I2C Core

Core Overview

I2C Core 可以使 FPGA 内部逻辑与外部器件通过 I2C 接口串行通信, I2C Core 提供了一个 Avalon Memory-Mapped (Avalon-MM) 从端口来允许 Avalon-MM 主端口设备(如 Nios II 处理器) 通过简单的读写控制和数据寄存器来与 IP 核沟通.

Functional Description

I2C Core 包含两个用户可见的端口:

- Avalon-MM Slave 接口, 可以通过该端口访问寄存器
- I2C 接口, 包括 SCL 和 SDA 信号

Configuration

I2C Clock (SCL) Rate

I2C Core 同步于提供给 Avalon-MM 接口的时钟. I2C 接口的 SCL 输出时钟由该时钟分频而来. 模块的输入时钟需要用户给定, 并且与实际所接的时钟频率一致.

I2C 的输出时钟频率为 0~3400kHz. 输入的时钟频率至少为输出频率的 4 倍. 如果输入频率不是输出频率的整数倍, SCL 的实际频率可能跟设定的目标频率不完全相同.

Software Programming Model

Software Files

I2C Core 提供了以下软件文件, 这些文件提供了硬件的底层接口.

- **i2c_avalon_driver.h**—这个文件提供了访问底层硬件的函数定义.
- **i2c_avalon_driver.c**—这个文件包含访问底层硬件函数的实现.

Register Map

Table 1-1: Register Map for I2C Core

Internal Address	Register Name	Type [R/W]	31..8	7..2	1	0
0	control/state	R/W			ACK	START/$\overline{\text{STOP}}$
1	txdata	W		TXDATA		
2	rxdata _ack	R		RXDATA		
3	rxdata _nack	R		RXDATA		

读取未定义的位会返回不确定的值. 向未定义的位写没有效果.

control/state Register

当向该地址进行写操作时为控制寄存器，对该地址进行读操作时为状态寄存器.

控制寄存器: Bit0 为起始/停止位，其他位都没有使用，向该位写 1 会在 I2C 总线上发起开始操作，向该为写 0 会在 I2C 总线上发起停止操作.

控制寄存器: Bit1 为 ACK 标志位，其他位都没有使用, ACK 标志位会在每次进行完 I2C 写操作后更新，读取该位会返回上一次的 ACK 结果.

txdata Register

向该地址写入的数据会发送到 I2C 数据总线上，并在每次发送完成以后读取 ACK 状态并更新相应的寄存器.

rxdata_ack Register

读取该寄存器会在 I2C 总线上发起读操作，并在读操作完成以后自动发送一个 ACK 标志，并把读取到的数据返回.

rxdata_nack Register

与上一个寄存器类似，读取该寄存器会在 I2C 总线上发起一个读操作，但在读取完成之后会自动发送一个 NACK 标志，并把读取到的读数据返回.

Software Function Introduction

i2c_start()

Prototype:	`void i2c_start(unsigned int base)`
Include:	< i2c_avalon_driver.h>
Description:	调用该函数会在 I2C 总线上发起开始操作.
Returns:	No.

i2c_stop()

Prototype:	`void i2c_stop(unsigned int base)`
Include:	< i2c_avalon_driver.h>
Description:	调用该函数会在 I2C 总线上发起停止操作.
Returns:	No.

i2c_get_ack ()

Prototype:	`unsigned char i2c_get_ack(unsigned int base);`
Include:	< i2c_avalon_driver.h>
Description:	通过该函数读取 ACK 寄存器的值.
Returns:	0 代表 ACK, 1 代表 NACK.

i2c_write ()

Prototype:	`void i2c_write(unsigned int base,` `unsigned char data)`
Include:	< i2c_avalon_driver.h>
Description:	向 I2C 总线上写 1Byte 数据, 并读取 ACK 的值.
Returns:	No.

i2c_read_with_ack ()

Prototype:	`unsigned char i2c_read_with_ack(unsigned int base)`
Include:	< i2c_avalon_driver.h>
Description:	从 I2C 总线上读取 1Byte 数据, 并发送一个 ACK 信号.
Returns:	读取到的 1Byte 数据.

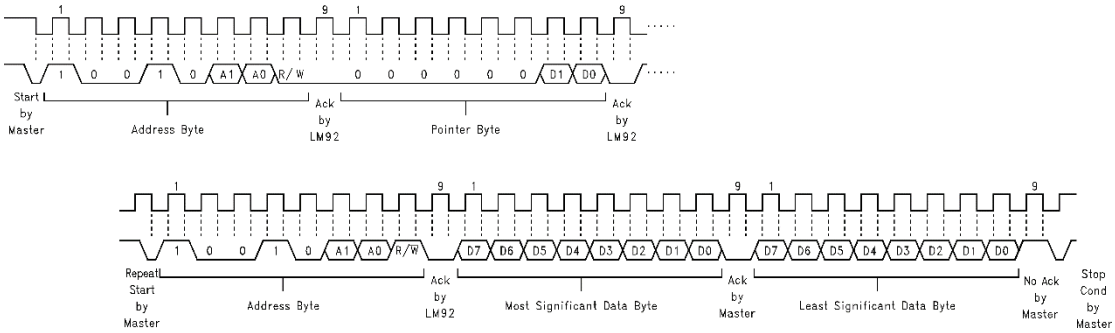
i2c_read_with_nack ()

Prototype:	`unsigned char i2c_read_with_nack(unsigned int base)`
Include:	< i2c_avalon_driver.h>
Description:	从 I2C 总线上读取 1Byte 数据, 并发送一个 NACK 信号.
Returns:	读取到的 1Byte 数据.

Example

Example 1

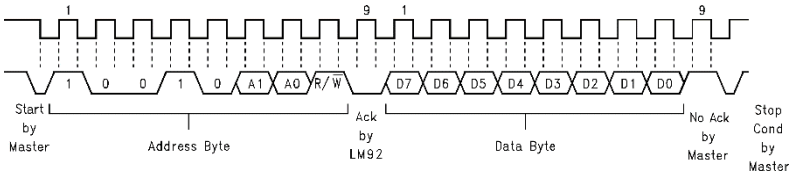
LM92: Typical Pointer Set Followed by Immediate Read for 2-Byte Register




```
#include "system.h"
#include <i2c_avalon_driver.h>

int main()
{
    unsigned char ack[3];
    unsigned char ans[2];

    i2c_start(I2C_AVALON_BASE);                     //Start by Master
    i2c_write(I2C_AVALON_BASE,0x90);                //Address Byte
    ack[0] = i2c_get_ack(I2C_AVALON_BASE);          //Get Ack by LM92
    i2c_write(I2C_AVALON_BASE,0x00);                //Pointer Byte
    ack[1] = i2c_get_ack(I2C_AVALON_BASE);          //Get Ack by LM92

    i2c_start(I2C_AVALON_BASE);                     //Repeat Start by Master
    i2c_write(I2C_AVALON_BASE,0x91);                //Address Byte
    ack[2] = i2c_get_ack(I2C_AVALON_BASE);          //Get Ack by LM92
    ans[1] = i2c_read_with_ack(I2C_AVALON_BASE);    //Data MSB with Ack
    ans[0] = i2c_read_with_nack(I2C_AVALON_BASE);   //Data LSB with No Ack
    i2c_stop(I2C_AVALON_BASE);                      //Stop Cond by Master

    return 0;
}
```

Example 2

LM92: Typical 1-Byte Read from Configuration Register with Preset Pointer




```
#include "system.h"
#include <i2c_avalon_driver.h>

int main()
{
    unsigned char ans;

    i2c_start(I2C_AVALON_BASE);                    //Start by Master
    i2c_write(I2C_AVALON_BASE,0x91);               //Address Byte
    ans = i2c_read_with_nack(I2C_AVALON_BASE);     //Data Byte
    i2c_stop(I2C_AVALON_BASE);                     //Stop Cond by Master

    return 0;
}
```

Document Revision History

Data	Version	Changes
August 2015	1.1	重写了核心代码, 提高了兼容性.
August 2015	1.0	第一次发布