INTRODUCTION TO FPGA PROGRAMMING

Lesson 04: Sequential Logic and Flip-Flops

Dr. Davide Cieri[1]

[1]Max-Planck-Institut für Physik, Munich

August 2024

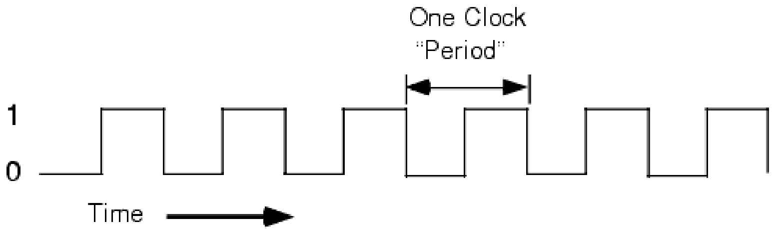
MAX-PLANCK-INSTITUT FÜR PHYSIK

REMINDER: COMBINATIONAL AND SEQUENTIAL

- A combinational circuit is one whose outputs depend only on the current inputs
- A sequential circuit is one whose outputs depend on the current in inputs and on previous outputs
- On FPGAs sequential logic is achieved using flip-flops or registers

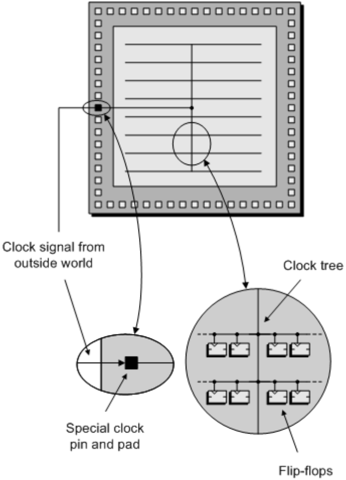
THE CLOCK SIGNAL

- The *Clock signal* or just *clock*, is a digital signal that alternates between 0 and 1, with a fixed frequency
- Changes in clock signals are called *edges*
  - Rising edge: 0 to 1
  - Falling edge: 1 to 0
- Duty Cycle: fraction of time a clock signal is high



CLOCKS IN AN FPGA

- Clocks inside an FPGA drive all sequential logics (Flip-Flops, RAMs, FIFOs, etc..)
- FPGAs typically support multiple clock domains
  - E.g. different interfaces might require different frequencies, more in another lesson
- Clocks are generated externally to the FPGA and input into special pin
  - Our Basys3 board has an oscillator chip generating a 100 MHz clock, input to pin W5
- A dedicated routing logic is available to minimise skew



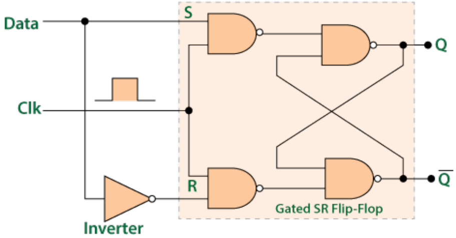
BASIC TIMING CONSTRAINT

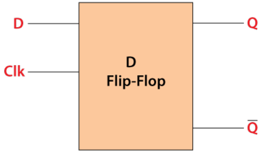
- Your design should know the frequency of the clock you are running
- As all other ports, clocks must be mapped to a physical pin and have a specific IO standard
- All this must be defined in the constraint XDC file

```
set_property PACKAGE_PIN W5 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports
    CLK]
# create_clock syntax explanation
create_clock -period <clock period in ns> \
             -name <a name for your clock> \
             -waveform <optional, otherwise set a half-duty clock by default> \
             -add <map the clock to a specific port>
```

FLIP-FLOPS

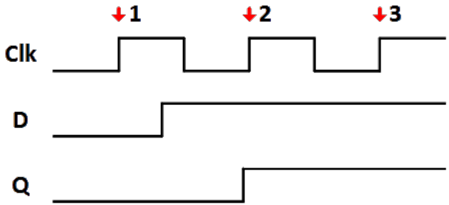
- Flip flops are special circuits used to store state information
- Several types of flip flops can be constructed
  - A good list is available here.
- Most of FPGA vendors employs a D-type Flip-Flop (or *data* flip-flop)

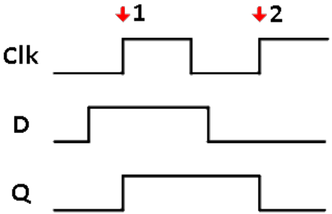




FLIP-FLOP BEHAVIOUR

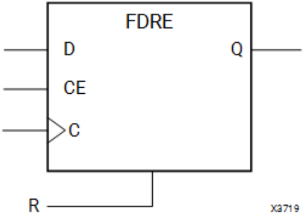
- The flip-flop checks the state of the input signal at each rising edge of the clock and set the output (*registering*)

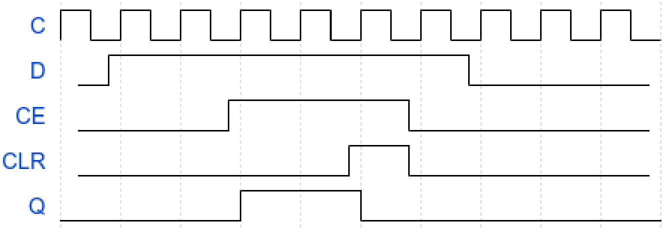




THE ACTUAL REALITY

- On the FPGA, you can instantiate Flip-Flops with a clock-enable (CE) and reset signal (synchronous or asynchronous)
- Example, with asynchronous reset





CONCURRENT STATEMENTS IN VHDL

- The `process` statement is used in VHDL to define blocks to be evaluated sequentially
- Statements inside a process are evaluated sequentially (like most programming languages)
- Multiple process blocks are evaluated concurrently

```
<process_name> : process
  begin
    statement 1;
    statement 2;
    ...
    statement N;
  end process <process_name>;
```

PROCESS SENSITIVITY LIST

- A process can have a sensitivity list
- List of signal to which te process is sensitive (for example a clock)
  - The process is executed only when there is a change to a signal in the sensitivity list

```
<process_name> : process(signalA , signalB)
begin
  statement 1;
  statement 2;
  ...
  statement N;
end process <process_name>;
```

CONDITIONAL STATEMENTS - IF

- `if` statement allows conditional execution inside a `process`
  - Condition should return a boolean
  - Allowed relational operators: Equal(`=`), Not Equal (`/=`), Less Than (`<`, Less Than or Equal to (`<=`), Greater Than (`>`), Greater Than or Equal to (`>=`)

```
if <condition1> then
  <vhdl statement>;
end if;
```

```
if <condition1> then
  <vhdl statement 1>;
else
  <vhdl statement 2>;
end if;
```

```
if <condition1> then
  <vhdl statement 1>;
elsif <condition2> then
  <vhdl statement 2>;
else
  <vhdl statement 3>;
end if;
```

CONDITIONAL STATEMENTS - CASE

```
case <signal> is
  when <condition A> => <statement A>;
  when <condition B> => <statement B>;
  when others => <statement C>;
end case;
```

- Similar to switch in C
- Default option `others` is available
- `null` is a valid VHDL statement that can be used if no assignment is wanted

CONDITIONAL STATEMENTS - EXAMPLES

```
process(a,b,sel)
begin
  if sel = '0' then
    y <= a;
  elsif sel = '1' then
    y <= b;
  else
    null;
  end if;
end process;
```

```
process(a,b,sel)
begin
  case sel is
    when '0' => y <= a;
    when '1' => y <= b;
    when others => null;
  end case;
end process;
```

EDGE DETECTION IN VHDL

- The `ieee.std_logic_1164.all` package contains useful functions to detect signal (and clock) edges
  - `rising_edge(s)` returns `true`, if there is a rising edge on the signal `s`
  - `falling_edge(s)` returns `true`, if there is a falling edge on the signal `s`

```
-- A D-Flip-Flop implementation
process(clk)
  begin
    if rising_edge(clk) then
      Q <= D;
    end if;
end process;
```

UNDERSTANDING SIGNAL ASSIGNMENTS IN VHDL PROCESSES

- Statements within a VHDL process are executed sequentially.
- However, signal assignments take effect only at the end of the process.
- Risk: Overwriting earlier assignments within the same process.
  - Example: Flip-Flop driven by multiple inputs within a single clock cycle.

```vhdl
process(clk) is
begin
    if rising_edge(clk) then
      a <= b; -- Overwritten by later assignments
      b <= c;
      c <= a;
      a <= c; -- This is the final assignment to 'a'
    end if;
end process;
```

RESETS

- Two types of resets can be defined.
  - Asynchronous Reset. Reset signal is not synchronous to the process clock.
  - Synchronous Reset. Reset signal is synchronous to the process clock.

```
-- D-Flip-Flop with Asynchronous Reset
process(clk, rst)
  begin
    if rst = '1' then
      Q <= '0';
    elsif rising_edge(clk) then
      Q <= D;
    end if;
end process;
```

```
-- D-Flip-Flop with Synchronous Reset
process(clk)
  begin
    if rising_edge(clk) then
      if rst = '1' then
        Q <= '0';
      else
        Q <= D;
      end if;
    end if;
end process;
```

CONSTANTS

- As in common programmable languages, it is possible to define constants in VHDL
  - Improve code readability
  - If used in multiple places, only one change needed. Improved code maintenance
  - Can be defined in `process`, `architecture` or `package` blocks

```
architecture RTL of MyModule is
  constant SIGNAL_WIDTH : integer := 16;
  signal my_signal : std_logic_vector(
      SIGNAL_WIDTH-1 downto 0);
begin
  ...
end architecture RTL;
```

```
process(clk)
  constant reset_value :
      std_logic_vector := "1010";
  begin
    if rising_edge(clk) then
      if mysignal = reset_value then
        mysignal <= (others => '0');
      end if;
    end if;
end process;
```

VARIABLES

- Variables are VHDL objects local to a VHDL process
  - They cannot be used outside the process, where they are defined
- They can be of any type
- Value assignment using the := symbol
- Variables take immediately the value of the assignment

```
EXAMPLE_VAR : process(clk)
  variable v_Count : integer := 0;
begin
  if rising_edge(clk) then
    v_Count := v_Count + 1;
    r_Var_Copy1 <= v_Count; -- r_Var_Copy2 updates to 5 the next clock cycle
    if v_Count = 5 then
      v_Count := 0;
    end if;
    r_Var_Copy2 <= v_Count; -- r_Var_Copy2 never gets to 5, since v_Count is immediately reset
  end if;
end process EX_VAR;
```

WHILE STATEMENT

- **Syntax:**

```
while <condition> loop
    <sequential statements>
end loop;
```

- **Description:**
  - Repeats a block of code as long as the condition is true.
  - The condition is evaluated before each iteration.
  - Used for loops where the number of iterations is not known beforehand.
- **Example:**

```
process
    variable i : integer := 0;
begin
    while i < 10 loop
        -- Perform some operation
        i := i + 1;
    end loop;
end process;
```

FOR STATEMENT IN VHDL

- **Syntax:**

```
for <loop_variable> in <range> loop
    <sequential statements>
end loop;
```

- **Description:**
  - Repeats a block of code a fixed number of times.
  - The loop variable automatically iterates over the specified range.
  - Used when the number of iterations is known beforehand.
- **Example:**

```
process
begin
    for i in 0 to 9 loop
        -- Perform some operation
    end loop;
end process;
```

LAB 07: COUNTERS AND DEBOUNCING

LAB 08: AN LED BLINKER

The figures in these slides are taken from:

- Digital Design: Principles and Practices, Fourth Edition, John F. Wakerly, ISBN 0-13- 186389-4.

- nandland.com
- docs.amd.com
- https://www.symmetryelectronics.com/
- https://www.edn.com/