FPGA-based-Prototyping of Accelerated Custom Computing

OpenROAD for LowCost ASIC Design and Innovation Workshop

Novel Integrated Electronics (NINE) Labs Indian Institute of Technology Guwahati

23-January-2024

Sachin B. Patkar
Department of Electrical Engineering
IIT Bombay

Why fpga-oriented talk in OpenROAD workshop ?

- So how do I make topic of this talk fit the agenda ?
- **fpga, by itself, is an ASIC** (application specific integrated chip) For which (specific) application ?
- **The application of fpga-as-an-asic is the one that takes a stream of bits which are shifted into the SRAM cells of the LUTs of the fpga-chip,**
- so that the LUTs are reconfigured as different combinational logic blocks,
- and the interconnections between these combinational logic blocks and FFs are also configured by using some bits of the bitstream to enable / disable connections between prefabricated interconect segments.
- **Thus a desired network of combinational logic blocks and FFs is created on the fpga chip,**
- and this makes the fpga-chip respond to do computation / control according to the digital logic design from which is represented by the bitstream.

FPGA

- FPGA : field programmable gate array
- Contains an array of pre-placed, prefabricated configurable logic elements
  - LUT (look-up-tables) that can be configured into different combinational logic blocks
  - FFs (flops) that can be configured as DFF / TFF , with or without enable-mechanism, with synchronous / asynchronous reset mechanisms etc.
- These pre-placed, pre-fabricated configurable logic blocks are also connected by a prefabricated signal routing conductor path segments
  - Appropriate paths are stitched together by configuring pre-placed, pre-fabricated switches
  - Switches are essentially pass-transistors whose gate-terminals are driven by a SRAM bit that can be configured to hold 0 or 1

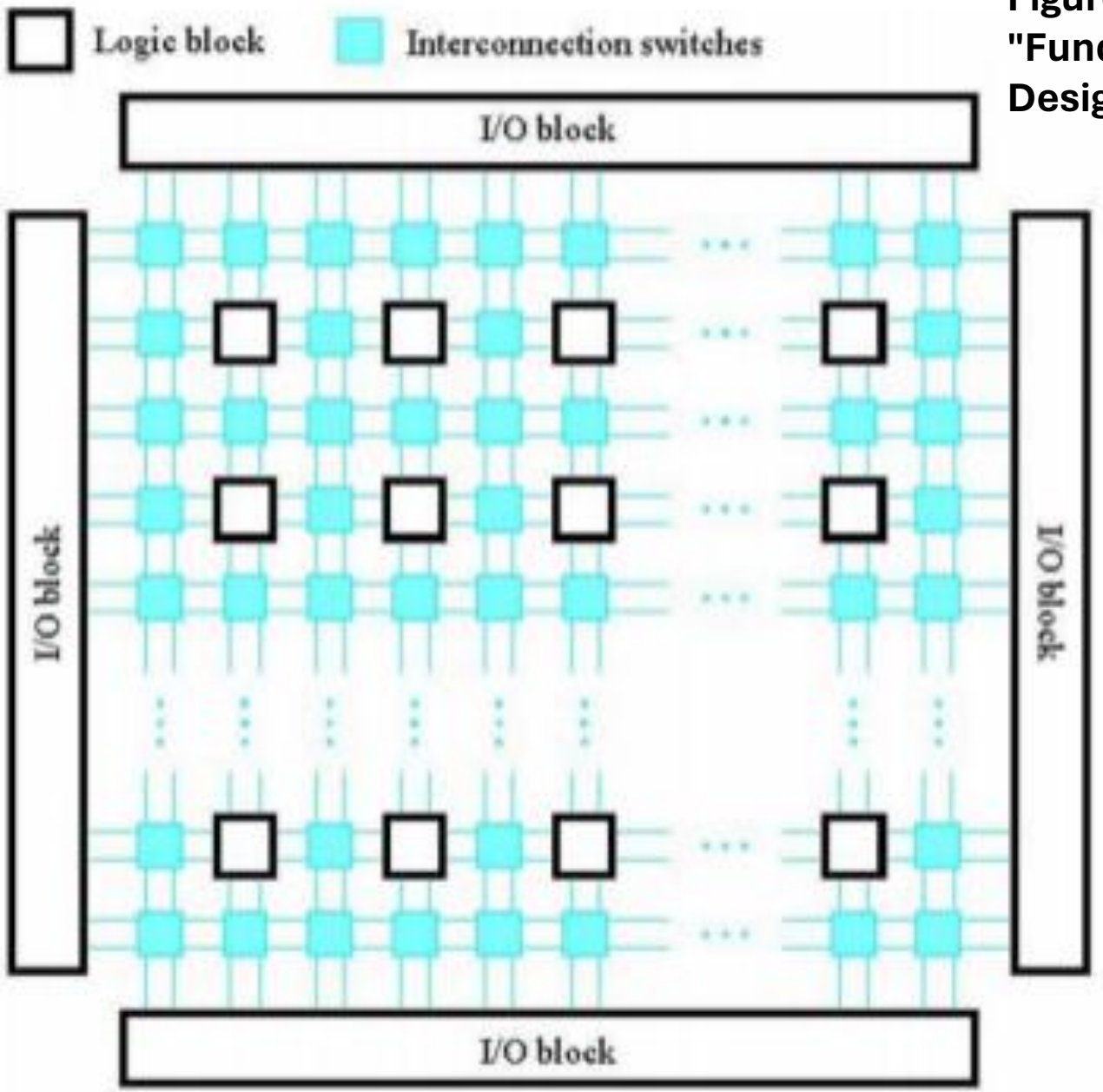
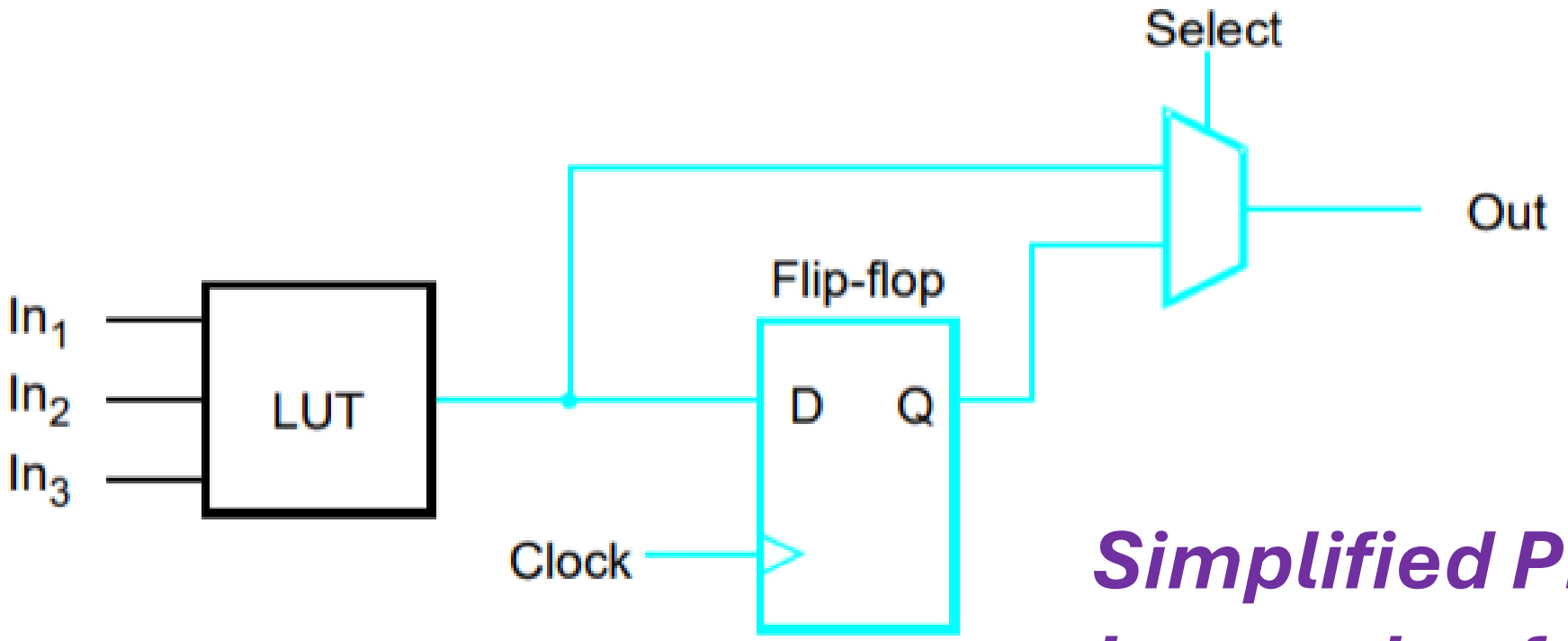


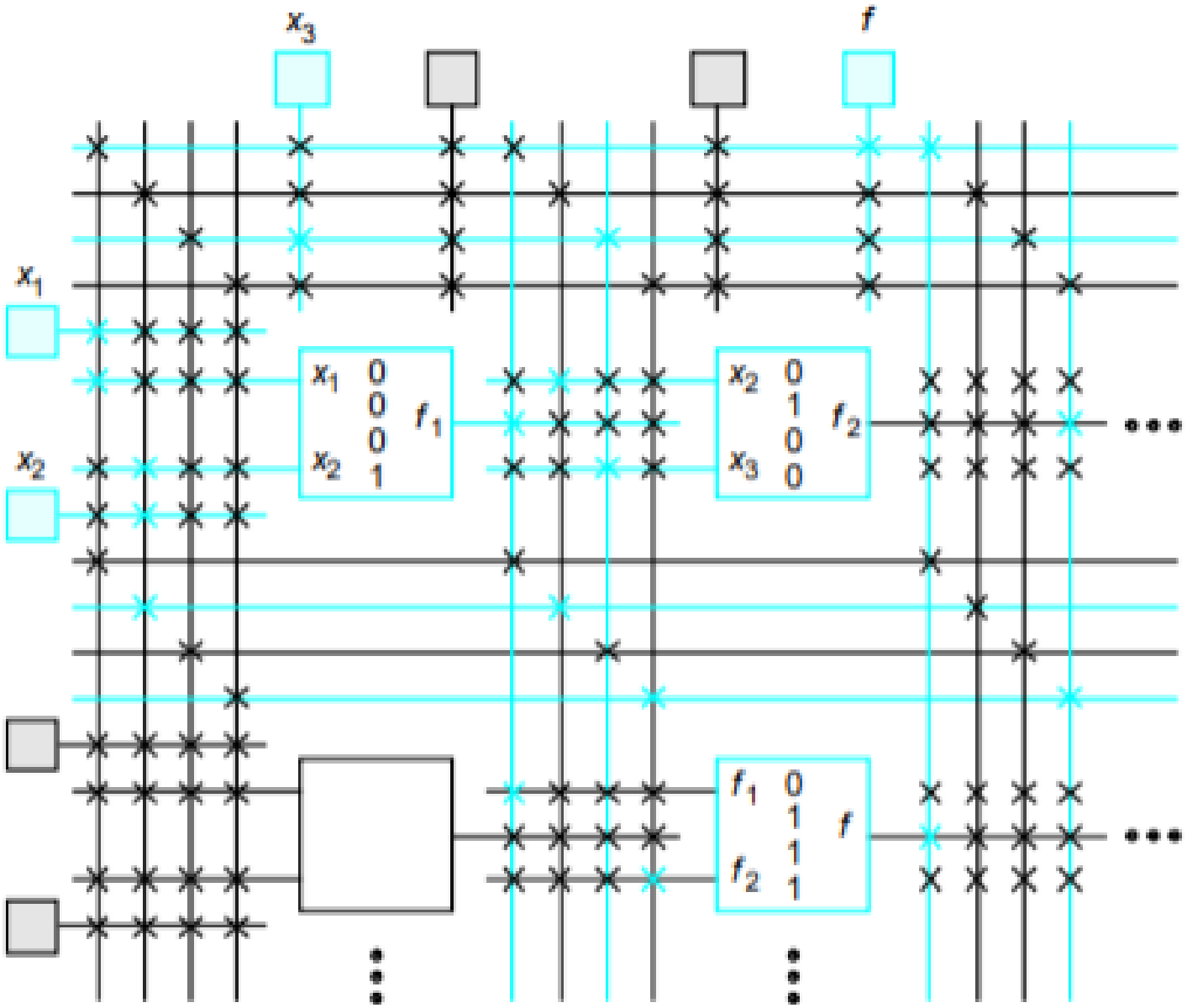
Figure courtesy Brown and Vanesic's "Fund. Of Digi Logic with VHDL/Verilog Design"

FPGA

Figure courtesy Brown and Vanesic's "Fund. Of Digi Logic with VHDL/Verilog Design"



Simplified Picture of Internals of CLB of an (SRAM based) FPGA



Result of Place-n-Route

Figure courtesy Brown and Vanesic's "Fund. Of Digi Logic with VHDL/Verilog

(continued) Why fpga-oriented talk in OpenROAD workshop ?

- Thus, though, fpga itself is an ASIC chip, we can make it work like a network of digital logic blocks implemented physically in silicon.
- Turnaround time, and cost of this implementation is very less compared to that of fabricating an ASIC-chip.
- Furthermore, I will allude to similarities and connections between the automation tools for fpga based design and asic based designs.
- I will informally talk about methodologies of fpga-based prototyping and validation,
- so that the asic-chip-designer can have more confidence in investing significant energy and resource in the asic-chip design.
- I will spend most time talking about simple, accessible things rather than research and novelties.

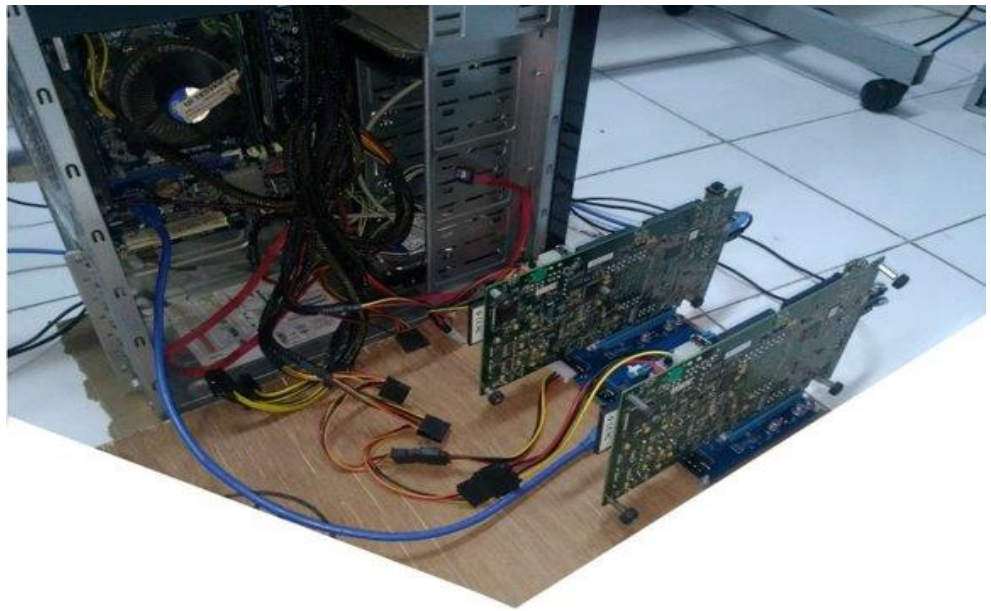
Plan of Presentation

- A brief mention of some fpga-based research projects in my group (no details provided)
- FPGA vs ASIC design flow (alluding to algorithmic approaches)
- FPGA based prototyping platforms
- FPGA based validation tutorial

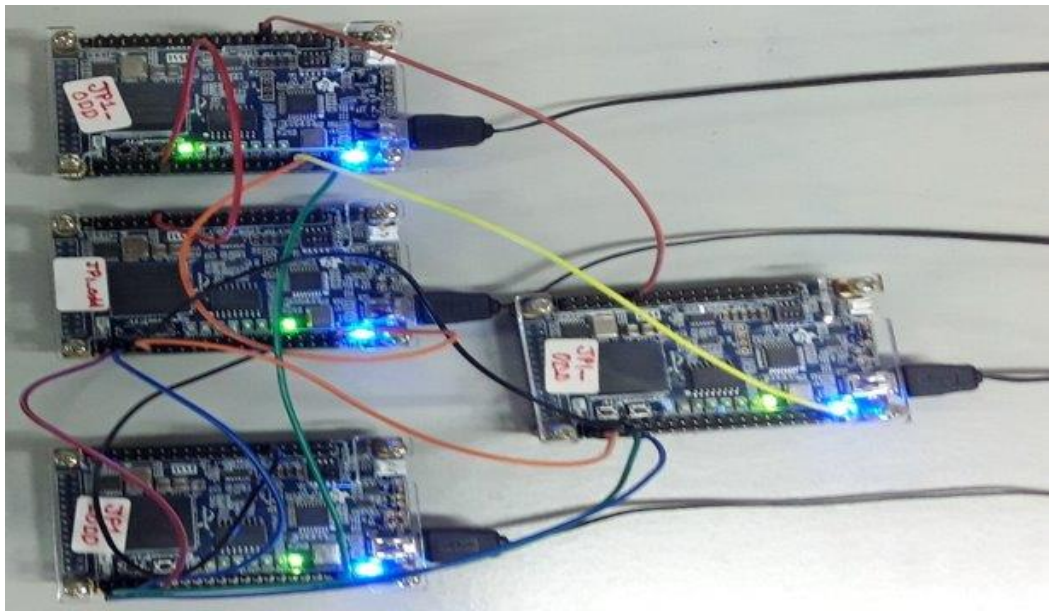
Programmable Logic Device (PLD) e.g. FPGA

- FPGA : field programmable gate array
- Contains an array of pre-placed, prefabricated configurable logic elements
  - LUT (look-up-tables) that can be configured into different combinational logic blocks
  - FFs (flops) that can be configured as DFF / TFF , with or without enable-mechanism, with synchronous / asynchronous reset mechanisms etc.
- These pre-placed, pre-fabricated configurable logic blocks are also connected by a prefabricated signal routing conductor path segments
  - Appropriate paths are stitched together by configuring pre-placed, pre-fabricated switches
  - Switches are essentially pass-transistors whose gate-terminals are driven by a SRAM bit that can be configured to hold 0 or 1

ZCU104

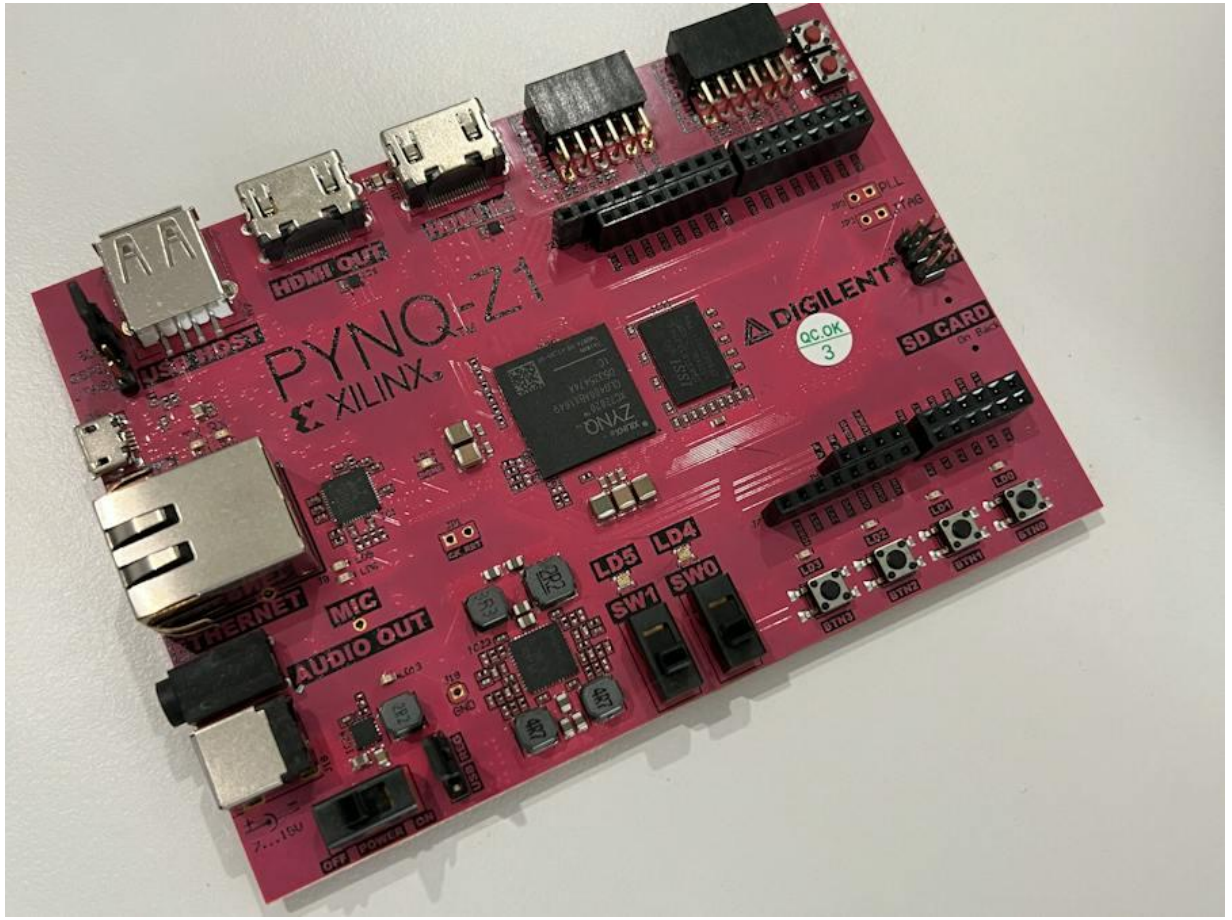


Multi-FPGA setup with ML605 Boards



Multi-FPGA prototyping using Deo-Nano Boards

PYNQ Z1 fpga-kit based on ZYNQ-fpga-device from AMD-Xilinx



Sponsored Research Projects (in the field of fpga-based computing) :

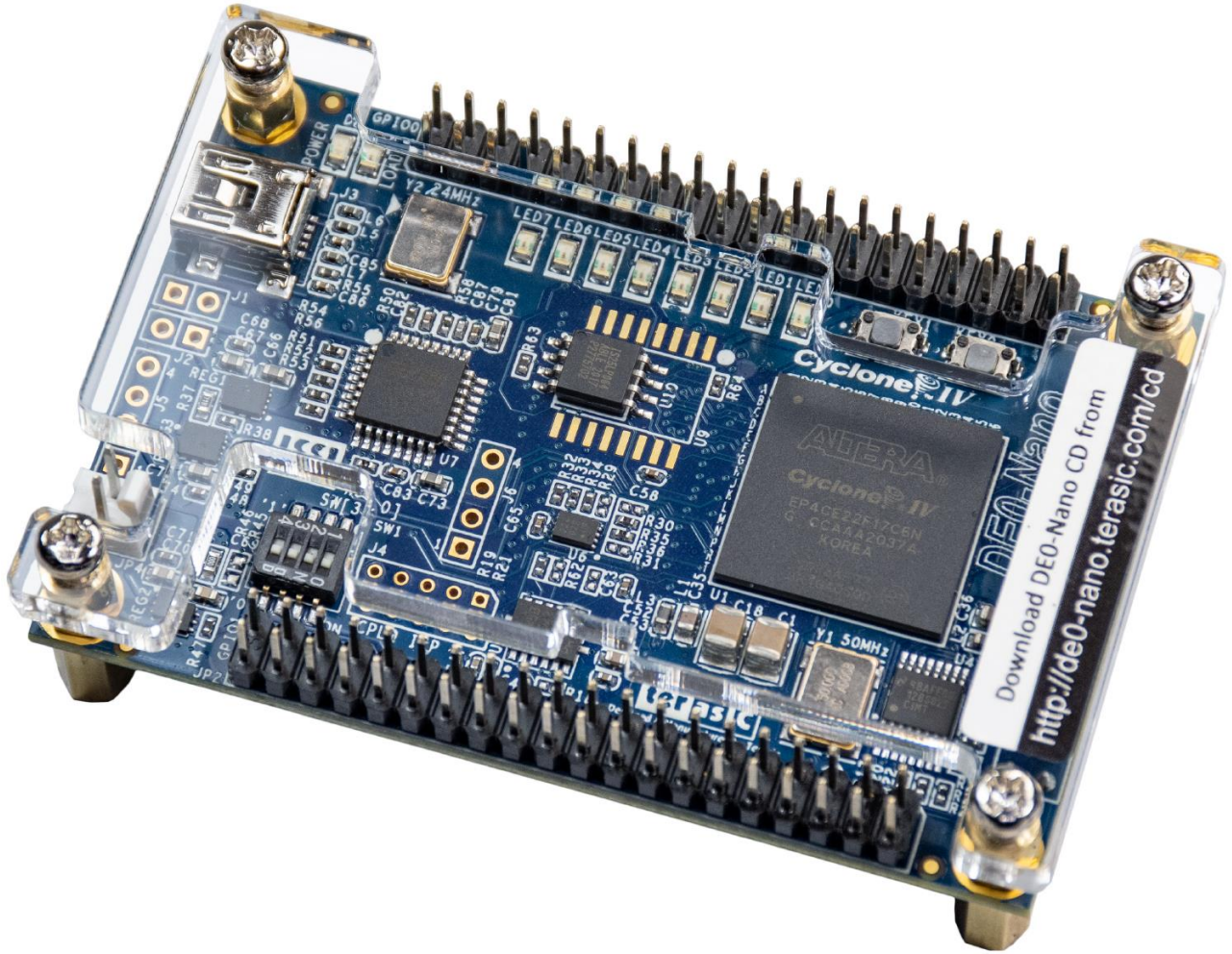
UAV- Mounted FPGA Based Stereo Vision

Ultrascale-96 FPGA for stereo-vision

- UAV-Mounted FPGA Based Stereo Vision
- FPGA-based object tracking using particle filter and FPGA-based circuit solvers

KINTEX 7 fpga kit

DE0-Nano fpga-kit based on a Cyclone-IV fpga device from Altera/Intel-FPGA



FPGA-based-Accelerator Designs using Combinatorial and Linear Geometries at HPeC Lab, EE, IITB

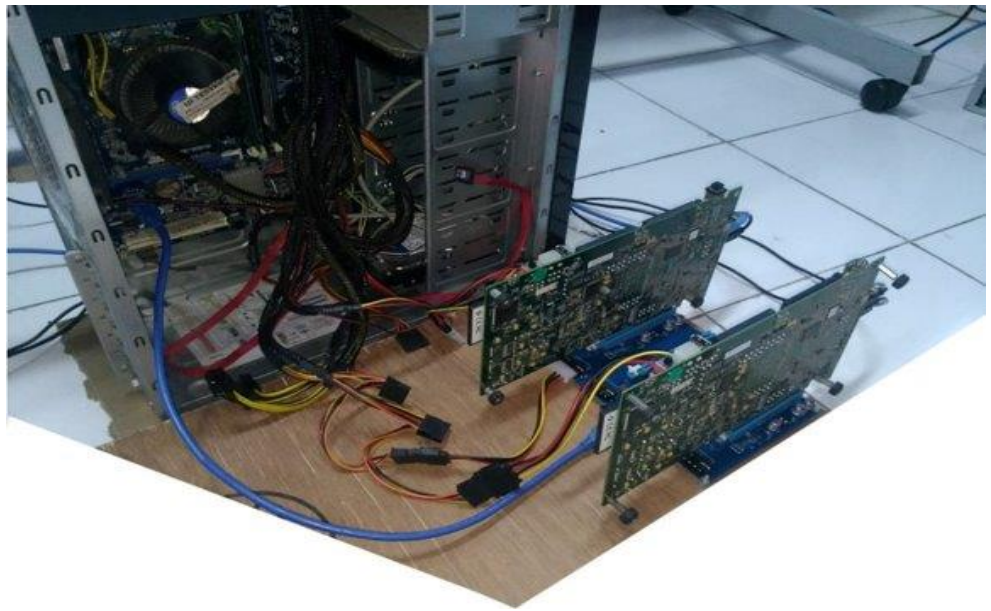
Sachin B. Patkar
Department of Electrical Engineering
IIT Bombay

Multi-FPGA Design Automation and Prototyping

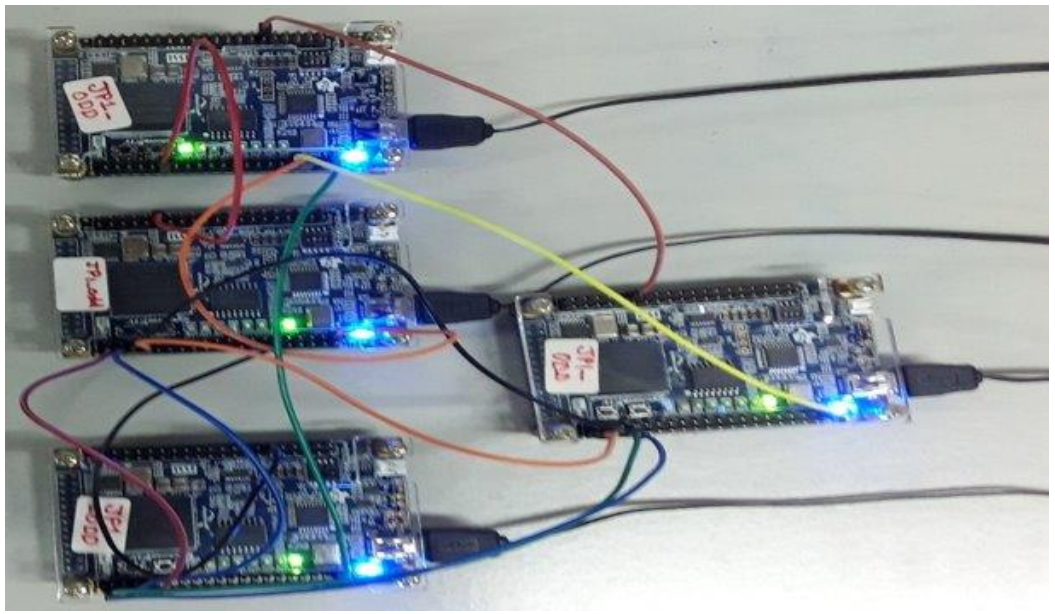
From PhD thesis of Vinay B.Y. (presently at IMEC, Belgium) and Mandar Datar (presently at Microsoft)

Globally Asynchronous Locally Synchronous Designs

- Specified using minimal MPI-lite syntax (send-receive calls)
- Scheduling / Binding etc. done using Combinatorial and Linear Systems Ideas
- A mini-compiler that creates scripts that create Bluespec-System-Verilog and Vivado-HLS wrappers and scripts



Multi-FPGA setup with ML605 Boards



Multi-FPGA prototyping using Deo-Nano Boards

Combinatorial Projective Geometry based Large Network of FPGAs for Integer Factorization

From PhD thesis of Mandar Datar (presently at Microsoft)

Results and Observations

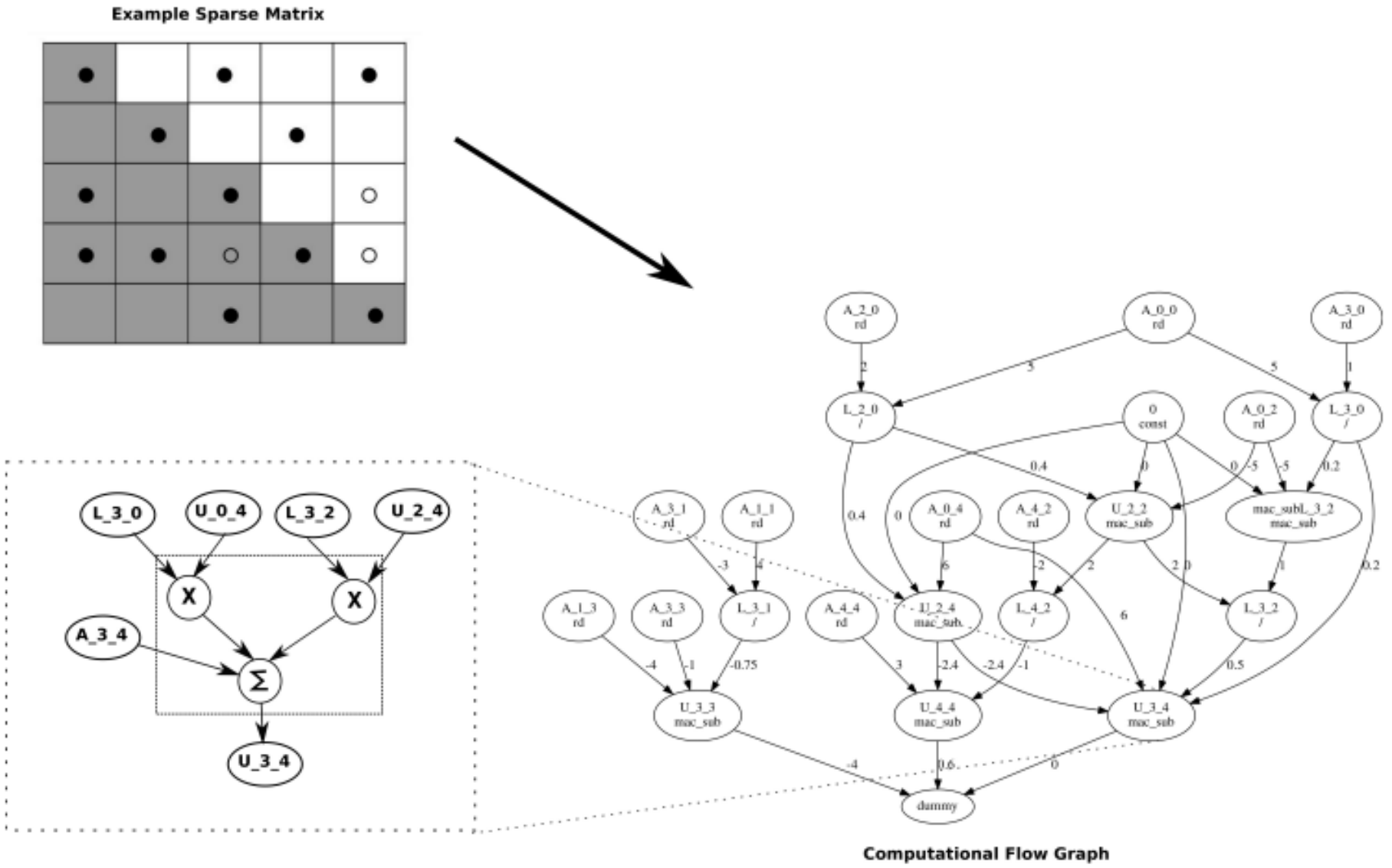
q	d	N	Clock Cycles	F	Ops(x10^{15})	Ops/FPGA /Cycle
2	3	7	61	21	0.011	5013
3	4	13	79	52	0.028	5392
4	5	21	98	105	0.059	5617
5	6	31	116	186	0.109	5837
7	8	57	152	456	0.280	6144
8	9	73	170	657	0.411	6254
9	10	91	188	910	0.577	6344
11	12	133	225*	1596	1.030	6457

Table: Simulation Results

FPGA-based-acceleration of Circuit Simulation

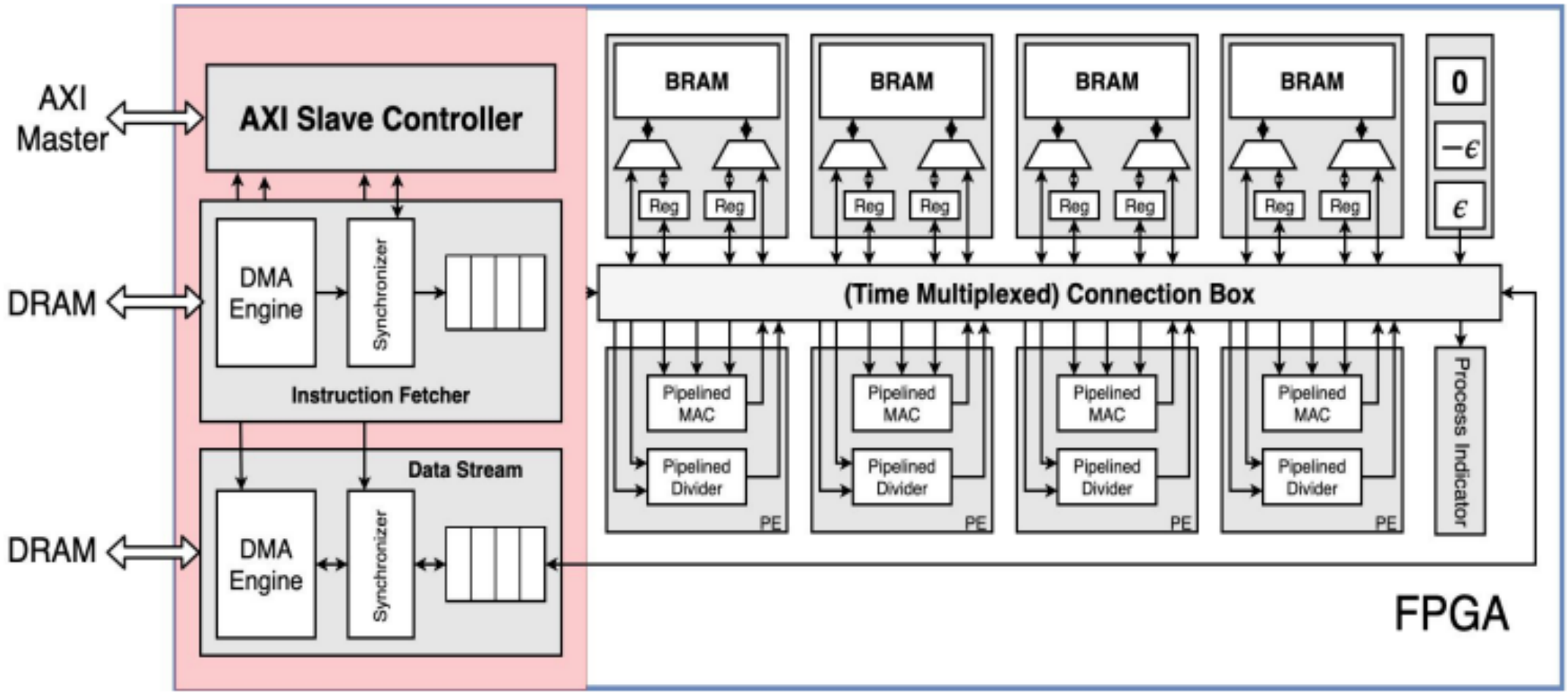
From PhD thesis work of Mini Namboothiripad (member of Faculty of U. Mumbai) and Mandar Datar (Microsoft) (with key contribution from DD students Yogesh Mahajan (presently at TI) and Shashank Obla (presently at CMU))

Computational Flow Graph Construction [1]



[1] Tim Davis, Kapre-deHon, Tarek Nechma - Zwolinski

Hardware Architecture for Sparse LU Decomposition



Network and Matrix Decomposition Based Acceleration of Circuit Simulation using FPGA

Network Decomposition

Multiport decomposition ➔ Make algorithms more parallel

Parallelism ➔ exploited by the FPGA

Matrix Decomposition

Key operation in circuit simulation

➔ LU decomposition of sparse matrix

Scheduling with column-dependency graph

➔ Exposes more parallelism

The available parallelism can be extracted by the FPGA

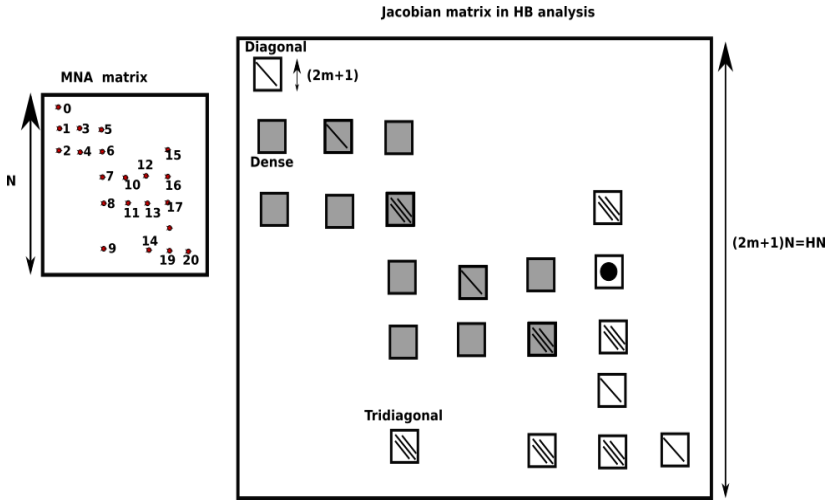
Targeted Problems

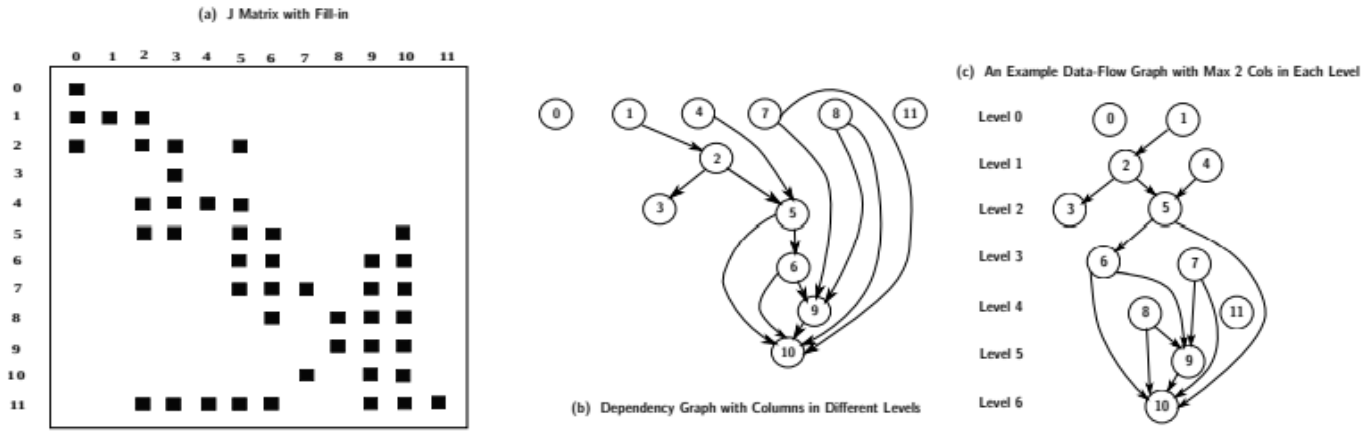
Real-Time Simulation of Power Electronic converters

Acceleration of Harmonic Balance Algorithm

Acceleration of Harmonic Balance Algorithm

- Column-dependency (Data-Flow) graph ➔ Columns are arranged in diff. levels using Data-Flow graph ➔ Improved Parallelism
- Developed a Matrix Generator and Netlist Parser Program in C++

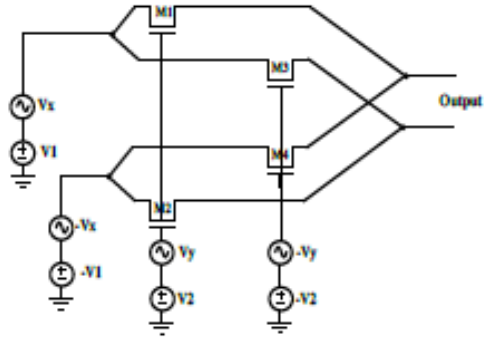




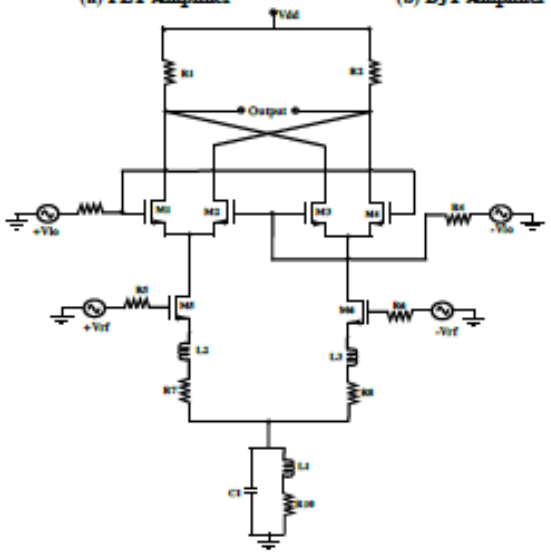
TEST CIRCUITS & RESULTS

(a) FET Amplifier

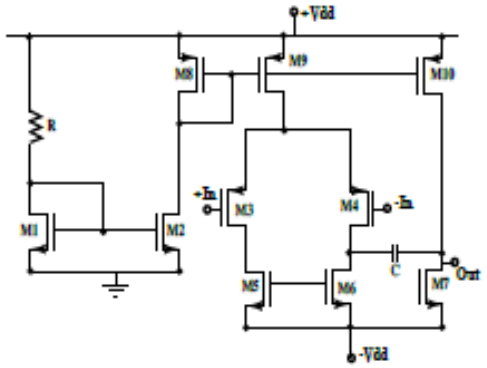
(b) BJT Amplifier



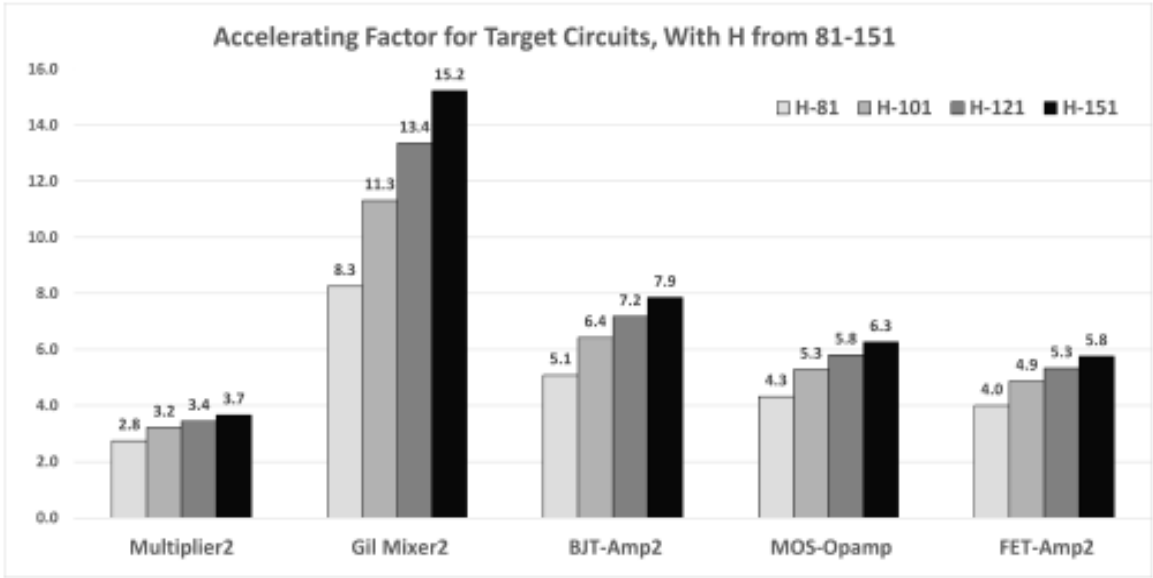
(c) Multiplier



(d) Gilbert's Double Balanced Mixer



(e) MOSFET-Based OpAmp



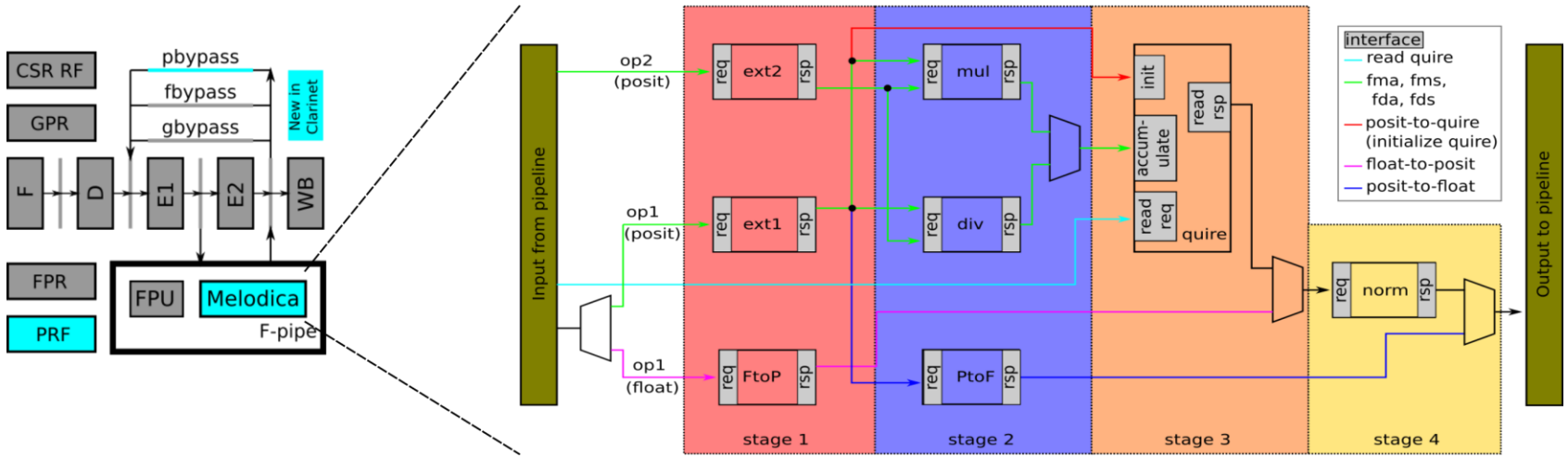
HW-acceleration of POSIT-arithmetic for RISC-V-SoC

From ongoing PhD thesis work of Niraj Nayan Sharma (ex-Bluespec and currently at InCoreSemi)

Clarinet: A Quire-elabled RISC-V-based Platform

Clarinet is a general-purpose CPU platform to experiment with Posits

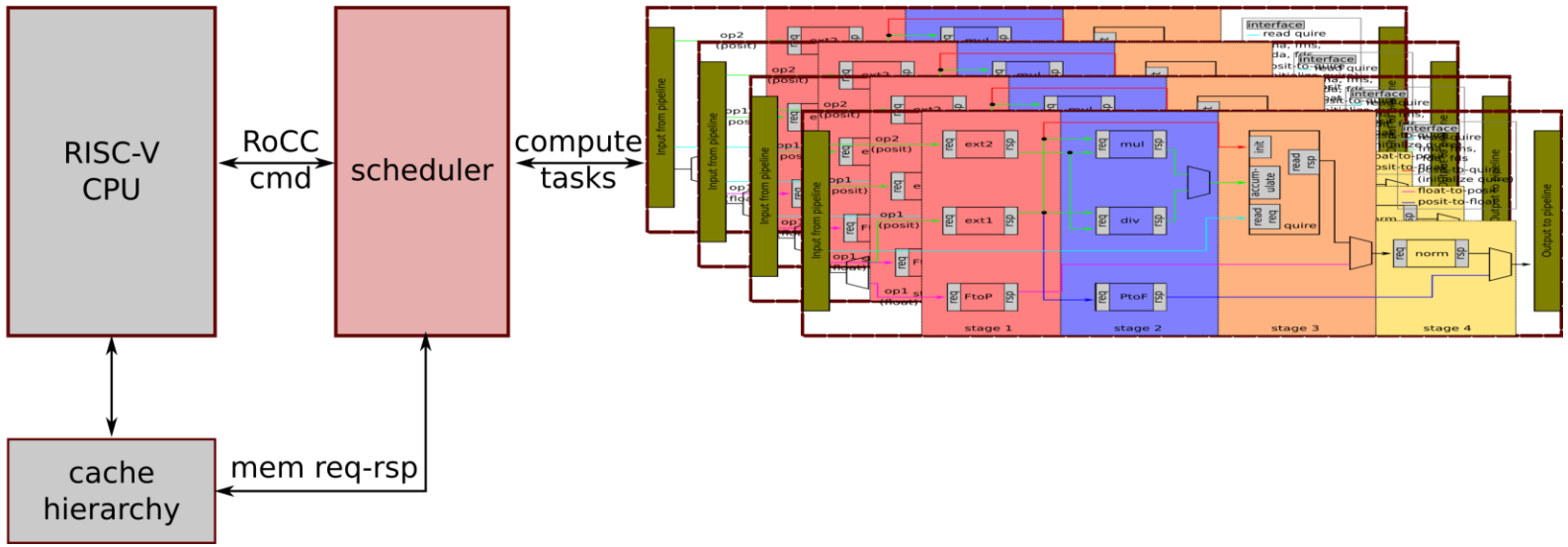
- First RISC-V CPU to integrate Quire accumulation with the RISC-V ISA
- First RISC-V CPU to support seamless coexistence of floating-point and posit numbers in the application software
- Customized RISC-V ISA assemblers to support posit operations
- Flexibility allows posit computing to be integrated as a functional unit or tightly coupled accelerator to the RISC-V pipeline
- Used to study efficacy of posits in applications like computer vision, linear algebra and motion estimation



Quills: A coprocessor for error-free linear algebra

Quills is a dense linear algebra coprocessor that use Posit arithmetic

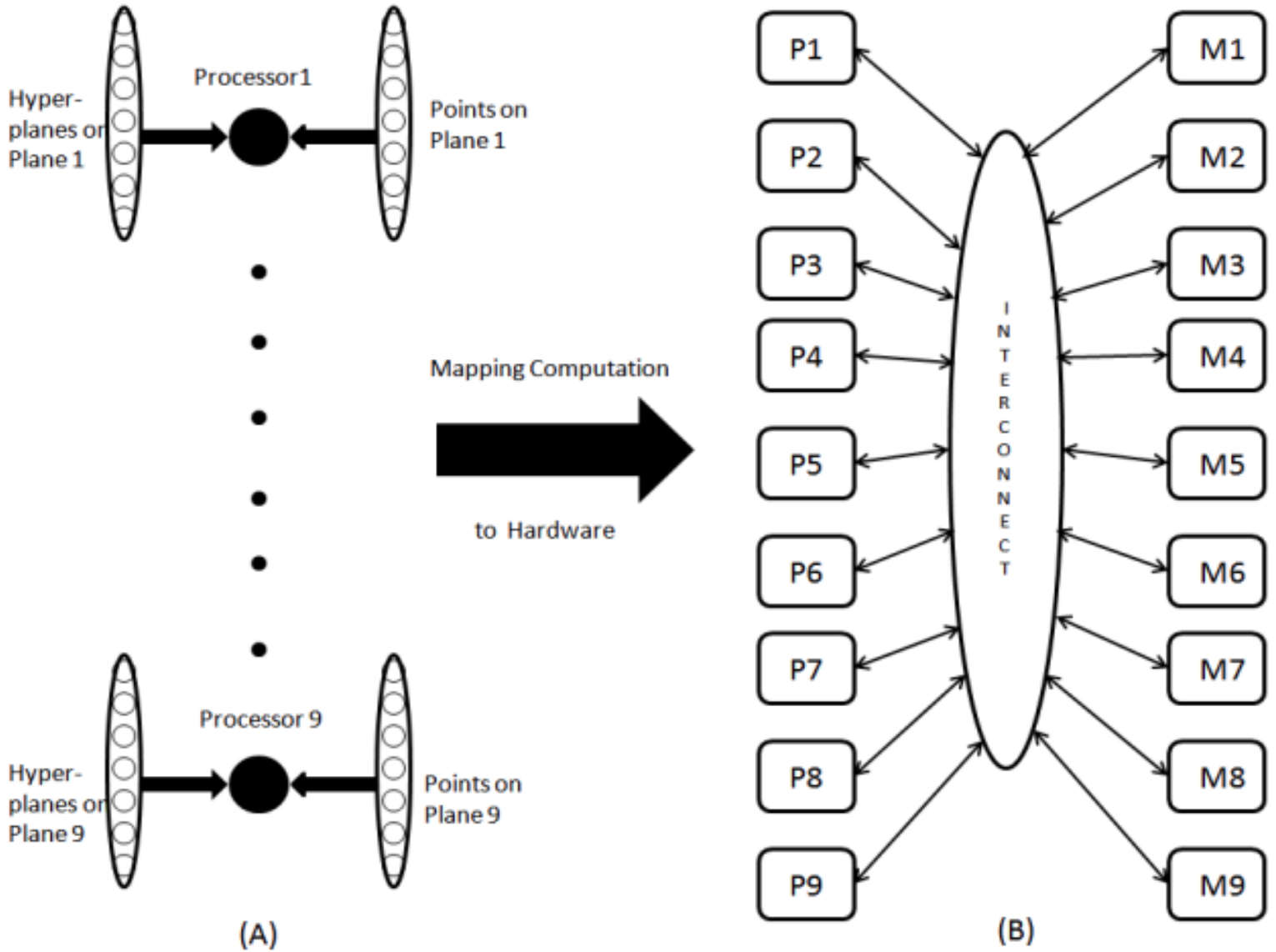
- Works with off-the-shelf RISC-V cores via a standard RoCC interface. Custom instructions in the RISC-V program drive commands into the coprocessor.
- The scheduler abstracts away the application-specific data flow from the computation in the kernel
- Uses multiple computational cores (Melodicas) for parallel computation
- Higher-order operations reuse lower-order operations by nesting operations in the scheduler - creating a compose-able hardware library of functions



Combinatorial Geometry based Architectures for Expander-graph-based Codes

From PhD thesis of Hrishikesh Sharma (TCS Research)

Hardware Design for $\mathbb{PG}(5, \mathbb{GF}(2))$ II



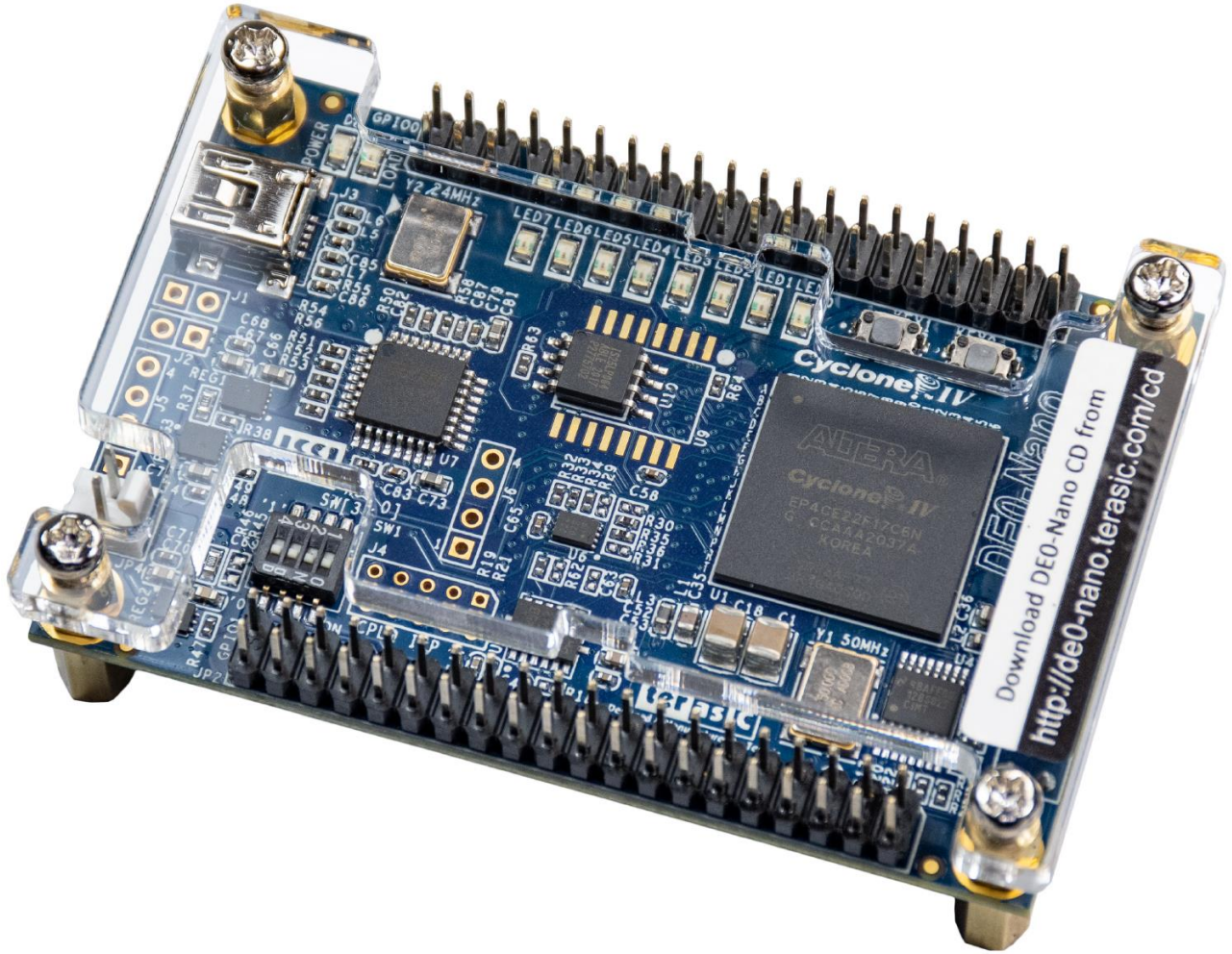
Hardware Implementation Results I

- We have implementated a prototype of the decoder on a XUPV5-LX110T FPGA board.
- Xilinx RS Decoder v7.0 used for sub-code decoding
- Post Place and Route frequency = 187.2 MHz
- Aproximate Throughput calculation :

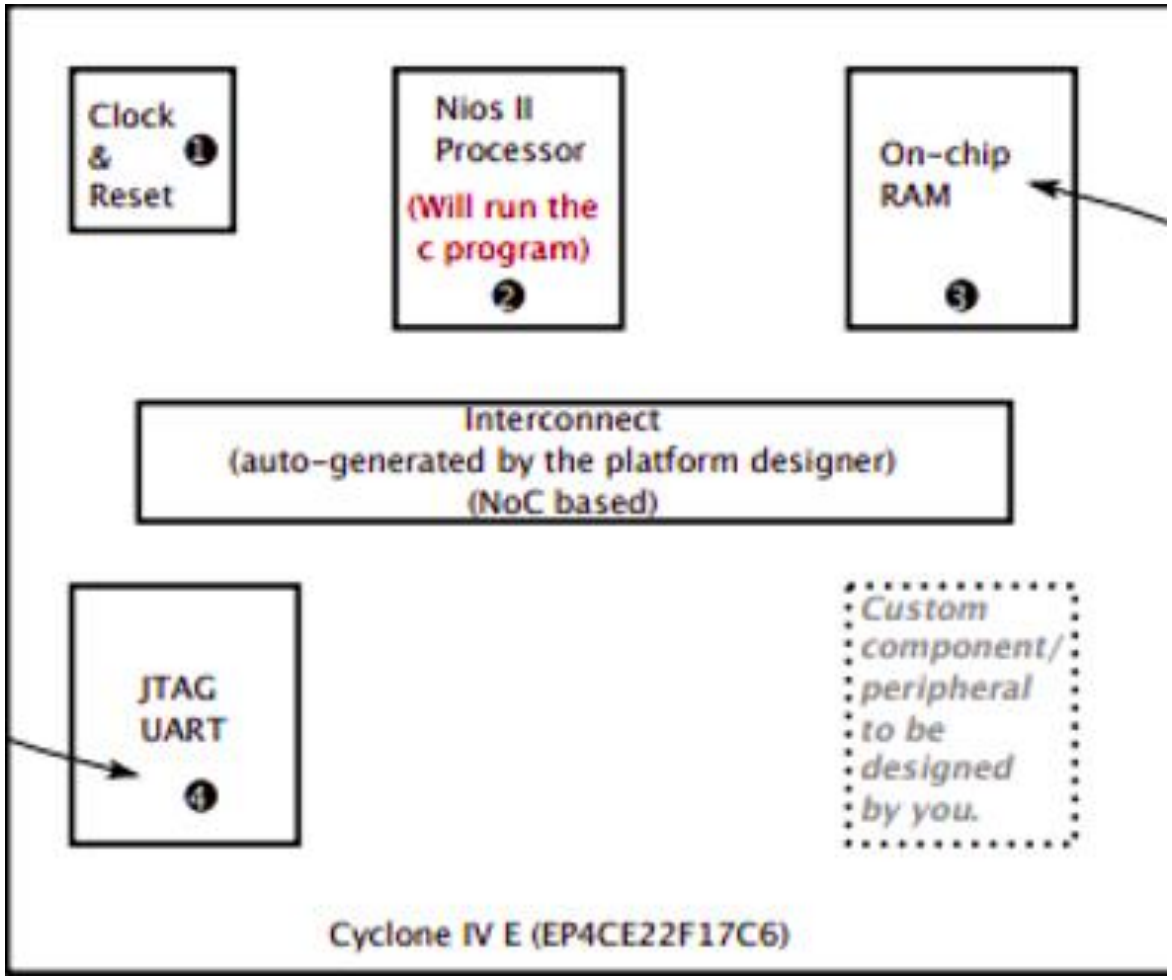
$$\text{Throughput} = \frac{\text{Number of Data symbols}}{\text{Number of clock cycles}} * \textit{frequency}$$

It is almost 130 MBytes/s when $\epsilon = 5$

DE0-Nano fpga-kit based on a Cyclone-IV fpga device from Altera/Intel-FPGA



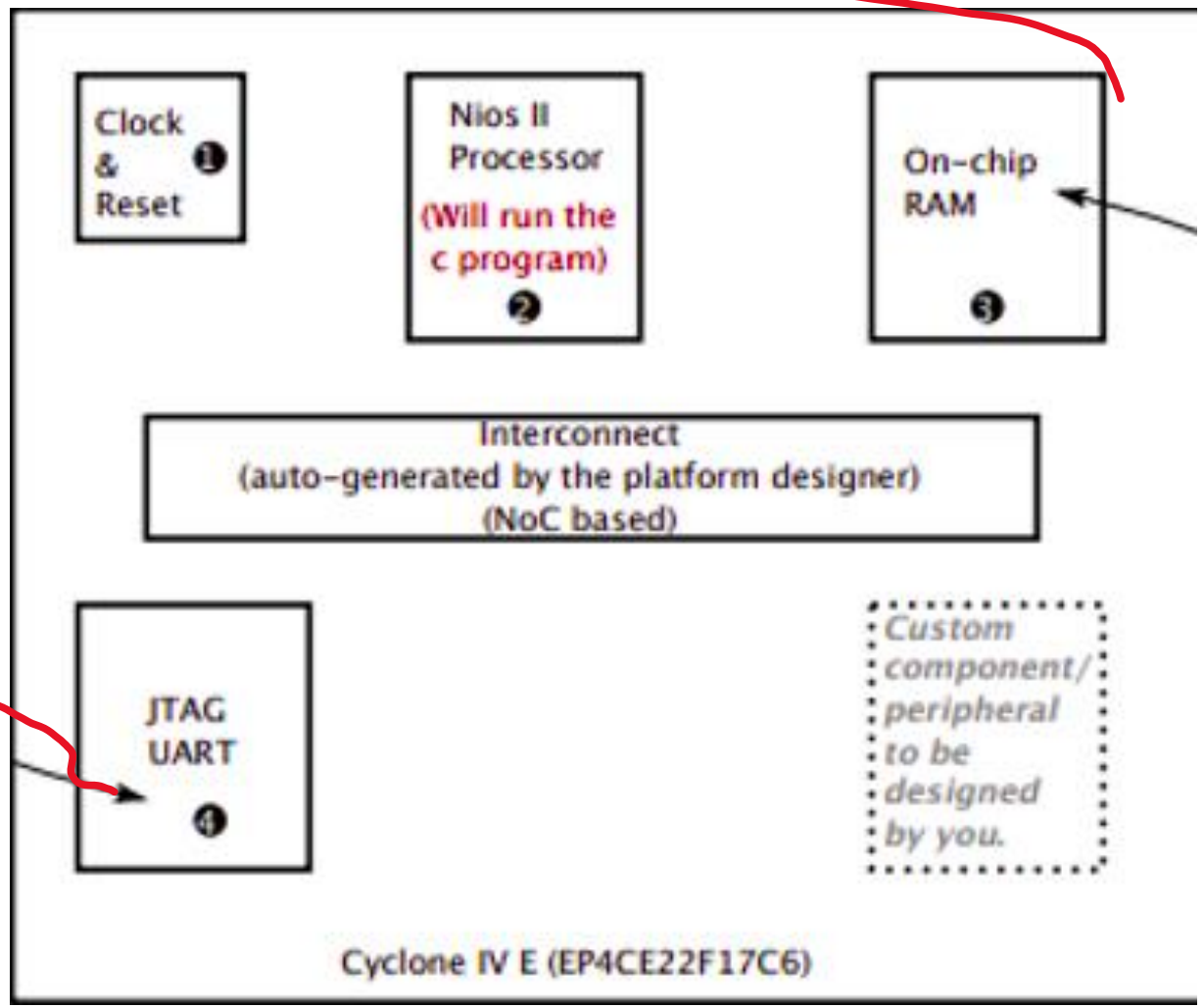
HW-SW SoC design on DE0-Nano kit using the Cyclone-IV E fpga device



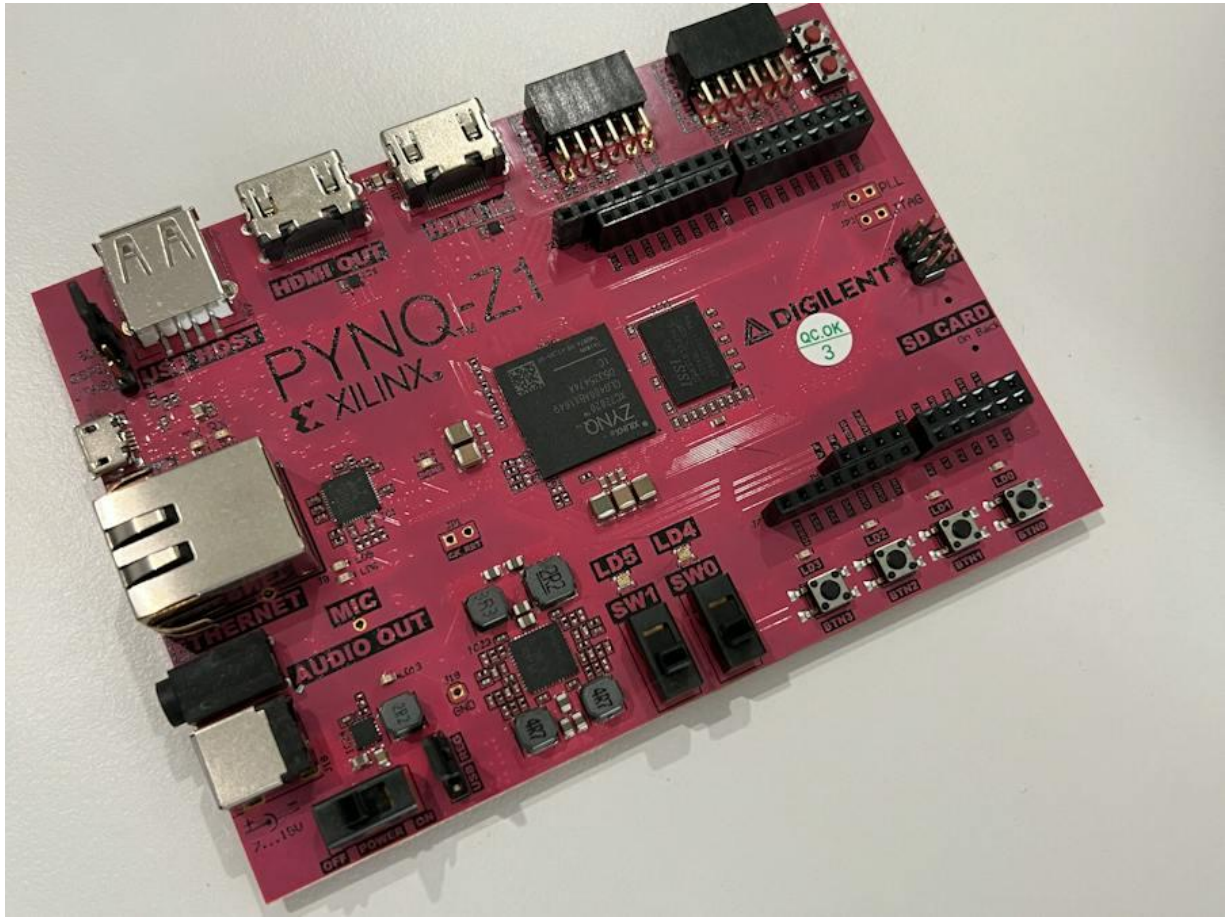
(A) The platform gets set-up into the FPGA, once we program the FPGA.
However, software is still not present.

(B) When "Run As -> Nios II Hardware",
the elf file for the software will be copied
into this memory, and nios processor will be reset.
Then, the processor will start executing the code
from the elf file.

(C) Data written by Nios II using
putstr(..), printf(..) etc. functions
will be sent to the host via JTAG UART module,
and will appear in nios console.



PYNQ Z1 fpga-kit based on ZYNQ-fpga-device from AMD-Xilinx



Fpga-based SoC design on PYNQ fpga-kit

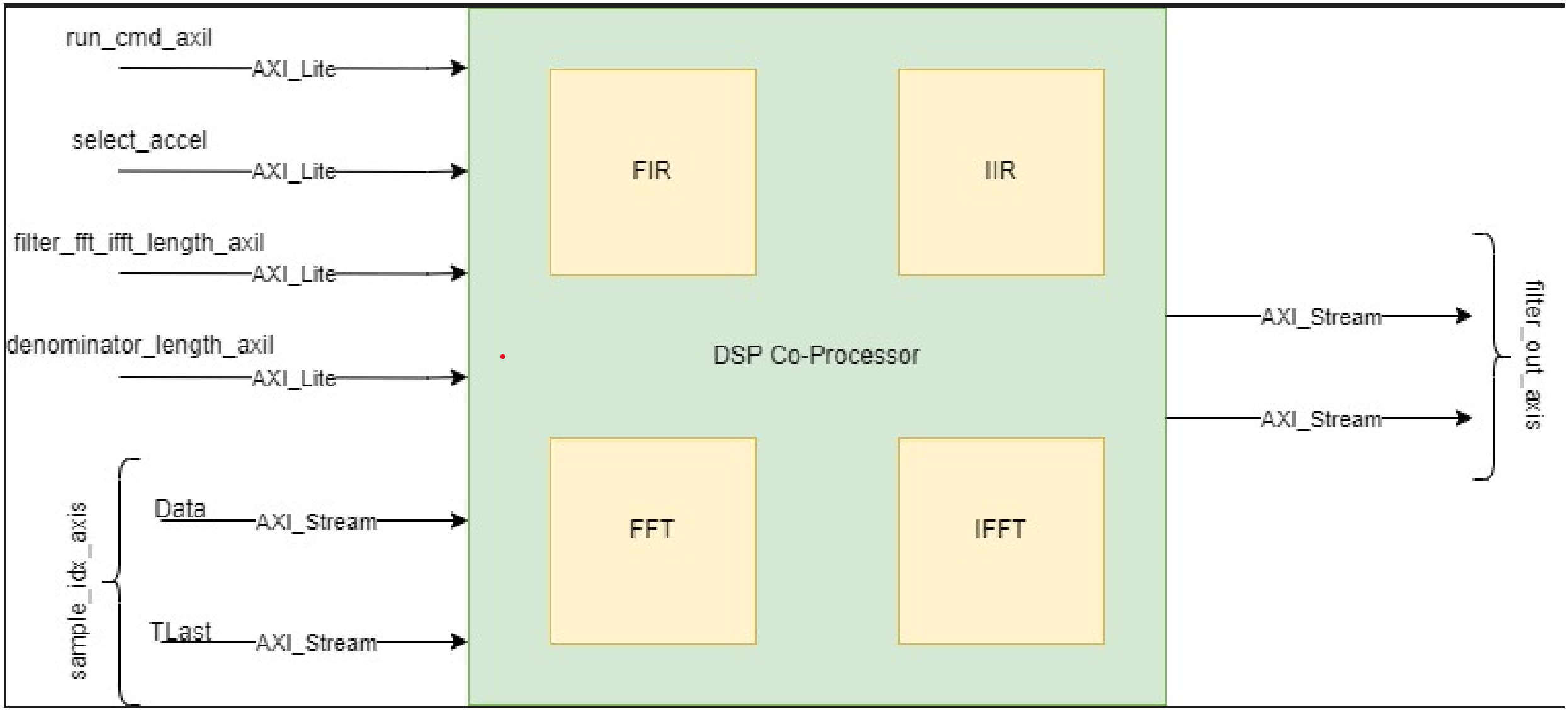
- PYNQ has Ubuntu Linux at its core that would run on
  - Processing System (PS) of ZYNQ-SoC
    - which is based on ARM cpu, and
  - special drivers for connecting to the RTL/HLS-generated IP block
    - that is in the PL (programmable logic area of the fpga)
- Use of python over this linux based system means avoiding tedious procedures of building and installing the C/C++ codes to run on the PS (ARM based processing system)
- Jupyter notebook instead of tedious serial terminal based front end to talk to IP.
- Productivity advantages …

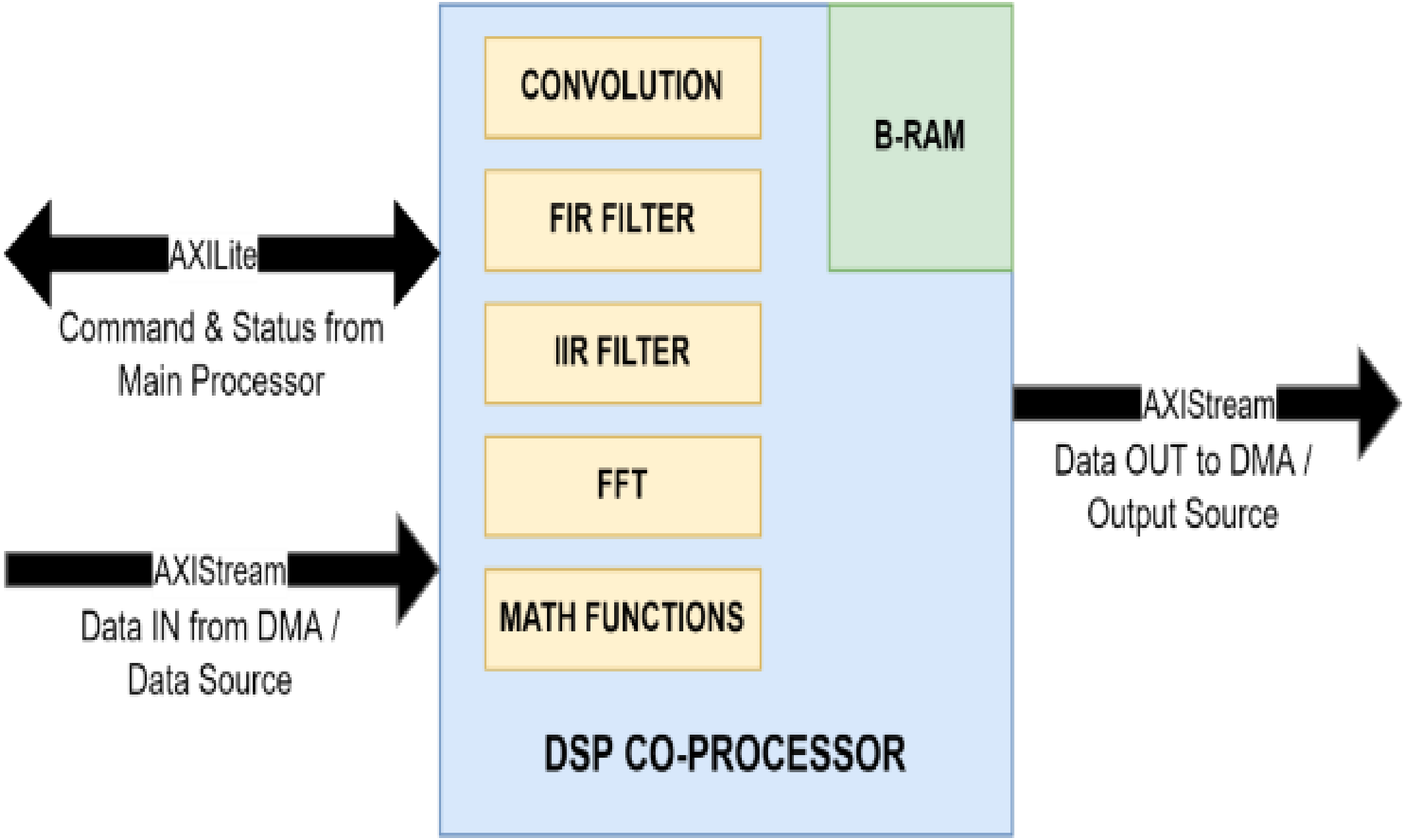
- User created IP block is typically coded at HLS level so that the AXI interfaces can be used to connect the IP block as part of AXI interconnect fabric.
  - Thereby enabling connectivity to FIFO-based streaming communication, memory mapped IO (mmio) comunication, DMA based direct access to large DRAM
- AXI
  - AXI4 (full)
  - AXI4lite
  - AXIstream
- Master / Slave interactions ….
- Connections are largely automated in order to build the bitstream modeling the whole SoC (without the hard-IP-block of ARM-based PS) that would now include custom-user-created IP blocks to enhance performance and features.

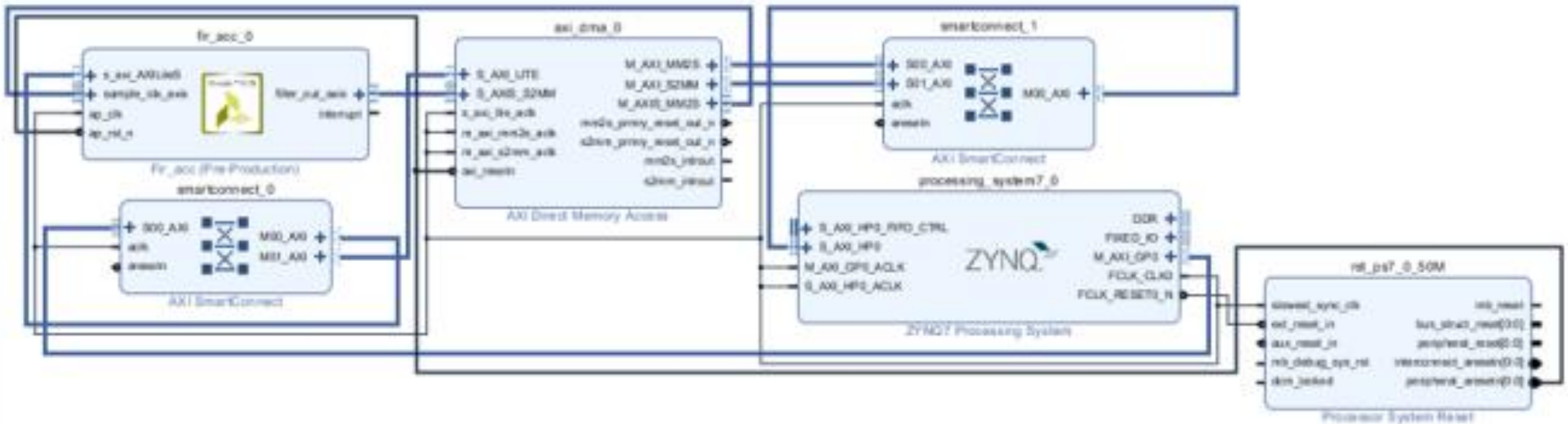
```
void DSP_acc (short int run_cmd_axil,
              short int select_accel,
              short unsigned int filter_fft_ifft_length_axil,
              short unsigned int denominator_length_axil,
              hls::stream<coeff_t_stream> &sample_idx_axis,
              hls::stream<coeff_t_stream> &filter_out_axis)
{

    #pragma HLS INTERFACE axis register both port=filter_out_axis
    #pragma HLS INTERFACE axis register both port=sample_idx_axis
    #pragma HLS INTERFACE s_axilite port=run_cmd_axil
    #pragma HLS INTERFACE s_axilite port= select_accel
    #pragma HLS INTERFACE s_axilite port=filter_fft_ifft_length_axil
    #pragma HLS INTERFACE s_axilite port=denominator_length_axil
    #pragma HLS INTERFACE s_axilite port=return
```

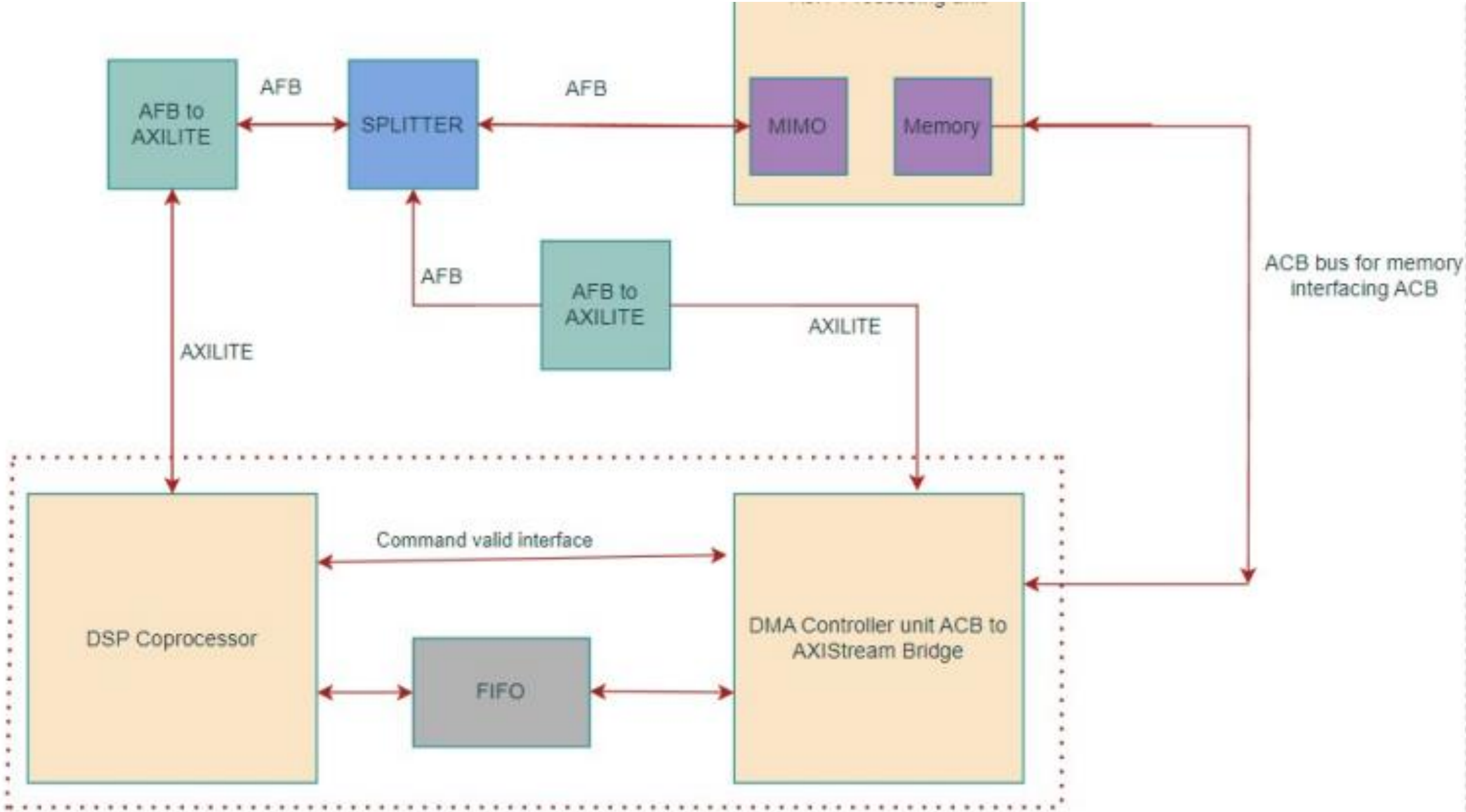
```python
from pynq import Overlay
import pynq.lib.dma
from pynq import allocate
from pynq import MMIO
import numpy as np
import time
import struct
base_addr = 0x43C00000
addr_range = 0x10000
mmio = MMIO(base_addr, addr_range)

overlay = Overlay('./design_1.bit')

dma = overlay.axi_dma_0
dma_send = dma.sendchannel
dma_recv = dma.recvchannel

mmio.write(0x18,11)  #filter length = 11
mmio.write(0x10,0x2222)  #run command
```

```python
dma_send.transfer(in_buffer)
dma_recv.transfer(out_buffer)
dma_recv.wait()
```


run_cmd_axil
AXI_Lite
select_accel
AXI_Lite
filter_fft_ifft_length_axil
AXI_Lite
denominator_length_axil
AXI_Lite
sample_idx_axis
Data
AXI_Stream
TLast
AXI_Stream
FIR
IIR
DSP Co-Processor
FFT
IFFT
AXI_Stream
AXI_Stream
filter_out_axis


CONVOLUTION
FIR FILTER
IIR FILTER
FFT
MATH FUNCTIONS
B-RAM
DSP CO-PROCESSOR
AXILite
Command & Status from
Main Processor
AXIStream
Data IN from DMA /
Data Source
AXIStream
Data OUT to DMA /
Output Source


fir_acc_0
s_axi_AXILiteS
sample_ide_axis
ap_clk
ap_rst_n
fifer_out_axis
interrupt
Fir_acc (Pre-Production)
smartconnect_0
S00_AXI
aclk
aresetn
M00_AXI
M01_AXI
AXI SmartConnect
axi_dma_0
S_AXI_LITE
S_AXIS_S2MM
s_axi_lite_aclk
m_axi_mm2s_aclk
m_axi_s2mm_aclk
axi_resetn
M_AXI_MM2S
M_AXI_S2MM
M_AXI_SG
mm2s_prmry_reset_out_n
s2mm_prmry_reset_out_n
mm2s_introut
s2mm_introut
AXI Direct Memory Access
smartconnect_1
S00_AXI
S01_AXI
aclk
aresetn
M00_AXI
AXI SmartConnect
processing_system7_0
S_AXI_HP3_FIFO_CTRL
S_AXI_HP3
M_AXI_GP0_ACLK
S_AXI_HP3_ACLK
ZYNQ
DDR
FIXED_IO
M_AXI_GP0
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
rst_ps7_0_50M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset


AFB to AXILITE
AFB
SPLITTER
AFB
MIMO
Memory
ACB bus for memory interfacing ACB
AFB
AFB to AXILITE
AXILITE
AXILITE
DSP Coprocessor
Command valid interface
FIFO
DMA Controller unit ACB to AXIStream Bridge

DSP co-processor

The co-processor works along with the main processor and it is customised to perform the following specific DSP related tasks.

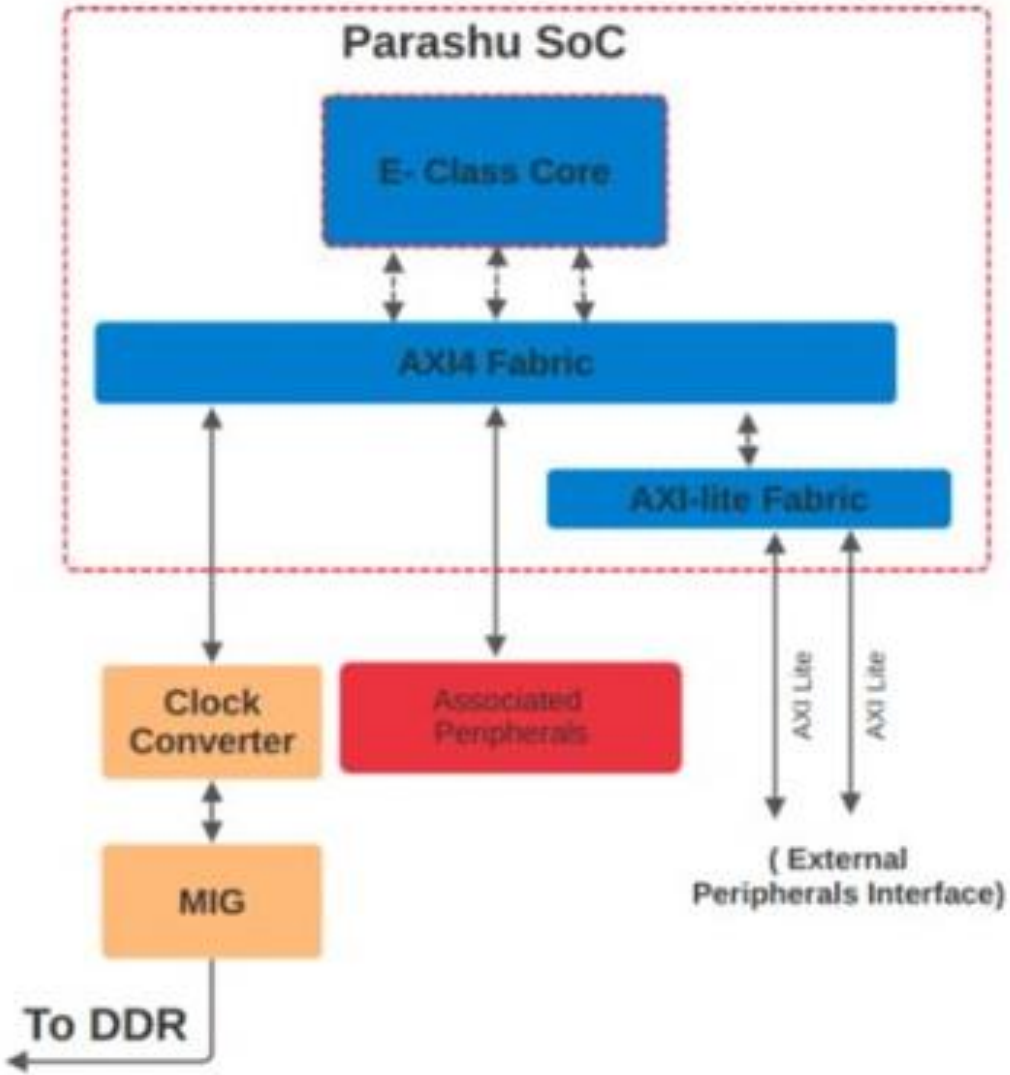
a) Convolution b) Finite Impulse Response Filter c) Infinite Impulse Response Filter
d) Fast Fourier Transform implementation e) Math functions

The designed DSP co-processor consists of various signal processing blocks with a common data memory as B-RAM and with a segregated memory space within the same B-RAM block for different types of data like filter coefficients, input samples and output samples etc.

SHAKTI E-CLASS SoC: PARASHU

The SHAKTI E-Class PARASHU variant's architecture features an E-Class microprocessor core that connects to both AXI4 fabric and extends to AXILite fabric.

External peripherals and hardware accelerator IPs can be integrated with either the AXI4 or AXILite fabrics.



The architectural details of the PARASHU variant

• Processor: A 3-stage pipelined, single core and single-thread processor

• Memory: DDR3 of 256 MB • ROM: 4 KB • PWM controllers (06 nos.)

• SPI controller (02 nos.) • UART controllers (02 nos.)

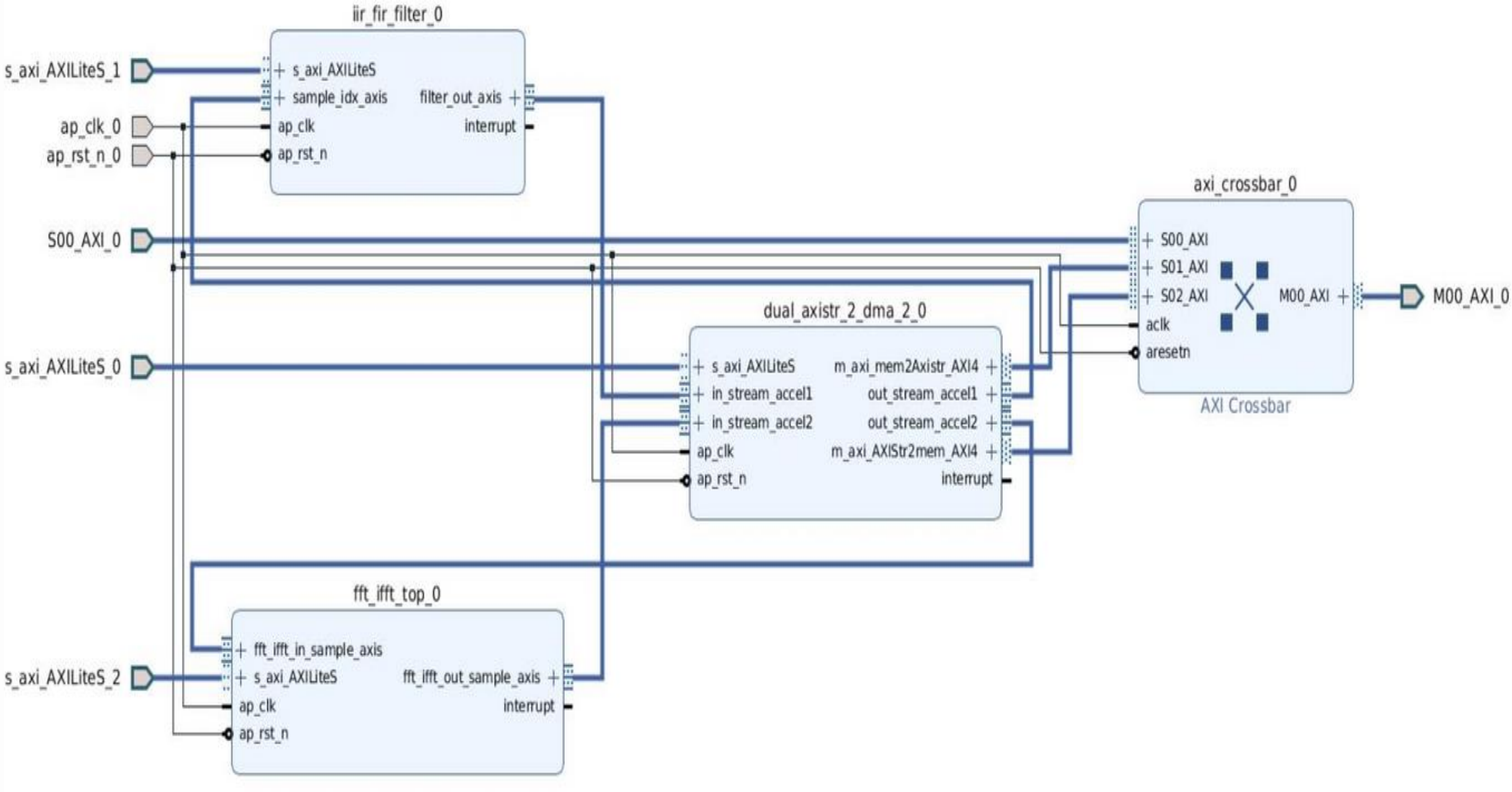
• I2C controllers (02 nos.) • GPIOs (32 nos.) etc

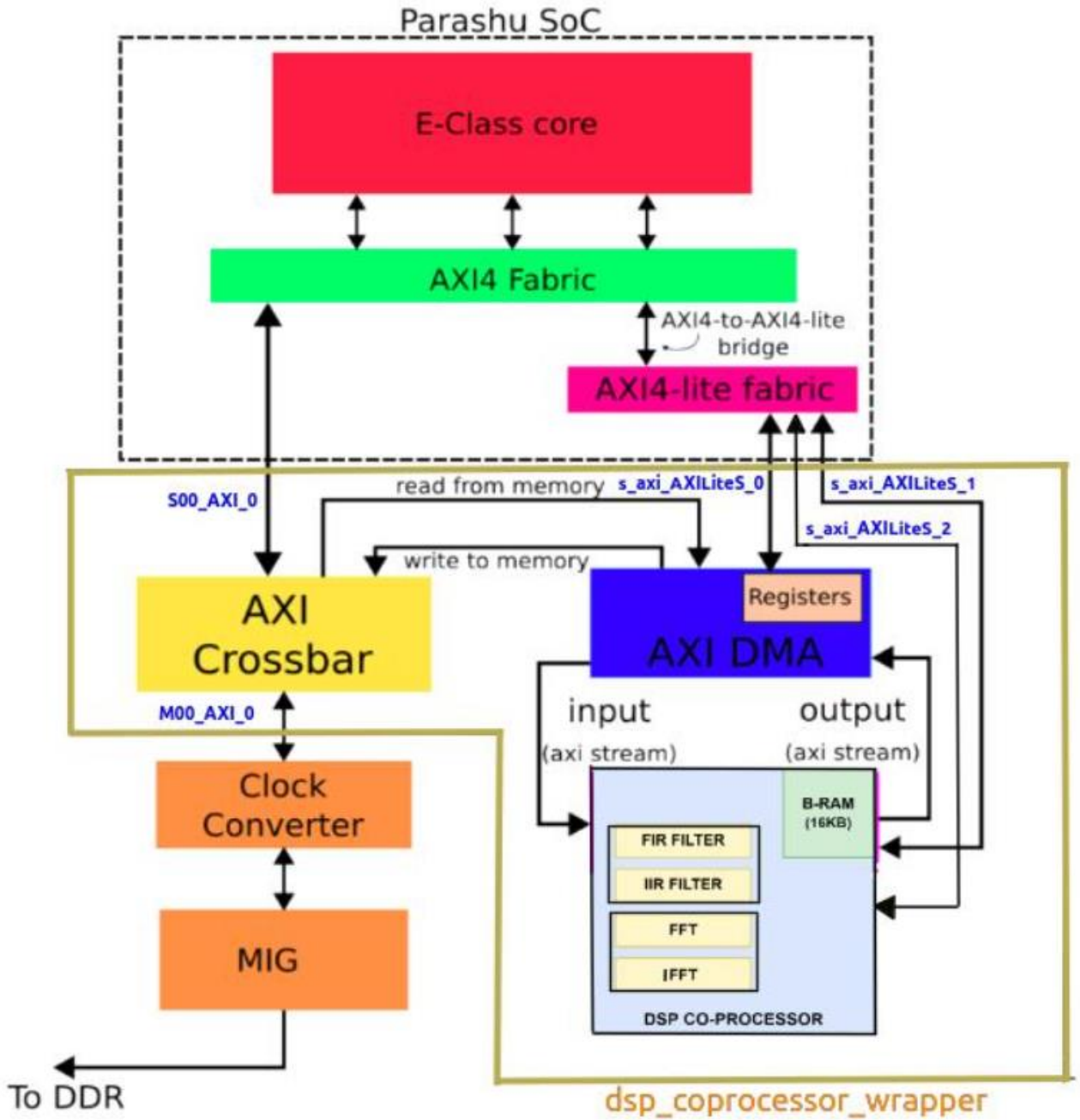
The integration of the Co-Processor with SHAKTI is integrated along a DMA controller

which allows the DSP Co-Processor to access memory directly and independently through AXI protocol.

The PARASHU SoC is compatible with Arty-7 100T FPGA board. The board consists of 63,400 LUT,

hence, the DSP Co-Processor is modified and added with specific blocks to fit it within the FPGA along with the PARASHU SoC.


iir_fir_filter_0
s_axi_AXILiteS_1
s_axi_AXILiteS
sample_idx_axis
filter_out_axis
ap_clk_0
ap_clk
interrupt
ap_rst_n_0
ap_rst_n
S00_AXI_0
axi_crossbar_0
S00_AXI
S01_AXI
S02_AXI
M00_AXI
M00_AXI_0
aclk
aresetn
AXI Crossbar
dual_axistr_2_dma_2_0
s_axi_AXILiteS_0
s_axi_AXILiteS
m_axi_mem2Axistr_AXI4
in_stream_accel1
out_stream_accel1
in_stream_accel2
out_stream_accel2
ap_clk
m_axi_AXIStr2mem_AXI4
ap_rst_n
interrupt
fft_ifft_top_0
fft_ifft_in_sample_axis
s_axi_AXILiteS_2
s_axi_AXILiteS
fft_ifft_out_sample_axis
ap_clk
interrupt
ap_rst_n


Parashu SoC
E-Class core
AXI4 Fabric
AXI4-to-AXI4-lite
bridge
AXI4-lite fabric
S00_AXI_0
read from memory
s_axi_AXILiteS_0
s_axi_AXILiteS_1
s_axi_AXILiteS_2
write to memory
AXI
Crossbar
Registers
AXI DMA
input
(axi stream)
output
(axi stream)
M00_AXI_0
Clock
Converter
B-RAM
(16KB)
FIR FILTER
IIR FILTER
FFT
IFFT
DSP CO-PROCESSOR
MIG
To DDR
dsp_coprocessor_wrapper

FPGA vs ASIC design (automation)

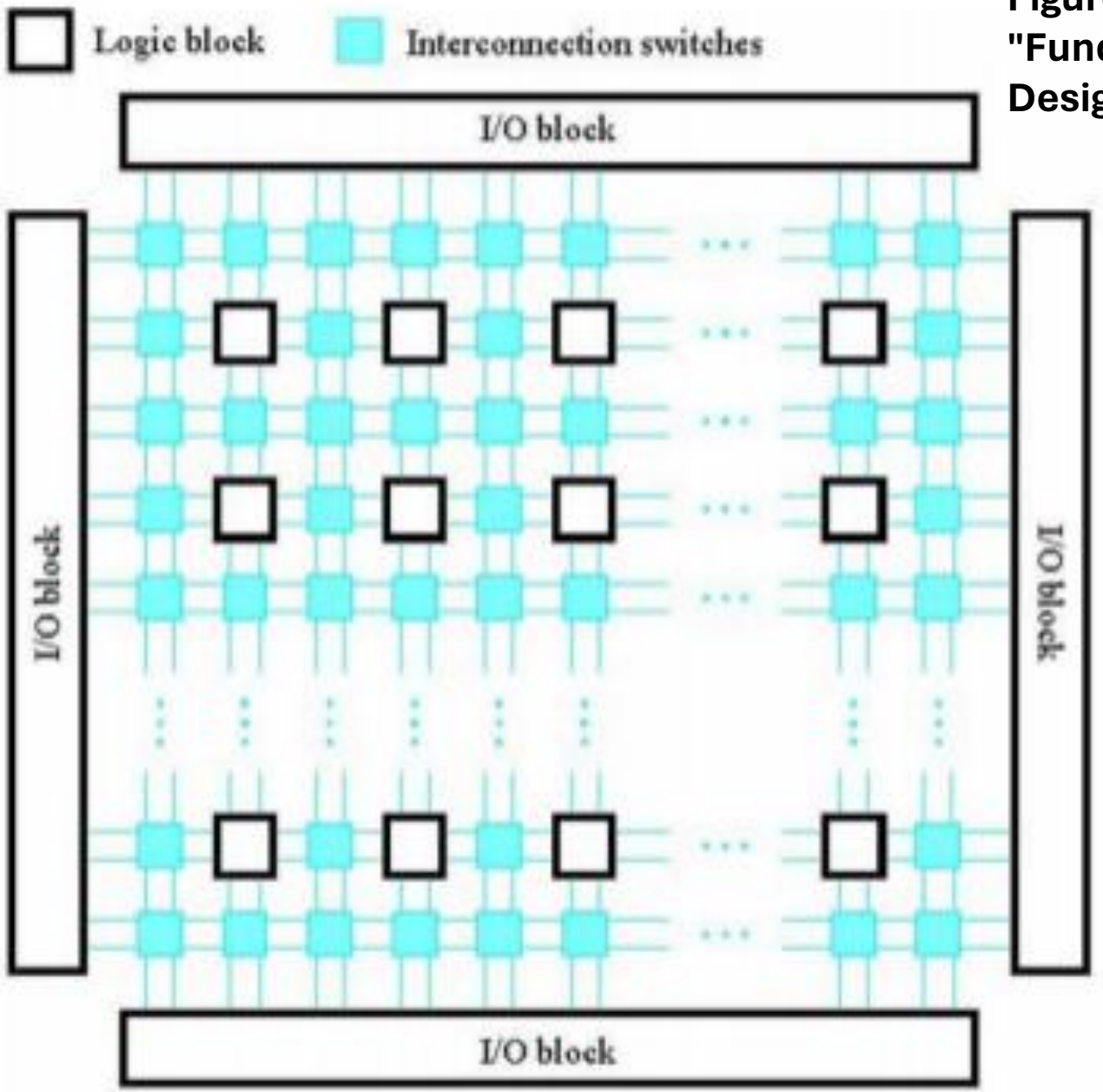
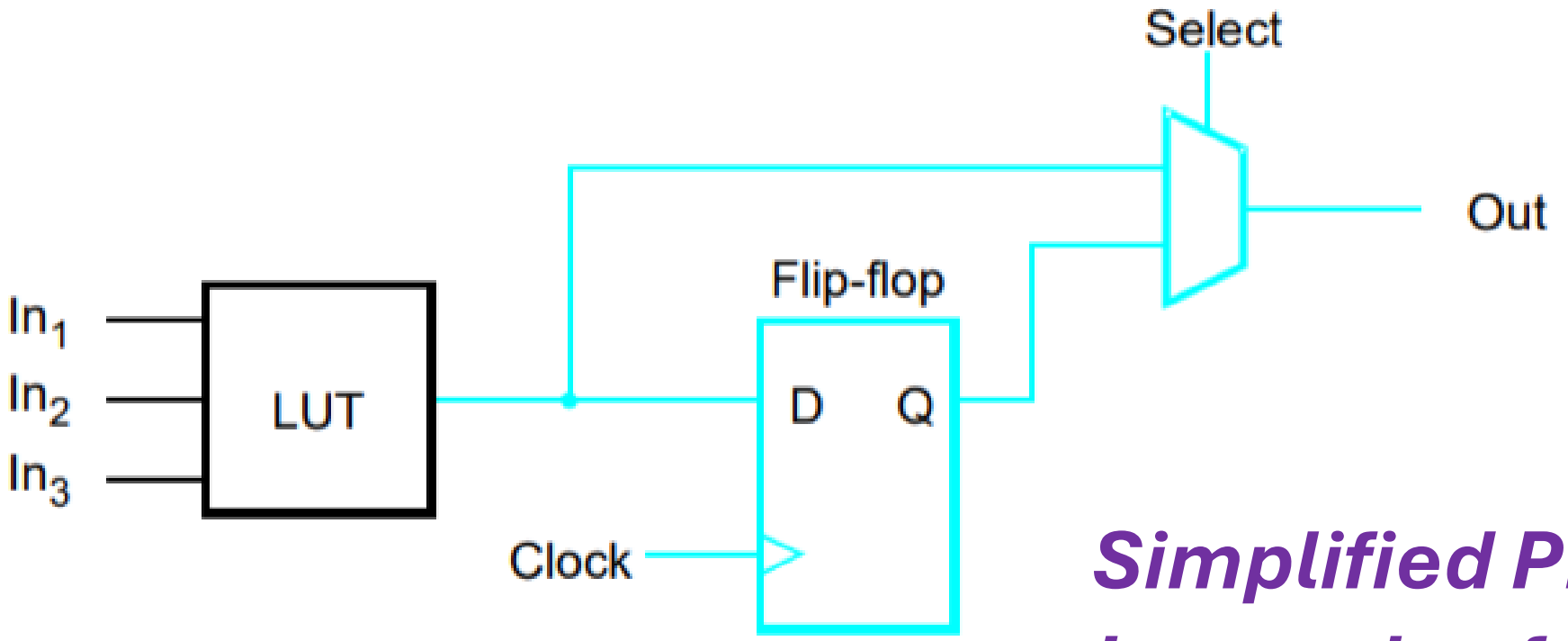


Figure courtesy Brown and Vanesic's "Fund. Of Digi Logic with VHDL/Verilog Design"

FPGA

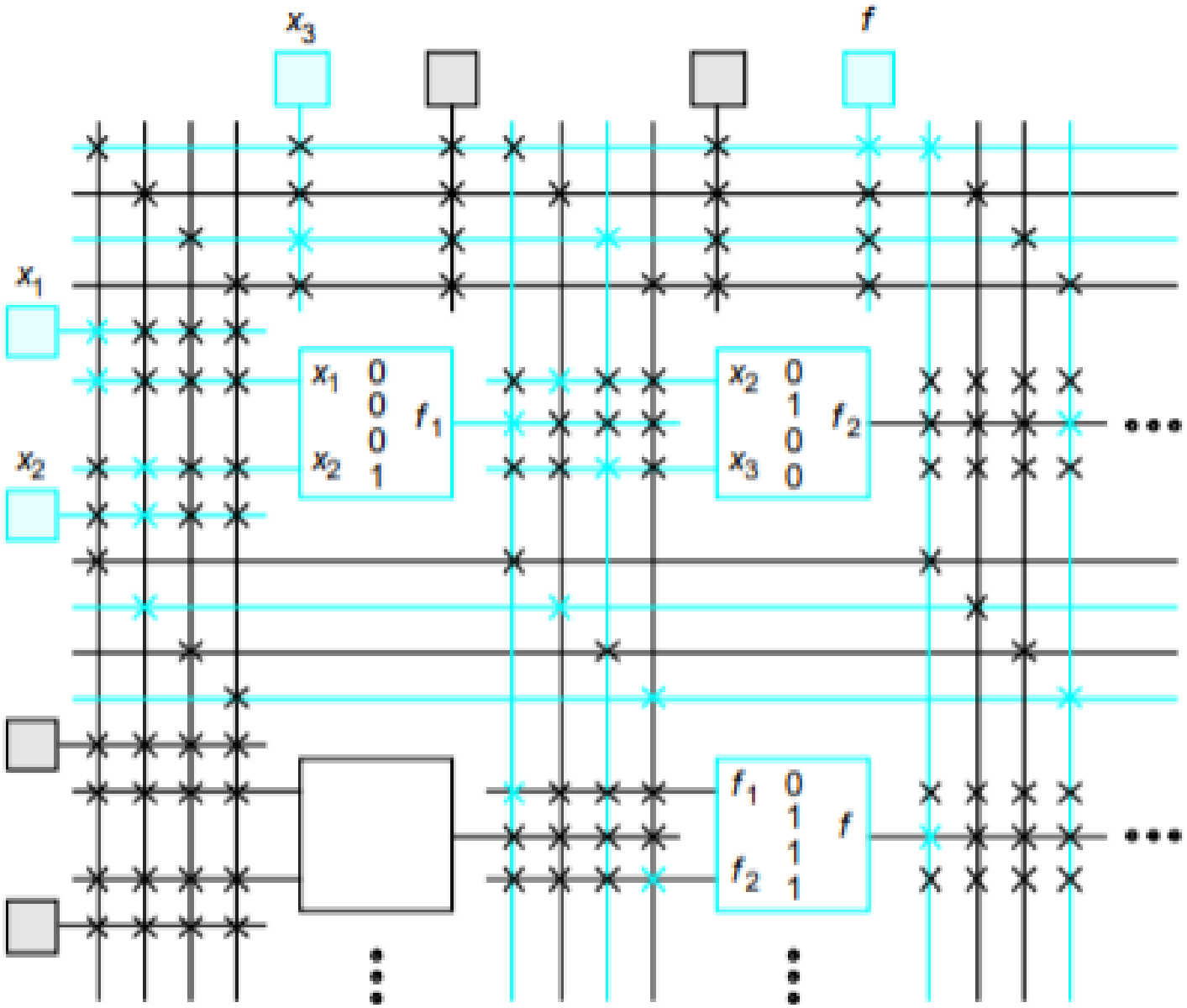
Figure courtesy Brown and Vanesic's "Fund. Of Digi Logic with VHDL/Verilog Design"



Simplified Picture of Internals of CLB of an (SRAM based) FPGA

Programmable Logic Device (PLD) e.g. FPGA

- FPGA : field programmable gate array
- Contains an array of pre-placed, prefabricated configurable logic elements
  - LUT (look-up-tables) that can be configured into different combinational logic blocks
  - FFs (flops) that can be configured as DFF / TFF , with or without enable-mechanism, with synchronous / asynchronous reset mechanisms etc.
- These pre-placed, pre-fabricated configurable logic blocks are also connected by a prefabricated signal routing conductor path segments
  - Appropriate paths are stitched together by configuring pre-placed, pre-fabricated switches
  - Switches are essentially pass-transistors whose gate-terminals are driven by a SRAM bit that can be configured to hold 0 or 1



Result of Place-n-Route

Figure courtesy Brown and Vanesic's "Fund. Of Digi Logic with VHDL/Verilog

Consider this simple VHDL code to follow through a simple illustration of HDL-to-IC flow

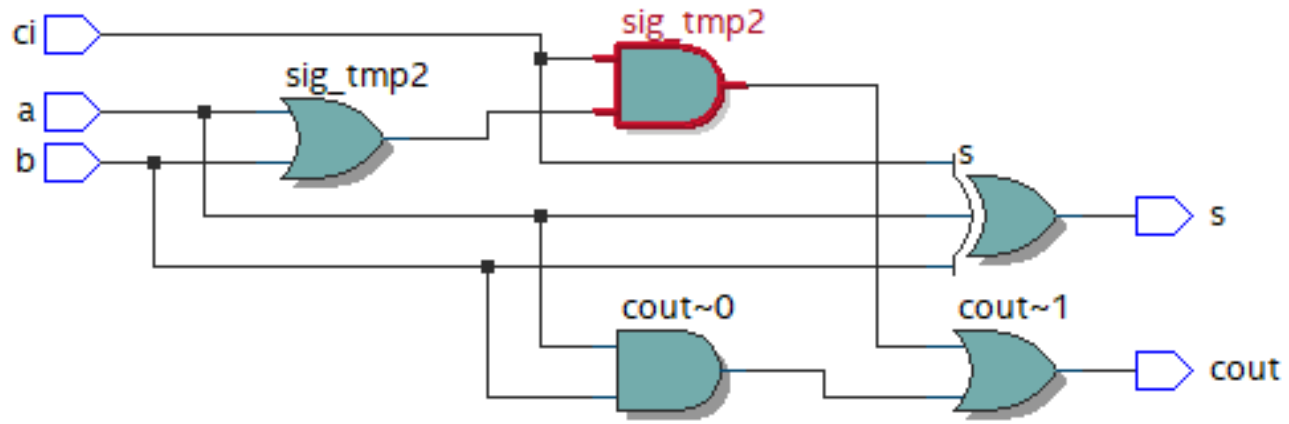
```
 1
 2  entity fulladder is
 3      port ( a, b, ci : in bit ;
 4                 s, cout : out bit ) ;
 5  end entity ;
 6
 7  architecture rch_FA_1 of fulladder is
 8      signal sig_tmp1, sig_tmp2 : bit := '0' ;
 9  begin
10      sig_tmp1 <= a xor b ;
11      s <=  sig_tmp1 xor ci ;
12
13      sig_tmp2 <= ( a or b ) and ci ;
14      cout <= sig_tmp2 or ( a and b ) ;
15  end architecture ;
```

FPGA : steps from HDL code to IC

- Translate
  - Convert the HDL model into a circuit of basic/abstract logic elements or logic equations
- Map / Technology-Mapping
  - Convert a circuit into a circuit of logic elements that can be realized on the target platform
  - Collection of logic elements on the target platform is called "technology library"
- Place and Route
  - Placement and Signal Routing for the circuit produced by the Tech-Map step
- Configuration Stream (bit-stream) generation
  - Different patches of this bitstream would configure different configurable logic elements of the PLD / fpga into the required logic element.
  - e.g a 3-input LUT can be configured to represent any of the 2**8 3-input boolean logic functions.
- Downloading the configuration stream on the programmable fpga

```
 1
 2   entity fulladder is
 3      port ( a, b, ci : in bit ;
 4                s, cout : out bit ) ;
 5   end entity ;
 6
 7   architecture rch_FA_1 of fulladder is
 8      signal sig_tmp1, sig_tmp2 : bit := '0' ;
 9   begin
10      sig_tmp1 <= a xor b ;
11      s <=  sig_tmp1 xor ci ;
12
13      sig_tmp2 <= ( a or b ) and ci ;
14      cout <= sig_tmp2 or ( a and b ) ;
15   end architecture ;
```

Translate step (first step of synthesis)



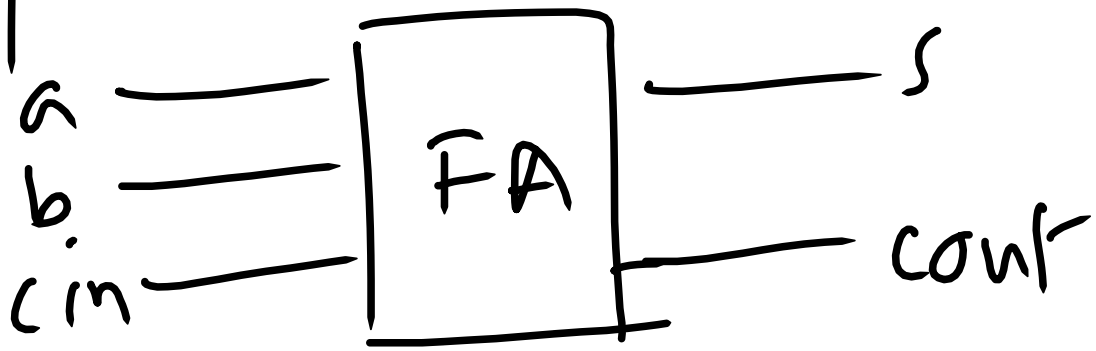
Logic synthesis flow

- Technology independent mapping : Map RTL to netlist of **abstract** logic gates
  - (which need not be same as those available as primitives on the **target technology**, e.g. fpga, standard-cells-based-asic)
- Technology (dependent) mapping (**TechMap**)
  - Takes in output from tech-indep-mapping, as a graph netlist
  - Target-Tech-library cells as a set of **pattern** graphs
  - Obtain efficient mapping / covering of the abstact-netlist-graph by pattern subgraphs

Tech Map

Translate Step figured out the abstract logic circuit in terms of equations or basic primitives

e.g

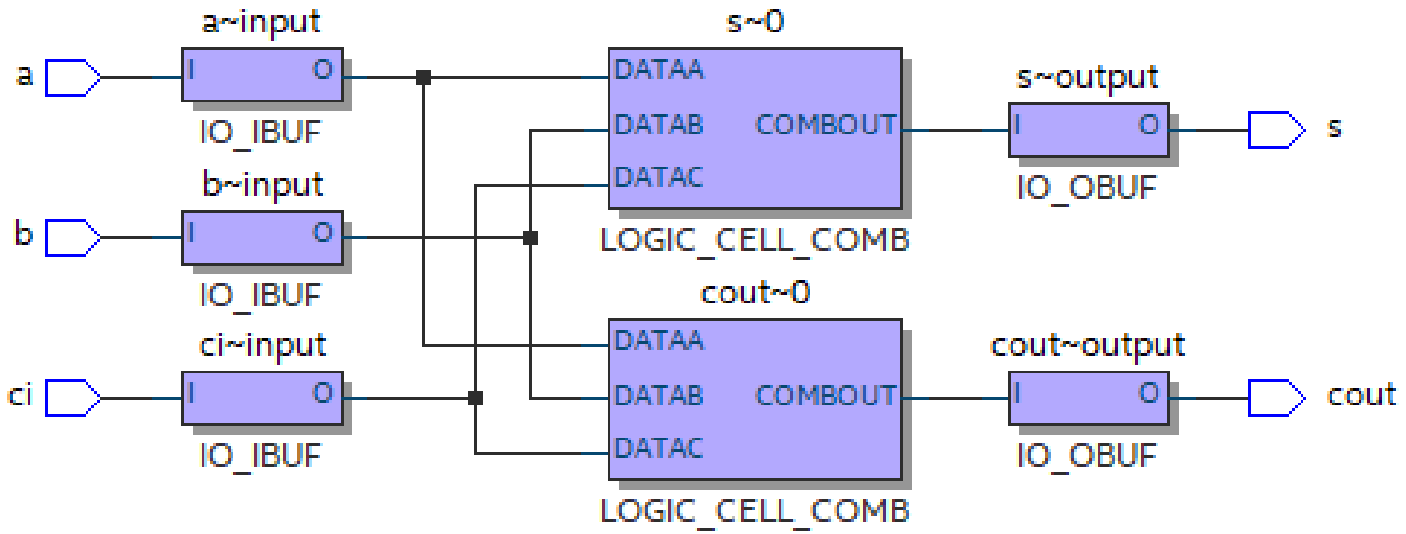


Translate step :

Converts the HDL model into a circuit of basic/abstract logic elements or logic equations

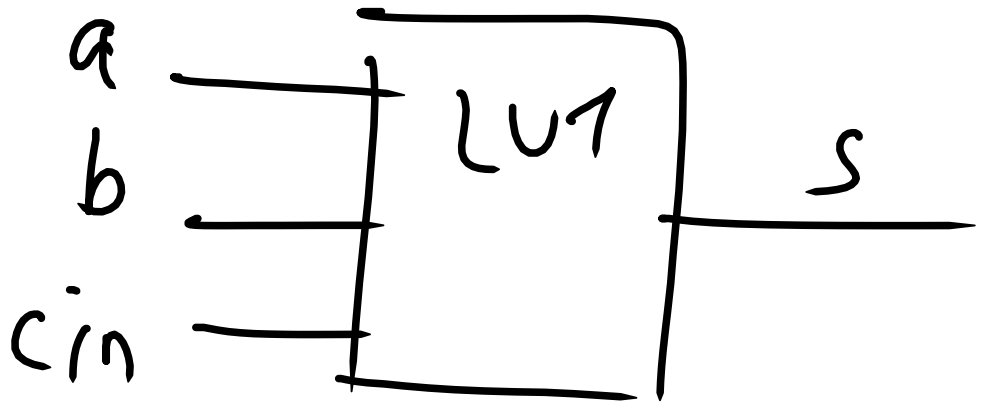
$$S = a \oplus b \oplus cin$$

$$Cout = majority3(a, b, cin)$$

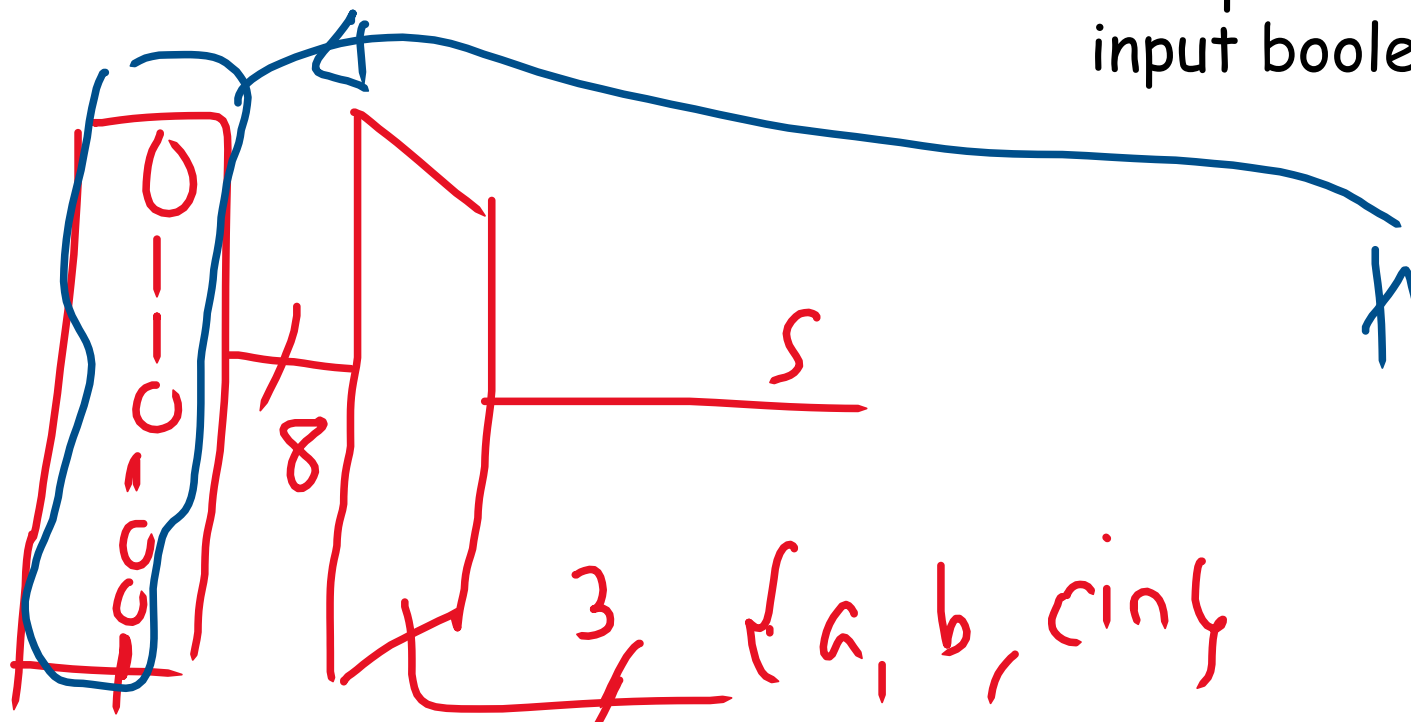


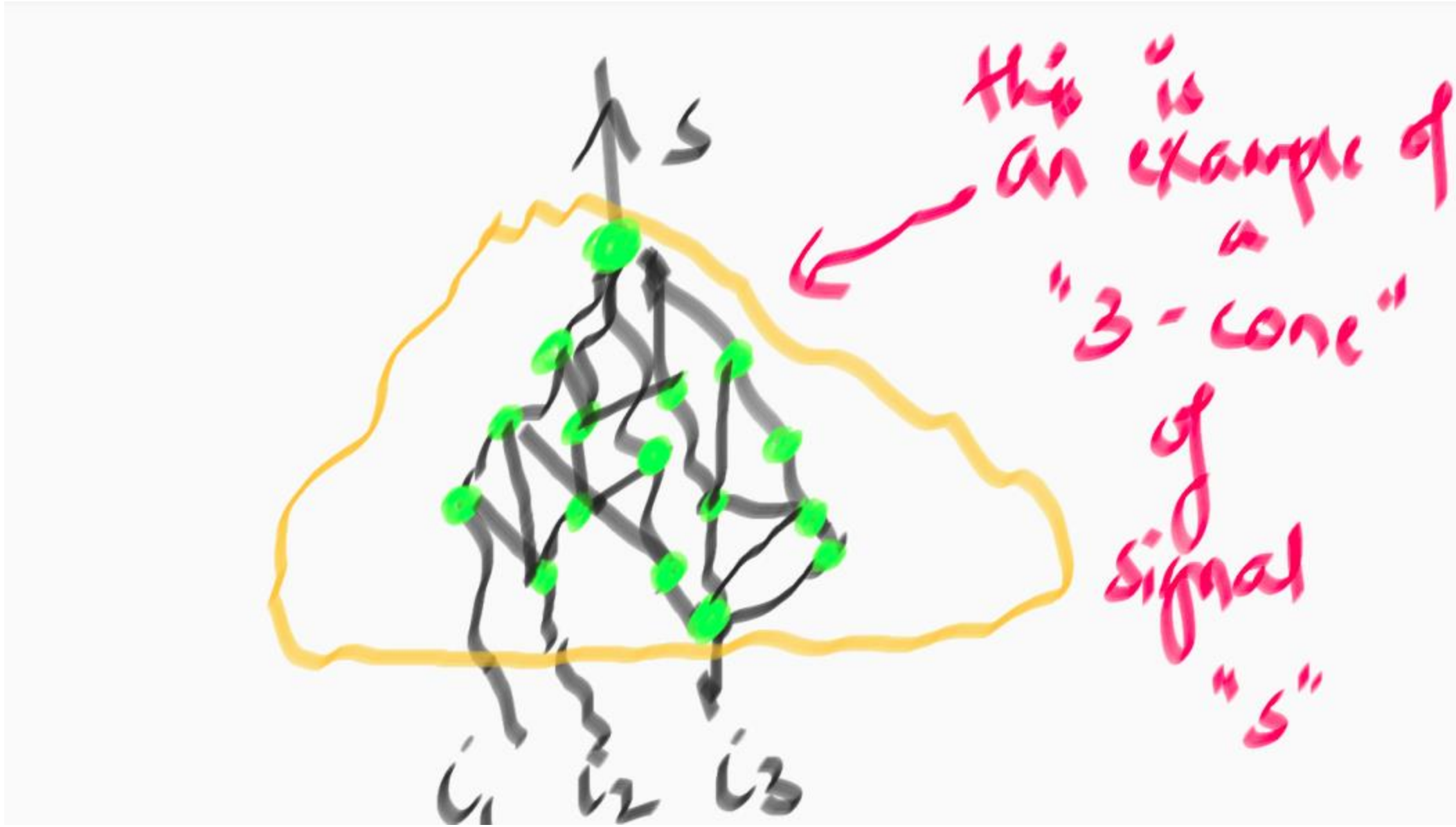
Tech-Map-for-intel-FPGA step of Synthesis has prepared a circuit of LUT and FFs

```
1
2   entity fulladder is
3       port ( a, b, ci : in bit ;
4              s, cout : out bit ) ;
5   end entity ;
6
7   architecture rch_FA_1 of fulladder is
8       signal sig_tmp1, sig_tmp2 : bit := '0' ;
9   begin
10      sig_tmp1 <= a xor b ;
11      s <=  sig_tmp1 xor ci ;
12
13      sig_tmp2 <= ( a or b ) and ci ;
14      cout <= sig_tmp2 or ( a and b ) ;
15  end architecture ;
```



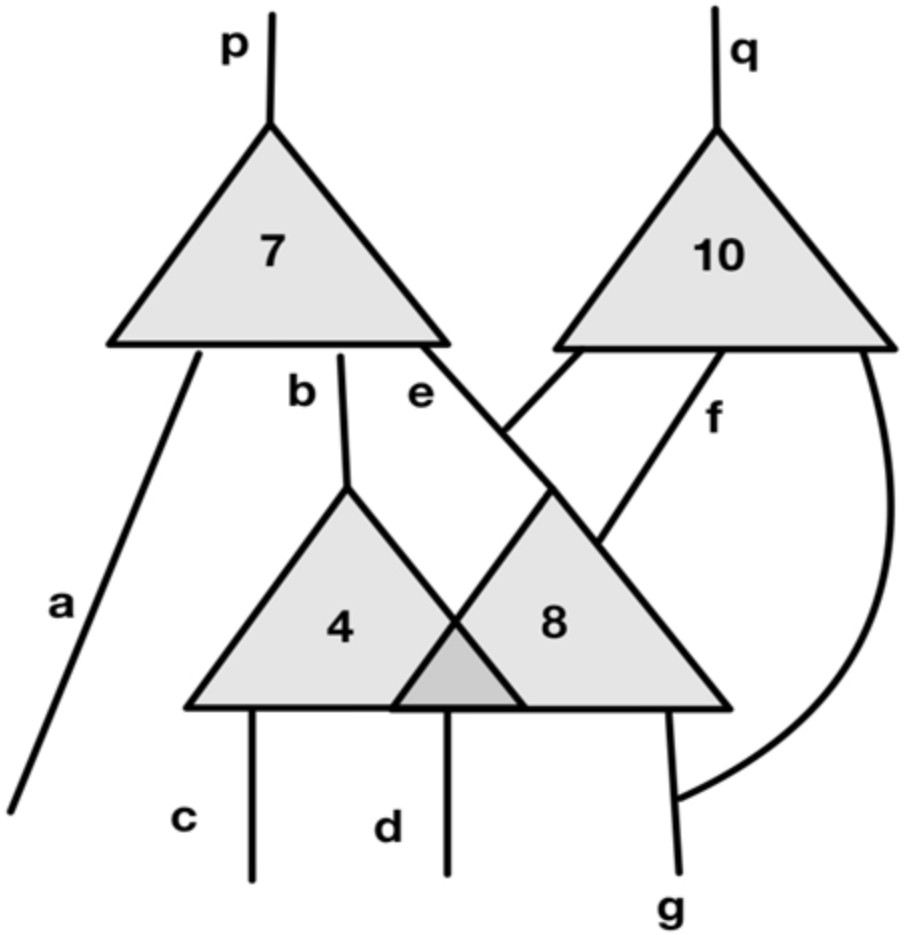
- Different patches of this bitstream would configure different configurable logic elements of the PLD / fpga into the required logic element.
- e.g. a 3-input LUT can be configured to represent any of the 2**8 3-input boolean logic functions.



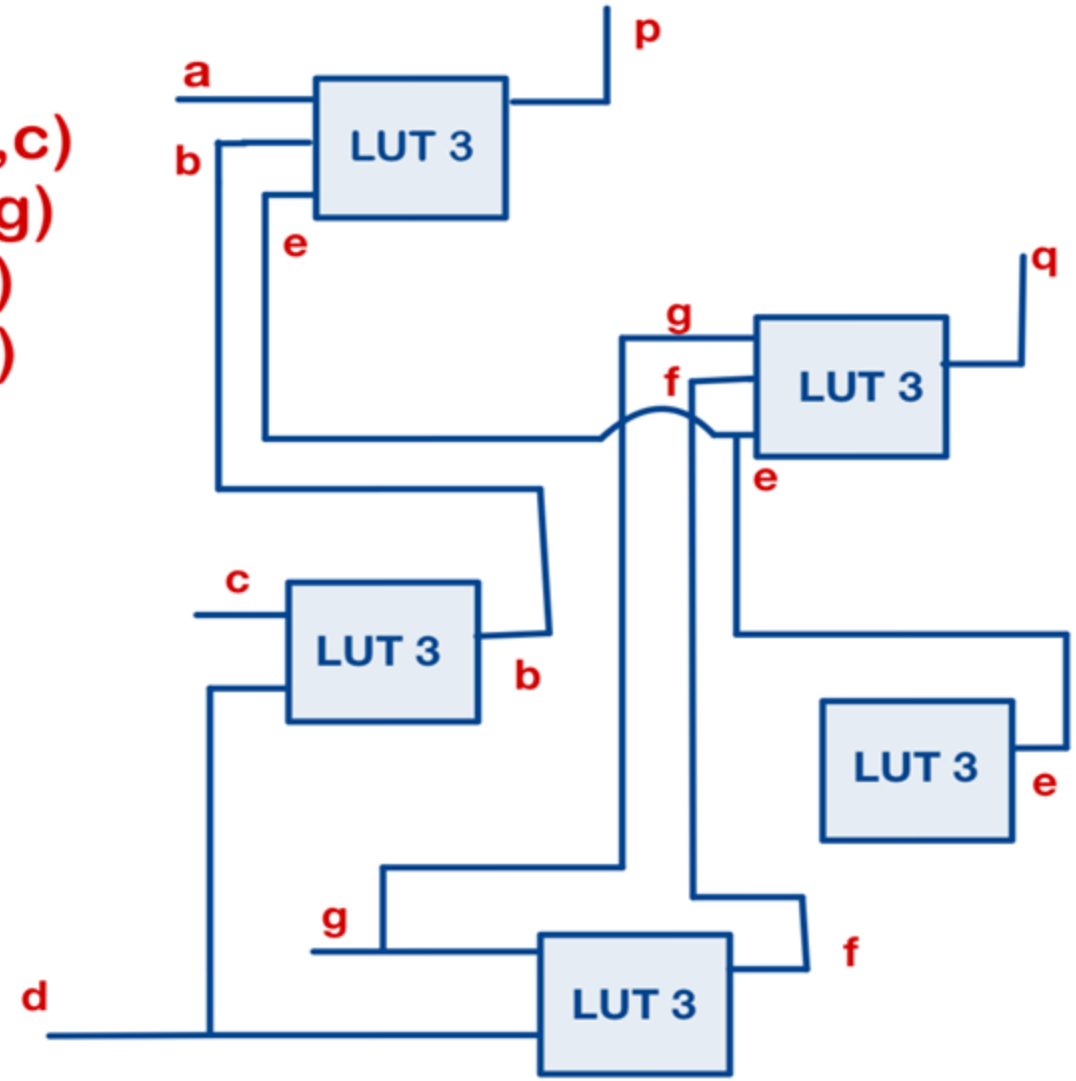

s
i1
i2
i3
This is an example of a "3-cone" of signal "s"

A 3-cone of a signal say 's' is a subcircuit that has at most 3 inputs, say i_1, i_2, i_3 such that all gates on paths from any input to 's' are inside this subcircuit

Tech-mapping algorithms typically find several such 3-cuts at each of the signals in the given full circuit and then cleverly choose as few as possible of these 3-cuts to make a circuit of LUT3



p(a,b,c)
q(e,f,g)
b(c,d)
e(d,g)
f(d,g)

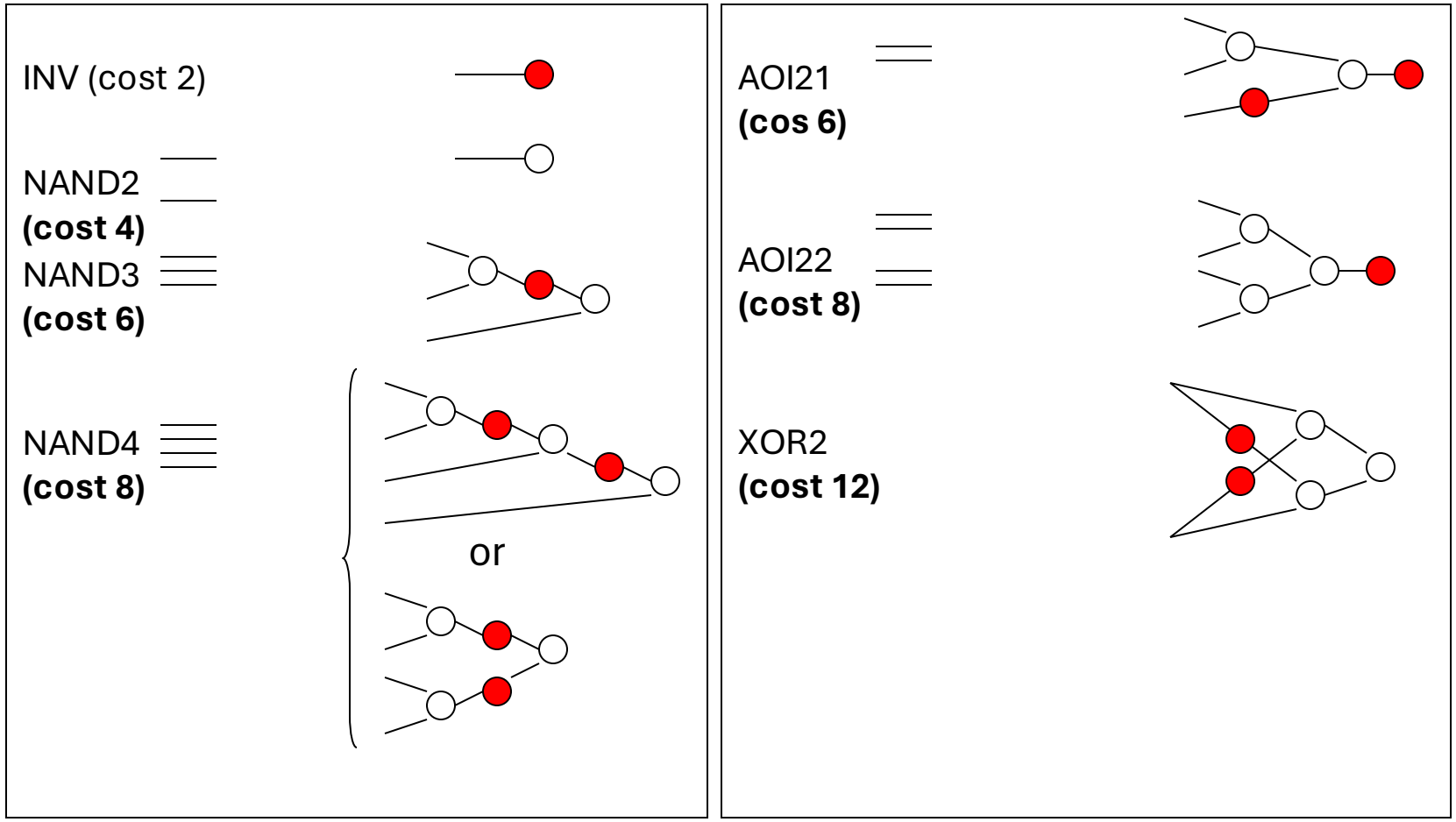


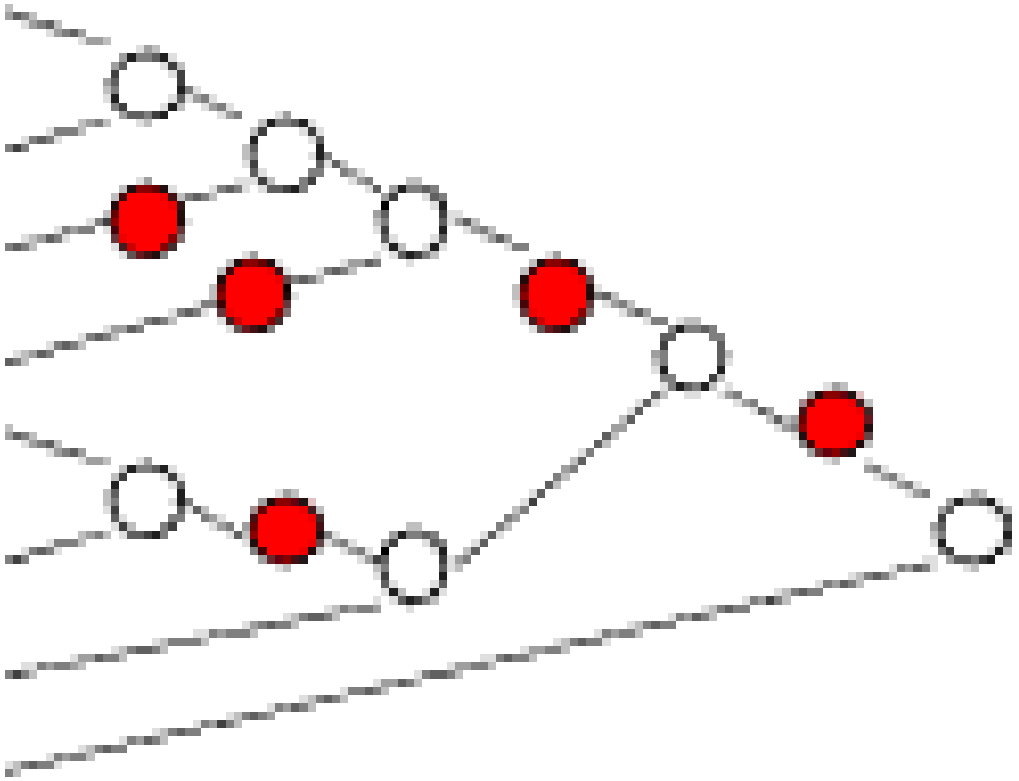
Circuit with 29 Gates

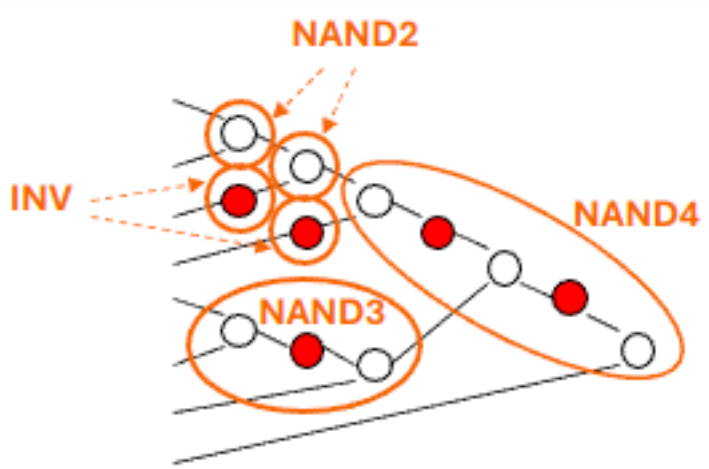
Circuit with 5 LUT-3's

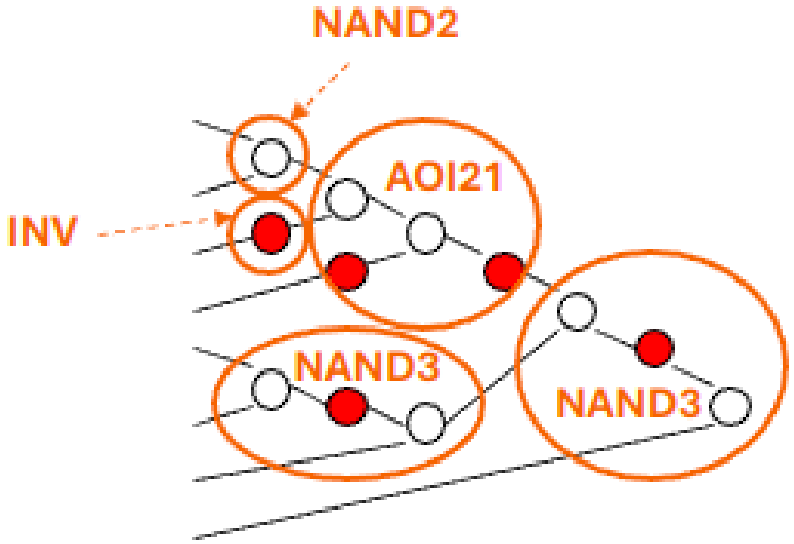
TechMap for standard-cells-based-ASIC design

pattern graphs / trees of target-tech-library-cells



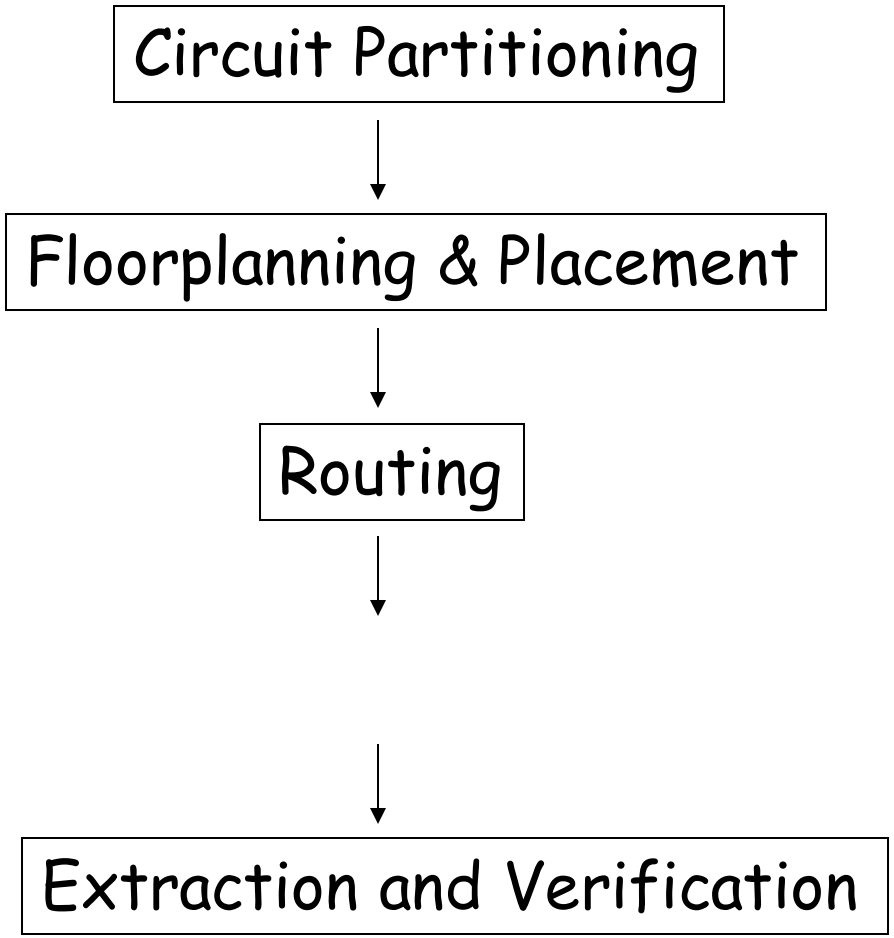



NAND2
INV
NAND4
NAND3


NAND2
AOI21
INV
NAND3
NAND3

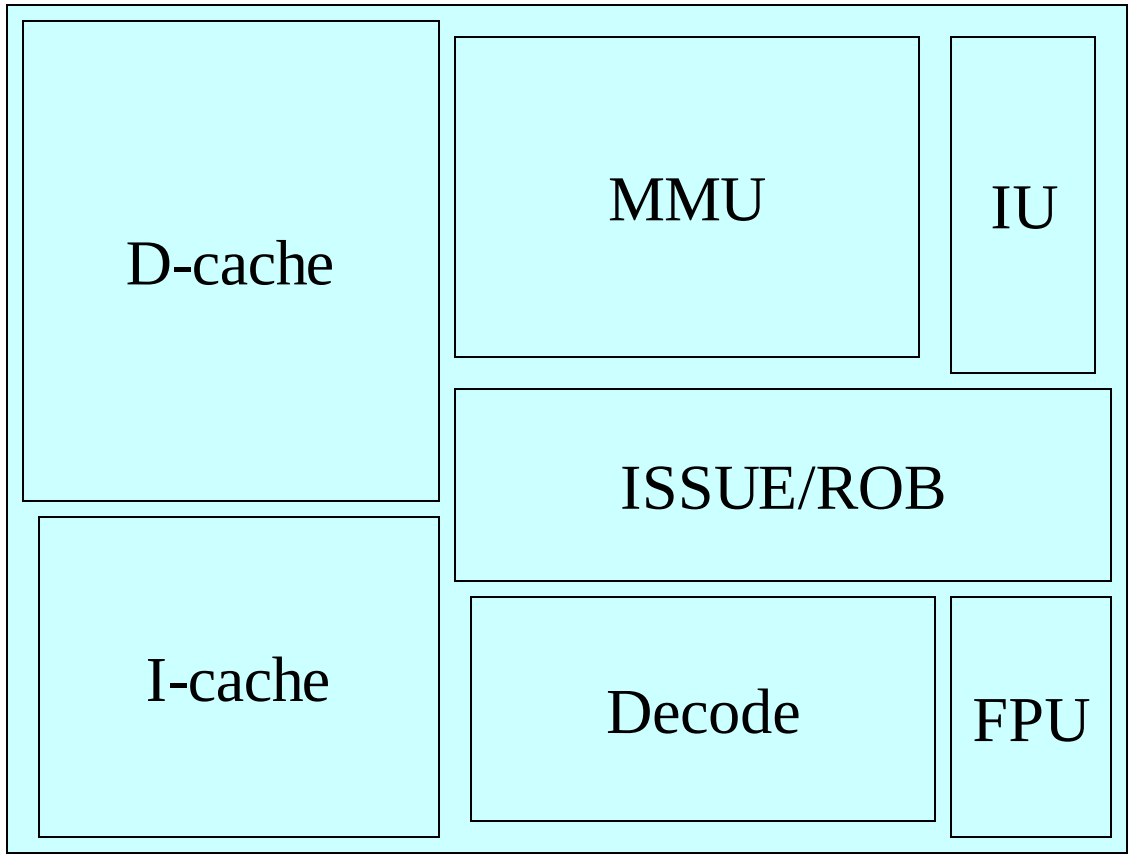
Algorithms for VLSI Layout Automation

Physical Design Cycle

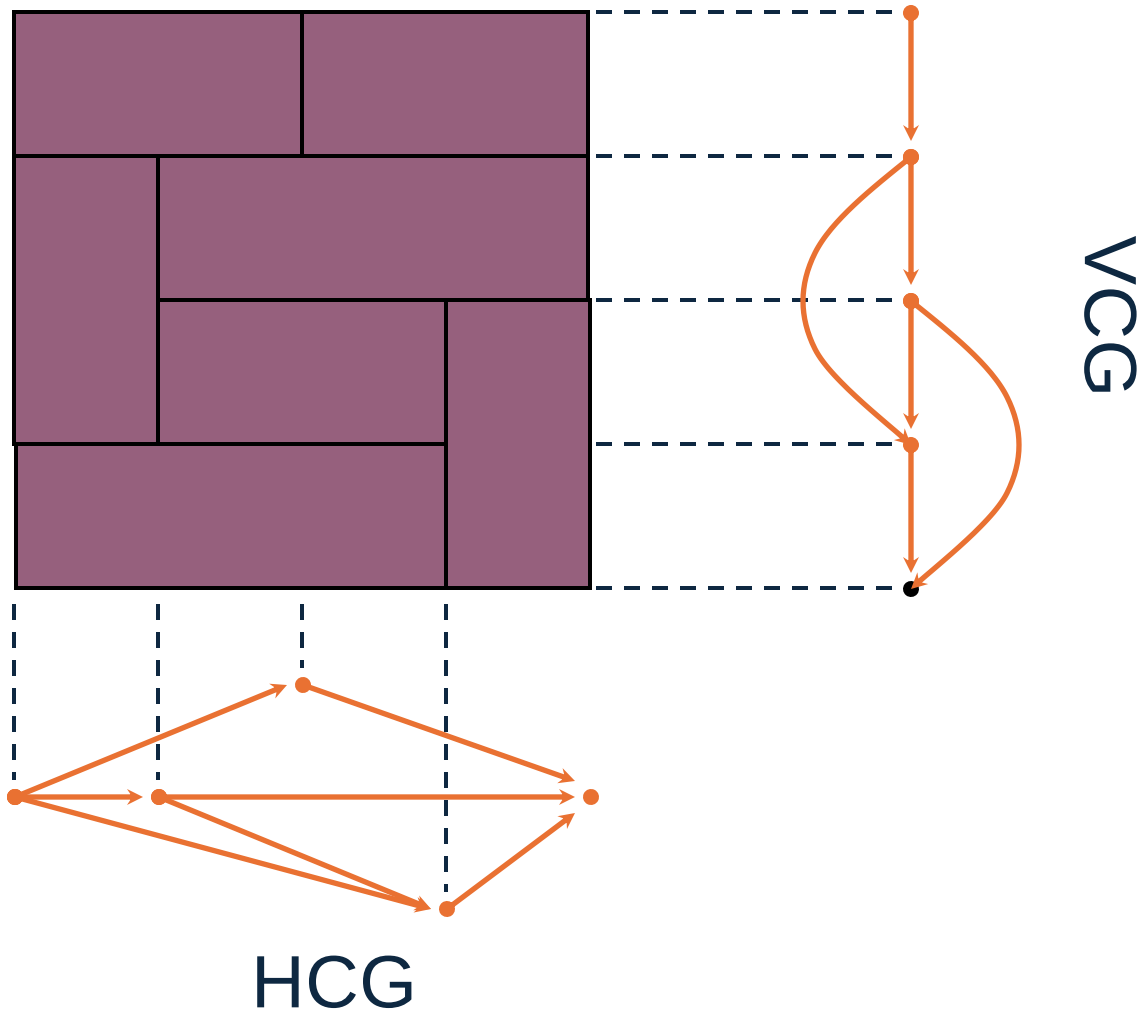


Tools in OpenROAD toolchain

- Verilog to DB (dbSTA)
- OpenDB: Open Database (odb)
- TritonPart: constraints-driven paritioner (par)
- Floorplan Initialization (ifp)
- ICeWall chip-level connections (pad)
- I/O Placement (ppl)
- PDN Generation (pdn)
- Tapcell and Welltie Insertion (tap)
- Triton Macro Placer (mpl)
- Hierarchical Automatic Macro Placer (mpl2)
- RePlAce Global Placer (gpl)
- Gate resizing and buffering (rsz)
- Detailed placement (dpl)
- Clock tree synthesis (cts)
- FastRoute Global routing (grt)
- Antenna check and diode insertion (ant)
- TritonRoute Detailed routing (drt)
- Metal fill insertion (fin)
- Design for Test (dft)
- OpenRCX Parasitic Extraction (rcx)
- OpenSTA timing/power analyzer (sta
- Graphical User Interface (gui)
- Static IR analyzer (psm)

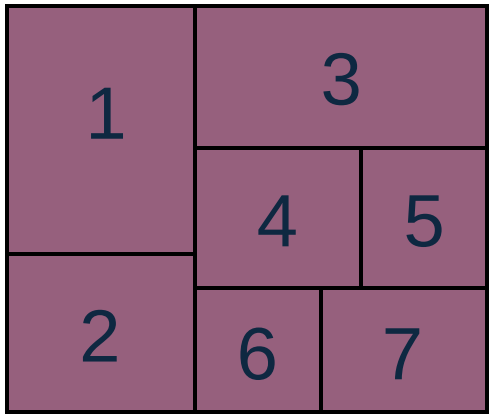

D-cache
MMU
IU
ISSUE/ROB
I-cache
Decode
FPU

HCG, VCG: Example

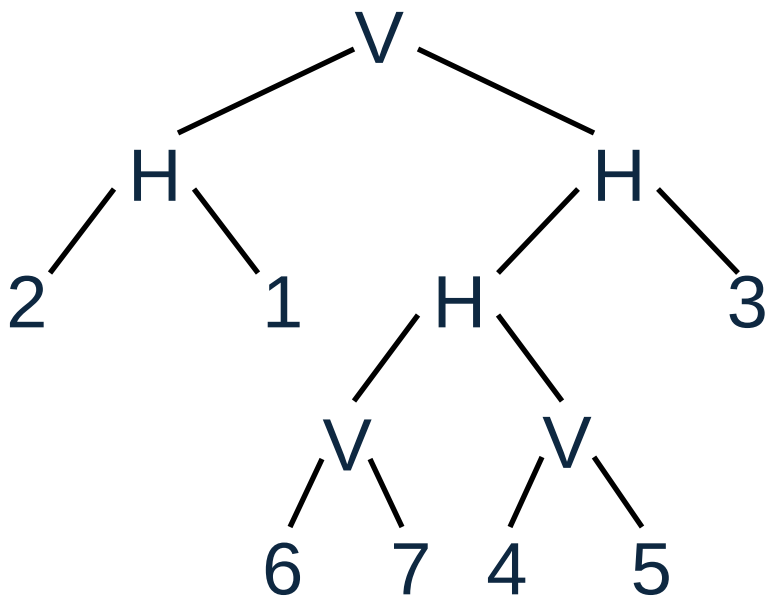


Representation of Slicing Floorplan

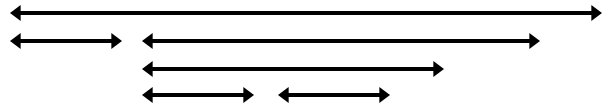
Slicing Floorplan



Slicing Tree



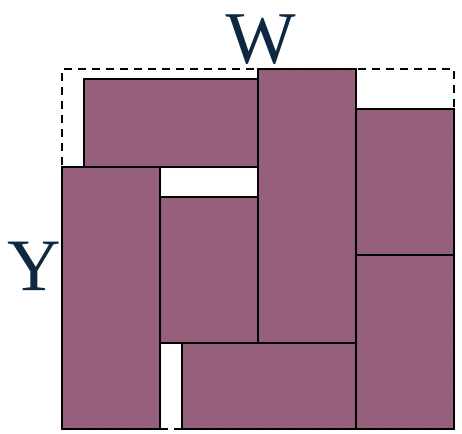
Polish Expression
(postorder traversal
of slicing tree)



21H67V45VH3HV

Floorplan Sizing Problem

- Minimize the packing area:
  - Assume that one dimension W is fixed.
  - Minimize the other dimension Y.
- Constraints should model non-overlap of blocks.
- Associate each block B_i with 4 variables:
  - x_i and y_i: coordinates of its lower left corner.
  - w_i and h_i: width and height.



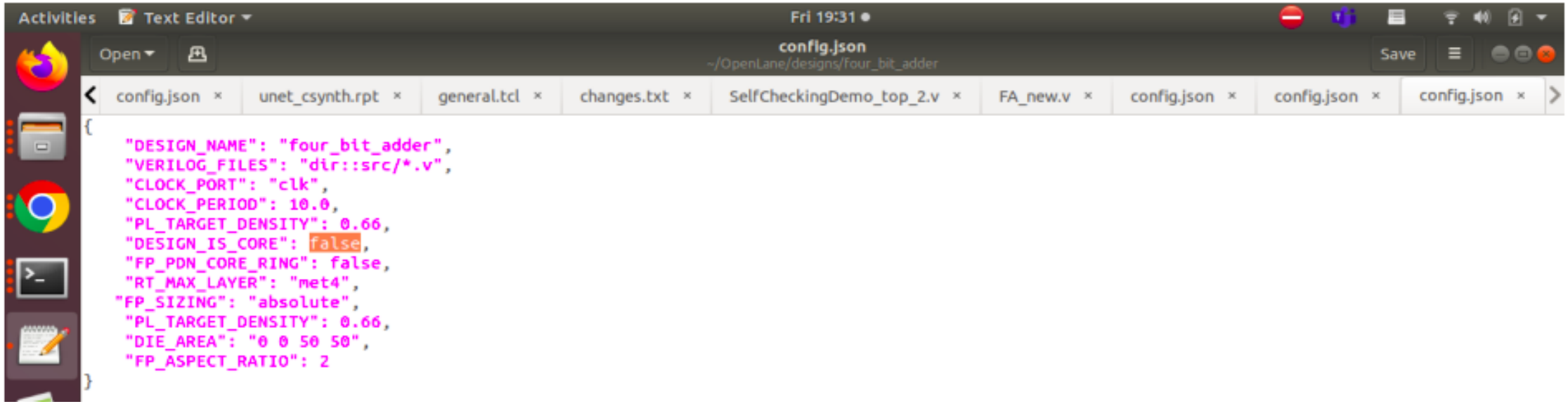
Formulation

Min.	Y	
s.t.	$0 \le x_i, x_i + w_i \le W$	$1 \le i \le n$
	$0 \le y_i, y_i + h_i \le Y$	$1 \le i \le n$
	$x_i + w_i \le x_j + W(x_{ij} + y_{ij})$	$1 \le i < j \le n$
	$x_i - w_j \ge x_j - W(1 + x_{ij} - y_{ij})$	$1 \le i < j \le n$
	$y_i + h_i \le y_j + H(1 - x_{ij} + y_{ij})$	$1 \le i < j \le n$
	$y_i - h_j \ge y_j - H(2 - x_{ij} - y_{ij})$	$1 \le i < j \le n$
	$x_{ij} = 0 \text{ or } 1$	$1 \le i < j \le n$
	$y_{ij} = 0 \text{ or } 1$	$1 \le i < j \le n$


Activities
openroad
Fri 19:29
OpenROAD
File View Tools Windows Options Help
Fit Find Inspect Timing
Display Control
Layers
li1
mcon
met1
via
met2
via2
met3
via3
met4
via4
met5
Nets
Instances
Blockages
0 3 10µm
Inspector
Name
Value
Inspector
Hierarchy Browser
Timing Report
Scripting
OpenROAD 1f720d3b442e2cd8dc6c5372535320b18a105e8d
This program is licensed under the BSD-3 license. See the LICENSE file for details.

Open the config.json file for submodules and make sure the **DESIGN_IS_CORE is set to false** and **FP_PDN_CORE_RING is set to false**

Configuration variable description can be found in the Openlane documentation site



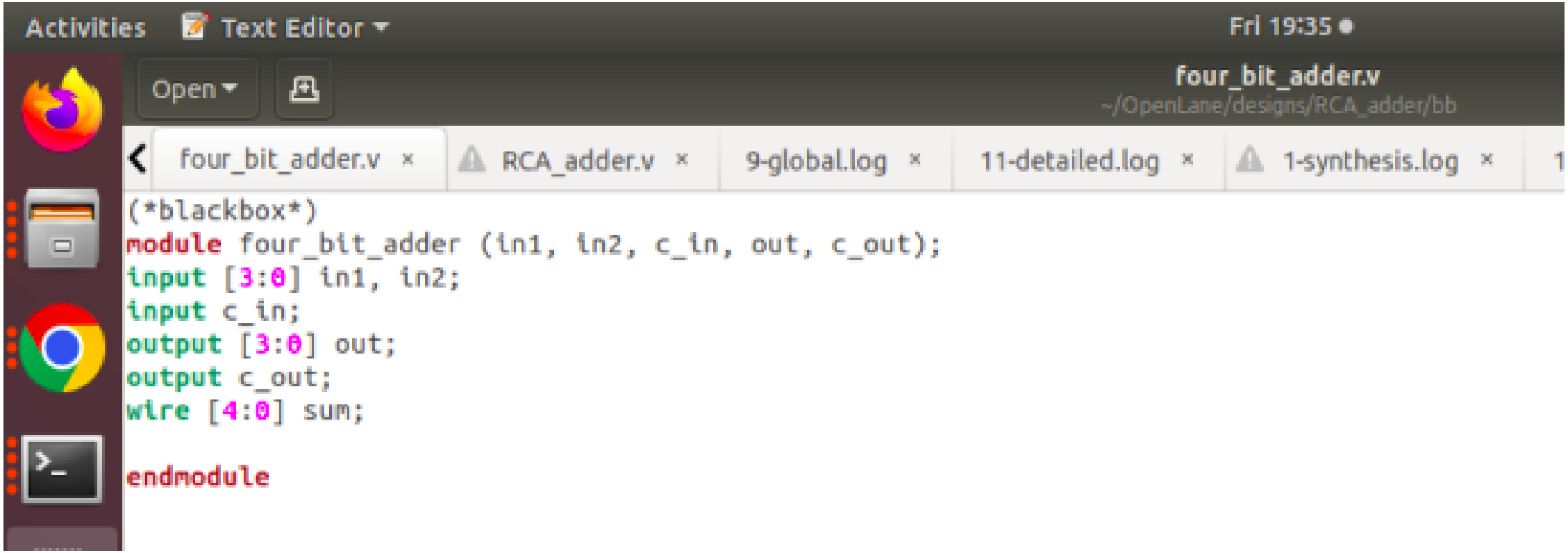
Create RCA_adder design that uses four_bit_adder module.

```
module RCA_adder (in1, in2, c_in, out, c_out, c_int);
  input [15:0] in1, in2;
  input c_in;
  output [2:0] c_int;
  output [15:0] out;
  output c_out ;

  wire [2:0]c_intermediate;
  assign c_int = c_intermediate;

  four_bit_adder dut1(in1[3:0], in2[3:0], c_in, out[3:0], c_intermediate[0]);
  four_bit_adder dut2(in1[7:4], in2[7:4], c_intermediate[0], out[7:4], c_intermediate[1]);
  four_bit_adder dut3(in1[11:8], in2[11:8], c_intermediate[1], out[11:8],
c_intermediate[2]);
  four_bit_adder dut4(in1[15:12], in2[15:12], c_intermediate[2], out[15:12], c_out);
endmodule
```

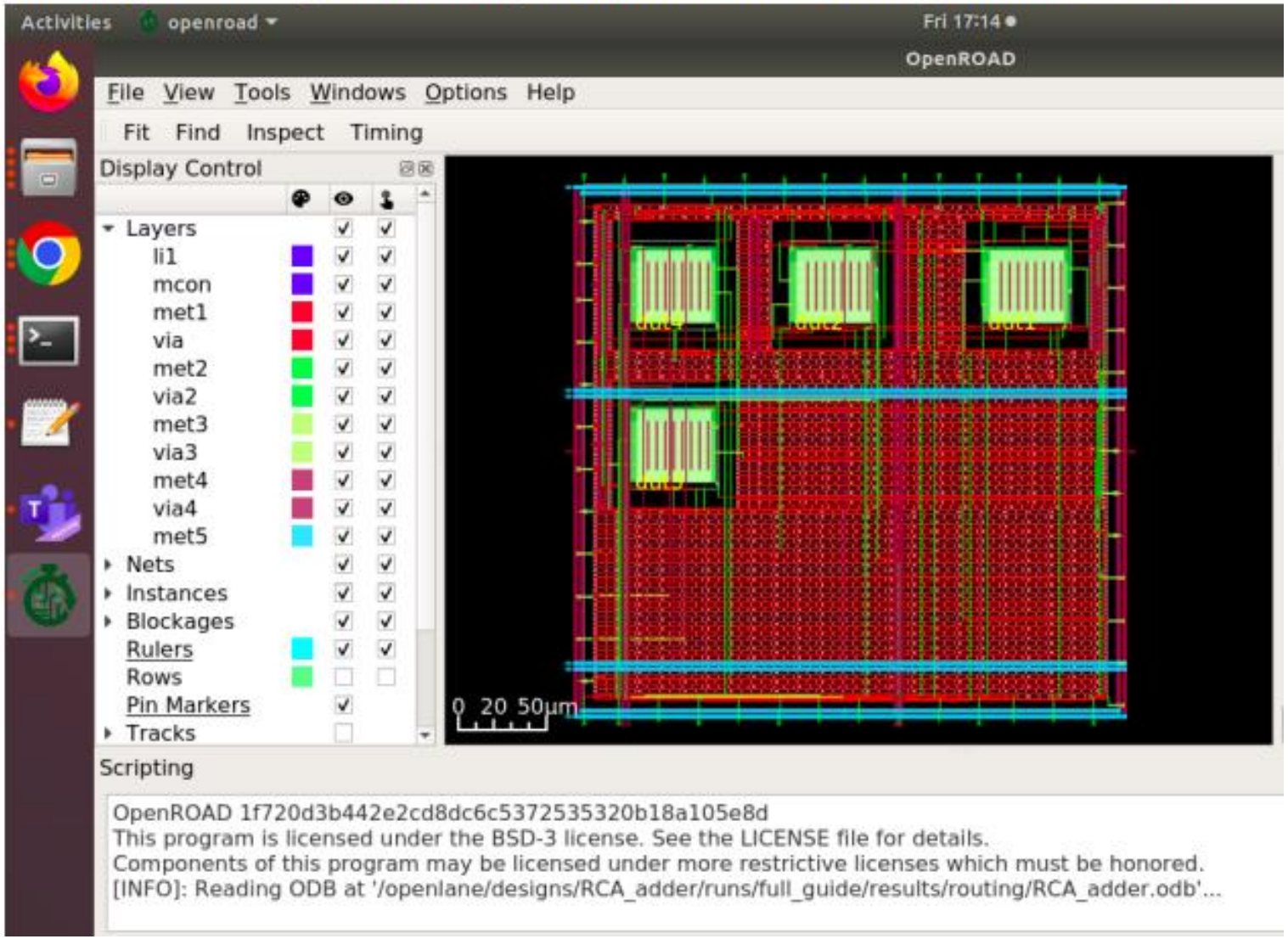
Open the config.json file for top module and make sure the **DESIGN_IS_CORE is set to true** and **FP_PDN_CORE_RING is set to true.**

Copy all the submodules GDS and LEF files in single separate folders (in our case there is a single re-used submodule) as shown and provide the path in the config file.

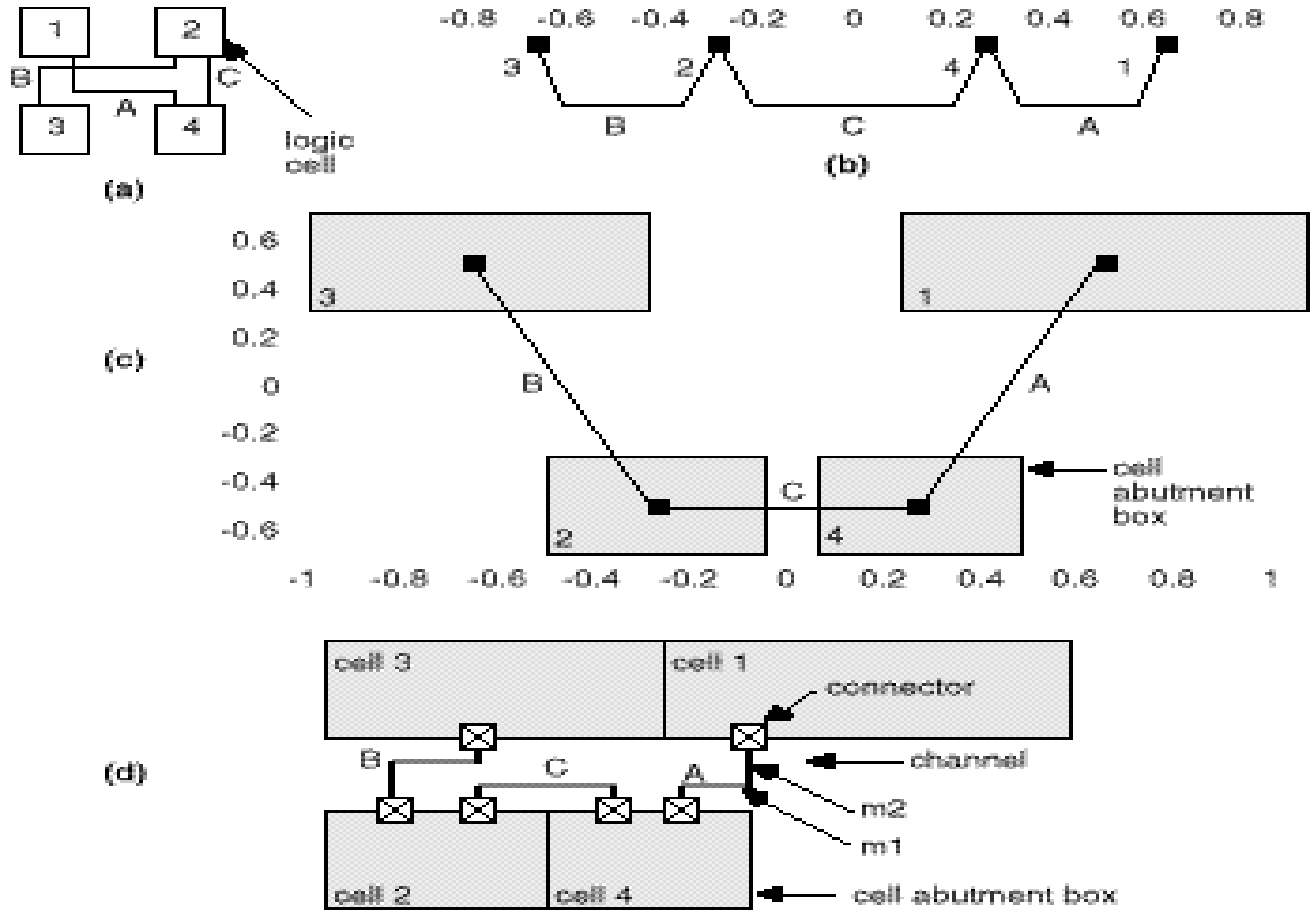
Also, copy the verilog code of the submodule and change the code to be a black box.



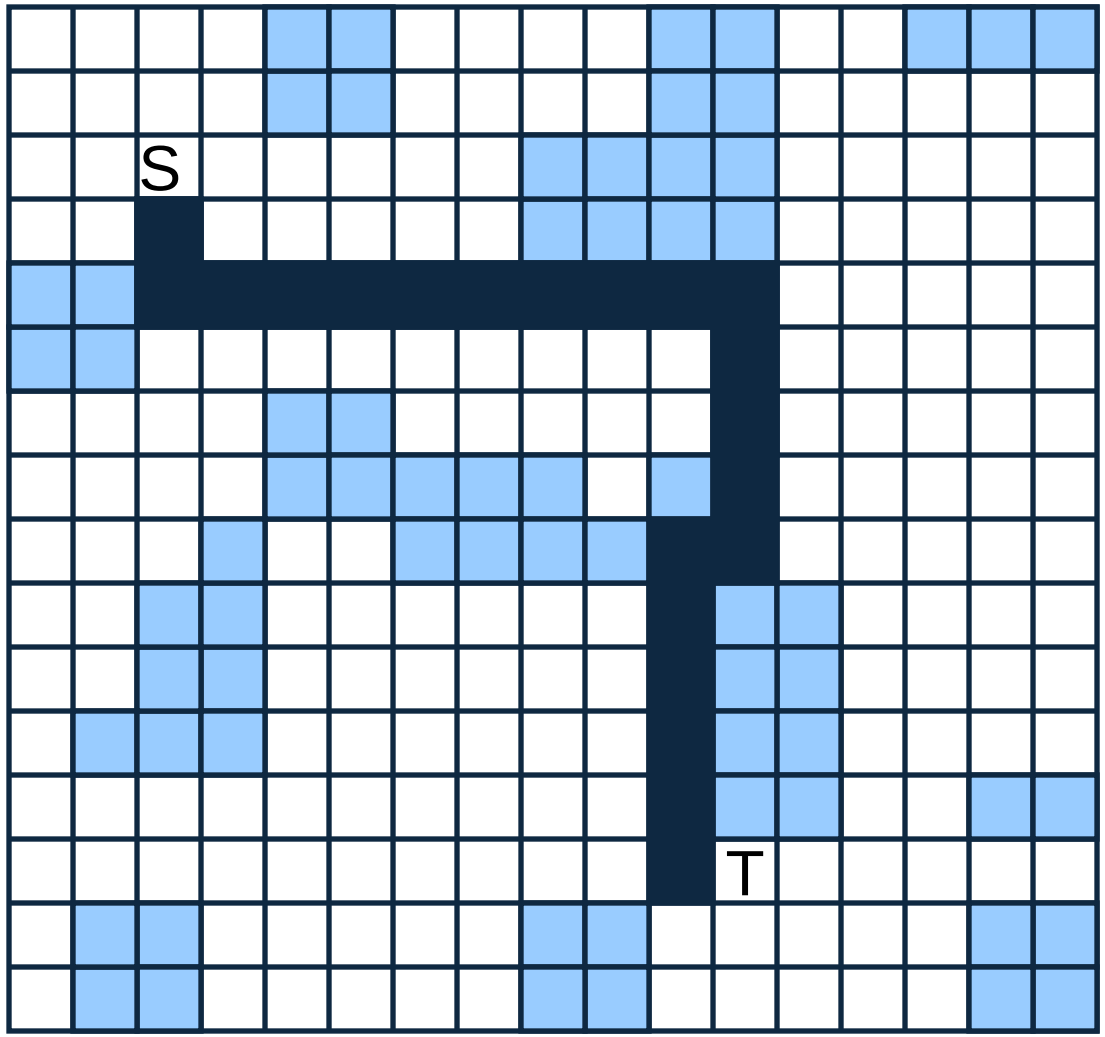
See the generated layout containing individual instances of four_bit adder macros.



16.2.5 Eigenvalue Placement Example

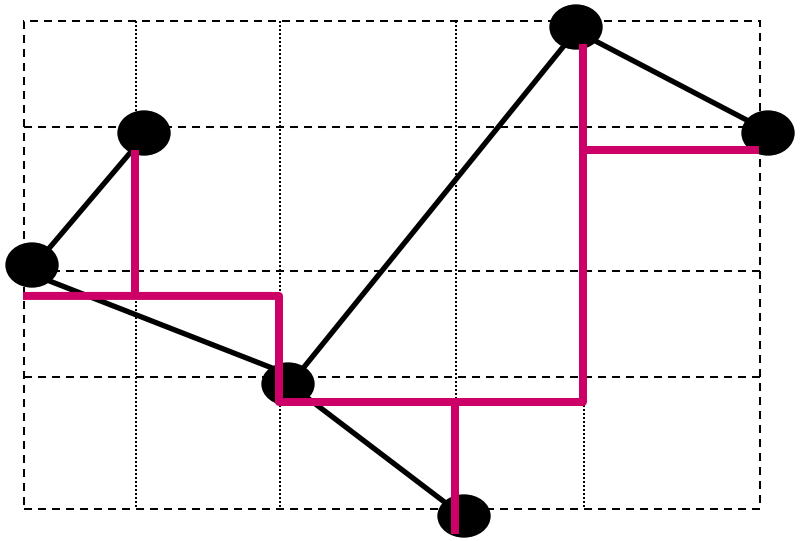


Maze Routing

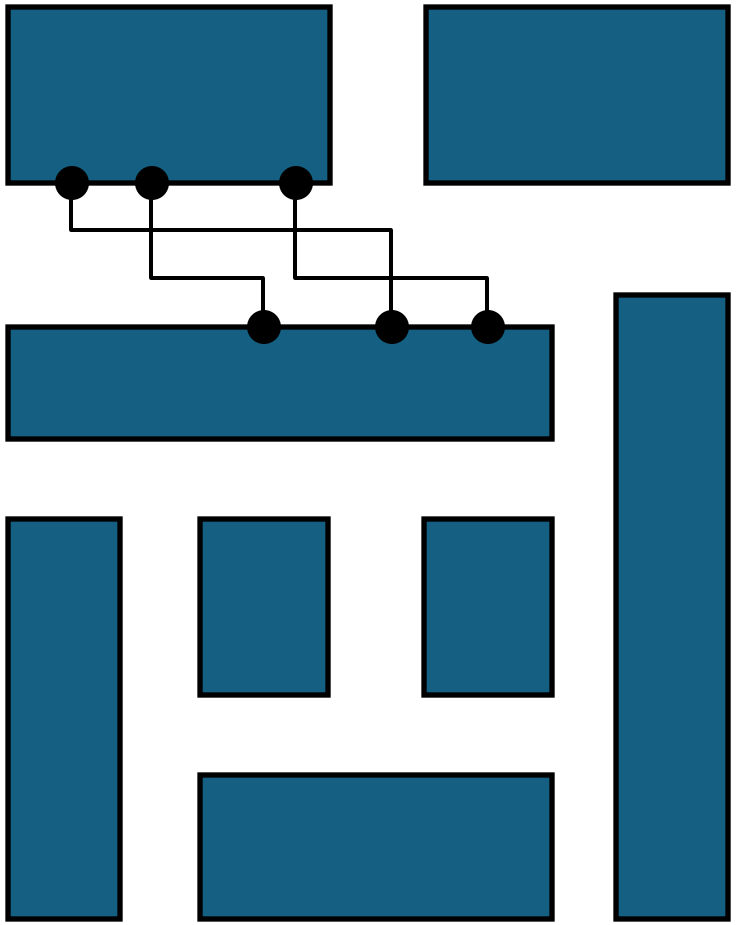


Steiner Tree Based Algorithms

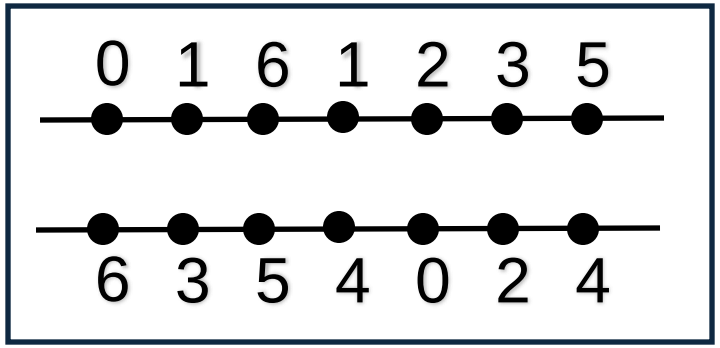
- For multi-terminal nets.
- Find Steiner tree instead of shortest path.
- Construct a Steiner tree from the minimum spanning trees (MST)



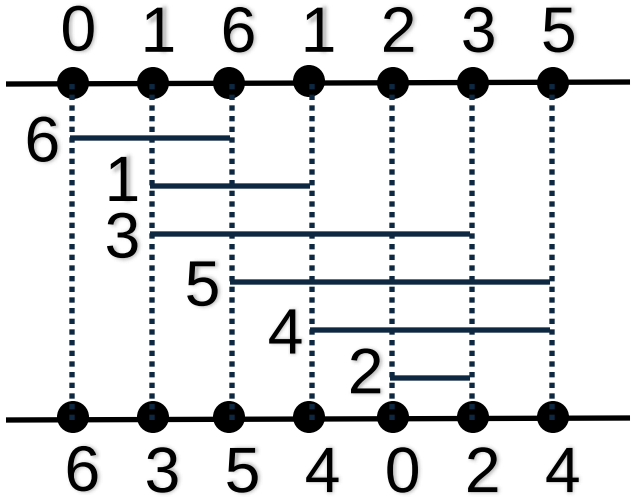
Channel Routing



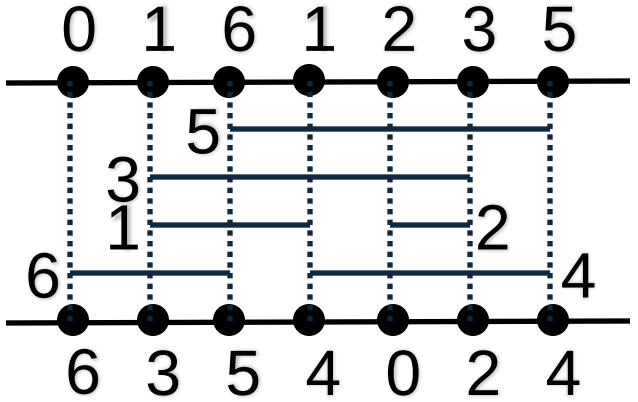
Left-Edge Algorithm



1. Sort by left end points.

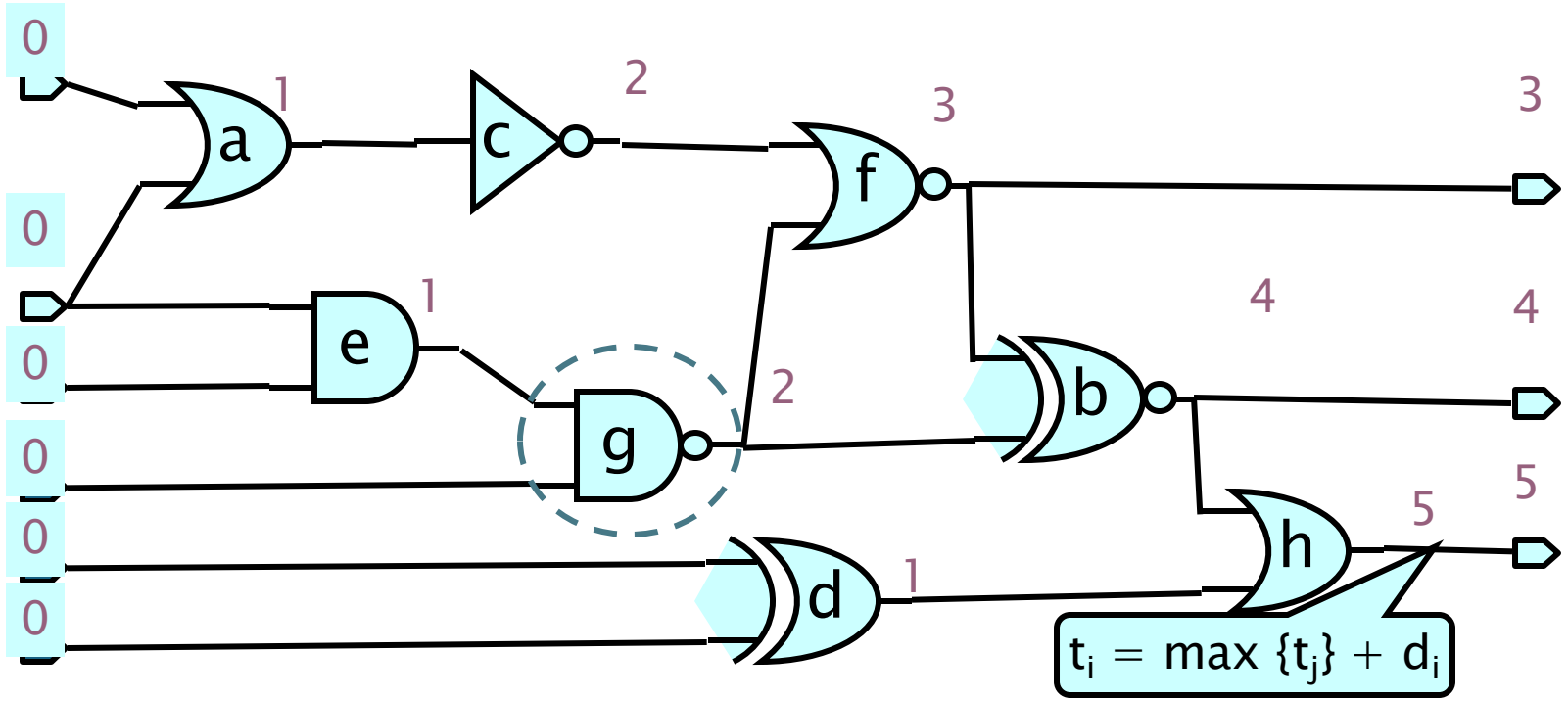


2. Place nets greedily.



STA Example: Arrival Times

- Assumptions:
  - –All inputs arrive at time 0, All gate delays = 1, All wire delays = 0
- Question: Arrival time of each gate? Circuit delay?



GCD example in RTL Verilog

A = 27 B = 15 ⇒ A - B = 12

12 15 → swap

15 12 → subtract A - B = 3

3 12 →

12 3 → 12 - 3 = 9

9 3 → 9 - 3 = 6

6 3 → 6 - 3 = 3

3 3 → 3 - 3 = 0

0 3 → swap

A = 27 B = 15 → 27 - 15

12 15 → swap

15 12 → 15 - 12

3 12 → swap

12 3 → ~~13~~ 12 - 3

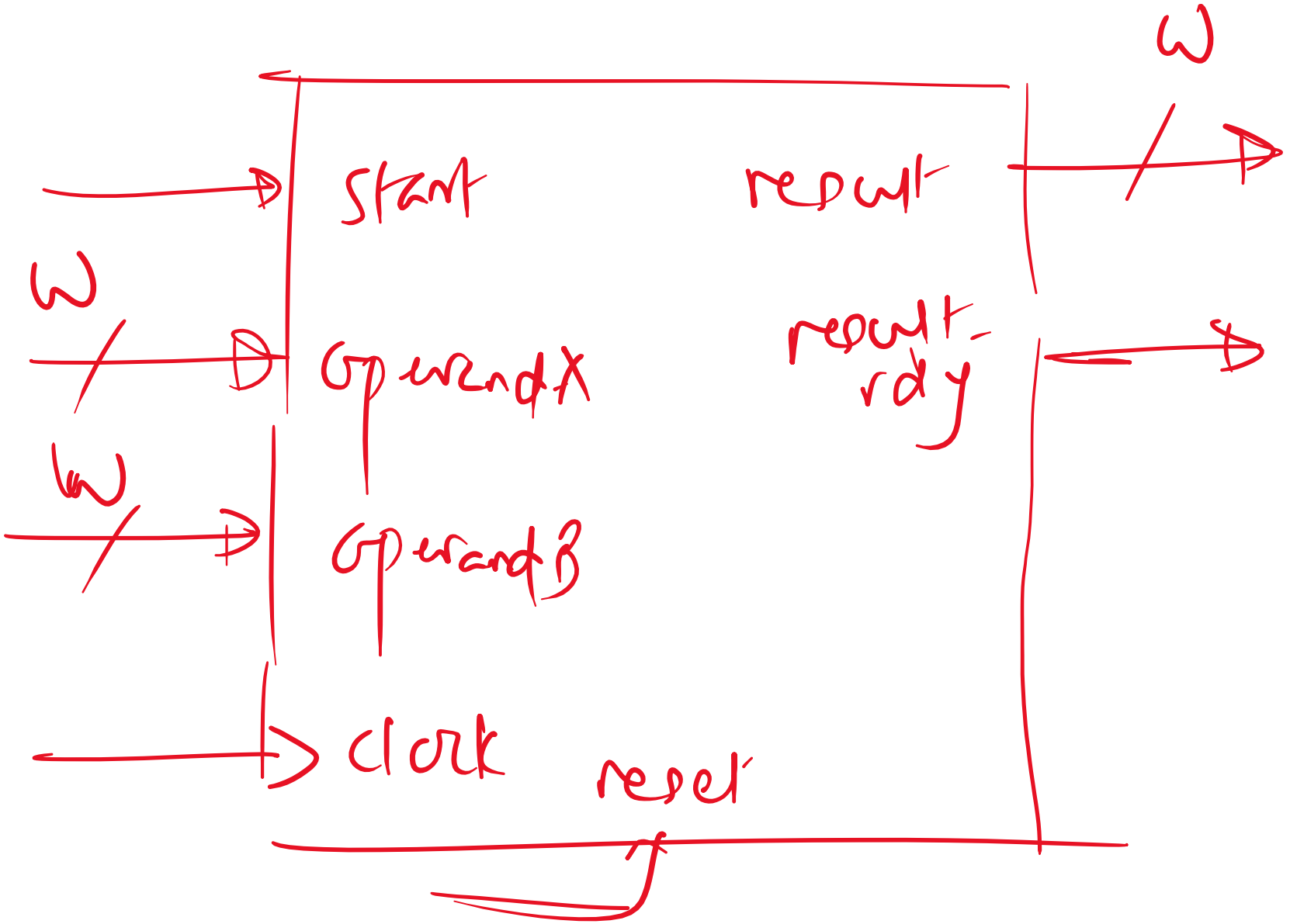
9 3 → 9 - 3

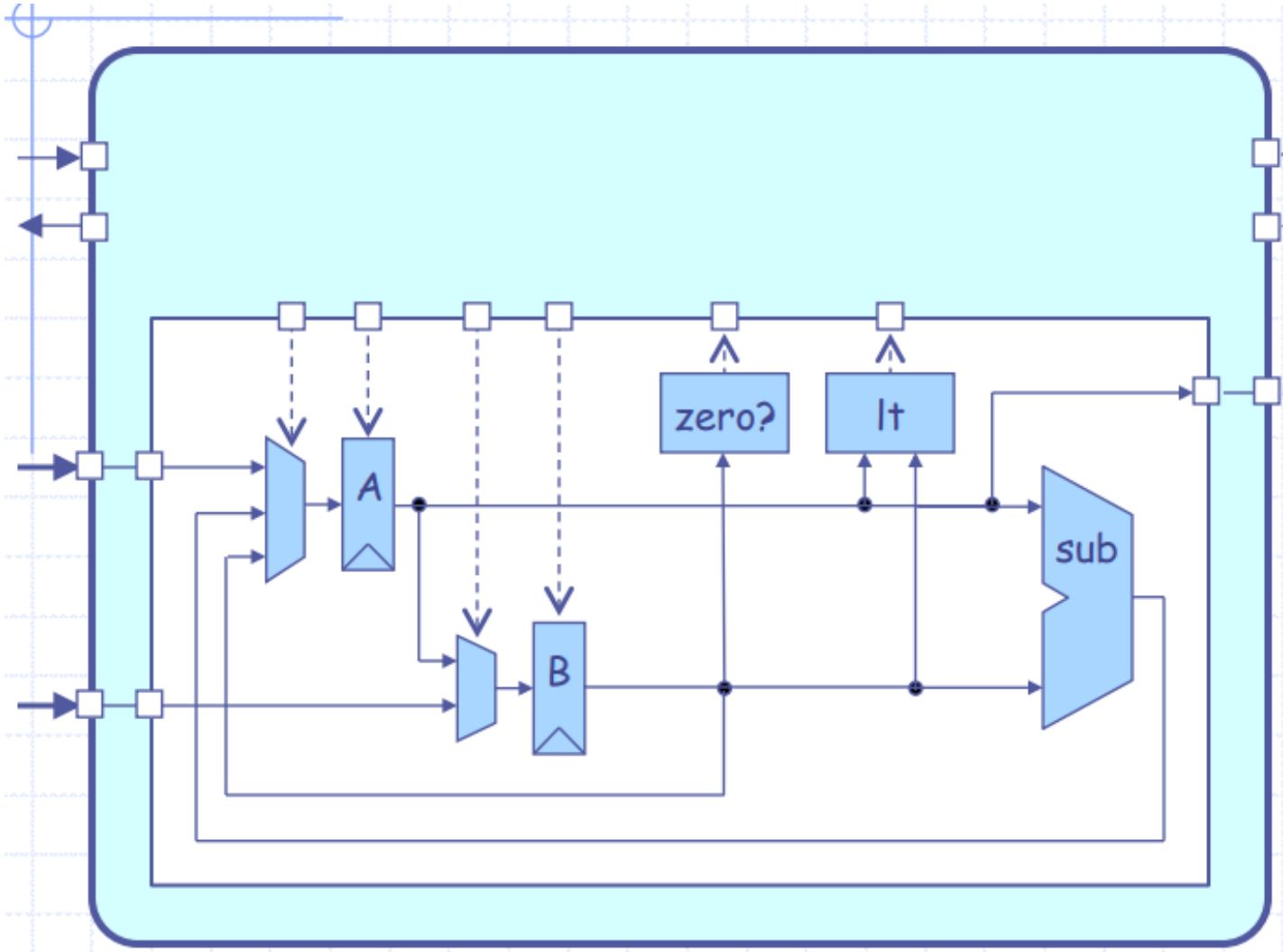
6 3 → 6 - 3

3 3 → 3 - 3

0 3 → swap

3 0 → B is zero


W
start
result
W
operandA
result_rdy
W
operandB
clock
reset




```
A = inA; B = inB;
while ( !done )
begin
 if ( A < B )
  swap = A;
  A = B;
  B = swap;
 else if (B != 0)
  A = A - B;
 else
  done = 1;
End
Y = A;
```

datapath

Courtesy of Arvind

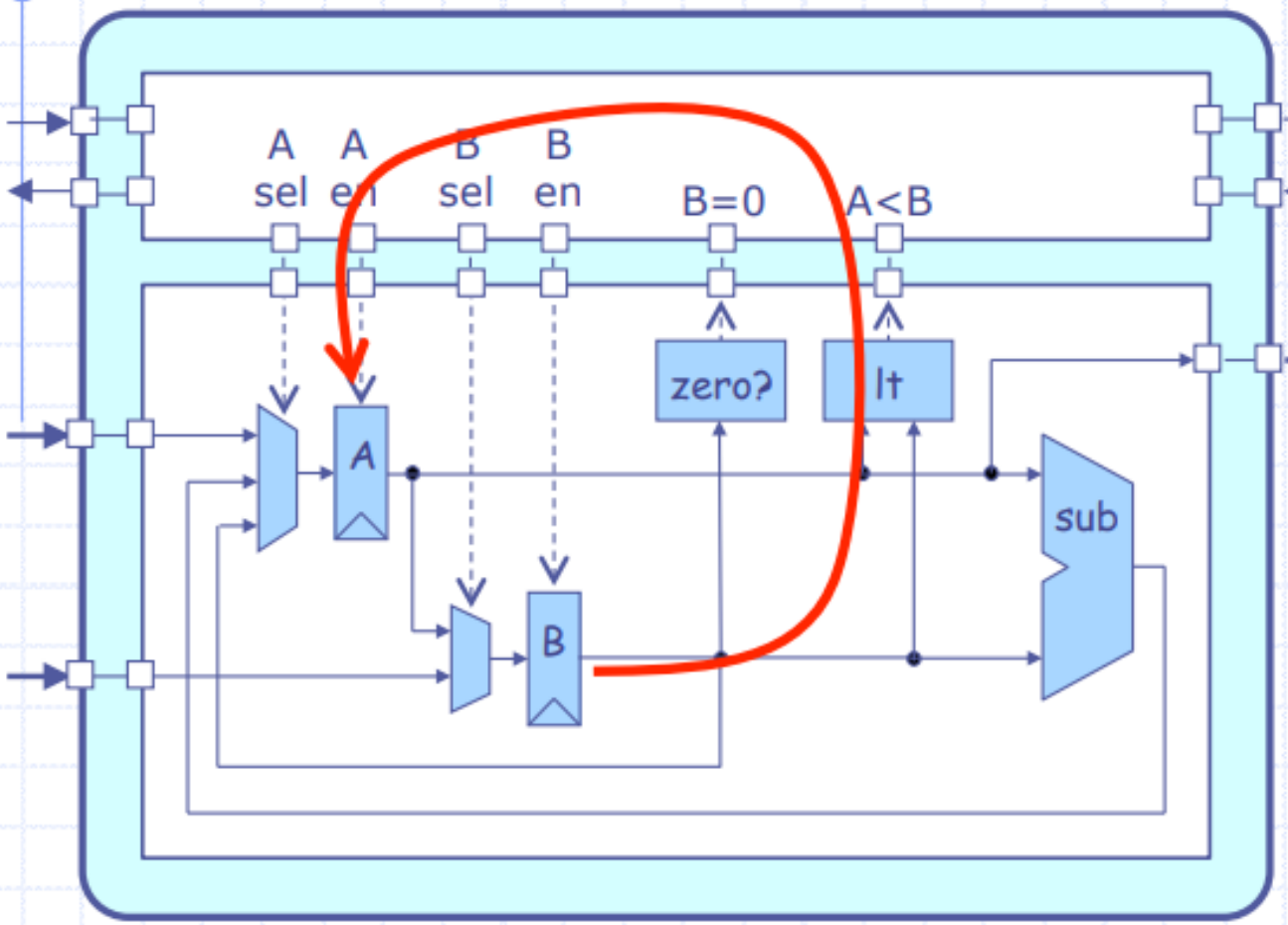
L03-17

https://cseweb.ucsd.edu/classes/sp10/cse141L/pdf/02/02-Verilog2.pdf

```
module GCDdatapath#( parameter W = 16 )
( input clk, input [W-1:0] operand_A, input [W-1:0] operand_B,
  output [W-1:0] result_data,
  input A_en, input B_en, input [1:0] A_sel, input B_sel,
  output B_zero, output A_lt_B
);
  reg [W-1:0] A; reg [W-1:0] B; wire [W-1:0] sub_out; wire [W-1:0] A_out;
  assign A_out = (A_sel==2'd0) ? operand_A : ( (A_sel==2'd1) ? B : sub_out ) ;
  always @( posedge clk ) begin if ( A_en ) A <= A_out ; end
  wire [W-1:0] B_out;
  assign B_out = (B_sel==0) ? operand_B : A ;
  always @( posedge clk ) begin if ( B_en ) B <= B_out ; end
  assign B_zero = (B==0); assign A_lt_B = (A < B);
  assign sub_out = A - B; assign result_data = A;
endmodule
```

- https://www.edaplayground.com/x/WQKx

Step 3: Add the control unit to sequence the datapath



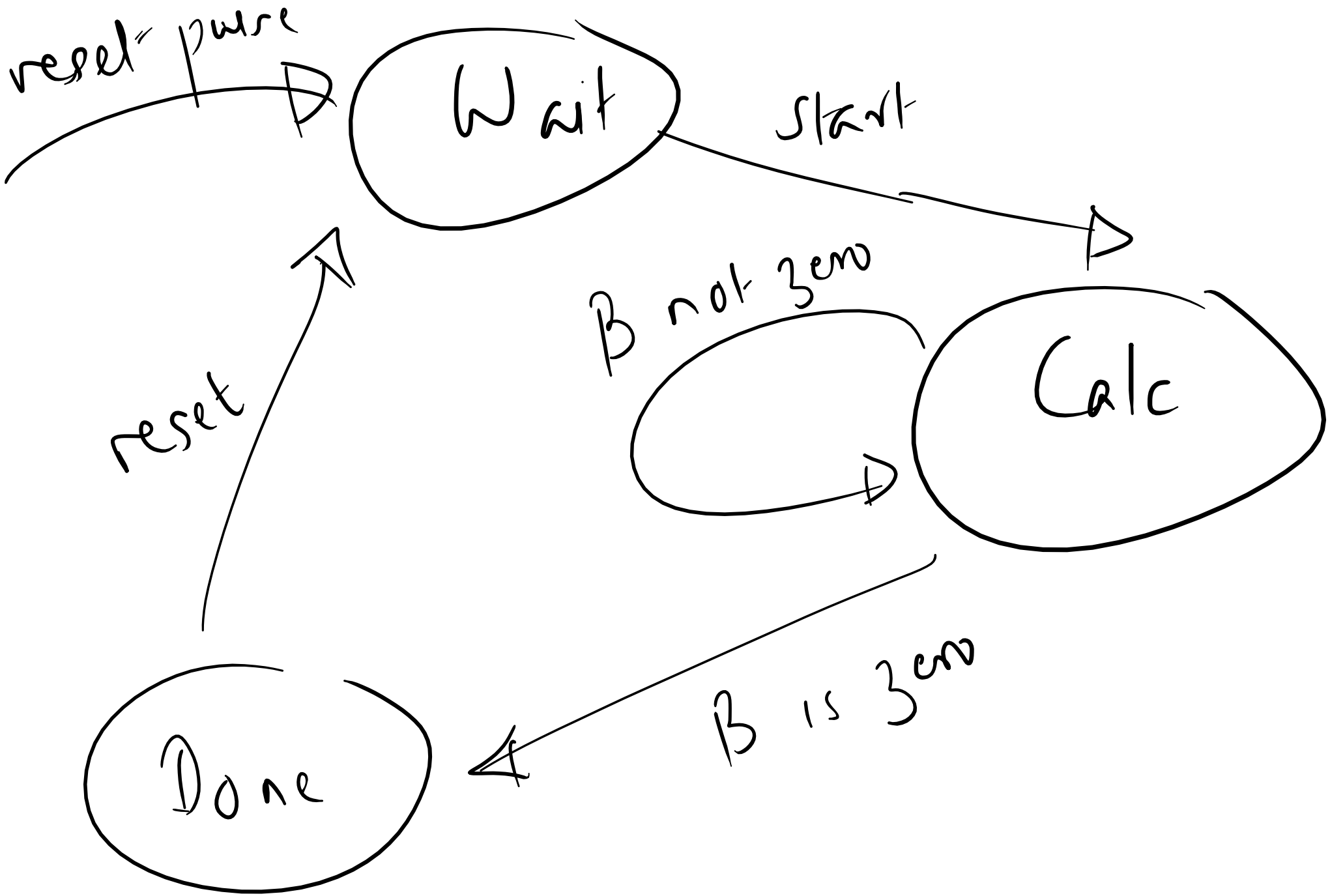
Control unit should be designed to be either busy or waiting for input or waiting for output to be picked up

```
A = inA; B = inB;
while ( !done )
begin
 if ( A < B )
   swap = A;
   A = B;
   B = swap;
 else if (B != 0)
   A = A - B;
 else
   done = 1;
End
Y = A;
```

Courtesy of Arvind

L03-18

https://cseweb.ucsd.edu/classes/sp10/cse141L/pdf/02/02-Verilog2.pdf

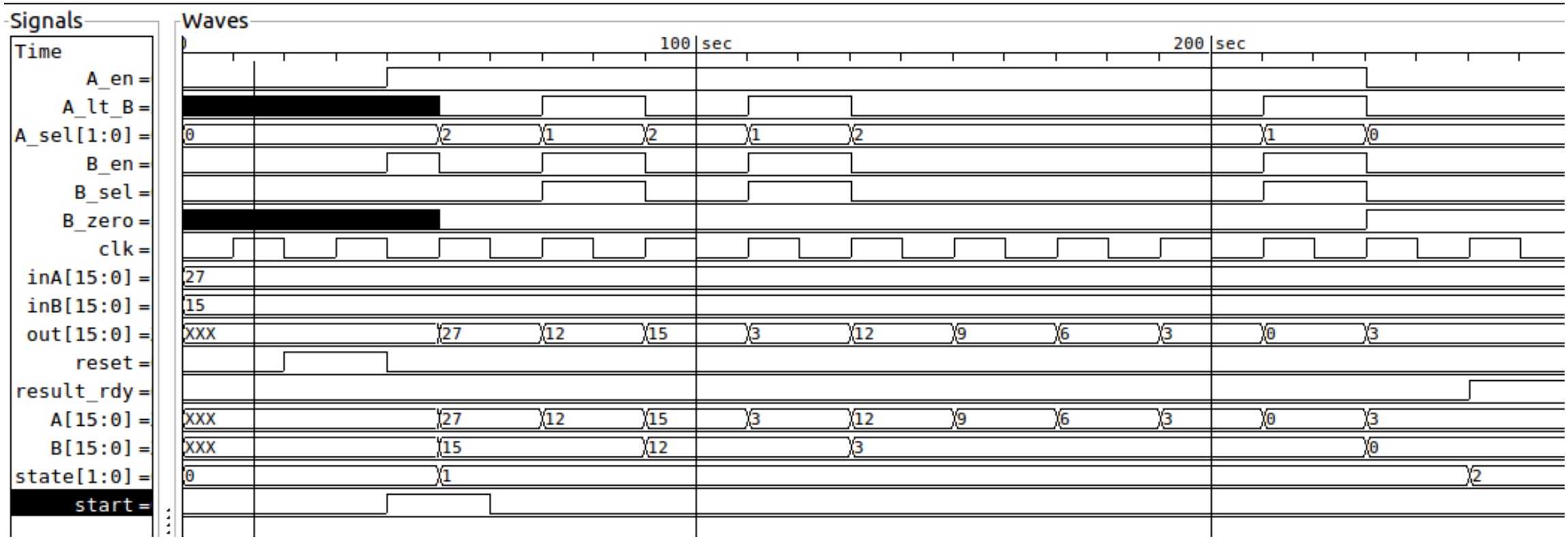

reset pwrc
Wait
start
B not zero
Calc
reset
Done
B is zero

```
module GCDcontroller#( parameter W = 16 )
( input clk, input reset, input start ,
  output reg A_en, output reg B_en, output reg [1:0] A_sel, output reg B_sel,
  input B_zero, input A_lt_B, output reg result_rdy
);
  localparam WAIT = 2'd0; localparam CALC = 2'd1; localparam DONE = 2'd2;
  reg [1:0] state_next; reg [1:0] state=WAIT;
  always @( posedge clk ) begin if ( reset ) state <= WAIT ; else state <= state_next ; end
  always @(*) begin
    A_sel = 0 ; A_en = 1'b0; B_sel = 0; B_en = 1'b0; result_rdy = 1'b0;
    case ( state )
      WAIT: begin A_sel = 0; B_sel = 0;
          if ( start ) begin A_en = 1'b1 ; B_en = 1'b1 ; end end
      CALC: if ( A_lt_B ) begin A_sel = 1; A_en = 1'b1; B_sel = 1; B_en = 1'b1; end
            else if ( !B_zero ) begin A_sel = 2; A_en = 1'b1; end
      DONE: result_rdy = 1'b1;
    endcase
  end
  always @(*) begin state_next = state;
    case ( state ) WAIT : if ( start ) state_next = CALC;
      CALC : if ( B_zero ) state_next = DONE; DONE : if ( reset ) state_next = WAIT;
    endcase
  end
endmodule
```

- https://www.edaplayground.com/x/WQKx

```
module GCDTestHarness;
  reg clk, reset, start ; reg [15:0] inA, inB; wire [15:0] out; wire result_rdy ;
  wire A_en, B_en, B_sel, B_zero, A_lt_B ; wire [1:0] A_sel ;
  always #10 clk = ~clk ;
  initial begin
    $dumpvars() ; clk=0; reset=0; @(negedge clk) reset=1 ; @(negedge clk) reset=0 ;
  end
  GCDdatapath#(16) gcd_dp_unit( clk, inA, inB, out, A_en, B_en, A_sel, B_sel, B_zero, A_lt_B ) ;
  GCDcontroller#(16) gcd_fsm_unit(clk, reset, start, A_en, B_en, A_sel, B_sel, B_zero, A_lt_B, result_rdy );
  initial begin
    start = 0 ; inA = 27; inB = 15;
    @(posedge clk) ; while ( reset ) @(posedge clk) ; while ( ~reset ) @(posedge clk) ;
    @(negedge clk) ; start = 1 ; @(negedge clk) ; start = 0 ; @(posedge clk) ;
    while ( ! result_rdy ) @(posedge clk) ;
    #1 ;
    if (out == 3) $display("Test gcd(27,15) succeeded, [%x==%x]", out, 3);
    else $display("Test gcd(27,15) failed, [%x != %x]", out, 3);
    $finish;
  end
endmodule
```

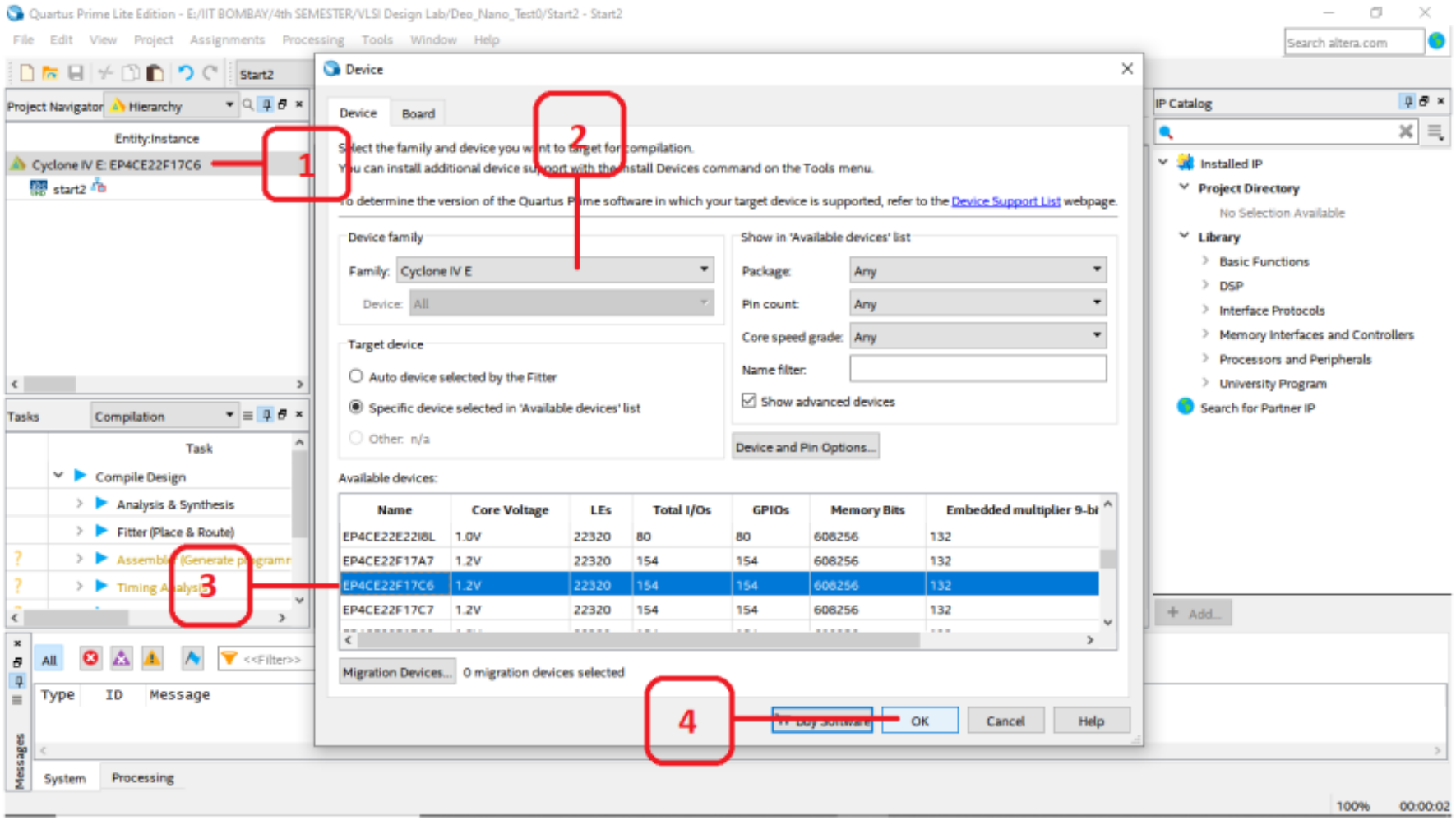
- https://www.edaplayground.com/x/WQKx



- https://www.edaplayground.com/x/WQKx

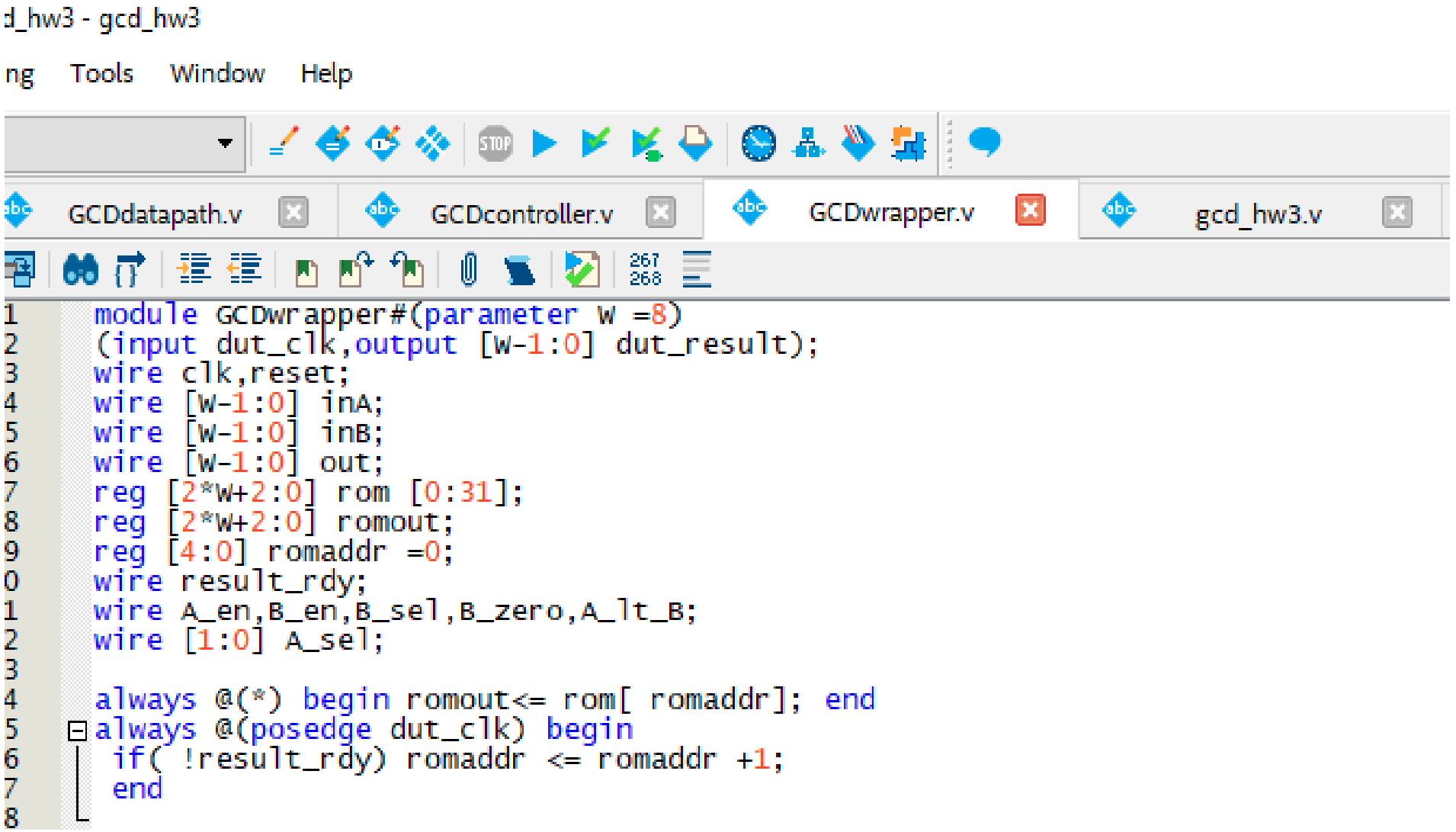
Hardwired-verif of gcd code using stimulus-waveforms stored in ROM.

1) Launch Quartus lite 18.1 > Create a project
2) Go to Assignments > Device... > Select Cyclone IV E in Device family, and EP4CE22F17C6 device and Click OK.



3) Add the following Files given in hw1: GDCdatapath.v and GCDcontroller.v

4) Create GCDwapper.v as shown below



```
module GCDwrapper#(parameter  W =8)
(input dut_clk,output [W-1:0] dut_result);

wire clk,reset; wire [W-1:0] inA;
wire [W-1:0] inB; wire [W-1:0] out;

reg [2*W+2:0] rom [0:31]; reg [2*W+2:0] romout;

reg [4:0] romaddr =0;

wire result_rdy;

wire A_en,B_en,B_sel,B_zero,A_lt_B;

wire [1:0] A_sel;

always @(*) begin romout<= rom[ romaddr]; end

always @(posedge dut_clk) begin
   if( !result_rdy) romaddr <= romaddr +1;
end

assign clk=romout[0];
assign reset=  romout[1];
assign start=romout[2];
assign inA=romout[W+2:3];
assign inB=romout[2*W+2:W+3];
assign dut_result=out;

GCDdatapath#(8) gcd_dp_unit( clk, inA, inB, out, A_en, B_en,
       A_sel, B_sel, B_zero, A_lt_B ) ;
GCDcontroller#(8) gcd_fsm_unit(clk, reset, start, A_en, B_en,
        A_sel, B_sel, B_zero, A_lt_B, result_rdy );

integer i;
initial  begin
  rom[0] ={8'b00011110,8'b00011001,1'b0,1'b0,1'b0};
  rom[1] ={8'b00011110,8'b00011001,1'b0,1'b0,1'b1};
  rom[2] ={8'b00011110,8'b00011001,1'b0,1'b1,1'b0};
  rom[3] ={8'b00011110,8'b00011001,1'b0,1'b1,1'b1};
  rom[4] ={8'b00011110,8'b00011001,1'b1,1'b0,1'b0};
  rom[5] ={8'b00011110,8'b00011001,1'b1,1'b0,1'b1};

  for(i=6;i<32;i=i+2)  begin
    rom[i] ={8'b00011110,8'b00011001,1'b0,1'b0,1'b0};
    rom[i+1] ={8'b00011110,8'b00011001,1'b0,1'b0,1'b1};
  end
end
endmodule
```

/gcd_hw3 - gcd_hw3

essing Tools Window Help

GCDdatapath.v | GCDcontroller.v | GCDwrapper.v | gcd_hw3.v | Compilation Report - gcd_hw3

```
16    if( !result_rdy) romaddr <= romaddr +1;
17    end
18
19     assign clk=romout[0];
20    assign reset= romout[1];
21    assign start=romout[2];
22    assign inA=romout[W+2:3];
23    assign inB=romout[2*W+2:W+3];
24    assign dut_result=out;
25
26    GCDdatapath#(8) gcd_dp_unit( clk, inA, inB, out, A_en, B_en, A_sel, B_sel, B_zero, A_lt_B ) ;
27    GCDcontroller#(8) gcd_fsm_unit(clk, reset, start, A_en, B_en, A_sel, B_sel, B_zero, A_lt_B, result_rdy );
28
29    integer i;
30    initial
31    begin
32    rom[0] ={8'b00011110,8'b00011001,1'b0,1'b0,1'b0};
33    rom[1] ={8'b00011110,8'b00011001,1'b0,1'b0,1'b1};
34    rom[2] ={8'b00011110,8'b00011001,1'b0,1'b1,1'b0};
35    rom[3] ={8'b00011110,8'b00011001,1'b0,1'b1,1'b1};
36    rom[4] ={8'b00011110,8'b00011001,1'b1,1'b0,1'b0};
37    rom[5] ={8'b00011110,8'b00011001,1'b1,1'b0,1'b1};
38
39    for(i=6;i<32;i=i+2);
40     begin
41      rom[i] ={8'b00011110,8'b00011001,1'b0,1'b0,1'b0};
42      rom[i+1] ={8'b00011110,8'b00011001,1'b0,1'b0,1'b1};
43     end
44    end
45    endmodule
```

Component of GDCdatapath and GCDcontroller are instantiated

"rom" has 32 entries, each entry is 18 bit wide.

column-0 contain the waveform of "clk"

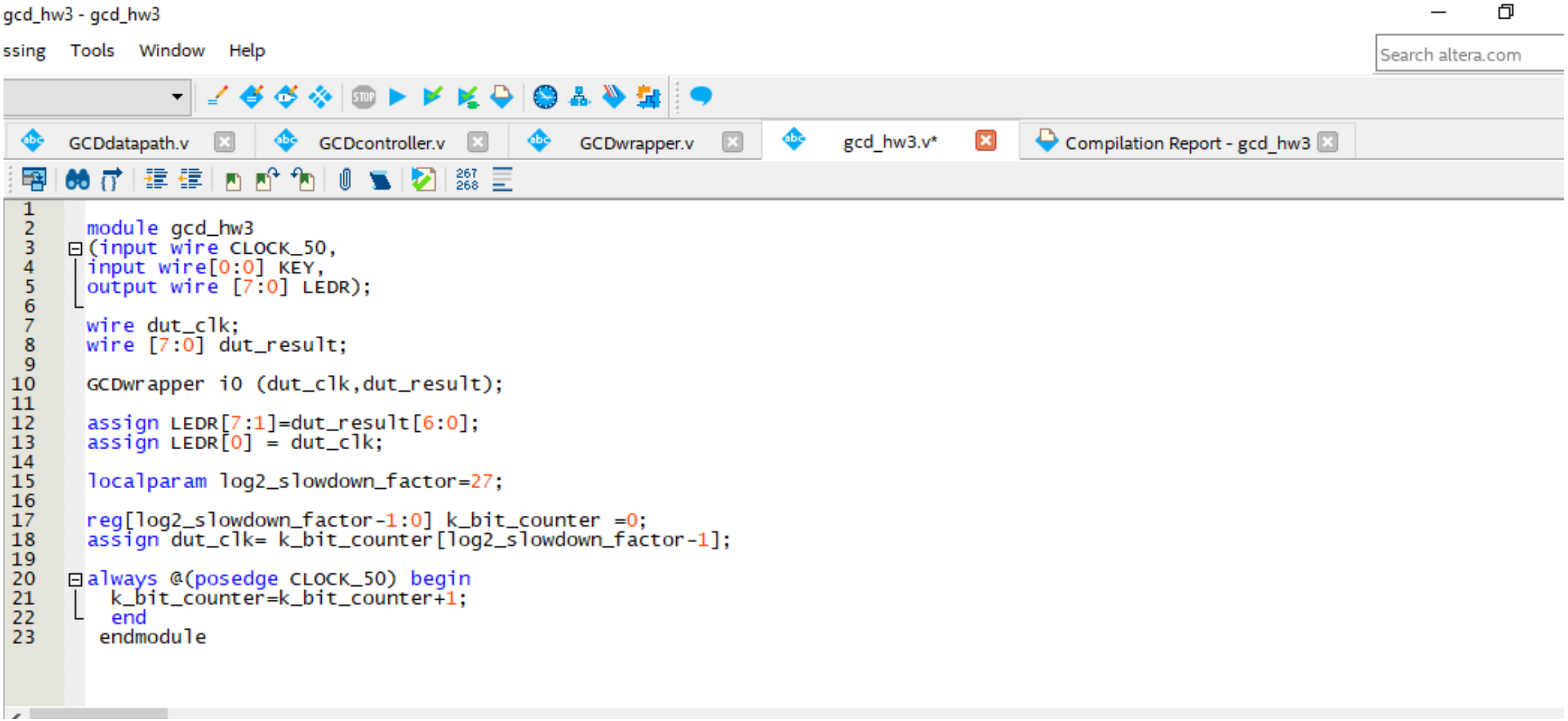
column-1 contain the waveform of "reset"

column-2 contain the waveform of "start"

Columns-indexed-10-downto-3 of "rom" contain the waveform of "inA"

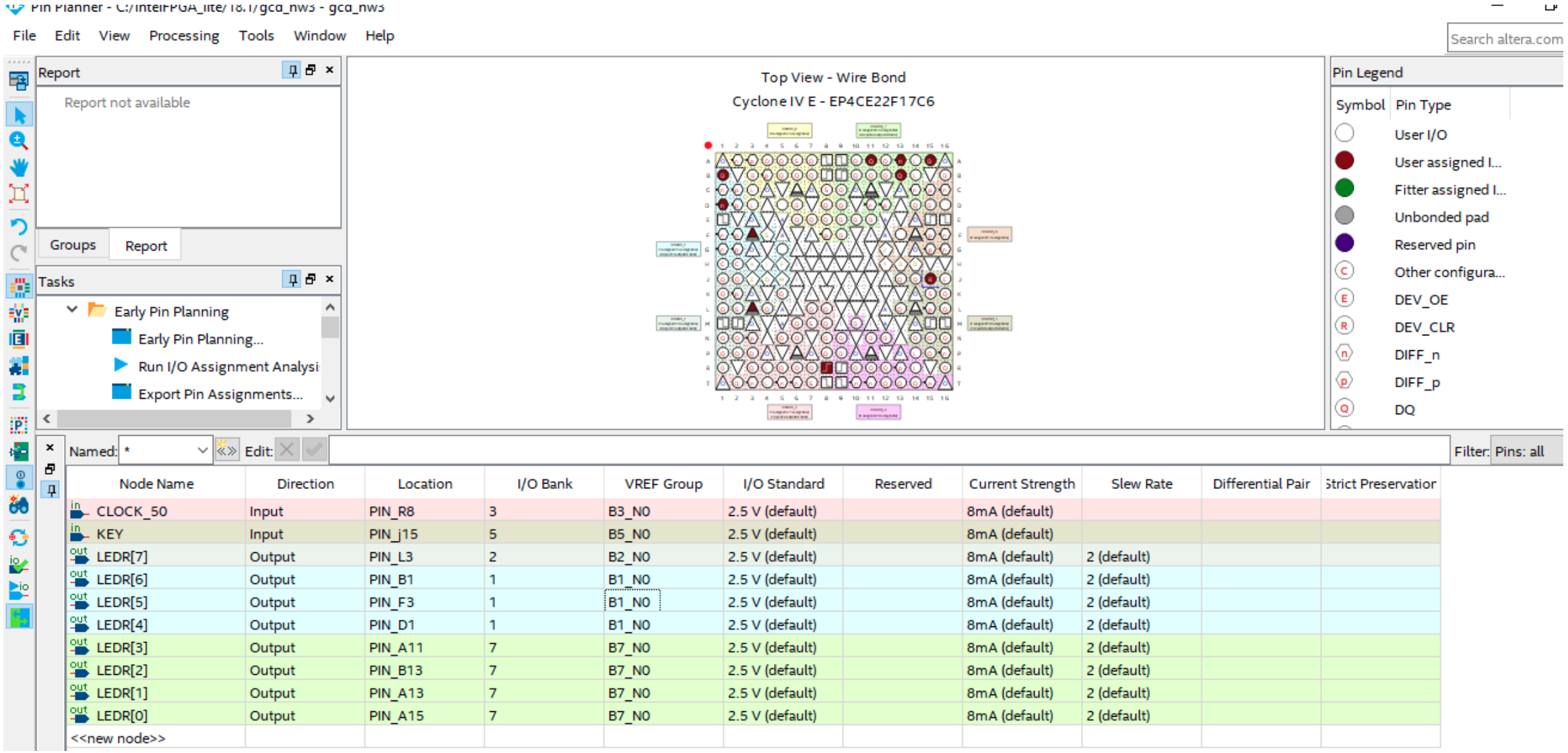
Columns-indexed-18-downto-11 of "rom" contain the waveform of "inB"

5) creating a file gcd_hw3.v and set it as top-level entity



- Original clock of 50Mhz is slowed down by "log2_slowdown_factor" and passed to GCDwrapper as dut_clk
- LEDR[0] is used to display the slow clock
- LEDR1 to LEDR7 represents the result
- KEY which is assigned to pushbutton is not used here but can be used as reset

6) Open Analysis & Synthesis > I/O Assignment Analysis > Pin Planner and select pins as follow



Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
CLOCK_50	Input	PIN_R8	3	B3_N0	2.5 V (default)		8mA (default)			
KEY	Input	PIN_j15	5	B5_N0	2.5 V (default)		8mA (default)			
LEDR[7]	Output	PIN_L3	2	B2_N0	2.5 V (default)		8mA (default)	2 (default)		
LEDR[6]	Output	PIN_B1	1	B1_N0	2.5 V (default)		8mA (default)	2 (default)		
LEDR[5]	Output	PIN_F3	1	B1_N0	2.5 V (default)		8mA (default)	2 (default)		
LEDR[4]	Output	PIN_D1	1	B1_N0	2.5 V (default)		8mA (default)	2 (default)		
LEDR[3]	Output	PIN_A11	7	B7_N0	2.5 V (default)		8mA (default)	2 (default)		
LEDR[2]	Output	PIN_B13	7	B7_N0	2.5 V (default)		8mA (default)	2 (default)		
LEDR[1]	Output	PIN_A13	7	B7_N0	2.5 V (default)		8mA (default)	2 (default)		
LEDR[0]	Output	PIN_A15	7	B7_N0	2.5 V (default)		8mA (default)	2 (default)		
<<new node>>										

Table 3-1 Pin Assignments for Push-buttons

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
KEY[0]	PIN_J15	Push-button[0]	3.3V
KEY[1]	PIN_E1	Push-button[1]	3.3V

Table 3-2 Pin Assignments for LEDs

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
LED[0]	PIN_A15	LED Green[0]	3.3V
LED[1]	PIN_A13	LED Green[1]	3.3V
LED[2]	PIN_B13	LED Green[2]	3.3V
LED[3]	PIN_A11	LED Green[3]	3.3V
LED[4]	PIN_D1	LED Green[4]	3.3V
LED[5]	PIN_F3	LED Green[5]	3.3V
LED[6]	PIN_B1	LED Green[6]	3.3V
LED[7]	PIN_L3	LED Green[7]	3.3V

Table 3-3 Pin Assignments for DIP Switches

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
DIP Switch[0]	PIN_M1	DIP Switch[0]	3.3V
DIP Switch[1]	PIN_T8	DIP Switch[1]	3.3V
DIP Switch[2]	PIN_B9	DIP Switch[2]	3.3V
DIP Switch[3]	PIN_M15	DIP Switch[3]	3.3V

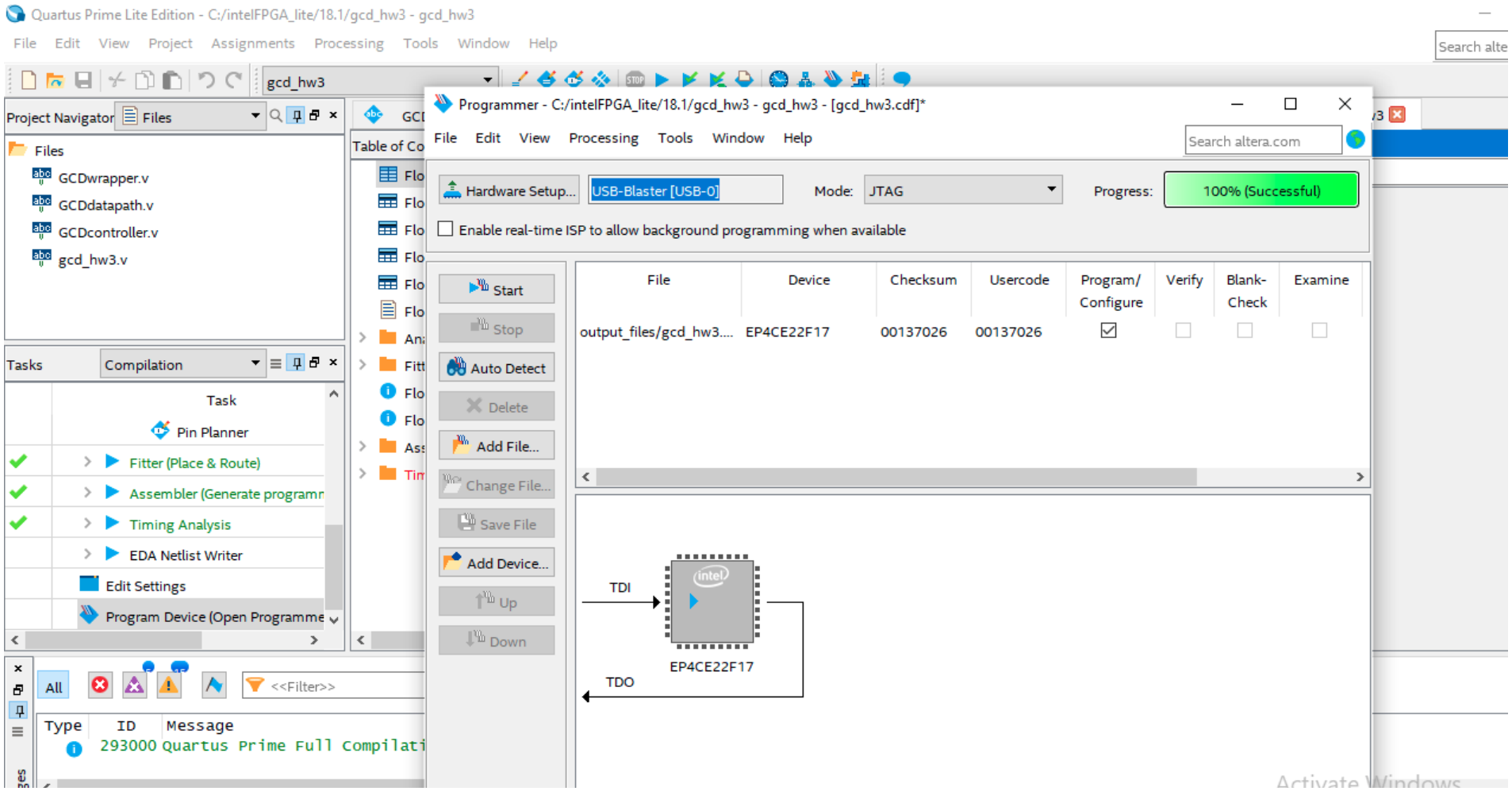
- Each pushbutton provides a high logic level when it is not pressed, and provides a low logic level when pressed. Since the pushbuttons are debounced, they are appropriate for using as clock or reset inputs.

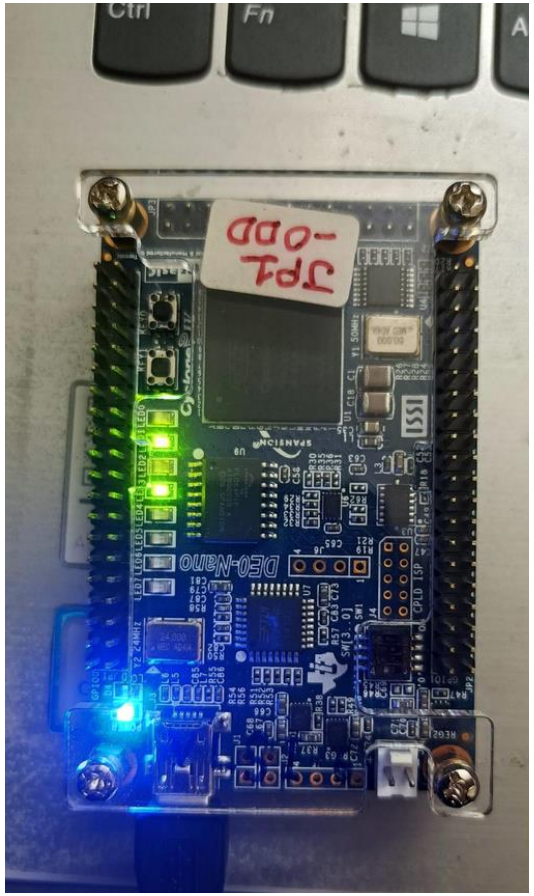
- CLOCK 50 = Pin R8. Pin R8 is the FPGA pin that is connected to a 50 MHz oscillator on the DE0-nano board.

Refernece manual : https://www.ti.com/lit/ug/tidu737/tidu737.pdf

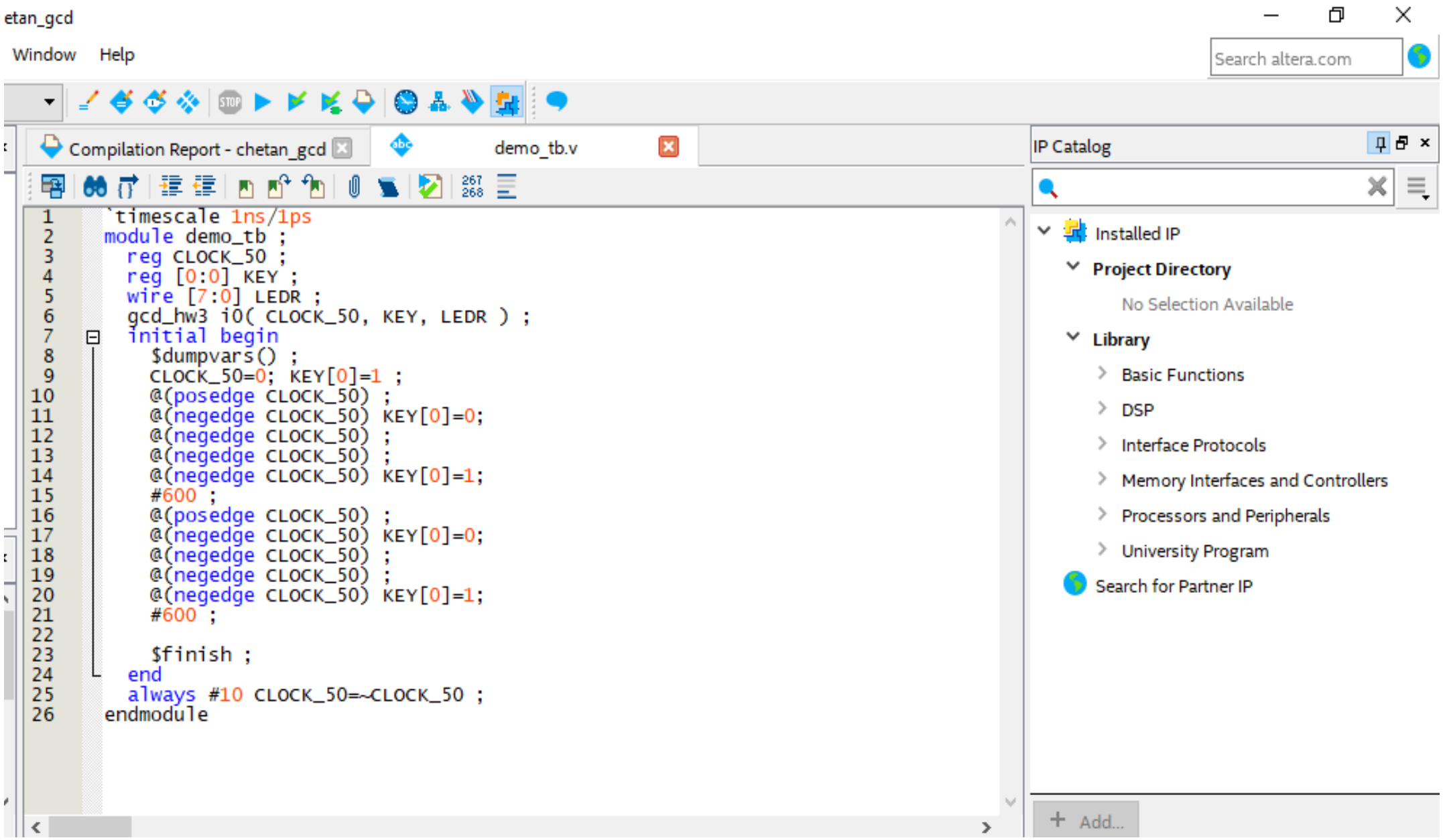
7) Compile the design

8) Go to Program Device (Open Programmer) > Hardware Setup and select USB-Blaster in Currently selected hardware.

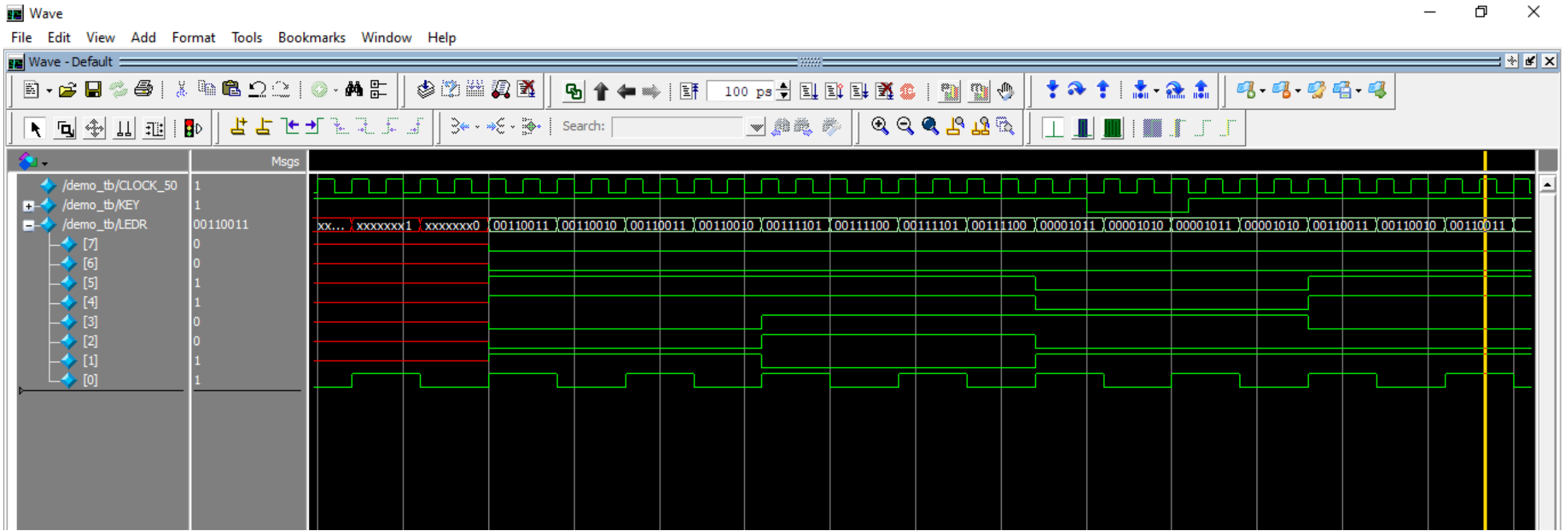




We can test the GCDwrapper file using the follwing testbench on modelsim for checking the correctness.



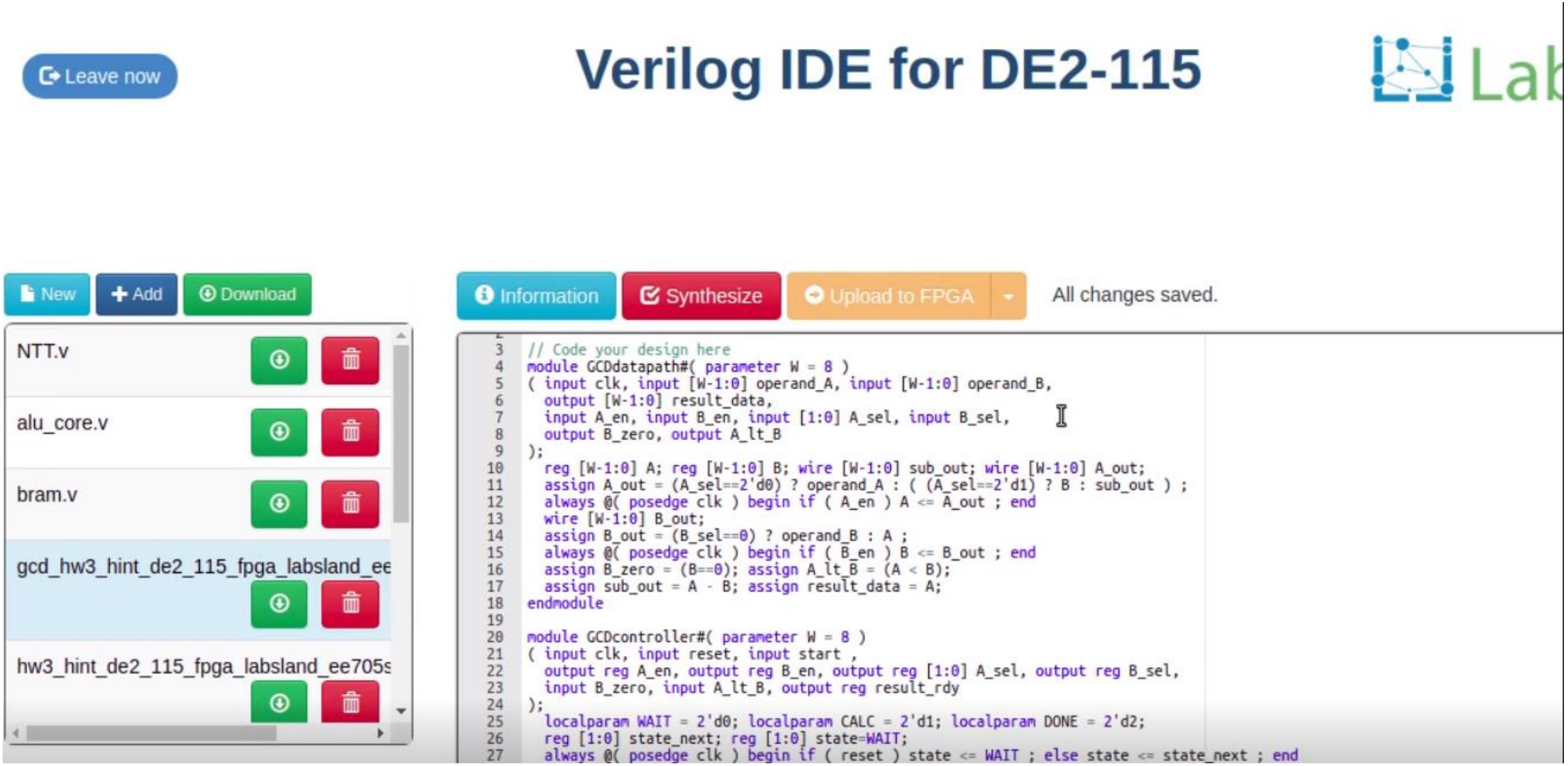

```
1   `timescale 1ns/1ps
2   module demo_tb ;
3     reg CLOCK_50 ;
4     reg [0:0] KEY ;
5     wire [7:0] LEDR ;
6     gcd_hw3 i0( CLOCK_50, KEY, LEDR ) ;
7     initial begin
8       $dumpvars() ;
9       CLOCK_50=0; KEY[0]=1 ;
10      @(posedge CLOCK_50) ;
11      @(negedge CLOCK_50) KEY[0]=0;
12      @(negedge CLOCK_50) ;
13      @(negedge CLOCK_50) ;
14      @(negedge CLOCK_50) KEY[0]=1;
15      #600 ;
16      @(posedge CLOCK_50) ;
17      @(negedge CLOCK_50) KEY[0]=0;
18      @(negedge CLOCK_50) ;
19      @(negedge CLOCK_50) ;
20      @(negedge CLOCK_50) KEY[0]=1;
21      #600 ;
22
23      $finish ;
24    end
25    always #10 CLOCK_50=~CLOCK_50 ;
26  endmodule
```



LED0 represents the slow clock

LED7 to LED1 represent the result which is 5 i.e GCD(30,25)

On Labsland Remote Labs (URL labsland.com)



Choose the type of instance

FPGA DE2 115

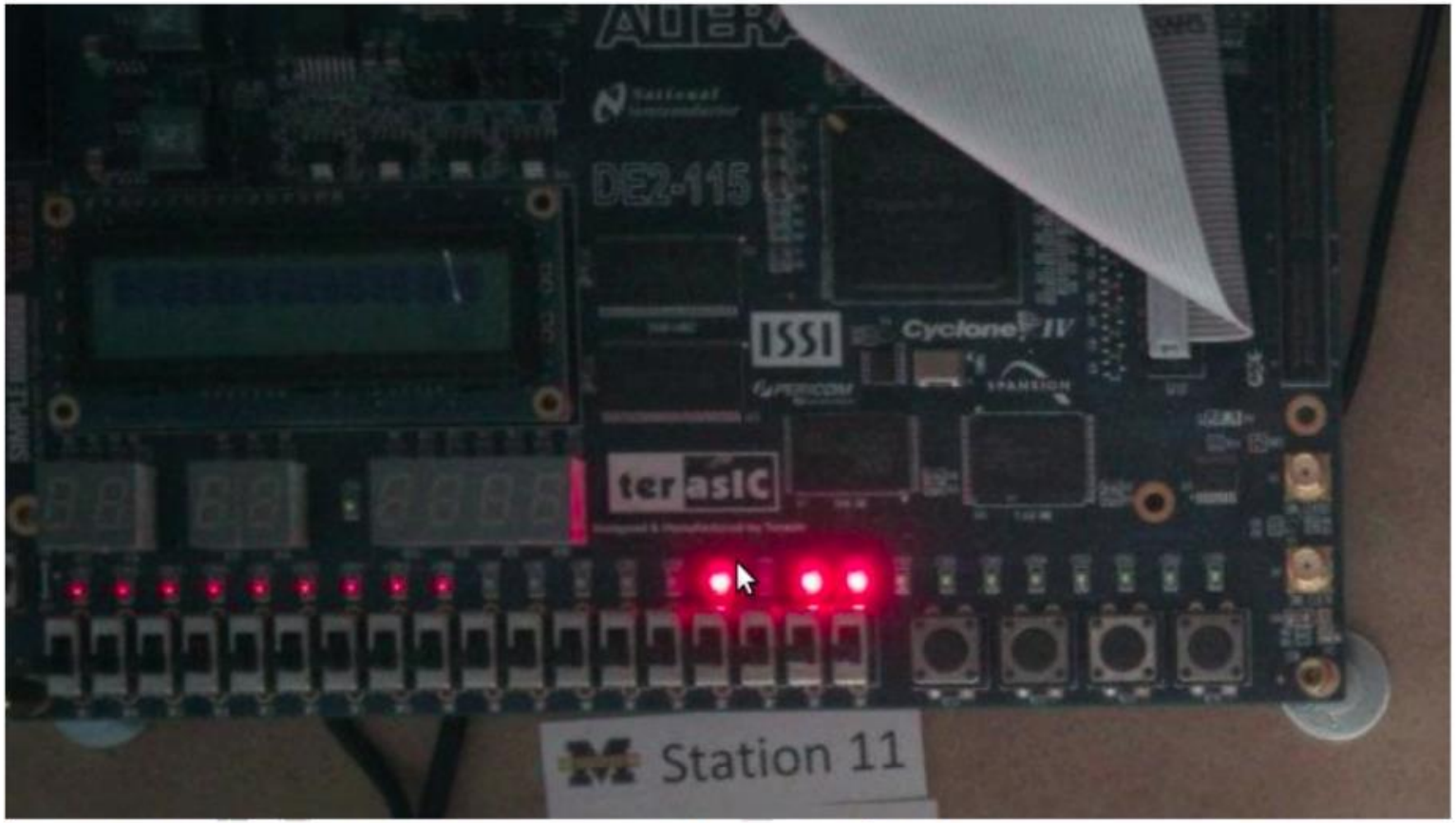
Upload now

Intel FPGA Laboratory

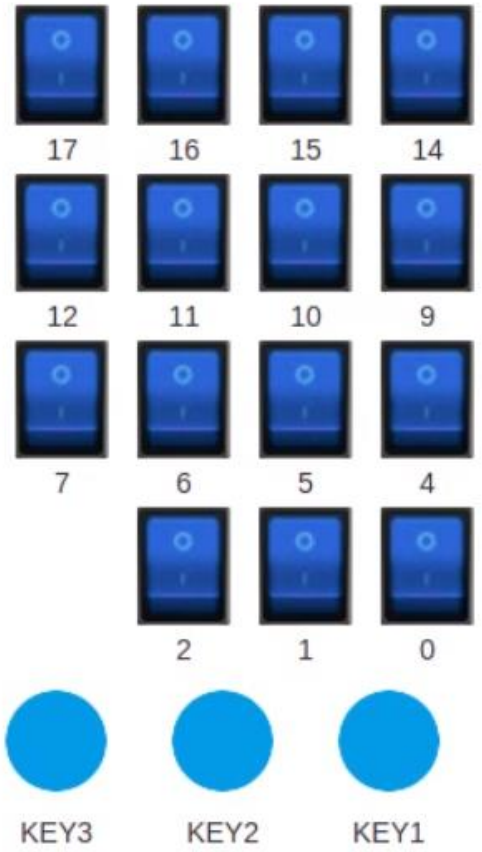
weblab.eecs.umich.edu/labs/intel-de2-115-s3i3/

01:49 Leave now

Intel FPGA Laboratory



You are using: umich-1-de2_115_s3i3. Experiencing any problem with this device? Let us know



Thank You