This lecture presents some Partitioning-based System Design Methodologies.
2.1 – Partitioning-based System Design Methodologies

**Basic design strategy**

Whenever the system to be implemented is complex enough not to be easily implemented by a single unit (i.e., a process), it’s convenient to partition it in *functional units*.

**Side effects**

Partitioning the system in several units improves design re-usability.
Partitioning hints

It’s usually convenient to introduce, at least:

- an Input Unit, charged of communicating with the external world and of transforming input signals in an internal “suitable” form
- an Output Unit, charged of transforming internal signals in the form required by the external world
- a Main body, charged of performing the target operations.

Note

Partitioning is a typical example of the top-down design methodology, through a hierarchical decomposition, early introduced in lecture 2.1
Example of hierarchical decomposition

Example of hierarchical decomposition
Example of hierarchical decomposition
Link with the hierarchy

Stopping condition

Partitioning stops when the “leaf” sub-systems are simpler enough to be “easily” implemented.
Basic assumptions

In the following, unless differently stated explicitly, we shall assume that:

• all the units run concurrently, clocked by a same master clock signal
• all the memory elements of all the units are clocked by the same clock edge (e.g., the rising one).

Design steps

Identify the Input / Output signals of the target system, in terms of their names, type and width.
Example

Elements definition: I/O

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>in</td>
<td>std_logic</td>
</tr>
<tr>
<td>reset</td>
<td>in</td>
<td>std_logic</td>
</tr>
<tr>
<td>data_in</td>
<td>in</td>
<td>std_logic</td>
</tr>
<tr>
<td>out_en</td>
<td>out</td>
<td>std_logic</td>
</tr>
<tr>
<td>counter</td>
<td>out</td>
<td>integer</td>
</tr>
</tbody>
</table>
### Elements definition:

To improve re-usability, never use numbers; always resort to constants.

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>in</td>
<td>std_logic single bit</td>
</tr>
<tr>
<td>reset</td>
<td>in</td>
<td>std_logic single bit</td>
</tr>
<tr>
<td>data_in</td>
<td>in</td>
<td>std_logic single bit</td>
</tr>
<tr>
<td>out_en</td>
<td>out</td>
<td>std_logic single bit</td>
</tr>
<tr>
<td>counter</td>
<td>out</td>
<td>integer</td>
</tr>
</tbody>
</table>

### Elements definition: constants

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT_WIDTH</td>
<td>constant</td>
<td>integer</td>
</tr>
</tbody>
</table>
### Elements definition: I/O

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>in</td>
<td>std_logic: single bit</td>
</tr>
<tr>
<td>reset</td>
<td>in</td>
<td>std_logic: single bit</td>
</tr>
<tr>
<td>data_in</td>
<td>in</td>
<td>std_logic: single bit</td>
</tr>
<tr>
<td>out_en</td>
<td>out</td>
<td>std_logic: single bit</td>
</tr>
<tr>
<td>counter</td>
<td>out</td>
<td>integer: 0 to OUT_WIDTH-1</td>
</tr>
</tbody>
</table>

### Design steps

Identify the functional units the system has to be partitioned in (Top-level partitioning)
11.4

Example of Top level partitioning

Input Unit

Main Unit

data_in

counter

out_en

Design steps

Define the communication channels among the functional units, in terms of their type and width.
Example

Design steps

Implement each functional unit separately.
Unit implementation

Describe the behavior of each unit as a process.

Two cases can usually occur:

- The unit is rather “simple”, i.e., its behavior can be described as a straight piece of behavioral VHDL, without resorting to the introduction of “states”
- The unit is more “complex”, i.e., its behavior can be described only by resorting to the introduction of “states” the process evolves through.

An example

A ROM memory (1K x 16) stores 1024 data of 16 bits each, represented according to the “2’s complement” notation.

On a serial transmission line, 16-bits numbers are transmitted synchronously w.r.t. a clock signal clk, one bit per clock cycle, according to the following protocol:

- Numbers are transmitted LSB first
- Each new number has a 4 bit header : “1111”
- 6,000+ clock cycles occur between two consecutive transmitted numbers.
An example (cont’d)

A system is to be designed such that, whenever a new number $N$ is received via the input line, determines the overall number $T$ of data $D$, stored in the ROM, are such that

$$|D| = N.$$  

At the operation completion, $T$ has to be visualized, on a proper set of 7-segment displays, till getting the next number $N$.

Top level view
It is in charge of converting the serial input in a form suitable for its analysis by the Main Unit.
It gets the new number N, performs the required analysis and provides the results to the Display Unit.

It converts the information provided by the Main Unit into proper control signals for the displays.
Top level partitioning (cont’d)

Input Unit  →  Main Unit  →  Display Unit

Design of the Main Unit

Main Unit

N  →  match_count

strobe  →  display_en
### Elements definition: constants

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM_WIDTH</td>
<td>constant</td>
<td>integer</td>
</tr>
<tr>
<td>ROM_SIZE</td>
<td>constant</td>
<td>integer</td>
</tr>
</tbody>
</table>

### Elements definition: I/O

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>in</td>
<td>std_logic</td>
</tr>
<tr>
<td>reset</td>
<td>in</td>
<td>std_logic</td>
</tr>
<tr>
<td>N</td>
<td>in</td>
<td>integer</td>
</tr>
<tr>
<td>strobe</td>
<td>in</td>
<td>std_logic</td>
</tr>
<tr>
<td>display_en</td>
<td>out</td>
<td>std_logic</td>
</tr>
<tr>
<td>match_count</td>
<td>out</td>
<td>integer</td>
</tr>
</tbody>
</table>
### Elements definition: internal signals

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc</td>
<td>signal</td>
<td>integer</td>
</tr>
<tr>
<td>address</td>
<td>signal</td>
<td>integer</td>
</tr>
</tbody>
</table>

### Intermediate STG of the Main Unit

```
match_count <= 0  
acc <= 0          
address <= 0    
display_en <= ‘0’
```

init
**init**

*If strobe = 1*

- Inizialize all the sequential elements:
- \[\text{match count} \leq 0;\]
- \[\text{acc} \leq 0;\]
- \[\text{address} \leq 0\]

- Turn the display off:
  \[\text{display_en} \leq '0'\]

*else*

**get new word**

\[\text{acc} \leq \text{ROM(address)}\]

**check**

*abs(acc) \leq N*

- \[\text{match_count} \leq \text{match_count} + 1;\]

*else*

**All the cells have been considered**

- \[\text{address} = \text{ROM_SIZE} - 1\]

**Turn the display on:**

- \[\text{display_en} \leq '1'\]
- \[\text{address} \leq \text{address} + 1;\]

*init*

*get new word*