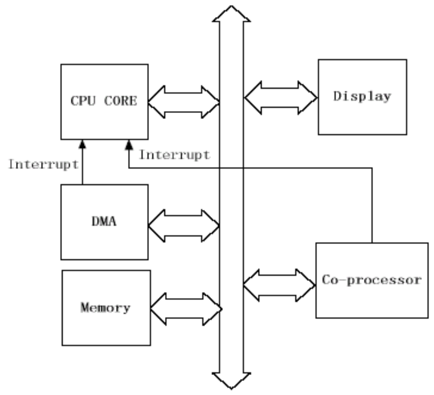
Problem: System Analysis **(20 Points)**



This is a simple single microprocessor core platform with a video coprocessor, which is configured to process 32 bytes of video data and produce a 1 byte result. Each display frame contains 30 X 10 words of data. The CPU or the DMA is used to transfer data between the memory, display or coprocessor. The bus cycle count for each transaction is:

- CPU CORE can access data from any module.
  - o Requires 3 bus cycles of overhead for each bus cycle. Remember it requires a load and store to move any word.
- DMA is Direct Memory Access module which can be used to transfer data between the memory block, the display module or co-processor module.
  - o 15 bus cycles for CPU to configure the DMA controller.
  - o 2 cycles per data word transfer (one read, one write).
- Access times (read and write):
  - o Memory: 1 bus cycle in addition to any overhead.
  - o Co-processor: 4 bus cycles in addition to any overhead
  - o Display: 165 bus cycles for each word in addition to any overhead cycles
- Interrupt
  - o The coprocessor and display module will generate interrupt after completing their tasks
  - o Interrupt subroutine takes 100 bus cycles.
- Co-processor cycle times:
  - o CPU CORE or DMA will write 32 data words to the coprocessor. The coprocessor will start processing the data **after** the CPU CORE or DMA writes a one word **command**. It will take 170 cycles for the co-processor to process the data and the result is 1 word, i.e., the coprocessor compresses 32 words down to one word. The CPU CORE or DMA will read the word in the coprocessor and transfer it.
- All ports and buses width are 1 word wide (32 bits)
- You cannot do more than one transfer on the bus at one time.

Question: What is the minimum number of bus cycles that it will take to process one frame of data (each frame is 30 X 10 words)? Use back of this page to show your work and any assumptions that you make.

Problem: System Level Languages **<u>(25 Points)</u>**

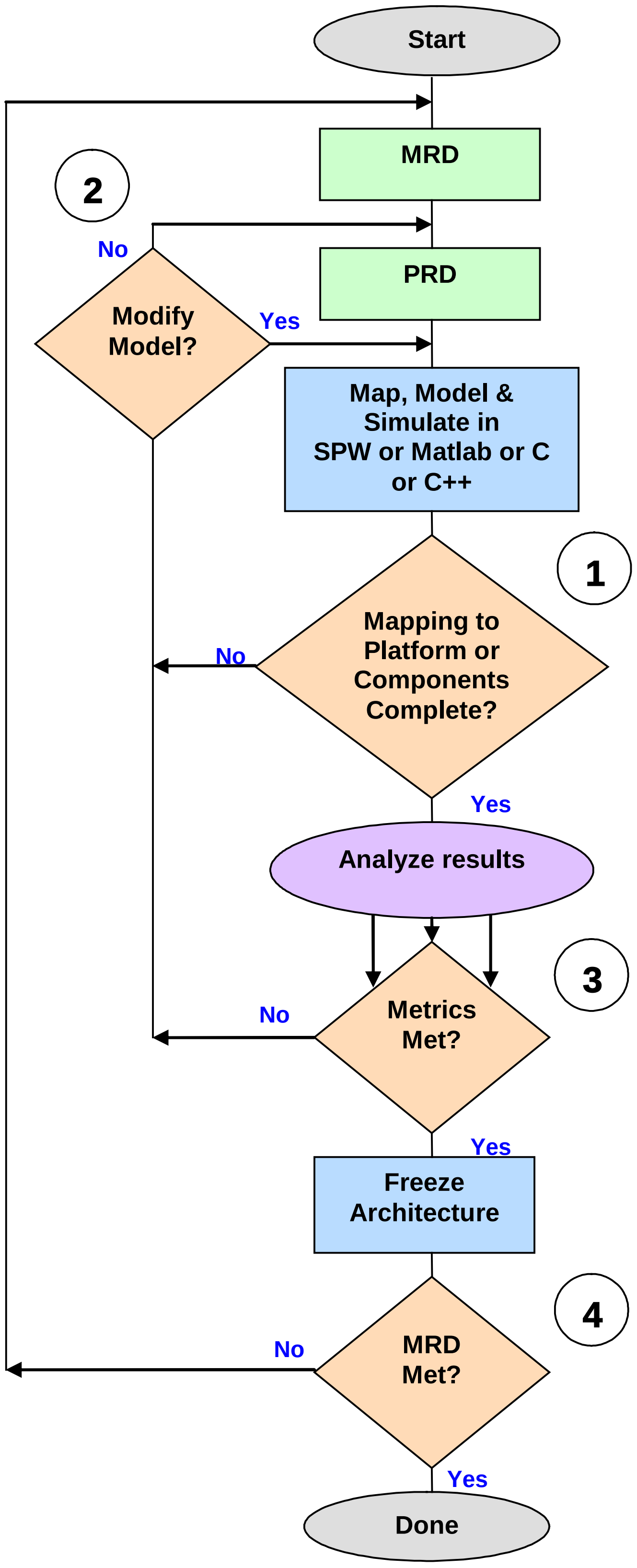
In recent years, there has been an increasing focus on electronic system-level (ESL) modeling languages, such as SystemC. Discuss the factors driving the high interest in ESL languages, and how such languages hope to address those issues.

Answer:

Problem: Design Methodology (<u>25 Points</u>)

Explain in detail what decisions are being made in this flow chart? Hint: there are 4 decision points.
Use the back of the sheet if you need more room to write.

Answer:



Problem: Software Analysis and Profiling **(20 Points)**

Given the inner loop of the BCH encoding algorithm in C below, identify the number of XOR and AND operations performed in the loop as a function of *k*. Assume that *length* = 1024, and that in any bit position, a 0 and a 1 are equally likely.

```
encode_bch()
/*
 * Compute redundancy bb[], the coefficients of b(x). The redundancy
 * polynomial b(x) is the remainder after dividing x^(length-k)*data(x)
 * by the generator polynomial g(x).
 * k = dimension (no. of information bits/codeword) of the code
 */
{
      register int    i, j;
      register int    feedback;

      for (i = 0; i < length - k; i++)
            bb[i] = 0;
      for (i = k - 1; i >= 0; i--) {
            feedback = data[i] ^ bb[length - k - 1];
            if (feedback != 0) {
                  for (j = length - k - 1; j > 0; j--)
                        if (g[j] != 0)
                              bb[j] = bb[j - 1] ^ feedback;
                        else
                              bb[j] = bb[j - 1];
                  bb[0] = g[0] && feedback;
            } else {
                  for (j = length - k - 1; j > 0; j--)
                        bb[j] = bb[j - 1];
                  bb[0] = 0;
            }
      }
}
```

Answer:

Problem: Software Analysis and Profiling <u>(20 Points)</u>

Given the following simplified code of the Viterbi Decode method:

```
CViterbiDecider:Decode(CVector<CFDistance>& vecNewDistance,
                       CVector<_DCSN>& vecOutBits)
{
   for (i = 0; i <  iNumBitsWithMemory; i++)           /* Main loop over all bits */
   {
        const int iPos0 = iDistCnt;

        if (veciTablePuncPat[i] == PP_TYPE_0001)
        {
                /* Pattern 0001 */
                METRICSET(i)[ 0] = vecNewDistance[iPos0].rTow0;
                METRICSET(i)[ 2] = vecNewDistance[iPos0].rTow0;
                METRICSET(i)[ 4] = vecNewDistance[iPos0].rTow0;
                METRICSET(i)[ 6] = vecNewDistance[iPos0].rTow0;
                METRICSET(i)[ 9] = vecNewDistance[iPos0].rTow1;
                METRICSET(i)[11] = vecNewDistance[iPos0].rTow1;
                METRICSET(i)[13] = vecNewDistance[iPos0].rTow1;
                METRICSET(i)[15] = vecNewDistance[iPos0].rTow1;
        }
        else {
                const int iPos1 = iDistCnt1;
                 /* Calculate "subsets" of bit-combinations. "rIRxx00" means that
                   the fist two bits are used, others are x-ed in an intermediate result (IR) */
                const _REAL rIRxx00 =
                        vecNewDistance[iPos1].rTow0 + vecNewDistance[iPos0].rTow0;
                const _REAL rIRxx10 =
                        vecNewDistance[iPos1].rTow1 + vecNewDistance[iPos0].rTow0;
                const _REAL rIRxx01 =
                        vecNewDistance[iPos1].rTow0 + vecNewDistance[iPos0].rTow1;
                const _REAL rIRxx11 =
                        vecNewDistance[iPos1].rTow1 + vecNewDistance[iPos0].rTow1;

                if (veciTablePuncPat[i] == PP_TYPE_0101)
                {       /* Pattern 0101 */
                        METRICSET(i)[ 0] = rIRxx00;
                        METRICSET(i)[ 2] = rIRxx00;
                        METRICSET(i)[ 4] = rIRxx10;
                        METRICSET(i)[ 6] = rIRxx10;
                        METRICSET(i)[ 9] = rIRxx01;
                        METRICSET(i)[11] = rIRxx01;
                        METRICSET(i)[13] = rIRxx11;
                        METRICSET(i)[15] = rIRxx11;
                }
                else if (veciTablePuncPat[i] == PP_TYPE_0011)
                {        /* Pattern 0011 */
                        METRICSET(i)[ 0] = rIRxx00;
                        METRICSET(i)[ 2] = rIRxx10;
                        METRICSET(i)[ 4] = rIRxx00;
                        METRICSET(i)[ 6] = rIRxx10;
                        METRICSET(i)[ 9] = rIRxx01;
                        METRICSET(i)[11] = rIRxx11;
                        METRICSET(i)[13] = rIRxx01;
                        METRICSET(i)[15] = rIRxx11;
                }
```

```
            else {  /* The following patterns need one more bit */
                    const int iPos2 = iDistCnt2;

                    if (veciTablePuncPat[i] == PP_TYPE_0111) {
                            /* Pattern 0111 */
                            METRICSET(i)[ 0] = vecNewDistance[iPos2].rTow0 + rIRxx00;
                            METRICSET(i)[ 2] = vecNewDistance[iPos2].rTow0 + rIRxx10;
                            METRICSET(i)[ 4] = vecNewDistance[iPos2].rTow1 + rIRxx00;
                            METRICSET(i)[ 6] = vecNewDistance[iPos2].rTow1 + rIRxx10;
                            METRICSET(i)[ 9] = vecNewDistance[iPos2].rTow0 + rIRxx01;
                            METRICSET(i)[11] = vecNewDistance[iPos2].rTow0 + rIRxx11;
                            METRICSET(i)[13] = vecNewDistance[iPos2].rTow1 + rIRxx01;
                            METRICSET(i)[15] = vecNewDistance[iPos2].rTow1 + rIRxx11;
                    } else {/* Pattern 1111 */
                            /* This pattern needs all four bits */
                            const int iPos3 = iDistCnt3;

                            /* Calculate "subsets" of bit-combinations. "rIRxx00" means
                               that the last two bits are used, others are x-ed.
                               "IR" stands for "intermediate result" */
                            const _REAL rIR00xx = vecNewDistance[iPos3].rTow0 +
                                    vecNewDistance[iPos2].rTow0;
                            const _REAL rIR10xx = vecNewDistance[iPos3].rTow1 +
                                    vecNewDistance[iPos2].rTow0;
                            const _REAL rIR01xx = vecNewDistance[iPos3].rTow0 +
                                    vecNewDistance[iPos2].rTow1;
                            const _REAL rIR11xx = vecNewDistance[iPos3].rTow1 +
                                    vecNewDistance[iPos2].rTow1;

                            METRICSET(i)[ 0] = rIR00xx + rIRxx00; /* 0 */
                            METRICSET(i)[ 2] = rIR00xx + rIRxx10; /* 2 */
                            METRICSET(i)[ 4] = rIR01xx + rIRxx00; /* 4 */
                            METRICSET(i)[ 6] = rIR01xx + rIRxx10; /* 6 */
                            METRICSET(i)[ 9] = rIR10xx + rIRxx01; /* 9 */
                            METRICSET(i)[11] = rIR10xx + rIRxx11; /* 11 */
                            METRICSET(i)[13] = rIR11xx + rIRxx01; /* 13 */
                            METRICSET(i)[15] = rIR11xx + rIRxx11; /* 15 */
                    }
            }
    }

    BUTTERFLY( 0,  1,  0, 32,  0, 15);      BUTTERFLY( 2,  3,  1, 33,  6,  9);
    BUTTERFLY( 4,  5,  2, 34, 11,  4);      BUTTERFLY( 6,  7,  3, 35, 13,  2);
    BUTTERFLY( 8,  9,  4, 36, 11,  4);      BUTTERFLY(10, 11,  5, 37, 13,  2);
    BUTTERFLY(12, 13,  6, 38,  0, 15);      BUTTERFLY(14, 15,  7, 39,  6,  9);
    BUTTERFLY(16, 17,  8, 40,  4, 11);      BUTTERFLY(18, 19,  9, 41,  2, 13);
    BUTTERFLY(20, 21, 10, 42, 15,  0);      BUTTERFLY(22, 23, 11, 43,  9,  6);
    BUTTERFLY(24, 25, 12, 44, 15,  0);      BUTTERFLY(26, 27, 13, 45,  9,  6);
    BUTTERFLY(28, 29, 14, 46,  4, 11);      BUTTERFLY(30, 31, 15, 47,  2, 13);
    BUTTERFLY(32, 33, 16, 48,  9,  6);      BUTTERFLY(34, 35, 17, 49, 15,  0);
    BUTTERFLY(36, 37, 18, 50,  2, 13);      BUTTERFLY(38, 39, 19, 51,  4, 11);
    BUTTERFLY(40, 41, 20, 52,  2, 13);      BUTTERFLY(42, 43, 21, 53,  4, 11);
    BUTTERFLY(44, 45, 22, 54,  9,  6);      BUTTERFLY(46, 47, 23, 55, 15,  0);
    BUTTERFLY(48, 49, 24, 56, 13,  2);      BUTTERFLY(50, 51, 25, 57, 11,  4);
    BUTTERFLY(52, 53, 26, 58,  6,  9);      BUTTERFLY(54, 55, 27, 59,  0, 15);
    BUTTERFLY(56, 57, 28, 60,  6,  9);      BUTTERFLY(58, 59, 29, 61,  0, 15);
    BUTTERFLY(60, 61, 30, 62, 13,  2);      BUTTERFLY(62, 63, 31, 63, 11,  4);
  }
}
```

Assume that that the puncturing patterns are equally likely to occur, i.e. for every 5 iterations the puncturing patterns (0001,0101,0011,0111,1111) will on average occur one times each.

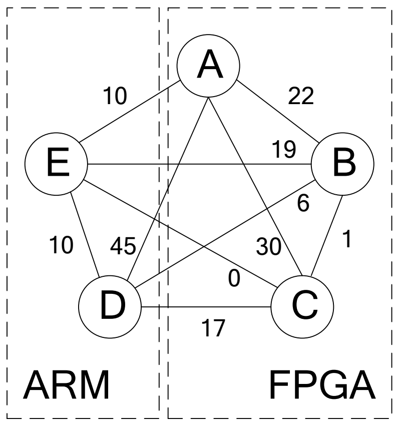
Furthermore, assume that *iNumBitsWithMemory* = 100 and that each BUTTERFLY() call contains four additions and two comparisons.

(a) For each ‘if’ statement, what is the probability of it being true when it is executed?

(b) Compute the average number of ALU (arithmetic and comparison) operations performed by a call to the Viterbi Decode() method.

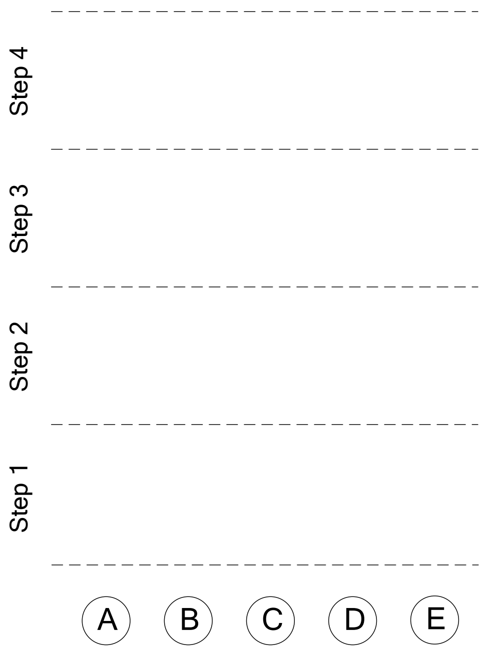
Problem: Partitioning (30 points)

Consider the following task graph and initial partitioning:

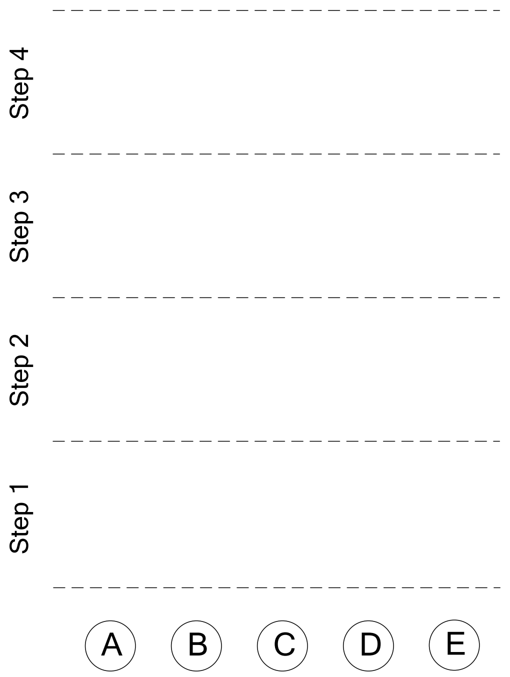


(a) Apply a modified Kernighan-Lin algorithm that iteratively moves a single node leading to the highest decrease in communication cost across the partition until no more gain can be achieved (or until only single node is left in a partition). Show the moves, partitions and communication costs in each step of the algorithm.

(b) Apply instead a hierarchical clustering solution to the task graph and show the final HW/SW partition and its communication cost. For computing the closeness values of the newly inserted edges, use the average value of the closeness values before clustering.

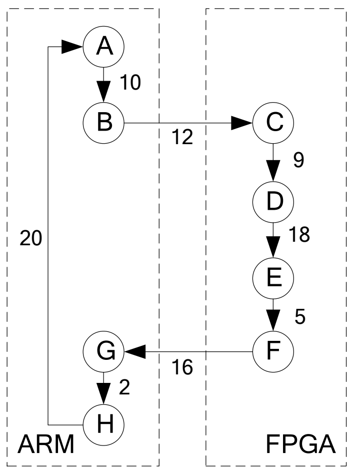


(c) Perform a hierarchical clustering, but use the *minimum* of the closeness values before clustering as the closeness of newly inserted edges.



Problem: Partitioning (30 points)

Consider the following task graph and initial partitioning:



(a) Apply a modified Kernighan-Lin algorithm that iteratively moves a single node leading to highest decrease in communication cost across the partition until no more gain can be achieved (or until only single node is left in a partition). Show the moves, partitions and communication costs in each step of the algorithm.

(b) Apply instead a hierarchical clustering solution to the task graph and show the final HW/SW partition and its communication cost.

(c) How would you extend the Kernighan-Lin algorithm in (a) to not only optimize for communication cost but also take processing times into account?

Problem: Task Scheduling (25 Points)

Consider a system comprised of three processes with the following execution times and periods:

Execution time	Period
$T_1 = 1$	$\tau_1 = 10$
$T_2 = 1$	$\tau_2 = 2$
$T_3 = 2$	$\tau_3 = 5$

Give a rate-monotonic-schedule (RMS) for the processes, and indicate the corresponding processor utilization. Could an early-deadline-first (EDF) scheduler also generate this schedule? Justify your answer (hint: give the EDF dynamic priorities at each relevant time step).

Answer:

Problem: Task Scheduling (25 Points)

Consider the problem of scheduling the following sets of tasks (assume that all tasks arrive at time 0).

Task	Period	Execution Time
A	20	5
B	60	10
C	40	10
D	30	5

(a) From an implementation perspective, what are the advantages/disadvantages of an RMS vs. an EDF scheduler?

(b) What is the utilization of a single processor running the tasks?

(c) Find an RMS schedule for the tasks.

(d) If the execution time of task C is increased to 15, find an RMS schedule for the new task set. What is the maximum execution time of C and corresponding processor utilization under which the task set is still schedulable?

(e) Perform EDF scheduling of the new task with an execution time for C of 15 time units (if there are two tasks with the same deadline break the tie in favor of the task with the shorter period). What is the maximum execution time of C and processor utilization under which the task set is still schedulable with an EDF strategy?

Problem: Task Scheduling (35 Points)

Consider the problem of scheduling the following sets of tasks.

Task	Period	Execution Time
A	80	20
B	120	30
C	40	10
D	60	10

(f) What is the utilization of a single processor running the tasks?

(g) Find and draw an RMS schedule for the tasks.

(h) Assume that Task B and C share a resource that is protected by a critical section/mutex, i.e. if the one of the tasks acquires the resource, the other task has to wait until the resource is relinquished. Assuming that both tasks hold the resource during all of their execution time in each period, find and draw an RMS schedule for the tasks.

(i) Briefly explain the concept of priority inversion and mark the priority inversion intervals on the schedule graph in (c).

(j) Briefly explain the priority ceiling protocol. Find and draw the RMS schedule with a priority ceiling implementation of the critical section.

(k) Briefly explain the priority inheritance protocol. Will the RMS schedule in (e) change for a priority inheritance instead of a priority ceiling implementation? If so, draw the modified schedule.

Problem: Tree Height Reduction and Operation Scheduling **(20 Points)**

A system requires the computation of the equation, x^3 + A.x^2 + B.x + C.

(a) If only two operations (multiplications or additions) can be done in one cycle, schedule the operations in order to complete the computation in the minimum number of cycles.

$\circledast$ $\oplus$ Use these symbols to represent the multiplication and addition.

(b) In order to reduce power, only one operation (multiplication or addition) can be done in one cycle. Find the schedule which obeys this constraint and takes the minimum number of cycles.

Problem: High-Level Synthesis (25 points)

Consider the following code fragment:

```
x = a + b + c;
x = x + c * d;
y = c * d * e;
z = x – y;
```

(a) Assuming one clock cycle per operation, derive minimum-latency ASAP and ALAP schedules for this code and determine the mobility for each operation.

(b) Assume a functional unit library that contains an ALU (adder/subtractor) with a delay of 25ns and a multiplier with a delay of 50ns. Furthermore, assume a resource constraint of allocating at most one multiplier. Schedule the code to minimize latency. Determine the final clock period.

(c) For your implementation in (b), determine the variable lifetimes and assign variables to a minimum number of registers. Assume that primary input variables are preloaded into their assigned registers before the beginning of the computation.

(d) Sketch a multiplexer-based realization of your final datapath.

(e) Show the state machine of the controller driving the datapath computation.

Problem: High-Level Synthesis (30 points)

A system requires the computation of the function: $y = a^2b + de^2 + cf + df + abc + def$

For the following questions, assume an adder has a delay of 1 cycle and the delay of a multiplier is 2 cycles.

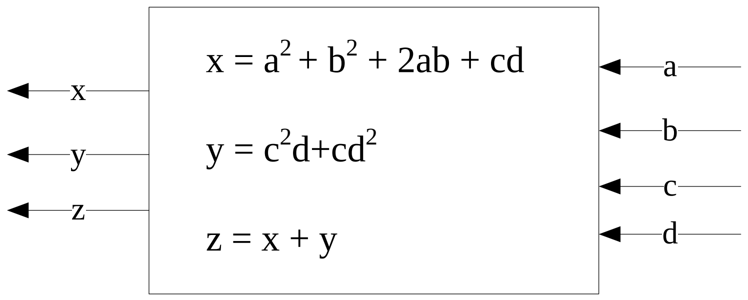
(a) Assuming unlimited resources, schedule the operations to compute the function in a minimum number of cycles.

(b) Assuming a resource constraint of a maximum of 2 multipliers and 1 adder. Schedule the operations into a minimum number of cycles using a list scheduling algorithm with the longest weighted path to the sink (i.e. the start time in an ALAP schedule) as priority.

(c) Is the schedule in (b) optimal or can you come up with a schedule with a shorter latency?

Problem: High-Level Synthesis (35 points)

Consider the following system:



Assuming the area cost of an adder is 1, the area cost of a multiplier is 4, and they both require one clock cycle per operation.

(a) Assuming one clock cycle per operation, derive minimum-latency ASAP and ALAP schedules for this system and determine the mobility for each operation.

(b) Assume the area cost of an adder is 1, the area cost of a multiplier is 4, and they both require one clock cycle per operation. Apply a force-directed scheduling (FDS) algorithm to determine a schedule that minimizes resource cost while not exceeding the minimum latency. Show the final area score. Is there a schedule that can achieve a lower cost?

(c) Assume an adder with a delay of 25ns and a multiplier with a delay of 50ns. Furthermore, assume a resource constraint of allocating at most one multiplier. Use a list scheduling algorithm to minimize latency. Show the final schedule and latency (in ns). Is there a schedule that can achieve a lower latency?

(d) For the FDS-generated implementation in (b), determine the variable lifetimes and assign variables to a minimum number of registers. Assume that primary input variables are preloaded into their assigned registers before the beginning of the computation.

(e) For your implementation in (d), draw a multiplexer-based realization of your final datapath.

(f) For your diagram in (e), show the state machine of the controller driving the datapath computation.