

# ECE 520 Project Final Report Utterance Matcher ASIC

## **Group 21**

Frankie Myers (fbmyers)

Eric Phillips (ecphilli)

April 19, 2006

# TABLE OF CONTENTS

1. Introduction	1
2. Design Descriptions	1
3. Descriptions of Modules	2
3.1. Phone Scorer and Sorter	2
3.2. Word Matcher	4
3.3. ASCII Interface	7
4. Verification Strategy	8
5. Results	9
5.1. Throughput	9
5.2. Clock selection	9
5.3. Timing reports	9
5.4. Area	10
6. Conclusion	10

## Appendices

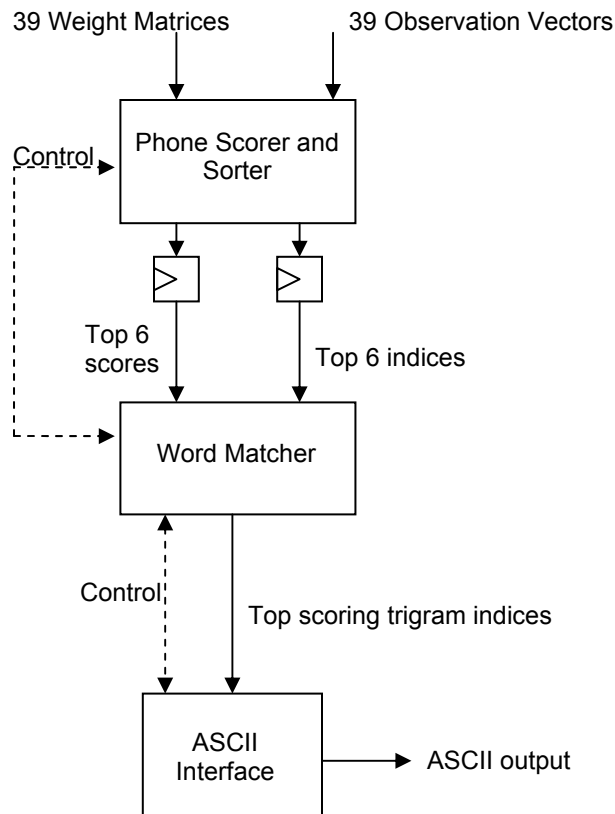
A: Verilog Modules and Test Benches	11
B: Synthesis scripts	36
C: Extracts from view_command.log	40
D: Plots from final design	51
E: Simulation results	54
F: High-level model of design	57

## 1. Introduction

This report introduces and describes a high-speed, optimized utterance matcher application specific integrated circuit (ASIC). The design identifies words spoken from the highest scoring string of phone observation vector inputs. A phone is a sequence of basic sound and can be combined to form a word or phrase. Each phone is a unique sequence. In order to constrain this project, only 39 out of the 51 that exist in American English are considered. The following sections detail the overall design, module descriptions, verification strategy, and design results from Synopsys.

## 2. Design Description

The overall design of the ASIC is shown below in Figure 1.



**Figure 1** – Block diagram of overall design

Preprocessed weight matrices are loaded at startup to the Phone Scorer. After this, the Ready signal is sent high and phone observation vectors are sent to the Phone Scorer. After all 39 scores have been calculated, the Scorer passes these scores to the Sorter which outputs the top 6 scores and their corresponding encoded phone indices 7 clock cycles later. This data is transferred to the Word Matcher and Trigram Matcher which look for valid combinations of phones and words, respectively, to find the most likely uttered trigram. Once the highest scoring trigram has been found, the ASCII Interface proceeds to read out the ASCII representation of the top three words from SRAM and outputs to the command line.

This design has been partitioned in this fashion so as to increase throughput by pipelining. As the stages execute in parallel, this allows for more data to be sent into the ASIC and reduces the amount of time that a stage is idle.

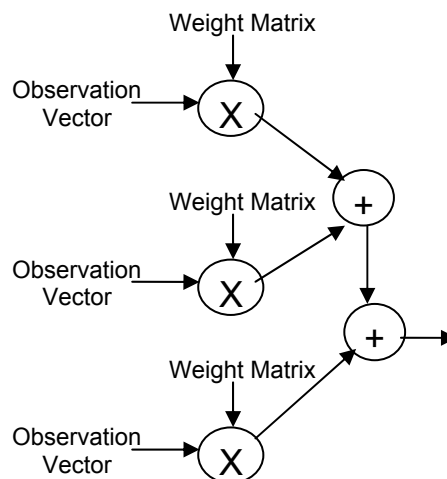
### 3. Descriptions of Modules

The design is divided into 3 modules to allow for effective grouping of tasks, pipelining, and for a sensible synthesis target. This was done by dividing tasks in the word matching process using a suggested module division technique from lecture. Each module receives inputs, processes them through combinational logic, and then stores the outputs in flip-flops. These outputs are then sent to the next module in the pipeline as inputs. The design was divided into the following modules: Phone Scorer and Sorter, Word Matcher, and ASCII Interface. Each module is described in further detail below.

#### 3.1. Phone Scorer and Sorter

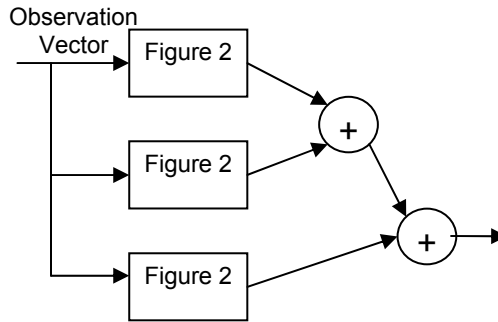
This module accomplishes two computational tasks. The first is to compute the score for each weight matrix multiplied by its corresponding phone observation input. The next task is to sort the 39 phones to find the top 6 scores and their corresponding index. The scoring and sorting functions were combined into one module due to several reasons. If the 39 phone scores were to be saved in flip-flops, this would add a clock cycle of delay as well as extra area. With only the top 6 phone scores being utilized in the word matching, there is no need to store all 39 phones. This combination also provides an efficient grouping of similar tasks. The front-end of the system is responsible for determining the top 6 scores and indices, therefore it makes sense to combine the scoring and sorting of phones into one module.

The phone scorer utilizes parallel operation to compute the scores of the 39 phones. This is accomplished by sending in all 39 phone observation vector inputs at the same time to compute the phone scores in parallel. Each phone scorer will access its specific weight matrix and multiply it with the incoming phone observation vector. This is then sent through an adder tree and produces the phone score vector for that phone. Figure 2 displays a block diagram of a phone vector scorer.



**Figure 2** – Block diagram of phone vector scorer

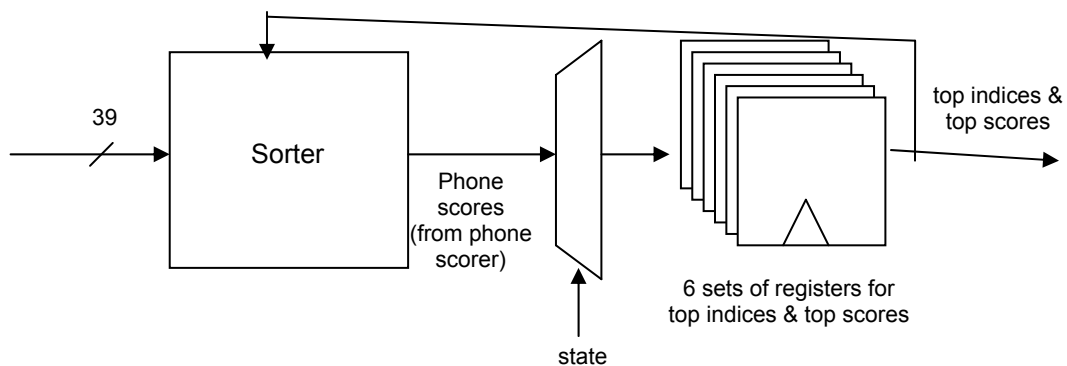
Each phone scorer contains 3 vector scorers added up to produce the phone score. This is also sent through an adder tree and is shown below in Figure 3.



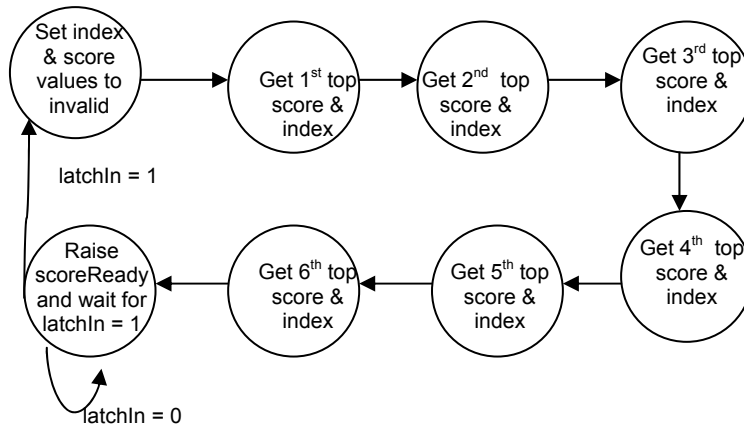
**Figure 3 – Block diagram of phone scorer**

An initial design structuring for the phone scorer utilized a carry-save adder structure from the DesignWare library. After further investigation, it was determined that using either an adder tree or the carry-save adder resulted in an equal number of adder structures. This is due to the fact that the carry-save adder was designed to be placed into a series of carry-save adders. Therefore the carry out of the phone vector scorer had to be added to the sum output before being fed into another carry-save adder that would produce the phone score. Therefore, in order to reduce complexity a simplified adder tree structure was utilized as shown above in Figures 2 and 3.

The sorter receives the 39 phone scores and indices and takes 7 clock cycles to determine the top 6 scores. It uses a modified cascaded binary search tree algorithm which is started by a control line and this begins a finite state machine. These are both shown below in Figures 4 and 5.



**Figure 4 – Modified cascaded binary search tree**



**Figure 5** – Finite State Machine representation of the phone sorter

Figure 4 utilizes a tree of comparators and multiplexers to determine the maximum score. An example line of code is shown below:

```
tree_path[19]=(tree_path[0]?nps0:nps1)>(tree_path[1]?nps2:nps3);
```

The tree\_path variables are the result of comparing two phone scores, where in this example tree\_path[0] determines if phone score 0 > phone score 1 and tree\_path[1] determines if phone score 2 > phone score 3. By combining these together, the maximum score can be determined. After computing which tree has the largest number, a variable topIndex is set to the maximum score by re-tracing through the tree\_path variables.

Each phone score passes through a multiplexer before reaching the comparator tree. This is utilized to block, or omit, high scores found in the previous stages. By doing so, the top 6 scores can be found. Without these multiplexers, each of the 6 phone scorers would determine the highest score was the maximum of all 39 phones. If a phone score has been identified as a maximum by the previous stages, the select line for that multiplexer is set high and the phone score is blocked by inputting a 0 into the comparator tree in its place. Figure 6 displays the block diagram of the phone sorter.

Once the latchIn signal is set high, it triggers the FSM shown in Figure 5. This causes each index and score to be set to an invalid number (63 for the index, 0 for the score) and the FSM sequentially begins to utilize the same sorter to determine the maximum number from each set and blocks out previously found maximum values. Once this process is complete, the FSM raises scoresReady high and waits to process the next cycle of inputs. An initial approach to the sorter included 6 sorter modules which were cascaded together and the latchOut signal of one was sent to the next sorter's latchIn signal, leading to a trickle-down sorting method. The sorter was redesigned to utilize only one sorter because it reduces area significantly and also allows for reusing redundant logic.

### 3.2 Word Matcher

Figure 5 displays the FSM representation of the word matching algorithm used in this design.



All of the hardware which computes potential word matches from the input scores and determines the best 3-word sequence (trigram) is contained in the word\_matcher module. This module uses a scratchpad SRAM to keep track of potential word matches.

At each input iteration, the module receives the top six phones/scores from the phoneScorer module. Based on these phones, it builds a list of potential words in the word\_matcher SRAM. This list includes word start/stop iteration number (start\_t, stop\_t), the score for the word (which is the sum of its individual phone scores), and a status field indicating if the word has been only partially identified, if it is a complete match, or if it did not "survive," meaning only some of the phones were identified and the word was discontinued at point. Also included is a field which indicates the number of stretched phones for that word. Everytime a stretched phone is detected, a new entry is added to the WM table, since the algorithm must consider both the case where the phone is stretched and where it is not stretched.

After the algorithm receives 30 inputs (consisting of 6 phones each) OR after the manual\_override input is raised, whichever comes first, the algorithm begins searching for the highest scoring valid trigram. A trigram is defined as a valid 3-word sequence where there are no gaps between the words and the first word starts at the first input iteration,  $t=0$ . When the top trigram is found, the 3 word index values are output and the done output signal is raised. The module then shifts the contents of the word matcher table to remove the entries which started before the last phone that was resolved into the trigram. That is, all the words (or potential words) that appeared within the time span of the trigram are eliminated, and the start\_t/stop\_t values of the remaining word entries are also subtracted to reflect this shift. In effect, the input iteration directly after the last iteration that was resolved becomes  $t=0$ . Then the process starts over again, and the module waits until it has refilled the word matcher table with 30 input iterations worth of data (or the manual\_override signal is raised).

The module uses an internal finite state machine to keep track of what stage in the algorithm it is currently executing. After reset, it waits in the WAITING\_FOR\_SCORES state for a new set of inputs to be placed on the input bus (and the scores\_ready signal to go high). Then, it begins looping through the LOOK\_FOR\_WORD1/2/3 states, where it searches through the entire word match table and determines, for each entry, if the corresponding word should survive the current iteration (in other words, if the next phone in that word exists). To do this, it consults the phone dictionary, which contains numbers representing the phones for each word. The phone dictionary indicates the end of a word with a null character (similar to a null-terminated string), and when the algorithm sees that the end of a word in the word matcher table has been reached, it sets a word\_completed flag, which indicates that, on the next input iteration, that an entire dictionary search should be performed to identify words which potentially start directly after the completed word. In this way, the word matcher table is populated with all the potential words in the input space, and words are only identified if they are next to a previous word (or if they start at  $t=0$ ).

After the LOOK\_FOR\_WORD1/2/3 loop completes its search of the word matches table, if the word\_completed flag is high (from the previous iteration),

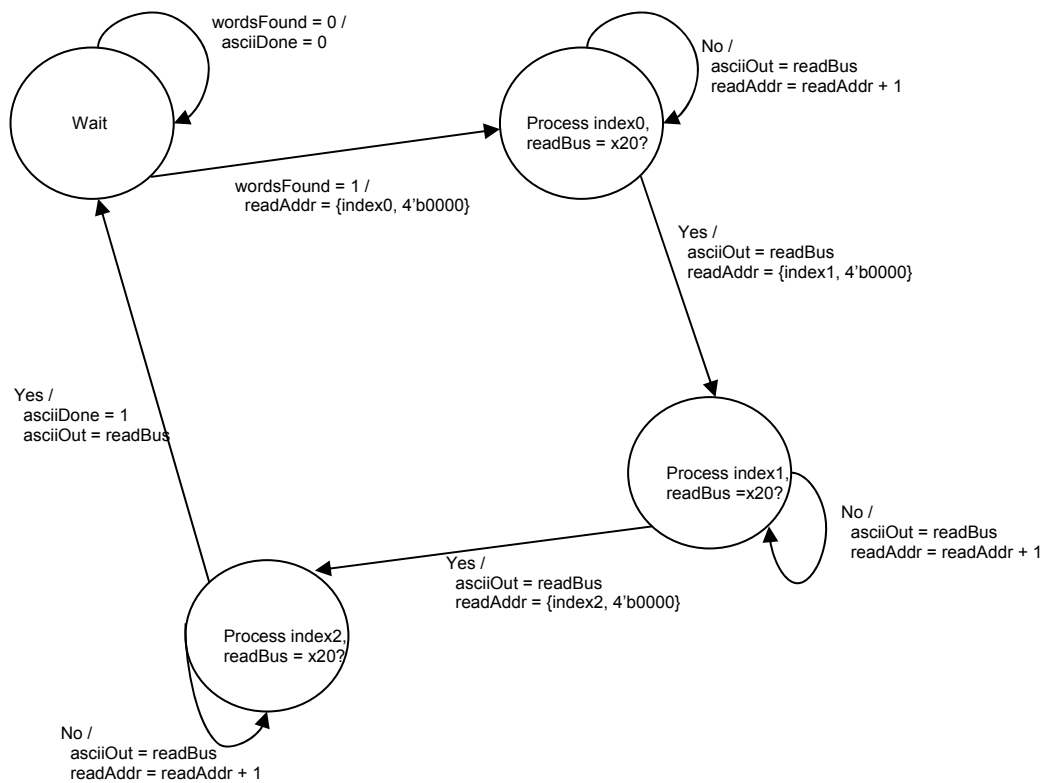
the SCAN\_FULL\_DICT state begins executing. It looks for words which start with one of the phones in the current 6, and for each such word it adds a row to the word matches table.

After 30 inputs have been received (or the manual\_override flag is raised), the module transitions to the LOOK\_FOR\_WORD1/2/3 states, which perform a recursive search for the highest-scoring 3-word sequence. Starting with LOOK\_FOR\_WORD1, the algorithm searches for (fully identified...i.e. status==MATCH) words in the word matches table which start at t=0. For each such word, it transitions to LOOK\_FOR\_WORD2, which searches for words which have been identified directly after the end of the first word. And finally, for every valid second word identified, the algorithm looks for valid third words which start after the end of the second word. For each third word identified, the algorithm compares the total score (word 1 score + word 2 score + word 3 score) with the current best score on record. If it is higher, the best score (and of course the word indices) are stored in registers. When the algorithm finishes, it raises the done output signal indicating that the best trigram has been found.

After the trigram scoring, the module transitions to the SHIFT\_WM\_TABLE1/2/3 states, which shift the contents of the word matches table according to the number of phones that were resolved into the trigram.

### **3.3. ASCII Interface**

The ASCII Interface is simply a 4 state Finite State Machine that is triggered by the controller module sending the wordsFound line high. This signifies that the trigram matcher has determined the indices of the three most likely uttered words. This leads to the module entering the state machine, which reads ASCII characters from an SRAM until it reads a space character (hexadecimal value of 20). It then outputs a space to the command prompt and moves on to the next index sent into the module. The FSM for this module is shown below in Figure 7.



**Figure 6** – Finite State Machine representation of the ASCII Interface

After sending all 3 words to the command prompt, the module sets the `asciiDone` signal high for one clock cycle to let the controller know that it is ready to accept the next 3 SRAM indices.

#### 4. Verification Strategy

Verification of a design is a very important process to complete when designing an ASIC. It not only provides a methodology for finding errors in the system, it can also provide a more thorough understanding of the control flow. This could result in minor changes within the code that could produce a more efficient design. One major issue in verification is trying to produce a reasonable number of test cases while not producing test cases that are very meticulous and account for every possible scenario.

The very naïve verification strategy of testing only the final design was avoided. Although this reduces time initially in not having to construct test benches this strategy only results in time spent debugging a system that is not well understood on a module-to-module basis. The design could produce incorrect outputs due to anything from a design error on a single line to a complete design failure in an entire module. Therefore this methodology was not used.

Therefore the methodology to test this design was incremental testing. Each module was simulated with specific inputs to test both functionality and ability to handle varying degrees of inputs. Once each module was successfully tested, a test was conducted between two modules to verify correct operation. After connecting more modules, the final verification step was reached with each module included. An example of this test strategy can be seen in Phone Scorer

and Sorter testing. The Scorer module was tested to check for one instantiation of the module producing the correct output. After this was successful, 39 instantiations of the module were checked for correct output by switching the input vectors among various indices. The Sorter module was next checked in a similar fashion by verifying one instantiation correctly found the top score. Then all 6 Sorter copies were tested together to check to verify the blocking feature worked correctly. After this, the Scorer and Sorter modules were combined into one test bench to verify that the correct input vectors were sorted accordingly and only the top 6 scores were sent to the output.

## **5. Results**

### **5.1. Throughput**

The throughput of this design is measured by calculating the number of acoustic vectors processed per unit time. This metric was found to be 26,084 clock cycles to process 30 vectors. Therefore, this design can process  $1.1501 \times 10^{-3}$  vectors per cycle. The clock selected for this design was 60ns, which is shown below in Section 5.2. Therefore by multiplying the vectors per cycle by cycles per second, the final throughput result of 19,168.333 vectors per second can be processed.

The performance per area can be calculated utilizing the throughput divided by the area. The total area found by Synopsys and SRAM utilization is shown below in Section 5.4 is 14601989.59. Therefore, the performance per unit area is  $1.31274 \times 10^{-3}$ .

### **5.2. Clock selection**

The design was synthesized using a 60ns clock period (50% duty cycle) with the following Synopsys Design Analyzer command:

```
Create_clock -period 60000 -waveform {0 30000} clock
```

With this clock period, the design is running at approximately 16.667MHz.

### **5.3. Timing reports**

Synopsys Design Analyzer allows for Verilog files to be read in and then begins to optimize and synthesize these designs into hardware. By utilizing scripts containing various Synopsys directives, a design can be synthesized fairly easily. The scripts used to synthesize this design are included in Appendix B.

The synthesis methodology used within this design is to first create a design and then check the critical path timing using the worst performing design library and check for setup violations. The timing report from Design Analyzer is shown in Appendix C on page 51 after completing this process, which is included in script2.sc.

Since the timing slack constraint has been met, the next step is to synthesize the design using the best performing design library and check for hold violations. This is done in script3.sc and the timing report from this run is shown in Appendix C on page 54.

At this point, the design library is reset to the worst performing and an additional script is run to recheck the setup constraints. This is due to the fact that Synopsys will usually insert additional logic into certain paths to correct hold violations. This additional logic could possibly affect the critical path and therefore affect the setup constraints. The timing report is shown in Appendix C on page 54 was generated after running script4.sc. This timing check also meets the slack requirements and therefore is acceptable.

#### 5.4. Area

After completing the optimization and timing requirements, the final script reads in the Synopsys command report\_area which calculates the area of the current design. This report shows that the total cell area of the design is 14,601,985.00 and is shown on page 57.

In order to account for the area added to the design by SRAM's, the following formula was used to account for the SRAM area:

$$\text{Area} = (4.4\mu) * (\# \text{ of rows}) + (5\mu) * (\# \text{ of columns}) + e * (\text{peripheral circuits})$$

$$\text{where } e = .1 * ((4.4\mu) * (\# \text{ of rows}) + (5\mu) * (\# \text{ of columns}))$$

There are 3 SRAM instantiations used within this design. They include two SRAM's interfacing with word\_matcher that are used to store the possible matches table and to store the phone representation of dictionary words. There is also an SRAM interfacing with the ASCII Interface that stores the dictionary words in an ASCII representation. Table 1 below displays the number of rows, columns, and area calculation for each SRAM.

	Rows	Columns	Area
phdict_sram	2145	6	2.0375136
wm_sram	1024	53	0.51427068
sramASCII	2144	8	2.03966608
<b>Total SRAM Area</b>	5313	67	4.59145036

**Table 1** – Area calculations for SRAM instantiations

Adding the above total SRAM area to the area reported from Synopsys, the total area of this design is 14601989.59.

## 6. Conclusion

This report has presented the design, test, and synthesis methodology to develop a high-speed, optimized utterance matcher ASIC. In utilizing efficient module division, this design was split into 3 modules. Each module is responsible for a different task within the pipeline from calculating phone scores and sorting them, to determining the most likely uttered trigram, to outputting the most likely uttered trigram to the command prompt. This project provided an excellent opportunity to develop and practice high-level design constructs while also providing the opportunity to design an entire system from specification to synthesis.

## Appendix A: Verilog module files (including test benches)

### top.v

```
/*
 * Top.v
 *
 * Authors: Frankie Myers (fbmyers) and Eric Phillips (ecphilli)
 * ECE 520 Project - Group 21
 *
 * This is the top module that combines all the sub-modules to form the ASIC
 */

`include "phoneScorer.v"
`include "word_matcher_3.v"
`include "ascii.v"

module top(clock, reset, latchIn, wm0, wm1, wm2, wm3, wm4, wm5, wm6, wm7, wm8, wm9, wm10,
wm11, wm12, wm13, wm14, wm15, wm16, wm17, wm18, wm19, wm20, wm21, wm22, wm23, wm24, wm25,
wm26, wm27, wm28, wm29, wm30, wm31, wm32, wm33, wm34, wm35, wm36, wm37, wm38, in0, in1,
in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13, in14, in15, in16, in17,
in18, in19, in20, in21, in22, in23, in24, in25, in26, in27, in28, in29, in30, in31, in32,
in33, in34, in35, in36, in37, in38, manual_override, readDataWMatchSRAM,
readDataPhoneDictSRAM, readASCIIBus, ready_for_input, writeEnable_WMatchSRAM,
writeAddrWMatchSRAM, readAddrWMatchSRAM, writeDataWMatchSRAM, readAddrPhoneDictSRAM,
asciiOut, readASCIIAddr);

    input clock;
    input reset;
    input latchIn;
    input [71:0] wm0, wm1, wm2, wm3, wm4, wm5, wm6, wm7, wm8, wm9, wm10, wm11, wm12, wm13,
wm14, wm15, wm16, wm17, wm18, wm19, wm20, wm21, wm22, wm23, wm24, wm25, wm26, wm27, wm28,
wm29, wm30, wm31, wm32, wm33, wm34, wm35, wm36, wm37, wm38;
    input [23:0] in0, in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13,
in14, in15, in16, in17, in18, in19, in20, in21, in22, in23, in24, in25, in26, in27, in28,
in29, in30, in31, in32, in33, in34, in35, in36, in37, in38;
    input manual_override;
    input [52:0] readDataWMatchSRAM; // data read from word match SRAM
    input [5:0] readDataPhoneDictSRAM; // data read from phone dictionary SRAM
    input [7:0] readASCIIBus; // ASCII SRAM interface; readBus line from ASCII SRAM

    output ready_for_input; // goes high when word matcher is ready for new input sets
    output writeEnable_WMatchSRAM; // write-enable for word match SRAM
    output [9:0] writeAddrWMatchSRAM; // write address for word match SRAM
    output [9:0] readAddrWMatchSRAM; // read address for word match SRAM
    output [52:0] writeDataWMatchSRAM; // data to write to the word match SRAM
    output [11:0] readAddrPhoneDictSRAM; // read address for phone dictionary SRAM
    output [7:0] asciiOut; // 8-bit hexadecimal value that is sent to the command prompt
    output [11:0] readASCIIAddr; // ASCII SRAM interface; readAddr line to ASCII SRAM

    // Net listings inside top.v module
    wire phoneScoreReady;
    wire [19:0] phoneScore0, phoneScore1, phoneScore2, phoneScore3, phoneScore4,
phoneScore5;
    wire [5:0] phoneIndex0, phoneIndex1, phoneIndex2, phoneIndex3, phoneIndex4,
phoneIndex5;
    wire wordsFound;
    wire [7:0] asciiIndex0, asciiIndex1, asciiIndex2;

    phoneScoreSort scoreSorter(clock, reset, latchIn, phoneScoreReady, phoneIndex0,
phoneIndex1, phoneIndex2, phoneIndex3, phoneIndex4, phoneIndex5, wm0, wm1, wm2, wm3, wm4,
wm5, wm6, wm7, wm8, wm9, wm10, wm11, wm12, wm13, wm14, wm15, wm16, wm17, wm18, wm19, wm20,
wm21, wm22, wm23, wm24, wm25, wm26, wm27, wm28, wm29, wm30, wm31, wm32, wm33, wm34, wm35,
wm36, wm37, wm38, in0, in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12,
in13, in14, in15, in16, in17, in18, in19, in20, in21, in22, in23, in24, in25, in26, in27,
in28, in29, in30, in31, in32, in33, in34, in35, in36, in37, in38, phoneScore0,
phoneScore1, phoneScore2, phoneScore3, phoneScore4, phoneScore5);

    word_matcher wordMatcher(phoneScore0, phoneScore1, phoneScore2, phoneScore3,
phoneScore4, phoneScore5, phoneIndex0, phoneIndex1, phoneIndex2, phoneIndex3,
phoneIndex4, phoneIndex5, asciiIndex0, asciiIndex1, asciiIndex2, ready_for_input,
wordsFound, phoneScoreReady, manual_override, writeEnable_WMatchSRAM,
writeAddrWMatchSRAM, readAddrWMatchSRAM, writeDataWMatchSRAM, readDataWMatchSRAM,
readAddrPhoneDictSRAM, readDataPhoneDictSRAM, clock, reset);

    ascii asciiInterface(clock, reset, wordsFound, asciiIndex0, asciiIndex1, asciiIndex2,
readASCIIAddr, readASCIIBus, asciiOut);
```

```
endmodule // top
```

## testTop1.v

```
`include "top.v"
`include "wm_sram.v"
`include "phdict_sram.v"
`include "sramASCII.v"

`define NUM_INPUTS 36+1 //we're appending an extra input so that words which run to the
end of the inputs will get identified

`define DEBUG

module testTop;
    reg clock;
    reg reset;
    reg latchIn;
    reg [71:0] wm0, wm1, wm2, wm3, wm4, wm5, wm6, wm7, wm8, wm9, wm10, wm11, wm12, wm13,
    wm14, wm15, wm16, wm17, wm18, wm19, wm20, wm21, wm22, wm23, wm24, wm25, wm26, wm27, wm28,
    wm29, wm30, wm31, wm32, wm33, wm34, wm35, wm36, wm37, wm38;
    reg [23:0] in0, in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13,
    in14, in15, in16, in17, in18, in19, in20, in21, in22, in23, in24, in25, in26, in27, in28,
    in29, in30, in31, in32, in33, in34, in35, in36, in37, in38;
    reg manual_override;
    reg [23:0] inputs [0:39*`NUM_INPUTS-1];
    reg [71:0] pwms [0:38];

    wire [52:0] readDataWMatchSRAM; // data read from word match SRAM
    wire [5:0] readDataPhoneDictSRAM; // data read from phone dictionary SRAM
    wire [7:0] readASCIIbus; // ASCII SRAM interface; readBus line from ASCII SRAM
    wire ready_for_input; // goes high when word matcher is ready for new input sets
    wire writeEnable_WMatchSRAM; // write-enable for word match SRAM
    wire [9:0] writeAddrWMatchSRAM; // write address for word match SRAM
    wire [9:0] readAddrWMatchSRAM; // read address for word match SRAM
    wire [52:0] writeDataWMatchSRAM; // data to write to the word match SRAM
    wire [11:0] readAddrPhoneDictSRAM; // read address for phone dictionary SRAM
    wire [7:0] asciiOut; // 8-bit hexadecimal value that is sent to the command prompt
    wire [11:0] readASCIIaddr; // ASCII SRAM interface; readAddr line to ASCII SRAM

    integer i,j;
    reg [52:0] wm_row;
    top topster(clock, reset, latchIn, wm0, wm1, wm2, wm3, wm4, wm5, wm6, wm7, wm8, wm9,
    wm10, wm11, wm12, wm13, wm14, wm15, wm16, wm17, wm18, wm19, wm20, wm21, wm22, wm23, wm24,
    wm25, wm26, wm27, wm28, wm29, wm30, wm31, wm32, wm33, wm34, wm35, wm36, wm37, wm38, in0,
    in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13, in14, in15, in16,
    in17, in18, in19, in20, in21, in22, in23, in24, in25, in26, in27, in28, in29, in30, in31,
    in32, in33, in34, in35, in36, in37, in38, manual_override, readDataWMatchSRAM,
    readDataPhoneDictSRAM, readASCIIbus, ready_for_input, writeEnable_WMatchSRAM,
    writeAddrWMatchSRAM, readAddrWMatchSRAM, writeDataWMatchSRAM, readAddrPhoneDictSRAM,
    asciiOut, readASCIIaddr);

    wm_sram wordMatchesScratchpad(clock, writeEnable_WMatchSRAM, writeAddrWMatchSRAM,
    readAddrWMatchSRAM, writeDataWMatchSRAM, readDataWMatchSRAM);
    phdict_sram phoneDict(clock, 1'b0, 12'b0, readAddrPhoneDictSRAM, 6'b0,
    readDataPhoneDictSRAM);
    sramASCII asciiDict(clock, 1'b0, 12'b0, readASCIIaddr, 8'b0, readASCIIbus);

    always #5 clock = ~clock;

    `ifdef DEBUG
        always@(posedge topster.wordMatcher.word_completed)
            $display("\n-----A WORD WAS FOUND, RELOADING DICTIONARY-----\n");
        always@(posedge topster.wordMatcher.stretching_found)
            $display("\n---STRETCHING FOUND, ADDING NEW (STRETCHED) WORD ENTRY---\n");
        always@(posedge topster.wordMatcher.done)
            begin
                $display("-----WORD MATCHING COMPLETE-----");
                $display ("Trigram Words: %0d\t%0d\t%0d",
                    topster.wordMatcher.best_trigram_index1,
                    topster.wordMatcher.best_trigram_index2,
                    topster.wordMatcher.best_trigram_index3);
                $display("Trigram Score: %0d",topster.wordMatcher.best_trigram_score);
                $display("# Phones Resolved:%0d\n",topster.wordMatcher.best_trigram_length);
                $display("\nASCII OUTPUT:");
            end
    `endif
end
```

```

always@(asciiOut)
    $write("%s",asciiOut);

initial
    begin
        //$dumpfile("waves.vcd");
        //$dumpvars(1,topster.wordMatcher);

        clock = 1;
        reset = 0;
        manual_override = 0;
        $readmemh("acousticvectors2.dat", inputs);
        $readmemh("phoneweightmatrices2.dat",pwms);
        $readmemh("phdict.dat", phoneDict.Register); // load in dictionary with words
        represented by phones
        $readmemh("asciidict.dat", asciiDict.Register); // load in dictionary with words
        represented by ASCII

        #10 reset = 1;

        // send in weight matrices
        #10
        wm0 = pwms[0];
        wm1 = pwms[1];
        wm2 = pwms[2];
        wm3 = pwms[3];
        wm4 = pwms[4];
        wm5 = pwms[5];
        wm6 = pwms[6];
        wm7 = pwms[7];
        wm8 = pwms[8];
        wm9 = pwms[9];
        wm10 = pwms[10];
        wm11 = pwms[11];
        wm12 = pwms[12];
        wm13 = pwms[13];
        wm14 = pwms[14];
        wm15 = pwms[15];
        wm16 = pwms[16];
        wm17 = pwms[17];
        wm18 = pwms[18];
        wm19 = pwms[19];
        wm20 = pwms[20];
        wm21 = pwms[21];
        wm22 = pwms[22];
        wm23 = pwms[23];
        wm24 = pwms[24];
        wm25 = pwms[25];
        wm26 = pwms[26];
        wm27 = pwms[27];
        wm28 = pwms[28];
        wm29 = pwms[29];
        wm30 = pwms[30];
        wm31 = pwms[31];
        wm32 = pwms[32];
        wm33 = pwms[33];
        wm34 = pwms[34];
        wm35 = pwms[35];
        wm36 = pwms[36];
        wm37 = pwms[37];
        wm38 = pwms[38];

        #10 wait(ready_for_input);

        for(i=0; i<`NUM_INPUTS; i=i+1)
            begin
                `ifdef DEBUG
                    $display("\n-----");
                    $display("INPUT #%0d (phone_t = %d)", i,topster.wordMatcher.phone_t);
                `endif
                in0 = inputs[i*39];
                in1 = inputs[i*39+1];
                in2 = inputs[i*39+2];
                in3 = inputs[i*39+3];
                in4 = inputs[i*39+4];
                in5 = inputs[i*39+5];
            end
    end

```

```

in6 = inputs[i*39+6];
in7 = inputs[i*39+7];
in8 = inputs[i*39+8];
in9 = inputs[i*39+9];
in10 = inputs[i*39+10];
in11 = inputs[i*39+11];
in12 = inputs[i*39+12];
in13 = inputs[i*39+13];
in14 = inputs[i*39+14];
in15 = inputs[i*39+15];
in16 = inputs[i*39+16];
in17 = inputs[i*39+17];
in18 = inputs[i*39+18];
in19 = inputs[i*39+19];
in20 = inputs[i*39+20];
in21 = inputs[i*39+21];
in22 = inputs[i*39+22];
in23 = inputs[i*39+23];
in24 = inputs[i*39+24];
in25 = inputs[i*39+25];
in26 = inputs[i*39+26];
in27 = inputs[i*39+27];
in28 = inputs[i*39+28];
in29 = inputs[i*39+29];
in30 = inputs[i*39+30];
in31 = inputs[i*39+31];
in32 = inputs[i*39+32];
in33 = inputs[i*39+33];
in34 = inputs[i*39+34];
in35 = inputs[i*39+35];
in36 = inputs[i*39+36];
in37 = inputs[i*39+37];
in38 = inputs[i*39+38];

latchIn = 1;
#20
latchIn = 0;

#100 wait(ready_for_input);

`ifdef DEBUG
//print out top scores/indicies
$display("\nTOP INDICIES/SCORES\n#0\t#1\t#2\t#3\t#4\t#5");
$display("%d/%d\t%d/%d\t%d/%d\t%d/%d\t%d/%d",
        topster.phoneIndex0,topster.phoneScore0,
        topster.phoneIndex1,topster.phoneScore1,
        topster.phoneIndex2,topster.phoneScore2,
        topster.phoneIndex3,topster.phoneScore3,
        topster.phoneIndex4,topster.phoneScore4,
        topster.phoneIndex5,topster.phoneScore5);
//print out word matcher table
$display("\n\nWORD MATCHER
TABLE\nIdx\tWord\tScore\tStart\tStop\tStretch\tLast Phone\tStatus");
for (j=0;j<topster.wordMatcher.wm_length;j=j+1)
begin
    wm_row = wordMatchesScratchpad.Register[j];
    if (wm_row[1:0]!=3) //only show if it's still "alive"
        $display("%0d:\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d",
                j,
                wm_row[52:45],
                wm_row[44:21],
                wm_row[20:16],
                wm_row[15:11],
                wm_row[10:8],
                wm_row[7:2],
                wm_row[1:0]);
end
`endif
end
`ifdef DEBUG
    $display("\nEND OF INPUT...INITIATING MANUAL OVERRIDE!!!");
`endif

#10 manual_override = 1;
#20 manual_override = 0;

```

```

    //at this point, it computes the best trigram and passes it to the ASCII module
    #10 wait (ready_for_input);
    $display("\n");
    #100
    $finish;
    end
endmodule // testTop1

```

## phoneScorer.v

```

/*
 * Phone Scorer and Sorter
 *
 * Authors: Frankie Myers (fbmyers) and Eric Phillips (ecphilli)
 * ECE 520 Project - Group 21
 *
 * Calculates phone score vector for all 39 phones
 * Input: 39 weight matrices (72 bits/matrix) and 39 input vectors (24 bits/phone)
 * Output: Sorted 16-bit phone scores (and indices) and a 1-bit signal telling the
WordMatcher that the scores are ready
 */

//a value that doesn't correspond to an actual phone index. this is used
//so that the phone scorer won't block any inputs. As top scores are found,
//this value is replaced by actual indicies which correspond to the top scores
//that have already been identified, so they won't get identified again.
`define INVALID_IDX 6'b111111

`define CLEAR_REGS 0
`define GET_SCORE1 1
`define GET_SCORE2 2
`define GET_SCORE3 3
`define GET_SCORE4 4
`define GET_SCORE5 5
`define GET_SCORE6 6
`define SCORES_READY 7

module phoneScoreSort(clock, reset, latchIn, scoreReady, index0, index1, index2, index3,
index4, index5, wm0, wm1, wm2, wm3, wm4, wm5, wm6, wm7, wm8, wm9, wm10, wm11, wm12, wm13,
wm14, wm15, wm16, wm17, wm18,wm19, wm20, wm21, wm22, wm23, wm24, wm25, wm26, wm27, wm28,
wm29, wm30, wm31, wm32, wm33, wm34, wm35, wm36, wm37, wm38, in0, in1, in2, in3, in4, in5,
in6, in7, in8, in9, in10, in11, in12, in13, in14, in15, in16, in17, in18, in19, in20,
in21, in22, in23, in24, in25, in26, in27, in28, in29, in30, in31, in32, in33, in34, in35,
in36, in37, in38, score0, score1, score2, score3, score4, score5);

    input clock;
    input reset;

    //input phone weight matrices
    input [71:0]wm0, wm1, wm2, wm3, wm4, wm5, wm6, wm7, wm8, wm9, wm10, wm11, wm12, wm13,
wm14, wm15, wm16, wm17, wm18,wm19, wm20, wm21, wm22, wm23, wm24, wm25, wm26, wm27, wm28,
wm29, wm30, wm31, wm32, wm33, wm34, wm35, wm36, wm37, wm38;

    //39 3-value inputs constituting an acoustic vector
    input [23:0] in0, in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13,
in14, in15, in16, in17, in18, in19, in20, in21, in22, in23, in24, in25, in26, in27, in28,
in29, in30, in31, in32, in33, in34, in35, in36, in37, in38;
    input latchIn;
    output [19:0] score0, score1, score2, score3, score4, score5;
    output [5:0] index0, index1, index2, index3, index4, index5;
    output scoreReady;

    reg scoreReady;
    reg [3:0] next_state;
    reg [5:0] index0, index1, index2, index3, index4, index5;
    reg [19:0] score0, score1, score2, score3, score4, score5;
    wire [5:0] topIndex;
    wire [19:0] topScore;
    wire [19:0] ps0, ps1, ps2, ps3, ps4, ps5, ps6, ps7, ps8, ps9, ps10, ps11, ps12, ps13,
ps14, ps15, ps16, ps17, ps18, ps19, ps20, ps21, ps22, ps23, ps24, ps25, ps26, ps27, ps28,
ps29, ps30, ps31, ps32, ps33, ps34, ps35, ps36, ps37, ps38;

    // Instantiate 39 phone scorer module copies
    // computes the score for input in parallel
    scorer phone0(wm0, in0, ps0);
    scorer phone1(wm1, in1, ps1);
    scorer phone2(wm2, in2, ps2);
    scorer phone3(wm3, in3, ps3);

```

```

scorer phone4(wm4, in4, ps4);
scorer phone5(wm5, in5, ps5);
scorer phone6(wm6, in6, ps6);
scorer phone7(wm7, in7, ps7);
scorer phone8(wm8, in8, ps8);
scorer phone9(wm9, in9, ps9);
scorer phone10(wm10, in10, ps10);
scorer phone11(wm11, in11, ps11);
scorer phone12(wm12, in12, ps12);
scorer phone13(wm13, in13, ps13);
scorer phone14(wm14, in14, ps14);
scorer phone15(wm15, in15, ps15);
scorer phone16(wm16, in16, ps16);
scorer phone17(wm17, in17, ps17);
scorer phone18(wm18, in18, ps18);
scorer phone19(wm19, in19, ps19);
scorer phone20(wm20, in20, ps20);
scorer phone21(wm21, in21, ps21);
scorer phone22(wm22, in22, ps22);
scorer phone23(wm23, in23, ps23);
scorer phone24(wm24, in24, ps24);
scorer phone25(wm25, in25, ps25);
scorer phone26(wm26, in26, ps26);
scorer phone27(wm27, in27, ps27);
scorer phone28(wm28, in28, ps28);
scorer phone29(wm29, in29, ps29);
scorer phone30(wm30, in30, ps30);
scorer phone31(wm31, in31, ps31);
scorer phone32(wm32, in32, ps32);
scorer phone33(wm33, in33, ps33);
scorer phone34(wm34, in34, ps34);
scorer phone35(wm35, in35, ps35);
scorer phone36(wm36, in36, ps36);
scorer phone37(wm37, in37, ps37);
scorer phone38(wm38, in38, ps38);

//one sorter is used to find all 6 scores
//the top-scoring index value from each iteration is fed back
//into this combinational algorithm, blocking it and ensuring
//that it doesn't get picked again. In this way, the 6 top
//scores are identified
sorter sortPhones(index0, index1, index2, index3, index4, ps0, ps1, ps2, ps3, ps4, ps5,
ps6, ps7, ps8, ps9, ps10, ps11, ps12, ps13, ps14, ps15, ps16, ps17, ps18, ps19, ps20,
ps21, ps22, ps23, ps24, ps25, ps26, ps27, ps28, ps29, ps30, ps31, ps32, ps33, ps34, ps35,
ps36, ps37, ps38, topScore, topIndex);

// FSM for moving top scores and indices into registers (score0/index0,
score1/index1...)
always@(posedge clock or negedge reset)
begin
if(!reset)
begin
next_state <= `CLEAR_REGS;
end
else
begin
case(next_state)
`CLEAR_REGS:
begin
scoreReady <= 0;
if (latchIn)
begin
index0 <= `INVALID_IDX;
index1 <= `INVALID_IDX;
index2 <= `INVALID_IDX;
index3 <= `INVALID_IDX;
index4 <= `INVALID_IDX;
index5 <= `INVALID_IDX;
next_state <= `GET_SCORE1;
end
end

`GET_SCORE1: begin
index0 <= topIndex;
score0 <= topScore;
next_state <= `GET_SCORE2;
end
end

```

```

        `GET_SCORE2: begin
            index1 <= topIndex;
            score1 <= topScore;
            next_state <= `GET_SCORE3;
        end

        `GET_SCORE3: begin
            index2 <= topIndex;
            score2 <= topScore;
            next_state <= `GET_SCORE4;
        end

        `GET_SCORE4: begin
            index3 <= topIndex;
            score3 <= topScore;
            next_state <= `GET_SCORE5;
        end

        `GET_SCORE5: begin
            index4 <= topIndex;
            score4 <= topScore;
            next_state <= `GET_SCORE6;
        end

        `GET_SCORE6: begin
            index5 <= topIndex;
            score5 <= topScore;
            next_state <= `SCORES_READY;
        end

        `SCORES_READY: begin
            scoreReady <= 1;
            next_state <= `CLEAR_REGS;
        end

        default: next_state <= `CLEAR_REGS;
    endcase
end
end

endmodule // phoneScorer

/*
 * Scorer module
 *
 * Figures 3 and 4 in ProjectPlan
 * Input: Weight Matrix (72 bits) and an input vector (24 bits)
 * Output: Phone Score (16 bits)
 */
module scorer(weightMatrix, inScore, phoneScore);
    input [71:0] weightMatrix;
    input [23:0] inScore;
    output [19:0] phoneScore;

    wire [15:0] Aresult, Bresult, Cresult, Dresult, Eresult, Fresult, Gresult, Hresult,
    Iresult;

    // Vector multiplication calculations
    assign Aresult = weightMatrix[71:64] * inScore[23:16];
    assign Bresult = weightMatrix[63:56] * inScore[15:8];
    assign Cresult = weightMatrix[55:48] * inScore[7:0];
    assign Dresult = weightMatrix[47:40] * inScore[23:16];
    assign Eresult = weightMatrix[39:32] * inScore[15:8];
    assign Fresult = weightMatrix[31:24] * inScore[7:0];
    assign Gresult = weightMatrix[23:16] * inScore[23:16];
    assign Hresult = weightMatrix[15:8] * inScore[15:8];
    assign Iresult = weightMatrix[7:0] * inScore[7:0];

    assign phoneScore = Aresult + Bresult + Cresult
        + Dresult + Eresult + Fresult
        + Gresult + Hresult + Iresult;
endmodule // scorer

/*

```

```

* Sorter module
*
* Sorts out all 39 phones to find top 6
* Input: 39 phone scores (22 bits/phone score), indicies to be blocked (those that were
previously picked) and an input vector (24 bits)
* Output: Phone Score (19 bits)
*/
module sorter(block0, block1, block2, block3, block4, ps0, ps1, ps2, ps3, ps4, ps5, ps6,
ps7, ps8, ps9, ps10, ps11, ps12, ps13, ps14, ps15, ps16, ps17, ps18, ps19, ps20, ps21,
ps22, ps23, ps24, ps25, ps26, ps27, ps28, ps29, ps30, ps31, ps32, ps33, ps34, ps35, ps36,
ps37, ps38, topScore, topIndex);
    input [5:0] block0, block1, block2, block3, block4;
    input [19:0] ps0, ps1, ps2, ps3, ps4, ps5, ps6, ps7, ps8, ps9, ps10, ps11, ps12, ps13,
ps14, ps15, ps16, ps17, ps18, ps19, ps20, ps21, ps22, ps23, ps24, ps25, ps26, ps27, ps28,
ps29, ps30, ps31, ps32, ps33, ps34, ps35, ps36, ps37, ps38;
    output [19:0] topScore;
    output [5:0] topIndex;

    wire [19:0] nps0, nps1, nps2, nps3, nps4, nps5, nps6, nps7, nps8, nps9, nps10, nps11,
nps12, nps13, nps14, nps15, nps16, nps17, nps18, nps19, nps20, nps21, nps22, nps23,
nps24, nps25, nps26, nps27, nps28, nps29, nps30, nps31, nps32, nps33, nps34, nps35,
nps36, nps37, nps38;
    reg [37:0] tree_path;
    reg [5:0] topIndex;
    reg [19:0] topScore;

    // Muxes to assign ps to system or 0 (i.e. that # has already been found as a top score
    assign nps0 =
((block0==0) || (block1==0) || (block1==0) || (block2==0) || (block3==0) || (block4==0)) ? 16'h0000
: ps0;
    assign nps1 =
((block0==1) || (block1==1) || (block1==1) || (block2==1) || (block3==1) || (block4==1)) ? 16'h0000
: ps1;
    assign nps2 =
((block0==2) || (block1==2) || (block1==2) || (block2==2) || (block3==2) || (block4==2)) ? 16'h0000
: ps2;
    assign nps3 =
((block0==3) || (block1==3) || (block1==3) || (block2==3) || (block3==3) || (block4==3)) ? 16'h0000
: ps3;
    assign nps4 =
((block0==4) || (block1==4) || (block1==4) || (block2==4) || (block3==4) || (block4==4)) ? 16'h0000
: ps4;
    assign nps5 =
((block0==5) || (block1==5) || (block1==5) || (block2==5) || (block3==5) || (block4==5)) ? 16'h0000
: ps5;
    assign nps6 =
((block0==6) || (block1==6) || (block1==6) || (block2==6) || (block3==6) || (block4==6)) ? 16'h0000
: ps6;
    assign nps7 =
((block0==7) || (block1==7) || (block1==7) || (block2==7) || (block3==7) || (block4==7)) ? 16'h0000
: ps7;
    assign nps8 =
((block0==8) || (block1==8) || (block1==8) || (block2==8) || (block3==8) || (block4==8)) ? 16'h0000
: ps8;
    assign nps9 =
((block0==9) || (block1==9) || (block1==9) || (block2==9) || (block3==9) || (block4==9)) ? 16'h0000
: ps9;
    assign nps10 =
((block0==10) || (block1==10) || (block1==10) || (block2==10) || (block3==10) || (block4==10)) ?
16'h0000 : ps10;
    assign nps11 =
((block0==11) || (block1==11) || (block1==11) || (block2==11) || (block3==11) || (block4==11)) ?
16'h0000 : ps11;
    assign nps12 =
((block0==12) || (block1==12) || (block1==12) || (block2==12) || (block3==12) || (block4==12)) ?
16'h0000 : ps12;
    assign nps13 =
((block0==13) || (block1==13) || (block1==13) || (block2==13) || (block3==13) || (block4==13)) ?
16'h0000 : ps13;
    assign nps14 =
((block0==14) || (block1==14) || (block1==14) || (block2==14) || (block3==14) || (block4==14)) ?
16'h0000 : ps14;
    assign nps15 =
((block0==15) || (block1==15) || (block1==15) || (block2==15) || (block3==15) || (block4==15)) ?
16'h0000 : ps15;

```

```

    assign nps16 =
((block0==16) || (block1==16) || (block1==16) || (block2==16) || (block3==16) || (block4==16)) ?
16'h0000 : ps16;
    assign nps17 =
((block0==17) || (block1==17) || (block1==17) || (block2==17) || (block3==17) || (block4==17)) ?
16'h0000 : ps17;
    assign nps18 =
((block0==18) || (block1==18) || (block1==18) || (block2==18) || (block3==18) || (block4==18)) ?
16'h0000 : ps18;
    assign nps19 =
((block0==19) || (block1==19) || (block1==19) || (block2==19) || (block3==19) || (block4==19)) ?
16'h0000 : ps19;
    assign nps20 =
((block0==20) || (block1==20) || (block1==20) || (block2==20) || (block3==20) || (block4==20)) ?
16'h0000 : ps20;
    assign nps21 =
((block0==21) || (block1==21) || (block1==21) || (block2==21) || (block3==21) || (block4==21)) ?
16'h0000 : ps21;
    assign nps22 =
((block0==22) || (block1==22) || (block1==22) || (block2==22) || (block3==22) || (block4==22)) ?
16'h0000 : ps22;
    assign nps23 =
((block0==23) || (block1==23) || (block1==23) || (block2==23) || (block3==23) || (block4==23)) ?
16'h0000 : ps23;
    assign nps24 =
((block0==24) || (block1==24) || (block1==24) || (block2==24) || (block3==24) || (block4==24)) ?
16'h0000 : ps24;
    assign nps25 =
((block0==25) || (block1==25) || (block1==25) || (block2==25) || (block3==25) || (block4==25)) ?
16'h0000 : ps25;
    assign nps26 =
((block0==26) || (block1==26) || (block1==26) || (block2==26) || (block3==26) || (block4==26)) ?
16'h0000 : ps26;
    assign nps27 =
((block0==27) || (block1==27) || (block1==27) || (block2==27) || (block3==27) || (block4==27)) ?
16'h0000 : ps27;
    assign nps28 =
((block0==28) || (block1==28) || (block1==28) || (block2==28) || (block3==28) || (block4==28)) ?
16'h0000 : ps28;
    assign nps29 =
((block0==29) || (block1==29) || (block1==29) || (block2==29) || (block3==29) || (block4==29)) ?
16'h0000 : ps29;
    assign nps30 =
((block0==30) || (block1==30) || (block1==30) || (block2==30) || (block3==30) || (block4==30)) ?
16'h0000 : ps30;
    assign nps31 =
((block0==31) || (block1==31) || (block1==31) || (block2==31) || (block3==31) || (block4==31)) ?
16'h0000 : ps31;
    assign nps32 =
((block0==32) || (block1==32) || (block1==32) || (block2==32) || (block3==32) || (block4==32)) ?
16'h0000 : ps32;
    assign nps33 =
((block0==33) || (block1==33) || (block1==33) || (block2==33) || (block3==33) || (block4==33)) ?
16'h0000 : ps33;
    assign nps34 =
((block0==34) || (block1==34) || (block1==34) || (block2==34) || (block3==34) || (block4==34)) ?
16'h0000 : ps34;
    assign nps35 =
((block0==35) || (block1==35) || (block1==35) || (block2==35) || (block3==35) || (block4==35)) ?
16'h0000 : ps35;
    assign nps36 =
((block0==36) || (block1==36) || (block1==36) || (block2==36) || (block3==36) || (block4==36)) ?
16'h0000 : ps36;
    assign nps37 =
((block0==37) || (block1==37) || (block1==37) || (block2==37) || (block3==37) || (block4==37)) ?
16'h0000 : ps37;
    assign nps38 =
((block0==38) || (block1==38) || (block1==38) || (block2==38) || (block3==38) || (block4==38)) ?
16'h0000 : ps38;

    // modified binary search tree algorithm
    always@(nps0 or nps1 or nps2 or nps3 or nps4 or nps5 or nps6 or nps7 or nps8 or nps9 or
nps10 or nps11 or nps12 or nps13 or nps14 or nps15 or nps16 or nps17 or nps18 or nps19 or
nps20 or nps21 or nps22 or nps23 or nps24 or nps25 or nps26 or nps27 or nps28 or nps29 or
nps30 or nps31 or nps32 or nps33 or nps34 or nps35 or nps36 or nps37 or nps38)
    begin
        tree_path[0]=(nps0>nps1);

```

```

tree_path[1]=(nps2>nps3);
tree_path[2]=(nps4>nps5);
tree_path[3]=(nps6>nps7);
tree_path[4]=(nps8>nps9);
tree_path[5]=(nps10>nps11);
tree_path[6]=(nps12>nps13);
tree_path[7]=(nps14>nps15);
tree_path[8]=(nps16>nps17);
tree_path[9]=(nps18>nps19);
tree_path[10]=(nps20>nps21);
tree_path[11]=(nps22>nps23);
tree_path[12]=(nps24>nps25);
tree_path[13]=(nps26>nps27);
tree_path[14]=(nps28>nps29);
tree_path[15]=(nps30>nps31);
tree_path[16]=(nps32>nps33);
tree_path[17]=(nps34>nps35);
tree_path[18]=(nps36>nps37);

tree_path[19]=(tree_path[0]?nps0:nps1)>(tree_path[1]?nps2:nps3);
tree_path[20]=(tree_path[2]?nps4:nps5)>(tree_path[3]?nps6:nps7);
tree_path[21]=(tree_path[4]?nps8:nps9)>(tree_path[5]?nps10:nps11);
tree_path[22]=(tree_path[6]?nps12:nps13)>(tree_path[7]?nps14:nps15);
tree_path[23]=(tree_path[8]?nps16:nps17)>(tree_path[9]?nps18:nps19);
tree_path[24]=(tree_path[10]?nps20:nps21)>(tree_path[11]?nps22:nps23);
tree_path[25]=(tree_path[12]?nps24:nps25)>(tree_path[13]?nps26:nps27);
tree_path[26]=(tree_path[14]?nps28:nps29)>(tree_path[15]?nps30:nps31);
tree_path[27]=(tree_path[16]?nps32:nps33)>(tree_path[17]?nps34:nps35);
tree_path[28]=(tree_path[18]?nps36:nps37)>nps38;

tree_path[29]=((tree_path[19]?((tree_path[0]?nps0:nps1):(tree_path[1]?nps2:nps3))>(tree_path[20]?((tree_path[2]?nps4:nps5):(tree_path[3]?nps6:nps7)))));

tree_path[30]=((tree_path[21]?((tree_path[4]?nps8:nps9):(tree_path[5]?nps10:nps11))>(tree_path[22]?((tree_path[6]?nps12:nps13):(tree_path[7]?nps14:nps15)))));

tree_path[31]=((tree_path[23]?((tree_path[8]?nps16:nps17):(tree_path[9]?nps18:nps19))>(tree_path[24]?((tree_path[10]?nps20:nps21):(tree_path[11]?nps22:nps23)))));

tree_path[32]=((tree_path[25]?((tree_path[12]?nps24:nps25):(tree_path[13]?nps26:nps27))>(tree_path[26]?((tree_path[14]?nps28:nps29):(tree_path[15]?nps30:nps31)))));

tree_path[33]=(tree_path[29]?((tree_path[19]?((tree_path[0]?nps0:nps1):(tree_path[1]?nps2:nps3)):(tree_path[20]?((tree_path[2]?nps4:nps5):(tree_path[3]?nps6:nps7))))>(tree_path[30]?((tree_path[21]?((tree_path[4]?nps8:nps9):(tree_path[5]?nps10:nps11)):(tree_path[22]?((tree_path[6]?nps12:nps13):(tree_path[7]?nps14:nps15))))));

tree_path[34]=(tree_path[31]?((tree_path[23]?((tree_path[8]?nps16:nps17):(tree_path[9]?nps18:nps19)):(tree_path[24]?((tree_path[10]?nps20:nps21):(tree_path[11]?nps22:nps23))))>(tree_path[32]?((tree_path[25]?((tree_path[12]?nps24:nps25):(tree_path[13]?nps26:nps27)):(tree_path[26]?((tree_path[14]?nps28:nps29):(tree_path[15]?nps30:nps31))))));

tree_path[35]=(tree_path[33]?((tree_path[29]?((tree_path[19]?((tree_path[0]?nps0:nps1):(tree_path[1]?nps2:nps3)):(tree_path[20]?((tree_path[2]?nps4:nps5):(tree_path[3]?nps6:nps7)))):(tree_path[30]?((tree_path[21]?((tree_path[4]?nps8:nps9):(tree_path[5]?nps10:nps11)):(tree_path[22]?((tree_path[6]?nps12:nps13):(tree_path[7]?nps14:nps15))))))>(tree_path[34]?((tree_path[23]?((tree_path[8]?nps16:nps17):(tree_path[9]?nps18:nps19)):(tree_path[24]?((tree_path[10]?nps20:nps21):(tree_path[11]?nps22:nps23)))):(tree_path[32]?((tree_path[25]?((tree_path[12]?nps24:nps25):(tree_path[13]?nps26:nps27)):(tree_path[26]?((tree_path[14]?nps28:nps29):(tree_path[15]?nps30:nps31))))));

tree_path[36]=(tree_path[27]?((tree_path[16]?nps32:nps33):(tree_path[17]?nps34:nps35)) > (tree_path[28]?((tree_path[18]?nps36:nps37):nps38));

tree_path[37]=(tree_path[35]?((tree_path[33]?((tree_path[29]?((tree_path[19]?((tree_path[0]?nps0:nps1):(tree_path[1]?nps2:nps3)):(tree_path[20]?((tree_path[2]?nps4:nps5):(tree_path[3]?nps6:nps7)))):(tree_path[30]?((tree_path[21]?((tree_path[4]?nps8:nps9):(tree_path[5]?nps10:nps11)):(tree_path[22]?((tree_path[6]?nps12:nps13):(tree_path[7]?nps14:nps15)))))):(tree_path[34]?((tree_path[23]?((tree_path[8]?nps16:nps17):(tree_path[9]?nps18:nps19)):(tree_path[24]?((tree_path[10]?nps20:nps21):(tree_path[11]?nps22:nps23)))):(tree_path[32]?((tree_path[25]?((tree_path[12]?nps24:nps25):(tree_path[13]?nps26:nps27)):(tree_path[26]?((tree_path[14]?nps28:nps29):(tree_path[15]?nps30:nps31))))))>(tree_path[36]?((tree_path[27]?

```

```
(tree_path[16]?nps32:nps33):(tree_path[17]?nps34:nps35):(tree_path[28]?nps36:nps37):nps38);
```

```
topScore=tree_path[37]?(tree_path[35]?(tree_path[33]?(tree_path[29]?nps0:nps1):(tree_path[1]?nps2:nps3):(tree_path[20]?nps4:nps5):(tree_path[3]?nps6:nps7))):(tree_path[30]?nps8:nps9):(tree_path[5]?nps10:nps11):(tree_path[22]?nps12:nps13):(tree_path[7]?nps14:nps15)):(tree_path[34]?nps16:nps17):(tree_path[23]?nps18:nps19):(tree_path[8]?nps20:nps21):(tree_path[11]?nps22:nps23)):(tree_path[32]?nps24:nps25):(tree_path[13]?nps26:nps27):(tree_path[26]?nps28:nps29):(tree_path[15]?nps30:nps31)):(tree_path[36]?nps32:nps33):(tree_path[17]?nps34:nps35):(tree_path[28]?nps36:nps37):nps38);
```

```
if(topScore==nps0)
  topIndex <= 0;
else if(topScore==nps1)
  topIndex <= 1;
else if(topScore==nps2)
  topIndex <= 2;
else if(topScore==nps3)
  topIndex <= 3;
else if(topScore==nps4)
  topIndex <= 4;
else if(topScore==nps5)
  topIndex <= 5;
else if(topScore==nps6)
  topIndex <= 6;
else if(topScore==nps7)
  topIndex <= 7;
else if(topScore==nps8)
  topIndex <= 8;
else if(topScore==nps9)
  topIndex <= 9;
else if(topScore==nps10)
  topIndex <= 10;
else if(topScore==nps11)
  topIndex <= 11;
else if(topScore==nps12)
  topIndex <= 12;
else if(topScore==nps13)
  topIndex <= 13;
else if(topScore==nps14)
  topIndex <= 14;
else if(topScore==nps15)
  topIndex <= 15;
else if(topScore==nps16)
  topIndex <= 16;
else if(topScore==nps17)
  topIndex <= 17;
else if(topScore==nps18)
  topIndex <= 18;
else if(topScore==nps19)
  topIndex <= 19;
else if(topScore==nps20)
  topIndex <= 20;
else if(topScore==nps21)
  topIndex <= 21;
else if(topScore==nps22)
  topIndex <= 22;
else if(topScore==nps23)
  topIndex <= 23;
else if(topScore==nps24)
  topIndex <= 24;
else if(topScore==nps25)
  topIndex <= 25;
else if(topScore==nps26)
  topIndex <= 26;
else if(topScore==nps27)
  topIndex <= 27;
else if(topScore==nps28)
  topIndex <= 28;
else if(topScore==nps29)
  topIndex <= 29;
else if(topScore==nps30)
  topIndex <= 30;
else if(topScore==nps31)
```

```

        topIndex <= 31;
    else if(topScore==nps32)
        topIndex <= 32;
    else if(topScore==nps33)
        topIndex <= 33;
    else if(topScore==nps34)
        topIndex <= 34;
    else if(topScore==nps35)
        topIndex <= 35;
    else if(topScore==nps36)
        topIndex <= 36;
    else if(topScore==nps37)
        topIndex <= 37;
    else
        //((topScore==nps38)
        topIndex <= 38;
end

endmodule // sorter

```

## testScorer.v

```

module testScorer;
    reg [71:0] weightMatrix;
    reg [23:0] inScore;
    wire [15:0] phoneScore;
    scorer phoneScoreTest(weightMatrix, inScore, phoneScore);
    initial
    begin
        weightMatrix = 72'h0C04060201030A0907;
        inScore = 24'h060803;
        #10 $display("Phone score 304: %d", phoneScore);
        #10 inScore = 24'h000000;
        #10 $display("Phone score 0: %d", phoneScore);
        #10 weightMatrix = 72'h080A09070E01000102;
        #10 $display("Phone score 0: %d", phoneScore);
        #10 inScore = 24'h070100;
        #10 $display("Phone score 130: %d", phoneScore);
        #10 inScore = 24'h0C0006;
        #10 $display("Phone score 252: %d", phoneScore);
        #10 weightMatrix = 72'h000000000000000000;
        inScore = 24'h060803;
        #10 $display("Phone score 0: %d", phoneScore);
        #10 $finish;
    end
endmodule // testScorer

```

## testSort.v

```

module testSort;
    reg clock;
    reg latchIn;
    reg [5:0] block0, block1, block2, block3, block4;
    reg [15:0] ps0, ps1, ps2, ps3, ps4, ps5, ps6, ps7, ps8, ps9, ps10, ps11, ps12, ps13,
    ps14, ps15, ps16, ps17, ps18, ps19, ps20, ps21, ps22, ps23, ps24, ps25, ps26, ps27, ps28,
    ps29, ps30, ps31, ps32, ps33, ps34, ps35, ps36, ps37, ps38;
    wire [15:0] outScore;
    wire [5:0] outIndex;
    wire latchOut;

    sorter sortTest(clock, latchIn, latchOut, block0, block1, block2, block3, block4, ps0,
    ps1, ps2, ps3, ps4, ps5, ps6, ps7, ps8, ps9, ps10, ps11, ps12, ps13, ps14, ps15, ps16,
    ps17, ps18, ps19, ps20, ps21, ps22, ps23, ps24, ps25, ps26, ps27, ps28, ps29, ps30, ps31,
    ps32, ps33, ps34, ps35, ps36, ps37, ps38, outScore, outIndex);
    always #5 clock = ~clock;
    initial
    begin
        $shm_open("waves.shm"); // save shm data base to mem.sav
        $shm_probe ("AS"); // save all signals in the hierarchy, A = save all top level
signals
        #0 clock = 0;
        latchIn = 0;
        block0 = 63; block1 = 63; block2 = 63; block3 = 63; block4 = 63;
        ps0 = 0; ps1 = 0; ps2 = 100; ps3 = 0; ps4 = 0; ps5 = 0; ps6 = 0; ps7 = 0; ps8 = 0;
        ps9 = 0; ps10 = 0; ps11 = 0; ps12 = 0; ps13 = 0; ps14 = 0; ps15 = 0; ps16 = 0;
        ps17 = 0; ps18 = 0; ps19 = 0; ps20 = 0; ps21 = 0; ps22 = 0; ps23 = 0; ps24 = 0;
        ps25 = 0; ps26 = 0; ps27 = 0; ps28 = 0; ps29 = 0; ps30 = 0; ps31 = 0; ps32 = 0;
        ps33 = 0; ps34 = 0; ps35 = 0; ps36 = 0; ps37 = 0; ps38 = 0;
        #15 latchIn = 1;
    end
endmodule

```

```

#10 $display("Top score (100) : %d", outScore);
    $display("Top index (2) : %d\n", outIndex);
#10 latchIn = 0;
    ps0 = 0; ps1 = 0; ps2 = 100; ps3 = 0; ps4 = 0; ps5 = 0; ps6 = 0; ps7 = 0; ps8 =
98;
    ps9 = 15; ps10 = 0; ps11 = 32; ps12 = 0; ps13 = 101; ps14 = 0; ps15 = 0; ps16 = 0;
    ps17 = 0; ps18 = 0; ps19 = 0; ps20 = 0; ps21 = 0; ps22 = 0; ps23 = 0; ps24 = 0;
    ps25 = 0; ps26 = 0; ps27 = 0; ps28 = 0; ps29 = 0; ps30 = 8; ps31 = 7; ps32 = 21;
    ps33 = 47; ps34 = 821; ps35 = 0; ps36 = 0; ps37 = 0; ps38 = 0;
#20 latchIn = 1;
#10 $display("Top score (821) : %d", outScore);
    $display("Top index (34) : %d\n", outIndex);
#10 latchIn = 0;
    ps0 = 0; ps1 = 0; ps2 = 100; ps3 = 0; ps4 = 0; ps5 = 0; ps6 = 0; ps7 = 0; ps8 =
0;
    ps9 = 0; ps10 = 0; ps11 = 0; ps12 = 0; ps13 = 0; ps14 = 0; ps15 = 0; ps16 = 0;
    ps17 = 0; ps18 = 0; ps19 = 0; ps20 = 0; ps21 = 0; ps22 = 0; ps23 = 0; ps24 = 72;
    ps25 = 100; ps26 = 91; ps27 = 17; ps28 = 85; ps29 = 521; ps30 = 3; ps31 = 7; ps32 = 1;
    ps33 = 1; ps34 = 0; ps35 = 0; ps36 = 0; ps37 = 0; ps38 = 0;
#20 latchIn = 1;
#10 $display("Top score (521) : %d", outScore);
    $display("Top index (29) : %d\n", outIndex);
#10 latchIn = 0;
    ps0 = 1; ps1 = 2; ps2 = 6; ps3 = 6; ps4 = 8; ps5 = 0; ps6 = 0; ps7 = 0; ps8 = 0;
    ps9 = 0; ps10 = 0; ps11 = 0; ps12 = 0; ps13 = 0; ps14 = 0; ps15 = 0; ps16 = 0;
    ps17 = 0; ps18 = 0; ps19 = 0; ps20 = 0; ps21 = 0; ps22 = 0; ps23 = 0; ps24 = 0;
    ps25 = 0; ps26 = 0; ps27 = 0; ps28 = 0; ps29 = 0; ps30 = 0; ps31 = 0; ps32 = 0;
    ps33 = 0; ps34 = 7; ps35 = 5; ps36 = 7; ps37 = 10; ps38 = 81;
#20 latchIn = 1;
#10 $display("Top score (81) : %d", outScore);
    $display("Top index (38) : %d", outIndex);
#10 $finish;
end

endmodule // testSort

```

## testScoreSort.v

```

module testScoreSort;
    reg clock;
    reg reset;
    reg [71:0]wm0, wm1, wm2, wm3, wm4, wm5, wm6, wm7, wm8, wm9, wm10, wm11, wm12, wm13,
    wm14, wm15, wm16, wm17, wm18, wm19, wm20, wm21, wm22, wm23, wm24, wm25, wm26, wm27, wm28,
    wm29, wm30, wm31, wm32, wm33, wm34, wm35, wm36, wm37, wm38;
    reg [23:0] in0, in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13,
    in14, in15, in16, in17, in18, in19, in20, in21, in22, in23, in24, in25, in26, in27, in28,
    in29, in30, in31, in32, in33, in34, in35, in36, in37, in38;
    reg latchIn;
    wire [15:0] s0, s1, s2, s3, s4, s5;
    wire [5:0] i0, i1, i2, i3, i4, i5;
    wire scoreReady;

    phoneScoreSort scoreSort(clock, reset, latchIn, scoreReady, i0, i1, i2, i3, i4, i5,
    wm0, wm1, wm2, wm3, wm4, wm5, wm6, wm7, wm8, wm9, wm10, wm11, wm12, wm13, wm14, wm15,
    wm16, wm17, wm18, wm19, wm20, wm21, wm22, wm23, wm24, wm25, wm26, wm27, wm28, wm29, wm30,
    wm31, wm32, wm33, wm34, wm35, wm36, wm37, wm38, in0, in1, in2, in3, in4, in5, in6, in7,
    in8, in9, in10, in11, in12, in13, in14, in15, in16, in17, in18, in19, in20, in21, in22,
    in23, in24, in25, in26, in27, in28, in29, in30, in31, in32, in33, in34, in35, in36, in37,
    in38, s0, s1, s2, s3, s4, s5);

    always #5 clock = ~clock;
    initial
    begin
        $dumpfile("scorer.vcd");
        $dumpvars;
        #0 clock = 0;
        reset = 0;
        #10 reset = 1;
        latchIn = 1'b0;
        wm0 = 72'h080100010503020100; wm1 = 72'h0; wm2 = 72'h0; wm3 = 72'h0; wm4 = 72'h0;
        wm5 = 72'h0; wm6 = 72'h0; wm7 = 72'h090701000707010004; wm8 = 72'h020501010308010009;
        wm9 = 72'h0; wm10 = 72'h040103000103010102; wm11 = 72'h0; wm12 = 72'h0;
        wm13 = 72'h0; wm14 = 72'h0; wm15 = 72'h0; wm16 = 72'h0;
        wm17 = 72'h0; wm18 = 72'h0; wm19 = 72'h0; wm20 = 72'h0;
        wm21 = 72'h0; wm22 = 72'h0; wm23 = 72'h0; wm24 = 72'h0;
        wm25 = 72'h0; wm26 = 72'h020801050301070907; wm27 = 72'h0; wm28 =
72'h010909070102010302;
        wm29 = 72'h010701020301010809; wm30 = 72'h0; wm31 = 72'h0; wm32 = 72'h0;

```



```

*
* After the algorithm receives 30 inputs (consisting of 6 phones each) OR after the
* manual_override input is raised, whichever comes first, the algorithm begins searching
* for the highest scoring valid trigram (that is, a valid 3-word sequence where there are
* no gaps between the words and the first word starts at the first input iteration, t=0.
* When the top trigram is found, the 3 word index values are output and the done output
* is raised.
*
* The module then shifts the contents of the word matcher table to remove the entries which
* started before the last phone that was resolved into the trigram. That is, all the words
* (or potential words) that appeared within the time span of the trigram are eliminated,
* and the start_t/stop_t values of the remaining word entries are also subtracted to reflect
* this shift. In effect, the input iteration directly after the last iteration that was
* resolved becomes t=0. Then the process starts over again.
*/

/*
* This module uses its own FSM to handle the different
* processing stages. These are the state ID's
*/
`define SHIFT_WM_TABLE1          1
`define SHIFT_WM_TABLE2          2
`define SHIFT_WM_TABLE3          3
`define SCAN_WORD_MATCHES1        4
`define SCAN_WORD_MATCHES2        5
`define SCAN_WORD_MATCHES3        6
`define ADD_STRETCHED_WORD        7
`define SCAN_FULL_DICT            8
`define WAITING_FOR_SCORES        9
`define LOOK_FOR_WORD1           10
`define LOOK_FOR_WORD2           11
`define LOOK_FOR_WORD3           12

/*
* Used in the word matcher table to indicate the status
* of a potential word match
*/
`define PARTIAL                   1
`define MATCH                     2
`define INVALID                   3

/*
* special character that indicates the end of a Phone word
* This is placed at the end of each entry in the phone
* dictionary by the preprocessor program
*/
`define NULL                       6'b111111

//number of words in the dictionary
`define DICT_LENGTH 134

//number of inputs to read before performing trigram matching
`define NUM_PHONES_TO_LOAD 30

module word_matcher(score0,score1,score2,score3,score4,score5,
                   index0,index1,index2,index3,index4,index5,

                   best_trigram_index1,best_trigram_index2,best_trigram_index3,
                   ready_for_input,done,scores_ready>manual_override,
                   wm_we,wm_waddr,wm_raddr,wm_wdata,wm_rdata,
                   phdict_raddr,phdict_rdata,
                   clock,reset);

    //input top-scoring phones (and their scores) from the phoneScorer module
    input [19:0] score0,score1,score2,score3,score4,score5;
    input [5:0]  index0,index1,index2,index3,index4,index5;

    //set this high when the next scores are ready be loaded in (1 clock cycle)
    input          scores_ready;

    //set this high to force the module to begin trigram matching (on fewer than 30
phones)
    input          manual_override;
    input          clock, reset;

    //goes high when the module has found the trigram

```

```

output          done;

//high when the module is ready for new scores to be loaded in
//phoneScorer should poll this and only put new scores on the bus when it is high
output          ready_for_input;

//the three words which constitute the trigram that has been identified
output [7:0]    best_trigram_index1, best_trigram_index2, best_trigram_index3;

//word match table sram I/O lines
output          wm_we;
output [9:0]    wm_waddr, wm_raddr;
output [52:0]   wm_wdata;
input  [52:0]   wm_rdata;

//phone dictionary sram I/O
output [11:0]   phdict_raddr;
input  [5:0]    phdict_rdata;

reg            done;
reg            ready_for_input;

reg [7:0]      best_trigram_index1, best_trigram_index2, best_trigram_index3;
reg [27:0]    best_trigram_score;
reg [4:0]     best_trigram_length;

//temporary values stored while searching for best trigram
reg [23:0]    word1_score, word2_score;
reg [7:0]     word1_index, word2_index;
reg [4:0]     word1_stop_t, word2_stop_t;
reg [9:0]     word1_wm_raddr, word2_wm_raddr;

//internal FSM variable
reg [3:0]     state;

//counting/index registers for this module
reg [9:0]     wm_length;
reg [4:0]     phone_t;
reg          word_completed;
reg [7:0]     wm_shift_loc;

//only used to compute values for a new "stretched word" row when it is added to the
table
reg          stretching_found; //debug signal
reg [5:0]    stretched_last_phone;
reg [23:0]   stretched_score;
reg [6:0]    num_stretched_rows_added;

//sram variables...more convenient to split each row into its constituent values
reg [9:0]    wm_raddr, wm_waddr;
reg          wm_we;
reg [7:0]    wm_wdata_index;
reg [23:0]   wm_wdata_score;
reg [4:0]    wm_wdata_start_t;
reg [4:0]    wm_wdata_stop_t;
reg [2:0]    wm_wdata_num_stretched;
reg [5:0]    wm_wdata_last_phone;
reg [1:0]    wm_wdata_status;

wire [7:0]   wm_rdata_index;
wire [23:0]  wm_rdata_score;
wire [4:0]   wm_rdata_start_t;
wire [4:0]   wm_rdata_stop_t;
wire [2:0]   wm_rdata_num_stretched;
wire [5:0]   wm_rdata_last_phone;
wire [1:0]   wm_rdata_status;

assign
{wm_rdata_index,wm_rdata_score,wm_rdata_start_t,wm_rdata_stop_t,wm_rdata_num_stretched,wm
_rdata_last_phone,wm_rdata_status} = wm_rdata;
assign wm_wdata =
{wm_wdata_index,wm_wdata_score,wm_wdata_start_t,wm_wdata_stop_t,wm_wdata_num_stretched,wm
_wdata_last_phone,wm_wdata_status};

//phone dictionary address upper bits correspond to the word, while the lower bits
correspond to the phone index
reg [7:0]    phdict_word_index;

```

```

reg [3:0] phdict_phone_index;
assign phdict_raddr = {phdict_word_index,phdict_phone_index};

always@(posedge clock or negedge reset)
begin
  if(!reset)
  begin //handle reset condition
    phone_t = 0;
    word_completed = 1; //this causes a full dictionary search on the first
set of inputs
    wm_length = 0;
    done = 0;
    wm_we = 0;
    wm_waddr = 0;
    wm_raddr = 0;
    ready_for_input = 1;
    state = `WAITING_FOR_SCORES;
  end
else
  begin
    casex (state)
      `SHIFT_WM_TABLE1: //iterates through the WM table until the shift
location is found
        begin
          if (wm_rdata_start_t>=best_trigram_length)
            begin
              wm_shift_loc = wm_raddr;
              state = `SHIFT_WM_TABLE2;
            end
          else
            begin
              if (wm_raddr>=wm_length)
                begin
                  wm_shift_loc = wm_raddr;
                  state = `SHIFT_WM_TABLE2;
                end
              else
                wm_raddr = wm_raddr + 1;
            end
          done = 0;
        end
      `SHIFT_WM_TABLE2: //actually iterates through the rows and copies
over the ones to be thrown out
        if (wm_raddr<wm_length)
          begin
            wm_wdata_index = wm_rdata_index;
            wm_wdata_score = wm_rdata_score;
            wm_wdata_start_t = wm_rdata_start_t-best_trigram_length;
            wm_wdata_stop_t = wm_rdata_stop_t-best_trigram_length;
            wm_wdata_last_phone = wm_rdata_last_phone;
            wm_wdata_num_stretched = wm_rdata_num_stretched;
            wm_wdata_status = wm_rdata_status;
            wm_we = 1;
            state = `SHIFT_WM_TABLE3;
          end
        else
          begin
            wm_raddr = 0;
            wm_waddr = 0;
            wm_we = 0;
            wm_length = wm_shift_loc;
            phone_t = phone_t - best_trigram_length +1;
            ready_for_input = 1;
            state = `WAITING_FOR_SCORES;
          end
      `SHIFT_WM_TABLE3: //increment address pointers
        begin
          wm_we = 0;
          wm_raddr = wm_raddr + 1;
          wm_waddr = wm_waddr + 1;
          state = `SHIFT_WM_TABLE2;
        end
      `SCAN_WORD_MATCHES1: //load in the corresponding phone from the
dictionary based on the current word index entry in WM
        begin
          phdict_word_index = wm_rdata_index;

```

```

        phdict_phone_index = phone_t - wm_rdata_start_t -
wm_rdata_num_stretched; //check to see if the next phone in the word equals any of the
top 6

        state = `SCAN_WORD_MATCHES2;
        end

`SCAN_WORD_MATCHES2: //update the WM table based on whether the
current entry should survive or not
        if (wm_raddr<(wm_length-num_stretched_rows_added))
            begin
//first, check if this is a partial match and if the current input iteration contains
//the next phone

                if ((phdict_rdata==index0 || phdict_rdata==index1 ||
                    phdict_rdata==index2 || phdict_rdata==index3 ||
                    phdict_rdata==index4 || phdict_rdata==index5) &&
                    wm_rdata_status==`PARTIAL)
                begin //the phone matched...add the corresponding
score value to it

                    if (phdict_rdata==index0)
                        begin
                            wm_wdata_score = wm_rdata_score + score0;
                            wm_wdata_last_phone = index0;
                        end
                    if (phdict_rdata==index1)
                        begin
                            wm_wdata_score = wm_rdata_score + score1;
                            wm_wdata_last_phone = index1;
                        end
                    if (phdict_rdata==index2)
                        begin
                            wm_wdata_score = wm_rdata_score + score2;
                            wm_wdata_last_phone = index2;
                        end
                    if (phdict_rdata==index3)
                        begin
                            wm_wdata_score = wm_rdata_score + score3;
                            wm_wdata_last_phone = index3;
                        end
                    if (phdict_rdata==index4)
                        begin
                            wm_wdata_score = wm_rdata_score + score4;
                            wm_wdata_last_phone = index4;
                        end
                    if (phdict_rdata==index5)
                        begin
                            wm_wdata_score = wm_rdata_score + score5;
                            wm_wdata_last_phone = index5;
                        end
                    wm_wdata_status = `PARTIAL;
                    wm_wdata_stop_t = wm_rdata_stop_t;
                end
                else if (phdict_rdata==`NULL && wm_rdata_status==`PARTIAL)
                begin //we reached the end of a word without it
getting coded as INVALID (i.e. it survived)
                    wm_wdata_status = `MATCH;
                    wm_wdata_score = wm_rdata_score;
                    wm_wdata_stop_t = phone_t; //we don't discover it
actually ended until we encounter the NULL char 1 iteration later (so this represents the
start time of the NEXT word)

                    wm_wdata_last_phone = wm_rdata_last_phone;
                    word_completed = 1;
                end
                else if (wm_rdata_status==`MATCH) //this entry has already
been matched, skip over it

                    begin
                        wm_wdata_status = `MATCH;
                        wm_wdata_score = wm_rdata_score;
                        wm_wdata_stop_t = wm_rdata_stop_t;
                        wm_wdata_last_phone = wm_rdata_last_phone;
                    end
                else //the word is invalid
                    begin
                        wm_wdata_status = `INVALID;
                        wm_wdata_score = wm_rdata_score;
                        wm_wdata_stop_t = wm_rdata_stop_t;
                        wm_wdata_last_phone = wm_rdata_last_phone;
                    end
                end
            end
        end

```

```

//see if the phone is stretched. i.e. one of the top scores equals the last phone for
// this word (stored in the WM table)
    if ((wm_rdata_last_phone==index0 || wm_rdata_last_phone==index1 ||
        wm_rdata_last_phone==index2 || wm_rdata_last_phone==index3 ||
        wm_rdata_last_phone==index4 || wm_rdata_last_phone==index5) &&
        wm_rdata_status=='PARTIAL')
        begin
            if (wm_rdata_last_phone==index0)
                stretched_score = wm_rdata_score + score0;
            if (wm_rdata_last_phone==index1)
                stretched_score = wm_rdata_score + score1;
            if (wm_rdata_last_phone==index2)
                stretched_score = wm_rdata_score + score2;
            if (wm_rdata_last_phone==index3)
                stretched_score = wm_rdata_score + score3;
            if (wm_rdata_last_phone==index4)
                stretched_score = wm_rdata_score + score4;
            if (wm_rdata_last_phone==index5)
                stretched_score = wm_rdata_score + score5;
            stretched_last_phone = wm_rdata_last_phone;
            stretching_found = 1;
            state = `ADD_STRETCHED_WORD; //goto extra state to
add this stretched word possibility to the table
        end
    else
        state = `SCAN_WORD_MATCHES3;
        wm_wdata_num_stretched = wm_rdata_num_stretched;
        wm_wdata_index = wm_rdata_index;
        wm_wdata_start_t = wm_rdata_start_t;
        wm_we = 1;
    end

    else
        begin //we have reached the end of the WM table, see if we
should do a full dict search b/c there might be a new word
        if (word_completed)
        begin //at least one word completed, so look for new words
starting directly after it
            word_completed = 0;
            phdict_word_index = 0;
            phdict_phone_index = 0;
            wm_raddr = wm_length; //jump to end of table
            wm_waddr = wm_length;
            state = `SCAN_FULL_DICT;
            end
        else
        begin
            if (phone_t=='NUM_PHONES_TO_LOAD-1)
                begin //end of the input set, look for a trigram
                    word1_wm_raddr = 0;
                    wm_raddr = word1_wm_raddr;
                    best_trigram_score = 0;
                    state = `LOOK_FOR_WORD1;
                end
            else //wait for next input iteration
                begin
                    ready_for_input = 1;
                    state = `WAITING_FOR_SCORES;
                end
            phone_t = phone_t + 1;
        end
    end

    end

`SCAN_WORD_MATCHES3: //lowers write signal and gets next WM row for
next iteration
        begin
            wm_we = 0;
            wm_raddr = wm_raddr + 1;
            wm_waddr = wm_raddr;
            state = `SCAN_WORD_MATCHES1;
        end

        `ADD_STRETCHED_WORD: //appends a new row for the stretched word
possibility, then resumes the SCAN_WORD_MATCHES loop
        begin

```

```

        wm_waddr = wm_length; //move the write pointer to
the end of the table while the read pointer remains at the row to be copied
        wm_wdata_index = wm_rdata_index;
        wm_wdata_start_t = wm_rdata_start_t;
        wm_wdata_stop_t = wm_rdata_stop_t;
        wm_wdata_last_phone = stretched_last_phone;
        wm_wdata_score = stretched_score;
        wm_wdata_status = `PARTIAL;
        wm_wdata_num_stretched = wm_rdata_num_stretched + 1;
        wm_we = 1;
        wm_length = wm_length + 1;
        num_stretched_rows_added = num_stretched_rows_added + 1;
        stretching_found = 0;
        state = `SCAN_WORD_MATCHES3;
    end
    `SCAN_FULL_DICT: //search the entire dictionary for words which
might start at this iteration (assumes phdict_raddr has been set to 0)
    if (wm_we)
        begin
            wm_we = 0;
            wm_raddr = wm_raddr + 1;
            wm_waddr = wm_waddr + 1;
        end
    else
        if (phdict_word_index < `DICT_LENGTH)
            begin
                if (phdict_rdata==index0 || phdict_rdata==index1 ||
                    phdict_rdata==index2 || phdict_rdata==index3 ||
                    phdict_rdata==index4 || phdict_rdata==index5)
                begin //add this potential word to the WM table
                    if (phdict_rdata==index0)
                        wm_wdata_score = score0;
                    if (phdict_rdata==index1)
                        wm_wdata_score = score1;
                    if (phdict_rdata==index2)
                        wm_wdata_score = score2;
                    if (phdict_rdata==index3)
                        wm_wdata_score = score3;
                    if (phdict_rdata==index4)
                        wm_wdata_score = score4;
                    if (phdict_rdata==index5)
                        wm_wdata_score = score5;
                    wm_wdata_index = phdict_word_index;
                    wm_wdata_start_t = phone_t;
                    wm_wdata_num_stretched = 0;
                    wm_wdata_last_phone = phdict_rdata;
                    wm_wdata_status = `PARTIAL;
                    wm_we = 1;
                    wm_length = wm_length + 1;
                end
                phdict_word_index = phdict_word_index + 1;
            end
        //skip to the next word in the dictionary, look at the first char
        phdict_phone_index = 0;
        end
    else
        begin
            if (phone_t==`NUM_PHONES_TO_LOAD-1)
            begin //end of the input set, look for a trigram
                word1_wm_raddr = 0;
                wm_raddr = word1_wm_raddr;
                best_trigram_score = 0;
                state = `LOOK_FOR_WORD1;
            end
        else
            begin //wait for next input iteration
                phone_t = phone_t + 1;
                ready_for_input = 1;
                state = `WAITING_FOR_SCORES;
            end
        end

        `LOOK_FOR_WORD1: //begin trigram matching, look for first word
(assumes word1_wm_raddr==0 AND that this has been latched into the read address port
wm_raddr)
        if (word1_wm_raddr < wm_length)
            begin //in bounds
                if (wm_rdata_status==`MATCH && wm_rdata_start_t==0)

```

```

begin //now look for a candidate 2nd word
    word1_index = wm_rdata_index;
    word1_stop_t = wm_rdata_stop_t; //we must
store these in internal registers since we need them later and we're moving to a
different SRAM address

    word1_score = wm_rdata_score;
    word2_wm_raddr = 0;
    wm_raddr = word2_wm_raddr;
    state = `LOOK_FOR_WORD2;

end

else
begin
    word1_wm_raddr = word1_wm_raddr + 1;
    wm_raddr = word1_wm_raddr;
end

end

else //end of the table has been reached, so trigram has
been found

begin
//set done flag high so ASCII module can start
//and also begin shifting the contents of the table
wm_raddr = 0;
wm_waddr = 0;
state = `SHIFT_WM_TABLE1;
done = 1;
end

`LOOK_FOR_WORD2: //look for second word
if (word2_wm_raddr < wm_length)
begin //in bounds
if (wm_rdata_status == `MATCH && wm_rdata_start_t == word1_stop_t)
begin //now look for 3rd word
    word2_index = wm_rdata_index;
    word2_stop_t = wm_rdata_stop_t; //we must
store these in internal registers since we need them later and we're moving to a
different SRAM address

    word2_score = wm_rdata_score;
    wm_raddr = 0; //get ready to do the 3rd
search, starting at top again

    state = `LOOK_FOR_WORD3;
end
end //skip to next entry
begin
    word2_wm_raddr = word2_wm_raddr + 1;
    wm_raddr = word2_wm_raddr;
end

end

else //traverse back up the recursive "stack"
begin
    word1_wm_raddr = word1_wm_raddr + 1;
    wm_raddr = word1_wm_raddr;
    state = `LOOK_FOR_WORD1;
end

end

`LOOK_FOR_WORD3: //look for third word
if (wm_raddr < wm_length)
begin
begin
if (wm_rdata_status == `MATCH && wm_rdata_start_t == word2_stop_t &&
(word1_score + word2_score + wm_rdata_score) > best_trigram_score)
begin //the trigram we're currently
looking at is better than the current high score, so replace
    best_trigram_score = word1_score + word2_score + wm_rdata_score;
    best_trigram_index1 = word1_index;
    best_trigram_index2 = word2_index;
    best_trigram_index3 = wm_rdata_index;
    best_trigram_length = wm_rdata_stop_t;
end
    wm_raddr = wm_raddr + 1;
end
end //traverse back up the recursive "stack"
begin
    word2_wm_raddr = word2_wm_raddr + 1;
    wm_raddr = word2_wm_raddr;
    state = `LOOK_FOR_WORD2;
end

end

```

```

        `WAITING_FOR_SCORES: //waiting for the phoneScorer module to place
new scores on the input bus
        begin
            if (scores_ready==1)
                begin
                    ready_for_input = 0;
                    wm_raddr = 0;
                    wm_waddr = 0;
                    num_stretched_rows_added = 0;
                    state = `SCAN_WORD_MATCHES1;
                end
            if (manual_override)
                begin
                    ready_for_input = 0;
                    word1_wm_raddr = 0;
                    wm_raddr = word1_wm_raddr;
                    best_trigram_score = 0;
                    state = `LOOK_FOR_WORD1;
                end
            end
        default:
            begin
                ready_for_input = 0;
                state = `WAITING_FOR_SCORES;
            end
        end
    endcase
end
endmodule

```

## phdict\_sram.v

```

module phdict_sram (clock, WE, WriteAddress, ReadAddress, WriteBus, ReadBus);
input  clock, WE;
input  [11:0] WriteAddress, ReadAddress;
input  [5:0] WriteBus;
output [5:0] ReadBus;

reg [5:0]  Register [0:2144]; // 64 5-bit words

// provide one write enable line per register
wire [2144:0] WELines;
integer i;

// Write '1' into write enable line for selected register
assign WELines = (WE << WriteAddress);

always@(posedge clock)
    for (i=0; i<=2144; i=i+1)
        if (WELines[i]) Register[i] <= WriteBus;

assign ReadBus = Register[ReadAddress];
endmodule

```

## wm\_sram.v

```

module wm_sram (clock, WE, WriteAddress, ReadAddress, WriteBus, ReadBus);
input  clock, WE;
input  [9:0] WriteAddress, ReadAddress;
input  [52:0] WriteBus;
output [52:0] ReadBus;

reg [52:0]  Register [0:1023]; // 1024 53-bit words

// provide one write enable line per register
wire [1023:0] WELines;
integer i;

// Write '1' into write enable line for selected register
assign WELines = (WE << WriteAddress);

always@(posedge clock)
    for (i=0; i<=1023; i=i+1)
        if (WELines[i]) Register[i] <= WriteBus;

assign ReadBus = Register[ReadAddress];

```

```
endmodule
```

## ascii.v

```
/*
 * ASCII Interface
 *
 * Authors: Frankie Myers (fbmyers) and Eric Phillips (ecphilli)
 * ECE 520 Project - Group 21
 *
 * Receive dictionary index number from WordMatcher and output ASCII data from Dictionary
 2 to output prompt
 * Input: Clock, wordsFound control signal, and 3 index values associated with a word at
 that SRAM row address
 * Output: ASCII characters to command prompt
 */

`define S0 0
`define S1 1
`define S2 2
`define S3 3

module ascii(clock, reset, wordsFound, index0, index1, index2, readAddr, readBus,
asciiOut);
    input clock;
    input reset;
    input wordsFound;
    input [7:0] index0, index1, index2;          // 8-bits so you can address all 134
"valid" words
    input [7:0] readBus;                        // 8 bits (i.e. ASCII value) from SRAM
    output [11:0] readAddr;                    // Address to send to SRAM to read at
    output [7:0] asciiOut;                     // 8 bits for hex

    reg [2:0] next_state;
    reg [7:0] asciiOut;
    reg [6:0] start, stop;
    reg [11:0] readAddr;

    always@(posedge clock or negedge reset)
        begin
            if(!reset)
                begin
                    next_state <= `S0;
                end
            else
                begin
                    case(next_state)
                        `S0: begin
                            if(wordsFound)
                                begin
                                    readAddr <= {index0, 4'b0000};
                                    next_state <= `S1;
                                end
                            else
                                begin
                                    next_state <= `S0;
                                end
                            end
                        `S1: begin
                            if(readBus == 8'h20)
                                begin
                                    asciiOut <= readBus;
                                    readAddr <= {index1, 4'b0000};
                                    next_state <= `S2;
                                end
                            else
                                begin
                                    asciiOut <= readBus;
                                    readAddr <= readAddr + 1'b1;
                                    next_state <= `S1;
                                end
                            end
                        `S2: begin
                            if(readBus == 8'h20)
                                begin
                                    asciiOut <= readBus;
                                end
                            end
                    endcase
                end
        end
endmodule
```

```

        readAddr <= {index2, 4'b0000};
        next_state <= `S3;
    end
else
    begin
        asciiOut <= readBus;
        readAddr <= readAddr + 1'b1;
        next_state <= `S2;
    end
end

`S3: begin
    if(readBus == 8'h20)
        begin
            asciiOut <= 8'h20;
            next_state <= `S0;
        end
    else
        begin
            readAddr <= readAddr + 1'b1;
            asciiOut <= readBus;
            next_state <= `S3;
        end
    end

default: begin
    next_state <= `S0;
end
endcase
end
end

endmodule // ascii;

```

## sramASCII.v

```

module sramASCII(clock, WE, WriteAddress, ReadAddress, WriteBus, ReadBus);
    input clock, WE;
    input [11:0] WriteAddress, ReadAddress;
    input [7:0] WriteBus;
    output [7:0] ReadBus;

    reg [7:0] Register [0:2143]; // 2,144 8-bit words

    // provide one write enable line per register
    wire [2143:0] WElines;
    integer i;

    // Write '1' into write enable line for selected register
    assign WElines = (WE << WriteAddress);

    always@(posedge clock)
        for (i=0; i<=2143; i=i+1)
            if (WElines[i]) Register[i] <= WriteBus;

    assign ReadBus = Register[ReadAddress];

endmodule

```

## testASCII.v

```

`include "sramASCII.v"
module testASCII;
    reg clock;
    reg reset;
    reg wordsFound;
    reg [7:0] index0, index1, index2;
    reg [11:0] writeAddr;
    reg [7:0] writeBus;
    reg WE;
    wire [7:0] readBus;
    wire [7:0] asciiOut;
    wire [11:0] readAddr;
    wire asciiDone;

    ascii asciiTester(clock, reset, wordsFound, index0, index1, index2, readAddr, readBus,
        asciiOut, asciiDone);
    sramASCII dictionary2(clock, WE, writeAddr, readAddr, writeBus, readBus);

```



## Appendix B: Synthesis scripts

### script1.sc

```
/*
 * Synthesis Script 1
 */
/*
 * Authors: Frankie Myers (fbmyers) and Eric Phillips (ecphilli)
 * ECE 520 Project - Group 21
 */

Read -f Verilog phoneScorer.v
Read -f Verilog word_matcher_3.v
Read -f Verilog ascii.v
Read -f Verilog top.v
current_design = top
```

### script2.sc

```
/*
 * Synthesis Script 2
 */
/*
 * Authors: Frankie Myers (fbmyers) and Eric Phillips (ecphilli)
 * ECE 520 Project - Group 21
 */

/*
 * Our first Optimization 'compile' is intended to
 * produce a design that will meet hold-time
 * under worst-case conditions:
 * - slowest process corner
 * - highest operating temperature and lowest Vcc
 * - expected worst case clock skew
 */

/*-----*/
/* Specify the worst case (slowest) libraries and
 * The library has not been characterised
 * for Operating conditions.
 */-----*/

target_library = {"ncsulib25_worst.db"}
link_library = {"ncsulib25_worst.db"}

/*-----*/
/* Specify a 5000 ps clock period with 50% duty cycle
 * and a skew of 300 ps
 */-----*/

Create_clock -period 60000 -waveform {0 30000} clock
set_clock_skew -uncertainty 300 clock

/*
 * Now set up the 'CONSTRAINTS' on the design:
 * 1. How much of the clock period is lost in the
 * modules connected to it
 * 2. What type of cells are driving the inputs
 * 3. What type of cells and how many (fanout) must it
 * be able to drive
 */

/*-----*/
/* ASSUME being driven by a slowest D-flip-flop
 * The DFF cell has a worst clock-Q delay of 900 ps
 * Allow another 200 ps for wiring delay
 * NOTE: THESE ARE INITIAL ASSUMPTIONS ONLY
 */-----*/

set_input_delay 1100 -clock clock all_inputs() - clock

/*-----*/
```

```

/* ASSUME this module is driving a D-flip-flip          */
/* The DFF cell has a worst set-up time of 750 ps      */
/* Allow another 200 ps for wiring delay               */
/* NOTE: THESE ARE INITIAL ASSUMPTIONS ONLY           */
/*-----*/

set_output_delay 950 -clock clock all_outputs()

/*-----*/
/* ASSUME being driven by a D-flip-flop               */
/*-----*/

set_driving_cell -cell "dp_2" -pin "q" all_inputs() - clock

/*-----*/
/* ASSUME the worst case output load is               */
/* 3 D-flip-flop (D-inputs) and                       */
/* and 0.5 units of wiring capacitance                 */
/*-----*/

port_load = 0.5 + 3 * load_of (ncsulib25_worst/dp_2/ip)
set_load port_load all_outputs()

/*****
/*
/* Now set the GOALS for the compile                  */
/*
/* In most cases you want minimum area, so set the   */
/* goal for maximum area to be 0                      */
/*
*****/

set_max_area 0

/*-----*/
/* During the initial map (synthesis), Synopsys might */
/* have built parts (such as adders) using its        */
/* DesignWare(TM) library. In order to remap the     */
/* design to our TSMC025 library AND to create scope */
/* for logic reduction, I want to 'flatten out' the  */
/* DesignWare components. i.e. Make one flat design  */
/*
/* 'replace_synthetic' is the cleanest way of doing this*/
/*-----*/

replace_synthetic -ungroup
uniquify

/*-----*/
/* check the design before optimization                */
/*-----*/
check_design
check_timing

/*****
/*
/* Now resynthesize the design to meet constraints,   */
/* and try to best achieve the goal, and using the    */
/* CMOSX parts. In large designs, compile can take   */
/* a llllooonnnnggg time                              */
*****/

/*-----*/
/* -map_effort specifies how much optimization effort  */
/* there is low, medium, and high                     */
/* use high to squeeze out those last picoseconds    */
/* -verify_effort specifies how much effort to spend  */
/* making sure that the input and output designs     */
/* are equivalent logically                            */
/*-----*/
compile -map_effort low

/*-----*/
/* Now trace the critical (slowest) path and see if  */
/* the timing works.                                  */
/*

```

```

/* If the slack is NOT met, you HAVE A PROBLEM and      */
/* need to redesign or try some other minimization    */
/* tricks that Synopsys can do                        */
/*-----*/
report_timing

```

### script3.sc

```

/*****
/* Synthesis Script 3                                */
/*
/* Authors: Frankie Myers (fbmyers) and Eric Phillips (ecphilli) */
/* ECE 520 Project - Group 21                          */
/*****

/*
/* This is your section to do different things to    */
/* improve timing or area - RTFM                      */
/*
/*-----*/

/*****
/*
/* Now resynthesize the design for the fastest corner */
/* making sure that hold time conditions are met      */
/*
/*-----*/

/*-----*/
/* Specify the fastest process corner and lowest temp */
/* And highest (fastest) Vcc                          */
/*-----*/

target_library = {"ncsulib25_best.db"}
link_library = {"ncsulib25_worst.db"}
translate

/*-----*/
/* Set the design rule to 'fix hold time violations'  */
/* Then compile the design again, telling Synopsys to  */
/* Only change the design if there are hold time     */
/* violations.                                         */
/*-----*/
set_fix_hold clock
compile -only_design_rule -incremental

/*-----*/
/* Report the fastest path. Make sure the hold       */
/* is actually met.                                   */
/*-----*/
report_timing -delay min

/*-----*/
/* Write out the 'fastest' (minimum) timing file     */
/* in Standard Delay Format. We might use this in later*/
/* verification.                                     */
/*-----*/
write_timing -output top_min.sdf -format sdf

```

### script4.sc

```

/*****
/* Synthesis Script 4                                */
/*
/* Authors: Frankie Myers (fbmyers) and Eric Phillips (ecphilli) */
/* ECE 520 Project - Group 21                          */
/*****

/*-----*/
/* Since Synopsys has to insert logic to meet hold   */
/* violations, we might find that we have setup      */
/* violations now. SO lets recheck with the slowest  */
/* corner etc.                                       */
/*
/* YOU have problems if the slack is NOT MET        */
/*
/* 'translate' means 'translate to new library'      */

```

```

/*-----*/
target_library = {"ncsulib25_worst.db"}
link_library = {"ncsulib25_worst.db"}
translate

report_timing

/*-----*/
/* Write out the resulting netlist in Verliog format */
/*-----*/
write -f verilog -o top_final.v

/*-----*/
/* Write out the resulting heirarchial netlist */
/* in Verliog format. We will need this */
/* for Silicon Ensemble */
/*-----*/

write -hierarchy -format verilog -output top_heirarchy.v

/*-----*/
/* Write out the 'slowest' (maximum) timing file */
/* in Standard Delay Format. We might use this in later*/
/* verification. */
/*-----*/
write_timing -output top_max.sdf -format sdf

/*-----*/
/* Report the area of the top module */
/*-----*/
current_design = top
report_area

```

## Appendix C: Extracts from view\_command.log

### script1.sc Read commands

```
Read -f Verilog phoneScorer.v
Loading Verilog file
'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/phoneScorer.v'
Running PRESTO HDLC
Loading db file '/afs/bp.ncsu.edu/dist/synopsys200108/libraries/syn/standard.sldb'
Loading db file '/afs/bp.ncsu.edu/dist/synopsys200108/libraries/syn/gtech.db'
Loading db file '/afs/eos.ncsu.edu/dist/cad446/local/TSMC025_deep/ncsulib25_worst.db'
Compiling source netlist file
/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/phoneScorer.v
```

```
Statistics for case statements in always block at line 93 in file
'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/phoneScorer.v'
```

```
=====
|           Line           | full/ parallel |
|-----|-----|
|           101           |    auto/auto   |
|-----|-----|
=====
```

```
Inferred memory devices in process
in routine phoneScoreSort line 93 in file
'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/phoneScorer.v'.
```

```
=====
| Register Name | Type | Width | Bus | MB | AR | AS | SR | SS | ST |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| score3_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| score3_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
| score4_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
| index3_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| score1_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| score1_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
| score5_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
| index2_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index0_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| score0_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
| score4_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| scoreReady_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index2_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| score5_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index1_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| index5_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| next_state_reg | Flip-flop | 4 | Y | N | Y | N | N | N | N |
| index0_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index4_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index5_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| score0_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| score2_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index4_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| index1_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index3_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| score2_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
```

```
Presto compilation completed successfully.
```

```
Current design is now
```

```
'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/phoneScoreSort.db:phoneScoreSort'
{"phoneScoreSort", "scorer", "sorter"}
```

```
Read -f Verilog word_matcher_3.v
```

```
Loading Verilog file
'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/word_matcher_3.v'
Running PRESTO HDLC
Compiling source netlist file
/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/word_matcher_3.v
```

```
Statistics for case statements in always block at line 108 in file
'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/word_matcher_3.v'
```

```
=====
|           Line           | full/ parallel |
|-----|-----|
|           124           |    auto/auto   |
|-----|-----|
=====
```

```
Inferred memory devices in process
in routine word_matcher line 108 in file
'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/word_matcher_3.v'.
```

```
=====
| Register Name | Type | Width | Bus | MB | AR | AS | SR | SS | ST |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| num_stretched_rows_added_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| num_stretched_rows_added_reg | Flip-flop | 6 | Y | N | N | N | N | N | N |
| state_reg | Flip-flop | 2 | Y | N | N | Y | N | N | N |
| word2_score_reg | Flip-flop | 23 | Y | N | N | N | N | N | N |
| word2_index_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| best_trigram_score_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
```

wm_wdata_score_reg	Flip-flop	23	Y	N	N	N	N	N	N	N
phdict_word_index_reg	Flip-flop	1	N	N	N	N	N	N	N	N
phdict_phone_index_reg	Flip-flop	3	Y	N	N	N	N	N	N	N
wm_wdata_last_phone_reg	Flip-flop	5	Y	N	N	N	N	N	N	N
phdict_phone_index_reg	Flip-flop	1	N	N	N	N	N	N	N	N
best_trigram_length_reg	Flip-flop	1	N	N	N	N	N	N	N	N
wm_wdata_status_reg	Flip-flop	1	N	N	N	N	N	N	N	N
wm_wdata_num_stretched_reg	Flip-flop	1	N	N	N	N	N	N	N	N
word2_stop_t_reg	Flip-flop	4	Y	N	N	N	N	N	N	N
word1_index_reg	Flip-flop	7	Y	N	N	N	N	N	N	N
wm_waddr_reg	Flip-flop	10	Y	N	Y	N	N	N	N	N
word1_stop_t_reg	Flip-flop	1	N	N	N	N	N	N	N	N
best_trigram_score_reg	Flip-flop	27	Y	N	N	N	N	N	N	N
best_trigram_index2_reg	Flip-flop	1	N	N	N	N	N	N	N	N
best_trigram_index3_reg	Flip-flop	7	Y	N	N	N	N	N	N	N
stretched_last_phone_reg	Flip-flop	5	Y	N	N	N	N	N	N	N
wm_wdata_start_t_reg	Flip-flop	1	N	N	N	N	N	N	N	N
word2_wm_raddr_reg	Flip-flop	1	N	N	N	N	N	N	N	N
wm_wdata_stop_t_reg	Flip-flop	1	N	N	N	N	N	N	N	N
ready_for_input_reg	Flip-flop	1	N	N	N	Y	N	N	N	N
word_completed_reg	Flip-flop	1	N	N	N	Y	N	N	N	N
wm_shift_loc_reg	Flip-flop	1	N	N	N	N	N	N	N	N
word1_wm_raddr_reg	Flip-flop	9	Y	N	N	N	N	N	N	N
best_trigram_index1_reg	Flip-flop	7	Y	N	N	N	N	N	N	N
word2_wm_raddr_reg	Flip-flop	9	Y	N	N	N	N	N	N	N
word1_score_reg	Flip-flop	1	N	N	N	N	N	N	N	N
stretched_score_reg	Flip-flop	23	Y	N	N	N	N	N	N	N
phdict_word_index_reg	Flip-flop	7	Y	N	N	N	N	N	N	N
best_trigram_length_reg	Flip-flop	4	Y	N	N	N	N	N	N	N
word2_score_reg	Flip-flop	1	N	N	N	N	N	N	N	N
wm_wdata_num_stretched_reg	Flip-flop	2	Y	N	N	N	N	N	N	N
best_trigram_index2_reg	Flip-flop	7	Y	N	N	N	N	N	N	N
word1_score_reg	Flip-flop	23	Y	N	N	N	N	N	N	N
best_trigram_index3_reg	Flip-flop	1	N	N	N	N	N	N	N	N
wm_we_reg	Flip-flop	1	N	N	Y	N	N	N	N	N
word1_stop_t_reg	Flip-flop	4	Y	N	N	N	N	N	N	N
stretched_score_reg	Flip-flop	1	N	N	N	N	N	N	N	N
wm_wdata_stop_t_reg	Flip-flop	4	Y	N	N	N	N	N	N	N
wm_wdata_index_reg	Flip-flop	1	N	N	N	N	N	N	N	N
done_reg	Flip-flop	1	N	N	Y	N	N	N	N	N
wm_length_reg	Flip-flop	10	Y	N	Y	N	N	N	N	N
wm_raddr_reg	Flip-flop	10	Y	N	Y	N	N	N	N	N
state_reg	Flip-flop	2	Y	N	Y	N	N	N	N	N
wm_wdata_status_reg	Flip-flop	1	N	N	N	N	N	N	N	N
word1_wm_raddr_reg	Flip-flop	1	N	N	N	N	N	N	N	N
best_trigram_index1_reg	Flip-flop	1	N	N	N	N	N	N	N	N
stretched_last_phone_reg	Flip-flop	1	N	N	N	N	N	N	N	N
wm_wdata_score_reg	Flip-flop	1	N	N	N	N	N	N	N	N
wm_wdata_last_phone_reg	Flip-flop	1	N	N	N	N	N	N	N	N
word1_index_reg	Flip-flop	1	N	N	N	N	N	N	N	N
phone_t_reg	Flip-flop	5	Y	N	Y	N	N	N	N	N
wm_wdata_index_reg	Flip-flop	7	Y	N	N	N	N	N	N	N
word2_stop_t_reg	Flip-flop	1	N	N	N	N	N	N	N	N
word2_index_reg	Flip-flop	7	Y	N	N	N	N	N	N	N
wm_shift_loc_reg	Flip-flop	7	Y	N	N	N	N	N	N	N
wm_wdata_start_t_reg	Flip-flop	4	Y	N	N	N	N	N	N	N

=====  
Presto compilation completed successfully.

Current design is now

'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire\_sys/word\_matcher.db:word\_matcher'  
{"word\_matcher"}

Read -f Verilog ascii.v

Loading Verilog file '/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire\_sys/ascii.v'

Running PRESTO HDLC

Compiling source netlist file

/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire\_sys/ascii.v

Statistics for case statements in always block at line 31 in file

'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire\_sys/ascii.v'

```

=====
| Line | full/ parallel |
|-----|-----|
| 39 | auto/auto |
=====

```

Inferred memory devices in process

in routine ascii line 31 in file

'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire\_sys/ascii.v'.

```

=====
| Register Name | Type | Width | Bus | MB | AR | AS | SR | SS | ST |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| asciiOut_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| next_state_reg | Flip-flop | 1 | N | N | Y | N | Y | N | N |
| asciiOut_reg | Flip-flop | 7 | Y | N | N | N | N | N | N |
| readAddr_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| next_state_reg | Flip-flop | 2 | Y | N | Y | N | N | N | N |
| readAddr_reg | Flip-flop | 11 | Y | N | N | N | N | N | N |
=====

```

Presto compilation completed successfully.

```

Current design is now
'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/ascii.db:ascii'
{"ascii"}
Read -f Verilog top.v
Loading Verilog file '/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/top.v'
Running PRESTO HDLC
Compiling source netlist file
'/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire_sys/top.v'

```

Statistics for case statements in always block at line 93 in file  
'phoneScorer.v'

```

=====
|           Line           | full/ parallel |
=====
|           101           |   auto/auto   |
=====

```

Inferred memory devices in process  
in routine phoneScoreSort line 93 in file  
'phoneScorer.v'.

```

=====
| Register Name | Type | Width | Bus | MB | AR | AS | SR | SS | ST |
=====
| score3_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| score3_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
| score4_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
| index0_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| index3_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| score1_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| score1_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
| score5_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
| score0_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
| score4_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| scoreReady_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index1_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index1_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| score5_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index5_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| next_state_reg | Flip-flop | 4 | Y | N | Y | N | N | N | N |
| index4_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index5_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| score0_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| score2_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index4_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| index0_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index2_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| index2_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| index3_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| score2_reg | Flip-flop | 19 | Y | N | N | N | N | N | N |
=====

```

Statistics for case statements in always block at line 108 in file  
'word\_matcher\_3.v'

```

=====
|           Line           | full/ parallel |
=====
|           124           |   auto/auto   |
=====

```

Inferred memory devices in process  
in routine word\_matcher line 108 in file  
'word\_matcher\_3.v'.

```

=====
| Register Name | Type | Width | Bus | MB | AR | AS | SR | SS | ST |
=====
| num_stretched_rows_added_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| num_stretched_rows_added_reg | Flip-flop | 6 | Y | N | N | N | N | N | N |
| state_reg | Flip-flop | 2 | Y | N | Y | N | N | N | N |
| word2_score_reg | Flip-flop | 23 | Y | N | N | N | N | N | N |
| wm_wdata_num_stretched_reg | Flip-flop | 2 | Y | N | N | N | N | N | N |
| word2_index_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| wm_wdata_score_reg | Flip-flop | 23 | Y | N | N | N | N | N | N |
| wm_raddr_reg | Flip-flop | 10 | Y | N | Y | N | N | N | N |
| word1_wm_raddr_reg | Flip-flop | 9 | Y | N | N | N | N | N | N |
| stretched_score_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| phdict_phone_index_reg | Flip-flop | 3 | Y | N | N | N | N | N | N |
| phdict_word_index_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| wm_wdata_start_t_reg | Flip-flop | 4 | Y | N | N | N | N | N | N |
| phdict_phone_index_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| best_trigram_length_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| wm_wdata_status_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| word2_stop_t_reg | Flip-flop | 4 | Y | N | N | N | N | N | N |
| word1_index_reg | Flip-flop | 7 | Y | N | N | N | N | N | N |
| word1_stop_t_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| best_trigram_index2_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| best_trigram_index3_reg | Flip-flop | 7 | Y | N | N | N | N | N | N |
| stretched_last_phone_reg | Flip-flop | 5 | Y | N | N | N | N | N | N |
| word2_wm_raddr_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
| wm_wdata_stop_t_reg | Flip-flop | 1 | N | N | N | N | N | N | N |
=====

```

ready_for_input_reg	Flip-flop	1	N	N	N	Y	N	N	N
word_completed_reg	Flip-flop	1	N	N	N	Y	N	N	N
wm_shift_loc_reg	Flip-flop	1	N	N	N	N	N	N	N
best_trigram_index1_reg	Flip-flop	7	Y	N	N	N	N	N	N
word2_wm_raddr_reg	Flip-flop	9	Y	N	N	N	N	N	N
phdict_word_index_reg	Flip-flop	7	Y	N	N	N	N	N	N
wm_wdata_last_phone_reg	Flip-flop	1	N	N	N	N	N	N	N
word1_score_reg	Flip-flop	1	N	N	N	N	N	N	N
best_trigram_length_reg	Flip-flop	4	Y	N	N	N	N	N	N
word2_score_reg	Flip-flop	1	N	N	N	N	N	N	N
word1_wm_raddr_reg	Flip-flop	1	N	N	N	N	N	N	N
wm_wdata_index_reg	Flip-flop	7	Y	N	N	N	N	N	N
best_trigram_index2_reg	Flip-flop	7	Y	N	N	N	N	N	N
word1_score_reg	Flip-flop	23	Y	N	N	N	N	N	N
best_trigram_score_reg	Flip-flop	1	N	N	N	N	N	N	N
best_trigram_index3_reg	Flip-flop	1	N	N	N	N	N	N	N
wm_wdata_start_t_reg	Flip-flop	1	N	N	N	N	N	N	N
wm_we_reg	Flip-flop	1	N	N	Y	N	N	N	N
wm_wdata_last_phone_reg	Flip-flop	5	Y	N	N	N	N	N	N
word1_stop_t_reg	Flip-flop	4	Y	N	N	N	N	N	N
phone_t_reg	Flip-flop	5	Y	N	Y	N	N	N	N
wm_wdata_stop_t_reg	Flip-flop	4	Y	N	N	N	N	N	N
wm_waddr_reg	Flip-flop	10	Y	N	Y	N	N	N	N
done_reg	Flip-flop	1	N	N	Y	N	N	N	N
state_reg	Flip-flop	2	Y	N	N	Y	N	N	N
wm_wdata_status_reg	Flip-flop	1	N	N	N	N	N	N	N
wm_wdata_num_stretched_reg	Flip-flop	1	N	N	N	N	N	N	N
best_trigram_index1_reg	Flip-flop	1	N	N	N	N	N	N	N
stretched_last_phone_reg	Flip-flop	1	N	N	N	N	N	N	N
stretched_score_reg	Flip-flop	23	Y	N	N	N	N	N	N
wm_wdata_index_reg	Flip-flop	1	N	N	N	N	N	N	N
word1_index_reg	Flip-flop	1	N	N	N	N	N	N	N
wm_shift_loc_reg	Flip-flop	7	Y	N	N	N	N	N	N
word2_stop_t_reg	Flip-flop	1	N	N	N	N	N	N	N
wm_length_reg	Flip-flop	10	Y	N	Y	N	N	N	N
wm_wdata_score_reg	Flip-flop	1	N	N	N	N	N	N	N
word2_index_reg	Flip-flop	7	Y	N	N	N	N	N	N
best_trigram_score_reg	Flip-flop	27	Y	N	N	N	N	N	N

Statistics for case statements in always block at line 31 in file  
'ascii.v'

Line	full/ parallel
39	auto/auto

Inferred memory devices in process  
in routine ascii line 31 in file  
'ascii.v'.

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
asciiOut_reg	Flip-flop	1	N	N	N	N	N	N	N
next_state_reg	Flip-flop	1	N	N	Y	N	Y	N	N
asciiOut_reg	Flip-flop	7	Y	N	N	N	N	N	N
readAddr_reg	Flip-flop	1	N	N	N	N	N	N	N
next_state_reg	Flip-flop	2	Y	N	Y	N	N	N	N
readAddr_reg	Flip-flop	11	Y	N	N	N	N	N	N

Presto compilation completed successfully.

Current design is now

"/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire\_sys/phoneScoreSort.db:phoneScoreSort"  
{ "phoneScoreSort", "scorer", "sorter", "word\_matcher", "ascii", "top" }

current\_design = top

Current design is 'top'.

"/afs/eos.ncsu.edu/lockers/workspace/ece/ece520/001/grp21/entire\_sys/top.db:top"

1

design\_analyzer> create\_schematic -size infinite -gen\_database

Loading db file '/afs/eos.ncsu.edu/dist/cad446/local/TSMC025\_deep/generic.sdb'

Loading db file '/afs/eos.ncsu.edu/dist/cad446/local/TSMC025\_deep/ncsulib25\_symbols.sdb'

Loading db file '/afs/eos.ncsu.edu/dist/cad446/local/TSMC025\_deep/basic.sdb'

Loading db file '/afs/bp.ncsu.edu/dist/synopsys200108/libraries/syn/1\_25.font'

## script2.sc clock setup

```
/*-----*/
/* Specify a 5000 ps clock period with 50% duty cycle */
/* and a skew of 300 ps */
/*-----*/
```

Create\_clock -period 60000 -waveform {0 30000} clock

Performing Create\_clock on port 'clock'.

1

set\_clock\_skew -uncertainty 300 clock

Performing set\_clock\_skew on clock 'clock'.

# script2.sc timing report

```
*****  
Report : timing  
        -path full  
        -delay max  
        -max_paths 1  
Design : top  
Version: 2001.08  
Date    : Sun Apr 16 22:47:47 2006  
*****
```

Operating Conditions:  
Wire Load Model Mode: top

Startpoint: in31[21] (input port clocked by clock)  
Endpoint: scoreSorter/index5\_reg[2]  
(rising edge-triggered flip-flop clocked by clock)  
Path Group: clock  
Path Type: max

Point	Incr	Path
clock clock (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
input external delay	1100.00	1100.00 r
in31[21] (in)	451.97	1551.97 r
scoreSorter/in31[21] (phoneScoreSort)	0.00	1551.97 r
scoreSorter/phone31/inScore[21] (scorer_7)	0.00	1551.97 r
scoreSorter/phone31/U2420/op (inv_2)	155.12	1707.10 f
scoreSorter/phone31/U256/op (nor2_2)	249.52	1956.62 r
scoreSorter/phone31/U1838/op (xor2_2)	262.56	2219.18 r
scoreSorter/phone31/U3305/op (xor2_2)	231.50	2450.67 r
scoreSorter/phone31/U1839/op (xor2_2)	233.57	2684.24 r
scoreSorter/phone31/U4968/op (inv_2)	115.97	2800.21 f
scoreSorter/phone31/U3306/op (xor2_2)	218.36	3018.58 r
scoreSorter/phone31/U1840/op (xor2_2)	235.53	3254.11 r
scoreSorter/phone31/U4969/op (inv_2)	112.85	3366.96 f
scoreSorter/phone31/U3307/op (xor2_2)	215.61	3582.58 r
scoreSorter/phone31/U1841/op (xor2_2)	235.53	3818.11 r
scoreSorter/phone31/U4970/op (inv_2)	83.82	3901.92 f
scoreSorter/phone31/U3308/op (xor2_2)	211.49	4113.41 r
scoreSorter/phone31/U1842/op (xor2_2)	235.54	4348.95 r
scoreSorter/phone31/U4971/op (inv_2)	112.85	4461.80 f
scoreSorter/phone31/U3309/op (xor2_2)	215.61	4677.41 r
scoreSorter/phone31/U1843/op (xor2_2)	231.77	4909.18 r
scoreSorter/phone31/U5035/op (inv_2)	123.52	5032.70 f
scoreSorter/phone31/U3310/op (xor2_2)	230.83	5263.53 r
scoreSorter/phone31/U1922/op (xor2_2)	231.60	5495.13 r
scoreSorter/phone31/U5076/op (inv_2)	112.35	5607.49 f
scoreSorter/phone31/U3316/op (xor2_2)	215.17	5822.66 r
scoreSorter/phone31/U1967/op (xor2_2)	235.36	6058.02 r
scoreSorter/phone31/U5130/op (inv_2)	110.28	6168.30 f
scoreSorter/phone31/U538/op (nor2_2)	157.86	6326.16 r
scoreSorter/phone31/U539/op (nor2_2)	75.81	6401.97 f
scoreSorter/phone31/U2634/op (or2_2)	252.63	6654.60 f
scoreSorter/phone31/U557/op (nor2_2)	148.77	6803.37 r
scoreSorter/phone31/U558/op (nor2_2)	94.09	6897.46 f
scoreSorter/phone31/U4333/op (or2_2)	242.44	7139.90 f
scoreSorter/phone31/U4334/op (inv_2)	71.38	7211.27 r
scoreSorter/phone31/U579/op (nor2_2)	82.78	7294.05 f
scoreSorter/phone31/U2636/op (or2_2)	252.63	7546.68 f
scoreSorter/phone31/U1407/op (nor2_2)	148.74	7695.43 r
scoreSorter/phone31/U1408/op (nor2_2)	93.17	7788.60 f
scoreSorter/phone31/U2982/op (or2_2)	259.75	8048.34 f
scoreSorter/phone31/U1410/op (nor2_2)	148.07	8196.41 r
scoreSorter/phone31/U1411/op (nor2_2)	93.68	8290.09 f
scoreSorter/phone31/U4723/op (or2_2)	242.06	8532.15 f
scoreSorter/phone31/U4724/op (inv_2)	71.37	8603.52 r
scoreSorter/phone31/U1415/op (nor2_2)	82.78	8686.30 f
scoreSorter/phone31/U3091/op (or2_2)	252.63	8938.93 f
scoreSorter/phone31/U1417/op (nor2_2)	148.43	9087.35 r
scoreSorter/phone31/U1418/op (nor2_2)	94.08	9181.44 f
scoreSorter/phone31/U3122/op (or2_2)	262.45	9443.89 f
scoreSorter/phone31/U1420/op (nor2_2)	148.60	9592.49 r
scoreSorter/phone31/U1421/op (nor2_2)	94.08	9686.58 f
scoreSorter/phone31/U4727/op (or2_2)	242.44	9929.01 f
scoreSorter/phone31/U4728/op (inv_2)	71.38	10000.39 r
scoreSorter/phone31/U1428/op (nor2_2)	82.78	10083.17 f
scoreSorter/phone31/U4733/op (or2_2)	232.65	10315.82 f
scoreSorter/phone31/U2303/op (xor2_2)	210.00	10525.82 r
scoreSorter/phone31/U5649/op (inv_2)	124.19	10650.01 f
scoreSorter/phone31/U3549/op (xor2_2)	231.06	10881.07 r
scoreSorter/phone31/U2304/op (xor2_2)	231.61	11112.68 r
scoreSorter/phone31/U5650/op (inv_2)	123.51	11236.19 f
scoreSorter/phone31/U3550/op (xor2_2)	230.83	11467.02 r
scoreSorter/phone31/U2327/op (xor2_2)	231.60	11698.62 r
scoreSorter/phone31/U5651/op (inv_2)	123.51	11822.13 f
scoreSorter/phone31/U3551/op (xor2_2)	230.83	12052.96 r

scoreSorter/phone31/U2349/op (xor2_2)	231.60	12284.56	r
scoreSorter/phone31/U5652/op (inv_2)	123.51	12408.07	f
scoreSorter/phone31/U3563/op (xor2_2)	230.83	12638.90	r
scoreSorter/phone31/U2370/op (xor2_2)	231.60	12870.50	r
scoreSorter/phone31/U5694/op (inv_2)	123.51	12994.01	f
scoreSorter/phone31/U1803/op (nor2_2)	153.52	13147.53	r
scoreSorter/phone31/U4946/op (or2_2)	188.39	13335.92	r
scoreSorter/phone31/U4947/op (inv_2)	64.02	13399.94	f
scoreSorter/phone31/U1806/op (nor2_2)	172.15	13572.09	r
scoreSorter/phone31/U3155/op (xor2_2)	194.88	13766.97	r
scoreSorter/phone31/phoneScore[17] (scorer_7)	0.00	13766.97	r
scoreSorter/sortPhones/ps31[17] (sorter)	0.00	13766.97	r
scoreSorter/sortPhones/U5153/op (nand2_2)	180.10	13947.07	f
scoreSorter/sortPhones/U6369/op (inv_2)	137.28	14084.35	r
scoreSorter/sortPhones/U1868/op (nor2_2)	96.75	14181.10	f
scoreSorter/sortPhones/U1869/op (nor3_2)	218.32	14399.43	r
scoreSorter/sortPhones/U1871/op (nor3_2)	115.55	14514.98	f
scoreSorter/sortPhones/U1873/op (nor2_2)	161.66	14676.64	r
scoreSorter/sortPhones/U1874/op (nor2_2)	73.48	14750.12	f
scoreSorter/sortPhones/U1356/op (nor2_2)	884.59	15634.71	r
scoreSorter/sortPhones/U4279/op (mux2_2)	803.85	16438.56	f
scoreSorter/sortPhones/U1941/op (nor2_2)	163.98	16602.54	r
scoreSorter/sortPhones/U1942/op (nor3_2)	99.20	16701.74	f
scoreSorter/sortPhones/U1945/op (nor3_2)	217.78	16919.52	r
scoreSorter/sortPhones/U1948/op (nor3_2)	109.20	17028.71	f
scoreSorter/sortPhones/U1951/op (nor3_2)	226.29	17255.01	r
scoreSorter/sortPhones/U1954/op (nor3_2)	109.45	17364.45	f
scoreSorter/sortPhones/U1957/op (nor3_2)	226.51	17590.96	r
scoreSorter/sortPhones/U1960/op (nor3_2)	109.45	17700.42	f
scoreSorter/sortPhones/U1963/op (nor3_2)	226.52	17926.93	r
scoreSorter/sortPhones/U1966/op (nor3_2)	109.45	18036.38	f
scoreSorter/sortPhones/U1969/op (nor3_2)	226.52	18262.90	r
scoreSorter/sortPhones/U1972/op (nor3_2)	109.45	18372.35	f
scoreSorter/sortPhones/U1975/op (nor3_2)	226.52	18598.87	r
scoreSorter/sortPhones/U1978/op (nor3_2)	109.45	18708.32	f
scoreSorter/sortPhones/U1981/op (nor3_2)	226.52	18934.84	r
scoreSorter/sortPhones/U1984/op (nor3_2)	109.45	19044.29	f
scoreSorter/sortPhones/U1987/op (nor3_2)	226.52	19270.81	r
scoreSorter/sortPhones/U1990/op (nor3_2)	109.45	19380.26	f
scoreSorter/sortPhones/U1992/op (nor3_2)	226.52	19606.78	r
scoreSorter/sortPhones/U1994/op (nor3_2)	115.55	19722.33	f
scoreSorter/sortPhones/U1360/op (nor2_2)	902.37	20624.69	r
scoreSorter/sortPhones/U4319/op (mux2_2)	812.56	21437.25	f
scoreSorter/sortPhones/U2181/op (nor2_2)	164.17	21601.42	r
scoreSorter/sortPhones/U2182/op (nor3_2)	99.22	21700.64	f
scoreSorter/sortPhones/U2185/op (nor3_2)	217.79	21918.44	r
scoreSorter/sortPhones/U2188/op (nor3_2)	109.20	22027.63	f
scoreSorter/sortPhones/U2191/op (nor3_2)	226.30	22253.93	r
scoreSorter/sortPhones/U2194/op (nor3_2)	109.45	22363.38	f
scoreSorter/sortPhones/U2197/op (nor3_2)	226.51	22589.89	r
scoreSorter/sortPhones/U2200/op (nor3_2)	109.45	22699.34	f
scoreSorter/sortPhones/U2203/op (nor3_2)	226.52	22925.86	r
scoreSorter/sortPhones/U2206/op (nor3_2)	109.45	23035.31	f
scoreSorter/sortPhones/U2209/op (nor3_2)	226.52	23261.83	r
scoreSorter/sortPhones/U2212/op (nor3_2)	109.45	23371.28	f
scoreSorter/sortPhones/U2215/op (nor3_2)	226.52	23597.79	r
scoreSorter/sortPhones/U2218/op (nor3_2)	109.45	23707.25	f
scoreSorter/sortPhones/U2221/op (nor3_2)	226.52	23933.76	r
scoreSorter/sortPhones/U2224/op (nor3_2)	109.45	24043.22	f
scoreSorter/sortPhones/U2227/op (nor3_2)	226.52	24269.73	r
scoreSorter/sortPhones/U2230/op (nor3_2)	109.45	24379.19	f
scoreSorter/sortPhones/U2232/op (nor3_2)	226.52	24605.70	r
scoreSorter/sortPhones/U2234/op (nor3_2)	115.55	24721.25	f
scoreSorter/sortPhones/U1368/op (nor2_2)	902.37	25623.62	r
scoreSorter/sortPhones/U4359/op (mux2_2)	812.71	26436.32	f
scoreSorter/sortPhones/U7950/op (inv_2)	79.05	26515.38	r
scoreSorter/sortPhones/U2661/op (nor2_2)	67.49	26582.87	f
scoreSorter/sortPhones/U2664/op (nor2_2)	146.47	26729.34	r
scoreSorter/sortPhones/U2667/op (nor3_2)	97.38	26826.72	f
scoreSorter/sortPhones/U2670/op (nor3_2)	216.46	27043.18	r
scoreSorter/sortPhones/U2673/op (nor3_2)	109.16	27152.34	f
scoreSorter/sortPhones/U2676/op (nor3_2)	226.26	27378.60	r
scoreSorter/sortPhones/U2679/op (nor3_2)	109.45	27488.04	f
scoreSorter/sortPhones/U2682/op (nor3_2)	226.51	27714.55	r
scoreSorter/sortPhones/U2685/op (nor3_2)	109.45	27824.01	f
scoreSorter/sortPhones/U2688/op (nor3_2)	226.52	28050.52	r
scoreSorter/sortPhones/U2691/op (nor3_2)	109.45	28159.98	f
scoreSorter/sortPhones/U2694/op (nor3_2)	226.52	28386.49	r
scoreSorter/sortPhones/U2697/op (nor3_2)	109.45	28495.95	f
scoreSorter/sortPhones/U2700/op (nor3_2)	226.52	28722.46	r
scoreSorter/sortPhones/U2703/op (nor3_2)	109.45	28831.91	f
scoreSorter/sortPhones/U2706/op (nor3_2)	226.52	29058.43	r
scoreSorter/sortPhones/U2709/op (nor3_2)	109.45	29167.88	f
scoreSorter/sortPhones/U2711/op (nor3_2)	226.52	29394.40	r
scoreSorter/sortPhones/U2713/op (nor3_2)	115.55	29509.95	f
scoreSorter/sortPhones/U2715/op (nor2_2)	161.66	29671.61	r
scoreSorter/sortPhones/U2716/op (nor2_2)	73.48	29745.09	f
scoreSorter/sortPhones/U1384/op (nor2_2)	876.52	30621.61	r
scoreSorter/sortPhones/U4400/op (mux2_2)	809.47	31431.08	f

scoreSorter/sortPhones/U3624/op (nor2_2)	164.10	31595.18	r
scoreSorter/sortPhones/U3625/op (nor3_2)	99.21	31694.39	f
scoreSorter/sortPhones/U3628/op (nor3_2)	217.79	31912.18	r
scoreSorter/sortPhones/U3631/op (nor3_2)	109.20	32021.38	f
scoreSorter/sortPhones/U3634/op (nor3_2)	226.30	32247.68	r
scoreSorter/sortPhones/U3637/op (nor3_2)	109.45	32357.12	f
scoreSorter/sortPhones/U3640/op (nor3_2)	226.51	32583.63	r
scoreSorter/sortPhones/U3643/op (nor3_2)	109.45	32693.09	f
scoreSorter/sortPhones/U3646/op (nor3_2)	226.52	32919.60	r
scoreSorter/sortPhones/U3649/op (nor3_2)	109.45	33029.05	f
scoreSorter/sortPhones/U3652/op (nor3_2)	226.52	33255.57	r
scoreSorter/sortPhones/U3655/op (nor3_2)	109.45	33365.02	f
scoreSorter/sortPhones/U3658/op (nor3_2)	226.52	33591.54	r
scoreSorter/sortPhones/U3661/op (nor3_2)	109.45	33700.99	f
scoreSorter/sortPhones/U3664/op (nor3_2)	226.52	33927.51	r
scoreSorter/sortPhones/U3667/op (nor3_2)	109.45	34036.96	f
scoreSorter/sortPhones/U3670/op (nor3_2)	226.52	34263.48	r
scoreSorter/sortPhones/U3673/op (nor3_2)	109.45	34372.93	f
scoreSorter/sortPhones/U3675/op (nor3_2)	226.52	34599.45	r
scoreSorter/sortPhones/U3677/op (nor3_2)	115.55	34715.00	f
scoreSorter/sortPhones/U1416/op (nor2_2)	943.04	35658.04	r
scoreSorter/sortPhones/U4543/op (mux2_2)	659.81	36317.85	r
scoreSorter/sortPhones/U8289/op (inv_2)	69.30	36387.15	f
scoreSorter/sortPhones/U3680/op (nor2_2)	137.33	36524.48	r
scoreSorter/sortPhones/U3681/op (nor3_2)	99.22	36623.70	f
scoreSorter/sortPhones/U3684/op (nor3_2)	217.80	36841.50	r
scoreSorter/sortPhones/U3687/op (nor3_2)	109.20	36950.70	f
scoreSorter/sortPhones/U3690/op (nor3_2)	226.30	37176.99	r
scoreSorter/sortPhones/U3693/op (nor3_2)	109.45	37286.44	f
scoreSorter/sortPhones/U3696/op (nor3_2)	226.51	37512.95	r
scoreSorter/sortPhones/U3699/op (nor3_2)	109.45	37622.40	f
scoreSorter/sortPhones/U3702/op (nor3_2)	226.52	37848.92	r
scoreSorter/sortPhones/U3705/op (nor3_2)	109.45	37958.37	f
scoreSorter/sortPhones/U3708/op (nor3_2)	226.52	38184.89	r
scoreSorter/sortPhones/U3711/op (nor3_2)	109.45	38294.34	f
scoreSorter/sortPhones/U3714/op (nor3_2)	226.52	38520.86	r
scoreSorter/sortPhones/U3717/op (nor3_2)	109.45	38630.31	f
scoreSorter/sortPhones/U3720/op (nor3_2)	226.52	38856.82	r
scoreSorter/sortPhones/U3723/op (nor3_2)	109.45	38966.28	f
scoreSorter/sortPhones/U3726/op (nor3_2)	226.52	39192.79	r
scoreSorter/sortPhones/U3729/op (nor3_2)	109.45	39302.25	f
scoreSorter/sortPhones/U3731/op (nor3_2)	226.52	39528.76	r
scoreSorter/sortPhones/U3733/op (nor3_2)	115.55	39644.31	f
scoreSorter/sortPhones/U1417/op (nor2_2)	902.37	40546.68	r
scoreSorter/sortPhones/U8320/op (mux2_2)	1182.29	41728.96	f
scoreSorter/sortPhones/U8321/op (inv_2)	307.73	42036.70	r
scoreSorter/sortPhones/U8363/op (inv_4)	322.12	42358.82	f
scoreSorter/sortPhones/U7514/op (xor2_2)	193.20	42552.02	f
scoreSorter/sortPhones/U6121/op (nand3_2)	105.85	42657.86	r
scoreSorter/sortPhones/U6122/op (nor3_2)	95.49	42753.36	f
scoreSorter/sortPhones/U7010/op (nand3_2)	130.35	42883.70	r
scoreSorter/sortPhones/U5436/op (nand3_2)	241.52	43125.22	f
scoreSorter/sortPhones/U7093/op (inv_2)	74.61	43199.82	r
scoreSorter/sortPhones/U5437/op (nand2_2)	173.20	43373.02	f
scoreSorter/sortPhones/U7067/op (inv_2)	75.80	43448.82	r
scoreSorter/sortPhones/U5438/op (nand3_2)	226.67	43675.49	f
scoreSorter/sortPhones/U7092/op (inv_2)	74.61	43750.09	r
scoreSorter/sortPhones/U5441/op (nand2_2)	231.48	43981.58	f
scoreSorter/sortPhones/U7060/op (inv_2)	79.93	44061.50	r
scoreSorter/sortPhones/U5443/op (nand3_2)	343.48	44404.98	f
scoreSorter/sortPhones/U7053/op (inv_2)	95.18	44500.16	r
scoreSorter/sortPhones/U5446/op (nand3_2)	354.85	44855.02	f
scoreSorter/sortPhones/U7046/op (inv_2)	98.66	44953.68	r
scoreSorter/sortPhones/U5449/op (nand3_2)	321.22	45274.90	f
scoreSorter/sortPhones/U7039/op (inv_2)	123.37	45398.27	r
scoreSorter/sortPhones/U5452/op (nand3_2)	361.07	45759.35	f
scoreSorter/sortPhones/U7032/op (inv_2)	99.12	45858.47	r
scoreSorter/sortPhones/U5454/op (nand3_2)	321.49	46179.96	f
scoreSorter/sortPhones/U7027/op (inv_2)	119.42	46299.38	r
scoreSorter/sortPhones/U5456/op (nand2_2)	225.17	46524.55	f
scoreSorter/sortPhones/U7022/op (inv_2)	78.69	46603.24	r
scoreSorter/sortPhones/U7185/op (nand2_2)	113.46	46716.70	f
scoreSorter/sortPhones/U1423/op (and3_2)	188.49	46905.19	f
scoreSorter/sortPhones/U1445/op (nand2_2)	104.34	47009.53	r
scoreSorter/sortPhones/topIndex[2] (sorter)	0.00	47009.53	r
scoreSorter/U542/op (nand2_2)	126.37	47135.89	f
scoreSorter/U543/op (nand2_2)	161.51	47297.41	r
scoreSorter/U544/op (nand2_2)	173.24	47470.65	f
scoreSorter/U324/op (nand2_2)	61.80	47532.45	r
scoreSorter/index5_reg[2]/ip (dp_2)	0.00	47532.45	r
data arrival time		47532.45	
clock clock (rise edge)	60000.00	60000.00	
clock network delay (ideal)	0.00	60000.00	
clock uncertainty	-300.00	59700.00	
scoreSorter/index5_reg[2]/ck (dp_2)	0.00	59700.00	r
library setup time	-109.15	59590.86	
data required time		59590.86	

```

data required time          59590.86
data arrival time          -47532.45
-----
slack (MET)                12058.40

```

## script3.sc timing report

```

*****
Report : timing
        -path full
        -delay min
        -max_paths 1
Design : top
Version: 2001.08
Date   : Sun Apr 16 23:46:59 2006
*****

```

Operating Conditions:  
Wire Load Model Mode: top

```

Startpoint: wordMatcher/wm_wdata_stop_t_reg[0]
            (rising edge-triggered flip-flop clocked by clock)
Endpoint:   wordMatcher/wm_wdata_stop_t_reg[0]
            (rising edge-triggered flip-flop clocked by clock)
Path Group: clock
Path Type:  min

```

Point	Incr	Path
-----		
clock clock (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
wordMatcher/wm_wdata_stop_t_reg[0]/ck (dp_2)	0.00	0.00 r
wordMatcher/wm_wdata_stop_t_reg[0]/q (dp_2)	101.21	101.21 r
wordMatcher/U7227/op (nand2_1)	84.81	186.02 f
wordMatcher/U7225/op (inv_4)	37.49	223.51 r
wordMatcher/U7226/op (inv_4)	11.49	235.00 f
wordMatcher/U7224/op (nand3_1)	13.84	248.84 r
wordMatcher/U7223/op (buf_1)	51.17	300.01 r
wordMatcher/wm_wdata_stop_t_reg[0]/ip (dp_2)	0.00	300.01 r
data arrival time		300.01
clock clock (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
clock uncertainty	300.00	300.00
wordMatcher/wm_wdata_stop_t_reg[0]/ck (dp_2)	0.00	300.00 r
library hold time	0.00	300.00
data required time		300.00
-----		
data required time		300.00
data arrival time		-300.01
-----		
slack (MET)		0.01

## script4.sc timing report

```

*****
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : top
Version: 2001.08
Date   : Sun Apr 16 23:57:21 2006
*****

```

Operating Conditions:  
Wire Load Model Mode: top

```

Startpoint: in31[21] (input port clocked by clock)
Endpoint:   scoreSorter/index4_reg[2]
            (rising edge-triggered flip-flop clocked by clock)
Path Group: clock
Path Type:  max

```

Point	Incr	Path
-----		
clock clock (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
input external delay	1100.00	1100.00 r
in31[21] (in)	451.97	1551.97 r
scoreSorter/in31[21] (phoneScoreSort)	0.00	1551.97 r
scoreSorter/phone31/inScore[21] (scorer_7)	0.00	1551.97 r
scoreSorter/phone31/U2420/op (inv_2)	155.12	1707.10 f
scoreSorter/phone31/U256/op (nor2_2)	249.52	1956.62 r
scoreSorter/phone31/U1838/op (xor2_2)	262.56	2219.18 r
scoreSorter/phone31/U3305/op (xor2_2)	231.50	2450.67 r
scoreSorter/phone31/U1839/op (xor2_2)	233.57	2684.24 r

scoreSorter/phone31/U4968/op (inv_2)	115.97	2800.21	f
scoreSorter/phone31/U3306/op (xor2_2)	218.36	3018.58	r
scoreSorter/phone31/U1840/op (xor2_2)	235.53	3254.11	r
scoreSorter/phone31/U4969/op (inv_2)	112.85	3366.96	f
scoreSorter/phone31/U3307/op (xor2_2)	215.61	3582.58	r
scoreSorter/phone31/U1841/op (xor2_2)	235.53	3818.11	r
scoreSorter/phone31/U4970/op (inv_2)	83.82	3901.92	f
scoreSorter/phone31/U3308/op (xor2_2)	211.49	4113.41	r
scoreSorter/phone31/U1842/op (xor2_2)	235.54	4348.95	r
scoreSorter/phone31/U4971/op (inv_2)	112.85	4461.80	f
scoreSorter/phone31/U3309/op (xor2_2)	215.61	4677.41	r
scoreSorter/phone31/U1843/op (xor2_2)	231.77	4909.18	r
scoreSorter/phone31/U5035/op (inv_2)	123.52	5032.70	f
scoreSorter/phone31/U3310/op (xor2_2)	230.83	5263.53	r
scoreSorter/phone31/U1922/op (xor2_2)	231.60	5495.13	r
scoreSorter/phone31/U5076/op (inv_2)	112.35	5607.49	f
scoreSorter/phone31/U3316/op (xor2_2)	215.17	5822.66	r
scoreSorter/phone31/U1967/op (xor2_2)	235.36	6058.02	r
scoreSorter/phone31/U5130/op (inv_2)	110.28	6168.30	f
scoreSorter/phone31/U538/op (nor2_2)	157.86	6326.16	r
scoreSorter/phone31/U539/op (nor2_2)	75.81	6401.97	f
scoreSorter/phone31/U2634/op (or2_2)	252.63	6654.60	f
scoreSorter/phone31/U557/op (nor2_2)	148.77	6803.37	r
scoreSorter/phone31/U558/op (nor2_2)	94.09	6897.46	f
scoreSorter/phone31/U4333/op (or2_2)	242.44	7139.90	f
scoreSorter/phone31/U4334/op (inv_2)	71.38	7211.27	r
scoreSorter/phone31/U579/op (nor2_2)	82.78	7294.05	f
scoreSorter/phone31/U2636/op (or2_2)	252.63	7546.68	f
scoreSorter/phone31/U1407/op (nor2_2)	148.74	7695.43	r
scoreSorter/phone31/U1408/op (nor2_2)	93.17	7788.60	f
scoreSorter/phone31/U2982/op (or2_2)	259.75	8048.34	f
scoreSorter/phone31/U1410/op (nor2_2)	148.07	8196.41	r
scoreSorter/phone31/U1411/op (nor2_2)	93.68	8290.09	f
scoreSorter/phone31/U4723/op (or2_2)	242.06	8532.15	f
scoreSorter/phone31/U4724/op (inv_2)	71.37	8603.52	r
scoreSorter/phone31/U1415/op (nor2_2)	82.78	8686.30	f
scoreSorter/phone31/U3091/op (or2_2)	252.63	8938.93	f
scoreSorter/phone31/U1417/op (nor2_2)	148.43	9087.35	r
scoreSorter/phone31/U1418/op (nor2_2)	94.08	9181.44	f
scoreSorter/phone31/U3122/op (or2_2)	262.45	9443.89	f
scoreSorter/phone31/U1420/op (nor2_2)	148.60	9592.49	r
scoreSorter/phone31/U1421/op (nor2_2)	94.08	9686.58	f
scoreSorter/phone31/U4727/op (or2_2)	242.44	9929.01	f
scoreSorter/phone31/U4728/op (inv_2)	71.38	10000.39	r
scoreSorter/phone31/U1428/op (nor2_2)	82.78	10083.17	f
scoreSorter/phone31/U4733/op (or2_2)	232.65	10315.82	f
scoreSorter/phone31/U2303/op (xor2_2)	210.00	10525.82	r
scoreSorter/phone31/U5649/op (inv_2)	124.19	10650.01	f
scoreSorter/phone31/U3549/op (xor2_2)	231.06	10881.07	r
scoreSorter/phone31/U2304/op (xor2_2)	231.61	11112.68	r
scoreSorter/phone31/U5650/op (inv_2)	123.51	11236.19	f
scoreSorter/phone31/U3550/op (xor2_2)	230.83	11467.02	r
scoreSorter/phone31/U2327/op (xor2_2)	231.60	11698.62	r
scoreSorter/phone31/U5651/op (inv_2)	123.51	11822.13	f
scoreSorter/phone31/U3551/op (xor2_2)	230.83	12052.96	r
scoreSorter/phone31/U2349/op (xor2_2)	231.60	12284.56	r
scoreSorter/phone31/U5652/op (inv_2)	123.51	12408.07	f
scoreSorter/phone31/U3563/op (xor2_2)	230.83	12638.90	r
scoreSorter/phone31/U2370/op (xor2_2)	231.60	12870.50	r
scoreSorter/phone31/U5694/op (inv_2)	123.51	12994.01	f
scoreSorter/phone31/U1803/op (nor2_2)	153.52	13147.53	r
scoreSorter/phone31/U4946/op (or2_2)	188.39	13335.92	r
scoreSorter/phone31/U4947/op (inv_2)	64.02	13399.94	f
scoreSorter/phone31/U1806/op (nor2_2)	172.15	13572.09	r
scoreSorter/phone31/U3155/op (xor2_2)	194.88	13766.97	r
scoreSorter/phone31/phoneScore[17] (scorer_7)	0.00	13766.97	r
scoreSorter/sortPhones/ps31[17] (sorter)	0.00	13766.97	r
scoreSorter/sortPhones/U5153/op (nand2_2)	180.10	13947.07	f
scoreSorter/sortPhones/U6369/op (inv_2)	137.28	14084.35	r
scoreSorter/sortPhones/U1868/op (nor2_2)	96.75	14181.10	f
scoreSorter/sortPhones/U1869/op (nor3_2)	218.32	14399.43	r
scoreSorter/sortPhones/U1871/op (nor3_2)	115.55	14514.98	f
scoreSorter/sortPhones/U1873/op (nor2_2)	161.66	14676.64	r
scoreSorter/sortPhones/U1874/op (nor2_2)	73.48	14750.12	f
scoreSorter/sortPhones/U1356/op (nor2_2)	884.59	15634.71	r
scoreSorter/sortPhones/U4279/op (mux2_2)	803.85	16438.56	f
scoreSorter/sortPhones/U1941/op (nor2_2)	163.98	16602.54	r
scoreSorter/sortPhones/U1942/op (nor3_2)	99.20	16701.74	f
scoreSorter/sortPhones/U1945/op (nor3_2)	217.78	16919.52	r
scoreSorter/sortPhones/U1948/op (nor3_2)	109.20	17028.71	f
scoreSorter/sortPhones/U1951/op (nor3_2)	226.29	17255.01	r
scoreSorter/sortPhones/U1954/op (nor3_2)	109.45	17364.46	f
scoreSorter/sortPhones/U1957/op (nor3_2)	226.51	17590.96	r
scoreSorter/sortPhones/U1960/op (nor3_2)	109.45	17700.42	f
scoreSorter/sortPhones/U1963/op (nor3_2)	226.52	17926.93	r
scoreSorter/sortPhones/U1966/op (nor3_2)	109.45	18036.39	f
scoreSorter/sortPhones/U1969/op (nor3_2)	226.52	18262.90	r
scoreSorter/sortPhones/U1972/op (nor3_2)	109.45	18372.36	f
scoreSorter/sortPhones/U1975/op (nor3_2)	226.52	18598.87	r

scoreSorter/sortPhones/U1978/op	(nor3_2)	109.45	18708.32	f
scoreSorter/sortPhones/U1981/op	(nor3_2)	226.52	18934.84	r
scoreSorter/sortPhones/U1984/op	(nor3_2)	109.45	19044.29	f
scoreSorter/sortPhones/U1987/op	(nor3_2)	226.52	19270.81	r
scoreSorter/sortPhones/U1990/op	(nor3_2)	109.45	19380.26	f
scoreSorter/sortPhones/U1992/op	(nor3_2)	226.52	19606.78	r
scoreSorter/sortPhones/U1994/op	(nor3_2)	115.55	19722.33	f
scoreSorter/sortPhones/U1360/op	(nor2_2)	902.37	20624.70	r
scoreSorter/sortPhones/U4319/op	(mux2_2)	812.56	21437.26	f
scoreSorter/sortPhones/U2181/op	(nor2_2)	164.17	21601.42	r
scoreSorter/sortPhones/U2182/op	(nor3_2)	99.22	21700.65	f
scoreSorter/sortPhones/U2185/op	(nor3_2)	217.79	21918.44	r
scoreSorter/sortPhones/U2188/op	(nor3_2)	109.20	22027.64	f
scoreSorter/sortPhones/U2191/op	(nor3_2)	226.30	22253.93	r
scoreSorter/sortPhones/U2194/op	(nor3_2)	109.45	22363.38	f
scoreSorter/sortPhones/U2197/op	(nor3_2)	226.51	22589.89	r
scoreSorter/sortPhones/U2200/op	(nor3_2)	109.45	22699.34	f
scoreSorter/sortPhones/U2203/op	(nor3_2)	226.52	22925.86	r
scoreSorter/sortPhones/U2206/op	(nor3_2)	109.45	23035.31	f
scoreSorter/sortPhones/U2209/op	(nor3_2)	226.52	23261.83	r
scoreSorter/sortPhones/U2212/op	(nor3_2)	109.45	23371.28	f
scoreSorter/sortPhones/U2215/op	(nor3_2)	226.52	23597.80	r
scoreSorter/sortPhones/U2218/op	(nor3_2)	109.45	23707.25	f
scoreSorter/sortPhones/U2221/op	(nor3_2)	226.52	23933.77	r
scoreSorter/sortPhones/U2224/op	(nor3_2)	109.45	24043.22	f
scoreSorter/sortPhones/U2227/op	(nor3_2)	226.52	24269.73	r
scoreSorter/sortPhones/U2230/op	(nor3_2)	109.45	24379.19	f
scoreSorter/sortPhones/U2232/op	(nor3_2)	226.52	24605.70	r
scoreSorter/sortPhones/U2234/op	(nor3_2)	115.55	24721.25	f
scoreSorter/sortPhones/U1368/op	(nor2_2)	902.37	25623.62	r
scoreSorter/sortPhones/U4359/op	(mux2_2)	812.71	26436.33	f
scoreSorter/sortPhones/U7950/op	(inv_2)	79.05	26515.38	r
scoreSorter/sortPhones/U2661/op	(nor2_2)	67.49	26582.87	f
scoreSorter/sortPhones/U2664/op	(nor2_2)	146.47	26729.34	r
scoreSorter/sortPhones/U2667/op	(nor3_2)	97.38	26826.72	f
scoreSorter/sortPhones/U2670/op	(nor3_2)	216.46	27043.18	r
scoreSorter/sortPhones/U2673/op	(nor3_2)	109.16	27152.34	f
scoreSorter/sortPhones/U2676/op	(nor3_2)	226.26	27378.60	r
scoreSorter/sortPhones/U2679/op	(nor3_2)	109.45	27488.05	f
scoreSorter/sortPhones/U2682/op	(nor3_2)	226.51	27714.56	r
scoreSorter/sortPhones/U2685/op	(nor3_2)	109.45	27824.01	f
scoreSorter/sortPhones/U2688/op	(nor3_2)	226.52	28050.53	r
scoreSorter/sortPhones/U2691/op	(nor3_2)	109.45	28159.98	f
scoreSorter/sortPhones/U2694/op	(nor3_2)	226.52	28386.49	r
scoreSorter/sortPhones/U2697/op	(nor3_2)	109.45	28495.95	f
scoreSorter/sortPhones/U2700/op	(nor3_2)	226.52	28722.46	r
scoreSorter/sortPhones/U2703/op	(nor3_2)	109.45	28831.92	f
scoreSorter/sortPhones/U2706/op	(nor3_2)	226.52	29058.43	r
scoreSorter/sortPhones/U2709/op	(nor3_2)	109.45	29167.88	f
scoreSorter/sortPhones/U2711/op	(nor3_2)	226.52	29394.40	r
scoreSorter/sortPhones/U2713/op	(nor3_2)	115.55	29509.95	f
scoreSorter/sortPhones/U2715/op	(nor2_2)	161.66	29671.61	r
scoreSorter/sortPhones/U2716/op	(nor2_2)	73.48	29745.09	f
scoreSorter/sortPhones/U1384/op	(nor2_2)	876.52	30621.61	r
scoreSorter/sortPhones/U4400/op	(mux2_2)	809.47	31431.08	f
scoreSorter/sortPhones/U3624/op	(nor2_2)	164.10	31595.18	r
scoreSorter/sortPhones/U3625/op	(nor3_2)	99.21	31694.40	f
scoreSorter/sortPhones/U3628/op	(nor3_2)	217.79	31912.19	r
scoreSorter/sortPhones/U3631/op	(nor3_2)	109.20	32021.38	f
scoreSorter/sortPhones/U3634/op	(nor3_2)	226.30	32247.68	r
scoreSorter/sortPhones/U3637/op	(nor3_2)	109.45	32357.12	f
scoreSorter/sortPhones/U3640/op	(nor3_2)	226.51	32583.63	r
scoreSorter/sortPhones/U3643/op	(nor3_2)	109.45	32693.09	f
scoreSorter/sortPhones/U3646/op	(nor3_2)	226.51	32919.60	r
scoreSorter/sortPhones/U3649/op	(nor3_2)	109.45	33029.05	f
scoreSorter/sortPhones/U3652/op	(nor3_2)	226.52	33255.57	r
scoreSorter/sortPhones/U3655/op	(nor3_2)	109.45	33365.02	f
scoreSorter/sortPhones/U3658/op	(nor3_2)	226.52	33591.54	r
scoreSorter/sortPhones/U3661/op	(nor3_2)	109.45	33700.99	f
scoreSorter/sortPhones/U3664/op	(nor3_2)	226.52	33927.51	r
scoreSorter/sortPhones/U3667/op	(nor3_2)	109.45	34036.96	f
scoreSorter/sortPhones/U3670/op	(nor3_2)	226.52	34263.48	r
scoreSorter/sortPhones/U3673/op	(nor3_2)	109.45	34372.93	f
scoreSorter/sortPhones/U3675/op	(nor3_2)	226.52	34599.45	r
scoreSorter/sortPhones/U3677/op	(nor3_2)	115.55	34715.00	f
scoreSorter/sortPhones/U1416/op	(nor2_2)	943.04	35658.04	r
scoreSorter/sortPhones/U4543/op	(mux2_2)	659.81	36317.85	r
scoreSorter/sortPhones/U8289/op	(inv_2)	69.30	36387.15	f
scoreSorter/sortPhones/U3680/op	(nor2_2)	137.33	36524.48	r
scoreSorter/sortPhones/U3681/op	(nor3_2)	99.22	36623.70	f
scoreSorter/sortPhones/U3684/op	(nor3_2)	217.80	36841.50	r
scoreSorter/sortPhones/U3687/op	(nor3_2)	109.20	36950.70	f
scoreSorter/sortPhones/U3690/op	(nor3_2)	226.30	37176.99	r
scoreSorter/sortPhones/U3693/op	(nor3_2)	109.45	37286.44	f
scoreSorter/sortPhones/U3696/op	(nor3_2)	226.51	37512.95	r
scoreSorter/sortPhones/U3699/op	(nor3_2)	109.45	37622.40	f
scoreSorter/sortPhones/U3702/op	(nor3_2)	226.52	37848.92	r
scoreSorter/sortPhones/U3705/op	(nor3_2)	109.45	37958.37	f
scoreSorter/sortPhones/U3708/op	(nor3_2)	226.52	38184.89	r

scoreSorter/sortPhones/U3711/op (nor3_2)	109.45	38294.34	f
scoreSorter/sortPhones/U3714/op (nor3_2)	226.52	38520.86	r
scoreSorter/sortPhones/U3717/op (nor3_2)	109.45	38630.31	f
scoreSorter/sortPhones/U3720/op (nor3_2)	226.52	38856.82	r
scoreSorter/sortPhones/U3723/op (nor3_2)	109.45	38966.28	f
scoreSorter/sortPhones/U3726/op (nor3_2)	226.52	39192.79	r
scoreSorter/sortPhones/U3729/op (nor3_2)	109.45	39302.25	f
scoreSorter/sortPhones/U3731/op (nor3_2)	226.52	39528.76	r
scoreSorter/sortPhones/U3733/op (nor3_2)	115.55	39644.31	f
scoreSorter/sortPhones/U1417/op (nor2_2)	902.37	40546.68	r
scoreSorter/sortPhones/U8320/op (mux2_2)	1182.29	41728.96	f
scoreSorter/sortPhones/U8321/op (inv_2)	307.73	42036.70	r
scoreSorter/sortPhones/U8363/op (inv_4)	322.12	42358.82	f
scoreSorter/sortPhones/U7514/op (xor2_2)	193.20	42552.02	f
scoreSorter/sortPhones/U6121/op (nand3_2)	105.85	42657.86	r
scoreSorter/sortPhones/U6122/op (nor3_2)	95.49	42753.36	f
scoreSorter/sortPhones/U7010/op (nand3_2)	130.35	42883.70	r
scoreSorter/sortPhones/U5436/op (nand3_2)	241.52	43125.22	f
scoreSorter/sortPhones/U7093/op (inv_2)	74.61	43199.82	r
scoreSorter/sortPhones/U5437/op (nand2_2)	173.20	43373.02	f
scoreSorter/sortPhones/U7067/op (inv_2)	75.80	43448.82	r
scoreSorter/sortPhones/U5438/op (nand3_2)	226.67	43675.49	f
scoreSorter/sortPhones/U7092/op (inv_2)	74.61	43750.09	r
scoreSorter/sortPhones/U5441/op (nand2_2)	231.48	43981.58	f
scoreSorter/sortPhones/U7060/op (inv_2)	79.93	44061.50	r
scoreSorter/sortPhones/U5443/op (nand3_2)	343.48	44404.98	f
scoreSorter/sortPhones/U7053/op (inv_2)	95.18	44500.16	r
scoreSorter/sortPhones/U5446/op (nand3_2)	354.85	44855.02	f
scoreSorter/sortPhones/U7046/op (inv_2)	98.66	44953.68	r
scoreSorter/sortPhones/U5449/op (nand3_2)	321.22	45274.90	f
scoreSorter/sortPhones/U7039/op (inv_2)	123.37	45398.27	r
scoreSorter/sortPhones/U5452/op (nand3_2)	361.07	45759.35	f
scoreSorter/sortPhones/U7032/op (inv_2)	99.12	45858.47	r
scoreSorter/sortPhones/U5454/op (nand3_2)	321.49	46179.96	f
scoreSorter/sortPhones/U7027/op (inv_2)	119.42	46299.38	r
scoreSorter/sortPhones/U5456/op (nand2_2)	225.17	46524.55	f
scoreSorter/sortPhones/U7022/op (inv_2)	78.69	46603.24	r
scoreSorter/sortPhones/U7185/op (nand2_2)	113.46	46716.70	f
scoreSorter/sortPhones/U1423/op (and3_2)	188.49	46905.19	f
scoreSorter/sortPhones/U1445/op (nand2_2)	104.34	47009.53	r
scoreSorter/sortPhones/topIndex[2] (sorter)	0.00	47009.53	r
scoreSorter/U542/op (nand2_2)	126.37	47135.89	f
scoreSorter/U543/op (nand2_2)	161.51	47297.41	r
scoreSorter/U564/op (nand2_2)	173.24	47470.65	f
scoreSorter/U330/op (nand2_2)	76.22	47546.87	r
scoreSorter/U812/op (buf_2)	106.43	47653.30	r
scoreSorter/index4_reg[2]/ip (dp_2)	0.00	47653.30	r
data arrival time		47653.30	
-----			
clock clock (rise edge)	60000.00	60000.00	
clock network delay (ideal)	0.00	60000.00	
clock uncertainty	-300.00	59700.00	
scoreSorter/index4_reg[2]/ck (dp_2)	0.00	59700.00	r
library setup time	-102.01	59597.99	
data required time		59597.99	
-----			
data required time		59597.99	
data arrival time		-47653.30	
-----			
slack (MET)		11944.69	

## script4.sc area report

```
*****
Report : area
Design : top
Version: 2001.08
Date   : Mon Apr 17 00:27:37 2006
*****
```

Library(s) Used:

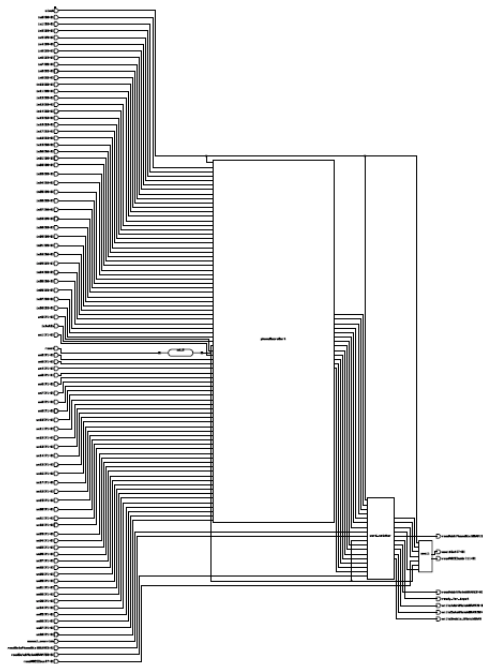
ncsulib25\_worst (File: /afs/eos.ncsu.edu/dist/cad446/local/TSMC025\_deep/ncsulib25\_worst.db)

```
Number of ports:      3922
Number of nets:       4105
Number of cells:      4
Number of references: 4
```

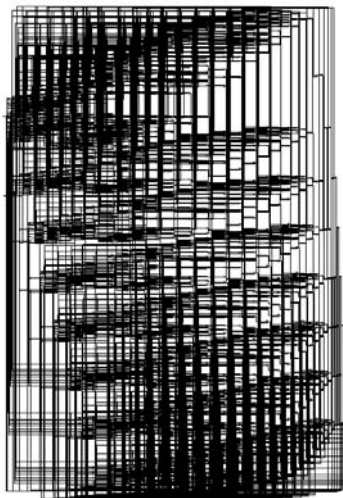
```
Combinational area:  14469939.000000
Noncombinational area: 132045.812500
Net Interconnect area: undefined (No wire load specified)
```

```
Total cell area:    14601985.000000
Total area:          undefined
```

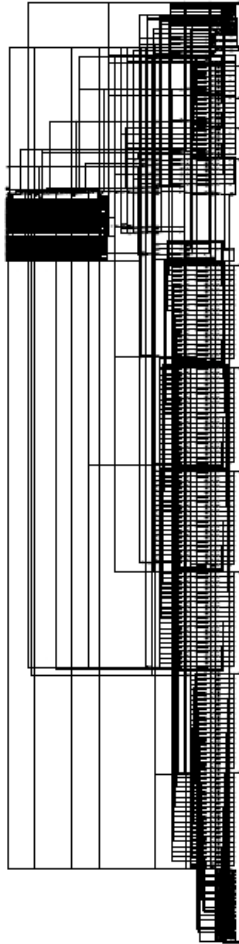
## Appendix D: Plots from final design



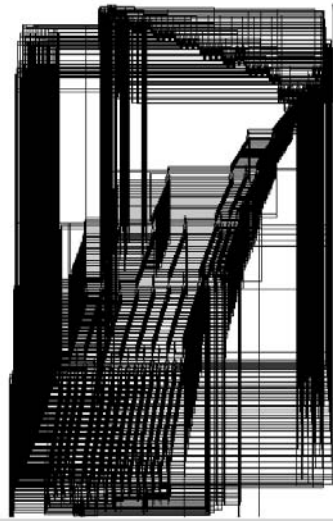
**Figure 7** – Synthesis result of top module



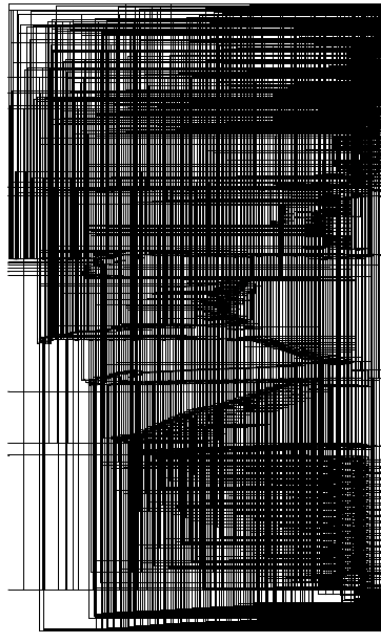
**Figure 8** – Synthesis result of scorer module



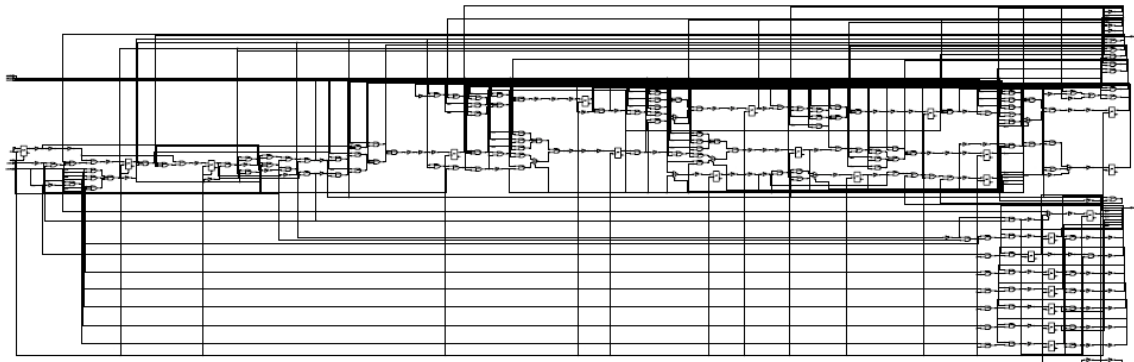
**Figure 9** – Synthesis result of phoneScorer module



**Figure 10** – Synthesis result of sorter module



**Figure 11** – Synthesis result of word matcher module



**Figure 12** – Synthesis result of ascii module

## Appendix E: Simulation results

The following command line output shows the utterance matching process for the example input of "THE SUPERCOOL ACTIVITY IS POSSIBLE THEORETICALLY"

```
Compiling source file "testTop1.v"
Compiling included source file "top.v"
Compiling included source file "phoneScorer.v"
Continuing compilation of source file "top.v"
Compiling included source file "word_matcher_3.v"
Continuing compilation of source file "top.v"
Compiling included source file "ascii.v"
Continuing compilation of source file "top.v"
Continuing compilation of source file "testTop1.v"
Compiling included source file "wm_sram.v"
Continuing compilation of source file "testTop1.v"
Compiling included source file "phdict_sram.v"
Continuing compilation of source file "testTop1.v"
Compiling included source file "sramASCII.v"
Continuing compilation of source file "testTop1.v"
Highest level modules:
testTop
-----A WORD WAS FOUND, RELOADING DICTIONARY-----
-----
INPUT #0 (phone_t = 0)
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
9/ 369735 33/ 278690 35/ 266750 20/ 237318 1/ 221392 19/ 201760
-----
INPUT #1 (phone_t = 1)
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
17/ 303221 33/ 286889 30/ 282909 29/ 248837 5/ 236631 10/ 235821
-----
INPUT #2 (phone_t = 2)
-----A WORD WAS FOUND, RELOADING DICTIONARY-----
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
26/ 321443 17/ 290995 28/ 277648 6/ 271220 15/ 260082 9/ 244516
-----INPUT #3 INPUT #3
(phone_t = 3)
-----A WORD WAS FOUND, RELOADING DICTIONARY-----
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
1/ 382604 33/ 368505 4/ 327397 26/ 319082 15/ 282994 35/ 276213
-----
INPUT #4 (phone_t = 4)
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
26/ 368457 5/ 327302 9/ 311137 1/ 306230 3/ 300840 2/ 292694
-----
INPUT #5 (phone_t = 5)
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
5/ 344230 4/ 278740 20/ 273511 11/ 257123 36/ 232564 29/ 221893
-----
INPUT #6 (phone_t = 6)
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
29/ 371443 2/ 340810 5/ 270066 3/ 226155 19/ 223868 4/ 216697
-----
INPUT #7 (phone_t = 7)
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
33/ 374727 30/ 286756 26/ 264090 23/ 263548 18/ 260613 1/ 251476
-----
INPUT #8 (phone_t = 8)
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
20/ 294156 26/ 285392 4/ 233172 33/ 222243 15/ 222106 35/ 221484
-----
INPUT #9 (phone_t = 9)
-----A WORD WAS FOUND, RELOADING DICTIONARY-----
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
1/ 382876 9/ 301563 33/ 287427 20/ 268818 18/ 249819 25/ 249624
-----
INPUT #10 (phone_t = 10)
-----A WORD WAS FOUND, RELOADING DICTIONARY-----
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
9/ 318542 1/ 302374 28/ 275722 4/ 265789 19/ 235404 29/ 229141
-----
INPUT #11 (phone_t = 11)
TOP INDICIES/SCORES
```

30/	#0 333483	33/	#1 321814	29/	#2 318134	7/	#3 312346	4/	#4 283924	15/	#5 281476
-----											
INPUT #12 (phone_t = 12)											
TOP INDICIES/SCORES											
16/	#0 365732	7/	#1 298558	29/	#2 298115	26/	#3 276071	3/	#4 256920	10/	#5 238341
-----											
INPUT #13 (phone_t = 13)											
TOP INDICIES/SCORES											
9/	#0 310279	2/	#1 278796	34/	#2 270344	26/	#3 261488	28/	#4 261068	15/	#5 259174
-----											
INPUT #14 (phone_t = 14)											
TOP INDICIES/SCORES											
2/	#0 364853	4/	#1 283738	36/	#2 272311	31/	#3 257097	3/	#4 253935	1/	#5 249816
-----											
INPUT #15 (phone_t = 15)											
TOP INDICIES/SCORES											
30/	#0 308797	31/	#1 271307	26/	#2 256973	19/	#3 240670	7/	#4 231332	9/	#5 228073
-----											
INPUT #16 (phone_t = 16)											
TOP INDICIES/SCORES											
2/	#0 355607	5/	#1 294510	17/	#2 281866	22/	#3 279912	7/	#4 244371	29/	#5 233789
-----											
INPUT #17 (phone_t = 17)											
-----A WORD WAS FOUND, RELOADING DICTIONARY-----											
TOP INDICIES/SCORES											
16/	#0 327289	30/	#1 302495	27/	#2 294094	33/	#3 278680	6/	#4 278612	36/	#5 256264
-----											
INPUT #18 (phone_t = 18)											
TOP INDICIES/SCORES											
28/	#0 298404	34/	#1 246046	26/	#2 245598	37/	#3 232666	33/	#4 226965	1/	#5 222218
-----											
INPUT #19 (phone_t = 19)											
-----A WORD WAS FOUND, RELOADING DICTIONARY-----											
TOP INDICIES/SCORES											
26/	#0 381200	4/	#1 288263	29/	#2 287531	9/	#3 280711	23/	#4 237629	7/	#5 234868
-----											
INPUT #20 (phone_t = 20)											
TOP INDICIES/SCORES											
2/	#0 380040	5/	#1 323381	4/	#2 294839	20/	#3 236121	0/	#4 232843	33/	#5 221731
-----											
INPUT #21 (phone_t = 21)											
-----A WORD WAS FOUND, RELOADING DICTIONARY-----											
TOP INDICIES/SCORES											
1/	#0 383472	28/	#1 294246	27/	#2 293445	33/	#3 290115	7/	#4 252005	26/	#5 251369
-----											
INPUT #22 (phone_t = 22)											
TOP INDICIES/SCORES											
2/	#0 361502	5/	#1 321128	1/	#2 301432	4/	#3 293310	18/	#4 266479	17/	#5 254754
-----											
INPUT #23 (phone_t = 23)											
TOP INDICIES/SCORES											
29/	#0 356484	6/	#1 315421	31/	#2 276797	20/	#3 261085	23/	#4 244414	1/	#5 233160
-----											
INPUT #24 (phone_t = 24)											
TOP INDICIES/SCORES											
29/	#0 427103	2/	#1 377907	31/	#2 317000	33/	#3 311917	30/	#4 310234	36/	#5 305875
-----											
INPUT #25 (phone_t = 25)											
TOP INDICIES/SCORES											
20/	#0 279549	6/	#1 263512	30/	#2 263437	18/	#3 250741	16/	#4 241378	36/	#5 229372
-----											
INPUT #26 (phone_t = 26)											
-----A WORD WAS FOUND, RELOADING DICTIONARY-----											
TOP INDICIES/SCORES											
9/	#0 324087	4/	#1 294220	31/	#2 293489	17/	#3 284119	35/	#4 250605	18/	#5 249186
-----											
INPUT #27 (phone_t = 27)											
TOP INDICIES/SCORES											
2/	#0 305087	9/	#1 302684	6/	#2 293952	17/	#3 279025	28/	#4 255636	36/	#5 250438
-----											
INPUT #28 (phone_t = 28)											
-----A WORD WAS FOUND, RELOADING DICTIONARY-----											

```

TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
29/ 325672 33/ 312559 30/ 275668 6/ 259117 2/ 254164 11/ 241952
-----
INPUT #29 (phone_t = 29)
-----WORD MATCHING COMPLETE-----
Trigram Words: 112 110 1
Trigram Score: 5380795
# Phones Resolved: 17

ASCII OUTPUT:
THE SUPERCOOL ACTIVITY
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
10/ 311349 30/ 310573 2/ 298765 18/ 253276 34/ 231924 33/ 231501
-----
INPUT #30 (phone_t = 13)
-----A WORD WAS FOUND, RELOADING DICTIONARY-----
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
30/ 323577 5/ 296486 1/ 284104 15/ 279866 21/ 250885 3/ 239090
-----
INPUT #31 (phone_t = 14)
-----A WORD WAS FOUND, RELOADING DICTIONARY-----
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
29/ 371208 16/ 341089 2/ 288594 28/ 271313 1/ 263634 30/ 262070
-----
INPUT #32 (phone_t = 15)
-----A WORD WAS FOUND, RELOADING DICTIONARY-----
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
26/ 340657 2/ 295694 33/ 282564 19/ 240484 31/ 236852 1/ 227236
-----
INPUT #33 (phone_t = 16)
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
1/ 387174 2/ 354413 26/ 307400 33/ 305167 35/ 273525 23/ 264234
-----
INPUT #34 (phone_t = 17)
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
9/ 311985 20/ 281553 1/ 276002 7/ 273660 29/ 267630 3/ 243245
-----
INPUT #35 (phone_t = 18)
-----A WORD WAS FOUND, RELOADING DICTIONARY-----
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
7/ 310511 4/ 297022 17/ 293342 33/ 255844 5/ 251640 25/ 246567
-----
INPUT #36 (phone_t = 19)
-----A WORD WAS FOUND, RELOADING DICTIONARY-----
TOP INDICIES/SCORES
#0 #1 #2 #3 #4 #5
38/ x 38/ x 38/ x 38/ x 38/ x 38/ x
END OF INPUT...INITIATING MANUAL OVERRIDE!!!
-----WORD MATCHING COMPLETE-----
Trigram Words: 5387 117
Trigram Score: 5762896
# Phones Resolved: 19

ASCII OUTPUT:
IS POSSIBLE THEORETICALLY

L239 "testTop1.v": $finish at simulation time 359280
0 simulation events (use +profile or +listcounts option to count)
CPU time: 0.1 secs to compile + 0.2 secs to link + 71.5 secs in simulation
End of Tool: VERILOG-XL 05.10.003-s Apr 17, 2006 01:42:13

```

## Appendix F: High-level model of design

```
/*
 * Frankie Myers, fbmyers
 * Eric Phillips, ecphilli
 *
 * ECE520
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_PHONES_PER_WORD 20
#define MAX_CHARS_PER_WORD 18
#define NUM_OBSERVATIONS_TO_BUFFER 30
//the locations in the wordTable where the phone and ascii characters start
#define PHONE_START_IDX 2
#define ASCII_START_IDX 2+MAX_PHONES_PER_WORD
#define NUM_PHONES 39
#define NUM_WORDS 148
#define SPACE_CHAR 0x20
typedef unsigned char BYTE;

//external tables
BYTE phoneTable[NUM_PHONES][3*3+2]; //each phone is represented by a 3x3 matrix, the
last two characters are the phonetic ASCII characters
BYTE wordTable[NUM_WORDS][2+MAX_PHONES_PER_WORD+MAX_CHARS_PER_WORD]; //first column is
the length of phones, second is length of ascii

//score table, stores all the scores for each phone at each input
BYTE scoreTable[NUM_OBSERVATIONS_TO_BUFFER][NUM_PHONES];

//points to current location in the score table
BYTE scoreTableIdx;

//function prototypes
void appendPhoneObservationVector(BYTE bi, BYTE bj, BYTE bk);
BYTE computePhoneScore(BYTE phoneVector[], BYTE bi, BYTE bj, BYTE bk);
BYTE findTopWordMatches();
BYTE computeWordScore(BYTE word, BYTE startIdx);
void printASCIIWords(BYTE word1, BYTE word2, BYTE word3);
BYTE getPhoneIdx(char* phone);

/*
 * Takes input from the observation input file and computes the resulting
 * phone score and adds this to the score table. Input is a 39x3 vector
 * (if 39 is the # of phones)
 */
void appendPhoneObservationVector(BYTE b[NUM_PHONES][3]){
    //loop through all the phones in the list and compute each phone score
    for (int i=0;i<NUM_PHONES;i++){
        scoreTable[scoreTableIdx][i] =
computePhoneScore(phoneTable[i],b[i][0],b[i][1],b[i][2]);
    }
}

/*
 * Computes a total phone score using a matrix multiply / add
 * from the 3x3 phone weight matrix (phoneVector) and a 3-value input, b
 */
BYTE computePhoneScore(BYTE phoneVector[], BYTE bi, BYTE bj, BYTE bk){
    //perform matrix multiply/add and store result in the provided spaces in the score
table
    return (phoneVector[0]*bi + phoneVector[1]*bj + phoneVector[2]*bk)
        + (phoneVector[3]*bi + phoneVector[4]*bj + phoneVector[5]*bk)
        + (phoneVector[6]*bi + phoneVector[7]*bj + phoneVector[8]*bk);
}

/*
 * Crude, but functional method that tries all possible
 * 3-word combinations and returns the set that produces the
 * highest score.
 * With each combination, the 3 individual word scores are computed and
 * summed.
 * byref variables are filled with the index to the 3 words
 * Returns the last phone index in the score table that it resolved into a
 * word
 */
```

```

BYTE findTopWordMatches(BYTE* bestWord1, BYTE* bestWord2, BYTE* bestWord3){
    long bestScore = 0;
    long tempScore = 0;
    BYTE idx = 0;
    BYTE i,j,k;

    //try each 3-word combination and find the max score
    for (i=0;i<NUM_WORDS;i++){
        for (j=0;j<NUM_WORDS;j++){
            for (k=0;k<NUM_WORDS;k++){
                tempScore = computeWordScore(i,0) +
computeWordScore(j,wordTable[i][0]) +
computeWordScore(k,wordTable[i][0]+wordTable[j][0]);
                if (tempScore>=bestScore){
                    bestScore = tempScore;
                    *bestWord1 = i;
                    *bestWord2 = j;
                    *bestWord3 = k;
                }
            }
        }
    }

    idx = wordTable[*bestWord1][0]+wordTable[*bestWord2][0]+wordTable[*bestWord3][0];
//add up all the lengths to get the last index location
    return idx;
}

/*
 * Computes the word score from the phone score table,
 * assuming the word starts at the specified index (startIdx)
 * word is the index of the word in the wordTable (dictionary)
 */
BYTE computeWordScore(BYTE word, BYTE startIdx){
    int accumResult = 0;
    //for each phone in this word (from the dictionary), lookup the calculated phone
score,
    //starting at the specified index, and add all the scores together.
    for (int phoneIdx=0;phoneIdx<wordTable[word][0];phoneIdx++)
        accumResult +=
scoreTable[startIdx+phoneIdx][wordTable[word][PHONE_START_IDX+phoneIdx]];

    return accumResult;
}

/*
 * Prints the 3 words to the console, using the ASCII
 * characters stored in the wordTable.
 */
void printASCIIWords(BYTE word1, BYTE word2, BYTE word3){
    for (int
asciiIdx=ASCII_START_IDX;asciiIdx<(ASCII_START_IDX+wordTable[word1][1]);asciiIdx++)
        printf("%c",wordTable[word1][asciiIdx]);
    printf(" ");
    for (int
asciiIdx=ASCII_START_IDX;asciiIdx<(ASCII_START_IDX+wordTable[word2][1]);asciiIdx++)
        printf("%c",wordTable[word2][asciiIdx]);
    printf(" ");
    for (int
asciiIdx=ASCII_START_IDX;asciiIdx<(ASCII_START_IDX+wordTable[word3][1]);asciiIdx++)
        printf("%c",wordTable[word3][asciiIdx]);
    printf("\n");
}

#define INPUT_OBS_VECTOR_FILENAME "input.txt"
#define PHONE_MATRICIES_FILENAME "phonematricies.txt"
#define DICTIONARY_FILENAME "dictionary.txt"

/*
 * Parses the input phone weight matricies file and stores it in phoneTable
 */
void loadPhones(){
    printf("Loading phoneme weight matricies...\n");
    FILE* fp = fopen(PHONE_MATRICIES_FILENAME,"rt");
    char pl[3];
    for (int i=0;i<NUM_PHONES;i++){
        if (fscanf(fp,"%n%s\n%d %d %d\n%d %d %d %d\n%d %d %d\n",

```

```

        &p1,
        &phoneTable[i][0],
        &phoneTable[i][1],
        &phoneTable[i][2],
        &phoneTable[i][3],
        &phoneTable[i][4],
        &phoneTable[i][5],
        &phoneTable[i][6],
        &phoneTable[i][7],
        &phoneTable[i][8])!=10){
            printf("Invalid phoneme matrix file.\n");
        }
        phoneTable[i][9] = p1[0];
        phoneTable[i][10] = p1[1];
    }
    fclose(fp);
}

/*
 * Parses the input dictionary and stores it in wordTable
 */
void loadWords(){
    FILE* fp;
    int num_phones, num_chars;
    printf("Loading dictionary...\n");
    fp = fopen(DICTIONARY_FILENAME,"rt");
    /*
    for (int i=0;i<NUM_WORDS;i++){
        fscanf(fp,"\n%d %d\n",&num_phones,&num_chars);
        wordTable[i][0] = (BYTE)num_phones;
        wordTable[i][1] = (BYTE)num_chars;
        for (int j=0;j<num_phones;j++){
            fscanf(fp,"%d",&wordTable[i][PHONE_START_IDX+j]);
        }
        fscanf(fp,"\n");
        for (int j=0;j<num_chars;j++){
            fscanf(fp,"%c",&wordTable[i][ASCII_START_IDX+j]);
        }
        fscanf(fp,"\n");
    }*/
    char line[100];
    char word[MAX_CHARS_PER_WORD];
    char phones[MAX_PHONES_PER_WORD][3];
    int i=0;
    while(1) {
        if (fgets(line,100,fp)==NULL)
            break;
        num_phones = sscanf(line,"%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s", &word, &phones[0], &phones[1], &phones[2], &phones[3], &phones[4],
        &phones[5], &phones[6], &phones[7], &phones[8], &phones[9], &phones[10], &phones[11],
        &phones[12], &phones[13], &phones[14], &phones[15], &phones[16], &phones[17]) - 1;

        //place this word in the phone-word dictionary
        num_chars = strlen(word);
        wordTable[i][0] = (BYTE)num_phones;
        wordTable[i][1] = (BYTE)num_chars;
        for (int j=0;j<num_phones;j++){
            wordTable[i][PHONE_START_IDX+j] = getPhoneIdx(phones[j]);
        }
        for (int j=0;j<num_chars;j++){
            wordTable[i][ASCII_START_IDX+j] = word[j];
        }
        i++;
    }
    fclose(fp);
}

/*
 * Searches the phoneTable and returns the index of the phone
 * whose ASCII spelling matches the string passed in.
 */
BYTE getPhoneIdx(char* phone){
    char phone_from_table[3];
    for (int i=0;i<NUM_PHONES;i++){
        phone_from_table[0] = phoneTable[i][9];
        phone_from_table[1] = phoneTable[i][10];
        phone_from_table[2] = '\0';

        if (strcmp(phone_from_table,phone)==0)

```

```

        return (BYTE)i;
    }
}

/*
 * Runs the program by loading the input dictionary and phone weight matrices
 * and then computing trigram matches from the input observation file.
 */
int main(){
    BYTE lastIdx;
    BYTE reading;
    int input_bi,input_bj,input_bk;

    scoreTableIdx = 0;
    reading = true;

    printf("ECE520 Project: Utterance Matcher\nEric Phillips & Frankie Myers\n\n");
    loadPhones();
    loadWords();
    printf("Loading input observation vectors...\n\n");

    FILE* fp = fopen(INPUT_OBS_VECTOR_FILENAME,"rt");

    printf("Trigram matches:\n");
    while (reading)
    {
        //load in observation vectors..each vector consists of NUM_PHONES * 3
        numbers
        BYTE observation_vector[NUM_PHONES][3];
        for (;scoreTableIdx<NUM_OBSERVATIONS_TO_BUFFER;scoreTableIdx++)
        {
            fscanf(fp,"\n");
            for (int i=0;i<NUM_PHONES;i++){
                if (fscanf(fp,"%d %d
%d\n",&input_bi,&input_bj,&input_bk)!=3)
                {
                    reading=false;
                    break;
                }
                observation_vector[i][0] = (BYTE)input_bi;
                observation_vector[i][1] = (BYTE)input_bj;
                observation_vector[i][2] = (BYTE)input_bk;
            }
            appendPhoneObservationVector(observation_vector);
        }

        BYTE bestWord1,bestWord2,bestWord3;
        lastIdx = findTopWordMatches(&bestWord1,&bestWord2,&bestWord3);
        printASCIIWords(bestWord1,bestWord2,bestWord3);

        //shift all the contents of the score table left
        for (int i=lastIdx;i<NUM_OBSERVATIONS_TO_BUFFER;i++)
            for (int j=0;j<NUM_PHONES;j++)
                scoreTable[i-lastIdx][j] = scoreTable[i][j];

        scoreTableIdx -= lastIdx; //now we start grabbing more phones at the end
of the incomplete word
    }
    printf("\nFile Closed");
    char c = getchar();
    fclose(fp);
}

```