EECS 222: Embedded System Modeling Lecture 15

Rainer Dömer

doemer@uci.edu

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

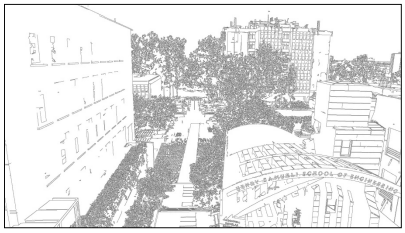
Lecture 15: Overview

- Project Discussion
  - Status and next steps
  - Profiling results
  - Back-annotation of timing estimates
  - Pipelining and parallelization of the DUT
  - ➢ Model development on the whiteboard
  - ➢ Discussion
- Homework Assignment 7
  - Pipelined/parallel model the Canny Edge Detector

EECS 222 Project

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

EngPlaza001.bmp

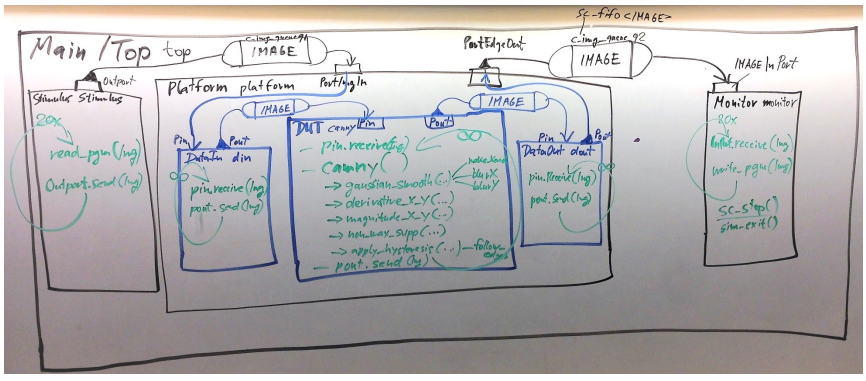


EngPlaza001_edges.pgm

  - 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

Homework Assignment 5

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



Homework Assignment 6

- Task: Performance Estimation of the Canny Edge Detector
  - Refine the structural hierarchy of the DUT block
  - Refine the structural hierarchy of the Gaussian Smooth block
  - Estimate the computational complexity of the DUT components
- Steps
  1. Create DUT structure: Gaussian Smooth, …, Apply Hysteresis
  2. Create Gaussian Smooth structure: Receive, Gauss, BlurX, BlurY
  3. Profile the application, obtain relative computational complexity
- Deliverables
  - **`Canny.sc`** or **`Canny.cpp`** (choose one!)
  - **`Canny.txt`** (with numerical values for block complexity!)
- Due
  - By next week: May 22, 2017, 12pm (noon!)

Homework 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
|------ c_img_queue q1
\------ c_img_queue q2
```

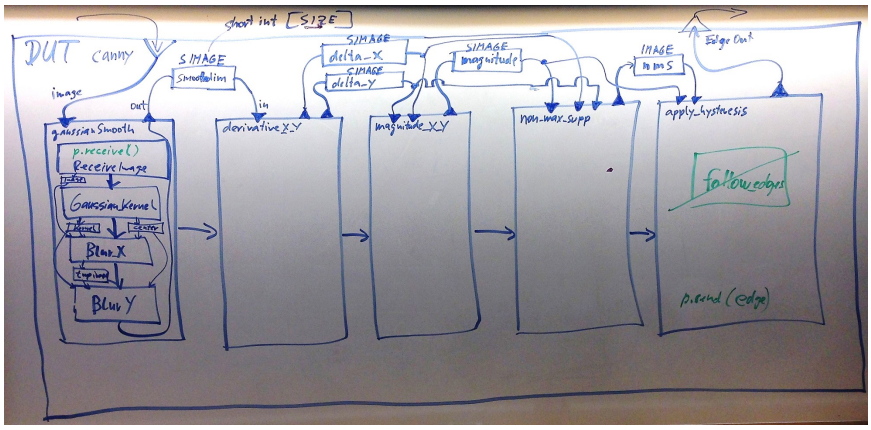
Homework Assignment 6

- Step 2: Refined hierarchy of the Gaussian Smooth block
  - Expected 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
```


Homework Assignment 6

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



Homework Assignment 6

- Step 3: Profile the application components
  - ➢ Performance Estimation of the Canny Edge Detector
  - – SpecC model profiling: Use SCE profiler
    - ➢ **`/opt/sce/bin/sce`**
    - Create a new project, import SpecC source code
    - Compile and simulate in SCE (with instrumentation)
    - Run the profiler, view raw computation graph for DUT components
  - – SystemC model profiling: Use GNU profiler
    - ➢ **`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 (**`self`**)

Homework Assignment 6

- Step 3: Profile the application components, obtain relative computational complexity
  - – Expected complexity comparison (in **`Canny.txt`**):
    Using GNU profiler on SystemC model:

```
Gaussian_Smooth                    53%
|------ Gaussian_Kernel    0%
|------ BlurX              43%
\------ BlurY              57%
Derivative_X_Y             10%
Magnitude_X_Y               5%
Non_Max_Supp               19%
Apply_Hysteresis           14%
                          100%
```

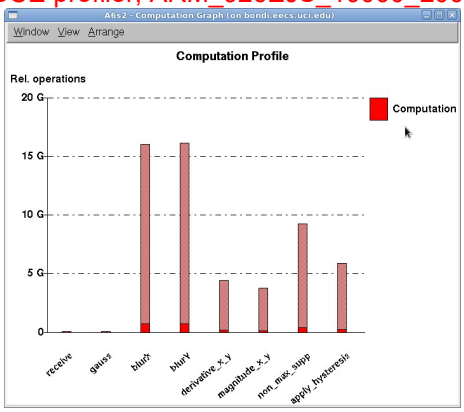
Homework Assignment 6

- Step 3: Profile the application components, obtain relative computational complexity
  - Expected complexity comparison (in **`Canny.txt`**):
    Using SCE profiler, ARM_926EJS_10000_20000_0 CPU:

```
Gaussian_Smooth                  813.5/1268.9=64%
|------ Receive_Image      0.0/854.2=  0%
|------ Gaussian_Kernel    0.0/854.2=  0%
|------ BlurX            416.8/854.2= 49%
\------ BlurY            437.4/854.2= 51%
Derivative_X_Y                    53.4/1268.9= 4%
Magnitude_X_Y                     53.4/1268.9= 4%
Non_Max_Supp                     272.0/1268.9=21%
Apply_Hysteresis                  76.5/1268.9= 6%
                                 1268.9/1268.9=100%
```

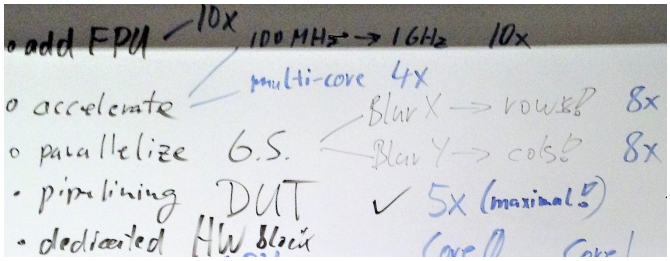

Homework Assignment 6

- Step 3: Profile the application components, obtain relative computational complexity
  - Expected complexity comparison (in **`Canny.txt`**):
    Using SCE profiler, ARM_926EJS_10000_20000_0 CPU



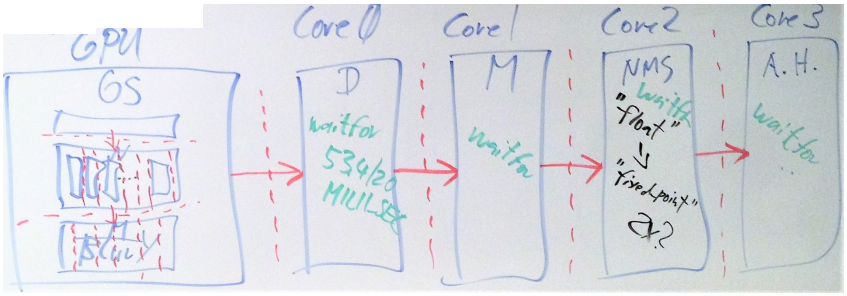
Homework Assignment 6, Next Steps

- Step 3: Profile the application components, obtain relative computational complexity
- Step 4: (discussion in class)
  - What about absolute timing? How long does it take?
  - Does it meet our real-time goals?
  - What can be done to improve the speed?



Homework Assignment 7

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



Homework Assignment 7

- Task: Pipelining and Parallelization of the Canny Model
  - Back-annotate estimated delays to observe timing in the 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. Pipeline the DUT into stages for each component
  4. Integrate Gaussian Smooth components into pipeline stages
  5. 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: By next week: May 31, 2017, 12pm (Wednesday, noon!)