ECPS 203
Embedded Systems Modeling and Design
Lecture 19

Rainer Dömer

doemer@uci.edu

Center for Embedded and Cyber-physical Systems
University of California, Irvine

Lecture 19: Overview

- Course Administration
  - Final course evaluation
- Project Discussion
  - A1: Introduction of Canny Edge Detector application
  - A2: Clean C++ model with static memory allocation
  - A4: From single image to video stream processing
  - A5: Test bench model in SystemC
  - A6: Structural DUT module and algorithm profiling
  - A7: Performance measurement on prototyping board
  - A8: Pipelined and parallel model with back-annotated timing
  - A9: Throughput optimization by pipeline load balancing
    - ➢ Discussion
- Final Project Report

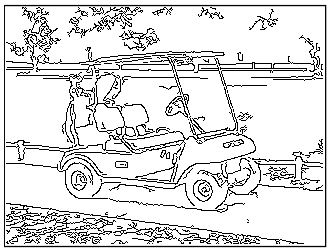
Course Administration

- Final Course Evaluation
  - Open until end of 10th week (Sunday night)
  - Nov. 26, 2018, through Dec. 9, 2018, 11:45pm
  - Online via EEE Evaluation application
- Mandatory Evaluation of Course and Instructor
  - Voluntary
  - Anonymous
  - Very valuable
- Please spend 5 minutes for this survey!
  - Your feedback is appreciated!

ECPS 203 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:
    - http://marathon.csee.usf.edu/edge/edge_detection.html
    - http://en.wikipedia.org/wiki/Canny_edge_detector

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, determine function-call tree
- Deliverables
  - Source code and text file: **`canny.c`**, **`canny.txt`**
- Due
  - Wednesday: October 10, 2018, 6pm

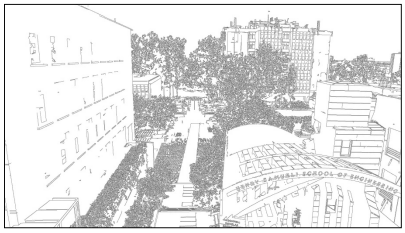
Project Assignment 2

- Task: Clean C++ model with static memory allocation
  - Prepare the C++ source code for modeling in SystemC
  - Configure parameters for specific application
  - Apply static memory allocation
- Steps
  1. Fix the off-by-one bug in the **`non_max_supp`** function
  2. Clean-up the code for compilation without warnings
  3. Fix configuration parameters to compile-time constants
  4. Remove or replace dynamic memory allocation
- Deliverables
  - Source code and text file: **`canny.cpp`**, **`canny.txt`**
- Due
  - Wednesday: October 17, 2018, 6pm

ECPS 203 Project

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

Engineering001.bmp



Engineering001_edges.pgm

- Process video shot by a drone flying over Engineering Plaza
  - Fly a drone over UCI Engineering Plaza, take video of buildings
  - Record a color video stream in high resolution
  - Extract a set of video frames suitable for use in our test bench

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2018 R. Doemer 7

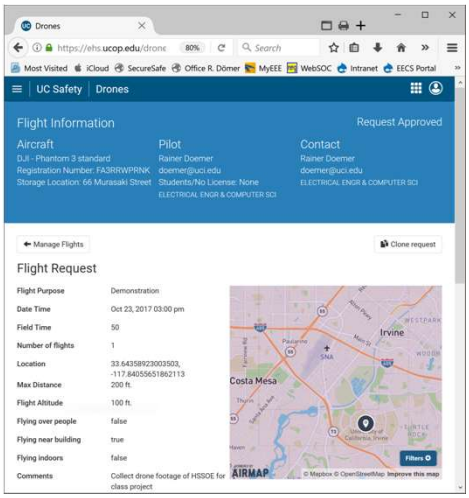
ECPS 203 Project: Drone Flight

- Capture Video Footage of Engineering Buildings
  - Google Map of UCI Engineering Quad

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2018 R. Doemer 8

ECPS 203 Project: Drone Flight

- Capture Video Footage of Engineering Buildings
  - Drone flights in US require approval by the Federal Aviation Administration (FAA)
  - On UCI campus, Environmental Health & Safety (EHS) department is in charge of Unmanned Aircraft Safety
  - ➢ Flight request approved
    - Thursday, October 19, 2017



ECPS 203 Project: Drone Flight

- Capture Video Footage of Engineering Buildings
  - Drone Equipment
    - DJI Phantom 3 Standard Quadcopter
    - Remote Control with Mobile Device

[Image source: dji.com]

  - Drone carrries a Camera attached to a Gimble
    - Video stream stored on a SD memory card, e.g. DJI_0001.MOV
    - Video is 30 frames per second
    - Frames are 2704 by 1520 pixels

ECPS 203 Project: Drone Flight

- Capture Video Footage of Engineering Buildings
  - Screen Shot of Drone Control App on Mobile Device

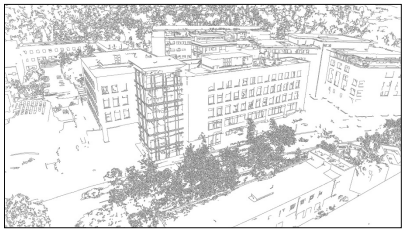
ECPS 203 Project: Drone Flight

- Capture Video Footage of Engineering Buildings
  - Drone flight demonstration (Fall 2017)

ECPS 203 Project

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

Engineering012.png



Engineering012_edges.pgm

  - Video taken by a drone flying over UCI Engineering Plaza
    - Available on the server: **`~ecps203/public/DroneFootage/`**
    - High resolution, 2704 by 1520 pixes
    - Representative sample, using 30 extracted frames for test bench model

Project Assignment 4

- Task: From Single Image to Video Stream Processing
  - Prepare a sequence of image frames from the video
  - Convert the Canny application to process the video frames
- Steps
  1. Extract 30 of video frames suitable for use in a test bench
  2. Convert the color frames to grey-scale images in PGM format
  3. Recode your Canny C++ model to process the video frames
     - To run Canny application successfully, increase stack size
     - Adjust Canny parameters for the “best looking” output images
- Deliverables
  - Source code and text file: **`Canny.cpp`**, **`Canny.txt`**
- Due
  - Wednesday, October 31, 2018, 6pm

Project Assignment 5

- Task: Test bench for the Canny Edge Detector
  - Convert C++ model to SystemC model
  - Add a test bench structure around the C++ model
  - Wrap DUT into a platform model with explicit I/O units
- Steps
  1. Create test bench structure: Stimulus, Platform, Monitor
  2. Create platform model: DataIn, DUT, DataOut
  3. Localize functions and use **`sc_fifo`** channels for communication
     - Pay attention to stack sizes for every thread
- Deliverables
  - SystemC source code and text file: **`Canny.cpp`**, **`Canny.txt`**
- Due
  - Wednesday, November 7, 2018, 6pm

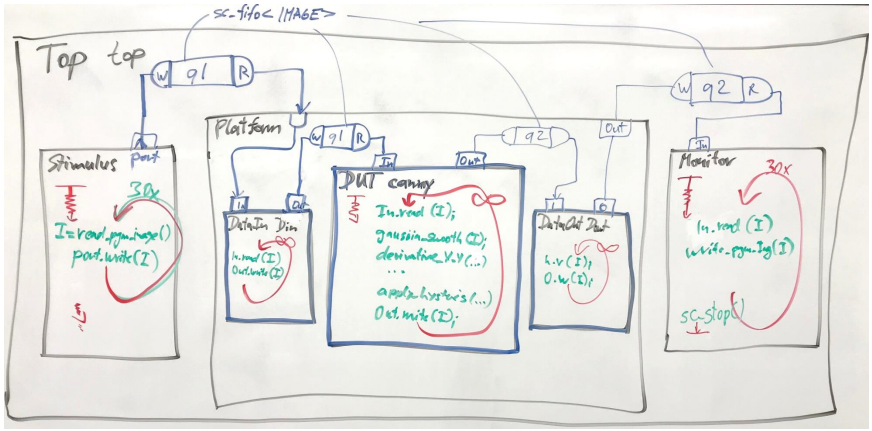
Project Assignment 5

- Task: Test bench for the Canny Edge Detector
  - Expected instance tree

```
Top top
|------ Monitor monitor
|------ Platform platform
|       |------ DUT canny
|       |------ DataIn din
|       |------ DataOut dout
|       |------ sc_fifo<IMAGE> q1
|       \------ sc_fifo<IMAGE> q2
|------ Stimulus stimulus
|------ sc_fifo<IMAGE> q1
\------ sc_fifo<IMAGE> q2
```

Project Assignment 5

- Task: Test bench for the Canny Edge Detector
  - Discussion on whiteboard: Top-level and Platform structure



Project Assignment 6

- Task: Structural refinement of the DUT module
  - Refine the structural hierarchy of the DUT module
  - Refine the structural hierarchy of the Gaussian Smooth module
  - Profile the relative complexity of the Canny functions
- Steps
  1. Create structure in DUT: Gaussian Smooth, …, Apply Hysteresis
  2. Create structure in Gaussian Smooth: Input, Gauss, BlurX, BlurY
  3. Profile the algorithm, obtain relative computational complexity
- Deliverables
  - **`Canny.cpp`** (refined structural model)
  - **`Canny.txt`** (profile of relative complexity of the DUT modules)
- Due
  - Wednesday, November 14, 2018, 6pm

Project Assignment 6

- Step 1: Refined structure of the DUT module
  - Expected module 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
```

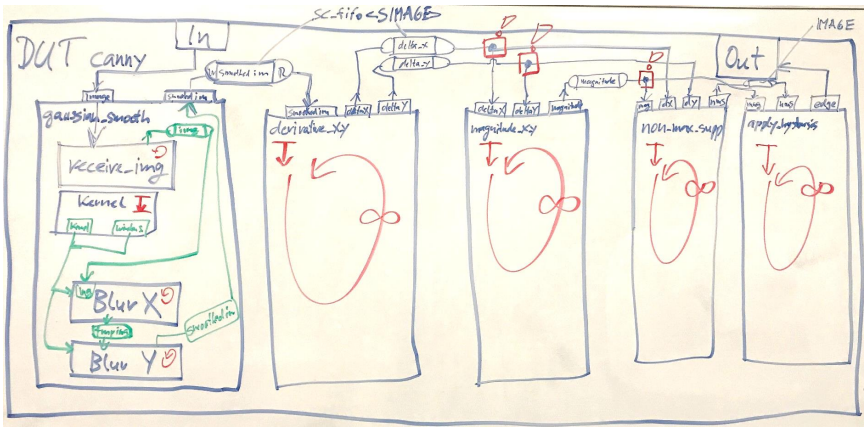

Project Assignment 6

- Step 2: Refined structure of the Gaussian Smooth block
  - Expected module 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

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



Project Assignment 6

- Step 3: Profile the Canny functions
  - ➢ Performance profiling of the Canny Edge Detector
  - ➢ Determine the relative complexity of the Canny functions
    - Is there any performance bottleneck?
    - If so, where?
  - Use the GNU C/C++ profiling tools
    - ➢ **`g++ -pg`**
    - ➢ **`gprof`**
    1. Compile the SystemC source code with option **`-pg`**
    2. Run the simulation once with instrumentation, obtain **`gmon.out`**
    3. Run the profiler: **`gprof Canny`**
    4. Validate the reported call tree
    5. Analyze the “flat profile” for the DUT components (**`self`**)

Project Assignment 6

- Step 3: Profile the Canny functions, obtain relative computational complexity
  - Profiled complexity comparison (in **`Canny.txt`**):

```
Gaussian_Smooth                    40.57%
|------ Gaussian_Kernel    0.00%
|------ BlurX              17.23%
\------ BlurY              23.34%
Derivative_X_Y              6.26%
Magnitude_X_Y              15.90%
Non_Max_Supp               23.98%
Apply_Hysteresis           12.29%
                           100%
```

  - ➢ Profiling results vary, but Gaussian Smooth is a bottleneck!

Project Assignment 7

- Task: Performance measurement on prototyping board
  - Run C++ model of Canny Edge Detector on Raspberry Pi
  - Obtain absolute timing measurements of Canny functions
- Steps
  1. Prepare the prototyping board with Raspbian operating system
  2. Upload **`Canny.cpp`** from Assignment 4 and compile it
  3. Instrument the source code with real-time measurements
  4. Note the computation delays of the major Canny functions
- Deliverables
  - **`Canny.cpp`** (model instrumented with timing measurements)
  - **`Canny.txt`** (table of measured delays)
- Due
  - Wednesday, November 21, 2018, 6pm

Project Assignment 7

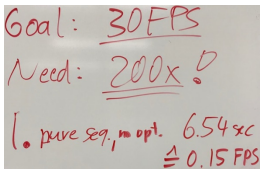
- Task: Performance measurement on prototyping board
  - Measured delays on Raspberry Pi 3 (in `Canny.txt`):

```
Gaussian_Smooth                  3.53 sec
|------ Gaussian_Kernel  0.00 sec
|------ BlurX             1.71 sec
\------ BlurY             1.82 sec
Derivative_X_Y                   0.48 sec
Magnitude_X_Y                    1.03 sec
Non_Max_Supp                     0.83 sec
Apply_Hysteresis                 0.67 sec
TOTAL                            6.54 sec
```

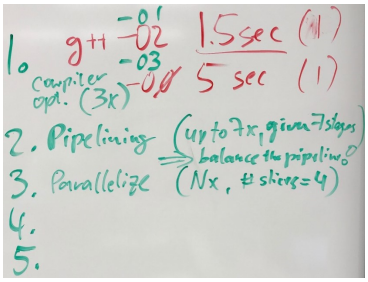
  - ➢ This performance is far too slow for real-time video!
  - ➢ Discussion: What options exist to speed this up?

Project Assignment 7

- Discussion: Measured delays on Raspberry Pi 3
  - `TOTAL` `6.54 seconds`
  - ➢ This performance is far too slow for real-time video!

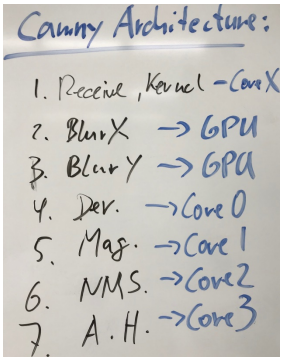


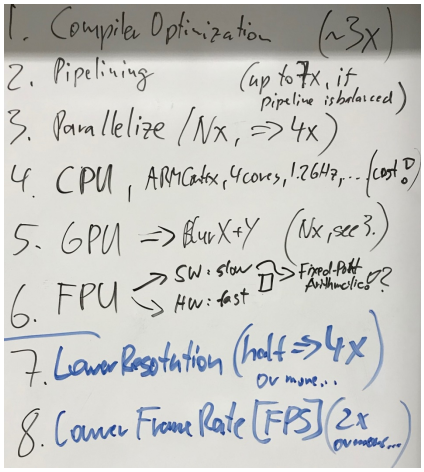
  - ➢ Discussion: What options exist to speed this up?



Project Assignment 7

- Discussion: Measured delays on Raspberry Pi 3
  - **`TOTAL`** **`6.54 seconds`**
  - ➢ Discussion: What options exist to speed this up?



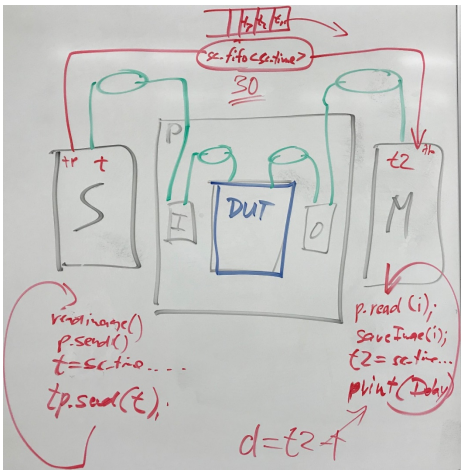


Project Assignment 8

- Task: Pipelining and parallelization of the DUT module
  - Back-annotate estimated delays to observe timing in the model
  - Pipeline and parallelize the model to improve throughput
- Steps
  1. Instrument model with simulated time to observe frame delay
  2. Back-annotate estimated timing into DUT components
  3. Improve test bench to observe frame throughput
  4. Pipeline the DUT into a sequence of 7 stages with buffer size 1
  5. Slice the BlurX and BlurY modules into 4 parallel threads
- Deliverables
  - **`Canny.cpp:`** pipelined and parallelized SystemC model
  - **`Canny.txt:`** table of observed frame delays and throughput
- Due: Wednesday, November 28, 2018, 6pm

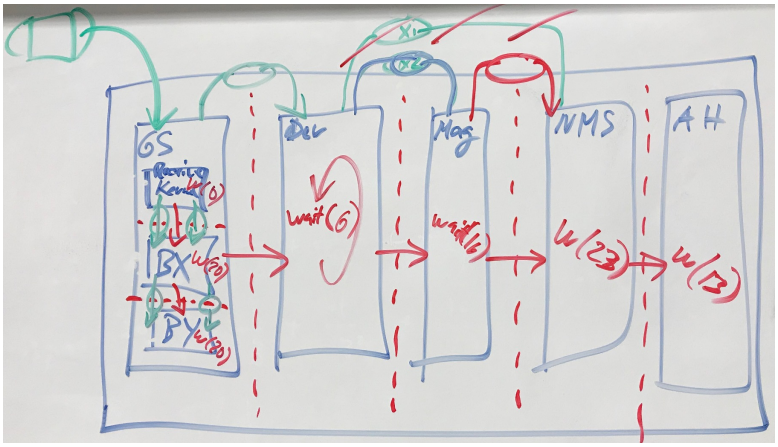
Project Assignment 8

- Timed test bench model for the Canny Edge Detector
  - Discussion on whiteboard: Chart of refined test bench structure



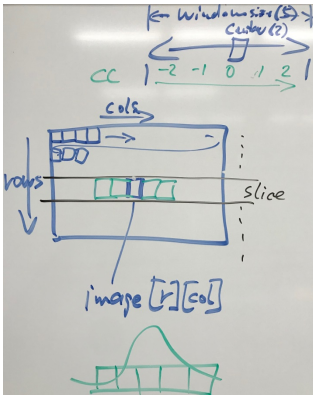
Project Assignment 8

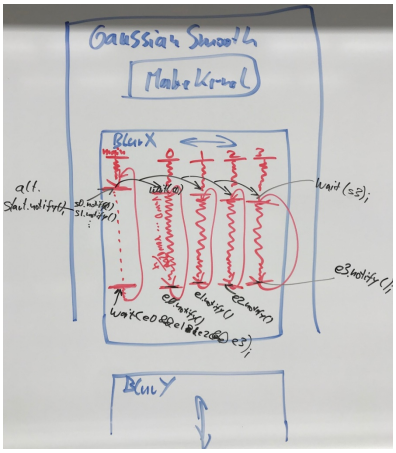
- Pipelined and parallel model of the Canny Edge Detector
  - Discussion on whiteboard: Chart of pipelined DUT structure



Project Assignment 8

- Pipelined and parallel model of the Canny Edge Detector
  - Discussion on whiteboard: Parallel BlurX, BlurY functions (step 5)





Project Assignment 8

- Pipelined and parallel model of the Canny Edge Detector
  - Back-annotation of measured timing delays
  - ➢ 4-way parallelization of BlurX and BlurY modules (step 5)

Receive, Make_Kernel	0 ms	0 ms
BlurX	1710 ms	427 ms
BlurY	1820 ms	455 ms
Derivative_X_Y	480 ms	480 ms
Magnitude_X_Y	1030 ms	1030 ms
Non_Max_Supp	830 ms	830 ms
Apply_Hysteresis	670 ms	670 ms
TOTAL:	6540 ms	3892 ms
Throughput:	1/1820ms	1/1030ms
	0.549 FPS	0.971 FPS

Project Assignment 8

- Pipelined and parallel model of the Canny Edge Detector
  - Expected execution log with timing (after step 5)

```
       0 s: Stimulus sent frame  1.
       0 s: Stimulus sent frame  2.
       0 s: Stimulus sent frame  3.
  [...]
  3422 ms: Stimulus sent frame 16.
  3892 ms: Monitor received frame  1 with  3892 ms delay.
  [...]
 30672 ms: Monitor received frame 27 with 15920 ms delay.
 30672 ms: 1.030 seconds after previous frame, 0.971 FPS.
 31702 ms: Monitor received frame 28 with 15920 ms delay.
 31702 ms: 1.030 seconds after previous frame, 0.971 FPS.
 32732 ms: Monitor received frame 29 with 15920 ms delay.
 32732 ms: 1.030 seconds after previous frame, 0.971 FPS.
 33762 ms: Monitor received frame 30 with 15920 ms delay.
 33762 ms: Monitor exits simulation.
```


Project Assignment 9

- Task: Throughput optimization by pipeline load balancing
  - Optimize the bottleneck stages to improve throughput
  - Prepare final report
- Steps
  1. Apply compiler optimizations for maximum execution speed
  2. Consider fixed-point instead of floating-point arithmetic
  3. Prepare draft of project report
- Deliverables
  - **`Canny.cpp`** (final SystemC model, graded)
  - **`Canny.txt`** (extended performance table, graded)
  - **`Canny.pdf`** (draft report, reviewed but not graded)
- Due
  - Wednesday, December 5, 2018, 6pm

Project Assignment 9

- Step 1: Apply compiler optimizations to reduce execution time
  - Experiment with various compiler options, including:

```
-O2
-O3
-mfloat-abi=hard
-fmpu=neon-fp-armv8
-mneon-for-64bits
```

  - Refer to documentation on
    - GNU compiler
    - ARMv8 Cortex-A53

Project Assignment 9

- Step 2: Consider fixed-point calculations instead of floating-point arithmetic
  - Focus on **`Non_Max_Supp`** module only
  - Convert **`float`** type variables to **`int`** types
  - Replace these lines of code…

```
xperp = -(gx = *gxptr)/((float)m00);
yperp = (gy = *gyptr)/((float)m00);
```

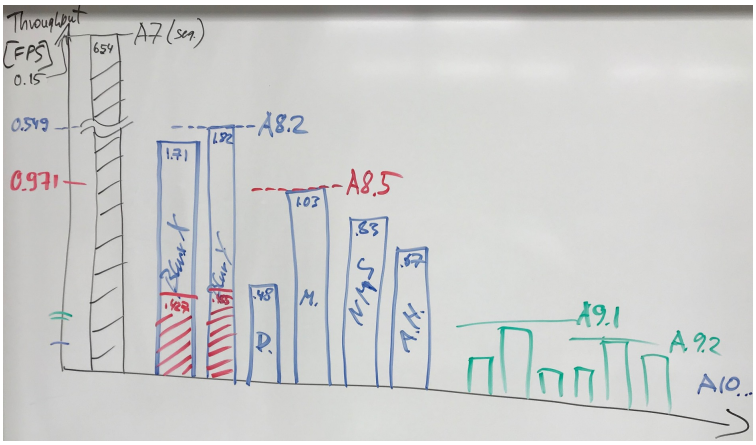
  - … with this code

```
gx = *gxptr;
gy = *gyptr;
xperp = -(gx<<16)/m00;
yperp = (gy<<16)/m00
```

  - Measure the timing difference on the prototyping board
  - Measure and evaluate the image quality (**`ImageDiff`**)

Project Optimization Chart

- Optimizations and their Effect on Throughput
  - Chart to visualize optimizations applied in assignments



Final Project Report

- Technical Report (in lieu of Final Exam)
  - Allocated time and room for final exam
    - Monday, December 10, 8:00-10:00am (DBH 1200)
    - ➢ *Not applicable, we use electronic submission instead!*
  - Format: Final Project Report
    - Submission script: `~ecps203/bin/turnin.sh`
    - Directory name: `final`
    - Deliverables: `ECPS203_Report.pdf` `Canny.cpp`
  - A9 deadline: Draft report (for early feedback)
    - Wednesday, December 5, 2018, 6pm
  - Hard deadline: Final report (graded!)
    - Monday, December 10, 2018, 6pm

Final Project Report

- Technical Report about the Course Project
  - Title
    - *Modeling of a Canny Edge Detector for Embedded Systems Design*
  - Contents
    - "Story" of the Canny Edge Detector project
      - From downloading the initial C reference code
      - Via modeling and simulating in SystemC
      - To performance optimization for real-time video
    - Describe relevant project assignments 1 through 9
    - Focus on the *reasoning* behind the optimizations
  - Length
    - About 12 pages (including title page, figures, and bibliography)

Final Project Report

1. Title page
   - Project title, author, date, course number and title
   - Abstract
2. Introduction
   - Embedded system modeling and design concepts
   - The IEEE SystemC language
3. Case Study of a Canny Edge Detector for Real-time Video
   - Structure of the Canny edge detection algorithm
   - Modeling and simulation in IEEE SystemC
   - Model refinement for pipelining and parallelization
   - Performance estimation and throughput optimization
   - Real-time video performance results
4. Summary and Conclusion
   - Lessons learned
   - Future work
5. References

Final Project Report

- Grading Criteria
  - A. General report quality
    1) Story line and readability
    2) Organization and structure, completeness
    3) Explanation and accuracy of obtained results
    4) References and citations
    5) Conclusions and lessons learned
  - B. Focus on engineering and reasoning

    Q1: Why was dynamic memory allocation removed in A2?
    Q2: Why was stack size a problem in A4 and later?
    Q3: Why did we model DataIn and DataOut modules in A5?
    Q4: Why is the relative timing different in A6, A7 and A9?
    Q5: Why did we choose to parallelize Gaussian Smooth in A8?
    Q6: Why does the simulator run-time not improve in A8?
    Q7: Why did you choose/not choose fixed-point arithmetic in A9?
    Q8: How can you achieve real-time video for the end user?