EECS 222: Embedded System Modeling Lecture 20

Rainer Dömer

doemer@uci.edu

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

Lecture 20: Overview

- Course Administration
  - Instructor evaluation
  - Final exam
- EECS 222 Project
  - Review
  - Discussion
- Unified Modeling Language (UML)
  - Overview
  - Example Diagrams

Course Administration

- Final Course Evaluation
  - 9th through 10th week
  - Feb. 25, 2019, through March 17, 2019, 11:45pm
  - Open until next Sunday night
  - Online via EEE evaluation application
- Evaluation of Course and Instructor
  - Voluntary
  - Anonymous
  - Very valuable!

➢ Please help to improve this class!

  - Please spend 5 minutes!

Course Administration

- Final Exam
  - Allocated time
    - Thursday, March 21, 2019, 8:00-10:00am
  - Location
    - Regular classroom, SSTR 103
  - 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!

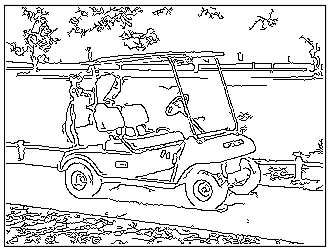
Project Review and Discussion

- 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
  - Structural refinement of DUT and Gaussian Smooth
- Project Assignment 7
  - 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:
    - 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
  4. Fix a bug and clean-up the source code
- Deliverables
  - Source code and text file: **`canny.c`**, **`canny.txt`**
- Due
  - Next week: January 16, 2019, 6pm

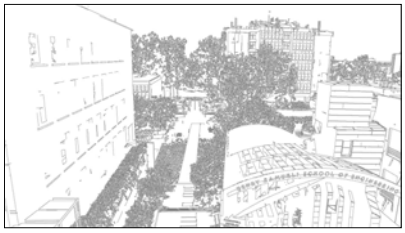
Project Assignment 4

- 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
  - Next week: February 6, 2019, 6pm

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

Project Assignment 5

- Task: Structural Test Bench Model
  - Convert the application to process a stream of video frames
  - Choose either SpecC or SystemC for modeling and simulation
  - Add test bench structure to the model from Assignment 4
- Steps
  1. Convert the application to process a stream of video frames
  2. Create test bench structure: Stimulus, Platform, Monitor
  3. Create platform structure: DataIn, DUT, DataOut
- Deliverables
  - **`Canny.sc`** or **`Canny.cpp`** (choose one!)
  - **`Canny.txt`**
- Due
  - Next week: February 13, 2019, 6pm

Project Assignment 5

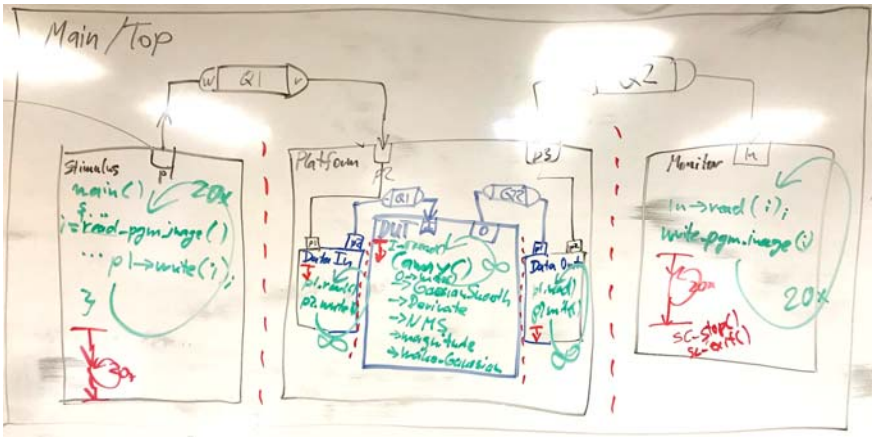
- Task: Structural Test Bench Model
  - Expected instance tree

```
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!

Project Assignment 5

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

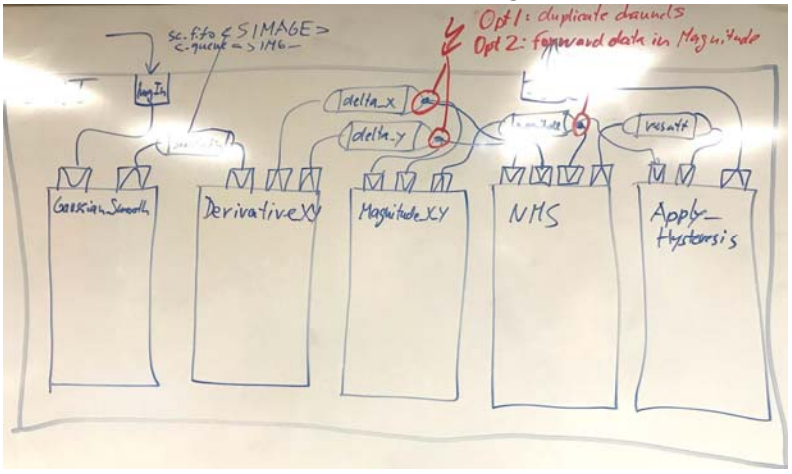


Project Assignment 6

- Task: Hierarchical DUT of the Canny Edge Detector
  - Refine the structural hierarchy of the DUT block
  - Refine the structural hierarchy of the Gaussian Smooth block
- Steps
  1. Refine the DUT structure
     - Gaussian Smooth, Derivative, …, Apply Hysteresis
  2. Refine the Gaussian Smooth structure
     - Receive Image, Gaussian Kernel, BlurX, BlurY
  3. Visualize the structural hierarchy of the model
- Deliverables
  - **`Canny.sc`** or **`Canny.cpp`** (choose one!)
  - **`Canny.txt`**
- Due: February 20, 2019, 6pm

Project Assignment 6

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



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
```

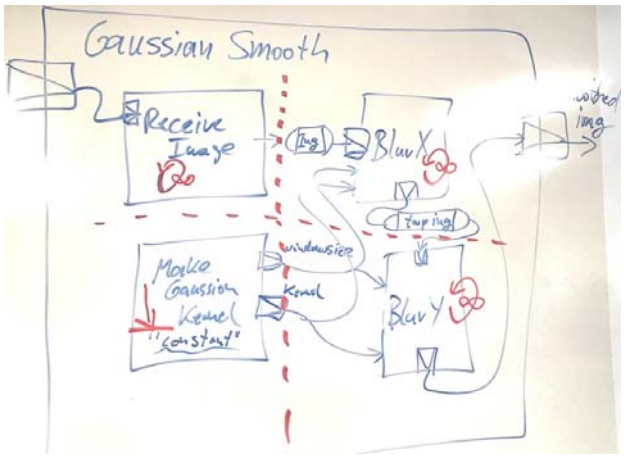

Project Assignment 6

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

Project Assignment 6

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



Project Assignment 7

- Task: Performance Estimation of Canny Edge Detector
  - Profiling to estimate relative computational complexity
  - Instrumentation to measure absolute timing as reference
- Steps
  1. Profile the application, identify performance bottle-necks
     - SpecC: Use SCE profiling tools
     - SystemC: Use GNU profiling tools
  2. Instrument the application, measure timing on reference platform
- Deliverables
  - **`Canny.sc`** or **`Canny.cpp`** (choose one!)
  - **`Canny.txt`** (with numerical values for obtained results)
- Due
  - Next week: February 27, 2019, 6pm

Project Assignment 7

- Step 1: 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, analyze tables and charts
  - – 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

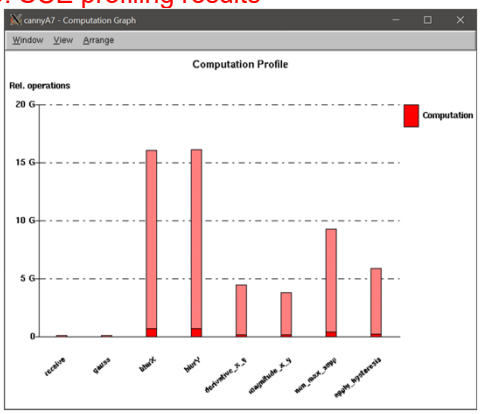
Project Assignment 7

- Step 1: Profile the application components, obtain relative computational complexity
  - – Expected complexity comparison (in **`Canny.txt`**):
    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 1: Profile the application components, obtain relative computational complexity
  - Expected complexity comparison (in **`Canny.txt`**):
    SpecC: SCE profiling results



Project Assignment 7

- Step 1: Profile the application components, obtain relative computational complexity
  - Expected complexity comparison (in **`Canny.txt`**):
    SystemC: GPROF profiling results

```
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%
```

Project Assignment 7

- Step 2: Instrument the application components
  - ➢ Performance Measurement of the Canny Edge Detector
  - ➢ 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

Project Assignment 7

- Step 2: Instrument the application components, obtain absolute timing on server platform
  - – Expected complexity comparison (also in **`Canny.txt`**):
    SpecC: Timing measurement results

```
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%
```

Project Assignment 7

- Step 2: Instrument the application components, obtain absolute timing on server platform
  - Expected complexity comparison (also in **`Canny.txt`**):
    SystemC: Timing measurement results

```
Gaussian_Smooth                  10.82s 57.8%
|------ Receive_Image     0.00s   0.0%
|------ Gaussian_Kernel   0.00s   0.0%
|------ BlurX             5.15s  27.5%
\------ BlurY             5.67s  30.3%
Derivative_X_Y                    1.93s  10.3%
Magnitude_X_Y                     1.49s   8.0%
Non_Max_Supp                      2.09s  11.2%
Apply_Hysteresis                  2.38s  12.7%
                                         100%
```

EECS222: Embedded System Modeling, Lecture 20 (c) 2019 R. Doemer 25

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 simulation 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: Next week: March 6, 2019, 6pm

EECS222: Embedded System Modeling, Lecture 20 (c) 2019 R. Doemer 26

Project Assignment 8

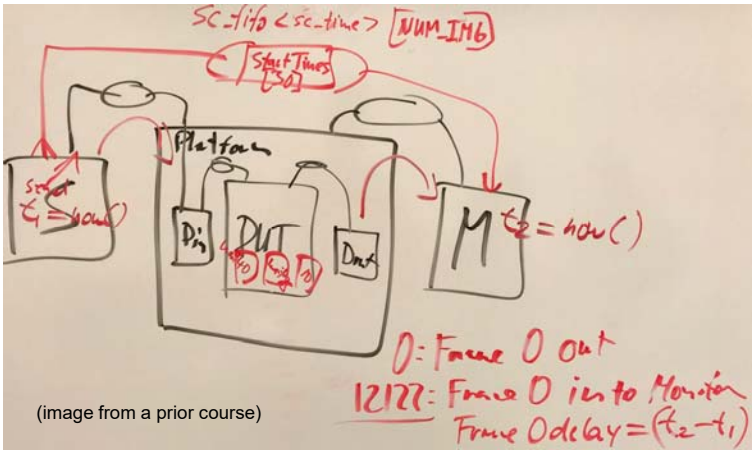
- Step 1: Logging of simulation time and frame delay
  - Expected execution log with timing instrumentation

```
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 20 (c) 2019 R. Doemer 27

Project Assignment 8

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



(image from a prior course)

EECS222: Embedded System Modeling, Lecture 20 (c) 2019 R. Doemer 28

Project Assignment 8

- Step 2: Back-annotate timing in DUT components
  - Insert wait-for-time statements into your model
  - Measured timing on Raspberry Pi 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
```

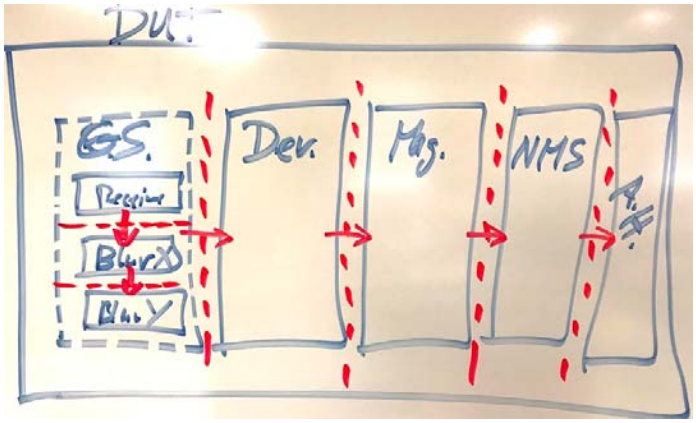

Project Assignment 8

- 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.
```

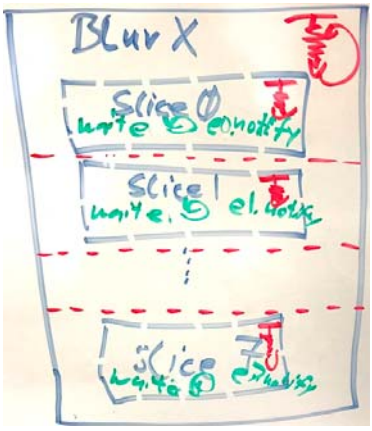
Project Assignment 8

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



Project Assignment 8

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



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
  - 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

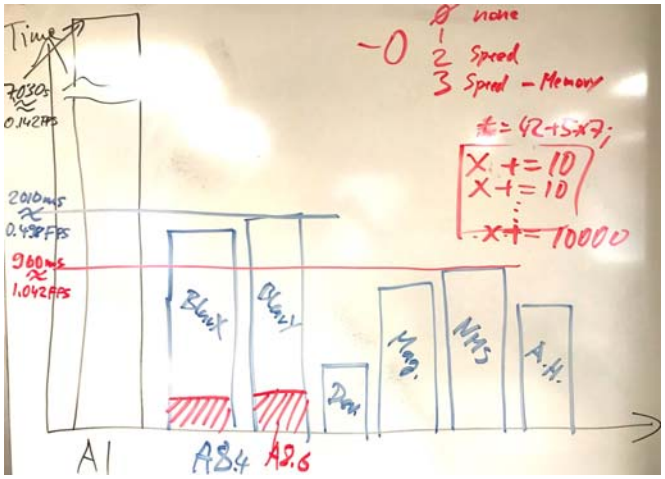
Model	Frame Delay	Throughput	Total time
CannyA8_step1	0 ms	n/a	0 ms
CannyA8_step2	17340 ms	n/a	45220 ms
CannyA8_step3	17340 ms	0.498 FPS	45220 ms
CannyA8_step4	17340 ms	0.498 FPS	45220 ms
CannyA8_step5	18900 ms	0.498 FPS	45220 ms
CannyA8_step6	12260 ms	1.042 FPS	21866 ms

Project Discussion

- Discussion Questions
  - Does the timing meet our real-time goals? No.
  - How far off is it? 7030/33 = 213x
  - What can be done to improve the speed?
    - Pipelining A8, steps 4 and 5
    - Parallelization A8, step 6
    - Hardware optimizations GPU or ASIC for BlurX, BlurY
    - Software optimizations A9, steps 1, 2, and 3
    - Application adjustments

Project Discussion

- Model Performance Overview
  - Discussion on the whiteboard



Project Assignment 9

- Task: Throughput optimization of Canny Edge Decoder
  - Utilize compiler optimizations
  - Replace floating-point with fixed-point arithmetic
- Steps
  1. Turn on compiler optimizations, measure speedup per block
  2. Apply speedup to back-annotated timing (overall 2.5x)
  3. 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:
  - Next week: March 13, 2019, 6pm

Project Assignment 9

- Deliverables
  - Speed-up values observed for each block: SpecC Model

```
T1  =  0.00ms / 0.00ms = n/a
T2  =  2.97ms / 0.97ms = 3.06
T3  =  3.86ms / 1.05ms = 3.68
T4  =  1.12ms / 0.39ms = 2.87
T5  =  1.04ms / 0.85ms = 1.22
T6  =  2.08ms / 1.32ms = 1.58
T7  =  2.02ms / 0.82ms = 2.46
Tot = 13.09ms / 5.40ms = 2.42
```

  - Timing observed after each step: SpecC Model

Model	Frame Delay	Throughput	Total time
CannyA9_step2	3752 ms	2.604 FPS	8746 ms
CannyA9_step3	3572 ms	2.747 FPS	8346 ms

Project Assignment 9

- Deliverables
  - Speed-up values observed for each block: SystemC Model

```
T1  =  0.00ms / 0.00ms = n/a
T2  =  5.15ms / 0.91ms = 5.66
T3  =  5.67ms / 1.19ms = 4.76
T4  =  1.93ms / 0.32ms = 6.03
T5  =  1.49ms / 0.85ms = 1.75
T6  =  2.09ms / 1.21ms = 1.73
T7  =  2.38ms / 0.79ms = 3.01
Tot = 18.71ms / 5.27ms = 3.55
```

  - Timing observed after each step: SystemC Model

Model	Frame Delay	Throughput	Total time
CannyA9_step2	5428 ms	2.604 FPS	8746500 us
CannyA9_step3	5020 ms	2.747 FPS	8338500 us

Project Discussion

- Discussion Questions
  - Does the timing meet our real-time goals? No.
  - How far off is it? 7030/33 = 213x
  - What can be done to improve the speed?

  - Pipelining — A8, steps 4 and 5
  - Parallelization — A8, step 6
  - Hardware optimizations — GPU or ASIC for BlurX, BlurY
  - Software optimizations — A9, steps 1, 2, and 3
  - Application adjustments — Discussion, future work
    - Keep improving pipeline bottlenecks
    - Accept lower image quality (i.e. fixed-point calculations)
    - Accept lower frame rate
    - Accept lower image resolution
    - …

Unified Modeling Language (UML)

- Goals
  - Raising the Level of Abstraction
  - Modeling of software applications
    - ➢ before coding!
  - Specification of software architecture
  - High-level description of software architecture to enable
    - scalability
    - security
    - robustness
    - maintenance
    - extendability
    - code reuse
  - Model Driven Architecture (MDA)
- Status
  - UML 2.0: Modeling Language in Software Engineering
  - standardized by OMG (Object Management Group) in 1997
  - standardized by ISO (Intl. Org. for Standardization) in 2005

Unified Modeling Language (UML)

- What is UML?
  - Graphical representation of …
    - Software architecture
    - Software structure
    - Software behavior
    - Object relations
    - ...
  - 13 standard diagrams
    - Specification
    - Design
    - Documentation
  - ➢ Not executable!
  - Commercial tools available for …
    - Graphical capture
    - Editing
    - Code generation (template code)

Unified Modeling Language (UML)

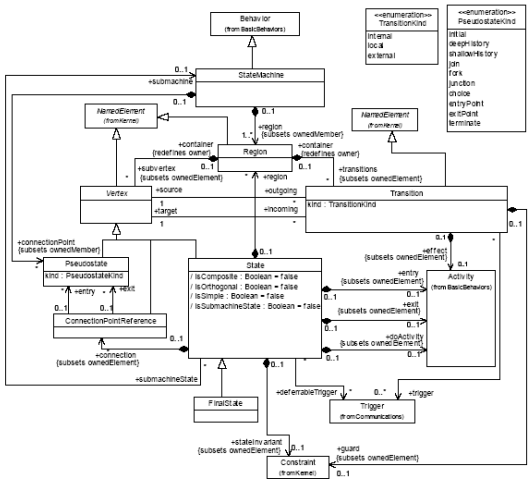
- UML Standard Diagrams
  - Structure Diagrams
    - Class Diagram
    - Object Diagram
    - Component Diagram
    - Composite Structure Diagram
    - Package Diagram
    - Deployment Diagram
  - Behavior Diagrams
    - Use Case Diagram
    - Activity Diagram
    - State Machine Diagram
  - Interaction Diagrams
    - Sequence Diagram
    - Communication Diagram
    - Timing Diagram
    - Interaction Overview Diagram

Unified Modeling Language (UML)

- UML Resources
  - Online Documents
    - Object Management Group (OMG)
      - **`www.uml.org`**
  - Online Tutorials
    - **`https://www.tutorialspoint.com/uml/`**
    - **`http://www.sparxsystems.com/uml-tutorial.html`**
  - Invited Talk at UCI in 2004
    - Dr. Wolfgang Mueller, C-LAB, Paderborn, Germany
    - Source of the following UML diagram examples

Unified Modeling Language (UML)

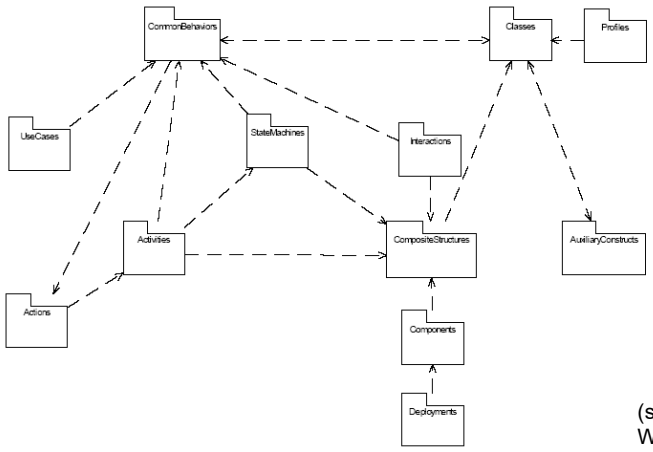
- Class Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

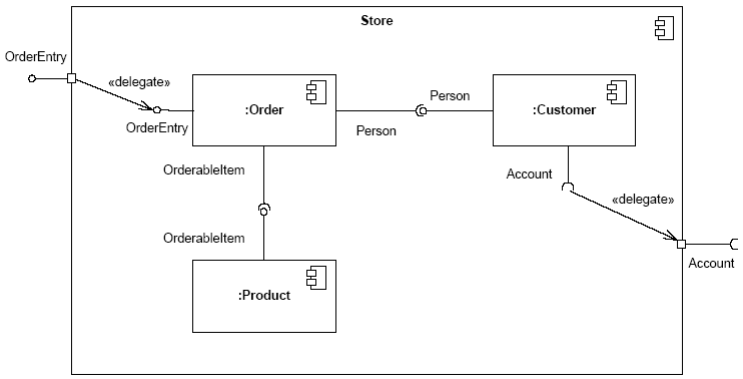
- Package Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

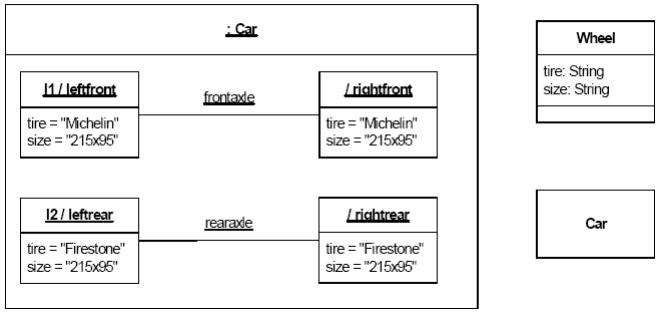
- Component Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

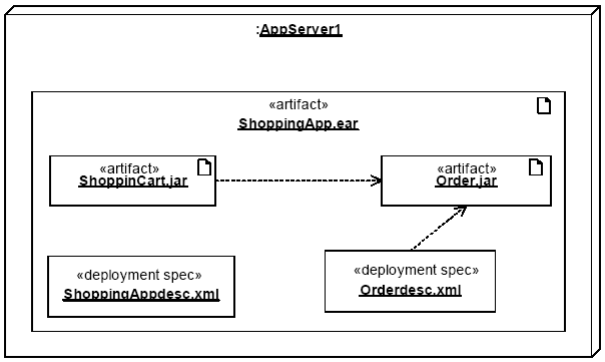
- Composite Structure Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

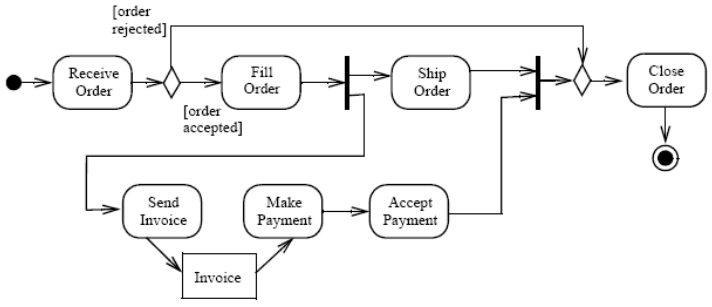
- Deployment Diagram Example



(source:
W. Mueller)

Unified Modeling Language (UML)

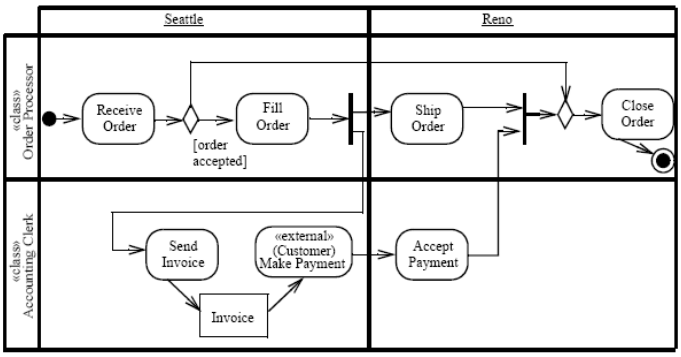
- Activity Diagram Example



(source:
W. Mueller)

Unified Modeling Language (UML)

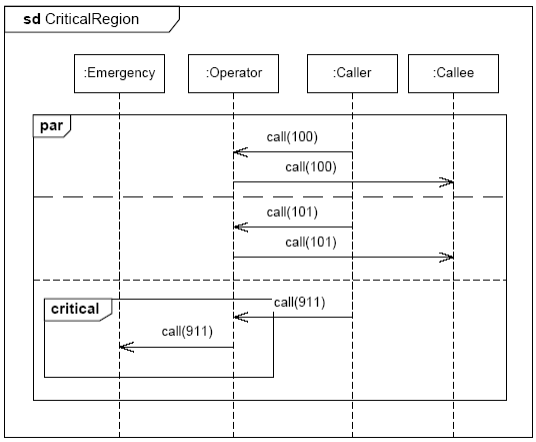
- Activity Diagram Example with “swim lanes”



(source: W. Mueller)

Unified Modeling Language (UML)

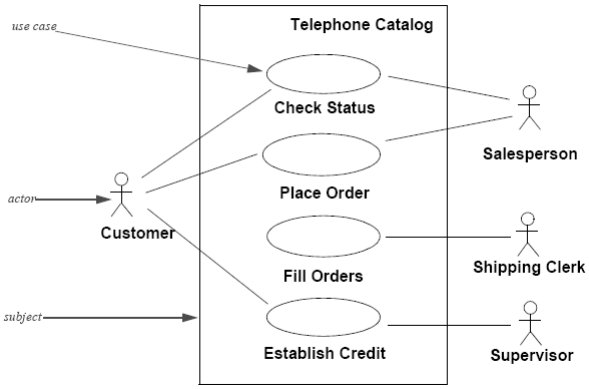
- Sequence Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

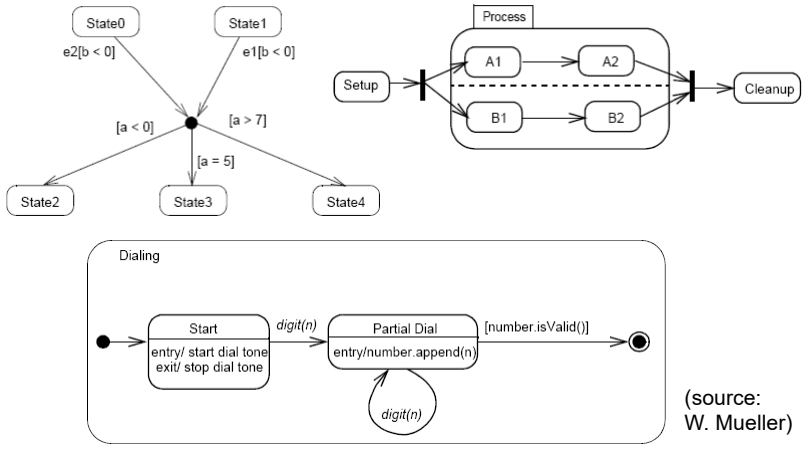
- Use Case Diagram Examples



(source: W. Mueller)

Unified Modeling Language (UML)

- State Machine Diagram Examples



(source: W. Mueller)