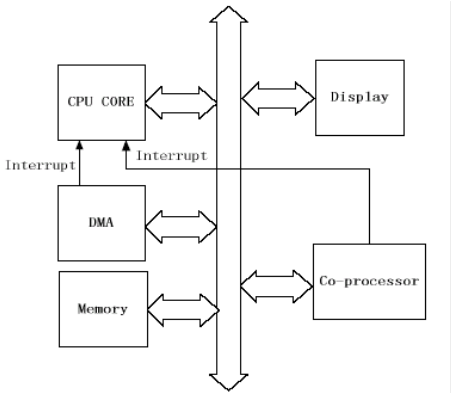
Problem: System Analysis (20 Points)



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:

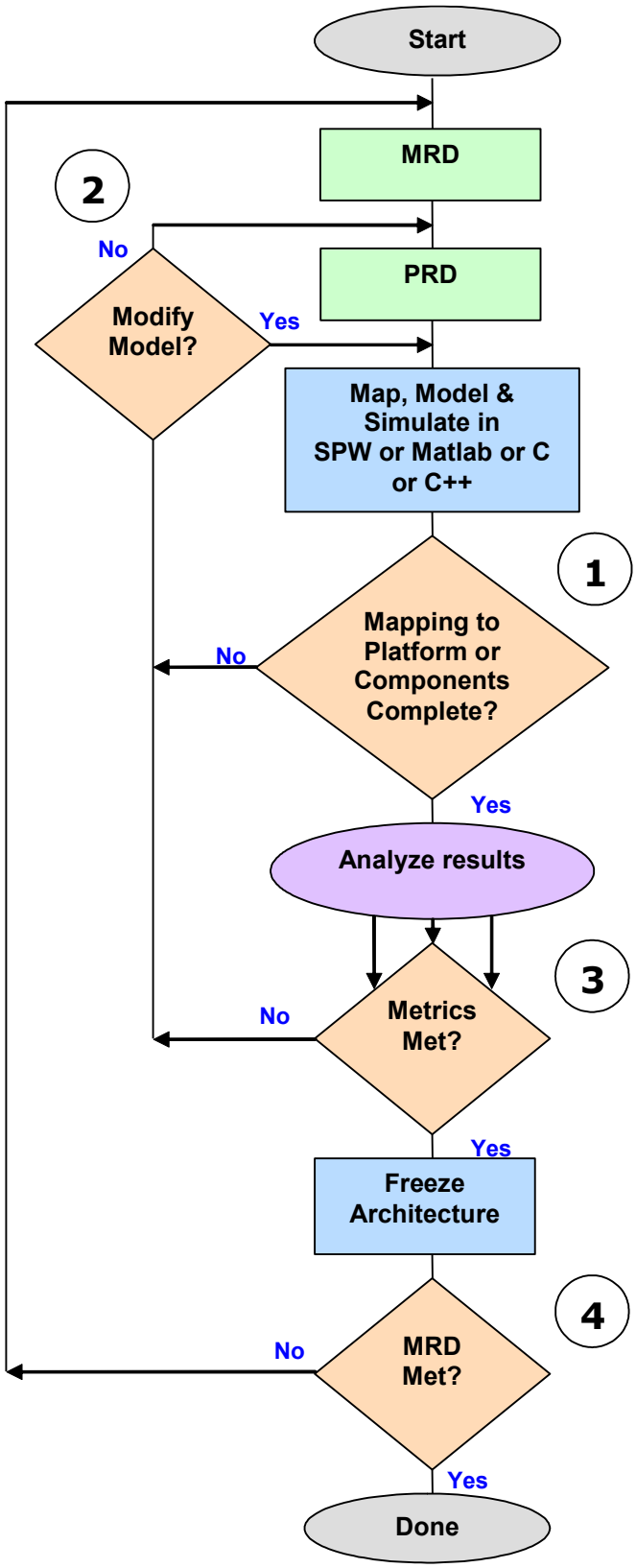
- CPU CORE can access data from any module.
  - Requires 3 bus cycles of overhead for each bus cycle. Remember it requires a load and store to move any word.
- DMA is Direct Memory Access module which can be used to transfer data between the memory block, the display module or co-processor module.
  - 15 bus cycles for CPU to configure the DMA controller.
  - 2 cycles per data word transfer (one read, one write).
- Access times (read and write):
  - Memory: 1 bus cycle in addition to any overhead.
  - Co-processor: 4 bus cycles in addition to any overhead
  - Display: 165 bus cycles for each word in addition to any overhead cycles
- Interrupt
  - The coprocessor and display module will generate interrupt after completing their tasks
  - Interrupt subroutine takes 100 bus cycles.
- Co-processor cycle times:
  - 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: Design Methodology (25 Points)

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;
            }
      }
}
```

Problem: Performance Analysis (25 points)

Consider the following compiler-generated intermediate representation code for the inner-most loop of a 3x3 GEMM:

```
oa = i*3;
ob = j;
oc = i*3 + j;
c = C[oc];
for (int k=0; k < 3; k++, oa += 1, ob += 3)
      c += A[oa] * B[ob];
C[oc] = c;
```

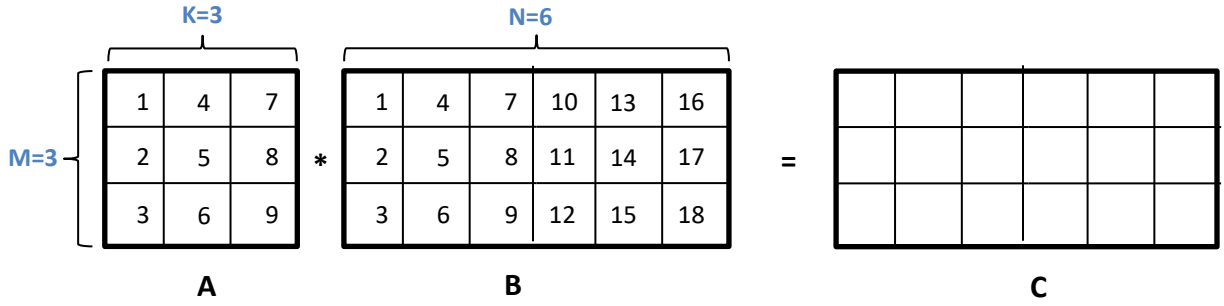
(a) Given an operator library that supports 2-input, 1-output operations LD (Indexed Load *a*[*b*] from memory given base address *a* and word offset *b*<<*2*), **+** (Addition, *a*+*b*), ***** (Multiplikation, *a***b*), **<** (Compare returning a true result if *a*<*b*, false otherwise) as well as a 1-input **←** (register initialization/assignment with constant) and a 3-input STR (Indexed Store to memory from base address *a* and offset *b*<<*2*), show the control-dataflow graph (CDFG) for the code above. Assume that A, B and C matrices are in memory with given base addresses, and variables i and j are already initialized in registers.

(b) Assuming a processor with a SIMD datapath (e.g. ARM NEON) that can perform (issue) up to four independent LD, ←, +, *, < or STR operations per cycle (LD/STR must have same base address), how many cycles are required to finish the inner-most GEMM loop? Assume assignments, loads and stores take 1 cycle (all data is in the cache), add, compare and multiply take 1, 1 and 3 cycles, respectively, and a 100% accurate branch predictor.

(c) Now apply loop unrolling optimizations to the inner-most GEMM code. Show the CDFG when fully unrolling the loop. How many cycles does it require to execute the unrolled code?

Problem: Execution Time Analysis (20 points)

Assume that we have a fully associative cache with 120 bytes capacity, where each cache line is 3 integers (i.e., 3*4 = 12 bytes) wide. The cache is initially empty and uses an LRU replacement policy with write-back. Given the following matrix multiplication algorithm and matrices with column-major layout, i.e. where elements are layed out in memory in the order as shown in the figure:



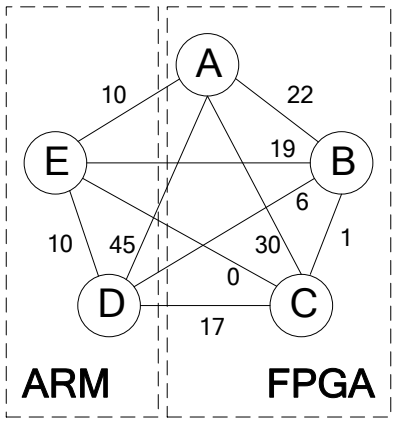

```
gemm: for (int m=0; m < 3; m++)
          for(int n=0; n < 6; n++)
                for(int k=0; k < 3; k++)
                      C[ m ][ n ] += A[ m ][ k ] * B[ k ][ n ]
```

(a) Assuming this code takes 564 cycles with perfect caching on a simple in-order core, what is the execution time if each cache miss takes 5 additional cycles?

(b) How can the code be improved to reduce the execution time?

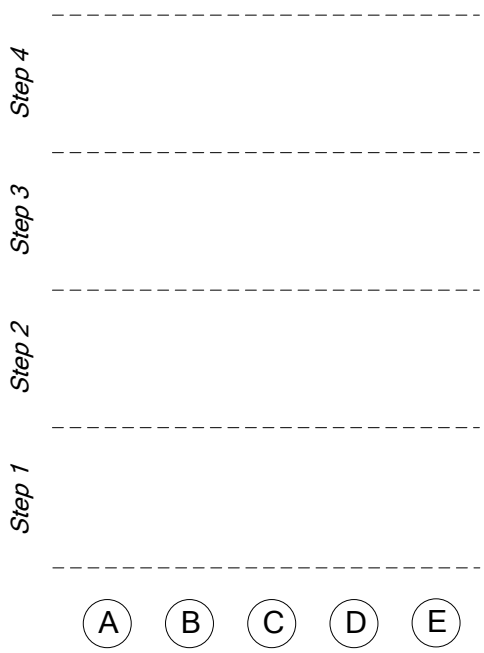
Problem: Partitioning (30 points)

Consider the following task graph and initial partitioning:

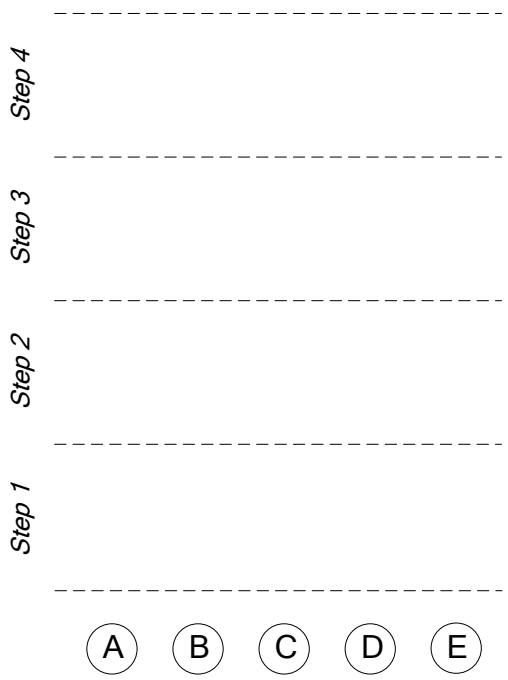


(a) Apply a modified Kernighan-Lin algorithm that iteratively moves a single node leading to the highest decrease in communication cost across the partition until no more gain can be achieved (or until only single node is left in a partition). Show the moves, partitions and communication costs in each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

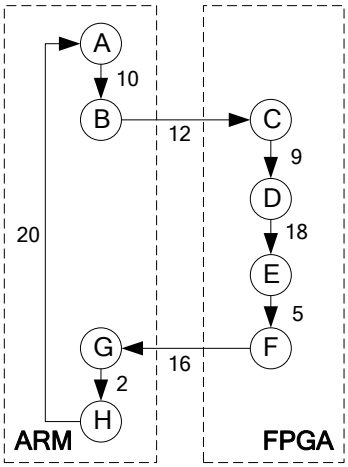


(c) Perform a hierarchical clustering, but use the *minimum* of the closeness values before clustering as the closeness of newly inserted edges.



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.

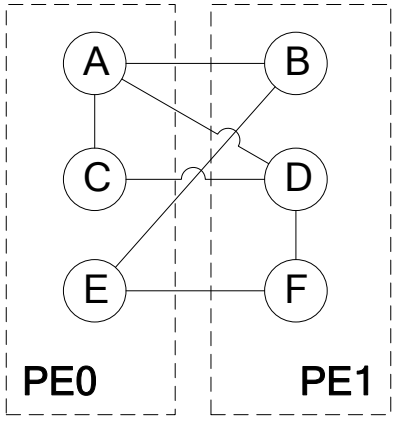
(b) Apply instead a hierarchical clustering solution to the task graph and show the final HW/SW partition and its communication cost.

(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: Partitioning (30 points)

In class and the homework we only looked at a simplified version of the Kernighan-Lin algorithm. The full Kernighan-Lin algorithm looks at complete sequences of node swaps. In each iteration of the algorithm, a set of possible partition candidates is constructed by consecutively swapping nodes that have not been swapped before and that result either in the largest gain or least loss in inter-partition communication cost per swap (i.e. considering intermediate swaps that may increase cost). The set of candidates is complete when all nodes have been considered for swapping, i.e. the graph is mirrored. Out of this set of candidate partitions, the algorithm selects the partition with the least cost, i.e. it actually only executes the partial subsequence of swaps that leads to the largest overall reduction in cost. This process (of constructing candidates and selecting the best) is repeated until no more gains can be achieved (there is no candidate that leads to any reduced cost).

Consider the following task graph with uniform communication costs per edge and initial partitioning:



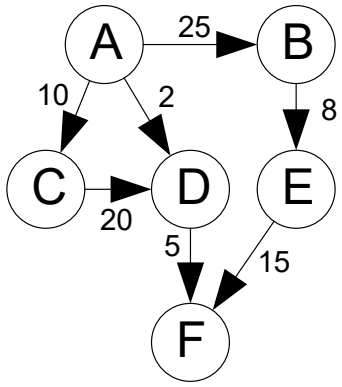
(a) Apply the full Kernighan-Lin algorithm to the task graph. Show the swap sequences, actually selected partitions and communication costs in each iteration of the algorithm.

(b) Can this algorithm still get stuck in a local minimum? Why or why not?

(c) How could this algorithm be extended to take computation costs and scheduling into account. Assume that tasks are periodic and that communication costs do not represent actual tasks dependencies (precedence constraints), but rather generally the fact whether two tasks ever exchange data or not (required connections).

Problem: Partitioning (25 points)

Consider the following task graph where communication costs indicate the number of kBytes transferred between tasks:



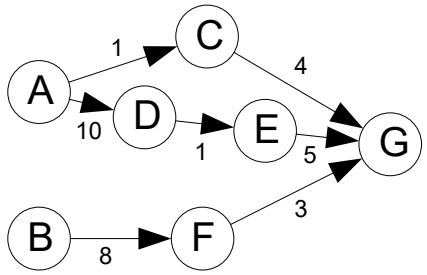
(a) Apply a hierarchical clustering algorithm where the communication cost of a clustered node is the sum of the bytes exchanged with other nodes. Show the graphs with communication costs after each clustering step. What is the final HW/SW partition on a system with one CPU and one hardware accelerator? What is the partition on a three-processor system (one CPU, one DSP and one accelerator)?

(b) Apply the (full) Kernighan-Lin algorithm to partition the graph into two groups with an equal number of nodes, starting from an initial A,B,C and D,E,F binning.

(c) Which algorithm gives the better solution for a 2-processor system? What are the tradeoffs between the two solutions? Is there a better 2-processor solution that minimizes communication costs?

Problem: Partitioning (25 points)

Consider the following task graph with communication costs that indicate the times required to transfer the data exchanged by tasks over the system bus connecting the processors:



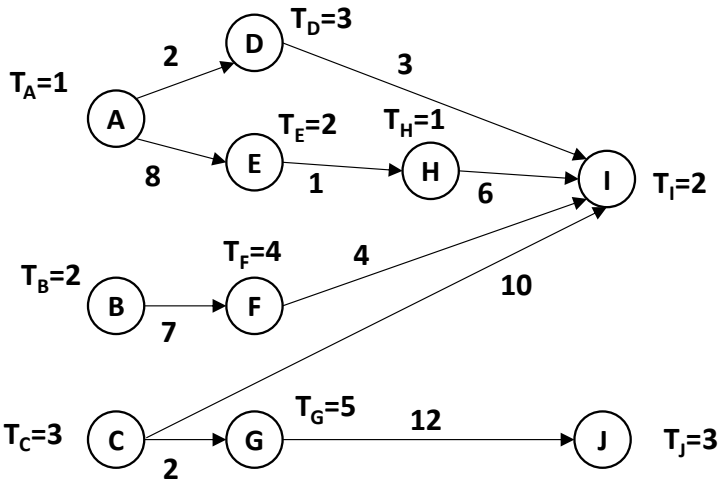
(a) Apply a hierarchical clustering algorithm to partition the graph onto two processors with a shared system bus where the cost of clustered nodes is the sum of communication delays with other nodes. Show the graphs and costs after each clustering step and the final partition.

A B C D E F G

(b) Apply a Kernighan-Lin algorithm, appropriately adapted to account for partitions of unequal size if necessary, on the two-processor solution from (a). Can Kernighan-Lin improve the communication cost further? Is there a different cut that achieves a lower communication cost?

Problem: Task Partitioning and Scheduling (40 points)

Consider the following task graph with given computation times and communication costs:



(a) Apply a hierarchical clustering algorithm to partition the graph onto three processors such that communication is minimized assuming a shared system bus where the cost of clustered nodes is the sum of communication delays with other nodes. Show the graphs and costs after each clustering step and the final partition.

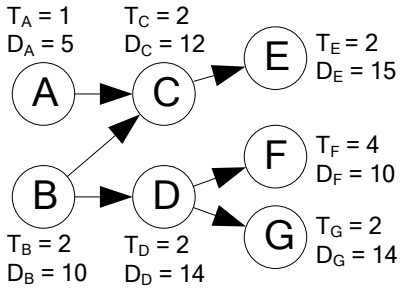
(b) How would you extend the Kernighan-Lin (KL) algorithm to support minimization of the sum of communication costs in a 3-processor system? Briefly describe your KL extensions, and assuming you start from the 3-processor solution in (a), show at least the first step(s) (one swap) of your extended algorithm (you do not need to show a complete iteration of the algorithm).

(c) Given the task graph with annotated execution times. assuming all communication costs are zero, describe an algorithm to schedule and map the graph onto three homogeneous processors such that the schedule makespan/length is minimized. Demonstrate your algorithm on the task graph. Is your algorithm optimal? If yes, why? If not generally, is your algorithm optimal for certain task graph instances, and if so which ones?

(d) Now assuming non-zero communication times, could you extend your algorithm to account for communication overhead and, if so, how?

Problem: Task Scheduling (35 points)

Consider a system that periodically executes the following graph of tasks with dependencies, (precedence constraints). Due to the dependencies, all tasks need to run at the same rate with a common period of 15 while all precedence relationships are maintained within each period. In addition, however, tasks may individually have stricter deadlines (relative to the start of the graph's period). Task execution times T_i and relative deadlines D_i are as indicated in the graph. Assume that tasks *A* and *B* are ready to execute at the beginning of each period.



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

(b) Apply an EDF algorithm and show the schedule for one period (relative to the start of the period), i.e. for one execution of the graph.

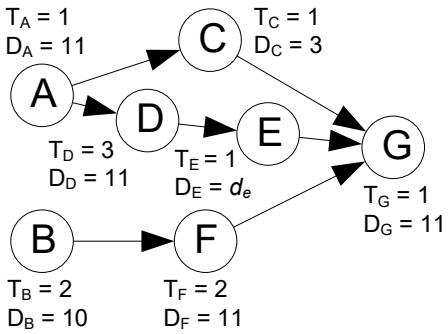
(c) In the presence of dependencies, EDF is no longer optimal in guaranteeing to find a schedule if it exists. However, a modified EDF* strategy becomes optimal by adjusting the deadlines of individual tasks to take their successors into account. This is done by starting with the sinks of the graph (nodes with no successors) and successively propagating deadlines that are adjusted for execution times upwards through the graph. Every time a deadline is propagated to a predecessor, the execution time of the current node is subtracted (such that it is guaranteed that the node will have enough time to execute once the predecessor has finished). At each node, a new deadline is then computed to be the smaller of its original deadline and of the minimum over adjusted deadlines propagated upwards from all its successors. Indicate the dependency-adjusted deadlines in the original task graph above and show the resulting EDF* schedule.

(d) Show the EDF* schedule for the task graph with adjusted deadlines executed on two processors. Assume that tasks can migrate between processors freely, i.e. strictly follow a strategy in which at any point in time the two tasks with the highest priority are running.

(e) Does any uni-processor, priority-based scheduling of tasks with dependencies ever require preemption? If so, under what conditions? If not, why not? How about in priority-based multi-processor scheduling?

Problem: Scheduling (25 points)

Consider a system that periodically executes the following graph of tasks with dependencies, (precedence constraints). Due to the dependencies, all tasks need to run at the same rate with a common period of 11 while all precedence relationships are maintained within each period. In addition, tasks individually have stricter deadlines (relative to the start of the graph's period). Task execution times T_i and relative deadlines D_i are as indicated in the graph. Assume that other than dependencies, all tasks are ready to execute at the beginning of each period, i.e. have arrival times of zero (relative to the start of the graph's period).



(a) What is the smallest deadline d_e of task E for which the graph is schedulable on a single processor using a modified EDF* algorithm that accounts for dependencies? Show the resulting schedule for one period, i.e. for one execution of the graph.

(b) What is the smallest deadline d_e of task E for which the graph is schedulable when executed on two processor strictly following a global EDF* algorithm in which tasks can freely migrate between processors and at any point the two ready tasks with the highest priority are running? Show the resulting schedule for one period.

(c) Can a smaller d_e be achieved on one or two processors using a different schedule? If so, show the schedule and achievable d_e. If not, will EDF* always be able to find the tightest uni- or multi-processor schedule that exists? Explain why or show a counter-example (e.g. a modification of the graph above for which EDF* does not achieve the smallest d_e).

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.



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.

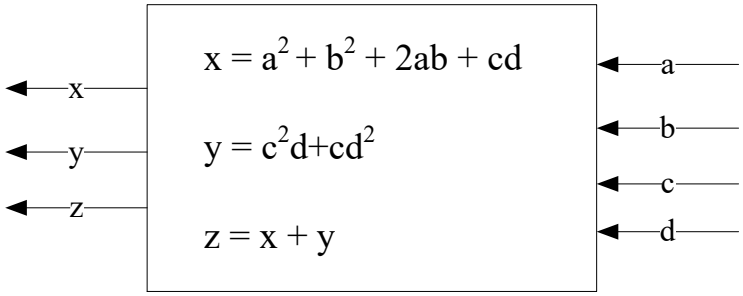
(a) Assuming unlimited resources, schedule the operations to compute the function in a minimum number of cycles.

(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) Derive minimum-latency ASAP and ALAP schedules.

(b) Derive the ILP formulation for a minimum-latency minimum-area scheduling of this system. Formulate an objective function that minimizes the total area and show the ILP equations for start time, dependency and resource constraints under a minimum latency assumption.

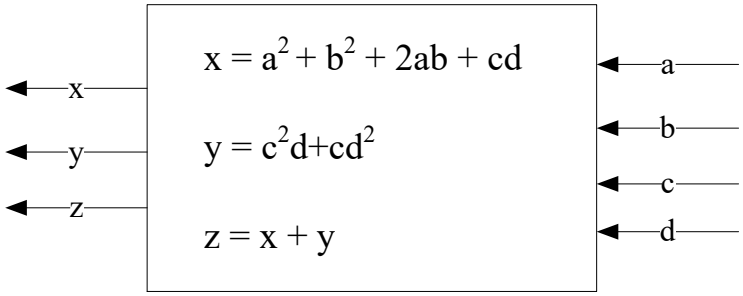
(c) Determine an optimal schedule and show the final area score as well as the corresponding assignment of ILP decision variables.

(d) Add additional ILP constraints to ensure that not more than 2 registers are used by the design internally. You don't have to determine the actual binding, just appropriate resource constraints. You can ignore inputs, i.e. assume that they are latched externally.

(e) Briefly sketch how energy minimization could be included in high-level synthesis and the ILP formulation. Similarly, how about peak power minimization?

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?

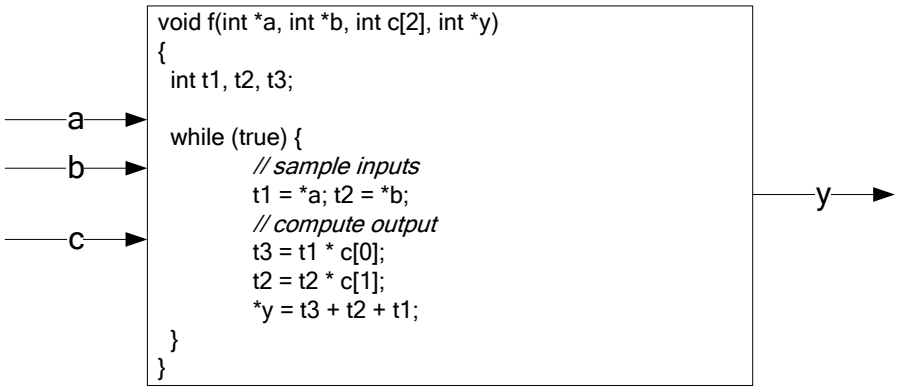
(d) For the FDS-generated implementation in (b), determine the variable lifetimes and assign variables to a minimum number of registers. Assume that primary input variables are preloaded into their assigned registers before the beginning of the computation.

(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.

Problem: High-Level Synthesis (35 points)

Consider the following system:



Assume a datapath resource constraint of one adder and one multiplier, where the adder requires one clock cycle while the multiplier is pipelined with a latency (delay) of two cycles and a throughput of one operation per cycle (i.e. a data introduction interval of 1).

(a) Derive a minimal-latency schedule for one iteration of the loop body inside *f* . How many cycles does it take to compute 100 output values?

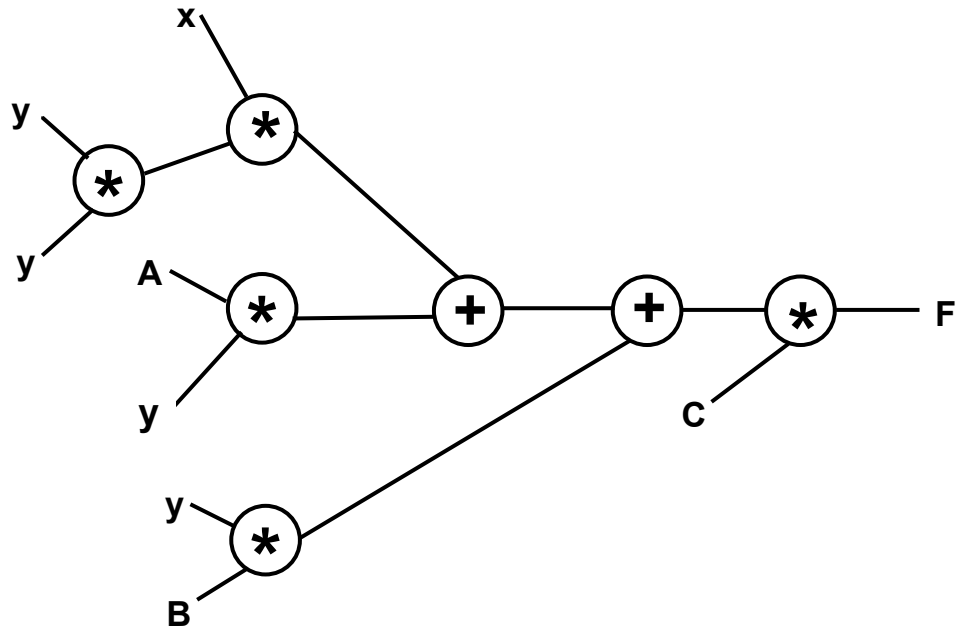
(b) Unroll the loop inside *f* one time and derive a minimal-latency schedule for one iteration of the new loop (which will contain two iterations of the original loop). How many cycles does it take to compute 100 output values?

(c) Instead of unrolling, pipeline the loop and derive a minimal-latency schedule for the pipelined loop body. Show the schedule for at least two overlapping loop iterations. What is the smallest loop introduction interval (II) that can be achieved? How many cycles does it take to compute 100 output values?

(d) Use the left-edge algorithm to determine a minimal set of registers and a corresponding register binding for your solution from (a).

Problem: High-Level Synthesis (25 points)

Consider the following dataflow graph (DFG) of a computation and an RTL resource library that contains 2-input adders with one cycle latency consuming *E* units of energy per addition and non-pipelined 2-input multipliers that require two cycles and 4*E* energy per multiplication.



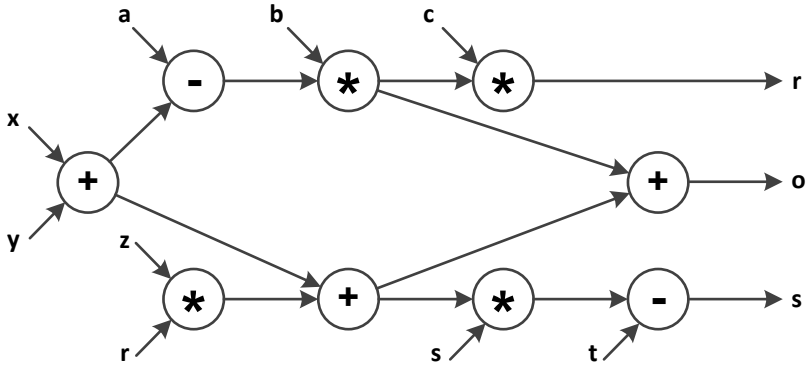
(a) Apply behavioral optimizations to the DFG in order to minimize the tree height in cycles, i.e. the ASAP schedule length. How many cycles does it take to execute the computation assuming unlimited resources?

(b) Apply behavioral optimizations in order to minimize the energy consumption without increasing the minimal latency required for the overall computation. What is the energy required for the original DFG, your DFG from (a) and the energy-optimized DFG?

(c) Assuming a resource allocation of one adder and one multiplier, apply a list scheduling algorithm using operation mobility as priority to schedule the graph into a minimum number of cycles. Show the steps of the algorithm and the final schedule and latency obtained.

Problem: High-Level Synthesis (30 points)

Consider the following dataflow graph (DFG):



(a) Assuming all operations execute in one cycle, can the graph be optimized and rewritten to minimize the ASAP schedule length? Show any such modifications and the ASAP and ALAP schedule of the final graph including all mobilities.

(b) Given a resource library that contains (non-pipelined) multipliers with 2 cycles latency and ALUs that require one clock cycle to execute. Assuming a resource allocation of one ALU and one multiplier, apply a list scheduling algorithm using an operation's distance to the sink as priority to schedule the original (unoptimized) graph into a minimum number of cycles. Show the final schedule and latency obtained.

(c) Assuming that input variables are already stored in separate external registers, use the left-edge algorithm to determine a minimal set of registers and a corresponding register binding for the intermediate variables in the scheduled DFG obtained in (b). Is a binding of the given schedule into a fewer number of registers possible or is the result optimal?

Problem: High-Level Synthesis (35 points)

Assume the following GEMM code written in Vitis High-Level-Synthesis (HLS) notation, where the *gemm_block* function multiplies 8x8 matrix blocks:

```
void gemm_block(unsigned int A[8][8], unsigned int B[8][8], unsigned
int C[8][8]) {
    int i,j,k;
    loop1: for (i=0; i<8; i++){
        loop2: for (j=0; j<8; j++){
            loop3: for (k=0; k<8; k++){
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

(a) Draw the control data flow graph (CDFG) for this function.

(b) Assuming no pipelining or unrolling of loops, no resource constraints and a mapping of function inputs and outputs *A*, *B* and *C* into separate single-ported SRAMs each, what is the expected latency of *gemm_block*() if multiplications and additions have a non-pipelined delay of 2 and 1 clock cycles, respectively, and SRAM accesses take 1 cycle per read/write?

(c) Given the same constraints and interface definition as in (a), what is the expected latency if the innermost loop is fully unrolled, i.e. a #pragma HLS unroll_factor=8 is applied to *loop3*? What limits the achievable speedup from unrolling? Without otherwise changing the loop or control structure (pipelining/unrolling/etc) itself, what is required to achieve the best latency of the unrolled version, and what is the ideal latency?

(d) What is the expected latency if *loop2* is pipelined in addition to fully unrolling *loop3* as in (c)? What limits the achievable initiation interval (II), and what is the II? What is required to achieve an II of 1, and what is the latency then?

(e) Assuming that a multiplication and an addition take 2 and 1 DSP slices to implement, respectively, how many DSPs do you expect to be utilized by a latency-optimal pipelined and unrolled implementation from (d)?

(f) Now assume that the optimized *gemm_block*() from (d) is part of a larger *gemm*() accelerator that repeatedly loads and stores 8x8 matrix blocks from/to external memory into/from local *A*, *B* and *C* block buffers in parallel with performing *gemm_block*() operations, i.e. where load/store and compute operations are overlapped. What is the minimal external bandwidth required to not be communication-bound?