Search Table

CPE-324 Final Project

Juan Tarrat

04/25/2021

Introduction

This lab consisted in implementing a Binary Search Table. This table acted like a RAM, it has 1024 addresses, each of those addresses is composed of 48 bits entries each of which has 16-bit data in it. So, it acts as a hash table where we search for an entry and the table returns the contents associated with that entry.

The implementation of this design was divided into two specific parts: one was the part in charge of reading data, and the other one oversees the writing part. Both are two separate FSM, each of which will not work if the other one is working.

Design description

Figure 1 shows a schematic of the search table module being implemented.

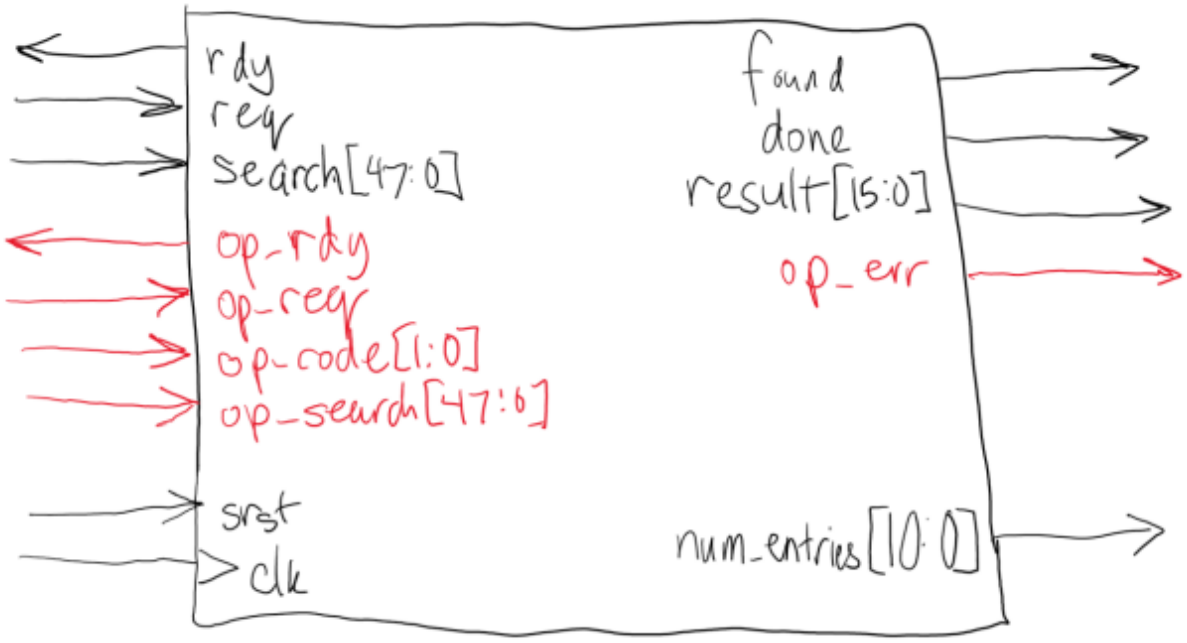

Figure 1: Search Table Module

The function of the input signals are as follows:

- ***REQ:*** Requests a search operation.
- ***SEARCH[47:0]***: Entry to be searched.
- ***SRST***: Synchronous reset.
- ***OP_REQ***: Requests a write operation based on ***OP_CODE[1:0]***
  - ***00:*** Add an entry to the table
  - ***01:*** Delete an entry from the table.
  - ***10:*** Modify data from an entry.
  - ***11:*** Clear table
- ***OP_SEARCH[47:0]***: Entry to perform a write operation in.
- ***OP_RESULT[15:0]***: Result to be written in, if needed by the op code.
- ***CLK:*** Input clock signal.

The function of the output signals are as follows:

- ***RDY:*** Informs that a new search operation can be performed.
- ***OP_READY:*** Informs that a new write operation can be performed.
- ***FOUND:*** High if item to be searched has been found.
- ***DONE:*** High when the search operation is finished.
- ***RESULT[15:0]***: Returns the data of the searched entry.
- ***OP_ERR:*** High if any of the operations fails.
- ***NUM_ENTRIES[10:0]***: Number of unwritten entries left.

Figure 2 shows the FSM in charge of the reading operation.

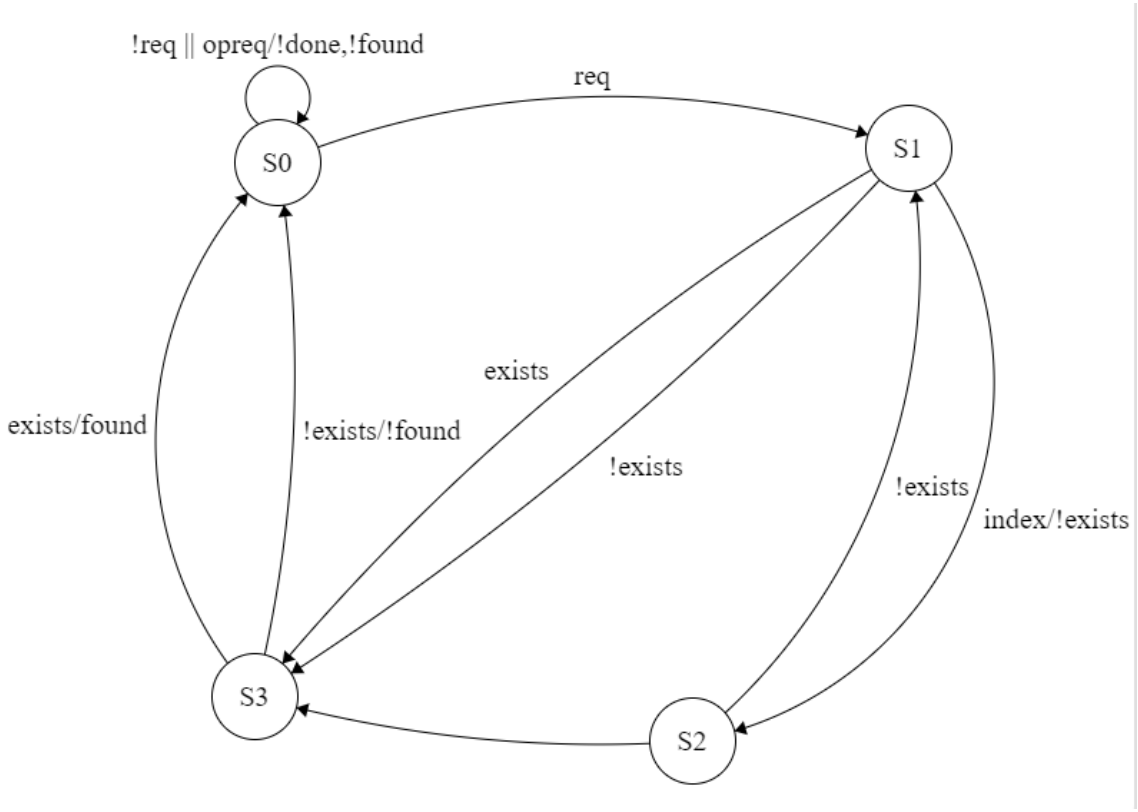

Figure 2: Read FSM

The way this FSM works is the following:

We stay in State 0 till the req signal is high or while there is a writing operation going on, once a reading is requested, we go to State 1 and check for special cases like an out of bounds search, an empty table or if the search is in either of the ends of the array. If neither of those conditions apply, we get an index to search for in State 2 and change our searching pointers if its not found. In the case it is not found we go back to State 1 to get a new index. In the case that we can conclude that an entry exists or not, we go to State 3 to set the outputs and go back to State 0 to wait for a new request.

Figure 3 shows the writing fsm.

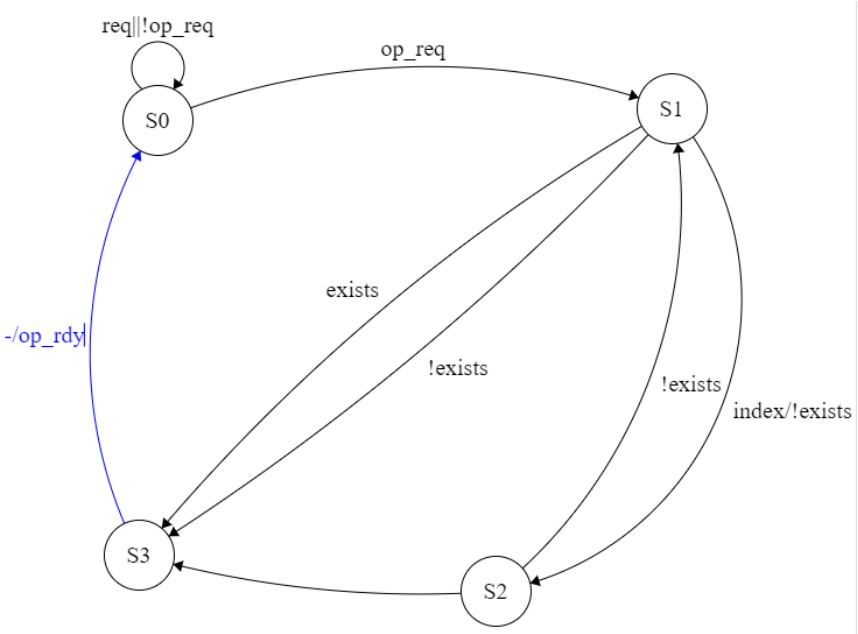

Figure 3: Write FSM

This fsm works in a very similar way as the other fsm with the only difference being in State 3, which is the state in which we perform the operation requested by input OP_CODE and we return OP_ERR if some operation did not go as expected. Some examples of situations that may trigger OP_ERR are: Trying to add an existing entry, trying to add to a full table, or trying to modify data from a non-existing entry.

The searches had to be performed using binary search, a well-known algorithm that looks for data in sorted arrays by dividing the search space by half each time. During early phases of the design I implemented the algorithm using while loops, however when trying to get something to work on Quartus, it was giving issues so I changed it to a FSM implementation. Figure 4 shows pseudo-code for the algorithm.

```
iLow = LBound(DataArray)
iHigh = UBound(DataArray)

Do While iLow <= iHigh
    iMiddle = (iLow + iHigh) / 2
    If Target = DataArray(iMiddle) Then
        bFound = True
        Exit Do
    ElseIf Target < DataArray(iMiddle) Then
        iHigh = (iMiddle - 1)
    Else
        iLow = (iMiddle + 1)
    End If
Loop
```

Figure 4: Binary Search pseudo code

Insertions have been implemented using bubble sort: when we want to add an entry we look for the index of the immediate entry above it and shift every other entry to a higher index or to a lower one if the operation selected is the DELETE one.

Issues and conclusion

My project is obviously “not finished”, or at least is not implemented in the FPGA as I would have wished. In the demo I made to the TAs I showed a working simulation in Model Sim, but that is the farthest I have reached. Ever since I started attempting a Quartus implementation I got the same error saying: Error (10119): Verilog HDL Loop Statement error. Loop with non-constant loop condition must terminate within 250 iterations. I managed to fix the issue in the while loops by changing the fsms but I still get errors in the for loops that perform the addition even though it worked in the simulation.

I tried everything. In fact, my code is not clean because I have been trying a bunch of different things. So, in my google drive upload I will upload both the code I used for the demo versus the code I ended up with while I write this report. The latter may have issues due to most recent changes.

Among my shots to try fixing the issue, apart of the changes made to the fsm, is also modify the iteration limit set by default in Quartus (250). This is done by modifying the QSF file and adding the following line:

set_global_assignment -name VERILOG_NON_CONSTANT_LOOP_LIMIT <number here>

That has not work either. Another thing I did is reduce the parameters of the memory to make it lighter but got no luck with that.

It has been a little frustrating because I have really tried to figure out what is wrong and I am disappointed with myself for not achieving a good functionality. I had the idea of including the hex display from previous labs and I even wrote the code to show the word “epty” when the table has no contents, but after not being able to get a successful compilation, I haven´t been able to test if it works. I will include the top file in the drive, however there is not a lot of information to gather from there since I have barely tried anything in it.

In conclusion, I am sad I did not get this to work. I really enjoyed designing the SearchTable module and if I had not have such a little time to work on this (because of finals and other projects) I would have probably been able to figure out a solution.

Link for the code: https://drive.google.com/drive/folders/1_8VsyOKZnMRwjJX2LG1xZd9-BdWdjdeW?usp=sharing