Performance Overhead with High Level Waveform Development

Stefan Nagel, Michael Schwall, Friedrich K. Jondral

2010 European Reconfigurable Radio Technologies Workshop
June 23-25, 2010
Mainz, Germany
Motivation
Motivation

Waveform

Platform

MIL-STD.exe
MIL-STD.out
MIL-STD.bin
Motivation

Waveform
- MIL-STD.exe
- MIL-STD.out
- MIL-STD.bin
- TETRA.exe
- TETRA.out
- TETRA.bin
- GSM.exe
- GSM.out
- GSM.bin
- W-LAN.exe
- W-LAN.out
- W-LAN.bin

Platform
Motivation

Waveform

MIL-STD.exe
MIL-STD.out
MIL-STD.bin

GSM.exe
GSM.out
GSM.bin

TETRA.exe
TETRA.out
TETRA.bin

W-LAN.exe
W-LAN.out
W-LAN.bin

Platform
Question:

How can we build waveforms running on platforms, we (perhaps) don‘t know?
Waveform Development Process

- To provide portability we followed a design process based on the Model Driven Architecture

CIM → PIM → PSM → Code

Computation
Independent Model:
- Description of functionality
- Specification of air interface

Platform
Independent Model:
- Simulation of functionality
- Without platform specific aspects

Platform Specific
Model:
- Extension of PIM
- With separation on different processing units
- With usage of platform APIs

Code:
- Binary Code running on the platform
Waveform and Platforms

- Used Waveform: IEEE 802.11 a/g
- Key parameters:
  - OFDM system with 64 carrier
  - 16 MHz bandwidth $\rightarrow$ 4 $\mu$s symbol duration
  - Frame structure:
    - 16 $\mu$s Training Sequence (STS, LTS)
    - 4 $\mu$s signal information (BPSK, $r = \frac{1}{2}$)
    - 16 $\mu$s data (QPSK, $r = \frac{1}{2}$)
- Evaluation:
  - Is this possible with Model Based Design on different platforms?
  - If not, which bandwidths and hence data rates are possible?
Waveform and Platforms

- Used Platform 1: USRP

RF-Frontend → ADC/DAC → FPGA → GPP

- ADC:
  - 64 MSPS
  - 12 bit
- DAC:
  - 128 MSPS
  - 14 bit
  - Integrated Interpolation of 4

- FPGA:
  - Altera Cyclone 2
  - 12060 LEs

- GPP:
  - Intel Core 2 Duo
  - 2x 2.26 GHz
  - L2 cache: 3 MB

- USB 2.0
  - Theoretical: 480 Mbps
  - Real: ~ 228 Mbps
  → 7.11 MHz BW
Waveform and Platforms

- Used Platform 2: SFF SDR DP

**RF-Frontend**
- Tunable RF
- Superhet Receiver

**ADC/DAC**
- ADC: 125 MSPS, 14 bit
- DAC: 500 MSPS, 16 bit, Integrated Interpolation of 2, 4 or 8

**FPGA**
- Xilinx Virtex-4
  - SX-35
  - 15360 Slices
  - 192 Multiplier

**DSP**
- TI C64x+
  - 594 MHz
  - L1 cache: 112 kB
  - L2 cache: 64 kB

- VPSS
  - Theoretical: 160 Mbps
  - Real: ~ 52 Mbps
  \[\rightarrow 1.5 \text{ MHz BW}\]
Computation Independent Model

- Take or write documents of the specification

### Computation Independent Model:
- **Description of functionality**
- **Specification of air interface**

### Platform Independent Model:
- **Simulation of functionality**
- **Without platform specific aspects**

### Platform Specific Model:
- **Extension of PIM**
- **With separation on different processing units**
- **With usage of platform APIs**

### Code:
- **Binary Code running on the platform**
Computation Independent Model

Supplement to IEEE Standard for Information technology—
Telecommunications and information exchange between systems—
Local and metropolitan area networks—
Specific requirements

Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications
High-speed Physical Layer in the 5 GHz Band

Adopted by the ISO/IEC and redesignated as
ISO/IEC 8802-11:1999/Amd 1:2000(E)

Sponsor
LAN/MAN Standards Committee
of the
IEEE Computer Society

Reaffirmed 12 June 2003
IEEE-SA Standards Board
Platform Independent Model

- Simulate air interface functionality

**CIM**
- Computation Independent Model:
  - •Description of functionality
  - •Specification of air interface

**PIM**
- Platform Independent Model:
  - •Simulation of functionality
  - •Without platform specific aspects

**PSM**
- Platform Specific Model:
  - •Extension of PIM
  - •With separation on different processing units
  - •With usage of platform APIs

**Code**
- Code:
  - •Binary Code running on the platform
Platform Independent Model

OFDM Transmit Side

Data → Modulation → Pilots & Zeropadding → IFFT → Cyclic Prefix & Symbol Shaping → To Channel

OFDM Receive Side

Data → Demodulation → Extract Symbol → Phase Sync. → Channel Equalization → Carrier Offset → FFT → Time & Freq. Synchronization → From Channel
Platform Specific Model

- Extend PIM for use on USRP

CIM ➔ PIM ➔ PSM ➔ Code

Computation Independent Model:
- Description of functionality
- Specification of air interface

Platform Independent Model:
- Simulation of functionality
- Without platform specific aspects

Platform Specific Model:
- Extension of PIM
- With separation on different processing units
- With usage of platform APIs

Code:
- Binary Code running on the platform
Platform Specific Model

<table>
<thead>
<tr>
<th>GPP</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>USTA</td>
<td>From GPP</td>
</tr>
<tr>
<td>Modulation</td>
<td>Sample Rate Conversion</td>
</tr>
<tr>
<td>Pilots &amp; Zero padding</td>
<td>To DAC</td>
</tr>
<tr>
<td>IFFT</td>
<td>From FPG A</td>
</tr>
<tr>
<td>Cyclic Prefix &amp; Symbol Shaping</td>
<td>To FPGA</td>
</tr>
<tr>
<td>Extract Symbol</td>
<td>Sample Rate Conversion</td>
</tr>
<tr>
<td>Phase Sync</td>
<td>From ADC</td>
</tr>
<tr>
<td>Channel Equalization</td>
<td></td>
</tr>
<tr>
<td>Carrier Offset</td>
<td></td>
</tr>
<tr>
<td>FFT</td>
<td></td>
</tr>
<tr>
<td>Time &amp; Frequency Synchronization</td>
<td></td>
</tr>
</tbody>
</table>
Code

- Generate the code

Computation
Independent Model:
- Description of functionality
- Specification of air interface

Platform
Independent Model:
- Simulation of functionality
- Without platform specific aspects

Platform Specific
Model:
- Extension of PIM
- With separation on different processing units
- With usage of platform APIs

Code:
- Binary Code running on the platform
Platform Specific Model

- Extend PSM for use on SFF

CIM → PIM → PSM → Code

Computation Independent Model:
- Description of functionality
- Specification of air interface

Platform Independent Model:
- Simulation of functionality
- Without platform specific aspects

Platform Specific Model:
- Extension of PIM
- With separation on different processing units
- With usage of platform APIs

Code:
- Binary Code running on the platform
Platform Specific Model
Code

- Generate the code

- CIM
  Computation Independent Model:
  • Description of functionality
  • Specification of air interface

- PIM
  Platform Independent Model:
  • Simulation of functionality
  • Without platform specific aspects

- PSM
  Platform Specific Model:
  • Extension of PIM
  • With separation on different processing units
  • With usage of platform APIs

- Code
  • Binary Code running on the platform
Code

CIM → PIM → PSM → Code

DSP.mdl → DSP.rtw → DSP.c → DSP.out

Real Time Workshop
Target Language Compiler
Code Composer Studio

FPGA.mdl → FPGA.ise → FPGA.bit

System Generator for DSP
Integrated Software Environment
Results:
Results:

![Bar chart showing the number of cycles for different processing blocks with different methods: GPP with VS 6.0, GPP with LCC, and DSP with CCS. The chart includes the following blocks:
- Modulation
- Pilots & Zeros
- IFFT
- Cyclic Prefix

The x-axis represents the number of cycles (x 10^4), and the y-axis represents the processing blocks.

- For Modulation, GPP with VS 6.0, GPP with LCC, and DSP with CCS have similar cycle counts.
- For Pilots & Zeros, GPP with VS 6.0 has the highest cycle count, followed by GPP with LCC and DSP with CCS.
- For IFFT, GPP with VS 6.0 has the highest cycle count, followed by GPP with LCC and DSP with CCS.
- For Cyclic Prefix, GPP with VS 6.0 has the highest cycle count, followed by GPP with LCC and DSP with CCS.

The chart indicates that the methods differ in terms of cycle counts for each processing block.
Results:

![Bar chart showing processing blocks and their respective process times in microseconds. The chart compares the performance of GPP with VS 6.0 and DSP with CCS.]
Results:
## Results:

<table>
<thead>
<tr>
<th>Blocks</th>
<th>Cyclone II on USRP</th>
<th>Virtex-4 on SFF SDR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logic Elements</td>
<td>Slices</td>
</tr>
<tr>
<td><strong>Tx</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APIs</td>
<td>1214</td>
<td>1907</td>
</tr>
<tr>
<td>Sample Rate Conversion</td>
<td>1523</td>
<td>687</td>
</tr>
<tr>
<td><strong>Rx</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APIs</td>
<td>3448</td>
<td>1981</td>
</tr>
<tr>
<td>Sample Rate Conversion</td>
<td>1587</td>
<td>658</td>
</tr>
<tr>
<td>Synchronization</td>
<td>x</td>
<td>5880</td>
</tr>
<tr>
<td>FFT</td>
<td>x</td>
<td>4016</td>
</tr>
</tbody>
</table>
Conclusions:

- Portable Waveform Development is possible with Model Based Design
- Partition of the waveform depends on the processing resources but also on the buses between them
- Code Generation for GPPs is straightforward
- Code Generation for DSPs need fixed point algorithms for lightweight code
- Code Generation for FPGAs is on a starting point but leads to good HDL code
Performance Overhead with High Level Waveform Development