Semantics-Preserving Implementation of Synchronous Specifications over Dynamic TDMA Distributed HW (an exercise in architecture abstraction)

D. Potop-Butucaru*
INRIA, France

A. Azim*
University of Waterloo, Canada

S. Fischmeister*

*) Work supported by the French and Canadian taxpayers
Real-Time scheduling

- Functional specification
- Timing model
- HW architecture
- RT implementation
Real-Time scheduling

- Functional specification
- Timing model
- HW model
- RT implementation model
Real-Time scheduling

What is the relation between the high-level model and the low-level reality?

(e.g. Communication costs are often abstracted as 0, whereas on real platforms they are not negligible.)
Architecture abstraction issues

- Functional specification
- Timing model
- HW model
- RT implementation model
- WCET analysis
- Code generation
- Protocols, libraries
- Simple model, Facilitates scheduling
- Abstraction layer
- Low-level models, code
- Low-level HW model
- Low-level timing model
- RT implementation model/code

- Close to the machine. Precise timing information (less variance and/or over-approximations)
Architecture abstraction issues

**Problems:**
- **Efficiency** (Overheads)
- **Correctness**
  * Functional
  * Timing
- **Robustness**
This paper

- Automatic control
- Synchronous dataflow
- Cycle-based execution
- Conditional execution (Scade, Polychrony, Simulink subsets)

**Abstraction:**
- Formal
- Tailored to the framework
- Low-overhead

**SynDEx approach:**
- Static
- Fully automatic
- Distributed

**Network Code toolset:**
- Time-triggered
- Implementation synthesis
- (automatic)

**WCET analysis**

**Code generation**

**Protocols, libraries**

**Abstraction layer**

**Clock drift model**

**Low-level HW model**

**Low-level timing model**

**RT implementation (NC programs)**

**Scheduling table (RT implem. model)**

**Timing (WCET&WCCT)**

**HW (interconnect graph)**

**Functional specification**
This paper

Functional specification

Automatic control
Synchronous dataflow
Cycle-based execution
Conditional execution (Scade, Polychrony, Simulink subsets)

Abstraction:
- Formal
- Tailored to the framework
- « Low-overhead »

Timing (WCET&WCCT)

HW (interconnect graph)

Scheduling table (RT implem. model)

SynDEx approach:
Static
Fully automatic
Distributed

Abstraction layer
Clock drift model

Low-level HW model

Low-level timing model

RT implementation (Network code)

Network Code toolset:
Time-triggered
Implementation synthesis
(automatic)
SynDEx: Functional specification

- Synchronous dataflow (à la Scade, Scicos, Simulink subsets)
SynDEx: HW & Timing model

- Topology
- Bus types
- WCETs, WCCTs
<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>read LP@true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>read FS@true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F1@ LP=false</td>
<td></td>
<td></td>
<td>Send(P1,LP)@true</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>G@</td>
<td></td>
<td>Send(P1,FS)@true</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>LP=false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>N@FS =true</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>Send(P1,ID) @(FS=false ∧ LP=false)</td>
</tr>
<tr>
<td>7</td>
<td>F2@ LP=false</td>
<td></td>
<td></td>
<td>Send(P1,ID) @(FS=false ∧ LP=true)</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>M@FS =false</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>Send(P1,V)@ LP=false</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>F3@</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>LP=false</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Network Code Framework:

- Precision time imperative programming language
- HW platform
- Dynamic TDMA communications
Network Code Framework:

- Precision time imperative programming language
  - Simple instruction set (assembly-like)
    - `wait (duration)`
    - `future (duration,label)`
      - when `duration` lapses, jump to `label`
    - `halt`
    - `if, goto, call, send, receive`
  - No parallelism
  - Formal timed semantics
    - Single time reference

START: wait(1)
L1: if true then
    future (L2,2)
    send (bus_id, sizeof(LP), LP)
    halt ()
endif
wait (16)
goto (START)
L2: if true then
    future (L3,2)
...

```
Network Code Framework:

- HW platform
  - Distributed
  - Node structure
    - Node-wide time reference
  - Automatic synthesis of MAC layer (HW/SW) and runtime
Network Code Framework:

- **Objective:** Dynamic TDMA bus communications
  - No bus contention
  - Data-dependent communications
- **Not provided:**
  - Computation programs
  - Communication programs
  - Clock synchronization (clock drift management)
### SynDEx: Scheduling table

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>read LP @ true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>read FS @ true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F1 @ LP = false</td>
<td></td>
<td></td>
<td>Send(P1,LP) @ true</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>G @ LP = false</td>
<td></td>
<td>Send(P1,FS) @ true</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>N @ FS = true</td>
<td>Send(P1,ID) @ (FS=false ∧ LP=false)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>Send(P1,ID) @ (FS=false ∧ LP=true)</td>
</tr>
<tr>
<td>7</td>
<td>F2 @ LP = false</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td>Send(P1,V) @ LP=false</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Computation program synthesis

P1

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>read LP at true</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>read FS at true</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F1 at LP=false</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>F2 at LP=false</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Absolute dates → durations
Control passing synthesis

START: future (L2,1)
call read_LP
halt
L2: future (L3,1)
call read_LP
halt
L3: if not LP then
    future (L4,3)
call F1
    halt
end
wait (15)
goto START
L4: future (L5,2)
call F2
halt
L5: wait (4)
goto START
Communication program synthesis

```
START: wait (1)
L1: future (L2,2)
    send (LP)
    halt
L2: future (L3,2)
    send (FS)
    halt
L3: if (not LP) and (not FS) then
    future (L5,8)
    send (ID)
    halt
end
wait (1)
L4: if LP and not FS then
    future (START,11)
    wait (5)
    receive (ID)
    halt
end...
```

Bus

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| 1 |   |   | Send(P1,LP)@true |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 2 |   |   | Send(P1,FS)@true |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 3 |   |   | Send(P1,ID) @(FS=false ∧ LP=false) | Send(P1,ID) @(FS=false ∧ LP=true) |   |   |   |   |   |   |    |    |    |    |    |    |
| 4 |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 5 |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 6 |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 7 |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 8 |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 9 |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 10|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 11|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 12|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 13|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 14|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 15|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 16|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
Clock drift management

Challenge:
Preserve the timing guarantees provided by SynDEx.
Preserve correction.

Solution:
Include worst-case overheads in the input WCET, WCCTs
Clock drift management

- Simple assumptions:
  - The cost of local control is negligible (if and goto take no time)
  - The real-time durations of two wait(d) statements differ by less than $\alpha \times d$ for some $\alpha$
  - The real-time duration of a communication can be precisely computed from the size $l$ of the transmitted data, as $\text{comm}(l)$
  - The low-level communication hardware detects and signals the end of send and receive operations
  - The end event of a receive occurs (in real time) after the end event of the send, but no later than $\beta$ time units later.
Clock drift management

- Simple drift management technique
  - Prior to scheduling: Increase each WCET and WCCT in the timing model by $\lceil 2 \times \alpha \times \gamma \rceil$, where $\gamma$ is the longest duration of a bus communication.
  - During code generation: Insert synchronization communications so that the bus cannot be idle for more than $\gamma$ time units. These messages do not change the schedule length.
  - At runtime: At each message reception event, update the local clock to the correct value, which can be computed exactly from the schedule table and the size of the transmitted data.

- Low complexity, but can be largely improved
## Clock drift management

### Example

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>read LP@true</td>
<td></td>
<td></td>
<td>Send(P1,LP)@true</td>
</tr>
<tr>
<td>read FS@true</td>
<td></td>
<td></td>
<td>Send(P1,FS)@true</td>
</tr>
<tr>
<td>F1@LP=false</td>
<td>G@LP=false</td>
<td></td>
<td>Send(P1,ID)@(FS=false ∧ LP=false)</td>
</tr>
<tr>
<td>F2@LP=false</td>
<td>N@FS=true</td>
<td></td>
<td>Send(P1,ID)@(FS=false ∧ LP=true)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sync@FS</td>
<td></td>
</tr>
<tr>
<td>F3@LP=false</td>
<td></td>
<td></td>
<td>Sync@true</td>
</tr>
</tbody>
</table>
Conclusion

• We built a full suite for the model-driven correct-by-construction synthesis of real-time embedded applications, combining:
  - A existing real-time scheduling approach
  - A existing code generation approach
  - A formal architecture abstraction serving as glue
    • Low-overhead (tailored to the existing parts)

• Future:
  - Multi-period implementations of multi-rate specs
  - Refine the timing model of the Network Code with the costs of control
Conclusion (2)

- Architecture abstraction is a fundamental problem in RT scheduling
- It involves modeling, timing analysis, and code generation aspects
- It can and must be done formally
- It can result in significant overheads, if not well done (e.g. independent of the scheduling technique, etc.)
- However, by considering both scheduling model and implementation architecture, costs can be reduced
Related work

- Distributed&RT implementation of conditional dataflow specifications
  - Caspi et al. - Scheduling over TTA
  - Elles et al. - Conditional task graphs
  - Previous SynDEx work
  - Other (OCRep, etc.)

- Distributed communication protocols