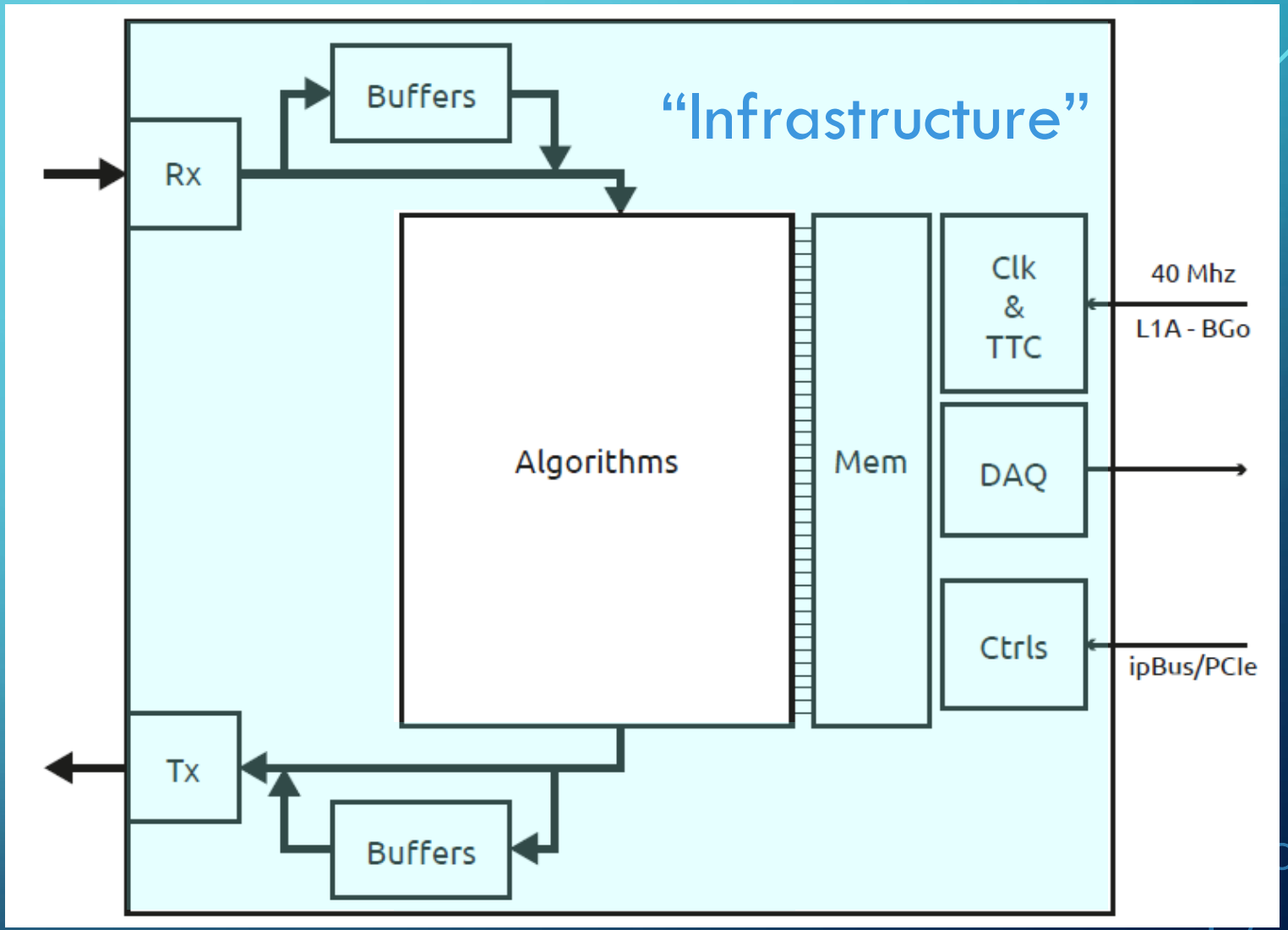
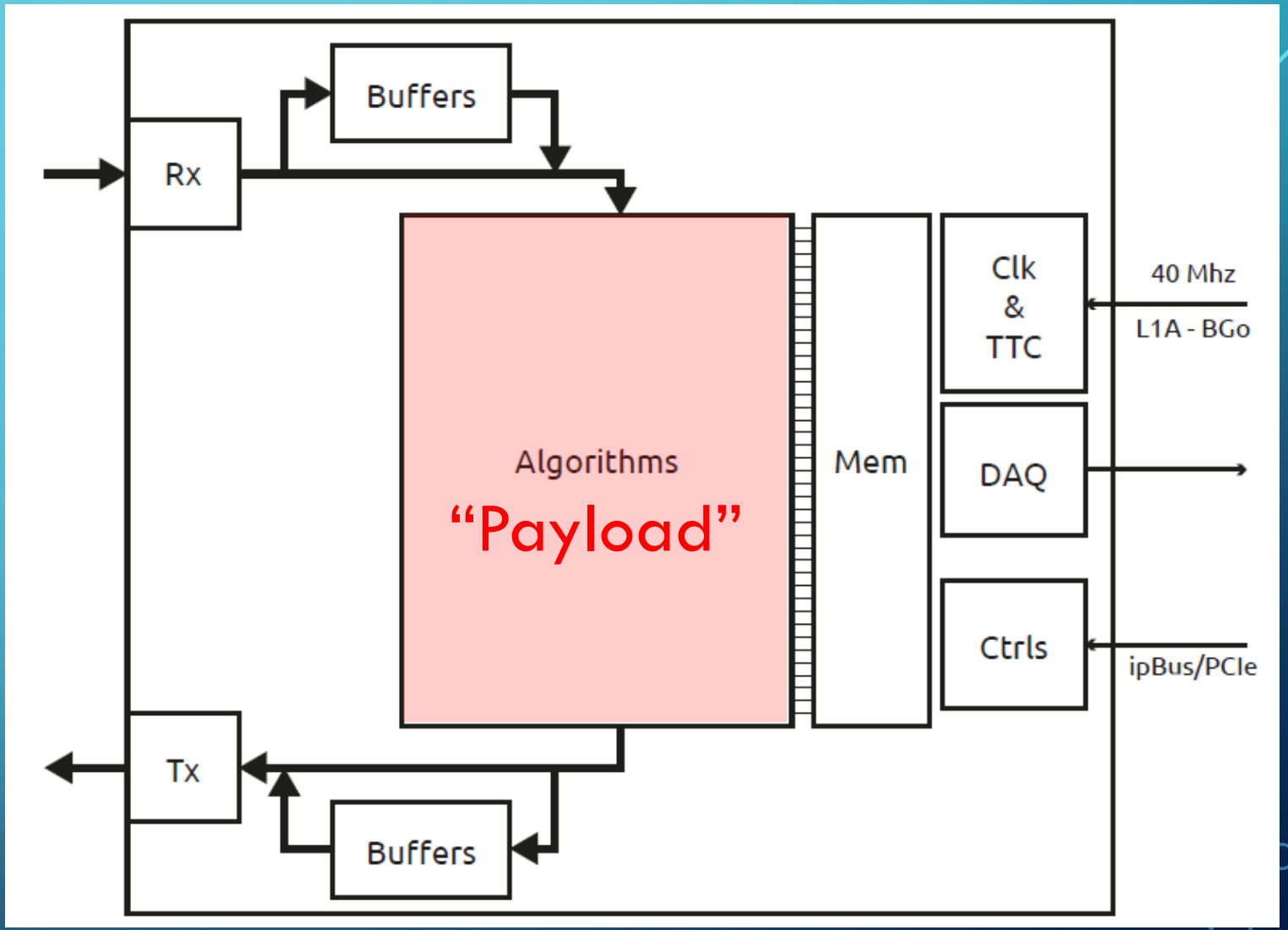
GETTING STARTED WITH THE EMP FRAMEWORK – PART 1

ANDY ROSE, IMPERIAL COLLEGE LONDON

RECALL


Buffers
"Infrastructure"
Rx
Algorithms
Mem
Clk
&
TTC
40 Mhz
L1A - BGo
DAQ
Ctrls
ipBus/PCIe
Tx
Buffers

RECALL


Buffers
Rx
Algorithms
"Payload"
Mem
Clk
&
TTC
40 Mhz
L1A - BGo
DAQ
Ctrls
ipBus/PCIe
Tx
Buffers

THE BASIC IDEA

1. The emp-fwk repo contains top-level designs for various different FPGAs and boards
2. Each of these designs instantiates an entity - named emp_payload - and connects its input/output ports to the clocking infrastructure, control bus, and input/output buffers.
3. Parameters that might have to be changed between different algorithms or different FPGAs are specified as constants in a VHDL package emp_project_decl.

1 - PREREQUISITES

```
# Set up python
sudo yum install python3-pip python3-devel
pip3 install --user pipenv


# Install the IPBB tool
curl -L https://github.com/ipbus/ipbb/archive/v0.5.2.tar.gz | tar xvz
source ipbb-0.5.2/env.sh
```

Already installed:
Included here for completeness

1 - PREREQUISITES

```
ipbb init my-firmware

cd my-firmware


ipbb add git https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-fwk.git -b feature/kcu105

ipbb add git https://gitlab.cern.ch/ttc/legacy_ttc.git -b v2.1

ipbb add git https://github.com/ipbus/ipbus-firmware -b v1.5
```

Already installed:
Included here for completeness

1 - PREREQUISITES: THREE THINGS TO NOTE

```
ipbb init my-firmware
cd my-firmware
```

EMP Framework currently stored on CERN Gitlab for our convenience:
How to manage wider distribution not currently thought through!

```
ipbb add git https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-fwk.git -b feature/kcu105
ipbb add git https://gitlab.cern.ch/ttc/legacy_ttc.git -b v2.1
ipbb add git https://github.com/ipbus/ipbus-firmware -b v1.5
```

1 - PREREQUISITES: THREE THINGS TO NOTE

```
ipbb init my-firmware

cd my-firmware


ipbb add git https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-
fwk.git -b feature/kcu105

ipbb add git https://gitlab.cern.ch/ttc/legacy_ttc.git -b v2.1

ipbb add git https://github.com/ipbus/ipbus-firmware -b v1.5
```

Using CMS legacy trigger and timing tools:
would need changing for a different experiment

1 - PREREQUISITES: THREE THINGS TO NOTE

```
ipbb init my-firmware

cd my-firmware


ipbb add git https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-fwk.git -b feature/kcu105

ipbb add git https://gitlab.cern.ch/ttc/legacy_ttc.git -b v2.1

ipbb add git https://github.com/ipbus/ipbus-firmware -b v1.5
```

IPbus is on Github:
completely public and independent of EMP

1 - PREREQUISITES

- You now have the build tools and all the infrastructure firmware you need for building a firmware for a typical HEP Trigger/DAQ application
  - SIMPLES!
- Only thing missing is a payload

2 - PAYLOAD: ADD FROM AN EXISTING REPOSITORY

```
ipbb add svn [repo_url]

ipbb add git [repo_url]
```

But we are not going to do this:
What would be the fun in that…

2 - PAYLOAD: STARTING FROM SCRATCH

- Firmware project sources have no natural “structure”
  - Traditionally, files & folders end up being a real mess
- IPBB imposes a fixed directory structure to keep everything organized

```
mkdir -p src/my-algo-repo/an-algo/firmware/{cfg,hdl}
```

2 - PAYLOAD: STARTING FROM SCRATCH

- You are now ready to create a payload
- Could start from blank file
  - but that would be tedious
- EMP framework includes a “Null” algorithm which can be copied
  - Don’t we spoil you :-)

```
cp src/emp-fwk/components/payload/firmware/hdl/emp_payload.vhd
   src/my-algo-repo/an-algo/firmware/hdl/
```

2 - PAYLOAD: STARTING FROM SCRATCH

- IPBB uses Dependency (“dep”) files as a distributed way of organizing sources
- If you want to use your new payload, you need to add it to a dep file
- Open `src/my-algo-repo/an-algo/firmware/cfg/top.dep`
- Add the line `src emp_payload.vhd`
- The IPBB “src” command defaults to the local “hdl” folder

2 - PAYLOAD: STARTING FROM SCRATCH

- The “-c” option changes the search path
- Most of the time we want to constrain the payload to the payload area
  - So add a design constraints file
  - This is a “standard” element, so we have one in the Repo
  - Add the line

```
src -c emp-fwk:components/payload ../ucf/emp_simple_payload.tcl
```

Repo

Path excluding the standardized “firmware” section

Recall – default folder is “hdl”, constraints stored in “ucf”

2 - PAYLOAD: STARTING FROM SCRATCH

- IPbus uses XML files to handle address space
- Add an address table so we can use IPbus later
- Add the line `addrtab -c emp-fwk:components/payload emp_payload.xml`

2 - PAYLOAD: STARTING FROM SCRATCH

- Finally, we need to target a specific board
  - We have already made a large number of board configurations
- We are using the KCU105 dev board
- Add the line `include -c emp-fwk:boards/kcu105`

Board	Dependency file command
HTG K800	include -c emp-fwk:boards/k800
MPUltra	include -c emp-fwk:boards/mpultra
VCU118	include -c emp-fwk:boards/vcu118
Serenity KU115 SO1	include -c emp-fwk:boards/serenity/dc_ku115 dc_ku115_so1.dep
Serenity KU115 TM1/BM1	include -c emp-fwk:boards/serenity/dc_ku115 dc_ku115_am1.dep
Serenity KU15P v1	include -c emp-fwk:boards/serenity/dc_ku15p dc_ku15p_sm1_v1.dep
Serenity KU15P v2	include -c emp-fwk:boards/serenity/dc_ku15p dc_ku15p_sm1_v2.dep

3 – CONFIGURING THE INFRASTRUCTURE

- emp_project_decl is a VHDL package that defines several configurable settings of the EMP framework, such as clock frequencies for receiving data from the I/O channels, and which firmware components are instantiated in each datapath region (e.g. input buffers, output buffers, input transceiver control logic, output transceiver control logic)

3 – CONFIGURING THE INFRASTRUCTURE

Name	Type	Meaning
PAYLOAD_REV	std_logic_vector(31 downto 0)	32-bit version number for the algorithm
LHC_BUNCH_COUNT	integer	Number of bunch crossings in an LHC orbit
LB_ADDR_WIDTH	integer	Address width for data words in I/O buffers
CLOCK_RATIO	integer	Ratio of frequency of I/O channel data clock to LHC clock
CLOCK_RATIO_AUX	clock_ratio_array_t	Ratio of frequency of clocks in clk_payload port (the auxiliary clocks) to LHC clock
CLOCK_COMMON_RATIO	integer	Ratio of frequency of feedback VCO in MMCM that generates clocks for ports clk_p and clk_payload to the LHC clock
REGION_CONF	region_conf_array_t	Specifies what components (buffers, formatters, transceivers) will be enabled in each datapath region.

3 – CONFIGURING THE INFRASTRUCTURE

Name	Type	Meaning
PAYLOAD_REV	std_logic_vector(31 downto 0)	32-bit version number for the algorithm
LHC_BUNCH_COUNT	integer	Number of bunch crossings in an LHC orbit
LB_ADDR_WIDTH	integer	Address width for data words in I/O buffers
CLOCK_RATIO	integer	Ratio of frequency of I/O channel data clock to LHC clock
CLOCK_RATIO_AUX	clock_ratio_array_t	Ratio of frequency of clocks in clk_payload port (the auxiliary clocks) to LHC clock
CLOCK_COMMON_RATIO	integer	Ratio of frequency of feedback VCO in MMCM that generates clocks for ports clk_p and clk_payload to the LHC clock
REGION_CONF	region_conf_array_t	Specifies what components (buffers, formatters, transceivers) will be enabled in each datapath region.

Recall we used CMS timing infra

3 – CONFIGURING THE INFRASTRUCTURE

Name	Type	Meaning
PAYLOAD_REV	std_logic_vector(31 downto 0)	32-bit version number for the algorithm
LHC_BUNCH_COUNT	integer	Number of bunch crossings in an LHC orbit
LB_ADDR_WIDTH	integer	Address width for data words in I/O buffers
CLOCK_RATIO	integer	Ratio of frequency of I/O channel data clock to LHC clock
CLOCK_RATIO_AUX	clock_ratio_array_t	Ratio of frequency of clocks in clk_payload port (the auxiliary clocks) to LHC clock
CLOCK_COMMON_RATIO	integer	Ratio of frequency of feedback VCO in MMCM that generates clocks for ports clk_p and clk_payload to the LHC clock
REGION_CONF	region_conf_array_t	Specifies what components (buffers, formatters, transceivers) will be enabled in each datapath region.

Sets the main payload clock as a multiple of the master clock

3 – CONFIGURING THE INFRASTRUCTURE

Name	Type	Meaning
PAYLOAD_REV	std_logic_vector(31 downto 0)	32-bit version number for the algorithm
LHC_BUNCH_COUNT	integer	Number of bunch crossings in an LHC orbit
LB_ADDR_WIDTH	integer	Address width for data words in I/O buffers
CLOCK_RATIO	integer	Ratio of frequency of I/O channel data clock to LHC clock
CLOCK_RATIO_AUX	clock_ratio_array_t	Ratio of frequency of clocks in clk_payload port (the auxiliary clocks) to LHC clock
CLOCK_COMMON_RATIO	integer	Ratio of frequency of feedback VCO in MMCM that generates clocks for ports clk_p and clk_payload to the LHC clock
REGION_CONF	region_conf_array_t	Specifies what components (buffers, formatters, transceivers) will be enabled in each datapath region.

Addition clocks are available

3 – CONFIGURING THE INFRASTRUCTURE

Name	Type	Meaning
PAYLOAD_REV	std_logic_vector(31 downto 0)	32-bit version number for the algorithm
LHC_BUNCH_COUNT	integer	Number of bunch crossings in an LHC orbit
LB_ADDR_WIDTH	integer	Address width for data words in I/O buffers
CLOCK_RATIO	integer	Ratio of frequency of I/O channel data clock to LHC clock
CLOCK_RATIO_AUX	clock_ratio_array_t	Ratio of frequency of clocks in clk_payload port (the auxiliary clocks) to LHC clock
CLOCK_COMMON_RATIO	integer	Ratio of frequency of feedback VCO in MMCM that generates clocks for ports clk_p and clk_payload to the LHC clock
REGION_CONF	region_conf_array_t	Specifies what components (buffers, formatters, transceivers) will be enabled in each datapath region.

"Voodoo" which instantiates all the highspeed links + infrastructure

3 – CONFIGURING THE INFRASTRUCTURE

- Again, could start from blank file, but that would be tedious: EMP framework again includes a default version which can be copied

```
cp src/emp-fwk/projects/examples/kcu105/firmware/hdl/kcu105_decl_full.vhd
   src/my-algo-repo/an-algo/firmware/hdl/emp_project_decl.vhd
```

- And, again, we need to add it to our dep file, so
- To the dep file, add the line `src emp_project_decl.vhd`

CONGRATULATIONS

- Congratulations! You now have all the elements for a fully-functioning[1] firmware for a typical HEP Trigger/DAQ application

[1] We will clarify this statement later

CONGRATULATIONS

- Congratulations! You now have all the elements for a fully-functioning[1] firmware for a typical HEP Trigger/DAQ application
- Now we just have to build it…
  - But IPBB does that for us too

[1] We will clarify this statement later

BUILDING

```
ipbb proj create vivado my_algo my-algo-repo:an-algo -t top.dep

cd proj/my_algo

ipbb vivado project
```

BUILDING

```
ipbb proj create vivado my_algo my-algo-repo:an-algo -t top.dep

cd proj/my_algo

ipbb vivado project
```

We are creating a Vivado project area

Project name

The path to our repo

BUILDING

```
ipbb proj create vivado my_algo my-algo-repo:an-algo -t top.dep

cd proj/my_algo

ipbb vivado project
```

Creating the Vivado project

BUILDING

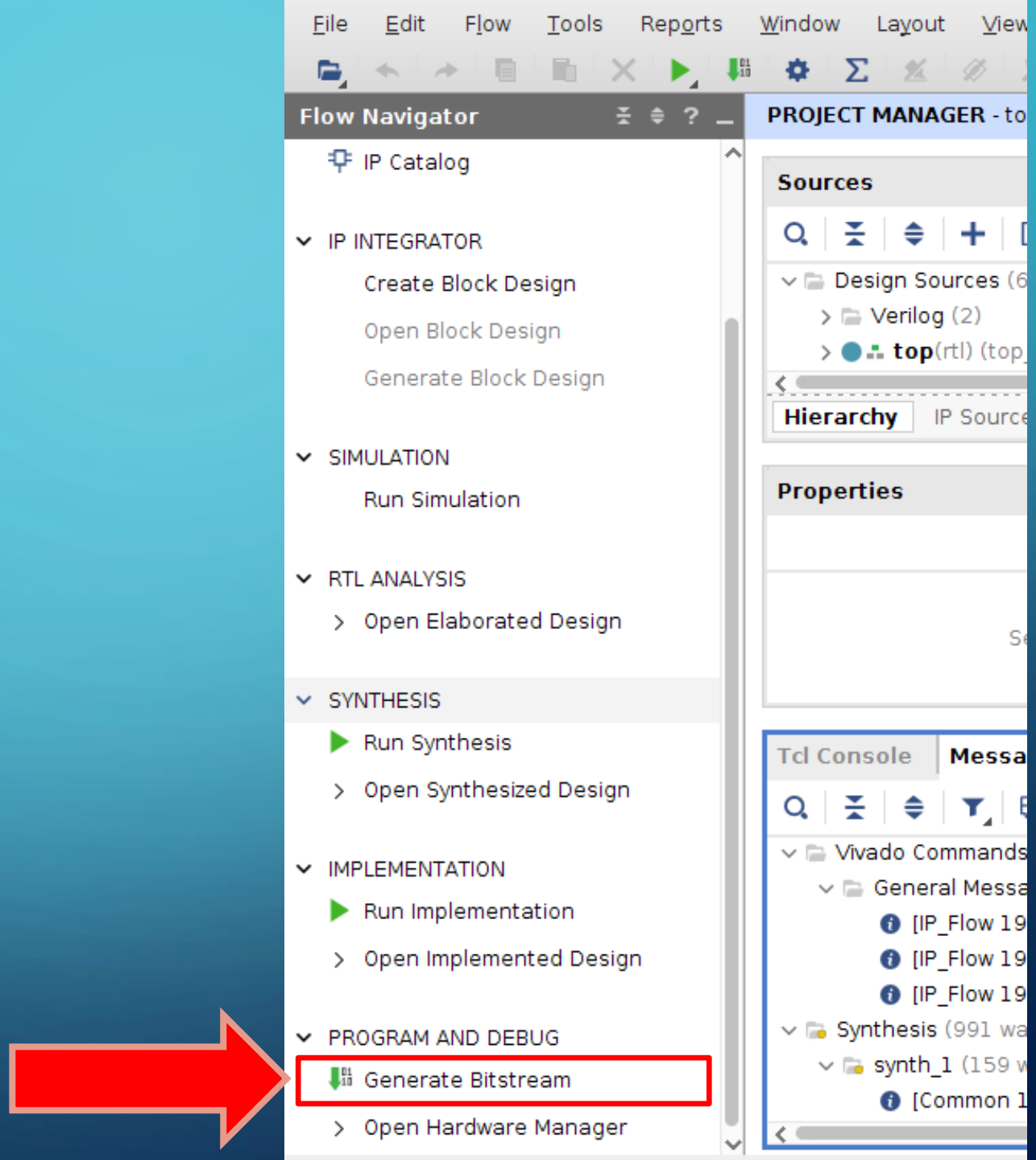
- We can then do

```
ipbb vivado synth -j4 impl -j4
ipbb vivado package
```

- Which will run all necessary steps to build a bit-file and package it up for distribution
- Else we can open the project manually `vivado my_algo/my_algo.xpr`
  - Which is more useful when we are debugging our work!
  - So let's do that

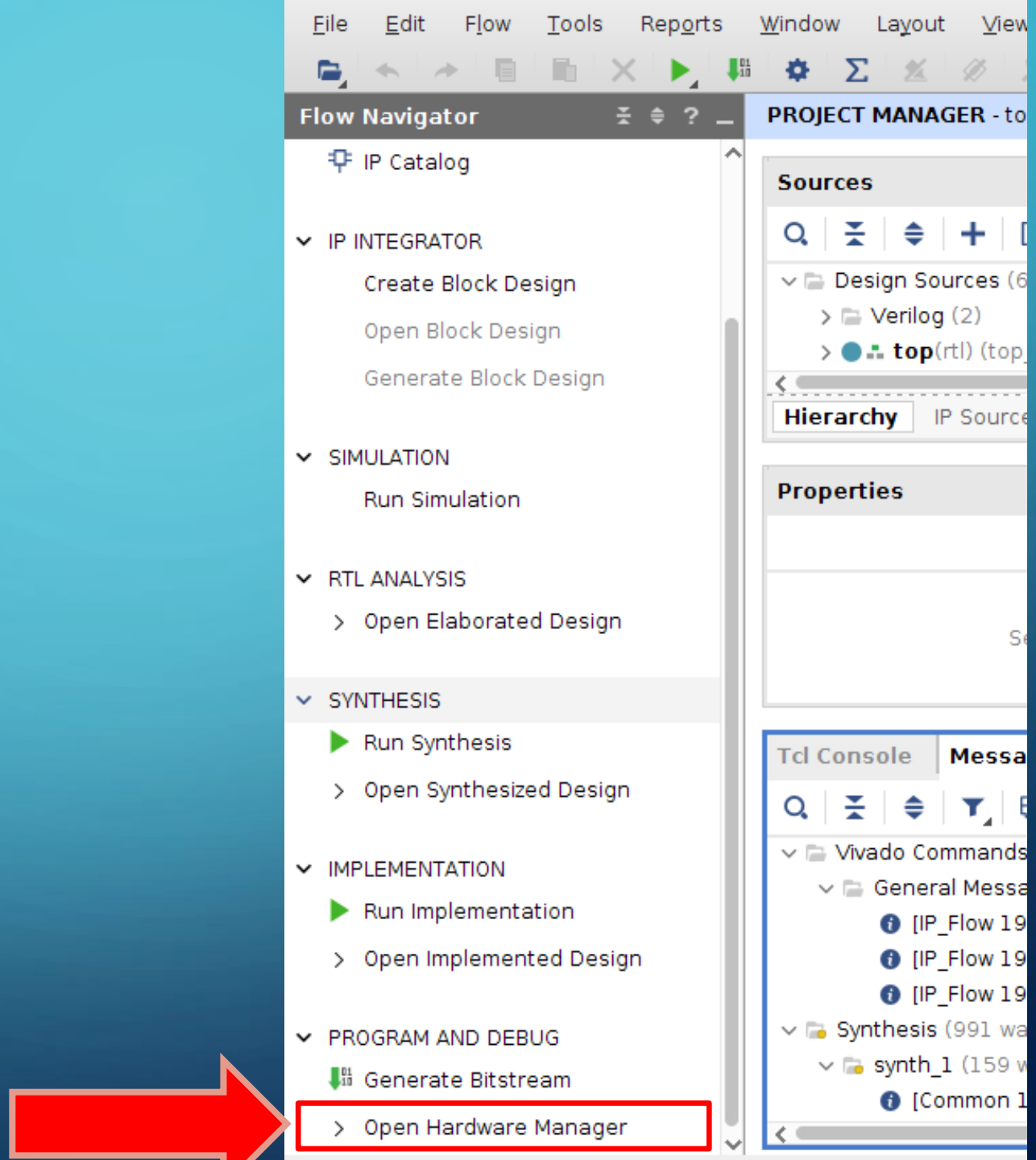
BUILDING

- And hit “yes/ok” to everything

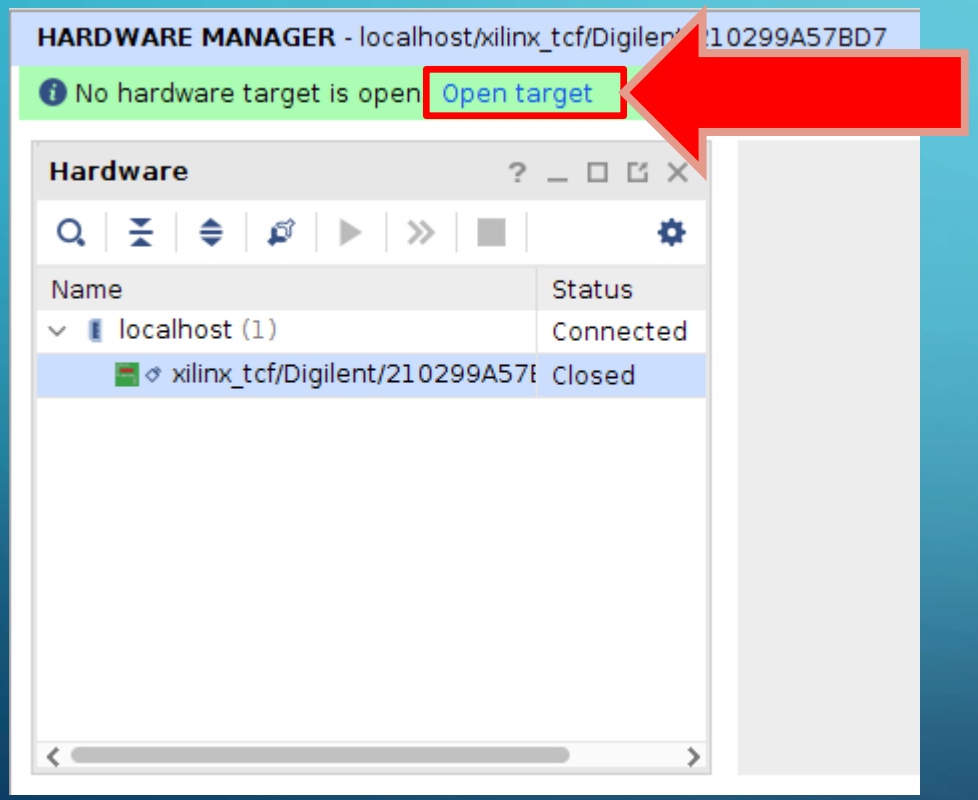


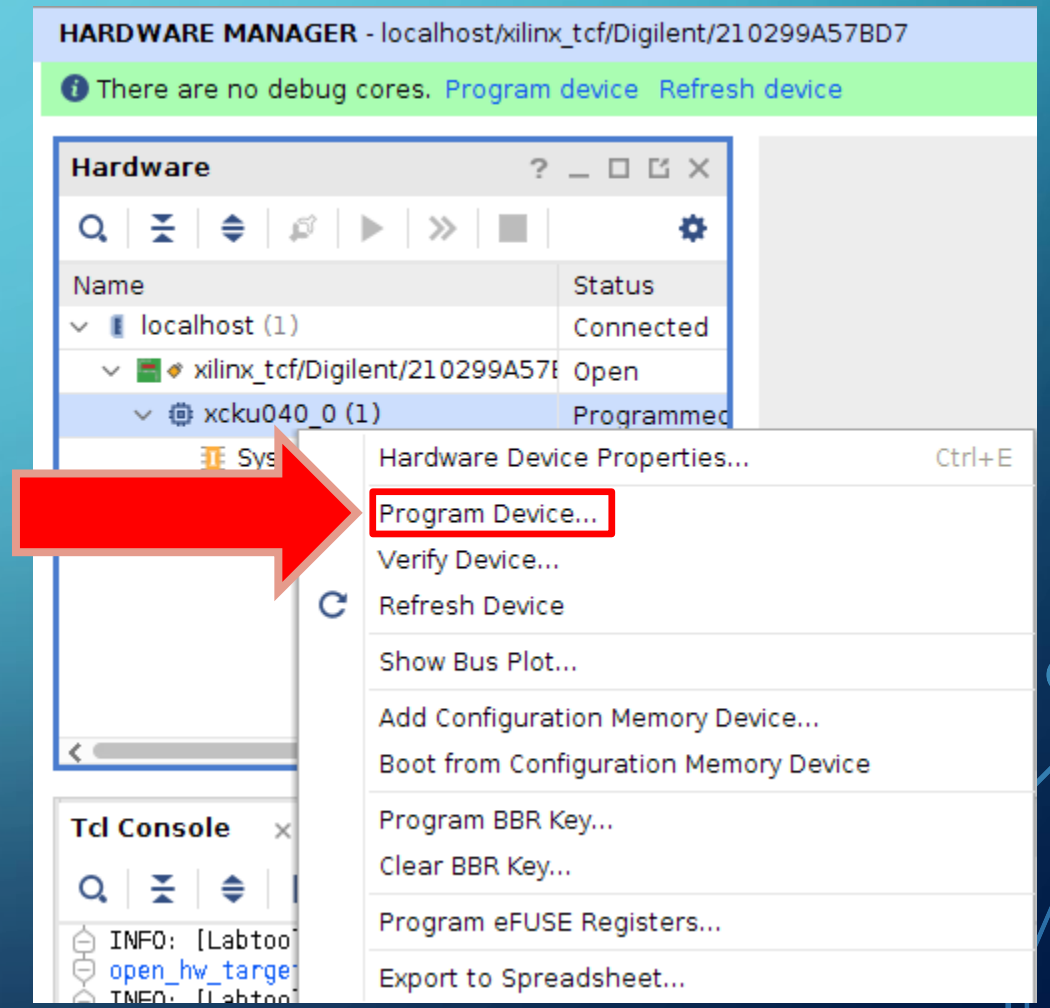
PROGRAMMING

- Open the hardware

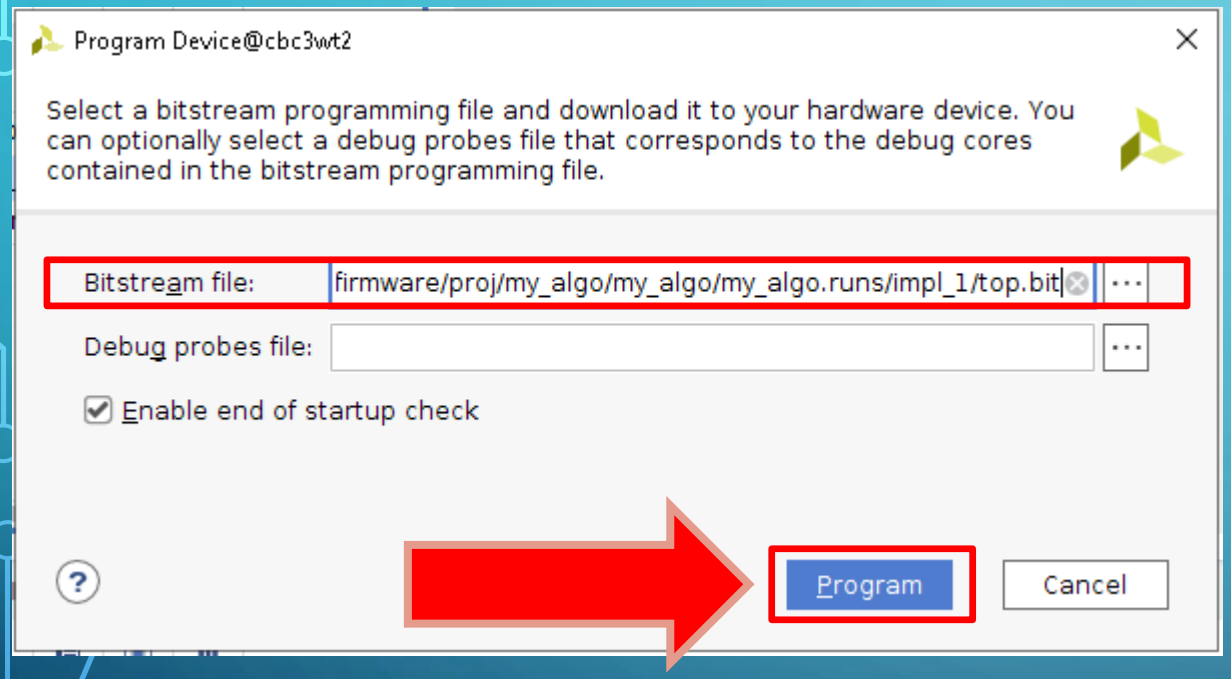


PROGRAMMING





PROGRAMMING



- After reprogramming, on the commandline: `pci_reconnect`

REFERENCES

- http://ipbus.web.cern.ch/ipbus/doc/user/html/firmware/ipbb-primer.html