Design of Microprogrammed Control Units (MCU) using VHDL Description
A hardwired control unit accomplishes a conditional transfer of control from step 5 to step 6 (STOP example).

This is the one-hot state assignment approach in which a separate flip-flop is dedicated to each state in the controller.
Hardwired control unit

DATA PATH

UNIT

CONTROL

UNIT

D  Q
¬Q

&

D  Q
¬Q

S5

S6

A

CLOCK

Arvutitehnika erikursus
Microprogrammed control unit

DATA PATH UNIT

CONTROL UNIT

AGP

MAR

ROM

MIR

conditions

a

w-b

a

b

SIGNALS

Arvutitehnika erikursus 4
In a microprogrammed control unit, the values of control signals are read from an appropriate address location in a ROM (instead of being generated by combinational logic gates). The contents of each address in the ROM are called a control word.
Basic microprogrammed control unit (example)

We use two-way branching address generation.

Each control word contains 64 bits of data.

The condition select field (CW(63:40)) contains information needed to select the input condition signal that is used to compute the address of the next instruction.

In the BMCU we use a “one-hot” code to select the condition signal.

<table>
<thead>
<tr>
<th>63</th>
<th>40</th>
<th>39</th>
<th>32</th>
<th>31</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>NAD</td>
<td>LCS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Next Address field

If Ci is selected and Ci = 1 then address NAD field is the address of the next control word to be fetched. If Ci = 0, then the memory address [MAR] is incremented to compute the next address.
Control word

Every designer must observe the constraints imposed by the controller. These constraints include such things as the number of control signals that can be generated, the number of status signals that can be handled, and limits imposed by the address generation logic.

```
   63  62  61  40
  |   |   |   |
 CS23 CS22 CS21 . . . CS0

Condition Select Field (CS)

  31  30  29  0
 |   |   |   |
 LCS31 LCS30 LCS29 . . . LCS0

Level Control Signal Field (LCS)

The maximum number of control signals is 32. ROM size is 256 words.
Address generation logic

The correct sequence of control signals is obtained by generating the proper sequence of addresses at the ROM inputs.

The address generating logic varies considerably from design to design and is a major contributor to the constraints imposed by the controller.

Timing signals for the control unit and the data unit must be closely coordinated.

The controller works best if the sequencing of addresses is relatively simple. Usually, the address generating logic circuit (AG) is a counter with the option to parallel load a new address when one wishes to jump to a new point in the control sequence. If the controller usually goes to the next higher sequential address for the next set of control signal values (increment) with just an occasional need to parallel load a new address (branch), the AG can be a low complexity circuit.
Address generation logic organization (example)

Address Generation Logic for BMCU can be designed using a vector multiplexer.

\[ AS = (CW(63))(C23) + (CW(62))(C22) + \ldots + (CW(40))(C0) \]

\[
\begin{align*}
\uparrow & \quad \uparrow & \quad \uparrow \\
CS(23) & \quad CS(22) & \quad CS(0)
\end{align*}
\]
Synthesis of microprogrammed controllers

1. Prepare a table showing all transitions from each state and the conditions that define each transition by scanning the statements in the VHDL description.
2. Make a list of the conditions from step 1. After the assignment of ROM addresses to states, some of these conditions may not be needed.
3. Identify a reset state.
4. Assign a ROM address to each state.
5. Remove the redundant signals from the condition list created in step 2 and assign the condition signals to the condition inputs in an arbitrary manner.
6. Create a list of transfers during each state and a list of outputs required during each state.
7. From the list of transfers and outputs created in step 6, generate a list of control signals needed to time the transfers and outputs.
8. Draw a block diagram showing the condition and control signals produced.
9. Determine the ROM program using the information in the lists produced in steps 1-8. This procedure is similar to that performed by an assembler program.
Table of transitions created by executing step 1

<table>
<thead>
<tr>
<th>PS</th>
<th>NS</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>R + ¬A</td>
</tr>
<tr>
<td>S1`</td>
<td>¬R</td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>¬R</td>
</tr>
<tr>
<td>S0</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
<td>¬R</td>
</tr>
<tr>
<td>S0</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>S4</td>
<td>¬R</td>
</tr>
<tr>
<td>S0</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>S5</td>
<td>¬R</td>
</tr>
<tr>
<td>S0</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>S1</td>
<td>¬R &amp; A</td>
</tr>
<tr>
<td>S0</td>
<td>R + ¬A</td>
<td></td>
</tr>
</tbody>
</table>
A list of the conditions

The list of conditions produced by step 2 is:

\[ R + \neg A \]
\[ \neg R \& A \]
\[ \neg R \]
\[ R \]

3 step.

The reset state S0 is defined in the problem specification
Assignment of a ROM address to each state (4)

Although the optimum assignment of addresses in step 4 is a difficult problem, the following simple approach will produce a reasonable assignment in a short time.

Programming the ROM for a controller is often accomplished using a tool similar to an assembler.

4.1. Let variable P be the current state.
Let LC represent the location counter which contains the next available memory address.
4.2. Assign state P to address LC. If all states are assigned to memory addresses, then stop.
Otherwise go to step 3.
4.3. To determine the number of ROM addresses needed to implement state P, let NSP be the number of next states for state P.
The number of ROM addresses needed to P

4.3.1. If NSP=1, set LC=LC+1. Only one ROM address is needed. If the next state is not yet assigned, set P equal to the next state, mark the next state with an I (for increment), and go to step 2. If the next state is already assigned to a ROM location, then mark the next state with B (for branch), arbitrarily select any unassigned state for P, and go to step 2.

4.3.2. If NSP>1, and if all states are already assigned to ROM addresses, then NSP memory locations are required to implement state P. Mark all next states with B, set LC=LC+NSP, arbitrarily select any unassigned state for P, and go to step2.

4.3.3. If NSP>1, and if at least one of the next states of P are not assigned to a ROM address, then arbitrarily set Q equal to one of the unassigned next states of P (this is where the optimality breaks down-choice for Q). NSP-1 memory locations are required for state P. Mark Q with I, mark all other next states of P with B, set LC=LC+NSP-1, set P=Q, and go to step2.
ROM address assignment

P:=Reset  LC:=0

LC:=P

NSP=1

no

yes

LC:=LC+1

I

no

yes

Ass?

B

P:=ArbUnasSt

All are Ass?

no

yes

Q:=ArbUnasSt

Q mark I

Other B

LC:=LC+NSP

All mark B

LC:=LC+NSP

P:=Q

P-current state variable; LC-location counter; NSP-the number of next states for P; Q-state variable;
### ROM address assignment (example STOP)

<table>
<thead>
<tr>
<th>PS</th>
<th>NS</th>
<th>Condition</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>R + ¬A</td>
<td>B</td>
</tr>
<tr>
<td>S1`</td>
<td>¬R</td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>¬R</td>
<td>I</td>
</tr>
<tr>
<td>S0</td>
<td>R</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
<td>¬R</td>
<td>I</td>
</tr>
<tr>
<td>S0</td>
<td>R</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>S3</td>
<td>S4</td>
<td>¬R</td>
<td>I</td>
</tr>
<tr>
<td>S0</td>
<td>R</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>S4</td>
<td>S5</td>
<td>¬R</td>
<td>I</td>
</tr>
<tr>
<td>S0</td>
<td>R</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>S5</td>
<td>S1</td>
<td>¬R &amp; A</td>
<td>B</td>
</tr>
<tr>
<td>S0</td>
<td>R + ¬A</td>
<td></td>
<td>B</td>
</tr>
</tbody>
</table>
ROM addresses assigned to steps (Step 4)

<table>
<thead>
<tr>
<th>State</th>
<th>ROM address</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>2</td>
</tr>
<tr>
<td>S3</td>
<td>3</td>
</tr>
<tr>
<td>S4</td>
<td>4</td>
</tr>
<tr>
<td>S5</td>
<td>5</td>
</tr>
</tbody>
</table>

The next available address would be address 7 because state S5 requires two memory locations.
Assignment of conditions (Step 5)

An arbitrary assignment of conditions to the condition input of the control unit:

<table>
<thead>
<tr>
<th>Input</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>C23</td>
<td>R</td>
</tr>
<tr>
<td>C22</td>
<td>R + ¬A</td>
</tr>
<tr>
<td>C21</td>
<td>¬R &amp; A</td>
</tr>
</tbody>
</table>

The condition signals that must be passed from the data unit to the control unit are those that correspond to lines of the table (list of transitions) that are marked with a B (branch).

Note that condition ¬R is not needed because all next states requiring condition ¬R are marked with I (increment) indicating that the transitions will be accomplished by incrementing the MAR.
## A list of transfers (Step 6)

<table>
<thead>
<tr>
<th>State</th>
<th>Transfer</th>
<th>Cond</th>
<th>Done</th>
<th>Cond</th>
<th>Z</th>
<th>Cond</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S1</td>
<td>Shift SR</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S2</td>
<td>Shift SR</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S3</td>
<td>Shift SR</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S4</td>
<td>Shift SR</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S5</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>SR</td>
<td>1</td>
</tr>
</tbody>
</table>

Note that condition 1 means an unconditional transfer or output. In this example, all transfers and outputs are unconditional.

Steps 1 and 6 are separated into two steps for clarity only.
### A list of control signals (Step 7)

<table>
<thead>
<tr>
<th>Output</th>
<th>Signal</th>
<th>Transfer/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCS31</td>
<td>SHIFT</td>
<td>SR &lt;= D &amp; SR[3:1]</td>
</tr>
<tr>
<td>LCS30</td>
<td>DONE_CONTROL</td>
<td>DONE = 1, Z = SR</td>
</tr>
</tbody>
</table>

The assignment of control signals to the output ports is arbitrary.

Since there are only two distinct output situations, the Outputs can be controlled by one control signal called DONE_CONTROL. There is only one transfer required, indicated by control signal SHIFT.
Block diagram showing intermodule signals

CONTROL UNIT

DATA UNIT

DONE

LCS31 LCS30

F

C0

C20...

C23 C22 C21

R A D

DONE_CONTROL

SHIFT

4 Z

...
STOP block-diagram

conditions

C21, C22, C23

CONTROL UNIT

control signals

SHIFT, DONE_CONTROL

status signals

microoperations

R
A
D

DATA PATH UNIT

DONE
Z
### ROM contents for control ROM of STOP

<table>
<thead>
<tr>
<th>State</th>
<th>Address</th>
<th>CS</th>
<th>NAD</th>
<th>LCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0</td>
<td>23</td>
<td>7</td>
<td>31</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>22</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>S2</td>
<td>2</td>
<td>21</td>
<td>20</td>
<td>29</td>
</tr>
<tr>
<td>S3</td>
<td>3</td>
<td>20</td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td>S4</td>
<td>4</td>
<td>19</td>
<td>20</td>
<td>27</td>
</tr>
<tr>
<td>S5</td>
<td>5</td>
<td>18</td>
<td>20</td>
<td>26</td>
</tr>
<tr>
<td>S6</td>
<td>6</td>
<td>17</td>
<td>20</td>
<td>25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CWORD bit</th>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>...</th>
<th>40</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Address</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Address</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Address</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

- **C22**-the condition for branch (labeled B) operation \((R + \neg A)\);
- **C21**---\((\neg R & A)\);
- **LCS31**---SHIFT;
- **LCS30**---DONE_CONTROL
Hardwired vs microcoded control unit

Control units can either be *microcoded* or *hardwired*.

The primary advantages of microcoded controller are as follows.

- A standard design can be developed for series of devices. The only difference is in the values of control signals that are stored in the ROM.
- Design changes are easier to accommodate.
- Design time and design cost is greatly reduced because the hardware is already designed and debugged.

The primary disadvantages are:

- Slower operation when compared to hardwired design because the read time for the ROM must be accommodated.
- The microcoded design will also be more costly for small devices because it includes a ROM and two registers at a minimum.
- Using a standard controller limits the designer to the use of features built into the controller. More features implies higher cost. Everyone must pay the higher cost, even if they do not use all of the features of the controller. The controller design is a trade-off between flexibility and cost.