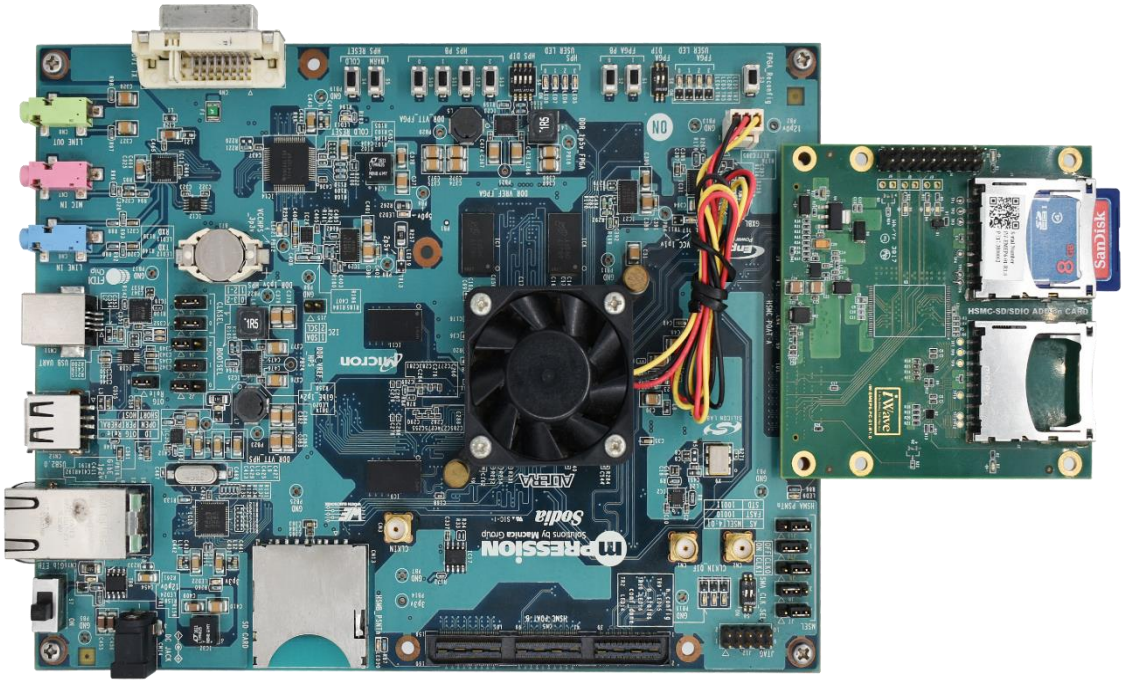
SD/SDIO Host Controller User Manual



\

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The purpose of this document is to help the software engineer to program and test the iWave SD Host controller and this will also guide to configure the Linux development environment in the Host PC and build the board support package.

1.2 Glossary

Term	Meaning
SD	Secure Digital
SDIO	Secure Digital Input Output
SDR50	Standard Data Rate 50
SDHC	Secure Digital High Capacity
SDXC	Secure Digital Extended Capacity
UART	Universal Asynchronous Receiver-Transmitter
UHS1	Ultra High Speed 1
UHS3	Ultra High Speed 3

Table 1: Acronyms & Abbreviations

2 Environment configuration

2.1 HyperTerminal configuration:

- The SD/SDIO Host Controller Test environment using Sodia Development Board and iWave's HSMC Add On card for the evaluation. Where Add-on card consists of both SD connector (J1) and SDIO connector (J2).
- Connect USB Type B cable with host PC and run HyperTerminal/Minicom
- Run a HyperTerminal application on the PC and configure it with the following parameters.

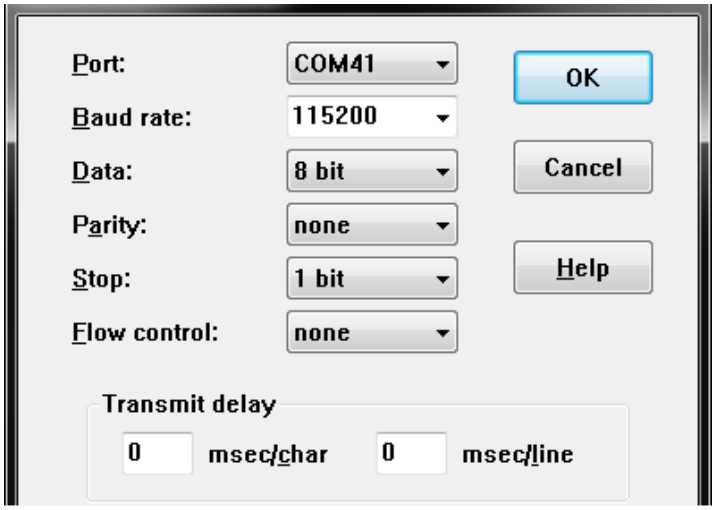


Figure 1:Hyperterminal Configuration

- Power on the board.
- Once the booting is completed, use HyperTerminal/Minicom to access the Linux console.

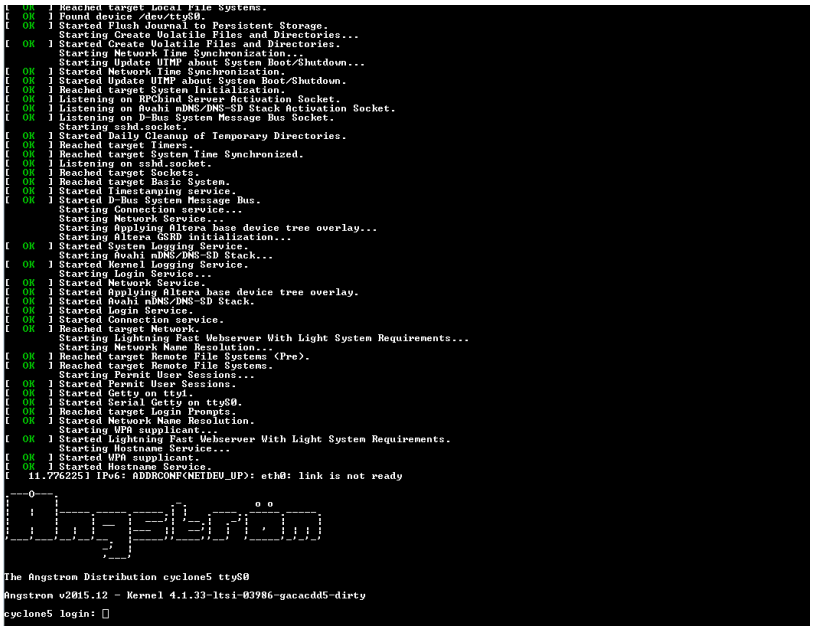


Figure 2: Boot Sequence

- After successful boot we can see the login prompt. Type root and press enter to login.

3 Software Development environment setup

3.1 Host Requirements

- A Linux host PC with latest version (ex. Ubuntu version 14.04)
- Root permission on the Development Host.
- Cross compiler package for Sodia development board.

3.2 Host package installation

- Open a terminal window and install the below packages in host PC.
  sudo apt-get update
  sudo apt-get install sed wget cvs subversion
  sudo apt-get install git-core coreutils unzip
  sudo apt-get install texi2html texinfo
  sudo apt-get install libsdl1.2-dev docbook-utils
  sudo apt-get install gawk python-pysqlite2 diffstat
  sudo apt-get install help2man make gcc vim
  sudo apt-get install build-essential g++ desktop-file-utils
  sudo apt-get install chrpath libgl1-mesa-dev install
  sudo apt-get install libglu1-mesa-dev mercurial
  sudo apt-get install autoconf automake groff libtool xterm

3.3 Standalone Compilation

The following steps will help to build the u-boot source code and linux kernel for Mpression Sodia Evaluation Board.

3.3.1 U-Boot

- Create a directory and open the directory in host to build the uboot
  host@host:$ mkdir <directory_name>; cd <directory_name>

- Un-tar the uboot-socfpga.tar.gz file in to newly created directory.
  host@host:/<directory>$tar -xvzf <Release_folder>/Source_Code/UBoot/uboot-socfpga.tar.gz

- Change the directory to u-boot source code directory.
  host@host:/<directory>$cd uboot-socfpga

- To export the Cross Compiler and tool chain path, execute the below command.
  host@host:/<directory>/uboot- socfpga ~$export PATH=$PATH:/opt/arm-linux-gnueabihf/bin/
  host@host:/<directory>/uboot- socfpga ~$export CROSS_COMPILE=arm-linux-gnueabihf-
  host@host:/<directory>/uboot- socfpga ~$export ARCH=arm

- To Configure for cyclone5 platform, execute the below command.
  host@host:/<directory>/uboot- socfpga ~$make socfpga_cyclone5_config
- To compile the u-boot source code, execute the below command.
  host@host:/<directory>/uboot- socfpga ~$make

- After successful compilation, boot loader image (u-boot.img) and pre-loader raw image will be created in below path.
  <Path to uboot-socfpga>/uboot-socfpga/u-boot.img
  <Path to uboot-socfpga>/uboot- socfpga /spl/u-boot-spl.bin

3.3.2 Pre-Loader

Pre-loader update on the source code is recommended on every changes of FPGA binary/FPGA project only. The pre-loader binary has to be changed in case of pre-loader source code update. The pre-loader raw file will be generated while u-boot compilation. This section describes about the step by step procedure to generate the pre-loader bootable binary.

- Change the directory to uboot compilation directory.
  host@host:/cd <Path to uboot-socfpga>/uboot- socfpga /spl/

- Run the bsp-editor tool from installed EDS Tool.
  host@host:/Path to uboot-socfpga>/uboot-socfpga/spl~$<Path to EDS Tool installed directory>/altera/13.1/embedded/embedded_command_shell. sh

- Embedded command shell will start running. Execute the following command to generate the pre-loader bootable binary.
  host@host:/Path to uboot-socfpga>/uboot-socfpga/spl~$ mkpimage –hv 0 –o preloader-mkpimage.bin u-boot-spl.bin u-boot-spl.bin u-boot-spl.bin u-boot-spl.bin

- Preloader bootable binary will be generated with the name **preloader-mkpimage.bin** in the current directory.

3.3.3 Linux kernel

- Extract toolchain.tar.gz in to the /opt directory.
  host@host/<Directory>~$tar -xvzf <Release_folder>/Software/Binaries/ toolchain.tar.gz -C /opt

- Create a directory and open the directory in host to build the Linux.
  host@host~$ mkdir <directory_name>
  host@host~$ cd <directory_name>

- Extract kernel_src.tar.gz file in to newly created directory.
  host@host/<Directory>~$tar -xvf <Release_folder>/Software/Source_Code/Kernel/linux_socfpga.tar.gz

- Copy the kernel patch file to current directory.
  host@host/<Directory>~$cp <Release_folder>/Source_Code/Kernel/PATCH001-iW-EMEP6-SC-01-R1.0-REL1.0-Linux4.1.33-Ltsi-iWave-SDHC-support.patch .

- Change the directory to Linux source code directory.
  host@host/<Directory>~$cd <path_to_ linux_socfpga > /linux_socfpga

- To apply the iWave SD Host controller patch file, execute the below command.

host@host/<Directory>~$Patch -Np1 < ../PATCH001-iW-EMEP6-SC-01-R1.0-REL1.0-Linux4.1.33-Ltsi-iWave-SDHC-support.patch

- To configure the kernel, execute the below command.
  host@host/<Directory>~$make ARCH=arm CROSS_COMPILE=/opt/arm-linux-gnueabihf/bin/arm-linux-gnueabihf- socfpga_defconfig

- To compile the kernel module drivers and kernel image, execute the below commands.
  host@host/<Directory>~$make ARCH=arm CROSS_COMPILE=/opt/arm-linux-gnueabihf/bin/arm-linux-gnueabihf- zImage

- To compile the dts, execute the below commands.
  host@host/<Directory>~$make ARCH=arm CROSS_COMPILE=/opt/arm-linux-gnueabihf/bin/arm-linux-gnueabihf- dtbs

- After successful compilation, kernel image (zImage) and device tree (.dtb) image will be created in the below path.
  ~/linux_socfpga/arch/arm/boot/zImage
  ~/linux-socfpga/arch/arm/boot/dts/socfpga_cyclone5_sodia.dtb

4 BINARY PROGRAMMING

Default Boot Device is QSPI flash. QSPI flash size is 64MB. Pre-loader, device tree binary, U-boot, Kernel Image, FPGA Image will be situated in QSPI flash.

For booting the platform for the first time, root file system is loaded using SD boot. After file system is successfully booted, complete file system tar is downloaded to the platform. This downloaded file system is flashed to the QSPI as jffs2 using MTD-utils. Further booting of file system will be from QSPI.

4.1 Requirements

To program the binaries to Sodia Development Board's QSPI flash, following Items are required:

- USB Blaster.
- Host PC (Windows) installed with Quartus and EDS Tools.
- Binary files (u-boot.img, zImage, preloader-mkpimage.bin, soc_system.dtb, sodia_ghrd.rbf, rootfs.tar.gz).
- The below figure shows the memory layout of the QSPI flash

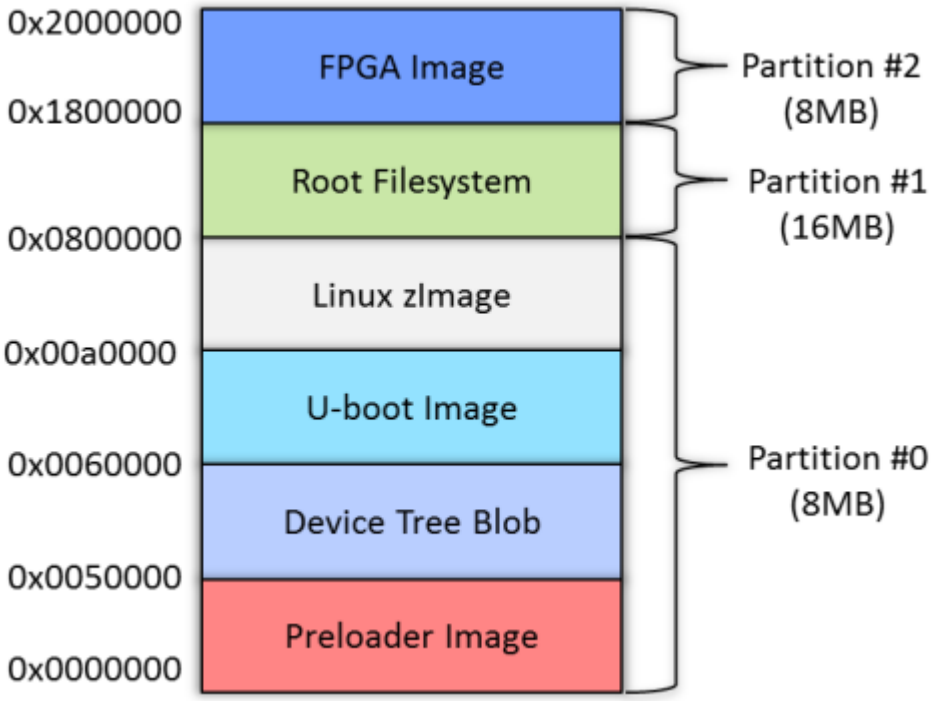


Figure 3: Boot device memory layout

4.2 Programming the Binaries

4.2.1 Programming QSPI flash

- SOC EDS tool is used for flashing QSPI in Windows PC/Linux.
- Place following binaries in a folder and rename the binary files extensions to .bin.

  preloader-mkpimage.bin, soc_system.dtb, u-boot.img, zImage, sodia_ghrd.rbf

  Note: The QSPI flash binaries are located in the below path.

  <Release_folder>/ Binaries/QSPI_Flash
- Connect the USB blaster to Sodia Development board.
- Connect the other end to the development PC USB port.
- Navigate to EDS tool installed directory and run the batch file Embedded_ Command_Shell.sh file
- Navigate to the directory where the binary files are stored.
- Power on the board.
- Execute the following commands to flash the binaries to the QSPI flash.

  <Path to the folder containing binaries>$quartus_hps -c 1 -o P -a 0x000000 preloader-mkpimage.bin

  <Path to the folder containing binaries>$quartus_hps –c 1 –o P –a 0x050000 soc_system.dtb

  <Path to the folder containing binaries>$quartus_hps –c 1 –o P –a 0x060000 u-boot.img

  <Path to the folder containing binaries>$quartus_hps –c 1 –o P –a 0x0a0000 zImage

  <Path to the folder containing binaries>$quartus_hps –c 1 –o P –a 0x01800000 sodia_ghrd.rbf

4.2.2 Downloading and flashing the rootfs file system using SD Boot.

Copy the *<Release_folder>/ Binaries/QSPI_Flash/rootfs.tar.gz* file in the release package to the boot SD Card (Windows Partition).

- Once Linux boot from SD Card is done mount the partition 1 of SD card.

  $ mount /dev/mmcblk0p1 /media
- To erase the QSPI flash block1 enter the following command

  $ flash_erase /dev/mtd1 0 0
- Enter the following command to mount the jffs2 file system in QSPI flash.

  $ mount -t jffs2 /dev/mtdblock1 /mnt/
- Untar the file system to QSPI Flash

  $ tar -xf /media/ rootfs.tar.gz -C /mnt/

Note: Change BSEL to boot from QSPI.

4.2.3 Booting from SD Card.

- Make sure BSEL in Sodia board is in SD Boot mode.
- Download the SD card images form the below link. *https://rocketboards.org/foswiki/pub/Documentation/MacnicaSodiaEvaluationBoard/sodia_sdimage_ACDS16.1_REL_GSRD_PR.tgz?t=1483617444*
- Refer **APPENDIX B : SD card programming** to create the SD image.
- Replace zImage, dtb and rbf files with the one provided in the SD_BOOT Release folder.

5 SD and SDIO Testing

5.1 SDR50 Tests

5.1.1 Card Detection

- Insert SDXC, UHS3 supported/SDHC, UHS1 supported, class 10 SD card in J1 connector.
- Once the card is detected below messages appears on the Linux console (HyperTerminal/Minicom).

Figure 4: Card Detection

5.1.2 File write and read by using the Linux cp command.

- Mount the card using the **mount** command as shown below
  mount /dev/mmcblk1p1 /mnt/
- Use **cp** command to read/write a file from card.
  cd /mnt

  Write:
  cp /home/root/sd.txt /mnt

  Read:
  cp /mnt/sd.txt /home/root/sd1.txt

  Note: sd.txt and sd1.txt are just example files any files can be copied using cp command.
- Check whether the file differs.
  diff /home/root/sd.txt /home/root/sd1.txt
  command should execute without any error messages.

5.1.3 File write and read by using the Linux dd command.

- Use **dd** command to read/write a file from card.
  dd if=/dev/urandom of=/tmp/data bs=1M count=10
  dd if=/tmp/data of=/dev/mmcblk1(p1) bs=1M count=10
  dd if=/dev/mmcblk1(p1) of=/tmp/data1 bs=1M count=10
- Use **md5sum** command to verify that both files match.
  md5sum /tmp/data /tmp/data1

sha values reported by md5sum should be equal for data and data1 files

5.1.4 Throughput check by using the Linux dd command.

- Mount the card using the **mount** command as shown below
  mount /dev/mmcblk1p1 /mnt/
- Use below command to check the read/write throughput.
  Write:
  time(time dd if=/dev/zero of=/mnt/test bs=1M count=512;time sync)

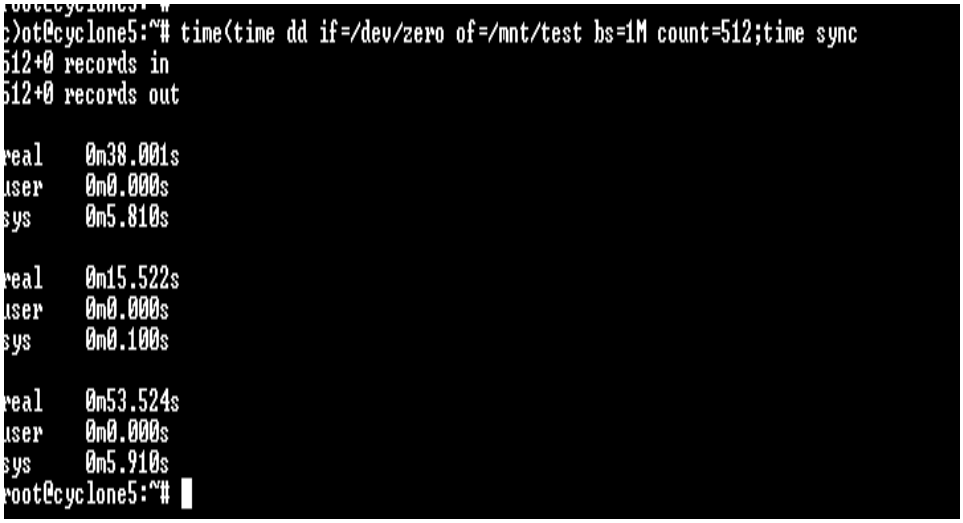


Figure 5: Write throughput using dd command.

Read:
time(time dd if=/mnt/test of=/dev/null bs=1M count=512;time sync)

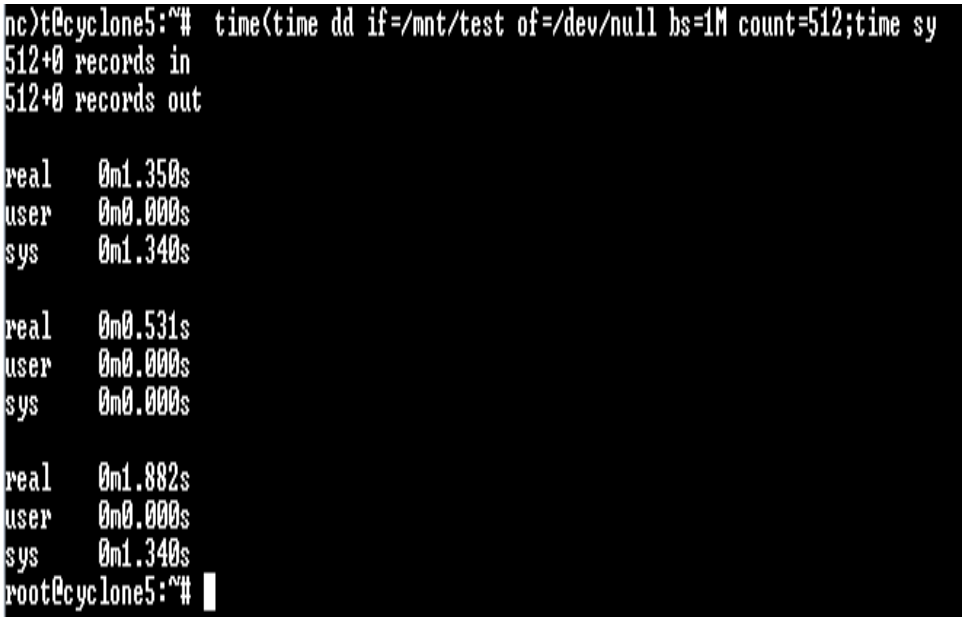


Figure 6:Read throughput using dd command

5.2 SDIO Tests

5.2.1 SDIO Wifi Module Detection and configuration.

- Insert SDIO Wifi module in J1/J2 connector.
- Once the card is detected below messages appears on the Linux console (HyperTerminal/Minicom).
- mmc1:new ultra high speed SDR50 SDIO card at address 0001

Figure 7: SDIO Wifi Module detection

- Insert modules.
  insmod /lib/modules/compat.ko
  insmod /lib/modules/cfg80211.ko
  insmod /lib/modules/wlan.ko

- Start application
  wpa_supplicant -i wlan0 -c wpa_supplicant.conf –B

  Note: Change the .conf file and set ssid as ssid of Wifi hotspot we are going to connect before running above application.
  Example: **wpa_supplicant.conf**

  ctrl_interface=/var/run/wpa_supplicant
  ctrl_interface_group=0
  update_config=1

  network={
  ssid="sdio-wifi-hotspot"
  auth_alg=OPEN
  key_mgmt=NONE
  }

- Set IP on board side
  ifconfig wlan0 192.168.1.50

5.2.2 Ping Test

- Ping the IP of Wifi hotspot created.
  Example:
  ping 192.168.1.10

- If the ping command runs successfully, the Execute Ping results screen displays a brief summary that looks something like this

  64 bytes from 192.168.1.50: seq=5 ttl=64 time=3.384 ms
  64 bytes from 192.168.1.50: seq=6 ttl=64 time=3.378 ms
  64 bytes from 192.168.1.50: seq=7 ttl=64 time=6.769 ms
  64 bytes from 192.168.1.50: seq=8 ttl=64 time=3.039 ms
  64 bytes from 192.168.1.50: seq=9 ttl=64 time=3.135 ms

5.2.3 iPerf Test

- Perform iPerf test.
  Make sure Wifi hotspot is enabled on a Linux PC and run below command in the same.
  host@host/~$: iperf -s

  On the board side run the below command to connect to server and and iperf test to execute
  $ cd /home/root/iperf2.0.9/
  $ iperf -c 192.168.1.50

- If the iperf command runs successfully, the result will display something like this on client side(board side)
  Client connecting to 192.168.1.50, TCP port 5001
  TCP window size: 43.8 KByte (default)
  --
  [3] local 192.168.1.10 port 33301 connected with 192.168.1.50 port 5001
  [ID] Interval Transfer Bandwidth
  [3] 0.0-10.2 sec 18.0 MBytes 14.9 Mbits/sec

6 Latency Check

6.1 Linux Kernel compilation for Latency check

- Create a directory and open the directory in host to build the Linux.
  host@host~$ mkdir <directory_name>
  host@host~$ cd <directory_name>

- Extract kernel_src.tar.gz file in to newly created directory.
  host@host/<Directory>~$tar -xvzf <Release_folder>/Software/Source_Code/LINUX_KERNEL /linux-socfpga.tar.gz

- Copy the kernel patch file to current directory.
  host@host/<Directory>~$cp <Release_folder>/Source_Code/LINUX_KERNEL/ PATCH001-iW-EMEP6-SC-01-R1.0-REL1.0-Linux4.1.33-Ltsi-iWave-SDHC-Latency-support.patch .

- Change the directory to Linux source code directory.
  host@host/<Directory>~$cd <path_to_ linux_socfpga > /linux_socfpga

- Navigate to the linux kernel directory(linux_socfpga) and use below mentioned command to apply patch file to the linux kernel for latency check.
  patch –Np1 < < Path to patch file>
  Example :
  host@host/<Directory>~$Patch -Np1 < ../PATCH001-iW-EMEP6-SC-01-R1.0-REL1.0-Linux4.1.33-Ltsi-iWave-SDHC-Latency-support.patch
- Read and write latency can be calculated by writing and reading from SD card respectively.
  Refer Section 5.1.2 to Read/Write from SD card.

Latency will be calculated using the PIOs which are connected to debug header J4 of the HSMC add on card. It uses total 6 GPIOs. Below table shows the mapping of the same.

Pin in the J4 debug header	Signal mapped
J4.2	Set to 1, when first write to the read buffer of the host controller is done for SD
J4.4	Set to 1, when first read from the write buffer of the host controller is done for SD
J4.6	Set to 1, when first write to the read buffer of the host controller is done for SDIO
J4.8	Set to 1, when first read from the write buffer of the host controller is done for SDIO
J4.10	Set to 1, when read command from driver and reset to 0, when the read data available interrupt is received by driver
J4.12	Set to 1, when driver sends the write command
J4.19, J4.20	Ground

Write latency: Write latency is calculated from the driver sending the write command to the first data read by the card from write buffer of the host controller.

For example:

Figure 8: Write latency

This latency is around 4.55ms.

Read Latency: Read latency is calculated from driver sending the read command to the driver receiving the read data available interrupt.

For example:

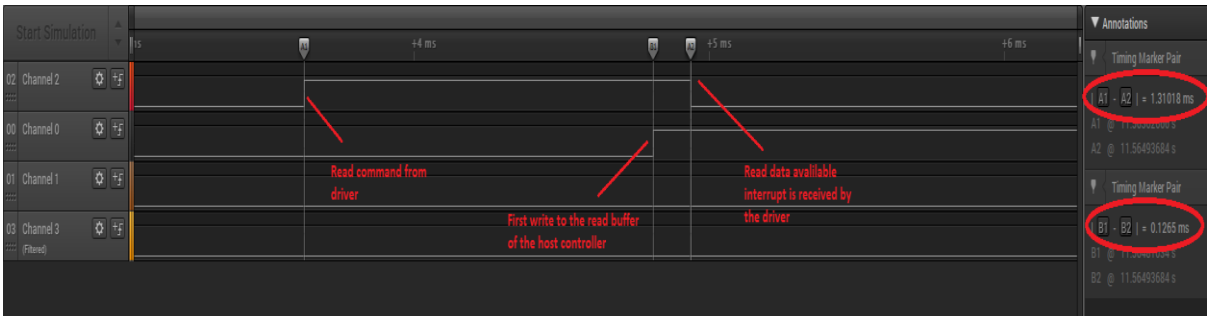


Figure 9: Read latency

The delay between the read command from driver to first write to the read buffer of the host controller is 1.18368ms and the delay between first write to the read buffer of the host controller to the read data available interrupt to the driver is 0.1265ms.

This latency is around 1.31ms.

7 APPENDIX A – EDS tool Installation

- The pre-loader can be generated using EDS tool. This section describes step by step procedure to install the EDS tool. The EDS tool can be downloaded from the below link.

  http://dl.altera.com/soceds/?direct_download=1&version_number=13.1&description=SoC+Embedded+Design+Suite+%28EDS%29&download_method=download&download_manager=direct&edition=standard&platform=linux&direct_file=SoCEDSSetup-17.1.0.590&filesize=1268787542

  For more information on EDS tool refer below link.

  http://dl.altera.com/soceds/?platform=linux&download_manager=direct

  Note : Here SoCEDSSetup-17.1.0.590.run used.

- Enter into EDS tool downloaded directory and change to executable.

  host@host:$cd <EDS tool downloaded directory> ; chmod +x SoCEDSSetup-17.1.0.590.run

- Execute the below command to install the EDS tool.

  host@host:$./SoCEDSSetup-17.1.0.590.run

8 APPENDIX B : SD card programming

This section explains how to create the SD card necessary to boot Linux, using the SD card image available with the precompiled Linux binaries package.

Creating SD Card on Linux

The required steps are:

- Copy SD card image file from the release package to PC:

  $ tar -xzf sodia_sdimage_ACDS16.1_REL_GSRD_PR.tgz
- This will create the file sdimage.img which contains the SD card image file.
- Determine the device associated with the SD card on the host by running the following command before and after inserting the card in the reader:

  $ cat /proc/partitions
- Let's assume it is /dev/sdx.
- Use dd utility to write the SD image to the SD card:

  $ sudo dd if=sodia_sdimage_ACDS16.1_REL_GSRD_PR.img of=/dev/sdx bs=1M
- Use sync utility to flush the changes to the SD card:

  $ sudo sync

Creating SD Card on Windows

This section explains how to write the SD image to the SD card on a Windows PC.

- Copy SD card image file (*sodia_sdimage_ACDS16.1_REL_GSRD_PR.tgz*) to the Host PC.
- Extract the SD card image from the compressed archive *sodia_sdimage_ACDS16.1_REL_GSRD_PR.tgz* using WinZip or similar tools. This creates the file *sodia_sdimage_ACDS16.1_REL_GSRD_PR.img*.
- UseWin32DiskImager to write the image to the SD card. The tool can be downloaded from here:
- https://sourceforge.net/projects/win32diskimager/files/latest/download
- Browse *sodia_sdimage_ACDS16.1_REL_GSRD_PR.img* file and click on Write button to program the SD card.

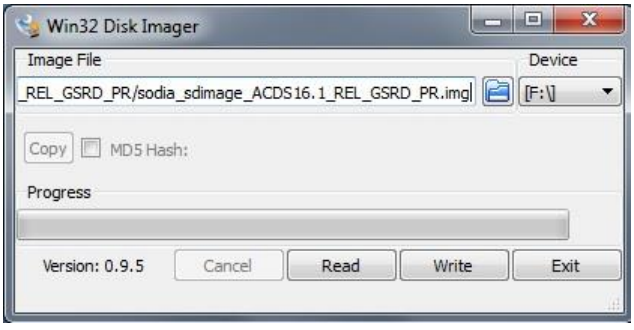


Figure 10: SD programming (Windows)