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 Detection 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, algorithm profiling
  - A7: Performance measurement on prototyping board
  - A8: Pipelined and parallel model with back-annotated timing
- Assignment 9
  - Throughput optimization by pipeline load balancing
    - ➢ Discussion

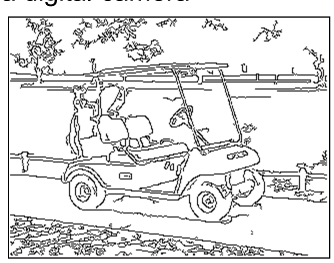
Course Administration

- Final Course Evaluation
  - Open until end of 10th week (Sunday night)
  - Nov. 28, 2017, through Dec. 10, 2017, 11pm
  - 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, next week: October 11, 2017, 6pm

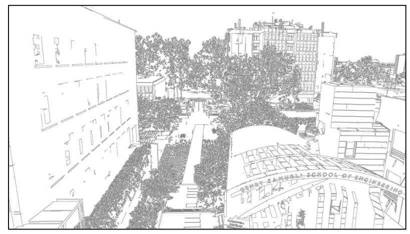
Project Assignment 2

- 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, next week: October 18, 2017, 6pm

ECPS 203 Project

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

Engineering001.bmp



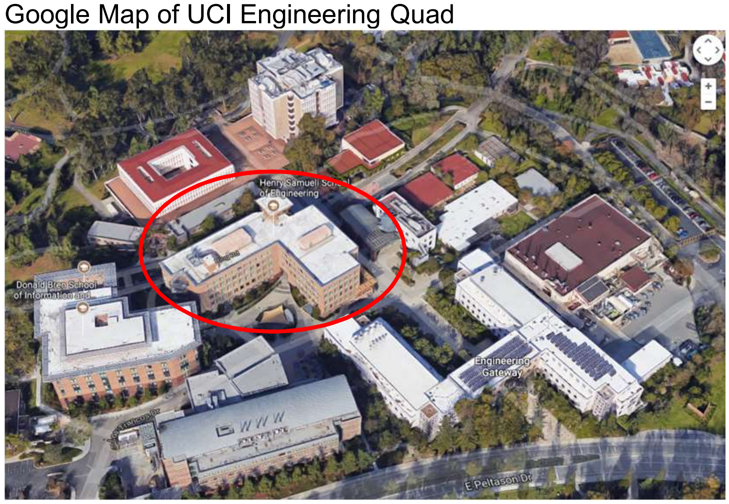
Engineering001_edges.pgm

  - 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, 2704 by 1520 pixels
    - ➢ Extract a set of video frames suitable for use in our test bench

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 7

ECPS 203 Project: Drone Flight

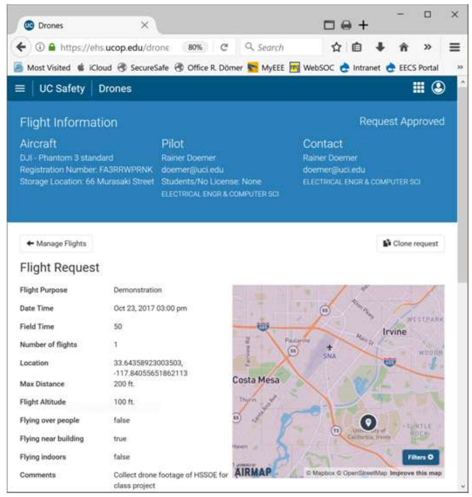
- Capture Video Footage of Engineering Buildings
  - Google Map of UCI Engineering Quad



ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 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

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 11

ECPS 203 Project: Drone Flight

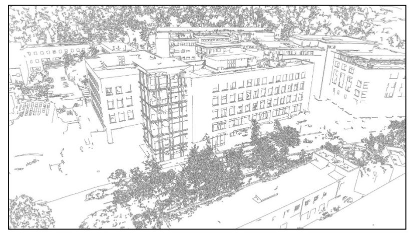
- Capture Video Footage of Engineering Buildings
  - Drone flight demonstration

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 12

ECPS 203 Project: Drone Flight

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

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 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
- Deliverables
  - Source code and text file: **`Canny.cpp`**, **`Canny.txt`**
- Due
  - Wednesday, November 1, 2017, 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 dedicated 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 thread stack sizes
- Deliverables
  – SystemC source code and text file: **`Canny.cpp`**, **`Canny.txt`**
- Due
  – Wednesday, November 8, 2017, 6pm

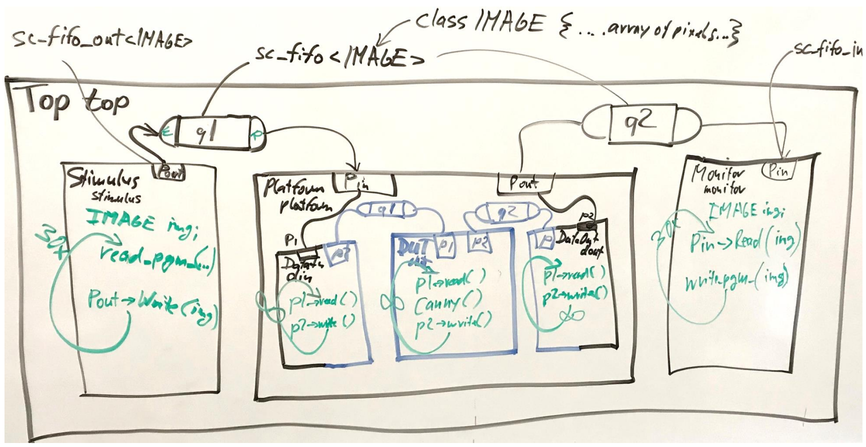
Project Assignment 5

- 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: Chart of top-level 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 15, 2017, 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
```

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 19

Project Assignment 6

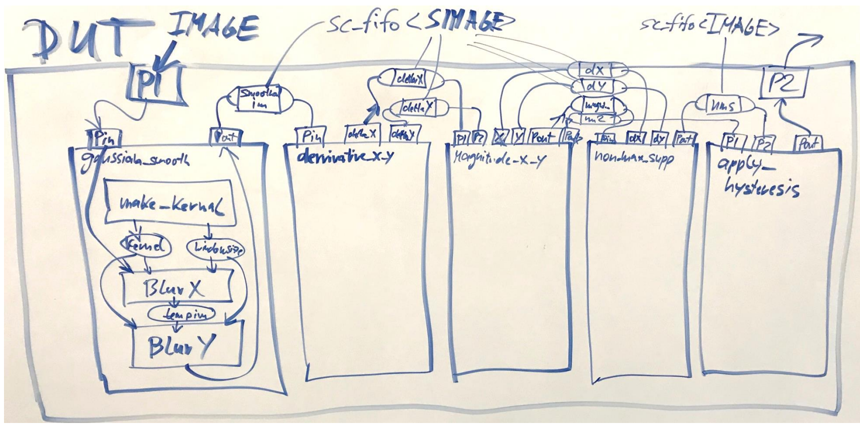
- Step 2: Refined structure of the Gaussian Smooth module
  - Expected module instance tree

```
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
```

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 20

Project Assignment 6

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



ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 21

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`**)

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 22

Project Assignment 6

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

```
Gaussian_Smooth                           42.64%
|------ Gaussian_Kernel   0%
|------ BlurX           22.73%
\------ BlurY           19.91%
Derivative_X_Y                             6.12%
Magnitude_X_Y                             16.09%
Non_Max_Supp                              25.16%
Apply_Hysteresis                           9.80%
                                        100%
```

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

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 23

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 A4 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 22, 2017, 6pm

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 24

Project Assignment 7

- Discussion: Measured Computation Delays
  - Table of measured delays on Raspberry Pi 3 (in `Canny.txt`):

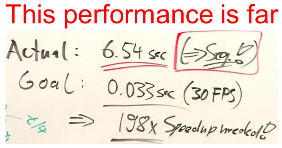
```
– Gaussian_Smooth              3.53 s
–   Gaussian_Kernel     0.00 s
–   BlurX               1.71 s
–   BlurY               1.82 s
– Derivative_X_Y               0.48 s
– Magnitude_X_Y                1.03 s
– Non_Max_Supp                 0.83 s
– Apply_Hysteresis             0.67 s
–                              ======
– TOTAL                        6.54 seconds
```

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

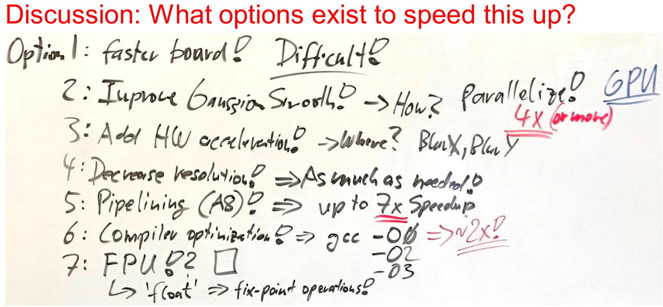
ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 25

Project Assignment 7

- Discussion: Measured Computation Delays
  - **TOTAL** 6.54 seconds
  - ➢ This performance is far too slow for real-time video!



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



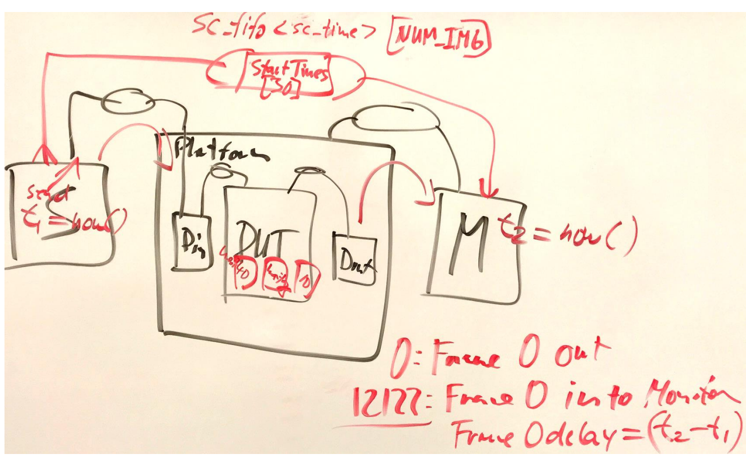
ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 26

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 simulation time to observe frame delay
  2. Back-annotate estimated timing in DUT components
  3. Pipeline the DUT into a sequence of 7 stages with buffer size 1
  4. Slice the BlurX and BlurY modules into parallel threads
- Deliverables
  - `Canny.cpp` (pipelined and parallelized SystemC model)
  - `Canny.txt` (table of observed frame delays)
- Due
  - Wednesday, November 29, 2017, 6pm

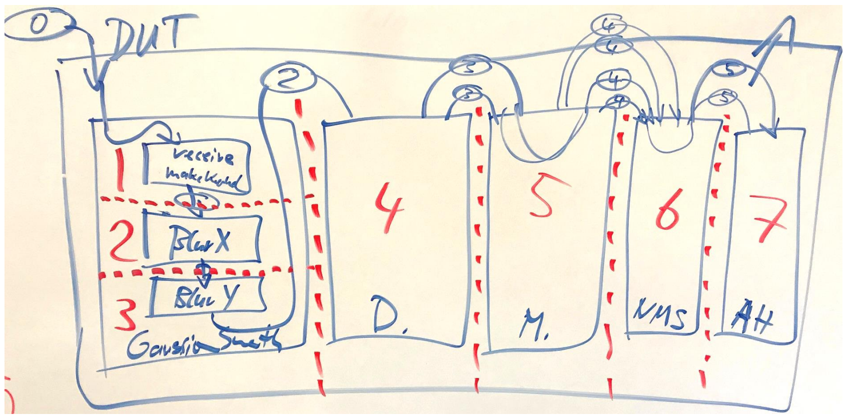
Project Assignment 8

- 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 refined DUT structure



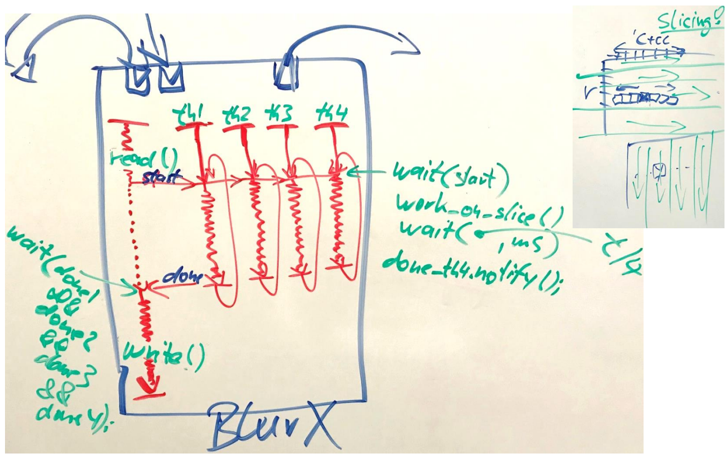
Project Assignment 8

- Pipelined and parallel model of the Canny Edge Detector
  - Back-annotation of measured timing delays (step 2)

```
Receive, Make_Kernel      0 ms
BlurX                  1710 ms
BlurY                  1820 ms
Derivative_X_Y          480 ms
Magnitude_X_Y          1030 ms
Non_Max_Supp            830 ms
Apply_Hysteresis        670 ms
                       =======
TOTAL:                 6540 ms
                       =======
```

Project Assignment 8

- Pipelined and parallel model of the Canny Edge Detector
  - Discussion on whiteboard: Parallel BlurX, BlurY functions (step 4)



ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 31

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 4)

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
	=======	=======

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 32

Project Assignment 8

- Pipelined and parallel model of the Canny Edge Detector
  - Expected execution log with timing (after step 4)

```
     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.
 4452 ms: Stimulus sent frame 17.
 4922 ms: Monitor received frame  2 with  4922 ms delay.
 [...]
17282 ms: Monitor received frame 14 with 14720 ms delay.
17842 ms: Stimulus sent frame 30.
18312 ms: Monitor received frame 15 with 15323 ms delay.
19342 ms: Monitor received frame 16 with 15920 ms delay.
 [...]
32732 ms: Monitor received frame 29 with 15920 ms delay.
33762 ms: Monitor received frame 30 with 15920 ms delay.
33762 ms: Monitor exits simulation.
```


Project Assignment 8

- Pipelined and parallel model of the Canny Edge Detector
  - Timing results observed after each step:

Model	Frame Delay	Total simulation time
CannyA8_step1	0 ms	0 ms
CannyA8_step2	<varies>	<varies>
CannyA8_step3	<varies>	59320 ms
CannyA8_step4	<varies>	33762 ms

  - Discussion:
    - Model in step 1 is untimed
    - Times observed in step 2 may vary due to different communication channels (passing data over/via an intermediate stage)
    - Frame delay in steps 3 and 4 may vary due to different buffer depth of time stamp channel in test bench
    - ➢ Frame delay is not a good measure of performance! What is?

Project Assignment 8

- Discussion of Performance
- Performance metrics observed in Assignment 8
  - Total simulated time
    - Total processing time for our stream of 30 frames
  - Frame delay
    - Processing time for each frame from pipeline input to output
    - Influenced by time-stamp channel depth
    - ➢ Not a good measure!
- Performance metrics in Assignment 9
  - Stage delay
    - Delay incurred in each pipeline stage; *maximum* matters!
  - Pipeline latency
    - *N*max(StageDelay)*, where *N* is the number of stages
  - ➢ Pipeline throughput
    - ➢ Number of frames coming out of the pipeline per second (FPS)

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 35

Project Assignment 9

- Task: Throughput optimization by pipeline load balancing
  - Observe pipeline throughput in the model, measure FPS
  - Optimize the bottleneck stages to improve throughput
- Steps
  1. Improve test bench to measure and display frame throughput
  2. Apply compiler optimizations to reduce execution time
  3. Replace floating-point with fixed-point arithmetic in NMS block
- Deliverables
  - **`Canny.cpp`** (optimized SystemC model)
  - **`Canny.txt`** (table of observed frame throughput)
- Due
  - Wednesday, December 6, 2017, 6pm

ECPS203: Embedded Systems Modeling and Design, Lecture 19 (c) 2017 R. Doemer 36

Project Assignment 9

- Step 1: Improve test bench to measure and display frame throughput
  - Expected log output

```
[...]
17282 ms: Monitor received frame 14 with 14720 ms delay.
17282 ms:   1.030 seconds after previous frame,  0.971 FPS.
17842 ms: Stimulus sent frame 30.
18312 ms: Monitor received frame 15 with 15323 ms delay.
18312 ms:   1.030 seconds after previous frame,  0.971 FPS.
[...]
```


Project Assignment 9

- Step 2: 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 3: Replace floating-point arithmetic with fixed-point calculations
  - Focus on **`Non_Max_Supp`** module only
  - Convert **`float`** type variables to **`int`** types
  - Replace this 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
  - Evaluate the image quality (**`ImageDiff`**)