eMMC Host Controller 5.1 IP Integration Manual

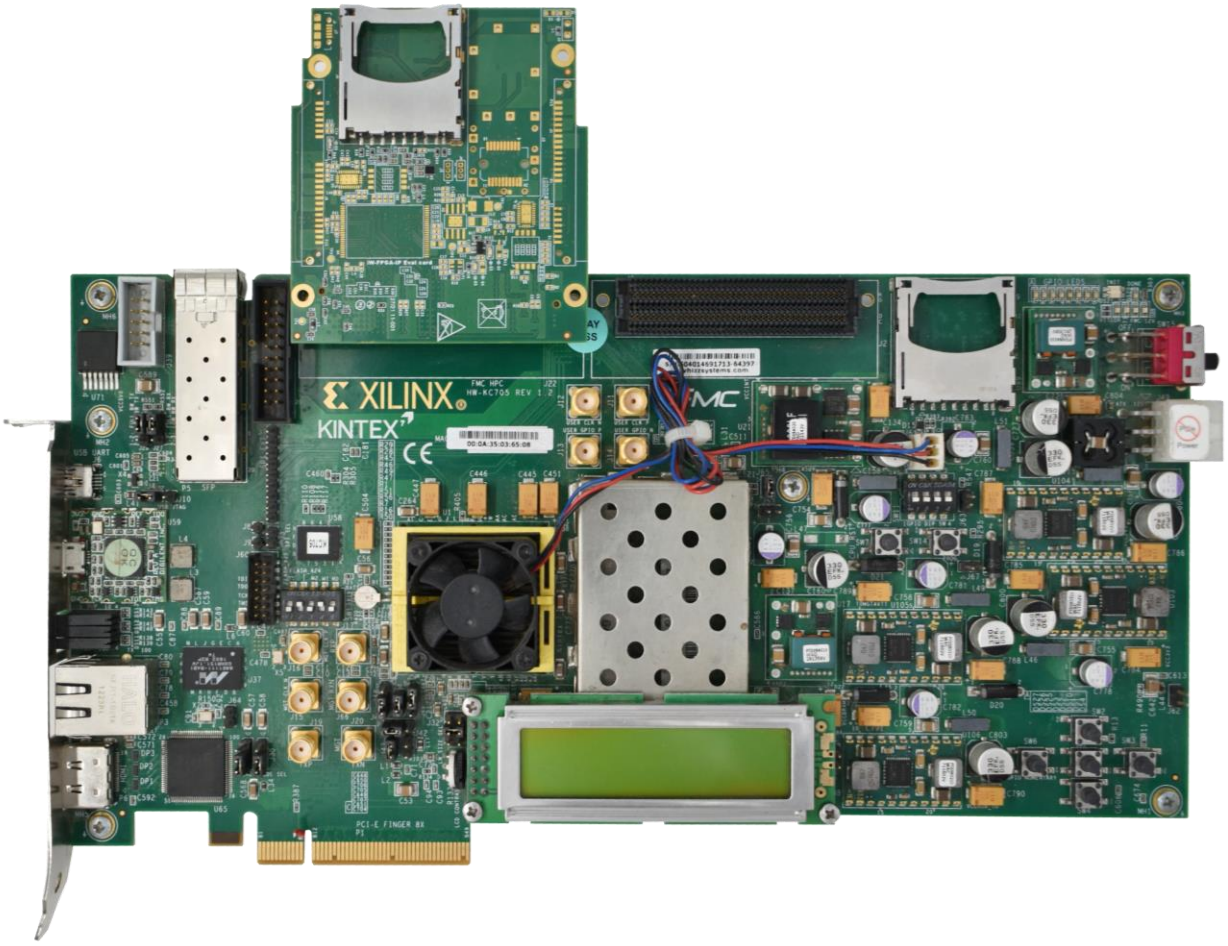

Table of Content

List Of Figures

List Of Tables

1 Introduction

1.1 Purpose

The purpose of this document is to describe eMMC Host Controller 5.1. IP Integration details.

1.2 Reference Document

- iW-EMFFC-DS-01-R1.0-REL1.0

1.3 Overview

iW-eMMC 5.1 Controller interfaces MMC / eMMC card to any processor with a generic interface. The interface towards the eMMC is realized by the eMMC protocol implemented in the controller. The core supports AXI4-Lite interface for the control and status register access and AXI4-MM interface for data transfer through ADMA2 mode. eMMC Host Controller interfaces the Microblaze (soft processor) through AXI4 bus and MiG Controller through AXI memory mapped interface, enabling the data transfers between each other.

1.4 Acronyms and Abbreviations

Table 1 : Acronyms & Abbreviations

Term	Meaning
FPGA	Field Programmable Gate Array
eMMC	Embedded Multi Media Card
GPIO	General Purpose Input Output
FMC	FPGA Mezzanine Card
LUT	Look Up Table
IO	Input and Output
FF	Flip Flop

2 IP Configuration and Instantiation

2.1 Example design

The eMMC / host controller IP example design mainly consists of,

1. **Microblaze soft core:** User can provide the inputs to run the different tests using the bare metal code running in the Microblaze. This will pass the different control signals to eMMC host controller based on the user selection.
2. **eMMC Host Controller IP:** eMMC host controller IP takes the command given from the microblaze soft core to the register set and with the ADMA the read and write operation will happen from and to eMMC Card

2.2 IP Configuration

The eMMC Host Controller IP is instantiated in the design as shown in the figure for generating the mmc_host with other files like the axi4_lite interface and the axi memory mapped interface wrapper files are added to make a user configured IP to be easy add in the block design.

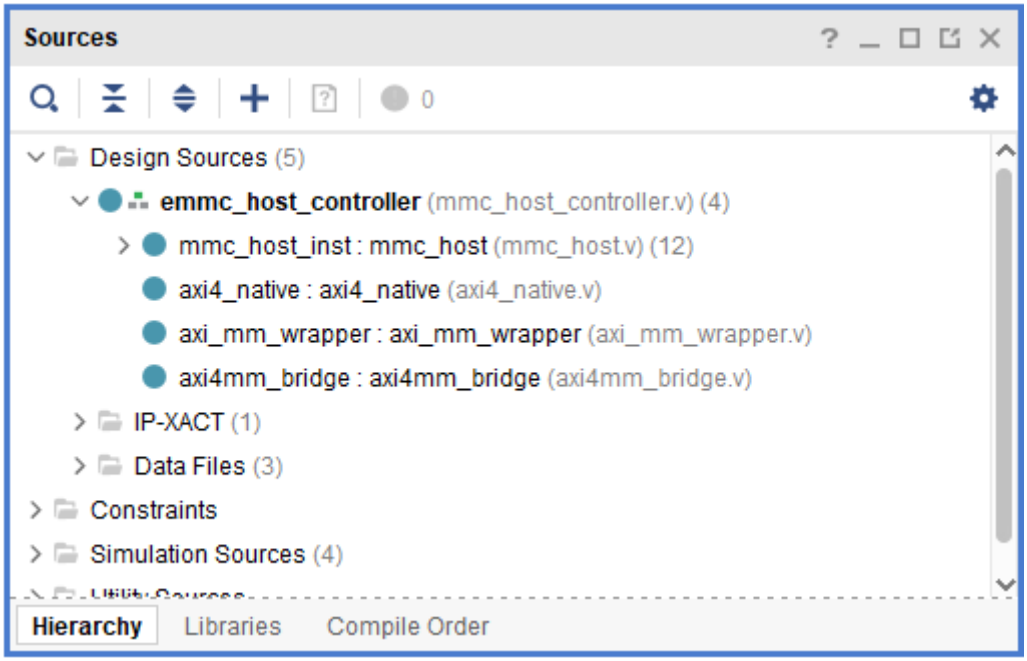

Figure 1 : Design sources where eMMC host is instantiated

Module emmc_host_controller is the top module of the project which integrates the all the files to form a single IP.

So once the IP is generated then make sure that the IP need to be added to the project so copy the IP folder from the path that was saved to the local to project folder and then go to

settings and add the IP to the Project Settings → IP → Repository → then add the IP path by clicking on the + sign after the IP is added click on Apply and close the dialogue box

Once the IP is added in the project repository open the block design after adding the other IPs like Microblaze, UART, MIG Controller and Clocking Wizard then add the eMMC host controller IP to the block design and make necessary connection as shown in the below figure.

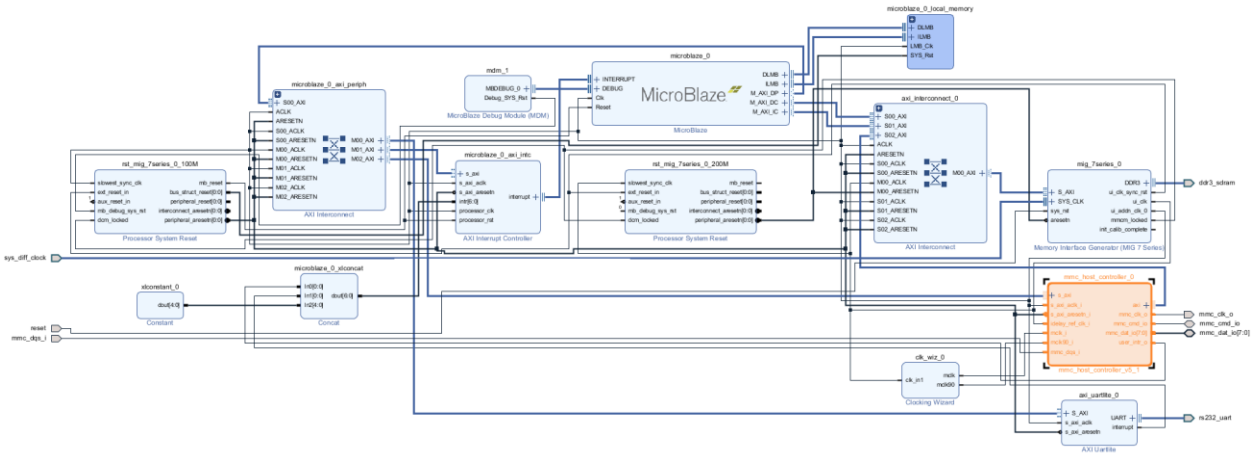

Figure 2 : eMMC Host controller integration in the block design

2.3 Steps to Instantiate Block Design

1. Add the eMMC host controller IP to the block design given in the release folder and then synthesis the block design and generate the HDL Wrapper. Then instantiate that in the top module.
2. Instantiate the top module of with the ports and signals of block design with all the IPs, in the top_module.v file

```
mb_preset_wrapper mb_preset_wrapper_inst(
  // Clock and Reset Signals
  .sys_diff_clock_clk_n  ( sys_diff_clock_clk_n ),
  .sys_diff_clock_clk_p  ( sys_diff_clock_clk_p ),
  .reset                 ( reset                ),
  // DDR Signals
  .ddr3_sdram_addr       ( ddr3_sdram_addr      ),
  .ddr3_sdram_ba         ( ddr3_sdram_ba        ),
  .ddr3_sdram_cas_n      ( ddr3_sdram_cas_n     ),
  .ddr3_sdram_ck_n       ( ddr3_sdram_ck_n      ),
  .ddr3_sdram_ck_p       ( ddr3_sdram_ck_p      ),
  .ddr3_sdram_cke        ( ddr3_sdram_cke       ),
  .ddr3_sdram_cs_n       ( ddr3_sdram_cs_n      ),
  .ddr3_sdram_dm         ( ddr3_sdram_dm        ),
  .ddr3_sdram_dq         ( ddr3_sdram_dq        ),
  .ddr3_sdram_dqs_n      ( ddr3_sdram_dqs_n     ),
  .ddr3_sdram_dqs_p      ( ddr3_sdram_dqs_p     ),
  .ddr3_sdram_odt        ( ddr3_sdram_odt       ),
  .ddr3_sdram_ras_n      ( ddr3_sdram_ras_n     ),
  .ddr3_sdram_reset_n    ( ddr3_sdram_reset_n   ),
  .ddr3_sdram_we_n       ( ddr3_sdram_we_n      ),
  // Uart Signal
  .rs232_uart_rxd        ( rs232_uart_rxd       ),
  .rs232_uart_txd        ( rs232_uart_txd       ),
  // eMMC Signals
  .mmc_clk_o             ( mmc_clk_o            ),
  .mmc_cmd_io            ( mmc_cmd_io           ),
  .mmc_dat_io            ( mmc_dat_io           ),
  .mmc_dqs_i             ( mmc_dqs_i            )
);
```

Figure 3 : Instantiation of block design in top module

3. The Constraint file (.xdc) provided in the design can be changed by user depending on requirements.
4. Give the required clock, Pin/IO constraints for eMMC host controller in the .xdc file and compile the custom design with eMMC host IP.

3 Implementation Details

3.1 Clock Domain

eMMC Host Controller 5.1 IP works on the HS400_ES mode and the base clock mclk_i (phase shifted 60 degree) drives the output clock to the eMMC card. User may need to adjust this clock between 100 MHz to 200 MHz based on the hardware setup. In other words, if user gets CRC error with say 200 MHz, then user can reduce the clock to say 190 MHz or below than to make sure write and read works fine with that base clock.

mclk90_i is a phase shifted version of the mclk_i and this phase shift may need to be tuned if we don't get the response for the CMD21. The phase shift will be initially kept 0 and later the phase shift can be added to this clock until we get the response for CMD21 from card. Idelay_ref_clk_i is the fixed 200 MHz clock which will be provided for the IDELAY primitive to be used for the tuning implementation block. s_axi_clk_i or hclk_i which is 100 MHz which will be used for the communication between the micro blaze and IP using the AXI4-lite interface for register access and AXI4-MM for DMA access.

So, to generate the all these clocks we give a 200MHz on board clock to MIG controller and 200 MHz is generated from the MIG and that is given to the clocking wizard to generate other required clocks.

3.2 Constraints

3.2.1 False path and False Route constraints:

The below figure shows the false path in between all the clocks which is configured for the eMMC host IP. These details will be available in the .xdc file.

- The below mentioned path is for mclk90_i and mclk_i clock which is base clock of eMMC host (200 MHz) and eMMC clock of IP (200MHz),

  set_false_path -from [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/clk_wiz_0/inst/mmcm_adv_inst/CLKOUT1]] -to [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/clk_wiz_0/inst/mmcm_adv_inst/CLKOUT0]]The below mentioned path is for mclk90_i and mclk_i clock which is base clock of eMMC host(200 MHz) and eMMC clock of IP (200MHz), set_false_path -from [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/clk_wiz_0/inst/mmcm_adv_inst/CLKOUT1]] -to [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/clk_wiz_0/inst/mmcm_adv_inst/CLKOUT0]]

- The below mentioned path is for mclk90_i and s_axi_aclk_i clock which is base clock of eMMC host (200 MHz) and AXI clock (100 MHz),

  set_false_path -from [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/clk_wiz_0/inst/mmcm_adv_inst/CLKOUT1]] -to [get_clocks -of_objects [get_pins

mb_preset_wrapper_inst/mb_preset_i/mig_7series_0/u_mb_preset_mig_7series_0_0_mig/u_ddr3_infrastructure/gen_ui_extra_clocks.mmcm_i/CLKFBOUT]]

- The below mentioned path is for s_axi_aclk_i clock and mclk90_i which is base clock of AXI clock (100 MHz) and eMMC host (200 MHz),

  set_false_path -from [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/mig_7series_0/u_mb_preset_mig_7series_0_0_mig/u_ddr3_infrastructure/gen_ui_extra_clocks.mmcm_i/CLKOUT0]] -to [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/clk_wiz_0/inst/mmcm_adv_inst/CLKOUT1]]

- The below mentioned path is for mclk90_i and idelay_ref_clk_i clock which is base clock of eMMC host (200 MHz) and Idealy reference clock of eMMC host (200MHz),

  set_false_path -from [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/clk_wiz_0/inst/mmcm_adv_inst/CLKOUT1]] -to [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/mig_7series_0/u_mb_preset_mig_7series_0_0_mig/u_ddr3_infrastructure/gen_ui_extra_clocks.mmcm_i/CLKFBOUT]]

- Clock route false since data strobe pin is connected to N side of Clock capable pin

  set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets mb_preset_wrapper_inst/mb_preset_i/mmc_host_controller_0/inst/mmc_dqs]

3.2.2 Pin Constraints:

Following figure shows the example pin and IO standard constraints.

```
set_property PACKAGE_PIN AF21 [get_ports mmc_cmd_io]
set_property PACKAGE_PIN AE23 [get_ports mmc_clk_o]
set_property PACKAGE_PIN AK25 [get_ports {mmc_dat_io[0]}]
set_property PACKAGE_PIN AE21 [get_ports {mmc_dat_io[1]}]
set_property PACKAGE_PIN AJ29 [get_ports {mmc_dat_io[2]}]
set_property PACKAGE_PIN AH22 [get_ports {mmc_dat_io[3]}]
set_property PACKAGE_PIN AK24 [get_ports {mmc_dat_io[4]}]
set_property PACKAGE_PIN AC25 [get_ports {mmc_dat_io[5]}]
set_property PACKAGE_PIN AH27 [get_ports {mmc_dat_io[6]}]
set_property PACKAGE_PIN AK30 [get_ports {mmc_dat_io[7]}]
set_property PACKAGE_PIN AC27 [get_ports mmc_dqs_i]
set_property PACKAGE_PIN AH20 [get_ports rst_n_o]

set_property IOSTANDARD LVCMOS18 [get_ports mmc_cmd_io]
set_property IOSTANDARD LVCMOS18 [get_ports mmc_clk_o]
set_property IOSTANDARD LVCMOS18 [get_ports {mmc_dat_io[7]}]
set_property IOSTANDARD LVCMOS18 [get_ports {mmc_dat_io[6]}]
set_property IOSTANDARD LVCMOS18 [get_ports {mmc_dat_io[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {mmc_dat_io[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {mmc_dat_io[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {mmc_dat_io[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {mmc_dat_io[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {mmc_dat_io[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports mmc_dqs_i]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n_o]
```

Figure 4 : Pin Constraints in example design .xdc file

NOTE: These constraints are in accordance to example design created for the KC705 Evaluation kit and there is no need of adding the MIG/ DDR IO pins, UART IO pins.
Define the constraints for clock, reset and UART pins accordingly for any other custom board.

4 Test setup

4.1 Test requirements

- Xilinx KC705 dev kit
- iWave FMC daughter card
- USB cables for JTAG and UART
- Power Supply

iWave FMC card requires Vadj -1.8V for its operation. In order to set Vadj to 1.8V, rework needs to be done in Xilinx KC705 board as its Vadj =2.5v by default. User need to do below rework,

- Remove resistor R576
- Removed inductor L47
- L47 should be connected to R576 resistor pads using some thick gauge wire.

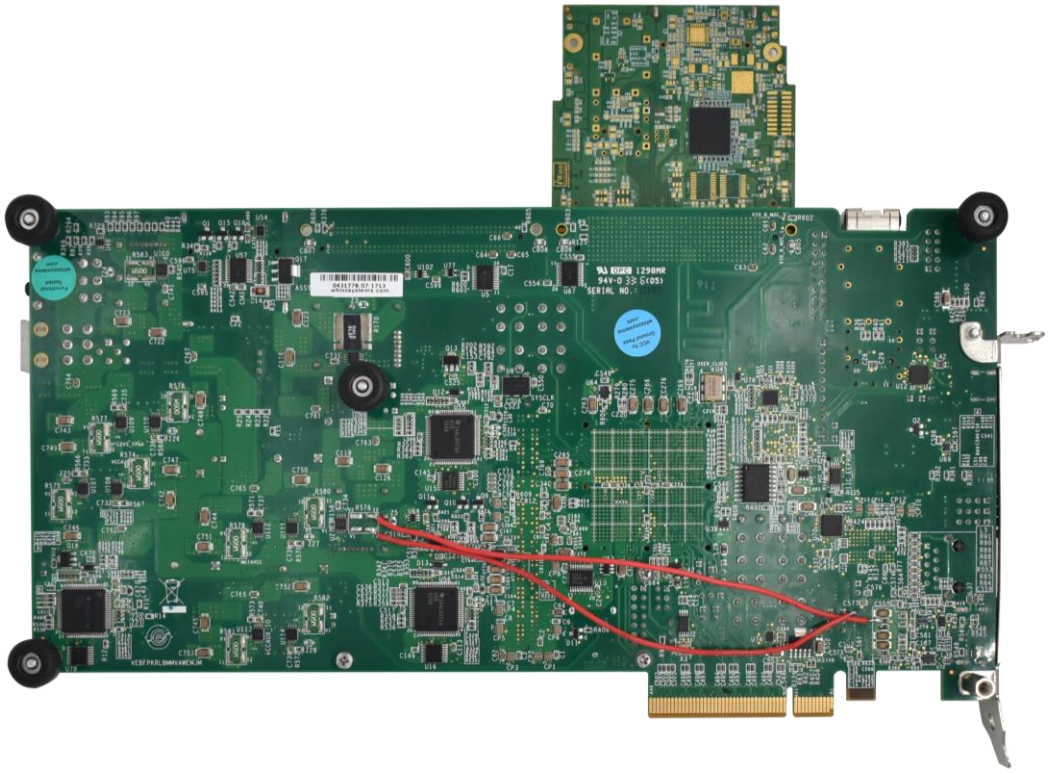

Figure 5 : Vadj Voltage control

Refer the above figure to set the FMC Vadj voltage to 1.8V. The Vadj value depends on the custom board and this rework is only needed for testing iWave FMC daughter card with KC705 dev kit.

4.2 Connections

Please make sure that FMC card of eMMC device interface is connected to FPGA board FMC bank voltage is set to 1.8V.

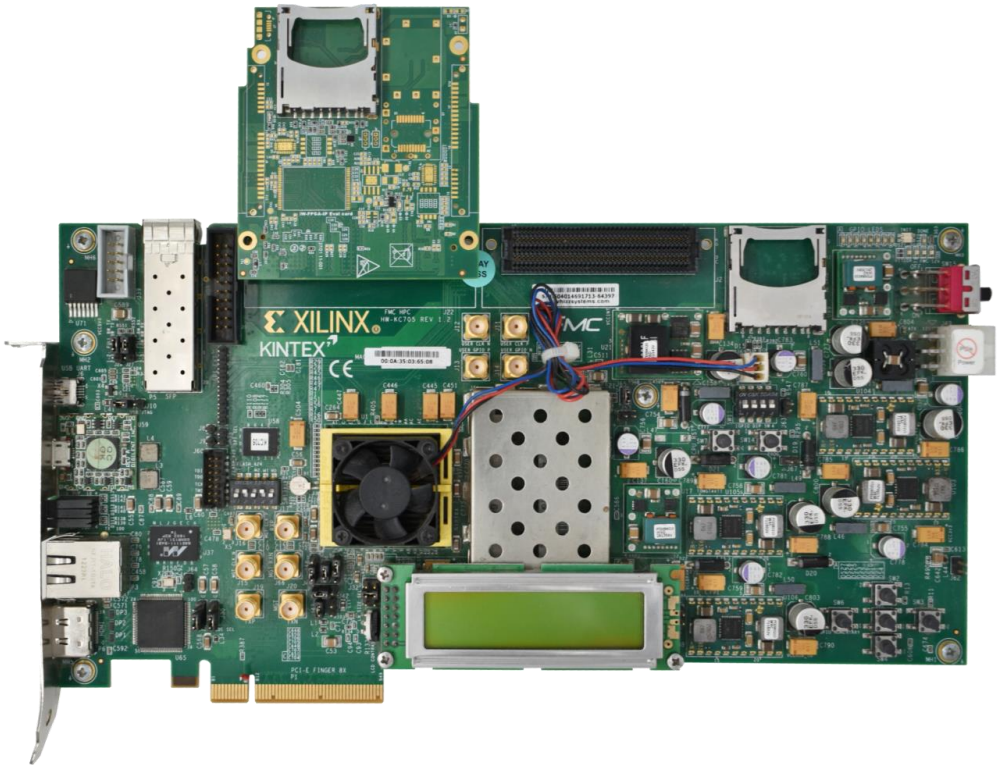

Figure 6 : FMC card connected to FMC bank of KC705 board

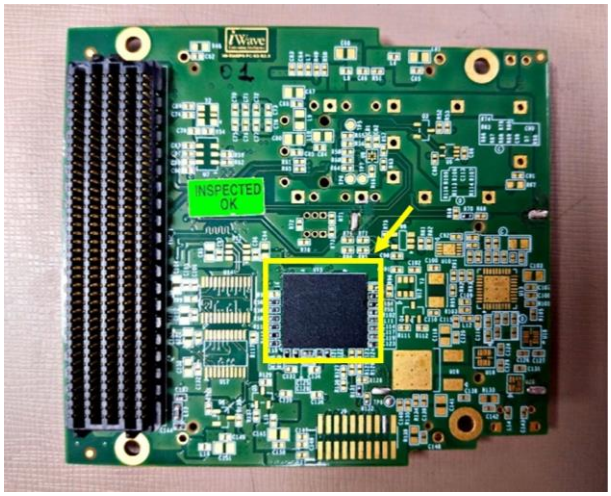

Figure 7 : eMMC FMC connector

4.3 Programming FPGA and running using Vitis.

- After generating bitstream in Vivado go to File→ Export hardware → select Fixed click next → select include bitstream and click next → give the XSA file name and keep directory as local to project and click Finish.
- After exporting the project go to tools and click on launch Vitis IDE, a new window will appear, create a local folder in the Project folder and give that as workspace folder for Vitis project and click ok.
- Once Vitis is opened then the click on creates new application project and then generate the platform file giving the .xsa file path by browsing to that file in the projects local folder and click next
- In Available templets click on empty project and click on finish.
- Once the project is open, copy the source code from the release folder (..\iW-EMFFC-PF-01-R1.0-REL1.1\iW-EMFFC-SF-01-R1.0-REL1.1\iW-EMFFC-SC-01-R1.0-REL1.1\sd_host_vitis) to Vitis project and then Save (CTRL+S) and Build (CTRL+B) the project
- Then right click on project go to run and select run configuration system project debug
- In the appeared window under target setup click on reset entire system and program FPGA as shown in the below figure.

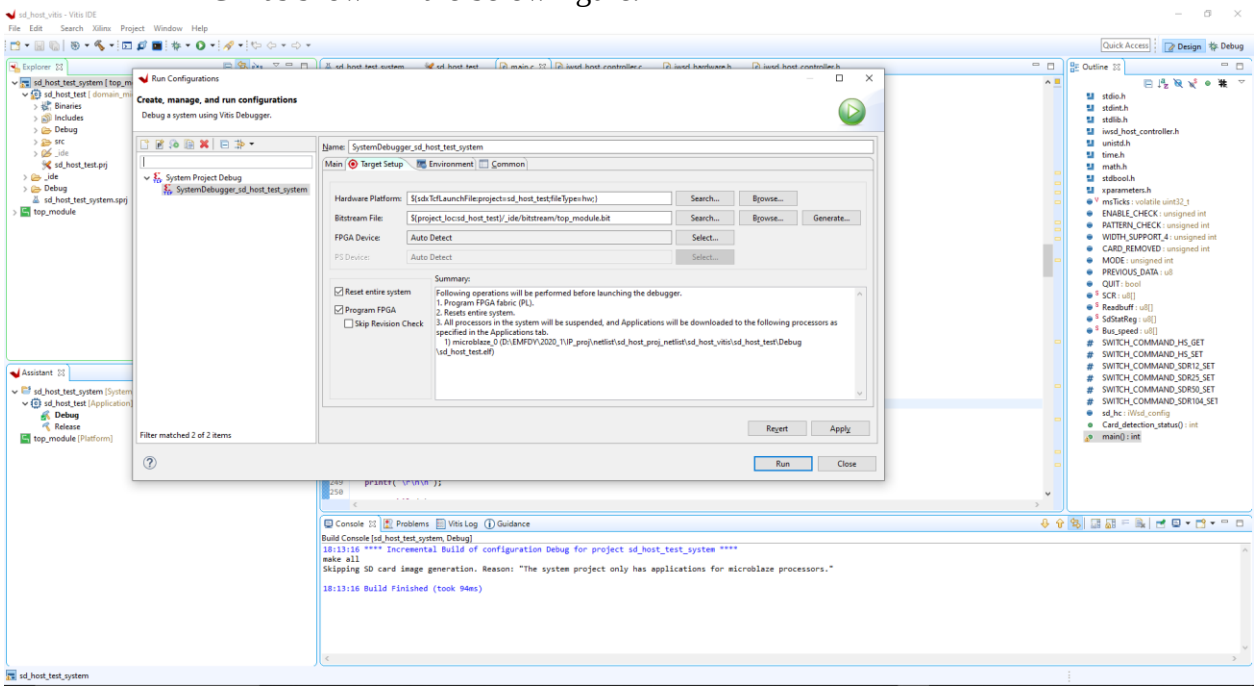

Figure 8 : Target setup in run configuration.

- Now click on Run

- After clicking on Run FPGA will be programmed and board is test ready.

NOTE: Before clicking on Run please make sure board is turned on with JTAG connect and proper tera term setup and if program FPGA is done using Vivado please don’t select program FPGA option in target setup.

5 Test Procedure

5.1 Tera term setup

- Open tera term in PC and select the com port.
- Select baud rate as 115200.

5.2 Testing Procedure

After set up follow the below procedure to test the eMMC Host controller IP.

5.2.1 Initial prints:

After clicking on Run in Run configuration software will program the FPGA. As soon as the programming is done and .elf file is executed, then below print will be displayed in Tera term.

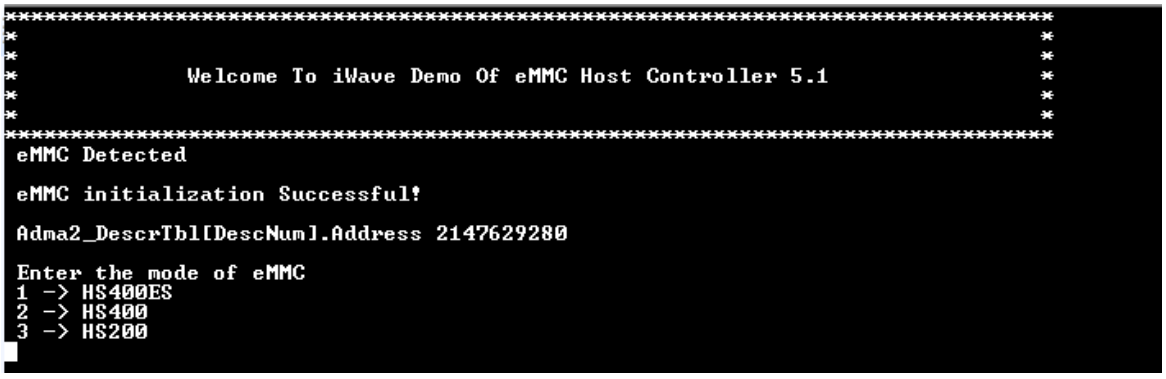

Figure 9 : Initial print after FPGA is programmed

After eMMC initialization is successful user can select the mode to operate i.e HS400_ES, HS400 or HS200 by selecting any one mode user can set the communication in a mode which users wish eMMC communication to take place

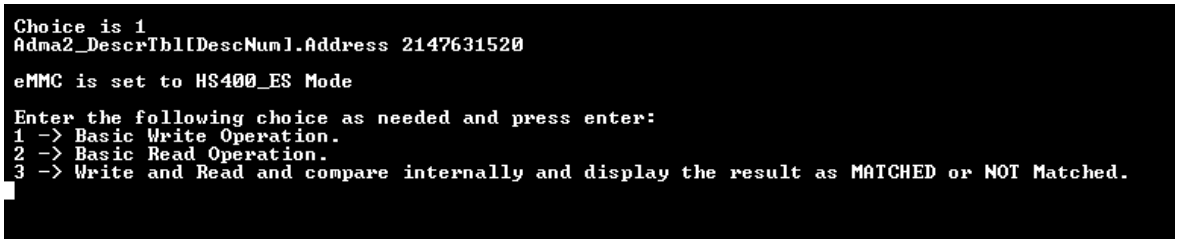

Figure 10 : eMMC is initialized to HS400_ES mode

As shown in the above figure eMMC is initialized to HS400_ES mode and 3 test cases are available as shown in the figure i.e.

1. Basic Write Operation
2. Basic Read Operation

3. Write and read and compare internally and display the result as Matched or Not Matched

Any other test case than the mentioned above will result in wrong choice. As shown in below figure.

```
eMMC is set to HS400_ES Mode

Enter the following choice as needed and press enter:
1 -> Basic Write Operation.
2 -> Basic Read Operation.
3 -> Write and Read and compare internally and display the result as MATCHED or NOT Matched.
Entered Choice : 5

Select a valid option
Enter the following choice as needed and press enter:
1 -> Basic Write Operation.
2 -> Basic Read Operation.
3 -> Write and Read and compare internally and display the result as MATCHED or NOT Matched.
```

Figure 11 : Wrong choice

5.2.2 Basic Write Operation:

Basic write operation is the 1st testcase in the current design. After initial prints, press "1" and then enter the pattern enter "1" for Incremental pattern and enter "2" for Toggling pattern. Once pattern is selected then enter the block count with minimum value of "1" and maximum value of "128" enter the block count and the enter the Start address, once the operation is completed successfully then user get the print "Write Completed."

```
Enter the following choice as needed and press enter:
1 -> Basic Write Operation.
2 -> Basic Read Operation.
3 -> Write and Read and compare internally and display the result as MATCHED or NOT Matched.
Entered Choice : 1

Enter the pattern and press enter:
1 -> Incremental Pattern
2 -> Toggling Pattern

Pattern Entered : 1

Enter the block count value between 1 and 128 inclusive.
Enter 1 for single block write.
Enter value greater than 1 for multiple block write.

Block Count Entered : 127

Address in decimal address
Start Address Entered : 300

Adma2_DescrTbl[DescNum].Address 2147629856
Write Completed


Enter the following choice as needed and press enter:
1 -> Basic Write Operation.
2 -> Basic Read Operation.
3 -> Write and Read and compare internally and display the result as MATCHED or NOT Matched.
```

Figure 12 : Basic Write Operation

5.2.3 Basic Read Operation:

After initial prints, press "2" for basic Read Operation and then enter the block count with minimum value of "1" and maximum value of "128" enter the block count and the enter the Start address from where the read operation need to start, once the read operation is completed successfully then user get the print "Read Completed."

```
Enter the following choice as needed and press enter:
1 -> Basic Write Operation.
2 -> Basic Read Operation.
3 -> Write and Read and compare internally and display the result as MATCHED or NOT Matched.
Entered Choice : 2

Enter the block count value between 1 and 1024 inclusive.
Enter 1 for single block read.
Enter value greater than 1 for multiple block read.

Block Count Entered : 127

Enter the starting address (0 to 2147483647).

Address in decimal
Start Address Entered : 400

Adma2_DescrTbl[DescNum].Address 2147629856

Read Completed


Enter the following choice as needed and press enter:
1 -> Basic Write Operation.
2 -> Basic Read Operation.
3 -> Write and Read and compare internally and display the result as MATCHED or NOT Matched.
```

Figure 13 : Basic Read Operation

5.2.4 Write, Read and Compare operation:

The 3rd test case is Write and read back and compare and display weather it's a match or not match. After initial prints, press "3" and then enter the pattern enter "1" for Incremental pattern and enter "2" for Toggling pattern. Once pattern is selected then enter the block count where the minimum value is "1" and maximum value is "128" enter the block count and then enter the Start address of the write and read operation to start.

```
Enter the following choice as needed and press enter:
1 -> Basic Write Operation.
2 -> Basic Read Operation.
3 -> Write and Read and compare internally and display the result as MATCHED or NOT Matched.
Entered Choice : 3

Enter the pattern and press enter:
1 -> Incremental Pattern
2 -> Toggling Pattern

Pattern Entered : 2

Enter the block count value between 1 and 1024 inclusive.
Enter 1 for single block write.
Enter value greater than 1 for multiple block write.

Block Count Entered : 10

Enter the starting address (0 to 2147483647).

Address in decimal
Start Address Entered : 500

Writing to card.
Adma2_DescrTbl[DescNum].Address 2147631520
Write Completed

Reading from the card.

Adma2_DescrTbl[DescNum].Address 2147631520

MATCHED
Read Completed


Enter the following choice as needed and press enter:
1 -> Basic Write Operation.
2 -> Basic Read Operation.
3 -> Write and Read and compare internally and display the result as MATCHED or NOT Matched.
```

Figure 14 : Write & Read Back and Compare

Once the address is entered the Write to Card Operation will start and once the operation is completed successfully then user get the print “Write Completed,” then Read from Card operation will start and once the read operation is completed successfully then user get the print of the read and write data is Matched or Not Matched and then “Read Completed” print will be observed in the Console as shown in the figure.

6 Resource Utilization

The table below shows the resource utilization summary for KC705 dev. kit. for eMMC Host Controller IP.

Table 2 : Resource Utilization for device for KC705 dev. kit

Resource	Utilization	Available
LUT	3298	203800
FF	3590	407600
BRAM	2	445