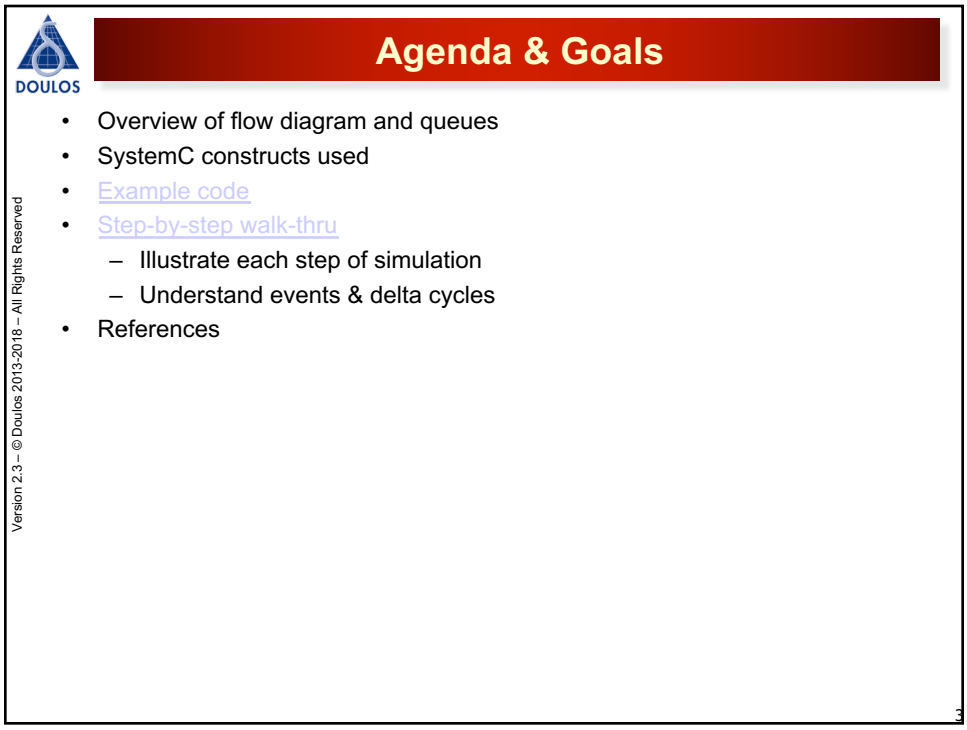
The SystemC Simulation Engine

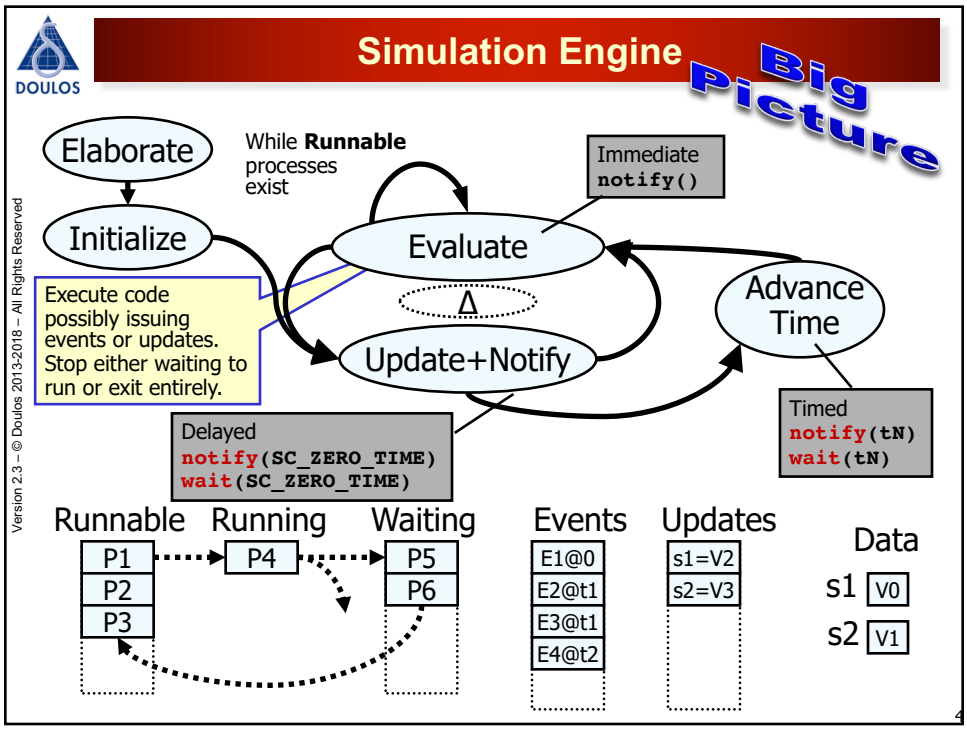
An Interactive Exploration of an Event Driven Simulator

Doulos Inc.
David C. Black, Senior Member Technical Staff

Copyright Permissions

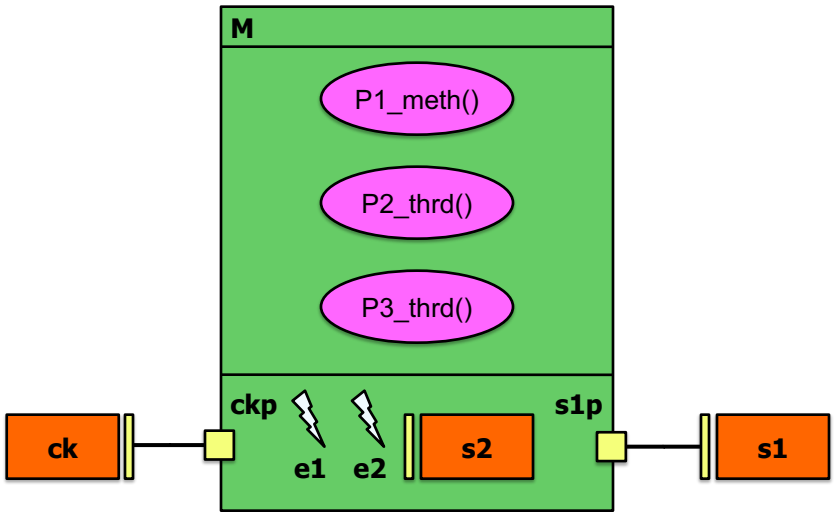
- This work is copyright 2013-2018 by Doulos Inc. with all rights reserved. Permission is hereby granted for professors of recognized licensed Colleges and Universities to use in lectures including the creation of handouts for registered students. Individuals may download and use for personal education, but not for display, public presentation nor redistribution electronically or in any other format. Any other commercial use is strictly prohibited without explicit written permission from Doulos Inc.
- In simpler language, competitors to Doulos training services may not use this material unless separately licensed. Display on the Internet is prohibited except from Doulos authorized locations.


DOULOS
Agenda & Goals
• Overview of flow diagram and queues
• SystemC constructs used
• Example code
• Step-by-step walk-thru
– Illustrate each step of simulation
– Understand events & delta cycles
• References
Version 2.3 – © Doulos 2013-2018 – All Rights Reserved
3


DOULOS
Simulation Engine
Big Picture
Elaborate
Initialize
While Runnable processes exist
Evaluate
Δ
Update+Notify
Advance Time
Immediate notify()
Execute code possibly issuing events or updates. Stop either waiting to run or exit entirely.
Delayed
notify(SC_ZERO_TIME)
wait(SC_ZERO_TIME)
Timed
notify(tN)
wait(tN)
Runnable
Running
Waiting
P1
P2
P3
P4
P5
P6
Events
E1@0
E2@t1
E3@t1
E4@t2
Updates
s1=V2
s2=V3
Data
s1 V0
s2 V1
Version 2.3 – © Doulos 2013-2018 – All Rights Reserved
4

DOULOS

Example Design



5

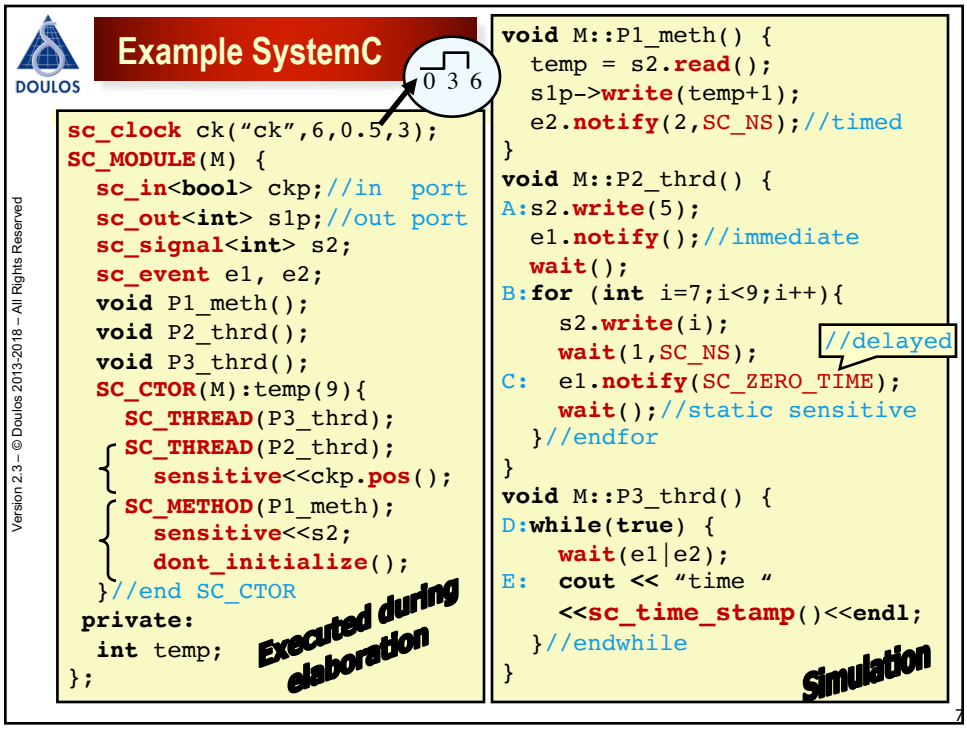
DOULOS

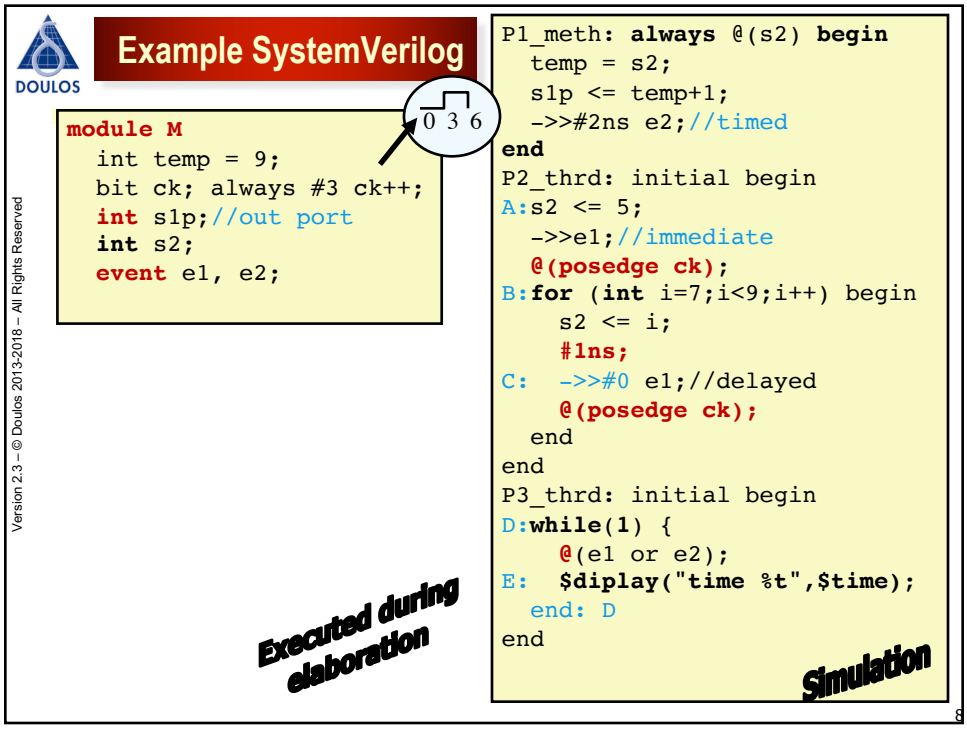
SystemC constructs used herein

- `SC_MODULE`, `SC_CTOR` - used to create modules
- `SC_THREAD`[1], `SC_METHOD`[1] - types of processes
- `sensitive`, `dont_initialize` - attributes of processes
- `sc_event`, `wait`[2], `notify`[3] - synchronization mechanisms
- `sc_signal`[4], `read`, `write` - primitive channel
- `sc_clock` - `sc_signal`<**bool**> with a generating process
- `sc_in` - `sc_port`<> specialization of type `sc_signal_in_if`<>
- `sc_out` - `sc_port`<> specialization of type `sc_signal_out_if`<>

[1] Verilog **`initial`** or **`always`** block; VHDL **`process`** block
[2] Verilog **`@`**, **`#`**, or **`wait`** statement; VHDL **`wait`** statement
[3] Verilog **`->`** statement, except SystemC is more flexible
[4] Verilog **`wire`** or **`var`** with **`<=`** type; VHDL **`signal`** type

6


DOULOS
Example SystemC
0 3 6
sc_clock ck("ck",6,0.5,3);
SC_MODULE(M) {
sc_in<bool> ckp;//in port
sc_out<int> s1p;//out port
sc_signal<int> s2;
sc_event e1, e2;
void P1_meth();
void P2_thrd();
void P3_thrd();
SC_CTOR(M):temp(9){
SC_THREAD(P3_thrd);
SC_THREAD(P2_thrd);
sensitive<<ckp.pos();
SC_METHOD(P1_meth);
sensitive<<s2;
dont_initialize();
}//end SC_CTOR
private:
int temp;
};
Executed during elaboration
void M::P1_meth() {
temp = s2.read();
s1p->write(temp+1);
e2.notify(2,SC_NS);//timed
}
void M::P2_thrd() {
A:s2.write(5);
e1.notify();//immediate
wait();
B:for (int i=7;i<9;i++){
s2.write(i);
wait(1,SC_NS);
C: e1.notify(SC_ZERO_TIME);
wait();//static sensitive
}//endfor
}
void M::P3_thrd() {
D:while(true) {
wait(e1|e2);
E: cout << "time "
<<sc_time_stamp()<<endl;
}//endwhile
}
//delayed
Simulation
Version 2.3 – © Doulos 2013-2018 – All Rights Reserved
7


DOULOS
Example SystemVerilog
0 3 6
module M
int temp = 9;
bit ck; always #3 ck++;
int s1p;//out port
int s2;
event e1, e2;
Executed during elaboration
P1_meth: always @(s2) begin
temp = s2;
s1p <= temp+1;
->>#2ns e2;//timed
end
P2_thrd: initial begin
A:s2 <= 5;
->>e1;//immediate
@(posedge ck);
B:for (int i=7;i<9;i++) begin
s2 <= i;
#1ns;
C: ->>#0 e1;//delayed
@(posedge ck);
end
end
P3_thrd: initial begin
D:while(1) {
@(e1 or e2);
E: $diplay("time %t",$time);
end: D
end
Simulation
Version 2.3 – © Doulos 2013-2018 – All Rights Reserved
8