FICSES usage

V_2.0

2019/08/10

Akram

Outline

- Perquisites
- Revisions
- High-level architecture
- FIC design modifications
- HLS modules
- Testing
- Remarks

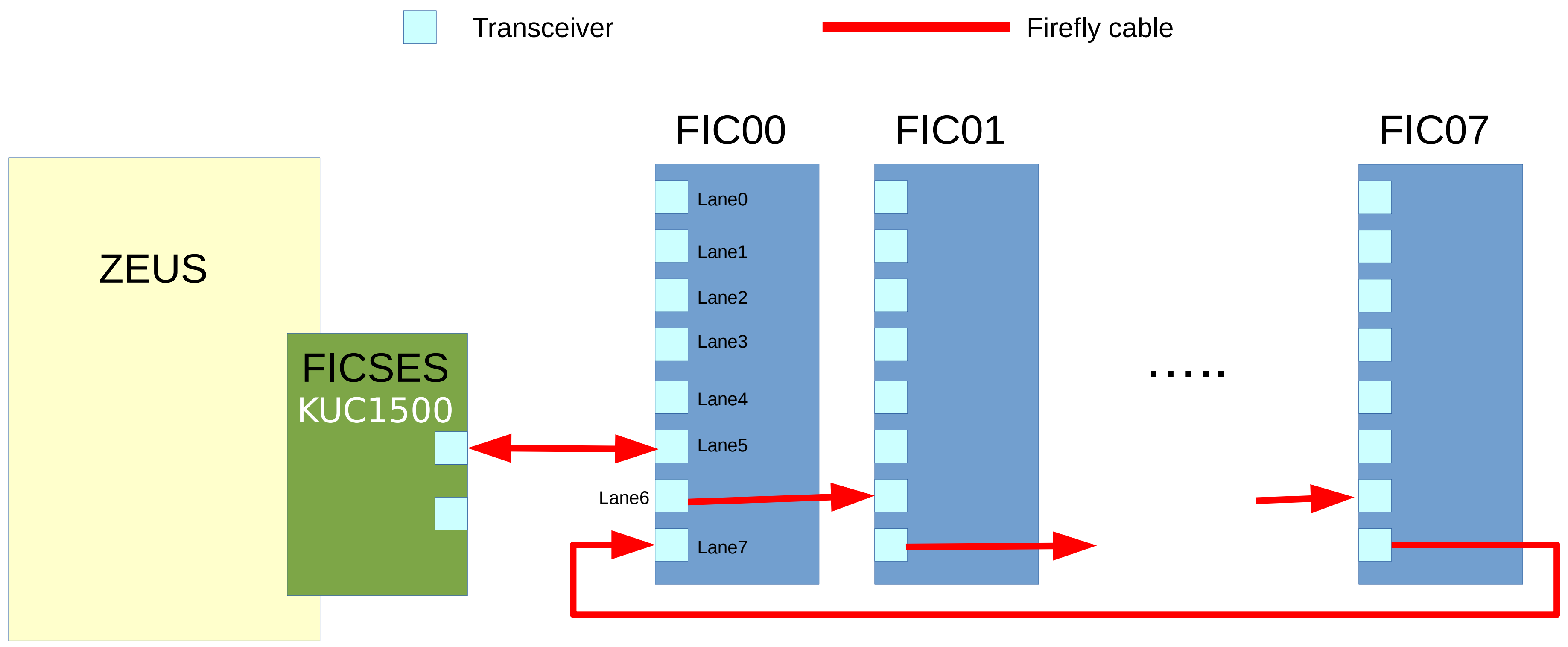
Perquisites

- This document is based on:
  - :drive/FIC_DDR4_Feb22_2019.pdf
  - :drive/FICSES.pptx
- Please read carefully the above two documents before going any further

Revisions

- V_1.0
  - Initial version
- V_2.0
  - Change the connecting lane between FICSES and FIC boards from Lane3 to Lane5
  - Add FICSES support to MARK2 boards
  - Notes about HLS output to FICSES
  - FICSES control can be done using C programs (ficses.h) instead of shell scripts

High-level architecture



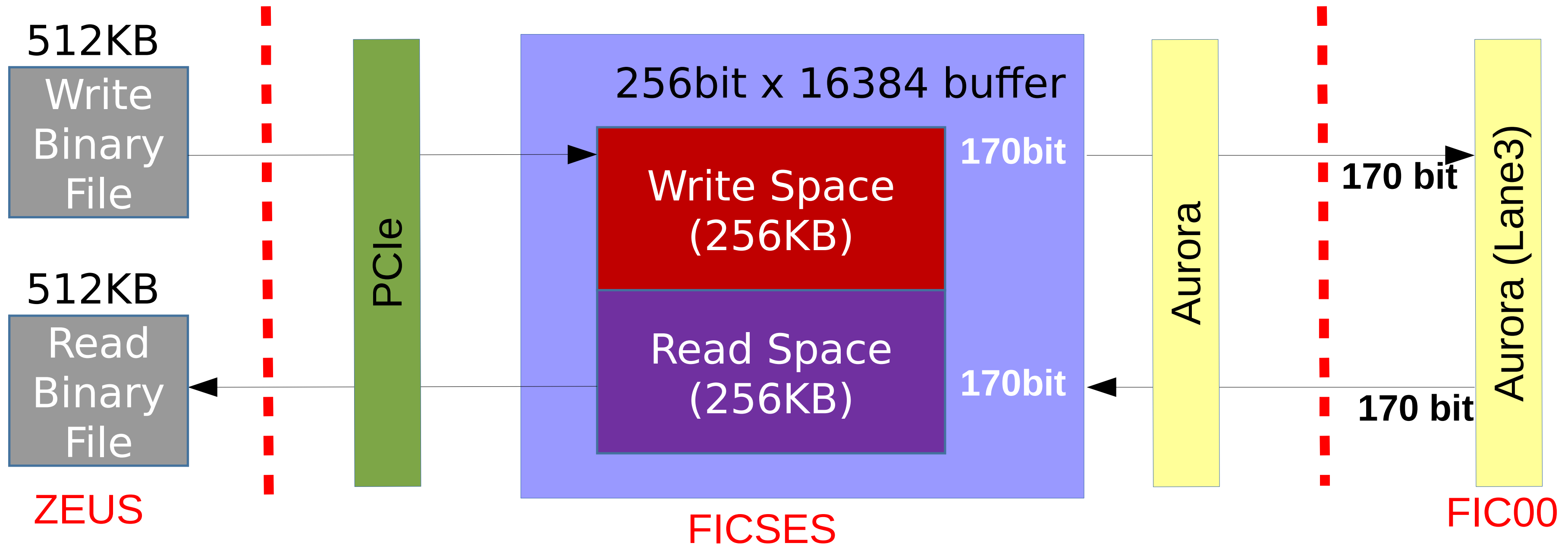
- FICSES is attached to ZEUS through PCIexpress.
- Two transceivers are available on FICSES. But only one is used at the moment.
- FICSES is connected to FIC00 via **Lane5** through a Firefly cable.
- In FIC00 ~ FIC07, **Lane6** and **Lane7** are reserved for a ring network to transfer the data and control from/to ZEUS
- Lane0~Lane4 can be used for the user network

High-level architecture

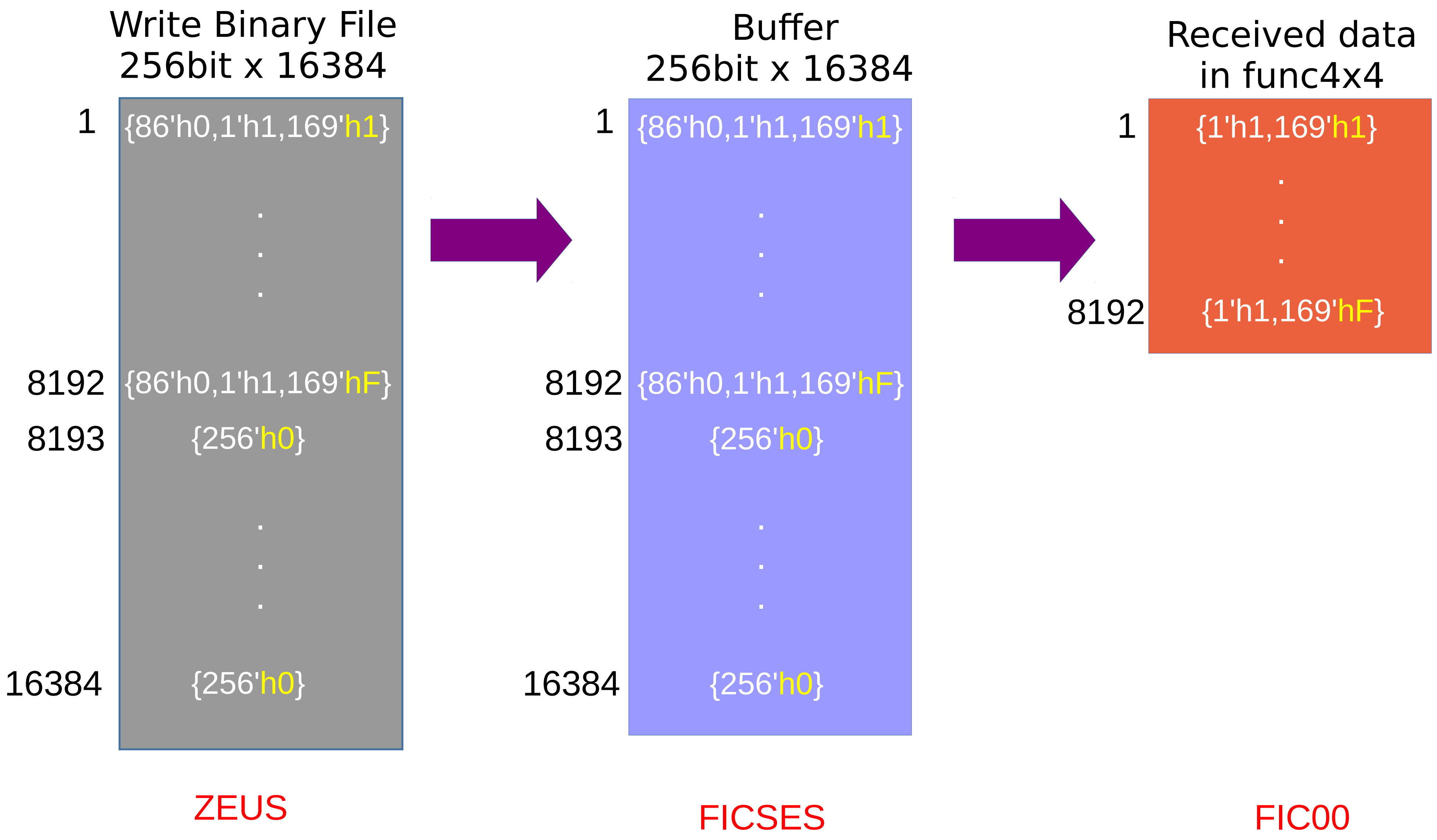
- The lane used by FICSES to connect to FIC00 has four channels.
  As explained in "*FICSES.pptx*", ZEUS writes a 512KB binary file to FICSES for each lane: 256KB write data + 256KB zeros (read space initialization).

- The received data in FICSES is first stored in a 512KB buffer (256bit x 16384).
  Then, only the the lowest 170 bit are forwarded to Aurora. **Data[255:170] is discarded!**
  The 170th bit must be "1" and it is used as the valid signal (Similar to FIC!).

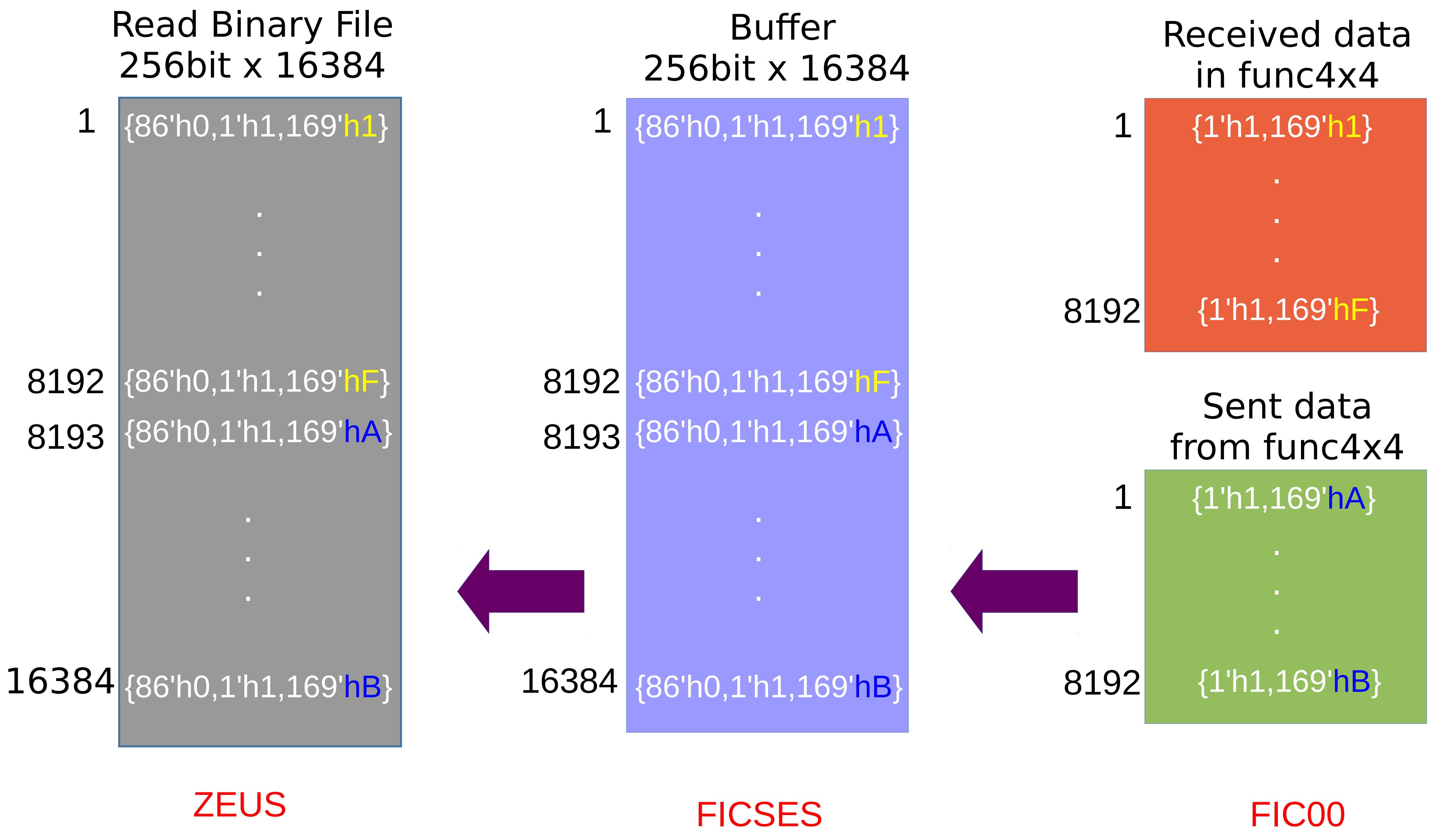
- In a similar approach, ZEUS reads the entire 512KB FICSES buffer and store it to a binary file.
  The "*Read Binary File*" contains both the previously write data as well as the new data read from FICSES.



High-level architecture: Zeus --> FIC00

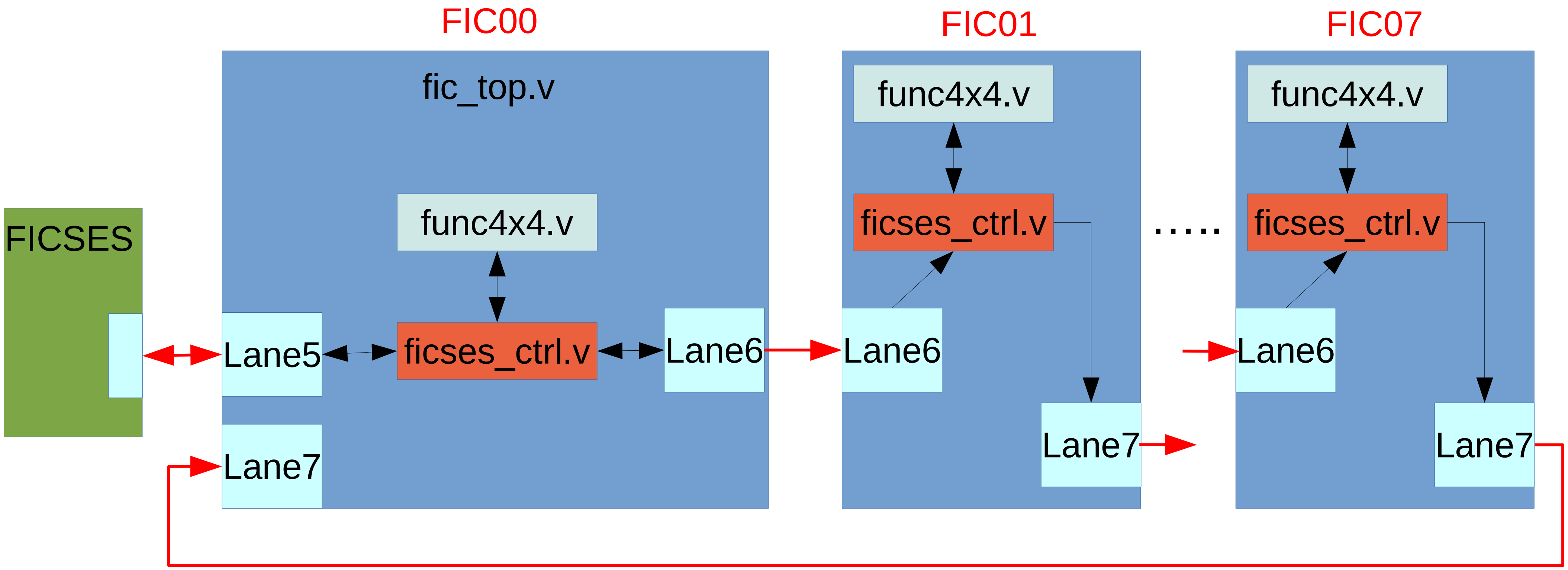


High-level architecture: FIC00 --> Zeus



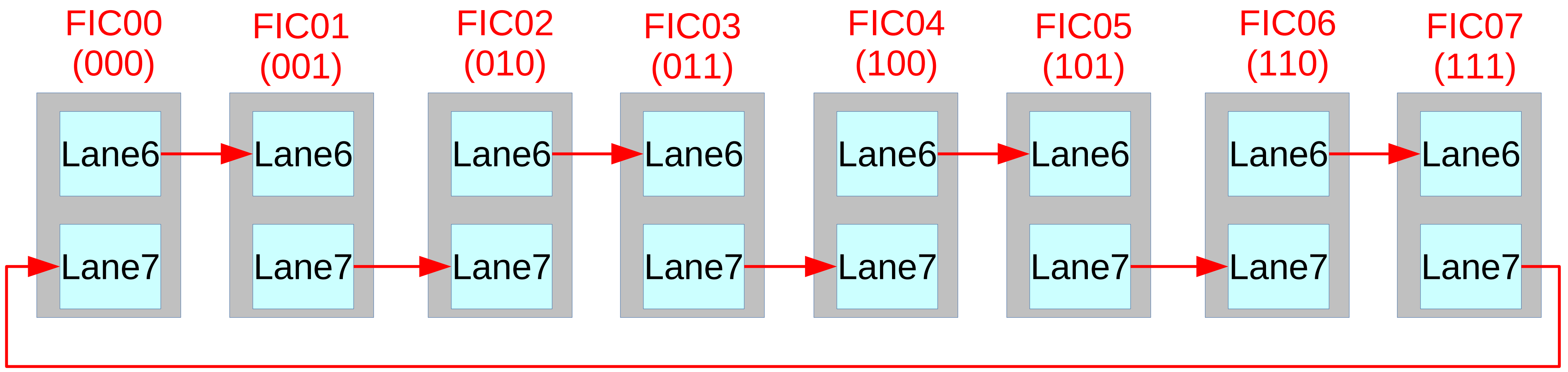
FIC design modification: Top level

- To allow receiving/sending packets from/to ZEUS, a new module is added at the top_level:
  - In FIC00, the new module (**ficses_ctrl.v**) receives data/control from FICSES through Lane5, and either pass it to the local **func4x4.v**, or forward it to other FIC boards through Lane6.
  - In FIC01~07, **ficses_ctrl.v** receives data/control from other FIC boards through Lane6/7, and either pass it to the local **func4x4.v**, or forward it to other FIC boards through Lane6/7.



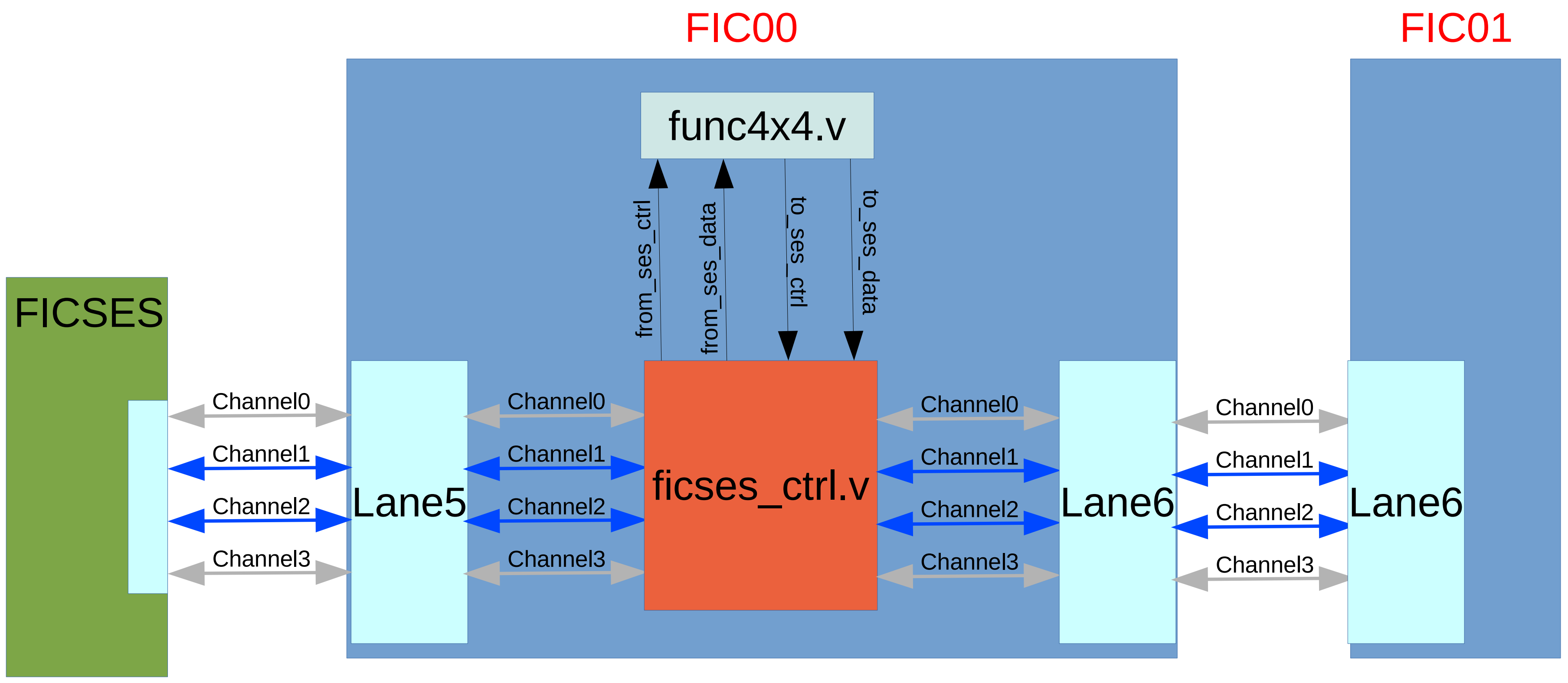
FIC design modification: Data/Control path

- When writing data (DDR or control reg) from ZEUS to FIC01~07, **ficses_ctrl** in FIC00 always forwards the write packet via Lane6.
- When reading data (DDR or control reg) from FIC01~07, this later should always make sure that the reply packet arrives to FIC00 via Lane7.
  - if(board_id[0] == 0) send reply through Lane6
  - if(board_id[0] == 1) send reply through Lane7



FIC design modification: Channel partition

- Although each lane has 4 channels, in the current design only two channels are used in FICSES:
  - Channel 0 (unused)
  - Channel 1 (Control): dedicated for register access in func4x4
  - Channel 2 (Data): dedicated for DDR data access (write/read)
  - Channel 3 (unused)



FIC design modification: Packet format

From ZEUS, users always send a 256bit packet. In FICSES, only 170bit are kept and transferred to FIC00 in the below format:

- **Valid**:
  - Should always be assigned to “1”; otherwise it will be dropped in FICSES
- ***Payload:***
  - Data to be written to DDR or control reg
- **Command**:
  - 9'h1: *ap_start*
  - 9'h2: *write*
  - 9'h3: *read*
  - 9'h4: *ap_reset*
  - 9'h5: *Routing table setting*

Valid 1bit	Command 9bit	Dest_board 8bit	Src_board 8bit	Register Address 16bit	Payload 128bit

FIC design modification: Packet format

- **Dest_board**:
  - Board ID to which ZEUS wants to write/read to/from.
  - The ID is represented by the Dip_switch value in the board.
  - When setting **Dest_board** to **0xBB**, a write packet is broadcasted to all the boards connected in the ring network (Read broadcast is not supported at the moment!!)
- **Src_board**:
  - Board ID from which the data/control is coming from.
  - **0xFF** represents ZEUS ID

Valid	Command	Dest_board	Src_board	Register Address	Payload
1bit	9bit	8bit	8bit	16bit	128bit

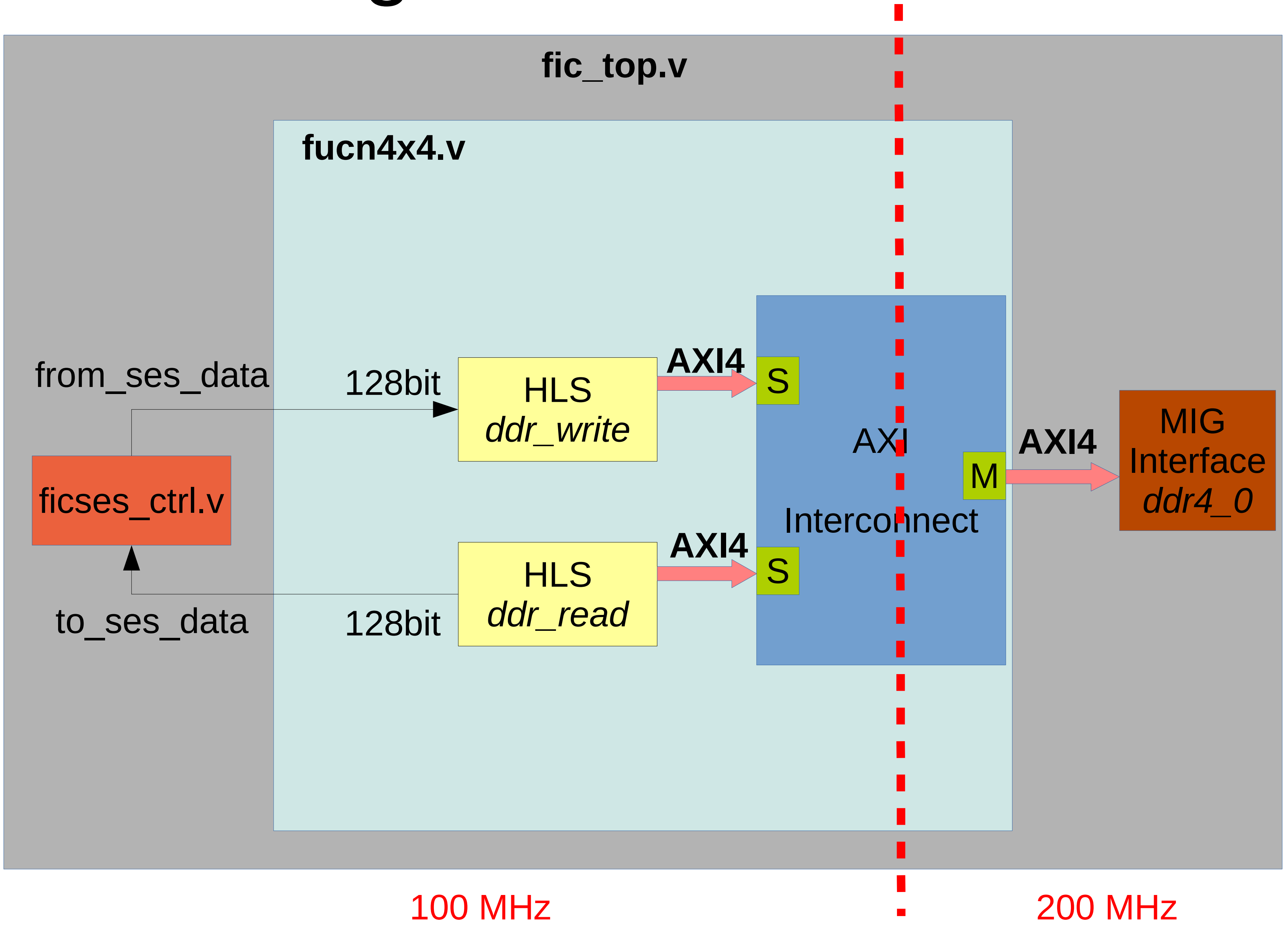
FIC design modification: Packet format

- **Register Address**:

– **SES_STAT**	16'hffff	// (R/W)	Status register
– **SES_TEST**	16'hfffe	// (R)	Direct read/write to HLS module
– **SES_LUP**	16'hfffd	// (R)	Link up
– **SES_DIPSW**	16'hfffc	// (R)	Dipsw value
– **SES_LED**	16'hfffb	// (R/W)	LED value.
– **SES_CHUP**	16'hfffa	// (R)	Channel up
– **SES_TIMER**	16'hfff9	// (R)	Timer
– **SES_SNUM**	16'hfff8	// (R/W)	Slot number register
– **SES_PKT0_3**	16'hfff7	// (R)	PKT monitor 0~3
– **SES_PKT4_7**	16'hfff6	// (R)	PKT monitor 4~7
– **SES_SLOT**	16'h2000	// (R/W)	Slot register
– **SES_DDR_WR_START**	16'hDDD1	// (W)	For ddr_write_start HLS module
– **SES_DDR_RD_START**	16'hDDD2	// (W)	For ddr_read_start HLS module
– **SES_DDR_ADDR**	16'hDDDA	// (R/W)	DDR address

Valid	Command	Dest_board	Src_board	Register Address	Payload
1bit	9bit	8bit	8bit	16bit	128bit

HLS modules: High-level view



HLS modules: DDR Access

- Two HLS modules are used in this design similar to “:drive/FIC_DDR4_Feb22_2019.pdf”
  - **ddr_write**: receives data from FICSES and write them to DDR
  - **ddr_read**: read from DDR and send them to FICSES
- In func4x4, **Payload** (128bit) is fetched from the 170bit **from_ses_data** packet coming from “ficses_ctrl” and fed to “ddr_write” HLS module
- Also, func4x4 adds the necessary fields to the read data (128bit) from the “ddr_write” HLS module and send it to “ficses_ctrl” as 170bit **to_ses_data** packet
  - Add the valid bit
  - Set the “Dest_board” to **0xFF** (Zeus ID)
  - Set the “Src_board” to the board ID read from the Dip_switch

HLS modules: ddr_write.c

```
#include "ap_cint.h"
#include "string.h"
#define MAXDATA 8192

void ddr_write (uint128 input[], volatile uint128 *ddr)
{
#pragma HLS INTERFACE m_axi port=ddr offset=direct max_write_burst_length=256
#pragma HLS INTERFACE axis port=input
int i;
uint128 buff_wr [MAXDATA];

    for (i = 0; i< MAXDATA; i++) {    // input from FICSES
    #pragma HLS PIPELINE
    buff_wr[i] = input[i];
    }
memcpy((uint128*)ddr, buff_wr, MAXDATA*sizeof(uint128));
return;
}
```

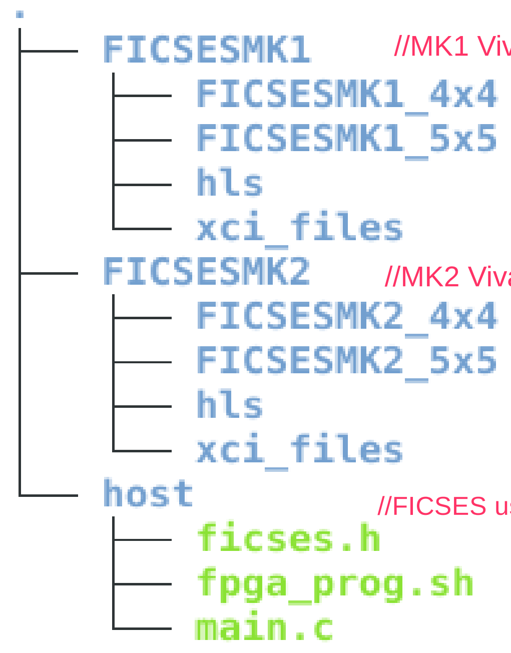
HLS modules: ddr_read.c

```
#include "ap_cint.h"
#include "string.h"
#define MAXDATA 8192

void ddr_read (uint128 output[], volatile uint128 *ddr)
{
#pragma HLS INTERFACE m_axi port=ddr offset=direct max_write_burst_length=256
#pragma HLS INTERFACE axis port=output
int i, j;
uint128 buff_rd [MAXDATA];

memcpy(buff_rd, (uint128*)ddr, MAXDATA*sizeof(uint128));

for (i = 0, j=0; i< MAXDATA; i++, j+= 2) {
#pragma HLS PIPELINE
   output[j] = buff_rd[i];
   output[j+1] = buff_rd[i];
}
return;
}
```

- FICSES requires that each packet sent over AURORA should be 2 cycles (@100MHz)
- If the HLS output is directly sent to FICSES, you should make sure that each data is sent over 2 cycles. Otherwise, the data will be invalid!
- To respect this requirement, in this example the same data is outputted twice.

Testing

- Example designs that allow users to perform DDR access and check for both MARK1 and MARK2, as well as sw4x4 and sw5x5, is provided under:
  /home/asap/fic/FICSES/
- The directory structure is as follows:



```
.
├── FICSESMK1          //MK1 Vivado projects
│   ├── FICSESMK1_4x4
│   ├── FICSESMK1_5x5
│   ├── hls
│   └── xci_files
├── FICSESMK2          //MK2 Vivado projects
│   ├── FICSESMK2_4x4
│   ├── FICSESMK2_5x5
│   ├── hls
│   └── xci_files
└── host
    ├── ficses.h       //FICSES user programs
    ├── fpga_prog.sh
    └── main.c
```

Testing

- The usual control from RaspberryPi is preserved.
- From Zeus (through FICSES) users can access any FIC board (MARK1 and MARK2) connected to the ring:
  - Start and reset HLS module
    - ✓ Through the control channel (**Ch1**)
  - Write/read a control register in func4x4/func5x5 on each board
    - ✓ Through the control channel (**Ch1**)
  - Setup routing table
    - ✓ Through the control channel(**Ch1**)
  - Write/read data to/from DDR
    - ✓ Through the data channel (**Ch2**)

Testing:
./host

- The “host” directory contains the following:
  - **fpga_prog.sh**: download bitsream to all 8 boards (by nyacom)
  - **ficses.h**: header file for FICSES control
  - **main.c**: User program

Testing: ficses.h

- **ficses.h** contains the following fuctions:
  - int ficses_write_128KB(char* binary_file, int ficses_channel)
  - int ficses_read_128KB(char* binary_file, int ficses_channel, int * read_data)
    - int read_binary_file (char* binary_file, int * read_data)
    - int ficses_start(int channel_enable)
  - int ficses_register_access(int BOARD_ID, int command, int address, int payload, int ficses_channel, int * read_data)
    - int ficses_register_read(int BOARD_ID, int address, int ficses_channel)
    - int ficses_register_write(int BOARD_ID, int address, int payload, int ficses_channel)
    - int ficses_ap_start(int BOARD_ID, int ficses_channel)
    - int ficses_ap_rst(int BOARD_ID, int ficses_channel)
  - int ficses_table_setting(int BOARD_ID, char* table_file, int ficses_channel)

  - int user_write_data(float *array, int BOARD_ID, int data_count, int ficses_channel) (Customizable)

Testing: main.c (example)

```c
#include "ficses.h"
#define TBL_FILE0 "/home/asap/fic/ras_hunga/tbl/loop0_1.dat"
#define TBL_FILE1 "/home/asap/fic/ras_hunga/tbl/pass0_1.dat"
#define FIC00_ID    0x0
#define FIC01_ID    0x1
#define BRODCAST    0xBB
#define SES_PKT0_3  0xfff7
#define DATA_SIZE   32

int main(int argc, char *argv[]) {

    float *wb;

    ficses_ap_rst(BRODCAST, FICSESL0_CH1);
    ficses_ap_start(FIC00_ID, FICSESL0_CH1);

    ficses_table_setting(FIC00_ID, TBL_FILE0, FICSESL0_CH1);
    ficses_table_setting(FIC01_ID, TBL_FILE1, FICSESL0_CH1);

    ficses_ap_start(FIC00_ID, FICSESL0_CH1);

    user_write_data (wb, FIC00_ID, DATA_SIZE, FICSESL0_CH2);

    ficses_register_read(FIC00_ID, SES_PKT0_3, FICSESL0_CH1);

    return 0;
}
```

Remarks

- You must login as root in order to execute the different functions in ficses.h
- The board IDs should be always selected as (odd-even-odd-even-..)
- To broadcast a write command to a register, use **0xBB** as the destination boards.
  (Read broadcast is not supported at the moment!!)