Q1. The figure below shows a Prefix Sum function along with its data flow graph.

```c
int tmp = 0;
for ( int i = 0; i < 3*N; i++ ) {
    tmp += in[i];
    out[i] = tmp;
}
```

(a) If we unroll the loop by a factor of 3, could you draw the dependence graph of the unrolled loop? Clearly identify each inter-iteration dependence with a distance label. Please ignore the code outside the loop and the part for updating the loop counter.
Could you calculate the MII if we were to pipeline the unrolled loop? Please provide both RecMII and ResMII. We assume that each operation takes a full cycle. No operation chaining is allowed. We also assume that the in array is mapped to a dual-ported memory (with two read-write ports). Likewise, out array is mapped to a separate dual-ported memory.
Q2. Mean filter is a useful technique in image processing to reduce noise in an image. One of the simplest forms of the mean filter is a 1-D filter that takes the average of the current pixel and its neighbors on the same horizontal (or vertical) line. The following C code snippet shows a mean filter with a 1x5 horizontal kernel. In this scheme, each pixel forms a kernel with four neighboring pixels on the same row and the mean value of these five pixels determines the value of the respective pixel in the output (filtered) image.

```c
/* scan the image row by row */
/* N = row#, M = col# */
for (i = 0; i < N; ++i)
    /* scan each row pixel by pixel from left to right */
    for (j = 0; j < M; ++j) {
        #pragma pipeline II=2
        /* get values of the pixels in the kernel */
        p1 = (j > 1) ? img[i][j-2] : 0;
        p2 = (j > 0) ? img[i][j-1] : 0;
        p3 = img[i][j];
        p4 = (j < M-1) ? img[i][j+1] : 0;
        p5 = (j < M-2) ? img[i][j+2] : 0;
        /* compute the mean */
        out[i][j] = (p1 + p2 + p3 + p4 + p5) / 5;
    }
```

Figure 1 further illustrates the memory access pattern of the given filter. For each center pixel, we extend to two pixels to the left and two pixels to the right to form the kernel for the mean value calculation.

![1-D mean filter with a 1x5 horizontal kernel.](image)

**Figure 1:** 1-D mean filter with a 1x5 horizontal kernel.

**Assumptions:** (i) The input image (i.e., the `img` array) and the output image (i.e., the `out` array) are stored in two separate RAM blocks with the right word widths; (ii) Both RAMs have two concurrent read ports and one write port; (iii) These RAMs cannot be further partitioned or banked.
(a) If we pipeline the inner loop (as indicated by the pragma in the code), what would be the best possible II without any additional code changes or architectural optimizations? Please identify the limiting factor(s) of the pipeline throughput (recurrence, resource, or both?).

(b) Can you exploit the memory access pattern of the 1-D mean filter and modify the C code in a way that a better II can be achieved? Note that your hand optimized code does not need to be syntactically sound. It is more important to explain the ideas behind the code changes.

Hint: You are allowed to introduce additional scalar/array variables and extend the iterations of the inner loop when necessary (unrolling will not help).
Q3. To form an integer linear programming (ILP) model for the constrained scheduling problem, we have learned to use a binary decision variable $X(i, k)$ to indicate if operation $i$ starts at cycle $k$, where $1 \leq k \leq L$ with $L$ being the maximum schedule latency.

In this problem, we will make use of these schedule variables to test if the output value of a given operation $v$ is live at a specific cycle $s$ ($1 \leq s \leq L$). Here we make the following assumptions:

- The output value of any operation $v$ (other than the primary outputs that do not have successors) is used by exactly ONE other operation, denoted as $USE[v]$.
- We also assume that each operation takes exactly one single full cycle and no chaining is allowed.

The definition of the liveness of a value is illustrated in the Figure 2.

![Figure 2: A scheduled graph with three operations.](image)

(a) Can you introduce a derived binary variable $Y(v, s)$ to indicate if an operation $v$ is scheduled before cycle $s$ (excluding $s$)? Hint: Make use of the relevant $X$’s.

(b) Can you introduce another derived binary variable $Z(v, s)$ to indicate if an operation $v$ is scheduled at or after cycle $s$? Hint: Make use of $Y$.

(c) Finally, can you make use of the variables introduced from (a) and (b) to derive binary variable $R(v, s)$, which indicates if the output value of $v$ is live at cycle $s$. Hint: You may need to introduce additional linear constraint(s) between $Y$, $Z$, and/or $R$. You may also make use of $USE[v]$ based on the assumptions stated.
Q4. Given a chain of \(N\) operations \(\{O_1, O_2, \ldots, O_N\}\) \((N > 1)\) where each operation \(O_i\) is associated with a positive integral delay value of \(d_i\) \((d_i \leq 10\text{ns})\), our goal is to automatically place (or insert) one or multiple registers into this chain to minimize the clock period of the circuit (i.e., maximize the operating frequency).

\[d_1 \rightarrow d_2 \rightarrow d_3 \rightarrow \ldots \rightarrow d_n\]

(a)

\[5 \rightarrow 4 \rightarrow 10 \rightarrow 1 \rightarrow 1\]

(b)

\[5 \rightarrow 4 \rightarrow 10 \rightarrow 1 \rightarrow 1\]

(c)

Figure 3: Register placement in an operation chain: (a) an abstract model without registers; (b) an optimal placement of one register; (c) an optimal placement of two registers.

Figure 3 (a) sketches the diagram of an \(N\)-operation chain. Without placing any registers between operations, the clock period would be the total combinational delay, which is \(D = \sum d_i\). A concrete example of a 5-operation chain is shown in Figure 3 (b), where an optimal one-register placement is also shown. In this case, the final clock period is \(\text{max}(5 + 4, 10 + 1 + 1) = 12\text{ns}\). Figure 3 (c) shows that we can further reduce the clock period to 10\text{ns} if we are allowed to insert two registers into the chain.

Assumptions: (i) All operations are combinational and they cannot be pipelined to form multi-cycle operations; (ii) We assume no resource sharing in this question.

(a) For an \(N\)-operation chain, can you devise an algorithm to insert ONE (and only one) register into the chain to achieve the minimum clock period? Please explain the key idea and justify the time complexity of your algorithm.

(b) For an \(N\)-operation chain, can you devise an efficient algorithm to place THREE registers on the chain to achieve the minimum clock period? Please explain the main idea and provide pseudocode or flow diagram to sketch out your algorithm. Please also analyze and justify the time complexity of your algorithm.