A Safety-Assured Development Approach for Real-Time Software

September 3, 2010

Eunkyoung Jee*, Shaohui Wang, Jeong Ki Kim, Jaewoo Lee,

Oleg Sokolsky, Insup Lee

PRECISE center, Department of Computer and Information Science

University of Pennsylvania

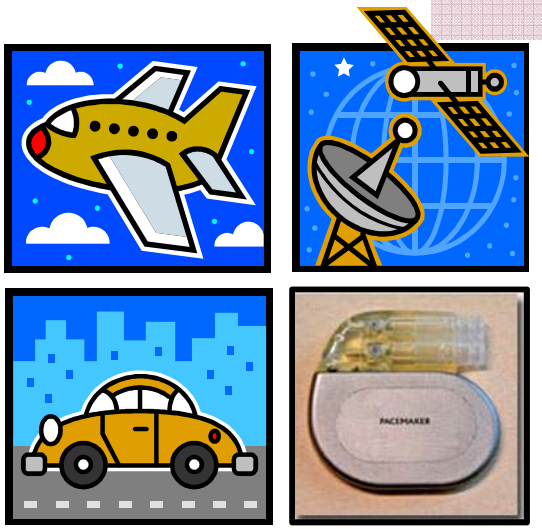
Penn Engineering

Outline

- Introduction
- Case Study: Pacemaker
- A Safety-Assured Development Approach for RTS
  - Formal Modeling and Verification
  - Implementation and Timing Analysis
  - Property Preservation
- Related Work
- Limitations and Possibilities
- Conclusion and Future Work

Introduction

- Guaranteeing timing properties is crucial for real-time safety critical systems



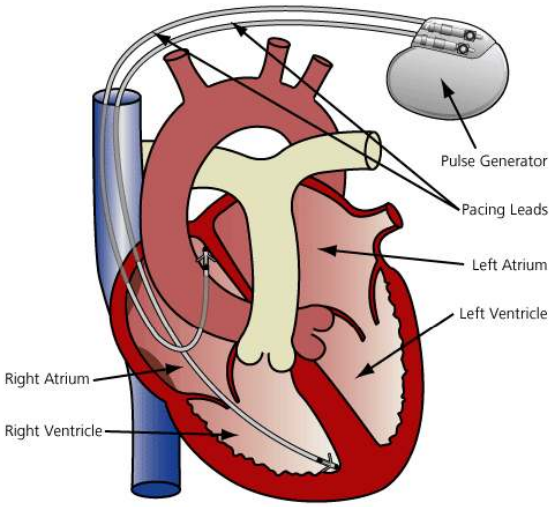
- We can have a formally verified model
- How can we implement time-guaranteed real-time software from a verified model?

- Present a safety-assured development approach for real-time software
  - Model-driven development + timing analysis
  - Pacemaker case study

Penn Engineering

Case Study: Pacemaker

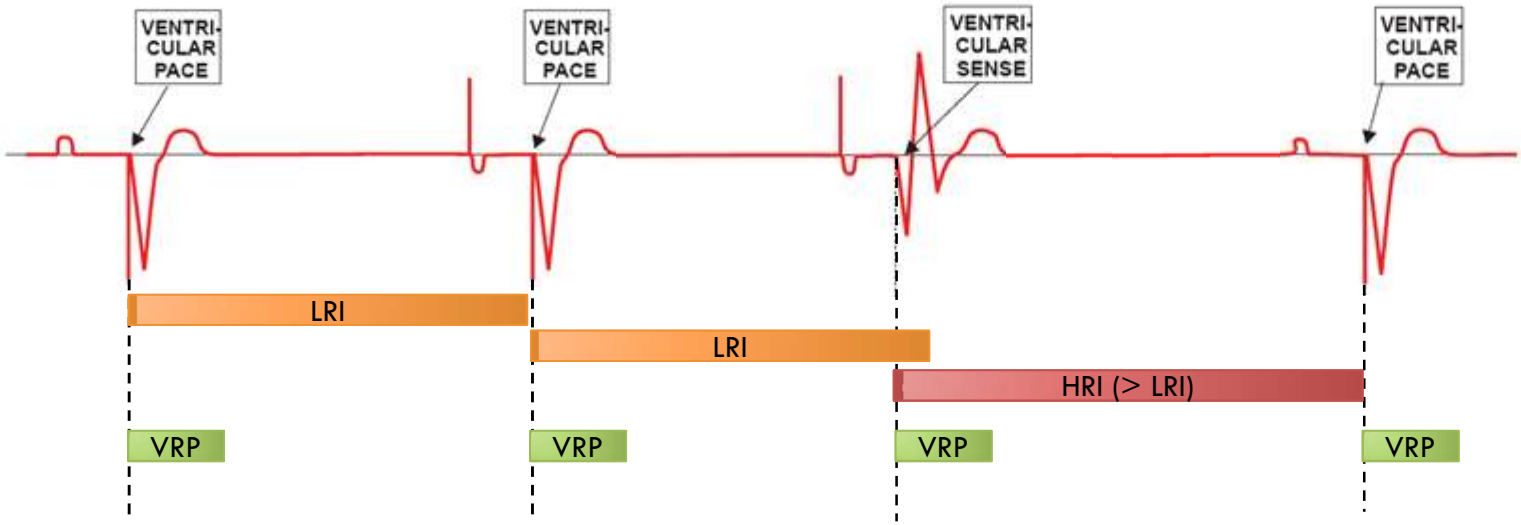
- Electronic device implanted in the body to regulate the heart beat
  - A life-critical real-time embedded system
- The Pacemaker Grand Challenge
  - The first certification challenge problem issued by the Software Certification Consortium (SCC)
  - Boston Scientific has released into the public domain the system specification for a previous generation pacemaker



img src: http://www.odec.ca/projects/2007/torr7m2/images/pacemaker.gif

Pacemaker Timing

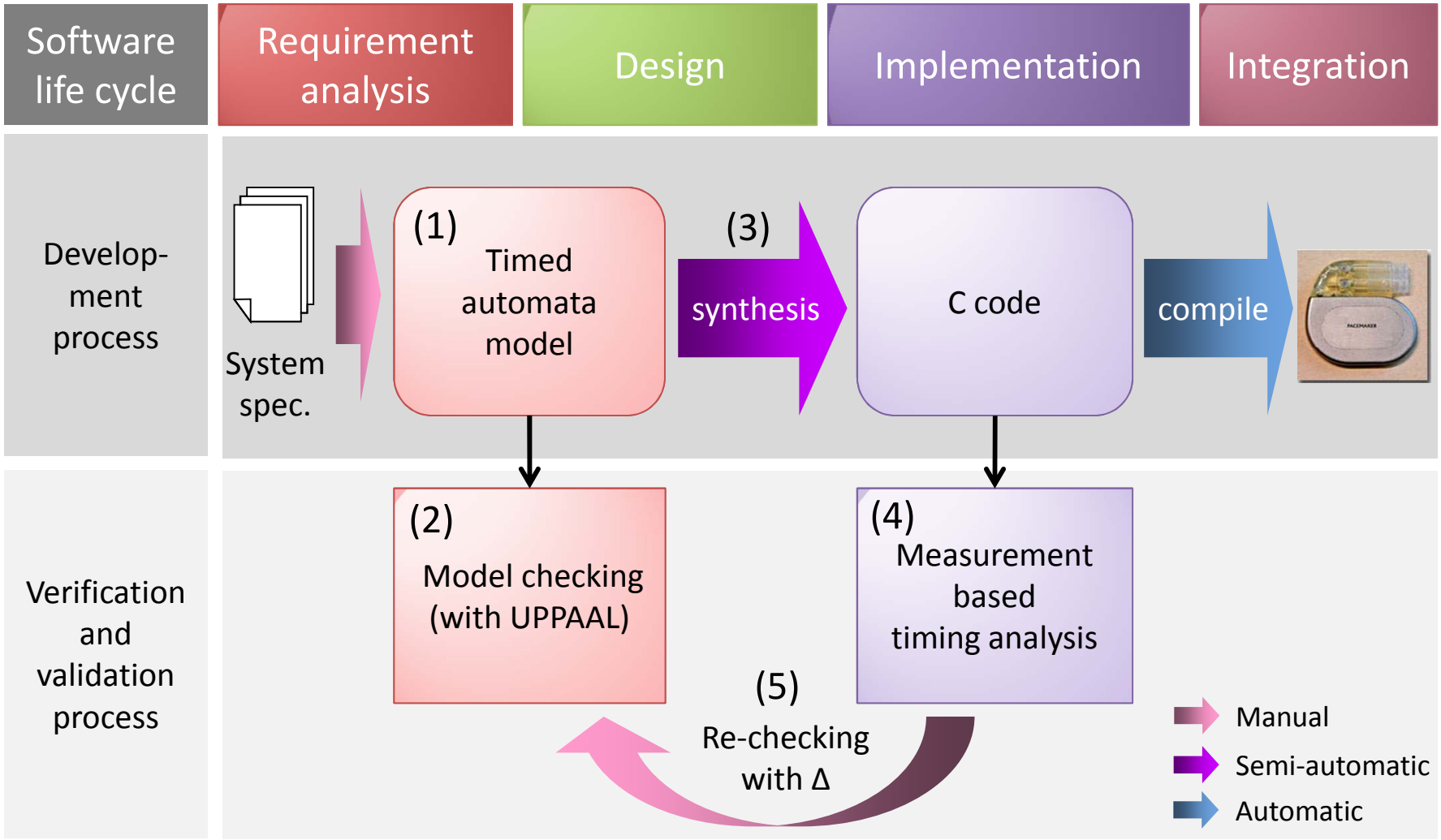
- Two basic functions
  - Pace
  - Sense intrinsic rhythm and inhibit
- Fundamental timing cycles of VVI mode (simplest mode)



- LRI: Lower Rate Interval (e.g., 1000ms)
- HRI: Hysteresis Rate Interval (e.g., 1200ms)
- VRP: Ventricular Refractory Period (e.g., 320ms)

Penn Engineering

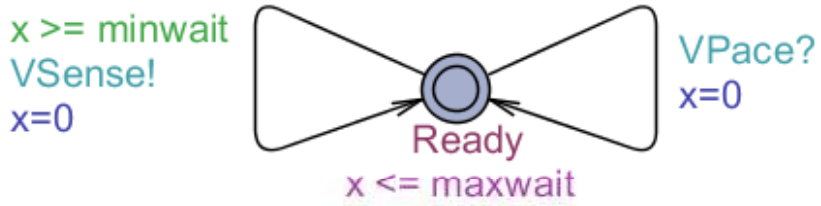
A Safety-Assured Development



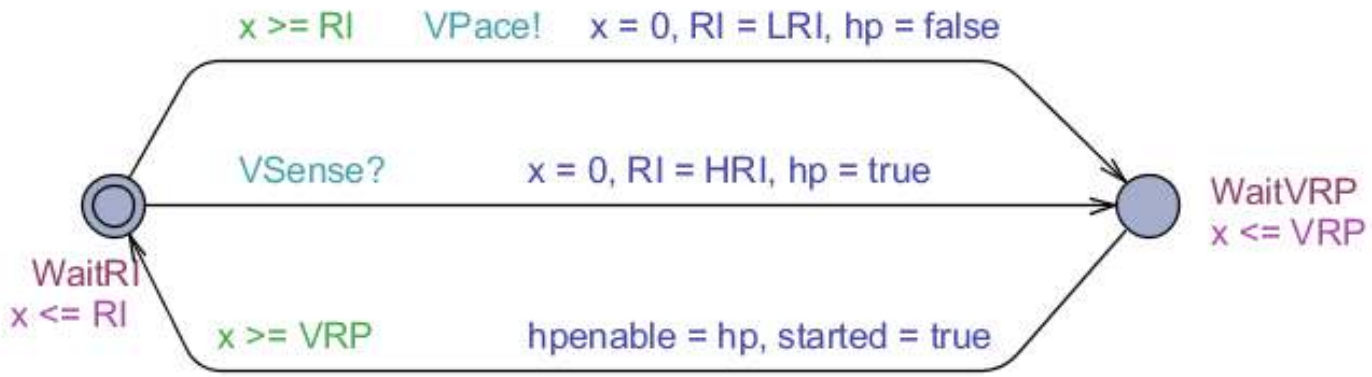
Formal Modeling in UPPAAL

Pacemaker on VVI mode

Heart



Ventricular controller



Formal Verification

- Model checking using UPPAAL
- Covered properties
  - PropLRI: Hysteresis pacing is deactivated → a ventricular pace or sense should be no later than LRI
    - A[] (!Ventricle.hpenable imply Ventricle.x <= Ventricle.LRI)
  - PropHRI: Hysteresis pacing is activated → a ventricular pace or sense should be no later than HRI
    - A[] (Ventricle.hpenable imply Ventricle.x <= Ventricle.HRI)
  - PropVRP: Sensing cannot occur before VRP ends
    - A[] ((Ventricle.WaitRI && Ventricle.started) imply Ventricle.x >= Ventricle.VRP)
  - Deadlock freeness
    - A[] (not deadlock)

All these properties were satisfied on the model

We assume that users capture all the important properties correctly.

Penn Engineering

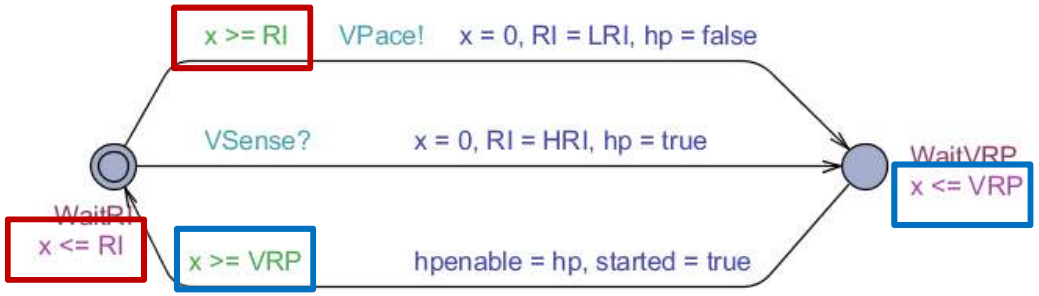
Implementation Goal

Goal

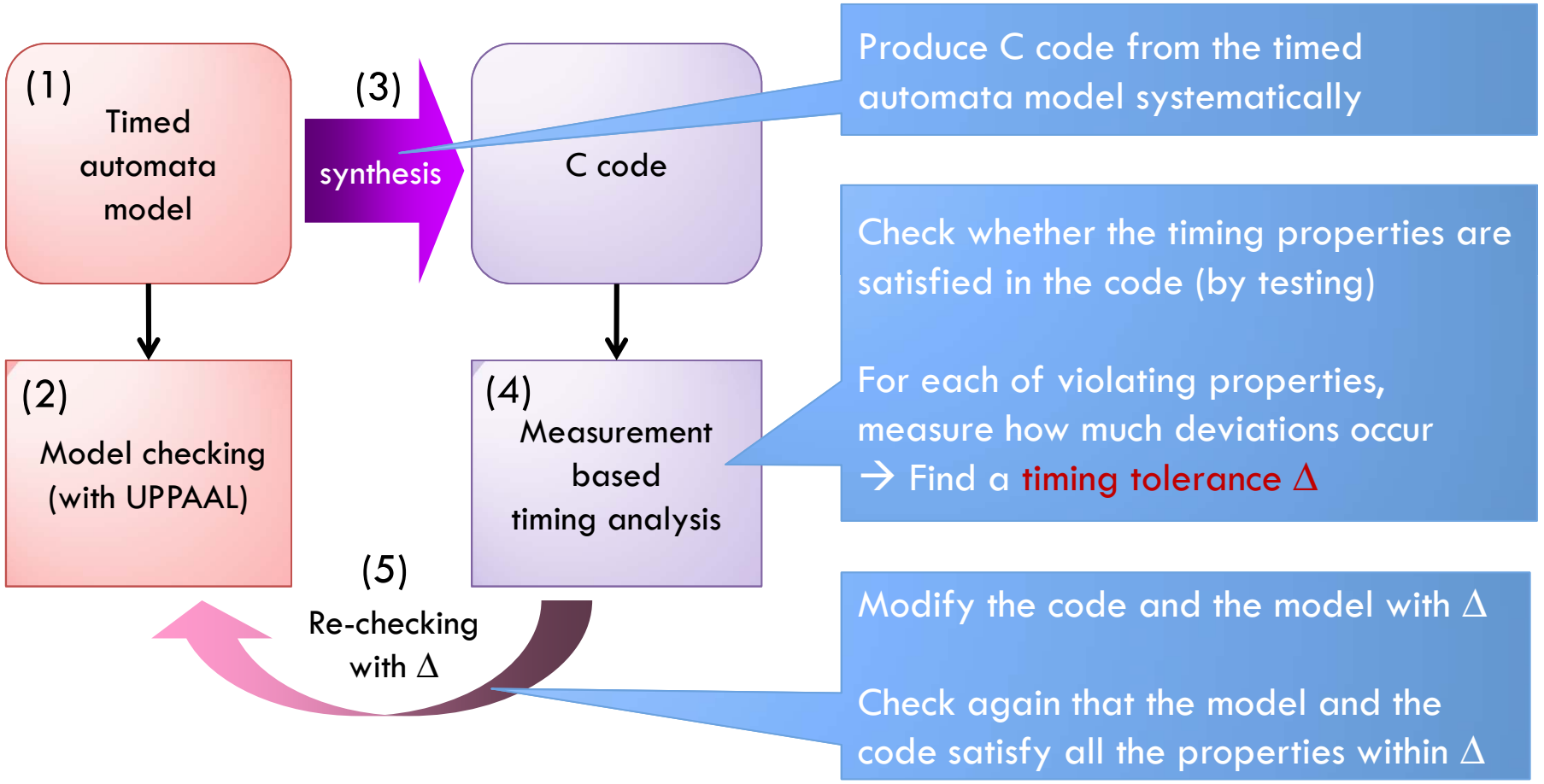
Generate C code guaranteeing properties transferred from the verified model within specific bounds

Obstacles

- Instantaneous responses in the model are not implementable
- Models use continuous clocks and implementations use digital clocks with finite precision

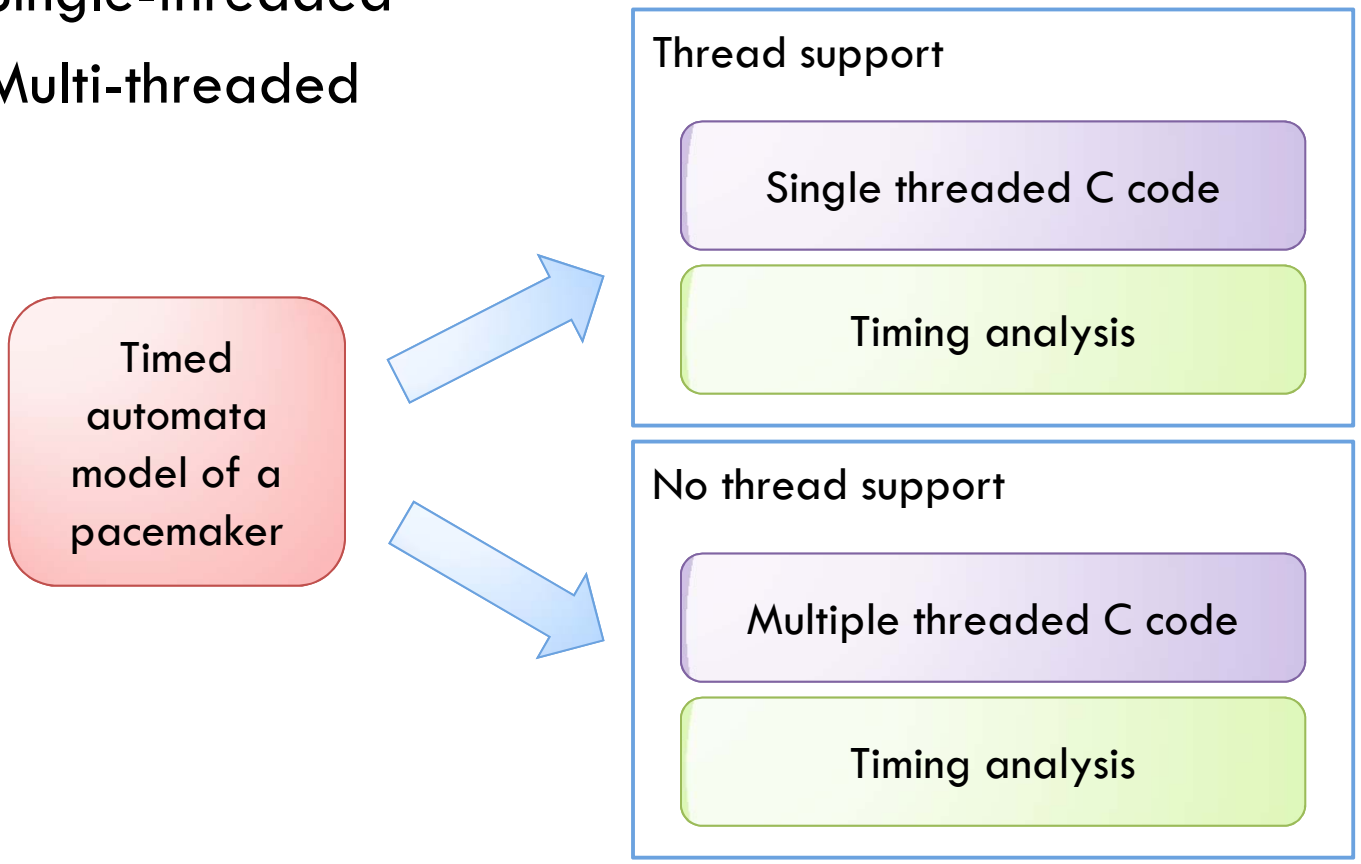


Code Synthesis & Timing Analysis



Code Synthesis

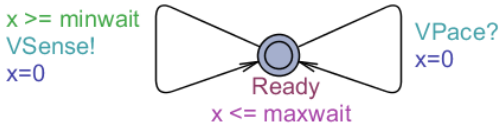
- Generated two flavors of implementation code
  - Single-threaded
  - Multi-threaded

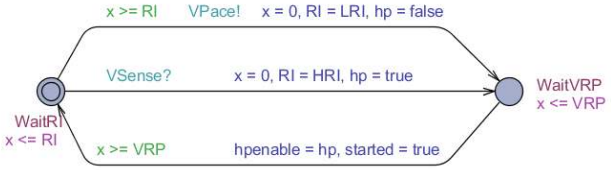


Penn Engineering

Single-threaded Code Structure

- One big loop
- Inspired by TIMES tool's code generation






```
…
for each trn do
     if (trn is active) and (eval_guard(trn))
          if (there is synchronization)
               if (compl_trn exists) and (eval_guard(compl_trn))
                    update variables of both trn and compl_trn;
                    take both trn and comp_trn to new states;
                    set trn to -1; /* check all outgoing transitions again */
               end if
          else      /* no synchronization */
                    update variables of trn;
                    take the transition trn to new state;
                    set trn to -1;
          end if
     end if
end for
…
```

Penn Engineering

Property Checking in Code

- Inserted “if-then-else” checking statement inside the loop for the properties which should be satisfied all the times

- PropLRI: A[] (!Ventricle.hpenable imply Ventricle.x <= Ventricle.LRI)

```
if(Ventricle_hpenable == false){
        if(Ventricle_clock <= Ventricle_LRI){
                print “T” and clock value;
        }
        else{
                print “F” and clock value;
        }
}
```

Property Checking by Testing in Code

- Performed simulation-based black box testing
  - CPU: Microchip PIC18F4520 (40 MHz)
  - Compiler: MPLAB® C Compiler for PIC18 MCUs
  - Triggered a pin interrupt manually to represent a heart sense
- Tested with sequences of test cases covering the following scenarios
  1. Pacing–Pacing: >= LRI
  2. Pacing–Sensing (during VRP): < VRP & < LRI
  3. Pacing–Sensing (after VRP): >= VRP & < LRI
  4. Sensing–Pacing: >= HRI
  5. Sensing–Sensing: < HRI

Result of Property Checking in Code

#	Heart Event (ms)	VRP	LRI	HRI
1	P:1000.960	–	No	–
2	P:1000.448	–	No	–
3	S:910.464	Yes	–	–
4	S:1059.328	Yes	–	–
5	P:1200.832	–	–	No
6	P:1000.448	–	No	–

- Findings
  - Code execution time harms property preservation
  - Characteristics of timing properties matters
    - clock_var $\geq$ limit (e.g. PropVRP) are guaranteed
    - clock_var $\leq$ limit (e.g. PropLRI, PropHRI) are not guaranteed
  - Time deviations were bounded in the tested scenarios (E.g., $\leq$ 1ms)

Penn Engineering

Timing Tolerance Δ

- Modify the code and the model referring to the violated properties to keep the properties transferred from the model in the code satisfied

- Strategy for converting the violated properties to the satisfied ones

  Find a timing tolerance Δ by measuring deviations from the desired time

  Make the desired events happen no later than the predetermined time T by evaluating guard at Δ time units earlier than T, while not violating all other properties

Modify the Code with Δ

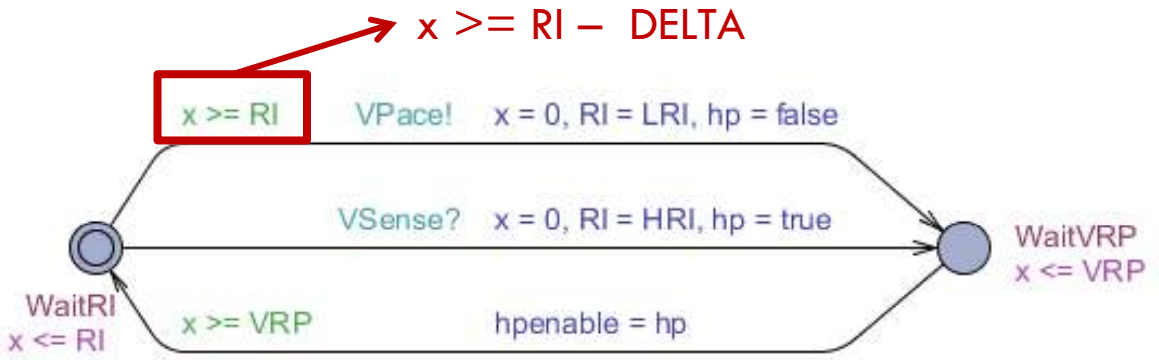
- To make guard evaluated Δ time units earlier

```
#define DELTA     (some value)
…
bool eval_guard(int trn) {
  switch(trn) {
  case 0: return (Heart_flag>0);
  case 2: return (rdClock(Ventricle_x)>=Ventricle_RI - DELTA;
  case 4: return (rdClock(Ventricle_x)>=Ventricle_VRP);
  … }
}
```

- Experimented using several values of Δ
- Three properties are satisfied in the code with Δ greater than or equal to 2ms for all tested scenarios

Modify the Model with Δ

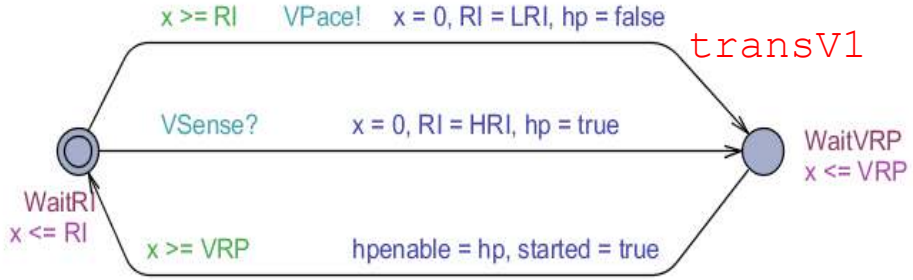
- Make the corresponding changes in the model



- Verify the modified model again w.r.t. all the properties
- Confirm that the modified model satisfies all the properties with the timing tolerance

Multi-threaded C code Structure

- A thread per each transition
- Uses semaphores for each location and each input event




```
int main(…)
{
    pthread_create(&threadV1,
         NULL, transV1, NULL);
    pthread_create(&threadV2,
         NULL, transV2, NULL);
    pthread_create(&threadV3,
         NULL, transV3, NULL);
    …
    pthread_join ();
}
```

```
void *transV1(void *ptr)
{
    while(1) {
        sem_wait(&WaitRI);
        t=wait2(&v_x,&ri);
        clearTimer(&v_x);
        current=ST_W_VRP;
        sem_post(&WaitVRP);
        sem_post(&Pace);
    }
}
```

Evaluate guard

Update

Result of Property Checking in Code

- Simulation-based black box testing on Linux
- Test inputs: Randomly generated heart sensing signals
- Execution delay was shown to be bounded by 2ms

#	Heart Event (ms)	VRP	LRI	HRI
1	P:1001	–	No	–
2	S:395	Yes	–	–
3	P:1202	–	–	No
4	S:687	Yes	–	–
5	S:1089	Yes	–	–
6	P:1202	–	–	No
7	S:419	Yes	–	–
8	S:1184	Yes	–	–
9	P:1202	–	–	No
10	S:642	Yes	–	–
11	P:1202	–	–	No
12	P:1001	–	No	–

First checking result

Δ: 2ms

#	Heart Event (ms)	VRP	LRI	HRI
1	S:872	Yes	–	–
2	P:1200	–	–	Yes
3	S:398	Yes	–	–
4	S:1153	Yes	–	–
5	S:351	Yes	–	–
6	P:1200	–	–	Yes
7	P:1000	–	Yes	–
8	S:444	Yes	–	–
9	P:1200	–	–	Yes
11	P:1000	–	Yes	–
12	P:999	–	Yes	–

Re-checking result

Penn Engineering

Related Work: Code Gen. from TA

- TIMES tool [AFP+03]
  - Generate code from TA extended with tasks for BrickOS platform
  - Under synchrony hypothesis (SH), the code synthesis is guaranteed to preserve safety properties transferred from models
  - 👍 Supports enriched TA, provides many types of automatic analysis
  - 👎 Preservation of properties is not guaranteed without SH
- ELASTIC2BRICK tool [DDR04]
  - Generate code from a simplified TA for BrickOS platform
  - Safety properties proven correct with Δ in the model are preserved
  - 👍 Formalized treatment of the synchrony hypothesis and correctness proofs
  - 👎 Limited and difficult applicability (e.g. no shared variables, no broadcasting, etc.)
- Our approach
  - Applicable without much restrictions while guaranteeing timing properties to some extents without SH

Penn Engineering

Limitations and Possibilities

- Type of timing properties
  - Considered two types of timing properties in the proposed approach
  - Combinations of complex timing properties need to be considered
- Instrumentation overhead
  - Time overhead from instrumentation may cause the code to fail in satisfying timing properties, although not in our example
  - Existing techniques for improving the performance and accuracy of time profilers based on code instrumentation can be applied to our approach
- Timing analysis on C code
  - Simple measurement technique to find timing tolerances can be replaced by WCET techniques

Limitations and Possibilities (cont.)

- Scalability
  - Semi-automatic code synthesis and manual modifications to the model can be automated by development of proper tools
- Generalization
  - Although we showed a few specific decisions for modeling languages, verification tools, code synthesis techniques, and timing analysis techniques, others can be used as long as they satisfy minimum requirements
    - Other code synthesis techniques can be used as long as it is systematic and sound
    - Other timing analysis techniques can be used as long as it can give information for finding Δ

Conclusion & Future Work

- Proposed a safety-assured development approach for real-time software
  - Combined the model-driven development methodology and the measurement-based timing analysis
  - Suggested a way to achieve property preservation within the timing tolerance in the code
  - Demonstrated the proposed approach using pacemaker software

- Future Work
  - Complement measurement-based timing analysis with formal analysis (e.g. WCET)
  - Complement testing by code level verification
  - Compare different code generation schemes

Questions?