MANCHESTER 1824

The University of Manchester

AsipIDE Tutorial

Bringing together GALS design and open-source tools in a hardware-software-FPGA co-simulation flow

Lilian Janin, Doug Edwards
ASYNC-NOCS 2010

MANCHESTER 1824

The University of Manchester

EU Project GALAXY

- GALs interfAce for compleX digital sYstem integration
- Supported by

  European Commission
  7^{th} Framework Programme (2007-2013)
  Objective ICT-2007.3.3:
  Embedded System Design

MANCHESTER 1824

The University of Manchester

EU Project GALAXY

- GALAXY Project Partners
  - IHP GmbH - Innovations for High Performance Microelectronics (Germany)
  - EPFL - Ecole Polytechnique Fédérale de Lausanne (Switzerland)
  - Università di Bologna (Italy)
  - Silistix UK Ltd. (United Kingdom)
  - Infineon Technologies AG (Germany)

MANCHESTER 1824

The University of Manchester

EU Project GALAXY

- GALAXY Project Goals
  - Remove existing barriers to the adoption of GALS technology
  - Integrated GALS design flow
  - Interoperability framework between existing open and commercial CAD tools
  - Heterogeneous systems at mixed levels of abstraction
  - Novel Network-on-Chip capabilities

MANCHESTER
1824

The University of Manchester

EU Project GALAXY

- GALAXY Demonstrator:
  Wireless communication system in 40nm CMOS process
  - Evaluate GALS approach to solve system integration issues
  - Prove robustness to process variability problems in nanoscale geometries
  - Explore the low EMI properties, inherent low-power features

MANCHESTER
1824

The University of Manchester

Tutorial Overview

- General presentation of tools and IDE
- Demo of main features
- Hands-on: A home surveillance system

MANCHESTER
1824

The University of Manchester

General presentation of tools and IDE

- Motivation
  - Bringing GALS to the masses!
  - Gcc brought software programming to home users
  - Many open-source/freely-available hardware design tools
    - Icarus Verilog simulator
    - Xilinx ISE
    - VHDL Alliance tools from Lip6
  - But full open-source hardware-software-FPGA design flow still unclear
  - Also integrates commercial tools

MANCHESTER
1824

The University of Manchester

General presentation of tools and IDE

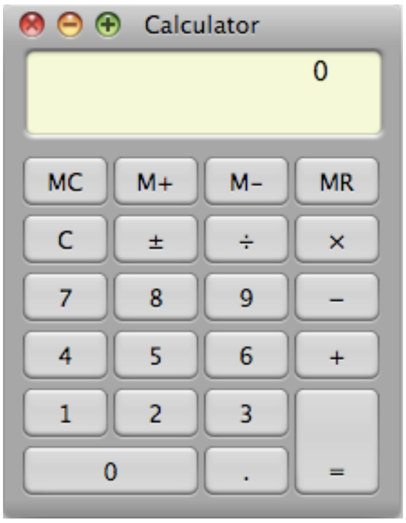
- AsipIDE connects existing tools together to form a (co-)simulation design flow
  - Iterative design methodology
    - Transforming software...
    - …to hardware
  - Hardware-software-FPGA co-simulation
  - Automatically generated GALS communications
  - Graphical debugging
    - Multiple abstractions represented together
    - Animation from simulation traces

General presentation of tools and IDE

- Calculator demo
  - Demonstrates iterative GALS prototyping
- Baseband processor&G3card demo
  - Demonstrates scalable environment
  - Navigation in large embedded system
- Features demo

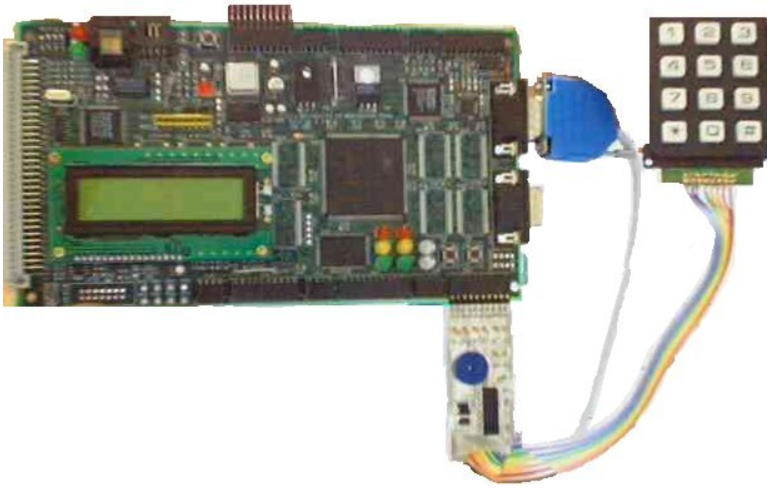
Calculator Demo

- **Calculator**:
  - Keyboard
  - LCD
  - Main program
    - polling the keyboard
    - processing
    - sending value to LCD
  - Keyboard asynchronous interface
  - LCD asynchronous interface

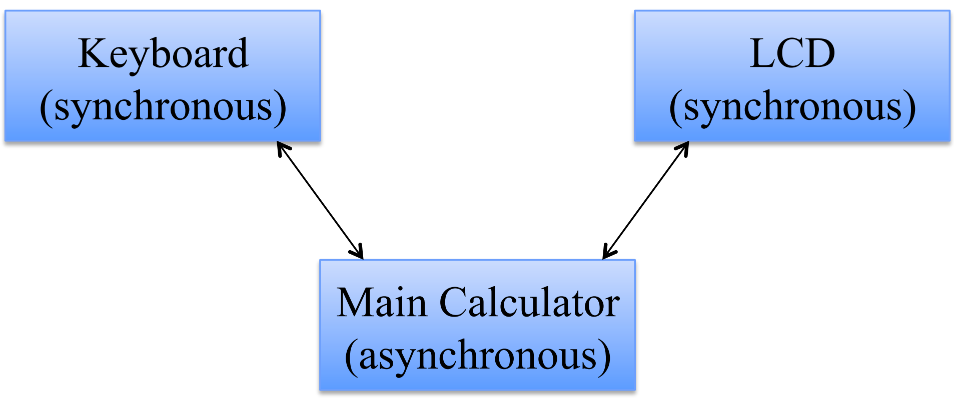


Calculator Demo

- High level architecture in C/C++
- Step-by-step implementation on hardware
- FPGA board prototyping



Calculator Demo

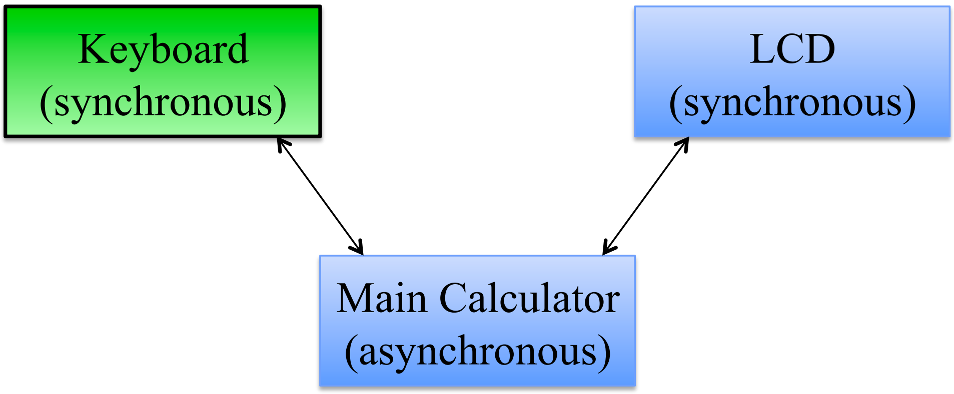


Step 1. All in Software

Software simulation

Hardware

Calculator Demo

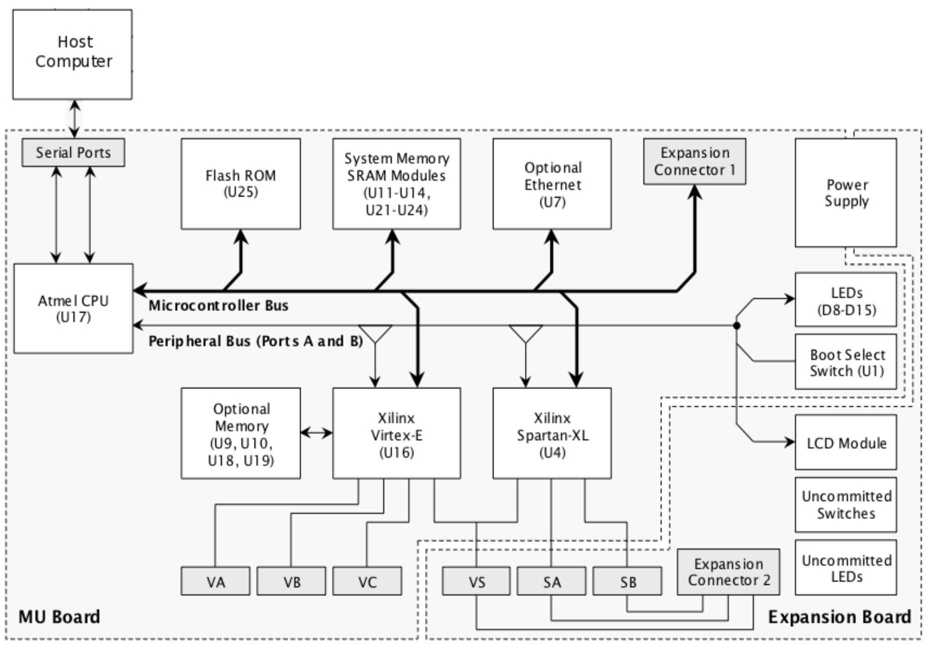


Step 2. Keyboard in Hardware

Software simulation

Hardware

Calculator Demo: FPGA board

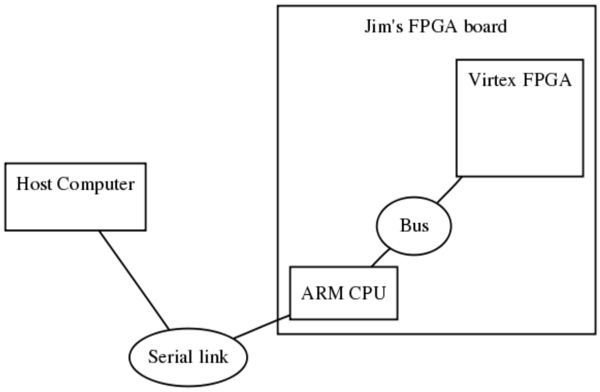


The University of Manchester

MANCHESTER 1824

Calculator Demo: FPGA board

- Simplified view of board for Galaxy tools

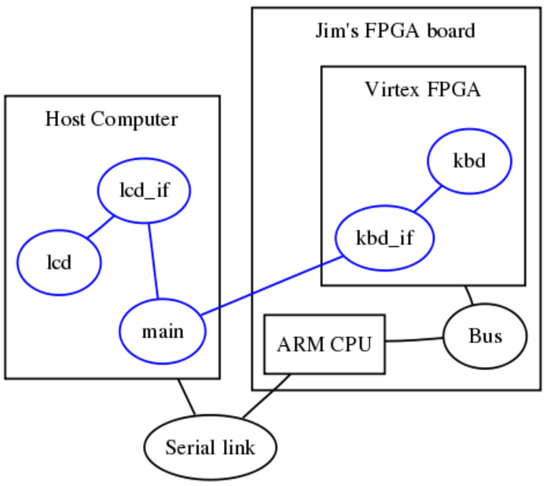


The University of Manchester

MANCHESTER 1824

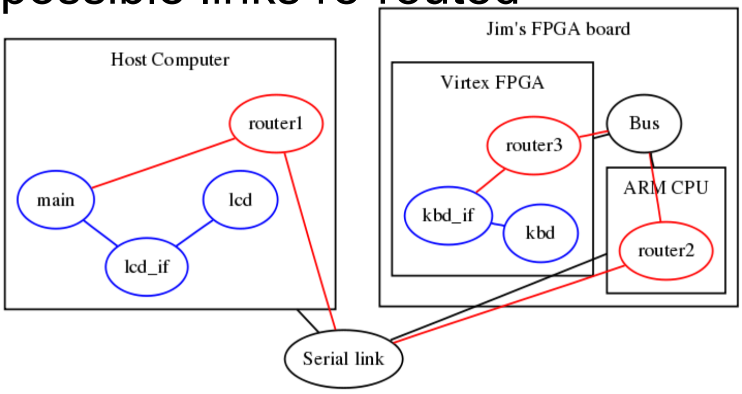
Calculator Demo: FPGA board

- Simplified view of board
- Importing components to targets



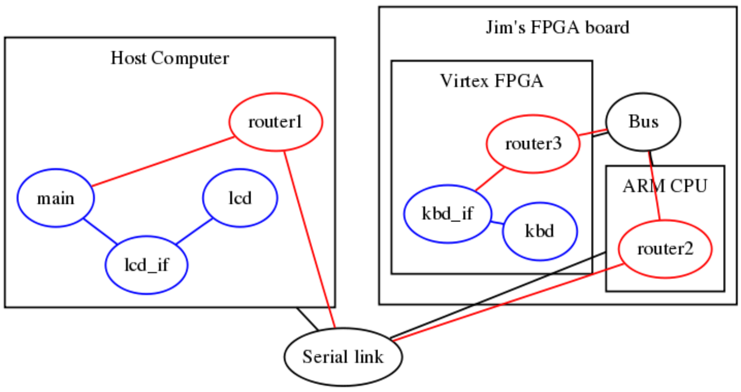
Calculator Demo: FPGA board

- Simplified view of board
- Importing components to targets
- Links Analysis
  - Impossible links re-routed



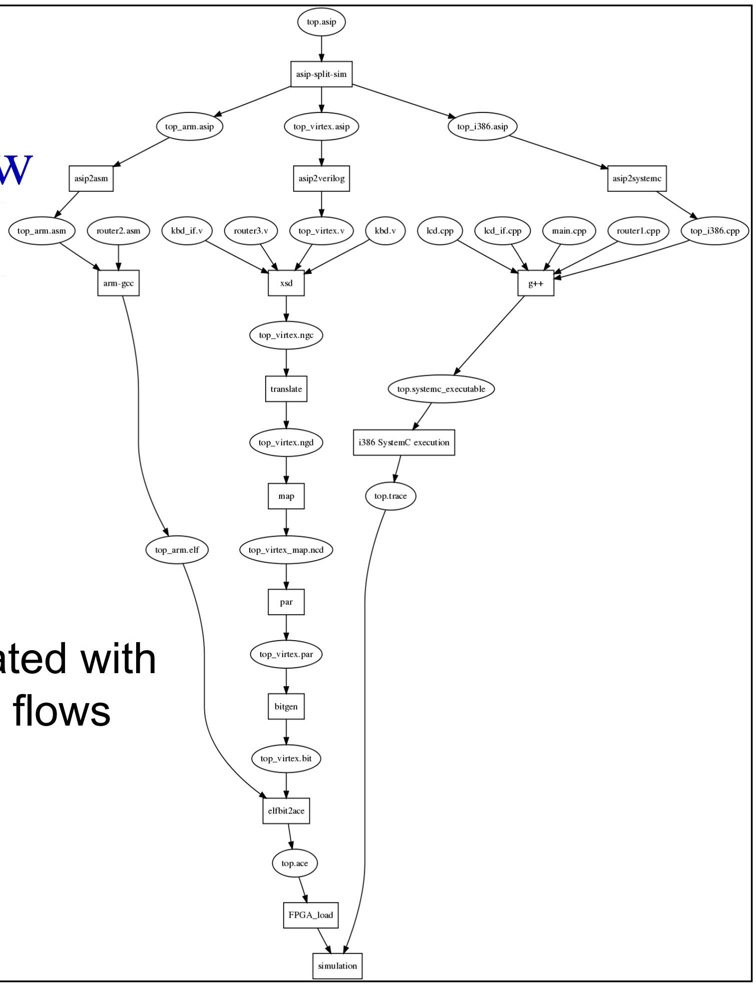
Calculator Demo: Hw-Sw Sync-Async Cosimulation

- Main execution on host
- SystemC transaction *"poll keyboard"*
  - Sent to *router1*
  - Routed through *router2* and *router3*
  - Converted to hardware asynchronous channel transaction
- Verilog *keyboard_if*:
  - Synchronous implementation
  - Asynchronous interface



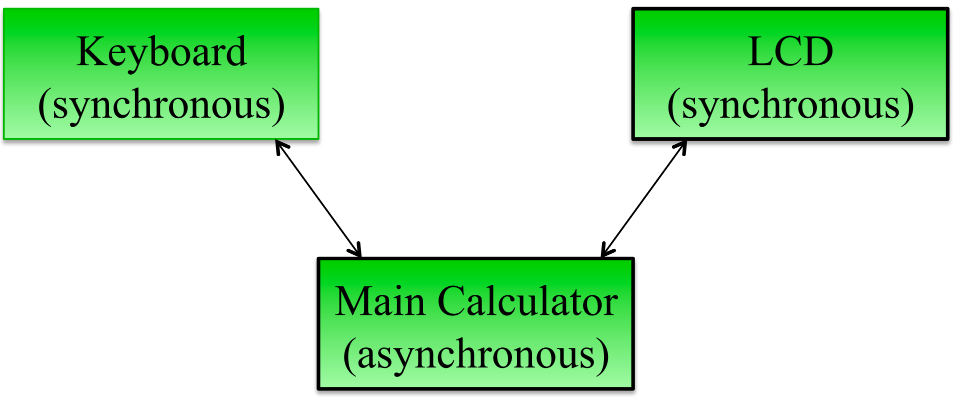
External Tool Flow

Xilinx tool flow integrated with SystemC and C++ flows



MANCHESTER 1824
The University of Manchester

Calculator Demo



Step 3. All in Hardware

Software simulation

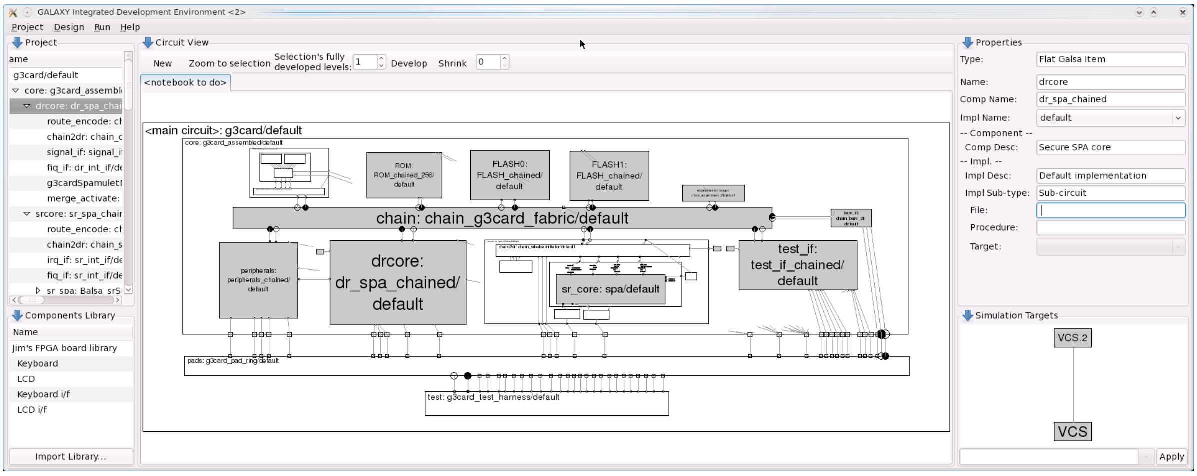
Hardware

MANCHESTER 1824

The University of Manchester

Baseband processor&G3card Demo

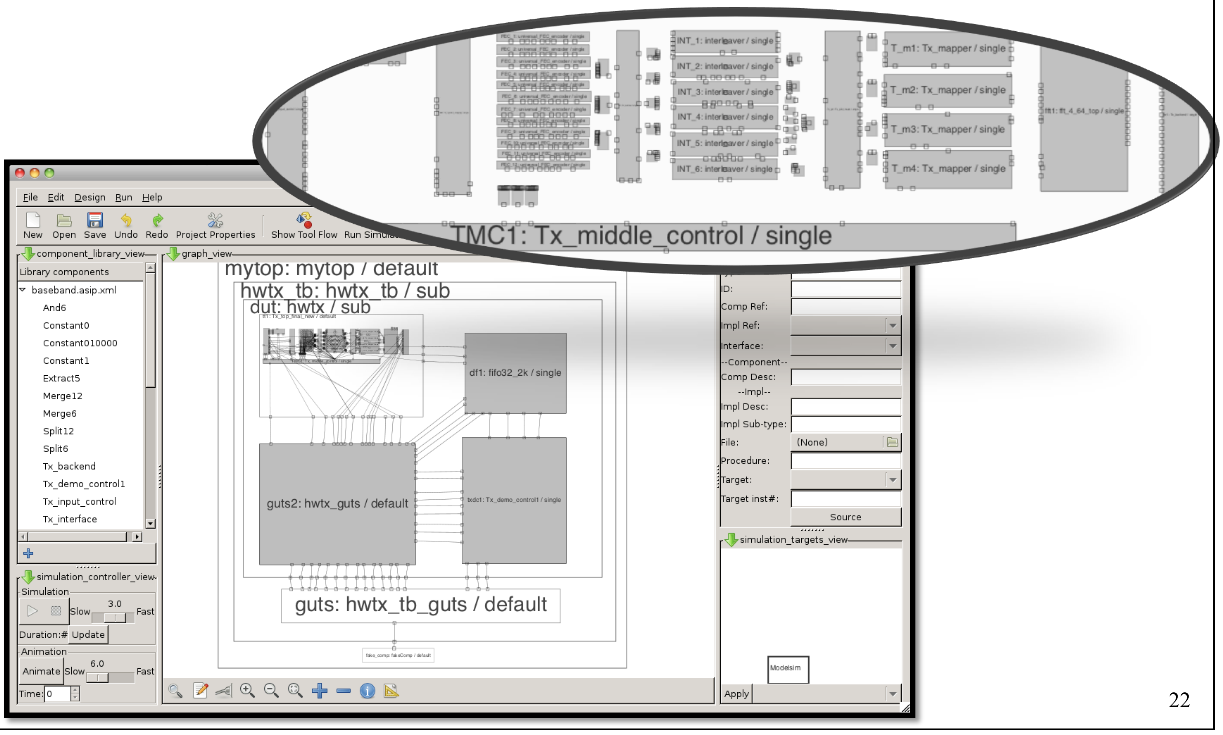
- Illustrating scalability
  - Zoom&pan inside large designs



MANCHESTER 1824

The University of Manchester

Baseband processor&G3card Demo



MANCHESTER 1824

The University of Manchester

Features Demo

- Automatic instantiation of adapters
- Automatic use of transactors
- Easy to switch components between multiple levels of abstractions, with always a proper interface regenerated
- Selection of any simulators or FPGA target
  - Ability to use asynchronous-specific simulators: Balsa, Petri nets
- Automatic use of local and remote tools for compilation, synthesis and simulation flow; remote resource sharing (queues)
- Trace file animation, debugging
  - Colour-based channel representation, clearer and saving space
  - Asynchronous debugging such as deadlock detection
- Asynchronous NoC
  - XPipes: graphical updates ➔ regenerates everything automatically

MANCHESTER 1824

The University of Manchester

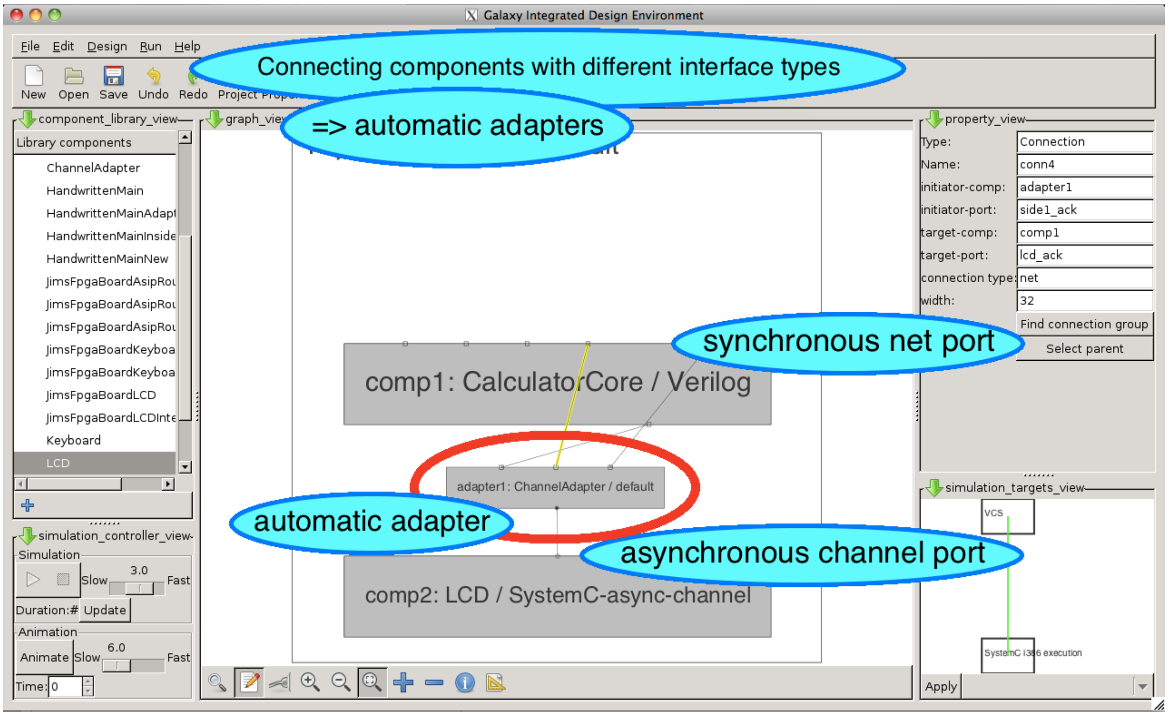
Features Demo

- **Automatic instantiation of adapters**
- Automatic use of transactors
- Easy to switch components between multiple levels of abstractions, with always a proper interface regenerated
- Selection of any simulators or FPGA target
  - Ability to use asynchronous-specific simulators: Balsa, Petri nets
- Automatic use of local and remote tools for compilation, synthesis and simulation flow; remote resource sharing (queues)
- Trace file animation, debugging
  - Colour-based channel representation, clearer and saving space
  - Asynchronous debugging such as deadlock detection
- Asynchronous NoC
  - XPipes: graphical updates ➔ regenerates everything automatically

MANCHESTER 1824

The University of Manchester

Features Demo: Automatic instantiation of adapters



MANCHESTER 1824

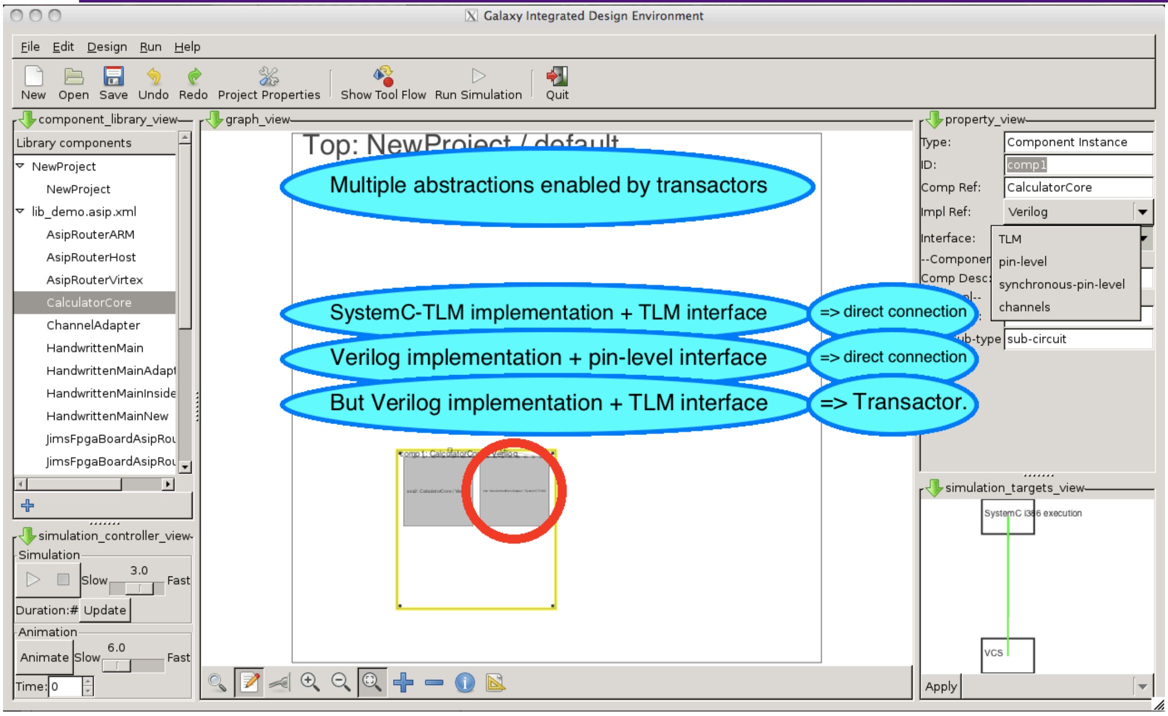
The University of Manchester

Features Demo

- Automatic instantiation of adapters
- **Automatic use of transactors**
- Easy to switch components between multiple levels of abstractions, with always a proper interface regenerated
- Selection of any simulators or FPGA target
  - Ability to use asynchronous-specific simulators: Balsa, Petri nets
- Automatic use of local and remote tools for compilation, synthesis and simulation flow; remote resource sharing (queues)
- Trace file animation, debugging
  - Colour-based channel representation, clearer and saving space
  - Asynchronous debugging such as deadlock detection
- Asynchronous NoC
  - XPipes: graphical updates ➔ regenerates everything automatically

The University of Manchester

Features Demo: Automatic use of transactors



The University of Manchester

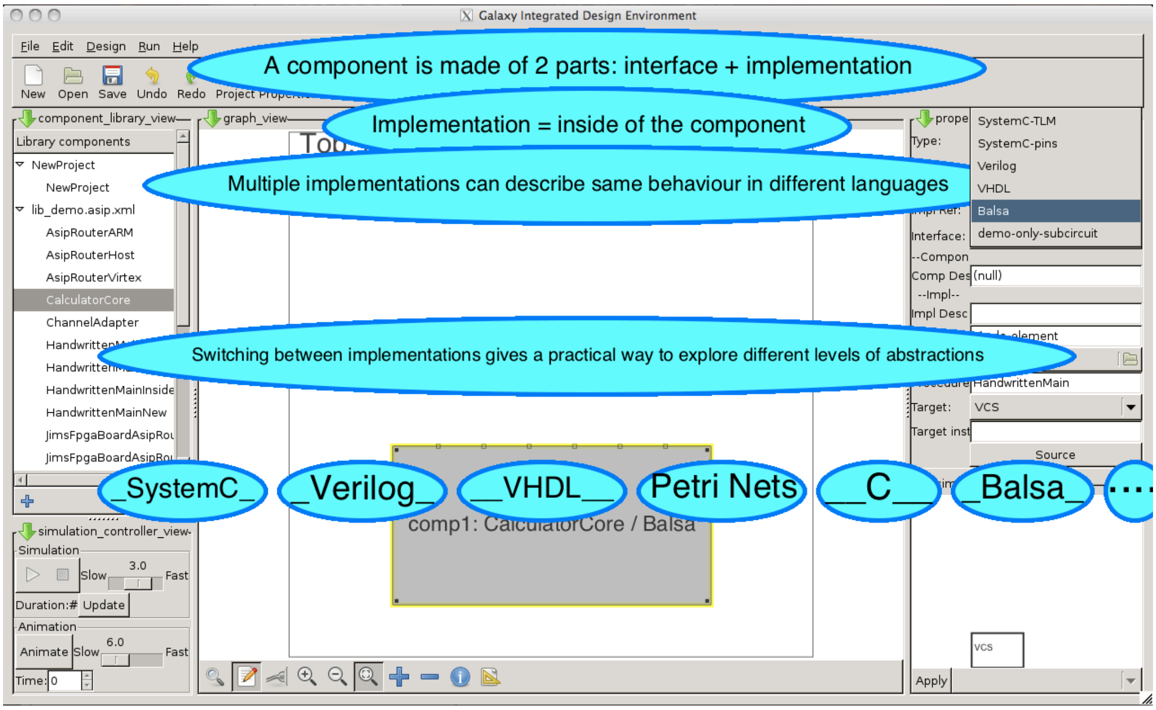
Features Demo

- Automatic instantiation of adapters
- Automatic use of transactors
- **Easy to switch components between multiple levels of abstractions, with always a proper interface regenerated**
- Selection of any simulators or FPGA target
  - Ability to use asynchronous-specific simulators: Balsa, Petri nets
- Automatic use of local and remote tools for compilation, synthesis and simulation flow; remote resource sharing (queues)
- Trace file animation, debugging
  - Colour-based channel representation, clearer and saving space
  - Asynchronous debugging such as deadlock detection
- Asynchronous NoC
  - XPipes: graphical updates ➔ regenerates everything automatically

MANCHESTER 1824

The University of Manchester

Features Demo: Switching between abstractions



MANCHESTER 1824

The University of Manchester

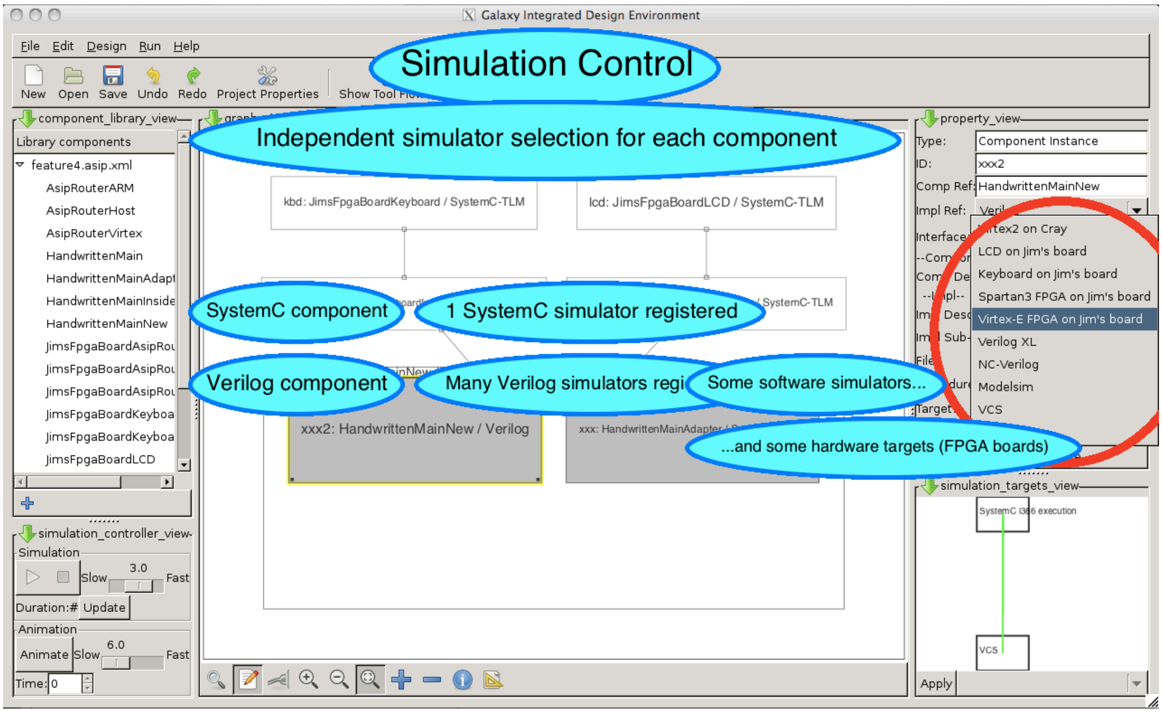
Features Demo

- Automatic instantiation of adapters
- Automatic use of transactors
- Easy to switch components between multiple levels of abstractions, with always a proper interface regenerated
- **Selection of any simulators or FPGA target**
  - **Ability to use asynchronous-specific simulators: Balsa, Petri nets**
- Automatic use of local and remote tools for compilation, synthesis and simulation flow; remote resource sharing (queues)
- Trace file animation, debugging
  - Colour-based channel representation, clearer and saving space
  - Asynchronous debugging such as deadlock detection
- Asynchronous NoC
  - XPipes: graphical updates ➔ regenerates everything automatically

MANCHESTER 1824

The University of Manchester

Features Demo: Simulator/FPGA selection



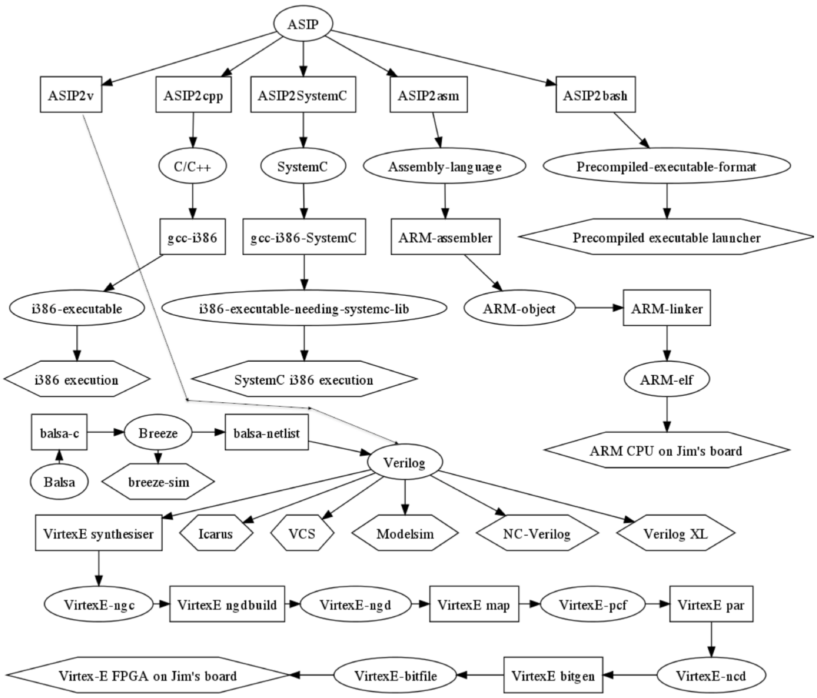
MANCHESTER 1824

The University of Manchester

Features Demo

- Automatic instantiation of adapters
- Automatic use of transactors
- Easy to switch components between multiple levels of abstractions, with always a proper interface regenerated
- Selection of any simulators or FPGA target
  - Ability to use asynchronous-specific simulators: Balsa, Petri nets
- **Automatic use of local and remote tools for compilation, synthesis and simulation flow; remote resource sharing (queues)**
- Trace file animation, debugging
  - Colour-based channel representation, clearer and saving space
  - Asynchronous debugging such as deadlock detection
- Asynchronous NoC
  - XPipes: graphical updates ➔ regenerates everything automatically

Features Demo: Tool flows



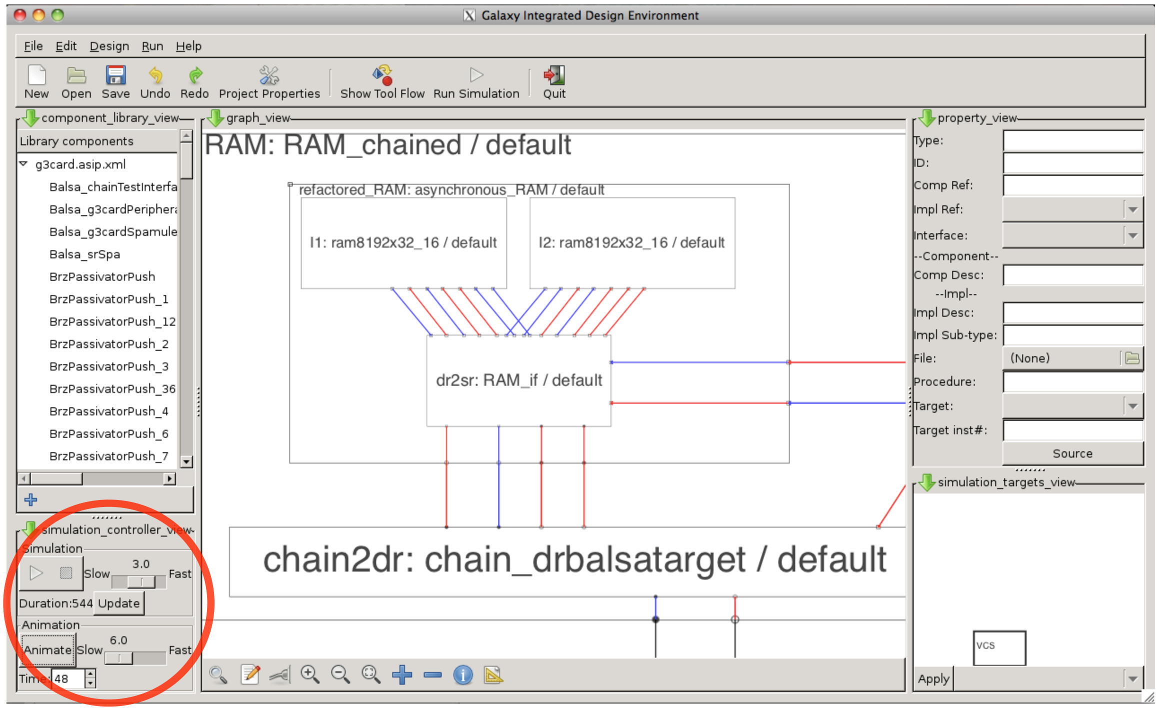
Features Demo

- Automatic instantiation of adapters
- Automatic use of transactors
- Easy to switch components between multiple levels of abstractions, with always a proper interface regenerated
- Selection of any simulators or FPGA target
  - Ability to use asynchronous-specific simulators: Balsa, Petri nets
- Automatic use of local and remote tools for compilation, synthesis and simulation flow; remote resource sharing (queues)
- **Trace file animation, debugging**
  - **Colour-based channel representation, clearer and saving space**
  - **Asynchronous debugging such as deadlock detection**
- Asynchronous NoC
  - XPipes: graphical updates ➔ regenerates everything automatically

MANCHESTER 1824

The University of Manchester

Features Demo: Trace file animation for debugging



MANCHESTER 1824

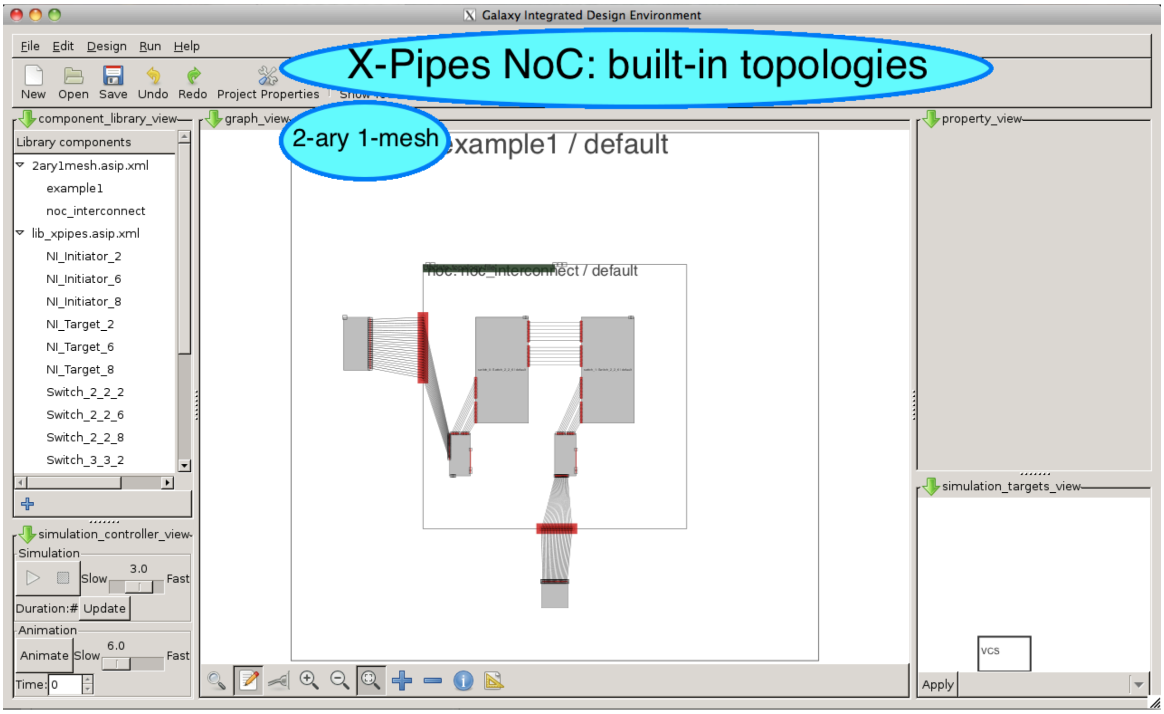
The University of Manchester

Features Demo

- Automatic instantiation of adapters
- Automatic use of transactors
- Easy to switch components between multiple levels of abstractions, with always a proper interface regenerated
- Selection of any simulators or FPGA target
  - Ability to use asynchronous-specific simulators: Balsa, Petri nets
- Automatic use of local and remote tools for compilation, synthesis and simulation flow; remote resource sharing (queues)
- Trace file animation, debugging
  - Colour-based channel representation, clearer and saving space
  - Asynchronous debugging such as deadlock detection
- **Asynchronous NoC**
  - **XPipes: graphical updates ➔ regenerates everything automatically**

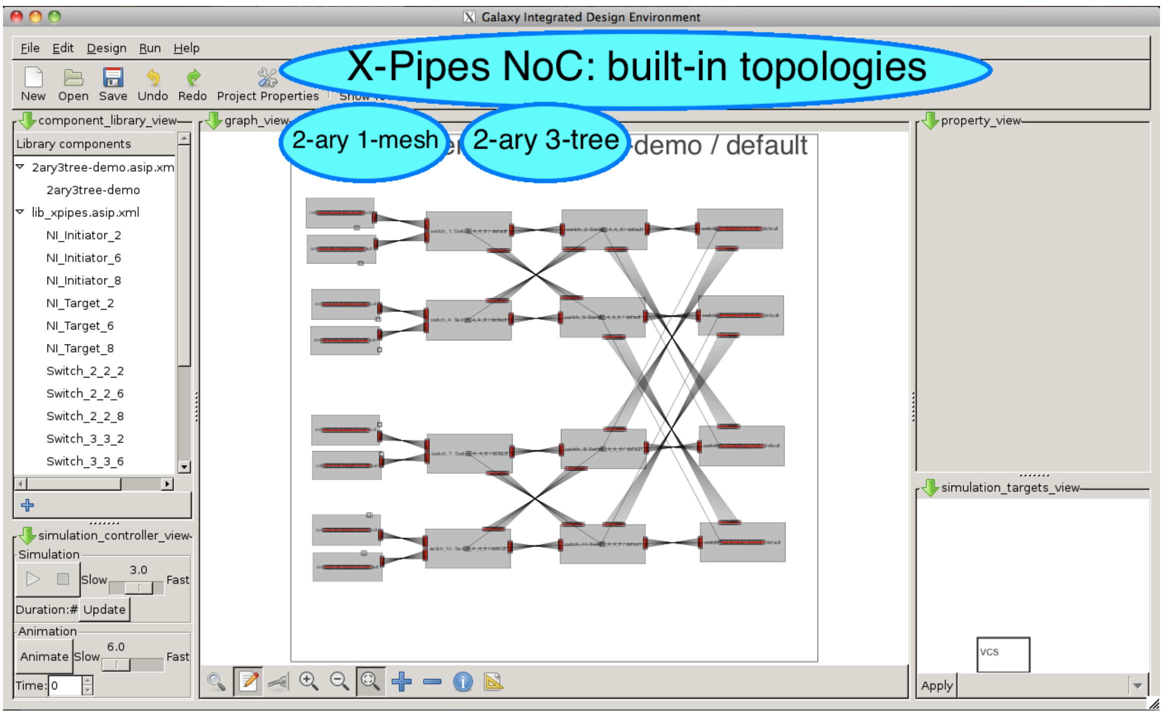
The University of Manchester

Features Demo: Asynchronous XPipes NoC

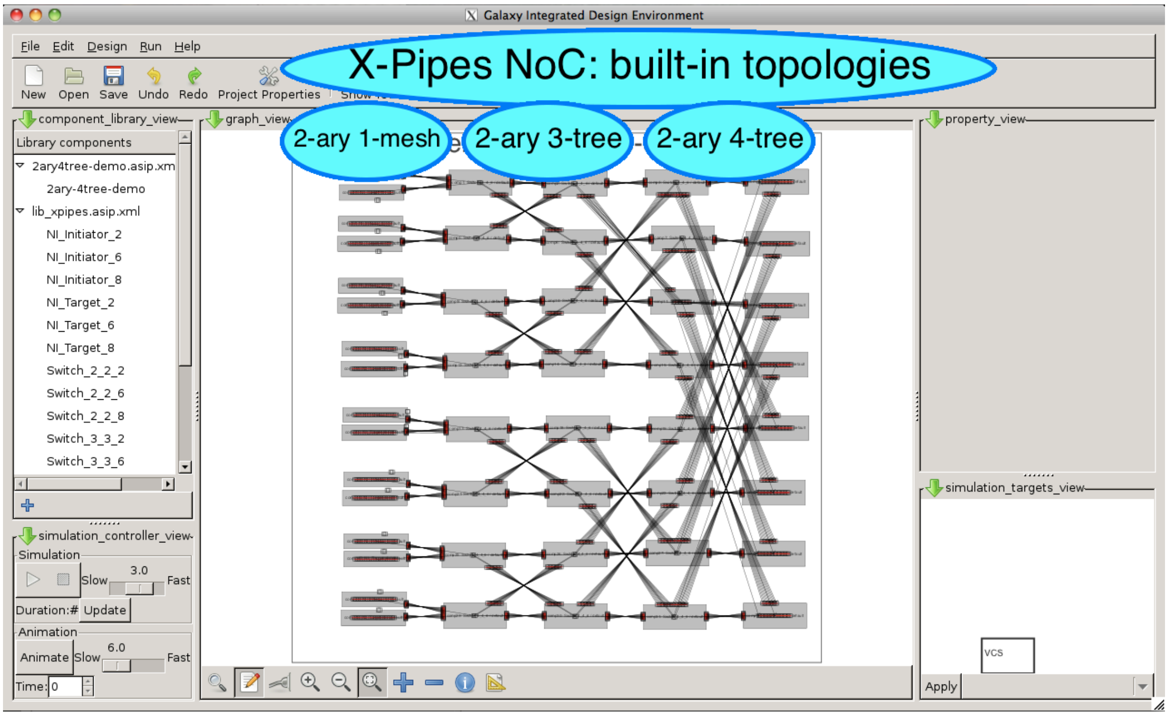


The University of Manchester

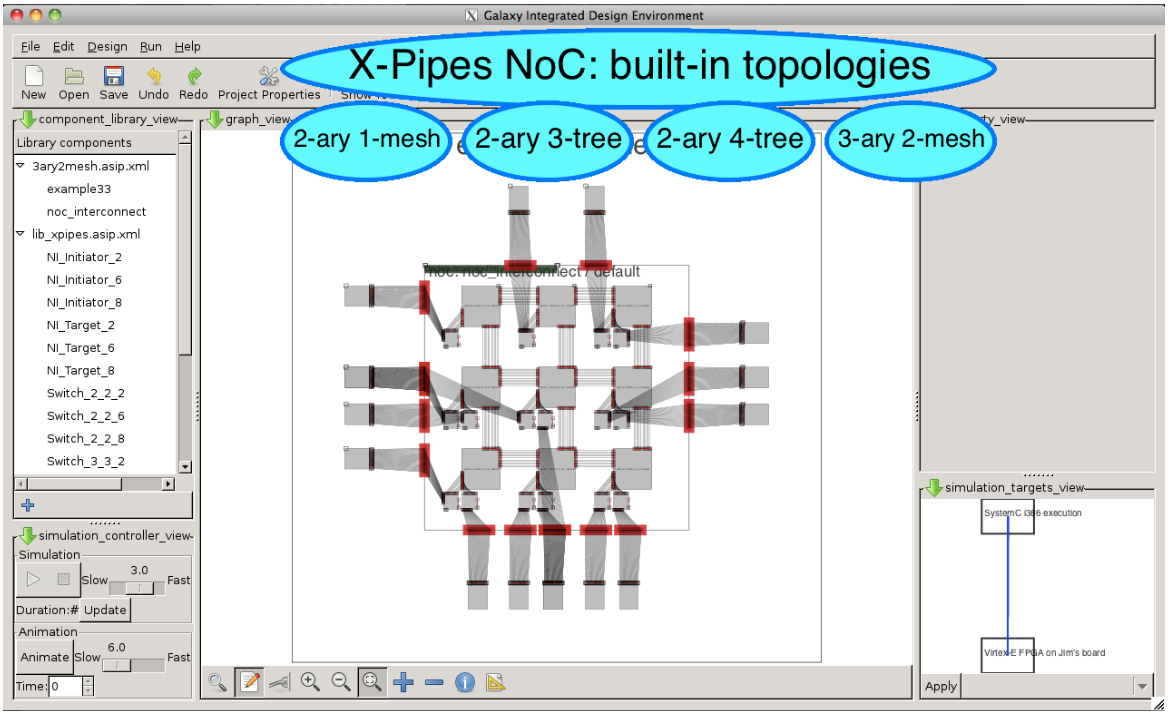
Features Demo: Asynchronous XPipes NoC



Features Demo: Asynchronous XPipes NoC



Features Demo: Asynchronous XPipes NoC



Hands-on: A home surveillance system

- Motivation
  - Typical application which home developers would like to use FPGAs for, but encounter design flow problems
  - Linux-based solutions available
    - USB webcam
    - Zoneminder analysis
    - Remote storage
    - – High purchase cost
    - – High consumption
  - FPGA cheaper final solution

Hands-on: A home surveillance system

- Video processing application
  - Webcam → motion detection → video encoding → ethernet streaming to remote server

- Outline
  - Requirements definition
  - Identification of re-usable open-source components
  - Creation of components, architecture exploration
  - Components assembly, automatic adapters
  - Automatic code generation, code running in SW
  - Iterative refinement of SW components to HW
  - Co-simulation software-FPGA

MANCHESTER 1824

The University of Manchester

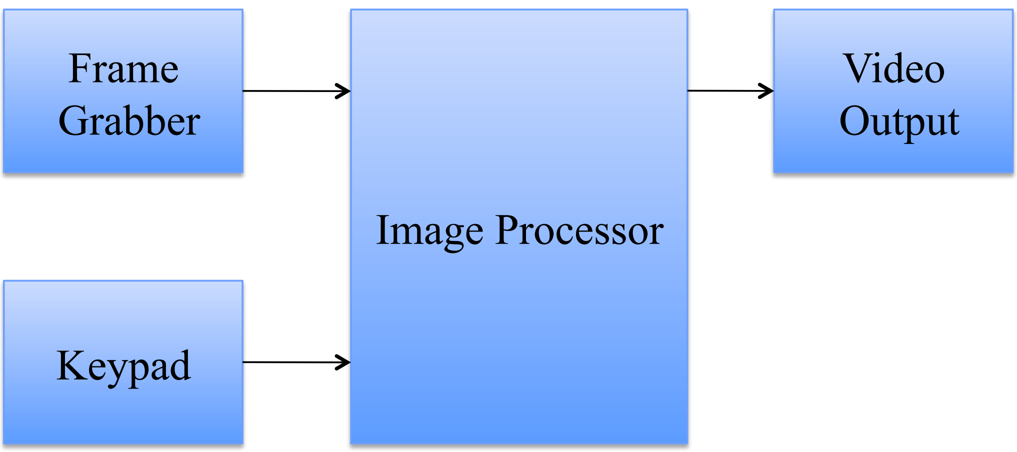
Requirements Definition

- Inputs: webcam + keypad
- Outputs:
  - Ethernet connection to send the motion-detected images/videos
  - Replaced by local VGA output for the demo
- Movement is detected by subtracting 2 consecutive frames
- Changes in input frame pixels start the recording
- Threshold set by keypad

MANCHESTER 1824

The University of Manchester

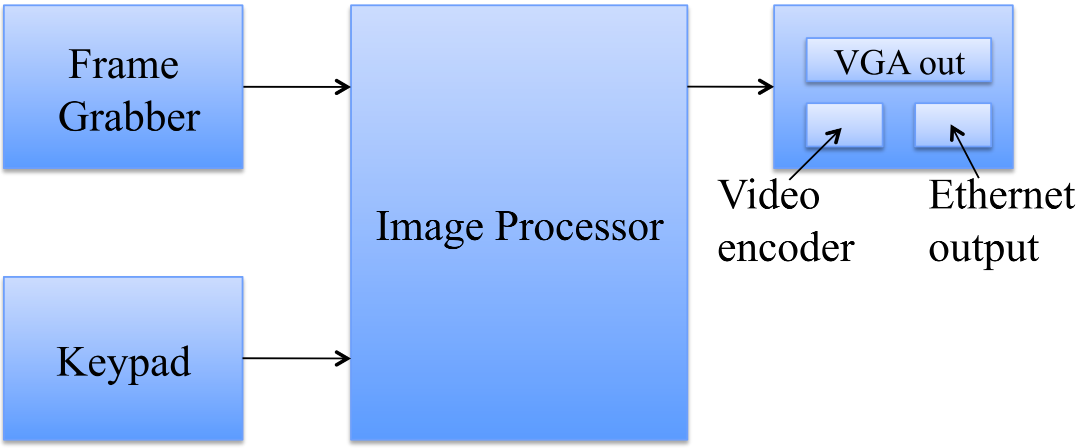
Block Diagram



The University of Manchester

MANCHESTER 1824

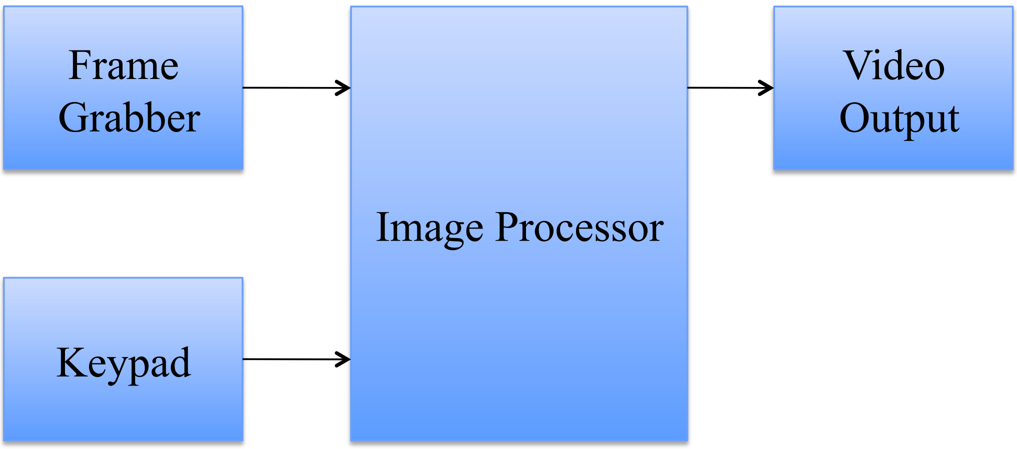
Block Diagram



The University of Manchester

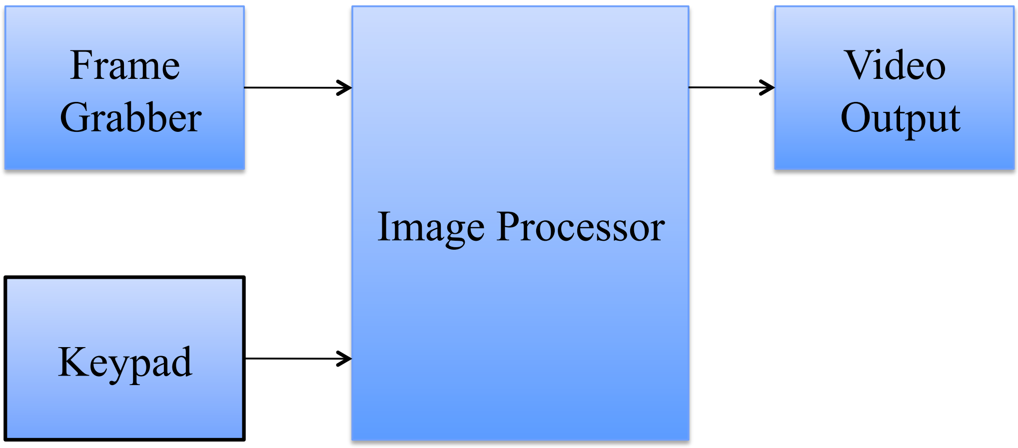
MANCHESTER 1824

Demo Contents



1. All simulated in software at TLM level

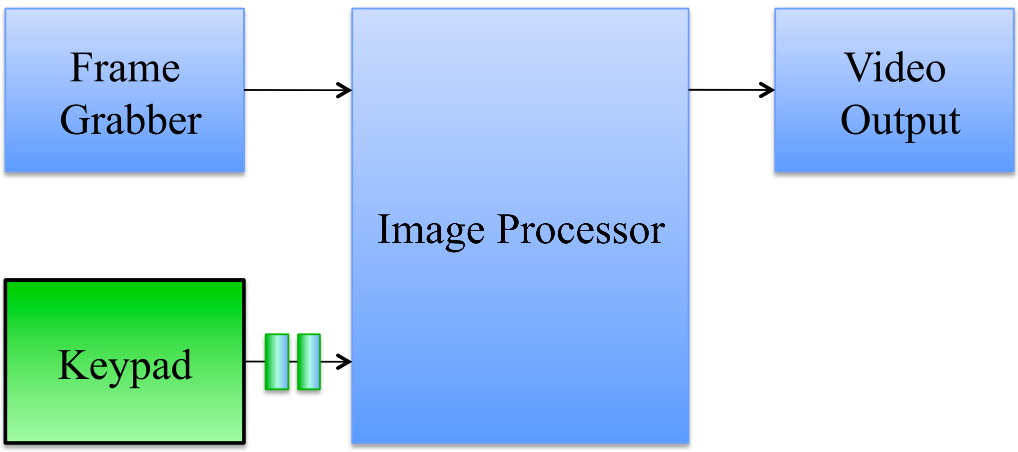
Demo Contents



2. Keypad refined to Verilog

Co-debugging SystemC TLM-Verilog

Demo Contents



3. Keypad moved to hardware

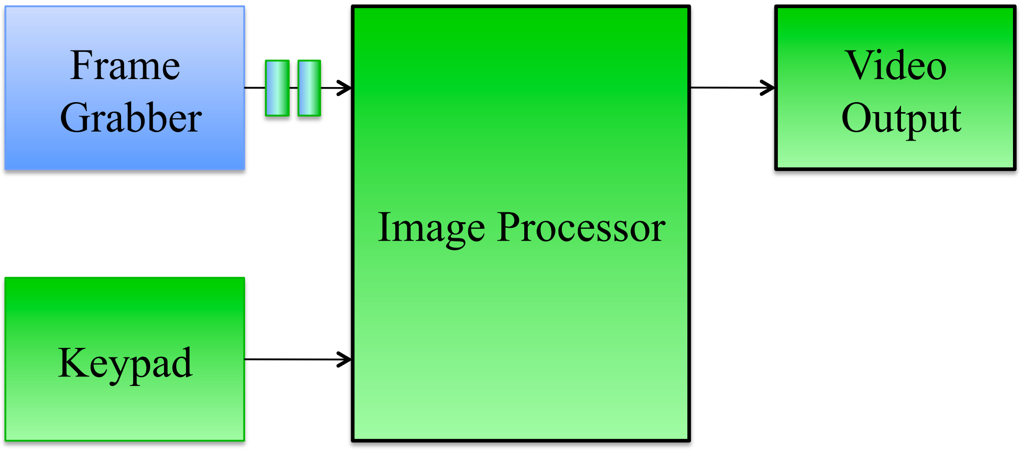
Transaction routing through board's CPU and FPGA

Co-debugging hardware-software

MANCHESTER 1824

The University of Manchester

Demo Contents



4. Processor and output moved to hardware

Based on re-usable open-source cores

MANCHESTER 1824

The University of Manchester

Identification of re-usable open-source components

Available from www.opencores.org

- Keypad scanner
- JPEG-MJPEG video encoder
- VGA/LCD controller
- Wishbone Memory wrapper

Start of Hands-on

- Initialise environment

```
source asipide_env_setup
```

- Create and enter your own directory

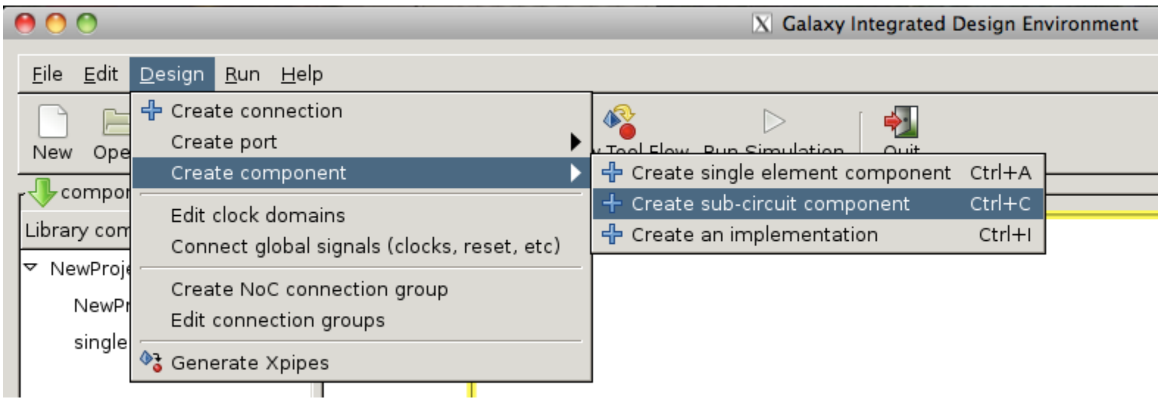
```
mkdir your_name
cd your_name
```

- Start IDE

```
asipide
```


Hands-on Step 1 - How to: Create a new component

- If it will contain sub-components
  - Select the parent component in the graphical design view
  - Design Menu → Create component → Create sub-circuit component
- If it will be a "leaf" component, referring to existing source code files
  - Select the parent component in the graphical design view
  - Design Menu → Create component → Create single element component



MANCHESTER 1824

The University of Manchester

Step 1 - How to: Create a new component

- After a component creation, the IDE enters "Edit mode"
  - Components can be moved
  - Components can be resized
- Next created component will inherit the same size
- You can edit the component and instance names in the Property View

MANCHESTER 1824

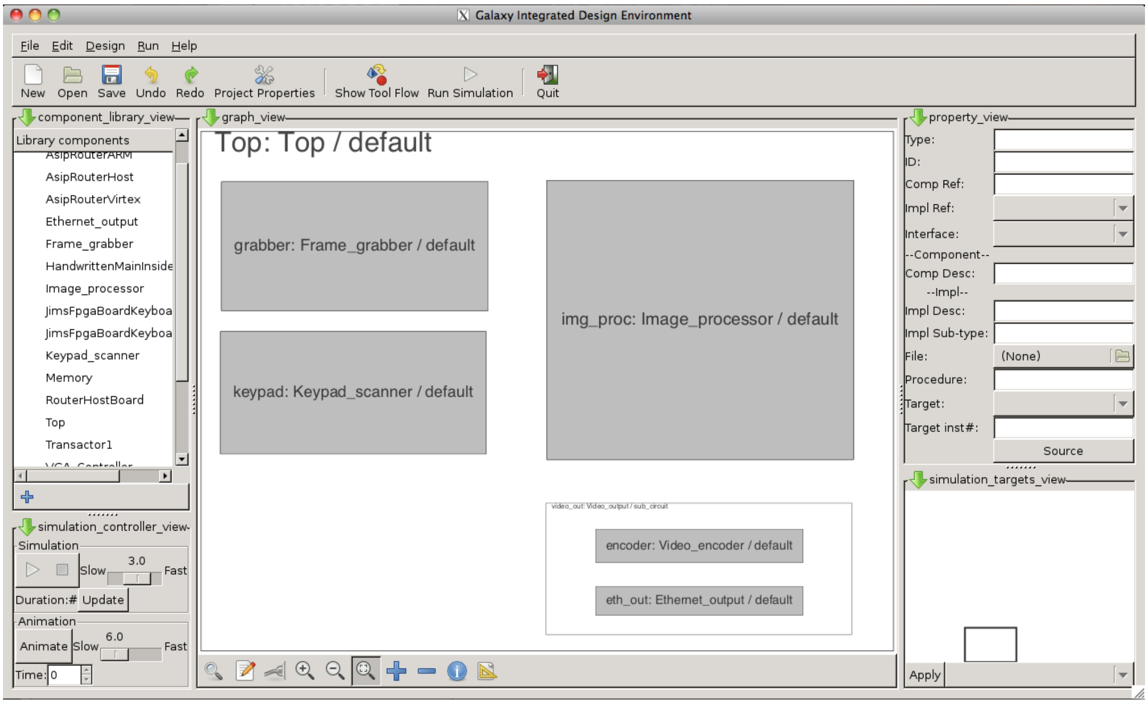
The University of Manchester

Step 1: Creation of component architecture

Create these 6 components:

- Image_processor
- Frame_grabber
- Keypad_scanner
- Video_output, with sub-components:
  - Video_encoder
  - Ethernet_output

Step 1 - Result: Created components



Components assembly

- We will now import and assemble the components together
- Often the hardware is not available at the start of a project. We need to do as much as possible using software and simulators.

➔ Version 1 (mostly to define the architecture and the main communication data types):

  - Frame_grabber component will take its input from files
  - Ethernet output will dump results to a file
  - Image processor will just subtract the new frame's pixels values from the previous frame's and check whether the max pixel change goes over a certain threshold
  - Video encoder will use free software encoders: ffmpeg/libavcodec

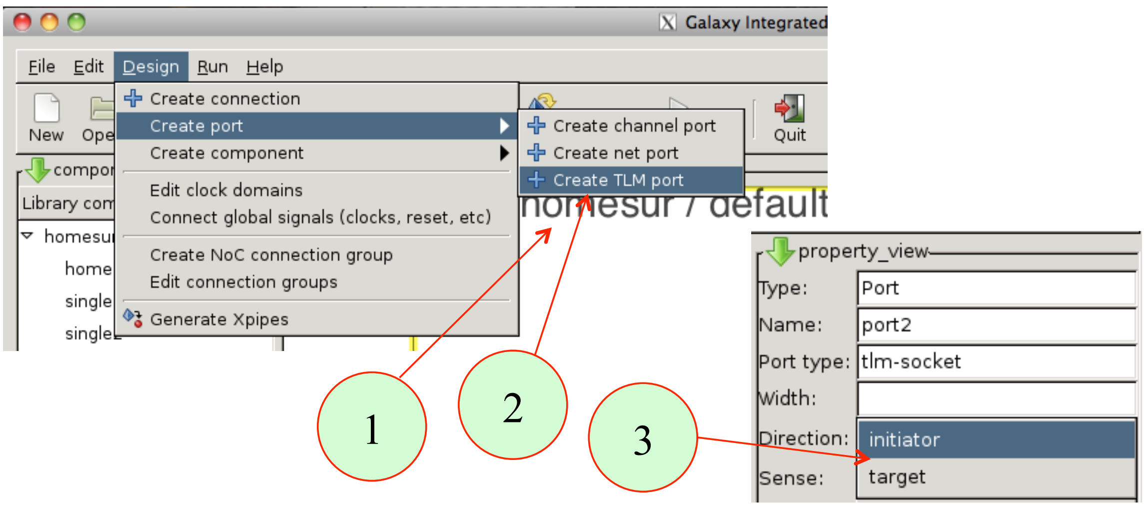
- Communications will use TLM, the highest level of abstraction integrated in AsipIDE.

Definition of ASIP communication types

- The *image processor* will be the main module (initiating requests)
  - Initiates requests to *frame grabber*
  - Sends probe requests to *keypad scanner*
  - Provides commands and data to *video output*
- Create TLM ports for each component
  - Select appropriately *target* or *initiator*
- Connect TLM ports

Hands-on Step 2 - How to: Create a component port

1. Select component in the graphical design view
2. Design Menu → Create port→ Create TLM port
3. In Property View: Select port direction



MANCHESTER 1824

The University of Manchester

Step 2: Creation of component ports

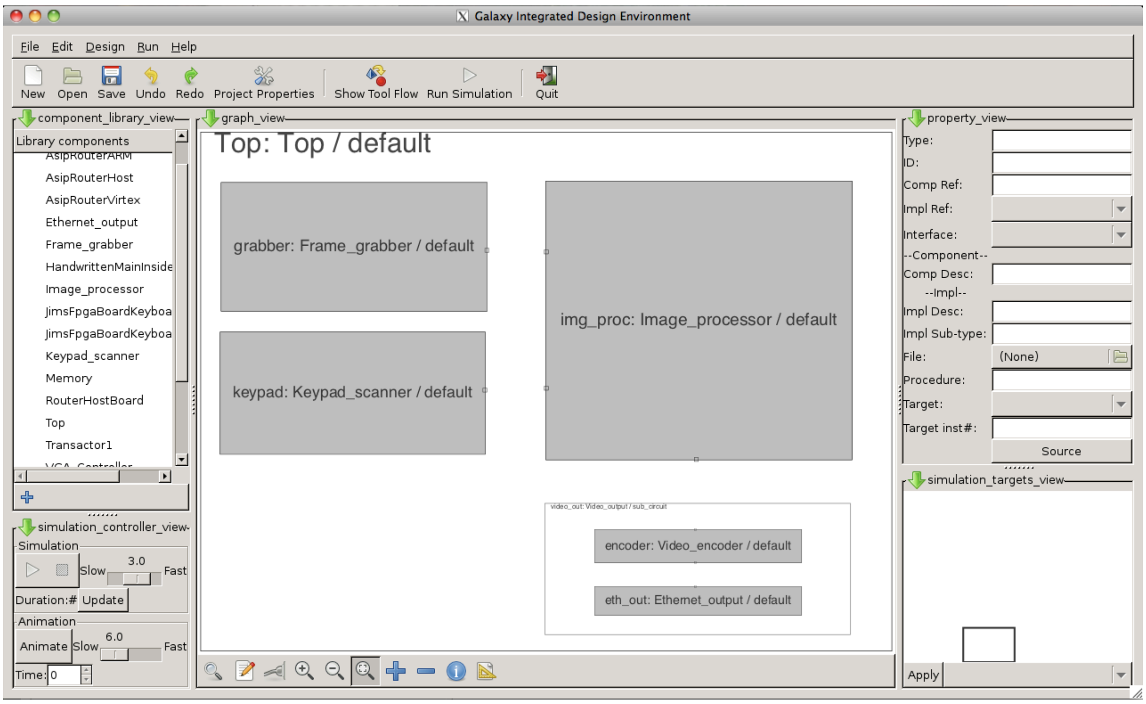
Create the following ports:

- Image processor
  - 3 TLM initiator ports
- Frame grabber
  - 1 TLM target port
- Keypad scanner
  - 1 TLM target port
- Video encoder
  - 1 TLM target port + 1 TLM initiator port
- Ethernet output
  - 1 TLM target port

MANCHESTER 1824

The University of Manchester

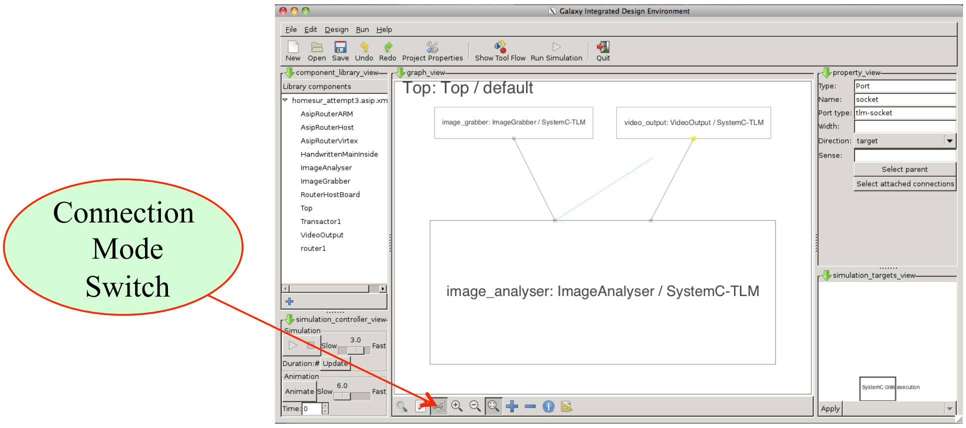
Step 2 - Result: Created ports



Hands-on Step 3 - How to: Connect two component ports

Several ways to do it, one being:

- Switch to “Connection mode” by clicking the first icon below the design view
- For each connection:
  - Move the mouse near port 1
    - It should get highlighted when you are close enough
  - Click and drag the mouse to port 2
  - Release
- Deactivate “Connection mode” by clicking the first icon below the design view



Step 3: Connecting components

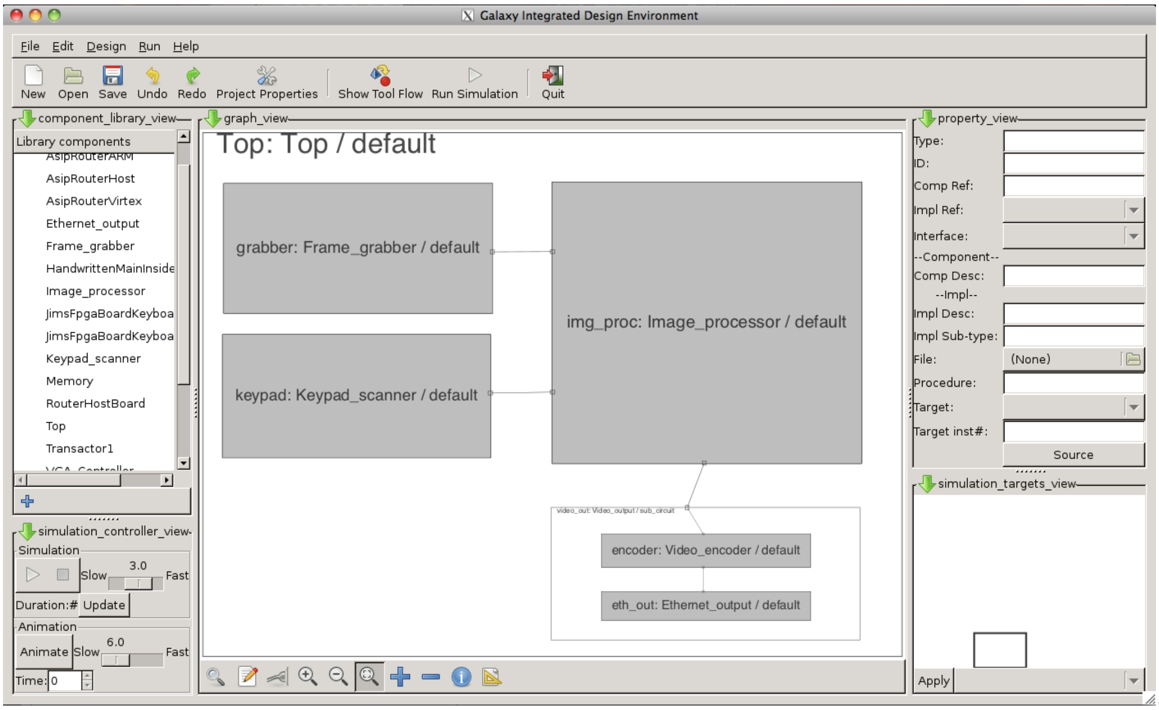
Create the following connections:

- Image processor → Frame grabber
- Image processor → Keypad scanner
- Video encoder → Ethernet output
- Image processor → Video encoder
  (Move the automatically created port to a better position)

MANCHESTER 1824

The University of Manchester

Step 3 - Result: Connected components

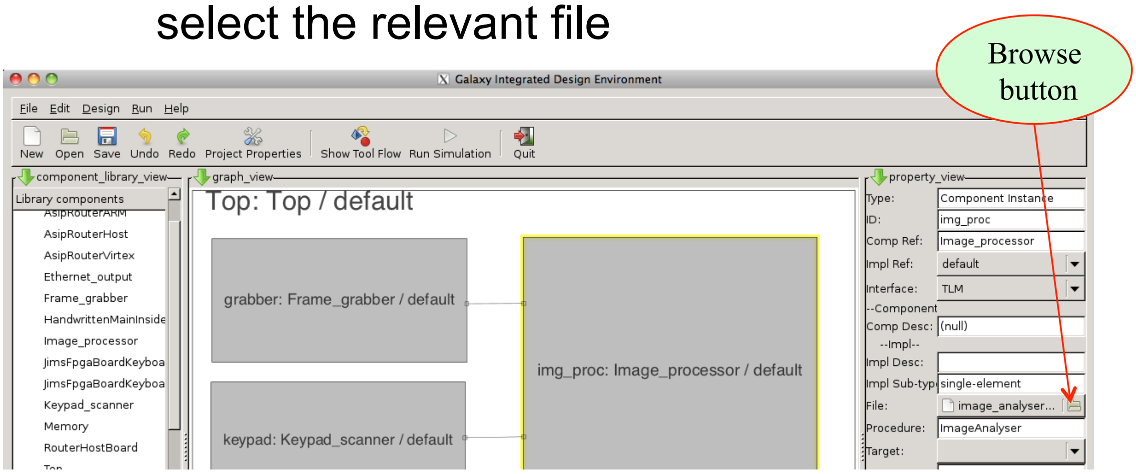


MANCHESTER 1824

The University of Manchester

Hands-on Step 4 - How to: Assign source code to a component

- Select the component
- In Property View
  - Use the File entry's "Browse…" button to select the relevant file



Step 4: Component implementations

- We prepared a SystemC implementation for each component
  - Available in directory *~/AsipIDE/SystemC/*
- Assign the some source code to each component
  - Image processor: *image_processor.cpp*
  - Frame grabber : *frame_grabber.cpp*
  - Keypad scanner: *keypad_scanner.cpp*
  - Video encoder: video_encoder*.cpp*
  - Ethernet output: *ethernet_output.cpp*

Hands-on Step 5: SystemC simulation

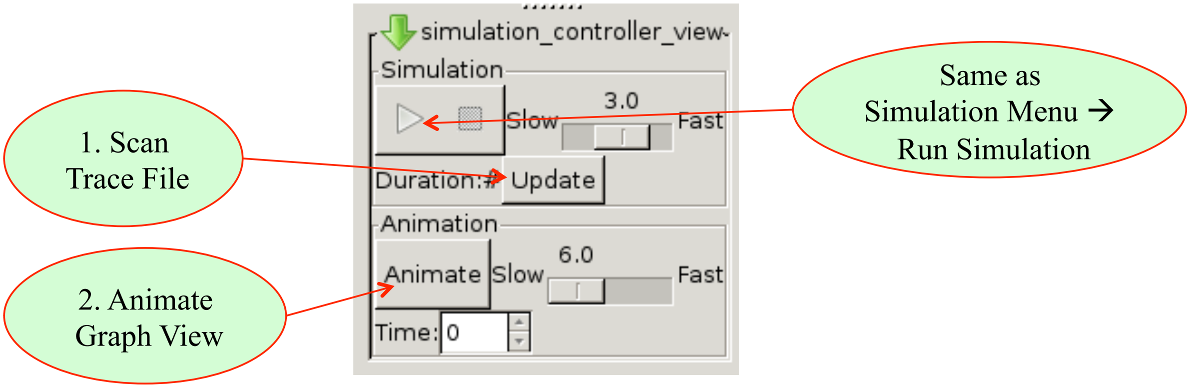
- Start simulation
  - Click Simulation Menu→Run Simulation
- Automatic generation of top-level SystemC code
- Reads input from directory *images*
- Streams output to file */tmp/asipide_tutorial.mpg*
  - Output can be played with mplayer

MANCHESTER 1824

The University of Manchester

Debugging: Design view animation

- Simulation trace observable in IDE
- Controlled via *Simulation controller view*



MANCHESTER 1824

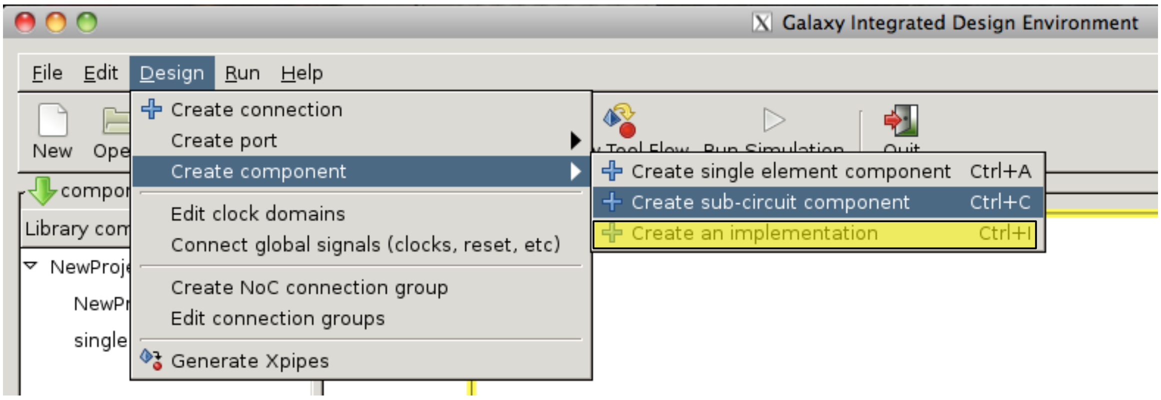
The University of Manchester

Connecting a real webcam

- Still in software
- New implementation of *frame grabber*
  - Access to webcam from the host computer
- SystemC source code provided

Hands-on Step 6 - How to: Create an extra implementation

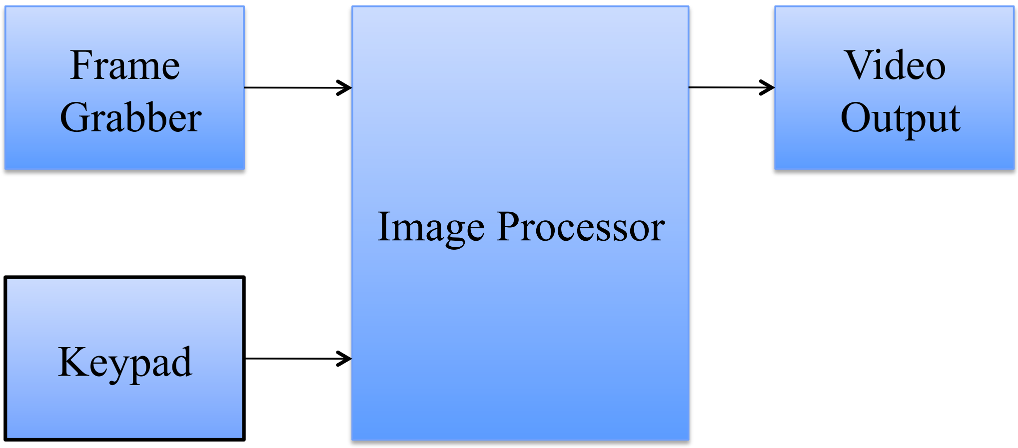
1. Select component in the graphical design view
2. Design Menu → Create component→ Create an implementation



Step 6: New component implementation

- Select the *frame_grabber* component
- Create a new implementation
  - Of type “single-element”
  - Attached to the source code *frame_grabber_webcam.cpp*
- Check that the new implementaion is selected
- Simulate

Demo Contents



2. Keypad refined to Verilog

Co-debugging SystemC TLM-Verilog

Iterative refinement to hardware

- With everything working at TLM level, we will slowly move components to Verilog and then to hardware
- TLM ports refined to pin level
- Starting with the keypad

Hands-on Step 7: Changing *keypad* to *keypad_v2*

1. Add library *demo_hardware_lib_1*
2. Drag&drop *keypad_v2* on top of *keypad_scanner*
3. A new dialog suggests how the connections from the old component can be transferred to the new component. Accept the suggested mapping.

MANCHESTER 1824
The University of Manchester

Hands-on Step 7: *keypad_v2* inspection

- Select *keypad_v2* for inspection
- Two implementations
  - Our first SystemC TLM implementation
  - A new Verilog implementation
- Two interfaces
  - TLM interface
  - Pin-level interface

MANCHESTER 1824

The University of Manchester

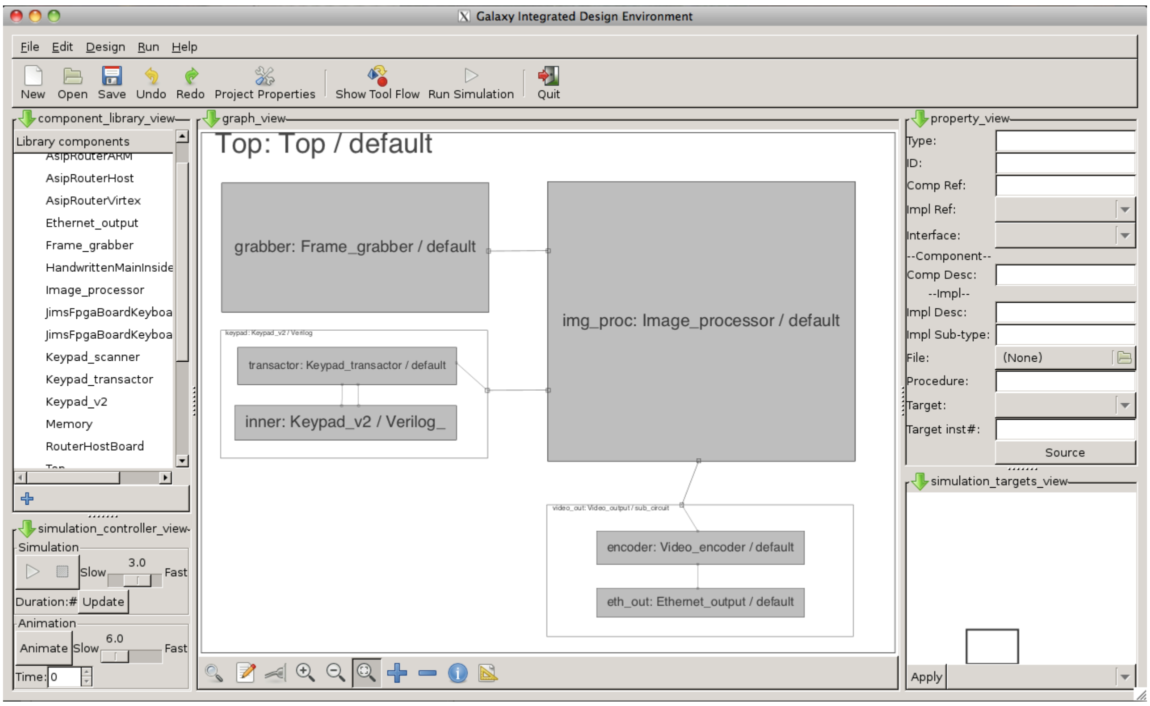
Hands-on Step 7: *keypad_v2* transactor

- Different combinations of implementation + interface possible
  - TLM impl. + TLM interface
    - as used until now
  - Verilog impl. + pins interface
    - No connections between *keypad_v2*'s pins and *image_processor*'s TLM ports
  - TLM impl. + pins interface
    - We haven't defined a transactor for this, as we don't plan to simulate using this configuration
  - Verilog impl. + TLM interface
    - Useful to us, as the TLM interface can connect to the *image_processor*'s TLM ports
    - Transactor automatically instantiated

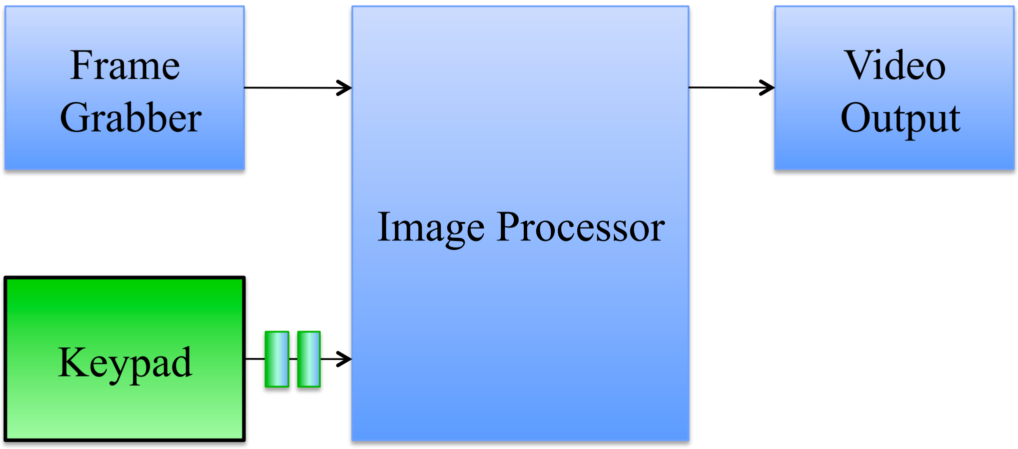
MANCHESTER 1824

The University of Manchester

Step 7 - Result: *keypad_v2* transactor



Demo Contents



3. Keypad moved to hardware

Transaction routing through board’s CPU and FPGA

Co-debugging hardware-software

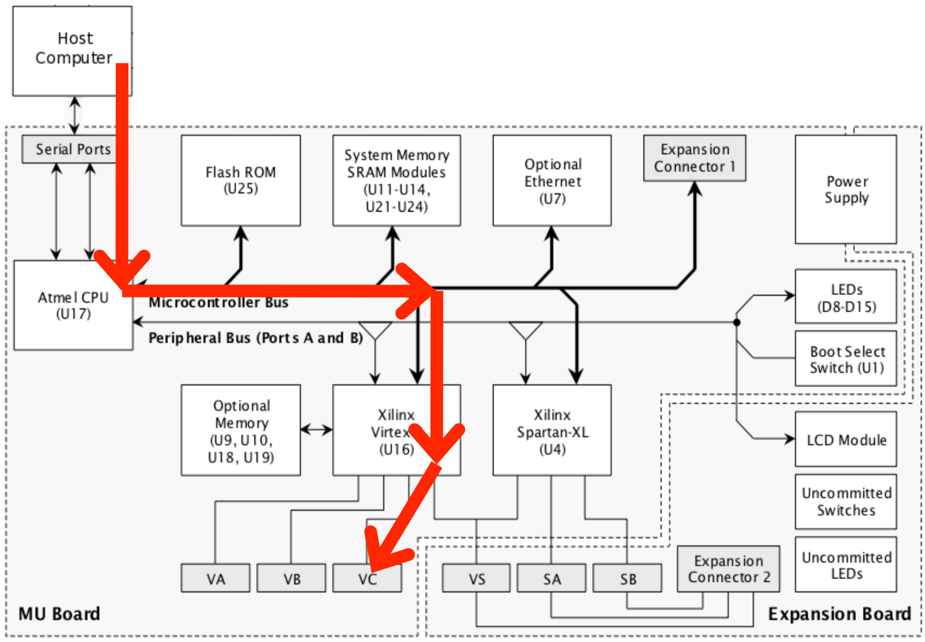
Targeting Hardware

- We want to use a real hardware keypad
- But keep the rest in simulators

Problem:

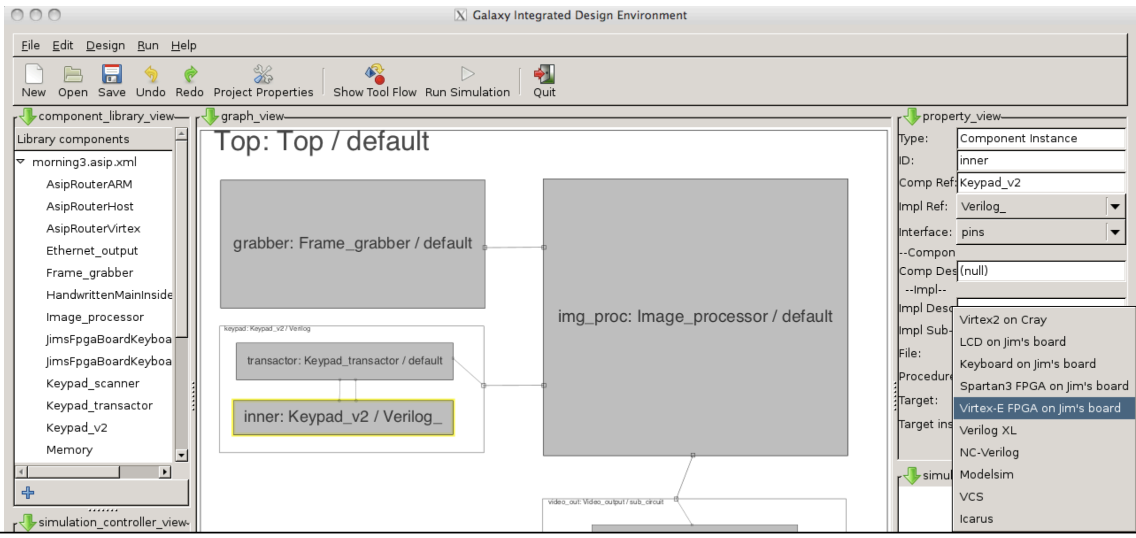
- No direct link between host and keypad
- Need to go through Host → Serial cable → ARM CPU → Bus → FPGA → Keypad

FPGA board



Hands-on Step 8: Targeting the FPGA board

- Select the inner component inside *keyboard_v2*
- In *Property View*, select the desired *Target*: "*Virtex FPGA on demo board*"

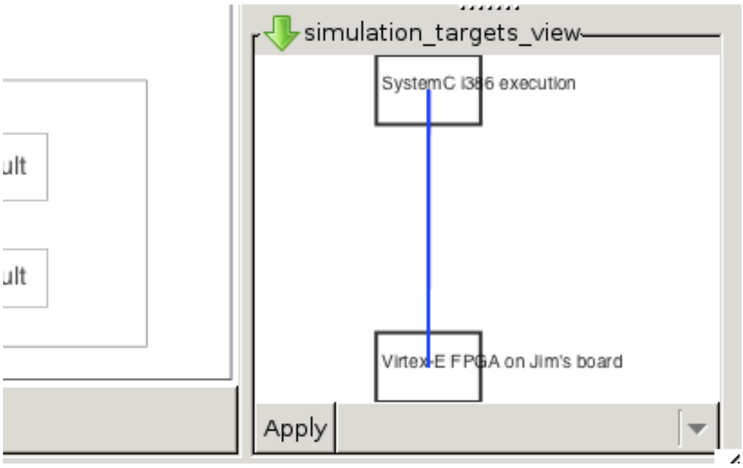


MANCHESTER 1824

The University of Manchester

Hands-on Step 8: Targeting the FPGA board

- The *Simulation targets view* detects that the 2 simulation targets cannot communicate directly (blue link)

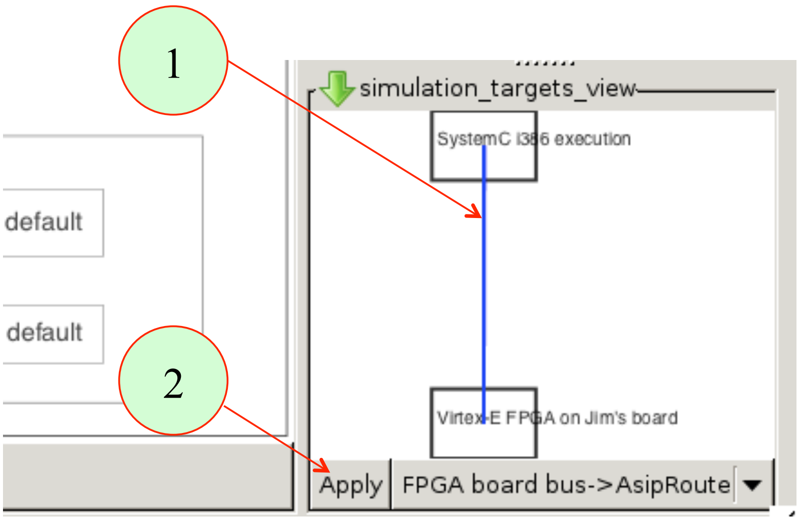


MANCHESTER 1824

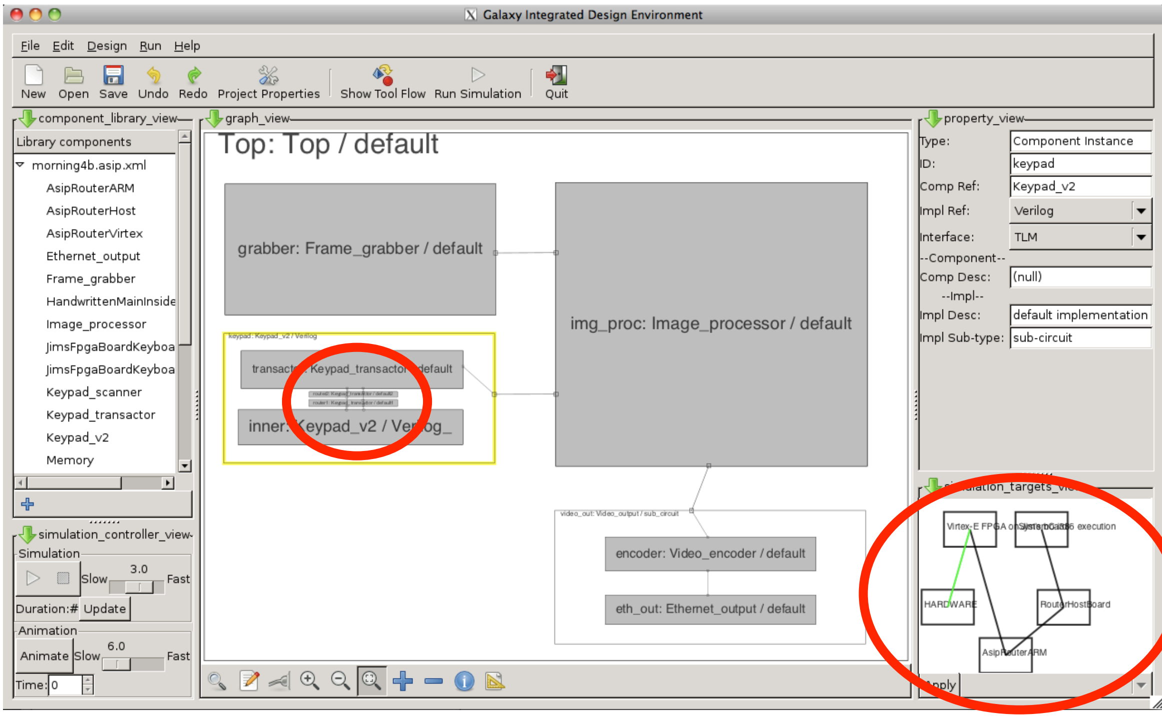
The University of Manchester

Hands-on Step 8: Targeting the FPGA board

1. Select the blue link
2. *Apply* the suggested ASIP routers



Step 8 - Results: Targeted FPGA board



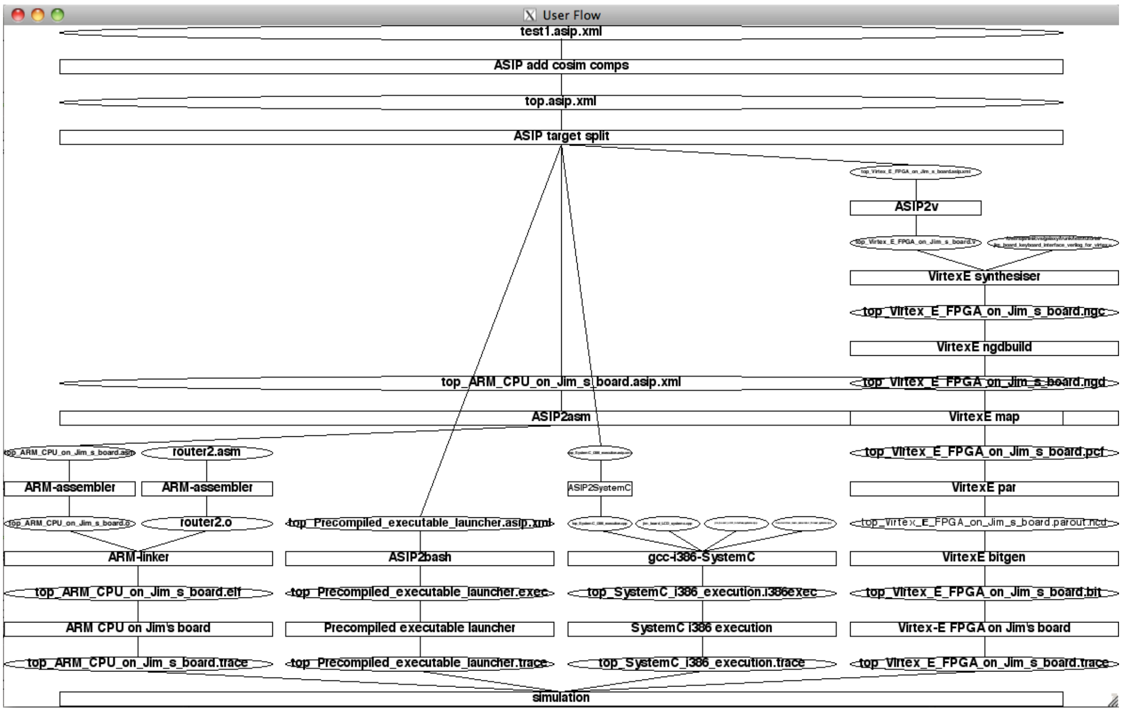
Hands-on Step 9: FPGA-software co-simulation

- Click on the icon “Show *Tool Flow*”
- Launching Simulation
  - Generates SystemC to FPGA board communications
  - Generates top-level code for each target
    - SystemC
    - Host to board controller (precompiled software)
    - Board’s ARM CPU
    - Board’s FPGA

MANCHESTER 1824

The University of Manchester

Step 9 - Results: Tool Flow



MANCHESTER 1824

The University of Manchester

Step 9 – Results: Hardware-software co-simulation

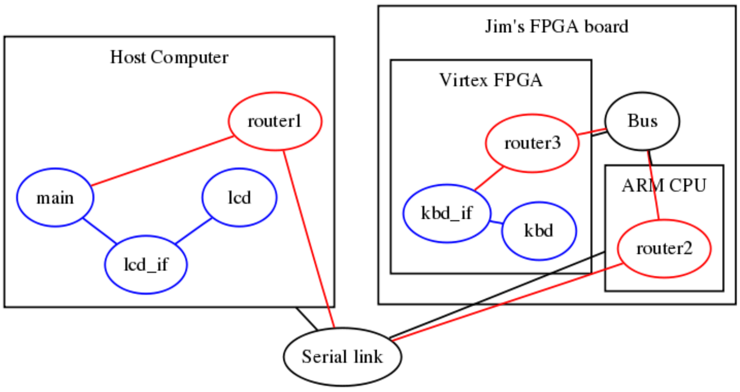
- SystemC requests to the keypad module are forwarded to the hardware keypad via:
  - Host to board controller
  - Board's ARM CPU
  - Board's FPGA
- Response forwarded back

MANCHESTER 1824

The University of Manchester

Hw-Sw Sync-Async Cosimulation

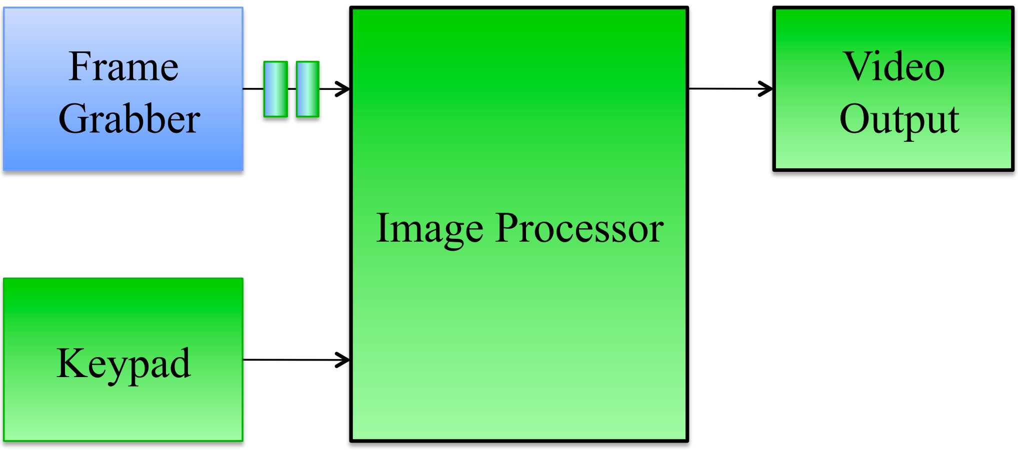
- Main execution on host
- SystemC transaction *"poll keyboard"*
  - Sent to *router1*
  - Routed through *router2* and *router3*
  - Converted to hardware asynchronous channel transaction
- Verilog *keyboard_if*:
  - Synchronous implementation
  - Asynchronous interface



MANCHESTER 1824

The University of Manchester

Demo Contents



4. Processor and output moved to hardware

Based on re-usable open-source cores

MANCHESTER 1824

The University of Manchester

Open-source IP Re-use

- IP re-use from www.opencores.org
  - Video output
    - *VGA/LCD Controller*
  - Keypad scanner
    - *Keypad Scanner*
  - Frame grabber
    - Hand-made component
- Opencores components use *Wishbone* interface

MANCHESTER 1824

The University of Manchester

Final refinement to hardware

- Definition of ports at pin level
- Verilog implementations of modules
- All modules moved to FPGA

The University of Manchester

MANCHESTER 1824

Hands-on Step 10: IP Re-use

- Open and inspect *tutorial_final_1.asip.xml*
- This project illustrates how the cores from opencores.org were imported and connected together in a synchronous way (1 clock domain) with Wishbone interconnect

The University of Manchester

MANCHESTER 1824

Hands-on Step 11: Integration in existing frameworks

- Launch Simulation
  - Environment is setup to demonstrate interactive use of ISE within AsipIDE compilation/synthesis flow
  - Instead of compiling and reporting errors in AsipIDE, designers can debug the Verilog inside ISE while other compilation branches (ARM ASM, SystemC, …) follow their own tool flows

Hands-on Step 12: Assisted GALS design

- Open and inspect *tutorial_final_2.asip.xml*
- This project illustrates "assisted GALS design"
  - Same cores from opencores.org
  - Wrapped by AsipIDE with GALS interfaces
  - Can serve to bootstrap GALS project or to learn about GALS

Quick Peek

- Feature coming soon: Embedded visualisation of HDL
  - E.g. Verilog components will show their inner synthesised netlist
  - Trace file events will be animated on the netlist in the GUI
  - Multiple languages visualised simultaneously

AsipIDE Tutorial

Thank You!