EN2911X: Reconfigurable Computing
Topic 04: Application Acceleration Using Reconfigurable Computing

Professor Sherief Reda
http://scale.engin.brown.edu
Electrical Sciences and Computer Engineering
School of Engineering
Brown University
Fall 2014
What are the pros and cons of soft vs hard processors?
Options to accelerate applications

1. Increase frequency
2. Increase core throughput & number of cores
3. Convert SW to HW:
   a. Custom instruction
   b. Using I/O peripherals
   c. Qsys component
   d. PCI-based components
1. Increasing frequency of soft processors
2. Increase core throughput & number of cores

Control access to shared resources using the mutex
3. Acceleration Options in Nios II SOPC

a. custom instruction

b. I/O-based accelerator

<table>
<thead>
<tr>
<th>I/O</th>
<th>Accelerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon bus</td>
<td></td>
</tr>
<tr>
<td>Nios II processor</td>
<td>Memory</td>
</tr>
</tbody>
</table>

c.d/ bus-based accelerator

<table>
<thead>
<tr>
<th>I/O</th>
<th>Accelerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon bus</td>
<td></td>
</tr>
<tr>
<td>Nios II processor</td>
<td>Memory</td>
</tr>
</tbody>
</table>
3.a Nios II custom instructions

- Extending processor ALU impacts clock frequency
- Processor and accelerator can’t work independently
- Custom instruction can be fixed or variable latency
Nios II custom instruction types

- **Combinational**: Single clock cycle custom logic blocks.
- **Multi-cycle**: Multi-clock cycle custom logic blocks of fixed or variable durations
- **Extended**: Custom logic blocks that are capable of performing multiple operations
- **Internal register file**: Custom logic blocks that access internal register files for input or output or both.
Single-cycle and multi-cycle accelerators

**Combinational**
Completes in one cycle

**Multicycle**
Two or more cycles to complete

Either fixed or variable number of cycles
Extended custom instructions

- Extended custom instruction allows a single custom logic block to implement several different operations. Extended custom instructions use an index to specify which operation the logic block performs. The index can be as many as eight bits wide, allowing a single custom logic block to implement as many as 256 different operations.
- Extended custom instructions can be combinational or multi-cycle custom instructions.
Internal Register File Custom Instructions

- The Nios II processor allows custom instruction logic to access its own internal.
- In addition, a custom instruction can write its results to the local register file rather than to the Nios II processor’s register file.
- Custom instructions containing internal register files use readra, readrb, and writerc signals to determine if the custom instruction should use the internal register file or the dataa, datab, and result signals.
Nios II custom instructions allow you to add an interface to communicate with logic outside of the processor’s datapath. At system generation, conduits propagate out to the top level of the Qsys system, where external logic can access the signals.

By enabling custom instruction logic to access memory external to the processor, external interface custom instructions extend the capabilities of the custom instruction logic.
Custom instruction software interface

- A C macro (defined in system.h function) is defined by the QSYS tool.
- During compilation, GCC converts the macro calls to the custom instructions

```c
#include "system.h"

int main (void)
{
    int a = 0x12345678;
    int a_swap = 0;

    a_swap = ALT_CI_BITSWAP(a);
    return 0;
}
```

- Assembly instruction format

![Assembly instruction format](image)
3. b Accelerator components using I/O

- Nios II uses a memory-mapped scheme to access I/O peripherals, with an interface of registers
- Enables the processor and accelerator to operate in parallel
- Good for cases with a few ports and simple timing requirement
3. b I/O-based accelerators
3.b Code example

C code on Nios

```c
printf("Perform division a / b = q remainder r\n");
printf("Enter a: ");
scanf("%d", &a);
printf("Enter b: ");
scanf("%d", &b);
/* wait until the division accelerator is ready */
while (1) {
    ready = pio_read(READY_BASE)& 0x00000001;
    if (ready==1) break;
}
/* send data to division accelerator */
pio_write(DVND_BASE, a);
pio_write(DVSR_BASE, b);
/* generate a start pulse */
printf("Start ...\n");
pio_write(START_BASE, 1);
pio_write(START_BASE, 0);
/* wait for completion */
while (1) {
    done = IORD(DONE_TICK_BASE,
        PIO_EDGE_REG_OFT) & 0x00000001;
    if (done==1) break;
}
```

Verilog

```verilog
nios_div1_cpu_unit
(.clk(clk),
 .reset_n(1'b1),
 .out_port_from_the_sseg(sseg),
 .out_port_from_the_start(start),
 .out_port_from_the_dvnd(dvnd),
 .out_port_from_the_dvsr(dvsr),
 .in_port_to_the_quo(quo),
 .in_port_to_the_rmd(rmd),
 .in_port_to_the_ready(ready),
 .in_port_to_the_done_tick(done_tick),
 .sram_addr_from_the_sram(sram_addr),
 .sram_ce_n_from_the_sram(sram_ce_n),
 .sram_dq_to_and_from_the_sram(sram_dq),
 .sram_lb_n_from_the_sram(sram_lb_n),
 .sram_oe_n_from_the_sram(sram_oe_n),
 .sram_ub_n_from_the_sram(sram_ub_n),
 .sram_we_n_from_the_sram(sram_we_n)
);
// instantiate division circuit
div #(.W(32), .CBIT(6)) d_unit
(.clk(clk), .reset(1'b0), .start(start),
 .dvsr(dvsr), .dvnd(dvnd), .quo(quo), .rmd(rmd),
 .ready(ready), .done_tick(done_tick));
```

...
3.3 Qsys-based accelerator

c.3/ bus-based accelerator

- Enables the processor and accelerator to operate in parallel
- Enables accelerator to access resources, e.g., memory with processor
Avalon system interconnects

The Avalon Memory-Mapped Interface is a bus-like protocol that allows two components to exchange data. One component implements a master interface that allows it to request and send data to slave components. A slave component can only receive and process requests.
Avalon-bus memory-mapped transactions

Timing diagram for read/write transactions from the master’s point of view.

Timing diagram for read/write transactions from the slave’s point of view.

module reg16_avalon_interface (clock, resetn, writedata, readdata, write, read, chipselect);
input [15:0] writedata;
output [15:0] readdata;
....
Creating Qsys components in component editor

Give your component’s name
Specify its design files
Map signals
Instantiate component
d. PCI-based accelerators

Using Altera’s automatic OpenCL HW compiler