The **Microelectronics Training Center**

The MTC is an initiative within the INVOMEC division

Industrialization & Training in Microelectronics

Lab-exercise

Lab 4:

Design of the memory traffic controller

Cluster: Cluster1
Module: Module3b

Target group: Students

Version: 1.0
Date: 14/12/06
Author: Osman Allam
Modified by: Geert Vanwijnsberghe
History : testbench added

This material was developed with support of the European Social Fund.
ESF: Prevent and combat unemployment by promoting employability, entrepreneurship, adaptability and equal opportunities between women and men, and by investment in people.
http://www.esf-agentschap.be

Introduction

The Memory Traffic Controller arbitrates memory access between 3 logical units: the data path, the fetch unit and the I/O unit. Units wishing to access the memory raise requests. According to an arbitration scheme, the memory traffic controller grants access to a unit for a single memory cycle.

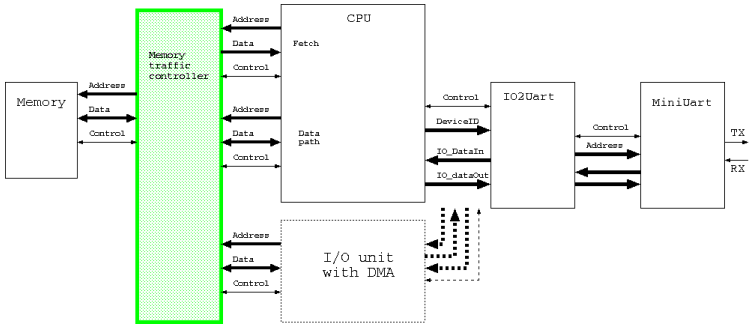


Figure 1: System architecture

Objectives

After completing this lab, you will be able to:

- Identify different models of Finite State Machines (FSM);
- Design Finite State Machines (FSM) in VHDL;
- Understand the difference between FSM models from a synthesis point of view.

Knowledge background

- Basic VHDL knowledge
- Understanding of the concepts of finite state machines

Classification

- Level: 3
- Duration: 1 hour

Input

VHDL template and testbench.

The lab

There are two types of Finite State Machines (FSMs):

1. **Moore Machine**
   - Outputs are determined only by the current state
   - Inputs of the machine in a given state determine the next state
   - Outputs are associated with the states

2. **Mealy Machine**
   - Outputs are determined by the current state and the inputs
   - Inputs of the machine in a given state determine the next state
   - Outputs are associated with state transitions (arcs, in the state transition graph)

Describing FSMs in VHDL

Describing FSMs in VHDL is basically answering the following questions:
1. What states can the machine reach?
2. How does the machine transit from one state to another?
3. Which output does the machine create in the different states?

The first question is answered by defining the states. It is recommended to declare an enumerated type for the states.

```
Type states is {state1, state2, state3, state4};
```

The second question is answered by the **next state decoding logic**. This is a combinational logic block accepting the current state and inputs of the machine as its inputs and returning the next state.

The third question is answered by the **output decoding logic**. This is also a combinational logic block accepting the current state, and probably the inputs of the machine as its inputs and returning the state machine outputs.

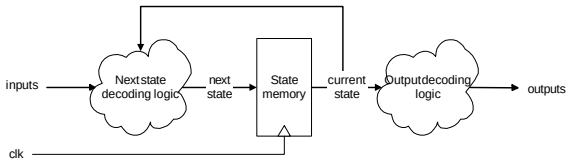
The state machine needs to *remember* its state. Usually this is done in the **state memory** where the current state is registered every clock cycle.

In some applications, it is required to register the outputs of the state machine in the **output register**, to avoid glitches and to achieve other implementation goals. However, registering the outputs delays the response of the state machine by one clock cycle.

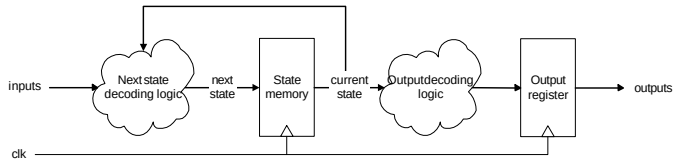
In summary, we have 4 blocks:
1. Next state decoding logic;
2. Output decoding logic;
3. State memory;
4. Possibly an output register.

Rearranging these blocks results in different models of state machines as illustrated below:

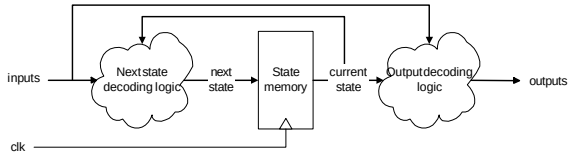
1. **Moore Machine** (used here)



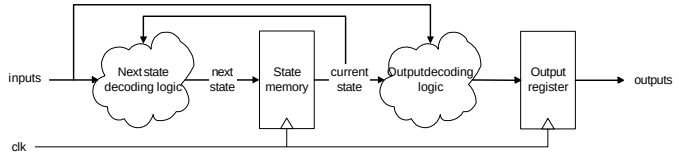
2. **Moore Machine with registered-outputs**



3. **Mealy Machine**



4. **Mealy Machine with registered-outputs**



Exercise

The memory traffic controller arbitrates memory access between 3 master units: the data path (CPU), the fetch unit (CPU) and the I/O unit. It is basically a 3-port multiplexer controlled by an FSM. For simplicity, you can view the memory traffic controller as the combination of the FSM and the multiplexer in one sequential circuit.

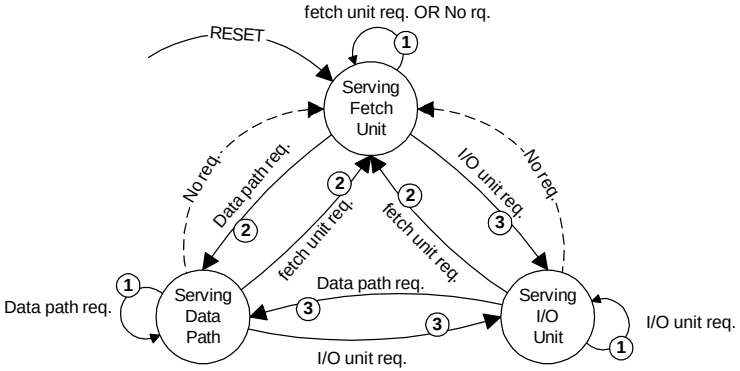
The competing units are assumed to keep their request lines active as long as they wish to access the memory. When the memory traffic controller is not busy, it checks which request lines are active. According to an arbitration scheme, only one unit gains control of the memory for as long as a memory operation would take. The memory traffic controller hooks in a state where it serves one unit and in this state other request lines are ignored. Such state is terminated by receiving the ready signal from the memory and propagating it to the controlling unit.

Note that a “grant” signal is not required. The controlling units will effectively halt until their requested memory operation is completed.

The arbitration scheme to be adopted by your design gives higher priority to the fetch unit requests, and then comes the data path and then the I/O unit. Note, however that requests from the currently controlling unit have the highest priority. This ensures that the active unit will keep accessing the memory until its requested operation(s) is (are) complete.

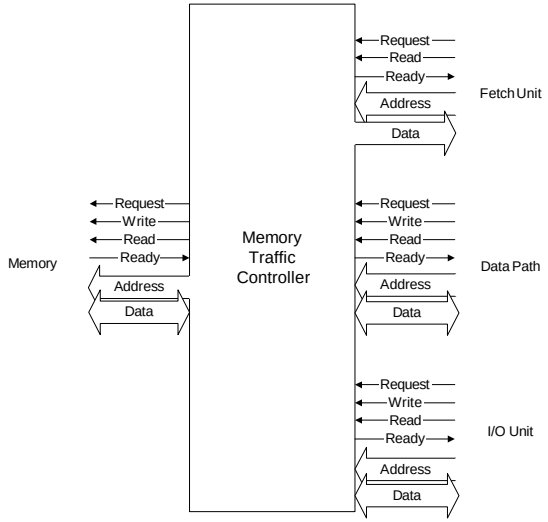
In case no units are requesting memory access, the memory traffic controller transits to the state where it serves the fetch unit. This is because the fetch unit accesses the memory more frequently than the 2 other units (at least 1 access per program instruction).

In a given state, the memory traffic controller connects the controlling unit to the memory. The "ready" signal is propagated to the controlling unit. Other units receive inactive "ready" signal from the memory traffic controller, which keeps their requests pending.



Use the template provided in the file `memCtrl.vhd`.

Note: The outputs of the FSM are not shown in the above diagram. However, since the memory traffic controller acts as a multiplexer, the outputs are assumed to be associated with the states. Hence, this is a Moore machine.



Use the testbench in tb_memCtrl.vhd to verify the controller. Remark that the testbench does not verify all state transitions.