Hardware/Software Co-design : Software Thread Manager

Michael Finley

EECS 891, Fall 2004
University of Kansas

Table of Contents

Page

Introduction

The goal of this project was to develop a module for use in a larger design which would demonstrate the use of FPGAs in Hardware/Software Co-design applications. The application being developed is a Real Time Operating System for use in a multi-threaded application platform targeted for the Xilinx Virtex-II Pro series of FPGAs. These devices combine one or more Power PC microprocessor cores (PPC 405) and an array of programmable logic cells.

This project focused on a module implementing a Software Thread Manager (SWTM). The SWTM is responsible for tracking the present status of each of the system's software threads and for providing a set of atomic functions that are used to manipulate those threads. Included within the SWTM functionality is a simple FIFO based scheduling mechanism to enable initial evaluation. This aspect of the design was setup to allow a separate Scheduler module to be incorporated later for implementing different scheduling algorithms. The design was specified using VHDL. A complete listing of the source files is included later in this report.

Background

Embedded system design has traditionally been an exercise in balancing seemingly opposing goals and real world constraints. The environment in which many of these systems are used is often limited in size, forcing this same constraint upon the implementation of the system. Other constraints that tend to complicate the solution include the desire to minimize the costs and the development time. How to achieve a high degree of specialized functionality in a limited space, with the lowest cost, and in the smallest time, is an example of these opposing goals.

Semiconductor device manufacturers typically produce generalized functions to be able to reach the largest market possible. Using these “building block” functions a designer can create the specialized functionality required by the system to be designed. The individual packaging of each device however forces the overall size of the system to be much larger than it would be if specialized devices, targeted directly for the application were available. The advent of the “micro-controller” helped address this need by combining a microprocessor with other select functions that would typically be required to build a basic system in the same package. Again, the functions included(memory, I/O interfaces, interrupt control) were selected and tailored to reach the broadest set of generalized applications possible. While a step in the right direction, these devices still could not address the diverse range of specialized applications that might benefit from the reduction in size.

Another solution industry has offered is the option of having an “Application Specific Integrated Circuit(ASIC)” produced. A system designer can specify the functionality to be incorporated into the device, which is then produced. The very specialized nature of these devices however tends to limit there use to the system for which they were designed and the overall quantity produced is low enough that the cost for each, and the initial setup charge, is prohibitive for most applications.

The advent of the Hybrid CPU/FPGA device has dramatically addressed the need for a high degree of specialized functionality in a small space. The fact that the FPGA configuration is stored internally in RAM memory also allows the device to be (re)programmed as needed to implement the specialized functionality required without sacrificing the device’s adaptability to the broadest range of applications. The device manufacturer’s need to reach the broadest market possible is preserved, and system designers are ecstatic. In a very real sense, system designers can now readily design their own application specific micro-controller using the basic device and libraries of building block functions.

Another important consideration in the design of a system is the amount of time and effort that will be required to accurately perform that design. The ability to reuse portions of an already existing design can have a substantial impact in reducing the required time and effort. An example of the usefulness of this approach is the use of an operating system within the system's programming. The operating system is a collection of services made available to the system's application programmer and includes an abstracted interface between the application and the hardware. The application programmer takes advantage of the time and effort that was required to produce the operating system by using it's services wherever possible thus allowing him/her to focus solely on the design of the new application. To the degree that the operating system is used as an interface between the user and the application, a certain familiarity and ease of use is also produced. This approach is partially responsible for the success and widespread use of today's personal computers.

Real time embedded systems could also benefit from adopting a similar approach. Including an operating system whose services are designed to create a reusable platform would allow a system designer to focus on the unique requirements of this particular system. When a company adopts this approach while designing a variety of related systems a number of benefits can result including; reduced time to market, a decrease in the variety of components to be procured allowing "bulk" purchases, and even a decreased reliance on the particular components that are used to create the platform since the operating system forms an abstraction layer between the application(s) and the hardware. Should the hardware platform have to be changed, the operating system could be updated to account for the new hardware and the application(s) might not be impacted. The time and effort required to update the operating system would then benefit and be shared by all of the systems that are based off the original platform.

The nature of the platform to be developed, and hopefully shared across multiple designs, is also an important consideration, as it will impact every design that utilizes it. Being easily adaptable to a diverse set of applications will encourage and enable its reuse and extend its "lifetime." The degree to which it encourages, and perhaps even imposes, certain architectural styles is also a benefit, as it can tend to raise the quality of all applications built upon it. While the ability to reuse a given platform across multiple designs is important, the ability to reuse certain portions of a given application can also be helpful when designing additional applications. One platform that encourages this ability, and the one chosen for the system of which this project is a component, is the multi-threaded application platform. Applications developed for this type of architecture divide the system's processing into individual "threads of execution." Each thread tends to be highly focused on minimal issues and is as independent as possible from the other threads within the application. This approach encourages modularity and allows the designer of a given thread to focus on a subset of the issues addressed by the system as a whole.

A design based on this programming style has quite a challenge before it in keeping track of all the threads, their current status, shared data structures, and the scheduling of when and why a given thread should run. Including services within the platform to address these details simplifies the design of applications and abstracts these low-level details from the application's programmer(s). This project implements the services required to create, delete, and modify/track the current status of the system's software threads.

Many of the modules being developed to create this platform, including this project, are utilizing Hardware/Software Co-design methodologies. Taking advantage of the ever-increasing size and density of today's programmable logic devices, functionality traditionally implemented in software is being implemented in hardware instead.

Functional Requirements

As the first step in planning for this portion of the system, and given the centralized role it plays, a series of meetings were held to discuss and ultimately specify the functional requirements for the SWTM module. A number of people attended and shared their thoughts and expertise. The "Acknowledgements" section at the end of this report includes the names of those people that attended. The ultimate outcome of these meetings has been summarized and is presented in the following section.

In the following discussion the functions supported by the SWTM are referred to as registers due to their similarity to standard memory mapped storage registers. One point of particular concern in the early planning of this module was the need to have atomic access to these functions. Most of these functions result in one or more read-modify-write accesses being performed on the data structures maintaining the present status of each thread. To invoke a standard function typically a parameter is passed to the function, which then computes a result and returns this value to the caller. To make this sequence atomic the function is accessed with a read instruction where the parameter is passed in a subset of the address lines used to specify which function is to be performed. The hardware extends the length of the bus cycle by as many cycles as necessary to complete the function before returning the result within a single atomic sequence.

SW Thread Manager Access Registers

The following registers are used to access a variety of functions built in to the FPGA. All registers are accessed as having a 32-bit width even though the implementation may only use a subset of the full 32 bits. In these cases, the least significant bits are utilized, and the upper bits are padded with zeros for read operations and ignored for write operations. Write operations performed on read only registers generate an exception to the CPU.

The depth of a given register specifies the number of successive, 32 bit locations that are utilized by this register. Registers having a depth greater than one are utilizing the least significant address lines to specify a parameter to be passed to a hardware function associated with this register. To keep each access on an even 32 bit boundary, the parameter to be passed in must be multiplied by 4 before adding to the base address of the register.

Example : a particular register accepts an 8 bit parameter encoded into the address used to access the register (depth=256). The address corresponding to a parameter of zero equals the base address of the register. The address corresponding to a parameter of one equals the base address of the register plus four.

References in the following descriptions to a thread's status refer to that thread's entry in the Thread ID Table and utilize the format specified in the Thread ID Table Encoding on page 10.

References in the following descriptions to “queue” refer to the Ready to Run Queue, implemented as a linked list within the Thread ID Table, where each thread has a pointer to the next thread in the queue.

que_length read only, depth = 1

This register holds the number of thread IDs currently in the queue. This register is updated automatically when a thread is added to, or removed from the queue and can be read at any time without side effects.

add_thread read only, depth = 256

Reading this register adds the encoded thread ID to the queue. Before the thread is added the following checks are made;

The queue must not be full ,
The thread’s status must be : used, ~exited, and ~queued

If either test fails, the value returned is the thread’s current status with the ERR_BIT set to indicate that the operation was unsuccessful. If the thread is successfully added to the queue, the previous thread’s Next field is set to point to this thread, this thread’s status is updated to show that it is now queued (used, ~exited, queued), and the value returned is zero.

idle_thread read/write, depth = 1

This register is written to specify the ID of the thread to be used as the “idle thread.” This value is returned when the software requests the ID of the next thread in the queue and the queue is empty. When reading this register, the ID is returned in bits 23..30 , with bit 31 set for all error conditions, else cleared.

next_thread read only, depth = 1

This register is read to retrieve the ID of the next thread in the queue. The hardware copies the value returned to the current_thread register, removes the thread’s ID from the queue, and updates the thread’s status to show that it is no longer in the queue. (used, ~exited, ~queued) If the queue is empty, the contents of the idle_thread register are returned instead. The thread ID is returned in bits 23..30 with bit 31 set for all error conditions.

current_thread read only, depth = 1

This register holds the ID of the thread currently running on the CPU and can be read at any time without side effects. Its contents are updated automatically when the next_thread register is read, and is used by the hardware to identify the current thread and to qualify certain operations. The ID is returned in bits 23..30, with bit 31 set for all error conditions, else cleared.

create_thread_detached read only, depth = 1

This register is read to retrieve an unused ID number in preparation for creating a new, detached thread. If no IDs are available the read operation returns zero + the ERR_BIT set. If an ID is available, the status of the new thread is updated to be; used, ~exited, ~queued, ~joined, detached, the thread's parent ID field(PID) is set to zero, and the ID number is returned in bits 23…30. (all other bits = 0).

create_thread_joinable read only, depth = 1

This register is read to retrieve an unused ID number in preparation for creating a new joinable thread. The new thread is not yet joined but must be later to prevent memory leaks. If no IDs are available the read operation returns zero + the ERR_BIT set. If an ID is available, the status of the new thread is updated to be; used, ~exited, ~queued, ~joined, ~detached, the thread's parent ID field(PID) is set to the value in the current_thread register, and the ID number is returned in bits 23…30. (all other bits = 0).

exit_thread read only, depth = 1

This register is read to update the status of the currently running thread to show that it has terminated. The effects of this operation vary depending on the current status of the thread and are summarized below.

If the thread's status shows it to be detached, the thread ID is de-allocated, changing its status to, ~used, ~exited, ~queued, ~joined, ~detached, and its PID field to zero. A value of zero is returned.

If the thread's status shows it to not be detached, the thread's status is updated to show that it has exited(used,exited) and then the following steps are performed;

```
If the thread's PID != 0, and the thread has been joined(joined=1) ,
    If the parent thread's status = used, ~exited, ~queued, and the queue is not full,
        set the parent thread's status to queued, add the parent thread's ID to the queue, and
            return zero.
    Else return the parent's status + ERR_BIT.
Else return 0
```

read_thread read (write if in debug "stop" mode), depth = 256

Reading this register returns the encoded thread IDs row from the Thread ID Table without producing any side effects. The ERR_BIT and auxiliary status bits (E1, E2, E3) are returned as zeros. If the design is in debug, stopped mode, writing to register sets this row value to the data written.

yield_thread read only, depth = 1

This register is read to place the current thread back on the queue and then return the ID of the next thread in the queue, which is also copied to the current_thread register. If the queue is empty, the current thread is not re-added to the queue, and its ID is returned instead.

clear_thread read only, depth = 256

Reading this register de-allocates the encoded thread ID by setting the thread's status to, ~used, ~exited, ~queued, ~joined, ~detached, and the thread's PID field to zero. Before the thread is de-allocated the following check is made,

The thread's PID field must equal the contents of the current_thread register

If this test fails, the thread's status is left unchanged and the value returned is the thread's current status with the ERR_BIT set to indicate that the operation was unsuccessful. If successful, a value of zero is returned.

join_thread read only, depth = 256

This register is read to join the encoded thread ID(child) to the current thread(parent). The child thread's status is first checked to verify that it is, used, ~joined, ~detached, and that its PID equals the contents of the current_thread register. If any of these tests fail, the child's status + ERR_BIT is returned. If all tests passed, the child's status is checked to see if it has already exited. If it has, the value 0 + THREAD_ALREADY_TERMINATED is returned, else the child's status is changed to joined, and a value of zero is returned.

detach_thread read only, depth = 256

This register is read to detach the encoded thread ID(child) from the current thread(parent). The child thread's status is first checked to verify that it is, used, ~exited, ~joined, ~detached, and that its PID equals the contents of the current_thread register. If any of these tests fail, the child's status + ERR_BIT is returned. If all tests passed, the child's status is changed to detached, the child's PID is set to zero, and a value of zero is returned.

The following registers are included to enhance troubleshooting the system's operation.

soft_start read, write, depth = 1

Writing any value to this register de-asserts the Soft_Stop and all Soft_Reset signals. Reading this register returns all zeros.

soft_stop read, write, depth = 1

Writing any value to this register asserts the Soft_Stop signal, used by all system IP's to halt operation. Reading this register returns the value of the Soft_Stop signal in the LSB, all other bits = zero.

soft_reset read, write, depth = 1

Writing to this register selectively asserts a number of soft_reset signals depending on the data written. Each bit corresponds to a particular IP; User IP(27), SpinLock(28), Semaphores(29), Scheduler(30), SWTM(31=LSB).

Reading this register returns all zeros with a one in any position(s) corresponding to an IP that failed to signal completion before an encoded time delay. (default delay = 4096 clock cycles)

exception_cause, exception_address read only, depth = 1

Certain events cause a critical exception to be raised. These events should not occur during normal operation and are indicative of a failure within the system. When one of these events is detected, a code representing the particular type of event is stored in the exception_cause register and the associated address that was being accessed is stored in the exception_address register. The system SW can then read these registers to determine the reason for the interrupt. The causes and codes are listed below.

Cause	Code returned
Write to Read Only Register	1
Undefined Address	2
Soft Reset Failure	3

Thread ID Table Encoding (each row)

```
 0        7  8          15 16         23  24  25   26  27  28  29  30   31
-------------|-------------|-------------|-----------------------------------
 Thread ID   |    Next     |     PID     |  D   J   S1  S0  E3  E2  E1  ERR_BIT
-------------|-------------|-------------|-----------------------------------
```

Each row is logically divided into two sections, the Next Available ID and the Thread Status.

The Next Available ID is contained in bits 0 through 7 and specifies a thread ID, ranging from 0..255. The collection of this section in all rows collectively form the Next Available Thread ID Stack. On initialization, rows 0 through 255 are programmed with the value 0 through 255 respectively. A separate pointer is maintained by the hardware which points to the top of the stack. As a thread is created the value at the top of the stack is used for the thread ID of the newly created thread and the stack pointer is advanced. Whenever a thread is de-allocated the thread ID of that thread is put back on to the stack at the present location pointed to by the stack pointer. As threads can come into and out of existence in any order, the value in any particular row can take on any value and won’t necessarily remain in the initial ordering of 0..255. The value stored in this field is not related in any way to the thread whose status is stored in this same row.

The Thread Status is stored in bits 8 through 31 and uses the encoding shown and detailed below. The “Next” field holds the thread ID of the next thread to run when this thread exits, yields, or is preempted by a scheduling timer event. The “PID” field holds the ID# of this thread’s parent thread. The remaining bits are encoded as shown below. Note that no interpretation of bits 28..30 is made unless bit 31 is set. Stated otherwise, the hardware will always set bit 31 to indicate an error condition and then encode additional information regarding that error in bits 28..30.

Thread Status

S1	S0	
0	0	unused
0	1	used, exited
1	0	used, not exited, not queued
1	1	used, not exited, queued

J = 0	this thread has not been joined
J = 1	this thread has been joined,
D = 0	this thread is not detached
D = 1	this thread is detached
PID =	this thread's Parent ID
Next =	next thread in queue

E3	E2	E1	
0	0	0	*error in thread PID,status , or no IDs available*
0	0	1	THREAD_ALREADY_TERMINATED
0	1	0	THREAD_ALREADY_QUEUED
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

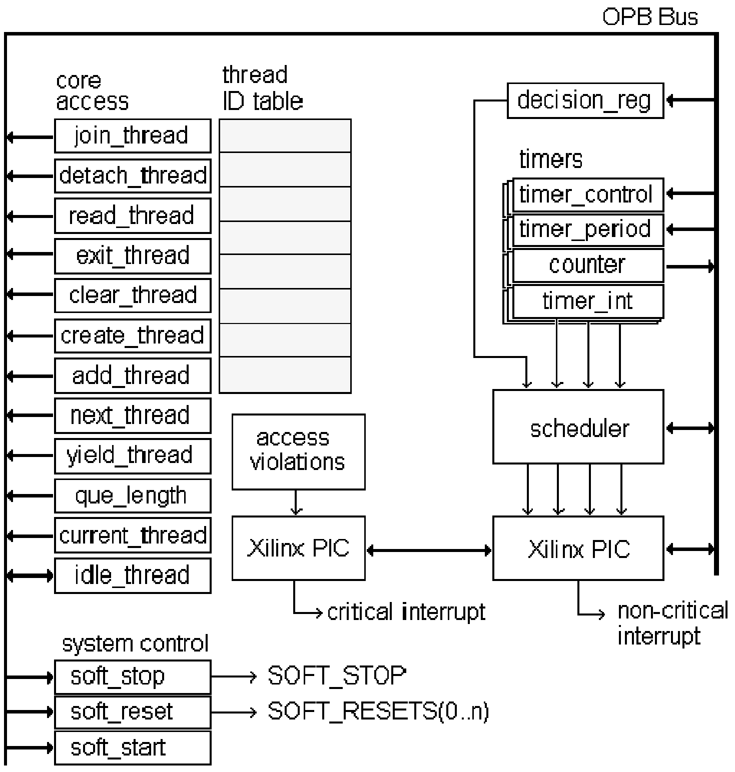
ERR_BIT = 0	no error occurred
ERR_BIT = 1	set for all errors

SWTM Memory Map

	address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
clear_thread	01020000	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**0**	**0**	**0**	**0**	**0**	*T*	*T*	*T*	*T*	*T*	*T*	*T*	*T*	0	0
join_thread	01020400	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**0**	**0**	**0**	**0**	**1**	*T*	*T*	*T*	*T*	*T*	*T*	*T*	*T*	0	0
detach_thread	01020800	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**0**	**0**	**0**	**1**	**0**	*T*	*T*	*T*	*T*	*T*	*T*	*T*	*T*	0	0
read_thread	01020C00	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**0**	**0**	**0**	**1**	**1**	*T*	*T*	*T*	*T*	*T*	*T*	*T*	*T*	0	0
add_thread	01021000	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**0**	**0**	**1**	**0**	**0**	*T*	*T*	*T*	*T*	*T*	*T*	*T*	*T*	0	0
create_thread_joinable	01021400	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**0**	**0**	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0	0
create_thread_detached	01021800	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**0**	**0**	**1**	**1**	**0**	0	0	0	0	0	0	0	0	0	0
exit_thread	01021C00	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**0**	**0**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0
next_thread	01022000	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**0**	**1**	**0**	**0**	**0**	0	0	0	0	0	0	0	0	0	0
yield_thread	01022400	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**0**	**1**	**0**	**0**	**1**	0	0	0	0	0	0	0	0	0	0
current_thread	01024000	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**1**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	0	0	0	0
idle_thread	01024400	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**1**	**0**	**0**	**0**	**1**	0	0	0	0	0	0	0	0	0	0
que_length	01024800	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**1**	**0**	**0**	**1**	**0**	0	0	0	0	0	0	0	0	0	0
exception_address	01024C00	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**1**	**0**	**0**	**1**	**1**	0	0	0	0	0	0	0	0	0	0
exception_register	01025000	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**1**	**0**	**1**	**0**	**0**	0	0	0	0	0	0	0	0	0	0
core_start	01025400	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**1**	**0**	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0	0
core_stop	01025800	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**1**	**0**	**1**	**1**	**0**	0	0	0	0	0	0	0	0	0	0
core_reset	01025C00	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**1**	**0**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0
core end	01027FFF	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**0**	**1**	**1**	**1**	**1**	**1**	1	1	1	1	1	1	1	1	1	1
MIR start	01028000	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MIR end	0102FFFF	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1. The least significant 10 address lines are zero for any base address.
2. Bits 16..21 are decoded internally to identify which register is being accessed.
3. Bits 30,31 are zero for ALL accesses to keep data transfers as a single, 32 bit bus cycle.
4. *T* in bits 22..29 for the first five registers specifies the thread ID of the thread to be manipulated.

FPGA Core : Block Diagram



The diagram to the right shows the initial platform used to test the operation of the SWTM. The timers, decision_reg, scheduler, and Xilinx PICs are not part of the SWTM but were added to create a minimal platform for evaluation and further development.

Implementation

The SWTM module is specified in two VHDL files; opb_threadCore and user_logic. The design is based on a reference design provided by Xilinx intended as a starting point for developing re-useable modules; Slave Services Package 0 (SSP0). This design is intended for applications not needing to be able to function as a bus master.

The top level module, opb_threadCore.vhd, establishes the interface to the PowerPC microprocessor, the rest of the system, and instantiates any lower level modules required by the design. For this design three modules are required;

1. opb_ipif_ssp0 - the slave services package for the OPB bus.
2. user_logic - the behavioral definitions for the SWTM component.
3. RAMB16_S36_S36 - a dual port BRAM component, 512 rows by 36 bits, used to create the Thread ID Table. This design only uses one of the two ports. The second port is left open in anticipation that a Scheduler IP module could utilize this port to directly modify the "Next" pointers of each thread to control the order in which the threads are scheduled to run.

The two VHDL files are listed in the following sections.

File : opb_threadCore.vhd

```
-- $Id: opb_core_ssp0_ref.vhd,v 1.1 2003/06/26 14:10:56 anitas Exp $
-- opb_threadCore.vhd
--
--  ***********************************************************************
--  **  Copyright(C) 2003 by Xilinx, Inc. All rights reserved.           **
--  **                                                                   **
--  **  This text contains proprietary, confidential                     **
--  **  information of Xilinx, Inc. , is distributed by                  **
--  **  under license from Xilinx, Inc., and may be used,                **
--  **  copied and/or disclosed only pursuant to the terms               **
--  **  of a valid license agreement with Xilinx, Inc.                   **
--  **                                                                   **
--  **  Unmodified source code is guaranteed to place and route,         **
--  **  function and run at speed according to the datasheet             **
--  **  specification. Source code is provided "as-is", with no          **
--  **  obligation on the part of Xilinx to provide support.             **
--  **                                                                   **
--  **  Xilinx Hotline support of source code IP shall only include      **
--  **  standard level Xilinx Hotline support, and will only address     **
--  **  issues and questions related to the standard released Netlist    **
--  **  version of the core (and thus indirectly, the original core source). **
--  **                                                                   **
--  **  The Xilinx Support Hotline does not have access to source        **
--  **  code and therefore cannot answer specific questions related      **
--  **  to source HDL. The Xilinx Support Hotline will only be able      **
--  **  to confirm the problem in the Netlist version of the core.       **
--  **                                                                   **
--  **  This copyright and support notice must be retained as part       **
--  **  of this text at all times.                                       **
--  ***********************************************************************

---------------------------------------------------------------------------
--  opb_threadCore v3.0
--
--  Author     : Michael Finley
--  Date       : 7/26/04
---------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

library opb_ipif_ssp0_v1_00_b;
use opb_ipif_ssp0_v1_00_b.all;

library opb_threadCore;
use opb_threadCore.all;

---------------------------------------------------------------------------
-- ports
---------------------------------------------------------------------------

-- Definition of Generics:
--      C_BASEADDR             -- User logic base address
--      C_HIGHADDR             -- User logic high address
--      C_MIR_BASEADDR         -- Base address of MIR/Reset register
--      C_MIR_HIGHADDR         -- High address of MIR/Reset register
--      C_USER_ID_CODE         -- User ID to place in MIR/Reset register
--      C_OPB_DWIDTH           -- OPB data bus width
--      C_OPB_AWIDTH           -- OPB address bus width
--      C_FAMILY               -- Target FPGA architecture
--      C_RESET_TIMEOUT        -- Soft Reset max clock cycles else timeout
--
-- Definition of Ports:
--  OPB Bus
--      OPB_ABus               -- OPB address bus
--      OPB_BE                 -- OPB byte enables
--      OPB_Clk                -- OPB clock
--      OPB_DBus               -- OPB data bus
--      OPB_RNW                -- OPB read not write
--      OPB_Rst                -- OPB reset
--      OPB_select             -- OPB Select
--      OPB_seqAddr            -- OPB sequential address
--      Sln_DBus               -- Slave read bus
--      Sln_errAck             -- Slave Error acknowledge
--      Sln_retry              -- Slave retry
--      Sln_toutSup            -- Slave Timeout Suppress
--      Sln_xferAck            -- Slave transfer acknowledge

-- User Logic
--     Access_Intr            -- access violation interrupt
--     Soft_Reset(s)          -- clear state machines, data structures
--     Soft_Reset(s)_Done     -- reset done response(s) from IPs
--     Soft_Stop              -- halt state machines
---------------------------------------------------------------------------

---------------------------------------------------------------------------
-- entity
---------------------------------------------------------------------------

entity opb_threadCore is
  generic
  (
    C_BASEADDR         : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_HIGHADDR         : std_logic_vector(0 to 31) := X"00000000";
    C_MIR_BASEADDR     : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_MIR_HIGHADDR     : std_logic_vector(0 to 31) := X"00000000";
    C_USER_ID_CODE     : integer                   := 3;
    C_OPB_AWIDTH       : integer                   := 32;
    C_OPB_DWIDTH       : integer                   := 32;
    C_FAMILY           : string                    := "virtex2";
    C_RESET_TIMEOUT    : natural                   := 4096
  );
  port
  (
    --Required OPB bus ports, do not add to or delete
    OPB_ABus     : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE       : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_Clk      : in  std_logic;
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW      : in  std_logic;
    OPB_Rst      : in  std_logic;
    OPB_select   : in  std_logic;
    OPB_seqAddr  : in  std_logic;
    Sln_DBus     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck   : out std_logic;
    Sln_retry    : out std_logic;
    Sln_toutSup  : out std_logic;
    Sln_xferAck  : out std_logic;

    Access_Intr  : out std_logic;

    Scheduler_Reset      : out std_logic;
    Scheduler_Reset_Done : in  std_logic;

    Semaphore_Reset      : out std_logic;
    Semaphore_Reset_Done : in  std_logic;

    SpinLock_Reset       : out std_logic;
    SpinLock_Reset_Done  : in  std_logic;

    User_IP_Reset        : out std_logic;
    User_IP_Reset_Done   : in  std_logic;

    Soft_Stop            : out std_logic;

    Current_Thread_ID : out std_logic_vector(0 to 7);

    Next_Thread_ID    : in  std_logic_vector(0 to 7);
    Dequeue_Request   : out std_logic;
    Next_Thread_Valid : in  std_logic;

    Thread_ID_2_Sched : out std_logic_vector(0 to 7);
    Enqueue_Request   : out std_logic;
    Enqueue_Busy      : in  std_logic;

    DOB   : out std_logic_vector(0 to 31);
    DOPB  : out std_logic_vector(0 to 3);
    ADDRB : in  std_logic_vector(0 to 8);
    CLKB  : in  std_logic;
    DIB   : in  std_logic_vector(0 to 31);
    DIPB  : in  std_logic_vector(0 to 3);
    ENB   : in  std_logic;
    SSRB  : in  std_logic;
    WEB   : in  std_logic
  );

  --fan-out attributes for XST
  attribute MAX_FANOUT                : string;
  attribute MAX_FANOUT   of OPB_Clk   : signal is "10000";
  attribute MAX_FANOUT   of OPB_Rst   : signal is "10000";

end entity opb_threadCore;


------------------------------------------------------------------------------
-- architecture
------------------------------------------------------------------------------

architecture imp of opb_threadCore is

  -- include OPB-In and OPB-Out pipeline registers
  --
  constant PIPELINE_MODEL : integer   := 5;


  --IP Interconnect (IPIC) signal list --do not delete
  --
  signal Bus2IP_Addr        : std_logic_vector(0 to C_OPB_AWIDTH-1);
  signal Bus2IP_BE          : std_logic_vector(0 to C_OPB_DWIDTH/8-1);
  signal Bus2IP_Burst       : std_logic;
  signal Bus2IP_Clk         : std_logic;
  signal Bus2IP_CS          : std_logic;
  signal Bus2IP_Data        : std_logic_vector(0 to C_OPB_DWIDTH-1);
  signal Bus2IP_RdCE        : std_logic;
  signal Bus2IP_Reset       : std_logic;
  signal Bus2IP_RNW         : std_logic;
  signal Bus2IP_WrCE        : std_logic;
  signal IP2Bus_Ack         : std_logic;
  signal IP2Bus_Data        : std_logic_vector(0 to C_OPB_DWIDTH-1);
  signal IP2Bus_Error       : std_logic;
  signal IP2Bus_PostedWrInh : std_logic;
  signal IP2Bus_Retry       : std_logic;
  signal IP2Bus_ToutSup     : std_logic;

  --BlockRam interconnect
  --
  signal DOA   : std_logic_vector(0 to 31);
  signal DOPA  : std_logic_vector(0 to 3);
  signal ADDRA : std_logic_vector(0 to 8);
  signal CLKA  : std_logic;
  signal DIA   : std_logic_vector(0 to 31);
  signal DIPA  : std_logic_vector(0 to 3);
  signal ENA   : std_logic;
  signal SSRA  : std_logic;
  signal WEA   : std_logic;
  --  signal DOB   : std_logic_vector(0 to 31);
  --  signal DOPB  : std_logic_vector(0 to 3);
  --  signal ADDRB : std_logic_vector(0 to 8);
  --  signal CLKB  : std_logic;
  --  signal DIB   : std_logic_vector(0 to 31);
  --  signal DIPB  : std_logic_vector(0 to 3);
  --  signal ENB   : std_logic;
  --  signal SSRB  : std_logic;
  --  signal WEB   : std_logic;


component RAMB16_S36_S36
  --
  --  thread ID table : dualport BRAM, 512 rows by 36-bits
  --
  --    port_A  : thread registers,ports and control
  --    port_B  : unused
  --
  generic (
    INIT_00  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_01  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_02  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_03  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_04  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_05  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_06  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_07  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_08  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_09  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0A  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0B  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0C  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0D  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0E  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0F  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_10  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_11  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_12  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_13  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_14  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_15  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_16  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_17  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_18  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_19  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_1A  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_1B  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_1C  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_1D  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_1E  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_1F  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_20  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_21  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_22  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_23  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_24  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_25  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_26  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_27  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_28  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_29  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_2A  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_2B  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_2C  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_2D  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_2E  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_2F  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_30  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_31  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_32  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_33  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_34  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_35  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_36  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_37  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_38  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_39  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_3A  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_3B  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_3C  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_3D  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_3E  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_3F  : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_A   : bit_vector := X"000000000000000000000000000000000000";
    INIT_B   : bit_vector := X"000000000000000000000000000000000000";
    INITP_00 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INITP_01 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INITP_02 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INITP_03 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INITP_04 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INITP_05 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INITP_06 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INITP_07 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    SRVAL_A  : bit_vector := X"000000000000000000000000000000000000";
    SRVAL_B  : bit_vector := X"000000000000000000000000000000000000";
    WRITE_MODE_A : string := "WRITE_FIRST";
    WRITE_MODE_B : string := "WRITE_FIRST"
  );

   -- synthesis translate_on
  port
  (
    DOA   : out STD_LOGIC_VECTOR (0 to 31);
    DOB   : out STD_LOGIC_VECTOR (0 to 31);
    DOPA  : out STD_LOGIC_VECTOR (0 to 3);
    DOPB  : out STD_LOGIC_VECTOR (0 to 3);
    ADDRA : in  STD_LOGIC_VECTOR (0 to 8);
    ADDRB : in  STD_LOGIC_VECTOR (0 to 8);
    CLKA  : in  STD_ULOGIC;
    CLKB  : in  STD_ULOGIC;
    DIA   : in  STD_LOGIC_VECTOR (0 to 31);
    DIB   : in  STD_LOGIC_VECTOR (0 to 31);
    DIPA  : in  STD_LOGIC_VECTOR (0 to 3);
    DIPB  : in  STD_LOGIC_VECTOR (0 to 3);
```

```
    ENA   : in  STD_ULOGIC;
    ENB   : in  STD_ULOGIC;
    SSRA  : in  STD_ULOGIC;
    SSRB  : in  STD_ULOGIC;
    WEA   : in  STD_ULOGIC;
    WEB   : in  STD_ULOGIC
  );
end component RAMB16_S36_S36;


component user_logic is
  --
  --  thread ID table registers/ports and control
  --
  generic (
    C_RESET_TIMEOUT    : natural := 4096
  );
  port
  (
    Bus2IP_Addr  : in  std_logic_vector(0 to 31);
    Bus2IP_Clk   : in  std_logic;
    Bus2IP_CS    : in  std_logic;
    Bus2IP_Data  : in  std_logic_vector(0 to 31);
    Bus2IP_RdCE  : in  std_logic;
    Bus2IP_Reset : in  std_logic;
    Bus2IP_WrCE  : in  std_logic;
    IP2Bus_Ack   : out  std_logic;
    IP2Bus_Data  : out  std_logic_vector(0 to 31);
    IP2Bus_Error       : out  std_logic;
    IP2Bus_PostedWrInh : out  std_logic;
    IP2Bus_Retry       : out  std_logic;
    IP2Bus_ToutSup     : out  std_logic;
    DOA   : in  std_logic_vector(0 to 31);
    DOPA  : in  std_logic_vector(0 to 3);
    ADDRA : out std_logic_vector(0 to 8);
    CLKA  : out std_logic;
    DIA   : out std_logic_vector(0 to 31);
    DIPA  : out std_logic_vector(0 to 3);
    ENA   : out std_logic;
    SSRA  : out std_logic;
    WEA   : out std_logic;
    Access_Intr          : out std_logic;
    Scheduler_Reset      : out std_logic;
    Scheduler_Reset_Done : in  std_logic;
    Semaphore_Reset      : out std_logic;
    Semaphore_Reset_Done : in  std_logic;
    SpinLock_Reset       : out std_logic;
    SpinLock_Reset_Done  : in  std_logic;
    User_IP_Reset        : out std_logic;
    User_IP_Reset_Done   : in  std_logic;
    Soft_Stop            : out std_logic;
    Current_Thread_ID    : out std_logic_vector(0 to 7);
    Next_Thread_ID       : in  std_logic_vector(0 to 7);
    Dequeue_Request      : out std_logic;
    Next_Thread_Valid    : in  std_logic;
    Thread_ID_2_Sched    : out std_logic_vector(0 to 7);
    Enqueue_Request      : out std_logic;
    Enqueue_Busy         : in  std_logic
  );
end component user_logic;


component opb_ipif_ssp0 is
  --
  --  OPB slave services package, ver.0
  --
  generic
  (
    C_BASEADDR         : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_HIGHADDR         : std_logic_vector(0 to 31) := X"00000000";
    C_MIR_BASEADDR     : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_MIR_HIGHADDR     : std_logic_vector(0 to 31) := X"00000000";
    C_USER_ID_CODE     : integer  := 1;
    C_PIPELINE_MODEL   : integer  := 4;
    C_OPB_AWIDTH       : integer  := 32;
    C_OPB_DWIDTH       : integer  := 32;
    C_FAMILY           : string   := "virtex2"
  );
  port
  (
    OPB_ABus           : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE             : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_Clk            : in  std_logic;
    OPB_DBus           : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW            : in  std_logic;
    OPB_Rst            : in  std_logic;
    OPB_select         : in  std_logic;
    OPB_seqAddr        : in  std_logic;
    Sln_DBus           : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck         : out std_logic;
    Sln_retry          : out std_logic;
    Sln_toutSup        : out std_logic;
    Sln_xferAck        : out std_logic;
    Bus2IP_Addr        : out std_logic_vector(0 to C_OPB_AWIDTH-1);
    Bus2IP_BE          : out std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    Bus2IP_Burst       : out std_logic;
    Bus2IP_Clk         : out std_logic;
    Bus2IP_CS          : out std_logic;
    Bus2IP_Data        : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Bus2IP_RdCE        : out std_logic;
    Bus2IP_Reset       : out std_logic;
    Bus2IP_RNW         : out std_logic;
    Bus2IP_WrCE        : out std_logic;
    IP2Bus_Ack         : in  std_logic;
    IP2Bus_Data        : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    IP2Bus_Error       : in  std_logic;
    IP2Bus_PostedWrInh : in  std_logic;
    IP2Bus_Retry       : in  std_logic;
    IP2Bus_ToutSup     : in  std_logic
  );
end component opb_ipif_ssp0;

------------------------------------------------------------------------------
begin
------------------------------------------------------------------------------

OPB_IPIF_SSP0_I : opb_ipif_ssp0
  generic map
  (
    C_BASEADDR         => C_BASEADDR,
    C_HIGHADDR         => C_HIGHADDR,
    C_MIR_BASEADDR     => C_MIR_BASEADDR,
    C_MIR_HIGHADDR     => C_MIR_HIGHADDR,
    C_USER_ID_CODE     => C_USER_ID_CODE,
    C_PIPELINE_MODEL   => PIPELINE_MODEL,
    C_OPB_AWIDTH       => C_OPB_AWIDTH,
    C_OPB_DWIDTH       => C_OPB_DWIDTH,
    C_FAMILY           => C_FAMILY
  )
  port map
  (
    OPB_ABus           => OPB_ABus,
    OPB_BE             => OPB_BE,
    OPB_Clk            => OPB_Clk,
    OPB_DBus           => OPB_DBus,
    OPB_RNW            => OPB_RNW,
    OPB_Rst            => OPB_Rst,
    OPB_select         => OPB_select,
    OPB_seqAddr        => OPB_seqAddr,
    Sln_DBus           => Sln_DBus,
    Sln_errAck         => Sln_errAck,
    Sln_retry          => Sln_retry,
    Sln_toutSup        => Sln_toutSup,
    Sln_xferAck        => Sln_xferAck,
    Bus2IP_Addr        => Bus2IP_Addr,
    Bus2IP_BE          => Bus2IP_BE,
    Bus2IP_Burst       => Bus2IP_Burst,
    Bus2IP_Clk         => Bus2IP_Clk,
    Bus2IP_CS          => Bus2IP_CS,
    Bus2IP_Data        => Bus2IP_Data,
    Bus2IP_RdCE        => Bus2IP_RdCE,
    Bus2IP_Reset       => Bus2IP_Reset,
    Bus2IP_RNW         => Bus2IP_RNW,
    Bus2IP_WrCE        => Bus2IP_WrCE,
    IP2Bus_Ack         => IP2Bus_Ack,
    IP2Bus_Data        => IP2Bus_Data,
    IP2Bus_Error       => IP2Bus_Error,
    IP2Bus_PostedWrInh => IP2Bus_PostedWrInh,
    IP2Bus_Retry       => IP2Bus_Retry,
    IP2Bus_ToutSup     => IP2Bus_ToutSup
  );

USER_LOGIC_I : user_logic
  generic map
  (
    C_RESET_TIMEOUT => C_RESET_TIMEOUT
  )
  port map
  (
    Bus2IP_Addr  => Bus2IP_Addr,
    Bus2IP_Clk   => Bus2IP_Clk,
    Bus2IP_CS    => Bus2IP_CS,
    Bus2IP_Data  => Bus2IP_Data,
    Bus2IP_Reset => Bus2IP_Reset,
    Bus2IP_RdCE  => Bus2IP_RdCE,
    Bus2IP_WrCE  => Bus2IP_WrCE,
    IP2Bus_Ack   => IP2Bus_Ack,
    IP2Bus_Data  => IP2Bus_Data,
    IP2Bus_Retry => IP2Bus_Retry,
    IP2Bus_Error => IP2Bus_Error,
    IP2Bus_ToutSup     => IP2Bus_ToutSup,
    IP2Bus_PostedWrInh => IP2Bus_PostedWrInh,
    DOA   => DOA,
    DOPA  => DOPA,
    ADDRA => ADDRA,
    CLKA  => CLKA,
    DIA   => DIA,
    DIPA  => DIPA,
    ENA   => ENA,
    SSRA  => SSRA,
    WEA   => WEA,
    Access_Intr => Access_Intr,
    Scheduler_Reset      => Scheduler_Reset,
    Scheduler_Reset_Done => Scheduler_Reset_Done,
    Semaphore_Reset      => Semaphore_Reset,
    Semaphore_Reset_Done => Semaphore_Reset_Done,
    SpinLock_Reset       => SpinLock_Reset,
    SpinLock_Reset_Done  => SpinLock_Reset_Done,
    User_IP_Reset        => User_IP_Reset,
    User_IP_Reset_Done   => User_IP_Reset_Done,
    Soft_Stop            => Soft_Stop,
    Current_Thread_ID    => Current_Thread_ID,
    Next_Thread_ID       => Next_Thread_ID,
    Dequeue_Request      => Dequeue_Request,
    Next_Thread_Valid    => Next_Thread_Valid,
    Thread_ID_2_Sched    => Thread_ID_2_Sched,
    Enqueue_Request      => Enqueue_Request,
    Enqueue_Busy         => Enqueue_Busy
  );

ID_TABLE : RAMB16_S36_S36
  port map
  (
    DOA   => DOA,
    DOB   => DOB,
    DOPA  => DOPA,
    DOPB  => DOPB,
    ADDRA => ADDRA,
    ADDRB => ADDRB,
    CLKA  => CLKA,
    CLKB  => CLKB,
    DIA   => DIA,
    DIB   => DIB,
    DIPA  => DIPA,
    DIPB  => DIPB,
    ENA   => ENA,
    ENB   => ENB,
    SSRA  => SSRA,
    SSRB  => SSRB,
    WEA   => WEA,
    WEB   => WEB
  );

end architecture imp;
```

File : user_logic.vhd

```
--SINGLE_FILE_TAG
-------------------------------------------------------------------------------
-- $Id: user_logic.vhd,v 1.1 2003/06/26 14:10:56 anitas Exp $
-------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
-------------------------------------------------------------------------------
--
--  ***************************************************************************
--  **  Copyright(C) 2003 by Xilinx, Inc. All rights reserved.             **
--  **                                                                     **
--  **  This text contains proprietary, confidential                       **
--  **  information of Xilinx, Inc. , is distributed by                    **
--  **  under license from Xilinx, Inc., and may be used,                  **
--  **  copied and/or disclosed only pursuant to the terms                 **
--  **  of a valid license agreement with Xilinx, Inc.                     **
--  **                                                                     **
--  **  Unmodified source code is guaranteed to place and route,           **
--  **  function and run at speed according to the datasheet               **
--  **  specification. Source code is provided "as-is", with no            **
--  **  obligation on the part of Xilinx to provide support.               **
--  **                                                                     **
--  **  Xilinx Hotline support of source code IP shall only include        **
--  **  standard level Xilinx Hotline support, and will only address       **
--  **  issues and questions related to the standard released Netlist      **
--  **  version of the core (and thus indirectly, the original core source). **
--  **                                                                     **
--  **  The Xilinx Support Hotline does not have access to source          **
--  **  code and therefore cannot answer specific questions related        **
--  **  to source HDL. The Xilinx Support Hotline will only be able        **
--  **  to confirm the problem in the Netlist version of the core.         **
--  **                                                                     **
--  **  This copyright and support notice must be retained as part         **
--  **  of this text at all times.                                         **
--  ***************************************************************************
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_misc.all;
library Unisim;
```

```
use Unisim.all;

---------------------------------------------------------------------------
--
--  Title    SW Thread Manager
--  Author   Mike Finley
--  Date     7/26/04
--  Version  3.0.1
--
---------------------------------------------------------------------------

---------------------------------------------------------------------------
-- Port declarations
---------------------------------------------------------------------------
-- Definition of Ports:
--  IPIC
--      Bus2IP_Addr         -- Bus to IP address
--      Bus2IP_Clk          -- Bus to IP clock
--      Bus2IP_CS           -- Bus to IP chip select
--      Bus2IP_Data         -- Bus to IP data bus
--      Bus2IP_RdCE         -- Bus to IP read chip enable
--      Bus2IP_Reset        -- Bus to IP reset
--      Bus2IP_WrCE         -- Bus to IP write chip enable
--      IP2Bus_Ack          -- IP to Bus data acknowledge
--      IP2Bus_Data         -- IP to Bus data bus
--      IP2Bus_Error        -- IP to Bus error during transaction
--      IP2Bus_PostedWrInh  -- IP to Bus disable bursts
--      IP2Bus_Retry        -- IP to Bus retry transaction
--      IP2Bus_ToutSup      -- IP to Bus time out suppress
--
--  BRAM portA
--      DOA                 -- ID table data to IP
--      DOPA                -- ID table data to IP (parity bits)
--      ADDRA               -- IP to ID table address
--      CLKA                -- IP to ID table clock
--      DIA                 -- IP to ID table data
--      DIPA                -- IP to ID table data(parity bits)
--      ENA                 -- IP to ID table enable
--      SSRA                -- IP to ID table sync-set/reset command
--      WEA                 -- IP to ID table write enable
--
--  Debug control
--      Access_Intr             -- Interrupt for access violations
--      Scheduler_Reset         -- Soft reset to Scheduler IP
--      Scheduler_Reset_Done    -- Scheduler done signal to SWTM
--      Semaphore_Reset         -- Soft reset to Semaphore IP
--      Semaphore_Reset_Done    -- Semaphore done signal to SWTM
--      SpinLock_Reset          -- Soft reset to SpinLock IP
--      SpinLock_Reset_Done     -- SpinLock done signal to SWTM
--      User_IP_Reset           -- Soft reset to User_IP
--      User_IP_Reset_Done      -- User_IP done signal to SWTM
--      Soft_Stop        -- stop state machines at appropriate point if 1
--
--  Scheduler IP interconnect
--      Current_Thread_ID       --
--      Next_Thread_ID          --
--      Thread_ID_2_Sched       --
--      Current_Thread_Update   --
--      Enqueue                 --
--      Scheduler_Busy          --
--

---------------------------------------------------------------------------
-- Entity section
---------------------------------------------------------------------------

entity user_logic is
  generic (
    C_RESET_TIMEOUT : natural
  );
  port (
    Bus2IP_Addr  : in  std_logic_vector(0 to 31);
    Bus2IP_Clk   : in  std_logic;
    Bus2IP_CS    : in  std_logic;
    Bus2IP_Data  : in  std_logic_vector(0 to 31);
    Bus2IP_RdCE  : in  std_logic;
    Bus2IP_Reset : in  std_logic;
    Bus2IP_WrCE  : in  std_logic;
    IP2Bus_Ack         : out  std_logic;
    IP2Bus_Data        : out  std_logic_vector(0 to 31);
    IP2Bus_Error       : out  std_logic;
    IP2Bus_PostedWrInh : out  std_logic;
    IP2Bus_Retry       : out  std_logic;
    IP2Bus_ToutSup     : out  std_logic;

    DOA   : in  std_logic_vector(0 to 31);
    DOPA  : in  std_logic_vector(0 to 3);
    ADDRA : out std_logic_vector(0 to 8);
    CLKA  : out std_logic;
    DIA   : out std_logic_vector(0 to 31);
    DIPA  : out std_logic_vector(0 to 3);
    ENA   : out std_logic;
    SSRA  : out std_logic;
    WEA   : out std_logic;

    Access_Intr : out std_logic;

    Scheduler_Reset      : out std_logic;
    Scheduler_Reset_Done : in  std_logic;

    Semaphore_Reset      : out std_logic;
    Semaphore_Reset_Done : in  std_logic;

    SpinLock_Reset       : out std_logic;
    SpinLock_Reset_Done  : in  std_logic;

    User_IP_Reset        : out std_logic;
    User_IP_Reset_Done   : in  std_logic;

    Soft_Stop   : out std_logic;

    Current_Thread_ID : out std_logic_vector(0 to 7);

    Next_Thread_ID    : in  std_logic_vector(0 to 7);
    Dequeue_Request   : out std_logic;
    Next_Thread_Valid : in  std_logic;

    Thread_ID_2_Sched : out std_logic_vector(0 to 7);
    Enqueue_Request   : out std_logic;
    Enqueue_Busy      : in  std_logic
    );
end entity user_logic;

----------------------------------------------------------------------------
-- Architecture section
----------------------------------------------------------------------------

architecture IMP of user_logic is

----------------------------------------------------------------------------
-- Signal declarations
----------------------------------------------------------------------------

--  Define the memory map for each register, Address[16 to 21]
--
constant C_CLEAR_THREAD     : std_logic_vector(0 to 5) := "000000";
constant C_JOIN_THREAD      : std_logic_vector(0 to 5) := "000001";
constant C_DETACH_THREAD    : std_logic_vector(0 to 5) := "000010";
constant C_READ_THREAD      : std_logic_vector(0 to 5) := "000011";
constant C_ADD_THREAD       : std_logic_vector(0 to 5) := "000100";

constant C_CREATE_THREAD_J  : std_logic_vector(0 to 5) := "000101";
constant C_CREATE_THREAD_D  : std_logic_vector(0 to 5) := "000110";
constant C_EXIT_THREAD      : std_logic_vector(0 to 5) := "000111";
constant C_NEXT_THREAD      : std_logic_vector(0 to 5) := "001000";
constant C_YIELD_THREAD     : std_logic_vector(0 to 5) := "001001";

constant C_CURRENT_THREAD   : std_logic_vector(0 to 5) := "010000";
constant C_IDLE_THREAD      : std_logic_vector(0 to 5) := "010001";
constant C_QUEUE_LENGTH     : std_logic_vector(0 to 5) := "010010";

constant C_EXCEPTION_ADDR   : std_logic_vector(0 to 5) := "010011";
constant C_EXCEPTION_REG    : std_logic_vector(0 to 5) := "010100";

constant C_SOFT_START       : std_logic_vector(0 to 5) := "010101";
constant C_SOFT_STOP        : std_logic_vector(0 to 5) := "010110";
constant C_SOFT_RESET       : std_logic_vector(0 to 5) := "010111";


constant Z32 : std_logic_vector(0 to 31) := (others => '0');
constant H32 : std_logic_vector(0 to 31) := (others => '1');

constant MAX_QUEUE_SIZE : std_logic_vector(0 to 7) := (others => '1');

constant TOUT_CYCLES : natural := 3;    -- assert timeout suppress
signal   cycle_count : natural := 0;
signal   state_count : natural := 0;


--  Extended Thread Error Codes returned in lower 4 bits
--
constant ERROR_IN_STATUS            : std_logic_vector(0 to 3) := "0001";
constant THREAD_ALREADY_TERMINATED  : std_logic_vector(0 to 3) := "0011";
constant THREAD_ALREADY_QUEUED      : std_logic_vector(0 to 3) := "0101";


--  Exception "cause" returned in Exception register
--
constant WRITE_TO_READ_ONLY  : std_logic_vector(0 to 3) := "0001";
constant UNDEFINED_ADDRESS   : std_logic_vector(0 to 3) := "0010";
constant SOFT_RESET_FAILURE  : std_logic_vector(0 to 3) := "0011";

--  SWTM and SCHEDULER disagree if thread was queued prior to dequeue
constant SCHEDULER_ERROR     : std_logic_vector(0 to 3) := "0100";

-- timeout while waiting for enqueue to begin
constant SCHEDULER_ERROR_5   : std_logic_vector(0 to 3) := "0101";

-- timeout while waiting for enqueue acknowledgement
constant SCHEDULER_ERROR_6   : std_logic_vector(0 to 3) := "0110";

-- timeout while waiting for dequeue to begin
constant SCHEDULER_ERROR_7   : std_logic_vector(0 to 3) := "0111";

-- timeout while waiting for dequeue acknowledgement
constant SCHEDULER_ERROR_8   : std_logic_vector(0 to 3) := "1000";


--  Address,Cause for access exceptions
--
signal Exception_Address  : std_logic_vector(0 to 31) := (others => '0');
signal Exception_Cause    : std_logic_vector(0 to 3)  := (others => '0');
signal access_error       : std_logic := '0';


--  Debug control signals
--
--  Soft reset signals, LSB = SWTM reset; reset IP(s) if '1'
--  Resets done, handshake from IPs if done resetting(1)
--  core_stop   , halt state machines at next appropriate point if '1'
--
signal soft_resets  : std_logic_vector(0 to 4) := (others => '0');
signal resets_done  : std_logic_vector(0 to 4);
signal reset_status : std_logic_vector(0 to 4);
signal core_stop    : std_logic := '0';


--  Declarations for each register

--  Current thread,Idle thread : bits 0..7 = ID, bit 8 = '1' = invalid
signal Current_Thread   : std_logic_vector(0 to 8) := Z32(0 to 7) & '1';
signal Idle_Thread      : std_logic_vector(0 to 8) := Z32(0 to 7) & '1';

signal Que_Length       : std_logic_vector(0 to 7) := (others => '0');


--  internal signals
--
signal next_ID          : std_logic_vector(0 to 8) := (others => '0');
signal addr             : std_logic_vector(0 to 5);
signal addr2            : std_logic;
signal testOK           : std_logic;

---------------------------------------------------------------------------
-- Begin architecture
---------------------------------------------------------------------------

begin -- architecture IMP

CLKA  <= Bus2IP_Clk;            --  BRAM port A clk = system clock
SSRA  <= '0';                   --  BRAM port A sync.reset, not used
DIPA  <= (others => '0');       --  BRAM port A parity inputs, not used

Soft_Stop <= core_stop;

Scheduler_Reset <= soft_resets(3);
resets_done(3)  <= Scheduler_Reset_Done;

Semaphore_Reset <= soft_resets(2);
resets_done(2)  <= Semaphore_Reset_Done;

SpinLock_Reset  <= soft_resets(1);
resets_done(1)  <= SpinLock_Reset_Done;

User_IP_Reset   <= soft_resets(0);
resets_done(0)  <= User_IP_Reset_Done;

Access_Intr <= access_error;

Current_Thread_ID <= Current_Thread(0 to 7);


ADDR_DECODE : process(Bus2IP_Addr) is
--
--  combine address bits to form a 6-bit address
--    to decode for memory mapping,
--  addr2 set to 0 for all valid addresses, else 1
--
begin
  if (Bus2IP_Addr(17 to 21) < 5)  or
     (Bus2IP_Addr(22 to 29) = Z32(0 to 7)) then
      addr2 <= Bus2IP_Addr(16) or Bus2IP_Addr(30) or Bus2IP_Addr(31);
  else
      addr2 <= '1';     -- invalid address
  end if;
  addr <= addr2 & Bus2IP_Addr(17 to 21);
end process ADDR_DECODE;


CYCLE_CONTROL : process(Bus2IP_Clk) is
--
--  create a counter for the number of elapsed cycles
--    in each bus transaction.
--  assert TimeOut suppress when count = TOUT_CYCLES
```

```
--
begin
  IP2Bus_Retry       <= '0';    -- no retry
  IP2Bus_Error       <= '0';    -- no error
  IP2Bus_PostedWrInh <= '1';    -- inhibit posted write
  --
  --  count the number of elapsed clock cycles in transaction
  --
  if Bus2IP_Clk'event and (Bus2IP_Clk = '1') then
     if (Bus2IP_CS = '0') then
        cycle_count <= 0;                 -- hold in reset, or
     elsif cycle_count < C_RESET_TIMEOUT then
        cycle_count <= cycle_count + 1;   -- next cycle, or
     else
        cycle_count <= C_RESET_TIMEOUT;   -- saturate counter
     end if;
  end if;
  --
  --  activate time out suppress if count exceeds TOUT_CYCLES
  --
  if cycle_count > TOUT_CYCLES then
     IP2Bus_ToutSup <= '1';     -- halt time out counter
  else
     IP2Bus_ToutSup <= '0';     -- release
  end if;
end process CYCLE_CONTROL;


MANAGER_ACCESS : process (Bus2IP_Clk) is
--
--  provide access to each of the SW Thread Manager's
--   mechanisms and registers
--
variable current_status : std_logic_vector(0 to 31);
variable new_ID         : std_logic_vector(0 to 7);

  procedure end_transaction is
    begin
    WEA <= '0';
    ENA <= '0';             -- protect BRAM
    IP2Bus_Ack <= '1';      -- done, "ack" the bus
    state_count <= 0;       -- reset present_state counter
    end procedure;

  procedure raise_Exception (cause : in std_logic_vector(0 to 3)) is
    begin
    Exception_Address <= Bus2IP_Addr(0 to 31); -- save address
    Exception_Cause   <= cause;                -- save reason
    access_error <= '1';    -- assert interrupt
    end_transaction;
    end procedure;

begin
  if Bus2IP_Clk'event and (Bus2IP_Clk = '1') then

    --  drive out zeros if not being accessed
    --  or in a write operation
    --
    if(Bus2IP_RdCE = '0') then
        IP2Bus_Data(0 to 31) <= (others => '0');
    end if;

    IP2Bus_Ack   <= '0';    -- pulse(010) to end bus transaction
    access_error <= '0';    -- pulse(010) for access error interrupt

    case addr is

      when C_SOFT_START | C_SOFT_STOP =>
        --
        --  write any data to soft_start to clear all
        --    soft reset signals and the Soft_Stop signal
        --    always returns zeros on read
        --
        --  write any data to Soft_Stop to assert the
        --    Soft_Stop signal
        --    returns signal level in LSB on read
        --
        if (Bus2IP_WrCE = '1') then
          if (addr = C_SOFT_START) then
            soft_resets <= (others => '0');
            resets_done(4) <= '0';  -- clear SWTM's reset done
            core_stop <= '0';       -- clear core_stop
          else
            core_stop <= '1';       -- assert core_stop
          end if;
          end_transaction;
        elsif (Bus2IP_RdCE = '1') then    -- perform read
          if (addr = C_SOFT_START) then
            IP2Bus_Data(0 to 31) <= Z32(0 to 31);
          else
            IP2Bus_Data(0 to 31) <= Z32(0 to 30) & core_stop;
          end if;
          end_transaction;
        end if;

      when C_SOFT_RESET =>
        --
        --  read/write to the soft resets register (1 bit per IP)
        --  write '1' to reset, reads '1' if timeout error occured
        --    before IP reports finished
        --
        --  SW Thread Manager = bit#4  (LSB)
        --  Scheduler         = bit#3
        --  Semaphore         = bit#2
        --  SpinLock          = bit#1
        --  User_IP           = bit#0
        --
        if (Bus2IP_WrCE = '1') then     -- write to soft_resets
          case cycle_count is
            when 0 =>
              soft_resets    <= Bus2IP_Data(27 to 31);
              reset_status   <= (others => '0');
              resets_done(4) <= '0';    -- clear SWTM's reset_done
            when 1 =>
              if (soft_resets(4) = '1') then
                --
                --  perform a soft reset on SWTM
                --
                next_ID         <= Z32(0 to 8);
                Que_Length      <= Z32(0 to 7);
                Current_Thread  <= Z32(0 to 7) & '1';
                WEA             <= '0';
                ENA             <= '0';
                core_stop       <= '0';
                Exception_Address <= (others => '0');
                Exception_Cause   <= (others => '0');
                Dequeue_Request <= '0';
                Enqueue_Request <= '0';
              end if;
            when others =>
              if (soft_resets(4) /= resets_done(4)) then
                --
                --  initialize the thread ID table to all zeros
                --   and the next available stack to 0..255
                --
                ADDRA <= next_ID;
                ENA <= '1';
                WEA <= '1';
                if (next_ID(0) = '0') then
                  --  init available ID stack & thread ID table
                  DIA <= next_ID(1 to 8) & Z32(0 to 23);
                else
                --  clear 2nd half of table (unused)
                DIA <= Z32(0 to 31);
              end if;
              if (next_ID = H32(0 to 8)) then
                resets_done(4) <= '1';    -- done
              end if;
              next_ID <= next_ID + 1;
            else
              --
              --  wait for all IPs to finish initialization or
              --   the maximum time to be exceeded then
              --   ack to finish transaction
              --
              if (resets_done = soft_resets) then     -- done
                end_transaction;
              elsif (cycle_count = C_RESET_TIMEOUT) then
                reset_status <= (resets_done xor soft_resets);
                raise_Exception(SOFT_RESET_FAILURE);  -- timeout
              end if;
            end if;
        end case;
      elsif (Bus2IP_RdCE = '1') then    -- return reset status
        -- returns 1's in bit positions that failed
        IP2Bus_Data(0 to 31) <= Z32(0 to 26) & reset_status;
        end_transaction;
      end if;

when C_CURRENT_THREAD | C_QUEUE_LENGTH =>
  --
  --  read the requested register, error if write
  --
  if (Bus2IP_WrCE = '1') then
    raise_Exception(WRITE_TO_READ_ONLY);
  elsif (Bus2IP_RdCE = '1') then
    if (addr = C_CURRENT_THREAD)  then
      IP2Bus_Data(0 to 31) <= Z32(0 to 22) & Current_Thread;
    else  -- C_QUEUE_LENGTH
      IP2Bus_Data(0 to 31) <= Z32(0 to 23) & Que_Length;
    end if;
    end_transaction;
  end if;

when C_EXCEPTION_ADDR | C_EXCEPTION_REG =>
  --
  --  read the requested register, error if write
  --
  if (Bus2IP_WrCE = '1') then
    raise_Exception(WRITE_TO_READ_ONLY);
  elsif (Bus2IP_RdCE = '1') then
    if (addr = C_EXCEPTION_ADDR) then
      IP2Bus_Data(0 to 31) <= Exception_Address;
    else      --  C_EXCEPTION_REG
      IP2Bus_Data(0 to 31) <= Z32(0 to 27) & Exception_Cause;
    end if;
    end_transaction;
  end if;

when C_IDLE_THREAD =>
  --
  --  read/write to the idle thread register
  --  LSB=1 for uninitialized register
  --
  if (Bus2IP_WrCE = '1') then       -- write new idle thread ID
    Idle_Thread <= Bus2IP_Data(24 to 31) & '0';
    end_transaction;
  elsif (Bus2IP_RdCE = '1') then    -- return idle thread ID
    IP2Bus_Data(0 to 31) <= Z32(0 to 22) & Idle_Thread;
    end_transaction;
  end if;
```

```
when C_READ_THREAD =>
  --
  --  read/write to the addressed row in ID Table
  --  writing is only enabled when core_stop = 1
  --
  ADDRA <= '0' & Bus2IP_Addr(22 to 29); -- thread ID
  if (Bus2IP_WrCE = '1') then
    --
    --  write to table if "stopped" else error
    --
    case cycle_count is
      when 0 =>                   -- initiate BRAM write
        if (core_stop = '1') then
          WEA  <= '1';
          ENA  <= '1';
          DIA  <= Bus2IP_Data(0 to 31);
        else
          raise_Exception(WRITE_TO_READ_ONLY);
        end if;
      when 1 =>               -- write done
        end_transaction;
      when others =>
        WEA  <= '0';
        ENA  <= '0';
    end case;
  elsif (Bus2IP_RdCE = '1') then
    --
    --  read the thread's status
    --
    case cycle_count is
      when 0 =>           -- initiate BRAM read
        WEA  <= '0';
        ENA  <= '1';
      when 1 => null;     -- still reading
      when 2 =>           -- set output data, signal done
        IP2Bus_Data(0 to 31) <= DOA;
        end_transaction;
      when others =>
        WEA  <= '0';
        ENA  <= '0';
    end case;
  end if;

when C_CREATE_THREAD_D | C_CREATE_THREAD_J =>
  --
  --  create a detached or joinable thread
  --
  --  perform bram (read, read,modify,write), ack
  --
  if (Bus2IP_WrCE = '1') then
    raise_Exception(WRITE_TO_READ_ONLY);
  elsif (Bus2IP_RdCE = '1') then
    --
    --  run "create" mechanism, return completion status
    --
    case cycle_count is
      when 0 =>
        if next_ID(0) = '1' then
          --  no IDs available, return with error bit set
          --
          IP2Bus_Data(0 to 31) <= Z32(0 to 30) & '1';
          end_transaction;
        else
          -- read next ID from stack
          --
          WEA   <= '0';
          ENA   <= '1';
          ADDRA <= next_ID;
        end if;
      when 1 => null;       -- still reading bram
      when 2 =>
```

```
        new_ID := DOA(0 to 7);  -- save new ID#
        ADDRA <= '0' & new_ID;  -- point to new thread
      when 3 => null;       -- still reading bram
      when 4 =>
        WEA   <= '1';   -- enable write to bram
        if addr = C_CREATE_THREAD_D then  -- set new thread status
          --  create detached
          DIA <= DOA(0 to 7) & Z32(0 to 7) &
              Z32(0 to 7) & "1010" & Z32(0 to 3);
        else
          --  create joinable
          DIA <= DOA(0 to 7) & Z32(0 to 7) &
              Current_Thread(0 to 7) & "0010" & Z32(0 to 3);
        end if;
      when 5 =>
        --  return new ID with no error,
        IP2Bus_Data(0 to 31) <= Z32(0 to 22) & new_ID & '0';
        --  point to next available ID
        next_ID <= next_ID + 1;
        end_transaction;
      when others =>
        WEA  <= '0';
        ENA  <= '0';
    end case;
  end if;


when C_CLEAR_THREAD | C_JOIN_THREAD | C_DETACH_THREAD =>
  --
  --  clear the encoded thread ID if its PID = current_thread
  --
  --  join on the encoded thread ID if its PID = current_thread
  --    and its status = used,~joined,~detached
  --
  --  detach the encoded thread ID if its PID = current_thread
  --    and its status = used,~exited,~joined,~detached
  --
  --  perform (read,modify,write, read,modify,write), ack
  --            thread status     available ID stack
  --
  if (Bus2IP_WrCE = '1') then
    raise_Exception(WRITE_TO_READ_ONLY);
  elsif (Bus2IP_RdCE = '1') then
    case cycle_count is
      when 0 =>           -- initiate BRAM read
        ADDRA  <= '0' & Bus2IP_Addr(22 to 29);  -- thread ID
        testOK <= '0';    -- set to '1' for ID deallocation
        WEA    <= '0';
        ENA    <= '1';
      when 1 => null;     -- still reading bram
      when 2 =>           -- check status

        if (addr = C_JOIN_THREAD)  and
          (DOA(16 to 23) & '0' = Current_Thread) and  -- PID = current thread
          (DOA(24 to 25) =  "00")  and                -- ~detached,~joined
          (DOA(26 to 27) /= "00")  then               -- not unused
          if DOA(26) = '0' then
            -- thread has already exited, return error code
            IP2Bus_Data(0 to 31) <= Z32(0 to 27) & THREAD_ALREADY_TERMINATED;
            testOK <= '1';                            -- recycle thread ID
            WEA <= '1';                               -- clear old status but
            DIA <= DOA(0 to 7) & Z32(0 to 23);        --  preserve ID stack
          else
            IP2Bus_Data(0 to 31) <= Z32;              -- success, return zero
            DIA <= DOA(0 to 24) & '1' & DOA(26 to 31);  -- set joined bit and
            WEA <= '1';                               -- preserve all other bits
          end if;

        elsif (addr = C_CLEAR_THREAD) and
          (DOA(16 to 23) & '0' = Current_Thread) and  -- PID = current thread
          (DOA(24) = '0') and                         -- not detached
          (DOA(26 to 27) /= "11")  then               -- not (used,~exited,queued)
          IP2Bus_Data(0 to 31) <= Z32;                -- success, return zero
          testOK <= '1';                              -- recycle thread ID
          WEA <= '1';                                 -- clear old status but
          DIA <= DOA(0 to 7) & Z32(0 to 23);          --  preserve ID stack

        elsif (addr = C_DETACH_THREAD)  and
          (DOA(16 to 23) & '0' = Current_Thread) and  -- PID = current thread
          (DOA(24 to 26) =  "001")  then              -- used,~exited,~detached,~joined
          IP2Bus_Data(0 to 31) <= Z32;                -- success, return zero
          WEA <= '1';
          -- set PID=0, set detached bit, preserve all other bits
          DIA <= DOA(0 to 15) & Z32(0 to 7) & '1' & DOA(25 to 31);

        else
          --  error occurred, return thread status w/ LSB=1
          --
          IP2Bus_Data(0 to 31) <= DOA(0 to 27) & ERROR_IN_STATUS;
        end if;
      when 3 =>
        WEA <= '0';   -- end bram write
      when 4 =>
        --
        --  deallocate thread ID on successful clear or
        --   already exited, joinable child, just now joined
        --
        if (testOK = '1')  and
           (next_ID /= Z32(0 to 8)) then
          ADDRA   <= next_ID - 1;
          next_ID <= next_ID - 1;
        else
          end_transaction;
        end if;
      when 5 => null;         -- still reading bram
      when 6 =>
        -- put ID back on stack,   preserve other bits
        DIA <= Bus2IP_Addr(22 to 29) & DOA(8 to 31);
        WEA <= '1';
      when 7 =>
        end_transaction;
      when others =>
        WEA <= '0';
        ENA <= '0';
    end case;
  end if;

when C_NEXT_THREAD =>
  --
  --  return the next thread in the queue, idle thread if empty
  --  set Current Thread = to the thread ID returned
  --
  --  perform (read, modify, write), ack
  --
  if (Bus2IP_WrCE = '1') then
    raise_Exception(WRITE_TO_READ_ONLY);
  elsif (Bus2IP_RdCE = '1') then
    case state_count is
      when 0 =>
        if Que_Length = 0 then
          --
          --  que is empty, return idle thread
          --
          IP2Bus_Data(0 to 31) <= Z32(0 to 22) & Idle_Thread;
          Current_Thread <= Idle_Thread;
          end_transaction;

        elsif Next_Thread_Valid = '1' then
          --
          --  the next thread has been identified,
          --  read from Scheduler and check thread status
          --  as stored by SWTM for consistency
          --
          WEA   <= '0';   -- initiate BRAM read
          ENA   <= '1';
          ADDRA <= '0' & Next_Thread_ID;
          state_count <= state_count + 1;  -- move to next state
        elsif (cycle_count = C_RESET_TIMEOUT) then
          raise_Exception(SCHEDULER_ERROR_7);
        else
          null;   -- waiting for valid status from Scheduler
        end if;
      when 1 =>
          state_count <= state_count + 1;  -- still reading BRAM
      when 2 =>
          if DOA(26 to 27) = "11" then
            --
            --  thread status is used,~exited,queued so
            --  prepate to return ID to system and
            --  update thread status to used,~exited,~queued
            --
            IP2Bus_Data(0 to 31) <= Z32(0 to 22) & Next_Thread_ID & '0';
            Current_Thread  <= Next_Thread_ID & '0';
            Que_Length  <= Que_Length - 1;
            DIA <= DOA(0 to 25) & "10" & DOA(28 to 31);
            WEA <= '1';
            state_count <= state_count + 1;  -- move to next state
          else
            --
            --  SWTM and SCHEDULER disagree if thread was queued
            --  return thread ID, set error bit and raise exception
            --
            IP2Bus_Data(0 to 31) <= Z32(0 to 22) & Next_Thread_ID & '1';
            raise_Exception(SCHEDULER_ERROR);
          end if;
      when 3 =>
          ENA <= '0';
          WEA <= '0';                 -- end bram write
          Dequeue_Request <= '1';     -- tell Scheduler to dequeue
          state_count <= state_count + 1;  -- move to next state
      when 4 =>
          if Next_Thread_Valid = '0' then
            Dequeue_Request <= '0';   -- deassert request
            end_transaction;          -- done
          elsif (cycle_count = C_RESET_TIMEOUT) then
            raise_Exception(SCHEDULER_ERROR_8);
          else
            null;  -- waiting for scheduler acknowledgement
          end if;
      when others =>
          WEA <= '0';
          ENA <= '0';
    end case;
  end if;

when C_ADD_THREAD =>
  --
  --  if the thread is already in the queue, don't re-add, return error
  --
  --  else add the encoded thread ID to the queue if its status is
  --  used, ~exited, ~queued, changing its queued bit to true.
  --
  --  perform (read, modify, write), ack
  --
  if (Bus2IP_WrCE = '1') then
    raise_Exception(WRITE_TO_READ_ONLY);
  elsif (Bus2IP_RdCE = '1') then
    case state_count is
      when 0 =>           -- initiate BRAM read
        WEA   <= '0';
        ADDRA <= '0' & Bus2IP_Addr(22 to 29);  -- encoded thread ID
        ENA   <= '1';
        state_count <= state_count + 1;
      when 1 =>
        state_count <= state_count + 1;  -- still reading BRAM
      when 2 =>
        --
        --  check to see if thread should/can be re-added to queue
        --
        if (DOA(26 to 27) = "11")  then
          --
          --  thread is already in the queue, return error code
          --
          IP2Bus_Data(0 to 31) <= Z32(0 to 27) & THREAD_ALREADY_QUEUED;
          end_transaction;
        elsif (DOA(26) = '0')  or
              (Que_Length = MAX_QUEUE_SIZE) then
          --
          --  thread is unused or exited, or queue is full
          --  operation failed, return error code
          --
          IP2Bus_Data(0 to 31) <= DOA(0 to 27) & ERROR_IN_STATUS;
          end_transaction;
        else
          --
          --  update status to show now queued and
          --  increment the queue length
          --
          IP2Bus_Data(0 to 31) <= Z32;    -- return 0, no error
          Que_Length <= Que_Length + 1;
          DIA <= DOA(0 to 25) & "11" & DOA(28 to 31);
          WEA <= '1';
          state_count <= state_count + 1;
        end if;
      when 3 =>
        WEA <= '0';   -- end BRAM write
        ENA <= '0';
        Thread_ID_2_Sched <= Bus2IP_Addr(22 to 29);    -- thread ID
        if Enqueue_Busy = '0' then
          state_count <= state_count + 1;
        elsif (cycle_count = C_RESET_TIMEOUT) then
            raise_Exception(SCHEDULER_ERROR_5);
        else
          null;   -- waiting for scheduler to be ready
        end if;
      when 4 =>
        Enqueue_Request <= '1';   -- assert request
        state_count <= state_count + 1;
      when 5 =>
        --
        --  wait for Scheduler to acknowledge then
        --  remove request and finish transaction
        --
        if Enqueue_Busy = '1' then
          Enqueue_Request <= '0';
          end_transaction;
        elsif (cycle_count = C_RESET_TIMEOUT) then
            raise_Exception(SCHEDULER_ERROR_6);
        else
          null;   -- waiting for scheduler to acknowledge
        end if;
      when others =>
        WEA <= '0';
        ENA <= '0';
    end case;
  end if;

when C_YIELD_THREAD =>
  --
  --  if the queue is not empty, re-add the current thread to the queue,
  --    then return the next thread in the queue,
  --  else return the current thread
  --
  --  perform (read,modify,write, read,modify,write, read,modify,write),ack
```

```
--           current thread,    last thread added,  next thread
--
if (Bus2IP_WrCE = '1') then
  raise_Exception(WRITE_TO_READ_ONLY);
elsif (Bus2IP_RdCE = '1') then
  case state_count is
    when 0 =>
      if Que_Length = 0 then
        --
        --  queue is empty, return current thread
        --
        IP2Bus_Data(0 to 31) <= Z32(0 to 22) & Current_Thread;
        end_transaction;
      else
        --
        --  read current thread's status
        --
        ADDRA <= '0' & Current_Thread(0 to 7);
        WEA   <= '0';
        ENA   <= '1';
        state_count <= state_count + 1;
      end if;
    when 1 =>
      state_count <= state_count + 1;  -- still reading bram
    when 2 =>
      current_status := DOA(0 to 31);  -- save for later use
      --
      --  check to see if thread's status is  used,~exited,~queued
      --
      if (DOA(26 to 27) = "10")    and
         (Que_Length /= MAX_QUEUE_SIZE) then
        --
        --  update status to show now queued
        --
        DIA <= DOA(0 to 25) & "11" & DOA(28 to 31);
        WEA <= '1';
        Que_Length <= Que_Length + 1;
        state_count <= state_count + 1;
      else
        --  operation failed, return error code
        --
        IP2Bus_Data(0 to 31) <= DOA(0 to 27) & ERROR_IN_STATUS;
        end_transaction;
      end if;
    when 3 =>
      WEA <= '0';   -- end bram write
      Thread_ID_2_Sched <= Current_Thread(0 to 7);
      if Enqueue_Busy = '0' then
        state_count <= state_count + 1;
      elsif (cycle_count = C_RESET_TIMEOUT) then
          raise_Exception(SCHEDULER_ERROR_5);
      else
        null;   -- wait for scheduler to be ready
      end if;
    when 4 =>
      Enqueue_Request <= '1';   -- assert request
      state_count <= state_count + 1;
    when 5 =>
      --
      --  wait for Scheduler to acknowledge
      --  then remove request
      --
      if Enqueue_Busy = '1' then
        Enqueue_Request <= '0';
        state_count <= state_count + 1;
      elsif (cycle_count = C_RESET_TIMEOUT) then
          raise_Exception(SCHEDULER_ERROR_6);
      else
        null;   -- waiting for scheduler to acknowledge
      end if;
    when 6 =>
```

```
        --
        --  wait for Scheduler to finish enqueuing and
        --  identify next thread to run, then
        --  read from Scheduler and check thread status
        --  as stored by SWTM for consistency
        --
        if (Enqueue_Busy = '0') and
           (Next_Thread_Valid = '1') then
          WEA   <= '0';
          ENA   <= '1';
          ADDRA <= '0' & Next_Thread_ID;
          state_count <= state_count + 1;  -- move to next state
        elsif (cycle_count = C_RESET_TIMEOUT) then
            raise_Exception(SCHEDULER_ERROR_7);
        else
          null;
        end if;
      when 7 =>
        state_count <= state_count + 1;    -- still reading BRAM
      when 8 =>
          if DOA(26 to 27) = "11" then
            --
            --  thread status is used,~exited,queued so
            --  prepate to return ID to system and
            --  update thread status to used,~exited,~queued
            --
            IP2Bus_Data(0 to 31) <= Z32(0 to 22) & Next_Thread_ID & '0';
            Current_Thread  <= Next_Thread_ID & '0';
            Que_Length  <= Que_Length - 1;
            DIA <= DOA(0 to 25) & "10" & DOA(28 to 31);
            WEA <= '1';
            state_count <= state_count + 1;  -- move to next state
          else
            --
            --  SWTM and SCHEDULER disagree if thread was queued
            --  return thread ID, set error bit and raise exception
            --
            IP2Bus_Data(0 to 31) <= Z32(0 to 22) & Next_Thread_ID & '1';
            raise_Exception(SCHEDULER_ERROR);
          end if;
      when 9 =>
          ENA <= '0';
          WEA <= '0';                  -- end bram write
          Dequeue_Request <= '1';      -- tell Scheduler to dequeue
          state_count <= state_count + 1;  -- move to next state
      when 10 =>
          if Next_Thread_Valid = '0' then
            Dequeue_Request <= '0';    -- deassert request
            end_transaction;           -- done
          elsif (cycle_count = C_RESET_TIMEOUT) then
            raise_Exception(SCHEDULER_ERROR_8);
          else
            null;  -- wait for scheduler acknowledgement
          end if;
      when others =>
          WEA <= '0';
          ENA <= '0';
    end case;
  end if;

when C_EXIT_THREAD =>
  --
  --  terminate the current thread
  --
  --  perform 3 read,modify,write cycles then ack
  --
  if (Bus2IP_WrCE = '1') then
    raise_Exception(WRITE_TO_READ_ONLY);
  elsif (Bus2IP_RdCE = '1') then
    case cycle_count is
      when 0 =>               -- read current thread's status
  IP2Bus_Data(0 to 31) <= Z32; -- change if failure occurs
  WEA   <= '0';
  ADDRA <= '0' & Current_Thread(0 to 7);
  ENA   <= '1';
when 1 => null;          -- still reading bram
when 2 =>                -- update current thread's status in table
  WEA <= '1';
  current_status := DOA(0 to 31);
  if (current_status(24) = '1') then
    --
    --  thread is detached, clear status,
    --  preserve ID stack and 'next' field
    --
    DIA <= current_status(0 to 15) & Z32(0 to 15);
  else
    --
    --  thread is joined or joinable, set status to used,exited
    --
    DIA <= current_status(0 to 25) & "01" & current_status(28 to 31);
  end if;
when 3 =>
  WEA <= '0';   -- disable write
when 4 =>
  if (current_status(24 to 25) /= "00") then
    --
    -- if not joinable, deallocate ID
    --
    ADDRA   <= next_ID - 1;
    next_ID <= next_ID - 1;
  end if;
when 5 => null;  -- still reading
when 6 =>
  if (current_status(24 to 25) /= "00") then
    WEA <= '1';
    DIA <= Current_Thread(0 to 7) & DOA(8 to 31);
  end if;
when 7 =>
  WEA <= '0';   -- disable write
when 8 =>
  if (current_status(25) = '1') then
    --
    -- if joined, read parent status
    --
    ADDRA <= '0' & current_status(16 to 23);
  else
    --
    -- thread is joinable or detached, finished
    --
    end_transaction;
  end if;
when 9 => null;     -- reading parent status
--
--  check to see if thread should/can be re-added to queue
--
when 10 =>
  if (DOA(26 to 27) = "11")  then
    --
    --  thread is already in the queue, return error
    --
    IP2Bus_Data(0 to 31) <= Z32(0 to 27) & THREAD_ALREADY_QUEUED;
    end_transaction;
  elsif (DOA(26) = '0')  or
        (Que_Length = MAX_QUEUE_SIZE) then
    --
    --  thread is unused or exited, or queue is full
    --  operation failed, return error code
    --
    IP2Bus_Data(0 to 31) <= DOA(0 to 27) & ERROR_IN_STATUS;
    end_transaction;
  else
    --
```

```
                --  update status to show now queued and
                --    increment the queue length
                --
                IP2Bus_Data(0 to 31) <= Z32;    -- return 0, no error
                Que_Length <= Que_Length + 1;
                DIA <= DOA(0 to 25) & "11" & DOA(28 to 31);
                WEA <= '1';
              end if;
            when 11 =>
              WEA <= '0';   -- end write
              ENA <= '0';
              state_count <= 0;
            when others =>
              --
              -- enqueue parent thread via the scheduler
              --
              case state_count is
                when 0 =>
                  Thread_ID_2_Sched <= current_status(16 to 23);
                  if Enqueue_Busy = '0' then
                    state_count <= state_count + 1;
                  elsif (cycle_count = C_RESET_TIMEOUT) then
                    raise_Exception(SCHEDULER_ERROR_5);
                  else
                    null;   -- wait for scheduler to be ready
                  end if;
                when 1 =>
                  Enqueue_Request <= '1';   -- assert request
                  state_count <= state_count + 1;
                when 2 =>
                  --
                  --  wait for Scheduler to acknowledge then
                  --  remove request and finish transaction
                  --
                  if Enqueue_Busy = '1' then
                    Enqueue_Request <= '0';
                    end_transaction;
                  elsif (cycle_count = C_RESET_TIMEOUT) then
                    raise_Exception(SCHEDULER_ERROR_6);
                  else
                    null;   -- waiting for scheduler to acknowledge
                  end if;
                when others => null;
              end case;
          end case;
        end if;

      when others =>
        if ((Bus2IP_WrCE = '1') or (Bus2IP_RdCE = '1')) then
            raise_Exception(UNDEFINED_ADDRESS);
        end if;

    end case;   -- case addr
  end if;       -- rising clock edge

end process MANAGER_ACCESS;


end architecture IMP;
```

Results

The following table shows the number of clock cycles inserted into the bus transaction, total cycles, and the resulting time required for each of the SWTM's operations. These values are based on a typical read operation requiring 3 clock cycles and a system clock frequency of 100 MHz.

	cycles added	total cycles	time (ns)
ADD_THREAD	5	8	80
CLEAR_THREAD	7	10	100
CREATE_THREAD_JOINABLE	5	8	80
CREATE_THREAD_DETACHED	5	8	80
CURRENT_THREAD	0	3	30
DETACH_THREAD	7	10	100
EXIT_THREAD	14	17	170
IDLE_THREAD	0	3	30
JOIN_THREAD	7	10	100
NEXT_THREAD	4	7	70
QUEUE_LENGTH	0	3	30
READ_THREAD	2	5	50
YIELD_THREAD	10	13	130
EXCEPTION_ADDRESS	0	3	30
EXCEPTION_REGISTER	0	3	30
SOFT_START	0	3	30
SOFT_STOP	0	3	30
SOFT_RESET	513	516	5160

Conclusion

This design was combined with a simple module consisting of three programmable timers and a programmable interrupt controller to form a basic system to facilitate testing and further research. The resulting platform required 2528 slices of the 4928 slices available within the "v2p7" device on the development board. Initial testing consisted of manual read and write operations to the SWTM to verify proper operation of each of the SWTM mechanisms. After all mechanisms had been successfully verified, more extensive testing was performed by team members writing typical applications in software. Further testing and development is ongoing and showing very promising results.

All in all, what started as a project to demonstrate what I had already learned ultimately turned in to a wonderful learning and growing experience. The "behavioral" style of specifying a design in VHDL can abstract the designer from the details of the implementation. While this can be a powerful advantage it has also taught me to never abstract oneself too far. Keeping in mind the "nuts and bolts" of how a design is put together is also a key to a truly successful design.

Acknowledgements

The following individuals provided valuable help and insight in the planning, testing, and ongoing research related to this design and the Hardware/Software Co-design project;

Mitch Trope
Razali Jidin
Jorge Ortiz
Wesley Peck
Jason Argon

Ed Komp
Dan Deavors
Dr. David Andrews
Dr. Douglas Niehaus
Dr. Jerry James

Publications

1. Programming Models for Hybrid FPGA-CPU Computational Components: A Missing Link David Andrews, Douglas Niehaus, Razali Jidin, Michael Finley, Wesley Peck, Michael Frisbie, Jorge Ortiz, Ed Komp, and Peter Ashenden; IEEE micro, July/August 2004

Bibliography

The following sources were referenced while performing this design;

1. Operating System Concepts, 6th edition; Silberschatz, Galvin, Gagne; ISBN 0-471-41743-2

2. The Student's Guide to VHDL, 1st edition; Peter J. Ashenden; ISBN 1-55860-520-7

3. Implementing the Thread Programming Model on Hybrid FPGA/CPU Computational Components; David Andrews, Douglas Niehaus, Razali Jidin;

4. Optimize MicoBlaze Processors for Consumer Electronics Products; John Carbone; Xcell Journal, Spring 2004

5. Xilinx website : www.xilinx.com

6. Virtex-II Pro Platform FPGAs : Functional Description (DS083-2.pdf, v3.1.1)

7. Virtex-II Pro Platform FPGAs : Functional Description (DS110-2.pdf, v1.1)

8. PowerPC 405 Processor Block Reference Guide (ppc405block_ref_guide.pdf, v3.1.2)