Lecture 9: Dataflow MoCs

Seyed-Hosein Attarzadeh-Niaki

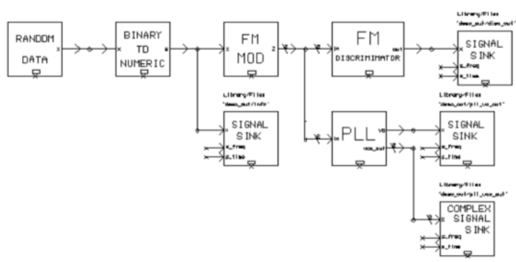
Some Slides due to Edward Lee

Review

- Composition of state machines
  - Synchronous composition
  - Asynchronous composition
- Hierarchical composition: StateCharts

Dataflow Models

- Proven to be useful for specifying *streaming applications*
  - e.g., signal processing and communications domains
- Simulation of the algorithm at the functional or behavioral level
- Synthesis to software (e.g., a C program) or hardware (e.g., VHDL)
- Block-diagram based visual programming



Dataflow Signals (Streams)

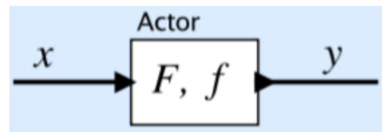
- In dataflow signals, communication between the actors is done as *sequences* of messages
- Each message is called a *token*.
- A signal *s* is a partial function of the form

$$s: N \rightharpoonup V_s$$

V_s is the type of the signal
- The signal is defined on an initial segment $\{0, 1, \ldots, n\} \subset N$, or (for infinite executions) on the entire set N.
- Unlike the synchronous reactive MoC, no necessary time relation between inputs and outputs of an actor.
  - They are not synchronized.

Dataflow Actors

- A (determinate) actor will be described as a function that maps input sequences to output sequences
- An *actor function* *F*, maps *entire* input sequences to *entire* output sequences
- A *firing function* *f*, maps a finite portion of the input sequences to output sequences
- A *firing rule* specifies the number of tokens required on each input port in order to fire the actor

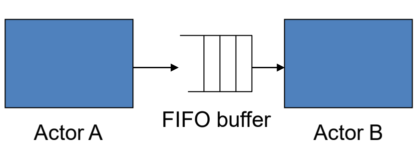


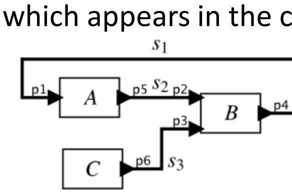
$$F(x_1,x_2,x_3,\cdots) = (ax_1,ax_2,ax_3,\cdots)$$

e.g., $f(x_1) = (ax_1)$

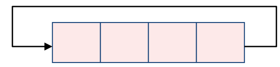
Composition of Dataflow Actors: Buffered Communication

- Since the firing of the actors is asynchronous, a token sent from one actor to another must be buffered.
- When the destination actor fires, it *consumes* one or more input tokens.
- Being able to execute a dataflow model forever is called an *unbounded execution*.
  - We need scheduling policies that deliver *bounded buffers.*
  - *Deadlock* is another challenge which appears in the cycles.

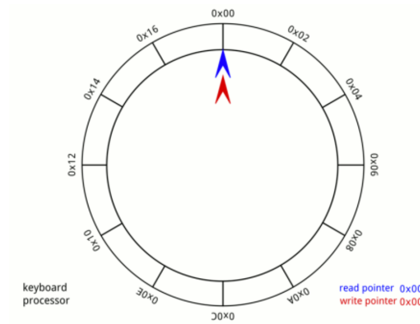




Implementing Buffers for Dataflow Models



- Unbounded buffers require memory allocation and deallocation schemes.
- Bounded size buffers can be realized as *circular buffers* or *ring buffers*, in a statically allocated array.
  - A *read pointer* r is an index into the array referring to the first empty location. Increment this after each read.
  - A *fill count* n is unsigned number telling us how many data items are in the buffer. (or *a tail pointer*)
  - The next location to write to is $(r + n)$ modulo buffer length.
  - The buffer is empty if $n == 0$
  - The buffer is full if $n ==$ buffer length
  - Can implement n as a (counting) semaphore, providing mutual exclusion for code that changes n or r.



Facts About (General) Dynamic Dataflow

- Whether there exists a schedule that does not *deadlock* is undecidable.
- Whether there exists a schedule that executes forever with *bounded memory* is undecidable.

Undecidable means that there is no algorithm that can answer the question in finite time for all finite models.

However, there are special cases where the models are analyzable

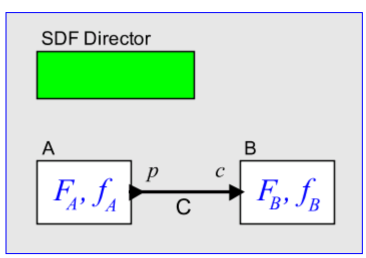
Dataflow: many variants, still active area of research

- Computation graphs [Karp & Miller - 1966]
- Process networks [Kahn - 1974]
- Static dataflow [Dennis - 1974]
- Dynamic dataflow [Arvind, 1981]
- K-bounded loops [Culler, 1986]
- Synchronous dataflow [Lee & Messerschmitt, 1986]
- Structured dataflow [Kodosky, 1986]
- PGM: Processing Graph Method [Kaplan, 1987]
- Synchronous languages [Lustre, Signal, 1980's]
- Well-behaved dataflow [Gao, 1992]
- Boolean dataflow [Buck and Lee, 1993]
- Multidimensional SDF [Lee, 1993]
- Cyclo-static dataflow [Lauwereins, 1994]
- Integer dataflow [Buck, 1994]
- Bounded dynamic dataflow [Lee and Parks, 1995]
- Heterochronous dataflow [Girault, Lee, & Lee, 1997]
- Parameterized dataflow [Bhattacharya and Bhattacharyya 2001]
- Structured dataflow (again) [Thies et al. 2002]
- …

now

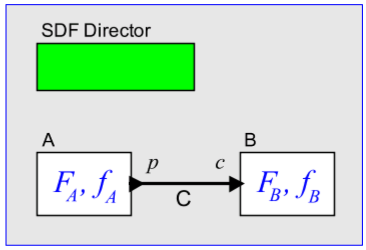
Lee 09: 6

Synchronous Dataflow (SDF)



- The number of tokens consumed and produced by the firing of an actor is constant
- Static analysis can tell us whether we can schedule the firings to get a useful execution
- If so, then a finite representation of a schedule for such an execution can be created.

Balance Equations



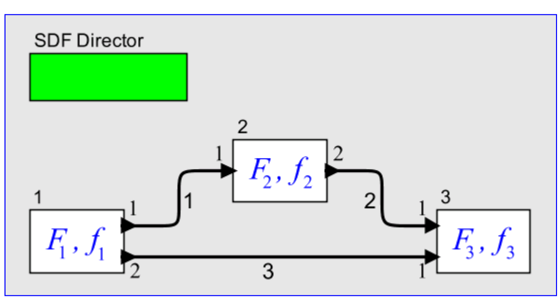
Let q_A, q_B be the number of firings of actors A and B.
Let p_C, c_C be the number of tokens produced and consumed on a connection C.
Then the system is *in balance* if for all connections C

$$q_A p_C = q_B c_C$$

where A produces tokens on C and B consumes them.

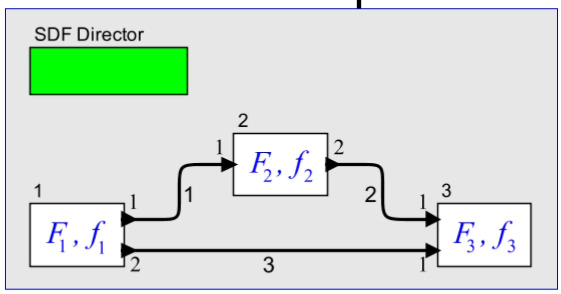
Example

Consider this example, where actors and arcs are numbered:



The balance equations imply that actor 3 must fire twice as often as the other two actors.

Compactly Representing The Balance Equations



production/consumption matrix (topology matrix)

Connector 1

$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$

Actor 1

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

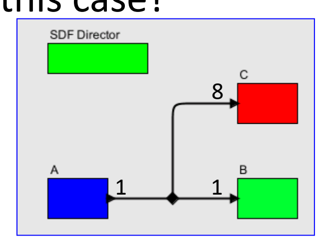
Firing vector (Repetition vector)

balance equations

$$\Gamma q = \vec{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

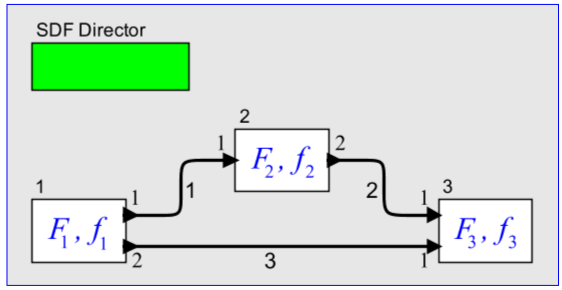
Question

- What is the production/consumption matrix in this case?



$$\Gamma = \begin{bmatrix} 1 & 0 & -1 \\ 1 & -8 & 0 \end{bmatrix}$$

Example

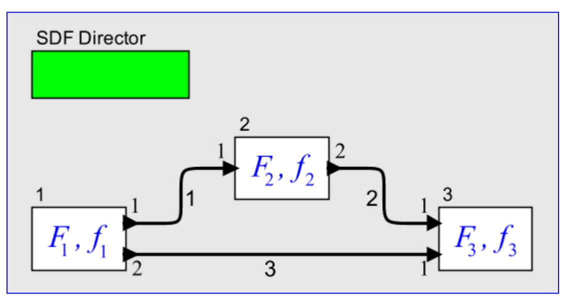


A solution to the balance equations:

$$q = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \qquad \Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & -1 \\ 2 & 0 & -1 \end{bmatrix} \qquad \Gamma q = \vec{0}$$

This tells us that actor 3 must fire twice as often as actors 1 and 2.

Example



But there are many solutions to the balance equations:

$$q = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \quad q = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad q = \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix} \quad q = \begin{bmatrix} -1 \\ -1 \\ -2 \end{bmatrix} \quad q = \begin{bmatrix} \pi \\ \pi \\ 2\pi \end{bmatrix} \qquad \Gamma q = \vec{0}$$

For "well-behaved" models, there is a unique least positive integer solution.

Least Positive Integer Solution to the Balance Equations

- Note that: if p_C, c_C , the number of tokens produced and consumed on a connection C, are non-negative integers, then the balance equation,

$$q_A p_C = q_B c_C$$

  implies:

  q_A is rational if and only if q_B is rational.

  q_A is positive if and only if q_B is positive.

- **Consequence**: Within any connected component, if there is any non-zero solution to the balance equations, then there is a unique least positive integer solution.

Rank of a Matrix

- The rank of a matrix Γ is the number of linearly independent rows or columns. The equation

$$\Gamma q = \vec{0}$$

  is forming a linear combination of the columns of Γ.
- Such a linear combination can only yield the *zero vector* if the columns are linearly dependent (this is what is means to be *linearly dependent*).
- If Γ has a columns and b rows, the rank cannot exceed $\min(a, b)$.
- If the columns or rows of Γ are re-ordered, the resulting matrix has the same rank as Γ.

Rank of the Production/Consumption Matrix

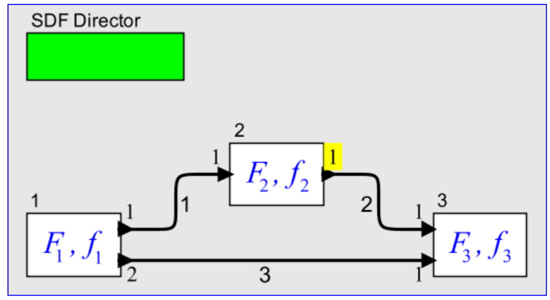
- Let a be the number of actors in a connected graph. Then the *rank* of the production/consumption matrix Γ is $\leq a$. (why?)
- If the model is a *spanning tree* (meaning that there are barely enough connections to make it connected) then Γ has a rows and $a-1$ columns.
- Theorem [Lee-Messerschmitt' 87]: Its rank is $a-1$.
  - Exercise: Prove it. (Hint: use induction).
- Corollary: the rank of any production/consumption matrix of a connected graph is either a or $a-1$. (why?)

Consistent Models

- Let a be the number of actors in a connected model. The model is *consistent* if Γ has rank $a-1$.
- If the rank is a, then the balance equations have only a trivial solution (zero firings).
- When Γ has rank $a-1$, then the balance equations always have a non-trivial solution.

Example of an Inconsistent Model: No Non-Trivial Solution to the Balance Equations

$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$



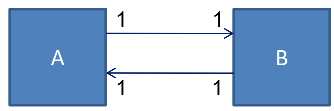
- This production/consumption matrix has rank 3, so there are no nontrivial solutions to the balance equations.
- Note that this model can execute forever, but it requires unbounded memory.

Necessary and sufficient conditions

Consistency is a necessary condition to have a (bounded-memory) infinite execution.

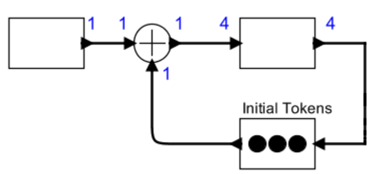
Is it sufficient?

Deadlock 1



Is this diagram consistent?

Deadlock 2



Some dataflow models cannot execute forever. In the above model, the feedback loop injects initial tokens, but *not enough* for the model to execute.

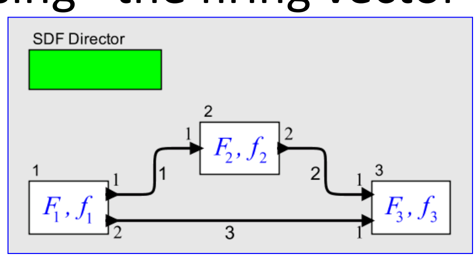
SDF: from static analysis to scheduling

- **Given**: SDF diagram
  Find: a bounded-buffer schedule, if it exists
- Step 0: check whether diagram is consistent. If not, then no bounded-buffer schedule exists.
- Step 1: find an integer solution to $\Gamma q = 0$.
- Step 2: "decompose" the solution q into a schedule, making sure buffers never become negative.

Step 2: "decomposing" the firing vector

Example 1:

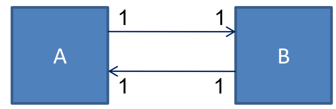
Schedule = (1;2;3;3)



$$b = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \; q = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \xrightarrow{\text{Fire 1}} b = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \; q = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \xrightarrow{\text{Fire 2}} b = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix}, \; q = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} \xrightarrow{\text{Fire 3}} b = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \; q = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \xrightarrow{\text{Fire 3}} b = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \; q = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Step 2: “decomposing” the firing vector

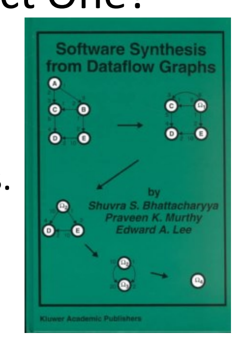
Example 2:



What happens if we try to run the previous procedure on this example?

So, we have both necessary and sufficient conditions for scheduling SDF graphs.

A Key Question: If More Than One Actor is Fireable in Step 2, How do I Select One?



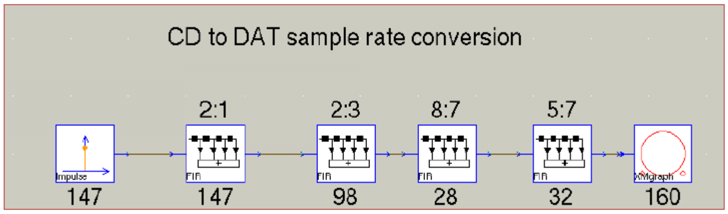
Optimization criteria that might be applied:

- Minimize buffer sizes.
- Minimize the number of actor activations.
- Minimize the size of the representation of the schedule (code size).

See S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs,* Kluwer Academic Press, 1996.

Beyond our scope here, but hints that it’s an interesting problem…

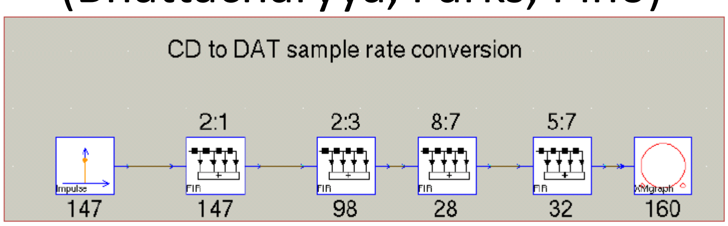
Minimum Buffer Schedule



A B A B C A B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C D E
A F F F F F B C A B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C
D E A F F F F F B A B C A B C A B A B C A B C D E A F F F F F B A B C A B C A
B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F F F F F E B C A
F F F F F B A B C A B C D E A F F F F F B A B C A B C A B A B C A B C D E A F
F F F F B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C A B A B C
D E A F F F F F B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C A
B C D E A F F F F F B A B C A B C A B A B C D E A F F F F F E B C A F F F F F B
A B C A B C A B A B C D E A F F F F F B C A B A B C A B C D E A F F F F F B A
B C A B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C D E A F F F
F F B C A B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C D E A F
F F F F B A B C A B C A B A B C A B C D E A F F F F F E B A F F F F F B C A B C
A B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F F F F F B C A
B A B C A B C D E A F F F F F B A B C A B C A B A B C A B C D E A F F F F F B
A B C A B C A B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F
F F F F B C A B A B C A B C D E F F F F F F E F F F F F

Source: Shuvra Bhattacharyya

Scheduling Tradeoffs (Bhattacharyya, Parks, Pino)

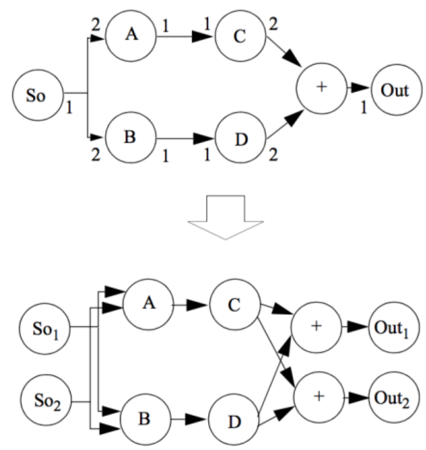


Scheduling strategy	Code	Data
Minimum buffer schedule, no looping	13735	32
Minimum buffer schedule, with looping	9400	32
Worst minimum code size schedule	170	1021
Best minimum code size schedule	170	264

Source: Shuvra Bhattacharyya

Homogeneous SDF

- An SDF graph in which every actor consumes and produces only one token from each of its inputs and outputs
- A general, consistent SDF graph that is not an HSDFG *can always be converted* into an equivalent HSDFG



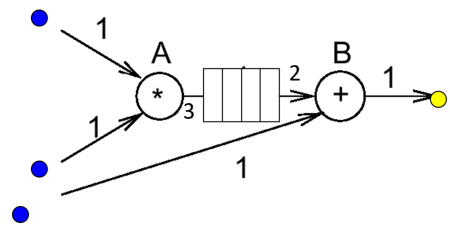
Graph-Based Techniques For (H)SDF Analysis

- Obtaining Feasible schedules using shortest paths in graphs
- Maximum Cycle Mean (MCM): Used to obtain the maximum available throughput of a self-timed HSDF graph ($throughput_{max}$ = 1/MCM).

$$MCM(G) = \max_{\text{cycle } C \text{ in } G} \left\{ \frac{\sum_{v \text{ is on } C} t(v)}{Delay(C)} \right\}$$

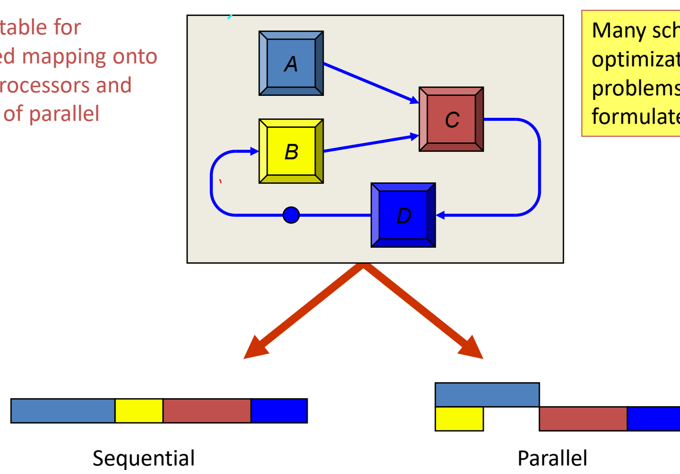
Reflection

- How can we model limited buffer sizes in SDF?



Parallel Scheduling of SDF Models

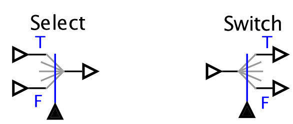
SDF is suitable for automated mapping onto parallel processors and synthesis of parallel circuits.



Many scheduling optimization problems can be formulated.

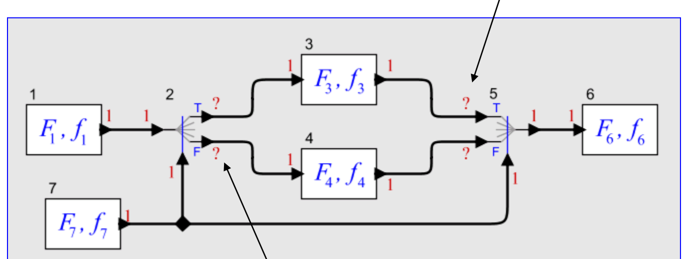
Boolean Dataflow (BDF)

- SDF cannot directly express conditional firing
  - e.g., where an actor fires only if a token has a particular value.
- Two basic actors SWITCH and SELECT are added in boolean datflow
- Any Turing machine can be expressed as a BDF graph



Boolean Dataflow

What consumption rate?



What production rate?

Imperative equivalent:

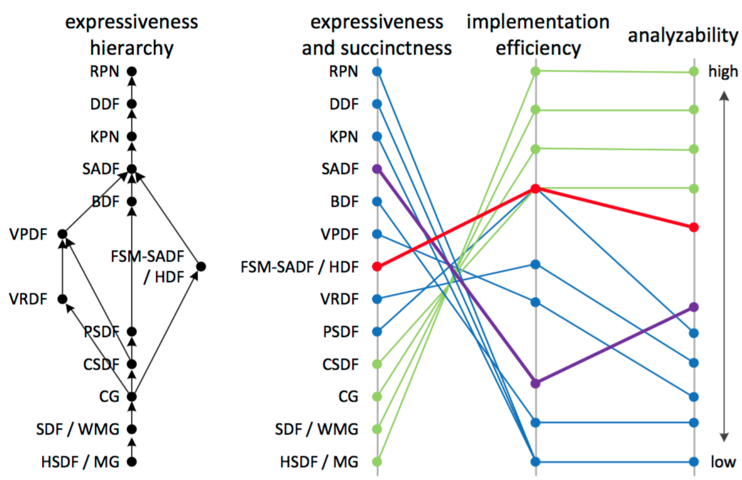
```
while (true) {
   x = f1();
   b = f7();
   if (b) {
     y = f3(x);
   } else {
     y = f4(x);
   }
   f6(y);
}
```

The if-then-else model is not SDF. But we can clearly give a bounded *quasi-static* schedule for it:

(1, 7, 2, b?3, !b?4, 5, 6)

guard

Comparison of Dataflow MoCs



Stuijk et al, SAMOS'11

Dataflow Modeling in Simulink Using Dataflow Domain in the DSP Toolbox

- Blocks inside the dataflow domain execute based on the availability of data as opposed to Simulink's sample time.
- Simulink automatically partitions the system into concurrent threads to increase data throughput for accelerated simulation and multicore code generation.