

ECE 464/520 ASIC DESIGN PROJECTS

As Amended Feb 4, 2008

SPRING 2009

Dr. Paul D. Franzon

Amendment list:

Jan 26: Minor changes in the writeup. (See project09_4 to see those changes in edit mode).

Feb 2: Changed table format for 520 project (Not in edit mode).

Introduction

The purpose of the project is for you to demonstrate your skills in designing a substantial digital system using the Verilog/Synthesis techniques taught in class. This document describes the projects and turn-in requirements. These projects are supported by the tutorials prescribed with the class homeworks. If you are in an on-campus section, 001, all projects are to be done in student pairs. If you are an EOL student (ECE 520-601) you do the ECE 464 project by yourself. *EOL students do NOT do a demo.*

Submission Instructions:

- **Project Plan.** Paper report due March 11 in class. Can turn in one week late with a 10% penalty. It must follow the format attached. There is a 10% penalty for not following the format.
- **Project Verilog and synthesis files.** Submitted electronically by 9 am, April 8. Can be turned in one week late with a 10% penalty. Please turn in the following:
 - All Verilog files
 - Synopsys view_command.log file from complete synthesis run
 - Simulation results file showing correct functioning
 - Project report
- **Project Report.** Complete report to be turned in electronically with project files. It must follow the format attached. There is a 10% penalty for not following the format. A hard copy of the cover sheets must be brought to the demo.

ECE520 Project

Utterance Matcher

Introduction

This project mimics part of a Speech Recognition system. Definition and description needed for the project are as follows:

Phone (or Phoneme) - A spoken word or phrase is represented as sequence of basic sound called phone. There are 51 phones in American English.

Example(1) - the word 'This' is phonetically represented as sequence of three phones 'Th' - 'ih' - 's'.

Example(2) ----- 'This is speech' is 'th ih s ih z s p iy sh'

Therefore, a word or a phrase is series of phones placed together.

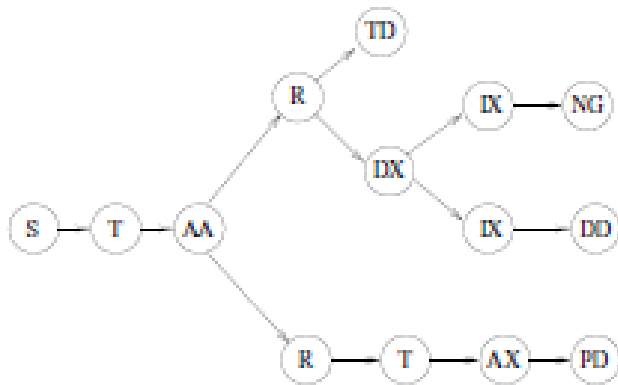
Dictionary – Collection of words represented as sequence of phones.

..... April - Ey-p-r-ah-l Area - Eh-r-iy-ah Fifth - F-ih-th This - Th-ih-s

Simplified Speech recognition

We are going to build a simplified speech recognition system (NOTE: This hides much of the complexity that is present in a full speech recognition system). The input to the system is a set of phones and their probabilities. i.e. The estimated *observation probability* that the speaker actually uttered this phone. For the purpose of this project, the phones will be enumerated from 1 to 51 and the probabilities expressed on a scale from 0 to 99. An example input sequence might be (1, 24), (4, 89), etc. meaning phone 1 observed with a probability of 24%, followed by phone 4 observed with a probability of 89%, etc.

These input vectors and then applied against a dictionary to determine the word sequence most likely being spoken (note in continuous speech, pauses are rare and can not be used to separate words). An example given below is taken from [1]. The diagram shows the phone sets that make up the words, “start”, “starting”, “started” and “startup”.



LEXICAL TREE

We will provide you with a set of words arranged as lexical trees. Each link between phones in the tree is listed with a *transition probability*, again expressed as an integer between 0 and 99. Note that in natural speech, not every phone is uttered, so sometimes there is a probability of a phone in a word being skipped (you can ignore this issue). (Note: this is the transition probability INTO this phone, not to the next phone.)

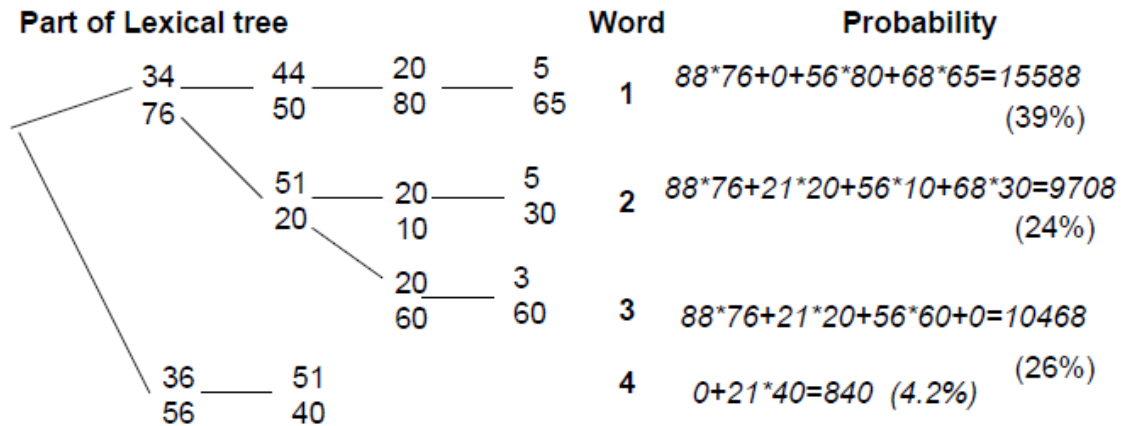
Your task is to work out the **most probable word being uttered**. This is the word with the maximum sum of products of probabilities.

$$\Sigma(\text{observation probability} \times \text{transition probability})$$

An example is illustrated below. In this example, part of an input sequence is presented and part of the lexical tree is also shown. . Since the probabilities are presented as integers (0 to 99 for 0% to 99%), you should calculate them as integers. This makes our life a lot easier – no floating point multipliers are needed! The perfect probability for words 1-3 is $100 \times 100 \times 4$, and for word 4, $100 \times 100 \times 2$, since it is only two phones long. In order to calculate the final % (given in parentheses), the totals have to be divided by the perfect probability for that length of word. Now that we have found the highest match probability for the phone sequence 34-5, the first phone for the next word is 8. Note, if word 4 was a higher probability, then that would be the match, and the next word would start with phone 20 (i.e. “backtracking” is possible).

Input vector:

Phone	34	51	20	5	8	10	45
Obs.Prob.	88	21	56	68	89	65	33



To create a common ground, you will be allowed to use three memories:

- **An I/O memory.** This will come pre-loaded with a 1,024 phone long sequence. You can also use part of this memory to output the word sequence. The last word should be '0' to delineate the end of the sequence. You can NOT use this memory for any other function. This memory is organized as follows: 2048 entries, each 16 bits wide. The first 1024 entries will be the phones, phone number in the first 8 bits, and observation probability in the second eight bits. The remaining entries will be used for words, with 0 written after the last word. **We will be providing you with examples of the contents of this memory as hex files. One example will be provided early. Another during the demonstration to check for correctness. I encourage you to develop and run other examples beside those we provide.**

A dictionary memory. This will be a 32-bit wide, 2048 word deep memory. The first 52 entries (entry 0 is blank) will contain one entry for each phone storing its transition probability and that points to the root of the lexical tree for that phone. Since the address is the phone #, that does not need to be stored.

Since each phone might have multiple daughters in the lexical tree, multiple addresses are needed in the address list. All addresses are RELATIVE – add them to the current address. All relative addresses are positive numbers. The second last entry is 0 to indicate the end of the next phone list. The word # occupies one entry in the memory, For the example given above, the relevant parts of the memory will be organized as follows:

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	phone no.				probability				no. of daughters				offset								first 51 phones											
	phone no.				probability				no. of daughters				offset								non terminal phones											
	0				0				0				word no								terminal phones											

Address	Data for example				
34	34	76	2	100	// root for phone 34 starts at 134
36	36	56	1	200	
134	44	50	1	2	// phone 44
135	51	20	2	4	// phone 51
136	20	80	1	1	
137	5	65	0	0	// 16-bit 0 => word finished
138	0	0	0	1	// word # 1
139	20	10	1	2	// phone 20
140	20	60	1	3	// other phone 20 branch
141	5	30	0	0	// word finished
142	0	0	0	2	// word #2
143	3	60	0	0	// word finished
144	0	0	0	3	// word #3
236	36	56	1	1	
237	51	40	0	0	// word finished
238	0	0	0	4	// word #4

This memory can not be written to (however implement it as RAM as described in the tutorial). We will give you a dictionary of up to 256 words, each of 1 to 10 phones long. Each will be given to you as a 32-bit wide hex file, organized as above. You can not change this memory.

- o **A single working memory.** The size and organization of this is up to you. However, it does count as part of your area.

To summarize:

Memory	Ports	Width x Length
IO	1 Rd, 1 Wr	16 x 2048
Dictionary	2 Rd	32 x 2048
Working	2 Rd, 1 Wr	Up to you

Here are some simplifying assumptions you can use:

1. If, after the first 4 phones in a word, or reaching the end of the word, whichever comes first, the accumulated probability is less than 20%, then this word is not considered a useful match.

Requirements: You must be able to produce the correct match word list for each of the two examples to be given to you.

Goals: Since the purpose of this hardware is to process archives of recorded speech, your goal will be to achieve the best performance per unit area. Performance will be measured in ns as follows:

$$1/(\# \text{ of cycles to process the TWO examples} * \text{clock period used in synthesis})$$

The area will be the total area of the synthesized cells plus the area of the working memory. The other memories are NOT included in the area.

i.e. We are really measuring area / performance

You are also to report the total energy used to process each example, but this will not be taken into account in grading. (It is quite likely that area and energy will be highly correlated).

[1] D. Chandra, "Speech recognition co-processor," PhD thesis, NCSU, 2007.

ECE 464 and ECE 520-601 Project

The purpose of this project is to find matching strings in two serial incoming string sequence,. Your I/O will be as follows:

Inputs

- Clock
- DataIn[7:0]; // Input ASCII character
- Go; // High when DataIn is changing
- Address[3:0]; // addresss to match memory
- Data[31:0]; // data from match memory

Outputs

- Found [31:0]; // Output found match.

You will have a match memory with the interface as specified above. It will contain 16 four-character words, which you will be searching for in the DataIn character stream.

The operation will be as follows:

An ongoing (never ending) stream of ASCII characters will be presented at 8-bit wide DataIn while Go is high. (i.e. If Go is low, DataIn is not changing). You are to find matches against the sixteen 32-bit words stored in the MatchMemory. When a match is found, put that match out on 32-bit wide Found for ONE cycle only (then clear). Note, the match inputs might change from time to time. Note, the words in the Match memory might be less than 4 characters, in which case they are zero padded in the high order bits. If there are overlapping match inputs, then you must find the longest one. The matches might be part of a longer word.

Example:

```
MatchMemory Contents =
    an          32'h0000616E
    fred
    and
    frod
    bat
    batt
    andi
    ande
    free
    bah
    a
    frog
    feed
    fee
    fed
    ant
```

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

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

There are brute force solutions to this problem. A more elegant solution is to use the Aho Corasick algorithm or something like that. You will be measured on performance per unit area using the metric clock-frequency/area. No memories should be used except the match memory, which you can not write to.

General Instructions

Project Plan

Your written Project Plan is due in class on **March 11**. **A specified format for report is attached.** Items to be included in this report include the following:

- Block diagrams of your design, clearly identifying any design hierarchy, all registers, all module I/O. Neat block diagrams are expected, not rough hand-drawn sketches. Include a written description of the function of each module. E.g. For a multiplier, which multiplier algorithm are you planning to use.
- Any appropriate high-level (e.g. C) code showing the algorithm has been correctly captured.
- A project execution plan, including the following:
 - Who is responsible for each module
 - A verification plan. How do you intend to verify that the design works correctly? Who will be responsible for system verification?
- A risk assessment plan. Where are the greatest risks in this project? E.g. What areas of the design are you unsure of? Where are your greatest concerns in executing this project? Note, this section is intended for your benefit (so you can focus your efforts on the parts you are least sure of). Do not expect written feedback on your risk assessment plan. However, feel free to ask questions of the teaching team.
- A milestone chart, including anticipated dates for completing module design, design integration and design verification.

Grading for this report will be as follows (out of 10):

9-10: All major design elements correctly identified; behavioral code complete with test cases. Neat and clear documentation.

7-8: At least one major design element missing, or C code incomplete, or confusing and poor documentation.

5-6: Scrappy, but honest, capturing some elements but conveying no real understanding of the design.

0-4: Extremely poor attempt.

A further 2 points is subtracted if the format below is NOT used.

Do not wait for feedback from us before proceeding with the project. If you need feedback, bring a copy of your report to myself or the TA for discussion. The main purpose of this milestone is to get you going early.

Main Project Report

Your main project report is due on **April 8**. There are penalties for late turn-in. This date is set by the desire to complete grading these before you need to prepare for the final exam.

Your project report is to include the following:

- Written description of your approach, including block diagram of your design, description of your verification strategy and a discussion of any part of the design, synthesis or verification that you consider “tricky”, novel or noteworthy.
 - Specifically note and document your *area* and your *throughput*
- Full listings of the following:
 - Verilog files, including test fixture
 - Synthesis scripts
 - Extracts from View_command.log generated by design_analyzer, including the results from the read command and timing verification. (If you use dc_shell, you will have to rerun the script in design_analyzer in order to obtain this).
 - Plots from the final design.
 - Simulation run results (waveforms or equivalent)
 - High level model of the design, if appropriate

The grading scheme for the final report and demonstration are attached below. Bonus points are awarded **PRIMARILY** on your demonstrated ability to maximize the design goal above. Some bonus points **MAY** be awarded to groups that make a contribution to the class project as a whole, beyond their individual design (this is rare). There are no other ways to earn bonus points.

Other Instructions

- You are **NOT** required to run the verify command in Synopsys. This takes lot of time and adds no educational value to the project (obviously in real life, you would run this command.)
- Make sure to use the 0.18 μm library, so we can compare area and performance accurately.

An SRAM .v file will be provided. Use this for the required SRAM. **DO NOT SYNTHESIZE ANY SRAMS.**

Checklist For Neophyte Verilog Designers

Module Design

1. Did you DESIGN BEFORE CODING?
2. Have you clearly identified the purpose of each register (flip-flop)?

A common beginner error is to have unnecessary flip-flops. (This is usually a symptom of NOT designing before coding). A flip-flop is only needed when data is stored between clock cycles. Excessive flip-flops makes a design LARGE, POWER HUNGRY and SLOWS down synthesis.

3. Do you use non-blocking assignment when assigning to flip-flops?
4. Do you make sure that every variable is only assigned within ONE procedural block or continuous assignment statement?

This will cause a LINT warning from Synopsys.

5. Do you make sure that you have NO feedback within combinational logic.

This will cause “timing arcs” in synthesis. An example is...

```
always@(B or C)
```

```
A=B+C;
```

```
always@(A or E)
```

```
C=A+E
```

Instead A or C must be registered.

6. Do you avoid unintentional latches by having no implied memory in combinational logic?
7. Do you avoid having 2D arrays (register arrays) in your modules?

Put large 2D arrays into your SRAM. Put smaller ones into their own register file module.

8. Did you “right-size” the module?

i.e. Make it as small as reasonable while containing critical paths, sharable logic, registered paths (see 9) and being “reasonable” to understand.

9. Does every path connecting input and output go through a flip-flop?
10. Are critical paths contained within one module, not split across two of them?
11. Did you have any while or FOR loops?

The ONLY valid construct is to use a for loop to iterate through an array of bits. Most attempts by neophytes to use these constructs are invalid.

12. Did you DESIGN BEFORE CODING?

Design Hierarchy

1. Did you avoid “glue” logic in modules containing other modules?
2. Did you bring all signals connecting to things outside your design up through the ports of your “top” module?
3. Did you instance your SRAMs only in your test fixture?

Simulation

1. If saving the variables takes too much disk space, save a .trn file instead
...\$shm_open("count.shm"); \$shm_probe(test, "AS"); (later \$shm_close this).
You can also choose to only save a subset of variables – see the manuals.

Synthesis

1. Did you set “current_design=top” before running “compile”?
2. Did you set “current_design=top” before running “report_area”?
3. If the compile is slow try setting “map_effort = low” and try a much slower clock.
4. Make sure you have a correct synopsys setup file and correct references to design ware, if used. (See <http://www.synopsys.com/products/designware/buildingblock.html> and for the Verilog files, refer to /afs/bp/dist/synopsys_syn/dw/sim_ver)
5. Did you look at the warnings and errors after the “read” statement?
6. Did you look at the timing reports?
7. If reset global signal has a high “fan-out” that is OK. This signal can be asynchronous.
8. If you get cell errors before “compile” they are usually OK. If they remain after compile (do a “report_cell”) then you have problems (usually caused by design ware).
9. Ignore cell conflict warnings after the translate command.

ECE 464/520 Final Project Grading Scoresheet

Name	Student ID	Group Core (80)	Individual Core (10)	Bonus (10)	Total (100)

Bonus Data:

Core Group Points

Requirement	Grading Scheme	Grade
Does the project run the first data set correctly?	35 = yes, with correct timing 25 = no, but almost does (minor problems only)	(35)
Does the project run the second data set correctly?	10 = yes 0 = no (any errors or problems at all earn 0)	(10)
Does the project synthesize without any significant problems?	20 = yes 10 = major problems, that would indicate functional flaws in final chip, including timing 0 = no results obtained from synthesis	(20)
What is the quality of the project report?	10 = High quality report in all regards, professionally presented, conveying all required content 5-9 = Some missing content, incorrect format, or scrappy presentation 1-4 = Major deficiencies	(10)
What is the quality of the Verilog code?	5 = Understandable. Reasonable use of comments. Reasonable test fixture and verification suite. 1-4 = Difficult to understand or follow 0 = almost structureless	(5)
Core Sub-Total (80)		

Written Comments:

Individual Core Points

Requirement	Grading Scheme	Partner 1	Partner 2
Does the partner's knowledge reflect that conveyed in the report, and is that contribution around 50% of the project?	10 = reporting was accurate, and reflected around half the effort 2-8 = reporting was somewhat inaccurate, or contribution was substantially less than half 0 = partner had little true knowledge of personal contribution or contribution was trivial.	(10)	(10)
	Individual Sub Total (10)		

Written Comments:

Submit: Verilog RTL, testbench RTL, final netlist as .db., view_command.log., .saif files, and .spef files (if going for power bonus points), + .txt file with clock name and top module name, (please tar gzip these)

Preliminary report format – Your first page MUST match this format

/10

Project Responsibilities:

StudentID. Name 1:

StudentID. Name 2:

Summary Risk Plan:

Schedule:

Brief Description of High Level Code. Include # lines and performance data.

Brief Description of Mode of operation, including selected algorithms

High level sketch. Add details on the following pages if necessary

Final Project Report First Page. Must match this format (Title)

Project Responsibilities (module names, other assigned tasks): StudentID. Name 1: StudentID. Name 2:
--

Delay (ns to run provided provided example). Clock period: # cycles”:	Area: (μm^2) Logic: Memory:	$1/(\text{delay.area}) (\text{ns}^{-1}.\mu\text{m}^{-2})$
Delay (TA provided example. TA to complete)	Energy: (J)	$1/(\text{delay.area}) (\text{TA})$

Abstract

Abstract should briefly summarize that the hardware function is (remember a future employer might be reading this), what your approach was, and the main results achieved.

Project Title
Student names

Abstract

Repeat this if you want a stand-alone report without the previous page.

Note. This outline is not intended as a rigid structure but as guidance.

1. Introduction

- What hardware is being designed here (remember, you might want to show this to a future employer who has not read the description I produced).
- Summary of key innovations if any claimed
- Summary of results achieved.
- Structure of the rest of this report

2. Micro-Architecture

- Hardware “algorithmic” approach used.
- High level architecture drawing, and description of data flow
- Details on claimed innovations

3. Interface Specification

- Detailed description of interface to your design, to data sheet standards.
- Include a table listing each signal, its width and function
- Include an interface timing diagram

4. Technical Implementation

- Discussion of high level modeling and results achieved
- Discussion of any hierarchy below that
- Discussion, if needed, of detailed implementation

5. Verification

- Description of approach used to verify correctness.

6. Results Achieved

- Throughput, area, energy, etc.

7. Conclusions

- Summary of project and key results

