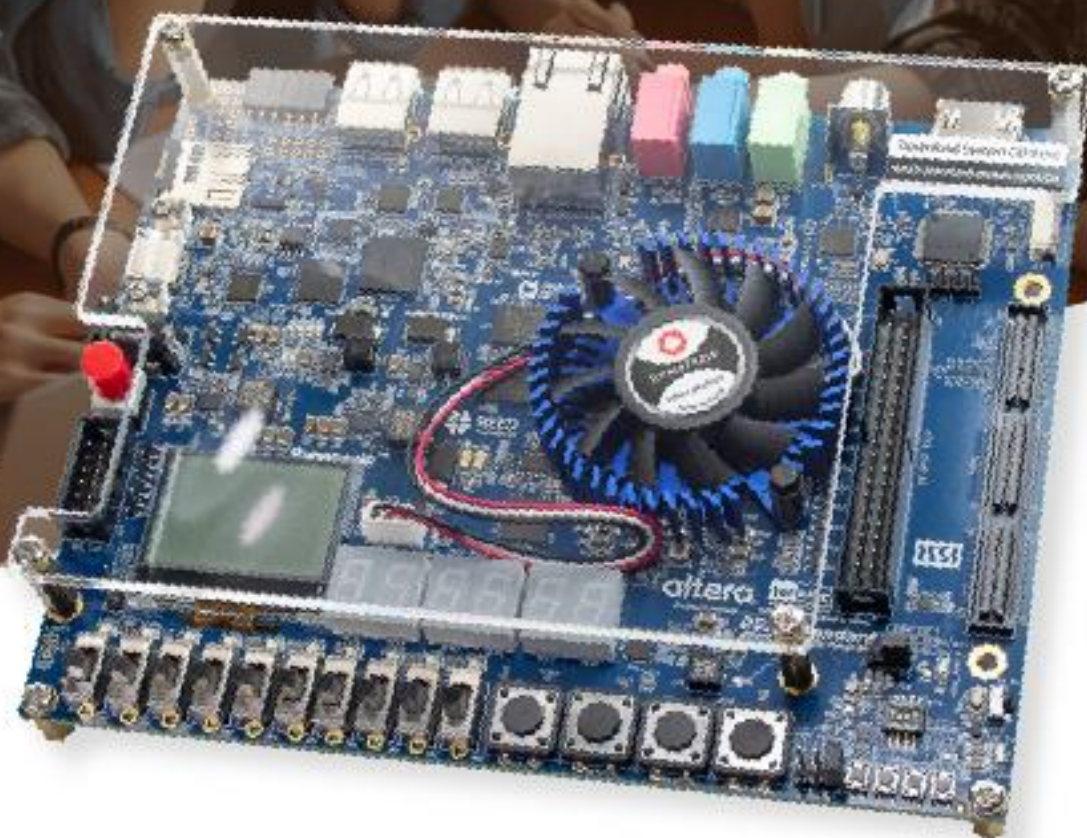


# DE25-Standard Development Kit



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

**Demonstration Manual**

FPGA

## Contents

<b>Chapter 1</b>	<b>Overview.....</b>	<b>3</b>
<b>Chapter 2</b>	<b>Examples For FPGA .....</b>	<b>4</b>
2.1	Basic Nios V control demo for Temperature/ Power/ Fan.....	4
2.2	Board Information IP .....	7
2.3	SDRAM Test in Verilog .....	11
2.4	DDR4 SDRAM Test by Nios V.....	12
2.5	Audio Recording and Playing .....	16
2.6	Karaoke Machine .....	19
2.7	IR Emitter LED and Receiver Demonstration .....	21
<b>Chapter 3</b>	<b>Examples for HPS SoC.....</b>	<b>27</b>
3.1	HPS LED/KEY.....	27
3.2	Network Socket .....	31
3.3	Build C/C++ Project.....	38
<b>Chapter 4</b>	<b>Additional Information .....</b>	<b>40</b>
4.1	Getting Help .....	40



# Chapter 1

---

## *Overview*

**T**his 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 three 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 24.1](#) 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.

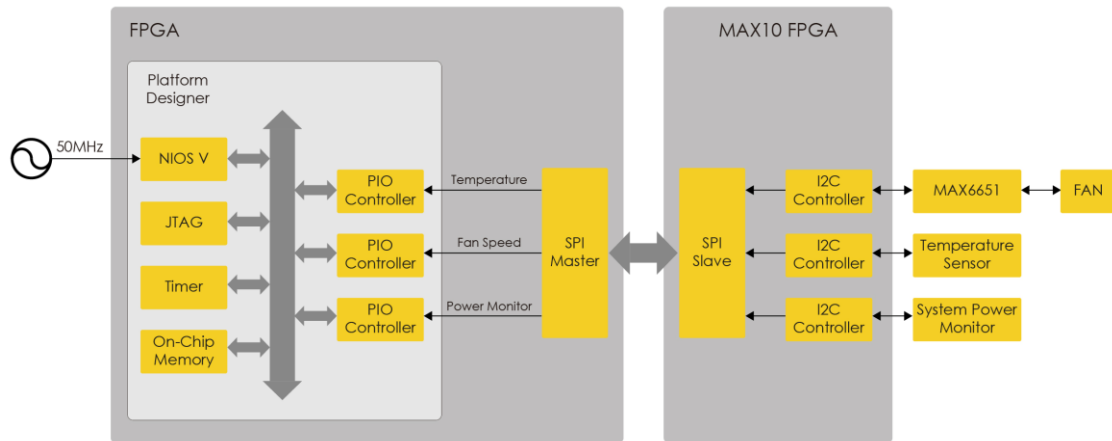
### 2.1 Basic Nios V control demo for Temperature/ Power/ Fan

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 Agilix FPGA can read it through the SPI interface.

In the Agilix 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 command line window, 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 choice: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 choice:

```

**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
- Bitstream used: Board\_Info.sof
- Software project directory: Board\_Info\software
- Demo batch file: Board\_Info\demo\_batch\test.bat, test.sh

### ■ 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. Power on the FPGA board.
3. Use the USB Cable to connect your PC and the FPGA board and install USB Blaster II driver if necessary.
4. Execute the demo batch file “test.bat” under the batch file folder: Board\_Info\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**.

```

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:

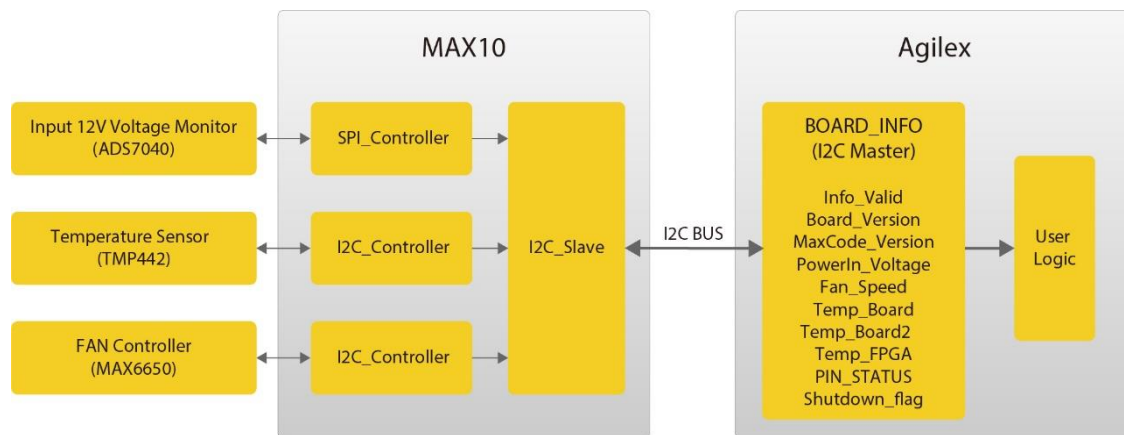
```

Figure 2-3 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**

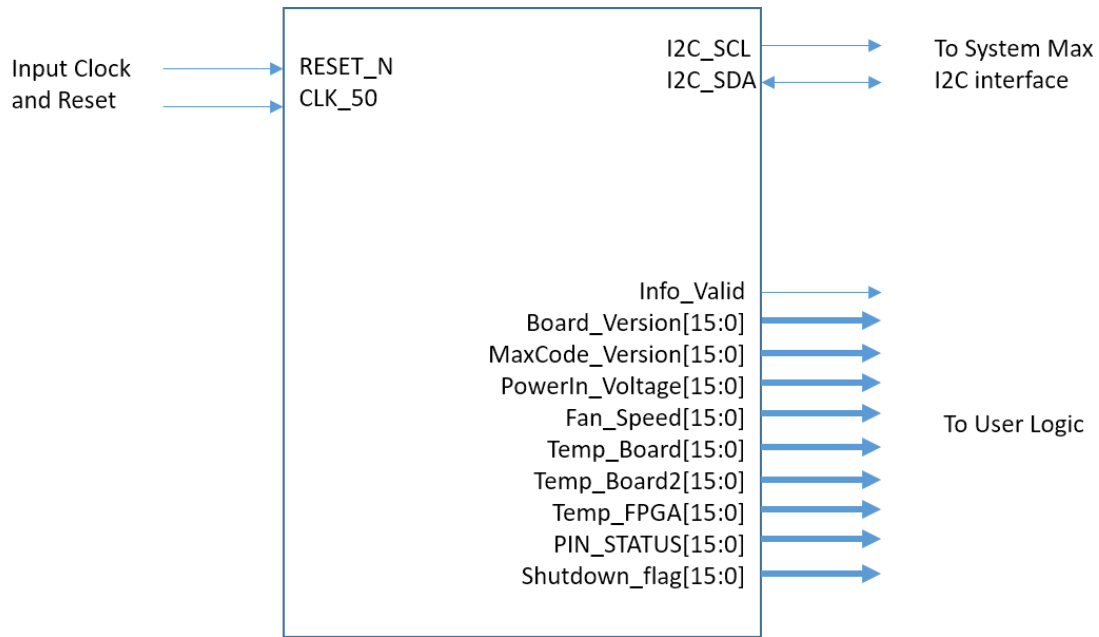
User can place a board information IP (**BOARD\_INFO.v** ; I2C master) provided by Terasic in the Agilex FPGA, the board status can be obtained via I2C interface from the system MAX FPGA and output to user logic.

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 board 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 <b>FPGA_I2C_SDAT</b> pin.
I2C_SCL	Output	1	Master I2C clock, I2C master output to salve. Please connect this signal to the <b>FPGA_I2C_SCLK</b> 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 <b>PowerIn_Voltage</b> 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 Board1 Temperature >=95°C BIT0: 1 ,when Board Temperature >=95°C
PIN_STATUS	Output	16	BIT8~15 : Reserved to 0. BIT7: <b>FAN_ALERT_n</b> , 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: <b>FPGA_CONF_DONE</b> ,FPGA Configure success, this bit is 1. BIT1: Reserved to 1. BIT0: Reserved to 1.

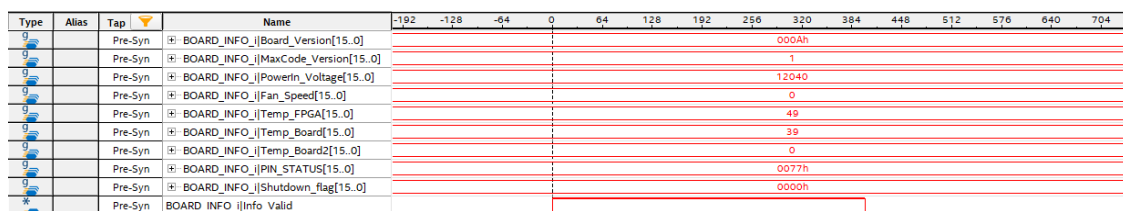


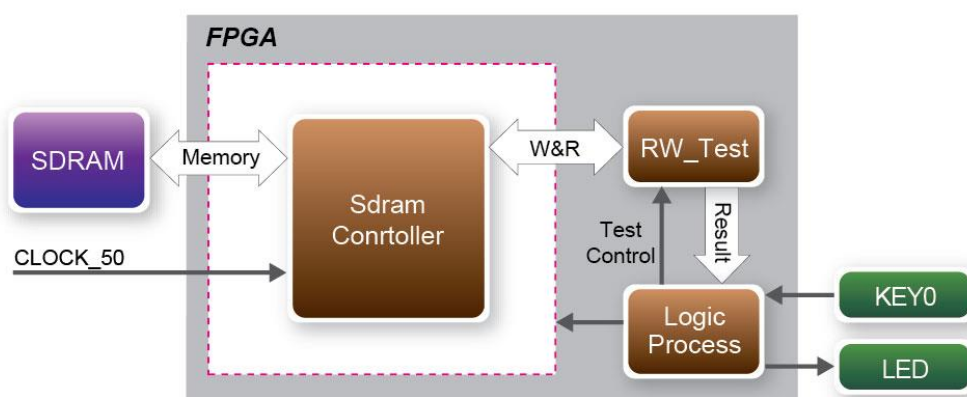
Figure 2-6 Waveform of the board status output

## 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 5 3.

### ■ Design Tools

- Quartus Prime 24.1 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:

- 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.
- 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 LEDR2 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

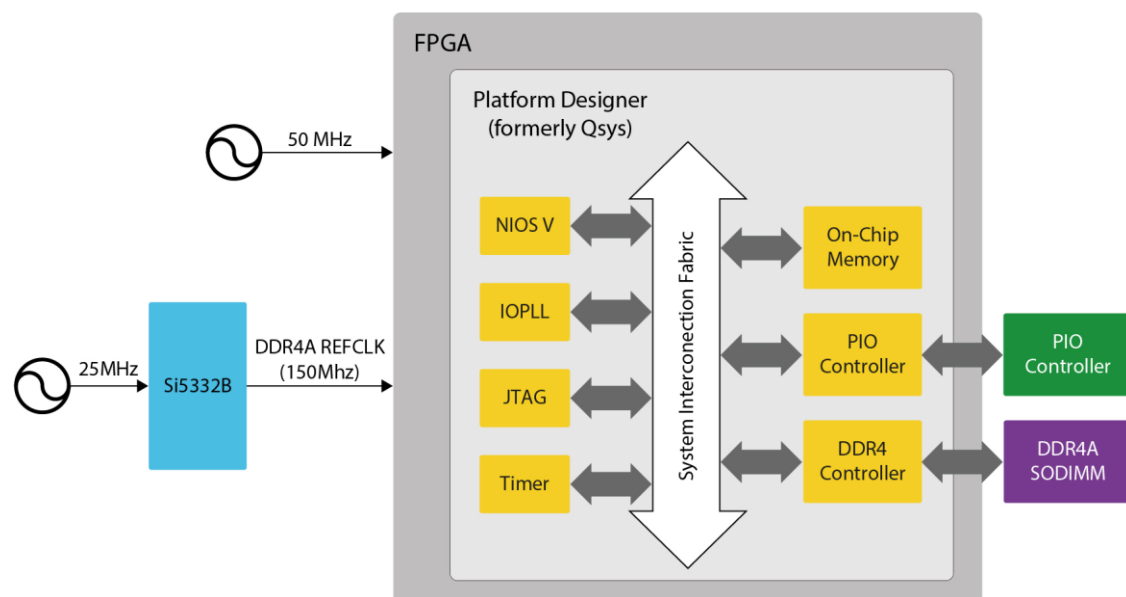
## 2.4 DDR4 SDRAM Test by Nios V

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 generator(Si5332B) are used. The OSC and clock generator 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)
A5ED065BB32AE4SR0	1200 (DDR4 2400)

## ■ Design Tools

- Quartus Prime 24.1 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. Power on the FPGA board.
3. Use a USB Cable to connect the PC and the FPGA board and install USB Blaster II driver if necessary.
4. Execute the demo batch file “test.bat” under the folder “NIOS\_DDR4\_Test\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 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.

```
0x00800004 in ?? ()
Loading section .entry, size 0x20 lma 0x800000
Loading section .exceptions, size 0x29c lma 0x800020
Loading section .text, size 0x22580 lma 0x8002bc
Loading section .rodata, size 0x1690 lma 0x822840
Loading section .rdata, size 0x1a00 lma 0x8258d0
Start address 0x008013f4, load size 153804
Transfer rate: 64 KB/sec, 11831 bytes/write.
[Inferior 1 (Remote target) detached]
INFO: Sending resume to OpenOCD.
===== Agilix 5 NIOS DDR4 Program =====
[0] DDR4 Full Test
[1] DDR4 Quick Test
Input your choice:0
===== DDR4 Test! Size=1GB =====

=====
Press any BUTTON on the board to start test [BUTTON-0] for continued test]
For continued test, press any BUTTON on the board to abort test.
=====> DDR4 Testing, Iteration: 1
== DDR4 Testing...
write...
```

Figure 2-9 Progress option [0] DDR4x2 Test

```
Loading section .entry, size 0x20 lma 0x800000
Loading section .exceptions, size 0x29c lma 0x800020
Loading section .text, size 0x22580 lma 0x8002bc
Loading section .rodata, size 0x1690 lma 0x822840
Loading section .rwdata, size 0x1a00 lma 0x8258d0
Start address 0x008013f4, load size 153804
Transfer rate: 67 KB/sec, 11831 bytes/write.
[Inferior 1 (Remote target) detached]
INFO: Sending resume to OpenOCD.
===== Agilx 5 NIOS DDR4 Program =====
[0] DDR4 Full Test
[1] DDR4 Quick Test
Input your choice 1
===== DDR Test! Size= 1GB =====

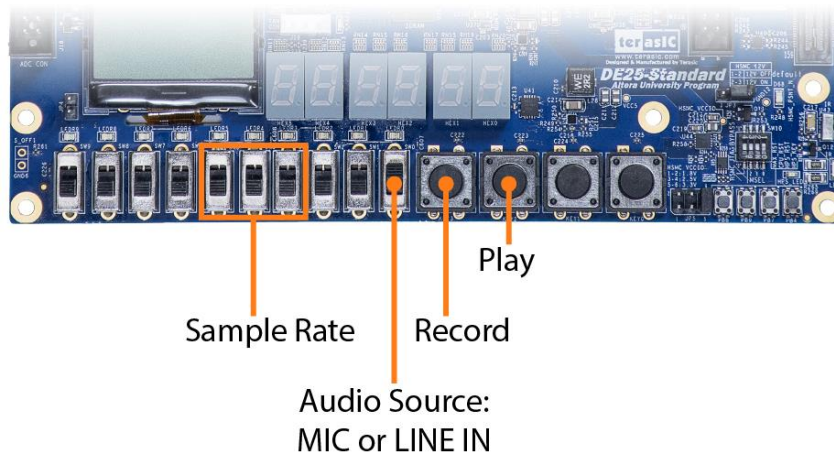
Press any BUTTON on the board to start test [BUTTON-0 for continued test].
====> DDR4 Testing, Iteration: 1
== DDR4 Testing...
PASS
DDR4 test:Pass, 4 seconds
====> DDR4 Testing, Iteration: 2
== DDR4 Testing...
PASS
DDR4 test:Pass, 4 seconds
====> DDR4 Testing, Iteration: 3
== DDR4 Testing...
PASS
DDR4 test:Pass, 4 seconds
====> DDR4 Testing, Iteration: 4
== DDR4 Testing...
```

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.
- SW1, SW2, and SW3 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.

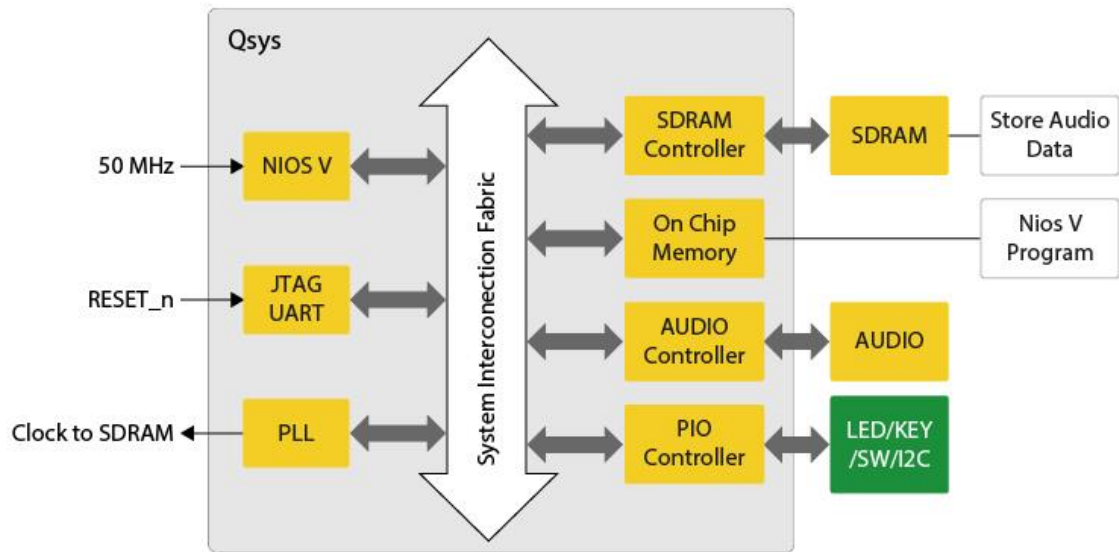


Figure 2-12 Block diagram of the audio recorder and player

## ■ 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
<b>SW0</b>	<b>Audio is from MIC-in</b>	<b>Audio is from Line-in</b>

**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

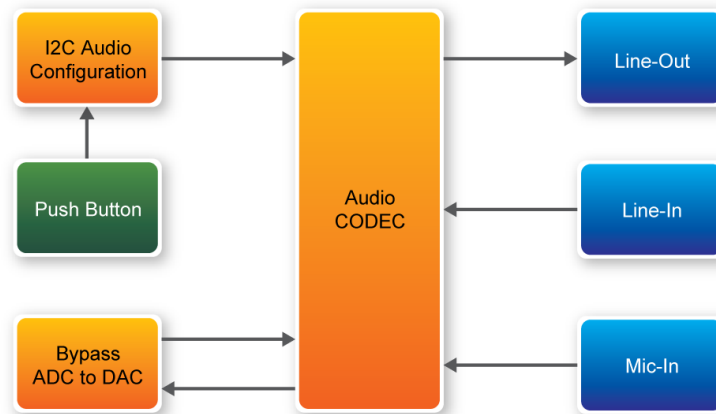
Note:

- (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

- 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.



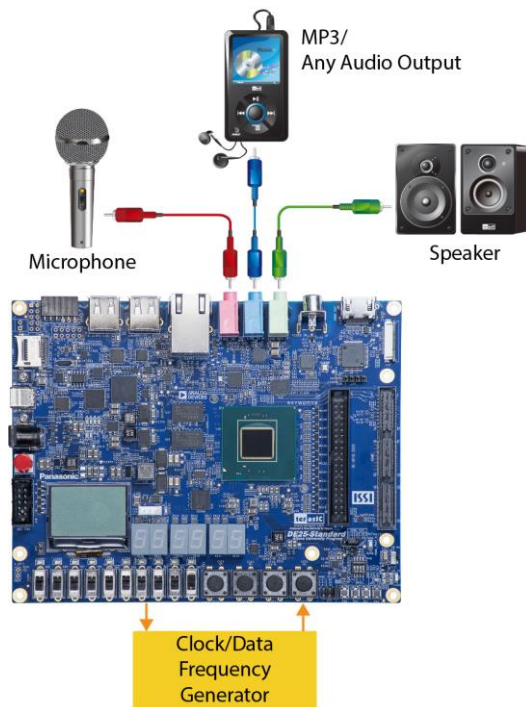
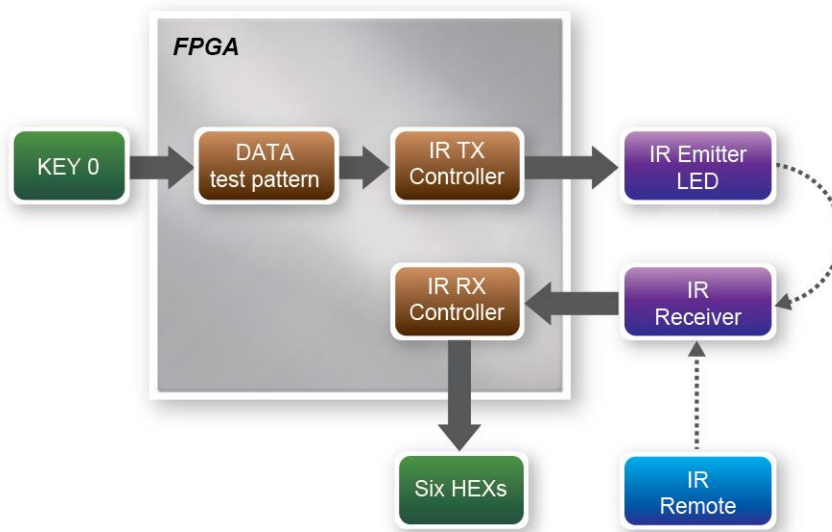


Figure 2-14 Setup for the Karaoke machine

## 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.

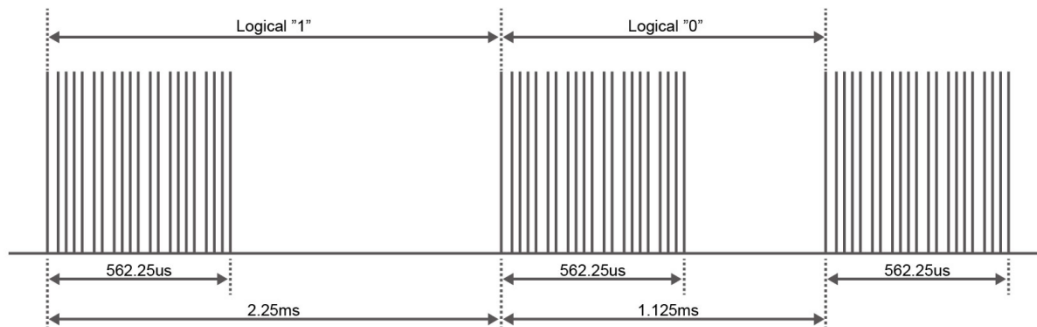
## ■ IR TX Controller

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).

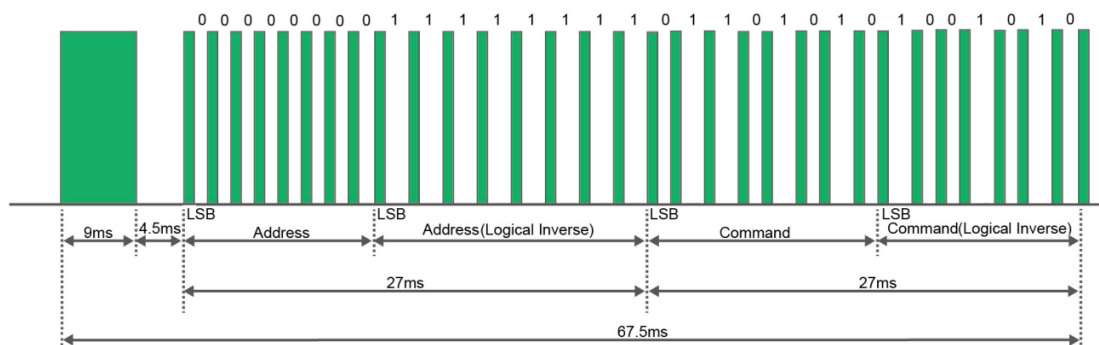
**Figure 2-16** shows the duration of logical “1” and “0”. Logical bits are transmitted as follows:

- 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

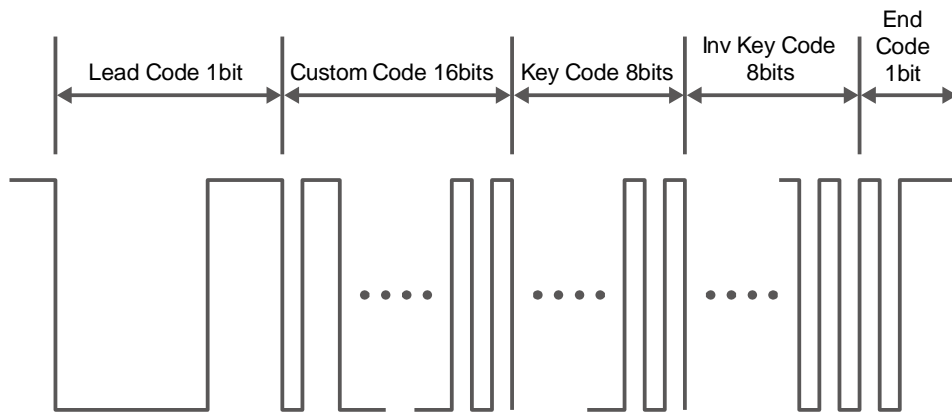
When a key on the remote control shown in Figure 2-18 is pressed, the remote control will emit a standard frame, as shown in Table 2-6. The beginning of the frame is the lead code, which represents the start bit, followed by the key-related information. The last bit end code represents the end of the 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
	0x0F		0x13		0x10		0x12
	0x01		0x02		0x03		0x1A
	0x04		0x05		0x06		0x1E
	0x07		0x08		0x09		0x1B
	0x11		0x00		0x17		0x1F
	0x16		0x14		0x18		0x0C

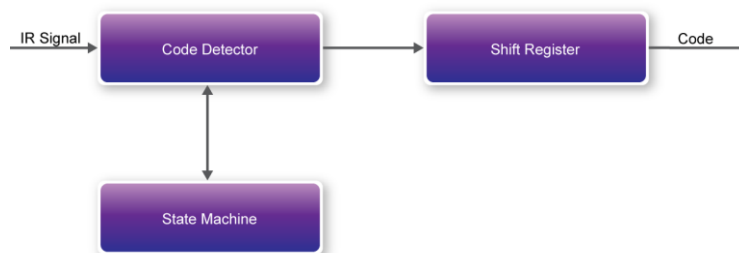


**Figure 2-19 The transmitting frame of the IR remote control**

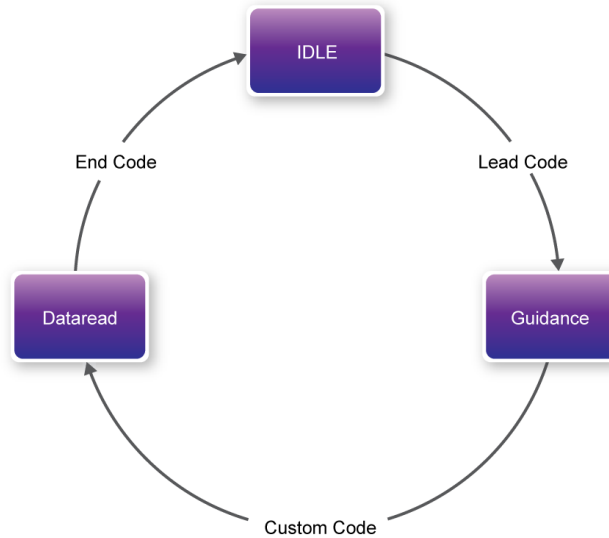
## ■ IR RX Controller

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 24.1 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)

## 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 Agilex Kit 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 Agilex Kit 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.

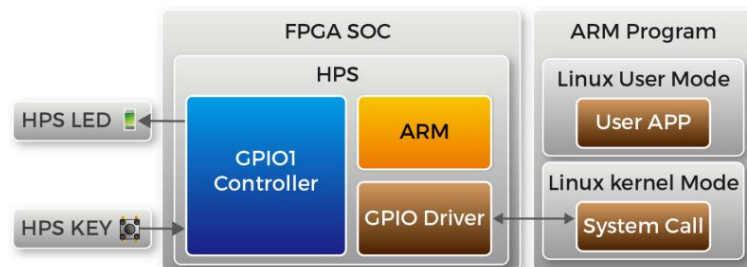
## ■ How to Read Button Status

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 3-22** 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 3-22 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 turn 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 3-23** 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 3-23 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/terasic**" folder in Linux.

3. Insert the Agilex SoC Board Linux BSP micro SD card into the Agilex SoC Board.
4. Power on the Agilex SoC Board.
5. Launch Putty to establish the connection between the UART port of Agilex SoC Board and the Host PC.
6. In the Putty UART terminal, type user name "terasic" and password "123" to login Linux.
7. Copy the executable file "hps\_led\_key" into the "/home/terasic" folder in Linux.
8. Type "sudo ./hps\_led\_key" in the UART terminal to start the program. Input password "123" if system query password for terasic.
9. You will see the key status is shown in Putty UART terminal as shown in **Figure 3-24**.
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.

```
terasic@localhost:~$ sudo ./hps_led_key
/sys/class/leds/hps_led0/brightness
key: 1
key: 1
key: 1
key: 1
key: 1
key: 1
key: 1
key: 1
key: 1
key: 1
key: 0
key: 1
key: 1
^C
terasic@localhost:~$
```

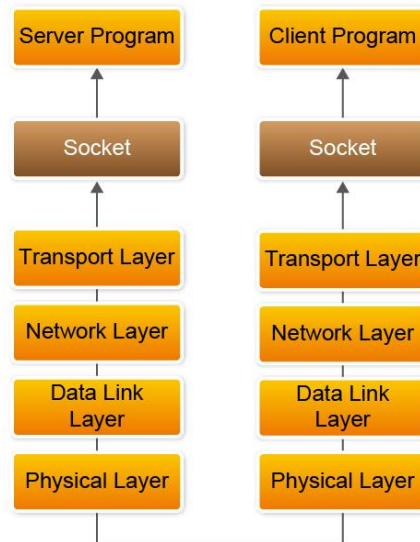
Figure 3-24 LED/KEY test

## 3.2 Network Socket

This demonstration shows how two remote application processes communication via socket in client-server model. Based on this design example, developers can make their Linux Application Software, run on SoC FPGA boards and easily communicate with other Hosts via a network socket.

### ■ Sockets

Sockets are the fundamental technology for programming software to communicate on the transport layer of networks shown in **Figure 3-25**. A socket provides a bidirectional communication endpoint for sending and receiving data with another socket. Socket connections normally run between two different computers on a LAN, or across the Internet, but they can also be used for interprocess communication on a single computer.



**Figure 3-25 Communicate on a network via a socket**

## ■ Client Server Model

Most interprocess communication uses the client server model. These terms refer to the two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server typically to make a request for information. A good analogy is a person who makes a phone call to another person.

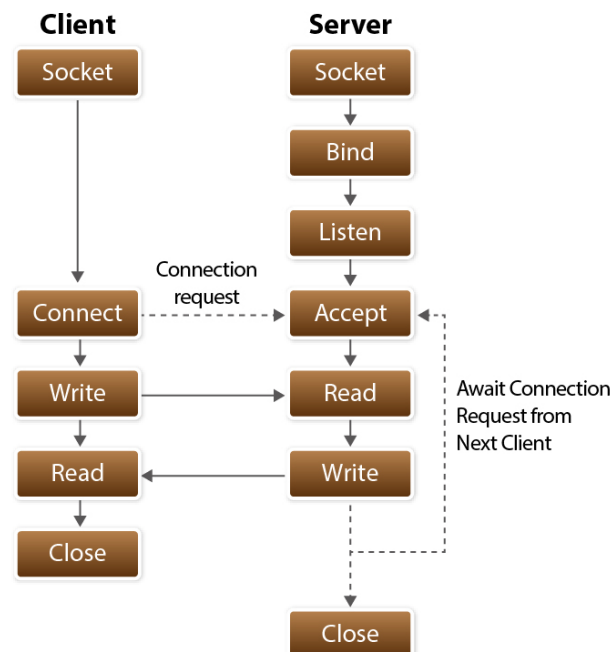
Notice that the client needs to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established.

Notice also that once a connection is established, both sides can send and receive information.

The system calls for establishing a connection which is somewhat different for the client and the server, but both involve the basic construct of a socket. A socket is one end of



an intercross's communication channel. The two processes each establish their own socket. **Figure 3-26** shows the communication diagram between the client and server.



**Figure 3-26 Client and Server communication**

The steps involved in establishing a socket on the client side are as follows:

- Create a socket with the **socket()** system call
- Connect the socket to the address of the server using the **connect()** system call
- Send and receive data. There are a number of ways to do this, but the simplest is to use the **read()** and **write()** system calls.

The steps involved in establishing a socket on the **server** side are as follows:

- Create a socket with the **socket()** system call
- Bind the socket to an address using the **bind()** system call. For a server socket on the Internet, an address consists of a port number on the Host machine.
- Listen for connections with the **listen()** system call
- Accept a connection with the **accept()** system call. This call typically blocks until a client connects with the server.
- Send and receive data. There are a number of ways to do this, but the simplest is to use the **read()** and **write()** system calls.

## ■ Example Code Explanation

The example design contains two projects. One is socket server project, and one is socket client project. The SOCK\_STREAM socket type is used in the design. The Linux Socket Library is used to provide socket functions, so remember to include the socket API header file – socket.h.

The major function of socket server program is to create a socket server based on the given port number and waiting a client to request to establish a connection. When a connection is established, the server is waiting for an incoming text message. When a message is received, it will show the receiver message on the console terminal, then send the message “I got your message” to the client socket, and then close the server program. **Figure 3-27** shows the socket relative code statement. In the program, **socket** API is used to create a SOCK\_STREAM socket, **bind** API is used to bind the socket to any incoming address and a specified port number. For connection, **listen** API is used to make the socket as a passive socket that is, as a socket that will be used to accept the incoming connection, and **accept** API is used to accept the incoming connection. The **accept** blocks until a client connects with the server. Data receiving and sending is implemented by the **read** and **write** API, and **close** is used to close the socket.

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);  
if (sockfd < 0)   
    error("ERROR opening socket");  
bzero((char *) &serv_addr, sizeof(serv_addr));  
portno = atoi(argv[1]);  
serv_addr.sin_family = AF_INET;  
serv_addr.sin_addr.s_addr = INADDR_ANY;  
serv_addr.sin_port = htons(portno);  
if (bind(sockfd, (struct sockaddr *) &serv_addr,  
    sizeof(serv_addr)) < 0)   
    error("ERROR on binding");  
listen(sockfd,5);  
clilen = sizeof(cli_addr);  
newsockfd = accept(sockfd,   
    (struct sockaddr *) &cli_addr,   
    &clilen);  
if (newsockfd < 0)   
    error("ERROR on accept");  
bzero(buffer,256);  
n = read(newsockfd,buffer,255);  
if (n < 0) error("ERROR reading from socket");  
printf("Here is the message: %s\n",buffer);  
n = write(newsockfd,"I got your message",18);  
if (n < 0) error("ERROR writing to socket");  
close(newsockfd);  
close(sockfd);
```

**Figure 3-27 Socket Server Code**

The major function of the socket client program is to create a connection based on given

Hostname (or IP address) and Host port. When a connection is established, it will show “Please enter the message:” message on console terminal to ask users to input a message. After get user’s input message, the message is sent to a remote socket server via the socket. If the remote server socket received the message, it will return a message “I got the message”. The client program will show the received message on the console terminal and exit the program. **Figure 3-28** shows the socket relative code statement. In the program, **socket** API is used to create a SOCK\_STREAM socket, **connect** API is used to connect the remove socket sever based on the given Hostname (or IPv4 Address) and port number. Data receiving and sending is implemented by **read** and **write** API, and **close** is used to **close** the socket.

```

sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
    error("ERROR opening socket");
server = gethostbyname(argv[1]);
if (server == NULL) {
    fprintf(stderr, "ERROR, no such host\n");
    exit(0);
}
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
bcopy((char *)server->h_addr,
      (char *)&serv_addr.sin_addr.s_addr,
      server->h_length);
serv_addr.sin_port = htons(portno);
if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
    error("ERROR connecting");
printf("Please enter the message: ");
bzero(buffer,256);
fgets(buffer,255,stdin);
n = write(sockfd,buffer,strlen(buffer));
if (n < 0)
    error("ERROR writing to socket");
bzero(buffer,256);
n = read(sockfd,buffer,255);
if (n < 0)
    error("ERROR reading from socket");
printf("%s\n",buffer);
close(sockfd);

```

**Figure 3-28 Socket Client Code**

## ■ Demonstration Source Code

The source code of the design example is located in the Demonstration folder as shown in **Figure 3-29**. The Demonstration folder contains three platform subfolders: **arm**, **linux** and **windows**. The project under the **arm** folder is designed for SoC FPGA board. The project under **linux** folder is designed for Linux running on Linux PC. The project under **windows** folder is designed for SoC EDS Shell running on Windows PC. Each platform subfolder contains socket\_client and socket\_server project folders.



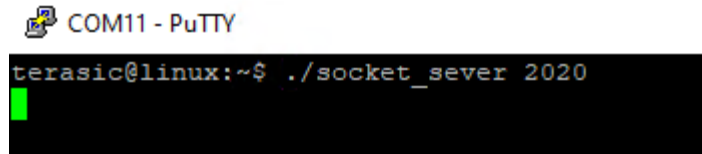
**Figure 3-29 Source Code Folder Tree**

The `socket_client` project includes a Makefile and a source file `main.c`. For different platforms, the Makefile content is different, but the `main.c` content is the same. The `socket_server` project has the file project architecture.

## ■ Demonstration Setup

Here we show the procedure to execute the socket client-server communication demonstration. In this setup procedure, the server program is running to Intel SoC FPGA board and the Socket Client is running on Windows PC.

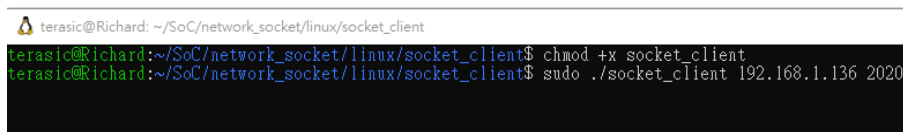
1. Connect the Agilex SoC Board to Network via Ethernet port (J9).
2. Connect a USB cable to the Micro USB connector (J10) on the board and the Host Windows PC.
3. Copy the executable file "**socket\_server**" into the microSD card under the `"/home/terasic"` folder in Linux. (board Linux BSP has pre-installed this code, so users can skip this copy action.)
4. Insert the Agilex SoC Board Linux BSP micro SD card into the Agilex SoC Board.
5. Power on the Agilex SoC Board.
6. Launch the Putty to connect Agilex SoC Board via the USB-to-UART link.
7. In the Putty, type user name "**terasic**" and password "**123**" to login Linux.
8. Type "**ifconfig**" to query the IP address which will be used in `socket_client`.
9. Type "**./socket\_server 2020**" to launch the server program with port number 2020 as shown in **Figure 3-30**. The port number can be any value between 2000 and 63500.



**Figure 3-30 Start Socket Server**

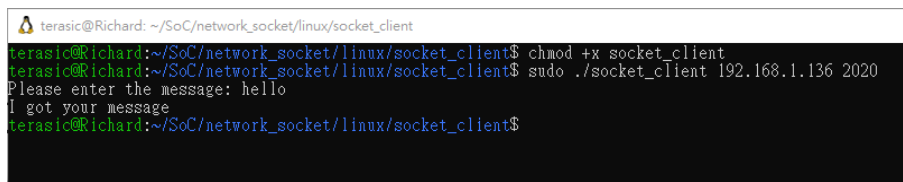
Here is the procedure to start the socket client program and communicate with the client server program:

1. Make sure the WSL is installed on your Windows and the Windows is connected to a network.
2. Launch WSL.
3. Copy the client program (linux/socket\_client/socket\_client) in the example kit to the WSL.
4. In the WSL, change the current directory to the directory where socket\_client is located.
5. Then, type “./socket\_client <ip address> 2020” to launch the client program to connect to the Host server with port number 2020 as shown in **Figure 3-31**.

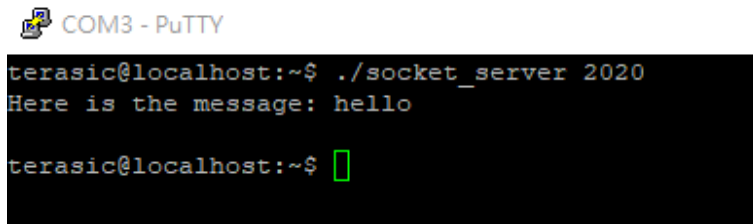


**Figure 3-31 Start Client Program**

6. If connection is established successfully, a prompt message “Please enter the message.” will appear. Type “**hello**”, then an echo message “**I got your message**” will be sent from the client server and shown on terminal as shown in **Figure 3-32**. At the same time, the socket server program will dump the received message at which point it is terminated as shown in **Figure 3-33**.



**Figure 3-32 Send Message in Client Program**



```
COM3 - PuTTY
terasic@localhost:~$ ./socket_server 2020
Here is the message: hello
terasic@localhost:~$
```

Figure 3-33 Server dumps received message

### 3.3 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:

1. Login Linux or WSL on Windows.
2. Type “cd ~”
3. 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](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)”
4. Type “tar xf gcc-arm-11.2-2022.02-x86\_64-aarch64-none-linux-gnu.tar.xz”
5. Type “export PATH=`pwd`/gcc-arm-11.2-2022.02-x86\_64-aarch64-none-linux-gnu/bin:\$PATH”
6. Type “export CROSS\_COMPILE=aarch64-none-linux-gnu-”
7. 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 3-34**.

```

terasic@Richard:~/SoC/hps_led_key$ make clean
rm -rf hps_led_key main.o led_lib.o gpio_lib.o *.objdump *.map
terasic@Richard:~/SoC/hps_led_key$ make
aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall -c main.c -o main.o
aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall -c led_lib.c -o led_lib.o
aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall -c gpio_lib.c -o gpio_lib.o
aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall main.o led_lib.o gpio_lib.o -o hps_led_key
aarch64-none-linux-gnu-nm hps_led_key > hps_led_key.map
terasic@Richard:~/SoC/hps_led_key$ ls
Makefile gpio_lib.c gpio_lib.h gpio_lib.o hps_led_key hps_led_key.map led_lib.c led_lib.h led_lib.o main.c main.o
terasic@Richard:~/SoC/hps_led_key$

```

**Figure 3-34 Build C/C++ Project**

# Chapter 4

## *Additional Information*

### 4.1 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](mailto:support@terasic.com)

Web: [www.terasic.com](http://www.terasic.com)

DE25-Standard Development Kit Web: DE25-STANDARD.[terasic.com](http://www.terasic.com)

■ **Revision History**

Date	Version	Changes
2024.08	First publication	