ECE 545
Lecture 12

Design of Controllers

Finite State Machines and Algorithmic State Machine (ASM) Charts
Required reading

- P. Chu, *RTL Hardware Design using VHDL*

  Chapter 10, *Finite State Machine: Principle & Practice*

  Chapter 11, *Register Transfer Methodology: Principle*

  Chapter 12, *Register Transfer Methodology: Practice*
Slides based partially on

- S. Brown and Z. Vranesic,
  *Fundamentals of Digital Logic with VHDL Design*
  
  *Chapter 8, Synchronous Sequential Circuits*
  
  *Sections 8.1-8.5*

*Chapter 8.10, Algorithmic State Machine (ASM) Charts*

*Chapter 10.2 Design Examples*
Datapath vs. Controller
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)
Datapath (Execution Unit)

- Manipulates and processes data
- Performs arithmetic and logic operations, shifting/rotating, and other data-processing tasks
- Is composed of registers, multiplexers, adders, decoders, comparators, ALUs, gates, etc.
- Provides all necessary resources and interconnects among them to perform specified task
- Interprets control signals from the Controller and generates status signals for the Controller
Controller (Control Unit)

- Controls data movements in the Datapath by switching multiplexers and enabling or disabling resources
  - Example: enable signals for registers
  - Example: control signals for muxes
- Provides signals to activate various processing tasks in the Datapath
- Determines the sequence of operations performed by the Datapath
- Follows Some ‘Program’ or Schedule
Programmable vs. Non-Programmable Controller

- Controller can be programmable or non-programmable
  - Programmable
    - Has a program counter which points to next instruction
    - Instructions are held in a RAM or ROM
    - Microprocessor is an example of programmable controller
  - Non-Programmable
    - Once designed, implements the same functionality
    - Another term is a “hardwired state machine,” or “hardwired FSM,” or “hardwired instructions”
    - In this course we will be focusing on non-programmable controllers.
Finite State Machines

- Digital Systems and especially their Controllers can be described as Finite State Machines (FSMs)
- Finite State Machines can be represented using
  - **State Diagrams and State Tables** - suitable for simple digital systems with a relatively few inputs and outputs
  - **Algorithmic State Machine (ASM) Charts** - suitable for complex digital systems with a large number of inputs and outputs
- All these descriptions can be easily translated to the corresponding synthesizable VHDL code
Hardware Design with RTL VHDL

- Pseudocode
- Interface
- Datapath
  - Block diagram
  - VHDL code
- 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
Finite State Machines Refresher
Finite State Machines (FSMs)

• Any Circuit with Memory Is a Finite State Machine
  • Even computers can be viewed as huge FSMs
• Design of FSMs Involves
  • Defining states
  • Defining transitions between states
  • Optimization / minimization
• Manual Optimization/Minimization Is Practical for Small FSMs Only
Moore FSM

Output Is a Function of a Present State Only

Inputs

Next State function

Next State

Present State

clock
reset

Present State register

Output function

Outputs
Mealy FSM

- Output Is a Function of a Present State and Inputs

![Diagram of Mealy FSM]

- Inputs
- Next State function
- Present State register
- Output function
- Outputs

- Present State
- Next State
- clock
- reset
- Inputs
State Diagrams
Moore Machine

- **State 1 / Output 1**
  - Transition condition 1

- **State 2 / Output 2**
  - Transition condition 2
Mealy Machine

state 1

transition condition 1 / output 1

state 2

transition condition 2 / output 2
Moore vs. Mealy FSM (1)

• Moore and Mealy FSMs Can Be Functionally Equivalent
  • Equivalent Mealy FSM can be derived from Moore FSM and vice versa
• Mealy FSM Has Richer Description and Usually Requires Smaller Number of States
  • Smaller circuit area
Moore vs. Mealy FSM (2)

- Mealy FSM Computes Outputs as soon as Inputs Change
  - Mealy FSM responds one clock cycle sooner than equivalent Moore FSM
- Moore FSM Has No Combinational Path Between Inputs and Outputs
  - Moore FSM is less likely to affect the critical path of the entire circuit
Moore FSM - Example 1

• Moore FSM that Recognizes Sequence “10”

Meaning of states:

S0: No elements of the sequence observed
S1: “1” observed
S2: “10” observed
Mealy FSM - Example 1

• Mealy FSM that Recognizes Sequence “10”

Meaning of states:

S0: No elements of the sequence observed
S1: “1” observed

0 / 0
1 / 0
1 / 0
reset

0 / 1
Moore & Mealy FSMs – Example 1

Clock timeline:
- 0
- 1
- 0
- 0
- 0

Input timeline:
- 0
- 1
- 0
- 0
- 0

Moore FSM:
- S0
- S1
- S2
- S0
- S0

Mealy FSM:
- S0
- S1
- S0
- S0
- S0
Finite State Machines in VHDL
FSMs in VHDL

• Finite State Machines Can Be Easily Described With Processes
• Synthesis Tools Understand FSM Description if Certain Rules Are Followed
  • State transitions should be described in a process sensitive to clock and asynchronous reset signals only
  • Output function described using rules for combinational logic, i.e. as concurrent statements or a process with all inputs in the sensitivity list
Moore FSM

process(clock, reset)

Inputs

Next State function

clock
reset

Next State

Present State Register

concurrent statements

Output function

Present State

Outputs
Mealy FSM

process(clock, reset)

- INPUTS
  - clock
  - reset

- PRESENT STATE REGISTER

- NEXT STATE

- OUTPUT

- CONCURRENT STATEMENTS
Moore FSM - Example 1

- Moore FSM that Recognizes Sequence “10”
Moore FSM in VHDL (1)

TYPE state IS (S0, S1, S2);
SIGNAL Moore_state: state;

U_Moore: PROCESS (clock, reset)
BEGIN
  IF(reset = '1') THEN
    Moore_state <= S0;
  ELSIF (clock = '1' AND clock'event) THEN
    CASE Moore_state IS
      WHEN S0 =>
        IF input = '1' THEN
          Moore_state <= S1;
        ELSE
          Moore_state <= S0;
        END IF;
      WHEN OTHERS =>
        Moore_state <= S0;
    END CASE;
  END IF;
END;
WHEN S1 =>
    IF input = '0' THEN
        Moore_state <= S2;
    ELSE
        Moore_state <= S1;
    END IF;
WHEN S2 =>
    IF input = '0' THEN
        Moore_state <= S0;
    ELSE
        Moore_state <= S1;
    END IF;
END CASE;
END IF;
END PROCESS;

Output <= '1' WHEN Moore_state = S2 ELSE '0';
Mealy FSM - Example 1

- Mealy FSM that Recognizes Sequence “10”
TYPE state IS (S0, S1);
SIGNAL Mealy_state: state;

U_Mealy: PROCESS(clock, reset)
BEGIN
    IF(reset = '1') THEN
        Mealy_state <= S0;
    ELSIF (clock = '1' AND clock'event) THEN
        CASE Mealy_state IS
            WHEN S0 =>
                IF input = '1' THEN
                    Mealy_state <= S1;
                ELSE
                    Mealy_state <= S0;
                END IF;
        END CASE;
    END IF;
END;
Mealy FSM in VHDL (2)

WHEN S1 =>
  IF input = '0' THEN
    Mealy_state <= S0;
  ELSE
    Mealy_state <= S1;
  END IF;
END CASE;
END IF;
END PROCESS;

Output <= '1' WHEN (Mealy_state = S1 AND input = '0') ELSE '0';
Algorithmic State Machine (ASM) Charts
Algorithmic State Machine

Algorithmic State Machine – representation of a Finite State Machine suitable for FSMs with a larger number of inputs and outputs compared to FSMs expressed using state diagrams and state tables.
Elements used in ASM charts (1)

- State name
- Output signals or actions (Moore type)
- Conditional outputs or actions (Mealy type)
- Condition expression
  - 0 (False)
  - 1 (True)

(a) State box
(b) Decision box
(c) Conditional output box
State Box

- **State box** – represents a state.
- Equivalent to a node in a state diagram or a row in a state table.
- Contains register transfer actions or output signals
- **Moore-type outputs are listed inside of the box.**
- It is customary to write only the name of the signal that has to be asserted in the given state, e.g., z instead of z<=1.
- Also, it might be useful to write an action to be taken, e.g., count <= count + 1, and only later translate it to asserting a control signal that causes a given action to take place (e.g., enable signal of a counter).
Decision Box

- **Decision box** – indicates that a given condition is to be tested and the exit path is to be chosen accordingly. The condition expression may include one or more inputs to the FSM.
Conditional Output Box

- **Conditional output box**
- Denotes output signals that are of the Mealy type.
- The condition that determines whether such outputs are generated is specified in the decision box.
ASMs representing simple FSMs

- Algorithmic state machines can model both Mealy and Moore Finite State Machines
- They can also model machines that are of the mixed type
Moore FSM – Example 2: State diagram

![State Diagram]

States:
- A/z = 0
- B/z = 0
- C/z = 1

Transitions:
- Reset
- w = 0
- w = 1
Moore FSM – Example 2: State table

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state ( w = 0 )</th>
<th>Next state ( w = 1 )</th>
<th>Output ( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>
ASM Chart for Moore FSM – Example 2
Example 2: VHDL code (1)

USE ieee.std_logic_1164.all;

ENTITY simple IS
  PORT (     clock   : IN STD_LOGIC;
             resetn  : IN STD_LOGIC;
             w          : IN STD_LOGIC;
             z          : OUT STD_LOGIC );
END simple;

ARCHITECTURE Behavior OF simple IS
  TYPE State_type IS (A, B, C);
  SIGNAL y : State_type;
BEGIN
  PROCESS ( resetn, clock )
  BEGIN
    IF resetn = '0' THEN
      y <= A;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      Begin
CASE y IS
  WHEN A =>
    IF w = '0' THEN
      y <= A ;
    ELSE
      y <= B ;
    END IF ;
  WHEN B =>
    IF w = '0' THEN
      y <= A ;
    ELSE
      y <= C ;
    END IF ;
  WHEN C =>
    IF w = '0' THEN
      y <= A ;
    ELSE
      y <= C ;
    END IF ;
END CASE ;
Example 2: VHDL code (3)

END IF ;
END PROCESS ;

z <= '1' WHEN y = C ELSE '0' ;

END Behavior ;
Mealy FSM – Example 3: State diagram
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Mealy IS
    PORT ( clock : IN STD_LOGIC;
           resetn : IN STD_LOGIC;
           w : IN STD_LOGIC;
           z : OUT STD_LOGIC );
END Mealy;

ARCHITECTURE Behavior OF Mealy IS
    TYPE State_type IS (A, B);
    SIGNAL y : State_type;
BEGIN
    PROCESS ( resetn, clock )
    BEGIN
        IF resetn = '0' THEN
            y <= A;
        ELSIF (clock'EVENT AND clock = '1') THEN
            Example 3: VHDL code (1)
CASE y IS
  WHEN A =>
    IF w = '0' THEN
      y <= A ;
    ELSE
      y <= B ;
    END IF ;
  WHEN B =>
    IF w = '0' THEN
      y <= A ;
    ELSE
      y <= B ;
    END IF ;
END CASE ;
Example 3: VHDL code (3)

    END IF ;
    END PROCESS ;

    z <= '1' WHEN (y = B) AND (w='1') ELSE '0' ;

    END Behavior ;
Control Unit Example: Arbiter (1)
Control Unit Example: Arbiter (2)
Control Unit Example: Arbiter (3)
ASM Chart for Control Unit - Example 4
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY arbiter IS
  PORT ( Clock, Resetn : IN STD_LOGIC;
        r : IN STD_LOGIC_VECTOR(1 TO 3);
        g : OUT STD_LOGIC_VECTOR(1 TO 3) );
END arbiter ;

ARCHITECTURE Behavior OF arbiter IS
  TYPE State_type IS (Idle, gnt1, gnt2, gnt3);
  SIGNAL y : State_type ;

BEGIN
  PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN y <= Idle ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN Idle =>
          IF r(1) = '1' THEN y <= gnt1 ;
          ELSIF r(2) = '1' THEN y <= gnt2 ;
          ELSIF r(3) = '1' THEN y <= gnt3 ;
          ELSE y <= Idle ;
          END IF ;
        WHEN gnt1 =>
          IF r(1) = '1' THEN y <= gnt1 ;
          ELSE y <= Idle ;
          END IF ;
        WHEN gnt2 =>
          IF r(2) = '1' THEN y <= gnt2 ;
          ELSE y <= Idle ;
          END IF ;
Example 4: VHDL code (3)

WHEN gnt3 =>
  IF r(3) = '1' THEN y <= gnt3 ;
  ELSE y <= Idle ;
  END IF ;
END CASE ;
END IF ;
END PROCESS ;

  g(1) <= '1' WHEN y = gnt1 ELSE '0' ;
g(2) <= '1' WHEN y = gnt2 ELSE '0' ;
g(3) <= '1' WHEN y = gnt3 ELSE '0' ;
END Behavior ;
1. Overview on FSM

- Contain “random” logic in next-state logic
- Used mainly used as a controller in a large system
- Mealy vs Moore output
2. Representation of FSM

- State diagram

Figure 10.2 Notation for a state.
• E.g. a memory controller
• ASM (algorithmic state machine) chart
  – Flowchart-like diagram
  – Provide the same info as an FSM
  – More descriptive, better for complex description
  – ASM block
    • One state box
    • One or more optional decision boxes: with T or F exit path
    • One or more conditional output boxes: for Mealy output
Figure 10.4  ASM block.
State diagram and ASM chart conversion

• E.g. 1.
E.g. 2.
• E.g. 3.
• E.g. 4.
- E.g. 6.

![Diagram](image-url)
• Difference between a regular flowchart and ASM chart:
  – Transition governed by clock
  – Transition done between ASM blocks

• Basic rules:
  – For a given input combination, there is one unique exit path from the current ASM block
  – The exit path of an ASM block must always lead to a state box. The state box can be the state box of the current ASM block or a state box of another ASM block.
• Incorrect ASM charts:
RTL Hardware Desy P. Chu
4. Moore vs Mealy output

• Moore machine:
  – output is a function of state

• Mealy machine:
  – output function of state and output

• From theoretical point of view
  – Both machines have similar “computation capability”

• Implication of FSM as a controller?
• E.g., edge detection circuit
  – A circuit to detect the rising edge of a slow “strobe” input and generate a “short” (about 1-clock period) output pulse.
• Three designs:
• Comparison
  – Mealy machine uses fewer states
  – Mealy machine responds faster
  – Mealy machine may be transparent to glitches
• Which one is better?
• Types of control signal
  – Edge sensitive
    • E.g., enable signal of counter
    • Both can be used but Mealy is faster
  – Level sensitive
    • E.g., write enable signal of SRAM
    • Moore is preferred
VHDL Description of FSM

• Follow the basic block diagram
• Code the next-state/output logic according to the state diagram/ASM chart
• Use enumerate data type for states
• E.g. 6.
library ieee;
use ieee.std_logic_1164.all;
entity mem_ctrl is
port ( 
    clk, reset: in std_logic;
    mem, rw, burst: in std_logic;
    oe, we, we_me: out std_logic);
end mem_ctrl ;

architecture mult_seg_arch of mem_ctrl is
    type mc_state_type is
        (idle, read1, read2, read3, read4, write);
    signal state_reg, state_next: mc_state_type;
begin
    -- state register
    process(clk, reset)
    begin
        if (reset='1') then
            state_reg <= idle;
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
        end if;
    end process;

-- next-state logic
process(state_reg, mem, rw, burst)
begin
  case state_reg is
    when idle =>
      if mem='1' then
        if rw='1' then
          state_next <= read1;
        else
          state_next <= write;
        end if;
      else
        state_next <= idle;
      end if;
    when write =>
      state_next <= idle;
  end case;
end process;
when read1 =>
  if (burst='1') then
    state_next <= read2;
  else
    state_next <= idle;
  end if;
when read2 =>
  state_next <= read3;
when read3 =>
  state_next <= read4;
when read4 =>
  state_next <= idle;
end case;
end process;
— moore output logic

process(state_reg)
begin
  we <= '0';  -- default value
  oe <= '0';  -- default value
  case state_reg is
    when idle =>
      we <= '1';
    when write =>
      oe <= '1';
    when read1 =>
      oe <= '1';
    when read2 =>
      oe <= '1';
    when read3 =>
      oe <= '1';
    when read4 =>
      oe <= '1';
  end case;
end process;
— *mealy output logic*

```vhdl
process(state_reg, mem, rw)
begin
  we_me <= '0';  -- default value
  case state_reg is
    when idle =>
      if (mem='1') and (rw='0') then
        we_me <= '1';
      end if;
    when write =>
    when read1 =>
    when read2 =>
    when read3 =>
    when read4 =>
  end case;
end process;
end mult_seg_arch;
```

`we_me <= '1' when ((state_reg=idle) and (mem='1') and (rw='0')) else '0';`
• Combine next-state/output logic together
process(state_reg, mem, rw, burst) begin
  oe <= '0';    -- default values
  we <= '0';
  we_me <= '0';
  case state_reg is
    when idle =>
      if mem='1' then
        if rw='1' then
          state_next <= read1;
        else
          state_next <= write;
        end if;
      else
        state_next <= idle;
      end if;
    when write =>
      state_next <= idle;
  we <= '1';
  end case;
when read1 =>
    if (burst='1') then
        state_next <= read2;
    else
        state_next <= idle;
    end if;
    oe <= '1';
when read2 =>
    state_next <= read3;
    oe <= '1';
when read3 =>
    state_next <= read4;
    oe <= '1';
when read4 =>
    state_next <= idle;
    oe <= '1';
end case;
end process;
sorting example
Sorting - Required Interface

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

Sort

DataOut
Done

N
L
N
## 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

#### Before Sorting

<table>
<thead>
<tr>
<th>Address</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=0 j=1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>i=0 j=2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>i=0 j=3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>i=1 j=2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>i=1 j=3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>i=2 j=3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

#### During Sorting

- **Legend:**
  - Position of memory indexed by i: $M_i$
  - Position of memory indexed by j: $M_j$

<table>
<thead>
<tr>
<th>Address</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=0 j=1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>i=0 j=2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>i=0 j=3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>i=1 j=2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>i=1 j=3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>i=2 j=3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

#### After Sorting

<table>
<thead>
<tr>
<th>Address</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=0 j=1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>i=0 j=2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>i=0 j=3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>i=1 j=2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>i=1 j=3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>i=2 j=3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
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

Datapath

Controller

DataIn  RAddr  WrInit  Rd  Clock  Resetn  s

N  L

Datapath

Controller

DataOut

Done

AgtB
zi
zj

Wr
Li
Ei
Lj
Ej
EA
EB
Bout
Csel