DANISH
TECHNOLOGICAL
INSTITUTE

Building Linux 2.6 on ML40x boards

by Senior Consultant Carsten Siggaard

1 Table of Contents

2 Introduction/Preface

The need for some kind of operating system (OS) is defined in many designs by the flexibility provided by the OS. The initial purpose of an operating system is to have something running in the background, and for many users the spooling of prints to the printer queue is for many users the most obvious reason. The purpose of using an OS on real-time systems is very much like the printer example; you wish to perform several parallel tasks but at the same time avoid the interruption of one task while the other is in progress. For one-processor systems it can only be performed virtually. The tasks are interrupted but the processor operates so quickly that the interrupted tasks are not affected by another task being serviced briefly. Another important task is the difference in transfer speed between IO units. Most likely you will have slow IO units requiring to be serviced immediately (e.g. Terminals), or you will have other IO units which are fast and can be serviced in a more relaxed manner (buffered IO or intelligent units).

Other reasons such as reducing the calculation complexity by dividing the volume of data processed do not apply to small real-time systems. There is, however, still a reason for using concurrent processing.

Any system where things have to be processed with virtual or real concurrency is a candidate for using an operating system.

You can write concurrent systems yourself without using any kind of operating system. It is, however, cumbersome, and unless you know exactly what to do, the implementation of your system is most likely to fail.

If you wish to implement another concurrent system, the reuse of obtained implementation knowledge is likely to be lost or the scheduling mechanisms do not apply to the new system being built.

There are several operating systems available and many are free of charge. However, one system has gained momentum during recent years – and this is Linux. Linux is a GNU-licensed (GPL) operating system which can be downloaded and used without having to pay a licence fee. It is not a FREE code in the sense that you own the code once you have downloaded it. It is free in the sense that you can use it provided that you accept the terms of the GNU licenses and of course obey the license restrictions of any other software used.

One of the advantages of Linux is its momentum and widespread use, not only for real-time systems but also in servers and desktops. Linux is portable between many systems, for example Intel and MIPS and of course PowerPC.

The vast volume of Linux supporters in the community is another advantage. The use of Linux in a real-time system will not prevent you from encountering problems but it will ensure that you can post a help request to commercial and non-commercial supporters, mailing lists or consultancy services.

However, the use of Linux on a FPGA is not trivial, especially if you decide to use a standard kernel. Although there are lots of drivers for Xilinx Peripherals, some work has to be carried out. The additions to the default handling of kernels are the topic of this paper. It is intended as a service

for those who wish to compile a standard kernel without having to face problems that have already been solved.

This document is a rewrite of the document entitled “Building Linux on ML40x boards” which covers implementation of Linux 2.4, this document covers Linux 2.6. This implies that the sections covering the Hardware configuration and Cross-Compilation are identical, but the sections covering Building the Kernel and generating board support packages are different.

3 Advice for the Reader

Some of terms used in this document will be explained below to ensure that the reader is familiar with the terms.

Linux System
When using the terms “Linux System” or ”Accessible from Linux” in this document, we are referring to a Linux platform on a PC, a UNIX server or even a windows server operating on Cygwin. We will deprecate the latter approach due to fact that this has not been thoroughly tested at our site. The term Linux System should never be confused with the terms “Embedded Linux System” which refers to the embedded Linux System that you wish to build on a Xilinx board.

Typing
Whenever the user is expected to do any typing, the **`boldface courier`** style is used. Expected output from the Embedded Linux System (or the Linux System) is written in `normal courier` style.

Generic Filenames
In some cases the file names may contain generic information such as `glibc-<glibc-version>`. This could be an abbreviation for `glibc-2.3.6` or any other version of the glibc library. We use these generic filenames if the version is a result of a choice the reader has to make – in this case we cannot predict what the precise filename will be.

Errors/Problems
We have invested a lot of effort to make sure that the reader will be able to cope with problems which may occur during the port. We have not tried to solve all the problems at the vendor or various software providers on the net. One exception is with regard to some problems with the Xilinx uartlite driver which we have reported to Xilinx Inc.

4 C-Cross Compilation

In order to be able to port the kernel, you need a compiler capable of generating the image of the Linux Kernel. If the compiler has to generate an image which runs on an architecture that is different from the architecture it runs on itself, the compiler is called a “cross-compiler”.

The following terms should be noted:

Build	The machine architecture on which an application (in this case the compiler) is built.
Host	The machine architecture on which an application shall run.
Target	This term is used for compilers. It defines the architecture whereto the compiler shall generate object files and executables.

Whenever working with GNU applications (including compilers), the above terms will apply. However, differences may occur.

In this document we will describe a system where the build architecture is identical to the host architecture, namely an i686 Linux system where the target is the 405 PowerPC architecture. Below we will outline a base tool chain using the standard GNU setup but most emphasis will be placed on a nice little tool called “cross-tool”. We recommend that you read both sections because “cross-tool” is actually a wrap-around of the gnu tool chain. If you understand the GNU tool chain, you will also understand the “cross-tool”.

4.1 Cross Compilation using the GNU tool chain

It is possible to generate a GNU compiler using the recipes described in [1]. It is somewhat cumbersome but by using these recipes you will manage to get through the entire process without much problem.

To build a GNU tool chain, 3 components have to interact:

1. bin-utils which contains the following components:

    - **ld** - the GNU linker.
    - **as** - the GNU assembler.
    - **addr2line** - Converts addresses into file names and line numbers.
    - **ar** - A utility for creating, modifying and extracting from archives.
    - **c++filt** - Filter to demangle encoded C++ symbols.
    - **gprof** - Displays profiling information.
    - **nlmconv** - Converts object code into an NLM.
    - **nm** - Lists symbols from object files.
    - **objcopy** - Copies and translates object files.
    - **objdump** - Displays information from object files.
    - **ranlib** - Generates an index to the contents of an archive.
    - **readelf** - Displays information from any ELF format object file.
    - **size** - Lists the section sizes of an object or archive file.
    - **strings** - Lists printable strings from files.
    - **strip** - Discards symbols.
    - **windres** - A compiler for Windows resource files.

2. GCC

    - the gnu compiler

3. glibc

    - The GNU c-library which add-ons such as pthreads and crypto facilities.

Each provides a versioning issue. The fact that you have to use gcc to compile gcc makes the generation of the cross-compiler possible but it is onerous.

Luckily there is a very simple way to generate a crosscompiler using CrossTool.

If you wish to make a cross-compiler in the usual way, a good starting point would be Karim Yaghmours book about the topic ([1]).

4.2 CrossTool

Compilation of the Cross-compiler using Crosstool is performed in Linux.

CrossTool is described at the website http://kegel.com/crosstool/. The website contains a description of the crosstool and a motivation for using it. Citation: "`Building a gcc / glibc cross-toolchain for use in embedded systems development used to be a scary prospect, requiring iron will, days if not weeks of effort, lots of Unix and Gnu lore, and sometimes willingness to take dodgy shortcuts…`"

Using CrossTool to create a toolchain takes less than an hour and therefore constitutes another motivation factor.

The CrossTool binary can be obtained from

http://kegel.com/crosstool/crosstool-0.42.tar.gz

or from

http://kegel.com/crosstool/crosstool-0.43.tar.gz

Locate the file somewhere accessible from Linux, and unpack it:
`$ tar xvzf crosstool-0.43.tar.gz`
Enter into the resulting directory structure (crosstool-0.43) :
`$ cd crosstool-0.43`

The directory is denoted the build directory.

The build process is very simple. Simply edit the following lines in `demo-powerpc-405.sh` to the following (or adapt the content to your needs).

```
TARBALLS_DIR=$HOME/downloads
RESULT_TOP=$HOME/crosstool
export TARBALLS_DIR RESULT_TOP
GCC_LANGUAGES="c,c++"
```

If you have a proxy, you have to remember to set the variable: HTTP_PROXY eg.

`$ export HTTP_PROXY=http://your.servername.com:8080`

First edit the file **`demo-powerpc-405.sh`** and make sure that the compiler you wish to use is enabled. **IMPORTANT: Never use a compiler versioned 4 or above (gcc-3.4.5-glibc-2.3.6 is recommended) !**

Hereafter run the file:

```
$ demo-powerpc-405.sh
```

During the build, there may be some download problems; e.g. the downloading of kernel headers fails. In this case, a simple solution is to download manually e.g. through Internet Explorer and to save the file in the directory whereto `TARBALLS_DIR` points (see above).

Another problem is that the auto generated file "`version-info.h`" is not generated properly. To fix this problem, do as follows (after the build has failed because of the content of this file):

- Unpack the `glibc-<glibc-versioninfo>-tar.bz2` file in the `TARBALLS_DIR` directory:
  `$ tar xvjf glibc-<glibc-versioninfo>-tar.bz2`
- Enter the resulting `glibc-<glibc-versioninfo>` directory
- Copy the "`version-info.h`" file from the build directory:
  `$ cp <build directory>/build/powerpc-405-linux-gnu/gcc-<gcc-versioninfo>-glibc-<glibc-versioninfo>/glibc-<glibc-versioninfo>/version-info.h glibc-<glibc-versioninfo>/csu/version-info2.h`
- Edit the file `glibc-<glibc-versioninfo>/csu/version.c`, and change the line `"#include "version-info.h"` to `#include "version-info2.h"`
- Re-create the archive:
  `$ tar cvjf glibc-<glibc-versioninfo>-tar.bz2 glibc-<glibc-versioninfo>`

Hereafter you can return to the build directory and restart the process (the files will not be re-downloaded):

```
$ demo-powerpc-405.sh
```

If everything goes well, you will have a crosscompiler located at the position where `$RESULT_TOP` points to.

5 Creating the HW

5.1 Different kinds of HW

The Compilation of the hardware for the ML board series is described in windows.

From Xilinx you can obtain several kinds of hardware capable of running Linux. Some examples are the ML300, ML403, ML405 and ML410 Evaluation boards. All are supported by the Xilinx Platform Studio which makes the hardware generation simple.

In this paper, we will focus on the ML405 board and also work on the ML403. The two boards do not differ much, so despite some minor problems you will be able to build Linux for both boards provided that you can build for one. Any differences between the two boards will be described later in this paper.

5.2 Building for the ML403

Building for the ML405 is much like building for the ML403, the only hardware differences are the MGT support and that the FPGA on the ML405 board is larger than the one residing in the ML403 board.

5.2.1 Creating the board support package

The board support package for the ML403 is completely identical to the package for the ML405 board. Please refer to the description on page 29.

Building for the ML405
Start the Xilinx Platform Studio and select the "Base System Builder Wizard" as depicted in Figure 1.

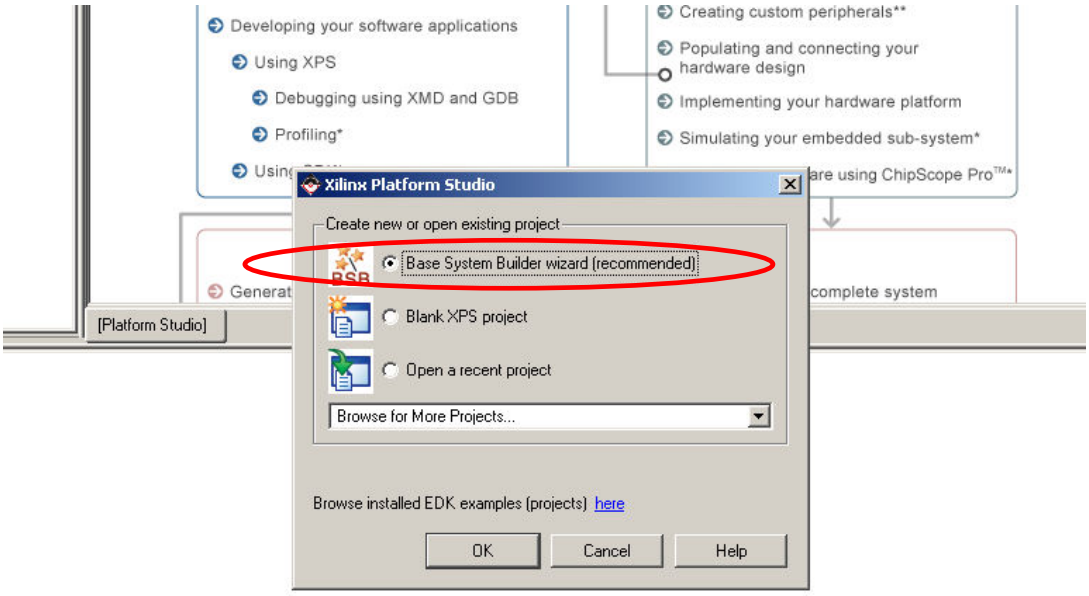

Figure 1: Xilinx Platform Studio start-up screen

In the next popup, the XPS will present a popup asking for the path of the project you wish to create.

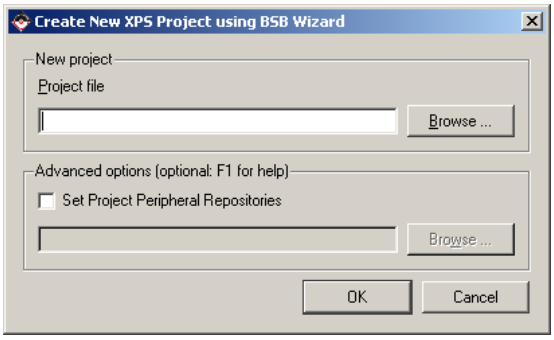

Figure 2: Choosing a new project

When creating a new project, we recommend the use of a new directory for each attempt. Otherwise the files from each of the wizard runs will be mixed and this is probably not what you want.

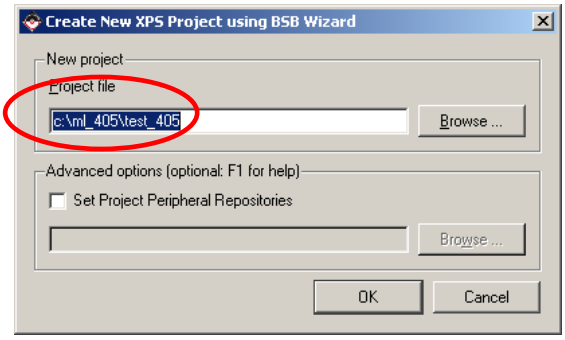

Figure 3: Choosing a new project with path entered

After you have entered the path, you may be asked whether XPS should create the directories of the path which does not exist (see Figure 4).

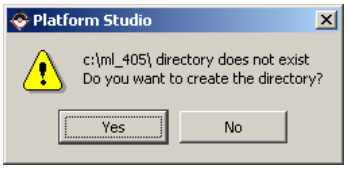

Figure 4: Platform Studio popup

Click on “Yes” to create the directories.

The next popup is the Welcome Window in which you can select whether you wish to create a new design or to load an existing design. Select "**I would like to create a new design**".

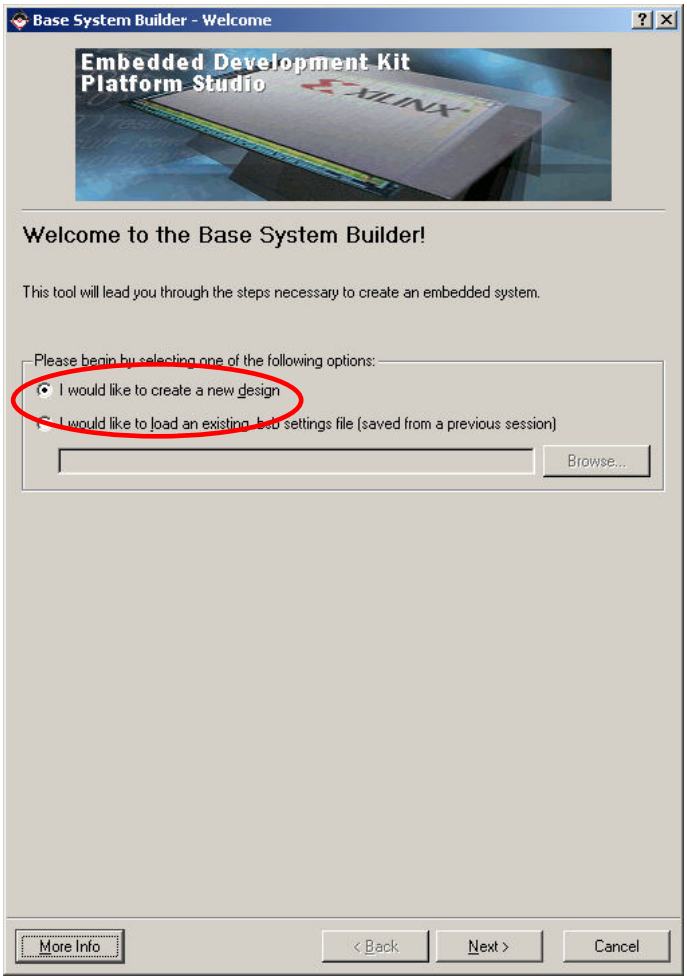

Figure 5: Base System Builder Start-up

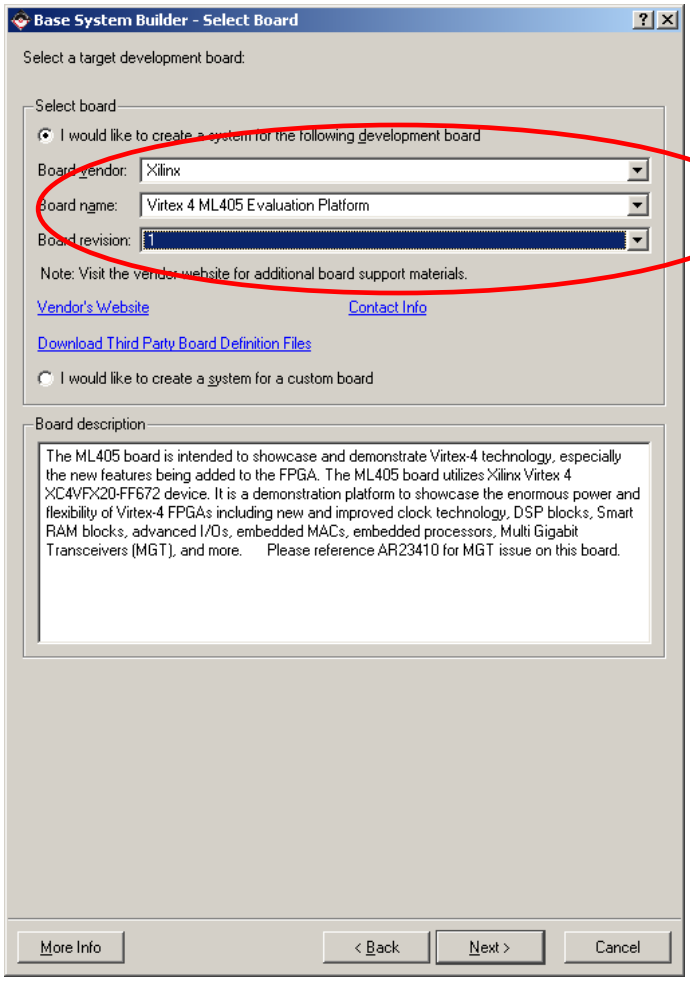

Figure 6: Select Board Window

For the ML40X board, the XPS is able to generate two different processors: MicroBlaze and PowerPC. MicroBlaze is a “soft-core” processor and the PowerPC processor is a “hard-core”. To use full-featured Linux, you have to select the PowerPC processor or a MicroBlaze processor with MMU.

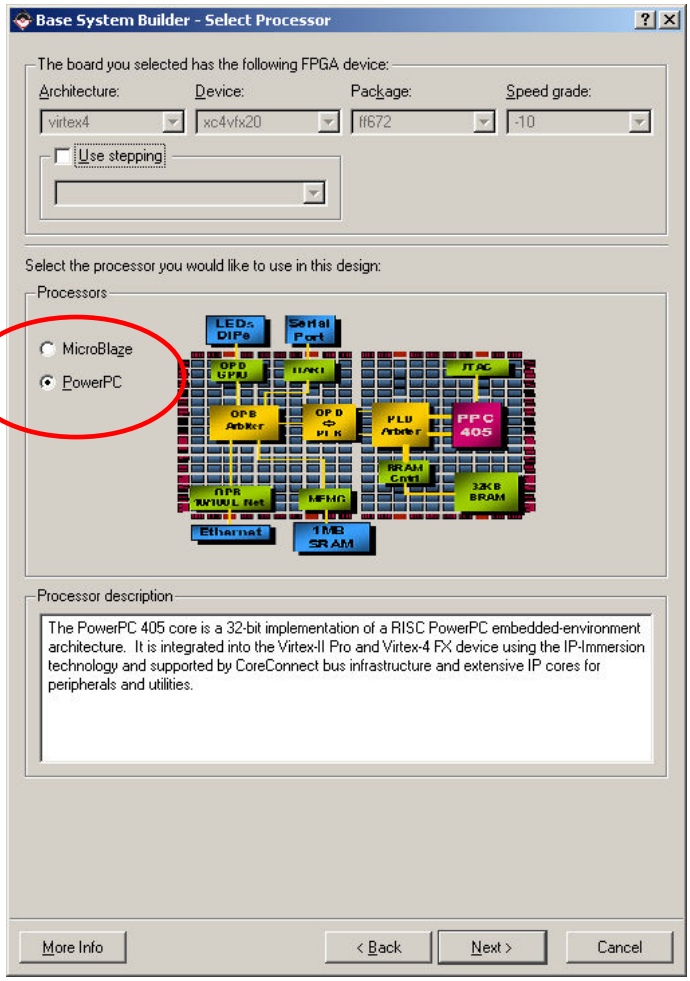

Figure 7: Select Processor Window

Once the processor is selected, it has to be configured. This is performed in the next popup window where you can enable the cache and the OCM (On-Chip-Memory). For testing purposes disable all. Later, you are encouraged to change the settings to improve the performance of your system. Please note that the OCM uses internal memory which is a limited resource.

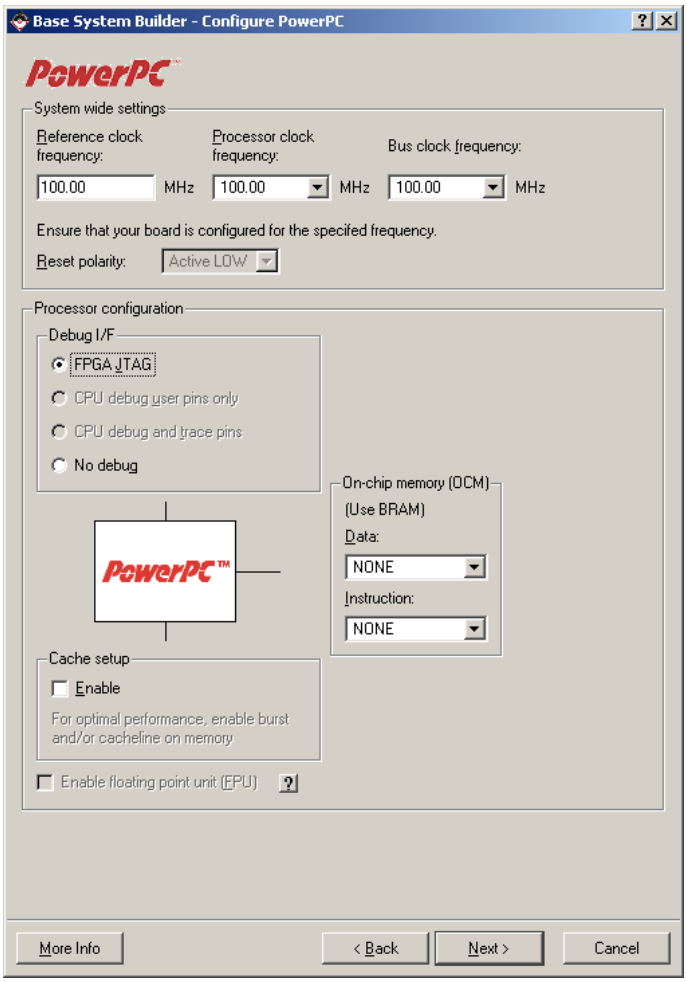

Figure 8: PowerPC Configuration Screen

Having configured the processor, you must continue configuring the IO interfaces. In most cases you have to use the default settings, and you MUST make **everything interrupt-driven**. The MGT interface is specific for the ML405 board. It is, however, not necessary to get Linux running so you can safely disable it (and leave it on if you like). Note that there is a locker adjacent to the IIC_EEPROM menu which indicates that the IIC is a commercial core, and that you have to pay a fee for using it. Furthermore the core is "time-bombed", i.e. it will cease to operate after a fixed period of time.

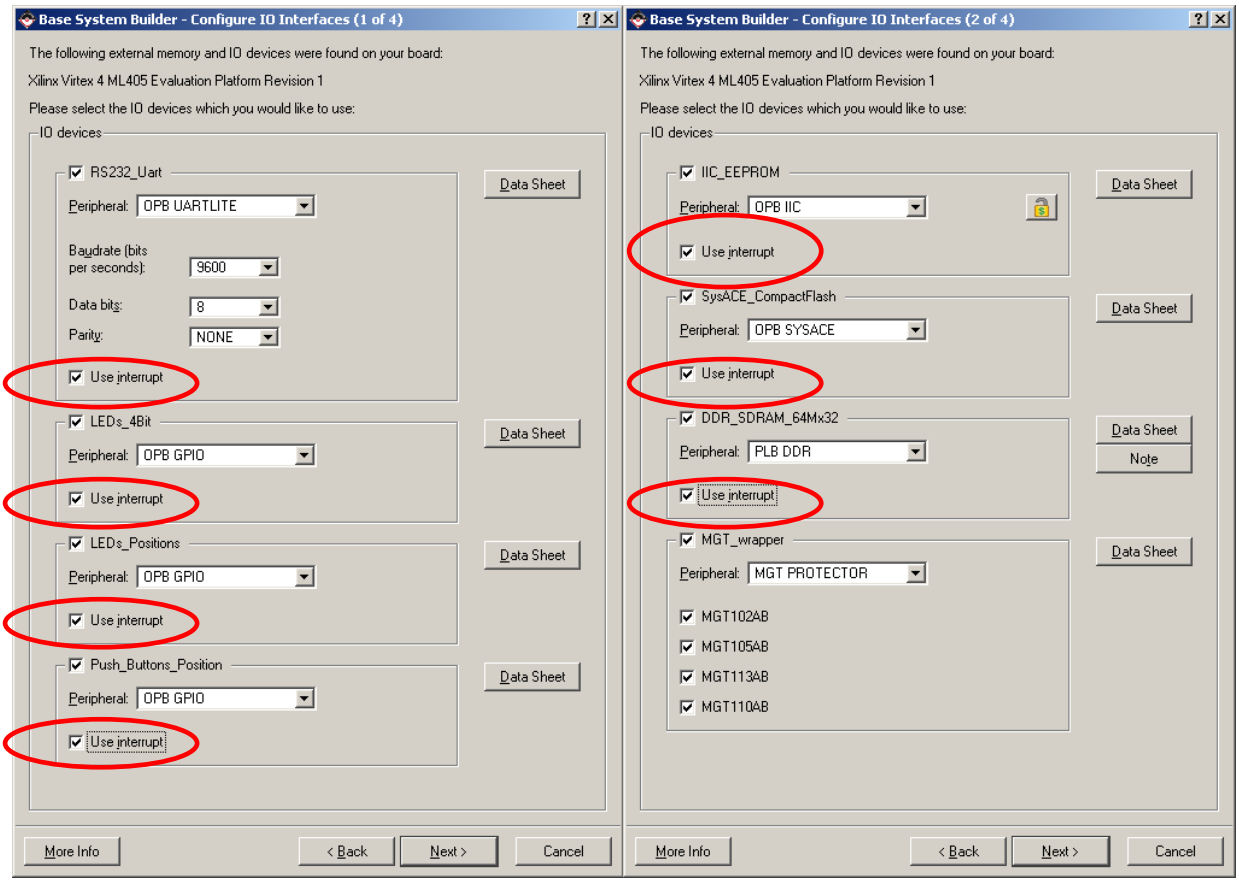

Figure 9: Configure IO Interfaces Window (1 and 2 of 4)

The last two popups regarding the IO Interfaces concern the Ethernet and the special memory configuration. Remember that the TriMode_MAC_GMII and the Ethernet MAC are mutually exclusive, that is you cannot select both. Note that the Ethernet MAC dropdown menu has a locker adjacent to it which indicates it is a commercial core, i.e. the use of it is restricted and you have to pay a fee to license it.

As usual you MUST make everything interrupt-driven.

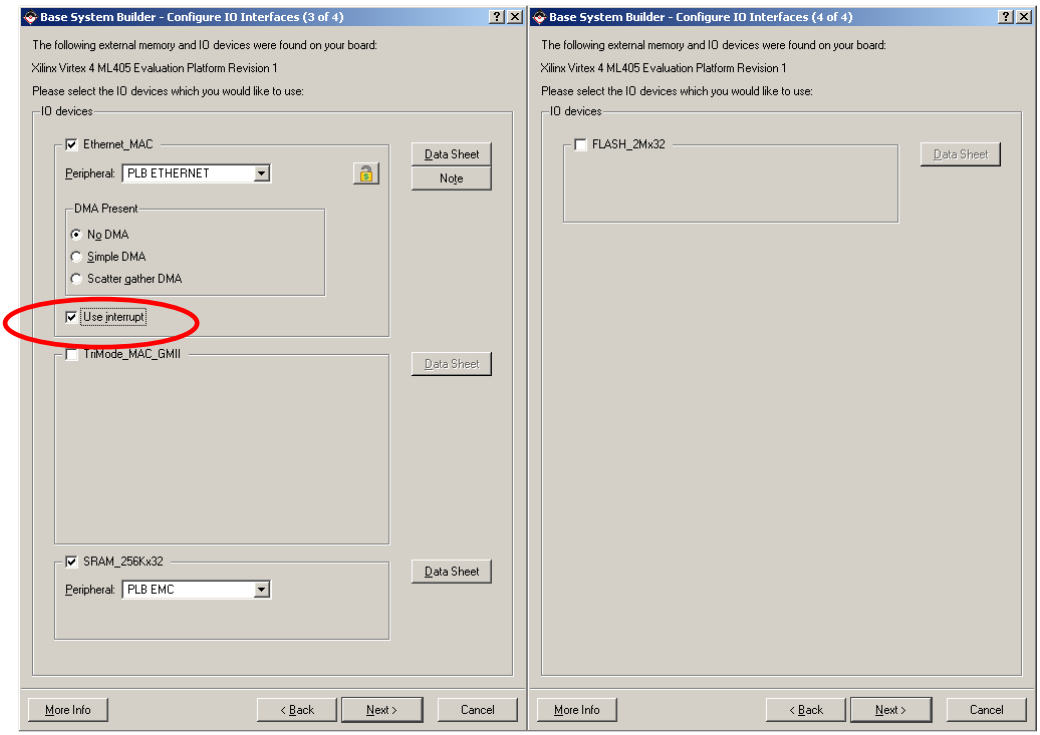

Figure 10: Configure IO Interface (3 and 4 of 4)

Having configured the IO interfaces, the internal peripherals must be configured. In this case there is only one - the “plb_bram_if_cntlr_1” - which is an internal memory attached to the PLB interface.

NOTE that the internal house-keeping of memory resources does not seem to work properly. Therefore, in order for the synthesis not to fail, you have to limit the volume to 64 KB. On the other hand, if you lower the volume of RAM below 16 KB, Linux will not be able to boot.

Yet this memory will contain the boot code so in order to get a fast start-up, it is advisable to use as much memory as possible.

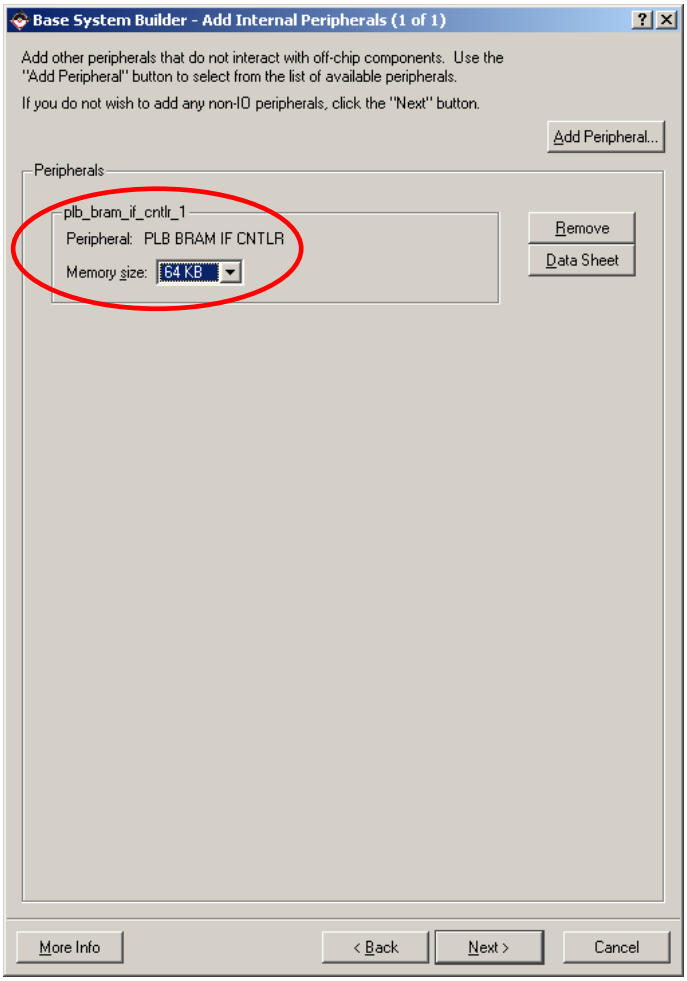

Figure 11: Add Internal Peripherals (1 of 1) window

The last window is the Software Setup Window. Its purpose is to define which IO the STDIN and STDOUT streams are attached to. In this case use the RS232_Uart in order to be able to use the serial interface of the ML40x board as console. You have to use “plb_bram_if_cntlr_1” as boot memory.
The sample applications are not necessary and should be deselected.

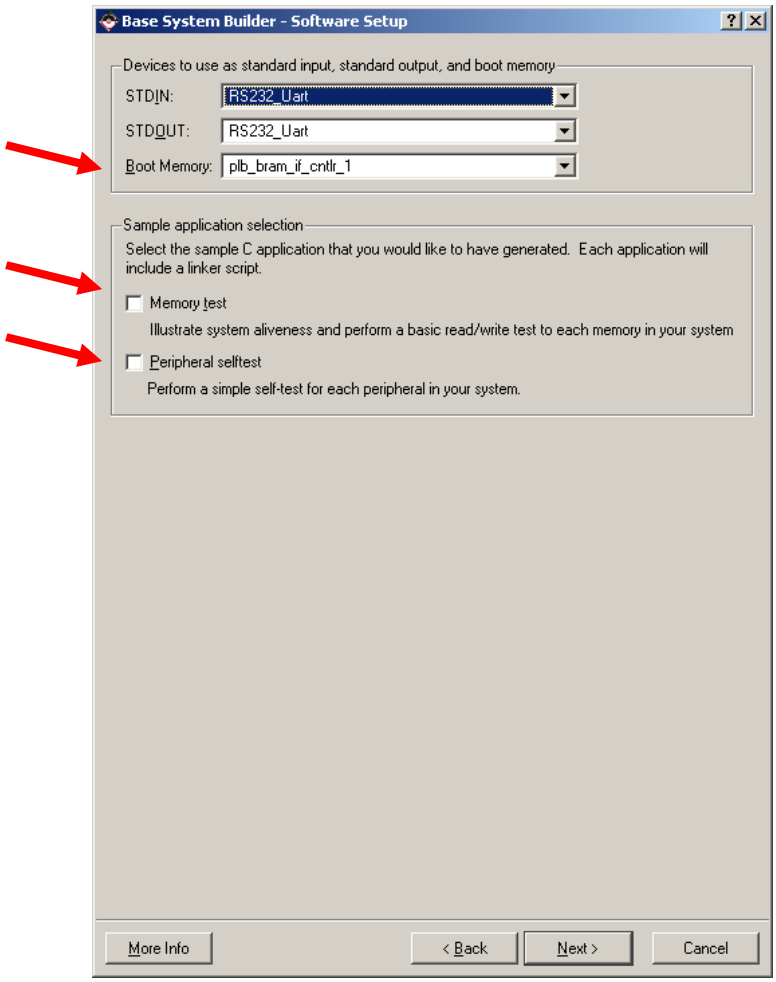

Figure 12: Software Setup Screen

After the Software Setup, the system will be created - but you will be presented with a resume window which should be verified carefully.

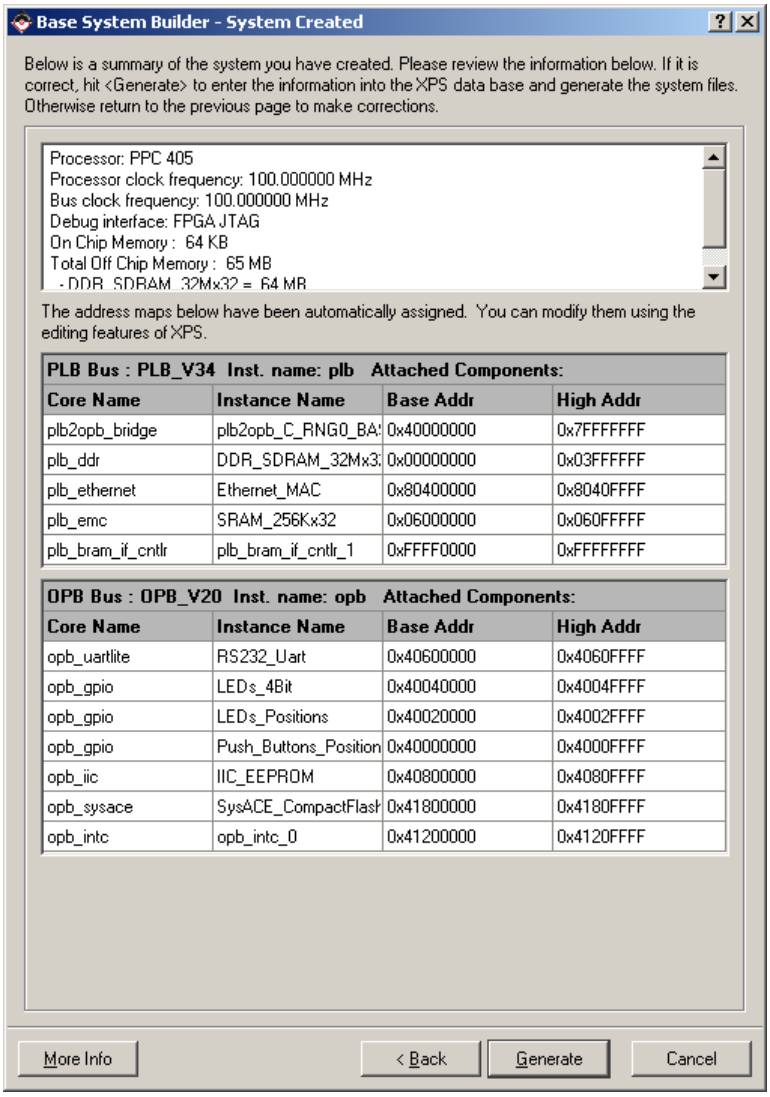

Figure 13: Resume Window

If everything is OK, press “Generate” and your system will be created.

After a while the Base System Builder will present a Finish window to indicate that your system has been created.

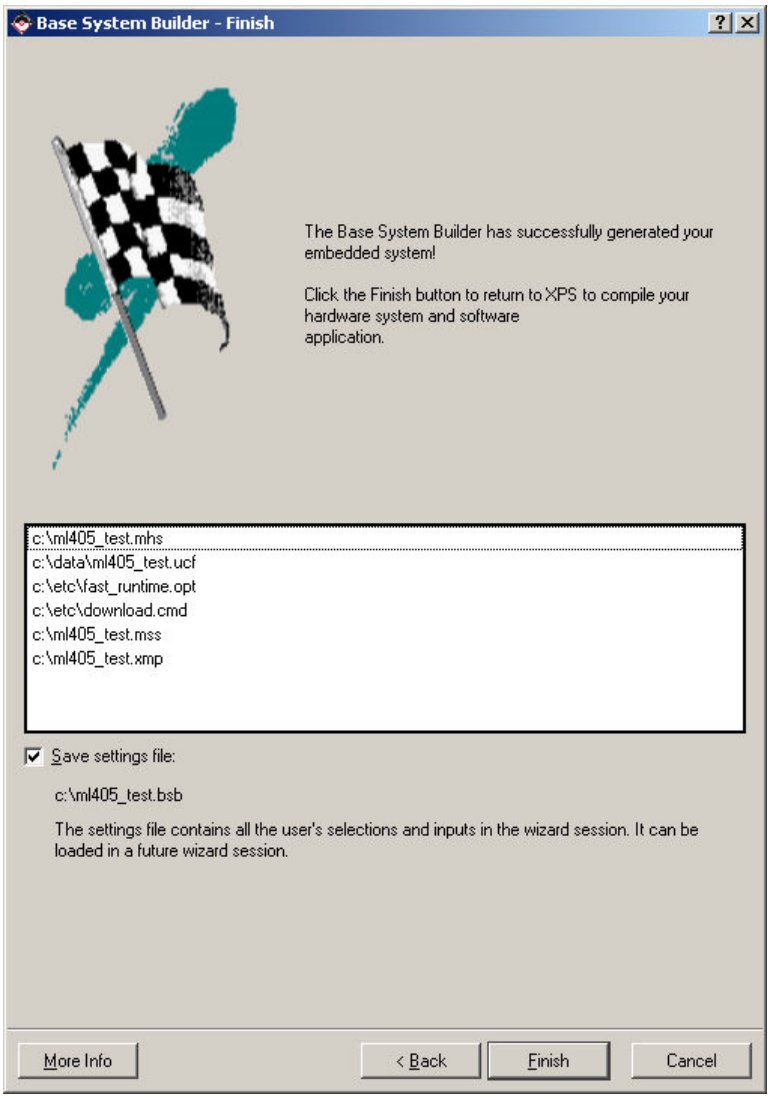

Figure 14: Final Window

Press “Finish” if you are satisfied.

When your build is complete, the main window of the Xilinx Platform studio will look like this:

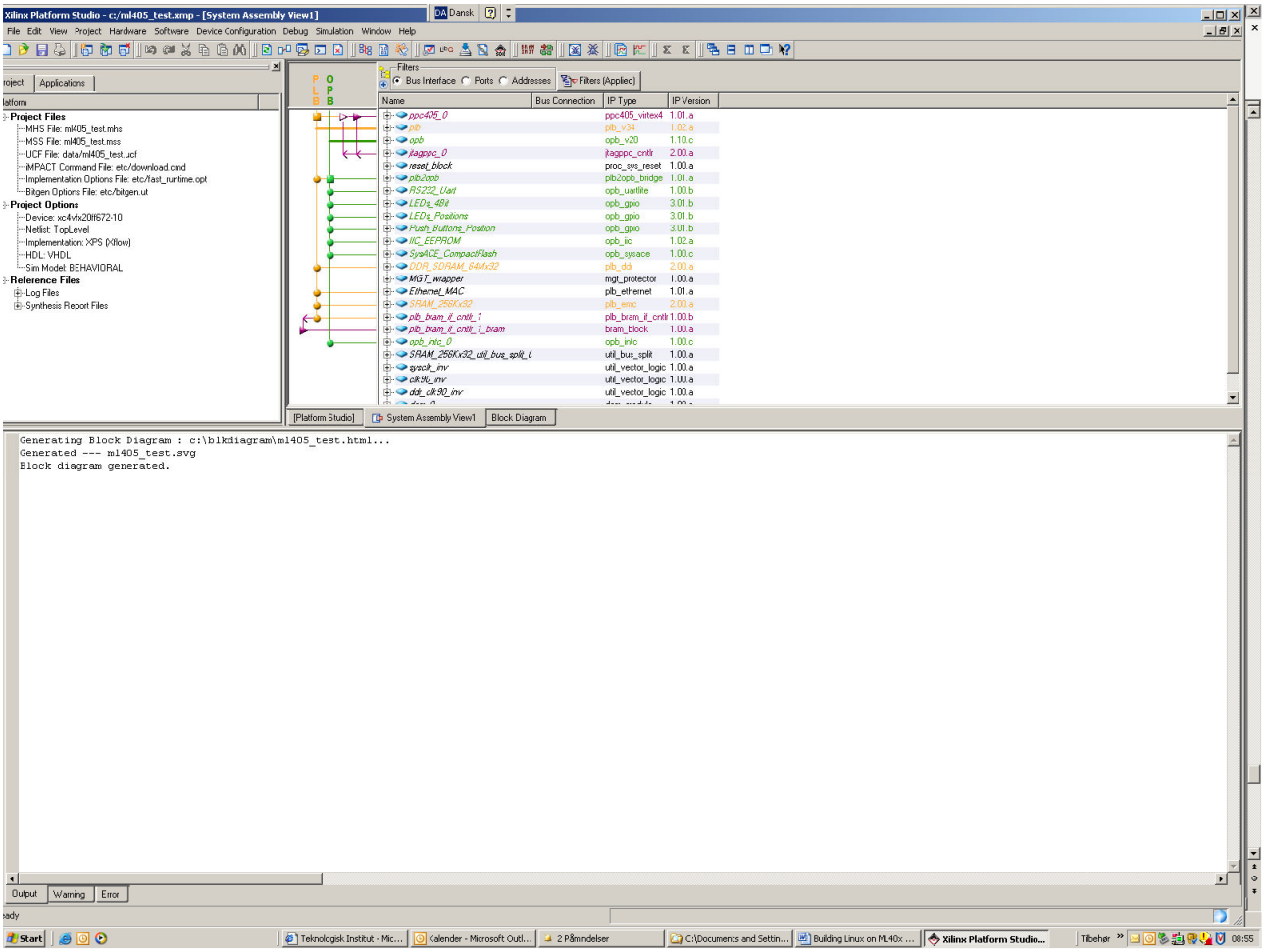

Figure 15: Xilinx Platform Studio main Window

You are now ready to generate your bitstream file - provided nothing else should be added.

If you are generating an image for the ML405 board, you have to carry out the following activity: Enter the <DESIGNROOT>/etc/bitgen.ut file and add the following statement:

```
-g Match_cycle:NoWait
```

If you omit this, the booting using the systemace will not work…

You can now generate your Bitstream. Select the "**Generate Bitstream**" item in the "**Hardware**" menu.

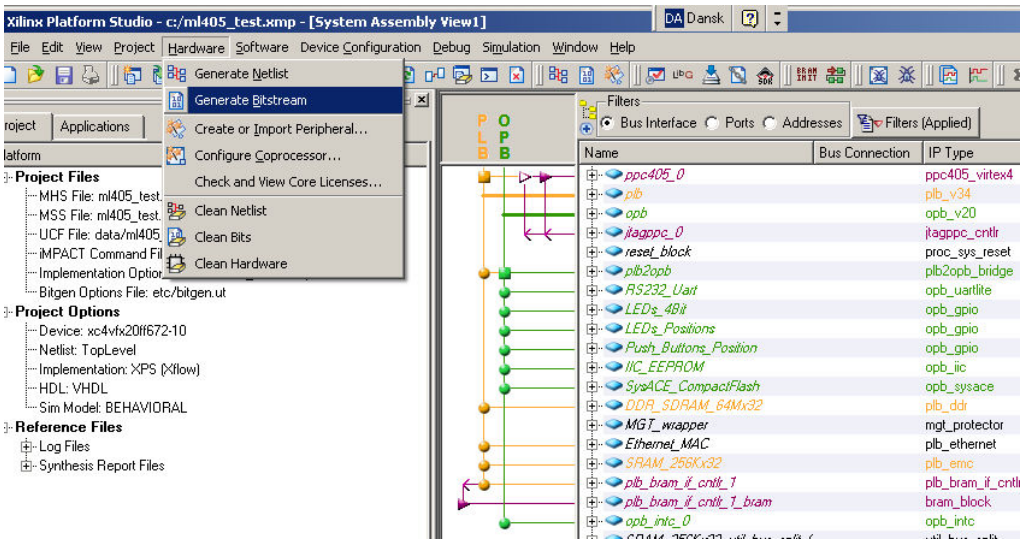

Figure 16: Selecting "Generate Bitstream" item in "Hardware" menu

After a while, and provided everything goes well, the XPS will present a popup window indicating the Design License Status. It indicates that there are unpaid cores and therefore you can only use the cores for design evaluation. The cores will cease to work after some time but you will be able to get your system running.

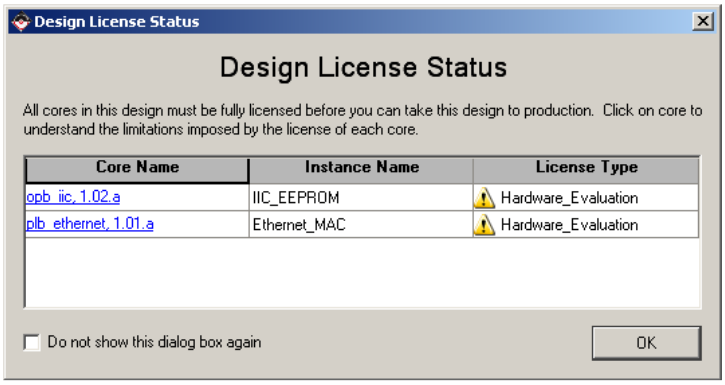

Figure 17: Design License Status Window.

Press the “OK” button.

The main window will look like this:

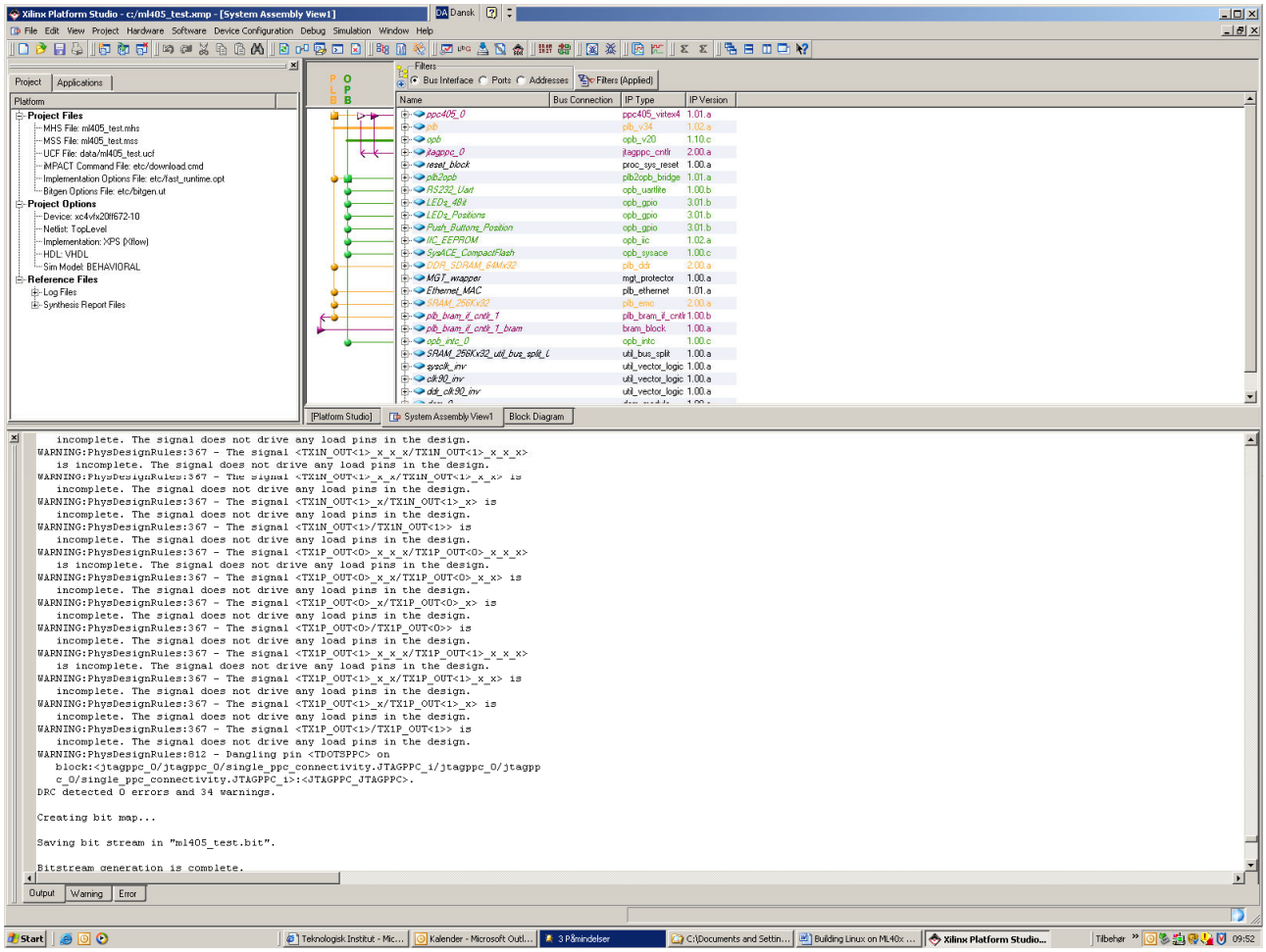

Figure 18: XPS main window after Bitstream Generation

The main thing to look for are the last two lines in the output window "**Saving bit stream in "ml405_test.bit".**" and "**Bitstream generation is complete.**". The two lines indicate that the bitfile is generated properly.

To be absolutely certain that everything is OK, it is advisable to verify the Logic Utilization and the routing summary.

The Logic Utilization indicates whether the device is over-utilized, and it also gives a hint as to whether you have sufficient resources in case you wish to write your own logic core.
Both reports are part of the log which can be found in the Reference file list of the Project Tab in the upper left window. The log you have to look into is the implementation/xflow.log.
If you click on this item, the xflow log will appear in the upper right window covering the system assembly window.

You have to use the slide bar to position the window in order for the Logic Utilization window to be viewed in the upper right window as depicted in Figure 19.

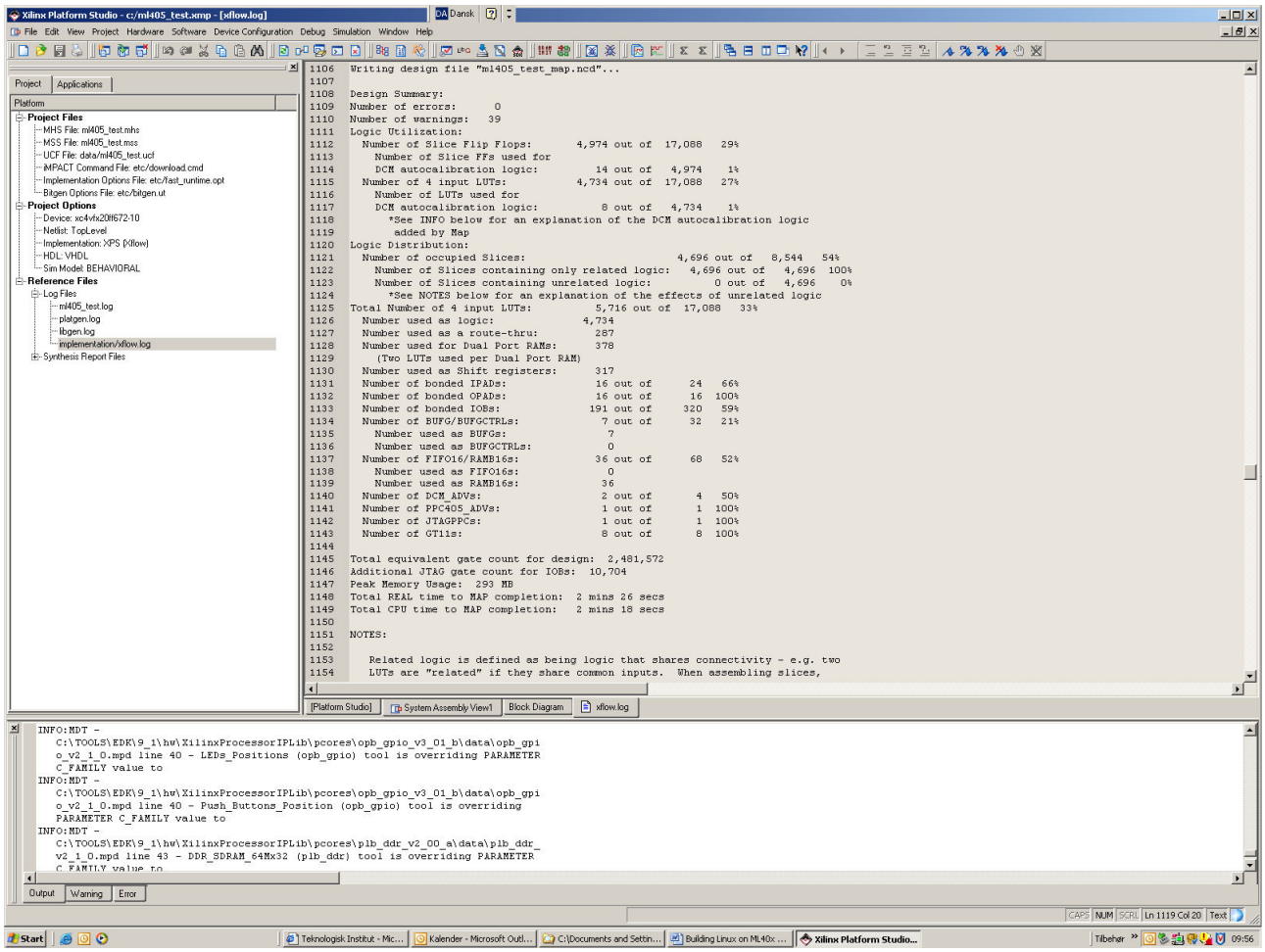

Figure 19: XPS Main window with the xflow log file in the upper right window

Later in the report, the routing summary is depicted (see Figure 20). The most important sentences to search for is “All constraints were met” and “All signals are completely routed.”.

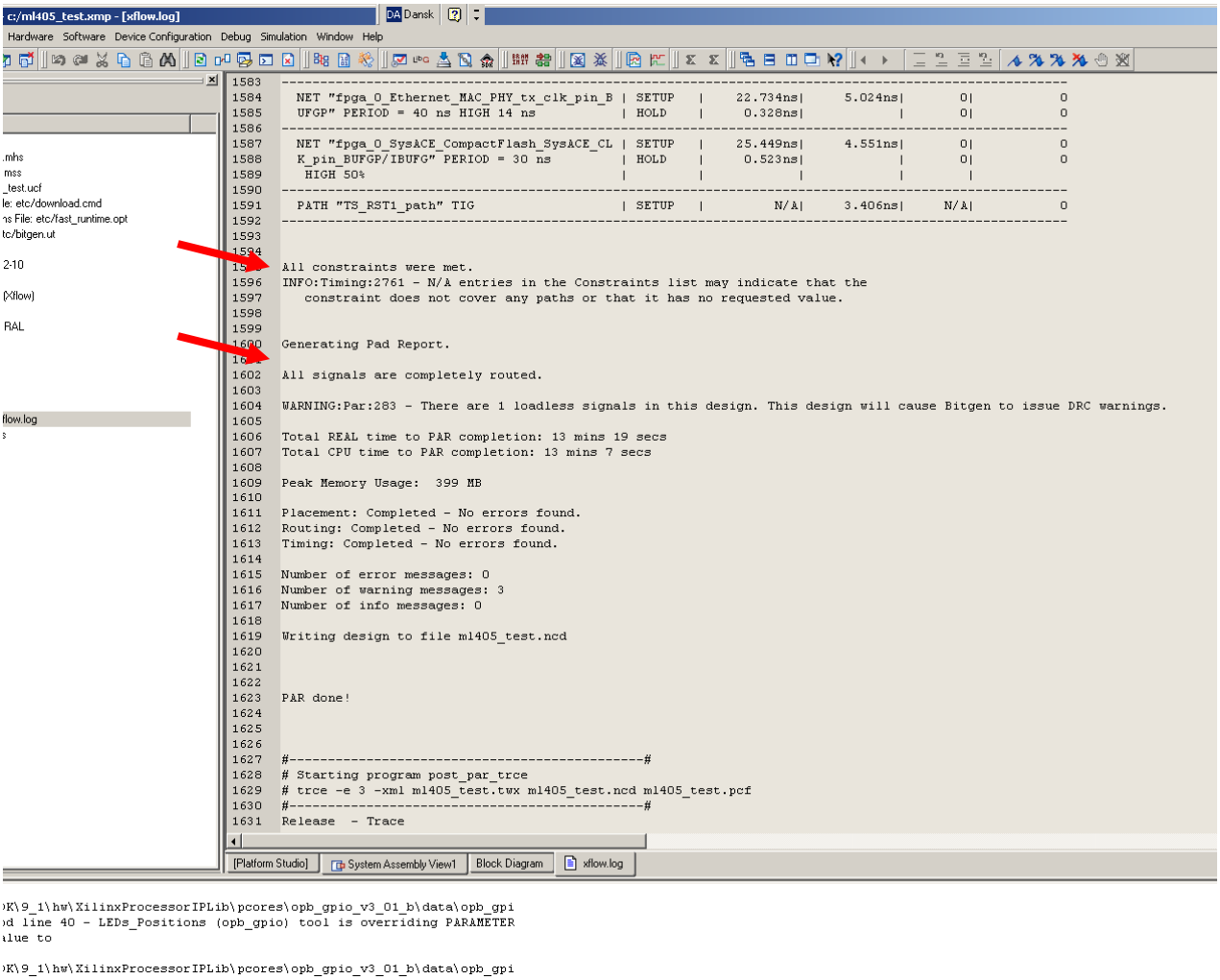

Figure 20: XPS Main window with the xflow log file in the upper right window

Now you have a bit file containing the functionality you wanted. However, the PowerPC within the bit file is not able to boot yet. Instructions on how to handle this is covered in the next section.

5.2.2 Inserting the Boot Loader into Block RAMs

To insert the boot loader into the Block RAMs you have to select the “Update Bitstream” item of the “Device Configuration” menu.

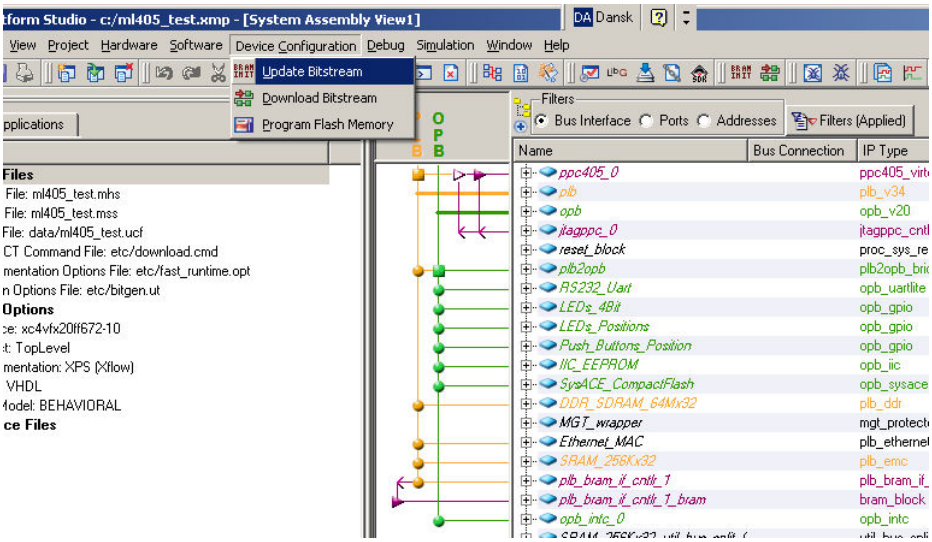

Figure 21: Selecting the ”Update Bitstream” item of the “Device Configuration” menu.

Everything is performed automatically, and therefore no selections have to be made.

5.2.3 Creating the board support package

To generate the Board Support Package, you first have to inform XPS that you wish to use Linux and supply additional information concerning the drivers. This is carried out by selecting the "Software Platform Settings…" item of the "Software" menu.

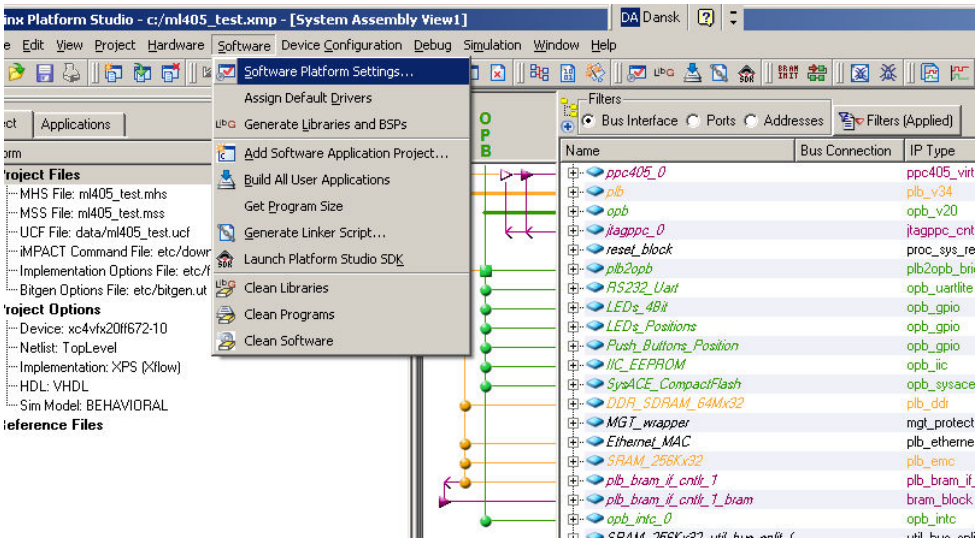

Figure 22: Selecting the "Software Platform Settings…" item of the "Software" menu

The Software Platform Settings dialog window is depicted in Figure 23.

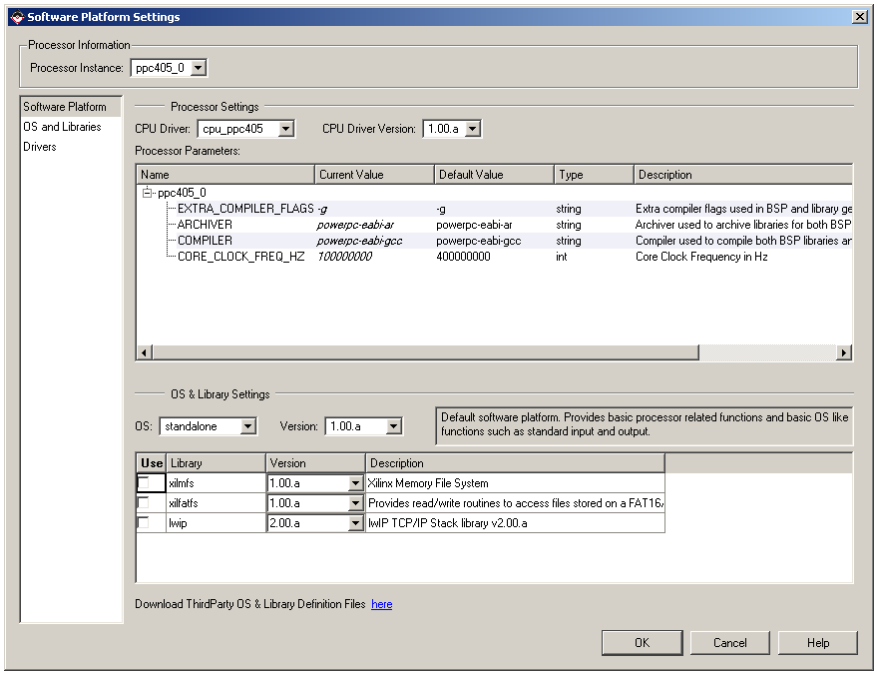

Figure 23: Software Platform Settings dialog window

First the OS has to be selected. To compile Linux kernel version 2.6: Select the corresponding item (linux_2_6) in the OS dropdown menu as depicted in Figure 24.

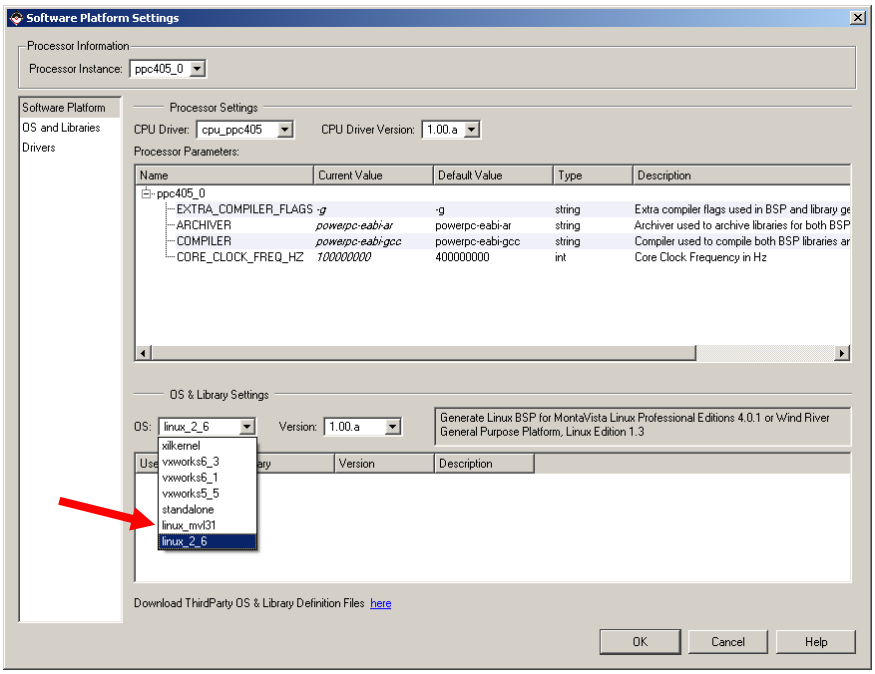

Figure 24: Selecting the "linux_2_6" option in the OS dropdown menu

Having selected the correct operating system, you have to select the "**OS and Libraries**" item in the list to the left. This will cause the **linux_2_6** configuration list to appear, as depicted in Figure 25. The list contains a sub-item labeled "**connected_periphs**" which has the "**Current Value**" called "**Edit…**". It has to be changed by double-clicking the "Current Value"-field in the list which causes the "**Add/Delete List of Parameter-Values**" popup to appear as depicted in Figure 26.

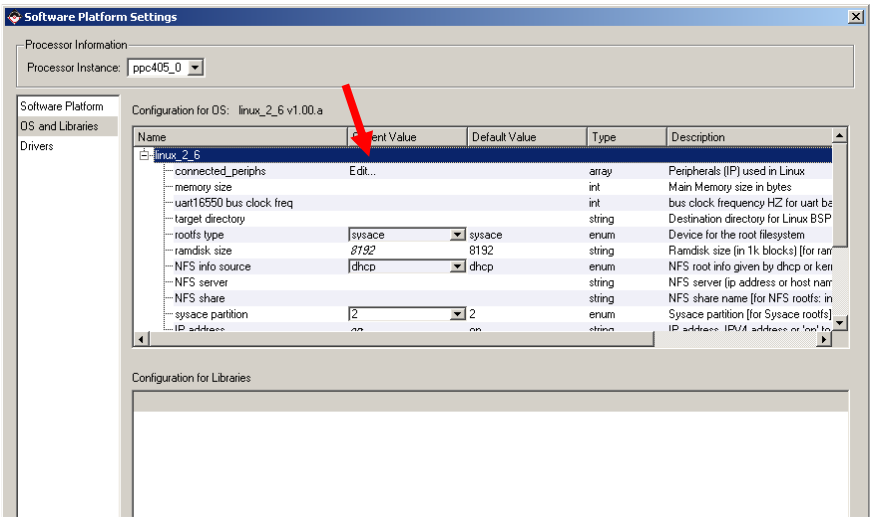

Figure 25: The Software Platform Settings dialog window, after selecting OS and libraries

To enter the correct parameter values, simply click "**Add All**". This causes a popup to appear as depicted in Figure 27.

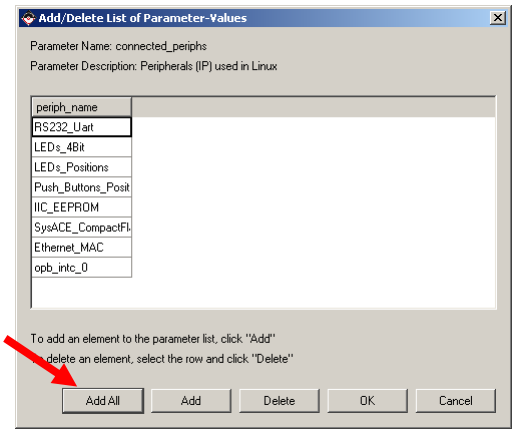

Figure 26: Add/Delete List of Parameter-Values window

This window is quite confusing but just accept it and click on the "Yes" button.

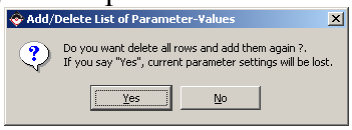

Figure 27: Add/Delete List of Parameter-Values popup window

After you have inserted the connected_periphs, the “Sofware Platform Settings” window shall appear as depicted in Figure 28.

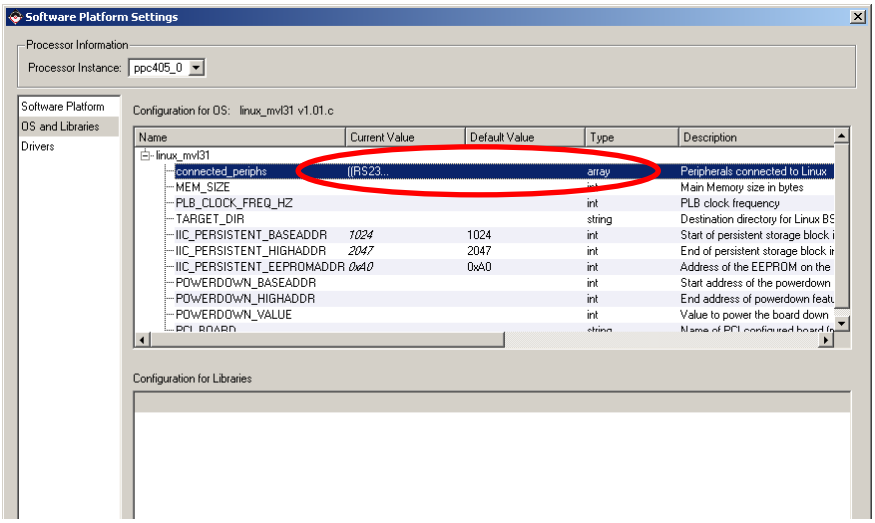

Figure 28: Software Platform Settings dialog box after adding the list of connected peripherals

The volume of memory to be entered is also a bit confusing. For both ML403 and ML405 boards you have to use the value 0x04000000 which is safe. To enter the value, perform a left double-click on the “Current Value” field of the MEM_SIZE sub-item. The field will transform into a number box which you can edit by using the keyboard.

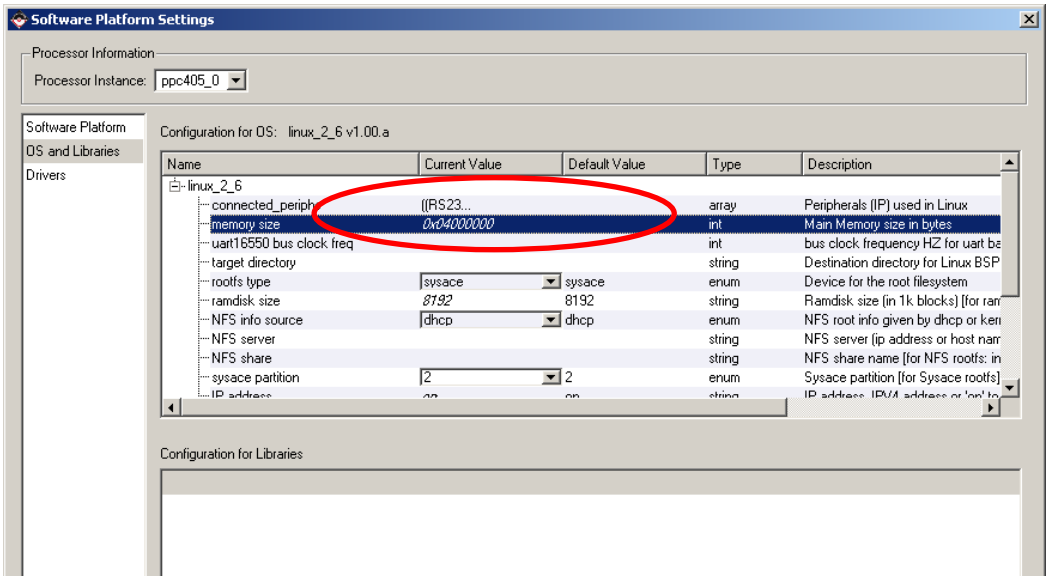

Figure 29: Software Platform Settings after insertion of the MEM_SIZE parameter

To enter the clock frequency of the PLB bus, perform a left double-click on the “Current Value” field of the “PLB_CLOCK_FREQ_HZ” sub_item. This field will also transform into a number box which you can edit by using the key board.
The last item to be changed is the TARGET_DIR sub-item. Again left double-click on the “Current Value” field of the TARGET_DIR sub_item. The field will transform itself into a text box which can be edited by using the keyboard.
NOTE: Do not use backslash, use forward slash instead.

After everything has been filled in, the “Software Platform Settings” dialog box will appear as depicted in Figure 30.

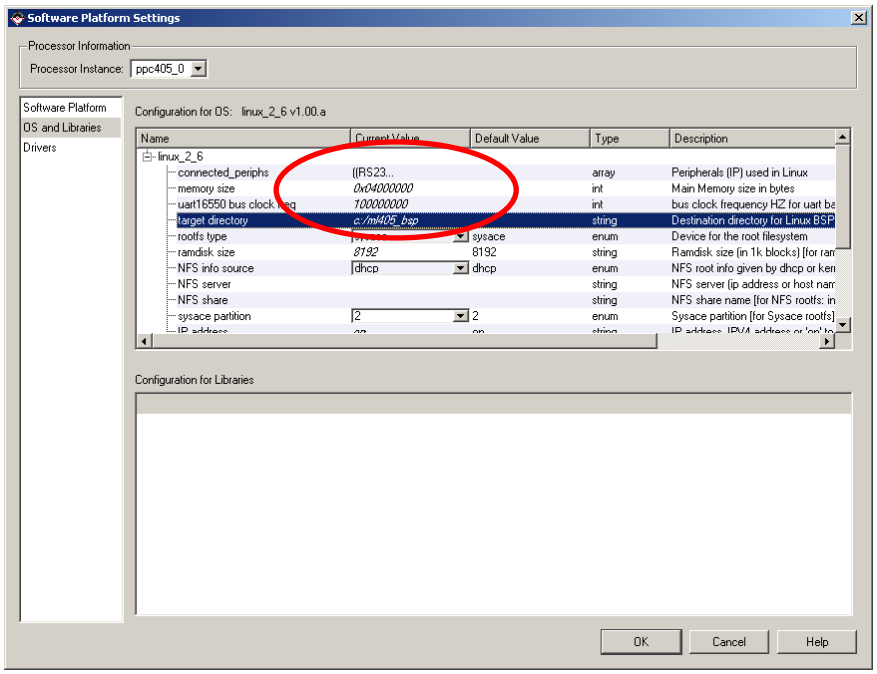

Figure 30: The Completely filled Software Platform Setting dialog box

The last thing to carry out is to ask XPS to generate the BSP. This is performed by selecting the "Generate Libraries and BSPs" item from the "Software" menu.

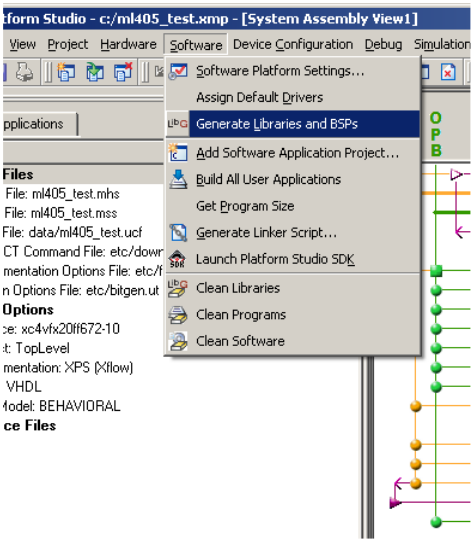

Figure 31: Selecting the "Generate Libraries and BSP" item from the Software Menu

During the generation of BSP, XPS will inform you that there is a problem in generating the GPIO driver:

```
WARNING: Gpio driver version 2.01.a not supported with MVL 3.1.
         Use version 1.00.a or 1.01.a
```

If the GPIO driver has not been used, you can safely ignore this message.

6 Getting, Patching, Setting up and Compiling the Kernel

6.1 Kernel Versions

The programme bit keeper was necessary in the previous document. For kernel 2.6 we will use a standard kernel and the bitkeeper is therefore no longer necessary.

6.2 Getting the Kernel Source

The kernel source used in this document is
http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.23.tar.bz2
However, any subsequent sub-versions of Linux 2.6 should be usable.

Having obtained the kernel source, it can be extracted using the command:

```
$ tar xvjf linux-2.6.23.tar.bz2
```

Now you can enter the root tree by using:

```
$ cd linux-2.6.23
```

6.3 Configuring the Makefile

Before setting up the kernel, it is necessary to configure the makefile in order for the configuration tool to identify which compiler and host to target your kernel to.

Make sure that the following variables are set in your kernel:

```
ARCH            := ppc

CROSS_COMPILE := powerpc-405-linux-gnu-
```

Make sure that you have at least one Line feed/Carriage return after both lines. You may encounter problems if the file has been generated or edited in windows.

The latter dash of the “CROSS_COMPILE” variable value is NOT an error!

6.4 Inserting an Ethernet Driver

The first set of files to be added is the “xilinx_common” library. This library was generated as part of the board support package and is located in

```
<bsp_root> /drivers/xilinx_common
```

The entire library has to be copied into

```
<linux_root>/drivers:
```

Use the command:

```
$ cp -R <bsp_root>/drivers/xilinx_common <linux_root>/drivers
```

A `Makefile` must be created with the following contents:

```
#
# Makefile for the Xilinx On Chip Peripheral support code
#
# The Xilinx OS independent code.
xilinx_ocp-objs += xbasic_types.o xdma_channel.o xdma_channel_sg.o \
                xipif_v1_23_b.o \
                xpacket_fifo_v2_00_a.o xpacket_fifo_l_v2_00_a.o \
                xversion.o



obj-$(CONFIG_XILINX_OCP) := xilinx_ocp.o

xilinx_ocp.o: $(xilinx_ocp-objs)
           $(LD) -r -o $@ $(xilinx_ocp-objs)

#include $(TOPDIR)/Rules.make

obj-$(CONFIG_XILINX_EMAC) += xilinx_ocp.o
```

The last file to be created is the "**xio.h**" file which contains important primitives for the Xilinx/PPC interface such as endianess coding. This file is NOT generated with the board support package.

This file has to contain at least the following code:

```
#ifndef XIO_H
#define XIO_H
/*
  This code is derived from Monta Vista, INC. and was released under GPL, which
implies that any code
  which uses this code, or is statically linked into the Linux Kernel must be
released under the same terms.

*/

#include "xbasic_types.h"
#include <asm/io.h>

typedef u32 XIo_Address;

extern inline u8
XIo_In8(XIo_Address InAddress)
{
           return (u8) in_8((volatile unsigned char *) InAddress);
}

extern inline u16
XIo_In16(XIo_Address InAddress)
{
           return (u16) in_be16((volatile unsigned short *) InAddress);
}

extern inline u32
```

```
XIo_In32(XIo_Address InAddress)
{
          return (u32) in_be32((volatile unsigned *) InAddress);
}

extern inline void
XIo_Out8(XIo_Address OutAddress,  u8  Value)
{
          out_8((volatile unsigned char *) OutAddress, Value);
}
extern inline void
XIo_Out16(XIo_Address OutAddress, u16 Value)
{
  out_be16((volatile unsigned short *) OutAddress, Value);
}

extern inline void
XIo_Out32(XIo_Address OutAddress, u32 Value)
{
          out_be32((volatile unsigned *) OutAddress, Value);
}
#endif
```

and has to be located in the `<linux_root>drivers/xilinx_common` directory.

After completing this activity, the following change has to be carried out to the Makefile located in the `<linux_root>/drivers` directory:

Old file:

```
obj-$(CONFIG_FB_INTEL)          += video/intelfb/
obj-y                           += serial/
obj-$(CONFIG_PARPORT)           += parport/
```

New file:

```
obj-$(CONFIG_FB_INTEL)          += video/intelfb/
obj-$(CONFIG_XILINX_EMAC)       += xilinx_common/
obj-y                           += serial/
obj-$(CONFIG_PARPORT)           += parport/
```

The dependency of CONFIG_XILINX_EMAC is a bit of a hack and is only valid when adding one Xilinx driver from XPS.

In the `<linux_root>/drivers/Kconfig` file the following change has to be carried out (if you compile a Linux 2.6.24 kernel, correct the `<linux_root>/drivers/net/Kconfig` file instead):

Old file:

```
config MII
          tristate "Generic Media Independent Interface device support"
          help
            Most ethernet controllers have MII transceiver either as an…
```

```
            or internal device.  It is safe to say Y or M here even if your
            ethernet card lack MII.

config MACB
          tristate "Atmel MACB support"
          depends on AVR32 || ARCH_AT91SAM9260 || ARCH_AT91SAM9263
          select PHYLIB
```

New File:

```
config MII
          tristate "Generic Media Independent Interface device support"
          help
            Most ethernet controllers have MII transceiver either as an….
            or internal device.  It is safe to say Y or M here even if your
            ethernet card lack MII.

config XILINX_EMAC
        tristate "Xilinx Ethernet Mac"
        help
          TEST TEST TEST


config MACB
          tristate "Atmel MACB support"
          depends on AVR32 || ARCH_AT91SAM9260 || ARCH_AT91SAM9263
          select PHYLIB
```

The addition of the above line will make it possible to select the Xilinx Ethernet MAC when configuring the kernel.

Now the Ethernet Driver has to be copied into the `<linux_root>/drivers/net` directory of the Linux distribution:

```
$ cp -R <bsp_root>/drivers/net/xilinx_emac <linux_root>/drivers/net
```

and the Makefile in <linux_root>/drivers/net has to be modified:

Old file:

```
obj-$(CONFIG_EHEA) += ehea/
obj-$(CONFIG_BONDING) += bonding/
obj-$(CONFIG_ATL1) += atl1/
obj-$(CONFIG_GIANFAR) += gianfar_driver.o
```

New file:

```
obj-$(CONFIG_EHEA) += ehea/
obj-$(CONFIG_BONDING) += bonding/
obj-$(CONFIG_ATL1) += atl1/
obj-$(CONFIG_XILINX_EMAC) += xilinx_emac/
obj-$(CONFIG_GIANFAR) += gianfar_driver.o
```

The last thing to copy is the parameter's file:

```
$ cp -R <bsp_root>/arch/ppc/platforms/4xx/xparameters/xparameters_ml40x.h \
<linux_root>/arch/ppc/platforms/4xx/xparameters/xparameters_ml403.h
```

Note that the name changes!

Now the entire distribution has been patched manually but the kernel still has no awareness of the Ethernet MAC resources (interrupt and memory location). They also have to be added manually.

6.5 Adding Resources

The file containing the resource initialization is the `<linux_root>/arch/ppc/syslib/virtex_devices.c` file.

The contents of the file has to be changed in two locations:

The first change is to declare the platform data, device features and such:

Old file:

```
                                .end = XPAR_UARTLITE_##num##_HIGHADDR, \
                                .flags = IORESOURCE_MEM, \
                        }, \
                        { \
                                .start = XPAR_INTC_0_UARTLITE_##num##_VEC_ID, \
                                .flags = IORESOURCE_IRQ, \
                        }, \
        }, \
}


/*
 * Full UART: shortcut macro for single instance + platform data structure
 */
#define XPAR_UART(num) { \
        .mapbase = XPAR_UARTNS550_##num##_BASEADDR + 3, \
        .irq = XPAR_INTC_0_UARTNS550_##num##_VEC_ID, \
        .iotype = UPIO_MEM, \
        .uartclk = XPAR_UARTNS550_##num##_CLOCK_FREQ_HZ, \
        .flags = UPF_BOOT_AUTOCONF, \
        .regshift = 2, \
```

New File:

```
                                .end = XPAR_UARTLITE_##num##_HIGHADDR, \
                                .flags = IORESOURCE_MEM, \
                        }, \
                        { \
                                .start = XPAR_INTC_0_UARTLITE_##num##_VEC_ID, \
                                .flags = IORESOURCE_IRQ, \
                        }, \
        }, \
}
```

```
struct xemac_platform_data {
            u32 device_flags;
            u32 dma_mode;
            u32 has_mii;
            u32 has_err_cnt;
            u32 has_cam;
            u32 has_jumbo;
            u32 tx_dre;
            u32 rx_dre;
            u32 tx_hw_csum;
            u32 rx_hw_csum;
            u8 mac_addr[6];
};

/* Flags related to XEMAC device features */
#define XEMAC_HAS_ERR_COUNT      0x00000001
#define XEMAC_HAS_MII            0x00000002
#define XEMAC_HAS_CAM            0x00000004
#define XEMAC_HAS_JUMBO                     0x00000008

/* Possible DMA modes supported by XEMAC */
#define XEMAC_DMA_NONE                      1
#define XEMAC_DMA_SIMPLE         2          /* simple 2 channel DMA */
#define XEMAC_DMA_SGDMA          3          /* scatter gather DMA */


static struct xemac_platform_data xemac_0_pdata = {
            // device_flags is used by older emac drivers. The new style is to
use separate feilds
            .device_flags = (XPAR_EMAC_0_ERR_COUNT_EXIST ? XEMAC_HAS_ERR_COUNT :
0) |
                        (XPAR_EMAC_0_MII_EXIST ? XEMAC_HAS_MII : 0) |
                        (XPAR_EMAC_0_CAM_EXIST ? XEMAC_HAS_CAM : 0) |
                        (XPAR_EMAC_0_JUMBO_EXIST ? XEMAC_HAS_JUMBO : 0),
            .dma_mode = XPAR_EMAC_0_DMA_PRESENT,
            .has_mii     = XPAR_EMAC_0_MII_EXIST,
            .has_err_cnt = XPAR_EMAC_0_ERR_COUNT_EXIST,
            .has_cam     = XPAR_EMAC_0_CAM_EXIST,
            .has_jumbo   = XPAR_EMAC_0_JUMBO_EXIST,
            .tx_dre      = XPAR_EMAC_0_TX_DRE_TYPE,
            .rx_dre      = XPAR_EMAC_0_RX_DRE_TYPE,
            .tx_hw_csum  = XPAR_EMAC_0_TX_INCLUDE_CSUM,
            .rx_hw_csum  = XPAR_EMAC_0_RX_INCLUDE_CSUM,
};

#define XPAR_XEMAC(num) { \
            .name = "xilinx_emac", \
            .id = num, \
            .dev.platform_data = &xemac_0_pdata, \
            .num_resources = 2, \
            .resource = (struct resource[]) { \
                        { \
                                    .start = XPAR_EMAC_##num##_BASEADDR, \
                                    .end = XPAR_EMAC_##num##_HIGHADDR, \
                                    .flags = IORESOURCE_MEM, \
                        }, \
                        { \
```

```
                                .start = XPAR_INTC_0_EMAC_##num##_VEC_ID, \
                                .flags = IORESOURCE_IRQ, \
                        }, \
        }, \
}

/*
 * Full UART: shortcut macro for single instance + platform data structure
 */
#define XPAR_UART(num) { \
        .mapbase = XPAR_UARTNS550_##num##_BASEADDR + 3, \
        .irq = XPAR_INTC_0_UARTNS550_##num##_VEC_ID, \
        .iotype = UPIO_MEM, \
        .uartclk = XPAR_UARTNS550_##num##_CLOCK_FREQ_HZ, \
        .flags = UPF_BOOT_AUTOCONF, \
        .regshift = 2, \
```

And the second change is to insert the EMAC into the device table:

Old File:

```
struct platform_device virtex_platform_devices[] = {
        /* UARTLITE instances */
#if defined(XPAR_UARTLITE_0_BASEADDR)
        XPAR_UARTLITE(0),
#endif
#if defined(XPAR_UARTLITE_1_BASEADDR)
```

New File:

```
struct platform_device virtex_platform_devices[] = {
        /* Xilinx MAC instances */
#if defined(XPAR_EMAC_0_BASEADDR)
        XPAR_XEMAC(0),
#endif
        /* UARTLITE instances */
#if defined(XPAR_UARTLITE_0_BASEADDR)
        XPAR_UARTLITE(0),
#endif
#if defined(XPAR_UARTLITE_1_BASEADDR)
```

Now you are ready to set up the kernel!

6.6 Setting up the kernel

To set up the kernel we recommend the use of the “make menuconfig” command. Other options are either cumbersome or have been reported erroneous. The “make menuconfig” command requires that you have ncurses installed. If it is not installed, the “make menuconfig” will not work. If so, contact your system administrator.

Below all configuration menus are described at top level, and if the menu point has more levels they will be indented in the description. Also note that all options (including sub-options) which are

either not mentioned or not set should not be set. Please note that for different sub-versions of Linux 2.6 the menus may differ slightly.

6.6.1 General Setup

```
General setup  --->
 [*] Prompt for development and/or incomplete code/drivers
 ()  Local version - append to kernel release
 [ ] Automatically append version information to the version string
 [ ] Support for paging of anonymous memory (swap)
 [ ] System V IPC
 [ ] POSIX Message Queues
 [ ] BSD Process Accounting
 [ ] Export task/process statistics through netlink (EXPERIMENTAL)
 [ ] User Namespaces (EXPERIMENTAL)
 [ ] Auditing support
 [ ] Kernel .config support
 (17) Kernel log buffer size (16 => 64KB, 17 => 128KB)
 [*] Create deprecated sysfs files
 [ ] Kernel->user space relay support (formerly relayfs)
 [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
 [*] Optimize for size (Look out for broken compilers!)
 [*] Configure standard kernel features (for small systems)  --->
 [*] Enable eventpoll support
 [*] Enable signalfd() system call
 [*] Enable eventfd() system call
 [*] Use full shmem filesystem
 [*] Enable VM event counters for /proc/vmstat
 Choose SLAB allocator (SLAB)  --->
    (X) SLAB
    ( ) SLUB (Unqueued Allocator)
    ( ) SLOB (Simple Allocator)
```

Many drivers used for the Xilinx FPGAs are development drivers. Furthermore some advanced settings are necessary.

6.6.2 Loadable Module Support

```
  [*] Enable loadable module support  --->
      --- Enable loadable module support
      [ ]   Module unloading (NEW)
      [ ]   Module versioning support (NEW)
      [ ]   Source checksum for all modules (NEW)
      [ ]   Automatic kernel module loading (NEW)
```

This is not necessary for the kernel to work but will certainly make device driver development simpler.

6.6.3 Block Layer

```
   [*] Enable the block layer  --->
       --- Enable the block layer
       [*]   Support for Large Block Devices
       [ ]   Support for tracing block io actions
       [ ]   Support for Large Single Files
       [ ]   Block layer SG support v4 (EXPERIMENTAL)
             IO Schedulers  --->
             <*> Anticipatory I/O scheduler
             <*> Deadline I/O scheduler
             <*> CFQ I/O scheduler
             Default I/O scheduler (CFQ)  --->
               ( ) Anticipatory
               ( ) Deadline
               (X) CFQ
               ( ) No-op
```

A lot has happened since the 2.4 kernel was developed. Please refer to Chapter “Block Device Drivers” in [2] to obtain information about the Block Layer Schedulers.

6.6.4 Processor

```
Processor  --->
  Processor Type (40x)  --->
    ( ) 6xx/7xx/74xx/52xx/82xx/83xx
    (X) 40x
    ( ) 44x
    ( ) 8xx
    ( ) e200
    ( ) e500
  [*] Math emulation
  [ ] kexec system call (EXPERIMENTAL)
  [ ] CPU Frequency scaling
      IBM 4xx options  --->
        [ ] PPC4xx DMA controller support
        TTYS0 device and default console (UART0)  --->
        (X) UART0
        ( ) UART1
```

The processor in the Xilinx Virtex 4 family is a PowerPC 405, and the settings for Xilinx-ML300 works on the ML403/ML405 boards as well.

Math emulation is also required because no FPU is configured on the Virtex 4 device (it is possible but occupies space).

```
The default console has to be configured to UART0.
     Platform options  --->
       PC PS/2 style Keyboard
         [ ] High memory support
         Timer frequency (250 HZ)  --->
         ( ) 100 HZ
         (X) 250 HZ
         ( ) 300 HZ
         ( ) 1000 HZ
       Preemption Model (Voluntary Kernel Preemption (Desktop))  --->
         ( ) No Forced Preemption (Server)
         (X) Voluntary Kernel Preemption (Desktop)
         ( ) Preemptible Kernel (Low-Latency Desktop)
       Memory model (Flat Memory)  --->
         (X) Flat Memory
       [ ] 64 bit Memory and IO resources (EXPERIMENTAL)
       [*] Kernel support for ELF binaries
       [ ] Kernel support for MISC binaries
       [*] Default bootloader kernel arguments
           (console=ttyUL0 root=/dev/xsa5 rw) Initial kernel command string
       [ ] Power Management support
       [ ] Enable seccomp to safely compute untrusted bytecode
```

The default kernel command string options mean that:

- The Xilinx uartlite port (`ttyUL0`) is used at 9600 Mbps speed. **Make sure that the device is `tt10` (tee-tee-el-zero) and not `tt10` (tee-tee-one-zero).**
- The Root system is located on the `/dev/xsa5` partition and mounted in read/write mode.
- That ip is started during boot (necessary to get DCHP operational).

6.6.5 Bus Options

```
Bus options  --->
  [ ] PCI support
  PCCARD (PCMCIA/CardBus) support  --->
```

```
        [ ] PCCard (PCMCIA/CardBus) support
```

No special busses are used.

6.6.6 Advanced Setup

```
    Advanced Setup  --->
    [*] Prompt for advanced kernel configuration
    [ ]   Set maximum low memory
    [ ]   Set custom kernel base address
    [ ]   Set custom user task size
    [ ]   Set custom consistent memory pool address
    [ ]   Set custom consistent memory pool size
    [ ]   Set the boot link/load address
```

No special option is used.

6.6.7 Networking

```
  Networking  --->
    [*] Networking support
        Networking options  --->
        [ ] Packet socket
        [*] Unix domain sockets
        [ ] Transformation user configuration interface
        [ ] Transformation sub policy support (EXPERIMENTAL)
        [ ] Transformation migrate database (EXPERIMENTAL)
        [*] PF_KEY sockets
        [ ]   PF_KEY MIGRATE (EXPERIMENTAL)
        [*] TCP/IP networking
        [*]   IP: multicasting
        [ ]   IP: advanced router
        [*]   IP: kernel level autoconfiguration
        [*]     IP: DHCP support
        [ ]     IP: BOOTP support
        [ ]     IP: RARP support
        [ ]   IP: tunneling
        [ ]   IP: GRE tunnels over IP
        [ ]   IP: multicast routing
        [ ]   IP: ARP daemon support (EXPERIMENTAL)
        [*]   IP: TCP syncookie support (disabled per default)
        [ ]   IP: AH transformation
        [ ]   IP: ESP transformation
        [ ]   IP: IPComp transformation
        [ ]   IP: IPsec transport mode
        [ ]   IP: IPsec tunnel mode
        [ ]   IP: IPsec BEET mode
        [ ]   INET: socket monitoring interface
        [ ]   TCP: advanced congestion control  --->
              --- TCP: advanced congestion control
        [ ]   TCP: MD5 Signature Option support (RFC2385) (EXPERIMENTAL)
        [ ]   The IPv6 protocol
        [ ] Security Marking
        [ ] Network packet filtering framework (Netfilter)  --->
            --- Network packet filtering framework (Netfilter)
        [ ] The DCCP Protocol (EXPERIMENTAL)  --->
            --- The DCCP Protocol (EXPERIMENTAL)
        [ ] The SCTP Protocol (EXPERIMENTAL)  --->
            --- The SCTP Protocol (EXPERIMENTAL)
        [ ] The TIPC Protocol (EXPERIMENTAL)  --->
            --- The TIPC Protocol (EXPERIMENTAL)
        [ ] Asynchronous Transfer Mode (ATM) (EXPERIMENTAL)
        [ ] 802.1d Ethernet Bridging
        [ ] 802.1Q VLAN Support
        [ ] DECnet Support
        [ ] ANSI/IEEE 802.2 LLC type 2 Support
        [ ] The IPX protocol
        [ ] Appletalk protocol support
        [ ] CCITT X.25 Packet Layer (EXPERIMENTAL)
        [ ] LAPB Data Link Driver (EXPERIMENTAL)
        [ ] Acorn Econet/AUN protocols (EXPERIMENTAL)
        [ ] WAN router
```

```
  QoS and/or fair queueing  --->
     [ ] QoS and/or fair queueing
  Network testing  --->
     [ ] Packet Generator (USE WITH CAUTION)
[ ] Amateur Radio support  --->
    --- Amateur Radio support
[ ] IrDA (infrared) subsystem support  --->
    --- IrDA (infrared) subsystem support
[ ] Bluetooth subsystem support  --->
    --- Bluetooth subsystem support
[ ] RxRPC session sockets
Wireless  --->
  [ ] Improved wireless configuration API
  [ ] Wireless extensions
  [ ] Generic IEEE 802.11 Networking Stack (mac80211)
  [ ] Generic IEEE 802.11 Networking Stack
[ ]   RF switch subsystem support  --->
      --- RF switch subsystem support
[ ]   Plan 9 Resource Sharing Support (9P2000) (Experimental)  --->
      --- Plan 9 Resource Sharing Support (9P2000) (Experimental)
```

Only basic IP networking is required. DHCP is required for kernel level auto-configuration. The kernel will be able to get its IP address during boot time.

6.6.8 Device Drivers

Important to note that the Xilinx Ethernet MAC is not part of the default kernel.

```
Device Drivers  --->
  Generic Driver Options  --->
    [*] Select only drivers that don't need compile-time external firmware
    [*] Prevent firmware from being built
    [*] Userspace firmware loading support
    [ ] Driver Core verbose debug messages
    [ ] Managed device resources verbose debug messages
  [ ] Connector - unified userspace <-> kernelspace linker  --->
    --- Connector - unified userspace <-> kernelspace linker
  [*] Memory Technology Device (MTD) support  --->
    --- Memory Technology Device (MTD) support
    [ ]   Debugging
    [ ]   MTD concatenating support
    [*]   MTD partitioning support
    [*]     RedBoot partition table parsing
    (-1)      Location of RedBoot partition table
    [ ]       Include unallocated flash regions
    [ ]       Force read-only for RedBoot system images
    [ ]     Command line partition table parsing
    ---   User Modules And Translation Layers
    [*]   Direct char device access to MTD devices
    ---   Common interface to block layer for MTD 'translation layers'
    [*]   Caching block device access to MTD devices
    [ ]   FTL (Flash Translation Layer) support
    [ ]   NFTL (NAND Flash Translation Layer) support
    [ ]   INFTL (Inverse NAND Flash Translation Layer) support
    [ ]   Resident Flash Disk (Flash Translation Layer) support
    [ ]   NAND SSFDC (SmartMedia) read only translation layer
    RAM/ROM/Flash chip drivers  --->
      [*] Detect flash chips by Common Flash Interface (CFI) probe
      [ ] Detect non-CFI AMD/JEDEC-compatible flash chips
      [ ] Flash chip driver advanced configuration options
      [ ] Support for Intel/Sharp flash chips
      [*] Support for AMD/Fujitsu flash chips
      [ ] Support for ST (Advanced Architecture) flash chips
      [*] Support for RAM chips in bus mapping
      [ ] Support for ROM chips in bus mapping
      [ ] Support for absent chips in bus mapping
      Mapping drivers for chip access  --->
        [ ] Support non-linear mappings of flash chips
        [ ] CFI Flash device in physical memory map
        [ ] Map driver for platform device RAM (mtd-ram)
      Self-contained MTD device drivers  --->
        [ ] Uncached system RAM
        [ ] Physical system RAM
        [ ] Test driver using RAM
```

```
      [ ] MTD using block device
      --- Disk-On-Chip Device Drivers
      [ ] M-Systems Disk-On-Chip 2000 and Millennium (DEPRECATED)
      [ ] M-Systems Disk-On-Chip Millennium-only alternative driver (DEPRECATED)
      [ ] M-Systems Disk-On-Chip Millennium Plus
      [ ]   NAND Device Support  --->
         --- NAND Device
      [ ]   OneNAND Device Support  --->
         --- OneNAND Device Support
UBI - Unsorted block images  --->
  [ ] Enable UBI
[ ] Parallel port support  --->
  --- Parallel port support
[*] Block devices  --->
  --- Block devices
  [ ]   Normal floppy disk support
  [*]   Loopback device support
  [ ]     Cryptoloop Support
  [*]   Network block device support
  [ ]   RAM disk support
  [ ]   Packet writing on CD/DVD media
  [ ]   ATA over Ethernet support
  [*]   Xilinx SystemACE support
[ ] Misc devices  --->
  --- Misc devices
[ ] ATA/ATAPI/MFM/RLL support  --->
  --- ATA/ATAPI/MFM/RLL support
SCSI device support  --->
  [ ] RAID Transport Class
  [ ] SCSI device support
[ ] Serial ATA (prod) and Parallel ATA (experimental) drivers  --->
  --- Serial ATA (prod) and Parallel ATA (experimental) drivers
[ ] Multiple devices driver support (RAID and LVM)  --->
  --- Multiple devices driver support (RAID and LVM)
[ ] Macintosh device drivers  --->
  --- Macintosh device drivers
[*] Network device support  --->
  --- Network device support
  [ ]   Netdevice multiple hardware queue support
  [ ]   Dummy net driver support
  [ ]   Bonding driver support
  [ ]   MAC-VLAN support (EXPERIMENTAL)
  [ ]   EQL (serial line load balancing) support
  [ ]   Universal TUN/TAP device driver support
  [ ]   PHY Device support and infrastructure  --->
    --- PHY Device support and infrastructure
  [*]   Ethernet (10 or 100Mbit)  --->
    --- Ethernet (10 or 100Mbit)
    [*]   Generic Media Independent Interface device support
    [*]   Xilinx Ethernet Mac
    [ ]   PowerPC 4xx on-chip Ethernet support
  [ ]   Ethernet (1000 Mbit)  --->
    --- Ethernet (1000 Mbit)
  [ ]   Ethernet (10000 Mbit)  --->
    --- Ethernet (10000 Mbit)
  Wireless LAN  --->
    [ ] Wireless LAN (pre-802.11)
    [ ] Wireless LAN (IEEE 802.11)
  [ ]   Wan interfaces support  --->
    --- Wan interfaces support
  [ ]   PPP (point-to-point protocol) support
    [ ]   SLIP (serial line) support
    [ ]   Traffic Shaper (OBSOLETE)
    [ ]   Network console logging support (EXPERIMENTAL)
  [ ] ISDN support  --->
     --- ISDN support
   [ ] Telephony support  --->
     --- Telephony support
   Input device support  --->
     --- Generic input layer (needed for keyboard, mouse, ...)
     [ ]   Support for memoryless force-feedback devices
     [ ]   Polled input device skeleton
     ---   Userland interfaces
     [ ]   Mouse interface
```

```
[ ]   Joystick interface
[ ]   Touchscreen interface
[ ]   Event interface
[ ]   Event debugging
  ---   Input Device Drivers
[ ]   Keyboards  --->
  --- Keyboards
[ ]   Mice  --->
  --- Mice
[ ]   Joysticks/Gamepads  --->
  --- Joysticks/Gamepads
[ ]   Tablets  --->
  --- Tablets
[ ]   Touchscreens  --->
  --- Touchscreens
[ ]   Miscellaneous devices  --->
  --- Miscellaneous devices
Hardware I/O ports  --->
  [*] Serial I/O support
  [ ]   i8042 PC Keyboard controller
  [ ]   Serial port line discipline
  [ ]   PS/2 driver library
  [ ]   Raw access to serio ports
  [ ] Gameport support
Character devices  --->
  [*] Virtual terminal
  [*]   Support for console on virtual terminal
  [ ]   Support for binding and unbinding console drivers
  [ ] Non-standard serial port support
  Serial drivers  --->
    [ ] 8250/16550 and compatible serial support
      --- Non-8250 serial port support
    [*] Xilinx uartlite serial port support
    [*]   Support for console on Xilinx uartlite serial port
  [*] Unix98 PTY support
  [*] Legacy (BSD) PTY support
  (256) Maximum number of legacy PTY in use
  [ ] IPMI top-level message handler  --->
    --- IPMI top-level message handler
  [ ] Watchdog Timer Support  --->
    --- Watchdog Timer Support
  [ ] Hardware Random Number Generator Core support
  [ ] /dev/nvram support
  [ ] Generic /dev/rtc emulation
  [ ] Siemens R3964 line discipline
  [ ] RAW driver (/dev/raw/rawN)
  [ ] TPM Hardware Support  --->
    --- TPM Hardware Support
[*] I2C support  --->
  --- I2C support
  [*]   I2C device interface
  I2C Algorithms  --->
    --- I2C bit-banging interfaces
    [*] I2C PCF 8584 interfaces
    [*] I2C PCA 9564 interfaces
  I2C Hardware Bus support  --->
    [ ] MPC107/824x/85xx/52xx/86xx
    [ ] OpenCores I2C Controller
    [*] Parallel port adapter (light)
    [ ] Simtec Generic I2C interface
    [ ] TAOS evaluation module
  Miscellaneous I2C Chip support  --->
    [ ] Dallas DS1337 and DS1339 Real Time Clock (DEPRECATED)


    [ ] Dallas DS1374 Real Time Clock (DEPRECATED)
    [ ] Dallas DS1682 Total Elapsed Time Recorder with Alarm
      [ ] EEPROM reader
      [ ] Philips PCF8574 and PCF8574A
      [ ] Philips PCA9539 16-bit I/O port
      [ ] Philips PCF8591
      [ ] ST M41T00 RTC chip (DEPRECATED)
      [ ] Maxim MAX6875 Power supply supervisor
      [ ] Taos TSL2550 ambient light sensor
```

```
  [ ]   I2C Core debugging messages
  [ ]   I2C Algorithm debugging messages
  [ ]   I2C Bus debugging messages
  [ ]   I2C Chip debugging messages
SPI support  --->
  [ ] SPI support
[ ] Dallas's 1-wire support  --->
  --- Dallas's 1-wire support
[ ] Power supply class support  --->
  --- Power supply class support
[ ] Hardware Monitoring support  --->
  --- Hardware Monitoring support
Multifunction device drivers  --->
  [ ] Support for Silicon Motion SM501
Multimedia devices
  [ ] Video For Linux
  [ ] DVB for Linux
  [ ] DAB adapters
Graphics support
  [ ] Backlight & LCD device support  --->
Display device support  --->
  [ ] Lowlevel video output switch controls
  [ ] Support for frame buffer devices
  [ ] Framebuffer support for IBM GXT4500P adaptor
  Console display driver support  --->
Sound  --->
  [ ] Sound card support
[ ] HID Devices  --->
  --- HID Devices
[ ] USB support  --->
  --- USB support
[ ] MMC/SD card support  --->
  --- MMC/SD card support
[ ] LED Support  --->
  --- LED Support
[ ] EDAC - error detection and reporting (EXPERIMENTAL)  --->
  --- EDAC - error detection and reporting (EXPERIMENTAL)
[ ] Real Time Clock  --->
  --- Real Time Clock
DMA Engine support  --->
  [ ] Support for DMA engines
  --- DMA Clients
  --- DMA Devices
Userspace I/O  --->
  [ ] Userspace I/O drivers
```

All MTD options are necessary for the system ACE to be running. This is required because the compact flash attached to the System ACE will contain the root file system.

6.6.9 File Systems

```
[*] Second extended fs support
[*]   Ext2 extended attributes
[*]     Ext2 POSIX Access Control Lists
[*]     Ext2 Security Labels
[ ]   Ext2 execute in place support
[*] Ext3 journalling file system support
[*]   Ext3 extended attributes
[*]     Ext3 POSIX Access Control Lists
[*]     Ext3 Security Labels
[ ] Ext4dev/ext4 extended fs support development (EXPERIMENTAL)
[ ] JBD (ext3) debugging support
[ ] Reiserfs support
[ ] JFS filesystem support
[ ] XFS filesystem support
[ ] GFS2 file system support
[ ] OCFS2 file system support
[ ] Minix fs support
[ ] ROM file system support
[ ] Inotify file change notification support
[ ] Quota support
[*] Dnotify support
```

```
[ ] Kernel automounter support
[ ] Kernel automounter version 4 support (also supports v3)
[ ] Filesystem in Userspace support
CD-ROM/DVD Filesystems  --->
  [ ] ISO 9660 CDROM file system support
  [ ] UDF file system support
DOS/FAT/NT Filesystems  --->
  [*] MSDOS fs support
  [ ] VFAT (Windows-95) fs support
  (437) Default codepage for FAT
  [ ] NTFS file system support
Pseudo filesystems  --->
  [*] /proc file system support
  [ ]   /proc/kcore support
  [*]   Sysctl support (/proc/sys)
  [*] sysfs file system support
  [*] Virtual memory file system support (former shm fs)
  [ ]   Tmpfs POSIX Access Control Lists
  [ ] Userspace-driven configuration filesystem (EXPERIMENTAL)
Miscellaneous filesystems  --->
  [ ] ADFS file system support (EXPERIMENTAL)
  [ ] Amiga FFS file system support (EXPERIMENTAL)
  [ ] Apple Macintosh file system support (EXPERIMENTAL)
  [ ] Apple Extended HFS file system support
  [ ] BeOS file system (BeFS) support (read only) (EXPERIMENTAL)
  [ ] BFS file system support (EXPERIMENTAL)
  [ ] EFS file system support (read only) (EXPERIMENTAL)
  [*] Journalling Flash File System v2 (JFFS2) support
  (2)   JFFS2 debugging verbosity (0 = quiet, 2 = noisy)
  [*]   JFFS2 write-buffering support
  [ ]   JFFS2 summary support (EXPERIMENTAL)
  [ ]   JFFS2 XATTR support (EXPERIMENTAL)
  [ ]   Advanced compression options for JFFS2
  [ ] Compressed ROM file system support (cramfs)
  [ ] FreeVxFS file system support (VERITAS VxFS(TM) compatible)
  [ ] OS/2 HPFS file system support
  [ ] QNX4 file system support (read only)
  [ ] System V/Xenix/V7/Coherent file system support
  [ ] UFS file system support (read only)
Network File Systems  --->
  [*] NFS file system support
  [*]   Provide NFSv3 client support
  [ ]     Provide client support for the NFSv3 ACL protocol extension
  [ ]   Provide NFSv4 client support (EXPERIMENTAL)
  [ ]   Allow direct I/O on NFS files
  [ ] NFS server support
  [ ] Root file system on NFS
  [ ] Support for rpcbind versions 3 & 4 (EXPERIMENTAL)
  [ ] Secure RPC: Kerberos V mechanism (EXPERIMENTAL)
  [ ] Secure RPC: SPKM3 mechanism (EXPERIMENTAL)
  [*] SMB file system support (to mount Windows shares etc.)
  [ ]   Use a default NLS
  [ ] CIFS support (advanced network filesystem for Samba, Window and other CIFS …
  [ ] NCP file system support (to mount NetWare volumes)
  [ ] Coda file system support (advanced network fs)
  [ ] Andrew File System support (AFS) (EXPERIMENTAL)
Partition Types  --->
  [*] Advanced partition selection
  [ ]   Acorn partition support
  [ ]   Alpha OSF partition support
  [ ]   Amiga partition table support
  [ ]   Atari partition table support
  [ ]   Macintosh partition map support
  [*]   PC BIOS (MSDOS partition tables) support
  [*]     BSD disklabel (FreeBSD partition tables) support
  [ ]     Minix subpartition support
  [ ]     Solaris (x86) partition table support
  [ ]     Unixware slices support
  [*]   Windows Logical Disk Manager (Dynamic Disk) support
  [ ]     Windows LDM extra logging
  [ ]   SGI partition support
  [ ]   Ultrix partition table support
  [ ]   Sun partition tables support
  [ ]   Karma Partition support
```

```
  [ ]   EFI GUID Partition support
  [ ]   SYSV68 partition table support
Native Language Support  --->
  --- Base native language support
  (cp437) Default NLS Option
  [*]   Codepage 437 (United States, Canada)
  [ ]   Codepage 737 (Greek)
  [ ]   Codepage 775 (Baltic Rim)
  [ ]   Codepage 850 (Europe)
  [ ]   Codepage 852 (Central/Eastern Europe)
  [ ]   Codepage 855 (Cyrillic)
  [ ]   Codepage 857 (Turkish)
  [ ]   Codepage 860 (Portuguese)
  [ ]   Codepage 861 (Icelandic)
  [ ]   Codepage 862 (Hebrew)
  [ ]   Codepage 863 (Canadian French)
  [ ]   Codepage 864 (Arabic)
  [ ]   Codepage 865 (Norwegian, Danish)
  [ ]   Codepage 866 (Cyrillic/Russian)
  [ ]   Codepage 869 (Greek)
  [ ]   Simplified Chinese charset (CP936, GB2312)
  [ ]   Traditional Chinese charset (Big5)
  [ ]   Japanese charsets (Shift-JIS, EUC-JP)
  [ ]   Korean charset (CP949, EUC-KR)
  [ ]   Thai charset (CP874, TIS-620)
  [ ]   Hebrew charsets (ISO-8859-8, CP1255)
  [ ]   Windows CP1250 (Slavic/Central European Languages)
  [ ]   Windows CP1251 (Bulgarian, Belarusian)
  [*]   ASCII (United States)
  [*]   NLS ISO 8859-1  (Latin 1; Western European Languages)
  [ ]   NLS ISO 8859-2  (Latin 2; Slavic/Central European Languages)
  [ ]   NLS ISO 8859-3  (Latin 3; Esperanto, Galician, Maltese, Turkish)
  [ ]   NLS ISO 8859-4  (Latin 4; old Baltic charset)
  [ ]   NLS ISO 8859-5  (Cyrillic)
  [ ]   NLS ISO 8859-6  (Arabic)
  [ ]   NLS ISO 8859-7  (Modern Greek)
  [ ]   NLS ISO 8859-9  (Latin 5; Turkish)
  [ ]   NLS ISO 8859-13 (Latin 7; Baltic)
  [ ]   NLS ISO 8859-14 (Latin 8; Celtic)
  [ ]   NLS ISO 8859-15 (Latin 9; Western European Languages with Euro)
  [ ]   NLS KOI8-R (Russian)
  [ ]   NLS KOI8-U/RU (Ukrainian, Belarusian)
  [ ]   NLS UTF-8
Distributed Lock Manager  --->
  [ ] Distributed Lock Manager (DLM)
```

As much as possible is deselected to make the kernel smaller.

6.6.10 IBM 40x options

```
IBM 40x options  --->
```

6.6.11 Library routines

```
CRC-CCITT functions
[ ] CRC16 functions
[ ] CRC ITU-T V.41 functions
--- CRC32 functions
[ ] CRC7 functions
[*] CRC32c (Castagnoli, et al) Cyclic Redundancy-Check
```

6.6.12 Profiling Support

```
[ ] Profiling support (EXPERIMENTAL)
```

6.6.13 Kernel Hacking

```
[*] Show timing information on printks
[ ] Enable __must_check logic
[ ] Magic SysRq key
[ ] Enable unused/obsolete exported symbols
[ ] Debug Filesystem
[ ] Run 'make headers_check' when building vmlinux
```

```
[*] Kernel debugging
[ ]   Debug shared IRQ handlers
[ ]   Detect Soft Lockups
[ ]   Collect scheduler debugging info
[ ]   Collect scheduler statistics
[ ]   Collect kernel timers statistics
[ ]   Debug slab memory allocations
[ ] RT Mutex debugging, deadlock detection
[ ] Built-in scriptable tester for rt-mutexes
[ ] Spinlock and rw-lock debugging: basic checks
[ ] Mutex debugging: basic checks
[ ] Spinlock debugging: sleep-inside-spinlock checking
[ ] Locking API boot-time self-tests
[ ] kobject debugging
[ ] Verbose BUG() reporting (adds 70K)
[*] Compile the kernel with debug info
[ ] Debug VM
[ ] Debug linked list manipulation
[ ] Force gcc to inline functions marked 'inline'
[ ] Fault-injection framework
[ ] Include kgdb kernel debugger
[ ] Include xmon kernel debugger
[*] Include BDI-2000 user context switcher
[ ] Support for early boot texts over serial port
```

These options make it possible to control the internal process using the JTAG/USB cable.

6.6.14 Security options

```
Security options  --->
  [ ] Enable access key retention support
  [ ] Enable different security models
```

6.6.15 Cryptographic API

```
[ ] Cryptographic API  --->
  --- Cryptographic API
```

6.7 Compiling the Kernel

Make sure that the cross-compiler is in your path. The path relative to the cross-compiler directory is: `<Compilation root>/gcc-3.4.6-glibc-2.3.6/powerpc-405-linux-gnu/bin` in case you generated the gcc compiler version 3.4.6 with glibc version 2.3.6.

6.7.1 Building the image

Once the changes have been performed, you have to make sure that the cross compiler is in your path and type:

```
$ make zImage
```

There are no problems when building the image as opposed to Linux 2.4.

7 Creating the ACE file

The following section describes what to do in the Windows operating system.

To boot the image you have to create an ACE file which contains HW image (bit-file) and the kernel. A simple approach is to copy kernel image and bit-file into the same sub-directory, e.g. locate it in "`C:\boot405`". The location of the kernel image is described at the end of the previous chapter. The bitstream is located in the "`<DESIGN_ROOT>/implementation/download.bit`" file.

First create an option file (e.g. `genace.opt`) with the following content:

```
-jprog
-board ml405
```

```
-hw ./download.bit
-elf ./zImage.elf
-ace system.ace
```

Provided that your board is an ml405 board and the name of the Bitstream and your kernel is `download.bit` and `zImage.elf` respectively.

The generation of the ACE file is best carried out by using the "`cygwin`" tool which can be downloaded from the www.cygwin.com site or by using the built-in "`cygwin`" in EDK. If you install it yourself, ensure that you have selected the DOS file format in the beginning of the installation process.

First, you MUST make sure that `powerpc-eabi-objdump.exe` is in your path. "cygwin" is an excellent tool to verify it, just type:

$ `which powerpc-eabi-objdump.exe`

and the executable will be printed with a full path, if it is in your path. Otherwise you will have to add it. Depending on your installation, it is likely to be located in:

```
$XILINX_EDK/gnu/powerpc-eabi/nt/bin
```

If so, put it into your path.

You can now create the ACE file by typing:

```
$ xmd -tcl genace.tcl -opt genace.opt
```

The output from the process shall look as follows (some blank lines are removed):

```
CASI@5ycrj2j /cygdrive/c/boot405
$ xmd -tcl genace.tcl -opt genace.opt
Xilinx Microprocessor Debug (XMD) Engine
Xilinx EDK 9.1.02 Build EDK_J_SP2.4
Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.
Executing xmd script : C:/TOOLS/EDK/9_1/data/xmd/genace.tcl

######################################################################
XMD GenACE utility. Generate SystemACE File from bit/elf/data Files
######################################################################
Using GenACE option file : genace.opt
Target Type not defined for Processor 1, Using default Target Type ppc_hw
GenACE Options:
        Board      : ml405
        Jtag Devs  : xcf32p xc4vFx20 xc95144xl
        FPGA pos   : 2
        JPROG      : true
        HW File    : ./download.bit
        ACE File   : system.ace
        nCPUs      : 1

        Processor ppc_hw_1 Information
                Debug opt : -debugdevice devicenr 2 cpunr 1
                ELF files : ./zImage.elf
                Start PC Address : 0x00400000
##########################################################
Converting Bitstream './download.bit' to SVF file './download.svf'
Executing 'impact -batch bit2svf.scr'
Copying ./download.svf File to  system.svf File
JTAG chain configuration
-----------------------------------------------
Device   ID Code        IR Length    Part Name
```

```
 1       05059093          16        xcf32p
 2       01e64093          10        xc4vFx20
 3       09608093           8        xc95144xl

PowerPC405 Processor Configuration
----------------------------------
Version............................0x20011430
User ID............................0x00000000
No of PC Breakpoints................4
No of Read Addr/Data Watchpoints....1
No of Write Addr/Data Watchpoints...1
User Defined Address Map to access Special PowerPC Features using XMD:
        I-Cache (Data).........0x70000000 - 0x70003fff
        I-Cache (TAG)..........0x70004000 - 0x70007fff
        D-Cache (Data).........0x78000000 - 0x78003fff
        D-Cache (TAG)..........0x78004000 - 0x78007fff
        DCR....................0x78004000 - 0x78004fff
        TLB....................0x70004000 - 0x70007fff
Copying ./zImage.svf File to  system.svf File
############################################################
Writing Processor JTAG "continue" command to SVF file 'sw_suffix.svf'

JTAG chain configuration
-----------------------------------------------------
Device    ID Code        IR Length    Part Name
 1       05059093          16        xcf32p
 2       01e64093          10        xc4vFx20
 3       09608093           8        xc95144xl

PowerPC405 Processor Configuration
----------------------------------
Version............................0x20011430
User ID............................0x00000000
No of PC Breakpoints................4
No of Read Addr/Data Watchpoints....1
No of Write Addr/Data Watchpoints...1
User Defined Address Map to access Special PowerPC Features using XMD:
        I-Cache (Data).........0x70000000 - 0x70003fff
        I-Cache (TAG)..........0x70004000 - 0x70007fff
        D-Cache (Data).........0x78000000 - 0x78003fff
        D-Cache (TAG)..........0x78004000 - 0x78007fff
        DCR....................0x78004000 - 0x78004fff
        TLB....................0x70004000 - 0x70007fff

Info:Processor started. Type "stop" to stop processor

RUNNING>
############################################################
Converting SVF file 'system.svf' to SystemACE file 'system.ace'
Executing 'impact -batch svf2ace.scr'

SystemACE file 'system.ace' created successfully
```

The errors most likely to occur are path problems such as the `powerpc-eabi-objdump.exe` file not being located in any directory of your path. It can be verified if you check that the kernel starting point is found. Search Start PC Counter - if so the path is correct (with respect to the `powerpc-eabi-objdump.exe` file).

The result from this process is a `system.ace` file. Depending on the board type, the file should be placed in the `ml405/myace` or the `ml403/myace` sub-directories in the compact flash drive.

8 The first boot Attempt

First attempt is to verify that everything performed so far is correct by doing as follows:

- Place the compact flash drive in the slot of the board.
- Connect your PC with the ML403/ML405 board.

- Start a terminal emulation programme on your PC, set the bit rate at 9600, 8 bits frame with no parity and with no flow control.
- Optionally connect Xilinx Platform Cable USB to the board.

Start the board either by using the terminal or the push buttons on the board. Select the "`My own ACE file`" option.
The following output should now occur in your terminal window.

```
Rebooting to System ACE Configuration Address 6...

loaded at:     00400000 004D61E0
board data at: 004D3138 004D3150
relocated to:  00405310 00405328
zimage at:     004058D5 004D2848
avail ram:     004D7000 04000000

Linux/PPC load: console=ttyUL0,9600 root=/dev/xsa5 rw ip=on
```

The latter line is the bootloader command you entered during the kernel configuration; you can either:

- Alter it, if it is not correct.
- Press `<Enter>` to continue.
- Or wait 5 seconds, the system will assume that you wish to continue.

If nothing else happens, it is very likely that you have entered the wrong terminal options during kernel configuration, or that the RS232 options you entered during HW configuration do not match.

Any mismatch of these options will cause the RS232 interface to fail.

Even if no output occurs, it is likely that your kernel does run. However, due to the lack of an interface, you are not able to verify it for sure.

Other problems are that the `/dev/xsa5` does not exist (the most likely error), or that the network connection fails which is indicated by that fact that no DHCP request is answered. In the latter case, you can either connect an Ethernet Cable (a prerequisite that you have a running DHCP server), or your can postpone the DHCP configuration by removing the `ip=on` bootloader option.

9 Creating a root file system

The following is described as it should be performed in the Linux Operating System.

9.1 Creating a usable file system

First we advise you to get a new Compact Flash device larger than 512 Mbytes (e.g. 2.0 GB) and to copy the partitions from the original Compact Flash device to the new device. The remaining space should be used to create partition 5 which shall contain the root file system.

The above can be achieved by performing the actions and commands specified below:

- Insert original flash card and determine its device id via `dmesg` or by reading `/var/log/messages`.
- Copy the flash image to the harddrive.

    - `dd if=/dev/<flash card device id> of=flash.img`
- Remove the flash card and insert clean/empty one in intead (Use the 2GB flash card mentionend earlier).
    - Again determine the flash card id via `dmesg` or by reading `/var/log/messages`.
- Copy the image file `flash.img` to the newly inserted flash card.
    - `dd if=flash.img of=/dev/<flash card device id>`
- The next part is to add a partition occupying the remainder of the flash disk. This can be carried out with the partitioning programme `fdisk.` on `/dev/<flash card device id>`
    - `fdisk /dev/<flash card device id>`
    - Create an extended partition
        - `press n <enter>`
        - `press e <enter> <enter>`
        - `press 3 <enter> <enter> <enter>`
            - Partition 3 has now become an extended partition which occupies the remainder of the disk.
    - Create a logical partition that occupies the entire extended partition.
        - `Press n <enter>`
        - `Press l <enter> <enter> <enter>`
            - Note that since this is the first logical partition, it will gain the partition number 5.
- The last thing to perform is to format the newly created partition with your favourite file system.
    - `mkfs.ext3 /dev/<flash device>5`

9.2 What does the kernel expect?

Several files are required during the boot. The two most important files are init, the `<inittab>` files and the RC files.

9.2.1 Init

Init is the mother of all processes. Init is started first and will have the lowest process ID. Stopping init means stopping the entire system. The system will not start if no init executable is present, and it will perform a "kernel panic".

9.2.2 Inittab

The `<inittab>` file is a configuration file for the Init process (the mother of all Linux Processes). It contains instructions of what to do in case of pressing **`<CTRL>+<ALT>+<Delete>`** and how to create the terminals used by kernel to communicate with the outside world.

The contents of a sample Inittab file could look like this:

```
::sysinit:/etc/init.d/rcS
::askfirst:-/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/sbin/swapoff -a
::shutdown:/bin/umount -a -r
::restart:/sbin/init
::respawn:/sbin/getty 9600 ttl/0
```

In some cases you may have problems in using a tty and need to change the latter line to:

```
::respawn:/bin/ash 9600
```

9.2.3 RC files

The RC files contain instructions of how to start the different services after init has been started. They do not take part in the kernel boot and are therefore not considered part of the basic kernel. The content of the RC files will not be discussed further in this document.

9.3 Generating a Full system

The generation of a full system is a very time-consuming process - although it is the most flexible approach. As the main topic of this document is the Linux kernel itself, we will present a much simpler approach by using Busybox.

9.4 Using Busybox and mkrootfs

Busybox is a simple one-binary system. It means that all functionality of the most commonly used binaries is placed in one binary. All other binaries are emulated by symbolic links to the BusyBox binary.

The advantages are that you can configure the Busybox binary once and copy it to the root file system and thereby make the proper links. You are ready to run!

The binaries for busybox can be found at the site http://www.busybox.net/.

The mkrootfs.sh is a script written by the Klingaufs, located at the site http://www.klingauf.de.

The mkrootfs script uses the busybox binaries and you therefore have to start with busybox.

9.4.1 How to download and configure busybox

Go to the http://www.busybox.net/. Download the source by clicking on the BusyBox 1.7.1. link (NOTE the version number may be different as busybox constantly evolves). Download it to a place accessible from either Linux or Cygwin as busybox is intended to be compiled from one of them. We recommend Linux for BusyBox compilation.

The downloaded file is a 'tar'-file that can be unpacked by using the following command (in Linux):

```
$ tar xvf busybox-1.7.1.tar.tar
```

The result is a complete kernel-like directory structure.
First you have to make sure that you really do cross-compile. There are several approaches, and we recommend that you alter the make file and ensure that the following options are set:

```
ARCH            := ppc
CROSS_COMPILE   :=powerpc-405-linux-gnu-
```

Then you can always be sure that the cross-compiler will compile the correct host.

To configure busybox type

```
$ make menuconfig
```

Most default settings are OK. However make sure that you have

- Added init support
- Let the install directory point to `../mkrootfs/_install` if you wish to use mkroofts
- Made sure that the following applets are turned ON:

```
Login/Password Management Utilites ->
   getty
   passwd
   su
   sulogin

Networking Utilities->
   inetd
   ifconfig
   netstat
   nslookup
   telnet
   telnetd
```

Having configured busybox, just type make, and the binary will be built. Try the following command to ensure that your busybox is correctly configured:

```
$file ./busybox
```

If the answer is something like

```
busybox: ELF 32-bit MSB executable, PowerPC or cisco 4500, version 1 (SYSV), for
GNU/Linux 2.4.3, dynamically linked (uses shared libs), for GNU/Linux 2.4.3,
stripped
```

you can turn your attention to mkrootfs.

9.4.2 How to download and configure mkrootfs

The script mkrootfs is an approach which can save you a LOT of time.

The original mkrootfs file can be downloaded from:

www.klingauf.de/v2p/index.phtml

Check out section 6.2 on the HTML page in which there is a link to mkroofs (the link is somewhat disguised).

The script uses some sh-script features which may not be supported.

Another approach is to use our script which can be downloaded from the website:

www.teknologisk.dk/22523

It is a modified script using simpler sh-script features.

10 Two Simple Applications

10.1 Creating an SSH server

A simple SSH server is dropbear which we have used to test the Linux system. The dropbear sources can be obtained from

http://matt.ucc.asn.au/dropbear/

The tarballs (.tgz – files) containing the sources can be found in this directory. Download version 0.50, it is the version we have tested. Place the downloaded file somewhere accessible from your Linux Box.

Having downloaded the tarball, it has to be unpacked by using the tar command:

```
$ tar xvzf dropbear-0.50.tar.gz
```

This command will cause the dropbear build directory to be created.

Finally enter the directory:

```
$ cd dropbear-0.50
```

10.1.1 Building the SSH server

To build the dropbear you have to configure it. Use the command:

```
$ ./configure --host=powerpc-405-linux-gnu  -disable-zlib
```

followed by:

```
$ make
```

After a while the dropbear is built but scp should also be created:

```
$ make scp
```

The resulting files:

- scp
- dropbear
- dropbearkey

should be placed in the `/bin` directory in your embedded Linux system.

10.1.2 Configuring the SSH server

First the keys have to be generated by using the following commands (the portion you shall type in is in boldface):

```
# dropbearkey -t rsa -f dropbear_rsa_host_key
Will output 1024 bit rsa secret key to 'dropbear_rsa_host_key'
Generating key, this may take a while...
```

```
Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAAgwCjAisRsS0RzCFCZl7Q9vDCiMk/cOyMBXJ5paDdJEyY
y2DaVrpCqir9tIoIMfaFESuR+Q5iX7NkmdKX1oRiVWg2inVUeKMrr7KXqMuodMYGL5Uni9Xp3vcy/S2Q
J3nH0nT4Bpc0Gd7XhY+mBkHWRQTyIOqQSRYJ4XmedMTE0xgGVtqF root@10.16.4.126
Fingerprint: md5 ba:d0:d5:2d:b9:e7:ea:5d:84:d4:c2:b8:85:72:40:de
# dropbearkey -t dss -f dropbear_dss_host_key
Will output 1024 bit dss secret key to 'dropbear_dss_host_key'
Generating key, this may take a while...
Public key portion is:
ssh-dss AAAAB3NzaC1kc3MAAACBANsCjR9KV/a2N3YfNA4/rsep54LDgaY9nQjzUiptchSuuH5SUUM8
XWs/Z4QbOZw860nFXtMOOnetQWXBecxgmNMdeHNELwOCJI/TY109iOAKAu6HYWaZRwZQKPp7FHPfDU1n
ScVFxQacKJpiTyXMo9Cqkz+fncIOnI6nR94hDwvHAAAAFQCm25qOMwgK2SxJ/6NxrTB+ic6mxQAAAIEA
qGDLbkJgddIHYFtex2VSsUV3E8qmLDP7nl8IHLKaaiOVZj485HP4Pjsro9GvuEbOu/K7H9MYi5cZmHxj
gMgc0gJEEM4D6OBgE8jBDtdB6vVo4hbSDYOCqXX5DbZcwQT9Tt0/bW6hlmxiLIBO5i4m11LCMlm2ag7G
qOqvqNnI79QAAACBAIYfS9s+OScJnFjSm+uzhsVvK+TYCTJnfTWJsqHfqoqaElENgD3sSWFc7gJN9kpf
smC+ToQuNYA/mfQSIqwCznf8GLxuwbJ+BVczQgsSsmqFqWdu0gdVxyRaKNS17SoL2+zkjB+AnXTPT8ju
Cws/W7gY0DqTkHWp/V4oWuOLcSxY root@10.16.4.126
Fingerprint: md5 8d:ca:45:82:48:a6:9c:4b:83:08:21:a3:b3:63:e2:8d
#
```

Now move the keys to the /etc/dropbear directory:

```
# mkdir /etc/dropbear
```

(In this case mv applets has not been compiled into the busybox binary)

```
# cp dropbear_dss_host_key /etc/dropbear/.
# rm dropbear_dss_host_key
# cp dropbear_rsa_host_key /etc/dropbear/.
# rm dropbear_rsa_host_key
#
```

Or you may wish to generate the key files directly into the correct location.
Hereafter you simply type

```
# dropbear
```

or

```
# dropbear -E
```

in order to get the SSH-deamon running. The '–E'-option will force the debugging message to the stdout device which may come in handy if you wish to debug something.

Now ensure that the shell used is a valid SSH shell. They are listed in the /etc/shells file. If ash is used, insert the following text into /etc/shells:

```
/bin/ash
```

Otherwise the dropbear ssh deamon will not accept any connection.

It is now possible to connect to the Linux kernel from any PC (equipped with a proper SSH client) to your Linux.

Furthermore "scp" - which also operates on port 22 - will function.

10.2 Creating an HTTPD server

This chapter has only been verified to work correctly on the ML403 board.

The HTTPD server tested is the built-in HTTPD applet of BusyBox. To build the server, enable the HTTPD option in the BusyBox configuration.

To start the HTTPD server, type:

```
$ httpd
```

The HTTP file root is then ".", also known as the directory in which the HTTPD was started. In this directory you must have a file denoted `index.html` – this file will be returned to the HTTP browser when requesting the URL of the board. If your board was given the address 10.16.4.171 then try the following URL:

http://10.16.4.171

to verify if the index.html file is returned to you.

11 References

[1] Building Embedded LINUX SYSTEMS; Concepts, Techniques, Tricks, and Traps. Karim Yaghmour. O'Reilly 2003, ISBN-13 978-0-596-00222-0. Though written in 2003 this book is creating a good starting point for the first-time tool chain builder. Concepts and different hardware are explained in adequate detail so that you can build your tool chain, the kernel, and the embedded system.

[2] (From I/O Ports to process Management) Understanding the Linux Kernel, "3rd Edition, covers version 2.6. Daniel P. Bovet & Marco Cesati. This (very) comprehensive book covers the kernel from above, i.e. where Rubinis Device driver book covers the device drivers, this book is lifted above the driver level into the kernel.

[3] (Where the Kernel meets the Hardware) Linux Device Drivers "3rd Edition" Jonathan Corbet, Alessandro Rubini and Greg Kroah-Hartman. O'Reilly 2005, ISBN-13 978-0-596-00590-0. This book is the third edition in a series of device drivers originally initiated by Rubini. It is known as "The Book" if you wish to write device drivers.