This homework consists of questions taken from the notes and open-ended questions. You must do the homework by yourself. **No collaborations are allowed.** Late homework will receive a 5% penalty per day. There are total 100 points. This homework is worth 10% of your overall grades.

1. (50 points) **DCT implementation**

Consider DCT for JPEG/MPEG image compression applications. Two dimensional, separable DCTs are applied to each 8 by 8 block of image \( f(m,n) \), \( 0 \leq m, n \leq 7 \) where \(-128 \leq f(m,n) \leq 127\) is the value (8-bit 2’s complement format) of the \((m,n)\) pixel. The 2D DCT coefficients are denoted by \( F(u,v) \), \( 0 \leq u, v \leq 7 \). The values of each cosine function \( \cos((2n+1)u\pi/16) \) will be represented with an 8-bit fixed point fractional number.

In pages 24-26 of the DSP note (posted in the course web page), a fast 1D 8-point DCT algorithm by Arai, Agui, and Nakajimi is given. You may download the MATLAB m-file from course web page to experiment it yourself. For convenience, the algorithm is listed below:

```matlab
Input: x(m), m = 0 to 7.
% step 1
% x(m,1) = x(m) + x(7-m), m = 0, 1, 2, 3
% x(m,1) = x(7-m) - x(m), m = 4, 5, 6, 7

% Step 2.
% x(m,2) = x(m,1) + x(3-m,1), m = 0, 1
% x(m,2) = x(3-m,1) - x(m,1), m = 2, 3
% x(4,2) = -x(4,1) - x(5,1)
% x(m,2) = x(m,1) + x(m+1,1), m = 5, 6
% x(7,2) = x(7,1)

% Step 3.
% x(0,3) = x(0,2) + x(1,2)
% x(1,3) = x(0,2) - x(1,2)
% x(2,3) = x(2,2) + x(3,2)
% x(4,3) = x(4,2) + x(6,2)
% x(m,3) = x(m,2), m = 3, 5, 6, 7

% Step 4.
% x(m,4) = x(m,3), m = 0, 1, 3, 7
% x(2,4) = x(2,3)*a1
% tmp = x(4,3) * a5
% x(4,4) = x(4,3)*a2 + tmp
% x(5,4) = x(5,3)*a3
% x(6,4) = x(6,3)*a4 + tmp

% Step 5.
% x(m,5) = x(m,4), m = 0, 1, 4, 6
```
We will consider a dedicated hardware implementation of this algorithm. We will use four types of components: (i) hardware array multiplier, M; (ii) hardware adder, A; (iii) dedicated buses, B, and (iv) registers R. We assume the eight n-bit inputs can be made available simultaneously if needed from input ports. The outputs will be stored in eight output registers. The output will not be made available to outside this hardware DCT module until all eight outputs are ready.

(a) (10 points) Derive the dependence graph of this given algorithm. Label input, output variables, as well as intermediate variables \{x(m,i); 0 \leq m \leq 7, 1 \leq i \leq 6\} provided space permits.

(b) (5 points, dynamic range) If we want to avoid any truncation error due to finite register length, how many bits, as a function of \(n\), will it require to store each intermediate or final result of the computation?

(c) (10 points, rounding) In practice, we want to use a fixed register length \(\ell\) (>\(n\)) for all registers. These \(\ell\) bits will contain both integer and fractional parts of the number. This implies that we need to round the intermediate results into \(\ell\)-bit fixed-point numbers if \(\ell\) is chosen to be smaller than the answer you obtained in part (b) of this problem. If we want the maximum rounding error at each output to be no more than \(2^{-1}\), find the minimum value of \(\ell\). Here, by using only \(\ell\)-bit registers, we assume that the output of each multiply and each addition will be rounded to no more than \(\ell\) bits before it is stored into a register.

(d) (15 points, Assignment and scheduling) Assume that there are one hardware multipliers (M) and two adders (A1, A2) are available, together with several registers as well as dedicated buses. For simplicity, assume a multiply or an addition will each takes 1 clock cycle to perform. Derive a schedule to implement this algorithm using (i) smallest number of clock cycles, and (ii) smallest number of registers and buses. A sample schedule has the following format:

<table>
<thead>
<tr>
<th>clock</th>
<th>M</th>
<th>A1</th>
<th>A2</th>
<th>Registers</th>
<th>#Buses</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>(x(m); m=0:7)</td>
<td>-</td>
<td>Input in registers</td>
</tr>
<tr>
<td>1</td>
<td>x(0,1)</td>
<td>x(7,1)</td>
<td>x(m); m=1:6, x(0,1), x(7,1)</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

At clock#1, \(x(0,1)\) is computed in adder A1, and \(x(7,1)\) is computed in adder A2. These two intermediate results need to be stored in registers. However, since \(x(0)\) and \(x(7)\) are no longer needed, these intermediate results can be stored back to the corresponding location. This is an example of in-place computation to save the number of registers.
needed. At this step, we still use 8 registers. To facilitate these two additions, four \( \ell \)-bit buses are needed, for \( x(0), x(7) \) (sources), and \( x(0,1), x(7,1) \) (results).

(c) (10 points, pipelining) One way to increase the speed of computation is pipelining. Our goal is to compute one 8-point DCT per clock cycle using pipelining. What is the minimum number of multipliers, adders, and registers needed to accomplish this goal? Note that all eight outputs must be available at the same clock cycle.

2. (40 points, Huffman Coding)

(a) (20 points, encoding) Consider a Huffman code encoder that has an \( n \)-bit input port and an 8-bit output port. Its output port is connected to a dedicated memory. Whenever the encoded output bits accumulated a byte, it will be exported to the memory. The remaining bits will be held within the module unless a specific symbol indicating end of input sequence. At that time, the remaining bits will be exported. Suppose that the input symbols are the usual 7-bit ASCII code words. Also, assume that the ASCII code 1111111 is used to indicate end of symbol sequence, and it does not have a corresponding Huffman code. The corresponding Huffman code for each of the remaining 127 symbols varies from 3 bits to 19 bits. Design a hardware, dedicated Huffman encoding module by giving a block diagram, and specify the design of individual blocks.

(b) (20 points, decoding) Given a binary sequence corresponding to a Huffman encoded symbol stream. Our goal is to decode this sequence such that we may achieve a decoding rate of **one symbol per clock cycle**, rather than one input bit per clock cycle. For simplicity, let us consider the following Huffman table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
</tr>
<tr>
<td>F</td>
<td>1111</td>
</tr>
</tbody>
</table>

Refer to the following papers (available online in the references section) for possible answers. (i) Specify how to decode it to achieve a rate of one symbol per clock cycle. (ii) Give a block diagram of your design and explain the function of each individual block.


3. (10 points, IIR filter) Consider the following IIR filter

\[
 y(n) = a_1 y(n-1) + a_2 y(n-2) + b x(n)
\]

Use the transformation stated in page 16 of the DSP note, re-write above equation into a state space formulation:
\[
\begin{bmatrix}
  v_1(n+1) \\
  v_2(n+1)
\end{bmatrix} =
\begin{bmatrix}
  p_{11} & p_{12} \\
  p_{21} & p_{22}
\end{bmatrix}
\begin{bmatrix}
  v_1(n) \\
  v_2(n)
\end{bmatrix} +
\begin{bmatrix}
  q_1 \\
  q_2
\end{bmatrix} x(n) = P \mathbf{v}(n) + \mathbf{q} x(n)
\]

\[
y(n) = \begin{bmatrix}
  r_1 \\
  r_2
\end{bmatrix}
\begin{bmatrix}
  v_1(n) \\
  v_2(n)
\end{bmatrix} + sx(n) = r^T \mathbf{v}(n) + sx(n)
\]

(a) (5 points) Find the parameters \(p_{11}, p_{12}, p_{21}, p_{22}, q_1, q_2, r_1, r_2,\) and \(s.\)

(b) (5 points) Observing the following recursion:

\[
v(n+1) = P \mathbf{v}(n) + \mathbf{q} x(n) = P \left[ P \mathbf{v}(n-1) + \mathbf{q} x(n-1) \right] + \mathbf{q} x(n) \\
= P^2 \mathbf{v}(n-1) + P \mathbf{q} x(n-1) + \mathbf{q} x(n) = \ldots
\]

Explain that a state space formulation allows easy un-winding of the 2\textsuperscript{nd} order IIR filter (just like the first order case) to speed up computation. Note that as long as the two roots of the characteristic polynomial

\[\lambda^2 - a_1 \lambda - a_2 = 0\]

have negative real parts, \(P^m \rightarrow 0\) as \(m \rightarrow \infty.\)