ECPS 203
Embedded Systems Modeling and Design
Lecture 20

Rainer Dömer

doemer@uci.edu

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

Lecture 20: Overview

- Course Administration
  - Final course evaluation
- Unified Modeling Language (UML)
  - Overview
  - Example Diagrams
- Project Discussion
  - A7: Performance measurement on prototyping board
  - A8: Pipelined and parallel model with back-annotated timing
  - A9: Throughput optimization by pipeline load balancing
- Final Report
  - Final exam
  - Grading criteria

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!

Unified Modeling Language (UML)

- Goals
  - Raising the level of abstraction
  - Modeling of software applications
    - ➢ before coding!
  - Specification of software architecture
  - Enabling
    - 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)

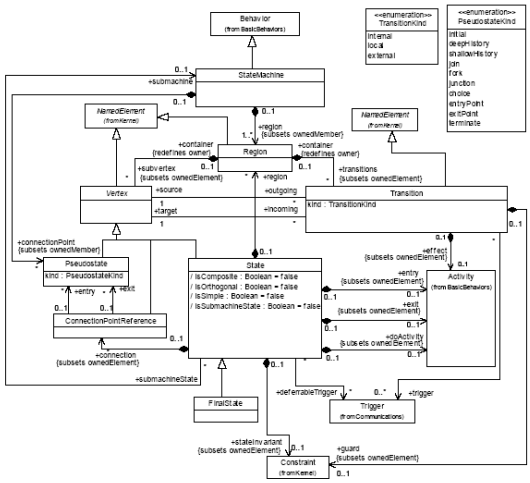
- UML Standard Diagrams
  - Structure Diagrams
    - Class Diagram
    - Object Diagram
    - Component Diagram
    - Composite Structure Diagram
    - Package Diagram
    - Deployment Diagram
  - Behavior Diagrams
    - Use Case Diagram
    - Activity Diagram
    - State Machine Diagram
  - Interaction 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)

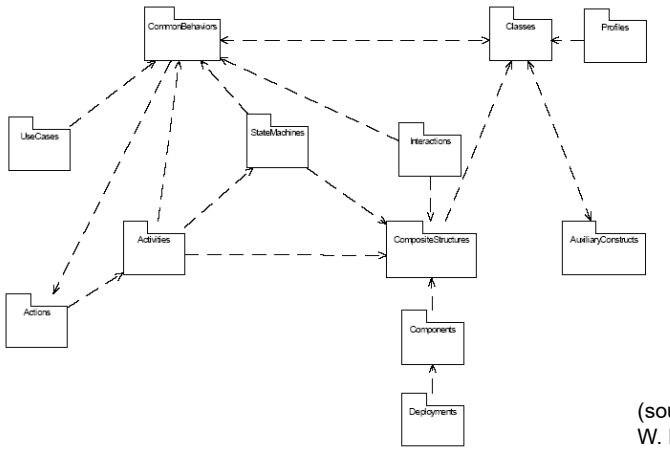
- Class Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

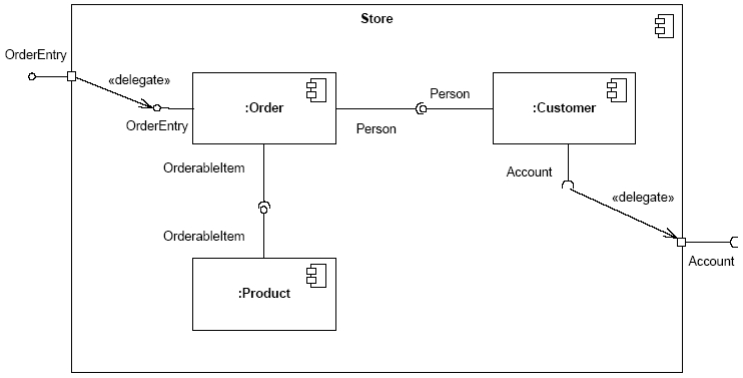
- Package Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

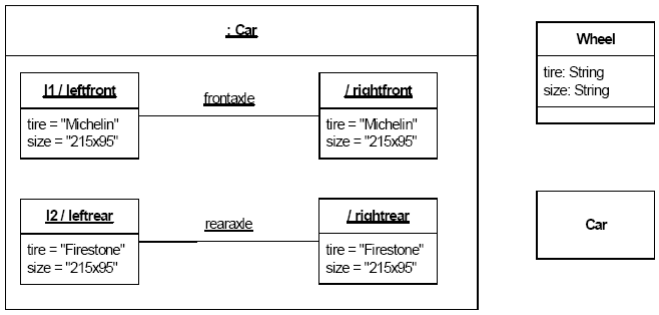
- Component Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

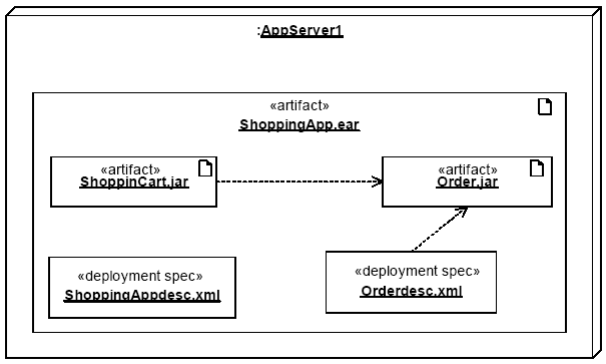
- Composite Structure Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

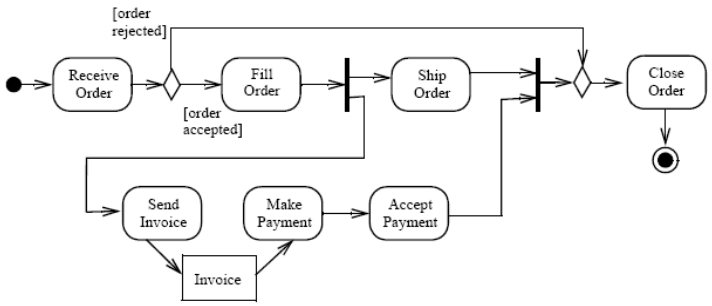
- Deployment Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

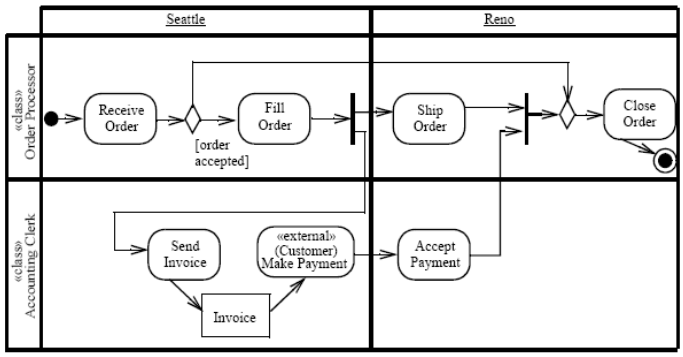
- Activity Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

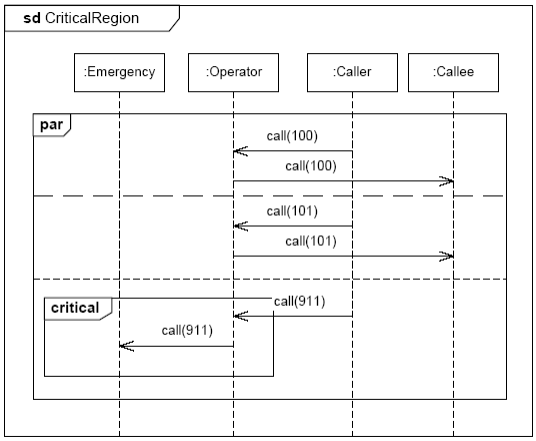
- Activity Diagram Example with “swim lanes”



(source: W. Mueller)

Unified Modeling Language (UML)

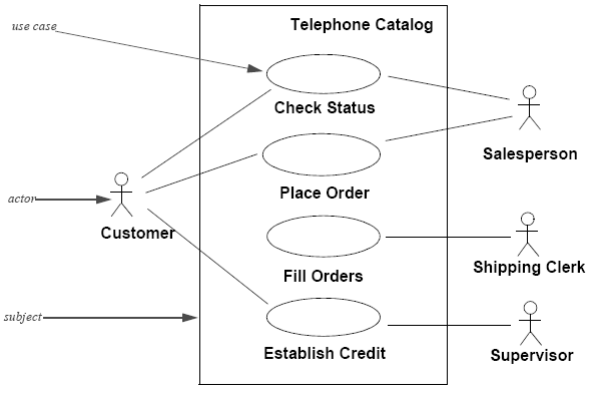
- Sequence Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

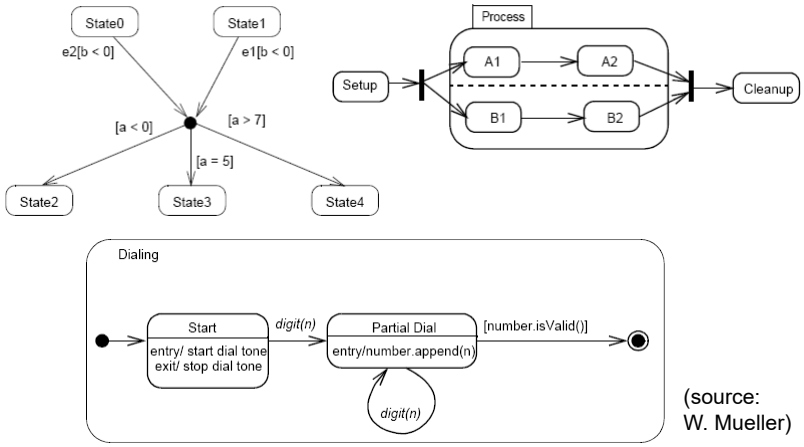
- Use Case Diagram Examples



(source: W. Mueller)

Unified Modeling Language (UML)

- State Machine Diagram Examples



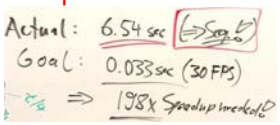
(source: W. Mueller)

Project Assignment 7

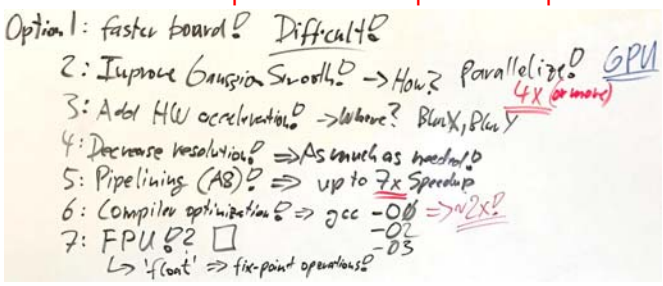
- Task: Performance measurement on prototyping board
  - Run C++ model of Canny Edge Detector on Raspberry Pi
  - Obtain absolute timing measurements of Canny functions
- Steps
  1. Prepare the prototyping board with Raspbian operating system
  2. Upload **`Canny.cpp`** from A4 and compile it
  3. Instrument 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
  - **`TOTAL 6.54 seconds`**
  - ➢ This performance is far too slow for real-time video!



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

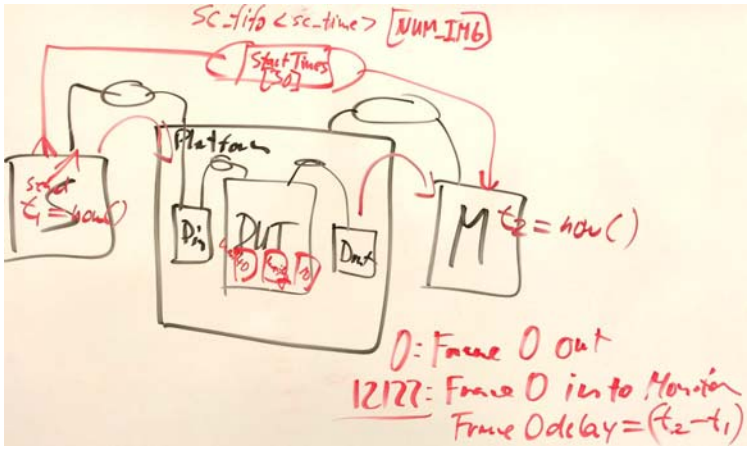


Project Assignment 8

- 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 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

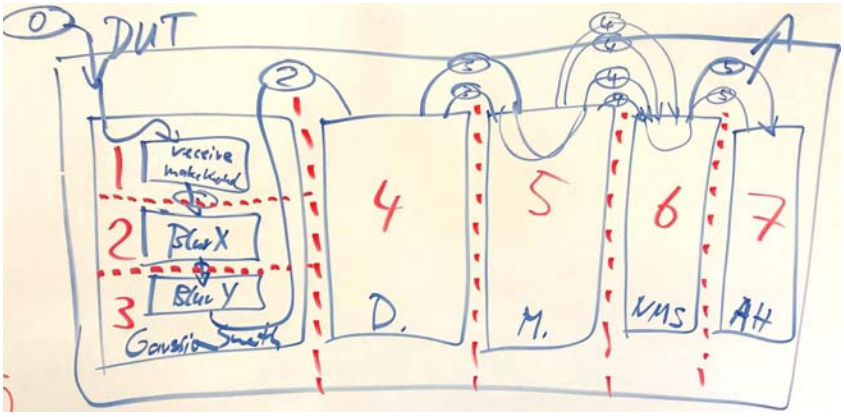
Project Assignment 8

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



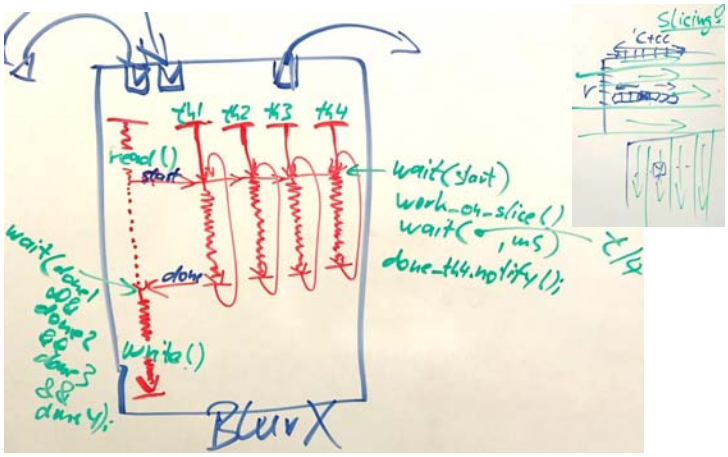
Project Assignment 8

- 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
  - 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)

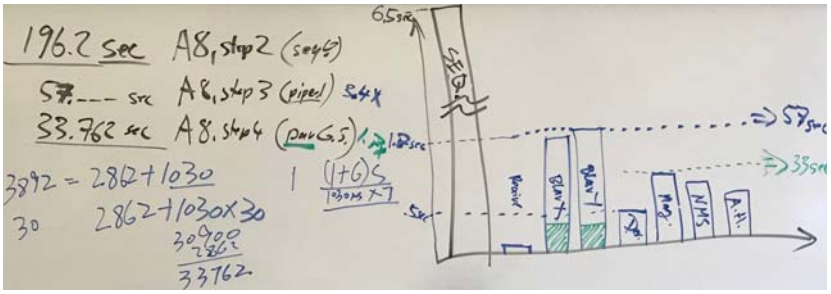
```
      0 s: Stimulus sent frame  1.
      0 s: Stimulus sent frame  2.
      0 s: Stimulus sent frame  3.
 [...]
 3422 ms: Stimulus sent frame 16.
 3892 ms: Monitor received frame  1 with  3892 ms delay.
 4452 ms: Stimulus sent frame 17.
 4922 ms: Monitor received frame  2 with  4922 ms delay.
 [...]
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

- 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)

Project Assignment 8

- Discussion of Performance



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

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);
```

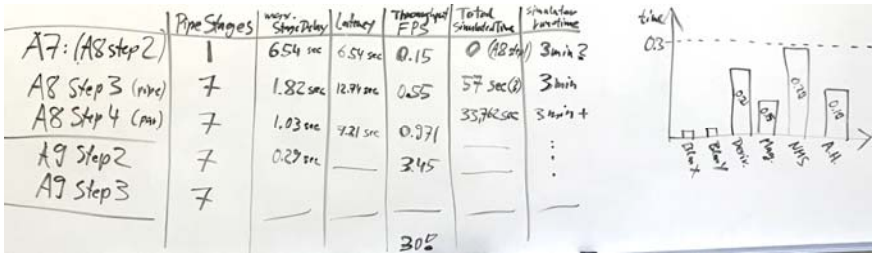
  - … with this code

```
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`**)

Project Assignment 9

- Discussion of Throughput Optimization



Final Project Report

- Final Report (in lieu of Final Exam)
  - Allocated time and room for final exam
    - Monday, December 11, 8:00-10:00am (DBH 1420)
    - ➢ *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`**
  - Soft deadline: Draft report (for early feedback)
    - Monday, December 11, 2017, 10am
  - Hard deadline: Final report (graded!)
    - Wednesday, December 13, 2017, 6pm

Final Project Report

- Technical Report about the Course Project
  - Title
    - *Specification and 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 the project assignments 1 through 9
    - Focus on the reasoning and the optimization results
  - 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
    3) Accuracy of results and completeness
    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 are the timing relations in A7 different from A6?
    - Q5: Why did we choose to parallelize Gaussian Smooth in A8?
    - Q6: Why does the simulator run-time not improve in A8?
    - Q7: How can you achieve real-time video for the end user?