1977-12

First ICTP Regional Microelectronics Workshop and Training on VHDL for Hardware Synthesis and FPGA Design in Asia-Pacific

16 June - 11 July, 2008

FPGA applications in High Energy Physics.

KLUGE Alexander
PH ESE FE Division
C E R N
385, rte Meyrin
CH-1211 Geneva 23
SWITZERLAND

FPGA applications in High Energy Physics

Alexander Kluge

CERN

Outline

- **CERN – electronics system concepts**
  a project cycle

- **Application: Data selection**

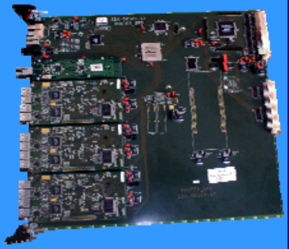
- **Application: Data processing**

CERN

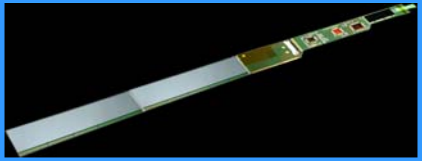
Application overview CERN

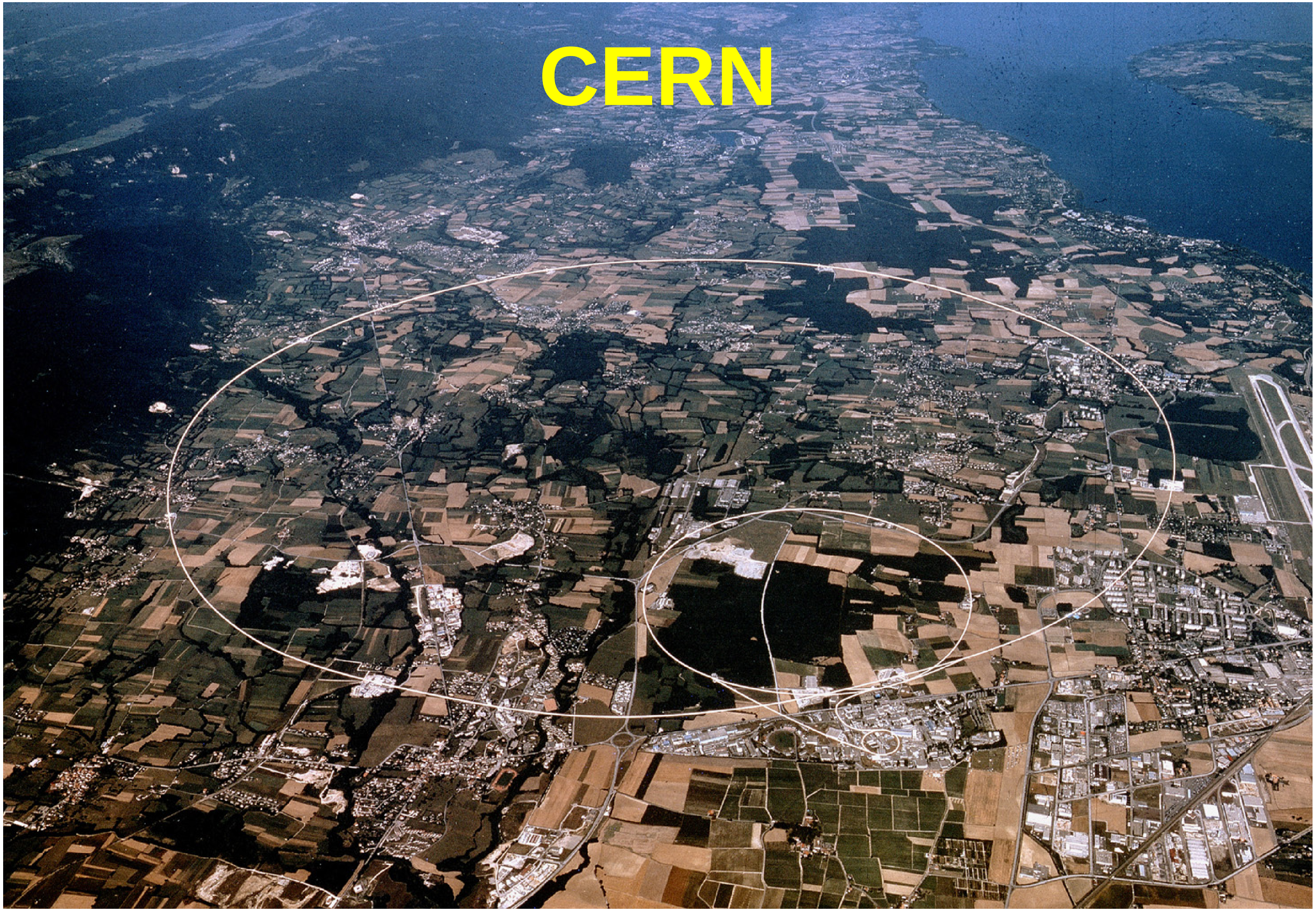
- CERN, experiments
  - Aim
  - General detector concept
  - Examples

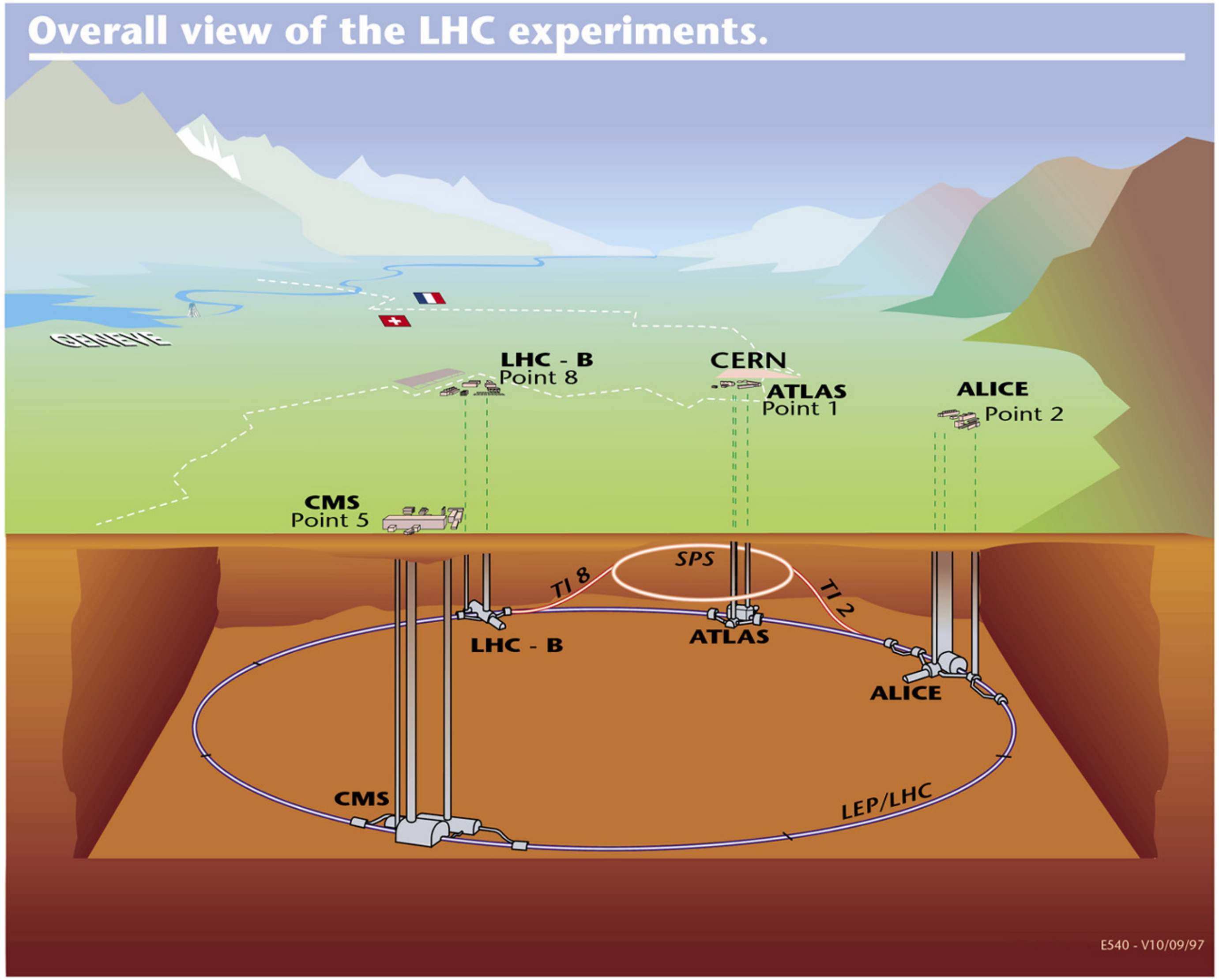
Principle of data acquisition & data flow

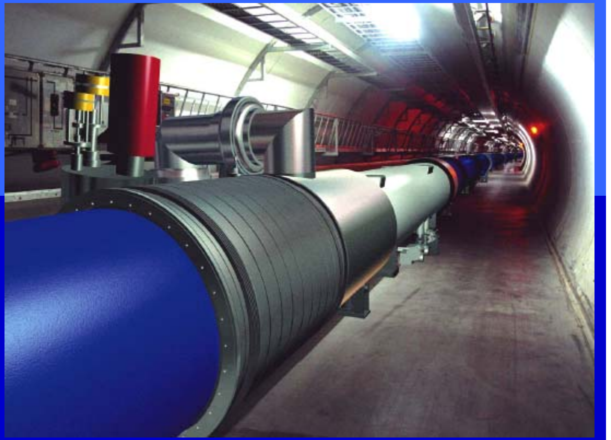


- data selection: trigger
- data processing



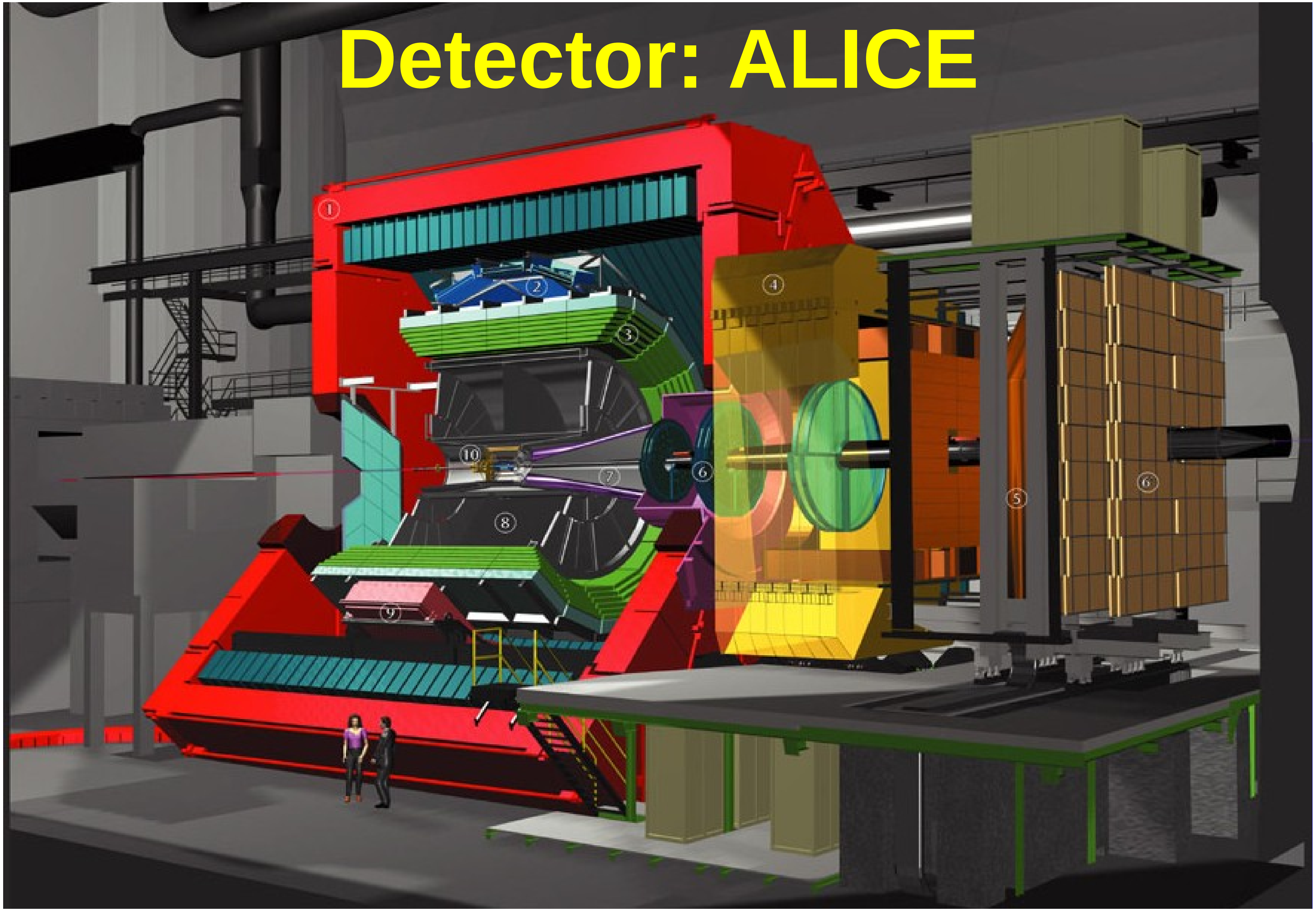

CERN


Overall view of the LHC experiments.
GENEVE
LHC - B
Point 8
CERN
ATLAS
Point 1
ALICE
Point 2
CMS
Point 5
SPS
TI 8
TI 2
LHC - B
ATLAS
ALICE
CMS
LEP/LHC
E540 - V10/09/97

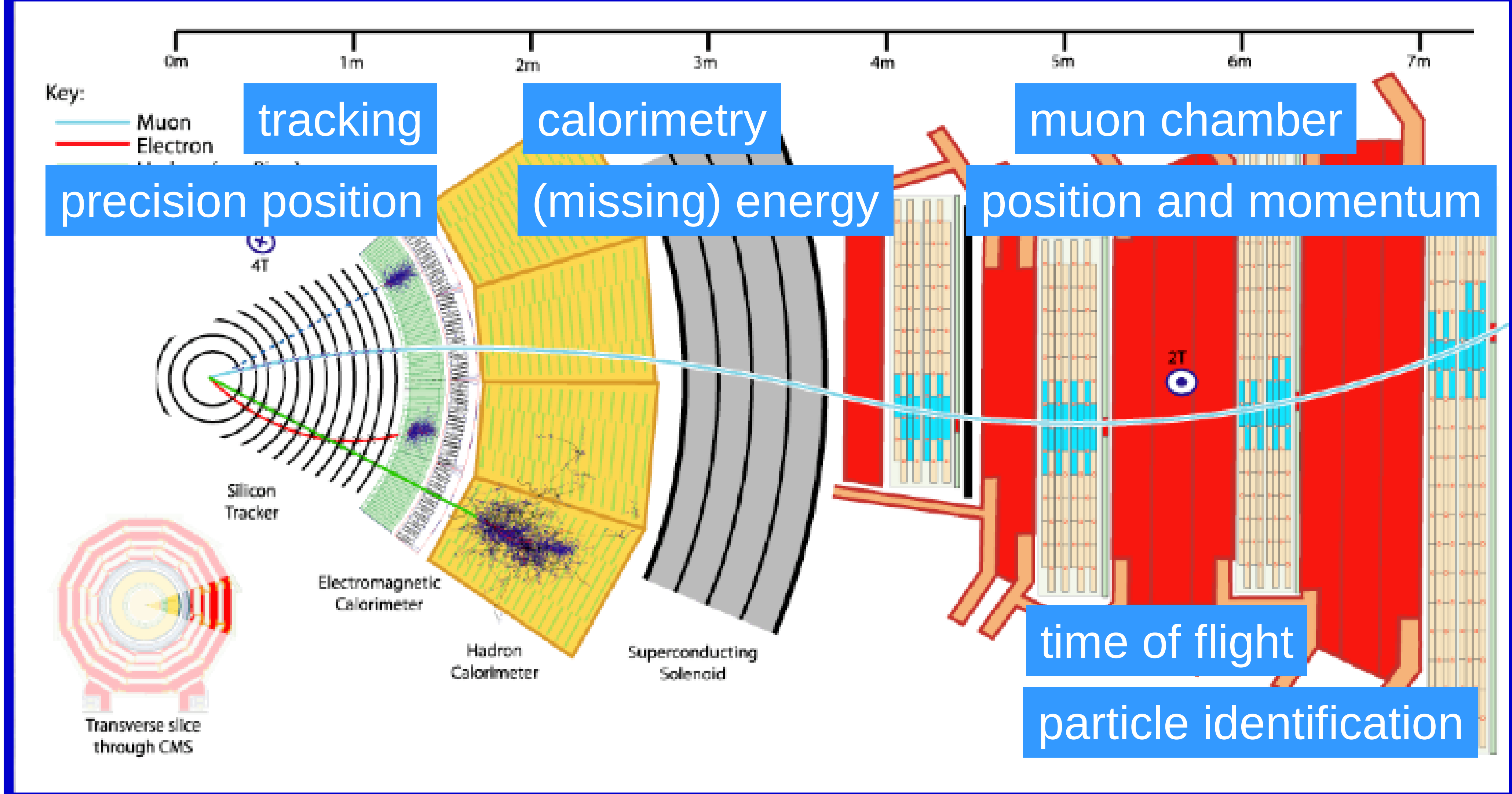


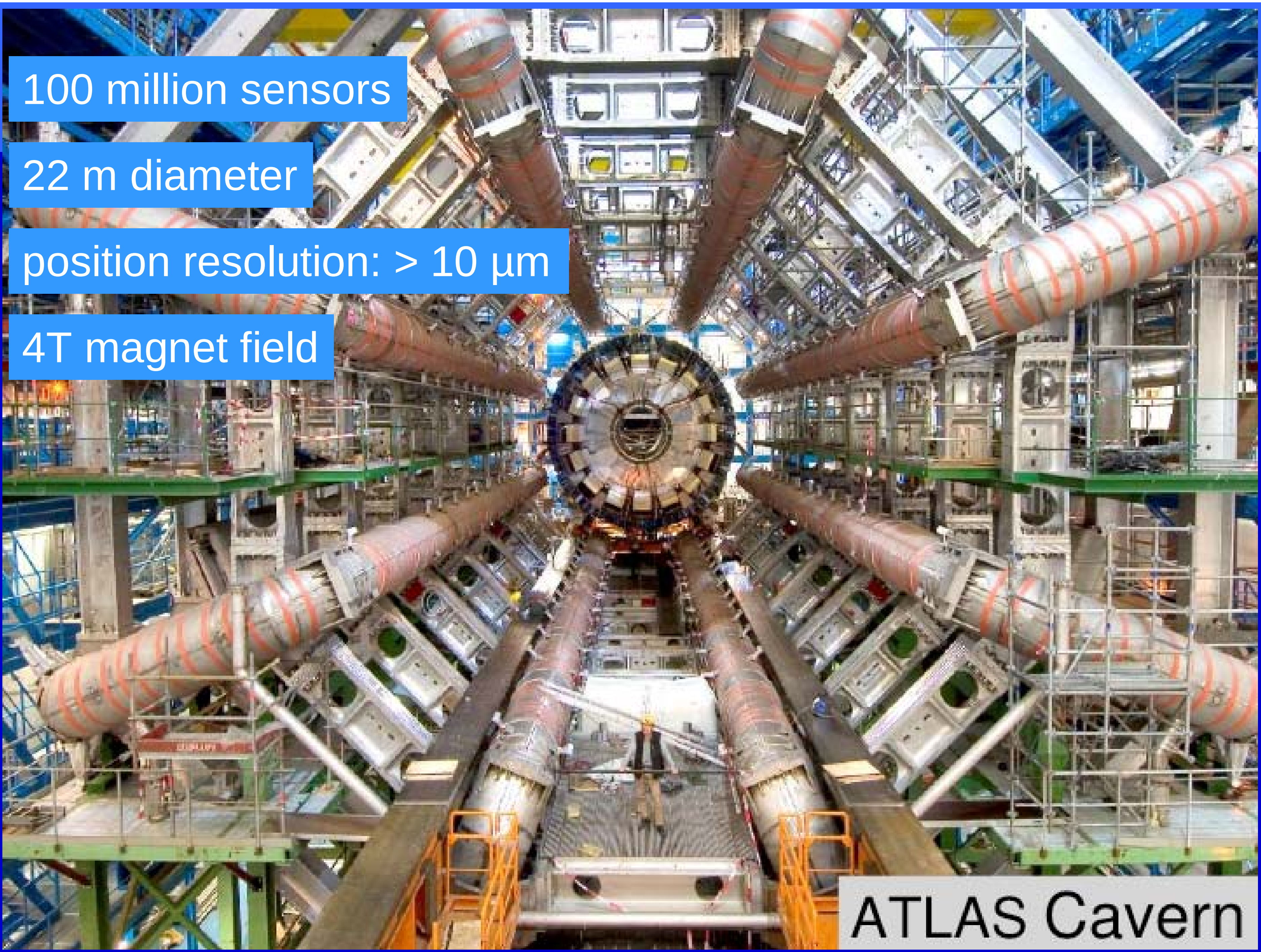
Experiments

Detector: ALICE



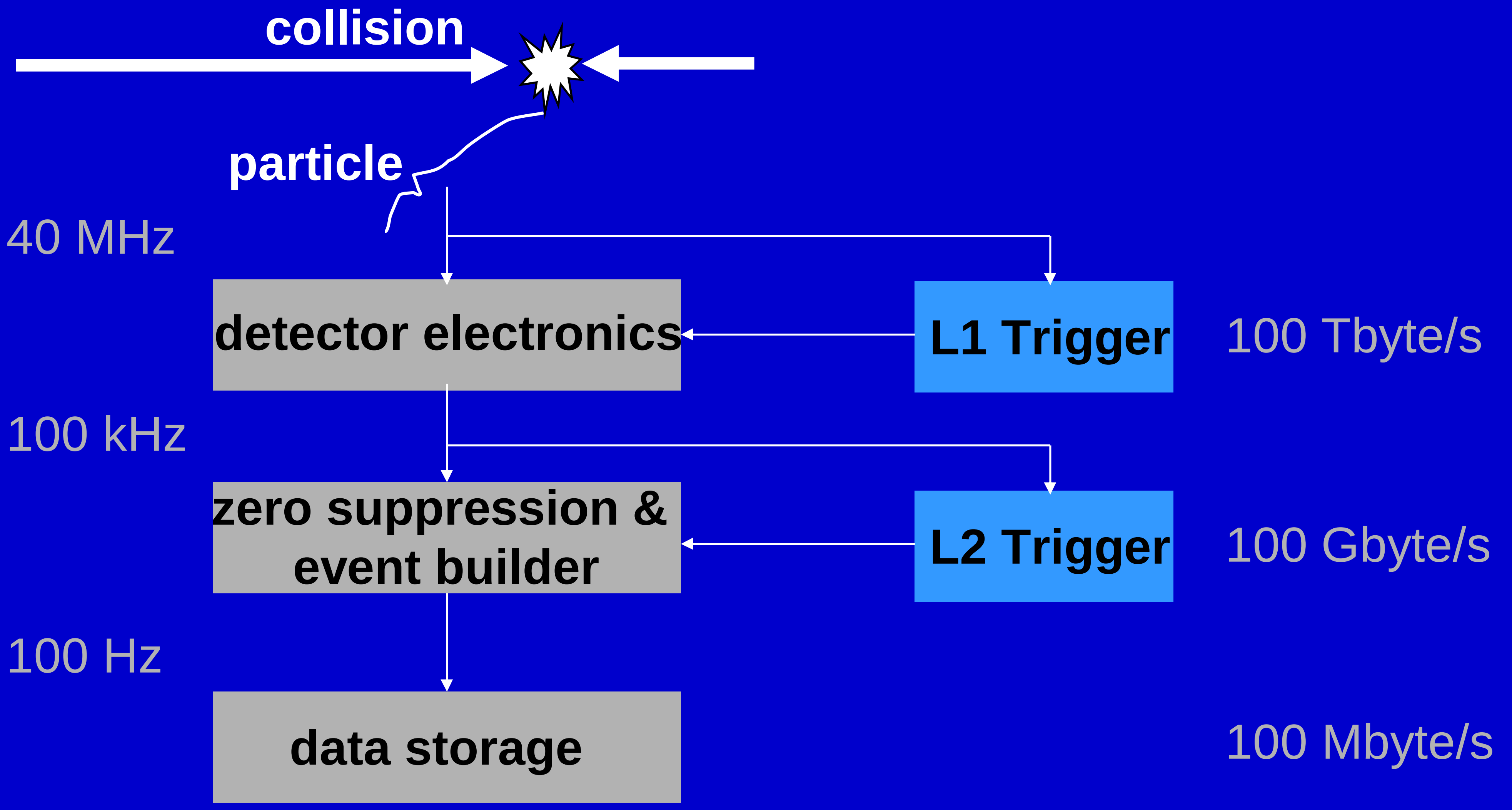
Principle of detectors



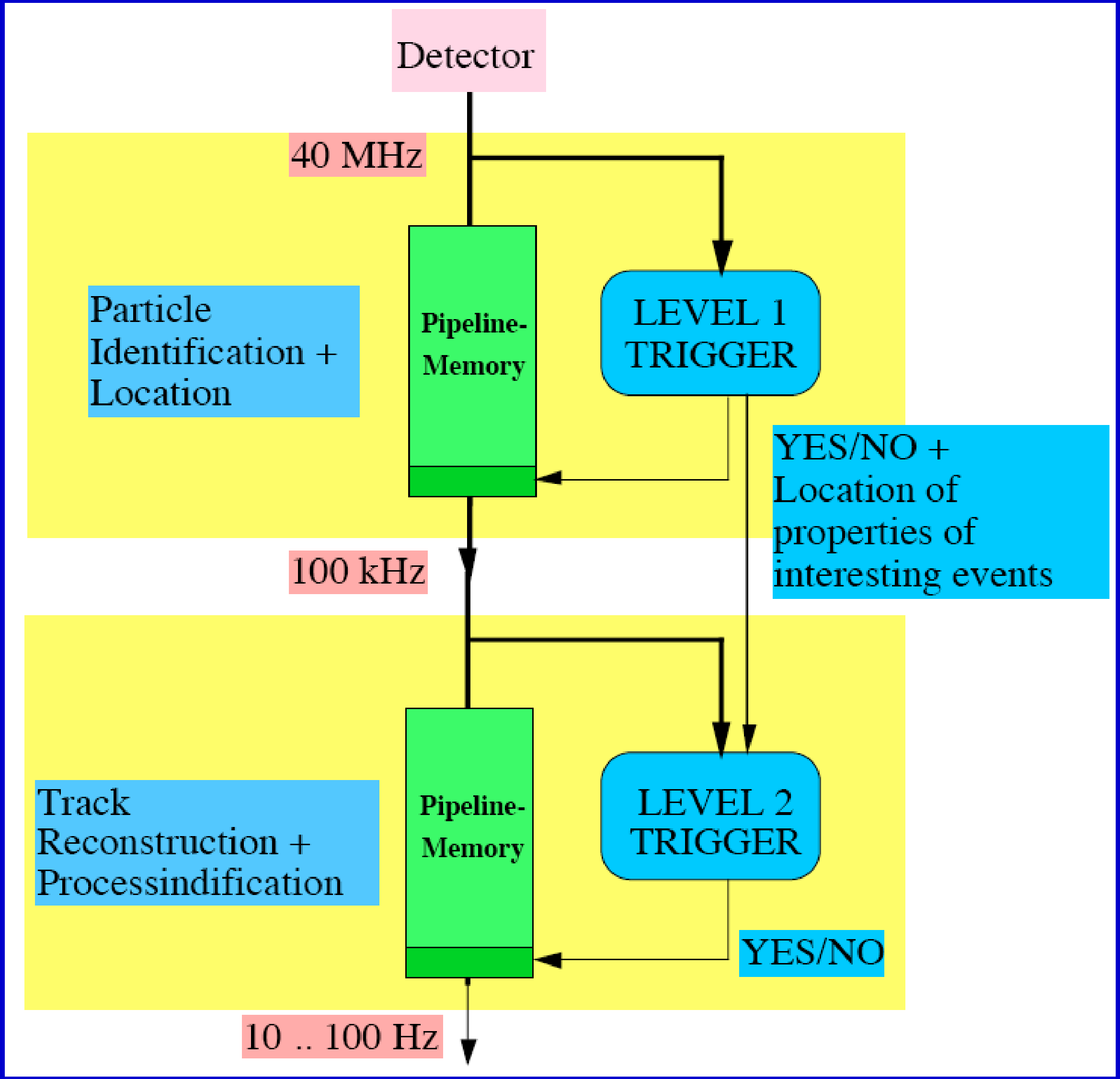

100 million sensors
22 m diameter
position resolution: > 10 µm
4T magnet field
ATLAS Cavern

Principle of Data acquisition

Data selection

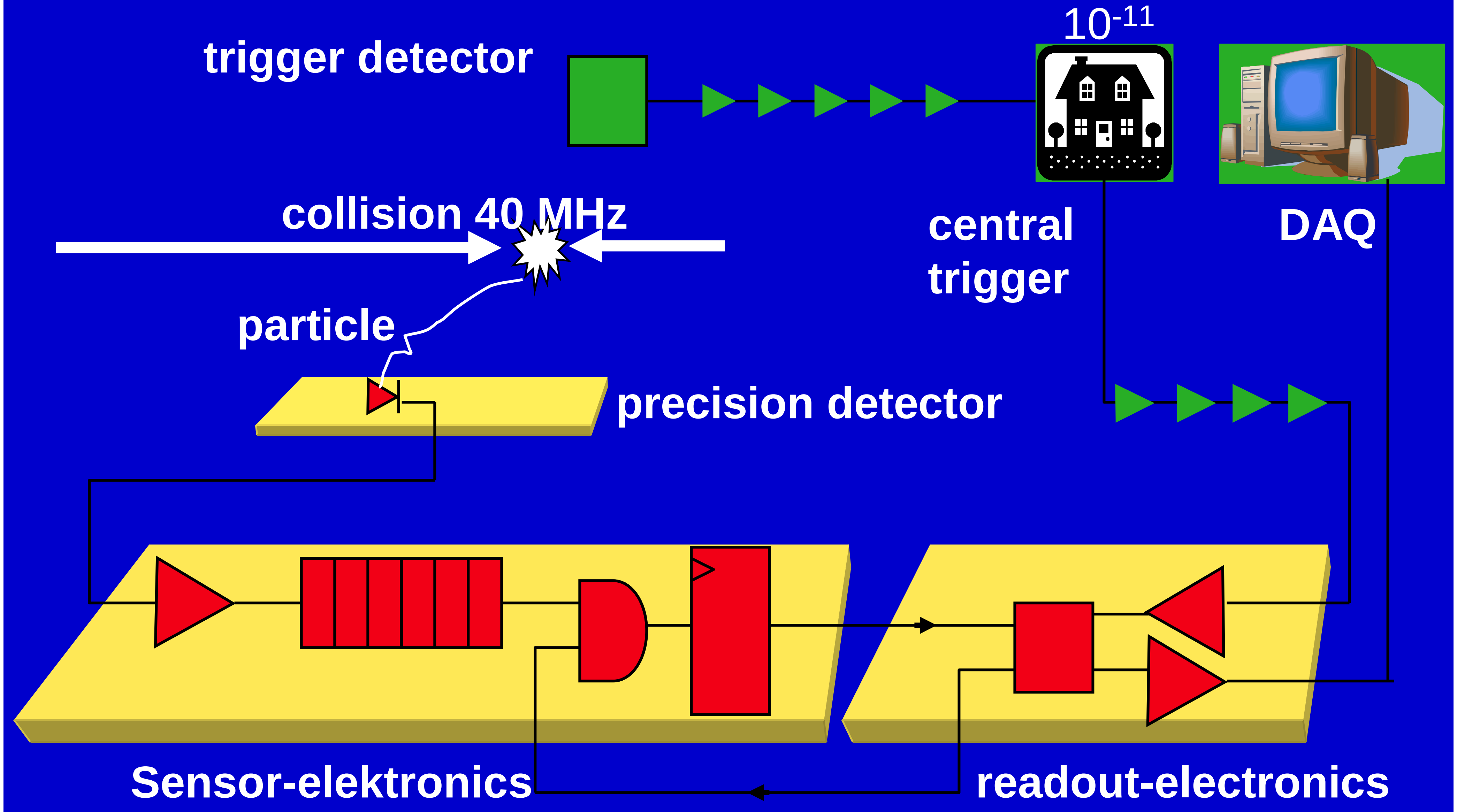


Data selection



Detector
40 MHz
Particle Identification + Location
Pipeline-Memory
LEVEL 1 TRIGGER
YES/NO + Location of properties of interesting events
100 kHz
Track Reconstruction + Processindification
Pipeline-Memory
LEVEL 2 TRIGGER
YES/NO
10 .. 100 Hz

Principle of data acquisition



Trigger processor

Challenge/Specifications

- **Fast**
  - the faster, the less data needs to be pipelined/stored
- **Compact**
  - Many data channels are going into one processor system
- **Connectivity**
  - High number
  - Transmission delay on cables (5ns/m -> 200 m -> 1µs)
- **Reliability**
  - Physics processes with a probability of 10^{-11} need to recorded
  - Processing and data transmission error rate >> 10^{-11}
- **Quality control**
  - Processes are verified in hardware and software processors
- **Radioactive environment**

Challenge/Specifications

- – Many (100.000) parallel inputs in 25 ns intervall
- – Parallel processing – pipelined processing
- – FPGA
  - • highly parallel because of many IOs and interconnectability
- – Example (Tracal Trigger)
  - • Specifications
    - – Calculate how many out of 1000 binary sensor inputs are active
    - – Each 25 ns a new set of 1000 bits
    - – Result required within 100 ns
  - • Solution possibilities
  - • Today and 5 and 10 years ago

Challenge/Specifications

System topology

- **high number of inputs ->**
  - operation to simplify data and reduce data amount
- **reduced number of inputs ->**
  - connected to more complex processing units
- **at the end of processing chain ->**
  - interest to integrate as much information into 1 FPGA to reduce interconnection

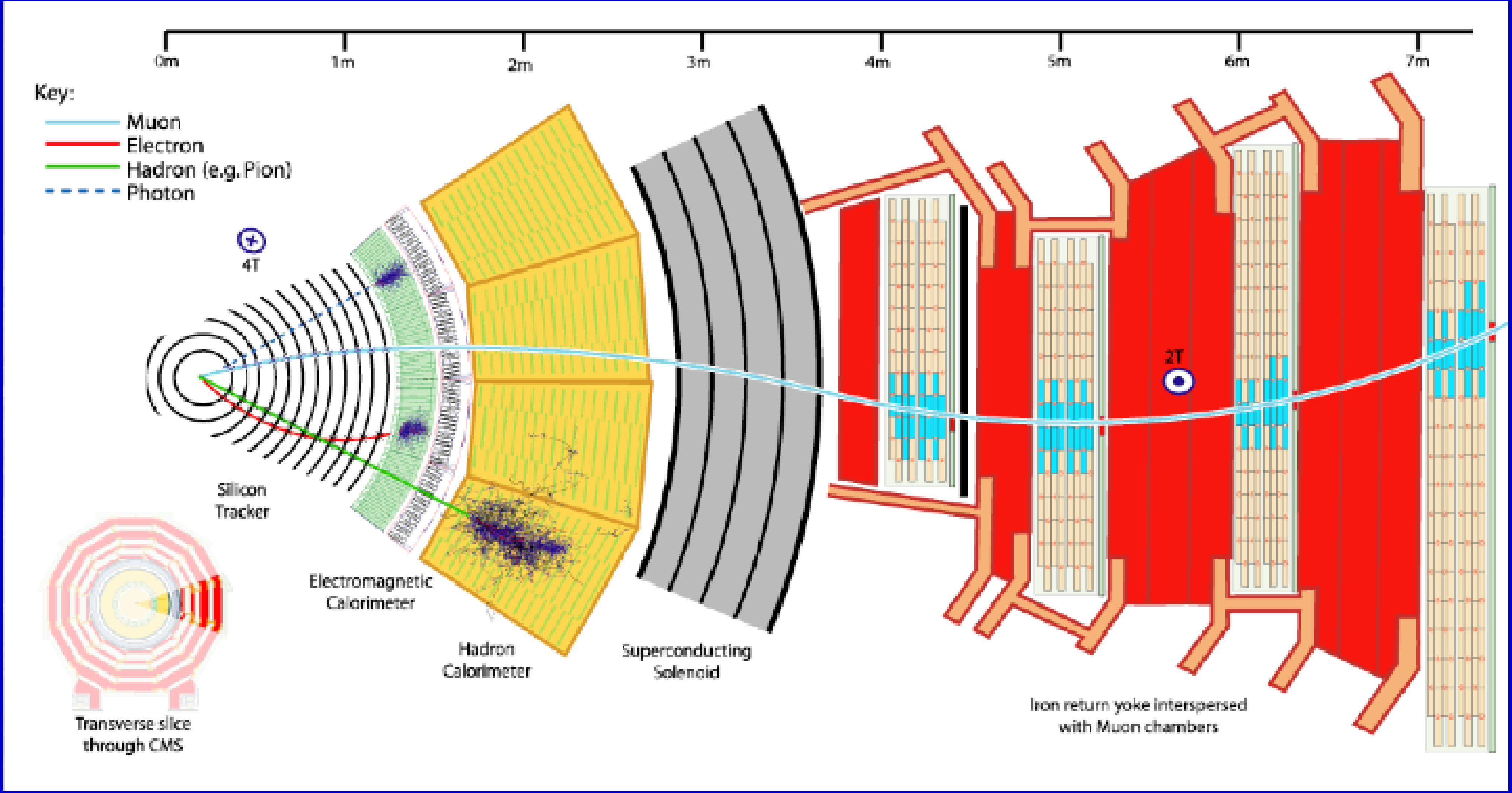
Challenge/Specifications

System topology

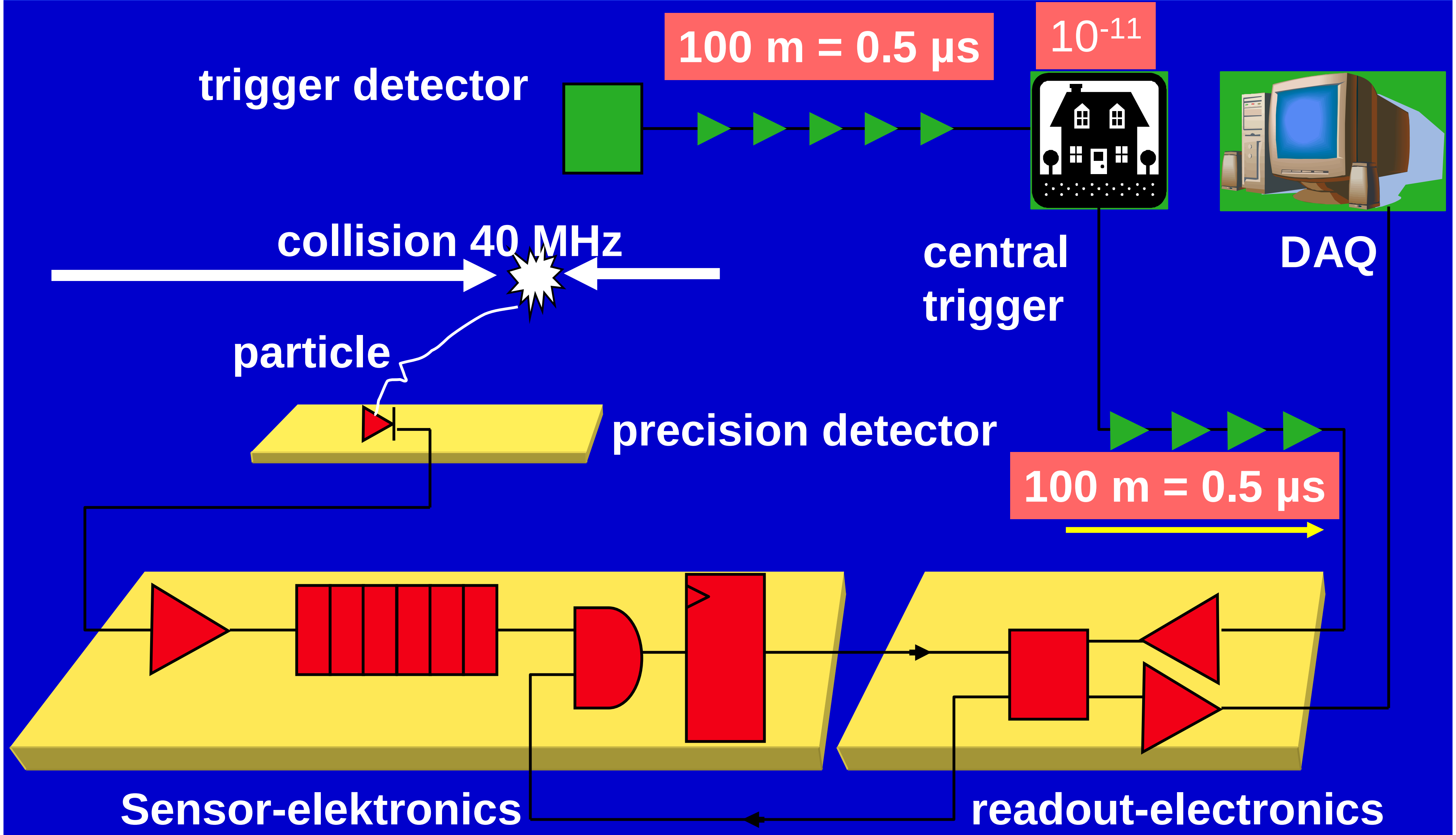
- **Interconnection:**
  - delay
    - (clock to pin, transmission outside FPGA, setup time)
  - Parallel interconnection:
    - high number of IOs, problem moved to board level
    - reliability impact due to solder joints or connectors
  - Serial interconnections at high speed
    - reduce reliability impact but increase delay (trigger needs to be fast)

Muon Track Finder Trigger Processor

Muon track finder trigger



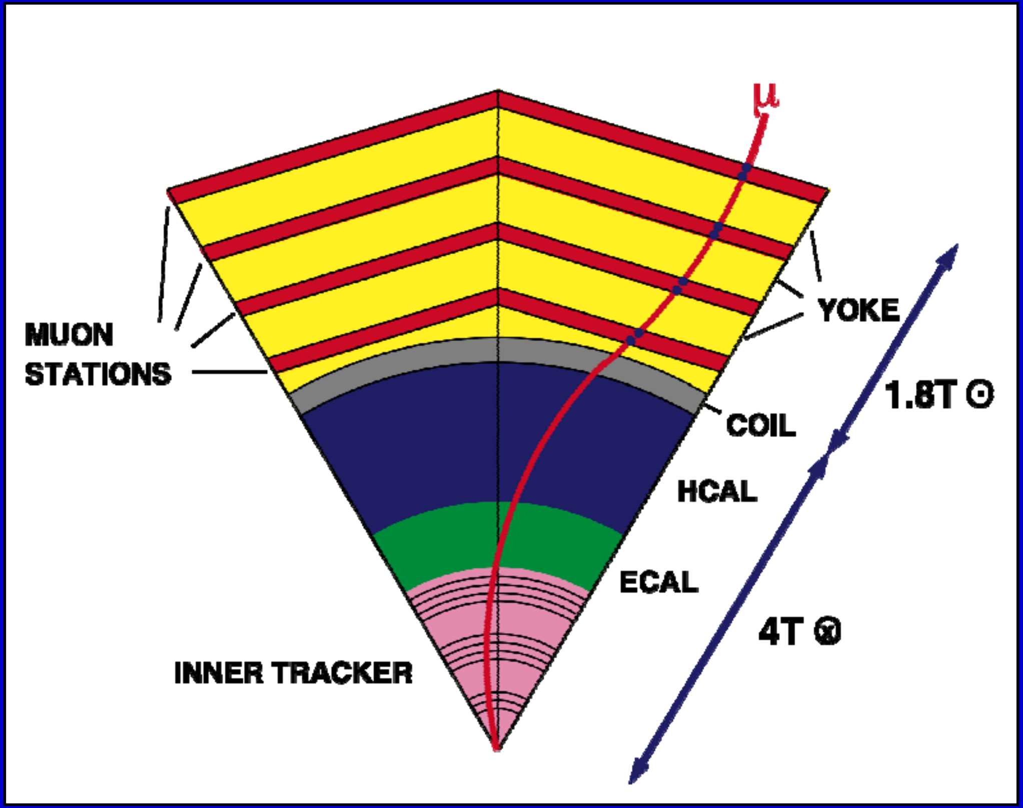
Principle of data acquisition

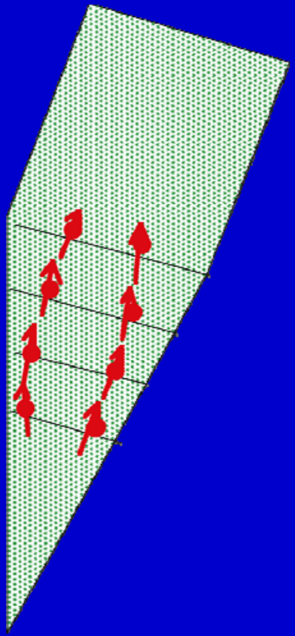


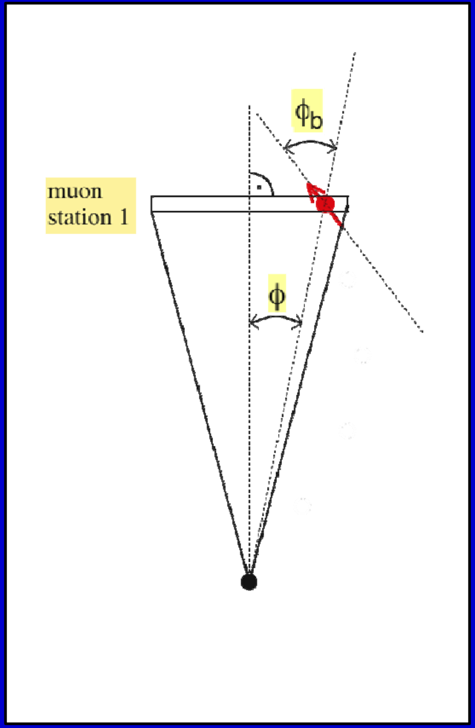
Muon track finder trigger

- **Size of detector system**
  - r = 14 m, length = 20 m
    - cable delay ~ 5 ns/m -> synchronisation
- **Each 25 ns new data set**
- **240 detector modules – 200.000 detector cells**
- **Identify particles (muons)**
- **Measure curvature = momentum of particles within 400 ns**
- **Find 4 particles with highest momentum**

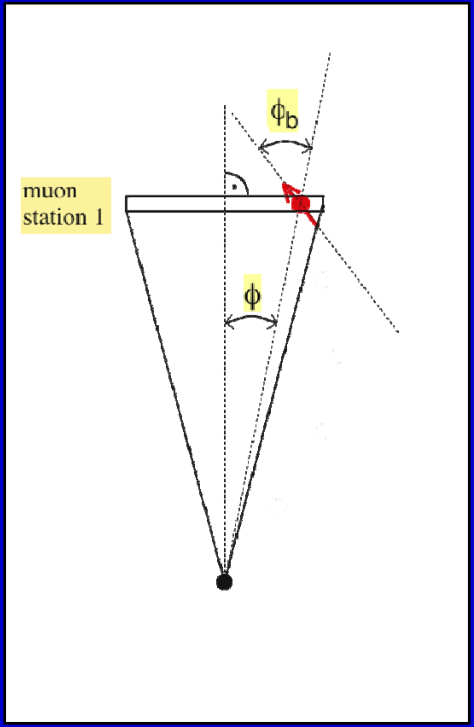
Muon track finder trigger







Muon track finder trigger



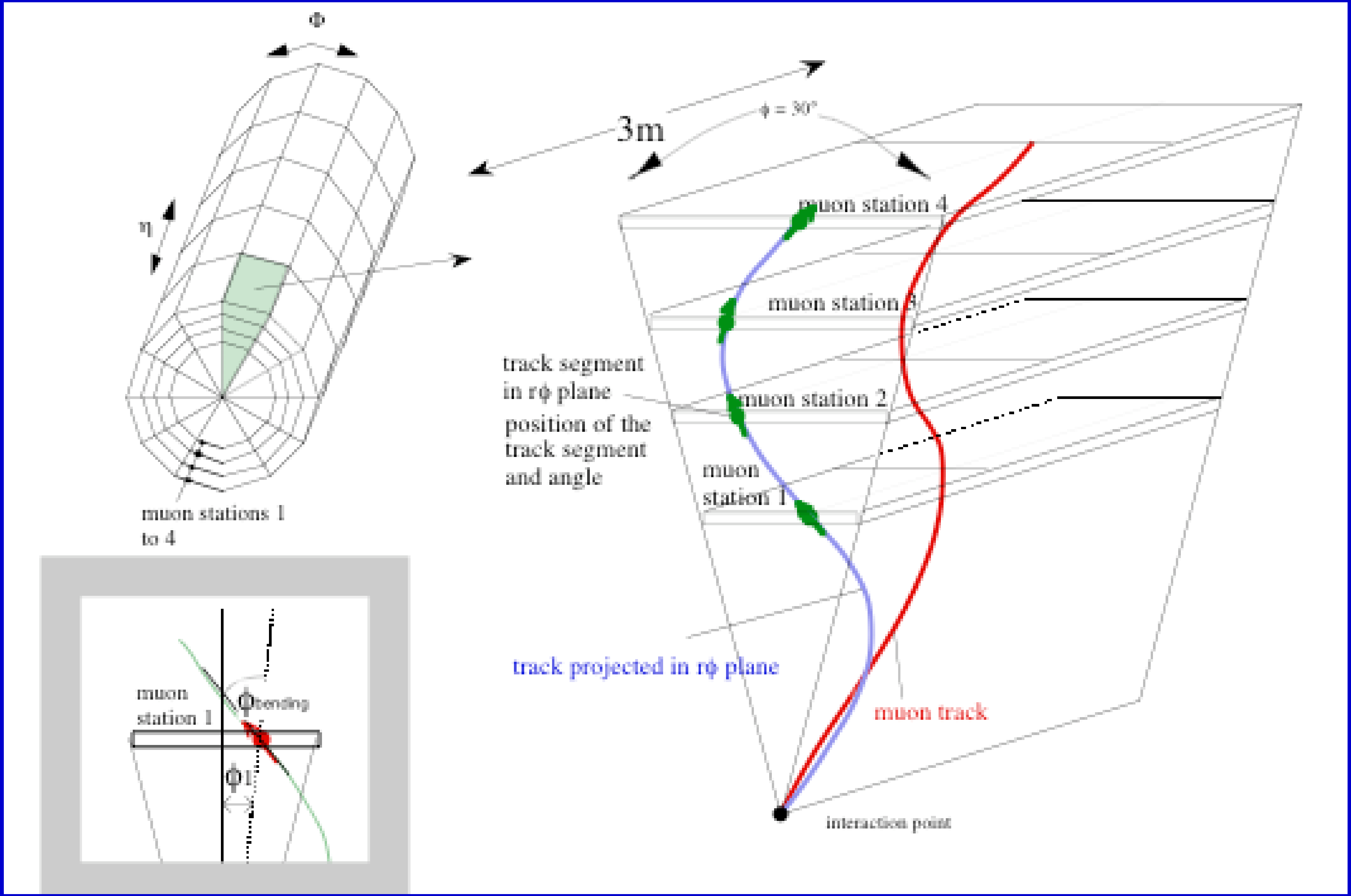
200.000 sensors ->

240 chambers x 2 track segments =
480 track segments

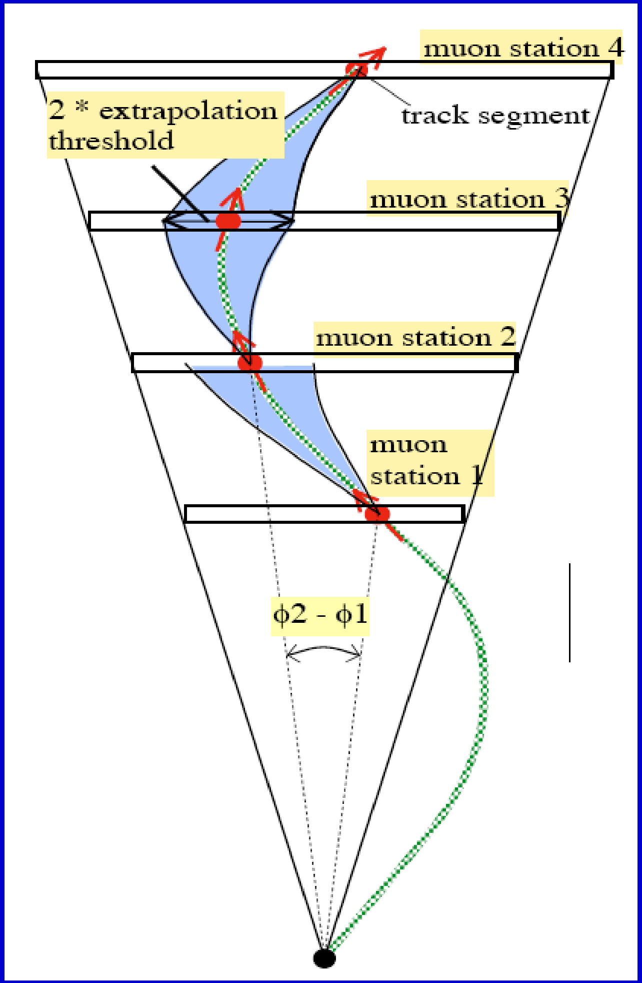
1 track segment
position (phi): 12 bits
angle (phi_b): 10 bits
quality code: 3 bits

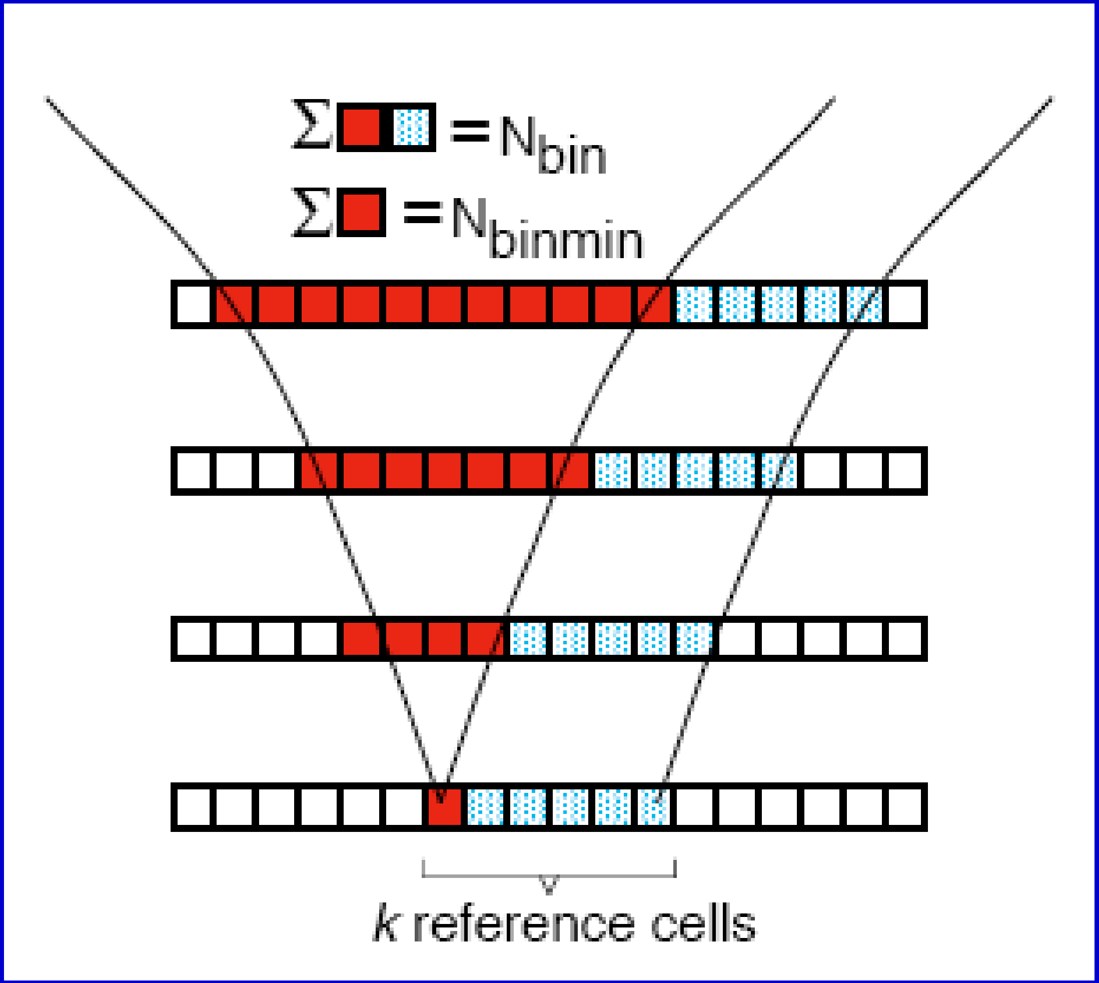
25 bits * 480 track segment = 12000 bits
12000 bits * 40 MHz = 480 Gbit/s

Muon track finder trigger

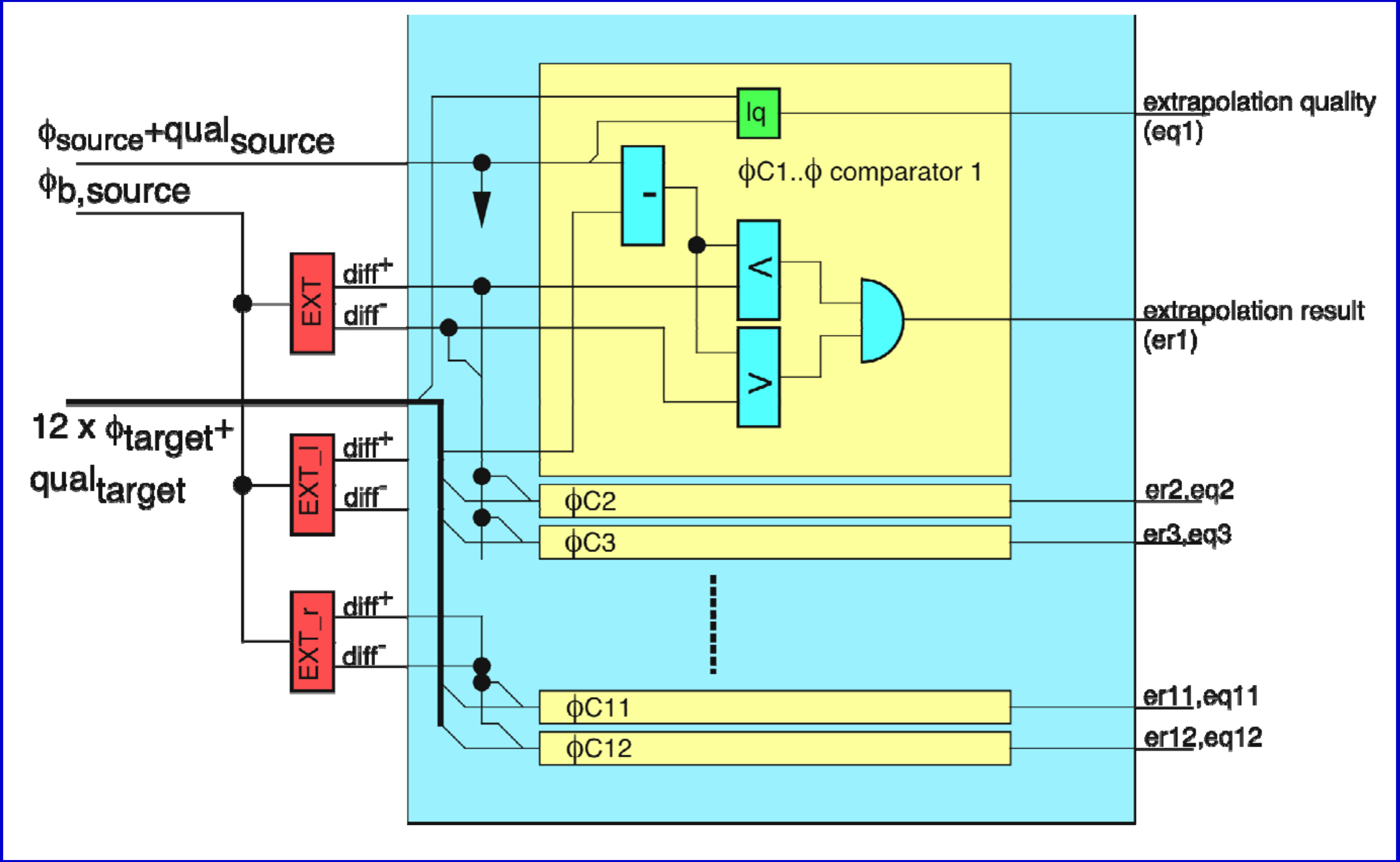


Muon track finder trigger

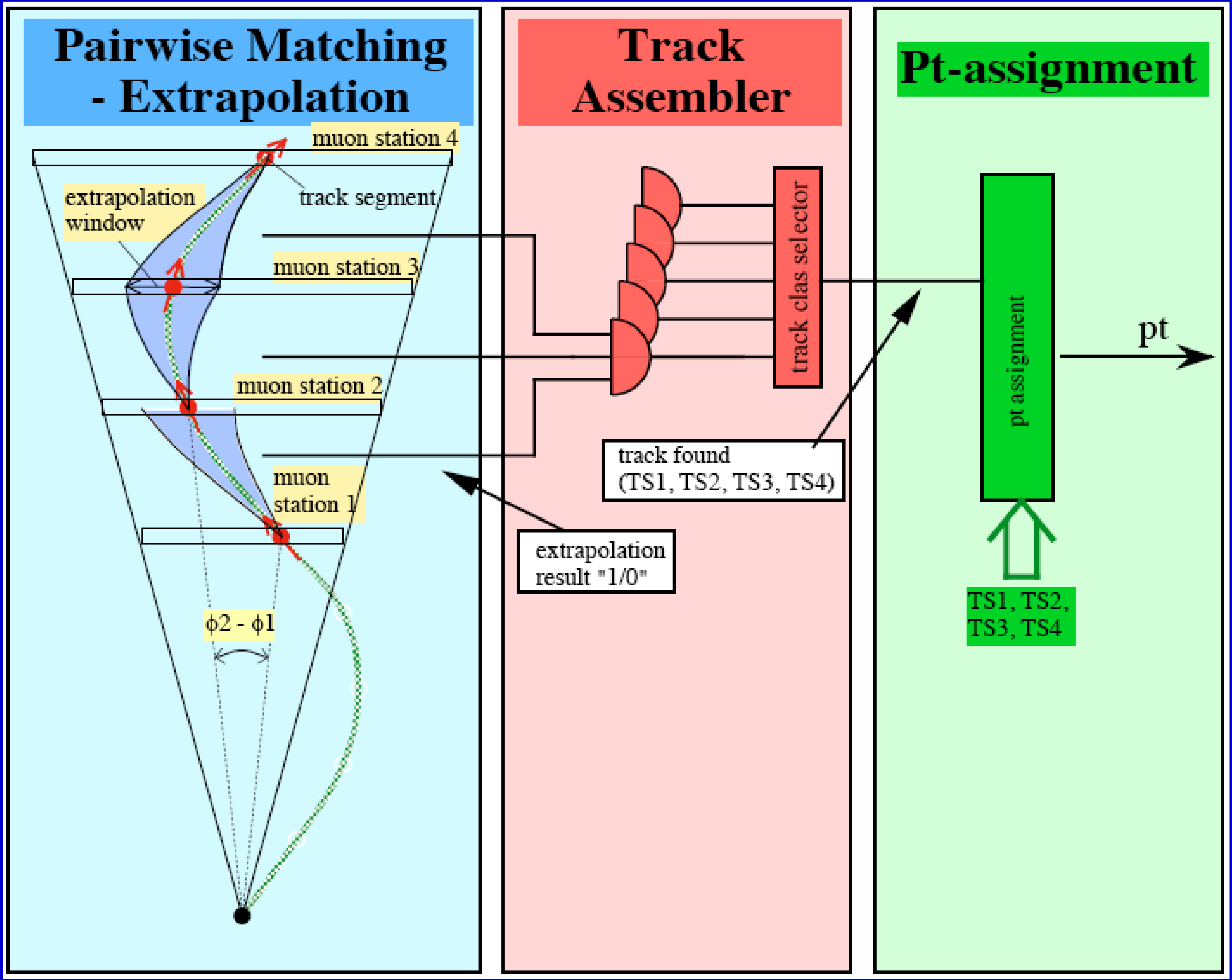




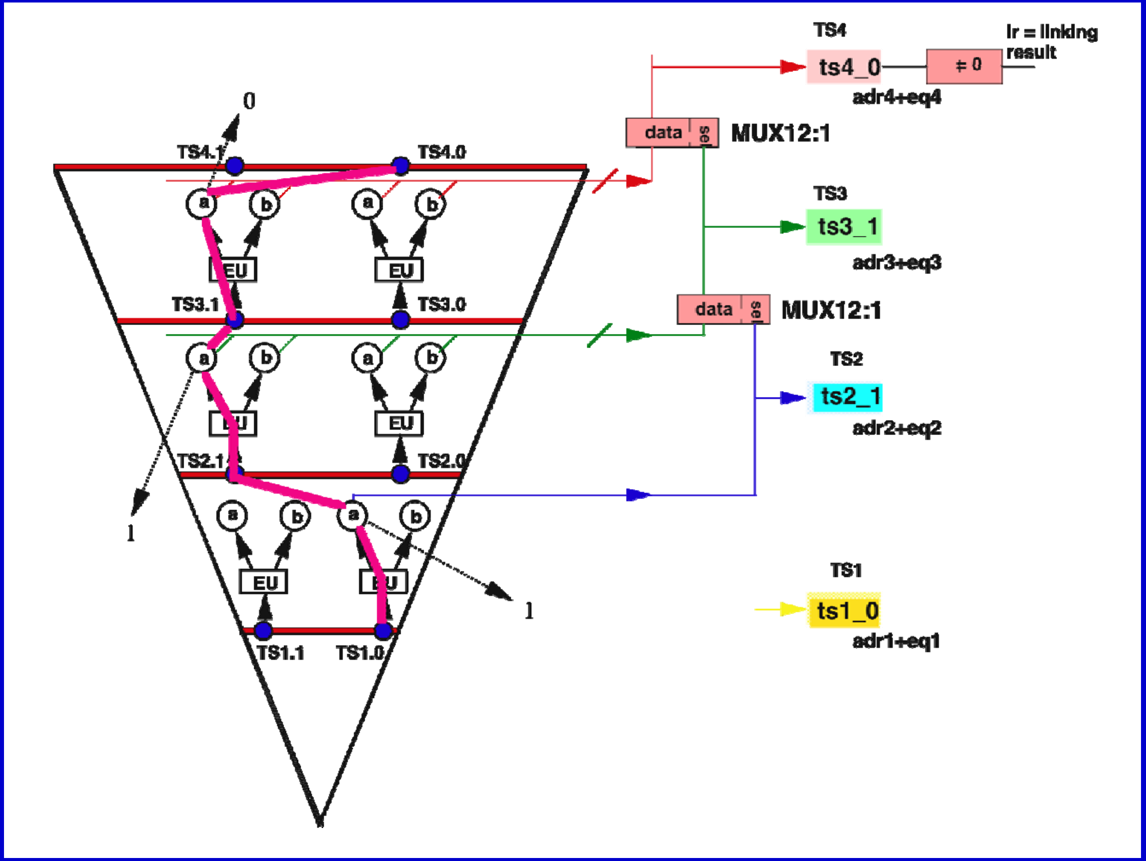
Muon track finder trigger



Muon track finder trigger

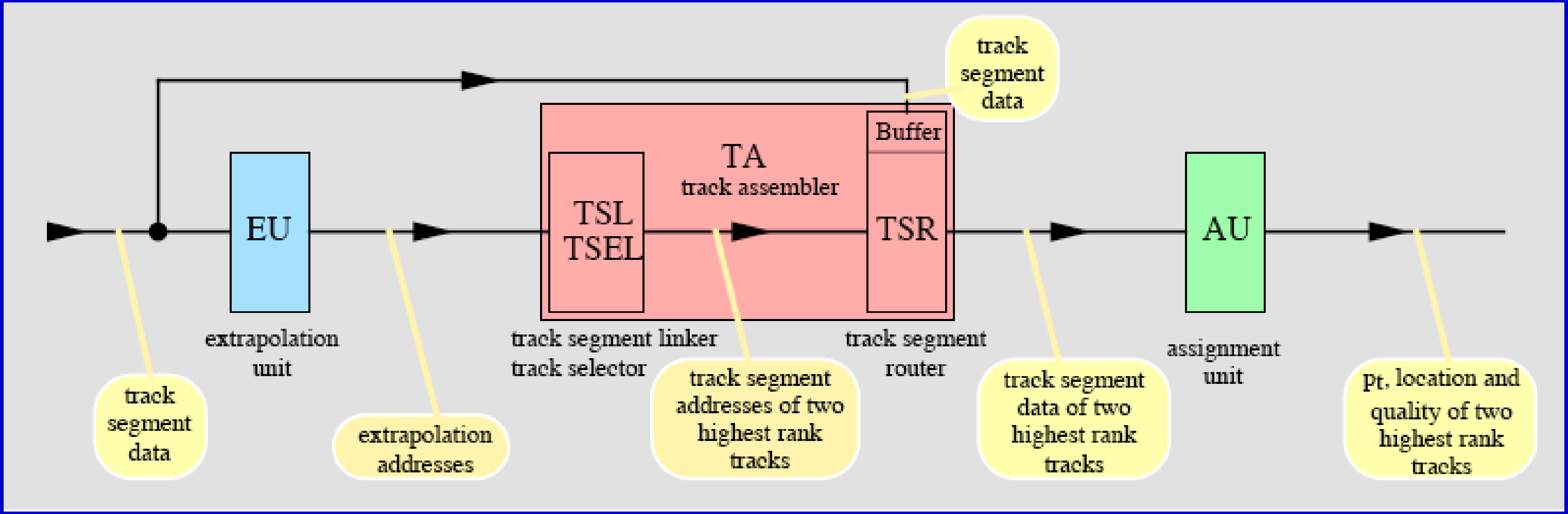


Muon track finder trigger



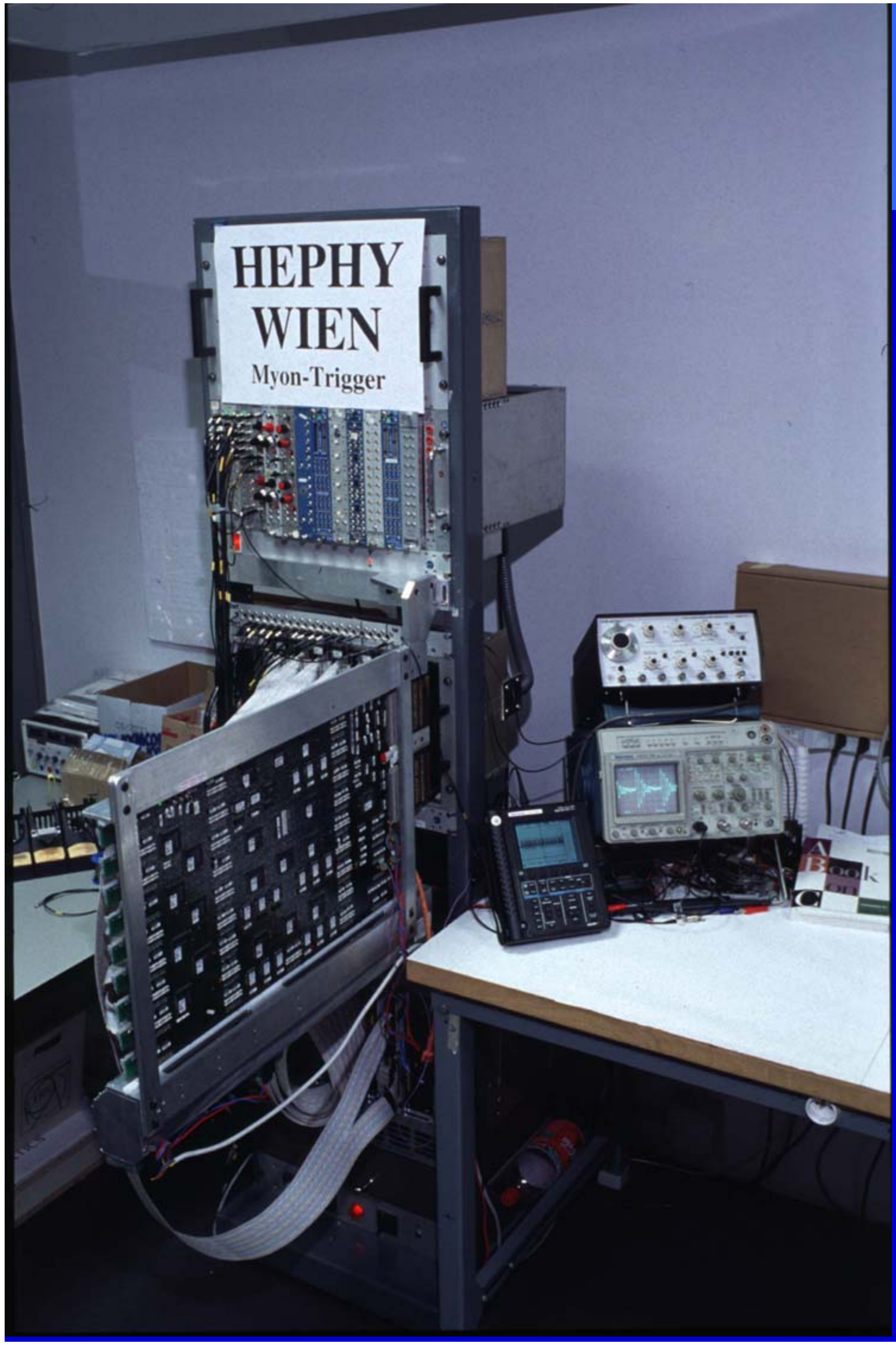
Result of all extrapolation units is 180 bits
-> data reduction
Track assembly units is combinatorial and looks for the longest possible track combination

Muon track finder trigger

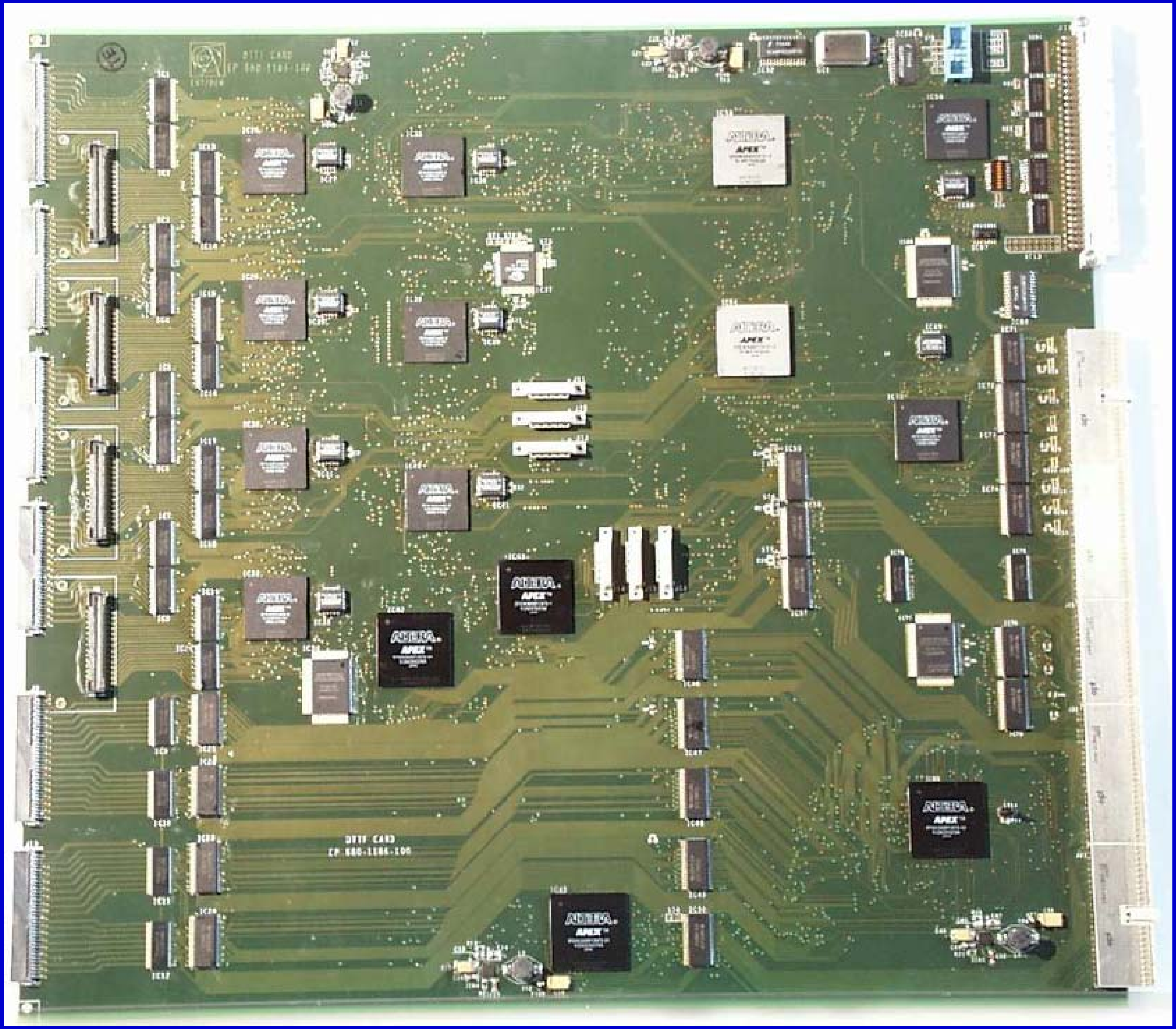


Parameter assignment unit: momentum (5 bits)
based on difference in position of layer 1 and 2

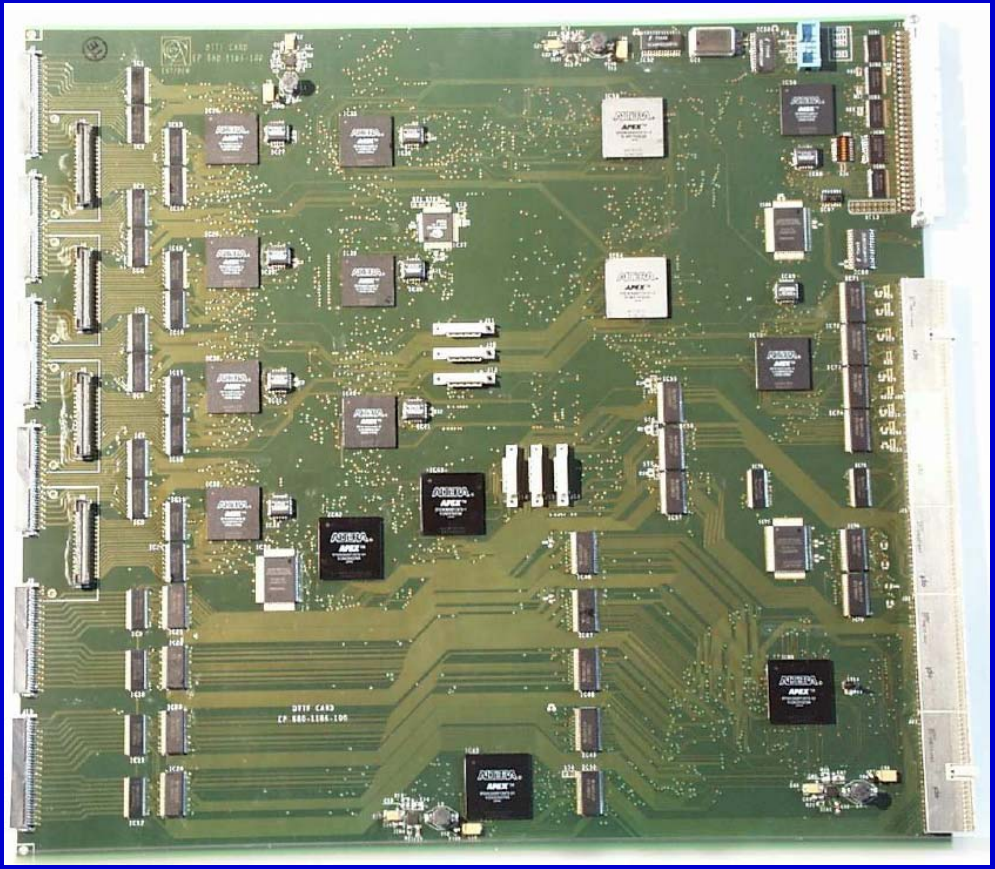
Muon track finder trigger



Muon track finder trigger



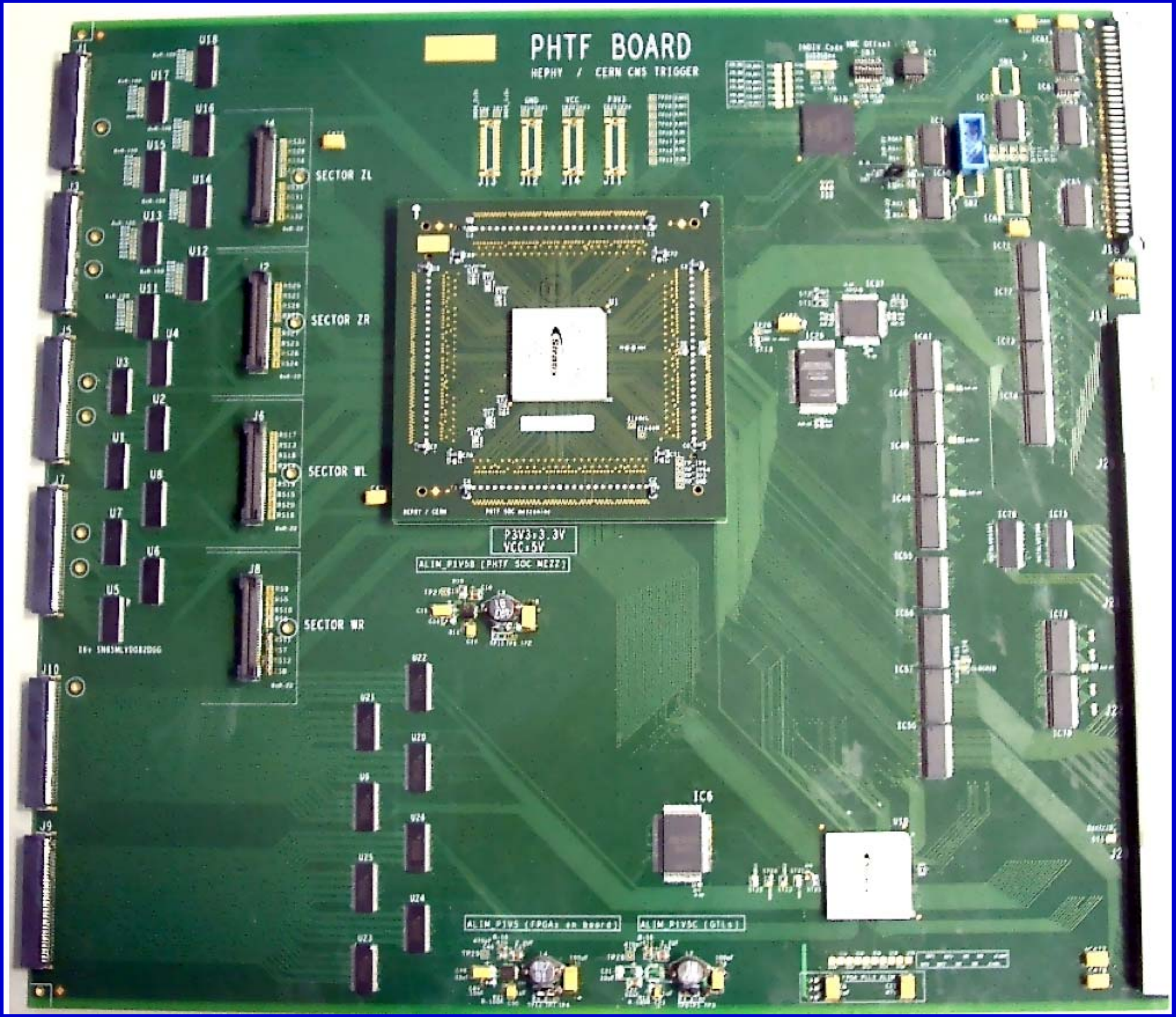
Muon track finder trigger



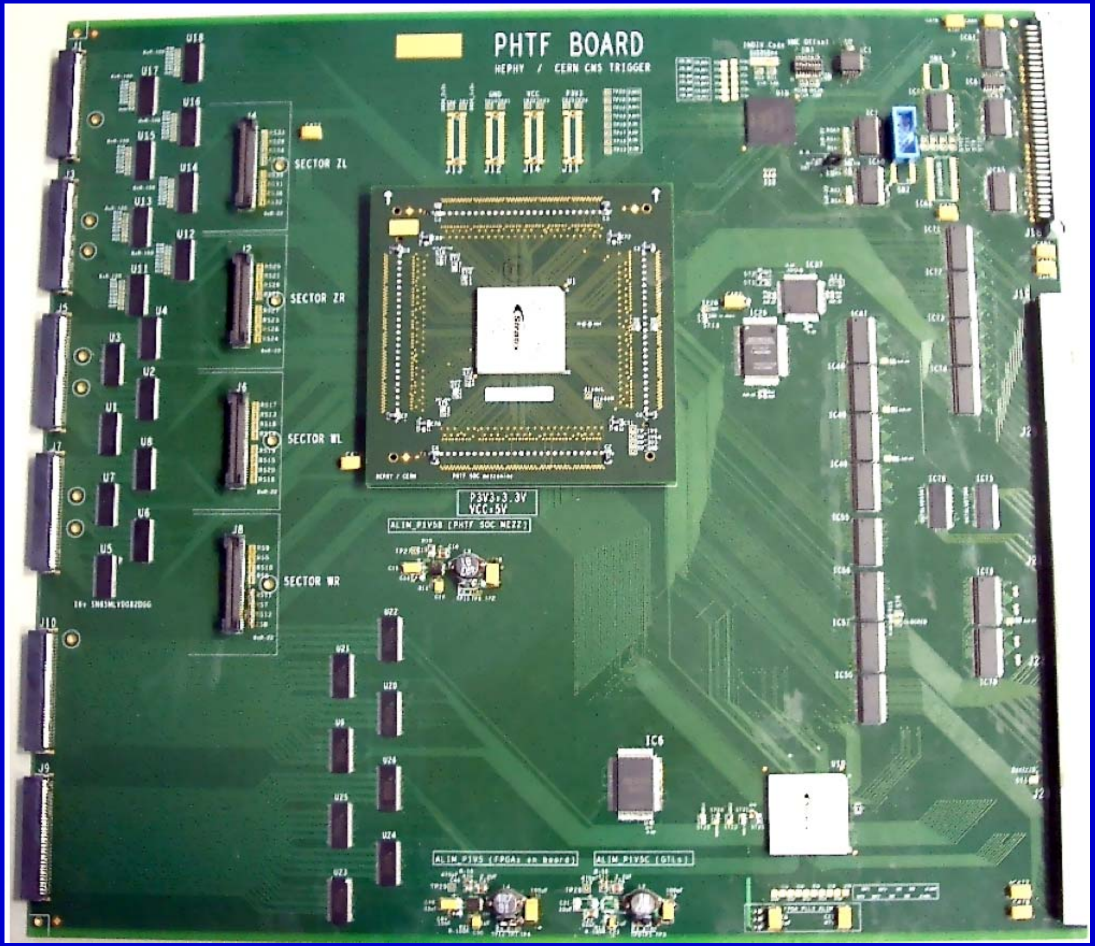
Extrapolation units: EP20k400EFC672
Data pipeline: 3 x EP1k100FC484
Track segment linker: EP20k300

16 layer PCB
no pin level back annotation
no board level simulation
Soldering problems with ball grid

Muon track finder trigger



Muon track finder trigger



All in EP1S40F1020C7

8 layer PCB
pin level back annotation
board level VHDL simulation
full JTAG boundary scan
FPGA on daughter card

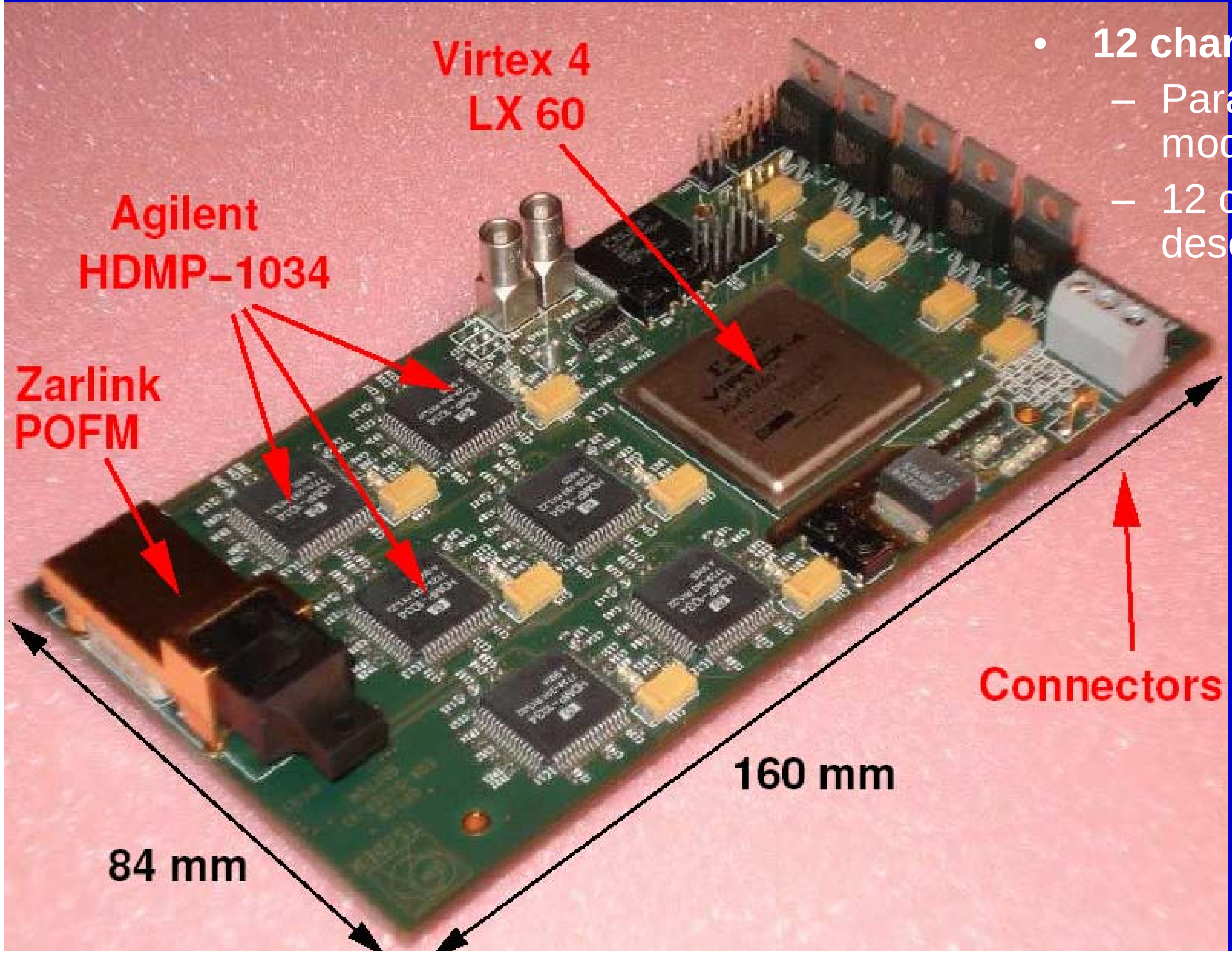
Muon track finder trigger

- Conclusion track finder:

- Data reduction
- Pipelining
- Feasibility study on possible algorithms
- Back annotation of Pins in FPGA after routing
- Full board – multiple FPGA VHDL simulation
- Stimulus files from (costumer) simulation
- Planning at FPGA level has impact on system implementation

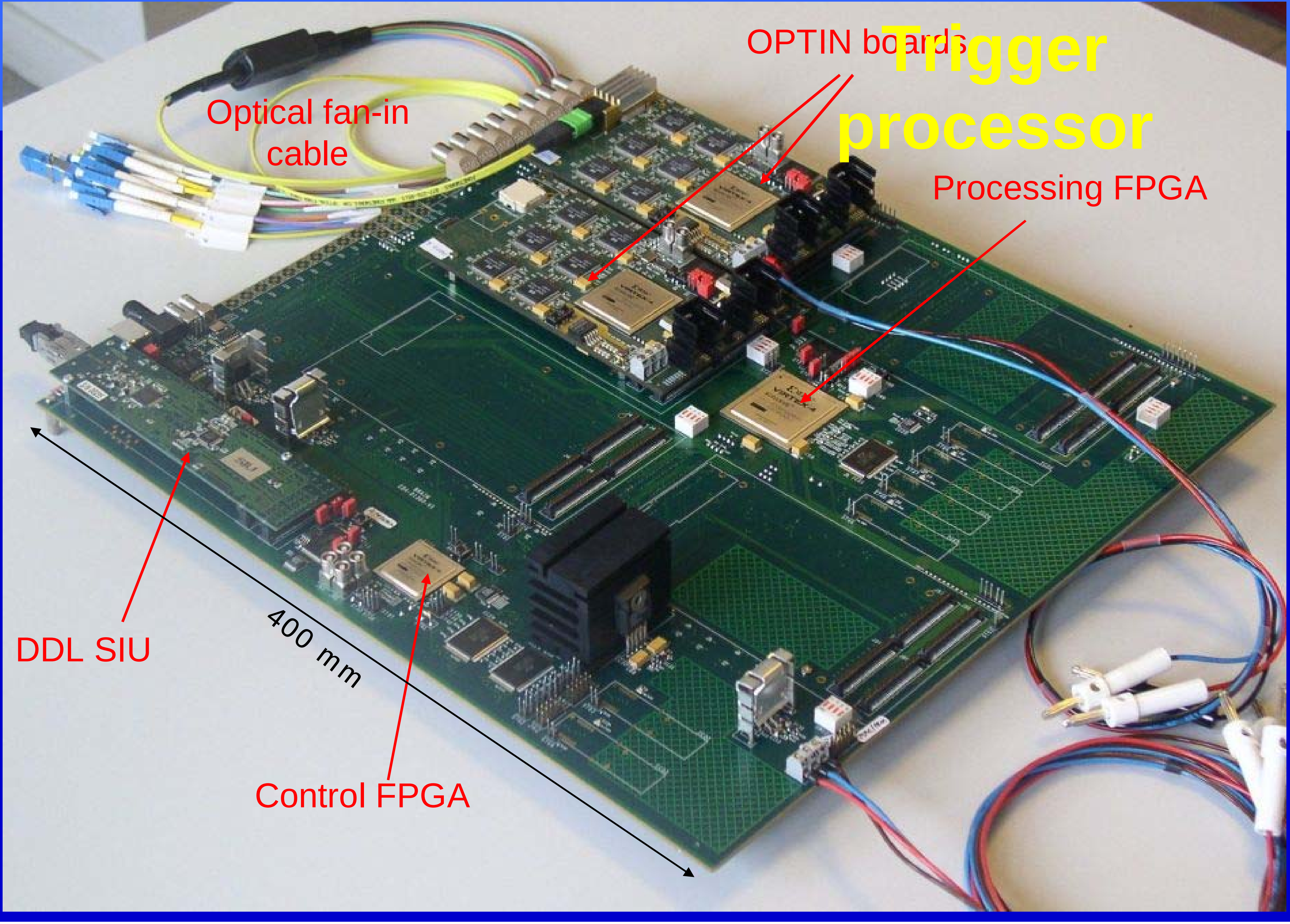
Example FPGA processors

Processor board with optical inputs

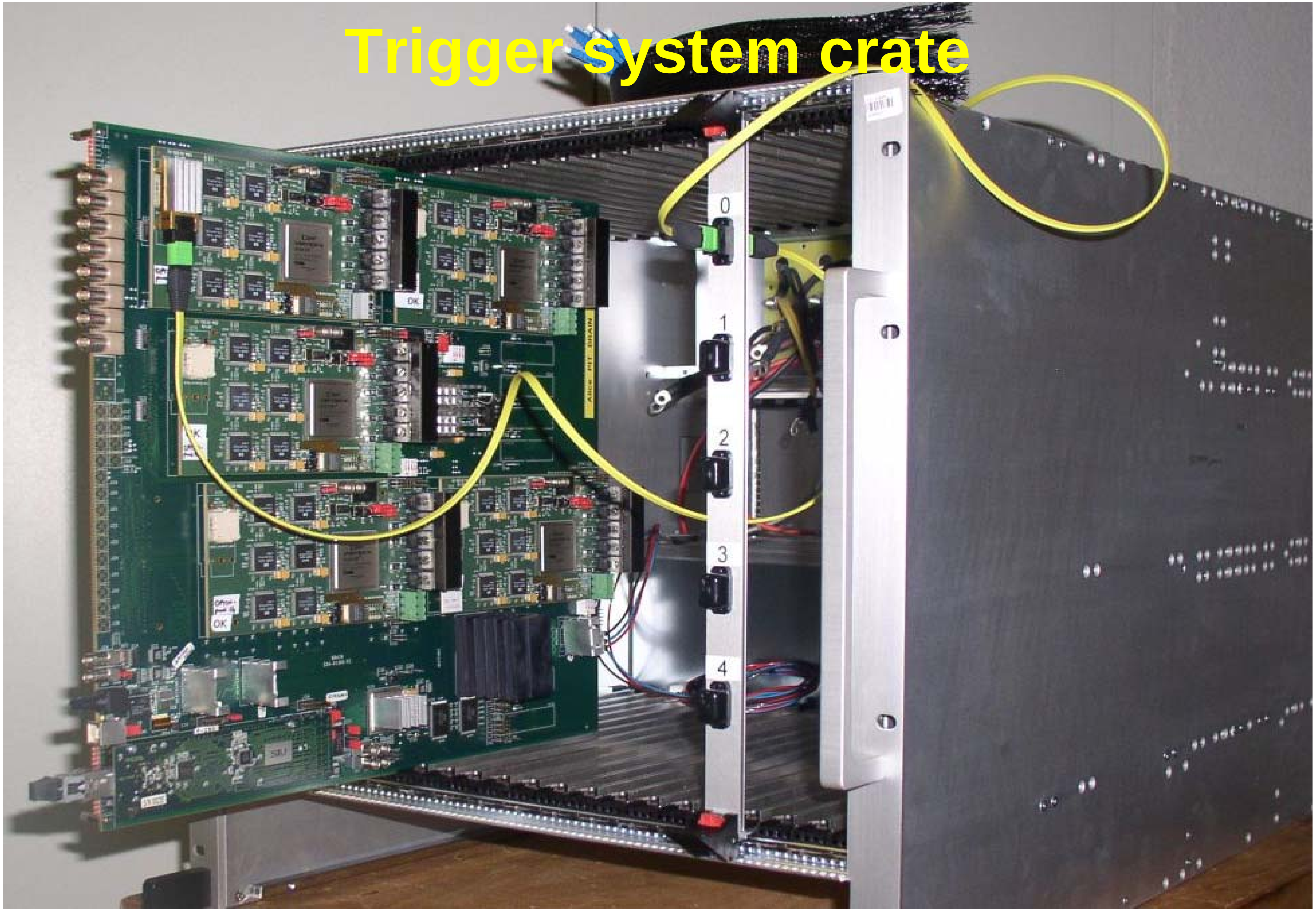


- **12 channels**
  - Parallel optical receiver module
  - 12 closely packed G-link deserializer ASICs

Trigger processor



Trigger system crate



Design cycle

Design cycle

- **System – Specifications**

- **Different approaches - possibilities**
  - ASICs, CAM -> FPGA
  - Pattern recognition / Analytical approach => Mixture

- **Simulation - Feasibility - Forecast to future technologies**

- **Data flow simulation/calculation**
  – buffer sizes
  – dead times

Design cycle

- **System – Specifications**

- **Different approaches - possibilities**
  - ASICs, CAM -> FPGA
  - Pattern recognition / Analytical approach => Mixture

- **Simulation - Feasibility - Forecast to future technologies**

- **Data flow simulation/calculation**
  – buffer sizes, dead times

Design cycle

- **Implementation scheme – propose technology independent architecture**
  - Do not push problems to a higher level - IO pins, PCB, system

- **Technology independent Simulation**
  - Full system: system input patterns – Qualification of data process
  - implement/integrate into system surrounding - work on FPGA code
  - Simulation together with environment
    - other FPGA
    - input data

Design cycle

- **Implementation - technology dependent**

- **Selection of components**
  - Performance, features, evaluation, availability
  - price, age/phase in product cycle,
    - if very new -> support and access to high quantity difficult -> close connection to distributor
- **Define strategy on Maintenance and upgrades**
  - FPGA might get too full & slow after implementation of more and more functions

Design cycle

- **FPGA simulation/synthesis/place&route/back-annotation**
- **Board Placing/routing**
  - FPGA -> board -> FPGA
  - FPGA Back annotation/Board level of pin position-Feedback on board layout
- **behavioral simulation of HDL code**
  - back annotated gate level after routing with board/system level
  - SEU simulation

Design cycle

- **Problems which are not solved on component level (ASIC/FPGA)**
  - are pushed to the system level,
    become expensive and time consuming

- **System level considerations ->**
  - System level simulation
  - Multi designer environment
  - Multi component environment

Design cycle

- **Example layout**
  - Prototype no internal design constraints on pin assignment for board layout -> 16 layer board ->
  - with assignment clean and 8 layer board
- **Missing board level simulation with two FPGAs**
  - simulation of each FPGA is OK
    together setup and hold time violations
    board delay (mealy/moore)
- **Evolution of FPGA technology:**
  - more than 1 FPGA with board routing ->
  - 1 FPGA no board routing

Design cycle

- **Software/Hardware development must go hand in hand**
- **Debugging features in FPGAs/system/history/status**
- **Remote control is often required**
  - how to implement
  - always one FPGA not reprogrammable as communication processor

Design cycle

- **Board production**
  - – JTAG boundary scan is mandatory for BGA
    - Full system JTAG especially with multiple FPGAs on board
    - reduces turn around time
    - gives proof of problems to manufacturer
    - X-ray test are not always conclusive (example not even copper on pads)
  - – Soldering problems with prototype series
  - – Test points

Design cycle

- **Define strategy on Reliability**
  - which date may be corrupted and which data must not be corrupted
  - radiation, SEU, cosmic rays on ground level
  - sub micron ASICs/FPGA

Starting to make an FPGA project

FPGA specifications

- **How to make an FPGA?**
  - What should it do?
  - How should it do it?

- **Systems / Requirements define detailed implementation scheme/architecture**

- **Specification need to be worked out before even one thinks about the FPGA type or code.**
  - Specification: understand user needs
  - define specification of system together with user/costumer

- **re-discuss, re-negotiate**
  - understand
  - task of designer to understand and translate specifications

FPGA specifications

- Costumer/boss says:
  “I need a system which can calculate the value each 100 ns.”

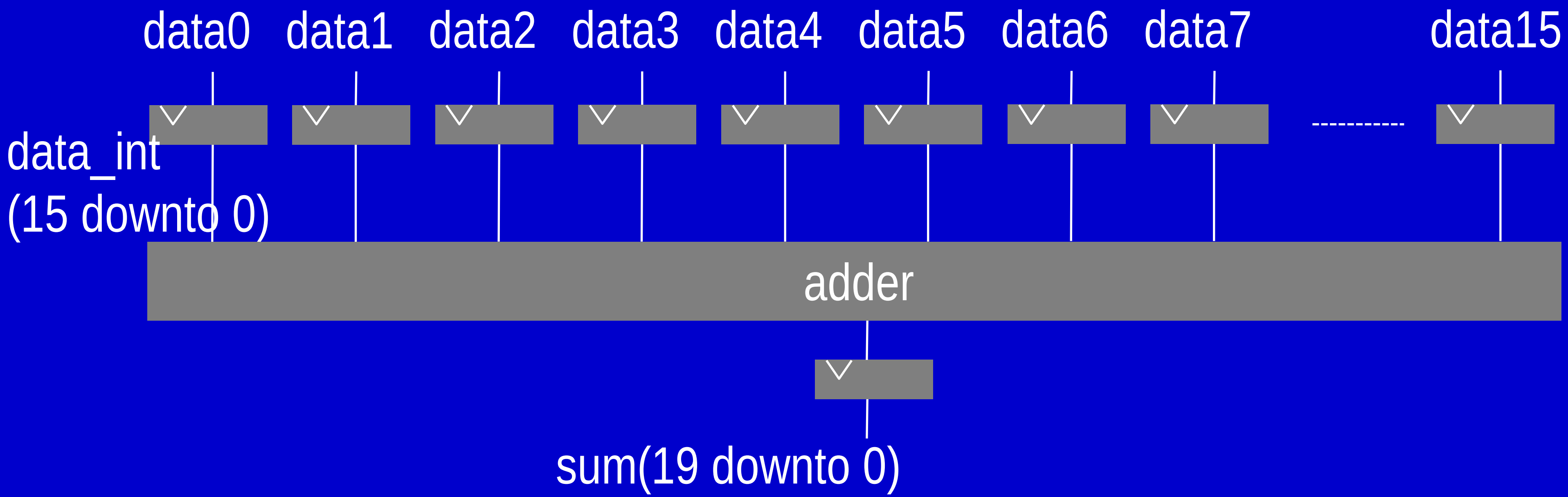
- What you might understand is:
  “The calculation needs to be finished within 100 ns”

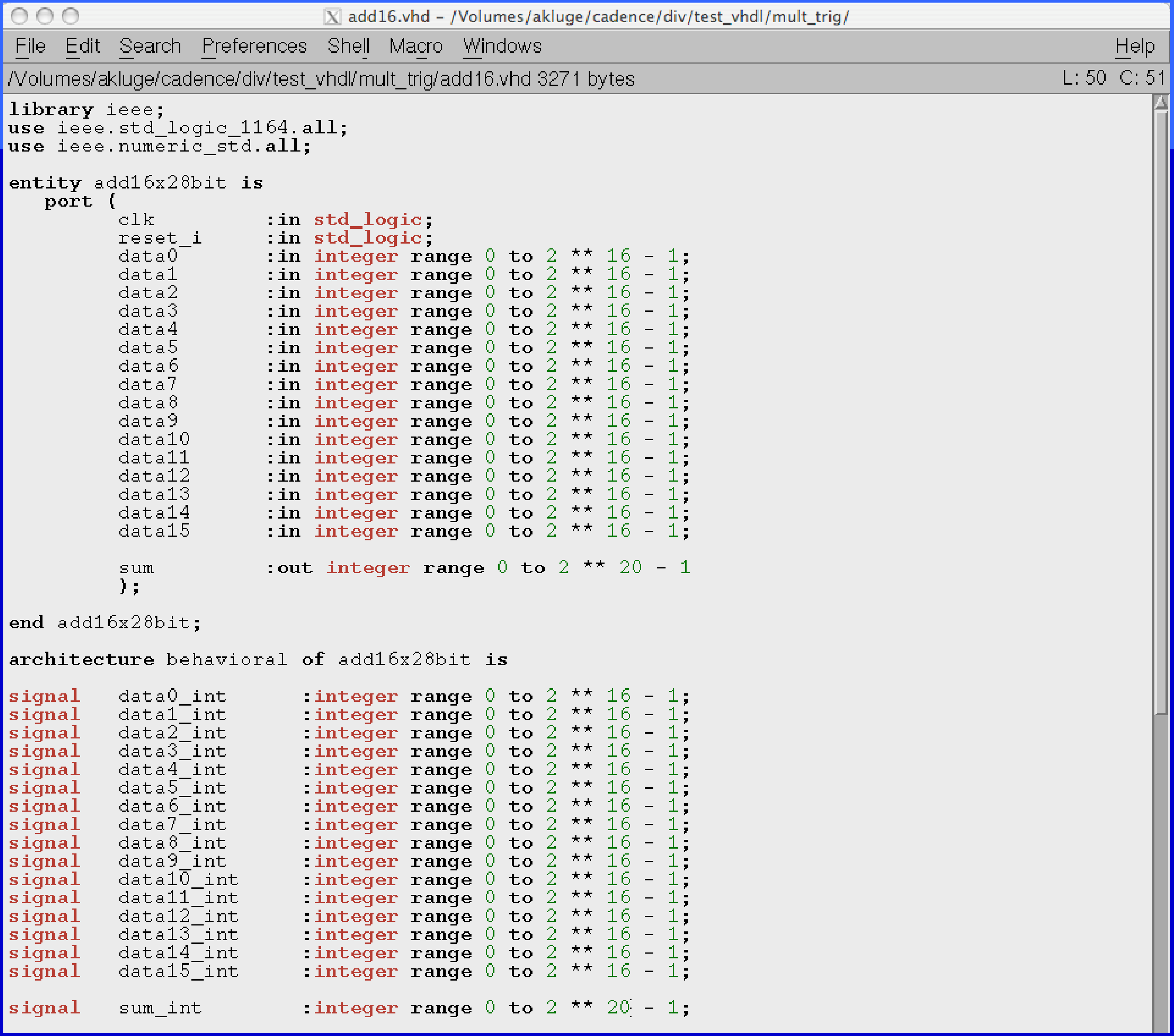
- What he means is:
  “A new value needs to be processed every 100 ns. How long it takes to present the result does not matter”

- First case: might be impossible, maybe not.
  Second case: Processors in parallel or in pipeline

Adder

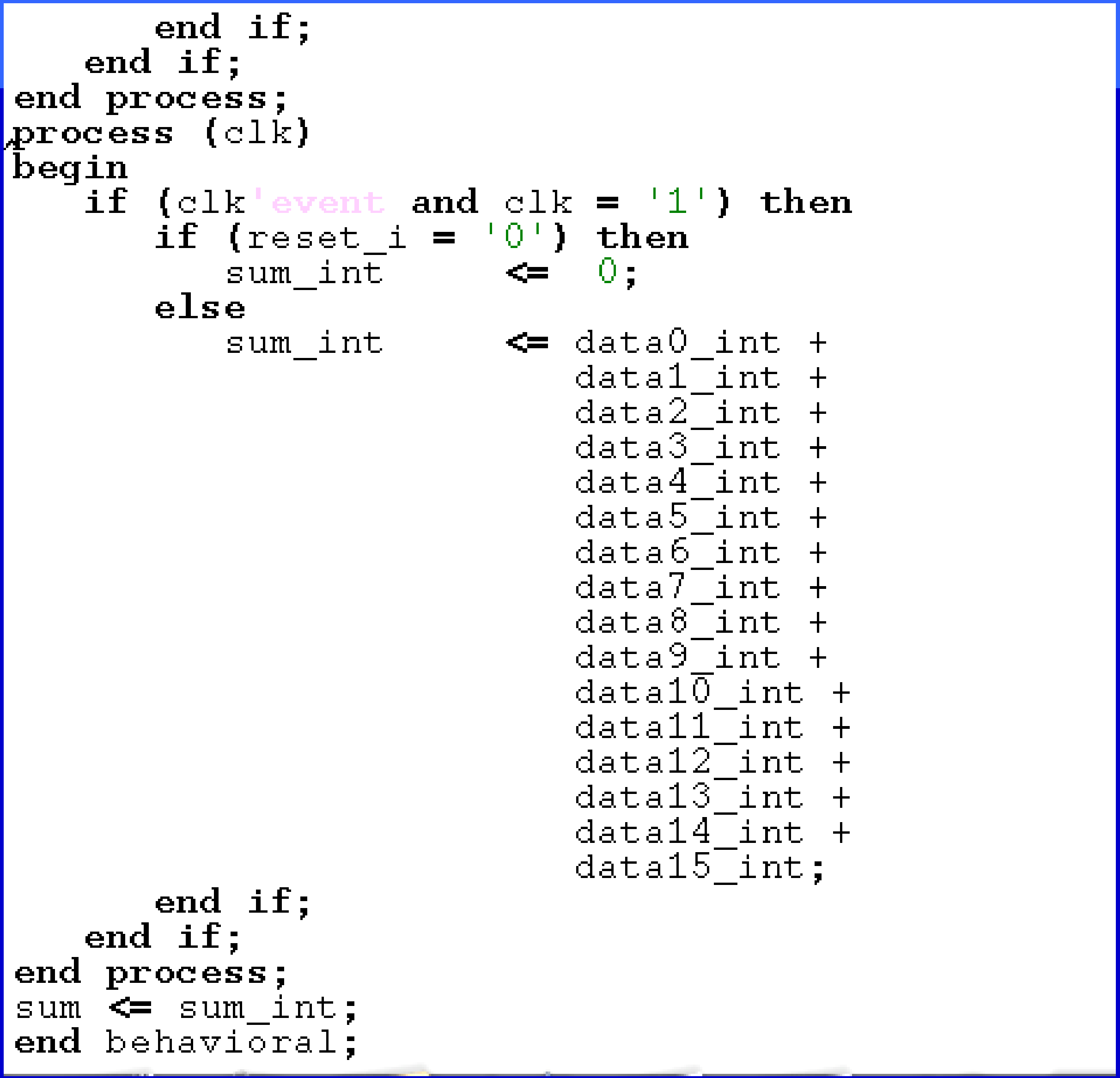
- **Example:**
  - add 16 16-bit values every 25 ns;




add16.vhd - /Volumes/akluge/cadence/div/test_vhdl/mult_trig/
File Edit Search Preferences Shell Macro Windows Help
/Volumes/akluge/cadence/div/test_vhdl/mult_trig/add16.vhd 3271 bytes
L: 50 C: 51

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

entity add16x28bit is
   port (
         clk            :in std_logic;
         reset_i        :in std_logic;
         data0          :in integer range 0 to 2 ** 16 - 1;
         data1          :in integer range 0 to 2 ** 16 - 1;
         data2          :in integer range 0 to 2 ** 16 - 1;
         data3          :in integer range 0 to 2 ** 16 - 1;
         data4          :in integer range 0 to 2 ** 16 - 1;
         data5          :in integer range 0 to 2 ** 16 - 1;
         data6          :in integer range 0 to 2 ** 16 - 1;
         data7          :in integer range 0 to 2 ** 16 - 1;
         data8          :in integer range 0 to 2 ** 16 - 1;
         data9          :in integer range 0 to 2 ** 16 - 1;
         data10         :in integer range 0 to 2 ** 16 - 1;
         data11         :in integer range 0 to 2 ** 16 - 1;
         data12         :in integer range 0 to 2 ** 16 - 1;
         data13         :in integer range 0 to 2 ** 16 - 1;
         data14         :in integer range 0 to 2 ** 16 - 1;
         data15         :in integer range 0 to 2 ** 16 - 1;

         sum            :out integer range 0 to 2 ** 20 - 1
         );

end add16x28bit;

architecture behavioral of add16x28bit is

signal   data0_int      :integer range 0 to 2 ** 16 - 1;
signal   data1_int      :integer range 0 to 2 ** 16 - 1;
signal   data2_int      :integer range 0 to 2 ** 16 - 1;
signal   data3_int      :integer range 0 to 2 ** 16 - 1;
signal   data4_int      :integer range 0 to 2 ** 16 - 1;
signal   data5_int      :integer range 0 to 2 ** 16 - 1;
signal   data6_int      :integer range 0 to 2 ** 16 - 1;
signal   data7_int      :integer range 0 to 2 ** 16 - 1;
signal   data8_int      :integer range 0 to 2 ** 16 - 1;
signal   data9_int      :integer range 0 to 2 ** 16 - 1;
signal   data10_int     :integer range 0 to 2 ** 16 - 1;
signal   data11_int     :integer range 0 to 2 ** 16 - 1;
signal   data12_int     :integer range 0 to 2 ** 16 - 1;
signal   data13_int     :integer range 0 to 2 ** 16 - 1;
signal   data14_int     :integer range 0 to 2 ** 16 - 1;
signal   data15_int     :integer range 0 to 2 ** 16 - 1;

signal   sum_int        :integer range 0 to 2 ** 20 - 1;
```

```
process (clk)
begin
    if (clk'event and clk = '1') then
        if (reset_i = '0') then
            data0_int   <= 0;
            data1_int   <= 0;
            data2_int   <= 0;
            data3_int   <= 0;
            data4_int   <= 0;
            data5_int   <= 0;
            data6_int   <= 0;
            data7_int   <= 0;
            data8_int   <= 0;
            data9_int   <= 0;
            data10_int  <= 0;
            data11_int  <= 0;
            data12_int  <= 0;
            data13_int  <= 0;
            data14_int  <= 0;
            data15_int  <= 0;
        else
            data0_int   <= data0;
            data1_int   <= data1;
            data2_int   <= data2;
            data3_int   <= data3;
            data4_int   <= data4;
            data5_int   <= data5;
            data6_int   <= data6;
            data7_int   <= data7;
            data8_int   <= data8;
            data9_int   <= data9;
            data10_int  <= data10;
            data11_int  <= data11;
            data12_int  <= data12;
            data13_int  <= data13;
            data14_int  <= data14;
            data15_int  <= data15;
        end if;
    end if;
end process;
process (clk)
begin
    if (clk'event and clk = '1') then
        if (reset_i = '0') then
            sum_int     <=  0;
        else
            sum_int
```



```vhdl
      end if;
   end if;
end process;
process (clk)
begin
   if (clk'event and clk = '1') then
      if (reset_i = '0') then
         sum_int     <=  0;
      else
         sum_int     <= data0_int +
                        data1_int +
                        data2_int +
                        data3_int +
                        data4_int +
                        data5_int +
                        data6_int +
                        data7_int +
                        data8_int +
                        data9_int +
                        data10_int +
                        data11_int +
                        data12_int +
                        data13_int +
                        data14_int +
                        data15_int;
      end if;
   end if;
end process;
sum <= sum_int;
end behavioral;
```

Adder

- 533 logic elements, 6%
- 278 pins, 74%
- 29.7 MHz => 33.6 ns

FPGA specifications

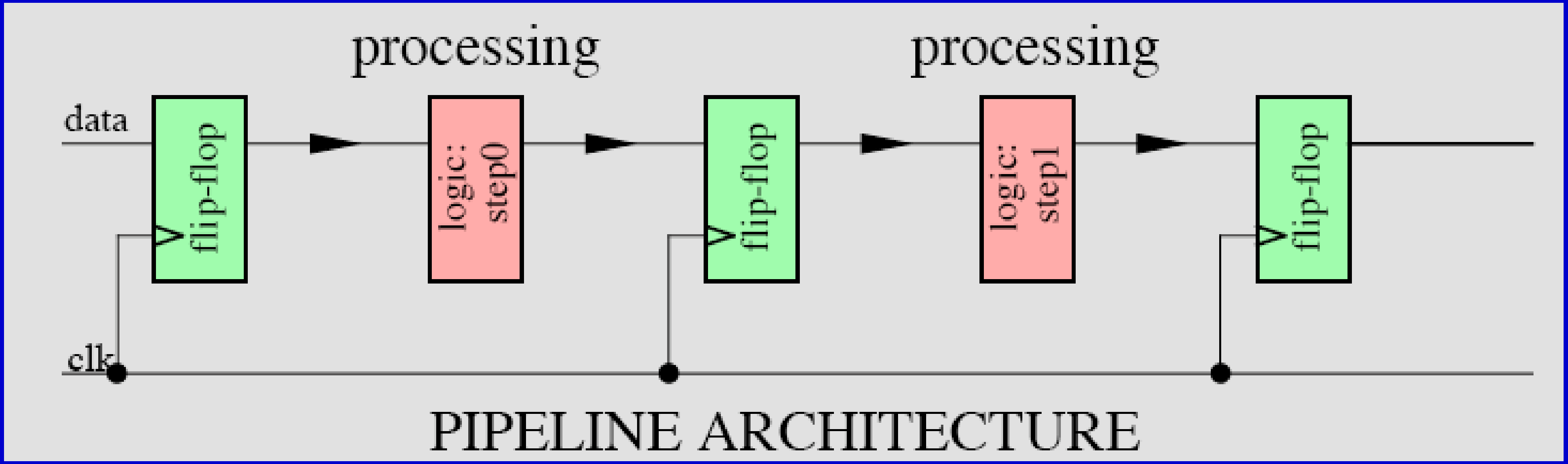
- Costumer/boss says:
  "I need a system which can calculate the value each 100 ns."

- What you might understand is:
  "The calculation needs to be finished within 100 ns"

- What he means is:
  "A new value needs to be processed every 100 ns. How long it takes to present the result does not matter"

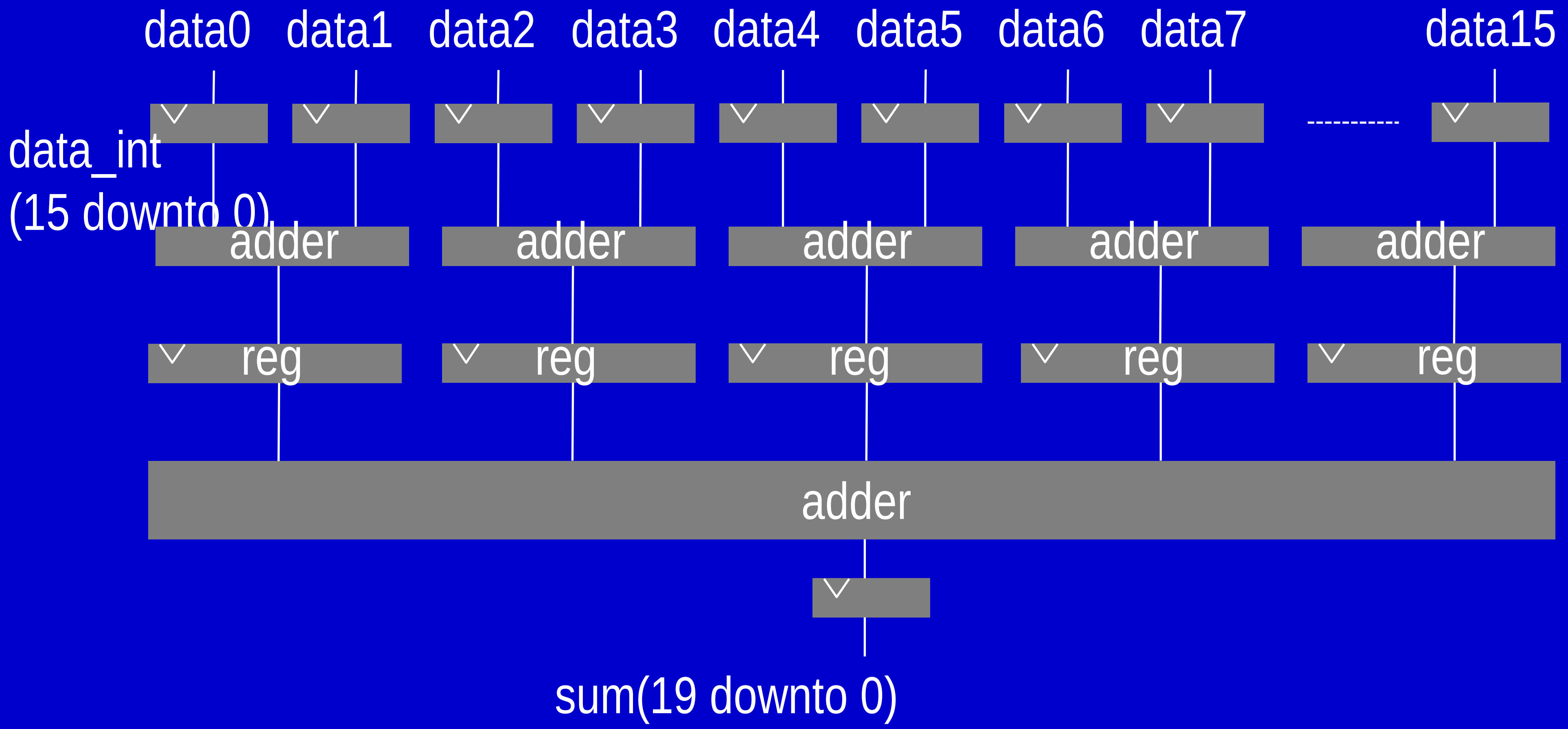
- First case: might be impossible, maybe not.
  Second case: Processors in parallel or in pipeline

Pipeline architecture



Adder with pipeline

- **Example:**
  - add 16 16-bit values within 25 ns;



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

entity add16Pipeline is
   port (
         clk          :in std_logic;
         reset_i      :in std_logic;
         data0        :in integer range 0 to 2 ** 16 - 1;
         data1        :in integer range 0 to 2 ** 16 - 1;
         data2        :in integer range 0 to 2 ** 16 - 1;
         data3        :in integer range 0 to 2 ** 16 - 1;
         data4        :in integer range 0 to 2 ** 16 - 1;
         data5        :in integer range 0 to 2 ** 16 - 1;
         data6        :in integer range 0 to 2 ** 16 - 1;
         data7        :in integer range 0 to 2 ** 16 - 1;
         data8        :in integer range 0 to 2 ** 16 - 1;
         data9        :in integer range 0 to 2 ** 16 - 1;
         data10       :in integer range 0 to 2 ** 16 - 1;
         data11       :in integer range 0 to 2 ** 16 - 1;
         data12       :in integer range 0 to 2 ** 16 - 1;
         data13       :in integer range 0 to 2 ** 16 - 1;
         data14       :in integer range 0 to 2 ** 16 - 1;
         data15       :in integer range 0 to 2 ** 16 - 1;

         sum          :out integer range 0 to 2 ** 20 - 1
         );

end add16Pipeline;

architecture behavioral of add16Pipeline is

signal   data0_int       :integer range 0 to 2 ** 16 - 1;
signal   data1_int       :integer range 0 to 2 ** 16 - 1;
signal   data2_int       :integer range 0 to 2 ** 16 - 1;
signal   data3_int       :integer range 0 to 2 ** 16 - 1;
signal   data4_int       :integer range 0 to 2 ** 16 - 1;
signal   data5_int       :integer range 0 to 2 ** 16 - 1;
signal   data6_int       :integer range 0 to 2 ** 16 - 1;
signal   data7_int       :integer range 0 to 2 ** 16 - 1;
signal   data8_int       :integer range 0 to 2 ** 16 - 1;
signal   data9_int       :integer range 0 to 2 ** 16 - 1;
signal   data10_int      :integer range 0 to 2 ** 16 - 1;
signal   data11_int      :integer range 0 to 2 ** 16 - 1;
signal   data12_int      :integer range 0 to 2 ** 16 - 1;
signal   data13_int      :integer range 0 to 2 ** 16 - 1;
signal   data14_int      :integer range 0 to 2 ** 16 - 1;
signal   data15_int      :integer range 0 to 2 ** 16 - 1;

signal   sum_int         :integer range 0 to 2 ** 20 - 1;
signal   sum_int0        :integer range 0 to 2 ** 17 - 1;
signal   sum_int1        :integer range 0 to 2 ** 17 - 1;
signal   sum_int2        :integer range 0 to 2 ** 17 - 1;
signal   sum_int3        :integer range 0 to 2 ** 17 - 1;
signal   sum_int4        :integer range 0 to 2 ** 17 - 1;
signal   sum_int5        :integer range 0 to 2 ** 17 - 1;
signal   sum_int6        :integer range 0 to 2 ** 17 - 1;
signal   sum_int7        :integer range 0 to 2 ** 17 - 1;

begin
```

```
process (clk)
begin
   if (clk'event and clk = '1') then
      if (reset_i = '0') then
         data0_int   <= 0;
         data1_int   <= 0;
         data2_int   <= 0;
         data3_int   <= 0;
         data4_int   <= 0;
         data5_int   <= 0;
         data6_int   <= 0;
         data7_int   <= 0;
         data8_int   <= 0;
         data9_int   <= 0;
         data10_int  <= 0;
         data11_int  <= 0;
         data12_int  <= 0;
         data13_int  <= 0;
         data14_int  <= 0;
         data15_int  <= 0;

      else
         data0_int   <= data0;
         data1_int   <= data1;
         data2_int   <= data2;
         data3_int   <= data3;
         data4_int   <= data4;
         data5_int   <= data5;
         data6_int   <= data6;
         data7_int   <= data7;
         data8_int   <= data8;
         data9_int   <= data9;
         data10_int  <= data10;
         data11_int  <= data11;
         data12_int  <= data12;
         data13_int  <= data13;
         data14_int  <= data14;
         data15_int  <= data15;
      end if;
   end if;
end process;
```

```
process (clk)
begin
   if (clk'event and clk = '1') then
      if (reset_i = '0') then
         sum_int0    <=  0;
      else
         sum_int0    <= data0_int +
                        data1_int;
      end if;
   end if;
end process;

process (clk)
begin
   if (clk'event and clk = '1') then
      if (reset_i = '0') then
         sum_int1    <=  0;
      else
         sum_int1    <= data2_int +
                        data3_int;
      end if;
   end if;
end process;

process (clk)
begin
   if (clk'event and clk = '1') then
      if (reset_i = '0') then
         sum_int2    <=  0;
      else
         sum_int2    <= data4_int +
                        data5_int;
      end if;
   end if;
```

```vhdl
process (clk)
begin
   if (clk'event and clk = '1') then
      if (reset_i = '0') then
         sum_int3     <=  0;
      else
         sum_int3     <= data6_int +
                         data7_int;
      end if;
   end if;
end process;

process (clk)
begin
   if (clk'event and clk = '1') then
      if (reset_i = '0') then
         sum_int4     <=  0;
      else
         sum_int4     <= data8_int +
                         data9_int;
      end if;
   end if;
end process;
```

```
process (clk)
begin
    if (clk'event and clk = '1') then
        if (reset_i = '0') then
            sum_int5     <=  0;
        else
            sum_int5     <= data10_int +
                            data11_int;
        end if;
    end if;
end process;

process (clk)
begin
    if (clk'event and clk = '1') then
        if (reset_i = '0') then
            sum_int6     <=  0;
        else
            sum_int6     <= data12_int +
                            data13_int;
        end if;
    end if;
end process;

process (clk)
begin
    if (clk'event and clk = '1') then
        if (reset_i = '0') then
            sum_int7     <=  0;
        else
            sum_int7     <= data14_int +
                            data15_int;
        end if;
    end if;
end process;
```

```
process (clk)
begin
   if (clk'event and clk = '1') then
      if (reset_i = '0') then
         sum_int     <=  0;
      else
         sum_int     <= sum_int0 +
                        sum_int1 +
                        sum_int2 +
                        sum_int3 +
                        sum_int4 +
                        sum_int5 +
                        sum_int6 +
                        sum_int7
                        ;
      end if;
   end if;
end process;


sum <= sum_int;

end behavioral;
```

Adder with pipeline

- Adder without pipeline
- 533 logic elements, 6%
- 278 pins, 74%
- 29.7 MHz => 33.6 ns

- Adder with pipeline
- 526 logic elements, 6%
- 278 pins, 74%
- 45.4 MHz => 22 ns

FPGA specifications

- **re-discuss, re-negotiate**
  - – understand
  - – task of designer to understand and translate specifications

Readout Processors

Read-out processors

- **Specification**
  - Challenge - many parallel inputs – 25 ns intervall - short processing time
  - Storage during trigger decision time
  - Data reduction/encoding (zero suppression)
  - pipelining, buffering (FIFO, dual port RAM)

Pixel detector

1 sensor

1 sensor

10 readout chips

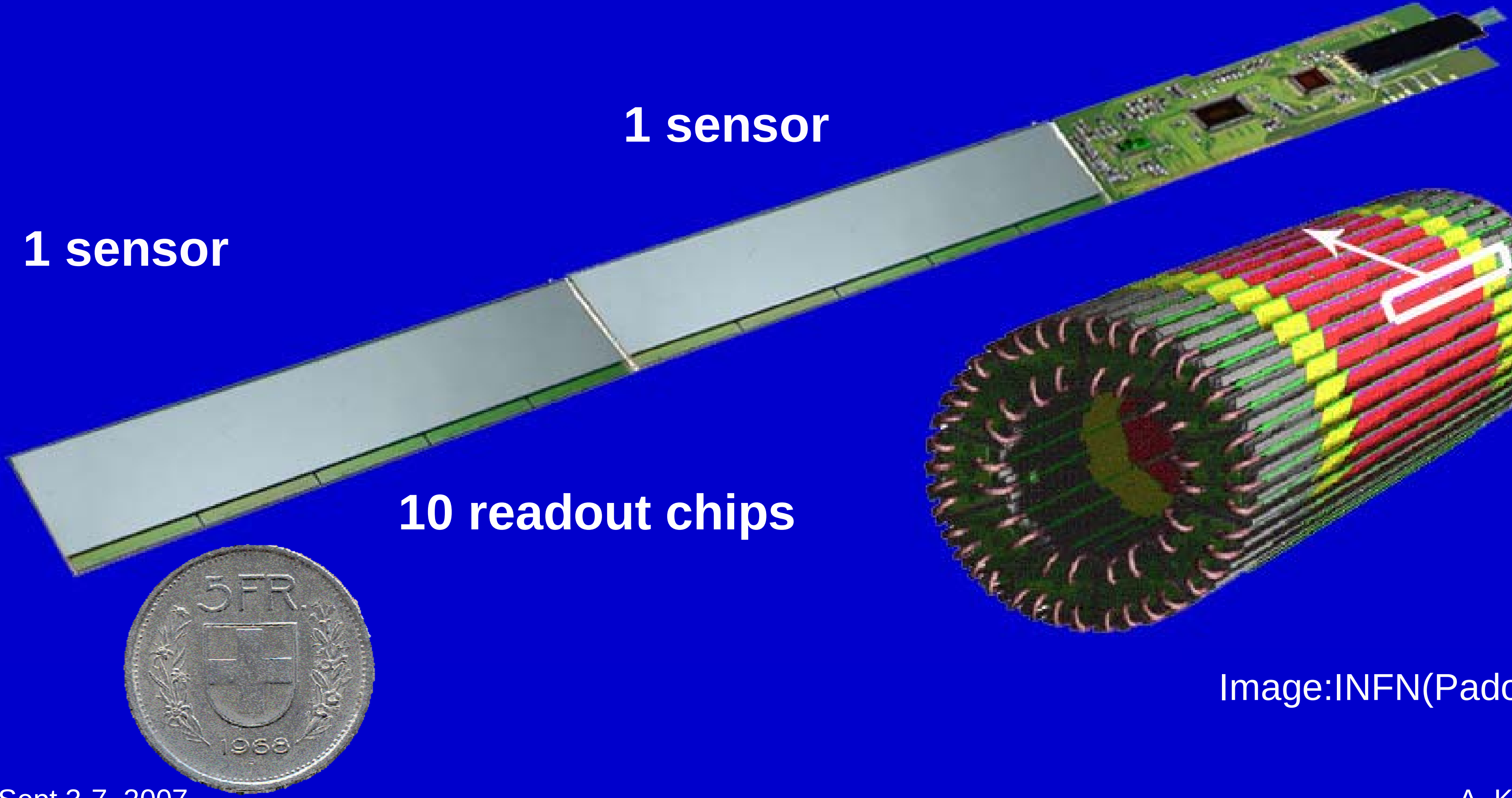


Image:INFN(Padova)

Pixel detector

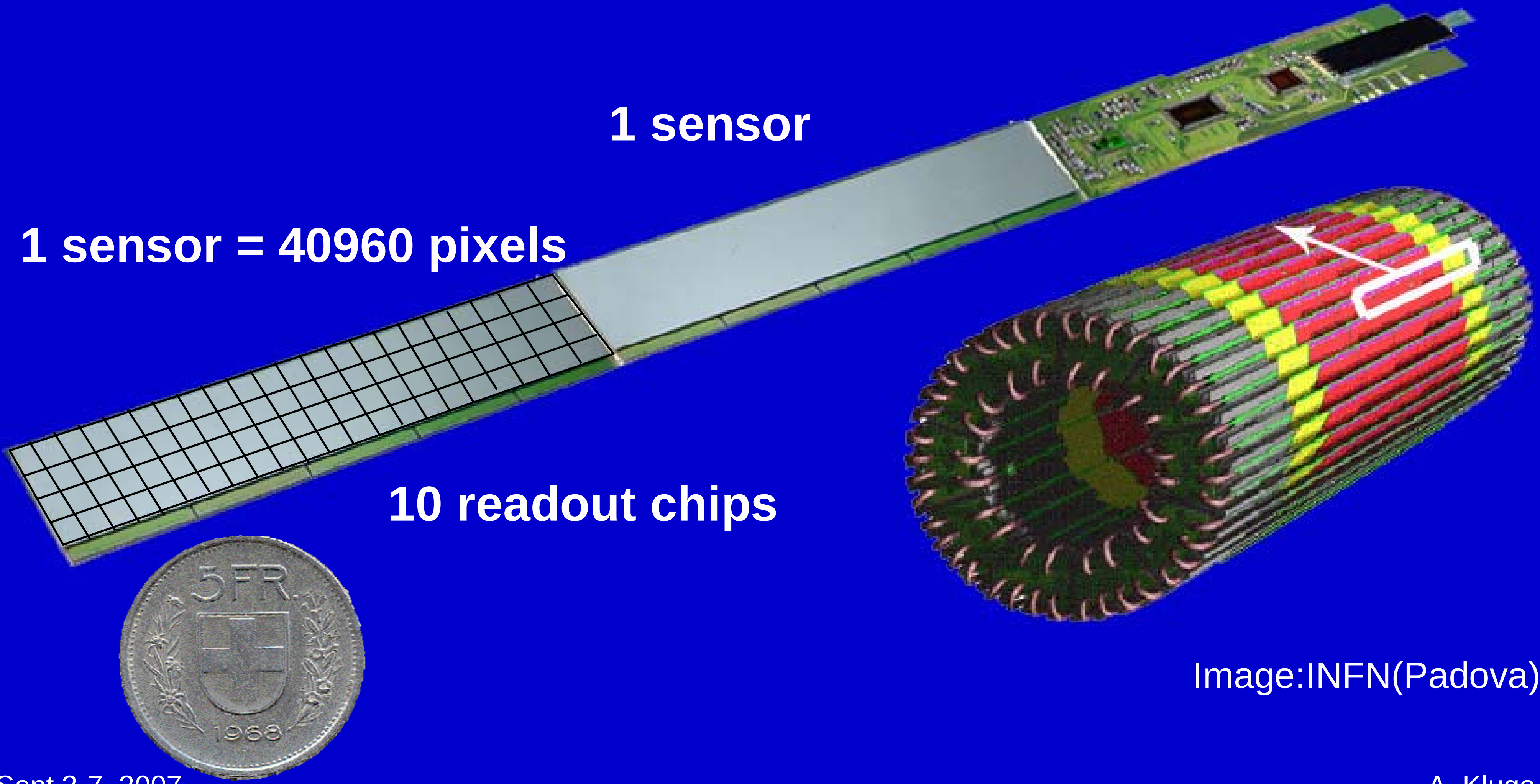
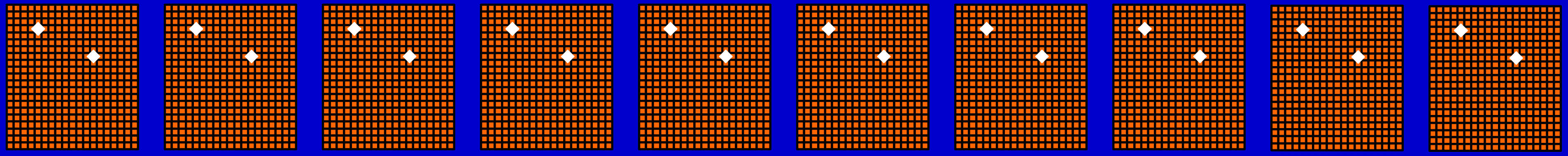


Image:INFN(Padova)

Pixel detector

full detector 120 x 2560 x 32 bits @ 10 MHz (100 ns) = ~100 Gbit/s

separate read-out for each detector module



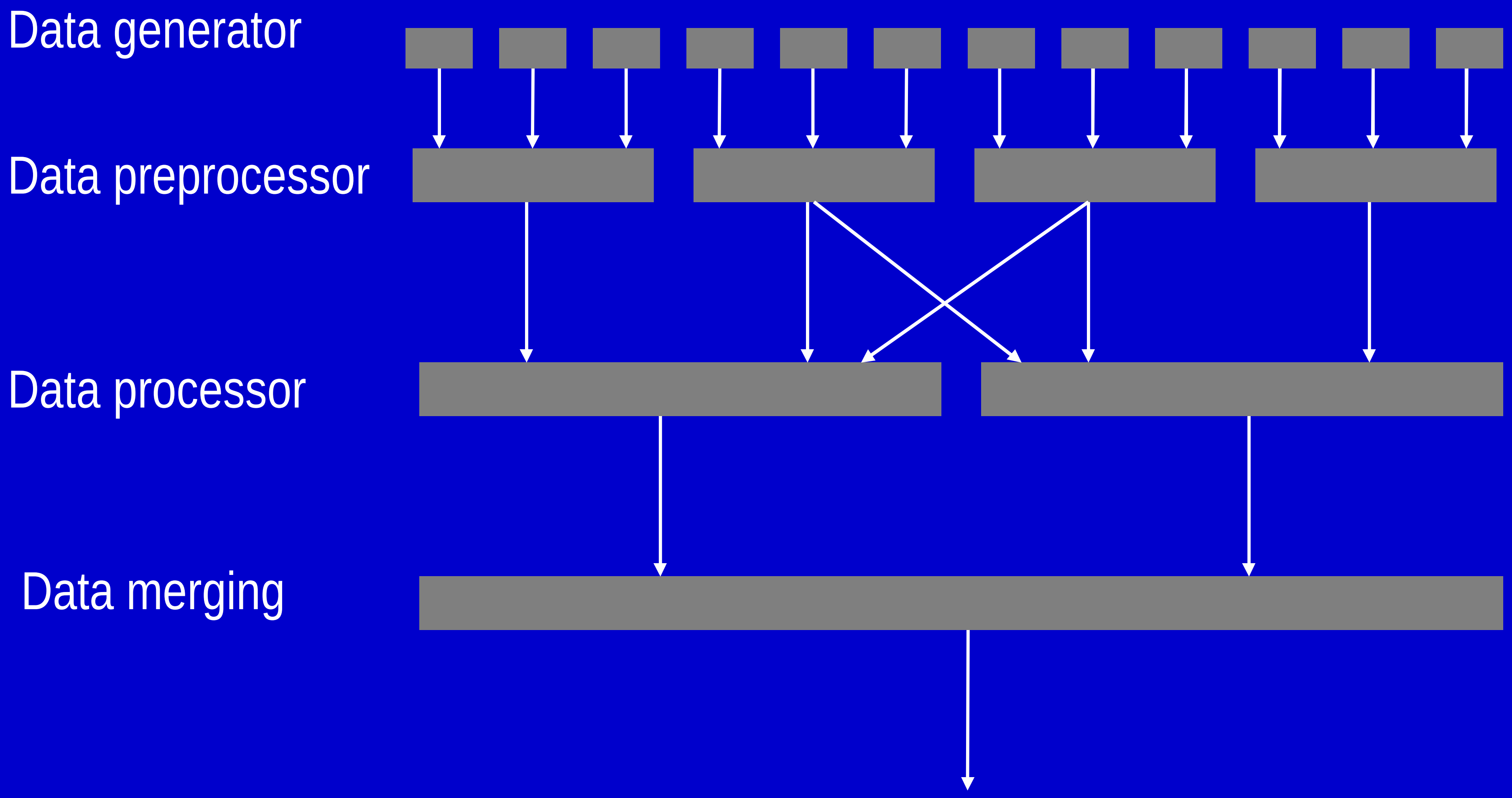
each detector module (10 chips)2560 x 32 bits @ 10 MHz

0000100000000000000

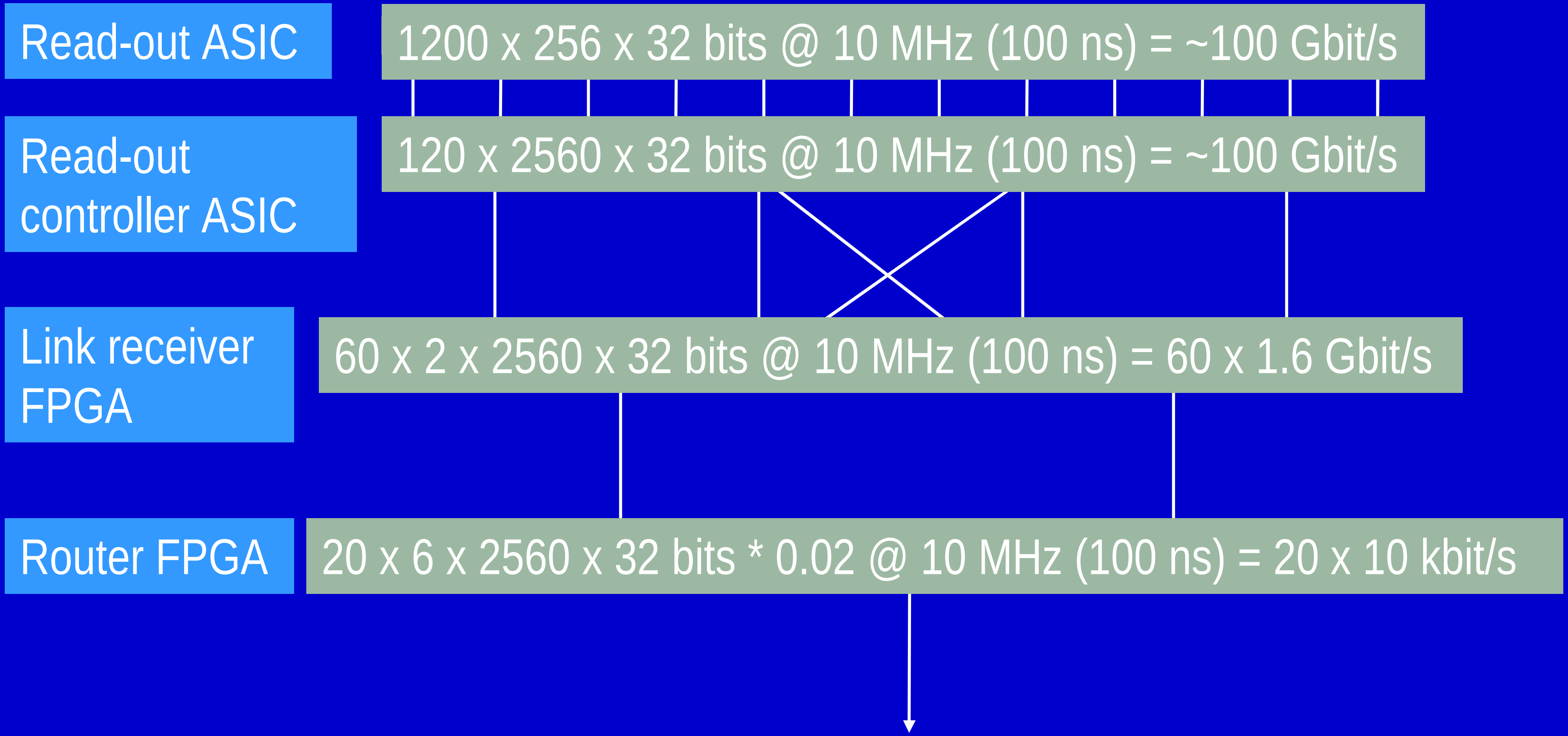
0000000000000000000

0000000000000100000

Data funnel


Data generator
Data preprocessor
Data processor
Data merging

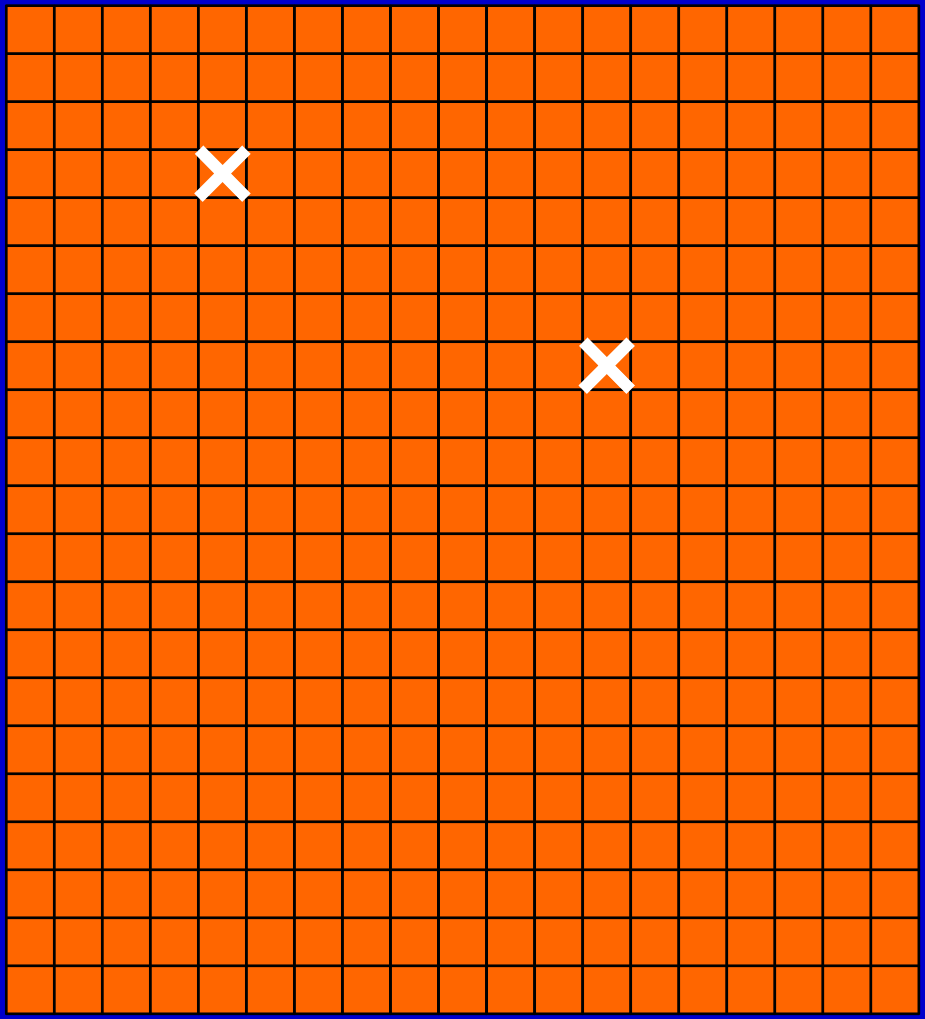
Data funnel


Read-out ASIC
1200 x 256 x 32 bits @ 10 MHz (100 ns) = ~100 Gbit/s
Read-out controller ASIC
120 x 2560 x 32 bits @ 10 MHz (100 ns) = ~100 Gbit/s
Link receiver FPGA
60 x 2 x 2560 x 32 bits @ 10 MHz (100 ns) = 60 x 1.6 Gbit/s
Router FPGA
20 x 6 x 2560 x 32 bits * 0.02 @ 10 MHz (100 ns) = 20 x 10 kbit/s

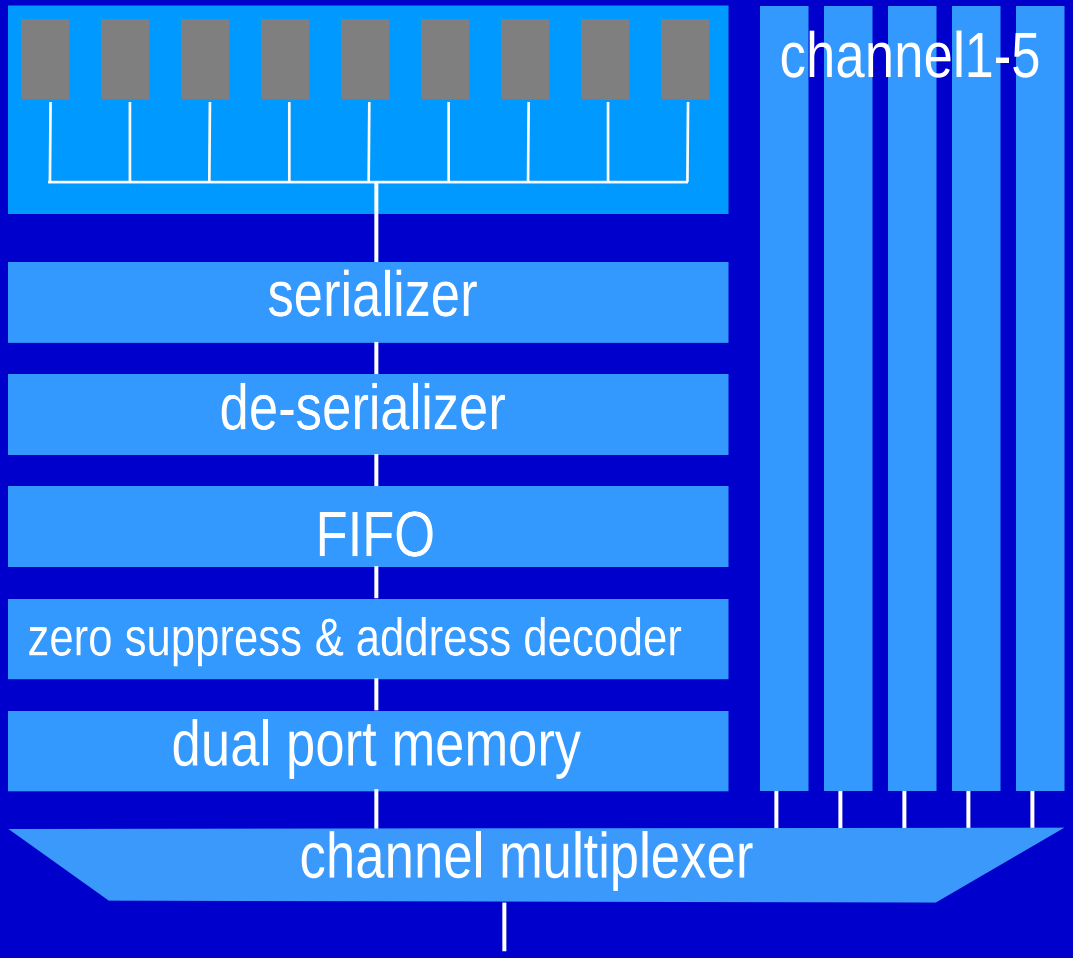
Pixel detector

Data generator
2560 x 32 bits

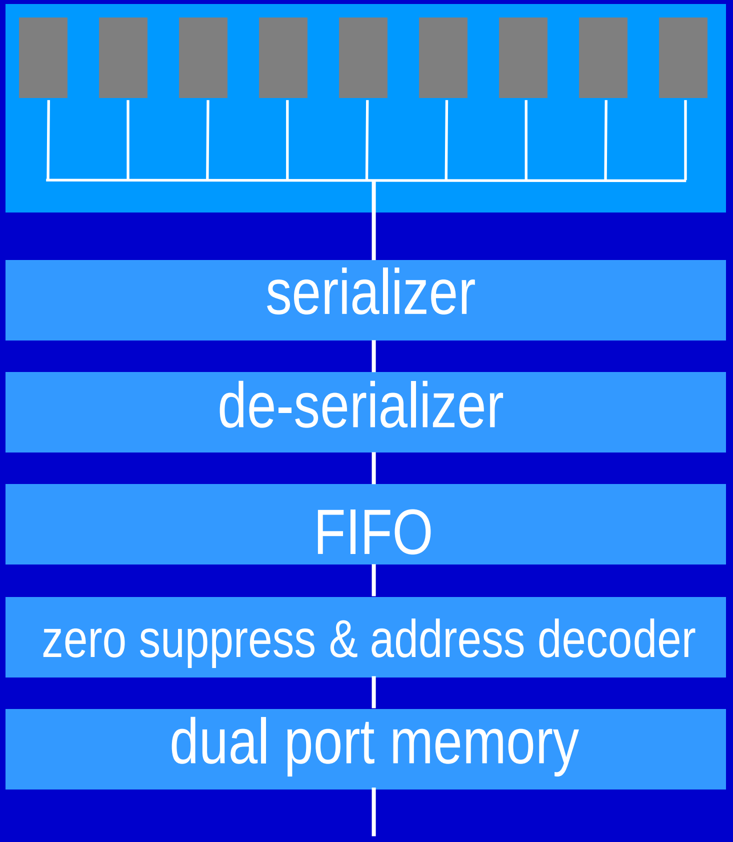
0000100000000000000
0000000000000000000
0000000000000100000



Pixel detector



Pixel detector



Pixel detector data processing

0	0	0	0	0	0	0	0	0	0	0	0	0

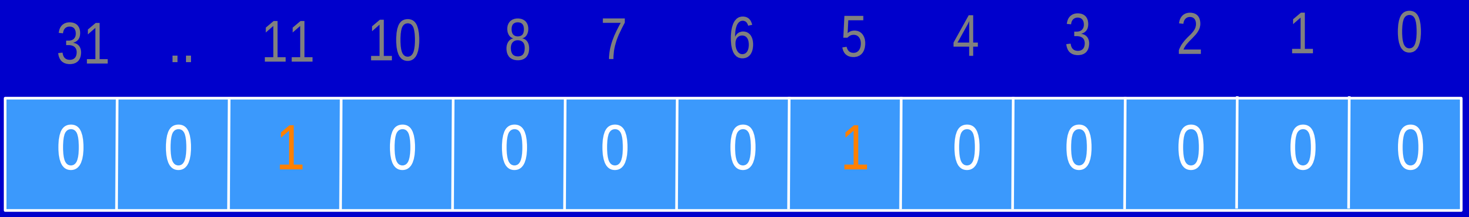
check if any hits

if no hits -> load new value from FIFO if available

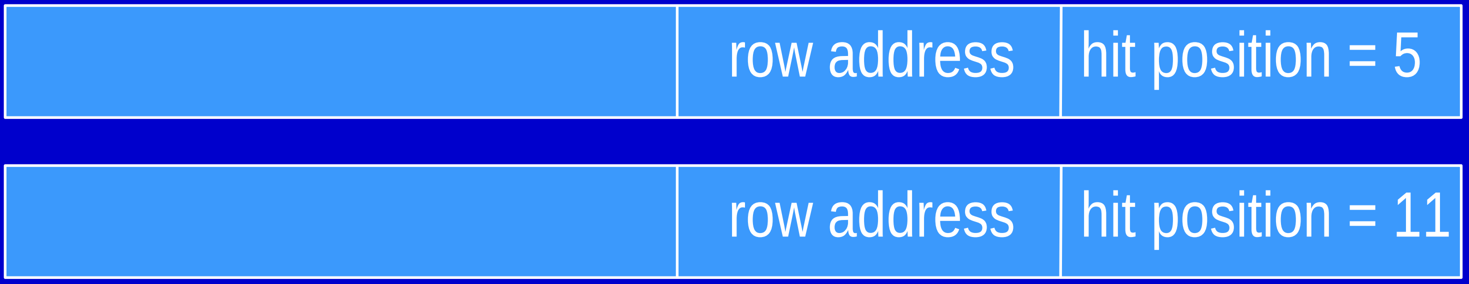
if 1 only -> decode the hit & request new value from FIFO

if more than one hit -> decode the hits

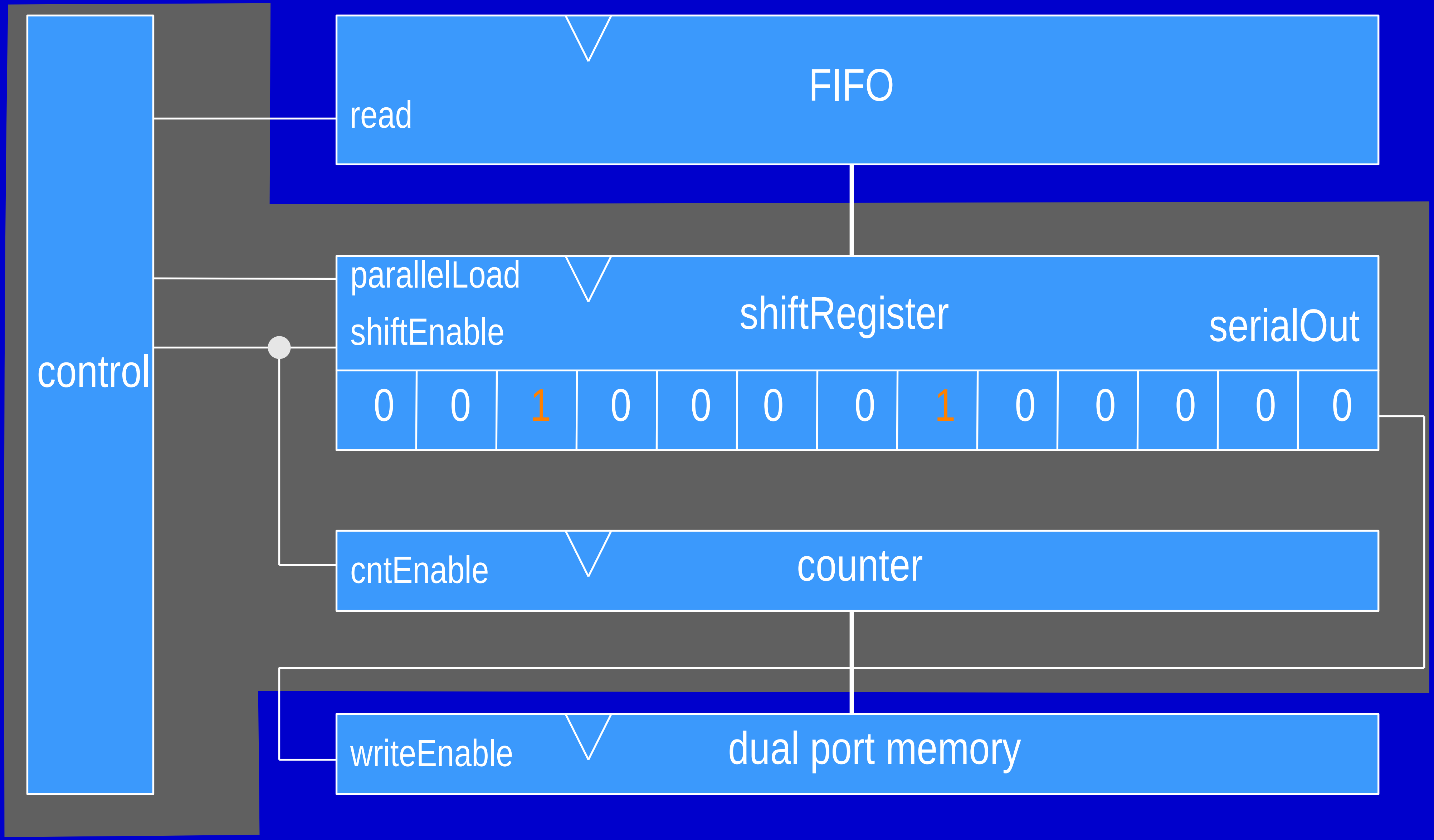
Pixel detector data processing



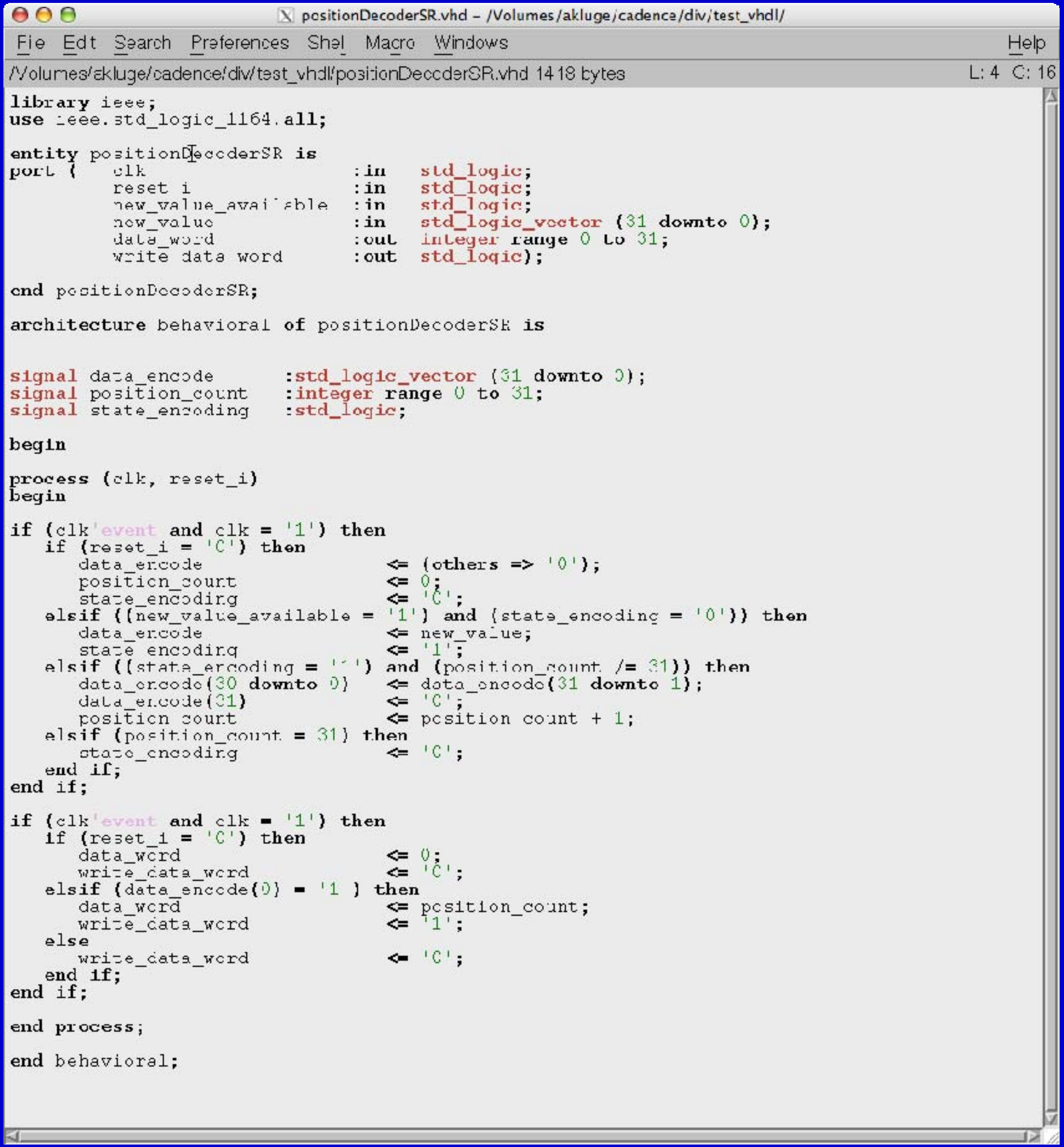
How to decode the address:
this line has two hits
the state machine must send two hits into the dual port memory



Pixel detector data processing

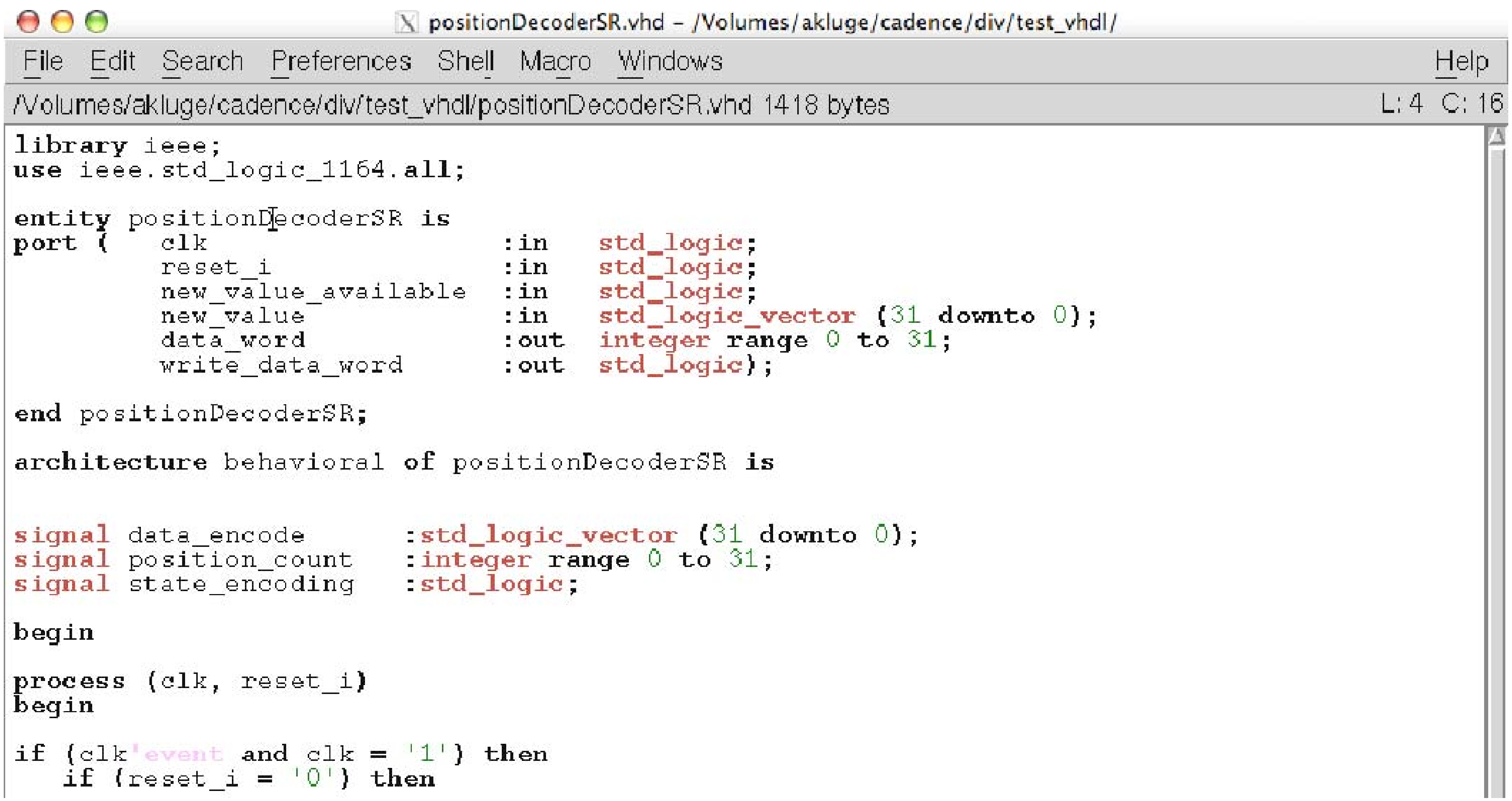


Position decoder – shift register


positionDecoderSR.vhd – /Volumes/akluge/cadence/div/test_vhdl/
File Edit Search Preferences Shell Macro Windows Help
/Volumes/akluge/cadence/div/test_vhdl/positionDecoderSR.vhd 1418 bytes L: 4 C: 16

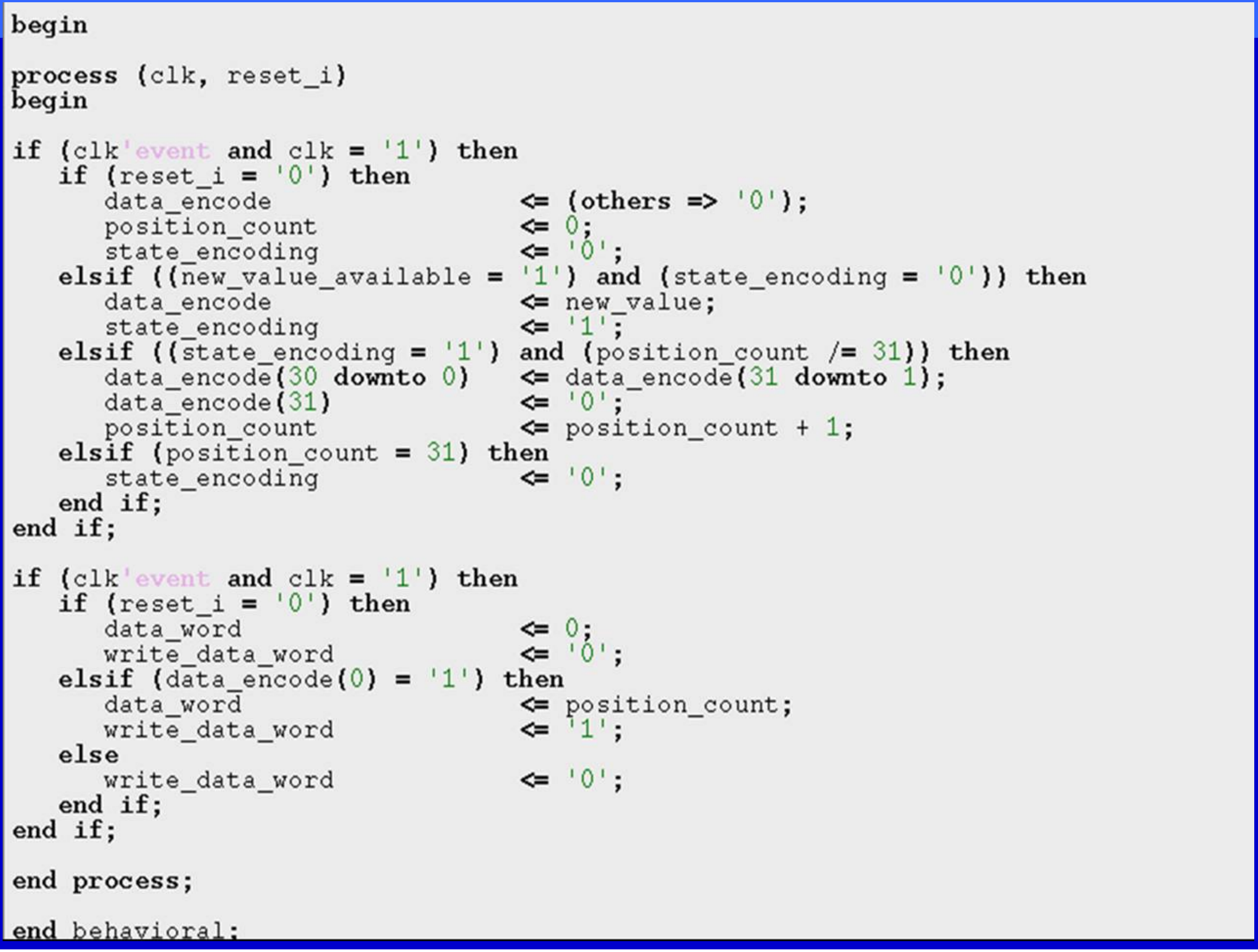
```
library ieee;
use ieee.std_logic_1164.all;

entity positionDecoderSR is
port (      clk                     :in   std_logic;
            reset_i                 :in   std_logic;
            new_value_available     :in   std_logic;
            new_value               :in   std_logic_vector (31 downto 0);
            data_word               :out  integer range 0 to 31;
            write_data_word         :out  std_logic);

end positionDecoderSR;

architecture behavioral of positionDecoderSR is


signal data_encode        :std_logic_vector (31 downto 0);
signal position_count     :integer range 0 to 31;
signal state_encoding     :std_logic;

begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_encode                  <= (others => '0');
      position_count               <= 0;
      state_encoding               <= '0';
   elsif ((new_value_available = '1') and (state_encoding = '0')) then
      data_encode                  <= new_value;
      state_encoding               <= '1';
   elsif ((state_encoding = '1') and (position_count /= 31)) then
      data_encode(30 downto 0)     <= data_encode(31 downto 1);
      data_encode(31)              <= '0';
      position_count               <= position_count + 1;
   elsif (position_count = 31) then
      state_encoding               <= '0';
   end if;
end if;

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_word                    <= 0;
      write_data_word              <= '0';
   elsif (data_encode(0) = '1') then
      data_word                    <= position_count;
      write_data_word              <= '1';
   else
      write_data_word              <= '0';
   end if;
end if;

end process;

end behavioral;
```

Position decoder – shift register


positionDecoderSR.vhd - /Volumes/akluge/cadence/div/test_vhdl/
File Edit Search Preferences Shell Macro Windows Help
/Volumes/akluge/cadence/div/test_vhdl/positionDecoderSR.vhd 1418 bytes
L: 4 C: 16

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

entity positionDecoderSR is
port (    clk                     :in    std_logic;
          reset_i                 :in    std_logic;
          new_value_available     :in    std_logic;
          new_value               :in    std_logic_vector (31 downto 0);
          data_word               :out   integer range 0 to 31;
          write_data_word         :out   std_logic);

end positionDecoderSR;

architecture behavioral of positionDecoderSR is


signal data_encode        :std_logic_vector (31 downto 0);
signal position_count     :integer range 0 to 31;
signal state_encoding     :std_logic;

begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
   if (reset_i = '0') then
```

Position decoder – shift register – VHDL code



```
begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_encode                  <= (others => '0');
      position_count               <= 0;
      state_encoding               <= '0';
   elsif ((new_value_available = '1') and (state_encoding = '0')) then
      data_encode                  <= new_value;
      state_encoding               <= '1';
   elsif ((state_encoding = '1') and (position_count /= 31)) then
      data_encode(30 downto 0)     <= data_encode(31 downto 1);
      data_encode(31)              <= '0';
      position_count               <= position_count + 1;
   elsif (position_count = 31) then
      state_encoding               <= '0';
   end if;
end if;

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_word                    <= 0;
      write_data_word              <= '0';
   elsif (data_encode(0) = '1') then
      data_word                    <= position_count;
      write_data_word              <= '1';
   else
      write_data_word              <= '0';
   end if;
end if;

end process;

end behavioral;
```

```
begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_encode                  <= (others => '0');
      position_count               <= 0;
      state_encoding               <= '0';
   elsif ((new_value_available = '1') and (state_encoding = '0')) then
      data_encode                  <= new_value;
      state_encoding               <= '1';
   elsif ((state_encoding = '1') and (position_count /= 31)) then
      data_encode(30 downto 0)     <= data_encode(31 downto 1);
      data_encode(31)              <= '0';
      position_count               <= position_count + 1;
   elsif (position_count = 31) then
      state_encoding               <= '0';
   end if;
end if;

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_word                    <= 0;
      write_data_word              <= '0';
   elsif (data_encode(0) = '1') then
      data_word                    <= position_count;
      write_data_word              <= '1';
   else
      write_data_word              <= '0';
   end if;
end if;

end process;

end behavioral;
```

shiftRegister

```
begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_encode                  <= (others => '0');
      position_count               <= 0;
      state_encoding               <= '0';
   elsif ((new_value_available = '1') and (state_encoding = '0')) then
      data_encode                  <= new_value;
      state_encoding               <= '1';
   elsif ((state_encoding = '1') and (position_count /= 31)) then
      data_encode(30 downto 0)     <= data_encode(31 downto 1);
      data_encode(31)              <= '0';
      position_count               <= position_count + 1;
   elsif (position_count = 31) then
      state_encoding               <= '0';
   end if;
end if;

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_word                    <= 0;
      write_data_word              <= '0';
   elsif (data_encode(0) = '1') then
      data_word                    <= position_count;
      write_data_word              <= '1';
   else
      write_data_word              <= '0';
   end if;
end if;

end process;

end behavioral;
```

counter

```
begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_encode                <= (others => '0');
      position_count             <= 0;
      state_encoding             <= '0';
   elsif ((new_value_available = '1') and (state_encoding = '0')) then
      data_encode                <= new_value;
      state_encoding             <= '1';
   elsif ((state_encoding = '1') and (position_count /= 31)) then
      data_encode(30 downto 0)   <= data_encode(31 downto 1);
      data_encode(31)            <= '0';
      position_count             <= position_count + 1;
   elsif (position_count = 31) then
      state_encoding             <= '0';
   end if;
end if;

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_word                  <= 0;
      write_data_word            <= '0';
   elsif (data_encode(0) = '1') then
      data_word                  <= position_count;
      write_data_word            <= '1';
   else
      write_data_word            <= '0';
   end if;
end if;

end process;

end behavioral;
```

control

```
begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_encode                <= (others => '0');
      position_count             <= 0;
      state_encoding             <= '0';
   elsif ((new_value_available = '1') and (state_encoding = '0')) then
      data_encode                <= new_value;
      state_encoding             <= '1';
   elsif ((state_encoding = '1') and (position_count /= 31)) then
      data_encode(30 downto 0)   <= data_encode(31 downto 1);
      data_encode(31)            <= '0';
      position_count             <= position_count + 1;
   elsif (position_count = 31) then
      state_encoding             <= '0';
   end if;
end if;

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_word                  <= 0;
      write_data_word            <= '0';
   elsif (data_encode(0) = '1') then
      data_word                  <= position_count;
      write_data_word            <= '1';
   else
      write_data_word            <= '0';
   end if;
end if;

end process;

end behavioral;
```

elsif invokes priority encoder -> more logic

state machine with case statement

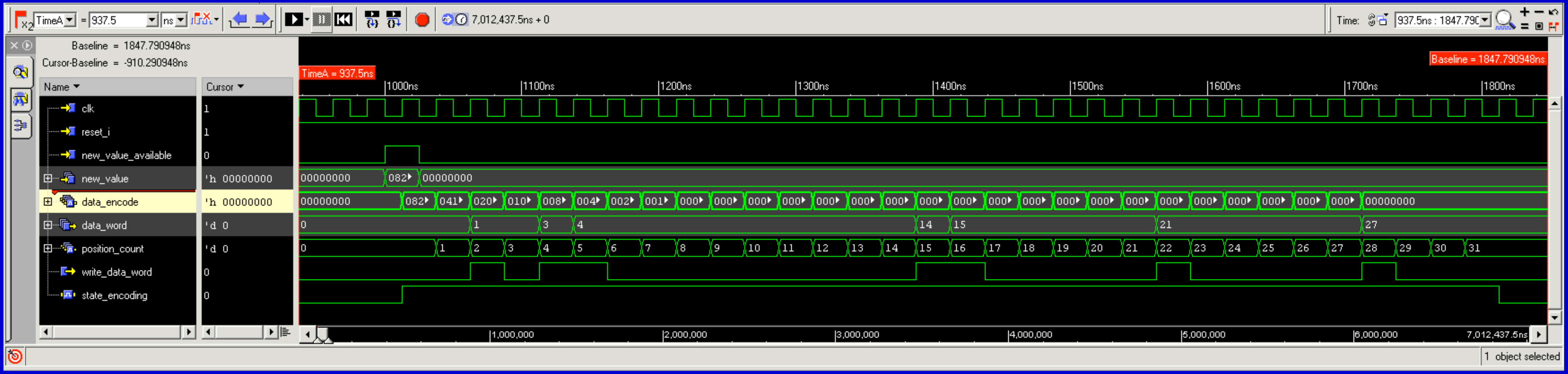
```
begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
    if (reset_i = '0') then
        data_encode                             <= (others => '0');
        position_count                          <= 0;
        state_encoding                          <= '0';
    else
        case (state_encoding) is
        when '0' =>
            if (new_value_available = '1') then
                data_encode                     <= new_value;
                state_encoding                  <= '1';
            end if;
        when '1' =>
            if (position_count /= 31) then
                data_encode(30 downto 0)        <= data_encode(31 downto 1);
                data_encode(31)                 <= '0';
                position_count                  <= position_count + 1;
            elsif (position_count = 31) then
                state_encoding                  <= '0';
            end if;
        when others =>
            data_encode                         <= (others => '0');
            position_count                      <= 0;
            state_encoding                      <= '0';
        end case;
    end if;
end if;

if (clk'event and clk = '1') then
    if (reset_i = '0') then
        data_word                               <= 0;
        write_data_word                         <= '0';
    elsif (data_encode(0) = '1') then
        data_word                               <= position_count;
        write_data_word                         <= '1';
    else
        write_data_word                         <= '0';
    end if;
end if;

end process;

end behavioral;
```

- Shift register is a parallel load register

Position decoder – shift register

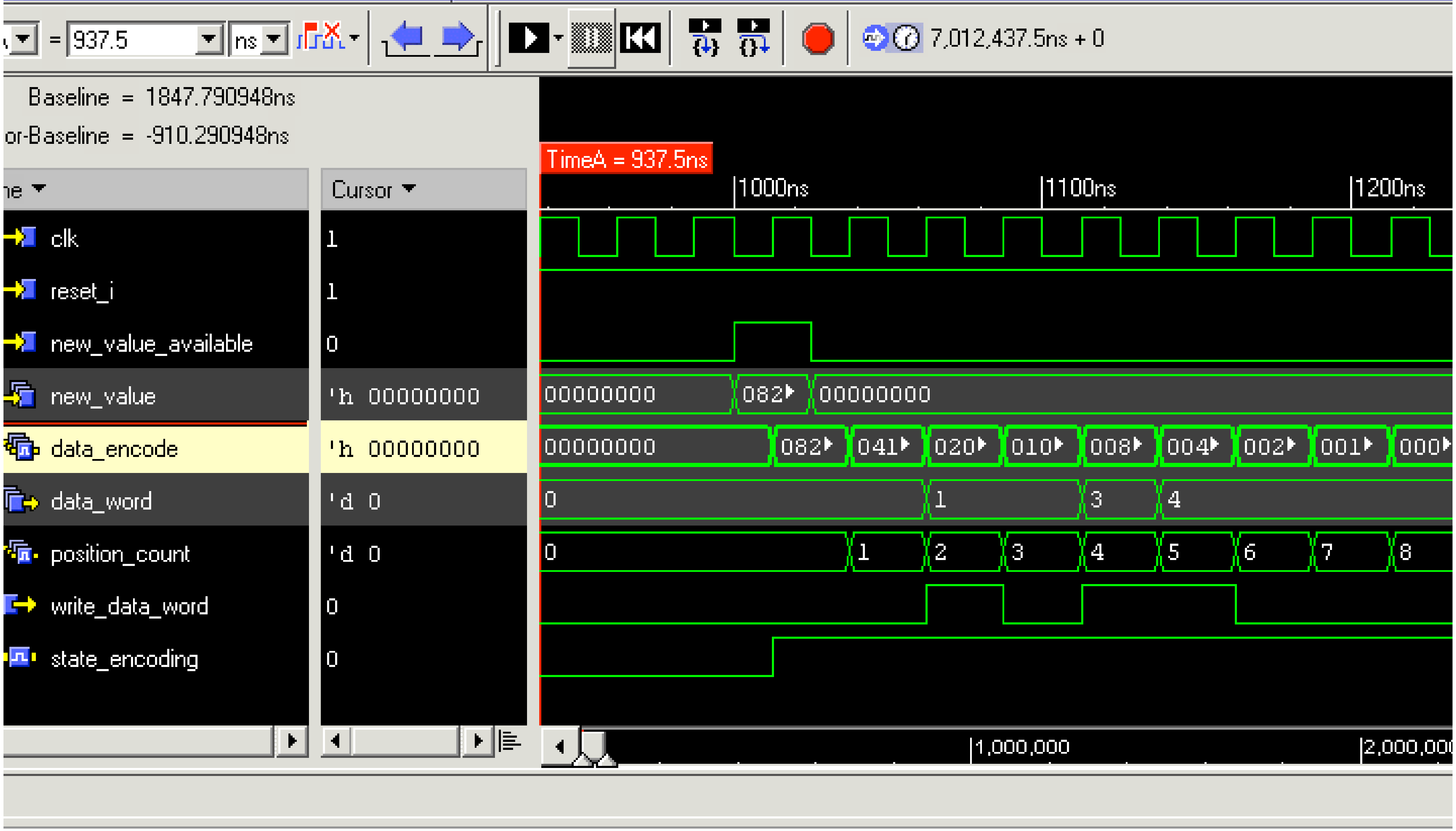
31	..	11	10	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1	0	0	0	0	0

"00001000000100000110000000000011010"


Baseline = 1847.790948ns
Cursor-Baseline = -910.290948ns

Name	Cursor
clk	1
reset_i	1
new_value_available	0
new_value	'h 00000000
data_encode	'h 00000000
data_word	'd 0
position_count	'd 0
write_data_word	0
state_encoding	0

Position decoder – shift register



937.5 ns

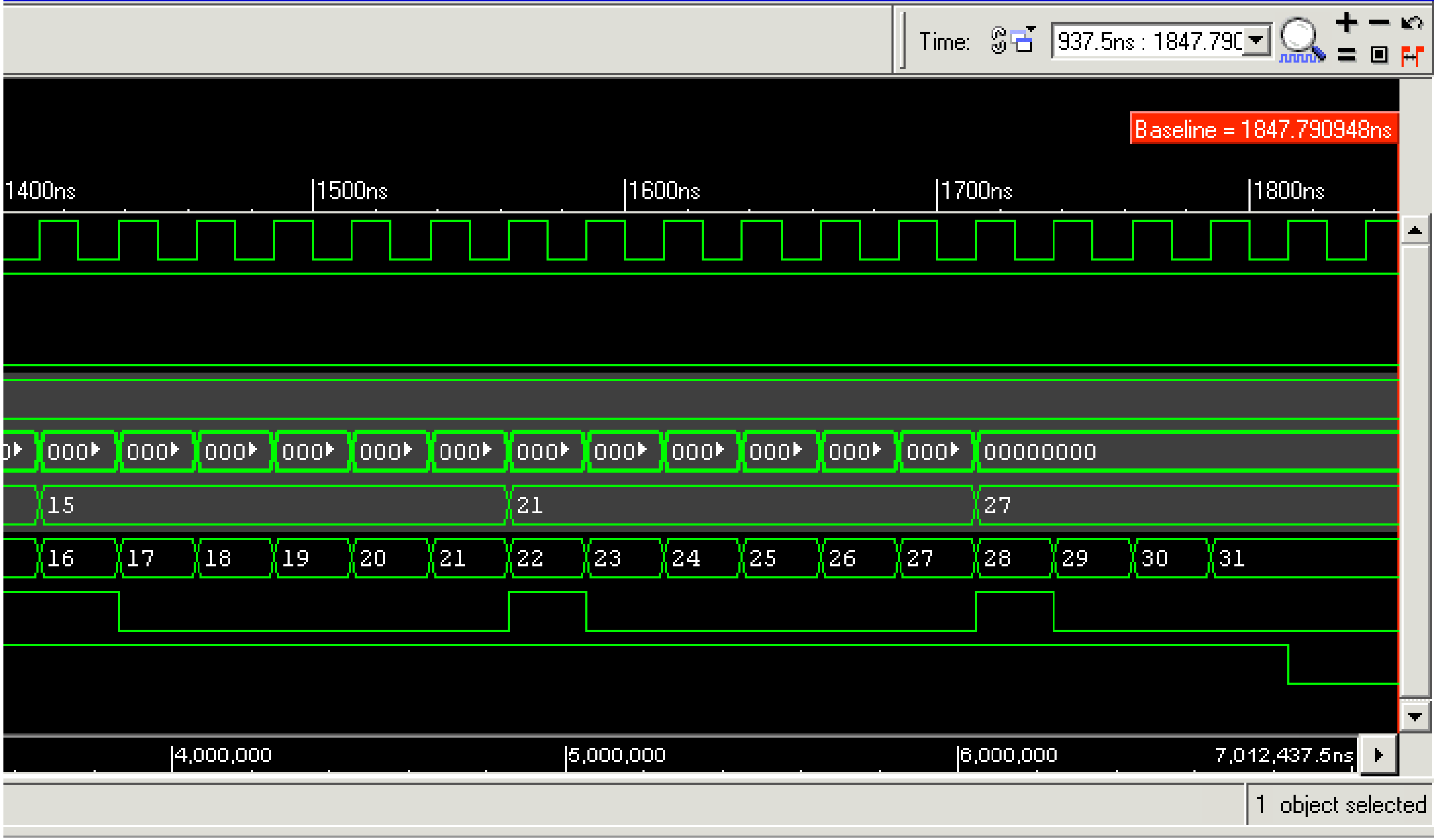
7,012,437.5ns + 0

Baseline = 1847.790948ns
or-Baseline = -910.290948ns

TimeA = 937.5ns

```
clk                  1
reset_i              1
new_value_available  0
new_value            'h 00000000    00000000 | 082 | 00000000
data_encode          'h 00000000    00000000 | 082 | 041 | 020 | 010 | 008 | 004 | 002 | 001 | 000
data_word            'd 0           0 | 1 | 3 | 4
position_count       'd 0           0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
write_data_word      0
state_encoding       0
```

Position decoder – shift register



Position decoder – shift register

31	..	11	10	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1	0	0	0	0	0

Shift register & counter (if then)
Result in an FPGA from 2002: (Altera EP20k200FC484-3)
81 out of 8320 logic elements
44 registers

11% (41/376) of pins

10.6 ns (94.5 MHz) position_count-> position_count

tco:	8.0 ns:	data_word_reg -> data_word
tsu:	7.0 ns:	new_value_available -> data_encode

Position decoder – shift register

31	..	11	10	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1	0	0	0	0	0

Shift register & counter (case)
Result in an FPGA from 2002: (Altera EP20k200FC484-3)
50 out of 8320 logic elements (with case statement)
44 registers

11% (41/376) of pins

9.1 ns (109.9 MHz) position_count-> data_encode

tco:	7.0 ns:	data_word_reg -> data_word
tsu:	6.3 ns:	new_value_available -> data_encode

Position decoder – shift register

- **Task fulfilled?**
  - Few logic cells
  - Timing constraints fulfilled
- **User requirements fulfilled?**
  - Processing per 32 bit line takes:
    - 32 bits * 25 ns = 800 ns
    - Data comes each 100 ns -> 1 out of 2560 32 bit line
    - Decoding time for all lines is: 2560 * 800 ns => 2 ms
    - Within 2 ms => 20480 data lines arrive
      - input FIFO would need to be at least 20k * 32 bit deep
    - During 2 ms no other trigger acquisition can take place
      - dead time => max trigger rate: 488 Hz
- **User requirements not fulfilled**

Position decoder – priority encoder

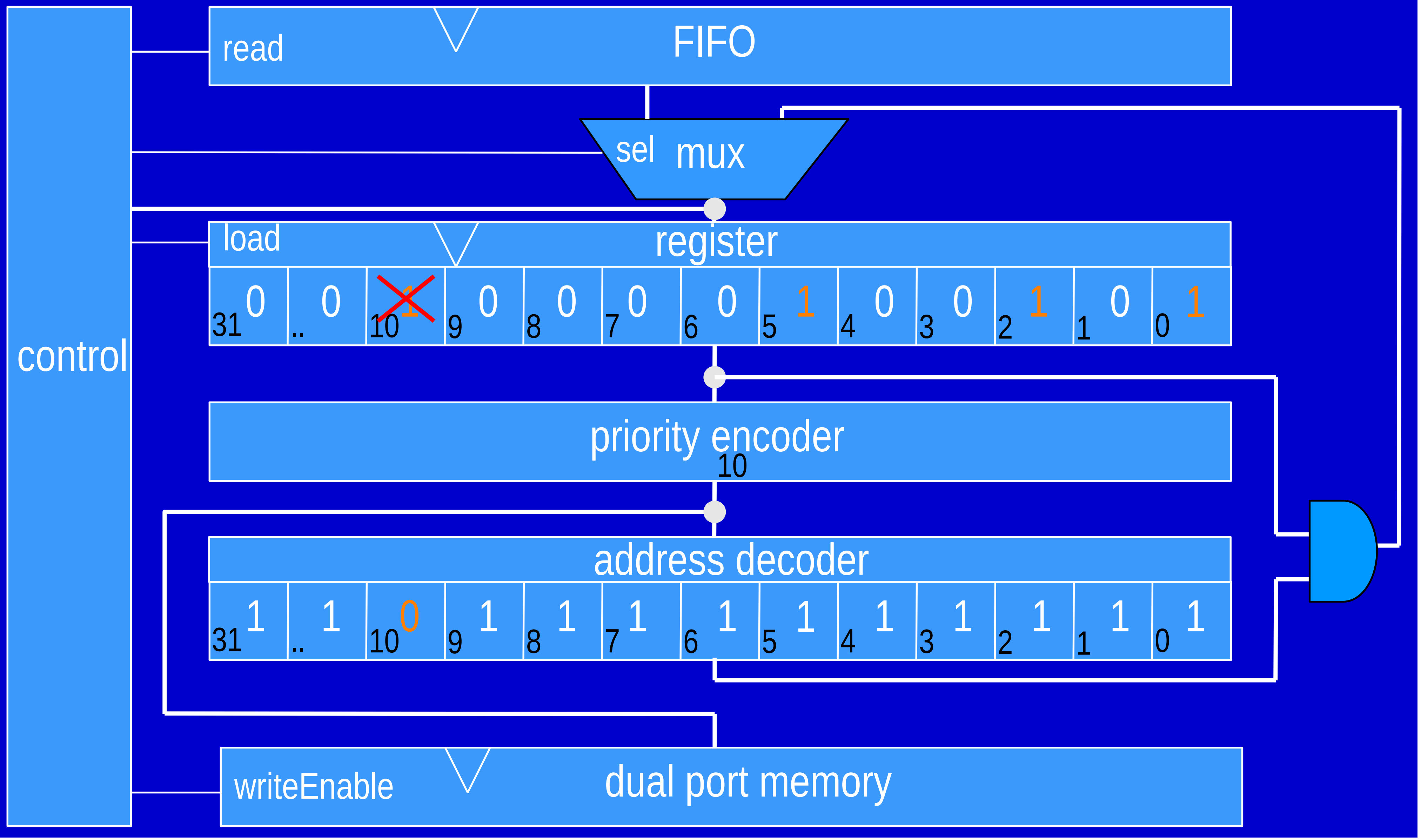
31	..	11	10	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1	0	0	0	0	0

How to decode the address:
this line has two hits
the state machine must send two hits into the dual port memory

	row address	hit position = 5

	row address	hit position = 11

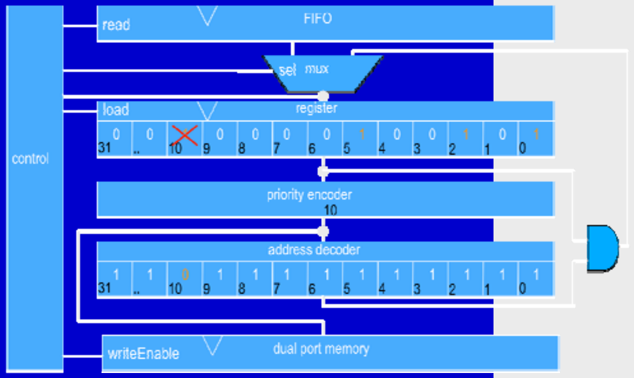
Position decoder – priority encoder



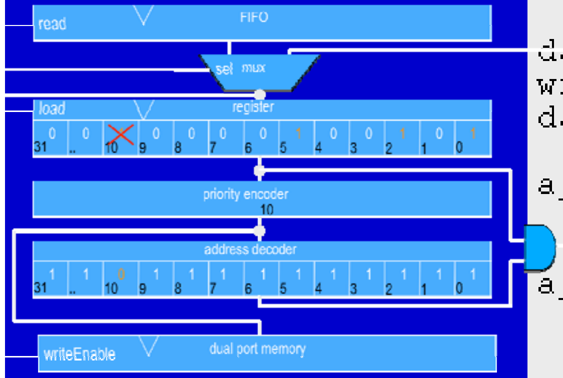
Position decoder – priority encoder

```vhdl
--*******************************************************************
--postionDecoderPri
--*******************************************************************

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

entity positionDecoder is
port (   clk                    :in   std_logic;
         reset_i                :in   std_logic;
         new_value_available    :in   std_logic;
         new_value              :in   std_logic_vector (31 downto 0);
         data_word              :out  integer range 0 to 31;
         write_data_word        :out  std_logic);

end positionDecoder;

architecture Priority of positionDecoder is

component prior32
port  (  inp      :in   std_logic_vector (31 downto 0);
         code     :out  std_logic_vector (4 downto 0));
end component;

component addressDecoder
port  (  inp      :in   std_logic_vector (4 downto 0);
         code     :out  std_logic_vector (31 downto 0));
end component;


signal data_encode              :std_logic_vector (31 downto 0);
signal state_encoding           :std_logic;
signal hit_address              :std_logic_vector (4 downto 0);
signal data_encode_actual       :std_logic_vector (31 downto 0);
signal data_encode_next         :std_logic_vector (31 downto 0);
signal data_encode_next_is_0    :std_logic;
signal new_value_is_0           :std_logic;
```

```vhdl
--******************************************************************
--postionDecoderPri
--******************************************************************

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

entity positionDecoder is
port (   clk                   :in   std_logic;
         reset_i               :in   std_logic;
         new_value_available   :in   std_logic;
         new_value             :in   std_logic_vector (31 downto 0);
         data_word             :out  integer range 0 to 31;
         write_data_word       :out  std_logic);

end positionDecoder;

architecture Priority of positionDecoder is

component prior32
port   (  inp        :in   std_logic_vector (31 downto 0);
          code       :out  std_logic_vector (4 downto 0));
end component;

component addressDecoder
port   (  inp        :in   std_logic_vector (4 downto 0);
          code       :out  std_logic_vector (31 downto 0));
end component;


signal data_encode            :std_logic_vector (31 downto 0);
signal state_encoding         :std_logic;
signal hit_address            :std_logic_vector (4 downto 0);
signal data_encode_actual     :std_logic_vector (31 downto 0);
signal data_encode_next       :std_logic_vector (31 downto 0);
signal data_encode_next_is_0  :std_logic;
signal new_value_is_0         :std_logic;
```



```
begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      data_encode                       <= (others => '0');
      state_encoding                    <= '0';
   else case state_encoding is
      when '0' =>
         if (new_value_available = '1') then
            data_encode                 <= new_value;
            state_encoding              <= not new_value_is_0;
         else
            state_encoding              <= '0';
         end if;
      when '1' =>
         if (data_encode_next_is_0 = '0') then
            data_encode                 <= data_encode_next;
            state_encoding              <= '1';
         elsif (data_encode_next_is_0 = '1') then
            data_encode                 <= data_encode_next;
            state_encoding              <= '0';
         end if;
      when others =>
         data_encode                    <= (others => '0');
         state_encoding                 <= '0';
      end case;
   end if;
end if;

end process;

data_encode_next        <= data_encode and data_encode_actual;
write_data_word         <= state_encoding;
data_word               <= to_integer (unsigned(hit_address));

a_prior32:        prior32
   port map (data_encode, hit_address);

a_addressDecoder: addressDecoder
   port map (hit_address, data_encode_actual);
```

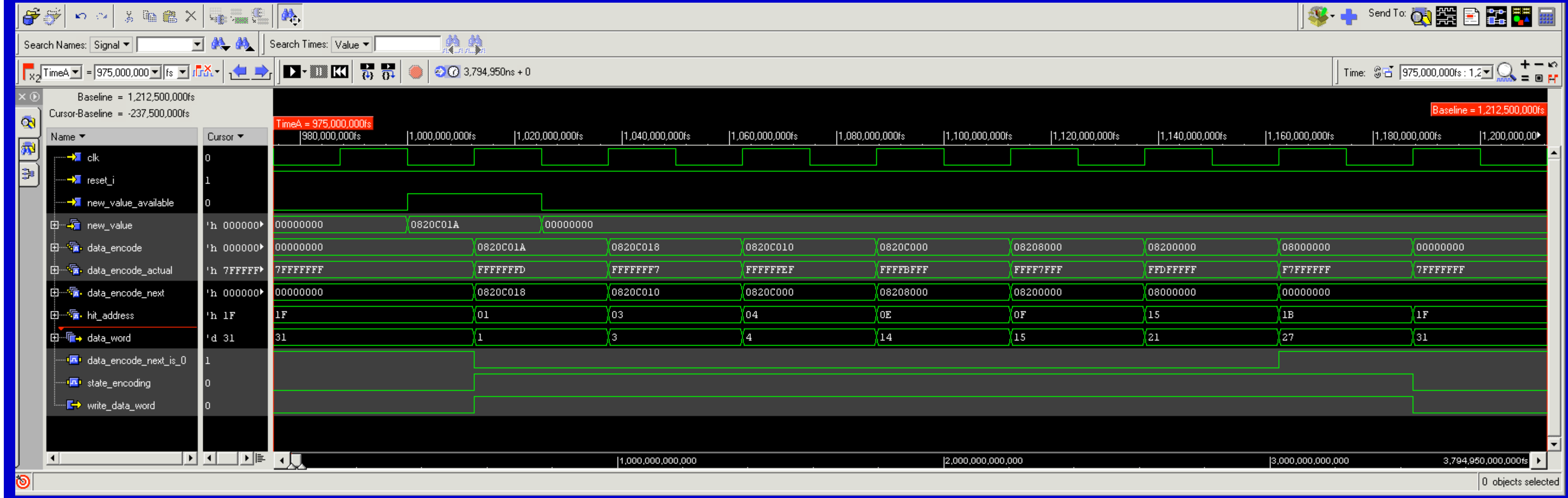


```
            data_encode                        <= data_encode_next;
            state_encoding                     <= '0';
         end if;
      when others =>
         data_encode                           <= (others => '0');
         state_encoding                        <= '0';
      end case;
   end if;
end if;

end process;

data_encode_next        <= data_encode and data_encode_actual;
write_data_word         <= state_encoding;
data_word               <= to_integer (unsigned(hit_address));

a_prior32:        prior32
   port map (data_encode, hit_address);

a_addressDecoder: addressDecoder
   port map (hit_address, data_encode_actual);



process (data_encode_next)
begin
   if (data_encode_next = "00000000000000000000000000000000") then
      data_encode_next_is_0   <= '1';
   else
      data_encode_next_is_0   <= '0';
   end if;
end process;

process (new_value)
begin
   if (new_value = "00000000000000000000000000000000") then
      new_value_is_0 <= '1';
   else
      new_value_is_0 <= '0';
   end if;
end process;

end Priority;
```

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

entity prior32 is
port  (  inp       :in   std_logic_vector (31 downto 0);
         code      :out  std_logic_vector (4 downto 0));
end prior32;

architecture behavioral0 of prior32 is

--************************************************************
--prior32
--************************************************************
begin

process (inp)
begin
   if    (inp(0) = '1') then      code <= "00000";
   elsif (inp(1) = '1') then      code <= "00001";
   elsif (inp(2) = '1') then      code <= "00010";
   elsif (inp(3) = '1') then      code <= "00011";
   elsif (inp(4) = '1') then      code <= "00100";
   elsif (inp(5) = '1') then      code <= "00101";
   elsif (inp(6) = '1') then      code <= "00110";
   elsif (inp(7) = '1') then      code <= "00111";
   elsif (inp(8) = '1') then      code <= "01000";
   elsif (inp(9) = '1') then      code <= "01001";
   elsif (inp(10) = '1') then     code <= "01010";
   elsif (inp(11) = '1') then     code <= "01011";
   elsif (inp(12) = '1') then     code <= "01100";
   elsif (inp(13) = '1') then     code <= "01101";
   elsif (inp(14) = '1') then     code <= "01110";
   elsif (inp(15) = '1') then     code <= "01111";
   elsif (inp(16) = '1') then     code <= "10000";
   elsif (inp(17) = '1') then     code <= "10001";
   elsif (inp(18) = '1') then     code <= "10010";
   elsif (inp(19) = '1') then     code <= "10011";
   elsif (inp(20) = '1') then     code <= "10100";
   elsif (inp(21) = '1') then     code <= "10101";
   elsif (inp(22) = '1') then     code <= "10110";
   elsif (inp(23) = '1') then     code <= "10111";
   elsif (inp(24) = '1') then     code <= "11000";
   elsif (inp(25) = '1') then     code <= "11001";
   elsif (inp(26) = '1') then     code <= "11010";
   elsif (inp(27) = '1') then     code <= "11011";
   elsif (inp(28) = '1') then     code <= "11100";
   elsif (inp(29) = '1') then     code <= "11101";
   elsif (inp(30) = '1') then     code <= "11110";
   elsif (inp(31) = '1') then     code <= "11111";
   else                           code <= "11111";
end if;

end process;

end behavioral0;
```

```vhdl
--*********************************************************************
--adressDecoder
--*********************************************************************

library ieee;
use ieee.std_logic_1164.all;

entity addressDecoder is
port  (  inp      :in   std_logic_vector (4 downto 0);
         code     :out  std_logic_vector (31 downto 0));
end addressDecoder;

architecture behavioral of addressDecoder is

begin

process (inp)
begin
   case (inp) is
      when "00000" =>   code  <= "11111111111111111111111111111110";
      when "00001" =>   code  <= "11111111111111111111111111111101";
      when "00010" =>   code  <= "11111111111111111111111111111011";
      when "00011" =>   code  <= "11111111111111111111111111110111";
      when "00100" =>   code  <= "11111111111111111111111111101111";
      when "00101" =>   code  <= "11111111111111111111111111011111";
      when "00110" =>   code  <= "11111111111111111111111110111111";
      when "00111" =>   code  <= "11111111111111111111111101111111";
      when "01000" =>   code  <= "11111111111111111111111011111111";
      when "01001" =>   code  <= "11111111111111111111110111111111";
      when "01010" =>   code  <= "11111111111111111111101111111111";
      when "01011" =>   code  <= "11111111111111111111011111111111";
      when "01100" =>   code  <= "11111111111111111110111111111111";
      when "01101" =>   code  <= "11111111111111111101111111111111";
      when "01110" =>   code  <= "11111111111111111011111111111111";
      when "01111" =>   code  <= "11111111111111110111111111111111";
      when "10000" =>   code  <= "11111111111111101111111111111111";
      when "10001" =>   code  <= "11111111111111011111111111111111";
      when "10010" =>   code  <= "11111111111110111111111111111111";
      when "10011" =>   code  <= "11111111111101111111111111111111";
      when "10100" =>   code  <= "11111111111011111111111111111111";
      when "10101" =>   code  <= "11111111110111111111111111111111";
      when "10110" =>   code  <= "11111111101111111111111111111111";
      when "10111" =>   code  <= "11111111011111111111111111111111";
      when "11000" =>   code  <= "11111110111111111111111111111111";
      when "11001" =>   code  <= "11111101111111111111111111111111";
      when "11010" =>   code  <= "11111011111111111111111111111111";
      when "11011" =>   code  <= "11110111111111111111111111111111";
      when "11100" =>   code  <= "11101111111111111111111111111111";
      when "11101" =>   code  <= "11011111111111111111111111111111";
      when "11110" =>   code  <= "10111111111111111111111111111111";
      when "11111" =>   code  <= "01111111111111111111111111111111";
      when others =>    code  <= "11111111111111111111111111111111";
   end case;

end process;

end behavioral;
```

Position decoder – priority encoder



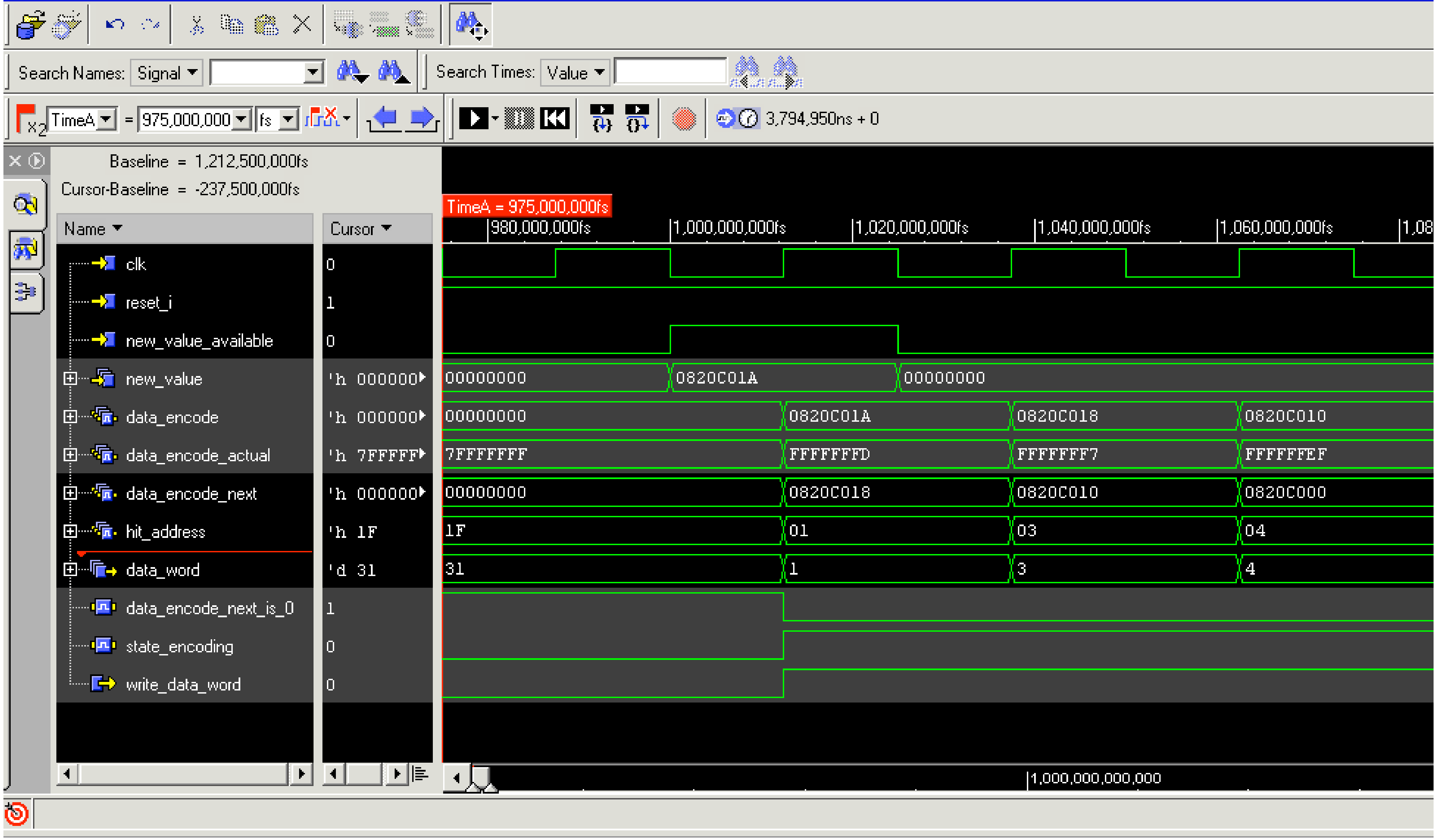
Baseline = 1,212,500,000fs
Cursor-Baseline = -237,500,000fs
TimeA = 975,000,000fs

```
Name                    Cursor
clk                     0
reset_i                 1
new_value_available     0
new_value               'h 000000
data_encode             'h 000000
data_encode_actual      'h 7FFFFF
data_encode_next        'h 000000
hit_address             'h 1F
data_word               'd 31
data_encode_next_is_0   1
state_encoding          0
write_data_word         0
```

```
new_value           00000000  0820C01A  00000000
data_encode         00000000  0820C01A  0820C018  0820C010  0820C000  08208000  08200000  08000000  00000000
data_encode_actual  7FFFFFFF  FFFFFFFD  FFFFFFF7  FFFFFFEF  FFFFBFFF  FFFF7FFF  FFDFFFFF  F7FFFFFF  7FFFFFFF
data_encode_next    00000000  0820C018  0820C010  0820C000  08208000  08200000  08000000  00000000
hit_address         1F        01        03        04        0E        0F        15        1B        1F
data_word           31        1         3         4         14        15        21        27        31
```

3,794,950ns + 0
0 objects selected

Position decoder – priority encoder



Search Names: Signal

Search Times: Value

TimeA = 975,000,000 fs

3,794,950ns + 0

Baseline = 1,212,500,000fs

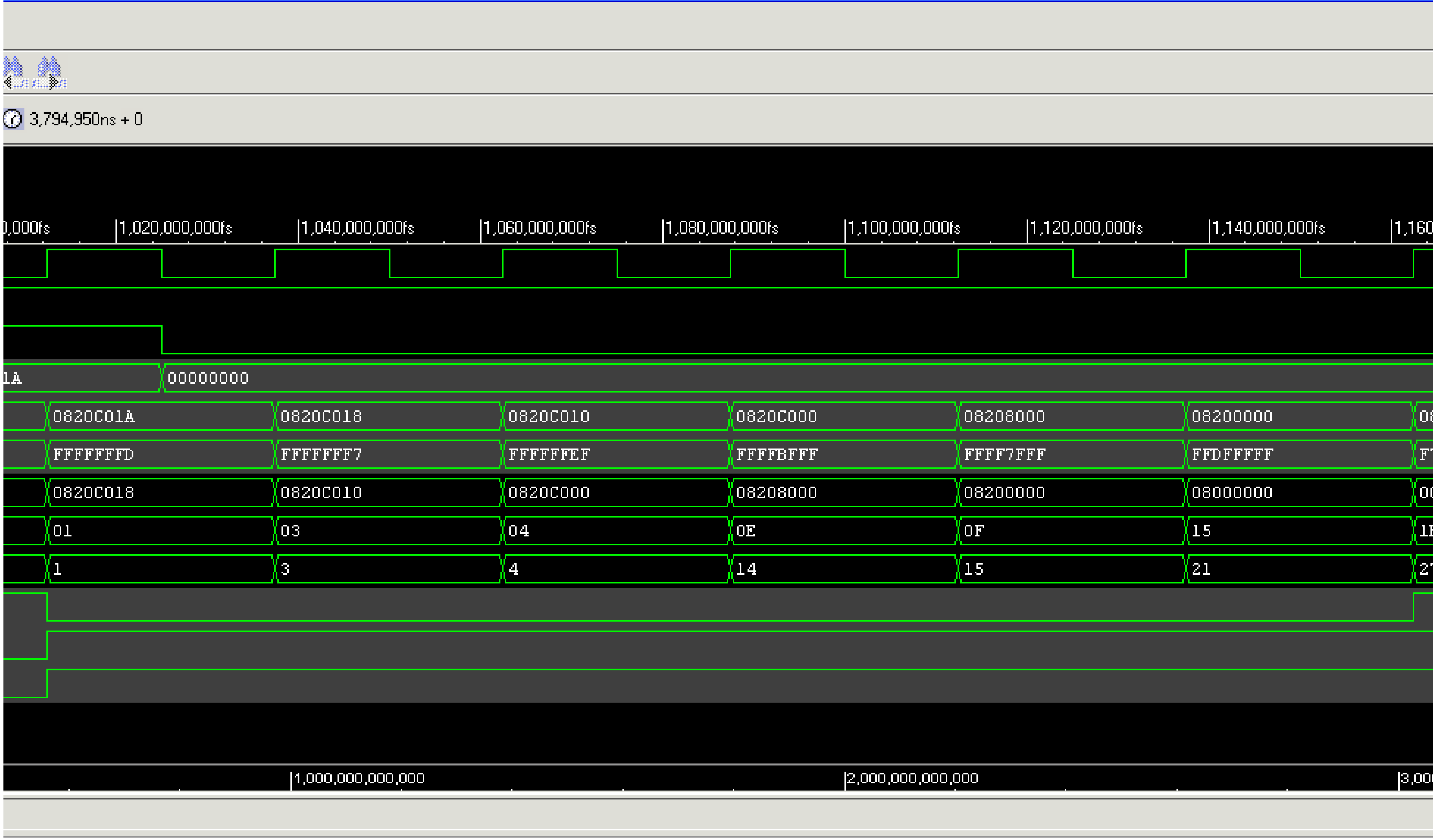
Cursor-Baseline = -237,500,000fs

TimeA = 975,000,000fs

```
Name                     Cursor       |980,000,000fs   |1,000,000,000fs  |1,020,000,000fs  |1,040,000,000fs  |1,060,000,000fs
clk                      0
reset_i                  1
new_value_available      0
new_value                'h 000000    00000000          0820C01A          00000000
data_encode              'h 000000    00000000                  0820C01A          0820C018          0820C010
data_encode_actual       'h 7FFFFF    7FFFFFFF                  FFFFFFFD          FFFFFFF7          FFFFFFEF
data_encode_next         'h 000000    00000000                  0820C018          0820C010          0820C000
hit_address              'h 1F        1F                        01                03                04
data_word                'd 31        31                        1                 3                 4
data_encode_next_is_0    1
state_encoding           0
write_data_word          0
```

|1,000,000,000,000

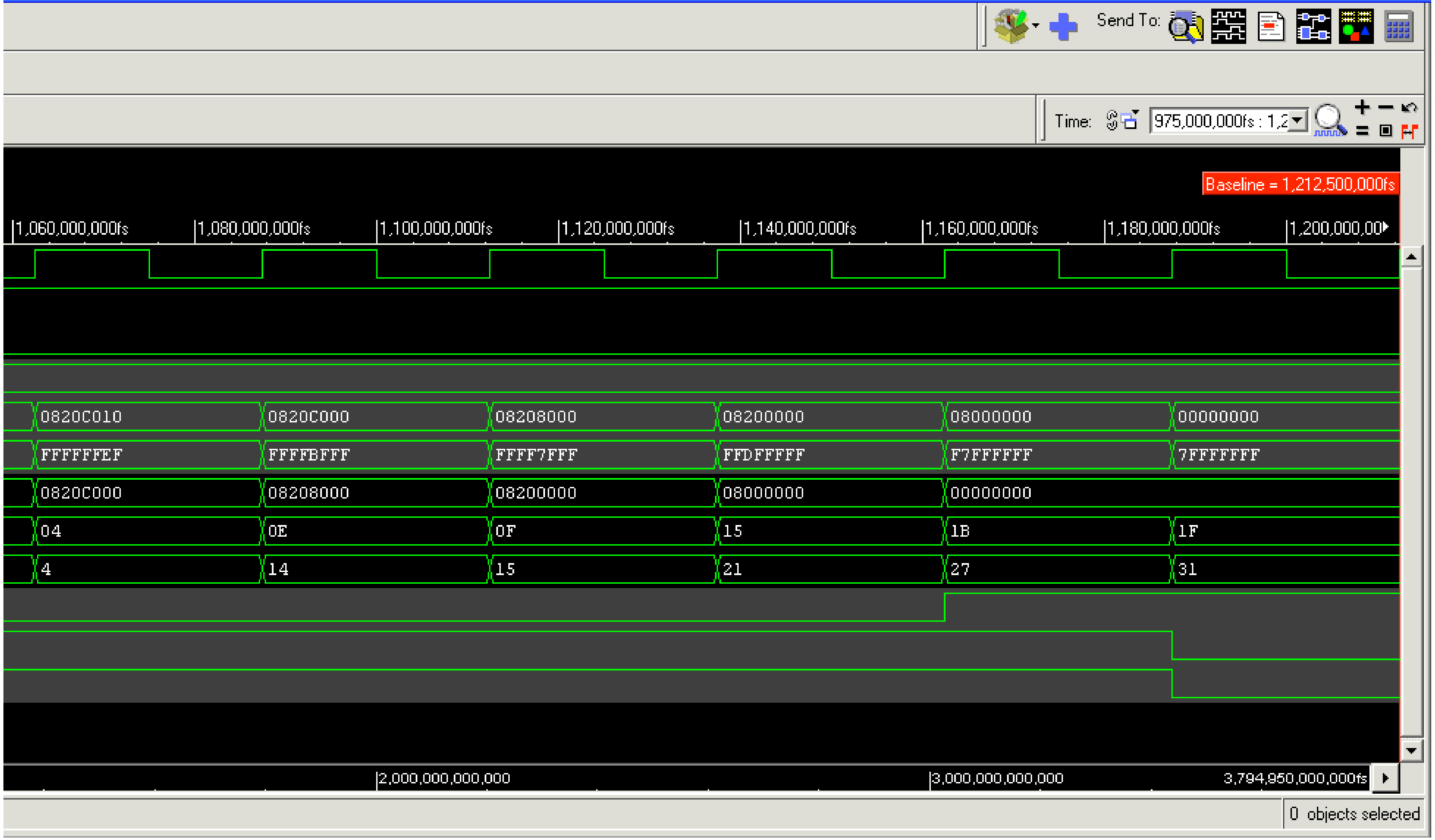
Position decoder – priority encoder



3,794,950ns + 0

```
),000fs   |1,020,000,000fs   |1,040,000,000fs   |1,060,000,000fs   |1,080,000,000fs   |1,100,000,000fs   |1,120,000,000fs   |1,140,000,000fs   |1,160

1A        00000000
          0820C01A    0820C018    0820C010    0820C000    08208000    08200000    08
          FFFFFFFD    FFFFFFF7    FFFFFFEF    FFFFBFFF    FFFF7FFF    FFDFFFFF    F7
          0820C018    0820C010    0820C000    08208000    08200000    08000000    00
          01          03          04          0E          0F          15          1E
          1           3           4           14          15          21          27

|1,000,000,000,000          |2,000,000,000,000          |3,00
```

Position decoder – priority encoder



Send To:

Time: 975,000,000fs : 1,2

Baseline = 1,212,500,000fs

```
|1,060,000,000fs  |1,080,000,000fs  |1,100,000,000fs  |1,120,000,000fs  |1,140,000,000fs  |1,160,000,000fs  |1,180,000,000fs  |1,200,000,00

0820C010   0820C000   08208000   08200000   08000000   00000000
FFFFFFEF   FFFFBFFF   FFFF7FFF   FFDFFFFF   F7FFFFFF   7FFFFFFF
0820C000   08208000   08200000   08000000   00000000
04         0E         0F         15         1B         1F
4          14         15         21         27         31

|2,000,000,000,000                |3,000,000,000,000          3,794,950,000,000fs
```

0 objects selected

Shift register & counter (case)
Result in an FPGA from 2002: (Altera EP20k200FC484-3)
50 with case out of 8320 logic elements
44 registers

11% (41/376) of pins

9.1 ns (109.9 MHz) position_count-> data_encode

tco: 7.0 ns: data_word_reg -> data_word
tsu: 6.3 ns: new_value_available -> data_encode

Position decoder – priority encoder

31	..	11	10	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1	0	0	0	0	0

Priority encoder
Result in an FPGA from 2002: (Altera EP20k200FC484-3)
172 (out of 8320) logic elements
33 registers
addressDecoder: 16
prior32: 54
11% (41/376) of pins

20.8 ns (48.0 MHz) data_encode -> state_encoding

tco: 17.1 ns:data_encode -> data_word
tsu: 14.9 ns:new_value -> state_encoding

Position decoder – priority encoder

- **Task fulfilled?**
  - Many logic cells
  - FPGA Timing constraints fulfilled
- **User requirements fulfilled?**
  - Processing per 32 bit line takes:
    - numbHits per line * 25 ns = ?
    - Data comes each 100 ns -> one out of 2560 32 bit line
    - Decoding time for all lines is: 2560 * ? ns => ? ms
    - Within ? ms => ? data lines arrive
      - input FIFO would need to be at least ? * 32 bit deep
    - During ? ms no other trigger acquisition can take place
      - dead time => max trigger rate: ? Hz
- **User requirements fulfilled ?**

Position decoder – priority encoder

- **Task fulfilled?**
  - Physics simulation:
    - max 2% of all pixels will be hit in one acquisition
- **User requirements fulfilled?**
  - Processing per 32 bit line takes:
    - (numbHits per line) * 25 ns = (32 * 0.02) * 25 ns = <25 ns
    - Data comes each 100 ns -> one out of 2560 32 bit line
    - One line with up to 4 hits can be decoded before the next line arrives
    - Input FIFO of 1000 * 32 bits implemented to buffer statistical fluctuations or calibration sequences
    - Dead time defined by transmission of data stream
      - 2560 lines each 100 ns => 256 µs => 3900 Hz
      - dead time => max trigger rate: 3900 Hz
- **User requirements fulfilled: yes**

Position decoder – priority encoder

- User requirements fulfilled: yes
- Can we do better?
- Can we do faster or with less logic?
- Do we know something which the synthesizer does not know?

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

entity prior32 is
port  (   inp       :in    std_logic_vector (31 downto 0);
          code      :out   std_logic_vector (4 downto 0));
end prior32;

architecture behavioral0 of prior32 is

--****************************************************
--prior32
--****************************************************
begin

process (inp)
begin
    if    (inp(0) = '1') then     code <= "00000";
    elsif (inp(1) = '1') then     code <= "00001";
    elsif (inp(2) = '1') then     code <= "00010";
    elsif (inp(3) = '1') then     code <= "00011";
    elsif (inp(4) = '1') then     code <= "00100";
    elsif (inp(5) = '1') then     code <= "00101";
    elsif (inp(6) = '1') then     code <= "00110";
    elsif (inp(7) = '1') then     code <= "00111";
    elsif (inp(8) = '1') then     code <= "01000";
    elsif (inp(9) = '1') then     code <= "01001";
    elsif (inp(10) = '1') then    code <= "01010";
    elsif (inp(11) = '1') then    code <= "01011";
    elsif (inp(12) = '1') then    code <= "01100";
    elsif (inp(13) = '1') then    code <= "01101";
    elsif (inp(14) = '1') then    code <= "01110";
    elsif (inp(15) = '1') then    code <= "01111";
    elsif (inp(16) = '1') then    code <= "10000";
    elsif (inp(17) = '1') then    code <= "10001";
    elsif (inp(18) = '1') then    code <= "10010";
    elsif (inp(19) = '1') then    code <= "10011";
    elsif (inp(20) = '1') then    code <= "10100";
    elsif (inp(21) = '1') then    code <= "10101";
    elsif (inp(22) = '1') then    code <= "10110";
    elsif (inp(23) = '1') then    code <= "10111";
    elsif (inp(24) = '1') then    code <= "11000";
    elsif (inp(25) = '1') then    code <= "11001";
    elsif (inp(26) = '1') then    code <= "11010";
    elsif (inp(27) = '1') then    code <= "11011";
    elsif (inp(28) = '1') then    code <= "11100";
    elsif (inp(29) = '1') then    code <= "11101";
    elsif (inp(30) = '1') then    code <= "11110";
    elsif (inp(31) = '1') then    code <= "11111";
    else                          code <= "11111";
end if;

end process;

end behavioral0;
```

```vhdl
architecture behavioral1 of prior32 is
signal code0   :  std_logic_vector (2 downto 0);
signal code1   :  std_logic_vector (2 downto 0);
signal code2   :  std_logic_vector (2 downto 0);
signal code3   :  std_logic_vector (2 downto 0);
signal code4   :  std_logic_vector (2 downto 0);
signal code5   :  std_logic_vector (2 downto 0);
signal code6   :  std_logic_vector (2 downto 0);
signal code7   :  std_logic_vector (2 downto 0);
begin
process (inp)
begin
   if    (inp(0) = '1') then      code0 <= "000";
   elsif (inp(1) = '1') then      code0 <= "001";
   elsif (inp(2) = '1') then      code0 <= "010";
   elsif (inp(3) = '1') then      code0 <= "011";
   else                           code0 <= "100";
   end if;

   if    (inp(4) = '1') then      code1 <= "000";
   elsif (inp(5) = '1') then      code1 <= "001";
   elsif (inp(6) = '1') then      code1 <= "010";
   elsif (inp(7) = '1') then      code1 <= "011";
   else                           code1 <= "100";
   end if;

   if    (inp(8) = '1') then      code2 <= "000";
   elsif (inp(9) = '1') then      code2 <= "001";
   elsif (inp(10) = '1') then     code2 <= "010";
   elsif (inp(11) = '1') then     code2 <= "011";
   else                           code2 <= "100";
   end if;

   if    (inp(12) = '1') then     code3 <= "000";
   elsif (inp(13) = '1') then     code3 <= "001";
   elsif (inp(14) = '1') then     code3 <= "010";
   elsif (inp(15) = '1') then     code3 <= "011";
   else                           code3 <= "100";
   end if;
   if    (inp(16) = '1') then     code4 <= "000";
   elsif (inp(17) = '1') then     code4 <= "001";
   elsif (inp(18) = '1') then     code4 <= "010";
   elsif (inp(19) = '1') then     code4 <= "011";
   else                           code4 <= "100";
   end if;

   if    (inp(20) = '1') then     code5 <= "000";
   elsif (inp(21) = '1') then     code5 <= "001";
   elsif (inp(22) = '1') then     code5 <= "010";
   elsif (inp(23) = '1') then     code5 <= "011";
   else                           code5 <= "100";
   end if;

   if    (inp(24) = '1') then     code6 <= "000";
   elsif (inp(25) = '1') then     code6 <= "001";
   elsif (inp(26) = '1') then     code6 <= "010";
   elsif (inp(27) = '1') then     code6 <= "011";
   else                           code6 <= "100";
   end if;

   if    (inp(28) = '1') then     code7 <= "000";
   elsif (inp(29) = '1') then     code7 <= "001";
   elsif (inp(30) = '1') then     code7 <= "010";
   elsif (inp(31) = '1') then     code7 <= "011";
   else                           code7 <= "100";
   end if;
end process;
```

Position decoder – priority encoder

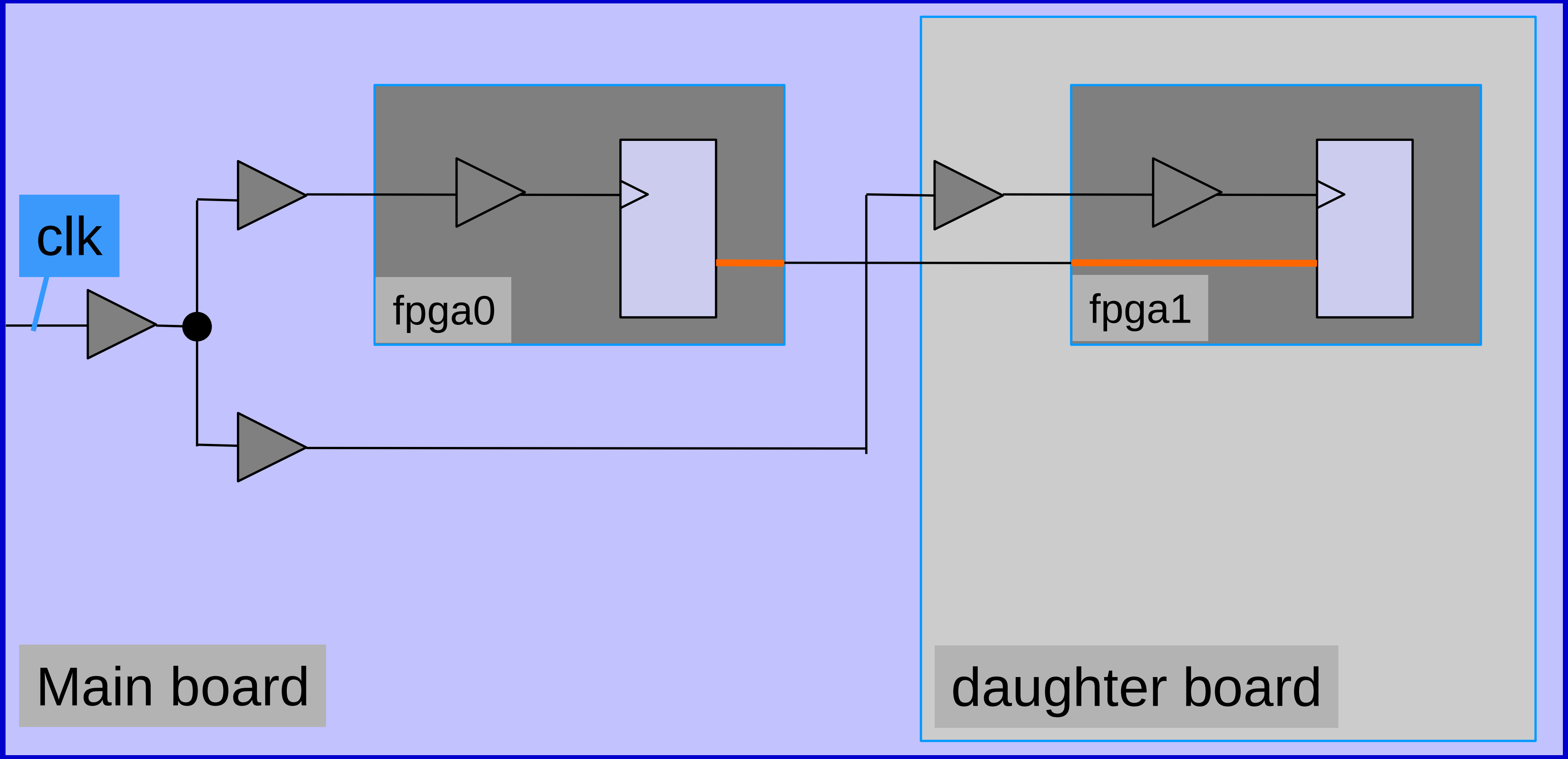
```
process (code0, code1, code2, code3, code4, code5, code6, code7)

begin
    if (code0(2) = '0') then
        code(4 downto 2)  <= "000";
        code(1 downto 0)  <= code0(1 downto 0);
    elsif(code1(2) = '0') then
        code(4 downto 2)  <= "001";
        code(1 downto 0)  <= code1(1 downto 0);
    elsif(code2(2) = '0') then
        code(4 downto 2)  <= "010";
        code(1 downto 0)  <= code2(1 downto 0);
    elsif(code3(2) = '0') then
        code(4 downto 2)  <= "011";
        code(1 downto 0)  <= code3(1 downto 0);
    elsif (code4(2) = '0') then
        code(4 downto 2)  <= "100";
        code(1 downto 0)  <= code4(1 downto 0);
    elsif(code5(2) = '0') then
        code(4 downto 2)  <= "101";
        code(1 downto 0)  <= code5(1 downto 0);
    elsif(code6(2) = '0') then
        code(4 downto 2)  <= "110";
        code(1 downto 0)  <= code6(1 downto 0);
    elsif(code7(2) = '0') then
        code(4 downto 2)  <= "111";
        code(1 downto 0)  <= code7(1 downto 0);
    else  code            <= "11111";
    end if;
end process;
```

Position decoder – priority encoder

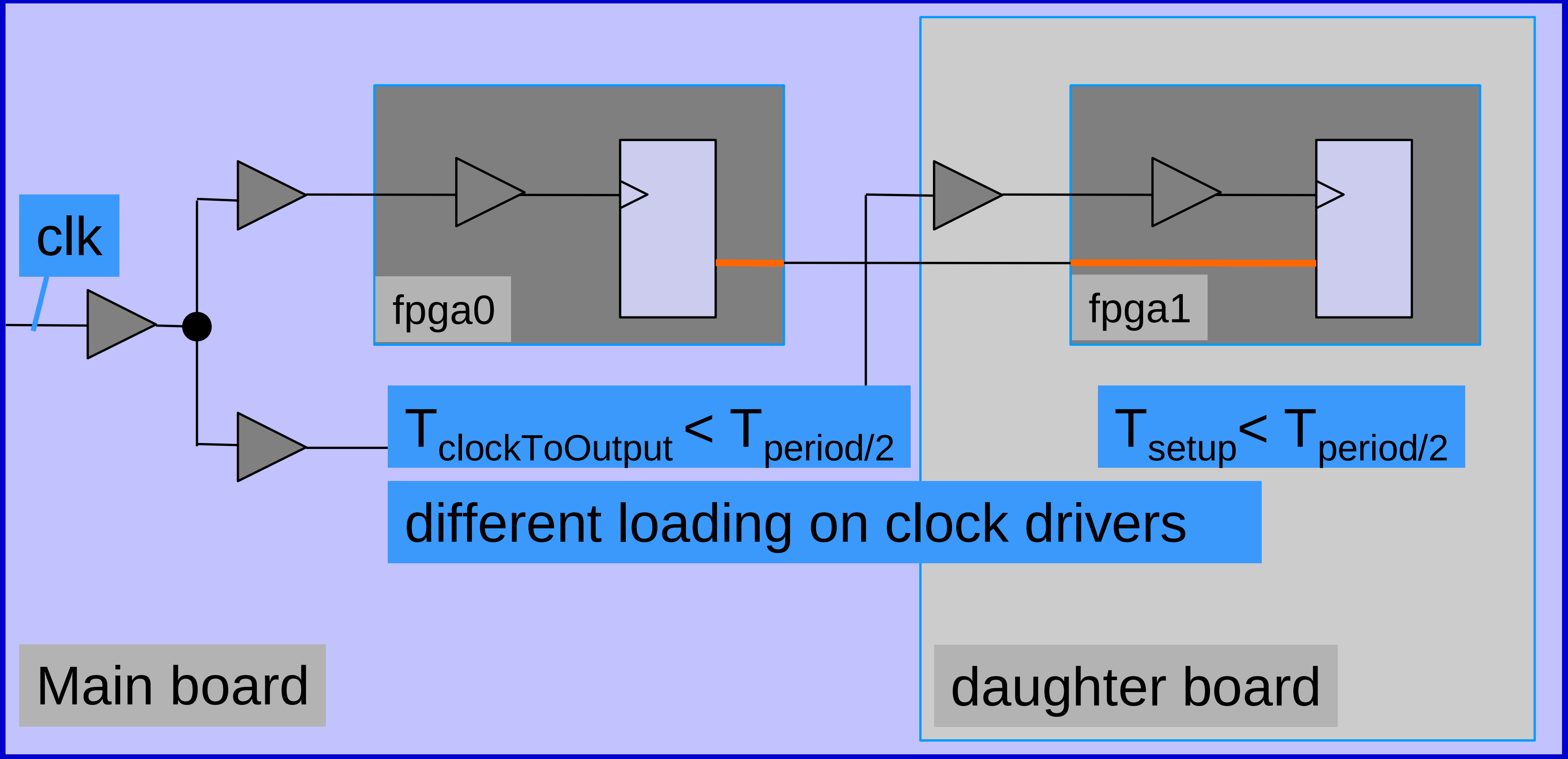
- **Knowledge of implementation in target technology is important**
- **Knowledge of what the synthesizer is doing is important**

Clock domains – multiple FPGA design

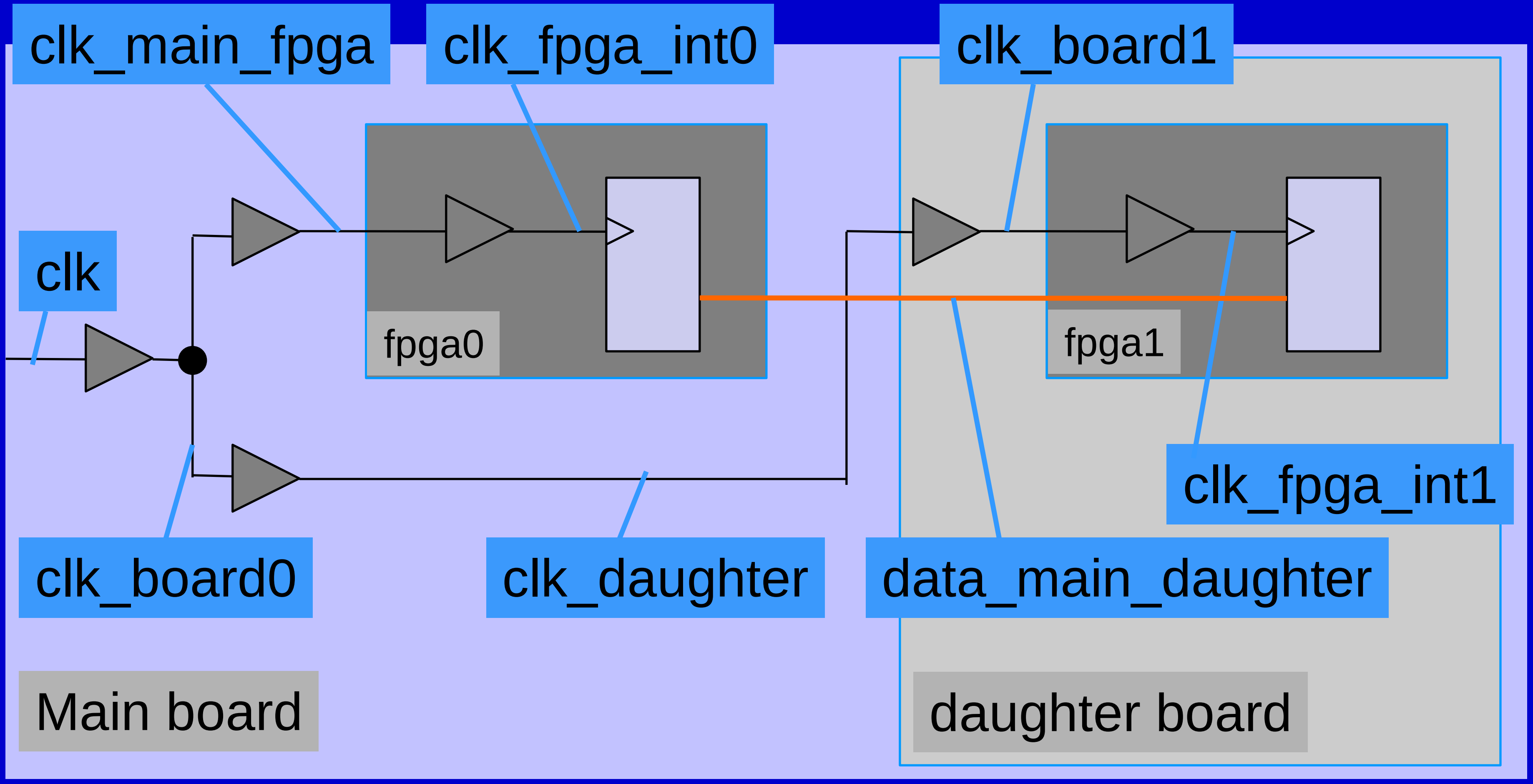
Clock distribution: multiple FPGAs



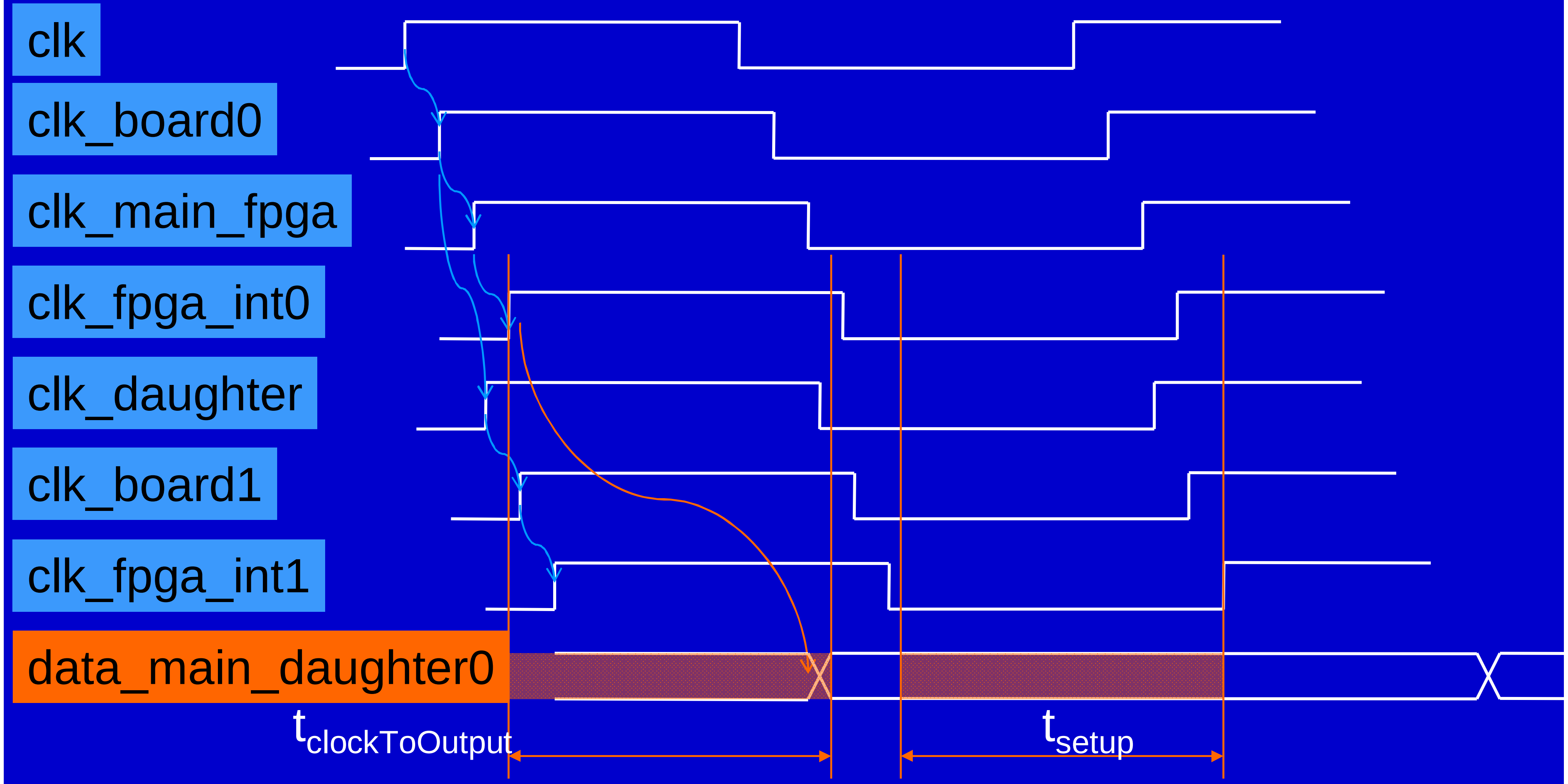
Clock distribution: multiple FPGAs


clk
fpga0
fpga1
$T_{clockToOutput} < T_{period/2}$
$T_{setup} < T_{period/2}$
different loading on clock drivers
Main board
daughter board

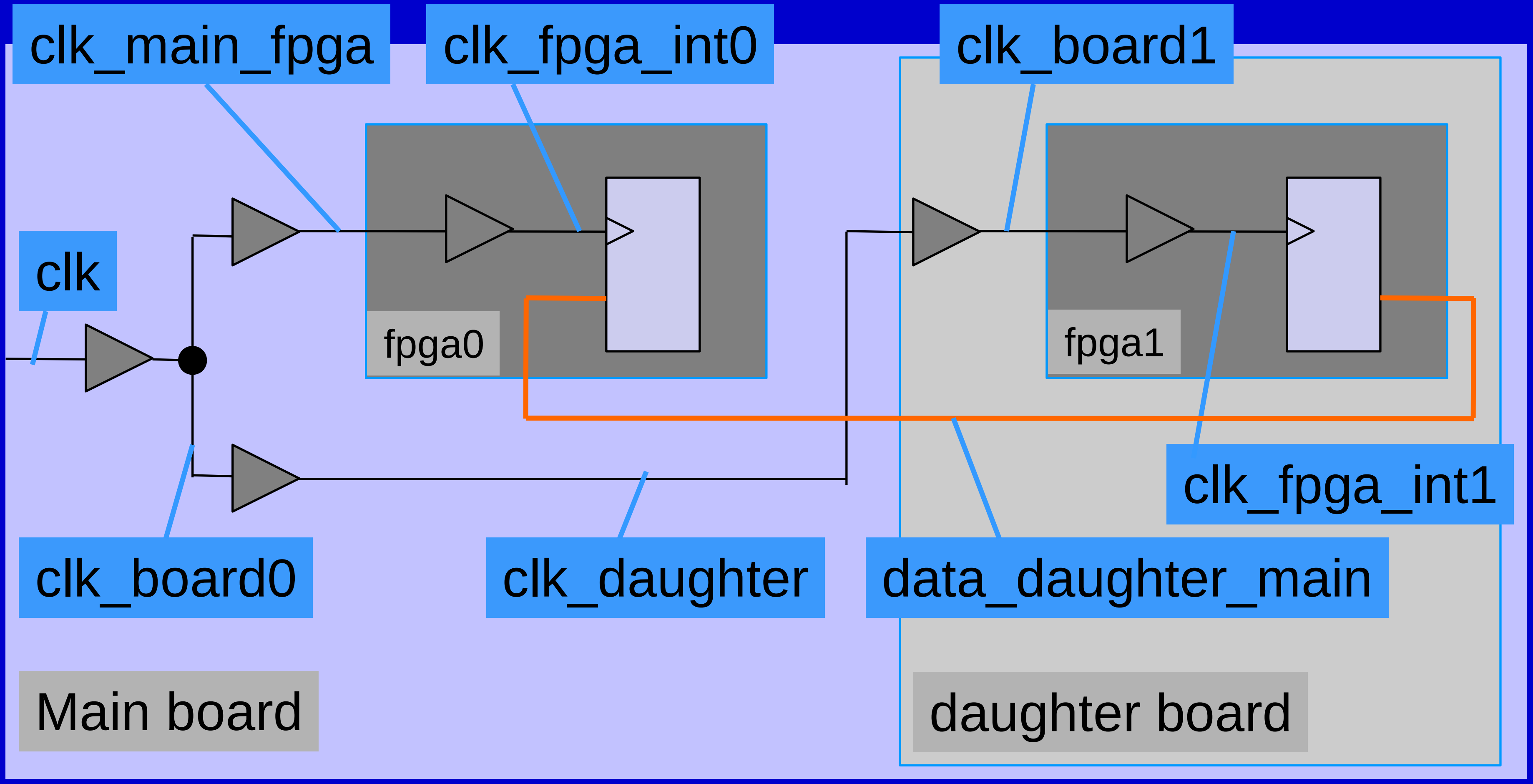
Clock distribution



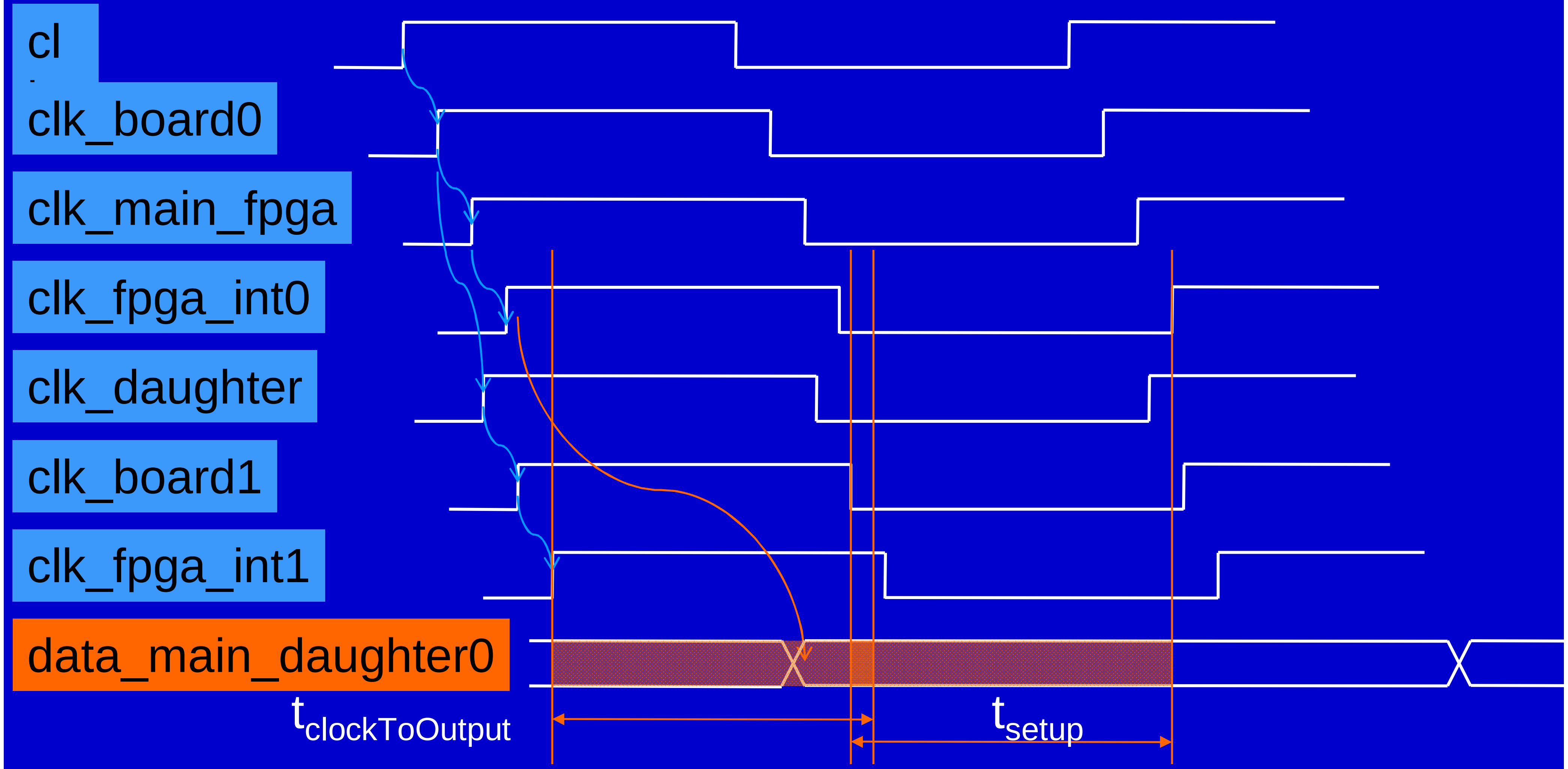
clock distribution/t_{co} & t_s /0-> 1



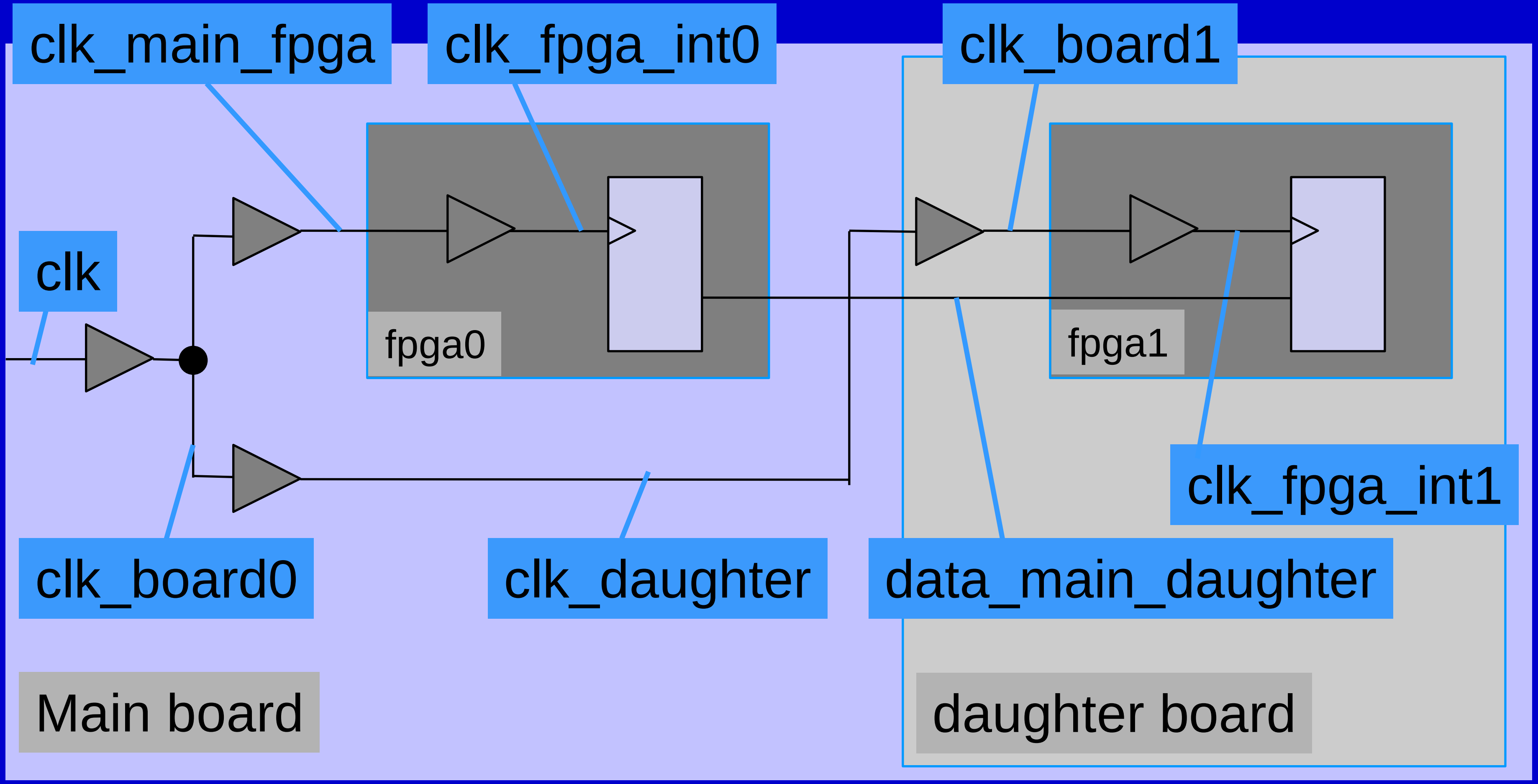
Clock distribution



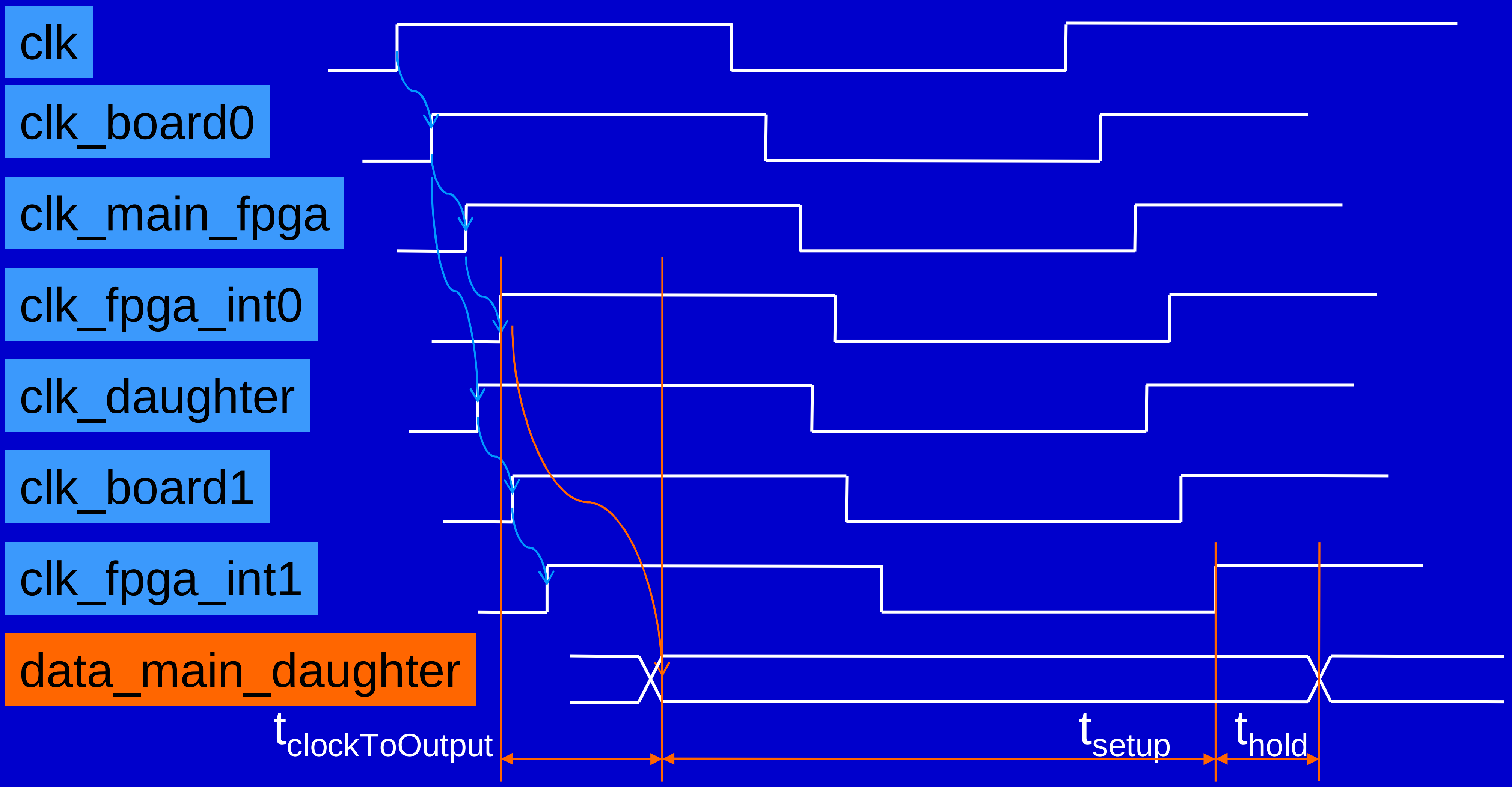
clock distribution/t_{co} & t_s /1-> 0



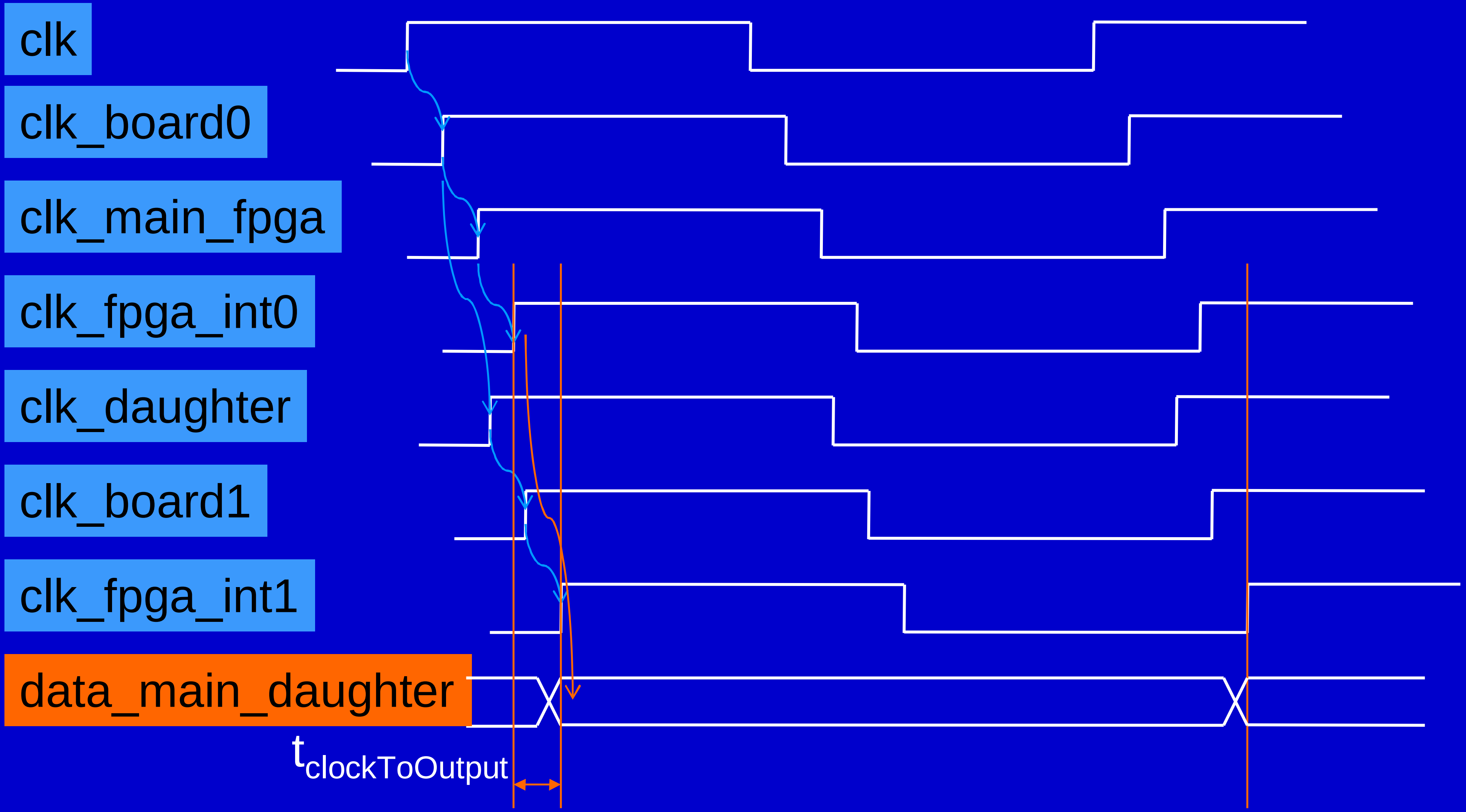
Clock distribution



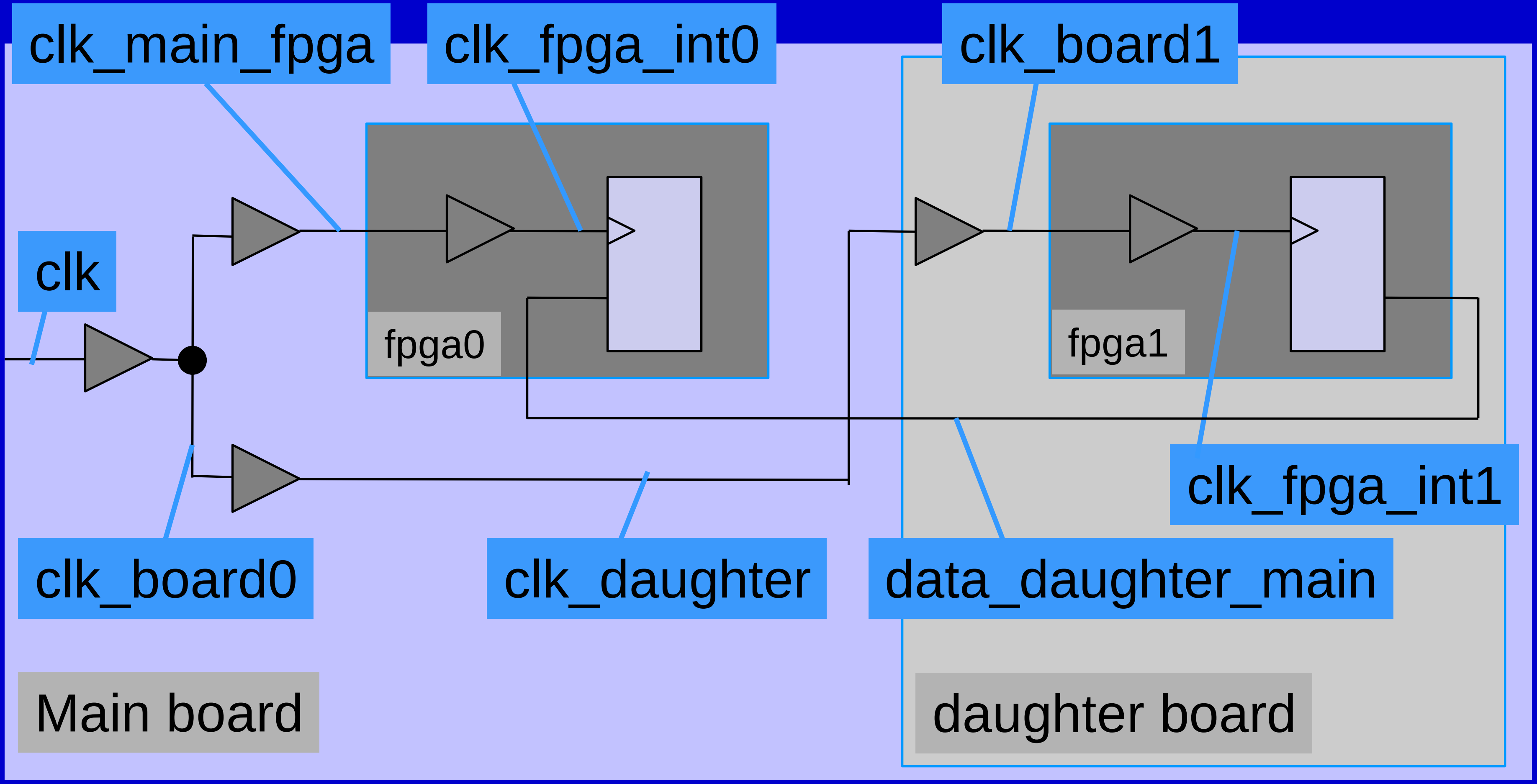
clock distribution/slow output 0->1



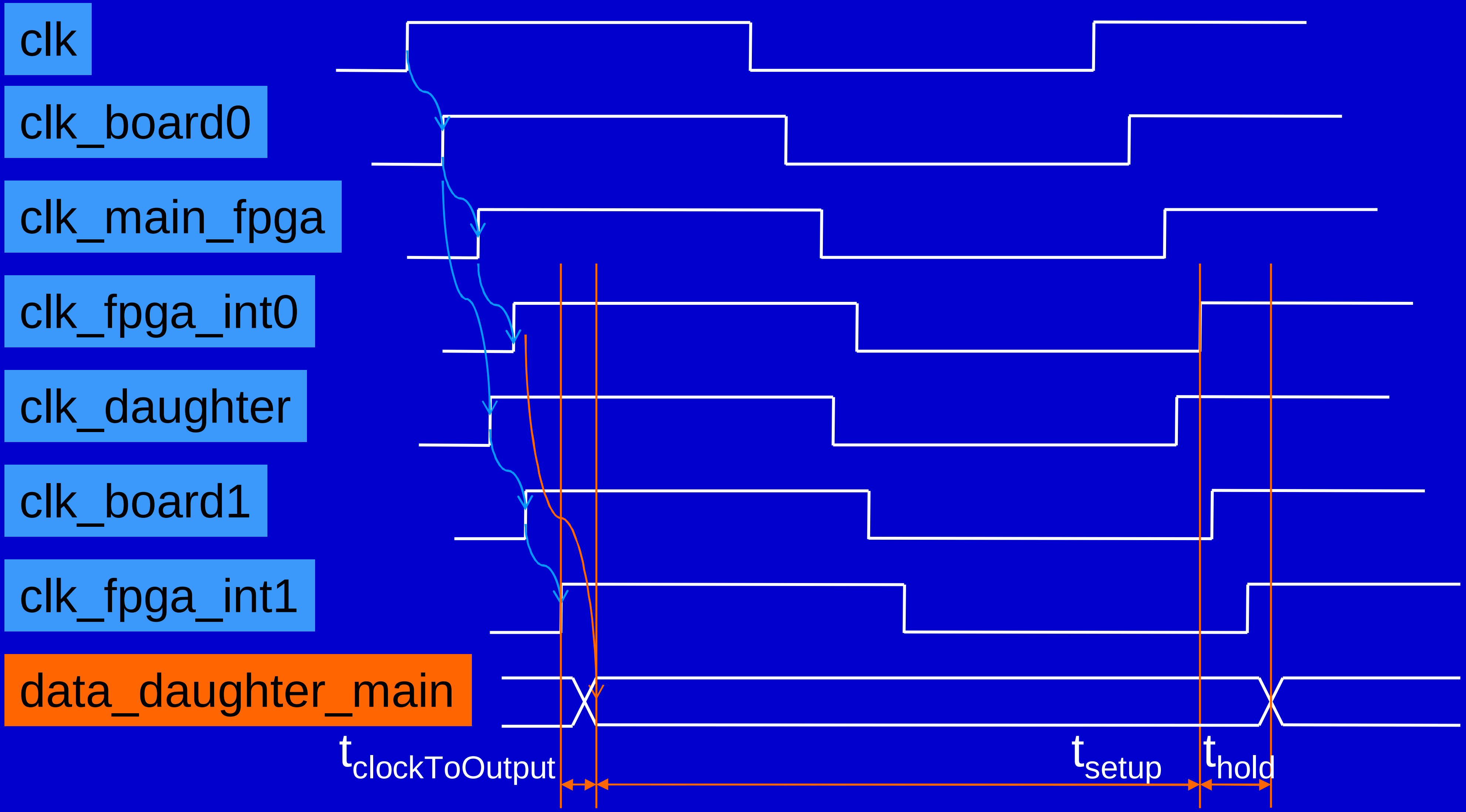
clock distribution/fast output 0->1



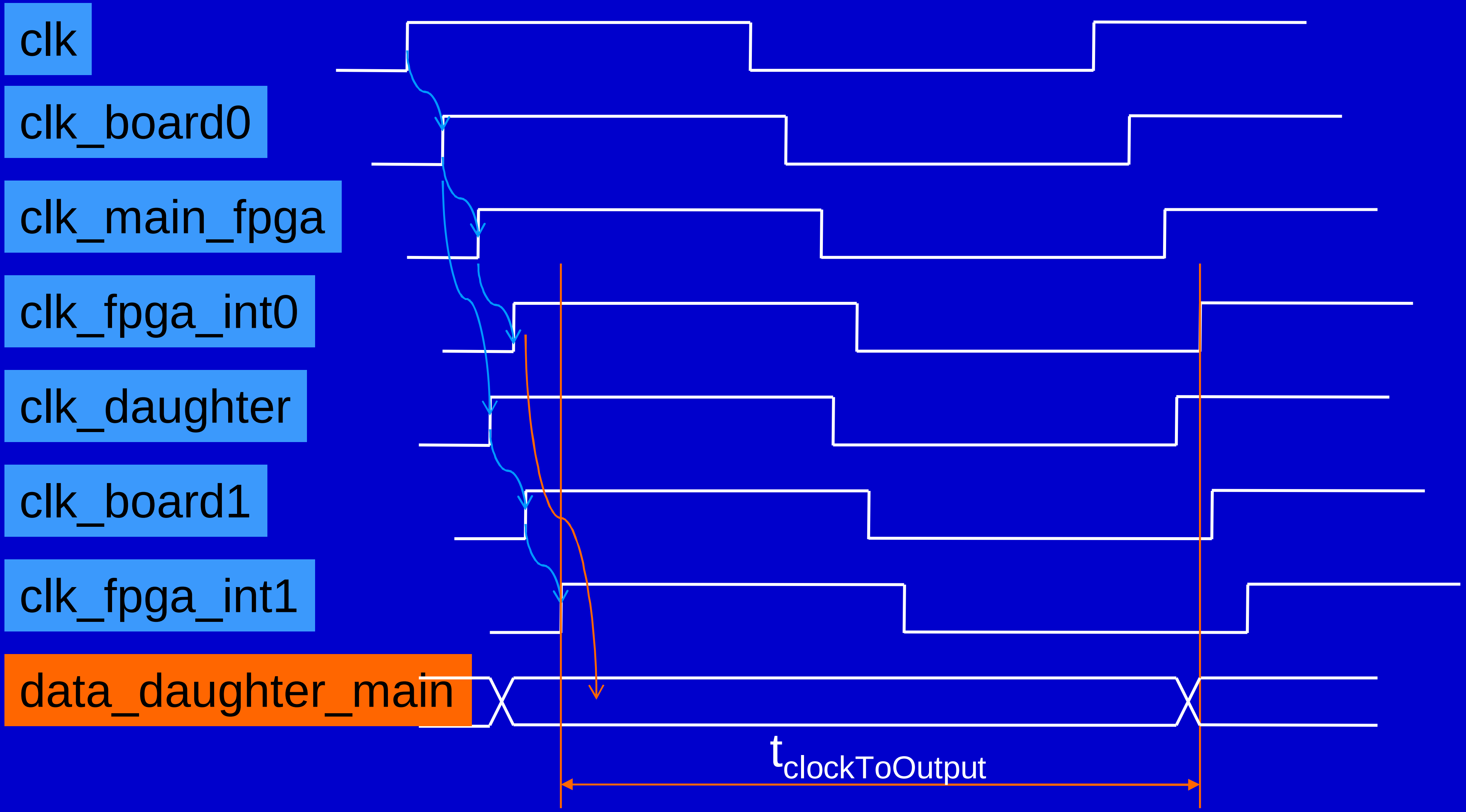
Clock distribution



clock distribution/fast output 1-> 0



clock distribution/slow output 1-> 0

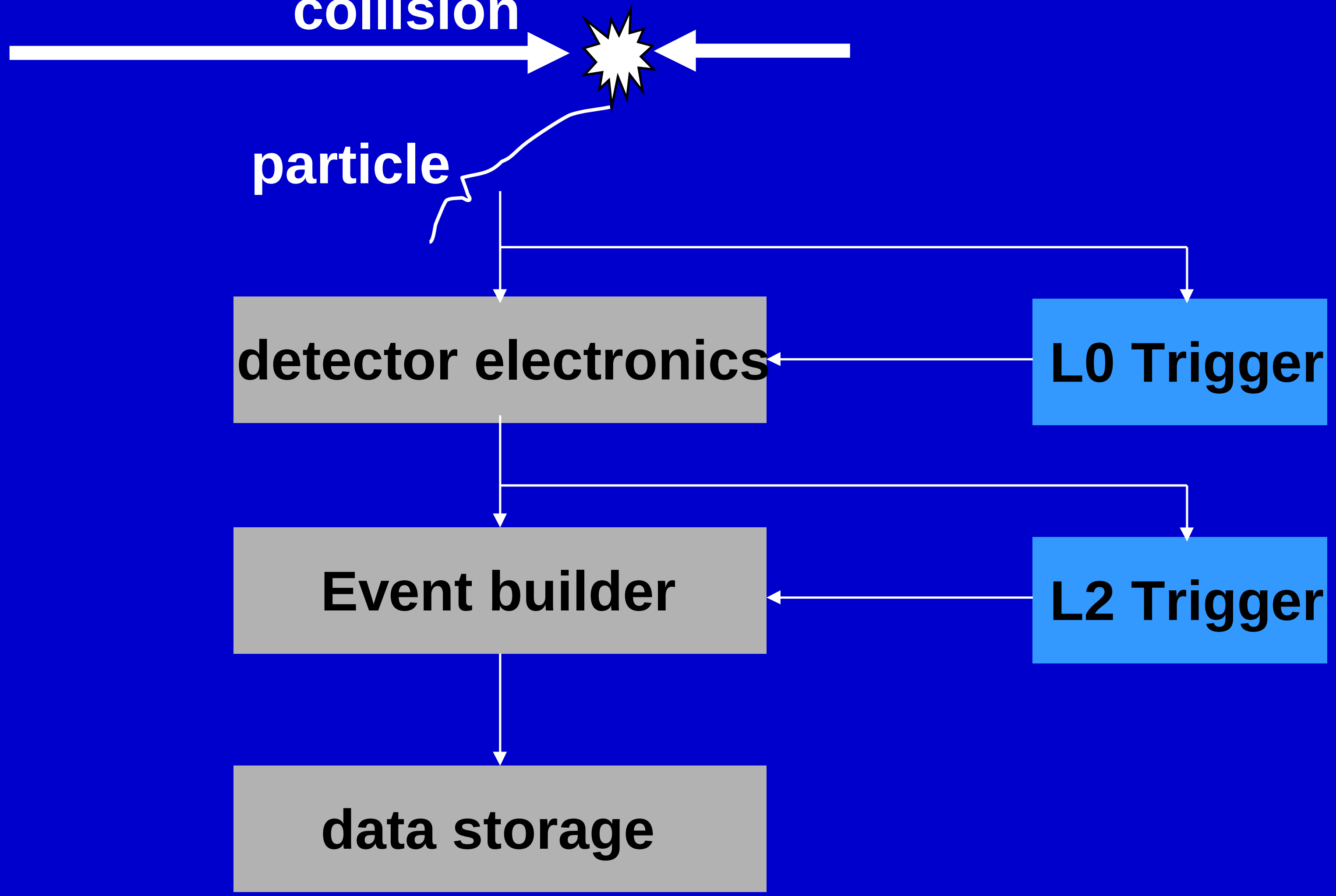


Constraints

- Fulfilling FPGA internal constraints is not sufficient.
- Perform system simulations
- Logic can be too fast

Data selection & delay

Data selection and delay



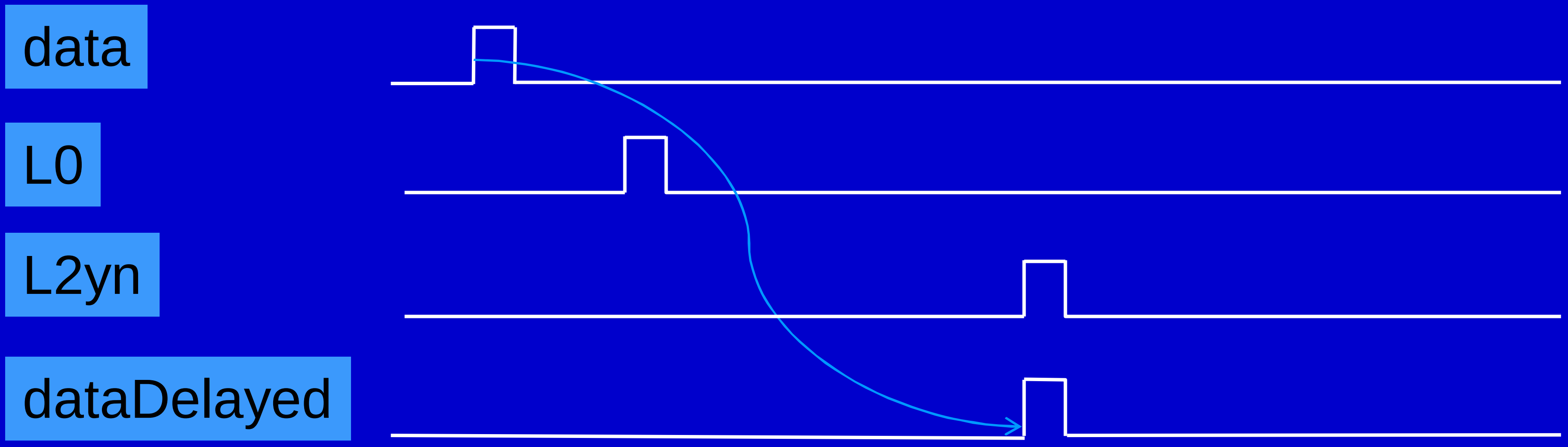
100 Tbyte/s

100 Gbyte/s

100 Mbyte/s

Data selection and delay

- Data (20 bits) every * 100 ns
- collision -> L0 (1µs)
- collision -> L2y or L2n (100 µs)



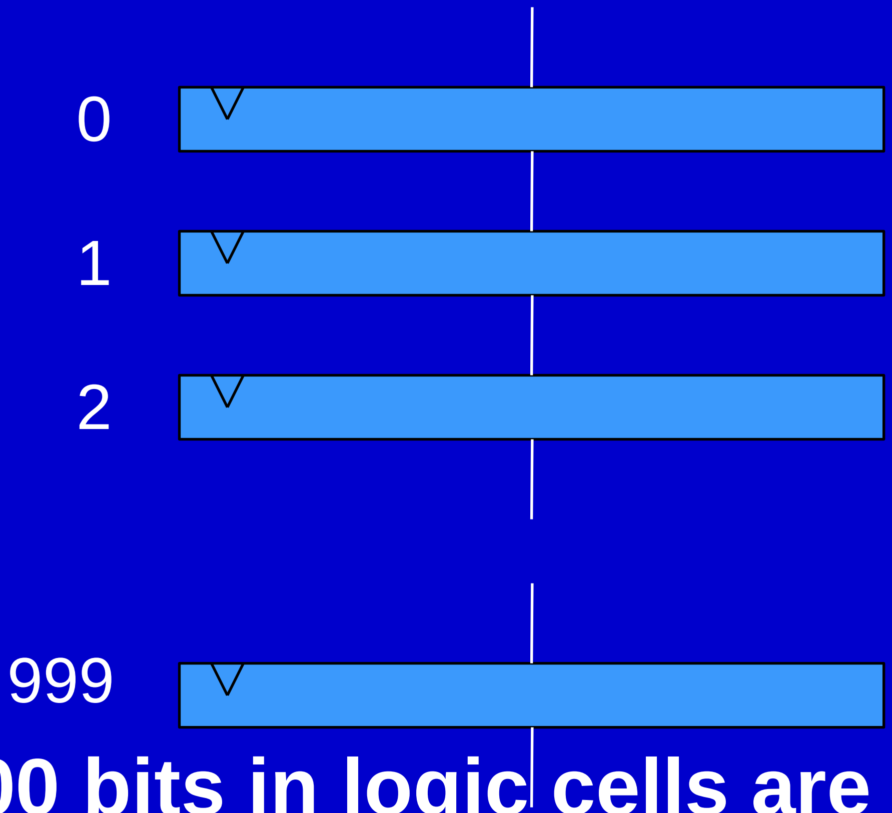
Data selection and delay

- **Data (20 bits) every * 100 ns**
- **collision -> L0 (1µs)**
- **collision -> L2y or L2n (100 µs)**

- **Options:**
  - Data pipeline with FIFO based on shift registers @ 10 MHz
    20 bits * 100 µs / 100 ns
    20 bits * 1000
    = 20 000 bits

Data selection and delay

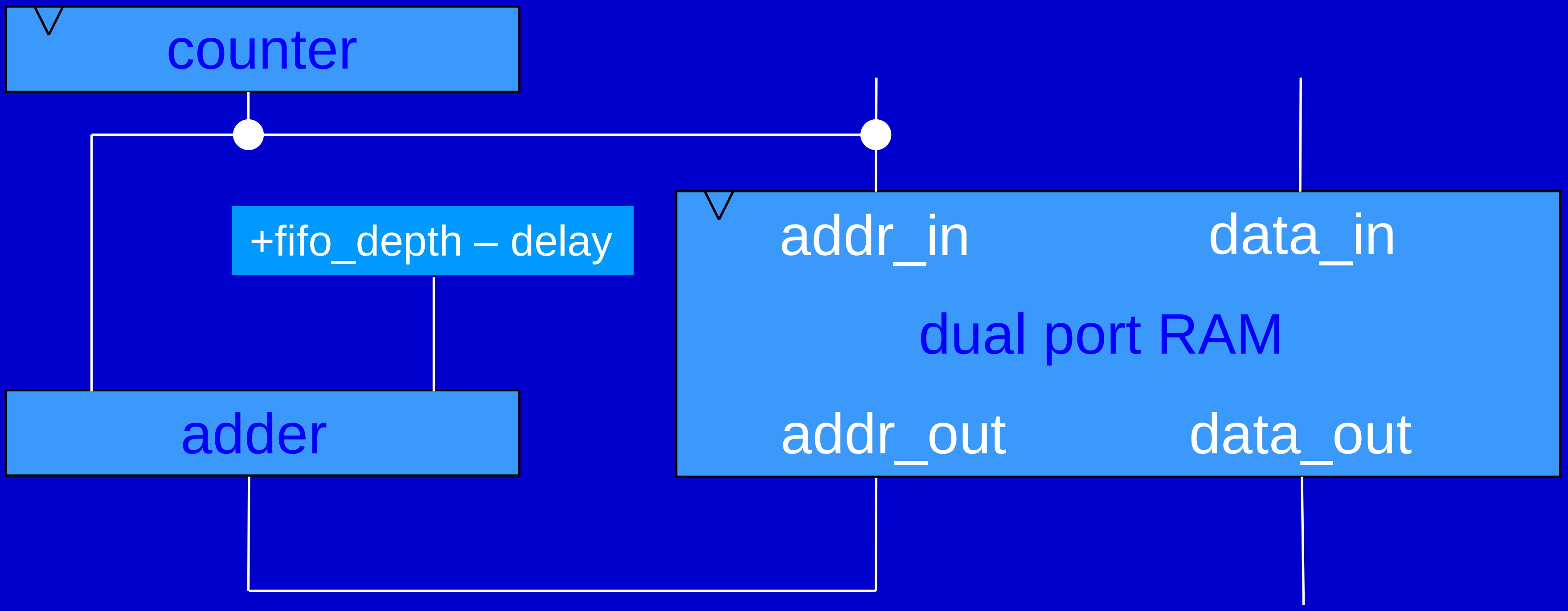
- **Data pipeline with FIFO with shift registers @ 10 MHz**
  20 bits * 1000 = 20 000 bits



20 000 bits in logic cells are used

Data selection and delay

- **Data pipeline with FIFO based on dual port RAM @ 10 MHz**
  20 bits * 1000 = 20 000 bits

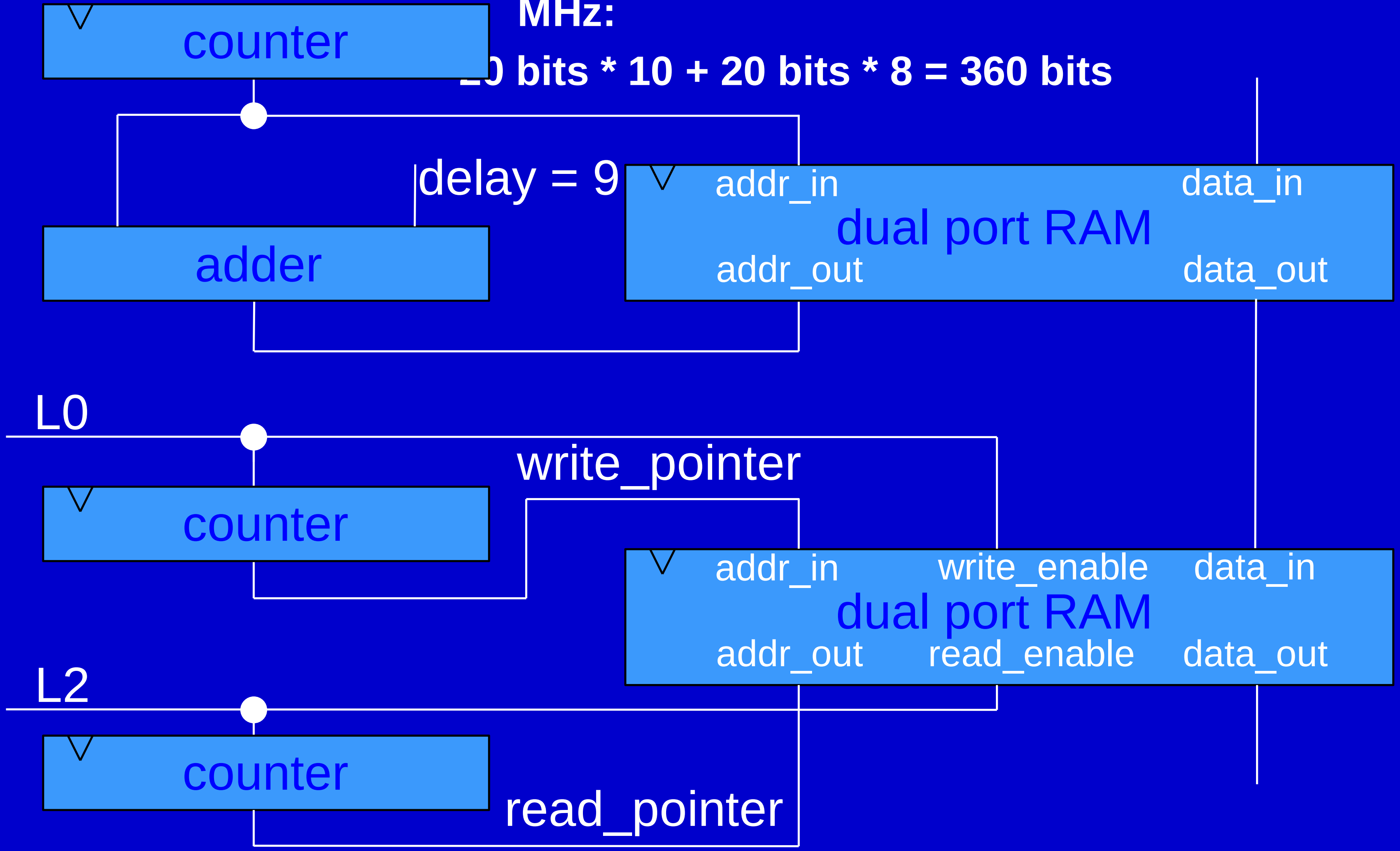


FPGAs have RAM cells in addition to logic blocks

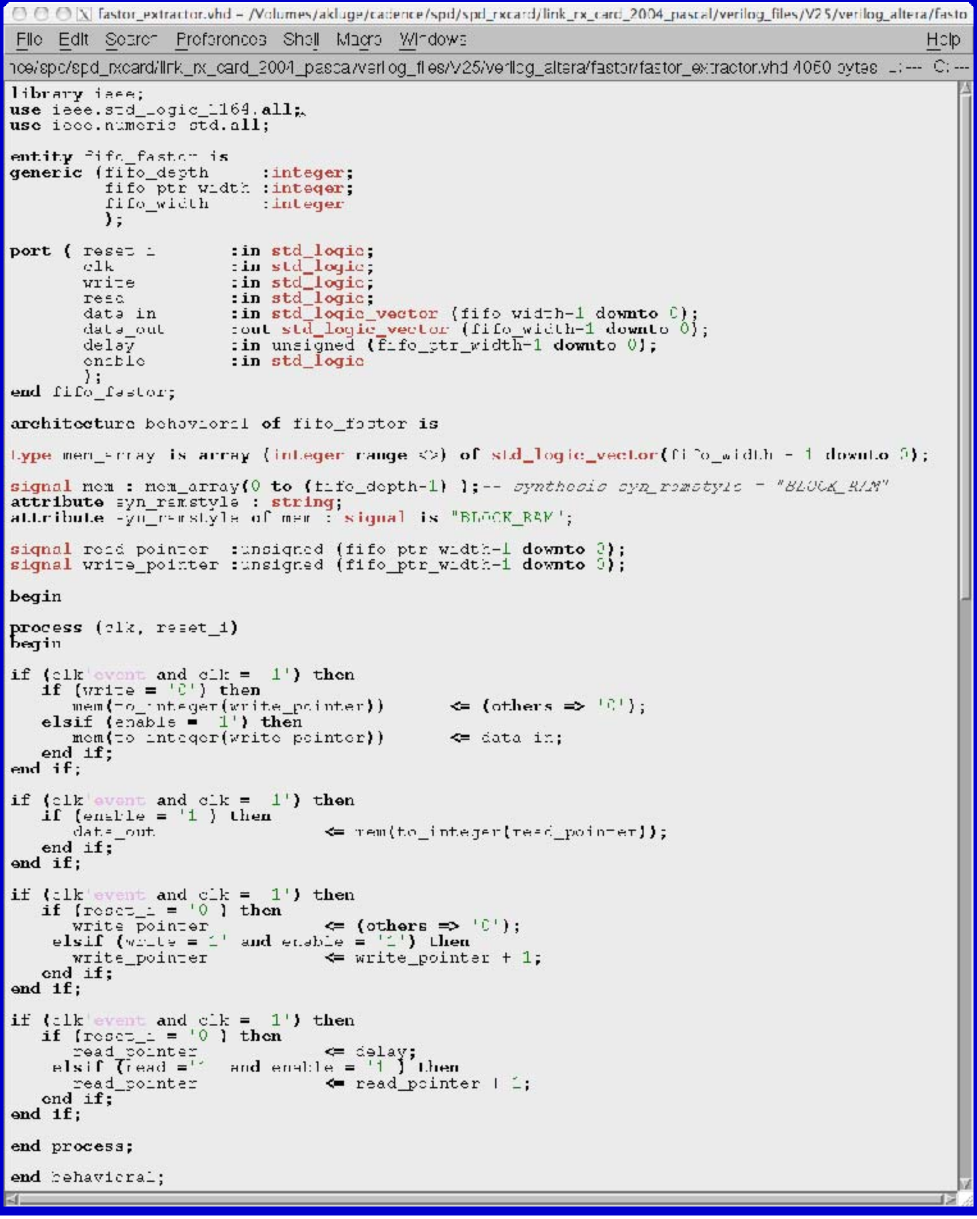
Data selection and delay

Data pipeline with 2 FIFOs based on dual port RAM@ 10 MHz:

20 bits * 10 + 20 bits * 8 = 360 bits

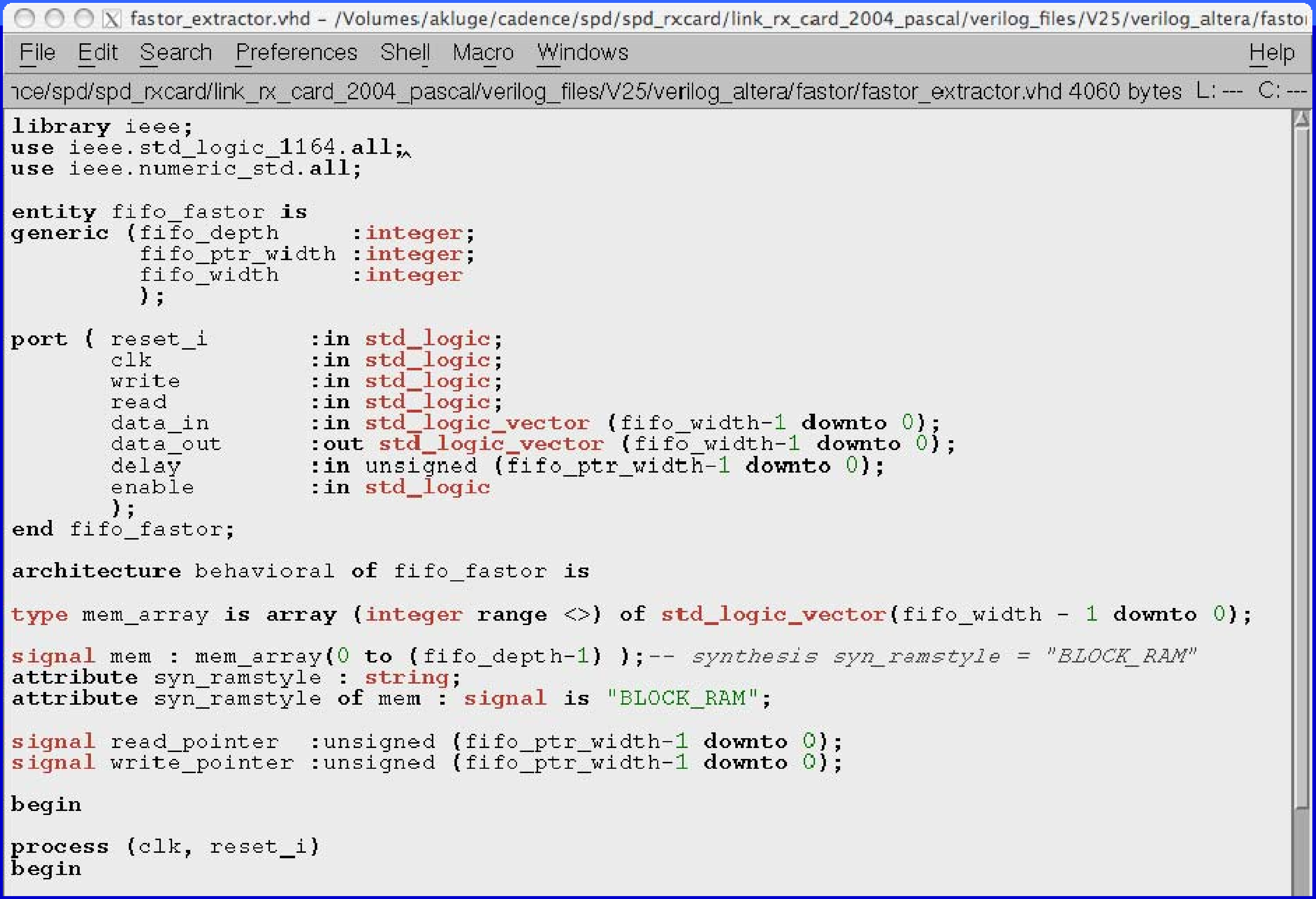


Data selection and delay


fastor_extractor.vhd – /Volumes/akluge/cadence/spd/spd_rxcard/link_rx_card_2004_pascal/verilog_files/V25/verilog_altera/fasto
File Edit Search Preferences Shell Macro Windows Help
nce/spd/spd_rxcard/link_rx_card_2004_pascal/verilog_files/V25/verilog_altera/fastor/fastor_extractor.vhd 4050 bytes

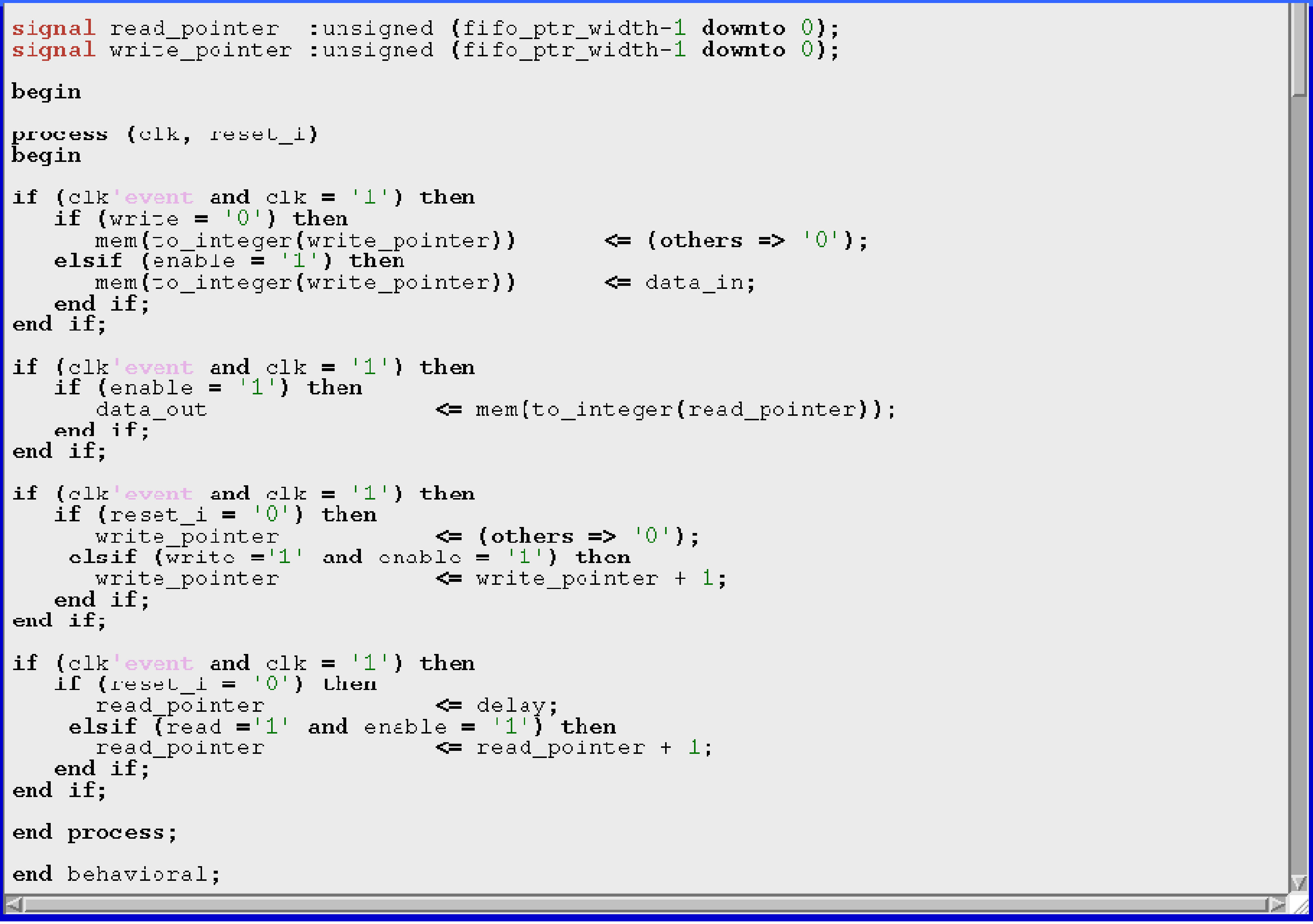
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fifo_fastor is
generic (fifo_depth      :integer;
         fifo_ptr_width  :integer;
         fifo_width      :integer
         );

port ( reset_i        :in std_logic;
       clk            :in std_logic;
       write          :in std_logic;
       read           :in std_logic;
       data_in        :in std_logic_vector (fifo_width-1 downto 0);
       data_out       :out std_logic_vector (fifo_width-1 downto 0);
       delay          :in unsigned (fifo_ptr_width-1 downto 0);
       enable         :in std_logic
       );
end fifo_fastor;

architecture behavioral of fifo_fastor is

type mem_array is array (integer range <>) of std_logic_vector(fifo_width - 1 downto 0);

signal mem : mem_array(0 to (fifo_depth-1) );-- synthesis syn_ramstyle = "BLOCK_RAM"
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "BLOCK_RAM";

signal read_pointer  :unsigned (fifo_ptr_width-1 downto 0);
signal write_pointer :unsigned (fifo_ptr_width-1 downto 0);

begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
   if (write = '0') then
      mem(to_integer(write_pointer))        <= (others => '0');
   elsif (enable = '1') then
      mem(to_integer(write_pointer))        <= data_in;
   end if;
end if;

if (clk'event and clk = '1') then
   if (enable = '1') then
      data_out                   <= mem(to_integer(read_pointer));
   end if;
end if;

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      write_pointer              <= (others => '0');
    elsif (write = '1' and enable = '1') then
      write_pointer              <= write_pointer + 1;
   end if;
end if;

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      read_pointer               <= delay;
    elsif (read = '1' and enable = '1') then
      read_pointer               <= read_pointer + 1;
   end if;
end if;

end process;

end behavioral;
```

Data selection and delay


fastor_extractor.vhd - /Volumes/akluge/cadence/spd/spd_rxcard/link_rx_card_2004_pascal/verilog_files/V25/verilog_altera/fasto
File Edit Search Preferences Shell Macro Windows Help
nce/spd/spd_rxcard/link_rx_card_2004_pascal/verilog_files/V25/verilog_altera/fastor/fastor_extractor.vhd 4060 bytes L: --- C: ---

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

entity fifo_fastor is
generic (fifo_depth     :integer;
         fifo_ptr_width :integer;
         fifo_width     :integer
         );

port ( reset_i          :in std_logic;
       clk              :in std_logic;
       write            :in std_logic;
       read             :in std_logic;
       data_in          :in std_logic_vector (fifo_width-1 downto 0);
       data_out         :out std_logic_vector (fifo_width-1 downto 0);
       delay            :in unsigned (fifo_ptr_width-1 downto 0);
       enable           :in std_logic
       );
end fifo_fastor;

architecture behavioral of fifo_fastor is

type mem_array is array (integer range <>) of std_logic_vector(fifo_width - 1 downto 0);

signal mem : mem_array(0 to (fifo_depth-1) );-- synthesis syn_ramstyle = "BLOCK_RAM"
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "BLOCK_RAM";

signal read_pointer  :unsigned (fifo_ptr_width-1 downto 0);
signal write_pointer :unsigned (fifo_ptr_width-1 downto 0);

begin

process (clk, reset_i)
begin
```

Data selection and delay



```
signal read_pointer  :unsigned (fifo_ptr_width-1 downto 0);
signal write_pointer :unsigned (fifo_ptr_width-1 downto 0);

begin

process (clk, reset_i)
begin

if (clk'event and clk = '1') then
   if (write = '0') then
      mem(to_integer(write_pointer))      <= (others => '0');
   elsif (enable = '1') then
      mem(to_integer(write_pointer))      <= data_in;
   end if;
end if;

if (clk'event and clk = '1') then
   if (enable = '1') then
      data_out                      <= mem(to_integer(read_pointer));
   end if;
end if;

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      write_pointer                 <= (others => '0');
    elsif (write ='1' and enable = '1') then
      write_pointer                 <= write_pointer + 1;
   end if;
end if;

if (clk'event and clk = '1') then
   if (reset_i = '0') then
      read_pointer                  <= delay;
    elsif (read ='1' and enable = '1') then
      read_pointer                  <= read_pointer + 1;
   end if;
end if;

end process;

end behavioral;
```

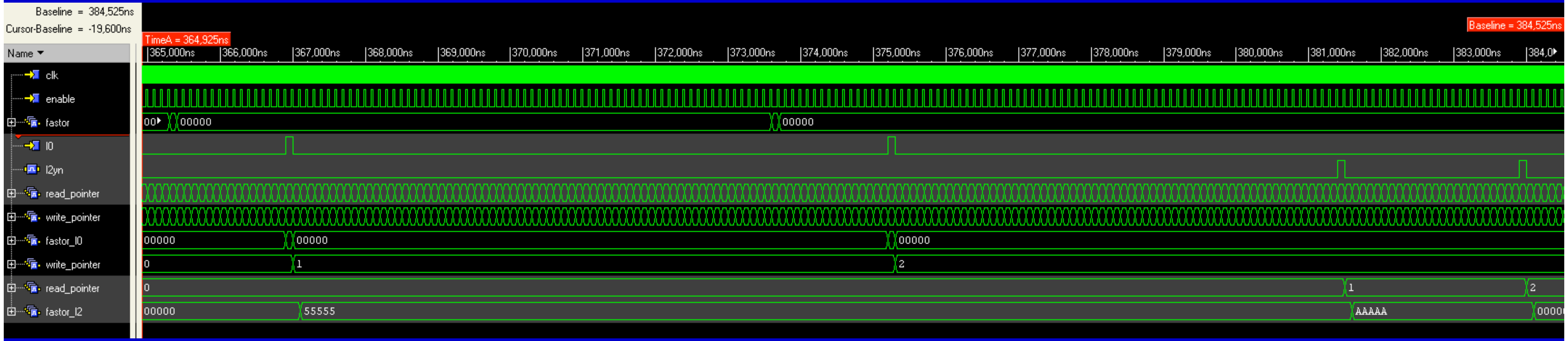
Data selection and delay

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

entity fastor_extractor is
generic (fifo_depth      :integer := 16;
         fifo_ptr_width :integer := 4;
         fifo_width      :integer := 20
         );

port ( reset_i           :in std_logic;
       clk               :in std_logic;
       fastor0           :in std_logic_vector (9 downto 0);
       fastor1           :in std_logic_vector (9 downto 0);
       l0                :in std_logic   :='0';
       l2y               :in std_logic   :='0';
       l2n               :in std_logic   :='0';
       delay_l0          :in unsigned (3 downto 0) := "1111";
       fastor_delayed0   :out std_logic_vector (9 downto 0);
       fastor_delayed1   :out std_logic_vector (9 downto 0);
       enable            :in std_logic
       );
end fastor_extractor;

architecture behavioral of fastor_extractor is

component fifo_fastor is
generic (fifo_depth      :integer;
         fifo_ptr_width :integer;
         fifo_width      :integer
         );
port ( reset_i           :in std_logic;
       clk               :in std_logic;
       write             :in std_logic;
       read              :in std_logic;
       data_in           :in std_logic_vector (fifo_width-1 downto 0);
       data_out          :out std_logic_vector (fifo_width-1 downto 0);
       delay             :in unsigned (fifo_ptr_width-1 downto 0);
       enable            :in std_logic
       );
end component;

signal fastor            :std_logic_vector (fifo_width-1 downto 0);
signal fastor_l0         :std_logic_vector (fifo_width-1 downto 0);
signal fastor_l2         :std_logic_vector (fifo_width-1 downto 0);
signal l2yn              :std_logic;
```

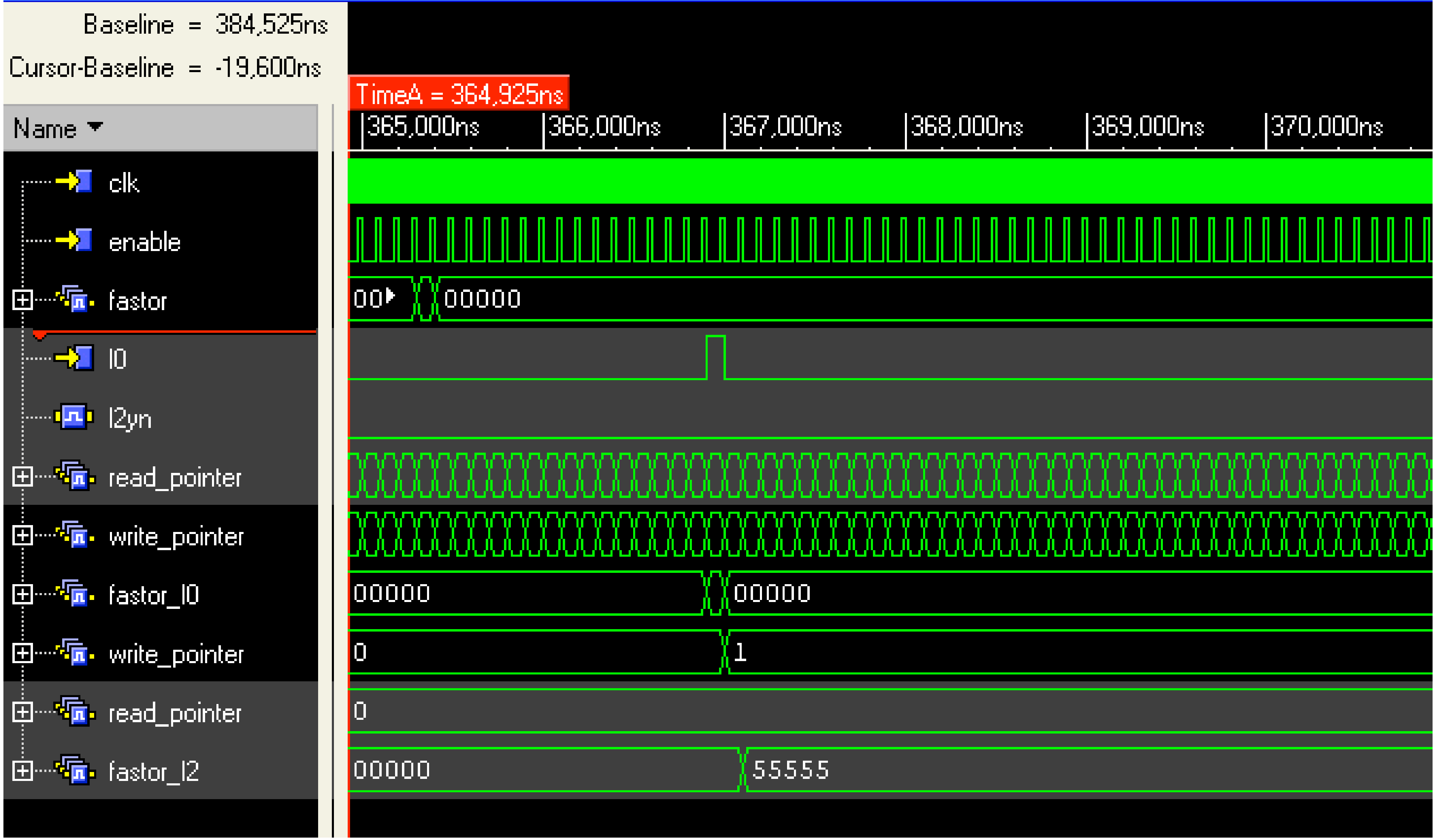
Data selection and delay

```
begin

fastor (19 downto 10)    <= fastor1;
fastor (9 downto 0)      <= fastor0;
l2yn                     <= l2y or l2n;
fastor_delayed1          <= fastor_l2(19 downto 10);
fastor_delayed0          <= fastor_l2(9 downto 0);

fifo_fastor_l0:   fifo_fastor generic map(fifo_depth,fifo_ptr_width,fifo_width)
                              port map (reset_i,
                                       clk       => clk,
                                       write     => '1',
                                       read      => '1',
                                       data_in   => fastor,
                                       data_out  => fastor_l0,
                                       delay     => delay_l0,
                                       enable    => enable
                                       );

fifo_fastor_l2:   fifo_fastor generic map(4,2,20)
                              port map (reset_i,
                                       clk       => clk,
                                       write     => l0,
                                       read      => l2yn,
                                       data_in   => fastor_l0,
                                       data_out  => fastor_l2,
                                       delay     => (others => '0'),
                                       enable    => enable
                                       );
end behavioral;
```

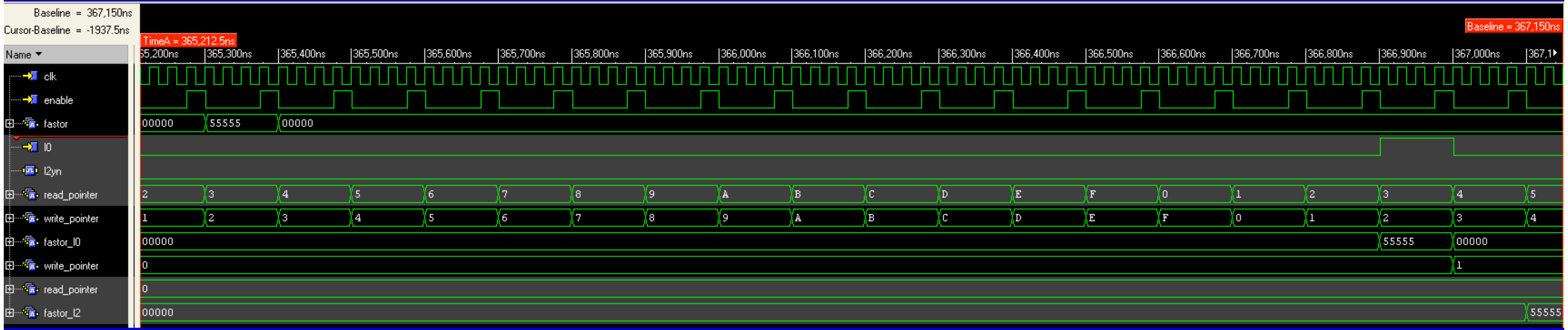
Data selection and delay



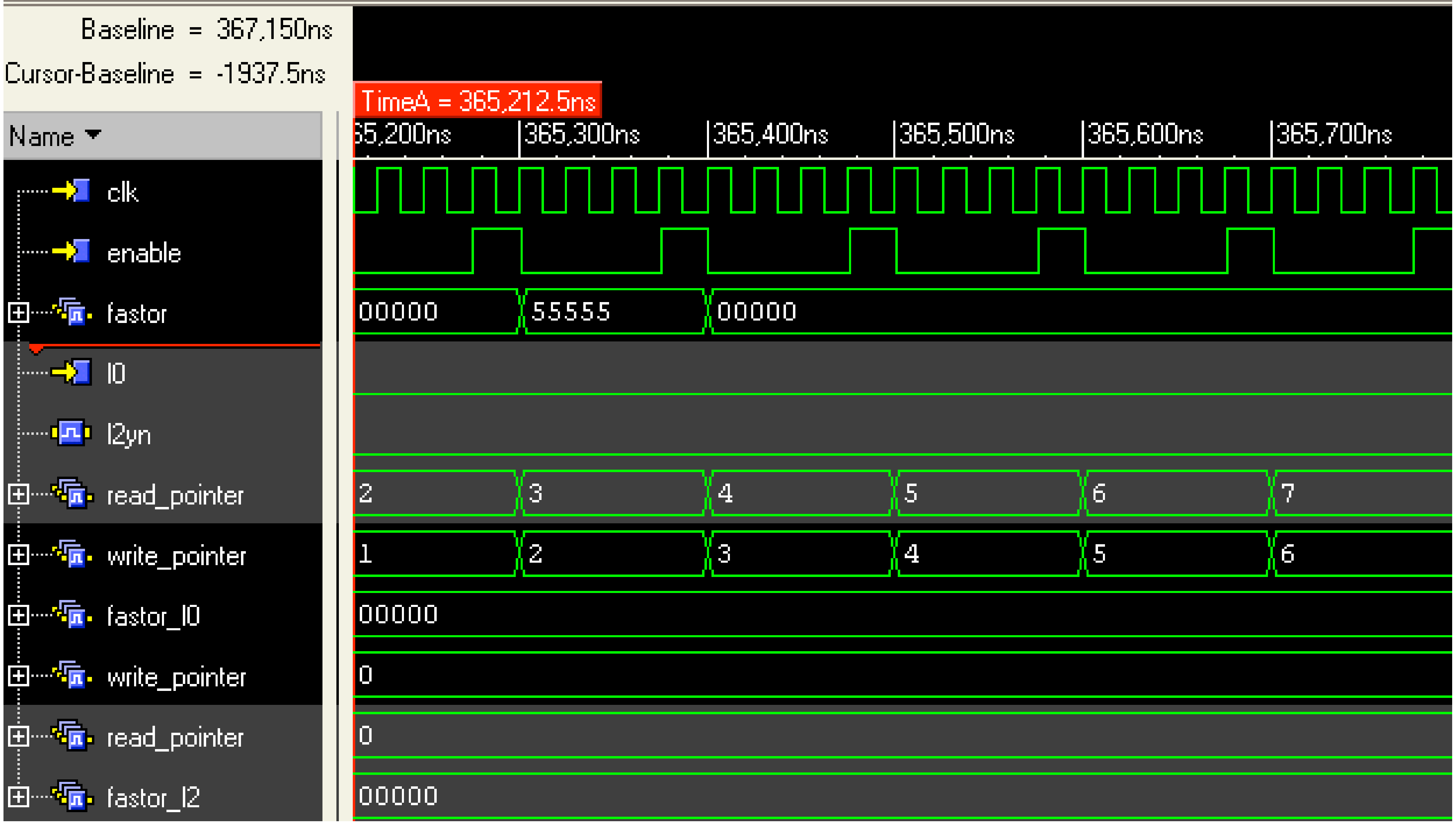
Data selection and delay



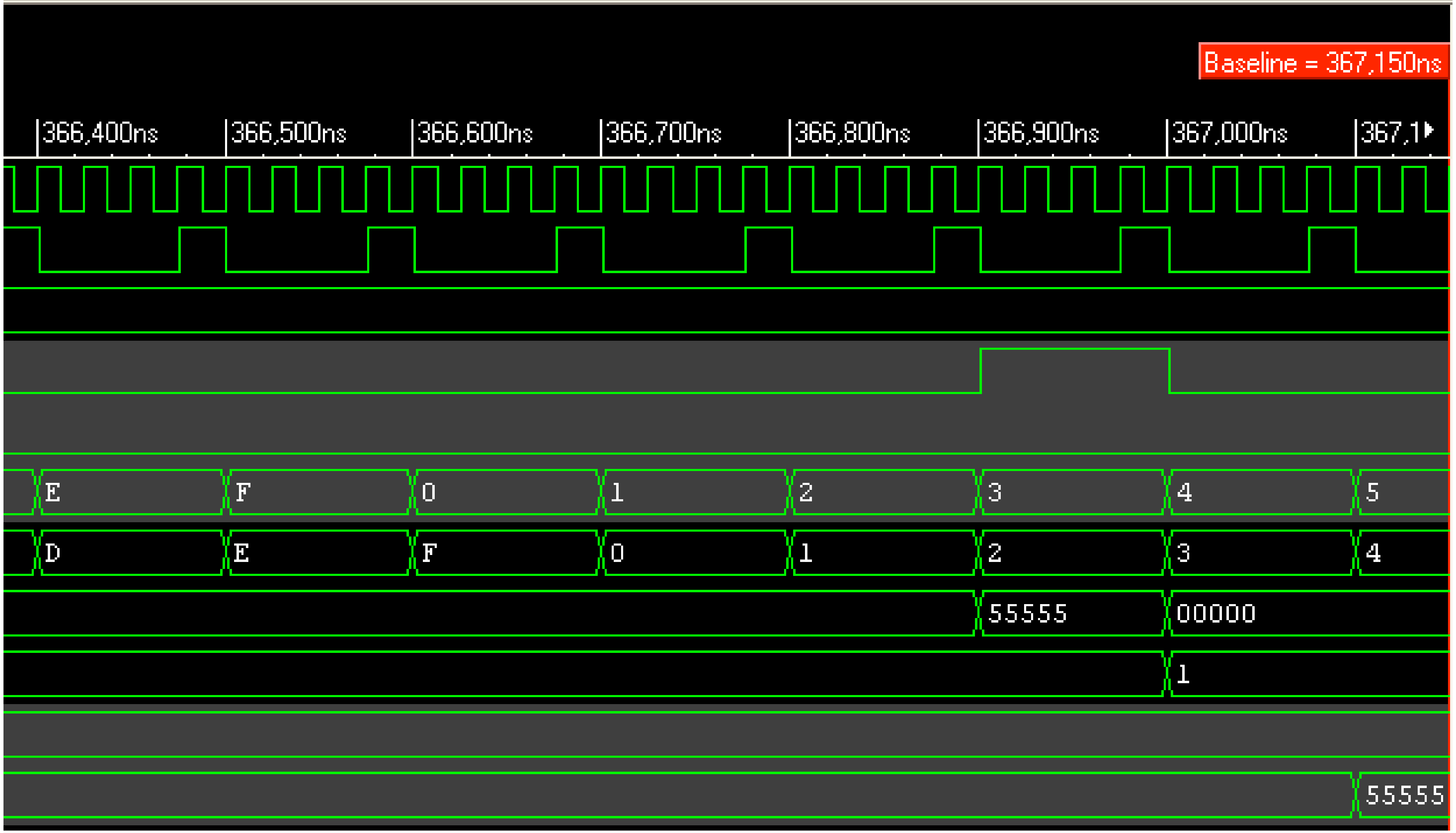
Data selection and delay



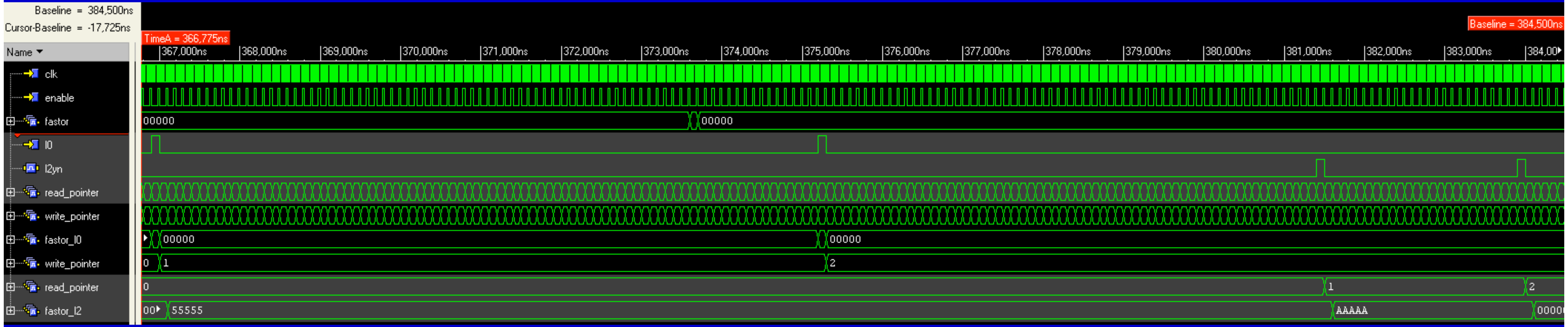
Data selection and delay



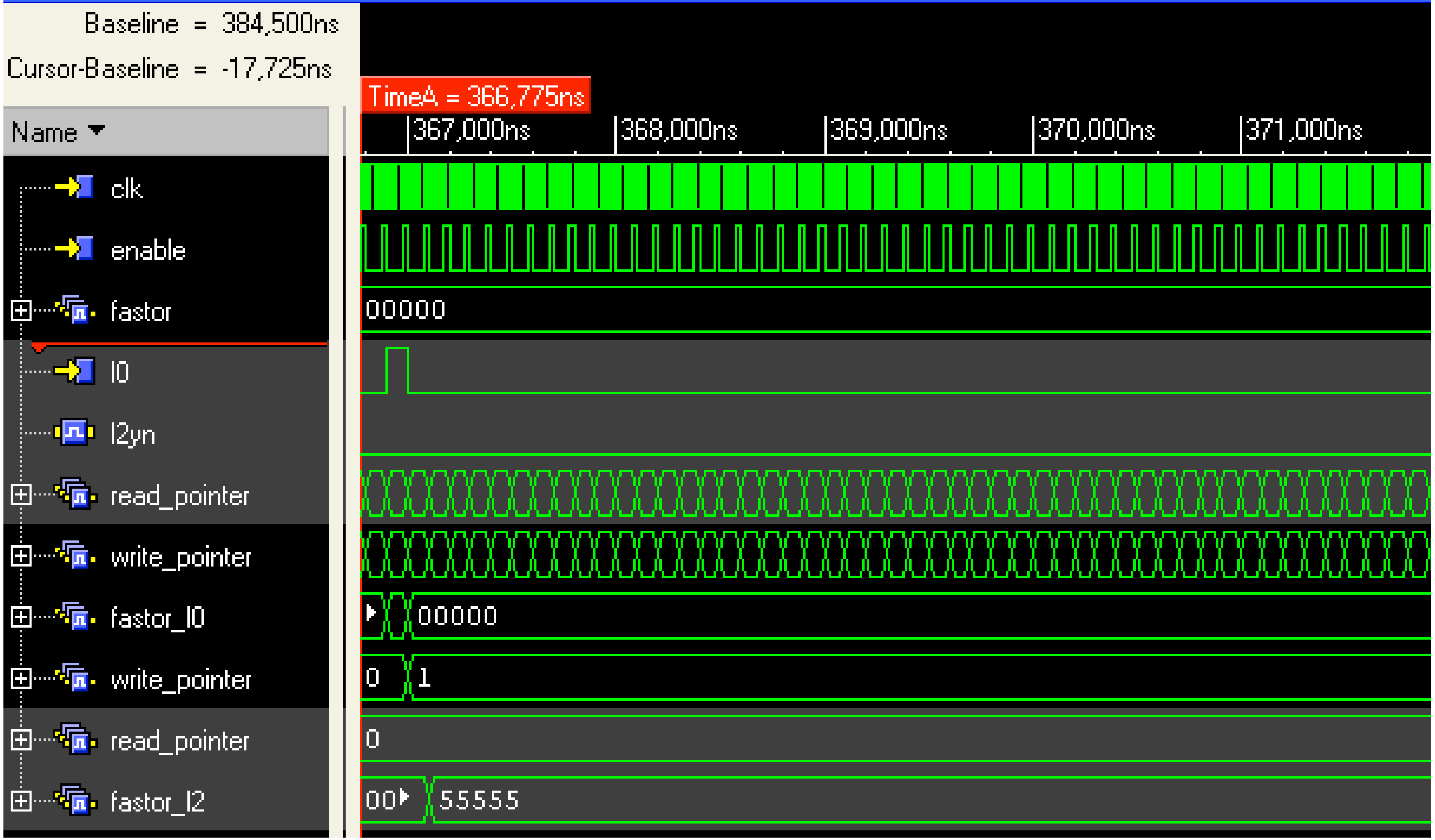
Data selection and delay



Data selection and delay



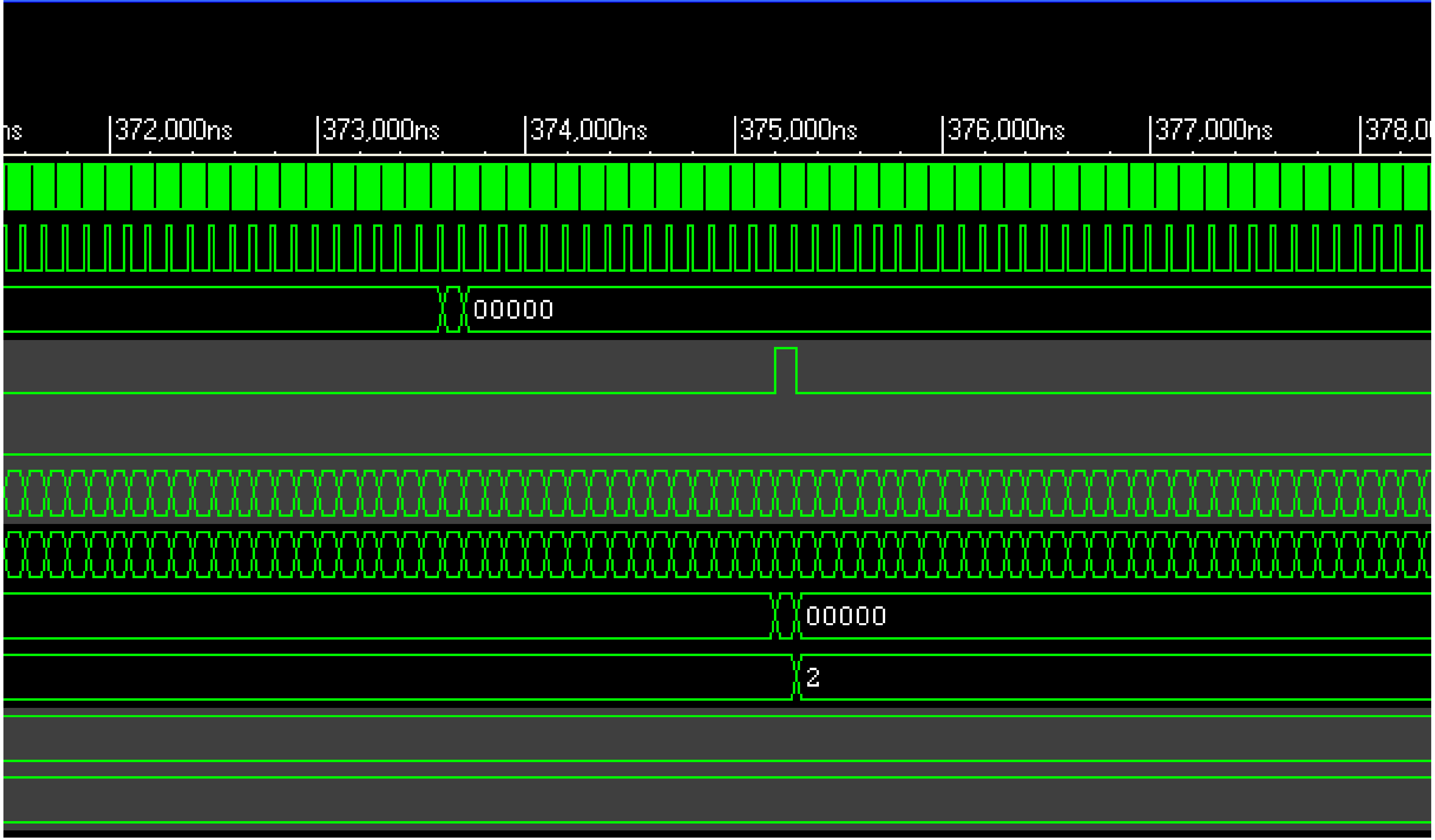
Data selection and delay



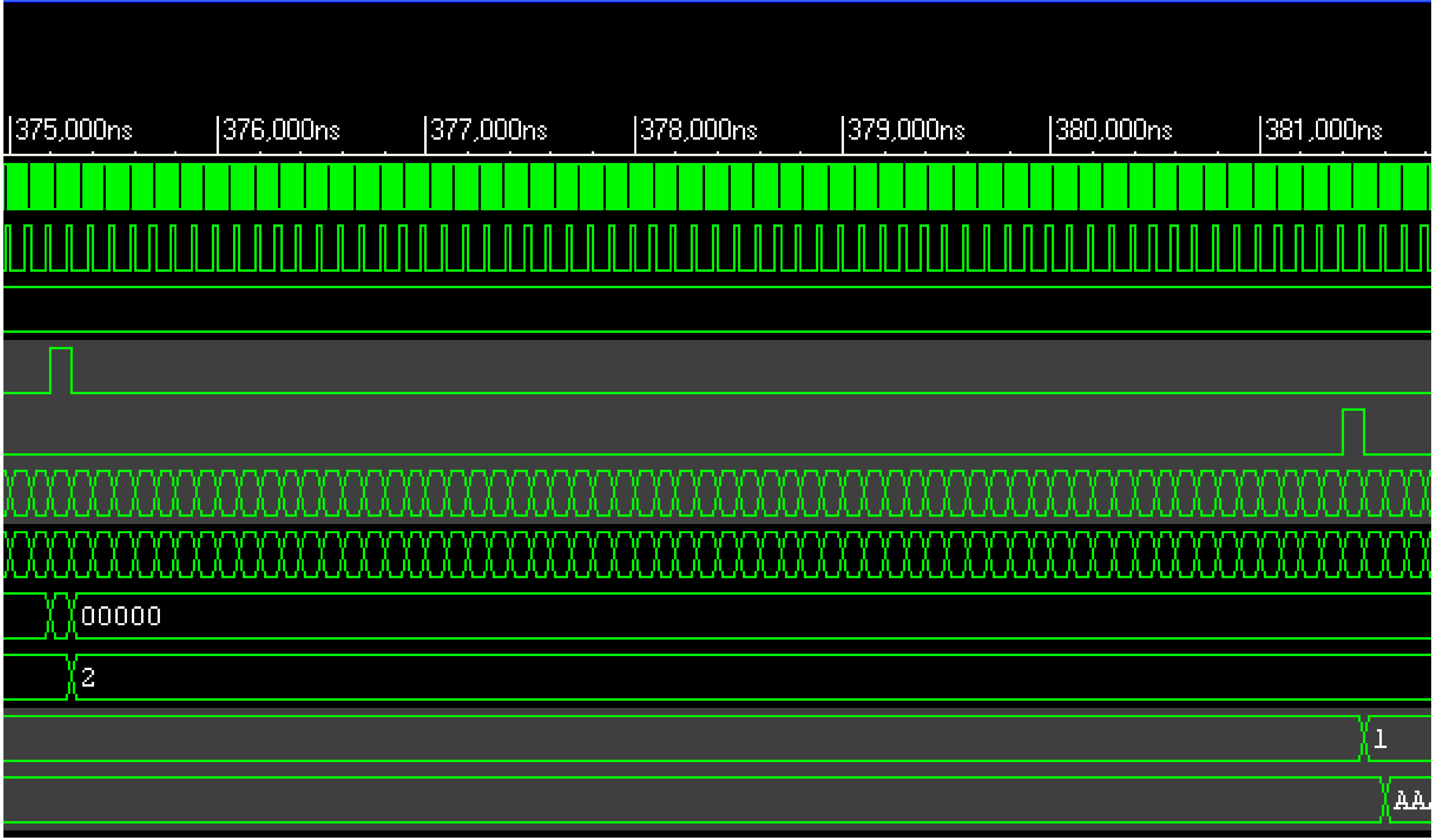
Baseline = 384,500ns
Cursor-Baseline = -17,725ns

TimeA = 366,775ns

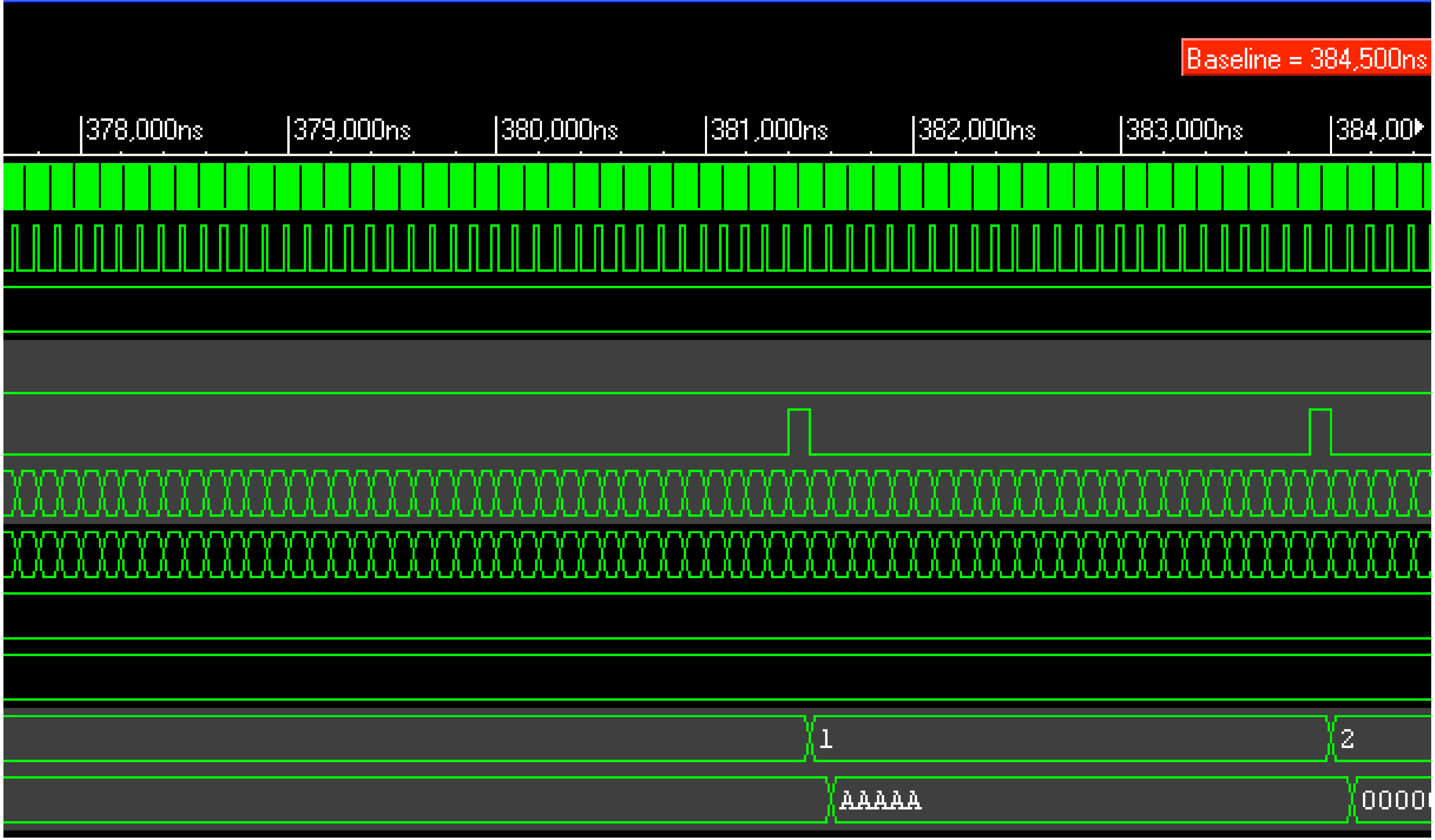
Data selection and delay



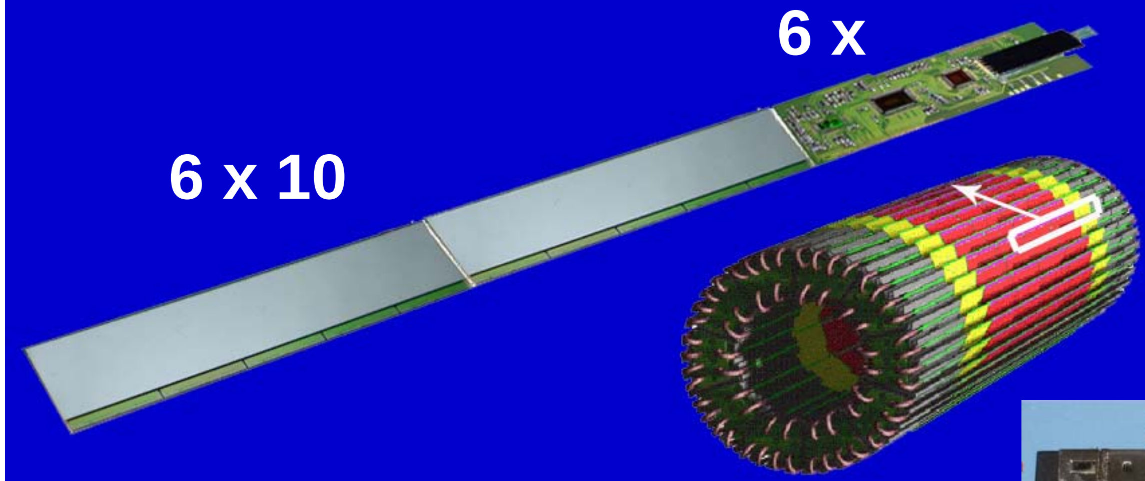
Data selection and delay

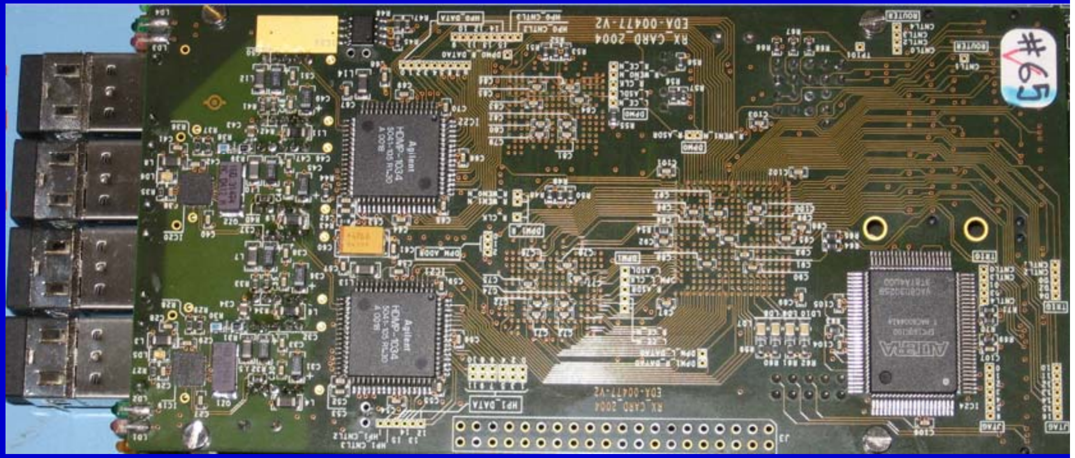


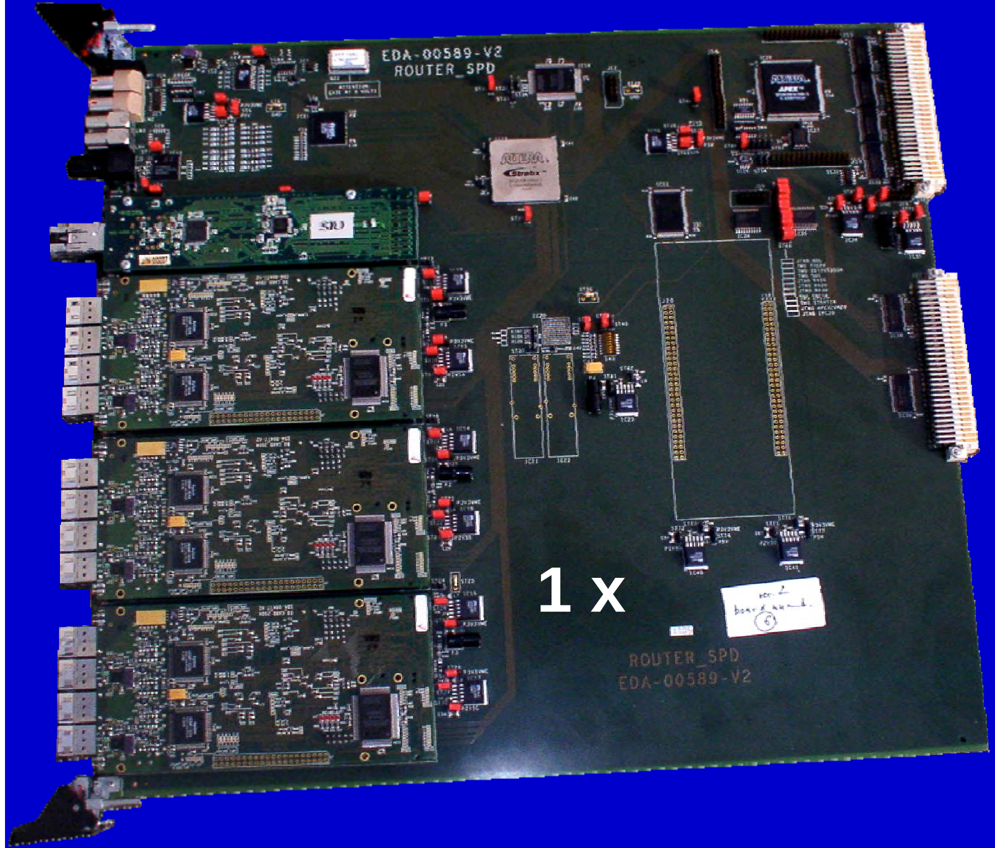
Data selection and delay



System level simulation







- 60 ASICs: simplified behavioral
- 40 ASICs: full behavioral
- 5 FPGA: full behavioral
- 7 SRAMs: full behavioral
- 4 PCBs

Conclusion

- **What happens if we have speed problems:**
  - Often because of inadequate logic architecture/coding style
    - evaluate logic architecture
    - rewrite HDL code to adapt structure to better data throughput
    - insert pipeline structure - often one clock cycle more latency does not matter
    - Understand the specifications
    - look for systematics which can help to simplify logic
    - adapt architecture and schematics/code
    - only then optimize placing & routing

Conclusion

- **What happens if we have speed problems:**
  - Often because of components too small and routing congestion
    - timing constraints
    - Routing constraint - placement constraint
    - Use bigger/faster component

Conclusion

- **FPGA application at CERN**
  - data selection/trigger (muon track finder trigger)
  - data processing (pixel detector)
- **Design cycle**
- **Defining Specifications**
- **Clock domains**
- **Data delay**