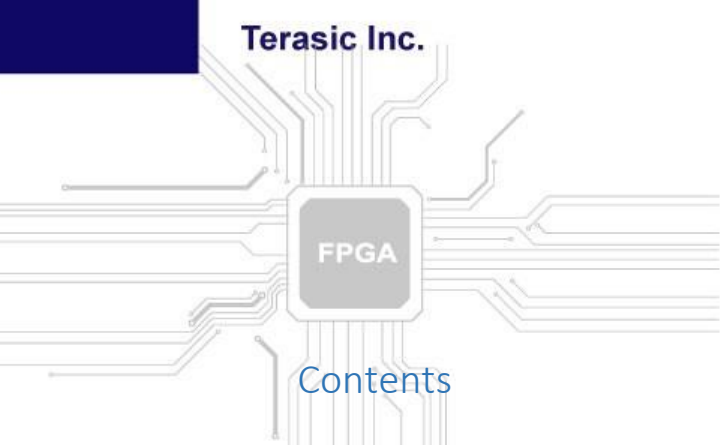
terasIC
www.terasic.com

DE25-Standard Development Kit

Powering next-level performance for digital logic and embedded applications !

Demonstration Manual



Contents

Chapter 1

Overview

This Manual will introduce the various application demonstrations on **DE25-Standard Development Kit**. These demonstrations cover most of the interfaces on the board. Let users familiarize using these interfaces of the board. Demonstrations according to FPGA fabrics and HPS are divided into two categories:

- **Pure use of FPGA fabric resources (Chapter 2)**
- **Pure use of HPS fabric resources (Chapter 3)**

Finally, to complete the following demonstration, user needs to install the following software in the computer:

- **Intel Quartus® Prime Pro Edition Software Version 24.2 or later.**
- **Intel SoC Embedded Design Suite(EDS) Professional Edition**

Note: To run the demo bath file with the Nios V CPU of the demonstration on windows system, user need to install the Windows Subsystem for Linux (WSL) first then you can run the batch file. Please refer to the link to install : Getting Start Install WSL

Chapter 2

Examples For FPGA

This chapter provides examples of advanced designs implemented by RTL or Qsys on the **DE25-Standard Development Kit**. These reference designs cover the features of peripherals connected to the FPGA, such as DDR4, temperature monitor, PLL clock setting and Power monitor. All the associated files can be found in the directory **\Demonstrations\FPGA** of DE25-Standard System CD.

2.1 Basic Nios V control demo for Temperature/ Power/ Fan

This demonstration shows how to use the Nios V processor to measure the power consumption based on the built-in power measure circuit. The demonstration also includes a function of monitoring system temperature with the on-board temperature sensor and monitoring fan rotation speed.

■ System Block Diagram

Figure 2-1 shows the system block diagram of this demonstration. The 12V input power monitor, temperature sensor and fan controller connected to the system MAX10 FPGA and controlled by internal logic circuits. All collected status data or control commands will be sent to the SPI slave block so that the Agilex FPGA can read it through the SPI interface.

In the Agilex FPGA, an SPI master IP (implemented by HDL) will read these external sensor data from the MAX10 FPGA through SPI interface. The Nios system will read these information through PIO controllers.

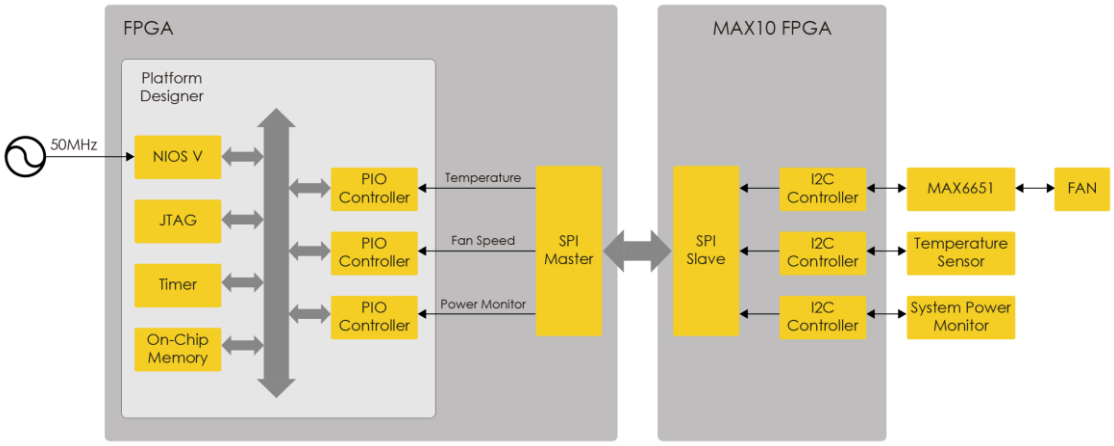


Figure 2-1 Block Diagram of the Nios V Basic Demonstration

The system provides a menu in commnad line windows, as shown in **Figure 2-2** to provide an interactive interface. With the menu, users can perform the test for the board info sensor. Note, pressing 'ENTER' should be followed with the choice number.

```
Loading section .entry, size 0x20 lma 0x0
Loading section .exceptions, size 0x29c lma 0x20
Loading section .text, size 0x21184 lma 0x2bc
Loading section .rodata, size 0x1530 lma 0x21440
Loading section .rwdata, size 0x1980 lma 0x242f0
Start address 0x0000061c, load size 148208
Transfer rate: 63 KB/sec, 11400 bytes/write.
[Inferior 1 (Remote target) detached]
INFO: Sending resume to OpenOCD.
======= DE25-Standard Demo Program =======
[0] Display Board Info
Input your chioce:0
==== Temperature ====
       FPGA: 39*C
       Board 1: 32*C
       Board 2: 37*C

==== Fan ====
       Fan RPM: 2070

==== Power (12V) Monitor ====
       Voltage       = 12.249 V
Display Board Info Test:PASS
======= DE25-Standard Demo Program =======
[0] Display Board Info
Input your chioce:
```

Figure 2-2 Menu of Demo Program

In board info test, the program will display local temperature, remote temperature, 12V

input power monitor and fan rotation speed. The remote temperature is the FPGA temperature, and the local temperature is the board temperature where the temperature sensor located. The board also provides circuitry to monitor important voltages, currents and power consumption in real time.

■ Demonstration File Location

- Hardware project directory: Board_Info_NiosV
- Bitstream used: golden_top.sof and Board_Info.elf
- Software project directory: Board_Info\software
- Demo batch file: Board_Info\demo_batch\test.bat

■ Install Ashling RiscFree IDE

Before executing this demo with RISC-V core, users need to install Ashling RiscFree IDE for Intel® FPGAs to ensure that this batch file can be executed correctly. The file name should be *RiscFreeSetup-<Quartus Version>-windows.exe.* Users can find it under the Quartus Pro download page (Individual Files tab).

■ Demonstration Setup and Instructions

1. Make sure Quartus Prime is installed on the Host PC.
2. Use the USB Cable to connect your PC and the FPGA board and install USB Blaster II driver if necessary.
3. Power on the FPGA board.
4. Execute the demo batch file “test.bat” under the batch file folder: Board_Info _NiosV\demo_batch.
5. After the Nios V program is downloaded and executed successfully, a prompt message will be displayed in command line window.
6. For temperature, power monitor and fan test, please input key ‘0’ and press ‘Enter’ in the nios-terminal, as shown in **Figure 2-3**.

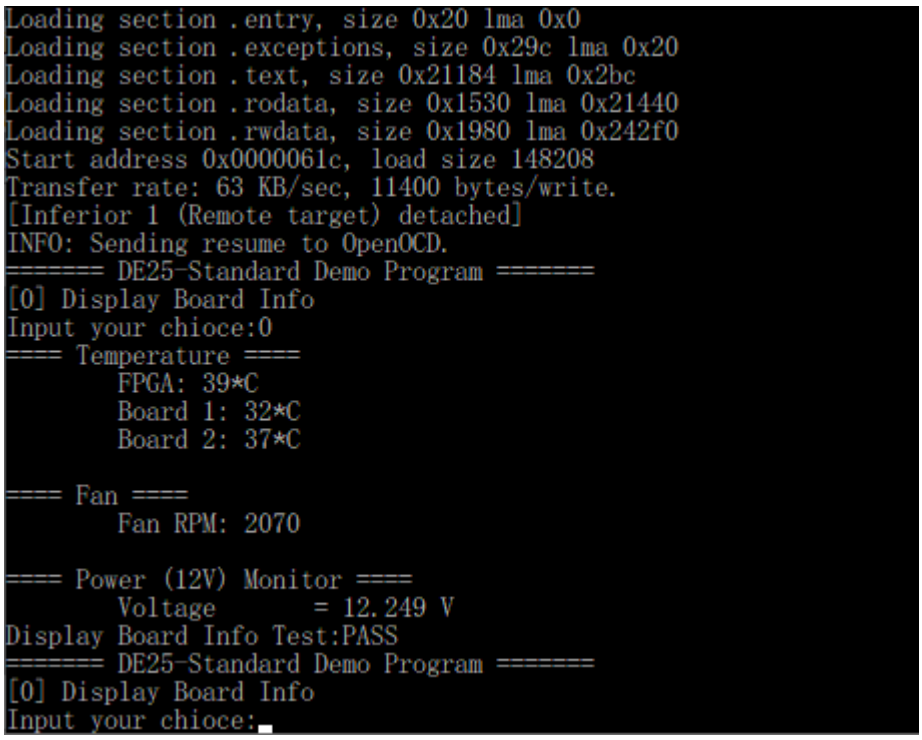


Figure 2-3 Board Info Demo

2.2 Board Information IP

This section will introduce an IP which can be placed in the Agilex FPGA and allows users to obtain board status information such as power, temperature status and fan speed on the DE25-Standard board.

The DE25-Standard board provides several sensors to monitor the status of the board, such as FPGA temperature, board power monitor, and fan speed status. These interfaces are connected to the system MAX FPGA on the board. The logic in the system MAX FPGA will automatically read the status values of these sensors and store them in the internal register. As shown in **Figure 2-4**, there is an SPI slave IP in the system MAX FPGA will read the value of the board status from these registers and it can be output to SPI master logic via SPI interface.

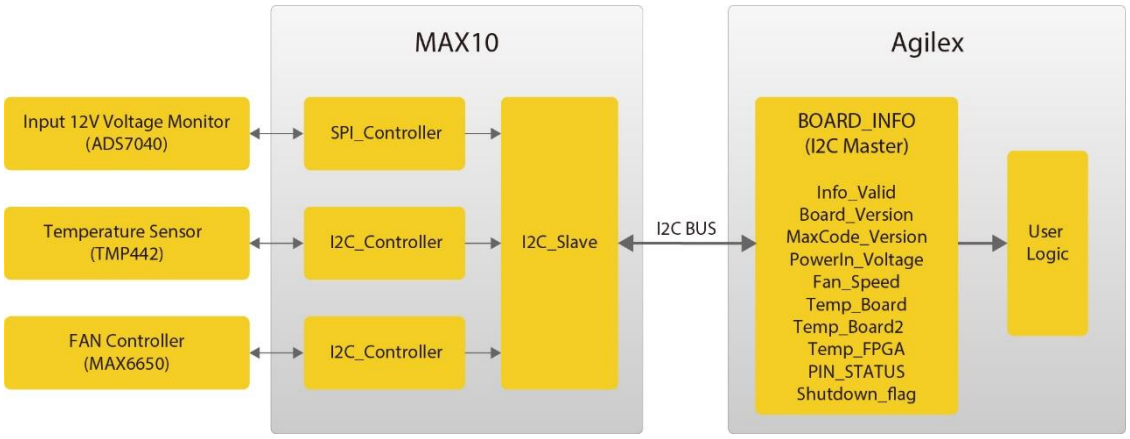


Figure 2-4 Block diagram of the fan speed control demonstration

User can placing a board information IP (BOARD_INFO.v ; I2C master) provided by Terasic in the Agilex FPGA, the board status can be obtained via I2C interface from the system MAX FPGA and output to user logic.

The board information IP can be obtained from the following path in the system CD: ***Demonstration/FPGA/Board_info_RTL/board_information_ip/BOARD_INFO.v***

Figure 2-5 shows the input and output pins of the board information IP. Detailed pin descriptions and functions can be obtained from **Table 2-1** Board information IP input and output ports. The user only needs to provide the IP 50Mhz clock and the reset control signal. The IP will automatically communicate with the system MAX FPGA to get the boar status value via the SPI interface. When the logic level of the Info_Valid signal is from low to high, it means that the board status has been updated and can be used.

Finally, **Figure 2-6** shows the status of the IP during execution.

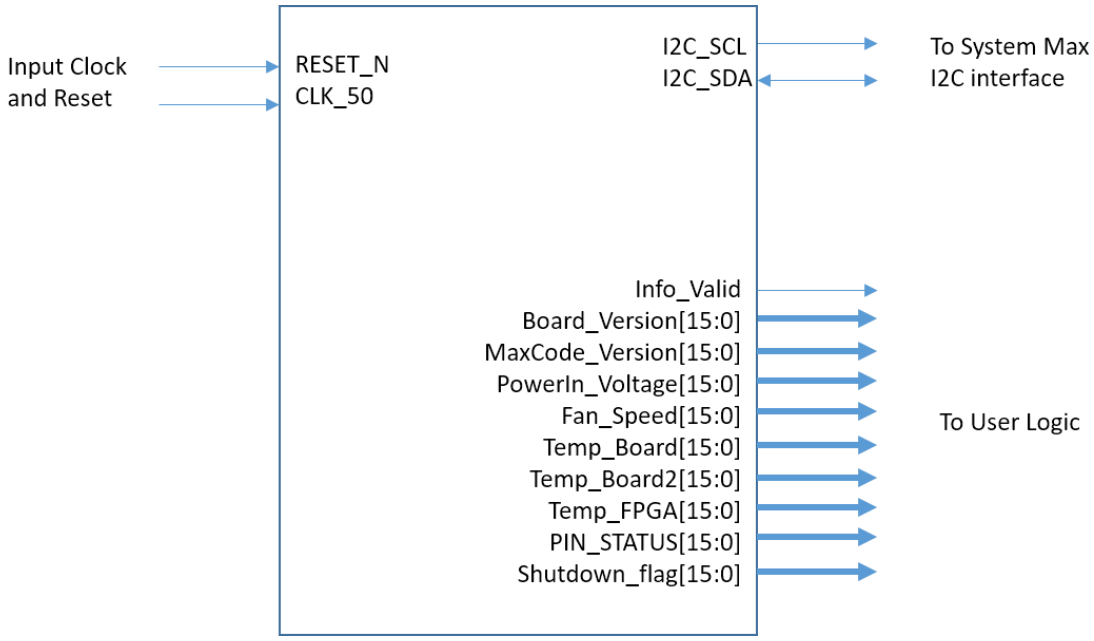


Figure 2-5 Pin out of the board information IP

Table 2-1 Board information IP input and output ports

Port Name	Direction	Width(Bit)	Description
CLK_50	Input	1	Clock input for IP, please input 50Mhz clock.
RESET_N	Input	1	Reset signal for IP, reset all logic.
I2C_SDA	BIR	1	Master I2C data . Please connect this signal to the **FPGA_I2C_SDAT**pin.
I2C_SCL	Output	1	Master I2C clock, I2C master output to salve. Please connect this signal to the **FPGA_I2C_SCLK** pin.
Info_Valid	Output	1	Information valid, logic high indicates board status updated ready.
Board_Version	Output	16	This information indicates the version of the DE25-STANDARD board. It will be started at 0x000A.
MaxCode_Version	Output	16	This information indicates the version of the System MAX 10 FPGA code. It will be started at 0x0001.
PowerIn_Voltage	Output	16	12V Voltage, the unit of the output value is mV. If the **PowerIn_Voltage** output value is “12050” that means 12.05V for 12V power

Fan_Speed	Output	16	First fan speed of the board. The unit of the output value is RPM.
Temp_Board	Output	16	First ambient temperature of the development board. The unit of the output value is Celsius.
Temp_Board2	Output	16	Second ambient temperature of the development board. The unit of the output value is Celsius.
Temp_FPGA	Output	16	Core FPGA temperature of the development board. The unit of the output value is Celsius.
Shutdown_flag	Output	16	BIT3~15 : Reserved to 0. BIT2: 1 ,when FPGA Temperature >=95˚C BIT1: 1 ,when Board2 Temperature >=95˚C BIT0: 1 ,when Board Temperature >=95˚C
PIN_STATUS	Output	16	BIT8~15 : Reserved to 0. BIT7: **FAN_ALERT_n**, When the fan speed is abnormal, this bit is 0. BIT6: Reserved to 1. BIT5: When shutdown occurs, this bit is 0. BIT4: Reserved to 1. BIT3: Reserved to 0. BIT2: **FPGA_CONF_DONE**,FPGA Configure success, this bit is 1. BIT1: Reserved to 1. BIT0: Reserved to 1.

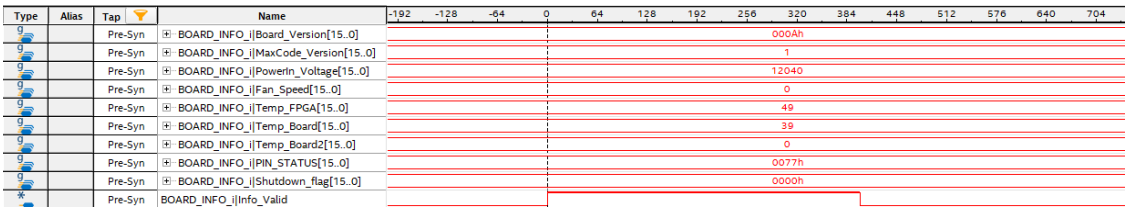


Figure 2-6 Waveform of the board status output

2.3 SDRAM Test in Verilog

DE25-Standard system CD offers another SDRAM test with its test code written in Verilog HDL. The memory size of the SDRAM bank tested is still 64MB.

■ System Block Diagram

Figure 2-7 shows the function block diagram of this demonstration. The SDRAM controller uses 50 MHz as a reference clock and generates 100 MHz as the memory clock.

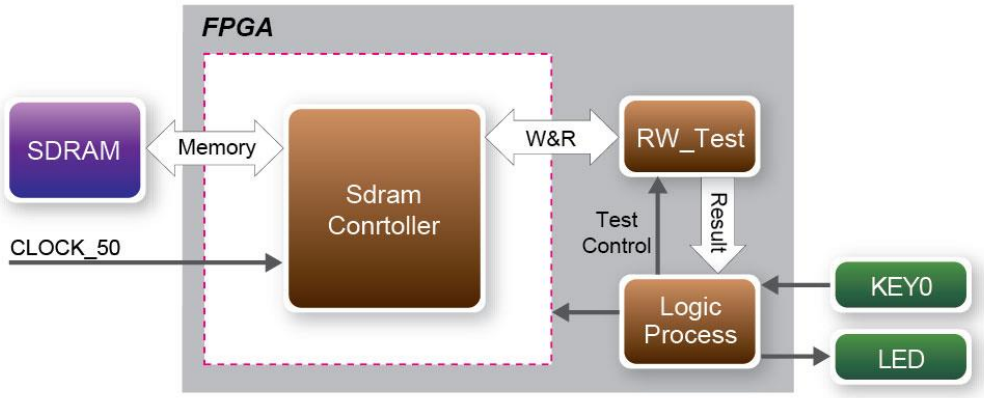


Figure 2-7 Block Diagram of the SDRAM test in Verilog

RW_test module writes the entire memory with a test sequence first before comparing the data read back with the regenerated test sequence, which is same as the data written to the memory. KEY0 triggers test control signals for the SDRAM, and the LEDs will indicate the test result according to **Table 2-2**.

■ Design Tools

- Quartus Prime 24.2 Pro Edition

■ Demonstration Source Code

- Quartus Project directory: DRAM_RTL_Test
- Bitstream used: golden_top.sof

■ Demonstration Batch File

Demo Batch File Folder: DRAM_RTL_Test\demo_batch

The demo batch file includes following files:

- Demo Batch File : test.bat
- FPGA Configure File: golden_top.sof

■ Demonstration Setup and Instructions

- Please make sure both Quartus II and USB-Blaster II driver are installed on the host PC.
- Power on the board.
- Execute the demo batch file " test.bat" from the directoy DRAM_RTL_Test\demo_batch.
- Press KEY0 on the board to start the verification process. When KEY0 is pressed, the LED [2:0] should turn on. When KEY0 is then released, LED1 and LED22 should start blinking.
- After approximately 8 seconds, LED1 should stop blinking and stay ON to indicate the test is PASS. **Table 2-2** lists the status of LED indicators.
- If LED1 is not blinking, it means 50MHz clock source is not working.
- If LED1 failed to remain ON after approximately 8 seconds, the SDRAM test is NG.
- Press KEY0 again to repeat the SDRAM test.

Table 2-2 Status of LED Indicators

Name	Description
LED0	Reset
LED1	ON if the test is PASS after releasing KEY0
LED2	Blinks

2.4 DDR4 SDRAM Test by Nios V

Many applications use a high performance RAM, such as a DDR4 SDRAM, to provide temporary storage. In this demonstration hardware and software designs are provided to illustrate how to perform DDR4 memory access in the Platform Designer (formerly Qsys). We describe how the memory controller Agilex External Memory Interfaces is used to access the on-board DDR4 SDRAM on the FPGA board, and how the Nios V

processor is used to read and write the SDRAM for hardware verification. The DDR4 SDRAM controller handles the complex aspects of using the DDR4 SDRAM by initializing the memory devices, managing the SDRAM banks, and keeping the devices refreshed at the appropriate intervals.

System Block Diagram

Figure 2-8 shows the system block diagram of this demonstration. In the Platform Designer (formerly Qsys), one 25 MHz OSC and clock gernerator(Si5332B) are used. The OSC and clock gernerator will provide a 150 Mhz clock to the on-board DDR4 SDRAM(DDR4A) as the reference clock. There is a DDR4 Controller which is used in the demonstrations. The controller is responsible for on-board DDR4 SDRAM (DDR4A). The DDR4 controllers are configured as 1GB DDR4 controller. Depending on the FPGA with different speed grade, the controller generates clocks with different speeds. For details, please refer to **Table 2-3**. The Nios V processor is used to perform the memory test. The Nios V program is running in the On-Chip Memory. A PIO Controller is used to monitor buttons status which is used to trigger starting memory testing.

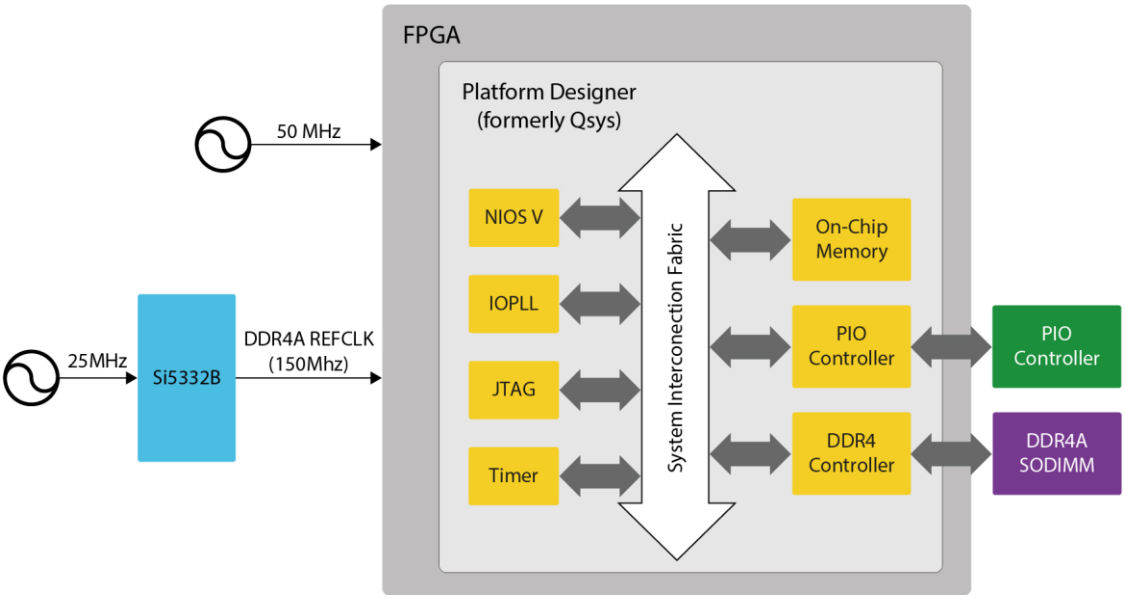


Figure 2-8 Block diagram of the DDR4 Basic Demonstration

The system flow is controlled by a Nios V program. First, the Nios V program writes test patterns into the whole 1GB of SDRAM. Then, it calls Nios V system function, alt_dache_flush_all(), to make sure all data has been written to SDRAM. Finally, it reads

data from SDRAM for data verification. Maybe the process takes a long time, and there is a quick test. The Nios V program writes a constant pattern into the address line and data line and reads it back for verification. The program will show progress in Nios V terminal when writing/reading data to/from the SDRAM. When verification process is completed, the result is displayed in the Nios V terminal.

Table 2-3 DDR4 clock frequency for each speed grade of FPGA

FPGA Speed Grade	DDR4 Clock Frequency(MHz)
A5ED065BB32A	1200 (DDR4 2400)

Design Tools

- Quartus Prime 24.2 Pro Edition

Demonstration Source Code

- Quartus Project directory: DDR4_Test_NiosV
- Nios V Eclipse: DDR4_Test_NiosV \software

Demonstration Batch File

Demo Batch File Folder: DDR4_Test_NiosV \demo_batch

The demo batch file includes following files:

- Batch File for USB-Blaster II: test.bat
- FPGA Configure File: golden_top.sof/
- Nios V Program: ddr4.elf

Demonstration Setup

Please follow below procedures to set up the demonstrations.

1. Make sure Quartus Prime and Nios V are installed on your PC.
2. Use a USB Cable to connect the PC and the FPGA board and install USB Blaster II driver if necessary.
3. Power on the FPGA board.
4. Execute the demo batch file “test.bat” under the folder “DDR4_Test_NiosV\demo_batch”.

5. After the Nios V program is downloaded and executed successfully, a prompt message will be displayed in the command-line terminal.
6. For DDR4 full test, please input key ‘0’ and press ‘Enter’ in the command-line terminal as shown in **Figure 2-9**. The program will display progressing and result information.
7. For DDR4 quick test, please input key ‘1’ and press ‘Enter’ in the command-line terminal as shown in **Figure 2-10**. The program will display progressing and result information. Press Button0 of the FPGA board to start SDRAM verify process. Note, this test may take about few minutes to run.

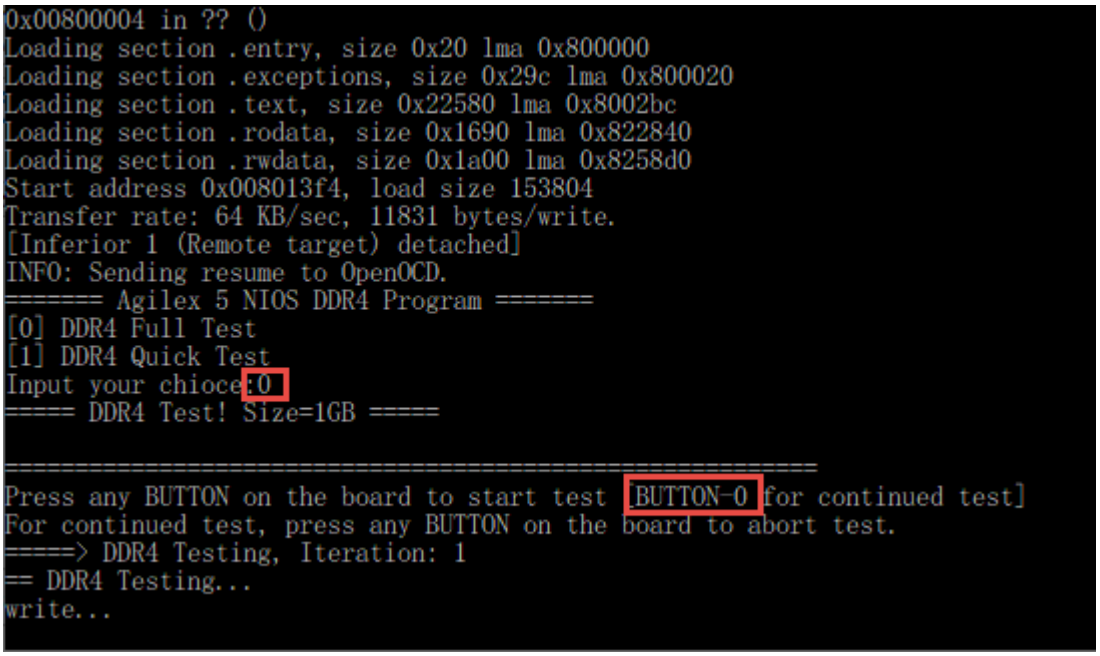


Figure 2-9 Progress option [0] DDR4x2 Test

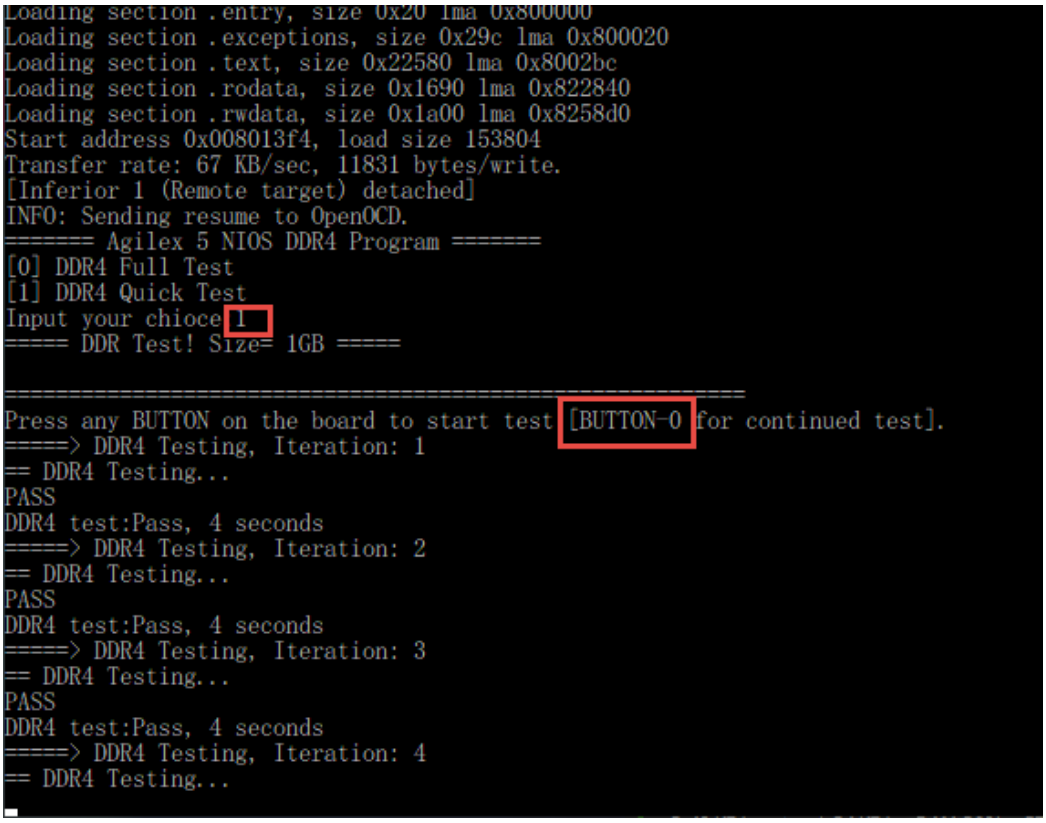


Figure 2-10 Progress and Result Information for "DDR4 Quick Test"

2.5 Audio Recording and Playing

This demonstration shows how to implement an audio recorder and player on DE25-Standard board with the built-in audio CODEC chip. It is developed based on Qsys and Eclipse. **Figure 2-11** shows the buttons and slide switches used to interact this demonstration onboard. Users can configure this audio system through two push-buttons and four slide switches:

- SW0 is used to specify the recording source to be Line-in or MIC-In.
- SW3, SW4, and SW5 are used to specify the recording sample rate such as 96K, 48K, 44.1K, 32K, or 8K.
- **Table 2-4** and **Table 2-5** summarize the usage of slide switches for configuring the audio recorder and player.

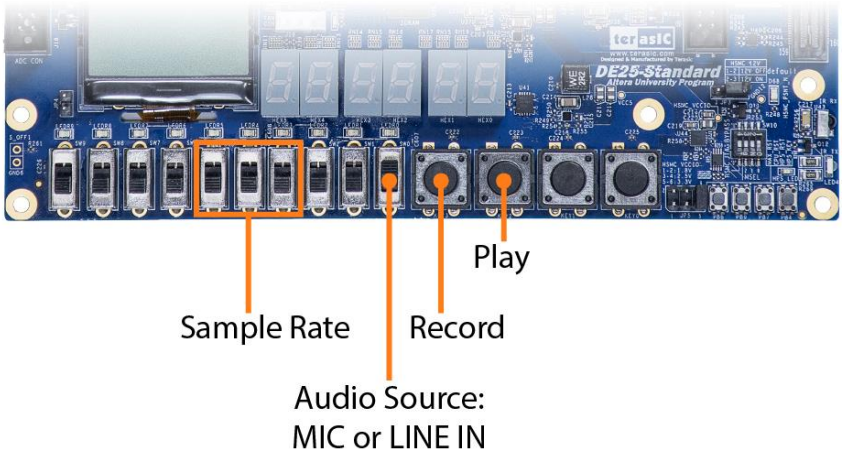


Figure 2-11 Buttons and switches for the audio recorder and player

Figure 2-11 shows the block diagram of audio recorder and player design. There are hardware and software parts in the block diagram. The software part stores the Nios V program in the on-chip memory. The hardware part is built under Qsys in Quartus Prime. The hardware part includes all the other blocks such as the “AUDIO Controller”, which is a user-defined Qsys component and it is designed to send audio data to the audio chip or receive audio data from the audio chip.

The audio chip is programmed through I2C protocol, which is implemented in C code. The I2C pins from the audio chip are connected to Qsys system interconnect fabric through PIO controllers. The audio chip is configured in master mode in this demonstration. The audio interface is configured as 16-bit I2S mode. 18.432MHz clock generated by the PLL is connected to the MCLK/XTI pin of the audio chip through the audio controller.

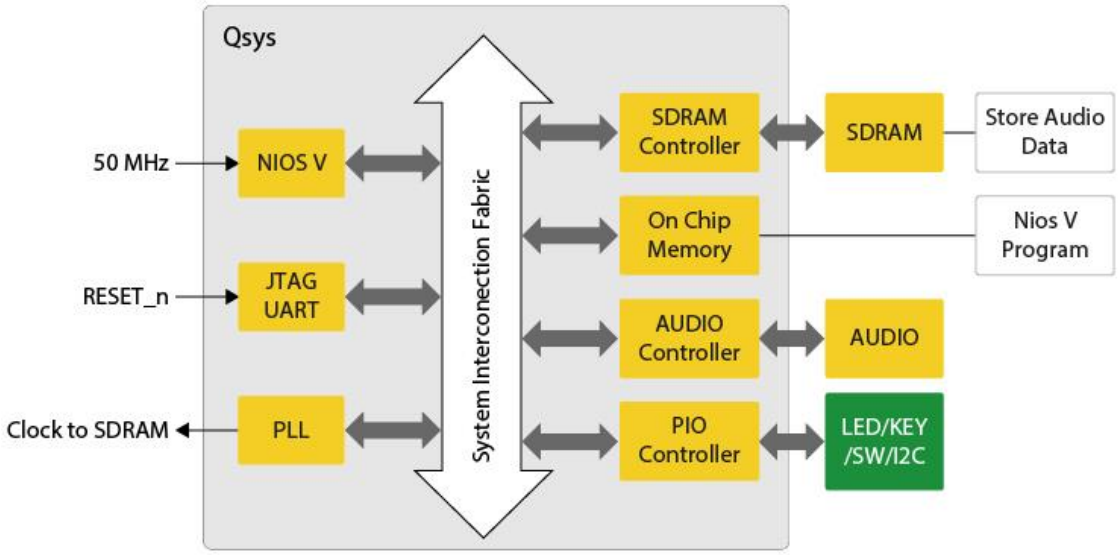


Figure 2-12 Block diagram of the audio recorder and player

■ Demonstration Setup, File Locations, and Instructions

- Hardware project directory: Audio
- Bitstream used: golden_top.sof
- Software project directory: Audio\software
- Connect an audio source to the Line-in port
- Connect a Microphone to the MIC-in port
- Connect a speaker or headset to the Line-out port
- Load the bitstream into the FPGA (note *1)
- Load the software execution file into the FPGA (note *1)
- Configure the audio with SW0, as shown in **Table 2-4**
- Press KEY3 to start/stop audio recording (note *2)
- Press KEY2 to start/stop audio playing (note *3)

Table 2-4 Slide switches usage for audio source

Slide Switches	0 – DOWN Position	1 – UP Position
SW0	**Audio is from MIC-in**	**Audio is from Line-in**

Table 2-5 Settings of switches for the sample rate of audio recorder and player

SW5 (0 – DOWN; 1- UP)	SW4 (0 – DOWN; 1-UP)	SW3 (0 – DOWN; 1-UP)	Sample Rate
0	**0**	**0**	**96K**
0	**0**	**1**	**48K**
0	**1**	**0**	**44.1K**
0	**1**	**1**	**32K**
1	**0**	**0**	**8K**
Unlisted combination			**96K**

Note:

(1). Execute Audio/demo_batch/test.bat to download .sof and .elf files.

(2). Recording process will stop if the audio buffer is full.

(3). Playing process will stop if the audio data is played completely.

2.6 Karaoke Machine

This demonstration uses the microphone-in, line-in, and line-out ports on DE25-Standard to create a Karaoke machine. The SSM2603 CODEC is configured in master mode. The audio CODEC generates AD/DA serial bit clock (BCK) and the left/right channel clock (LRCK) automatically. The I2C interface is used to configure the audio CODEC, as shown in **Figure 2-13**. The sample rate and gain of the CODEC are set in a similar manner, and the data input from the line-in port is then mixed with the microphone-in port. The result is sent out to the line-out port.

The sample rate is set to 48 kHz in this demonstration. The gain of the audio CODEC is reconfigured via I2C bus by pressing the pushbutton KEY0, cycling within ten predefined gain values (volume levels) provided by the device.

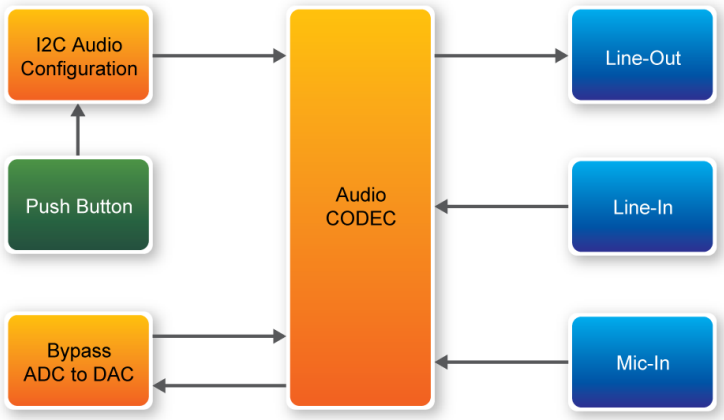


Figure 2-13 Block diagram of the Karaoke machine demonstration

■ Demonstration Setup, File Locations, and Instructions

- Project directory: i2sound
- Bitstream used: golden_top.sof
- Connect a microphone to the microphone-in port (pink color)
- Connect the audio output of a music player, such as a MP3 player or computer, to the line-in port (blue color)
- Connect a headset/speaker to the line-out port (green color)
- Load the bitstream into the FPGA by executing the batch file 'test.bat' in the directory i2sound\demo_batch
- Users should be able to hear a mixture of microphone sound and the sound from the music player
- Press KEY0 to adjust the volume; it cycles between volume level 0 to 9

Figure 2-14 illustrates the setup for this demonstration.

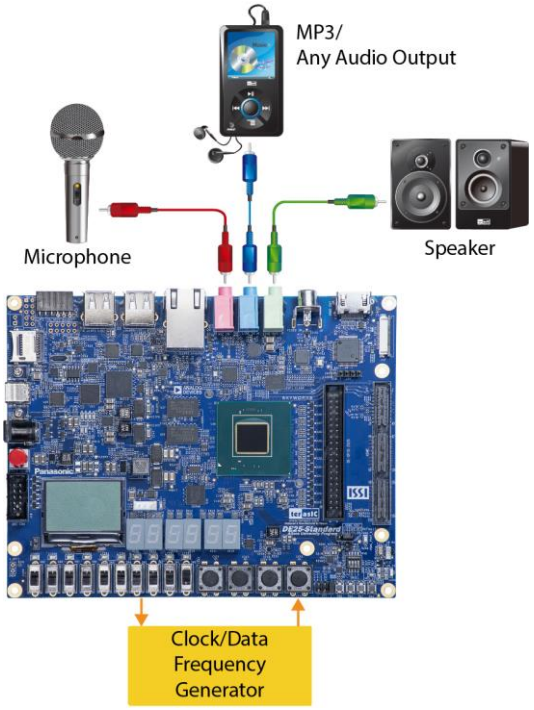


Figure 2-14 Setup for the Karaoke machine

2.7 IR Emitter LED and Receiver Demonstration

DE25-Standard system CD has an example of using the IR Emitter LED and IR receiver. This demonstration is coded in Verilog HDL.

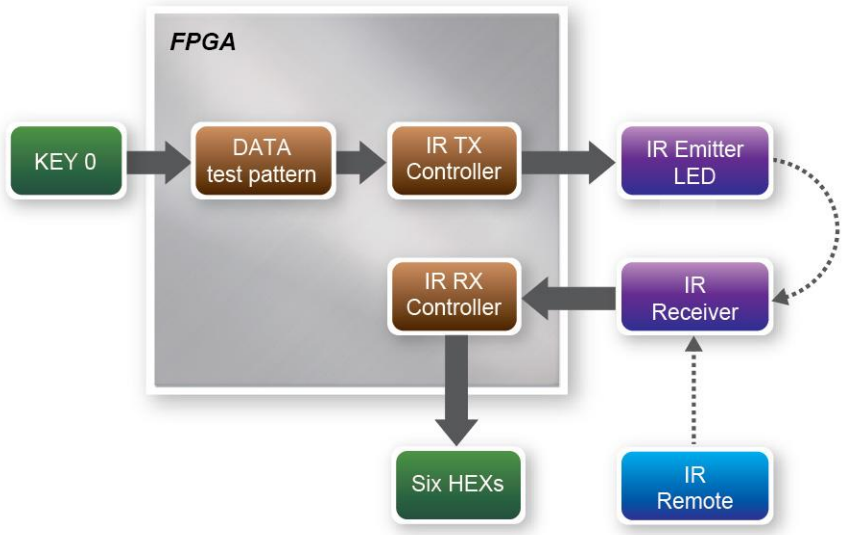


Figure 2-15 Block diagram of the IR emitter LED and receiver demonstration

Figure 2-15 shows the block diagram of the design. It implements a IR TX Controller and a IR RX Controller. When KEY0 is pressed, data test pattern generator will generate data to the IR TX Controller continuously. When IR TX Controller is active, it will format the data to be compatible with NEC IR transmission protocol and send it out through the IR emitter LED. The IR receiver will decode the received data and display it on the six HEXs. Users can also use a remote control to send data to the IR Receiver. The main function of IR TX /RX controller and IR remote control in this demonstration is described in the following sections.

■ IR TX Controller

Users can input 8-bit address and 8-bit command into the IR TX Controller. The IR TX Controller will encode the address and command first before sending it out according to NEC IR transmission protocol through the IR emitter LED. The input clock of IR TX Controller should be 50MHz.

The NEC IR transmission protocol uses pulse distance to encode the message bits. Each pulse burst is 562.5μs in length with a carrier frequency of 38kHz (26.3μs).

Figure 2-16 shows the duration of logical “1” and “0”. Logical bits are transmitted as follows:

- Logical '0' – a 562.5µs pulse burst followed by a 562.5µs space with a total transmit time of 1.125ms
- Logical '1' – a 562.5µs pulse burst followed by a 1.6875ms space with a total transmit time of 2.25ms

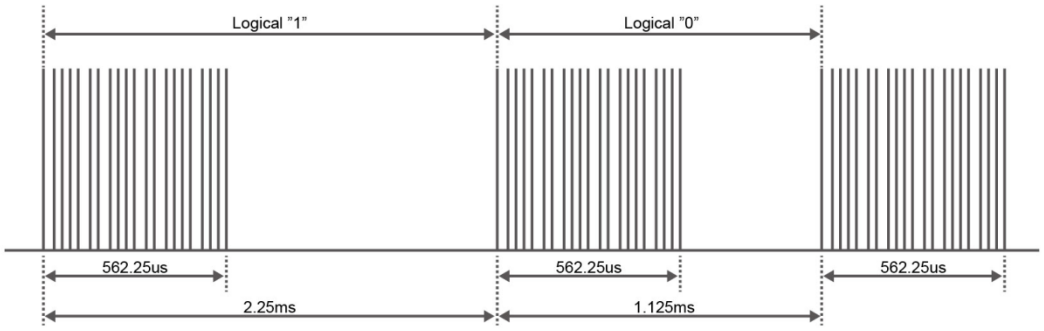


Figure 2-16 Duration of logical "1"and logical "0"

Figure 2-17 shows a frame of the protocol. Protocol sends a lead code first, which is a 9ms leading pulse burst, followed by a 4.5ms window. The second inversed data is sent to verify the accuracy of the information received. A final 562.5µs pulse burst is sent to signify the end of message transmission. Because the data is sent in pair (original and inverted) according to the protocol, the overall transmission time is constant.

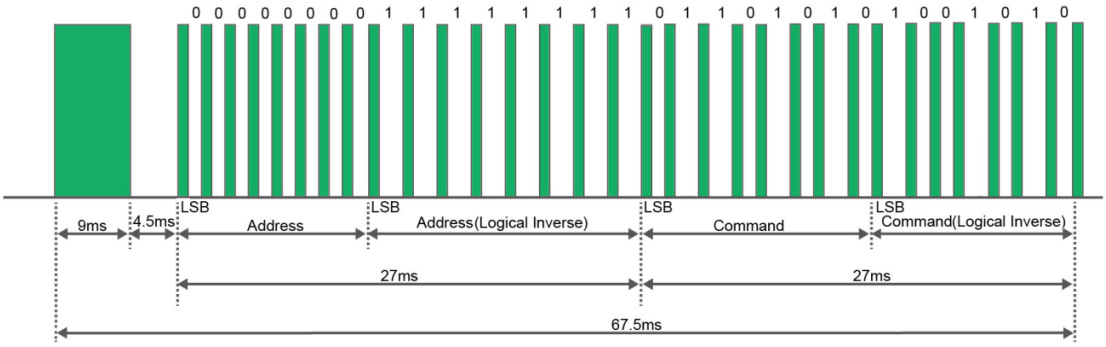


Figure 2-17 16 Typical frame of NEC protocol

Note: The signal received by IR Receiver is inverted. For instance, if IR TX Controller sends a lead code 9 ms high and then 4.5 ms low, IR Receiver will receive a 9 ms low

and then 4.5 ms high lead code.

■ IR Remote

When a key on the remote control shown in Figure 2-18 is pressed, the remote control will emit a standard frame, as shown in **Table 2-6**. The beginning of the frame is the lead code, which represents the start bit, followed by the key-related information. The last bit end code represents the end of the frame. The value of this frame is completely inverted at the receiving end.

Figure 2-18 The remote control used in this demonstration

Table 2-6 Key Code Information for Each Key on the Remote Control

Key	*Key Code*	*Key*	*Key Code*	*Key*	*Key Code*	*Key*	*Key Code*
A	0x0F	B	0x13	C	0x10	⏻	0x12
1	0x01	2	0x02	3	0x03	▲	0x1A
4	0x04	5	0x05	6	0x06	▼	0x1E
7	0x07	8	0x08	9	0x09	▲	0x1B
⏏	0x11	0	0x00	↩	0x17	▼	0x1F
⏯	0x16	◀	0x14	▶	0x18	🔇	0x0C

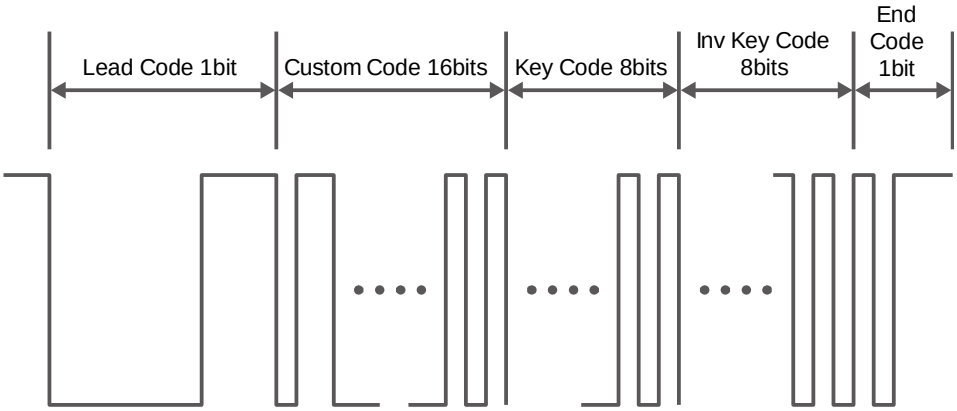


Figure 2-19 The transmitting frame of the IR remote control

■ IR RX Controller

The following demonstration shows how to implement the IP of IR receiver controller in the FPGA. Figure 2-20 shows the modules used in this demo, including Code Detector, State Machine, and Shift Register. At the beginning the IR receiver demodulates the signal inputs to the Code Detector. The Code Detector will check the Lead Code and feedback the examination result to the State Machine.

The State Machine block will change the state from IDLE to GUIDANCE once the Lead Code is detected. If the Code Detector detects the Custom Code status, the current state will change from GUIDANCE to DATAREAD state. The Code Detector will also save the receiving data and output to the Shift Register and display on the 7-segment. Figure 5 20 shows the state shift diagram of State Machine block. The input clock should be 50MHz.

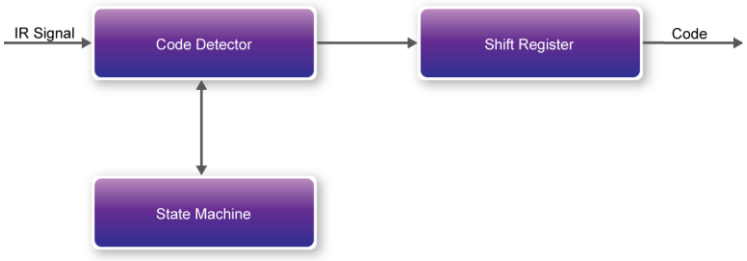


Figure 2-20 Modules in the IR Receiver controller

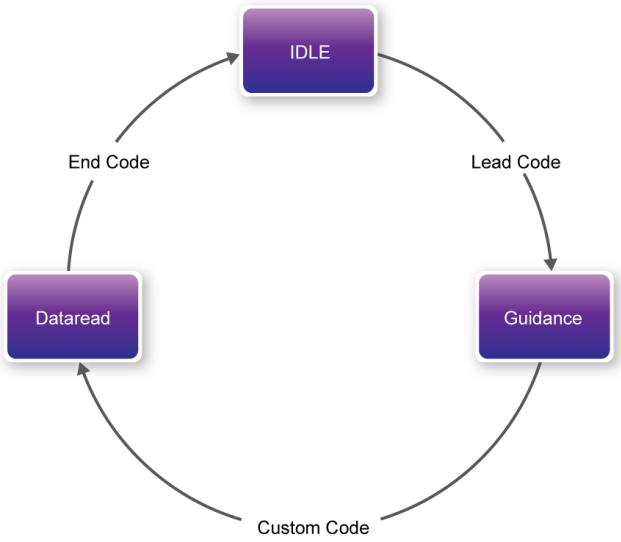


Figure 2-21 State shift diagram of State Machine block

■ Design Tools

- Quartus Prime 24.2 Pro Edition

■ Demonstration Source Code

- Quartus Project directory: IR
- Bitstream used: golden_top.sof

■ Demonstration Batch File

Demo Batch File Folder: IR\demo_batch

The demo batch file includes following files:

- Batch File for USB-Blaster II: test.bat
- FPGA Configure File: golden_top.sof

■ Demonstration Setup and Instructions

- Please make sure both Quartus II and USB-Blaster II driver are installed on the host PC.
- Power on the board.
- Load the bitstream into the FPGA by executing IR\demo_batch\ test.bat
- Keep pressing KEY[0] to enable the pattern to be sent out continuously by

the IR TX Controller.

- Observe the six HEXs according to **Table 2-7**.
- Release KEY[0] to stop the IR TX.
- Point the IR receiver with the remote control and press any button.
- Observe the six HEXs according to **Table 2-7**.

Table 2-7 Detailed Information of the Indicators

Indicator Name	Description
HEX5	**Inversed high byte of DATA(Key Code)**
HEX4	**Inversed low byte of DATA(Key Code)**
HEX3	**High byte of ADDRESS(Custom Code)**
HEX2	**Low byte of ADDRESS(Custom Code)**
HEX1	**High byte of DATA(Key Code)**
HEX0	**Low byte of DATA (Key Code)**

2.8 ADC Reading

This demonstration illustrates steps to evaluate the performance of the 8-channel 12-bit A/D Converter LTC2308. The DC 5.0V on the 2x5 header is used to drive the analog signals by a trimmer potentiometer. The voltage should be adjusted within the range between 0 and 4.096V. The 12-bit voltage measurement is displayed on the NIOS V console. **Figure 2-22** shows the block diagram of this demonstration.

The default full-scale of ADC is 0~4.096V.

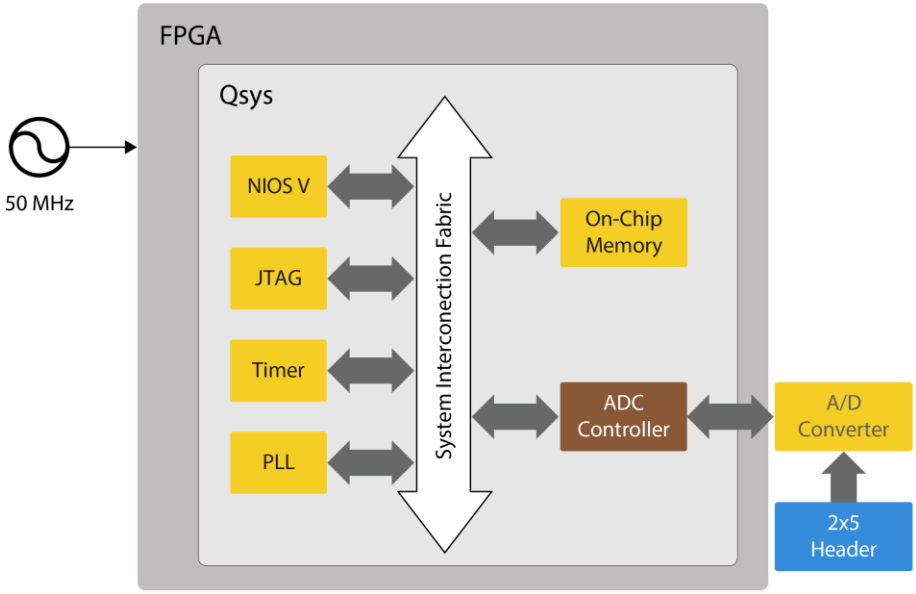


Figure 2-22 Block diagram of ADC reading

Figure 2-23 depicts the pin arrangement of the 2x5 header. This header is the input source of ADC convertor in this demonstration. Users can connect a trimmer to the specified ADC channel (ADC_IN0 ~ ADC_IN7) that provides voltage to the ADC convert. The FPGA will read the associated register in the convertor via serial interface and translates it to voltage value to be displayed on the Nios V console.

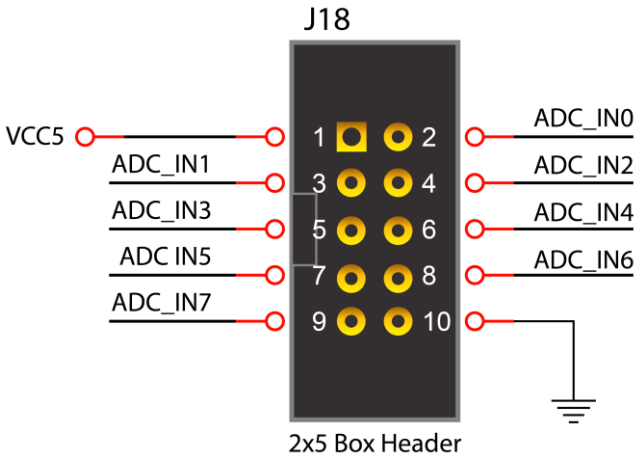


Figure 2-23 Pin distribution of the 2x5 Header for the ADC

The LTC2308 is a low noise, 500ksps, 8-channel, 12-bit ADC with an SPI/MICROWIRE compatible serial interface. The internal conversion clock allows the external serial output data clock (SCK) to operate at any frequency up to 40MHz.In this demonstration,

we realized the SPI protocol in Verilog, and packet it into Avalon MM slave IP so that it can be connected to Qsys.

Figure 2-24 is SPI timing specification of LTC2308.

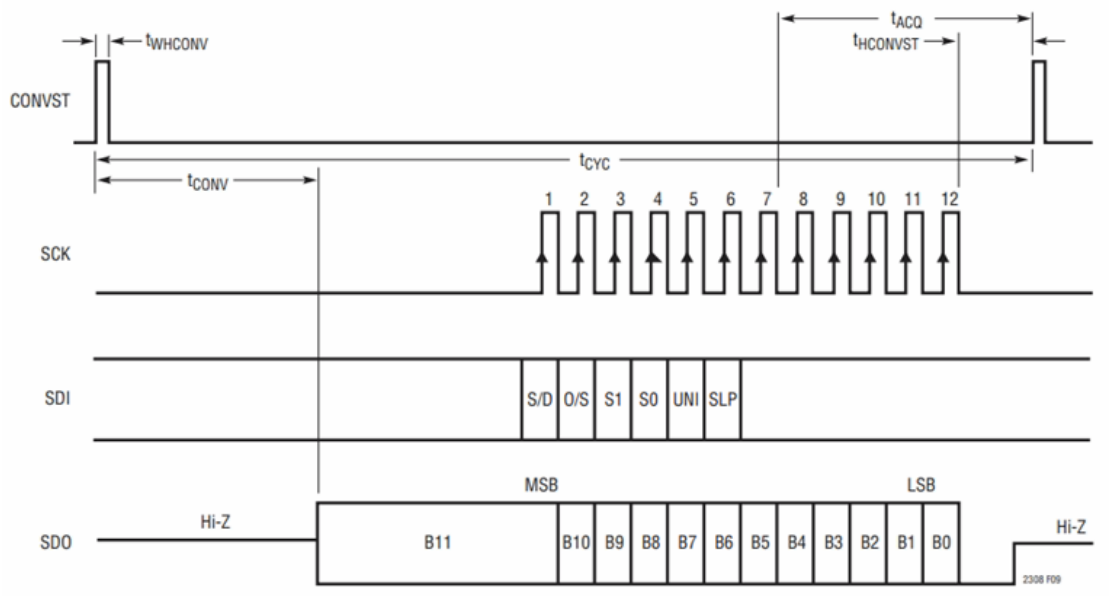


Figure 2-24 LTC2308 Timing with a Short CONVST Pulse

Important: Users should pay more attention to the impedance matching between the input source and the ADC circuit. If the source impedance of the driving circuit is low, the ADC inputs can be driven directly. Otherwise, more acquisition time should be allowed for a source with higher impedance.

To modify acquisition time tACQ, user can change the tHCONVST macro value in adc_ltc2308.v. When SCK is set to 40MHz, it means 25ns per unit. The default tHCONVST is set to 320, achieving a 100KHz fsample. Thus adding more tHCONVST time (by increasing tHCONVST macro value) will lower the sample rate of the ADC Converter.

```
`define tHCONVST        320
```

Figure 2-25 shows the example MUX configurations of ADC. In this demonstration, it is configured as 8 signal-end channel in the verilog code. User can change SW[2:0] to measure the corresponding channel.The default reference voltage is 4.096V.

The formula of the sample voltage is:

Sample Voltage = ADC Data / full scale Data * Reference Voltage.

In this demonstration, full scale is 2^12 =4096. Reference Voltage is 4.096V. Thus

ADC Value = ADC data/4096*4.096 = ADC data /1000

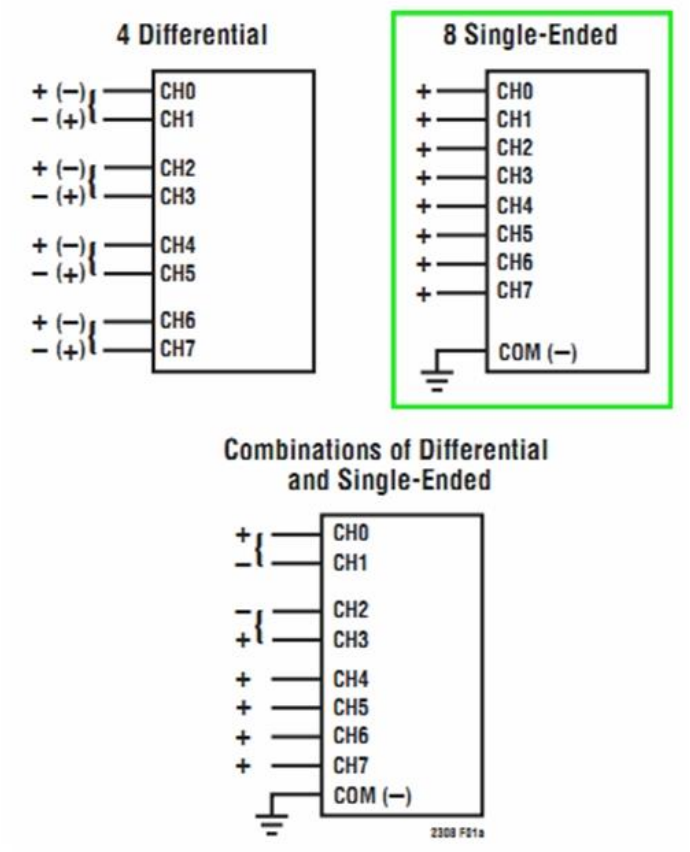


Figure 2-25 Example MUX Configurations

■ System Requirements

- DE25-Standard board x1
- Trimmer Potentiometer x1
- Wire Strip x3

Demonstration File Locations

- Hardware project directory: ADC
- Bitstream used: golden_top.sof
- Software project directory: software
- Demo batch file : ADC\demo_batch\test.bat

Install Ashling RiscFree IDE

Before executing this demo with RISC-V core, users need to install Ashling RiscFree IDE for Intel® FPGAs to ensure that this batch file can be executed correctly. The file name should be *RiscFreeSetup-<Quartus Version>-windows.exe.* Users can find it under the Quartus Pro download page (Individual Files tab).

Demonstration Setup

Please follow below procedures to set up the demonstrations.

1. Connect the trimmer to corresponding ADC channel on the 2x5 header, as shown in **Figure 2-26**, as well as the +5V and GND signals. The setup shown above is connected to ADC channel 0.
2. Connect a USB cable to the Type-C USB connector (J16) on the DE25-Standard board and the host PC.
3. Execute the demo batch file test.bat to load the bitstream and software execution file to the FPGA.
4. The command-line window will display the voltage of the specified channel voltage result information.
5. Provide any input voltage to other ADC channels and set SW[2:0] to the corresponding channel if user want to measure other channels

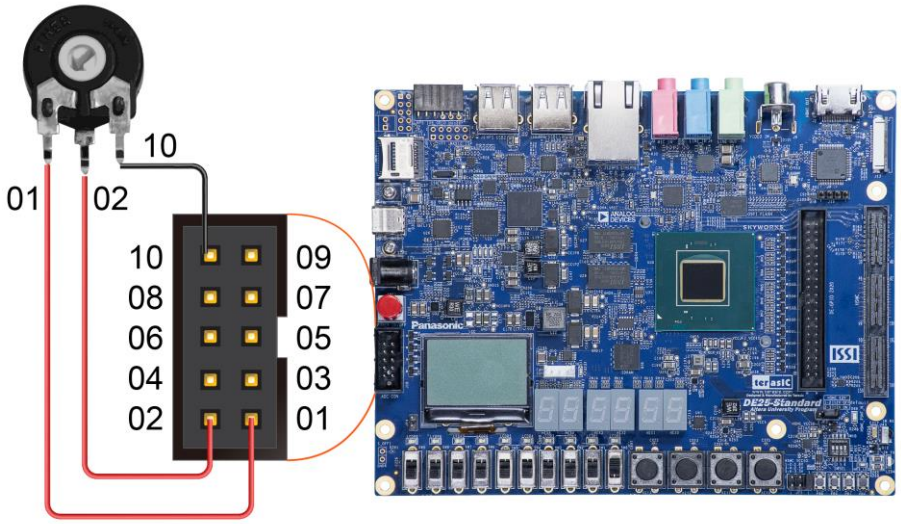


Figure 2-26 Hardware setup for the ADC reading demonstration

2.9 TV Box Demonstration

This demonstration turns DE25-Standard board into a TV box by playing video and audio from a DVD player using the HDMI output and the TV decoder on the DE25-Standard board. **Figure 2-27** shows the block diagram of the design.

When the demonstration bitstream is loaded into FPGA. The I2C_AV_Config and I2C_HDMI_Config block will configure the TV decoder and HDMI transmitter chip via I2C bus.

The register values of the TV decoder are used to configure the TV decoder via the I2C_AV_Config block, which uses the I2C protocol to communicate with the TV decoder. The TV decoder will be unstable for a time period upon power up, and the Lock Detector block is responsible for detecting this instability.The ITU-R 656 Decoder block extracts YcrCb 4:2:2 (YUV 4:2:2) video signals from the ITU-R 656 data stream sent from the TV decoder. It also generates a data valid control signal, which indicates the valid period of data output. the video signal for the TV decoder is interlaced; de-interlacing needs to be performed on the data source. The SDRAM Frame Buffer and a field selection multiplexer (MUX), which is controlled by the VGA Controller, are used to perform the de-interlacing operation. The VGA Controller also generates data request and odd/even selection signals to the SDRAM Frame Buffer and filed selection multiplexer (MUX). The YUV422 to YUV444 block converts the selected YcrCb 4:2:2 (YUV 4:2:2) video

data to the YcrCb 4:4:4 (YUV 4:4:4) video data format.

Finally, the YcrCb_to_RGB block converts the YcrCb data into RGB data output. The VGA Controller block generates standard HDMI synchronous signals to HDMI transmitter chip.

The figure also shows the TV decoder (ADV7180) and the HDMI transmitter (ADV7513) chip used.

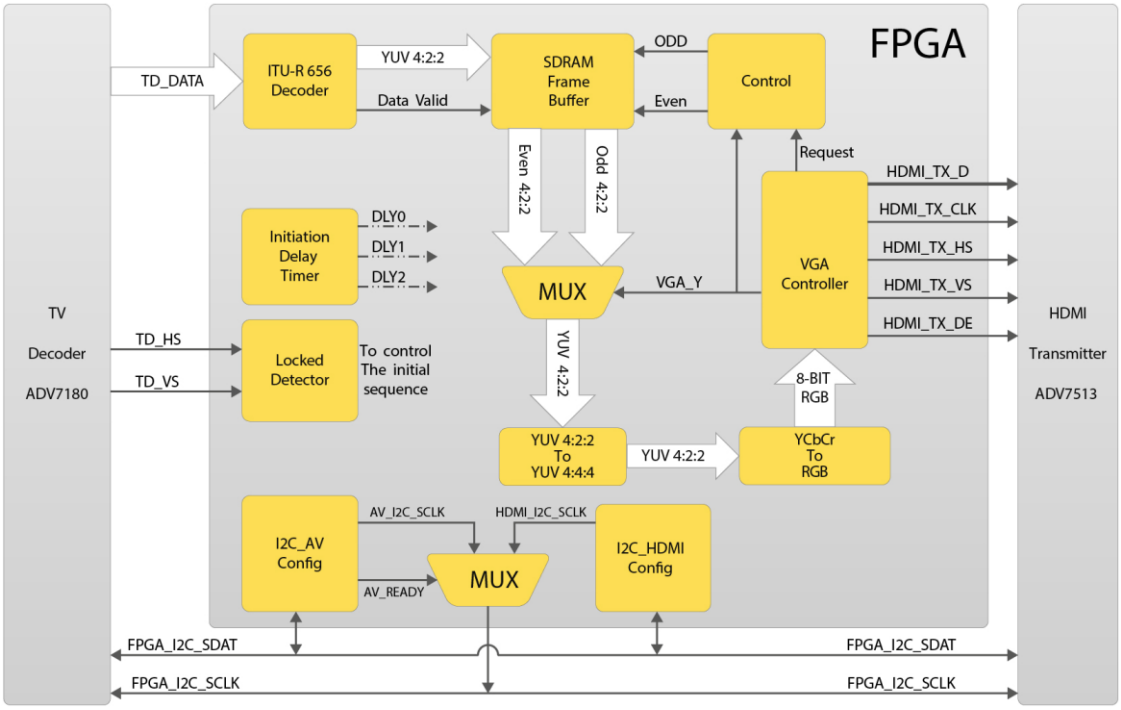


Figure 2-27 Block diagram of TV box demonstration

■ Demonstration Source Code

- Project directory: TV
- Bitstream used: golden_top.sof

■ Demonstration File Locations

Demo batch directory: \TV\demo_batch
The folder includes the following files:

- Batch file: test.bat
- FPGA configuration file : golden_top.sof

■ Demonstration Setup

Please follow below procedures to set up the demonstrations.

1. Connect a DVD player's composite video output (yellow plug) to the Video-in RCA jack (J8) on the DE25-Standard board, as shown in Figure 5 10. The DVD player has to be configured to provide:

   - NTSC output
   - 60Hz refresh rate
   - 4:3 aspect ratio
   - Non-progressive video

2. Connect the HDMI output of the DE25-Standard board to a HDMI interface monitor.
3. Connect a USB cable to the Type-C USB connector (J16) on the DE25-Standard board and the host PC.
4. Load the bitstream into the FPGA by executing the batch file 'test.bat' from the directory \TV\demo_batch\. Press KEY0 on the DE25-Standard board to reset the demonstration.
5. **Figure 2-28** Setup for the TV box demonstration

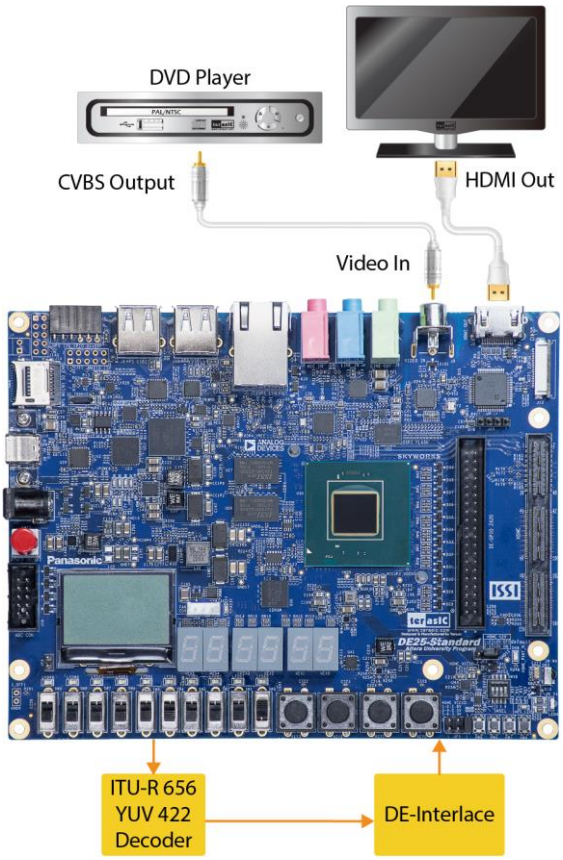


Figure 2-28 Setup for the TV box demonstration

Chapter 3

Examples for HPS SoC

This chapter provides several C-code examples based on the Intel SoC Linux These examples demonstrate major features connected to HPS interface on Agilex board such as users LED/KEY and Network Communication. All the associated files can be found in the directory CD/Demonstrations/SOC of the DE25-Standard System CD.

To install the demonstrations on the Host computer: Copy the directory Demonstrations into a local directory of your choice. ARM Toolchain is required for users to compile the c-code project.

3.1 HPS LED/KEY

This demonstration shows how to use the system call with built-in LED and GPIO driver to control the LED and KEY which are connected to HPS GPIO ports. The built-in GPIO driver is included the DE25-Standard Linux BSP.

■ How to control LED

Here is an example procedure to control the HPS LED:

1. Open LED device: Open device file “/sys/class/leds/hps_led0/brightness”.
2. Turn on/off LED: Write data to the device file for LED control. Write “1” to turn on LED, write “0” to turn off LED.
3. Close LED device: Close the device file.

■ How to Read Button Status

User space GPIO driver is used to read button status. Since linux 4.8 the GPIO sysfs interface is deprecated. User space should use the character device instead. This

library encapsulates the ioctl calls and data structures behind a straightforward API. Here is an example procedure to read the HPS Button Status:

1. Open button character device: Open character device file “/dev/gpiochip0”.
2. Configure line 1 (HPS_KEY is connected to GPIO1_IO1 in schematic) of /dev/gpiochip0 as Input GPIO
3. Read line 1 status from the button character device.
4. Close button character device: Close the character device file.

■ Function Block Diagram

Figure 3-1 shows the function block diagram of the HPS LED/KEY demonstration. LED and KEY are connected to GPIO1 Controller. The built-in GPIO driver offers access interfaces for user application program. System call open, write and close are used to control LED, and open, ioctl and closed are used to control the button.

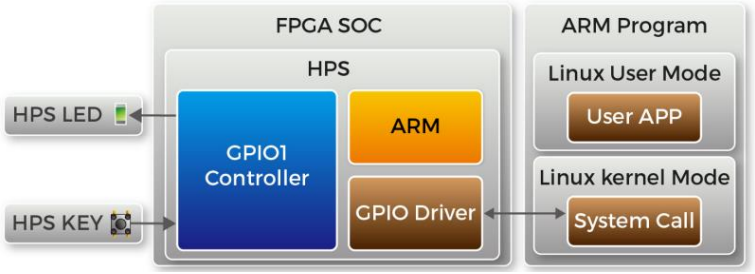


Figure 3-1 Function block diagram of HPS LED/KEY demonstration

■ Function Implement

The c project include main.c, gpio_lib.c and led_lib.c files. The main.c implements the demo main flow. The gpio_lib.c implement the KEY control functions. The led_lib.c implements LED control functions.

The led_lib implement three LED functions. With the file descriptor return by led_fd_open function, user can use led_fd_write to trun on/off the LED. Call “led_fd_write(fd_led, “1”, 2) “ will turn on the LED, and Call “led_fd_write(fd_led, “0”, 2) “ will turn off the LED. Library API is described as following:

- **int led_fd_open (unsigned int led):**

The led_fd_open function is used to open the LED device file with the specified LED number as parameter. The function return a file descriptor for the LED device.

- **int led_fd_write (int fd, const void *buf, size_t count):**

The led_fd_write function is used to write data to the LED device file. It is used to turn on/off the LED.

- **int led_fd_close(int fd):**

The led_fd_close function is used to close a file descriptor.

The gpio_lib implement three button functions described as following:

- **GPIO_HANDLE* gpio_open_line(char *dev_name, int line, int direction):**

The gpio_open_line will open the character device file specified by *dev_name, and query the line information. All relative information are stored in a data structure GPIO_HANDLE which is dynamic created by malloc . The function returns a pointer points to a data structure GPIO_HANDLE.

- **bool gpio_get_line_value(GPIO_HANDLE *pHandle, unsigned int *pValue):**

The gpio_get_line_value reads HPS KEY status. The status is return via pValue. If HPS button is pressed, *pValue return 0, otherwise 1 is return.

- **void gpio_close_line(GPIO_HANDLE *pHandle):**

The gpio_close_line will release resource and close the open character device file.

■ Flow Control Implement

The flow control is implemented in main.c. The LED is blinking, and keep lighten when HPS KEY is pressed. The GPIO functions implemented in gpio_lib.c are used to monitor HPS KEY status. The LED functions implemented in led_lib.c is used to turn on/off the HPS LED.

Figure 3-2 shows the procedure in main.c file, you can find it's very clear.

```
line_key = gpio_open_line("/dev/gpiochip0", 1/*line 1 for KEY*/, 0 /*input*/);
fd_led = led_fd_open(io_led);

loop = 20;
while (loop >= 0) {
    //gpio_get_value(io_key, &value);
    gpio_get_line_value(line_key, &key_value); // key_value is low active
    if (!key_value || bLedLight)
        led_fd_write(fd_led, "1", 2); // light led
    else
        led_fd_write(fd_led, "0", 2); // unlight led
    printf("key: %x\n", key_value);
    bLedLight = bLedLight?false:true;
    usleep(500*1000); // 0.5 second
    loop--;
}

led_fd_close(fd_led);
gpio_close_line(line_key);
```

Figure 3-2 LED/KEY implemented in c code

■ Demonstration Source Code

- Build tool: ARM GNU/Linux Toolchain
- Project directory: \Demonstration\SoC\hps_led_key
- Binary file: hps_led_key
- Build command: make ('make clean' to remove all temporal files)
- Execute command: sudo ./hps_led_key

■ Demonstration Setup

1. Connect a USB cable to the Micro USB connector (J10) on the FPGA Board and the Host PC.
2. Copy the executable file "**hps_led_key**" into the microSD card under the "**/home/root**" folder in Linux.
3. Insert the DE25-Standard Board Linux BSP micro SD card into the DE25-Standard Board.
4. Power on the DE25-Standard Board.
5. Launch Putty to establish the connection between the UART port of DE25-Standard Board and the Host PC.
6. In the Putty UART terminal, type “root” to login Linux.
7. Copy the executable file "hps_led_key" into the "/home/root" folder in Linux.
8. Type “./hps_led_key” in the UART terminal to start the program.
9. You will see the key status is shown in Putty UART terminal as shown in **Figure 3-3**.
10. Press HPS KEY will make key value become 0 and HPS LED keep lighten.
11. The program will be automatically terminated in 20 seconds, or press CTRL+C to terminate the program immediately.

```
root@agilex5:~# ./hps_led_key
/sys/class/leds/hps_led0/brightness
key: 1
key: 1
key: 1
key: 1
key: 1
```

Figure 3-3 LED/KEY test

3.2 SPI Interfaced Graphic LCD

This demonstration shows how to control the Graphic LCD by using the Linux SPI driver and GPIO drivers. The GPIO driver is used to control the LCD reset, backlight and mode. The SPI driver is used to delivery command and display data.

■ Function Block Diagram

Figure 3-4 shows the function block diagram of this demonstration. The LCD interfaces includes SPI and GPIO. The bot interfaces are connected to the **SPIM0**, **GPIO1** controller in HPS on this board. The SPI interface is used to transfer Data or Command from HPS to LCD. Because the LCD is write-only, only three SPI signals **LCM_SPIM_CLK**, **LCM_SPIM_SS**, and **LCM_SPIM_MOSI** are required. The **LCM_D_C** signal is used to indicate the signal transferred on the SPI bus is Data or Command. When **LCM_D_C** signal is pulled high, it means the signal on SPI bus is Data. When **LCM_D_C** signal is pulled low, it means the signal on SPI bus is Command. The **LCD_RST_n** is the reset control signal of LCD. This signal is low active. The **LCM_BK** signal is used to turn on/off the black light of the LCD. When this signal is pulled high, LCD backlight is turned on.

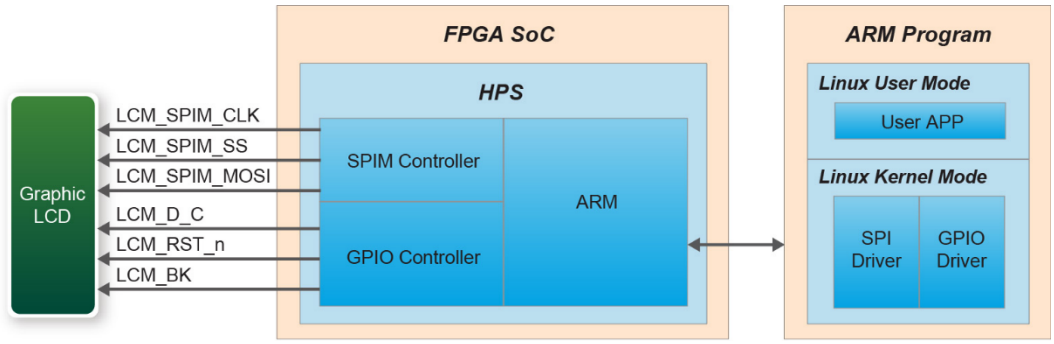


Figure 3-4 Block Diagram of the Graphic LCD Demonstration

■ LCD Control

Developer needs to initialize the LCD before sending any display data. The initialization includes:

- Common output mode select (Code: 0xC0~0xCF)
- Power sontrol set (Code: 0x28~0x2F)
- Display start line set (Code: 0x40~0x7F)
- Page address set (Code: 0xB0~0xB8)
- Column address set (Code: 0x00 to 0x18)
- Display ON/OFF (Code: 0xAE~0xAF)

For details of command sets, please refer to the ST7567 datasheet in the System CD. After the LCD is initialized, developer can start transferring display data. Due to the display area is divided into 8 page, developer must first specify target page and column address before starting to transfer display data. **Figure 3-5** shows the relationship between image data bits and LCD display pixels when page = 0, column = 0, and start line = 0.

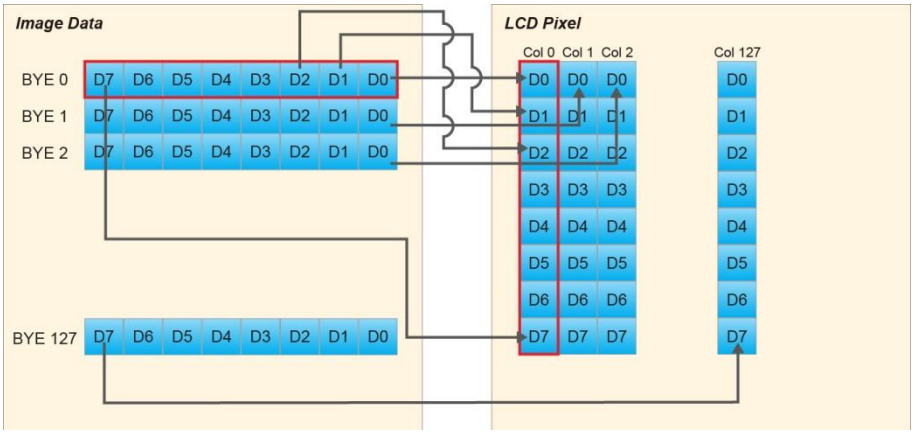


Figure 3-5 Relation between LCD display pixel and image data bits

■ SPIM Controller

In this demonstration, the HPS SPIM0 controller is driven by the SPI driver "/dev/spidev1.0". Please refer to the function "LCDHW_Init" in LCD_Hw.c for details. The spidev driver is already built in Terasic Linux BSP.

■ C-code Explanation

This demonstration includes the following major files:

- Gpio_lib.c: GPIO API for LCD IO control via GPIO Driver.
- LCD_HW.c: API to access LCD hardware via SPI and GPIO Drivers.
- LCD_Driver.c: LCD configuration API
- LCD_Lib.c: Top-level LCD control API
- lcd_graphic.c: Graphic and font APIs for LCD
- font.c: Font bitmap resource used by lcd_graphic.c
- main.c: Main program for this demonstration

The main program main.c calls "LCDHW_Init" to initialize the SPIM0 and GPIO controllers, which are used to control the LCD. It then calls "LCDHW_BackLight" to turn on the backlight of LCD. "LCD_Init" is called to initialize LCD configuration. Finally, the APIs in lcd_graphic.c are called to draw graphic on the LCD.

APIs in lcd_graphic.c don't drive LCD to draw graphic pixels directly. All graphic pixels are stored in a temporary image buffer called "Canvas". When API "DRAW_Refresh" is called, all drawing data in the Canvas is transferred to LCD. In this demonstration, main program calls "DRAW_Clear" to clear LCD Canvas first. "DRAW_Rect" and "DRAW_Circle" are called to draw geometry in Canvas. "DRAW_PrintString" is called to draw font in Canvas. Finally, "DRAW_Refresh" is called to move Canvas data onto LCD.

■ Demonstration Source Code

- Build tool: ARM GNU/Linux Toolchain
- Project directory: \Demonstration\SoC\hps_lcd
- Binary file: hps_lcd
- Build command: make ('make clean' to remove all temporal files)
- Execute command: sudo ./ hps_lcd

■ Demonstration Setup

1. Connect the USB cable to the Type-C connector (J16) on the FPGA board and host PC.
2. Make sure the executable file "hps_lcd" is copied into the SD card under the /home/root folder in Linux.
3. Insert the booting micro SD card into the FPGA board.

4. Power on the FPGA board.
5. Launch PuTTY to connect to the UART port of the FPGA board and type "root" to login Linux.
6. In the UART terminal of PuTTY, type "./hps_lcd" to start the LCD demo, as shown in **Figure 3-6**.

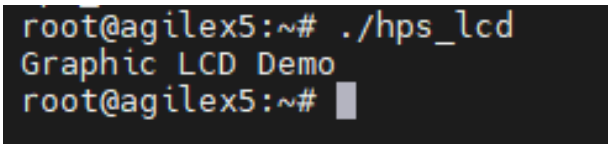


Figure 3-6 Launch LCD Demonstration

7. Users should see the LCD displayed as shown in **Figure 3-7**.

Figure 3-7 LCD display for the LCD Demonstration

3.3 I2C Interfaced G-sensor

This demonstration shows how to control the G-sensor by accessing its registers through the built-in I2C kernel driver in embedded Linux.

■ Function Block Diagram

Figure 3-8 shows the function block diagram of this demonstration. The G-sensor on the FPGA SoC board is connected to the I2C controller in HPS. The G-Sensor I2C 7-bit device address is 0x53. The system I2C bus driver is used to access the register files in the G-sensor. The G-sensor interrupt signal is connected to the PIO controller. This demonstration uses polling method to read the register data.

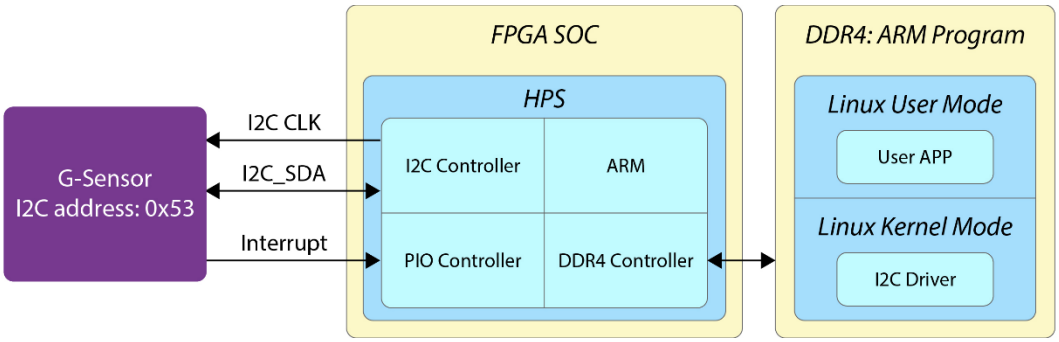


Figure 3-8 Block diagram of the G-sensor demonstration

■ I2C Driver

The procedures to read a register value from G-sensor register files by the existing I2C bus driver in the system are:

1. Open I2C bus driver "/dev/i2c-0": file = open("/dev/i2c-0", O_RDWR);
2. Specify G-sensor's I2C address 0x53: ioctl(file, I2C_SLAVE, 0x53);
3. Specify desired register index in g-sensor: write(file, &Addr8, sizeof(unsigned char));
4. Read one-byte register value: read(file, &Data8, sizeof(unsigned char));

The G-sensor I2C bus is connected to the I2C0 controller, as shown in the **Figure 3-9**. The driver name given is '/dev/i2c-0'.

Figure 3-9 Connection of HPS I2C signals

The step 4 above can be changed to the following to write a value into a register.

write(file, &Data8, sizeof(unsigned char));

The step 4 above can also be changed to the following to read multiple byte values.

read(file, &szData8, sizeof(szData8)); // where szData is an array of bytes

The step 4 above can be changed to the following to write multiple byte values.

write(file, &szData8, sizeof(szData8)); // where szData is an array of bytes

■ G-sensor Control

The ADI ADXL345 provides I2C and SPI interfaces. I2C interface is selected by setting the CS pin to high on the DE25-Standard board.

The ADI ADXL345 G-sensor provides user-selectable resolution up to 13-bit ± 16g. The resolution can be configured through the DATA_FORAMT(0x31) register. The data format in this demonstration is configured as:

- Full resolution mode
- ± 16g range mode
- Left-justified mode

The X/Y/Z data value can be derived from the DATAX0(0x32), DATAX1(0x33), DATAY0(0x34), DATAY1(0x35), DATAZ0(0x36), and DATAX1(0x37) registers. The DATAX0 represents the least significant byte and the DATAX1 represents the most significant byte. It is recommended to perform multiple-byte read of all registers to prevent change in data between sequential registers read. The following statement reads 6 bytes of X, Y, or Z value.

read(file, szData8, sizeof(szData8)); // where szData is an array of six-bytes

■ Demonstration Source Code

- Build tool: ARM GNU/Linux Toolchain
- Project directory: \Demonstration\SoC\hps_gsensor

- Binary file: gsensor
- Build command: make ('make clean' to remove all temporal files)
- Execute command: ./gsensor [loop count]

■ Demonstration Setup

Please follow below procedures to set up the demonstrations.

6. Connect a USB cable to the Type-C USB connector (J16) on the DE25-Standard board and the host PC.
7. Copy the executable file "gsensor" into the microSD card under the "/home/root" folder in Linux.
8. Insert the booting microSD card into the DE25-Standard board.
9. Power on the DE25-Standard board.
10. Launch PuTTY to establish connection to the UART port of DE25-Standard board. Type "root" to login Linux.
11. Execute "./gsensor" in the UART terminal of PuTTY to start the G-sensor polling.
12. The demo program will show the X, Y, and Z values in the PuTTY, as shown in **Figure 3-10**.

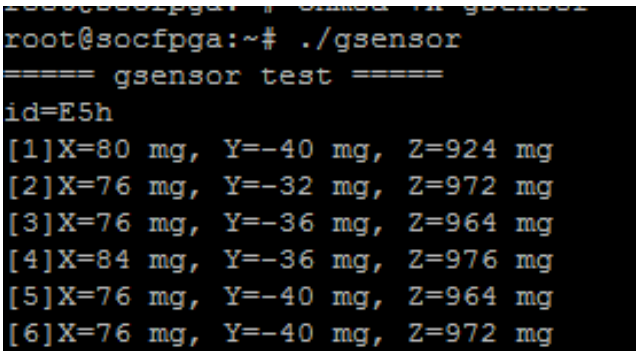


Figure 3-10 Terminal output of the G-sensor demonstration

13. Press "CTRL + C" to terminate the program.

3.4 Build C/C++ Project

This section describes how to recompile the above C/C++ project included in the

System CD.

First, user need to download and install ARM GNU/Linux tool chain:

5. Login Linux or WSL on Windows.
6. Type “cd ~”
7. Type “wget https://developer.arm.com/-/media/Files/downloads/gnu/11.2-2022.02/binrel/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz”
8. Type “tar xf gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz”
9. Type “export PATH=`pwd`/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin:$PATH”
10. Type “export CROSS_COMPILE=aarch64-none-linux-gnu-”
11. Type “git clone https://github.com/altera-opensource/intel-socfpga-hwlib” to download HPS hardware library.

Here is the procedure to compile the example projects in System CD:

1. Login Linux or WSL on Windows.
2. Type “cd ~”
3. Type “export PATH=`pwd`/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin:$PATH”
4. Type “export CROSS_COMPILE=aarch64-none-linux-gnu-”
5. Copy the CD Demo project into the Linux System and go to the project folder.
6. Type “make” to build project as shown in **Figure 3-11.**

Figure 3-11 Build C/C++ Project

Chapter 4

Program the Configuration File into QSPI Flash

There is a flash on the board for placing the bitstream files designed by the user. After the FPGA is powered up, the bit-stream file can be loaded into the FPGA from the flash.

This chapter will introduce how to program the factory default code into the flash on the board. It will also introduce how to convert the your own .sof file into a .jic file and program it into the flash.

■ MSEL switch setting

Before program the flash on the board, please make sure the MSEL pin of the FPGA is set to AS mode. So that user can program the flash via FPGA and JTAG interface. Please refer to the **Figure 4-1** to set the SW10 to AS mode: **MSEL[2:0] = "001"**.

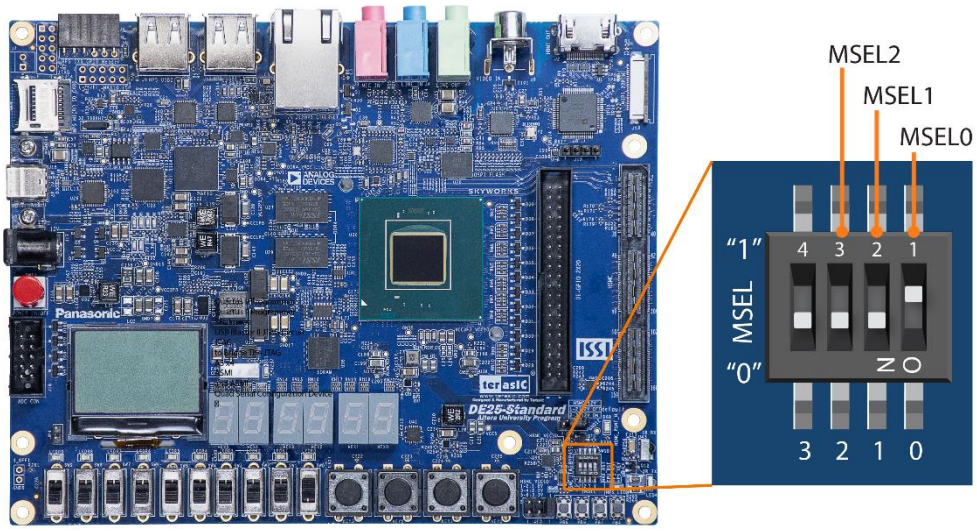


Figure 4-1 **Position of slide switches SW10 for Configuration Mode**

■ Program the Factory image file

The flash on the Board had programmed the factory file when shipped.
After power on, user can check if the user LEDR[9] is flashing, and after 10 seconds of booting. If not, please refer to following steps to re-program the flash with the factory code.

- Connect the USB cable to USB blaster II connector of the Board.
- Make sure MSEL switch is set to AS mode.
- Copy the factory code to host from the path:

System CD\Demonstration\SoC_FPGA\GHRD\output_files\program_qspi_flash

- Execute "**flash_program.bat**" as shown in **Figure 4-2** to erase and program the flash.

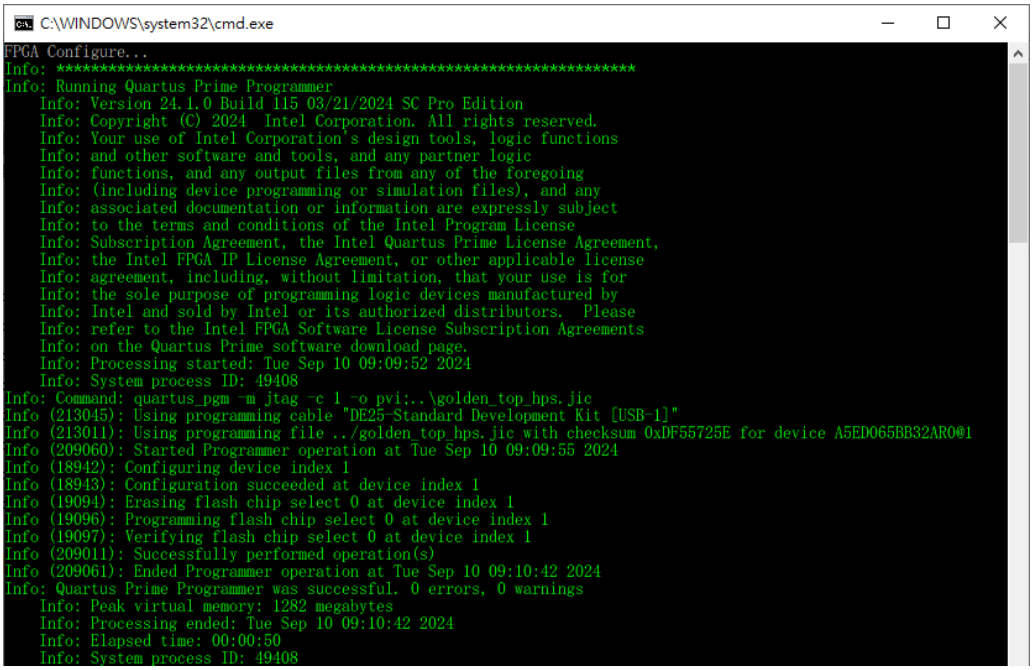


Figure 4-2 Program the factory code to the board

■ Programming User's own design into Flash

If users wants to program their own bitstream file into the flash of the board, Terasic had provided some batch files that help the user to convert the .sof file into a .jic file and then program it into the flash.

Users can obtain these tools from the following path:
System CD/Demonstration/FPGA/LED_Blink/demo_batch

There are some files under this path, and the detailed functions are shown in **Figure 4-3** and **Table 4-1**:

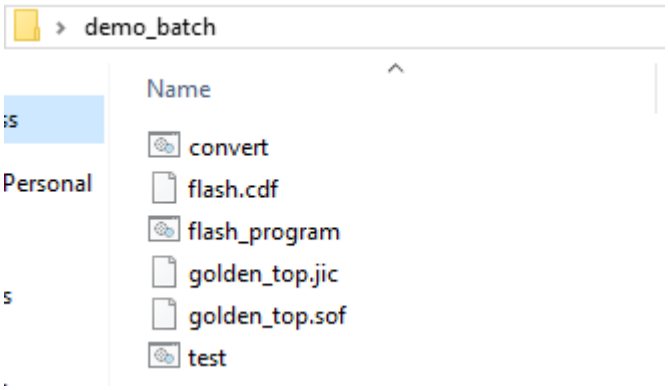


Figure 4-3 Batch files for programming flash

Table 4-1 File descriptions of the flash programming tool

File Name	*Functions*
convert.bat	This batch file can convert the golden_top.sof to golden_top.jic
flash.cdf	The flash programming configuration file
flash_program.bat	This batch file will program the .jic file into flash
golden_top.jic	The file that can be programmed into flash
golden_top.sof	The bitstream file of User's design
test.bat	This batch file can program the golden_top.sof into the batch file of FPGA

The following are the steps on how to use this tool to program the user's own bitstream file into flash.

1. Copy this demo_batch folder to host PC
2. Rename the user's own .sof file to *golden_top.sof* and overwrite it into the demo_batch folder to replace the original *golden_top.sof*.
3. Execute **convert.bat** to convert *golden_top.sof* to *golden_top.jic*.
4. Confirm that the MSEL switch on the board is set to AS Mode.
5. Connect the USB cable to USB blaster II connector of the Board.
6. Execute **flash_program.bat** to program the *golden_top.jic* file into flash.
7. Turn the power off and on of the board again to check whether your design is running on the FPGA.

Chapter 5

Additional Information

5.1 Getting Help

Here are the addresses where you can get help if you encounter problems:

■ Terasic Technologies

No.80, Fenggong Rd., Hukou Township, Hsinchu County 303035. Taiwan
Email: support@terasic.com
Web: www.terasic.com
DE25-Standard Development Kit Web: de25-standard.terasic.com

■ Revision History

Date	Version	Changes
2024.08	First publication	
2024.09	V1.1	Add SPI LCD and G-sensor demo