ECE 520 Final Project Report

Nikhil Khatu
Student ID: 000507705

Delay (ns to run provided example)
Clock period: 2.56ns
#cycles: 34

Total: 87.04ns

1/(delay*area) in ($ns^{-1}*um^{-2}$)

1/(87.04*917) = 0.00001253

Area (um^2)
Logic: 917 – 526.6
= 390.32 um^2

Memory: 110 reg * 4.778
= 526.68 um^2

Total: 917 um^2

Energy (Joules)

n/a

For use by TA:

Delay
Clock period:

#cycles:

1/(delay*area) in ($ns^{-1}*um^{-2}$)

ABSTRACT

Hardware will be designed to find matches between an incoming data stream of ASCII characters and words in a dictionary stored in external SRAM. A project plan and proof of concept was created to meet the required specifications. The plan included Design execution, Verification, and Risk Assessment. After execution of the project plan the design was optimized and then synthesized for on chip implementations e.g. ASIC, FPGA, standard cell, Gate Array, etc.

TABLE OF CONTENTS

1. **Introduction**
2. **Interface Specification**
3. **Design Process**
4. **Verification**
5. **Synthesis**
6. **Conclusion/Results**

1) INTRODUCTION

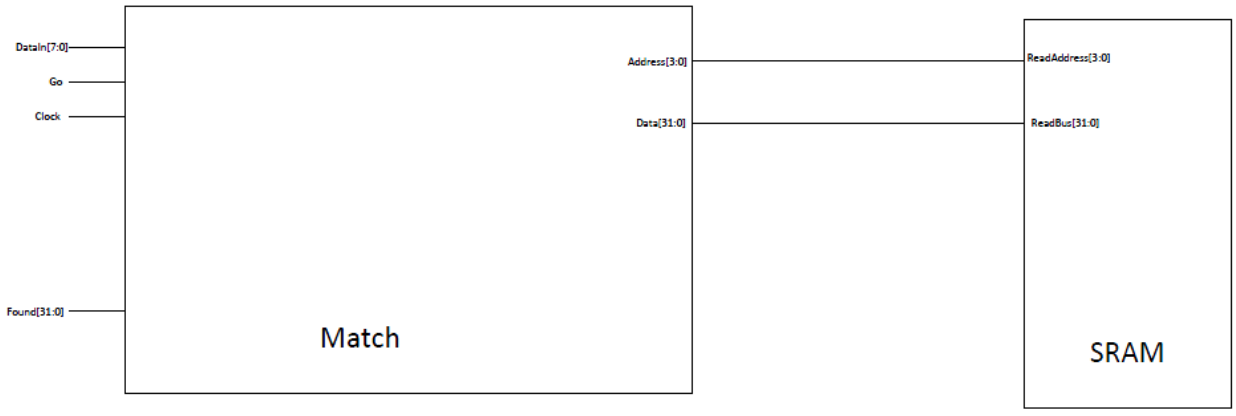
The designed hardware is a Finite State Machine implementation that waits for a 8-bit ASCII DataIn; when the 'Go' flag is raised the FSM begins the search for a match in the provided (4 ASCII letters)32-bit X 16 word dictionary. In summary, the design is achieving a clock period of 2.56 ns using the Nangate 45nm OpenCell Library. This translates to an operating Frequency of 390.62 MHz. The overall throughput rate of DataIn achieved is 11.49 MHz (one ASCII char every 87.04 nanoseconds). This report will provide details of implementing the execution plan, verification results, and synthesis results.

2) INTERFACE SPECIFICATION

Our port list for the Match Module is given as:

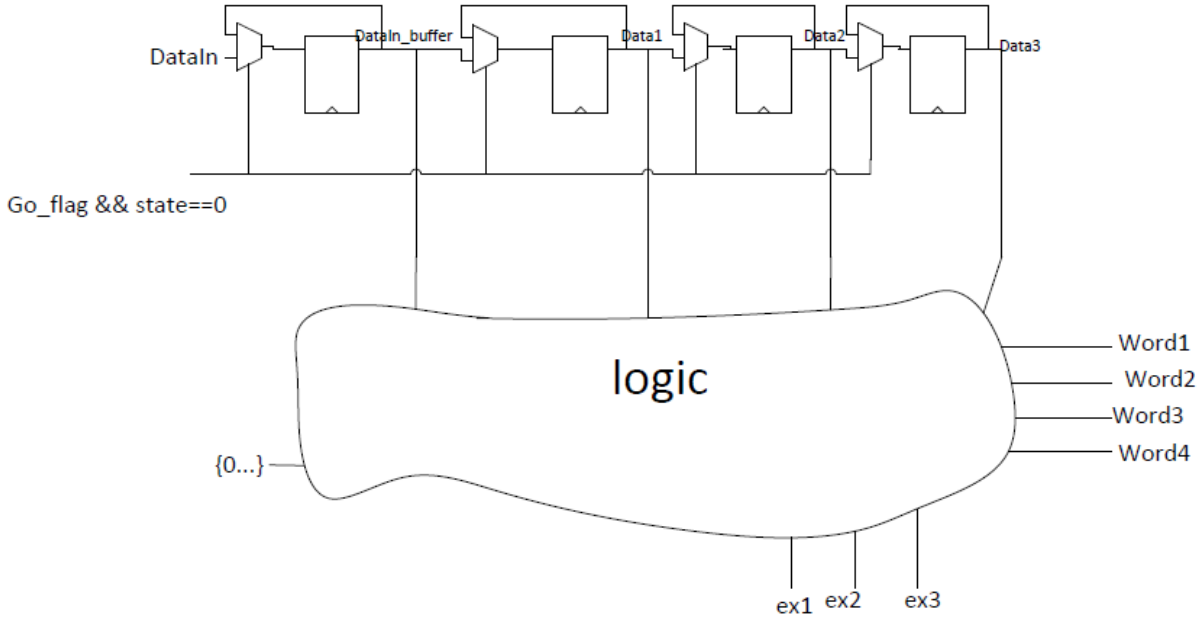
Port	Input/Output	Rate	Width	Description
DataIn	Input	11.49MHz	8 bits	Stream of ASCII characters
Go	Input	11.49MHz	1 bit	Indicates ASCII character is ready
Clock	Input	390.63MHz	1 bit	Global clock
Found	Output	n/a	32 bits	(4 x ASCII) match has been found in dictionary
Address	Output	Up to 390MHz	4 bits	Indicates address needed from SRAM (0 through 15)
Data	Input	n/a	32 bits	(4 x ASCII) word from SRAM

The designed Match module will interface with the given SRAM module which has a given delay of 300ps.

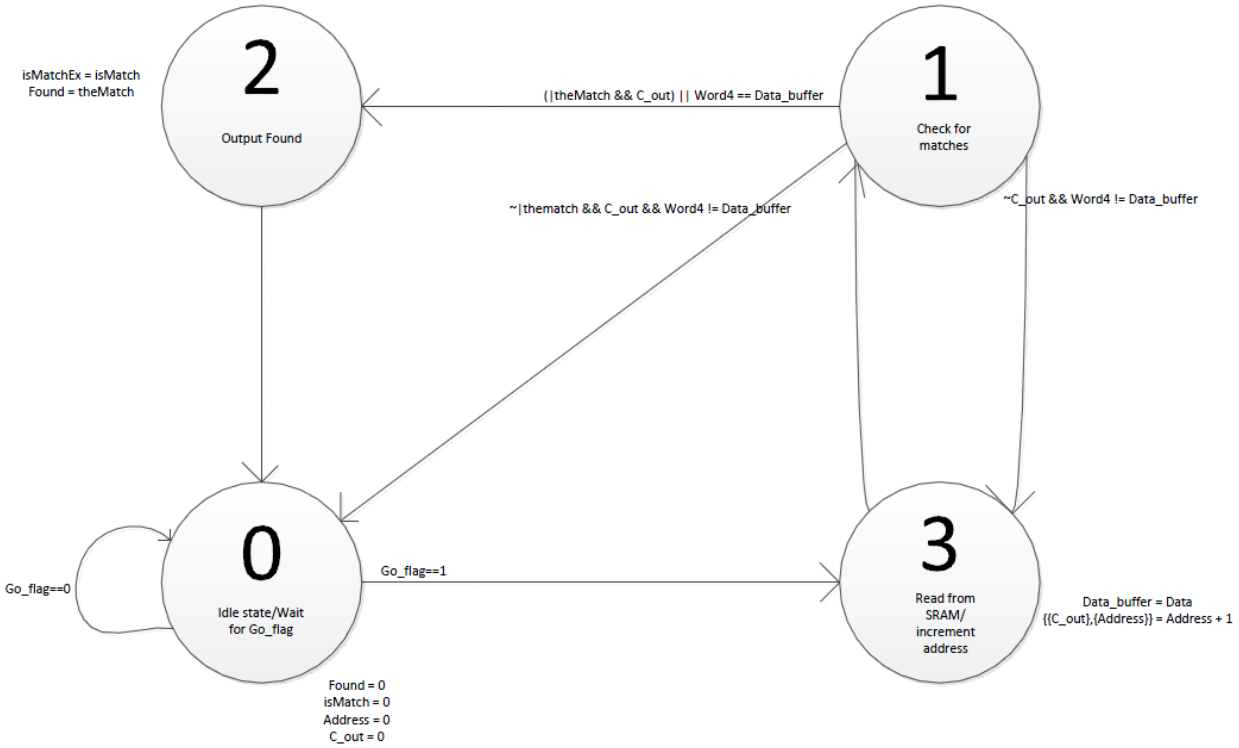


3) DESIGN PROCESS

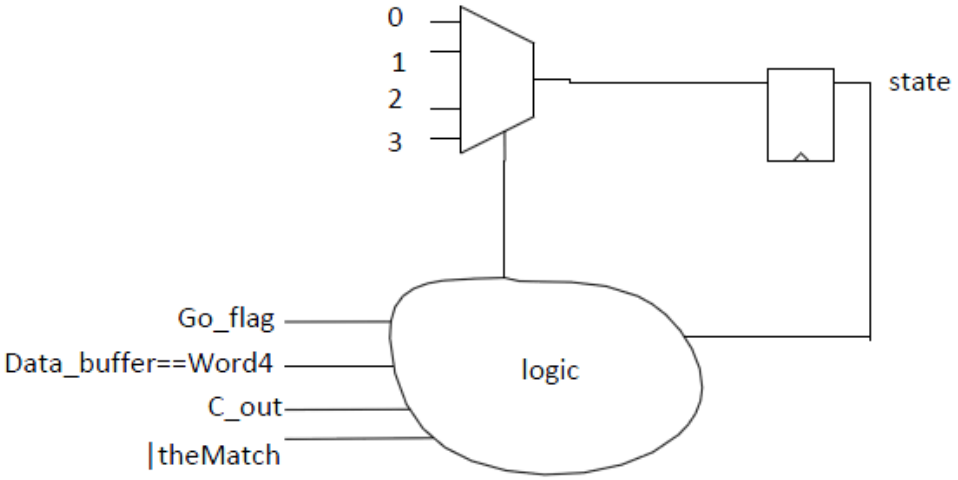
In order to implement this design we need to identify what data is needed for the logic used throughout the design. The fundamental registers used to remember the four most current ASCII inputs are identified as *Datain_buffer, Data1, Data2,* and *Data3* (shown in the figure below). Now that we have identified the registers, we can define the (a combination of zeros and the registers) combination of words we will use to compare against the dictionary. *Word1, Word2, Word3, Word4, ex1, ex2, and ex3* are all derived from these four registers. *Word1, Word2, Word3, and Word4* are the current words formed from the latest *DataIn. ex1, ex2*, and *ex3* are the corresponding set *of Word1, Word2, and Word3* used during the previous *DataIn*.



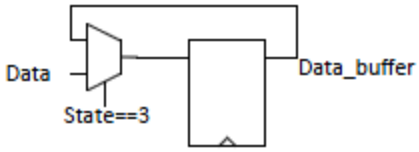
In order to move forward with the design we need a way to read data from the provided external SRAM module. The SRAM module has a simulated gate delay of 300 ps. An initial challenge in the design was: how do we account for the delay in our design? With the implementation of a Finite State Machine we can create a separate state just to read the Data from SRAM into a buffer called *Data_buffer*. The only limitation in doing so is the clock period should be above the 300ps SRAM delay. Since our synthesized clock period is 2.56ns we have 2.26ns of slack during this clock cycle. This translates to 2.26*16= 36.16ns of slack per DataIn. This shows that it is in our best interest to minimize the critical path timings when creating logic and also to reduce the total number of states accessed per DataIn. Each time another State is accessed it costs a period of 2.56ns. With this in mind we now have the following FSM that will wait for 'Go'(thus changing the Data registers), check for matches while iterating through the SRAM dictionary, and then output the Match when found.



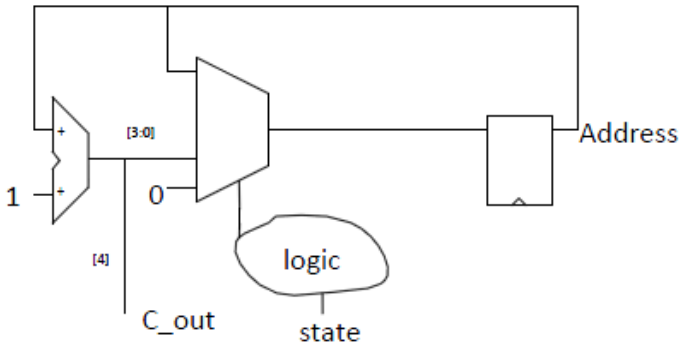
Our Finite State Machine diagram gives us the following hardware block. This includes the *state* register and the logic to determine the state during the next clock cycle.



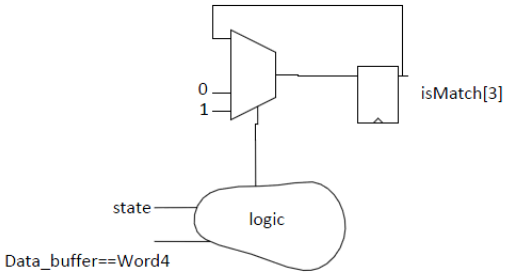
The logic for the *Data_buffer* registers is as follows:

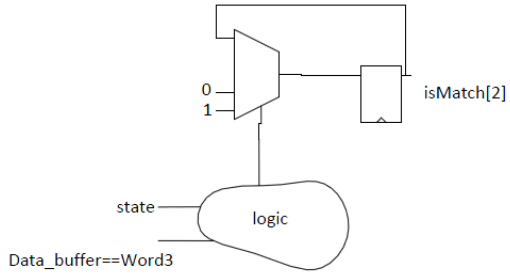


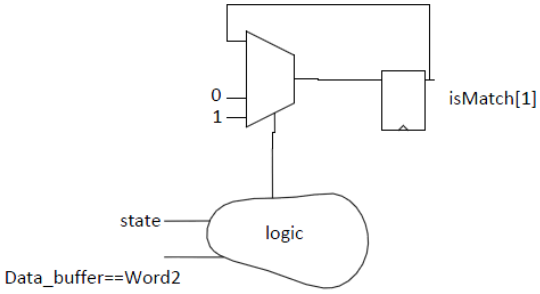
Our FSM suggest that we iterate between states ‘1’ and ‘3’ sixteen times(size of the dictionary). If in state ‘3’ we increment *Address* and if we’re in state ‘0’ we reset *Address* to 0. If in state ‘1’ or state’2’ *Address* stays the same.

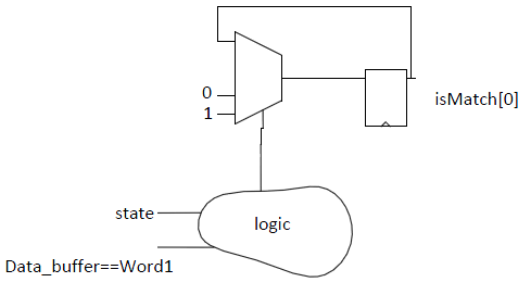


Now that we have a medium for receiving data from SRAM we need a way to check for matches between the latest *Data_buffer* and latest *Word1, Word2, Word3, Word4* while in state ‘1’. When in state ‘0’, *isMatch* is reset to 0. If matches are found the *isMatch* flag is raised to 1.

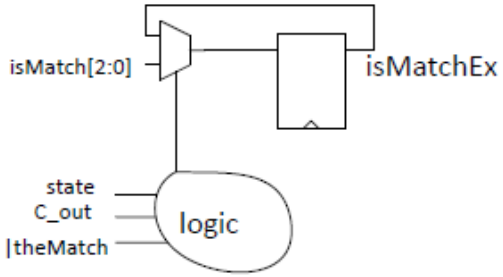




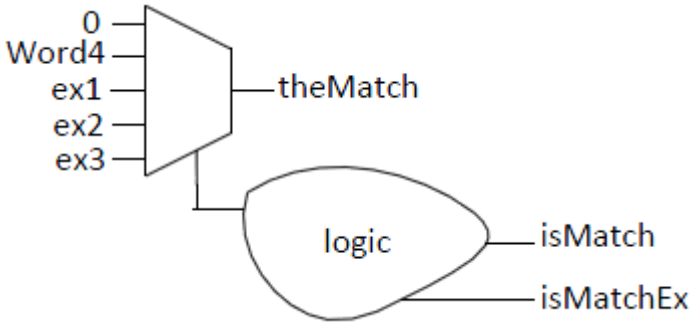




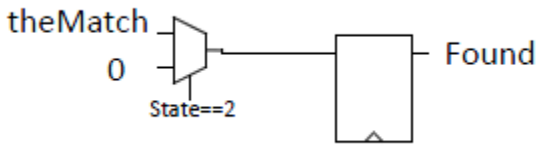
After iterating through all 16 SRAM locations we save *isMatch* to *isMatchEx*.
Note: A fourth *isMatchEx* bit is not needed, since we do not need to remember whether there was a *Word4* match. If there is a *Word4* match we are directly sent to state '2' for output on *Found*.



The following logic block uses input flags *isMatch* and *isMatchEx* to determine if there is a match worthy of putting on *Found.*
i.e. If there was a match with the previous *DataIn*, but there isn't a better match with the latest *DataIn* leads to an output. If *isMatch[3]* is raised then there will be a match of *Word4*.



theMatch is only put on the *Found* output register when in state '2' for one clock cycle.



4) ALGORITHM APPROACH

In the Project Plan there is discussion of using Brute Force, Boyer-Moore, or Aho-Corasick string matching algorithms to find the best possible match. In our design implementation the Brute Force algorithm is the best approach to finding a match. With Brute Force we can take advantage of parallelism in our design. Over time we have every synchronized combination given the set of DataIns saved in the registers. These combinations are compared in parallel with the current word/Data from the SRAM dictionary. The comparison is implemented/ synthesized in hardware with a series of XNOR and NAND gates.

Note: The use of Boyer-Moore and Aho-Corasick will both need extra states, which also need extra clock cycles to implement the algorithm. (Although there might be the added benefit of a faster critical path)

5) VERIFICATION

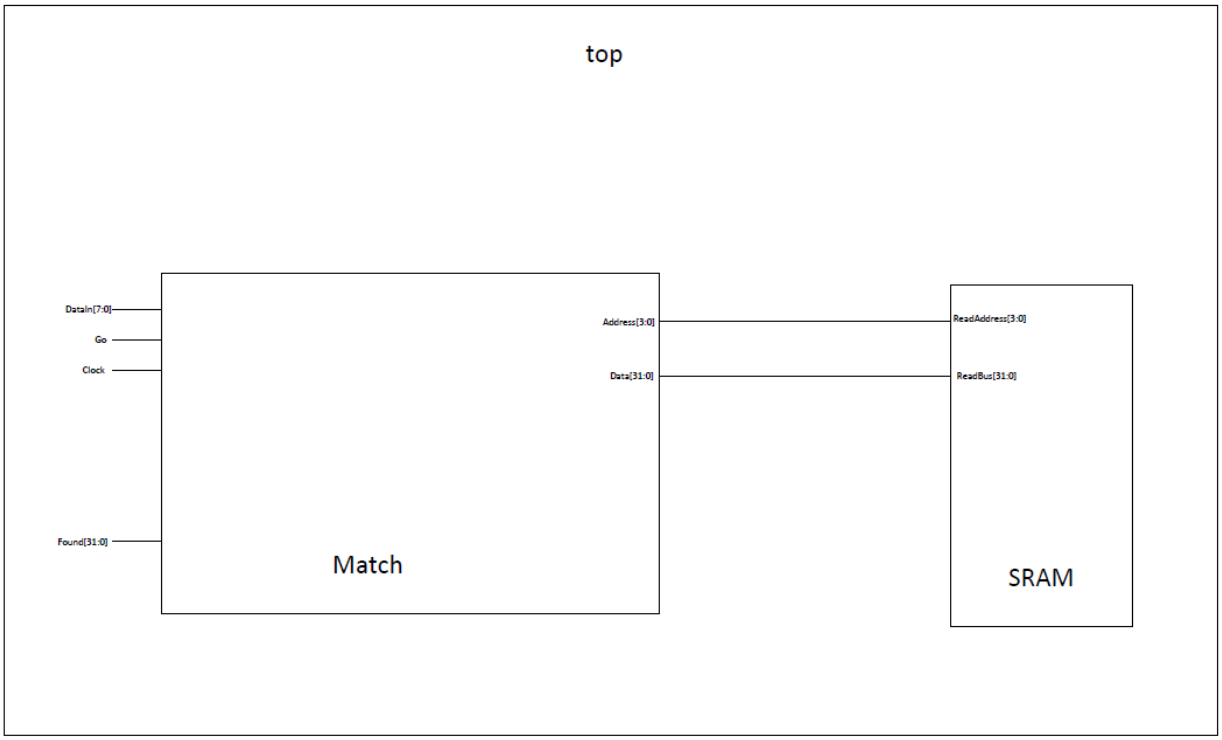
For verification purposes we are provided with a set of DataIn input vectors and expected results. The input vectors are in format: two hex characters and a 'Go" flag indication of 0 or 1. The two hex characters convert to one ASCII of *DataIn*. This *DataIn* and *Go* is used in each iteration of a for loop. The for loop iterates through the entire set of input test vectors.

Once we have the input vectors we can begin simulation of the FSM. Each iteration of the 'for loop' is held for the max window of 34 clock periods. Then we can begin testing with the next vector. This cycle is repeated 500 times for 500 test vectors. The command $fmonitor() was particularly helpful to report changes in the *found* output to a file. One can also use a waveform viewer to debug unexpected results.

Note: 34 periods =

1 ← period in state '0'
+ (16 *2) ← 16 periods in state '1' & 16 periods in state '3' when iterating SRAM
+ 1 ← period in state '2'

An instance of the top module is created under test_match for verification purposes. This is indicated in the following depiction.



6) SYNTHESIS

After completion of the design and verification we are now ready for synthesis. Since NCSU EDA is a contributor; the Nangate 45nm Open Cell Library is available for research purposes. Using the Synopsys design tool we can compile the design using this available library. Our setup.tcl, read.tcl, Constraints.tcl, and CompileAnalyze.tcl scripts will compile the Match design, and check for timing violations in worst case scenarios (given our specified clock period).

Our fastest clock period is found by decrementing the clock, running the script to generate the timing report, and then repeating this process until there is a violation. The lowest clock period where all timing reports are met is our fastest achievable clock period. The Synopsys Design Vision tool can be used to find the critical path by specifying the input and output pin. After finding the critical path we can make adjustments/determine if there is a way to reduce the timing.

7) CONCLUSION/ RESULTS

With the completion of design and verification we are able to run the match module successfully in the test bench. After verification is completed it is determined that there is extra output in the test run compared to our expected output. This is due to differences in the way the specification is interpreted.

e.g. We have the following words in the dictionary: '000a' and 'batt'
Note: '00at' is not in the dictionary.
When we have a DataIn stream of 'b' 'a' 't' 't' :

- The expected result is: "batt"
- However in our test bench we output: "000a" and "batt"

Once the verified design is Synthesized the fasted clock period is determined to be 2.56 nanoseconds. This translates to a clock frequency of 390.63 MHz. Each iteration in the for loop of the test bench runs for 34 periods or 34*2.56ns = 87.04ns. This is the longest time the FSM will take to find if there is a match. Exceptions to this time are:
- when a direct match is found (Match to *Word4*)
- when the 'Go' flag stays low for the duration of the iteration in the for loop

The 87.04ns time per DataIn translates to a throughput rate of 11.44MHz.

The synthesized area results are as follows:

```
design_vision-xg-t> report_area
****************************************
Report : area
Design : Match
Version: X-2005.09-SP3
Date   : Mon Jul 30 14:15:38 2012
****************************************
Library(s) Used:
   NangateOpenCellLibrary_PDKv1_2_v2008_10_typical_nldm (File:
/afs/eos.ncsu.edu/lockers/research/ece/wdavis/tech/nangate/NangateOpenCellLibrary_PDKv1_2_v2008_10/liberty/
520/NangateOpenCellLibrary_PDKv1_2_v2008_10_typical_nldm.db)
Number of ports:            78
Number of nets:            586
Number of cells:           449
Number of references:       38

Combinational area:      390.488159
Noncombinational area:   526.679688
Net Interconnect area:    undefined  (No wire load specified)

Total cell area:         917.168030
Total area:              undefined
```