University of Pennsylvania
Department of Electrical and Systems Engineering
Electronic Design Automation

ESE535, Spring 2009 | Assignment #1 | Wednesday, January 23

Due: Monday, February 2, beginning of class.

Resources You are free to use any books, articles, notes, or papers as references. Provide citations in your writeup as appropriate.

Collaboration Please work independently on this assignment.

Writeup Writeup should be in an electronically readable format (HTML or PDF preferred—I do not want to decipher handwriting or hand-drawn figures). State any assumptions you need to make.

Problems

1. [**2pts**] Consider the following C function:

```
int f(int a, int b, int c, int d)
{
   int t;
   t=(a+b)+(c+d);
   t=t>>2;
   a=a-t;
   b=b-t;
   c=c-t;
   d=d-t;
   if (a>b)  { t=a; a=b; b=t; }
   if (c>d)  { t=c; c=d; d=t; }
   if (b>d)  { t=b; b=d; d=t; }
   if (a>c)  { t=a; a=c; c=t; }
   return(d-a);
}
```

   (a) Create the dataflow graph for this function.

   (b) What is the critical path length for the resulting dataflow graph?

(c) Schedule the dataflow graph onto 3 ALUs.
Assume the ALU has an operation: res←mux(binary_condition, A, B) where

- binary_condition is a value you have previously computed (not an expression)
- res←A if binary_condition is true
- res←B if binary_condition is false

2. **[2pts]** Consider the graph in Figure 1. Assuming this is scheduled onto a device with 5 functional units, each of which takes two cycles to perform an operation:

(a) What is the latency bound?

(b) What is the compute bound?

(c) Show the schedule that results from using the algorithm shown in Figure 2 to schedule the graph ($F = 5$, $T_{op} = 2$). (*N.b.* This algorithm is not necessarily good; in fact we have seen better in class. We provide it as a strawman against which you can compare your better algorithm for the more interesting problem in the following question.)

3. **[5pts]** Consider an expanded architecture:

- 5 functional units that take 2 cycles to perform an operation and consume 1 Energy Unit (EU) per cycle.
- 1 functional unit that takes 1 cycle to perform an operation and consumes 4 EUs per cycle.
- Power constraints that limit you to operations that do not require more than 5 EUs per cycle. (*i.e.* you can run all 5 "slow" functional units, or you can run 1 fast functional unit and one slow).

(a) How would you calculate the compute bound for this architecture for any dataflow graph? (This is an equation.)

(b) What is the compute bound for the graph in Figure 1?

(c) How would you calculate the latency bound for this architecture for any dataflow graph?

(d) What is the latency bound for the graph in Figure 1?

(e) **[3pts]** Provide a general algorithm to schedule a dataflow graph onto this architecture minimizing compute time while obeying the power contraint above. (Describe in psuedocode similar to Figure 2.)

(f) Show the schedule that results from using your algorithm to schedule the graph shown in Figure 1.

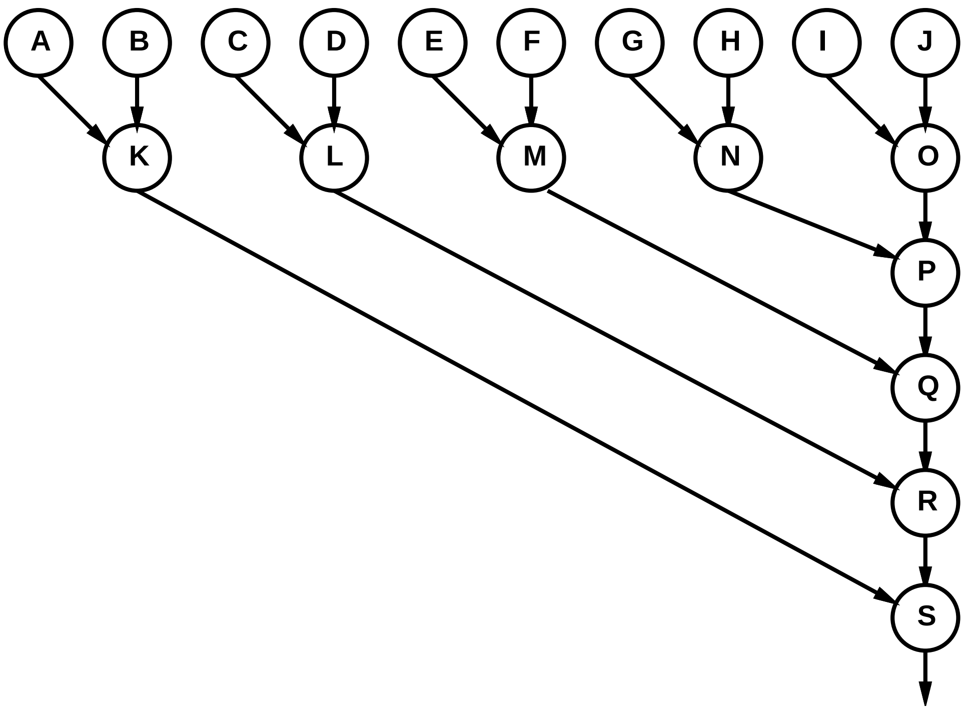

Figure 1: Sample Graph to Schedule

```
Input: Directed Dataflow Graph G = (V, E),
       Number of functional units F,
       Cycles per Operation T_op

//Unscheduled is a set that holds all unscheduled nodes.
Unscheduled=Empty Set
// ReadyQueue holds nodes that are ready to execute.
// Data extracted from Queue is in strict First-In-First-Out (FIFO) order.
ReadyQueue=Empty Queue
// Cycle is an integer indicating the cycle we are scheduling
Cycle=0
// Mark all nodes as unscheduled.
for each v_i ∈ V
    Unscheduled.Add(v_i)
    // Initialize the ReadyQueue with nodes which depend on nothing.
    if v_i has no predecessors in G
        ReadyQueue.Add(v_i)
// Assign nodes to functional units and time slots.
While (not(Empty(Unscheduled)))
    fu=0; // fu is a counter to keep track of assigned functional units
    Running=Empty Set // nodes scheduled at Cycle
    // Start as many nodes running as possible.
    while ((fu< F) and (not(Empty(ReadyQueue))))
        tmpNode=ReadyQueue.removeOldest()
        Assign tmpNode to Functional Unit fu at cycle Cycle
        Running.add(tmpNode)
    // Advance Cycle to completion time of these jobs.
    Cycle=Cycle+T_op
    // Mark nodes as schedule.
    for each node n_i ∈ Running
        Unscheduled.Remove(n_i)
    // Put all nodes enabled by the nodes which just completed into the ReadyQueue
    for each node n_i ∈ Running
        for each successor s_i to node n_i
        if no predecesors to s_i ∈ Unscheduled
            ReadyQueue.Add(s_i)
```

Figure 2: Simple First-Come-First-Served Scheduling Algorithm