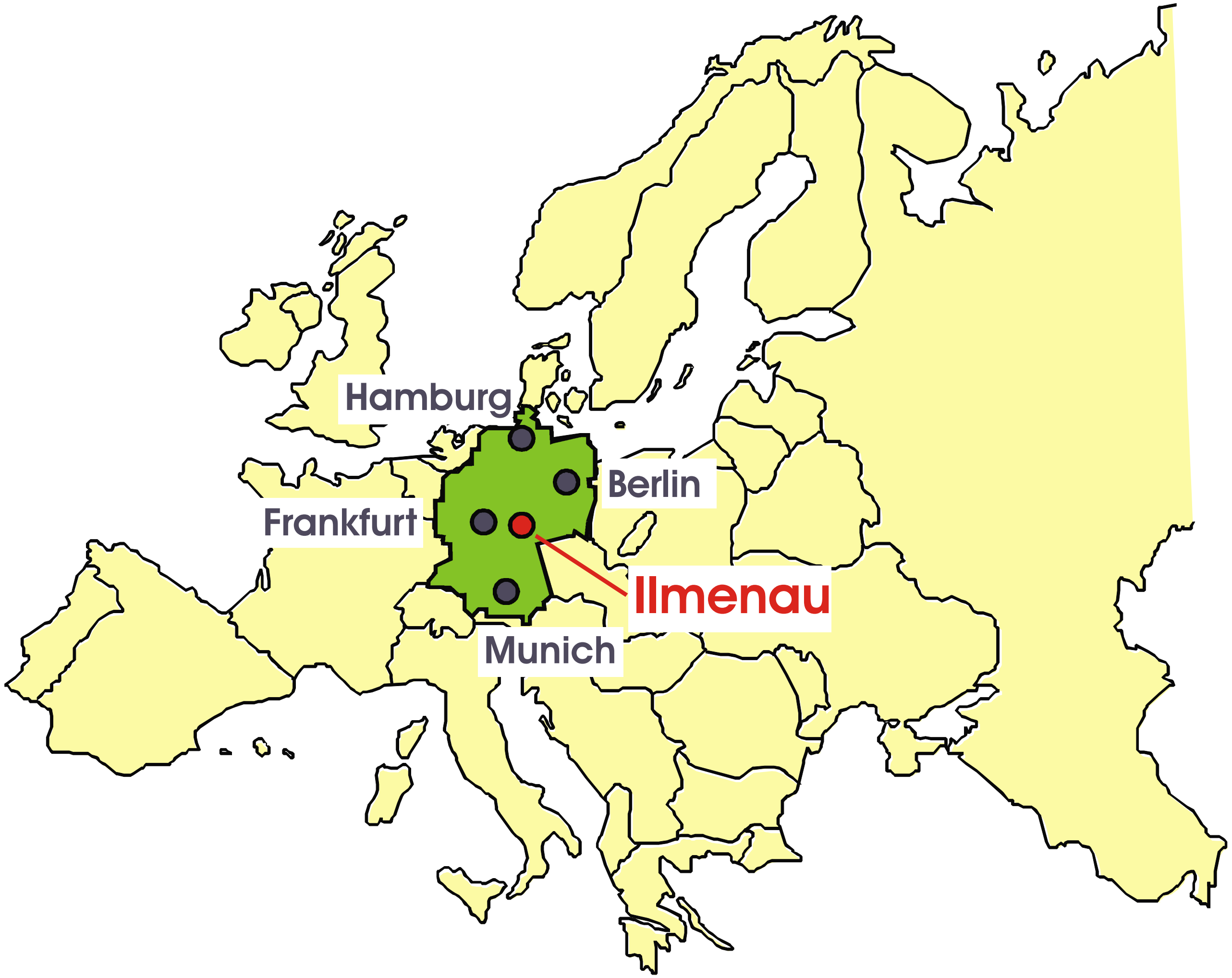
Implementing Mixed Discrete-Continuous Models into Realtime Environments

Wolfgang Fengler, Bernd Däne

Ilmenau Technical University, Germany

Where is Ilmenau?



Topics

1. Introduction
2. Modelling Environment
3. Requirements for Implementation
4. Scheduling Strategies
5. Signal Buffers
6. Delay Times
7. Partitioning the Model
8. Results and Conclusion

Supported by the German Research Council (DFG) under SFB 622.

1. Introduction

Model-Based Design:

- Software design or hardware-software codesign based on formal models
- Checking the model by analysis and simulation
- Avoiding some errors, optimizing the design

- Mixed-domain models desired:
  discrete-event and continuous-time

Implementation Issues

- Limited support for synthesis in many tools
- Conflicting demands of discrete and continuous parts (e.g. scheduling)
- Limited resources in target systems
  (e.g. performance, memory, file system, OS services)

Case study: Finding ways for implementation from a general-purpose modelling tool

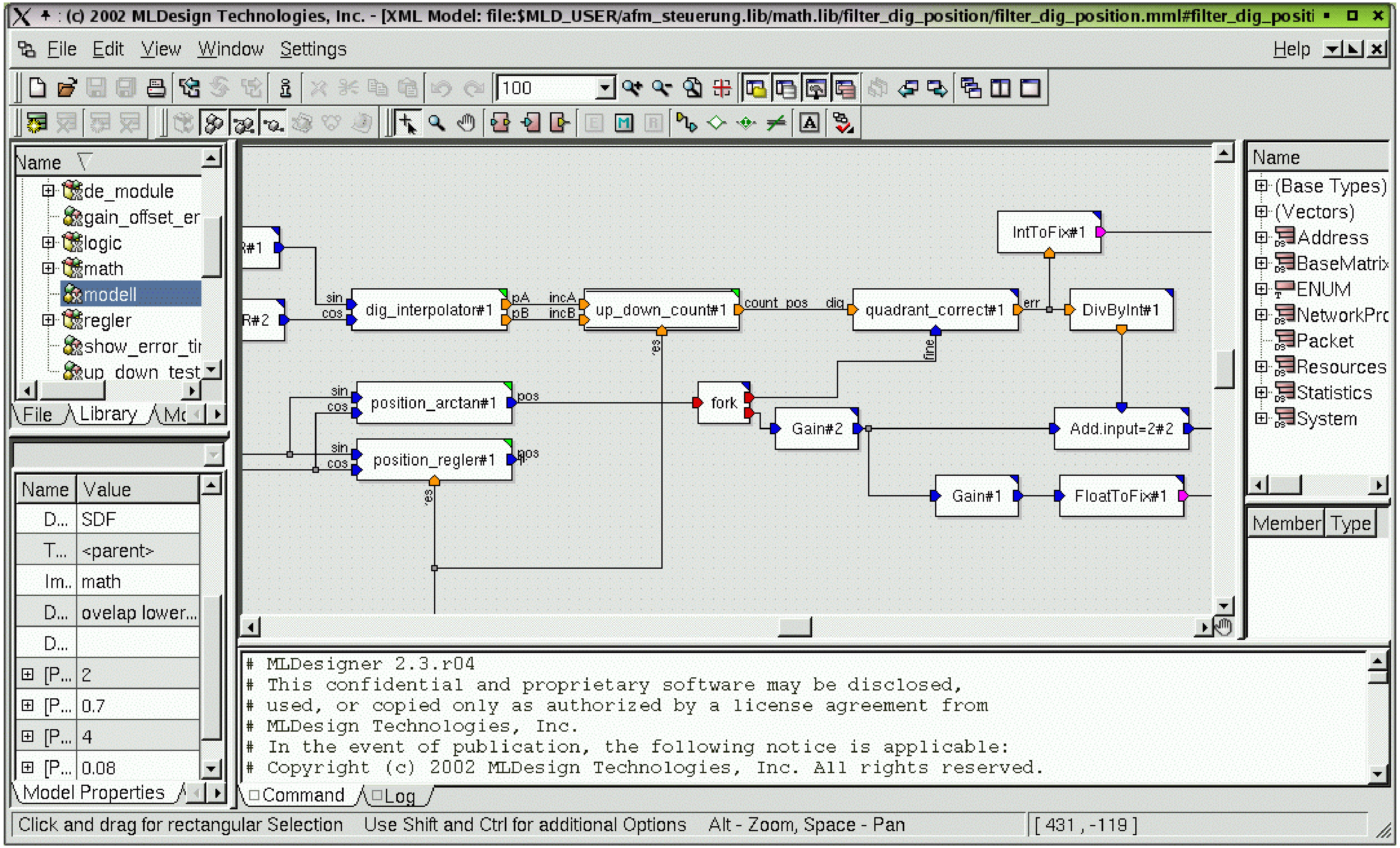
2. Modelling Environment

Modelling tool under consideration:

MLDesigner® from MLDesign Technologies, Inc.

- Hierarchical multi domain modeling framework
- Covers module, system and strategy levels
- Capabilities for simulation, design check, export
- Derived from well-known Ptolemy tool (University of Berkeley)

MLDesigner Sample Workspace



3. Requirements for Implementation

- Handle limited amount of memory
- Deal with real time requirements
  (e.g. response times)
- Adapt to existing runtime environment
  (scheduling, limited OS services)
- Deal with limited language support
  (e.g. C instead of C++)

Initial Restrictions

- Cannot be completely avoided
- Must be chosen carefully

In this study:

- No dynamically sized data
- Two modelling domains only:
  - **DE** (*discrete event*)
  - **SDF** (*synchronous data flow*, kind of continuous time)

4. Scheduling Strategies

- **Single-domain models:**
  Solutions derived from the tool's simulation system and from other sources
- **Mixed-domain models:**
  Solutions discussed for different wormhole hierarchies

Wormhole:

- Concept for nesting a model inside another of different domain
- Boundaries constitute syntactic and semantic interfaces

DE (Discrete Event) Scheduling

- Blocks fire on demand (when events arrive)
- Multitask system with one task per block
- Scheduling in fixed or random order
- Event propagation by message system

Chosen in study: Own round-robin mini scheduler
(for overhead reasons)

SDF (Synchr. Data Flow) Scheduling

- Blocks fire periodically at fixed rates
- Static schedules possible
- Optimize for: code size, data size, time overhead

Chosen in study: Monotonic-rate static scheduling
(available from operating system)

DE Wormholes inside SDF Models

- Periodic events force periodic activation of DE blocks
- DE blocks included in monotonic-rate scheduling
- Time consumption must be certain:
  Exclude uncertainty and infinity by model analysis.

SDF Wormholes inside DE Models

- SDF blocks fire aperiodically (on demand)
- Activation triggers one sequence of the wormhole‘s static schedule
- This sequence appears as just one task
- Periodic activation by explicit clock source only

Deeper Nested Wormholes

- SDF – DE – SDF - ...
  Monotonic-rate behavior is inherited from top level

- DE – SDF – DE - ...
  On-demand behavior is inherited from top level

➔ No extra scheduling policies needed.

5. Signal Buffers

Signal Buffering: At each block‘s inputs

Buffer properties desired:

- Fixed size
- Minimum size
- Static assignment
- Consecutivity

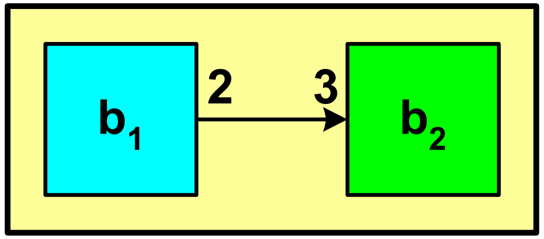
Buffer Sizes

- DE domain:
  Buffer sizes virtually infinite (random schedule)
  - ➔ Limiting size by rule

- SDF domain:
  Buffer sizes fixed (static schedule)
  - ➔ Properties affected by schedule chosen

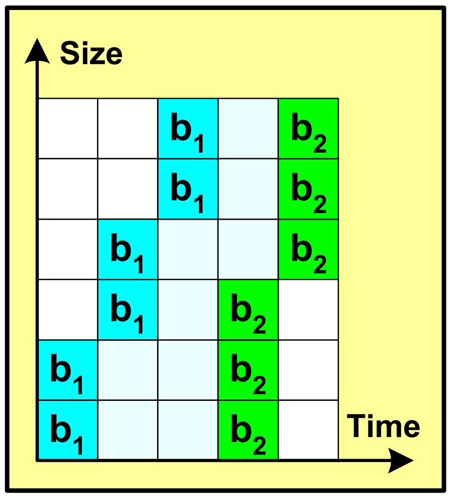
SDF Scheduling Affects Buffers (1)

SDF system:



Schedule **1**: $S_1 = \{b_1, b_1, b_1, b_2, b_2\}$

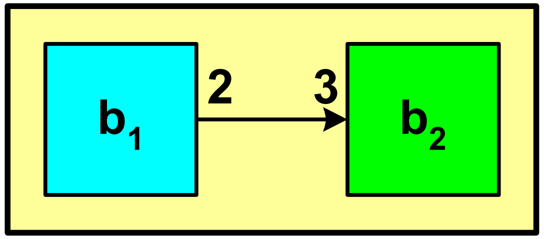
Buffer sequence:



- Size = 6
- Static
- Consecutive

SDF Scheduling Affects Buffers (2)

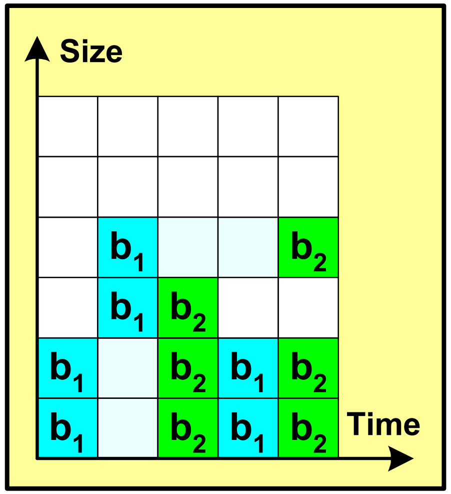
SDF system:



Schedule **2**:

$$S_2 = \{b_1, b_1, b_2, b_1, b_2\}$$

Buffer sequence:



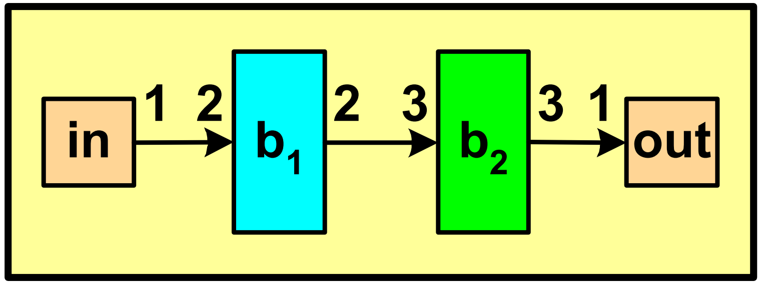
- Size = 4
- Static
- Not consecutive
- *Consecutivity by circular adressing*

6. Delay Times

- Rate monotonic input / output sampling:
  required for digital filters, control loops etc.
- Delay time: response from input to output
  (in SDF systems)
- Very important for closed loop control
  (stability!)
- Affected by schedule chosen

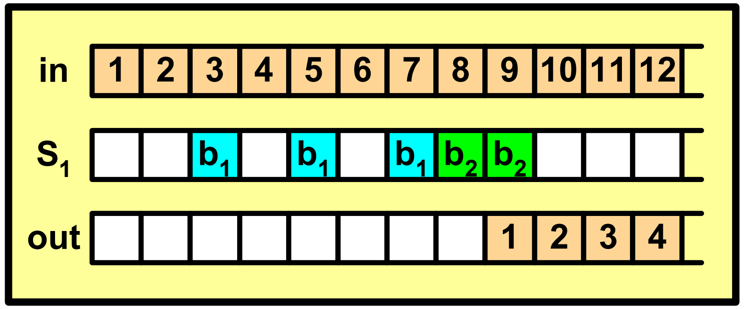
SDF Scheduling Affects Delay (1)

SDF system:



Schedule **1**: $S_1 = \{b_1, b_1, b_1, b_2, b_2\}$

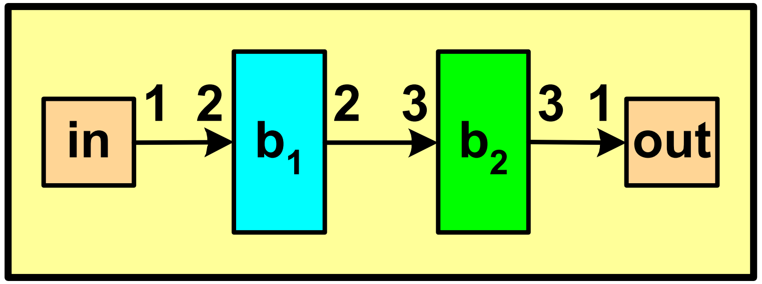
Timing diagram:



Delay = 8

SDF Scheduling Affects Delay (2)

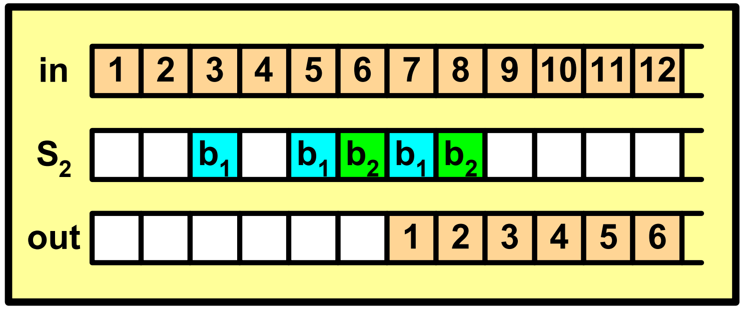
SDF system:



Schedule **2**:

$$S_2 = \{b_1, b_1, b_2, b_1, b_2\}$$

Timing diagram:



Delay = 6

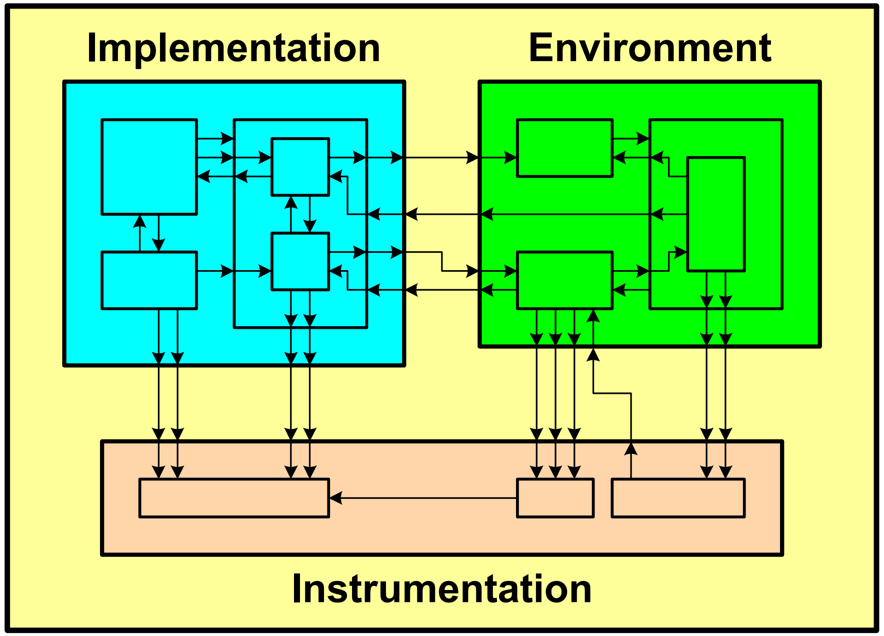
7. Partitioning the Model

Partitions:

- Implementation (Embedded system designed)
- Environment (Embedding process)
- Instrumentation
  (Observation and interaction during simulation)

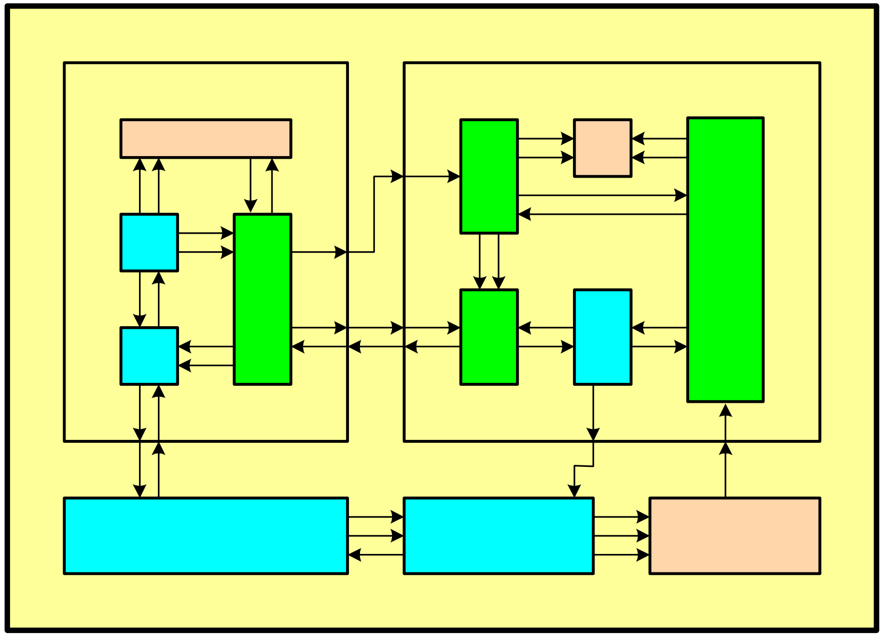
Several partitioning strategies are possible.

One Block per Partition



- Simple notation
- Simple derivation of the implementation‘s interface
- Poor structuring

Individually Flagged Blocks



- Notation requires tool modification
- Better changeable
- Good structuring

8. Results and Conclusion

- Case study: Several tests carried out
- Scheduling and buffer handling demonstrated
- Experimental implementation tool realized

Further work:

- Testing „individually flagged blocks“ approach
- Improvements and generalization of the implementation tool

Contact

Prof. Dr. Wolfgang Fengler
Ilmenau Technical University
Dept. of Computer Architectures

P.O. Box 100565
98684 Ilmenau, Germany

Phone: +49-3677-69-2825

Wolfgang.Fengler@tu-ilmenau.de

http://tin.tu-ilmenau.de/ra/