Mapping DFGs to Hardware

As indicated in previous lectures, SDF graphs are particularly well-suited for mapping directly into hardware

For SDFs with *single-rate* schedules, all *actors* can execute in parallel using the same clock

The rules for mapping *single-rate* SDFs to hardware are straightforward

- All *actors* map to a single *combinational* hardware circuit
- All FIFO *queues* map to wires (without storage)
- Each initial token in a *queue* is replaced with a register

This means that two *actors* with no token on the *queue* between them will be effectively represented as two back-to-back combinational circuits

Timing requirements regarding the delay of the combinational logic must be met, i.e., all computation within the combinational logic **must complete in 1 clock cycle**

Therefore, in practice, the length of the *actor* sequence will be limited

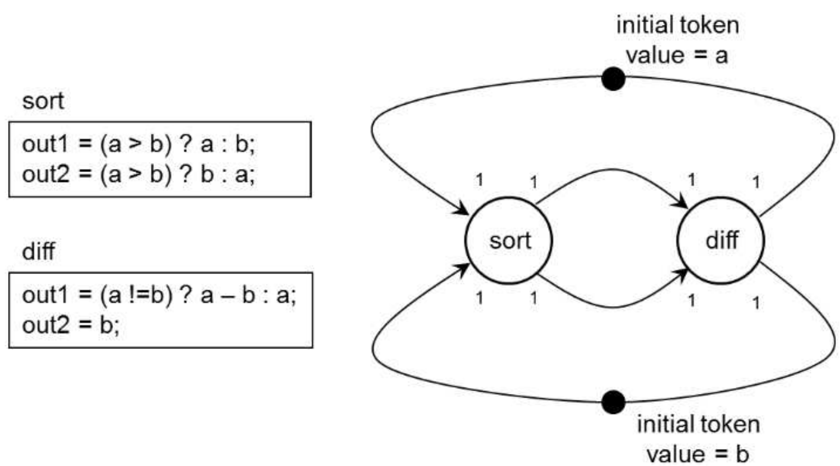
Mapping DFGs to Hardware

The performance annotations we've used in DFGs are only estimates of delay

Obtaining precise combinational circuit delays falls in the realm of CAD tools

We will treat the delay annotations for an *actor* as the **critical path delay**, i.e., the delay of the worst-case path in the *actor's* combinational circuit

Let's use Euclid's Greatest Common Divisor algorithm to illustrate the process



Euclid's GCD Algorithm

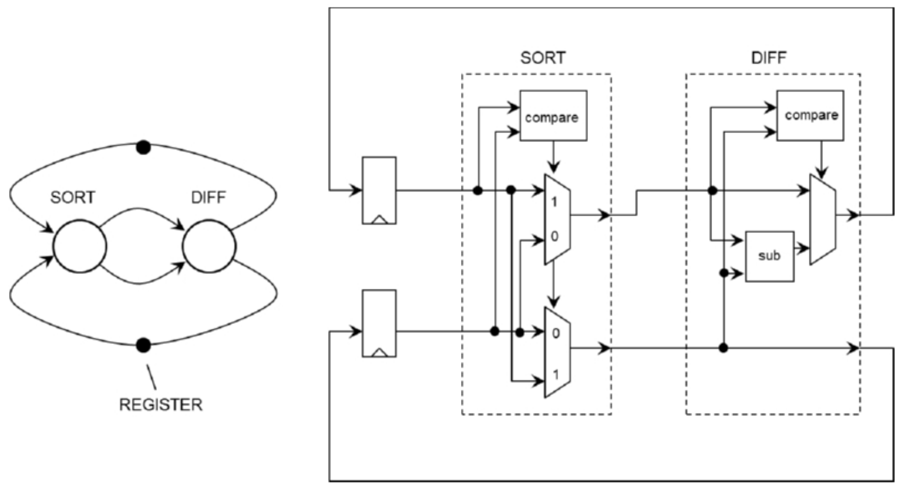
The *sort actor* reads the two initial token values *a* and *b*, sorts them and reproduces them on the output

The *diff actor* subtracts the smallest number from the largest one as long as they are not equal

Mapping DFGs to Hardware

You should convince yourself that a PASS exists using the techniques described earlier for SDFs

Following the mapping rules given above, we obtain the following circuit



The *sort* and *diff* actors are implemented using a comparator, a subtractor plus a few multiplexers

Note that the delay through the circuits of both *actors* define the maximum achievable clock frequency

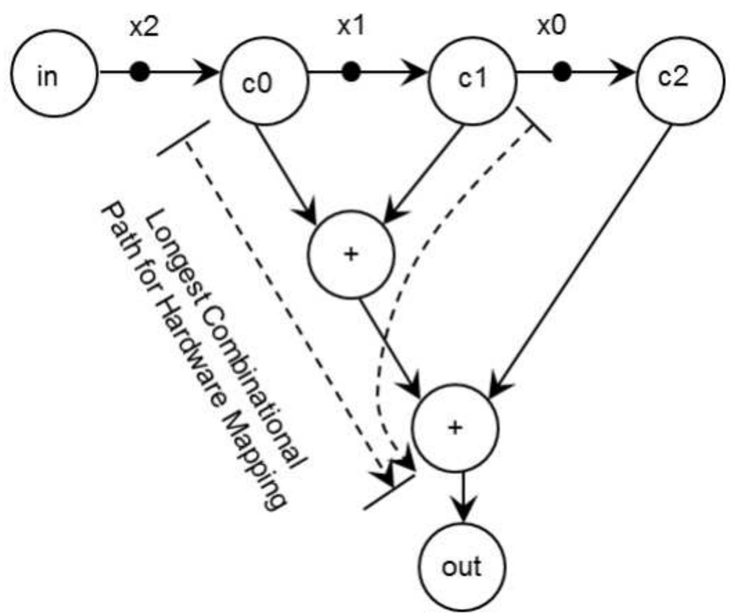
If the critical path delays are 5 and 15 ns, then max. operating freq. is 50 MHz

Mapping DFGs to Hardware: Pipelining

The **pipelining** transformation discussed earlier is a very popular technique to deal with path delays that limit clock frequency

Consider a dataflow specification of a **digital filter**

The filter computes a *weighted sum* of the samples arriving from the input stream as *out* = *x0.c2* + *x1.c1* + *x2.c0*

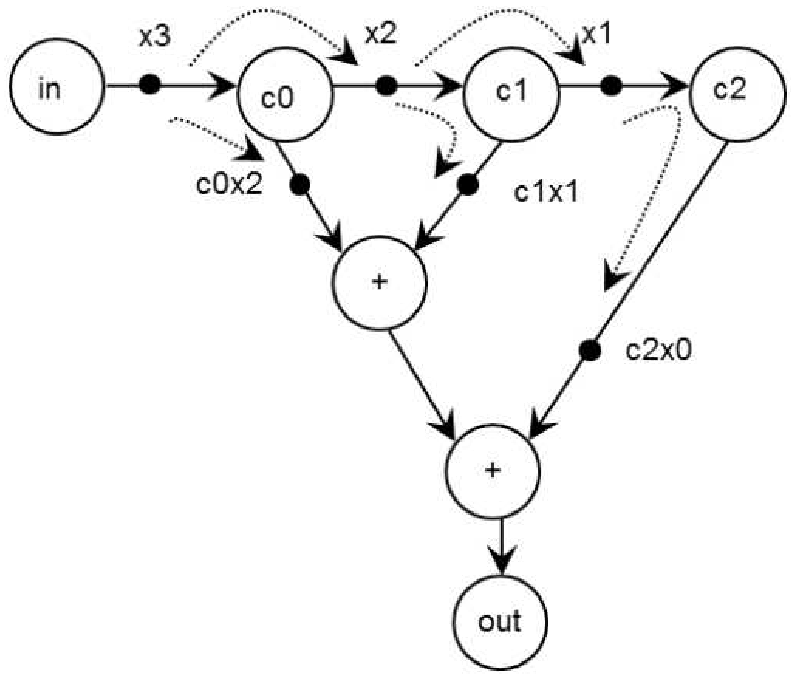


The critical path of this graph is associated with the one of the multiplier *actors*, *c0* or *c1*, in series with two addition *actors*

Mapping DFGs to Hardware: Pipelining

Pipeling allows additional tokens to be added (at the expense of increase latency)

Here, one is introduced by the *in* actor



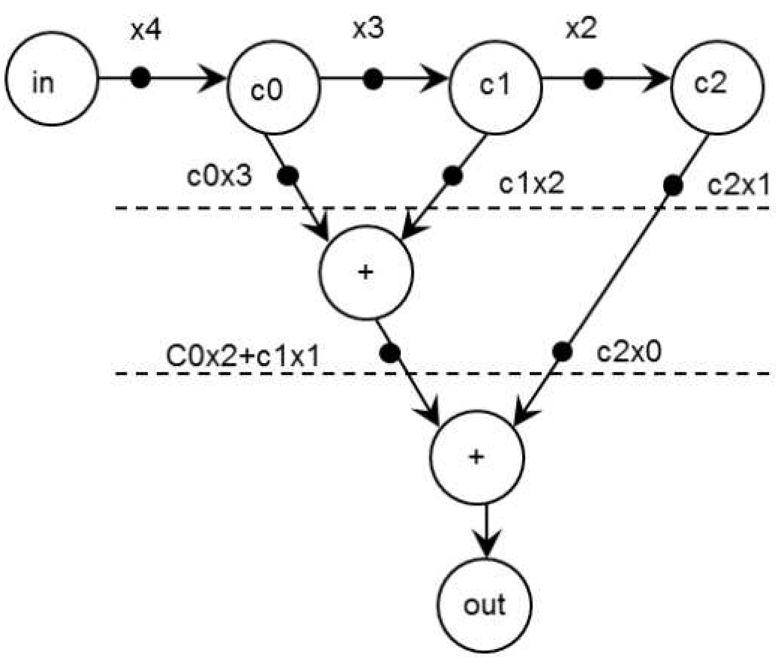
And then retiming is used to redistribute the tokens, as shown by the above marking, which is produced after *firing c0*, *c1* and *c2*

Retiming creates additional registers and an additional pipeline stage

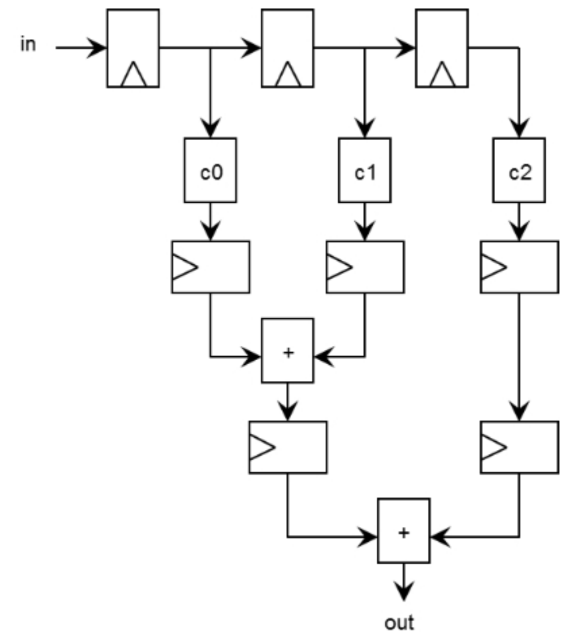
The critical path is now reduced to two back-to-back adder *actors*

Mapping DFGs to Hardware: Pipelining

This final marking is obtained by allowing the *in actor* to add one more token and then using retiming to *fire c0*, *c1*, *c2* and the top add *actor*



The resulting **pipelined** DFG implementation

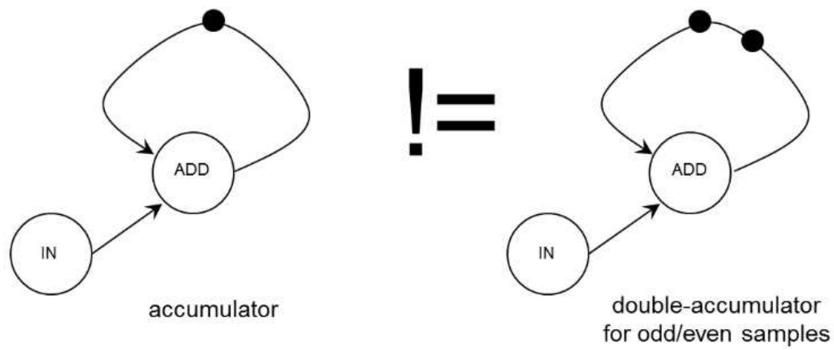


The schematic on the right shows a fully pipelined implementation

Note that it is **not** possible to introduce *arbitrary* initial tokens in a graph *without* following the actor's *firing* rules

Doing so would likely change the behavior of the system

Mapping DFGs to Hardware: Pipelining



This change in behavior is obvious in the case of feedback loops, as shown here for an *accumulator* circuit

- Using a **single** *token* in the feedback loop of an add *actor* will accumulate all input samples, as shown on the left
- Using **two** *tokens* in the feedback loop will accumulate the odd samples and even samples separately

When pipelining a DFG, be sure to follow the rules for subsequent markings

When adding new *tokens*, add them only at the input or output of the DFG, outside of any loops

And then use retiming to redistribute the *tokens* to reduce critical path delay