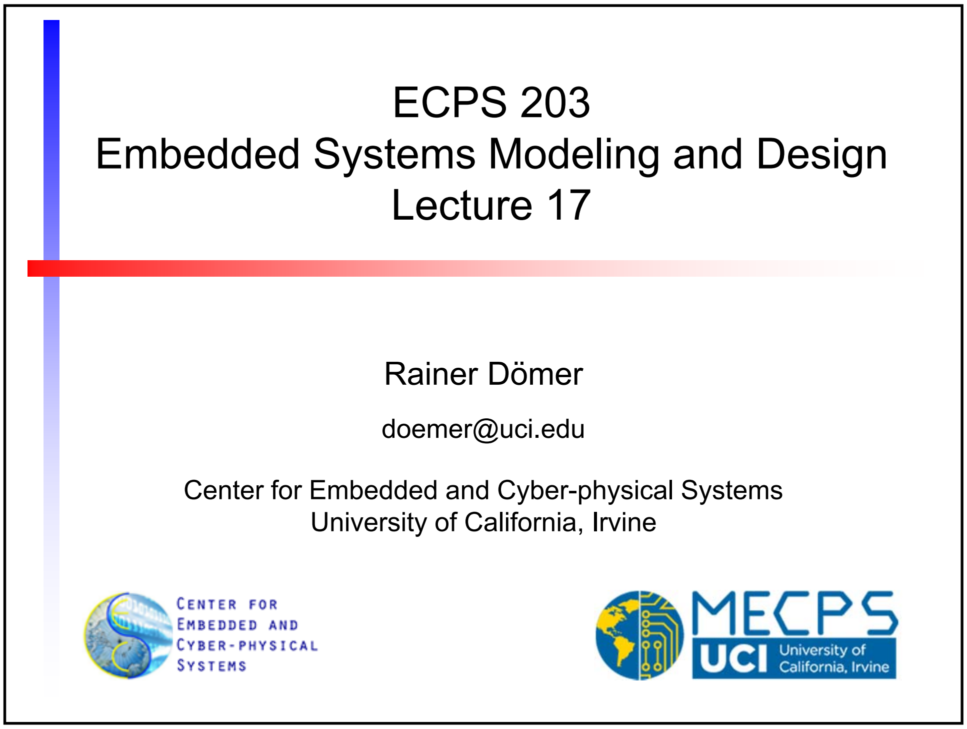
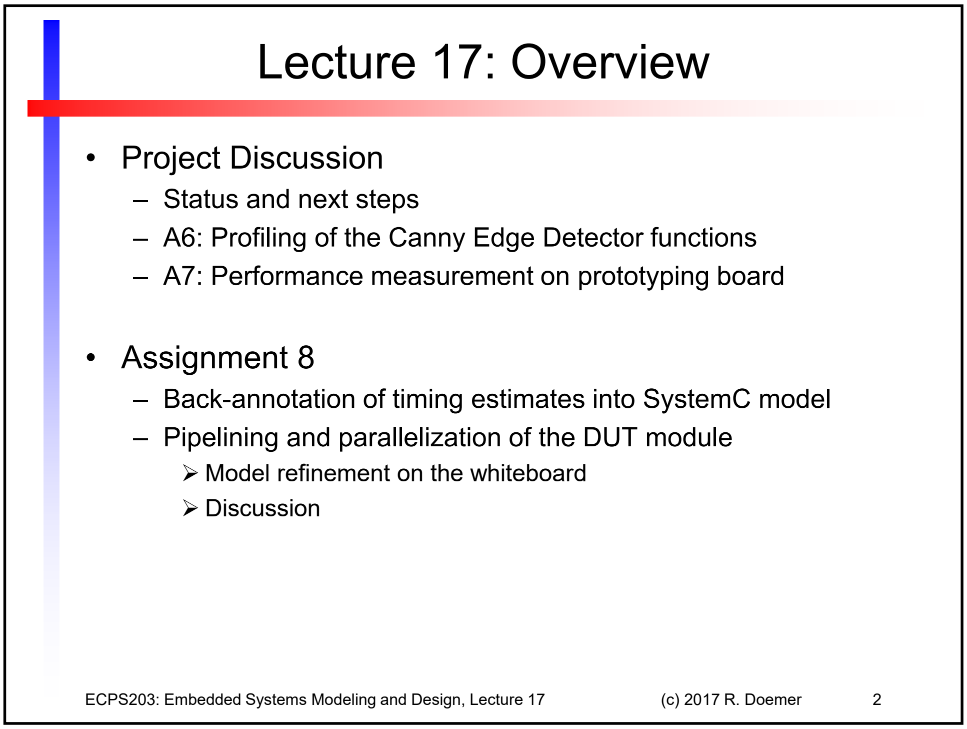


ECPS 203
Embedded Systems Modeling and Design
Lecture 17

Rainer Dömer

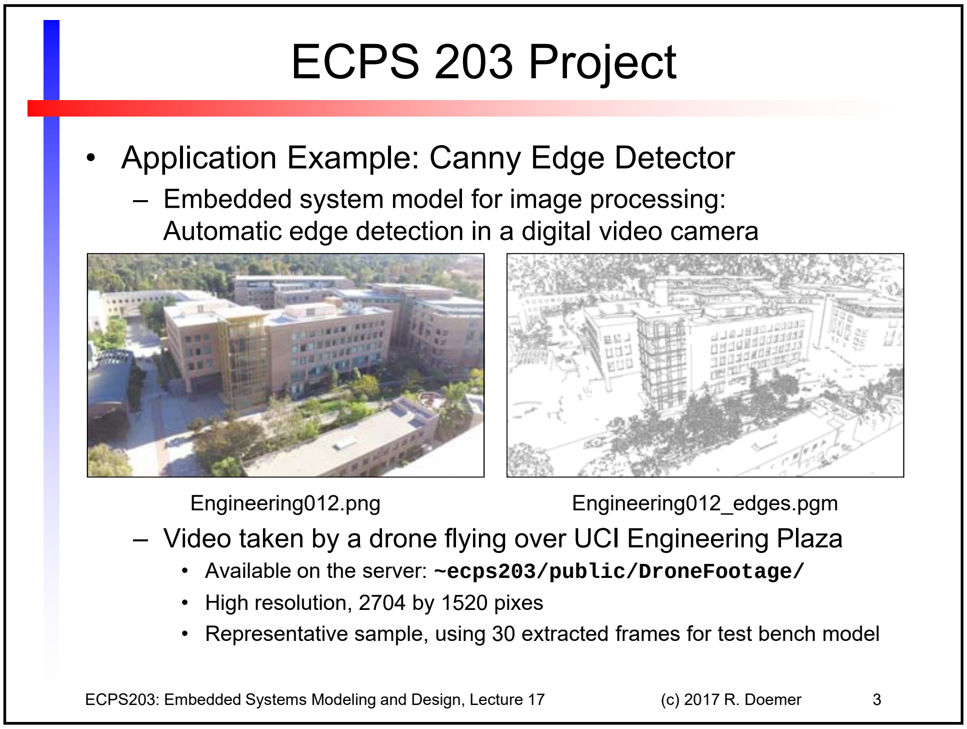
doemer@uci.edu

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



Lecture 17: Overview

- Project Discussion
  - Status and next steps
  - A6: Profiling of the Canny Edge Detector functions
  - A7: Performance measurement on prototyping board
- Assignment 8
  - Back-annotation of timing estimates into SystemC model
  - Pipelining and parallelization of the DUT module
    - ➢ Model refinement on the whiteboard
    - ➢ Discussion



ECPS 203 Project

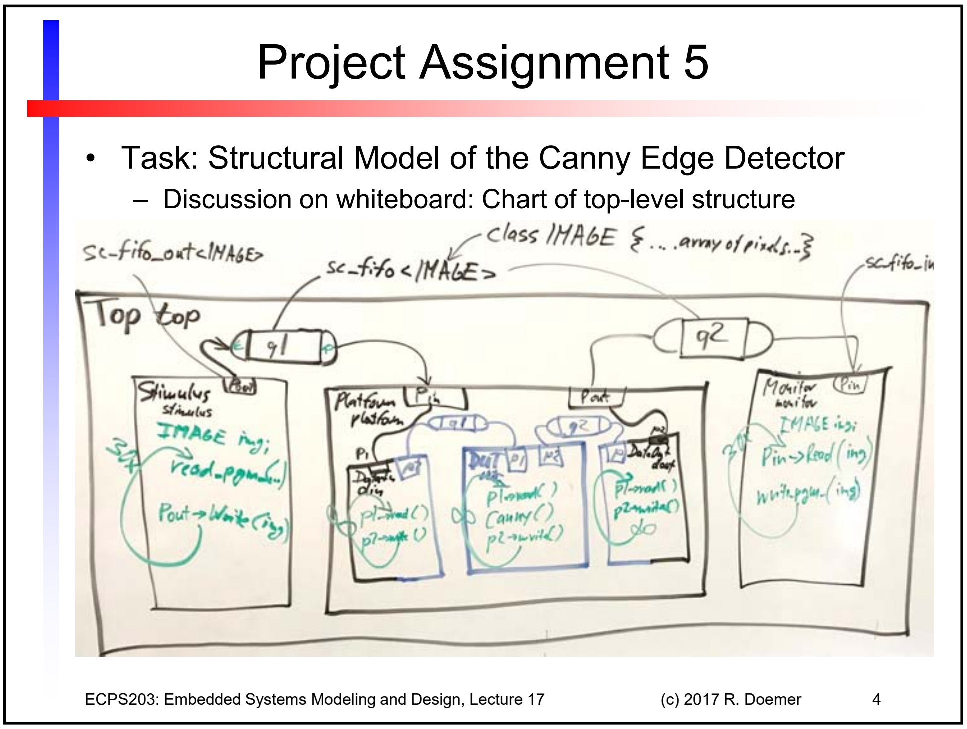
- Application Example: Canny Edge Detector
  - Embedded system model for image processing: Automatic edge detection in a digital video camera

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

ECPS203: Embedded Systems Modeling and Design, Lecture 17 (c) 2017 R. Doemer 3



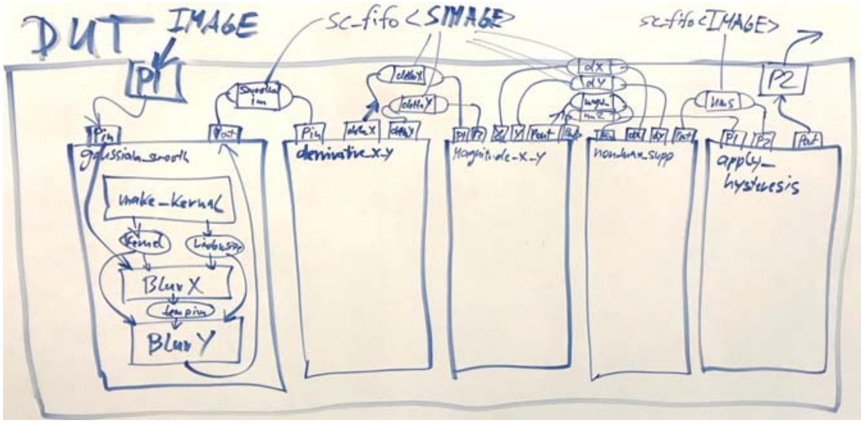
Project Assignment 5

- Task: Structural Model of the Canny Edge Detector
  - Discussion on whiteboard: Chart of top-level structure

ECPS203: Embedded Systems Modeling and Design, Lecture 17 (c) 2017 R. Doemer 4

Project Assignment 6

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

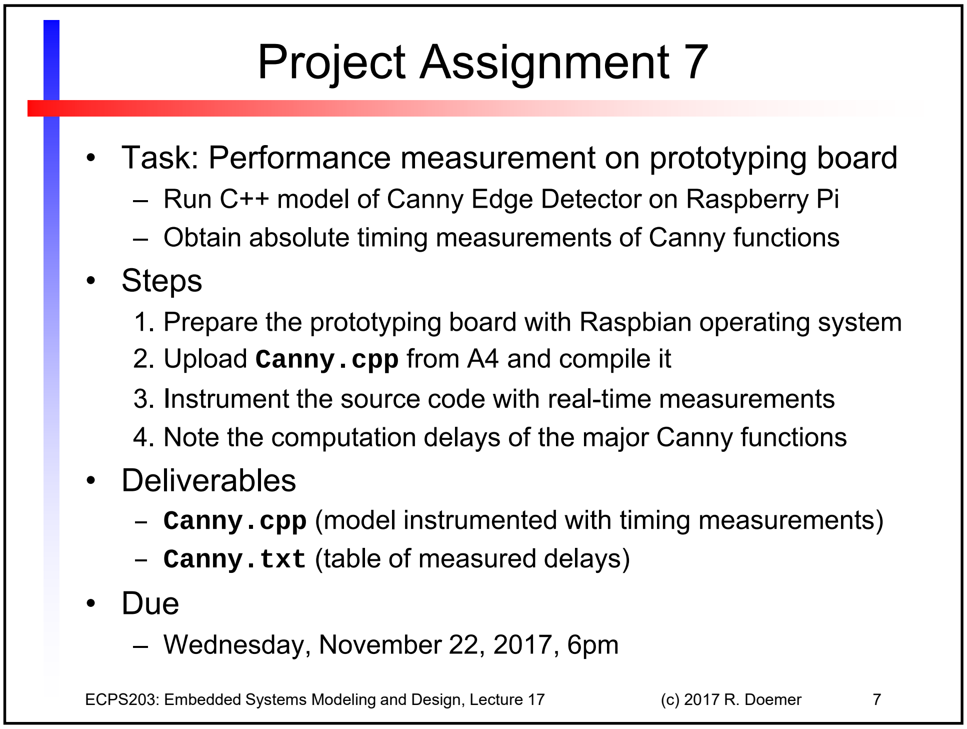


Project Assignment 6

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

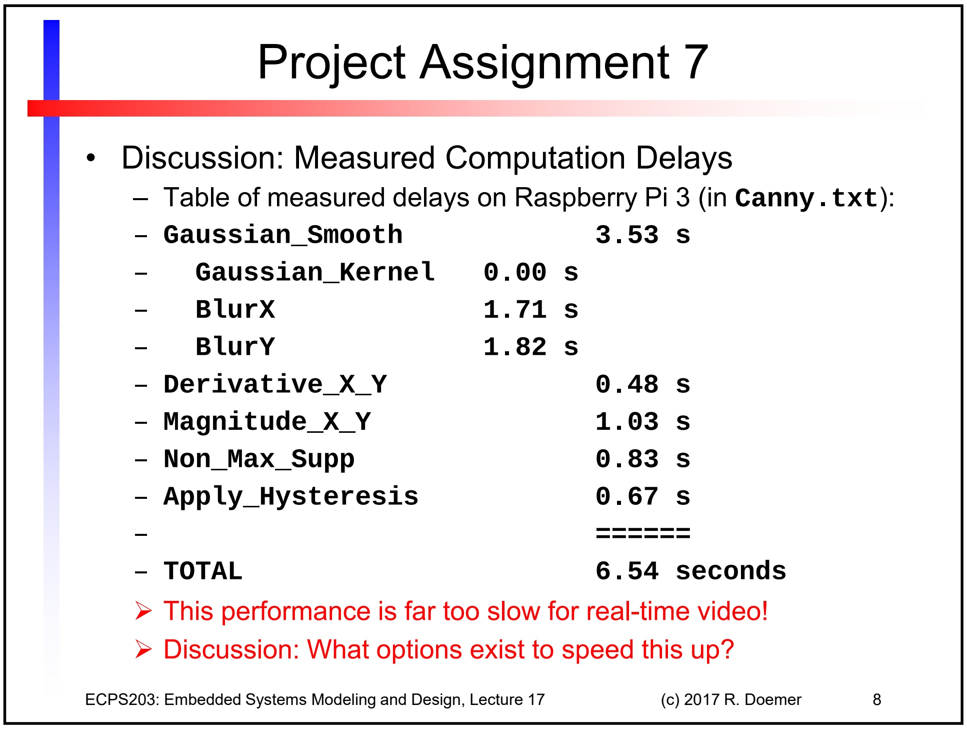
```
Gaussian_Smooth                          42.64%
|------ Gaussian_Kernel   0%
|------ BlurX           22.73%
\------ BlurY           19.91%
Derivative_X_Y                            6.12%
Magnitude_X_Y                            16.09%
Non_Max_Supp                             25.16%
Apply_Hysteresis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

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



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



Project Assignment 7

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

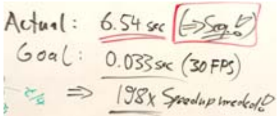
```
– Gaussian_Smooth                3.53 s
–   Gaussian_Kernel    0.00 s
–   BlurX              1.71 s
–   BlurY              1.82 s
– Derivative_X_Y                 0.48 s
– Magnitude_X_Y                  1.03 s
– Non_Max_Supp                   0.83 s
– Apply_Hysteresis               0.67 s
–     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

➢ This performance is far too slow for real-time video!

➢ Discussion: What options exist to speed this up?

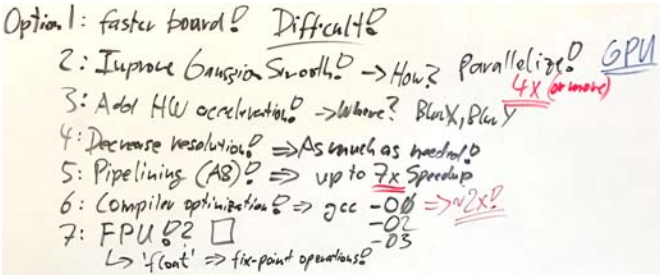
Project Assignment 7

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



Actual: 6.54 sec (⇒ Seq.!)
Goal: 0.033 sec (30 FPS)
⇒ 198x Speedup needed!

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



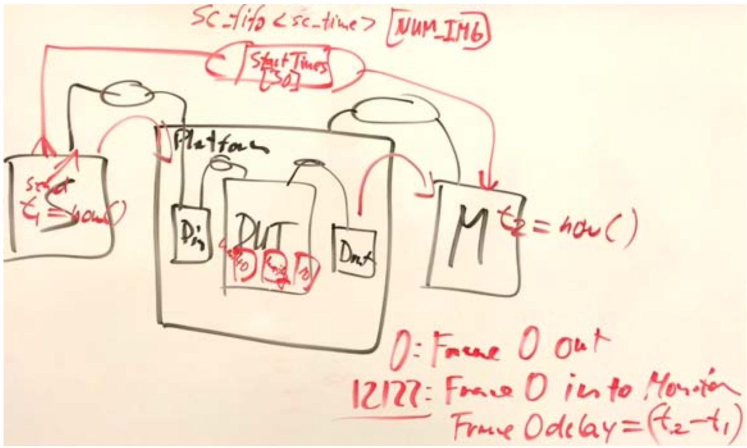
Option 1: faster board! Difficult!
2: Improve Gaussian Smooth! → How? Parallelize! GPU 4x (or more)
3: Add HW acceleration! → Where? BlurX, BlurY
4: Decrease resolution! ⇒ As much as needed!
5: Pipelining (A8)! ⇒ up to 7x Speedup
6: Compiler optimization! ⇒ gcc -O0 ⇒ ~2x!
-O2
-O3
7: FPU!? □
↳ 'float' ⇒ fix-point operations!

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

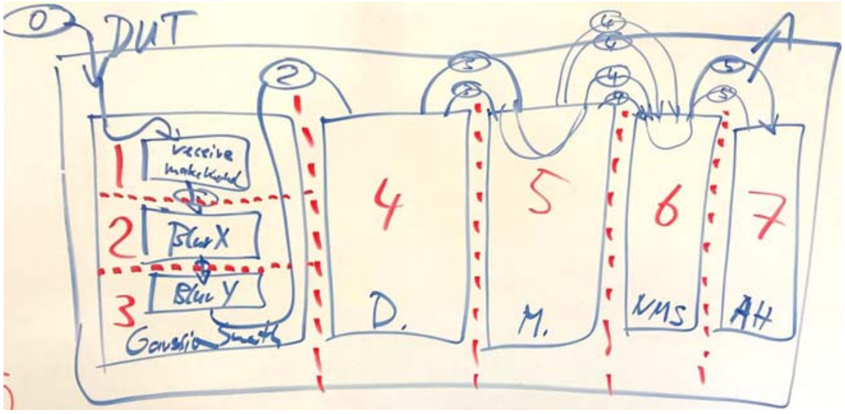
Project Assignment 8

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


sc_fifo<sc_time> [NUM_IMG]
StartTimes [30]
Platform
Start
t_1 = now()
Din
DUT
Dout
M
t_2 = now()
0: Frame 0 out
12122: Frame 0 into Monitor
Frame 0 delay = (t_2 − t_1)

Project Assignment 8

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



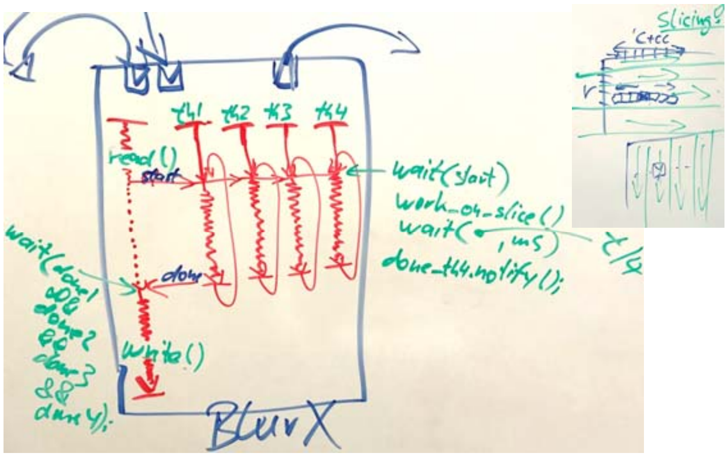
Project Assignment 8

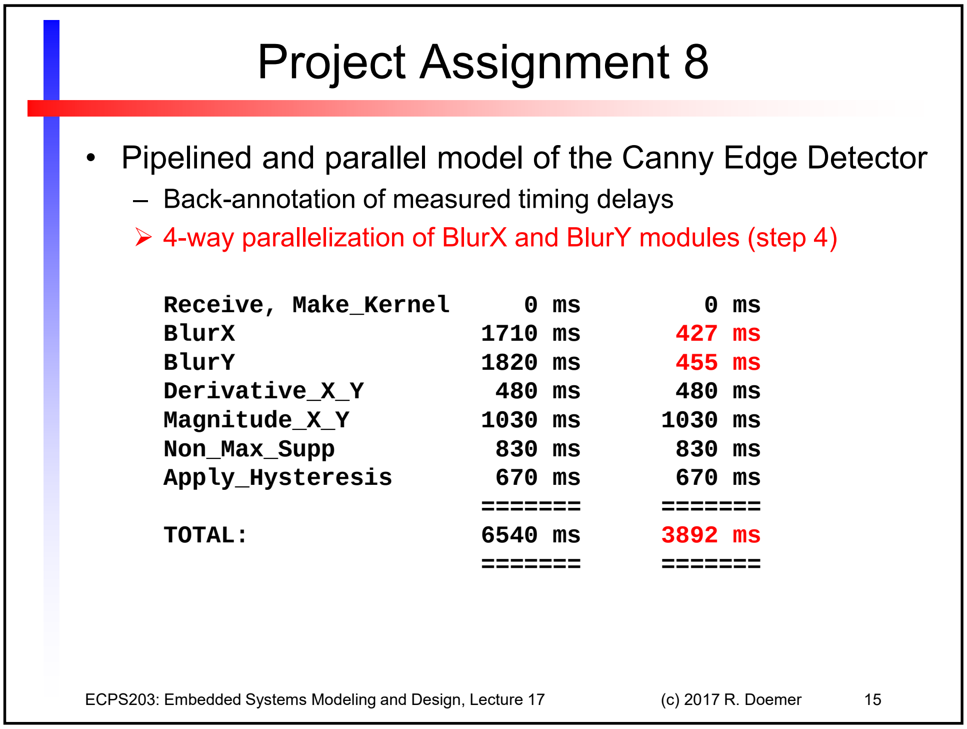
- Pipelined and parallel model of the Canny Edge Detector
  - Back-annotation of measured timing delays (step 2)

```
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)

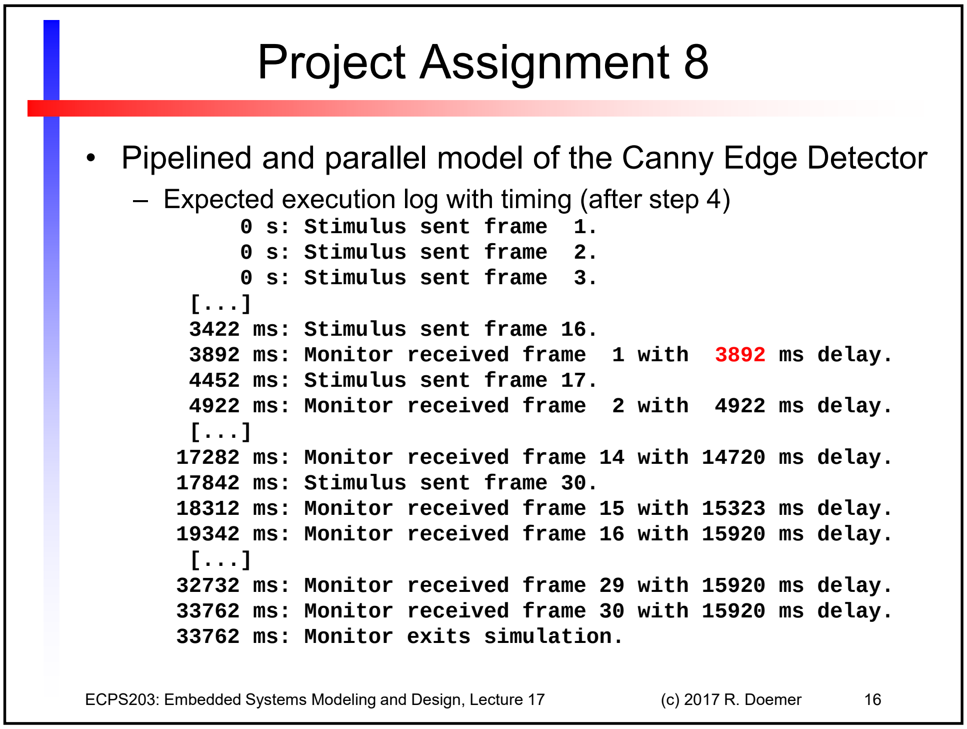




Project Assignment 8

- 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)

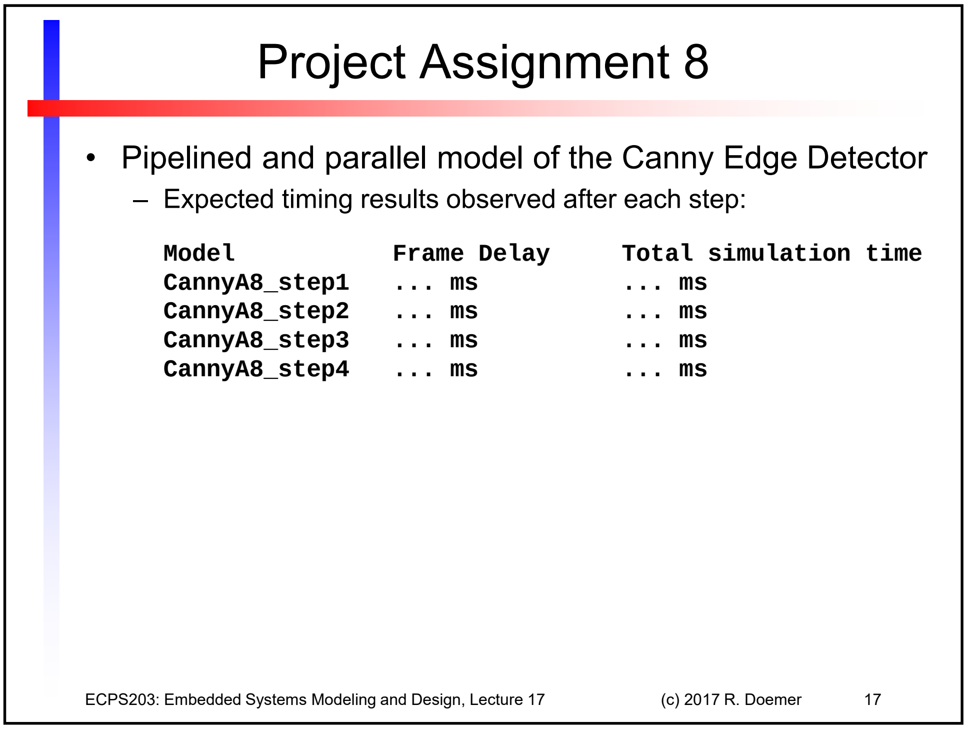
```
Receive, Make_Kernel      0 ms         0 ms
BlurX                  1710 ms       427 ms
BlurY                  1820 ms       455 ms
Derivative_X_Y          480 ms       480 ms
Magnitude_X_Y          1030 ms      1030 ms
Non_Max_Supp            830 ms       830 ms
Apply_Hysteresis        670 ms       670 ms
                       =======      =======
TOTAL:                 6540 ms      3892 ms
                       =======      =======
```



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
  - Expected timing results observed after each step:

Model	Frame Delay	Total simulation time
CannyA8_step1	... ms	... ms
CannyA8_step2	... ms	... ms
CannyA8_step3	... ms	... ms
CannyA8_step4	... ms	... ms