Hardware Implementation of Data-Flow Model (A Practical Introduction to HW/SW Codesign, P. Schaumont)

We can implement SDF actors as dedicated hardware engines.

It is possible to map FIFO queues and actor firing rules directly into hardware. We are particularly interested in simple, optimized implementations.

Single-rate SDF Graphs

Consider an SDF graph that has a direct, one-to-one mapping to hardware elements. Each actor translates to a single combinational hardware module, and each FIFO queue translates to wires or registers.

Fig. 2.22 Hardware Implementation of Euclid’s algorithm
Hardware Implementation of Data-Flow Model

Procedure:

- Map each queue to a wire
- Map each queue containing a token to a register
  The initial value of the token sets the initial value of the register
- Map each actor that completes a firing within a clock cycle to a combinational circuit
  The sort and diff actors require a comparator, subtractor plus a few multiplexers

Note that this translation procedure works on the following SDF types:

- A single-rate data-flow graph, which has a PASS firing vector with all '1’s in it
- All actors can be implemented using combinational logic

In addition, the above method may result in circuits with long combinational paths

Even with these restrictions, the basic concept of the transformation is useful, particularly when used with the transformations discussed below
Hardware Implementation of Data-Flow Model

Pipelining: Pipelining of data-flow graphs can be used to break long combinational paths

Consider a dataflow specification of a digital filter

It evaluates a weighted sum of samples of an input stream, with the sum defined as \( out = x_0.c2 + x_1.c1 + x_2.x0 \)

![Diagram of SDF graph of a simple moving-average application](image)

Fig. 2.23 SDF graph of a simple moving-average application
**Hardware Implementation of Data-Flow Model**

The critical path of this graph is equal to a constant multiplication (with $c_0$ or $c_1$) and two additions.

Consider what happens when we ’push down’ the initial tokens into the adder tree:

```
Fig. 2.24  Pipelining the moving-average filter by inserting additional tokens (1)

Here, the *in* actor produces an additional token, $x_3$, and $c_0$, $c_1$, $c_2$ and *add* actors fire. This causes tokens to appear on queues that had **no** tokens previously.
Hardware Implementation of Data-Flow Model

By letting the in actor produce another token, the longest combinational path is further reduced to a single addition.

The resulting pipelined data-flow graph implementation

The moving-average filter by inserting additional tokens (2)

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.
Hardware Implementation of Data-Flow Model

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 data-flow graph, be sure to follow the normal steps for data-flow marking.

For example, do not introduce any initial token unless it can be obtained by a sequence of firings from actors.

Fig. 2.27 Loops in SDF Graphs cannot be pipelined
Hardware Implementation of Data-Flow Model: Multi-rate Expansion

Single-rate dataflow graphs can be directly mapped into hardware, and are therefore an advantage over multi-rate

However, we can transform multi-rate data-flow graphs systematically to single-rate dataflow graphs by:

• Determine the PASS firing rates of each actor
• Duplicate each actor the number of times indicated by its firing rate

For example, given an actor A with a firing rate of 2, we create identical actors A0 and A1

• Convert each multi-rate actor input/output to multiple single-rate input/outputs

For example, if an actor input has a consumption rate of 3, we replace it with three single-rate inputs

• Re-introduce the queues in the dataflow system to connect all actors

• Re-introduce the initial tokens in the system, distributing them sequentially over the single-rate queues
**Hardware Implementation of Data-Flow Model: Multi-rate Expansion**

Consider the following example in which actor $A$ produces three tokens per firing, and actor $B$ consumes two tokens per firing.

After completing the steps above, we obtain the following dataflow graph:

![Multi-rate data flow-graph](Fig. 2.28)

Here, the actors are duplicated according to their firing rates, and all multi-rate I/O are converted to single-rate I/O.
Hardware Implementation of Data-Flow Model: Multi-rate Expansion

The distribution of initial tokens follows the sequence of queues between $A$, $B$ (i.e. they follow the order $a$, $b$, etc.)

The resulting single-rate graph can now be mapped directly into a hardware circuit. This circuit will have two inputs ($in0$, $in1$) and three outputs ($out0$, $out1$, $out2$), which corresponds to the original specification.

The inputs are consumed at rate 2, and the outputs are being produced at rate 3.