Sequential code

Xiaojun Zhai
xzhai@essex.ac.uk

School of Computer Science and Electronic Engineering
University of Essex (UK)

CE339/CE869 - Lecture 3
Jan 2020

Outline

Concurrent code

- VHDL code is inherently concurrent (parallel)
- VHDL is in "concurrent mode" by default (unless explicitly indicated by the use of `PROCESS`, `FUNCTION`, `PROCEDURE`)
- The order of the instructions is irrelevant

```
statement1;        statement2;        statement3;        ...
statement2;   ≡    statement1;   ≡    statement1;   ≡    ...
statement3;        statement3;        statement2;        ...
```

Concurrent code

- VHDL code is inherently concurrent (parallel)
- VHDL is in "concurrent mode" by default (unless explicitly indicated by the use of `PROCESS`, `FUNCTION`, `PROCEDURE`)
- The order of the instructions is irrelevant

```
statement1;        statement2;        statement3;        ...
statement2;   ≡    statement1;   ≡    statement1;   ≡    ...
statement3;        statement3;        statement2;        ...
```

- Purely concurrent code cannot be used to implement synchronous circuits (with a small exception)

In order to build sequential logic we need sequential code

Sequential code

- Sequential code is the code enclosed in `PROCESS`, `FUNCTION`, `PROCEDURE` blocks
- The order of the instructions is relevant

Sequential code

- Sequential code is the code enclosed in `PROCESS`, `FUNCTION`, `PROCEDURE` blocks
- The order of the instructions is relevant
- Sequential code can be used to implement combinational and sequential synchronous/asynchronous circuits

Sequential code is a key aspect of VHDL coding

Sequential code

- Sequential code is executed in order

Sequential code

- Sequential code is executed in order
- Is there a kind of CPU (with a sort of centralized ALU)?
- Is there a kind of program counter (or instruction pointer) register scanning the sequential statements in order?

Sequential code

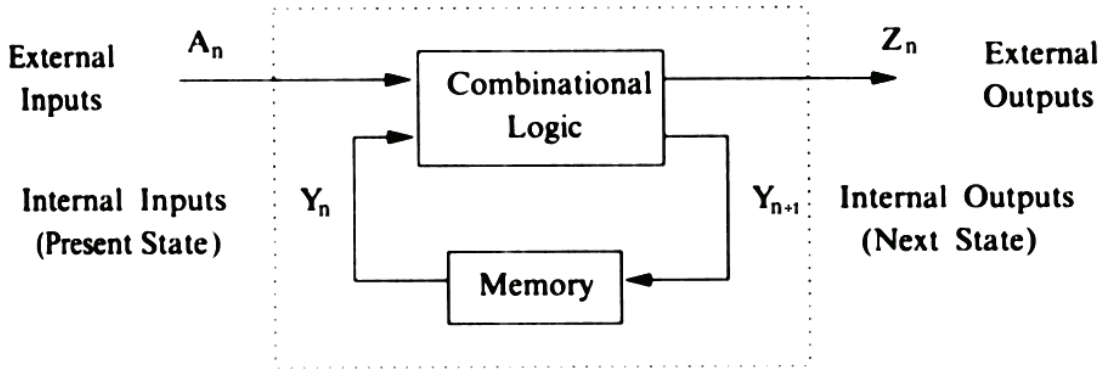
- Sequential code is executed in order ... not really!
- Is there a kind of CPU (with a sort of centralized ALU)? ... definitely not!
- Is there a kind of program counter (or instruction pointer) register scanning the sequential statements in order? ... absolutely not!

Sequential code

- Sequential code is executed in order ... not really!
- Is there a kind of CPU (with a sort of centralized ALU)? ... definitely not!
- Is there a kind of program counter (or instruction pointer) register scanning the sequential statements in order? ... absolutely not!

What happens then?

- Instructions are **"virtually" executed in order**
- The synthesizer infers an **equivalent sequential circuit** (like in the diagram) that mimics the execution of the "program" (sequence of statements within a sequential block)

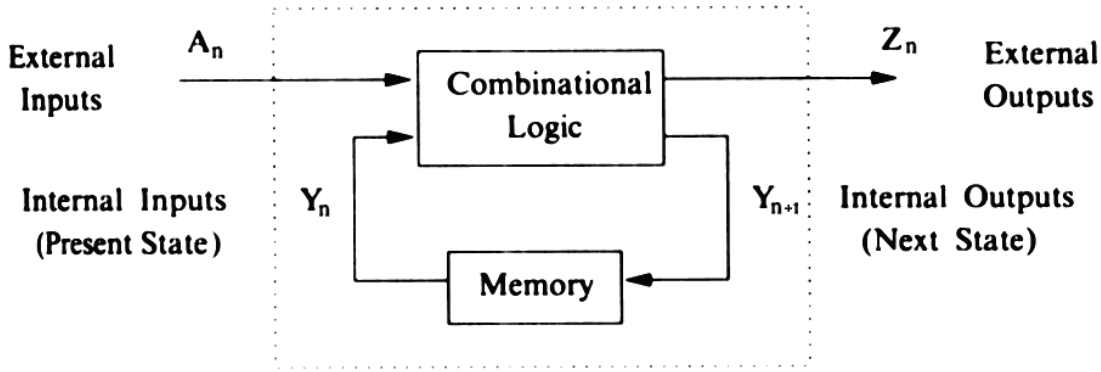


Sequential code

- Sequential code is executed in order ... not really!
- Is there a kind of CPU (with a sort of centralized ALU)? ... definitely not!
- Is there a kind of program counter (or instruction pointer) register scanning the sequential statements in order? ... absolutely not!

What happens then?

- Instructions are **"virtually" executed in order**
- The synthesizer infers an **equivalent sequential circuit** (like in the diagram) that mimics the execution of the "program" (sequence of statements within a sequential block)



- The architecture is still parallel but has a state (memory), if needed

Outline

- Concurrent/sequential code
  - Concurrent code
  - Sequential code

- Sequential code: statements
  - Operators
  - `PROCESS`
  - `IF`
  - `CASE`
  - More examples

Operators

VHDL operators can be used both in concurrent and sequential code.

Family	Operators	Operands type	Return type
Logical	`AND OR XOR NAND NOR XNOR NOT`	`BOOLEAN BIT STD_LOGIC STD_ULOGIC` (and their array extension)	same as operands
Shift	`SLL SRL SLA SRA ROL ROR`	`BOOLEAN BIT STD_LOGIC STD_ULOGIC` (and their array extension) first operand / `INTEGER` second operand	same as first operand
Relational	`= /= < > <= >=`	Any scalar type or discrete array type	`BOOLEAN`
Arithmetic	`+ - * / mod rem`	`SIGNED UNSIGNED INTEGER REAL PHYSICAL`	same as operands
Concatenation	`&`	array or element type	array type

PROCESS with WAIT

This form of PROCESS block has the following syntax:

```
[label:] PROCESS
  [VARIABLE declarations]
BEGIN
  WAIT UNTIL condition;
  (sequential code)
END PROCESS [label];
```

PROCESS with WAIT

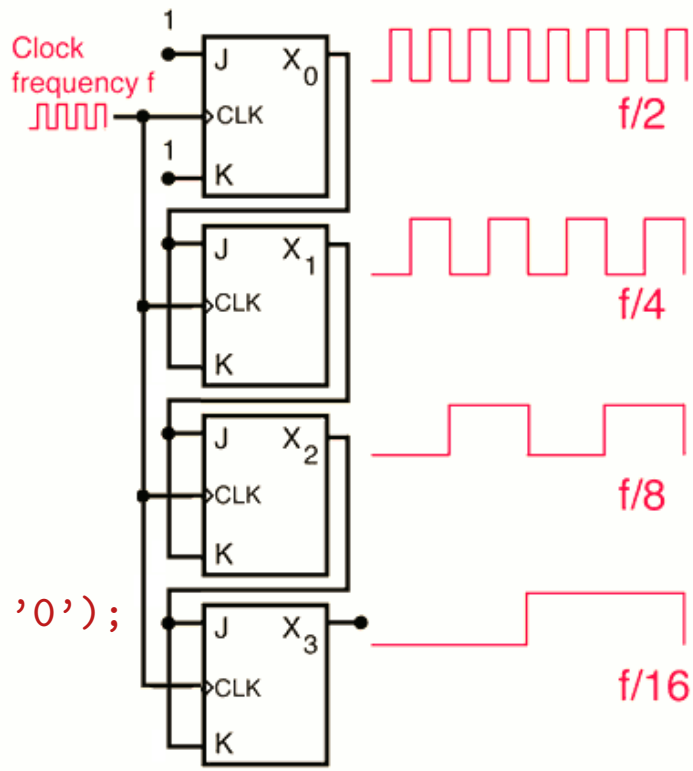
This form of PROCESS block has the following syntax:

```
[label:] PROCESS
  [VARIABLE declarations]
BEGIN
  WAIT UNTIL condition;
  (sequential code)
END PROCESS [label];
```

- A PROCESS is a section of sequential VHDL code and must be placed in the architecture body
- In this form (with WAIT), it is executed every time the condition related to WAIT is fulfilled
- It can have a label (for code readability)
- It can have IF CASE LOOP instructions (explained in the following) but no WHEN (concurrent code)

PROCESS with WAIT

Example: Frequency divider




```
LIBRARY ieee;
USE ieee.std_logic_1164.all; -- for STD_LOGIC and RISING_EDGE
USE ieee.numeric_std.all;    -- for UNSIGNED

ENTITY fr_div IS PORT (
  mclk : IN  STD_LOGIC;
  q : OUT STD_LOGIC );
END fr_div;

ARCHITECTURE behav OF fr_div IS
  SIGNAL cnt : UNSIGNED (25 downto 0):=(others => '0');
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL RISING_EDGE(mclk); -- from IEEE 1164 library
    cnt <= cnt + 1;
  END PROCESS;
  q <= cnt(25); -- bit 25 of cnt has a frequency 1/(2^26) that of mclk
END behav;
```

PROCESS with sensitivity list

This form of PROCESS block has the following syntax:

```
[label:] PROCESS (sensitivity_list)
  [VARIABLE declarations]
BEGIN
  (sequential code)
END PROCESS [label];
```

PROCESS with sensitivity list

This form of PROCESS block has the following syntax:

```
[label:] PROCESS (sensitivity_list)
  [VARIABLE declarations]
BEGIN
  (sequential code)
END PROCESS [label];
```

- A PROCESS is a sequential section of VHDL code and must be placed in the architecture body
- The sensitivity list is a list of signals, e.g. `(ck, rst)`
- In this form (with sensitivity list), statements inside the block are executed every time any of the signals in the sensitivity list changes
- It can (must) have IF CASE LOOP instructions (explained in the following) but no WHEN (concurrent code)

IF

The IF statement in VHDL has a syntax and a functionality that are very similar to other languages:

```
IF condition1 THEN
  (statements1)
ELSIF condition2 THEN
  (statements2)
...
ELSE
  (statementsN)
END IF;
```

IF

The IF statement in VHDL has a syntax and a functionality that are very similar to other languages:

```
IF condition1 THEN
  (statements1)
ELSIF condition2 THEN
  (statements2)
...
ELSE
  (statementsN)
END IF;
```

- As soon as the first TRUE condition is encountered, the corresponding statements are executed
- The conditions are BOOLEAN expressions and do not need to be mutually exclusive
- If no condition is satisfied, the statements following the last ELSE are executed

For these reasons, IF resembles WHEN...ELSE

IF

The IF statement in VHDL has a syntax and a functionality that are very similar to other languages:

```
IF condition1 THEN
  (statements1)
ELSIF condition2 THEN
  (statements2)
...
ELSE
  (statementsN)
END IF;
```

- As soon as the first TRUE condition is encountered, the corresponding statements are executed
- The conditions are BOOLEAN expressions and do not need to be mutually exclusive
- If no condition is satisfied, the statements following the last ELSE are executed

For these reasons, IF resembles WHEN...ELSE but:

- IF can only be used in sequential code while WHEN...ELSE only in concurrent
- The statements1..N are not limited to one assignment but can be more assignments, a nested IF, ...

CASE

We briefly introduce the CASE statement:

```
CASE identifier IS
WHEN value1 =>
  (statements1)
WHEN value2 =>
  (statements2)
...
END CASE;
```

CASE is basically the sequential counterpart of WITH...SELECT...WHEN

CASE

We briefly introduce the CASE statement:

```
CASE identifier IS
WHEN value1 =>
  (statements1)
WHEN value2 =>
  (statements2)
...
END CASE;
```

CASE is basically the sequential counterpart of WITH...SELECT...WHEN but:

- CASE can only be used in sequential code while WITH...SELECT...WHEN only in concurrent
- The statements1..N are not limited to one assignment but can be more assignments, a nested IF, CASE, ...

More examples: Counter 0 to 5

```
LIBRARY ieee;
USE ieee.std_logic_1164.all; -- for STD_LOGIC and RISING_EDGE
USE ieee.numeric_std.all;    -- for UNSIGNED

ENTITY cnt5 IS PORT (
  ck  : IN  STD_LOGIC;
  cnt : OUT UNSIGNED (2 downto 0));
END cnt5;

ARCHITECTURE Behavioural OF cnt5 IS
SIGNAL q: UNSIGNED (2 downto 0);
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL RISING_EDGE(ck);
    IF q = 5 THEN
      q <= "000";
    ELSE
      q <= q + 1;
    END IF;
  END PROCESS;
  cnt <= q;
END Behavioural;
```

More examples: Counter with sync/async reset

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

ENTITY cnt_rst IS PORT (
  ck : IN  STD_LOGIC;
  rst : IN  STD_LOGIC;
  cnt : OUT UNSIGNED (2 downto 0));
END cnt_rst;

ARCHITECTURE Behavioural OF cnt_rst IS
SIGNAL q: UNSIGNED (2 downto 0);
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL RISING_EDGE(ck);
    IF rst = '1' THEN
      q <= "000";
    ELSE
      q <= q + 1;
    END IF;
  END PROCESS;
  cnt <= q;
END Behavioural;
```

More examples: Counter with sync/async reset

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

ENTITY cnt_rst IS PORT (
  ck : IN  STD_LOGIC;
  rst : IN  STD_LOGIC;
  cnt : OUT UNSIGNED (2 downto 0));
END cnt_rst;

ARCHITECTURE Behavioural OF cnt_rst IS
SIGNAL q: UNSIGNED (2 downto 0);
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL RISING_EDGE(ck);
    IF rst = '1' THEN
      q <= "000";
    ELSE
      q <= q + 1;
    END IF;
  END PROCESS;
  cnt <= q;
END Behavioural;
```

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

ENTITY cnt_rst IS PORT (
  ck : IN  STD_LOGIC;
  rst : IN  STD_LOGIC;
  cnt : OUT UNSIGNED (2 downto 0));
END cnt_rst;

ARCHITECTURE Behavioural OF cnt_rst IS
SIGNAL q: UNSIGNED (2 downto 0);
BEGIN
  PROCESS (ck,rst)
  BEGIN
    IF rst = '1' THEN
      q <= "000";
    ELSIF RISING_EDGE(ck) THEN
      q <= q + 1;
    END IF;
  END PROCESS;
  cnt <= q;
END Behavioural;
```

Thank you
for your attention

Sequential code

Xiaojun Zhai
xzhai@essex.ac.uk

School of Computer Science and Electronic Engineering
University of Essex (UK)

CE339/CE869 - Lecture 3
Jan 2020