Example Code and Tools
for Virtex-II Prototyping Board

Version 1.1, June 2003

Contents

Revision History

Date	Version	Description
March 14, 2003	1.0	Initial release
June 15, 2003	1.1	Some text rearrangements, improved formatting, table of contents New examples *FPGA Heater*, *Tetris Game* and *UART Demo* C subroutines overview added

Information about new products and new developments can be found on the ErSt Electronic Website:
http://www.erst.biz or **http://www.erst.ch**

ErSt Electronic GmbH
Aeschstrasse 171
CH-8123 Ebmatingen
Switzerland

Phone +41 1 980 61 44
Fax +41 1 980 61 30
Internet **info@erst.ch**
http://www.erst.biz
http://www.erst.ch

Example Code

Overview

Xilinx project files (.NPL) are available for each possible board configuration. They assume the following directory structure:

Main Directory, e.g. v2demo
Contains all source and constraints files (.c, .vhd and .ucf) as well as all the ones generated by the C-compiler and the batch files.

Subdirectory, e.g. 2v1000
Initially, this subdirectory only contains the .npl and .edf files.
The synthesis process will place its own files in this directory too.
The final bit stream can also be found here.

This directory structure is present on the CD. You can just copy the whole "example_code**"** directory to your hard disk and load the appropriate project file.

Ready to use bit streams and PROM configuration files are collected in the "bit_stream" directory.

To compile the C source code you need the free version of the CC5X compiler. It can be found in the software section of the CD. Unzip the *cc5xfree.zip* file into a directory and set the PATH variable to this directory.

Note: The build of the example code (C source compilation, batch files) only works on Win32 platforms.

Note: To edit the PATH variable

1) Open the *control panel*, then *System*
2) Choose the *Advanced* tab and press the *Environment Variables* button
3) Edit the path variable

Hardware Prerequisites

If not otherwise noted, all the examples assume the following hardware configuration:

- The VCCO voltages of all banks are set to 3.3V
- The text-to-speech processor is connected with the FPGA
- Both on-board crystal oscillators are selected

Compilation and Implementation

To implement a design you need to do the following:

1) Set up the recommended directory structure (see above) and copy all source files there:

   In the main directory: All C and VHDL source files, the user constraints files (.UCF) and all project specific batch (.BAT) files.

   In the subdirectory: **p16core.edf** and the project file **xxx_2vyyyy.npl** where xxx refers to the project name and yyyy refers to your particular FPGA device**.** Such a structure is present on the CD and you can just copy the whole directory tree to your hard disk.

2) Use the given .NPL file if you use Xilinx software or create a VHDL design flow project for your target device.

3) Depending on the target device you must use either the *_FF896.ucf (for XC2V1000-2000) or the *_FF1152.ucf (for XC2V3000-8000) user constraints file during implementation.

4) Compile the C source using the *compile.bat* utility to generate the ROM entities rom1.vhd (and rom2.vhd if needed).

5) Synthesize and implement the design and create a bit stream.

FPGA Heater

Description

The purpose of this example is to show how power is consumed by the FPGA by toggling many flip flops concurrently at a high rate. As a consequence, the FPGA heats up. By measuring the temperature and comparing it with a predefined threshold, the equilibrium temperature is kept constant by turning the internal clock off and on.

The implementation consists of a large shift register with 8000 flip flops in the FPGA. The clock of the shift register is generated with a DCM (digital clock manager) in frequency synthesizer mode that generates a 200 MHz internal clock from the 50 MHz input frequency. An additional 100 I/Os are connected to the shift register to consume additional power in the IO ring.

The temperature control logic is implemented in the PIC16 processor core that executes the main program written in C. If the FPGA temperature reaches a predefined limit, the 200 MHz clock is turned off until the FPGA has cooled down below the threshold.

You can enter a new limit temperature by pressing the **yellow** button followed by two digits. Although you can enter values between 0 ^{o}C and 99 ^{o}C, only values above ambient temperature make sense since the FPGA can not cool below the temperature of the environment. In the default configuration the maximum die temperature reaches about 80 to 85 ^{o}C.

Application: By setting the limit temperature above the maximum (free running mode) you can test the efficiency of various coolers (plates, cooling fins, coolers with fan etc.) attached to the metal heatsink on top of the BGA package.

File List

The following files are needed to build the example:

- **heater.c:** This is the C code for the program. It contains the temperature control logic.
  - **rom1.vhd:** This file is generated from the above source file when running *compile.bat* (see description). It contains the ROM entity with initialization vectors for the processor.
  - **rom1.mem:** This file contains the raw ROM contents and is used by *data2bram* to modify the bit stream.
  - **rom1.bmm:** This file contains the block RAM memory map for use with the *data2bram* utility.
- **lcd.c:** This file contains routines to access the liquid crystal display. It is located in the **common** directory.
- **i2c.c:** This file contains routines to access the temperature sensor device over the I^{2}C serial bus. It is located in the **common** directory.
- **adm1021.c:** This file contains routines to access the ADM1021 temperature sensor. It is located in the **common** directory.
- **util.c:** This file contains routines used to convert from binary to BCD and vice versa. It is located in the **common** directory.
- **heater.vhd**: This file contains the **main entity** of the design. It is the root of the design hierarchy.
- **p16core.edf:** EDIF netlist of the PIC16 processor core. It can be used for any FPGA device and must be copied to your project subdirectory.
- **p16lib.vhd:** A package that contains some types and constants used by the PIC16 core.
- **heater_FF896.ucf:** This is the user constraints file for the FF896 package. It must be used for XC2V1000 to XC2V2000 FPGAs.
- **heater_FF1152.ucf:** This is the user constraints file for the FF1152 package. It must be used for XC2V3000 to XC2V8000 FPGAs.

Tetris Game

Description

This is a very simple version of the popular tetris game. It demonstrates the use of the liquid crystal display, the LEDs, the buttons and the text-to-speech processor (TTSP). The implementation consists of a microprocessor (PIC16 processor core) that executes the main program written in C. Clock generation and I/O buffers are implemented with VHDL.

The rules of the game are very simple. Objects fly in from the left side of the display and proceed until they encounter an obstacle that is either the right side of the display or any previous object that encountered an obstacle. It is your task to shift and rotate the flying object such that is (partially) fits into a hole in the obstacle. Whenever a column is completely filled, this column is removed and the score counter is incremented.

The speed of the flying objects is increased to a certain limit with the number of completed columns. If the stack height grows above half the display width, speed is decreased.

The game ends if the stack grows too high, e.g. reaches the left side of the display.

You can control the flying object with the following buttons:

Digit 2: Shift object down by one line.

Digit 8: Shift object up by one line.

Yellow: Rotate object counter clock wise by 90 degrees.

Blue: Rotate object clock wise by 90 degrees.

Digit 0: Drop object, e.g. object is no longer controllable and flies as far as possible.

Volume control of the TTSP is possible with switch 6-8 of the left DIP-switch block (see v2demo below).

File List

The following files are needed to build the example:

- **tetris.c:** This is the C code for the program. It contains the main game logic.
  - **rom1.vhd:** This file is generated from the above source file when running *compile.bat* (see description). It contains the ROM entity with initialization vectors for the first processor.
  - **rom1.mem:** This file contains the raw ROM contents and is used by *data2bram* to modify the bit stream.
  - **rom1.bmm:** This file contains the block RAM memory map for use with the *data2bram* utility.
- **lcd.c:** This file contains routines to access the liquid crystal display. It is located in the **common** directory.
- **ttsp.c:** This file contains routines to access the text-to-speech processor. It is located in the **common** directory.
- **tetris.vhd**: This file contains the **main entity** of the design. It is the root of the design hierarchy.
- **p16core.edf:** EDIF netlist of the PIC16 processor core. It can be used for any FPGA device and must be copied to your project subdirectory.
- **p16lib.vhd:** A package that contains some types and constants used by the PIC16 core.
- **tetris_FF896.ucf:** This is the user constraints file for the FF896 package. It must be used for XC2V1000 to XC2V2000 FPGAs.
- **tetris_FF1152.ucf:** This is the user constraints file for the FF1152 package. It must be used for XC2V3000 to XC2V8000 FPGAs.

UART Demo

Description

The purpose of this example is to show how data can be transferred between the prototyping board and a PC by means of the serial interface. An RS-232 transmitter/receiver is implemented in the FPGA and controlled by a C program. Baud rate is fixed at 57,600 kbits/s.

The counterpart on the PC side is provided as well and is named **commio**. To connect the board with the PC you need a cable that has a female D-Sub 9 connector on one side and a male D-Sub 9 connector on the other side. The cable must connect the pins of the connectors in a one-to-one fashion.

The demonstration can run in two modes that can be selected by switch 5 of the left DIP-switch block:

Echo mode (switch 5 off): In this mode the board waits for a key to be pressed and then sends a key code value to the PC. The *commio* program receives the key code, prints it to the standard output and sends the code back to the board. This in turn receives the key code and displays it on the LCD. You can verify this by disconnecting the serial cable. You will now neither see anything displayed on the PC nor on the LCD.

The *commio* program must be started with the *–e* option to work in echo mode.

Narrator mode (switch 5 on): In this mode the board receives single words that are sent to the text-to-speech processor that converts them to spoken speech. The input of the *commio* program is a simple text file. This file is scanned for words (separated by spaces or new lines) that are sent to the board via the serial interface.

The *commio* program must be started with the *–n file* option to work in narrator mode. There are several .txt files in the *commio* subdirectory that you can download and listen to.

Volume control of the TTSP is possible with switch 6-8 of the left DIP-switch block (see v2demo below). However, volume is only affected if the setting is changed before a new text is downloaded.

File List

The following files are needed to build the example:

- **uartdemo.c:** This is the C code for the program. It controls the serial receiver and text-to-speech conversion.
  - **rom1.vhd:** This file is generated from the above source file when running *compile.bat* (see description). It contains the ROM entity with initialization vectors for the processor.
  - **rom1.mem:** This file contains the raw ROM contents and is used by *data2bram* to modify the bit stream.
  - **rom1.bmm:** This file contains the block RAM memory map for use with the *data2bram* utility.
- **lcd.c:** This file contains routines to access the liquid crystal display. It is located in the **common** directory.
- **ttsp.c:** This file contains routines to access the text-to-speech processor. It is located in the **common** directory.
- **uartdemo.vhd**: This file contains the **main entity** of the design and instantiates a serial receiver and transmitter. It is the root of the design hierarchy.
- **uart.vhd:** This file contains the implementation of a RS-232 transmitter and receiver for 8 data bits, no parity bit and 1 stop bit. The baud rate is fixed at 57,600 kbits/s for an input frequency of 50 MHz.
- **p16core5.edf:** EDIF netlist of the PIC16 processor core. In contrast to the *p16core.edf* this processor core supports five 8-bit ports instead of four. It can be used for any FPGA device and must be copied to your project subdirectory.
- **p16lib.vhd:** A package that contains some types and constants used by the PIC16 core.

- **uartdemo_FF896.ucf:** This is the user constraints file for the FF896 package. It must be used for XC2V1000 to XC2V2000 FPGAs.
- **uartdemo_FF1152.ucf:** This is the user constraints file for the FF1152 package. It must be used for XC2V3000 to XC2V8000 FPGAs.

CommIO

The **commio** program, located in the *commio* subdirectory, implements the PC part of this demonstration. It can be run in two modes, echo mode and narrator mode.

In echo mode the program simply writes the received data to the standard output (screen) and sends the data back to the prototyping board where the same data is displayed on the LCD.

In narrator mode the program reads a text file and sends individual words (terminated by spaces) to the prototyping board where these words are sent to the text-to-speech processor that converts them to spoken speech.

Command Line

commio *[-e] [-n file]*

Options and parameters

-e	Echo mode, send back each received frame
-n file	Narrator mode, text-to-speech processor narrates text in file 'file'

V2demo

Description

This example code demonstrates the usage and function of several peripheral components. These are the liquid crystal display, the temperature sensor, the real time clock and the text-to-speech processor. The whole implementation consists of two microprocessors (PIC16 processor cores) and VHDL code for clock generation and input/output buffers.

The example code executes several tasks:

- The user LEDs flash randomly, one or more at the same time with pauses of random duration.
- The text-to-speech processor narrates a welcome message. Pressing a number key or the yellow or blue key also generates a spoken response.
- The LCD provides a menu that allows you to set/display the time/date and to measure the ambient temperature and the temperature of the FPGA.

The left DIP-switch block (the one next to the JTAG configuration connector) allows for the following configuration:

Switch 1: Assignment of LCD. The LCD is assigned to time/date and temperature function if this switch is in the OFF position. When the switch is in the ON position, the LCD is assigned to the text-to-speech processor.

Switch 6-8: Volume control of TTSP. Maximum volume is configured when all three switches are in the OFF position. Minimum volume is configured if all three switches are in the ON position. Other combinations specify intermediate binary coded values.

The FPGA operation can be interrupted and restarted by pressing the (red) reset button.

File List

The following files are needed to build the example:

- **v2demo_1.c:** This is the C code for the program of the first processor. It contains routines to access the liquid crystal display, the temperature sensor and the real time clock.
  - **rom1.vhd:** This file is generated from the above source file when running *compile.bat* (see description). It contains the ROM entity with initialization vectors for the first processor.
  - **rom1.mem:** This file contains the raw ROM contents and is used by *data2bram* to modify the bit stream.
  - **rom1.bmm:** This file contains the block RAM memory map for use with the *data2bram* utility.
- **v2demo_2.c:** This is the C code for the second processor. It contains routines to access the text-to-speech processor.
  - **rom2.vhd:** This file is generated from the above source file when running *compile.bat* (see description). It contains the ROM entity with initialization vectors for the second processor.
  - **rom2.mem:** This file contains the raw ROM contents and is used by *data2bram* to modify the bit stream.
  - **rom2.bmm:** This file contains the block RAM memory map for use with the *data2bram* utility.
- **lcd.c:** This file contains routines to access the liquid crystal display. These routines are use by both processors. The file is located in the **common** directory.
- **v2demo.vhd**: This file contains the **main entity** of the design. It is the root of the design hierarchy.
- **randled.vhd:** This file contains an entity that controls the LEDs.
- **p16core.edf:** EDIF netlist of the PIC16 processor core. It can be used for any FPGA device and must be copied to your project subdirectory.
- **p16lib.vhd:** A package that contains some types and constants used by the PIC16 core.

- **v2demo_FF896.ucf:** This is the user constraints file for the FF896 package. It must be used for XC2V1000 to XC2V2000 FPGAs.
- **v2demo_FF1152.ucf:** This is the user constraints file for the FF1152 package. It must be used for XC2V3000 to XC2V8000 FPGAs.

ErSt Electronic GmbH
Aeschstrasse 171
CH-8123 Ebmatingen
Switzerland

Phone +41 1 980 61 44
Fax +41 1 980 61 30
Internet **info@erst.ch**
http://www.erst.biz
http://www.erst.ch

Tools (Executable Files)

These tools are located in the **tools** directory of the CD-ROM. They are used by the various batch files to convert data files.

bin2vhd.exe

The **bin2vhd** utility converts a binary image to a synthesizable VHDL entity that instantiates four block RAMs with initialization vectors. It also generates .MEM files of the binary image for use with the *data2bram* tool.

Command Line

bin2vhd [*-4][-b][-e name] binary_input vhdl_output*

Options and parameters

-4	Use four RAMB4_S4 block RAMs for ROM
-b	Binary representation of ROM contents, default is hex
-e name	Name of ROM entity, default is 'ROM'
binary_input	This is the binary image that is produced by the hex2bin utility
vhdl_output	The VHDL file that implements the ROM entity consisting of block RAMs whose initialization vectors contain the binary image. This entity can be instantiated in the main code.

hex2bin.exe

The **hex2bin** utility converts .HEX files (the output of the C compiler) to a binary image.

Command Line

hex2bin *input_file output_file*

Parameters

input_file	This file contains the HEX input data (produced by the C compiler)
output_file	This file is generated and contains the binary image of the input

Tools (Batch Files)

These tools are located in the **tools** directory of the CD-ROM. They are used to automate the tasks of C source file compilation, bit stream manipulation and FPGA configuration.

c2vhd.bat

This batch file runs the compiler, converts the .HEX output to binary and finally calls the bin2vhd utility.

Command Line

c2vhd *c_file entity_name*

Parameters

c_file	Name of the C source file without the .c extension
entity_name	Name of the ROM entity, e.g. 'rom1'

compile.bat

This batch file just calls the c2vhd.bat to compile the C source files. There exists a special version for each example project.

Command Line

compile

Parameters

none

modbram.bat

This batch file calls the *data2bram* tool (Xilinx) to modify the bit stream with the contents of the rom?.mem files. It must be called with the FPGA device type and bit stream file stem as parameters.

Command Line

modbram fpga_type bitstream_name

Parameters

fpga_type	This argument specifies the FPGA type and must be one of 2v1000, 2v1500, 2v2000, 2v3000, 2v4000, 2v6000 or 2v8000
bitstream_name	This is the name stem (name without .bit extension) of the .bit file you want to modify

Examples

1) **modbram 2v3000 tetris**
   Modifies the bit stream *tetris.bit* in the *2v3000* project subdirectory.

2) **modbram 2v1000 v2demo**
   Modifies the bit stream *v2demo.bit* in the *2v1000* project subdirectory.

c2fpga.bat

This batch file runs the *compile.bat* and the *modbram.bat* files and finally calls the *impact* tool (Xilinx) in batch mode to download the bit stream to the FPGA. It is assumed that a parallel cable for JTAG programming is connected to the parallel port 1 (LPT1) of the computer and that the appropriate PROM module is in the JTAG chain.

Command Line

c2fpga fpga_type c_name

Parameters

fpga_type	This argument specifies the FPGA type and must be one of 2v1000, 2v1500, 2v2000, 2v3000, 2v4000, 2v6000 or 2v8000
c_name	This is the name stem (name without .c or .bit extension, respectively) name of the .c and .bit file you want to compile and modify

Examples

1) **c2fpga 2v3000 tetris**
Compiles the *tetris.c* source file and modifies the bit stream *tetris.bit* in the 2v3000 project subdirectory.

2) **c2fpga 2v1000 v2demo**
Compiles the *v2demo_1.c* and *v2demo_2.c* source files and modifies the bit stream *v2demo.bit* in the 2v1000 project subdirectory.

3) **c2fpga 2v6000 uartdemo**
Compiles the *uartdemo.c* source file and modifies the bit stream *uartdemo.bit* in the 2v6000 project subdirectory.

Appendix A: C Subroutines Overview

This appendix lists the C source files of the common subdirectory and gives a short description of each subroutine. Please refer to the appropriate file for routine arguments and return values.

adm1021.c

This file contains routines to access the ADM1021 temperature sensor.

adm1021_read: Read a register from the ADM1021.

adm1021_write: Write to a register in the ADM1021.

adm1021_init: Initialize temperature sensor.

adm1021_wait: Wait till conversion is finished.

adm1021_getLocal: Read local temperature from ADM1021. It waits till the conversion is finished and then reads the local temperature register RLTS.

adm1021_getRemote: Read remote temperature from ADM1021. It waits till the conversion is finished and then reads the remote temperature register RRTE.

button.c

This file contains routines to access the key pad (Digits 0 to 9, blue and yellow key).

btn_isAny: Check if any button is pressed. This routine does not block.

btn_waitRel: Wait until all buttons are released.

btn_getKey: Wait for a key to be pressed and released.

btn_getDigit: Wait for a digit key to be pressed.

btn_getDigit2: Calls *btn_getDigit2* twice to get two key values and returns them packed into one byte, the first key value in the four higher order bits and the second key value in the four lower order bits.

i2c.c

This file contains routines to deal with the I^2C interfaces of the RTC and the temperature sensor. The timing is for a 12.5 MHz processor core.

i2c_delay: Delay to satisfy I2C bus timing (max. 400 kHz).

i2c_scl_high, i2c_scl_low: Set the SCL clock signal high or low, respectively.

i2c_sda_high, i2c_sda_low: Set the SDA data signal high or low, respectively.

i2c_sda_in: Read the current slave data on the SDA line.

i2c_start: Generate a start condition and write slave address.

i2c_stop: Generate a stop condition.

i2c_write: Send 8 bits of data.

i2c_read: Receive 8 bits of data.

i2c_send: Send data to a register at a given address in the slave.

i2c_receive: Receive data from a register at a specified address in the slave.

lcd.c

This file contains routines to control the liquid crystal display. The read/write timing is for a 12.5 MHz processor clock.

delay_1ms: Delay for a multiple of 1 ms @ 12.5MHz.

lcd_write: Write an 8-bit value to the LCD.

lcd_read: Read an 8-bit value from the LCD.

lcd_wait: Wait till busy flag is clear.

lcd_writeInstr: Write instruction code to instruction register.

lcd_writeData: Write to data register.

lcd_clear: Clear display, cursor home.

lcd_init: Initialize LCD. Calls this routine before any of the others.

lcd_setPos: Set cursor position to a given row and column.

lcd_printBCD: Show two digit BCD value at current position.

lcd_printChar: Print a character at the current position.

lcd_printString: Print a string at the current position.

lcd_printStringAt: Print a string at the given position.

rtc8564.c

This file contains routines to access the real time clock.

rtc_read: Read a register from the RTC-8564.

rtc_write: Write to a register in the RTC-8564.

rtc_getSec, rtc_getMin, rtc_getHour, rtc_getDay, rtc_getMonth, rtc_getYear: Functions to read the actual time.

rtc_setSec, rtc_setMin, rtc_setHour, rtc_setDay, rtc_setMonth, rtc_setYear: Functions to set the actual time.

rtc_init: Initialize the real time clock chip.

ttsp.c

This file contains routines to access the text-to-speech processor.

getVolume: Read DIP-Switch configuration for volume control.

wait_ready: Wait till slave is ready to receive more data.

spi_sendByte: Send a single byte over the SPI bus.

spi_sendCmd: Send a command word, type I - single word transaction.

spi_sendData: Send command with data, type III - transaction that sends data.

spi_readData: Send command that reads data, type IV - transaction that reads data.

wts_setReg: Set a WTS701 register to a new value.

wts_init: Initialize the text-to-speech processor. Call this routine before any other ones from this file.

text_to_speech: Convert a text to speech.

util.c

This file contains some general purpose utility routines.

convBCD: Convert a binary value to BCD. This only works for numbers between 0 and 99.

convBinary: Convert a BCD value to binary.

ErSt Electronic GmbH
Aeschstrasse 171
CH-8123 Ebmatingen
Switzerland

Phone +41 1 980 61 44
Fax +41 1 980 61 30
Internet **info@erst.ch**
http://www.erst.biz
http://www.erst.ch