EECS 222: Embedded System Modeling Lecture 16

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 16: Overview

- Course Administration
  - Instructor evaluation
  - Final exam
- Project Discussion
  - Pipelining and parallelization of the Canny Edge Detector
  - Status and next steps
- Project Assignment 9
  - Software and hardware optimizations
- Modeling of Hardware in Embedded Systems
  - Register Transfer Level (RTL) abstraction
  - Accellera RTL Semantics
  - RTL description and modeling in SpecC

Course Administration

- Final Course Evaluation
  - Open now until Sunday night of 10th week
  - Feb. 24, 2020, through March 15, 2020, 11:45pm
  - Online via EEE evaluation application
- Evaluation of Course and Instructor
  - Voluntary
  - Anonymous
  - Very valuable!

➢ Please help to improve this class!

  - Please spend 5 minutes!

Course Administration

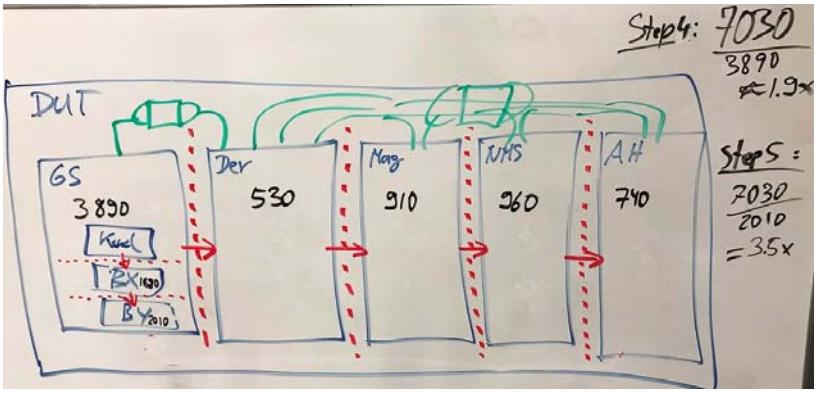
- Final Exam
  - Allocated time
    - Thursday, March 19, 2020, 8:00-10:00am
  - Location
    - Regular classroom, MSTB 120
  - Format: Written Exam
    - Exam sheet with questions
    - Answers to be filled in
    - Open notes, open course materials
    - Open laptop, open browser, open server login
    - No emails, no instant messaging!

Project Assignment 8

- Task: Pipelining and Parallelization of the Canny Model
  - Pipeline and parallelize the model to maximize throughput
- Steps
  1. Instrument model with logging of simulated time and frame delay
  2. Back-annotate estimated timing in DUT components
  3. Instrument model with logging of throughput (FPS)
  4. Pipeline the DUT into stages for each component
  5. Integrate Gaussian Smooth components into pipeline stages
  6. Slice the BlurX and BlurY blocks into parallel components
- Deliverables
  - **`canny.sc`** or **`canny.cpp`** (choose one!)
  - **`canny.txt`** (with observed timing and frame delays)
- Due: February 26, 2020, 6pm

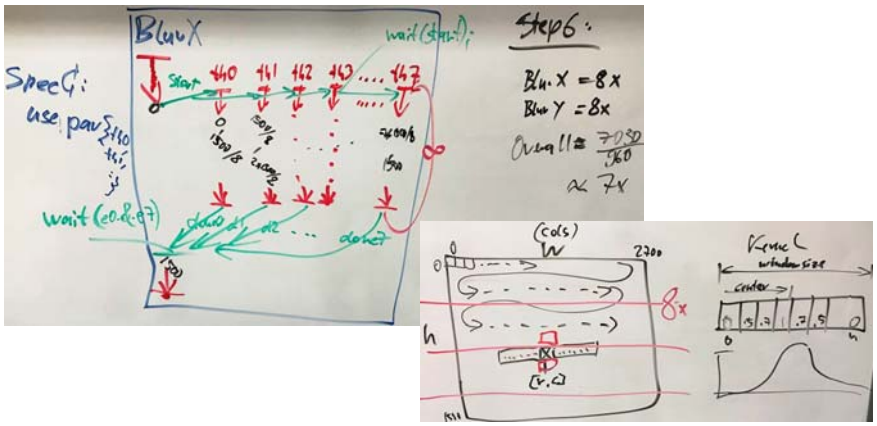
Project Assignment 8

- Step 4: Pipeline the DUT into stages
- Step 5: Integrate Gaussian Smooth into pipeline stages
  - Discussion on whiteboard: Chart of refined DUT structure



Project Assignment 8

- Step 6: Slice the BlurX and BlurY blocks into parallel components
  - Discussion on white board



Project Assignment 8

- Deliverable
  - Observed timing results after each refinement step:

Model	Frame Delay	Throughput	Total time
CannyA8_step1	... ms		... ms
CannyA8_step2	... ms		... ms
CannyA8_step3	... ms	... FPS	... ms
CannyA8_step4	... ms	... FPS	... ms
CannyA8_step5	... ms	... FPS	... ms
CannyA8_step6	... ms	... FPS	... ms

Project Assignment 8

- Deliverable
  - Timing observed after each step: SpecC models

Model	Frame Delay	Throughput	Total time
CannyA8_step1	0 ms	n/a	0 ms
CannyA8_step2	28120 ms	n/a	140600 ms
CannyA8_step3	28120 ms	0.142 FPS	140600 ms
CannyA8_step4	27970 ms	0.257 FPS	90210 ms
CannyA8_step5	18830 ms	0.498 FPS	48850 ms
CannyA8_step6	9380 ms	1.042 FPS	21866 ms

  - Timing observed after each step: SystemC models

Model	Frame Delay	Throughput	Total time
CannyA8_step1	0 ms	n/a	0 ms
CannyA8_step2	17340 ms	n/a	45220 ms
CannyA8_step3	17340 ms	0.498 FPS	45220 ms
CannyA8_step4	17340 ms	0.498 FPS	45220 ms
CannyA8_step5	18900 ms	0.498 FPS	45220 ms
CannyA8_step6	12260 ms	1.042 FPS	21866 ms

Project Discussion

- Discussion Questions
  - Does the timing meet our real-time goals? No.
  - How far off is it? 7.030/0.0333 = 211x
  - What can be done to improve the speed?
    - Pipelining A8, steps 4 and 5
    - Parallelization A8, step 6
    - Hardware optimizations A9, step 3
    - Software optimizations A9, steps 1, 2, and 4
    - Application adjustments Discussion, future work

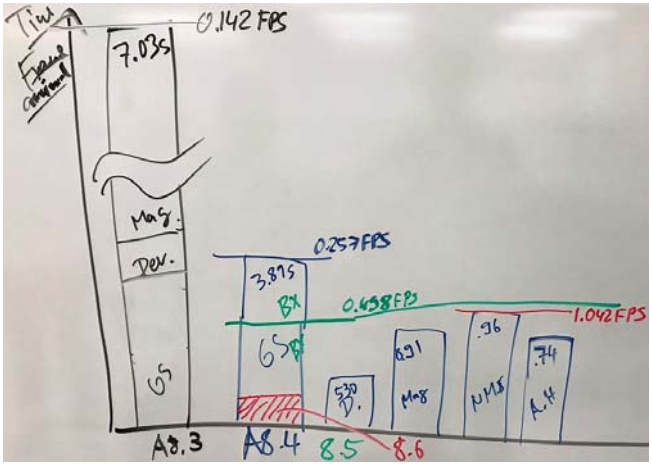
Project Discussion

- Performance Estimation on **Prototyping** Platform
  - Measured timing on Raspberry Pi 3 board:
    ARM-based quad-core processor (1.2GHz)

`Receive_Image`	`0 ms per frame`
`Make_Kernel`	`0 ms per frame`
`BlurX`	`1880 ms per frame`
`BlurY`	`2010 ms per frame`
`Derivative_X_Y`	`530 ms per frame`
`Magnitude_X_Y`	`910 ms per frame`
`Non_Max_Supp`	`960 ms per frame`
`Apply_Hysteresis`	`740 ms per frame`
`Total`	`7030 ms per frame`

Project Discussion

- Model Performance Overview
  - Discussion on the whiteboard



Project Assignment 9

- Task: Throughput optimization of Canny Edge Decoder
  - Apply software optimizations
  - Apply platform optimization
- Steps
  1. Turn on compiler optimizations, measure speedup per block
  2. Apply speedup to back-annotated timing (overall 2.5x)
  3. Replace Raspberry Pi 3 with new Raspberry Pi 4 platform
  4. Replace floating-point with fixed-point arithmetic in NMS block and observe speed-vs.-quality trade-off
- Deliverables
  - **`canny.sc`** or **`canny.cpp`** (choose one!)
  - **`canny.txt`** (with observed throughput and frame delays)
- Due: March 4, 2020, 6pm

Project Assignment 9

- Deliverables
  - Speed-up values observed for each block:

```
T1  =  ...ms / ...ms = ...
T2  =  ...ms / ...ms = ...
T3  =  ...ms / ...ms = ...
T4  =  ...ms / ...ms = ...
T5  =  ...ms / ...ms = ...
T6  =  ...ms / ...ms = ...
T7  =  ...ms / ...ms = ...
Tot =  ...ms / ...ms = ...
```

  - Timing observed after each step:

Model	Frame Delay	Throughput	Total time
CannyA9_step2	... ms	... FPS	... ms
CannyA9_step3	... ms	... FPS	... ms
CannyA9_step4	... ms	... FPS	... ms

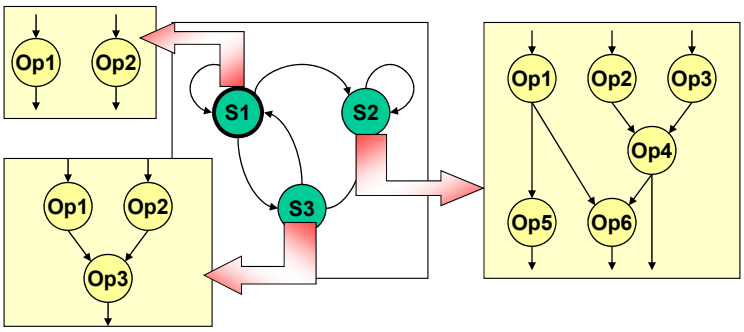
Project Assignment 9

- Performance Estimation on new **Prototyping** Platform
  - Measured timing on Raspberry Pi 4 board:
    ARM-based quad-core processor (1.5GHz)

`Receive_Image`	`0 ms per frame`
`Make_Kernel`	`0 ms per frame`
`BlurX`	`440 ms per frame`
`BlurY`	`625 ms per frame`
`Derivative_X_Y`	`260 ms per frame`
`Magnitude_X_Y`	`170 ms per frame`
`Non_Max_Supp`	`320 ms per frame`
`Apply_Hysteresis`	`295 ms per frame`
`Total`	`2110 ms per frame`

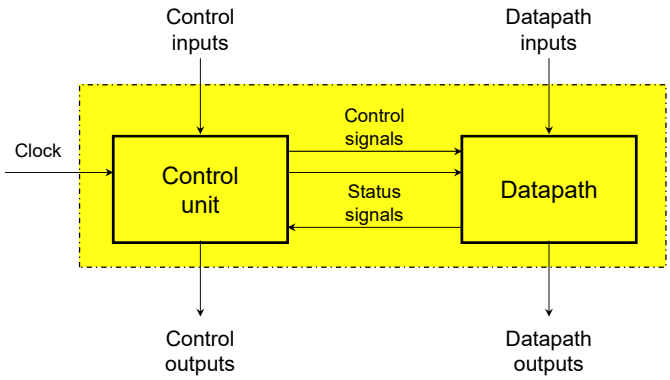
Hardware Modeling in Embedded Systems

- Register Transfer Level (RTL) Abstraction
  - FSMD Model: Finite State Machine with Data
    - Combined model for control and computation
    - Implementation: controller plus datapath



Hardware Modeling in Embedded Systems

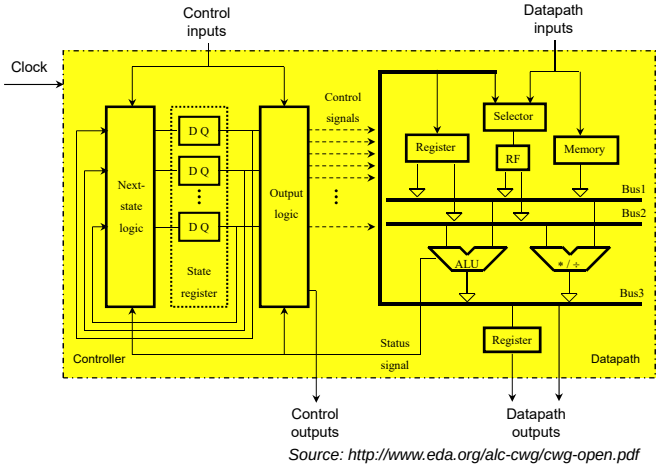
- Register Transfer Level (RTL) Abstraction
  - Block diagram of generic RTL component (high level)



Source: http://www.eda.org/alc-cwg/cwg-open.pdf

Hardware Modeling in Embedded Systems

- Register Transfer Level (RTL) Abstraction
  - Block diagram of generic RTL component (low level)



Source: http://www.eda.org/alc-cwg/cwg-open.pdf

Hardware Modeling in Embedded Systems

- Register Transfer Level (RTL) Synthesis
  - Allocation
    - Type and number of functional units
    - Type and number of storage units
    - Type and number of interconnecting busses
  - Scheduling
    - Representation of basic blocks as super-states
    - Scheduling of operations to clock cycles
  - Binding
    - Bind functional operations to functional units
    - Bind variables to storage units
    - Bind assignments/transfers to busses
  - Result:
    - Clock-cycle accurate structural model at RTL abstraction

Hardware Modeling in Embedded Systems

- System-level Modeling of RTL components
  - Five styles of modeling states: *Accellera RTL Semantics*
    - Style 1: *unmapped*
      - `a = b * c;`
    - Style 2: *storage mapped*
      - `R1 = R1 * RF2[4];`
    - Style 3: *function mapped*
      - `R1 = ALU1(MULT, R1, RF2[4]);`
    - Style 4: *connection mapped*
      - `Bus1 = R1;`
      - `Bus2 = RF2[4];`
      - `Bus3 = ALU1(MULT, Bus1, Bus2);`
    - Style 5: *exposed control*
      - `ALU_CTRL = 011001b;`
      - `RF2_CTRL = 010b;`
      - `…`

Source: http://www.eda.org/alc-cwg/cwg-open.pdf

The SpecC Language

- Register Transfer Level (RTL) Modeling
  - Types specific to RTL
    - **`bit[l:r]`** Two-value logic vector of arbitrary length
    - **`bit4[l:r]`** Four-value logic vector of arbitrary length
  - Type modifiers specific to RTL
    - **`resolved`** Four-value logic resolution
    - **`buffered`** Storage
    - **`signal`** Communication
  - Control flow specific to RTL
    - **`fsmd`** Explicit finite state machine with datapath

The SpecC Language

- Bit vector type: **`bit`**
  - signed or unsigned
  - arbitrary length
  - standard operators
    - logical operations
    - arithmetic operations
    - comparison operations
    - type conversion
    - type promotion
  - concatenation operator
    - `a @ b`
  - slice operator
    - `a[l:r]`

```
typedef bit[7:0] byte;  // type definition
byte             a;
unsigned bit[16] b;

bit[31:0] BitMagic(bit[4] c, bit[32] d)
{
 bit[31:0] r;

 a = 11001100b;          // constant
 b = 1111000011110000ub; // assignment

 b[7:0] = a;             // sliced access
 b = d[31:16];

 if (b[15])              // single bit
    b[15] = 0b;          // access

 r = a @ d[11:0] @ c     // concatenation
     @ 11110000b;

 a = ~(a & 11110000b);   // logical op.
 r += 42 + 3*a;          // arithmetic op.

 return r;
}
```

The SpecC Language

- Four-value bit vector type: **`bit4`** (extension in 2005)
  - 4-value logic
    - 1 (**`0q1`**) high
    - 0 (**`0q0`**) low
    - x (**`0qx`**) unknown (result of operation on 'x' or 'z')
    - z (**`0qz`**) high impedance (no driver)
  - signed or unsigned
  - arbitrary length
  - standard operators
    - same as for regular bit vectors
  - resolution function
    - type modifier **`resolved`**
    - ➢ resolves multiple drivers, e.g for bus wires

```
bit4[31:0] BitMagic(bit4[8] a, bit4[32] b)
{
 resolved bit4[31:0] r;

 a = 0q11001100;
 b = 0q11110000zzzzxxxx;
 r = a @ b[31:24] @ 0xq0011001100001111;

 return r;
}
```


The SpecC Language

- **`buffered`** type modifier
  - Representation of storage in RTL models
    - register
    - register file
    - memory
  - Update at notification of specified events
    - synchronized with explicit clock

```
event Clk1, Clk2;                // system clock
buffered[Clk1] bit[32] R1;       // register
buffered[Clk1] bit[32] R2;

buffered[CLK2] bit[16] RF[64];   // register file
buffered[CLK2] bit[ 8] M1[1024]; // memory

R1 = R2;      // swap contents of R1 and R2
R2 = R1;
wait CLK1;

RF[2] = RF[0] + RF[1];
...
wait CLK2;
```

The SpecC Language

- **`signal`** type modifier
  - Representation of wires and busses in RTL models
  - Semantics as in VHDL, Verilog

```
signal bit[31:0] addr;   // address bus
signal bit[31:0] data;   // data bus
buffered[CLK] M[1024];

wait addr;               // memory read access
data = M[addr];
...

wait addr && data;
M[addr] = data;          // memory write access
...
```

  - Implemented as buffered variables with associated event

```
signal int x;     ⇔  buffered int x_v; event x_e;
x = 55;           ⇔  x_v = 55; notify x_e;
y = x + 2;        ⇔  y = x_v + 2;
wait x;           ⇔  wait x_e;
notify x;         ⇔  notify x_e;
```


The SpecC Language

- RTL Control Flow
  - **`fsmd`** construct
    - Similar to **`fsm`** construct, but specifically for RTL
    - Explicit states and state transitions
    - State actions represent well-defined register transfers
      - limited to conditional/unconditional assignments and function calls
      - general loops, exceptions, synchronization, timing are not allowed
    - Explicit clock specifier
      - event list (external clock)
      - time delay (internal clock)
    - Explicit sensitivity list
      - needed for Mealy machine support
    - Explicit reset state
      - synchronous reset
      - asynchronous reset

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity
        {
          bit[32] a, b, c, d, e;            // local variables

                      { Outport = 0;        // default
                        Done = 0b;          // assignments
                       }
          if (RST)    { goto S0;            // reset actions
                       }
          S0 :        { if (Start) goto S1;
                        else       goto S0;
                       }
          S1 :        { a = b + c;          // state actions
                        d = Inport * e;     // (register transfers)
                        Outport = a;
                        goto S2;
                       }
          ... }
    }
};
```


The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity
        {
          bit[32] a, b, c, d, e;            // local variables

                      { Outport = 0;        // default
                        Done = 0b;          // assignments
                       }
          if (RST)    { goto S0;            // reset actions
                       }
          S0 :        { if (Start) goto S1;
                        else       goto S0;
                       }
          S1 :        { a = b + c;          // state actions
                        d = Inport * e;     // (register transfers)
                        Outport = a;
                        goto S2;
                       }
          ... }
    }
};
```

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK; Inport, Start)               // clock + sensitivity
       {
         bit[32] a, b, c, d, e;             // local variables

                    { Outport = 0;          // default
                      Done = 0b;            // assignments
                    }
         if (RST)   { goto S0;              // reset actions
                    }
         S0 :       { if (Start) goto S1;
                      else       goto S0;
                    }
         S1 :       { a = b + c;           // state actions
                      d = Inport * e;       // (register transfers)
                      Outport = a;
                      goto S2;
                    }
         ... }
   }
};
```


The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK; RST)                         // asynchronous reset
       {
         bit[32] a, b, c, d, e;             // local variables

                    { Outport = 0;          // default
                      Done = 0b;            // assignments
                    }
         if (RST)   { goto S0;              // reset actions
                    }
         S0 :       { if (Start) goto S1;
                      else       goto S0;
                    }
         S1 :       { a = b + c;           // state actions
                      d = Inport * e;       // (register transfers)
                      Outport = a;
                      goto S2;
                    }
         ... }
   }
};
```

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity
        {
          bit[32] a, b, c, d, e;            // local variables

                     { Outport = 0;         // default
                       Done = 0b;           // assignments
                      }
          if (RST)   { goto S0;             // reset actions
                      }
          S0 :       { if (Start) goto S1;
                       else       goto S0;
                      }
          S1 :       { a = b + c;          // state actions
                       d = Inport * e;      // (register transfers)
                       Outport = a;
                       goto S2;
                      }
          ... }
    }
};
```


The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity
        {
          bit[32] a, b, c, d, e;            // local variables

                     { Outport = 0;         // default
                       Done = 0b;           // assignments
                      }
          if (RST)   { goto S0;             // reset actions
                      }
          S0 :       { if (Start) goto S1;
                       else       goto S0;
                      }
          S1 :       { a = b + c;          // state actions
                       d = Inport * e;      // (register transfers)
                       Outport = a;
                       goto S2;
                      }
          ... }
    }
};
```

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity
        {
          bit[32] a, b, c, d, e;            // local variables

                     { Outport = 0;         // default
                       Done = 0b;           // assignments
                      }
          if (RST)   { goto S0;             // reset actions
                      }
          S0 :       { if (Start) goto S1;
                       else       goto S0;
                      }
          S1 :       { a = b + c;          // state actions
                       d = Inport * e;     // (register transfers)
                       Outport = a;
                       goto S2;
                      }
          ... }
    }
};
```


The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity
        {
          bit[32] a, b, c, d, e;            // unmapped variables

                     { Outport = 0;         // default
                       Done = 0b;           // assignments
                      }
          if (RST)   { goto S0;             // reset actions
                      }
          S0 :       { if (Start) goto S1;
                       else       goto S0;
                      }
          S1 :       { a = b + c;          // Accellera style 1
                       d = Inport * e;     // (unmapped)
                       Outport = a;
                       goto S2;
                      }
          ... }
    }
};
```

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,         // system clock
        signal in  bool       RST,         // system reset
        signal in  bit[31:0]  Inport,      // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,     // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                             // clock + sensitivity
        {
         buffered[CLK] bit[32] RF[4];      // register file

                      { Outport = 0;       // default
                        Done = 0b;         // assignments
                       }
          if (RST)    { goto S0;           // reset actions
                       }
          S0 :        { if (Start) goto S1;
                        else       goto S0;
                       }
          S1 :        { RF[0]=RF[1]+RF[2]; // Accellera style 2
                        RF[3]=Inport*RF[4];// (storage mapped)
                        Outport = RF[0];
                        goto S2;
                       }
          ... }
    }
};
```


The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,         // system clock
        signal in  bool       RST,         // system reset
        signal in  bit[31:0]  Inport,      // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,     // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                             // clock + sensitivity
        {
         buffered[CLK] bit[32] RF[4];      // register file

                      { Outport = 0;       // default
                        Done = 0b;         // assignments
                       }
          if (RST)    { goto S0;           // reset actions
                       }
          S0 :        { if (Start) goto S1;
                        else       goto S0;
                       }
          S1 :        { RF[0] =            // Accellera style 3
                          ADD0(RF[1],RF[2]);// (function mapped)
                        RF[3] =
                          MUL0(Inport,RF[4]);
                        Outport = RF[0];
                        goto S2;
                       }
    }
};
```

The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,         // system clock
        signal in  bool       RST,         // system reset
        signal in  bit[31:0]  Inport,      // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,     // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                             // clock + sensitivity
        {
          buffered[CLK] bit[32] RF[4];     // register file
          bit[32] BUS0, BUS1, BUS2;        // busses
                        { Outport = 0;     // default
                          Done = 0b;       // assignments
                        }
          if (RST)      { goto S0;         // reset actions
                        }
          S0 :          { if (Start) goto S1;
                          else       goto S0;
                        }
          S1 :          { BUS0 = RF[1];    // Accellera style 4
                          BUS1 = RF[2];    // (connection mapped)
                          BUS3 = ADD0(BUS0,BUS1);
                          RF[0]= BUS3;
                          ...
                          goto S2;
                        }
    }
};
```


The SpecC Language

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,         // system clock
        signal in  bool       RST,         // system reset
        signal in  bit[31:0]  Inport,      // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,     // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                             // clock + sensitivity
        {
          signal bit[5:0] RF_CTRL;         // control wires
          signal bit[1:0] ADD0_CTRL, MUL0_CTRL;
                        { Outport = 0;     // default
                          Done = 0b;       // assignments
                        }
          if (RST)      { goto S0;         // reset actions
                        }
          S0 :          { if (Start) goto S1;
                          else       goto S0;
                        }
          S1 :          { RF_CTRL = 011000b; // Accellera style 5
                          ADD0_CTRL = 01b;   // (exposed control)
                          MUL0_CTRL = 11b;
                          ...

                          goto S2;
                        }
    }
};
```