

# HSR\* Hands-On Lab-Book

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

기안도

[adki@future-ds.com](mailto:adki@future-ds.com)

주최/주관: 한국통신학회 군통신연구회 / 명지대학교

장소: 숭실대학교 조만식기념관 427호

일자: 2019년6월7일

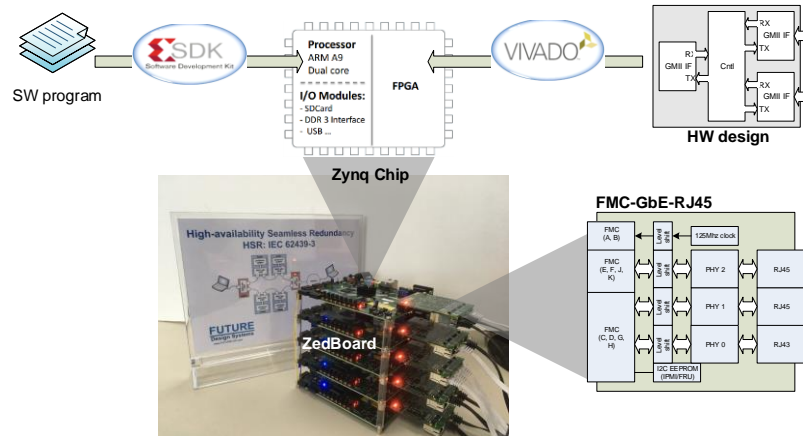
\* HSR: High-availability Seamless Redundancy

\*\* 이중화네트워크와 시각동기화 워크샵

## Table of Contents

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>■ Development environment</li> <li>■ Gigabit Ethernet MAC</li> <li>■ Gigabit Ethernet HSR</li> <li>■ DANH with ARM bare metal</li> <li>■ DANH with ARM PetaLinux</li> <li>■ RedBox with Raspberry Pi</li> </ul> | <ul style="list-style-type: none"> <li>■ Development environment               <ul style="list-style-type: none"> <li>▶ Overview</li> <li>▶ Xilinx tools for FPGA                   <ul style="list-style-type: none"> <li>☞ Vivado</li> <li>☞ SDK</li> </ul> </li> <li>▶ GTKwave</li> <li>▶ FPGA board: ZedBoard</li> <li>▶ Project codes</li> </ul> </li> </ul> |
|--|---|

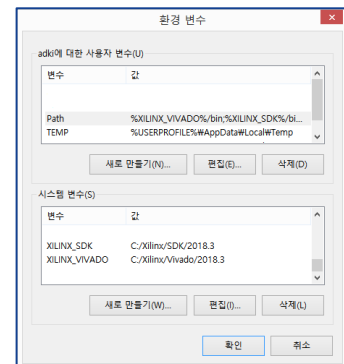
## Development environment (1/3): overview



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

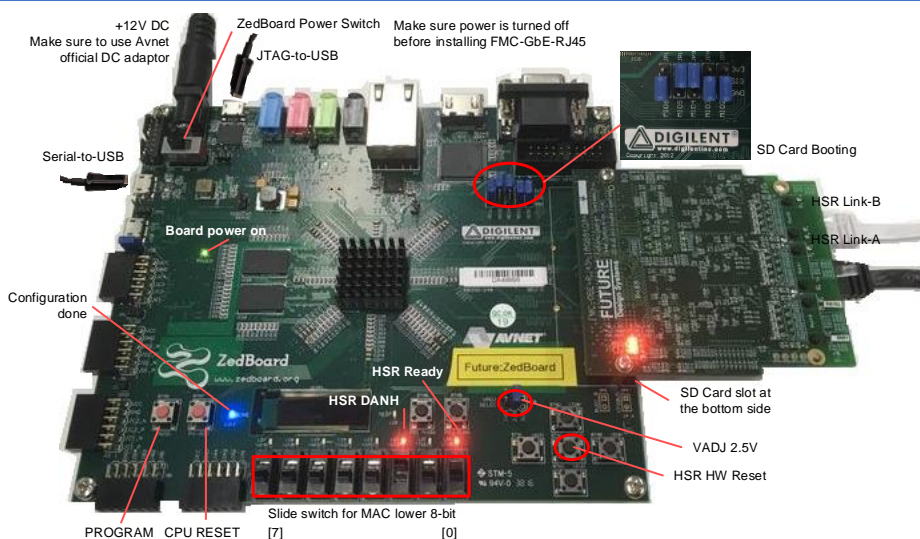


### ■ Windows case

- ▶ 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

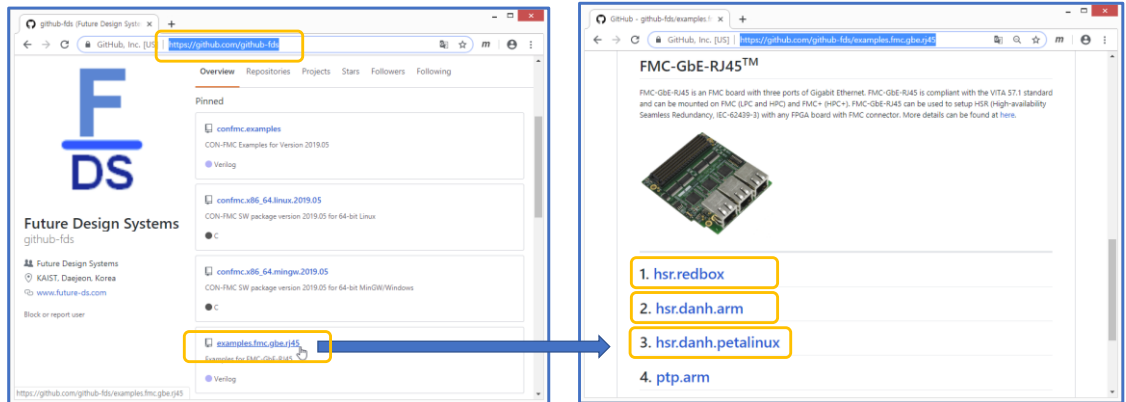


## FPGA board: ZedBoard with FMC-GbE-RJ45



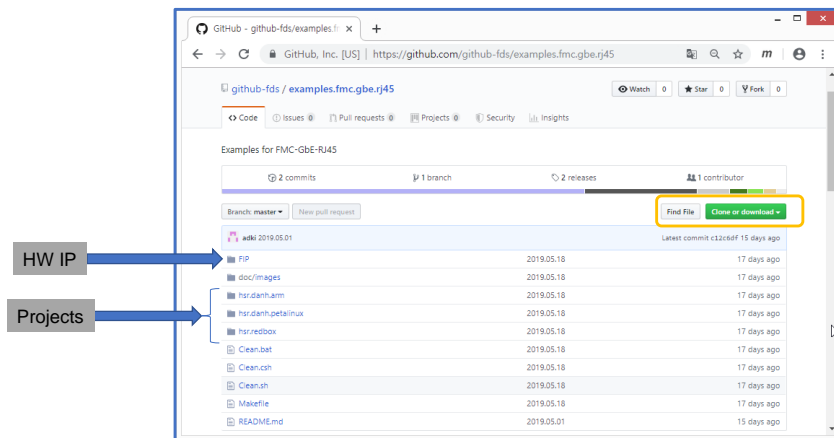
## Project code (1/2)

- 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>



- ## Gigabit Ethernet MAC: block

- ```
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)

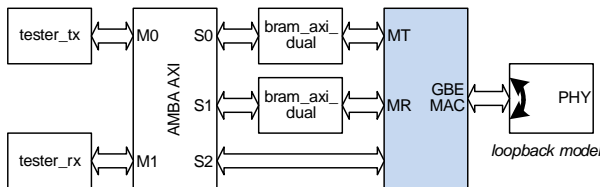
- ▶ PHY: loopback model
- ▶ Two-buffer dual-port memories for sending and receiving packets
- ▶ AMBA AXI bus
  - ☞ refer to '[https://github.com/adki/gen\\_amba](https://github.com/adki/gen_amba)'
- ▶ Two tester for generating and checking testing scenarios

### ■ 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)

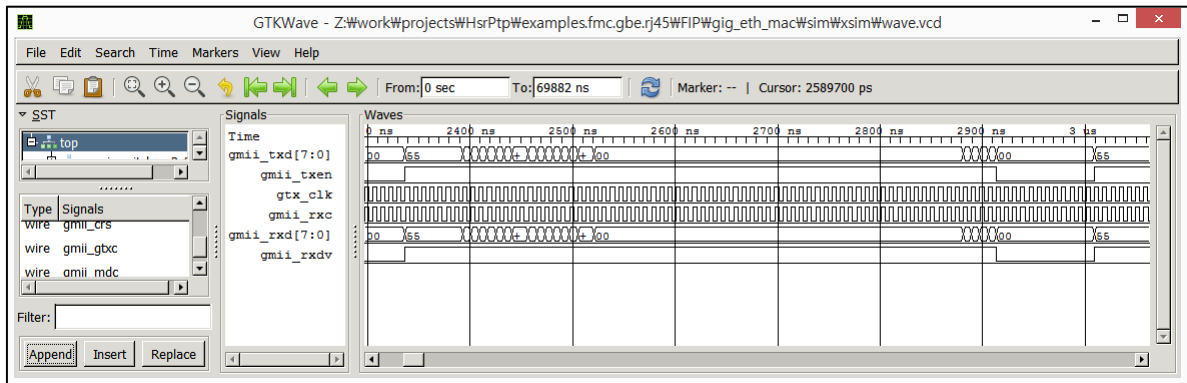
### ■ Linux case

- ▶ 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`

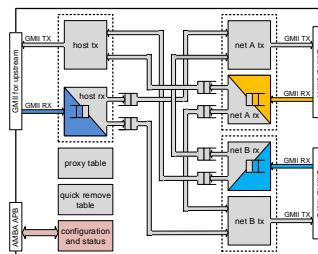
## Gigabit Ethernet MAC: simulation (2/2)



## Gigabit Ethernet HSR: block

### Functional block diagram

- ▶ GMII port for upstream
- ▶ GMII port for HSR link-A & B
- ▶ Configuration and status block with AMBA APB interface
- ▶ Proxy table for RedBox
- ▶ Quick remove table for HSR links



### Look at

- ▶ \$(PROJECT)/FIP/gig\_eth\_hsr/rtl/verilog

### API

- ▶ \$(PROJECT)/FIP/gig\_eth\_hsr/api/c

```
extern int gig_eth_hsr_set_mac_addr( uint8_t mac[6] );
extern int gig_eth_hsr_get_mac_addr( uint8_t mac[6] );
extern int gig_eth_hsr_set_control( uint8_t snoop
                                   , uint8_t enable_qr
                                   , uint8_t drop_non_hsr
                                   , uint8_t promiscuous);
```

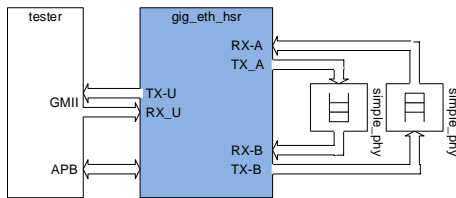
## Gigabit Ethernet HSR: testing setup

### ■ Testing setup (block simulation)

- ▶ PHY: loopback model
- ▶ Tester for generating and checking testing scenarios

### ■ Look at

- ▶ `$(PROJECT)/FIP/gig_eth_hsr/bench/verilog`



## Gigabit Ethernet HSR: simulation (1/2)

### ■ Linux case

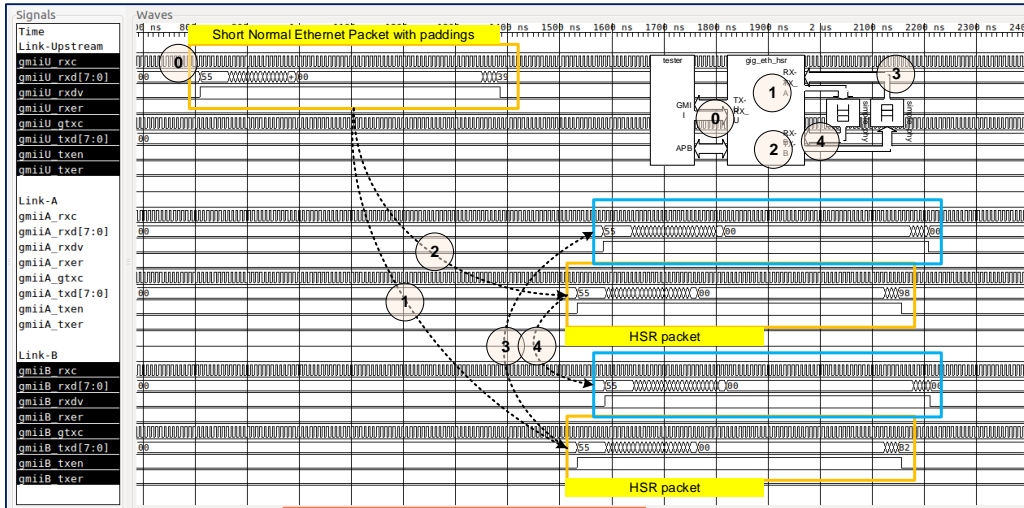
- ▶ Step 1: go to your project directory
  - ☞ `$(PROJECT)` stands for the project directory.)
  - ☞ `[user@host] cd $(PROJECT)/FIP/gig_eth_hsr/sim/xsim`
- ▶ Step 2: see the codes
- ▶ Step 3: run
  - ☞ (It actually invokes Xilinx 'xelab' and 'xsim'.)
  - ☞ `[user@host] make`
- ▶ Step 4: check the wave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

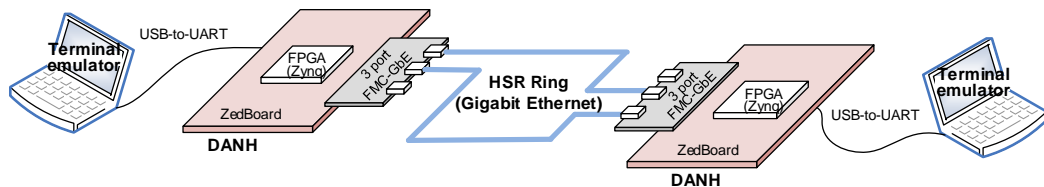


## Table of Contents

- Development environment
- Gigabit Ethernet MAC
- Gigabit Ethernet HSR
- DANH with ARM bare metal
- DANH with ARM PetaLinux
- RedBox with Raspberry Pi
- DANH with ARM bare metal
  - ▶ Overall setup
  - ▶ Functional block diagram
  - ▶ Directory structure
  - ▶ Testing setup for simulation and Simulation
  - ▶ Overall steps
  - ▶ Preparing HW and SW
  - ▶ Download bit-stream and ELF: JTAG case
  - ▶ Download bit-stream and ELF: SD Card case
  - ▶ FDS monitor
  - ▶ Two DANH's
  - ▶ Single and multiple packets

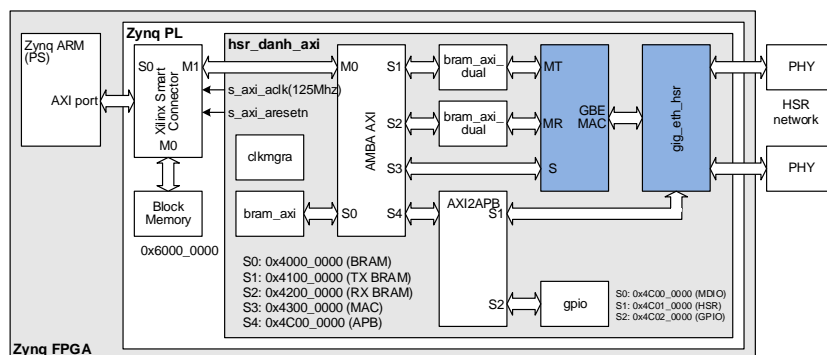
## Overall setup

- This example uses two DANHs (Double Attached Node implementing High-availability Seamless Redundancy) as shown in the picture below, where PS (Processing System) in the FPGA chip, i.e., ARM runs program without operating system.



## Functional block diagram

- PS (Zynq ARM) and Zynq PL
- Zynq PL contains a DANH sub-system comprising of MAC, HSR, AXI and so on.



AMBA AXI: [https://github.com/adki/gen\\_amba/](https://github.com/adki/gen_amba/)

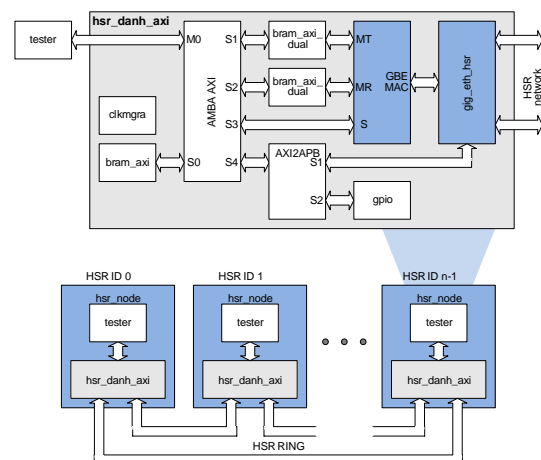
## Directory structure

| directory | remarks                                                                                                                                           |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>hw</b> | Hardware building projects                                                                                                                        |
| design    | 'hsr_danh_axi' building<br>verilog                                                                                                                |
| bench     | Test-bench<br>verilog                                                                                                                             |
| sim       | simulation<br>modelsim.vivado                                                                                                                     |
| syn       | Preparing 'hsr_danh_axi.edn'<br>vivado.zedboard.lpc                                                                                               |
| gen_ip    | This project requires 'syn/vivado.zedboard.lpc/hsr_danh_axi.edn' and prepares 'hsr_danh_axi.xpr'<br>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'<br>zedboard.lpc |

| directory        | remarks                                                                                                                                                           |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>sw.arm</b>    | 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.                                                                                                                   |
| <b>bootgen</b>   | This project prepares SD Card image to boot the ZeDBoard.<br>It requires 'sw.arm/fsbl/fsbl_0.elf' and 'hw/impl/zedboard.lpc/zed_bd_wrapper.bit' and user program. |

## Testing setup for simulation

- The picture shows hardware structure to simulate a HSR system, where several HSR nodes are connected to build ring and each HSR node consists of 'hsr\_danh\_axi' and 'tester'.



## Simulation (1/2)

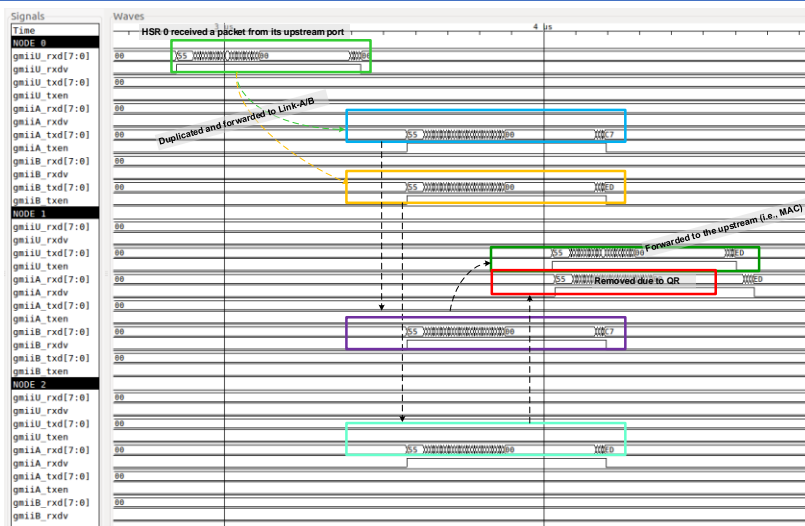
### Linux case

- ▶ Step 1: go to your project directory
  - ⊗ `$(PROJECT)` stands for the project directory.)
  - ⊗ `[user@host] cd $(PROJECT)/hsr.danh.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`

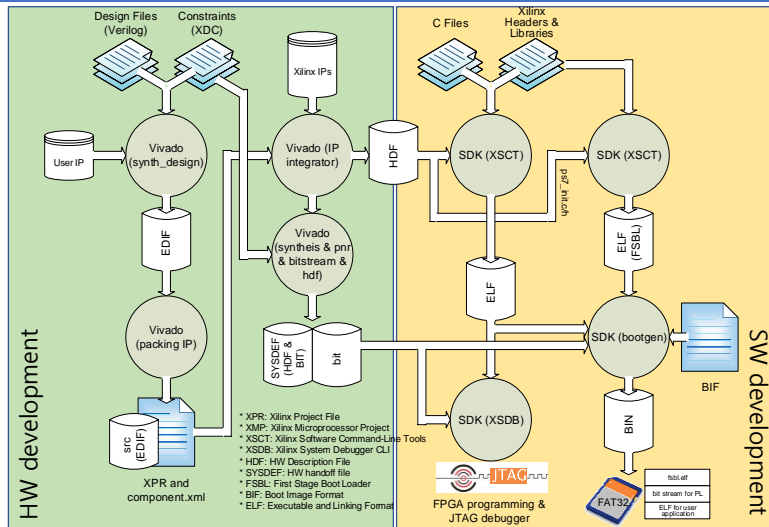
### Windows case

- ▶ Step 1: go to your project directory
  - ⊗ `(%PROJECT%)` stands for the project directory.
  - ⊗ `> cd %PROJECT%/hsr.danh.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`

## Simulation (2/2)



## 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

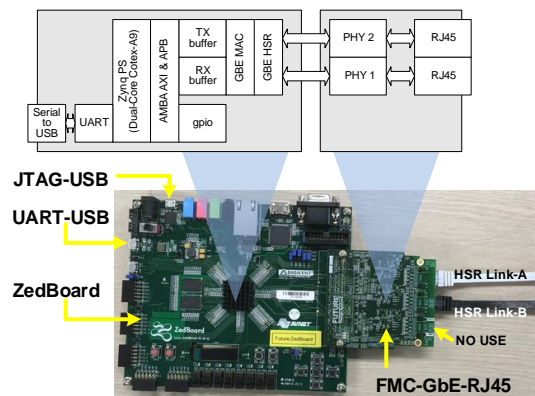
Note Vivado WebPack complains about part, but it is listed using 'get\_parts' TCL command.  
 "No parts matched 'xc7z020clg484-1'

## Preparing SW: preparing

- 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

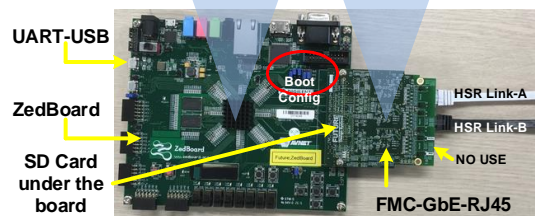
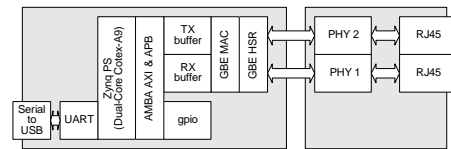


ELF: executable and lining format

## 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

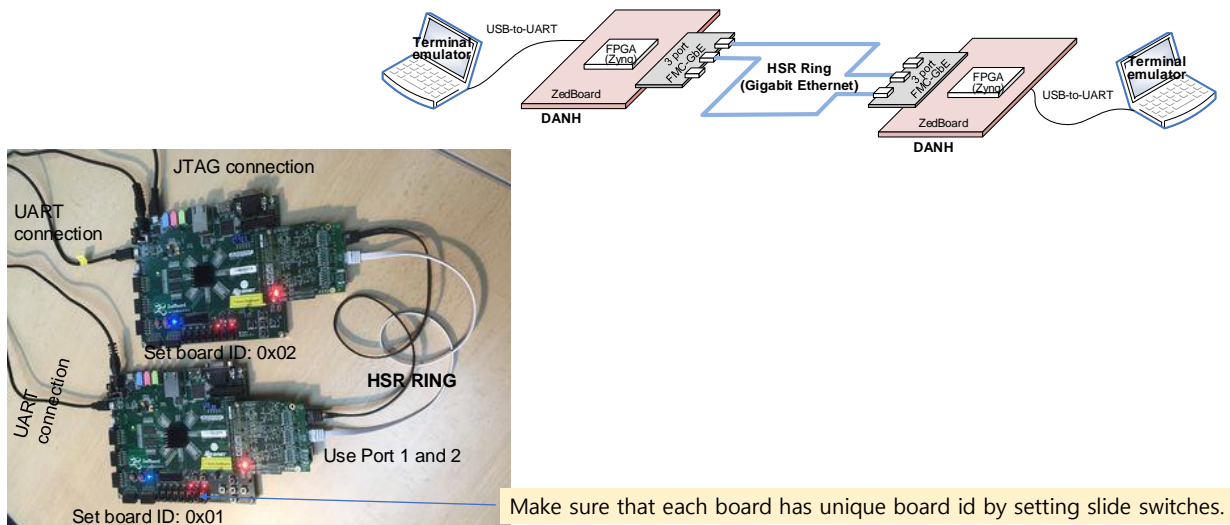
FdsMON - 2019.03.02.
Copyright (c) 2018-2019 by Future Design Systems
www.future-ds.com
This software is provided 'as-is', without any express or implied warranty. In
no event will the authors be held liable for any damages arising from the use of
this software.
Developed by Ando Ki (adki@future-ds.com)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

## Two DANH's



## Single packet case

- 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 (HSR Node 0)                                                 | Terminal B (HSR Node 1)                                                 | Remarks                                                                                       |
|-------------------------------------------------------------------------|-------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| monitor> <i>mac_init</i>                                                | monitor> <i>mac_init</i>                                                | initialize                                                                                    |
| monitor> <i>mac_addr -r</i><br>MAC 0x021234567801<br>HSR 0x021234567801 | monitor> <i>mac_addr -r</i><br>MAC 0x021234567802<br>HSR 0x021234567802 | check MAC and HSR addresses. These two should be the same. Actual values depends on board ID. |
| monitor> <i>pkt_snd -b 0x021234567802</i>                               |                                                                         | Send a packet to the other                                                                    |
|                                                                         | monitor> <i>pkt_rcv -v 3</i><br>ETH mac dst:                            | Receive a packet                                                                              |



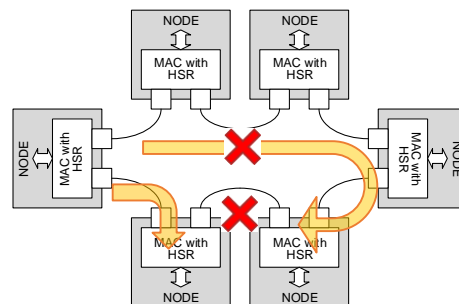
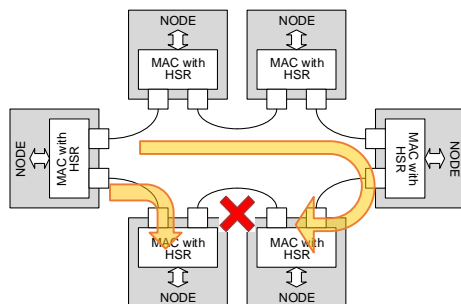
## Multiple-packet case

- 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><br>MAC 0x021234567801<br>HSR 0x021234567801 | monitor> <i>mac_addr -r</i><br>MAC 0x021234567802<br>HSR 0x021234567802 | check MAC and HSR addresses. These two should be the same. Actual values depends on board I D. |
|                                                                         | 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.                                             |

## HSR testing

- Disconnect one of two HSR links
- Disconnect both links

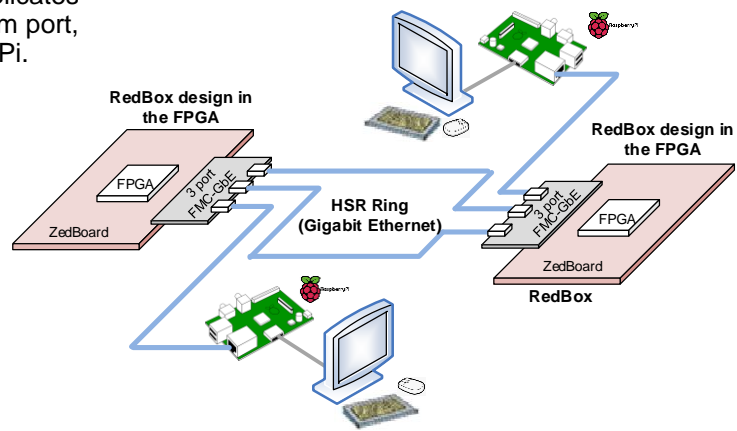


## Table of Contents

- Development environment
- Gigabit Ethernet MAC
- Gigabit Ethernet HSR
- DANH with ARM bare metal
- DANH with ARM PetaLinux
- RedBox with Raspberry Pi

## Overall setup

- This example uses two RedBoxes (Redundancy Box) as shown in the picture below, in which each RedBox duplicates packets received from its upstream port, which is connected to Raspberry Pi.

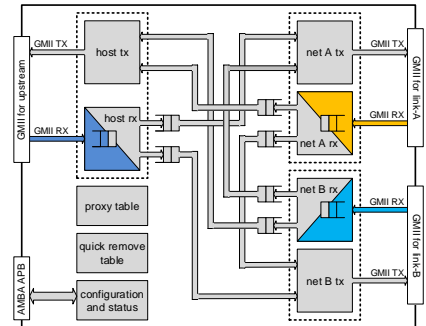
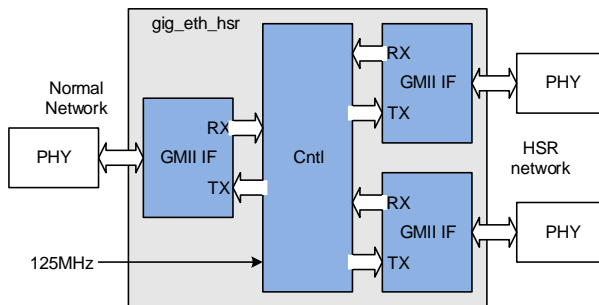


## Functional block diagram

- Zynq PL contains a RedBox sub-system, while PS does nothing to do with this example.

- The HSR block supports 3 Ethernet ports and contains proxy table and quick remover.

- The proxy table maintains MAC addresses from up-stream port.
- The quick-remove table maintains source MAC addresses and its HSR sequence numbers.

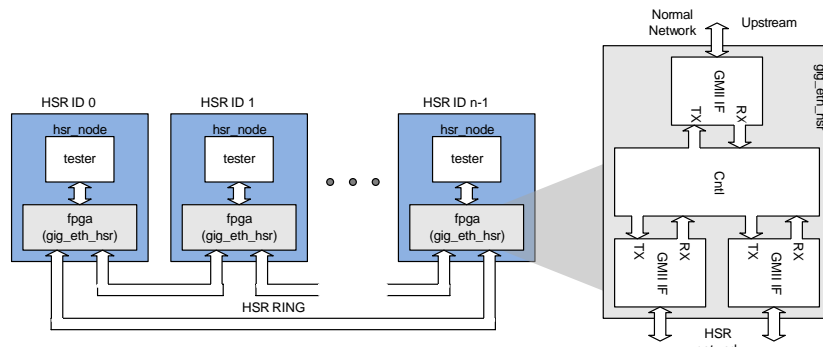


## Directory structure

| directory       | remarks                       |
|-----------------|-------------------------------|
| rtl             | RTL design directory          |
| verilog         | <code>gig_eth_hsr.v</code>    |
| fifo_async      | Asynchronous FIFO project     |
| v6              | ise14                         |
| z7              | vivado.2018.3                 |
| fifo_sync       | Synchronous FIFO project      |
| v6              | ise14                         |
| z7              | vivado.2018.3                 |
| bench           | Test-bench directory          |
| verilog         | <code>top.v</code>            |
| sim             | RTL simulation directory      |
| modelsim.ise    | Simulation for ISE devices    |
| modelsim.vivado | Simulation for Vivado devices |

## Testing setup for simulation

- The picture shows hardware structure to simulate a RedBox system, in which 'tester' plays a role of processor to generate packets.



## Simulation (1/3)

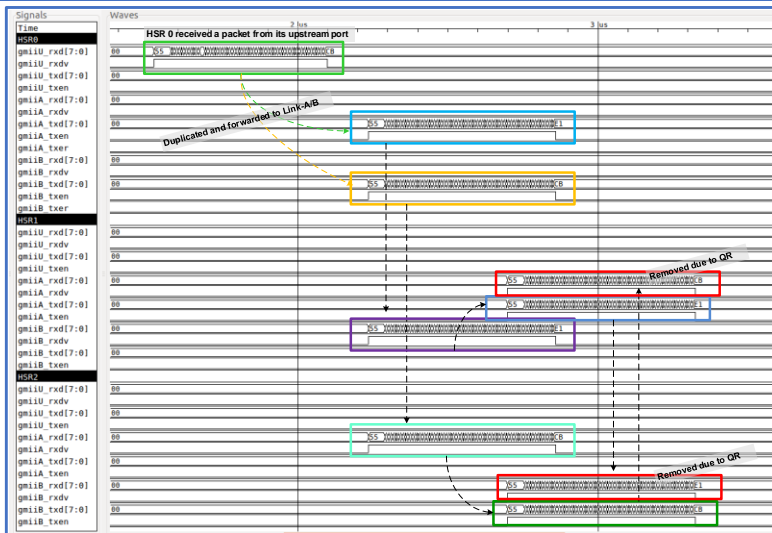
### Linux case

- ▶ Step 1: go to your project directory
  - ☞ `$(PROJECT)` stands for the project directory.)
  - ☞ `[user@host] cd $(PROJECT)/hsr.redbox/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`

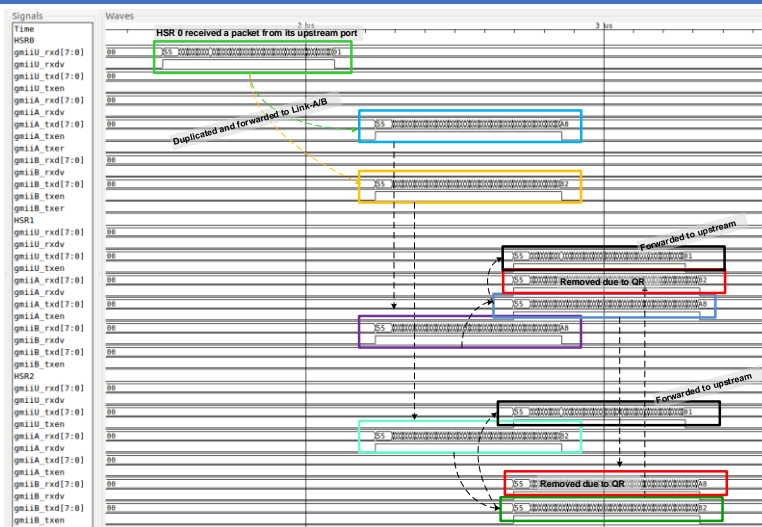
### Windows case

- ▶ Step 1: go to your project directory
  - ☞ `(%PROJECT%)` stands for the project directory.
  - ☞ `> cd %PROJECT%/hsr.redbox/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`

## Simulation (2/3)



## Simulation (3/3)



## 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)/pnr/vivado.zedboard.lpc'
  - ▶ Run 'make' or 'RunMe.bat' depending on platform
    - Followings should be ready.
      - 'fpga.bit': bit-stream

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

## Download bit-stream: JTAG case

- Note that these steps downloads HW bit-stream
  - ▶ It uses Xilinx SDK and requires 'XILINX\_SDK' environment variable.
  - ▶ Followings should be connected properly.
    - 12V DC
    - JTAG-USB
    - Raspberry Pi
  - ▶ Go to '\$(PROJECT)/pnr/vivado.zedboard.lpc/download'
  - ▶ Turn on power
  - ▶ Run 'make' or 'RunMe.bat'
    - Linux: \$ make
    - Windows: > RunMe.bat -download

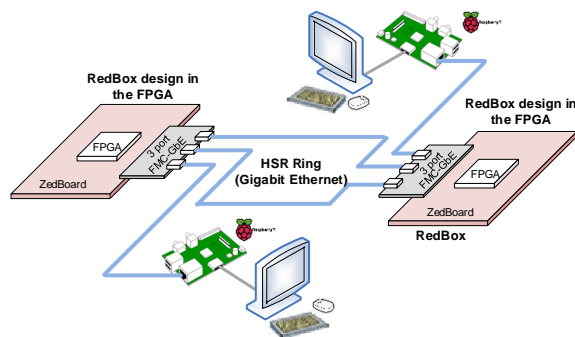


## Download bit-stream: SD Card case

- Note that these steps prepare HW bit-stream
  - ▶ It uses Xilinx SDK and requires 'XILINX\_SDK' environment variable.
  - ▶ Followings should be connected properly.
    - 12V DC
    - JTAG-USB
    - Boot configuration jump for SD Card
    - Raspberry Pi
  - ▶ Go to '\$(PROJECT)/pnr/vivado.zedboard.lpc/bootgen'
  - ▶ Run 'make' or 'RunMe.bat'
  - ▶ Copy 'BOOT.bin' to SD Card
  - ▶ Insert SD Card to ZedBoard and turn on power



## Two RedBoxes with Raspberry Pi



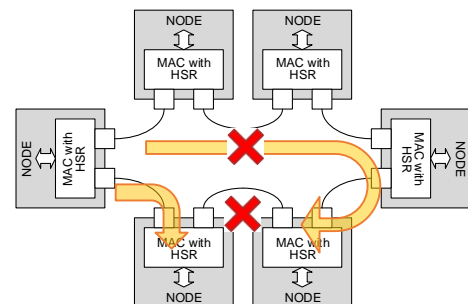
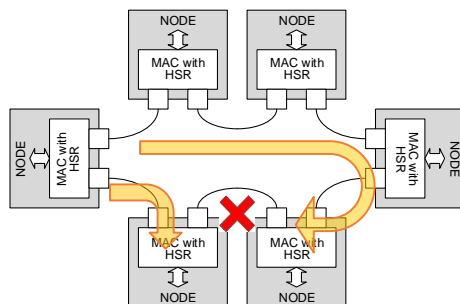
## Ping case

- One Raspberry Pi runs 'ping' command to check the other Raspberry Pi over Ethernet.

| Terminal A (Raspberry Pi)                                               | Terminal B (Raspberry Pi)                                               | Remarks                                         |
|-------------------------------------------------------------------------|-------------------------------------------------------------------------|-------------------------------------------------|
| Open a terminal                                                         | Open a terminal                                                         | Prepare two terminals                           |
| <code>\$ ifconfig</code><br>eth0: ... inet 192.168.1.152                | <code>\$ ifconfig</code><br>eth0: ... inet 192.168.1.153                | Check IP address<br><code>\$ hostname -I</code> |
| <code>\$ ping 192.168.1.153</code>                                      | <code>\$ ping 192.168.1.152</code>                                      | Each runs ping command.                         |
| 64 bytes from 192.168.1.152: icmp<br>_seq=1 ttl=64 time=0.594 ms<br>... | 64 bytes from 192.168.1.152: icmp_se<br>q=1 ttl=64 time=0.594 ms<br>... |                                                 |

## HSR testing

- Disconnect one of two HSR links
- Disconnect both links



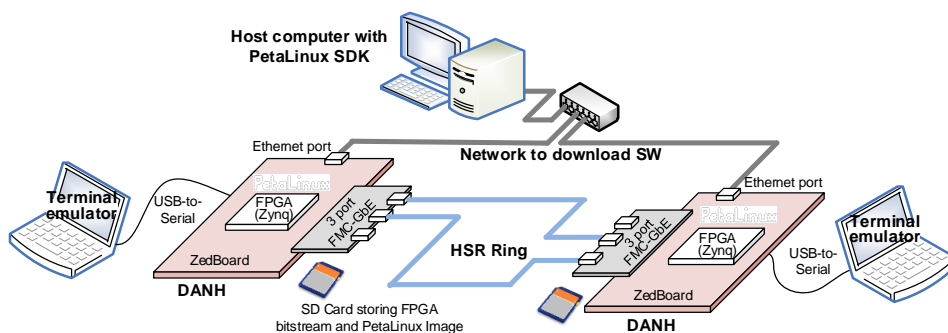


## Table of Contents

- Development environment
- Gigabit Ethernet MAC
- Gigabit Ethernet HSR
- DANH with ARM bare metal
- DANH with ARM PetaLinux
- RedBox with Raspberry Pi
- DANH with ARM PetaLinux

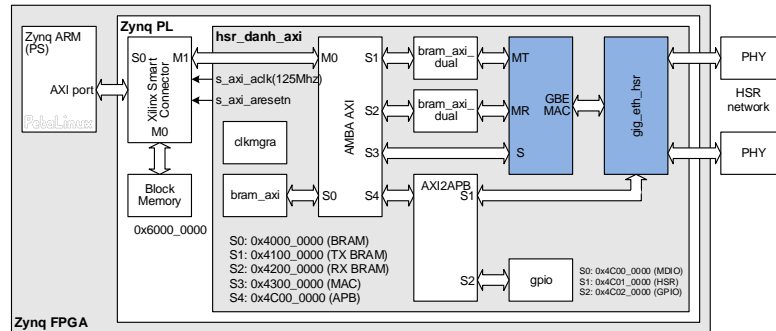
## Overall setup

- This example uses two DANHs (Double Attached Node implementing High-availability Seamless Redundancy) as shown in the picture below, where PS (Processing System) in the FPGA, i.e., ARM runs program with PetaLinux.



## Functional block diagram

- PS (Zynq ARM) and Zynq PL
- Zynq PS runs PetaLinux
- Zynq PL contains a DANH sub-system comprising of MAC, HSR, AXI and so on.



AMBA AXI: [https://github.com/adki/gen\\_amba/](https://github.com/adki/gen_amba/)

## Directory structure

| directory                                                                                |                             |                                                              | remarks |
|------------------------------------------------------------------------------------------|-----------------------------|--------------------------------------------------------------|---------|
| petalinux                                                                                | PetaLinux building projects |                                                              |         |
|                                                                                          | Makefile                    | It build 'zed-plnx' and calls 'make' in the 'zed-fsbl'.      |         |
|                                                                                          | zed-fsbl                    | FSBL building project.<br>This project prepares boot loader. |         |
|                                                                                          |                             | Makefile                                                     |         |
|                                                                                          | zed-plnx                    | Test-bench                                                   |         |
| sw.arm.petalinux                                                                         | Software building projects  |                                                              |         |
|                                                                                          | hello                       | 'Hello world' project                                        |         |
|                                                                                          |                             | Makefile                                                     |         |
|                                                                                          | eth_send_receive            | HSR DANH testing program                                     |         |
|                                                                                          | Makefile                    |                                                              |         |
| <b>NOTE:</b><br>This project uses HW platform prepared in the 'hsr.danh.arm' project[8]. |                             |                                                              |         |

## Building system

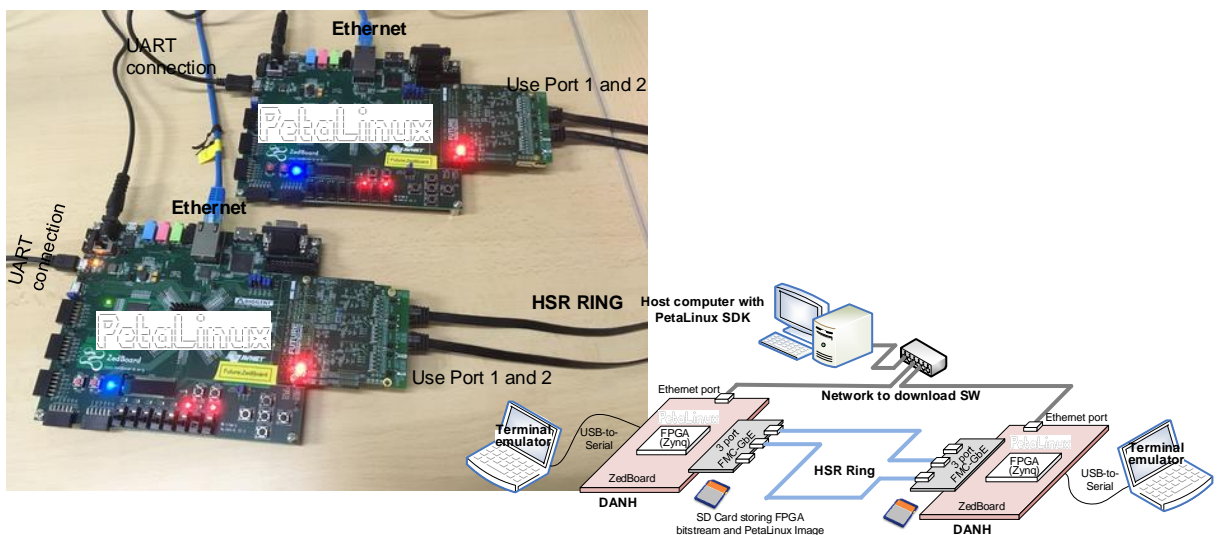
### ■ Preparing bit file for HW

- ▶ `$ cd $(PROJECT)/hsr.danh.arm`
- ▶ `$ ./RunAll.sh -step hw`
- ▶ At last, following files should be ready as a result of building HW platform.
  - ➔ `hw/impl/zedboard.lpc/zed_bd_wrapper.bit`
  - ➔ `hw/impl/zedboard.lpc/zed_bd_wrapper_sysdef.hdf`

### ■ Preparing PetaLinux image

- ▶ More details see following page
- ▶ <https://github.com/github-fds/examples.fmc.gbe.rj45/tree/master/hsr.danh.petalinux>

## Two DANH with PetaLinux



## Sending and receiving packets

| Terminal A                                                       | Terminal B                                                                                                                     | Remarks                                                                                                                               |
|------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| \$ ./eth_send_receive                                            | \$ ./eth_send_receive                                                                                                          |                                                                                                                                       |
| monitor> mac_init                                                | monitor> mac_init                                                                                                              | initialize                                                                                                                            |
| monitor> mac_addr -r<br>MAC 0x021234567801<br>HSR 0x021234567801 | monitor> mac_addr -r<br>MAC 0x021234567802<br>HSR 0x021234567802                                                               | check mac and hsr address.<br>MAC and HSR should be the same<br>.                                                                     |
| monitor> pkt_snd -b 0x0213456788902                              |                                                                                                                                | send a packet to the other                                                                                                            |
|                                                                  | monitor> pkt_rcv -v 3<br>ETH mac dst: 0x02123456782<br>ETH mac src: 0x021234567801<br>ETH type leng: 0x0001<br>[60] 02:12:.... | receive a packet                                                                                                                      |
|                                                                  | monitor> pkt_rcv -v 3 -r                                                                                                       | let receive packets                                                                                                                   |
| monitor> pkt_snd -b 0x021234567801 -r                            |                                                                                                                                | send and receive packets                                                                                                              |
|                                                                  |                                                                                                                                | One sends packets continuously.<br>The other receives packets.<br>Try to dis-connect network cable in<br>order to check HSR recovery. |

㈜퓨처디자인시스템  
 34051 대전광역시 유성구 문지로 193, KAIST 문지캠퍼스, F723호  
 (042) 864-0211~0212 / [contact@future-ds.com](mailto:contact@future-ds.com) / [www.future-ds.com](http://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](mailto:contact@future-ds.com) / [www.future-ds.com](http://www.future-ds.com)



**FUTURE**  
 Design Systems