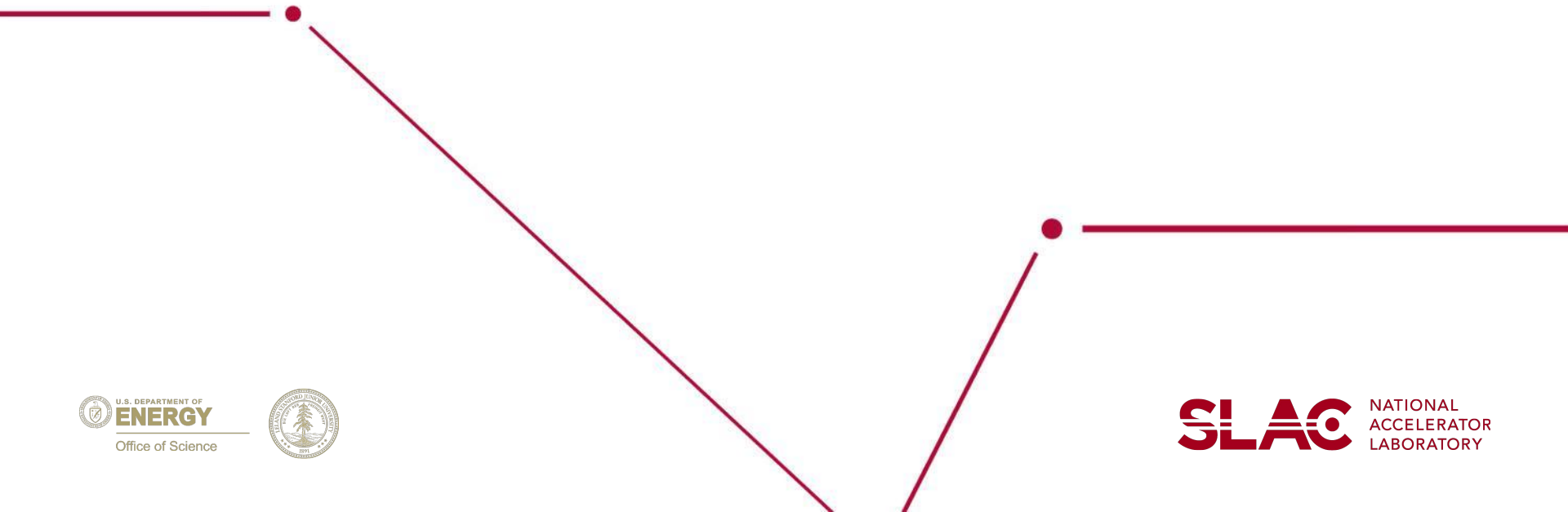
July 28, 2016

TID-AIR Electronics
FPGA Build Systems & Libraries

Larry Ruckman



U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

TID-AIR Electronics' Build System: Vivado Support

SLAC

- Based on a combination of Makefile scripts and TCL scripts
  - Support both a GUI and batch mode design flow
- Version 2014.1 through Version 2016.2 Supported
- Vivado Linux install
  - Better runtime performance than a Windows install
  - Windows users can use Cygwin or Linux Virtual Machine
- Support the following Vivado software tools in our build system:
  - Vivado (RTL programming)
    - VCS Simulation
    - Incremental Compile
    - Partial Configuration
    - IP Integrator (Example: Microblaze)
  - Vivado HLS (C/C++ programming)
  - System Generator (Matlab's Simulink programming)

Build System Support Homepage

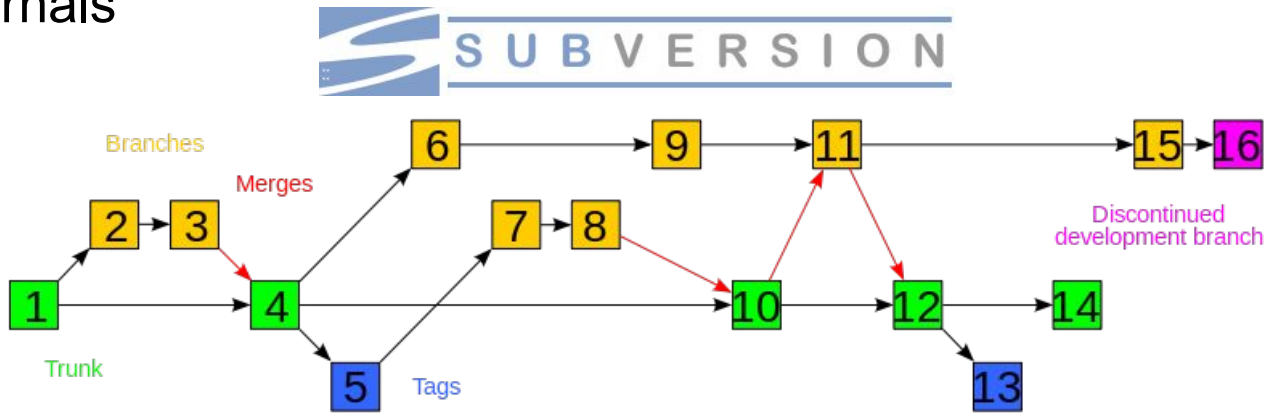
https://confluence.slac.stanford.edu/x/n4-jCg

TID-AIR Electronics' Build System: ISE Support

- We still support ISE support
  - Version 14.7 (Final software Release)
  - Required for Series 6 and earlier support
  - Not recommend for Series 7 FPGAs
- Requires some expert support for setting up a project
- Only batch mode building supported
  - GUI mode building is not supported
- This presentation will be primarily focused on Vivado support

Build System: Firmware Revision Control: Overview

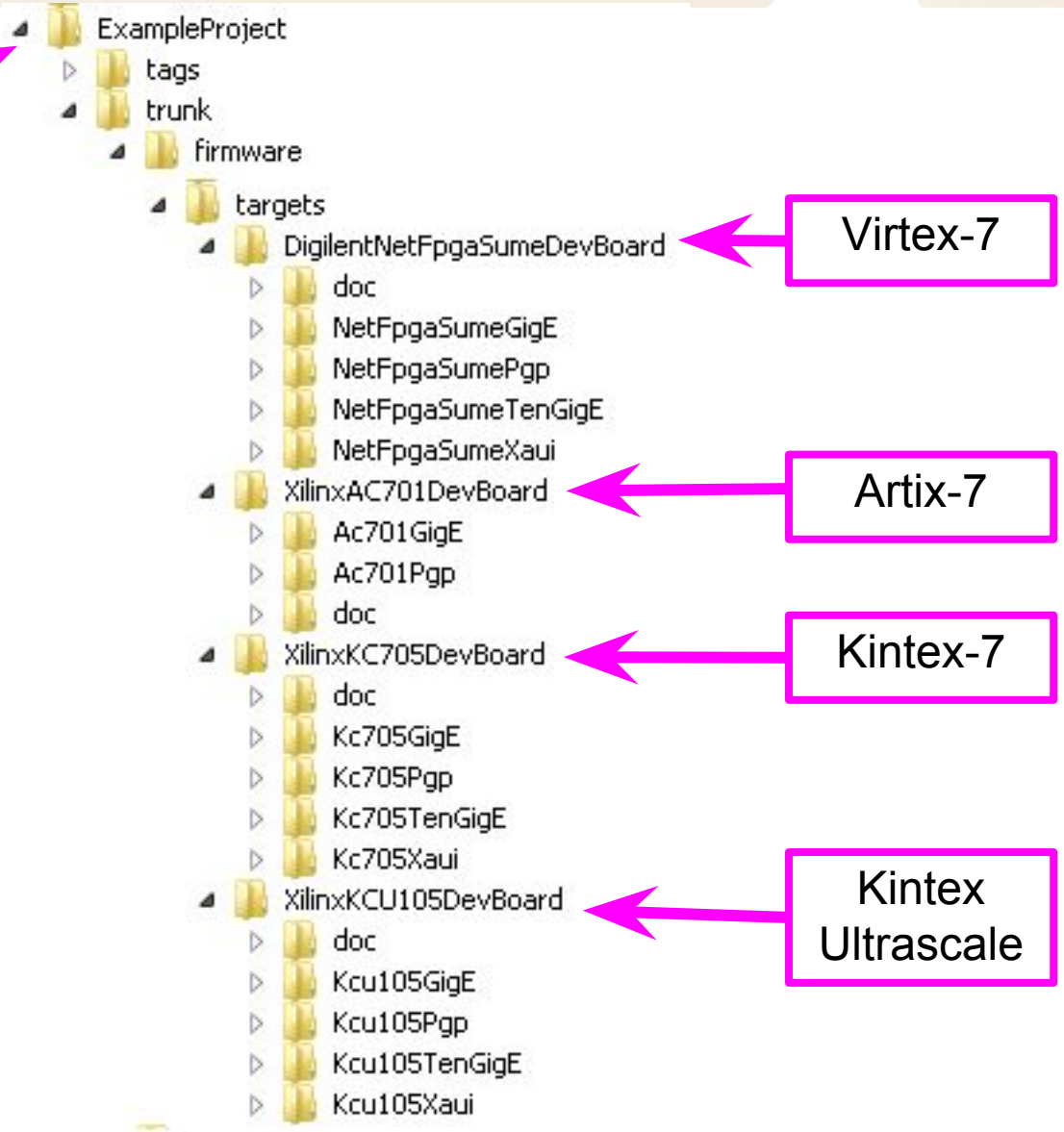
- All firmware is revision controlled in an Apache SVN repository:
  - file:///afs/slac/g/reseng/svn/repos
- Requires a SLAC account
  - Currently working on migrating SVN to GIT to remove this requirement
- Every project and every library has its own directory in the SVN repository
  - Sourcing source code from other project or library is done via SVN externals



Build System: Development Board Support

SLAC

All development board support located in "ExampleProject"



Build System: Design Flow: Overview

SLAC

1. Log into a SLAC Linux server:

   ```
   $ ssh rdusr217
   ```

2. Check out the subversion repository:

   ```
   rdusr217 ~$ cd $home/projects
   rdusr217 ~$ svn checkout  file:///afs/slac/g/reseng/svn/repos/ExampleProject/trunk/ ExampleProject
   ```

3. Setup the Vivado environment and create build soft-link:

   ```
   rdusr217 ~$ cd $home/projects/ExampleProject/firmware/
   rdusr217 ~$ source setup_env.csh
   ```

4. Compile the firmware code:

   ```
   rdusr217 ~$ cd $home/projects/ExampleProject/firmware/targets/XilinxAC701DevBoard/Ac701Pgp
   rdusr217 ~$ make clean
   rdusr217 ~$ make
   ```

5. Program the FPGA using the new image files (.mcs or .bit):

   ```
   rdusr217 ~$ cd $home/projects/ExampleProject/firmware/targets/XilinxAC701DevBoard/Ac701Pgpt/images/
   ```

6. Review the design reports:

   ```
   rdusr217 ~$ cd $home/projects/ExampleProject/firmware/targets/XilinxAC701DevBoard/Ac701Pgp
   rdusr217 ~$ make gui
   ```

Build System: Firmware Revision Control: Checkout

- SVN checkouts can be made either on a local **SLAC server** or **remote server**:
- Local SLAC Server:

```
##################
   ## SVN checkout ##
   ##################
   $ svn checkout file:///afs/slac/g/reseng/svn/repos/ExampleProject/trunk/    ExampleProject
```

- Remote Server:

```
######################
   ## SVN+SSH checkout ##
   ######################
   $ svn co svn+ssh://$USER@rhel6-64.slac.stanford.edu/afs/slac/g/reseng/svn/repos/ExampleProject/trunk/    ExampleProject
```

  - Note: $USER is your SLAC username
  - Note: Add the following lines to your .ssh/config before executing svn+ssh command:
    - ControlMaster auto
    - ControlPath ~/.ssh/master-%r@%h:%p

Build System: Setting up Vivado Software and Licensing

- Local SLAC Server :

```
################################
   ## Setup the Licensing Server Path ##
   ################################
   $ setenv LM_LICENSE_FILE 2100@rdlic1:1717@rdlic1:2100@rdlic2:1717@rdlic2:2100@rdlic3:1717@rdlic3
################################
   ## Source the Vivado Setup Script  ##
   ################################
   $ source /afs/slac/g/reseng/xilinx/vivado_2014.4/Vivado/2014.4/settings64.csh
```

- Remote Server :
  - You (or your IT department) will be responsible for setting up the FLEX_LM licensing server and install Vivado software

Build System: Vivado Build Directory

- After checking out the SVN project, it is required to make a “build” directory
  - Local SLAC Server:

```
##############################
    ## Navigate to firmware directory ##
    ##############################
    $ cd ExampleProject/firmware/
####################################
    ## create a soft-link to the local hard drive  ##
## to prevent building in a slow AFS space ##
    ####################################
    $ ln -s /u1/$USER/build build
```

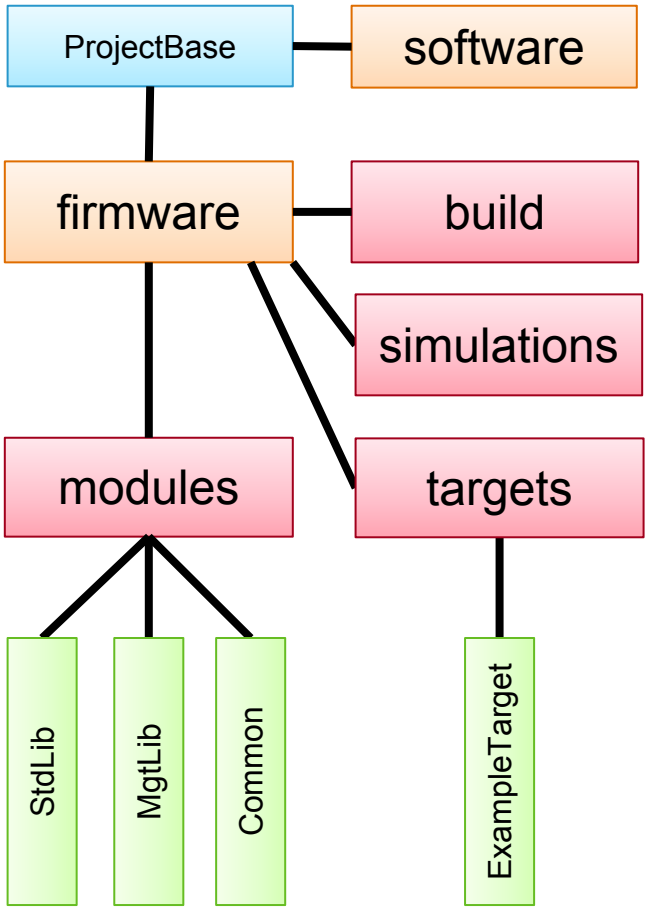
  - Remote Server :

```
##############################
    ## Navigate to firmware directory ##
    ##############################
    $ cd ExampleProject/firmware/
##############################
    ## Make the build output directory ##
    ##############################
    $ mkdir build
```

  - Note: Only source code should be committed to SVN, which is why the build directory is not included in the SVN checkout
  - Note: Using a separate build directory allows a clean restart with make clean, and avoids mixing source code with build files

Build System: File Structure: Overview/Definition

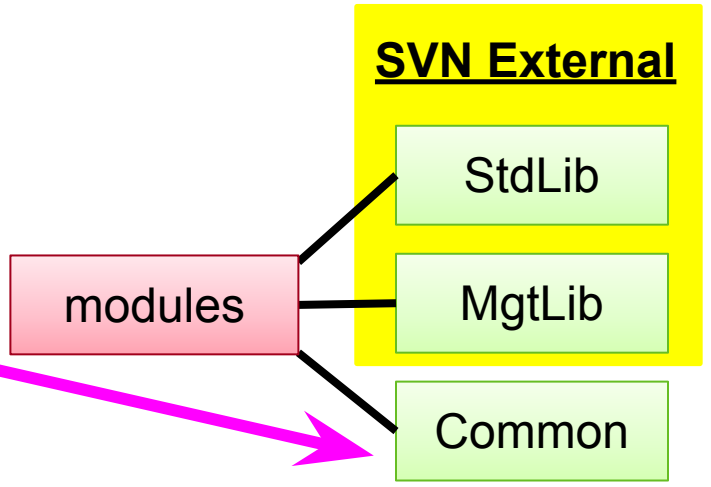
- "ProjectBase"
  - The base directory for both the firmware and software
- "firmware":
  - This directory maintains a standard for file structuring, which is required for TID-AIR electronics' Vivado build system
  - No file structure requirements on the software because independent of the firmware build system
- "modules":
  - Contains bulk of source files, including non-common
- "targets":
  - Contains the top level projects information and Makefiles
  - minimal source lives here
- "build":
  - Common output directory for the targets and simulations
  - Typically, build is soft-linked to a directory on the server's hard drive to improve runtime performance
- "simulations":
  - Contains the top level simulation testbeds



Build System: File Structure: Modules and Common Externals

SLAC

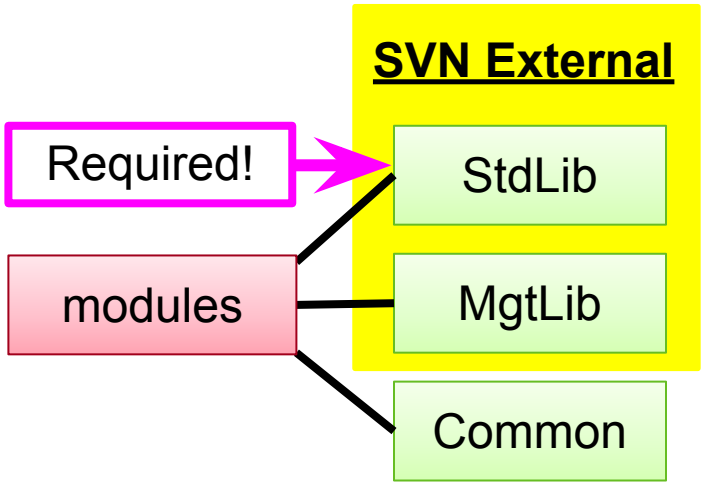
- "modules" directory contains bulk of source files, including non-common
- This example only shows SVN externals
- Additional **non-external project specific** firmware cores can be placed in this directory



Build System: File Structure: Modules and Common Externals

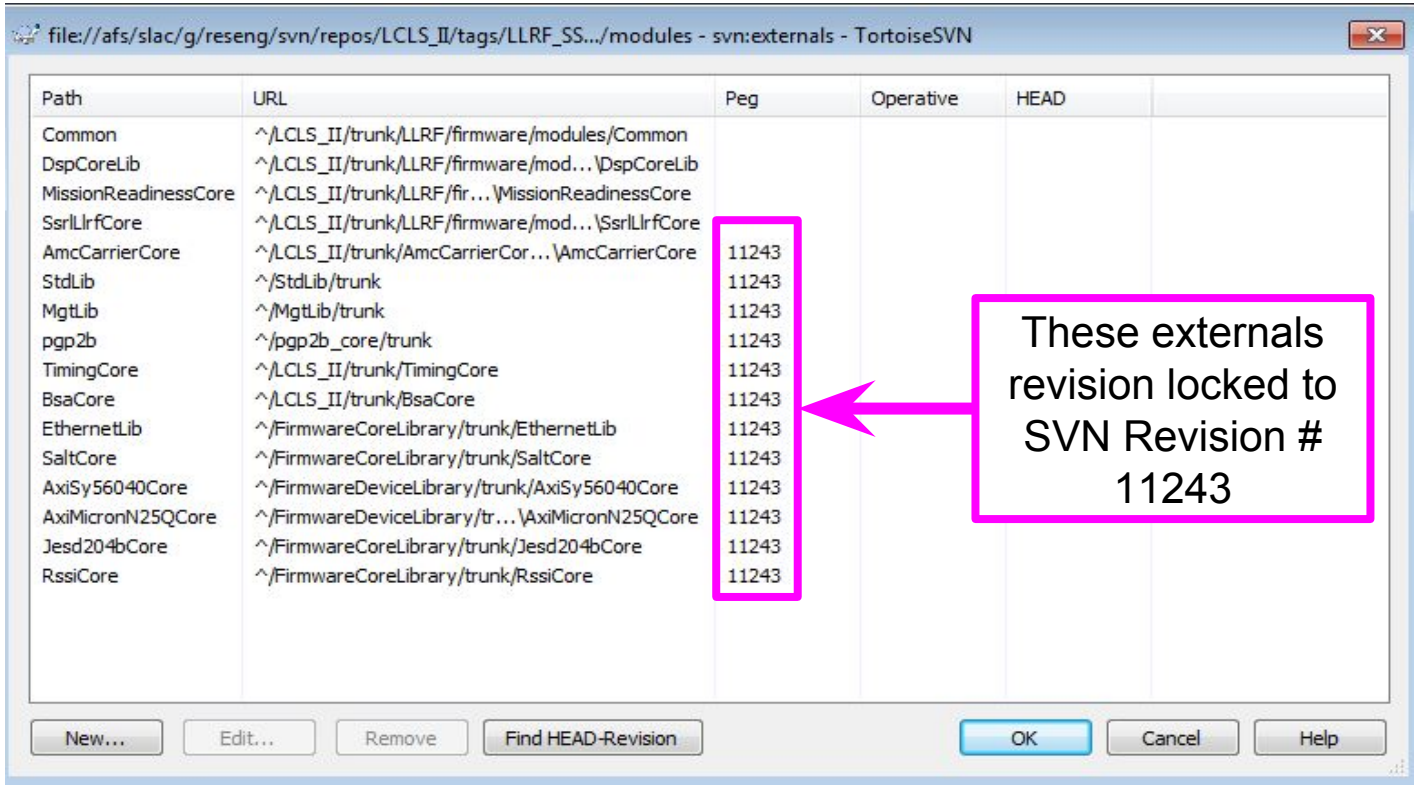
SLAC

- StdLib is the RED electronics' build system standard library
  - Contains commonly used source code and IP cores across many projects
  - Contains all the build system scripts
  - StdLib is the **only required SVN external** because it contains the build system scripts



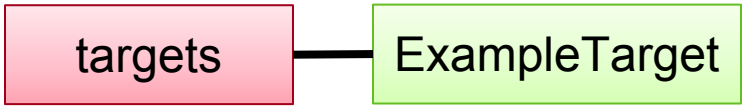
Build System: File Structure: Modules and Common Externals

- Able to revision lock the SVN externals
- Currently adding SVN tagging to common libraries:
  - Example: ^/StdLib/tags/2.1/
  - Tagging is automatic when using GIT



Build System: File Structure: Targets

- All top-level firmware design information is located in the targets directory
- For this example project, there is only one target
- Each design supports only one FPGA firmware build configuration
  - If a project contains more than one FPGA design, then you would use multiple design directories
  - Enables minor top level changes to an FPGA design to support board revision changes (pin location or level change)



Build System: Steering Files

- Steering files are located in all module sub-directories and all design sub-directories.
- These files are used to "**steer**" the Makefile to all the source code for a given design
- Two types of steering Files: common & design specific
- Common Steering Files:
  - **sources.txt**: points to all RTL (.vhd, .v, and .dcp) source code
  - **sim.txt**: points to all simulation (.vhd, .v, .sv, and .vh) source code
  - **cores.txt**: points to all IP cores (.xci) file
  - **block_design.txt**: points to block design firmware register mapping (.bd) file
    - "Block Design" is A.K.A. Vivado IP Integrator
  - **yaml.txt**: points to firmware register mapping (.yaml) file
- Design Specific Steering Files:
  - **constraints.txt**: points to all user constraint (.xdc) files
    - May point to target constraints as well as module constraints

Build System: Steering Files: Source.txt Examples

- file:///afs/slac/g/reseng/svn/repos/FirmwareDeviceLibrary/trunk/AxiMicronN25QCore/sources.txt

```
#############
## VHDL Only
#############
vhdl work rtl/AxiMicronN25QCore.vhd
vhdl work rtl/AxiMicronN25QReg.vhd
```

- file:///afs/slac/g/reseng/svn/repos/FirmwareCoreLibrary/trunk/SaltCore/xilinxUltraScale/sources.txt

```
#############
## VHDL+DCP
#############
vhdl work rtl/SaltUltraScale.vhd
vhdl work coregen/SaltUltraScaleCore.dcp
vhdl work coregen/SaltUltraScaleRxOnly.dcp
```

- file:///afs/slac/g/reseng/svn/repos/LCLS_II/trunk/TimingCore/LCLS-I/sources.txt

```
###############
## VHDL+Verilog
###############
vhdl   work rtl/EvrV1Core.vhd
vhdl   work rtl/EvrV1Databuff.vhd
verilog  work rtl/EvrV1DbusDecode.v
```

Build System: Steering Files: Sim.txt Examples

- file:///afs/slac/g/reseng/svn/repos/LCLS_II/trunk/AmcCarrierCore/firmware/simulations/AmcCarrierDdrMemTb/sim.txt

```
##############################
## VHDL+Verilog+SystemVerilog
##############################
vhdl work    tb/AmcCarrierDdrMemTb.vhd
vhdl work    tb/Ddr3ModelWrapper.vhd
verilog work tb/ddr3_model/ddr3.sv
verilog work tb/ddr3_model/1024Mb_ddr3_parameters.vh
verilog work tb/ddr3_model/2048Mb_ddr3_parameters.vh
verilog work tb/ddr3_model/4096Mb_ddr3_parameters.vh
verilog work tb/ddr3_model/8192Mb_ddr3_parameters.vh
```

Build System: Steering Files: Cores.txt Examples

- file:///afs/slac/g/reseng/svn/repos/LBNE/trunk/firmware/modules/LbneRceCommon/dpm/cores.txt

```
##################
## IP core (.XCI File)
##################
coregen/LbneDpmRamBasedShiftRegister.xci
```

- file:///afs/slac/g/reseng/svn/repos/FirmwareDeviceLibrary/trunk/AxiXadcMinimumCore/cores.txt

```
#########################
## Using the .DCP file instead
#########################
# coregen/AxiXadcMinimum.xci
```

- file://afs/slac/g/reseng/svn/repos/FirmwareDeviceLibrary/trunk/AxiXadcMinimumCore/sources.txt

```
#########################
## Using the .DCP file instead
#########################
vhdl work rtl/AxiXadcMinimumCore.vhd
vhdl work coregen/AxiXadcMinimum.dcp
```

Reduce build time by not having to recompile the IP core every time

Build System: Steering Files: Constraints.txt Examples

- file://afs/slac/g/reseng/svn/repos/ExampleProject/trunk/firmware/targets/XilinxAC701DevBoard/Ac701GigE/constraints.txt

```
##################
## Simple Example
##################
hdl/Ac701GigE.xdc
```

- file://afs/slac/g/reseng/svn/repos/LBNE/trunk/firmware/targets/LbneDpm/constraints.txt

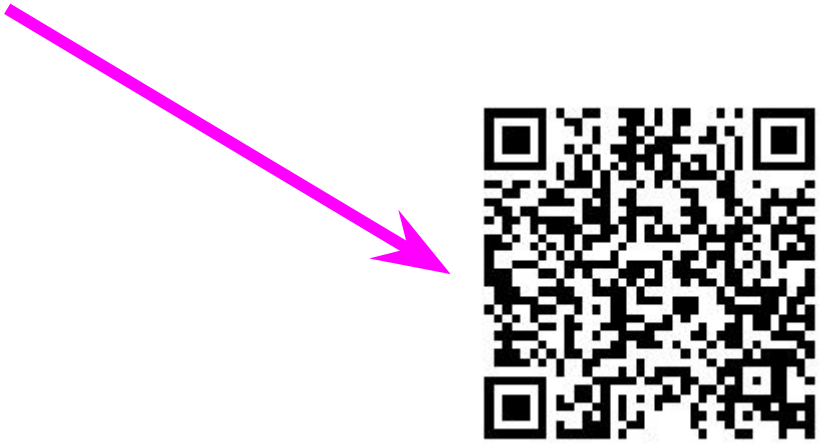
```
######################
## Typical RCE Example
######################
../../modules/DpmCore/hdl/DpmCore.xdc
hdl/LbneDpm.xdc
```

Using a common .XDC for pinout and timing for all RCEs

Build System: Steering Files: Block_designs.txt Examples

- file://afs/slac/g/reseng/svn/repos/FirmwareCoreLibrary/trunk/MicroblazeBasicCore/block_design.txt

```
##################
## Simple Example
##################
bd/MicroblazeBasicCore.bd
```



How to include/build Microblaze

https://confluence.slac.stanford.edu/x/n4-jCg

Build System: Steering Files: yaml.txt Examples

- file://afs/slac/g/reseng/svn/repos/LCLS_II/trunk/AmcCarrierCore/firmware/targets/Kcu105RssiFwTestV2/yaml.txt

```
####################
## Top Level Example
####################
manual yaml/000TopLevel.yaml
manual yaml/DigFpga.yaml
```

- file://afs/slac/g/reseng/svn/repos/FirmwareCoreLibrary/trunk/RssiCore/yaml.txt

```
####################
## Module Example
####################
manual yaml/RssiCore.yaml
```

Build System: File Structure: targets/DESIGN: Required

SLAC

- **Required Files:**
  - **sources.txt**: common steering file
  - **cores.txt**: common steering file
  - **constraints.txt**: design specific steering file
  - **Version.vhd**: Described in detail in a later slide
- **Required Directories:**
  - hdl directory:
    - Contains the top-level RTL and the user constraint file
  - Images directory:
    - Makefile's targeted output dump

Example Target File Tree

```
ExampleTarget/
|-- Makefile
|-- Version.vhd
|-- constraints.txt
|-- coregen
|   |-- PgpClkCore.xci
|   `-- SysClkCore.xci
|-- cores.txt
|-- hdl
|   |-- ExampleTarget.vhd
|   `-- ExampleTarget.xdc
|-- images
|   |-- ExampleTarget_00000001.bit
|   `-- ExampleTarget_00000001.mcs
|-- sources.txt
`-- vivado
    `-- promgen.tcl
```

Build System: File Structure: targets/DESIGN: Optional

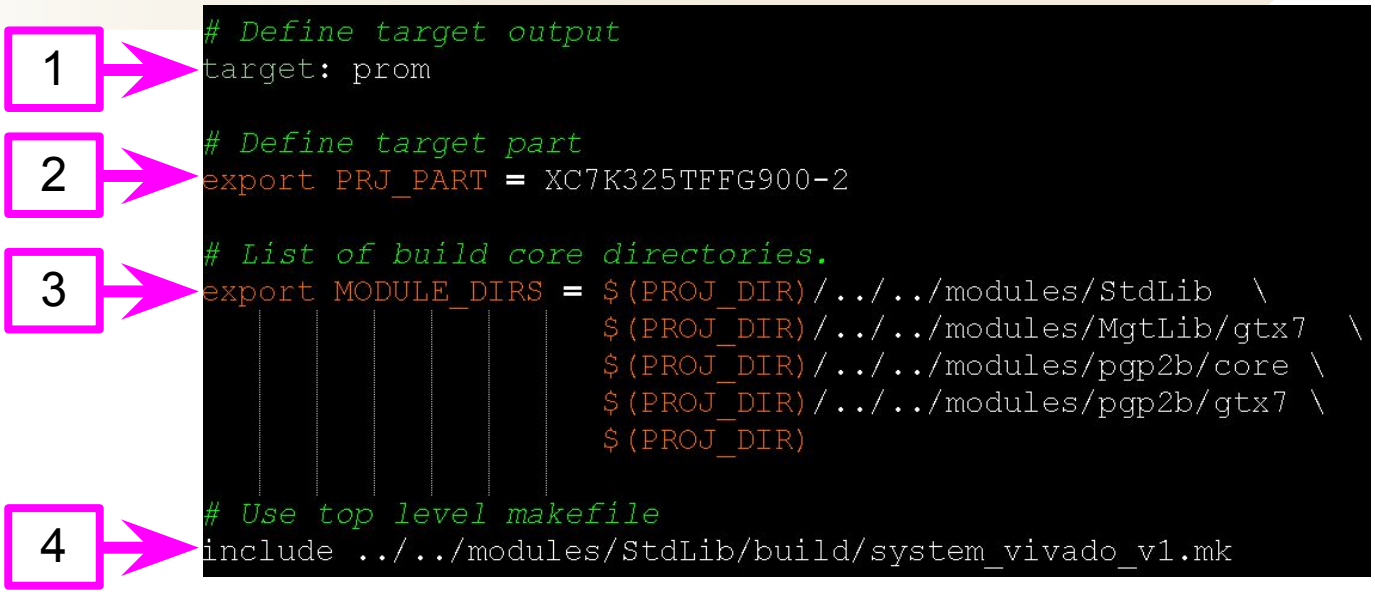
SLAC

- **Optional Directories:**
  - coregen directory :
    - Top-level IP coregen modules
  - vivado directory:
    - TCL Directive scripts for the Vivado build system
      - Typically for advance users
      - Gives to ability to override ANY vivado default configuration
    - No scripts required for simple .bit generation
    - promgen.tcl required for .mcs generation to define the PROM attributes

Example File Tree

```
ExampleTarget/
|-- Makefile
|-- Version.vhd
|-- constraints.txt
|-- coregen
|   |-- PgpClkCore.xci
|   `-- SysClkCore.xci
|-- cores.txt
|-- hdl
|   |-- ExampleTarget.vhd
|   `-- ExampleTarget.xdc
|-- images
|   |-- ExampleTarget_00000001.bit
|   `-- ExampleTarget_00000001.mcs
|-- sources.txt
`-- vivado
    `-- promgen.tcl
```

Build System: File Structure: targets/DESIGN: Makefile



- Only four parameters to be set
  1. Target: Set the build type
     - Typical Options: gui, syn, bit, prom, dcp, or vcs
  2. PRJ_PART: Sets the targeted FPGA's part number
     - This values is passed to the Vivado Project settings
  3. MODULE_DIRS: Defines which modules to include in the target
  4. "include" points to the primary Vivado Makefile
     - Currently, only one version (system_vivado_v1.mk)

Build System: File Structure: targets/DESIGN: Version.vhd

```
-------------------------------------------------------------------------------
-- file:///afs/slac/g/reseng/svn/repos/ExampleProject/truck/firmware/targets/ExampleTarget/Version.vhd
-------------------------------------------------------------------------------
-- Project    : Example Project
-- File       : Version.vhd
-- Author     : Larry Ruckman  <ruckman@slac.stanford.edu>
-- Company    : SLAC National Accelerator Laboratory
-- Created    : 2015-01-09
-- Last update: 2015-01-09
-- Platform   : Vivado 2014.4
-- Standard   : VHDL'93/02
-------------------------------------------------------------------------------
-- Description:
-- Version Constant Module
-------------------------------------------------------------------------------
-- Copyright (c) 2015 by SLAC. All rights reserved.
-------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

package Version is

constant FPGA_VERSION_C : std_logic_vector(31 downto 0) := x"00000001"; -- MAKE_VERSION

constant BUILD_STAMP_C : string := "ExampleTarget: Built Fri Jan  9 11:30:22 PST 2015 by ruckman";

end Version;

-------------------------------------------------------------------------------
-- Revision History:
-- 01/09/2015 (0xCEC83000): Initial Build
-------------------------------------------------------------------------------
```

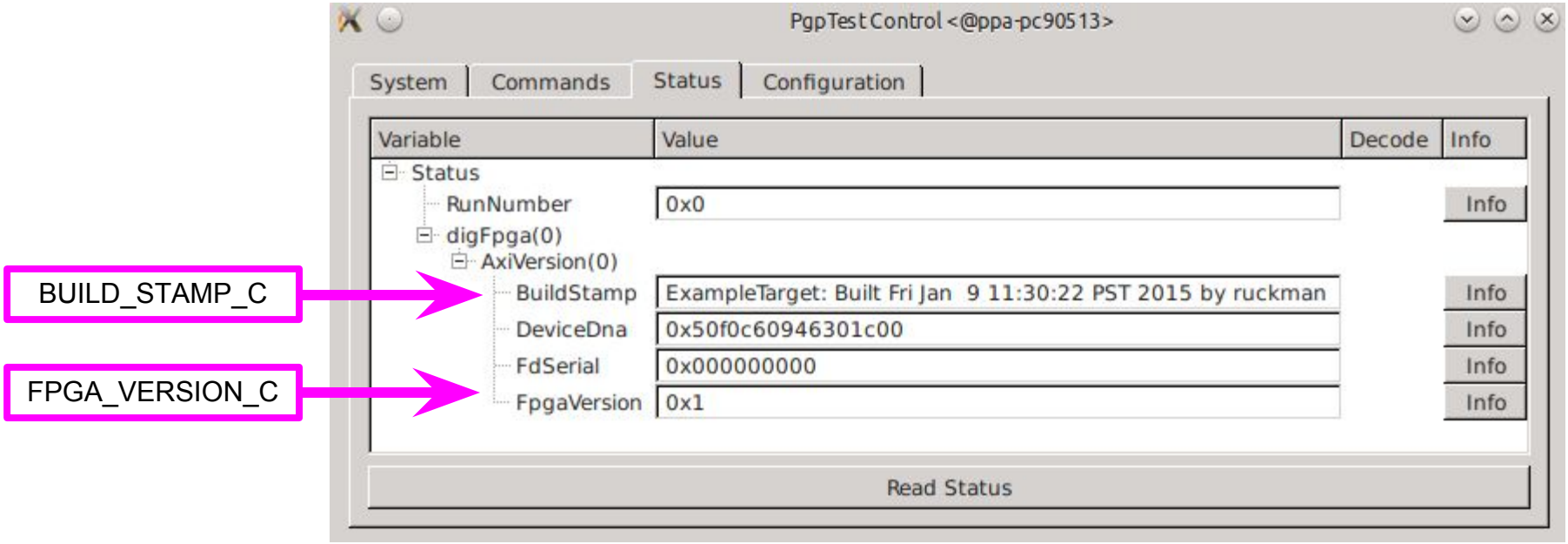
Both constants can be synthesized in firmware

Manually Generated Constant

Auto Generated Constant

Build System: File Structure: targets/DESIGN: Version.vhd

- Supported by AxiVersion.vhd in StdLib
- Firmware constants accessible via software



Build System: Other Example Projects

SLAC

- file:///afs/slac/g/reseng/svn/repos/ExampleProject/trunk
- file:///afs/slac/g/reseng/svn/repos/ExampleRceProject/trunk
- file:///afs/slac/g/reseng/svn/repos/Atlas/trunk/AtlasCsc/truck
- file:///afs/slac/g/reseng/svn/repos/heavyp/trunk
- file:///afs/slac/g/reseng/svn/repos/LBNE/trunk
- file:///afs/slac/g/reseng/svn/repos/cob_dpm/trunk

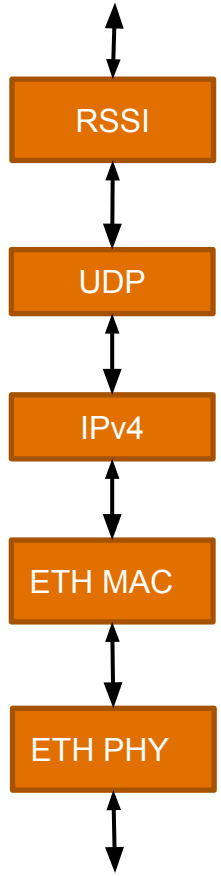
Build System: IP Cores: Standard Library (StdLib)

- file:///afs/slac/g/reseng/svn/repos/StdLib/trunk
- RAMs
  - All memory components placed as parameterized VHDL modules
  - Avoid Vivado IP or coregen usage
  - Supports both distributed ram and block ram
  - Variable sizes
  - Support for other architectures (Altera etc.)
- FIFOs
  - All FIFOs placed as parameterized VHDL module
  - Avoid Vivado IP or coregen usage
  - Supports ASYNC and SYNC FIFOs
  - Variable depths and cascade levels
  - Parameterized full thresholds
  - Supports both distributed ram and block ram
  - Choice between RED electronics' control logic and Xilinx built in
  - Support for other architectures (Altera etc.)
- Metastability Synchronizers
- Coders (8B10B, Gray, CRC)
- I2C
  - Both slave and master
  - 7-bit and 10-bit address support
- SPI
  - Both slave and master

Build System: IP Cores: Ethernet Library (EthernetLib)

- file:///afs/slac/g/reseng/svn/repos/FirmwareCoreLibrary/trunk/EthernetLib
- Support multiple Ethernet layers in firmware:
  - 1st/2nd Layer Ethernet Modules:
    - 4 lane 10 GigE PHY/MAC (XAUI)
    - 4 lane 20 GigE PHY/MAC (DXAUI)
    - 1 lane 10 GigE PHY/MAC (10G-BASER)
    - 1 GigE PHY/MAC (1000BASE-SX)
  - 3rd Layer Ethernet Modules:
    - IPv4 Engine
    - Raw Ethernet Framer
  - 4th Layer Ethernet Modules:
    - UDP Engine
  - 5th Layer Ethernet Modules:
    - Reliable SLAC Streaming Interface (RSSI)
    - Reliable communications layer based upon RUDP (Cisco implementation: refer to RFC-908 and RFC-1151)
    - file:///afs/slac/g/reseng/svn/repos/FirmwareCoreLibrary/trunk/RssiCore

RSSI Documentation



https://confluence.slac.stanford.edu/x/1IyfD

Build System: IP Cores: Additional Libraries

SLAC

- PGP2B
  - AXI4-Stream based protocol over Xilinx GTP/GTX
  - Support for non-Xilinx architectures (Altera, ASIC, etc)
  - Unlimited frame size
  - Frames delivered as cells with CRC protection
  - Fixed latency support for timing and trigger delivery
  - Virtual channel support for QOS
  - 8B10B based up to 5.0 Gbps
  - 64/66 version in early development for > 5.0 Gbps
  - 96% efficient after 8B/10B or 64/66 losses

TID-AIR Protocols Documentation

https://confluence.slac.stanford.edu/x/WRmVD

Build System: Design Flow: Both GUI and Batch mode

SLAC

- Support both a GUI and batch mode
- Batch mode:
  - Pros:
    - High performance building
    - No X11 forwarding required
    - Minimal build overhead
  - Cons:
    - Cryptic error messaging
    - Difficult to analysis design from terminal printouts
- GUI mode:
  - Pros:
    - Native Vivado interaction and design analysis tools
    - User friendly error messaging
  - Cons:
    - Very X11 intensive
    - Increased runtime compared to batch mode

Start Batch Mode

```
$ make clean
$ make
```

Start GUI Mode

```
$ make clean
$ make gui
```

- **Typical Design Flow:**
  1. Enter the GUI mode
  2. Find all synthesizing errors
  3. Analysis the synthesized design (clock constraints, resource usage, pinouts, and etc.)
  4. Exit the GUI
  5. Launch the batch mode
  6. Wait for the batch mode to generate the .bit and/or .mcs file
     - If an error occurs (timing error or placement error), launch the GUI mode (without make clean) to analysis the implementation error(s)
  7. Optional: Once batch mode is complete, launch the GUI mode (without make clean) to review the implementation reports (resources, timing, power estimates, and etc.)

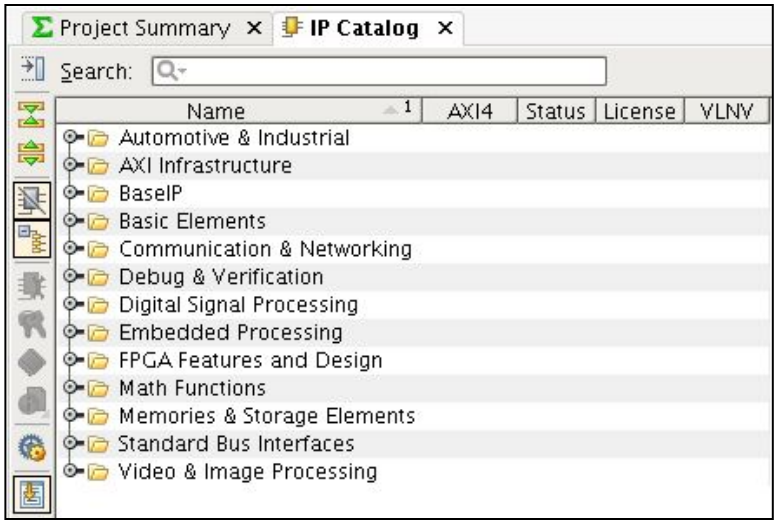
Build System: Simulation

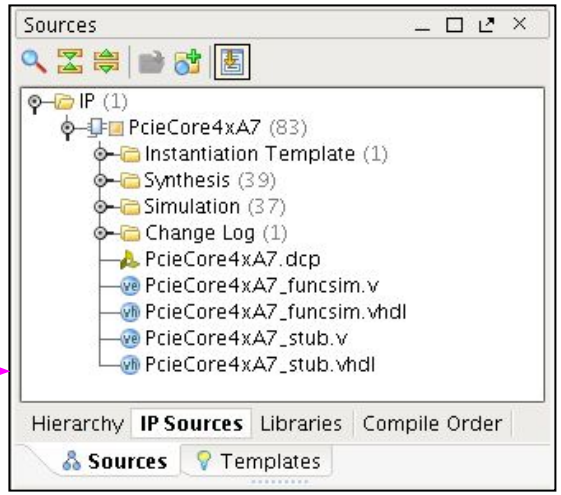
SLAC

- Support two simulation modes:
  - Vivado Native Simulator:
    - Pros:
      - Quickly prototype and simulate firmware
    - Cons:
      - No external software interface
  - Synopsys VCS Simulator:
    - Uses to the Vivado framework to generate the VCS build scripts
    - Pros:
      - Able to attach a software interface via shared memory interface
      - Able to simulate mixed signals (analog + digital)
    - Cons:
      - Some additional setup time compared to Vivado Native simulator

Design Flow: IP Core Generation

- Typical Design Flow:
  1. Open the GUI mode (“make gui”)
  2. Open the “IP Catalog”
  3. Select the IP that you want to configure
  4. Generate the IP core
  5. Close the GUI mode
  6. Copy the IP core’s .xci file from the build/ directory to the module’s Coregen directory
  7. Update the module’s core.txt with the newly add IP core
  8. Remove the obsolete files in the build/ directory with a “make clean” command
  9. Re-open the GUI mode (“make gui”)
  10. Confirm that your new IP got included into the “IP Sources”





Build System: System Generator (Matlab's Simulink Programming)

SLAC

- Supported in the build system
- Build System Tutorial
  - file:///afs/slac/g/reseng/svn/repos/LCLS_Timing_DSP/trunk/firmware/targets/AdcToDacLoopBackTest/README.txt
  - file:///afs/slac/g/reseng/svn/repos/LCLS_Timing_DSP/trunk/firmware/targets/LaserLocker/README.txt
- **Before starting this build system tutorial**, please review the following Xilinx User Guides:
  - UG640: System Generator for DSP: User Guide
  - UG897: Vivado: Designing with System Generator

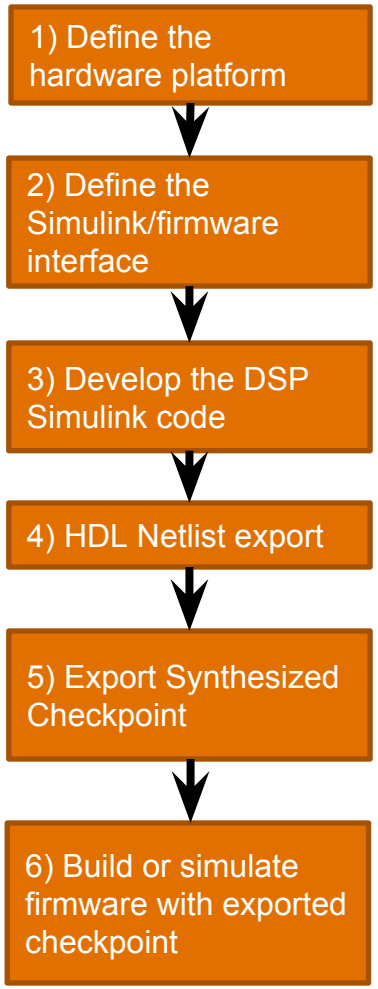
System Generator Homepage

http://www.origin.xilinx.com/products/design-tools/vivado/integration/sysgen.html

Build System: System Generator (Matlab's Simulink Programming)

SLAC

1. Define the hardware platform
   - Development board or custom board?
   - Select ADCs and DACs
     - Many firmware drivers already exist
     - New firmware drivers can be added into the our firmware driver library
   - Digital inputs/outputs?
   - Software interface(s)?
2. Define the Simulink/firmware interface
   - Number of ADC/DAC channels
   - Number of configuration/status registers
   - Digital inputs/outputs (Example: LEDs)
   - Preferably using a standard bus interface (AXI-Lite or AXI-Stream)
3. Develop the DSP Simulink code
   - Create Simulink module with inputs/outputs with the defined firmware interface
   - Include a simulation testbed to verify code in Matlab
4. HDL Netlist export
   - Synthesis and Implement the design in Vivado
   - Check the timing and resource usage
5. Export Synthesized Checkpoint
   - Generates a single Design CheckPoint (.DCP) file
   - Add .dcp file to source list in the module directory's IP core directory
6. Build or simulate project using appropriate target



Design Flow: Vivado HLS (C/C++ programming)

- Vivado High Level Synthesis converts C/C++ code into RTL
- Supported in the build system
- Build System Tutorial (AXI4 Streaming & AXI-Lite):
  - file:///afs/slac/g/reseng/svn/repos/ExampleProject/tags/VivadoHlsExample/trunk/firmware/README.txt
- **Before starting this build system tutorial**, please review the following Xilinx User Guides:
  - UG902: Vivado Design Suite User Guide: High-Level Synthesis
  - UG871: Vivado Design Suite Tutorial: High-Level Synthesis
- Lots of HLS Video Tutorials as well:
  - http://www.xilinx.com/training/vivado/index.htm

Vivado HLS Homepage

http://www.origin.xilinx.com/products/design-tools/vivado/integration/esl-design.html

Design Flow: Partial Configuration

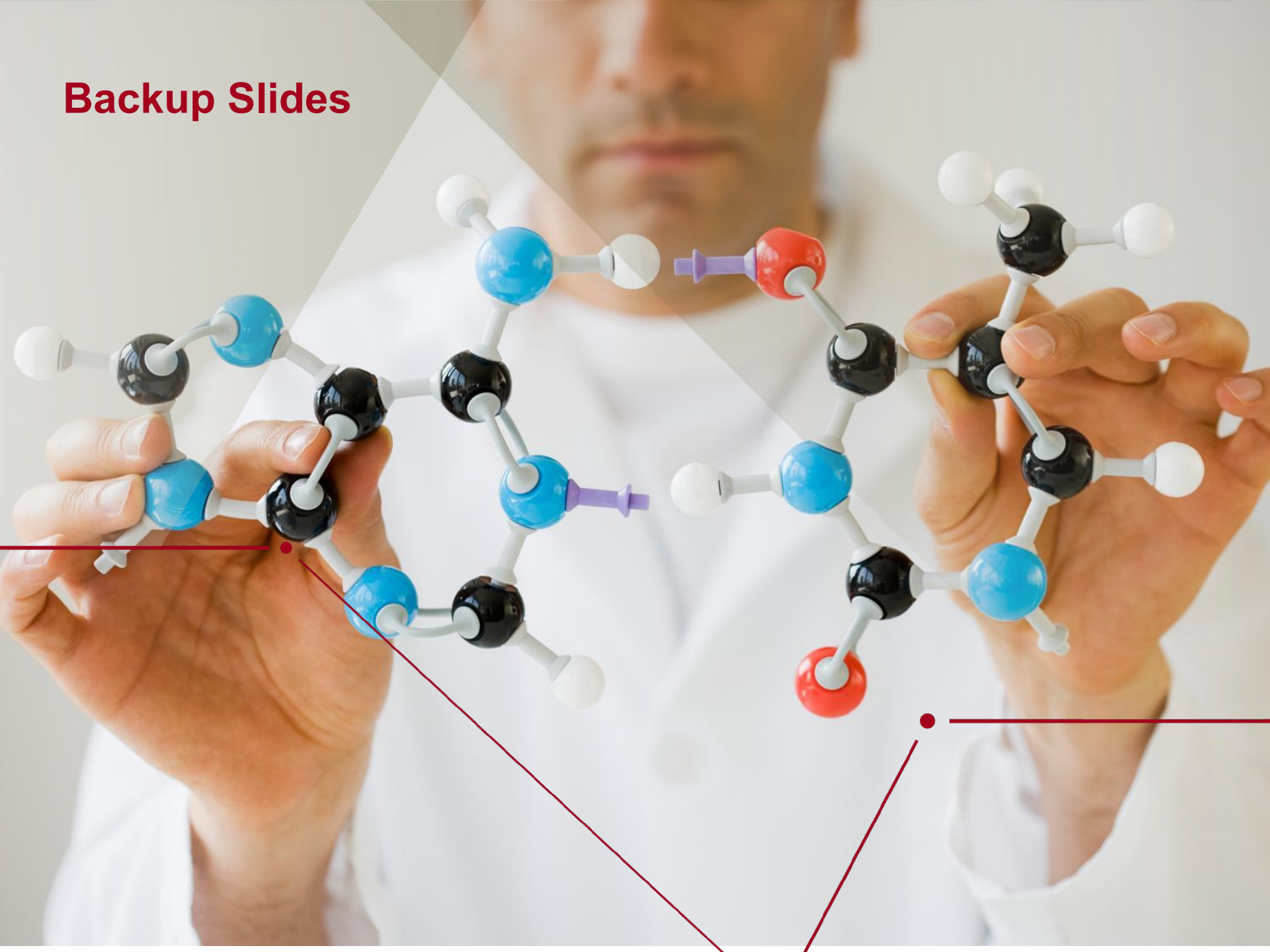
SLAC

- Partial Configuration allows base code to be locked in the FPGA, while the user generate project specific code
- Supported in the build system
- Build System Tutorial:
  - file:///afs/slac/g/reseng/svn/repos/ExampleProject/tags/PartialReconfigurationExample/trunk/firmware/README.txt
  - file:///afs/slac/g/reseng/svn/repos/ExampleProject/tags/PartialReconfigurationExample/trunk/firmware/targets/README.txt
- **Before starting this build system tutorial**, please review the following Xilinx User Guides:
  - UG909: Vivado Design Suite User Guide: Partial Reconfiguration
  - UG947: Vivado Design Suite Tutorial: Partial Reconfiguration

Partial Configuration Homepage

http://www.origin.xilinx.com/products/design-tools/vivado/implementation/partial-reconfiguration.html

Backup Slides



How to Start a New Project

SLAC

```
# Create a new project called “TestProject”
$  cd $home/projects
$  svn export file:///afs/slac/g/reseng/svn/repos/StdLib/trunk/build/svn_create_new_project.sh
$  ./ svn_create_new_project.sh TestProject
```