1. Overview

Computers often store data using a compression algorithm to save space in memory. In this project, you will be implementing a hardware-based method of encoding and decoding data using dictionaries.

1. Instructions

* Download the collateral provided on Canvas and use the skeletal files provided. Do not change the name of the files.
* Add your full name and UIN to the introductory comment present in each .hdl file.
* Implement the .hdl for each chip.
* Test the basic chips using the completed .tst and .comp files provided.

1. Chips

You may find the starter files for the chips to be constructed in this project in [something]

Build all of the chips in the list below.

**Chips:** Implement .hdl for the following (complete .tst and .cmp files are provided)

|  |  |  |
| --- | --- | --- |
| **Chip Name** | **File Name** | **Description** |
| DictionaryEncoder | DictionaryEncoder.hdl | Encodes input using dictionary |
| DictionaryDecoder | DictionaryDecoder.hdl | Decodes encoded input using dictionary |

1. Proposed Implementation Sequence
2. Implement **DictionaryEncoder.hdl** using 2 Rom32K with Dictionary
3. Implement **DictionaryDecoder.hdl** using 1 Rom32K with Dictionary Inverse
4. **How Things Work**
5. DictionaryEncoder:   
   Design a chip with 3 Inputs: Recur[3], InA[16], and InB[16].

* Recur stores encoded value for how many times input pattern is repeated. Value ranges from 0-7.
* InA stores the first 16-bit pattern to be encoded.
* InB stores the first 16-bit pattern to be encoded.

Dictionary is loaded into both Rom32k Chips, and the address value for each is set to InA and InB respectively.

If either rom32K returns a 0 (default), this means encoding has failed, and output will be set to InA.

If both encodes are successful, output will be stored as

1010101110101011

Where:

out[15] (Blue) stores a 1 if encoding was successful, 0 if not

out[12..14] (Red) Stores the number of pattern repetitions, in this case, 2

out[6..11] (Green) Stores the encoded value of InA

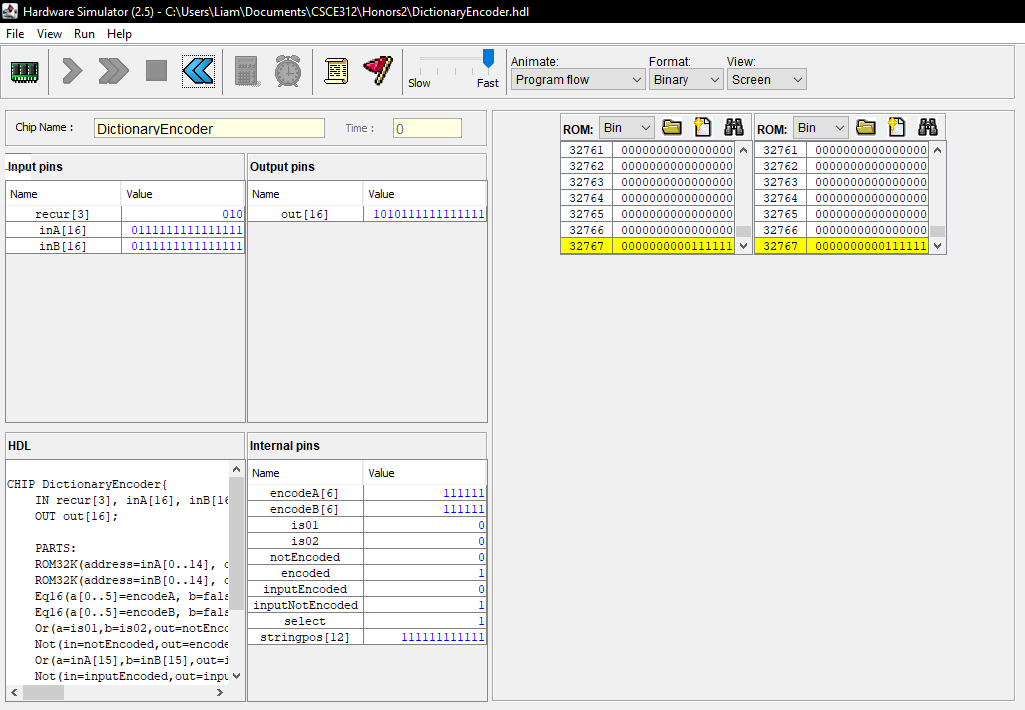
out[6..11] (Purple) Stores the encoded value of InB

1. DictionaryDecoder:

Design a chip with 1 16-bit input.

1. If In[15] = 1, signal is encoded and should be decoded, otherwise outA should be set to In, and outB should be set to 0.
2. In[0.5] is sent to Rom32K, which stores the Dictionary Inverse. Like before, an output of 0 means decoding has failed, and output should be the same as if signal was not encoded.
3. In[12..14] stores the number of recursions. This chip only has 2 outputs, so one must only check if In[12..14] = 1 or 2.
4. If 0, Input is decoded and sent to outA only, sending 0 to outB, outC, outD.
5. If 1, Input is decoded and sent to both outA and outB.
6. If 2, Input is decoded and sent to both outA, outB, outC, and outD.
7. Dictionary and Inverse:
   * Dictionary and Inverse are obtained by Running **DictionaryCreator.py** with **DictionaryCreatorInput.txt** as an argument. This text file teaches the Dictionary and Inverse what data to store for efficient compression. Default output files are Dictionary.hack and DictionaryInv.hack
8. Other Files:
   * There are 2 other Python Files in sources. These are DictionaryEncoder.py and DictionaryDecoder.py
   * **DictionaryEncoder.py** uses the same method of compression as your **DictionaryEncoder.hdl** to compress input data. **It also shows the size difference between the input and output**. Looking at this file may give clues on how to implement DictionaryEncoder.hdl.
   * **DictionaryDecoder.py** uses the same method of compression as your **DictionaryDecoder.hdl** to decompress input data. The output matches the input file of DictionaryEncoder.hdl.
   * The data in DictionaryCreatorInput.txt is the same as in DictionaryEncoder.tst, the only difference is representation in ASCII/Hex form for inputs.

Notes:

* Valid inputs for DictionaryEncoder must not start with a 1. This is for 2 reasons: Rom32K has only 15 address bits, and a 1 indicates an encoded signal. If input starts with a 1, encoding should automatically fail and InA should be returned.
* Valid inputs for DictionaryDecoder must start with a 1 for the same reason as before.
* **VERY IMPORTANT:** In order to run DictionaryEncoder.tst properly, one must first load the script, and complete one step.
* Following this, View must be set to screen, and each of the 2 ROMs must be set to Dictionary.hack manually, since this cannot be done from the .tst.
* 
* Following this, the rest of the script file can be run.

Examples:

* DictionaryEncoder:
  + Recur[3] = 010  
    InA= 0100000101000010

InB= 0100000101000010  
Dictionary:  
0100000101000010-> 000010  
0100000101000010 -> 000010  
Out = 1 010 000010000010

* DictionaryDecoder:
  + In = 1010000010000010  
    DictionaryInverse:  
    0000010 -> 0100000101000010

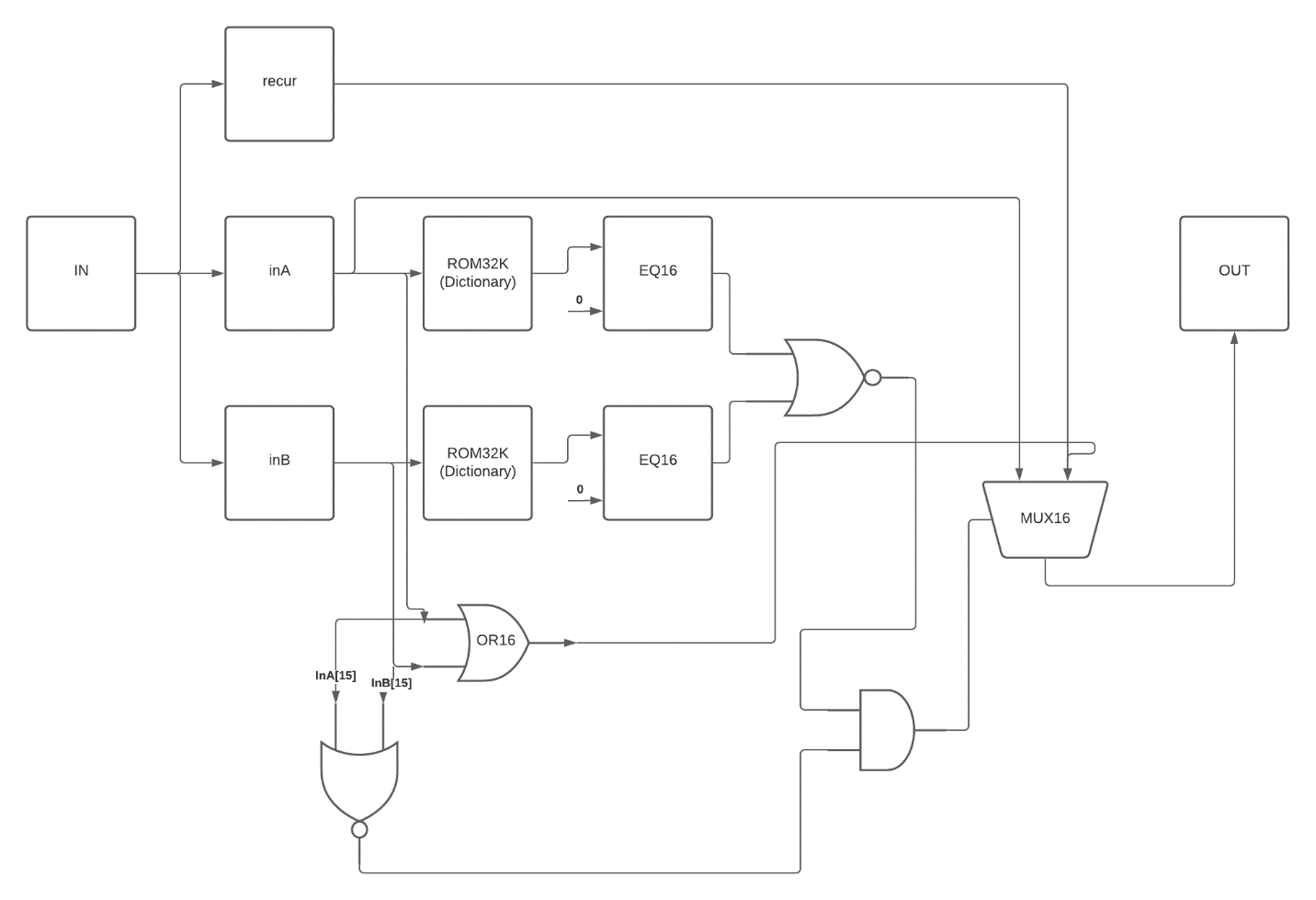
Recur = 010

OutA = 0100000101000010

OutB = 0100000101000010

OutC = 0100000101000010

OutD = 0100000101000010

Logic Diagrams:   
DictionaryEncoder:

DictionaryDecoder:

