OpenCPI & GNU Radio Integration for Heterogeneous Processing

David Pocratsky
9/14/2017
Agenda

- Introduction to OpenCPI
- Comparisons to Existing Frameworks
- Motivation for GRC Integration
- OpenCPI and GRC (Demo)
- Road Map for Future Development
- How to Get More Information
Introduction to OpenCPI

- Open Component Portability Infrastructure

- Component-based framework for developing portable processing applications targeting various processing technologies

- Processing technologies can include multiple field programmable gate arrays (FPGAs), general purpose processors (GPPs), graphical processing units (GPUs)*, and a variety of interconnection technologies
  - Designed to be platform agnostic and easily adaptable to new platforms

- Integrates existing tools for FPGA development such as ISE, Vivado, XSim/ISim, ModelSim, Quartus II, etc.

*in development/coming soon
Introduction to OpenCPI

Portable, reusable components are developed in native language for the final target platform (i.e. VHDL/C++) independent of intended application.

Applications are constructed from existing components.

Application deploys to target platform, with components executing on disparate parts of platform.
OpenCPI – Applications and Components

Application – A collection of interconnected component specs

Component – A specification of a function or operation implemented by Workers to run on specific platforms

Worker – A concrete implementation of a component
OpenCPI – Containers and Assemblies

Container – Execution environment that runs on some platform that executes workers.
OpenCPI - Platforms

Platform – Physical device which houses one or more interconnected Containers and associated I/O
OpenCPI - Platforms

Assembly – Collocated subset of an application built from one or more HDL workers
OpenCPI – Interconnect/Bus/Network & I/O

Interconnect/Bus/Network & I/O – Physical connections between Platforms that is typically abstracted away from the developer
OpenCPI – OpenCPI System

System – A collection of interconnected OpenCPI-enabled Platforms that can be used to execute Applications.
Comparisons to Existing Frameworks

<table>
<thead>
<tr>
<th></th>
<th>GNU Radio</th>
<th>RF NoC</th>
<th>OpenCL</th>
<th>REDHAWK</th>
<th>OpenCPI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Support</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>FPGA Support</strong></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supported with vendor specific extensions</td>
<td>Treats FPGA as black box</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Access FPGA I/O (ADC/DAC)</strong></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>GPU Support</strong></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>Planned for future release</td>
</tr>
<tr>
<td><strong>Platform Agnostic</strong></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Leverages Native Platform Language</strong></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Distributed</strong></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>Support for distributed processing across embedded targets planned for future release</td>
</tr>
</tbody>
</table>
Motivation for GRC Integration

- Provide a front end GUI for the OpenCPI Application Developer to quickly test, design, and run OpenCPI Applications or Assemblies – Mostly work currently done by hand
- Leverage the existing work that was done in GRC that does not fall within OpenCPI's domain
- Introduce OpenCPI to the GNU Radio community in a way that is familiar and easy to use
- Enhance GRC to be able to run certain blocks on various hardware platforms such as FPGAs, GPPs, GPUs, etc
OpenCPI and GRC – General Outline

- Install OpenCPI and build the assets for the desired platforms
- (Optional) Write and build your own OpenCPI workers specific to your application
- Run ocpigr to import the workers into GRC as GRC blocks
- Create flowgraph as you normally would with vanilla GRC
- Set the deployment to target the desired containers for each block
- Build/Run the application
OpenCPI and GRC – ocpigr Tool

- Tool in the OpenCPI repo that will parse all OpenCPI workers/components found in the OCPI_LIBRARY_PATH and translate their XML into GRC block xml
- Creates an OpenCPI Container Platform block for each platform that was encountered during parsing
- Takes advantage of the GRC Domain concept by making a domain for each Platform encountered during parsing

```
[dpocratsky@embdev015 data/tmp/openpci/tools/ocpigr target-linux-c7-x86_64]$ export OCPI_LIBRARY_PATH="$OCPI_LIBRARY_PATH:/data/tmp/ocpi_project"
[dpocratsky@embdev015 data/tmp/openpci/tools/ocpigr target-linux-c7-x86_64]$ /ocpigr -D /data/gnuradio/install/share/gnuradio/grc/blocks
```
OpenCPI and GRC – ocpigr Tool

The ocpigr tool imported the OpenCPI Component Specs under the “OpenCPI” Block Tree.
OpenCPI blocks can be used to create a flow graph just as you would with GNURadio Blocks.
A valid flowgraph can then be generated into an OpenCPI project and run using the standard GRC buttons.
OpenCPI and GRC – Running a Flowgraph

OpenCPI provides deployment logic if a deployment (association of where to run the components of an algorithm) isn't specified.
OpenCPI and GRC – Running a Flowgraph

GRC will autodetect OpenCPI blocks and use the appropriate generate and run options.

Auto-detected OpenCPI block. Generate options have been overridden with the OpenCPI generator.

Generating: /data/gnuradio/install/bin/top_block.xml

Auto-detected OpenCPI block. Generate options have been overridden with the OpenCPI generator.

Executing: /opt/opencpi/cdk/bin/linux-c7-x86_64/opcrun /data/gnuradio/install/bin/top_block_deployment.xml
Each GRC block has a “Container” Property which dictates on which container in the targeted system the block should execute.
OpenCPI and GRC – Set Deployment

Different containers are represented visually with different colors (via GRC Domains)
OpenCPI and GRC – Build Assembly

When Targeting Containers that require Artifacts that need to be built (ex. HDL bit file), the “build” button becomes active and will orchestrate the project generation and building for you.
OpenCPI and GRC – Set Deployment

To move from a simulation environment to actual hardware, simply select a different deployment for the intended target.
OpenCPI and GRC – Set Deployment

Example worker that was built for multiple platforms
Example of a BFSK OpenCPI application which is writing to a named pipe/fifo on the local filesystem.
This allows GRC to read from the named pipe "real time" to either do further processing or verification of current the processing chain.
OpenCPI and GRC – Example BFSK App

This flowgraph implements the RX/demod portion of the app to verify the TX is operating correctly.
OpenCPI and GRC – Example BFSK App

If everything looks good, switch deployments to a different target to build and run on actual hardware. Building is only necessary if the assembly is not already built.
Road Map for Future Development

- Have GRC and OpenCPI blocks talk directly or through a translation block
- Enable GRC to determine if the required artifacts are available and automatically build them if they are not
- Expose more features of OpenCPI to the GRC user
  - Select specific worker
  - Customize HDL Container
- Allow deployment across multiple systems on a network
- OpenCL/GPU support
- Support more platforms “out of the box”
How to Get More Information

• OpenCPI
  - https://www.opencpi.org
  - https://github.com/opencpi/opencpi (branch 2017.Q2)
  - issues@opencpi.org
Backup