KSETA Workshop 2013

Dipl.-Inform. Steffen Bähr
Dipl.-Inform. Tanja Harbaum
Dipl.-Ing. Christian Amstutz
Dipl.-Ing. Uros Stevanovic

Data Analysis in Hardware
- A Tutorial on VHDL and FPGAs
Overview

- Technologies Overview
- Introduction to Field Programmable Gate Arrays (FPGAs)
  - Technology: Xilinx Virtex5
  - Example: Full Adder
- FPGA Design Flow
  - VHDL
  - Xilinx ISE
Comparison between Technologies

General-Purpose Processors (GPP)
Complex Instruction Set Computer (CISC)
Reduced Instruction Set Computer (RISC)

Special-Purpose Processors
Microcontroller (µC)
Digital Signal Processor (DSP)
Application Specific Instruction Set Processor (ASIP)
Graphic Processing Unit (GPU)

Programmable Hardware
Field Programmable Gate Array (FPGA)

Application Specific Integrated Circuit (ASIC)

Performance, Time to Market
Flexibility, Power Consumption
Example: Xilinx Virtex5 110T

- Configurable Logic Block (CLB)
- Digital Signal Processor (DSP)
- Block RAM (BRAM)
- Digital Clock Memory (DCM)
- I/O Bank
Example: Xilinx Virtex5 110T

- One CLB consists of two Slices
- Programmable Switch Matrix
- Fast local routing inside a CLB
- Global routing between CLBs
Example: Xilinx Virtex5 110T
How to map a full adder into a Slice

- Step 1: Create a truth table

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>Cin</td>
<td>S</td>
<td>Cout</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
How to map a full adder into a Slice

Step 2: Produce Disjunctive Normal Form (DNF)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Cin</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
    s = (\overline{x} \wedge \overline{y} \wedge \text{Cin}) \lor (\overline{x} \wedge y \wedge \overline{\text{Cin}}) \lor (x \wedge \overline{y} \wedge \text{Cin}) \lor (x \wedge y \wedge \text{Cin})
\]

\[
    \text{Cout} = (\overline{x} \wedge y \wedge \text{Cin}) \lor (x \wedge y) \lor (x \wedge \text{Cin})
\]
How to map a full adder into a Slice

- Step 2: Realization with two Lookup Tables

\[
C_{out} = (\overline{x} \wedge y \wedge C_{in}) \lor (x \wedge y) \lor (x \wedge C_{in})
\]

\[
s = (\overline{x} \wedge \overline{y} \wedge C_{in}) \lor (\overline{x} \wedge y \wedge \overline{C_{in}}) \lor (x \wedge \overline{y} \wedge C_{in}) \lor (x \wedge y \wedge C_{in})
\]
How to map a full adder into a Slice

- Step 2: Realization with two Lookup Tables

\[
C_{out} = (\overline{x} \land y \land C_{in}) \lor x \land (y \lor C_{in})
\]

\[
s = (C_{in}) \oplus (x \oplus y)
\]
Full adder in a Virtex5 SliceL
FPGA Design Flow

1. Plan & Budget
2. Translate
3. Map
4. Place & Route
5. Create Code/Schematic
6. HDL RTL Simulation
7. Synthesize to create netlist
8. Functional Simulation
9. Timing Simulation
10. Implement
11. Generate BIT File
12. Configure FPGA
13. Attain Timing Closure
VHDL-Introduction

- VHSCIC HDL – Very high Speed Integrated Circuits Hardware Description Language
- Enforced by the US-Department of Defense for documentation of ASICs
VHDL - Top-Down Design

- Hardware Designs are typically described in a top-down fashion

- [[Diagram showing hierarchical structure of Full_Adder]]
VHDL-Entity

- Description of a module's interface
  - **Ports** as mean of communication
    - **Name**
    - **Direction**
    - **Type**

```vhdl
entity full_adder
  Port(
    x : in std_logic;
    y : in std_logic;
    cin : in std_logic;
    s : out std_logic;
    cout : out std_logic
  );
end full_adder;
```

Entity description of full adder in VHDL

Graphical representation of full adder entity
VHDL-Signals

- Special Container for Data in VHDL
  - Parallel assignments of values
  - Connection between Modules
  - Coupling with timing information for simulations

VHDL signal declaration syntax

1. `signal name : type;`

VHDL signal declaration

1. `signal result : std_logic;`
VHDL-Architecture

- Couples Entities with
  - Structural description
    - Declaration of used modules
    - Instantiation of modules
    - Connection of modules
  - Behavioural description
    - What does the module do?

```vhdl
architecture structure of full_adder is
..
end structure full_adder;
```

Architecture declaration of full adder in VHDL
Component declaration of full adder entity in VHDL

```vhdl
component half_adder
  Port(
    a : in std_logic;
    b : in std_logic;
    s : out std_logic;
    cout : out std_logic;
  );

component half_adder;
```
VHDL-Structural Description(2)

Component instantiation of half adder in VHDL

```vhdl
ha1 : half_adder
    port map(
        a        => x,
        b        => y,
        s        => ha1_s,
        cout     => ha1_cout
    );
```
Component instantiation of half adder in VHDL

```vhdl
ha2 : half_adder
    Port Map(
        a       => ha1_s,
        b       => cin,
        s       => s,
        cout    => ha2_cout
    );
```
### VHDL-Structural Description (3)

#### Connection of OR-Gate in VHDL

```vhdl
1 cout <= ha1_cout or ha2_cout;
```

Full Adder
VHDL-Behavioural Description

- VHDL Process
  - Describes the behaviour of modules
  - Executed in parallel

```
1 s <= a or b; --implicit process syntax
2
3 process( a, b ) --explicit process syntax
4 begin
5
6     if( a = '0' and b = '0') then
7         cout <= '0';
8     else
9         cout <= '1';
10     end if;
11
12 end process;
```

Behavioural description „or“ in VHDL
Half Adder Code Example

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity half_adder is
  PORT ( 
    a : in std_logic;
    b : in std_logic;
    s : out std_logic;
    cout : out std_logic 
  );
end half_adder;

architecture Behavioral of half_adder is
begin
  s <= a and b; --implicit process syntax

  process( a, b ) --explicit process syntax
  begin
    if( a = not b ) then
      cout <= '0';
    else
      cout <= '1';
    end if;
  end process;
end Behavioral;
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full_adder is
  Port(
    x : in  std_logic;
    y : in  std_logic;
    cin : in  std_logic;
    s   : out std_logic;
    cout : out std_logic
  );
end full_adder;

architecture Behavioral of full_adder is
begin

  ha1 : half_adder
    Port Map(
      a => x,
      b => y,
      s => hai_s,
      cout => hai_cout
    );

  ha2 : half_adder
    Port Map(
      a => hai_s,
      b => cin,
      s => s,
      cout => ha2_cout
    );

  cout <= hai_cout or ha2_cout;
end Behavioral;
VHDL-Parallelism

- Modules can be instantiated multiple times and execute in parallel

```vhdl
fa_1 : full_adder
  Port Map(
    x => x_1,
    y => y_1,
    cin => cin_1,
    cout => cout_1,
    s => s_1
  );

fa_N : full_adder
  Port Map(
    x => x_n,
    y => y_n,
    cin => cin_n,
    cout => cout_n,
    s => s_n
  );
```

Multiple Full Adders executing in parallel
VHDL-Clocking

- Using a clock for synchronisation in VHDL Modules
  - Clock signal included in sensitivity list of process
  - Rising or falling clock edge

```vhdl
process(clk)
begin
  if( clk = '1' and clk'event) then
    result <= 1;
  end if;
end process;
```

Diagram showing clocking in VHDL with trigger and execute process.
XILINX Design Tool ISE

A Tutorial on VHDL and FPGAs
Demo Synthesize

Synthesize to create netlist
Demo P&R

Implement

Translate

Map

Place & Route

INPUT/OUTPUT

routet net

Slice

Connect Matrix
Demo P&R
High-Level-Design Approaches

- FPGA design by HDLs:
  - Time consuming
  - Error-prone
  - Difficult verification
  - Large teams
- Algorithms are not designed in HDL

- Is there a faster/simpler way to design FPGAs?

Yes! But ...
Considerations for High-Level Approaches

- There is no "Push-Button" approach
- Unlike for software compilation
- Designer must keep hardware in mind
System Designer

- Design of (Embedded) Systems based on building blocks
- IP-blocks connected by a standard bus
- Available blocks:
  - Arithmetics: Adder, Multiplier, sqrt, sin/cos, ...
  - Digital Signal Processing: FIR filters, FFT, ...
  - Processors, RAMs, ROMs, Peripherial controller, ...

- Limitations
  - Limited to the available blocks (included, purchased)
  - Describing a complex algorithm solely by basic blocks is cumbersome

- Applications: Altera Qsys / Xilinx EDK
Xilinx EDK: Simple Adder
Xilinx EDK: Complete Processor System
High-Level-Synthesis (HLS)

- Converts an algorithm written in C/C++/SystemC to HDL
- C is used as most of its constructs could be supported
- Everything great?
  - Algorithm must be parallelizable
  - Data types should fit hardware
    - Minimal bit width which ensures the accuracy of algorithm
    - Floating point is expensive in hardware
  - Different optimization goals → Design Space Exploration
    - Throughput
    - Latency
    - Chip area
    - Power
- Applications: Xilinx Vivado HLS, Calypto Catapult
Design Space Exploration (Example)

- Vector Addition:

\[
\begin{pmatrix}
  c_0 \\
  c_1 \\
  c_2 \\
  \vdots \\
  c_n
\end{pmatrix} = \begin{pmatrix}
  a_0 \\
  a_1 \\
  a_2 \\
  \vdots \\
  a_n
\end{pmatrix} + \begin{pmatrix}
  b_0 \\
  b_1 \\
  b_2 \\
  \vdots \\
  b_n
\end{pmatrix}
\]

```c
void addVectors(int a[], int b[], int c[], int n) {
    for (int i = 0; i < n; i++) {
        c[i] = a[i] + b[i];
    }
}
```

n=4:

```
a0          b0          a1          b1          a2          b2          a3          b3
  |            |            |            |            |            |            |            |
  |            |            |            |            |            |            |            |
  |            |            |            |            |            |            |            |
  +-----------+-----------+-----------+-----------+-----------+-----------+-----------+
  |            |            |            |            |            |            |            |
  |            |            |            |            |            |            |            |
  |            |            |            |            |            |            |            |
  +-----------+-----------+-----------+-----------+-----------+-----------+-----------+
  |            |            |            |            |            |            |            |
  |            |            |            |            |            |            |            |
  |            |            |            |            |            |            |            |
  +-----------+-----------+-----------+-----------+-----------+-----------+-----------+
        c0    c1    c2    c3
```

31.01.2014   A Tutorial on VHDL and FPGAs
Design Space Exploration – CPU / 1 Adder

a0  b0  a1  b1  a2  b2  a3  b3

c0  c1  c2  c3

CPU
Design Space Exploration – CPU / 1 Adder
Design Space Exploration – CPU / 1 Adder

- Input variables: a0, b0, a1, b1, a2, b2, a3, b3
- Output variable: c0, c1, c2, c3
- CPU connecting all inputs and outputs
Design Space Exploration – CPU / 1 Adder
Design Space Exploration – CPU / 1 Adder

- Variables: a0, b0, a1, b1, a2, b2, a3, b3
- Outputs: c0, c1, c2, c3
- CPU
Design Space Exploration – 4 Adder
Design Space Exploration – 4 Adder

\[ a_0 + b_0 \rightarrow c_0 \]
\[ a_1 + b_1 \rightarrow c_1 \]
\[ a_2 + b_2 \rightarrow c_2 \]
\[ a_3 + b_3 \rightarrow c_3 \]
Design Space Exploration – 2 Adder

\[ a_0 \quad b_0 \quad a_1 \quad b_1 \quad a_2 \quad b_2 \quad a_3 \quad b_3 \]

\[ \text{+} \quad \text{+} \]

\[ c_0 \quad c_1 \quad c_2 \quad c_3 \]
Design Space Exploration – 2 Adder
Design Space Exploration – 2 Adder
### Design Space Exploration – Comparison

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>1 Adder</th>
<th>2 Adder</th>
<th>4 Adder</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Area</strong></td>
<td>1.2</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td><strong>Results/Cycle</strong></td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td><strong>Clock [MHz]</strong></td>
<td>600</td>
<td>400</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td><strong>Throughput [MOPs]</strong></td>
<td>150</td>
<td>100</td>
<td>200</td>
<td>400</td>
</tr>
</tbody>
</table>

- There is not a single optimal solution
- Many of these optimizations needs to be done manually
Simulink / Matlab

- Graphical FPGA Design
- Many Simulink blocks supported
- Verification is integrated

Examples:
- Mathworks HDL Coder
- Altera DSP Compiler
- Xilinx System Generator
Open Computing Language (OpenCL)

- Extension to standard C
- Description of massively parallel algorithms
- Kernels describe parallel parts of algorithms
- Kernel could execute on different computation units
  - CPU
  - GPGPU
  - FPGA

Limitations:
- No stand alone FPGAs – Kernels called by host program
- (so far) only 1 FPGA board supported

Application: Altera SDK for OpenCL
OpenCL – (very simplified) Example

```c
int main() {
    a = readMatrixFromFile();
    b = readMatrixFromFile();

    ConfigureAndCompileKernel();
    CopyDataToDevice();

    RunKernel(addVectors, a, b, c, vector_length)

    CopyDataToHost();
    PrintMatrix(c);
}
```

```c
__kernel void addVectors(a, b, c, n) {
    i = get_global_id(0);
    c[i] = a[i] + b[i];
}
```
FPGA - Applications

- Digital signal processing
- ASIC prototyping
- Computer vision
- Military applications
- Medical applications
- Automotive applications
- Consumer electronics
- Industrial applications
- High performance computing
- Space and aeronautics
Mars exploration rovers (MER)

- Rad tolerant Xilinx XQVR FPGA
- Control of the pyrotechnic operations during descent and landing procedure
- Control of the motors for the wheels, steering, arms, cameras, instrumentation
- On-board re-programmability allowed design changes and updates even after the rover has landed

Curiosity Rover
Trigger and DAQ systems in particle physics

**FPGA-based trigger for NA62 Experiment**

- ATLAS Trigger system
- L1 is hardware based
Banking applications

- Calculating company’s collateral debt obligation (CDO) in near real-time
- Prior to the FPGA solution, main risk model for analyzing CDO portfolio took 8-12 hrs, based on x1000 x86 cores → in case of error, no time to resubmit for the day!
- With the speedup, same risk model took 4 minutes → multiple scenarios throughout the day
- Final hardware system is 40-node hybrid cluster
- Each cluster contains 8 Xeon (24GB) cores with 2 FPGA (Xilinx Virtex 5) (12GB)
- Time-critical, compute-intensive pieces of C++ risk model was ported to FPGA
- Advantage of the FPGA → exploit of the fine-grained parallelism and pipelines → many more calculations per watt vs. the CPU
KIT UFO camera

- Ultra Fast X-ray imaging of scientific processes with On-line assessment and data-driven process control

- High-speed data transfer
- Image-based process control
- Programmable camera
- On-camera image processing
Readout chain and FPGA architecture

- 2.2 MP CMOS sensor
- 340 fps @ full view
- 14 Gb/s streaming with Bus Master DMA
- Virtex-6 Xilinx FPGA
Sub-sampling strategy for rows 'fast reject'

Generating fast reject signals (triggers) @ rows level → starting from a reference image
Subsampling strategy is used to allow a fast readout frame @ kHz range

Reference image

Event trigger

Row s comparison

Number of pixel changed

Beam fluctuation detection

Veto

Row trigger signals

DAQ State Machine in FPGA core

Event Readout

To limit the amount of data and increase the frame rate from sensor, windowing in Y direction is possible. The number of lines and start address can be set by SPI

Readout of a window around the row trigger signal or full frame can be requested

Optionally the multiple windows can be defined when a multiple rows trigger are presents

31.01.2014
A Tutorial on VHDL and FPGAs
Image based trigger – architecture and performance

FPGA architecture

Performance

Small region detection (20 rows)

Large region detection (100 rows)
## CPU, GPU vs. FPGA

Often FPGAs and CPU/GPU are complementary: they co-exist in the same system and perform different tasks.

<table>
<thead>
<tr>
<th>FPGA Advantages</th>
<th>CPU/GPU Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• more flexible processing</td>
<td>• programming a CPU in normally easier than programming an FPGA (no understanding of the digital electronics)</td>
</tr>
<tr>
<td>• more flexible input/output</td>
<td>• faster compilation</td>
</tr>
<tr>
<td>• parallel processing</td>
<td>• easier code portability</td>
</tr>
<tr>
<td>• multi-clock</td>
<td>• lower unit costs - for any volume</td>
</tr>
<tr>
<td>• timing operations</td>
<td>• GPU offers massive parallel execution resources and high memory bandwidth</td>
</tr>
<tr>
<td>• highly customizable</td>
<td>• “fast” applications</td>
</tr>
<tr>
<td>• “real-time” applications</td>
<td></td>
</tr>
</tbody>
</table>
## Applications suitability for GPU and FPGA

<table>
<thead>
<tr>
<th>FPGAs</th>
<th>GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computations involves lots of detailed low-level hardware control operations, with no efficient implementation in high level language</td>
<td>No independence in the data flow and computation can be done in parallel</td>
</tr>
<tr>
<td>A certain degree of complexity is required and the implementation take advantage of data streaming and pipelining</td>
<td>Applications contain a lot of parallelism but involve computations which cannot be efficiently implemented on GPUs</td>
</tr>
<tr>
<td>Applications that require a lot of complexity in the logic and data flow design</td>
<td>Applications have a lot of memory accesses and have limited parallelism</td>
</tr>
</tbody>
</table>
Q & A