

Avnet Virtex-5 FXT Evaluation Board Executing From Flash Reference Design



**Version 1.0
July 2008**

1 Introduction

This document describes a simple PowerPC design that executes code directly from the on-board parallel Flash. The design illustrates the use of Xilinx Micro-Kernel (XMK) operating system. Please refer to the “Using Xilkernel” chapter of the Platform Studio User Guide for information on the Xilinx Micro-Kernel operating system.

2 Reference Design Requirements

This reference design will require the following software and hardware setups.

2.1 Software

The software requirements for this reference design are:

- Windows XP
- Xilinx ISE 10.1 with Service Pack 2
- Xilinx EDK 10.1 with Service Pack 2

2.2 Hardware

The hardware setup for this reference design is:

- Computer with 1 GB RAM and 1 GB virtual memory (recommended)
- Avnet Virtex-5 FXT evaluation board
- Straight through RS232 cable
- Power supply
- JTAG programming cable (USB or PC4)

3 Reference Design Block Diagram

The following figure shows a high-level block diagram of the reference design. The design consists of:

- PowerPC Processor
- 32KB of BRAM
- 32MB of Flash
- Timer and Interrupt Controller
- RS232 Port

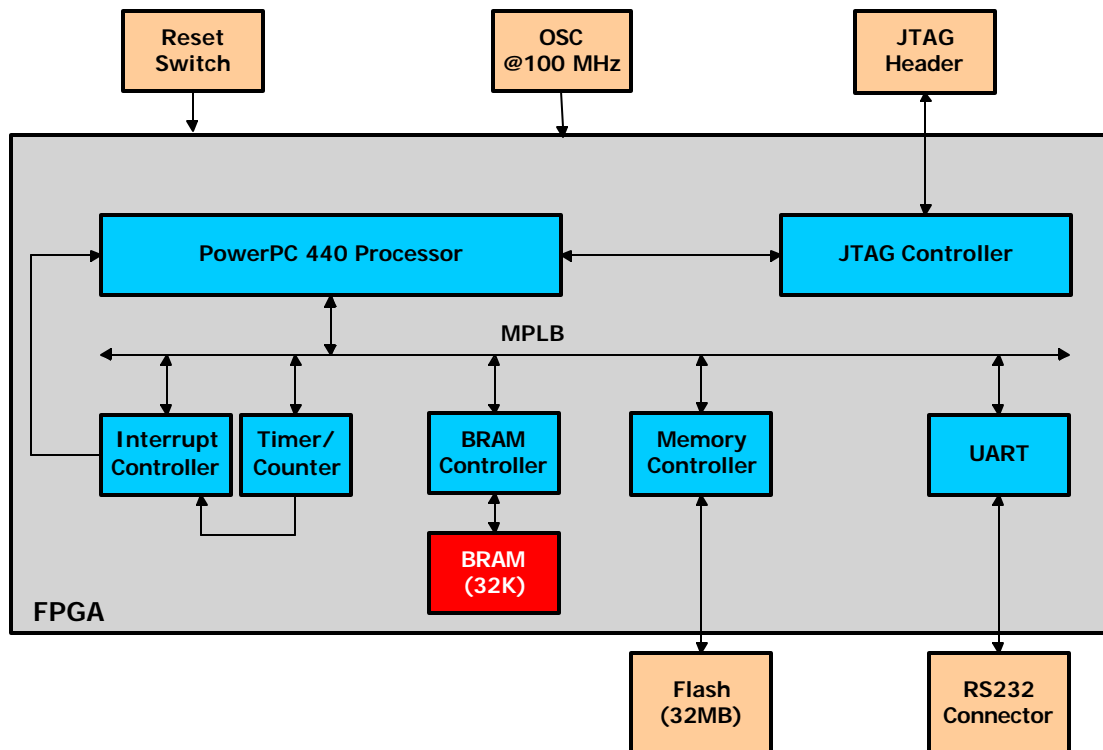


Figure 1 – Reference Design Block Diagram

4 MB XMK Design Software

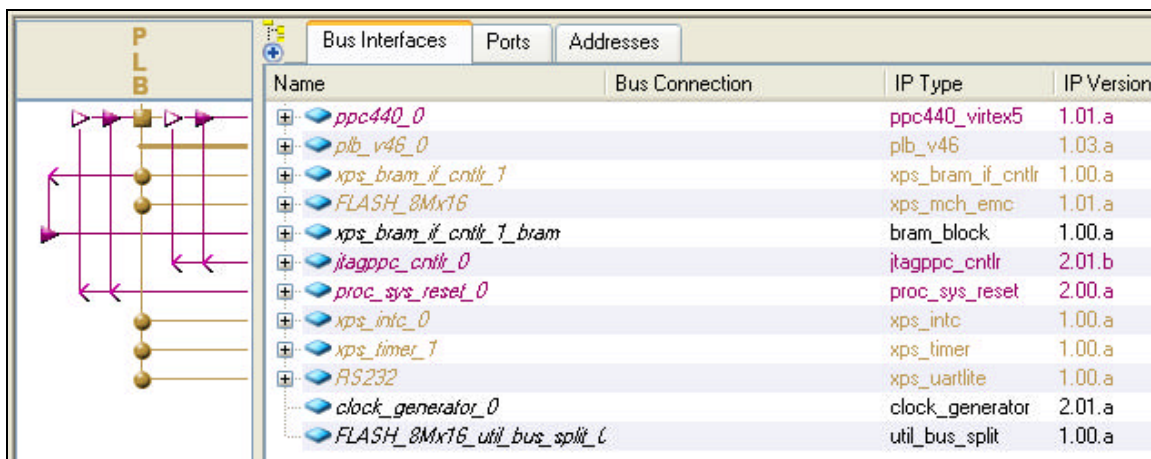
The MB XMK software source code is located in the `/xilkernel_demo` folder of the project directory. The MB XMK software consists of the following threads:

Thread	Description
<i>shell</i>	This is the main controlling thread and presents a shell with a few simple commands from which you can launch the other demo threads.
<i>prodcon</i>	Producer consumer example thread(s) using message queues.
<i>llist</i>	Linked list demo using the buffer memory allocation interfaces.
<i>sem</i>	Semaphore example showing multiple competing threads using semaphores to coordinate.
<i>TicTacToe</i>	Simple tic-tac-toe game, which illustrates how to dynamically assign stack memory to a thread when creating it.
<i>TimerTest</i>	Simple time management demo.
<i>prio</i>	Thread illustrating dynamically changing priorities and priority queues in the kernel structures.
<i>mutex</i>	Mutex demo, illustrating pthread mutex locks.
<i>clock</i>	Simple thread, using the second timer device and handling interrupts from it, to keep track of wall-clock time. This illustrates user-level interrupt handling.
<i>standby</i>	Simple standby thread.

The XMK application code for this reference design will reside in the on-board Flash and internal Block RAM. Specifically, the read/write data portion of the application will reside in BRAM while the read-only data and the code will reside in the on-board Flash.

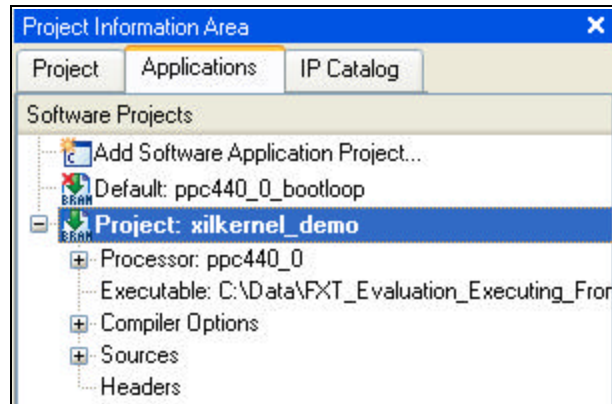
5 Linking and Compiling the Application Code

1. Open the design in XPS and you should see system view of the project as shown in the following figure.

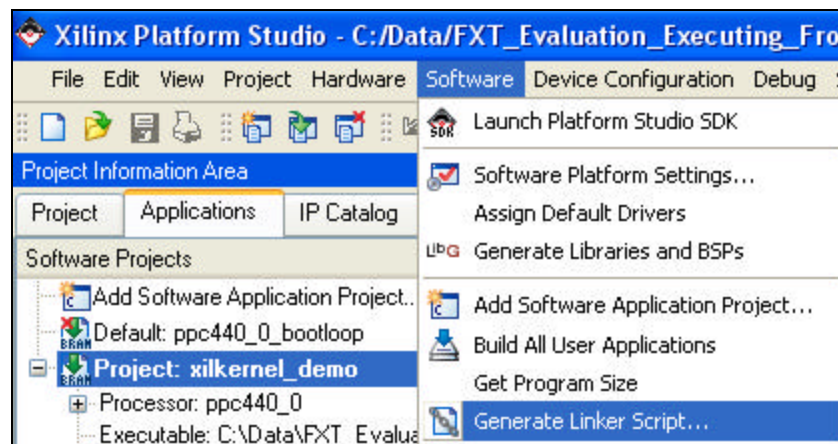


2. Click on the **Applications** tab to view the software project associated with this design. As shown in the following figure, the **xilkernel_demo** software project is marked to be initialized in the FPGA BRAM.

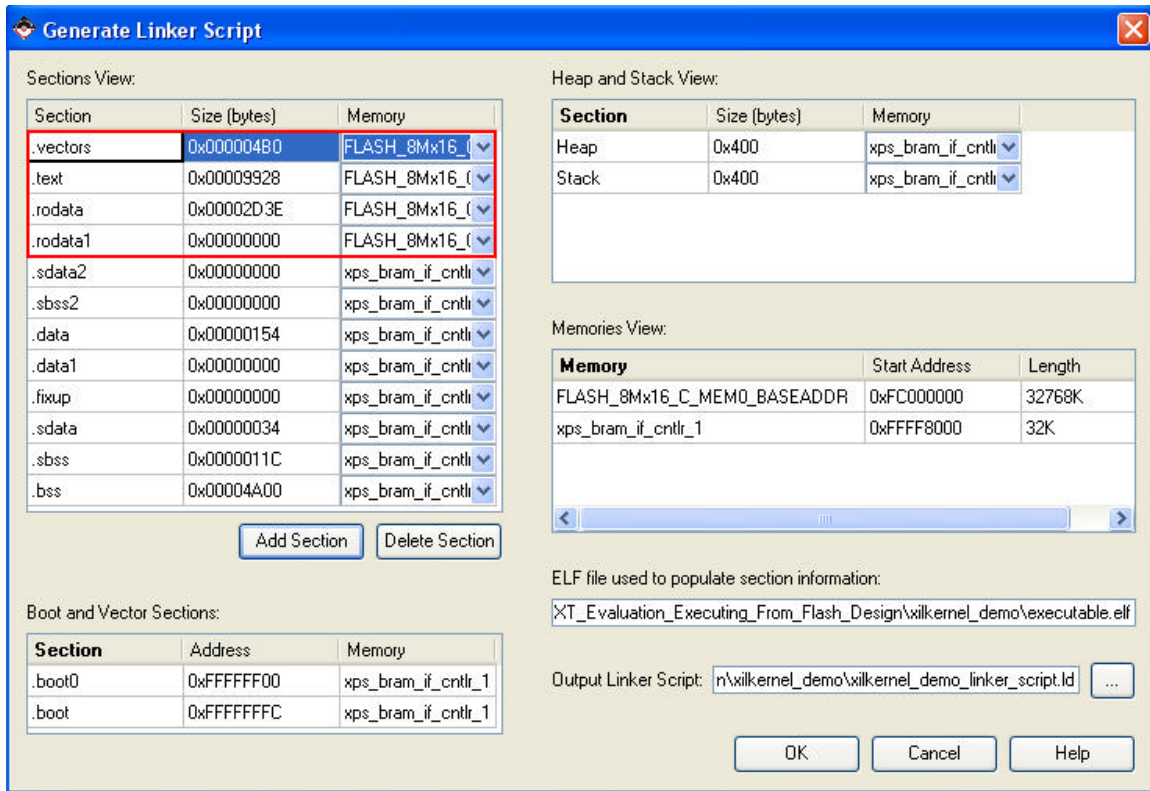
Since the software for this application will reside in the on-board Flash as well as the FPGA BRAM, the “**Mark to Initialize BRAMs**” for the **xilkernel_demo** software project will store the read/write data portion of the application software in the FPGA BRAM when the FPGA is configured. The read-only portion of the data along with the code must be programmed into the on-board Flash.



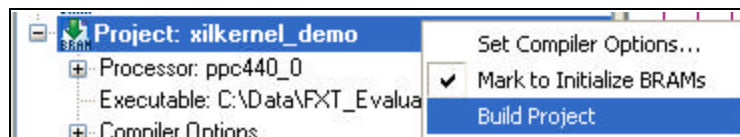
3. Select **Software > Generate Linker Script...** from the XPS GUI to view the linker script for the **xilkernel_demo** software project as shown in the following figure.



- The linker script dialog box will appear as shown in the following figure. The **.vectors**, **.text** (code), **.rodata** (read-only data), and the **.rodata1** sections are mapped to the on-board Flash, while the remaining sections, including the Stack and Heap areas are mapped to the FPGA BRAM. Click **Cancel** to continue.

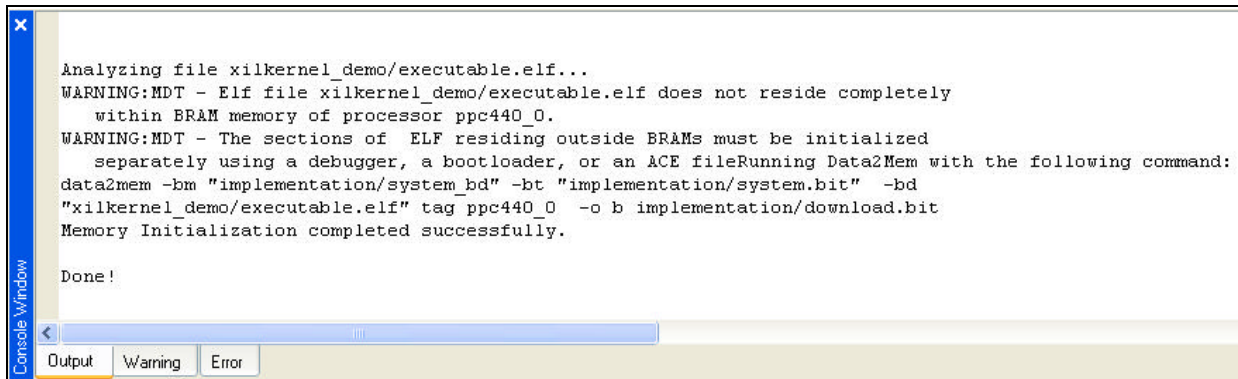


- Right-click on the **xilkernel_demo** software project and select **Build Project** to compile the application software and generate the **executable.elf** file (this file will be stored in the **/xilkernel_demo** folder of the project).



6 Implementing and Running the Design

1. Select **Device Configuration > Update Bitstream** from the XPS GUI to build the design and initialize the BRAM. As shown in the following figure, you will get a warning message stating that the entire code and data cannot be placed in the FPGA BRAM. This is a valid warning message as the code section as well as the read-only data will be placed in the on-board Flash.



The screenshot shows the XPS Console Window with the following text:

```
Analyzing file xilkernel_demo/executable.elf...
WARNING:MDT - Elf file xilkernel_demo/executable.elf does not reside completely
within BRAM memory of processor ppc440_0.
WARNING:MDT - The sections of ELF residing outside BRAMs must be initialized
separately using a debugger, a bootloader, or an ACE fileRunning Data2Mem with the following command:
data2mem -bm "implementation/system_bd" -bt "implementation/system.bit" -bd
"xilkernel_demo/executable.elf" tag ppc440_0 -o b implementation/download.bit
Memory Initialization completed successfully.

Done!
```

The console window has tabs for Output, Warning, and Error. The Warning tab is currently selected.

7 Setting Up the Board

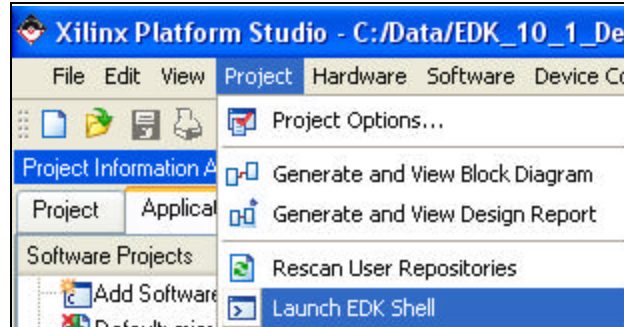
Perform the following steps to setup the board for running the application software.

1. Verify the Power switch, **SW7**, is in the **OFF** position.
2. Install a jumper on JP3 pins 2-3
3. Install a jumper on JP2 pins 2-3
4. Install a jumper on JP1 pins 1-2
5. Install a jumper on JP5 pins 2-3 (FPGA JTAG mode)
6. Connect the power supply to the J11 connector on the FXT evaluation board and also plug it into the AC outlet.
7. Connect the USB JTAG cable to J9 and the USB port of the PC.
8. Connect a straight through RS232 cable to the board DB-9 connector (P1) and the serial port of the PC. Alternatively, you can use an RS232-USB adapter and connect this adapter to the DB-9 connector and the USB port of the PC. In this case, you must install the RS232-USB driver for the adapter.
9. Slide the power switch to the **ON** position
10. Start a Hyper Terminal session and set the serial port parameters to 19200 baud rate, 8 bits, 1 stop bit, no parity and no flow control.

8 Generating the Flash Binary File

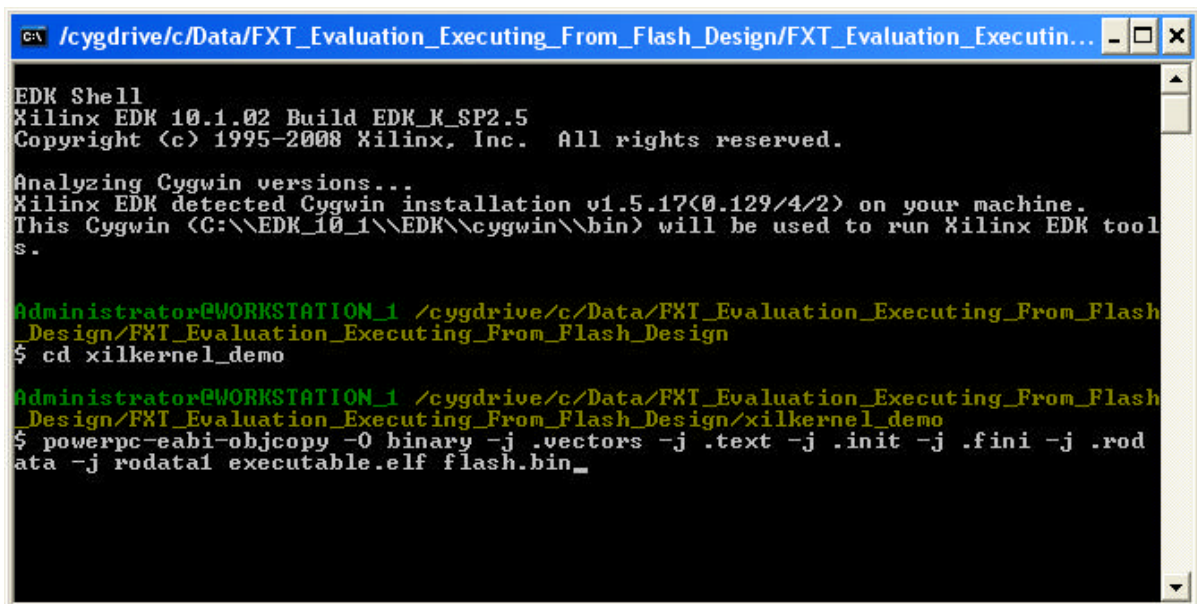
Prior to programming the Flash, the **executable.elf** file must be converted to a binary file that contains the read-only and code sections of the program leaving out the sections that are stored in the FPGA BRAM.

1. Launch the EDK Shell command window by selecting **Project > Launch EDK Shell** from the XPS GUI as shown in the following figure.



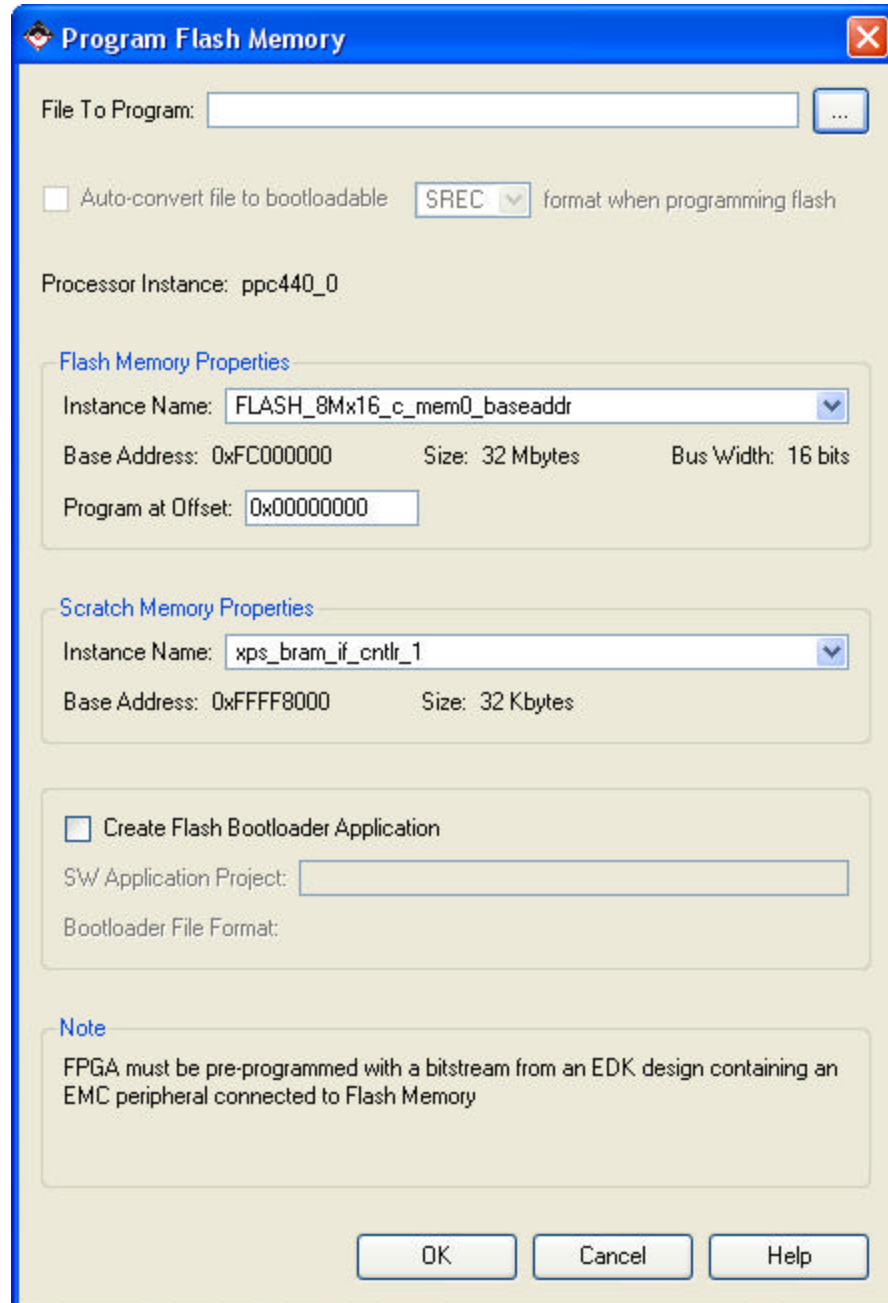
2. At the EDK Shell prompt, enter the following lines hitting the return key after each line as shown in the following figure. This will generate a file called **flash.bin** in the **/xilkernel_demo** folder that contains the **.text**, **.init**, **.fini**, and **.rodata** sections of the application software.

```
cd xilkernel_demo
powerpc-eabi-objcopy -O binary -j .vectors -j .text -j .init -j .fini -j .rodata -j
rodata1 executable.elf flash.bin
```



9 Programming the Flash

1. Select **Device Configuration > Download Bitstream** from the XPS GUI to download the design to the board.
2. Select **Device Configuration > Program Flash Memory** from the XPS GUI, the **Program Flash Memory** dialog box will appear as shown in the following figure.



3. Please set the following parameters on the **Program Flash Memory** dialog box:
 - a. Under the **File to Program**, browse to the **/xilkernel_demo** folder and select **flash.bin** file (you need to select show all file types to see this file as the search window defaults to **.elf** files).
 - b. Under the **Scratch Memory Properties**, use the drop-down box and select **xps_bram_if_cntlr_1**.
 - c. The **Program Flash Memory** dialog box should look as shown in the following figure. Click **OK** to continue and program the Flash.

Program Flash Memory

File To Program: ...

☐ Auto-convert file to bootloadable format when programming flash

Processor Instance: ppc440_0

Flash Memory Properties

Instance Name:

Base Address: 0xFC000000 Size: 32 Mbytes Bus Width: 16 bits

Program at Offset:

Scratch Memory Properties

Instance Name:

Base Address: 0xFFFF8000 Size: 32 Kbytes

☐ Create Flash Bootloader Application

SW Application Project:

Bootloader File Format:

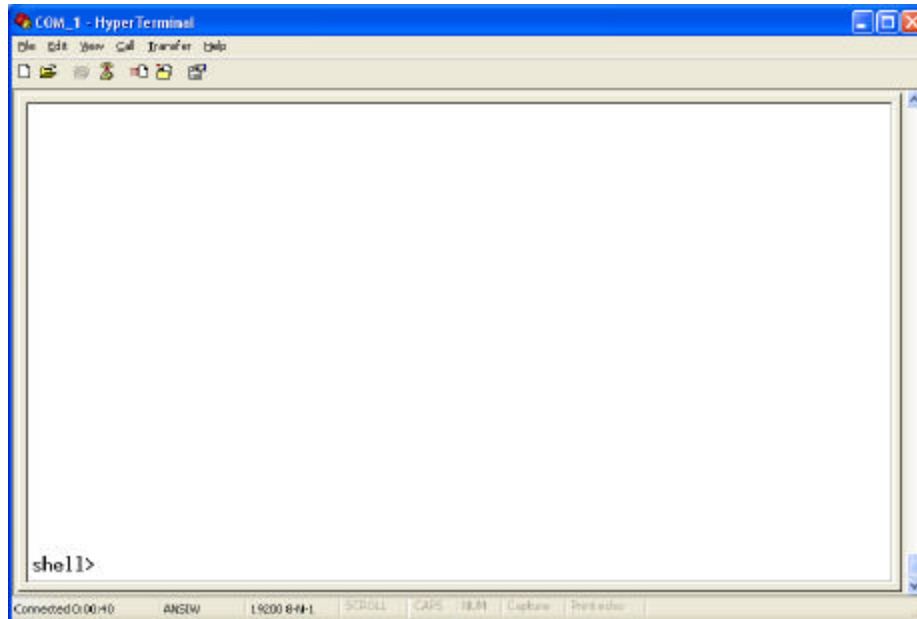
Note

FPGA must be pre-programmed with a bitstream from an EDK design containing an EMC peripheral connected to Flash Memory

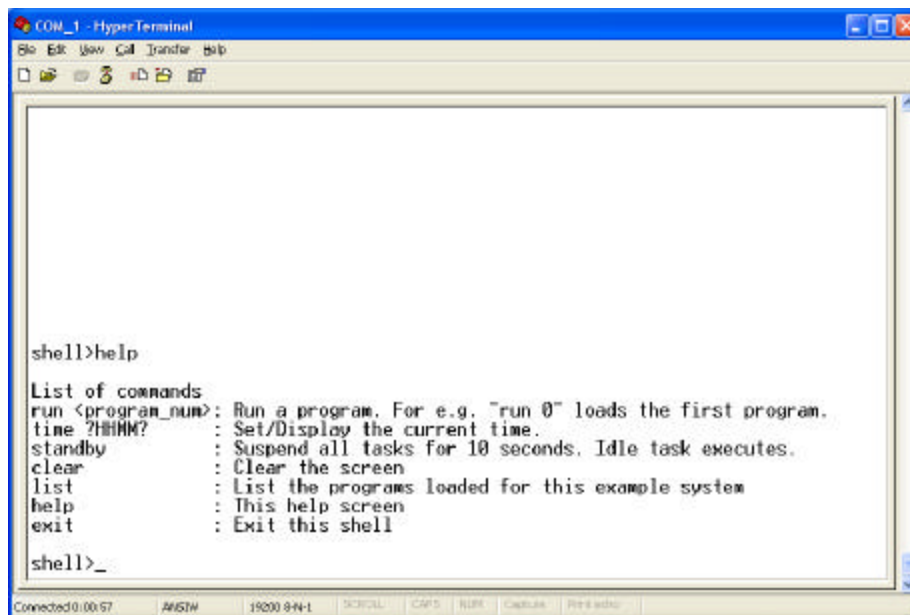
OK Cancel Help

10 Running the demo

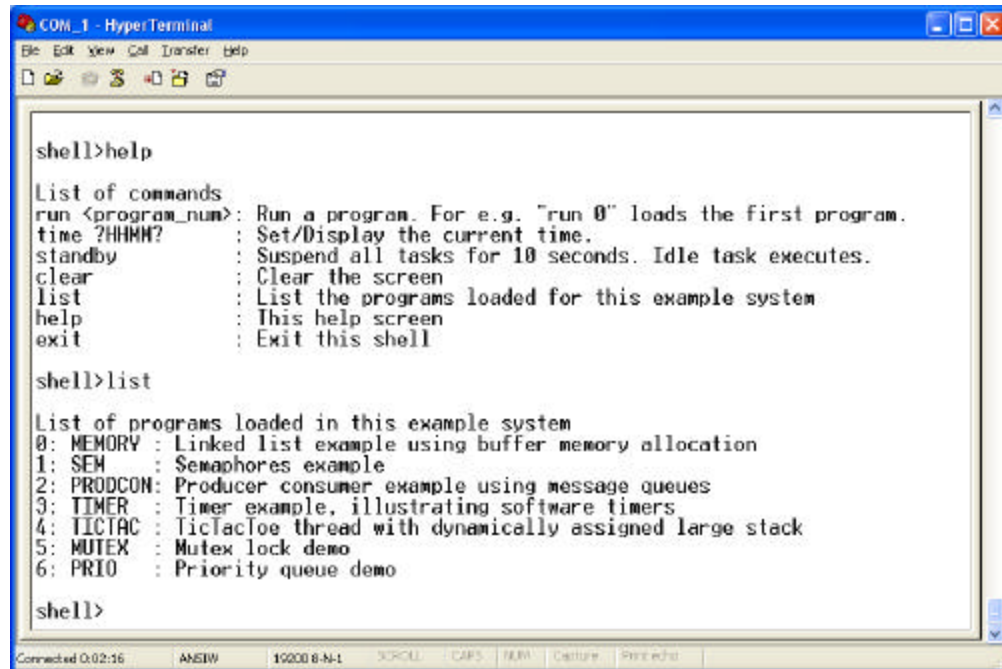
1. Since the FPGA BRAM was used as scratch area for the Flash programmer, the XMK demo data in the BRAM is over written. So, the FPGA must be configured again to load the BRAM with the application data. Select **Device Configuration > Download Bitstream** from the XPS GUI to configure the FPGA. Once the FPGA is configured, the XMK program will begin to run and you should see the following on the Hyper Terminal.



- Enter “help” to get a list of commands.



- Enter **“list”** to get a list of programs.



```

COM_1 - HyperTerminal
File Edit View Call Transfer Help

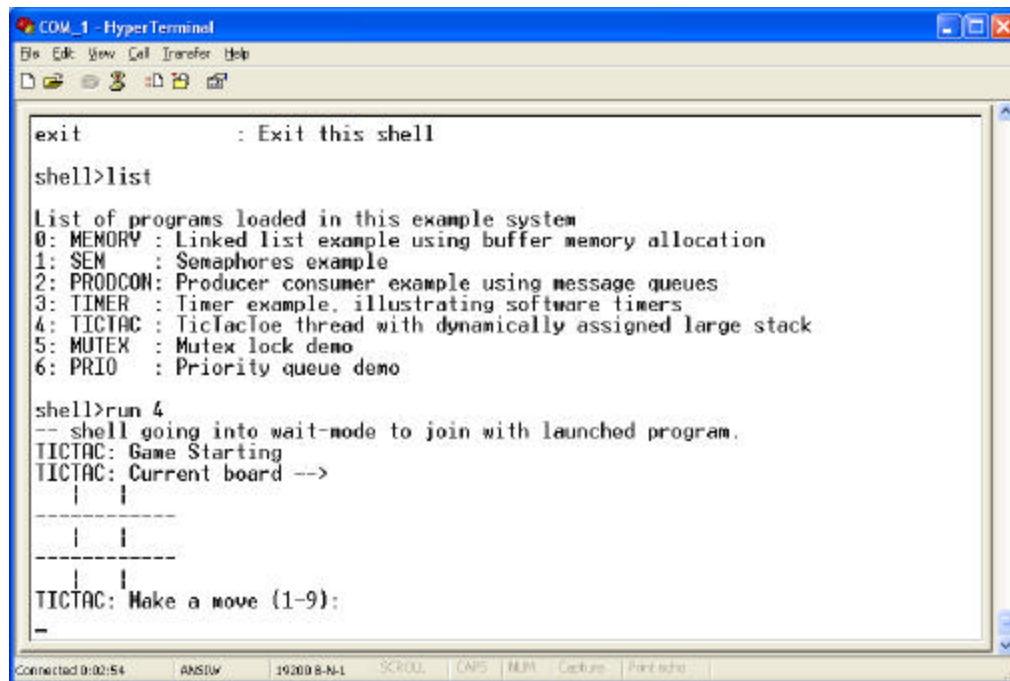
shell>help
List of commands
run <program_num>: Run a program. For e.g. "run 0" loads the first program.
time ?HHMM?      : Set/Display the current time.
standby          : Suspend all tasks for 10 seconds. Idle task executes.
clear            : Clear the screen
list             : List the programs loaded for this example system
help            : This help screen
exit            : Exit this shell

shell>list
List of programs loaded in this example system
0: MEMORY : Linked list example using buffer memory allocation
1: SEM     : Semaphores example
2: PRODCON: Producer consumer example using message queues
3: TIMER   : Timer example, illustrating software timers
4: TICTAC  : TicTacToe thread with dynamically assigned large stack
5: MUTEX   : Mutex lock demo
6: PRIO    : Priority queue demo

shell>

```

- Enter **“run”** followed by a number (0-6) to run a program. For example, enter **“run 4”** to play the Tic-Tac-Toe game.



```

COM_1 - HyperTerminal
File Edit View Call Transfer Help

exit      : Exit this shell

shell>list
List of programs loaded in this example system
0: MEMORY : Linked list example using buffer memory allocation
1: SEM     : Semaphores example
2: PRODCON: Producer consumer example using message queues
3: TIMER   : Timer example, illustrating software timers
4: TICTAC  : TicTacToe thread with dynamically assigned large stack
5: MUTEX   : Mutex lock demo
6: PRIO    : Priority queue demo

shell>run 4
-- shell going into wait-mode to join with launched program.
TICTAC: Game Starting
TICTAC: Current board -->
|  |
|  |
|  |
|  |
|  |
TICTAC: Make a move (1-9):
_

```