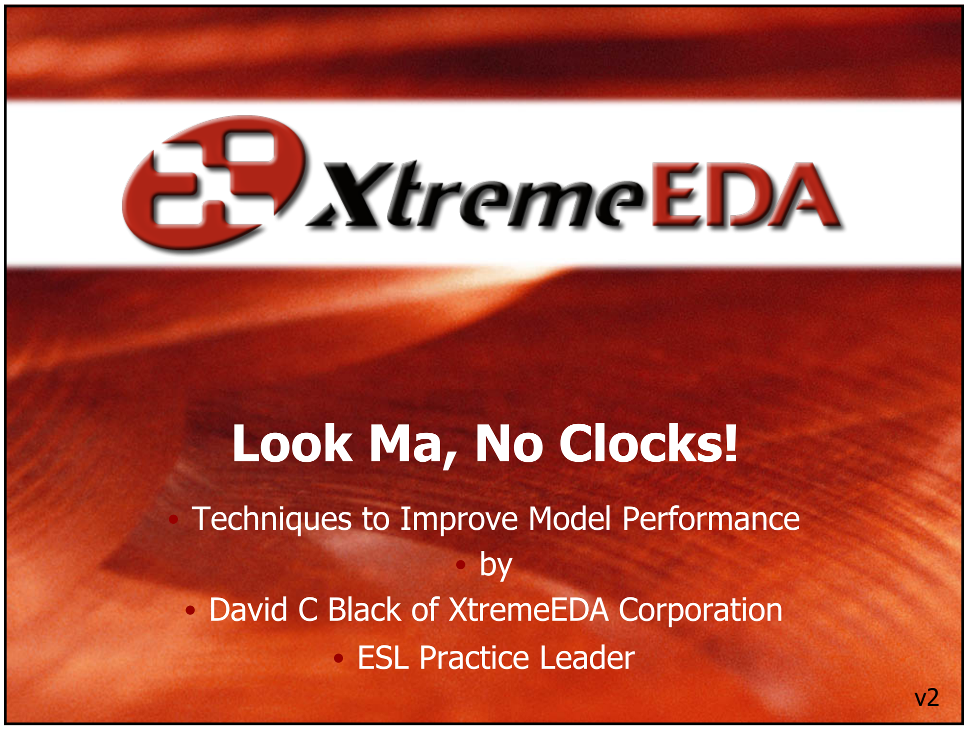
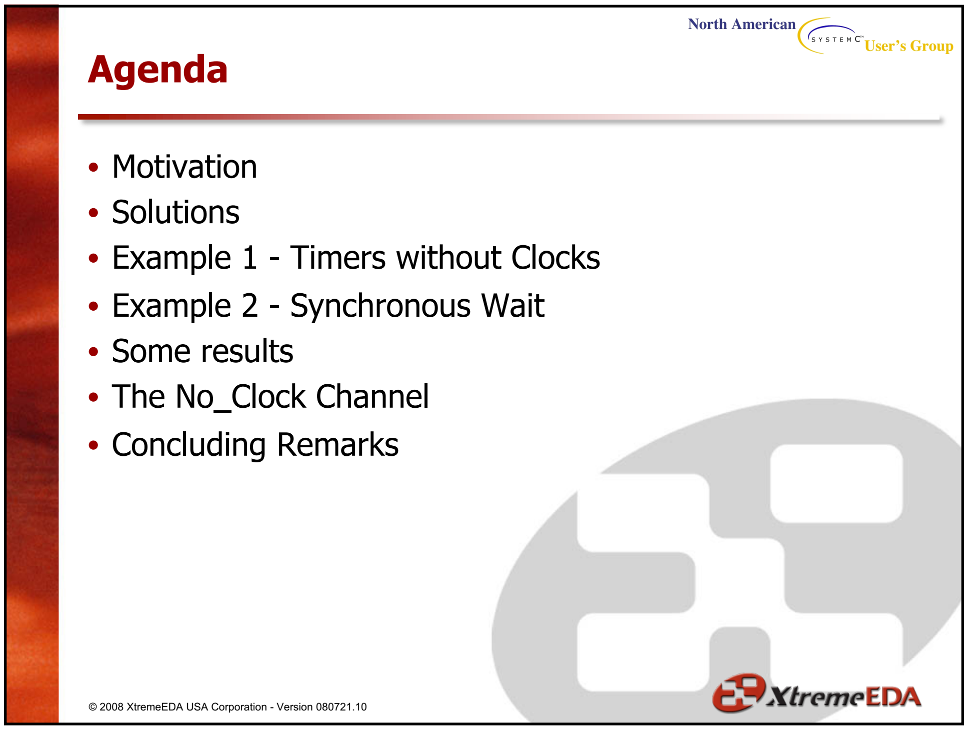


Look Ma, No Clocks!

- Techniques to Improve Model Performance
- by
- David C Black of XtremeEDA Corporation
- ESL Practice Leader

v2

1

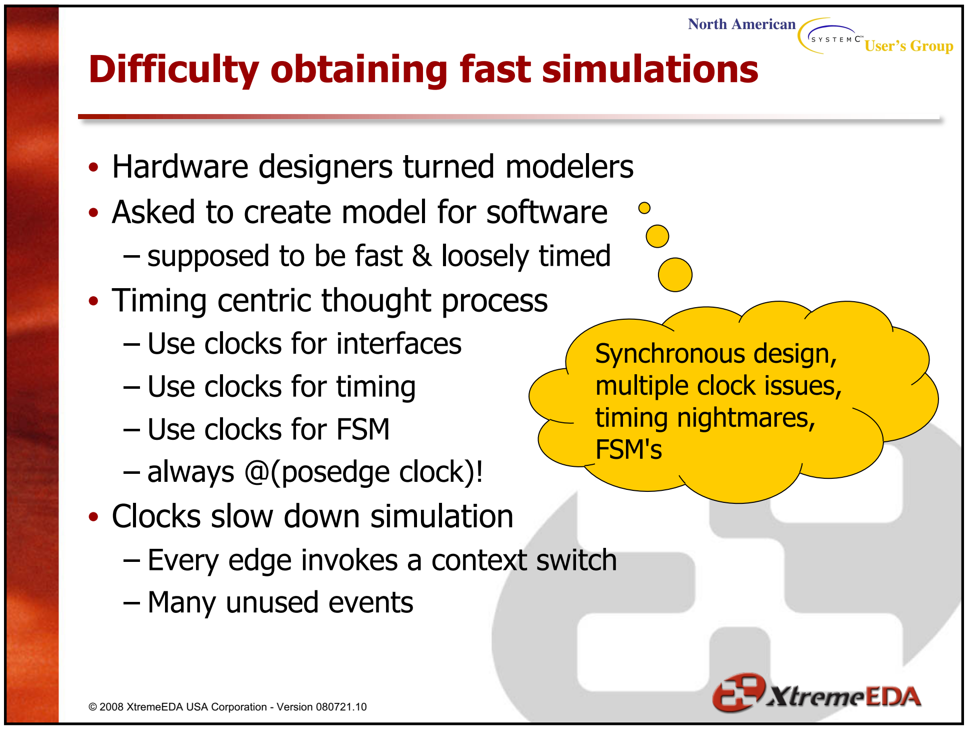


North American SystemC User's Group

Agenda

- Motivation
- Solutions
- Example 1 - Timers without Clocks
- Example 2 - Synchronous Wait
- Some results
- The No_Clock Channel
- Concluding Remarks

3



North American SystemC User's Group

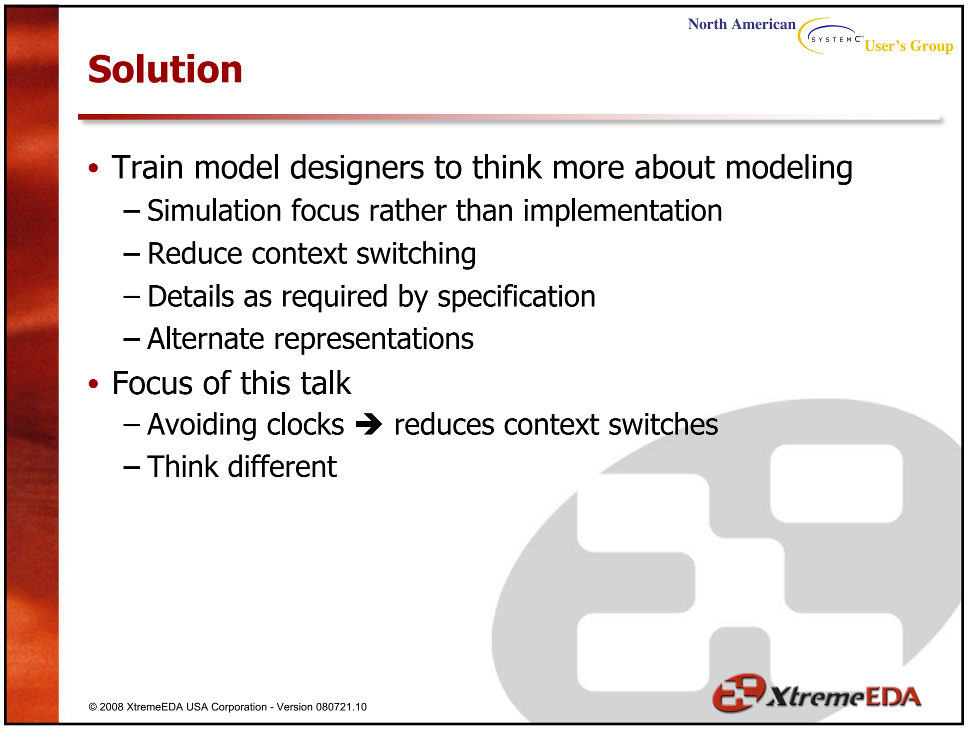
Difficulty obtaining fast simulations

- Hardware designers turned modelers
- Asked to create model for software
  - supposed to be fast & loosely timed
- Timing centric thought process
  - Use clocks for interfaces
  - Use clocks for timing
  - Use clocks for FSM
  - always @(posedge clock)!
- Clocks slow down simulation
  - Every edge invokes a context switch
  - Many unused events

Synchronous design, multiple clock issues, timing nightmares, FSM's

XtremeEDA

4



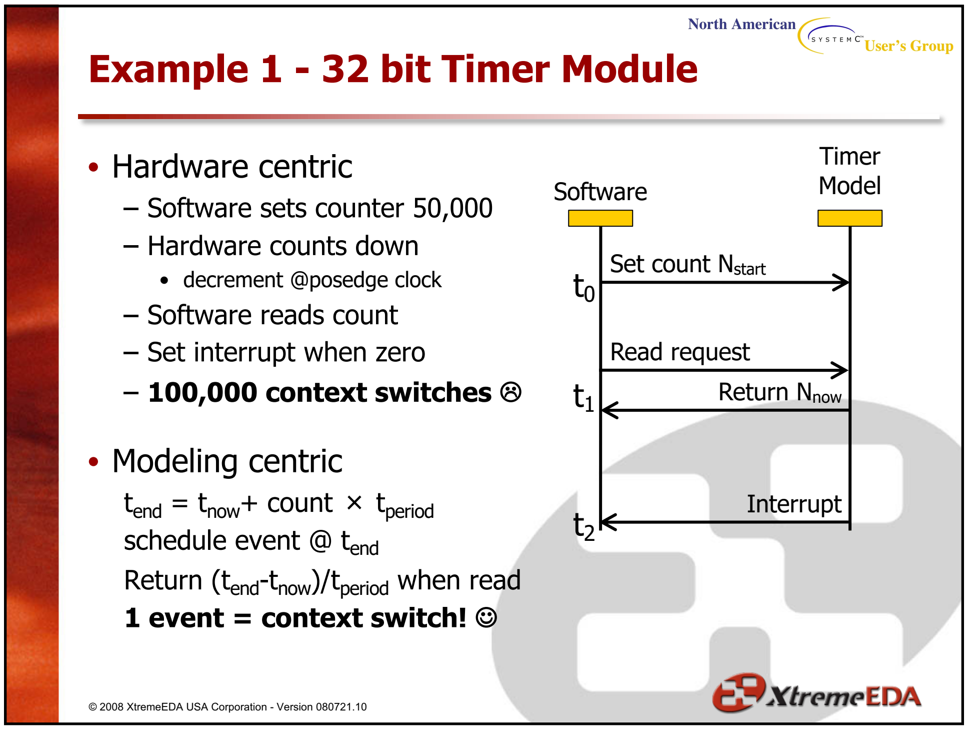
North American SystemC User's Group

Solution

- Train model designers to think more about modeling
  - Simulation focus rather than implementation
  - Reduce context switching
  - Details as required by specification
  - Alternate representations
- Focus of this talk
  - Avoiding clocks ➔ reduces context switches
  - Think different

XtremeEDA

5



Example 1 - 32 bit Timer Module

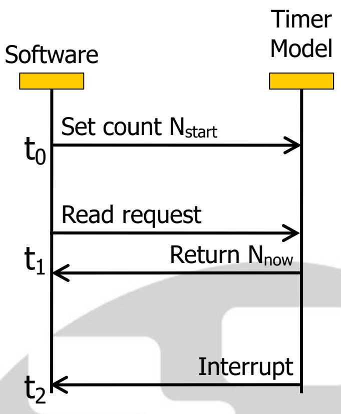
- Hardware centric
  - Software sets counter 50,000
  - Hardware counts down
    - decrement @posedge clock
  - Software reads count
  - Set interrupt when zero
  - **100,000 context switches ☹**
- Modeling centric

  t_{end} = t_{now}+ count × t_{period}

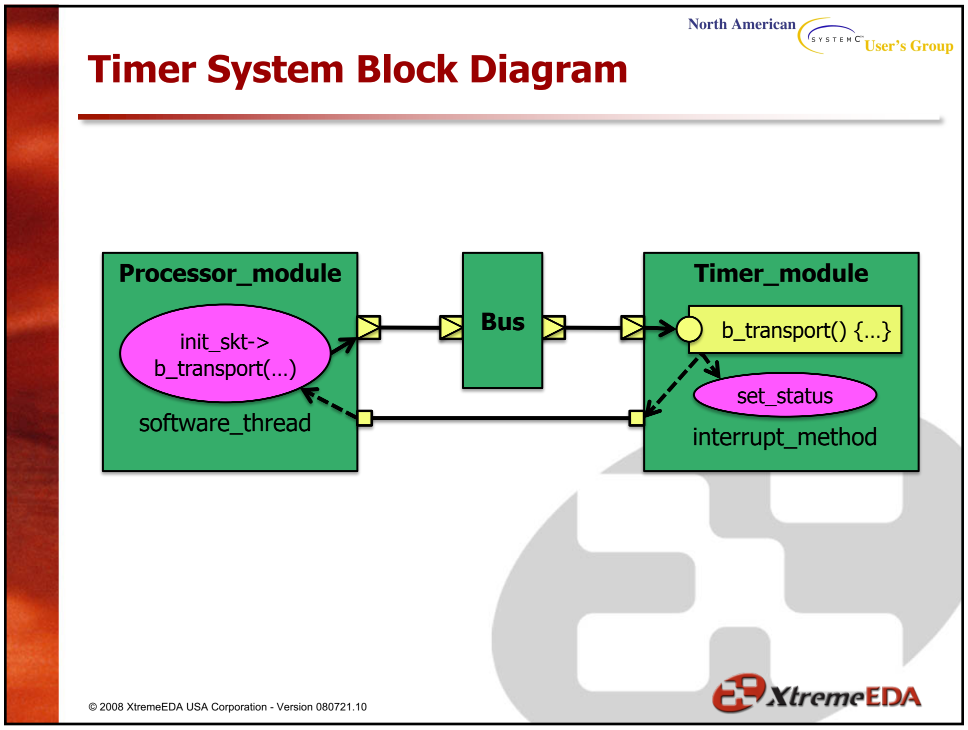
  schedule event @ t_{end}

  Return (t_{end}-t_{now})/t_{period} when read

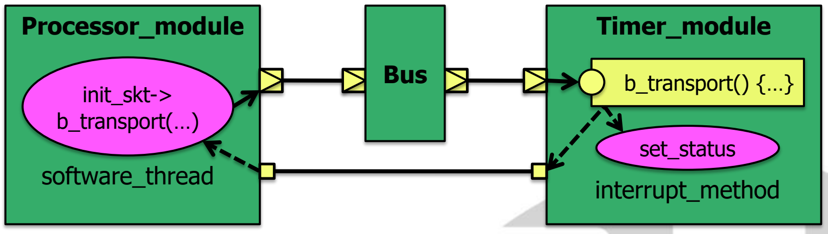
  1 event = context switch! ☺



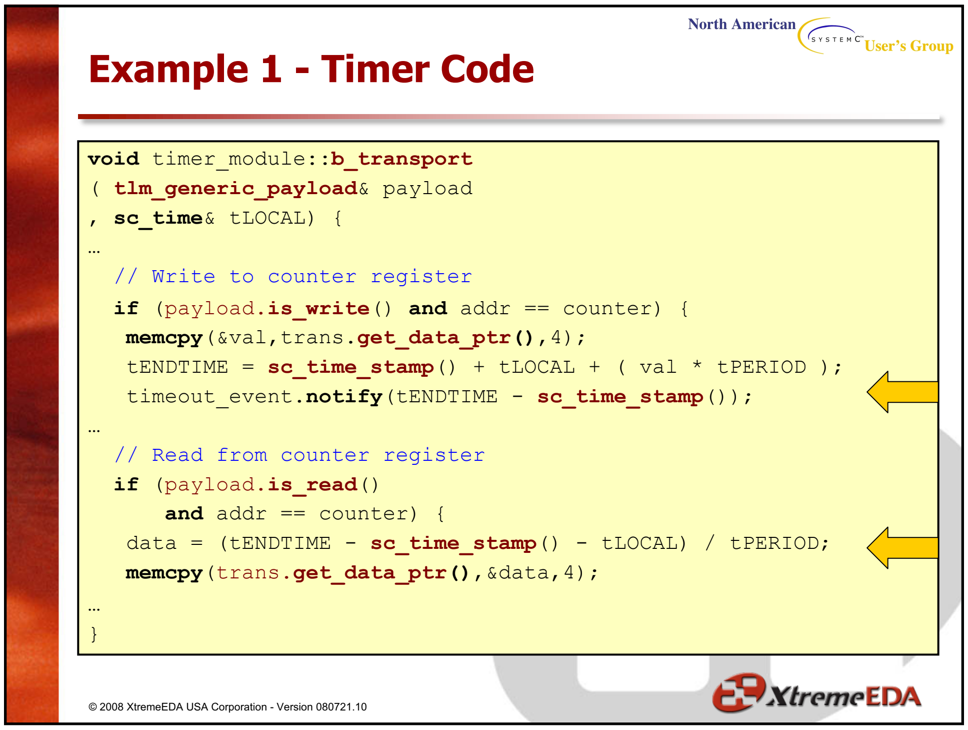
6



Timer System Block Diagram



7



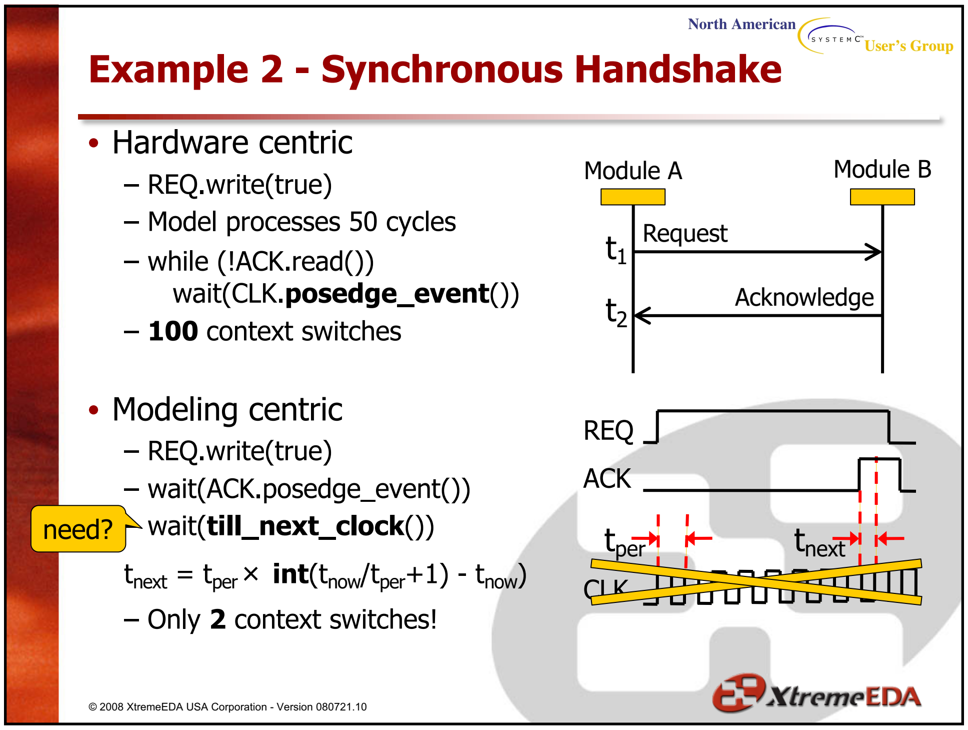
North American SystemC User's Group

Example 1 - Timer Code

```
void timer_module::b_transport
( tlm_generic_payload& payload
, sc_time& tLOCAL) {
…
  // Write to counter register
  if (payload.is_write() and addr == counter) {
   memcpy(&val,trans.get_data_ptr(),4);
   tENDTIME = sc_time_stamp() + tLOCAL + ( val * tPERIOD );
   timeout_event.notify(tENDTIME - sc_time_stamp());
…
  // Read from counter register
  if (payload.is_read()
      and addr == counter) {
   data = (tENDTIME - sc_time_stamp() - tLOCAL) / tPERIOD;
   memcpy(trans.get_data_ptr(),&data,4);
…
}
```

XtremeEDA

8



North American SystemC User's Group

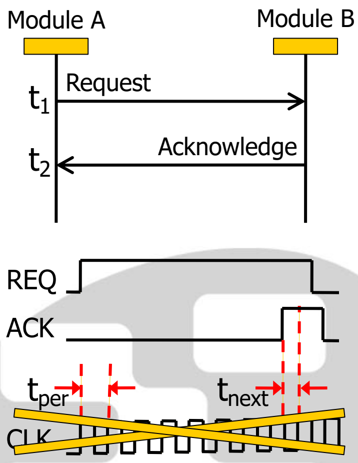
Example 2 - Synchronous Handshake

- Hardware centric
  - REQ.write(true)
  - Model processes 50 cycles
  - while (!ACK.read())
    wait(CLK.**posedge_event**())
  - **100** context switches
- Modeling centric
  - REQ.write(true)
  - wait(ACK.posedge_event())
  - wait(**till_next_clock**())

need?

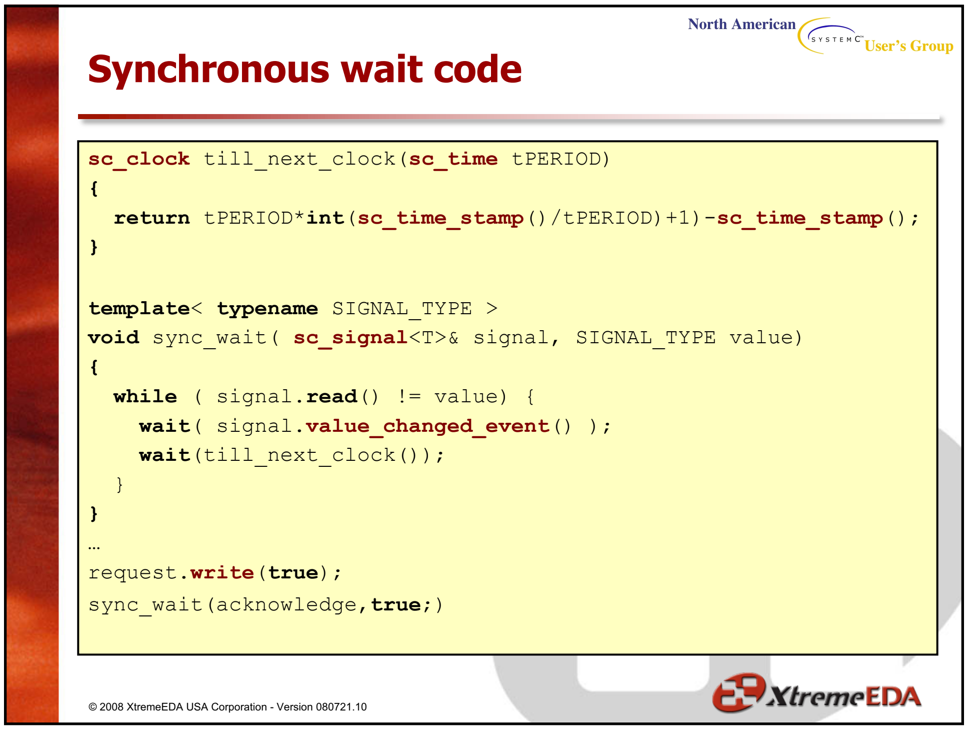
$t_{next} = t_{per} \times \textbf{int}(t_{now}/t_{per}+1) - t_{now}$

  - Only **2** context switches!



XtremeEDA

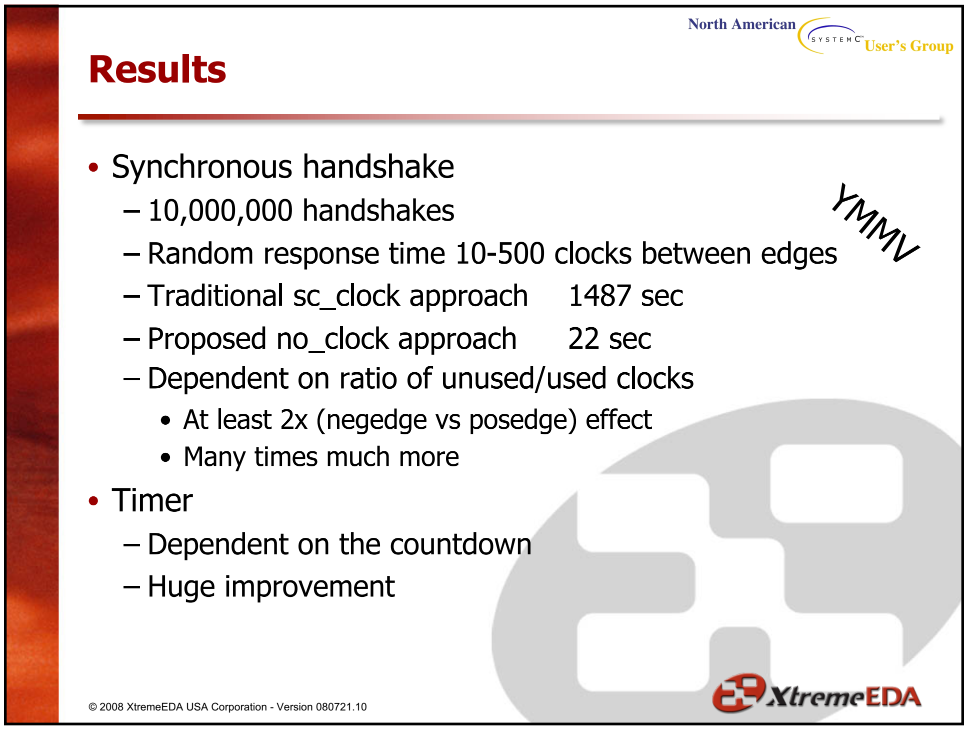
9



Synchronous wait code

```
sc_clock till_next_clock(sc_time tPERIOD)
{
  return tPERIOD*int(sc_time_stamp()/tPERIOD)+1)-sc_time_stamp();
}

template< typename SIGNAL_TYPE >
void sync_wait( sc_signal<T>& signal, SIGNAL_TYPE value)
{
  while ( signal.read() != value) {
    wait( signal.value_changed_event() );
    wait(till_next_clock());
  }
}
…
request.write(true);
sync_wait(acknowledge,true;)
```

10

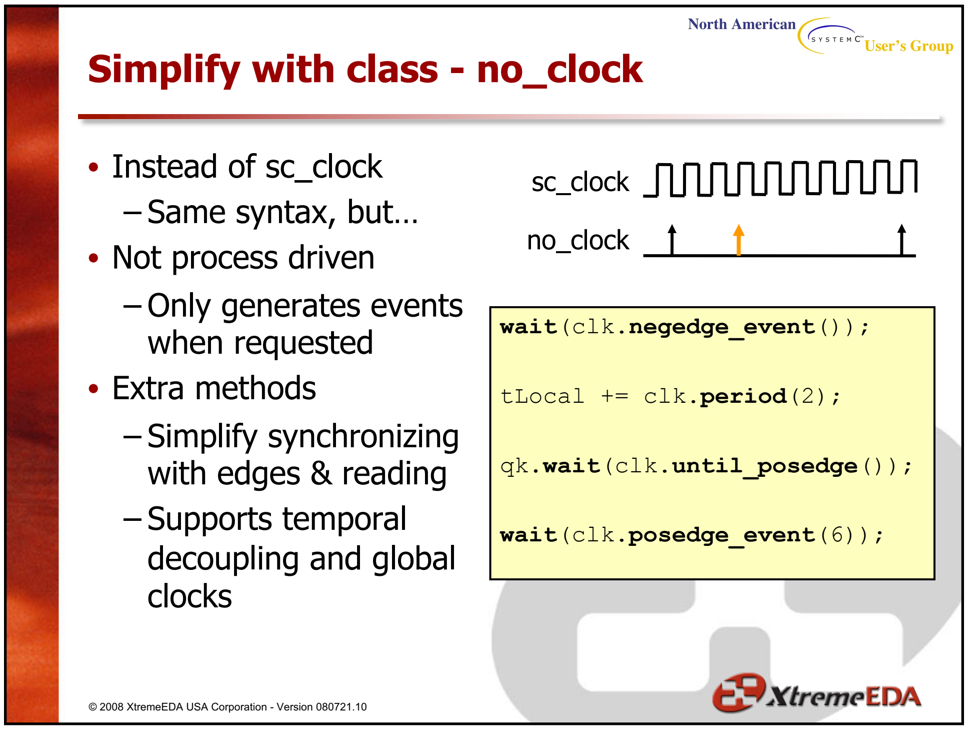


Results

- Synchronous handshake
  - 10,000,000 handshakes
  - Random response time 10-500 clocks between edges
  - Traditional sc_clock approach 1487 sec
  - Proposed no_clock approach 22 sec
  - Dependent on ratio of unused/used clocks
    - At least 2x (negedge vs posedge) effect
    - Many times much more
- Timer
  - Dependent on the countdown
  - Huge improvement

YMMV

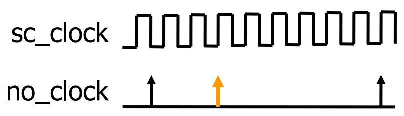
11



North American SystemC User's Group

Simplify with class - no_clock

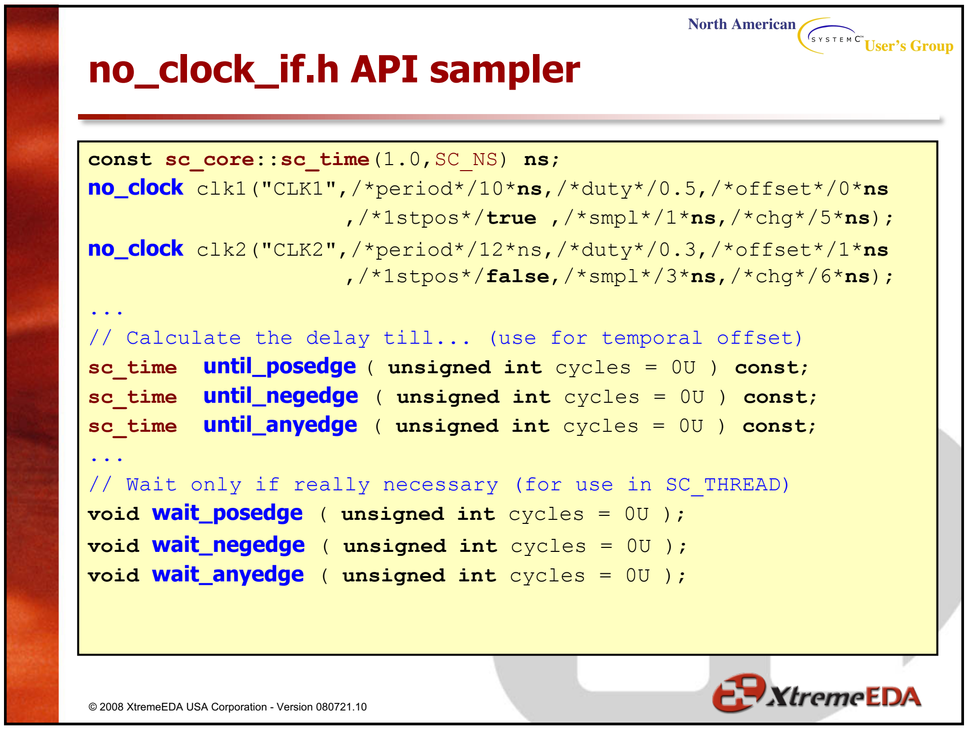
- Instead of sc_clock
  - Same syntax, but…
- Not process driven
  - Only generates events when requested
- Extra methods
  - Simplify synchronizing with edges & reading
  - Supports temporal decoupling and global clocks



```
wait(clk.negedge_event());

tLocal += clk.period(2);

qk.wait(clk.until_posedge());

wait(clk.posedge_event(6));
```

XtremeEDA

12



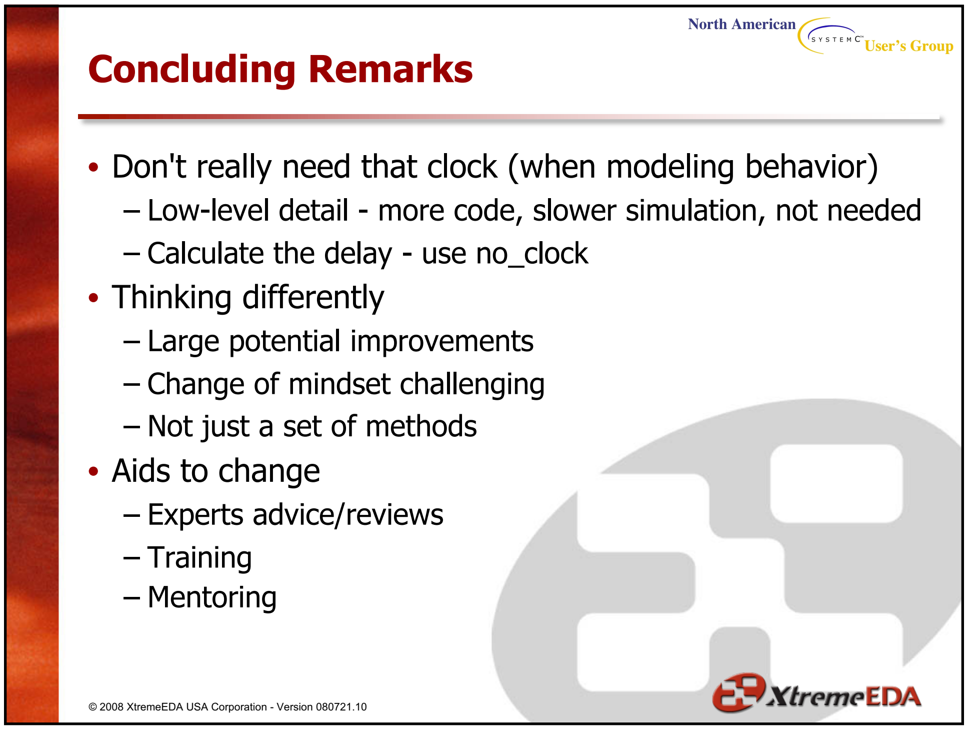
North American SystemC User's Group

no_clock_if.h API sampler

```
const sc_core::sc_time(1.0,SC_NS) ns;
no_clock clk1("CLK1",/*period*/10*ns,/*duty*/0.5,/*offset*/0*ns
                    ,/*1stpos*/true ,/*smpl*/1*ns,/*chg*/5*ns);
no_clock clk2("CLK2",/*period*/12*ns,/*duty*/0.3,/*offset*/1*ns
                    ,/*1stpos*/false,/*smpl*/3*ns,/*chg*/6*ns);
...
// Calculate the delay till... (use for temporal offset)
sc_time  until_posedge ( unsigned int cycles = 0U ) const;
sc_time  until_negedge ( unsigned int cycles = 0U ) const;
sc_time  until_anyedge ( unsigned int cycles = 0U ) const;
...
// Wait only if really necessary (for use in SC_THREAD)
void wait_posedge ( unsigned int cycles = 0U );
void wait_negedge ( unsigned int cycles = 0U );
void wait_anyedge ( unsigned int cycles = 0U );
```

XtremeEDA

13



North American SYSTEMC User's Group

Concluding Remarks

- Don't really need that clock (when modeling behavior)
  - Low-level detail - more code, slower simulation, not needed
  - Calculate the delay - use no_clock
- Thinking differently
  - Large potential improvements
  - Change of mindset challenging
  - Not just a set of methods
- Aids to change
  - Experts advice/reviews
  - Training
  - Mentoring

XtremeEDA

14

XtremeEDA

Questions?

SYSTEMC™

15