Apollo Platform Update

D. Gastler *Boston University*
for Apollo Blade Platform Team

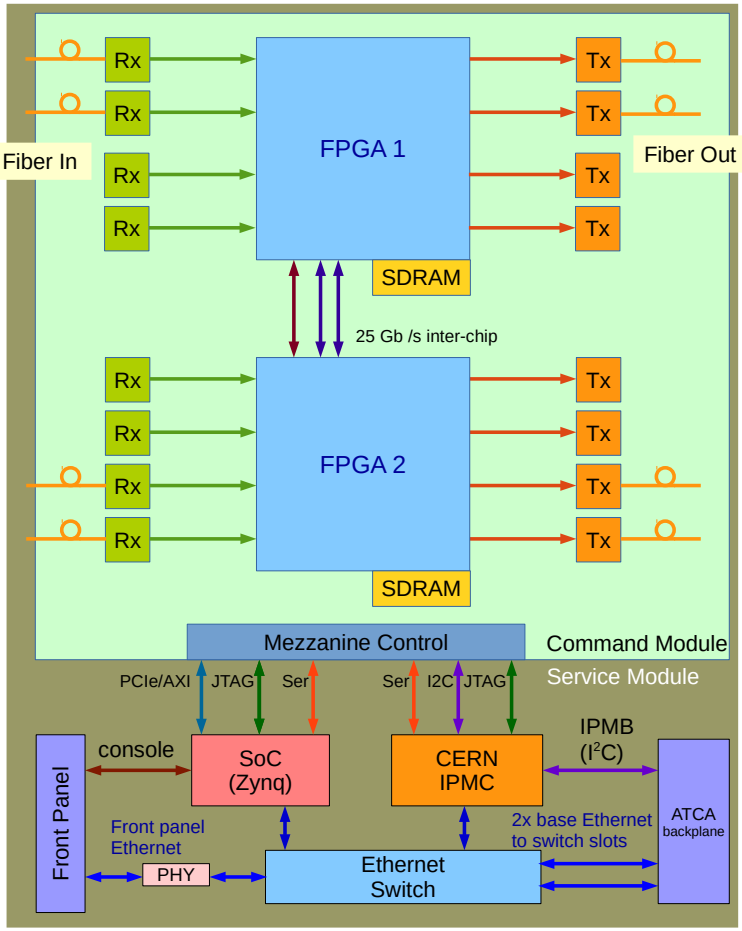
June 8, 2021

Outline:

- Apollo Platform (review)
- Apollo SM+CM Rev2/2a
- Moving to ZynqMP
- FW/SW Interface Updates
- Useful TCL

Apollo Platform (review)

APOLLO Platform Block Diagram

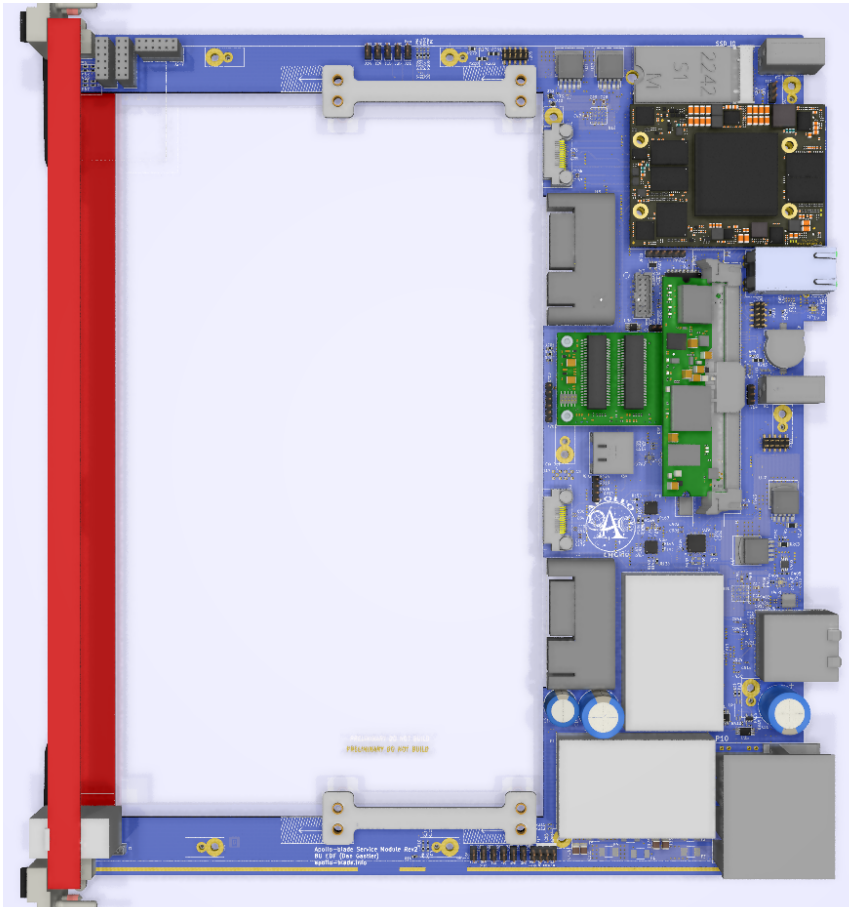


- "Service Module (SM)":
  - 400W power
    (12V to Command Module)
  - Zynq SoC (Enclustra)
    (70xx Rev1; US+ Rev2)
  - IPMC (CERN,UW, or Open)
  - Wisconsin ESM Ethernet Switch
- "Command Module (CM)":
  - Large FPGAs
  - Fiber transceivers
  - MCU control & sensors
  - Two Rev1 designs exist
    Starting Cornell Rev2 design

Will be used in CMS-IT-DTC, CMS-TF, & ATLAS-L0MDT $\approx$ 250 blades

Apollo SM Rev 2/2a

APOLLO Service Module (SM) Rev 2/2a



- Bug Fixes:
  - Power-on sequence
  - Zone3 Ethernet speed
  - JTAG muxing (now a CPLD)
- Improvements:
  - New Enclustra SoC options
    XU8 and other US+ Zynqs
    Still compatible with ZX1
  - M.2 2242 SSD (optional)
  - Power monitoring
  - Improved cover mechanics
  - More GPIOs (CM,SM,IPMC,ext)
- Status:
  - Two Rev2s built
  - CMS TCDS2 oversight fixed in Rev2a
  - One more fix on Rev2 before Rev2a order

APOLLO Service Module (SM) Rev 2/2a

- Bug Fixes:
  - Power-on sequence
  - Zone3 Ethernet speed
  - JTAG muxing (now a CPLD)
- Improvements:
  - New Enclustra SoC options
    XU8 and other US+ Zynqs
    Still compatible with ZX1
  - M.2 2242 SSD (optional)
  - Power monitoring
  - Improved cover mechanics
  - More GPIOs (CM,SM,IPMC,ext)
- Status:
  - Two Rev2s built
  - CMS TCDS2 oversight fixed in Rev2a
  - One more fix on Rev2 before Rev2a order

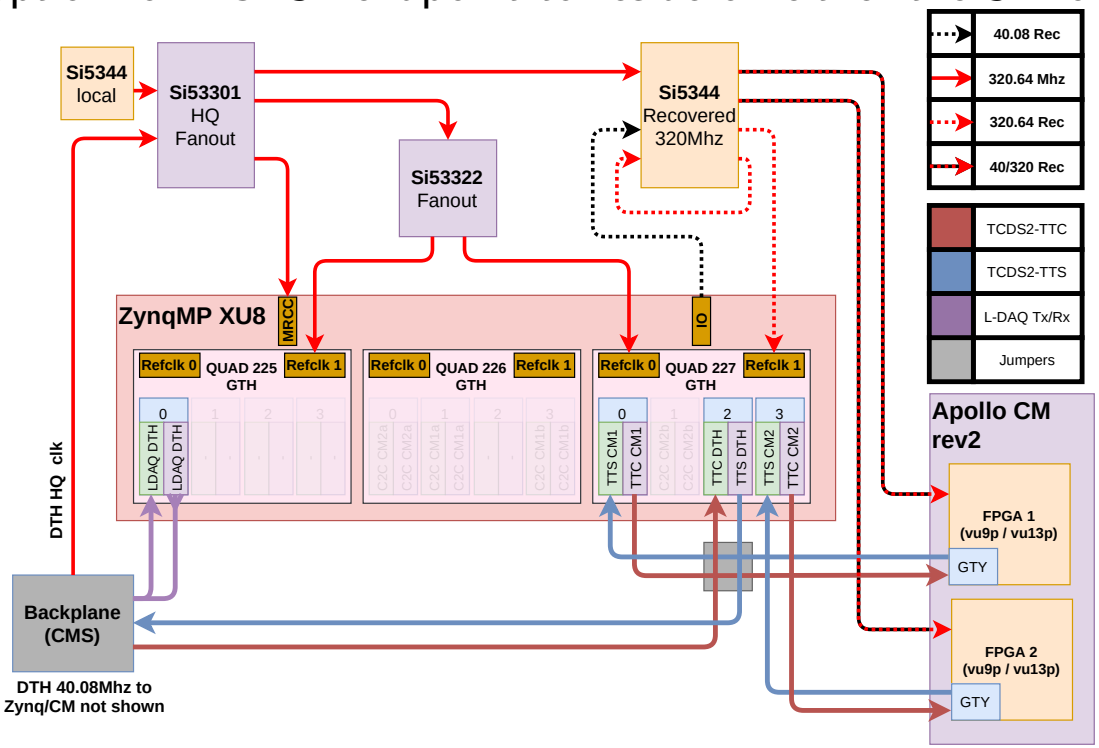
APOLLO SM 2a

Fixes/Upgrades:

- Changing to TerraGreen material from Nelco-4000 for Halogen-free requirements
- Additional SD-card slot to help with SD-card problems in Rev-2
- Misc. fixes

Upgrades for CMS TCDS2

- Re-arranged transceivers for TCDS all into the same quad
- Additional Si5344 for TCLink clock cleaning
- Remove spurious silicon (fanouts, crosspoint switches) from signal path
- Provide option for TCDS2 endpoint to reside on either the SM or CM



Moving to ZynqMP

Rev2 with ZynqMP SoC

- FW/SW updates
  - FW update was relatively easy due to Enclustra example design.
  - Small differences complicated our tcl infrastructure, but generalized FW easy to make
  - Similar small changes in device-tree/kernel cause similar SW generalization.
  - Booting and device-tree required more manipulation in USP than in 7-Series
- Bring-up Issues
  - In ZynqMP, SDCard interface and Ethernet interface didn't work.
  - When swapping in Zynq-7 SoC, SDCard and Ethernet worked.
  - Huge slowdown in bring-up. Used EMMC and UART file transfers...
  - Everything magically worked in Vivado/petalinux 2020.2! Why???

Building Many Related Firmwares

- The Apollo platform is used by different groups
- Users have a mix of Rev1, Rev2, and Rev2a(soon)
- Solved via shared and separate source directories in the repos
- Common FW and cores are in ./src ./cores
- Revision specific FW and cores are in ./configure/RevID/
- FPGA settings, file list, axi config in ./configure/RevID
- Kernel options for each build are in kernel/configs/RevID
- Makefile rules can build any configuration

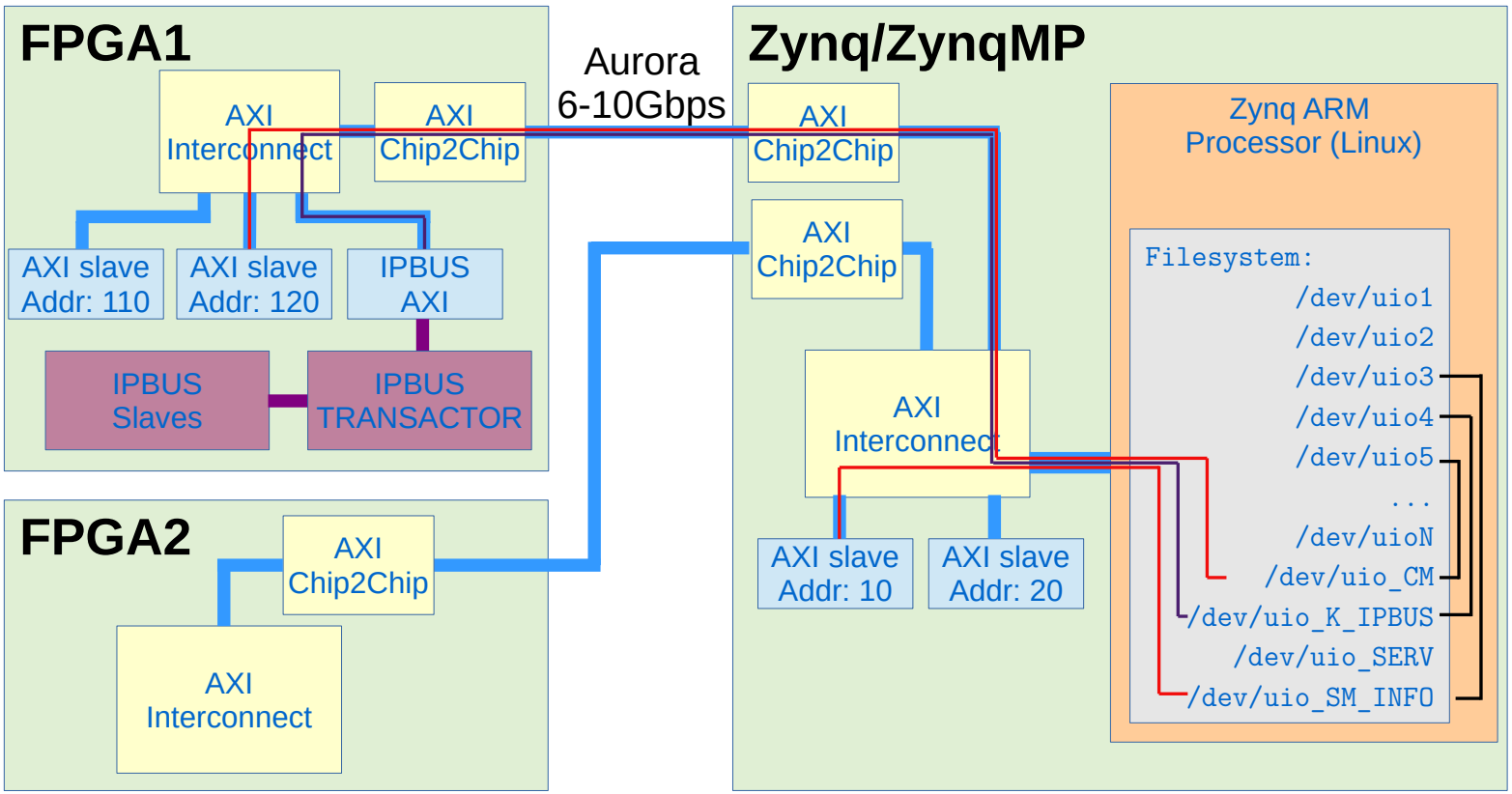
```
configs
├── rev1_xc7z035
├── rev1_xc7z045
├── rev2a_xczu7ev
├── rev2_xc7z035
├── rev2_xc7z045
├── rev2_xczu7ev
│   ├── files.tcl
│   ├── settings.tcl
│   ├── slaves.yaml
│   ├── top.vhd
│   └── top.xdc
└── rev2_xczu7ev_testing
kernel
└── configs
    ├── rev1_xc7z035
    ├── rev1_xc7z045 -> rev1_xc7z035
    ├── rev2_xc7z035
    ├── rev2_xc7z045 -> rev2_xc7z035
    ├── rev2_xczu7ev
    │   ├── ATF
    │   ├── configs
    │   ├── device-tree
    │   ├── fsbl
    │   ├── hw_user
    │   └── kernel
    ├── rev2_xczu7ev_qspi
    └── rev2_xczu7ev_testing
```

```
[dan@tesla SM_ZYNQ_FW]$ make list

Apollo SM config:
rev1_xc7z035            rev2_xc7z035            rev2_xczu7ev
rev1_xc7z045            rev2_xc7z045            rev2_xczu7ev_testing
```

FW/SW Interface Update

Apollo μHAL & C2C (Physical/Firmware)



- Local and remote AXI slaves (remote via Xilinx Chip2Chip)
- Slaves memory-mapped using userspace UIO driver
- Modified μHAL for AXI access

AXI Endpoint Generation

YAML Slave file

```
AXI_CONTROL_SETS:
  AXI_MASTER_CTRL:
      axi_interconnect: "${::AXI_INTERCONNECT_NAME}"
      axi_clk: "${::AXI_MASTER_CLK}"
      axi_rstn: "${::AXI_MASTER_RSTN}"
      axi_freq: "${::AXI_MASTER_CLK_FREQ}"
  AXI_C2C_CTRL:
      axi_interconnect: "${::AXI_C2C_INTERCONNECT_NAME}"
      axi_clk: "${::AXI_MASTER_CLK}"
      axi_rstn: "${::AXI_MASTER_RSTN}"
      axi_freq: "${::AXI_MASTER_CLK_FREQ}"

AXI_SLAVES:
  C2C:
    TCL_CALL:
```

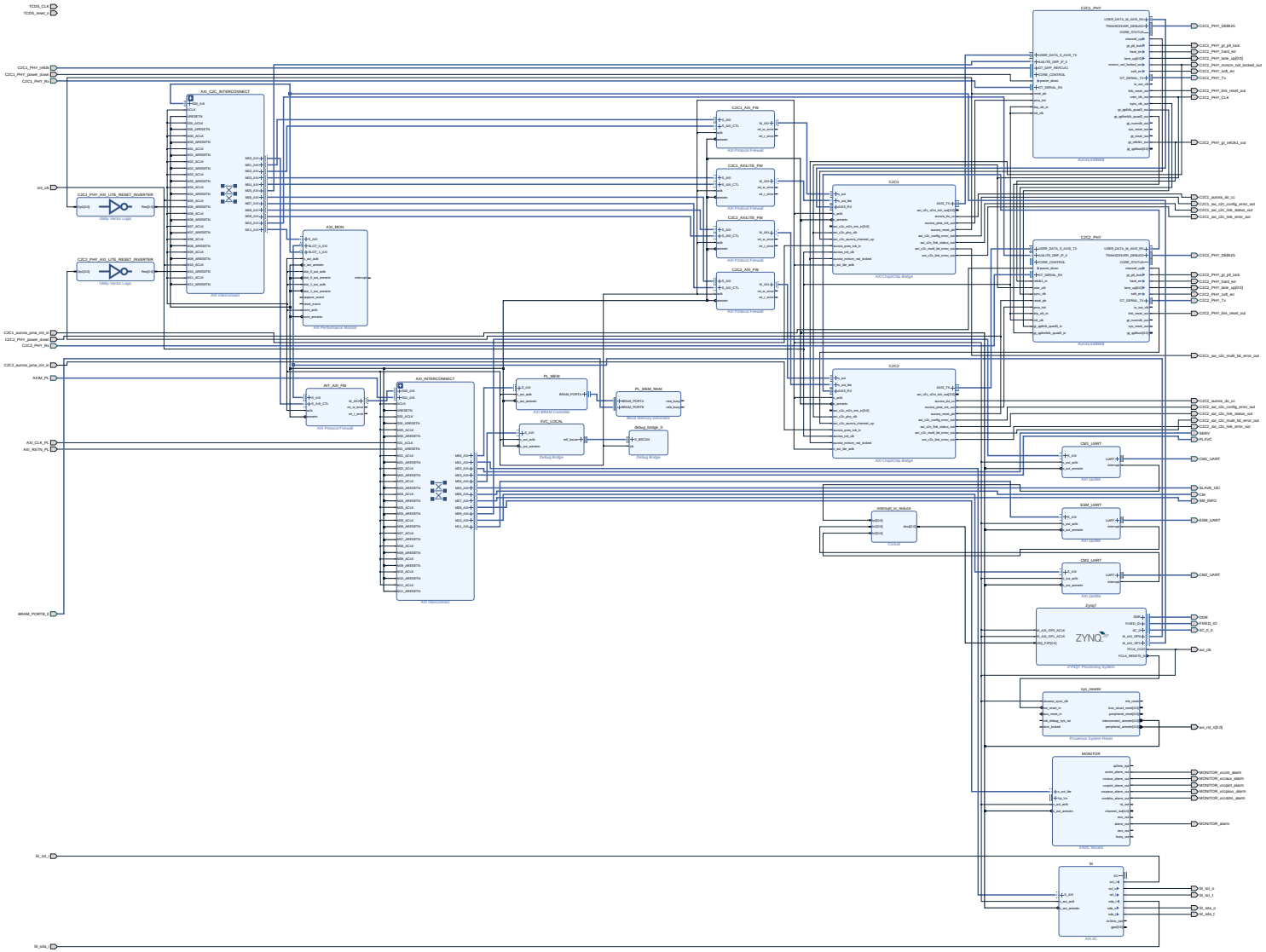
```
  SM_INFO:
    TCL_CALL:
      command: "AXI_PL_DEV_CONNECT"
      axi_control: "${::AXI_MASTER_CTRL}"
      addr:
        offset: -1
        range: "8K"
    XML:
      - "address_table/modules/FW_INFO.xml"
    UHAL_BASE: 0x0A000000
    HDL:
      out_dir: "src/SM_info"
      map_template: "axi_generic/template_map_withbram.vhd"

  MONITOR:
    TCL_CALL:
      command: "AXI_IP_SYS_MGMT"
      axi_control: "${::AXI_MASTER_CTRL}"
      enable_i2c_pins: 0
      addr:
        offset: -1
        range: "64K"
    XML:
      - "address_table/modules/MONITOR_USP.xml"
    UHAL_BASE: 0x0B000000
```

- AXI_CONTROL_SETS
  - AXI interconnects to connect slaves to
  - Stores name and associated clk/resets
  - AXI slaves auto expand these interconnects
- AXI_SLAVES:
  - Each node is one AXI slave
  - TCL_CALL:
    tcl call to make for building slave
    AXI connection and addressing info to use
    Additional arguments for specific slave
  - XML:
    list of μHAL XML files for this slave
    First listed is the top for this slave
  - HDL:
    Controls automatic AXI register map decoder
    and package of records generation

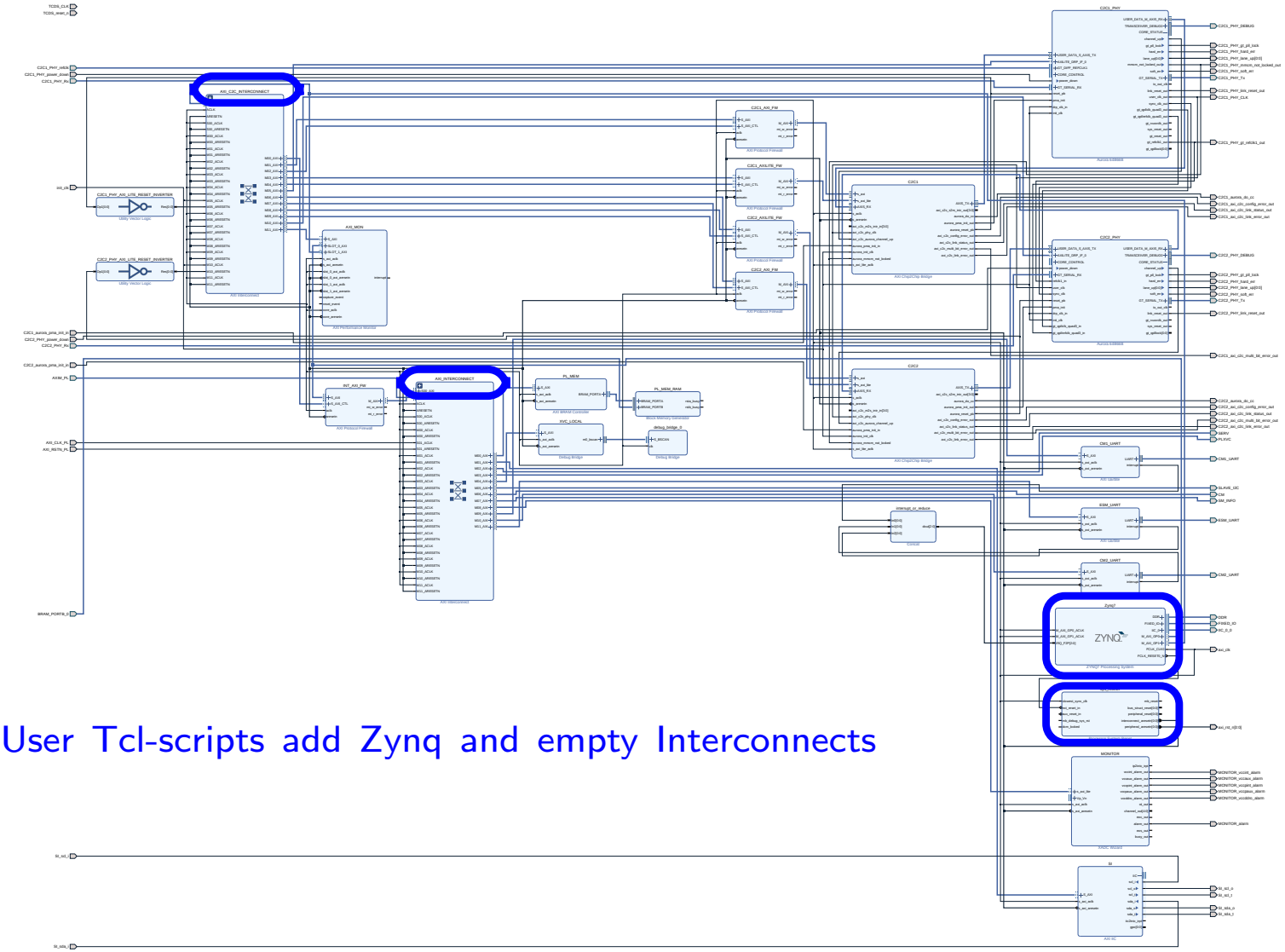
AXI Endpoint Generation

YAML Slave Generated Block Design



AXI Endpoint Generation

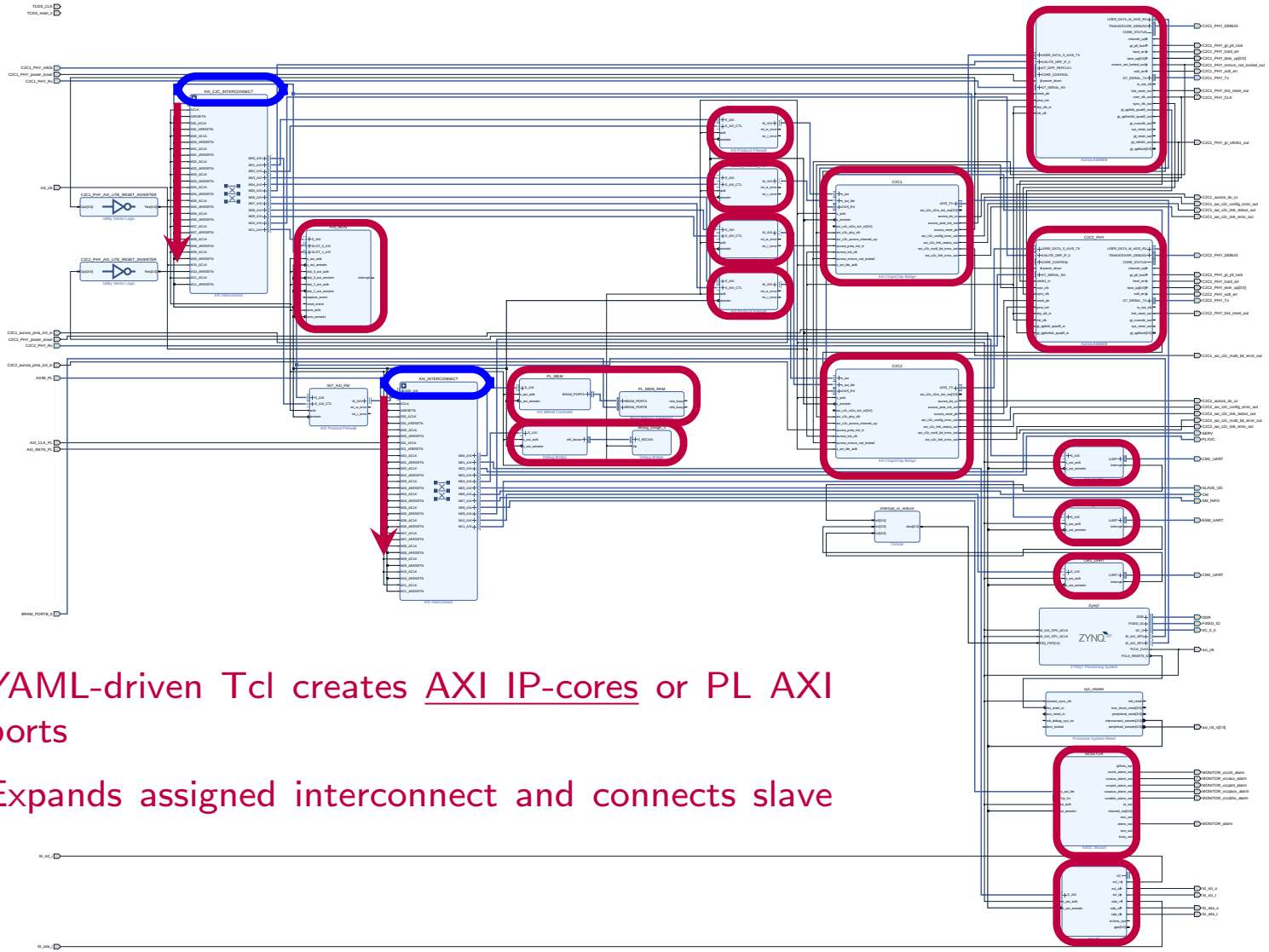
YAML Slave Generated Block Design



User Tcl-scripts add Zynq and empty Interconnects

AXI Endpoint Generation

YAML Slave Generated Block Design

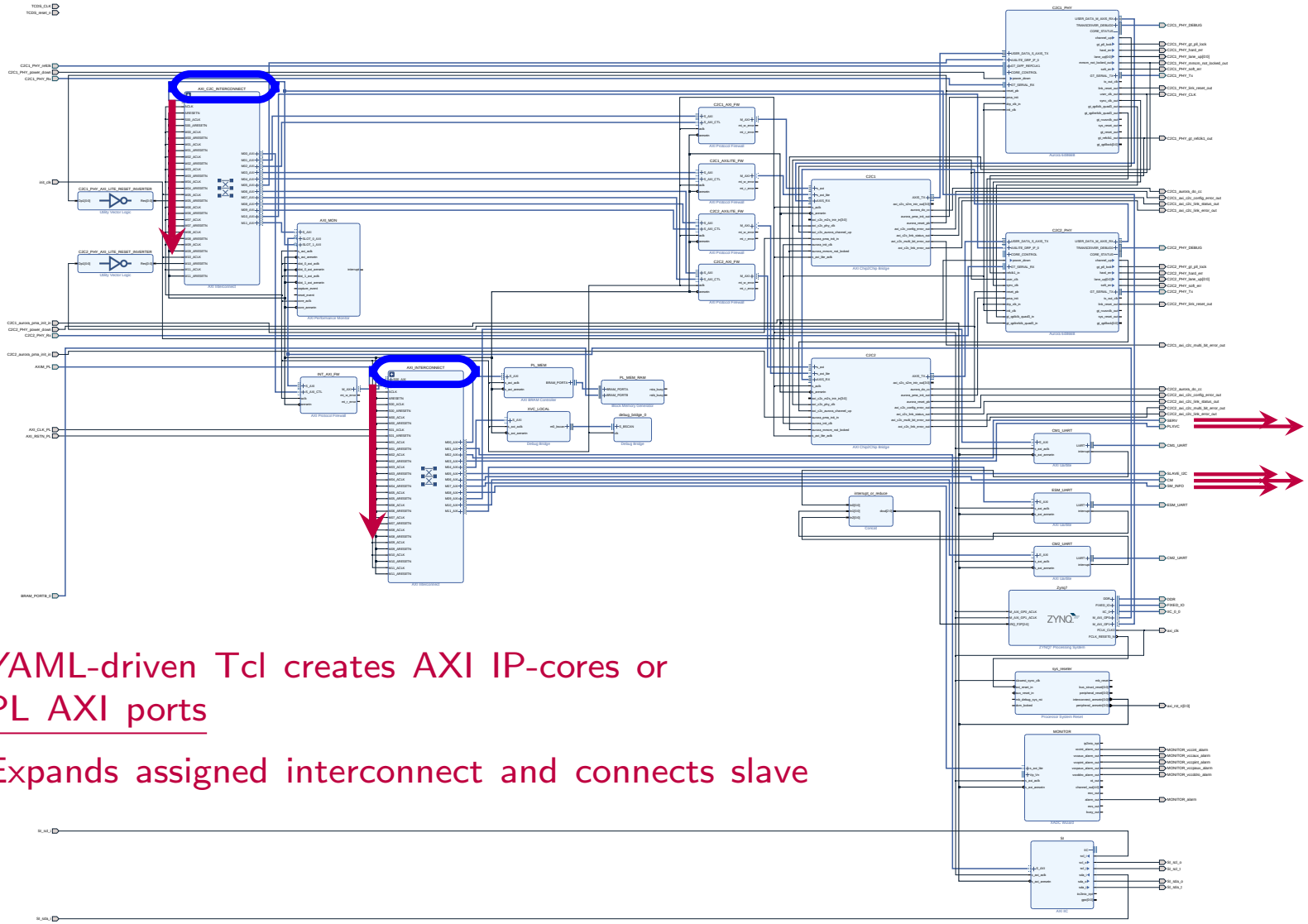


YAML-driven Tcl creates AXI IP-cores or PL AXI ports

Expands assigned interconnect and connects slave

AXI Endpoint Generation

YAML Slave Generated Block Design

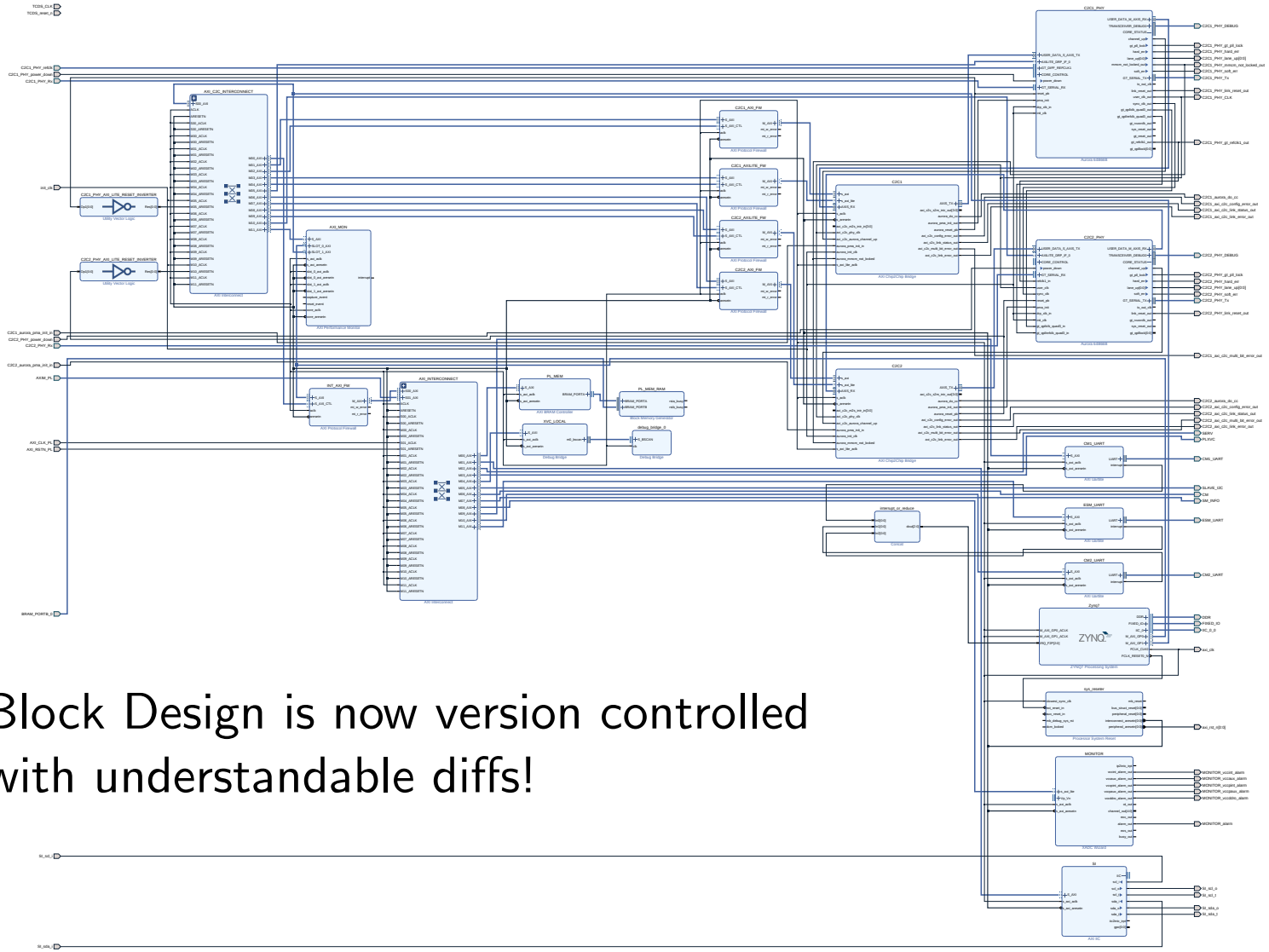


YAML-driven Tcl creates AXI IP-cores or PL AXI ports

Expands assigned interconnect and connects slave

AXI Endpoint Generation

YAML Slave Generated Block Design



Block Design is now version controlled with understandable diffs!

AXI Endpoint Generation

AXI Address VHDL & C files

```c
#ifndef __AXI_ADDR_MAP__
#define __AXI_ADDR_MAP__
#define AXI_ADDR_INT_AXI_FW 0xB0000000
#define AXI_ADDR_C2C1_AXI_FW 0xB0010000
#define AXI_ADDR_C2C1_AXILITE_FW 0xB0020000
#define AXI_ADDR_C2C1_PHY 0xB0030000
#define AXI_ADDR_C2C2_AXI_FW 0xB0040000
#define AXI_ADDR_C2C2_AXILITE_FW 0xB0050000
#define AXI_ADDR_C2C2_PHY 0xB0060000
#define AXI_ADDR_XVC_LOCAL 0xA00C0000
#define AXI_ADDR_PL_MEM 0xA0000000
#define AXI_ADDR_SI 0xA0010000
#define AXI_ADDR_SERV 0xA0020000
#define AXI_ADDR_PLXVC 0xA0030000
#define AXI_ADDR_SLAVE_I2C 0xA0040000
#define AXI_ADDR_CM 0xA0050000
#define AXI_ADDR_SM_INFO 0xA0060000
#define AXI_ADDR_MONITOR 0xA0070000
#define AXI_ADDR_AXI_MON 0xB0070000
#define AXI_ADDR_MEM_TEST 0xA00B0000
// ranges
#define AXI_RANGE_INT_AXI_FW 0x10000
#define AXI_RANGE_C2C1_AXI_FW 0x10000
#define AXI_RANGE_C2C1_AXILITE_FW 0x10000
#define AXI_RANGE_C2C1_PHY 0x10000
#define AXI_RANGE_C2C2_AXI_FW 0x10000
#define AXI_RANGE_C2C2_AXILITE_FW 0x10000
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package AXISlaveAddrPkg is
constant AXI_ADDR_INT_AXI_FW : unsigned(31 downto 0) := x"B0000000";
constant AXI_ADDR_C2C1_AXI_FW : unsigned(31 downto 0) := x"B0010000";
constant AXI_ADDR_C2C1_AXILITE_FW : unsigned(31 downto 0) := x"B0020000";
constant AXI_ADDR_C2C1_PHY : unsigned(31 downto 0) := x"B0030000";
constant AXI_ADDR_C2C2_AXI_FW : unsigned(31 downto 0) := x"B0040000";
constant AXI_ADDR_C2C2_AXILITE_FW : unsigned(31 downto 0) := x"B0050000";
constant AXI_ADDR_C2C2_PHY : unsigned(31 downto 0) := x"B0060000";
constant AXI_ADDR_XVC_LOCAL : unsigned(31 downto 0) := x"A00C0000";
constant AXI_ADDR_PL_MEM : unsigned(31 downto 0) := x"A0000000";
constant AXI_ADDR_SI : unsigned(31 downto 0) := x"A0010000";
constant AXI_ADDR_SERV : unsigned(31 downto 0) := x"A0020000";
constant AXI_ADDR_PLXVC : unsigned(31 downto 0) := x"A0030000";
constant AXI_ADDR_SLAVE_I2C : unsigned(31 downto 0) := x"A0040000";
constant AXI_ADDR_CM : unsigned(31 downto 0) := x"A0050000";
constant AXI_ADDR_SM_INFO : unsigned(31 downto 0) := x"A0060000";
constant AXI_ADDR_MONITOR : unsigned(31 downto 0) := x"A0070000";
constant AXI_ADDR_AXI_MON : unsigned(31 downto 0) := x"B0070000";
constant AXI_ADDR_MEM_TEST : unsigned(31 downto 0) := x"A00B0000";
-- ranges
constant AXI_RANGE_INT_AXI_FW : unsigned(31 downto 0) :=  x"10000";
```

- Files auto-generated when YAML file is processed
- Handles both YAML fixed and Vivado auto-assigned AXI addresses
- C header(left) used for FSBL modifications & future Cortex-R5F core usage
- VHDL Package used for PL based AXI Masters to know slave addresses.

Skipping device-tree DTSI_chunk/post_chunk file creation (in backup slides)
Question: Is there a way to restrict the range of the Xilinx auto-assiged addresses

AXI Endpoint Generation

Generating HDL

- μHAL XML is used to generate VHDL records and VHDL AXI $\Leftrightarrow$ record decoder
- Records have the same hierarchy as the XML address table
- Mon (reads) & Ctrl (writes) are the top level records

User input XML: CM_USP.xml

```
<node id="CM">
  <node id="CM_1" address="0x00"  fwinfo="type=array" module="file://CM_sing
  <node id="CM_2" address="0x100" fwinfo="type=array" module="file://CM_sing
</node>
```

Auto-generated VHDL records:

```
type CM_MON_t is record
  CM                              :CM_CM_MON_t_ARRAY;
end record CM_MON_t;

type CM_CM_MON_t is record
  CTRL                            :CM_CM_CTRL_MON_t;
  C2C                             :CM_CM_C2C_MON_t_ARRAY;
  MONITOR                         :CM_CM_MONITOR_MON_t;
end record CM_CM_MON_t;
type CM_CM_MON_t_ARRAY is array(1 to 2) of CM_CM_MON_t;
```

AXI Endpoint Generation

Generating HDL

- *μHAL* XML is used to generate VHDL records and VHDL AXI ⇔ record decoder
- Records have the same hierarchy as the XML address table
- Mon (reads) & Ctrl (writes) are the top level records

User input XML: CM_USP.xml

```
<node id="CM">
  <node id="CM_1" address="0x00"  fwinfo="type=array" module="file://CM_sing
  <node id="CM_2" address="0x100" fwinfo="type=array" module="file://CM_sing
</node>
```

Auto-generated VHDL records:

```
type CM_MON_t is record
  CM                              :CM_CM_MON_t_ARRAY;
end record CM_MON_t;

type CM_CM_MON_t is record
  CTRL                            :CM_CM_CTRL_MON_t;
  C2C                             :CM_CM_C2C_MON_t_ARRAY;
  MONITOR                         :CM_CM_MONITOR_MON_t;
end record CM_CM_MON_t;
type CM_CM_MON_t_ARRAY is array(1 to 2) of CM_CM_MON_t;
```

AXI Endpoint Generation

Generating HDL

- *μHAL* XML is used to generate VHDL records and VHDL AXI ⇔ record decoder
- Records have the same hierarchy as the XML address table
- Mon (reads) & Ctrl (writes) are the top level records

User input XML: CM_USP.xml → CM_single_USP.xml

```
<node id="CM">
  <!-- CTRL 0x0 -> 0x3 mapped to 0x00 -> 0x03 -->
 <node id="CTRL"         address="0x00"  module="file://CM_CTRL.xml"/>
  <!-- CTRL 0x0 -> 0x19 mapped to 0x10 -> 0x29 -->
  <node id="C2C_1"       fwinfo="type=array" address="0x10"  module="file://CM_C2
  <!-- CTRL 0x0 -> 0x19 mapped to 0x30 -> 0x49 -->
  <node id="C2C_2"       fwinfo="type=array" address="0x30"  module="file://CM_C2
  <!-- MONITOR 0x0 -> 0xA mapped to 0x50 -> 0x5A -->
  <node id="MONITOR"     address="0x50"  module="file://CM_Mon.xml"/>
</node>
```

Auto-generated VHDL records:

```
type CM_CM_MON_t is record
  CTRL                          :CM_CM_CTRL_MON_t;
  C2C                           :CM_CM_C2C_MON_t_ARRAY;
  MONITOR                       :CM_CM_MONITOR_MON_t;
end record CM_CM_MON_t;
type CM_CM_MON_t_ARRAY is array(1 to 2) of CM_CM_MON_t;
```

AXI Endpoint Generation

Generating HDL

- *μHAL* XML is used to generate VHDL records and VHDL AXI ⇔ record decoder
- Records have the same hierarchy as the XML address table
- Mon (reads) & Ctrl (writes) are the top level records

User input XML: CM_USP.xml → CM_single_USP.xml

```
<node id="CM">
  <!-- CTRL 0x0 -> 0x3 mapped to 0x00 -> 0x03 -->
 <node id="CTRL"         address="0x00"  module="file://CM_CTRL.xml"/>
  <!-- CTRL 0x0 -> 0x19 mapped to 0x10 -> 0x29 -->
  <node id="C2C_1"       fwinfo="type=array" address="0x10"  module="file://CM_C2
  <!-- CTRL 0x0 -> 0x19 mapped to 0x30 -> 0x49 -->
  <node id="C2C_2"       fwinfo="type=array" address="0x30"  module="file://CM_C2
  <!-- MONITOR 0x0 -> 0xA mapped to 0x50 -> 0x5A -->
  <node id="MONITOR"     address="0x50"  module="file://CM_Mon.xml"/>
</node>
```

Auto-generated VHDL records:

```
type CM_CM_MON_t is record
  CTRL                          :CM_CM_CTRL_MON_t;
  C2C                           :CM_CM_C2C_MON_t_ARRAY;
  MONITOR                       :CM_CM_MONITOR_MON_t;
end record CM_CM_MON_t;
type CM_CM_MON_t_ARRAY is array(1 to 2) of CM_CM_MON_t;
```

AXI Endpoint Generation

Generating HDL

- *μHAL* XML is used to generate VHDL records and VHDL AXI ⇔ record decoder
- Records have the same hierarchy as the XML address table
- Mon (reads) & Ctrl (writes) are the top level records

User input XML: CM_USP.xml → CM_single_USP.xml → CM_C2C_USP.xml

```
<node id="C2C">
  <node id="ENABLE_PHY_CTRL"                mask="0x0800"     permission="rw" descri
  <node id="PHY_LANE_STABLE"   address="0x1" mask="0xFFFFFFFF" permission="rw" descri
  <node id="PHY_READ_TIME"     address="0x2" mask="0xFFFFFF"   permission="rw" descri

  <!-- STATUS 0x0 -> 0x0 mapped to 0x04 -> 0x04 -->
  <node id="STATUS"                 address="0x4" module="file://CM_C2C_Status.xml"/>

  <!-- CTRL 0x0 -> 0x2 mapped to 0x05 -> 0x07-->
  <node id="LINK_DEBUG"             address="0x5" module="file://CM_C2C_DEBUG_USP.xml

  <!-- CTRL 0x0 -> 0x9 mapped to 0x10 -> 0x19 -->
  <node id="CNT"                    address="0x10" module="file://CM_C2C_CNT.xml"/>
```

Auto-generated VHDL records:

```
type CM_CM_C2C_MON_t is record
  STATUS                    :CM_CM_C2C_STATUS_MON_t;
  LINK_DEBUG                :CM_CM_C2C_LINK_DEBUG_MON_t;
  CNT                       :CM_CM_C2C_CNT_MON_t;
end record CM_CM_C2C_MON_t;
type CM_CM_C2C_MON_t_ARRAY is array(1 to 2) of CM_CM_C2C_MON_
```

AXI Endpoint Generation

Generating HDL

- *μHAL* XML is used to generate VHDL records and VHDL AXI ⇔ record decoder
- Records have the same hierarchy as the XML address table
- Mon (reads) & Ctrl (writes) are the top level records

User input XML: CM_USP.xml → CM_single_USP.xml → CM_C2C_USP.xml

```
<node id="C2C">
  <node id="ENABLE_PHY_CTRL"                mask="0x0800"     permission="rw" descri
  <node id="PHY_LANE_STABLE"   address="0x1" mask="0xFFFFFFFF" permission="rw" descri
  <node id="PHY_READ_TIME"     address="0x2" mask="0xFFFFFF"   permission="rw" descri

  <!-- STATUS 0x0 -> 0x0 mapped to 0x04 -> 0x04 -->
  <node id="STATUS"                  address="0x4" module="file://CM_C2C_Status.xml"/>

  <!-- CTRL 0x0 -> 0x2 mapped to 0x05 -> 0x07-->
  <node id="LINK_DEBUG"              address="0x5" module="file://CM_C2C_DEBUG_USP.xml

  <!-- CTRL 0x0 -> 0x9 mapped to 0x10 -> 0x19 -->
  <node id="CNT"                     address="0x10" module="file://CM_C2C_CNT.xml"/>
```

?

rw-registers fed back in decoder so not in Mon records

Auto-generated VHDL records:

```
type CM_CM_C2C_MON_t is record
  STATUS                    :CM_CM_C2C_STATUS_MON_t;
  LINK_DEBUG                :CM_CM_C2C_LINK_DEBUG_MON_t;
  CNT                       :CM_CM_C2C_CNT_MON_t;
end record CM_CM_C2C_MON_t;
type CM_CM_C2C_MON_t_ARRAY is array(1 to 2) of CM_CM_C2C_MON
```

AXI Endpoint Generation

Generating HDL

- μHAL XML is used to generate VHDL records and VHDL AXI $\Leftrightarrow$ record decoder
- Records have the same hierarchy as the XML address table
- Mon (reads) & Ctrl (writes) are the top level records

User input XML: CM_USP.xml $\rightarrow$ CM_single_USP.xml $\rightarrow$ CM_C2C_USP.xml $\rightarrow$ CM_CNT.xml

```
<node id="CNT">
  <node id="INIT_ALLTIME"          address="0x0" permission="r"  description="C
  <node id="INIT_SHORTTERM"        address="0x1" permission="r"  description="C
  <node id="CONFIG_ERROR_COUNT"    address="0x2" permission="r"  description="C
  <node id="LINK_ERROR_COUNT"      address="0x3" permission="r"  description="C
  <node id="MB_ERROR_COUNT"        address="0x4" permission="r"  description="C
  <node id="PHY_HARD_ERROR_COUNT"  address="0x5" permission="r"  description="C
  <node id="PHY_SOFT_ERROR_COUNT"  address="0x6" permission="r"  description="C
  <node id="PHYLANE_STATE"         address="0x7" permission="r"  description="C
  <node id="RESET_COUNTERS"        address="0x8" permission="w" description="Re
  <node id="PHY_ERRORSTATE_COUNT"  address="0x9" permission="r"  description="C
  <node id="USER_CLK_FREQ"         address="0xA" permission="r"  description="F
```

Made it to read registers.

Matching XML names

helps reading/writing code

Auto-generated VHDL records:

```
type CM_CM_C2C_CNT_MON_t is record
  INIT_ALLTIME              :std_logic_vector(31 downto 0);
  INIT_SHORTTERM            :std_logic_vector(31 downto 0);
  CONFIG_ERROR_COUNT        :std_logic_vector(31 downto 0);
  LINK_ERROR_COUNT          :std_logic_vector(31 downto 0);
  MB_ERROR_COUNT            :std_logic_vector(31 downto 0);
  PHY_HARD_ERROR_COUNT      :std_logic_vector(31 downto 0);
  PHY_SOFT_ERROR_COUNT      :std_logic_vector(31 downto 0);
  PHYLANE_STATE             :std_logic_vector( 2 downto 0);
  PHY_ERRORSTATE_COUNT      :std_logic_vector(31 downto 0);
```

AXI Endpoint Generation

Using Generated HDL

```
--For AXI
CM_interface_1: entity work.CM_interface
  port map (
    clk_axi          => clk_axi,
    reset_axi_n      => reset_axi_n,
    slave_readMOSI   => slave_readMOSI,
    slave_readMISO   => slave_readMISO,
    slave_writeMOSI  => slave_writeMOSI,
    slave_writeMISO  => slave_writeMISO,
    Mon              => Mon,
    Ctrl             => Ctrl);
```

- Add python auto-generated decoder module (left)
- Use Mon & Ctrl records to connect signals (below)
- Child records can be sent to modules to simplify interfaces
- Showing AXI decoder, can generate wishbone

```
-------------------------------------------------------------------------------
-- Phy_lane_control
-------------------------------------------------------------------------------
Phy_lane_control_X: entity work.CM_phy_lane_control
  generic map (
    CLKFREQ          => CLKFREQ,
    DATA_WIDTH       => DATA_WIDTH,
    ERROR_WAIT_TIME  => ERROR_WAIT_TIME)
  port map (
    clk              => clk_axi,
    reset            => reset,
    reset_counter    => CTRL.CM(iCM).C2C(iLane).CNT.RESET_COUNTERS,
    enable           => phycontrol_en(linkID),
    phy_lane_up      => CM_C2C_Mon.Link(linkID).status.phy_lane_up(0),
    phy_lane_stable  => CTRL.CM(iCM).C2C(iLane).PHY_LANE_STABLE,
    READ_TIME        => CTRL.CM(iCM).C2C(iLane).PHY_READ_TIME,
    initialize_out   => aurora_init_buf(linkID),
    lock             => phylanelock(linkID),
    state_out        => Mon.CM(iCM).C2C(iLane).CNT.PHYLANE_STATE,
    count_error_wait => Mon.CM(iCM).C2C(iLane).CNT.PHY_ERRORSTATE_COUNT
```

AXI Endpoint Generation

Interface Embedding

- It would be convenient to embed other interfaces in our decoder
- This is now possible
  right: BRAM example
- Specify a different decoder template (template_map_with_bram.vhd)
- Mark as a memory in XML
  Data width set in fw_info tag
  Depth in size tag
- Now BRAMs and signals can be delivered as a group
- Access via normal memory mapped interface

```
blockram: entity work.rams_sp_wf
  generic map (
    RAM_WIDTH => 13,
    RAM_DEPTH => 256)
  port map (
    clk    => Ctrl.MEM1.clk,
    we     => Ctrl.MEM1.wr_enable,
    en     => Ctrl.MEM1.enable,
    addr   => Ctrl.MEM1.address,
    di     => Ctrl.MEM1.wr_data,
    do     => Mon.MEM1.rd_data,
    do_valid => Mon.MEM1.rd_data_valid);

other_blockram: entity work.rams_sp_wf
  generic map (
    RAM_WIDTH => 13,
    RAM_DEPTH => 256)
  port map (
    clk    => Ctrl.LEVEL_TEST.MEM.clk,
    en     => Ctrl.LEVEL_TEST.MEM.enable,
    we     => Ctrl.LEVEL_TEST.MEM.wr_enable,
    addr   => Ctrl.LEVEL_TEST.MEM.address,
    di     => Ctrl.LEVEL_TEST.MEM.wr_data,
    do     => Mon.LEVEL_TEST.MEM.rd_data,
    do_valid => Mon.LEVEL_TEST.MEM.rd_data_valid);
```

```
<node id="THING"  address="0x20" permission="rw"/>
<node id="MEM1" address="0x100" mode="incremental" size="0x100" permission="rw" fwinfo="type=mem13"/>

<node id="LEVEL_TEST" address="0x300">
  <node id="THING"  address="0x0" permission="rw"/>
  <node id="MEM" address="0x100" mode="incremental" size="0x100" permission="rw" fwinfo="type=mem13"/>
</node>
```

AXI Endpoint Generation

Interface Embedding

- It would be convenient to embed other interfaces in our decoder
- This is now possible
  right: BRAM example
- Specify a different decoder template (template_map_with_bram.vhd)
- Mark as a memory in XML
  Data width set in fw_info tag
  Depth in size tag
- Now BRAMs and signals can be delivered as a group
- Access via normal memory mapped interface

```
blockram: entity work.rams_sp_wf
  generic map (
    RAM_WIDTH => 13,
    RAM_DEPTH => 256)
  port map (
    clk    => Ctrl.MEM1.clk,
    we     => Ctrl.MEM1.wr_enable,
    en     => Ctrl.MEM1.enable,
    addr   => Ctrl.MEM1.address,
    di     => Ctrl.MEM1.wr_data,
    do     => Mon.MEM1.rd_data,
    do_valid => Mon.MEM1.rd_data_valid);

other_blockram: entity work.rams_sp_wf
  generic map (
    RAM_WIDTH => 13,
    RAM_DEPTH => 256)
  port map (
    clk    => Ctrl.LEVEL_TEST.MEM.clk,
    en     => Ctrl.LEVEL_TEST.MEM.enable,
    we     => Ctrl.LEVEL_TEST.MEM.wr_enable,
    addr   => Ctrl.LEVEL_TEST.MEM.address,
    di     => Ctrl.LEVEL_TEST.MEM.wr_data,
    do     => Mon.LEVEL_TEST.MEM.rd_data,
    do_valid => Mon.LEVEL_TEST.MEM.rd_data_valid);
```

```
<node id="THING"  address="0x20" permission="rw"/>
<node id="MEM1" address="0x100" mode="incremental" size="0x100" permission="rw" fwinfo="type=mem13"/>

<node id="LEVEL_TEST" address="0x300">
  <node id="THING"  address="0x0" permission="rw"/>
  <node id="MEM" address="0x100" mode="incremental" size="0x100" permission="rw" fwinfo="type=mem13"/>
</node>
```

UIOuHAL and Simplified UIO searching

- Original method linking AXI slave to UIO# was complicated
  - Search of XML for labels
  - Search of /proc/device-tree for label
  - Get associated AXI addr for label
  - Use address for /sys/class/uio search
  - Get UIO #
- Kernel UIO update
  - mainlined around 2020.2
  - driver looks for "linux,uio-name"
  - can be used by udev for naming
- Now each UIO is sym-linked with a name
  Just search for to /dev/uio_label
- UIOμ*HAL* now supports both modes

UIOuHAL and Simplified UIO searching

- Original method linking AXI slave to UIO# was complicated
  - Search of XML for labels
  - Search of /proc/device-tree for label
  - Get associated AXI addr for label
  - Use address for /sys/class/uio search
  - Get UIO #
- Kernel UIO update
  - mainlined around 2020.2
  - driver looks for "linux,uio-name"
  - can be used by udev for naming
- Now each UIO is sym-linked with a name Just search for to /dev/uio_label
- UIOμ*HAL* now supports both modes

```
/dev/uio0
/dev/uio1
/dev/uio10
/dev/uio11
/dev/uio12
/dev/uio13
/dev/uio14
/dev/uio15
/dev/uio16
/dev/uio17
/dev/uio18
/dev/uio19
/dev/uio2
/dev/uio20
/dev/uio3
/dev/uio4
/dev/uio5
/dev/uio6
/dev/uio7
/dev/uio8
/dev/uio9
/dev/uio_AXI_MON -> uio0
/dev/uio_axi-pmon -> uio17
/dev/uio_C2C1_AXI_FW -> uio2
/dev/uio_C2C1_AXILITE_FW -> uio1
/dev/uio_C2C1_PHY -> uio3
/dev/uio_C2C2_AXI_FW -> uio5
/dev/uio_C2C2_AXILITE_FW -> uio4
/dev/uio_C2C2_PHY -> uio6
/dev/uio_CM -> uio11
/dev/uio_INT_AXI_FW -> uio7
/dev/uio_MEM_TEST -> uio15
/dev/uio_MONITOR -> uio8
/dev/uio_PL_MEM -> uio9
/dev/uio_PLXVC -> uio13
/dev/uio_SERV -> uio14
/dev/uio_SI -> uio10
/dev/uio_SLAVE_I2C -> uio16
/dev/uio_SM_INFO -> uio12
```

UIOuHAL and Simplified UIO searching

- Original method linking AXI slave to UIO# was complicated
  - Search of XML for labels
  - Search of /proc/device-tree for label
  - Get associated AXI addr for label
  - Use address for /sys/class/uio search
  - Get UIO #
- Kernel UIO update
  - mainlined around 2020.2
  - driver looks for "linux,uio-name"
  - can be used by udev for naming
- Now each UIO is sym-linked with a name
  Just search for to /dev/uio_label
- UIOμ*HAL* now supports both modes
- udev rule for automatic symlinking

```
[root@apollo201 ~]# cat /etc/udev/rules.d/uio.rules
SUBSYSTEM=="uio", MODE="0666", SYMLINK+="uio_$attr{name}"
[root@apollo201 ~]#
```

```
/dev/uio0
/dev/uio1
/dev/uio10
/dev/uio11
/dev/uio12
/dev/uio13
/dev/uio14
/dev/uio15
/dev/uio16
/dev/uio17
/dev/uio18
/dev/uio19
/dev/uio2
/dev/uio20
/dev/uio3
/dev/uio4
/dev/uio5
/dev/uio6
/dev/uio7
/dev/uio8
/dev/uio9
/dev/uio_AXI_MON -> uio0
/dev/uio_axi-pmon -> uio17
/dev/uio_C2C1_AXI_FW -> uio2
/dev/uio_C2C1_AXILITE_FW -> uio1
/dev/uio_C2C1_PHY -> uio3
/dev/uio_C2C2_AXI_FW -> uio5
/dev/uio_C2C2_AXILITE_FW -> uio4
/dev/uio_C2C2_PHY -> uio6
/dev/uio_CM -> uio11
/dev/uio_INT_AXI_FW -> uio7
/dev/uio_MEM_TEST -> uio15
/dev/uio_MONITOR -> uio8
/dev/uio_PL_MEM -> uio9
/dev/uio_PLXVC -> uio13
/dev/uio_SERV -> uio14
/dev/uio_SI -> uio10
/dev/uio_SLAVE_I2C -> uio16
/dev/uio_SM_INFO -> uio12
```

C2C Simplification with DT overlays

DTSI Overlay and DTBO files

Overview:

- Originally, the Zynq device-tree would have to be build with the C2C endpoint slave entries before boot.
- In practice this lead to a slow development as the endpoint users had to interact with the Zynq build.
- Device Tree Overlays allow for appending to the device tree after boot
- Now C2C build emits dtsi overlay files along with dtsi_chunk files (right)

C2C Simplification with DT overlays

DTSI Overlay and DTBO files

Overview:

- Originally, the Zynq device-tree would have to be build with the C2C endpoint slave entries before boot.
- In practice this lead to a slow development as the endpoint users had to interact with the Zynq build.
- Device Tree Overlays allow for appending to the device tree after boot
- Now C2C build emits dtsi overlay files along with dtsi_chunk files (right)

```
/dts-v1/;
 /plugin/;

/ {
  fragment@0 {
    target = <&amba_pl>;
    __overlay__ {
      axiSlaveCM_K_INFO: CM_K_INFO@BE003000 {
        compatible = "generic-uio";
        #address-cells = <2>;
        #size-cells = <2>;
        reg = <0x0 0xBE003000 0x0 0x1000>;
        label = "CM_K_INFO";
        linux,uio-name = "CM_K_INFO";
      };
    };
  };
};
```

C2C Simplification with DT overlays

DTSI Overlay and DTBO files

Overview:

- Originally, the Zynq device-tree would have to be build with the C2C endpoint slave entries before boot.
- In practice this lead to a slow development as the endpoint users had to interact with the Zynq build.
- Device Tree Overlays allow for appending to the device tree after boot
- Now C2C build emits dtsi overlay files along with dtsi_chunk files (right)

```
/dts-v1/;
 /plugin/;

/ {
  fragment@0 {
    target = <&amba_pl>;
    __overlay__ {
      axiSlaveCM_K_INFO: CM_K_INFO@BE003000 {
        compatible = "generic-uio";
        #address-cells = <2>;
        #size-cells = <2>;
        reg = <0x0 0xBE003000 0x0 0x1000>;
        label = "CM_K_INFO";
        linux,uio-name = "CM_K_INFO";
      };
    };
  };
};
```

Usage:

- DTSI overlay files must be converted to dtbo files
- DTC command:
  dtc -O dtb -o CM_K_INFO.dtbo -b 0 - CM_K_INFO.dtsi
- DTBO files copied to Zynq for loading

C2C Simplification with DT overlays

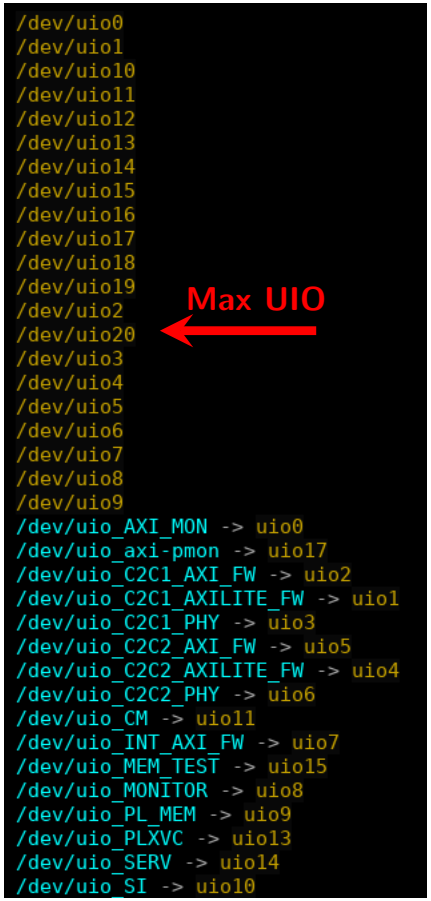
Using DTBO files

General:

- DTBO files are relatively easy to add at any time
- Current plan is to load them at the end of Linux boot.
- Script loads all dtbo files in a specified directory

C2C Simplification with DT overlays

Using DTBO files



General:

- DTBO files are relatively easy to add at any time
- Current plan is to load them at the end of Linux boot.
- Script loads all dtbo files in a specified directory

Example:

C2C Simplification with DT overlays

Using DTBO files

```
/dev/uio0
/dev/uio1
/dev/uio10
/dev/uio11
/dev/uio12
/dev/uio13
/dev/uio14
/dev/uio15
/dev/uio16
/dev/uio17
/dev/uio18
/dev/uio19
/dev/uio2
/dev/uio20
/dev/uio3
/dev/uio4
/dev/uio5
/dev/uio6
/dev/uio7
/dev/uio8
/dev/uio9
/dev/uio_AXI_MON -> uio0
/dev/uio_axi-pmon -> uio17
/dev/uio_C2C1_AXI_FW -> uio2
/dev/uio_C2C1_AXILITE_FW -> uio1
/dev/uio_C2C1_PHY -> uio3
/dev/uio_C2C2_AXI_FW -> uio5
/dev/uio_C2C2_AXILITE_FW -> uio4
/dev/uio_C2C2_PHY -> uio6
/dev/uio_CM -> uio11
/dev/uio_INT_AXI_FW -> uio7
/dev/uio_MEM_TEST -> uio15
/dev/uio_MONITOR -> uio8
/dev/uio_PL_MEM -> uio9
/dev/uio_PLXVC -> uio13
/dev/uio_SERV -> uio14
/dev/uio_SI -> uio10
```

Max UIO

General:

- DTBO files are relatively easy to add at any time
- Current plan is to load them at the end of Linux boot.
- Script loads all dtbo files in a specified directory

Example:

- Place DTBO file in /lib/firmware search path
- Create dir in configfs DT overlays
- Inject filename into path special file

```
[root@apollo201 ~]# ln -s /fw/dtbo/dtbo/CM_K_INFO.dtbo /lib/firmware/CM_K_INFO.dtbo
[root@apollo201 ~]# mkdir -p /sys/kernel/config/device-tree/overlays/CM_K_INFO
[root@apollo201 ~]# echo -n "CM_K_INFO.dtbo" > /sys/kernel/config/device-tree/overlays/CM_K_INFO/path
```

C2C Simplification with DT overlays

Using DTBO files

```
/dev/uio0
/dev/uio1
/dev/uio10
/dev/uio11
/dev/uio12
/dev/uio13
/dev/uio14
/dev/uio15
/dev/uio16
/dev/uio17
/dev/uio18
/dev/uio19
/dev/uio2
/dev/uio20
/dev/uio3
/dev/uio4
/dev/uio5
/dev/uio6
/dev/uio7
/dev/uio8
/dev/uio9
/dev/uio_AXI_MON -> uio0
/dev/uio_axi-pmon -> uio17
/dev/uio_C2C1_AXI_FW -> uio2
/dev/uio_C2C1_AXILITE_FW -> uio1
/dev/uio_C2C1_PHY -> uio3
/dev/uio_C2C2_AXI_FW -> uio5
/dev/uio_C2C2_AXILITE_FW -> uio4
/dev/uio_C2C2_PHY -> uio6
/dev/uio_CM -> uio11
/dev/uio_INT_AXI_FW -> uio7
/dev/uio_MEM_TEST -> uio15
/dev/uio_MONITOR -> uio8
/dev/uio_PL_MEM -> uio9
/dev/uio_PLXVC -> uio13
/dev/uio_SERV -> uio14
/dev/uio_SI -> uio10
```

Max UIO

General:

- DTBO files are relatively easy to add at any time
- Current plan is to load them at the end of Linux boot.
- Script loads all dtbo files in a specified directory

Example:

- Place DTBO file in /lib/firmware search path
- Create dir in configfs DT overlays
- Inject filename into path special file

New UIO device automatically loaded!

```
/dev/uio0
/dev/uio1
/dev/uio10
/dev/uio11
/dev/uio12
/dev/uio13
/dev/uio14
/dev/uio15
/dev/uio16
/dev/uio17
/dev/uio18
/dev/uio19
/dev/uio2
/dev/uio20
/dev/uio21
/dev/uio3
/dev/uio4
/dev/uio5
/dev/uio6
/dev/uio7
/dev/uio8
/dev/uio9
/dev/uio_AXI_MON -> uio0
/dev/uio_axi-pmon -> uio17
/dev/uio_C2C1_AXI_FW -> uio2
/dev/uio_C2C1_AXILITE_FW -> uio1
/dev/uio_C2C1_PHY -> uio3
/dev/uio_C2C2_AXI_FW -> uio5
/dev/uio_C2C2_AXILITE_FW -> uio4
/dev/uio_C2C2_PHY -> uio6
/dev/uio_CM -> uio11
/dev/uio_CM_K_INFO -> uio21
/dev/uio_INT_AXI_FW -> uio7
/dev/uio_MEM_TEST -> uio15
/dev/uio_MONITOR -> uio8
/dev/uio_PL_MEM -> uio9
/dev/uio_PLXVC -> uio13
/dev/uio_SERV -> uio14
/dev/uio_SI -> uio10
```

New max UIO

```
[root@apollo201 ~]# ln -s /fw/dtbo/dtbo/CM_K_INFO.dtbo /lib/firmware/CM_K_INFO.dtbo
[root@apollo201 ~]# mkdir -p /sys/kernel/config/device-tree/overlays/CM_K_INFO
[root@apollo201 ~]# echo -n "CM_K_INFO.dtbo" > /sys/kernel/config/device-tree/overlays/CM_K_INFO/path
```

Useful Tcl

Useful Tcl

Quad Print

```
QUAD: 224
QUAD: 225
     Ref Clk     :   J8 /   J7  :       REFCLK_CMS_P[1] / REFCLK_CMS_N[1]
     Tx/Rx       :   N4 /   N3  :   AXI_C2C_CM2_RX_P[0] / AXI_C2C_CM2_RX_N[0]
     Tx/Rx       :   P6 /   P5  :   AXI_C2C_CM2_TX_P[0] / AXI_C2C_CM2_TX_N[0]
     Tx/Rx       :   K2 /   K1  :   AXI_C2C_CM1_RX_P[1] / AXI_C2C_CM1_RX_N[1]
     Tx/Rx       :   L4 /   L3  :   AXI_C2C_CM1_TX_P[1] / AXI_C2C_CM1_TX_N[1]
     Tx/Rx       :   J4 /   J3  :   AXI_C2C_CM1_RX_P[0] / AXI_C2C_CM1_RX_N[0]
     Tx/Rx       :   K6 /   K5  :   AXI_C2C_CM1_TX_P[0] / AXI_C2C_CM1_TX_N[0]
QUAD: 226
     Ref Clk     :  H10 /   H9  :         REFCLK_C2C2_P / REFCLK_C2C2_N
     Ref Clk     :  F10 /   F9  :      REFCLK_C2C1_P[1] / REFCLK_C2C1_N[1]
     Tx/Rx       :   H2 /   H1  :            LDAQ_RX_P / LDAQ_RX_N
     Tx/Rx       :   H6 /   H5  :            LDAQ_TX_P / LDAQ_TX_N
QUAD: 227
     Ref Clk     :  D10 /   D9  :       REFCLK_CMS_P[0] / REFCLK_CMS_N[0]
     Ref Clk     :  B10 /   B9  :         REFCLK_REC_P / REFCLK_REC_N
     Tx/Rx       :   D2 /   D1  :        CM1_TCDS_TTS_P / CM1_TCDS_TTS_N
     Tx/Rx       :   D6 /   D5  :        CM1_TCDS_TTC_P / CM1_TCDS_TTC_N
     Tx/Rx       :   C4 /   C3  :            TCDS_TTC_N / TCDS_TTC_P
     Tx/Rx       :   C8 /   C7  :            TCDS_TTS_P / TCDS_TTS_N
     Tx/Rx       :   B2 /   B1  :   AXI_C2C_CM2_RX_P[1] / AXI_C2C_CM2_RX_N[1]
     Tx/Rx       :   B6 /   B5  :   AXI_C2C_CM2_TX_P[1] / AXI_C2C_CM2_TX_N[1]
     Tx/Rx       :   A4 /   A3  :        CM2_TCDS_TTS_P / CM2_TCDS_TTS_N
     Tx/Rx       :   A8 /   A7  :        CM2_TCDS_TTC_P / CM2_TCDS_TTC_N
QUAD: 505
```

```
QUAD: 224
     Ref Clk     :   R8 /   R7  :              un-used / un-used
     Ref Clk     :   N8 /   N7  :              un-used / un-used
     Tx/Rx       :   V2 /   V1  :              un-used / un-used
     Tx/Rx       :   W4 /   W3  :              un-used / un-used
     Tx/Rx       :   U4 /   U3  :              un-used / un-used
     Tx/Rx       :   V6 /   V5  :              un-used / un-used
     Tx/Rx       :   T2 /   T1  :              un-used / un-used
     Tx/Rx       :   T6 /   T5  :              un-used / un-used
     Tx/Rx       :   P2 /   P1  :              un-used / un-used
     Tx/Rx       :   R4 /   R3  :              un-used / un-used
QUAD: 225
     Ref Clk     :   L8 /   L7  :              un-used / un-used
     Ref Clk     :   J8 /   J7  :       REFCLK_CMS_P[1] / REFCLK_CMS_N[1]
     Tx/Rx       :   N4 /   N3  :   AXI_C2C_CM2_RX_P[0] / AXI_C2C_CM2_RX_N[0]
     Tx/Rx       :   P6 /   P5  :   AXI_C2C_CM2_TX_P[0] / AXI_C2C_CM2_TX_N[0]
     Tx/Rx       :   M2 /   M1  :              un-used / un-used
     Tx/Rx       :   M6 /   M5  :              un-used / un-used
     Tx/Rx       :   K2 /   K1  :   AXI_C2C_CM1_RX_P[1] / AXI_C2C_CM1_RX_N[1]
     Tx/Rx       :   L4 /   L3  :   AXI_C2C_CM1_TX_P[1] / AXI_C2C_CM1_TX_N[1]
     Tx/Rx       :   J4 /   J3  :   AXI_C2C_CM1_RX_P[0] / AXI_C2C_CM1_RX_N[0]
     Tx/Rx       :   K6 /   K5  :   AXI_C2C_CM1_TX_P[0] / AXI_C2C_CM1_TX_N[0]
QUAD: 226
     Ref Clk     :  H10 /   H9  :         REFCLK_C2C2_P / REFCLK_C2C2_N
     Ref Clk     :  F10 /   F9  :      REFCLK_C2C1_P[1] / REFCLK_C2C1_N[1]
     Tx/Rx       :   H2 /   H1  :            LDAQ_RX_P / LDAQ_RX_N
```

- Prints all the Quads in an FPGA
- Lists the transceiver clk/io pins and the signals connected to them
- Source tcl file here.

Useful Tcl

Tcl based IPCores

```
    #create IP
    create_ip -vlnv [get_ipdefs -filter {NAME == clk_wiz}] -module_name ${name} -dir ${output_path}


    set_property -dict [list \
                            CONFIG.PRIM_SOURCE {Differential_clock_capable_pin} \
                            CONFIG.PRIM_IN_FREQ {100} \
                            CONFIG.PRIMARY_PORT {clk_in} \
                            CONFIG.NUM_OUT_CLKS {2} \
                            CONFIG.CLKOUT2_USED {true} \
                            CONFIG.CLK_OUT1_PORT {clk_50Mhz} \
                            CONFIG.CLK_OUT2_PORT {clk_200Mhz} \
                            CONFIG.CLKOUT1_REQUESTED_OUT_FREQ {50} \
                            CONFIG.CLKOUT2_REQUESTED_OUT_FREQ {200} \
                           ] [get_ips ${name} ]
}
```

- Capture Xilinx IP Core creation from Vivado
- Use get_ipdefs to select the latest version of the IP in this Vivado session.
- Easy to change parameters like clock frequency or ila ports & sizes.
- Version control diffs are meaningful.
- Source clock example file here.
- Source ila example file here.

Summary

- Apollo SM Rev2 has been built and is running with Enclustra ZynqMP SoC
- Apollo SM Rev2a is being produced to solve CMS TCDS2 issue
- Much work has been done on FW/SW build integration
  - YAML configuration file for AXI slave automation
  - Updated UIO interface for easier μHAL interface
  - Use of device-tree overlays to map Command Modules at runtime.

Bonus slides

OLD! AXI μHAL (Device-tree)

- AXI slaves can be accessed in userspace via UIO
- Modify "compatible" tag to make a UIO device (system-user.dtsi)
- Tag "label" helps map slave to UIO #

Appended to system-user.dtsi

```
&XVC1{
  compatible = "generic-uio";
    label = "XVC1";
};

amba_pl {
  axiSlaveSERV: SERV@43C20000 {
    compatible = "generic-uio";
    reg = <0x43C20000 0x10000>;
    label = "SERV";
  };
};
```

Address table top nodes link "label" to μHAL address

```
<node id="TOP">
  <node id="XVC1" address="0x06000000">
    <node id="LENGTH"  address="0x0" permission="rw" description="Length of shift operation in bits"/>
    <node id="TMS_VECTOR"  address="0x1" permission="rw" description="Test Mode Select (TMS) Bit Vector"/>
    <node id="TDI_VECTOR"  address="0x2" permission="rw" description="Test Data In (TDI) Bit Vector"/>
    <node id="TDO_VECTOR"  address="0x3" permission="rw" description="Test Data Out (TDO) Capture Vector"/>
    <node id="GO"  address="0x4" mask="0x1" permission="rw" description="Enable shift operation"/>
    <node id="LOOPBACK"  address="0x4" mask="0x2" permission="rw" description="Control bit to loopback TDI to TDO
inside Debug Bridge IP"/>
  </node>
  <node id="SERV" address="0x04000000">
    <node id="SI5344"  address="0x00" permission="r">
      <node id="OE" address="0x0" permission="rw" mask="0x1" description="Enable Si5344 outputs"/>
      <node id="EN" address="0x0" permission="rw" mask="0x2" description="Power on Si5344"/>
```

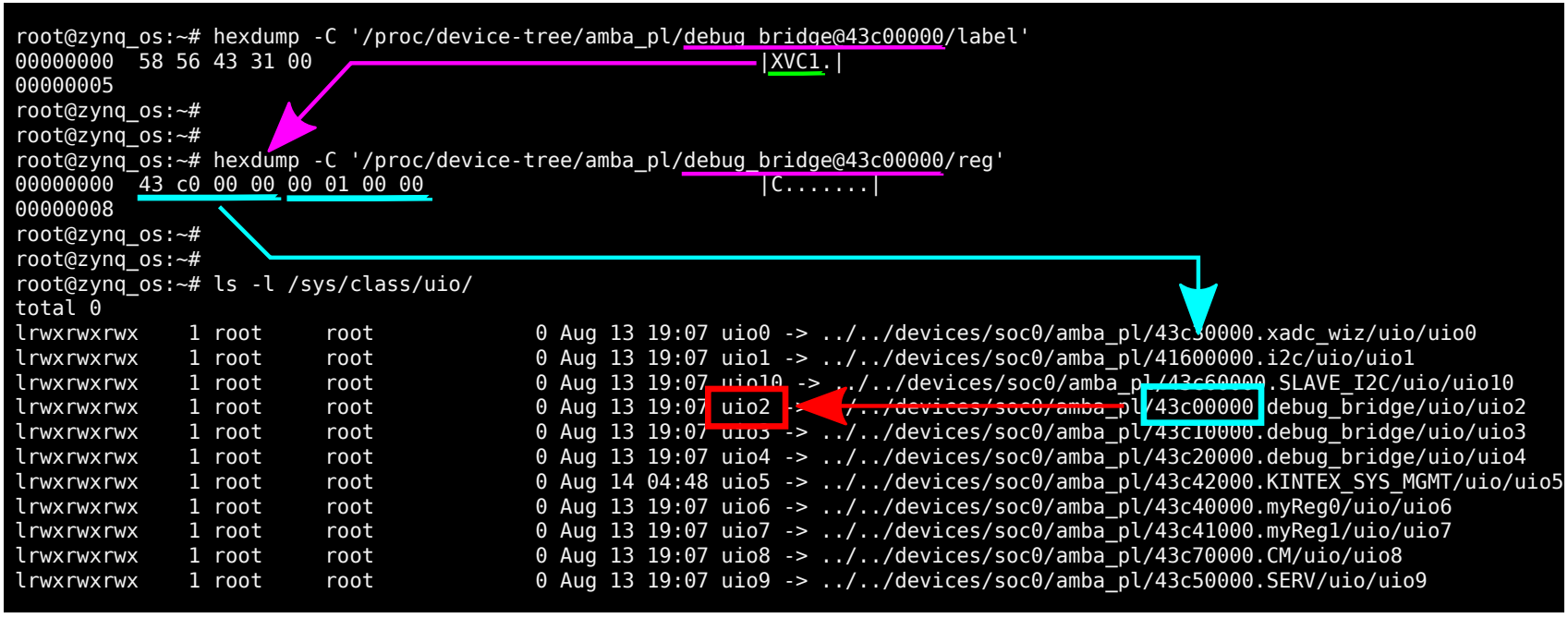
OLD! AXI μHAL (Address table)

- Address tables map AXI slave name to user friendly names and code friendly addresses.
- Nodes under the top node point to AXI slaves with the same name.
- These nodes also set the code level addresses (won't change)
- Mapping from Address table name to UIO to AXI slave handled automatically

Example Address Table:

```
<node id="TOP">
  <node id="XVC1" address="0x06000000">
    <node id="LENGTH"  address="0x0" permission="rw" description="Length of shift operation in bits"/>
    <node id="TMS_VECTOR"  address="0x1" permission="rw" description="Test Mode Select (TMS) Bit Vector"/>
    <node id="TDI_VECTOR"  address="0x2" permission="rw" description="Test Data In (TDI) Bit Vector"/>
    <node id="TDO_VECTOR"  address="0x3" permission="rw" description="Test Data Out (TDO) Capture Vector"/>
    <node id="GO"  address="0x4" mask="0x1" permission="rw" description="Enable shift operation"/>
    <node id="LOOPBACK"  address="0x4" mask="0x2" permission="rw" description="Control bit to loopback TDI to TDO
inside Debug Bridge IP"/>
  </node>
  <node id="SERV" address="0x04000000">
    <node id="SI5344"  address="0x00" permission="r">
      <node id="OE" address="0x0" permission="rw" mask="0x1" description="Enable Si5344 outputs"/>
      <node id="EN" address="0x0" permission="rw" mask="0x2" description="Power on Si5344"/>
      <node id="SOMETHING" address="0x0" permission="rw" mask="0x4"/>
      <node id="INT" address="0x0" permission="r" mask="0x10" description="Si5344 i2c interrupt"/>
      <node id="LOL" address="0x0" permission="r" mask="0x20" description="Si5344 Loss of lock"/>
      <node id="LOS" address="0x0" permission="r" mask="0x40" description="Si5344 Loss of signal"/>
    </node>
    <node id="TCDS" address="0x04" permission="r">
      <node id="TTC_SOURCE" address="0x0" permission="rw" mask="0x1" description="TTC source select (0:TCDS,1:TTC_FAKE"/>
```

OLD! AXI μHAL (Software Internals)



Mapping AXI slave name to UIO device number

- Linux proc filesystem lists the tags from the device-tree
- AXI slave label can be found here
- AXI slave physical address info can be used to find UIO number in sys filesystem
- μHAL can use this info to map address table nodes to UIO numbers