Network Architecture for FPGA-Based Event Filtering at $\overline{\text{P}}$ANDA/FAIR

Sören Fleischer

December 10, 2013

Inhaltsverzeichnis

A Detector Experiment

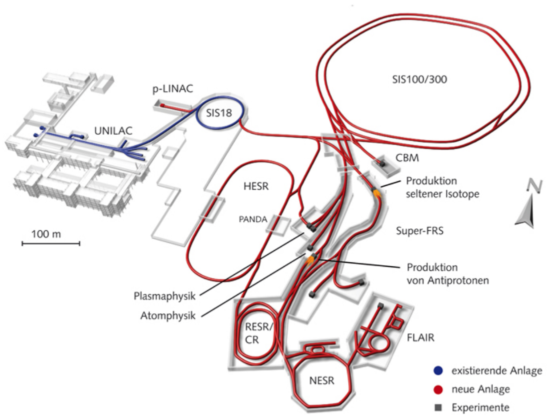


Figure: Topology of $\overline{\text{P}}$ANDA at FAIR

A Detector Experiment

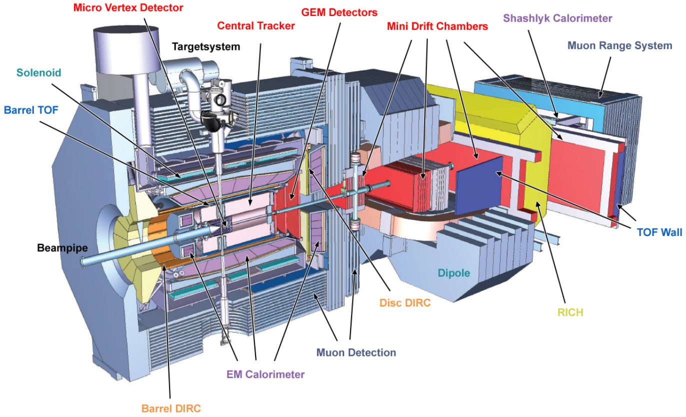


Figure: The $\overline{\text{P}}$ANDA Detector

A Detector Experiment

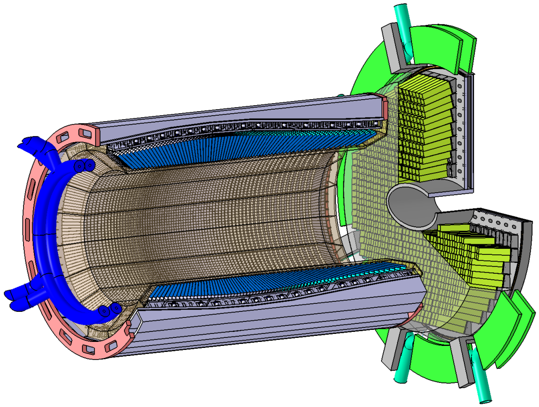


Figure: $\overline{\text{P}}$ANDA's Electromagnetic Calorimeter

A Detector Experiment

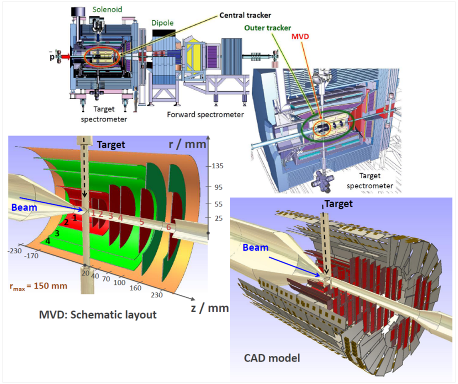


Figure: $\overline{\text{P}}$ANDA's Micro Vertex Detector

Online Data Reduction

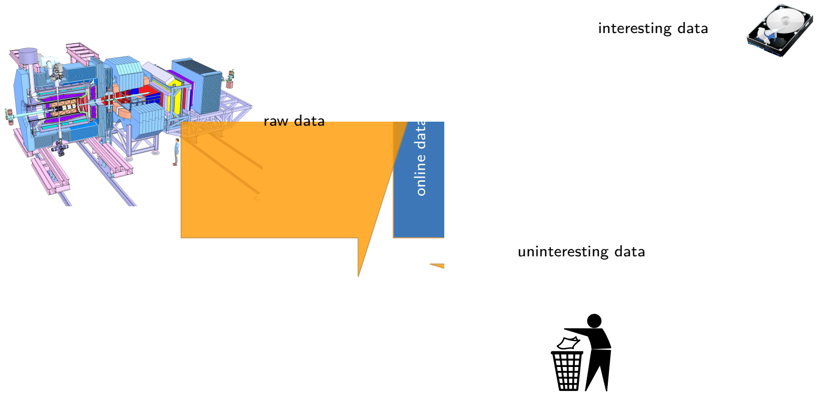


Figure: Online Data Reduction

Compute Node v3

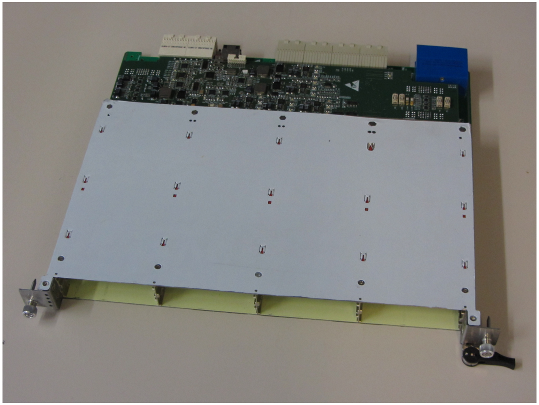


Figure: CN v3 Mother Board

Compute Node v3

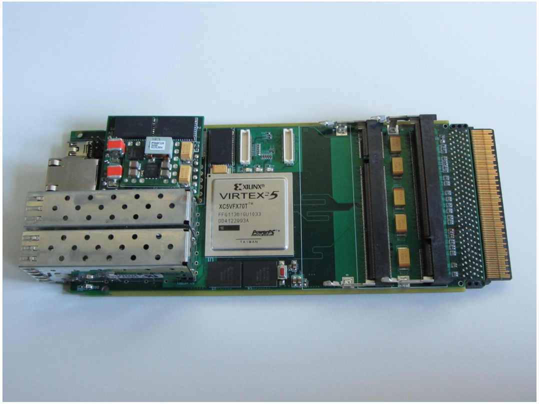


Figure: CN v3 Daughter Board, no RAM installed

Compute Node v3

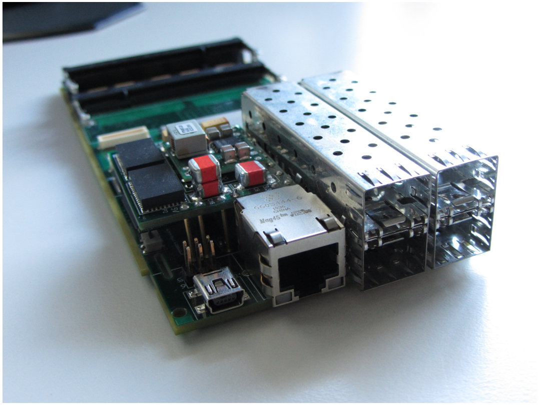


Figure: CN v3 Daughter Board, connectors

Compute Node Network Topology

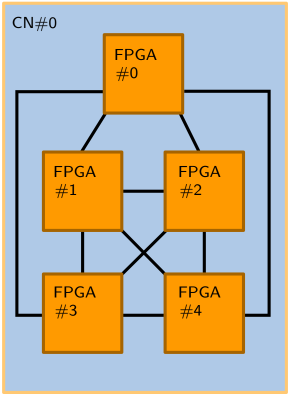


Figure: Fixed data paths on each CN

Compute Node Network Topology

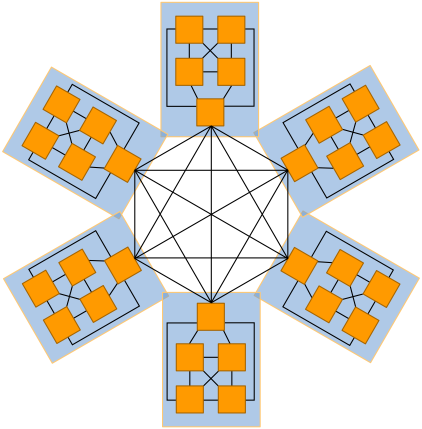


Figure: Possible topology in a shelf of 6 CNs

Example of a Tree network

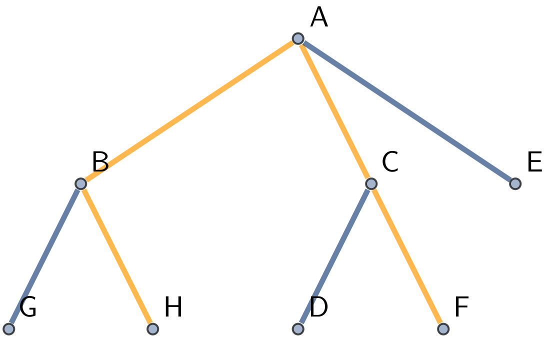
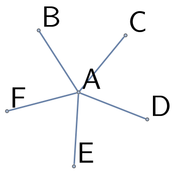
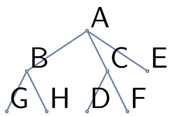


Figure: Example of a Tree network

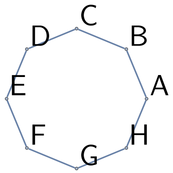
Network Topologies



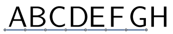
(a) Star



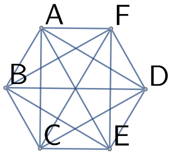
(b) Tree



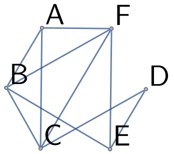
(c) Ring



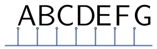
(d) Line



(e) Full mesh



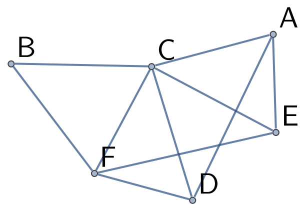
(f) Partial mesh



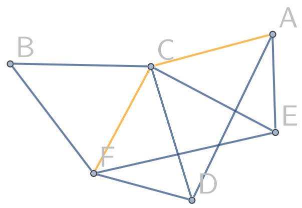
(g) Bus

Figure: Network topologies (example graphs)

Network Topologies

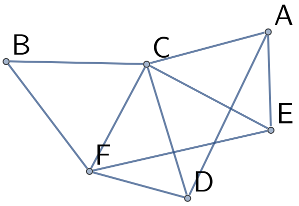


(a) A partial mesh network

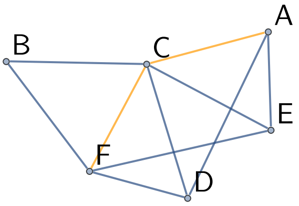


(b) Path 1

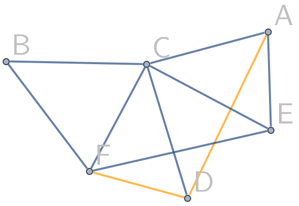
Network Topologies



(a) A partial mesh network

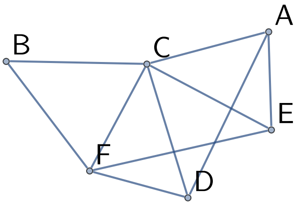


(b) Path 1

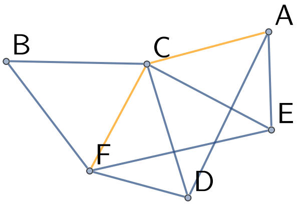


(c) Path 2

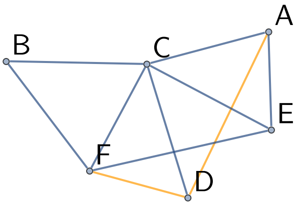
Network Topologies



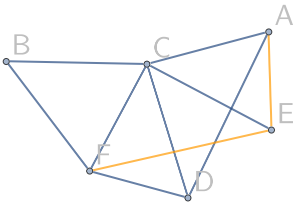
(a) A partial mesh network



(b) Path 1

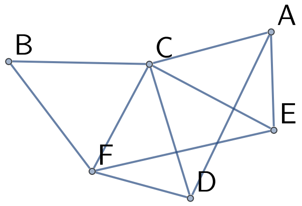


(c) Path 2

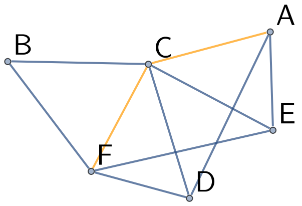


(d) Path 3

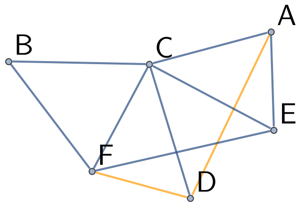
Network Topologies



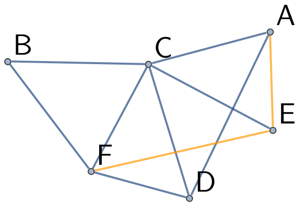
(a) A partial mesh network



(b) Path 1



(c) Path 2



(d) Path 3

Requirements

- Scalability
  Some 100 FPGAs
- Flexibility
  Changing topology without changing any bitstream

Requirements

- Scalability
  Some 100 FPGAs
- Flexibility
  Changing topology without changing any bitstream
- Being lightweight
  Don't consume too much precious Logic Cells on the FPGAs

Requirements

- Scalability
  Some 100 FPGAs
- Flexibility
  Changing topology without changing any bitstream
- Being lightweight
  Don't consume too much precious Logic Cells on the FPGAs
- Speed/Throughput
  The maximum speed of Xilinx' RocketIO of 6.5$\frac{\mathrm{G}b}{\text{sec}}$ should be achieved

Requirements

- Scalability
  Some 100 FPGAs
- Flexibility
  Changing topology without changing any bitstream
- Being lightweight
  Don't consume too much precious Logic Cells on the FPGAs
- Speed/Throughput
  The maximum speed of Xilinx' RocketIO of 6.5$\frac{\mathrm{G}b}{\text{sec}}$ should be achieved

Circuit Switching or Packet Switching

- Circuit Switching
  - Easier to implement

Circuit Switching or Packet Switching

- Circuit Switching
  - Easier to implement
  - Guaranteed speed for a given circuit

Circuit Switching or Packet Switching

- Circuit Switching
  - Easier to implement
  - Guaranteed speed for a given circuit
  - “Line Busy” error condition

Circuit Switching or Packet Switching

- Circuit Switching
  - Easier to implement
  - Guaranteed speed for a given circuit
  - "Line Busy" error condition
- Packet Switching

Circuit Switching or Packet Switching

- Circuit Switching
  - Easier to implement
  - Guaranteed speed for a given circuit
  - “Line Busy” error condition
- Packet Switching
  - Harder to implement

Circuit Switching or Packet Switching

- Circuit Switching
  - Easier to implement
  - Guaranteed speed for a given circuit
  - “Line Busy” error condition
- Packet Switching
  - Harder to implement
  - No guaranteed speed

Circuit Switching or Packet Switching

- Circuit Switching
  - Easier to implement
  - Guaranteed speed for a given circuit
  - “Line Busy” error condition
- Packet Switching
  - Harder to implement
  - No guaranteed speed
  - Might require more buffers (Block RAM) on the FPGAs

Circuit Switching or Packet Switching

- Circuit Switching
    - Easier to implement
    - Guaranteed speed for a given circuit
    - “Line Busy” error condition
- Packet Switching
    - Harder to implement
    - No guaranteed speed
    - Might require more buffers (Block RAM) on the FPGAs
    - Throttling mechanism

Circuit Switching or Packet Switching

- Circuit Switching
  - Easier to implement
  - Guaranteed speed for a given circuit
  - "Line Busy" error condition
- Packet Switching
  - Harder to implement
  - No guaranteed speed
  - Might require more buffers (Block RAM) on the FPGAs
  - Throttling mechanism
  - Packet packing/unpacking overhead

Circuit Switching or Packet Switching

- Circuit Switching
  - Easier to implement
  - Guaranteed speed for a given circuit
  - “Line Busy” error condition
- Packet Switching
  - Harder to implement
  - No guaranteed speed
  - Might require more buffers (Block RAM) on the FPGAs
  - Throttling mechanism
  - Packet packing/unpacking overhead

Circuit Switching or Packet Switching

Figure: Circuit switching in the olden days

Routing

- Where should the routing take place?
- Not on the precious FPGAs. Instead, on a PC which sends the routes to the FPGAs.

Routing

- Where should the routing take place?
- Not on the precious FPGAs. Instead, on a PC which sends the routes to the FPGAs.
- Dijkstra's Algorithm for route calculation

Routing

- Where should the routing take place?
- Not on the precious FPGAs. Instead, on a PC which sends the routes to the FPGAs.
- Dijkstra's Algorithm for route calculation

Complete Reduction System

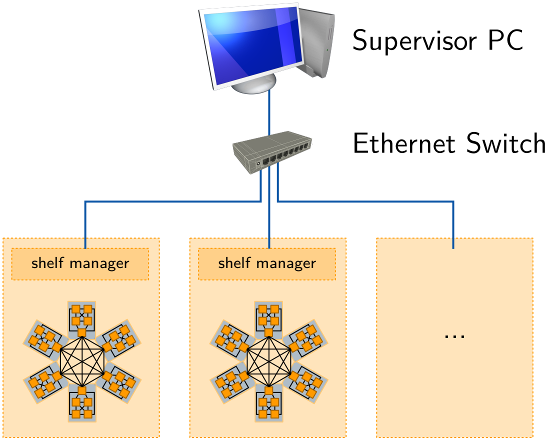


Figure: High level diagram of the reduction system

One Compute Node

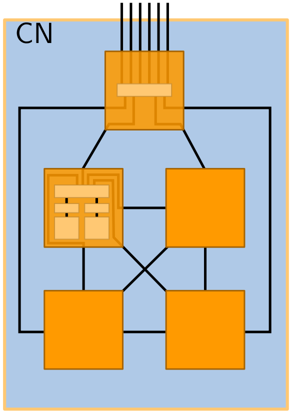


Figure: One Compute Node

One FPGA#0

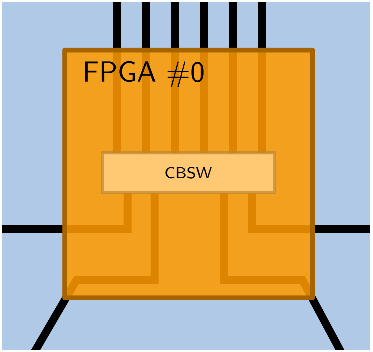


Figure: One Switching FPGA (#0)

Crossbar Switch

- Has n interfaces
- For each interface, it has

Crossbar Switch

- Has *n* interfaces
- For each interface, it has
    - A status (st_idle, st_a_connected, st_b_connected, st_lo_connected)

Crossbar Switch

- Has *n* interfaces
- For each interface, it has
  - A status (st_idle, st_a_connected, st_b_connected, st_lo_connected)
  - A target interface number

Crossbar Switch

- Has *n* interfaces
- For each interface, it has
  - A status (st_idle, st_a_connected, st_b_connected, st_lo_connected)
  - A target interface number
- For connected interfaces: Copies data from source interface to target interface

Crossbar Switch

- Has *n* interfaces
- For each interface, it has
  - A status (st_idle, st_a_connected, st_b_connected, st_lo_connected)
  - A target interface number
- For connected interfaces: Copies data from source interface to target interface
- For idle interfaces: Waits for requests to establish a connection or to identify oneself

Crossbar Switch

- Has *n* interfaces
- For each interface, it has
  - A status (st_idle, st_a_connected, st_b_connected, st_lo_connected)
  - A target interface number
- For connected interfaces: Copies data from source interface to target interface
- For idle interfaces: Waits for requests to establish a connection or to identify oneself

Example for Crossbar Switch state signals

Interface number	Interface status	Target interface
1	st_idle	0
2	st_a_connected	4
3	st_lo_connected	3
4	st_b_connected	2

Table: Example of the content of target and interface status information within a 4-interface crossbar switch

State signals and 2 state machines

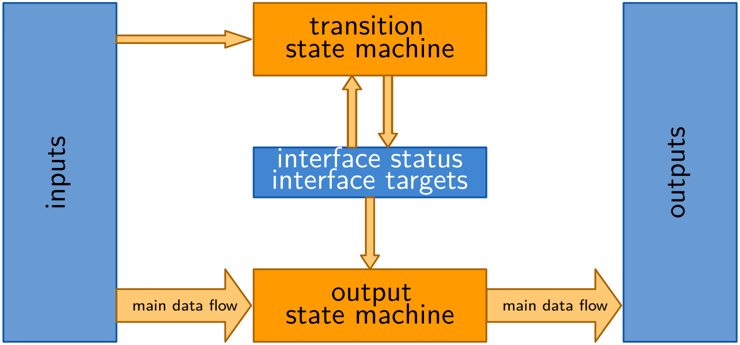
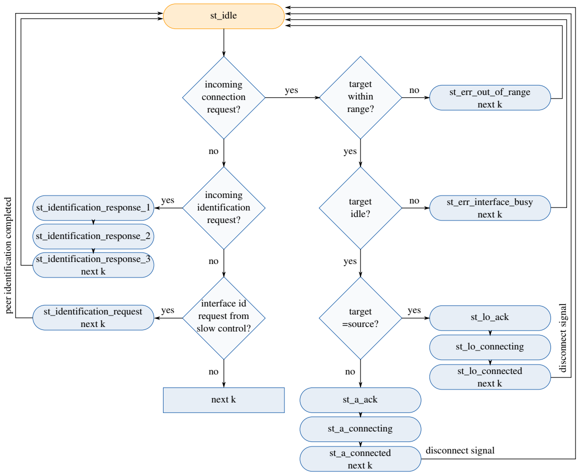


Figure: Schematic data flow inside a crossbar switch

State Diagram of the Crossbar Switch


st_idle
incoming connection request?
yes
no
target within range?
no
st_err_out_of_range next k
yes
incoming identification request?
yes
st_identification_response_1
st_identification_response_2
st_identification_response_3 next k
peer identification completed
no
interface id request from slow control?
yes
st_identification_request next k
no
next k
target idle?
no
st_err_interface_busy next k
yes
target =source?
yes
st_lo_ack
st_lo_connecting
st_lo_connected next k
disconnect signal
no
st_a_ack
st_a_connecting
st_a_connected next k
disconnect signal

One FPGA#1-#4

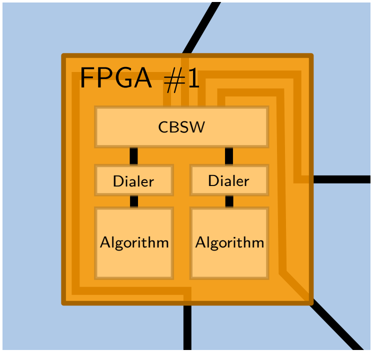


Figure: One Algorithm FPGA (#1-#4)

Data flow between algorithms

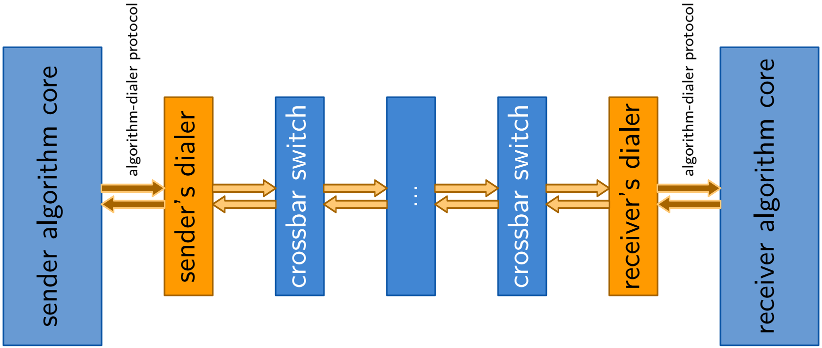


Figure: Schematic data flow from sender algorithm to receiver algorithm

Dialer

- Establishes a connection to a receiver algorithm, as requested by the associated sender algorithm.
- Has a “phone book” of receiver algorithms who the associated sender algorithm wants to send data to. Consisting of:

Dialer

- Establishes a connection to a receiver algorithm, as requested by the associated sender algorithm.
- Has a "phone book" of receiver algorithms who the associated sender algorithm wants to send data to. Consisting of:
  - ID/Class of the receiver algorithm

Dialer

- Establishes a connection to a receiver algorithm, as requested by the associated sender algorithm.
- Has a “phone book” of receiver algorithms who the associated sender algorithm wants to send data to. Consisting of:
  - ID/Class of the receiver algorithm
  - The route to the receiver algorithm

Dialer

- Establishes a connection to a receiver algorithm, as requested by the associated sender algorithm.
- Has a “phone book” of receiver algorithms who the associated sender algorithm wants to send data to. Consisting of:
    - ID/Class of the receiver algorithm
    - The route to the receiver algorithm
    - Error counters for problems which can arise while establishing that connection:

Dialer

- Establishes a connection to a receiver algorithm, as requested by the associated sender algorithm.
- Has a "phone book" of receiver algorithms who the associated sender algorithm wants to send data to. Consisting of:
  - ID/Class of the receiver algorithm
  - The route to the receiver algorithm
  - Error counters for problems which can arise while establishing that connection:
    - Line busy

Dialer

- Establishes a connection to a receiver algorithm, as requested by the associated sender algorithm.
- Has a "phone book" of receiver algorithms who the associated sender algorithm wants to send data to. Consisting of:
  - ID/Class of the receiver algorithm
  - The route to the receiver algorithm
  - Error counters for problems which can arise while establishing that connection:
    - Line busy
    - Out of range

Dialer

- Establishes a connection to a receiver algorithm, as requested by the associated sender algorithm.
- Has a “phone book” of receiver algorithms who the associated sender algorithm wants to send data to. Consisting of:
  - ID/Class of the receiver algorithm
  - The route to the receiver algorithm
  - Error counters for problems which can arise while establishing that connection:
    - Line busy
    - Out of range
    - Timeout

Dialer

- Establishes a connection to a receiver algorithm, as requested by the associated sender algorithm.
- Has a “phone book” of receiver algorithms who the associated sender algorithm wants to send data to. Consisting of:
    - ID/Class of the receiver algorithm
    - The route to the receiver algorithm
    - Error counters for problems which can arise while establishing that connection:
        - Line busy
        - Out of range
        - Timeout

Example for a Dialer Memory

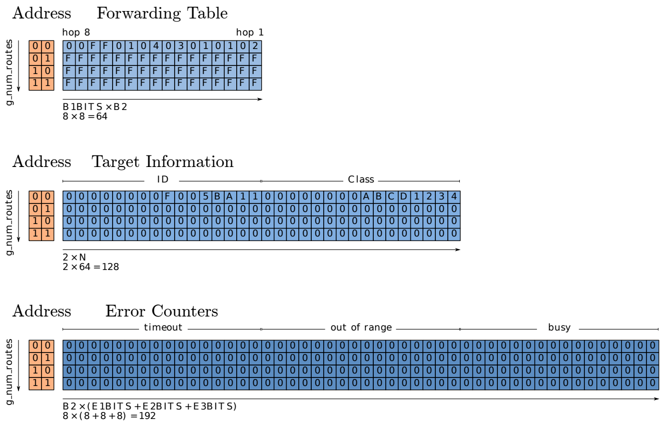
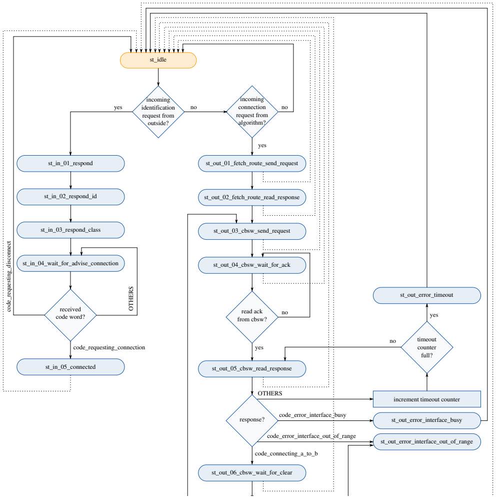


Figure: Example of a dialer memory with g_num_routes=4

Dialer State Diagram


st_idle
incoming identification request from outside?
yes
no
incoming connection request from algorithm?
no
yes
st_in_01_respond
st_in_02_respond_id
st_in_03_respond_class
st_in_04_wait_for_advise_connection
received code word?
OTHERS
code_requesting_disconnect
code_requesting_connection
st_in_05_connected
st_out_01_fetch_route_send_request
st_out_02_fetch_route_read_response
st_out_03_cbsw_send_request
st_out_04_cbsw_wait_for_ack
read ack from cbsw?
no
yes
st_out_05_cbsw_read_response
OTHERS
response?
code_error_interface_busy
code_error_interface_out_of_range
code_connecting_a_to_b
st_out_06_cbsw_wait_for_clear
st_out_error_timeout
yes
timeout counter full?
no
increment timeout counter
st_out_error_interface_busy
st_out_error_interface_out_of_range

Dialer State Diagram

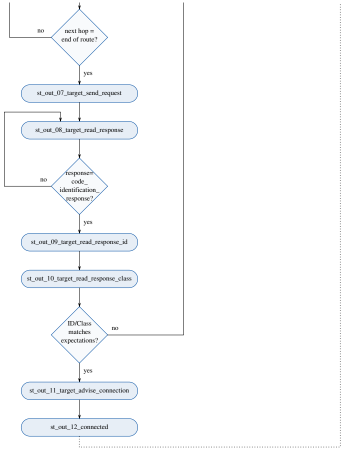
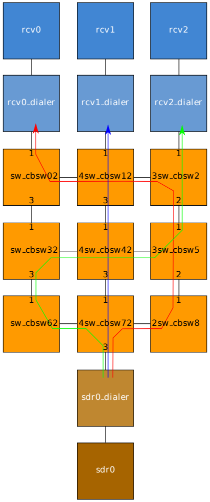
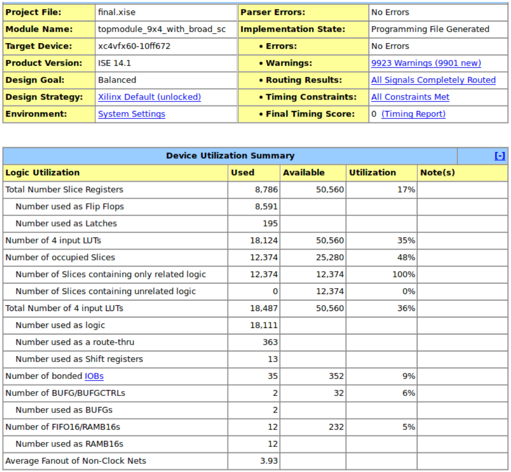

no
next hop = end of route?
yes
st_out_07_target_send_request
st_out_08_target_read_response
no
response= code_ identification_ response?
yes
st_out_09_target_read_response_id
st_out_10_target_read_response_class
ID/Class matches expectations?
no
yes
st_out_11_target_advise_connection
st_out_12_connected

Figure: Dialer Flow Chart (Part 2)

Test project with 9 CBSWs



Test project with 9 CBSWs

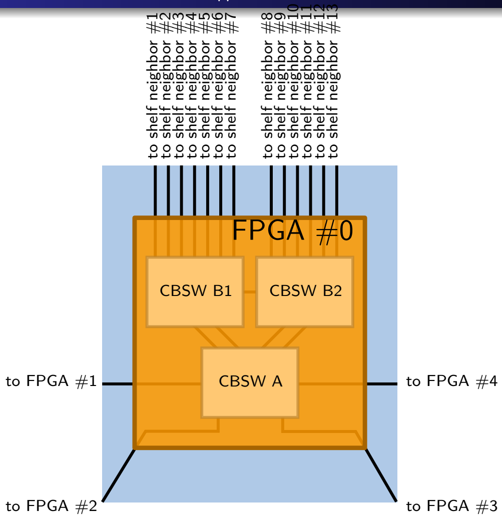


Project File:	final.xise	**Parser Errors:**	No Errors
Module Name:	topmodule_9x4_with_broad_sc	**Implementation State:**	Programming File Generated
Target Device:	xc4vfx60-10ff672	• **Errors:**	No Errors
Product Version:	ISE 14.1	• **Warnings:**	9923 Warnings (9901 new)
Design Goal:	Balanced	• **Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	All Constraints Met
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

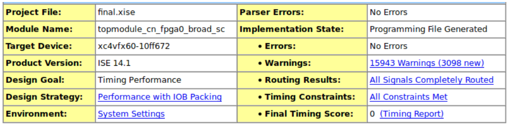
Device Utilization Summary [-]

Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	8,786	50,560	17%	
Number used as Flip Flops	8,591			
Number used as Latches	195			
Number of 4 input LUTs	18,124	50,560	35%	
Number of occupied Slices	12,374	25,280	48%	
Number of Slices containing only related logic	12,374	12,374	100%	
Number of Slices containing unrelated logic	0	12,374	0%	
Total Number of 4 input LUTs	18,487	50,560	36%	
Number used as logic	18,111			
Number used as a route-thru	363			
Number used as Shift registers	13			
Number of bonded IOBs	35	352	9%	
Number of BUFG/BUFGCTRLs	2	32	6%	
Number used as BUFGs	2			
Number of FIFO16/RAMB16s	12	232	5%	
Number used as RAMB16s	12			
Average Fanout of Non-Clock Nets	3.93			

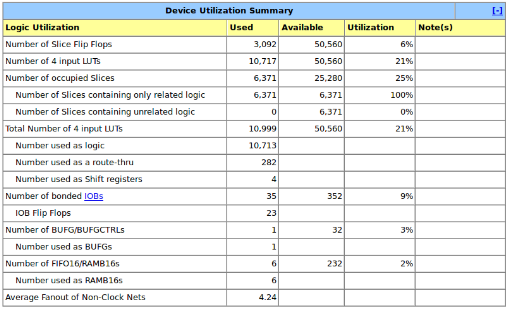
Test project for an FPGA #0



Test project for an FPGA #0



Project File:	final.xise	**Parser Errors:**	No Errors
Module Name:	topmodule_cn_fpga0_broad_sc	**Implementation State:**	Programming File Generated
Target Device:	xc4vfx60-10ff672	**• Errors:**	No Errors
Product Version:	ISE 14.1	**• Warnings:**	15943 Warnings (3098 new)
Design Goal:	Timing Performance	**• Routing Results:**	All Signals Completely Routed
Design Strategy:	Performance with IOB Packing	**• Timing Constraints:**	All Constraints Met
Environment:	System Settings	**• Final Timing Score:**	0 (Timing Report)



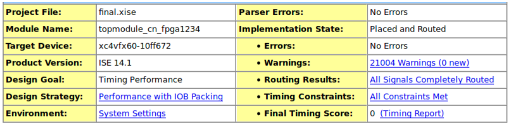
Device Utilization Summary [-]

Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	3,092	50,560	6%	
Number of 4 input LUTs	10,717	50,560	21%	
Number of occupied Slices	6,371	25,280	25%	
Number of Slices containing only related logic	6,371	6,371	100%	
Number of Slices containing unrelated logic	0	6,371	0%	
Total Number of 4 input LUTs	10,999	50,560	21%	
Number used as logic	10,713			
Number used as a route-thru	282			
Number used as Shift registers	4			
Number of bonded IOBs	35	352	9%	
IOB Flip Flops	23			
Number of BUFG/BUFGCTRLs	1	32	3%	
Number used as BUFGs	1			
Number of FIFO16/RAMB16s	6	232	2%	
Number used as RAMB16s	6			
Average Fanout of Non-Clock Nets	4.24			

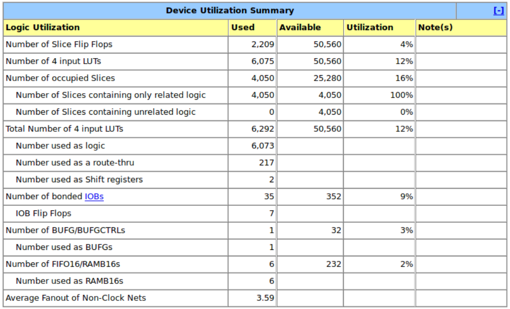
Test project for an algorithm FPGA

- Crossbar Switch with 10 interfaces
- 1 dummy sender
- 9 dummy receivers

Test project for an algorithm FPGA



Project File:	final.xise	**Parser Errors:**	No Errors
Module Name:	topmodule_cn_fpga1234	**Implementation State:**	Placed and Routed
Target Device:	xc4vfx60-10ff672	**• Errors:**	No Errors
Product Version:	ISE 14.1	**• Warnings:**	21004 Warnings (0 new)
Design Goal:	Timing Performance	**• Routing Results:**	All Signals Completely Routed
Design Strategy:	Performance with IOB Packing	**• Timing Constraints:**	All Constraints Met
Environment:	System Settings	**• Final Timing Score:**	0 (Timing Report)



Device Utilization Summary [-]

Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	2,209	50,560	4%	
Number of 4 input LUTs	6,075	50,560	12%	
Number of occupied Slices	4,050	25,280	16%	
Number of Slices containing only related logic	4,050	4,050	100%	
Number of Slices containing unrelated logic	0	4,050	0%	
Total Number of 4 input LUTs	6,292	50,560	12%	
Number used as logic	6,073			
Number used as a route-thru	217			
Number used as Shift registers	2			
Number of bonded IOBs	35	352	9%	
IOB Flip Flops	7			
Number of BUFG/BUFGCTRLs	1	32	3%	
Number used as BUFGs	1			
Number of FIFO16/RAMB16s	6	232	2%	
Number used as RAMB16s	6			
Average Fanout of Non-Clock Nets	3.59			

RocketIO Instantiation

- So far, only internal signals have been used for data transport
- Real life scenarios require the use of RocketIO/Aurora cores, which need to be instantiated

RocketIO Instantiation

- So far, only internal signals have been used for data transport
- Real life scenarios require the use of RocketIO/Aurora cores, which need to be instantiated
- Possible issues: The high latency (some 100 clock cycles) will at least require tuning of the timeouts.

RocketIO Instantiation

- So far, only internal signals have been used for data transport
- Real life scenarios require the use of RocketIO/Aurora cores, which need to be instantiated
- Possible issues: The high latency (some 100 clock cycles) will at least require tuning of the timeouts.

Slow Control

- The supervisor PC needs to know the network of the FPGAs
- Therefore, it needs to contact each FPGA and ask it for its “inventory” (Crossbar Switches and Dialers)

Slow Control

- The supervisor PC needs to know the network of the FPGAs
- Therefore, it needs to contact each FPGA and ask it for its “inventory” (Crossbar Switches and Dialers)
- Each Crossbar Switch must be asked for their respective neighbors

Slow Control

- The supervisor PC needs to know the network of the FPGAs
- Therefore, it needs to contact each FPGA and ask it for its “inventory” (Crossbar Switches and Dialers)
- Each Crossbar Switch must be asked for their respective neighbors
- Requires some mechanism of communication between PC and FPGAs (Ethernet? IPMI?)

Slow Control

- The supervisor PC needs to know the network of the FPGAs
- Therefore, it needs to contact each FPGA and ask it for its “inventory” (Crossbar Switches and Dialers)
- Each Crossbar Switch must be asked for their respective neighbors
- Requires some mechanism of communication between PC and FPGAs (Ethernet? IPMI?)

Supervisor PC software

- Implementation of Dijkstra’s algorithm
- Or any other suitable mechanism to determine routes (e.g. manual definition)

Supervisor PC software

- Implementation of Dijkstra's algorithm
- Or any other suitable mechanism to determine routes (e.g. manual definition)

Thank you for your attention