XAPP799 (v1.1) August 2, 2005

An SMBus/I2C-Compatible Port Expander

Summary

Today's microcontrollers and microprocessors often limit the number of General Purpose I/O (GPIO) ports in order to conserve pin count and to reduce package sizes. Unfortunately, for many designs, the number of I/O ports required exceeds the number of available I/O ports on a microprocessor. Hence, Complex Programmable Logic Devices (CPLDs) can often be found in a system alongside a microcontroller functioning as a port expander. This Application Note presents a design of a port expander that fits into a CoolRunner-II XC2C32A device -- A port expander that is SMBus and I2C compatible.

CoolRunner-II Advantages

A key advantage of using any Xilinx CPLD as a port expander is the integration of logic functions across the entire board. The flexibility of programmable fabric allows for a variety of dissimilar functions to be implemented on one chip. A CPLD can be used as a port expander, an LED driver, an address decoder and a memory controller, all at once.

CoolRunner-II devices add further value being the lowest cost and lowest power CPLDs on the market. These CoolRunner-II parts also provide I/O Banking capability, allowing for level translation between the microcontroller and its external peripherals. Both features are favorable for an SMBus or I2C design, as these environments are typically multi-voltage environments requiring low power.

Table 1 summarizes the level translation abilities of the CoolRunner-II family.

Table 1: **I/O Standards for the CoolRunner-II XC2C32A**

IOSTANDARD Attribute	Output V_{CCIO}	Input V_{CCIO}	Input V_{REF}	Board Termination Voltage V_T
LVTTL	3.3	3.3	N/A	N/A
LVCMOS33	3.3	3.3	N/A	N/A
LVCMOS25	2.5	2.5	N/A	N/A
LVCMOS18	1.8	1.8	N/A	N/A
LVCMOS15(1)	1.5	1.5	N/A	N/A

(1) LVCMOS15 requires the use of Schmitt Trigger

Using the CoolRunner-II SMBus/I2C Port Expander Design

The port expansion reference design shown in Figure 1 provides 8 output ports and 8 input ports controlled through an I2C compatible serial interface. This design is scalable. The code can be modified to increase or decrease the number of desired output ports and input ports. This section describes the default code implementing 8 outputs and 8 inputs. The fully working

project, with source code, is available for download at
http://www.xilinx.com/bvdocs/publications/XAPP799_designfiles.zip.

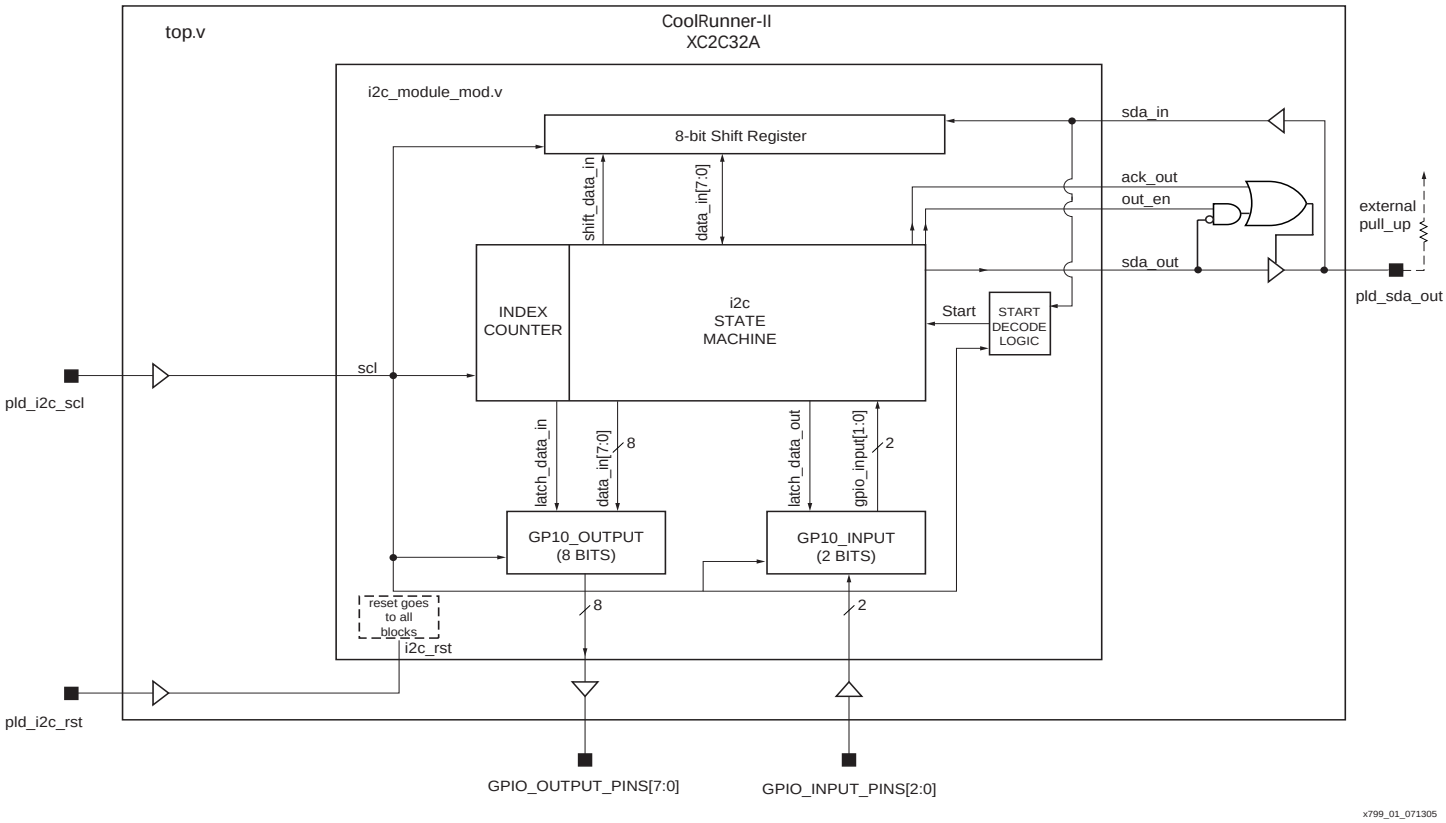

Figure 1: **I2C Port Expander Block Diagram**

Table 2: **Pin Descriptions**

Name	Function
pld_i2c_scl	Serial Clock Line
pld_i2c_sda	Serial Data Line
pld_i2c_rst	Active High Reset
GPIO_Output_Pins	Port Expansion Outputs
GPIO_Input_Pins	Port Expansion Inputs

Serial Interface

This port expander design uses a serial data line (SDA) and a serial clock line (SCL) to achieve bidirectional communication with a master. The master, typically a microcontroller or microprocessor, initiates all data transfers to and from the CPLD. The master is also responsible for generating the SCL clock that synchronizes the data transfer.

Each transaction consists of a start condition sent by a master, followed by the CPLD's predefined 7-bit slave address plus an R/$\overline{W}$ bit, and one data byte that is either transmitted from the master to the CPLD (for a write operation) or one data byte that is transmitted by the CPLD to the master (for a read operation). An Acknowledge bit is used at the end of each data byte, and data is transferred on every rising clock pulse.

Start Condition

Both SCL and SDA remain high when the interface is inactive. The master signals the start condition by transitioning SDA from high to low while SCL is high (Figure 2). Once a start condition is recognized by the CPLD, an internal ‘start’ pulse is generated, and the state machine will begin.

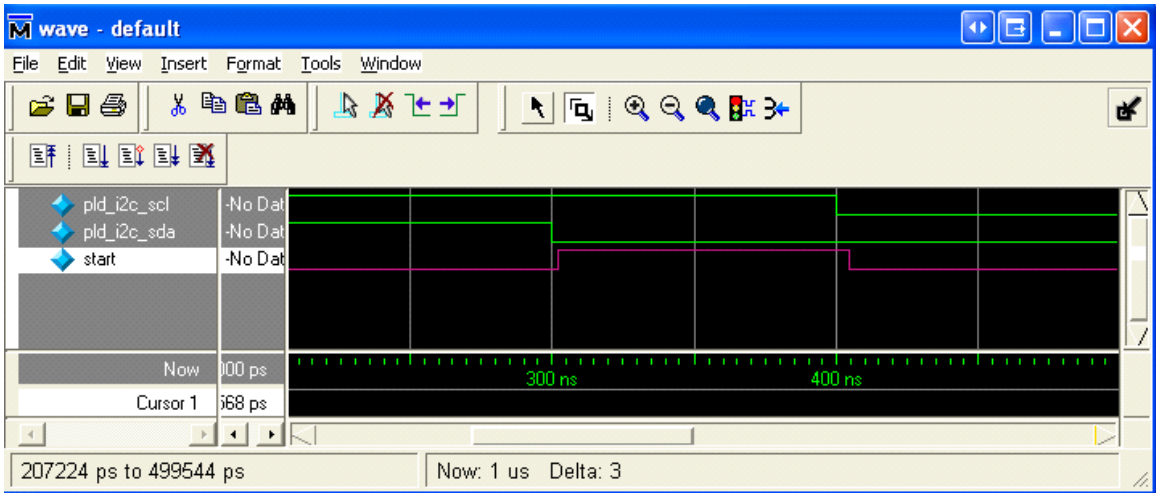

Figure 2: **Start Condition Initiated**

Acknowledge

The acknowledge bit is sent every 9th bit, indicating receipt of each data byte. Each byte transferred effectively requires 9 bits. The master generates the 9th clock pulse, and the recipient pulls down SDA during the acknowledge pulse. This means that when the master is transmitting to the CPLD, the CPLD generates the acknowledge bit. When the CPLD is transmitting to the master, the master generates the acknowledge bit.

I2C Slave Address

The CPLD has a 7-bit long I2C slave address that can be preprogrammed into the device through the source code. Edit the top level Verilog source file (top.v) and define your desired 7-bit binary value for the “my_i2c_addr” parameter. The 8th bit following the 7-bit slave address is the R/W bit. Set this bit low for a write command and high for a read command.

Performing a Write Command

After a slave address with a write command has been sent, one final data byte must be sent from the master to complete the transaction. This data byte defines the state of each of the 8 I/O’s. The MSB of the data byte defines the value on ‘GPIO_OUTPUT_PINS[7]’ and the LSB of the data byte defines the value on ‘GPIO_OUTPUT_PINS[0]’. The outputs assume their defined values during the 9th clock cycle, together with the acknowledge pulse.

Figure 3 shows a full Write Command transaction. The master initiates a start command, then sends a write request to the CPLD, located at slave address 0x56. The CPLD acknowledges the write request on the 9th clock edge, and the master continues delivering data to the CPLD

dictating the values on the GPIO output ports. On the 9th clock cycle, the CPLD acknowledges this second data byte, and sets the GPIO output pins accordingly.

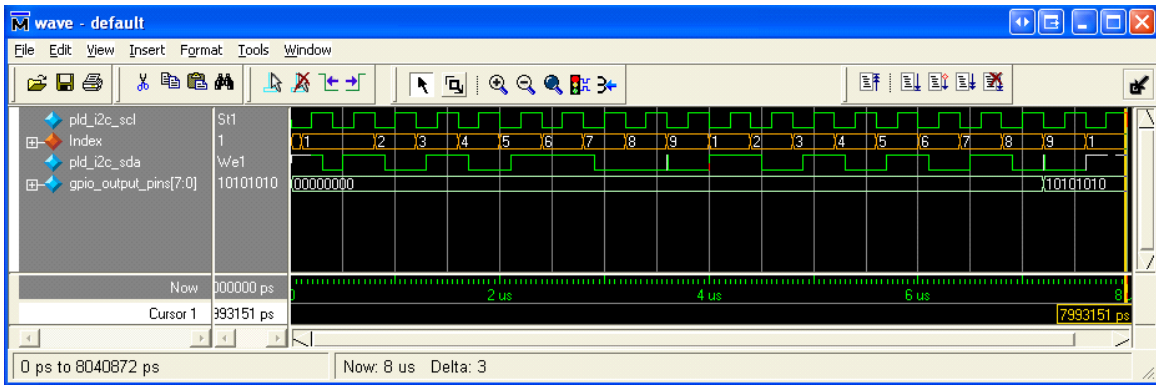

Figure 3: **Write Command Transaction**

Performing a Read Command

If a slave address with a read command has been sent, one final data byte is sent from the CPLD to the master. The data byte relays the values present on the 'GPIO_INPUT_PINS' bus.

Figure 4 shows a full Read Command transaction. The master initiates a start command, then sends a read request to the CPLD located at slave address 0x56. The CPLD acknowledges the read request on the 9th clock edge. On the next data byte, the CPLD returns the value on the GPIO input pins. In this example, since there are only 2 GPIO input pins in the design, only the data on the 7th and 8th clock are valid. The first 6 data bits are high by default, and are ignored by the master. On the 9th clock cycle, the master acknowledges this second data byte, and the read transaction completes.

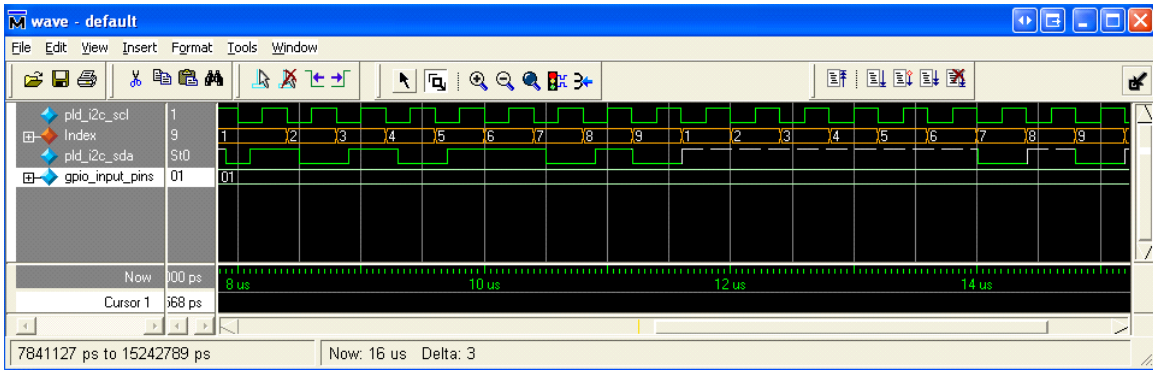

Figure 4: **Read Command Transaction**

Standby

The CoolRunner-II family of CPLDs automatically enters "standby" when all pins are set to Vcc or GND. Standby current is specified in the individual device data sheets. This design fits into an XC2C32A device, which has a typical standby number of 22 uA -- ideal for SMBus and I2C applications.

Utilization

The following are the utilization statistics:

Table 3: **Device Utilization**

Macrocells Used/Total	Product Terms Used/Total	Function Block Inputs Used/Total	Registers Used/Total	Pins Used/Total
32/32 (100%)	71/112 (63%)	36/80 (45%)	32/32 (100%)	13/33 (39%)

Customizing the Design

This reference design can be customized to fit any specific design target. If the design requires more GPIOs, the state machine in the source code can be modified accordingly. Should bidirectional I/O's be required, the design can be modified to use the R/$\overline{W}$ bit as a signal to decode whether the I/O pin should function as an input or output. There are many possibilities, and the source code has been written for the most general case.

Conclusion

This port expander design can be used as a complete turnkey design that fits readily into a CoolRunner-II XC2C32A device. You can easily edit the port expansion module to fit your needs, and additional functionality can be integrated into the CPLD. The CoolRunner-II family of CPLDs is unique in that it is the only low cost, easy to use, low power device available today.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/19/05	1.0	Initial Xilinx release.
08/02/05	1.1	Fixed broken link to design files.