Modification of CFSMs in Polis to allow Multirate Operations

Fabian Wenzel
David Ruiz Alonso
Contents

• Overview
• Designflow in the Polis environment
• Implementation of Queues
• Example
• Summary, Further Directions
Current Implementation

- CFSM: Model of Computation used by Polis
  - One-place buffers
  - Lossy behavior
  - Suitable for control operation

- Transmission of data requires Protocol
Dataflow using CFSMs

- Multirate operations are hard to implement
- Queues are needed to allow design with less overhead
Our goal: Queues

- Sent data is stored automatically in the Buffer if necessary
- Data loss only occurs if the Queue is full
Possible behavior

• What happens to events in the queue?
  - After considering events being consumed the buffer gets shifted (i.e. used data)
  - At the next “await” statement the buffer is shifted

• Both possibilities might be useful, depending on the application
Polis Designflow

Esterel  \rightarrow  SHIFT

PTL  \rightarrow  Simulation

HW  \rightarrow  Implementation

SW
Parts to change

- *Software* is most important as it is usually slower than Hardware
- *Simulation* is also necessary as design goes hand in hand with simulation
- Ideas can be transferred e.g. to Hardware–Hardware – communication
Components of the Software synthesis consists of:

- OS.c
- fsm1.c
- fsm2.c
- fsmN.c
FSM.c - files

- One single function that is called by the os
- Checks for incoming events
- Executes a transition
- Emit output events
- Variable v_st determines the state the FSM is currently in
Modified FSM.c

- Checks for incoming events
- Executes a transition
- Emit output events
- If data has been consumed tell the os to check the queue, otherwise keep data
os.c – the RTOS core

- Scheduler successively executes modules when new inputs for them are available
- Declares “emit” – functions for each output that delivers events
- Contains data structures for communication:
  - Input places for each module
  - Memory for valued events
Modified os.c

- Emit – function now takes care of buffers:
  - If the receiver has no data, just follow the normal behavior
  - If data is already waiting at the receiver, place the new one in the queue

- Check–Buffer:
  - After a transition, this function checks the queues and delivers an event if available

- Update–value:
  - Update data to be used by a module
Queues

Os.c

Buffer

Sender

Receiver
Implementation in Ptolemy

Components of the Ptolemy implementation:

FSM1.pl
FSM2.pl
FSM1.cc
FSM2.cc

Generated out of SHIFT by Polis
Generated by Ptolemy
FSM.c – files

- FSM files are the same ones synthesized by polis but with timing information added

   → Changes are the same
FSM.pl – files

- Encapsulate the module from the main Ptolemy scheduler
- Takes care of the timing behavior of the module
- Contain information for the UI
- Can refire itself
Modified FSM.pl – files

- The events which are delivered from the main scheduler are passed to the transition function if the queue is empty
- Otherwise events are stored in the queue
- After a transition, if data has been consumed it checks the queue and delivers a new event if possible
- Buffer–size is included as a parameter to enable changes on the fly, inside Ptolemy environment
What did we show so far?

- Implementation of queues in both the Software and the simulation environment
- Behavior of the queues: After a successful transition the contents get shifted

- Problem: If the relevant value is accessed after the transition, the old value is lost.
Keeping old values in FSMs

- Values are possibly to remain until the next buffer access
- This fact can only be determined in the transition function!
- After each transition, buffers are shifted depending on the current state, not only on the fact that the value has been accessed
Summary

- Two possible implementations of buffers have been shown
- “Simple” problem of buffer implementation affects several and different components of the Polis – Design environment
- Future directions:
  - Automatic implementation of buffers by modifying compilers
  - Buffer status can easily be used to add multirate behavior