Largest Number Divisible by 7 (largest_num_divisible_by_7.v, largest_num_divisible_by_7_tb.v):

Given an array of 16 unsigned numbers find the largest nonzero number (MAX) that is evenly divisible by seven (7 = 0111). There may be zeros among the 16 numbers. The array is held in a 16x8 memory M[I]. A 4-bit counter I produces index (address) I into this array (memory).

Here, since we are looking for non-zero numbers only, we initialize the MAX register to zero in the INITIAL state. In later states, we process only those numbers which are larger than the current MAX. At the end you need to go to DONE_NF (done, not found) state if you could not find any non-zero divisible-by-7 number, otherwise you go to DONE_F (done, found) state. MAX = 0 can be used in your state transition conditions.

In the LOAD_X state, we load the next M[I] into X but we will try dividing it only if that M[I] is greater than the current MAX as it is wasteful (to check if it divisible by 7), if a larger divisible-by-7 number was previously found and stored in MAX already. We use our standard repetitive subtraction method (X <= X - 7) for dividing X by 7 in the DIVIDE_n_UPDATE_MAX state. We will update MAX with M[I] if it is found to be divisible by 7.

All states and state transition arrows are in place in the incomplete state diagram on the next page.

You need to
(a) write the needed RTL in the two states, LOAD_X and DIVIDE_n_UPDATE_MAX
(b) write the needed state transition conditions for the 4 diverging arrows, diverging away from each of these two states.

Carefully decide, when you want to increment I, when you want to update MAX, and when you want to quit. While the given incomplete state diagram leads to one solution, there may be several other equally good solutions. So if you want, you can add additional states and state transition conditions as long as you are not spending more clocks or more hardware. You can not, for example, compare with the test number with all multiples of 7 (7, 14, 21, 28, ...). You can only compare with zero or MAX or 7. First answer the following questions to guide you in your design.

1.1 Partial designs of two students are shown on the side. The only difference is the state transition condition. Out of the two you prefer ________________ (M[I] > MAX / X > MAX). Do you think that they should have used >= (greater than or equal) rather than > (greater than). YES / NO

1.2 In LOAD_X state, you would like to increment I ____________________________ (conditionally / unconditionally / never).

1.3 In DIVIDE_n_UPDATE_MAX state, you would like to increment I ____________________________ (conditionally / unconditionally / never).
1.4 Now complete the state diagram.
2 (16 points) 25 min.

2.1 Make A close to B (verilog files: make_A_close_to_B.v, make_A_close_to_B_tb.v)

Given two 12-bit unsigned numbers, A and B, (A is less than B initially), we need to increase A to come close to B (but not greater than B). A can become equal to B but the final A should not exceed B.

The datapath provides for either adding $100_{10}$ (hundred decimal) to A ($A \leftarrow A + 100_{10}$) or subtracting $10_{10}$ (ten decimal) from A ($A \leftarrow= A - 10_{10}$). The suggested method is to add 100’s until A becomes equal to B or slightly exceeds B. If it exceeds B, then we subtract 10’s until A becomes equal to B or it gets slightly below B. Make use of a flag F (single-bit flip-flop, which is initially cleared) to set it to remember or record an event such as "added enough hundreds and now it is time to subtract tens".

We do not want you to waste any clocks in the ADJUST state! You have only one comparator to compare A with B and produce $A > B$, $A = B$, and $A < B$. So you can not write, "if $A + 100 > B$" or "if $A - 10 < B$".

Examples:

- $A_{in} = 138; B_{in} = 312$;
  
  Then $A = 138; 238, 338, 328, 318, 308$; In DONE, A is 308.

- $A_{in} = 112; B_{in} = 312$;
  
  Then $A = 112; 212, 312$; In DONE, A is 312.

- $A_{in} = 132; B_{in} = 312$;
  
  Then $A = 132; 232, 332, 322, 312$; In DONE, A is 312.
**3 \((17+28) + 16 + 2 + 3 = 66 \text{ points} \) 45 \text{ min.}\)**

Copying two parts of a sorted array:

You are given a sorted array \(M\) of **ten** 4-bit unsigned numbers. \(M[0]\) is the smallest and \(M[9]\) is the largest. We need to copy the elements of array \(M\) to array \(N\). However, while in array \(N\), every number is treated as a signed number represented in 2’s complement notation. You know that sorting will be different for signed numbers. Actually it is interestingly not that different. For example, \(1110\) is always higher (or greater) than \(1100\).

\(1110 = 14\) in unsigned numbers; \(1110 = -2\) in signed numbers
\(1100 = 12\) in unsigned numbers; \(1100 = -4\) in signed numbers

So, if you consider the original array \(M\) of unsigned numbers as made up of two separate chunks of sorted numbers, first chunk of numbers starting with "0" in MSB and the second chunk of numbers starting with "1" in MSB, then to resort and arrange them in array \(N\), we need to simply copy the second chunk (of array \(M\)) as the first chunk (of array \(N\)) and then copy the first chunk (of array \(M\)) as the second chunk (of array \(N\)).

Note: the array \(M\) can have only one chunk (i.e. all numbers starting with either "0" or "1"). "I" is the index into memory array \(M\). "J" is the index into memory array \(N\).

\(N[J] \leq M[I]\) causes copying of one element.

**3.1 Implementation #1: States:** (see the state diagram on the next page)

(Verilog files: `copy_array_to_array_impl1.v`, `copy_array_to_array_impl1_tb.v`)

**LS2C** Locate Start of the 2nd Chunk in array \(M\)
(Increment \(I\) until MSB of \(M[I]\) is a "1"; but what if there are no numbers starting with a "1"?)

**C221** Copy 2 to 1 (= Copy Chunk 2 of \(M\) to Chunk 1 of \(N\))

**C122** Copy 1 to 2 (= Copy Chunk 1 of \(M\) to Chunk 2 of \(N\))

After copying one chunk, "I" has to wrap around before you start copying the second chunk. But "J" goes on! So, instead of worrying about what value of "I" indicates the end of the second chunk copying, one can look at the terminal value of "J". In fact, if \(J\) is at its terminal value, can you go to DONE state, whether you are at C221 or C122? Yes / No.

Wrapping "I" around: You would use \(I = 9 / I = 10\) as wrap-around condition.

Is it possible for I" to become 9 in LS2C itself? Yes / No
In what kind of situation do you need to initiate wrapping of "I" around in **LS2C**?

In the example **M** array on the previous page, \( M[6] \) is the first element with its MSB \( M[I][3] = 1 \). You detect this in **LS2C**. Will you transfer this element of **M** to **N** in **LS2C** itself and *save* a clock, or will you *waste* a clock and transfer it in **C221** state? You will __________ (save / waste) a clock. Hardware designers do not waste clocks!

Which of the following 4 **M** arrays will result in taking the *least* number of clocks in all three states’ **LS2C, C221, C112**, put together? _______ (write the roman numeral(s) for the case(s)). How many clocks is that least number of clocks? _______

(i) all elements of the array **M** start with a "0" in their MSB.
(ii) all elements of the array **M** start with a "1" in their MSB.
(iii) the first 9 elements of the array **M** start with a "0" and the last starts with a "1".
(iv) the first element of the array **M** starts with a "0" and the rest of 9 start with a "1".

Which of the above 4 **M** arrays will result in taking the *maximum* number of clocks in the three states, **LS2C, C221, C112**, put together? _______ (write the roman numeral(s) for the case(s)). How many clocks is that maximum number of clocks? _______

\[
\text{Reset, Start} \quad \text{INI} \\
\quad I <= 0; \\
\quad J <= 0; \\
\text{Start} \quad \text{ACK} \\
\text{DONE} \\
\quad \text{ACK} \\
\text{Start} \\
\text{LS2C} \quad \text{C221} \quad \text{C122} \\
\text{Reset, Start} \quad \text{INI} \\
\quad I <= 0; \\
\quad J <= 0; \\
\text{Start} \quad \text{ACK} \\
\text{DONE} \\
\quad \text{ACK} \\
\text{Start} \\
\text{LS2C} \quad \text{C221} \quad \text{C122} \\
\text{Reset, Start} \quad \text{INI} \\
\quad I <= 0; \\
\quad J <= 0; \\
\text{Start} \quad \text{ACK} \\
\text{DONE} \\
\quad \text{ACK} \\
\text{Start} \\
\text{LS2C} \quad \text{C221} \quad \text{C122} \\
\]
3.2 Implementation #2:
(Verilog files: copy_array_to_array_imp2.v, copy_array_to_array_imp2_tb.v)

States:
- **LS2C**: Same as before
- **CBC**: Copy Both Chunks (Of course, copy Chunk 2 of \( M \) to Chunk 1 of \( N \) first, followed by Chunk 1 of \( M \) to Chunk 2 of \( N \) in one single state.)

Your TA, Mr. Trojan, says that you can combine the two states, **C221** and **C122**, of the previous implementation into a single state **CBC** as defined above, as basically, in both these states, you copy an element of \( M \) into \( N \) and advance both "\( I \)" and "\( J \)". Complete the following state diagram.

3.3 Based on the number of clocks taken in different cases, the implementation #2 is _____________ (superior to / inferior to / sometimes superior sometimes inferior / always at the same level as) the implementation #1.

3.4 For a good hardware design, how many clocks do you spend in the CBC state?

- **Student #1**: Well there are 10 elements in the array. So 10 clocks.
- **Student #2**: No, not 10, it is 9 clocks. You copy one element in LS2C.
- **Student #3**: But, you may not have a chance to copy an element in LS2C always. What if there is no element with MSB = 1?
- **Student #4**: So, are you saying it is data-dependent? Sometimes, 9 clocks and sometimes 10 clocks?
4 (20 points) 20 min.

Matrix Multiplication

(Verilog files: matrix_multiplication.v, matrix_multiplication_tb.v)

\[ C \left[ m \times p \right] = A \left[ m \times n \right] \times B \left[ n \times p \right] \]

Matrix multiplication

Extract from http://en.wikipedia.org/wiki/Matrix_Multiplication

Matrix multiplication is defined between two matrices only if the number of columns of the first matrix is the same as the number of rows of the second matrix.

Formally, for

\[ A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p} \]

then

\[ (AB) \in \mathbb{R}^{m \times p} \]

where the elements of \(AB\) are given by

\[ (AB)_{i,j} = \sum_{r=1}^{n} A_{i,r}B_{r,j} \]

for each pair \(i\) and \(j\) with \(1 \leq i \leq m\) and \(1 \leq j \leq p\).

Instead of 1 to \(m\), it is better to go from 0 to \(m-1\) in hardware implementation with counters.

Extract from http://en.wikipedia.org/wiki/Matrix_Multiplication

\[
(AB)_{1,2} = \sum_{r=1}^{2} a_{1,r}b_{r,2} = a_{1,1}b_{1,2} + a_{1,2}b_{2,2}
\]

\[
(AB)_{3,3} = \sum_{r=1}^{3} a_{3,r}b_{r,3} = a_{3,1}b_{1,3} + a_{3,2}b_{2,3}
\]
Pseudo code using a nested for loop

Here, we assume that the result matrix, C[m x p] is cleared initially.

```
for (i = 0; i <= m-1; i = i + 1)
    for (j = 0; j <= p-1; j = j + 1)
        for (k = 0; k <= n-1; k = k + 1)
            C[i,j] = C[i,j] + A[i,k] * B[k,j];
```

Pseudo code using a nested if construct

Here again, we assume that i, j, and k are cleared initially.

```
if (i <= m-1)
    { if (j <= p-1)
        { if (k <= n-1)
            { C[i,j] = C[i,j] + A[i,k] * B[k,j];
              k = k + 1;
            }
        else
            { k = 0;
              j = j + 1;
            }
      }
    else
        { j = 0;
          i = i + 1;
        }
    }
```
Pseudo code adjusted for Hardware implementation

\[ C[i,j] = C[i,j] + A[i,k] \times B[k,j]; \]

outside the “if”

Only RTL inside the state circle.

if \((i \leq m-1)\)

\{ if \((j < p-1)\)

\{ if \((k < n-1)\)

\quad k \leq k + 1;

\quad if \((k = n-1)\)

\quad \{ k \leq 0;

\quad \quad j \leq j + 1;

\quad \}\}

\}\}

if \((j = p-1)\)

\{ if \((k < n-1)\)

\quad k \leq k + 1;

\quad if \((k = n-1)\)

\quad \{ k \leq 0;

\quad \quad j \leq 0;

\quad \quad i \leq i + 1;

\quad \}\}

\}\}

Note: We are using non-blocking assignment operator (\(<=\)) throughout this code.

Note: \(i\) is preparing to increment while (not after) \(k\) and \(j\) are preparing to go to zero simultaneously.

Hardware Pseudo code rewritten

Terminal value of \(i\)

(\textit{along with appropriate terminal values of} \(j\) \textit{and} \(k\))

goes into the state transition conditions.

\[ C[i,j] = C[i,j] + A[i,k] \times B[k,j]; \]

if \((j=p-1) && (k=n-1)\)

\quad i \leq i + 1; j \leq 0; k \leq 0;

else if \((k=n-1)\)

\quad j \leq j + 1; k \leq 0;

else

\quad k \leq k + 1;
The last iteration should prepare you to exit.

Serial Matrix Multiplication $C(m \times p) = A(m \times n) \times B(n \times p)$

C(i, j) <= C(i, j) + A(i, k) * B(k, j)
if $i++, j <= 0; k <= 0$
else if $j++, k <= 0$
else $k++$

INIT
$m_1 <= m_1_value$
$n_1 <= n_1_value$
$p_1 <= p_1_value$
C(m \times p) <= 0$
// in testbench

DONE

EXIT =