

Connectal Developer's Guide

Contents

1	Connectal Project Structure	3
1.1	Project Makefile	3
1.2	Project Source	4
1.2.1	Interface Definitions	4
1.2.2	Software	4
1.2.3	Hardware	4
1.2.4	Top.bsv	5
2	Compiling and Running Connectal Project	7
2.1	Compiling on AWS	7
2.2	Compiling Locally	8
2.3	Executing The Design	8
3	Flow Control	9
4	Host Interface	10

1 Connectal Project Structure

The set of files composing the input to the Connectal toolchain is referred to as a project. A collection of out-of-tree example projects is available at <https://github.com/connectal-examples>. To illustrate the structure of a project, this chapter uses the example <https://github.com/connectal-examples/leds>, which can be executed using the Bluesim or Zynq target platforms.

1.1 Project Makefile

The top-level Makefile (<https://github.com/connectal-examples/leds/blob/master/Makefile>) defines parameters building and executing the project. In its simplest form, it specifies which Bluespec interfaces to use as portals, the hardware and software source files, and the libraries to use for the hardware and software compilation:

```
1 INTERFACES = LedControllerRequest
2 BSVFILES = LedController.bsv Top.bsv
3 CPPFILES= testleds.cpp
4 NUMBER_OF_MASTERS =0
5 include \$(CONNECTALDIR)/Makefile.connectal
```

INTERFACES is a list of names of BSV interfaces which may be used to communicate between the HW and SW components. In addition to user-defined interfaces, there are a wide variety of interfaces defined in Connectal libraries which may be included in this list.

BSVFILES is a list of bsv files containing interface definitions used to generate portals and module definitions used to generate HW components. Connectal bsv libraries can be used without being listed explicitly.

CPPFILES is a list of C/C++ files containing software components and **main**. The Connectal C/C++ libraries can be used without being listed explicitly.

NUMBER_OF_MASTERS is used to designate the number of host bus masters the hardware components will instantiate. For PCIe-based platforms, this value can be set to 0 or 1, while on Zynq-based platforms values from 0 to 4 are valid.

CONNECTALDIR must be set so that the top-level Connectal makefile can be included. This brings in the default definitions of all project build parameters as well as the Connectal hardware and software libraries. When running the toolchain on AWS, this variable is set automatically in the build environment. (Section 2)

1.2 Project Source

1.2.1 Interface Definitions

When generating portals, the Connectal interface compiler searches the Connectal bsv libraries and the files listed in `BSVFILES` for definitions of all the interfaces listed in `INTERFACES`. If the definition of a listed interface is not found, an error is reported and the compilation aborts. The interfaces in this list must be composed exclusively of `Action` methods. Supported method argument types are `Bit#(n)`, `Bool`, `Int#(32)`, `UInt#(32)`, `Float`, `Vector#(t)`, `enum`, and `struct`.

1.2.2 Software

The software in a Connectal project consists of at least one C++ file which instantiates the generated portal wrappers and proxies and implements `main()`. The following source defines the SW component of the example, which simply toggles LEDs on the Zedboard (<https://github.com/connectal-examples/leds/blob/master/testleds.cpp>):

```
1  #include <unistd.h>
2  #include "LedControllerRequest.h"
3  #include "GeneratedTypes.h"
4  int main(int argc, const char **argv)
5  {
6      LedControllerRequestProxy *device =
7          new LedControllerRequestProxy(IfcNames_LedControllerRequest);
8      for (int i = 0; i < 20; i++) {
9          device->setLeds(10, 10000);
10         sleep(1);
11         device->setLeds(5, 10000);
12         sleep(1);
13     }
14 }
```

The makefile listed `LedControllerRequest` as the only communication interface. The generated proxies and wrappers for this interface are in `LedControllerRequest.h` which is included, along with C++ implementations of all additional interface types in `GeneratedTypes.h`. Line 9 instantiates the proxy through which the software invokes the hardware methods ([Section 3](#)).

1.2.3 Hardware

Connectal projects typically have at least one BSV file containing interface declarations and module definitions. The implementation of the interfaces and all supporting infrastructure is standard BSV. Interfaces being used as portals are subject to the type restrictions described earlier ([Section 1.2.1](#)).

1.2.4 Top.bsv

In `Top.bsv` (<https://github.com/connectal-examples/leds/blob/master/Top.bsv>), the developer instantiates all hardware modules explicitly. Interfaces which can be invoked through portals need to be connected to the generated wrappers and proxies. To connect to the host processor bus, a parameterized standard interface is used, making it easy to synthesize the application for different CPUs or for simulation:

```

1  // Connectal Libraries
2  import CtrlMux::*;
3  import Portal::*;
4  import Leds::*;
5  import MemTypes::*;
6  import MemPortal::*;
7  import HostInterface::*;
8  import LedControllerRequest::*;
9  import LedController::*;
10
11 typedef enum {LedControllerRequestPortal} IfcNames deriving (Eq,Bits);
12
13 module mkConnectalTop(StdConnectalTop#(PhysAddrWidth));
14     LedController ledController <- mkLedControllerRequest();
15     LedControllerRequestWrapper ledControllerRequestWrapper <-
16         mkLedControllerRequestWrapper(LedControllerRequestPortal,
17         ledController.request);
18
19     Vector#(1,StdPortal) portals;
20     portals[0] = ledControllerRequestWrapper.portalIfc;
21     let ctrl_mux <- mkSlaveMux(portals);
22
23     interface interrupt = getInterruptVector(portals);
24     interface slave = ctrl_mux;
25     interface masters = nil;
26     interface leds = ledController.leds;
27 endmodule

```

Like the SW components, the HW begins by importing the generated wrappers and proxies corresponding to the interfaces listed in the project Makefile. The user-defined implementation of the `LedControllerRequest` interface is instantiated on line 15, and wrapped on line 16. This wrapped interface is connected to the bus using the library module `mkSlaveMux` on line 22 so it can be invoked from the software. At the end of the module definition, the top-level interface elements must be connected. A board-specific top-level module will include this file, instantiate `mkConnectalTop` and connect the interfaces to the actual peripherals. The name of the file must be `Top.bsv` and the name of the module must be `mkConnectalTop`.

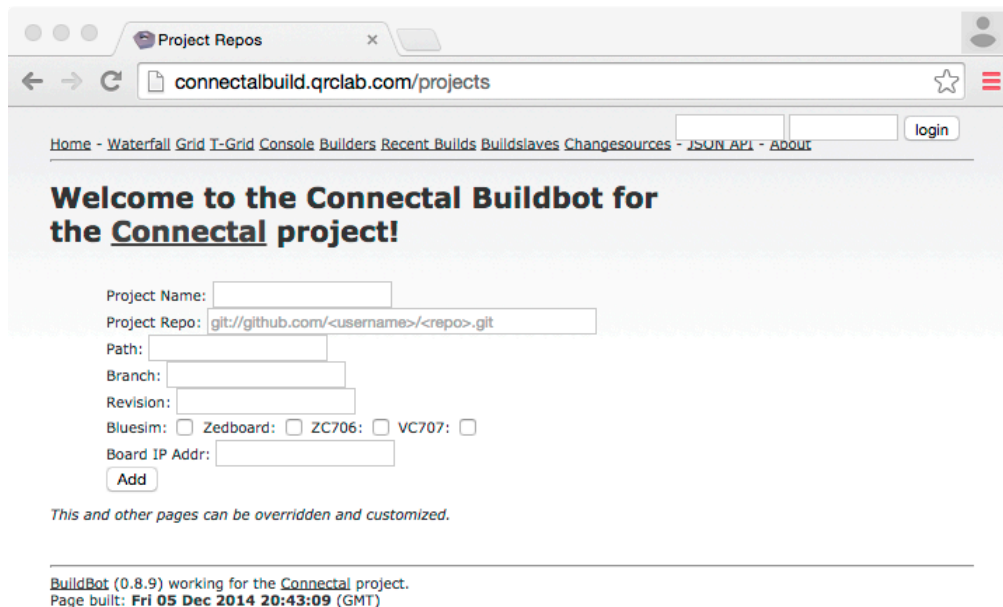
The Bluespec compiler generates a Verilog module from the top level BSV module, in which the methods of exposed interfaces are implemented as Verilog ports. Those ports are associated to physical pins on the FPGA using a physical constraints file. If CPU specific interface signals

are needed by the design (for example, extra clocks that are generated by the PCIe core), then an optional CPU-specific interface can also be used. If the design uses multiple clock domains or additional pins on the FPGA, those connections are also made here by exporting a 'Pins' interface ([Section 4](#)).

2 Compiling and Running Connectal Project

2.1 Compiling on AWS

The Connectal toolchain can be run on Amazon Web Services using the following Buildbot web interface: <http://connectalbuild.qrclab.com/projects>.



The screenshot shows a web browser window with the address bar displaying `connectalbuild.qrclab.com/projects`. The page has a navigation bar with links: [Home](#), [Waterfall](#), [Grid](#), [T-Grid](#), [Console](#), [Builders](#), [Recent Builds](#), [Buildslaves](#), [Changesources](#), [JSON API](#), and [About](#). A `login` button is in the top right. The main heading reads: **Welcome to the Connectal Buildbot for the Connectal project!**

The form contains the following fields and options:

- Project Name:** A text input field.
- Project Repo:** A text input field with the placeholder `git://github.com/<username>/<repo>.git`.
- Path:** A text input field.
- Branch:** A text input field.
- Revision:** A text input field.
- Bluesim:** A section with three checkboxes: ☐ Zedboard, ☐ ZC706, and ☐ VC707.
- Board IP Addr:** A text input field.
- Add:** A button to submit the form.

Below the form, a note states: *This and other pages can be overridden and customized.*

At the bottom, a footer indicates: **BuildBot** (0.8.9) working for the Connectal project. Page built: **Fri 05 Dec 2014 20:43:09** (GMT).

Before submitting a project, you must sign in using your github credentials. Next, enter a name for the project, which will be used for subsequent build requests through the Buildbot web interface. The project must be in a publicly accessible git-hub repository, whose Repo location is entered beginning with “`git://`” as follows [git://github.com/connectal-examples/leds.git](http://github.com/connectal-examples/leds.git). If the project makefile is not in the root directory of the repository, enter it’s relative path in the ‘Path’ field of the form. If a particular branch or revision number are desired, enter these as well. Check the button to select the build target. If you have selected a zynq-based platform and would like the tool-chain to automatically program the device and execute the design as it’s final step, then enter the IP address of your board. This works only because **adb** doesn’t require authentication. SSH keys required to run on PCIe-based platforms are not currently supported. Finally, don’t forget to click ‘Add’. If the project name has already been used, you will be prompted to enter a new one at this point.

2.2 Compiling Locally

Before compiling a project locally, you will need to install the toolchain. After setting the `CONNECTALDIR` to the root of the connectal source tree, enter the command 'Make

2.3 Executing The Design

3 Flow Control

4 Host Interface