Java on FPGA

By Morten Schultz Sørensen (morten.schultz.sorensen@teknologisk.dk)

1 Contents

2 Preface

The Java Optimized Processor (JOP) has been created by Martin Schoeberl as part of his Ph.D. project (2005) and is continuously being developed as an open source project. JOP is a soft-core Java-processor written in VHDL which makes it possible to execute Java code on a field-programmable gate array (FPGA). JOP is designed to be time predictable, and the number of Java byte codes (JVM instructions set) can be predicted in clock cycles. It is therefore possible to build real time Java applications.

Why use Java in embedded systems? The introduction of Java in embedded systems enables multithreading and object-oriented programming (OOP). It is easier to create more readable and safer coding with Java which is also platform independent. The use of JOP will facilitate the acceptance of Java for embedded systems, and furthermore Java can broaden the use of FPGA.

Since JOP operates on a FPGA, it is easy to implement new peripheral devices to the processor. It provides the possibility of creating custom designed hardware for specifically required specifications. It also provides flexible boundaries between hardware and software. For example, time consuming tasks or real time specifications which cannot be obeyed with Java can be moved from software to hardware.

For real-time systems, critical parts of the Java code can be optimized in terms of creating customized Java-byte code which calls a set of implemented JOP-microcode instructions. The alternative would be to design new JOP-microcode instructions implemented in VHDL and executed directly in hardware. It is thereby possible to reduce e.g. math calculations written in Java code into taking only one processor cycle and reducing the execution time by more than 1000 %.

Java combined with JOP is an interesting platform applicable for real-time embedded systems. JOP is able to handle even hard real-time systems. A system is termed hard real-time if it has deadlines which cannot be missed and in the event the system would fail. JOP is a deterministic platform which is predictable when using a simple architecture. It is possible to calculate the Worse Case Execution Time (WCET) for all critical operations. It requires knowledge of how the Java-code is translated into Java-bytecode and JOP-microcode. In the implementation of the JOP platform, some restrictions have been necessary to enable the fulfillment of the demand for optimal real-time programming. As an example, the Java real-time profile for JOP has no dynamic class loading.

JOP is an open source, and the source code is available at http://www.opencores.org. Additional information about JOP can be found at the official site http://www.jopdesign.com.

This guide will show the basic steps of how to set up JOP from a Windows platform and start it on a Xilinx Spartan 3 starter kit. It contains an example of how to write a simple “Hello World” Java application running on the JOP platform.

JOP requires basic knowledge of Java. Basic VHDL knowledge is required if new peripheral devices have to be added. The standard JOP package for spartan3 starter kit has a serial interface and a watchdog output. JOP is available with a number of different Java and VHDL examples; however some requires the adding of hardware to JOP. This guide will also present the basic procedure of adding hardware to JOP, written in VHDL.
A version of JOP with added button, switch and seven-segment-display driver, for the Spartan 3 starter kit, can be downloaded at the same webpage as this guide.

This guide includes the use of the following items:

Hardware:
- Xilinx Spartan 3 Starter Kit board

Software:
- Xilinx ISE 9.2i
- Java SE Development Kit (JDK) update 6
- Cygwin version 2.573.2.2
- JOP open source project.

3 Windows set-up of required programmes

All required tools are free of charge and available from their respective websites.

3.1 Cygwin

Cygwin is used by software developers to migrate applications from a Linux platform to Windows platform.

Install Cygwin environment from http://www.cygwin.com/

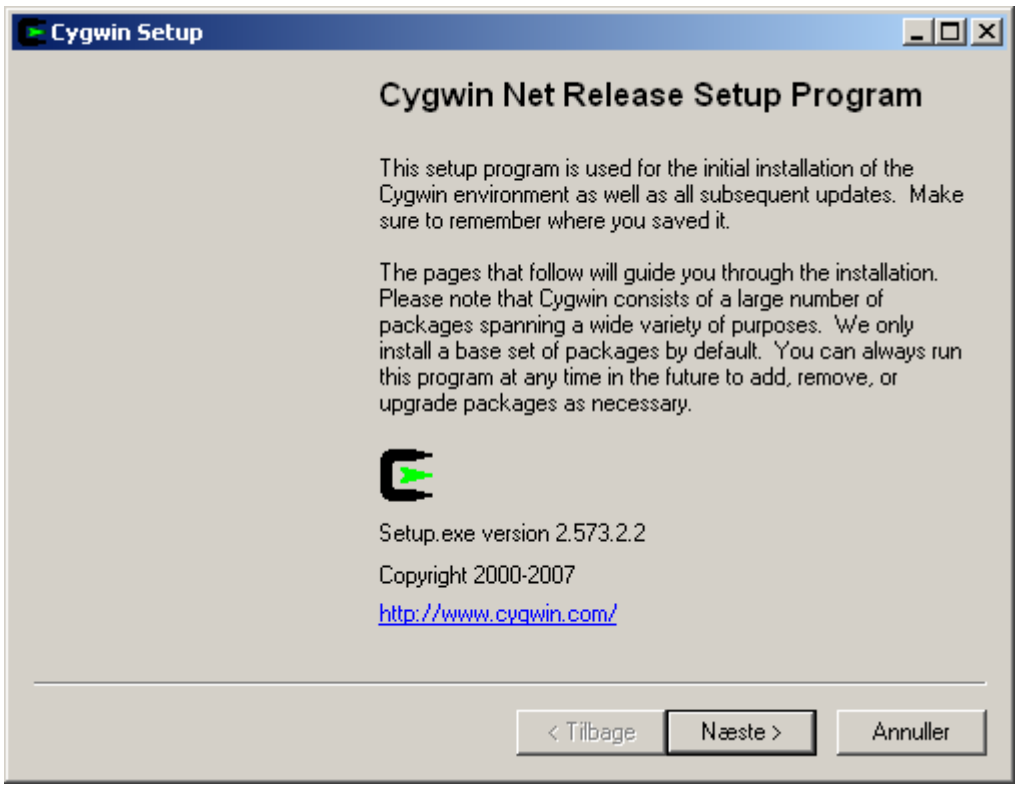

Figure 1 Cygwin Setup

Use the default settings until the "*Select packages*" window as shown in Figure 2.

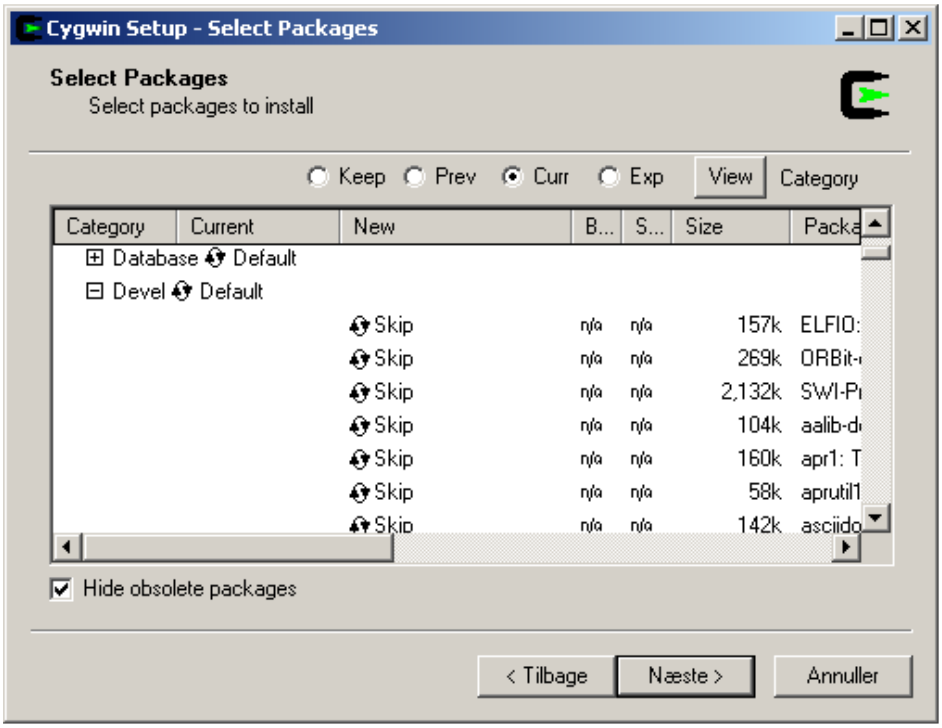

Figure 2 Cygwin setup - Select packages

Select install *"CVS"," GCC"* and *"MAKE"* in the Devel sub-category.

3.2 Java SE Development kit

Install Java SE (JDK) from http://java.sun.com/

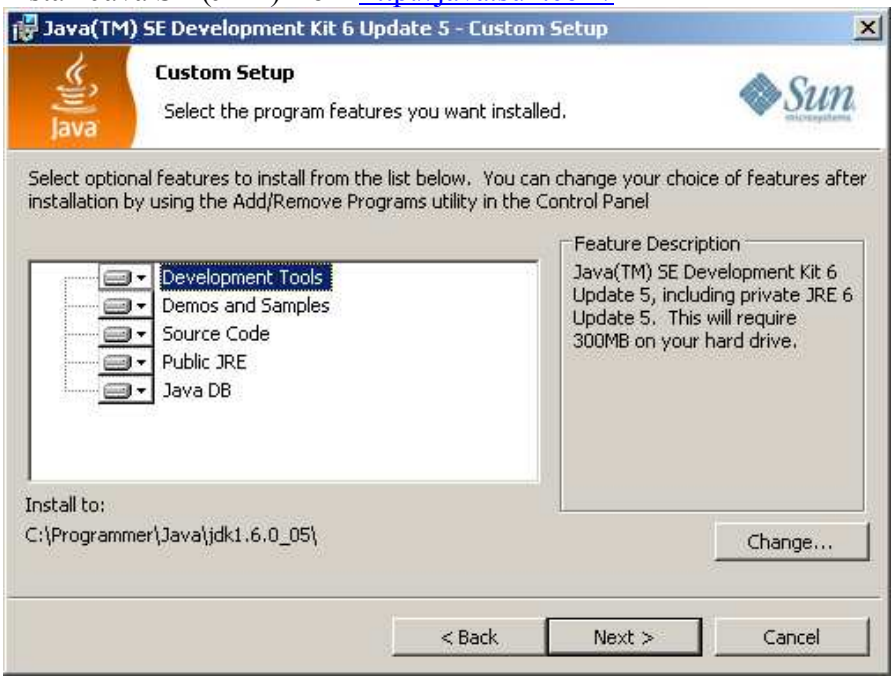

Figure 3 Java SE development kit set-up

Use default settings during installation.

3.3 Xilinx Ise

Install Xilinx web-pack from http://www.xilinx.com/

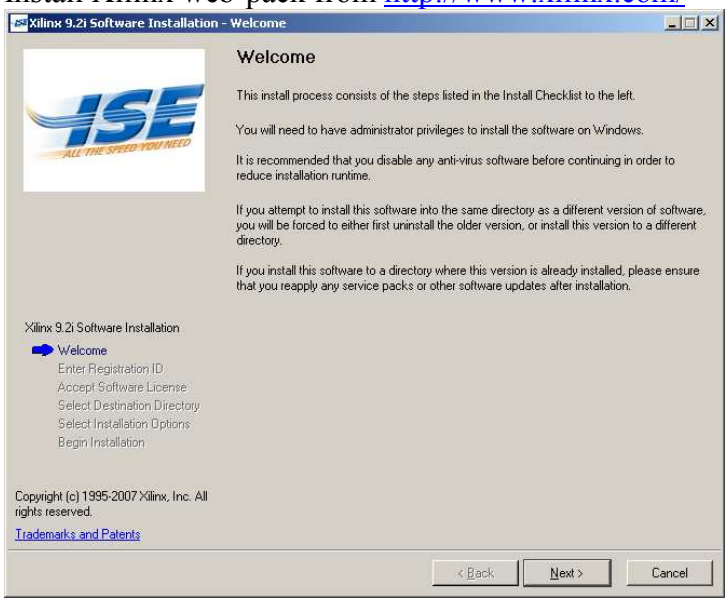

Figure 4 Xilinx Installation

Use default settings during installation.

3.4 Include programme paths

The folders of the following programmes have to be added to global the path: *CVS.exe, GCC.exe Javac.exe and Make.exe.*

Click on "*System*" in Windows "*Control panel*"

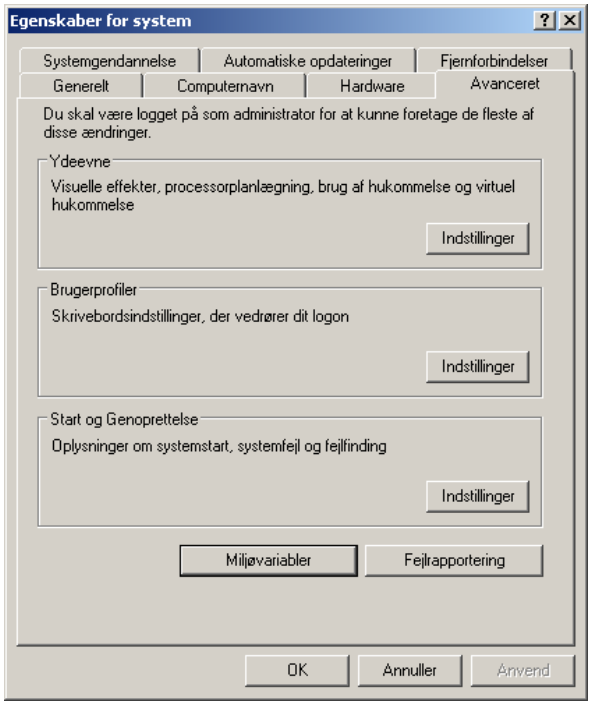

Figure 5 System ➔ Advanced

Press "*Environment variables*"

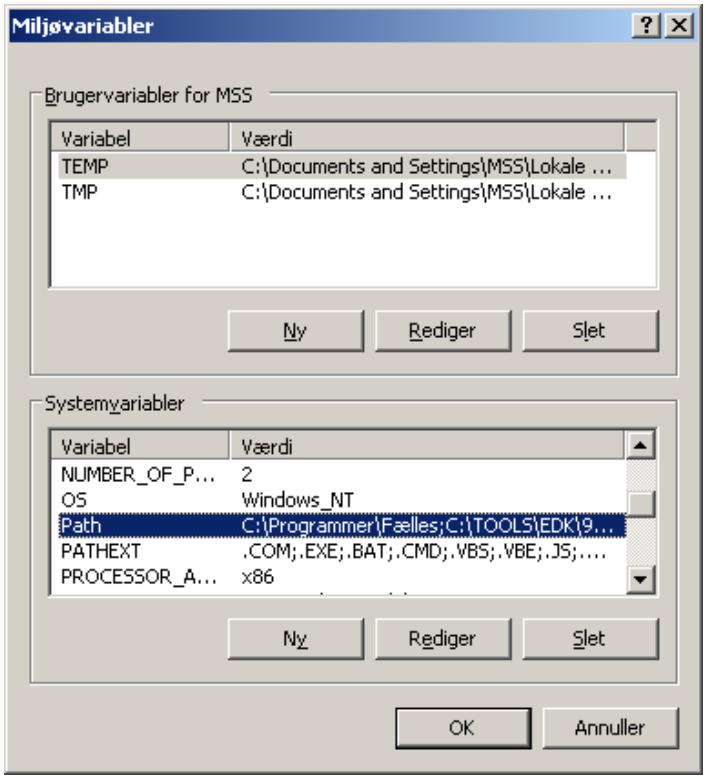

Figure 6 Environment variables

Find and mark "*Path*" in the list box "*System variables*" and press "*Edit*"

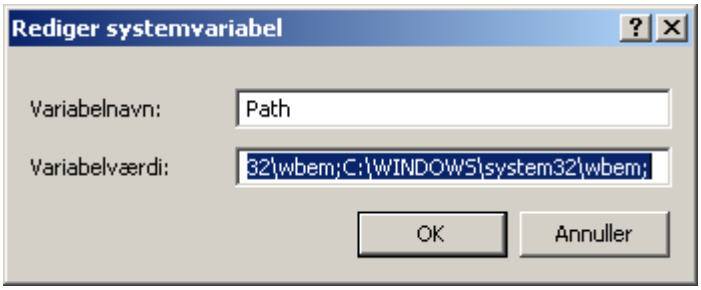

Figure 7 Edit System variables

Include the path to which the programs were installed.

In this guide, the installed path for Cygwin is
"*C:\cygwin\bin*"

The installed path for JDK is
"*C:\Programmer\Java\jdk1.6.0_04*"
Add "*\bin*" to the Java path and separate each location with ";"

Check that the paths have been correctly inserted by starting a command prompt and typing the following commands.

- *javac*
- *cvs*
- *gcc*
- *make*

The outcome of the commands should be like Figure 8.

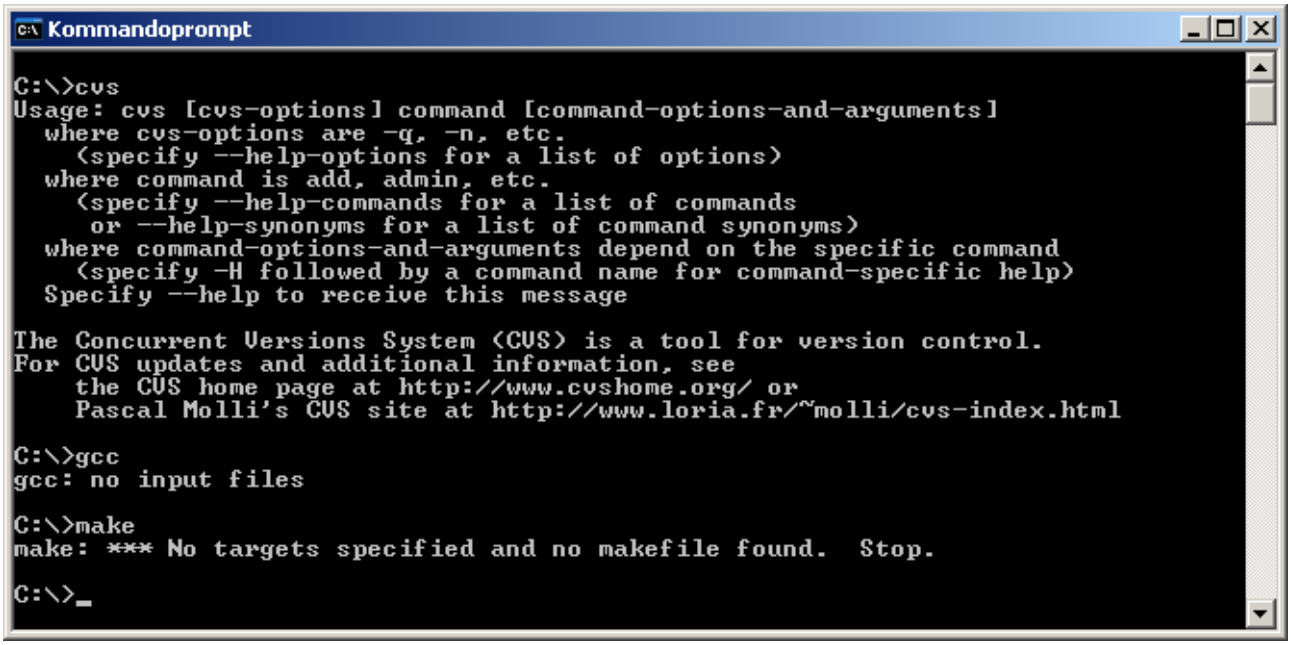

Figure 8 Included paths

4 Download JOP

Start a command prompt and type:

"*Cvs -d :pserver:anonymous@cvs.opencores.org:/cvsroot/anonymous -z9 co -P jop*"

This command will download JOP from http://www.opencores.org/ into the "\JOP" folder. When download is completed, type "*cd jop*" to enter the JOP folder and type "*make tools*" as shown in Figure 9.
Notice: the version of JOP, which can be downloaded at the same webpage as this guide, is not the same version as the one which can be downloaded at Opencores.org, and it should not be used to follow this guide.

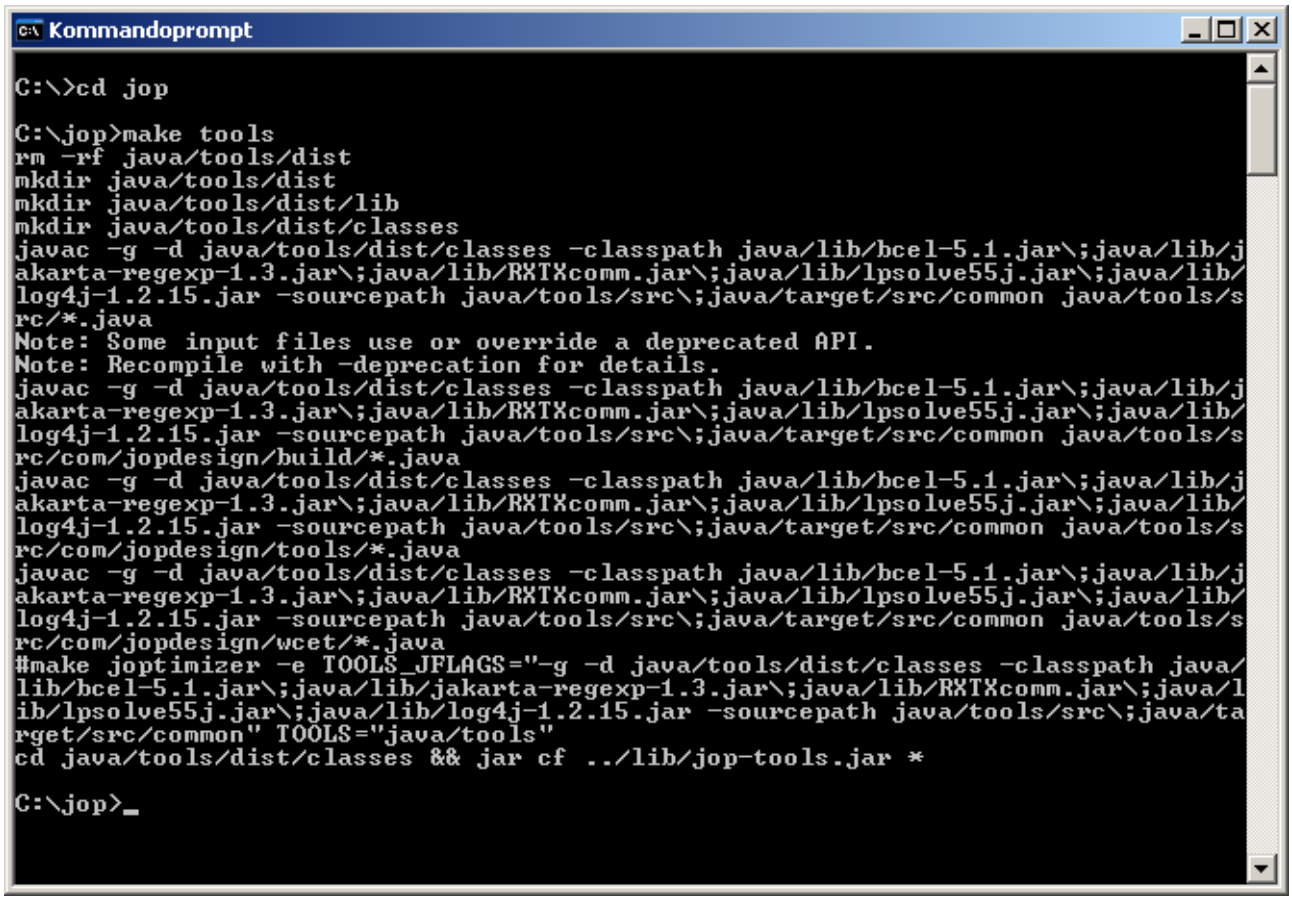

Figure 9 JOP - make tools command

Type "*cd asm*" and next type "*jopser*".

JOP is now ready for Xilinx ISE.

5 JOP and Xilinx ISE.

Connect the programmer cable to the Spartan3 starter kit. Start Xilinx ISE, press "*open project*", go to "*jop\xilinx\s3k*" folder, change the file type from *"*.ise"* to *"*.npl"* and open the *"jop.npl"* project file.

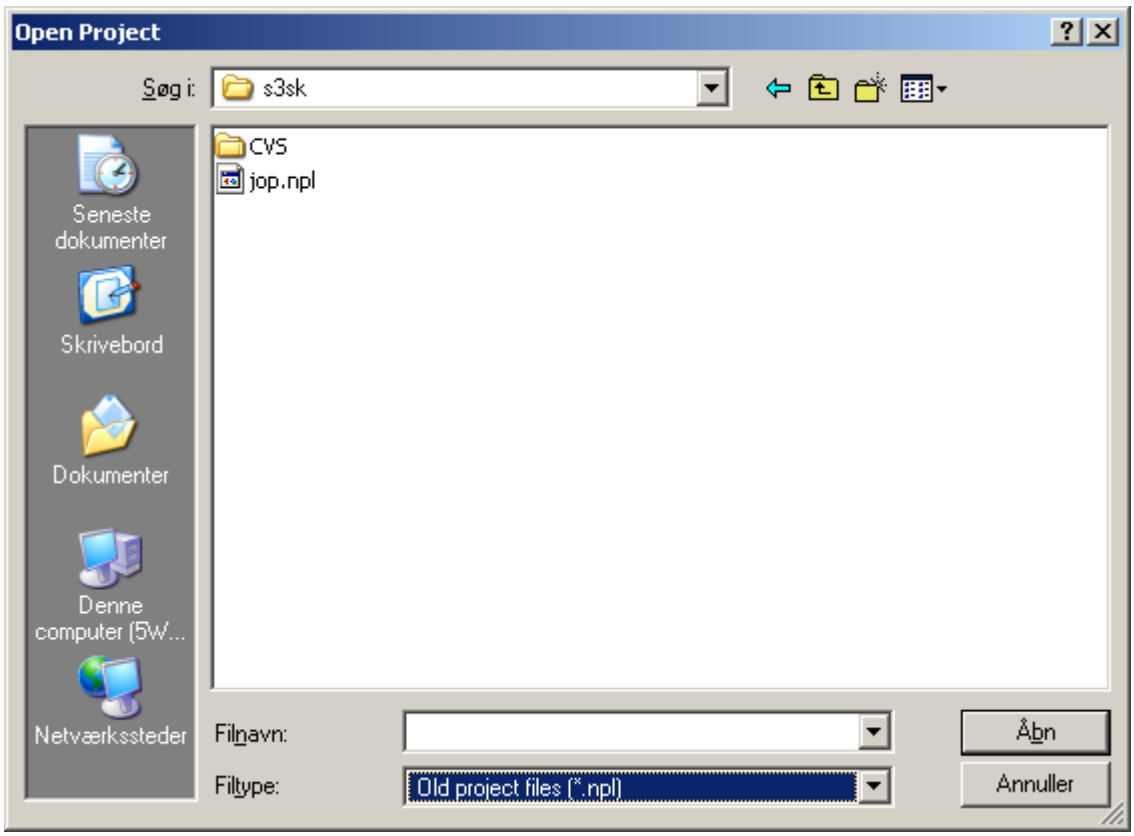

Figure 10 Xilinx Open project

Press "*Yes*" to update the project as shown in Figure 11.

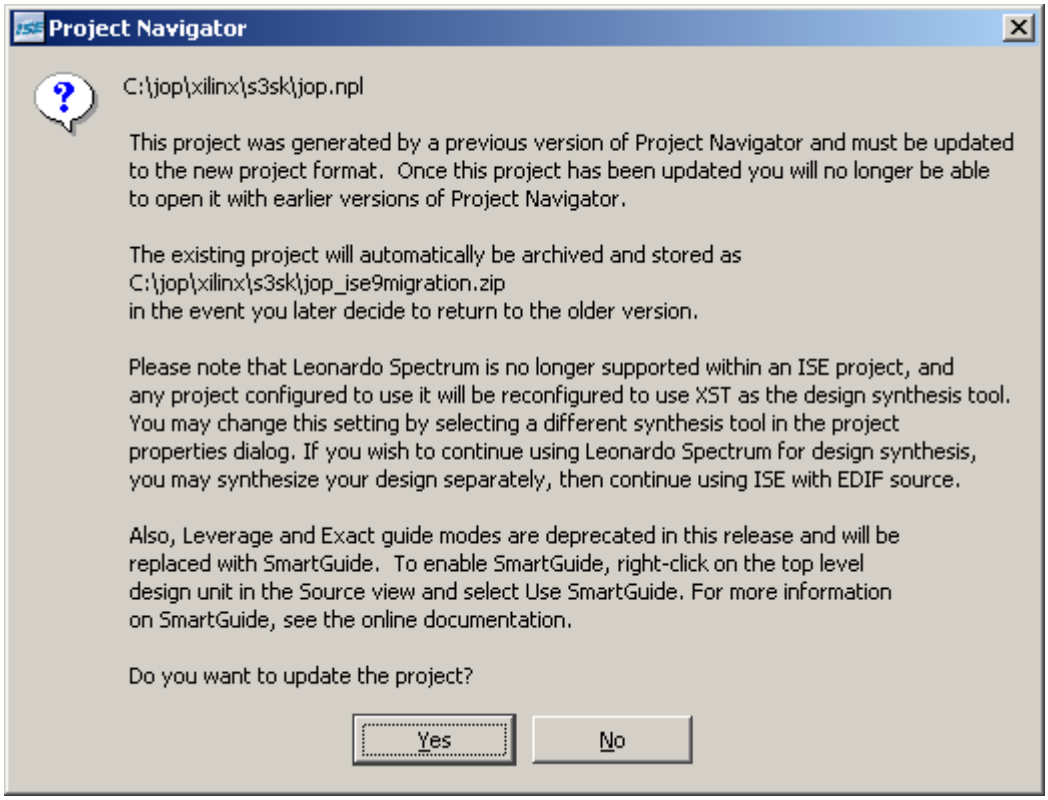

Figure 11 Xilinx ISE - Update project

Press "*Generate PROM, ACE or JTAG file*" and programme the spartan3 board.

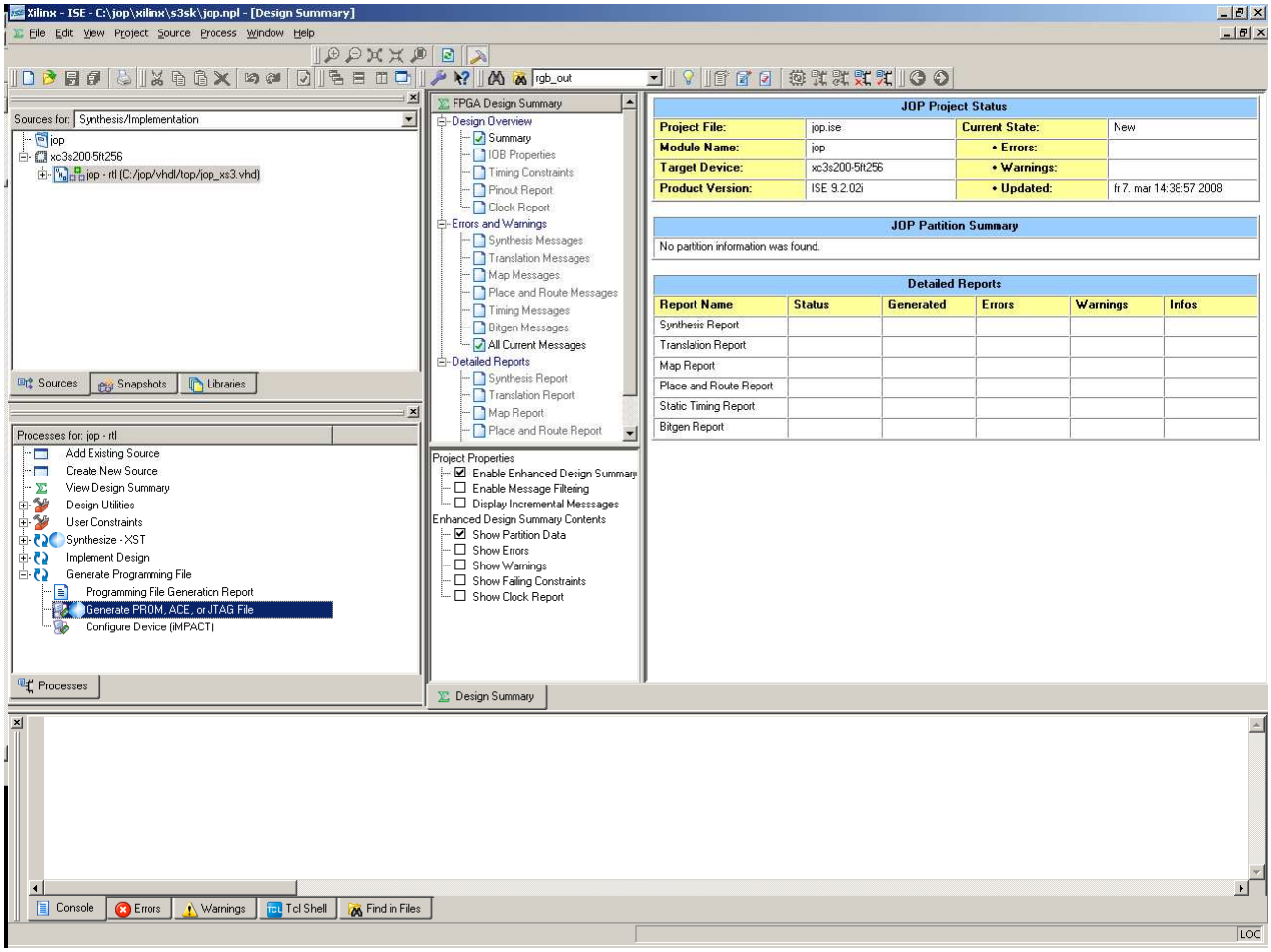

Figure 12 Xilinx generate PROM, ACE or JTAG file

Afterwards select the file "jop.bit" as the new configuration file as shown in Figure 13 and download it to the FPGA. Now JOP is ready to receive a Java application and execute it.

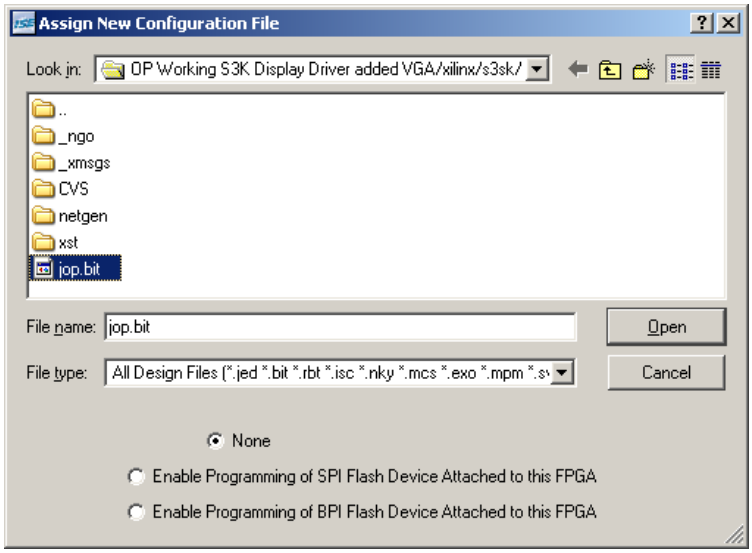

Figure 13 Assign New Configuration File

6 JOP applications

This section shows how to download and run a JOP application. *Open "makefile"* in the *"\JOP"* folder with a text editor such as WordPad. Find *"COM_PORT=COM1"*. If you are using a comport other than "*COM1*", change the number to match your comport. Connect the Spartan 3 Starter kit to the comport.

6.1 "Hello World" Application

Start a command prompt in the "*\JOP*" folder and type "*make java_app*". This will compile the Java application, default is set to a "*Hello World*" application.

After compiling is complete, type "*make download*". The Java application will be downloaded through the serial port to the Spartan 3 starter kit. The Java application is downloaded to JOP's main memory, in this case the memory (RAM) on the Spartan 3 starter kit.

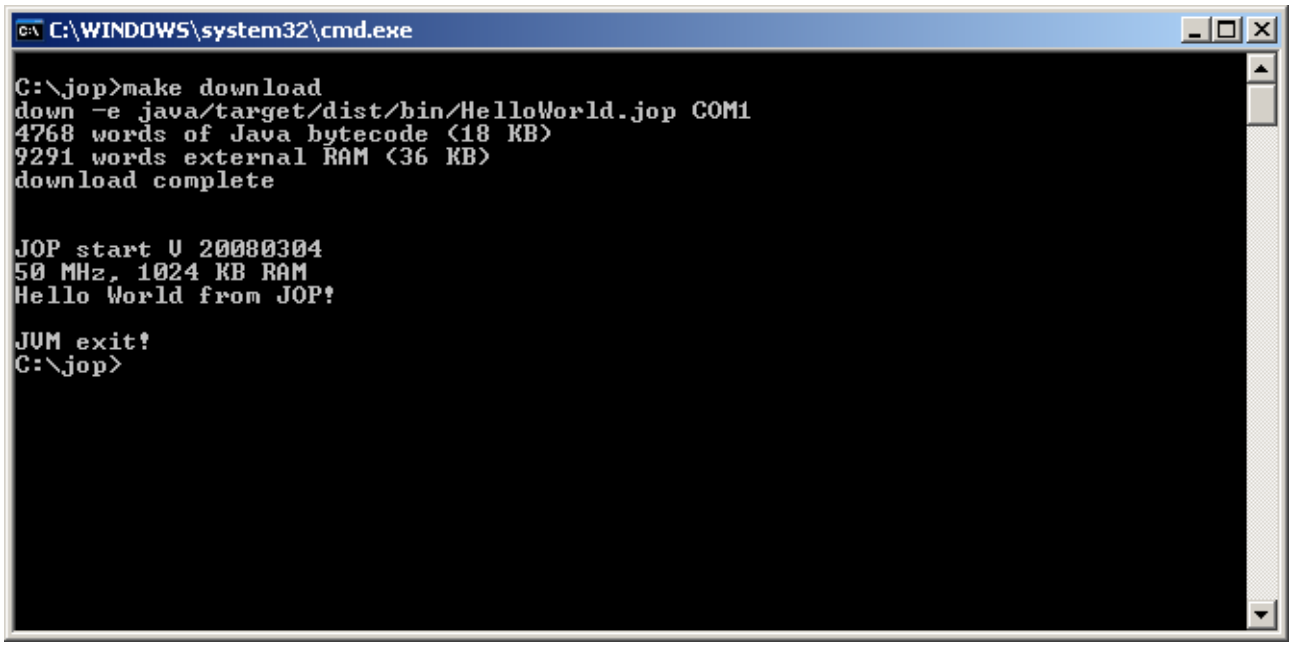

Figure 14 JOP response

When downloading is completed, JOP will respond with a "*Hello World from JOP!*" - see Figure 14. To download another Java application, the Spartan 3 starter kit has to be reconfigured (programmed again) from Xilinx ISE again.
If another application other than "Hello World" is required, open the "*makefile*" again and find:
"

P1=test
P2=test
P3=HelloWorld"
These lines show where to locate the .java file we wish to compile. P1 and P2 are folders; P3 is the application that will be compiled. The path starts in the folder "*\jop\java\target\src*".

6.1.1 Example: "how to use the watchdog I/O"

- Reconfigure the FPGA in Xilinx ISE.
- Create a file and name it "TestWD.java" in the folder "\JOP\java\target\src\test\test\".
- Change "P3" from "HelloWorld" to match the new name "TestWD" in "makefile" in the "\JOP" folder:

"
```
P1=test
P2=test
P3=TestWD
```
"

- Open "TestWD.java" with your preferred Integrated Development Environment (IDE) and write:

"
```
package test;

import com.jopdesign.sys.Const;
import com.jopdesign.sys.Native;

public class TestWD {

public static void main(String[] args) {

            System.out.println("Hello World from JOP!");
            Native.wr.(1, Const.IO_WD);
            }
}
```
"

- Open a command prompt in the "\JOP" folder and type "make java_app". The Java application is now compiled.
- Type "make download". The application will now be downloaded to the Spartan 3 starter kit.

The application starts, writes "Hello World from JOP!" to the UART and turns the led "LD0" on. "LD0" is default the watchdog (WD) output, and it is defined in the JOP Xilinx project.

7 Adding new Hardware

Adding new hardware to JOP requires basic VHDL knowledge.

SimpCon is the system on a chip (SoC) interconnect standard which JOP uses to connect to peripheral devices. SimpCon is a point to point connection between a master and a slave where the master starts read and write transactions.

With default settings, JOP may have up to 8 peripheral devices, each with 16 addresses, and be able to transfer up to 32bit data. The first 3 peripheral device address areas are reserved for JOP to System, UART and USB.

A version of JOP with added button, switch and seven-segment-display driver, for Spartan 3 starter kit, can be downloaded at the same webpage as this guide. This version can be used as an example of how hardware can be added.

7.1 VHDL part

When starting a new project, it is advisable to make a copy of "*scio_min.vhd*" from the folder "*\jop\vhdl\scio\"* and name it after the specific project. For instance, when using Xilinx Spartan 3 starter kit, name it "*scio_xs3sk.vhd*". Afterwards remove the "*scio_min.vhd*" from the project and add "scio_xs3sk.vhd".

The "*scio_xs3sk.vhd*" defines the address range for each peripheral module, and 16 standard addresses are passed on.
The address range starts from the base address which by convention is 0xffffff80.
For instance: "*sc_sys.vhd*" is device number 0. It starts at the base address and is given the first 16 addresses (0xffffff80 to 0xffffff8f). "*sc_uart.vhd*" has device number 1 and is given the next 16 addresses (0xffffff90 to 0xffffff9f) and so on (these values are used for the address mapping in Java).

The constant "*SLAVE_CNT*" in "*scio_xs3sk.vhd*" specifies the number of peripheral devices connected to JOP. JOP uses the first 3 device numbers for basic operations (System, UART and USB).

"*constant SLAVE_CNT : integer := 3;"*

The constant "*DECODE_BITS*" in "*scio_xs3sk.vhd*" specifies the number of bits required to decode the number of slaves ($2^{DECODE\_BITS} >= SLAVE\_CNT$).

"*constant DECODE_BITS : integer := 2;"*

The "*scio_xs3sk.vhd*" instantiate all the peripheral modules and is instantiated by the top-module.

When adding new peripheral devices, it is advisable to make a copy of "sc_test_slave.vhd" from the folder "\jop\vhdl\scio\" and rename the copy "*sc_XXXX.vhd*". The "*XXXX*" is the name of the specific peripheral device. For instance the Uart device is named "sc_uart.vhd".
Clean up the file by deleting the entries to *"cnt"* and "*xyz*". Rename "*sc_test_slave*" associations in

the module to match your new file name. Add the specific VHDL code for the peripheral device. There are two 4 bit multiplexers in this module, one for writing data and one for reading data. It is where the exact address is being connected from Java to VHDL. Here it is possible to get data from Java to hardware and vice versa. The read multiplexer may look like this:
“

```
...
elsif rising_edge(clk) then
        if rd='1' then
                case address(3 downto 0) is
                when "0000" =>
                        rd_data(7 downto 0) <= IO_SlideSwitch_internal;
                        rd_data(31 downto 8) <= (others => '0');
                when "0001" =>
                        rd_data(3 downto 0) <= IO_PushButton_internal;
                        rd_data(31 downto 4) <= (others => '0');
                when others =>
                        -- do nothing
                end case;
        end if;
end if;
...
```

“

In this read multiplexer code, the signals “*IO_SlideSwitch_internal*” or “*IO_PushButton_internal*” are read into Java when accessing the specific address. Please notice that there are 8 slide switches and 4 push buttons in the Spartan 3 starter kit. Since it is possible to transfer 32bit, the unused bit is set at 0.

First add the IO’s to the top layer of the Xilinx XS3 project (“*jop_xs3.vhd*”). Then map the IO’s to “*scio_xs3sk.vhd*”. Map the IO’s from “*scio_xs3sk.vhd*” to “*sc_XXXX.vhd*” together with the required SimpCon connection:
“

```
cmp_XX: entity work.sc_XXXX
generic map (
        addr_bits => SLAVE_ADDR_BITS
)
port map (
        clk => clk,
        reset => reset,
--SimpCon
        address => sc_io_out.address(SLAVE_ADDR_BITS-1 downto 0),
        wr_data => sc_io_out.wr_data,
        rd => sc_rd(X),
        wr => sc_wr(X),
        rd_data => sc_dout(X),
        rdy_cnt => sc_rdy_cnt(X),

-- Your mapping here:
```

“

Notice that X has to be changed to match the device number, for instance the System ("*sc_sys.vhd*") has device number 0.

7.2 Java part

Accessing hardware from Java requires the VHDL address (from "*sc_XXXX.vhd*") to be defined in "*const.java*" which can be found in the folder "*Jop\java\target\src\common\com\jopdesign\sys*". Open this file with a text editor or your preferred IDE.

The base address for hardware mapping is by convention 0xffffff80. Since there are 16 (same as the hex value 0x10) addresses to a peripheral device, the second peripheral device start address is 0xffffff80 + 0x10.

For instance: The system ("*sc_sys.vhd*") has device number 0 which means that it starts at the base address. The watchdog has the address number 3 in the write multiplexer in "*sc_sys.vhd*", the "*const.java*" address mapping for the watchdog may be like this:

```
"public static final int IO_BASE = 0xffffff80;"
"public static final int IO_WD = IO_BASE+3;"
```

As an example the Spartan 3 starter kit push buttons have been added as device number 3. The push buttons have address 1 in the read multiplexer, in "*sc_XXXX.vhd*" VHDL file:

```
"public static final int IO_PUSHBUTTON_S3SK = IO_BASE+0x30+1;
//Pushbutton Input 3 downto 0"
```

In the Java file specified in "*JOP\makefile*" is it now possible to access the hardware – for example by writing:

```
"Native.wr(1, Const.IO_WD);"
```

The watchdog output is set high.

By writing:

```
"Native.rd(Const.IO_PUSHBUTTON_S3SK);"
```

The value of push buttons is returned.