FYS3240
PC-based instrumentation and microcontrollers

FPGA and GPU

Spring 2012 – Lecture #11
Hardware acceleration

• In computing, Hardware acceleration is the use of computer hardware to perform some function faster than is possible in software running on the general-purpose CPU. Examples of hardware acceleration includes using graphics processing units (GPUs) and instructions for complex operations in CPUs.

• Normally, processors are sequential, and instructions are executed one by one. Various techniques are used to improve performance; hardware acceleration is one of them. The main difference between hardware and software is concurrency, allowing hardware to be much faster than software. Hardware accelerators are designed for computationally intensive software code.

• The hardware that performs the acceleration, when in a separate unit from the CPU, is referred to as a hardware accelerator, or often more specifically as graphics accelerator or floating-point accelerator, etc. Those terms, however, are older and have been replaced with less descriptive terms like video card or graphics card.

• Many hardware accelerators are built on top of field-programmable gate array (FPGA) chips.
FPGA = Field Programmable Gate Array
FPGA advantages

- High reliability
- High determinism
- High performance
- True parallelism
- Reconfigurable
Common Applications for FPGAs

- High-speed control
- Intelligent DAQ
- Digital communication protocols
  - e.g. SPI and Wireless
- Sensor simulation
- Onboard processing and data reduction
  - e.g. video processing
- Co-processing
  - offload the CPU
Processor / FPGA Co-Processor Features

- Operating System
- Legacy code base
- Floating point math
- High-level abstraction (ex. C / C++)
- Complex decision making
- Non time-critical
- Interrupt-driven
- Fixed peripheral set

- Parallel execution
- High computation rates in fixed-point math
- Repetitive calculations
- Nested inner loops
- Fast access to deeply pipelined time-skew buffers
- Wide data words
- Custom peripherals
To configure FPGAs on NI hardware

NI LabVIEW FPGA Module

- Define your own control algorithms with loop rates up to 40 MHz
- Execute multiple tasks simultaneously and deterministically
- Create your own I/O hardware without VHDL coding or board design
- Implement custom timing and triggering logic with 25 ns resolution
- Graphically configure FPGAs on NI reconfigurable I/O (RIO) hardware targets

[+] Enlarge Picture

Data Sheet
From LabVIEW to Hardware

Translation → Optimization → Synthesis → Bit Stream

VHDL Generation → Analysis Logic Reduction → Place and Route Timing Verification → Generation Download/Run

0010100101010001010
1001010010010100101
0010010101010111101
0010110101010001010
0100101001010010010
0100100010010010000
LabVIEW FPGA Hardware Targets

- NI CompactRIO
- NI Single-Board RIO
- NI FlexRIO
- NI R Series Multifunction RIO
- NI Compact Vision System

RIO = Reconfigurable I/O
FPGAs in DAQ-systems ("intelligent DAQ")

- DAQ-cards with a programmable FPGA
- Multi-rate sampling
  - Allows different sampling frequencies on the I/O channels
  - For comparison, when using an “ordinary” DAQ-card (without a user reconfigurable FPGA) all channels must have the same sampling frequency
- User defined processing in the FPGA
- FPGA-based hardware timing/synchronization

Both for PXI and PCI
LabVIEW FPGA VIs

Comparison

Fixed-Point and Integer Arithmetic

Boolean

Data Manipulation
LabVIEW FPGA VIs

Memory
- VI-Scoped Memory
- Memory Write
- Memory Read
- VI-Scoped FIFO
- FIFO Write
- FIFO Read
- FIFO Clear

Execution Structures
- For Loop
- While Loop
- Timed Structure
- Case Structure
- Flat Sequence
- Stacked Sequence
- Diagram
- Conditional
- Local Variable
- Global Variable
- Decorations
- Feedback Node

Synchronization
- Occurrences
- First Call?
- Interrupt

I/O Integration
- I/O Node
- I/O Constant
- I/O Method
- I/O Property

Timing
- Loop Timer
- Wait
- Tick Count
LabVIEW FPGA VIs

Filters and Signal
- Butterworth...
- Notch Filter
- Rational Resa...
- LUT 1D
- Analog Period
- Scaled Window
- FFT
- DC-RMS

Nonlinear Systems
- Backlash
- Friction
- Quantizer
- Dead Zone
- Rate Limiter
- Relay
- Saturate
- Switch
- Boolean Cross...
- Zero Crossing
- Memory Element
- Trigger

Signal Generation
- Sine Wave
- Square Wave
- White Noise

Data Manipulation
- PID (FPGA).vi
- LUT 1D

Linear Systems
- Normalized Increment
- Unit Delay
- Delay
- Zero-Order Hold
- Initial Condition
- Control Filter

New in LabVIEW 8.6
With the LabVIEW FPGA Module you develop FPGA applications on a host computer running Windows, and then LabVIEW compiles and implements the code in hardware.

http://zone.ni.com/devzone/cda/tut/p/id/3358
Host Application for Live Communication with the FPGA

- The **FPGA interface pallet** makes it easy to perform real-time communication between the FPGA and the real-time or Windows host application.
- The *Open FPGA Reference* function is first used to open and run a specified FPGA application.
- The *FPGA Read/Write Control* can be used to read data from the FPGA indicators (outputs) or write data to the controls (inputs).
Digital I/O nodes can take advantage of the specialized structure called the **single-cycle timed loop** in LabVIEW FPGA, with which you can execute code at specified rates ranging from 2.5 to 200 MHz. Using a 40 MHz clock, for example, you can use a single-cycle timed loop to create a 40 MHz counter on any digital line.
LabVIEW FPGA - Parallel Operations

- Two parallel loops with different sampling rates
  - Run in parallel
  - No shared resources between the two loops
Single Cycle Timed Loop (SCTL)

- All operations within the SCTL loop must fit into one clock cycle
- Some functions are not supported in SCTL, among others:
  - Long sequences of serial code
  - Loop timer, wait functions
  - Analog input an analog output I/O nodes
  - Loop structures (While Loop, For Loop)
- LabVIEW Help informs which functions are supported in SCTL
Optimization

Loop rates limited by longest path
- A0 takes about 35 ticks
- DI takes 1 tick (HW Specific)

DI (digital input) limited by AO (analog output) when in same loop

NOTE: A while loop takes 3 ticks

Clock frequency = 40 MHz

38 Ticks ~ 1 uSec
Optimization - Parallel loops

- Separate (parallel) loops allow DI to run independent of AO.
- This allows DI to be sampled 10 times faster by using a separate loop.
Include HDL-code in LabVIEW

- The **IP Integration Node** (replaces the HDL Interface from LabVIEW 2010) can bring in third-party Xilinx IPs.
  - a wizard to import files and configure the interface step by step
- You can also use the IP Integration Node to include your own **VHDL code**
- Once you have configured the node, you can use the IP just like any other LabVIEW node with inputs and outputs.

IP = Intellectual Property
Cycle-Accurate Simulation with ModelSim

• New in LabVIEW FPGA 2010
LabVIEW FPGA IPNet

IPNet

Browse, Download and Share LabVIEW FPGA IP
The LabVIEW FPGA IPNet is your one-step resource for browsing, understanding, and downloading LabVIEW FPGA functions or IP (intellectual property). The site is a collection of FPGA IP and examples gathered from the LabVIEW FPGA function palette, internal National Instruments developers, and the LabVIEW FPGA community.

What's New

Newest Links
Analog Devices ADIS16060 Digital Gyroscope: SPI LabVIEW FPGA IP
Testing Motor Controllers Using FPGA-Based High-Speed Hardware-in-the-Loop

Highlights
Multiply Accumulate (MAC) for Virtex 5 FPGAs
PID Control (with multichannel support)
SPI Protocol

www.ni.com/ipnet
How to Learn More ...

GPUs

- GPU = Graphics Processing Unit
- GPUs can be used as hardware accelerators for numerical/computational tasks
- Can be used in Real-Time High-Performance Computing systems

240 cores

NVIDIA® TeslaTM C1060
Need for Speed ...

- GPUs have more transistors dedicated for processing than a CPU
  - The performance gain when using GPUs can be significant

- A single workstation with two x16 PCIe slots can have two GPUs

- Two TeslaTM C1060 GPUs gives 480 cores.

- The next generation CUDA architecture, code named “Fermi”, have up to 512 CUDA cores.
CUDA

- CUDA = Compute Unified Device Architecture
- CUDA is developed by Nvidia and is a GPU interface for C
CUDA example

```c
void saxpy_serial(int n, float a, float* x, float* y)
{
    for (int i=0; i<n; ++i)
        y[i] = a*x[i] + y[i];
}

// Kjør seriell saxpy
saxpy_serial(n, 2.0, x, y);

__global__ void saxpy_parallel(int n, float a, float* x, float* y)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

// Kjør parallell saxpy med 256 tråder/blokk
int nBlocks = (n + 255) / 256;
saxpy_parallel<<<nBlocks, 256>>>(n, 2.0, x, y);
```
Application examples
Application example II

Building a 100TF datacenter

CPU 1U Server
- 4 CPU cores
- 0.07 Teraflop
- $2000
- 400 W
- 1429 CPU servers
- $3.1 M
- 571 KW

Tesla 1U System
- 4 GPUs: 960 cores
- 4 Teraflops
- $8000
- 800 W
- 25 GPU servers
- 25 Tesla systems
- $0.31 M
- 27 KW

10x lower cost
21x lower power

http://www.top500.org/
LabVIEW and GPUs

- GPUs cannot be directly programmed with LabVIEW

- However, a framework is designed to integrate GPU execution into LabVIEW's parallel execution, to execute the CUDA code

- Some applications are tailor made for deployment to GPUs, such as those related to matrix operations
LabVIEW and GPUs II

• The GPU interface is made up of two LabVIEW libraries – lvcuda.lvlib and lvcublas.lvlib – that contain LabVIEW data types and VIs

http://zone.ni.com/devzone/cda/tut/p/id/11972
Example Hybrid architecture using COTS components

- **Numerical Computing**
  - two NI 8353 multicore computers
  - one nVIDIA® Tesl GPU computing system (with four GPUs)

- **Real-Time Measurement and Control**
  - two PXIe chassis
  - embedded controllers with a multicore CPU
  - FPGA-based data acquisition/control boards