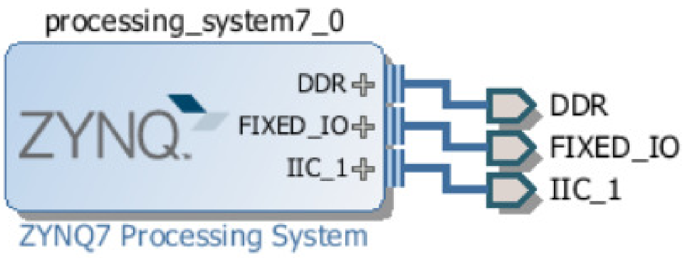
Name: ______________________ ID: ___________ Start Date: Monday, June 10, 2019
Name: ______________________ ID: ___________ Due Date: Wednesday, June 12, 2019

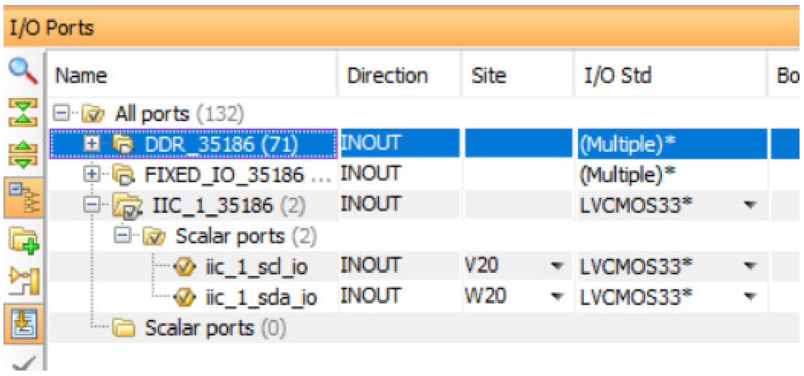
Software and Hardware Co-Design with Zybo, Spring 2019 HUST
Lab #9 Part 1: Reading Two TMP101 Temperature Sensors from MIO for One and SelectIO for the Other of Zybo

This is a group lab. Each group of two students should demonstrate your I2C TMP101 reading implementation on Zybo board and submit one pdf copy of report for this lab. Late lab submission will be accepted with a grade reduction of 10% for each day that it is late.

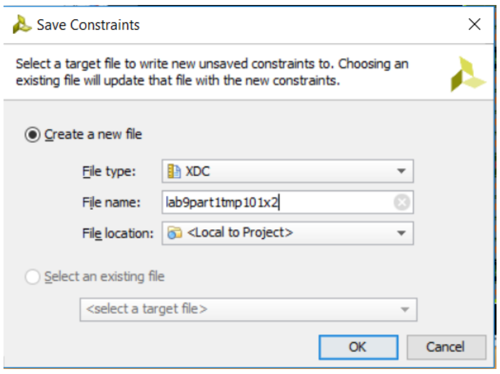
I. Objectives

(1) *Construct a Zynq 7010 circuit to implement the two I2C interfaces with the I2C controllers I2C 0 and I2C 1: one on MIO pins and the other on SelectIO pins. Your block design should have just one I2C interface connection for I2C Module 1 of Zynq. This interface needs to be connected to two Select I/O pins on Zybo. The following example connects SCL pin to JB3 (pin V20) and SDA pin to JB4 (pin W20) of Pmod JB connector. You are free to use other Pmod connector pins manually through I/O Ports menu. A XDC file will need to be created to save this assignment.*
You should verify that your I2C Module 1 is connected to MIO pins 14 and 15, which are accessible from Pmod JF of Zybo.




I/O Ports

Name	Direction	Site	I/O Std	Bo
All ports (132)				
DDR_35186 (71)	INOUT		(Multiple)*	
FIXED_IO_35186 ...	INOUT		(Multiple)*	
IIC_1_35186 (2)	INOUT		LVCMOS33*	
Scalar ports (2)				
iic_1_sd_io	INOUT	V20	LVCMOS33*	
iic_1_sda_io	INOUT	W20	LVCMOS33*	
Scalar ports (0)				

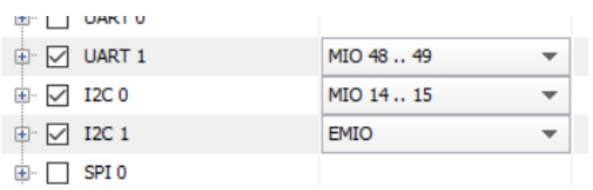

Save Constraints

Select a target file to write new unsaved constraints to. Choosing an existing file will update that file with the new constraints.

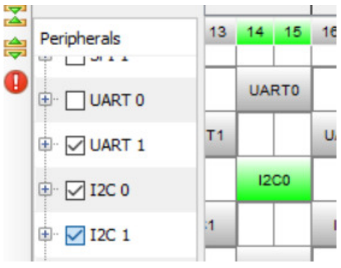
(●) Create a new file
File type: XDC
File name: lab9part1tmp101x2
File location: <Local to Project>

( ) Select an existing file
<select a target file>

OK Cancel



☐ UART 0	
☑ UART 1	MIO 48 .. 49
☑ I2C 0	MIO 14 .. 15
☑ I2C 1	EMIO
☐ SPI 0	

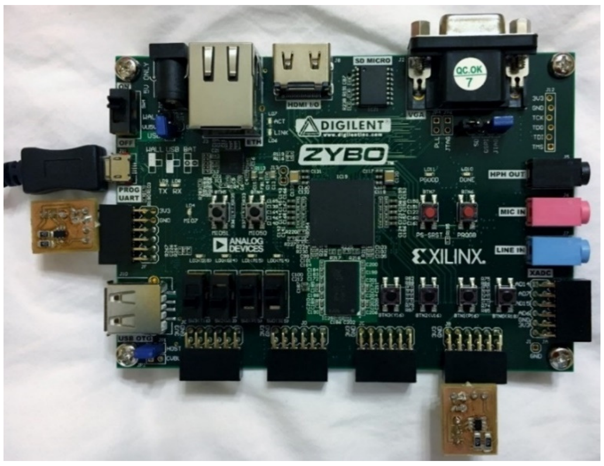


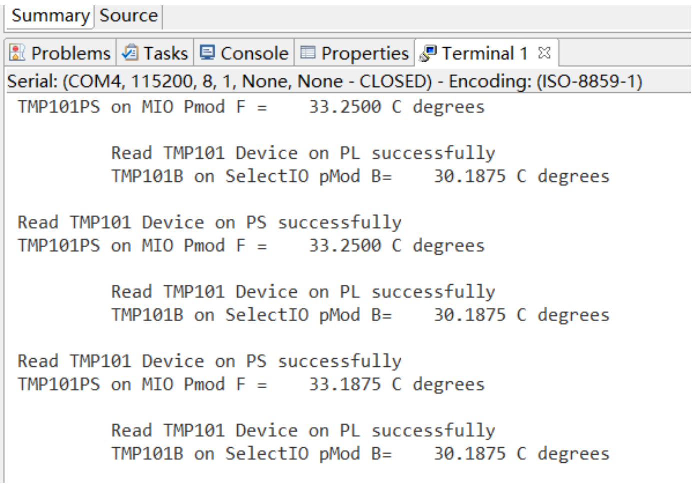
(2) *Connect two TMP101 breakout boards on Zybo so that they can be read by Zynq 7010 and their temperatures.*

(3) *Develop a C program for ARM I2C modules to read the TMP101 temperature sensors in 12 bit resolution and display their temperatures in both Celsius and Fahrenheit, also in 12 bit resolution, on the series terminal of the SDK. You will need to configure your TMP101 to transmit two byte temperature values of 12 bits.*

(4) *Use printf() for floating point display because xil_printf() does not support floating point. Format for floating point printing is 8.4f, where 8 is the total number of digits and 4 is the number of digits after decimal point. See an I2C example program from Xilinx in the appendix of this handout. This program can be used to test your hardware.*

(5) *Submit a memo to explain how you have implemented and verified your design and implementation. Include relevant screen captures and your source code in your memo.*



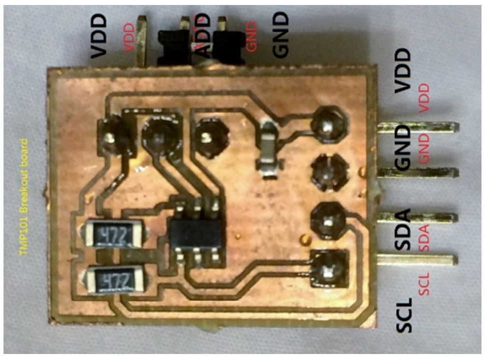


Serial: (COM4, 115200, 8, 1, None, None - CLOSED) - Encoding: (ISO-8859-1)

```
TMP101PS on MIO Pmod F =    33.2500 C degrees

        Read TMP101 Device on PL successfully
        TMP101B on SelectIO pMod B=    30.1875 C degrees

Read TMP101 Device on PS successfully
TMP101PS on MIO Pmod F =    33.2500 C degrees

        Read TMP101 Device on PL successfully
        TMP101B on SelectIO pMod B=    30.1875 C degrees

Read TMP101 Device on PS successfully
TMP101PS on MIO Pmod F =    33.1875 C degrees

        Read TMP101 Device on PL successfully
        TMP101B on SelectIO pMod B=    30.1875 C degrees
```

Figure 1. Zybo board with two TMP101 temperature sensors on Port F and Port B. UART 1 on Zynq must be selected to be on MIO 48 and 49.

The TMP101 breakout board in Port F is connected to the PS through MIO pins and is connected to Port F are as follows: SDK on JF9 (M14) and SDA on JF10 (M15). The TMP101 on Port B is connected to the PS through SelectIO pins and is connected so that JB3 (V20) is SCL and JB4 (W20) is SDA.



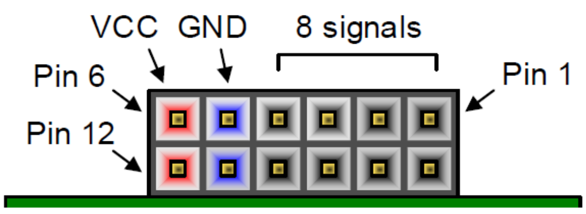
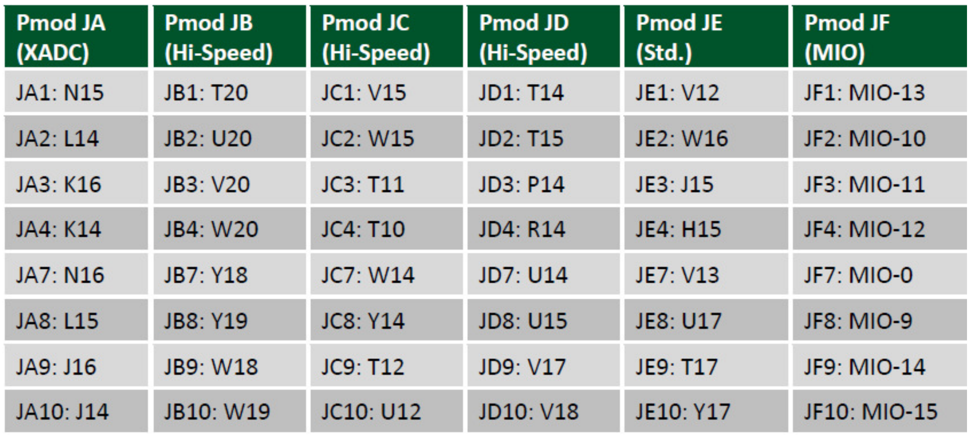
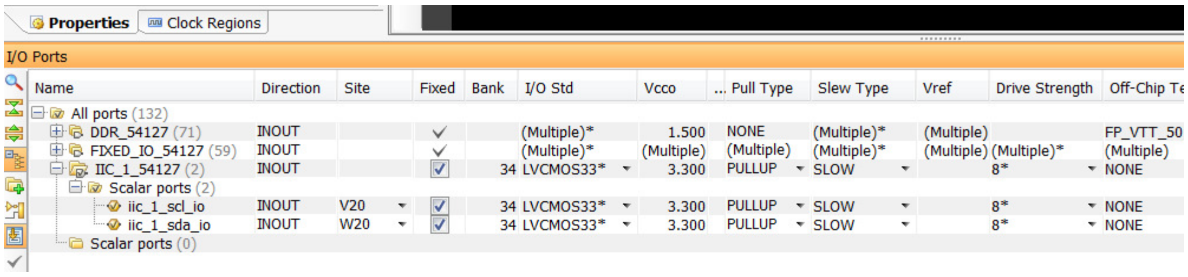


Figure 16. Pmod diagram.



Pmod JA (XADC)	Pmod JB (Hi-Speed)	Pmod JC (Hi-Speed)	Pmod JD (Hi-Speed)	Pmod JE (Std.)	Pmod JF (MIO)
JA1: N15	JB1: T20	JC1: V15	JD1: T14	JE1: V12	JF1: MIO-13
JA2: L14	JB2: U20	JC2: W15	JD2: T15	JE2: W16	JF2: MIO-10
JA3: K16	JB3: V20	JC3: T11	JD3: P14	JE3: J15	JF3: MIO-11
JA4: K14	JB4: W20	JC4: T10	JD4: R14	JE4: H15	JF4: MIO-12
JA7: N16	JB7: Y18	JC7: W14	JD7: U14	JE7: V13	JF7: MIO-0
JA8: L15	JB8: Y19	JC8: Y14	JD8: U15	JE8: U17	JF8: MIO-9
JA9: J16	JB9: W18	JC9: T12	JD9: V17	JE9: T17	JF9: MIO-14
JA10: J14	JB10: W19	JC10: U12	JD10: V18	JE10: Y17	JF10: MIO-15

Assign Pin V20 to I2C 1 clock pin and Pin W20 to I2C 1 data pin.



Name	Direction	Site	Fixed	Bank	I/O Std	Vcco	... Pull Type	Slew Type	Vref	Drive Strength	Off-Chip Te
All ports (132)											
DDR_54127 (71)	INOUT		✓		(Multiple)*	1.500	NONE	(Multiple)*	(Multiple)		FP_VTT_50
FIXED_IO_54127 (59)	INOUT		✓		(Multiple)*	(Multiple)	(Multiple)	(Multiple)*	(Multiple)	(Multiple)*	(Multiple)
IIC_1_54127 (2)	INOUT		✓	34	LVCMOS33*	3.300	PULLUP	SLOW		8*	NONE
Scalar ports (2)											
iic_1_scl_io	INOUT	V20	✓	34	LVCMOS33*	3.300	PULLUP	SLOW		8*	NONE
iic_1_sda_io	INOUT	W20	✓	34	LVCMOS33*	3.300	PULLUP	SLOW		8*	NONE
Scalar ports (0)											

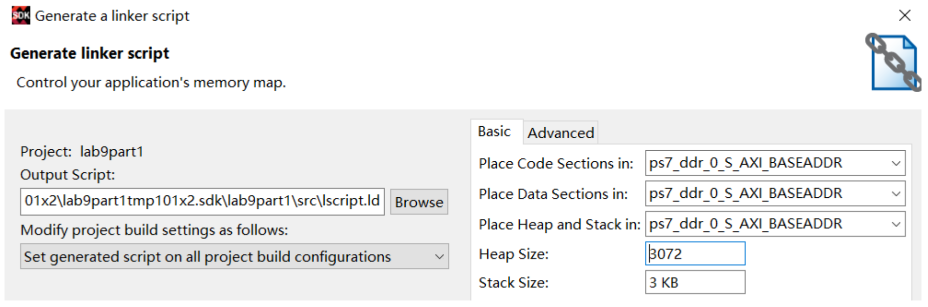
II. Software Development

Use the I2C example program in the appendix as a starting point, read two I2C TMP101 sensors with different addresses. Display their temperatures in both Celsius and Fahrenheit are displayed on the series terminal of the SDK.

III. Possible Errors

III.1 Printf() errors

Printf() uses stack to operate. If the stack size is too small, printf() may work strangely. Make sure the stack size is at last 3KB.



Generate linker script

Control your application's memory map.

Project: lab9part1

Output Script: 01x2\lab9part1tmp101x2.sdk\lab9part1\src\lscript.ld

Modify project build settings as follows: Set generated script on all project build configurations

Place Code Sections in: ps7_ddr_0_S_AXI_BASEADDR

Place Data Sections in: ps7_ddr_0_S_AXI_BASEADDR

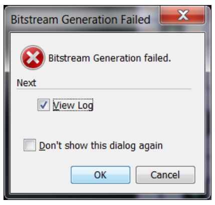
Place Heap and Stack in: ps7_ddr_0_S_AXI_BASEADDR

Heap Size: 3072

Stack Size: 3 KB

III.2 An Error on I/O Standard.

If you see the following error, change the I/O standard for pin V20 and W20 to LVCMOS33.

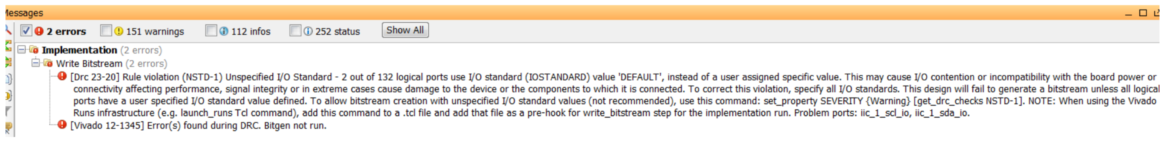


Bitstream Generation failed.

Next

[✓] View Log

[ ] Don't show this dialog again

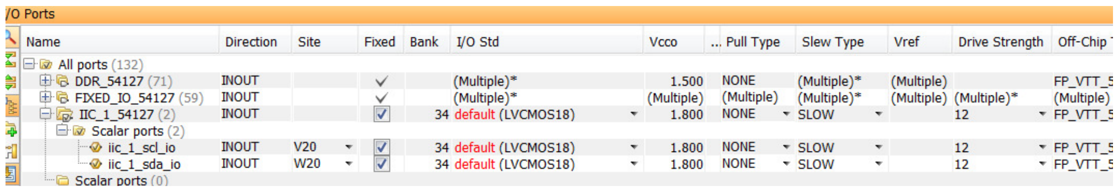


Messages

☑ 2 errors ☐ 151 warnings ☐ 112 infos ☐ 252 status Show All

```
Implementation (2 errors)
  Write Bitstream (2 errors)
    [Drc 23-20] Rule violation (NSTD-1) Unspecified I/O Standard - 2 out of 132 logical ports use I/O standard (IOSTANDARD) value 'DEFAULT', instead of a user assigned specific value. This may cause I/O contention or incompatibility with the board power or connectivity affecting performance, signal integrity or in extreme cases cause damage to the device or the components to which it is connected. To correct this violation, specify all I/O standards. This design will fail to generate a bitstream unless all logical ports have a user specified I/O standard value defined. To allow bitstream creation with unspecified I/O standard values (not recommended), use this command: set_property SEVERITY {Warning} [get_drc_checks NSTD-1]. NOTE: When using the Vivado Runs infrastructure (e.g. launch_runs Tcl command), add this command to a .tcl file and add that file as a pre-hook for write_bitstream step for the implementation run. Problem ports: iic_1_scl_io, iic_1_sda_io.
    [Vivado 12-1345] Error(s) found during DRC. Bitgen not run.
```

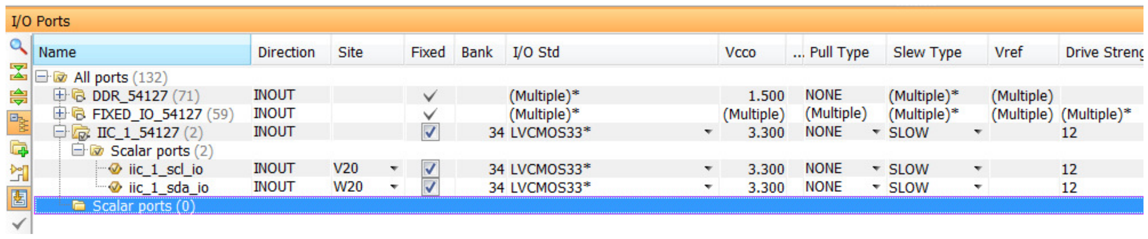
The default I/O standard is LVCMOS18 (1.8V) but the required I2C standard should be 3.3V.



I/O Ports

Name	Direction	Site	Fixed	Bank	I/O Std	Vcco	...	Pull Type	Slew Type	Vref	Drive Strength	Off-Chip T
All ports (132)												
DDR_54127 (71)	INOUT		✓		(Multiple)*	1.500		NONE	(Multiple)*	(Multiple)		FP_VTT_5
FIXED_IO_54127 (59)	INOUT		✓		(Multiple)*	(Multiple)		(Multiple)	(Multiple)*	(Multiple)	(Multiple)*	(Multiple)
IIC_1_54127 (2)	INOUT		☑	34	default (LVCMOS18)	1.800		NONE	SLOW		12	FP_VTT_5
Scalar ports (2)												
iic_1_scl_io	INOUT	V20	☑	34	default (LVCMOS18)	1.800		NONE	SLOW		12	FP_VTT_5
iic_1_sda_io	INOUT	W20	☑	34	default (LVCMOS18)	1.800		NONE	SLOW		12	FP_VTT_5
Scalar ports (0)												

Here is the correct I/O standard: LVCOMS33

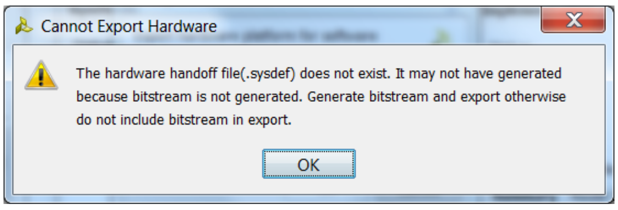


I/O Ports

Name	Direction	Site	Fixed	Bank	I/O Std	Vcco	...	Pull Type	Slew Type	Vref	Drive Streng
All ports (132)											
DDR_54127 (71)	INOUT		✓		(Multiple)*	1.500		NONE	(Multiple)*	(Multiple)	
FIXED_IO_54127 (59)	INOUT		✓		(Multiple)*	(Multiple)		(Multiple)	(Multiple)*	(Multiple)	(Multiple)*
IIC_1_54127 (2)	INOUT		☑	34	LVCMOS33*	3.300		NONE	SLOW		12
Scalar ports (2)											
iic_1_scl_io	INOUT	V20	☑	34	LVCMOS33*	3.300		NONE	SLOW		12
iic_1_sda_io	INOUT	W20	☑	34	LVCMOS33*	3.300		NONE	SLOW		12
Scalar ports (0)											

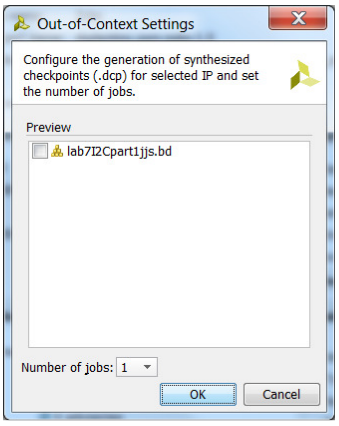
III.3 Out of Context settings error: the hardware handoff (.sysdef) does not exist

This error occurs when out of context settings is checked. Disabling "out of context settings" would fix this error.



Cannot Export Hardware

The hardware handoff file(.sysdef) does not exist. It may not have generated because bitstream is not generated. Generate bitstream and export otherwise do not include bitstream in export.



Out-of-Context Settings

Configure the generation of synthesized checkpoints (.dcp) for selected IP and set the number of jobs.

Preview

☐ lab7I2Cpart1jjs.bd

Number of jobs: 1

IV. Appendix xiicps_polled_master_example.c

The following example, xiicps_polled_master_example.c is from C:\Xilinx\SDK\2015.2\data\embeddedsw\XilinxProcessorIPLib\drivers\iicps_v3_0\examples.

```
/* Copyright (C) 2010 - 2014 Xilinx, Inc.  All rights reserved. */
/**
* @file xiicps_polled_master_example.c
* A design example of using the device in polled mode as master.
* The example uses buffer size 132. Please set the send buffer of the
* Aardvark device to be continuous 64 bytes from 0x00 to 0x3F.
* <pre> MODIFICATION HISTORY:
* Ver   Who Date     Changes
* ----- --- -------- -----------------------------------------------
* 1.00a jz  01/30/10 First release
*/
#include "xparameters.h"
#include "xiicps.h"
#include "xil_printf.h"
/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user
 * can easily change all the needed parameters in one place.
 */
#define IIC_DEVICE_ID                    XPAR_XIICPS_0_DEVICE_ID

/* The slave address to send to and receive from.*/
#define IIC_SLAVE_ADDR                   0x55
#define IIC_SCLK_RATE                    100000

/*The following constant controls the length of the buffers to be sent
 * and received with the IIC.  */
#define TEST_BUFFER_SIZE    132

/***************** Function Prototypes *****************/

int IicPsMasterPolledExample(u16 DeviceId);
/******************* Variable Definitions **************/

XIicPs Iic;                  /**< Instance of the IIC Device */

/* The following buffers are used in this example to send and
 * receive data with the IIC. */
u8 SendBuffer[TEST_BUFFER_SIZE];  /**< Buffer for Transmitting Data */
u8 RecvBuffer[TEST_BUFFER_SIZE];  /**< Buffer for Receiving Data */

/**********************************************/
* Main function to call the polled master example.
* @param        None.
* @return       XST_SUCCESS if successful, XST_FAILURE if unsuccessful.
* @note                 None.
**********************************************/
int main(void)
{
        int Status;

        xil_printf("IIC Master Polled Example Test \r\n");
```

```
/* Run the Iic polled example in master mode, specify the Device
 * ID that is specified in xparameters.h. */
	Status = IicPsMasterPolledExample(IIC_DEVICE_ID);
	if (Status != XST_SUCCESS) {
		xil_printf("IIC Master Polled Example Test Failed\r\n");
		return XST_FAILURE;
	}

xil_printf("Successfully ran IIC Master Polled Example Test\r\n");
	return XST_SUCCESS;
}

/**************************************************/
* This function sends data and expects to receive data from
* slave as modular of 64.
* This function uses interrupt-driven mode of the device.
* @param	DeviceId is the Device ID of the IicPs Device and is the
*		XPAR_<IICPS_instance>_DEVICE_ID value from xparameters.h
* @return	XST_SUCCESS if successful, otherwise XST_FAILURE.
* @note		None.
*************************************************/
int IicPsMasterPolledExample(u16 DeviceId)
{
	int Status;
	XIicPs_Config *Config;
	int Index;

/* Initialize the IIC driver so that it's ready to use
 * Look up the configuration in the config table, then initialize it*/
	Config = XIicPs_LookupConfig(DeviceId);
	if (NULL == Config) {
		return XST_FAILURE;
	}
	Status = XIicPs_CfgInitialize(&Iic, Config, Config->BaseAddress);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}

/* Perform a self-test to ensure that the hardware
 * was built correctly. */
	Status = XIicPs_SelfTest(&Iic);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}

/* Set the IIC serial clock rate. */
	XIicPs_SetSClk(&Iic, IIC_SCLK_RATE);

/* Initialize the send buffer bytes with a pattern to send and the
 * the receive buffer bytes to zero to allow the receive data to be
 * verified. */
	for (Index = 0; Index < TEST_BUFFER_SIZE; Index++) {
		SendBuffer[Index] = (Index % TEST_BUFFER_SIZE);
		RecvBuffer[Index] = 0;
	}
/* Send the buffer using the IIC and ignore the number of bytes sent
 * as the return value since we are using it in interrupt mode. */
	Status = XIicPs_MasterSendPolled(&Iic, SendBuffer,
                          TEST_BUFFER_SIZE, IIC_SLAVE_ADDR);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }
/* Wait until bus is idle to start another transfer. */
        while (XIicPs_BusIsBusy(&Iic)) {
                /* NOP */
        }
        Status = XIicPs_MasterRecvPolled(&Iic, RecvBuffer,
                          TEST_BUFFER_SIZE, IIC_SLAVE_ADDR);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }
/* Verify received data is correct. */
         for(Index = 0; Index < TEST_BUFFER_SIZE; Index ++) {

/* Aardvark as slave can only set 64 bytes for output */
                if (RecvBuffer[Index] != Index % 64) {
                        return XST_FAILURE;
                }
        }
        return XST_SUCCESS;
}
```