Lecture 8

RTL Design Methodology

Transition from Pseudocode & Interface to a Corresponding Block Diagram
Structure of a Typical Digital System

Datapath (Execution Unit)

Data Inputs

Control & Status Inputs

Control Signals

Status Signals

Data Outputs

Control & Status Outputs

Controller (Control Unit)
Hardware Design with RTL VHDL

Pseudocode

Interface

Datapath

Block diagram

Controller

Block diagram

State diagram or ASM chart

VHDL code

VHDL code

VHDL code
Steps of the Design Process

1. Text description
2. Interface
3. Pseudocode
4. Block diagram of the Datapath
5. Interface with the division into the Datapath and the Controller
6. ASM chart of the Controller
7. RTL VHDL code of the Datapath, the Controller, and the Top Unit
8. Testbench of the Datapath, the Controller, and the Top Unit
9. Functional simulation and debugging
10. Synthesis and post-synthesis simulation
11. Implementation and timing simulation
12. Experimental testing
Steps of the Design Process Practiced in Class Today

1. Text description
2. Interface
3. **Pseudocode**
4. Block diagram of the Datapath
5. **Interface with the division into the Datapath and the Controller**
6. ASM chart of the Controller
7. RTL VHDL code of the Datapath, the Controller, and the Top Unit
8. Testbench of the Datapath, the Controller, and the Top Unit
9. Functional simulation and debugging
10. Synthesis and post-synthesis simulation
11. Implementation and timing simulation
12. Experimental testing
Steps of the Design Process Covered Before the Midterm Exam

1. Text description
2. Interface
3. Pseudocode
4. Block diagram of the Datapath
5. Interface with the division into the Datapath and the Controller
6. ASM chart of the Controller
7. RTL VHDL code of the Datapath, the Controller, and the Top Unit
8. Testbench of the Datapath, the Controller, and the Top Unit
9. Functional simulation and debugging
10. Synthesis and post-synthesis simulation
11. Implementation and timing simulation
12. Experimental testing
Statistics example
Circuit Interface

clk  →  done
reset  →
din  →  dout
        n
        2
        dout_mode

go  →
no_1 = no_2 = no_3 = sum = 0
for i=0 to k-1 do
    sum = sum + din
    if din > no_1 then
        no_3 = no_2
        no_2 = no_1
        no_1 = din
    elseif (din > no_2) then
        no_3 = no_2
        no_2 = din
    elseif (din > no_3) then
        no_3 = din
        no_2 = din
    end if
end for
avr = sum / k
## Interface Table

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>1</td>
<td>System clock.</td>
</tr>
<tr>
<td>reset</td>
<td>1</td>
<td>System reset.</td>
</tr>
<tr>
<td>din</td>
<td>n</td>
<td>Input Data.</td>
</tr>
<tr>
<td>go</td>
<td>1</td>
<td>Signal active high for k clock cycles during which Input Data is read by the circuit.</td>
</tr>
<tr>
<td>done</td>
<td>1</td>
<td>Signal set to high after the output is ready.</td>
</tr>
<tr>
<td>dout</td>
<td>n</td>
<td>Output dependent on the dout_mode input.</td>
</tr>
<tr>
<td>dout_mode</td>
<td>2</td>
<td>Control signal determining value available at the output. 00: avr, 01: no_1, 10: no_2, 11: no_3.</td>
</tr>
</tbody>
</table>
Sorting example
Sorting - Required Interface

- Clock
- Resetn
- DataIn
- RAdd
- WrInit
- S (0=initialization, 1=computations)
- Rd

Input:
- Clock
- Resetn
- DataIn
- RAdd
- WrInit
- S (0=initialization, 1=computations)
- Rd

Output:
- DataOut
- Done

Sorting
## Sorting - Required Interface

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>1</td>
<td>System clock</td>
</tr>
<tr>
<td>Resetn</td>
<td>1</td>
<td>System reset – clears internal registers. Active low.</td>
</tr>
<tr>
<td>DataIn</td>
<td>N</td>
<td>Input data bus</td>
</tr>
<tr>
<td>RAdd</td>
<td>L</td>
<td>Address of the internal memory where input data is stored</td>
</tr>
<tr>
<td>WrInit</td>
<td>1</td>
<td>Synchronous write control signal</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
<td>Operating mode: 0 = initialization, 1 = computations.</td>
</tr>
<tr>
<td>Rd</td>
<td>1</td>
<td>Read enable. 0 = high impedance on the output bus, 1 = valid output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>on the output data bus.</td>
</tr>
<tr>
<td>DataOut</td>
<td>N</td>
<td>Output data bus used to read results</td>
</tr>
<tr>
<td>Done</td>
<td>1</td>
<td>Asserted when all results are ready</td>
</tr>
</tbody>
</table>
Simulation results for the sort operation (1)
Loading memory and starting sorting
Simulation results for the sort operation (2)
Completing sorting and reading out memory
## Sorting - Example

<table>
<thead>
<tr>
<th>Address</th>
<th>Before sorting</th>
<th>During Sorting</th>
<th>After sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i=0 j=1</td>
<td>i=0 j=2</td>
<td>i=1 j=3</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Legend:
- \( M_i \): position of memory indexed by \( i \)
- \( M_j \): position of memory indexed by \( j \)
Pseudocode

FOR k = 4
[load input data]
for i = 0 to 2 do
    A = M_i;
    for j = i + 1 to 3 do
        B = M_j;
        if B < A then
            M_i = B;
            M_j = A;
            A = M_i;
        endif;
    endfor;
endfor;
[read output data]

FOR any k ≥ 2
[load input data]
for i = 0 to k-2 do
    A = M_i;
    for j = i + 1 to k - 1 do
        B = M_j;
        if B < A then
            M_i = B;
            M_j = A;
            A = M_i;
        endif;
    endfor;
endfor;
[read output data]
Pseudocode

wait for s=1
for i=0 to k-2 do
    A = M_i
    for j=i+1 to k-1 do
        B = M_j
        if A > B then
            M_i = B
            M_j = A
            A = M_i
        end if
    end for
end for
Done
wait for s=0
go to the beginning
Block diagram of the Execution Unit
Interface with the division into the Datapath and the Controller

Datapath

Controller

DataIn RAddr WrInit Rd Clock Rsetn

N L

Done

DataOut

AgtB zi zj

Wr Li Ei Lj Ej Ej EA EB Bout Csel
RC5
One of the fastest ciphers

Basic operations:

Rotation by a variable number of bits

Addition modulo $2^w$
where $w$ is the size of operands $A$ and $B$

$$C = A + B \mod 2^w$$
RC5  w/r/b

w - word size in bits  \( w = 16, 32, 64 \)

input/output block = 2 words = 2\( \cdot \)w bits

Typical value:
\( w=32 \Rightarrow 64\)-bit input/output block

r - number of rounds

b - key size in bytes  \( 0 \leq b \leq 255 \)

key size in bits = 8\( \cdot \)b bits

Recommended version:  RC5 32/12/16

64 bit block
12 rounds
128 bit key
Pseudocode

Split M into two halves A and B, \( w \) bits each

\[
A = A + S[0] \\
B = B + S[1]
\]

for \( j = 1 \) to \( r \) do

\[
A' = ((A \oplus B) \ll B) + S[2j] \\
B' = ((B \oplus A') \ll A') + S[2j+1]
\]

\[
A = A' \\
B = B'
\]

\[
C = A \| B
\]
Notation

A, B, A’, B’ = w-bit variables
S[2j], S[2j+1] = a pair of round keys, each round key is a w-bit variable
⊕ = an XOR of two w-bit words
+ = unsigned addition mod $2^w$
A <<< B = rotation of the variable A by a number of positions given by the current value of the variable B
A || B = concatenation of A and B

The algorithms has two parameters:
• r = number of rounds (e.g., 3)
• w = word size (always a power of 2, e.g., $w = 2^4 = 16$)
Circuit Interface

RC5

clk
reset
M
write_M
Si
write_Si
i

2w

2w
C
Done

w
m
# Circuit Interface

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>1</td>
<td>System clock.</td>
</tr>
<tr>
<td>reset</td>
<td>1</td>
<td>System reset – clears internal registers.</td>
</tr>
<tr>
<td>M</td>
<td>$2w$</td>
<td>Message block.</td>
</tr>
<tr>
<td>write_M</td>
<td>1</td>
<td>Synchronous write control signal for the message block M. After the block M is written to the RC5 unit, the encryption of M starts automatically.</td>
</tr>
<tr>
<td>Si</td>
<td>$w$</td>
<td>Round key $S[i]$ loaded to one of the two internal memories. The first memory stores values of $S[i=2j]$, i.e., only round keys with even indices. The second memory stores values of $S[i=2j+1]$, i.e. only round keys with odd indices.</td>
</tr>
<tr>
<td>write_Si</td>
<td>1</td>
<td>Synchronous write control signal for the round key $S[i]$.</td>
</tr>
<tr>
<td>i</td>
<td>$m$</td>
<td>Index of the round key $S[i]$ loaded using input Si.</td>
</tr>
<tr>
<td>C</td>
<td>$2w$</td>
<td>Ciphertext block = Encrypted block M.</td>
</tr>
<tr>
<td>DONE</td>
<td>1</td>
<td>Asserted when ciphertext is ready, and available at the output.</td>
</tr>
</tbody>
</table>

**Note:**

$m$ is a size of index $i$. It is a minimum integer, such that $2^m \geq 2r+2$. 
Protocol (1)

An external circuit first loads all round keys $S[0], S[1], S[2], \ldots, S[2r], [2r+1]$ to the two internal memories of the RC5 unit.

The first memory stores values of $S[i=2j]$, i.e., only round keys with even indices. The second memory stores values of $S[i=2j+1]$, i.e. only round keys with odd indices.

Loading round keys is performed using inputs: $Si, i, write_Si, clk$.

Then, the external circuits, loads a message block $M$ to the RC5 unit, using inputs: $M, write_M, clk$.

After the message block $M$ is loaded to the RC5 unit, the encryption starts automatically.
Protocol (2)

When the encryption is completed, signal Done becomes active, and the output C changes to the new value of the ciphertext.

The output C keeps the last value of the ciphertext at the output, until the next encryption is completed. Before the first encryption is completed, this output should be equal to zero.
Assumptions

• one round of the main for loop of the pseudocode executes in one clock cycle
• you can access only one position of each internal memory of round keys per clock cycle

As a result, the entire encryption of a single message block $M$ should last $r+1$ clock cycles.