2177-20

ICTP Latin-American Basic Course on FPGA Design for Scientific Instrumentation

15 - 31 March 2010

Advanced FPGA Applications

KLUGE Alexander
CERN
Geneva
Switzerland

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

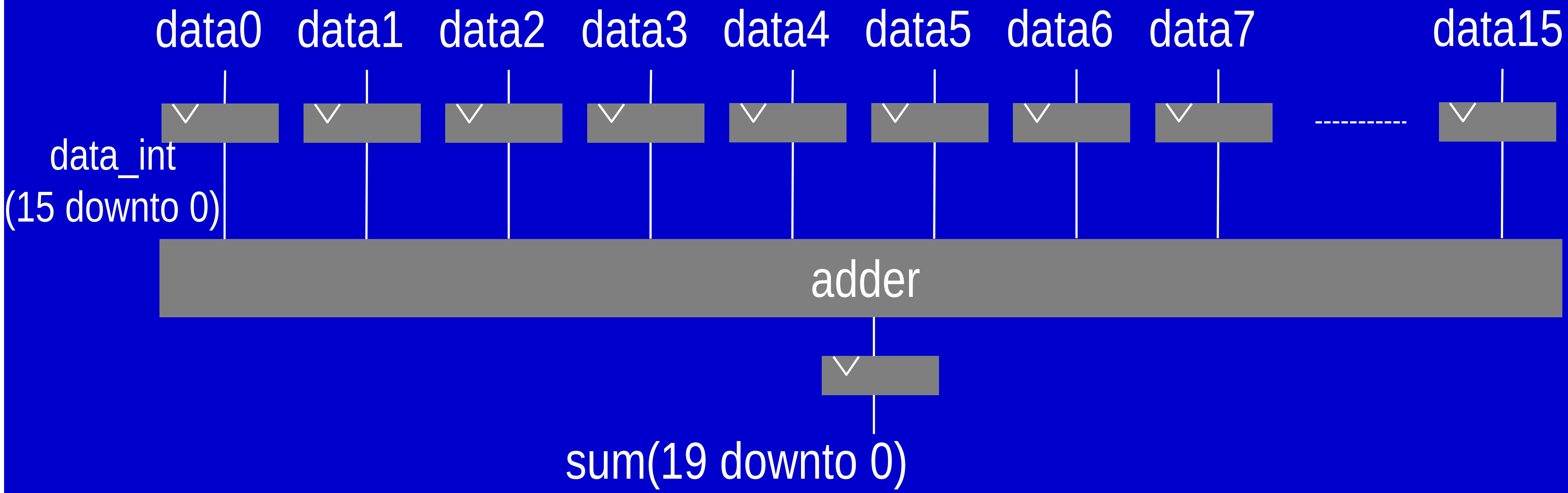
FPGA specifications

- Costumer/boss says:
  “I need a system which can calculate the value each 25 ns.”

- What you might understand is:
  “The calculation needs to be finished within 25 ns”

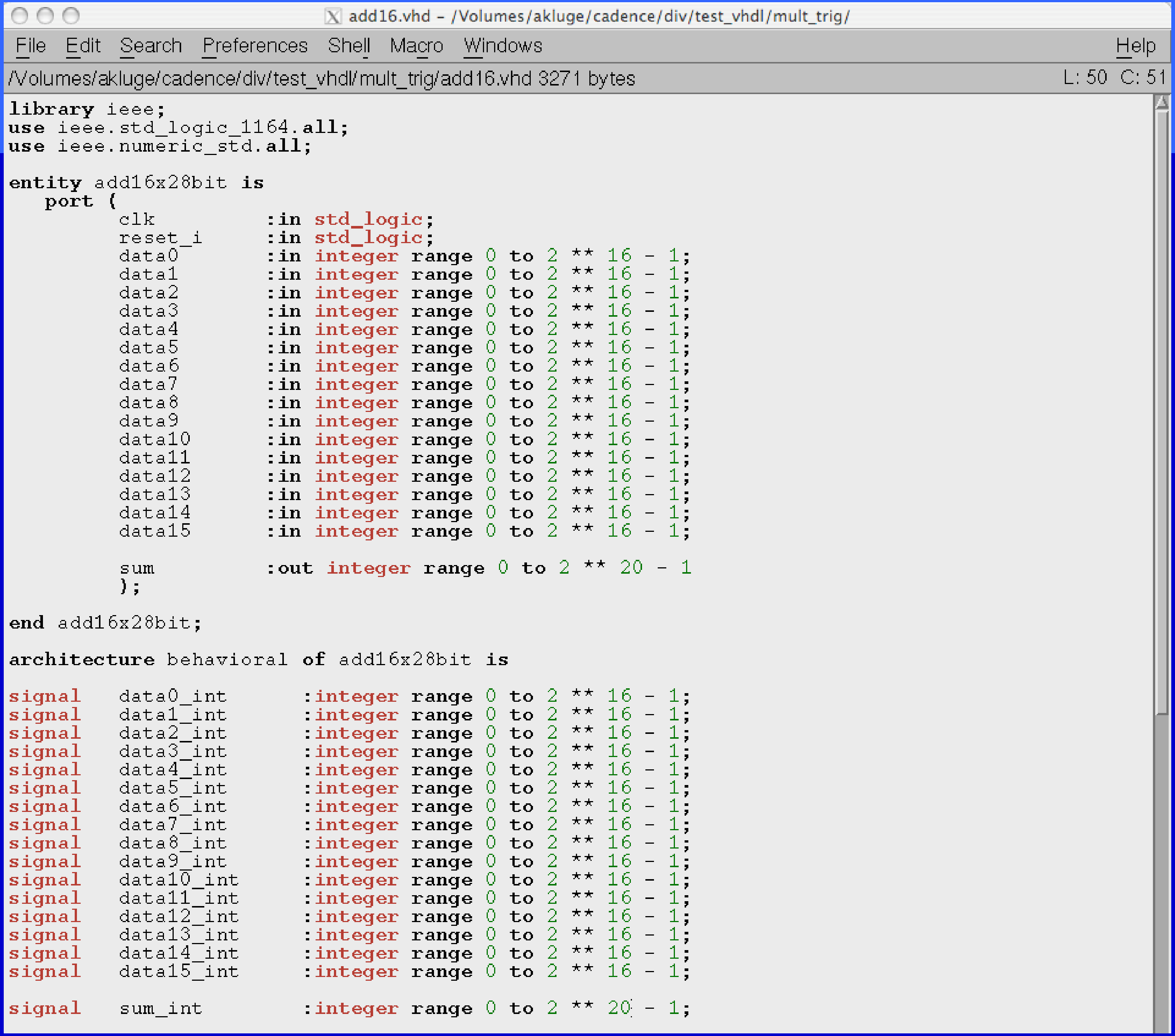
Adder

- **Example:**
  - add 16 16-bit values in 25 ns



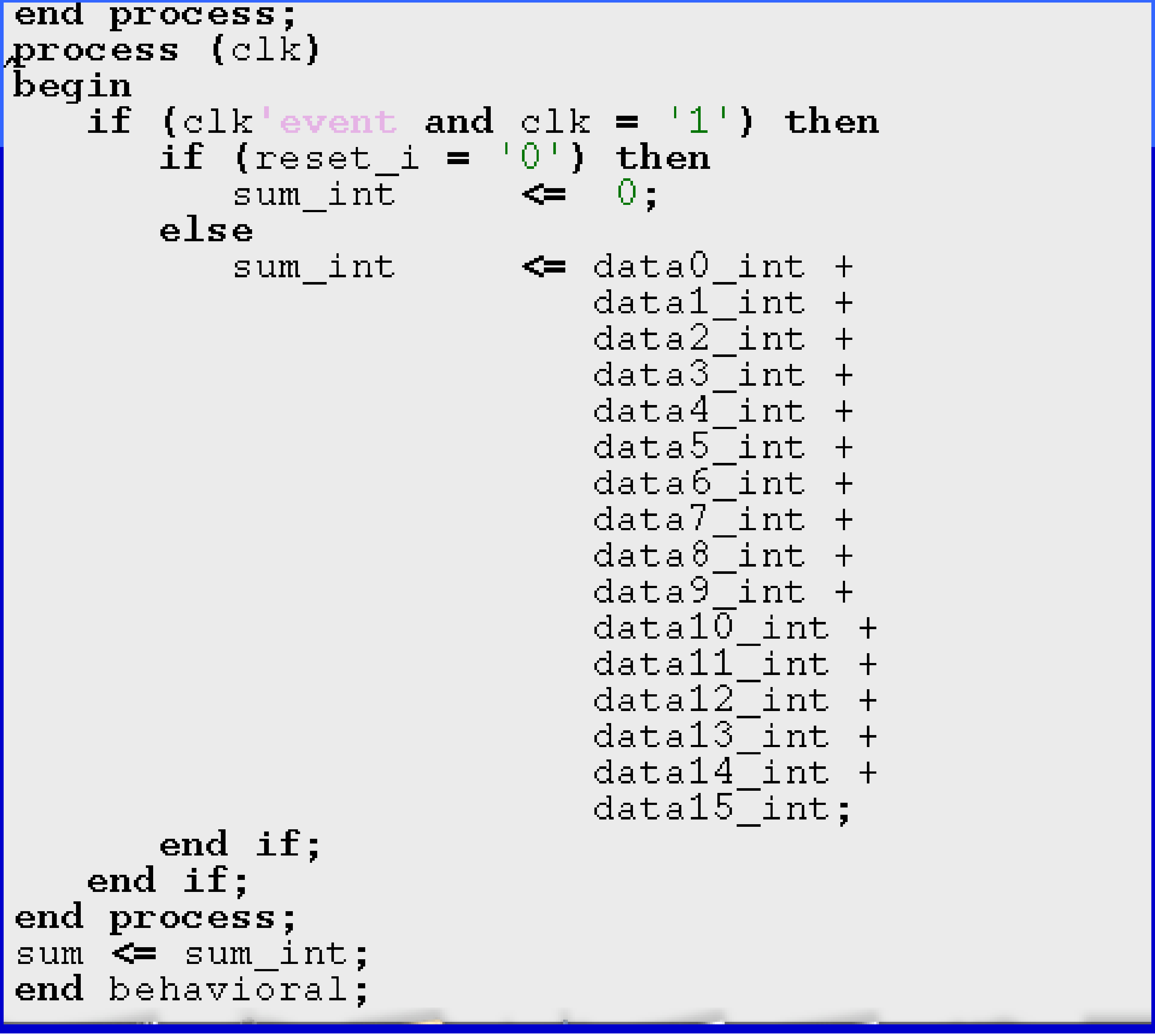
24

20


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

```
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)
```



```
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

- 33.6 ns > 25 ns -> too slow

Adder

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

- 33.6 ns > 25 ns -> too slow

- Ask boss to buy faster, more expensive FPGA
- Work (manually) on FPGA placing&routing
- Help synthesizer to make fater adder
- Ask whether you have understood specification

FPGA specifications

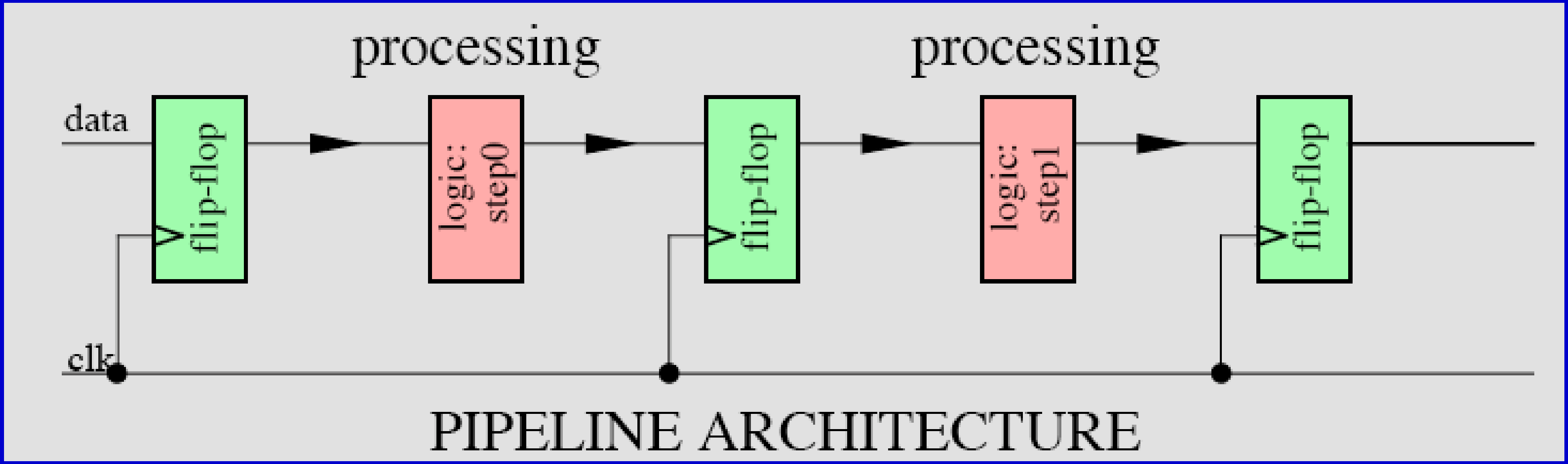
- Costumer/boss says:
  "I need a system which can calculate the value each 25 ns."

- What you might understand is:
  "The calculation needs to be finished within 25 ns"

- What he means is:
  "A new value needs to be processed every 25 ns. How long it takes to present the result does not matter"

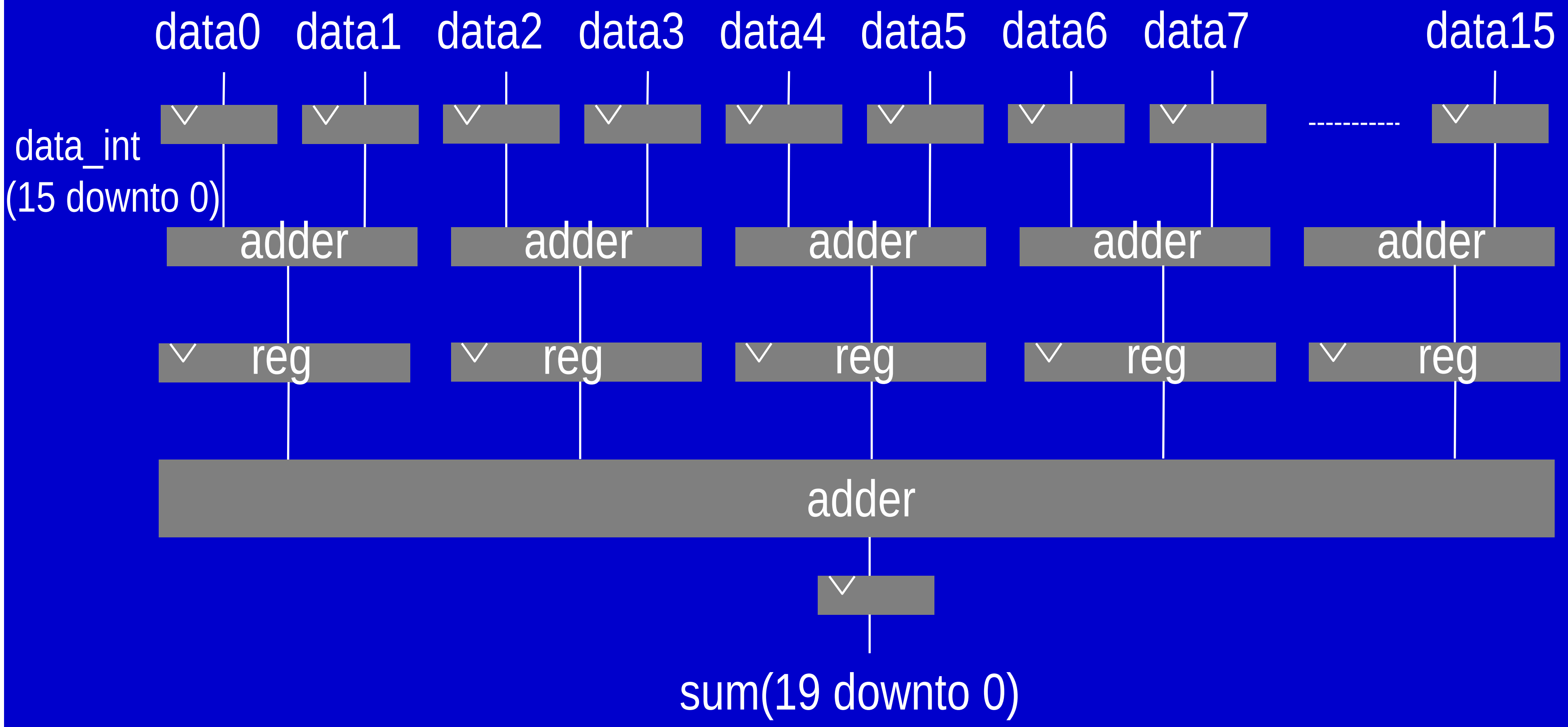
- First case: might be impossible, maybe not.
  Second case: Processors in parallel or in pipeline

Pipeline architecture



Adder with pipeline

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

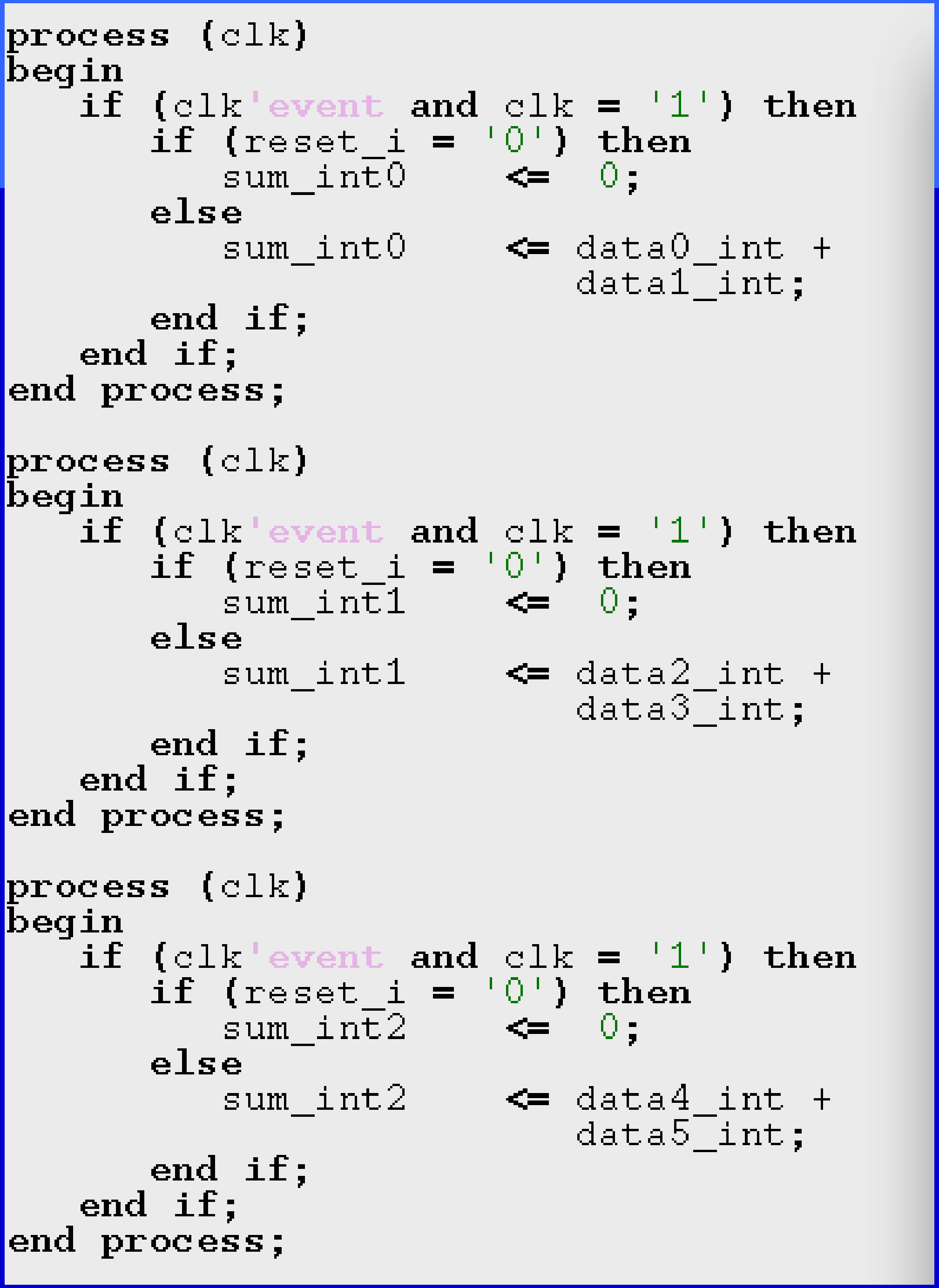


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

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;
```

24

20



```
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;
end process;
```

```
        end if;
    end if;
end process;

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;
```

```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     <= 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

- 22ns < 25 ns, fast enough and less logic

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

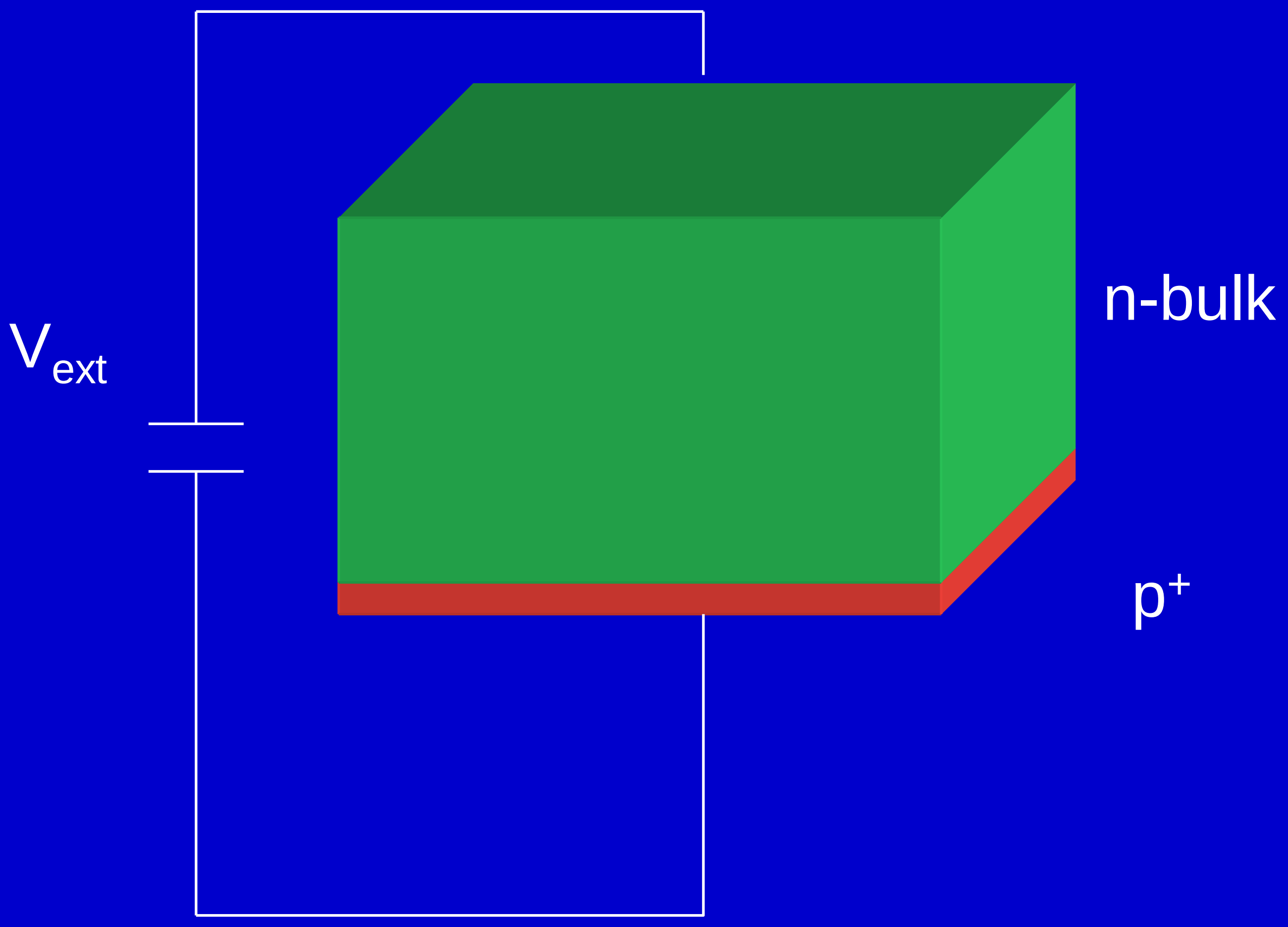
What do we need to know?

Silicon Sensor

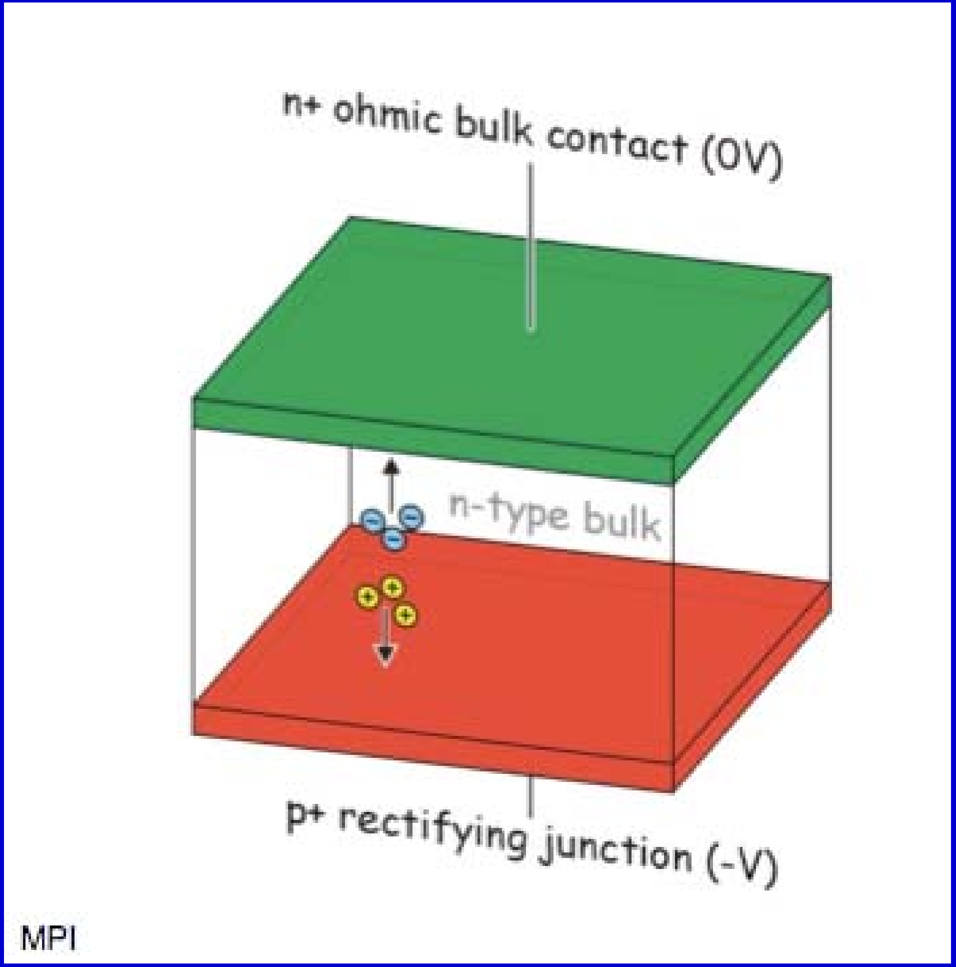
Position resolution: 10 µm

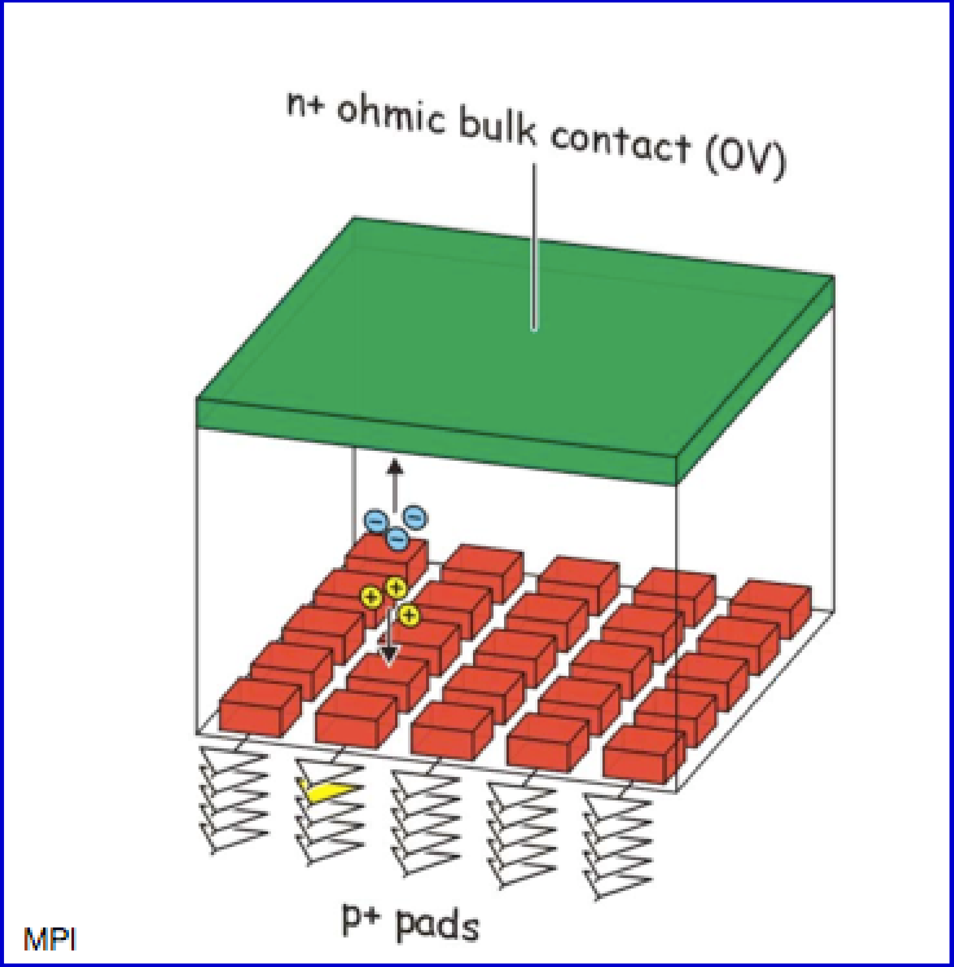
light material: 1 % X_0 oder 2 mm

Silicon Sensors

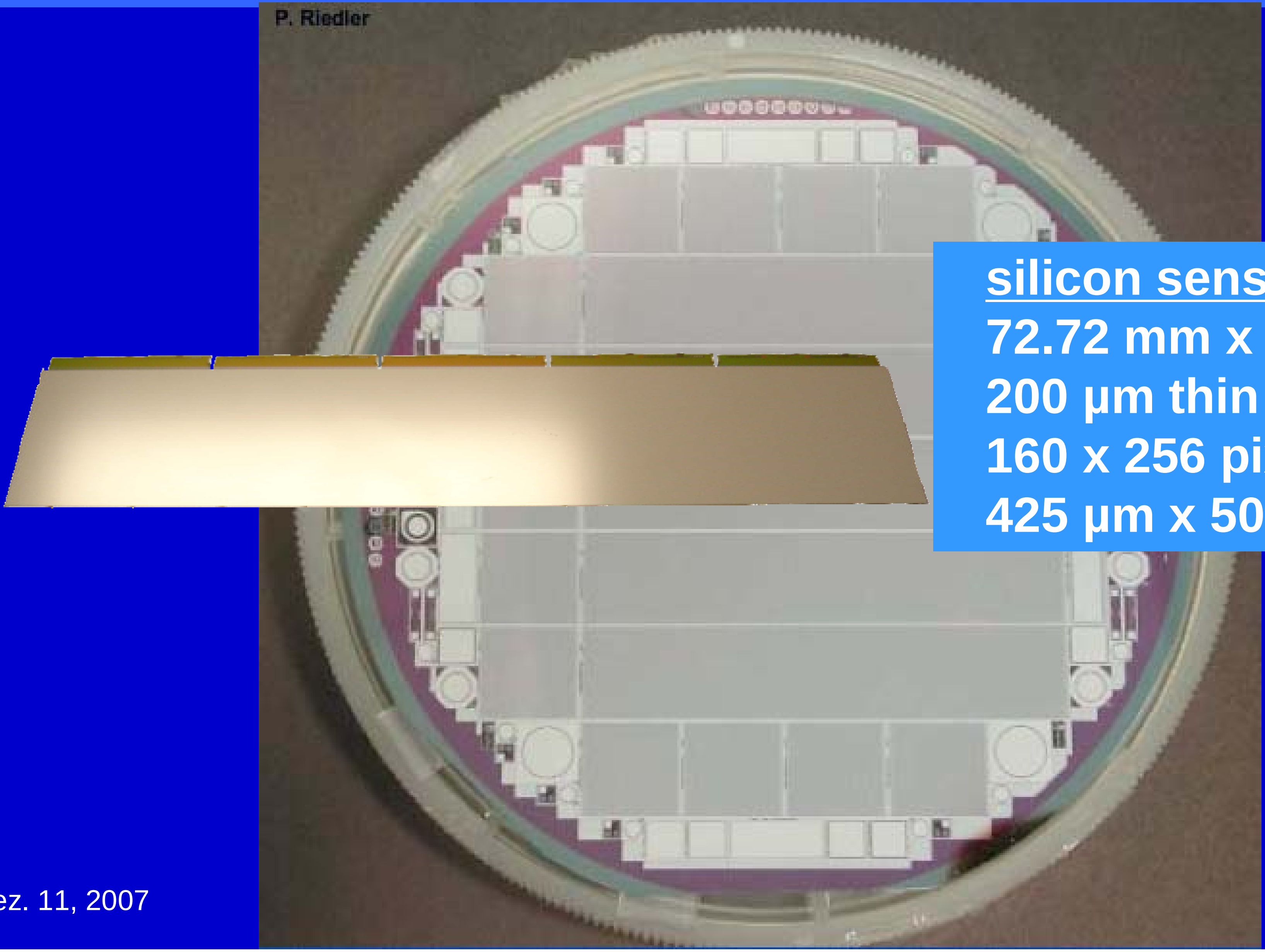


Silicon Pixel sensors





Silicon Pixel Wafers



silicon sensor

72.72 mm x 13.92 mm

200 µm thin

160 x 256 pixel

425 µm x 50 µm

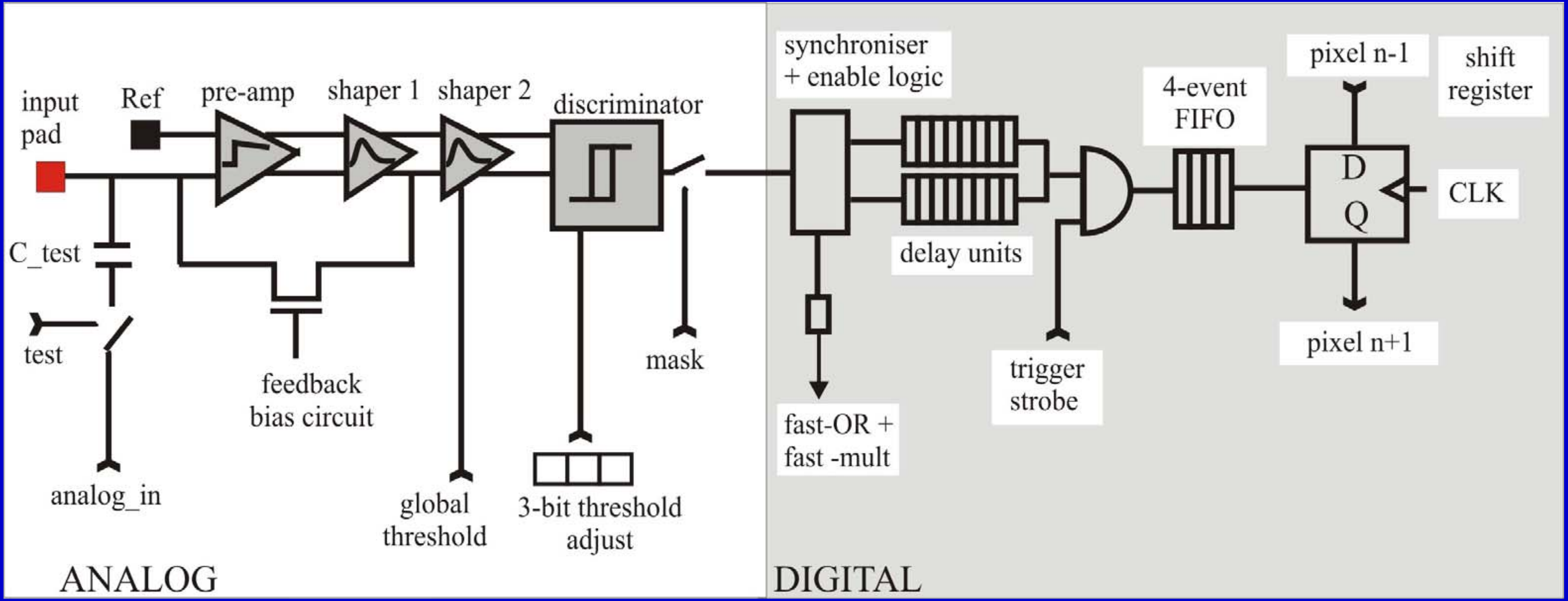
Pixel read out chip

Time resolution: 25 ns

Repetition frequency: 40 MHz

Storage time: > 3.2 µs

Pixel chip


input pad
Ref
pre-amp
shaper 1
shaper 2
discriminator
C_test
test
feedback bias circuit
mask
analog_in
global threshold
3-bit threshold adjust
ANALOG
synchroniser + enable logic
delay units
fast-OR + fast -mult
trigger strobe
4-event FIFO
pixel n-1
shift register
D
Q
CLK
pixel n+1
DIGITAL

Pixel detector

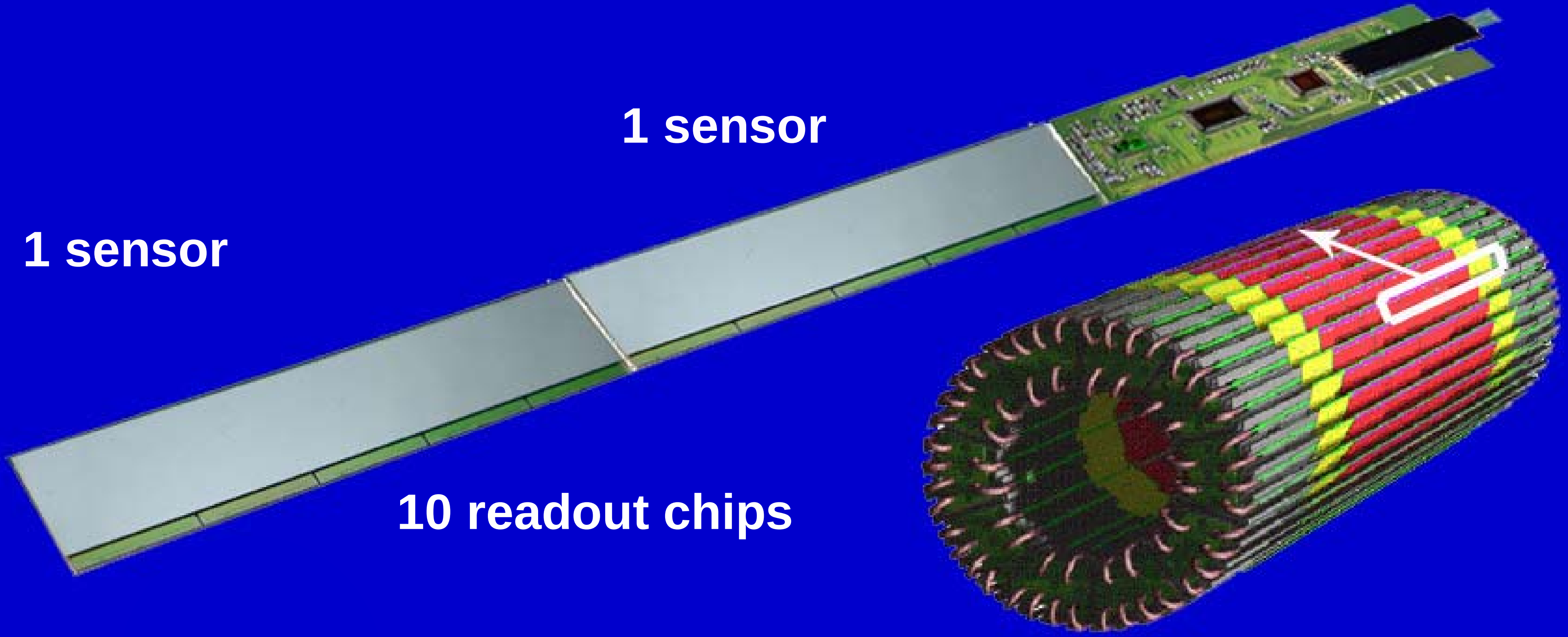


Image:INFN(Padova)

Pixel detector

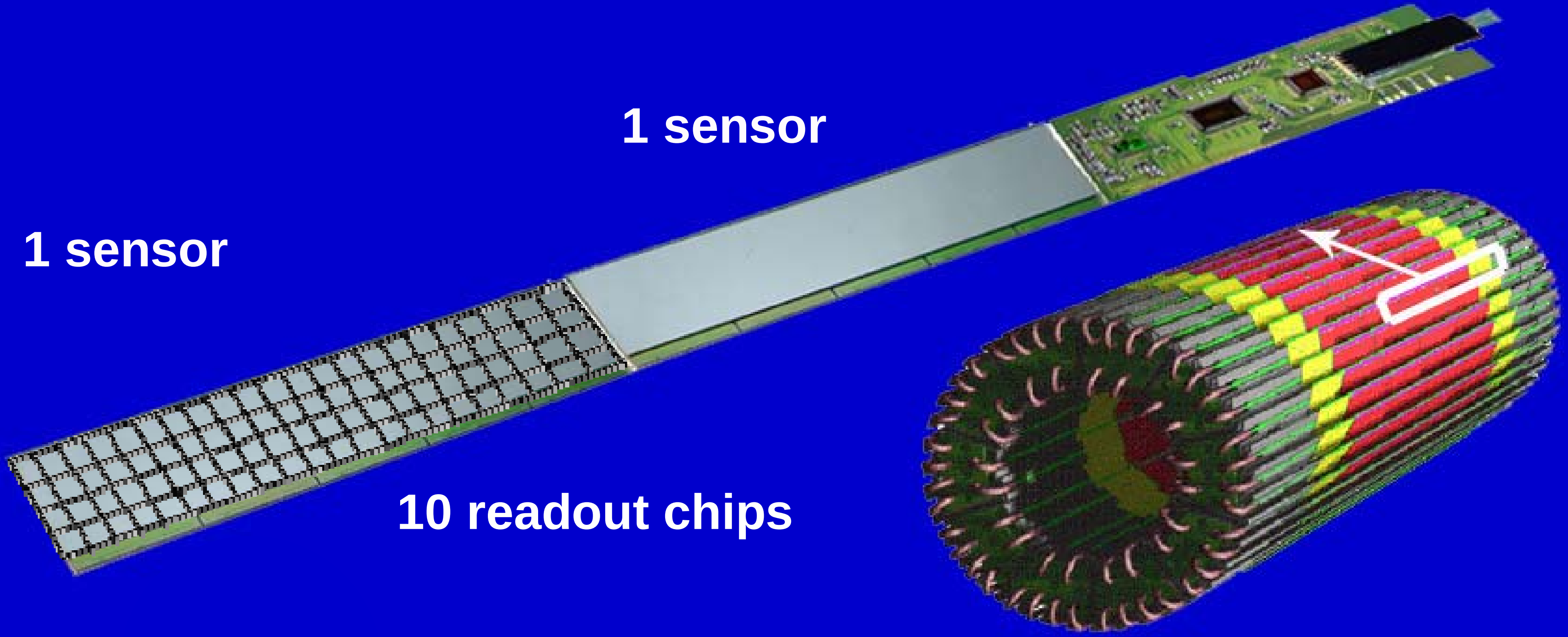
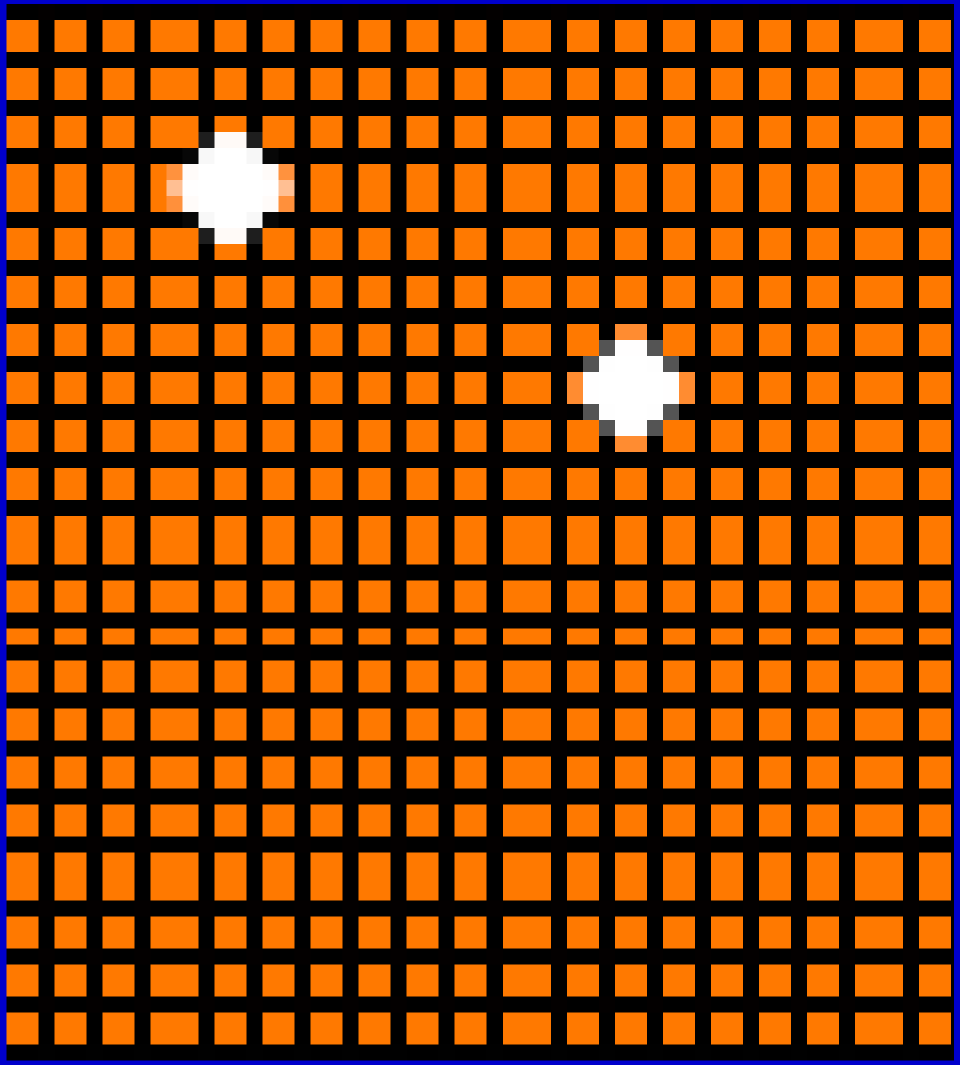


Image:INFN(Padova)

Pixel detector



00001000000000000000000000

00000000000000000100000000

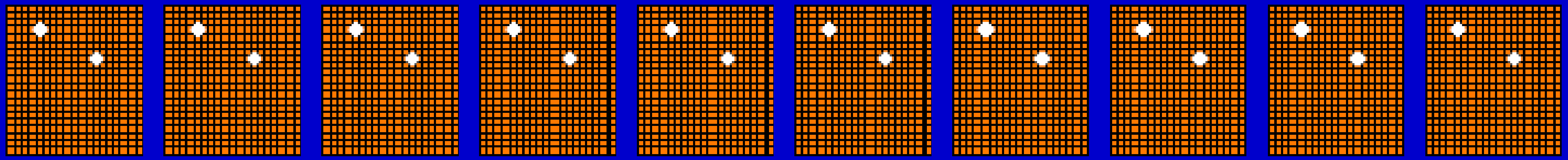
00000000001000000100000100

00000000000000000000000000

Pixel detector

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

Separate read-out for each detector module



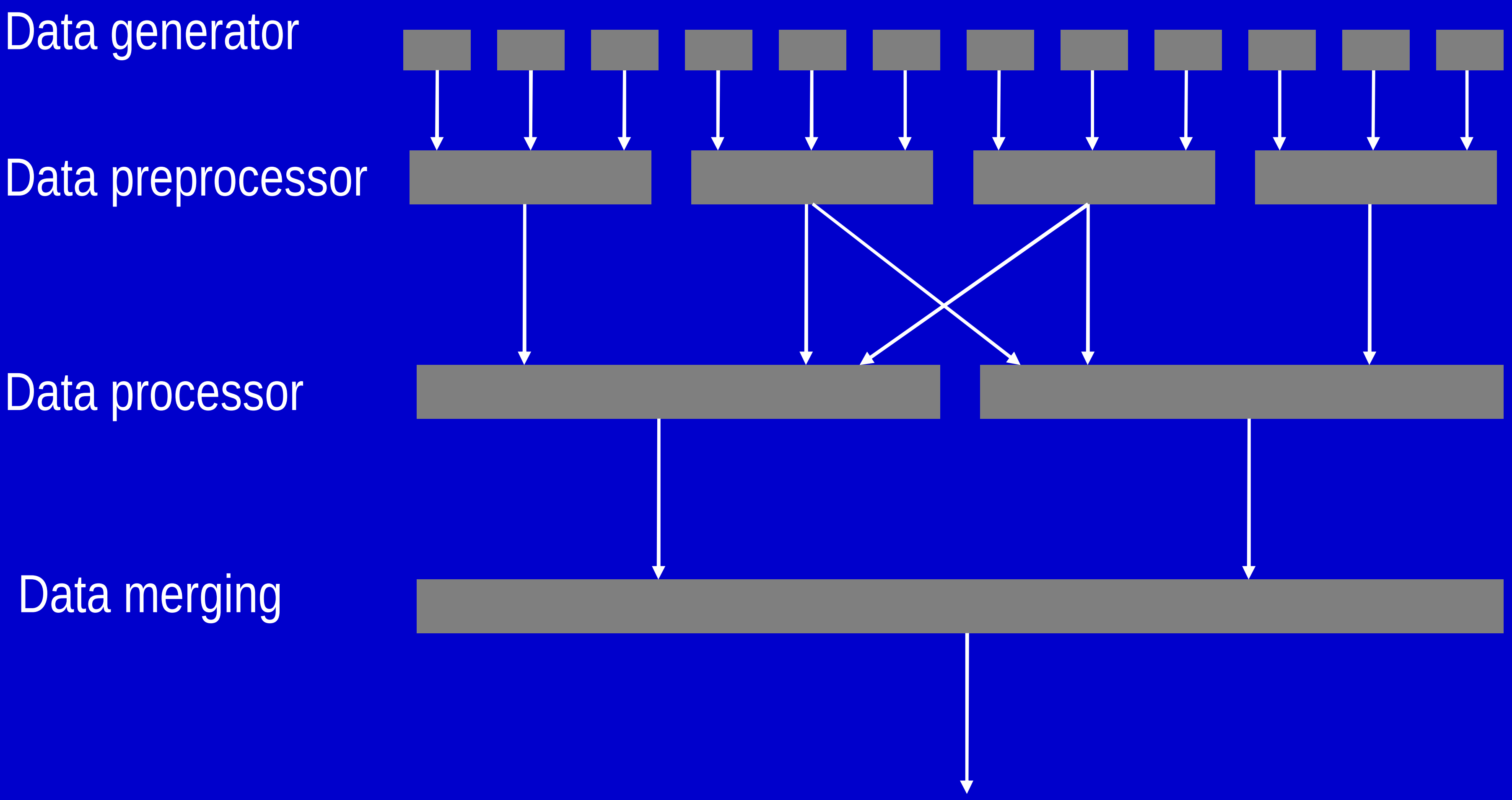
Each detector module (10 chips) 1 x 2560 x 32 bits @ 10 MHz

00001000000000000000000000000000

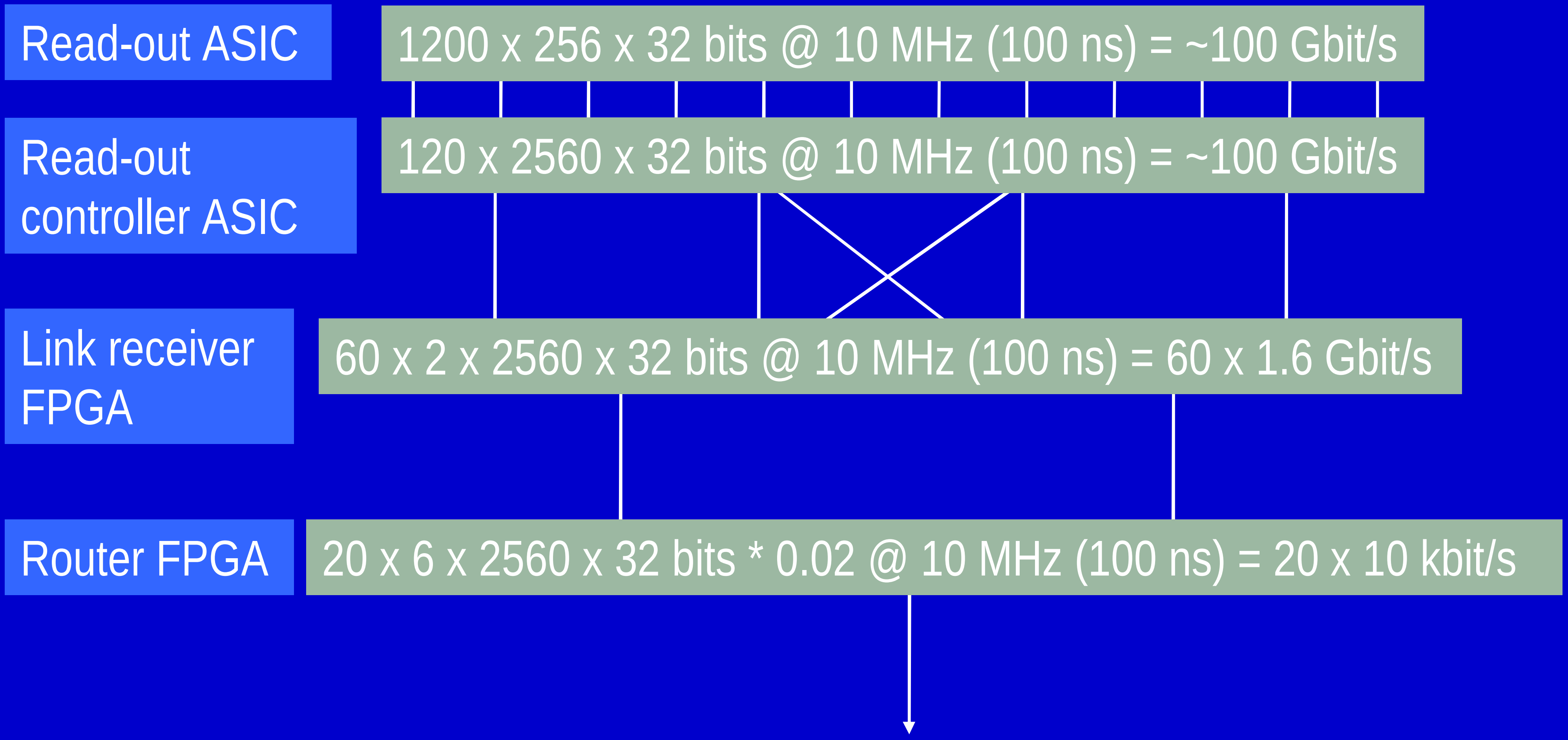
00000000000000000100000000000000

00000000000100000001000000000000

Data funnel


Data generator
Data preprocessor
Data processor
Data merging

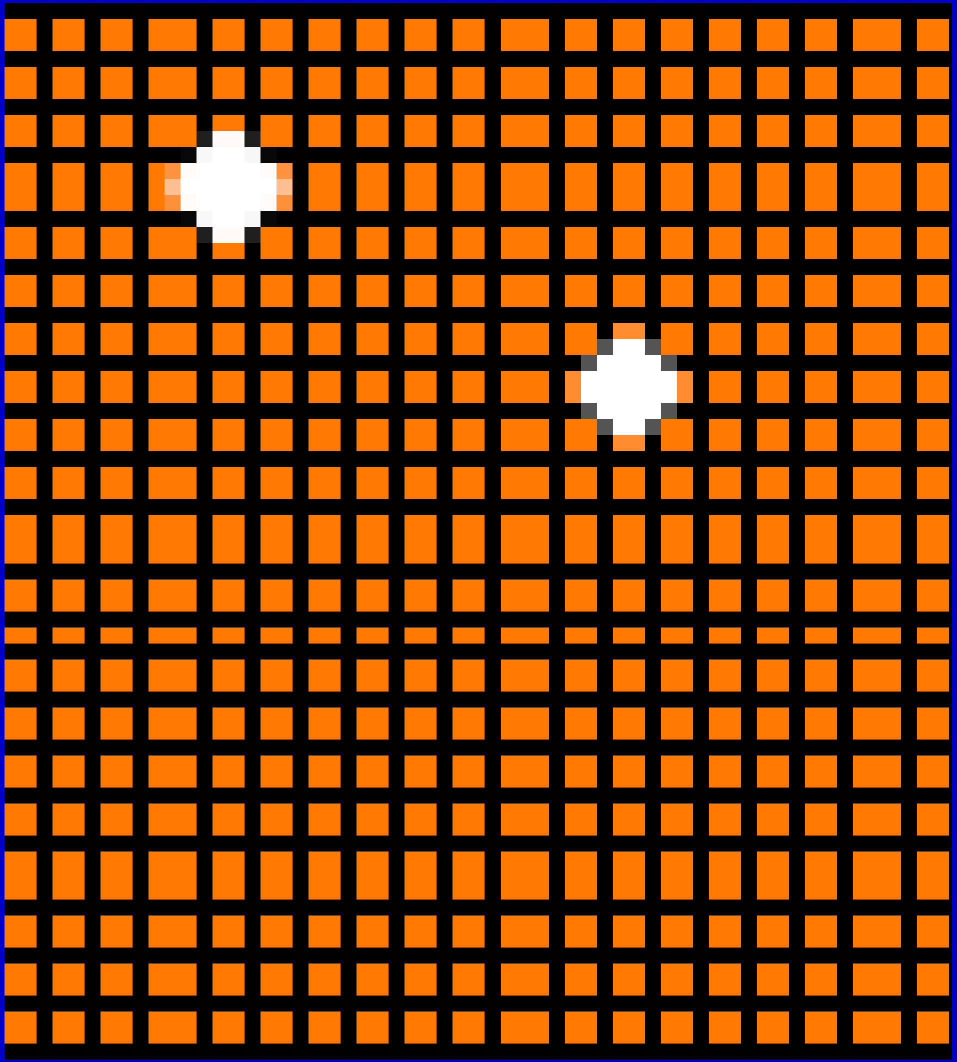
Data funnel



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

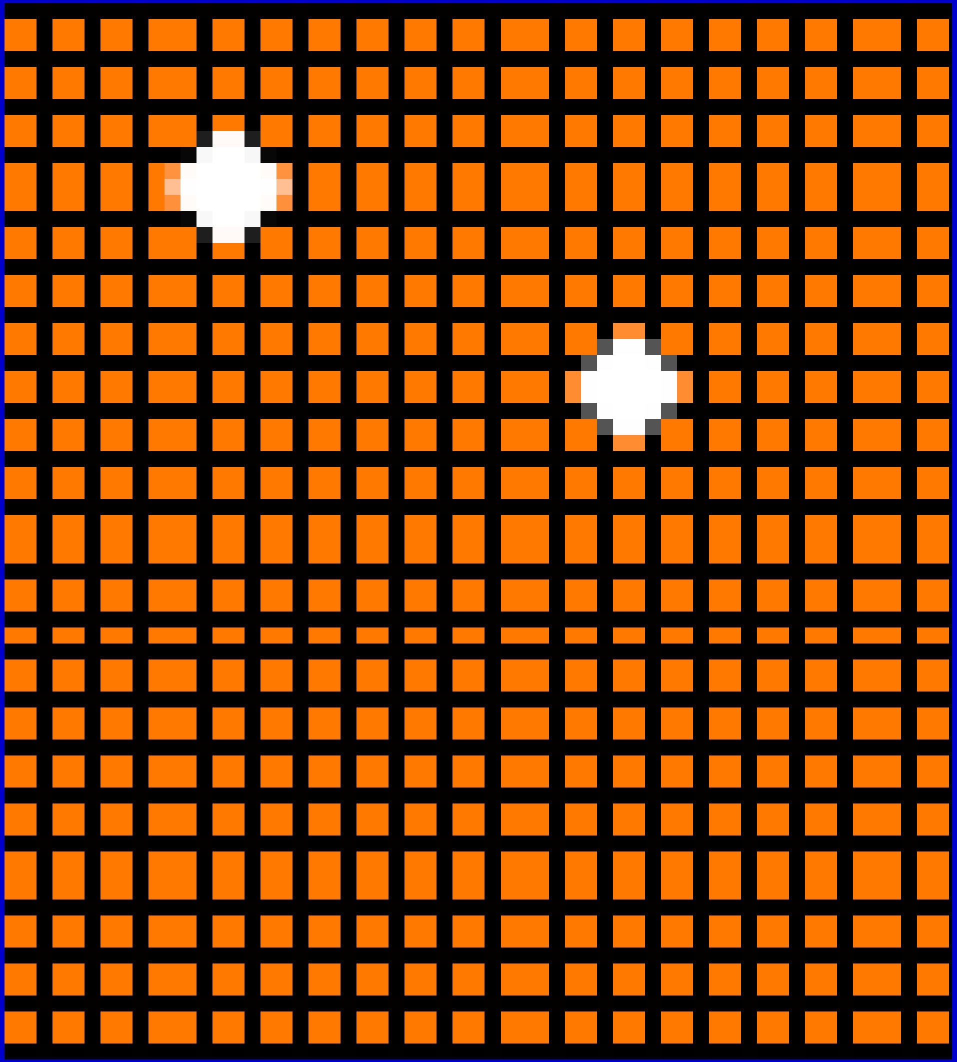
Data generator
2560 x 32 bits



```
00001000000000000000000000
00000000000000000100000000
00000000001000000100000100
00000000000000000000000000
```

Pixel detector

What is the strategy?



00001000000000000000000000

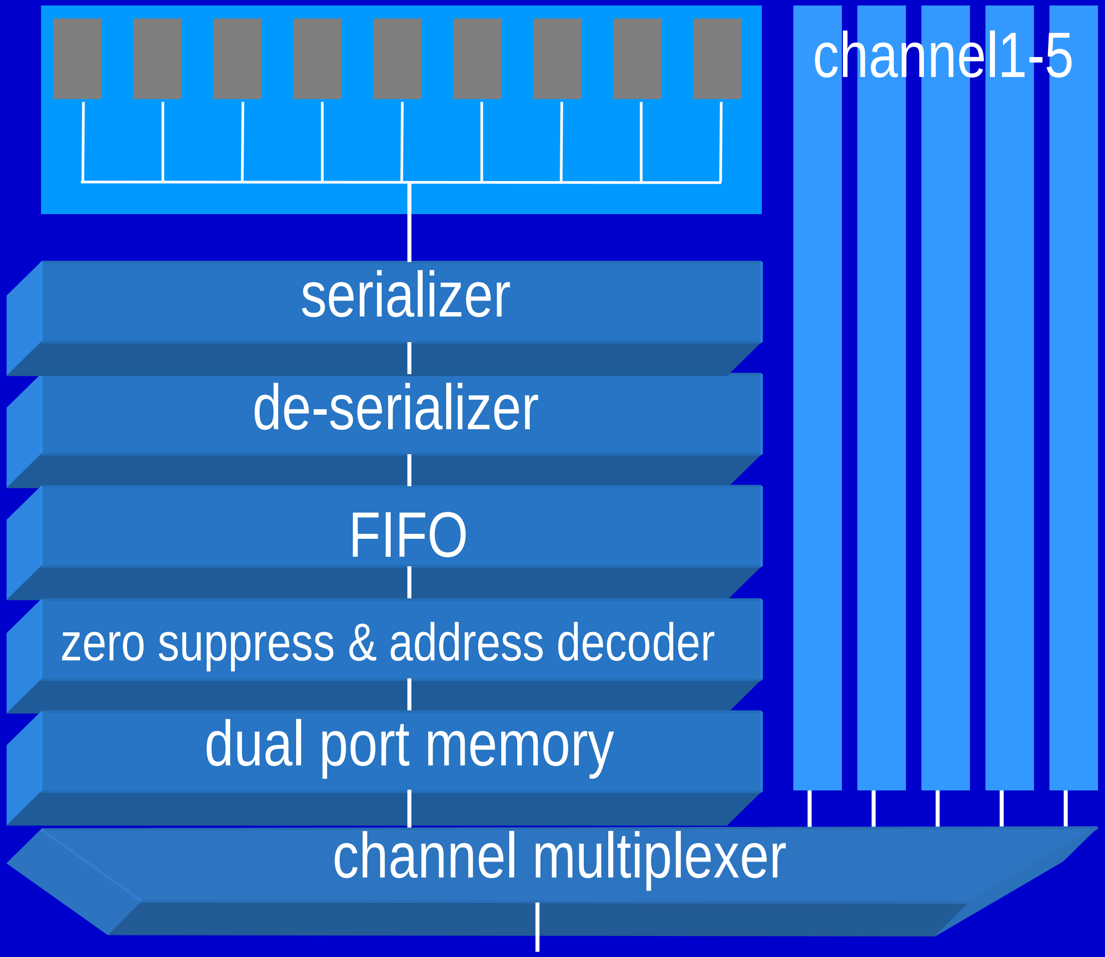
00000000000000000100000000

00000000001000000100000100

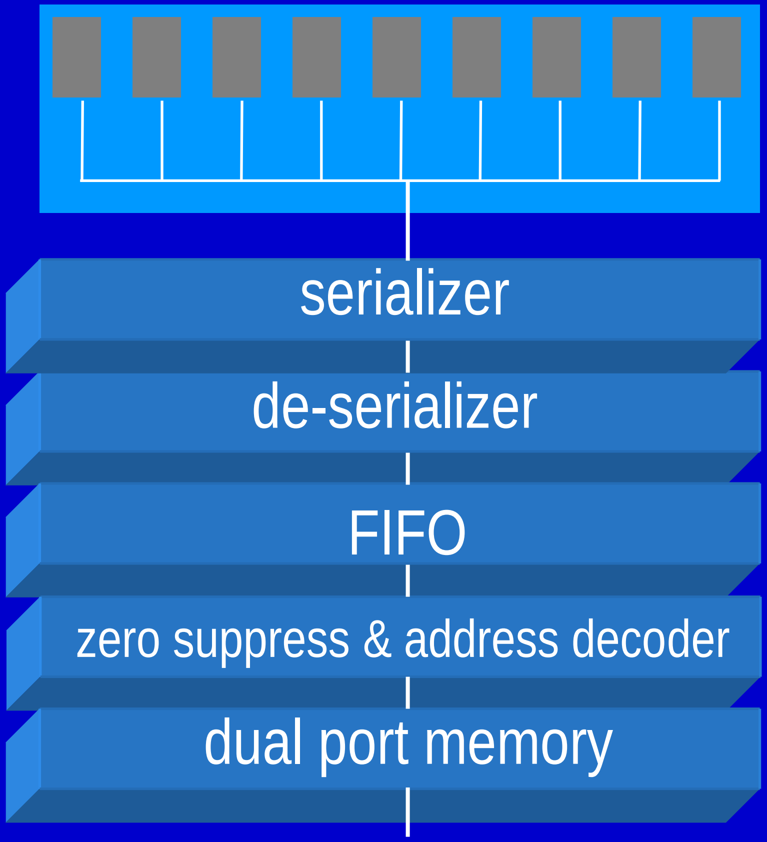
00000000000000000000000000

Some body counts values all the time, find out whether they can be divided by three, what to you do in real life? Include serial and dpm

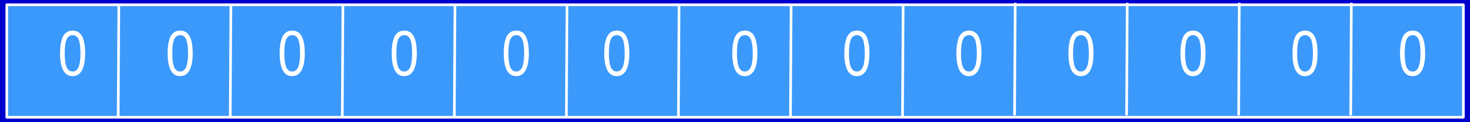
Pixel detector



Pixel detector



Pixel detector data processing

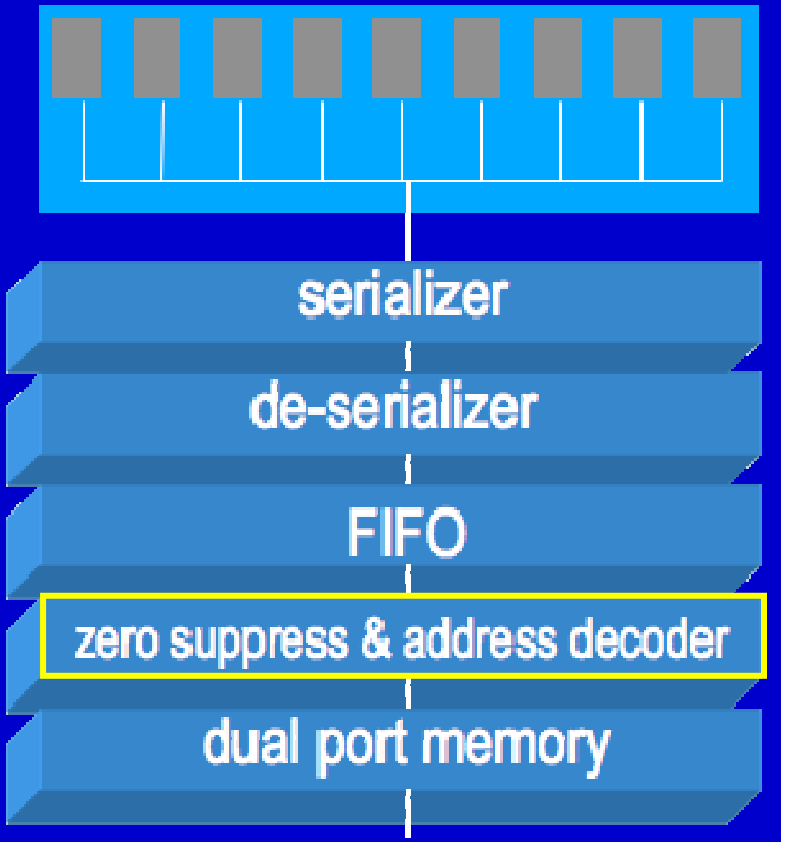


check if any hits

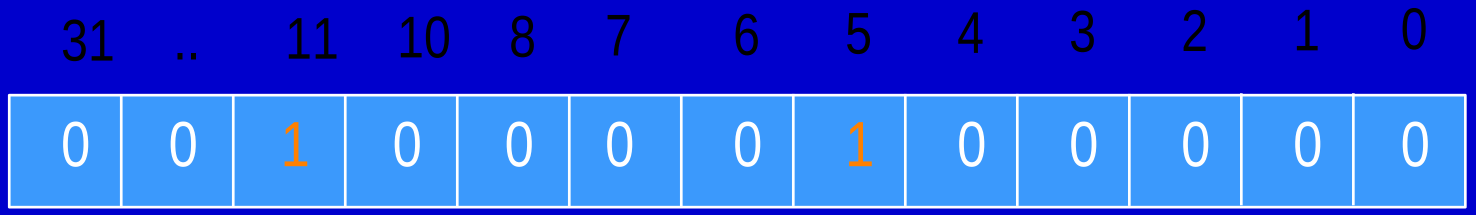
if no hits ->
load new value from FIFO

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

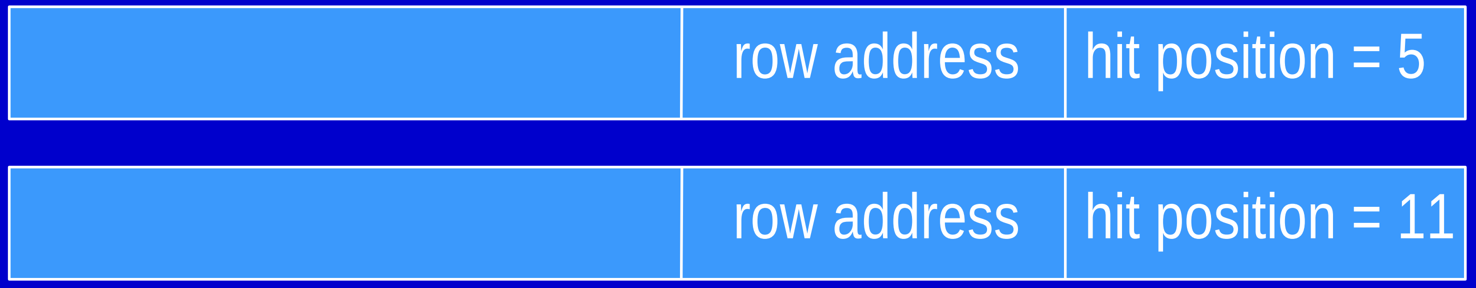
if more than one hit -> decode the hits



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

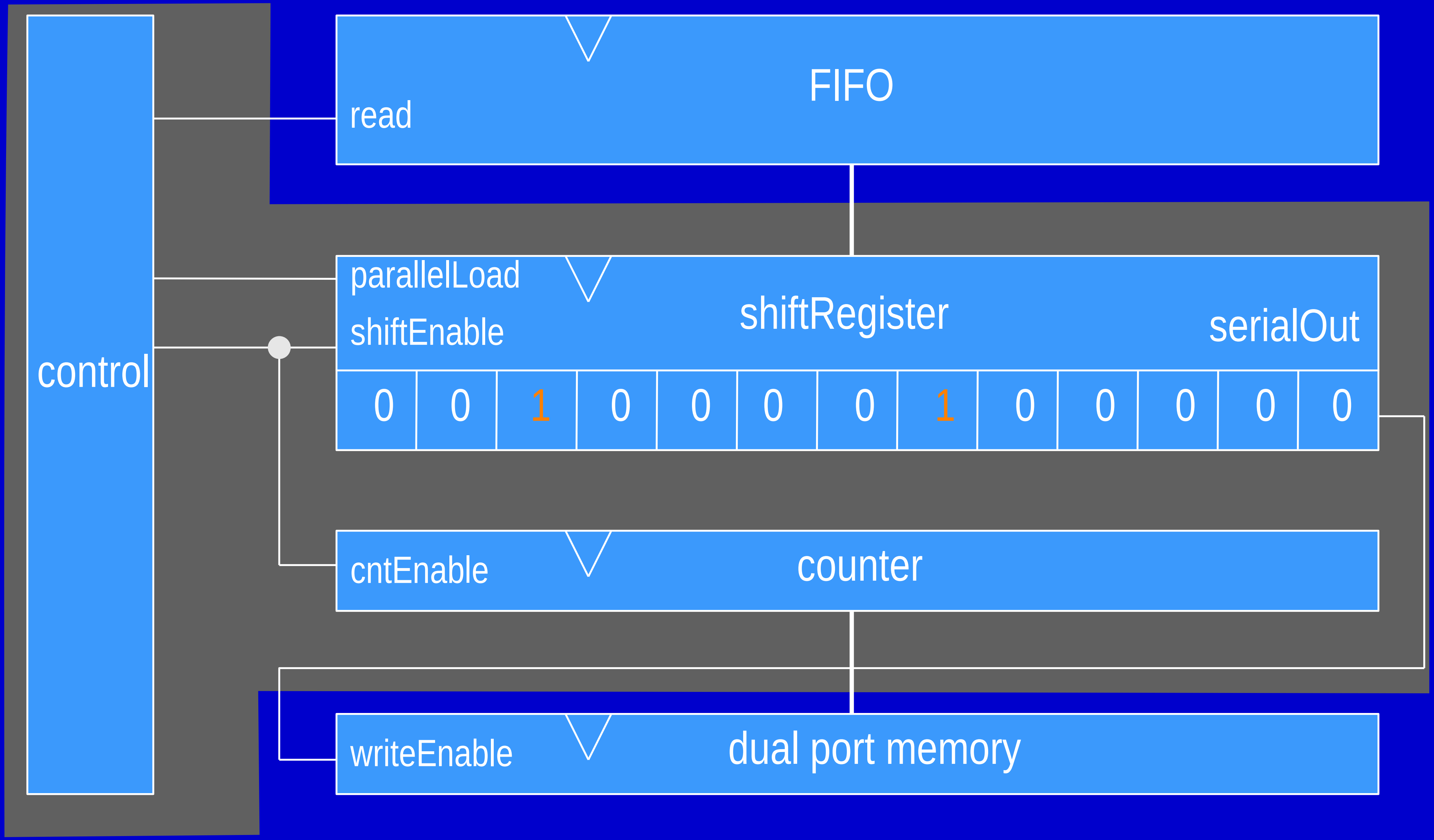
31	..	11	10	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1	0	0	0	0	0

Do we know enough to start the project?
How do we encode the address?

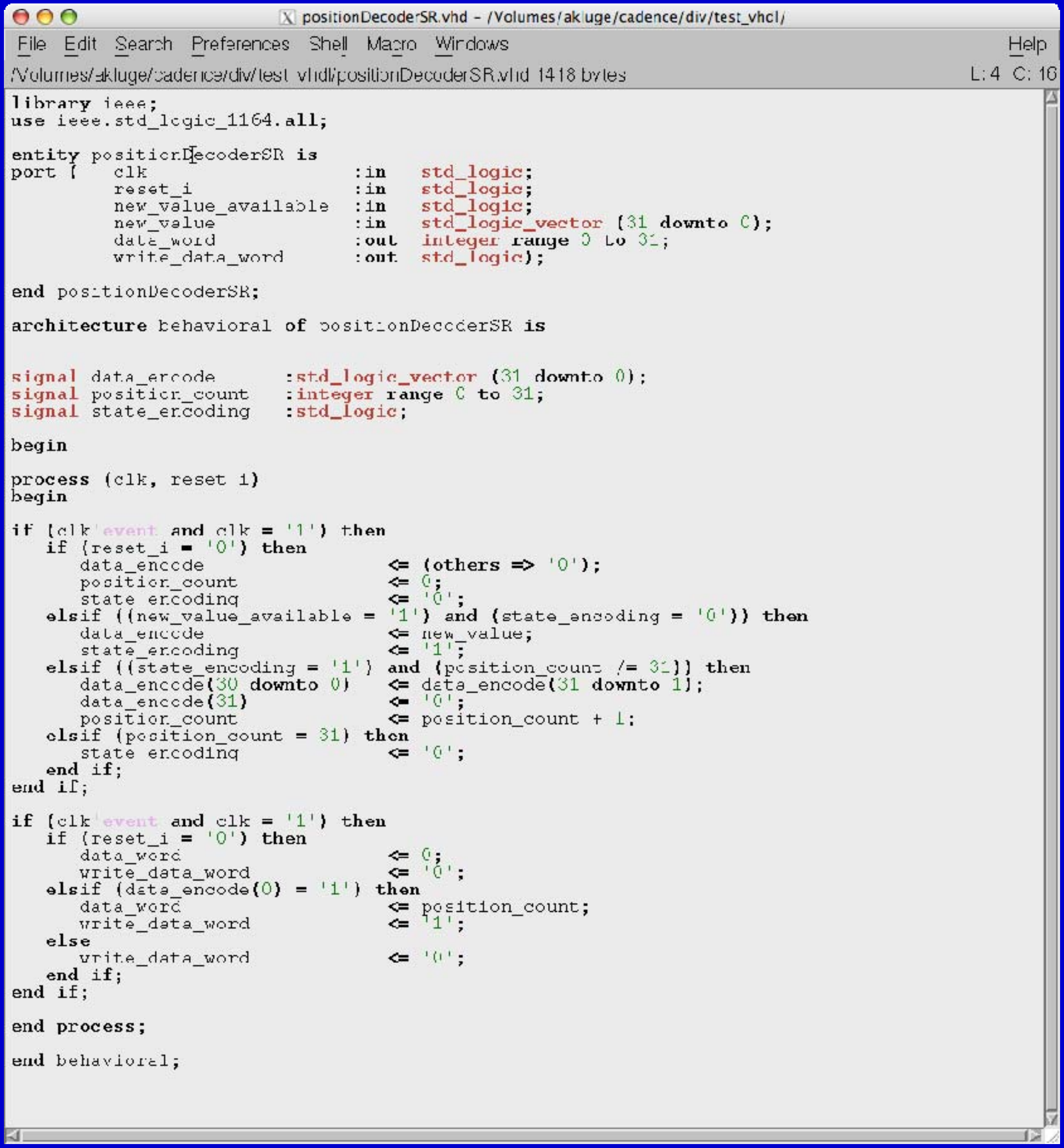
	row address	hit position = 5

	row address	hit position = 11

Pixel detector data processing

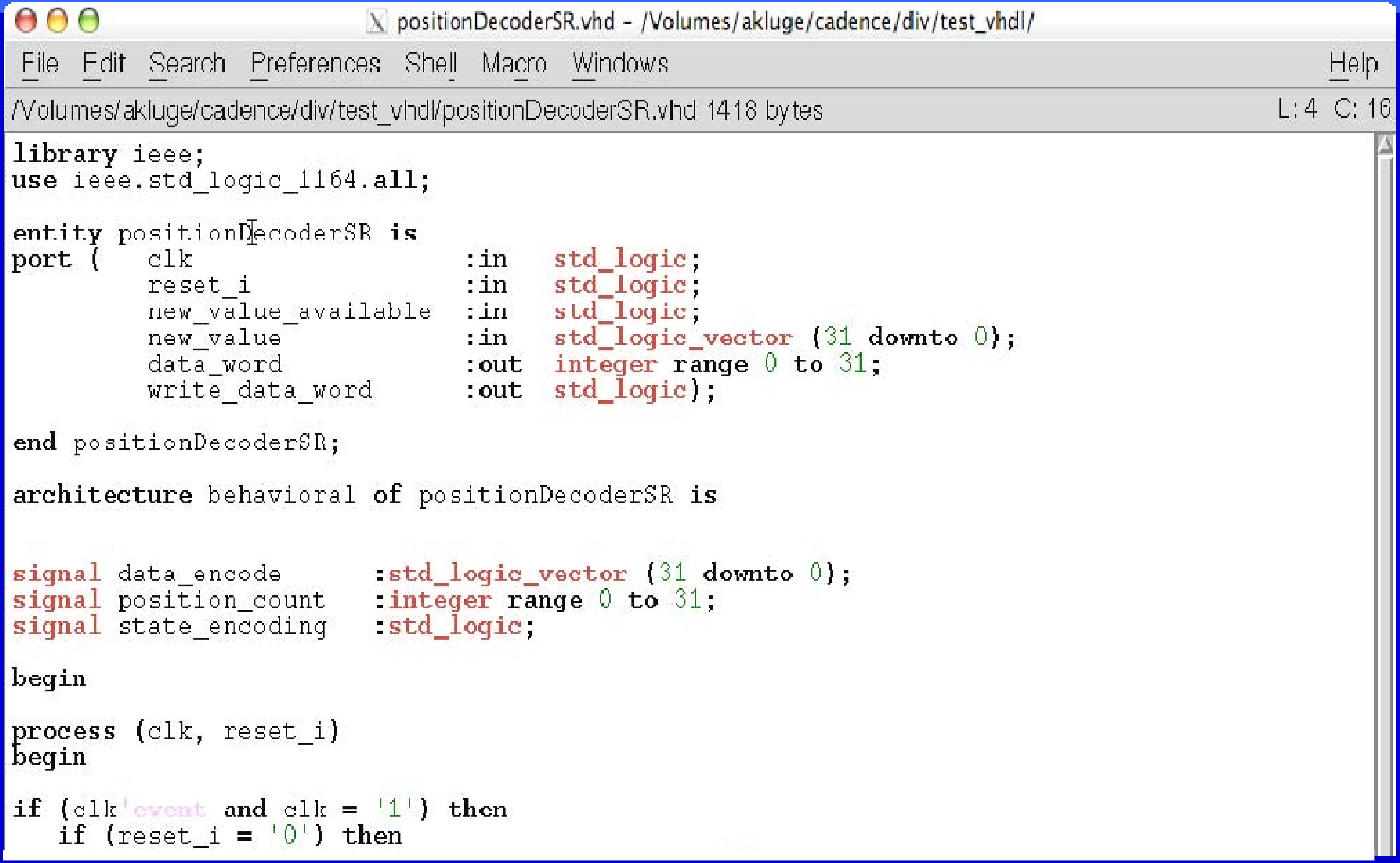


Position decoder – shift register


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

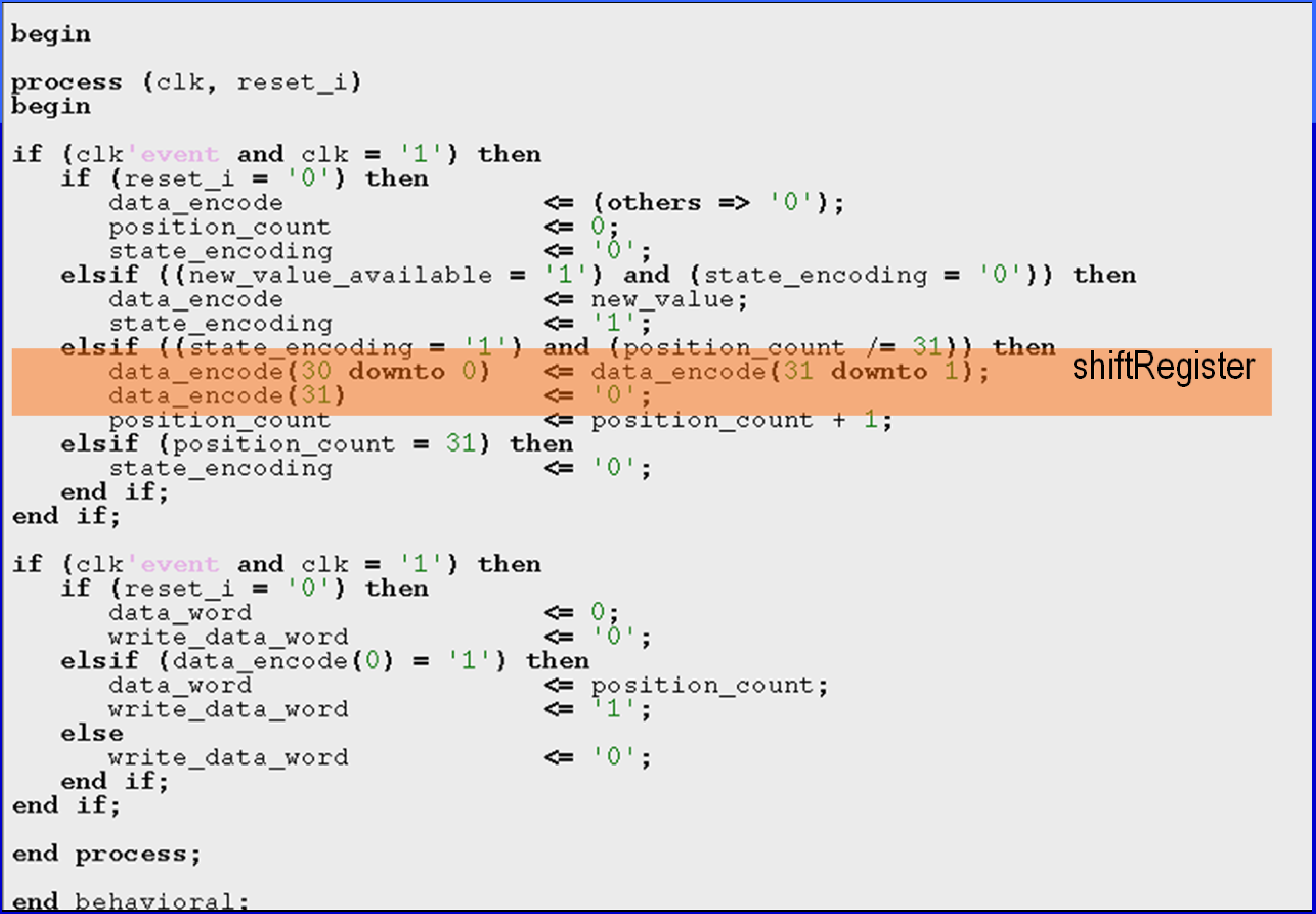
```vhdl
library ieee;
use ieee.std_logic_1164.all;

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

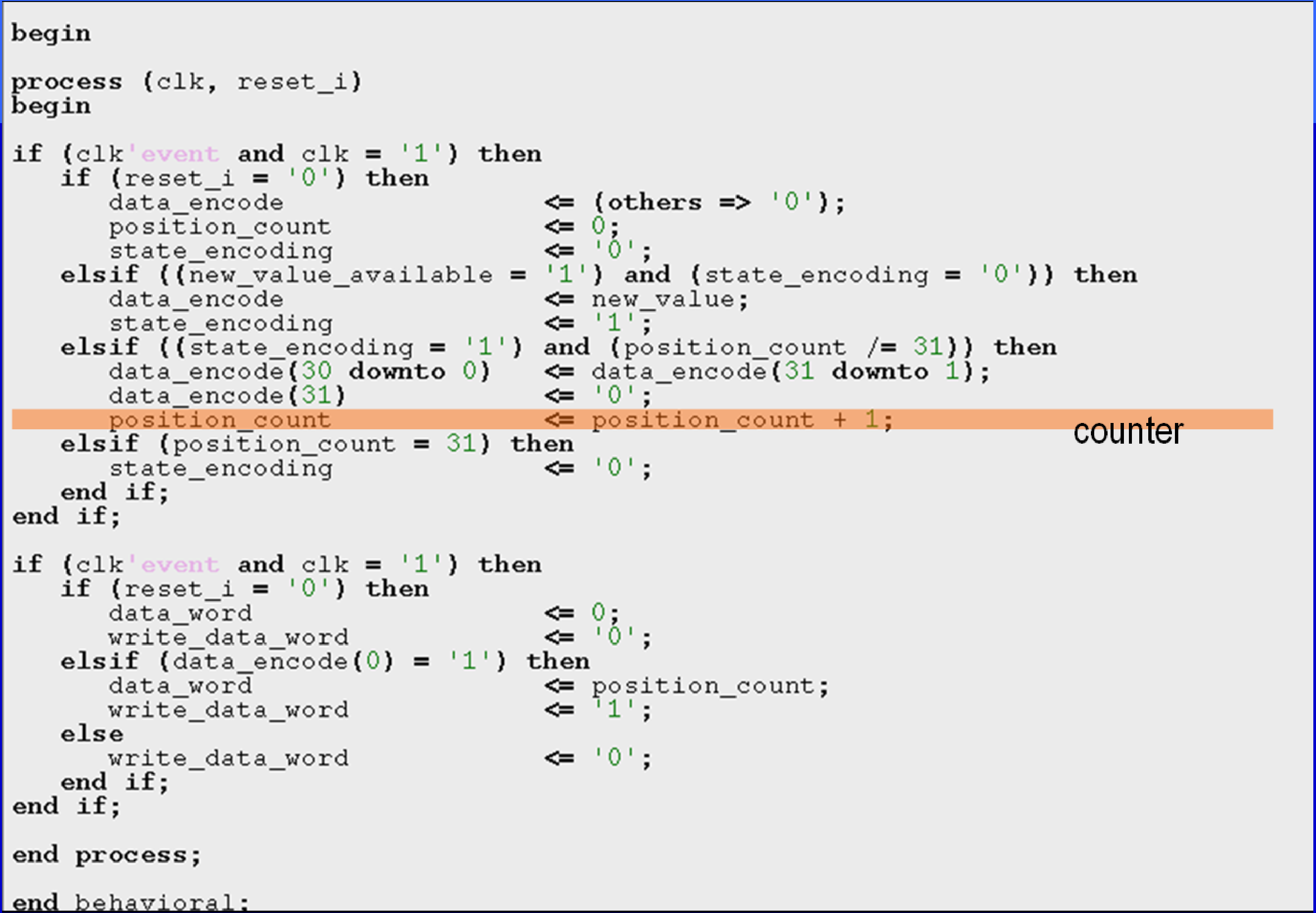
```vhdl
library ieee;
use ieee.std_logic_1164.all;

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
```

```
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

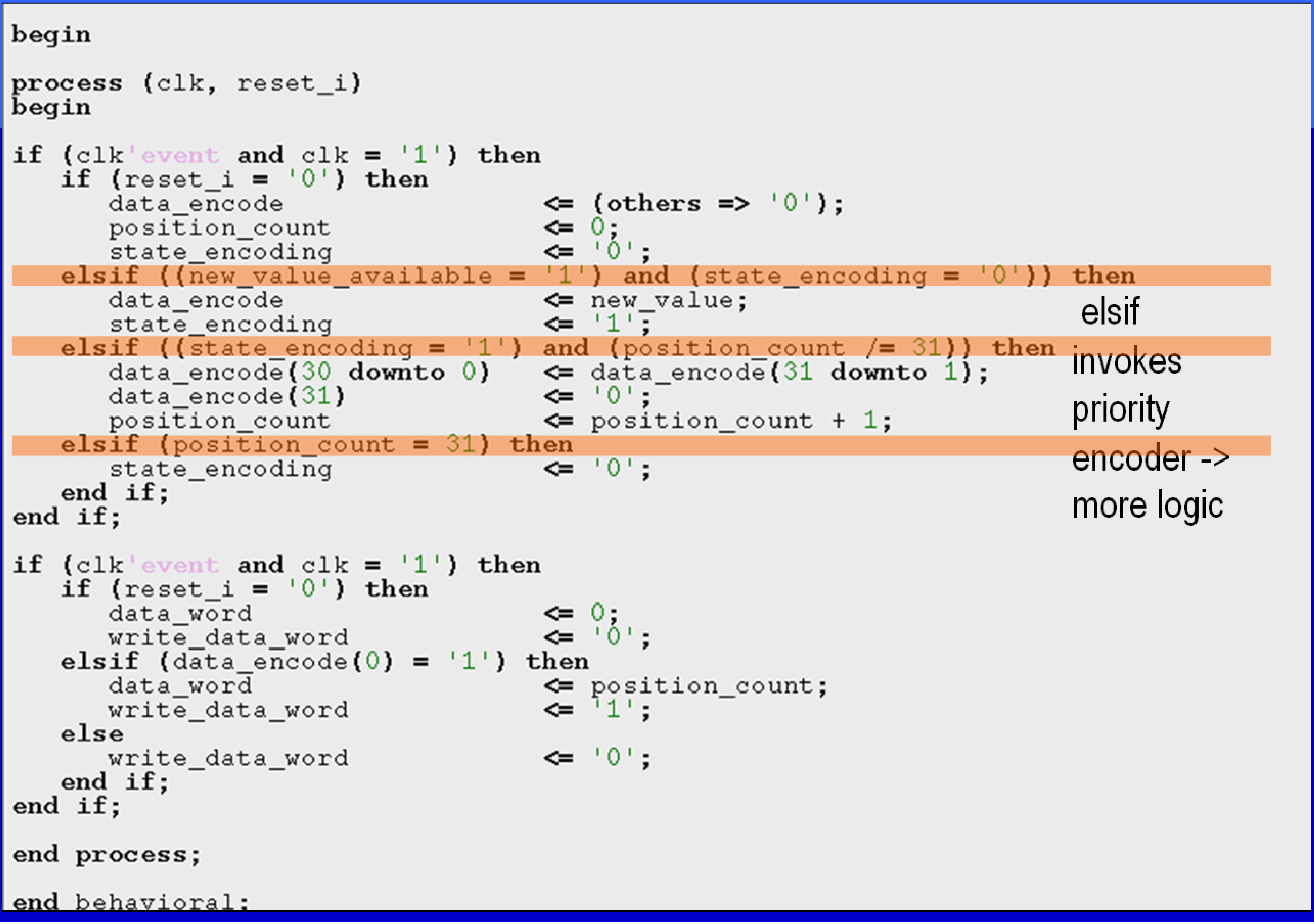


```
begin

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

```vhdl
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

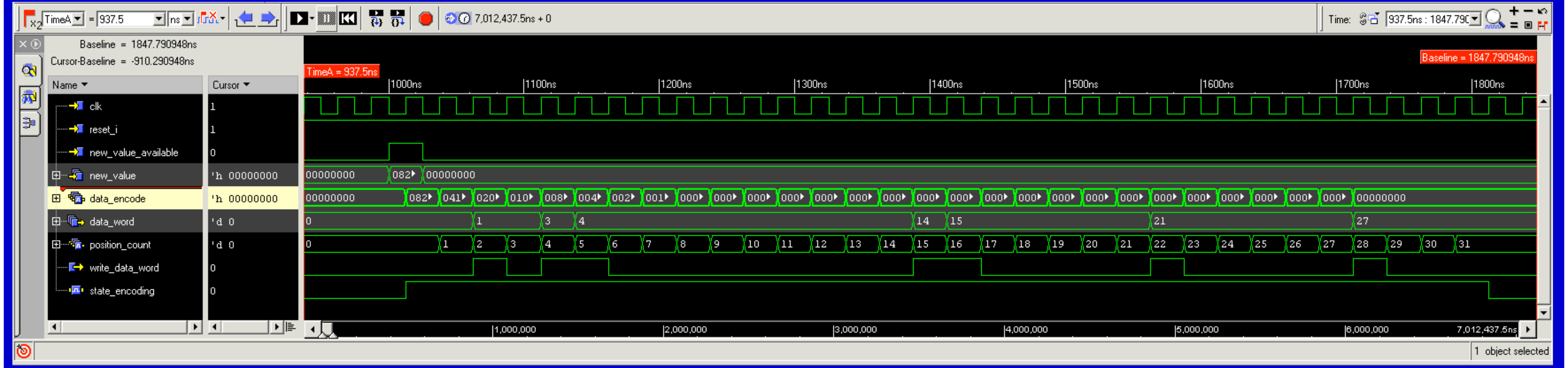
```
process (clk, reset_i)
begin

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
```

- Shift register is a parallel load register

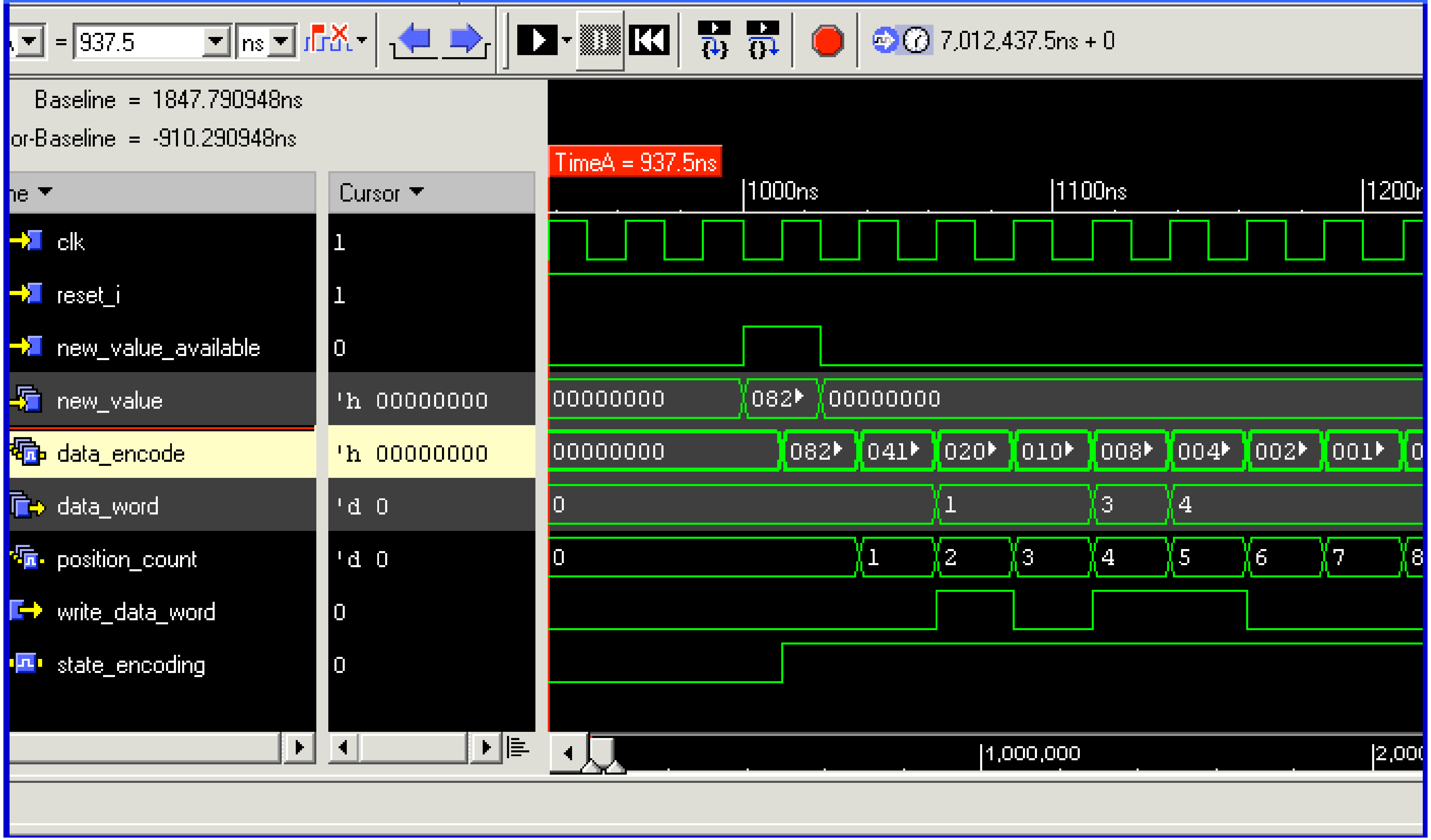
Position decoder – shift register

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

"00001000000100000011000000000011010"

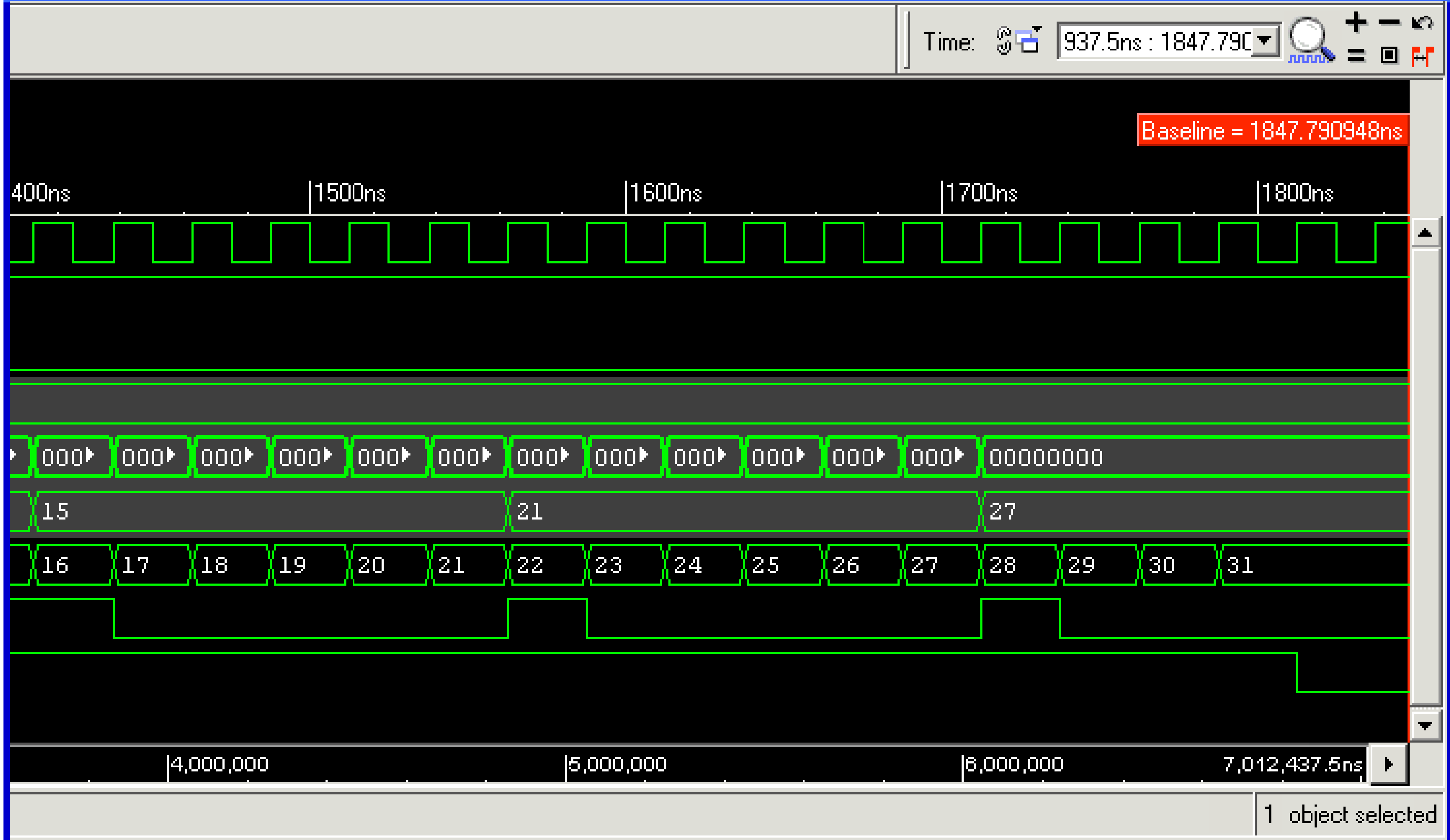


Position decoder – shift register



```
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 | 0
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?**
- **User requirements fulfilled?**

  - Did we ask enough to understand user needs?
  - Maybe at the time of project start, but now is the time to rediscuss

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

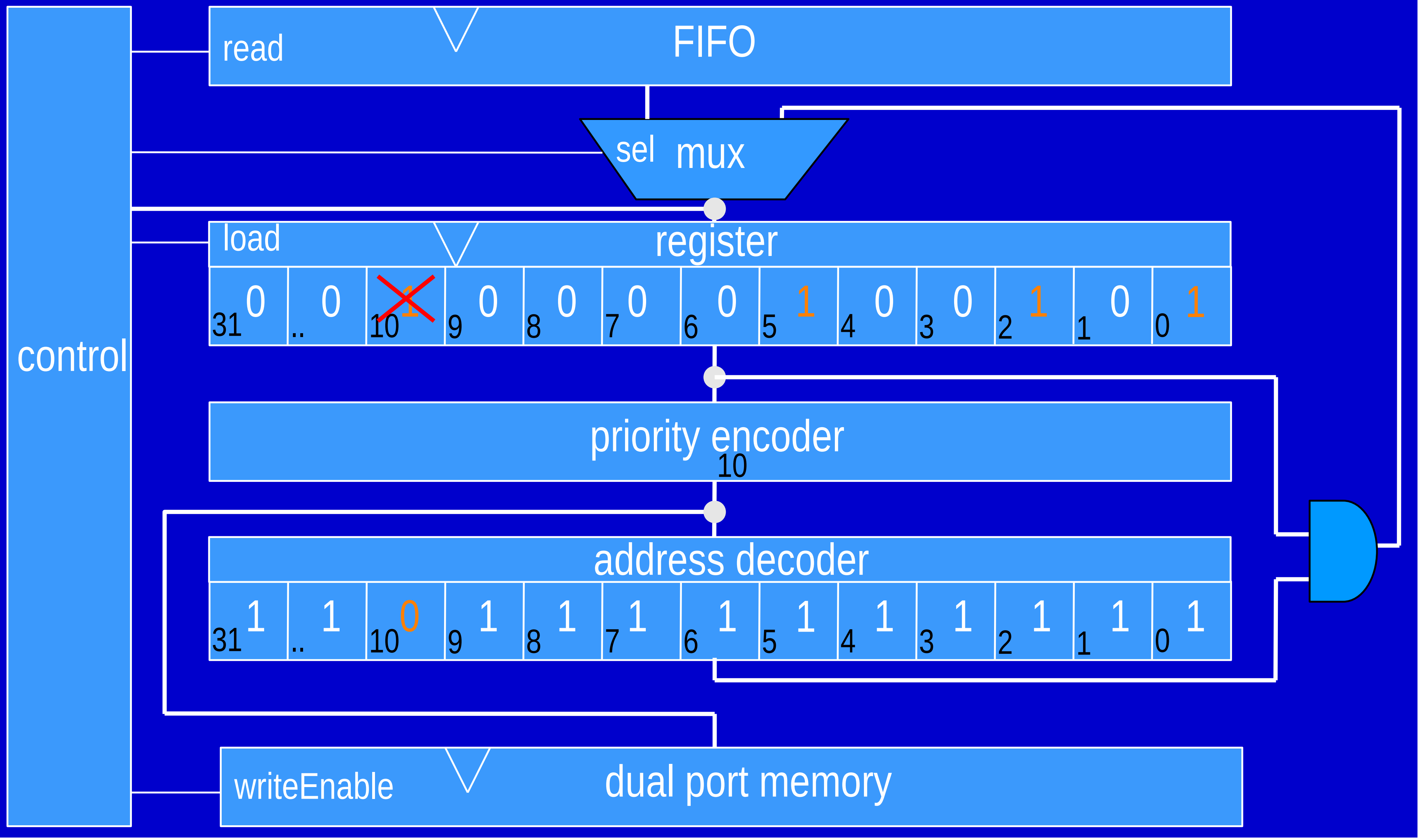
31	..	11	10	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1	0	0	0	0	0

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

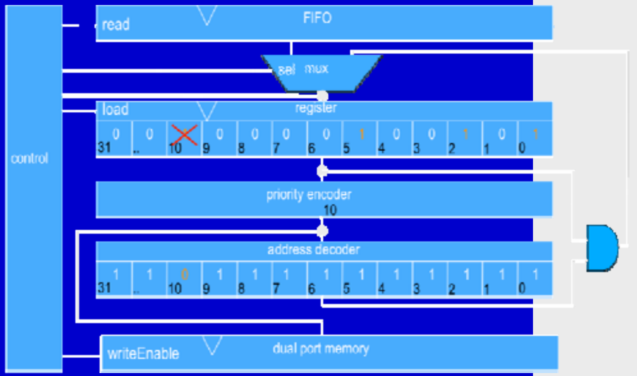
Position decoder – priority encoder



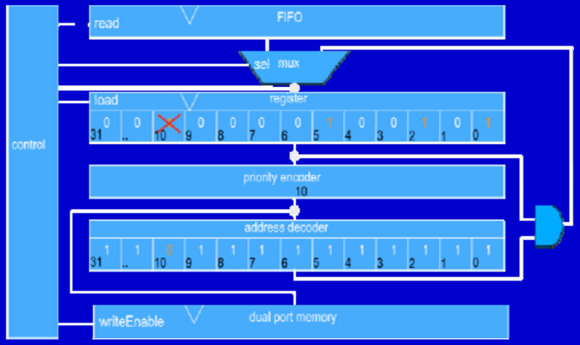
Position decoder – priority encoder

```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);
```

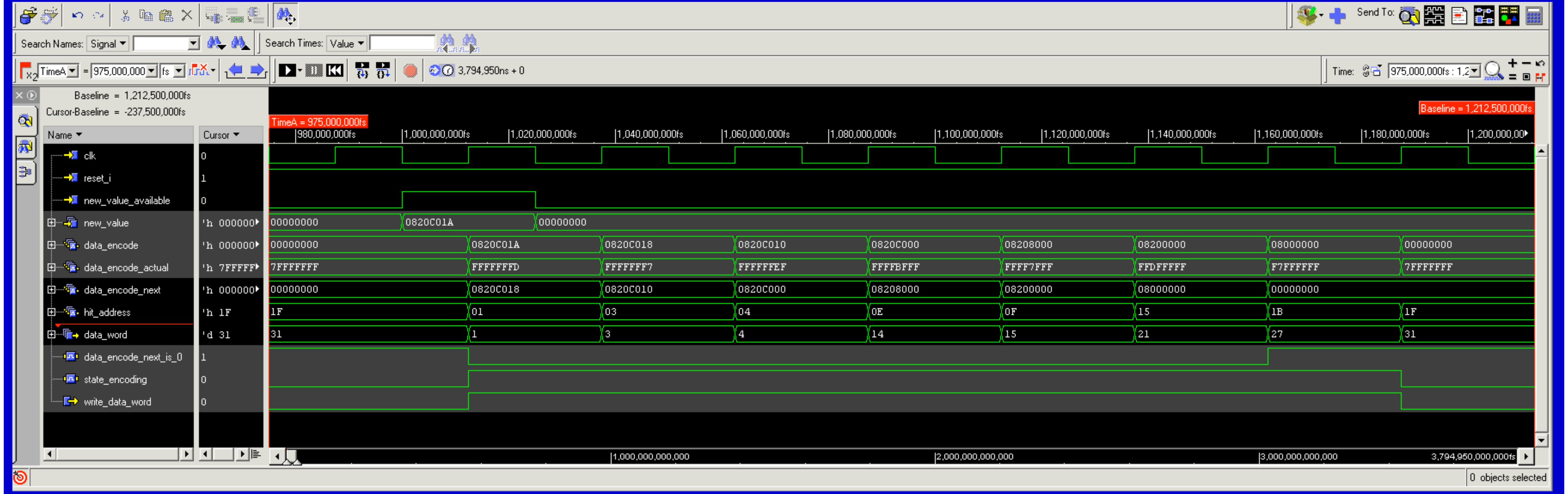


```
end process;

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;
```

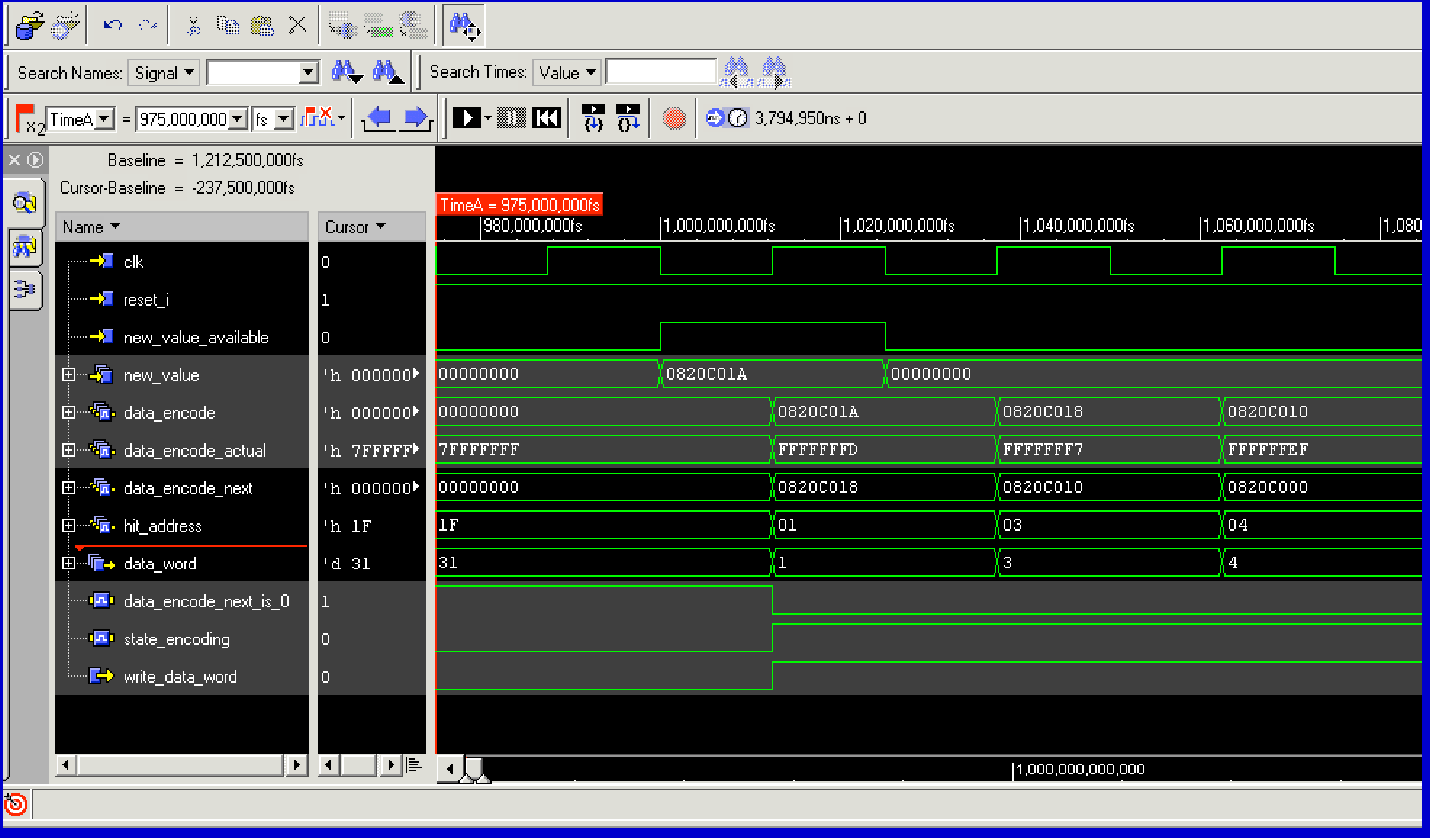
```vhdl
--**********************************************************************
--adressDecoder
--**********************************************************************

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
Baseline = 1,212,500,000fs

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
```

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

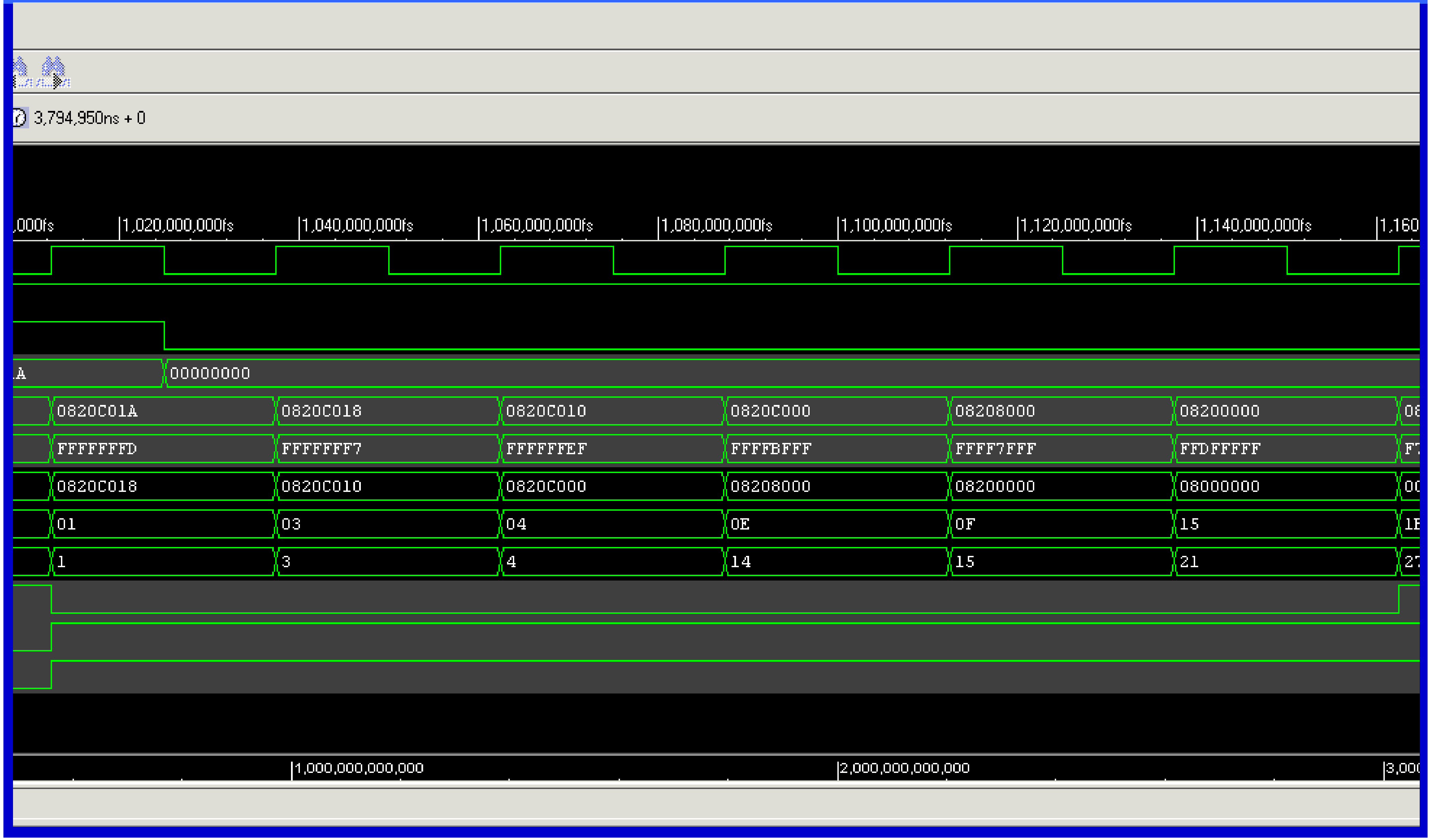
TimeA = 975,000,000fs

980,000,000fs | 1,000,000,000fs | 1,020,000,000fs | 1,040,000,000fs | 1,060,000,000fs | 1,080

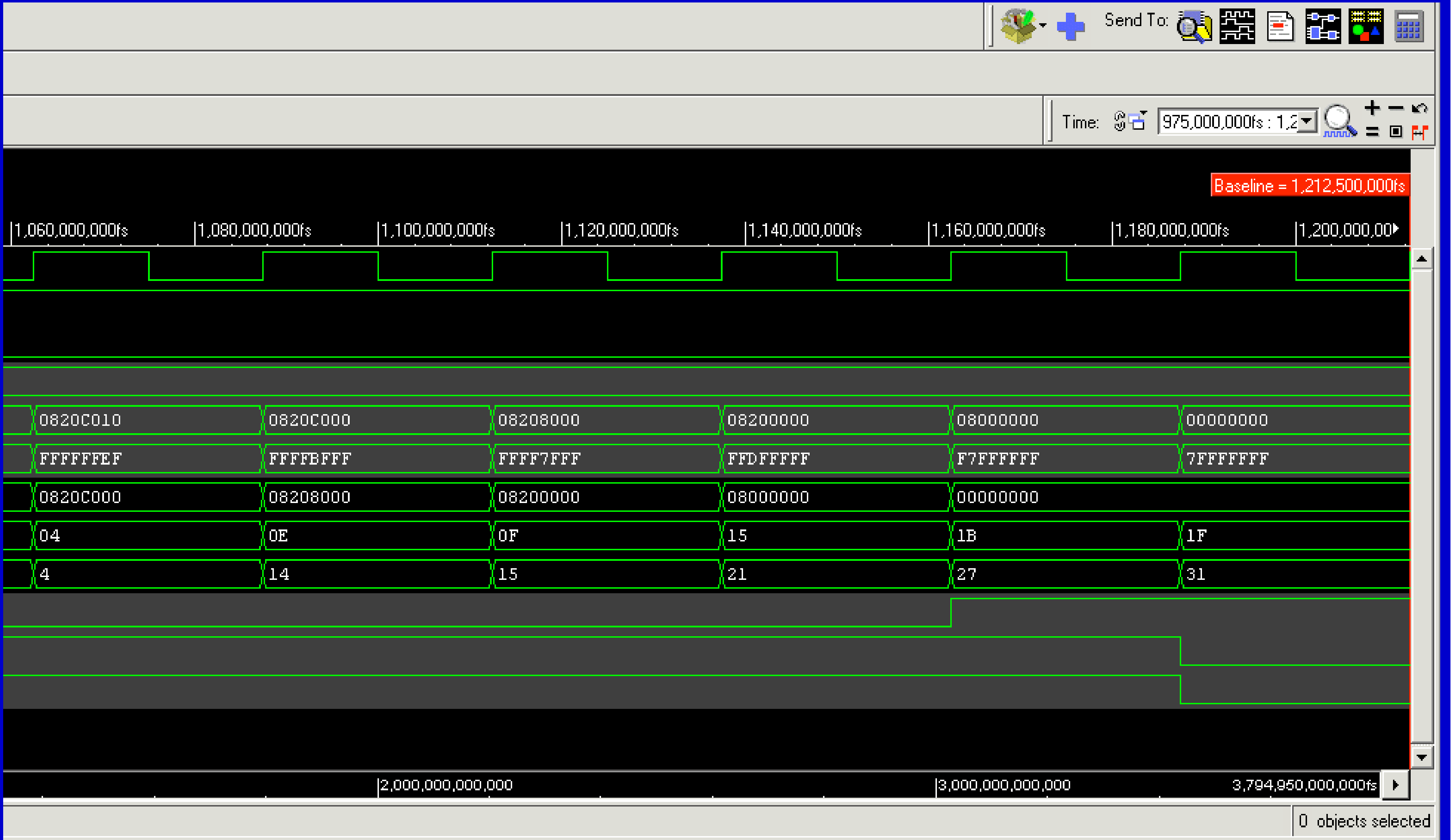
Name	Cursor				
clk	0				
reset_i	1				
new_value_available	0				
new_value	'h 000000	00000000	0820C01A	00000000	
data_encode	'h 000000	00000000	0820C01A	0820C018	0820C010
data_encode_actual	'h 7FFFFF	7FFFFFFF	FFFFFFFD	FFFFFFF7	FFFFFFEF
data_encode_next	'h 000000	00000000	0820C018	0820C010	0820C000
hit_address	'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



Position decoder – priority encoder



```
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
```

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

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
```

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

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 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
```

Priority encoder
Result in an FPGA from 2002: (Altera EP20k200FC484-3)
172 (out of 8320) logic elements -> more logic cells
33 registers

addressDecoder:	16
prior32:	54

11% (41/376) of pins

20.8 ns (48.0 MHz) data_encode -> state_encoding
-> slower state machine, but faster processing

```
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

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 -> more logic cells

20.8 ns (48.0 MHz) data_encode -> state_encoding
-> slower state machine, but faster processing

Slower and more logic can mean more elegant and effective

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?

```vhdl
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;
```

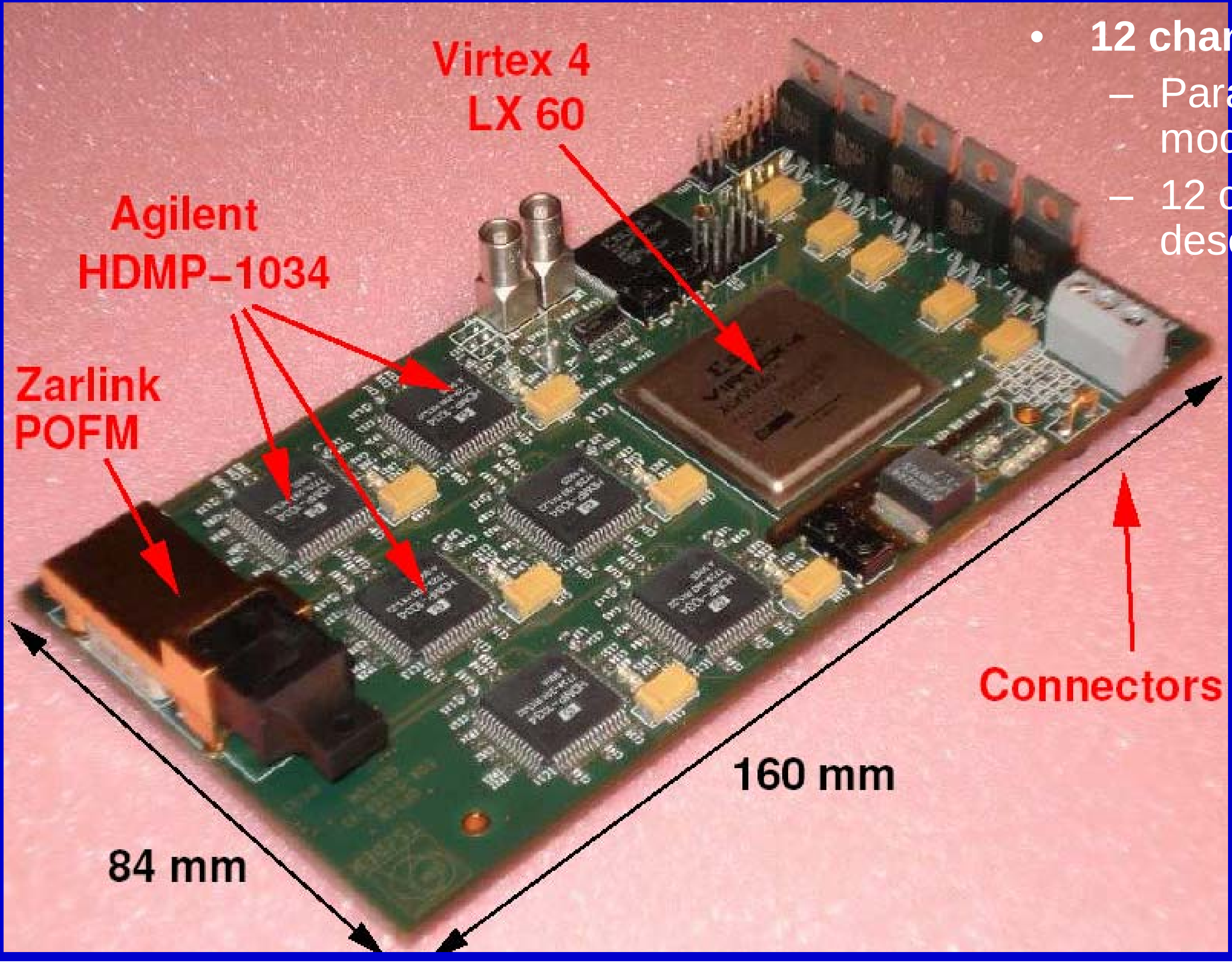
Position decoder – priority encoder

```
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

Processor board with optical inputs



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