Programming for performance in FPGAs using multiple processors and accelerators with C/C++ programming

Kees Vissers  
kees.vissers@xilinx.com
Xilinx
MPSoC 2011
Contents

- Today’s Semiconductor Landscape
- Processors and Pipelines
- C to RTL works
- Next step: ARM processors + FPGA
- Medical Application
- Conclusion
Key Industry Drivers for Change

- Insatiable Bandwidth
- Ubiquitous Connected Computing
- The Programmable Imperative

Broadening Markets
Fewer Companies Can Do it All

Investments from a few must be leveraged by many

Source: IBS March 2010
The Era of ‘Crossovers’
## Processors and Pipelines

<table>
<thead>
<tr>
<th>Design approach</th>
<th>RISC Proc.</th>
<th>Proc. w/ accels.</th>
<th>Folded datapath</th>
<th>Pipelined datapath</th>
<th>Replicated datapath</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock:sample</td>
<td>1000:1</td>
<td>100:1</td>
<td>10:1</td>
<td>1:1</td>
<td>1:10</td>
</tr>
<tr>
<td>Data Rate (200MHz clock)</td>
<td>200Ks/s</td>
<td>2Ms/s</td>
<td>20Ms/s</td>
<td>200Ms/s</td>
<td>2 Gs/s</td>
</tr>
<tr>
<td>Applications</td>
<td>control → audio → mobile video → HDTV → comms → networking</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram showing the relationship between processors, FPGAs, and C to RTL tools.](image-url)
High-Level Synthesis Tools for FPGA

- **Code Restructuring**
  - Macro-architecture description
  - Parameterization
  - FPGA Optimizations

- **Directives (pragmas)**
  - Specify performance
  - Mapping resources

- **C/C++ level verification**
  - Use traditional tools (C/C++ compiler, Matlab)
  - Re-use C/C++ testbench
C to FPGA tools summary

Programming with C to FPGA tools for performance is comparable to programming for performance on a DSP

FPGAs deliver 30x the cost performance benefit compared to this DSP, programmed in C/C++

The results of good RTL design are comparable to what these C to FPGA tools can achieve

Xilinx acquired AutoESL in January 2011
Mimo Sphere decoder application

- Results compared with good standard design: tool is mostly better
- Performance of this problem in the range of 150 Gops (mostly 16 bit)
- All C++ code with AutoESL directives
- Synthesized at 225MHz

<table>
<thead>
<tr>
<th>Metric</th>
<th>SysGen</th>
<th>AutoESL – Expert Result</th>
<th>% Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Time</td>
<td>16.5</td>
<td>5</td>
<td>-9%</td>
</tr>
<tr>
<td>LUTs</td>
<td>27,870</td>
<td>29,060</td>
<td>+4%</td>
</tr>
<tr>
<td>Registers</td>
<td>42,035</td>
<td>31,000</td>
<td>-26%</td>
</tr>
<tr>
<td>DSP48 slices</td>
<td>237</td>
<td>201</td>
<td>-15%</td>
</tr>
<tr>
<td>18K Brams</td>
<td>138</td>
<td>99</td>
<td>-28%</td>
</tr>
</tbody>
</table>
Time to result

AutoPilot: 8x8 RVD-QRD design aimed at 225 MHz - LUT/FF Usage

- Execution time
- FF usage
- LUT usage
- Required ex. time
- SysGen FF usage
- SysGen LUT usage

Processing time for 360 channels (fs)

Post-PAR resource usage (#LUTs, #FFs)

Effort (days)
Zynq Architecture
Matrix Multiply Program;

```c
void main() {
    for(i=0; i<DIM; i++) {
        for(j=0; j<DIM; j++) {
            a_re[i][j] = (i+1);
            b_re[i][j] = (j+1);
            out_re[i][j] = 0.0f;
        }
    }
    matrix_multiply_accelerator(a_re, b_re, out_re);
}
```

The programmer’s view: Partition and Integrate

**Develop Algorithm**
- C/C++

**Perform HW / SW System Partitioning**
- C/C++

**Develop Executable Hardware Specification**
- C/C++

**Develop Synthesizable Model**
- C/C++

**Generate Accelerator Block Using HLS**
- C/C++

**Integrate into Embedded System**

**Debug and Release**
The programmers view: partition the code

Matrix Multiply Program;

```c
void main() {

    for(i=0; i<DIM; i++) {
        for(j=0; j<DIM; j++) {
            a_re[i][j] = (i+1);
            b_re[i][j] = (j+1);
            out_re1[i][j] = 0.0f;
        }
    }

    matrix_multiply_accelerator(a_re, b_re, out_re);
}
```

![Diagram showing Embedded CPU and Accelerator with matrix multiplication example](image-url)
Data movement matters

Matrix Multiply Program;

```c
void main() {
    init_hw_accelerator();
    for(i=0; i<DIM; i++) {
        for(j=0; j<DIM; j++) {
            a_re[i][j] = (i+1);
            b_re[i][j] = (j+1);
            out_re1[i][j] = 0.0f;
        }
    }
    matrix_multiply_accelerator(a_re, b_re, out_re);
}
```

Software programmer
Abstracts from data movement

Conceptually,
all variables are free
And can be used everywhere

Performance programmer
Needs Abstraction
from the details:

- Feedback on total data
- Impact on total time

Many ways to move data:
- memcpy
- DMA, central, distributed
- with ACP, with on chip memory
- flush cache and external mem.
Some interesting challenges

- Simple example: 32 x 32 matrix multiply floating point calculation.
- A 32 x 32 x 4 byte matrix = 4Kbyte, is this page aligned?
- Do we run an OS on the processor(s)? Linux? SMP?
- Do we have the ACP port in cache coherency mode?
- Do we use DMA, what burst length? Buffer sizing?
- Is it faster using the On-chip memory?
- Are the accelerators using the same floating point math, same order of compute?
- Translation of User address space to physical address space?
- Can I stream multiple matrix multiplies back to back?

I’m not an SOC builder, I’m a software programmer: your platform should take care of this……
Abstractions for the software programmer

- Abstract the core generated by C to FPGA tools
- Provide main memory interface abstraction
- Provide DMA abstraction, buffer size abstraction, driver abstraction
- Software Programmers view for the total system
Back Projection problem

- The backprojection algorithm is used in a variety of tomography applications, including CAT scanners.
- Takes raw data from a scan at different angles and reconstructs an image based on that data.
- This design recreates a 256x256 pixel image from a 256x367 dataset (256 angles, each of which is 256*sqrt(2) = 367 elements)
- Floating point application.
- Runs on Arm processors with Neon
- Significant acceleration with 2 dedicated accelerators
The output and our FPGA based prototype
Zynq Products in context

<table>
<thead>
<tr>
<th>Design approach</th>
<th>RISC Proc.</th>
<th>Proc. w/ accels.</th>
<th>Folded datapath</th>
<th>Pipelined datapath</th>
<th>Replicated datapath</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock:sample</td>
<td>1000:1</td>
<td>100:1</td>
<td>10:1</td>
<td>1:1</td>
<td>1:10</td>
</tr>
<tr>
<td>Data Rate (200MHz clock)</td>
<td>200Ks/s</td>
<td>2Ms/s</td>
<td>20Ms/s</td>
<td>200Ms/s</td>
<td>2 Gs/s</td>
</tr>
<tr>
<td>Applications</td>
<td>control → audio → mobile video → HDTV → comms → networking</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Modern Arm processors
Several Gops

Fabric
10 – 500 Gops
Concluding Remarks

- C to FPGA tools work well in the signal processing domain and for floating point calculations.
- FPGA fabric can issue, the ‘equivalent’ of 100 -1000 risc operations every clock cycle.
- Communication latency and bandwidth between the processors and accelerators is important.
- Overlap transfer and compute: DMA, multi-thread.
- Next generation Processors + FPGA fabric become affordable.
- Rapid cost reduction: 28nm, 22nm and further.
- Pre- fabricated Heterogeneous programmable platforms offer power efficient flexibility.
- We do the hard platform construction, so that you can use it.
References


- Kees Vissers, Stephen Neuendorffer, and Juanjo Noguera: **Building real-time HDTV applications in FPGAs using processors, AXI interfaces and high level synthesis tools**. Design Automation and Test Europe Conference (DATE). 2011.


- www.bdti.com/resources/benchmarkresults/hlstcp/autopilot