ASCII Memory Match Project Plan

By: Nikhil G. Khatu – 000507705

Summary:

Given an incoming 8-bit stream(indicated when the 'Go' flag is high) of ASCII characters the Memory matching hardware to be designed will find the best suited word match traversing through 16 words by 8 bit ASCII characters by 4 characters in a given dictionary. The best suited match will be the longest match between consecutive ASCII characters and words in the dictionary.

Given the dictionary contents; “ an,fred, and,frod, bat,batt,andi,ande,free, bah, a,frog,feed, fee, fed, ant” the sample I/O of Go flag, DataIn, and Found will be:

Clock	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Go	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
DataIn					c		a		n		sp		b		a		n		d		y		a		f
Found														a							a				
														n							n				
																					d				

Clock	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Go	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
DataIn	r		e		d		d																		
Found						f																			
						r																			
						e																			
						d																			

Project Execution plan:

1) While considering the specifications and requirements of the design pick an algorithm that is best suited. Understand the performance implications of picking this algorithm during design.

2) Verify the algorithm is correctly captured in a High-level code (e.g. 'C').

3) Explore the design. Design the higher level structure and hierarchy of hardware. Keep these few things in mind during the design:

- Top-down design: Consider the use of Parallelism and pipe lining to meet timing constraints. What micro-operations (multiply accumulates, memory references) will be used to evaluate data? What resources are needed to achieve the performance targets. Design so these resources are at a minimum and kept busy which will keep the design efficient. Given the algorithm how often and how fast can we access the registers?
- Bottom-up design: Design the data paths. What data will be used to make the logical decisions? Is this data time sensitive? Consider how many registers will be needed to hold this data? and for how many clock cycles will the data be needed? Once the data path is designed consider the control and status lines. What control approach is needed? e.g. Counters, FSMs, Pipeline control? After implementing the controller will the design meet timing constraints?

4) Code the design in verilog.

5) Verify the design is working according to specifications. Use test fixtures, simulation software, and tools to verify a correctly working implementation before Synthesis.

6) Synthesis – Use synthesis tools to get a gate level implementation that will maximize efficiency, minimize power consumption, area, and delay.

Miscellaneous things to consider: How fast can we access the registers? What is the rate of DataIn? Can we speed up the lookup time by use of a data mapping algorithm?

Verification plan

Once the design is set in place we will need to verify full functionality of the design. We need to verify that the design is working as expected. This step is critical in the overall outcome of the project; much money is spent on fabrication of ASICs. If the design is not correct the first time this investment is a big loss.

How do we go about verifying the design works correctly?
A few approaches to verification:

- An initial approach to debugging the design is to observe wave forms of the test fixture. We can view the transient data over a give period of simulated time.
- Given a set of asserted inputs(Clock, DataIn, the Data, and the Go flag) are we observing the correct outputs on Found?
- Test each individual module and as a group for verification,
- Are the timing constraints met? How about after synthesis?
- If the design is too extensive to test every individual test case we can write self-checking test fixtures to execute exhaustive tests. This is time efficient and less prone to human error.

Since this design is scaled down it is feasible to create individual test cases. Although 100% functional coverage is ideal, this becomes harder as the design is scaled up. Since this a small scale design we can expect up to 99.999999% functional verification coverage of the design.

High-level code:

The high level simulation of the design is written in 'C.' This language was chosen because it is a lower level language and can also hold a close representation of a hardware system e.g. Each 8 bits of ASCII char are equivalent of a char in 'C'. Since we have 4 ASCII chars in a word, the 'C' program is holding this word as an array of four 'char' values. The simulation is written in approximately 150 lines of code as follows:

- Create the Memory Array
- Initialize the variables to initial conditions, or 'x' (unknown)
- Enter the while loop:
  - Define the Data lines and registers
  - Save the last cycle's word and if there was a logical match
  - Wait for user input
  - Check SRAM memory for data matches
  - Output logic

The performance of the 'C' program will not translate into hardware. In hardware we can make use of pipe-lining and parallelism. The performance will have close relevance to the search algorithm used.

This is elaborated in the next section.

Algorithms/ Mode of operation:

Aho-Corasick-
O(m+n)
This algorithm requires dynamic allocation of states in an FSM during preprocessing (Also refererred to as a 'trie'). Due to its complexity it is not viable for this project.

Brute Force –
Searching: O(m*n)
This algorithm searches for every possible match through the dictionary. With the given time complexity, although viable this is not the best performing algorithm.

Boyer-Moore –
Pre-processing: O(m+o)
Searching: O(m*n)
Best case: O(n/m)

This is the best performing algorithm to iterate through a string to search for a match. Although the design will take advantage of parallelism, a version of Boyer-Moore will be implemented.

Risk Assessment Plan:
The biggest risk lies in the beginning of the project. Improper planning will contribute to a bad overall design which will bring more concerns later in the execution cycle. The current unknowns of the project are my biggest concern going forward. These are the specifics of the design. The overall, higher level design has been layed out in the CAD depiction.

Schedule:

7/2/2012
Pick the algorithm and code in high level language i.e. 'C'

7/16/2012
Design, Coding, and Verification completed.

7/20/2012
Synthesis and findings reported Final Presentation