Embedded System Design for Machine Learning

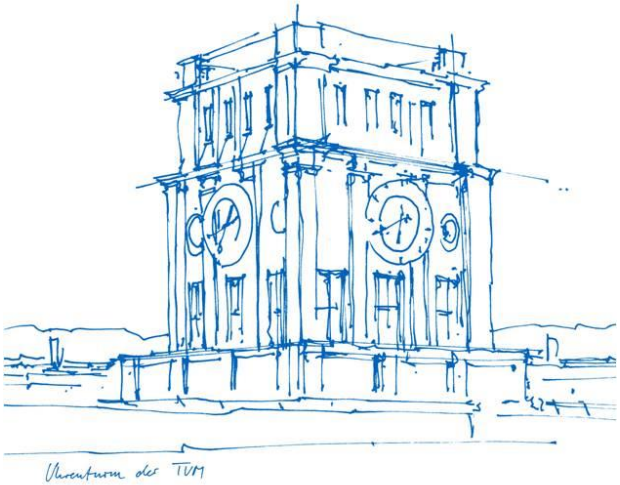
Lab: Vectorized Multiply

Contributors:

Wolfgang Ecker,

Sathya Ashok,

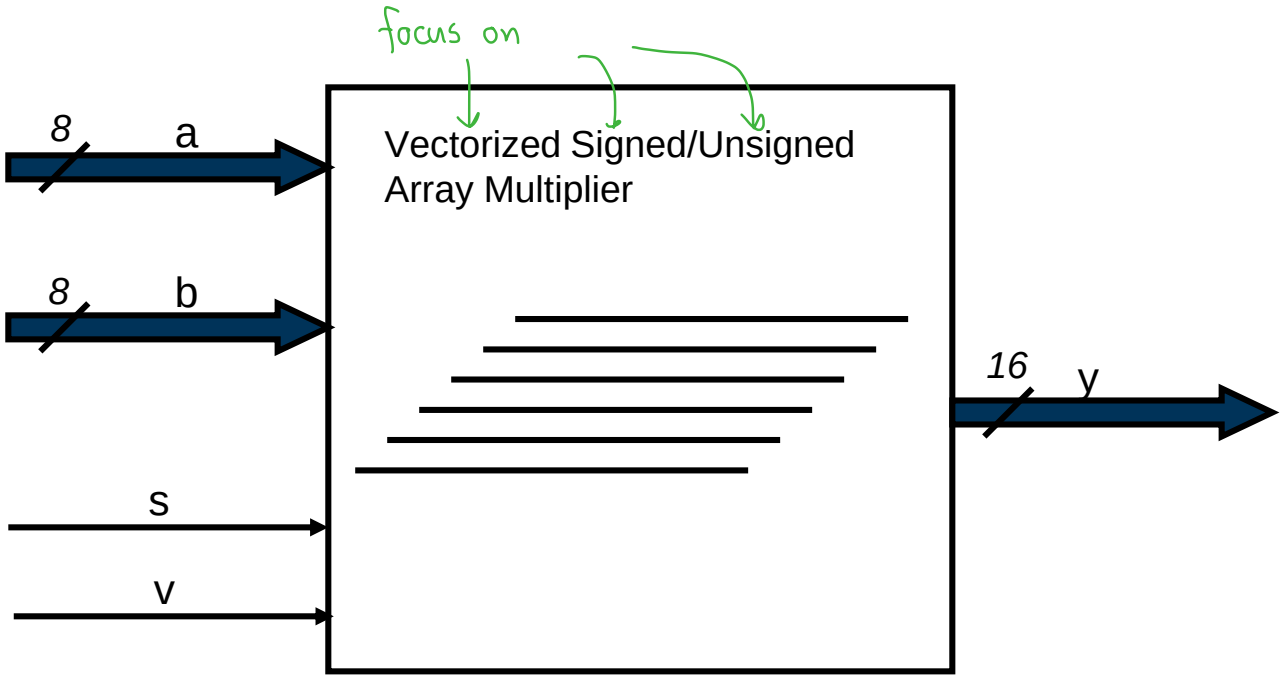
Sebastian Prebeck



What we will cover

- Lab Description
- VHDL Primer
- Vivado Introduction

.txt file


focus on
8
a
8
b
Vectorized Signed/Unsigned
Array Multiplier
16
y
s
v

```
entity multiplier is
   port(
      A : in bit_vector( 7 downto 0);
      B : in bit_vector( 7 downto 0);
      S : in bit;
      V : in bit;
      Y : out bit_vector( 15 downto 0)
   );
end multiplier;
```

Spec

signed (s)	Vectored (v)	Function
0	0	`Y = UNSIGNED(A) * UNSIGNED(B)`
0	1	`Y = UNSIGNED(A[7...4]) * UNSIGNED(B[7...4]) &` `UNSIGNED(A[3...0]) * UNSIGNED(B[3...0])`
1	0	`Y = SIGNED(A) * SIGNED(B)`
1	1	`Y = SIGNED(A[7...4]) * SIGNED(B[7...4]) &` `SIGNED(A[3...0]) * SIGNED(B[3...0])`

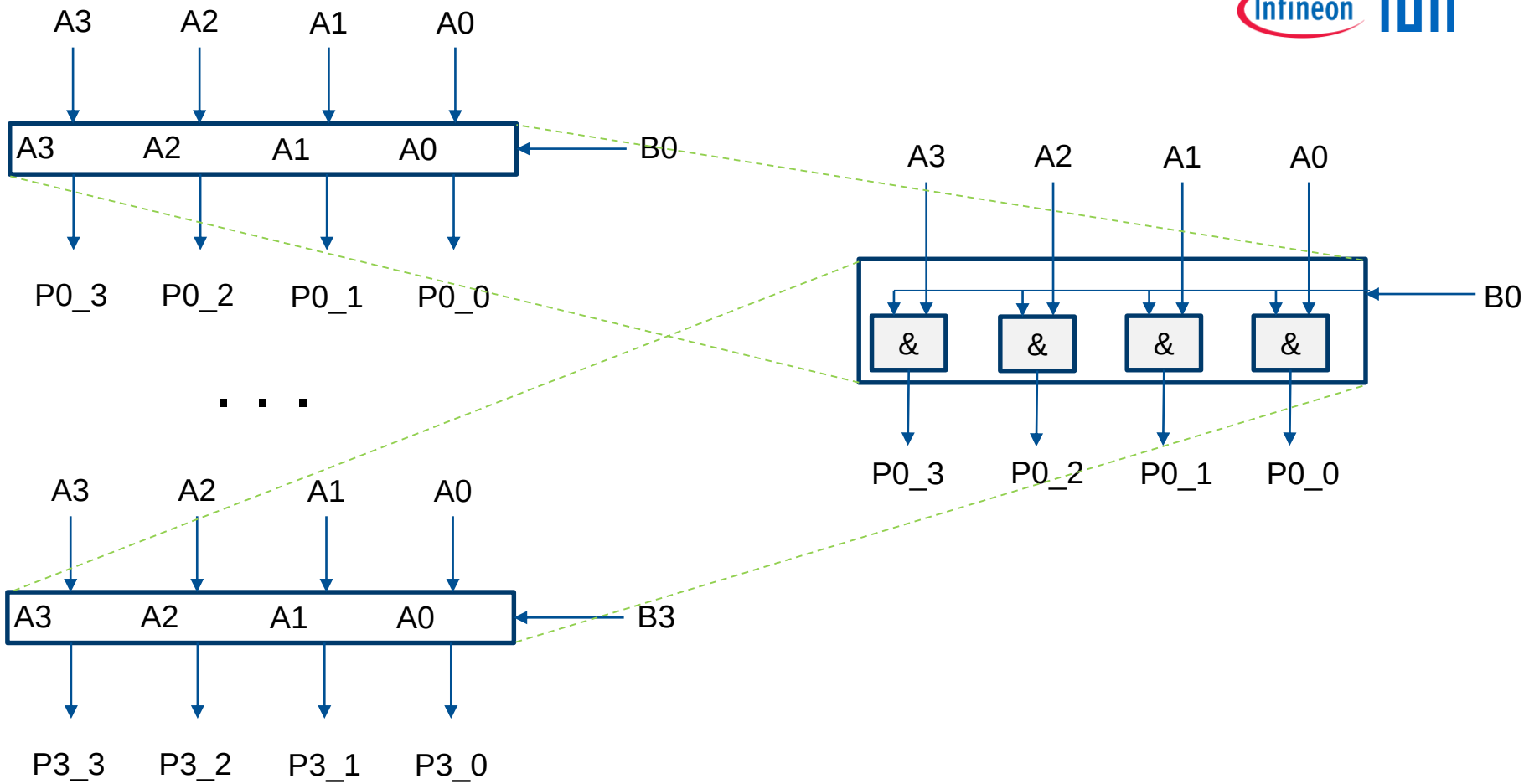
```
library ieee;
use ieee.mumeric_bit.all;

architecture behavioral of multiplier is
begin
   Y <= bit_vector( unsigned(A) * unsigned(B) )
           when s ='0' AND v = '0' else
        bit_vector(   signed(A) *   signed(B) )
           when s ='1' AND v = '0' else
        bit_vector( unsigned(A(7 downto 4)) * unsigned(B(7 downto 4)) ) &
        bit_vector( unsigned(A(3 downto 0)) * unsigned(B(3 downto 0)) )
           when s ='0' AND v = '1' else
        bit_vector(   signed(A(7 downto 4)) *   signed(B(7 downto 4)) ) &
        bit_vector(   signed(A(3 downto 0)) *   signed(B(3 downto 0)) );
end behavioral;
```

Architectural Aspects

1. Build an array multiplier (no booth encoding, no tree structure)
    1. Compute partial products by bit x bit or bit x bit_vector multiplication
    2. Use RCA or CSA (carry save/carry propagate) adders
2. Extend the adder to do be able to consider the sign-bit (two‘s complement) properly
3. Introduce vectorization
    - Split and connect the adders in a way so that they add the approproate partial producs
    - Multiplex the adder outputs

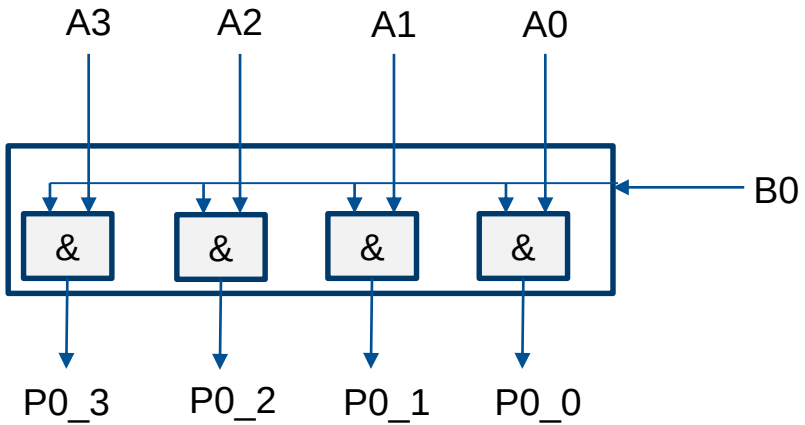
Lab Submission

1. Provide your results in ASCII only, i.e. just provide the VHDL code of the design and the testbench. Testbench and RTL Multiplier in two different files.
2. The points are given as follows
   - simple multiplier that multiplies two unsigned 8-bit numbers **(10P)**
   - multiplier that can either multiply signed and unsigned **(5P)**
   - multiplier that supports vectorized/non-vectorized **(5P)**
   - code style / following best practices **(5P)**
   - meaningful testbench provided **(5P)**


A3
A2
A1
A0
A3
A2
A1
A0
B0
P0_3
P0_2
P0_1
P0_0
A3
A2
A1
A0
B0
&
&
&
&
P0_3
P0_2
P0_1
P0_0
A3
A2
A1
A0
A3
A2
A1
A0
B3
P3_3
P3_2
P3_1
P3_0

no processes !

only concurrent signal assignments




```vhdl
-- VHDL
P0_3 <= B0 AND A3;
P0_2 <= B0 AND A2;
P0_1 <= B0 AND A1;
P0_0 <= B0 and A0;
```

```vhdl
-- VHDL
for i in 3 downto 0 generate
   P0(i) <= B(0) AND A(i);
end generate;
```

```
-- VHDL with generate
for i in 3 downto 0 generate
   P0(i) <= B(0) AND A(i);
   P1(i) <= B(1) AND A(i);
   P2(i) <= B(2) AND A(i);
   P3(i) <= B(3) AND A(i);
end generate;
```

```
-- VHDL with conditional signal
--    assignment
   P0 <= B when A(i) = '1' else
         "0000";
```

```
type Ptype is
   array(integer range 3 downto 0,
         integer range 3 downto 0)
      of bit;
signal P : Ptype;
--
for j in 3 downto 0 generate
   for i in 3 downto 0 generate
      P(j,i) <= B(0) AND A(i);
      P(j,i) <= B(1) AND A(i);
      P(j,i) <= B(2) AND A(i);
      P(j,i) <= B(3) AND A(i);
   end generate;
end generate;
```

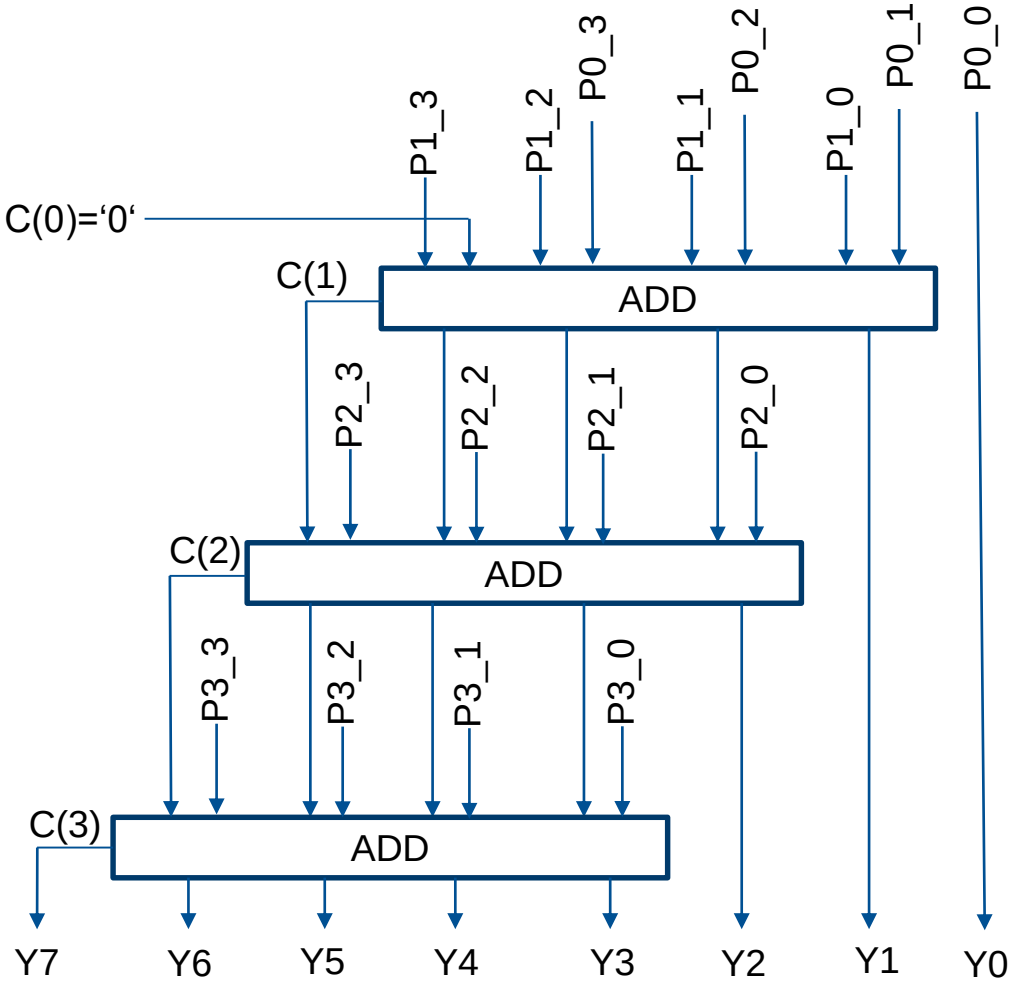
Not supported by all synthesis tools

```
type Ptype is
   array(integer range 3 downto 0,
         integer range 3 downto 0)
      of bit;
signal P : Ptype;
--
for j in 3 downto 0 generate
   for i in 3 downto 0 generate
      P(j,i) <= B(j) AND A(i);
end generate;
end generate;
```

Not supported by all synthesis tools

```
signal P :
   bit_vector(15 downto 0);
--
for j in 3 downto 0 generate
   for i in 3 downto 0 generate
      P(j*4+i) <= B(j) AND A(i);
end generate;
end generate;
```

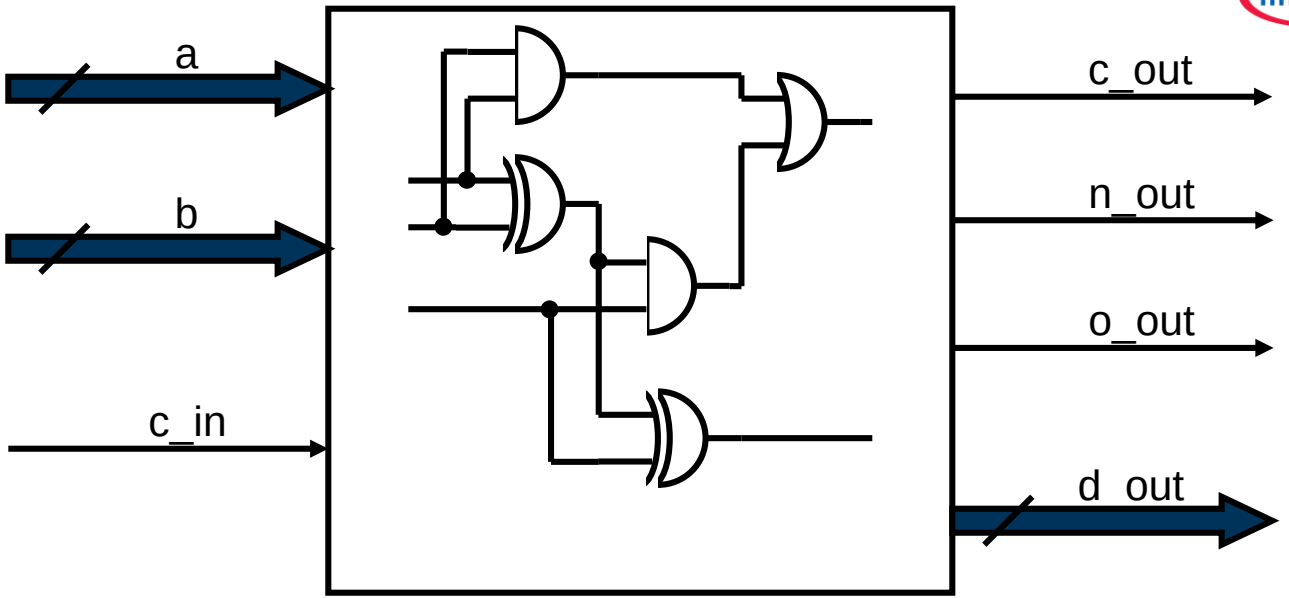
2D array to 1D vector mapping



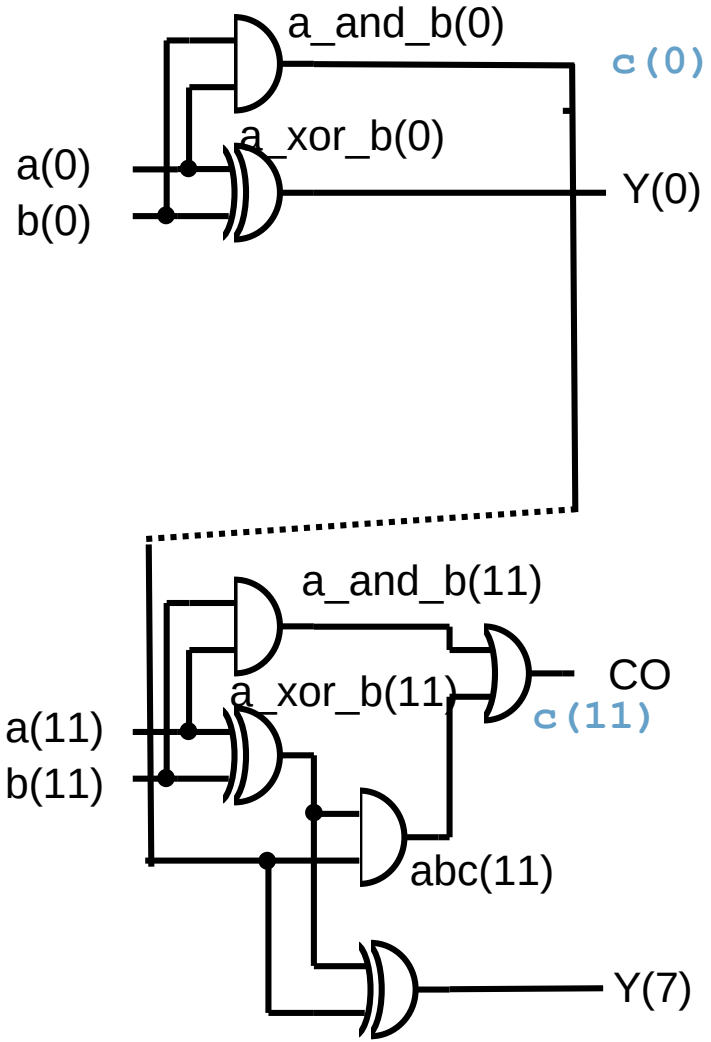
Ripple Carry Adder?

```
Y(0) <= A(0) AND B(0);
--
A1 <= '0' & A( 7 downto 1) when B(0) = '1' else "00000000";
B1 <= A when B(1) = '1' else "00000000";
X1: entity work.adder(dataflow) port map( A1, B1, S1, C1 );
Y(1) <= S1(0);
--
A2 <= C1 & S1(7 downto 1);
B2 <= A when B(2) = '1' else "00000000";
X2: entity work.adder(dataflow) port map( A2, B2, S2, C2 );
Y(2) <= S2(0);
--
-- ...
--
A7 <= C6& S6(7 downto 1);
B7<= A when B(7) = '1' else "00000000";
X7: entity work.adder(dataflow) port map( A7, B7, S7, C7 );
Y(14 downto 7 ) <= S7;
Y(15) <= C7;
```

Already 8 bit solution




```
entity adder is
   port(
      A, B : in  bit_vector(7 downto 0);
      Y    : out bit_vector(7 downto 0);
      CO   : out bit);
end adder;
```



only use gates and mux
no + or *

```
architecture dataflow of adder is
   signal a_and_b, a_xor_b, abc, c:
      bit_vector(7 downto 0 );
begin
   g1:for i in 0 to 7 generate
      a_and_b(i) <= a(i) AND b(i);
      a_xor_b(i) <= a(i) XOR b(i);
      g2:if i = 0 generate -- half adder
         y(i) <= a_xor_b(i);
         c(i) <= a_and_b(i);
      end generate;
      g3:if i /= 0 generate -- full adder
         abc(i) <= c(i-1) AND a_xor_b(i);
         y(i) <= c(i-1) XOR a_xor_b(i);
         c(i) <= a_and_b(i) OR abc(I);
      end generate;
   end generate;
   CO <= c(7);
end dataflow;
```

```
entity multipliers_test is end multipliers_test;

architecture Behavioral of multipliers_test is
   signal A, B   : bit_vector( 7 downto 0);
   signal S, V   : bit;
   signal Y1, Y2 : bit_vector( 15 downto 0);
begin
   process
   begin
      S <= '0'; V <= '0';
      for AI in 0 to 2**8-1 loop
         A <= bit_vector( to_unsigned( AI, 8 ) );
         for BI in 0 to 2**8-1 loop
            B <=  bit_vector( to_unsigned( BI, 8 ) );
            wait for 1 ns;
            assert Y1 = Y2;
         end loop;
      end loop;
      wait;
   end process;
   UUT1: entity work.multiplier(behavioral) port map(A,B,S,V,Y1);
   UUT2: entity work.multiplier(dataflow) port map(A,B,S,V,Y2);
end Behavioral;
```

What we will cover

- Lab Description
- **VHDL Primer**
- Vivado Introduction

VHDL Dataflow

- VHDL Concurrent Statement

```
targetSignal <= expression;
```

- Expression can include the boolean operators
  - `AND, NAND, OR, NOR, XOR, NXOR, NOT`
  - `NAND` and `NOR` are only binary operators, `NOT` is only unary operator
  - The operators are defined amongs others for bit and bit_vectors of the same size
  - There is also a conditional signal assignment

```
targetSignal <= expression1 when condition1 else
                expression2 when condition2 else
                -- ...
                defaultExpression;
```

To support arithmetic operations on std_logic and bit, two additional standard VHDL packages exist: `IEEE.numeric_std` and `IEEE.numeric_bit`

They define types `SIGNED` and `UNSIGNED` as array of `bit` and `std_logic`
Operators:

```
ABS, '-'              SIGNED → SIGNED

'+','-','*','/'       UNSIGNED,UNSIGNED → UNSIGNED  SIGNED,SIGNED → SIGNED
rem,mod               UNSIGNED,NATURAL → UNSIGNED   NATURAL,UNSIGNED → UNSIGNED
                                  INTEGER,SIGNED → SIGNED       SIGNED,INTEGER → SIGNED

'<','>','<=','>='     UNSIGNED,UNSIGNED → BOOLEAN   SIGNED,SIGNED → BOOLEAN
'=','/='              UNSIGNED,NATURAL → BOOLEAN    NATURAL,UNSIGNED → BOOLEAN
                                  INTEGER,SIGNED → BOOLEAN      SIGNED,INTEGER → BOOLEAN

not                                UNSIGNED → UNSIGNED           SIGNED → SIGNED
and,nand,or,nor,xor,xnor      UNSIGNED,UNSIGNED → UNSIGNED  SIGNED,SIGNED → SIGNED

sll,srl,rol,ror       UNSIGNED,INTEGER → UNSIGNED   SIGNED,INTEGER → SIGNED
```

Functions:

shift_left,shift_right rotate_left,rotate_right	SIGNED,NATURAL → SIGNED	UNSIGNED,NATURAL → SIGNED
resize	UNSIGNED,NATURAL → UNSIGNED	SIGNED,NATURAL → SIGNED
to_integer	UNSIGNED → NATURAL	SIGNED → INTEGER
to_unsigned	NATURAL,NATURAL → UNSIGNED	
to_signed	INTEGER,NATURAL → SIGNED	

Special Functions (bit or std_logic based only):

rising_edge,falling_edge	BIT → BOOLEAN	
std_match	STD_ULOGIC,STD_ULOGIC → BOOLEAN SIGNED,SIGNED → BOOLEAN STD_LOGIC_VECTOR,STD_LOGIC_VECTOR → BOOLEAN STD_ULOGIC_VECTOR,STD_ULOGIC_VECTOR → BOOLEAN	UNSIGNED,UNSIGNED → BOOLEAN
to_01	UNSIGNED,STD_LOGIC → UNSIGNED SIGNED,STD_LOGIC → SIGNED	

Package header, with declaration of signed and unsigned

```
package numeric_bit is
   type SIGNED   is array( natural range <> ) of bit;
   type UNSIGNED is array( natural range <> ) of bit;
   --implementations
   function "+"(a: SIGNED; b: UNSIGNED) return SIGNED;
end numeric_bit;
```

Make Package visible

```
library IEEE;
use IEEE.numeric_bit.all;
```

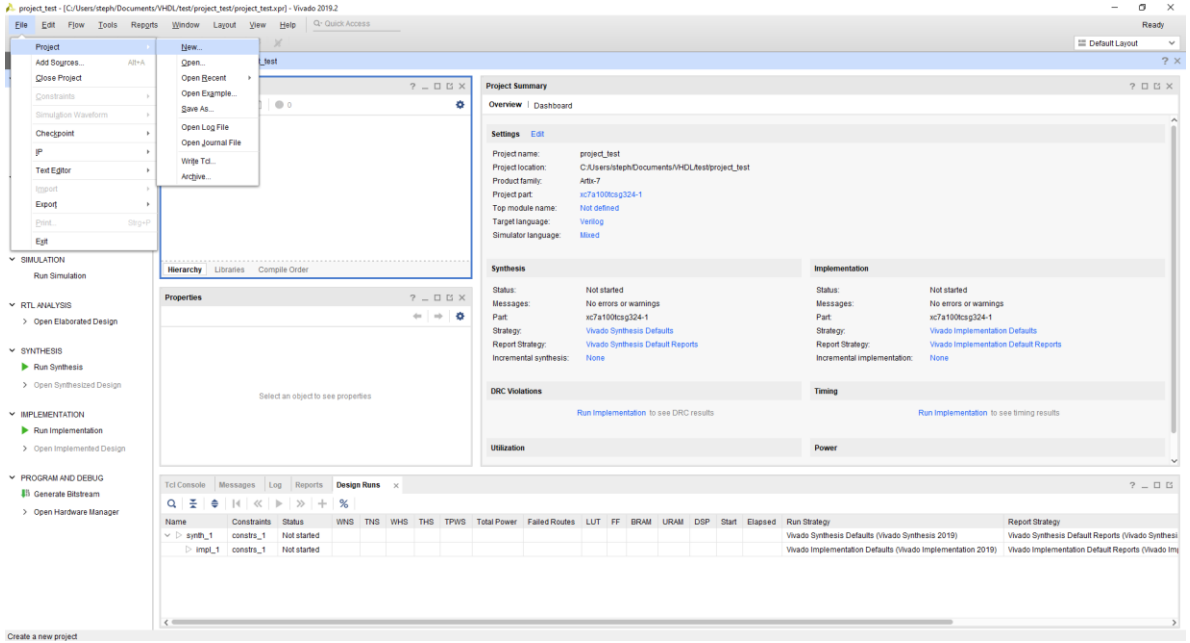
Use operators via casting

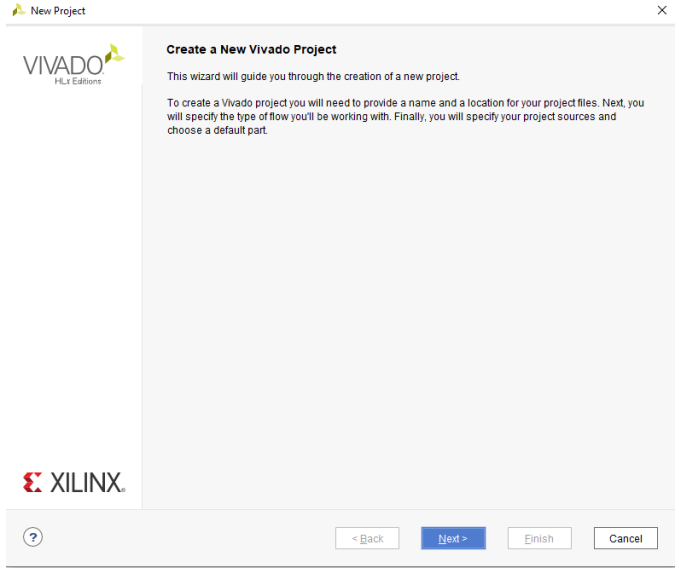
```
   variable UA4 : UNSIGNED(3 downto 0) := "0100"; -- decimal  4
   variable BV4 : bit_vector(3 downto 0) := "1000"; -- decimal  8
begin
   assert UA4 + UNSIGNED(BV4) = UNSIGNED'("1100")
```

What we will cover

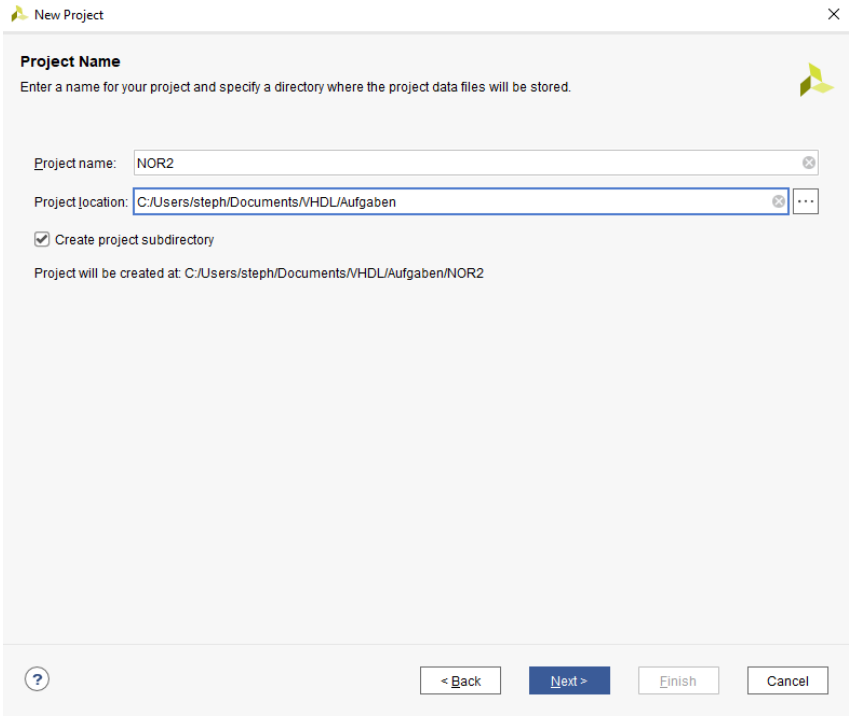
- Lab Description
- VHDL Primer
- **Vivado Introduction**

Projekt->neu


project_test - [C:/Users/steph/Documents/VHDL/test/project_test/project_test.xpr] - Vivado 2019.2
File Edit Flow Tools Reports Window Layout View Help
Project
Add Sources... Alt+A
Close Project
Constraints
Simulation Waveform
Checkpoint
IP
Text Editor
Import
Export
Print... Ctrl+P
Exit
New...
Open...
Open Recent
Open Example...
Save As...
Open Log File
Open Journal File
Write Tcl...
Archive...
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
Hierarchy Libraries Compile Order
Properties
Select an object to see properties
Project Summary
Overview | Dashboard
Settings Edit
Project name: project_test
Project location: C:/Users/steph/Documents/VHDL/test/project_test
Product family: Artix-7
Project part: xc7a100tcsg324-1
Top module name: Not defined
Target language: Verilog
Simulator language: Mixed
Synthesis
Status: Not started
Messages: No errors or warnings
Part: xc7a100tcsg324-1
Strategy: Vivado Synthesis Defaults
Report Strategy: Vivado Synthesis Default Reports
Incremental synthesis: None
Implementation
Status: Not started
Messages: No errors or warnings
Part: xc7a100tcsg324-1
Strategy: Vivado Implementation Defaults
Report Strategy: Vivado Implementation Default Reports
Incremental implementation: None
DRC Violations
Run Implementation to see DRC results
Timing
Run Implementation to see timing results
Utilization
Power
Tcl Console Messages Log Reports Design Runs
Name Constraints Status WNS TNS WHS THS TPWS Total Power Failed Routes LUT FF BRAM URAM DSP Start Elapsed Run Strategy Report Strategy
synth_1 constrs_1 Not started Vivado Synthesis Defaults (Vivado Synthesis 2019) Vivado Synthesis Default Reports (Vivado Synthesi
impl_1 constrs_1 Not started Vivado Implementation Defaults (Vivado Implementation 2019) Vivado Implementation Default Reports (Vivado Im
Create a new project

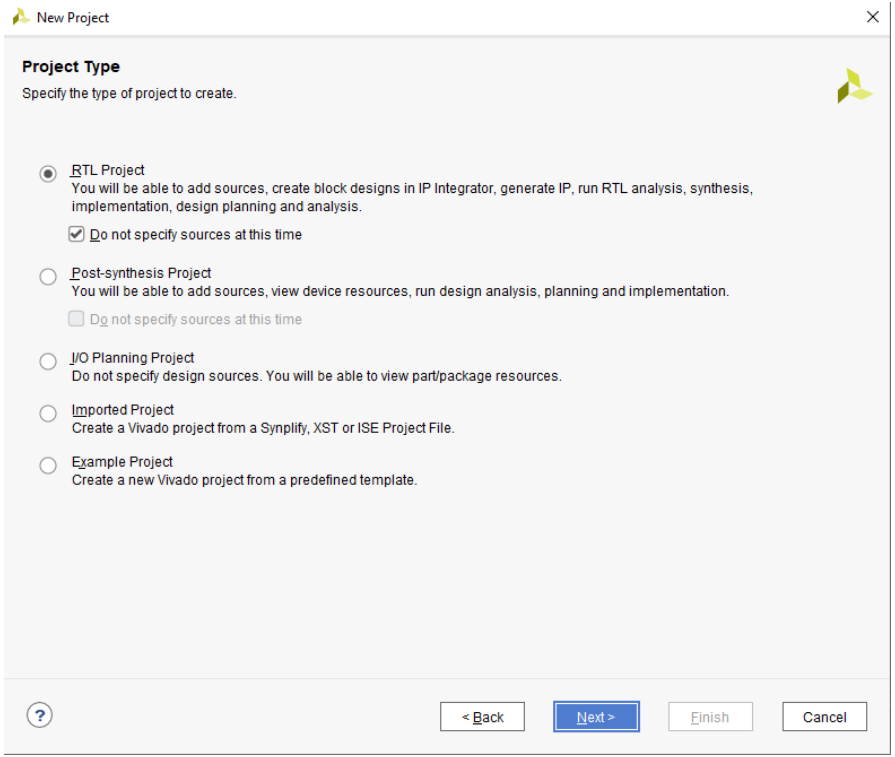

New Project
VIVADO HLx Editions
Create a New Vivado Project
This wizard will guide you through the creation of a new project.
To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.
XILINX
< Back
Next >
Finish
Cancel

Specify Project Name and Direction

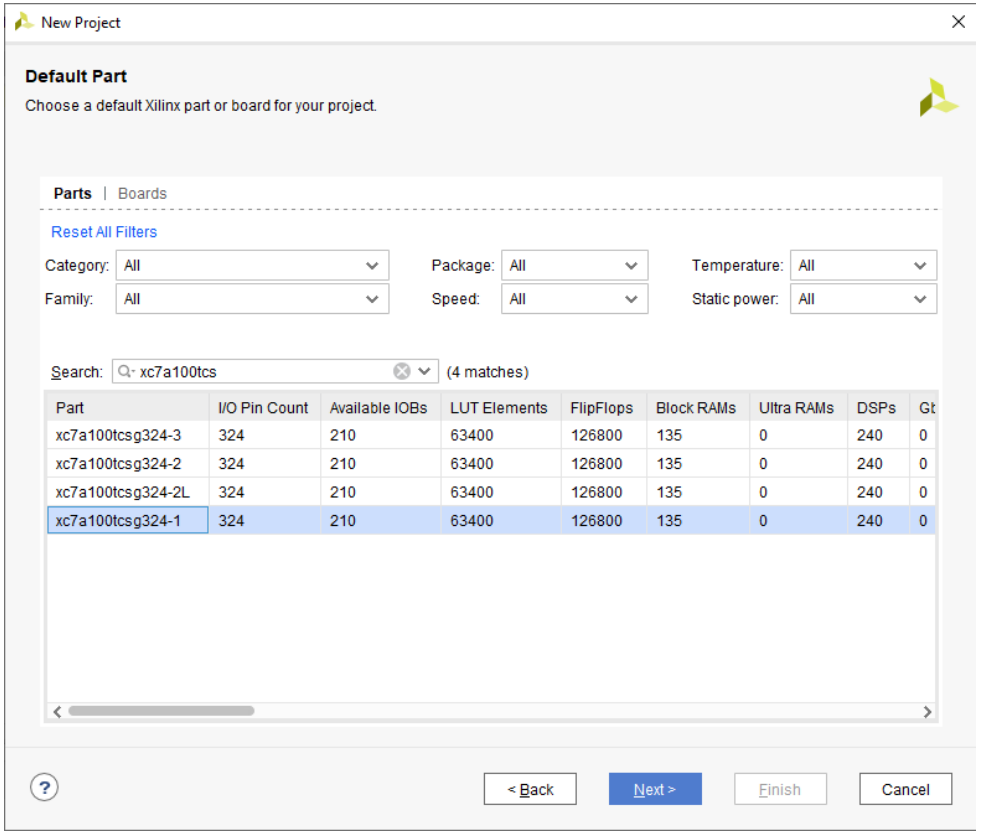


Tick RTL-project

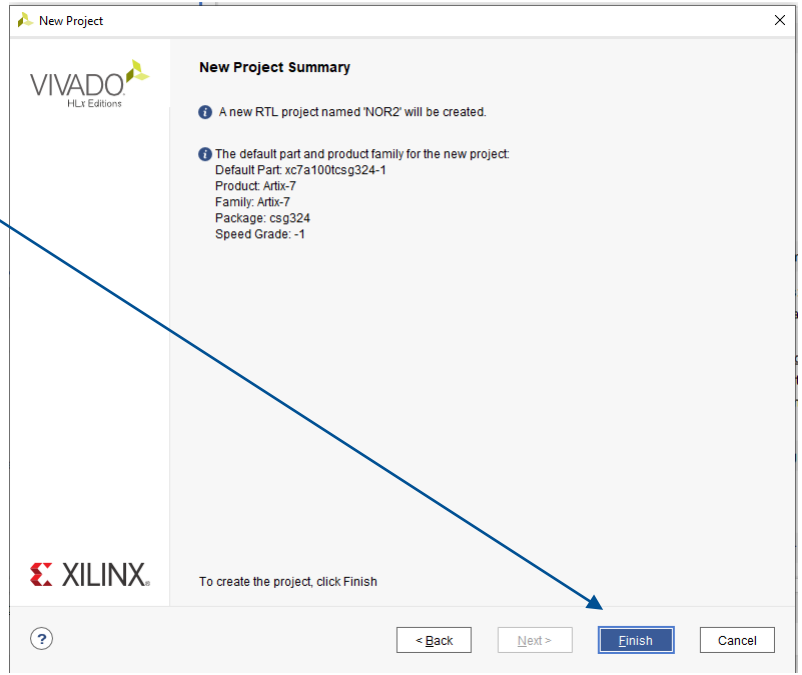
Tick „do not specify sources“ (important)



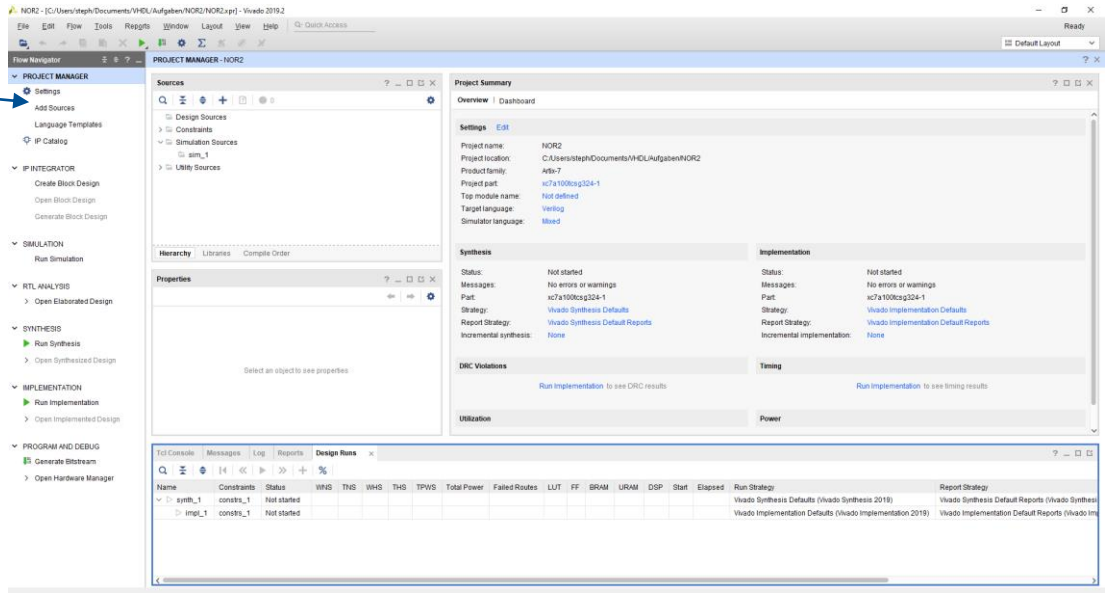
Choose xc7a100tcsg324-1
Isn‘t important which one
because we are not working
with an external board

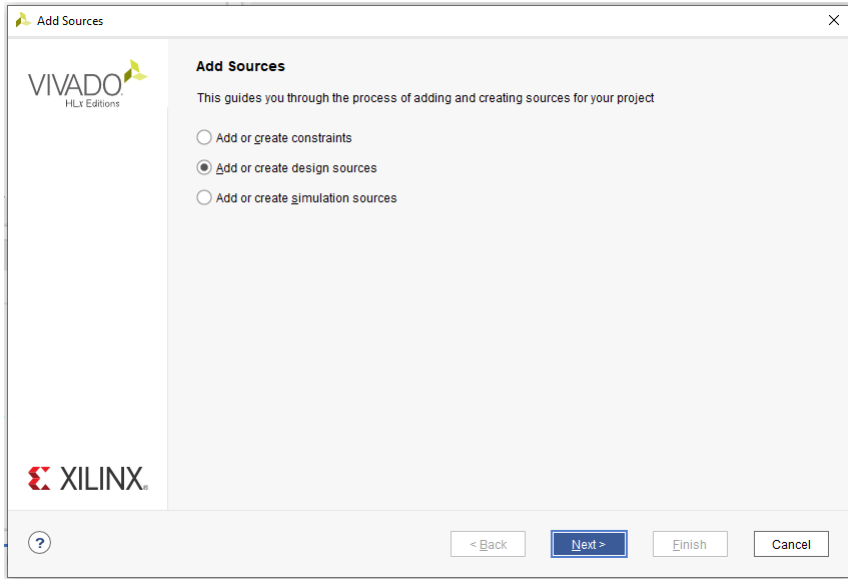


Finish creating a new project

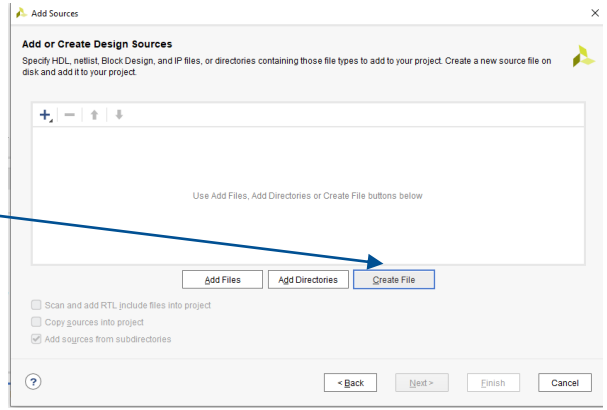


Add Sources

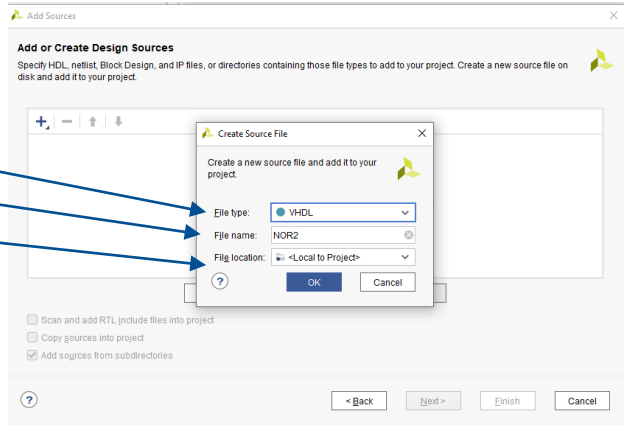

Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
PROJECT MANAGER - NOR2
Sources
Design Sources
Constraints
Simulation Sources
sim_1
Utility Sources
Hierarchy
Libraries
Compile Order
Properties
Select an object to see properties
Project Summary
Overview
Dashboard
Settings
Edit
Project name: NOR2
Project location: C:/Users/steph/Documents/VHDL/Aufgaben/NOR2
Product family: Artix-7
Project part: xc7a100tcsg324-1
Top module name: Not defined
Target language: Verilog
Simulator language: Mixed
Synthesis
Status: Not started
Messages: No errors or warnings
Part: xc7a100tcsg324-1
Strategy: Vivado Synthesis Defaults
Report Strategy: Vivado Synthesis Default Reports
Incremental synthesis: None
Implementation
Status: Not started
Messages: No errors or warnings
Part: xc7a100tcsg324-1
Strategy: Vivado Implementation Defaults
Report Strategy: Vivado Implementation Default Reports
Incremental implementation: None
DRC Violations
Run Implementation to see DRC results
Timing
Run Implementation to see timing results
Utilization
Power
Tcl Console
Messages
Log
Reports
Design Runs
Name
Constraints
Status
WNS
TNS
WHS
THS
TPWS
Total Power
Failed Routes
LUT
FF
BRAM
URAM
DSP
Start
Elapsed
Run Strategy
Report Strategy
synth_1
constrs_1
Not started
Vivado Synthesis Defaults (Vivado Synthesis 2019)
impl_1
constrs_1
Not started
Vivado Implementation Defaults (Vivado Implementation 2019)


Add Sources
VIVADO
HLx Editions
Add Sources
This guides you through the process of adding and creating sources for your project
Add or create constraints
Add or create design sources
Add or create simulation sources
XILINX
< Back
Next >
Finish
Cancel

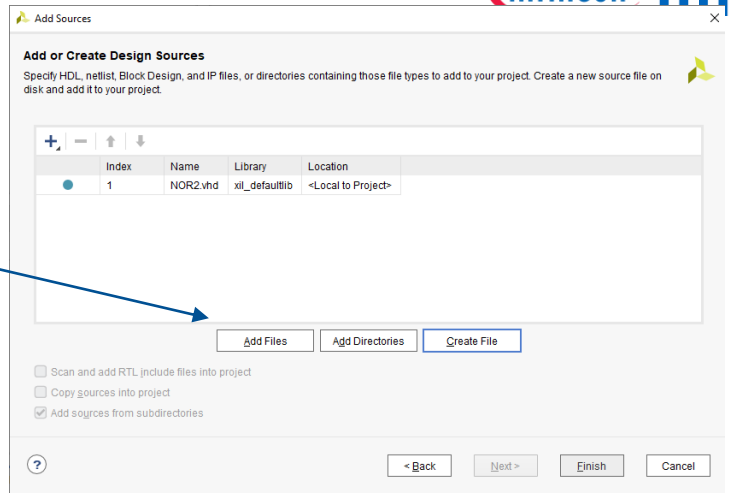
Create File



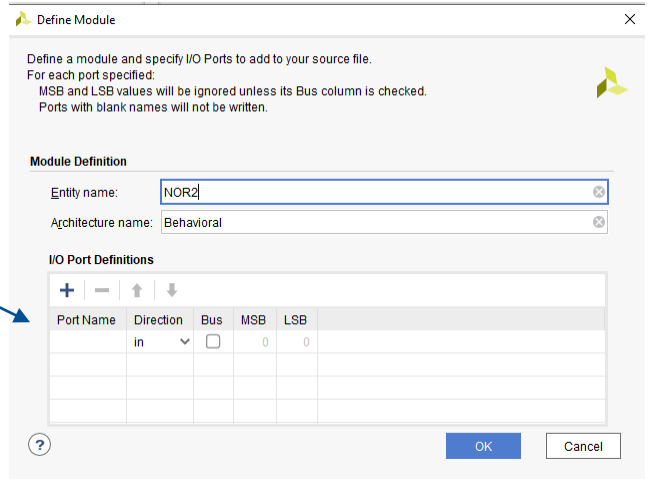
File type:VHDL
Filename: NOR2
Specify file location



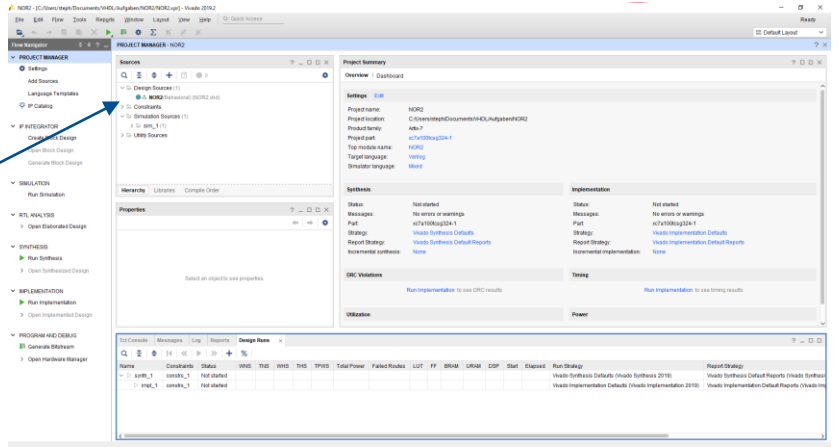
Add possible other files



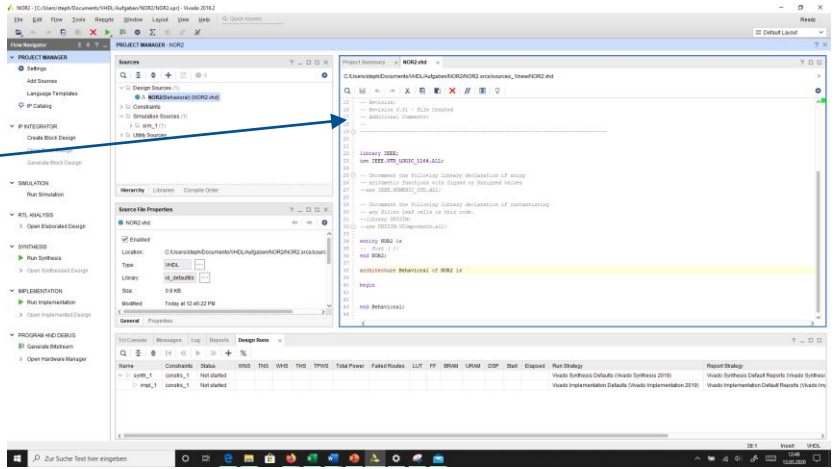
Define Modules
Possible in/outputs

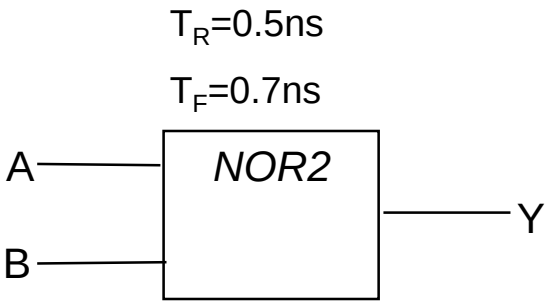


Created design sources
Open by clicking on the source



Can now be changed






```
entity NOR2 is
   generic( Tr: Time := 0.5 ns;
            Tf: Time := 0.7ns );
   port( A, B : in Bit;
         Y    : out Bit );
end NOR2;


architecture timed_dataflow
   of NOR2 is
begin
   Y <= '1' after Tr
           when (A OR B)= '0' else
        '0' after Tf;
end timed_dataflow;
```

Not that old VHDL style

```
entity NOR2TEST is
end NOR2TEST;
architecture TB of NOR2TEST is
signal A,B,Y : bit;
begin
UUT: entity WORK.NOR2
        generic map( 0.6 ns, 0.8 ns)
        port map( A,B,Y);
STIM: process
    begin -- enumertage pattern
        wait for 5 ns; A <= '1';
        wait for 2 ns; B <= '1';
        wait for 3 ns; A <= '0';
        wait;
    end process;
end TB;
```

```
STIM: process
    begin -- generate pattern
        for AP in bit'('0') to bit'('1') loop
            for BP in bit'('0') to bit'('1') loop
                A <= AP; B <= BP;
                wait for 10 ns;
            end loop;
        end loop;
        wait;
    end process;
```

type qualification

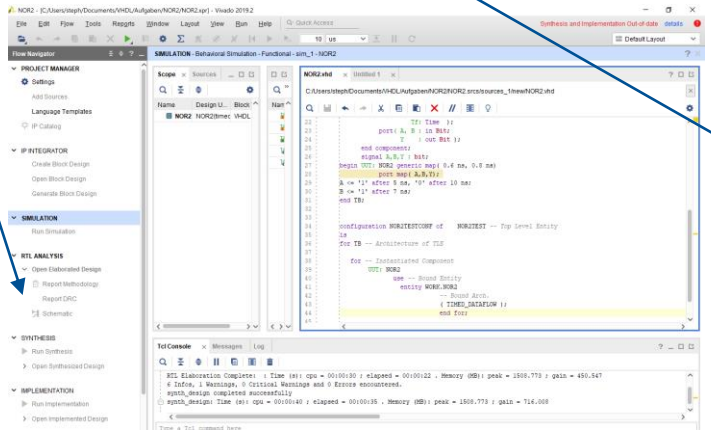
iterates through all combinat.

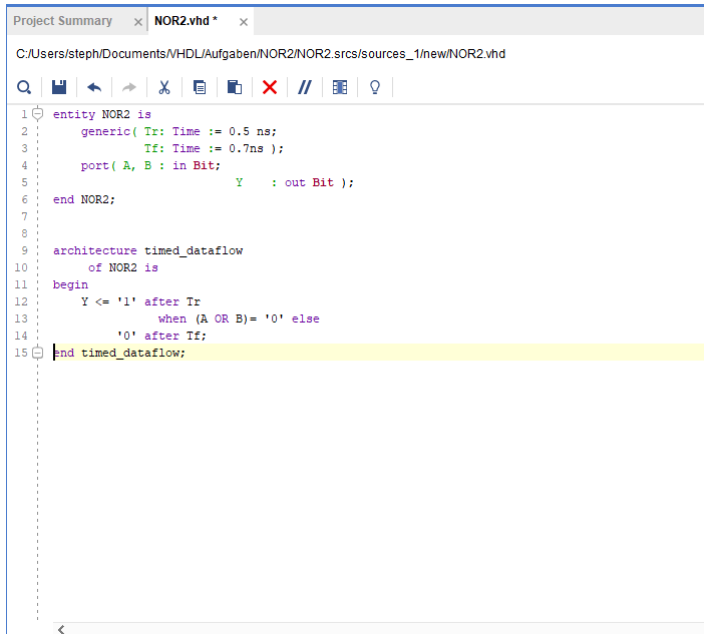
Very old VHDL style – not recommended

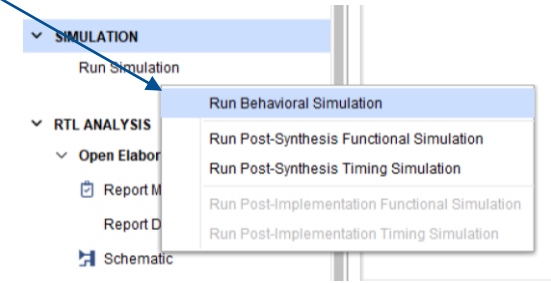
```
entity NOR2TEST is
end NOR2TEST;
architecture TB of NOR2TEST is
   component NOR2
      generic( Tr: Time;
               Tf: Time  );
      port( A, B : in Bit;
            Y    : out Bit );
   end component;
   signal A,B,Y : bit;
begin
UUT: NOR2 generic map( 0.6 ns, 0.8 ns)
          port map( A,B,Y);
A <= '1' after 5 ns, '0' after 10 ns;
B <= '1' after 7 ns;
end TB;
```

```
configuration NOR2TESTCONF of
   NOR2TEST -- Top Level Entity
is
for TB -- Architecture of TLE

   for -- Instantiated Component
      UUT: NOR2
         use -- Bound Entity
           entity WORK.NOR2
             -- Bound Arch.
             ( TIMED_DATAFLOW );
   end for;

end for;
end NOR2TESTCONF;
```

Run behavioral simulation







Check if the wave forms are feasible