DE NAYER Instituut
J. De Nayerlaan 5
B-2860 Sint-Katelijne-Waver
Tel. (015) 31 69 44
Fax. (015) 31 74 53
e-mail: ppe@denayer.wenk.be
ddr@denayer.wenk.be
tti@denayer.wenk.be
website: emsys.denayer.wenk.be

Microblaze EDK 3.2 Tutorial

Xilinx platform Version 1.01

HOBU-Funds
Project IWT 020079

Title : Embedded systemdesign based upon Soft- and Hardcore FPGA's

Projectleader : Ing. Patrick Pelgrims

Projectassistants : Ing. Dries Driessens
Ing. Tom Tierens

I Introduction

This tutorial is created to help you design your first embedded system with a Microblaze softcore processor. Before you proceed you must have the following software and hardware:

Software:
- Xilinx ISE 5.x
- Xilinx EDK 3.2

Hardware:
- windows-PC
- Xilinx FPGA development board

Chapter II is a general chapter where you can learn how to define your Microblaze embedded system.

In chapter III, there are possible 2 tracks to follow:

TRACK 1: completing the design with Xilinx Platform Studio:
Easiest way to create basic designs without clock dividers or other custom IP-cores or when you don't drive any additional ports on your FPGA.

TRACK 2: design in Xilinx Platform Studio, synthesis and P&R in Xilinx ISE:
This track is the easiest track to drive additional ports or to create more complex designs with clock-dividers or your own IP-cores. By modifying the code itself, you get complete control over the design of your embedded system.

Chapter IV describes how to finally download the embedded system into the FPGA and how to run it.

If things don't go as they should, you can find some hints in chapter V how to solve your problem.

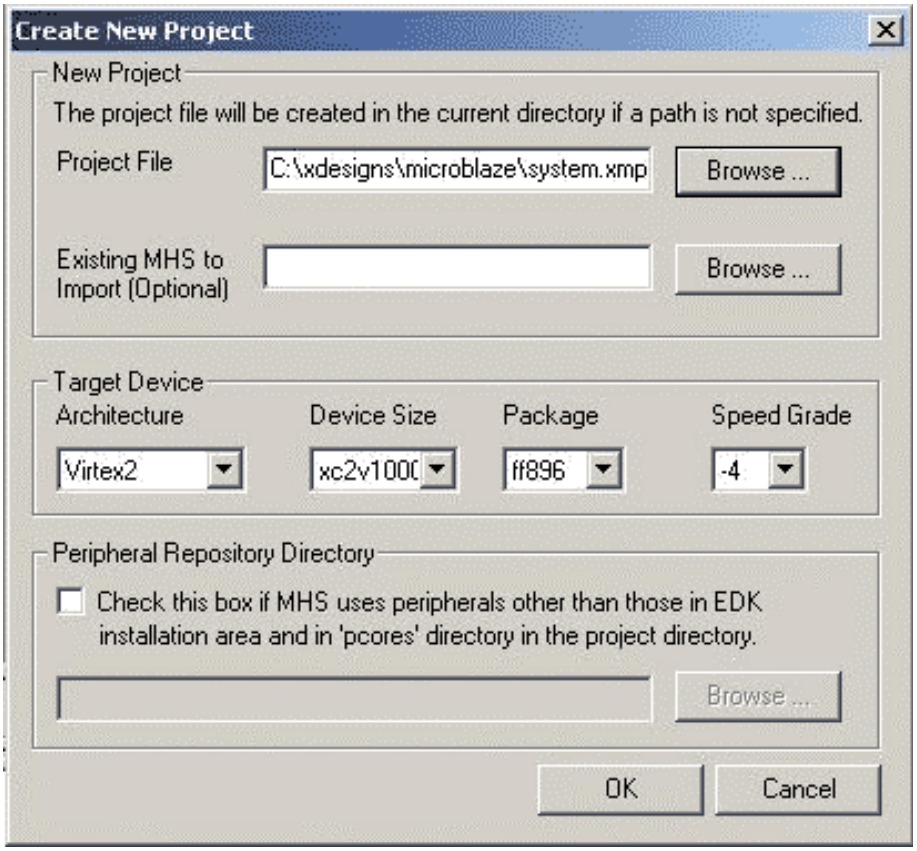
The Addendum has the following chapters:
A. a general description how you can add custom IP-cores
B. an example where a clockdivider IP-block is added in Xilinx Platform Studio
C. an example where the same clockdivider is added to the VHDL-code for synthesis in ISE.

II Defining the embedded system (GENERAL)

1) start Xilinx Platform Studio (XPS) in *Start -> Programs -> Xilinx Embedded Development Kit*

2) select '*New Project'* in the menu '*File'*, a window will appear like figure 1:
   a) fill in the path where the new project should be placed (note: do not use directories that contain spaces in their names)
   b) select the correct target device that your FPGA development board contains
   c) press '*OK'*
   d) Press '*Yes'* when get the warning that you haven't specified an MHS file.

Figure 1: Create New Project Window



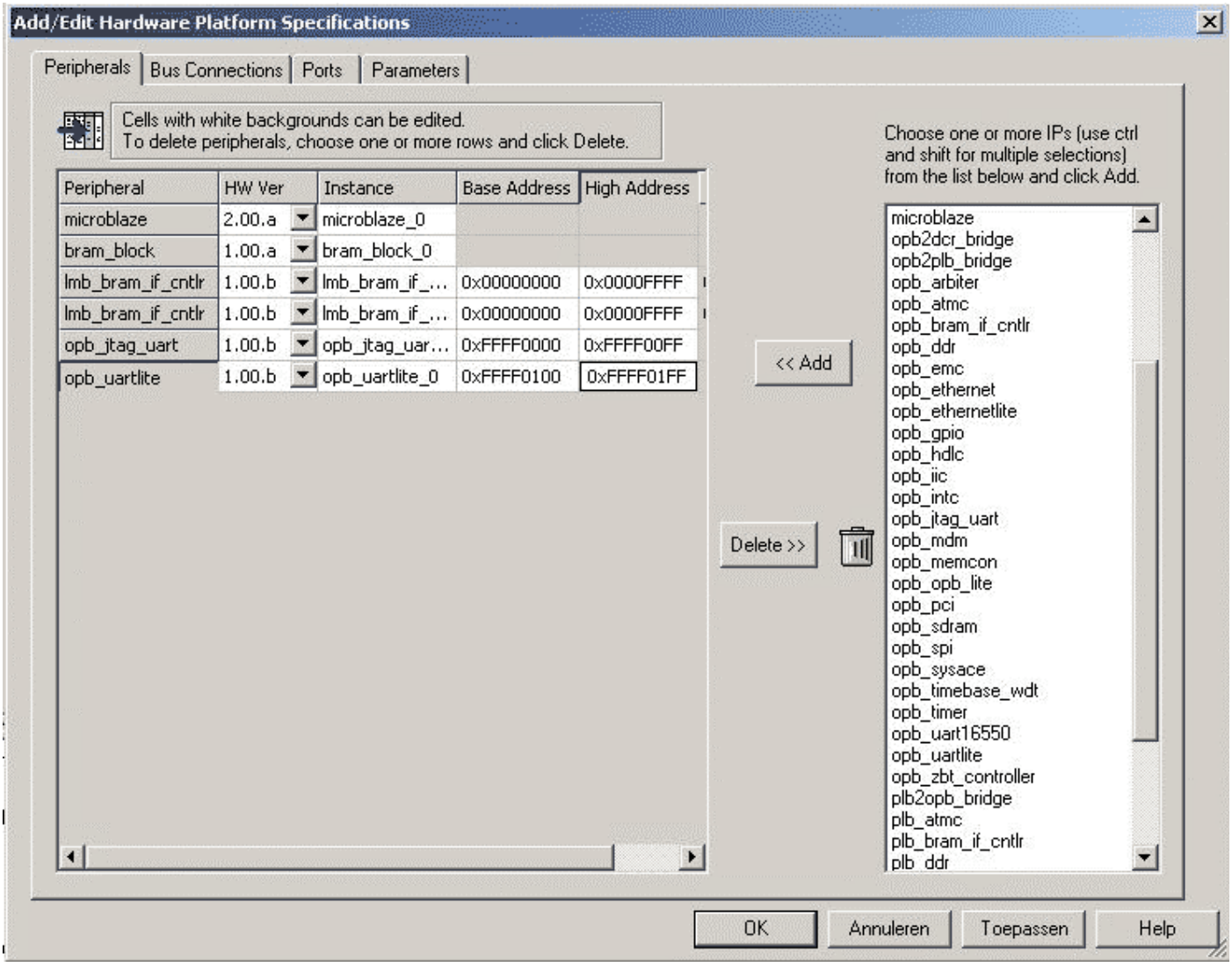
3) Open the Hardware Specifications window:
   a) Click with your right-mouse-button on '*System BSP*' (left side of the screen)
   b) Click '*Add/Edit Cores... (dialog)*'
   c) Specify the necessary peripherals at the '*Peripherals'* tab (figure 2):
      i) Add the following peripherals from the list on the right:
         (1) '*bram_block'*
         (2) two '*lmb_bram_if_cntlr*': one for the instruction and one for the data bus
         (3) '*microblaze'*
         (4) '*opb_gpio*' if you have some leds on your board
         (5) '*opb_uartlite*' if you have a serial connector
         (6) '*opb_jtag_uart*' (you can of course use 2 uartlites, but in this tutorial we don't)
      ii) Correct the addresses of the peripherals:

HINT: XPS needs to create bus logic for the LMB and the OPB bus, so therefor you should have a address range for each bus and the busses shouldn't overlap each-other.

(1) For both lmb_bram_if_cntlrs change:
   (a) Base Address : 0x00000000 (because Microblaze boots from here)
   (b) High Address : depending on the amount of block RAMs (check the FPGA datasheet) this should be 0x00000FFF (4kB), 0x00001FFF (8kB), 0x00003FFF (16kB), 0x00007FFF (32kB), 0x0000FFFF (64 kB), 0x0001FFFF (128kB), 0x0003FFFF (256kB), 0x0007FFFF (512kB) or 0x000FFFFF (1MB). No other values are allowed because of limitations to address-bus logic.

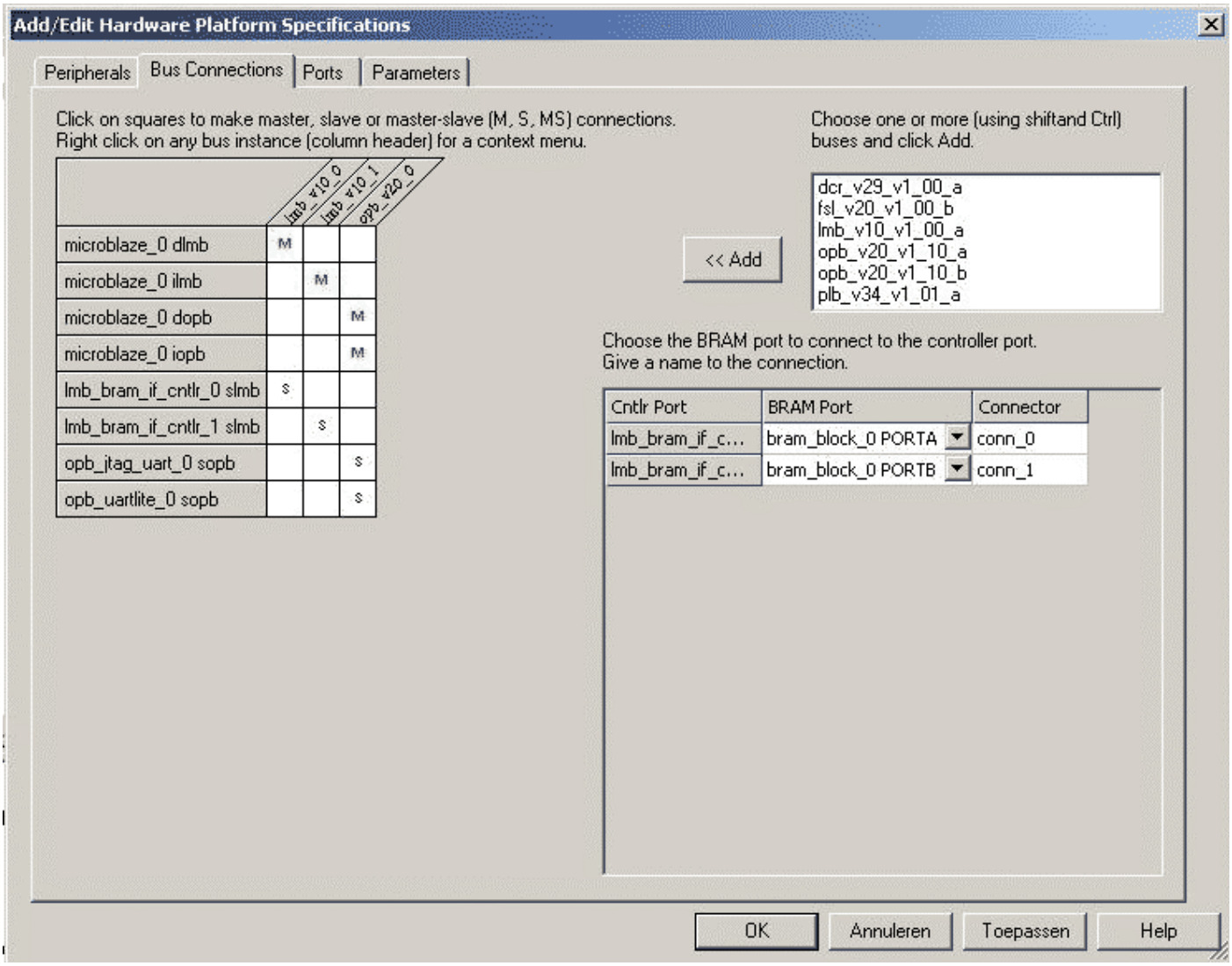
(2) For all '*opb*' peripherals choose:
   (a) Base Address : most significant byte(s) different than the *'lmb'*-bus (i.e. FFFF) least significant bytes 00.
   (b) High Address : Base Address + FF

Figure 2: Add/Edit Hardware Platform Specification: Peripherals Window



d) Connect the Busses at the '*Bus Connections*' tab (Figure 3):
   i) On the right side add the following busses:
      (1) two '*lmb_v10_v1_00_a*' busses
      (2) one '*opb_v20_v1_10_b*' bus
   ii) now make the connections by clicking on the squares on the left:
      (1) '*microblaze_0 dlmb*' on one lmb bus
      (2) '*microblaze_0 ilmb*' on the other lmb bus
      (3) one '*lmb_bram_if_cntlr*' for each lmb bus
      (4) both dopb and iopb interfaces on the opb bus
      (5) all the opb peripherals on the opb bus

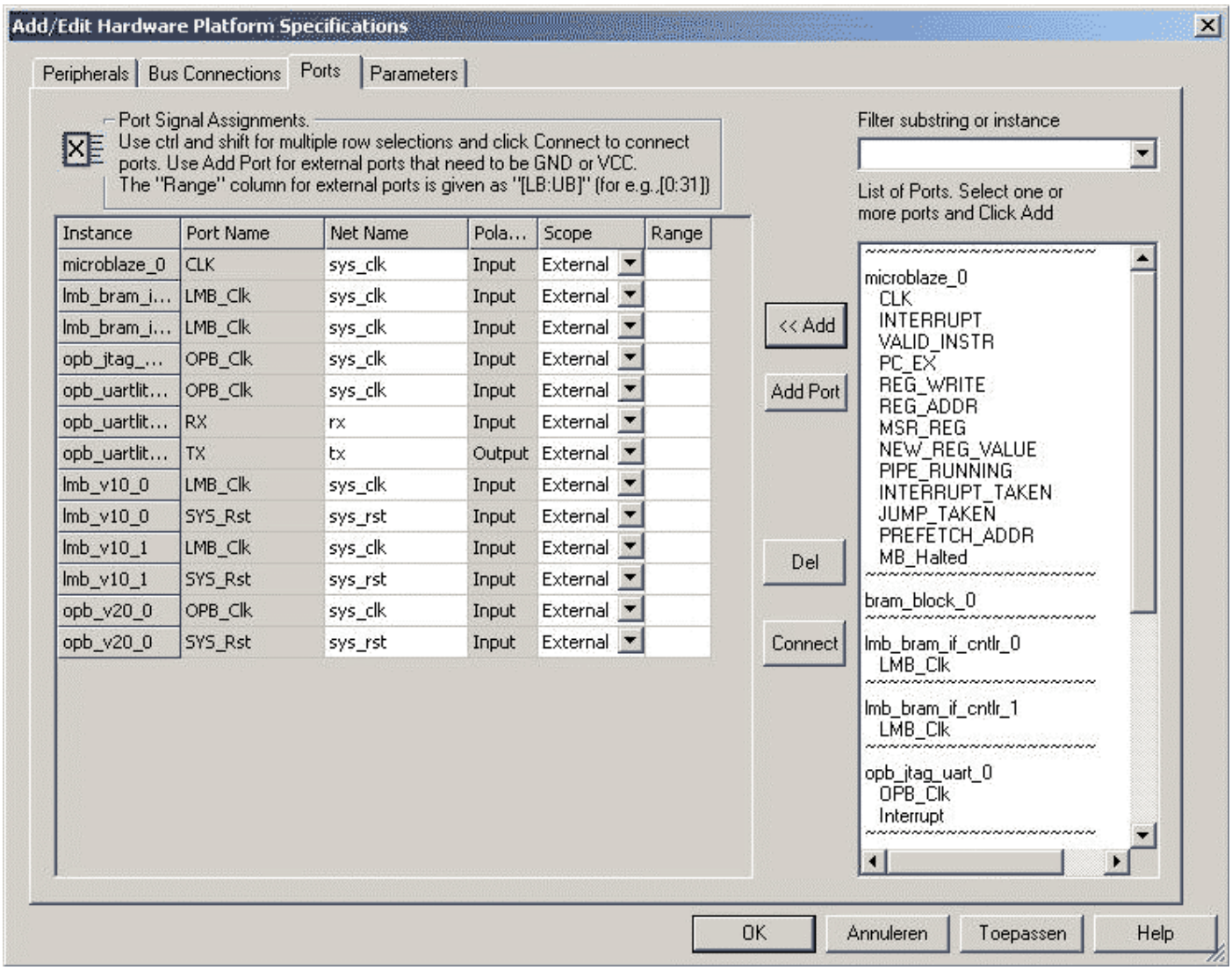
Figure 3 Add/Edit Hardware Platform Specifications: Bus Connections Window



---

e) Configure the embedded system ports at the '*Ports'* tab (Figure 4)
   i) Add all the clock signals: select all the CLK, LMB_Clk and OPB_Clk ports on your right and press *'<< Add'*. If necessary select all these ports in the list on your left by holding the shift key and press 'Connect' to change their 'net name' specified in your .ucf-file.
   ii) To add the reset signals: select all SYS_Rst ports and press *'<< Add'*. You should select these ports again and press 'Connect' to correct their 'net name' to the reset name in your ucf-file.
   iii) If you added the gpio core, you can add the port. If you connect a bus specify the 'Range' given as *'[LSB:MSB]'*.
   iv) Finally add the '*RX'* and '*TX'* ports of the uartlite. It's best to change the default netnames into the correct ones.

Figure 4: Add/Edit Hardware Platform Specifications : Ports Window

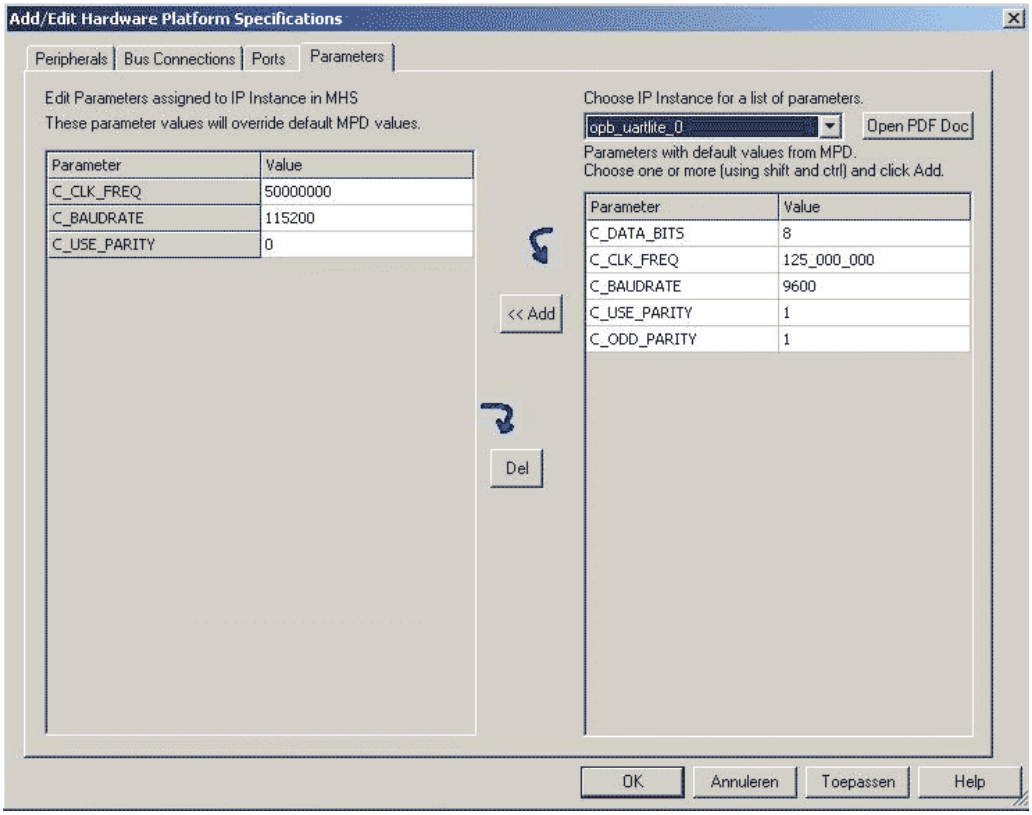


---

f) Last but not least, you have to adjust the parameters of the uartlite at the '*Parameters'* tab. Choose the '*opb_uartlite_0'* IP instance on your right. Then add the '*c_clk_freq'*, *'c_baudrate'* and *'c_use_parity'* parameters. Correct these figures on your left.

EXTRA: usually reset-signals are active low. By default XPS assumes reset signals to be active high. Therefor it is absolutely necessary to change the '*c_ext_reset_high'* parameter into '0' of all busses.

Figure 5: Add/Edit Hardware Platform Specifications : Ports Window

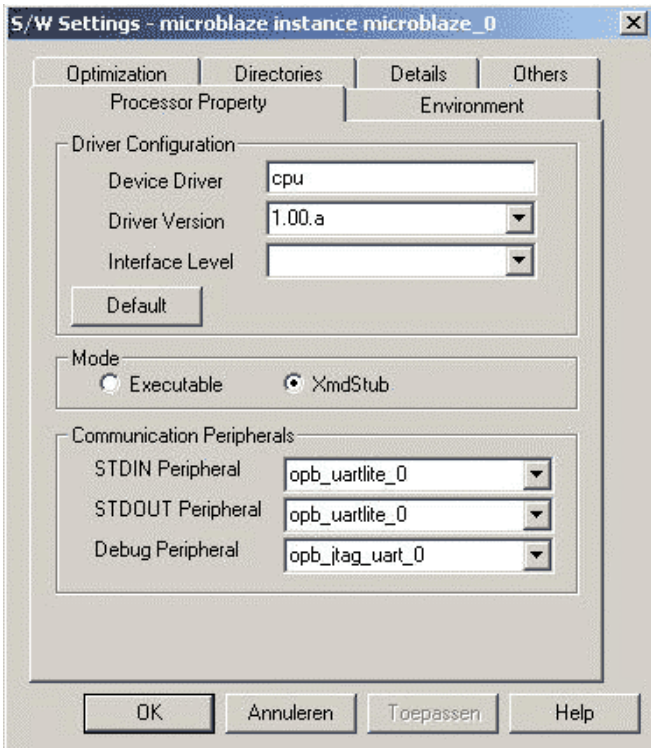


---

g) You can finalize the configuration of your microblaze embedded system by clicking 'OK'.

4) Now you have to configure the software settings by clicking with your right-mouse button on '*microblaze_0'* of 'System Tab' and then select *'S/W Settings'*. A window like fig. 6 will pop-up.
   a) At the *'Processor Property'* tab you should change the default *'Executable'* mode to *'XmdStub'* mode. This is the easiest way to run and debug your first Microblaze system.
   b) You should also change the *'STDIN Peripheral'* and *'STDOUT Peripheral'* to *'opb_uartlite_0'*. Set the *'Debug Peripheral'* to *'opb_jtag_uart_0'*.
   c) All the other variables are good by default. You can optionally change the *'Debug Options'* at the *'Optimization'* tab to *'Create symbols for assembly'*. Hereby the compiler adds the origninal c-code to the executable so you can see the c and not the assembler-code in GDB.

Figure 6: S/W Settings window



5) Now it's time to write a simple hello world program in 'c'.

```
#include <stdio.h>

main()
{
   int i;
   print("Hello World\n\r");       // send "Hello World" to the PC
   putnum(i);                     // send the integer to the PC
}
```

Create a sub-directory (i.e.: code) in your project-directory and save the file above as a c-file.

Add the file by clicking with your right-mouse button on *'Source'* (microblaze_0) and then click *'Add Files'*.

EXTRA: If you added the GPIO core, you can use the *'xgpio.h'* library to facilate writing to and reading from your GPIO ports. By default XPS doesn't generate the xgpio library, to enble this you have to disable the low-level driver by right-clicking on opb_gpio_0, choosing *'S/W Settings'* and then changing the *'Interface Level'* to 1.

```
#include "xgpio.h"
#include “xparameters.h”

main();
{
Xgpio var; //instanciation
…
Xgpio_Initialize(&var,XPAR_OPB_GPIO_0_DEVICE_ID); //initialisation
XGpio_SetDataDirection(&var, 0x00000000); //set as output
…
XGpio_DiscreteWrite(&var, waarde);
}
```

6) You can now generate the netlist by selecting ‘Generate Netlist’ in the ‘Tools’ menu.

7) While Platgen and XST are generating the netlist, you can create your **system.ucf** (pin file). You must place it in the ‘*data’* directory. Basically you need the following signals in your ucf-file for the embedded system of this tutorial:

```
NET “sys_clk” LOC = “H16”;
NET “sys_rst” LOC = “AA27”;
NET “rx” LOC = “U28”;
NET “tx” LOC = “T27”;
```

To use **busses** in the ucf-file, just use <0> at the end of the bus-name. Like this example

```
NET “gpio<0>” LOC = “H16”;
NET “gpio<1>” LOC = “AA27”;
```

Do not forget to add constraints for the clock. Otherwise the system might not be optimized for the clock-frequency of your board.

```
NET "sys_clk" NODELAY;
NET "sys_clk" TNM_NET = "clk50";
TIMESPEC "TS_clk50" = PERIOD "clk50" 20 ns HIGH 50 %;
```

It **might** also be useful and sometimes even necessary to define the ‘iostandard’ of all input-output-blocks. For example:

```
NET “gpio<0>” IOSTANDARD = LVCMOS18;
#low voltage digital controlled impedance:
NET “gpio<1>” IOSTANDARD = LVDCI_33;
NET “sys_clk” IOSTANDARD = LVDCI_18;
NET “sys_rst” IOSTANDARD = LVTTL;
```

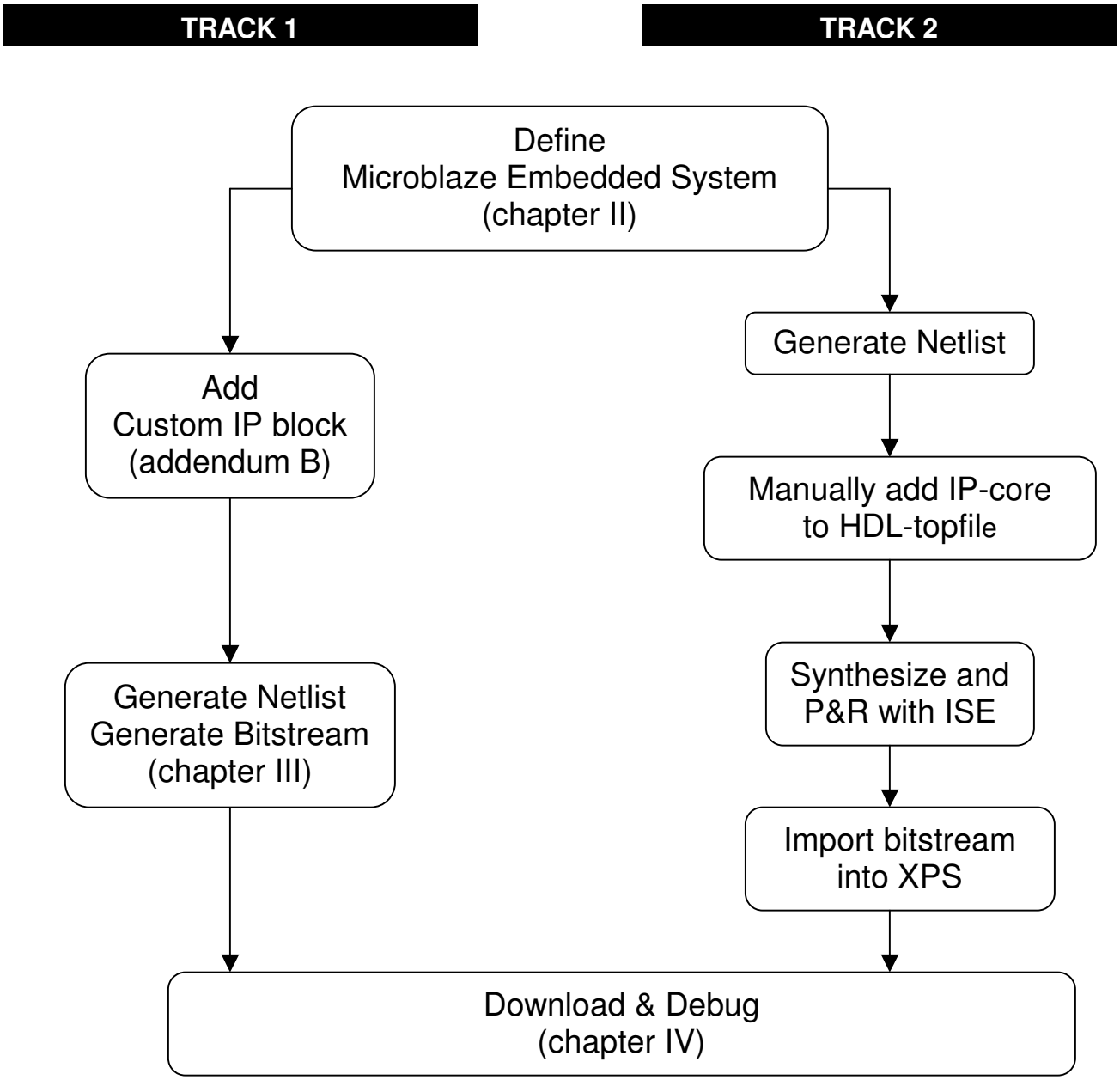
HINT: do not forget to name the file ‘*system.ucf’* and to put it in the *‘data’* sub-directory of the project-directory.

III Completing the Design

From now on, you have two possible tracks you can follow to finish your first programmable embedded design:

TRACK 1: completing the design with Xilinx Platform Studio:
Easiest way to create basic designs that contain clock dividers or other custom IP-cores or when you don't drive any additional ports on your FPGA. More complex designs are possible, but are more inconvient to implement.

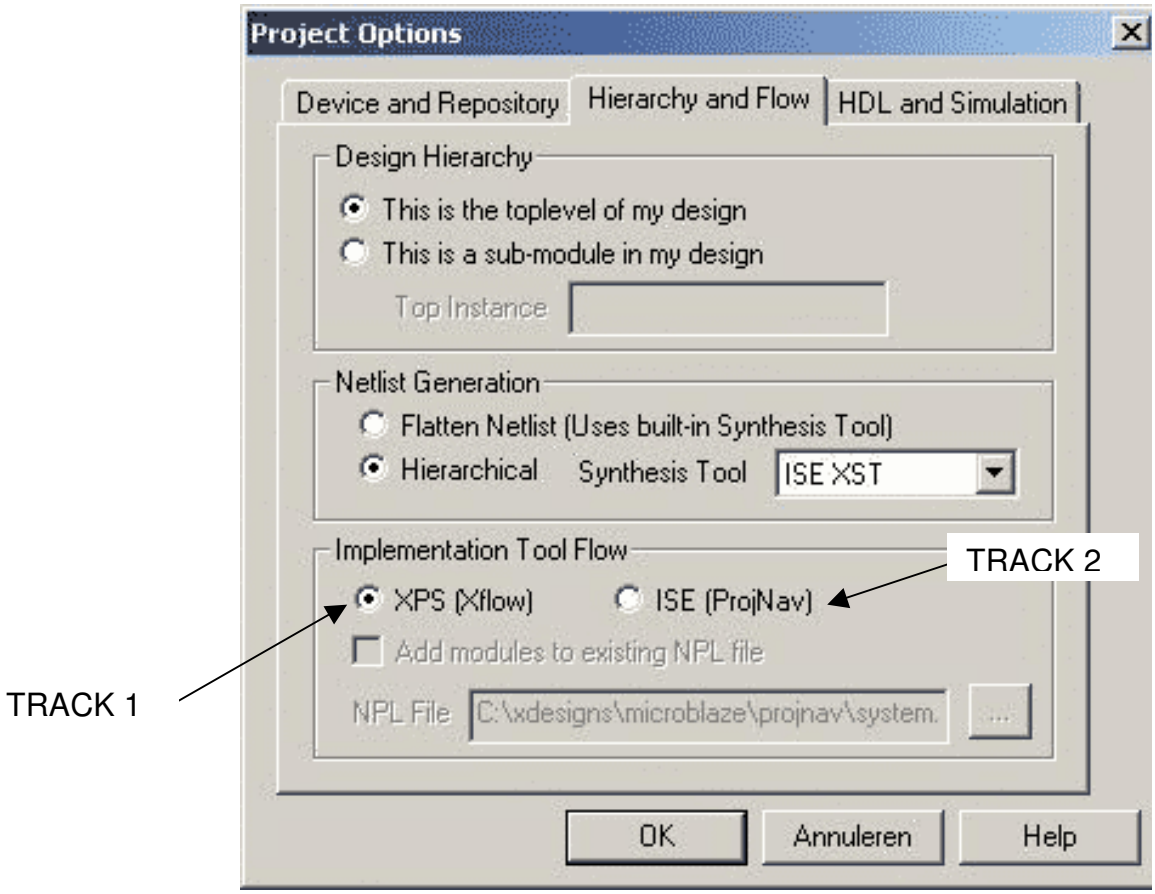
TRACK 2: design in Xilinx Platform Studio, synthesis and P&R in Xilinx ISE:
This track is the easiest track when your design contains additional static ports or when your design is more complex (clock-dividers, own IP-cores, ...). By modifying the generated toplevel-HDL-code itself, you get complete control over the design of your embedded system.



1) By default you can generate a bitstream of your embedded system in XPS by clicking on *'Generate Bitstream'* in the *'Tools'* menu.

   HINT: If for some reason this item is disabled, you can enable it back by clicking *'Project Options'* in the 'Options' menu. At the *'Hierarchy and Flow'* tab you can change the *'Implementation Tool Flow'* back to *'XPS (Xflow)'* as figure 7 shows.

Figure 7: Project Options Window



2) After XPS created the bitstream, you can compile and add the XMDSTUB (for connection with the Microblaze) to the bitstream by clicking *'Update Bitstream'* in the *'Tools'* menu.

You have now finished building your embedded system in XPS and can proceed to chapter IV

1) Change the default *'Implementation Tool Flow'* of XPS by clicking *'Project Options'* in the *'Options'* menu. At the tab *'Hierarchy and Flow'* you can enable the *'ISE (ProjNav)'* flow, like on Figure 6.

2) Export the XPS project to ISE by clicking *'Export to ProjNav'* in the *'Tools'* menu.

3) After XPS created an ISE-project, you can start the Xilinx *'Project Navigator'* in '*Start'* → *'programs'* → *'Xilinx ISE 5'*

4) Open the project XPS created by '*File'* → *'open project'* Then select *'system'* which is located in the sub-directory *'projnav'* of your project-directory.

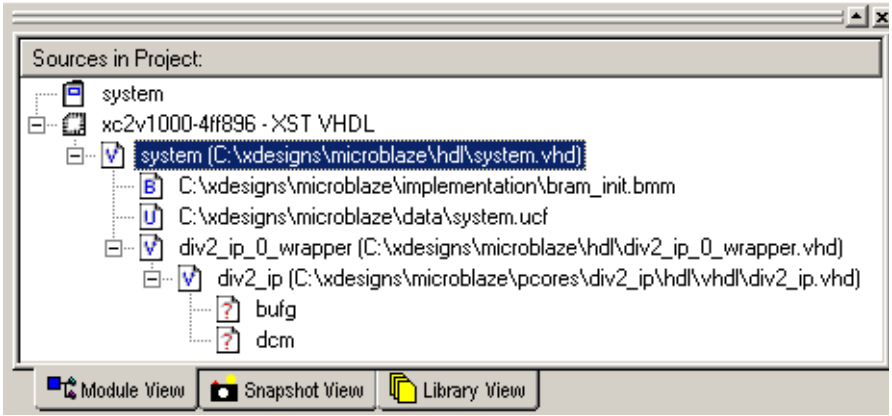
5) You can now make changes to the system topfile *'system.vhd'*. This file is located in the *'hdl'* sub-directory. The easiest way to make changes is to alter the *'system.vhd'* file.

   HINT: It is recommended to backup the original *'system.vhd'* and the final *'system.vhd'.* The reason why is that the commonly used *'clean all'* command in XPS erases all files and directories generated by XPS and that includes the *'hdl'* directory.

6) Before you generate the bitstream in ISE, make sure the project includes the ucf-file and the bram_init.bmm (A BMM file is a Block RAM Memory Map file in ASCII format and describes the organization of Block RAM memory.) file. By default you only have to add the ucf-file. Add this file by right-clicking the *'Sources in Project'* window in the upper-left corner and then choose *'Add Source'*. You can now select the ucf-file and press *'Open'*.

7) To generate the bitstream, click on the topfile *'system'* in the *'Sources in Project'* window (fig. 8). Then double-click *'Generate Programming File'* in the *'Processes for Current Source'* window.

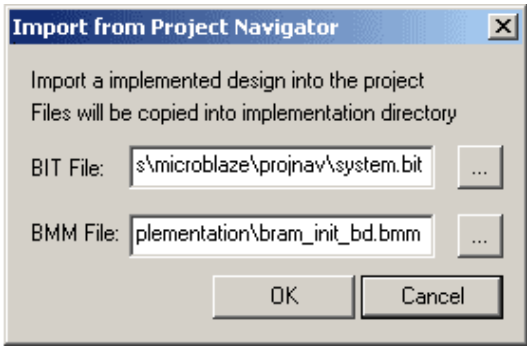
Figure 8: Sources in Project Window



---

8) After the bitstream has been succesfully generated, switch back to XPS and import the bitstream by clicking *'Import from Project Navigator'* in the *'Tools'* menu (figure 9). By default the location of the BIT file is good (\projnav\system.bit). You only have to select the correct BMM file which is located in the *'implementation'* subdirectory and is called *'bram_init_**bd**.bmm'*.

Figure 9: Import from Project Navigator Window



9) You now have to update the bitstream to include XMDSTUB program. Just click *'Update Bitstream'* to compile and include it.

IV Download and debug (GENERAL)

1) First make sure you correctly connected the downloadcable (parallel cable IV, …).

2) Start iMPACT *Start → programs → Xilinx ISE 5 → Accessories → iMPACT*

3) Now the connection wizard opens the *'Operation Mode* Selection' window. By default the "*configure devices*" is good, so press *'Next'*.

4) Configure device with "*Boundary-Scan Mode*" is OK, so press *'Next'*.

5) *'Automatically connect to cable and identify Boundar-Scan chain' is good again*, so press *'Next'*.

6) If all goes well, iMPACT identifies all devices on your board and asks to assign the configuration file. When the target device is selected, assign the *'download.bit'* file which is located in the *'implementation'* sub-directory. All other devices of your board don't need to be reprogrammed, so select *'cancel'.*

7) Finally right-click the target FPGA in the Boundary Scan chain and select *'Program...'* and then click on *'OK'.* IMPACT will now download the bitstream to the FPGA.
HINT: When you finished downloading, close iMPACT because it interferes with the XMDSTUB connection.

8) Next thing to do is to compile the '*Hello World'* program. Because of the XMDSTUB mode that was selected in chapter II, only the connection software (XMDSTUB) is compiled and downloaded together with the bitstream. This way, you can easily change the software without completely generating a new bitstream file. To compile your program, click *'Compile Program Sources'* in the *'Tools'* menu. After compilation you can see how much your program takes down below in the console window. Make sure it doesn't exceed the amount of memory you have in your embedded system.

9) Now open *'XMD'* in the *'Tools'* menu. XPS will open a Xygwin command window. To make connection with the Microblaze XMDSTUB, type:

```
Xilinx Microprocessor Debug (XMD) Engine
Xilinx EDK 3.2.1 Build EDK_Cm.19
Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.
XMD% mbconnect stub -comm jtag
```

It is possible you must also specify the position of the FPGA *(i.e. –posit 2)*
At the end there should be the following message:

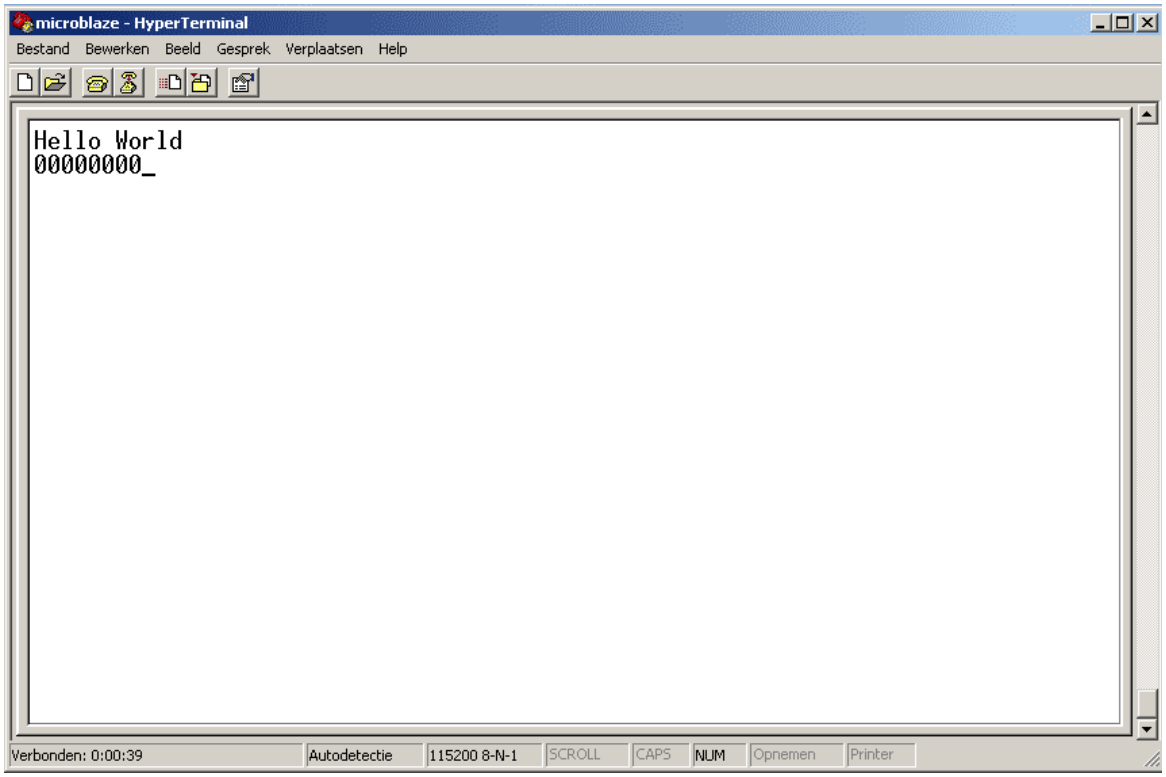
```
Connecting to XMD stub..
XMD communication stub initialized. Version No: 2
Communicating with XMD stub on target board
Connected to MicroBlaze "stub" target. id = 0
Starting GDB server for "stub" target (id = 0) at TCP port no 1234
```

Don't close this window until you've finished communication with the microblaze.

10) Open a terminal and set the correct parameters (115kbaud, no parity, no hardware flow). Also make sure you connected your serial port with the board.

11) In XPS, open the *'Software Debugger'* in the *'Tools'* menu. A GDB window will appear. In You can now run the hello world program by selecting *'Run'* in the *'Run'* menu. Make sure *'Target'* is *'Remote/TCP : XMD'*, '*hostname'* is '*localhost'* and '*port'* is '*1234'*. After a few seconds GDB has made connection and will execute your program. The terminal should display 'Hello World' and '00000000' like in figure 10. If you want you can add breakpoints and step the program.

Figure 9: Hello World terminal



V Solutions for Problems and Errors

We have experienced quite a few difficulties when building our first Microblaze embedded system. A few things we encountered and you may want to check:

- Has the uartlite the correct *baudrate* and *clockspeed*-parameters? (don't use _ in the clockspeed parameter)
- You're system might be permanently resetting, so check if you have correctly set the *'c_active_high_reset'* parameter of the lmb and opb bus.
- Is your clock too fast for your system? Make sure you've added constraints to your ucf file as described in chapter III.
- Is your memory too small for your program? Make sure you've used the maximum of blockRAMs possible. Try to find some code you can leave out. Always start with a simple *'xil_printf("Hello World\n\r");'*.

For more troubleshooting go to http://support.xillinx.com or to the Xilinx embedded processor forum: http://toolbox.xilinx.com/cgi-bin/forum?14@@/Embedded%20Processors.

ADDENDUM

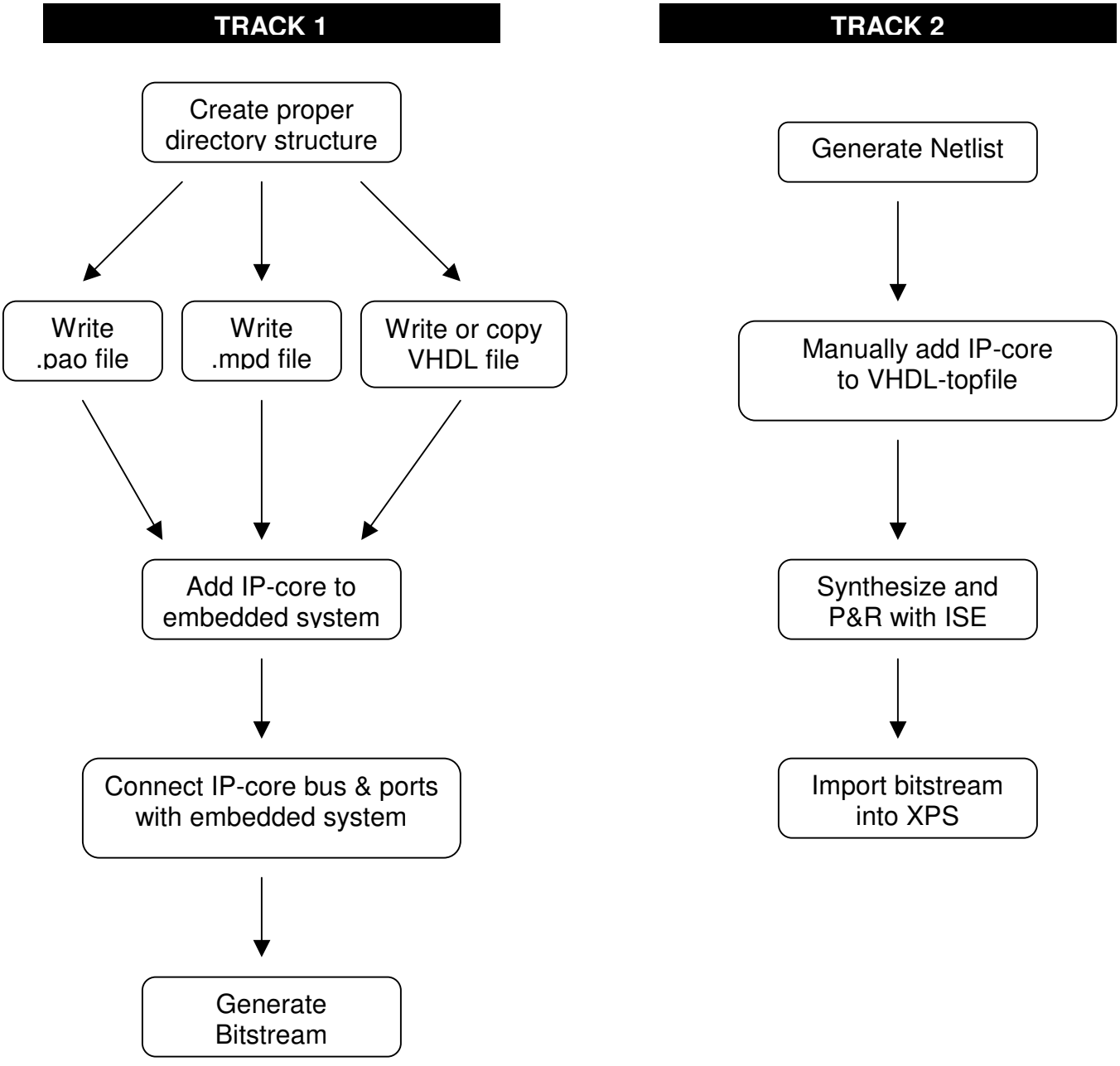
A. General description how to add custom IP-cores

You have two possibilities to add your own IP-cores to a Microblaze embedded system:

- Create your own XPS compatible IP-core
- Manually add your IP-core to the toplevel VHDL-code.

The flows are explained by the flowcharts below. As you may notice track 2 has a less complicated flow. Therefor it is recommended to use track 2.

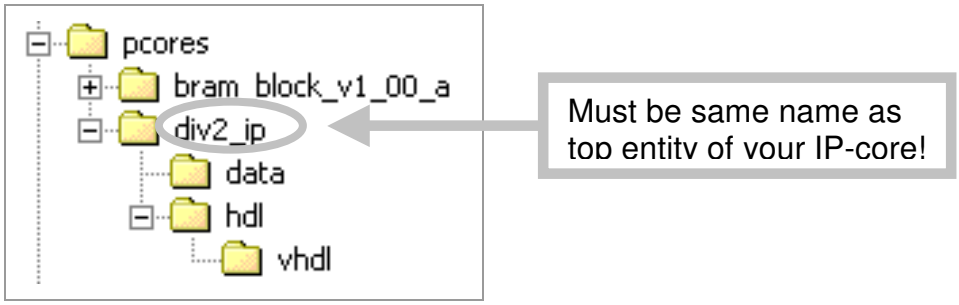
Both tracks are illustrated in a clockdivider example in chapter B (TRACK 1) and C (TRACK 2). They reason why we used clockdividers is because they are commonly used in programmable embedded systems to alter the development board clockfrequencies.



B. *Adding the clockdivider to the design for Xilinx Platform Studio (TRACK 1)*

1) First you will need to create the proper directory structure. You should create a new sub-directory in the sub-directory *'pcores'* of the project-directory. Figure 1 gives an example of the directory-structure you must create.

Figure a: proper directory structure



---

2) Create a new file in the '*data'* directory:
   a) *'div2_ip_v2_0_0.pao'*. It is essential that the name ends on v2_0_0. Otherwise the file won't be recognized. This file contains all the libraries that the wrapper should use. In our simple example we don't really need this feature, but without a file XPS will not work. Just put the following line in the file:

```
lib div2_ip div2_ip
```

   b) Create a '*div2_ip_v2_0_0.mpd'* file. This file describes the ip-core. In this example we only describe a simple clockdivider. So the file should look like:

```
BEGIN div2_ip, IPTYPE=IP, HDL=VHDL

PORT sys_clk = "", DIR=in
PORT div2_out = "", DIR=out
END
```

3) The biggest part is to write a new VHDL-file. Of course you may import your own clockdivider IP-core, but if you don't have one, you can use this example. There are several ways to write a clockdivider. In this example we modify a DCM-module. The example is a clockdivider which divides the clock in 2. To change the rate, simply change the '*clkdv_divide'* parameter. Possible values are: 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0

```
library ieee;
use ieee.std_logic_1164.all;

entity div2_ip is
  port (
          sys_clk   : in std_logic;
          div2_out  : out std_logic);
end entity div2_ip;

architecture div2_ip_arch of div2_ip is

component DCM
-- synthesis translate_off
generic (CLK_FEEDBACK :string := "1X";
          CLKDV_DIVIDE : real := 2.0;
          CLKFX_DIVIDE : integer :=1;
          CLKFX_MULTIPLY : integer := 1;
          CLKIN_DIVIDE_BY_2 : boolean := FALSE;
          CLKOUT_PHASE_SHIFT: string := "NONE";
          DESKEW_ADJUST: string := "SYSTEM_SYNCHRONOUS";
          DFS_FREQUENCY_MODE: string := "LOW";
          DLL_FREQUENCY_MODE: string := "LOW";
          DSS_MODE: string := "NONE";
          DUTY_CYCLE_CORRECTION : Boolean := TRUE;
          FACTORY_JF : bit_vector := X"C080";
          PHASE_SHIFT: real := 0;
          STARTUP_WAIT :boolean := FALSE);
-- synthesis translate_on
port ( CLKIN     : in std_logic;
    CLKFB      : in std_logic;
    DSSEN      : in std_logic;
    PSINCDEC   : in std_logic;
    PSEN       : in std_logic;
    PSCLK      : in std_logic;
    RST        : in std_logic;
    CLK0       : out std_logic;
    CLK90      : out std_logic;
    CLK180     : out std_logic;
    CLK270     : out std_logic;
    CLK2X      : out std_logic;
    CLK2X180   : out std_logic;
    CLKDV      : out std_logic;
    CLKFX      : out std_logic;
    CLKFX180   : out std_logic;
    LOCKED     : out std_logic;
    PSDONE     : out std_logic;
    STATUS     : out std_logic_vector(7 downto 0)
  );
end component DCM;
```

```
  component BUFG
      port (
       O : out std_logic;
       I : in std_logic
      );
  end component;

  signal div2_i : std_logic;
  signal div2_int : std_logic;
  signal clk0_i : std_logic;
  signal clk0_int : std_logic;
begin
  div2_inst: DCM
     port map(
               CLKIN          => sys_clk,
          CLKFB          => clk0_i,
          DSSEN          => '0',
          PSINCDEC       => '0',
          PSEN           => '0',
          PSCLK          => '0',
          RST            => '0',
          CLK0           => clk0_int,
          CLK90          => open,
          CLK180         => open,
          CLK270         => open,
          CLK2X          => open,
          CLK2X180       => open,
          CLKDV          => div2_int,
          CLKFX          => open,
          CLKFX180       => open,
          LOCKED         => open,
          PSDONE         => open,
          STATUS         => open
          );

  BUFG_I : BUFG
  port map (
               O => clk0_i,
               I => clk0_int
          );

  BUFG_N : BUFG
  port map (
               O => div2_i,
               I => div2_int
          );

  div2_out <= div2_i;

end architecture div2_ip_arch;
```

4) Open XPS, open the project, right-click on *'System BSP'* and select *'Add/Edit Cores...(dialog)'*. There you will notice that the '*div2_ip'* core has been added to the list of IP cores. So now you only have to add it and connect the ports at the *'ports'* tab.

   HINT: Make sure you adjusted the *'c_clk_freq'* parameter of the uartlite at the 'parameter' tab to the new frequency.

C. Manually adding the clockdivider to the code for synthesis & P&R in ISE (TRACK 2)

1) To begin, you must create or copy a clockdivider VHDL file. You can use a clockdivider you have used for other projects or you can use the clockdivider which is used in chapter B. You can either copy this VHDL file in the *'hdl'* subdirectory or add the file to the ISE project.

2) Because we need to manually add the clockdivider to the '*system.vhd'* topfile, we need to first generate the topfile. Therefor complete chapter II of the tutorial.
   In this example we will add the clockdivider of chapter B of this addendum.
   a) First you have to define the clock divider ip module. To do this, add the following lines right after *'ARCHITECTURE IMP OF system IS':*

```
component div2_ip is
port (
          sys_clk   : in std_logic;
          div2_out  : out std_logic);
end component;
```

   b) Next thing to do is disabling the buffer for the incoming clock signal. Normally this instantiation is located at the end of the file. Just search for the incoming clock signal name and disable the module by erasing the code or put some – before the lines. Remember the output signal name of the buffer module. (<clock-signal-name>_BUFGP)

   c) Finally add the clockdivider module (for convenience, at the end of the file). It should look like:

```
clkdiv : div2_ip
port map (
          sys_clk   => sys_clk,
          div2_out  => sys_clk_BUFGP);
```

To proceed, just follow chapter III and IV of the tutorial.