_PRESCALE to 12

would enable BER measurements down to about 10^{-9}. Larger values of ES_PRESCALE would result in the BER divide-by-zero problem.)

The dynamic adjustment of ES_PRESCALE in the MATLAB code below is contained within the COUNT state between the “gear shifting start” and “gear shifting end” comment lines. This code adjusts ES_PRESCALE based upon the es_error_count from the previous measurement:

- If es_error_count is too large, ES_PRESCALE is decreased for the next measurement. The amount of decrease is proportional to the size of es_error_count. Because only a few errors need to be counted for a reasonably accurate, repeatable BER measurement, this saves considerable time. (Accumulating 65535 errors at a BER of 10^{-9} and a 10 Gb/s data rate takes 109 minutes for one pixel, while accumulating only 10 errors takes just 1 second.)
- If es_error_count is too small, ES_PRESCALE is increased for the next measurement. If es_error_count is zero or below the minimum to guarantee the desired resolution, the measurement is repeated with the new ES_PRESCALE value without incrementing the location.

The threshold parameters that determine “too large,” “too small,” and “desired resolution” are subjective, depending on the horizontal and vertical step sizes as well as the steepness of the BER change with position. The values in the code are reasonable, but optimal values would depend on the above parameters and BER steepness.

Predicting the best ES_PRESCALE setting for the next measurement based on the previous measured values assumes that the change in BER is relatively smooth. This strongly suggests that (for DFE mode), ut_sign should not be incremented by the inner loop because this would result in discontinuous changes in BER in successive measurements.

Eye Scan Controller Code

Running on the MCS, the C function es_controller manages Eye Scan measurements on multiple channels and handles flow of data between the transceivers and the host computer. Below is a description of its main tasks. Refer to the design files for source code.

Starting a Scan

The controller starts an Eye Scan measurement with test parameters specified by the host computer. These parameters include scan step size, rate mode, maximum prescale, parallel data width, and equalizer mode (DFE or LPM). The host computer writes these values into the data storage block RAM. The values are then loaded into the data structure (eye_struct) by es_controller at the beginning of the scan.

Running the Scan

The es_controller runs the Eye Scan by repeatedly calling es_simple_eye_acq for measuring error and sample counts across the eye. After error and sample counts for a particular horizontal and vertical offset location are complete, the controller stores the count values along with offset, prescale, and ut sign information to the data storage block RAM.

Storing and Uploading the Scan Data

After the data storage block RAM section for the particular lane is full, the controller pauses scan for that lane and indicates to the host computer that data is ready to be uploaded. After the upload is done, the controller resumes the scan. This process is repeated until the entire eye is measured.

Table 3 lists the contents of the data storage block RAM, which include both user-specified scan settings and data from Eye Scan measurement. Each transceiver lane occupies a separate section of the data storage block RAM. If only one lane’s section is full, the remaining lanes can continue to accumulate data.

Table 3: **Contents of One Lane's Data Storage Block RAM**

Item	Byte Offset (Hex)	Write Access	Read Access	Description
1	0x0000	Host computer	MCS	Test enable.
2	0x0004	Host computer	MCS	Maximum horizontal offset (32 for full rate, 64 for half rate, 128 for quarter rate, 256 for octal rate, 512 for hex rate).
3	0x0008	Host computer	MCS	Equalizer mode (1 for LPM, 0 for DFE).
4	0x000C	Host computer, MCS	Host computer, MCS	Upload ready (This is set to 1 by the MCS when the block RAM is full. It is set to 0 by the host PC when the upload is done).
5	0x0010	Host computer	MCS	Horizontal scan step size.
6	0x0012	Host computer	MCS	Parallel data width.
7	0x0014	Host computer	MCS	Vertical scan step size.
8	0x0016	Host computer	MCS	Maximum prescale value.
9	0x0018 – 0x07FF	MCS	Host computer	Eye Scan results.

Notes:

1. Refer to Table 1 for starting address for each lane.

Access to the transceiver DRP is handled by the drp_write and drp_read functions in `drp.c`. These functions translate attribute names into specific DRP addresses in the 7 series GTX transceiver.

TCL Code

TCL scripts are provided to execute tests from a host computer through the XMD shell (refer to *Embedded System Tools Reference Manual* (UG111) for more information on XMD). Two TCL files are provided:

- `es_pc_host.tcl`: This script contains functions for executing the scans and uploading data from the FPGA. In addition, it post-processes data to a format that is compliant with IBERT Plot Viewer and that can plot an ASCII pass/fail eye diagram.
- `test_es_pc_host.tcl`: This script contains an example script for running a scan and invoking post-processing functions. This file should be edited to specify scan parameters such as horizontal step size, vertical step size, maximum horizontal offset, maximum prescale, data width, and equalizer mode (DFE or LPM).

Resource Utilization

Table 4 gives the approximate resource utilization of the MCS, data storage block RAM, and accompanying Eye Scan logic on the Virtex-7 XC7VX485T-FFG1761 device.

Table 4: **Resource Utilization of Eye Scan Design**

LUTs	Flip-Flops
910	870

Notes:

1. This data excludes the logic utilization of the Wizard example design.

Extensions and Modifications

The reference design serves to demonstrate the use of an embedded processor to control 7 series device transceiver Eye Scan. This section discusses a few possible improvements to the design.

Scaling to More Channels

This section discusses hardware and software considerations when modifying the design for more than four lanes.

Hardware Considerations

- Size of data storage block RAM: The depth of the data storage block RAM affects the performance and area usage. As the number of channels increases, a larger block RAM is necessary to maintain scan speed. If the block RAM size is not increased, the data storage buffer would fill up more quickly, and would result in more frequent stalling of the scan.
- DRP or block RAM access controller: It is necessary to modify the drp_bridge module which serves to decode the MCS I/O bus address (Table 1) and properly directs read/write commands to either the data storage block RAM or one of the transceiver DRP ports. Only two address bits [12:11] are necessary to distinguish between the four transceiver lanes.

Software Considerations

- MCS address mapping: It is necessary to modify the MCS I/O bus address mapping in Table 1 to accommodate more lanes. This modification requires the following C code to be updated:
  - Constant definitions in the file `drp.h`: Add new DRP address offset constants, DRP_LANE*_OFFSET, for any additional lanes. DRP_LANE*_OFFSET corresponds to the starting DRP address for each lane (Table 1, items 1–4).
  - Function set_lane_offset in the file `drp.c`: Modify the switch statement to accommodate more lanes.
  - Constant definitions in the file `es_controller.h`: Modify these constant definitions:
    - Modify NUM_LANES to reflect the number of lanes to be scanned.
    - Modify LANE*_OFFSET to allocate portions of the data storage block RAM to additional lanes. LANE*_OFFSET corresponds to the starting address for each lane in the data storage block RAM (Table 1, items 5–8).
    - Modify DATA_STORE_OFFSET which corresponds to starting address of the data storage block RAM in the MCS address space (Table 1, item 5).
  - Function main in the file `es_controller.c`: Modify initialization of the arrays lane_offset, lane_names, and test_enable to accommodate additional lanes.
- TCL code:
  - Function es_host_run in the file `es_pc_host.tcl`: Modify LANE_OFFSET to reflect any changes in MCS address mapping. LANE_OFFSET corresponds to the starting address of each lane in the data storage block RAM (Table 1, items 5–8).
  - Function es_host_process_dump in the file `es_pc_host.tcl`: Modify the START_ADDRESS and END_ADDRESS to reflect any changes in MCS address mapping. These addresses correspond to the start and end address of the Eye Scan results section in the data storage block RAM (Table 1, items 5–8; Table 3, item 9).
    - Function es_host_check_range in the file `es_pc_host.tcl`: Modify NUM_LANES to reflect additional lanes.
    - Function run_test in `test_es_pc_host.tcl`: Update ch_list and the NUM_CH variable to reflect additional lanes.

Multiple MCS Instances

Another method of extending the design to multiple quads is by instantiating multiple MCS cores. For example, in a stacked silicon interconnect technology (SSIT) device, one MCS core can be instantiated in every super logic region (SLR). Refer to *LogiCORE IP MicroBlaze Micro Controller System* (DS865) for guidelines on using multiple MCS cores in the same design.

Alternative to XMD for Communicating with MCS

The reference design uses the XMD to send scan parameters and start-of-test signals to the MCS, as well as to offload scan results. Because communication between the XMD and MCS is done exclusively through the data storage block RAM (Table 3), it is possible to replace the functionality of the XMD with another controller by modifying the data storage block RAM to be a dual-port memory.

Figure 25 shows an example in which the user chooses to replace the XMD with an existing processor core. Port A of the dual-port memory is connected to the DRP or block RAM access control logic. Port B is connected to the data and address I/O bus of the existing processor.

The existing processor can start a scan and specify scan parameters by writing into the dual-port RAM (Table 3). By reading the upload-ready signal in the block RAM, the processor can poll the scan status and read the scan results when ready.

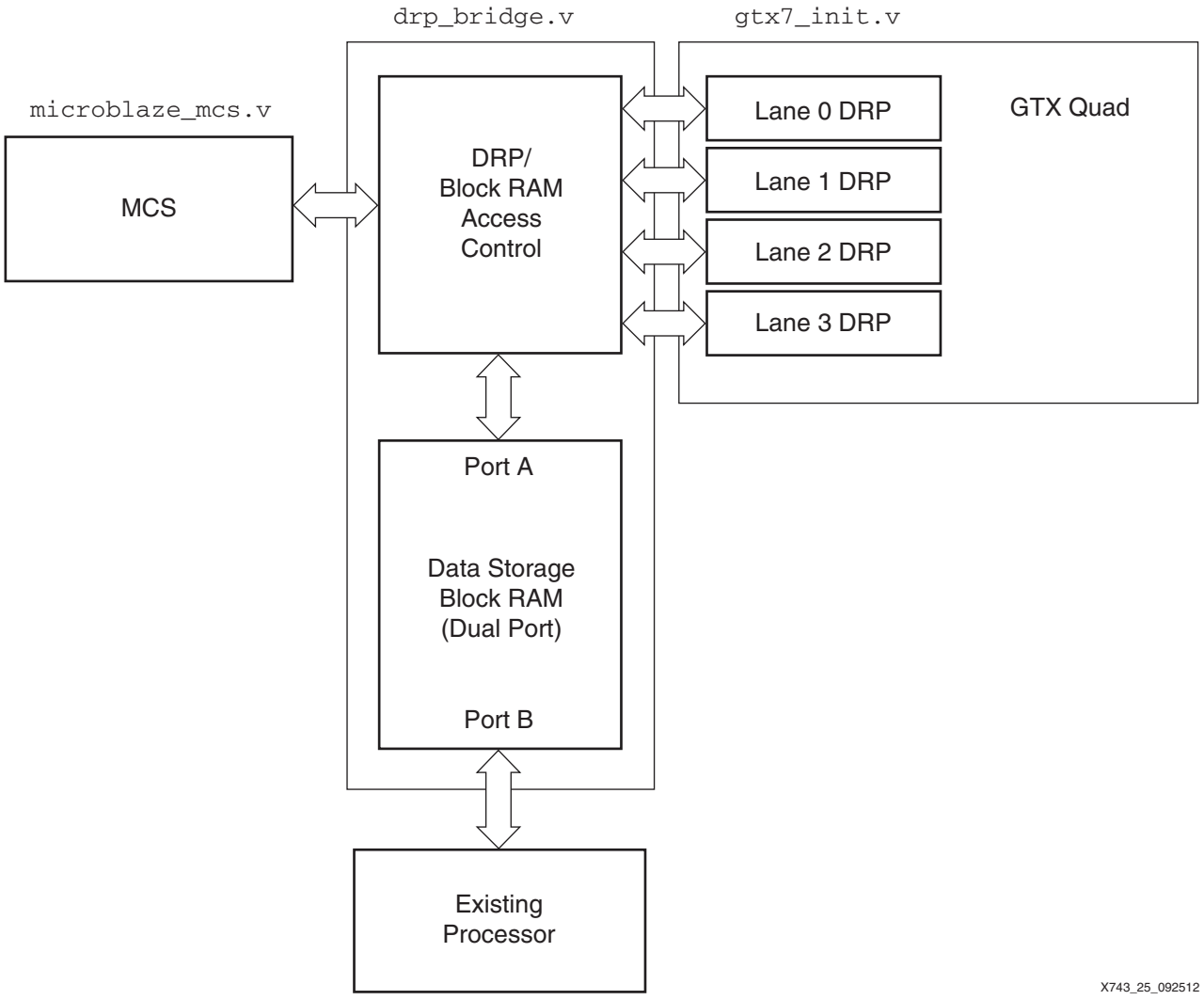


Figure 25: **Using an Existing Processor to Replace Functions of the XMD**

Reference Design

The reference design files for this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=194462.

The reference design checklist is shown in Table 5.

Table 5: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Xilinx
Target devices (stepping level, ES, production, speed grades)	7 series FPGAs with GTX transceivers
Source code provided	Yes
Source code format	Verilog
Design uses code and IP from existing Xilinx application note and reference designs, CORE Generator software, or third party	Yes
Simulation	
Functional simulation performed	Yes
Timing simulation performed	Yes
Test bench used for functional and timing simulations	No
Test bench format	N/A
Simulator software/version used	Synopsys VCS 2009.06
SPICE/IBIS simulations	No
Implementation	
Synthesis software tools/version used	XST 14.2
Implementation software tools/versions used	ISE® Design Suite 14.2
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	VC7203 board

Source Files

This section describes the source files included in the reference design. The following gives an overview of the relevant directory structure:

- `hw_design` (contains hardware design files)
  - `gtx7` (example design generated by 7 Series Transceiver Wizard)
    - `example_design` (example design files including Verilog code for Eye Scan)
    - `implement` (ISE® tools run directory)
      - `results` (ISE tools compilation results)
  - `microblaze_mcs` (CORE Generator tool files related to MCS)
  - `block_ram_2048x8` (CORE Generator tool files related to data storage block RAM)
- `sw_design` (contains software design files)

- matlab (MATLAB code)
- c_src (C code)
- SDK (SDK projects)
  - eye_scan_hw_spec (hardware specification project)
  - eye_scan_bsp (board support package project)
  - eye_scan_sw (C project)
- tcl (TCL code)

Hardware Design Source Files

The example design is tested with the XC7VX485T-FFG1761 device on Quad 113. Table 6 lists hardware design source files for bitstream generation. The ISE tools run directory is located in hw_design/gtx7/implement and compilation results are located in hw_design/gtx7/implement/results.

Table 6: **Hardware Source Files**

File Name	Path	Description
gtx7_exdes.v	hw_design/gtx7/example_design	Top-level module generated by the Wizard and then modified to include MCS and other logic.
drp_bridge.v	hw_design/gtx7/example_design	State machine for controlling DRP or block RAM access.
microblaze_mcs.v, microblaze_mcs.ngc, microblaze_mcs.bmm	hw_design	MCS generated by the CORE Generator tool.
block_ram_2048x8.v, block_ram_2048x8.ngc	hw_design	Data storage RAM generated by the CORE Generator tool.
icon.ngc	hw_design	ICON core using boundary scan 2 generated by the CORE Generator tool.
gtx7_init.v	hw_design/gtx7/example_design	Wrapper and GTX related files generated by the Wizard. The gtx7_gt.v is modified to set the PMA_RSV2[5] attribute to 1.
tx_startup_fsm.v		
rx_startup_fsm.v		
recclk_monitor.v		
tx_manual_phase_align.v		
auto_phase_align.v		
gtx7_gt_usrclk_source.v		
gtx7_gt_frame_gen.v		
gtx7_gt_frame_check.v		
ila.ngc		
data_vio.ngc		
gtx7_gt.v	hw_design	
gtx7.v	hw_design	
gtx7_exdes.ucf	hw_design/gtx7/example_design	User constraint file
Chipscope_project.cpj	hw_design/gtx7/implement	scope project file

Software Design Source Files

The MATLAB statistical Eye Scan algorithm is contained in `es_simple_eye_acq.m`, located in the `sw_design/matlab` folder. Table 7 lists the C code source files for the MicroBlaze processor. The files are located in the `sw_design/c_src` folder.

Table 7: **MicroBlaze C Code Source Files**

File Name	Path	Description
`es_controller.c` `es_controller.h`	`sw_design/c_src`	Controller code
`es_simple_eye_acq.c` `es_simple_eye_acq.c`		Eye Scan algorithm code
`drp.c drp.h`		DRP read/write code

The SDK projects are included for direct import into the SDK workspace (refer to *EDK Concepts, Tools, and Techniques* (UG683) for details). These folders are located in `sw_design/SDK`:

- eye_scan_hw_spec: Xilinx hardware platform specification project
- eye_scan_bsp: Xilinx board support package project
- eye_scan_sw: C project

Table 8 lists the TCL files to be run on the host computer's XMD shell. They are located in the `sw_design/tcl` folder.

Table 8: **TCL Code Source Files**

File Name	Path	Description
`es_pc_host.tcl`	`sw_design/tcl`	TCL code for executing Eye Scan from XMD and for post-processing results
`test_es_pc_host.tcl`		Example run file

Executing the Reference Design

This section describes how to restore and compile the software design in SDK, execute an Eye Scan, and display the results. The main steps are:

1. Compiling and Running the Software
2. Establishing the Link
3. Executing Eye Scan
4. Displaying a Statistical Eye Plot

Compiling and Running the Software

This section describes how to compile the software design using SDK.

Step 1: Specify the Workspace

Eclipse organizes projects within a folder called a workspace. In SDK, a workspace can only contain projects for one specific hardware platform. When SDK starts up, specify a folder to contain software projects for a particular hardware design.

Step 2: Import Archived SDK Projects

All required SDK projects are included in the `sw_design/SDK` folder and can be imported directly into SDK.

1. Select **File > Import** to open the Import Wizard.

2. Select **General > Existing Projects into Workspace**, then select **Next**.
3. Click on **Select root directory** then **Browse** to select **sw_design/SDK**. The Projects pane should list these three projects:
   - eye_scan_hw_spec
   - eye_scan_bsp
   - eye_scan_sw

   Click on **Select All** to import all projects. Verify that the **Copy projects into workspace** option is checked. Click **Finish**.

Note: If the bitstream or BMM files have been updated, it is necessary to update the `system.bit` and `system_bd.bmm` files in the `eye_scan_hw_spec` project.

Step 3: Compile the Software Application

The SDK project is set to automatically compile the projects and generate an ELF file. To verify that compilation has finished correctly, view the Console window and ensure that the “elfcheck passed” message is displayed.

Step 4: Download the Bitstream

To configure the FPGA directly from the SDK:

1. Select **Xilinx Tools > Program FPGA** to open the Program FPGA window.
2. Verify that the Bitstream field points to `system.bit` in the `eye_scan_hw_spec` project path. The BMM File field should point to `system_bd.bmm` in the `eye_scan_hw_spec` project path.
3. Under ELF File to Initialize in Block RAM, select the `eye_scan_sw.elf` file in the `eye_scan_sw` project path.
4. Select **Program**.

Establishing the Link

After configuring the FPGA, follow these steps to bring up the serial links:

1. Connect the four transceivers in the quad in a TX-to-RX loopback configuration.
2. Set the i_mcs_enable signal to High to enable the MCS. The signal is assigned to pin G41.
3. Open the ChipScope Pro analyzer.
4. Select **JTAG Chain** and select the appropriate download cable.
5. Select **File > Open Project** and browse to **chipscope_project.cpj**.
6. On the Project pane, expand **UNIT:0 MYVIO0 (VIO)**, then select **VIO Console**. The VIO Console 0 window should be highlighted. Toggle the gttxreset and gtrxreset pushbuttons to reset the TX and RX paths, respectively.
7. On the Project pane, expand UNIT:1 MYVIO1 (VIO) then select VIO Console. VIO Console 1 window should be highlighted. Toggle the txuserrdy push button to assert TX user ready signal.

   The txresetdone signal should go High (LED turns green).
8. On the Project pane, expand **UNIT:2 MYVIO2 (VIO)**, then select **VIO Console**. The VIO Console 2 window should be highlighted. Toggle the rxuserrdy pushbutton to assert the RX user ready signal.

   The rxresetdone signal should go High (LED turns green).
9. Select **JTAG Chain > Close Cable** to close the JTAG connection cable before executing Eye Scan from the XMD.

Executing Eye Scan

After the link is running, follow these steps to execute Eye Scan:

1. Open an MS DOS command shell.
2. To set proper environment variables, execute `settings32.bat` for a 32-bit system, or `settings64.bat` for a 64-bit system.

   The files are typically located in the `ISE_DS` directory in the ISE tools installation area.
3. Execute **`xmd`** to start the XMD shell.
4. Execute **`connect mb mdm`** to connect to the MicroBlaze processor.
5. Navigate to the directory where the TCL files are located.
6. Execute **`source test_es_pc_host.tcl`**.

   To change scan parameters such as scan step size and prescale, edit the run_test procedure in `test_es_pc_host.tcl`.
7. To begin the test, execute run_test.
8. Wait for the SCAN IS DONE message to be displayed for all channels under test.

   While running, the TCL code stops the processor to send scan settings and polls the data ready signals. Therefore, it is normal to see these messages:

   - `Processor stopped.`
   - `Processor started. Type "stop" to stop processor.`

To provide immediate results, a simple ASCII pass/fail eye diagram is displayed on the shell (Figure 26).

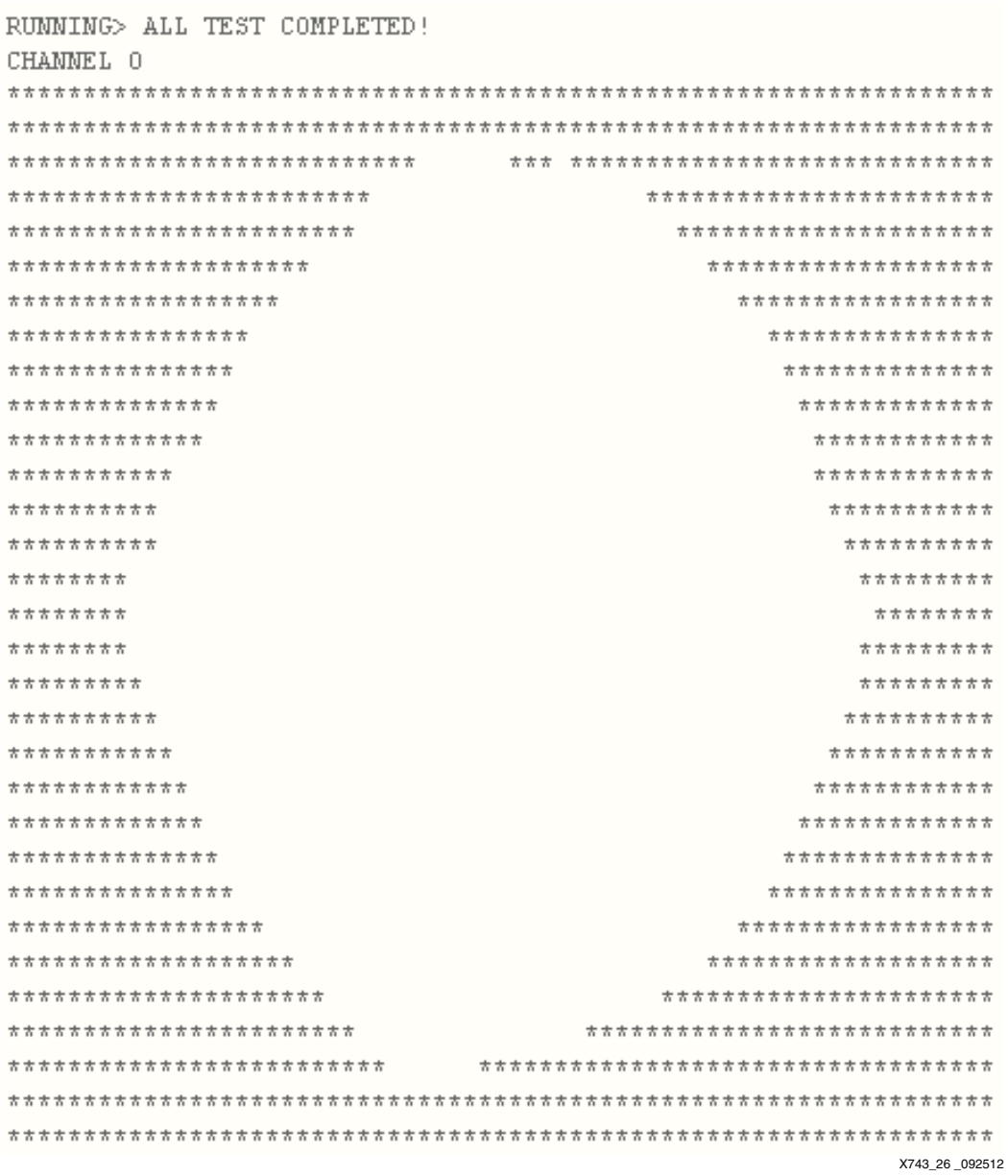


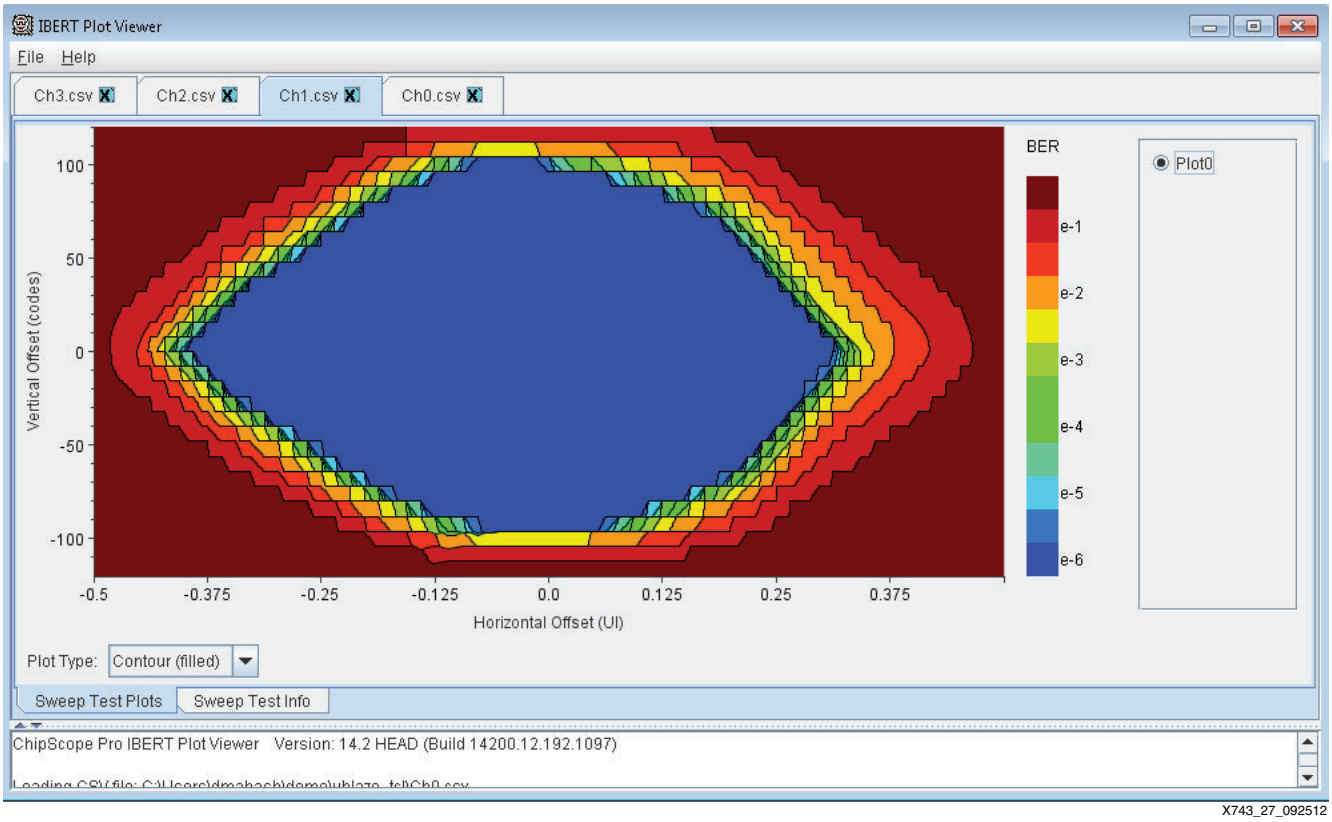
Figure 26: **ASCII Pass/Fail Eye Diagram**

Displaying a Statistical Eye Plot

Follow these steps to plot Eye Scan results using the IBERT Plot Viewer:

1. Open `ibertplotter.exe` to launch IBERT Plot Viewer.

   The executable is typically located in the `ISE_DS\ISE\bin\nt64` or `ISE_DS\ISE\bin\nt` directory in the ISE tools installation area.
2. Navigate to the Eye Scan run directory.
3. Select **File > Open CSV File**, then choose the CSV file generated by the Eye Scan to plot the statistical eye diagram. Default file names are `Ch0.csv`, `Ch1.csv`, `Ch2.csv`, `Ch3.csv`.



X743_27_092512

Figure 27: **Example Statistical Eye Plot in IBERT Plot Viewer**

Viewing the Raw Data

To view the raw data, open the CSV file in a spreadsheet application or text editor. The file format is described here:

- File header information is on lines 1–12. The relevant fields are:
  - samples per ui: Indicates the total number of horizontal sampling points in one UI. This is determined by the rate of the serial link: full, half, quarter, octal, or hex.
  - voltage interval: Indicates the scan vertical step size.
- Scan data start on line 14. The relevant columns are:
  - Voltage: Indicates vertical offset locations included in the scan.
  - RX Sampling Point (tap): Indicates horizontal offset locations included in the scan.
  - BER: Indicates the bit error ratio for the particular vertical and horizontal offset.

Creating New SDK Projects

This section provides instructions on creating new SDK projects. Because archived SDK projects are provided with this application note, it is not necessary to complete the steps in this section to run the demonstration (see Compiling and Running the Software, page 34).

Step 1: Specify the Workspace

Eclipse organizes projects within folders called workspaces. In SDK, a workspace can only contain projects for one specific hardware platform. When SDK starts up, specify a folder to contain software projects for a particular hardware design.

Step 2: Create Xilinx Hardware Platform Specification

To create a new SDK hardware project:

1. Select **File > New > Xilinx Hardware Platform Specification** to bring up the New Hardware Project window.
2. For Project name, use **eye_scan_hw_spec**.
3. Under Target Hardware Specification, select **Browse** and select **hw_design/microblaze_mcs_sdk.xml**.

   If the MCS core has been regenerated, select the newly generated XML file instead. The XML file is created in the directory where the CORE Generator tool is run.
4. Click **Bitstream and BMM Files** to specify bitstream and BMM files. For Bitstream, select **Browse** and select **hw_design\gtx7\implement\results\routed.bit**. For BMM File, select **Browse** and select **hw_design\gtx7\example_design\microblaze_mcs_bd.bmm**.

   If the bitstream has been regenerated, select the newly generated BMM and bitstream files instead. If compilation is run using the included `implement.sh` or `implement.bat` scripts, the bitstream would be generated in the `results` directory, while the BMM file would be generated in the `example_design` directory.
5. Click **Finish**.

Step 3: Create Xilinx Board Support Package

To create a new SDK board support package project:

1. Select **File > New > Xilinx Board Support Package** to bring up the New Board Support Package Project window.
2. For Project name, select **eye_scan_bsp**.
3. For the Hardware Platform, use **eye_scan_hw_spec**.
4. Under Board Support Package OS, select **standalone**.
5. Click **Finish**.
6. When the Board Support Package Settings window appears, select **OK** without making any changes.

Step 4: Create Xilinx C Project

To create a new SDK C project:

1. Select **File > New > Xilinx C Project** to bring up the New Project window.
2. For Project name, use **eye_scan_sw**.
3. For the Hardware Platform, use **eye_scan_hw_spec**.
4. For Target Software, use **Standalone**.
5. Under Select Project Template, select **Empty Application**.
6. Click **Next**.
7. Select **Target an existing Board Support Package**, then select **eye_scan_bsp {OS: standalone}**.
8. Select **Finish**.
9. In the Project Explorer pane, double-click **eye_scan_sw**. Right-click **src**, then select **Import** to bring up the Import window.
10. Double-click **General**, then select **File System**. Select **Next**.
11. Next to the From directory, select **Browse**. Select the **sw_design\c_src** directory. The following files should be listed in the window:
    - `drp.c`

- drp.h
- es_controller.c
- es_controller.h
- es_simple_eye_acq.c
- es_simple_eye_acq.h

12. Click **Select All** to import all of the above files. Verify that the Into folder points to eye_scan_sw/src.
13. Click **Finish**.

Conclusion

This application note showcases a MicroBlaze processor MCS-based design for executing Eye Scan on a 7 series FPGA GTX transceiver. This software-centric approach for implementing a statistical eye measurement provides a flexible and extensible method for incorporating Eye Scan capability into existing designs.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
10/18/12	1.0	Initial Xilinx release.
10/28/13	1.0.1	Updated description of Enable Debug Support and Enable MicroBlaze Trace Bus options in MCS. Corrected table reference in Hardware Design Source Files.

Notice of Disclaimer

Automotive Applications Disclaimer