SD/SDIO Host Controller 3.0 IP Integration Manual

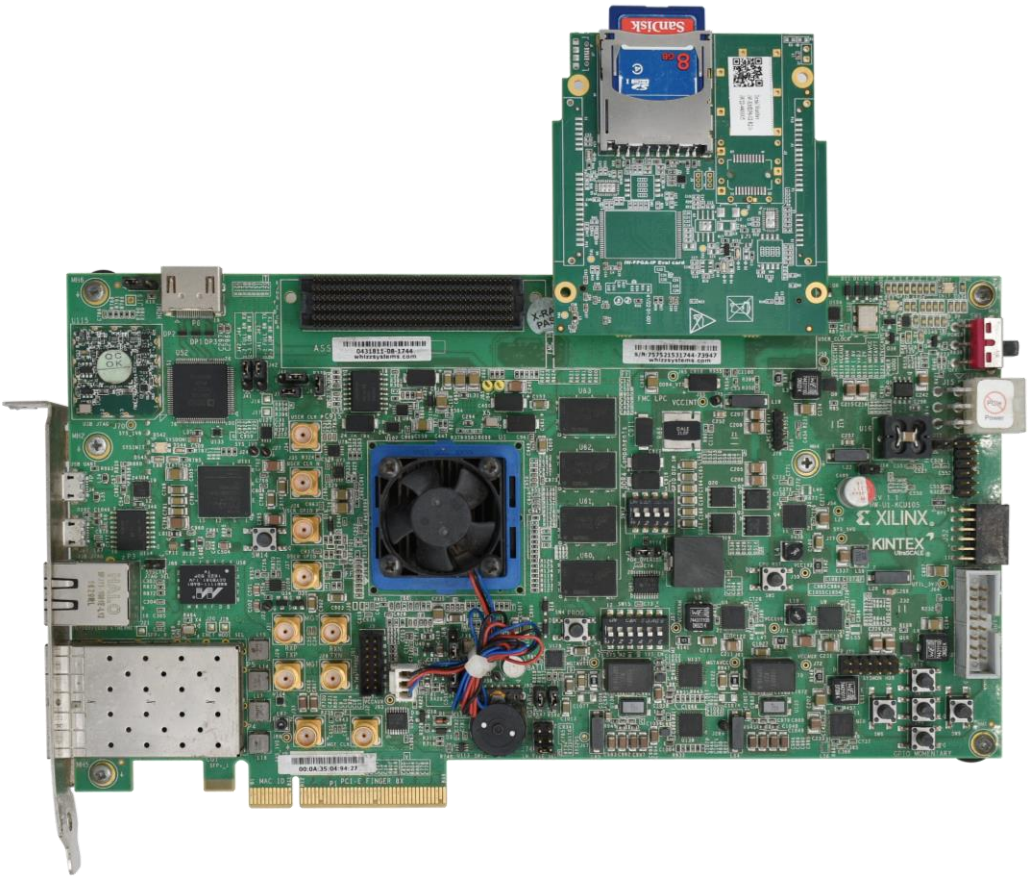

Table of Content

List Of Figures

List Of Tables

1 Introduction

1.1 Purpose

The purpose of this document is to describe SD / SDIO Host Controller 3.0. IP Integration details.

1.2 Reference Document

- IP datasheet

1.3 Overview

SD/SDIO Host Controller interfaces the Microblaze through AXI4 bus and MiG Controller through AXI memory mapped interface, enabling the data transfers between each other. Microblaze IP will send the response to the SD host depending on the command issued and also communicates with the MIG controller for Data read & write.

1.4 Acronyms and Abbreviations

Table 1: Acronyms & Abbreviations

Term	Meaning
FPGA	Field Programmable Gate Array
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 SD/SDIO 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 SD host controller based on the user selection
2. **SD/SDIO Host Controller IP:** SD 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 the SD Car0064

2.2 IP Configuration

The register set, ADMA configuration , tuning block , command and data path files are pre synthesized. After synthesis the post synthesis file i.e., mmc_host.edn file is instantiated in the design as shown in the Figure 1 . The SD Host controller with other files like the mmc_host.edn file, mmc_host stub file, axi4_lite interface and the AXI memory mapped interface wrapper files are added to make a user configured IP.

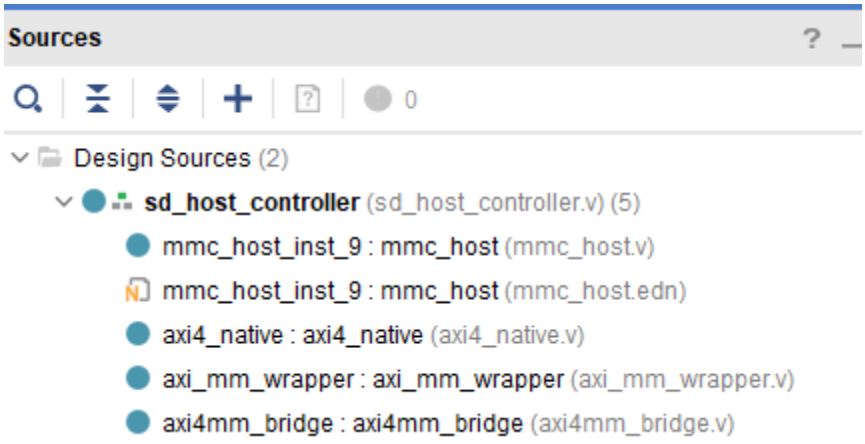

Figure 1:Design sources where SD host netlist is instantiated

Module sd_host_controller is the top module of the project which integrates the all the other files (as shown in the Figure 1) to form a single IP.
Once the IP is generated make sure that the IP needs to be added in the block design of the Vivado project 2020.1.

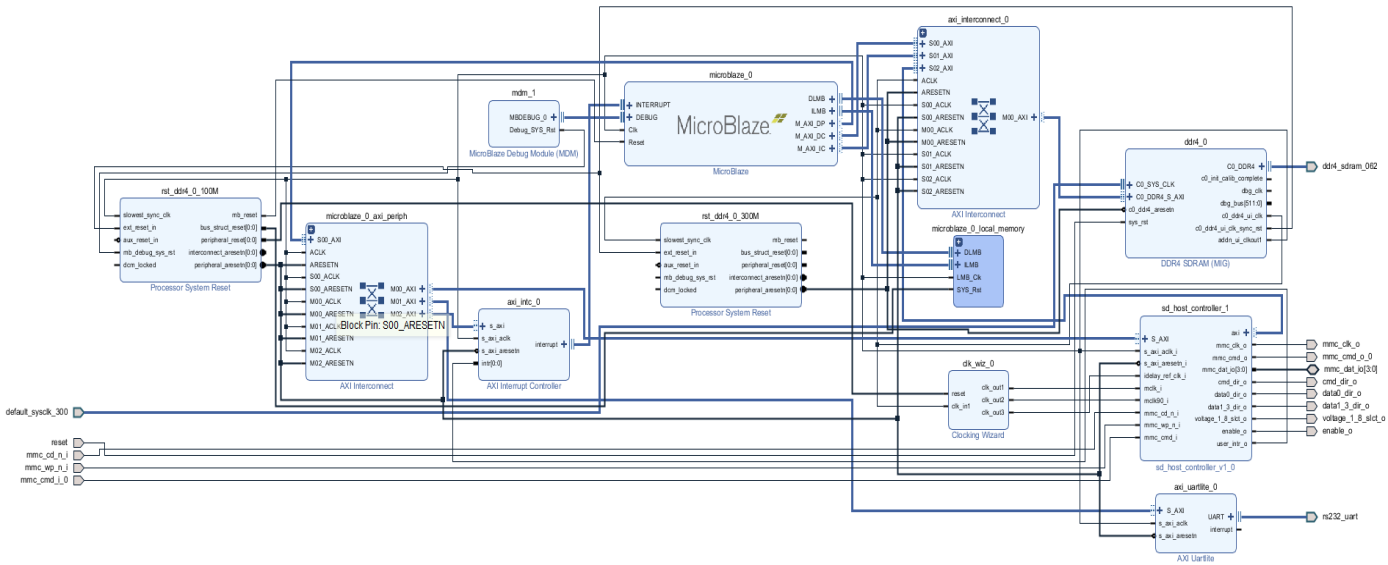

Figure 2:SD_Host Controller added to the block design

2.3 Steps to Instantiate SD HOST IP

- Install the required Vivado Design Suite 2020.1 for the host PC adding the license path. https://www.xilinx.com/support/download/index.html/content/xil%09%09inx/en/downloadNav/vivado-design-tools/archive.html
- Copy the project from "….\EMFC9_Release1.0_SD_3.0_Host\iW-EMFC9-PF-01-R1.0-REL1.0\iW-EMFC9-FF-01-R1.0-REL1.0\iW-EMFC9-ED-01-R1.0-REL1.0\ iW_EMFC9_REL_1.0\project_1" to your project directory in the host PC
- Open the Xilinx Vivado tool 2020.1 and click on "Open Project" and select the required project from the folder by clicking ok
- Now the project selected will be opened in Vivado Design Suite 2020.1
- Go to settings of Vivado 2020.1 and add the IP to the Project Settings → IP → Repository→then add the IP path

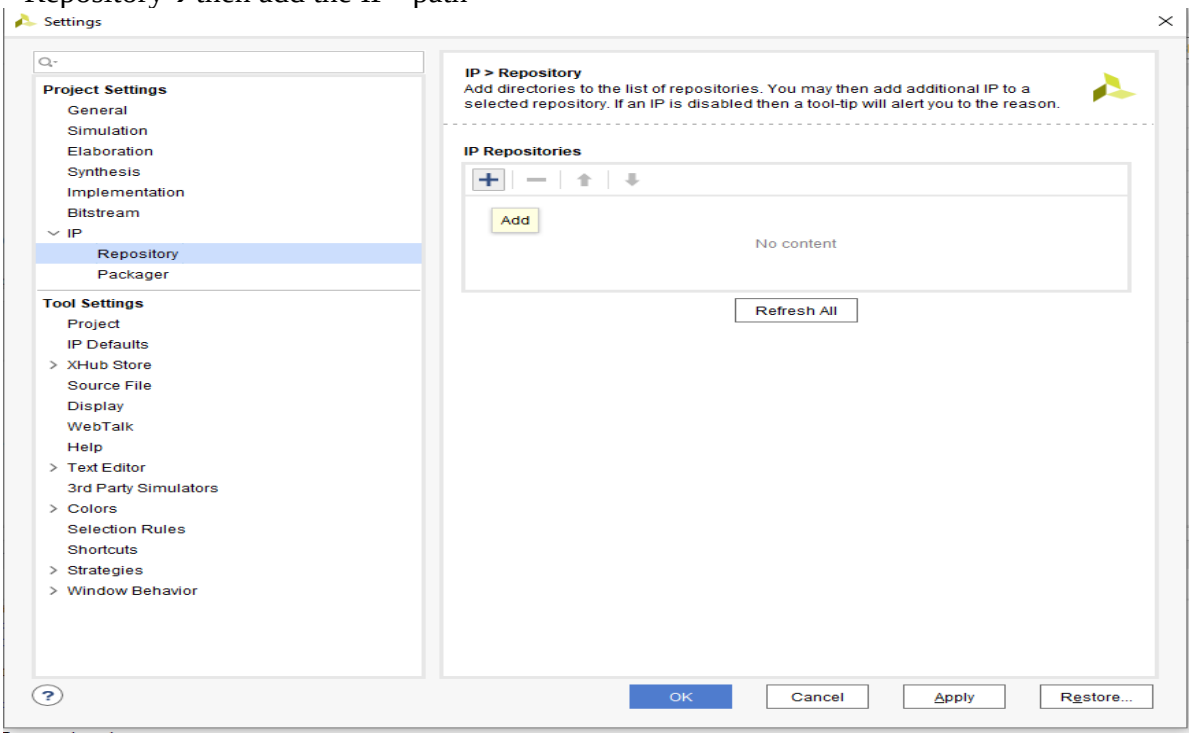

Figure 3:Adding SD_Host_controller IP to the IP repository

IP location is ..\EMFC9_Release1.0_SD_3.0_Host\iW-EMFC9-PF-01-R1.0-REL1.0\iW-EMFC9-FF-01-R1.0-REL1.0\iW-EMFC9-ED-01-R1.0-REL1.0\ iW_EMFC9_REL_1.0\ project_1.srcs \sources_1 \IP .Add IP by clicking on the + sign as shown in Figure 3. 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 by clicking on mb _preset under top_module →mb_preset_wrapper present in the source tab.

1. After adding the other IPs like Microblaze ,UART ,MIG Controller and Clocking Wizard, add the SD host controller IP to the block design as shown in Figure 2 by clicking on the + sign

2. Synthesize the block design first by validating the design(F6) and generate the output products by right clicking on mb_preset under top_module →mb_preset_wrapper present in the source tab and create HDL Wrapper
3. Instantiate mb_preset_wrapper in the top module with the ports and signals of block design, in the top_module.v file as shown in Figure 4

```
mb_preset_wrapper design_inst (
   //DDR3 IO signals
   .reset                  ( !reset              ) ,
   .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 IO signals
   .rs232_uart_rxd         ( rs232_uart_rxd      ) ,
   .rs232_uart_txd         ( rs232_uart_txd      ) ,
   //SD_HOST IO signals
   .enable_o               ( enable_o            ) ,
   .mmc_cd_n_i             ( mmc_cd_n_i          ) ,
   .mmc_clk_o              ( mmc_clk_o           ) ,
   .mmc_dat_io             ( mmc_dat_io          ) ,
   .mmc_wp_n_i             ( mmc_wp_n_i          ) ,
   .cmd_dir_o              ( cmd_dir_o           ) ,
   .cmd_in                 ( cmd_in              ) ,
   .cmd_out                ( cmd_out             ) ,
   .data0_dir_o            ( data0_dir_o         ) ,
   .data1_3_dir_o          ( data1_3_dir_o       ) ,
   .sys_diff_clock_clk_n   ( sys_diff_clock_clk_n ) ,
   .sys_diff_clock_clk_p   ( sys_diff_clock_clk_p ) ,
   .voltage_1_8_slct_o     ( voltage_1_8_slct_o  )
);
```

Figure 4:Instantiation of block design IO signals in the top module

4. The Constraint file (.xdc) provided in the design is for the Xilinx KCU105 Board. It should be changed for custom boards other than Xilinx KCU105 Board
5. Give the required clock, Pin/IO constraints for SD host controller in the .xdc file and compile the custom design with SD host IP

3 Implementation Details

3.1 Clock Domain

SD Host Controller 3.0 IP works on the SDR104 mode and the base clock mclk_i drives the output clock to the SD card. User may need to adjust this clock between 104 MHz to 208 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 which is 200MHz and this clock is phase shifted based on the response from the SD card. Phase Shift of 55° is added.
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 100 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 Clock constraints:

The figure 5 shows the clock constraints used in this design. The constraints given here is the system clock whose frequency is 300Mhz for Xilinx KCU105 board.
This constraint is added in the .xdc file. All other generated clocks will be automatically handled by the Vivado tool since those are derived from the MMCM and will have known relationship. User need to make sure there is no unconstrained clock in the design by checking the clock network report in the Open implemented design from Vivado tool.

```
create_clock -period 3.333 [get_ports default_sysclk_300_clk_p]
```

Figure 5:Clock constraints

3.2.2 False path constraints:

The Figure 6 shows the false path in between all the clocks which is configured for the SD host IP. This details will be available in the .xdc file.

```
set_false_path -from [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/ddr4_0/inst/u_ddr4_infrastructure/gen_mmcme3.u_mmcme_adv_inst/CLKOUT0]]
  -to [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/ddr4_0/inst/u_ddr4_infrastructure/gen_mmcme3.u_mmcme_adv_inst/CLKOUT1]]
set_false_path -from [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/ddr4_0/inst/u_ddr4_infrastructure/gen_mmcme3.u_mmcme_adv_inst/CLKOUT1]]
  -to [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/clk_wiz_0/inst/mmcme3_adv_inst/CLKOUT1]]
set_false_path -from [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/ddr4_0/inst/u_ddr4_infrastructure/gen_mmcme3.u_mmcme_adv_inst/CLKOUT1]]
  -to [get_clocks -of_objects [get_pins mb_preset_wrapper_inst/mb_preset_i/ddr4_0/inst/u_ddr4_infrastructure/gen_mmcme3.u_mmcme_adv_inst/CLKOUT0]]
```

Figure 6:False path constraints

3.2.3 Pin Constraints:

Figure 8 shows the pin constraints in the .xdc file of example design.

```
#kcu105 hpc fmc
set_property PACKAGE_PIN A13 [get_ports {mmc_dat_io[3]}]
set_property PACKAGE_PIN J8 [get_ports {mmc_dat_io[2]}]
set_property PACKAGE_PIN L12 [get_ports {mmc_dat_io[1]}]
set_property PACKAGE_PIN L8 [get_ports {mmc_dat_io[0]}]
set_property PACKAGE_PIN K10 [get_ports mmc_clk_o]
set_property PACKAGE_PIN F8 [get_ports mmc_cmd_io]
set_property PACKAGE_PIN K11 [get_ports data0_dir_o]
set_property PACKAGE_PIN E10 [get_ports data1_3_dir_o]
set_property PACKAGE_PIN L13 [get_ports cmd_dir_o]
set_property PACKAGE_PIN J9 [get_ports voltage_1_8_slct_o]
set_property PACKAGE_PIN D13 [get_ports mmc_cd_n_i]
set_property PACKAGE_PIN C13 [get_ports mmc_wp_n_i]
set_property PACKAGE_PIN H19 [get_ports enable_o]


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_clk_o]
set_property IOSTANDARD LVCMOS18 [get_ports mmc_cmd_io]
set_property IOSTANDARD LVCMOS18 [get_ports data0_dir_o]
set_property IOSTANDARD LVCMOS18 [get_ports data1_3_dir_o]
set_property IOSTANDARD LVCMOS18 [get_ports cmd_dir_o]
set_property IOSTANDARD LVCMOS18 [get_ports voltage_1_8_slct_o]
set_property IOSTANDARD LVCMOS18 [get_ports mmc_cd_n_i]
set_property IOSTANDARD LVCMOS18 [get_ports mmc_wp_n_i]
set_property IOSTANDARD LVCMOS18 [get_ports enable_o]
```

Figure 7:Pin Constraints in example design .xdc file

NOTE: These constraints are in accordance to example design created for the KCU105 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 KCU105 dev kit
- iWave FMC daughter card
- USB cables for JTAG and UART
- SD card for testing
- Power Supply 12 V

NOTE: iWave FMC card requires V_{adj} =1.8V for its operation. Please make sure that V_{adj} =1.8V for iWave FMC daughter card from the user board.

4.2 Connections

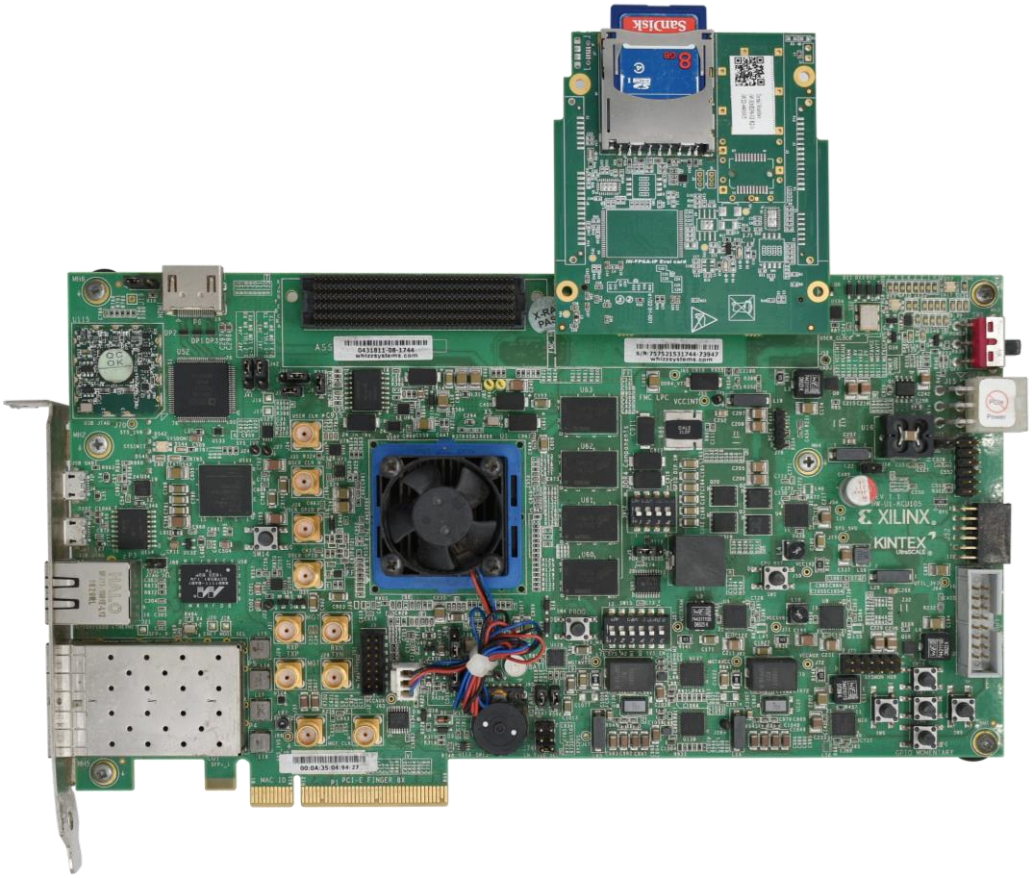

Figure 8:Test Setup of KCU105 board with iW_FMC card

Please make sure that FMC card of SD connector interface is connected to HPC FMC of Xilinx KCU105 board. Connect UART .There will be two serial port available one For system control and other one for FPGA design(Debug_ UART). Select the port for system control and observe the prints as shown in below figure. For example in example design

serial port 10 is for system control and serial port 9 is for FPGA design. System control is used to set the FMC voltage in 1.8V

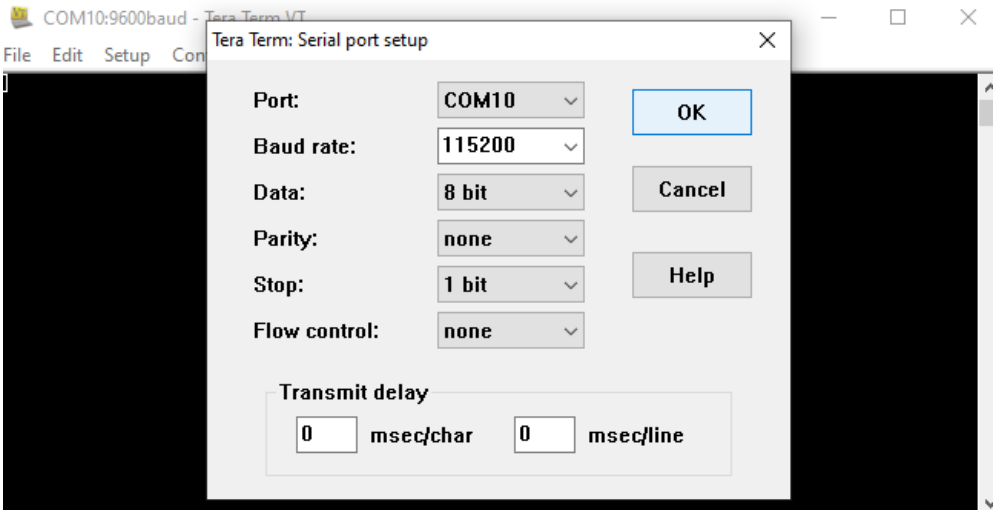

Figure 9:Selecting Serial port to adjust voltage level

After Selecting the serial port check for tera term prints of KCU105 board as shown in figure below.

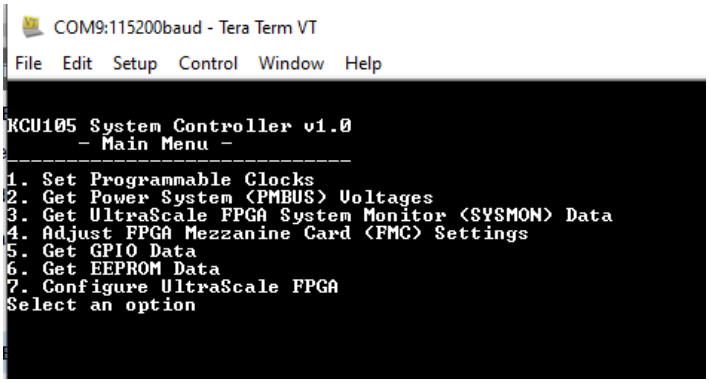

Figure 10:Tera term prints in KCU105 board.

In main menu select option 4 i.e. adjust FPGA FMC settings. Then in FMC menu select Option 4 i.e. Set FMC VADJ to 1.8V. as shown in the figure below.

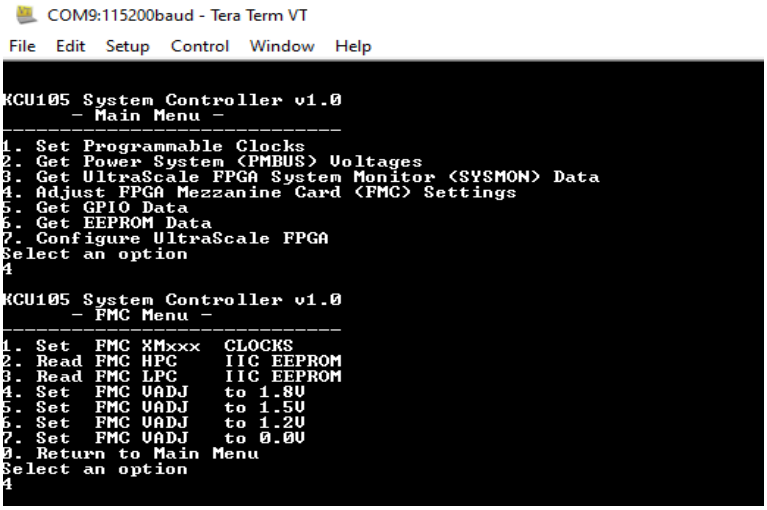

Figure 11:FMC menu in KCU105

Please make sure that SD connector FMC card voltage is set to **1.8V** and 2 Blue LED in FMC card is glowing as shown in the Figure below.

Figure 12:SD Connector FMC

4.3 Programming FPGA and running using Vitis.

- After generating bitstream in Vivado 2020.1 go to File→ Export hardware → select Fixed click next → select include bitstream and click next → give the XSA file name where you want to store your xsa and keep directory as local to project and click Finish
- Create a local folder in the Vivado Project folder and give that as workspace folder for Vitis 2020.1 project and click ok
- After exporting the project go to tools in Vivado and give the path of the local folder created as a workspace folder for Vitis (mentioned in step 2), click on launch Vitis IDE as shown in Figure 11 a new window will appear

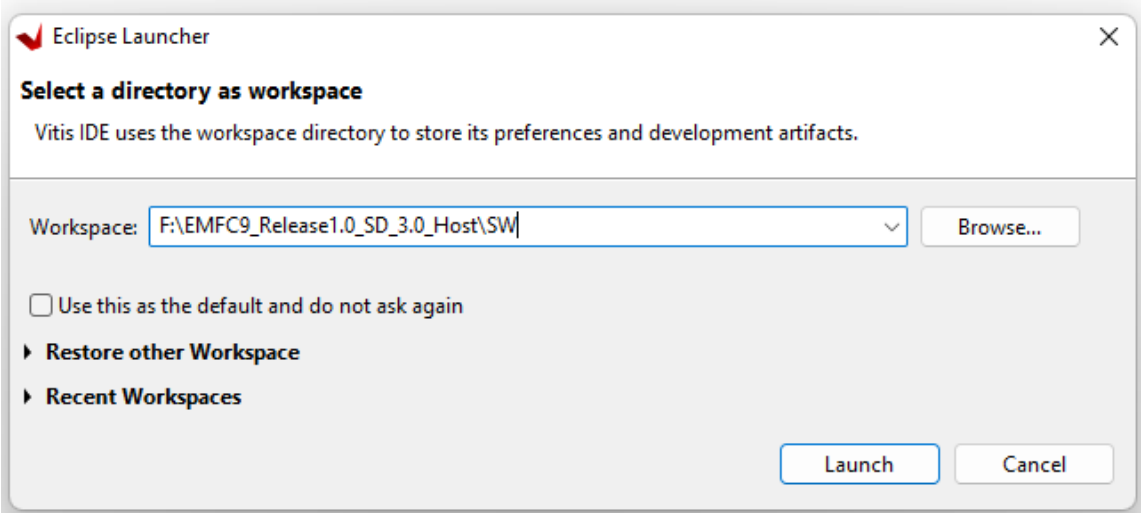

Figure 13:Launching Vitis 2020.1

- Once Vitis 2020.1 is opened then click on Create application project .Then click next, Create a new platform from hardware(XSA)→give the .xsa file path by browsing to that file where you had exported your xsa (project local folder)and click next ,give application project name as shown in the below figure and click next

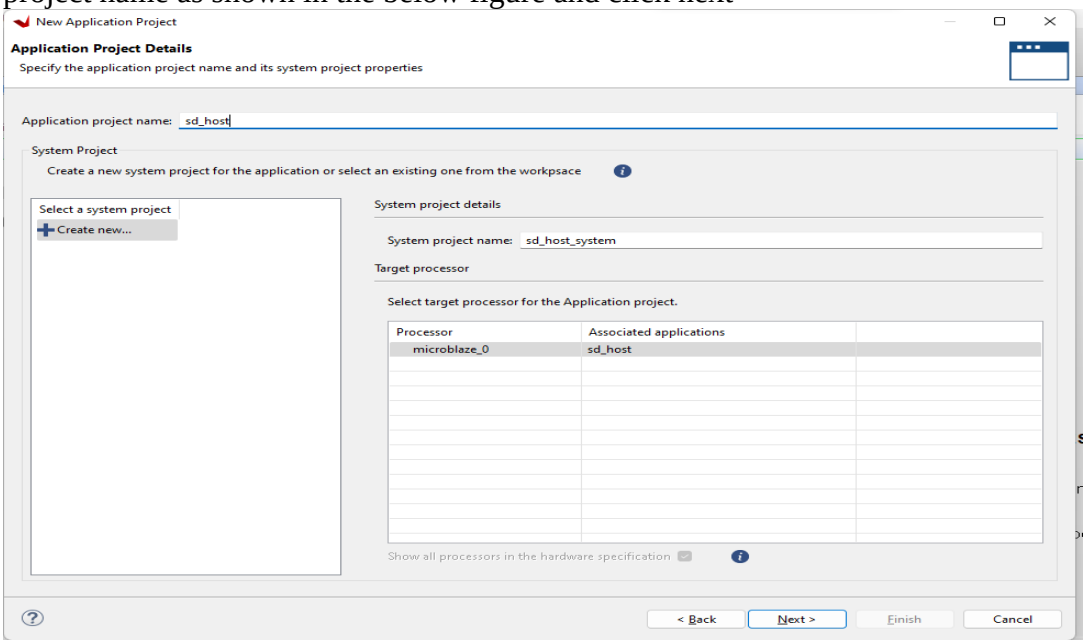

Figure 14:Application project name

- In Available templates click on empty application and click on finish
- Once the project is open, copy the source code from the release folder (…\EMFC9_Release1.0_SD_3.0_Host\iW-EMFC9-PF-01-R1.0-REL1.0\iW-EMFC9-SF-01-R1.0-REL1.0\iW-EMFC9-SC-01-R1.0-REL1.0\SD_HOST_vitis_REL_1.0\SW\source_vitis) to Vitis project src folder and then Save (CTRL+S) and Build(CTRL+B) the project

NOTE: Copy only iwsd_hardware.h, iwsd_host_controller.h, iwsd_host_controller.c and main.c from the release folder to the src folder present under the application project

- Then right click on project go to run and click 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

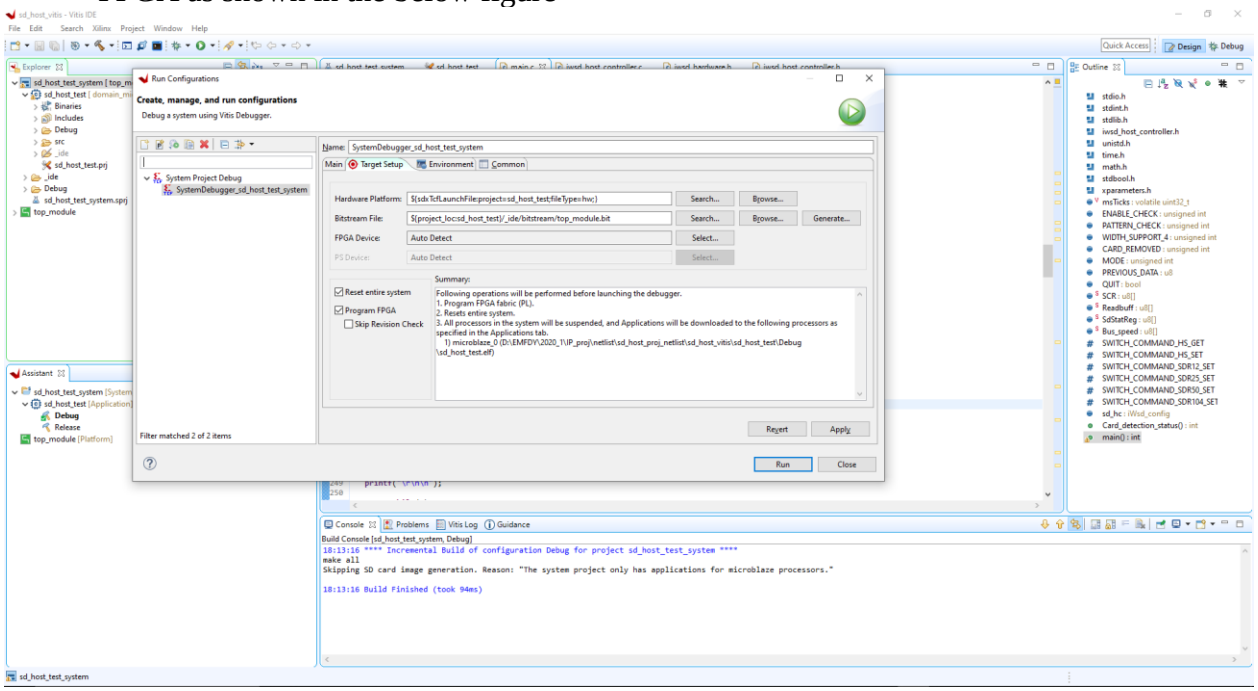

Figure 15:Target setup in run configuration.

- 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 Xilinx KCU105 board is turned on with JTAG connections, 12V power supply and proper tera term setup.
If program FPGA is done using Vivado 2020.1 please don't select program FPGA option in target setup Vitis 2020.1.

5 Test Procedure

5.1 Tera term setup

- Open tera term in PC and select the com port other than COM1 (File→ New connection →Serial)
- Select baud rate as 115200 in setup →serial port → Speed

5.2 Testing Procedure

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

5.2.1 Initial prints:

Make the necessary connection to program the target FPGA board. Connect UART and JTAG along with FMC Card of SD Connector interface to target FPGA board as shown in the Figure 9 and do the Proper tera term setup.
After clicking on Run in Run configuration ,software will program the FPGA. Once the programming is done and .elf file is executed, then below print will be displayed in Tera term. Please wait for some time for the initial prints to be displayed.

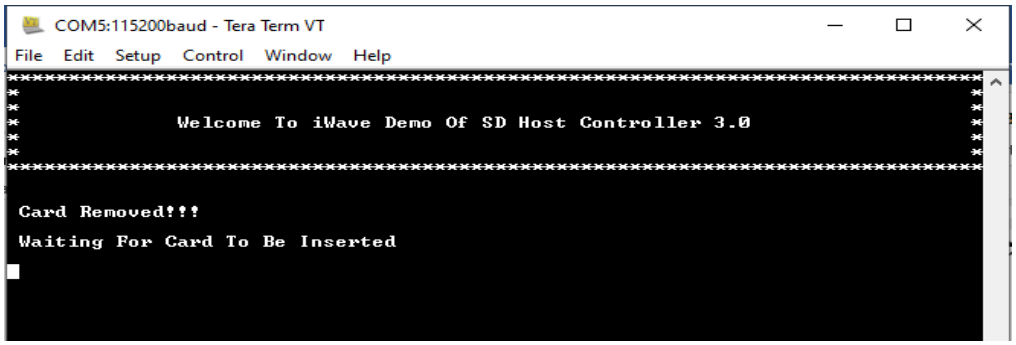

Figure 16:Initial print after FPGA is programmed

As shown in the Figure 13 the application is waiting for the card to be Inserted , so insert the card and then the check that the D5 Red led on the FMC card is glowing which indicates that SD card is detected refer Figure 14, if that is glowing then the following prints will come as shown in Figure 15

Figure 17:Card Inserted to FMC

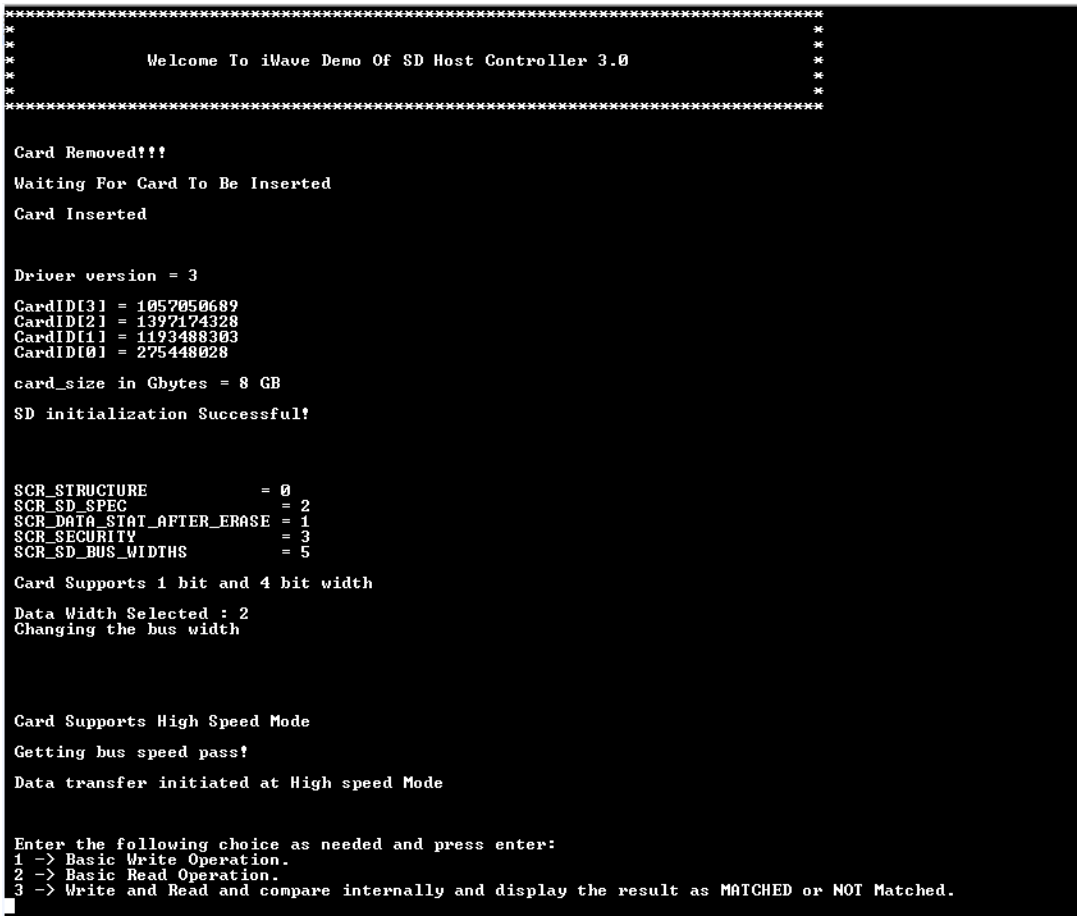

Figure 18:Card Initialization

As shown in the Figure 15 test case is available 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
4. Any other test case than the mentioned above will result in wrong choice i.e.” NOT Matched” as shown in Figure 16

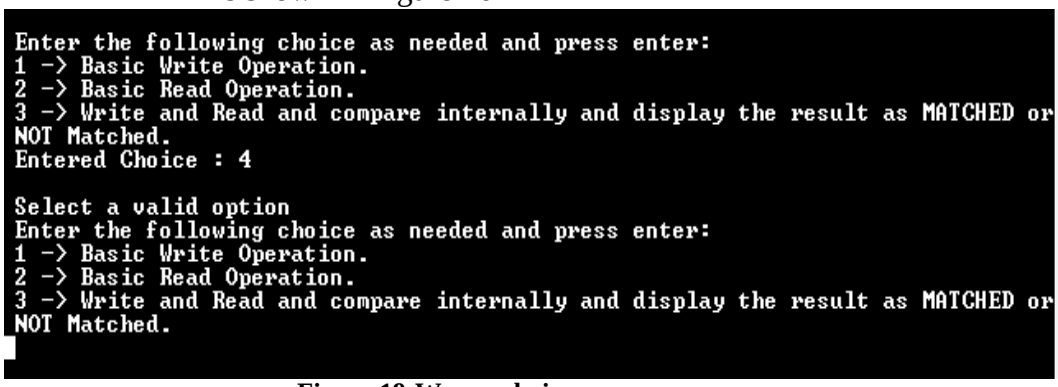

Figure 19:Wrong choice

5.2.2 Basic Write Operation:

- Basic write operation is the 1st testcase in the current design
- Enter the block count with minimum value of "**1**" and maximum value of "**1024**"
- Enter the **Start address** in decimal, once the operation is completed successfully then user gets 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 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 : 512

Address in decimal address
Start Address Entered : 1000

Writing incremental data to card.
Total Block count is: 262144
Write Completed
```

Figure 20: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 "**1024**"
- Enter the block count and the enter the **Start address** from where the read operation needs to be started, 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 : 512

Enter the starting address (0 to 31283199).

Address in decimal
Start Address Entered : 1000

Total Block count is: 262144

Read Completed
```

Figure 21:Basic Read Operation

5.2.4 Write , Read and Compare operation:

- The 3rd test case is Write and read back and compare and display whether it's a match or not match
- Enter the block count where the minimum value is "**1**" and maximum value is "**1024**" enter the block count and then enter the **Start address** of the write and read operation to start

- Once the address is entered the Write to Card Operation will start and once the operation is completed successfully then user gets the print "**Write Completed**," then Read from Card operation will start and print will be obtained same as in write operation
- Write and read data is "**Matched** or **Not Matched**" print will be observed in the Console

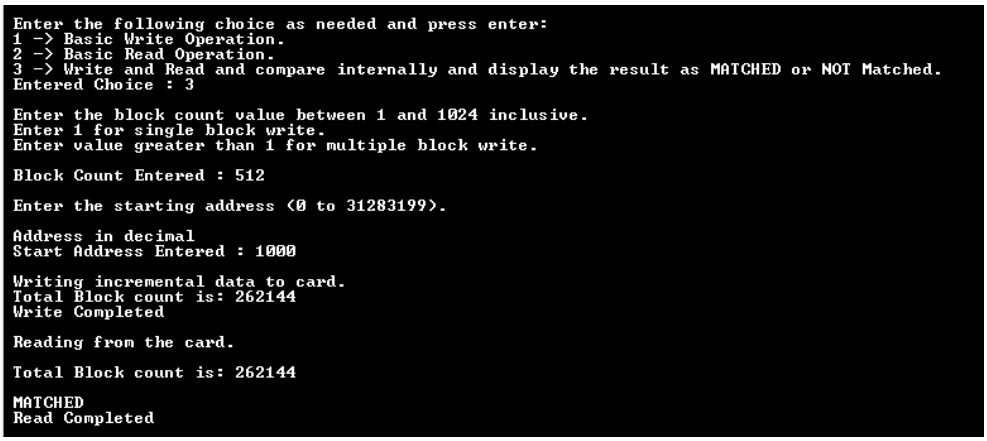

Figure 22:Write & Read Back and Compare

NOTES:

- **The current base clock is running at 300 MHz which is the maximum clock that can be used with KCU105 dev kit**
- **Block count up to 1024 is supported**

6 Design modification to be done for Custom Board

- Update the FPGA part number/board according to the FPGA device used
- Update the complete design for the selected FPGA device
- Update the pin constraints for SD interface, clock, reset and UART pins
- Update the clock constraint according to the input clock frequency for the selected FPGA device
- Recompile the design to generate the new binaries and use the XSA file to create the new application project in Vitis

7 Resource Utilization

The table below shows the resource utilization summary for KCU105 dev. kit. for SD Host Controller IP.

Table 2: Resource Utilization for device for SD Host Controller IP.

Resource	Utilization	Available
LUT	2532	242400
FF	2230	484800
BRAM	0	600