Hardware/Software Co-design : Software Thread Manager

Michael Finley
EECS 891, Fall 2004
University of Kansas

Committee Members

- Dr. David Andrews (chair)
- Dr. Perry Alexander
- Dr. Jerry James

Thank you ...

The Big Picture

Real Time FPGA project (RT-FPGA, Hybrid Threads) :

Utilizing Hardware/Software Co-design techniques, develop a Real Time Operating System supporting a Multi-threaded Application platform.

Mitch Trope	Ed Komp
Razali Jidin	Dan Deavors
Jorge Ortiz	Dr. David Andrews
Wesley Peck	Dr. Douglas Niehaus
Jason Agron	Dr. Jerry James

The Big Picture

Real Time FPGA project (RT-FPGA, Hybrid Threads) :

Utilizing Hardware/Software Co-design techniques, develop a Real Time Operating System supporting a Multi-threaded Application platform.

Publications

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

Project Goals

- Develop a Software Thread Manager module for use in the RT-FPGA project.
- Create a basic “core” platform for testing this and future functional modules.
- Demonstrate the use and advantage of the Hardware/Software Co-design methodology in system design.

A Traditional Approach

- System requirements are developed and then analyzed to determine the “level” of technology required to fulfill these needs.
- Hardware and software teams then independently develop their designs combining late in the development cycle for first prototype testing.
- Tends to create a generalized hardware platform and specialized software.

Hardware/Software Co-design

- Seeks to move “specialized” functionality from software into hardware.
- Takes advantage of hardware’s ability to perform multiple tasks in parallel versus the sequential, nature of software execution.
- Tends to create a more balanced distribution of the application’s specifics across the hardware and software, reducing software complexity.

Software Thread Manager (SWTM)

- Provide the services and data structures needed to track the present status of each of the system's software threads.
- Coordinate access to these services to ensure proper functionality.
- Implement a Ready to Run Queue and simple FIFO based scheduling mechanism.
- Provide interface to separate Scheduler module for implementing additional scheduling algorithms.

SWTM Challenges

- Defining a “full” set of services to be implemented and their semantics not yet knowing the application or other FPGA based modules it might interact with.

- Ensuring proper operation of each service, given the environment in which they are utilized.
  - a service might modify multiple data structures
  - can be interrupted at any time

SWTM Services

- Thread services
  - create_thread_detached
  - create_thread_ joinable
  - exit_thread
  - clear_thread
  - join_thread
  - detach_thread
  - read_thread (R/W)
- Queue management
  - add_thread
  - next_thread
  - current_thread
  - yield_thread
  - que_length
  - idle_thread (R/W)

SWTM Services

- System debug (R/W)
  - soft_start
  - soft_stop
  - soft_reset
    - 27 - User
    - 28 - SpinLock
    - 29 - Semaphore
    - 30 - Scheduler
    - 31 - SWTM
- SWTM debug
  - exception_address
  - exception_cause
    - write to read only
    - undefined address
    - soft reset failure

SWTM State : Thread Status

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
-------------|-------------|-------------|-----------------------------------
```

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 is not joined
J = 1 this thread is joined

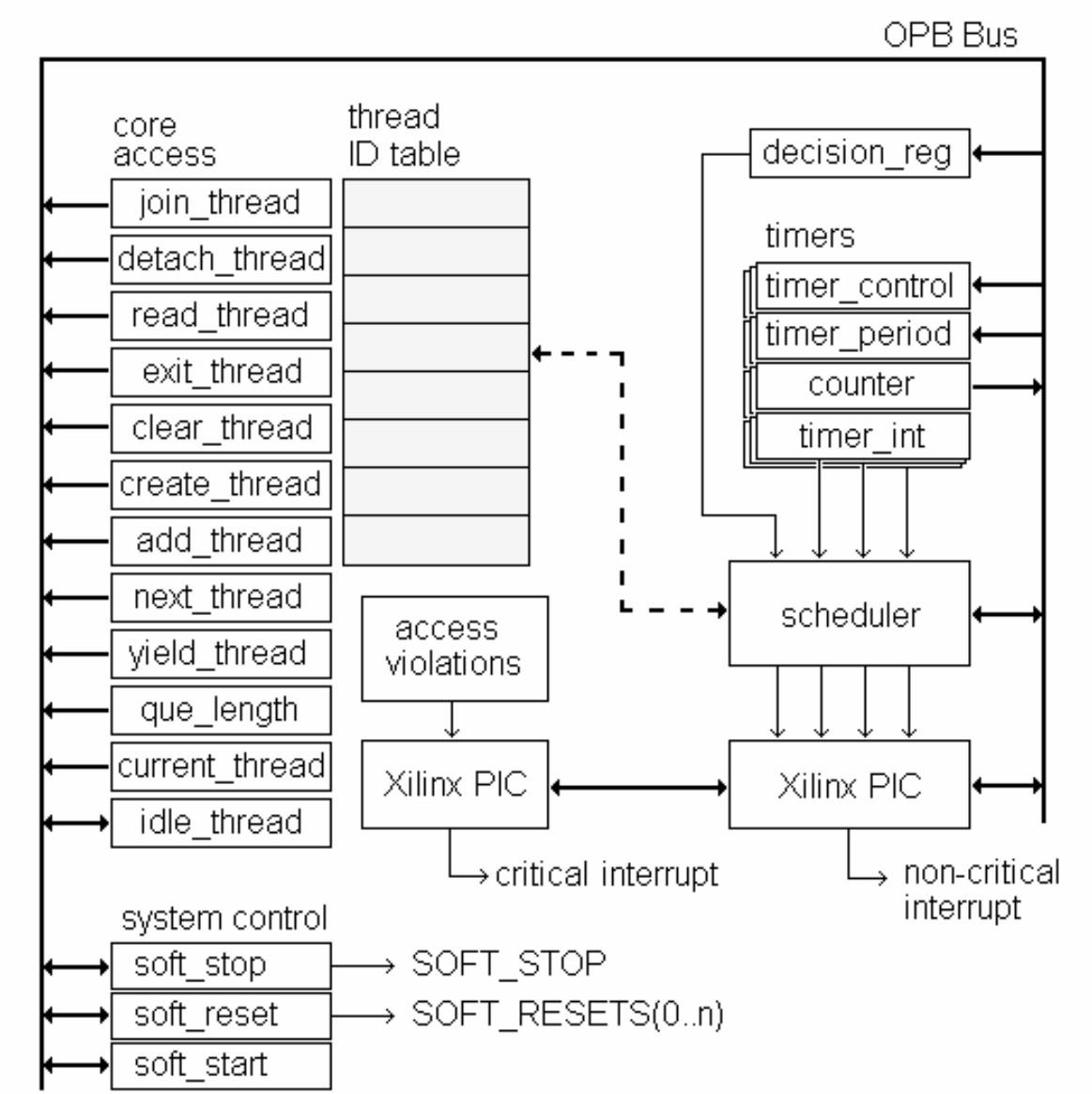
D = 0 this thread is not detached
D = 1 this thread is detached

PID = this thread's Parent ID
Next = next thread in queue

E3	E2	E1	
0	0	0	error in status or no IDs
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

Basic Thread Support



Accessing SWTM Services

- Access begins by telling SWTM which service to perform and supplying any additional parameters needed by the particular service. *write(s)*
- Caller must wait for the service to finish. *read(s)*
- Caller reads result. *read*

Sequence is not atomic and additional means are required to ensure sequence is not interrupted until after caller receives the result.

Atomic Hardware Function

- Takes advantage of the atomic nature of a simple “read” instruction in assembly.
- The initial *write(s)* required are performed by passing the values on the address bus to the SWTM.
  - Base address specifies which service to perform
  - Lower order bits pass the parameter, if needed
- The varying length of time to process is managed by inserting wait states to extend the instruction cycle.
- The result is then returned as the value for the *read*.

SWTM Service Decoding

```
0                 15 16          21 22  23  24  25  26  27  28  29  30  31
--------------------|--------------|-----------------------------------------
SWTM Base Address   |   Service    | T   T   T   T   T   T   T   T   0   0
--------------------|--------------|-----------------------------------------
```

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

    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 SERVICE_1 =>   -- code to perform SERVICE_1
      when SERVICE_2 =>   -- code to perform SERVICE_2
        .
        .
      when SERVICE_n =>   -- code to perform SERVICE_n
      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;
```

```
when C_READ_THREAD =>
  ADDRA <= '0' & Bus2IP_Addr(22 to 29); -- thread ID
  if (Bus2IP_WrCE = '1') then
    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
    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;
```

SWTM Service Summary

	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

Future Work

- Experiment with coding “style” to produce smaller foot print for implementation.
- Adapt existing services for optimal interaction with future system modules as they are developed.
- Adapt design for use in other FPGA architectures.

Additional information : http://people.eecs.ku.edu/~mfinley/