Main application of SDF: DSP hardware modeling

Orthogonal Frequency Division Multiple Access (OFDMA) application modeled in (timed) SDF:

IFFT and FIR actors are configurable IPs from the Xilinx Coregen library, while the rest are user defined actors (source: National Instruments, e.g., see [Tripakis et al., 2011, Tripakis et al., 2012]).

Where do execution times come from?

- For existing hardware: measure clock cycles.
- To-be-synthesized hardware: estimate (harder problem).
Timed SDF

- 3, 7, 5: execution times
  - how long it takes an actor to complete a firing
  - we will assume discrete time, can be extended to dense time
Timed SDF semantics

A Timed SDFG defines a labeled transition system:

- States record:
  - number of tokens in every channel;
  - number of instances of every actor running;
  - how much time is left for each instance until it completes its firing.

- Transitions: 3 types
  - a new instance begins firing;
  - an instance ends firing;
  - time passes.

Example

This timed SDFG defines the following LTS:

\[
\begin{align*}
(g = (0, 0), h = \{\}) \xrightarrow{\text{begin } A} (g = (0, 0), h(A) = \{3\}) \\
\xrightarrow{\text{tick}} (g = (0, 0), h(A) = \{2\}) \\
\xrightarrow{\text{begin } A} (g = (0, 0), h(A) = \{3, 3\}) \\
\end{align*}
\]
Timed SDF formal semantics

A Timed SDFG defines a **labeled transition system**:

- **State**: a tuple \((g, h)\) containing
  - a vector \(g\) describing how many tokens are in every channel
  - a vector \(h\) describing
    - how many instances of each actor are running and
    - how much time is left until each instance completes its firing.
  - \(h\) associates to each actor \(A\) a **multiset** \(h(A)\), i.e., a set which may contain multiple “copies” of the same element, e.g., \(\{2, 2, 0\}\).
  - Why a multiset? 
    \(⇒\) could have multiple instances of the same actor active at the same time (parallelism, called **autoconcurrency** in SDF jargon).

- **Initial state** (unique):
  - \(g(c)\) = number of initial tokens of channel \(c\)
  - \(h(A) = \emptyset\) for all \(A\) (no instances running initially).

Timed SDF formal semantics

Transitions: of 3 possible forms

- \((g, h) \xrightarrow{\text{begin}A} (g', h')\): a new instance of \(A\) begins to fire.
  - Precondition on \(g\): enough tokens must be in the input queues of \(A\).
  - \(g'\): obtained by \(g\) by removing the corresponding tokens from the input queues of \(A\).
  - \(h'(A)\): add to \(h(A)\) the execution time of \(A\).

- \((g, h) \xrightarrow{\text{tick}} (g, h')\): one time unit elapses.
  - Precondition on \(h\): there is no \(A\) such that \(0 \in h(A)\) (this would mean one actor instance is ready to finish firing).
  - \(h'\): decrement all actor instance execution times by 1.

- \((g, h) \xrightarrow{\text{end}A} (g', h')\): one instance of \(A\) finishes firing.
  - Precondition on \(h\): \(0 \in h(A)\).
  - \(h'\): remove one 0 from \(h(A)\).
  - \(g'\): add to the output queues of \(A\) the tokens that \(A\) produces.
This timed SDFG defines the following LTS:

\[(g = (0, 0), h = \{\}) \xrightarrow{\text{begin} A} (g = (0, 0), h(A) = \{3\}) \xrightarrow{\text{tick}} (g = (0, 0), h(A) = \{2\}) \xrightarrow{\text{begin} A} (g = (0, 0), h(A) = \{3, 3\}) \cdots\]

As with untimed SDF, this LTS is generally infinite.

Forbidding autoconcurrency

How to forbid a new instance from starting until the previous one finishes?
Pictorially

with autoconcurrency

without autoconcurrency

ASAP LTS ("self-timed execution")

The *As Soon As Possible (ASAP)* LTS of a Timed SDFG is defined as before, with the additional rule that:

*A tick transition is allowed only if no begin transitions are enabled.*

Justification: to estimate throughput, no need to delay actor firings unnecessarily.

\[\text{\footnotesize 1} \text{also if no end transitions are enabled, but that was already the case in the original semantics}\]
Determinacy of ASAP execution

We can group together all non-\textit{tick} transitions in a single transition:

- Group everything that happens instantaneously.
- Deterministic! (as with Kahn networks – interleavings don't affect final result)

Similarly with \textit{end} transitions.
Strongly-connected (SDF) Graphs

Where every node (actor) can be reached by a directed path from every other node.

Strongly-connected:

Not strongly-connected:

Determinacy and finiteness of ASAP execution

**Theorem ([Ghamarian et al., 2006])**

*In consistent and strongly-connected timed SDFGs the deterministic ASAP execution is a lasso (an initial segment followed by a cycle).*

A lasso:
Example: lasso of deterministic ASAP execution

Timed SDFG:

```
\begin{tikzpicture}
  \node (A) at (0,0) {$A$};
  \node (B) at (2,0) {$B$};
  \draw[->] (A) to node [above] {2} (B);
  \draw[->] (B) to node [below] {1} (A);
  \draw[->] (A) to node [below] {3} (B);
  \draw[->] (B) to node [above] {7} (A);
\end{tikzpicture}
```

Deterministic ASAP execution:

```
\begin{align*}
(g = (0,1), h = \{\}) & \xrightarrow{\text{begin}_A} (g = (0,0), h(A) = \{3\}) & \xrightarrow{\text{tick}} (g = (0,0), h(A) = \{2\}) & \xrightarrow{\text{tick}} (g = (0,0), h(A) = \{0\}) \\
(g = (2,0), h(A) = \{0\}, h(B) = \{\}) & \xrightarrow{\text{end}_A} (g = (2,0), h(A) = \{\}, h(B) = \{\}) & \xrightarrow{\text{begin}_B} (g = (0,0), h(A) = \{\}, h(B) = \{7\}) & \xrightarrow{\text{end}_B}
\end{align*}
```

Determinacy and finiteness of ASAP execution

**Theorem ([Ghamarian et al., 2006])**

*In consistent and strongly-connected timed SDFGs the deterministic ASAP execution is a lasso (an initial segment followed by a cycle).*

Does the theorem hold also in graphs that deadlock?

Yes: *tick* transitions are always possible (time cannot be stopped).

```
\begin{tikzpicture}
  \node (1) at (0,0) {};
  \node (2) at (1,0) {};
  \node (3) at (2,0) {};
  \node (4) at (3,0) {};
  \node (5) at (4,0) {};
  \draw[->] (1) to (2);
  \draw[->] (2) to (3);
  \draw[->] (3) to (4);
  \draw[->] (4) to (5);
  \draw[->] (5) to (1);
  \node (6) at (3.5,0) {\text{\textit{tick}}};
\end{tikzpicture}
```
Strongly-connected SDFGs ⇒ finite state-space and bounded queues

Strongly-connected ⇒ state space is finite ⇒ queues remain bounded.

What about non-strongly-connected SDFGs?

Non-strongly-connected SDFGs

A can fire arbitrarily many times in parallel (autoconcurrency) ⇒ queue from A to B grows unbounded ⇒ infinite state-space.
Non-strongly-connected SDFGs

What about timed SDFGs without autoconcurrency?

\[
\begin{array}{ccc}
A & \xrightarrow{2} & B \\
3 & & 7 & \xrightarrow{1} & C \\
& & 2 & \xrightarrow{2} & \\
& & & & 5
\end{array}
\]

\(A\)'s rate of firing (throughput) only depends on its execution time.

Rate of firing of \(B\), \(C\) (downstream strongly-connected sub-graph) constrained by that of \(A\) (upstream strongly-connected sub-graph).

⇒ We will analyze throughput separately for each strongly-connected sub-graph.

What about this example?

The SDFG below is not strongly connected:

\[
\begin{array}{cccccccc}
\text{Src} & \xrightarrow{1} & \text{ZeroPad} & \xrightarrow{600} & \text{IFFT} & \xrightarrow{2048} & \text{FIR} & \xrightarrow{2560} & \text{Sink} \\
1 & & 2048 & \text{ET}=1 & 2048 & \text{ET}=7838 & 24 & \text{ET}=144 & 1 \\
& & \text{ET}=2048 & & \text{ET}=7838 & & \text{ET}=144 & & \text{ET}=1
\end{array}
\]

Does it mean we won’t analyze throughput for this application?

In practice:

- Problem: optimize buffer size.
- Target throughput: 25 M samples/sec.

⇒ buffer size is an input to the problem ⇒ finite queues ⇒ strongly-connected graph.

\[
\begin{array}{c}
P_1 \\
\text{queue of size } k \\
P_2
\end{array} = \begin{array}{c}
P_1 \\
1 \\
\text{k} \\
P_2
\end{array}
\]
THROUGHPUT ANALYSIS

Throughput

Throughput (in general) = average rate.
- e.g., communication networks: rate of data transmission

In our case:
- average rate of token productions (in given channel)
- average rate of actor firings (for given actor)
- these are multiples of each other:
  - let $\mu_A$ be the rate of firings of actor $A$
  - let $\mu_\alpha$ be the rate of token productions in channel $\alpha$
  - if $A$ produces $k$ tokens in $\alpha$ every time it fires, then
    \[ \mu_\alpha = k \cdot \mu_A \]
Actor Throughput

Let $A$ be an actor in the timed SDFG of interest. Let $\sigma$ be an execution of the timed SDFG: $\sigma$ is a trace in the corresponding LTS.

Define throughput of $A$ in $\sigma$:

$$Th(A, \sigma) \triangleq \lim_{n \to \infty} \frac{\# endA transitions up to first $n$ ticks of $\sigma$}{n}$$

Let $Th(A)$ denote the “best” (maximum) throughput:

$$Th(A) \triangleq \max_{\sigma} Th(A, \sigma)$$

**Theorem ([Ghamarian et al., 2006])**

$$\max_{\sigma} Th(A, \sigma) = Th(A, \text{the ASAP execution})$$

Quiz: Consider an SDFG that deadlocks and let $A$ be an actor in that SDFG.

What is the actor throughput $Th(A)$? Does $Th(A)$ depend on $A$ in this case?
Computing the Actor Throughput

Let $C$ denote the cycle of the ASAP execution lasso of the timed SDFG.

**Theorem ([Ghamarian et al., 2006])**

$$Th(A) = \frac{m_A}{m}$$

where $m_A$ is the number of $\text{end}A$ transitions in $C$ and $m$ is the number of $\text{tick}$ transitions in $C$.

**Example**

![Diagram of the example](attachment:example-diagram.png)

ASAP execution:

$$(g = (0, 1), h = \{\}) \xrightarrow{\text{begin}A} (g = (0, 0), h(A) = \{3\}) \xrightarrow{\text{tick}} (g = (0, 0), h(A) = \{2\}) \xrightarrow{\text{tick} \text{tick}} (g = (0, 0), h(A) = \{0\})\xrightarrow{\text{end}A} (g = (2, 0), h(A) = \{\}, h(B) = \{\})$$

$$\xleftarrow{\text{begin}B} (g = (0, 0), h(A) = \{\}, h(B) = \{0\}) \xrightarrow{\text{tick} \text{tick} \text{tick} \text{tick} \text{tick} \text{tick} \text{tick} \text{tick}} (g = (0, 0), h(A) = \{\}, h(B) = \{7\})$$

$$Th(A) = Th(B) = \frac{1}{10}$$
Could’t we do this more easily?

1. Compute periodic schedule (while checking for deadlocks)
   \[(AB)\omega\]

2. Compute total duration of schedule
   \[ET(A) + ET(B) = 3 + 7 = 10\]

3. Compute throughput
   \[Th(A) = \frac{\# \text{ times } A \text{ appears in schedule}}{10} = \frac{1}{10}\]

Does this method work for all SDFGs?

Parallelism and pipelining!

The simplistic method of the previous slide gives:
\[ET(A) + ET(B) = 9, \text{ therefore, estimates throughput to be } \frac{1}{9}\]

The ASAP execution takes pipelining into account:

and computes the correct (maximal, assuming parallelism) throughput:
\[Th(A) = Th(B) = \frac{2}{9}\]
