bluespec

BSV Training

Section: FPGAs for emulation, modeling and acceleration

Using Bluespec's tools to execute and debug models/designs/tests on FPGA platforms
(even when your target is an ASIC)

www.bluespec.com

Why execute on an FPGA?

- *Emulation*: take RTL developed for an ASIC IP or SoC and execute it on an FPGA for verification purposes,
  - Why? Traditional RTL simulation is 1,000 to 1,000,000 times too slow, because of the complexity and/or the scale of the IP/SoC
  - Primary community: Verification Engineers
- *Modeling*: for architectural exploration, or for early SoC software development, or early firmware development, we execute a model of an IP or an SoC
  - Why? traditional virtual platforms in C, C++, SystemC or RTL are 1,000 to 1,000,000 times too slow, because of the complexity and/or the scale of the model
  - Primary community: Architects, SW and firmware developers
- *Acceleration*: we execute a computationally intensive algorithm on an FPGA
  - Why? Fine-grain, complex parallelism can be executed much faster than on a traditional multi-core or multi-threaded processor, or GPGPU
  - Primary community: Scientists in High Performance Computing (HPC)

bluespec

All these need common capabilities and infrastructure

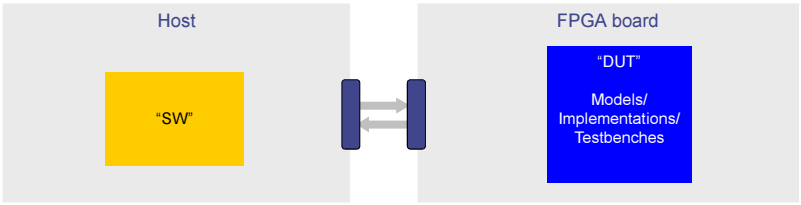
Although emulation, modeling and acceleration are quite different reasons for using an FPGA, and the communities involved are quite different (almost disjoint), they all require the same kind of capabilities and infrastructure:

- High-level design language for FPGA
- Co-execution of host software with FPGA model/IP
- High-level host ⇔ FPA communication facilities
- FPGA debugging support
- Portability across FPGA platforms
- Low cost FPGA platforms

BSV obviously addresses the first requirement.
In this lecture we provide a brief overview of what Bluespec has to offer for the remaining requirements.

bluespec

What is needed to execute your DUT on an FPGA?



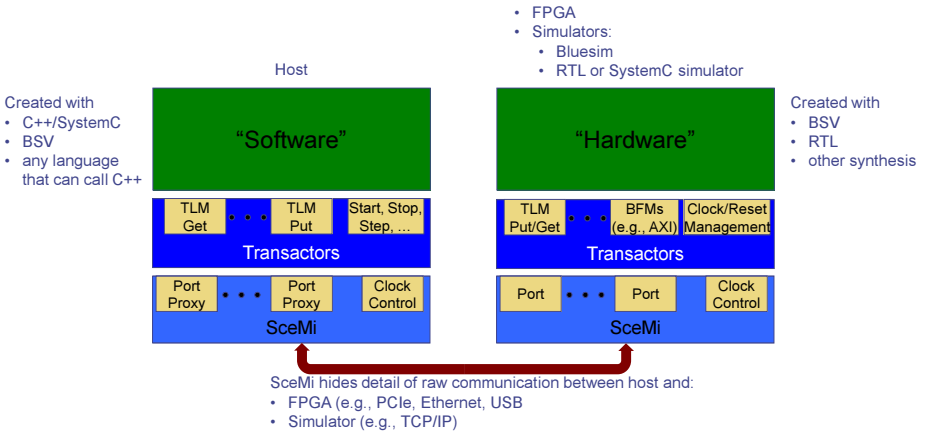
Of course, your DUT needs to be synthesizable to an FPGA, but that is only part of the story …

- High-level control of your DUT from the host (start, stop, reset, single step, …)
- High-level communication of data between host and DUT
  - Abstraction above particular physical communication medium between host and DUT (e.g., PCIe, Ethernet, USB, …)
  - Mapping between different bit-representations of data on host and DUT
  - TLM ("transaction level") transactors, instead of signal-level communication
- Synchronization of host with DUT
- Visibility, from the host, of data inside the DUT
- Libraries of pre-built components usable inside and beside the DUT

Analogy:
- *We rarely write software for a "raw" computer---an OS (operating system) provides more comfortable, higher-level abstractions of the raw hardware and devices*
- *Similarly, Bluespec provides tools and infrastructure that make FPGA execution and debug easy, allowing you to focus on your DUT rather than wrestling with raw FPGA board hardware*

bluespec

Overall architecture of host-FPGA system



Bluespec provides:

- *A full SceMi implementation (accessible from BSV/RTL on HW side, and from BSV/C++ on SW side)*
- *Many transactors to raise level of abstraction from SceMi ports to TLM (Get/Put/Client/Server/…)*
- *Many transactors to connect TLM to standard buses such as ARM AXI, AHB, APB, …*

Note: SceMi (Standard Co-Emulation and Modeling Interface) is an Accellera.org standard

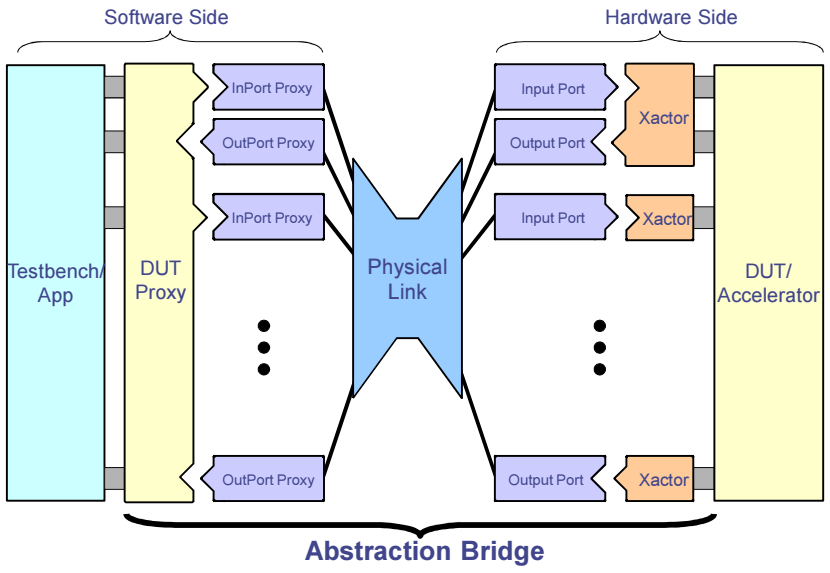
bluespec

In the next few slides, we'll go bottom-up:

- SceMi
- Bluespec's transactors
- Overall flow

bluespec

SceMi communication: message ports and port-proxies

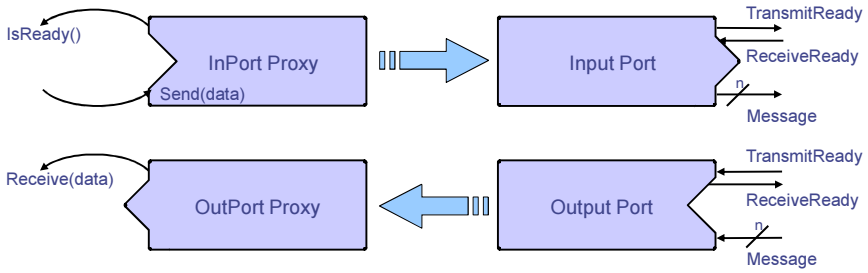


bluespec

SceMi message port APIs

Untimed C/C++/SystemC TB

Device in Emulation



In Accellera's document for SceMi, InPort and OutPort Proxies are specified in C++. User code:

- instantiates proxies
- registers callbacks for IsReady(), Receive()
- initiates a "service thread" which asynchronously monitors the proxy ports and invokes the callbacks when an InPort Proxy is ready to send, or when a message arrives on an Outport Proxy

Bluespec provides these C++ libraries.

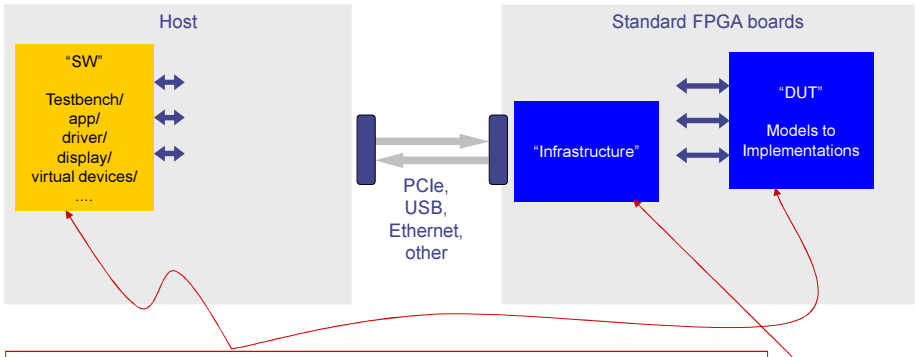
In Accellera's document for SceMi, InPorts and OutPorts are specified in RTL

- User code instantiates ports
- User code registers callbacks for IsReady() and Receive()

Bluespec provides BSV modules (which synthesize to the standard RTL API)

bluespec

The need for DUT clock control



SW is usually "untimed".
DUT usually has some notion of time (RTL clocks or "model clocks").
Need mechanism to regulate (stall, release) DUT clock to keep it "in sync" with SW
(*"time dilation"*).

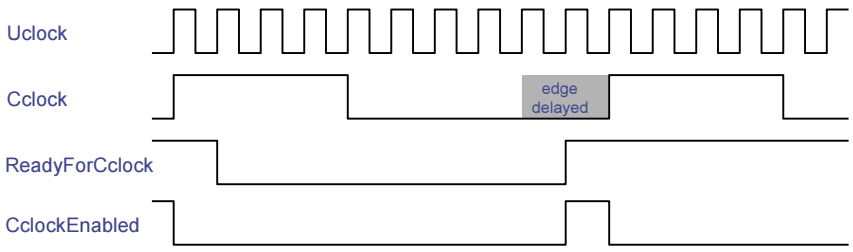
E.g., burst output from DUT delivered as single TLM object to Testbench.
E.g., request from DUT to Testbench, with response expected in "10 cycles".

Note: other parts of the FPGA must continue to be clocked while the DUT is stalled.
E.g., request-response to Testbench while DUT is stalled.

bluespec

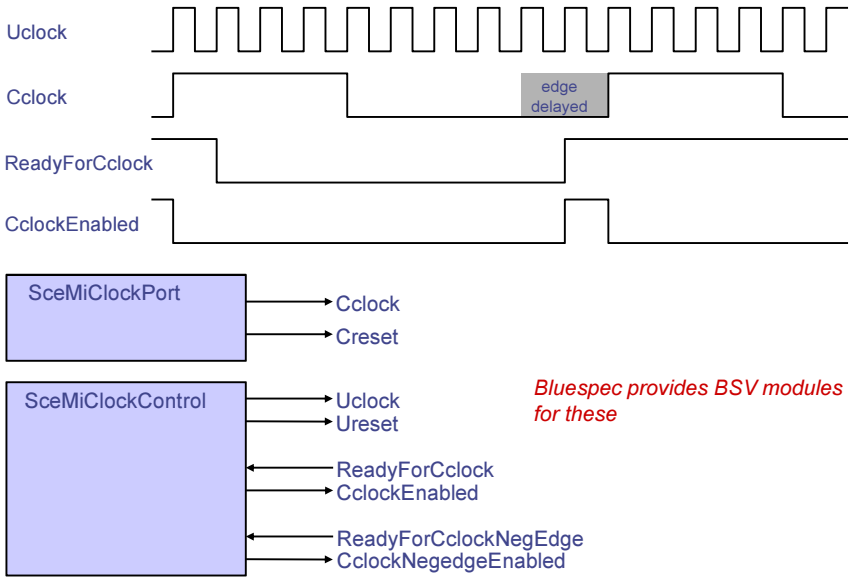
SceMi clock controls: basic concepts

SceMi terminology:
- "cclock" = *controlled* clock for the DUT on the FPGA
- "uclock" = *uncontrolled* clock on the FPGA (for SceMi infrastructure etc.)
- SceMi APIs provide for deriving and controlling cclocks from a uclock



bluespec

SceMi clock controls



Bluespec provides BSV modules for these

bluespec

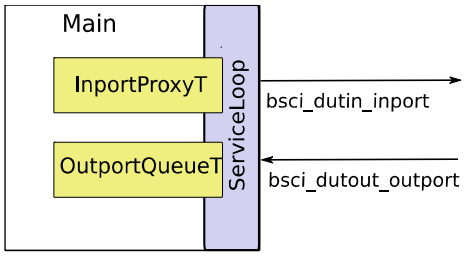
Beyond basic SceMi: TLM abstractions

Basic SceMi has rather low-level communication APIs:
- Callbacks on the SW side
- InPorts and OutPorts on the HW side

Bluespec enables higher-level communications on top of SceMi:
- TLM (Transaction Level Modeling) operations Get/Put on both the SW and the HW side
- Automatic representation-conversion for data types

bluespec

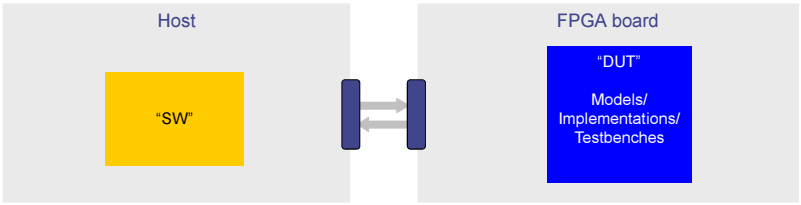
TLM abstractions on the SW side



- Callbacks are messy because they are called “asynchronously” and need to careful about consistent access to state shared with the main computation
- Bluespec provides simple queues:
  - Blocking/non-blocking put/get calls to send/receive messages
  - Callbacks are internal implementation details

bluespec

Data Type conversion



Consider the following similar type definitions in C++ and BSV:

```
// C++

enum Baz {  Red, Green };

struct  Foo {
   int  x;
   bool y;
   Baz  z;
};
```

```
// BSV

typedef enum {  Red, Green } Baz;

typedef struct {
   int  x;
   Bool y;
   Baz  z;
} Foo;
```

Although logically the same, their bit-representations may be quite different.

Bluespec provides automatic converters that can be embedded in the SW-side transactors.

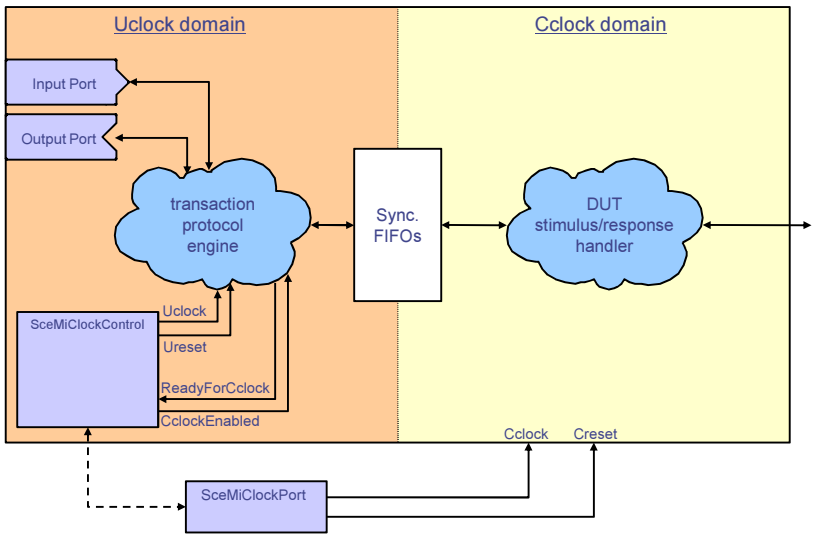
bluespec

TLM abstractions on the HW side

Bluespec provides standard BSV Get, Put, Client and Server transactors

bluespec

BSV HW-side transactors package much SceMi detail



bluespec

Bluespec's SceMi tool flow

A host-FPGA system is a classical "distributed system":
- Hosts and FPGAs are multiple (and dissimilar) computing agents that communicate with each other

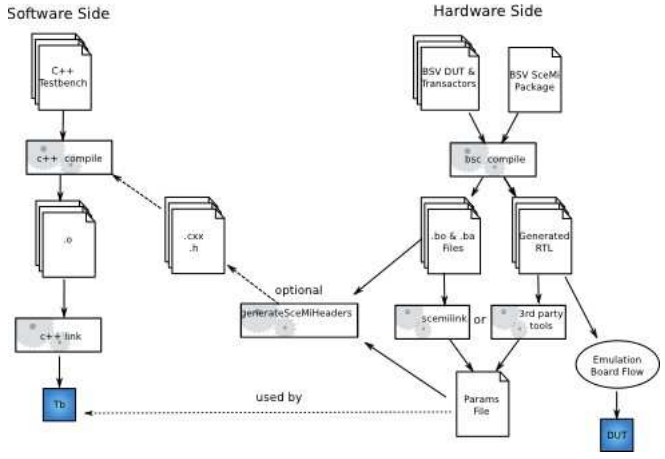
It is therefore not surprising that building and deploying such a system involves similar tasks as in classical distributed computing
- Building multiple executables with different tools
- Protocols for agents to establish communications with each other
- etc.

Bluespec's emulation toolbox has a 'build' script to automate this flow.

References:
$(BLUESPEC_HOME)/training/BSV/tutorials/emVM/tutorial-emVM.pdf
$(BLUESPEC_HOME)/doc/BSV/emVM.pdf

bluespec

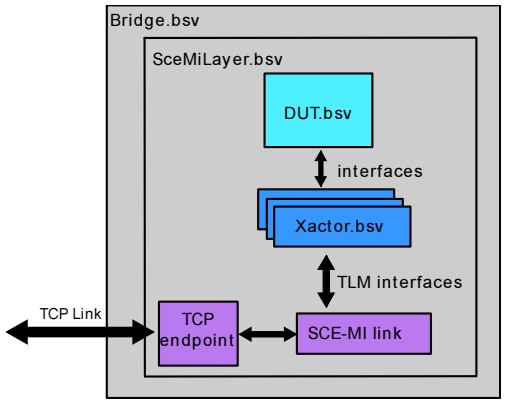
Bluespec's SceMi tool flow



- The Params file is used by the SW side to discover features of the DUT (e.g., number and type of SceMi ports, communication protocol with HW side, etc.)
- The generateSceMiHeaders tool is used to generate type conversions, etc.

bluespec

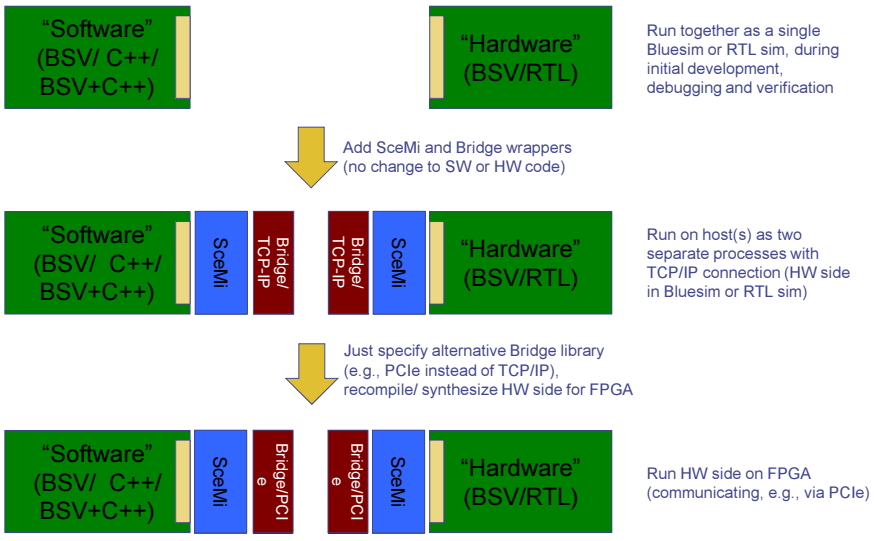
Structure of BSV HW side



- SceMiLayer.bsv is written for each DUT
  - DUT-specific, Link-independent
  - Instantiates DUT
  - Instantiates SceMi ports/transactors and clock controls, connects them to DUT
    - Completely boiler-plate, if the DUT just has a standard interface (like Server)
- Bridge.bsv is a standard library file, chosen for the particular link type (e.g., TCP, PCIe, …)
  - DUT-independent, Link-specific

bluespec

Typical BSV "refinement" methodology



Note: because there is no change to the SW or HW side, and the SceMi and Bridge layers are just wrappers, changes in the SW or HW side can be repeatedly tested in all three scenarios.

bluespec

Bluespec's emulation suite

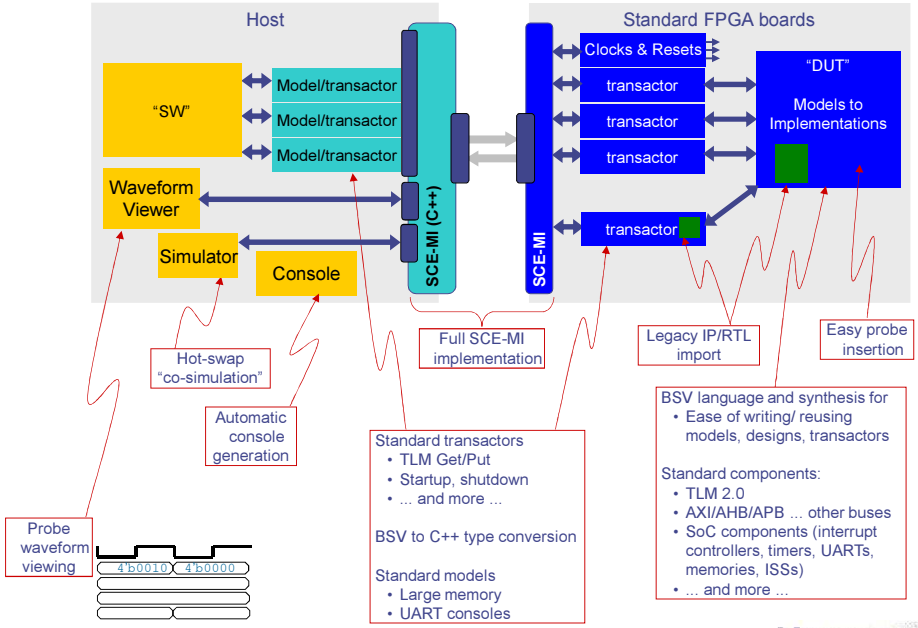
The following slide mentions more advanced features of Bluespec's emulation suite, including:

- Probes
  - Ability to view waveforms from signals inside the FPGA DUT
    - (As usual, in BSV source terms, if the DUT was written in BSV)
- Hot-swap Co-simulation
  - Ability to co-simulate a Verilog module in a Verilog simulator in exact lock-step with the *same* module that is running in the FPGA DUT, thereby providing the full visibility of a Verilog simulator
- Automatic "control console" generation by which you can
  - control your FPGA DUT's execution
  - switch on/off probes, hot-swap co-simulation, etc.

These are described in detail in the references:
$(BLUESPEC_HOME)/training/BSV/tutorials/emVM/tutorial-emVM.pdf
$(BLUESPEC_HOME)/doc/BSV/emVM.pdf

bluespec

Bluespec technology for host-FPGA execution and debug



bluespec

BClib (Bluespec Convey library)

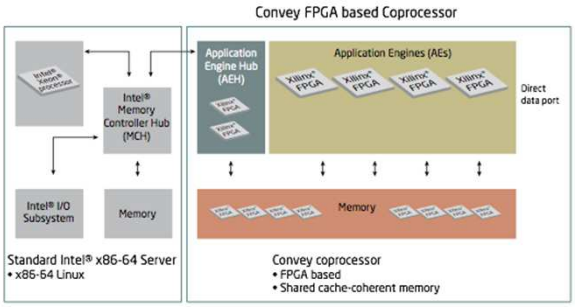
Bluespec's library for developing applications on the Convey HPC (High Performance Computing) platform

(also portable to other host-FPGA HPC platforms)

bluespec

Context: Convey "Hybrid Computing" (HC) platforms

(illustrations from Convey literature, www.conveycomputer.com)



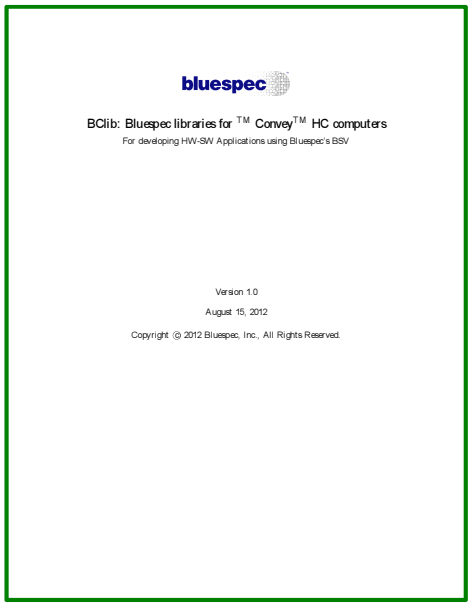
Rack-mount box with two boards
- A standard Intel x86 server board
- A custom board with 4 user-programmable FPGAs (Virtex 5/6) and large, high-performance memory

x86 and FPGAs have shared view of all memory (coherent, same virtual address space)
Intel x86 "coprocessor instruction" protocol to hand-off from x86 to FPGA and back
Since coprocessor instruction semantics are implemented on FPGAs, each such setup is called a "personality"

FPGA memory subsystem:
- 8 memory banks with parallel access from each FPGA (80 GB/s total)
- Pipelined (up to ~256 requests in flight)
- Out-of-order read responses. Requests/responses carry user-supplied 32b tag
- Fire-and-forget writes; weak ordering; memory 'fence' op to wait for write-completions

bluespec

BClib: complete Kit and Reference Manual


bluespec

BClib: Bluespec libraries for ™ Convey™ HC computers

For developing HW-SW Applications using Bluespec's BSV

Version 1.0

August 15, 2012

BClib kit includes:

- C and BSV libraries to dramatically simplify SW-HW linkage and communication
- Integrated Bluesim simulation of full SW-HW app
- BSV facilities to deal with high-performance memory pipelining, memory ordering, memory routing
- Reference Manual with detailed worked-out examples

It is possible to build sophisticated apps now in days/weeks/few months, compared to a year or more

We expect BClib libraries to be easily portable to other host-FPGA platforms.

bluespec

bluespec

End

Questions?
Join online forums at www.bluespec.com, and ask your question,
or send an e-mail to support@bluespec.com