

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

- Course Administration
 - Final course evaluation
 - Final exam
- Project Discussion
 - A7: Performance measurement on prototyping board
 - A8: Back-annotation of timing estimates into SystemC model
 - A8: Pipelining and parallelization of the DUT module
 - A9: Throughput optimization by pipeline load balancing
- Unified Modeling Language (UML)
 - Overview
 - Example Diagrams

Course Administration

- Final Course Evaluation
 - Open until end of 10th week (Sunday night)
 - Nov. 25, 2019, through Dec. 8, 2019, 11:45pm
 - 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!

ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

3

Course Administration

- Final Exam
 - Allocated time
 - Monday, December 9, 2019, 8:00-10:00am
 - Location
 - Regular classroom, DBH 1200
 - 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!

ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

4

Project Assignment 7

- Task: Performance measurement on prototyping board
 - Measured delays on Raspberry Pi 3 (in Canny.txt):

Gaussian_Smooth	3.53 sec
----- Gaussian_Kernel	0.00 sec
----- BlurX	1.71 sec
\----- BlurY	1.82 sec
Derivative_X_Y	0.48 sec
Magnitude_X_Y	1.03 sec
Non_Max_Supp	0.83 sec
Apply_Hysteresis	<u>0.67 sec</u>
TOTAL	6.54 sec

- 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 17 (c) 2019 R. Doemer 5

Project Assignment 7

- Discussion: Measured delays on Raspberry Pi **3 4**
 - TOTAL **6.54 seconds**
 - This performance is far too slow for real-time video!
 - Discussion: What options exist to speed this up?

Optimization Options

- 30 Rpi's, distributed
- Compilation, Optimize -O2
- Pipelining, upto 7x
- Parallelization

2704x 1500 4.056 Mpix	1800x 900 1.4 Mpix	852x480 smaller img	0.4 Mpix
GS	K	0	0
0.42	BX	0.15	0.039 sec
0.63	BY	0.21	0.05 sec
0.26 Dev		0.06	0.013 sec
0.17 Mag		0.05	0.016 sec
0.29 NMS		0.09	0.03 sec
0.29 A.H.		0.09	0.032 sec
<u>2.08</u>		<u>0.67</u>	<u>0.18 total sec</u>
Goal: 30FPS Δ 0.033 sec			
6.54 sec is 200 off			

ECPS203: Embedded Systems Modeling and Design, Lecture 17 (c) 2019 R. Doemer 6

Project Assignment 7

- Discussion: Performance Improvement Options

Canny Architecture

Core 0	Dev. (+6ausker)
Core 1	Magn.
Core 2	NMS
Core 3	A.H.
GPU	BlurX, BlurY

Optimizations

1. Compiler Opt.	2-3x
2. Pipelining	upto 7x
3. Parallelization	Nx ←
4. CPU, faster clock	~2x
5. GPU (BlurX, Y)	Nx ←
6. FPU (← SW → HW) → 9.2	Float vs Fixed 4x
7. Lower Resolution	
8. Lower FPS	2x

ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

7

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 simulated time to observe frame delay
 2. Back-annotate estimated timing into DUT components
 3. Improve test bench to observe frame throughput
 4. Pipeline the DUT into a sequence of 7 stages with buffer size 1
 5. Slice the BlurX and BlurY modules into 4 parallel threads
- Deliverables
 - **Canny.cpp**: pipelined and parallelized SystemC model
 - **Canny.txt**: table of observed frame delays and throughput
- Due: Wednesday, November 27, 2019, 6pm

ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

8

Project Assignment 8

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



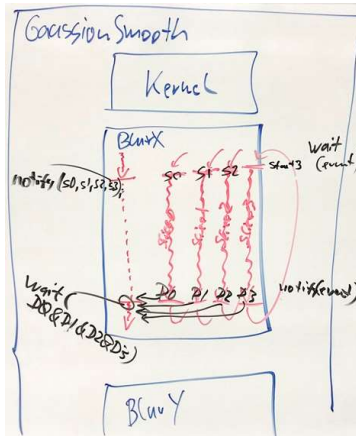
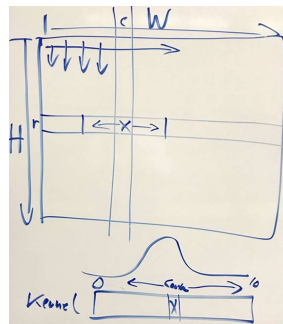
ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

9

Project Assignment 8

- Pipelined and parallel model of the Canny Edge Detector
 - Discussion on whiteboard: Parallel BlurX, BlurY functions (step 5)



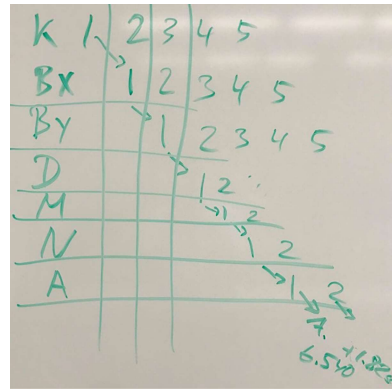
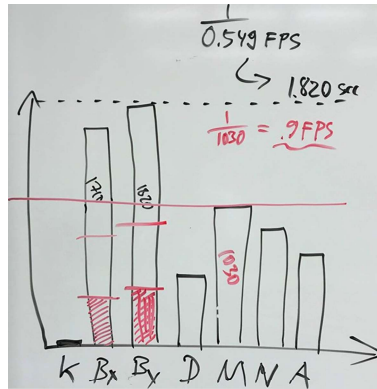
ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

10

Project Assignment 8

- Pipelined and parallel model of the Canny Edge Detector
 - Discussion on whiteboard: Parallel BlurX, BlurY functions (step 5)



ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

11

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 5)

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
	=====	=====
Throughput:	1/1820ms	1/1030ms
	0.549 FPS	0.971 FPS

ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

12

Project Assignment 8

- Pipelined and parallel model of the Canny Edge Detector
 - Expected execution log with timing (after step 5)

```

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.
[...]
30672 ms: Monitor received frame 27 with 15920 ms delay.
30672 ms: 1.030 seconds after previous frame, 0.971 FPS.
31702 ms: Monitor received frame 28 with 15920 ms delay.
31702 ms: 1.030 seconds after previous frame, 0.971 FPS.
32732 ms: Monitor received frame 29 with 15920 ms delay.
32732 ms: 1.030 seconds after previous frame, 0.971 FPS.
33762 ms: Monitor received frame 30 with 15920 ms delay.
33762 ms: Monitor exits simulation.

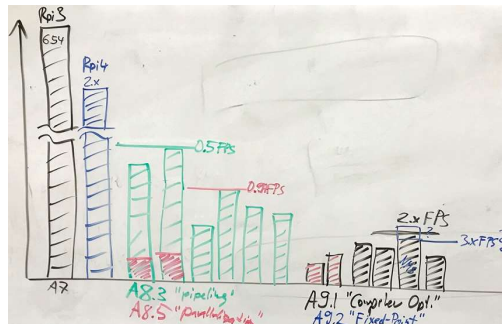
```

Project Assignment 9

- Task: Throughput optimization by pipeline load balancing
 - Optimize the bottleneck stages to improve throughput
- Steps
 1. Apply compiler optimizations for maximum execution speed
 - Try GNU compiler options `-O2`, `-O3`, and others
 - Back-annotate the best performance you can find
 2. Consider fixed-point instead of floating-point arithmetic
 - Use fixed-point arithmetic in NMS module
 - Evaluate the trade-off between speed and accuracy
- Deliverables
 - `Canny.cpp` (final SystemC model with best performance)
 - `Canny.txt` (performance table, design decisions, reasoning)
- Due
 - Wednesday, December 4, 2019, 6pm

Project Assignment 9

- Step 1: Apply compiler optimizations to improve throughput
 - Try GNU compiler options
 - -O2, -O3, and others
 - Back-annotate the best performance you can find
 - Improvement will be different for different modules



ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

15

Project Assignment 9

- Step 2: Consider fixed-point calculations instead of floating-point arithmetic
 - Focus on **Non_Max_Supp** module only
 - Convert `float` type variables to `int` types
 - Replace these lines of 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
 - Measure and evaluate the image quality (**ImageDiff**)

ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

16

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

ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

17

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)

ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

18

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

ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

19

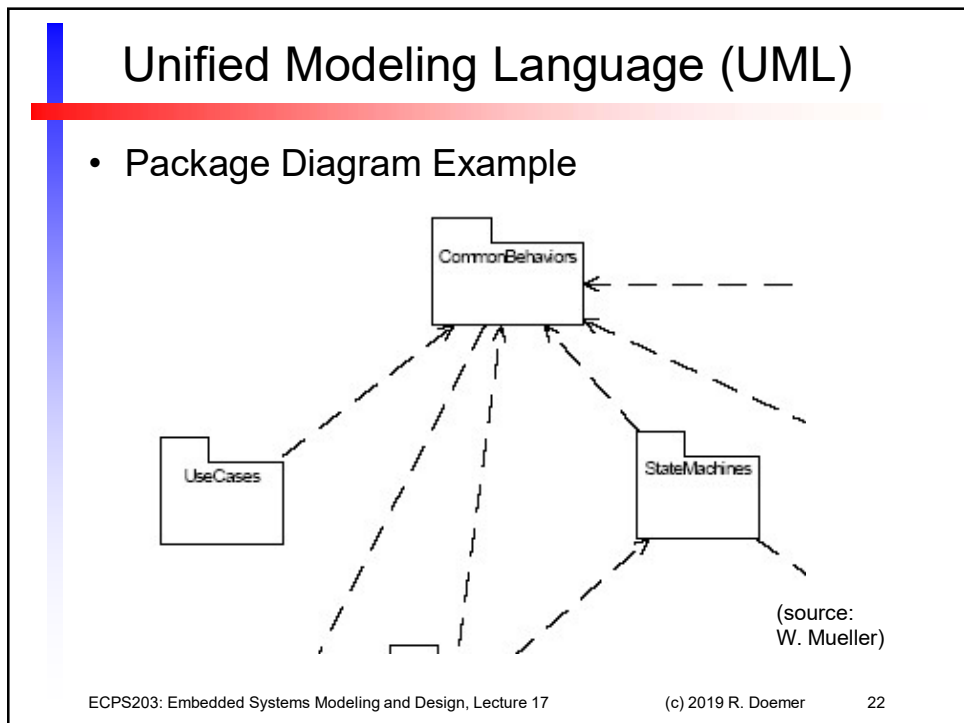
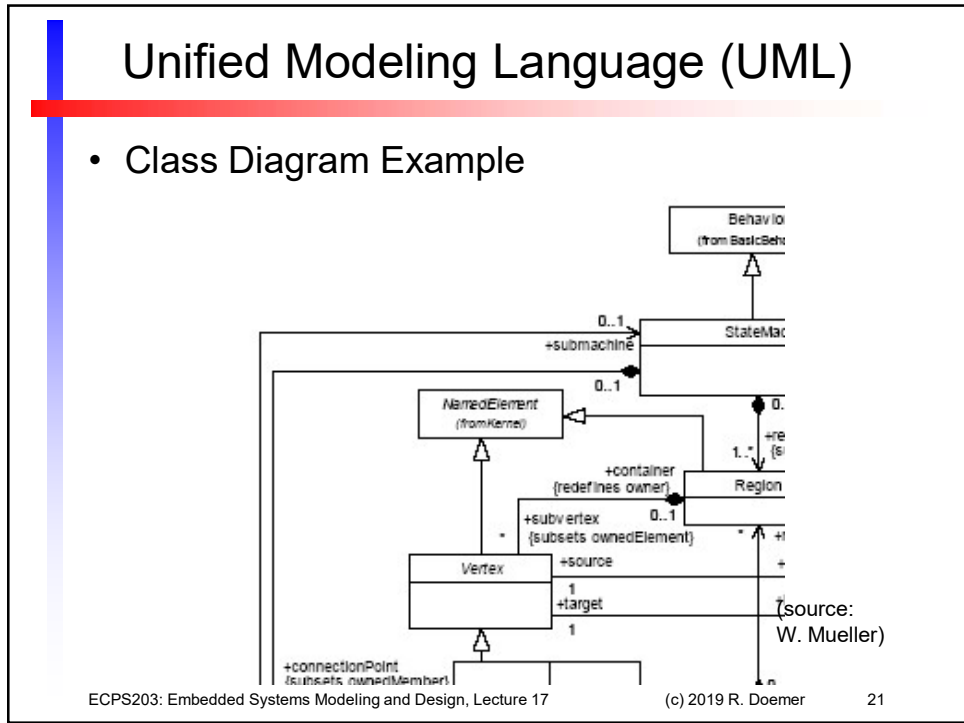
Unified Modeling Language (UML)

- 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
 - Source of the following UML diagram examples
 - Dr. Wolfgang Mueller, C-LAB, Paderborn, Germany

ECPS203: Embedded Systems Modeling and Design, Lecture 17

(c) 2019 R. Doemer

20



Unified Modeling Language (UML)

- Component Diagram Example

(source: W. Mueller)

ECPS203: Embedded Systems Modeling and Design, Lecture 17 (c) 2019 R. Doemer 23

Unified Modeling Language (UML)

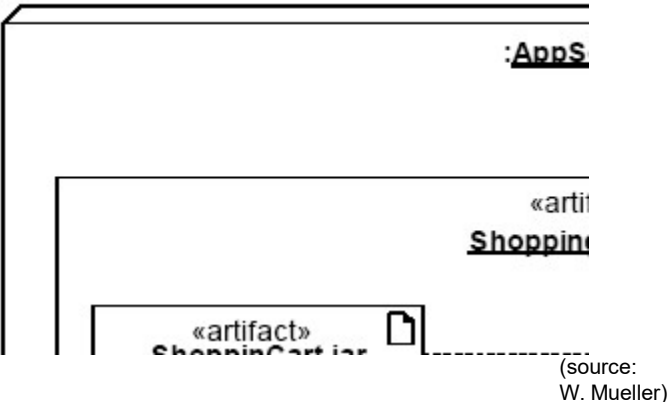
- Composite Structure Diagram Example

(source: W. Mueller)

ECPS203: Embedded Systems Modeling and Design, Lecture 17 (c) 2019 R. Doemer 24

Unified Modeling Language (UML)

- Deployment Diagram Example



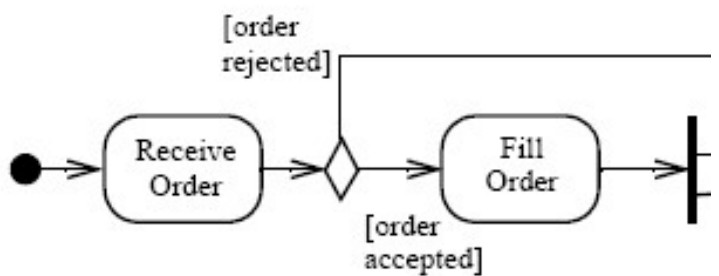
The diagram shows a nested structure of nodes. The outermost node is labeled **:AppS**. Inside it is a node labeled **«arti Shopping**. Inside that is a node labeled **«artifact» ShoppingCart.jar** with a file icon. A dashed line connects the **ShoppingCart.jar** node to the text **(source: W. Mueller)**.

(source:
W. Mueller)

ECPS203: Embedded Systems Modeling and Design, Lecture 17 (c) 2019 R. Doemer 25

Unified Modeling Language (UML)

- Activity Diagram Example



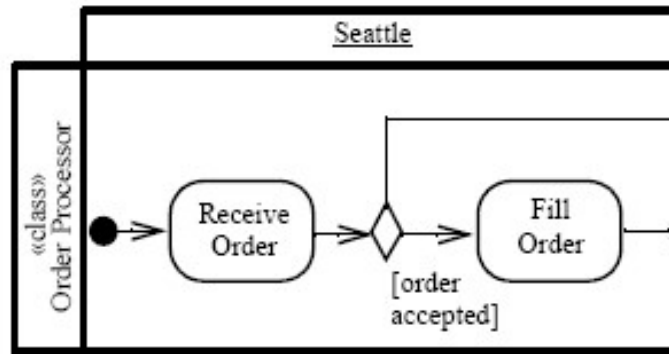
The diagram shows a flow starting from a black circle (start node) to an arrow pointing to a rounded rectangle labeled **Receive Order**. From there, an arrow points to a diamond-shaped decision node. Two paths emerge from the diamond: one labeled **[order rejected]** loops back to the arrow entering the diamond; the other labeled **[order accepted]** points to another rounded rectangle labeled **Fill Order**. An arrow from **Fill Order** points to a vertical bar with a T-shape (end node).

(source:
W. Mueller)

ECPS203: Embedded Systems Modeling and Design, Lecture 17 (c) 2019 R. Doemer 26

Unified Modeling Language (UML)

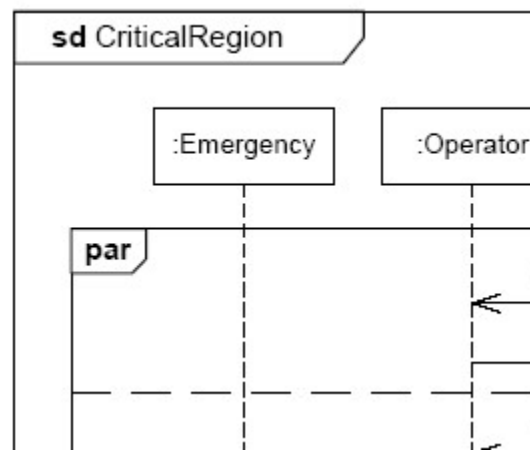
- Activity Diagram Example with “swim lanes”



(source: W. Mueller)

Unified Modeling Language (UML)

- Sequence Diagram Example



(source: W. Mueller)

Unified Modeling Language (UML)

- Use Case Diagram Examples

(source: W. Mueller)

ECPS203: Embedded Systems Modeling and Design, Lecture 17 (c) 2019 R. Doemer 29

Unified Modeling Language (UML)

- State Machine Diagram Examples

(source: W. Mueller)

ECPS203: Embedded Systems Modeling and Design, Lecture 17 (c) 2019 R. Doemer 30