Digital System Construction

Lecture 4: Math, memories, PRNG

Arithmetic

Memories

Pipelines and buffers

Pseudorandom numbers

IP core generation in Vivado

Introduction to Lab 3

Arithmetic with vectors

Different math libraries available, including:

- ieee.std_logic_unsigned.all
  - Simple arithmetic, vectors represent unisigned integers
  - Easy to use, but not an official standard
- ieee.numeric_std.all
  - Standard library, support for both unsigned and signed vector arithmetic
  - Adds `signed` and `unsigned` vector types
  - Higher learning curve
  - Recommended for new designs

Unsigned adder (example)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;

ENTITY adder16 IS
    PORT (X, Y          : IN   STD_LOGIC_VECTOR(15 DOWNTO 0) ;
          S             : OUT  STD_LOGIC_VECTOR(15 DOWNTO 0) ;
          Cout          : OUT  STD_LOGIC ) ;
END adder16 ;
```

Unsigned adder: architecture

```
ARCHITECTURE Behavior OF adder16 IS
    SIGNAL uX : UNSIGNED(15 DOWNTO 0);  -- 16 bits
    SIGNAL uY : UNSIGNED(15 DOWNTO 0);
    SIGNAL Sum : UNSIGNED(16 DOWNTO 0); -- 17 bits
BEGIN
    uX <= unsigned(X);
    uY <= unsigned(Y);
    Sum <= ('0' & uX) + ('0' & uY);
    S <= std_logic_vector(Sum(15 DOWNTO 0)) ;
    Cout <= Sum(16) ; -- Top bit is overflow/carry
END Behavior ;
```

cast to unsigned type

cast result to std_logic_vector

Representing signed numbers

- A vector can represent a signed or unsigned value
  - 8 bits unsigned: 0 to 255
  - 8 bits signed: -128 to +127
- Encoding signed numbers:
  - Top bit (MSB) is the sign (0=positive, 1=negative)
  - Positive numbers: simple binary representation
    - Decimal +53 = “00110101”
  - Negative numbers: “twos complement”
    - Invert the bits of the positive value, then add one
    - Decimal -53 = “11001010” + 1 = “11001011”

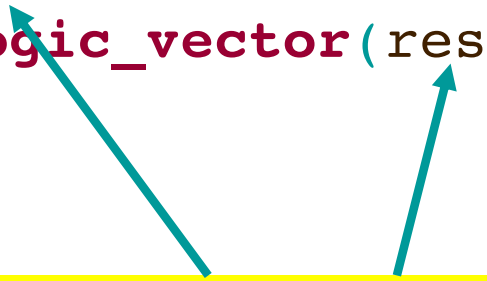
Signed adder (example)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;


ENTITY adder16 IS
    PORT (X, Y          : IN    STD_LOGIC_VECTOR(15 DOWNTO 0);
          S             : OUT   STD_LOGIC_VECTOR(15 DOWNTO 0)
    );
END adder16 ;
```

Signed adder: architecture

```
ARCHITECTURE Behavior OF adder16 IS
    SIGNAL Xs : SIGNED(15 DOWNTO 0);
    SIGNAL Ys : SIGNED(15 DOWNTO 0);
    SIGNAL Sum : SIGNED(16 DOWNTO 0) ;
BEGIN
    Xs <= signed(X);
    Ys <= signed(Y);
    Sum <= resize(Xs,Sum'LENGTH) + Ys;
    S <= std_logic_vector(resize(Sum,16)) ;
END Behavior ;
```



”resize” changes the length of a signed vector while keeping the sign in the MSB

ieee.numeric_std has overloaded operators

overloaded operator	description	data type of operand a	data type of operand b	data type of result
abs a - a	absolute value negation	signed		signed
a * b a / b a **mod** b a **rem** b a + b a - b	arithmetic operation	unsigned unsigned, natural signed signed, integer	unsigned, natural unsigned signed, integer signed	unsigned unsigned signed signed
a = b a /= b a < b a <= b a > b a >= b	relational operation	unsigned unsigned, natural signed signed, integer	unsigned, natural unsigned signed, integer signed	boolean boolean boolean boolean

Introduction to memories

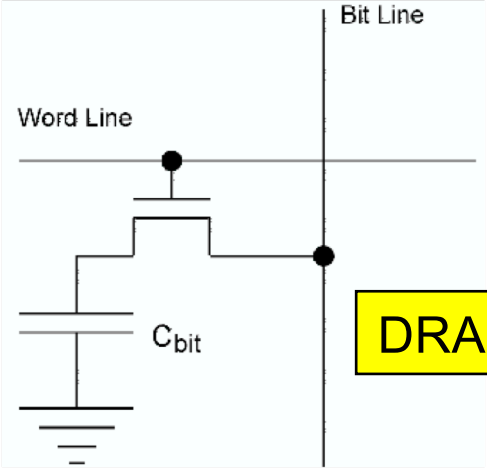
- Memories are essentially data storage arrays
  - One or more bits of data at each address
- Different kinds:
  - ROM (Read-Only Memory)
    - No write capability during normal operations
    - Some are re-writable (e.g. by applying higher voltage)
  - RAM (Random Access Memory)
    - Read and write, can directly access any address
- Volatility
  - Volatile memories lose data when powered off
  - Most RAMs are volatile, ROMs are non-volatile

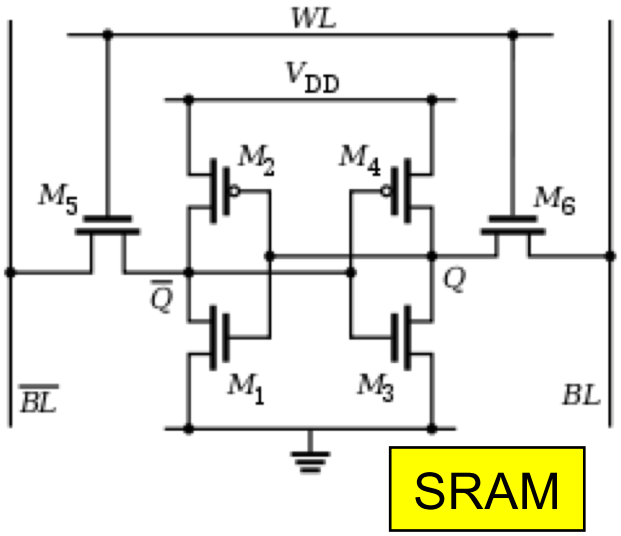
Random Access Memory (RAM)

- Addressable data storage array
  - Directly access contents of each address
  - Read and write operations supported
- About the same time to access any address
  - not quite true for modern DRAM
- Can be synchronous (clocked) or asynchronous
  - Block RAM on Xilinx FPGAs is synchronous

Static vs. Dynamic RAM

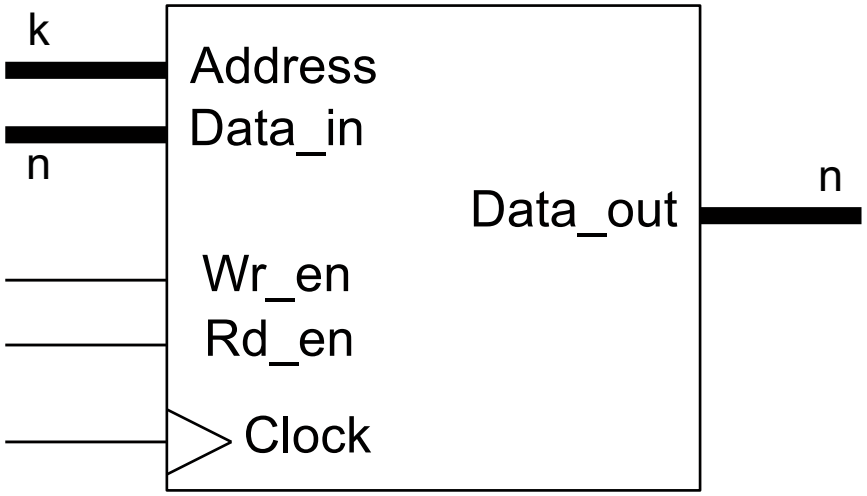
- DRAM: charge stored in quantum wells (similar to capacitors)
  - Small and inexpensive (good)
  - Data disappears if the contents are not refreshed (less good)
- SRAM: data stored in latches
  - Stable storage w/o refresh (good)
  - Several transistors per data bit
    - More expensive
    - Lower bit density
- Most FPGA block RAM is SRAM





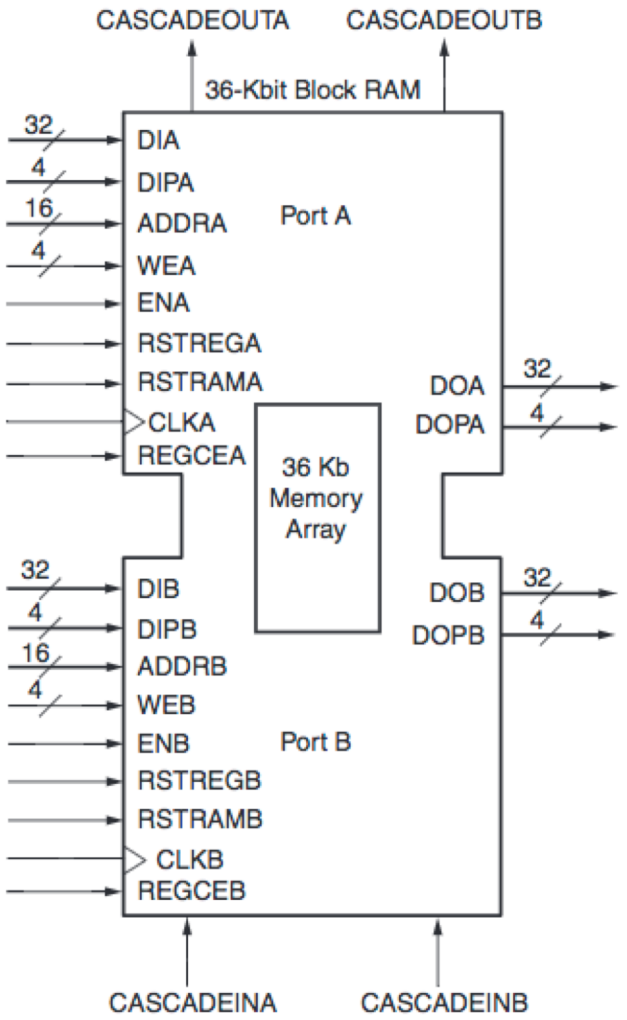
Single-port RAM block

- Word size: *n* bits
- Capacity: 2^k words
- Synchronous RAM is clocked
- To write:
  - Set target address and data to be written
  - Set Wr_en and wait for the next clock edge
- To read:
  - Set target address to be read and Rd_en
  - Data at address available at the next clock edge



Xilinx 7-series block RAM

- Dual port RAM
  - Two ports can independently access memory contents
  - Can have different address and data widths to access the same number of of total bits
- Synchronous (clocked) read and write operations
  - Possible to use different clocks for each port (if you want to)



Common RAM applications

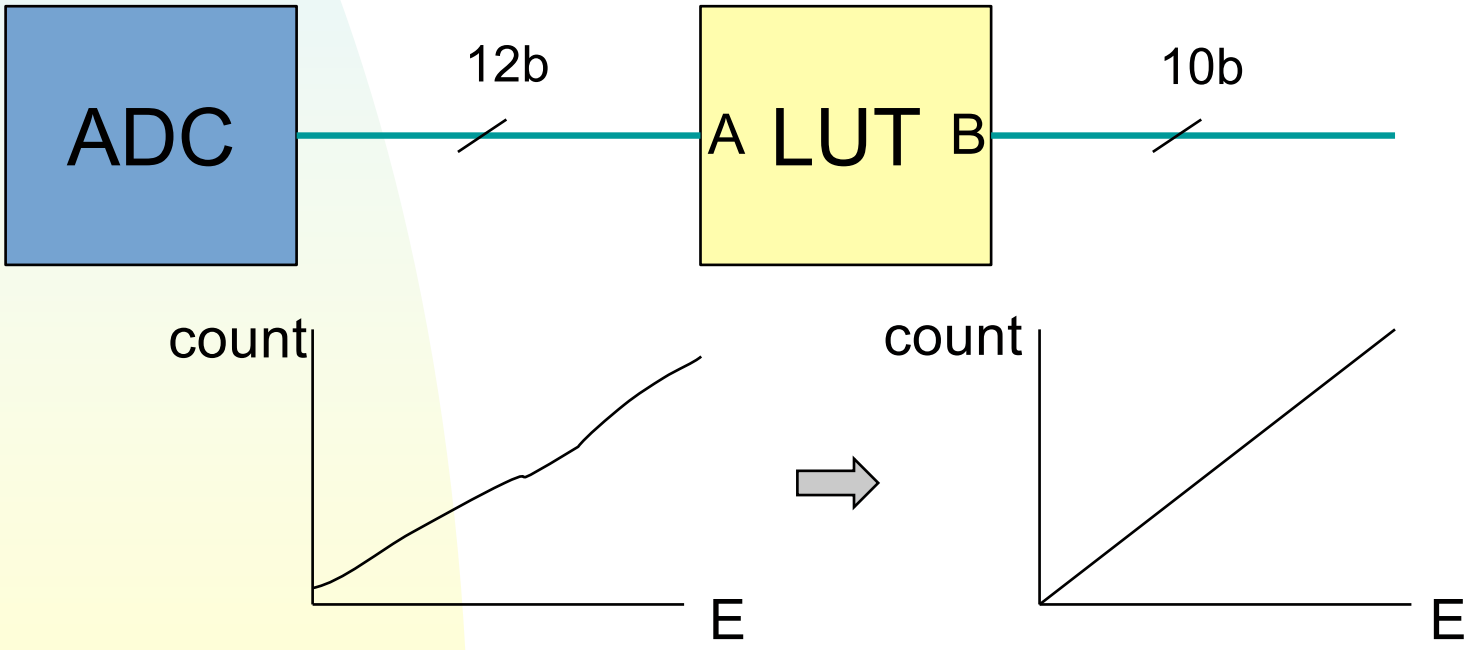
- Local memory for embedded CPUs (advanced)
- Fast, flexible implementation of complex algorithms
  - Memory look-up tables (LUT)
  - Content-addressable memory (advanced)
- Diagnostics and testing
  - Capture data and read it out (“spy memory”)
  - Feed test patterns through logic (“playback memory”)
- Data buffering for temporary storage and readout
  - Synchronous memory buffer (“pipeline”)
  - Asynchronous buffer (“FIFO”)

Memory lookup table (LUT)

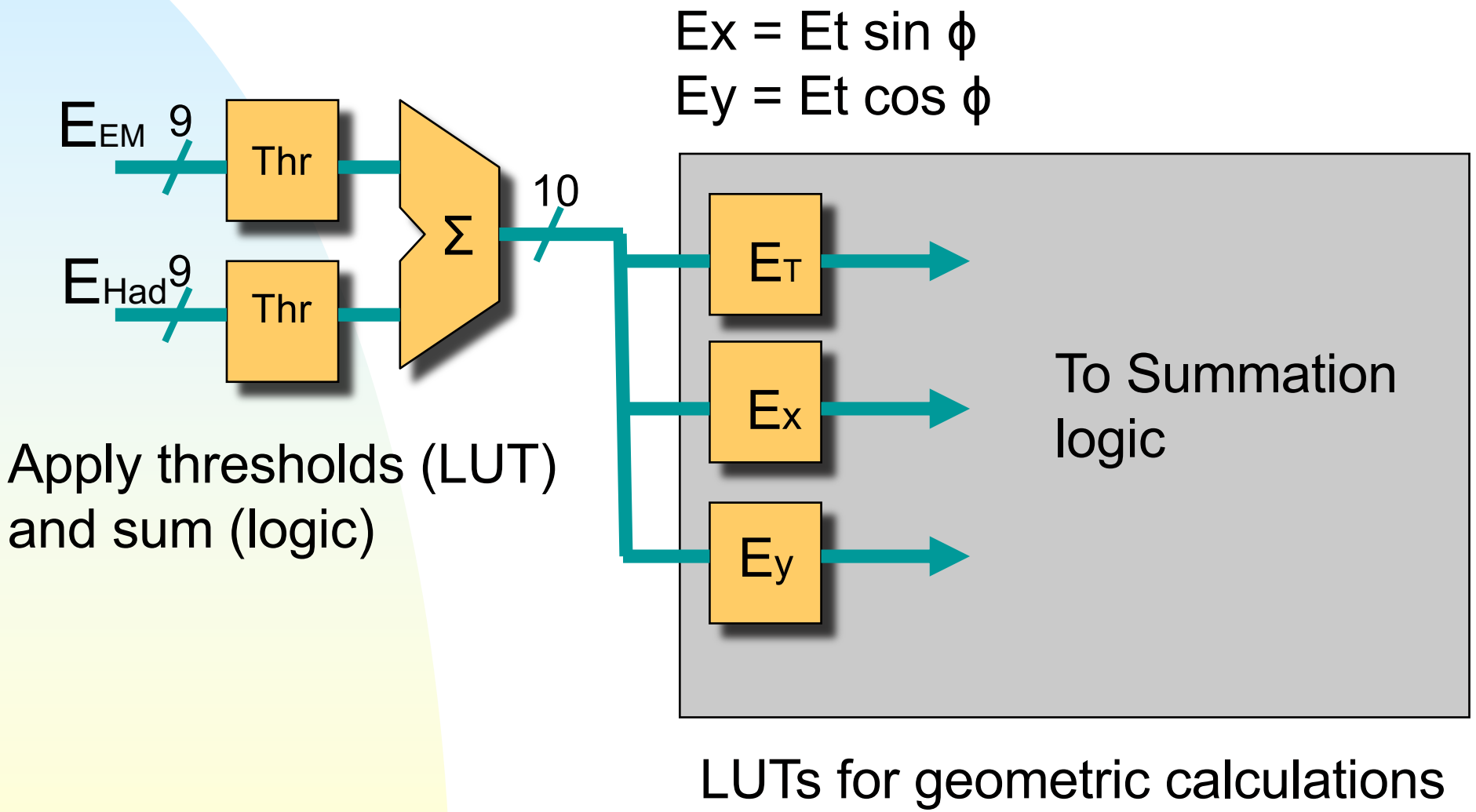
Common application: Use LUT to correct raw data

- calibration, linearity, etc.

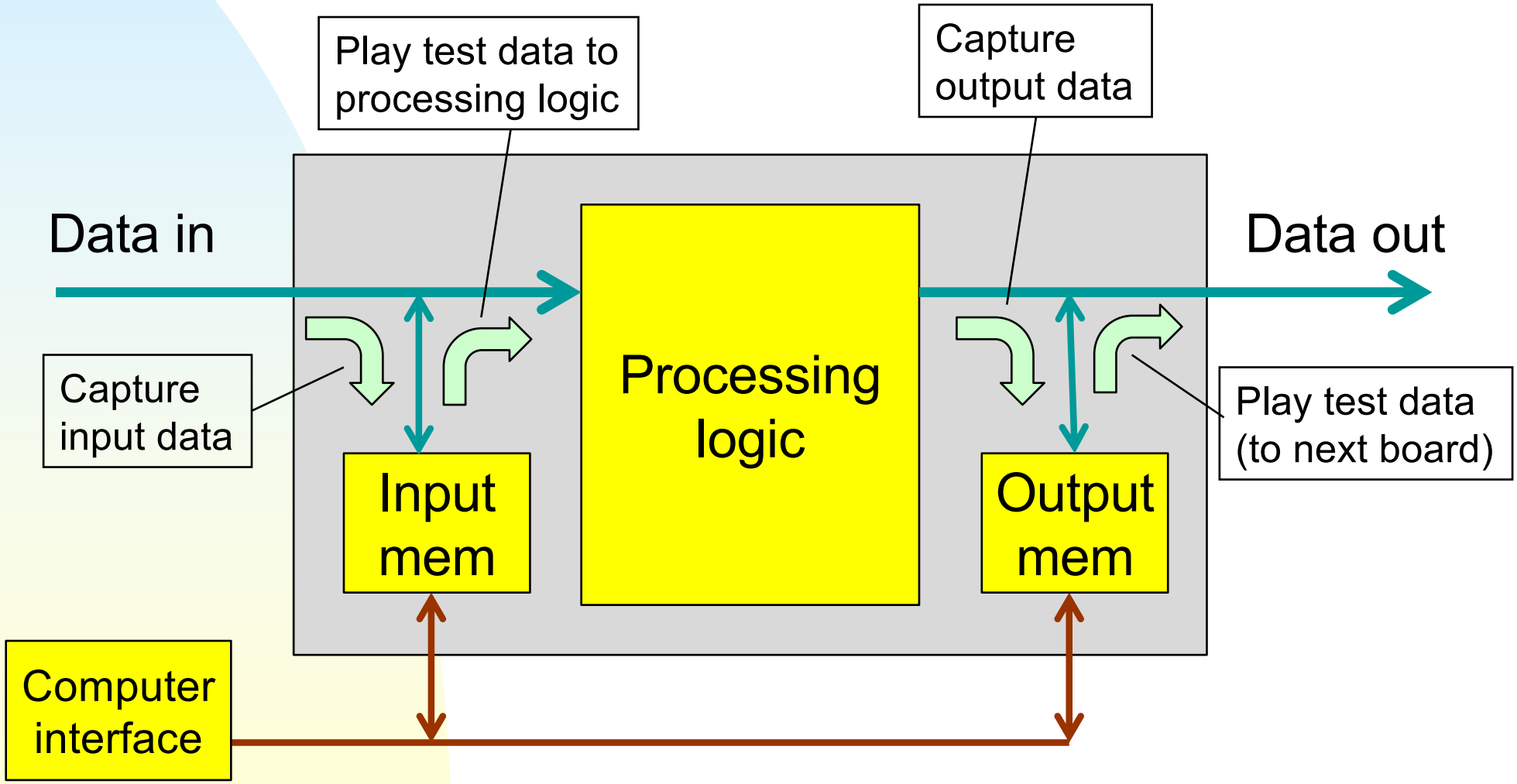
Can also do geometric and other calculations quickly



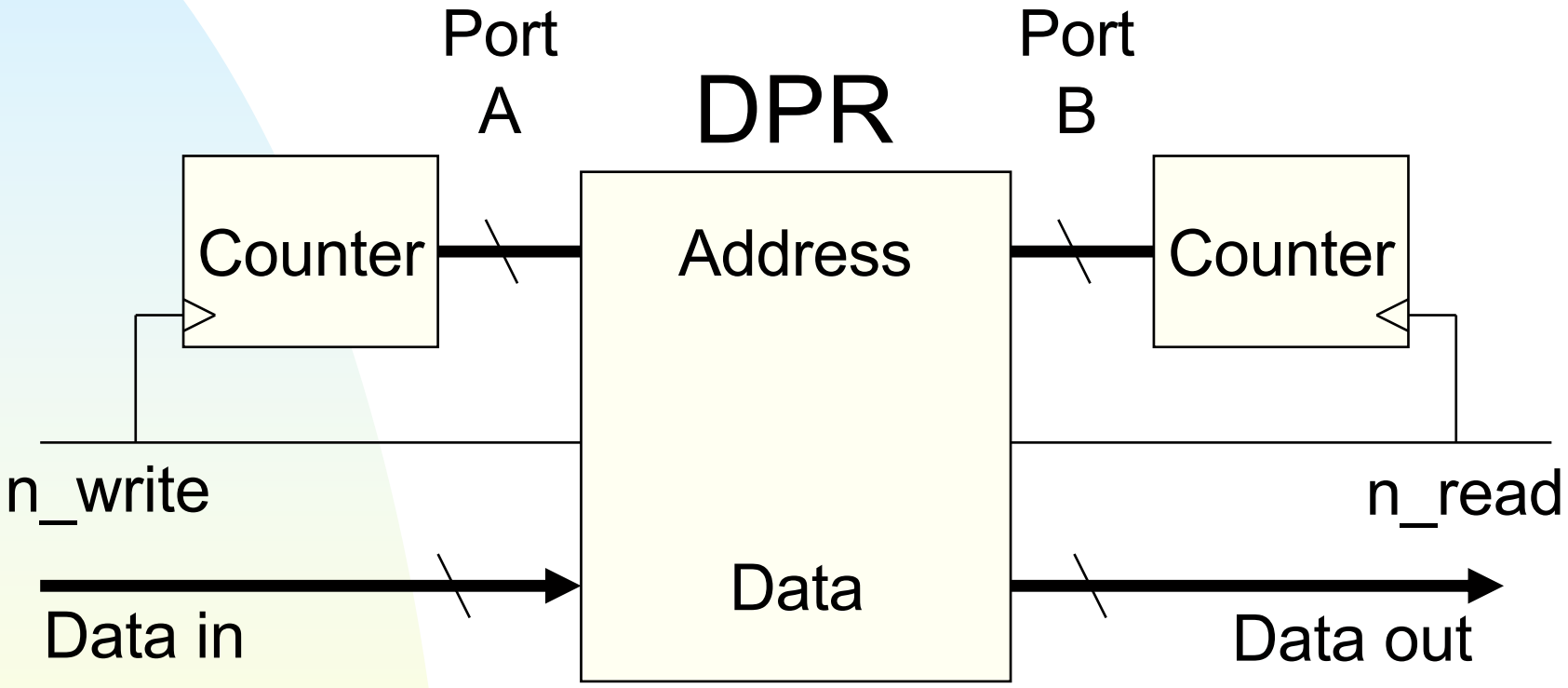
ATLAS L1Calo example



Spy/playback memories

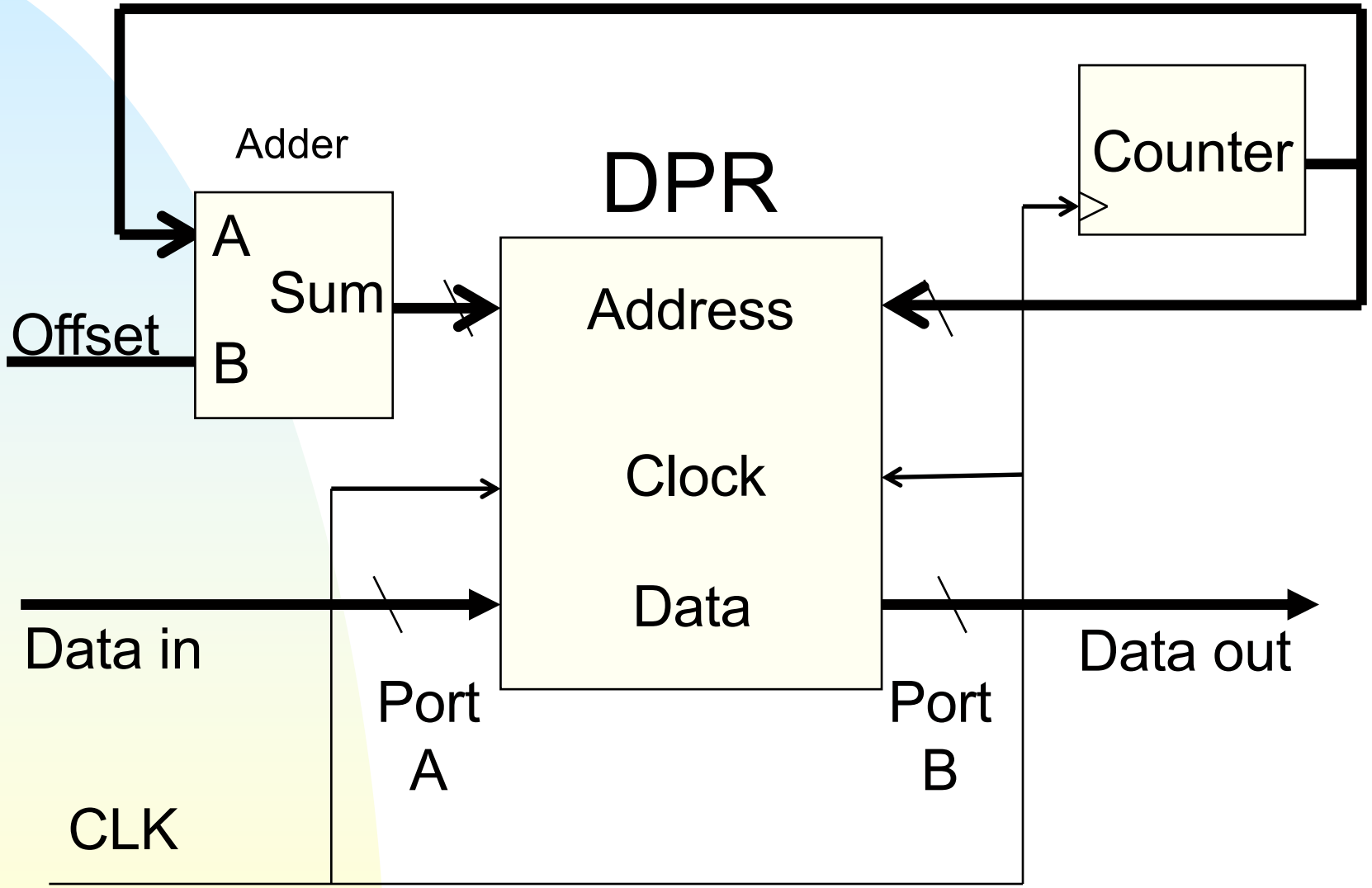


Asynchronous data buffer



Simplified diagram. Write to port A and read from port B. Address of each port incremented during read/write

Synchronous delay buffer

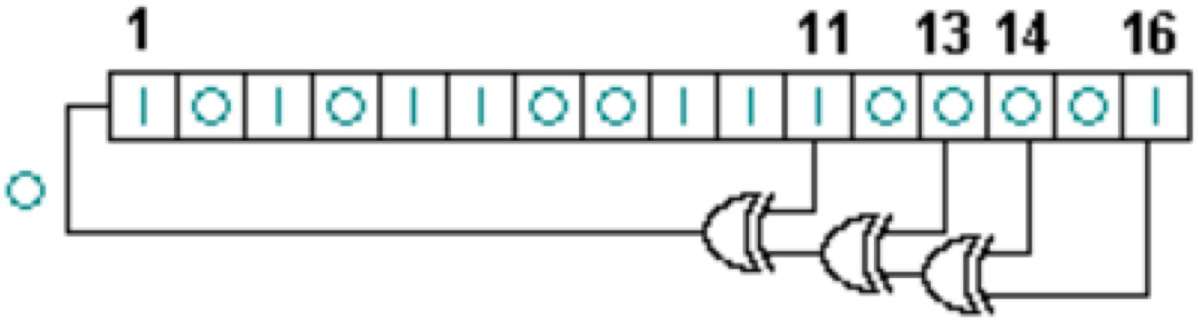


Other topics for Lab 3

- Pseudorandom number generation
  - You will use this to test a synchronous data pipeline with adjustable length
- Implementing device-specific features
  - IP Catalog in Vivado
  - You will use this to implement a dual-port RAM

Pseudorandom number generator

- Useful to test designs with many possible inputs
- Basic concept:
  - Start with a random sequence of bits circulating in a shift register ('seed')
  - Continuously change the contents of the seed
    - use XOR of other bits (to maintain 1/0 balance)



Single-bit PNRG: entity

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity prng is
   generic(init_seed : std_logic_vector (15 downto 0)
           := "0101110110010101");
   port(   clk : in std_logic;
           random : out std_logic);
end prng;
```

need a different seed for each PNRG

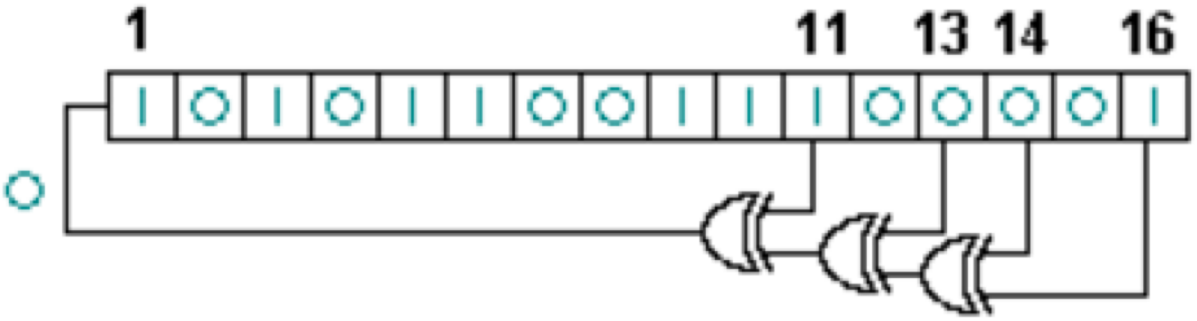
Single-bit PNRG: architecture

```
architecture prng_arch of prng is
   signal seed : std_logic_vector (15 downto 0) := init_seed;
begin
   gen_number: process(clk)
   begin
      if rising_edge(clk) then
       seed <= seed(14 downto 0) &
             (seed(15) xor seed(13) xor seed(12) xor seed(10));
      end if;
   end process;

  random <= seed(15);

end architecture;
```

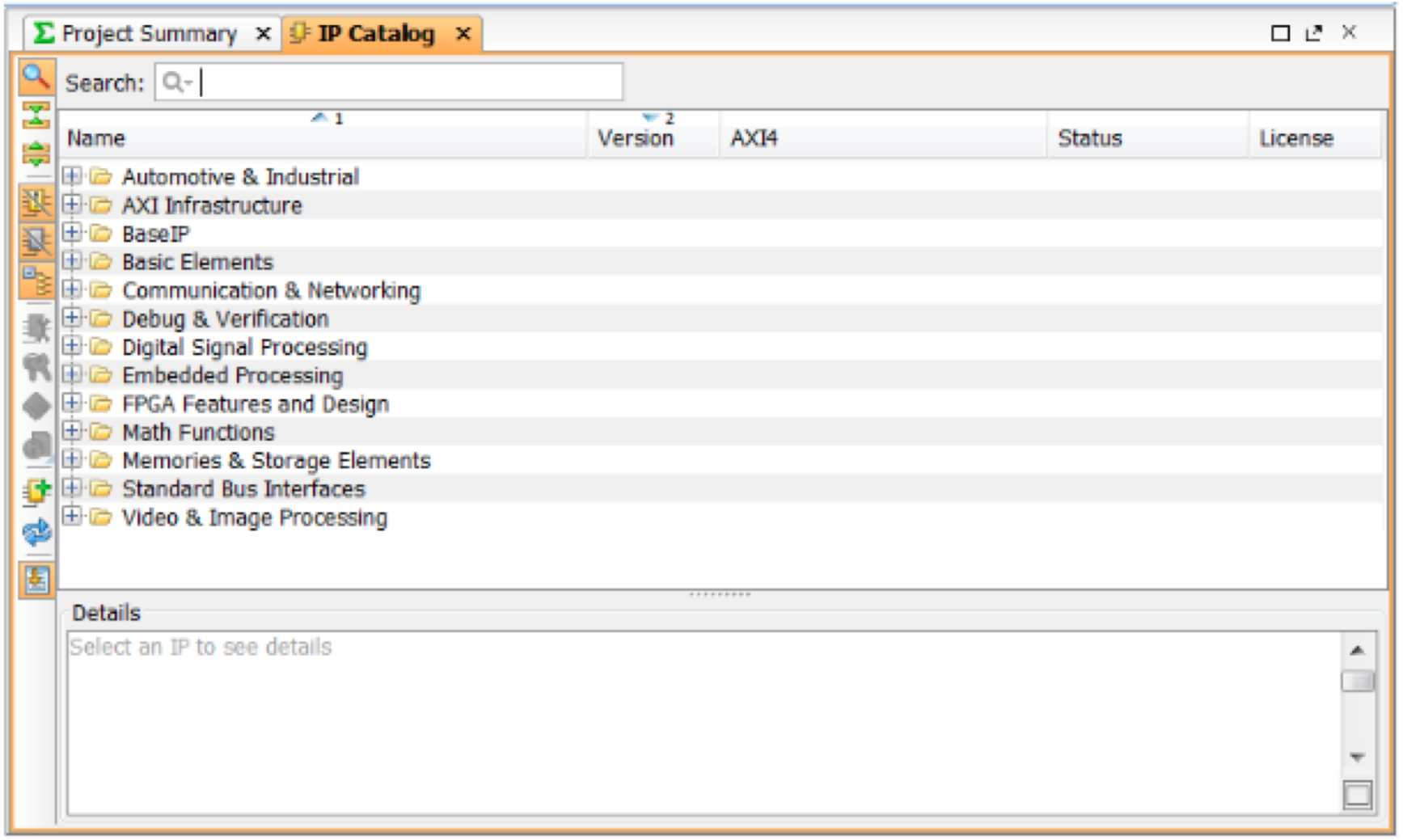
Maximum-length LFSR
"linear feedback shift register"



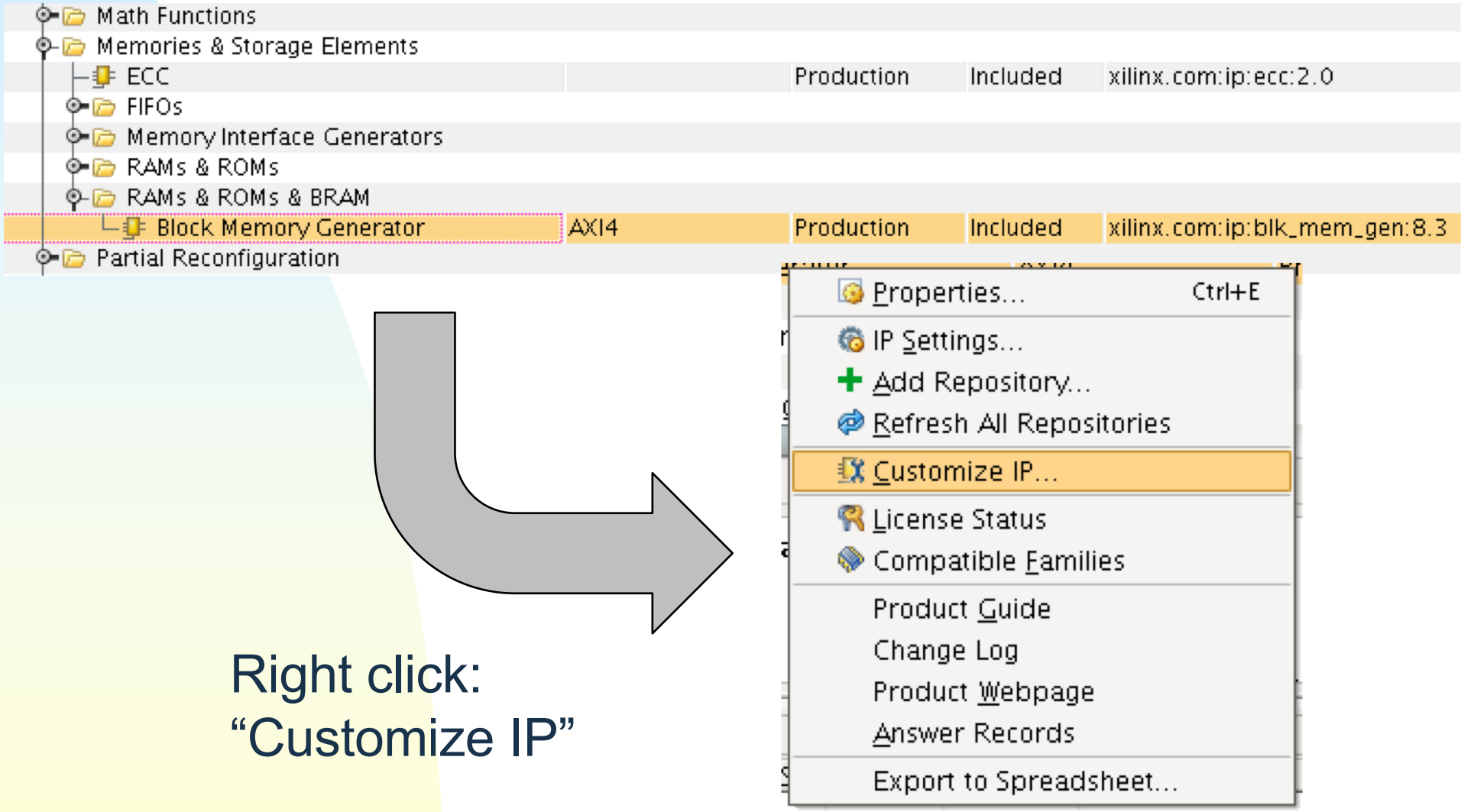
Generating IP cores in Vivado

- General-purpose design elements can be well-described with pure VHDL code
- But device-specific features can be more complicated to specify and declare:
  - Block memories, dedicated multipliers, digital clock managers, embedded CPUs, serial transceivers, etc.
- FPGA Vendors often provide tools to generate design modules that you can customize for your own needs
  - Xilinx Vivado: IP Catalog

Launching IP catalog

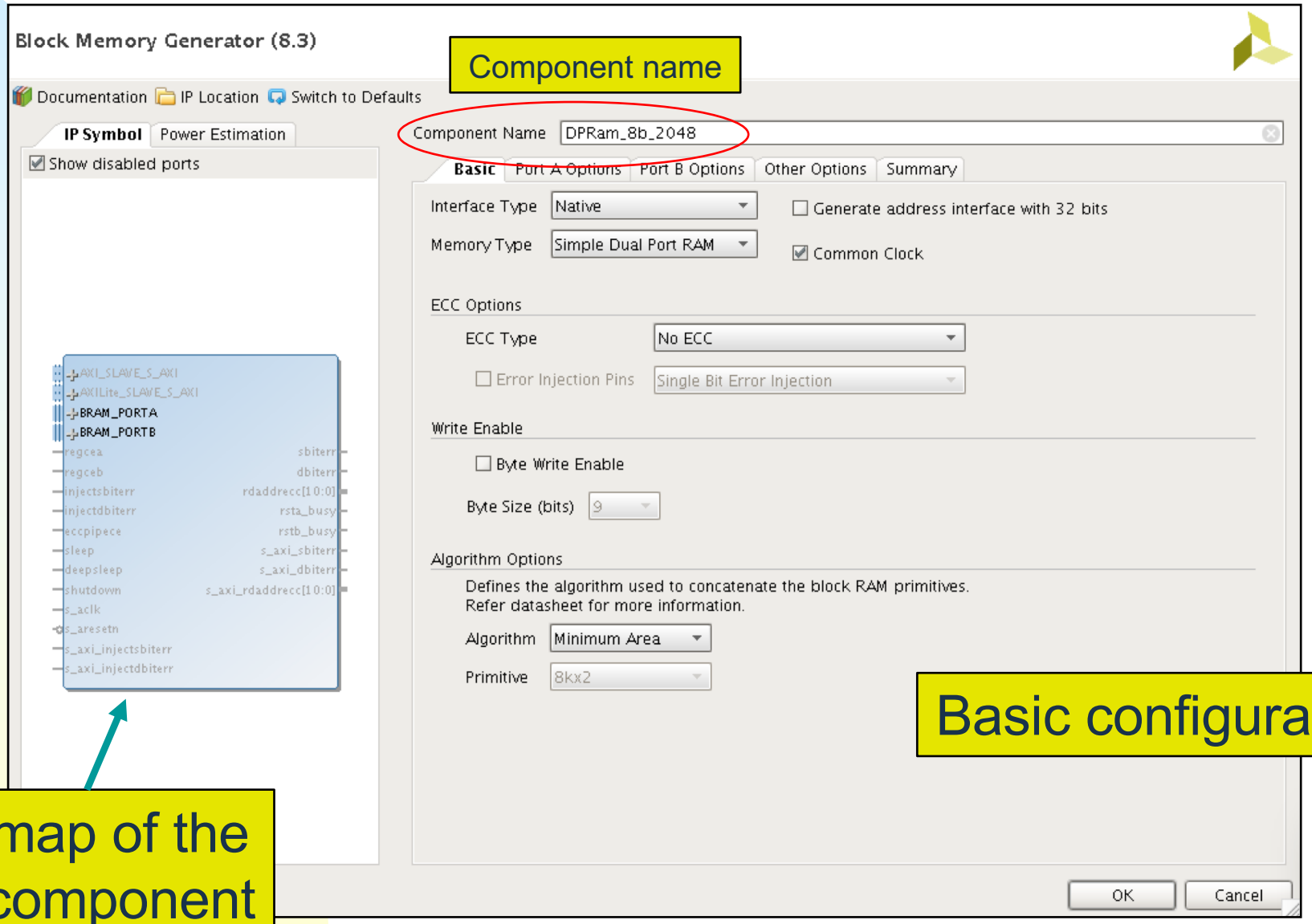


Select the core you want



Right click:
“Customize IP”

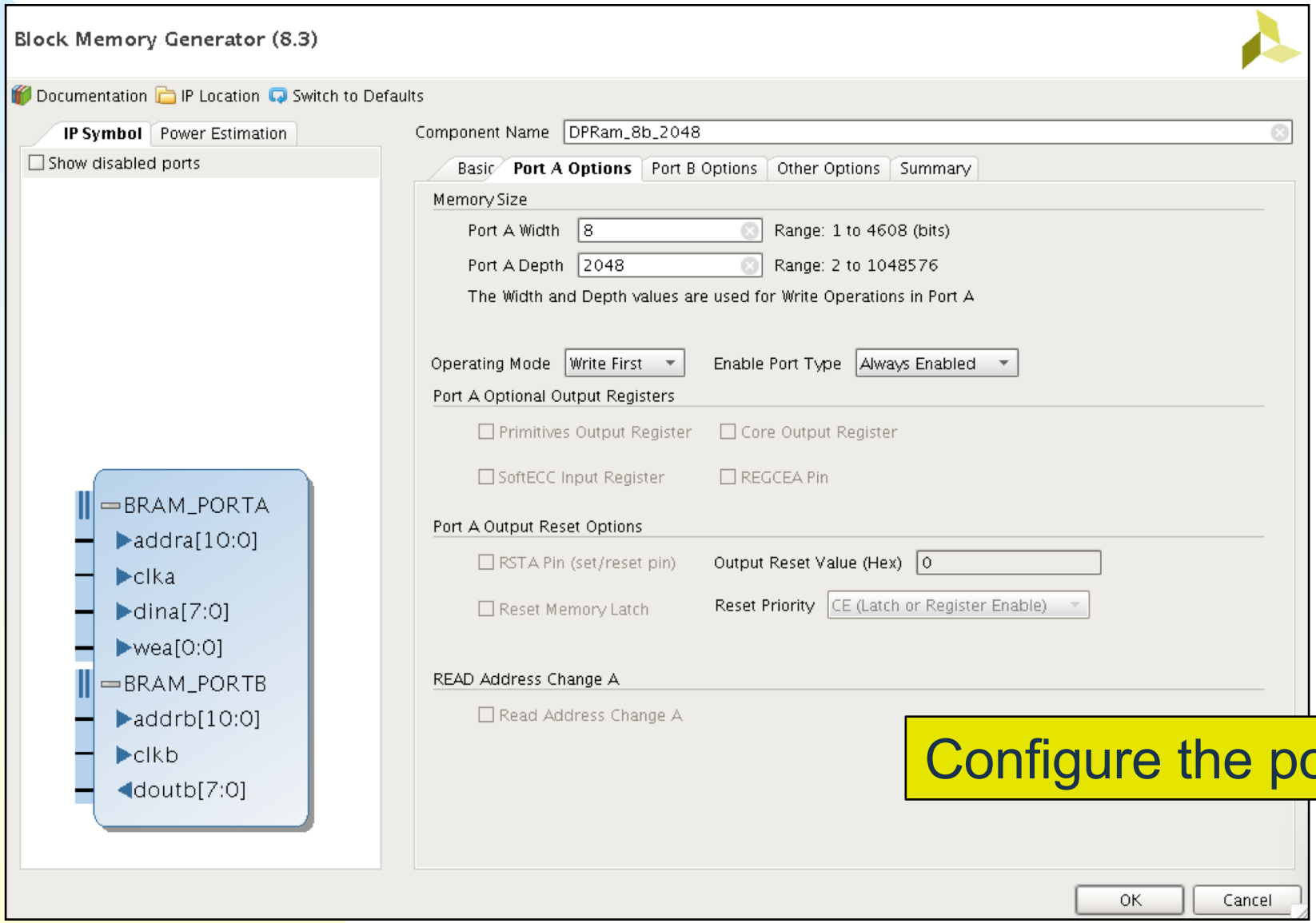
Customize the component



Port map of the final component

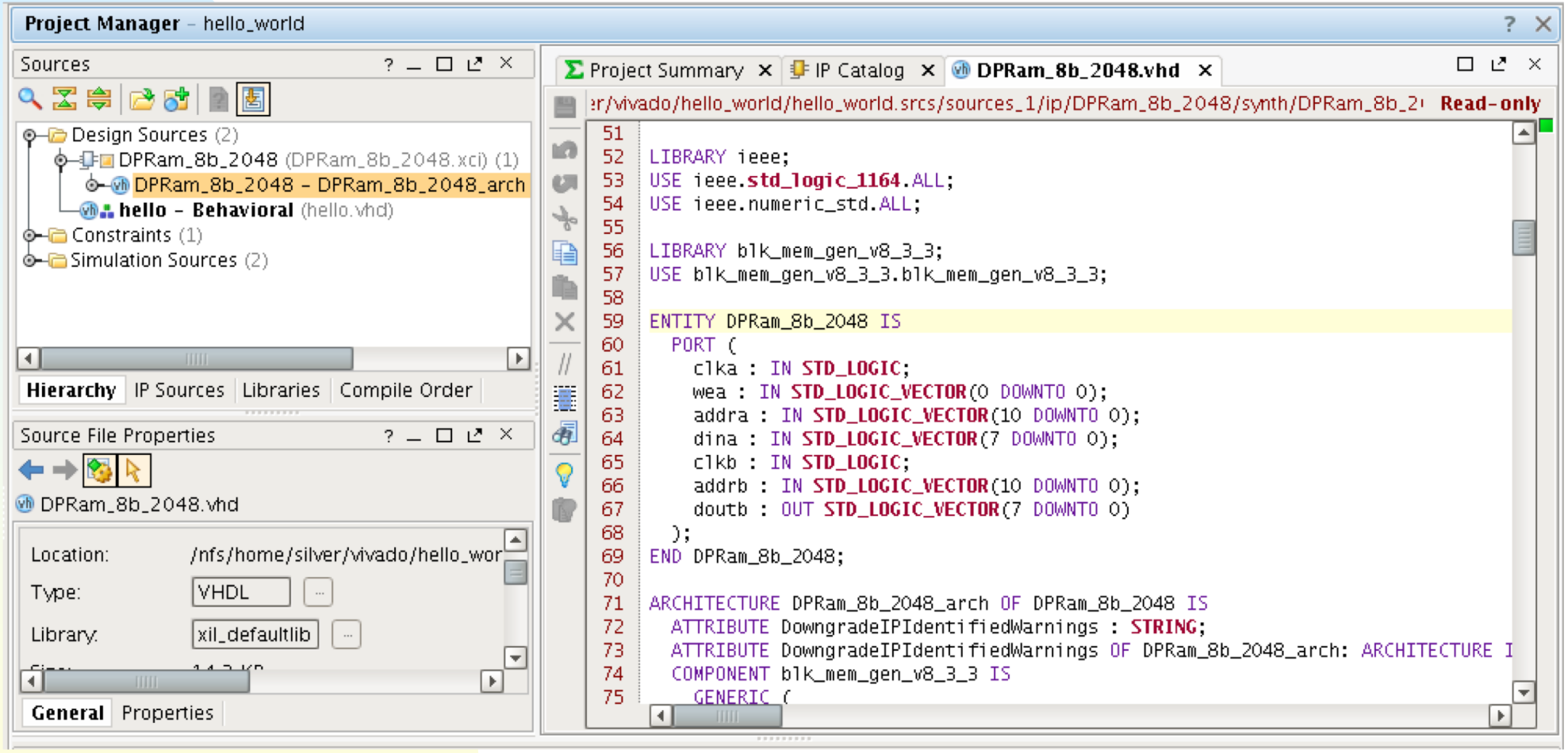
Basic configuration

Customize the component



Configure the ports

Generated component



Lab 3:

- Design a 4-bit PRNG
- Design a programmable-delay pipeline buffer with a block RAM
- Connect the PRNG output to the input of the buffer input and observe the delay between input and output
- Test in simulation and hardware