FPGA 中級トレーニングコース
Exercise manual

2023.06.14

KEK IPNS E-sys

Ryotaro Honda, Yun-Tsung Lai

The device for this exercise

AMANEQ
(A main electronics for network oriented trigger-less data acquisition system)

AMANEQ is general-purpose electronics developed for J-PARC experiments. It is a circuit board with Xilinx Kintex-7, which is a high-performance FPGA with many general-purpose IOs, DDR3-SDRAM, and 10 Gbps optical fiber for communication. In this exercise, you will write the firmware, download it to the FPGA, and actually run it to check the result.

See the Open-It page for a details of AMANEQ. Materials used in the Physics Society have been uploaded:
https://openit.kek.jp/project/StrHRTDC

AMANEQ' s photo and the functionalities used in this exercise

Fig. 1 shows the photo of AMANEQ. In this exercise, AMANEQ is powered by an AC/DC adapter. Connect the provided adapter to the jack.
AMANEQ is equipped with Kintex-7 FPGA with speed grade -2. EX1 and EX3 check the internal operation with ILA without communicating to the outside of the FPGA. The wiring pattern of the child board connected to the left side of AMANEQ is directly connected to the PAD of the FPGA. For EX2-1 and EX2-2, make a signal loopback on the slave board and check the operation of the IOSERDES. This circuit has two SFP+ module slots. Each is routed to a multi-gigabit transceiver (MGT) bank. Use these to learn how the high-speed serial communication works with EX4.

Common points in examples

About Vivado project

FPGA: xc7k160tffg676-2
(speed grade -2, XC7K-160T FPGA. Package is FFG 676.)

HDL file

Do coding on the file name as toplevel.vhd or toplevel.v without using hierarchical structure. (Except the modules ported from example design in EX4)

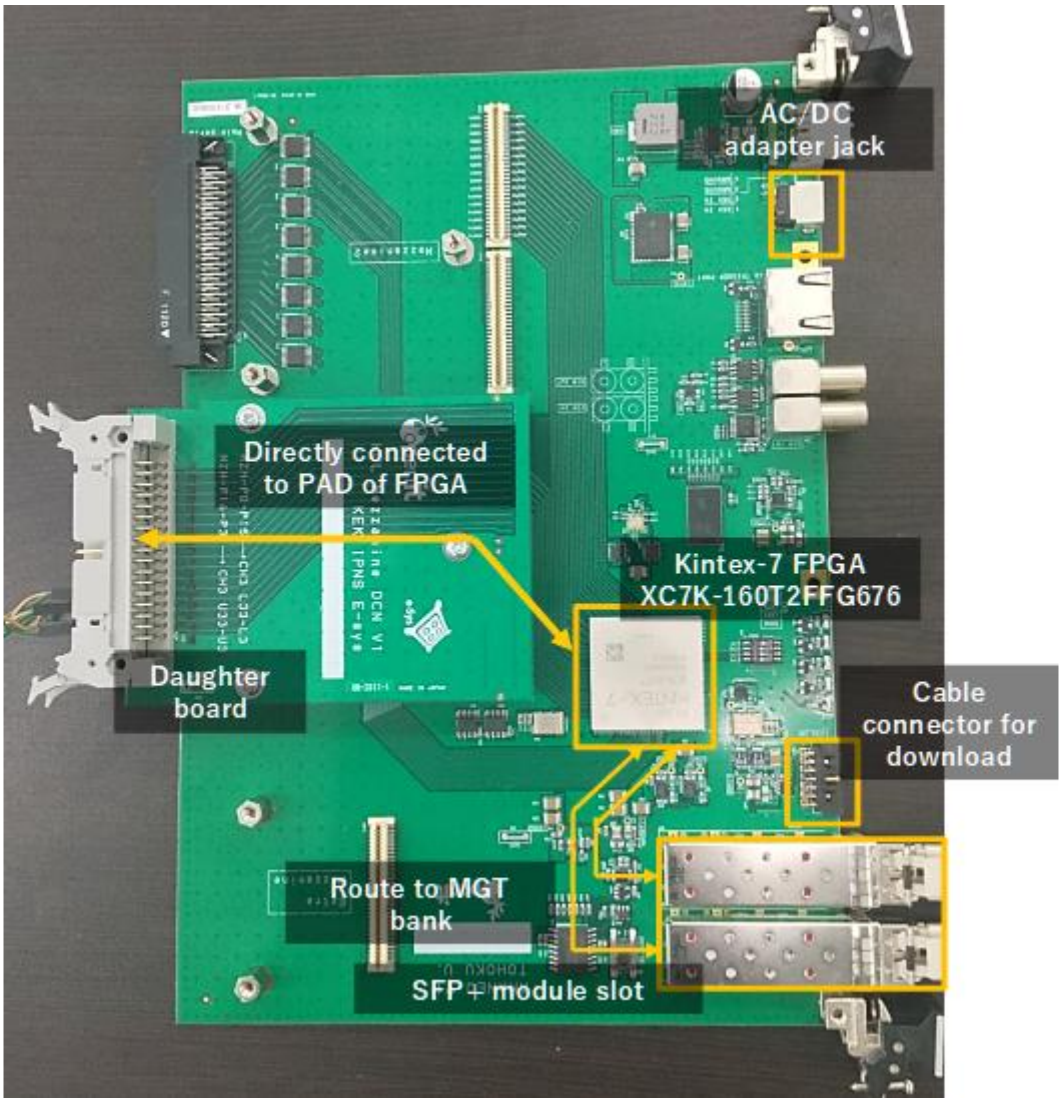
Constraint file

Please generate three types of constraint files: pins.xdc, timing.xdc, and impl.xdc. Make pin-related constraints in pins.xdc, and timing constraints in timing.xdc. Also, specify impl.xdc as the target constraint file, and set impl.xdc as the constraint file to where the ILA description is written when doing setup debug.

Top-level port pin
CLKP, CLKN
・100 MHz oscillator input.
・Constrain it as LVDS_25.

RSTB_SW
・Push button SW2 on the board.
・The signal switches from 1 to 0 when the push button is pressed.
・Constrain it as LVCMOS33.

About the reset logic
・Active high. The RSTB_SW signal should be inverted.


AC/DC adapter jack
Directly connected to PAD of FPGA
Kintex-7 FPGA XC7K-160T2FFG676
Daughter board
Cable connector for download
Route to MGT bank
SFP+ module slot
Fig. 1 : AMANEQ' s photo and the functionalities used in this exercise

EX1

Implementing a moving average filter using DSP

Firmware construction

Fig. 2 shows the block diagram of EX1. The areas surrounded by dashed lines are top-level ports which are connected to FPGA pads. Synchronize the RSTB_SW signal using double FF after sandwiching the inverter. Distribute the signal after being synchronized to each module as a reset signal.

Please generate the input (integer part) to the DSP with an 8-bit free-running counter. Connect to the A and D ports of the DSP after inserting appropriate delay elements. At that time, extend the lower 2 bits (decimal part) and fill them with 0.

Please use the CLKP and CLKN inputs as global clock.
Please give mark_debug constraint to the signal lines marked with star, so that it is routed to ILA.

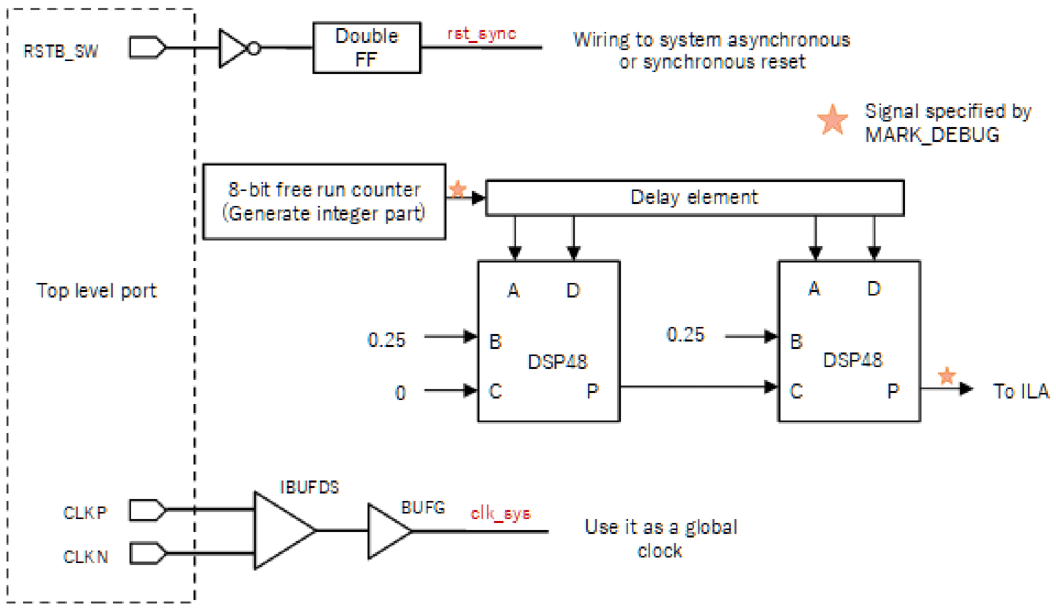

RSTB_SW
Double FF
rst_sync
Wiring to system asynchronous or synchronous reset
Signal specified by MARK_DEBUG
8-bit free run counter (Generate integer part)
Delay element
Top level port
A
D
B
C
P
DSP48
0.25
0
To ILA
CLKP
CLKN
IBUFDS
BUFG
clk_sys
Use it as a global clock

Fig. 2 : Block diagram of EX I

Design condition

The synchronous circuits including DSP belong to the clk_sys' s domain. Write the input clock' s constraints in tining.xdc.

Make the pin constraint of the top-level port as following.

Port name	Pin number	IO spec.	Other constraints
CLKP	F22	LVDS_25	DIFF_TERM
CLKN	E23	LVDS_25	DIFF_TERM
RSTB_SW	H9	LVCMOS33	

Input operand to DSP

・signed fixed point (integer part 8-bit, decimal part 2-bit)

- The input operands looks signed to the DSP. It doesn't mean you have to figure out the 2's complement by yourself.

・The integer part is given by the counter, and the decimal part is set as 0 at the input.

- If you have time, you can try to use other input patterns.

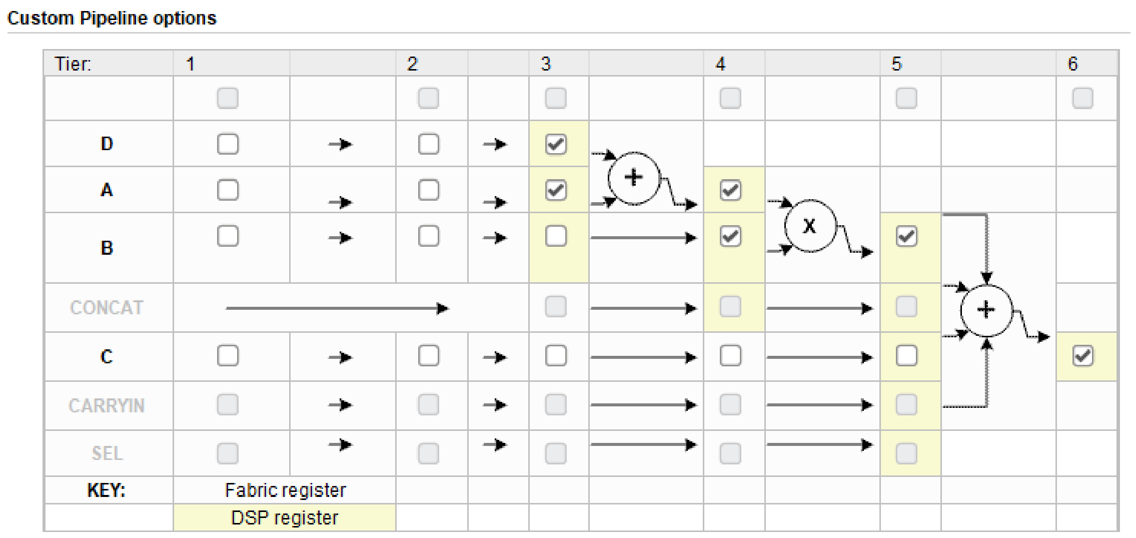
Conditions to generate DSP

Instruction for implementation

・(A+D)*B+C

Port width
・A, B, D: 10 bit
・C, P: 48 bit

Pipeline register’ s setup



Custom Pipeline options

Tier:	1	2	3	4	5	6
	☐	☐	☐	☐	☐	☐
D	☐	☐	✓			
A	☐	☐	✓	✓		
B	☐	☐	☐	✓	✓	
CONCAT			☐	☐	☐	
C	☐	☐	☐	☐	☐	✓
CARRYIN	☐	☐	☐	☐	☐	
SEL	☐	☐	☐	☐	☐	

KEY: Fabric register, DSP register

Fig. 3 : DSP Pipeline register’ s setup

Check the operation

・ Please check if the output of the moving average filter is as our expectation.

・ Please check if the output delay for the input is correctly set as 5.

The result of multiplication has an integer part of 20 bits and a decimal part of 4 bits. It may be easier to check if the integer part and the decimal part of the output P are separated and routed to the ILA.

EX2
Implementing IOSERDES (without IDELAY)

Firmware constructoin

EX2’ s block diagram is shown in Fig. 4. This example uses an MMCM to generate global clocks of 100 MHz and 400 MHz from an oscillator clock. Those two clocks are used to drive the OSERDES. The 100 MHz clock is used for parallel data capture, and the 400 MHz clock is used for serial data transmission. The transmission data rate is 800 Mbps since the OSERDES is driven in DDR mode. Input a fixed pattern to the parallel data input to OSERDES. Route the output of the OSERDES directly to a top-level port. Connect SDOP (N) to SDIP (N) with a cable on the board as a loopback.

The clock forward signal (CLK_FWD_O_P(N)) is also connected to CLK_FWD_I_P(N) by a cable outside the FPGA as a loopback. This signal is the 400 MHz clock input to the OSERDES, so BUFIO and BUFR with a divide-by-4 setting would generate 400 MHz and 100 MHz clocks. These clocks are not global clocks without BUFGs. The generated 100 MHz regional lock drives the ISERDES, the VIO, and the circuit for edge detection.

The ISERDES performs bitslip while the bitslip input is 1. It is necessary to input a one-shot pulse to check the state of the bitslip one by one. Since VIO does not have a function to output a one-shot pulse, please make a circuit to detect the rising edge of the 1st bit of probe_out0 and extracts the edge.

Design conditions

Assume that the circuit for transmitting belongs to the 100 MHz global clock domain, and the circuit for receiving belongs to the 100 MHz regional clock domain. Define a 100 MHz clock for the oscillator and a 400 MHz loopback forward clock in timing.xdc.

Make the pin constraint of the top-level port as following.

Port name	Pin number	IO spec.	Other constraints
CLKP	F22	LVDS_25	DIFF_TERM
CLKN	E23	LVDS_25	DIFF_TERM
RSTB_SW	H9	LVCMOS33	
SDOP	D26	LVDS_25	
SDON	C26	LVDS_25	
CLK_FWD_O_P	H23	LVDS_25	
CLK_FWD_O_N	H24	LVDS_25	
SDIP	G24	LVDS_25	DIFF_TERM
SDIN	H24	LVDS_25	DIFF_TERM
CLK_FWD_I_P	G22	LVDS_25	DIFF_TERM
CLK_FWD_I_N	H24	LVDS_25	DIFF_TERM

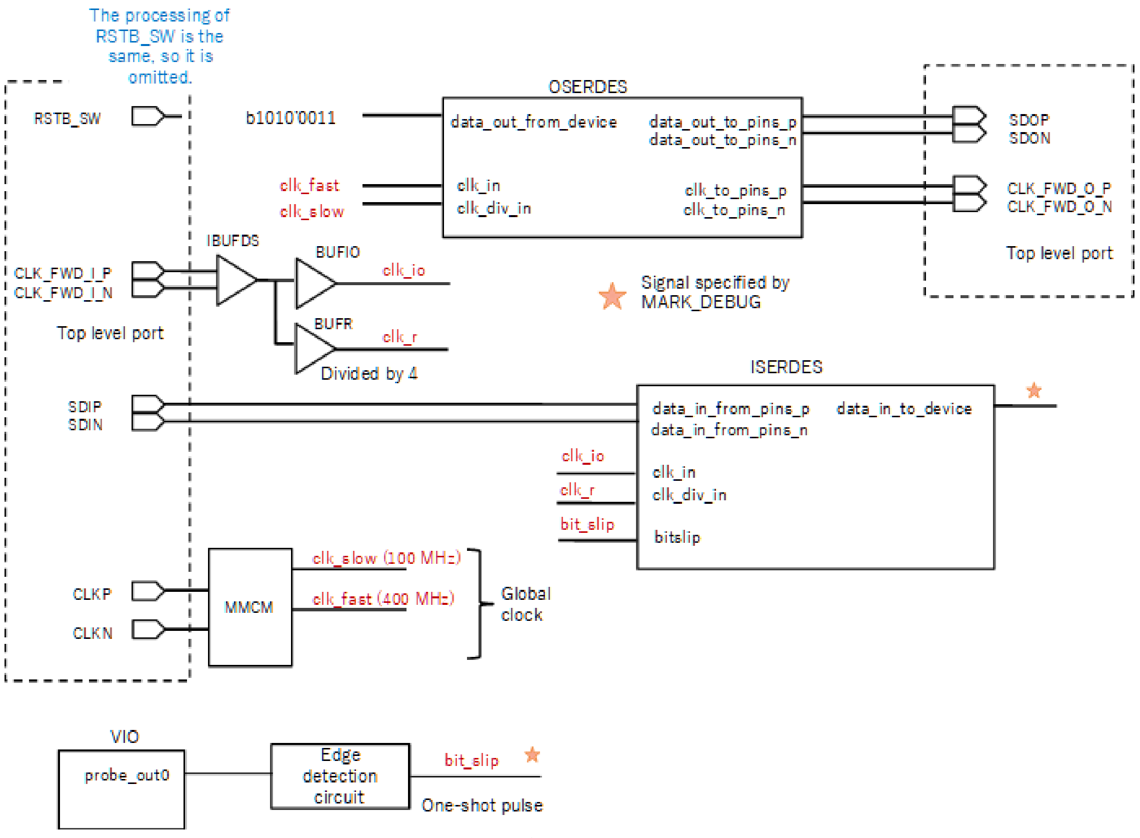


Fig. 4 : EX2' s block diagram

Conditions to generate OSERDES

· Interface type:	Custom
· Data Bus Direction:	Output
· Data Rate:	DDR
· Serialization factor:	8
· External Data Width:	1
· I/O signaling:	Differential, LVDS_25
· Clock strategy:	Internal clock
· Clock forwarding:	Enabled

Conditions to generate ISERDES

· Interface type:	Custom
· Data Bus Direction:	Input
· Data Rate:	DDR
· Serialization factor:	8
· External Data Width:	1
· I/O signaling:	Differential, LVDS_25

・Clock strategy:	Internal clock
・Data delay type:	None
・Clock delay type:	None

Wiring on the board

Do the wiring just as Fig. 5 to form loopback.

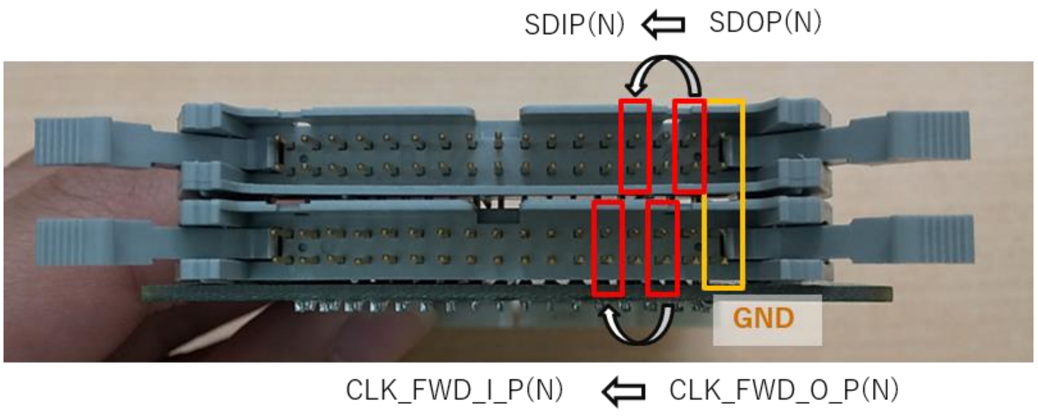


Fig. 5： loopback

Check the operation

・After downloading the bit file and ltx file to the FPGA, check if the ILA starts up. If the loopback connection is wrong, the regional clock driving the ILA will be lost, and the ILA will not run.

・Check the bit pattern of the ISERDES’ s data output. It's probably not the bit pattern that we sent.

・Check if a bit shift is performed each time VIO's probe_out0 transits from 0 to 1. Also make sure the shift pattern matches the one described in UG471.

EX3
ECC implementation for memory resources

Firmware construction

Fig. 6 shows EX3’ s block diagram. In this example, we try two ECC implementations. Generate block RAM with SDP mode and give it input data, address and write enable from VIO. Write enable is given by the 3rd bit of VIO probe_out0. Generate artificial data corruption into the block RAM using injectsbiterr (single bit error) or injectdbiterr (double bit error). These signals must be one-shot pulses, so detect the edges of the 1st and 2nd bits of probe_out0 to generate one-shot pulses. Also, error injection requires seeting write enable as 1.

Next, implement the ECC encoder and decoder. These are used to add ECC to memory resources without builtin ECC functionality, such as distributed RAM or external SDRAM chips. Give the data from VIO and the bit-reversed pattern to the check bits from VIO to simulate data corruption, and connect it to a XOR gate with the output of the encoder. This allows the insertion of single or multiple bit errors at arbitrary bit positions.

Design conditions

The synchronous circuits with DSP belong to the clk_sys’ s domain. Write the input clock constraints in tining.xdc.

Make the pin constraint of the top-level port as following.

Port name	Pin number	IO spec.	Other constraints
CLKP	F22	LVDS_25	DIFF_TERM
CLKN	E23	LVDS_25	DIFF_TERM
RSTB_SW	H9	LVCMOS33	

Conditions to generate Block RAM

Interface type:	Native
Memory type:	Simple dual port RAM
ECC type:	Builtin ECC
Error injection pins:	Enabled
Port A width:	64
Port A depth:	16

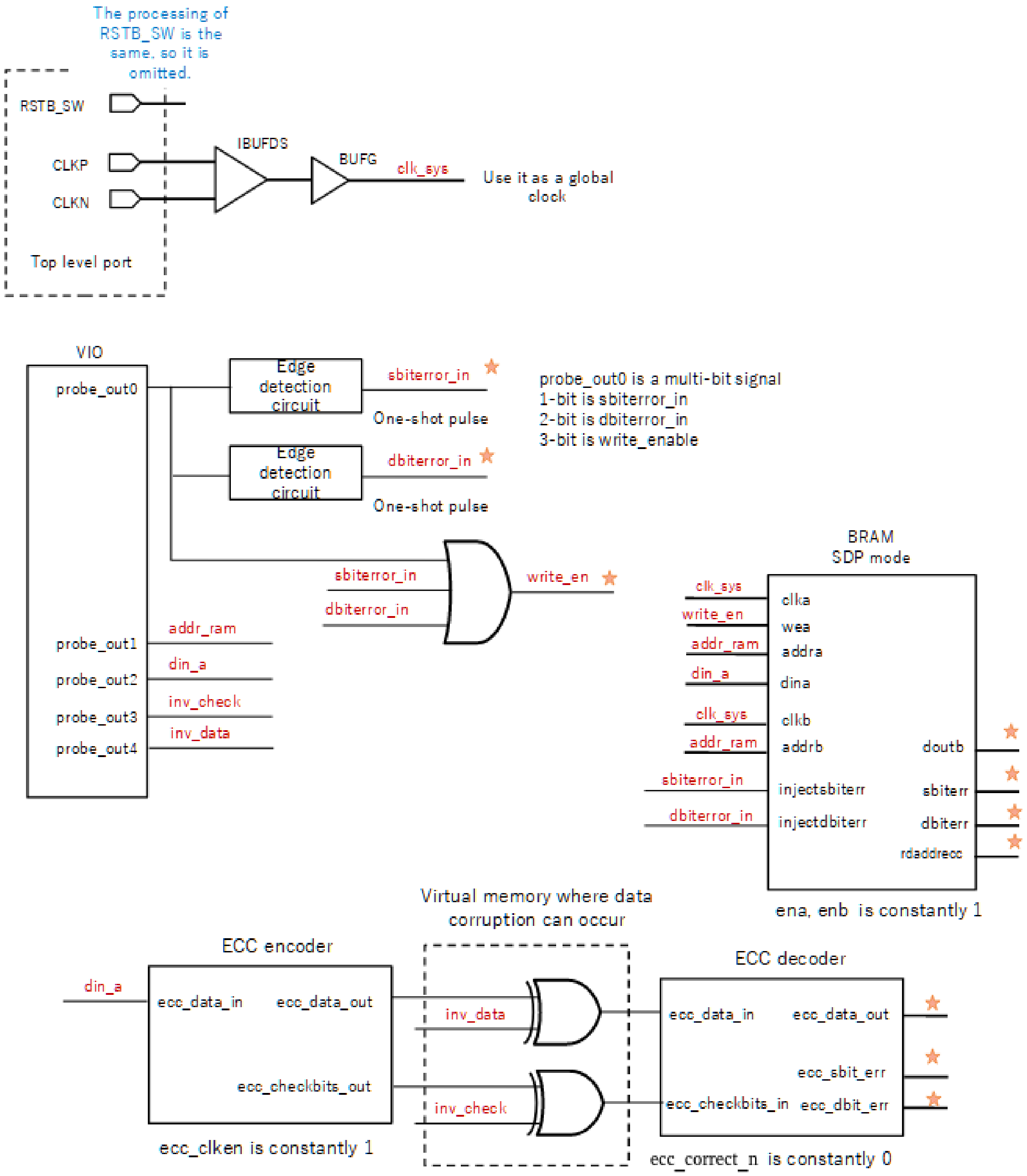


Fig. 6：EX3’ s block diagram

Conditions to generate ECC encoder

Algorithm Options:	Hamming
ECC mode:	Encoder
Data width:	64
Enable input registering:	enabled
Enable output registering:	enabled

Conditions to generate ECC decoder

Algorithm Options:	Hamming
ECC mode:	Decoder
Data width:	64
Enable input registering:	enabled
Enable output registering:	enabled

Check the operation

・Asserting injects(d)biterr causes data corruption in the memory contents stored at the location specified by the address. Let's check how the values of doutb, sbiterr, dbiterr, and rdaddrecc change after generateing an error. ECC also corrects the data output but leaves the memory contents unchanged. When specifying an address without other error, and then specifying the address with data corruption, and reading again, let's see what happens.

・For the ECC encoder/decoder, check that the data given to the encoder is correctly output from the decoder before changing the pattern of inv_data and inv_check. Then change the inv pattern and see how the ECC decoder signal changes.

Development topic H I
Implementing IOSERDES (with IDELAY)

Firmware construction

Fig. 7 shows H I’ s block diagram. For the difference from EX2, the ISERDES has a port for IDELAY control and it requires a reference clock for IDELAYCTRL. Generate a 200 MHz clock with MMCM and input it to the ref_clock port of the ISERDES. This example uses in_delay_tap_in for IDELAY control. Give the desired number of taps to this port from VIO. in_delay_tap_in is reflected when in_delay_reset becomes 1. Detect the edge of the 2nd bit of probe_out0 of VIO and connect it to this port as the load_idelay signal.

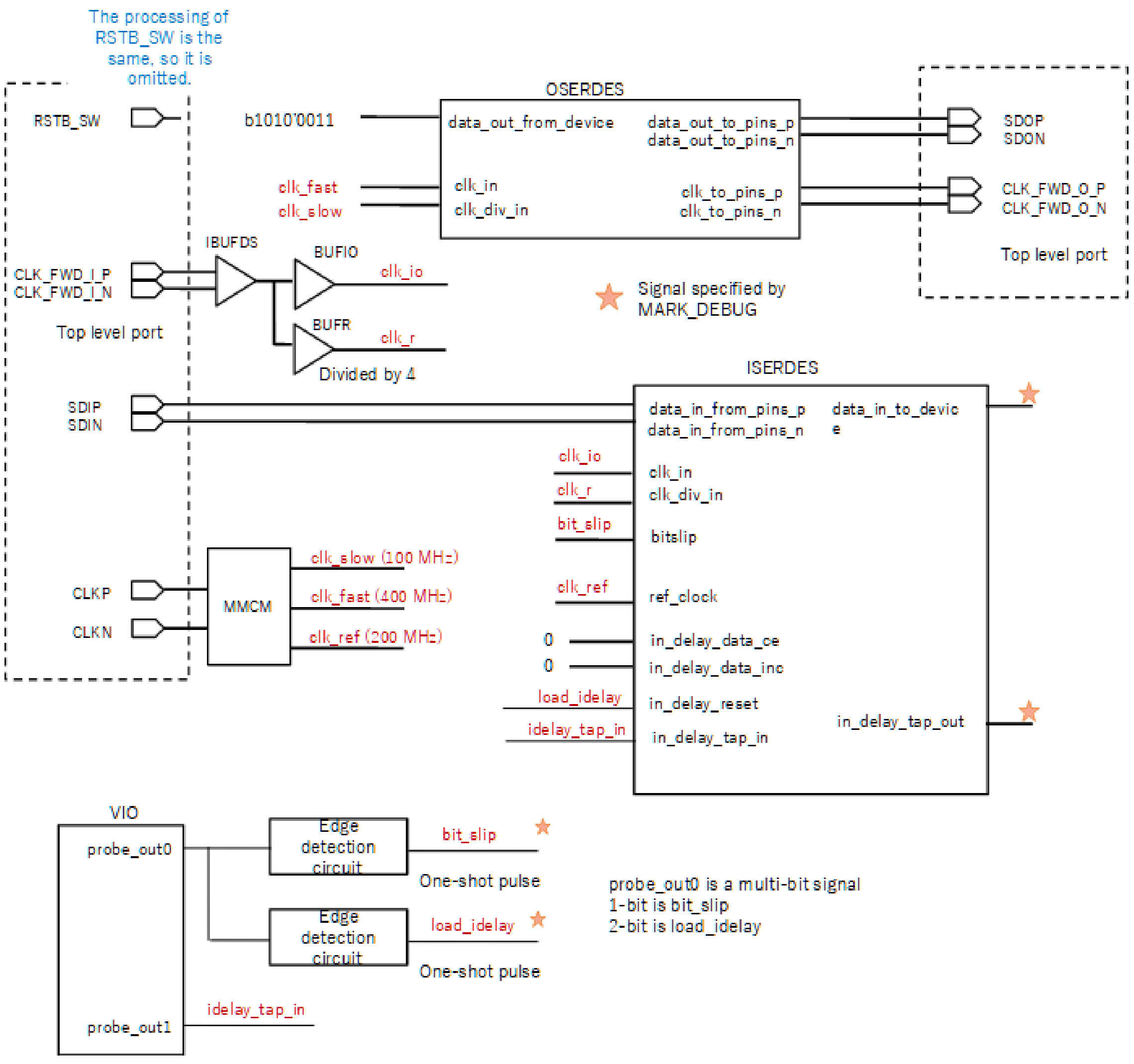


Fig. 7：H I’ s block diagram

Design condition

Use the same constraint as the constraint files of EX2.

Conditions to generate OSERDES

・Interface type:	Custom
・Data Bus Direction:	Output
・Data Rate:	DDR
・Serialization factor:	8
・External Data Width:	1
・I/O signaling:	Differential, LVDS_25
・Clock strategy:	Internal clock
・Clock forwarding:	Enabled

Conditions to generate ISERDES

・Interface type:	Custom
・Data Bus Direction:	Input
・Data Rate:	DDR
・Serialization factor:	8
・External Data Width:	1
・I/O signaling:	Differential, LVDS_25
・Clock strategy:	Internal clock
・Data delay type:	Variable loadable
・Clock delay type:	None
・Include IDELAYCTRL:	Enabled

Check the operation

- The amount of delay for 1 tap of IDELAY is 100 ps or less. The data_in_to_device pattern does not change immediately. IDELAY is controlled correctly if in_delay_tap_out changes after load_idelay is asserted. Let's check it first.
- The maximum number of taps for IDELAY is 32. Somewhere, you should find a tap setting making the data_in_to_device output pattern unstable. Let's find it.

Development topic H2
Implementing Aurora8b10b

Firmware construction

Fig. 8 shows H2’s block diagram. Since it is difficult to do all the wiring for Aurora8b10b, only the clock wiring is described in this block diagram. For other signal lines, please refer to the example design of Aurora8b10 IP and the source codes of the teaching material for this exercise. The Aurora8b10b core requires GTREFCLK and an independent clock. Generate a 100 MHz clock from CLKP (N) and use it as a clock which is independent of GTREFCLK. The Aurora8b10 core user interface is synchronous to user_clk. We also need a sync_clk for the GTX channel. Both of these two are from tx_out_clk. user_clk is a global clock, and the peripheral circuits of Aurora8b10b should also be driven by user_clk.

Please port gt_common_wrapper and tx_clock_module, and support_reset_logic from the Aurora8b10b example design although not shown in this diagram. A submodule of support_reset_logic, cdc_sync_exdes, also needs porting.

Design condition

Define the GTREFCLK in timing.xdc. AMANEQ's GTREFCLK oscillator is 156.25 MHz. Also, false_path has to be specified. A reference xdc file exists in the imports directory of the Aurora8b10 core’s example design. Please port the part of description while referring to it. If you don't know, please refer to the source code in the teaching material (answer).

Make the pin constraint of the top-level port as following.

Port name	Pin number	IO spec.	Other constraints
CLKP	F22	LVDS_25	DIFF_TERM
CLKN	E23	LVDS_25	DIFF_TERM
RSTB_SW	H9	LVCMOS33	
GTREFCLK_P	D6		
GTREFCLK_N	D5		
GTX_TXP	D2		
GTX_TXN	D1		
GTX_RXP	G4		
GTX_RXN	G3		

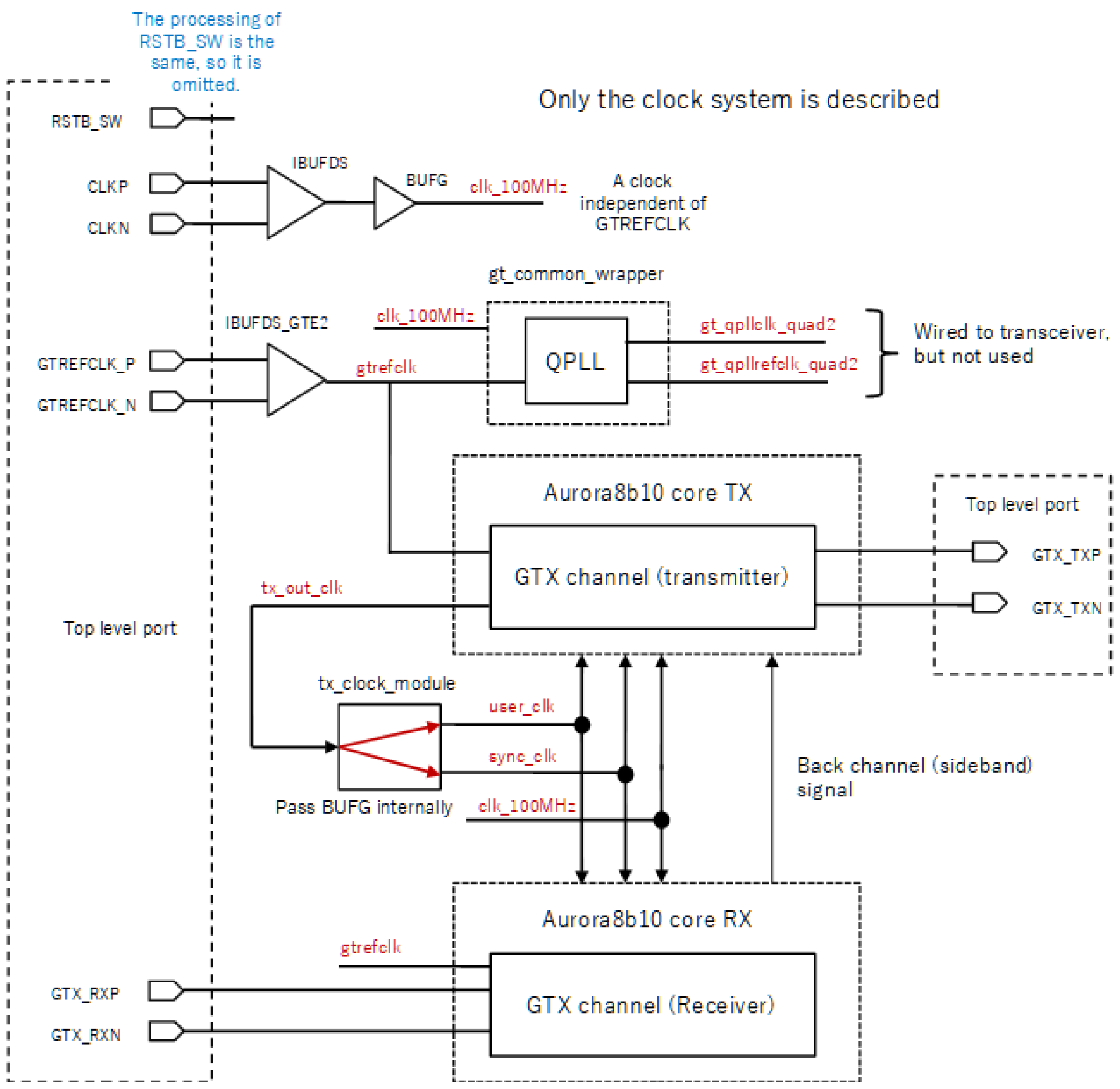


Fig. 8 : H2' s block diagram

Conditions to generate Aurora8b10 core

Lane Width:	2
Line rate:	1.25 Gbps
GT refclk:	156.25 MHz
INIT clk:	100 MHz
DRP clk:	100 MHz
Dataflow mode:	Simplex
Interface:	Framing
Flow control:	None
Back channel:	Sidebands

Choose to include the shared logic in the example design.

AXI4-Stream’ s control

Connect the output of the 16-bit counter to s_axi_tx_tdata. Controlling the three signals, tvalid, tlast, and tkeep is needed to input frame-structured data to the user interface on the TX side. Connect tx_channel_up to tvalid. This makes tvalid transit to 1 when the core is ready for transmission. In this time, we will define 16-byte as one frame. Every time tlast transiting to 1, data is input to the core 8 times, and the core inserts control characters marking the start and end of an Aurora frame. While inserting control characters, tready is 0 and user data is not fetched. Data is captured by the core when both tvalid and tready are 1, so stop incrementing the 16-bit counter while tready is 0. Always put 0 for tkeep.

Necessity of back channel

When the RX side starts the initialization process, it informs the TX side through the back channel. Based on it, the TX side controls the data flowing to the lane and runs the core initialization process. In the case of full-duplex, doing this on both ends enables the correct transceiver and core initialization. In the case of Simplex, the status of each other cannot be known as a one-way street, so simplex core initialization is harder than full-duplex

In this example, we will intentionally use two GTX channels and connecte the cores implemented in simplex with a back channel. In full-duplex, such connections are made inside the core.

If you still have time, try to disconnect the back channel and see what happens. The lane will come up but the core will not be able to initialize. For this, the importance of the back channel and the difficulty of simplex can be seen.

If it is difficult for you

Consider the following points by comparing the distributed source code in the teaching material, and the source code inside the imports of the Aurora8b10b example design.

- Which modules inside the imports have been ported to the material source code?
- What kind of changes have been made to the port?

If you just want to drive the Aurora8b10b, you should notice that the example design source code can be used directly as it is. If you understand the following: Which modules should be ported, what changes should be made, and why, then you should be able to do exercise H2.

Check the operation

・Check if lane_up and channel_up of TX and RX are set to 1. If channel_up remains

0, you may have forgotten to connect the back channel.

・Check if the data sent by TX match the data received by RX or not, and if tlast on the RX side transits according to the frame structure correctly. If the TX and RX data do not match (data jumps on the RX side) with each other, the tx_tready monitoring and circuit response to it are wrong.

Development topic H3
GbE communication with a GTREFCLK of 156.25 MHz and the way to modify IP

This exercise is for people who usually use SiTCP[1]. In this exercise, instead of writing your own HDL codes, we consider the relation between the GTREFCLK frequency and the clock frequency inside the FPGA based on the distributed codes, and then modify the IP based on it.

In this exercise, you will learn how to rewrite the contents of the IP by the user. Please fully understand that changing an incorrect IP has the risk of causing serious trouble.

Relation between SiTCP and PCS/PMA

Fig. 9 shows the relation between PCS/PMA and SiTCP for GbE communication using MGT. Since GbE uses 8b10b, the serial link' s line rateis fixed at 1.25 Gbps. First, we need to enable the PMA-TX to output data at this line rate. Next, GMII is used to exchange data between PCS and SiTCP. A 125 MHz clock is absolutely necessary here to produce a data rate of 1 Gbps. These two points must be suppressed for GbE communication.

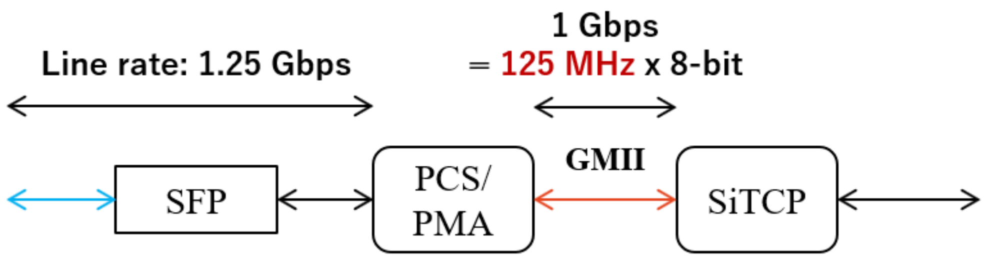


Fig. 9: Relation between SiTCP and PCS/PMA

Relation between GTREFCLK frequency and generated clock frequency

In many cases, PCS/PMA for GbE is generated in the IP Catalog. GTREFCLK is assumed to be input at 125 MHz, and the TX side' s clock wiring is specified as in Fig. 10. From here, it is assumed that both the TX line rate and TXOUCLK depend on GTREFCLK, but it seems that the TX line rate can be changed by changing the CPLL settings. But what about TXOUTCLK?

[1] SiTCP is BBT's product: https://www.bbtech.co.jp/sitcp/

From here, please refer to the H3 project for distribution codes. There are several modules in toplevel.vhd, but for this exercise, we need GtClockDistributor2, GbEPcsPma and SiTCP. The clock wiring diagrams are shown in Fig. 11. This is the exercise. Please try to answer the following questions.

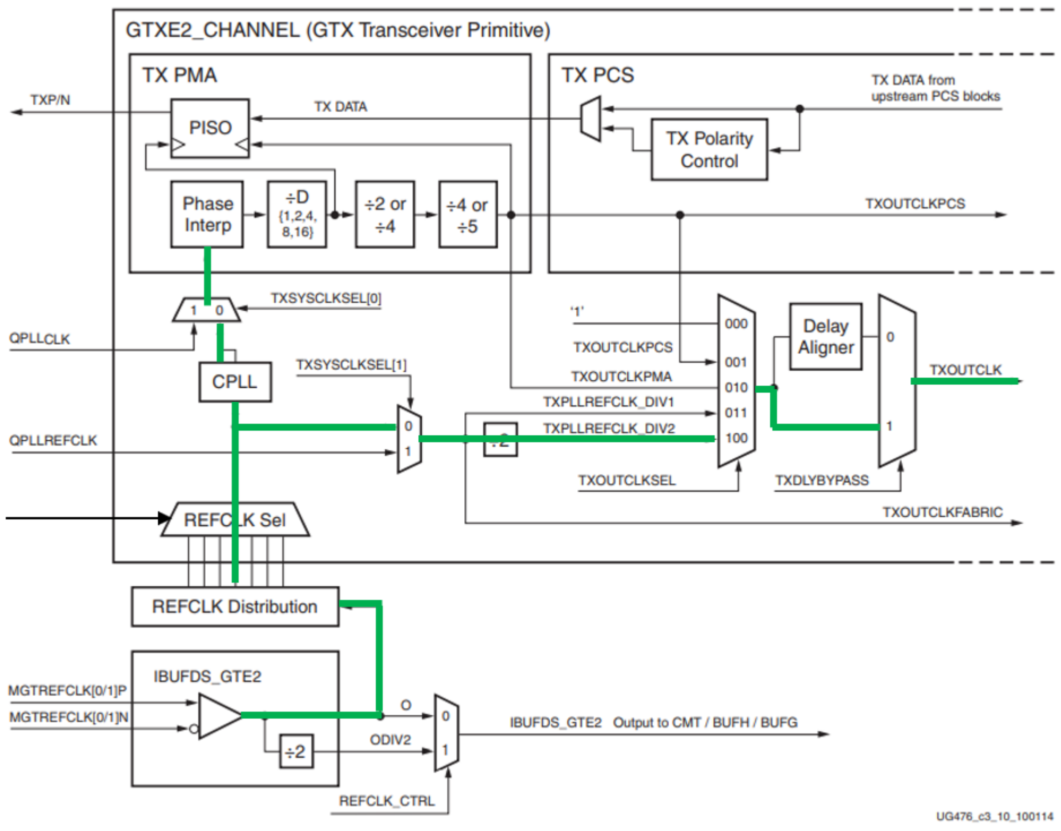


Figure 3-28: **TX Serial and Parallel Clock Divider**

Fig. 10: TX clock route when generating PCS/PMA from IP Catalog for GbE

Q1

Complete the wiring diagram in Fig. 11 based on the distributed source code.

Q2

Enter the frequencies of TXOUTCLK, RXOUTCLK, USERCLK, USERCLK2, and GMII_TX_CLK with GTREFCLK of 125 MHz and 156.25 MHz.

Q3

The oscillator in AMANEQ for GTREFCLK is 156.25 MHz. What kind of settings need to be changed for GbE communication?
Hint: there are two.

If Q3 is answered correctly, please proceed to Q4.

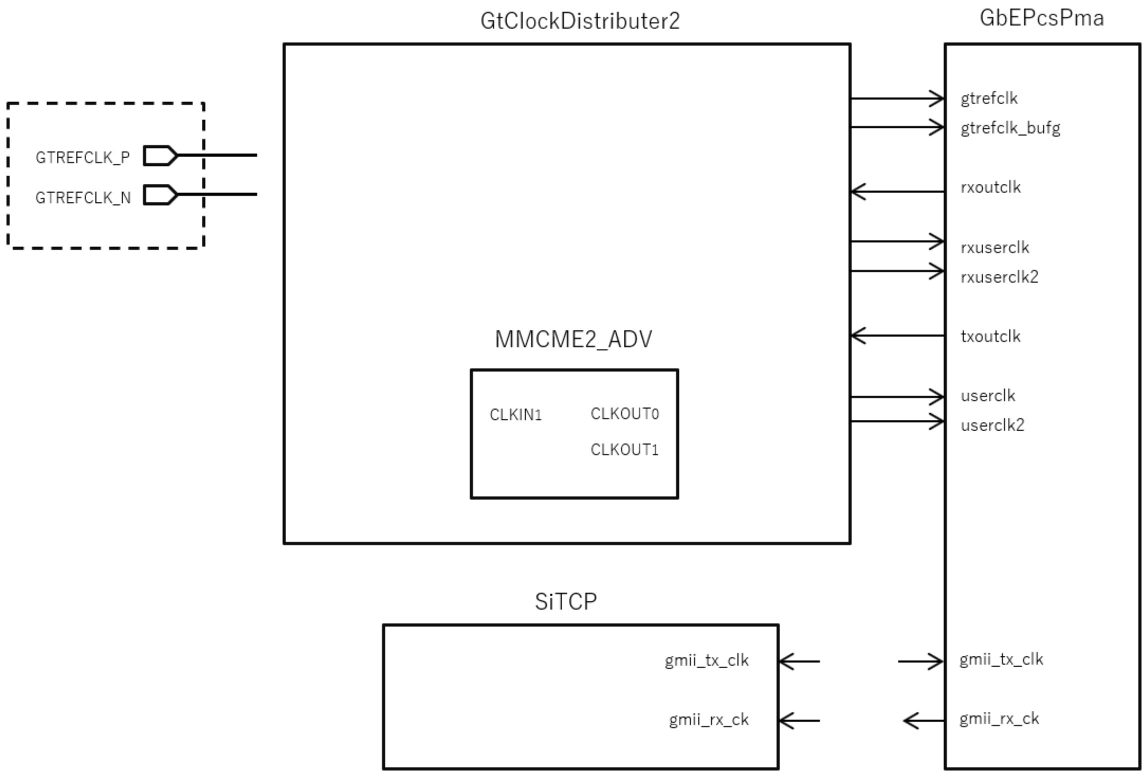


Fig. 11: Block diagram of the distributed code

Q4

Please change the settings of MMCM and CPLL, synthesize, and generate bitstream file. It is necessary to change the CPLL settings, but the CPLL settings are written inside the IP, and cannot be changed by the programmer. Generate a custom IP by allowing the user to change the IP contents by referring to the following.

You have to change both the HDL code and the IP's XDC file. Find the parts depending on the frequency of GTREFCLK, and change all of them.

Q5

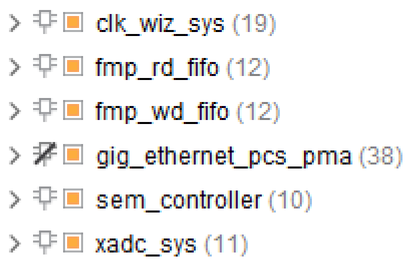
Please download the generated bitstream file to AMANEQ and confirm if ping is possible. Please talk to the lecturer when you do Q5.

That way to change IP

CPLL parameters are given by the generic port of GTXE2_CHANNEL, which is instantiated inside gig_ethernet_pcs_pma_gtwizard_gt.vhd from IP. This HDL code is controlled by Vivado, so any changes you make will not be reflected in the final IP product. To generate user-customed IP, it must be unmanaged by Vivado. To do so, run the following tcl command from the Tcl console.

```
set_property IS_MANAGED false [get_files <ip_name>.xci]
```
<ip_name> is gig_ethernet_pcs_pma in this case.

After running the commend, a slash will pop out as in Fig. 12. For the IPs in this state, they are out of Vivado management, and become user-managed. To make the necessary HDL or XDC changes and to include them in the final IP product, manually run the IP synthesis run by executing the following tcl command:



```
clk_wiz_sys (19)
fmp_rd_fifo (12)
fmp_wd_fifo (12)
gig_ethernet_pcs_pma (38)
sem_controller (10)
xadc_sys (11)
```

Fig. 12: User-managed IP

```
reset_run <ip_name>_synth_l
```
If running this command, the final product of the original IP will be destroyed.

```
launch_run <ip_name>_synth_l
```
Running this command will generate an IP including the changes you made. You can check if the dcp is updated to see if it's working.

Up to here, synthesize, and place and route the project as usual to generate a bitstream.

Development topic H4
Implementing a TDC with 1ns precision using ISERDES

There are several ways to implement a time-to-digital converter (TDC) using FPGA. But when the TDC LSB has an accuracy of about 100-1000 ps, it is common to use a multi-phase clock implementation. For example, a TDC with 1 ns accuracy can be implemented using 4-phase 250 MHz clocks. Although this method is well known, it is necessary to adjust the wiring length up to the D-FF where sampling is performed, and it is necessary to block the clock domain transfer according to the division ratio with the system clock. Know-how for implementation is needed.

Using the ISERDES primitive, a TDC with an accuracy of about 1 ns (depending on the maximum frequency which BUFG can handle) can be implemented with a much smaller amount of code than implementation using multi-phase clocks. In this exercise, we will implement a TDC that measures the rise and fall of a pulse with 1ns accuracy, and measure the pulse width of the input pulse.

Conceptual diagram of TDC using ISERDES

Fig. 13 shows the conceptual diagram. Let's say the ISERDES samples input pulses in 500 MHz DDR mode and outputs parallel data with a 125 MHz clock. Since these two clocks are divided by 4, the output data width is 8-bit. The ISERDES then takes a snapshot of the input pulse travel in every 8 ns (1/125 MHz) with 1 ns accuracy. At the exact timing when the pulse arrives, data with a mixture of 0 and 1 should be output as shown in the figure, and such data is called a thermometer code. By performing binary encoding on this, we get the fine count, which is the lower 3-bits of the TDC. Since the thermometer code is updated in every 8 ns, a free-running counter that counts up at 125 MHz should be used for the upper bits of TDC. Then we can obtain TDC data with a data structure of 3-bit fine count + N-bit coarse count.

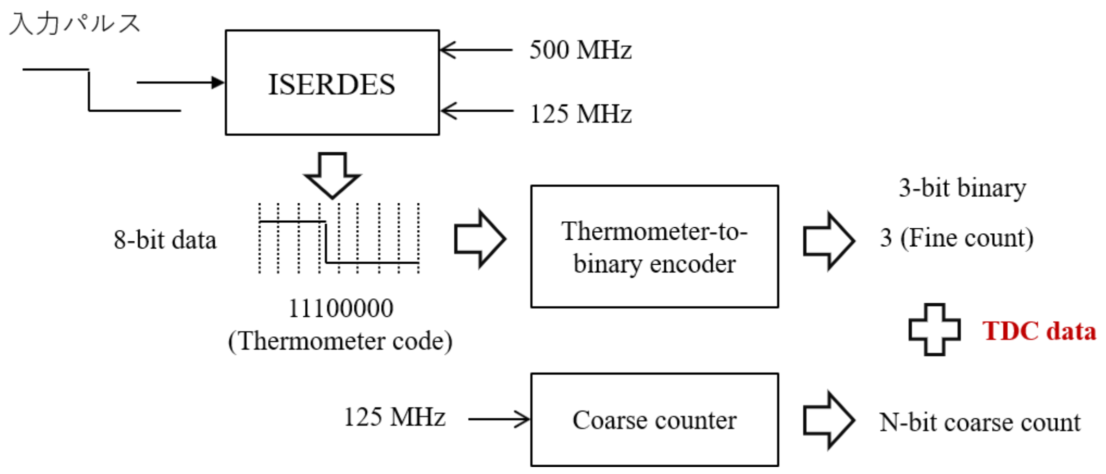


Fig. 13：The conceptual diagram of ISERDES TDC

Firmware construction

Fig. 14 shows the block diagram. H4 does not write down all the wiring like an example because it is a development exercise. In this exercise, the pulse is input via the NIM input port, and it is generated by a function generator. In this task, the fast clock and the input data are not synchronous to begin with, so we don't need to worry about the phase with the slow clock. That's why I don't use bitslip.

To measure the pulse width, both the leading edge and the trailing edge must be measured. It is sufficient to find the 0→1 transition for the leading edge and the 1→0 transition for the trailing edge. Encoders should output a binary value as well as a hit signal (data valid) at the same time. Let's use the hit signal to store it in a register along with the course counter value and subtract it to get the pulse width.

Please use ILA to monitor the signal in a suitable place to confirm the operation. In this exercise, we does not use VIO.

Design conditions

Make the pin constraint of the top-level port as following.

Port name	Pin number	IO spec.	Other constraints
CLKP	F22	LVDS_25	DIFF_TERM
CLKN	E23	LVDS_25	DIFF_TERM
RSTB_SW	H9	LVCMOS33	
NIM_IN	V8	LVCMOS15	

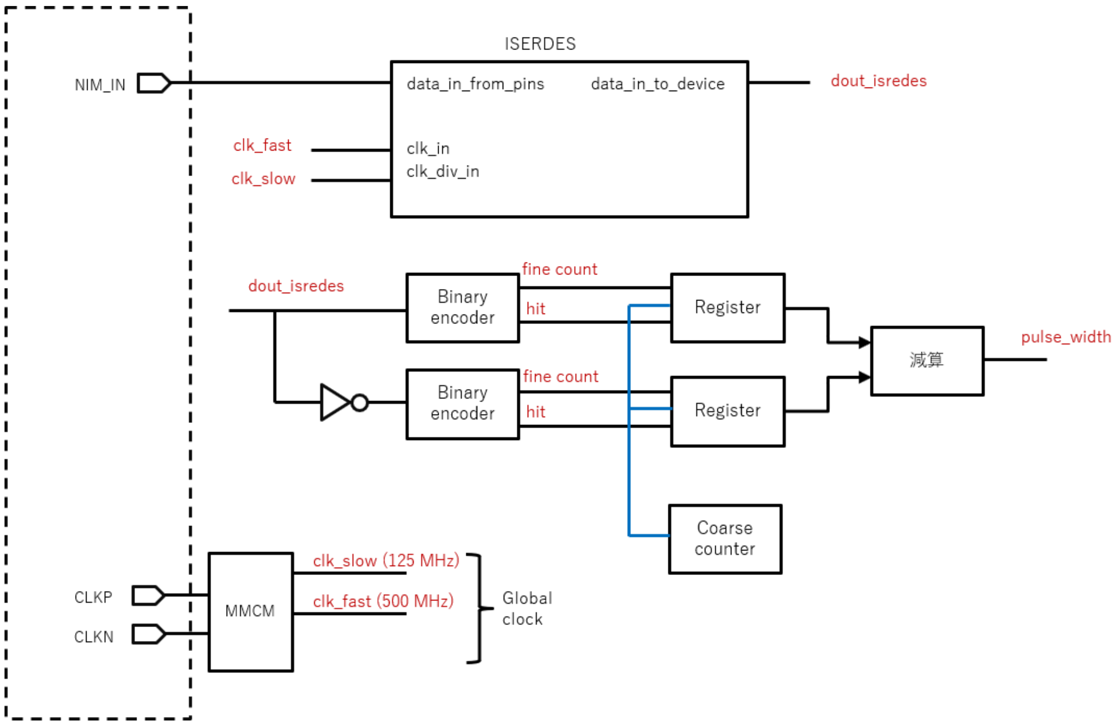


Fig. 14：H4's block diagram

Q1

Let' s implement a thermometer-to-binary encoder.

Consideration 1: Will the rising edge of the pulse (0→1) appear from Q0 of the ISERDES? Will it appear from Q7?

Consideration 2: The hit signal should be a one-shot pulse. How to detect the hit? Hint: The output of the ISERDES is always all 0s before the pulse input.

Q2

Let' s implement a coarse counter.

Consideration 1: The input signal width should be within 1 us. What should be the bit width?

Consideration 2: Suppose the TDC data is made up of a simple vector concatenation of a fine count and a coarse count. When generating TDC data based on this definition, should the coarse count be generated by count up or count down? Why?

Q3

Let's download the created TDC to AMANEQ and confirm that the pulse width can be measured by actually inputting a pulse. Please ask your lecturer for a function generator.