

PTP* Hands-On Lab-Book

- Part of workshop** on “HSR/PRP and PTP: Network Redundancy and Time Clock Synchronization” -

기안도

adki@future-ds.com

주최/주관: 한국통신학회 군통신연구회 / 명지대학교

장소: 숭실대학교 조만식기념관 427호

일자: 2019년6월7일

* PTP: Precision Time Protocol

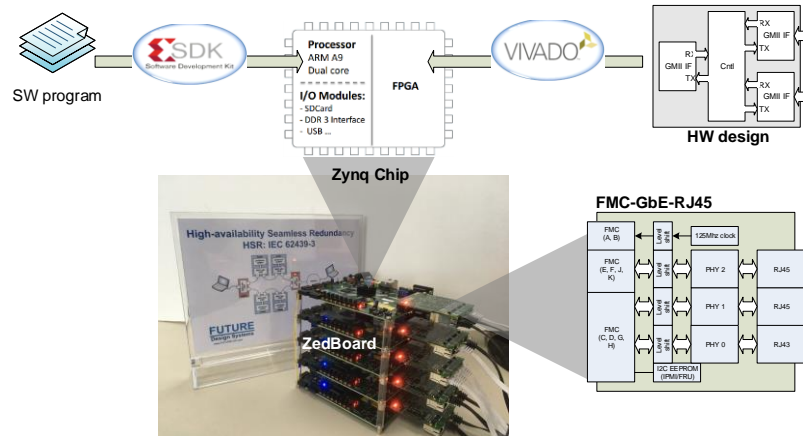
** 이중화네트워크와 시각동기화 워크샵

Table of Contents

- Development environment
- Gigabit Ethernet MAC
- Gigabit Ethernet PTP
- PTP with ARM bare metal

- Development environment
 - ▶ Overview
 - ▶ Xilinx tools for FPGA
 - ☞ Vivado
 - ☞ SDK
 - ▶ GTKwave
 - ▶ FPGA board: ZedBoard
 - ▶ Project codes

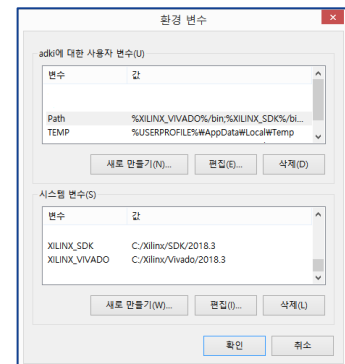
Development environment (1/3): overview



Vivado: FPGA development tool from Xilinx
 SDK: ARM software development tool from Xilinx
 ZedBoard: Zynq 7Z020 mounted FPGA board from Avnet
 FMC-GbE-RJ45: 3-port Gigabit board from Future Design Systems

Development environment (2/3): Xilinx tools

- If not ready, install Xilinx Vivado WebPack from <https://www.xilinx.com/products/design-tools/vivado/vivado-webpack.html>
 - ▶ This WebPack contains Vivado and SDK
- HW development tool: Xilinx Vivado 2018.3
 - ▶ C:/Xilinx/Vivado/2018.3
- SW development tool: Xilinx SDK 2018.3
 - ▶ C:/Xilinx/SDK/2018.3
- Environment variables
 - ▶ XILINX_VIVADO
 - ⇒ C:/Xilinx/Vivado/2018.3
 - ▶ XILINX_SDK
 - ⇒ C:/Xilinx/SDK/2018.3
 - ▶ Path
 - ⇒ %XILINX_VIVADO%/bin;%XILINX_SDK%/bin;...



Development environment (3/3): GTKwave

■ Ubuntu case

- ▶ Install gtkwave package
 - \$ sudo apt-get update
 - \$ sudo apt-get install gtkwave

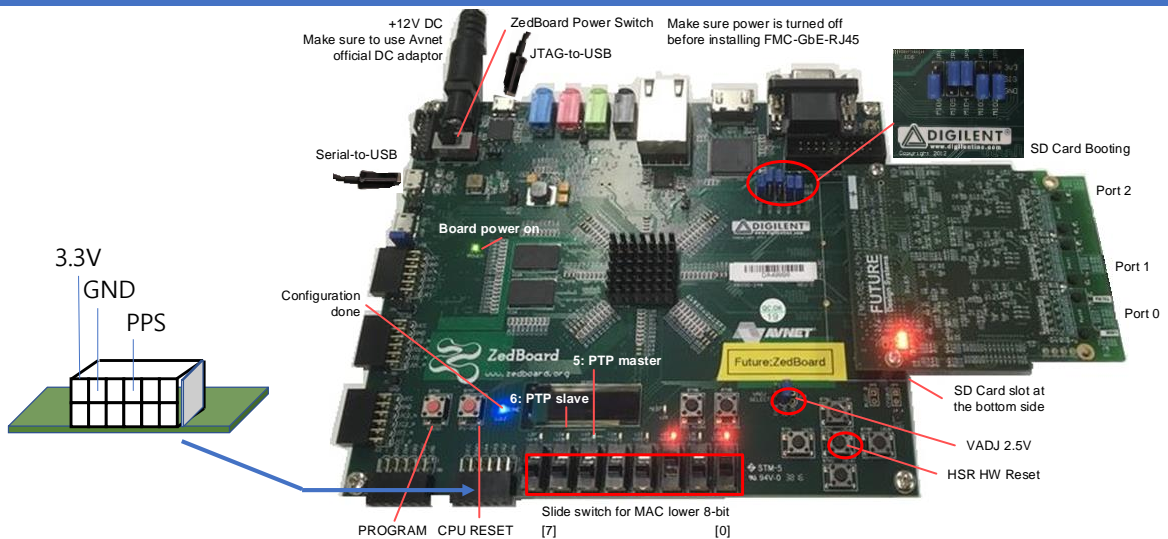


■ Windows case

- ▶ Get GTKwave from
 “<https://sourceforge.net/projects/gtkwave/files/>”
- ▶ Unzip somewhere in the directory
 - C:/gtkwave-3.3-100-bin-win32
- ▶ Set environment variables
 - GTKWAVE
 - C:/gtkwave-3.3-100-bin-win32/gtkwave/bin/gtkwave.exe

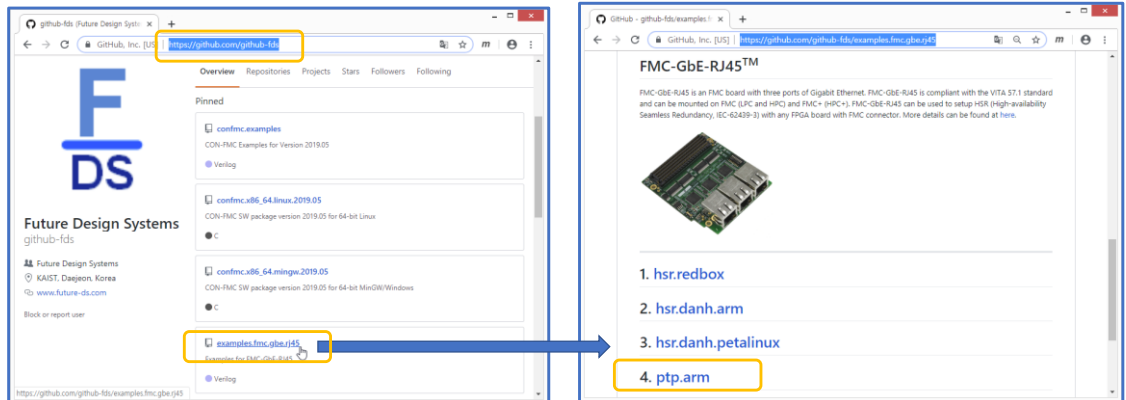


FPGA board: ZedBoard with FMC-GbE-RJ45



Project code (1/2)

- Get (i.e., clone) following code from GitHub
 - ▶ <https://github.com/github-fds/examples.fmc.gbe.rj45>



Project code (2/2)

- Get (i.e., clone) following code from GitHub
 - ▶ <https://github.com/github-fds/examples.fmc.gbe.rj45>

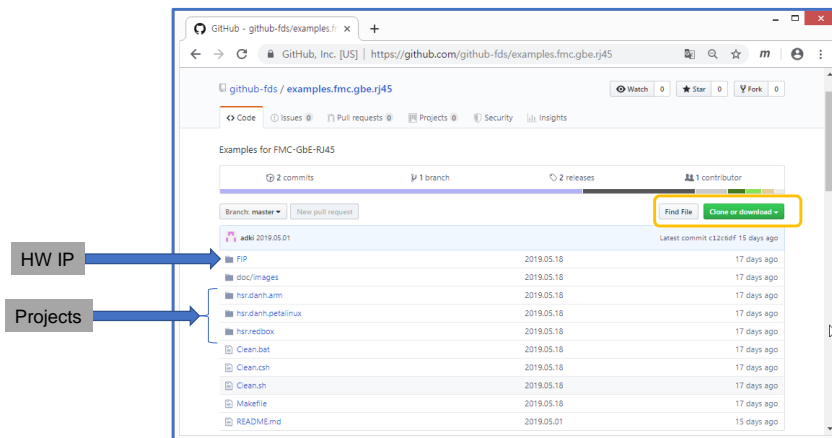
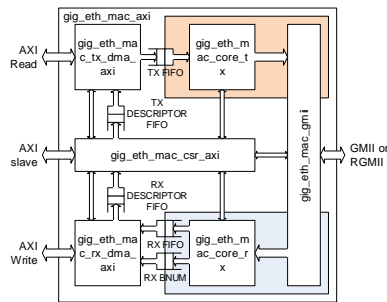


Table of Contents

- Development environment
- Gigabit Ethernet MAC
 - ▶ Block
 - ▶ Simulation
- Gigabit Ethernet PTP
- PTP with ARM bare metal
- Gigabit Ethernet MAC
 - ▶ Block
 - ▶ Simulation
- Gigabit Ethernet PTP
 - ▶ Block
 - ▶ Simulation

Gigabit Ethernet MAC: block

- Functional block diagram
 - ▶ CSR (AXI ACLK domain)
 - ▶ sending DMA (AXI ACLK domain)
 - ▶ receiving DMA (AXI ACLK domain)
 - ▶ GMII TX interface (gtxclk domain)
 - ▶ GMII RX interface (rxclk domain)
- Look at
 - ▶ \$(PROJECT)/FIP/gig_eth_mac/rtl/verilog
- API
 - ▶ \$(PROJECT)/FIP/gig_eth_mac/api/c



```
int gig_eth_mac_set_config(uint8_t  conf_tx_jumbo_en
                          , uint8_t  conf_tx_no_gen_crc
                          , uint16_t  conf_tx_bchunk
                          , uint8_t  conf_rx_jumbo_en
                          , uint8_t  conf_rx_no_chk_crc
                          , uint8_t  conf_rx_promiscuous
                          , uint16_t  conf_rx_bchunk);

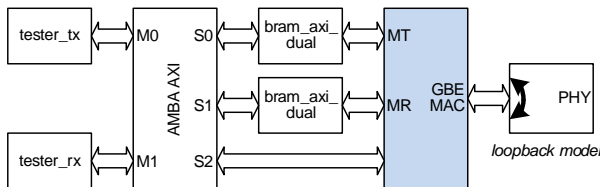
int gig_eth_mac_send(uint8_t  *addr
                    , uint16_t  blen
                    , int        time_out);

int gig_eth_mac_receive(uint8_t  *addr
                      , uint16_t  bnum
                      , int        time_out);
```

Gigabit Ethernet MAC: testing setup

■ Testing setup (block simulation)

- ▶ PHY: loopback model
- ▶ Two-buffer dual-port memories for sending and receiving packets
- ▶ AMBA AXI bus
 - ⇒ refer to 'https://github.com/adki/gen_amba'
- ▶ Two tester for generating and checking testing scenarios



■ Look at

- ▶ \$(PROJECT)/FIP/gig_eth_mac/bench/verilog

■ Testing tasks

```
gig_mac_send_packet(mac_dst // dst
                    ,mac_src // src
                    ,idx); // type-leng
gig_mac_receive_packet( slow );
```

Gigabit Ethernet MAC: simulation (1/2)

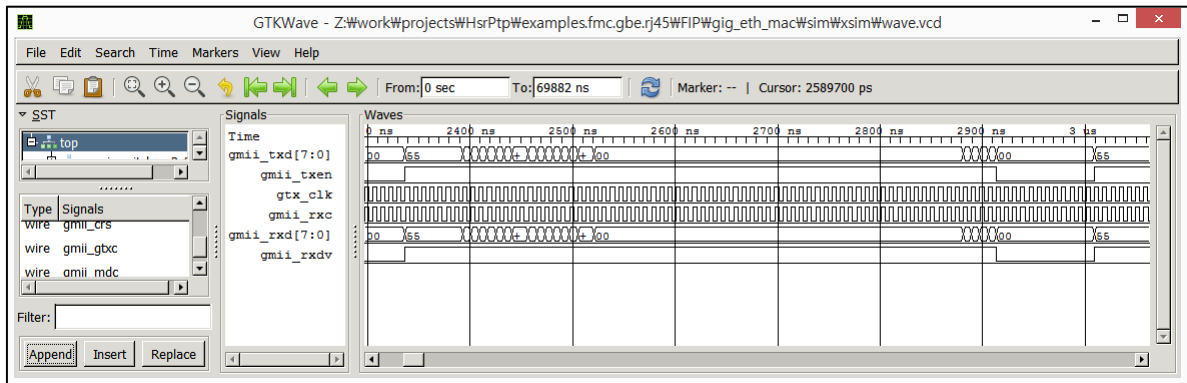
■ Linux case

- ▶ Step 1: go to your project directory
 - ⇒ (\$PROJECT stands for the project directory.)
 - ⇒ [user@host] `cd $(PROJECT)/FIP/gig_eth_mac/sim/xsim`
- ▶ Step 2: see the codes
- ▶ Step 3: run
 - ⇒ (It actually invokes Xilinx 'xelab' and 'xsim'.)
 - ⇒ [user@host] `make`
- ▶ Step 4: check the wave
 - ⇒ (It actually invokes GTKwave.)
 - ⇒ [user@host] `make wave`

■ Windows case

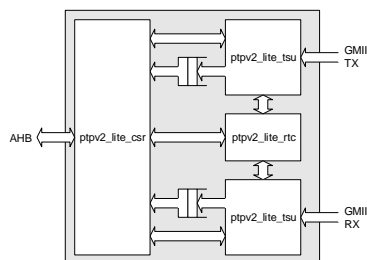
- ▶ Step 1: go to your project directory
 - ⇒ (%PROJECT% stands for the project directory.)
 - ⇒ > `cd %PROJECT%/FIP/gig_eth_mac/sim/xsim`
- ▶ Step 2: see the codes
- ▶ Step 3: compile (elaboration)
 - ⇒ > `RunMe.bat -elab`
- ▶ Step 4: simulation
 - ⇒ > `RunMe.bat -sim`
- ▶ Step 5: check the wave
 - ⇒ > `RunMe.bat -wave`

Gigabit Ethernet MAC: simulation (2/2)



Gigabit Ethernet PTP: block

- Functional block diagram
 - ▶ GMII port for timestamping
 - ▶ 48-bit sec, 32-bit nsec Real Time Clock
 - ▶ Configuration and status block with AMBA interface (AXI, AHB, or APB)
- Look at
 - ▶ \$(PROJECT)/FIP/gig_eth_ptpv2_lite/rtl/verilog
- API
 - ▶ \$(PROJECT)/FIP/gig_eth_ptpv2_lite/api/c



```
extern int ptpv2_lite_get_tod( uint16_t *sec_msb
                             , uint32_t *sec_lsb
                             , uint32_t *nano );
extern int ptpv2_lite_set_tod( uint16_t sec_msb
                             , uint32_t sec_lsb
                             , uint32_t nano);
extern int ptpv2_lite_adj_tod( uint8_t dec
                             , uint8_t sec
                             , uint32_t nano);
extern int ptpv2_lite_set_inc( uint8_t nano
                             , uint32_t nano_frac);
extern int ptpv2_lite_adj_inc( uint8_t dec
                             , uint8_t nano
                             , uint32_t nano_frac);
extern int ptpv2_lite_get_inc( uint8_t *nano
                             , uint32_t *nano_frac);
```

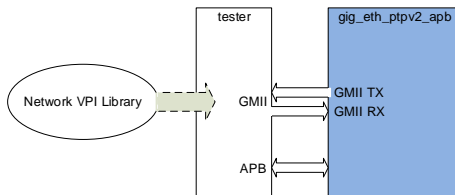
Gigabit Ethernet PTP: testing setup

■ Testing setup (block simulation)

- ▶ Network VPI library to generate UDP/IP/Ethernet packet
 - ⊕ Written in C
- ▶ Tester for generating and checking testing scenarios

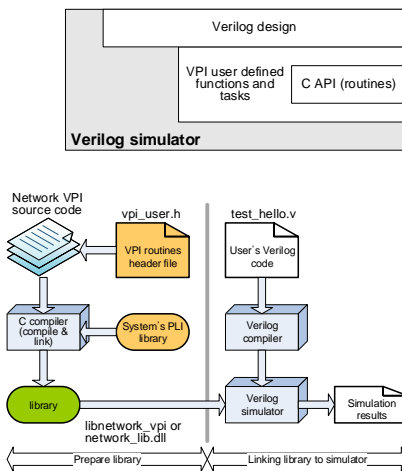
■ Look at

- ▶ \$(PROJECT)/FIP/gig_eth_ptpv2_lite/bench/v
erilog



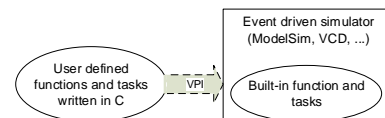
Network VPI library

VPI: Verilog Programming Interface



■ Examples of user defined tasks

- ▶ \$pkt_ethernet()
- ▶ \$pkt_ip()
- ▶ \$pkt_udp()
- ▶ \$msg_ptpv2_ethernet();
- ▶ \$msg_ptpv2_udp_ip_ethernet();



https://github.com/adki/network_vpi_lib/

Gigabit Ethernet PTP: simulation

■ Linux case

- ▶ Step 1: go to your project directory
 - ☞ (\$(PROJECT) stands for the project directory.)
 - ☞ [user@host] `cd $(PROJECT)/FIP/gig_eth_ptpv2_lite/sim/xsim`
- ▶ Step 2: see the codes
- ▶ Step 3: run
 - ☞ (It actually invokes Xilinx 'xelab' and 'xsim'.)
 - ☞ [user@host] `make`
- ▶ Step 4: check the wave
 - ☞ (It actually invokes GTKwave.)
 - ☞ [user@host] `make wave`

■ Windows case

- ▶ Step 1: go to your project directory
 - ☞ (%PROJECT% stands for the project directory.)
 - ☞ > `cd %PROJECT%/FIP/gig_eth_ptpv2_lite/sim/xsim`
- ▶ Step 2: see the codes
- ▶ Step 3: compile (elaboration)
 - ☞ > `RunMe.bat -elab`
- ▶ Step 4: simulation
 - ☞ > `RunMe.bat -sim`
- ▶ Step 5: check the wave
 - ☞ > `RunMe.bat -wave`

*Not covered in this hands-on practice
since Vivado XSIM does not support PLI/VPI.*

Table of Contents

- Development environment
- Gigabit Ethernet MAC
- Gigabit Ethernet PTP
- PTP with ARM bare metal

- PTP with ARM bare metal
 - ▶ Overall setup
 - ▶ Functional block diagram
 - ▶ Directory structure
 - ▶ Testing setup for simulation
 - ▶ Simulation
 - ▶ Overall steps
 - ▶ Preparing HW: implementing
 - ▶ Preparing SW
 - ▶ Download bit-stream and ELF: JTAG case
 - ▶ Download bit-stream and ELF: SD Card case
 - ▶ FDS monitor
 - ▶ PTP set up
 - ▶ Running

Directory structure

directory		remarks
hw	Hardware building projects	
	design	'mac_ptp_axi' building verilog
	bench	Test-bench verilog
	sim	simulation modelsim.vivado
	syn	Preparing 'mac_ptp_axi.edn' vivado.zedboard.lpc
	gen_ip	This project requires 'syn/vivado.zedboard.lpc/mac_ptp_axi.edn' and prepares 'mac_ptp_axi.xpr' zedboard.lpc
	impl	This project requires 'gen_ip/zedboard.lpc/mac_ptp_axi.xpr' and prepares 'zed_board_wrapper.bit' and 'zed_bd_wrapper_sysdef.hdf' zedboard.lpc

directory		remarks
sw.arm	Software building projects	
	fsbl	This project prepares boot loader.
	eth_send_receive	This project compiles user application program to test the hardware by sending and receiving packets
	ptp_udp	This project runs PTP
bootgen	This project prepares SD Card image to boot the ZedBoard. It requires 'sw.arm/fsbl/fsbl_0.elf' and 'hw/impl/zedboard.lpc/zed_bd_wrapper.bit' and user program.	

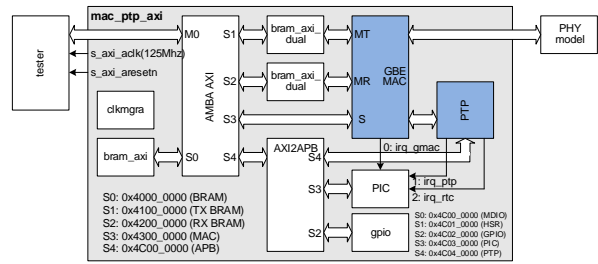
Directory structure

directory		remarks
hw	Hardware building projects	
	design	'hsr_danh_axi' building verilog
	bench	Test-bench verilog
	sim	simulation modelsim.vivado
	syn	Preparing 'hsr_danh_axi.edn' vivado.zedboard.lpc
	gen_ip	This project requires 'syn/vivado.zedboard.lpc/hsr_danh_axi.edn' and prepares 'hsr_danh_axi.xpr' zedboard.lpc
	impl	This project requires 'gen_ip/zedboard.lpc/hsr_danh_axi.xpr' and prepares 'zed_board_wrapper.bit' and 'zed_bd_wrapper_sysdef.hdf' zedboard.lpc

directory		remarks
sw.arm	Software building projects	
	fsbl	This project prepares boot loader.
	eth_send_receive	This project compiles user application program to test the hardware by sending and receiving packets
	mem_test	This project compiles a memory testing program.
bootgen	This project prepares SD Card image to boot the ZedBoard. It requires 'sw.arm/fsbl/fsbl_0.elf' and 'hw/impl/zedboard.lpc/zed_b_wrapper.bit' and user program.	

Testing setup for simulation

- The picture shows hardware structure to simulate a PTP system.



Simulation

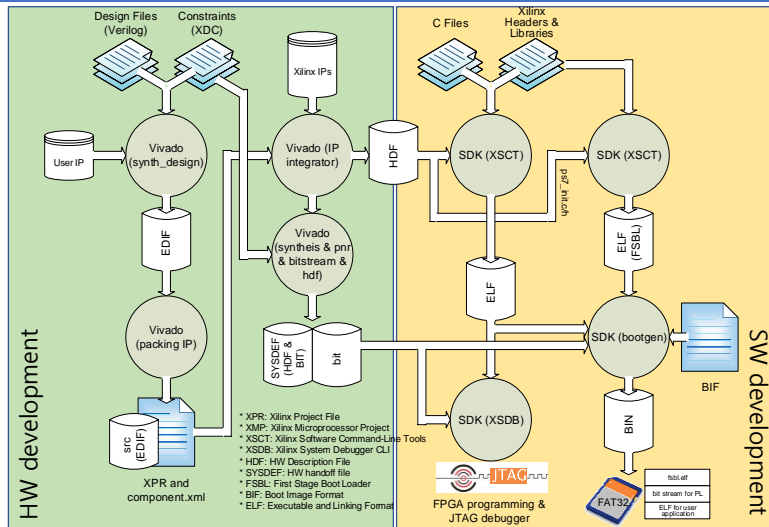
Linux case

- ▶ Step 1: go to your project directory
 - ⊗ `$(PROJECT)` stands for the project directory.)
 - ⊗ `[user@host] cd $(PROJECT)/ptp.arm/hw/sim/xsim`
- ▶ Step 2: see the codes
- ▶ Step 3: run
 - ⊗ (It actually invokes Xilinx 'xelab' and 'xsim'.)
 - ⊗ `[user@host] make`
- ▶ Step 4: check the wave
 - ⊗ (It actually invokes GTKwave.)
 - ⊗ `[user@host] make wave`

Windows case

- ▶ Step 1: go to your project directory
 - ⊗ `(%PROJECT%)` stands for the project directory.
 - ⊗ `> cd %PROJECT%/ptp.arm/hw/sim/xsim`
- ▶ Step 2: see the codes
- ▶ Step 3: compile (elaboration)
 - ⊗ `> RunMe.bat -elab`
- ▶ Step 4: simulation
 - ⊗ `> RunMe.bat -sim`
- ▶ Step 5: check the wave
 - ⊗ `> RunMe.bat -wave`

Overall steps



Preparing HW: implementing

- Note that these steps carries out synthesis and place & routing, so it takes time.
 - ▶ It uses Xilinx Vivado and requires 'XILINX_VIVADO' environment variable.
- ▶ Go to '\$(PROJECT)/hw/syn/vivado.zedboard.lpc'
- ▶ Run 'make' or 'RunMe.bat' depending on platform
- ▶ Go to '\$(PROJECT)/hw/gen_ip/zedboard.lpc'
- ▶ Run 'make' or 'RunMe.bat' depending on platform
- ▶ Go to '\$(PROJECT)/hw/impl/zedboard.lpc'
- ▶ Run 'make' or 'RunMe.bat' depending on platform
 - ➡ Followings should be ready.
 - 'zed_bd_wrapper.bit': bit-stream
 - 'zed_bd_wrapper_sysdef.hdf': HW description file

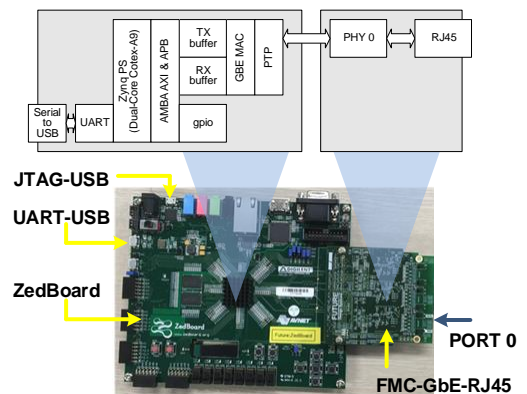
Note Vivado WebPack complains about part, but it is listed using 'get_parts' TCL command.
 "No parts matched 'xc7z020clg484-1'

Preparing SW

- Note that these steps compiles ARM program
 - ▶ It uses Xilinx SDK and requires 'XILINX_SDK' environment variable.
 - ▶ Go to '\$(PROJECT)/sw/eth_send_receive'
 - ▶ Run 'make' or 'RunMe.bat' depending on platform
 - ⇒ Linux: \$ make
 - ⇒ Windows: > RunMe.bat -compile
 - ⇒ Followings should be ready.
 - 'eth_send_receive.elf'

Download bit-stream and ELF: JTAG case

- Note that these steps downloads HW bit-stream and SW ELF image
 - ▶ It uses Xilinx SDK and requires 'XILINX_SDK' environment variable.
 - ▶ Followings should be connected properly.
 - ⇒ 12V DC
 - ⇒ JTAG-USB
 - ⇒ UART-USB
 - ▶ Go to '\$(PROJECT)/sw/eth_send_receive'
 - ▶ Turn on power
 - ▶ Invoke text-terminal emulator
 - ⇒ teraterm or hyperterminal
 - ⇒ 15200-baud, 8-bit, no-parity, 1-bit stop, no flow control
 - ▶ Run 'make' or 'RunMe.bat'
 - ⇒ Linux: \$ make
 - ⇒ Windows: > RunMe.bat -download

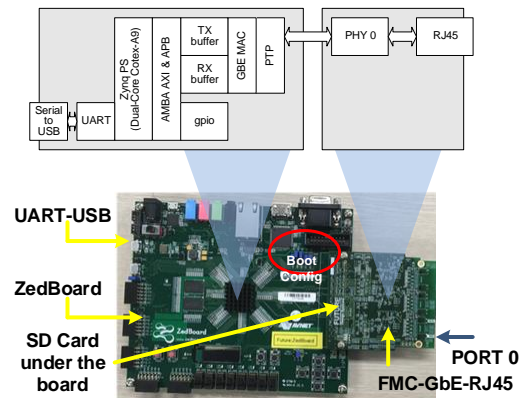


ELF: executable and lining format

Download bit-stream and ELF: SD Card case

■ Note that these steps prepare SD CARD for HW bit-stream and SW ELF image

- ▶ It uses Xilinx SDK and requires 'XILINX_SDK' environment variable.
- ▶ Followings should be connected properly.
 - ➡ 12V DC
 - ➡ JTAG-USB
 - ➡ UART-USB
 - ➡ Boot configuration jump for SD Card
- ▶ Go to '\$(PROJECT)/sw/fsbl' and run 'make'
- ▶ Go to '\$(PROJECT)/bootgen' and run 'make'
- ▶ Copy 'BOOT.bin' to SD Card
- ▶ Insert SD Card to ZedBoard and turn on power
- ▶ Invoke text-terminal emulator
 - ➡ teraterm or hyperterminal
 - ➡ 115200-baud, 8-bit, no-parity, 1-bit stop, no flow control



ELF: executable and lining format

FDS monitor

```

COM6 - Tera Term VT
File Edit Setup Control Window Help

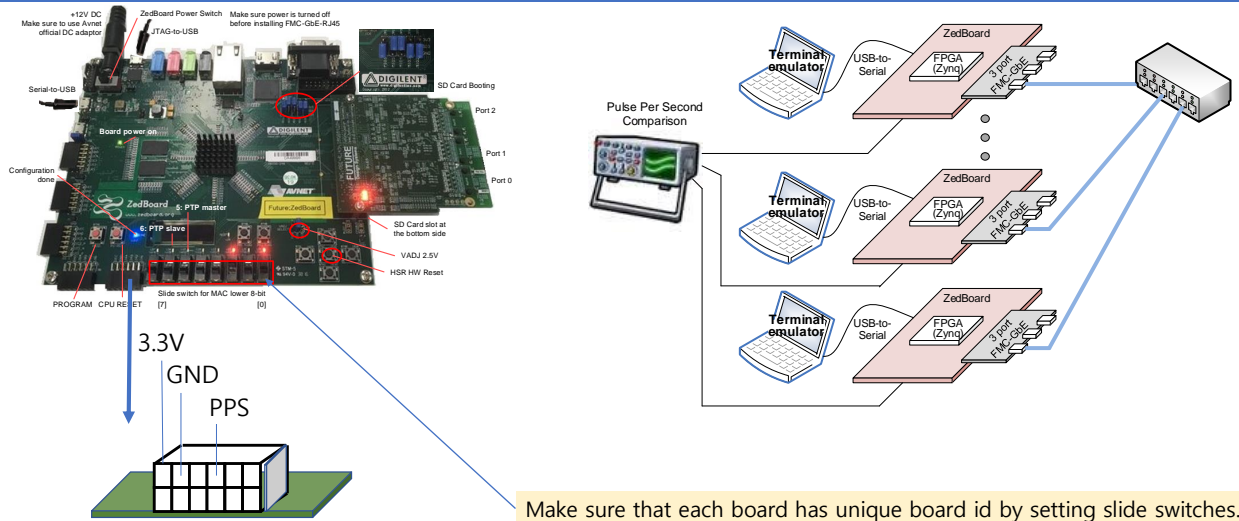
FdsMON - 2019.03.02.
Copyright (c) 2018-2019 by Future Design Systems
www.future-ds.com
This software is provided 'as-is', without any express or implied warranty. In
no event will the authors be held liable for any damages arising from the use of
this software.
Developed by Ando Ki (adki@future-ds.com)
monitor>
  
```

■ Use 'help' to see help message

■ General steps of commands

- ▶ Monitor> mac_init
- ▶ Monitor> mac_addr -r
- ▶ MAC 0x021234567801
- ▶ HSR 0x021234567801
- ▶ pkt_send -b 0x021234567802 -r
- ▶ pkt_rcv -r -v 3

PTP set up



Running

- Following sequence demonstrates how 'Terminal A' sends a packet to 'Terminal B', in which 'Terminal A' and 'Terminal B' have MAC address 0x021234567801 and 0x021234567802, respectively.

Terminal A	Terminal B	Remarks
monitor> <i>mac_init</i>	monitor> <i>mac_init</i>	initialize
monitor> <i>mac_addr -r</i> MAC 0x021234567801 HSR 0x021234567801	monitor> <i>mac_addr -r</i> MAC 0x021234567802 HSR 0x021234567802	check MAC and HSR addresses. These two should be the same. Actual values depends on board ID.
	monitor> <i>pkt_rcv -v 3 -r</i>	Receive packets
monitor> <i>pkt_snd -b 0x021234567802 -r</i>		Send packets
		Terminal B prints packet information continuously.

㈜퓨처디자인시스템

34051 대전광역시 유성구 문지로 193, KAIST 문지캠퍼스, F723호
(042) 864-0211~0212 / contact@future-ds.com / www.future-ds.com

Future Design Systems, Inc.

Faculty Wing F723, KAIST Munji Campus, 193 Munji-ro, Yuseong-gu, Daejeon 34051, Korea
+82-042-864-0211~0212 / contact@future-ds.com / www.future-ds.com



FUTURE
Design Systems