University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE532, Fall 2017

Design and Function Milestone

Wednesday, November 1

Due: Friday, Nov. 10, 5:00PM

Group: Develop functional code. Identify design space options.

Individual: Writeup is an individual task.

1. Develop a functional implementation for the project task that can run on the Zynq ARM and produce a valid compressed output stream that works with the supplied decompressor.
   - we recommend you aim for complete functionality with reasonable compression for this milestone
   - however, we are not requiring that all pieces perform their full final operation this week; they should work well enough to process data and produce a valid output stream, but may have reduced functionality
   - examples of reduced functionality
     - create fixed-sized blocks instead of good chunks
     - never identify a chunk match; so send all chunks to LZW
     - send all characters within a chunk as literals rather than using substring matches
   - this technique of using a minimum functionality placeholder so that you can assemble a complete flow early then incrementally upgrading components to more complete functionality is generally a good one, and we suggest you use it during your development on the way to full functionality, even if you can achieve full functionality for this milestone.
2. Writeup on functional implementation should include
   (a) Status of all operations (how much functionality is there, what functionality is missing)
   (b) Code sources (e.g., URLs) for any open-source code you used as a starting point or as a primary reference
   (c) Current compression status and breakdown of contribution from deduplication and from LZW compression
   (d) Description of all validation performed on your current functional implementation.
3. Turn in a tar file with your functional code to the designated assignment component in canvas.

4. Identify major design space axes that could be explored for your implementation.
   - For this milestone, aim for breadth (quantity of options)
   - Each axis description can be 1–3 sentences. Identify challenge being addressed, basic solution opportunity, and continuum.
   - Include a simple equation to illustrate ideal benefit (e.g., running N tasks in parallel reduces runtime by a factor of N; $T(N) = T(1)/N$).
   - Cover all operations that must be accelerated.
   - Aim for at least 6 axes per function.
   - Some of this should build on the parallelism opportunities you identified on the previous milestone.

Chunk Validation

Using an SHA-256 signature, the probability of having a collision where two chunks share the same signature is extremely low. For the project, we will consider equality of SHA-256 signatures adequate to determine that a chunk is a duplicate. This means you do not need to read back the chunk and validate that it is, in fact, identical. If you had terabytes of data, or if the consequences of error were high, you would want to perform the check. This only applies to the full 256b signature. If you use smaller hashes for indexing, you will still need to validate that there is a match on the 256b signature.

Compressed Format

- Compressed stream is a sequential concatenation of chunks.
- Each chunk has a 32b header that identifies it as Duplicate Chunk or LZW Chunk.
  - A Duplicate Chunk is a 32b value
    - bit 0 is a 1 to signify a Duplicate Chunk
    - bits 31–1 is the Chunk Index of previously encoded block to be duplicated. Only LZW Chunks are indexed. The first LZW chunk has index 0, the next 1, etc.
  - An LZW Chunk is
    - a 32b bit header
      - bit 0 is 0 to signify an LZW Chunk
      - bits 31–1 is the *compressed* chunk length in bytes
    - LZW-compressed contents of the chunk. LZW implementations vary. Our implementation satisfies the following properties:
      - Entries 0–255 of the dictionary are initialized to the 256 literals, e.g. a byte with value 27 would be encoded as 27.
      - The next dictionary entries are used for prefixes: sequences of 2 or more bytes.
      - No special keywords such as end-of-file are contained in the dictionary.
      - The dictionary size is only limited by the chunk size limitatation.
      - The code length depends on the size of the dictionary at the time an input sequence is encoded. When the dictionary has N entries, the code words are $\lceil \log_2 N \rceil$ bits.
      - Code words are output MSB-first. Assuming nothing has been output yet, a 9b code with binary value $x_8x_7x_6x_5x_4x_3x_2x_1x_0$ results in two consecutive bytes with values $x_8x_7x_6x_5x_4x_3x_2x_1$ and $x_0$0000000. The next code word with value $y_8y_7y_6y_5y_4y_3y_2y_1y_0$ changes the second byte to $x_0y_8y_7y_6y_5y_4y_3y_2$ and the third to $y_1y_0$000000.
    - Padding so that the entire LZW chunk ends on an 8b boundary; that is, chunks of either type always begin on 8b boundaries.

Supplied Resources

- Reference implementation of decoder
- We provide several datasets that you can use for testing. We encourage you to create your own simple datasets for unit testing. Note that the tar-files are not meant to be unpacked. Following are the datasets that we provide.
  - Simple example. This archive contains three files, two of which are identical. An encoded version is provided here as well.
  - Benjamin Franklin's autobiography. This is a simple text file that you can modify for your own purposes. The current file probably has few duplicate areas. (390 KB)
  - GTK+ source code. This file contains several subsequent versions of the GTK+ source, which provides ample opportunity for deduplication. (177 MB)
  - Linux source code. This file contains several subsequent versions of the source. (191 MB)
  - Several Linux kernels. As opposed to the other data sets, this set contains prevalently binary data. (66 MB)

  Note: you should take these as examples, not a definitive list of test cases. In particular, you should create many other focused test examples to facilitate your debugging and validation.
- You can use the compression pipeline of the homeworks as an example of how to access the SD-card.