Towards Open Source Toolflow For Designing Astronomical Signal Processing Systems

Nimish Sane\textsuperscript{1}, Alan Langman\textsuperscript{2}, John Ford\textsuperscript{3}, Shuvra S. Bhattacharyya\textsuperscript{1}, and Andrew Harris\textsuperscript{1}

\textsuperscript{1} University of Maryland, College Park, MD, USA
\textsuperscript{2} University of Cape Town, South Africa
\textsuperscript{3} National Radio Astronomy Observatory, Green Bank, WV, USA
Outline

• Current design flow and toolflow
  – Motivation for alternative approaches
• Model based design approach
• Leveraging efforts in other application domains
• Future directions – suggestions
Designing Astronomical Signal Processing Systems

- Push a button on a “Black box”
- Let us accept – this is not happening (at least in the near future)

- We have to go through a systematic design flow visible to the designer
How do we design? (I)

- What is the algorithm?
- What are the input and output data rates?
- Do we need to process the data in real time?
- Which hardware platforms are available?
- Which are the tools available to program those platforms?
- Can we afford it?
- How much time does one need to design?
- How steep is learning curve associated with the tool?
- Would the design be flexible, configurable, scalable, and portable?
How do we design? (II)

We have to implement FFT based filtering.

We start by drawing Simulink block diagram.

We need Matlab/Simulink, XSG.

We can run simulation.

We hope that the final synthesized design works.

We know some smart insects out there such as BEE(2).
Design Flow and Toolflow Questions (I)

• Did we *really* choose hardware at the end?
• Are the available tools and corresponding licenses affordable?
• Did we have a usable application specification before deciding the platform and synthesis tool?
• Could we guarantee working of our design before actually implementing it?

Nimish Sane et al., CASPER Workshop 2010
How do we design? (III)

We have to implement FFT based filtering.

We start by drawing Simulink block diagram.

We need Matlab/Simulink, XSG.

We can run simulation.

We hope that the final synthesized design works.

We know some smart insects out there such as BEE(2), ROACH.
Design Flow and Toolflow Questions (II)

• Is the toolflow independent of hardware technology?
• Are high level components of design flow – application specification, functional and system level verification – platform-independent?
• Do we have scalable, portable, reconfigurable representations?
Current Toolflow: Advantages

• Easy to use
• Reusable high level coarser IPs (e.g. FFT, DDC, PFB, Accumulators, DRAM controllers)
• Support for functional modules with mixed granularity
• DSP support in Matlab/Simulink (e.g. filter-design packages, noise generators)
• Support for bit-level simulation
• Visualization of pipelined processing of streaming data
Current Toolflow: Disadvantages

- Installation and setup – painful (many tools to install, licensing issues, some environment to set up)
- Black-box nature of XSG IPs and toolflow – no access for modification or debugging
- Dependence on a particular tool and vendor
- Matlab is expensive and dodgy
- Non-pipelined logic is hard to debug or understand. State machine blocks offered do not help this
- Simulation environment is not very powerful. Scopes are not flexible
- Design of test-benches more difficult compared to lower level languages
- System level simulation is difficult (e.g. flow of network data, interaction of hardware and software, RF issues)
Model-based Design Flow

- Models of computation
- High level description languages offer scalable representations
- Platform-independent functional prototypes
- Portability across hardware technology
- Faster simulations
- Testing (possibly automated testing)
**Models of Computation (MoC)**

- **Definition:** A **MoC** provides semantics (meaning) for the interaction between functional modules in a system.
- **Strongly influences all significant design processes** – Specification, Simulation, Formal Verification, Implementation/Optimization, Interface to environmental or external systems.
- **Places constraints on the design of specification languages**.
- **Influences the way designer think about applications**.
- **An important area for innovation in domain-specific technology and design research**.
- **Examples** –
  - Von Neumann model (e.g. “C” and other imperative languages)
  - Finite state Machines
  - Kahn Process Networks
  - Dataflow
  - Synchronous/Reactive
  - Discrete-event
Considerations for MoC in Embedded System Design

• Designer should specify *only* design details that are necessary for *correctness*

• Model should facilitate manual *refinement* into implementation alternatives and automatic *verification* and *synthesis*
  – Model should not encourage *over-specification* of the system — artificially constrains refinement and synthesis
  – Should expose high-level structure effectively — facilitates verification and synthesis

• Should match designer’s intuition

• Critical trade-off: tractable verification and optimization vs. expressibility
MoC based languages and tools

- Advanced Design System (Agilent Technologies)
- CAL
- Compaan/Laura (Leiden University)
- DIF (Dataflow Interchange Format)
- Kepler
- OpenDF
- LabVIEW (National Instruments)
- PeaCE (Seoul National University)
- Ptolemy (UC Berkeley)
- SILAGE
- StreamIt
- SysteMoc
Looking Around

• Embedded system design research community
• Synergies with other applications domains
  – Software defined radio (GNU-Radio)
  – Streaming applications (video coding and video standards)
• No “unique” solution, but number of open-source/ open-access tools, open standards used in other domains of embedded system design
Design Flow Using DIF

The DIF Language (TDL) → DIF Specification → DIF-to-C

The DIF Package (TDP)

Dataflow Models
- Static SDF, MDSDF, HSDF, CSDF
- Dynamic CFDF, BDF
- Meta-Modeling PDF, BLDF

DSP Designs
- Signal Proc
- Image/Video
- Comm Sys

DSP Libraries
- VSIPIL
- TI
- Other

Dataflow-based DSP Design Tools
- Ptolemy II (Berkeley)
- ADS (Agilent)
- LabVIEW (National Instruments)
- Sys Gen (Xilinx)
- LWDF (UMD)

Embedded Processing Platforms
- Java
- Java VM
- Other Embedded Platforms
- HDL
- FPGA
- CUDA
- GPU
- C
- DSP

C to Gates

Nimish Sane et al., CASPER Workshop 2010
## Future Directions (Parallel Tracks)

| High level application benchmarks | Automated code generation from application descriptions |
| MoC based application prototype exchange formats | Integrating architecture-level optimizations |
| Platform-independent unit tests and system-level tests | Exploring future hardware and software technologies |
| Mixed grain IP development |  |
Summary

• Though an end-to-end design flow is currently available, it has certain disadvantages
• Model based approach would allow future tools to have mathematically verifiable semantic foundations
• Efforts in other domains of embedded system design should be leveraged
• For developing open source design flow/toolflow we have to look around with an open mind!