XAPP1295 (v1.0) September 19, 2017

Automatic Insertion of Debug Logic for Transceivers in Synthesis DCP

Author: Erik Schidlack

Summary

This application note provides a method to insert debug logic into an existing design without the need to change the HDL code. The debug logic can be accessed in hardware manager with generated Tcl procedures from the Tcl console or virtual input/output (VIO) in the Vivado® Design Suite. The main focus in this document is on debugging transceiver designs, but the provided method can also be used on the flip-flops of a design without transceivers.

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

Introduction

To debug transceiver designs it is often necessary to observe or control ports of the transceiver or its supporting logic that are not easily accessible in the existing application design. Either the ports are not used at all or the logic is not easily changed, for example, an encrypted netlist of an IP core. Also, the transceiver instance is usually at a relatively low level of the hierarchy, and a change could require signals to cross this hierarchy. Some IP cores have the option to bring out a selection of transceiver ports for debug, but they might not be the right ones. All together, it would require a change of the existing design setup and HDL code, and this can be error prone and time consuming.

The Tcl script provided with this application note aims to make this process easier. Its starting point is a design checkpoint (DCP) written out after synthesis of the full application design for the device. No further design files are necessary, so the design setup and HDL code remain untouched. Every transceiver fabric port, except clock ports, can be chosen to be accessible in hardware manager through the Vivado tools Tcl console or VIO. The same control can be added for every flip-flop data port in the design.

To easily access the inserted debug logic through the Tcl console in hardware manager, a Tcl file is generated that provides Tcl procedures for every selected port. This makes it easy to set up more complex debug sequences. As an example, an eye scan script is already included.

In summary, the main features of the script are:

- Access to any transceiver fabric port and flip-flop data port for observation and control
- No change of HDL code necessary
- Access through Tcl console or VIO in HW manager
- Tcl procedures for easy access in Tcl console

Debug Logic Features

Figure 1 shows the general debug structure that is added to the design for selected signals.

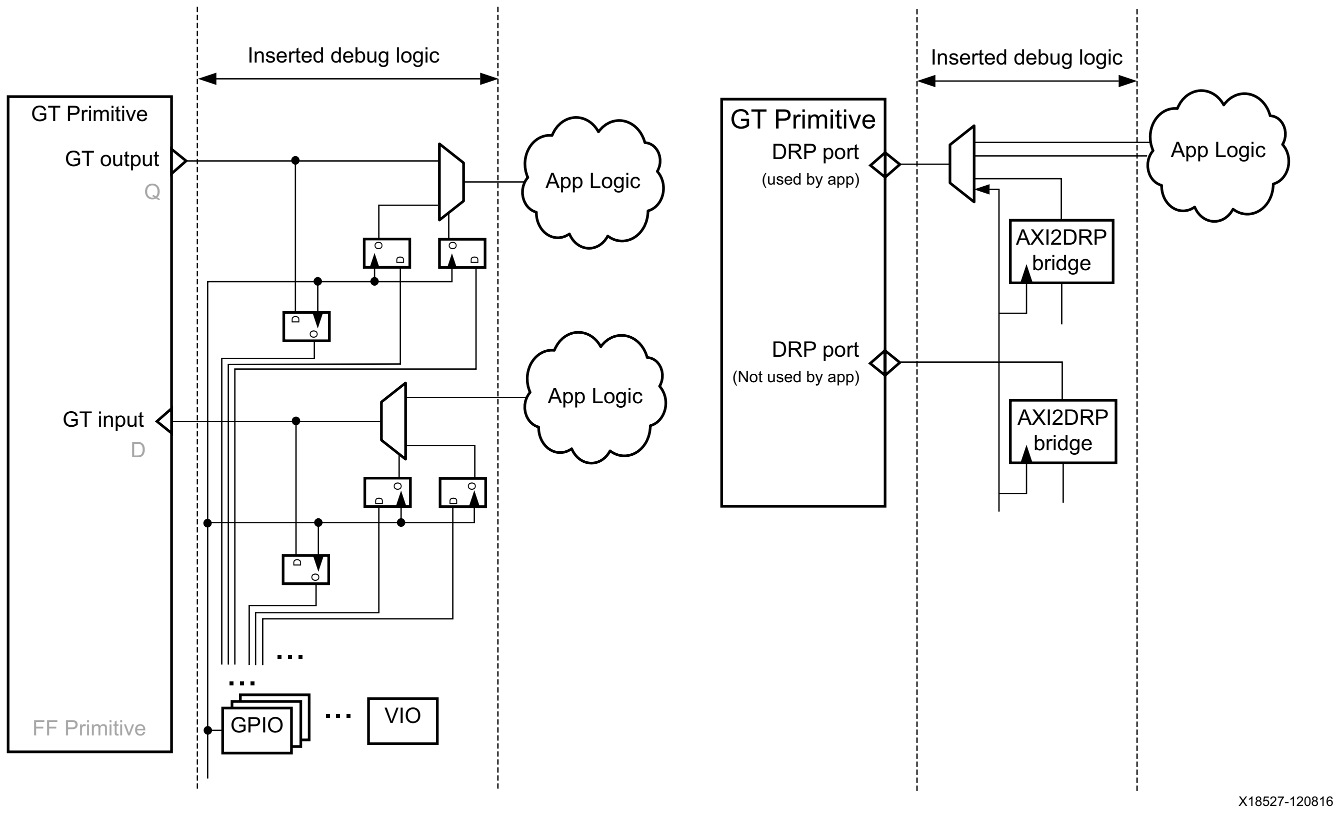


Figure 1: **Debug Logic Insertion**

Output Port Debug Logic

When selecting a transceiver or flip-flop output port for debug, the port value can be read back. A multiplexer is also inserted that can be controlled to either feed the port value or an externally set value to the application logic. This structure can be seen in the top left of Figure 1 and in more detail in Figure 2. The registers of the debug logic are clocked with the debug clock. The application clock is not used for synchronization.

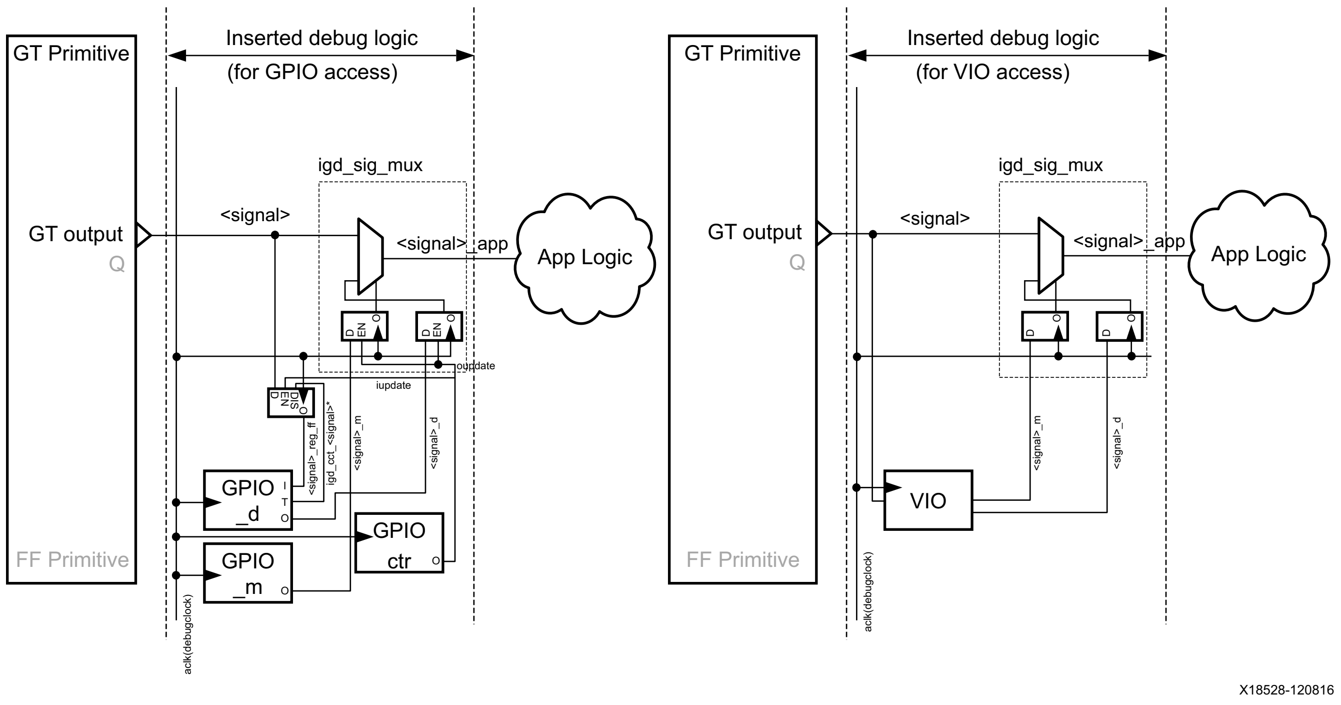


Figure 2: **Output Debug Logic**

Input Port Debug Logic

When selecting a transceiver or flip-flop input port for debug, a multiplexer is inserted that can be controlled to either feed the application logic value or an externally set value to the port. The port value can be read back. This structure can be seen in the bottom left of Figure 1 and in more detail in Figure 3. The registers of the debug logic are clocked with the debug clock. The application clock is not used for synchronization.

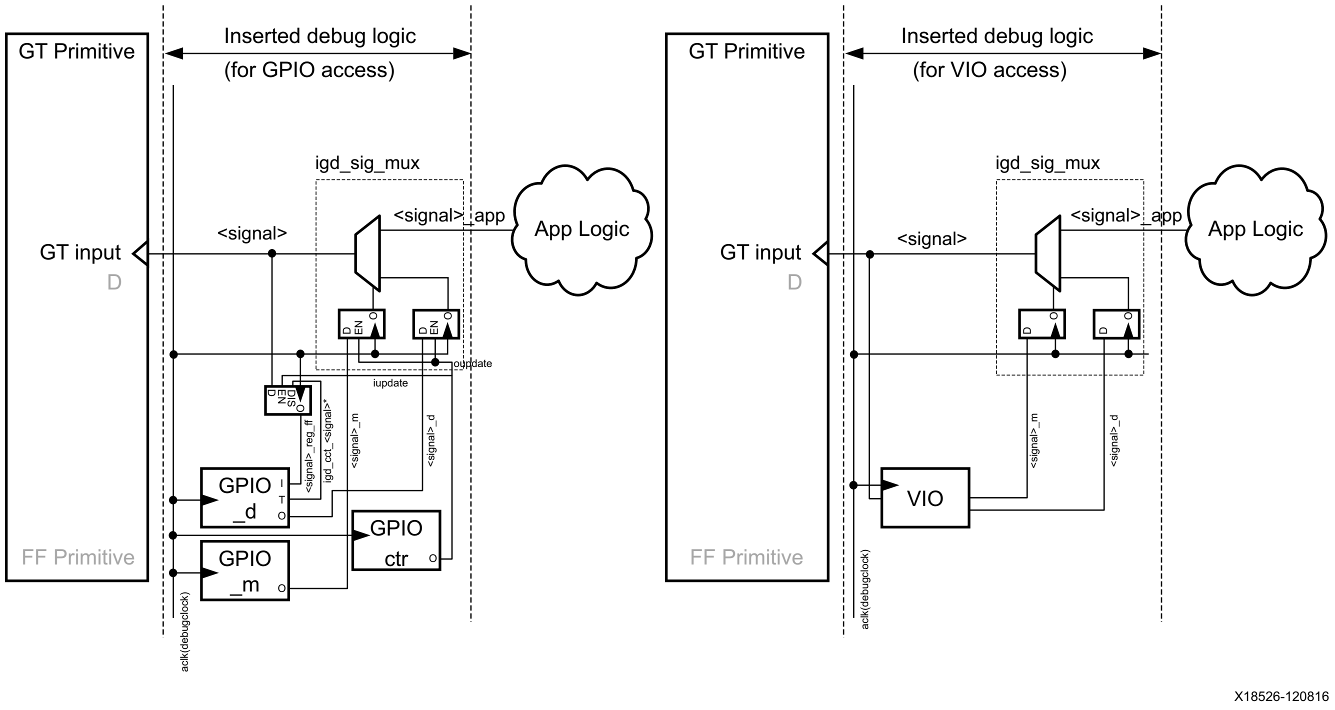


Figure 3: **Input Debug Logic**

Debug Logic Access

There are two possible ways to access the debug logic—through AXI general-purpose I/O (GPIO) registers or VIO. When using AXI GPIO registers, the debug logic can be controlled through the Tcl console in the Vivado tools. A JTAG2AXI and AXI interconnect is used to access the GPIO registers. The width and number of the GPIO registers that are used depends on the number of selected ports. If a port is a bus, its bits might be spread over several GPIO registers. To avoid reading or writing time split values from or to the port, some global control bits are introduced. Setting `oupdate` = 0 disables changes to the registers that are passed to the multiplexer. When the registers for the complete bus are written, setting `oupdate` = 1 passes the complete change to the multiplexer in one go. Similarly, on the read side `iupdate` = 0 allows the read of a complete bus value.

When using VIO, the debug logic can be controlled later directly though the standard GUI for the VIO. `Oupdate` and `iupdate` have no effect here.

DRP Port Debug Logic

The DRP port of a transceiver instance can only be selected as a full interface. If selected, an AXI2DRP bridge is implemented for this port. If an existing application connection to the port is detected, a dynamic reconfiguration port (DRP) arbiter (DRPMUX) is also added. This arbiter synchronizes the access from the AXI2DRP bridge to the DRP clock used in the application for this port. If the DRP access from the application interferes with the DRP access of the debug logic, a control bit (`drpappdis`) can be used to stop the application access. Each DRPMUX has its own `drpappdis` bit that can be controlled through the provided Tcl procedures.

ILA Cores within Debug Logic

Additionally, it is possible to add integrated logic analyzer (ILA) cores. Each selected DRP interface can have its own ILA core that is clocked with the DRP clock on that port. All other selected signals can be added to one ILA core that is clocked with the debug clock for the inserted logic.

Clock Changes

For certain transceiver clock inputs it is possible to connect them to the debug clock if they are unconnected in the application. The following is a list of clock inputs for which this is currently supported:

- DMONITORCLK
- CLKRSVD[n]
- CPLLLOCKDETCLK
- PLL0LOCKDETCLK
- PLL1LOCKDETCLK
- QPLLLOCKDETCLK
- QPLL0LOCKDETCLK
- QPLL1LOCKDETCLK
- TXLATCLK
- RXLATCLK
- SIGVALIDCLK
- TXPHDLYTSTCLK

Other clock connections cannot be changed.

Requirements and Limitations

There are some requirements for the debug clock described earlier. This clock must already be available in the application design and it must be free-running. Otherwise, the debug logic is not detected correctly in the hardware manager. If the frequency of this clock is outside the DRP limits of the used transceiver, the clock is divided by 2, 4, or 8, depending on which setting brings the frequency first into this limit. If there are already debug cores in the application, the clock for the existing debug hub has to be used, and it will be preselected.

There is one restriction for port selection. If the net connected to a selected port is connected to a debug core in the application (implying a DONT_TOUCH), this connection cannot be altered and the insertion might fail.

The number of ports that can be selected for GPIO or DRP control is restricted by the implemented AXI interconnect, which provides a maximum of 64 master interfaces that can be used. Each of these can be connected by either a GPIO register block, controlling up to 64 signal bits, or an AXI2DRP bridge. One interface is used for debug logic control through a GPIO register block, 5 bits are used for general control, and the remaining 59 bits can be used for `drpappdis` bits for a DRPMUX. So without usage of DRP a maximum of 63 x 64 signal bits can be controlled, which is reduced by 64 for each selected DRP. Without control of any signal, a maximum of 63 DRP interfaces can be controlled, of which a maximum of 59 can have a DRPMUX.

Debug Logic Insertion

This section describes the necessary steps for the debug logic insertion:

- Preparation: Get the files and software ready.
- Design Analysis: Run analysis on the design and prepare changes in the generated `insert_gt_dbg.do` file.
- Design Modification: Run the modification.
- Design Implementation: Implement the changed design.

Preparation

The first step for the debug insertion is to write out a DCP of the full application design after synthesis. Then, do the following:

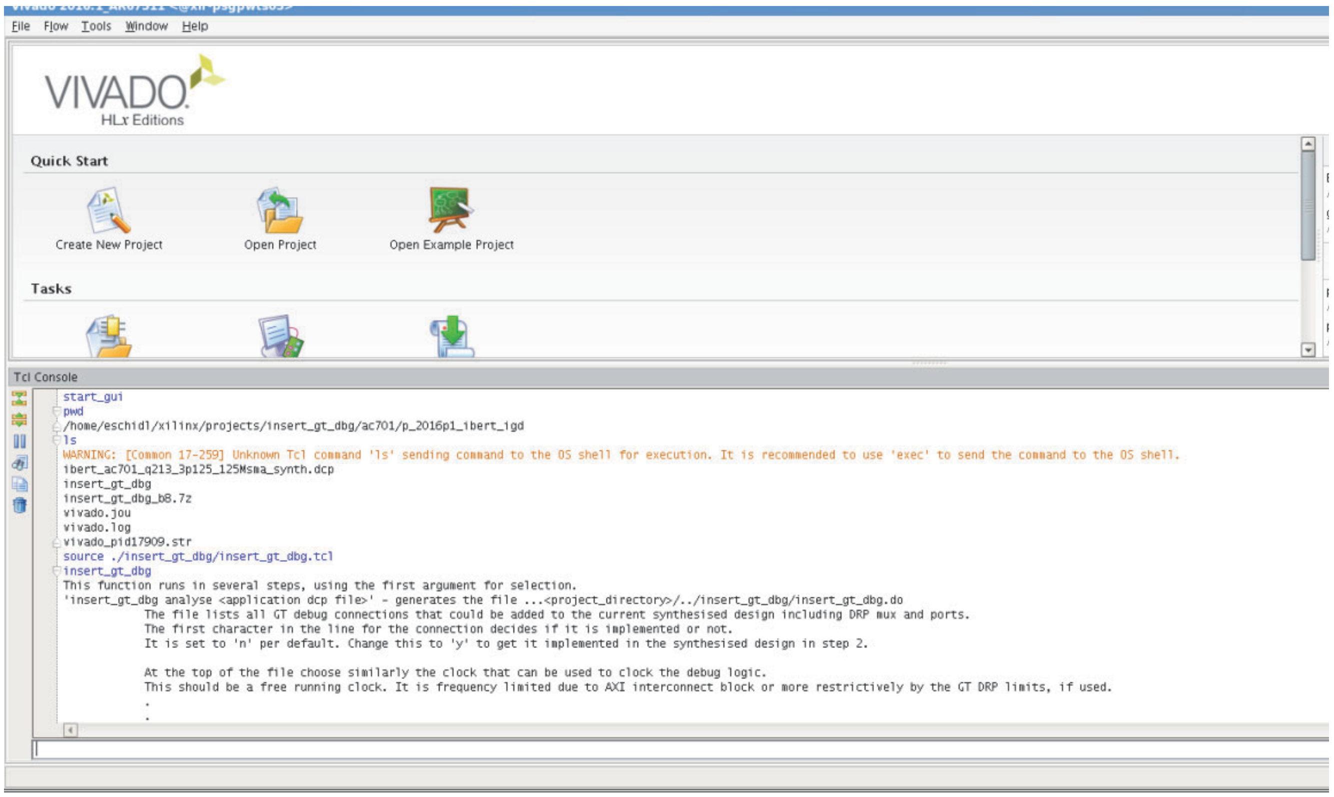
1. Create a new directory (`<dbgdir>`) that is not within an existing Vivado tools project structure and copy the DCP file into that directory.
2. Unpack the provided `insert_gt_dbg_<version>.7z` file in this directory.

   You should now have the following two entries in `<dbgdir>`:

   - The DCP file
   - A directory named `insert_gt_dbg`
3. Open the Vivado tools and make sure you are in the `<dbgdir>` directory in the Tcl console (`pwd`).
4. Execute the following command in the Tcl console:

   ```
   source ./insert_gt_dbg/insert_gt_dbg.tcl
   ```

You have now added the `insert_gt_dbg` command (Figure 4).



```
start_gui
pwd
/home/eschidl/xilinx/projects/insert_gt_dbg/ac701/p_2016p1_ibert_igd
ls
WARNING: [Common 17-259] Unknown Tcl command 'ls' sending command to the OS shell for execution. It is recommended to use 'exec' to send the command to the OS shell.
ibert_ac701_q213_3p125_125Msma_synth.dcp
insert_gt_dbg
insert_gt_dbg_b8.7z
vivado.jou
vivado.log
vivado_pid17909.str
source ./insert_gt_dbg/insert_gt_dbg.tcl
insert_gt_dbg
This function runs in several steps, using the first argument for selection.
'insert_gt_dbg analyse <application dcp file>' - generates the file ...<project_directory>/../insert_gt_dbg/insert_gt_dbg.do
            The file lists all GT debug connections that could be added to the current synthesised design including DRP mux and ports.
            The first character in the line for the connection decides if it is implemented or not.
            It is set to 'n' per default. Change this to 'y' to get it implemented in the synthesised design in step 2.

            At the top of the file choose similarly the clock that can be used to clock the debug logic.
            This should be a free running clock. It is frequency limited due to AXI interconnect block or more restrictively by the GT DRP limits, if used.
            .
            .
```

X18514-120816

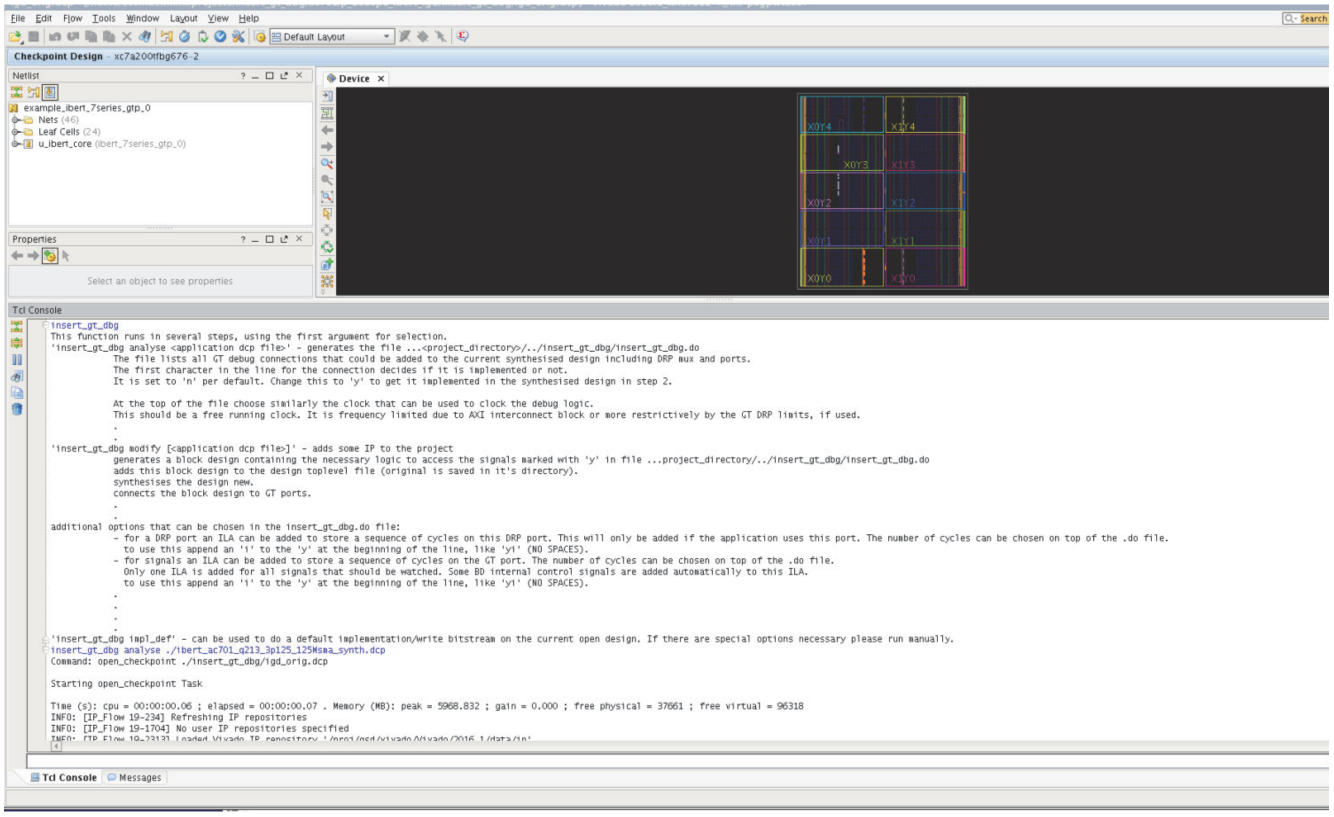
Figure 4: **Adding insert_gt_dbg**

Design Analysis

To analyze the application DCP file, execute the following command in the Tcl console:

```
insert_gt_dbg analyze <DCP file>
```

This opens the DCP file and extracts the necessary information to build the debug design. No change is done to the DCP (Figure 5). (If you need to come back to this step at a later stage, close all checkpoints/projects in the Vivado tools session beforehand.)


File Edit Flow Tools Window Layout View Help
Default Layout
Search
Checkpoint Design - xc7a200tfbg676-2
Netlist
example_ibert_7series_gtp_0
Nets (46)
Leaf Cells (24)
u_ibert_core (ibert_7series_gtp_0)
Properties
Select an object to see properties
Device
X0Y4
X1Y4
X0Y3
X1Y3
X0Y2
X1Y2
X0Y1
X1Y1
X0Y0
X1Y0
Tcl Console
Messages

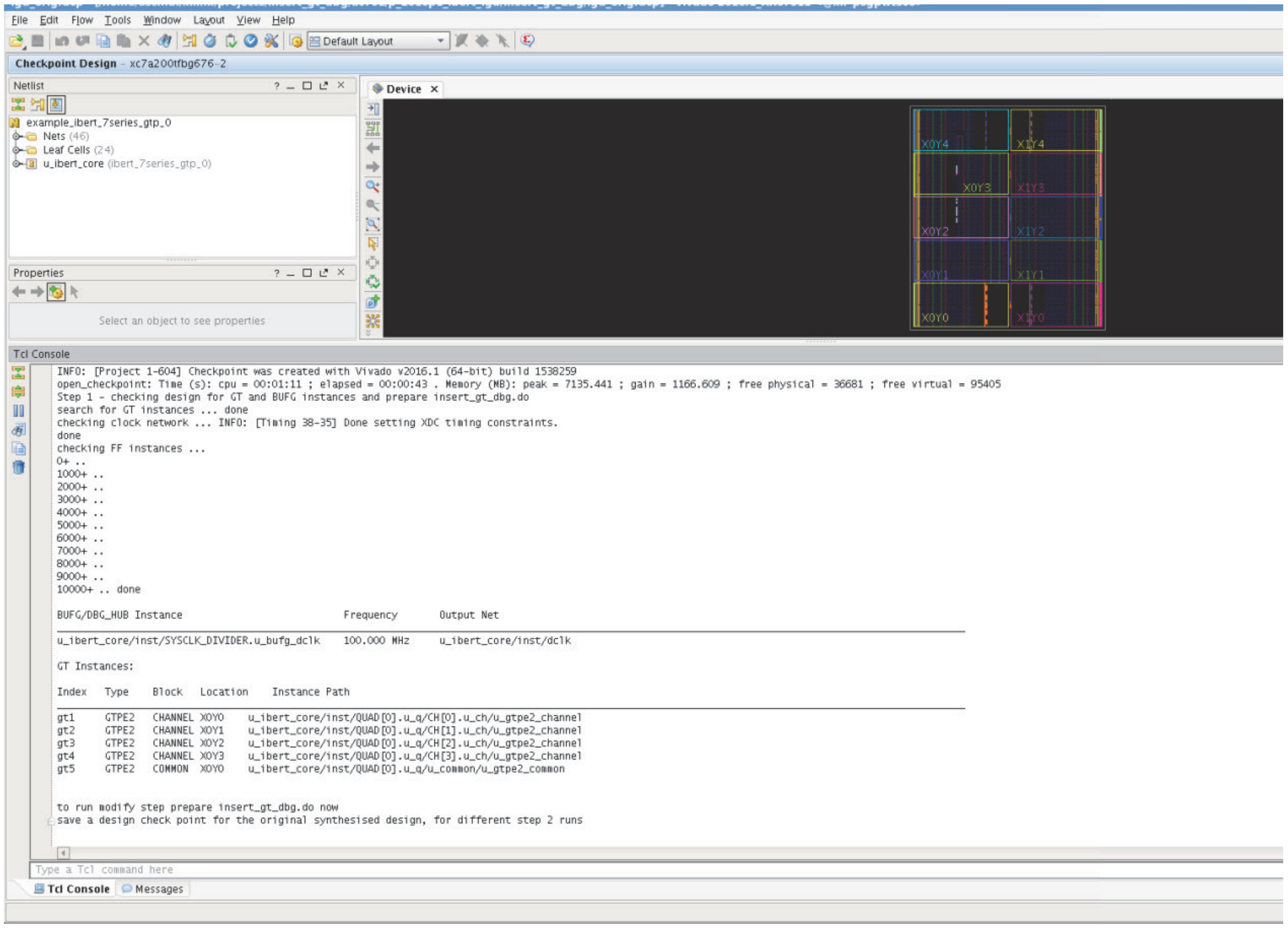
```
insert_gt_dbg
This function runs in several steps, using the first argument for selection.
'insert_gt_dbg analyse <application dcp file>' - generates the file ...<project_directory>/../insert_gt_dbg/insert_gt_dbg.do
            The file lists all GT debug connections that could be added to the current synthesised design including DRP mux and ports.
            The first character in the line for the connection decides if it is implemented or not.
            It is set to 'n' per default. Change this to 'y' to get it implemented in the synthesised design in step 2.

            At the top of the file choose similarly the clock that can be used to clock the debug logic.
            This should be a free running clock. It is frequency limited due to AXI interconnect block or more restrictively by the GT DRP limits, if used.
            .
            .
'insert_gt_dbg modify [<application dcp file>]' - adds some IP to the project
            generates a block design containing the necessary logic to access the signals marked with 'y' in file ...project_directory/../insert_gt_dbg/insert_gt_dbg.do
            adds this block design to the design toplevel file (original is saved in it's directory).
            synthesises the design new.
            connects the block design to GT ports.
            .
            .
additional options that can be chosen in the insert_gt_dbg.do file:
            - for a DRP port an ILA can be added to store a sequence of cycles on this DRP port. This will only be added if the application uses this port. The number of cycles can be chosen on top of the .do file.
              to use this append an 'i' to the 'y' at the beginning of the line, like 'yi' (NO SPACES).
            - for signals an ILA can be added to store a sequence of cycles on the GT port. The number of cycles can be chosen on top of the .do file.
              Only one ILA is added for all signals that should be watched. Some BD internal control signals are added automatically to this ILA.
              to use this append an 'i' to the 'y' at the beginning of the line, like 'yi' (NO SPACES).
            .
            .
            .
            .
'insert_gt_dbg impl_def' - can be used to do a default implementation/write bitstream on the current open design. If there are special options necessary please run manually.
insert_gt_dbg analyse ./ibert_ac701_q213_3p125_125Msma_synth.dcp
Command: open_checkpoint ./insert_gt_dbg/igd_orig.dcp

Starting open_checkpoint Task

Time (s): cpu = 00:00:00.06 ; elapsed = 00:00:00.07 . Memory (MB): peak = 5968.832 ; gain = 0.000 ; free physical = 37661 ; free virtual = 96318
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/proj/xsd/vivado/Vivado/2016.1/data/ip'
```

X18515-120816

Figure 5: **`insert_gt_dbg` Help and Start of Analyze Step**

When finished, you can see a list of all transceiver instances in the log and the following line (Figure 6):

```
To run the "modify" step, please edit "insert_gt_dbg.do" now
```


File Edit Flow Tools Window Layout View Help
Default Layout
Checkpoint Design - xc7a200tfbg676-2
Netlist
example_ibert_7series_gtp_0
Nets (46)
Leaf Cells (24)
u_ibert_core (ibert_7series_gtp_0)
Properties
Select an object to see properties
Device
X0Y4
X1Y4
X0Y3
X1Y3
X0Y2
X1Y2
X0Y1
X1Y1
X0Y0
X1Y0
Tcl Console
Type a Tcl command here
Tcl Console
Messages

```
INFO: [Project 1-604] Checkpoint was created with Vivado v2016.1 (64-bit) build 1538259
open_checkpoint: Time (s): cpu = 00:01:11 ; elapsed = 00:00:43 . Memory (MB): peak = 7135.441 ; gain = 1166.609 ; free physical = 36681 ; free virtual = 95405
Step 1 - checking design for GT and BUFG instances and prepare insert_gt_dbg.do
search for GT instances ... done
checking clock network ... INFO: [Timing 38-35] Done setting XDC timing constraints.
done
checking FF instances ...
0+ ..
1000+ ..
2000+ ..
3000+ ..
4000+ ..
5000+ ..
6000+ ..
7000+ ..
8000+ ..
9000+ ..
10000+ .. done

BUFG/DBG_HUB Instance                       Frequency       Output Net
---------------------------------------------------------------------------------
u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk    100.000 MHz     u_ibert_core/inst/dclk

GT Instances:

Index   Type    Block   Location    Instance Path
---------------------------------------------------------------------------------
gt1     GTPE2   CHANNEL X0Y0    u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/u_gtpe2_channel
gt2     GTPE2   CHANNEL X0Y1    u_ibert_core/inst/QUAD[0].u_q/CH[1].u_ch/u_gtpe2_channel
gt3     GTPE2   CHANNEL X0Y2    u_ibert_core/inst/QUAD[0].u_q/CH[2].u_ch/u_gtpe2_channel
gt4     GTPE2   CHANNEL X0Y3    u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/u_gtpe2_channel
gt5     GTPE2   COMMON  X0Y0    u_ibert_core/inst/QUAD[0].u_q/u_common/u_gtpe2_common


to run modify step prepare insert_gt_dbg.do now
save a design check point for the original synthesised design, for different step 2 runs
```

X18516-120816

Figure 6: **End of Analyze Step**

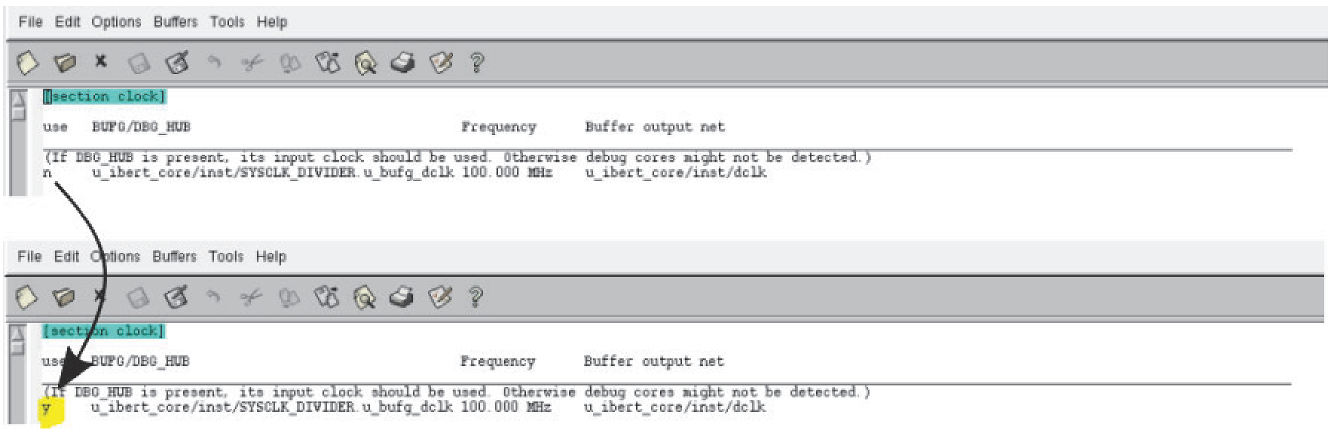
To run the modify step, open the file `<dbgdir>/insert_gt_dbg/insert_gt_dbg.do` in the text editor of your choice and make the necessary edits. You only need to change entries in the first column of a line. Leave everything else untouched.

The `.do` file is separated into several sections.

[section clock]

Here all clocks present in the application design are listed with their buffer instance, frequency, and clock net (Figure 7). If there is a debug hub already in the design, its clock is preselected with a `y` in the first column. Otherwise there will be an `n` in the first column.

Select only one clock from the list. It must be free running.


File Edit Options Buffers Tools Help
File Edit Options Buffers Tools Help

```
[section clock]
use   BUFG/DBG_HUB                                   Frequency      Buffer output net
(If DBG_HUB is present, its input clock should be used. Otherwise debug cores might not be detected.)
n     u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk 100.000 MHz     u_ibert_core/inst/dclk
```

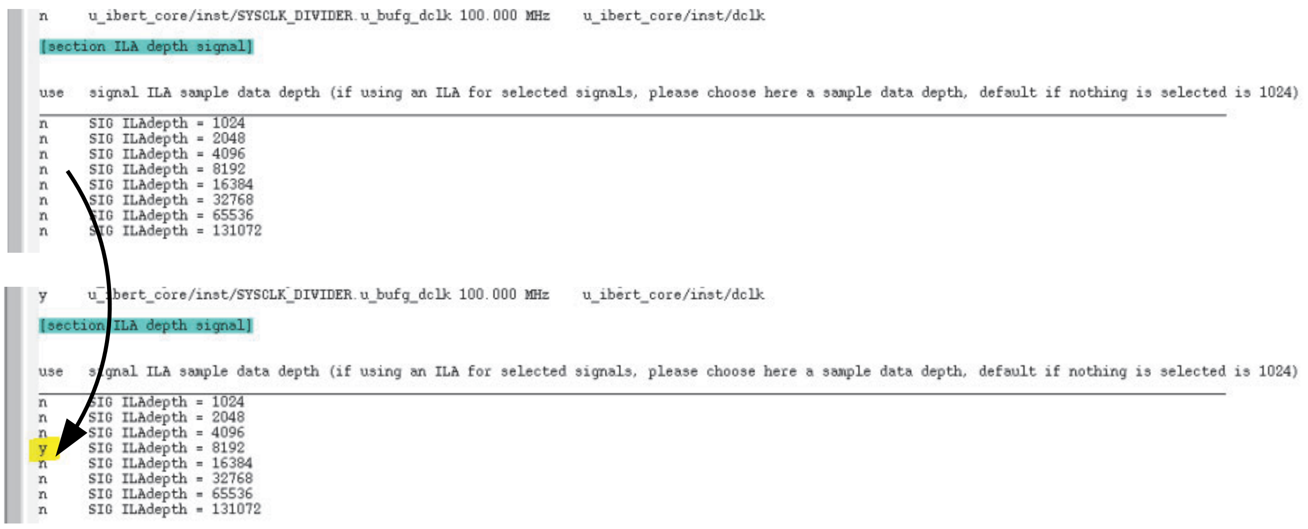
```
[section clock]
use   BUFG/DBG_HUB                                   Frequency      Buffer output net
(If DBG_HUB is present, its input clock should be used. Otherwise debug cores might not be detected.)
y     u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk 100.000 MHz     u_ibert_core/inst/dclk
```

X18517-120816

Figure 7: **Clock**

[section ILA depth signal]

This section allows you to select the data depth of the ILA core when choosing to add a signal to the signal ILA core (Figure 8). Change the first column from `n` to `y` for the selection. If two `y` are found, the last selection is taken. If nothing is selected, 1024 is used. The available resources in the design limit the ILA depth that you can select.



```
n     u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk 100.000 MHz     u_ibert_core/inst/dclk
[section ILA depth signal]

use   signal ILA sample data depth (if using an ILA for selected signals, please choose here a sample data depth, default if nothing is selected is 1024)
n     SIG ILAdepth = 1024
n     SIG ILAdepth = 2048
n     SIG ILAdepth = 4096
n     SIG ILAdepth = 8192
n     SIG ILAdepth = 16384
n     SIG ILAdepth = 32768
n     SIG ILAdepth = 65536
n     SIG ILAdepth = 131072
```

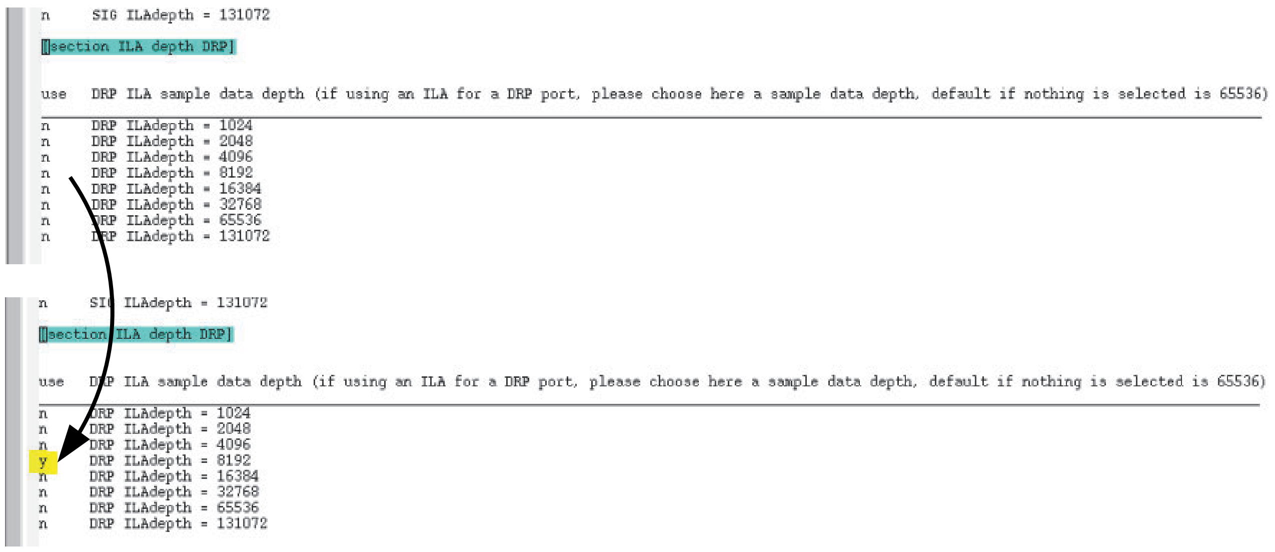
```
y     u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk 100.000 MHz     u_ibert_core/inst/dclk
[section ILA depth signal]

use   signal ILA sample data depth (if using an ILA for selected signals, please choose here a sample data depth, default if nothing is selected is 1024)
n     SIG ILAdepth = 1024
n     SIG ILAdepth = 2048
n     SIG ILAdepth = 4096
y     SIG ILAdepth = 8192
n     SIG ILAdepth = 16384
n     SIG ILAdepth = 32768
n     SIG ILAdepth = 65536
n     SIG ILAdepth = 131072
```

X18524-120816

Figure 8: **ILA Depth Signal**

[section ILA depth DRP]

This section allows you to select the data depth of the ILA core when choosing to add an ILA core for a DRP port (Figure 9). Change the first column from `n` to `y` for the selection. The last selection is taken. If nothing is selected, 65536 is used. The available resources in the application design might determine what you can select.



```
n      SIG ILAdepth = 131072
[section ILA depth DRP]

use    DRP ILA sample data depth (if using an ILA for a DRP port, please choose here a sample data depth, default if nothing is selected is 65536)
n      DRP ILAdepth = 1024
n      DRP ILAdepth = 2048
n      DRP ILAdepth = 4096
n      DRP ILAdepth = 8192
n      DRP ILAdepth = 16384
n      DRP ILAdepth = 32768
n      DRP ILAdepth = 65536
n      DRP ILAdepth = 131072
```

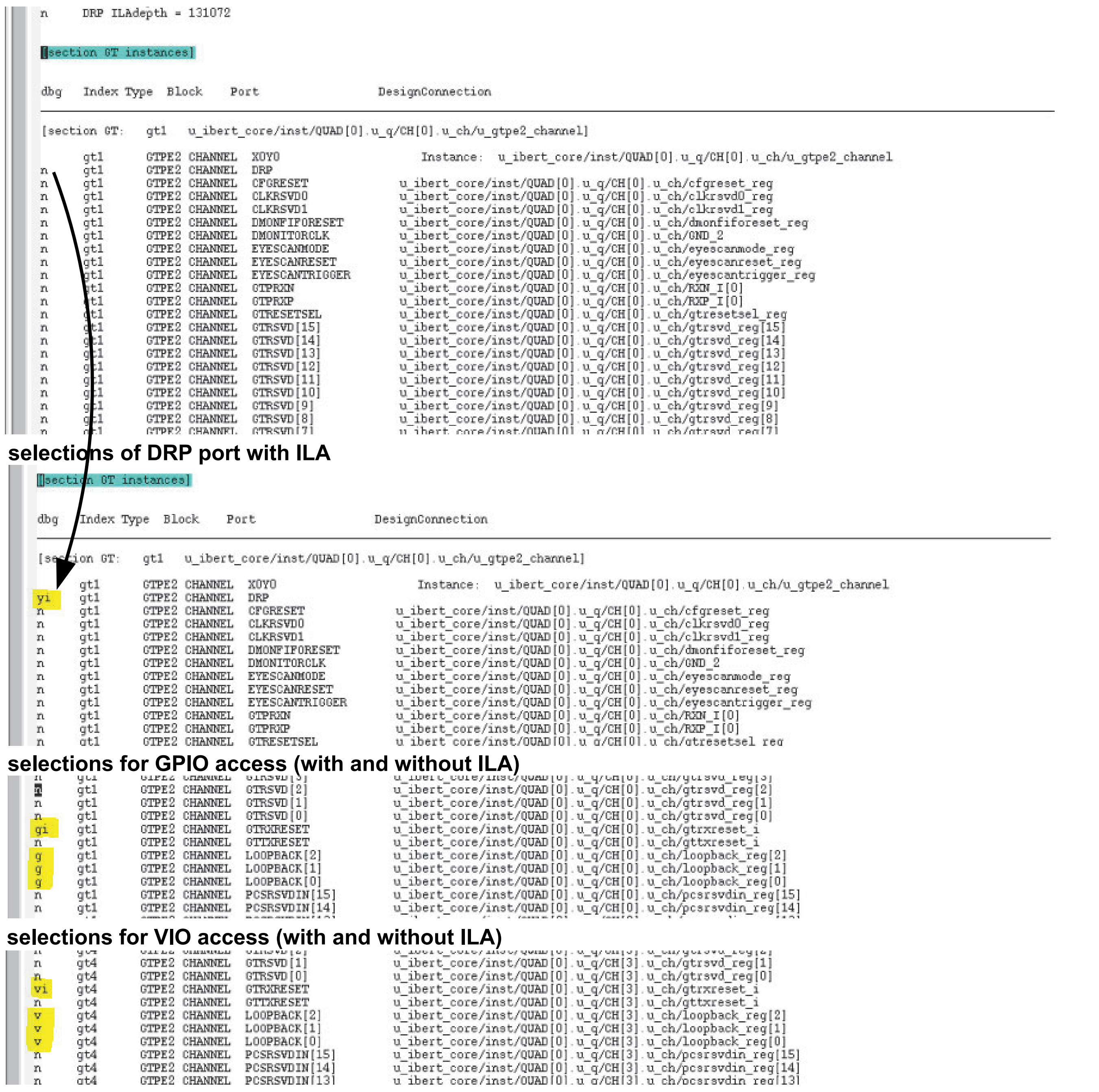
```
n      SIG ILAdepth = 131072
[section ILA depth DRP]

use    DRP ILA sample data depth (if using an ILA for a DRP port, please choose here a sample data depth, default if nothing is selected is 65536)
n      DRP ILAdepth = 1024
n      DRP ILAdepth = 2048
n      DRP ILAdepth = 4096
y      DRP ILAdepth = 8192
n      DRP ILAdepth = 16384
n      DRP ILAdepth = 32768
n      DRP ILAdepth = 65536
n      DRP ILAdepth = 131072
```

X18525-120816

Figure 9: **ILA Depth DRP**

[section GT instances]

This section is divided into subsections for every transceiver instance where you can do the port selections for signals that you want instrumented for debug (Figure 10).


selections of DRP port with ILA
selections for GPIO access (with and without ILA)
selections for VIO access (with and without ILA)
X18504-120816

```
n     DRP ILAdepth = 131072

[section GT instances]

dbg   Index Type  Block    Port                    DesignConnection
------------------------------------------------------------------------------------------------------
[section GT:   gt1   u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/u_gtpe2_channel]

      gt1      GTPE2 CHANNEL  XOYO                 Instance:  u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/u_gtpe2_channel
n     gt1      GTPE2 CHANNEL  DRP
n     gt1      GTPE2 CHANNEL  CFGRESET           u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/cfgreset_reg
n     gt1      GTPE2 CHANNEL  CLKRSVD0           u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/clkrsvd0_reg
n     gt1      GTPE2 CHANNEL  CLKRSVD1           u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/clkrsvd1_reg
n     gt1      GTPE2 CHANNEL  DMONFIFORESET      u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/dmonfiforeset_reg
n     gt1      GTPE2 CHANNEL  DMONITORCLK        u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/GND_2
n     gt1      GTPE2 CHANNEL  EYESCANMODE        u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/eyescanmode_reg
n     gt1      GTPE2 CHANNEL  EYESCANRESET       u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/eyescanreset_reg
n     gt1      GTPE2 CHANNEL  EYESCANTRIGGER     u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/eyescantrigger_reg
n     gt1      GTPE2 CHANNEL  GTPRXN             u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/RXN_I[0]
n     gt1      GTPE2 CHANNEL  GTPRXP             u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/RXP_I[0]
n     gt1      GTPE2 CHANNEL  GTRESETSEL         u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtresetsel_reg
n     gt1      GTPE2 CHANNEL  GTRSVD[15]         u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[15]
n     gt1      GTPE2 CHANNEL  GTRSVD[14]         u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[14]
n     gt1      GTPE2 CHANNEL  GTRSVD[13]         u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[13]
n     gt1      GTPE2 CHANNEL  GTRSVD[12]         u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[12]
n     gt1      GTPE2 CHANNEL  GTRSVD[11]         u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[11]
n     gt1      GTPE2 CHANNEL  GTRSVD[10]         u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[10]
n     gt1      GTPE2 CHANNEL  GTRSVD[9]          u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[9]
n     gt1      GTPE2 CHANNEL  GTRSVD[8]          u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[8]
n     gt1      GTPE2 CHANNEL  GTRSVD[7]          u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[7]
```

```
[section GT instances]

dbg   Index Type  Block    Port                    DesignConnection
------------------------------------------------------------------------------------------------------
[section GT:   gt1   u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/u_gtpe2_channel]

      gt1      GTPE2 CHANNEL  XOYO                 Instance:  u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/u_gtpe2_channel
yi    gt1      GTPE2 CHANNEL  DRP
n     gt1      GTPE2 CHANNEL  CFGRESET           u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/cfgreset_reg
n     gt1      GTPE2 CHANNEL  CLKRSVD0           u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/clkrsvd0_reg
n     gt1      GTPE2 CHANNEL  CLKRSVD1           u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/clkrsvd1_reg
n     gt1      GTPE2 CHANNEL  DMONFIFORESET      u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/dmonfiforeset_reg
n     gt1      GTPE2 CHANNEL  DMONITORCLK        u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/GND_2
n     gt1      GTPE2 CHANNEL  EYESCANMODE        u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/eyescanmode_reg
n     gt1      GTPE2 CHANNEL  EYESCANRESET       u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/eyescanreset_reg
n     gt1      GTPE2 CHANNEL  EYESCANTRIGGER     u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/eyescantrigger_reg
n     gt1      GTPE2 CHANNEL  GTPRXN             u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/RXN_I[0]
n     gt1      GTPE2 CHANNEL  GTPRXP             u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/RXP_I[0]
n     gt1      GTPE2 CHANNEL  GTRESETSEL         u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtresetsel_reg
```

```
n     gt1      GTPE2 CHANNEL  GTRSVD[3]          u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[3]
n     gt1      GTPE2 CHANNEL  GTRSVD[2]          u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[2]
n     gt1      GTPE2 CHANNEL  GTRSVD[1]          u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[1]
n     gt1      GTPE2 CHANNEL  GTRSVD[0]          u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrsvd_reg[0]
gi    gt1      GTPE2 CHANNEL  GTRXRESET          u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gtrxreset_i
n     gt1      GTPE2 CHANNEL  GTTXRESET          u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/gttxreset_i
g     gt1      GTPE2 CHANNEL  LOOPBACK[2]        u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/loopback_reg[2]
g     gt1      GTPE2 CHANNEL  LOOPBACK[1]        u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/loopback_reg[1]
g     gt1      GTPE2 CHANNEL  LOOPBACK[0]        u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/loopback_reg[0]
n     gt1      GTPE2 CHANNEL  PCSRSVDIN[15]      u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/pcsrsvdin_reg[15]
n     gt1      GTPE2 CHANNEL  PCSRSVDIN[14]      u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/pcsrsvdin_reg[14]
```

```
n     gt4      GTPE2 CHANNEL  GTRSVD[2]          u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/gtrsvd_reg[2]
n     gt4      GTPE2 CHANNEL  GTRSVD[1]          u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/gtrsvd_reg[1]
n     gt4      GTPE2 CHANNEL  GTRSVD[0]          u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/gtrsvd_reg[0]
vi    gt4      GTPE2 CHANNEL  GTRXRESET          u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/gtrxreset_i
n     gt4      GTPE2 CHANNEL  GTTXRESET          u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/gttxreset_i
v     gt4      GTPE2 CHANNEL  LOOPBACK[2]        u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/loopback_reg[2]
v     gt4      GTPE2 CHANNEL  LOOPBACK[1]        u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/loopback_reg[1]
v     gt4      GTPE2 CHANNEL  LOOPBACK[0]        u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/loopback_reg[0]
n     gt4      GTPE2 CHANNEL  PCSRSVDIN[15]      u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/pcsrsvdin_reg[15]
n     gt4      GTPE2 CHANNEL  PCSRSVDIN[14]      u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/pcsrsvdin_reg[14]
n     gt4      GTPE2 CHANNEL  PCSRSVDIN[13]      u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/pcsrsvdin_reg[13]
```

Figure 10: **GT Instances**

[section GT: <GT index> <instance name>]

`<GT index>`: This is the index used by the tool consisting of gt<integer>.

`<instance name>`: This gives the instance name used in the application.

The first line after the `[section …]` shows spaces as the first characters on the line:

```
<GT index> <GT type> <GT block> <GT location> Instance: <instance name>
```

`<GT type>`: For example GTPE2 or GTYE4.

`<GT block>`: Can be CHANNEL or COMMON.

`<GT location>`: Gives the device location for the instance, for example X0Y3. If not available in the application design, it is set to undef.

Nothing should to be done to this line. It is there for information purposes.

The next line allows you to select the DRP port.

```
n     <GT index> <GT type> <GT block> DRP
```

To do so change `n` to `y`. To add an additional ILA port for this DRP port, set it to `yi`.

After this point, all transceiver ports for the current transceiver instance are listed. The list is sorted alphabetically after the port name with the inputs first and then the outputs.

```
n     <GT index> <GT type> <GT block> <GT port name> <application net name>
```

`<application net name>`: Shows the name of the net connected to the port in the application design. If empty, the port is not connected in the application design.

To select the port to be controlled through the GPIO, change `n` to `g`. To add the port to the signal ILA, set it to `gi`.

To select the port to be controlled through the VIO, change `n` to `v`. To add the port to the signal ILA, set it to `vi`.

To select a clock port to be connected with the debug clock, change `n` to `c`. For the DMONITOR clock input only, if it is necessary to connect the DMONITOR clock input with the clock provided from the transceiver, change `n` to `ci` (DMONITOROUT should also be selected from bit 0 upwards. A separate ILA is added for it, running with the internal clock).

[section ILA hierarchies]

Separate to the above debug logic, this section allows insertion of ILA cores depending on the application design hierarchy (Figure 11). If a hierarchy is selected all flip-flop data ports in that hierarchy are added to an ILA, except for flip-flops that have the ASYNC_REG property set. Inputs and outputs can be selected separately.

The top of the section lists the clocks used in the design:

```
n                <pin driving the clock>            <clock net>
```

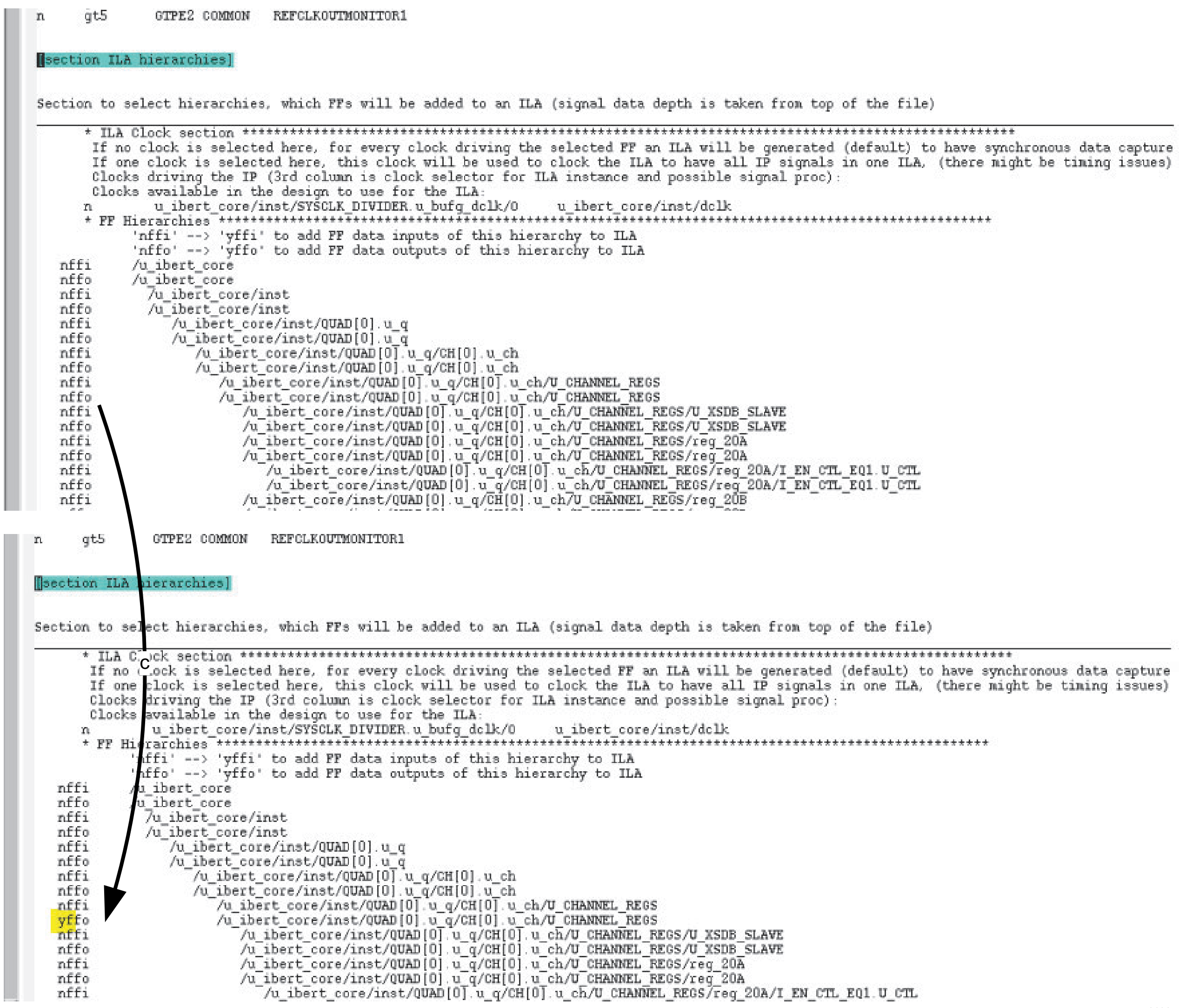
If you want to have all resulting signals in one ILA, change the `n` to `y` in the line of the clock you want to use for the ILA.

If no clock is selected here, the default behavior is to check for every flip-flop what clock it is related to. For every clock that is used for the selected hierarchies, a separate ILA is created to allow synchronous data capture in the ILA.

The hierarchy is listed after the clocks. Each hierarchy has two lines, one for flip-flop inputs and one for flip-flop outputs. The varying indentation between the lines shows the level of hierarchy. An example is shown here:

```
nffi                    /gt_exdes_top
nffo                    /gt_exdes_top
nffi                      /gt_exdes_top/example_wrapper_inst
nffo                      /gt_exdes_top/ example_wrapper_inst
nffi                         /gt_exdes_top/example_wrapper_inst/common_wrapper_inst
nffo                         /gt_exdes_top/example_wrapper_inst/common_wrapper_inst
```

To select a hierarchy, change `nffi` to `yffi` or `nffo` to `yffo`. Do not remove the leading space characters in the line.



```
n     gt5      GTPE2 COMMON   REFCLKOUTMONITOR1

[section ILA hierarchies]

Section to select hierarchies, which FFs will be added to an ILA (signal data depth is taken from top of the file)
      * ILA Clock section *********************************************************************************
       If no clock is selected here, for every clock driving the selected FF an ILA will be generated (default) to have synchronous data capture
       If one clock is selected here, this clock will be used to clock the ILA to have all IP signals in one ILA, (there might be timing issues)
       Clocks driving the IP (3rd column is clock selector for ILA instance and possible signal proc):
       Clocks available in the design to use for the ILA:
      n        u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk/0     u_ibert_core/inst/dclk
      * FF Hierarchies ***************************************************************************
            'nffi' --> 'yffi' to add FF data inputs of this hierarchy to ILA
            'nffo' --> 'yffo' to add FF data outputs of this hierarchy to ILA
   nffi     /u_ibert_core
   nffo     /u_ibert_core
   nffi       /u_ibert_core/inst
   nffo       /u_ibert_core/inst
   nffi          /u_ibert_core/inst/QUAD[0].u_q
   nffo          /u_ibert_core/inst/QUAD[0].u_q
   nffi             /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch
   nffo             /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch
   nffi                /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS
   nffo                /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS
   nffi                   /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/U_XSDB_SLAVE
   nffo                   /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/U_XSDB_SLAVE
   nffi                   /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/reg_20A
   nffo                   /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/reg_20A
   nffi                      /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/reg_20A/I_EN_CTL_EQ1.U_CTL
   nffo                      /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/reg_20A/I_EN_CTL_EQ1.U_CTL
   nffi                   /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/reg_20B
```

```
n     gt5      GTPE2 COMMON   REFCLKOUTMONITOR1

[section ILA hierarchies]

Section to select hierarchies, which FFs will be added to an ILA (signal data depth is taken from top of the file)
      * ILA Clock section *********************************************************************************
       If no clock is selected here, for every clock driving the selected FF an ILA will be generated (default) to have synchronous data capture
       If one clock is selected here, this clock will be used to clock the ILA to have all IP signals in one ILA, (there might be timing issues)
       Clocks driving the IP (3rd column is clock selector for ILA instance and possible signal proc):
       Clocks available in the design to use for the ILA:
      n        u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk/0     u_ibert_core/inst/dclk
      * FF Hierarchies ***************************************************************************
            'nffi' --> 'yffi' to add FF data inputs of this hierarchy to ILA
            'nffo' --> 'yffo' to add FF data outputs of this hierarchy to ILA
   nffi     /u_ibert_core
   nffo     /u_ibert_core
   nffi       /u_ibert_core/inst
   nffo       /u_ibert_core/inst
   nffi          /u_ibert_core/inst/QUAD[0].u_q
   nffo          /u_ibert_core/inst/QUAD[0].u_q
   nffi             /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch
   nffo             /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch
   nffi                /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS
   yffo                /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS
   nffi                   /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/U_XSDB_SLAVE
   nffo                   /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/U_XSDB_SLAVE
   nffi                   /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/reg_20A
   nffo                   /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/reg_20A
   nffi                      /u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/reg_20A/I_EN_CTL_EQ1.U_CTL
```

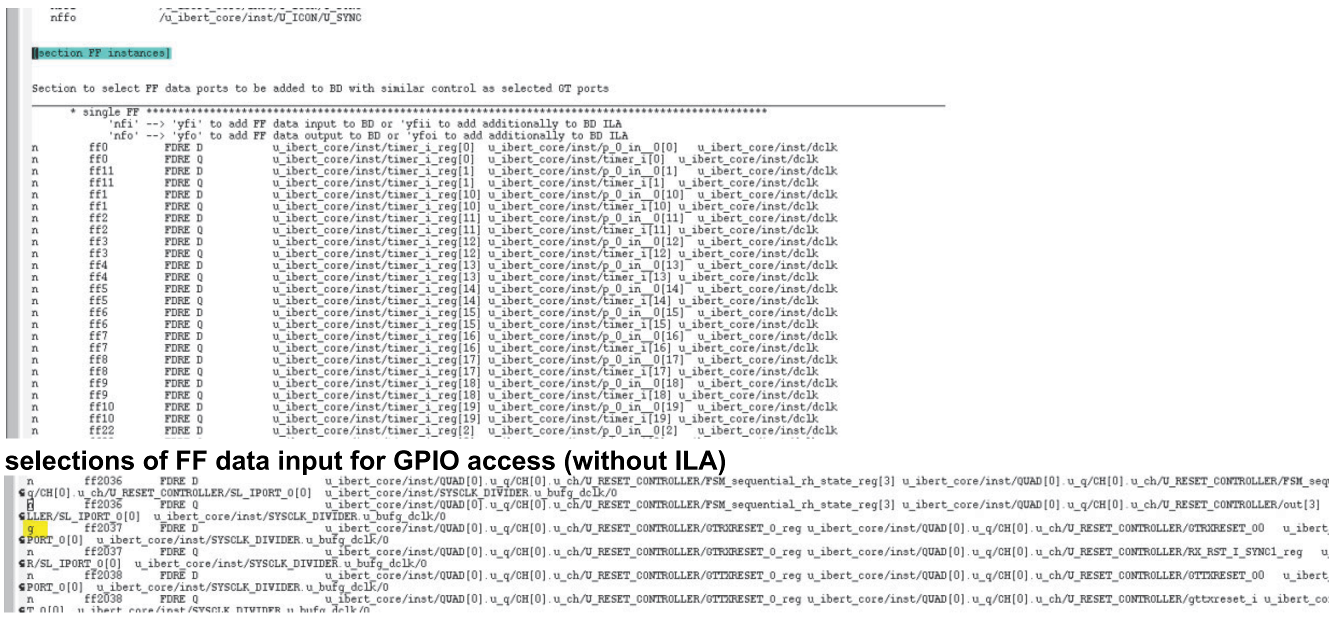
Figure 11: **ILA Hierarchies**

[section FF instances]

This section lists all flip-flop instances found in the application design. For every flip-flop, a separate line is given for input and output.

```
n      <FF index> <FF type> D <FF instance name> <FF input net> <FF clock net> <driving
clock port>
n     <FF index> <FF type> Q <FF instance name> <FF output net> <FF clock net> <driving
clock port>
```

The same selections as for GT ports can be done here (`g`, `gi`, `v`, `vi`).



```
nffo            /u_ibert_core/inst/U_ICON/U_SYNC

[section FF instances]

Section to select FF data ports to be added to BD with similar control as selected GT ports
__________________________________________________________________________________________
      * single FF ****************************************************************************
           'nfi' --> 'yfi' to add FF data input to BD or 'yfii to add additionally to BD ILA
           'nfo' --> 'yfo' to add FF data output to BD or 'yfoi to add additionally to BD ILA
n       ff0       FDRE D         u_ibert_core/inst/timer_i_reg[0]  u_ibert_core/inst/p_0_in__0[0]   u_ibert_core/inst/dclk
n       ff0       FDRE Q         u_ibert_core/inst/timer_i_reg[0]  u_ibert_core/inst/timer_i[0]  u_ibert_core/inst/dclk
n       ff11      FDRE D         u_ibert_core/inst/timer_i_reg[1]  u_ibert_core/inst/p_0_in__0[1]   u_ibert_core/inst/dclk
n       ff11      FDRE Q         u_ibert_core/inst/timer_i_reg[1]  u_ibert_core/inst/timer_i[1]  u_ibert_core/inst/dclk
n       ff1       FDRE D         u_ibert_core/inst/timer_i_reg[10] u_ibert_core/inst/p_0_in__0[10]  u_ibert_core/inst/dclk
n       ff1       FDRE Q         u_ibert_core/inst/timer_i_reg[10] u_ibert_core/inst/timer_i[10] u_ibert_core/inst/dclk
n       ff2       FDRE D         u_ibert_core/inst/timer_i_reg[11] u_ibert_core/inst/p_0_in__0[11]  u_ibert_core/inst/dclk
n       ff2       FDRE Q         u_ibert_core/inst/timer_i_reg[11] u_ibert_core/inst/timer_i[11] u_ibert_core/inst/dclk
n       ff3       FDRE D         u_ibert_core/inst/timer_i_reg[12] u_ibert_core/inst/p_0_in__0[12]  u_ibert_core/inst/dclk
n       ff3       FDRE Q         u_ibert_core/inst/timer_i_reg[12] u_ibert_core/inst/timer_i[12] u_ibert_core/inst/dclk
n       ff4       FDRE D         u_ibert_core/inst/timer_i_reg[13] u_ibert_core/inst/p_0_in__0[13]  u_ibert_core/inst/dclk
n       ff4       FDRE Q         u_ibert_core/inst/timer_i_reg[13] u_ibert_core/inst/timer_i[13] u_ibert_core/inst/dclk
n       ff5       FDRE D         u_ibert_core/inst/timer_i_reg[14] u_ibert_core/inst/p_0_in__0[14]  u_ibert_core/inst/dclk
n       ff5       FDRE Q         u_ibert_core/inst/timer_i_reg[14] u_ibert_core/inst/timer_i[14] u_ibert_core/inst/dclk
n       ff6       FDRE D         u_ibert_core/inst/timer_i_reg[15] u_ibert_core/inst/p_0_in__0[15]  u_ibert_core/inst/dclk
n       ff6       FDRE Q         u_ibert_core/inst/timer_i_reg[15] u_ibert_core/inst/timer_i[15] u_ibert_core/inst/dclk
n       ff7       FDRE D         u_ibert_core/inst/timer_i_reg[16] u_ibert_core/inst/p_0_in__0[16]  u_ibert_core/inst/dclk
n       ff7       FDRE Q         u_ibert_core/inst/timer_i_reg[16] u_ibert_core/inst/timer_i[16] u_ibert_core/inst/dclk
n       ff8       FDRE D         u_ibert_core/inst/timer_i_reg[17] u_ibert_core/inst/p_0_in__0[17]  u_ibert_core/inst/dclk
n       ff8       FDRE Q         u_ibert_core/inst/timer_i_reg[17] u_ibert_core/inst/timer_i[17] u_ibert_core/inst/dclk
n       ff9       FDRE D         u_ibert_core/inst/timer_i_reg[18] u_ibert_core/inst/p_0_in__0[18]  u_ibert_core/inst/dclk
n       ff9       FDRE Q         u_ibert_core/inst/timer_i_reg[18] u_ibert_core/inst/timer_i[18] u_ibert_core/inst/dclk
n       ff10      FDRE D         u_ibert_core/inst/timer_i_reg[19] u_ibert_core/inst/p_0_in__0[19]  u_ibert_core/inst/dclk
n       ff10      FDRE Q         u_ibert_core/inst/timer_i_reg[19] u_ibert_core/inst/timer_i[19] u_ibert_core/inst/dclk
n       ff22      FDRE D         u_ibert_core/inst/timer_i_reg[2]  u_ibert_core/inst/p_0_in__0[2]   u_ibert_core/inst/dclk
```

selections of FF data input for GPIO access (without ILA)

```
 n       ff2036     FDRE D             u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/FSM_sequential_rh_state_reg[3] u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/FSM_seq
q/CH[0].u_ch/U_RESET_CONTROLLER/SL_IPORT_0[0]  u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk/0
 n       ff2036     FDRE Q             u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/FSM_sequential_rh_state_reg[3] u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/out[3]
LLER/SL_IPORT_0[0]  u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk/0
 g       ff2037     FDRE D             u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/GTRXRESET_0_reg u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/GTRXRESET_00   u_ibert
PORT_0[0]  u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk/0
 n       ff2037     FDRE Q             u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/GTRXRESET_0_reg u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/RX_RST_I_SYNC1_reg   u
R/SL_IPORT_0[0]  u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk/0
 n       ff2038     FDRE D             u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/GTTXRESET_0_reg u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/GTTXRESET_00   u_ibert
PORT_0[0]  u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk/0
 n       ff2038     FDRE Q             u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/GTTXRESET_0_reg u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_RESET_CONTROLLER/gttxreset_i u_ibert_co
T_0[0]  u_ibert_core/inst/SYSCLK_DIVIDER.u_bufg_dclk/0
```

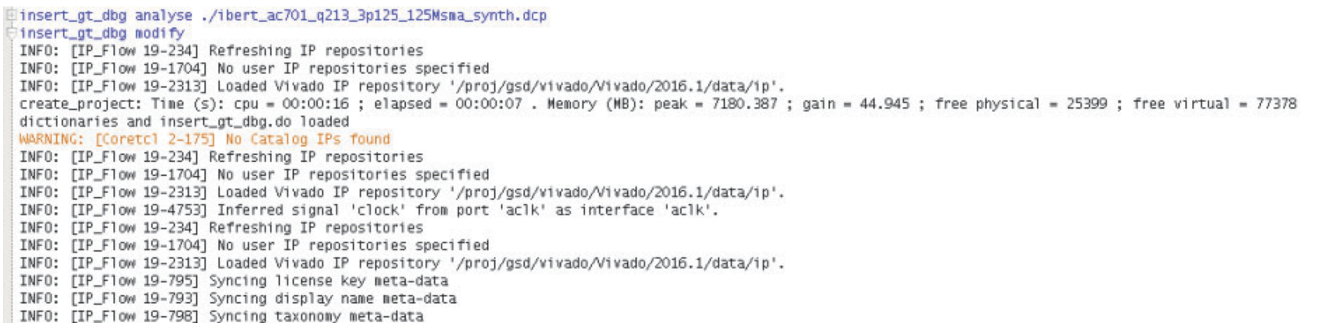
X18506-120816

Figure 12: **FF Instances**

Design Modification

There are two possibilities for the modify command. If still using the same Vivado tools session in which the analysis was run, execute the following command in the Tcl console (Figure 13):

```
insert_gt_dbg modify
```



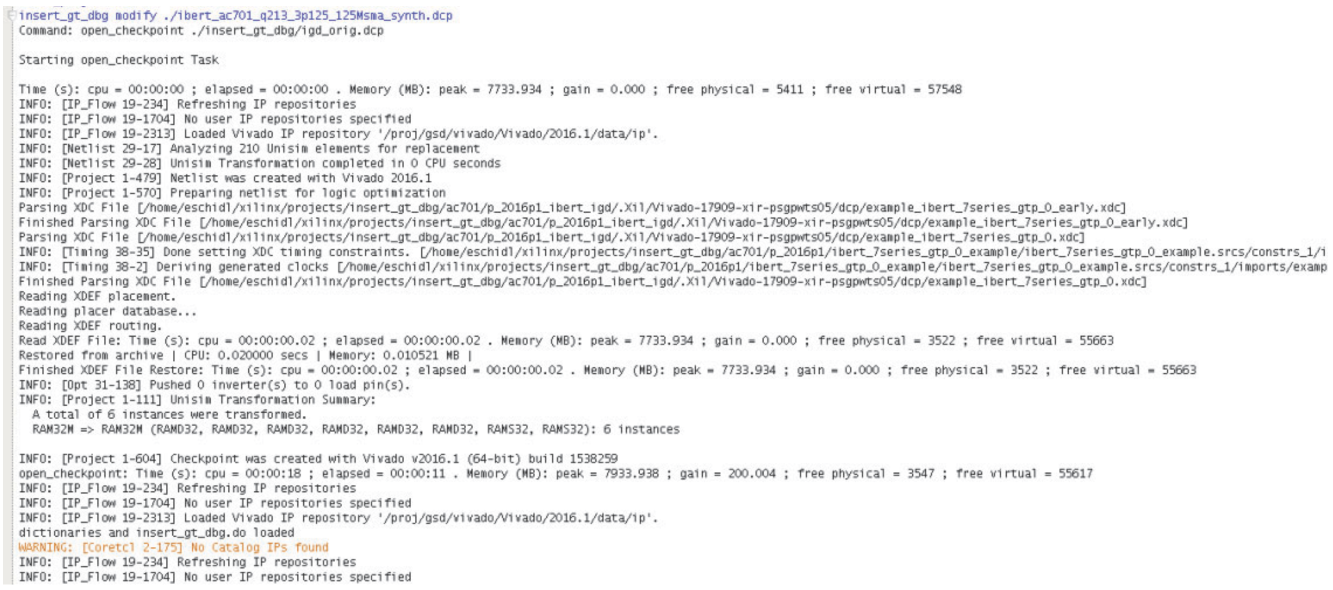
```
insert_gt_dbg analyse ./ibert_ac701_q213_3p125_125Msma_synth.dcp
insert_gt_dbg modify
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/proj/gsd/vivado/Vivado/2016.1/data/ip'.
create_project: Time (s): cpu = 00:00:16 ; elapsed = 00:00:07 . Memory (MB): peak = 7180.387 ; gain = 44.945 ; free physical = 25399 ; free virtual = 77378
dictionaries and insert_gt_dbg.do loaded
WARNING: [Coretcl 2-175] No Catalog IPs found
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/proj/gsd/vivado/Vivado/2016.1/data/ip'.
INFO: [IP_Flow 19-4753] Inferred signal 'clock' from port 'aclk' as interface 'aclk'.
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/proj/gsd/vivado/Vivado/2016.1/data/ip'.
INFO: [IP_Flow 19-795] Syncing license key meta-data
INFO: [IP_Flow 19-793] Syncing display name meta-data
INFO: [IP_Flow 19-798] Syncing taxonomy meta-data
```

X18507-120816

Figure 13: **Log of Modify Step Start Directly After Analysis in the Same Tcl Console**

If a new Vivado tools session was started as mentioned in Preparation, page 6, and the analysis was already run before in `<dbg_dir>`, execute the following command (Figure 14):

```
insert_gt_dbg modify <DCP file>
```



```
insert_gt_dbg modify ./ibert_ac701_q213_3p125_125Msma_synth.dcp
Command: open_checkpoint ./insert_gt_dbg/igd_orig.dcp

Starting open_checkpoint Task

Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 7733.934 ; gain = 0.000 ; free physical = 5411 ; free virtual = 57548
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/proj/gsd/vivado/Vivado/2016.1/data/ip'.
INFO: [Netlist 29-17] Analyzing 210 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-479] Netlist was created with Vivado 2016.1
INFO: [Project 1-570] Preparing netlist for logic optimization
Parsing XDC File [/home/eschidl/xilinx/projects/insert_gt_dbg/ac701/p_2016p1_ibert_igd/.Xil/Vivado-17909-xir-psgpwts05/dcp/example_ibert_7series_gtp_0_early.xdc]
Finished Parsing XDC File [/home/eschidl/xilinx/projects/insert_gt_dbg/ac701/p_2016p1_ibert_igd/.Xil/Vivado-17909-xir-psgpwts05/dcp/example_ibert_7series_gtp_0_early.xdc]
Parsing XDC File [/home/eschidl/xilinx/projects/insert_gt_dbg/ac701/p_2016p1_ibert_igd/.Xil/Vivado-17909-xir-psgpwts05/dcp/example_ibert_7series_gtp_0.xdc]
INFO: [Timing 38-35] Done setting XDC timing constraints. [/home/eschidl/xilinx/projects/insert_gt_dbg/ac701/p_2016p1/ibert_7series_gtp_0_example/ibert_7series_gtp_0_example.srcs/constrs_1/i
INFO: [Timing 38-2] Deriving generated clocks [/home/eschidl/xilinx/projects/insert_gt_dbg/ac701/p_2016p1/ibert_7series_gtp_0_example/ibert_7series_gtp_0_example.srcs/constrs_1/imports/examp
Finished Parsing XDC File [/home/eschidl/xilinx/projects/insert_gt_dbg/ac701/p_2016p1_ibert_igd/.Xil/Vivado-17909-xir-psgpwts05/dcp/example_ibert_7series_gtp_0.xdc]
Reading XDEF placement.
Reading placer database...
Reading XDEF routing.
Read XDEF File: Time (s): cpu = 00:00:00.02 ; elapsed = 00:00:00.02 . Memory (MB): peak = 7733.934 ; gain = 0.000 ; free physical = 3522 ; free virtual = 55663
Restored from archive | CPU: 0.020000 secs | Memory: 0.010521 MB |
Finished XDEF File Restore: Time (s): cpu = 00:00:00.02 ; elapsed = 00:00:00.02 . Memory (MB): peak = 7733.934 ; gain = 0.000 ; free physical = 3522 ; free virtual = 55663
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
  A total of 6 instances were transformed.
  RAM32M => RAM32M (RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMS32, RAMS32): 6 instances

INFO: [Project 1-604] Checkpoint was created with Vivado v2016.1 (64-bit) build 1538259
open_checkpoint: Time (s): cpu = 00:00:18 ; elapsed = 00:00:11 . Memory (MB): peak = 7933.938 ; gain = 200.004 ; free physical = 3547 ; free virtual = 55617
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/proj/gsd/vivado/Vivado/2016.1/data/ip'.
dictionaries and insert_gt_dbg.do loaded
WARNING: [Coretcl 2-175] No Catalog IPs found
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
```

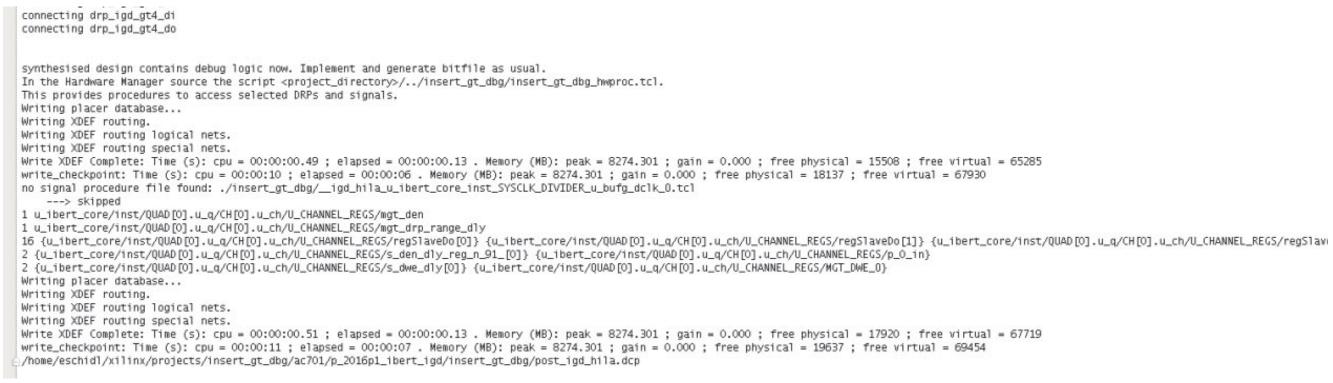
X18508-120816

Figure 14: **Log of Modify Step Start when Analysis Was Not the Previous Step in Tcl Console**

First some IP cores can be generated for the provided Verilog modules necessary to implement functionality. After this, a block design is built containing the debug logic. This block design is then inserted into the application DCP and connected. The resulting design is saved in a DCP file in the `<dbgdir>/insert_gt_dbg` directory.

Several windows might open and close in the process. Leave them untouched until the following message appears in the log (Figure 15):

```
Synthesized design contains debug logic now. Implement and generate bitfile as usual.
In the Hardware Manager source the script
<project_directory>/../insert_gt_dbg/insert_gt_dbg_hwproc.tcl.
This provides procedures to access selected DRPs and signals.
```



```
connecting drp_igd_gt4_di
connecting drp_igd_gt4_do


synthesised design contains debug logic now. Implement and generate bitfile as usual.
In the Hardware Manager source the script <project_directory>/../insert_gt_dbg/insert_gt_dbg_hwproc.tcl.
This provides procedures to access selected DRPs and signals.
Writing placer database...
Writing XDEF routing.
Writing XDEF routing logical nets.
Writing XDEF routing special nets.
Write XDEF Complete: Time (s): cpu = 00:00:00.49 ; elapsed = 00:00:00.13 . Memory (MB): peak = 8274.301 ; gain = 0.000 ; free physical = 15508 ; free virtual = 65285
write_checkpoint: Time (s): cpu = 00:00:10 ; elapsed = 00:00:06 . Memory (MB): peak = 8274.301 ; gain = 0.000 ; free physical = 18137 ; free virtual = 67930
no signal procedure file found: ./insert_gt_dbg/__igd_hila_u_ibert_core_inst_SYSCLK_DIVIDER_u_bufg_dclk_0.tcl
    ---> skipped
1 u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/mgt_den
1 u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/mgt_drp_range_dly
16 {u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/regSlaveDo[0]} {u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/regSlaveDo[1]} {u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/regSlav
2 {u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/s_den_dly_reg_n_91_[0]} {u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/p_0_in}
2 {u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/s_dwe_dly[0]} {u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/U_CHANNEL_REGS/MGT_DWE_0}
Writing placer database...
Writing XDEF routing.
Writing XDEF routing logical nets.
Writing XDEF routing special nets.
Write XDEF Complete: Time (s): cpu = 00:00:00.51 ; elapsed = 00:00:00.13 . Memory (MB): peak = 8274.301 ; gain = 0.000 ; free physical = 17920 ; free virtual = 67719
write_checkpoint: Time (s): cpu = 00:00:11 ; elapsed = 00:00:07 . Memory (MB): peak = 8274.301 ; gain = 0.000 ; free physical = 19637 ; free virtual = 69454
/home/eschidl/xilinx/projects/insert_gt_dbg/ac701/p_2016p1_ibert_igd/insert_gt_dbg/post_igd_hila.dcp
```

X18509-120816

Figure 15: **Log of Modify Step End**

Design Implementation

To implement the modified DCP, the command provides the option to start a default implementation with the following:

```
insert_gt_dbg impl_def
```

This includes the steps opt_design, place_design, route_design, report_timing_summary -verbose, write_bitstream, and write_debug_probes without any specific further options. After each step a checkpoint is saved. Output files can be found in the directory `<dbgdir>/insert_gt_dbg` (`igd_impl.timrptsum`, `igd_dbg.bit`, `igd_dbg.ltx`).

If more options are necessary for the implementation of the design, these steps would have to be done manually.

Usage in Hardware Manager

Preparation

Open the Vivado tools, start the hardware manager, and program the device with the generated bitstream and probe file. If only VIO was selected to control the signals, you don't need anything else. For DRP and GPIO control, make sure you are in the `<dbgdir>` directory in the Tcl console.

Source the generated file `insert_gt_dbg_hwproc.tcl` as follows:

```
source ./insert_gt_dbg/insert_gt_dbg_hwproc.tcl
```

All provided Tcl procedures have the prefix `igd_`. All global variables used have the prefix `_igd_`.

List of Tcl Procedures

- igd_axi_res
- igd_axi_rd <base address> <address>
- igd_axi_wr <base address> <address> <data>
- igd_axi_wrm <base address> <address> <data> <mask>
- igd_axi_rmw <base address> <address> <data> <mask>
- igd_data_modify <original data> <new data> <mask> <width>
- igd_data_slice <data> <width> <from> <to>
- igd_data_concat {<data> <width>} [{<data> <width>} ...]
- igd_gpio_ctr_wr <GPIO channel> <data> [<mask>]
- igd_gpio_ctr_drpmux_reset_rst
- igd_gpio_ctr_drpmux_reset_set yes
- igd_gpio_ctr_oupdate_rst
- igd_gpio_ctr_oupdate_set

- igd_gpio_ctr_iupdate_rst
- igd_gpio_ctr_iupdate_set
- igd_drp_<GT identifier>_rd <DRP address>
- igd_drp_<GT identifier>_wr <DRP address> <data>
- igd_drp_<GT identifier>_rmw <DRP address> <data> <mask>
- igd_gpio_ctr_<GT identifier>_drpappdis_rst
- igd_gpio_ctr_<GT identifier>_drpappdis_set
- igd_drp_rd <GT identifier> <attribute name>
- igd_drp_wr <GT identifier> <attribute name> <value>
- igd_gpio_<GT/FF identifier>_<signal name>_rd_gt_d
- igd_gpio_<GT/FF identifier>_<signal name>_rd_reg_d
- igd_gpio_<GT/FF identifier>_<signal name>_wr_d <data> <mask>
- igd_gpio_<GT/FF identifier>_<signal name>_rd_m
- igd_gpio_<GT/FF identifier>_<signal name>_wr_m <data> <mask>
- igd_gpio_<GT/FF identifier>_<signal name>_rst_m
- igd_gpio_<GT/FF identifier>_<signal name>_set_m

General Tcl Procedures

igd_axi_res

This procedure allows you to reset `hw_axi`, should that become necessary. It does not have any further options.

igd_axi_rd <base address> <address>

This procedure reads an AXI register and returns the read data.

igd_axi_wr <base address> <address> <data>

This procedure writes an AXI register and returns the written data.

igd_axi_wrm <base address> <address> <data> <mask>

This procedure writes bits of an AXI register depending on the mask and returns the written data. It is used for GPIO access. The current output value of a GPIO register cannot be read back from hardware. The output value is saved in the Tcl dictionary `_igd_gpio_wrv`, which is used to generate the masked value.

igd_axi_rmw <base address> <address> <data> <mask>

This is the read-modify-write operation of an AXI register depending on the mask. This procedure returns the written data and is used for DRP access.

igd_data_modify <original data> <new data> <mask> <width>

This procedure modifies bits of `<original data>` to bits of `<new data>` depending on `<mask>`. The `<width>` is used to get the correct length for the return value as a hexadecimal value.

igd_data_slice <data> <width> <from> <to>

This procedure cuts a bit range out of `<data>`. The `<width>` is the bit width of the range, `<from>` is the LSB of the range, and `<to>` is the MSB. This procedure returns a list with the format {`<slice value> <width>`}.

igd_data_concat {<data> <width>} [{<data> <width>} ...]

This procedure combines several data values to a single value. The arguments to the procedure are lists containing the data value and the bit width of the current part of the data to connect. The first argument is placed at the MSB of the resulting value and the last at the LSB. This procedure returns a list for the combined data in the format {`<value> <width>`}.

igd_gpio_ctr_wr <GPIO channel> <data> [<mask>]

This procedure writes the selected channel of the GPIO control register for the debug logic. It returns the value of that channel after the write.

> `<mask>`: If omitted, it defaults to `0xffffffff`.
>
> GPIO channel 1 bit[0]: DRPMUX reset for all implemented arbiters. This bit should not be necessary to set. Use it only when you are sure that a DRPMUX does not recover normally. This can corrupt the DRP protocol on the application side. Use with `igd_gpio_ctr_drpmux_reset_rst` and `igd_gpio_ctr_drpmux_reset_set`.
>
> GPIO channel 1 bit[1]: `Oupdate`. When 0, writes to GPIO registers have no effect on debug logic. When 1, previous and further writes have immediate effect. Use with `igd_gpio_ctr_oupdate_rst` and `igd_gpio_ctr_oupdate_set`.
>
> GPIO channel 1 bit[2]: `Iupdate`. Used as an enable for the read flip-flop. Consistent bus values can be read when set to 0. Use with `igd_gpio_ctr_iupdate_rst` and `igd_gpio_ctr_iupdate_set`. These are used in the procedures for signal access (see igd_gpio_ctr_iupdate_rst and igd_gpio_ctr_iupdate_set). It should not be necessary to use them explicitly.

Further bit usage depends on DRP arbiter usage. Every DRP arbiter gets its own `drpappdis` bit which, when set, disables the DRP access from the application side. Use with the specific reset and set procedure for this bit generated for the specific DRP port.

Because there are 64 bits available in the GPIO control register over the two channel register, there are a maximum of 61 DRP arbiters that the script can currently implement. This should suit most purposes.

igd_gpio_ctr_drpmux_reset_rst

This procedure resets the reset for the DRP arbiters and returns nothing. For more information, see igd_gpio_ctr_wr <GPIO channel> <data> [<mask>].

igd_gpio_ctr_drpmux_reset_set yes

This procedure sets the reset for the DRP arbiters and returns nothing. For more information, see igd_gpio_ctr_wr <GPIO channel> <data> [<mask>]. The `yes` is introduced to prevent an accidental reset when unintentionally calling the procedure.

igd_gpio_ctr_oupdate_rst

This procedure resets `oupdate` and returns nothing. For more information, see igd_gpio_ctr_wr <GPIO channel> <data> [<mask>].

igd_gpio_ctr_oupdate_set

This procedure sets `oupdate` and returns nothing. For more information, see igd_gpio_ctr_wr <GPIO channel> <data> [<mask>].

igd_gpio_ctr_iupdate_rst

This procedure resets `iupdate` and returns nothing. For more information, see igd_gpio_ctr_wr <GPIO channel> <data> [<mask>].

igd_gpio_ctr_iupdate_set

This procedure sets `iupdate` and returns nothing. For more information, see igd_gpio_ctr_wr <GPIO channel> <data> [<mask>].

Tcl Procedures for DRP Access

There are three procedures provided for every DRP port instrumented for read, write, and read-modify-write:

igd_drp_<GT identifier>_rd <DRP address>

igd_drp_<GT identifier>_wr <DRP address> <data>

igd_drp_<GT identifier>_rmw <DRP address> <data> <mask>

`<GT identifier>`: If the location of the transceiver instance is known during analysis it is used as an identifier here, e.g., for CHANNEL X0Y0 it is `cX0Y0` (`igd_drp_cX0Y0_rd`) and for COMMON X0Y1 it is `qX0Y1` (`igd_drp_qX0Y1_rd`). If the location is not known, the index used in the `.do` file is used (`igd_drp_gt2_rd`).

The return value for the read is the value read from hardware. The return value for writes is the actual written data.

When a DRP arbiter is used for the instance, two additional procedures are provided to enable (rst) or disable (set) application access:

igd_gpio_ctr_<GT identifier>_drpappdis_rst

This procedure enables application access to the DRP port and returns nothing.

igd_gpio_ctr_<GT identifier>_drpappdis_set

This procedure disables application access to the DRP port and returns nothing.

Additionally there are two general procedures to read/write transceiver attributes:

igd_drp_rd <GT identifier> <attribute name>

This procedure reads the attribute value from the transceiver `<GT identifier>`. It performs the necessary `igd_drp_<GT identifier>_rd` calls to read the full attribute. The address mapping is provided through Tcl files in the `<dbgdir>/insert_gt_dbg` directory which are sourced through `insert_gt_dbg_hwproc.tcl`. This procedure returns the attribute value as a hexadecimal value. For example:

```
igd_drp_rd cX0Y0 ES_SDATA_MASK
```

igd_drp_wr <GT identifier> <attribute name> <value>

This procedure writes the attribute value of the transceiver `<GT identifier>`. It performs the necessary `igd_drp_<GT identifier>_rmw` calls to write the full attribute. The address mapping is provided through Tcl files in the `<dbgdir>/insert_gt_dbg` directory which are sourced through `insert_gt_dbg_hwproc.tcl`. This procedure returns the written attribute value as a hexadecimal value. For example:

```
igd_drp_wr cX0Y0 ES_SDATA_MASK 0xffffffffff0000000000
```

Tcl Procedures for Signal Access

This section provides a set of procedures for each instrumented signal, beginning with a list of procedures that are meant to be used by the user. Other procedures are supportive of these initial procedures.

> `<signal name>`: For a single-bit transceiver port, this is the port name (e.g., GTRXRESET). For a transceiver port bus or part of the bus, it is the port name plus the LSB of the selected part (e.g., for TXDATA[31:24] it is TXDATA_24). For a flip-flop port it is the port name plus the register instance name (e.g., D_GTRXRESET_0_reg).

igd_gpio_<GT/FF identifier>_<signal name>_rd_gt_d

This procedure reads and returns the signal value from hardware. It handles `iupdate` and the tristate port of the GPIO IP automatically.

igd_gpio_<GT/FF identifier>_<signal name>_rd_reg_d

This procedure reads and returns the written GPIO register value for the signal from the Tcl dictionary `_igd_gpio_wrv`.

igd_gpio_<GT/FF identifier>_<signal name>_wr_d <data> <mask>

This procedure writes the GPIO register value for the signal in hardware and updates the dictionary `_igd_gpio_wrv`. It resets `oupdate` before writing and recovers its value in the end. The procedure returns nothing.

igd_gpio_<GT/FF identifier>_<signal name>_rd_m

This procedure reads and returns the written GPIO register value for the signal MUX control from the Tcl dictionary `_igd_gpio_wrv`.

igd_gpio_<GT/FF identifier>_<signal name>_wr_m <data> <mask>

This procedure writes the GPIO register value for the signal MUX control and updates the dictionary `_igd_gpio_wrv`. It resets `oupdate` before writing and recovers its value in the end. The procedure returns nothing.

igd_gpio_<GT/FF identifier>_<signal name>_rst_m

This procedure resets all bits of the GPIO register value for the signal MUX control. It returns nothing.

igd_gpio_<GT/FF identifier>_<signal name>_set_m

This procedure sets all bits of the GPIO register value for the signal MUX control. It returns nothing.

The following is a list of procedures that are used within the above procedures:

- `igd_gpio_<GT/FF identifier>_<signal name>_rd_d_<GPIO block number>_<GPIO channel number>`
- `igd_gpio_<GT/FF identifier>_<signal name>_wr_d_<GPIO block number>_<GPIO channel number> <[list data width]> <[list mask width]>`
- `igd_gpio_<GT/FF identifier>_<signal name>_wr_t_<GPIO block number>_<GPIO channel number> <[list data width]> <[list mask width]>`
- `igd_gpio_<GT/FF identifier>_<signal name>_wr_m_<GPIO block number>_<GPIO channel number> <[list data width]> <[list mask width]>`

The numbering for the GPIO block and channel depends on how the signal is mapped to them. This can change for every build. There can be several procedures for buses, depending on the mapping. It should not be necessary to call these procedures manually.

For example, to set the near-end PMA loopback for CHANNEL X0Y1, use these procedures:

- `igd_gpio_cX0Y1_LOOPBACK_0_wr_d 0x2 0x7`
- `igd_gpio_cX0Y1_LOOPBACK_0_set_m`
- `igd_gpio_ctr_oupdate_set`

Additional Scripts for More Complex Use Cases

Eye Scan

Because doing an eye scan in an existing application is one of the most desirable debug methods for transceiver channels, a separate script is provided to do an eye scan based on the debug logic and procedures discussed above. There is no additional logic in hardware used to speed up a full eye scan. Thus, two methods are used to improve scan time: vector scan and matrix scan.

The vector scan is done along vectors starting from the center of the eye. The number of vectors can be chosen as a power of 2. For example, 2 would show the horizontal eye opening, and 4 would show the horizontal and vertical eye openings. Increasing numbers would divide the eye more and show more of its details, but the opening can already be recognized with a low number of vectors.

The full length of each vector is not looked at, but the edge of the eye is searched for with a dividing algorithm. The starting point is the center, then the midpoint of the vector is looked at. If there are no errors, the midpoint of the outer half is looked at. Otherwise, the midpoint of the inner half is tested. This goes on until two adjacent points are found with one passing and one failing. If a point is already measured from another vector due to rounding of the integer coordinates, the previous result is used. Both methods together reduce the number of points that need to be tested for a reasonable eye diagram.

Two approaches can be used for the vector scan. First, a range of prescale settings can be set and for each a separate scan is done. Valid prescale settings are –1 to 31. The values 0 to 31 are the prescale settings described in the relevant transceiver user guide [Ref 1] [Ref 2] and are used with the COUNT branch of the eye scan finite state machine (FSM). This provides BER results. A setting of –1 selects the ARMED branch of the FSM and provides the quickest possible scan with just pass/fail results.

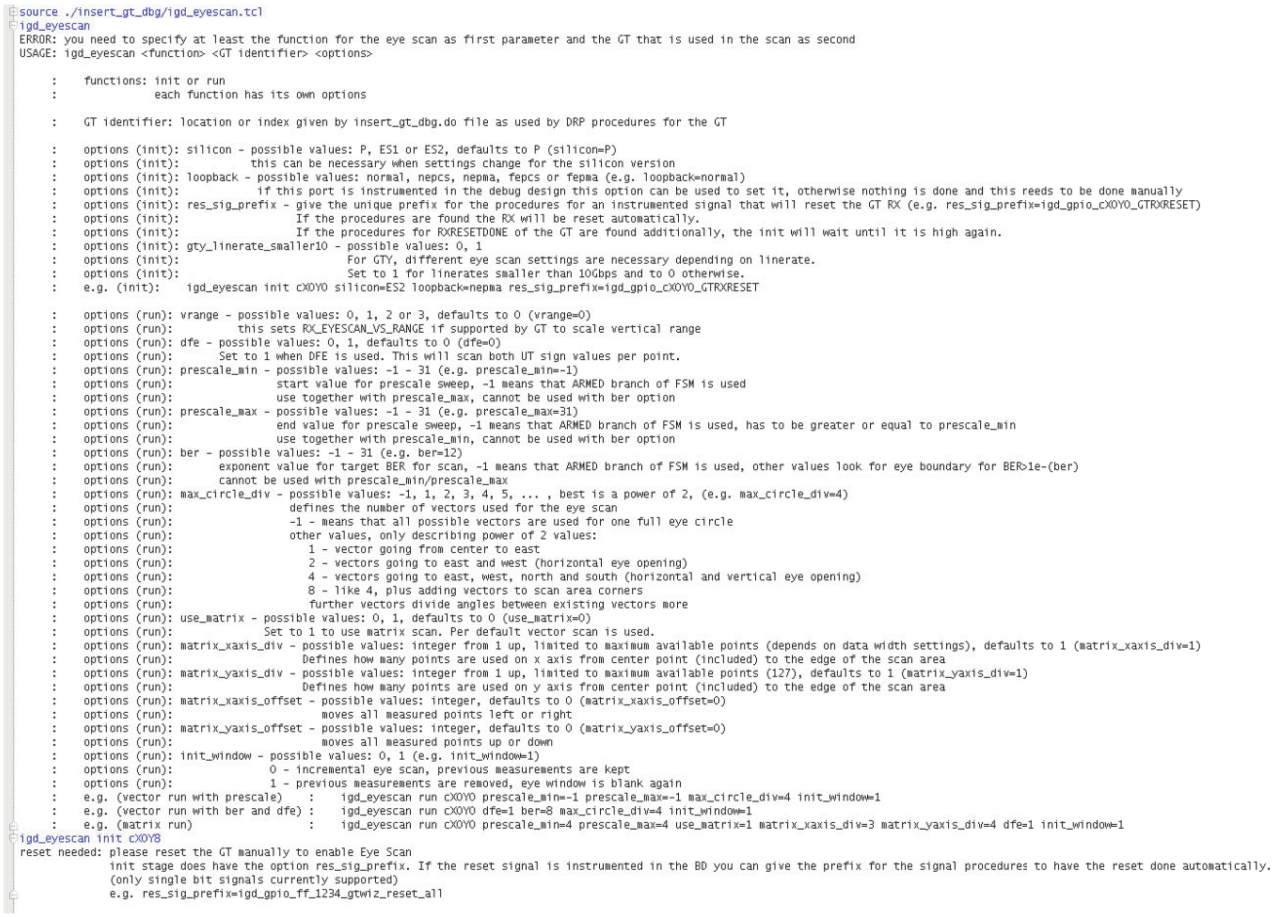
Second, the BER exponent can be set for the error rate to test for. Here, the vector is first scanned through the ARMED branch to find the eye border. Then the relevant prescale value is set for the wanted BER, and from the previously found border the actual border for the BER is searched for in small steps along the vector.

The matrix scan allows you to scan the full eye at equidistant points. The number of points used can be defined separately for the horizontal and vertical directions, starting from the center point out to the edge. Additionally, an offset can also be given in the horizontal and vertical directions. For example, when setting the number of points to 1 in one direction, a horizontal or vertical line can be scanned, and with the offset this line can be moved.

Source the provided Tcl script `igd_eyescan.tcl` to use the eye scan.

```
source ./insert_gt_dbg/igd_eyescan.tcl
```

The main procedure to use the scan is igd_eyescan. When called by itself, this procedure gives the help message shown in Figure 16.



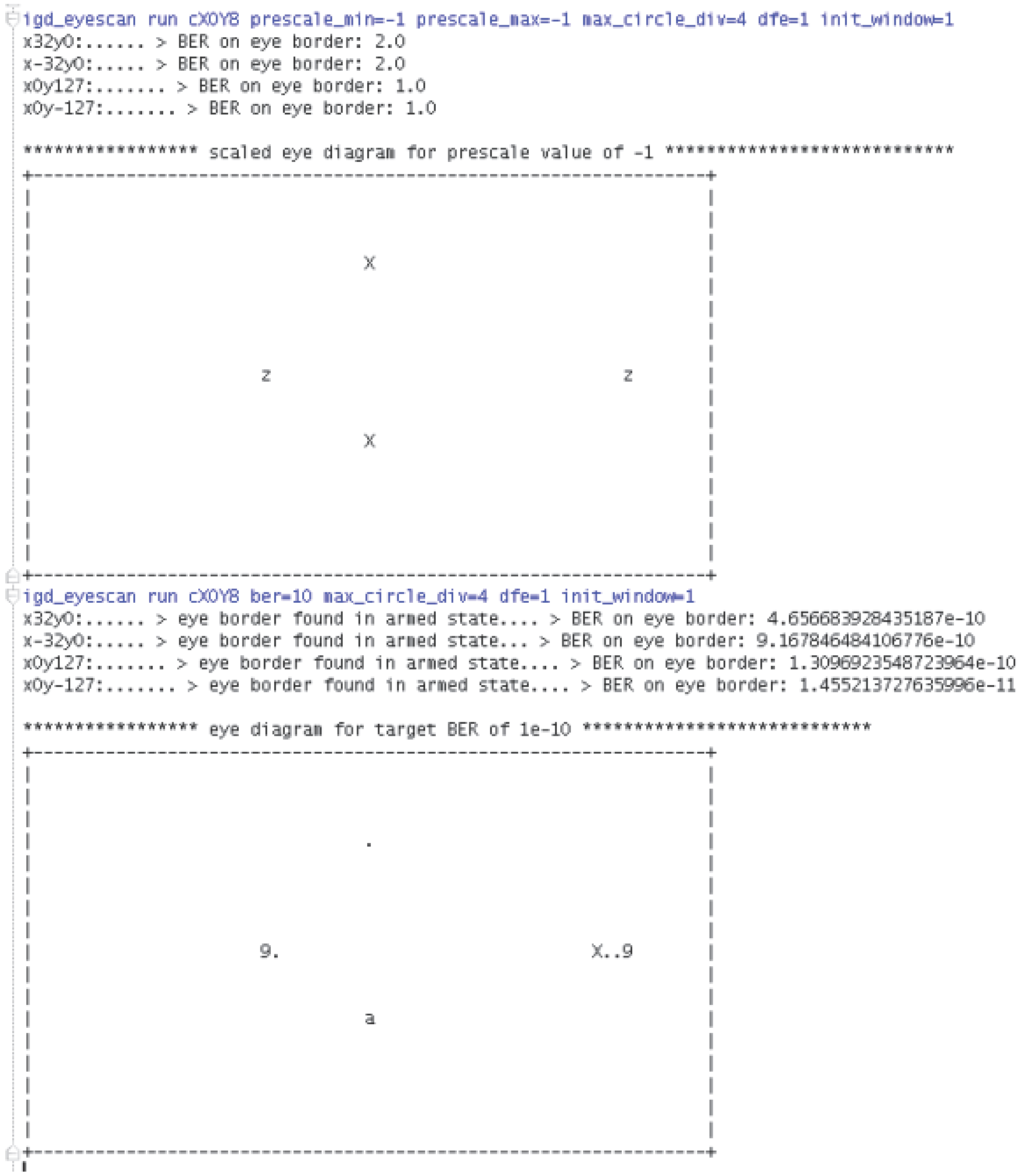
```
source ./insert_gt_dbg/igd_eyescan.tcl
igd_eyescan
ERROR: you need to specify at least the function for the eye scan as first parameter and the GT that is used in the scan as second
USAGE: igd_eyescan <function> <GT identifier> <options>

     :    functions: init or run
     :               each function has its own options

     :    GT identifier: location or index given by insert_gt_dbg.do file as used by DRP procedures for the GT

     :    options (init): silicon - possible values: P, ES1 or ES2, defaults to P (silicon=P)
     :    options (init):           this can be necessary when settings change for the silicon version
     :    options (init): loopback - possible values: normal, nepcs, nepma, fepcs or fepma (e.g. loopback=normal)
     :    options (init):            if this port is instrumented in the debug design this option can be used to set it, otherwise nothing is done and this reeds to be done manually
     :    options (init): res_sig_prefix - give the unique prefix for the procedures for an instrumented signal that will reset the GT RX (e.g. res_sig_prefix=igd_gpio_cX0Y0_GTRXRESET)
     :    options (init):                  If the procedures are found the RX will be reset automatically.
     :    options (init):                  If the procedures for RXRESETDONE of the GT are found additionally, the init will wait until it is high again.
     :    options (init): gty_linerate_smaller10 - possible values: 0, 1
     :    options (init):                          For GTY, different eye scan settings are necessary depending on linerate.
     :    options (init):                          Set to 1 for linerates smaller than 10Gbps and to 0 otherwise.
     :    e.g. (init):    igd_eyescan init cX0Y0 silicon=ES2 loopback=nepma res_sig_prefix=igd_gpio_cX0Y0_GTRXRESET

     :    options (run): vrange - possible values: 0, 1, 2 or 3, defaults to 0 (vrange=0)
     :    options (run):          this sets RX_EYESCAN_VS_RANGE if supported by GT to scale vertical range
     :    options (run): dfe - possible values: 0, 1, defaults to 0 (dfe=0)
     :    options (run):       Set to 1 when DFE is used. This will scan both UT sign values per point.
     :    options (run): prescale_min - possible values: -1 - 31 (e.g. prescale_min=-1)
     :    options (run):                start value for prescale sweep, -1 means that ARMED branch of FSM is used
     :    options (run):                use together with prescale_max, cannot be used with ber option
     :    options (run): prescale_max - possible values: -1 - 31 (e.g. prescale_max=31)
     :    options (run):                end value for prescale sweep, -1 means that ARMED branch of FSM is used, has to be greater or equal to prescale_min
     :    options (run):                use together with prescale_min, cannot be used with ber option
     :    options (run): ber - possible values: -1 - 31 (e.g. ber=12)
     :    options (run):       exponent value for target BER for scan, -1 means that ARMED branch of FSM is used, other values look for eye boundary for BER>1e-(ber)
     :    options (run):       cannot be used with prescale_min/prescale_max
     :    options (run): max_circle_div - possible values: -1, 1, 2, 3, 4, 5, ... , best is a power of 2, (e.g. max_circle_div=4)
     :    options (run):                  defines the number of vectors used for the eye scan
     :    options (run):                  -1 - means that all possible vectors are used for one full eye circle
     :    options (run):                  other values, only describing power of 2 values:
     :    options (run):                     1 - vector going from center to east
     :    options (run):                     2 - vectors going to east and west (horizontal eye opening)
     :    options (run):                     4 - vectors going to east, west, north and south (horizontal and vertical eye opening)
     :    options (run):                     8 - like 4, plus adding vectors to scan area corners
     :    options (run):                     further vectors divide angles between existing vectors more
     :    options (run): use_matrix - possible values: 0, 1, defaults to 0 (use_matrix=0)
     :    options (run):              Set to 1 to use matrix scan. Per default vector scan is used.
     :    options (run): matrix_xaxis_div - possible values: integer from 1 up, limited to maximum available points (depends on data width settings), defaults to 1 (matrix_xaxis_div=1)
     :    options (run):                    Defines how many points are used on x axis from center point (included) to the edge of the scan area
     :    options (run): matrix_yaxis_div - possible values: integer from 1 up, limited to maximum available points (127), defaults to 1 (matrix_yaxis_div=1)
     :    options (run):                    Defines how many points are used on y axis from center point (included) to the edge of the scan area
     :    options (run): matrix_xaxis_offset - possible values: integer, defaults to 0 (matrix_xaxis_offset=0)
     :    options (run):                       moves all measured points left or right
     :    options (run): matrix_yaxis_offset - possible values: integer, defaults to 0 (matrix_yaxis_offset=0)
     :    options (run):                       moves all measured points up or down
     :    options (run): init_window - possible values: 0, 1 (e.g. init_window=1)
     :    options (run):               0 - incremental eye scan, previous measurements are kept
     :    options (run):               1 - previous measurements are removed, eye window is blank again
     :    e.g. (vector run with prescale)    :    igd_eyescan run cX0Y0 prescale_min=-1 prescale_max=-1 max_circle_div=4 init_window=1
     :    e.g. (vector run with ber and dfe) :    igd_eyescan run cX0Y0 dfe=1 ber=8 max_circle_div=4 init_window=1
     :    e.g. (matrix run)                  :    igd_eyescan run cX0Y0 prescale_min=4 prescale_max=4 use_matrix=1 matrix_xaxis_div=3 matrix_yaxis_div=4 dfe=1 init_window=1
igd_eyescan init cX0Y8
reset needed: please reset the GT manually to enable Eye Scan
             init stage does have the option res_sig_prefix. If the reset signal is instrumented in the BD you can give the prefix for the signal procedures to have the reset done automatically.
             (only single bit signals currently supported)
             e.g. res_sig_prefix=igd_gpio_ff_1234_gtwiz_reset_all
```

X18510-041217

Figure 16: **Help Message for `igd_eyescan`**

This gives an overview of all options for the procedure. The `init` function should be called in the beginning followed by an RX reset. Then the `run` function can be called several times.

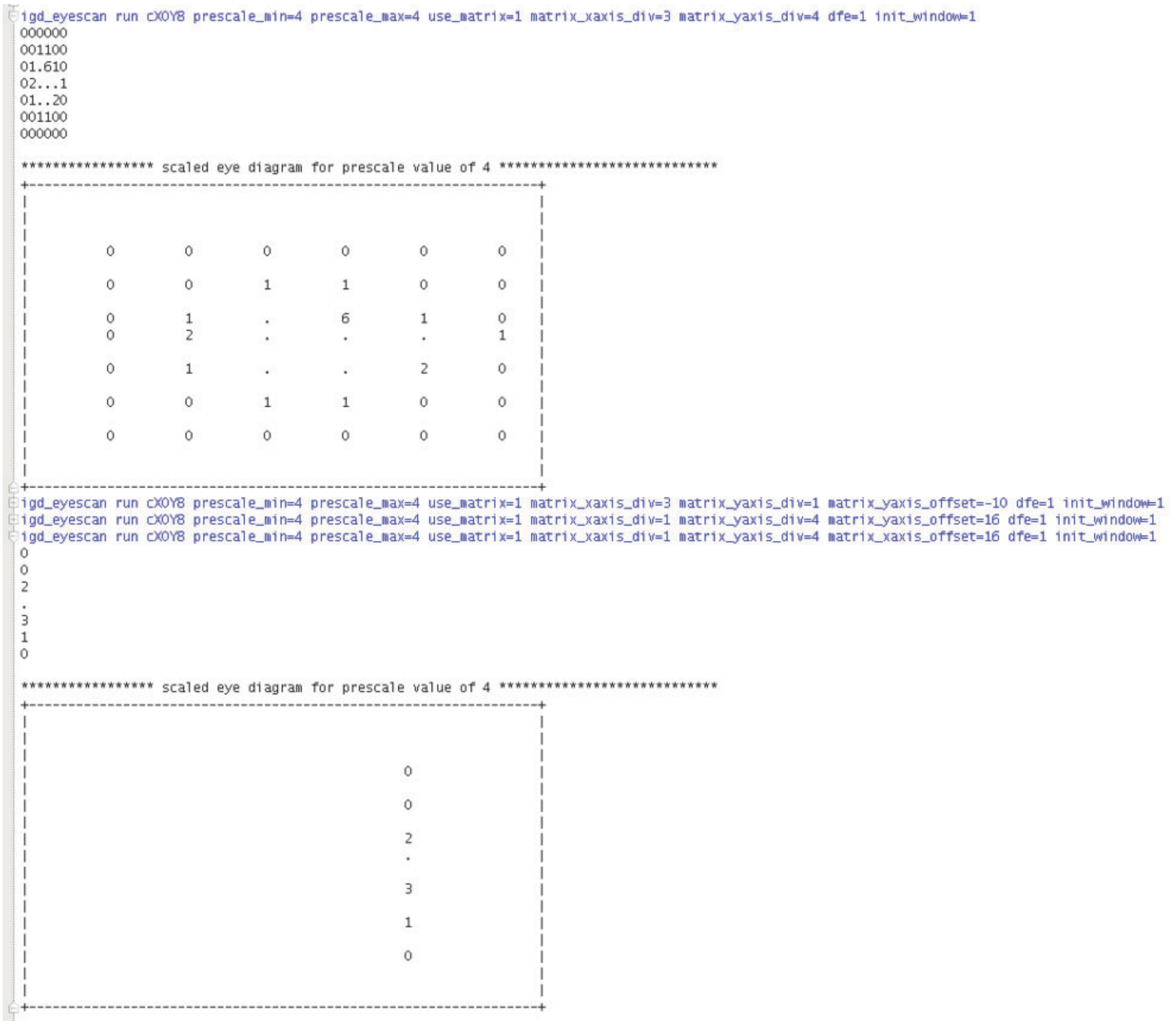
After each run, a downscaled eye plot is printed in the log as shown in Figure 17 for the vector scan and in Figure 18 for the matrix scan. The scan data is stored in two dictionaries. The `ber_d` dictionary contains the full scan data, and the `ber_plot_d` dictionary contains the scaled version.



```
igd_eyescan run cX0Y8 prescale_min=-1 prescale_max=-1 max_circle_div=4 dfe=1 init_window=1
x32y0:...... > BER on eye border: 2.0
x-32y0:..... > BER on eye border: 2.0
x0y127:....... > BER on eye border: 1.0
x0y-127:....... > BER on eye border: 1.0

***************** scaled eye diagram for prescale value of -1 ****************************
+-----------------------------------------------------------------+
|                                                                 |
|                                                                 |
|                                                                 |
|                                X                                |
|                                                                 |
|                                                                 |
|                                                                 |
|                                                                 |
|                      z                                  z       |
|                                                                 |
|                                                                 |
|                                X                                |
|                                                                 |
|                                                                 |
|                                                                 |
|                                                                 |
|                                                                 |
+-----------------------------------------------------------------+
igd_eyescan run cX0Y8 ber=10 max_circle_div=4 dfe=1 init_window=1
x32y0:...... > eye border found in armed state.... > BER on eye border: 4.656683928435187e-10
x-32y0:..... > eye border found in armed state... > BER on eye border: 9.167846484106776e-10
x0y127:....... > eye border found in armed state.... > BER on eye border: 1.3096923548723964e-10
x0y-127:....... > eye border found in armed state.... > BER on eye border: 1.4552137276359966e-11

***************** eye diagram for target BER of 1e-10 ****************************
+-----------------------------------------------------------------+
|                                                                 |
|                                                                 |
|                                                                 |
|                                .                                |
|                                                                 |
|                                                                 |
|                                                                 |
|                                                                 |
|                      9.                              X..9       |
|                                                                 |
|                                                                 |
|                                a                                |
|                                                                 |
|                                                                 |
|                                                                 |
|                                                                 |
|                                                                 |
+-----------------------------------------------------------------+
```

X18511-041217

Figure 17: **Vector Scan Eye Plot for `igd_eyescan`**



```
igd_eyescan run cXOY8 prescale_min=4 prescale_max=4 use_matrix=1 matrix_xaxis_div=3 matrix_yaxis_div=4 dfe=1 init_window=1
000000
001100
01.610
02...1
01..20
001100
000000

**************** scaled eye diagram for prescale value of 4 ****************************
+-----------------------------------------------------------------+
|                                                                 |
|                                                                 |
|                                                                 |
|          0         0         0         0         0         0    |
|                                                                 |
|          0         0         1         1         0         0    |
|                                                                 |
|          0         1         .         6         1         0    |
|          0         2         .         .         .         1    |
|                                                                 |
|          0         1         .         .         2         0    |
|                                                                 |
|          0         0         1         1         0         0    |
|                                                                 |
|          0         0         0         0         0         0    |
|                                                                 |
|                                                                 |
+-----------------------------------------------------------------+
igd_eyescan run cXOY8 prescale_min=4 prescale_max=4 use_matrix=1 matrix_xaxis_div=3 matrix_yaxis_div=1 matrix_yaxis_offset=-10 dfe=1 init_window=1
igd_eyescan run cXOY8 prescale_min=4 prescale_max=4 use_matrix=1 matrix_xaxis_div=1 matrix_yaxis_div=4 matrix_yaxis_offset=16 dfe=1 init_window=1
igd_eyescan run cXOY8 prescale_min=4 prescale_max=4 use_matrix=1 matrix_xaxis_div=1 matrix_yaxis_div=4 matrix_xaxis_offset=16 dfe=1 init_window=1
0
0
2
.
3
1
0

**************** scaled eye diagram for prescale value of 4 ****************************
+-----------------------------------------------------------------+
|                                                                 |
|                                                                 |
|                                                                 |
|                                                0                |
|                                                                 |
|                                                0                |
|                                                                 |
|                                                2                |
|                                                .                |
|                                                3                |
|                                                                 |
|                                                1                |
|                                                                 |
|                                                0                |
|                                                                 |
|                                                                 |
+-----------------------------------------------------------------+
```

X19287-060117

Figure 18: **Matrix Scan Eye Plot for `igd_eyescan`**

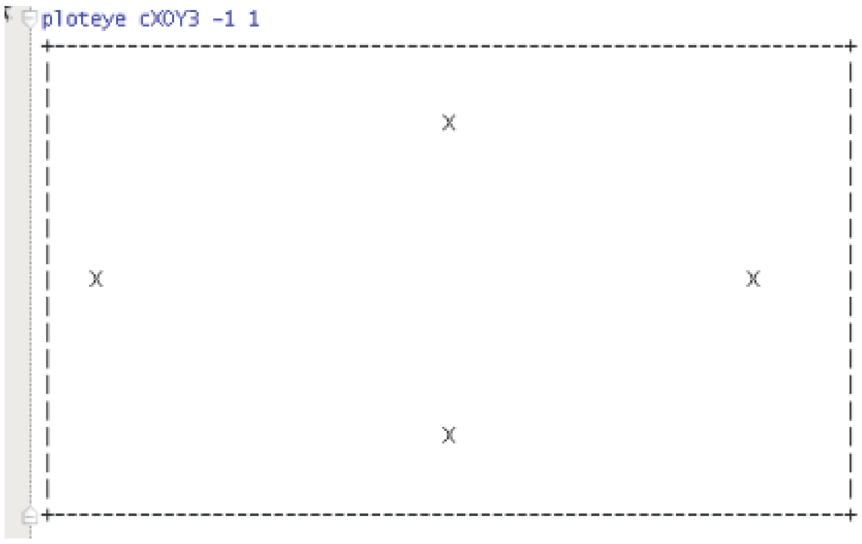
With the ploteye procedure, either the full eye or the scaled eye can be printed in the Tcl console.

ploteye <GT identifier> <selection> <prescale or BER coefficient> [<scaled>]

The `<GT identifier>` is the same as for DRP procedures. The `<selection>` can be either ps or ber. When ps is used, the next parameter is `<prescale>`. When ber is used, the next parameter is the BER coefficient (e.g., 8 for 10^{-8}). The `<prescale>` can be any value as the ones defined above for `prescale_min` and `prescale_max`. The `<scaled>` can be 1 or 0. It defaults to 1, which prints the scaled eye. The scaled plot only shows the edge for the eye of each vector (Figure 19). The full eye shows all measured points (Figure 20). The characters in the plot represent the following:

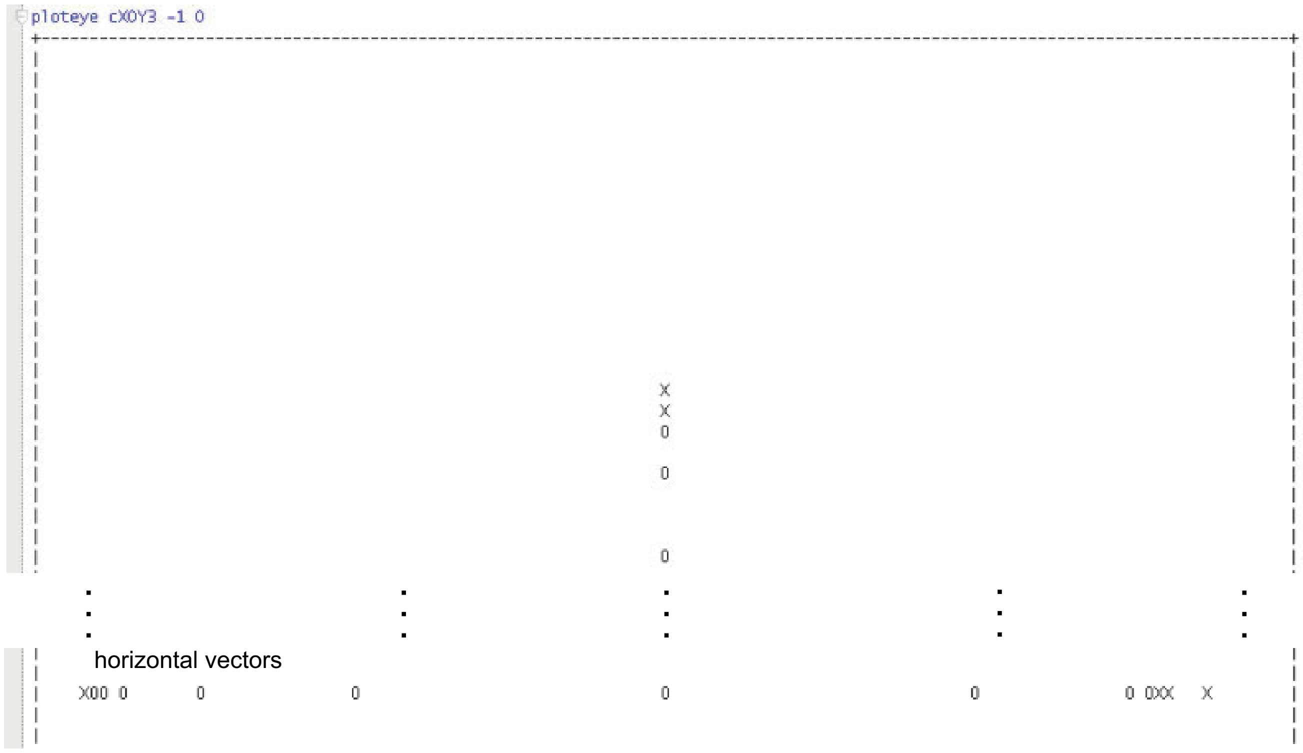
- X – fail, measured with ARMED scan (1.0)

- . – pass, no errors (0.0)
- 0-9 – represent measured BER with exponent 10^{-0} to 10^{-9}
- a-v – represent measured BER with exponent 10^{-10} to 10^{-32}
- z – anything else



```
ploteye cX0Y3 -1 1
```

Figure 19: **Scaled Eye Plot**



```
ploteye cX0Y3 -1 0
```

Figure 20: **Start and Horizontal Vectors of Full Eye Plot**

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 name	Erik Schidlack
Target devices	7 series, UltraScale, and UltraScale+ FPGAs
Source code provided	The `insert_gt_dbg.tcl` and `igd_eyescan.tcl` are encrypted. Other supporting files and generated files are open.
Source code format	Tcl, Verilog
Design uses code and IP from existing Xilinx application note and reference designs or third party	No, only IP provided from Vivado tools installation
Simulation	
Functional simulation performed	N/A
Timing simulation performed	N/A
Test bench used for functional and timing simulations	N/A
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/versions used	Vivado tools 2014.4 and later
Implementation software tools/versions used	Vivado tools 2014.4 and later
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	AC701, KC705, VC709, KCU105, KCU110, KCU1250, and VCU1287 boards

Conclusion

This application note provides an easy way to insert debug logic into an existing design and to allow automated debug through Tcl scripts.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

1. 7 series transceivers user guides:
    - *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)
    - *7 Series FPGAs GTP Transceivers User Guide* (UG482)
2. UltraScale architecture transceivers user guides:
    - *UltraScale Architecture GTH Transceivers User Guide* (UG576)
    - *UltraScale Architecture GTY Transceivers User Guide* (UG578)
3. Vivado Design Suite User Guide: Programming and Debugging (UG908)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/19/2017	1.0	Initial Xilinx release.

Please Read: Important Legal Notices