

## Microprogram Control Unit Design: Merging Two Arrays

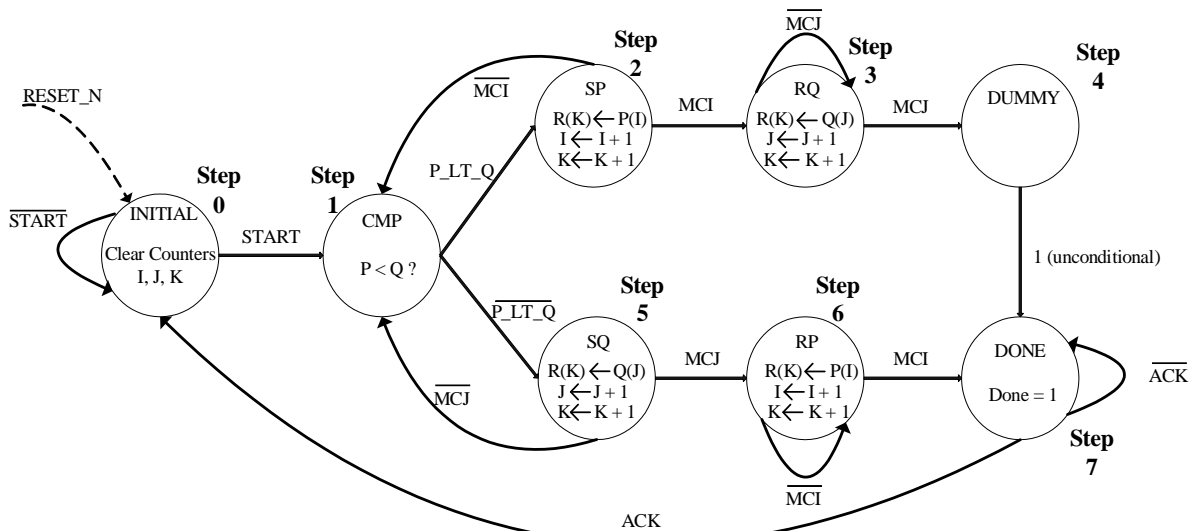
### 1. Synopsis:

The purpose of this lab is to implement a state machine by using a microprogram control unit design. Microprogramming allows flexibility in the implementation of a state machine because the next state logic and the output function logic are stored as bits in a ROM rather than hard-wired as logic gates. For this exercise, a complete data path unit is provided along with a completed “top” design. You are required to arrive at the correct microprogram control unit and a simple test bench to test it.

### 2. Merging Algorithm:

The goal of the small system that will be implemented in this exercise is to merge two 4-element arrays, P and Q, each containing *sorted* 4-bit numbers in ascending order into one 8-element *sorted* array R. This is a step in the well-known merge-sort algorithm. See the animation at [http://en.wikipedia.org/wiki/Merge\\_sort](http://en.wikipedia.org/wiki/Merge_sort).

To perform this merging the following algorithm is implemented: Compare an element of array P with an element of array Q. Select the smaller of the two to store in array R. Advance either the pointer I (which points to the next element of P to be compared) or J (which points to the next element of Q to be compared) and also the pointer K of array R. When all elements of one of the two arrays (P or Q) are stored in R, then simply transfer the remaining elements of the other array (Q or P) to array R. Note that there is no need for any more comparisons in this transfer step as the original arrays P & Q were sorted. A simple Moore state machine that performs this function is given below.



The state machine starts in the INITIAL state where all three pointers, I, J and K (which are implemented as counters) are cleared. As the user presses the START button, the state machine goes to COMPARE (CMP) state. In this state, the comparator compares the appropriate elements in array P and array Q pointed by the I and J pointers (counters) respectively. If the element in array P is smaller than the one in array Q, the state machine goes to STORE\_P (SP) state in which the element in array P is stored into array R and both the pointers of array P and array R are advanced. If pointer I reaches the bottom of the array P, then the array P has been stored in R completely and so the state machine steps into REMAINING\_Q (RQ) state where all remaining elements of Q are transferred into R. For the case where the element is Q is smaller than the element of P being compared, the parallel sequences of STORE\_Q and REMAINING\_P will be followed. Once the merging is complete, the state machine enters DONE state where it waits for the ACK signal from the user to go high. The 8 states in the state diagram are explained below:

State	Explanation
INITIAL	Clear pointers (Counters I, J and K)
CMP	Compare element of P with element of Q
SP (STORE_P)	Select the smaller P to store in R. Increment I and K.
SQ (STORE_Q)	Select the smaller Q to store in R. Increment J and K.
RP (REMAINING_P)	Transfer remaining elements of P into R. Increment I and K.
RQ (REMAINING_Q)	Transfer remaining elements of Q into R. Increment J and K.
DUMMY & DONE	“Done” signal goes active in the Done State.

### 3. Microprogrammed Control Unit:

In previous labs, we implemented the control units in our systems using either one-hot coded states or binary-coded state machines where the next state logic and output function logic were hard-wired using logic gates. In this lab, we implement the control unit using a different technique called the microprogrammed control unit (MCU) design. Unlike the state machine implementations seen in previous labs, a MCU consists of three components: control memory, microprogram counter, and a condition select multiplexer. The control memory is actually a ROM (Read Only Memory). The data in each line of the ROM is called a *microinstruction*. The depth of this memory (the number of locations in the memory) is determined by the number of states of the state machine. The width of this memory (the number of bits per location) is determined by the number of control signals to be produced to control the data path unit (and the branch address field and the condition select field) . The microprogram counter acts as the pointer to the control memory. In order to realize branching, a "load" control is provided for the microprogram counter so that it can be loaded with the branch address if the branch condition is met. Note that the branch address and condition select fields are also incorporated into each microinstruction.

The typical microinstruction format is:

Condition Select	Branch Address	Control Signals
------------------	----------------	-----------------

## 4. Hardware Implementation

In this lab, the top-level design interfacing with the Buttons/Switches/Leds/SSDs is given to you in Verilog in completed form. This design instantiates your schematic core design. The core design is a 2-page schematic. The second page is the datapath and it is nearly complete (90% complete). It consists of the three arrays, P, Q, and R and also the three counters I, J and K acting as pointers into the arrays. You need to decide what are all the control signals in this data path, give suitable names to these control signals, and label those nets. The first page (of the 2-page schematic) is nearly blank for you to design the micro-programmed control unit and generate the control signals needed in the 2nd page. The major components on the first page are (a) the control memory (b) the microprogram counter (c) the condition select mux.

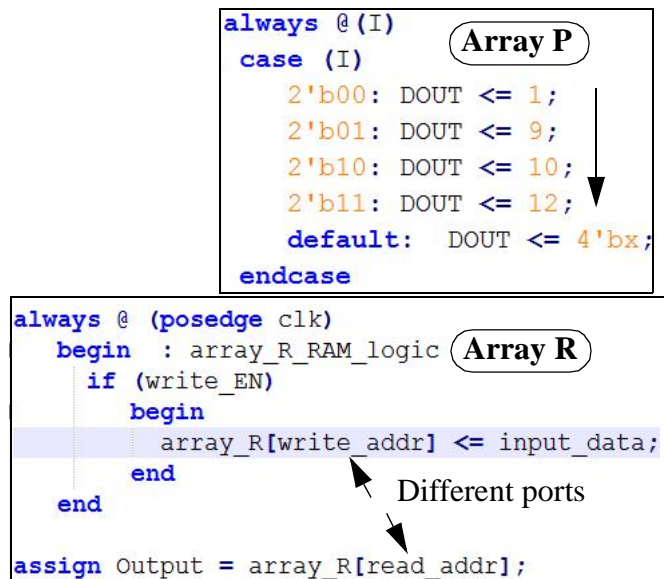
There are 8 major components on the 2nd page of the schematic. Please see its printout included in this handout towards the end. For the three array and the 4-bit wide 2-to-1 mux, we have provided you component symbols (array\_P\_ROM.sch, array\_Q\_ROM.sch, array\_R\_RAM.sch, mux\_2\_to\_1.sch) and underlying verilog files (array\_P\_ROM.v, array\_Q\_ROM.v, array\_R\_RAM.v, mux\_2\_to\_1.v). For the three counters I, J, and K, we have used the Xilinx schematic components **CB2RE**, **CB2RE**, and **CB4RE** (Cascadable Counter Binary 4-bit with Synchronous Reset and Enable). You can look up the data sheet (Help => Software manuals => Libraries Guides => Spartan-6 Libraries Guide for Schematic Designs => Chapter 3 => **CB4RE**)

### CB4RE

**Macro: 4-Bit Cascadable Binary Counter with Clock Enable and Synchronous Reset**

For the 4-bit comparator to compare P[I] with Q[J], we used the Xilinx schematic component **COMPM4**.

Arrays P and Q are implemented as read-only memories. Array R cannot be declared as a read-only memory (using the format that was used for arrays P and Q) because data has to be written to it. Therefore, it is declared as an array of registers. Furthermore, array R has a write strobe which must be asserted along with the address to which data has to be written. If the write strobe, write address and write data are valid before the clock edge, the input data is stored into the array *at the clock edge*. Counter K serves as the write address pointer for array R. To facilitate inspection of the final data in the R array, we have provided an independent read port (with read address driven by the three switches Sw2, Sw1, Sw0).



```

always @(I)
case (I)
    2'b00: DOUT <= 1;
    2'b01: DOUT <= 9;
    2'b10: DOUT <= 10;
    2'b11: DOUT <= 12;
    default: DOUT <= 4'bx;
endcase

```

**Array P**

```

always @ (posedge clk)
begin : array_R_RAM_logic Array R
    if (write_EN)
    begin
        array_R[write_addr] <= input_data;
    end
end
assign Output = array_R[read_addr];

```

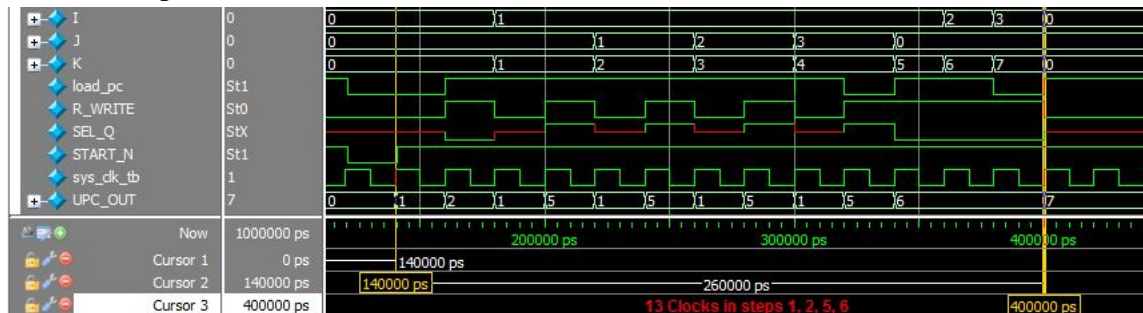
Different ports

To facilitate completing the first page of the schematic, we have provided a set of generic verilog files which can be modified as needed depending on the size of components you need. These are: u\_program\_memory.v and counter\_3bit.v. Edit these as needed to serve as your microprogramming memory and the microprogram counter.

A testbench (merge\_arrays\_mcu\_tb.v) is also provided to you to verify your core design in simulation.

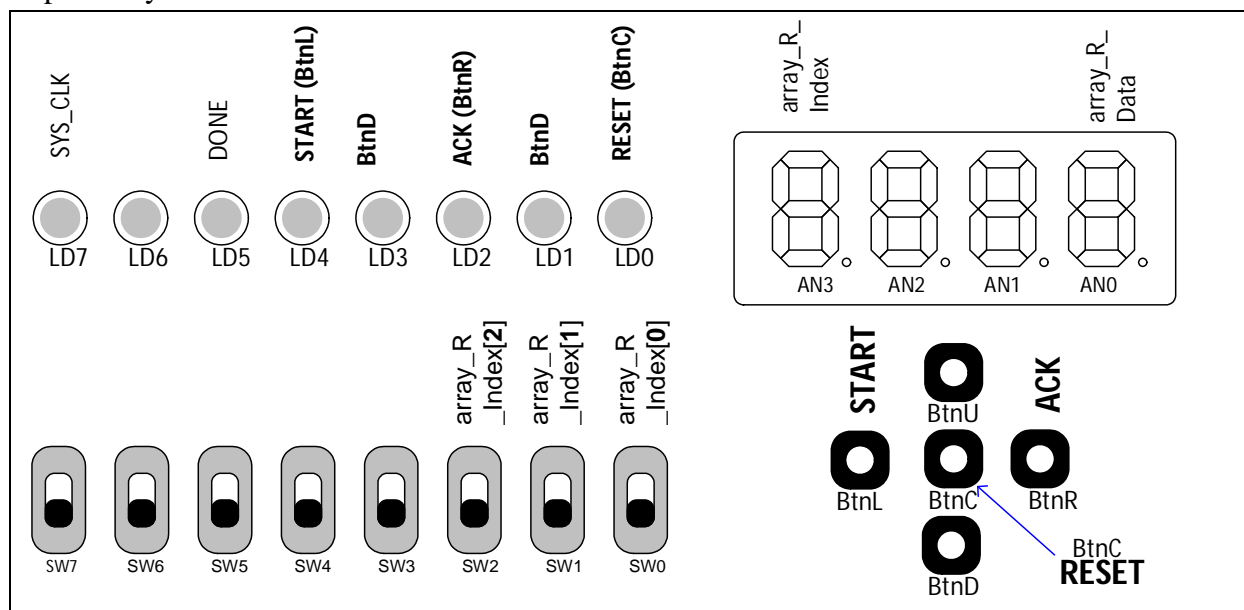
## 5. Simulate the core design

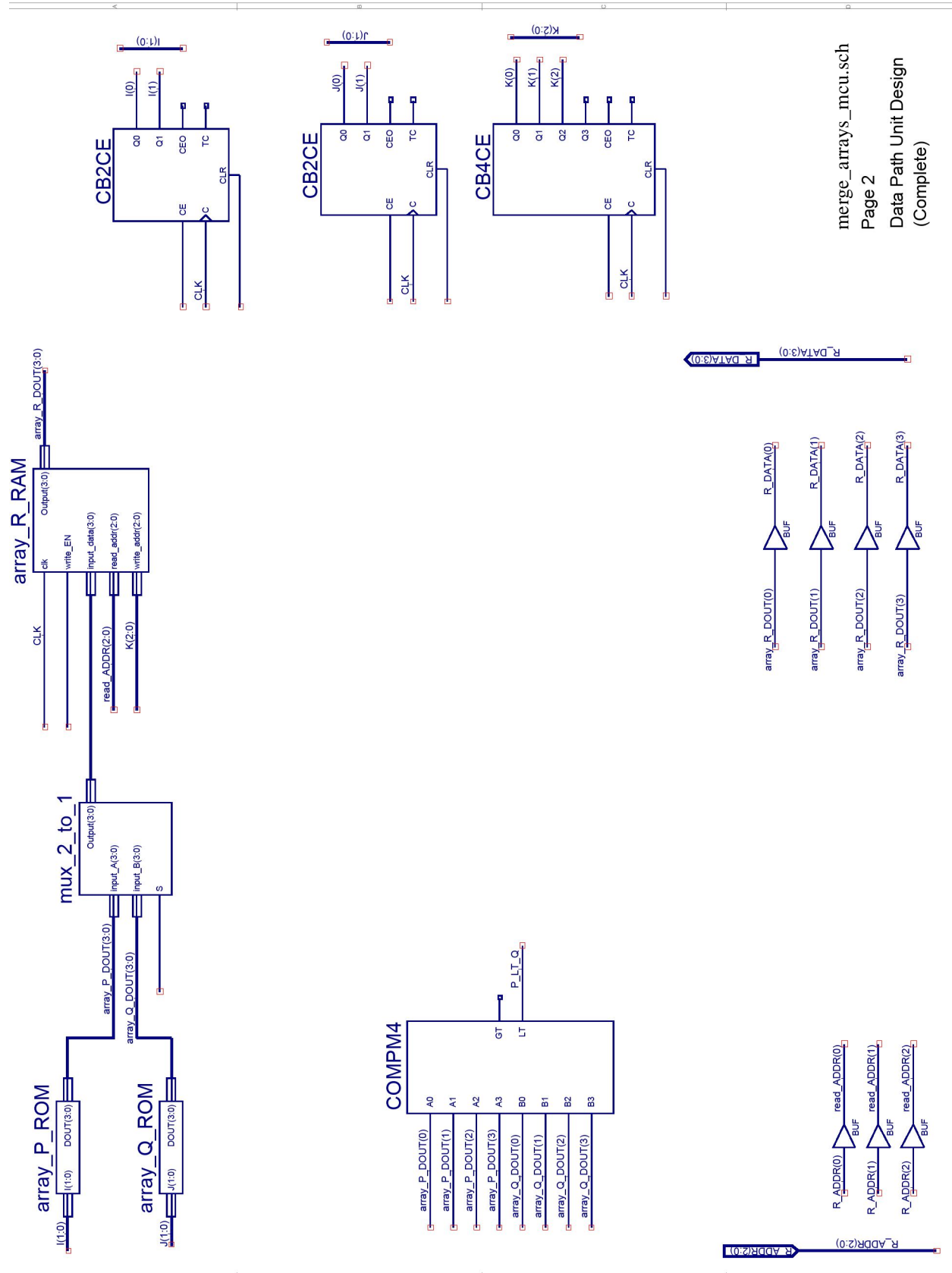
You are given a testbench (merge\_arrays\_mcu\_tb.v) which instantiates your core design (merge\_arrays\_mcu.sch) as UUT. Simulate your core design and verify. Notice that it took 13 clocks in the states (steps) other than INITIAL, DUMMY, and DONE.



## 6. User Interface

You are given a top design (merge\_arrays\_mcu\_top.v) which is complete. It instantiates your core design (merge\_arrays\_mcu.sch). BtnC is used as Reset and LD7 shows the system clock (~1.5 Hz). The DONE signal is shown on LD5. In the INITIAL state the machine waits until user presses BtnL to signal START. Once the state machine finishes merging the two arrays, it goes to the DONE state and asserts the DONE signal. Note that at 1.5 Hz, 13 clocks take about **8.7sec**. So many of us may think that our design failed because we fail to see the Done signal (on LD5) immediately after pressing the Start button. In the DONE state, user can use SW2-SW0 to set address to index into the result array R and inspect the data stored in it. The address set on these three switches and the contents of the addressed location in R are shown on SSD3 & SSD0 respectively as shown below.





**7. Prelab**

Q 7. 1: Suppose the data in array P is 0, 3, 10 and 12 and in array Q is 1, 2, 14 and 15. What would be the final contents of array R (after the arrays P and Q have been merged)? (3 pts)

Location:	0	1	2	3	4	5	6	7
array_P:	0	3	10	12				
array_Q:	1	2	14	15				
array_R:	—	—	—	—	—	—	—	—

Q 7. 2: Given the above as contents of P and Q, what will be the sequences of states during the merge? (10 pts)

INITIAL, CMP, SP (to store 0), CMP, SQ (to store 1), \_\_\_\_\_ , \_\_\_\_\_ ,

\_\_\_\_\_ , \_\_\_\_\_ , \_\_\_\_\_ , \_\_\_\_\_ , \_\_\_\_\_ , \_\_\_\_\_ , \_\_\_\_\_ , \_\_\_\_\_ , DUMMY, DONE

Q 7. 3: Record all **branches** (conditional or unconditional) below. (10 pts)

State Transition	Branch Condition
INITIAL to INITIAL	~START

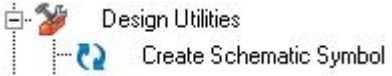
Q 7. 4: What is the required size of the microprogram counter and the control ROM? (5 pts)

Q 7. 5: Read the verilog files, array\_P\_ROM.v and array\_Q\_ROM.v, and state the contents of these arrays. (2 pts)

Location:	0	1	2	3
array_P:	—	—	—	—
array_Q:	—	—	—	—

## 8. Procedure

8.1 You are provided with ee201l\_merge\_arrays.zip which contains a partially complete Xilinx Project. This project contains both schematics and Verilog files in the design and the testbench for the core state machine. The project contains following significant files:

File	Description
merge_arrays_mcu.sch	This 2-page schematic file contains <u>incomplete control unit design</u> (page 1) and nearly complete data path design (page 2). Identify <i>all</i> the control signals of the data path unit, label the nets, and design the control unit to generates these signals. Building blocks, that are not available as standard library components, are provided (array_P_ROM.v, array_P_ROM.sym, array_Q_ROM.v, array_Q_ROM.sym, array_R_RAM.v, array_R_RAM.sym, mux_2_to_1.v, mux_2_to_1.sym).
u_program_memory.v (to serve an example)	This is the Verilog code for a 4-location deep, 2-bit wide microprogram control ROM. <u>Edit this Verilog</u> file to the required size and fill it with the correct data to be used in your MCU.
u_program_memory.sym (to serve an example)	This is the symbol corresponding to the above ROM component. Once the Verilog description of the ROM has been altered, <u>re-create this symbol</u> to match the new port sizes and use it in the MCU design. To create a symbol, select the Verilog file and under the process “Design Utilities”, double-click Create Schematic Symbol. 
counter_2bit.v & counter_2bit.sym (to serve an example)	You are provided with the Verilog description and the associated symbol for a 2-bit loadable counter. <u>Modify</u> this Verilog description and the associated symbol and use it as the microprogram counter in your design.
merge_arrays_mcu_tb.v	This is the testbench for the core design. Use this to test your design. Please be sure to retain the port-interface of the core design or modify the testbench (and top) accordingly.
merge_arrays_mcu_top.v	Top design, written in Verilog, instantiates the core design which is implemented in schematics (merge_arrays_mcu.sch).

8.2 Complete the core state machine design by implementing the control unit and adding the appropriate control signal names to the data path unit. Modify the data path, if necessary.

8.3 Simulate the design using the provided testbench (modify, if necessary) and show working design to your TA. Add important signals from the instantiated UUT to the waveform to display merging activity (example: I, J, and K counter outputs, P, Q, and R array data outputs, uPC values, branch addresses, etc.). Notice that the array\_R\_RAM.v is designed as a dual-port

```
merge_arrays_mcu UUT (
    .SYS_CLK(sys_clk) ,
    .RESET(Reset) ,
    .START(Start) ,
    .ACK(Ack) ,
    .R_ADDR(array_R_Index)
    .R_DATA(array_R_Data) ,
    .DONE (Done)
);
```



memory with separate read and write ports. Both the test bench and the top designs use the read port to inspect the contents of the array\_R.

8.4 Implement the design on to the FPGA board using the provided (complete) top. Modify the top, if necessary.

## 9. Lab Report:

Name: _____	Date: _____
Lab Session: _____	TA's Signature: _____

<b>For TAs:</b> Pre-lab (10): _____ Implementation (70): _____ Report (20): _____
Comments:

9.1 Is the DUMMY state necessary in this design? Why? (2pts)

9.2 Is the given state machine Moore or Mealy? Draw the equivalent other on the next page. (12pts)  
It is \_\_\_\_\_ (Moore/Mealy) because \_\_\_\_\_

Currently the I and J counters are 2-bit counters and the K counter is a 3-bit counter. If need to change any of the sizes for your design on the next page, please state which of these you need to change and why.

---

---

---

---

---

9.3 A microprogram control unit consists of three main components: microprogram counter, control ROM and condition select multiplexer. The microinstructions in the control ROM bear the format:

Condition Select	Branch Address	Control Signals
------------------	----------------	-----------------

Identify which component(s) or part of components serve(s) as the Next State Logic, State Memory and Output function Logic in a microprogram control unit. (6pts)

Next State Logic: \_\_\_\_\_

State Memory: \_\_\_\_\_

Output Function Logic: \_\_\_\_\_



