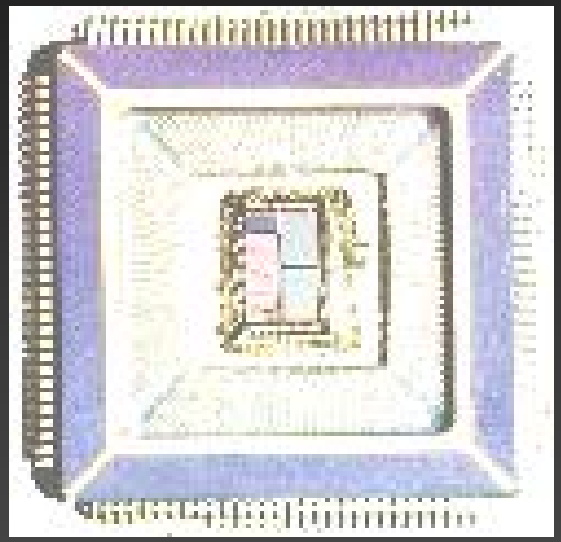


Hardware Modeling

Advanced Synthesis

Vienna University of Technology
Department of Computer Engineering
ECS Group

Contents

- Memory instances
- Common synthesis pitfalls
- Live Demo

Memory Instances

- ROM
  - Asynchronous
  - Synchronous
- RAM
  - Single Port (rw)
  - Dual Port (r/rw)
  - Triple Port (r/r/w)

Asynchronous ROM

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

entity async_rom is
  port
  (
    address : in std_logic_vector(7 downto 0);
    data    : out std_logic_vector(7 downto 0)
  );
end entity async_rom;
```

Asynchronous ROM

```
architecture beh of async_rom is
  subtype ROM_ENTRY_TYPE is std_logic_vector(7 downto 0);
  type ROM_TYPE is array (0 to (2 ** 8) – 1) of ROM_ENTRY_TYPE;
  constant rom : ROM_TYPE :=
  (
    0 => x"FF", 1 => x"AA", 30 => x"55", ...,
    others => x"00"
  );
begin
  process(address)
  begin
    data <= rom(to_integer(unsigned(address)));
  end process;
end architecture beh;
```

Synchronous ROM

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

entity sync_rom is
  port
  (
    clk     : in std_logic;
    address : in std_logic_vector(7 downto 0);
    data    : out std_logic_vector(7 downto 0)
  );
end entity sync_rom;
```

Synchronous ROM

```
architecture beh of sync_rom is
  subtype ROM_ENTRY_TYPE is std_logic_vector(7 downto 0);
  type ROM_TYPE is array (0 to (2 ** 8) – 1) of ROM_ENTRY_TYPE;
  constant rom : ROM_TYPE :=
  (
    0 => x”FF”, 1 => x”AA”, 30 => x”55”, ...,
    others => x”00”
  );
begin
  process(clk)
  begin
    if rising_edge(clk) then
      data <= rom(to_integer(unsigned(address)));
    end if;
  end process;
end architecture beh;
```

Synchronous Single Port RAM

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

entity sp_ram is
  generic
  (
    ADDR_WIDTH : integer range 1 to integer'high;
    DATA_WIDTH : integer range 1 to integer'high
  );
  port
  (
    clk      : in  std_logic;
    address  : in  std_logic_vector(ADDR_WIDTH - 1 downto 0);
    data_out : out std_logic_vector(DATA_WIDTH - 1 downto 0);
    wr       : in  std_logic;
    data_in  : in  std_logic_vector(DATA_WIDTH - 1 downto 0)
  );
end entity sp_ram;
```

Synchronous Single Port RAM

```
architecture beh of sp_ram is
  subtype RAM_ENTRY_TYPE is std_logic_vector(DATA_WIDTH - 1 downto 0);
  type RAM_TYPE is array (0 to (2 ** ADDR_WIDTH) – 1) of RAM_ENTRY_TYPE;
  signal ram : RAM_TYPE := (others => x”00”);
begin
  process(clk)
  begin
    if rising_edge(clk) then
      data_out <= ram(to_integer(unsigned(address)));
      if wr = ‘1’ then
        ram(to_integer(unsigned(address))) <= data_in;
      end if;
    end if;
  end process;
end architecture beh;
```

Synchronous Dual Port RAM

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

entity dp_ram is
  generic
  (
    ADDR_WIDTH : integer range 1 to integer'high;
    DATA_WIDTH : integer range 1 to integer'high
  );
  port
  (
    clk        : in  std_logic;
    address1   : in  std_logic_vector(ADDR_WIDTH - 1 downto 0);
    data_out1  : out std_logic_vector(DATA_WIDTH - 1 downto 0);
    wr1        : in  std_logic;
    data_in1   : in  std_logic_vector(DATA_WIDTH - 1 downto 0);
    address2   : in  std_logic_vector(ADDR_WIDTH - 1 downto 0);
    data_out2  : out std_logic_vector(DATA_WIDTH - 1 downto 0)
);
end entity dp_ram;
```

Synchronous Dual Port RAM

```
architecture beh of dp_ram is
  subtype RAM_ENTRY_TYPE is std_logic_vector(DATA_WIDTH - 1 downto 0);
  type RAM_TYPE is array (0 to (2 ** ADDR_WIDTH) – 1) of RAM_ENTRY_TYPE;
  signal ram : RAM_TYPE := (others => x”00”);
begin
  process(clk)
  begin
    if rising_edge(clk) then
      data_out1 <= ram(to_integer(unsigned(address1)));
      data_out2 <= ram(to_integer(unsigned(address2)));
      if wr1 = ‘1’ then
        ram(to_integer(unsigned(address1))) <= data_in1;
      end if;
    end if;
  end process;
end architecture beh;
```

Synchronous Triple Port RAM

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

entity tp_ram is
  generic
  (
    ADDR_WIDTH : integer range 1 to integer'high;
    DATA_WIDTH : integer range 1 to integer'high
  );
  port
  (
    clk                              : in  std_logic;
    address1, address2, address3     :
      in  std_logic_vector(ADDR_WIDTH - 1 downto 0);
    data_in1                         :
      in std_logic_vector(DATA_WIDTH - 1 downto 0);
    wr1                              : in  std_logic;
    data_out2, data_out3             :
      out std_logic_vector(DATA_WIDTH - 1 downto 0)
);
end entity tp_ram;
```

Synchronous Triple Port RAM

```
architecture beh of tp_ram is
  subtype RAM_ENTRY_TYPE is std_logic_vector(DATA_WIDTH - 1 downto 0);
  type RAM_TYPE is array (0 to (2 ** ADDR_WIDTH) – 1) of RAM_ENTRY_TYPE;
  signal ram : RAM_TYPE := (others => x”00”);
begin
  process(clk)
  begin
    if rising_edge(clk) then
      data_out2 <= ram(to_integer(unsigned(address2)));
      data_out3 <= ram(to_integer(unsigned(address3)));
      if wr1 = ‘1’ then
        ram(to_integer(unsigned(address1))) <= data_in1;
      end if;
    end if;
  end process;
end architecture beh;
```

Contents

- Memory instances
- Common synthesis pitfalls
- Live Demo

Common Synthesis Pitfalls

- Complexity
- Latches

Complexity

- and
- add
- mul
- div
- mod

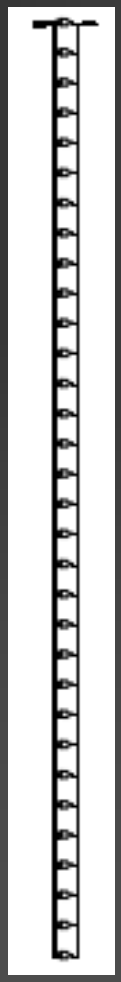
Complexity

```
entity operation is
  port
  (
    a, b: in  std_logic_vector(31 downto 0);
    o   : out std_logic_vector(31 downto 0)
  );
end entity operation;
```

```
architecture beh of operation is
begin
  o <= a OPERATION b;
end architecture beh;
```

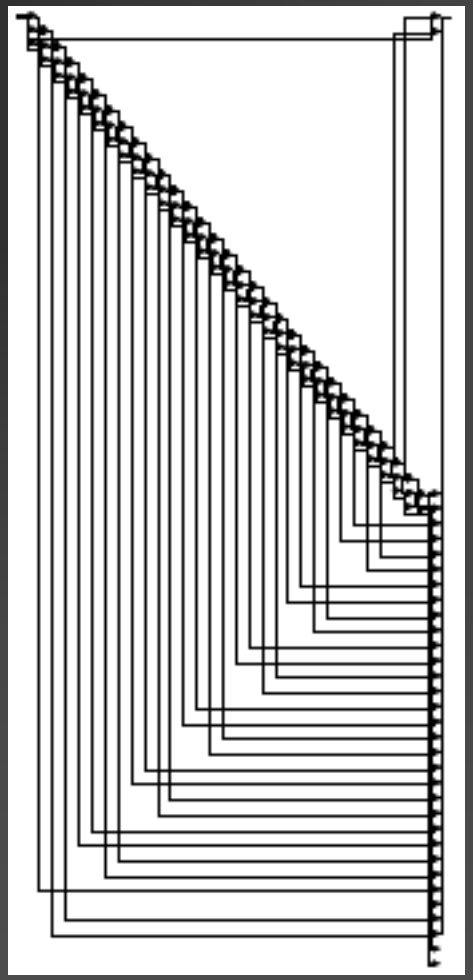
Complexity - AND

- o <= a and b;



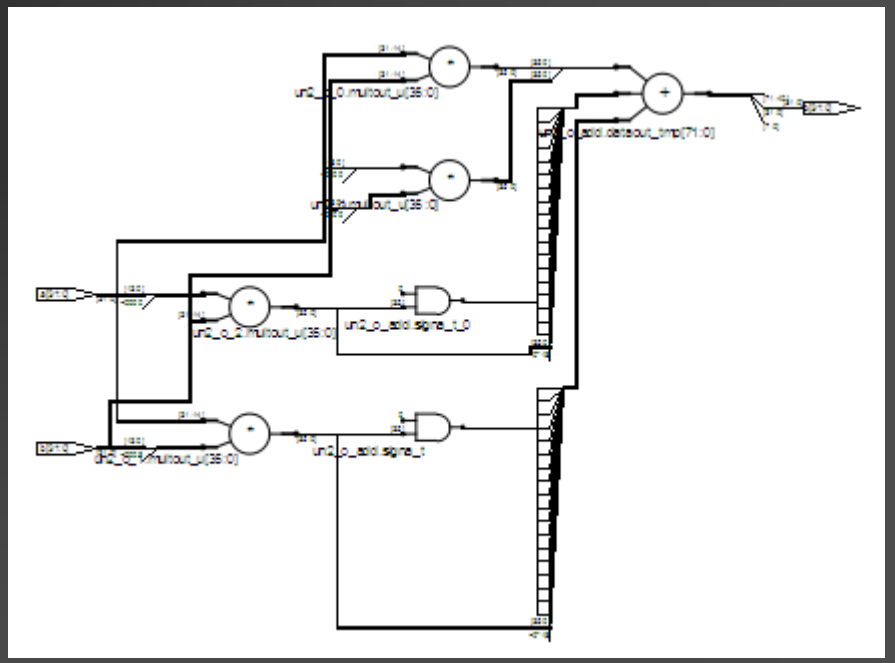
Complexity - ADD

- o <= a + b;



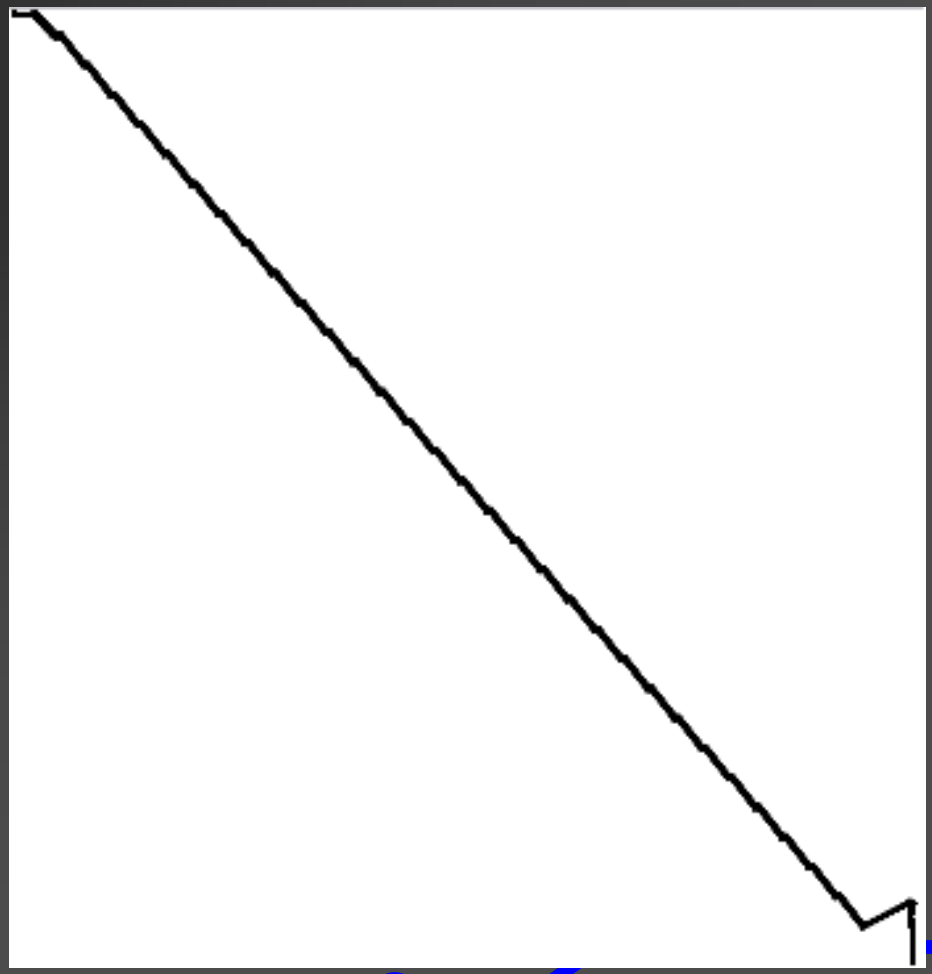
Complexity - MUL

- o <= a * b;



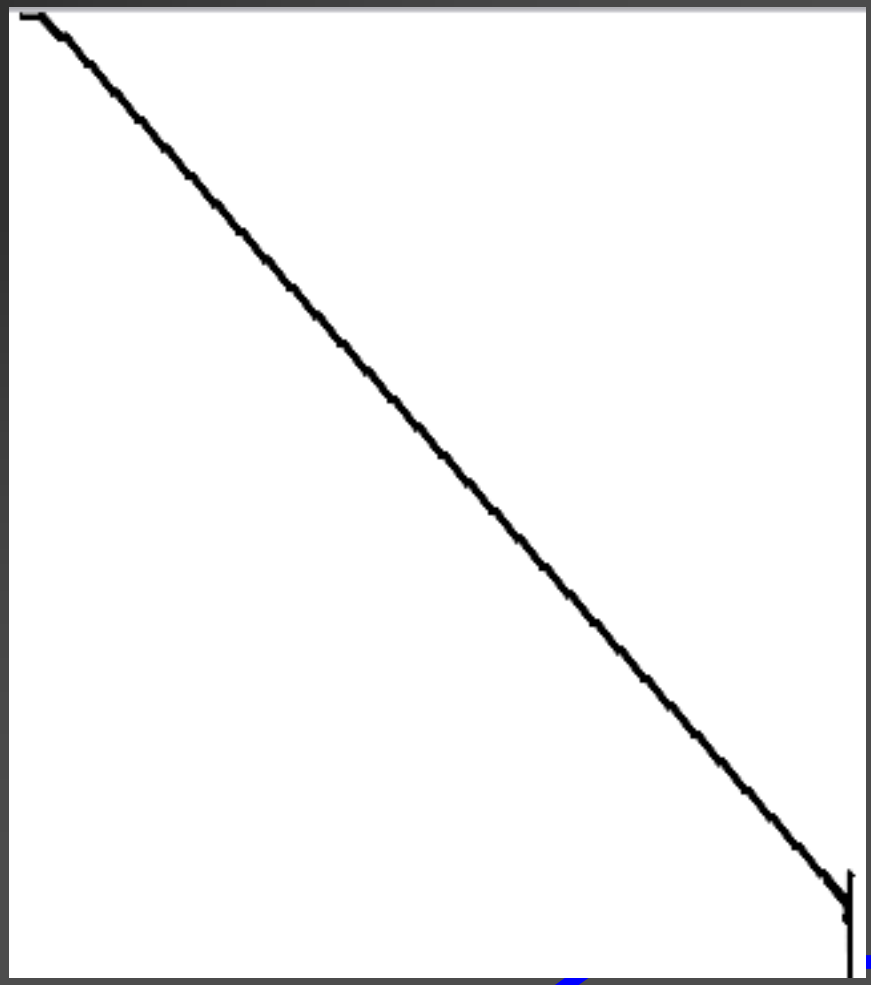
Complexity - DIV

- o <= a / b;



Complexity - MOD

- o <= a mod b;



Complexity - Results

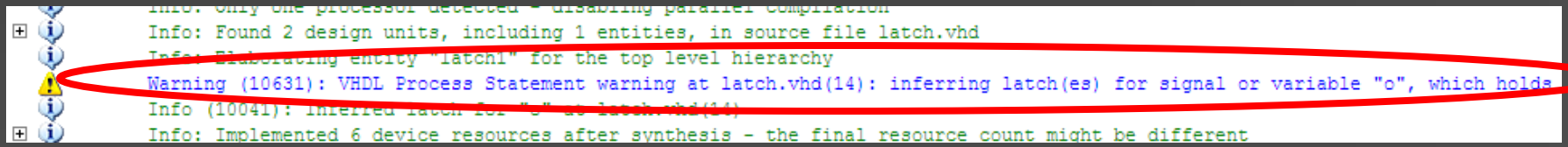
Operation	Estim. Frequency	# LUTs	#DSP Blocks à 8 9-bit multiplier
and	845 MHz	32	0
add	320 MHz	32	0
mul	250 MHz	0	1
div	14 MHz	2152	0
mod	7 MHz	2220	0

Latches

```
process(i1, i2, i3)
begin

  if i1 = '1' and i2 = '1' then
    o <= i3;
  end if;
end process;
```

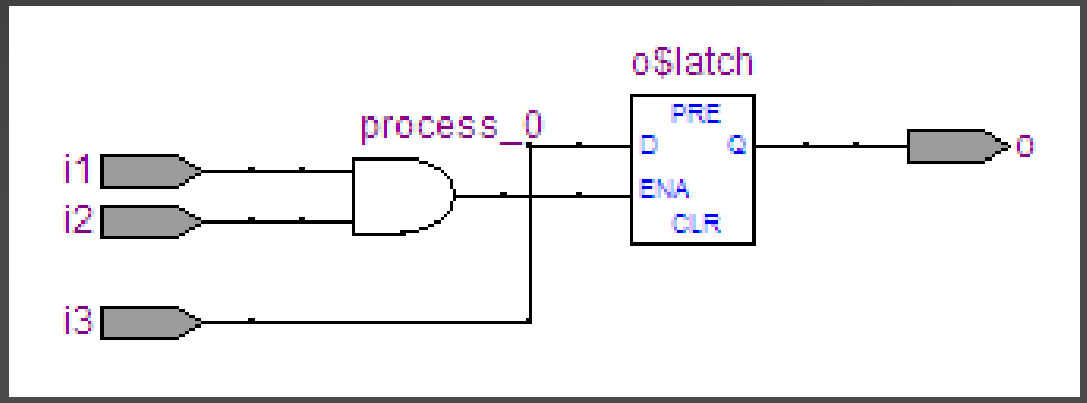
Latches

```
process(i1, i2, i3)
begin

  if i1 = '1' and i2 = '1' then
    o <= i3;
  end if;
end process;
```



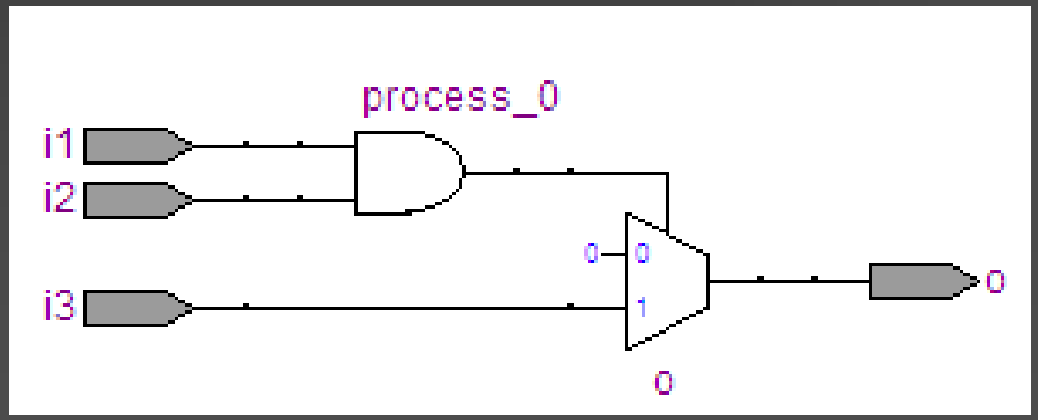
Latches

```
process(i1, i2, i3)
begin

  if i1 = '1' and i2 = '1' then
    o <= i3;
  end if;
end process;
```



Latches

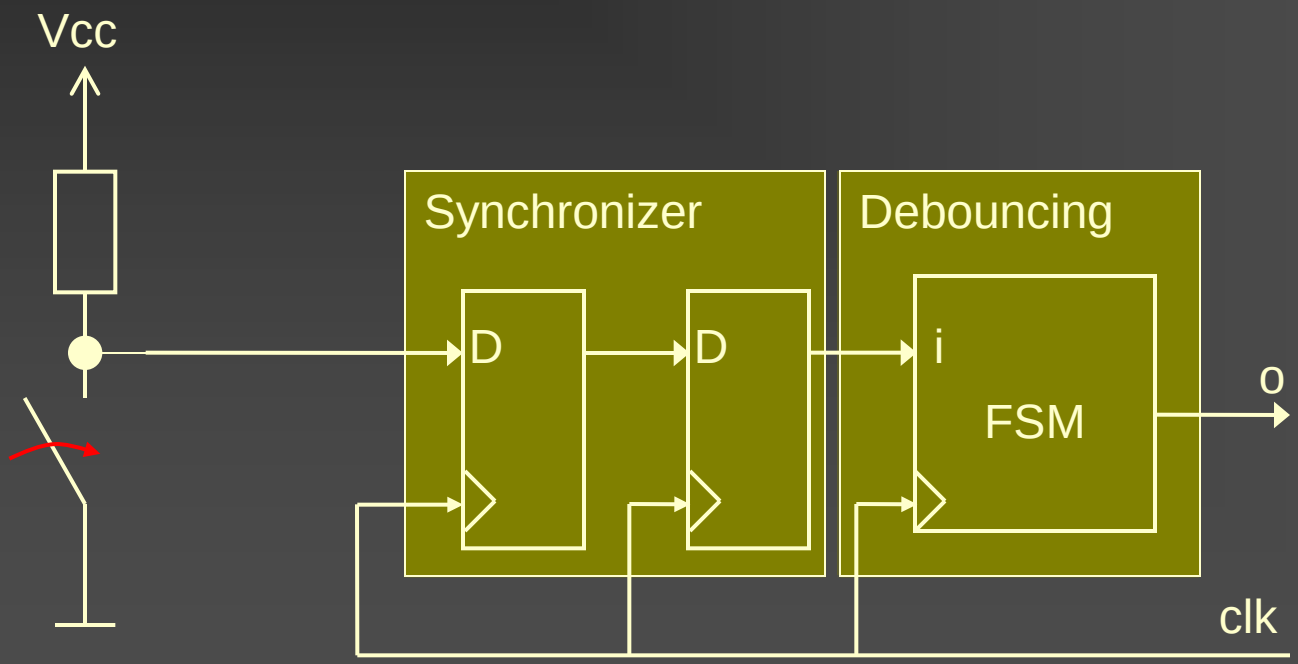
```
process(i1, i2, i3)
begin
  o <= '0';
  if i1 = '1' and i2 = '1' then
    o <= i3;
  end if;
end process;
```



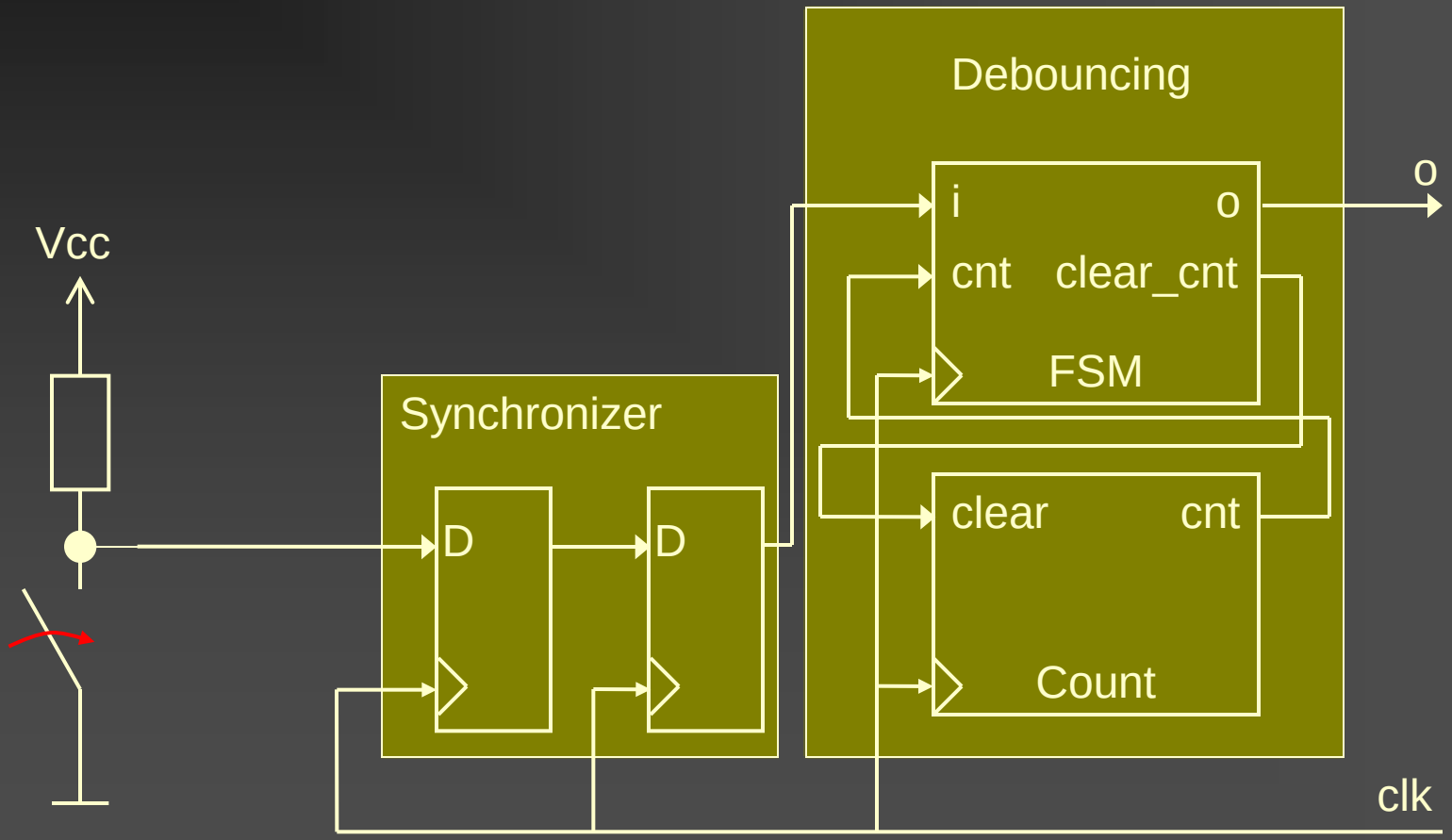
Contents

- Memory instances
- Common synthesis pitfalls
- Live Demo

Live Demo - Debouncing



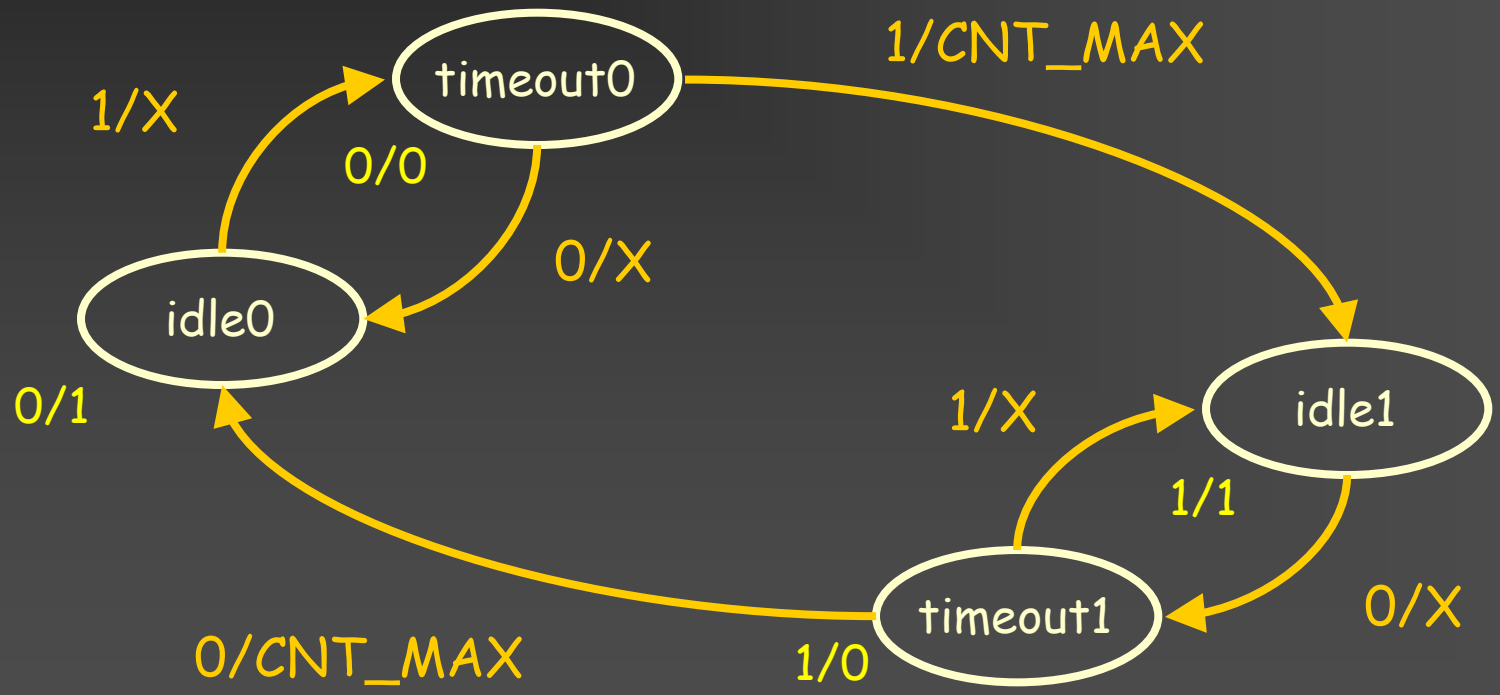
Live Demo - Debouncing



Live Demo - Debouncing

Inputs: i/cnt

Outputs: o/clear_cnt



Live Demo - Debouncing

Inputs: i/cnt

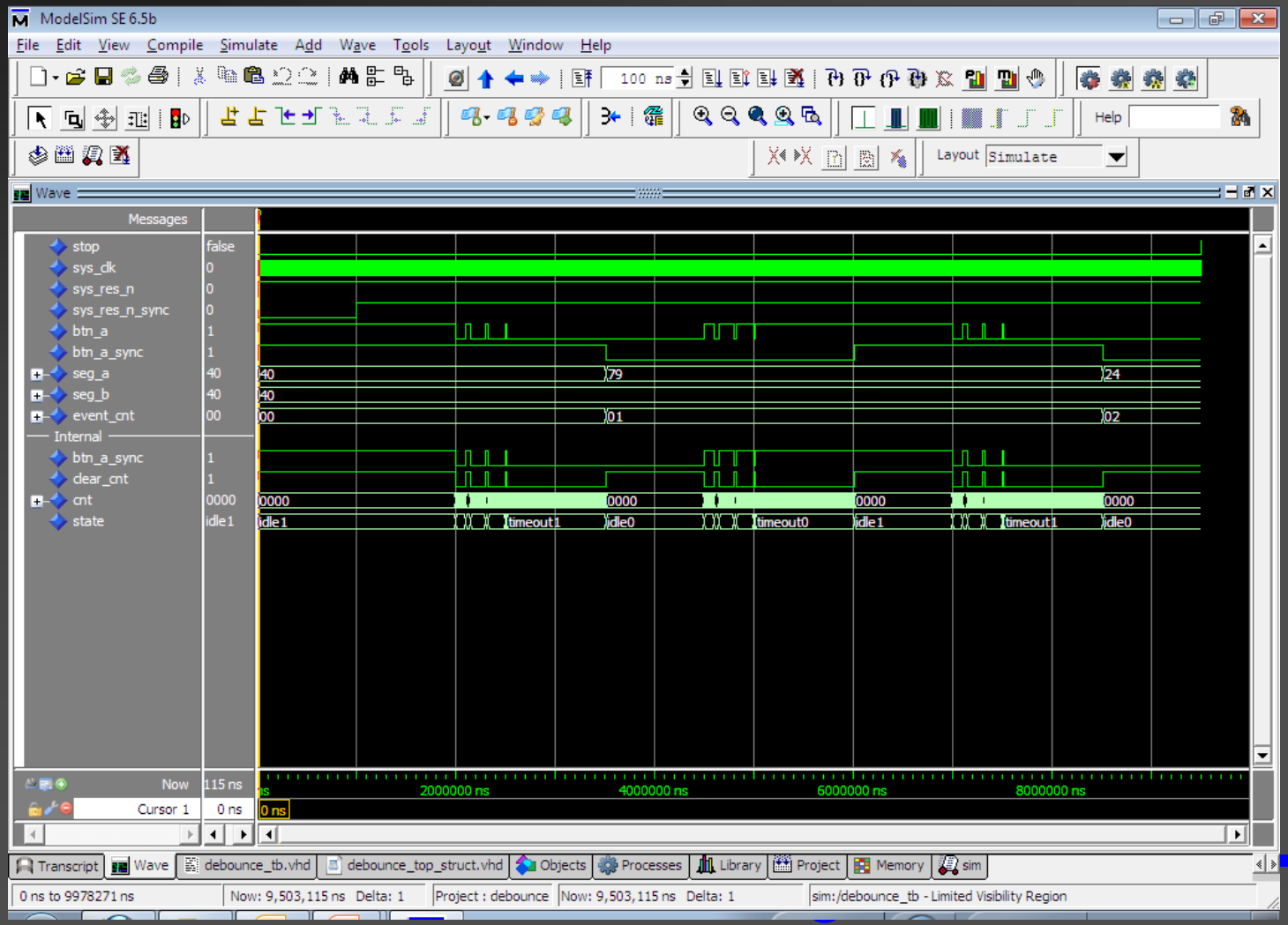
Outputs: o/clear_cnt

State	Output	Condition	Next state
idle0	0/1	1/X	timeout0
timeout0	0/0	0/X	idle0
		1/CNT_MAX	idle1
idle1	1/1	0/X	timeout1
timeout1	1/0	1/X	idle1
		0/CNT_MAX	idle0

Live Demo - Debouncing

- Design Entry

Live Demo - Debouncing



Live Demo - Debouncing

- Synthesis: Quartus

Additional Literature

- Device specific guidelines
  - Device datasheets
  - Manufacturer user guides and recommendations
- Platform specific guidelines
  - Board datasheet
- VHDL language guide
  - Peter Ashenden – The Designer's Guide to VHDL, 3rd edition

Summary

- Memory instances
- Common synthesis pitfalls
  - Complexity
  - Latches
- Live Demo