Connectal

A Framework for Software-Driven Hardware Development

Myron King, Jamey Hicks, John Ankcorn
Quanta Research Cambridge
Suppose you want to build a wheeled zedboard robot

• Buy robot kit
  – Some assembly required
• Start writing software
  – Batteries not included
  – Much assembly required
  – Zedboard embedded Linux tutorial 52 steps!
  – How many steps and files are really required?
  – How do I connect HW to SW?
Suppose you want to build a big data analytics accelerator?

• Use off-the-shelf FPGA board
• Build a Flash daughter card
• Start writing software
  – How many steps and files are really required?
  – Can’t I focus on the application?
  – Can I use python or javascript?
  – What if I change which FPGA board I use?
Talk Outline

• Motivation
• Target Systems
• Connecting Hardware to Software
• Sharing Memory
• What is Connectal?
Target Systems

Accelerating data-centers

Controlling embedded systems

Novel architectures for Big Data

Many common implementation challenges
Zedboard Robot Software Stack Using Connectal

• 3 user-defined components
• Each with its own computational model
• Each with its own programming language
• Can we eliminate the need for the Sensor Device Driver?
• Yes: User-mode access to hardware
• Now only 2 user-defined components!
Connecting Hardware to Software

Verilog
module gyro(
  input CLK,
  ...
);
...
endmodule

Software
gyro->sample(...)

www.connectal.org
BSV reduces the gap

BSV
module gyro(Gyro);

  method Action sample(...);
    ...
  endmethod

  ...
endmodule

Software
gyro->sample(...)

A BSV method is like an AXI stream port to a Verilog module
Asynchronous Remote Method Invocation

• Long latency to hardware -> asynchronous
• Software invokes remote software by sending a message through a socket or shared memory
• Software invokes hardware by sending a message
• Hardware invokes software by sending a message
Invoking Methods through Portals

• Define HW/SW interfaces using BSV

```vhdl
interface StrstrRequest;
  method Action setup(Bit#(8) needle_char);
  method Action search(Bit#(32) haystack_ptr,
                        Bit#(32) haystack_len);
endinterface

interface StrstrIndication;
  method Action found(Int#(32) v);
endinterface
```

• Connectal generates code to marshall/demarshall messages
Software Invokes Hardware

userIfc.methodName() 

fifo.notFullReq() 

fifo.notFullResp() 

fifo.write() 

fifo.rdReq() 

fifo.rdResp() 

app.methodName()
Hardware Wakes up Software

lib.portalExec_poll() → poll() → enableInt() → fifo.enq() → signalInt() → disableInt() → return()

lib.portalExec_event()
Hardware Invokes Software

Hardware
  
  Invokes
  
  Software

main

ind::

wrapper

event

lib

driver

bus

HW

fifos

HW

proxy

app

HW

lib.portalExec_event()
Portal Message Transports

• Just showed memory mapped hardware FIFOs
• Other options
  – Shared memory FIFOs
• Software-to-Software IPC
  – Shared memory FIFOs
  – Unix Sockets
  – TCP
  – Web Sockets (to talk to your Javascript!)
• Binary or JSON message encoding
Sharing Memory (DMA)

• Achieving performance goals often requires direct HW access to shared memory
  – String search example
  – Samples from sensors such as gyroscope

• PortalMem
  – Provides reference-counted sharing of memory buffers between user processes and HW
  – Provides logical/physical translations of non-contiguous physical memory for HW
Shared Memory HW

- Simple MMU for non-contiguous physical regions
- BSV libraries for bus-mastering (DMA engines)
- Infra for sharing full-speed bus access among many hardware clients
The Connectal Framework

1. Easy declaration and invocation of remote methods between host software and FPGA using Portals

2. Direct user-mode access to accelerators from software

3. High performance read/write bus master access to system memory from FPGA

4. Infrastructure for sharing full-speed memory port access between clients in FPGA

5. Portability across platforms (CPUs, Operating Systems, buses, FPGAs)

6. Fully integrated tool-chain support for dependency builds and device configuration
Connectal Framework

- Minimal source code needed
  - App.cpp
  - App.bsv
  - Makefile
  - Pins.json (optional)

- Provides a streamlined flow for building and running HW/SW apps
  - make build.zedboard
  - make run.zedboard

- Cloud-based toolchain available

www.connectal.org 18
Platform Portability and Tool Chain Support

Tool-chain support
• Available as a service at connectalbuild.qrclab.com
• Program some devices directly from your browser
Connectal

• Open source on github:
  – https://github.com/cambridgehackers/connectal

• Documentation (some)
  – http://www.connectal.org/

• Public build service
  – Free for open source projects
  – Compile and run on your zedboard with zero software installs
  – http://build.connectal.org/
BACKUP
What is a Portal?

Using Bluespec System Verilog (BSV), the user declares logical groups of unidirectional “send” methods, each of which is implemented as a fifo channel by the Connectal interface compiler; all channels corresponding to a single BSV interface are grouped together into a single **portal**.
Remote Methods through Portals

1. Define HW/SW interfaces using BSV
2. Invoke interface compiler to generate Wrappers and Proxies
3. Connect user-software and user-hardware using generated “glue”

```
void found(int loc){
  ...}

Main(){
  ...configure(‘a’);
  configure(‘b’);
  search(hptr,hlen);
  ...}
```
Remote Methods through Portals

**Indication Thread**

Connectal Libraries

**Poller**

```c
void* PortalPoller::portalExec(void* __x)
{
    while (1) {
        portalExec_poll(portalExec_timeout);
        portalExec_event();
    }
}
```

User Software

```c
void found(int loc){
    ...
}
Main(){
    ...
    portalExec_start();
    configure(‘a’);
    configure(‘b’);
    search(hptr,hlen);
    ...
}
```

**Linux**

- Portal discovery
- Create /dev/portalXXX
- Register ISR
- MMAP

**Interrupt Service Routine**

**Legend:**
- User generated
- Connectal generated
- Connectal libraries
- Linux

**Indication Proxy**

**FIFO**

**Request Wrapper**

**Request Proxy**

**H/W bus**