"FPGA-Programming" Exercise Sheet XII:

Implementing Algorithms in Hardware

Goal of this exercise is to learn how to use algorithmic state machine charts to implement algorithms as hardware circuit.

As target hardware always choose FPGA chip Cyclone V SoC 5CSEMA5F31C6

Algorithmic State Machine (ASM) charts are a design tool that allow the specification of digital systems in a form similar to a **flow chart**. An example of an ASM chart is shown infollowing figure.

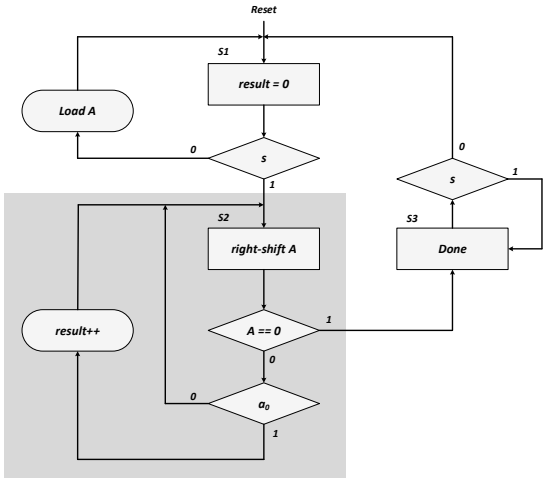

Figure 0.1: ASM chart for a bit counting circuit

This ASM chart represents a circuit that counts the number of bits set to 1 in an n-bit input A ($A = a_{n-1}a_{n-2}\ldots a_1a_0$). The rectangular boxes in this diagram represent the states of the digital system, and actions specified inside of a state box occur on each active clock edge in this state. Transitions between states are specified by arrows. The diamonds in the ASM chart represent conditional tests, and the ovals represent actions taken only if the corresponding conditions are either true (on an arrow labeled 1) or false (on an arrow labeled 0).

In this ASM chart, state $\mathbf{S1}$ is the initial state. In this state the result is initialized to 0, and data is loaded into a register $\mathbf{A}$, until a start signal, $\mathbf{s}$, is asserted. The ASM chart then transitions to state $\mathbf{S2}$, where it increments the **result** to count the number of 1-bits in register $\mathbf{A}$. Since state $\mathbf{S2}$ specifies a shifting operation, then $\mathbf{A}$ should be implemented as a shift register. Also, since the result is incremented, then this variable should be implemented as a counter. When register $\mathbf{A}$ contains 0 the ASM chart transitions to state $\mathbf{S3}$, where it sets an output **Done = 1** and waits for the signal $\mathbf{s}$ to be deasserted.

A key distinction between ASM charts and flow charts is a concept known as **implied timing**. The implied timing specifies that all actions associated with a given state take place only when the system is in that state when an active clock edge occurs. For example, when the system is in state $\mathbf{S1}$ and the start signal $\mathbf{s}$ becomes 1, then the next active clock edge performs the

following actions: initializes result to 0, and transitions to state $\mathbf{S2}$. The action right-shift $\mathbf{A}$ does not happen yet, because the system is not yet in state $\mathbf{S2}$. For each active clock cycle in state $\mathbf{S2}$, the actions highlighted above figure take place, as follows: increment result if bit $\mathbf{a_0} = \mathbf{1}$, change to state $\mathbf{S3}$ if $\mathbf{A} = \mathbf{0}$ (or else remain in state $\mathbf{S2}$), and shift $\mathbf{A}$ to the right.

The implementation of the bit counting circuit includes the counter to store the **result** and the shift register $\mathbf{A}$, as well as a finite state machine. The FSM is often referred to as the **control circuit**, and the other components as the **datapath circuit**.

1) *Logic High Bit Counting Circuit*

   In this part the high bit counting circuit described in above ASM chart should be implemented.

   Perform following steps:

   i) Create a new Quartus project for the circuit, named **e_my_bitcounter**.

   ii) First describe the circuit of a 7-segment decoder named **e_hex7seg** which is able to decode a 4-bit binary number to drive a 7-segment display with hexadecimal notation. Use a concurrent *conditional signal assignment*.

   iii) Describe the circuit of a D-type flip-flop (**e_flipflop**). The circuit should exhibit a *synchronous low-active* reset.

   iv) Describe the top-level entity (**e_my_bitcounter**). To the VHDL code add the **datapath circuit** by using a *sequential process* and implement the finite state machine for the **control circuit**. Use switches $SW[7-0]$ as 8-bit data input and push-button $KEY[0]$ as reset input. Use switch $SW[9]$ for the start signal s. Use the 50 MHz input clock. Ensure that the start signal s will be synchronised to the clock. Therefore use two instances of **e_flipflop**. Display the amount of counted logic high bits on the 7-segment display $HEX0$. $LEDR[9]$ should be used to signal that the execution of the algorithm has been completed.

   v) Apply proper timing constraints on the input clock and create a timing constraints file (**e_my_bitcounter.sdc**).

   vi) Create a (self-checking) test-bench (**e_my_bitcounter.vht**) for the circuit and verify the functional correctness of the circuit with simulation.

   vii) Include a Signal Tap II file (**e_my_bitcounter.stp**) to the project, add relevant nodes and set proper trigger conditions to investigate signal transitions once implemented in hardware.

   viii) Import the necessary pin assignments, compile and download the circuit. Test the correct functionality in hardware.

2) *Bit Pattern searching circuit*

In this part a circuit should be designed, which searches through an array to locate an 8-bit pattern $\mathbf{A}$. The 8-bit value $\mathbf{A}$ should be specified via switches $SW[7-0]$. Following figure shows a block diagram of the circuit.

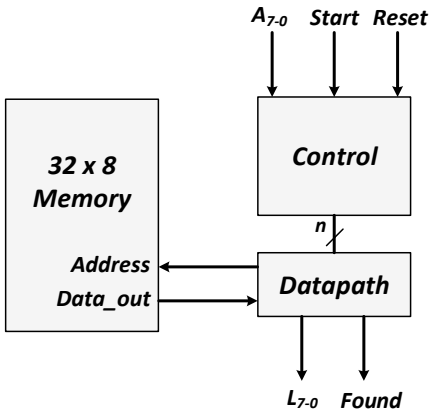

Figure 2.1: A block diagram for a circuit that performs a binary search

The binary search algorithm works on a sorted array. Rather than comparing each value in the array to the one being sought, we first look at the middle element and compare the sought value to the middle element. If the middle element has a greater value, then we know that the element we seek must be in the first half of the array. Otherwise, the value we seek must be in the other half of the array. By applying this approach recursively, we can locate the sought element in only a few steps.

In this circuit, the array is stored in an on-chip memory created with the Quartus **IP Catalog**. Use the **RAM: 1-PORT** LPM module (**Basic Functions** $\rightarrow$ **On Chip Memory**). Create an IP variation named **e_memory_block.vhd** with an **aspect ratio** of $\mathbf{32 \times 8}$. The output of the memory should not be registered. Furthermore a **MIF** file named **my_array.mif** should be generated in which an ordered array of numbers can be specified.

The circuit should produce a 5-bit output $\mathbf{L}$, which yields the address of the memory which holds the bit pattern $\mathbf{A}$ searched for. Furthermore a **Found** signal should be used to signalize that the number $\mathbf{A}$ has been found in memory.

Perform following steps:

i) Create the ASM diagram for the bit pattern searching circuit. Consider that the input signals of the memory will be registered. Assume that the array has a fixed size of 32 elements.

ii) Create a new Quartus project for the circuit, named **e_my_patternsearch**.

iii) Include the circuit of the 7-segment decoder **e_hex7seg** of part 1).

iv) Include the circuit of the D-type flip-flop **e_flipflop** of part 1).

v) For the description of the datapath create a own VHDL entity named **e_datapath**. Describe the behaviour.

vi) For the control unit code a own VHDL entity named **e_FSM**. Describe the finite state machine.

vii)) Create a RAM memory named **e_memory_block**.

viii) Create a file named **my_array.mif** and fill it with a sorted amount of 32 8-bit numbers.

ix) Describe the top-level entity (**e_my_patternsearch**). Use switch $SW[9]$ as **Start** input, switches $SW[7-0]$ as input of value **A**. Use push-button $KEY[0]$ as a low-active reset and the $50MHz$ clock signal as input clock. Ensure that the **Start** signal will be synchronised to the clock. Therefore use two instances of **e_flipflop**. Display the memory address of the number **A**, if it has been found in memory on the 7-segment displays $HEX1$ and $HEX0$ in hexadecimal notation. Use $LEDR[9]$ for the **Found** signal.

x) Import the necessary pin assignments, compile and download the circuit. Test the correct functionality in hardware.