XILINX®

XAPP575 (v1.1.1) August 5, 2005

UltraController-II: Minimal Footprint Embedded Processing Engine

Author: Punit Kalra

Summary

UltraController-II is a minimal footprint embedded processing engine based on the PowerPC™ 405 (PPC405) processor core embedded within Virtex™-4 FX and Virtex-II Pro Platform FPGAs. Computing performance is maximized and FPGA resource usage minimized by running code strictly from the integrated PPC405 caches. General-purpose input/output (GPIO) is available directly from the PPC405 core. Interrupt handling is provided for a user defined external interrupt line, a programmable interval timer (PIT), and a fixed interval timer (FIT). System designers can easily incorporate the UltraController-II black-box processing engine into larger ISE designs to gain additional degrees of freedom by balancing usage of the high-performance FPGA fabric with the algorithmic flexibility of software.

This application note presents the features and benefits of the UltraController-II along with a brief overview of the tutorial applications included with the design. The accompanying tutorials and reference designs include VHDL, Verilog, and example C-code applications with step-by-step procedures. Performance characteristics and how to field an UltraController-II system using Xilinx configuration solutions are also provided.

Introduction

The UltraController-II reference design is a black-box processing engine that includes 32 bits of user-defined general purpose input and output as well as interrupt handling capability. UltraController-II applications are developed within the 16 KB instruction-side and 16 KB data-side cache memory of the PPC405 core. Users developing embedded designs that require PLB and OPB bus resources are directed to the Xilinx Platform Studio (XPS) toolset for creating expandable processor designs that leverage the full set of IP and software drivers offered by Xilinx (see http://www.xilinx.com/edk).

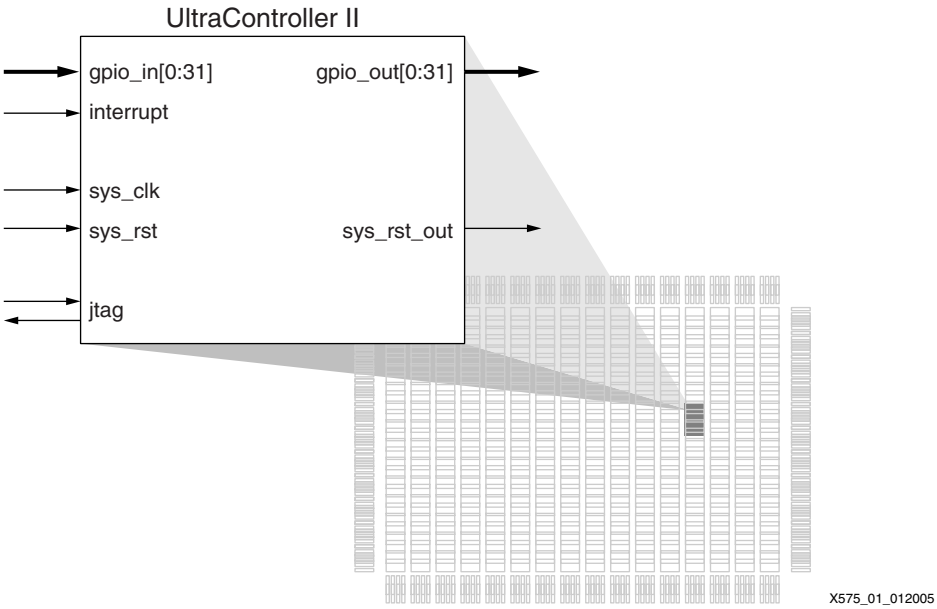


Figure 1: **Black-Box View of UltraController-II**

© 2005 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc.
All other trademarks are the property of their respective owners.

www.BDTIC.com/XILINX

Features

- Scalable CPU clock
  - Up to 450 MHz in a Virtex-4 FX -12 speed grade device
    - See DS112 and DS302 for information about Virtex-4 devices
  - Up to 400 MHz in a Virtex-II Pro -7 speed grade device
    - See DS083 for information about Virtex-II Pro devices
- Integrated cache-based program store
  - 16 KB I-side
  - 16 KB D-side
- No block RAMs used
- 32-bit output
- 32-bit input
- External user interrupt line (EXT)
- Programmable interval timer (PIT)
- Fixed interval timer (FIT)
- Watchdog timer (WDT)

Benefits

- Processing power is determined by program execution speed. UltraController-II can be clocked at the maximum PPC405 input clock frequency, which far exceeds any soft-core processor implementation
- Program instruction and data access speeds are maximized by using the integrated caches
- A minimal footprint processing engine frees up FPGA logic and block RAM (BRAM) resources
- 32-bit output and 32-bit input ports can interface with logic internal or external to the FPGA
- An external interrupt line allows UltraController-II applications to perform high-speed, base-level computation and handle time-critical events as they occur
- Timer resources (PIT and FIT) are available for commonly implemented embedded solutions for:
  - Time-of-day computation
  - Data-logging for system-service routines
  - Periodic servicing of time-sensitive external devices
- A watchdog timer (WDT) is available to monitor system sanity and recover from upsets by issuing a system reset

www.BDTIC.com/XILINX

Quick Start

The most current UltraController-II reference design, along with highly visual and detailed step-by-step presentations on running the quick-start tutorials, are available at http://www.xilinx.com/ultracontroller. This section provides a brief overview of the tutorials as run on a Memec Virtex-4 FX LC UltraController-II development platform. The reference design includes an ISE project for implementing the hardware design and an EDK project for building the software applications. The ISE project consists of an example top-level design that instantiates and connects an UltraController-II module. Implementation of the ISE project produces a common bitstream for use across multiple software applications, and highlights the features of UltraController-II. Benefits of the hardware and software separation within in the UltraController-II design are covered in the "UltraController-II Internals" section.

Implement the UltraController-II Hardware Design in ISE

1. Unzip the uc2_1ppc_v4_vhdl.zip or uc2_1ppc_v4_vlog.zip reference design
2. Launch ISE and open the top_uc2_example.ise project
3. Select the top-level module, top_uc2_example, in the "Sources in Project" ISE window
4. In the "Process for Source" window, Select **Generate Programming File** → **Rerun All**
5. The generated bitstream, top_uc2_example.bit, is in the **projnav** directory

Build the Software Applications in Platform Studio

1. Launch XPS and open the uc2.xmp project
2. Select **Tools** → **Build All User Applications**
3. The output of the build process is the executable and linked format (ELF) files placed in ppc405_0/code/

Run the Software Applications from Platform Studio

The three applications provided with the UltraController-II reference design target Virtex-4 and Virtex-II Pro embedded development platforms. The target board must be configured with a bitstream before running any software applications.

1. Connect a Xilinx Parallel Cable IV (PC4) JTAG cable between the host PC and target board
2. From the open ISE project's "Process for Source" window, select **Configure Device (iMPACT)** → **Run**
3. Browse to the **projnav** directory and program the FPGA with top_uc2_example.bit
4. Close iMPACT

Application 1: Simon

This application code re-implements the simple game provided with the original UltraController as an example of software design migration. The reference application contains examples of an LED driver, an LCD character display driver, square-wave sound generation, reading pushbuttons, and a software state machine. The application code remains the same, only the I/O interface routines are modified. The I/O handling code, encapsulated in gpio.c, is new since UltraController-II GPIO is implemented differently than in the original UltraController. However, the low-level differences of the I/O handling are transparent to the application since the same names and parameters of the I/O interface routines are maintained. A recompile of the previously completed software application, simon.c, will incorporate the new I/O handling code, and can be run on Virtex-4 FX or Virtex-II Pro embedded development platforms.

1. From the open EDK project, Launch XMD (**Tools** → **XMD**)
2. Launch GDB (**Tools** → **Software Debugger**) and select the ppc405_0_simon application
3. Download and run the application by selecting **Run** → **Run** from the GDB source window menu bar
4. Interact with the running Simon "melody-repeat" game
5. Stop the application by exiting GDB

www.BDTIC.com/XILINX

Application 2: Interrupt Handling (External, PIT, FIT)

The interrupt handling capability of UltraController-II is demonstrated by showing a count of the three types of interrupts recognized by UltraController-II on the LCD display. Counts are displayed for the non-critical external interrupt line, the PIT interrupt, and the FIT interrupt. Application code sets the interrupt rate, keeps track of the number of interrupts, and displays the resulting counts modulo 16.

1. Launch GDB (**Tools** → **Software Debugger**) and select the ppc405_0_interrupts application
2. Download and run the application by selecting **Run** → **Run** from the GDB source window menu bar
3. Observe the LCD display showing the counts (modulo 16) for each type of interrupt seen
4. Press pushbutton 1 to generate an external interrupt
5. Stop the application by exiting GDB

Application 3: Watchdog Timer

The watchdog timer application demonstrates the processor reset handling capabilities of UltraController-II. The watchdog timer is initialized, and resets the processor after a software-specified number of processor clock cycles, provided that the timer has not been cleared by the running application. The application continually sequences through two software states handling WDT timeouts. LEDs and messages on the LCD display indicate the code state.

In the first state, the program loops for 20 seconds and during each pass of the loop the watchdog timer is cleared before it expires. An LED is illuminated and a message appears on the LCD. After reaching the maximum number of loop iterations in the first state, the second software state is entered, causing a different LED to illuminate and a new message to appear on the LCD. In this second state, the watchdog timer is allowed to expire, thereby resetting the processor. The code now begins execution again from the reset vector entry point and repeats its course of events by continually passing through the two states.

1. Launch GDB (**Tools** → **Software Debugger**) and select the ppc405_0_watchdog application
2. Download and run the application by selecting **Run** → **Run** from the GDB source window menu bar
3. Observe the software state sequences as the WDT is cleared and allowed to expire
4. Stop the application by exiting GDB
5. Exit XPS

www.BDTIC.com/XILINX

UltraController-II Internals

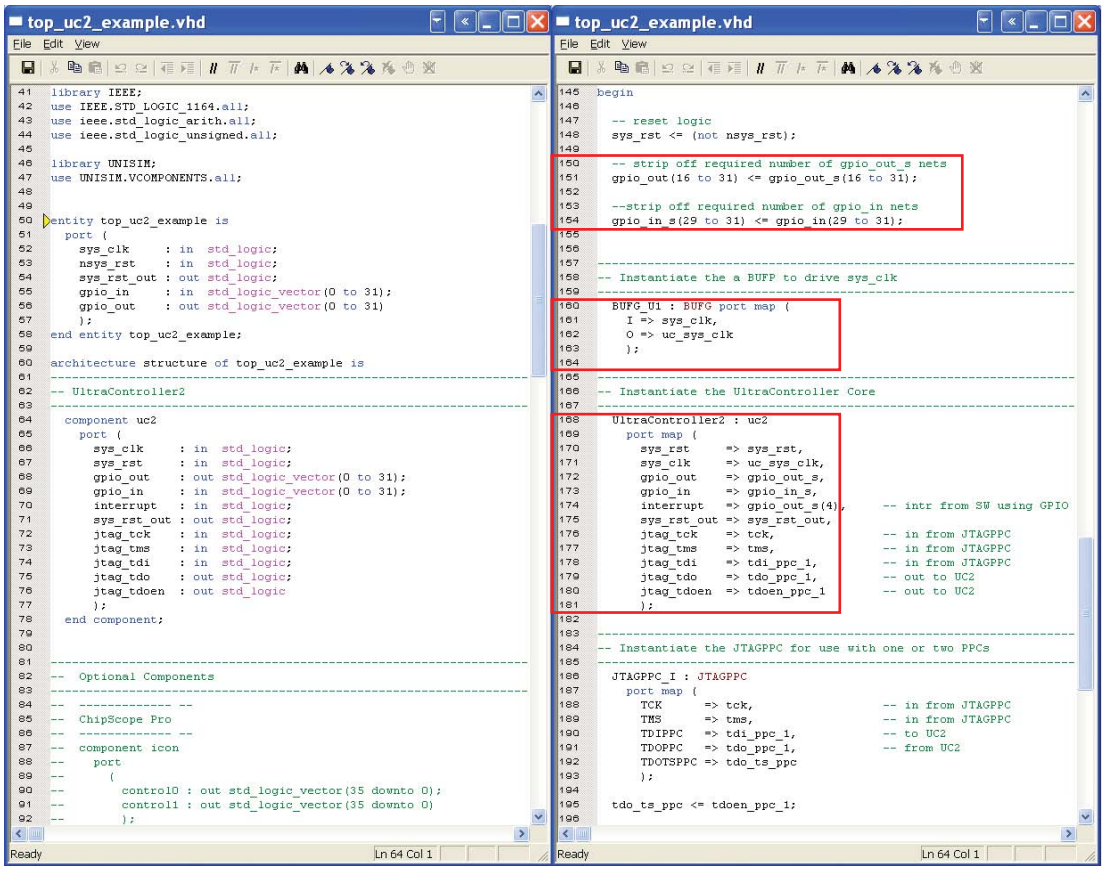
In today's complex and competitive design environment, products must be designed and verified more rapidly than ever before. This can be accomplished by partitioning designs into functional sub-blocks. The UltraController-II solution inherently separates a design into hardware and software functional sub-blocks, or components, due to its cache-based nature. The original BRAM-based UltraController provides users the ability to initialize both the hardware and software components of the system through the bitstream. Compared with traditional embedded systems of similar size, a single bitstream file with both hardware and software components eliminates the need for external non-volatile storage that only contains software. UltraController-II designs also use a single initialization file. In addition, UltraController-II provides the capability of being able to independently modify the hardware or software components by taking advantage of Xilinx configuration solutions. Multiple software design iterations can be created without modifying the bitstream, thereby limiting the scope of any introduced design changes. Once an UltraController-II black-box is integrated into a larger ISE design and verified, the hardware can be locked down and become a “golden” bitstream. This golden bitstream provides designers independence from development tool revisions and the ability to reestablish a known hardware state at any time in the future.

UltraController-II offers additional built-in functionality and a reduced resource footprint when compared with UltraController. Program storage for both the instructions and data now resides within the PPC405 caches, thereby eliminating the need for any block RAM. General purpose input/output (GPIO) is available directly from the PPC405 block and provides designers access to 32 bits of input and output. Exception handling permits designers to process a user interrupt line, the Programmable Interval Timer (PIT) and the Fixed Interval Timer (FIT) interrupts. Reset and boot logic are also covered. The “ISE Integration” section shows how an UltraController-II black-box module is integrated into a larger ISE design. The sections that follow present the salient features of UltraController-II.

ISE Integration

The UltraController-II reference design includes an example top-level design, top_uc2_example.vhd (.v). Figure 2, page 6 shows how the UltraController-II black-box design is embedded within an ISE system-level design. The left-hand screen shot shows the VHDL component declarations, whereas the right-hand screen shot shows the component instantiations. The top-level module describes the clocking, reset, interrupt, and GPIO signal wiring used in an example system design incorporating the UltraController-II. Line 160 shows that the input clock is buffered and routed through the design on a global clock line. Lines 151-154 show that this example uses only a portion of the 32-bits of available UltraController-II GPIO and that these signals are available as top-level ports for routing to FPGA pins. The UltraController-II signals can be directed to external pins or to other user logic in the FPGA design. Lines 168-181 show how the UltraController-II black-box component is wired to other logic in the top-level module. The JTAGPPC component is used to connect the UltraController-II into the FPGA JTAG chain.

www.BDTIC.com/XILINX



X575_02_072905

Figure 2: **Example Top-Level Module with Integrated UltraController-II Black-Box**

www.BDTIC.com/XILINX

Figure 3 shows a detailed block diagram of the UltraController-II module and Table 1 provides its memory map.

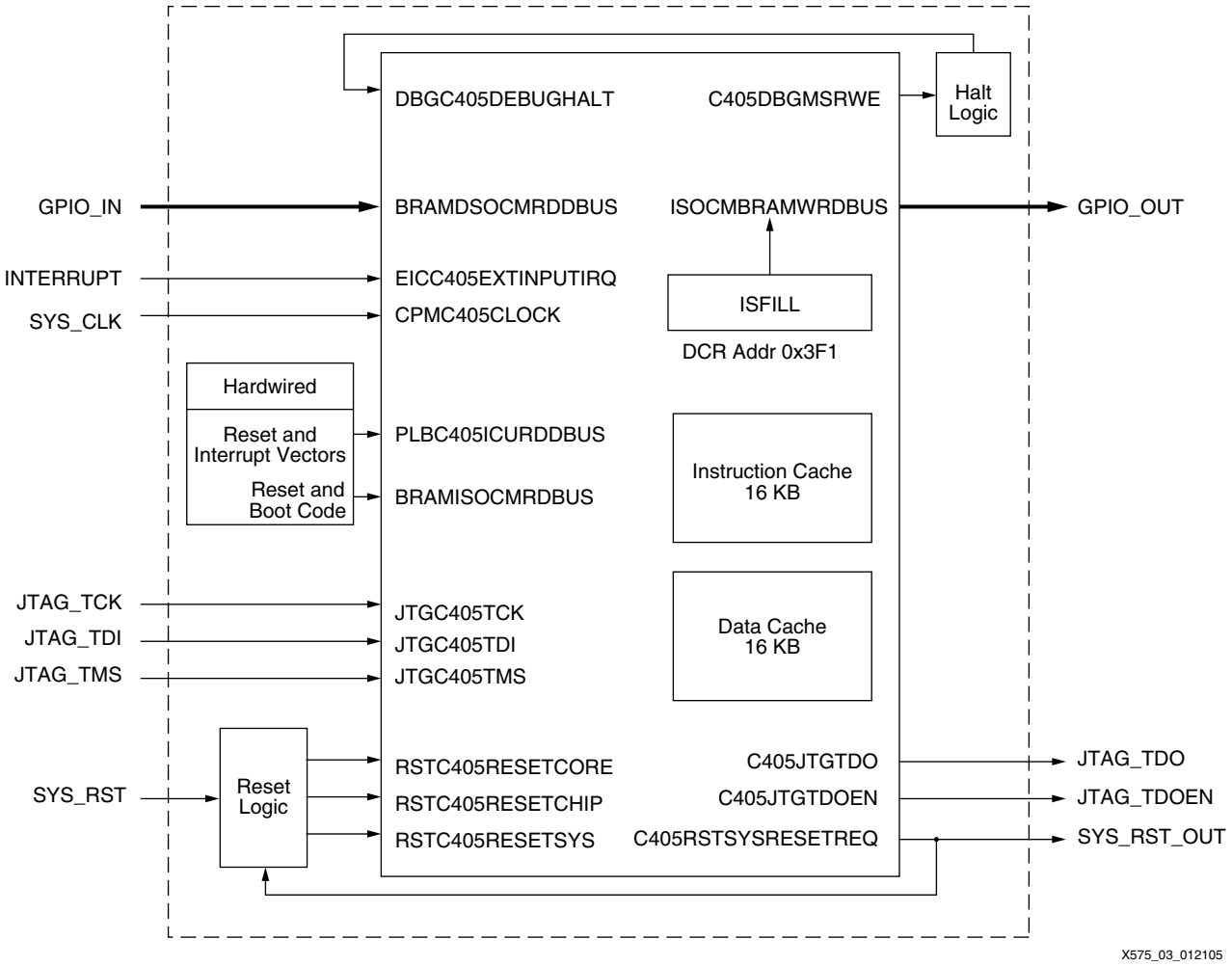


Figure 3: **Detailed UltraController-II Black-Box Block Diagram**

Table 1 presents a programmer's view of the UltraController-II memory map.

Table 1: **UltraController-II Memory Map**

Description	Address	Comment
Start of Instruction Storage	0xFFFF_0000	Cache-resident location defined in linker script
Start of Data Storage	0xFFFF_8000	Cache-resident location defined in linker script
Input Port	0x0100_0000	Input read from OCM data-side read bus
Output Port	0x3F1	Output written to OCM instruction-side write bus through the DCR ISFILL register
Interrupt Vector Trap	0xFFFF_3FF8	All interrupts vector through this location to the interrupt handler
Interrupt Vector Table	0xF000_0000	Instruction-side PLB is hardwired to jump to the Interrupt Vector Trap address for all interrupts
Reset Vector	0xFFFF_FFFC	On reset, processor executes code here and wraps program counter to 0x0000_0000 on OCM to continue execution

www.BDTIC.com/XILINX

Cache-Based Processing

UltraController-II executable software images must fit within the combined cache storage space of 16 KB for instruction code and 16 KB for data. Cache-based code executes at the fastest possible speed for a given processor clock frequency. Developing cache-based code is not much different than code that would target the block RAM or external memory. During the software compilation process, an UltraController-II linker script (Figure 4, page 9) describes the memory address ranges for the compiler to use when associating memory locations for the instructions, data, and other pertinent sections of executable and linked format (ELF) file that initializes the caches with runnable machine code. In general, embedded applications should be written with re-entrancy in mind and have variables initialized at run-time. Debugging cache-based code requires tools that can manipulate hardware breakpoints and display cache-resident data.

Loading and running cache-based applications is accomplished through the UltraController-II JTAG port. The boot sequence is as follows:

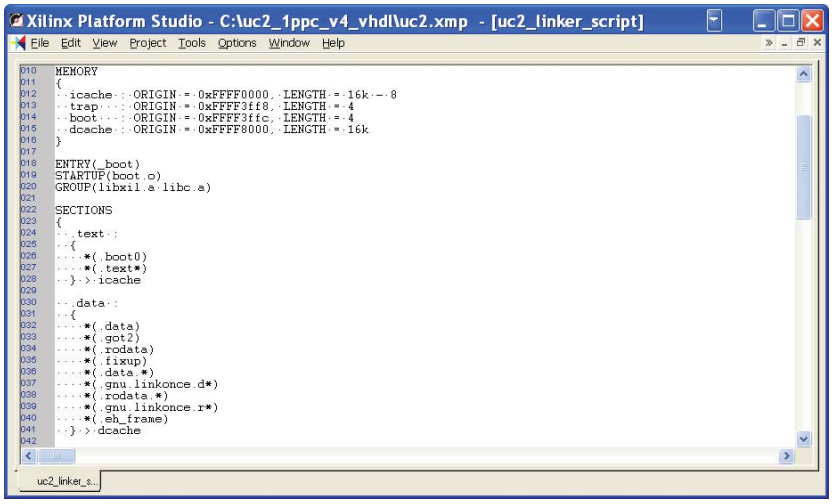
1. The FPGA is configured with a bitstream. The UltraController-II hardware holds the processor in a halted state.
2. Processor is unhalted and commanded, through JTAG, to load software into the caches.
3. Code execution starts at the reset vector address (0xFFFFFFFC).
4. Processor instructions hard-wired onto the PLB and OCM buses begin executing from the reset vector to turn on the instruction cache and jump to address 0xFFFF3FFC. This is the entry point for user code as defined by the UltraController-II linker script.
5. Data cache is enabled through code in crt0.S. User applications are compiled with crt0.S to set up the run-time environment and start running user code beginning at main().

During code execution, a user- or software-initiated reset can be issued. On reset, the processor disables caches as a part of its reset sequence and then begins code execution at the reset vector. UltraController-II then follows steps 3 through 5 above to restart the user application.

Table 2: **Files Provided for UltraController-II Applications**

File Name	Description
crt0.S	Contains startup code and initializes the processor with the C run-time environment.
xvectors.S	Provides the exception handling code prologue and epilogue with a hook for the user C-code handler. Entry is through the Interrupt Vector Trap address.
xexception_l.c	C-code interrupt initialization and handling routines.
gpio.c	Application interface for I/O access.
lcd.c	Implements low-level LCD character display access routines.
uc2_linker_script	Linker script that defines software load areas.

www.BDTIC.com/XILINX



X575_04_072905

Figure 4: **UltraController-II Linker Script**

GPIO

A 32-bit wide input port and a 32-bit wide output port are available to UltraController-II applications. The input port is located at 0x01000000 and is provided by the 32-bit DSOCM read-data bus. The output port is at location 0x3F1 in DCR address space and is provided by the 32-bit ISOCM ISFILL register.

The gpio.c routine (Figure 5, page 10), included as part of an UltraController-II reference application, contains the low-level GPIO access code while providing functions to abstract the application code from the implementation details. The following functions within gpio.c are available for use by application code:

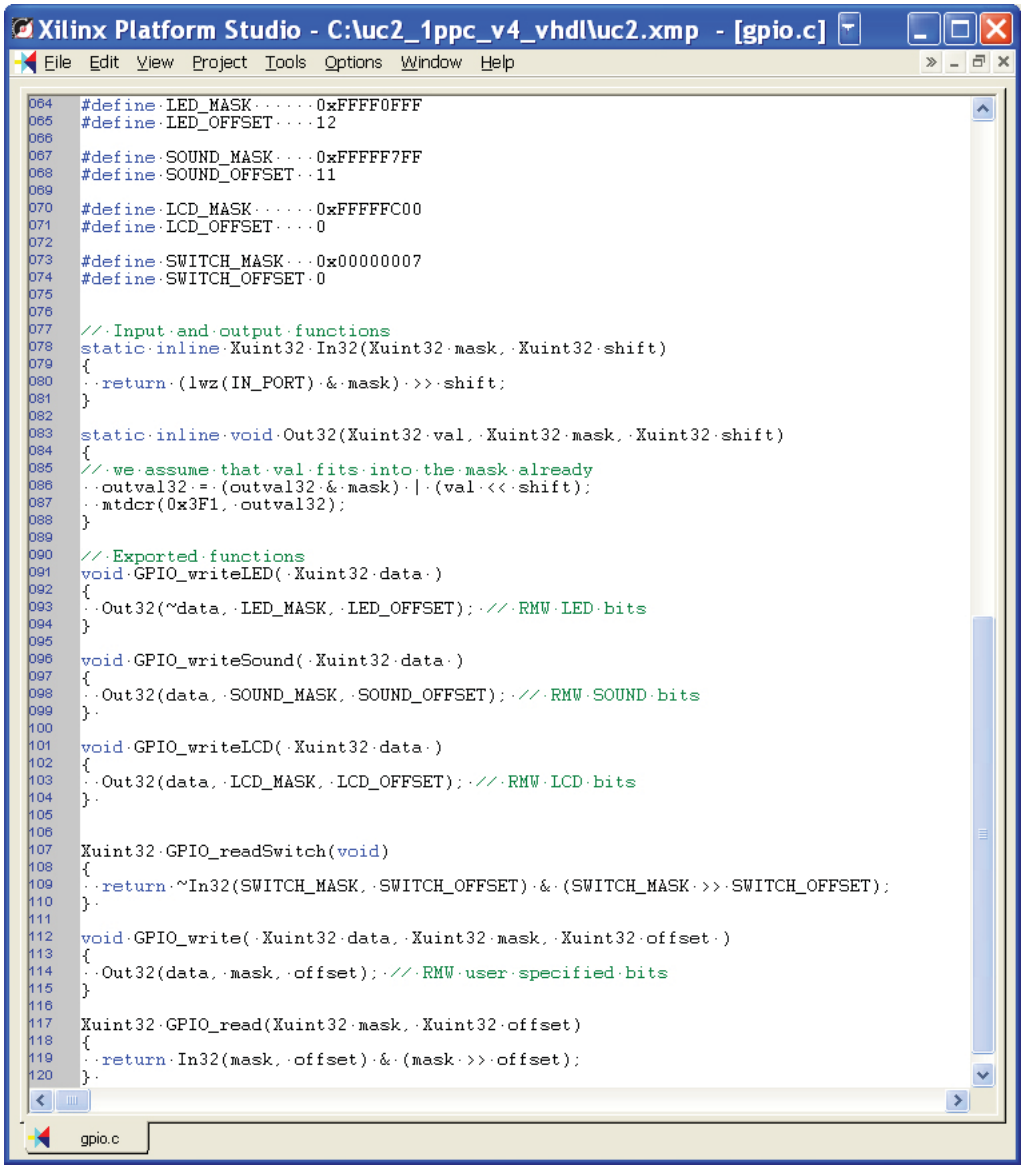
- GPIO_writeLED
- GPIO_writeSound
- GPIO_writeLCD
- GPIO_readSwitch

These same four routines are available in the original UltraController design. UltraController-II maintains these four original routines, plus offers two general-purpose routines in which the user can set the mask and offset information.

- GPIO_write
- GPIO_read

Additional user-written GPIO routines can be added to gpio.c in a similar manner.

www.BDTIC.com/XILINX




```
064 #define LED_MASK      0xFFFF0FFF
065 #define LED_OFFSET    12
066
067 #define SOUND_MASK    0xFFFFF7FF
068 #define SOUND_OFFSET  11
069
070 #define LCD_MASK      0xFFFFFC00
071 #define LCD_OFFSET    0
072
073 #define SWITCH_MASK   0x00000007
074 #define SWITCH_OFFSET 0
075
076
077 // Input and output functions
078 static inline Xuint32 In32(Xuint32 mask, Xuint32 shift)
079 {
080   return (lwz(IN_PORT) & mask) >> shift;
081 }
082
083 static inline void Out32(Xuint32 val, Xuint32 mask, Xuint32 shift)
084 {
085 // we assume that val fits into the mask already
086   outval32 = (outval32 & mask) | (val << shift);
087   mtdcr(0x3F1, outval32);
088 }
089
090 // Exported functions
091 void GPIO_writeLED( Xuint32 data )
092 {
093   Out32(~data, LED_MASK, LED_OFFSET); // RMW LED bits
094 }
095
096 void GPIO_writeSound( Xuint32 data )
097 {
098   Out32(data, SOUND_MASK, SOUND_OFFSET); // RMW SOUND bits
099 }
100
101 void GPIO_writeLCD( Xuint32 data )
102 {
103   Out32(data, LCD_MASK, LCD_OFFSET); // RMW LCD bits
104 }
105
106
107 Xuint32 GPIO_readSwitch(void)
108 {
109   return ~In32(SWITCH_MASK, SWITCH_OFFSET) & (SWITCH_MASK >> SWITCH_OFFSET);
110 }
111
112 void GPIO_write( Xuint32 data, Xuint32 mask, Xuint32 offset )
113 {
114   Out32(data, mask, offset); // RMW user specified bits
115 }
116
117 Xuint32 GPIO_read(Xuint32 mask, Xuint32 offset)
118 {
119   return In32(mask, offset) & (mask >> offset);
120 }
```

X575_05_072905

Figure 5: **UltraController-II gpio.c Routine**

Most processor-based systems incorporate a storage hierarchy to balance performance demands against resource cost. In general, off-chip storage is the least costly, but provides the slowest-access memory. Some examples of internal storage, listed in order of increasing access speed, are on-chip memory (such as block RAMs), on-processor memory (such as caches), and in-processor storage using processor registers. GPIO port performance is highly application-code dependent. When developing I/O code, the designer must consider trade-offs such as scalability, the level of abstraction offered by the I/O routines, and the I/O rate dictated by the application. The port performance numbers presented in this application note resulted from an application that invoked the Xtime_GetTime function to measure the number of clock cycles that elapsed during the course of executing of a linear sequence of 1000 I/O reads or writes. The I/O read and write routines were optimized for speed and were not required to perform any masking, shifting, or read-modify-write operations. Code for bursting the contents of an initialized memory buffer to the output port, similar to that shown in Table 3 through Table 5, page 11, was used to measure cycle counts. Register-based measurements were

www.BDTIC.com/XILINX

made by specifying the register storage class for certain C-language variables. In all cases, the Platform Studio GNU C-compiler with optimization set to Level 2 was used.

Table 3: **GPIO Timing Code**

```
Xtime_GetTime( &start );
Out32Fast( buf[0] );
Out32Fast( buf[1] );
…
Out32Fast( buf[999] );
Xtime_GetTime( &end );
```

Table 4: **Low-level Input Routine**

```
static inline Xuint32
In32Fast()
{
   return( lwz( IN_PORT ) );
}
```

Table 5: **Low-level Output Routine**

```
static inline void
Out32Fast( Xuint32 val32 )
{
   mtdcr( 0x3F1, val32 );
}
```

Table 6 shows that writing data to the output port from a cache memory buffer requires 10 clock cycles, whereas fetching data from the input port and placing it into a cache memory buffer requires 4 cycles. Table 6 also shows that when register storage is used a new output value is still generated every 10 clock cycles, but a new input value can now be obtained every 2 cycles.

Table 6: **CPU Cycle Counts for I/O**

Storage Type	I/O Operation	
	Output	Input
Cache	10 cycles	4 cycles
Register	10 cycles	2 cycles

Table 7, page 12 provides data on how often (Figure 6, T_{SAMPLE}) a new data sample can be placed on the output port or retrieved from the input port as a function of both storage type and CPU frequency.

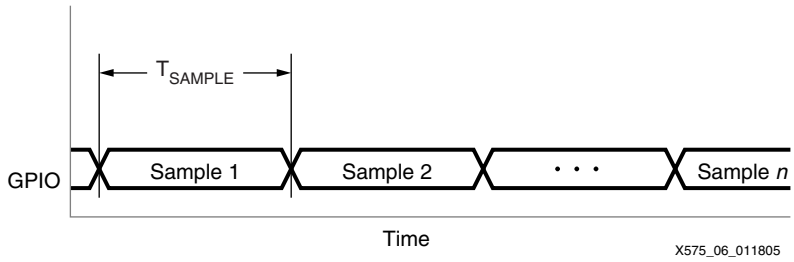


Figure 6: **Data Sample Timing**

www.BDTIC.com/XILINX

Table 7: **I/O Sample Time**

SYS_CLK Frequency	I/O Operation			
	Output		Input	
	Cache	Register	Cache	Register
100 MHz	100 ns	100 ns	40 ns	20 ns
200 MHz	50 ns	50 ns	20 ns	10 ns
300 MHz	33.3 ns	33.3 ns	13.3 ns	6.7 ns
400 MHz	25 ns	25 ns	10 ns	5 ns
450 MHz(1)	22.2 ns	22.2 ns	8.9 ns	4.4 ns

Notes:

1. CPU clock frequency of 450 MHz is available on Virtex-4 FX (-12 speed grade) devices only. For device data sheets, see "References," page 21.

Interrupts

UltraController-II supports interrupts and provides a reference application with an exception handler that recognizes timer interrupts from the PIT and FIT, as well as non-critical external interrupts generated by the user-wired UltraController-II external interrupt input line. Interrupt handling can be added to user applications by including two UltraController-II reference files, xexception_l.h and xexception_l.c, to user software projects.

When an external interrupt or a timer interrupt occurs, the processor fetches the next instruction to be executed from the start of the exception vector table plus an address offset specific to the particular type of interrupt. The UltraController-II exception vector table begins at address 0xF0000000 and has all the vector table entries hardwired to jump to location 0xFFFF3FF8. This is a cache-resident location containing a jump to the exception handler code. The exception handler determines the type of interrupt that occurred and then calls the appropriate user-registered interrupt handler.

The PIT and FIT interrupts are used for generating internal interrupts based on time scales determined by the processor clock and the timer registers. See the EDK *PowerPC Processor Reference Guide* ("References," page 21) for details on using the timers. Software interfaces for the timers are located in the EDK stand-alone BSP library files, xtime_l.h and xtime_l.c.

The UltraController-II interrupt input line can be wired to user logic internal to the FPGA, or to an external interrupt source connected through a user-specified FPGA pin. This interrupt input is level sensitive and active-High.

Figure 7 shows the UltraController-II external interrupt latency using a stand-alone software application and a digital storage oscilloscope. The ChipScope Pro™ VIO core or software controlling the GPIO lines can be used to generate an interrupt signal. Interrupt latency is the time interval from the rising edge of the interrupt to the falling edge of the ISR_OUTPUT trace. Table 8, page 13 shows that the interrupt latency scales linearly with the system clock frequency and is only 0.42 µs at 450 MHz. The length of time spent within the interrupt service routine (ISR), as indicated by $Time_{ISR}$ in Figure 7, is determined by the amount of code in the ISR. The maximum rate of interrupts is 1/(Latency + $Time_{ISR}$). The ISR_OUTPUT signal is created by monitoring a GPIO bit that toggles on entry and exit of the C-coded user ISR. Once the ISR exits, the external interrupt is again recognized.

www.BDTIC.com/XILINX

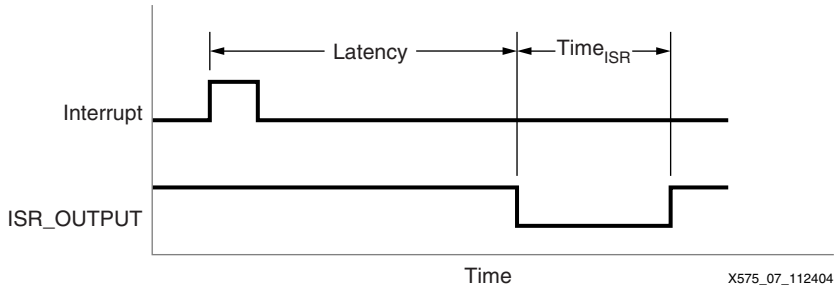


Figure 7: **UltraController-II Interrupt Latency**

Table 8: **Interrupt Latency**

SYS_CLK Frequency	Latency
100 MHz	1.9 µs
200 MHz	0.94 µs
300 MHz	0.63 µs
400 MHz	0.47 µs
450 MHz(1)	0.42 µs

Notes:

1. CPU clock frequency of 450 MHz is available on Virtex-4 FX (-12 speed grade) devices only. For device data sheets, see "References," page 21.

Reset Handling

UltraController-II contains a processor reset logic block that handles external reset requests, such as those generated manually using a pushbutton, as well as WDT-generated resets. Reset timing follows the requirements specified in the EDK *PowerPC 405 Processor Block Reference Guide*. (See "References," page 21.)

Internally, UltraController-II forwards the PPC405 reset request signal, C405RSTSYSRESETREQ, to its processor reset logic allowing software applications using the WDT to generate an UltraController-II system reset. Software applications can initialize and control the WDT through routines provided by the XPS xtime_l.c source file and employed by the ppc405_0_watchdog example application in the reference design.

The UltraController-II SYS_RST_OUT signal is available to notify external logic of the system reset event.

The following sequence of events is initiated by a reset:

1. Code execution starts at the reset vector address (0xFFFFFFFC).
2. Processor instructions hard-wired onto the PLB and OCM buses begin executing from the reset vector to turn on the instruction cache and jump to address 0xFFFF3FFC. This is the entry point for user code as defined by the UltraController-II linker script.
3. Data cache is enabled through code in crt0.S. User applications are compiled with crt0.S to set up the run-time environment and start running user code beginning at main().

Debug Halt Logic

The debug and halt logic used by UltraController-II is described in XAPP571: *DEBUGHALT Controller for PowerPC Boot and Reset Operations*. Since UltraController-II is cache based and does not contain any bitstream initialized block RAM for holding a bootloop application, the DEBUGHALT controller provides the startup logic necessary to control the processor through a JTAG interface.

www.BDTIC.com/XILINX

Creating a Fielded Solution

Up to this point, a single UltraController-II bitstream has been used to initialize the FPGA hardware, and a series of software applications have been demonstrated running on a common hardware platform. GDB and a JTAG connection has been used to control and load various applications in a laboratory style debug environment with desktop bring-up tools. Figure 8 shows the lab tool flow for an UltraController-II design. A hardware design is implemented through the map, place and route phase to produce an ISE NCD file. The NCD file is then translated into the bitstream, used to configure a specific FPGA. After configuration, the ELF file produced by the XPS compiler can be downloaded to test the complete UltraController-II system from the desktop.

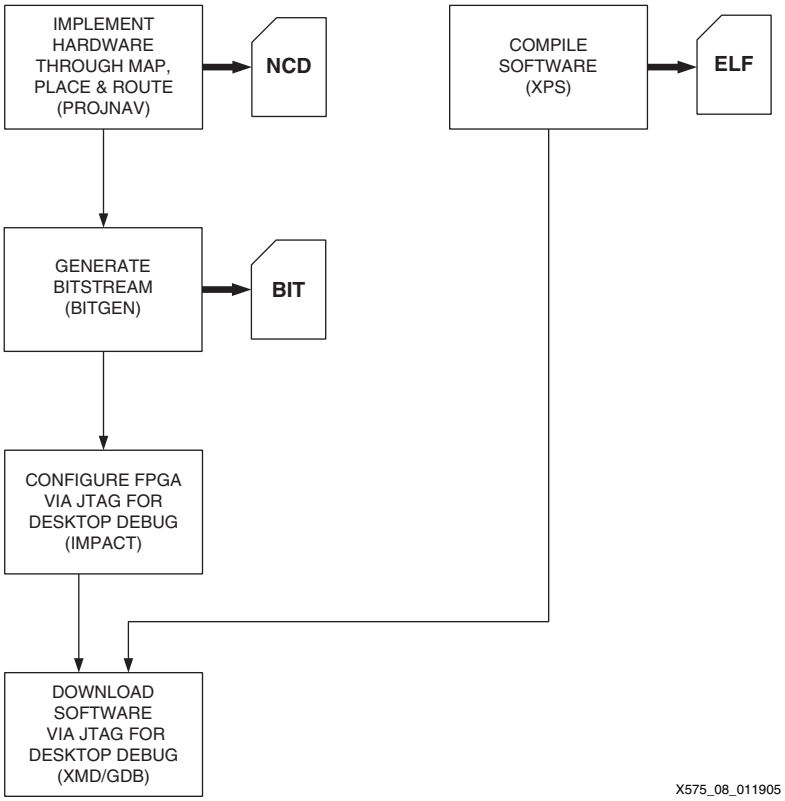


Figure 8: **Tool Flow for Desktop Debug**

By using the Xilinx System ACE CF controller or a Xilinx PROM to field a solution, the umbilical cord type of connection to a desktop can be eliminated. The System ACE CF solution is based on an ACE file and a JTAG connection to the FPGA, whereas the PROM solution uses an MCS file and one of the FPGA's configuration modes. The various FPGA configuration modes only differ in the source of the configuration clock and whether the interface width is 1 bit or 8 bits. In master mode, the FPGA supplies a clock to the PROM. In slave mode, an external device supplies the FPGA and PROM with the configuration clock. The "PROM Loading Method" section in this application note presents an UltraController-II solution fielded with the master-serial configuration mode. For further information on master-serial mode, and other configuration solutions, see UG071 listed in the "References" section. UG071 (Chapter 2) details the configuration interfaces and the standard board layout considerations for Virtex-4 Platform FPGAs. Configuration of Virtex-II Pro Platform FPGAs is presented in UG012 (Chapter 4).

www.BDTIC.com/XILINX

Although the initialization files stored on a CompactFlash card or a PROM are different, the general concept of how the hardware is configured and the software is loaded is identical. Figure 9 shows that an initialization file contains two sections, a hardware component followed by a software component. The UltraController-II reference design provides scripts that utilize ISE and XPS tools to combine the hardware and software components for generating ACE or MCS files. On power-up the hardware component of the initialization file configures the FPGA fabric logic and establishes a JTAG path to the PPC405 core. Once the hardware configuration completes, the software component of the initialization file is transferred across the JTAG connection into the PPC405 to load the processor caches. The UltraController-II is now ready to run the application software.

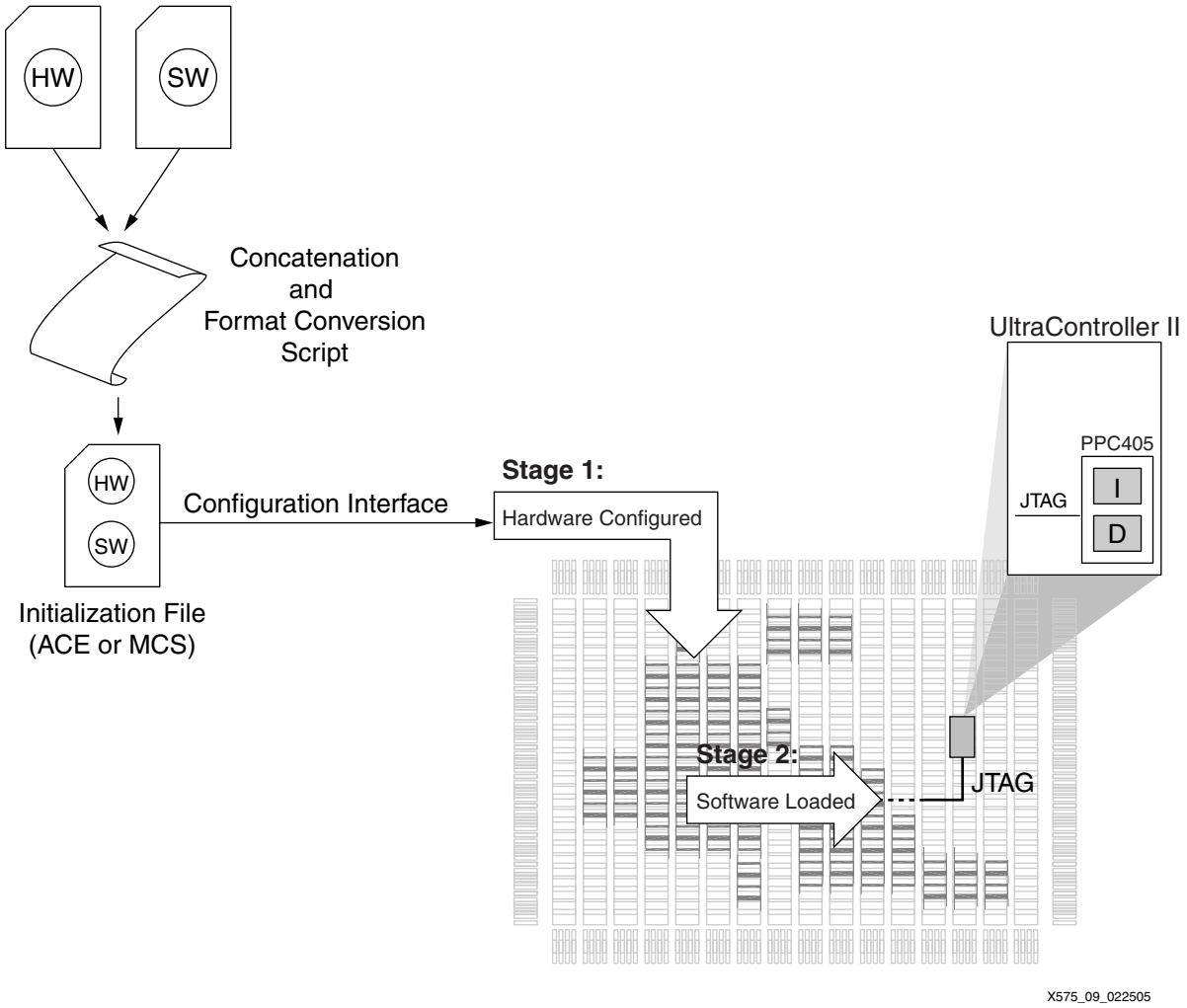


Figure 9: **Configuring the FPGA with Hardware and Software Components**

www.BDTIC.com/XILINX

System ACE CF Method

The Xilinx System ACE CF solution carries forward a JTAG method of configuring either hardware or software using an ACE file read from a local, non-volatile CompactFlash card. Instantiating and wiring the JTAGPPC hard core primitive in the top level of an UltraController-II based design establishes a JTAG pathway from the FPGA's dedicated JTAG pins into the processor core. Figure 10 shows the tool flow used for UltraController-II designs that are fielded with the System ACE CF controller. For the “Generate Initialization File” step, the reference design (uc2_1ppc_v4_vhdl.zip) contains a script that invokes the existing XPS tools for the user’s convenience.

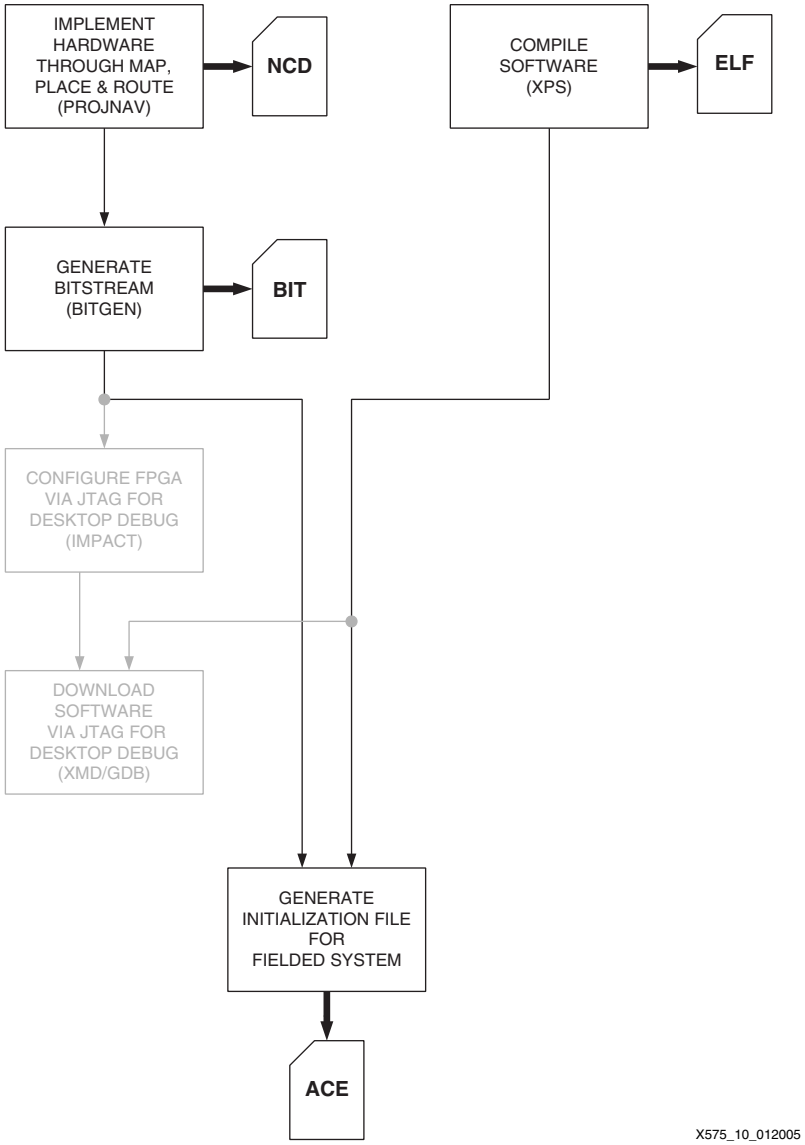


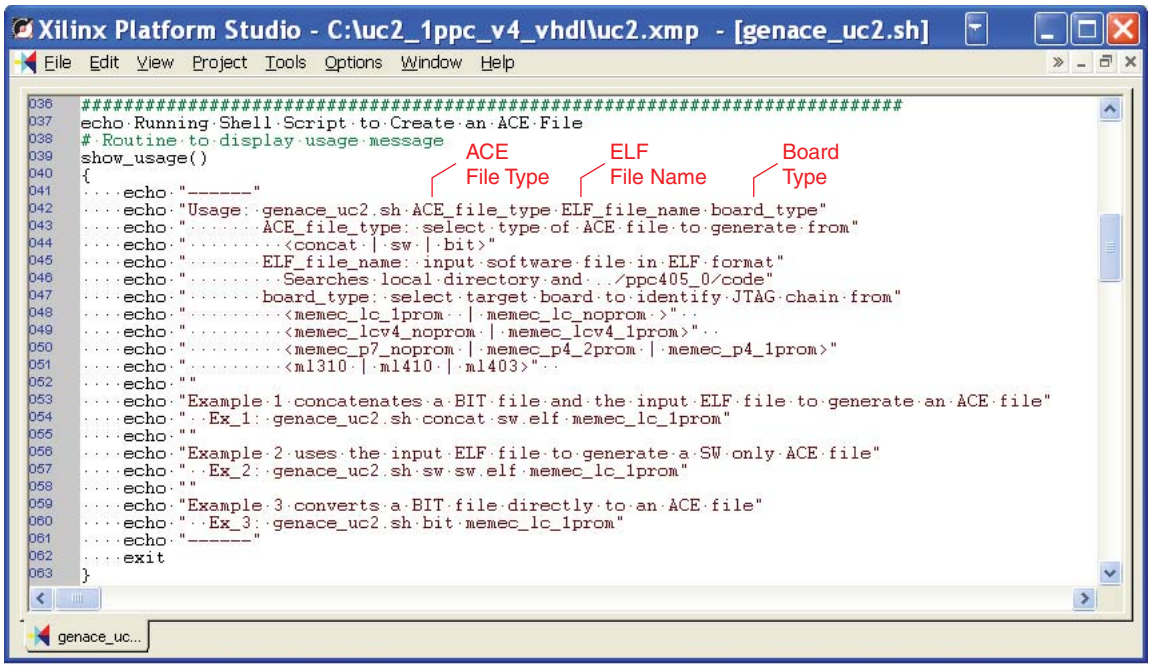
Figure 10: **Tool Flow for Fielding a System ACE CF-based UltraController-II Design**

XPS provides the tools necessary to create ACE files and offers examples in the EDK *Platform Studio User Guide*. (See “References,” page 21.) The UltraController-II reference design contains an example shell script (genace_uc2.sh) that invokes the XPS ACE file generation tools with the JTAG chain description and file creation options preset for the development platforms presented on the UltraController lounge at http://www.xilinx.com/ultracontroller. The genace_uc2.sh script within the **ace** directory creates an ACE file using a local genace.tcl file, the top_uc2_example.bit hardware configuration file, and a user-specified software (ELF)

www.BDTIC.com/XILINX

application. Users can modify this script as desired to create ACE files for custom target platforms.

Figure 11 shows that there are three input parameters to the genace_uc2.sh script; the type of ACE file to generate, the input ELF file, and the board type to identify the target platform's JTAG chain. An ACE file can be the single initialization file used to bring up a system, when it is formed by concatenating the bitstream with the executable software ELF file. Invoking the genace_uc2.sh script with an ACE file type of “concat” will generate such a file. The script can also generate a software only type of ACE file, which allows the user to change just the software component of a system with a previously configured FPGA. The software only type of ACE file offers a software update capability similar to what is available when debugging a system using a JTAG cable and desktop debug tools.




```
036 ##############################################################################
037 echo Running Shell Script to Create an ACE File
038 # Routine to display usage message
039 show_usage()
040 {
041     echo "------"
042     echo "Usage: genace_uc2.sh ACE_file_type ELF_file_name board_type"
043     echo "       ACE_file_type: select type of ACE file to generate from"
044     echo "         <concat | sw | bit>"
045     echo "       ELF_file_name: input software file in ELF format"
046     echo "         Searches local directory and ../ppc405_0/code"
047     echo "       board_type: select target board to identify JTAG chain from"
048     echo "         <memec_lc_1prom  | memec_lc_noprom >"
049     echo "         <memec_lcv4_noprom | memec_lcv4_1prom>"
050     echo "         <memec_p7_noprom | memec_p4_2prom | memec_p4_1prom>"
051     echo "         <ml310 | ml410 | ml403>"
052     echo ""
053     echo "Example 1 concatenates a BIT file and the input ELF file to generate an ACE file"
054     echo "  Ex_1: genace_uc2.sh concat sw.elf memec_lc_1prom"
055     echo ""
056     echo "Example 2 uses the input ELF file to generate a SW only ACE file"
057     echo "  Ex_2: genace_uc2.sh sw sw.elf memec_lc_1prom"
058     echo ""
059     echo "Example 3 converts a BIT file directly to an ACE file"
060     echo "  Ex_3: genace_uc2.sh bit memec_lc_1prom"
061     echo "------"
062     exit
063 }
```

X575_11_080105

Figure 11: **genace_uc2.sh Shell Script Used to Create an ACE File**

www.BDTIC.com/XILINX

PROM Loading Method

By replacing the JTAGPPC module with the prom2jtag HDL module provided in the **projnav** directory, a Xilinx PROM device can be used to configure the FPGA fabric and load an application into the PPC405 caches through the master-serial configuration mode. The target board must also have an FPGA user I/O pin wired to the CCLK net in order to support UltraController-II PROM loading on Virtex-4 FX and Virtex-II Pro devices. Incorporating the prom2jtag module does not alter the UltraController-II black box, and adds only 19 look-up tables (LUTs) and 15 flip-flops (FFs) to the top-level ISE design. Figure 12 shows the tool flow used for fielding a PROM-based UltraController-II system. A script is provided in the reference design (uc2_1ppc_v4_vhdl_prom.zip) for the step outlined by the dashed rectangle.

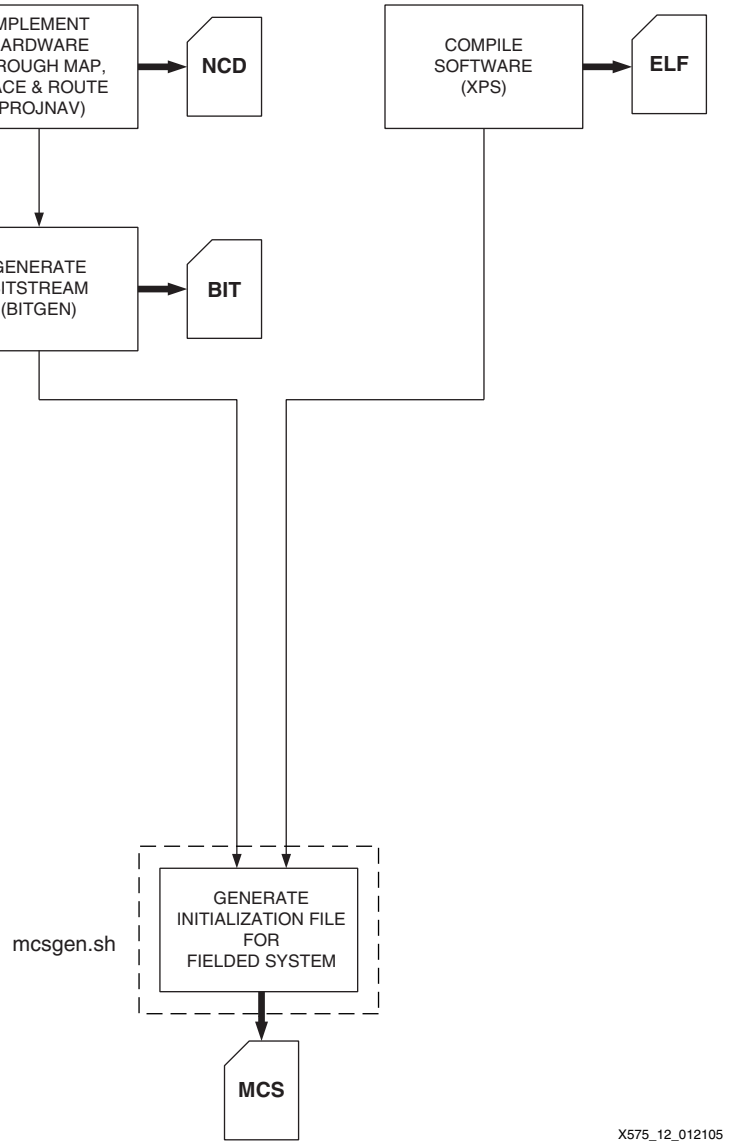


Figure 12: **Tool Flow for Fielding a PROM-based UltraController-II Design**

www.BDTIC.com/XILINX

Xilinx Platform Studio - C:\uc2_1ppc_v4_vhdl_prom\uc2.xmp - [mcsgen.sh]

File Edit View Project Tools Options Window Help

```
062 #
063 # convert ELF file into binary file
064 #
065
066 echo "------"
067 echo "Convert the ELF file into a binary file via xmd"
068 echo "Output: executable_cache.bin"
069 echo "------"
070 #connect ppc hw -cable type xilinx_binfile fname executable_cache.bin -configdevice devicenr 1 idcode 0x127e093
071 ./xmd.exe <<EOF
072 connect ppc hw -cable type xilinx_plugin modulename binfile fname executable_cache.bin -configdevice devicenr 1
073 rwr msr 0x40000
074 rwr msr 0
075 dow $prog
076 run
077 exit
078 EOF
079
080 #
081 # concatenate the binary files
082 #
083
084 echo "."
085 echo "------"
086 echo "Concatenate the software bin file with the hardware bit file"
087 echo "  system_cache.bit > system_hw_sw.bin"
088 echo "  executable_cache.bin >> system_hw_sw.bin"
089 echo "Output: system_hw_sw.bin"
090
091 cat system_cache.bit > system_hw_sw.bin
092 cat executable_cache.bin >> system_hw_sw.bin
093
094 #
095 # flip the bits in the binary file
096 #
097
098 echo "------"
099 echo "Running flipbit on system_hw_sw.bin"
100 echo "Output: system_hw_sw_fb.bin"
101 xtclsh flipbit.tcl system_hw_sw.bin system_hw_sw_fb.bin
102
103 #
104 # convert back to mcs
105 #
106
107 echo "------"
108 echo "Converting system_hw_sw_fb.bin to an mcs file"
109 echo "Output: "$self_name"_cache.mcs"
110 powerpc-eabi-objcopy -I binary -O ihex system_hw_sw_fb.bin $self_name"_cache.mcs"
```

mcsgen.sh

X575_13_072905

Figure 13: **mcsgen.sh Script Used to Create a PROM File**

In the master-serial mode, the FPGA generates a clock (CCLK) to read out the hardware initialization data from the PROM. Once the hardware configuration is completed, CCLK stops and the prom2jtag block begins to drive the clock input of the PROM from a user specified I/O pin. The software component of the PROM data is now read into the FPGA on its serial configuration pin DIN, and converted to JTAG signaling by the prom2jtag block before being driven onto the PPC405 JTAG inputs. The mcsgen.sh script, shown in Figure 13 and provided within the **prom** directory of the reference design, combines a bitstream and ELF file to generate an MCS file. The MCS file consists of the concatenated hardware (bitstream) and software (ELF) components in a format suitable for programming the Platform Flash on Virtex-4 FX and Virtex-II Pro UltraController-II development platforms. The mcsgen.sh script is also useful as an example tool flow for creating MCS files and can be tailored by the user for alternative target platforms.

Dual Processor Designs

A dual PPC405 UltraController-II reference design is available in the uc2_2ppc_v4_vhdl.zip (_vlog.zip) file. Since the UltraController-II black-box design has JTAG ports, it can be connected to other devices in the JTAG chain as shown in the top_uc2_example.vhd (.v) file. The reference designs maintain JTAG continuity by wiring both processors to the JTAGPPC primitive as described in the EDK *PowerPC 405 Processor Block Reference Guide*. (See "References," page 21.)

www.BDTIC.com/XILINX

Performance

Table 9 shows the UltraController-II compute performance, measured in units of Dhrystone Millions of Instructions per Second (DMIPS) and power consumption in milliwatts versus frequency. The respective figures of merit are 1.56 DMIPS/MHz and 0.45 mW/MHz for Virtex-4 FX Platform FPGAs, whereas 1.56 DMIPS/MHz and 0.90 mW/MHz are the figures of merit for the Virtex-II Pro family.

Table 9: **Dhrystone MIPS and Power Consumption versus Clock Frequency**

CPU Clock Frequency	Dhrystone MIPS	Power	
		Virtex-4 FX	Virtex-II Pro
100 MHz	156 DMIPS	45 mW	90 mW
200 MHz	312 DMIPS	90 mW	180 mW
300 MHz	468 DMIPS	135 mW	270 mW
400 MHz	624 DMIPS	180 mW	360 mW
450 MHz	684 DMIPS	203 mW	-

Notes:

1. CPU clock frequency of 450 MHz is available on Virtex-4 FX (-12 speed grade) devices only. For device data sheets, see "References," page 21.

Table 10 summarizes the data presented in the "GPIO" and "Interrupts" sections.

Table 10: **Interrupt and I/O Performance versus Clock Frequency**

CPU Clock Frequency	Interrupt Latency	Minimum Time between Output Samples	Minimum Time between Input Samples
100 MHz	1.9 µs	100 ns	20 ns
200 MHz	0.94 µs	50 ns	10 ns
300 MHz	0.63 µs	33.3 ns	6.7 ns
400 MHz	0.47 µs	25 ns	5 ns
450 MHz(1)	0.42 µs	22.2 ns	4.4 ns

Notes:

1. CPU clock frequency of 450 MHz is available on Virtex-4 FX (-12 speed grade) devices only. For device data sheets, see "References," page 21.

www.BDTIC.com/XILINX

Resource Utilization

Table 11 and Table 12 illustrate the small footprint of the UltraController-II black-box solution. Resource utilization numbers are the same for both single-processor and dual-processor versions.

Table 11: **Virtex-4 FX Resource Utilization**

Resource	Usage	Percent of FPGA Fabric Used			
		XC4VFX12	XC4VFX40	XC4VFX60	XC4VFX100
Slice Flip-Flops	5	0.046%	0.013%	0.009%	0.006%
LUTs	6	0.055%	0.016%	0.012%	0.007%
BRAMs	0	0.0%	0.0%	0.0%	0.0%

Table 12: **Virtex-II Pro Resource Utilization**

Resource	Usage	Percent of FPGA Fabric Used			
		XC2VP4	XC2VP30	XC2VP70	XC2VP100
Slice Flip-Flops	5	0.083%	0.018%	0.008%	0.006%
LUTs	6	0.100%	0.022%	0.009%	0.007%
BRAMs	0	0.0%	0.0%	0.0%	0.0%

References

1. EDK documentation, http://www.xilinx.com/ise/embedded/edk_docs.htm
   - Xilinx, Inc., UG018: PowerPC 405 Processor Block Reference Guide
   - Xilinx, Inc., UG113: Platform Studio User Guide
   - Xilinx, Inc., PowerPC Processor Reference Guide
2. UltraController website, http://www.xilinx.com/ultracontroller
3. Memec Virtex-4 and Virtex-II Pro LC Development Kits, http://www.memec.com
   - Virtex-4 FX LC UltraController-II board (DS-KIT-4VFX12LC)
   - Virtex-II Pro LC UltraController board (DS-KIT-2VP4LC)
4. ML310 evaluation platform website, http://www.xilinx.com/ml310
   - Xilinx, Inc., UG068: ML310 User Guide
5. Xilinx, Inc., XAPP571: DEBUGHALT Controller for PowerPC Boot and Reset Operations, http://www.xilinx.com/bvdocs/appnotes/xapp571.pdf
6. Xilinx FPGA configuration information
   - Xilinx, Inc., UG071: Virtex-4 Configuration Guide, http://www.xilinx.com/bvdocs/userguides/ug071.pdf
   - Xilinx, Inc., UG012: Virtex-II Pro and Virtex-II Pro X FPGA User Guide (Chapter 4: Configuration), http://www.xilinx.com/bvdocs/userguides/ug012.pdf
7. Xilinx FPGA data sheets
   - Xilinx, Inc., DS112: Virtex-4 Family Overview, http://www.xilinx.com/bvdocs/publications/ds112.pdf
   - Xilinx, Inc., DS302: Virtex-4 Data Sheet: DC and Switching Characteristics, http://www.xilinx.com/bvdocs/publications/ds302.pdf
   - Xilinx, Inc., DS083: Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet, http://www.xilinx.com/bvdocs/publications/ds083.pdf

www.BDTIC.com/XILINX

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/27/05	1.0	Initial Xilinx release.
02/02/05	1.0.1	Added reference to DS-KIT-2VP4LC.
02/25/05	1.0.2	Minor text updates.
08/02/05	1.1	Updated tables and references for Virtex-4 FX support.
08/05/05	1.1.1	Minor text updates. Added references to Xilinx FPGA data sheets.

www.BDTIC.com/XILINX