Lab #8 Reading Two TMP101 Temperature Sensors from MIO for One and SelectIO for the Other of Zybo

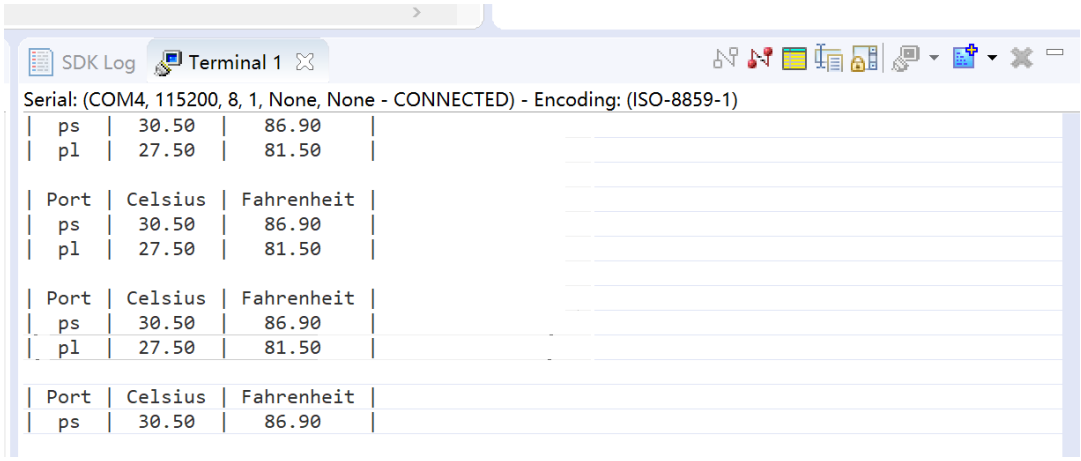
Member: Cunyang Liu , Zeming Jin Date:2021/08/20

Summary


```
    Construct a Zynq 7010 circuit to implement the two I2C interfaces with the I2C controllers I2C 0 and I2C 1: I2C 0 on MIO pins 14 and 15 and I2C 1on SelectIO pins JB3 and JB4. This interface needs to be connected to two Select I/O pins on Zybo.
    There are two I2C interfaces, controller I2C 0 and I2C 1. one is on MIO pins and the other is on SelectIO pins. And we display temperature on the series terminal of SDK.
```


Details(screenshot and code)

- Screenshot of vivado SDK terminal



- source code

```c
// lab8I2ConARM_ZMJ_CYL.c

/***************************************************************************
 * @data    2021-8-20
 * @brief   Finishing ARM I2C modules
 * @author  Cunyang Liu, Zeming Jin
 ***************************************************************************/

// include files
#include <stdio.h>
#include <xiicps.h>
#include "xparameters.h"

// parameters define
#define PS_I2C_DEVICE_ID     XPAR_PS7_I2C_0_DEVICE_ID
#define PS_DATA_RATE         100000 // 100KHz
#define PS_TMP101_ADDRESS    0b1001001 // floating
```

```c
#define PS_TMP101_RESOLUTION 0b01100000 // 12 bit resolution for TMP101

#define PL_I2C_DEVICE_ID     XPAR_PS7_I2C_1_DEVICE_ID
#define PL_DATA_RATE         200000 // 200KHz
#define PL_TMP101_ADDRESS    0b1001001 // ground
#define PL_TMP101_RESOLUTION 0b00100000 // 10 bit resolution for TMP101

XIicPs i2c_ps, i2c_pl; // device instances

#define DELAYLOOPCOUNT       80000000
// function prototype
int ReadTemperature(XIicPs *I2Cinstance, float *temperature, int I2C_address);

/*****************************************************************
* This function ConfigureTMP101() configures an I2C TMP101 instance.
*
* @param    I2Cinstance is a pointer to an I2C instance.
* @param    I2CAddress is the address of I2C TMP101 chip.
* @param    TEMPresolution is the resolution of TMP101 temperature as 0b0RR00000
*
* @return
*       - XST_SUCCESS if everything went well.
*       - XST_FAILURE if failed.
*
*****************************************************************/

int ConfigureTMP101(XIicPs *I2Cinstance, u8 I2CAddress, u8 TEMPresolution)
{
    int status;
    u8 SetPointertoZero = 0b00000000;
    u8 SetResolution[] = {0b00000001, 0b00000000};

    SetResolution[1]=TEMPresolution;

    // Set resolution
    status = XIicPs_MasterSendPolled(I2Cinstance, SetResolution, 2, I2CAddress);
    if (status != XST_SUCCESS) {
        printf("\r\n-- Failed to set resolution");
        return XST_FAILURE;
    }

    // set pointer back to 0x00 to point to the temperature value
    status = XIicPs_MasterSendPolled(I2Cinstance, &SetPointertoZero, 1, I2CAddress);
    if (status != XST_SUCCESS) {
        printf("\r\n-- Failed to set pointer back to 0x00 to point to the temperature value");
        return XST_FAILURE;
    }

    return XST_SUCCESS;
} // end ConfigureTMP101()

/*****************************************************************
* This function ConfigureI2Cinstance() configures an I2C instance.
```

```
*
* @param    I2Cinstance is a pointer to an I2C instance.
* @param    I2CdeviceID is the ID of I2C device defined in xparamenters.h.
* @param    I2C_ClockRate is clock frequency of I2C TMP101 in Hz.
*
* @return
*       - XST_SUCCESS if everything went well.
*       - XST_FAILURE if failed.
*
*****************************************************************/
int ConfigureI2Cinstance(XIicPs *I2Cinstance, int I2CdeviceID, int
I2C_ClockRate) {
    int status;
    XIicPs_Config *I2C_config;
    // Call lookup table to find I2C hardware module for I2CdeviceID
    I2C_config = XIicPs_LookupConfig(I2CdeviceID);
    if (NULL == I2C_config) {
        printf("\r\n-- Failed to find I2CdeviceID %d", I2CdeviceID);
        return XST_FAILURE;
    }
    // Initialize I2C instance. Return XST_FAILTURE if failed.
    status = XIicPs_CfgInitialize(I2Cinstance, I2C_config, I2C_config-
>BaseAddress);
    if (status != XST_SUCCESS) {
        printf("\r\n-- Failed to initialize I2C instance");
        return XST_FAILURE;
    }

    // Self test. Return XST_FAILTURE if failed
    status = XIicPs_SelfTest(I2Cinstance);
    if (status != XST_SUCCESS) {
        printf("\r\n-- Failed to self test");
        return XST_FAILURE;
    }

    // Set I2C clock frequency. Return XST_FAILTURE if failed
    status = XIicPs_SetSClk(I2Cinstance, I2C_ClockRate);
    if (status != XST_SUCCESS) {
        printf("\r\n-- Failed to set I2C clock frequency");
        return XST_FAILURE;
    }

    // Wait when I2Cdevice is busy until bus is idle to start another
transfer
    while (XIicPs_BusIsBusy(I2Cinstance)) {

    }

    return XST_SUCCESS;
} // end ConfigureI2Cinstance()

int main() {
    float temp_ps, temp_pl;
    unsigned int loopcount;
    int status = XST_FAILURE;
    printf("\r\n\n -- Start Lab #8 I2C for TMP101 --\r\n");

    // configure I2C instance and tmp101 board on PS port
```

```
125     status = ConfigureI2Cinstance(&i2c_ps, PS_I2C_DEVICE_ID, PS_DATA_RATE);
126     if(status == XST_FAILURE) printf("\r\n-- Failed to configure I2C
    instance on PS.");
127     status = ConfigureTMP101(&i2c_ps, PS_TMP101_ADDRESS,
    PS_TMP101_RESOLUTION);
128     if(status == XST_FAILURE) printf("\r\n-- Failed to configure TMP101 on
    PS.");
129
130     // configure I2C instance and tmp101 board on PL port
131     status = ConfigureI2Cinstance(&i2c_pl, PL_I2C_DEVICE_ID, PL_DATA_RATE);
132     if(status == XST_FAILURE) printf("\r\n-- Failed to configure I2C
    instance on PL.");
133     status = ConfigureTMP101(&i2c_pl, PL_TMP101_ADDRESS,
    PL_TMP101_RESOLUTION);
134     if(status == XST_FAILURE) printf("\r\n-- Failed to configure TMP101 on
    PL.");
135
136     while(1) {
137         // Read tmp101 board and calculate temperature value on PS port
138         status = ReadTemperature(&i2c_ps, &temp_ps, PS_TMP101_ADDRESS);
139         if(status == XST_FAILURE) printf("\r\n-- Failed to read TMP101 from
    PS on bottom row of Connector JF.");
140
141         // Read tmp101 board and calculate temperature value on PL port
142         status = ReadTemperature(&i2c_pl, &temp_pl, PL_TMP101_ADDRESS);
143         if(status == XST_FAILURE) printf("\r\n-- Failed to read TMP101 from
    PL on top row of Connector JB.");
144
145
146         // printing floating numbers is not supported by xil_printf().
147         // Use printf() from <stdio.h> to print floating points.
148         // Mixing Xil_printf() with printf() may cause some printf() being
    dropped.
149         // Display temperature in floating point number with 4 digits after
    decimal point
150         printf("\n\r");
151         printf("\r\n| Port | Celsius | Fahrenheit |");
152         printf("\r\n|  ps  |  %.2f  |   %.2f    |", temp_ps, temp_ps * 1.8
    + 32);
153         printf("\r\n|  pl  |  %.2f  |   %.2f    |", temp_pl, temp_pl * 1.8
    + 32);
154
155         // delay loop to pause for a while
156         for(loopcount=0; loopcount < DELAYLOOPCOUNT; ++loopcount) {}
    //delay time
157     } // while(1)
158     while(1); // hold just in case
159 } // end main()
160
161 /*************************************************************
162 * This function ReadTemperature() reads temperature of an TMP101.
163 *
164 * @param    I2Cinstance is a pointer to an I2C instance.
165 * @param    temperature is the returned floaing point temperature value.
166 * @param    I2C_address is the address of I2C TMP101 chip.
167 *
168 * @return
169 *       - XST_SUCCESS if everything went well.
```

```
*       - XST_FAILURE if failed.
*
******************************************************************/
int ReadTemperature(XIicPs *I2Cinstance, float *temperature, int
I2C_address) {
    int status;
    u8 temp[2]; //2 byte temperature
    u8 SetPointertoZero = 0b00000000;
    // set pointer back to 0x00 to point to the temperature value
    XIicPs_MasterSendPolled(I2Cinstance, &SetPointertoZero, 1,
I2C_address);

    // Read temperature. Return XST_FAILURE if failed
    status = XIicPs_MasterRecvPolled(I2Cinstance, temp, 2, I2C_address);
    if (status != XST_SUCCESS) {
        printf("\r\n-- Failed to read temperature");
        return XST_FAILURE;
    }

    // Convert temperature to floating number
    *temperature = temp[0] + ((float)(temp[1] >> 7)) / 2;

    return XST_SUCCESS;
} // end of ReadTemperature()
```