I2C Master (Using 1-bit Ports) Component

Document Number: XM002037A

Publication Date: 2014/4/30

XMOS

Table of Contents

1 Resource Requirements

IN THIS CHAPTER

The following resources on the xCORE Tile are used:

1.1 Resources

Operation	Resource Type	Number required	Notes
I2C master SCL	1 bit port (output)	1	I2C master clock, needs external pull-up
I2C Master SDA	1 bit port	1	data line, needs external pull-up
I2C IO timing	timers	2	

1.2 Memory

The whole library uses approximately 1500 Bytes.

2 Evaluation Platforms

IN THIS CHAPTER

2.1 Recommended Hardware

This module may be evaluated using the sliceKIT Modular Development Platform, available from digikey. Required board SKUs are:

- XP-SKC-L16 (sliceKIT L16 Core Board) plus XA-SK-GPIO plus XA-SK-XTAG2 (sliceKIT xTAG adaptor) plus xTAG-2 (debug adaptor), OR
- XK-SK-L16-ST (sliceKIT Starter Kit, containing all of the above).

2.2 Demonstration Application

Example usage of this module can be found within the xSOFTip suite as follows:

- Package: sw_gpio_examples
- Application: app_slicekit_com_demo

3 Hardware Requirements

This module requires 1K pull-up resistors be present on SCL and SDA externally.

4 Programming Guide

IN THIS CHAPTER

The module can be instantiated with different bus speeds with each instantiation, and comprises four functions that implement I2C master.

4.1 Structure

All of the files required for operation are located in the `module_i2c_master/src` directory. The files that are need to be included for use of this component in an application are:

File	Description
`i2c.h`	Header file for simplified I2C master module and API interfaces.
`i2c-mm.xc`	Module function library

4.2 Types

r_i2c

Struct that holds the data for instantiating the I2C module - it just comprises two ports (the clock port and the data port), and the speed of the bus which is a compile time define.

Fields

`port scl` Port on which clock wire is attached.

Must be on bit 0

`port sda` Port on which data wire is attached.

Must be on bit 0

`unsigned int clockTicks`
Number of reference clocks per I2C clock, set to 1000 for 100 Khz.

4.3 API

i2c_master_init()

Function that initialises the ports on an I2C device.

Type

```
void i2c_master_init(struct r_i2c &i2c_master)
```

Parameters

`i2c_master` struct containing the clock and data ports. Both should be declared as unbuffered bidirectional ports.

i2c_master_rx()

Function that reads data from an I2C device.

Type

```
int i2c_master_rx(int device,
                  unsigned char data[],
                  int nbytes,
                  struct r_i2c &i2c)
```

Parameters

`device` Bus address of device, even number between 0x00 and 0xFE.

`data` Array where data is stored.

`nbytes` Number of bytes to read and store in data.

`i2c` struct containing the clock and data ports. Both should be declared as unbuffered bidirectional ports.

i2c_master_read_reg()

Function that reads a register from an I2C device.

Note that this function uses the same interface as module_i2c but that the fields master_num and clock_mul are ignored by this function.

Type

```
int i2c_master_read_reg(int device,
                        int reg_addr,
                        unsigned char data[],
                        int nbytes,
                        struct r_i2c &i2c_master)
```

Parameters

`device`	Bus address of device, even number between 0x00 and 0xFE.
`reg_addr`	Address of register to read, value between 0x00 and 0x7F.
`data`	Array where data is stored.
`nbytes`	Number of bytes to read and store in data.
`i2c_master`	struct containing the clock and data ports. Both should be declared as unbuffered bidirectional ports.

i2c_master_write_reg()

Function that writes to a register on an I2C device.

Note that this function uses the same interface as module_i2c but that the fields master_num and clock_mul are ignored by this function.

Type

```
int i2c_master_write_reg(int device,
                         int reg_addr,
                         unsigned char data[],
                         int nbytes,
                         struct r_i2c &i2c_master)
```

Parameters

`device`	Bus address of device, even number between 0x00 and 0xFE.
`reg_addr`	Address of register to write to, value between 0x00 and 0x7F.
`data`	Array where data is stored.
`nbytes`	Number of bytes to read and store in data.
`i2c_master`	struct containing the clock and data ports. Both should be declared as unbuffered bidirectional ports.

4.4 Example Usage

Example usage of Module I2C Master is shown below:

```
void app_manager()
{
      unsigned button_press_1,button_press_2,time;
      int button =1;
      timer t;
      unsigned char data[1]={0x13};
      unsigned char data1[2];
      int adc_value;
      unsigned led_value=0x0E;
      p_PORT_BUT_1:> button_press_1;
      set_port_drive_low(p_PORT_BUT_1);
      i2c_master_write_reg(0x28, 0x00, data, 1, i2cOne); //Write configuration
        ↪ information to ADC
      t:>time;
      printstrln("** WELCOME TO SIMPLE GPIO DEMO **");
      while(1)
      {
              select
              {
                      case button => p_PORT_BUT_1 when pinsneq(button_press_1):>
                        ↪ button_press_1: //checks if any button is pressed
                            button=0;
                            t:>time;
                            break;

                      case !button => t when timerafter(time+debounce_time):>void: //
                        ↪ waits for 20ms and checks if the same button is pressed or not
                            p_PORT_BUT_1:> button_press_2;
                            if(button_press_1==button_press_2)
                            if(button_press_1 == BUTTON_PRESS_VALUE) //Button 1 is
                              ↪ pressed
                            {
                                    printstrln("Button 1 Pressed");
                                    p_led<:(led_value);
                                    led_value=led_value<<1;
                                    led_value|=0x01;
                                    led_value=led_value & 0x0F;
                                    if(led_value == 15)
                                    {
                                            led_value=0x0E;
                                    }
                            }
                            if(button_press_1 == BUTTON_PRESS_VALUE-1) //Button 2 is
                              ↪ pressed
                            {
                                    data1[0]=0;data1[1]=0;
                                    i2c_master_rx(0x28, data1, 2, i2cOne); //Read ADC
                                      ↪ value using I2C read
                                    printstrln("Reading Temperature value....");
                                    data1[0]=data1[0]&0x0F;
                                    adc_value=(data1[0]<<6)|(data1[1]>>2);
                                    printstr("Temperature is :");
                                    printintln(linear_interpolation(adc_value));
                            }

                            button=1;
                            break;
              }
      }
}
```