EECS 222: Embedded System Modeling Lecture 18

Rainer Dömer

doemer@uci.edu

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

Lecture 18: Overview

- Course Administration
  - Instructor evaluation
  - Final exam
- Project Assignment 8
  - Pipelining and parallelization of the Canny Edge Detector
- Project Discussion
  - Status and next steps
- Project Assignment 9
  - Compiler optimizations
  - Application optimizations
- Modeling of Hardware in Embedded Systems
  - RTL component modeling in SystemC

Course Administration

- Final Course Evaluation
  - 9th through 10th week
  - Feb. 25, 2019, through March 17, 2019, 11:45pm
  - Open until next Sunday night
  - 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 21, 2019, 8:00-10:00am
  - Location
    - Regular classroom, SSTR 103
  - 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 simulation 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: Next week: March 6, 2019, 6pm

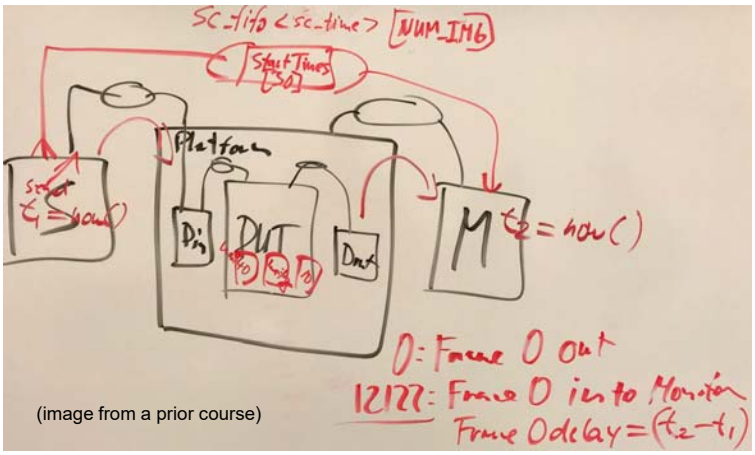
Project Assignment 8

- Step 1: Logging of simulation time and frame delay
  - Expected execution log with timing instrumentation

```
0: Stimulus sent frame  1.
0: Stimulus sent frame  2.
0: Monitor received frame  1 with     0 ms delay.
0: Stimulus sent frame  3.
0: Monitor received frame  2 with     0 ms delay.
0: Stimulus sent frame  4.
0: Monitor received frame  3 with     0 ms delay.
[...]
0: Stimulus sent frame 20.
0: Monitor received frame 19 with     0 ms delay.
0: Monitor received frame 20 with     0 ms delay.
0: Monitor exits simulation.
```

Project Assignment 8

- Step 1: Logging of simulation time and frame delay
  - Extended test bench structure:



(image from a prior course)

Project Assignment 8

- Step 2: Back-annotate timing in DUT components
  - Insert wait-for-time statements into your model
  - Assume Rasberry Pi performance:

```
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
```

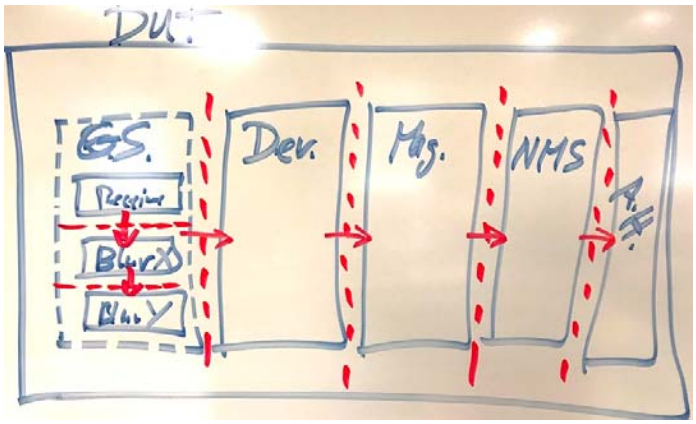
Project Assignment 8

- Step 3: Logging of frame throughput
  - Expected execution log with throughput instrumentation

```
[...]
133570: Monitor received frame 19 with    28120 ms delay.
133570:   7.030 seconds after previous frame,  0.142 FPS.
140600: Monitor received frame 20 with    28120 ms delay.
140600:   7.030 seconds after previous frame,  0.142 FPS.
140600: Monitor exits simulation.
```


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



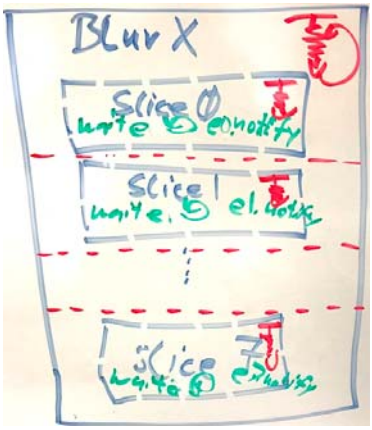
Project Assignment 8

- Step 6: Slice the BlurX and BlurY blocks into parallel components

```
DUT canny
|------ Gaussian_Smooth gaussian_smooth
|       |------ Receive_Image receive
|       \------ Gaussian_Kernel gauss
|------ BlurX blurX
|       |------ BlurX_Slice sliceX1
|       |------ BlurX_Slice sliceX2
|       |       [...]
|       \------ BlurX_Slice sliceX8
|------ BlurY blurY
|       |------ BlurY_Slice sliceY1
|       |       [...]
|       \------ BlurY_Slice sliceY8
|------ Derivative_X_Y derivative_x_y
|------ Magnitude_X_Y magnitude_x_y
|------ Non_Max_Supp non_max_supp
\------ Apply_Hysteresis apply_hysteresis
```


Project Assignment 8

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



Project Assignment 8

- Deliverable
  - Timing observed after each step:

Model	Frame Delay	Throughput	Total time
CannyA8_step1	... ms	... FPS	... ms
CannyA8_step2	... ms	... FPS	... 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

- Performance Estimation on **Prototyping** Platform
  - Measured timing on Raspberry Pi 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`

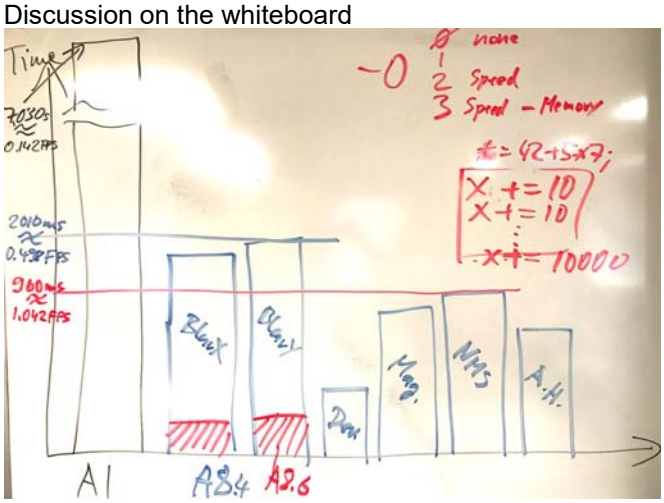
Project Discussion

- Discussion Questions
  - Does the timing meet our real-time goals? No.
  - How far off is it? 7030/33 = 213x
  - What can be done to improve the speed?

  - Pipelining A8, steps 4 and 5
  - Parallelization A8, step 6
  - Hardware optimizations GPU or ASIC for BlurX, BlurY
  - Software optimizations A9, steps 1, 2, and 3
  - Application adjustments Discussion, future work

Project Discussion

- Model Performance Overview
  - Discussion on the whiteboard



Project Assignment 9

- Task: Throughput optimization of Canny Edge Decoder
  - Utilize compiler optimizations
  - Replace floating-point with fixed-point arithmetic
- Steps
  1. Turn on compiler optimizations, measure speedup per block
  2. Apply speedup to back-annotated timing (overall 2.5x)
  3. 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:
  - Next week: March 13, 2019, 6pm

Project Assignment 9

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

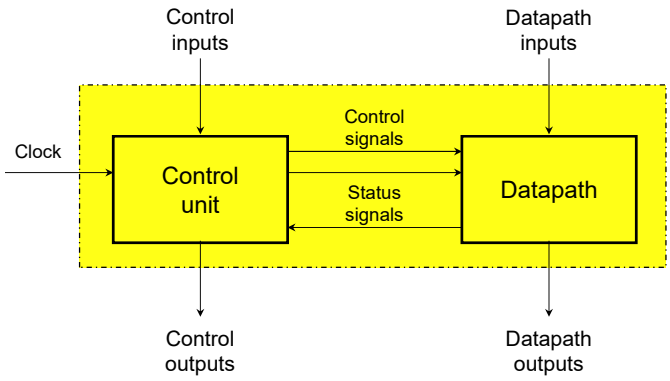
```
T1  =  ...ms / ...ms = ...
T2  =  ...ms / ...ms = ...
T3  =  ...ms / ...ms = ...
T4  =  ...ms / ...ms = ...
T5  =  ...ms / ...ms = ...
T6  =  ...ms / ...ms = ...
T7  =  ...ms / ...ms = ...
Tot =  ...ms / ...ms = ...
```

  - Timing observed after each step:

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

Hardware Modeling in Embedded Systems

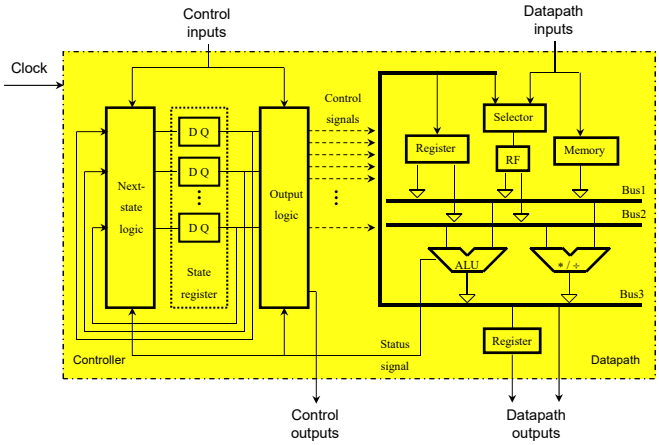
- Register Transfer Level (RTL) Abstraction
  - Block diagram of generic RTL component (high level)



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

RTL Modeling in SystemC

- Example Modules of RTL Components
  - D-FF
  - D-FF with Asynchronous Reset
  - Shifter
  - Counter
  - State machine
  - Memory

➢ Source: Aleksandar Milenkovic (ECE at UAH)
➢ *CPE 626:*
The SystemC Language – VHDL, Verilog Designer's Guide
➢ homepages.cae.wisc.edu/~ece734/SystemC/cpe626-SystemC-L2.ppt

RTL Modeling in SystemC

- Example Modules of RTL Components
  - D-FF

```
// dff.h
#include "systemc.h"
SC_MODULE(dff)
{
  sc_in<bool> din;
  sc_in<bool> clock;
  sc_out<bool> dout;

  void doit()
  {
     dout = din;
  };
 SC_CTOR(dff)
 {
   SC_METHOD(doit);
   sensitive_pos << clock;
 }
};
```

- Source: Aleksandar Milenkovic (ECE at UAH)

RTL Modeling in SystemC

- Example Modules of RTL Components
  - D-FF with asynchronous reset

```
// dffa.h
#include "systemc.h"
SC_MODULE(dffa)
{ sc_in<bool> clock;
  sc_in<bool> reset;
  sc_in<bool> din;
  sc_out<bool> dout;
  void do_ffa()
  { if (reset) {
       dout = false;
    } else if (clock.event()) {
        dout = din;}
  };
  SC_CTOR(dffa) {
    SC_METHOD(do_ffa);
          sensitive(reset);
          sensitive_pos(clock);
  }
};
```

- Source: Aleksandar Milenkovic (ECE at UAH)

RTL Modeling in SystemC

- Example Modules of RTL Components
  - Shifter

```
// shift.h
#include “systemc.h”
SC_MODULE(shift)
{
  sc_in<sc_bv<8> > din;
  sc_in<bool> clk;
  sc_in<bool> load;
  sc_in<bool> LR;
  sc_out<sc_bv<8> > dout;
  sc_bv<8> shiftval;
  void shifty();
  SC_CTOR(shift)
  {
     SC_METHOD(shifty);
       sensitive_pos (clk);
  }
};
```

```
// shift.cc
#include “shift.h”
void shift::shifty()
{
  if (load) {
     shiftval = din;
  } else if (!LR) {
     shiftval.range(6,0) = shiftval.range(7,1);
     shiftval[7] = ‘0’;
  } else if (LR) {
     shiftval.range(7,1) = shiftval.range(6,0);
     shiftval[0] = ‘0’;
  }
  dout = shiftval;
}
```

- Source: Aleksandar Milenkovic (ECE at UAH)

RTL Modeling in SystemC

- Example Modules of RTL Components
  - Counter

```
#include "systemc.h"
SC_MODULE(counter)
{
  sc_in<bool> clock;
  sc_in<bool> load;
  sc_in<bool> clear;
  sc_in<sc_int<8> > din;
  sc_out<sc_int<8> > dout;
  int countval;
  void onetwothree();

  SC_CTOR(counter)
  {
     SC_METHOD(onetwothree);
          sensitive_pos (clock);
  }
};
```

```
// counter.cc
#include "counter.h"
void counter::onetwothree()
{
  if (clear) {
     countval = 0;
  } else if (load) {
  countval = din.read(); // use read when a
  // type conversion is happening
  // from an input port
  } else {
    countval++;
   }
   dout = countval;
}
```

- Source: Aleksandar Milenkovic (ECE at UAH)

RTL Modeling in SystemC

- Example Modules of RTL Components
  - State Machine
    - Voicemail controller
    - States
      - Main
      - Send
      - Review
      - Repeat, Erase, Record
    - Outputs
      - play, recrd, erase, save and address
    - Two SC_METHOD processes
      - getnextst – calculates the next state based on input and current state
      - setstate – copies the calculated next_state to the current_state every positive clock edge on input clk
    - Source: Aleksandar Milenkovic (ECE at UAH)

RTL Modeling in SystemC

- Example Modules of RTL Components
  - State Machine

```
// stmach.h
#include "systemc.h"
enum vm_state {
  main_st, review_st, repeat_st,
  save_st, erase_st, send_st,
  address_st, record_st,
  begin_rec_st, message_st
};
SC_MODULE(stmach)
{
  sc_in<bool> clk;
  sc_in<char> key;
  sc_out<sc_logic> play;
  sc_out<sc_logic> recrd;
  sc_out<sc_logic> erase;
  sc_out<sc_logic> save;
  sc_out<sc_logic> address;
  sc_signal<vm_state> next_state;
  sc_signal<vm_state> current_state;
  void getnextst();
  void setstate();

SC_CTOR(stmach)
{
  SC_METHOD(getnextst);
     sensitive << key << current_state;
  SC_METHOD(setstate);
    sensitive_pos (clk);
}
};
```

    - Source: Aleksandar Milenkovic (ECE at UAH)

RTL Modeling in SystemC

- Example Modules of RTL Components
  - State Machine (cont'd)

```
// stmach.cc
#include "stmach.h"
void stmach::getnextst()
{
  play = sc_logic_0;
  recrd = sc_logic_0;
  erase = sc_logic_0;
  save = sc_logic_0;
  address = sc_logic_0;
  switch (current_state) {
    case main_st:
      if (key == '1') {
          next_state = review_st;
      } else {
        if (key == '2') {
            next_state = send_st;
        } else {
            next_state = main_st;
        }
      }
```

```
    case review_st:
      if (key == '1') {
          next_state = repeat_st;
      } else {
        if (key == '2') {
           next_state = save_st;
        } else {
          if (key == '3') {
             next_state = erase_st;
          } else {
            if (key == '#') {
              next_state = main_st;
            } else {
              next_state = review_st;
            }
          }
        }
      }
```

- Source: Aleksandar Milenkovic (ECE at UAH)

RTL Modeling in SystemC

- Example Modules of RTL Components
  - State Machine (cont'd)

```
    case repeat_st:
       play = sc_logic_1;
       next_state = review_st;
    case save_st:
       save = sc_logic_1;
       next_state = review_st;
    case erase_st:
       erase = sc_logic_1;
       next_state = review_st;
    case send_st:
       next_state = address_st;
    case address_st:
       address = sc_logic_1;
       if (key == '#') {
          next_state = record_st;
       } else {
          next_state = address_st;
       }
```

```
    case record_st:
       if (key == '5') {
         next_state = begin_rec_st;
       } else {
         next_state = record_st;
       }
     case begin_rec_st:
       recrd = sc_logic_1;
       next_state = message_st;
     case message_st:
       recrd = sc_logic_1;
       if (key == '#') {
          next_state = send_st;
       } else {
          next_state = message_st;
       }
  } // end switch
} // end method
void stmach::setstate()
{
  current_state = next_state;
}
```

- Source: Aleksandar Milenkovic (ECE at UAH)

RTL Modeling in SystemC

- Example Modules of RTL Components
  - Memory

```
// ram.h
#include "systemc.h"
SC_MODULE(ram)
{
  sc_in<sc_int<8> > addr;
  sc_in<bool> enable;
  sc_in<bool> readwr;
  sc_inout_rv<16> data;

  void read_data();
  void write_data();
  sc_lv<16> ram_data[256];

SC_CTOR(ram)
{
  SC_METHOD(read_data);
    sensitive << addr << enable << readwr;
  SC_METHOD(write_data);
   sensitive << addr << enable << readwr << data;
}
};
```

  - Source: Aleksandar Milenkovic (ECE at UAH)

RTL Modeling in SystemC

- Example Modules of RTL Components
  - Memory (cont'd)

```
// ram.cc
#include "ram.h"
void ram::read_data()
{
  if (enable && ! readwr ) {
    data = ram_data[addr.read()];
  } else {
    data = "ZZZZZZZZZZZZZZZZ";
  }
}

void ram::write_data()
{
  if (enable && readwr) {
    ram_data[addr.read()] = data;
  }
}
```

  - Source: Aleksandar Milenkovic (ECE at UAH)

RTL Modeling in SystemC

- Example Modules of RTL Components
  - Source: Aleksandar Milenkovic (ECE at UAH)
  - D-FF
  - D-FF with Asynchronous Reset
  - Shifter
  - Counter
  - State machine
  - Memory