ECPS 203
Embedded Systems Modeling and Design
Lecture 8

Rainer Dömer

doemer@uci.edu

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

Lecture 8: Overview

- Course Administration
  - Midterm course evaluation
- SystemC Simulation Semantics
  - Motivating Examples
  - Discrete Event Simulation Algorithm
- Project Discussion
  - Status and next steps
  - Assignment 4

Course Administration

- Midterm Course Evaluation
  - One week, starting today!
    - Wednesday, Oct. 24, 8am – Tuesday, Oct. 30, 8pm
  - Online via EEE+ Evaluations
- Feedback from students to instructors
  - Completely voluntary
  - Completely anonymous
  - Very valuable
    - Help to improve this class!
- Final Course Evaluation
  - expected for week 10 (TBA)

SystemC Simulation Semantics

- Motivating Example 1
  - Given:

```
SC_MODULE(Top)
{
  int x;

  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { th1();
    th2();
  }
};
```

```
void Top::th1(void)
{
  x = 5;
};
```

```
void Top::th2(void)
{
  x = 6;
};
```

  - What is the value of `x` at the end of simulation?
  - Answer: x = 6

SystemC Simulation Semantics

- Motivating Example 2
  - Given:

```
SC_MODULE(Top)
{
  int x;

  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  x = 5;
};
```

```
void Top::th2(void)
{
  x = 6;
};
```

  - What is the value of `x` at the end of simulation?
  - Answer: The model is non-deterministic!
    `x` may have the value 5 or 6.

SystemC Simulation Semantics

- Motivating Example 3
  - Given:

```
SC_MODULE(Top)
{
  int x;

  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  wait(10, SC_NS);
  x = 5;
};
```

```
void Top::th2(void)
{
  x = 6;
};
```

  - What is the value of `x` at the end of simulation?
  - Answer: `x = 5`

SystemC Simulation Semantics

- Motivating Example 4
  - Given:

```
SC_MODULE(Top)
{
  int x;

  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  wait(10, SC_NS);
  x = 5;
};
```

```
void Top::th2(void)
{
  wait(10, SC_NS);
  x = 6;
};
```

  - What is the value of `x` at the end of simulation?
  - Answer: The model is non-deterministic!
    `x` may have the value 5 or 6.

SystemC Simulation Semantics

- Motivating Example 5
  - Given:

```
SC_MODULE(Top)
{
  int x;
  sc_event e;
  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  x = 5;
  e.notify();
};
```

```
void Top::th2(void)
{
  wait(e);
  x = 6;
};
```

  - What is the value of `x` at the end of simulation?
  - Answer: The model is non-deterministic!
    `x` may have the value 5 or 6
    (immediate notification may get lost!)

SystemC Simulation Semantics

- Motivating Example 6
  - Given:

```
SC_MODULE(Top)
{
  int x;
  sc_event e;
  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  x = 5;
  e.notify(
     SC_ZERO_TIME);
};
```

```
void Top::th2(void)
{
  wait(e);
  x = 6;
};
```

  - What is the value of `x` at the end of simulation?
  - Answer: `x = 6`

SystemC Simulation Semantics

- Motivating Example 7
  - Given:

```
SC_MODULE(Top)
{
  int x;
  sc_event e;
  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  e.notify(
     SC_ZERO_TIME);
  x = 5;
};
```

```
void Top::th2(void)
{
  wait(e);
  x = 6;
};
```

  - What is the value of `x` at the end of simulation?
  - Answer: `x = 6`

SystemC Simulation Semantics

- Motivating Example 8
  - Given:

```
SC_MODULE(Top)
{
  int x;
  sc_event e;
  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  wait(10, SC_NS);
  x = 5;
  e.notify(
     SC_ZERO_TIME);
};
```

```
void Top::th2(void)
{
  wait(e);
  x = 6;
};
```

  - What is the value of **x** at the end of simulation?
  - Answer: **x = 6**

SystemC Simulation Semantics

- Motivating Example 9
  - Given:

```
SC_MODULE(Top)
{
  int x;
  sc_event e;
  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  x = 5;
  e.notify(
     SC_ZERO_TIME);
};
```

```
void Top::th2(void)
{
  wait(10, SC_NS);
  wait(e);
  x = 6;
};
```

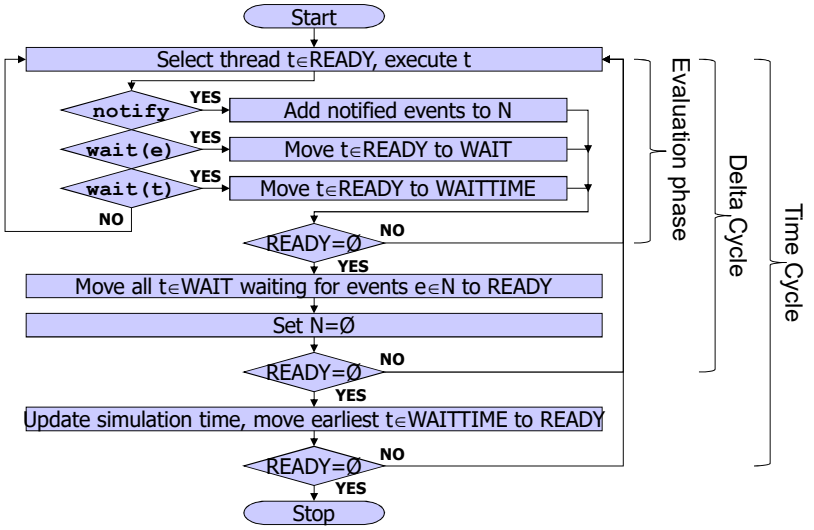
  - What is the value of **x** at the end of simulation?
  - Answer: **x = 5**
    Thread **th2** never completes,
    notified event **e** expires and is lost!

SystemC Simulation Semantics

- Discrete Event Simulation (DES) Algorithm
  - described in SystemC LRM (but noted in a different format)
  - ⇒ abstract definition defines a set of valid implementations
  - ⇒ intentionally defined with non-deterministic thread ordering
- Definitions:
  - At any time, each thread t is in one of the following sets:
    - **READY**: set of threads ready to execute (aka. **RUNNABLE**)
    - **WAIT**: set of threads suspended by `wait(event)`
    - **WAITTIME**: set of threads suspended by `wait(time)`
  - Notified events are stored in a set **N**
    - `notify e1` adds event `e1` to **N**
    - `wait e1` will wakeup when `e1` is in **N**
    - Consumption of event `e` means event `e` is taken out of **N**
    - Expiration of notified events means **N** is set to Ø

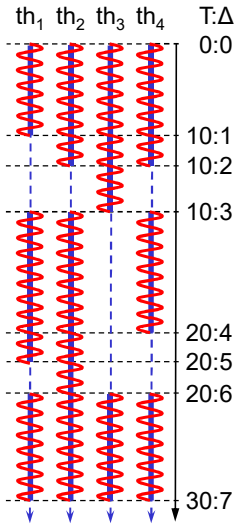
SystemC Simulation Semantics

- Discrete Event Simulation (DES) Algorithm



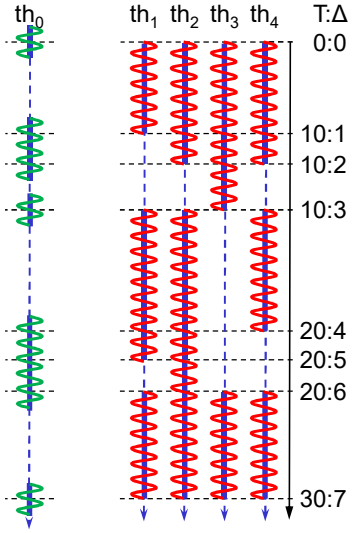
SystemC Simulation Semantics

- Discrete Event Simulation (DES)
  - Concurrent threads of execution
  - Managed by a central scheduler
  - Driven by events and time advances
    - Delta cycle
    - Time cycle
  - ➢ Partial temporal order with barriers
- Reference Simulator
  - IEEE SystemC specifies cooperative multi-threading
  - ➢ A single thread is active at any time (even if multiple cores are available)
  - Example: Execution of four threads



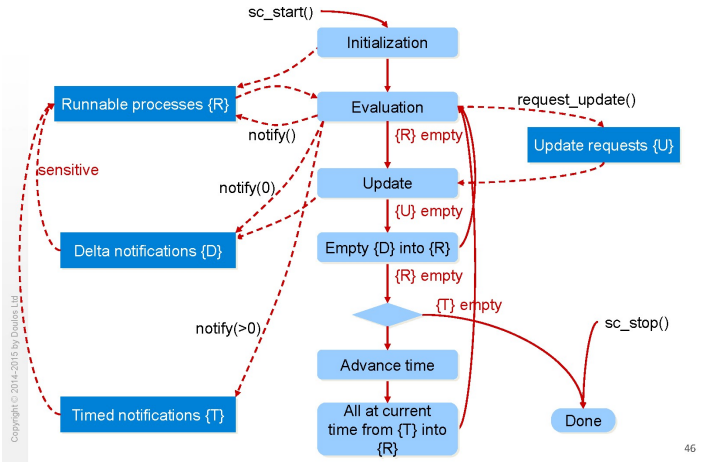
SystemC Simulation Semantics

- Accellera SystemC Proof-of-Concept Library
- ➢ uses an extra root thread for the following tasks:
  - Elaboration phase
  - Scheduling
    - Event notifications
    - Channel updates
    - Delta cycle updates
    - Simulation time updates
  - SC_METHOD calls
    - (not shown)



SystemC Simulation Semantics

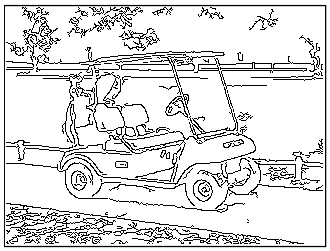
- Discrete Event Simulation (DES) Algorithm
  - The SystemC Scheduler (by Doulos)



ECPS 203 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, determine function-call tree
- Deliverables
  - Source code and text file: **`canny.c`**, **`canny.txt`**
- Due
  - Wednesday: October 10, 2018, 6pm

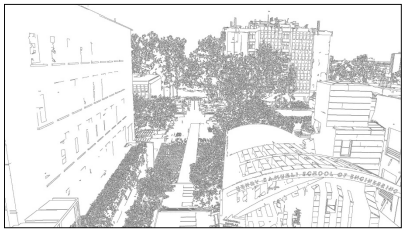
Project Assignment 2

- Task: Clean C++ model with static memory allocation
  - Prepare the C++ source code for modeling in SystemC
  - Configure parameters for specific application
  - Apply static memory allocation
- Steps
  1. Fix the off-by-one bug in the **`non_max_supp`** function
  2. Clean-up the code for compilation without warnings
  3. Fix configuration parameters to compile-time constants
  4. Remove or replace dynamic memory allocation
- Deliverables
  - Source code and text file: **`canny.cpp`**, **`canny.txt`**
- Due
  - Wednesday: October 17, 2018, 6pm

ECPS 203 Project

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

Engineering001.bmp



Engineering001_edges.pgm

- Process video shot by a drone flying over Engineering Plaza
  - Fly a drone over UCI Engineering Plaza, take video of buildings
  - Record a color video stream in high resolution
  - Extract a set of video frames suitable for use in our test bench

ECPS203: Embedded Systems Modeling and Design, Lecture 8 (c) 2018 R. Doemer 21

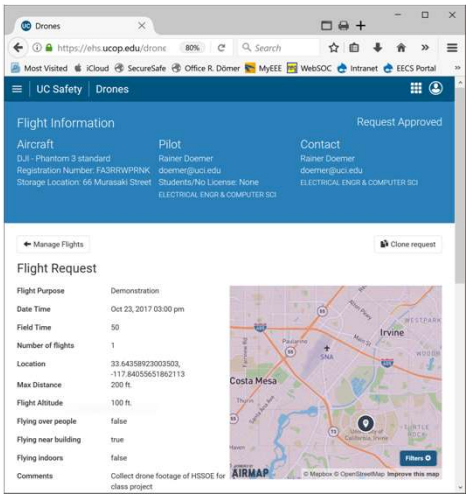
ECPS 203 Project: Drone Flight

- Capture Video Footage of Engineering Buildings
  - Google Map of UCI Engineering Quad

ECPS203: Embedded Systems Modeling and Design, Lecture 8 (c) 2018 R. Doemer 22

ECPS 203 Project: Drone Flight

- Capture Video Footage of Engineering Buildings
  - Drone flights in US require approval by the Federal Aviation Administration (FAA)
  - On UCI campus, Environmental Health & Safety (EHS) department is in charge of Unmanned Aircraft Safety
  - ➢ Flight request approved
    - Thursday, October 19, 2017



ECPS 203 Project: Drone Flight

- Capture Video Footage of Engineering Buildings
  - Drone Equipment
    - DJI Phantom 3 Standard Quadcopter
    - Remote Control with Mobile Device

[Image source: dji.com]

  - Drone carrries a Camera attached to a Gimble
    - Video stream stored on a SD memory card, e.g. DJI_0001.MOV
    - Video is 30 frames per second
    - Frames are 2704 by 1520 pixels

ECPS 203 Project: Drone Flight

- Capture Video Footage of Engineering Buildings
  - Screen Shot of Drone Control App on Mobile Device

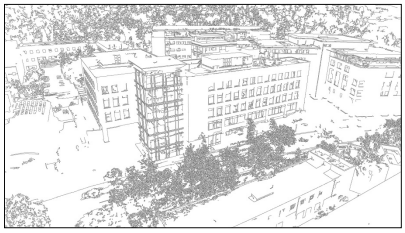
ECPS 203 Project: Drone Flight

- Capture Video Footage of Engineering Buildings
  - Drone flight demonstration (Fall 2017)

ECPS 203 Project

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

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

Project Assignment 4

- Task: From Single Image to Video Stream Processing
  - Prepare a sequence of image frames from the video
  - Convert the Canny application to process the video frames
- Steps
  1. Extract 30 of video frames suitable for use in a test bench
  2. Convert the color frames to grey-scale images in PGM format
  3. Recode your Canny C++ model to process the video frames
     - To run Canny application successfully, increase stack size
     - Adjust Canny parameters for the “best looking” output images
- Deliverables
  - Source code and text file: **`Canny.cpp`**, **`Canny.txt`**
- Due
  - Wednesday, October 31, 2018, 6pm