IP-XACT extensions for IP interoperability guarantees and software model generation

Tom Perry\textsuperscript{1,2}, Richard Walke\textsuperscript{1}, Rob Payne\textsuperscript{1}, Stefan Petko\textsuperscript{1,2}, Khaled Benkrid\textsuperscript{2}

\textsuperscript{1}Xilinx Scotland  \textsuperscript{2}ISLI/University of Edinburgh
Introduction

» Contributions:
  - IP-XACT extensions
  - Software model generation

» Outline
  - Problem statement
  - Proposed solution
  - Metadata details
  - Results and summary

» Sponsors:
Xilinx LTE physical layer systems
LTE design challenges

- IP is incompatible at “presentation layer”
  - Flag errors?
  - Coerce types automatically?

- System is too complex to implement in HDL from scratch: need a model first
  - Need to generate test vectors in software which specify required data transactions on hardware block interfaces
  - Model development requires time and effort: 16,000 lines of C++
State-of-the-art in Xilinx

- AXI interfaces on cores – “Plug & Play IP” initiative
- IP-XACT metadata for cores

  - Idea: can we use IP-XACT to describe presentation-layer compatibility?

- Bit-accurate software simulation models for each core
- Vivado IP Integrator

  - Idea: can we use IP Integrator to generate software models?
LTE DL TX system in IP Integrator
Current design flow

- HDL
  - IP Packager
  - IP Catalog
    - User HDL
    - XIP
    - XIP
  - System with IP, HDL
    - XIP
    - User HDL

» Hard to do system-level architecture exploration
Our modified design flow

- **IP Packager**
  - **HDL**
  - **MoC metadata, SW model code**
  - **Software model**
    - **XIP**
    - **XIP SW**
    - **User SW**
- **IP Catalog**
  - **User HDL**
  - **XIP**
  - **Test vectors**
- **IP Integrator design**
  - **System with IP, HDL**
  - **XIP**
  - **User HDL**
  - **XIP**

**Data type metadata**
Aim: extend the IP-XACT metamodel to capture additional aspects of Xilinx IP cores

Data type metadata is needed for two purposes:
- Checking IP compatibility
- Specifying test vector encoding

Model of computation metadata is also useful:
- Can associate software processing with “actions”
- Can ensure that latency-sensitive IP cores are not instantiated into a variable-latency system (not discussed in detail here)
Data type storage

Type description added to a port

```xml
<spirit:port>
  <spirit:vendorExtensions>
    <x:dataType>
      ...
    </x:dataType>
  </spirit:vendorExtensions>
</spirit:port>

<spirit:port>
  <spirit:vendorExtensions>
    <x:dataTypeRef
      spirit:vendor="example.com"
      spirit:library="lte"
      spirit:name="resource_block"
      spirit:version="1.0" />
  </spirit:vendorExtensions>
</spirit:port>
```

Type description stored externally and referenced
Leaf types

- 5-bit signed fixed-point type with 4-bit fractional part
  - Must distinguish width of type from width of container, e.g. 8-bit bus
  - If \(<x:\text{bitWidth}>\) not present, type occupies the full size of the container
- Similar approach for integers, bools, floats
- Complex type is a pair of integers or reals
Hierarchical types

```xml
<x:dataType>
  <x:structure>
    <spirit:field>
      <spirit:name>first</spirit:name>
      <spirit:bitWidth>...</spirit:bitWidth>
      <spirit:bitOffset>...</spirit:bitOffset>
    </spirit:field>
    <spirit:vendorExtensions>
      <x:dataType>
        ...
      </x:dataType>
    </spirit:vendorExtensions>
    </spirit:field>
    <spirit:field>
      ...
    </spirit:field>
    ...
  </x:structure>
</x:dataType>
```

▶ Structure

- Reuses `<spirit:field>` element from IP-XACT register definitions
Hierarchical types

```xml
<x:dataType>
  <x:array>
    <x:name>antennas</x:name>
    <x:size>4</x:size>
    <x:dataType>
      <x:array>
        <x:name>codewords</x:name>
        <x:size>4</x:size>
        <x:dataType>
          ...
        </x:dataType>
      </x:array>
    </x:size>
  </x:dataType>
</x:array>
</x:dataType>
```

**Hierarchical array**

- Dimensions are named, allowing incompatible orderings to be detected or corrected
- Each dimension has a size
- Dimensions may have a stride...
Parameterisable types

Need to handle types that change depending on core parameters
- In the DUC/DDC LogiCORE, this can be done with variable stride lengths

TDM: false, NANT: 2

TDM: true, NANT: 4

© Copyright 2012 Xilinx
Dataflow action metadata

- Each component can be described as an actor with “actions” in CAL language

- C++ processing functions can be scheduled using action metadata

- For Xilinx bit-accurate IP simulation models, we reference [wrapped] software entry point in metadata

```xml
<x:action>
  <x:input spirit:busRef="din">
    <x:tokenCount>1</x:tokenCount>
  </x:input>
  <x:output spirit:busRef="dout">
    <x:tokenCount>1</x:tokenCount>
    <x:function>
      <x:name>bitacc_simulate</x:name>
    </x:function>
  </x:output>
</x:action>
```
Results

- Data type and MoC metadata can be used to assist in the generation of a software simulation model which outputs encoded test vectors.

- Original software models: 16,000 lines of C++ code.

- New models: 3,000 lines of DSL (data types, CAL/NL dataflow) generating 18,000 lines of C++, and 4,000 lines of C++ ‘action functions’ created manually.

- Total code requirement reduced by >50%, C++ requirement reduced by ~75%.
Final comments

Data type extensions documented as an IP-XACT “PSS”

Limitations

– Highly dynamic types, e.g. IP optional headers
– Specification of dynamic array sizes
– Differences between XIP software models → need to add wrappers

Ongoing work

– Descriptions of data types in IP Packager (Andrew Dow, Xilinx Edin.)
– Generation of heterogeneous software/hardware systems
– Mapping components to software (simulation model) or hardware

More information available from Stefan and I

– Happy to answer any questions