OpenCores.Org

Cyc2-openrisc Application Note

Author: Steve Fielding
sfielding@base2designs.com

Rev. 1.0
November 18, 2008

Revision History

Rev.	Date	Author	Description
1.0	11/05/08	Sfielding	Created

Contents

1 Introduction

Cyc2-openrisc is an implementation of the the Opencores OR1200 RISC processor

http://www.opencores.org/projects.cgi/web/or1k/openrisc_1200

including peripherals, and is targeted at a Base2Designs hardware development platform.

The aim is to make the entire package easy to use, and useful to developers. Thus the project includes instructions for tool installation on a Windows PC, simulation files, and Altera Quartus project files.

2 Architecture

Cyc2-openrisc is targeted at the Base2Designs development hardware. The hardware consists of a main board containing an Altera EP2C20 FPGA, a Micron 4Mx32 SDRAM, and a 48Mhz oscillator (See Figure 2.1). The main board has expansion connectors for a Santa Cruz daughter card, and a FPGA support board.

The FPGA support board consists of a SD card slot, SPI flash, C8051 microcontroller, C8051 debug port, RS-232 serial port, FPGA JTAG connector, and a RISC JTAG connector. The FPGA support board implements the the Base2Designs FPGA configuration scheme

http://www.opencores.org/projects.cgi/web/fpgaconfig/overview

which supports FPGA configuration from FPGA image files stored in SPI flash or SD memory cards.

The Santa Cruz expansion header is an Altera developed expansion header, with expansion cards available from many vendors including Base2Designs. Cyc2-openrisc has support for the following Opencores projects;

SpiMaster http://www.opencores.org/projects.cgi/web/spimaster/overview

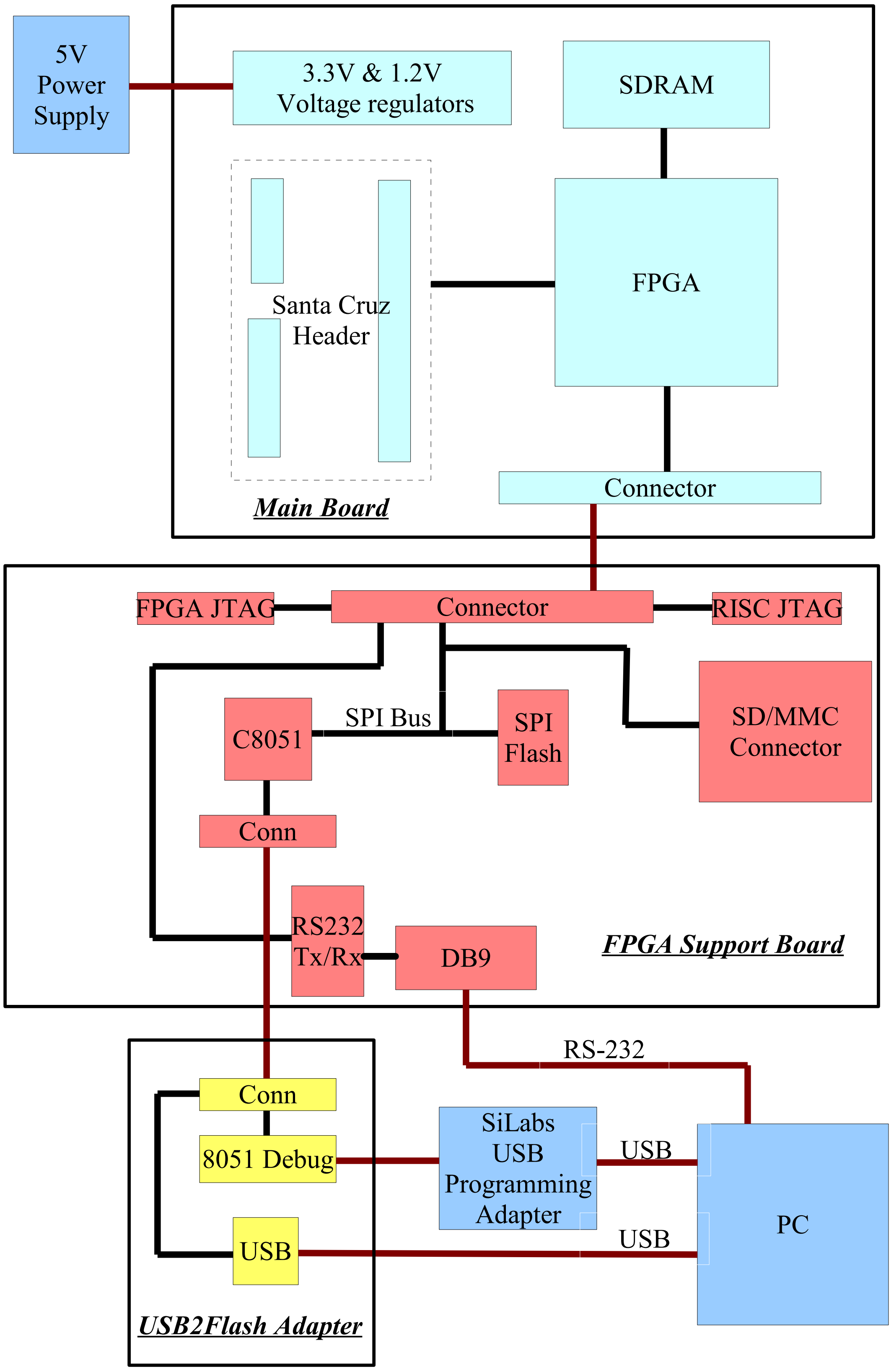
UsbHostSlave http://www.opencores.org/projects.cgi/web/usbhostslave/overview

Santa Cruz daughter cards are available for all these projects;

http://www.base2designs-store.com/

WISHBONE COMPATIBLE

Figure 2.1 – cyc2-openrisc hardwdare block diagram



3 Operation

The Main Board contains a minimal number of components required to support the EP2C20 Cyclone2 FPGA in an OpenRISC application. OpenRISC, is an open source RISC processor, available from OpenCores;

http://www.opencores.org/projects.cgi/web/or1k/overview

A 30-pin ribbon cable connects the Main Board to the FPGA support board. The FPGA Support Board contains all the components required to configure the FPGA using the Base2Designs FPGA configuration scheme. Base2Designs FPGA configuration scheme allows the storage of FPGA configuration files in commodity flash memory, either SPI flash, or SD/MMC flash cards. In addition to storing the FPGA configuration files, the flash memory can be used to store OpenRISC program files. The Support Board also contains a DB9 supporting RS-232 serial communications with the OpenRISC processor, a FPGA JTAG connector useful for running Altera Signal Tap logic analyser, an OpenRISC JTAG connector for OpenRISC software debug, and a USB2Flash connector which supports the Base2Designs USB2Flash Programming adapter.

Let's take a look at the sequence of operation from power on.

When power is first applied to the main board the FPGA is unconfigured. This means that the state of the internal look up tables (LUTs) the internal FPGA interconnect, and the internal block RAM are undefined. At power up the FPGA is in slave serial configuration mode (set by mode pins MSEL1, and MSEL0), and waits for a master configuration device to begin configuration. In this case the master configuration device is the C8051 located on the FPGA Support board.

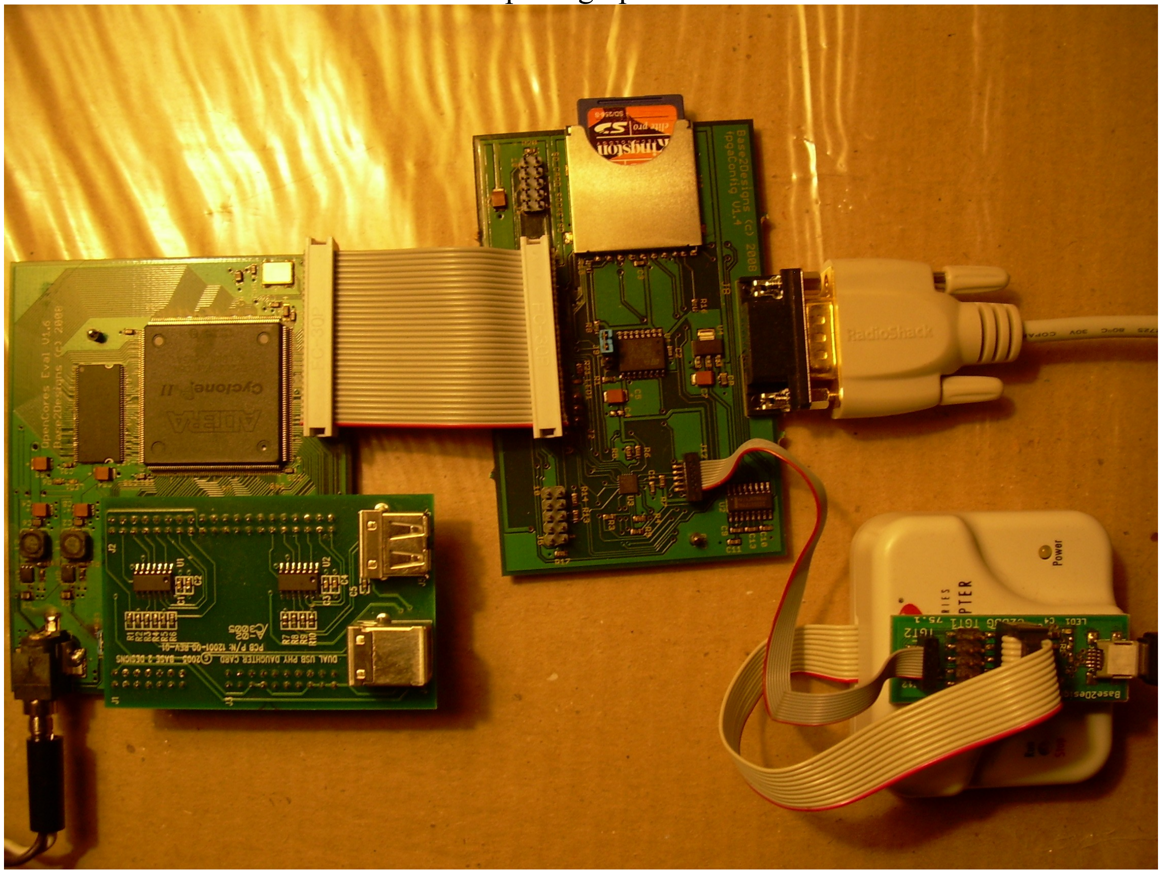
At power up the C8051 executes code from its own internal flash memory. The program reads an FPGA configuration file from either SPI flash or SD/MMC flash memory and copies the file to the FPGA through the FPGA serial configuration port. The configuration file loaded into the FPGA can initialize FPGA block RAM. Thus it is possible to initialize FPGA RAM with an OpenRISC program, and the OpenRISC can execute this program at power up. If the program is small enough to fit inside FPGA RAM, then there is no need for any additional memory. Currently 2Kx32 words of FPGA RAM is used as OpenRISC program memory. For much larger programs, it is necessary to load programs into SDRAM memory. This can be achieved by initializing FPGA RAM with a boot loader program. The boot loader initializes SDRAM memory, and reads a second program from flash memory, copies this program into SDRAM, and then jumps to SDRAM to execute the new program.

New FPGA programming files and OpenRISC programs can be written to flash memory using the Base2Designs USB2Flash programming adapter, and the associated PC console application fpgaConfig

4

Getting Started

Connect all the boards as shown in the photograph



Note that the Main board is shown with the optional dual USB PHY Santa Cruz daughter card installed.

Powering on for the first time

Connect the FPGA Support Board RS-232 port to a PC. You may have to use a RS-232 to USB adapter if your PC does not have a RS-232 serial port. On the PC use a terminal program such as Windows Hyperterminal, and configure the attached serial port for 11520 baud, 8 data bits, 1 stop bit, no parity, no flow control.

Assuming you are using the pre-programmed SD card that ships with the evaluation kit, you can now apply power to the main board, and monitor the output from the serial port.

When the board is powered on, you should see LED1 (+3.3V power) on the Main Board continuously lit, and LED1 (watchdog strobe) on the FPGA support board flashing. As the OpenRISC first executes the boot loader, and then copies and executes the test program from flash to SDRAM, you should see something similar to the following:

```
SD Boot loader V1.0
Starting SD Init
Waiting transaction complete ...
Detected boot file, size 00002A00
File load OK

DRAM test
Press any key to stop
Testing DRAM at address 0x40040000
Testing DRAM at address 0x40080000
...
Testing DRAM at address 0x40FC0000
Passed DRAM memory test
Press a key and see the key+1 echoed
```

On completion the program waits for terminal key presses, and echoes the ASCII value of the key press plus one.

Loading new FPGA configuration files

Let's copy a new FPGA configuration file from the PC to the support Board flash memory.

First connect the USB2Flash adapter to the PC using a type A to Mini B USB cable. On the PC install USB2Flash console application fpgaConfig, either from the Base2Designs install disk or from:

http://www.base2designs.com/downloads/fpgaConfig_v1_1.zip

Using Windows file explorer browse to

cyc2-openrisc\progFiles\2008_11_02\blockRAMresident_memTest

Double click download.bat

This copies the binary FPGA configuration file cyc_or12_mini_top_memTest.rbf to flash memory at address 0, and forces the FPGA to be reconfigured, thus copying the new file from flash to the FPGA. You should see the following at the terminal screen:

```
DRAM test

Press any key to stop

Testing DRAM at address 0x40040000

Testing DRAM at address 0x40080000

...

Testing DRAM at address 0x40FC0000

Passed DRAM memory test

Press a key and see the key+1 echoed
```

This is the same memory test as before but you will notice that the boot loader does not run, and the test completes quicker than before. This is because the test program was run from FPGA RAM rather than DRAM.

Loading new software image files

First we need to copy the boot loader to flash memory:

Using Windows file explorer browse to:

cyc2-openrisc\progFiles\2008_11_02\bootLoader

Execute downloadFPGAimage.bat

Using the two batch files downloadSoftware_SDTest.bat, and downloadSoftware_DRAMmemTest.bat, you can copy software programs into flash memory, and monitor the status of the programs from the terminal window. Note that these batch files copy the software image files to flash memory at address 0x90000, and then force an FPGA re-configuration. After re-configuration the OpenRISC will run the bootloader program which will then copy the new software image file from flash to DRAM, jump to DRAM, and execute the new program.

5 Building Software

First you will need to install Cygwin. You will need several of the development tools that are available with Cygwin, and also the OpenRisc tool chain.

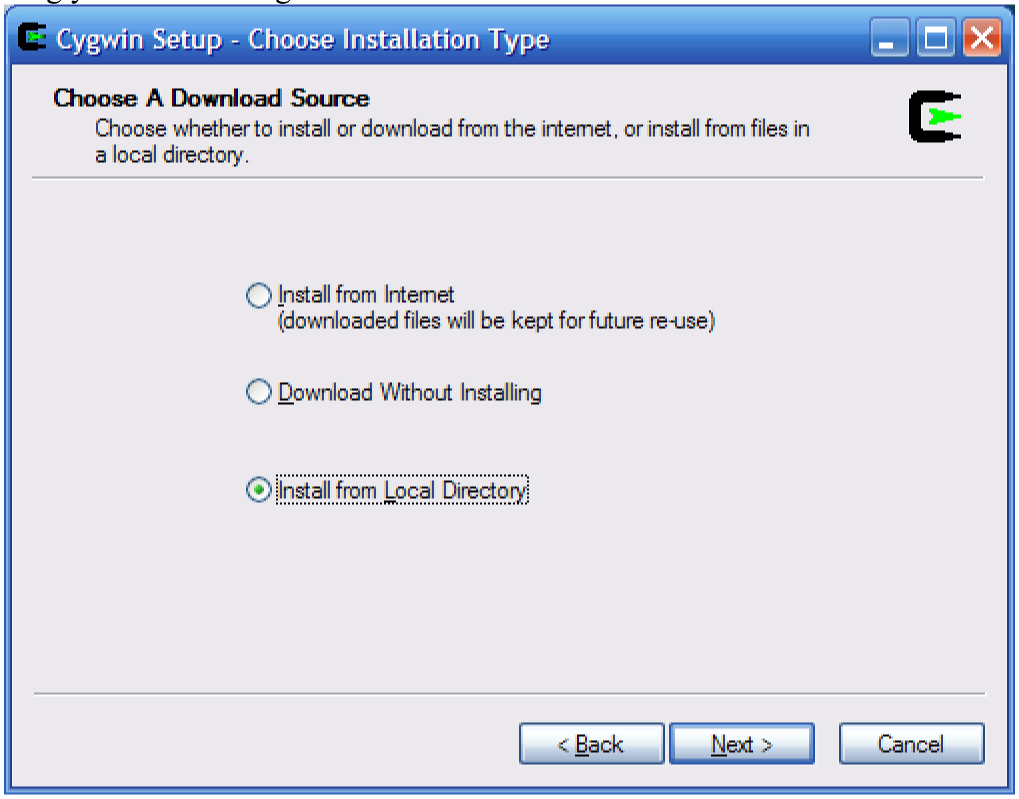
There are two options for installing Cygwin. One is to install from the internet, and the other is to install from a local directory. If you install from the internet, get Setup from:

http://www.cygwin.com/setup.exe

or from the install disk:

Cygwin\setup.exe

Assuming you are installing from disk:



Cygwin Setup - Choose Installation Type

Choose A Download Source

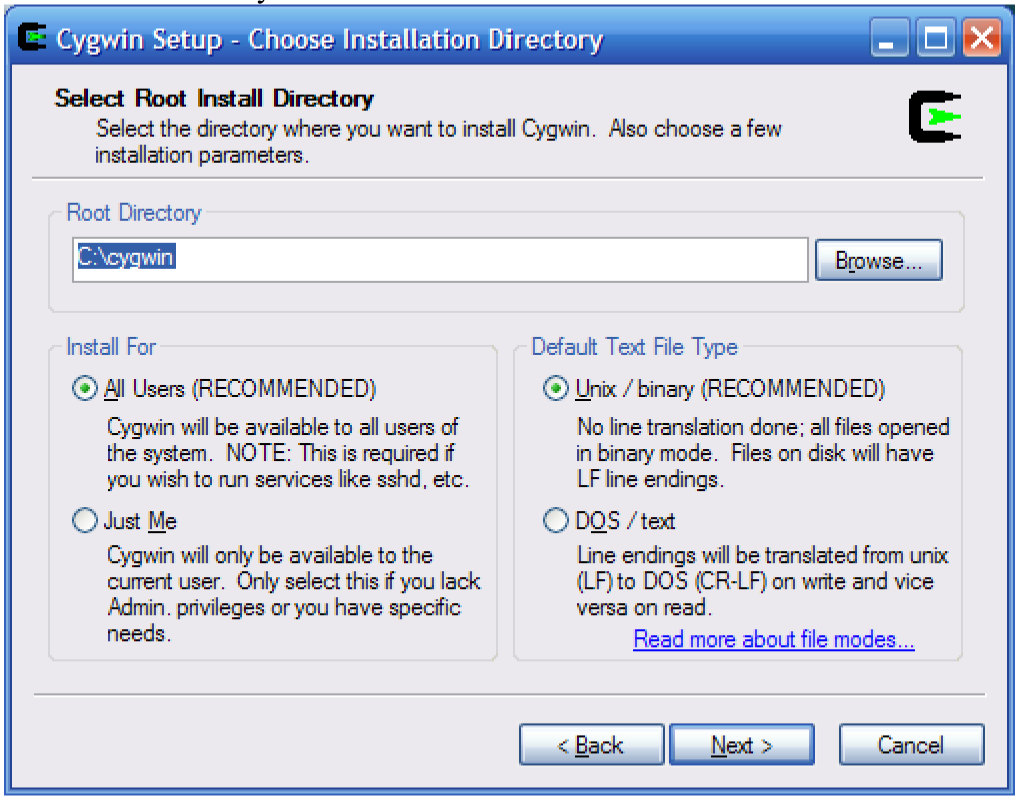
Choose whether to install or download from the internet, or install from files in a local directory.

○ Install from Internet (downloaded files will be kept for future re-use)

○ Download Without Installing

● Install from Local Directory

Browse to install directory.



Cygwin Setup - Choose Installation Directory

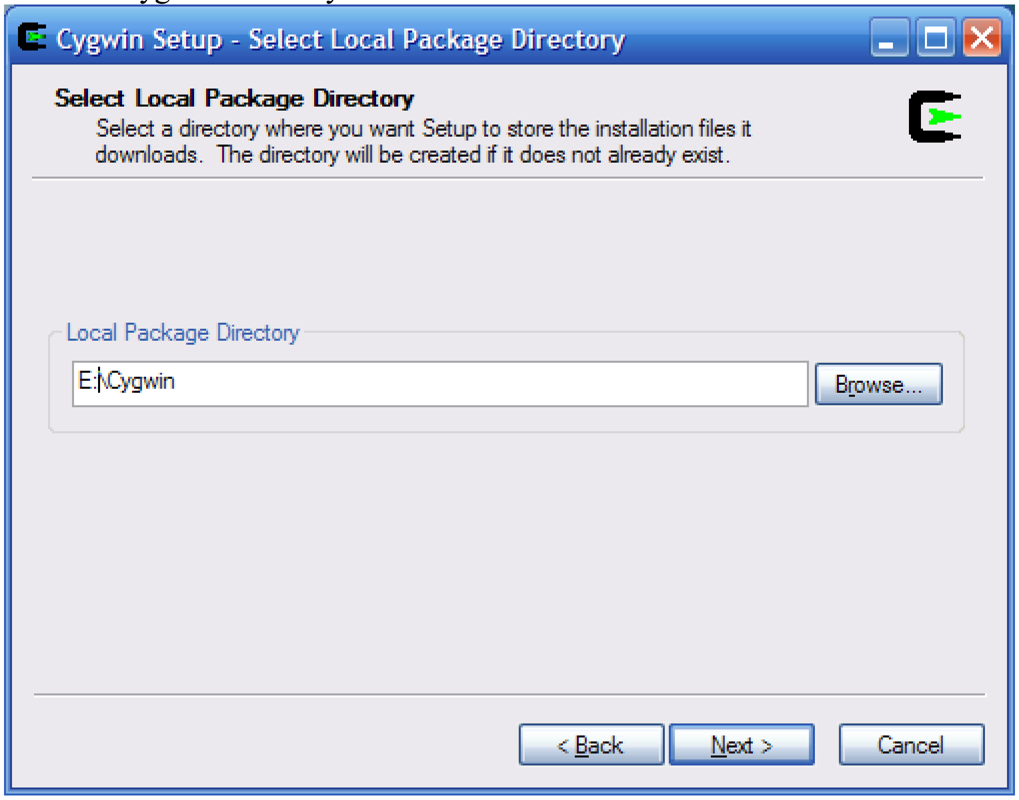
Select Root Install Directory
Select the directory where you want to install Cygwin. Also choose a few installation parameters.

Root Directory
C:\cygwin

Install For
[✓ All Users (RECOMMENDED)]
Cygwin will be available to all users of the system. NOTE: This is required if you wish to run services like sshd, etc.
Just Me
Cygwin will only be available to the current user. Only select this if you lack Admin. privileges or you have specific needs.

Default Text File Type
[✓ Unix / binary (RECOMMENDED)]
No line translation done; all files opened in binary mode. Files on disk will have LF line endings.
DOS / text
Line endings will be translated from unix (LF) to DOS (CR-LF) on write and vice versa on read.
Read more about file modes...

Browse to the Cygwin directory on the CDROM

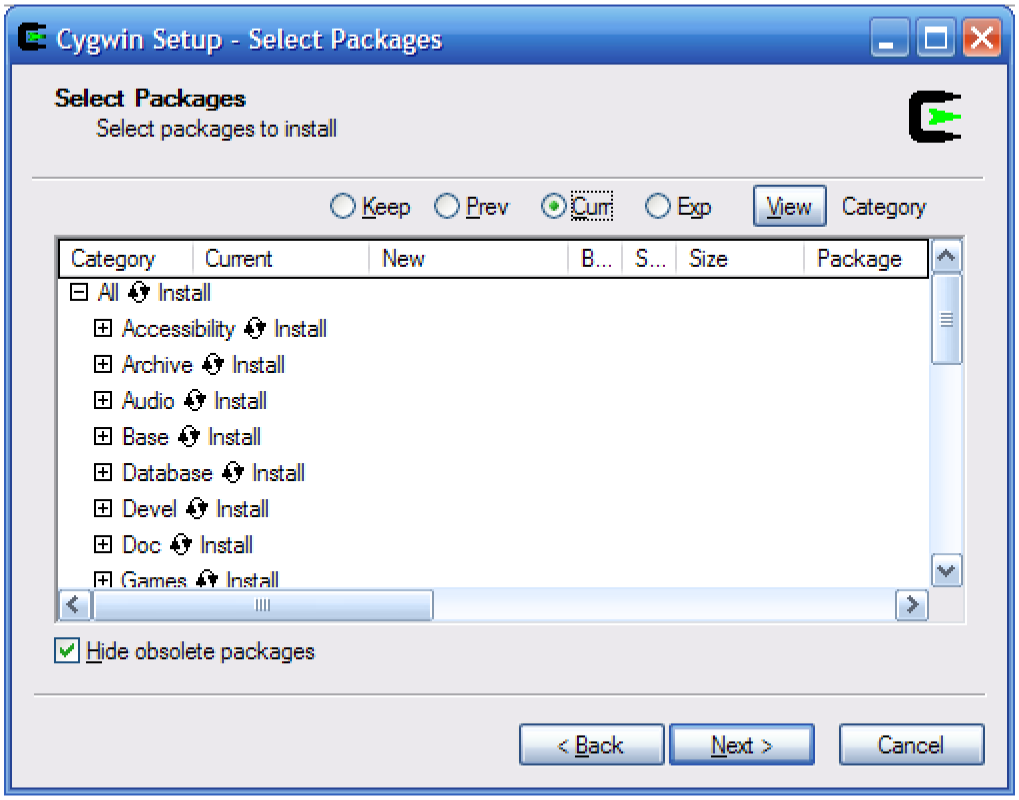


Cygwin Setup - Select Local Package Directory

Select Local Package Directory
Select a directory where you want Setup to store the installation files it downloads. The directory will be created if it does not already exist.

Local Package Directory
E:\Cygwin

Click on 'Default' next to 'All' packages, until 'install' appears. If you are installing from the internet you will want to be more selective. Just change 'Devel Default' to 'Devel install'.



All Install
Accessibility Install
Archive Install
Audio Install
Base Install
Database Install
Devel Install
Doc Install
Games Install

Next you need to install the Openrisc tool chain, either from the install disk or from:

http://www.opencores.org/projects.cgi/web/or1k/or1k-bin/or32-uclinux-2003-04-13.cygwin.tar.gz

Copy the files from the install disk:

/cygwin/opt

to your Cygwin install directory

Cygwin install directory, is c:/cygwin by default.

Add the install path for the new tools to your .bash_profile. C:/cygwin/home/myName by default. Try to use a plain text editor such as vim, as Wordpad can introduce unwanted formating that will confuse Cygwin.

# Set PATH to or32 tools

PATH=$PATH:/opt/or32-uclinux/bin

Start cygwin, and change directory to the install path for cyc2-openrisc/sw. You will need to cd to cygdrive/c to get at your c: drive root. Eg

cd /cygdrive/c/myProjects/cyc2-openrisc/sw/memTest

Now rebuild the project

make clean all

and download to flash memory

./download.bat

Now try re-building the SD test project, and downloading to flash.

cd ../sdTest2

make clean all

./download.bat

6
Building Hardware

First you will need to install Quartus. You can download Quartus Web Edition for free:

http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html

Now open Quartus, and load the project file:

File >> Open Project

browse to syn/cyc_or12_mini_top.qpf

Build the project:

Processing >> Start Compilation

Once compilation is complete, create the binary programming file

File >> Convert Programming Files...

In the new window, select

Open Conversion Setup Data...

and browse to syn/cyc_or12_mini_top.cof

Now select:

Generate

This will generate cyc_or12_mini_top.rbf

Execute syn/download.bat , to download the newly created programming file to flash memory.

Changing FPGA RAM initialization

By default the project is configured to initialize FPGA RAM with the bootloader program.

You can change this by editing the file

cyc_or12_mini_top.qsf

Eg to select memTest, comment out onchip_ram_loadRAM, and remove the comment from onchip_ram_memTest:

```
# set_global_assignment -name VERILOG_FILE ../rtl/mem_if/onchip_ram_loadRAM.v
set_global_assignment -name VERILOG_FILE ../rtl/mem_if/onchip_ram_memTest.v
```

Now test the changes by rebuilding the project, and downloading the new FPGA configuration file.

Configure hardware for USB or SD Santa Cruz daughter card

Next you need to edit cyc_or12_mini_top.qsf to select the appropriate top
level file based on the Santa Cruz daughter card that you are using. If you
are not using a Santa Cruz daughter card then either configuration will work fine.
There are two choices. You can edit the following lines to select
between them;

```
set_global_assignment -name VERILOG_FILE ../rtl/top/cyc_or12_mini_top.v
# set_global_assignment -name VERILOG_FILE ../rtl/top/cyc_or12_mini_top_sdCard.v
```

cyc_or12_mini_top.v supports the USB daughter cards.
cyc_or12_mini_top_sdCard.v supports SD daughter card.

If you are using a USB daughter card, you will also need to edit
top/cyc_or12_defines.v to select NXP ISP1105 daughter card (remove comments),
or Fairchild USB1T11A daughter card (leave comments).

```
//
// Define NXP ISP1105 USB PHY
//
//`define PHY_ISP1105
```

Configure hardware for custom Santa Cruz daughter card

----

1. Edit the Santa Cruz I/O.

Edit the direction for the Santa Cruz I/O.

You will need to edit the following section to ensure that the Santa Cruz I/O is set to the correct direction;

```
 output SC_P_CLK;
 output SC_RST_N;
 output SC_CS_N;
 input SC_P0;
 output SC_P1;
 output SC_P2;
 ...
 output SC_P38;
 output SC_P39;
```

----

2. Modify the Wishbone bus interface

If the new peripheral(s) have a Wishbone bus interface then create local interface wires for each new peripheral, eg;

```
//
// Santa Cruz SD card i/f wires
//
wire   [31:0]      wb_sdCard_dat_i;
wire   [7:0]       wb_sdCard_dat_8bit;
wire   [31:0]      wb_sdCard_dat_o;
wire   [31:0]      wb_sdCard_adr_i;
wire   [3:0]       wb_sdCard_sel_i;
wire               wb_sdCard_we_i;
wire               wb_sdCard_cyc_i;
wire               wb_sdCard_stb_i;
wire               wb_sdCard_ack_o;
```

----

3. Add peripheral wire connections

Add local interface wires for each wired connection to the Santa Cruz header, eg;

```
 //
 // Santa Cruz SD card
 //
 wire sdSpiClk;
 wire sdSpiMasterDataIn;
 wire sdSpiMasterDataOut;
 wire sdSpiCS_n;
```

----

4. Add instance(s) of the new peripheral(s), eg;

```
spiMaster u_sdSpiMaster (
 //Wishbone bus
 .clk_i(wb_clk),
 .rst_i(wb_rst),
 .address_i(wb_sdCard_adr_i[7:0]),
 .data_i(wb_sdCard_dat_i[7:0]),
 .data_o(wb_sdCard_dat_8bit),
 .strobe_i(wb_sdCard_stb_i),
 .we_i(wb_sdCard_we_i),
 .ack_o(wb_sdCard_ack_o),
 // SPI logic clock
 .spiSysClk(clk),
 //SPI bus
 .spiClkOut(sdSpiClk),
 .spiDataIn(sdSpiMasterDataIn),
 .spiDataOut(sdSpiMasterDataOut),
 .spiCS_n(sdSpiCS_n)
);
```

----

5. Configure the address

If the new peripheral(s) have a Wishbone bus interface then
modify the parameter block for tc_top to set the address(es) of your new peripheral(s).
Typically you will be modifying the offset address of Target 2 through 8

```
tc_top #(`APP_ADDR_DEC_W,
         `APP_ADDR_SRAM,   //Target 0 address
         `APP_ADDR_DEC_DRAM_W,
         `APP_ADDR_DRAM,   //Target 1 address
         `APP_ADDR_DECP_W,
         `APP_ADDR_PERIP,  //Target 2-8 address base
         `APP_ADDR_DEC_W,
         `APP_ADDR_VGA,     //Target 2 address offset
         `APP_ADDR_ETH,     //Target 3 address offset
         `APP_ADDR_SD_CARD, //Target 4 address offset
         `APP_ADDR_UART,    //Target 5 address offset
         `APP_ADDR_USB2,    //Target 6 address offset
         `APP_ADDR_SD,      //Target 7 address offset
         `APP_ADDR_RES2     //Target 8 address offset
```

----

6. Set address constants

Modify the address map in cyc_or12_defines.v to add your new address offsets;

```
//
// Address map
```

```
//
`define APP_ADDR_DEC_W          8
`define APP_ADDR_SRAM  `APP_ADDR_DEC_W'h00
`define APP_ADDR_DEC_DRAM_W  2
`define APP_ADDR_DRAM `APP_ADDR_DEC_DRAM_W'b01
`define APP_ADDR_DECP_W  4
`define APP_ADDR_PERIP  `APP_ADDR_DEC_W'h9
`define APP_ADDR_VGA    `APP_ADDR_DEC_W'h97
`define APP_ADDR_ETH    `APP_ADDR_DEC_W'h92
`define APP_ADDR_USB1   `APP_ADDR_DEC_W'h9d
`define APP_ADDR_SD_CARD      `APP_ADDR_DEC_W'h9d
`define APP_ADDR_UART  `APP_ADDR_DEC_W'h90
`define APP_ADDR_USB2   `APP_ADDR_DEC_W'h94
`define APP_ADDR_SD       `APP_ADDR_DEC_W'h9e
`define APP_ADDR_RES2   `APP_ADDR_DEC_W'h9f
```

----

7. Connect I/O to the Santa Cruz card

Set the assignments required to connect the Santa Cruz I/O to the new peripheral(s), eg;

```
 assign SC_P_CLK = 1'b0;
 assign SC_RST_N = 1'b0;
 assign SC_CS_N = 1'b0;
 assign sdSpiMasterDataIn = SC_P0;
 assign SC_P1 = sdSpiClk;
 assign SC_P2 = sdSpiMasterDataOut;
 assign SC_P3 = sdSpiCS_n;
 assign SC_P4 = 1'b0;
 assign SC_P5 = 1'b0;
 assign SC_P6 = 1'b0;
 assign SC_P7 = 1'b0;
```

```
assign SC_P8 = 1'b0;
assign SC_P9 = 1'b0;
assign SC_P10 = 1'b0;
assign SC_P11 = 1'b0;
assign SC_P12 = 1'b0;
assign SC_P13 = 1'b0;
assign SC_P14 = 1'b0;
assign SC_P15 = 1'b0;
assign SC_P16 = 1'b0;
assign SC_P17 = 1'b0;
assign SC_P18 = 1'b0;
assign SC_P19 = 1'b0;
assign SC_P20 = 1'b0;
assign SC_P21 = 1'b0;
assign SC_P22 = 1'b0;
assign SC_P23 = 1'b0;
assign SC_P24 = 1'b0;
assign SC_P25 = 1'b0;
assign SC_P26 = 1'b0;
assign SC_P27 = 1'b0;
assign SC_P28 = 1'b0;
assign SC_P29 = 1'b0;
assign SC_P30 = 1'b0;
assign SC_P31 = 1'b0;
assign SC_P32 = 1'b0;
assign SC_P33 = 1'b0;
assign SC_P34 = 1'b0;
assign SC_P35 = 1'b0;
assign SC_P36 = 1'b0;
assign SC_P37 = 1'b0;
assign SC_P38 = 1'b0;
assign SC_P39 = 1'b0;
```

----

8. Create a software project.

Finally create a software project to test your new peripheral.

You can use sw/memTest as a basis for your new software project.
The memTest software project creates output files that are suitable
for use in simulation and hardware. Of course, if you want to target
a simulation you will need to create a simulation model of whatever
hardware exists on your Santa Cruz daughter card.
Add a constant to board.h to define the address of your new periperal, eg;

```
#define SD_CARD_BASE  0x9d000000
```

You can access your new periperal using the REG8 etc macros, eg;

```
REG8(SD_CARD_BASE + SPI_TX_FIFO_DATA_REG) = dataWrite;
dataRead = REG8(SD_CARD_BASE + SPI_RX_FIFO_DATA_REG);
```

7 Simulation

Install simulator and waveform viewer

First install Icarus Verilog and GTKwave. You can get the Icarus install file from either Base2Designs install disk, or from:

ftp://icarus.com/pub/eda/verilog/v0.8/Windows

or

http://www.base2designs.com/downloads/iverilog-0.8-setup.exe

You can get GTKWave from the Base2Designs intsall disk, or from:

http://www.base2designs.com/downloads/GTKwave.msi

Build the software

First build the software application. sw/memTest can be used

as an example.

From Cygwin window, cd to sw/memTest. Note you will need to cd to cygdrive/c to get at your c: drive root. Eg

cd /cygdrive/c/myProjects/cyc2-openrisc/sw/memTest

make clean all

This will build memTestSim.8bit.hex, and copy the file to the sim directory

as memory.hex

Build and run the simulation

Build the project

build_icarus.bat

and run the simulation

run_icarus.bat

You should see DRAM activity reported to the

command window. You can turn this off by setting Debug = 1'b0 in model/mt48lc2m32.v. When you see the DRAM write activity finish, leave the simulation to run for one more minute (to let the UART output complete), and then stop the simulation using

^C ^C

And quit the simulation.

View the results

You can view FPGA block RAM activity in sram.log, and the UART output in uart.log

From file explorer, browse to the sim directory and execute viewWave.bat

From the GTKWave application;

Search >> Signal Search Tree

and browse design hierarchy and select the signals you wish to view.

Note, if you want to test changes to a peripheral, it may be easier and faster to simulate the peripheral separately from the cyc2-openrisc project. See spiMaster project for example.

8 Clocks

Name	Source	Rates (MHz)			Remarks	Description
		Max	Min	Res		
clk	Input Pad	48.02	47.98	-	Duty cycle 50/50.	System clock. Restricted to 48Mhz for compatibility with USB IP core.

Table 1: List of clocks

9 IO Ports

Port	Width	Direction	Description
System			
clk	1	input	System clock. 48MHz
watchDogStrobe	1	output	Watch dog output pulse. Monitored by external device. Indicates that System is operating correctly.
OpenRISC UART port			
uart_stx	1	output	Tx data
uart_srx	1	input	Rx data
OpenRISC JTAG debug port			
jtag_tdi	1	input	Data in
jtag_tms	1	input	Mode select
jtag_tck	1	input	Clock
jtag_trst	1	input	Reset
jtag_tdo	1	output	Data out
SDRAM			
mc_addr	12	output	Row/Column Address
mc_ba	2	output	Bank address
mc_dq	32	inout	Data
mc_dqm	4	output	Data mask
mc_we_	1	output	Write enable
mc_cas_	1	output	Column address strobe
mc_ras_	1	output	Row address strobe
mc_cke_	1	output	Clock enable
sdram_cs	1	output	Chip select
sdram_clk	1	output	Clock
SPI			
spiClk	1	output	Clock
spiMasterDataIn	1	input	Data in
spiMasterDataOut	1	output	Data out
spiCS_n	1	output	Chip select
Santa Cruz expansion bus			
SC_P_CLK	1	output	Clock.
SC_RST_N	1	output	Reset

Port	Width	Direction	Description
SC_CS_N	1	output	Chip select
SC_P	40	User defined	User defined

Table 2: List of IO ports

10 Wishbone Datasheet

WISHBONE DATASHEET for USBHostSlave IP Core		
Description	Specification	
General Description:	8-bit slave input and output port	
Supported cycles:	SLAVE READ/WRITE	
Data port Size:	8-bit	
Data port granularity:	8-bit	
Data port, max operand size:	8-bit	
Data transfer ordering:	N/A	
Data transfer sequencing:	Undefined	
Supported signal list and cross reference to equivalet WISHBONE signals:	Signal Name	WISHBONE Equiv.
	address_i	ADR_I
	data_i[7:0]	DAT_I()
	data_o[7:0]	DAT_O()
	we_i	WE_I
	strobe_i	STB_I
	ack_o	ACK_O
	clk_i	CLK_I
	rst_i	RST_I

Table 3: WISHBONE data sheet

11 Resource Utilization

Design Entity	Logic Cells	Memory bytes
cyc_or12_mini_top	13401	18075

Table 4 Resource utilization for Altera CycloneEP2C20