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
- EECS 222 Project
  - Review
  - Discussion
  - Wrap up

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!

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 3

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!

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 4

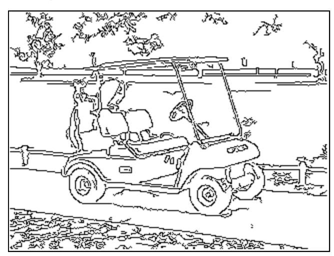
Project Review and Discussion

- Project Assignment 1
  - Introduction to the Canny Edge Detector in ANSI C
- Project Assignment 4
  - SLDL model in SpecC or SystemC
- Project Assignment 5
  - Video stream processing and structural test bench model
- Project Assignment 6 (shortened)
  - Structural refinement of DUT and Gaussian Smooth
- Project Assignment 7 (shortened)
  - Performance estimation and measurement
- Project Assignment 8
  - Pipelining and parallelization of the model
- Project Assignment 9
  - Compiler and application optimizations

EECS 222 Project

- Application Example: Canny Edge Detector
  - Embedded system model for image processing: Automatic edge detection in a digital camera

golfcart.pgm



golfcart.pgm_s_0.60_l_0.30_h_0.80.pgm

  - Application source and documentation:
    - John Canny, *"A Computational Approach to Edge Detection"*, IEEE TPAMI, 1986.
    - http://en.wikipedia.org/wiki/Canny_edge_detector
    - ftp://figment.csee.usf.edu/pub/Edge_Comparison/source_code/canny.src

Project Assignment 1

- Task: Introduction to Application Example
  - Canny Edge Detector
  - Algorithm for edge detection in digital images
- Steps
  1. Setup your Linux programming environment
  2. Download, adjust, and compile the application C code with the GNU C compiler (**`gcc`**)
  3. Study the application
  4. Fix a bug and clean-up the source code
- Deliverables
  - Source code and text file: **`canny.c`**, **`canny.txt`**
- Due
  - Wednesday, January 15, 2020, 6pm

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 7

Project Assignment 4

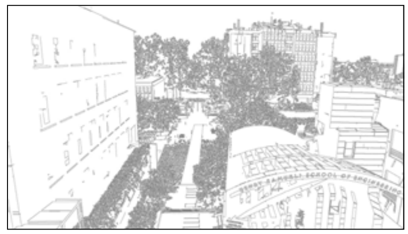
- Task: SLDL Model of the Canny Edge Detector
  - Convert ANSI-C source code into SLDL model
  - Choose either SpecC or SystemC for simulation
- Steps
  1. Prepare clean SLDL source code without compiler warnings
  2. Fix configuration parameters to compile-time constants
  3. Remove or replace dynamic memory allocation
     - ➢ No calls to **`malloc()`**, **`calloc()`**, and **`free()`** in the model
- Deliverables
  - **`canny.sc`** or **`canny.cpp`** (choose one!)
  - **`canny.txt`**
- Due
  - Wednesday, February 5, 2020, 6pm

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 8

EECS 222 Project

- Application Example: Canny Edge Detector
  - Embedded system model for image processing: Automatic Edge Detection in a Digital Video Camera

EngPlaza001.bmp



EngPlaza001_edges.pgm

  - Video taken by a drone hovering over UCI Engineering Plaza
    - Available on the server: **~eecs222/public/video/**
    - High resolution, 2704 by 1520 pixes
    - Video length 9 seconds, using 20 extracted frames for test bench model

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 9

Project Assignment 5

- Task: Structural Test Bench Model
  - Expected instance tree

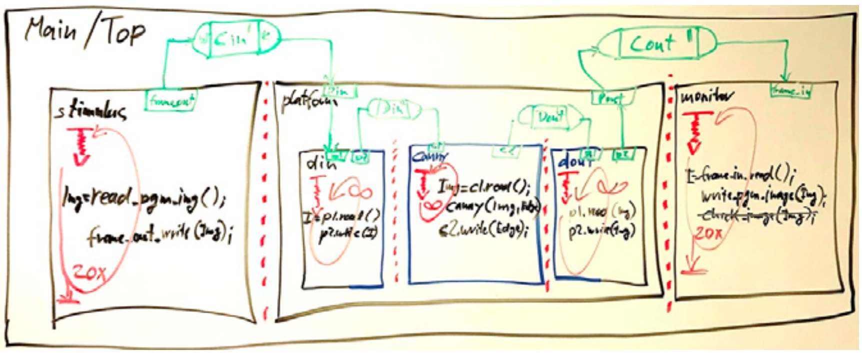
```
Main / Top
|------ Stimulus stimulus
|------ Platform platform
|       |------ DataIn din
|       |------ DUT canny
|       \------ DataOut dout
\------ Monitor monitor
```

  - Communication via standard channels
    - SystemC: **sc_fifo<IMAGE>** based on class **IMAGE**
    - SpecC: **c_img_queue** based on typedef **img**
  - ➢ Pay attention to stack sizes!

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 10

Project Assignment 5

- Structural Test Bench for the Canny Edge Detector
  - Discussion on whiteboard: Top-level structure, platform for DUT



Project Assignment 6

- Task: Hierarchical DUT of the Canny Edge Detector
  - Refine the structural hierarchy of the DUT block
  - (skipped: refine the structural hierarchy of Gaussian Smooth)
- Steps
  1. Refine the DUT structure
     - Gaussian Smooth, Derivative, …, Apply Hysteresis
  2. Visualize the structural hierarchy of the model
  - (skipped: decomposition of Gaussian Smooth)
- Deliverables
  - **`canny.sc`** or **`canny.cpp`** (choose one!)
  - **`canny.tree`**
- Due: February 19, 2020, 6pm

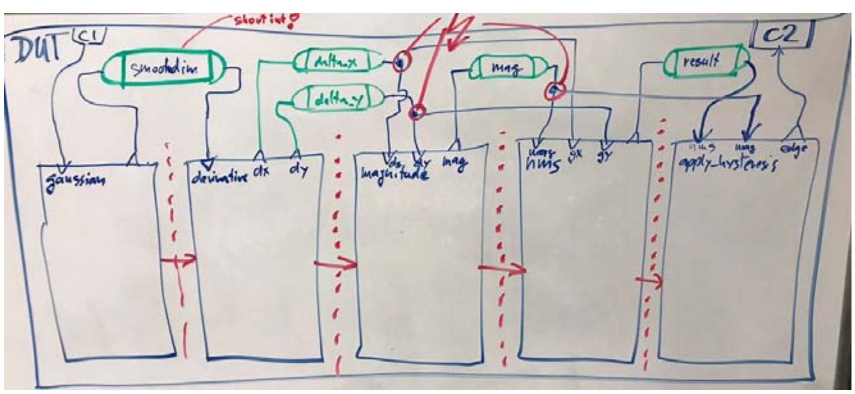
Project Assignment 6

- Step 1: Refined hierarchy of the DUT block
  - Expected instance tree

```
Platform platform
|------ DataIn din
|------ DUT canny
|       |------ Gaussian_Smooth gaussian_smooth
|       |------ Derivative_X_Y derivative_x_y
|       |------ Magnitude_X_Y magnitude_x_y
|       |------ Non_Max_Supp non_max_supp
|       \------ Apply_Hysteresis apply_hysteresis
\------ DataOut dout
```


Project Assignment 6

- Structural model of the DUT of the Canny Edge Detector
  - Discussion on whiteboard: Refined DUT structure



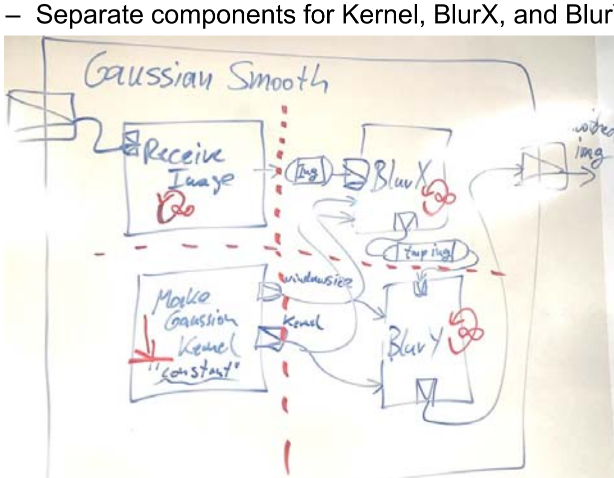
Project Assignment 6

- Skipped: Refined Hierarchy of Gaussian Smooth block
  - Instance tree

```
DUT canny
|------ Gaussian_Smooth gaussian_smooth
|       |------ Receive_Image receive
|       |------ Gaussian_Kernel gauss
|       |------ BlurX blurX
|       \------ BlurY blurY
|------ Derivative_X_Y derivative_x_y
|------ Magnitude_X_Y magnitude_x_y
|------ Non_Max_Supp non_max_supp
\------ Apply_Hysteresis apply_hysteresis
```


Project Assignment 6

- Skipped: Refined Hierarchy of Gaussian Smooth block
  - Separate components for Kernel, BlurX, and BlurY



(whiteboard image from a prior course)

Project Assignment 7

- Task: Performance Estimation of the Canny Example
  - Profiling to estimate relative computational complexity
  - Instrumentation to measure absolute timing as reference
- Steps
  1. Profile the application, identify performance bottlenecks
     - Relative complexity: Use GNU profiling tools
  2. Instrument the application, measure timing on reference platform
     - Absolute timing: Use Linux timing APIs
- Deliverable
  - **`canny.txt`** (including tables of obtained results)
- Due
  - February 19, 2020, 6pm (combined with A6)

Project Assignment 7

➢ Performance Estimation of the Canny Edge Detector

- Step 1: Profile the application components , obtain relative computational complexity
  - Use a provided C++ model (derived from SpecC model)
  - Use GNU profiling tools
    - ➢ **`g++ -pg, gprof`**
    - Compile the SystemC source code with option **`-pg`**
    - Run the simulation once (with instrumentation, **`gmon.out`**)
    - Run the profiler: **`gprof Canny`**
    - Validate the reported call tree
    - Analyze the “flat profile” for the DUT components
    - Select the main functions of interest

Project Assignment 7

- Step 1: Profile the application components, obtain relative computational complexity
  - Expected complexity comparison (in **canny.txt**):

```
Gaussian_Smooth                          ...%
|------ Receive_Image     ...%
|------ Gaussian_Kernel   ...%
|------ BlurX             ...%
\------ BlurY             ...%
Derivative_X_Y                           ...%
Magnitude_X_Y                            ...%
Non_Max_Supp                             ...%
Apply_Hysteresis                         ...%
                                         100%
```

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 19

Project Assignment 7

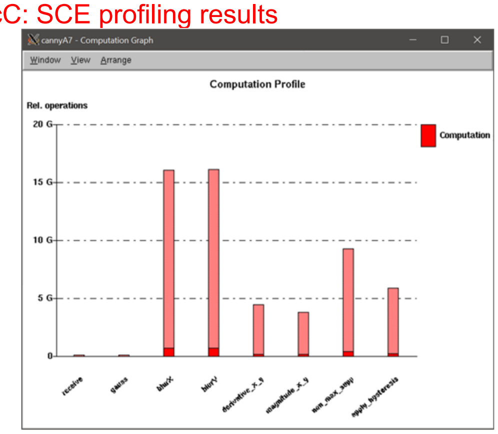
- Step 1: Profile the application components, obtain relative computational complexity
  - Expected complexity comparison (in **canny.txt**):

```
Gaussian_Smooth                          9.15s 61.7%
|------ Receive_Image     0.00s  0.0%
|------ Gaussian_Kernel   0.00s  0.0%
|------ BlurX             4.34s 29.2%
\------ BlurY             4.81s 32.4%
Derivative_X_Y                           0.95s  6.4%
Magnitude_X_Y                            0.66s  4.4%
Non_Max_Supp                             2.10s 14.2%
Apply_Hysteresis                         1.98s 13.3%
                                               100%
```

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 20

Project Assignment 7

- (skip): Profile the application components, obtain relative computational complexity
  - Reference: An alternative profiling approach
    SpecC: SCE profiling results



Project Assignment 7

- (skip): Profile the application components, obtain relative computational complexity
  - Reference: An alternative profiling approach
    SpecC: SCE profiling results

```
Gaussian_Smooth                        30.5G 56.9%
|------ Receive_Image      0.0G  0.0%
|------ Gaussian_Kernel    0.0G  0.0%
|------ BlurX             15.2G 28.4%
\------ BlurY             15.3G 28.5%
Derivative_X_Y                          4.3G  8.1%
Magnitude_X_Y                           3.7G  6.9%
Non_Max_Supp                            9.2G 17.2%
Apply_Hysteresis                        5.8G 10.8%
                                              100%
```

Project Assignment 7

- Step 2: Instrument the application components, obtain absolute timing on reference platform
  - ➢ Since we do not have a prototyping platform available, we use the department server as reference
  - – Instrument your model source code:

```
#include <time.h>
clock_t Tstart, Tstop;
double T1 = 0.0;
...
Tstart = clock();
f();
Tstop = clock();
T1 = (double)(Tstop-Tstart)/CLOCKS_PER_SEC;
```

  - – Use global variables for this temporary instrumentation

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 23

Project Assignment 7

- Step 2: Instrument the application components, obtain absolute timing on reference platform
  - – Expected complexity comparison (also in **canny.txt**):

```
Gaussian_Smooth                              ...sec ...%
|------ Receive_Image     ...sec ...%
|------ Gaussian_Kernel   ...sec ...%
|------ BlurX             ...sec ...%
\------ BlurY             ...sec ...%
Derivative_X_Y                               ...sec ...%
Magnitude_X_Y                                ...sec ...%
Non_Max_Supp                                 ...sec ...%
Apply_Hysteresis                             ...sec ...%
                                                    100%
```

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 24

Project Assignment 7

- Step 2: Instrument the application components, obtain absolute timing on reference platform
  - Expected complexity comparison (also in **`canny.txt`**):
    C++ model: Timing measurement results on Linux server

```
Gaussian_Smooth                          6.83s 52.2%
|------ Receive_Image     0.00s  0.0%
|------ Gaussian_Kernel   0.00s  0.0%
|------ BlurX             2.97s 22.7%
\------ BlurY             3.86s 29.5%
Derivative_X_Y                           1.12s  8.6%
Magnitude_X_Y                            1.04s  7.9%
Non_Max_Supp                             2.08s 15.9%
Apply_Hysteresis                         2.02s 15.4%
                                                100%
```

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 25

Project Discussion

- Reference: Instrument the application components, obtain absolute timing on **prototyping** platform
  - Measured timing on Raspberry Pi 3 board: ARM-based quad-core processor (1.2GHz)

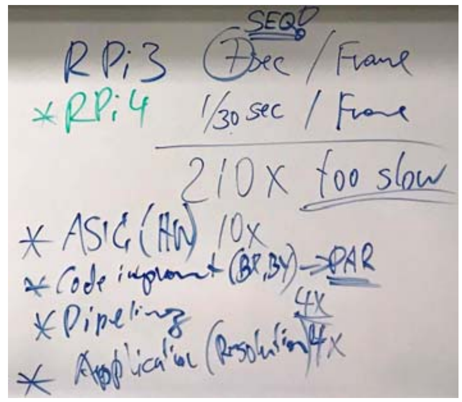
```
Receive_Image          0 ms per frame
Make_Kernel            0 ms per frame
BlurX               1880 ms per frame
BlurY               2010 ms per frame
Derivative_X_Y       530 ms per frame
Magnitude_X_Y        910 ms per frame
Non_Max_Supp         960 ms per frame
Apply_Hysteresis     740 ms per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

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 26

Project Discussion

- Discussion Questions
  - Does the timing meet our real-time goals?
  - What can be done to improve the speed?

  - ➢ Pipelining
  - ➢ Parallelization
  - ➢ Hardware optimizations
  - ➢ Software optimizations
  - ➢ Application adjustments



EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 27

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

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 28

Project Assignment 8

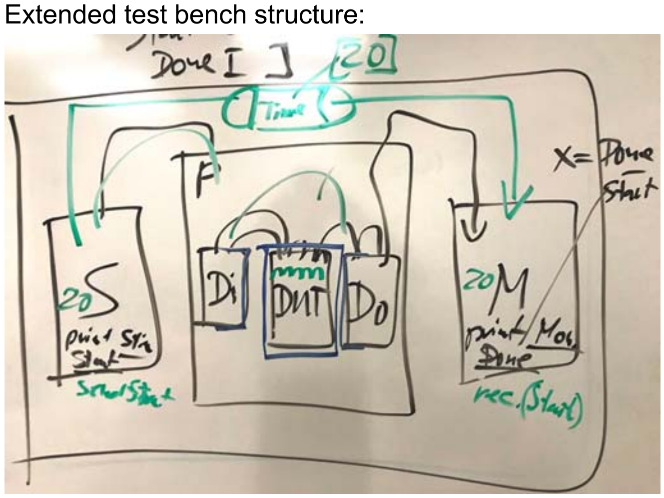
- Step 1: Logging of simulated time and frame delay
  - Expected execution log with timing instrumentation

```
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.
```

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 29

Project Assignment 8

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



EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 30

Project Assignment 8

- Step 2: Back-annotate timing in DUT components
  - Insert wait-for-time statements into your model
  - Assume Rasberry Pi 3 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
```

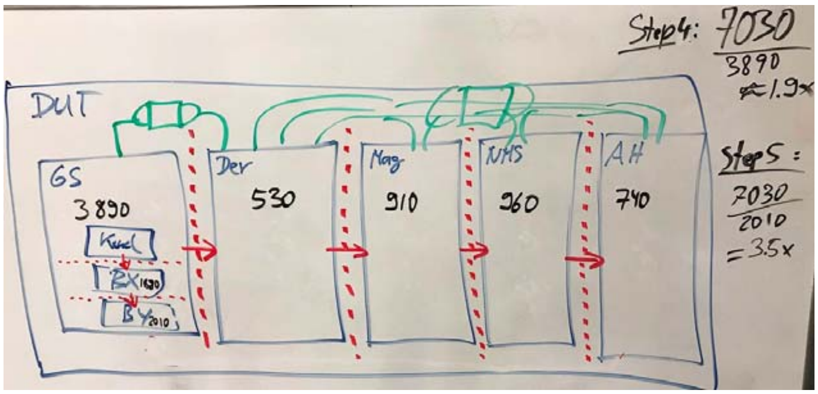

Project Assignment 8

- 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

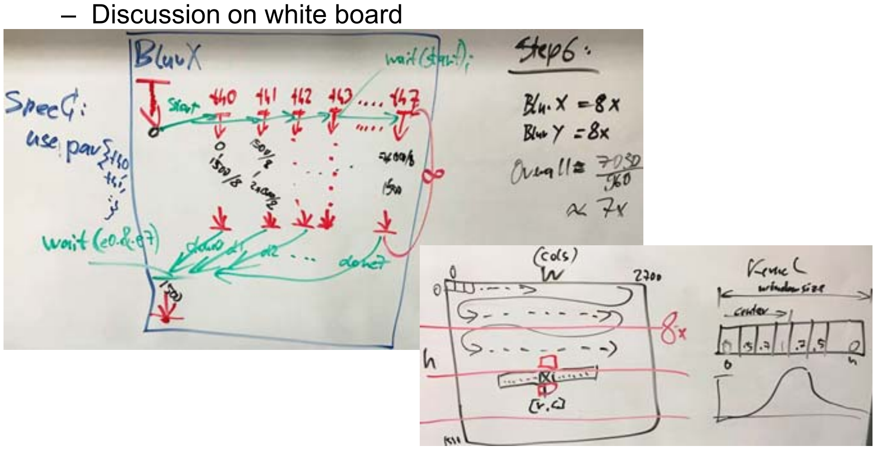
- Step 4: Pipeline the DUT into stages
- Step 5: Integrate Gaussian Smooth into pipeline stages
  - Discussion on whiteboard: Chart of refined DUT structure



EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 33

Project Assignment 8

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



EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 34

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

- 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

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 37

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

EECS222: Embedded System Modeling, Lecture 18 (c) 2020 R. Doemer 38

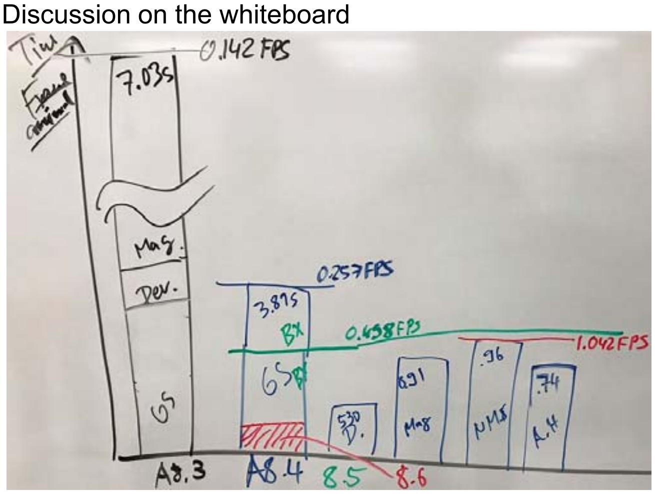
Project Discussion

- 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

- 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
```

Project Assignment 9

- Deliverables
  - Compiler Optimizations: Speed-up 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

- Deliverables
  - Compiler Optimizations: Speed-up observed for each block :

```
Tgk =  0.00 ms / 0.00 ms = n/a
Tbx =  4.79 ms / 0.96 ms = 5.00
Tby =  3.36 ms / 1.04 ms = 3.23
Tde =  1.13 ms / 0.36 ms = 3.14
Tma =  1.01 ms / 0.86 ms = 1.17
Tnm =  2.09 ms / 1.34 ms = 1.56
Tah =  2.09 ms / 0.81 ms = 2.58
Tot = 14.47 ms / 5.37 ms = 2.69
```

Project Assignment 9

- Deliverables
  – Timing observed after each step: SpecC Model

Model	Frame Delay	Throughput	Total time
CannyA9_step2	3752 ms	2.604 FPS	8746 ms
CannyA9_step3	1270 ms	7.812 FPS	2939 ms
CannyA9_step4	1180 ms	8.475 FPS	2737 ms

  – Timing observed after each step: SystemC Model

Model	Frame Delay	Throughput	Total time
CannyA9_step2	5428 ms	2.604 FPS	8746500 us
CannyA9_step3	1810 ms	7.812 FPS	2903250 us
CannyA9_step4	1608 ms	8.475 FPS	2701250 us

Project Discussion

- Status before A8 and A9:
  – 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, parallelization A8
  ➢ HW and SW optimizations A9
- Final questions:
  – Does the timing meet our real-time goals? Still no.
  – How far off is it? (0.295/2.5)/0.0333 = 3.54x
  – What can be done to improve the speed?
  ➢ Keep improving pipeline bottlenecks
  ➢ Accept lower image resolution
  ➢ Accept lower frame rate
  ➢ …