CSEE 3827: Fundamentals of Computer Systems

Project #6

Huffman Decoder with Cache

Due 12/14/17 at 11:59pm

1 Specification

Your objective is to design a cache for the Huffman Decoder from P3. The tree is encoded exactly as in the previous assignment. The only change here is that the TREE_ROM is now a SLOW_TREE_ROM. It accepts addresses via a latency insensitive channel and, ten cycles later, provides the corresponding data.

Interface The HUFFDEC module has four latency insensitive interfaces:

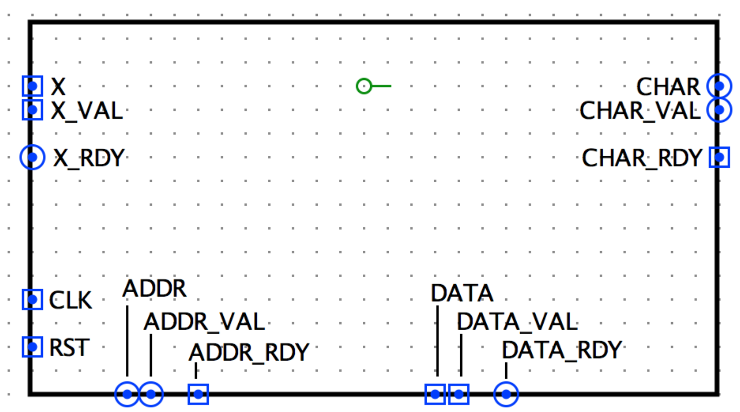
- **X** is one bit wide and streams encoded bits from the test harness to HUFFDEC,
- **CHAR** is 8 bits wide and streams decoded characters out of HUFFDEC to the test harness,
- **ADDR** is 8 bits wide and streams addresses from HUFFDEC to SLOW_TREE_ROM,
- **DATA** is 17 bits wide and streams read data from SLOW_TREE_ROM to HUFFDEC.

Baseline Design The baseline design is provided in the scaffolding. It functions with the SLOW_TREE_ROM, but makes no attempts at caching the data extracted from the slow memory.

2 Test Harness

The test harness is identical to that of P5, except that latency_mode has been removed.

Note the original interface to HUFFDEC. Your submission must match this exactly.



3 Workloads

Huffman codes are based on character frequency, using short codewords for frequent characters and longer codewords for infrequent characters. The trees we will use are not laid out in memory with any eye towards locality. Each workload consists of a tree and set of characters, with the characters following a uniform random distribution. The workload provided in the scaffolding includes the same tree as in P3 and a sequence of 100 characters to decode.

4 Scoring Process and Rubric

To be scored, your submission must:

- match the original name and appearance of HUFFDEC,
- use only builtin Memory compoents to hold state (no buiding your own flip-flops),
- not exceed 2048 bits of total state in HUFFDEC,
- halt without error (e.g., oscillations or undefined wire values), and
- be uploaded in a single circ file via courseworks.

We will first perform a functional check. This will require decoding ten characters using the tree provided in the scaffolding. If a submission fails either of the following two tests, we will proceed no further, and it will receive a zero.

- $NUM\_PASS_{\overline{perf\_mode}} == 10$
- $NUM\_PASS_{perf\_mode} == 10$

Submissions that pass this check will be assessed for function and performance on sequences of 1000 characters. We compute an average speedup across five different input sets and trees. Each of these five tests will use a different Huffman code and tree.

We measure the submission's speedup relative to the baseline, using a modified formula that admits some incorrect outputs:

$$Speedup = \frac{CYCLES_{baseline}}{CYCLES_{submission} + 20 \cdot (1000 - NUM\_PASS)}$$

All values are measured in perf_mode.

We will assess the *resource* usage of your design by counting $StateBits$, the total bits of state contained in HUFFDEC.

	Total	Formula
Score	100pts	If $Speedup < 1$, 0 else, $\lceil [500] \cdot \frac{log_2(Speedup)}{log_2(StateBits)} \rceil$
Style	10 pts	If $Score > 50$, the usual extra credit for most beautiful schematics.