Embedded Microprocessors

Module 12
Topics

• Microcontrollers versus FPGAS
  – What is the best for your application?

• Debugging embedded designs
  – Chipscope Pro and Agilent FPGA Dynamic Probe
  – Use VHDL models of system components (e.g. SDRAM)

• Xilinx Virtex-4, Virtex-5
  – DSPslices, Ethernet MAC, integrated processors

• Adding Microcontrollers to FPGAs
  – Picoblaze, Microblaze, Power PC

• Adding DSP functions to FPGAs
  – Xilinx System Generator for Simulink

• Lessons learned and pitfalls to be avoided
Microcontrollers versus FPGAs

- **Microcontrollers**
  - Execute compiled programs written in ‘C’ or Assembler
    - Program may be on the micro or in external flash memory
    - Micro fetches instructions, decodes, manipulates data or I/O
      - Sequential execution
    - Requires multiple clock cycles for most operations
  - Can be easily optimized for battery powered operations
    - Only wake up in response to external interrupts
  - Limited I/O of standard logic

- **FPGAs**
  - ‘Sea of gates’ can be configured for *any* required logic
  - Very good for high-speed parallel data processing (e.g. networks)
  - Lots of I/O – versatile logic interface (LVDS etc)
  - Can be easily partitioned into different functional blocks
  - Not appropriate for simple control applications (not vending machine!)
Digital Controller Board – Virtex II FPGA

SDRAM

Config PROM

FPGA

LVDS Interface

Ethernet Transceiver
FPGA Functional Blocks
DDR SDRAM Controller – 128 byte writes
Start of Write Cycle
Start of ROW read cycle (2048 bytes)
Part of SDRAM Initialization Sequence
## Virtex 4

<table>
<thead>
<tr>
<th>Device</th>
<th>System Gates</th>
<th>CLBs (4 slices)</th>
<th>CLB flip-flops</th>
<th>Block Ram (bits)</th>
<th>User IO</th>
<th>Price (250K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3S100E</td>
<td>100K</td>
<td>240</td>
<td>1,920</td>
<td>72K</td>
<td>108</td>
<td>&lt;$2</td>
</tr>
<tr>
<td>XC3S500E</td>
<td>500K</td>
<td>1,164</td>
<td>9,312</td>
<td>360K</td>
<td>232</td>
<td>$30 1-off</td>
</tr>
<tr>
<td>XC3S1600E</td>
<td>1.6M</td>
<td>3,688</td>
<td>29,504</td>
<td>648K</td>
<td>376</td>
<td>&lt;$9</td>
</tr>
<tr>
<td>XC4VLX100</td>
<td>5M</td>
<td>12,288</td>
<td>98,304</td>
<td>4,320K</td>
<td>960</td>
<td>$1,000 (25 off)</td>
</tr>
</tbody>
</table>

Jim Duckworth, WPI
PPL Transceiver – Based on Virtex 4
Transceiver Box
Debugging Embedded Designs

- VHDL models of standard devices
  - Micron models of all SDRAM devices – simulates behavior
- Xilinx ChipScope Pro
  - Poor man’s logic analyzer!
  - Connected and analyzed through JTAG port
  - See on-line introduction (with CoreGen)
- Agilent FPGA Dynamic Probe
  - Interfaces through special core inside FPGA
Integration in System Design

Programmable Systems usher in a new era of system design integration possibilities

Jim Duckworth, WPI
Embedded Processors

- **Soft core processors** – use on any FPGA
  - Picoblaze 8-bit microcontroller
    - Programmed in assembler
    - Very useful for replacing complex state machines
  - Microblaze 32-bit microcontroller
    - Use Xilinx EDK – base system builder
    - Program in ‘C’

- **Hard Processor cores**
  - IBM Power PC 405 32/64-bit Processor
    - 16 KB data cache and 16 KB instruction cache
    - On Virtex II Pro and Virtex-4 devices

- **Can add and mix multiple processors!**
  - Need to be familiar with micros (as well as logic design)
Xilinx - Embedded Processing

New Webcast:
Quick start your embedded software development with Linux and Xilinx for Virtex-4 FX FPGAs

Technology Solutions > Product & Services > Embedded Processing

Xilinx delivers an innovative and flexible range of processing solutions for your unique embedded applications:
- High performance PowerPC 440 and PowerPC 405 32-bit hard processors
- Flexible 32-bit MicroBlaze soft processor
- Award winning Platform Studio and the Embedded Development Kit (EDK)
- Broad range of Intellectual Property (IP) and processing peripherals
- Support embedded development with reference examples
- Robust third party ecosystem support

Our processing capabilities provide performance and customization across a wide range of end markets including: aerospace and defense, wired and wireless communications, automotive, industrial control, and measurement, and consumer. Xilinx embedded processing solutions are supported on numerous Virtex™ and Spartan™ FPGAs.

PowerPC Hard Processor
The IBM PowerPC 440 and 405 cores are hard IP JPEG encoder IP core solutions that can be integrated directly into Xilinx Virtex FPGA fabric to implement high performance embedded applications. The combination of dual core hard PowerPC core systems integrated with co-processing capability enables a wide range of performance optimization options.

The PowerPC 440 is instantiated in the Virtex-5 FX FPGA family with a dedicated processor block (APB) container and high-bandwidth memory links. The PowerPC 440 processor core is a 32-bit RISC processor with a 32-bit instruction set optimized for embedded applications in Virtex FPGAs. Its architecture allows you to control cache sizes, interfaces, and execution units. It also customizable, you can tailor features for size, cost, price and performance goals.

The core is licensed as part of the Embedded Development Kit (EDK).

MicroBlaze Soft Processor
The MicroBlaze processor is a flexible 32-bit Harvard RISC architecture with a rich instruction set optimized for embedded applications in Virtex FPGAs. Its architecture allows you to control cache sizes, interfaces, and execution units. It also customizable, you can tailor features for size, cost, price and performance goals.

Embedded Design Tools
Intelligent, platform-aware tools simplify design and accelerate the embedded development process. Embedded design tools help you design, verify and debug your design, as well as share designs and manage project data.

The Xilinx Embedded Development Kit (EDK)

Jim Duckworth, WPI 18 Embedded Microprocessors
Picoblaze 8-bit microcontroller
Picoblaze 8-bit microcontroller
Picoblaze overview

Picoblaze 8-bit Microcontroller Reference Design for FPGAs and CPLDs

Jim Duckworth, WPI
Picoblaze User Guide

Chapter 1

Introduction

The Picoblaze™ microcontroller is a compact, capable, and cost-effective 8-bit embedded RISC microcontroller, optimized for the Spartan-3, Virtex-II, and Virtex-II Pro FPGAs. The Picoblaze microcontroller provides customizable microcontroller-based control and simple data processing.

The Picoblaze microcontroller is optimized for efficiency and low deployment cost. It supports the Picoblaze C compiler, as well as PIC18 C compilers, and is used in a variety of applications, including industrial control, communications, and data acquisition systems.

The Picoblaze microcontroller is used in a wide range of applications, from simple data processing to complex control systems. It is designed to be easy to use and easy to integrate into existing systems.

Picoblaze Microcontroller Features

As shown in the block diagram in Figure 1, the Picoblaze microcontroller supports the following features:

- Instruction set that includes general purpose data registers
- 16-bit width for general purpose data registers
- 16 instructions of programmable on-chip program store, automatically loaded during FPGA configuration

Picoblaze 8-bit Embedded Microcontroller

Jim Duckworth, WPI

Embedded Microprocessors
Picoblaze Block diagram

Figure 1-1: Picoblaze Embedded Microcontroller Block Diagram
Micro versus Logic

<table>
<thead>
<tr>
<th></th>
<th>PicoBlaze Microcontroller</th>
<th>FPGA Logic</th>
</tr>
</thead>
</table>
| **Strengths**        | • Easy to program, excellent for control and state machine applications  
                        • Resource requirements remain constant with increasing complexity  
                        • Re-uses logic resources, excellent for lower-performance functions  | • Significantly higher performance  
                        • Excellent at parallel operations  
                        • Sequential vs. parallel implementation trade-offs optimize performance or cost  
                        • Fast response to multiple, simultaneous inputs |
| **Weaknesses**       | • Executes sequentially  
                        • Performance degrades with increasing complexity  
                        • Program memory requirements increase with increasing complexity  
                        • Slower response to simultaneous inputs  | • Control and state machine applications more difficult to program  
                        • Logic resources grow with increasing complexity |
Picoblaze connected to ROM

Figure 7-1: Standard Implementation using a Single 1Kx18 Block RAM as the Instruction Store
The connection of the program ROM is a straightforward. The KCPSM3 processor and the 'uclock' program are instantiated and signals used to make direct connections. In this design the reset port is not required so it is tied to ground.

The complete reference design is connected to a single clock source 'clk'. We will see this signal used in the more generic VHDL code, but here it is again a direct connection to the instantiated components.
Adding Inputs (UART example)

The Hardware - Inputs

The UART macros provide the only inputs to KCPSM3. There is the data received from the ‘rx’ serial input and captured by the UART_RX macro, but there is also the status signals associated with the FIFO buffers within the UART receiver and transmitter macros which KCPSM3 will need to monitor. This leads to a simple multiplexer and address decoding logic.

Notes

Minimum of address decoding to support the two ports.

Pipeline registers are included to ensure that performance is maintained. The port_id is valid for 2 clock cycles during an INPUT operation.
UART Transmit (assembler example)

Software - UART Transmit

At the lowest level of the code, the software must communicate with the hardware. With CONSTANT directives defining the actual port addresses to be used, and in some cases, the meanings of specific bits, this code can be much more pleasant to write. Obviously the code which interacts with hardware often needs to consider how the external logic behaves in order to operate efficiently as well as correctly.

The 'uclock' program includes a simple sub routine to send a byte of data (or character) to the UART transmitter. The named register 'UART_data' is used to pass data to the sub routine.

```
LOAD UART_data, character
CALL send_to_UART
```

All this routine really needs to do is to write the data contained in the 'UART_data' register to the appropriate port which has been named 'UART_write_ports'. This is achieved with just one OUTPUT instruction. However, the 'uart_tx' macro contains a FIFO buffer, and it is therefore wise to determine if there is space in the buffer to accept the new character before attempting to write it.

```
send_to_UART: INPUT s0, UART_status_port
              TEST s0, tx_full
              JUMP Z, UART_write
              CALL update_time
              JUMP send_to_UART
UART_write: OUTPUT UART_data, UART_write_port
RETURN
```

What if the FIFO buffer really is full?

The program will need to wait until there is space, which could be as long as it takes the UART to transmit one whole character (260μs at 38400 BAUD). This could be an undesirable situation if it prevents other processing from being performed.

'uclock' demonstrates 2 solutions to this issue:

1) Interrupts are used to ensure that the 1μs reference clock pulses are never missed.

2) The subroutine includes a call to the 'update_time' subroutine each time it encounters the full condition. This routine executes in under 4 μs (55MHz clock) which is insignificant to the UART operation but ensures that the real time is accurate and the alarm will still be activated. Your programs could include a call to a similar subroutine.

XILINX

Jim Duckworth, WPI

Embedded Microprocessors
Simple Picoblaze Project

```vhdl
entity ece574_pico is
  Port (  
    sysclk : in std_logic;
    sysreset : in std_logic;
    txd : in std_logic;
    rxd : in std_logic;
    lcsd : out std_logic_vector(7 downto 0);
    dip_sw : in std_logic_vector(7 downto 0)  
  );
end ece574_pico;

architecture Behavioral of ece574_pico is
  -- ***************** components used in toplevel  ***
  -- ************** components used in toplevel  ***

component pico_processor is
  Port (  
    reset : in std_logic;
    clk : in std_logic;
    txd : out std_logic;
  );
end component;
```

Jim Duckworth, WPI
Picoblaze resources – very low!

### Design Summary

**Design Overview for ece574_pico**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Name:</td>
<td>c:\ece574\picoblaze_1\picoblaze_1</td>
</tr>
<tr>
<td>Target Device:</td>
<td>xc3s200</td>
</tr>
<tr>
<td>Report Generated:</td>
<td>Tuesday 11/01/05 at 12:47</td>
</tr>
<tr>
<td>Printable Summary (View as HTML)</td>
<td><a href="#">ece574_pico_summary.html</a></td>
</tr>
</tbody>
</table>

#### Device Utilization Summary

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
<th>Note(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice Flip Flops:</td>
<td>146</td>
<td>3,840</td>
<td>3%</td>
<td></td>
</tr>
<tr>
<td>Number of 4 input LUTs:</td>
<td>165</td>
<td>3,840</td>
<td>4%</td>
<td></td>
</tr>
</tbody>
</table>

**Logic Distribution:**

| Number of occupied Slices:         | 162  | 1,920     | 8%          |         |
| Number of Slices containing only related logic: | 162 | 162 | 100% |         |
| Number of Slices containing unrelated logic: | 0 | 162 | 0% |         |
| **Total Number 4 input LUTs:**    | 285  | 3,840     | 7%          |         |

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number used as logic:</td>
<td>165</td>
</tr>
<tr>
<td>Number used as a route-thru:</td>
<td>16</td>
</tr>
<tr>
<td>Number used for Dual Port RAMs:</td>
<td>16</td>
</tr>
<tr>
<td>Number used for 32x1 RAMs:</td>
<td>52</td>
</tr>
<tr>
<td>Number used as Shift registers:</td>
<td>36</td>
</tr>
<tr>
<td>Number of bonded I/Os:</td>
<td>20</td>
</tr>
<tr>
<td>Number of Block RAMs:</td>
<td>1</td>
</tr>
<tr>
<td>Number of GCLKs:</td>
<td>1</td>
</tr>
</tbody>
</table>

**Performance Summary**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Unrouted Signals:</td>
<td>All signals are completely routed.</td>
</tr>
<tr>
<td>Number of Failing Constraints:</td>
<td>0</td>
</tr>
</tbody>
</table>

Ready
KCPSM3 with ROM and UARTS

```vhdl
-- declaration of KCPSM

component kcpsm
    Port ( 
        address : out std_logic_vector(9 downto 0); 
        instruction : in std_logic_vector(17 downto 0); 
        port_id : out std_logic_vector(7 downto 0); 
        writeStroke : out std_logic; 
        readStroke : out std_logic; 
        outPort : out std_logic_vector(7 downto 0); 
        inPort : in std_logic_vector(7 downto 0); 
        interrupt : in std_logic; 
        interrupt_ack : out std_logic; 
        reset : in std_logic; 
        clk : in std_logic); 
end component;

-- declaration of program ROM

component eces574_rom
    Port ( 
        address : in std_logic_vector(9 downto 0); 
        instruction : out std_logic_vector(17 downto 0); 
        clk : in std_logic); 
end component;

-- declaration of UART transmitter with integral 16 byte FIFO buffer

component uart_tx
    Port ( 
        data_in : in std_logic_vector(7 downto 0); 
        write_buffer : in std_logic; 
        reset_buffer : in std_logic; 
        en16_x_baud : in std_logic; 
        serial_out : out std_logic; 
        buffer_full : out std_logic; 
        buffer_empty : out std_logic; 
        clk : in std_logic); 
end component;

-- declaration of UART Receiver with integral 16 byte FIFO buffer

component uart_rx
    Port ( 
        serial_in : in std_logic; 
        ... 
end component;
```

Jim Duckworth, WPI 31 Embedded Microprocessors
Creating the ROM – using pBLAZE IDE

Jim Duckworth, WPI
Simulating the picoblaze code
VHDL template to create ROM

-- Ken Chapman (Xilinx Ltd) October 2002
-- This is the VHDL template file for the KCPSM assembler.
-- Adapted for pBlazIDE by Henk van Kampen, www.mediatronix.com, March 2003
-- It is used to configure a Virtex(E, II) and Spartan-II(E, 3) block RAM to act as
-- a single port program ROM.
-- This VHDL file is not valid as input directly into a synthesis or simulation tool.
-- The assembler will read this template and insert the data required to complete the
-- definition of program ROM and write it out to a new '.vhd' file as specified by the
-- '.psm' file being assembled.
-- The assembler identifies all text enclosed by {} characters, and replaces these
-- character strings. All templates should include these {} character strings for
-- the assembler to work correctly.
-- The next line is used to determine where the template actually starts and must exist.

{begin template}

library IEEE ;
use IEEE.STD_LOGIC_1164.all ;
use IEEE.STD_LOGIC_ARITH.all ;
use IEEE.STD_LOGIC_UNSIGNED.all ;
library unisim ;
use unisim.vcomponents.all ;

entity {name} is

    port ( 
        clk : in std_logic ;
        address : in std_logic_vector( 9 downto 0 ) ;
        instruction : out std_logic_vector( 17 downto 0 ) 
    ) ;

{end template}
Simulating the whole design
Microblaze 32-bit Soft Processor

The Microblaze™ 32-bit soft processor core is the industry's fastest soft
processing solution. Running at 150 MHz, the Microblaze processor delivers up
to 120 DMIPS in the Virtex® II Pro for building complex systems for the networking,
telecommunication, data communication, embedded and consumer markets.

The Microblaze processor features a RISC architecture with Harvard-
style, separate 32-bit instruction and data buses running at full speed to execute
programs and access data from both on-chip and external memory. A standard
set of peripherals are also CoreConnect™ enabled to offer Microblaze
designers compatibility and reuse.

Download the new Microblaze Microcontroller Reference Design More...
Microblaze Overview

Developing high-performance processor-based systems can be very challenging in today's competitive environment. You need a processor that's easy to use, very cost-effective, optimized for cost-sensitive designs, and able to support you well into the future.

The MicroBlaze® 32-bit RISC soft processor core delivers exceptional performance for creating powerful systems – complete with peripherals, memory, high-speed I/O, and logic – all on a single, programmable logic device. This gives you the capability to tailor the MicroBlaze processor core to meet the needs of a particular application. Plus, our easy-to-use development tools make it simple for you to build your design and update it at any time – even after it's in your customer's hands.

MicroBlaze – The Low-Cost and Flexible Processing Solution

The MicroBlaze soft processor core is fast, cost-effective, cost-performance, and completely hardware compatible with Xilinx® PowerPC® hard cores and products. This enables you to create highly optimized single-chip multi-processor systems that work together seamlessly to give you the best system performance with the lowest possible cost on a single FPGA.

Key features include:
- Design Flexibility – The MicroBlaze 32-bit RISC soft processor architecture is a derivative of the PowerPC®-based processors from Xilinx and the hard-embodied PowerPC core in Virtex® II Pro and Virtex-4 FPGA families of FPGAs, to provide a seamless migration path for software from hard-embodied processors.
- Security Acceleration – The MicroBlaze family contains a microcontroller interface that allows secure embedded code to be stored on a secure memory and protects against data piracy.
- Software Development Tools – The MicroBlaze includes embedded software development tools and support for both bare-metal and Linux environments.
- Embedded System Tools – The MicroBlaze includes embedded software development tools and support for both bare-metal and Linux environments.
- Comprehensive Processor Peripheral IP – The MicroBlaze includes processors compatible with Xilinx® PowerPC® hard cores and Virtex® II and Virtex-4 FPGA families of FPGAs.

Embedded System Tools
As part of the Xilinx, Inc. family, the MicroBlaze is easily configured to meet your needs by using the Xilinx PowerPC® hard cores and Virtex-II Pro and Virtex-4 FPGA families of FPGAs to provide the necessary hardware for your applications.

Key features include:
- Comprehensive PowerPC® hard core and Virtex-II Pro and Virtex-4 FPGA families of FPGAs
- Embedded Software Development Tools – The MicroBlaze includes embedded software development tools and support for both bare-metal and Linux environments.
- Comprehensive Processor Peripheral IP – The MicroBlaze includes processors compatible with Xilinx® PowerPC® hard cores and Virtex-II Pro and Virtex-4 FPGA families of FPGAs.

Embedded Microprocessors

Jim Duckworth, WPI
MicroBlaze-based Embedded Design

- MicroBlaze™ 32-Bit RISC Core
- I-Cache BRAM
- D-Cache BRAM
- On-Chip Peripheral Bus
- Custom Functions
- UART
- 10/100 E-Net
- On-Chip Peripheral
- Off-Chip Memory
- FLASH/SRAM

Flexible Soft IP

- Configurable Sizes
- Possible in Virtex-II Pro
- Dedicated Hard IP
- PowerPC 405 Core
- Processor Local Bus
Embedded Development
Tool Flow Overview

Standard Embedded SW Development Flow

- C Code
- Compiler/Linker
- (Simulator)
- Object Code

CPU code in off-chip memory

CPU code in on-chip memory

Download to Board & FPGA

Debugger

Standard FPGA HW Development Flow

- VHDL/Verilog
- Synthesizer
- Simulator
- Place & Route

Download to FPGA

Data2MEM

Bitstream
EDK (Xilinx Platform Studio)
Base System Builder

Select a target development board:

- Board Vendor: Xilinx
- Board Name: Spartan-3 Starter Board
- Board Revision: E

Vendor's Website  Contact Info  Download Third Party Board Definition Files

I would like to create a system for a custom board

Board Description:
Spartan-3 Starter Kit Board utilizes Xilinx Spartan-3 XC3S200-4FT256 device. The board includes 1 FT232 serial port, 2 256x16 fast SRAM 8 DIP switches, 4 push buttons, four digital 7 segment LEDs, 8 LEDs, 1 VGA port, 1 PS/2 port. Push button 1 is used as system reset. 2 SRAMs are combined to form a 32 bit data bus.

Select the processor you would like to use in this design:

- MicroBase
- PowerPC

Processor Description:
The MicroBase(TM) 32 bit soft processor is a RISC-based engine with a 32 register by 32 bit LUT RAM based Register File, with separate instructions for data and memory access. It supports both on-chip Block RAM and/or external memory. All peripherals are implemented on the FPGA fabric and operate off the on-chip peripheral bus (JPIB).
Configuring Processor and I/O

[Images of software interfaces for configuring processor and I/O]

Jim Duckworth, WPI

Embedded Microprocessors
Microblaze Hardware Block Diagram
Microblaze Software – C Program

```c
#include <parameters.h>
#include <mpio.h>

/*
   Routine to write a pattern set to a GPIO
   which is configured as an output
   PARAMETER C_ALL_INPUTS = 0

   void WriteToGPIOOutput(uint32 BaseAddress, int gpio_width) {
     int i, j, k, n;
     int numTimes = 50;

     NGpio_setDataDirection(BaseAddress, 1, 0x00000000); /* Set all outputs */
     while (numTimes > 0)
     {
       for (i = 0; i < gpio_width-1; i++)
         NGpio_setDataOut(BaseAddress, i);
       numTimes --;
     }
   }
```

Jim Duckworth, WPI
PowerPC-based Embedded Design

- Dedicated Hard IP
  - PowerPC 405 Core
  - DSOCM BRAM
  - ISOCM BRAM

- Flexible Soft IP
  - RocketIO
  - IBM CoreConnect™ on-chip bus standard PLB, OPB, and DCR

- On-Chip Peripheral Bus
  - PLB
  - OPB
  - Arbiter

- Off-Chip Memory
  - ZBT SSRAM
  - DDR SDRAM
  - SDRAM

- Peripherals
  - UART
  - GPIO
  - Hi-Speed Peripheral
  - e.g. Memory Controller
  - GB E-Net
  - On-Chip Peripheral

Full system customization to meet performance, functionality, and cost goals.
Lessons Learned and Pitfalls to be Avoided

- Do we start with VHDL for synthesis or modeling?
- How do we understand timing behavior?
- VHDL is not a programming language
- Where have my signals or ports gone?
- How do I debug my design?
- Use std_logic types for all ports
- Make a test bench as soon as possible
- How do I debug my embedded microprocessor?
- How do I keep track of all the acronyms?