Scheduling and Code Generation in CoCentric System Studio

Joseph T. Buck
Advanced Technology Group
Synopsys, Inc.

CoCentric System Studio (marketing)

- Fast
  - system simulation
- Easy
  - design management
- Complete
  - functional modeling
Combining Dataflow and Control

- System Studio enables design of arbitrarily nested data flow and control at a high abstraction level
  - Data flow is expressed as data flow graph (DFG)
  - Control is expressed in terms of extended finite state machines (+ hierarchy, concurrency, gating)
  - Control model can be instantiated in a DFG
  - Data flow model can be controlled by an FSM

A Simplified Example from Mobile Communications

- Speech Codec
- TDMA Framer
- Modulator
- Channel
- Correlator
- Equalizer
- Data Reframe
- Speech Decoder
- Energy Detector
- Sync Burst Detect
- FSM (finite state machine)
Related work

- Fully heterogeneous solutions
  - Ptolemy Classic [Buck, Ha, Lee, Messerschmitt ’94]
  - Ptolemy *-charts [Girault, Lee, Lee ’98]
- “Structured” (limited) heterogeneity approaches
  - SDL
  - Polis (aka Felix, VCC)
- Hierarchical control
  - Esterel [G. Berry]
  - Harel’s Statecharts + variants (Argos, SyncCharts)
  - ECL [Lavango, Sentovich 99]
- Dataflow (Ptolemy, COSSAP, Grape)

System Studio Models

- Hierarchical models
  - Data flow graph (DFG)
  - AND, OR, and GATED models
- Leaf level (atomic) models
  - Primitive blocks – prim_model or COSSAP primitive (SDS)
  - atomic state
- All models have:
  - Parameters
  - Ports
  - Type parameters (for generic models)
The control/data flow interface

- What happens when a data flow graph is inside control?
  - Each port of the DFG is bound to a port or signal of the containing control model
  - If binding is to a signal, one value is passed from signal to DFG or vice versa
  - If binding is to a port, a user-specified expression determines the number of values.
### Prim_model with static I/O

```plaintext
class Reverse {
  type_param T = double;
  param read_on_reset int BlockSize = 4;
  port in  T Inp;
  port out T OutP;

  main_action {
    int j;
    T buffer[BlockSize];
    for (j = 0; j < BlockSize; j++) {
      read (Inp);
      buffer[j] = Inp;
    }
    for (j = Blocksize-1; j >= 0; j--) {
      Outp = buffer[j];
      write (Outp);
    }
  }
}
```

- **Data type** is specified by parameter `T`.
- **I/O pattern** is static (data independent).

### Prim_model with dynamic I/O

```plaintext
class select {
  type_param T = double;
  port in  T InTrue, InFalse;
  port out T OutP;
  port in bool control;

  main_action {
    read (control);
    if (control) {
      read (InTrue);
      Outp = InTrue;
    } else {
      read (InFalse);
      Outp = InFalse;
    }
    write (Outp);
  }
}
```

- **Input pattern** is dynamic (data dependent).
Fast compiled simulation

- Two code generators:
  - action code (prim_model bodies, or-state actions) processed as internal trees, generated as C
  - control structures converted to Esterel intermediate code (“ic”)
    - S. Edwards’ Esterel compiler (CODES ‘99) generates control skeleton
- COSSAP stream-driven simulator handles dynamic dataflow
- Slavable simulations can be produced

Tree structures in CCSS

- Transparent modeling: model -> tree
- Trees used for:
  - types (may have parameters)
  - expressions (side-effect free)
  - actions (AST’s for user C/C++ code)
  - control trees
    - Esterel-like abstract control representation
    - Esterel rules for values vs signals relaxed
    - Leaves of control trees are expressions and actions
    - Each tree type has a C++ class
Control model translation

- Approach resembles Andre’s SyncCharts
- Models converted to “control tree”
  - And-models -> Esterel parallel
  - Gated models -> Esterel suspend
  - Or-models are more complex
  - Leaves of control trees are generated as separate functions.

Or-models to Esterel

```plaintext
signal enter_A, enter_B in
  emit enter_A;
  loop
    present enter_A then
      weak abort
      run A
      case c do
        emit out(1); emit enter_B
        end weak abort
      else pause
      end present
    end loop
  end loop; end signal
```

A

B

C/out=1

C/out=0
Dataflow code generation

- Cyclostatic dataflow [Bilsen et al 1995]
- Our formulation: *phase ranges*
  - Rates may be symbolic even at run time

```c
main_action {
    int j;
    T buffer[BlockSize];
    for (j = 0; j < BlockSize; j++) {
        read (Inp);
        buffer[j] = Inp;
    }

    for (j = BlockSize-1; j >= 0; j--) {
        Outp = buffer[j];
        write (Outp);
    }
}
```

Delays and CSDF

- SDF requires a separate mechanism (initial delays) to allow for recurrences
- CSDF allows an alternative
- For multiple delays, specifying I/O order is overconstraining
- CCSS also has a built-in delay block

```c
Prim_model Delay1 {
    type_param T = double;
    port in T Inp;
    port out T Outp;
    param read_on_reset T Init = 0;
    T state = Init;
    main_action {
        Outp = state;
        write (Outp);
        read (Inp);
        state = Inp;
    }
}
```
Single appearance schedules

- For SDF, we have a single appearance schedule unless we have a tightly interdependent SCC (Bhattacharrya)

Loose dependence and CSDF

- Compute repetition vector for an SCC
  - If q=1, and inputs not needed to produce all outputs, we have loose interdependence
  - Input to C below is a loose dependence
  - C must be flagged as a “split node”
Dataflow scheduling

- Scheduling algorithm alternates merge and loop transformations
  - Merge pass: combine instances with matching rates
    - CSDF allows phases to interleave: less memory required than for SDF
  - Loop pass: alter instances to match rates of neighbors
    - Transformations: stall, sum-up, do-while, SDF loop
  - Symbolic rates can be handled
  - Reduction to one prim_model -> static

Dynamic merging

- If a prim_model has only one I/O call to a port, it can sometimes be merged with a neighbor
- The merged cluster may then be static.

```cpp
Pr{im_model Delete {  
  type_param T = double;  
  port in T Inp;  
  port out T Outp;  
  param read_on_reset int Ndel;  
  int count = 0;  
  main_action {  
    read (Inp);  
    if (count < Ndel)  
      count++;  
    else {  
      Outp = Inp;  
      write (Outp);  
    }  
  }  }
```

Src can replace this read
Sink can replace this write
Cross-domain optimization

- Or-models, gated models can have “dataflow form” if simple enough
  - Model is converted to equivalent prim_model
- Likewise, a static prim_model can have a control form (a simple loop)
- Otherwise a separate controller is generated for control-in-dataflow

Status

- Product is in general release
- Many reference design kits available for standards (CDMA-2000, Bluetooth, MPEG, GSM etc)
- Future directions ...