University of Pennsylvania
Department of Electrical and System Engineering
Computer Organization

ESE534, Spring 2012

Final Exercise

April 2, 2012

FM1 Due: Monday, April 9, 12:00PM
FM2 Due: Monday, April 16, 12:00PM
Final Due: Tuesday, May 8, 5:00PM

Late Policy: The final exercise is due May 8th. The standard late policy for homeworks does not apply. If you turn it in later than 5:00PM on May 8th, we reserve the right to give you **no** credit. This is designed as a **one month** project. The milestone due dates (FM1, FM2) are designed to encourage you to to get started on this now and properly pace your progress. The May 8th due date gives you flexibility—it is **not** a signal that you should start this assignment later than assignments that have earlier due dates. You can always plan to finish this one earlier to be compatible with your other deadlines. Normal late policy applies to FM1 and FM2.

Overall goal: Design an energy-efficient programmable architecture for lossless data compression and evaluate on a simple Lemple-Ziv (LZ) [4] compression algorithm.

1 Setup

We identified compression as a way to reduce the amount of data that needs to be sent across an energy expensive interface (*e.g.* an off-chip data bus, long wires crossing from one side of the chip to another, over a wired or wireless network link). Consequently, energy efficient compression might be an important building block to have on a future computing component. Nonetheless, exactly which kind of compression and how to tune it might be very application and data dependent (*e.g.* [1]). This suggests a need for a programmable architecture that can be used to implement compression.

We can imagine that tomorrow's highly programmable chips contain a heterogeneous computing fabric built from a mix of various kinds of programmable computing arrays (*e.g.* processor cores, FPGA blocks, GPUs). As such, it might be viable to customize portions of the array for the compression engine. It would, however, also be good if the resources for the compression function were usable for other tasks when they weren't needed for compression. To motivate the inclusion of some programmable architectural components tuned for compression, we should demonstrate that they have significant energy savings over simply using processor cores or FPGA arrays.

2 Problem Specification

- **FM1 (Final Milestone 1):**
  1. Design and estimate the energy required for a μcoded implementation for the tree-based (Section 5.1) LZ encoding on a processor-like architecture (See Section 6.1). You may add instructions and tweak the implementation to customize for the compression problem.
     (a) Draw your resulting architecture.
     (b) Develop an energy model for your architecture. In addition to other parameters, this should be parametrized by the window size, D.
     (c) Describe the specific algorithm you are implementing.
     (d) Provide the instructions required to implement the tree-based LZ on your processor architecture.
     (e) Based on your model and the technology constants provided, estimate the energy required per byte encoded when $D = 1024$.
  2. Assuming you use this basic processor-style architecture, how could you optimize it to reduce energy on this problem? We suggest you provide at least three ideas. More are welcomed. It will probably be useful to analyze where energy is going in your design and think about how the major sources of energy might be reduced. Describe each idea in a few sentences, using simple equations for trends as appropriate. The point of this is to get you started thinking about opportunities for optimization.
- **FM2 (Final Milestone 2):**
  1. Design and estimate the energy required for an FPGA implementation of the systolic-style (Section 5.2) LZ encoding. Use an architecture similar to the FPGA on HW9 with memory banks (See Section 6.2).
     (a) Draw your resulting architecture.
     (b) Develop an energy model for your architecture.
     (c) Describe your LZ design and show how it is mapped on top of your architecture.
     (d) Based on your model and the technology constants provided, estimate the energy required per byte encoded for a window size $D = 1024$.
  2. Assuming you use this basic FPGA-style architecture, how could you optimize it to reduce energy on this problem? We suggest you provide at least three ideas. More are welcomed. As with FM1, provide a few sentences and trend equations as appropriate for each idea.
  3. Now that you have explored how conventional designs support this computation, identify ideas for an architecture to minimize energy and the parameters you may need to explore to find the most efficient architecture.

(a) What optimization techniques might be relevant and worth exploring? We recommend you try to come up with a large list, then try to prioritize them by likely impact.

(b) What architectural features do these techniques suggest?

(c) Describe the parameters you may need to explore to optimize your design.

This problem is specifically intended to get you started developing ideas for the architecture you will design for the final. It also provides an opportunity for the instructor to give you feedback on your preliminary ideas.

- **Final:**

  1. Describe the parametrized architecture you have chosen.
  2. Develop an energy model for your parametrized architecture.
  3. Identify the parameter settings that minimize energy and report the associated energy to encode each byte.
  4. Compare this result to the processor- and FPGA-based designs you developed for FM1 and FM2.
  5. Writeup your results as described in Section 3.

3 Final Report Format

Please provide your solution in the form of a memo to an engineering group manager.

- Summary [No longer than 4 pages] including:
  - One paragraph overall recommendation.
  - Diagram for each of the three architectures.
  - Table describing the parameters used in your architecture and identifying the parameter value you selected. This should include references to graphs in the appendix that support the selection of the specific parameters.
  - Table comparing the energy of the three architectures for three different window sizes ($D = 512$, $D = 1024$, $D = 2048$)
  - Prose description detailing why and how your architecture is able to save energy.
- Consultant log [1 page] summarizing all interactions with other students and advisers. (Date/time, length of session, participants, topics discussed)
- Please include a statement on your final submission:

  I, *your-name-here*, certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this final exercise.

  You can review the Code of Academic Integrity here: `http://www.upenn.edu/academicintegrity/ai_codeofacademicintegrity.html`
- Appendix [as long as necessary] showing the derivation and composition of each model and selection of parameters. This should include everything requested for the Final. It may include FM1 and FM2 details. It may include refined designs from FM1 and FM2 if instructor feedback or your own insight leads you to refine the designs.

4 Collaboration

Informally: you may discuss strategy and architectural parametrization in groups through the last day of classes (April 24th). All detailed analysis and writeup, for both milestones and final, must be done individually, and no collaboration is allowed after April 24th.

More precisely:

- Each person must individually prepare his or her own solution memos and milestone assignments.
- After April 24th, you should not discuss the project and solutions with anyone. (You may ask clarifying question of the instructor after April 23rd — nonetheless, the instructor will be traveling starting April 23rd, and will be slow with answers. He will be more generous with answers and discussion before April 23rd than after.)
- Before April 24th, you may discuss:
  - base μcoded and FPGA architectures
  - general design strategies for mapping onto μcoded and FPGA architectures
  - techniques and architecture for minimizing energy
  - diagrams and equations on a white-board, napkin, and/or pencil and paper
- Your writeup must document your discussion groups and sessions.
- You may not:
  - develop final analysis equations with others
  - share analysis code
  - share final diagrams
  - have a joint coding or drawing session for analysis and diagrams
  - show others your computer-drawn diagrams or computer-rendered analysis

5 Lempel-Ziv Compression Algorithms

It's possible there are bugs in the fine details here. I will continue to review. Let me know if you find clear errors.

The basic idea in a Lempel-Ziv compression is to use a window into the data already transmitted and code substrings relative to that data. A key parameter in the algorithm is the size of the window.

A rough version follows:

While there is still data to send:

- Find the longest match between the data in the window and the data you need to send next.
- Send a description of that match (e.g. the address of the match and the associated length)
- Update the window to include the data you just sent.

Two variants worth considering are the tree-based version [2, 5] and the systolic version [3, 4].[1] Both of these papers talk about using CAM, but they use it in different ways. These two variations store data slightly differently.

5.1 Tree Version

The tree version [5] effectively stores a tree of string prefixes. It always keeps around a tree for each possible starting character in order to initialize the process and make sure we can always encode every string.

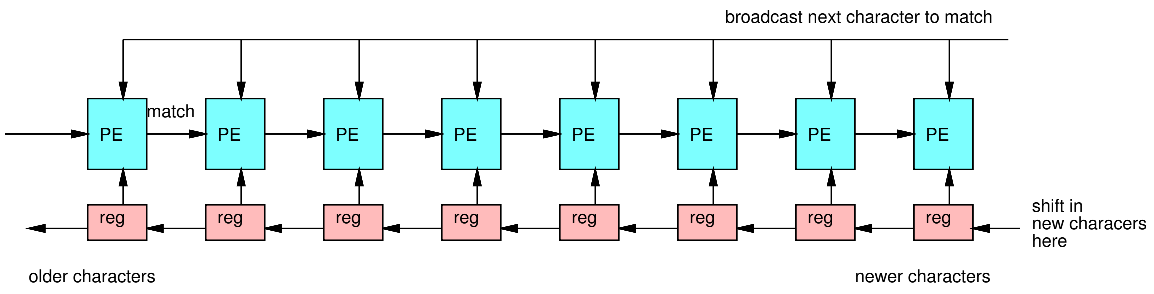
```
1. Start with the first character to send
2. current-tree-node = tree in window that matches this first character
3. while (next input character is a child of current-tree-node)
   (a) current-tree-node=associated child node
   (b) get next input
4. // when it exits the loop, the current-tree-node does not have a child matching the next
   input character
5. send id associated with current-tree-node // this encodes the entire substring down to
   the point of mismatch
6. add a child of current-tree-node for next input character // this extends the tree by
   one; so if this longer substring is encountered again, it can be sent.
7. make next input the first character and go to step 2 // this makes the next input the
   beginning of the next sequence we will encode.
```

We also have to deal with the window filling up. [2] shows example trees and describes solutions for dealing with finite window memory.

[1] We're not worried about dealing with errors for this assignment, so you can ignore that part of [3].

5.2 Systolic Version

The systolic version [4] simple keeps a window of the last D characters sent. For simplicity, imagine a chain of PEs each holding a character.


broadcast next character to match
match
PE
reg
shift in new characers here
older characters
newer characters

1. Start with the first character to send
2. Broadcast the character to all PEs with an indication this is a new match *// all PEs participate when it is a new match*
3. Each PE: if the character matches, send a match signal out to the PE to its right neighbor (the PE that represents the next most recent character)
4. $L = 0$ *// length of match*
5. While at least one of the PEs is indicating a match
    (a) broadcast the next character to all PEs indicating this is a match extension
    (b) Each PE: if the PE received a match indication from its left (previous) neighbor PE and the character matches, send a match signal out to the PE to its right.
    (c) $L = L + 1$
6. *// when it exits the loop, there is no match up to the final character*
7. Send the address of the first PE that received a match from its left but was not able to match the current broadcast character along with the length of the match, L.
8. Shift the set of characters just matched into the array of PEs (left shift).
9. Broadcast a signal telling all PEs to now treat the last (unmatched) input as a new match and go to step 3.

In this case, we must also add provision for their being no match in the window. In these cases, we will send the character unencoded. This means we will have to have a way to distinguish when we are sending a position into the array and when we are sending a literal. The simplest version is to reserve one bit for this. [3] gives examples of this scheme and describes two ways of implementing this, one using a particular kind of CAM. While it calls one systolic and the other CAM-based, the are similar in that they keep this sequential window of previous characters.

We've deliberately tried to describe relatively simple versions of these algorithms here. You will find many variations in the literature. You are welcome to tweak the algorithm or use a variant of a more sophisticated version from the literature. If you make large changes, you may need to simulate it to validate that it has comparable compression capabilities. In any case, you will need to describe the algorithm(s) you use, the associated parameters, and your selection of the parameters.

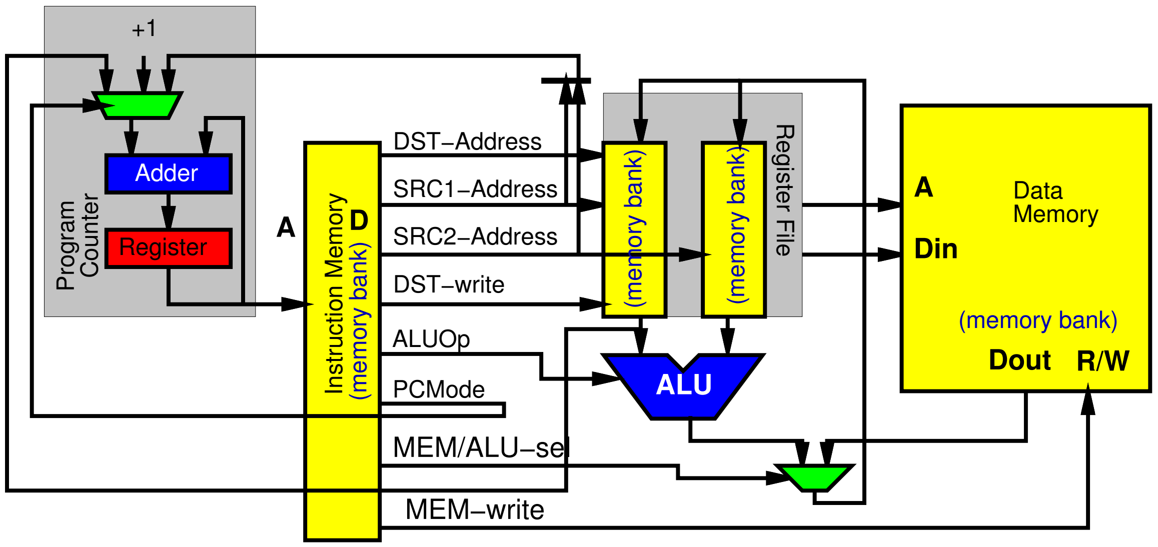
Wikipedia:

- `http://en.wikipedia.org/wiki/LZ77_and_LZ78`
- `http://en.wikipedia.org/wiki/Lempel-Ziv-Welch`

6 Architectures

6.1 Processor Architecture with Memory

The processor architecture is an extension of the one used in HW4.



Here we add a main data memory as a single, monolithic bank to hold the window of data for encoding (*e.g.* the encoding trees). This is a load-store architecture. You can either transfer a value from the memory on a cycle or use the ALU to perform arithmetic. You will need to add suitable instructions to access the memory.

Model the memory banks (including instruction memory and register file memory banks) according to the model you developed from HW7 (See Section 7).

6.2 FPGA Architecture with Memory Banks

Use the clustered 4-LUT architecture from HW9 ($k = 4$, $n = 10$, 22 inputs to the CLB). Assume full crossbar connectivity between the 22 inputs + 10 outputs and the 4×10 inputs to the 4-LUTs (so a 32×40 crossbar).

For simplicity, assume you can replace any of the CLB clusters with a memory bank that requires fewer inputs than the 22 inputs to a CLB and fewer outputs than the 10 outputs from a CLB. This way, the I/O for the memory bank remains the same as the CLB so it integrates into the same mesh network. For the preliminary (FM2) version, the maximum memory capacity is 2048 bits. You may also select the fraction of CLB clusters you replace with memory banks. The input bits will need to be able to specify R/W control (1 bit), Address, and Data.

We select 2048 bits above as (crudely) being comparable in size to the LUT cluster. For the final, you are defining your own architecture and may rethink all of these sizes as long as you model them.

7 Technology Model

V_{dd}=0.75V.

LUT: Model the 4-LUT as an 16:1 mux followed by a register and 2-input mux to select between the registered and unregistered output.

Register: $C_{reg} = 4\times 10^{-16}$F. This is the total effective capacitance for the register, including input, internal, output, and clock load.

Interconnect: Model interconnect according to HW9.

Crossbar/Mux: We will model crossbars and muxes equivalently based on their inputs, outputs. That is, an N-input, M-output crossbar is M N:1 muxes.

- The capacitive load on each data input is 2×10^{-16}F.
- The capacitive load on the output of the mux is 2×10^{-16}F.
- The capacitive load on each control input is $N \times 10^{-16}$F.
- The capacitive load switched internal to the mux is 2×10^{-15}F.

Memory: Model memory banks according to HW7 (corrected as necessary based on feedback). Use ITRS 2015 (21nm half pitch) LOP process node for $C_{g,total}$ and other parameters as necessary. If you use multiported memories, extend your model to deal with the additional word and bit-lines. Multiplying the memory energy by the number of ports, p, is an acceptable level of modeling.

Wire: For the most part, let's make the simplifying assumption that wire capacitance does not dominate other capacitances. However, if you do have long wires in your architecture—either because you must wire across large PEs or because you have wires that span multiple PE widths, model them similar to HW9 with $C_{metal} = 3.0 \times 10^{-15}$F per 30,000nm.[2] To keep the scope of this down, we're not generally asking you to calculate areas. In practice areas determine these wirelengths and could be a non-trivial component of capacitance. If you find yourself building large structures (memory banks significantly larger than the 2048b memory mentioned for the FPGA), it may be necessary to make some estimates.

Memory: If you do need to make crude estimates of the sizes of large memories, use the HW7 area model:

$$(A_{mem}(d\text{-entry}, w\text{-bit wide}) = A_{bit}\left(d(2\log_2(d) + w) + 10w\right) \tag{1}$$

For a p ported memory cell, use the Day 20 model:

$$A_{bit}(p) = \left(250\text{F}^2 + p \times 125\text{F}^2\right) \tag{2}$$

If you need additional information about the technology:

- Estimate from ITRS 21nm LOP.
- Propose a suitable model and run it by the instructor.
- As a last result, ask for the instructor suggestions for where to get started on a model **before** April 23rd.

In any case, document your assumptions, values, and models in your writeups.

[2]My ballpark estimate for the side length of the $k = 4$, $n = 10$ CLB cluster in the 21nm process.

References

[1] Kenneth Barr and Krste Asanovic. Energy aware lossless data compression. In *Proceedings of the The First International Conference on Mobile Systems, Applications, and Services*, 2003. `http://groups.csail.mit.edu/cag/scale/papers/compression-mobisys2003.pdf`.

[2] Suzanne Bunton and Gaetano Borriello. Practical dictionary management for hardware data compression. *Commun. ACM*, 35(1):95–104, January 1992.

[3] Wei-Je Huang, Nirmal R. Saxena, and Edward J. McCluskey. A reliable lz data compressor on reconfigurable coprocessors. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 249–258, 2000.

[4] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.

[5] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, September 1978.