Dependable Systems & Software Engineering

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Formal Verification of Embedded Software in Medical Devices Considering Stringent Hardware Constraints

L. Cordeiro, B. Fischer, H. Chen, J. P. Marques-Silva

Lucas Cordeiro
lcc08r@ecs.soton.ac.uk

Agenda

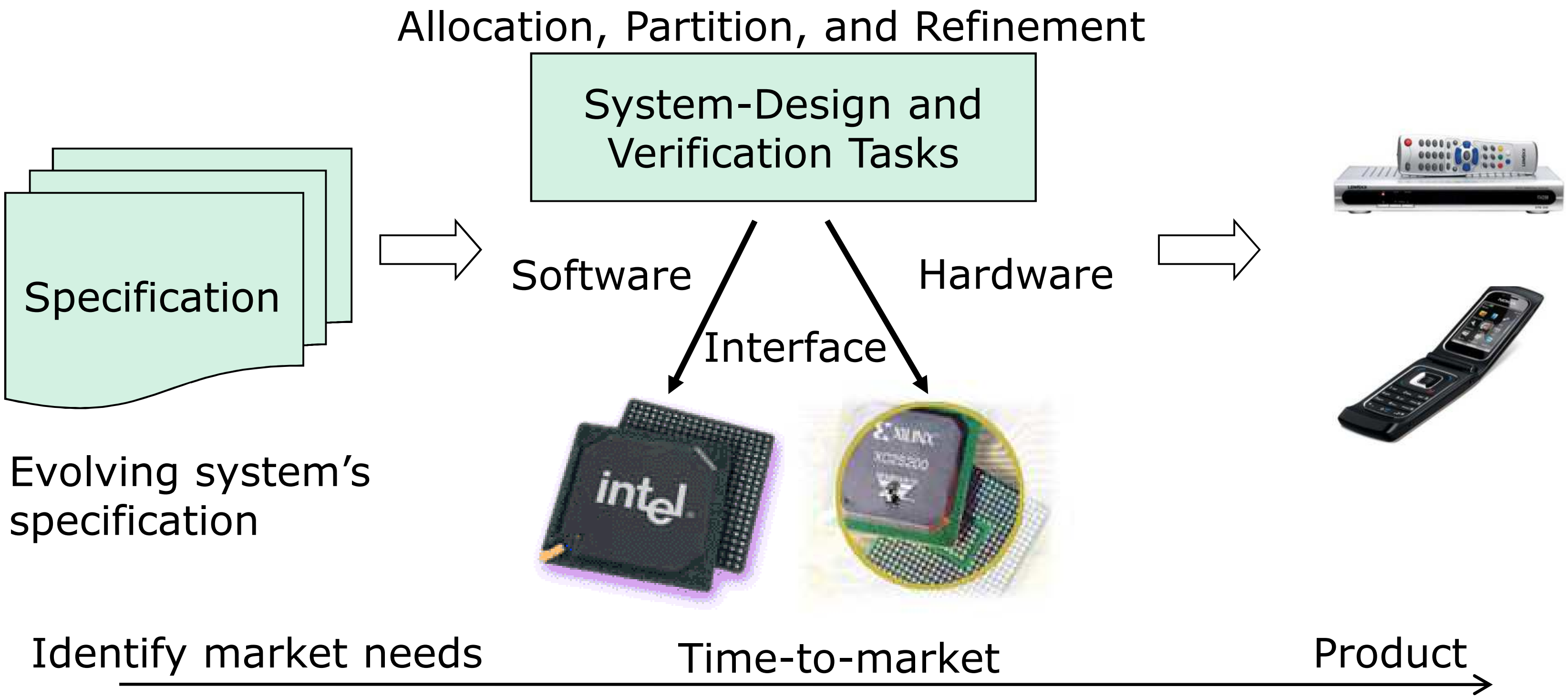
UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Introduction
- Formal Verification Methodology
- Case Study and Experimental Results
- Conclusions and Future Work

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Introduction

- Design HW/SW that implements functionalities and satisfies constraints.



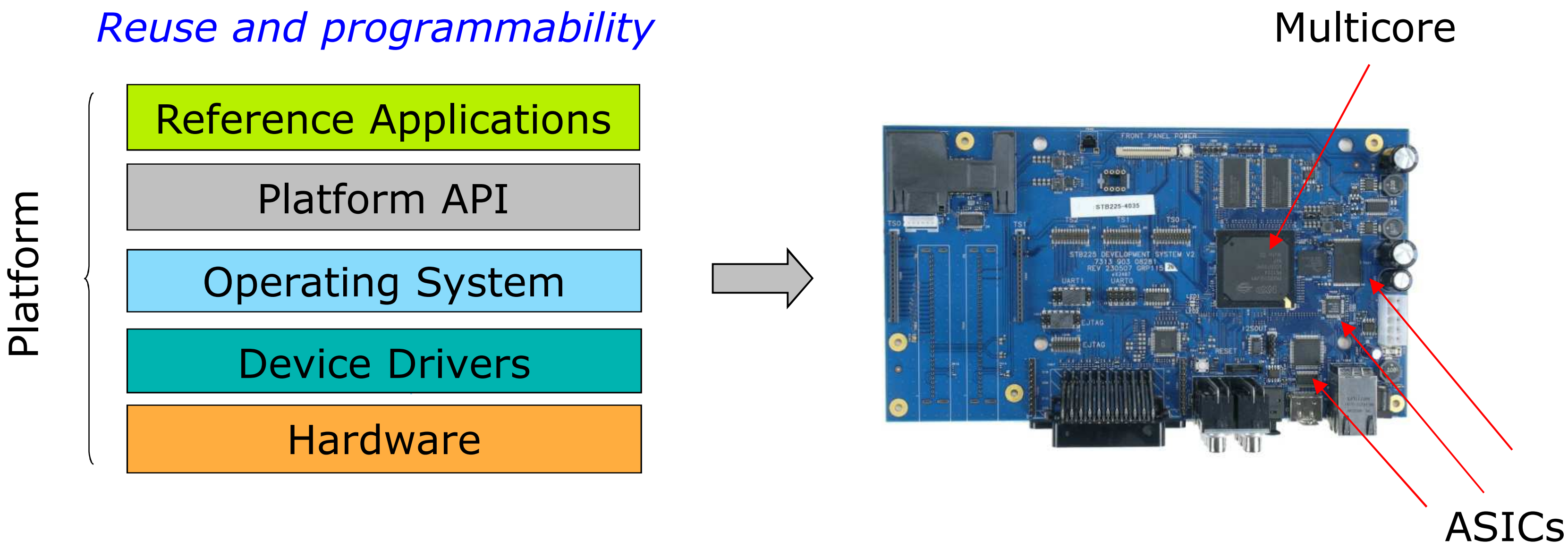
- The complexity of ESW increased in embedded products

Platform-Based Design

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Design methodologies looks for solutions to reduce time-to-market, manufacturing and design costs.

Reuse and programmability



- The size of ESW is increasing to millions of LOC.

- Software builds are produced on a weekly or daily basis.

Verification Methodologies and Challenges

- State-of-the-art ESW verification methodologies aim to:

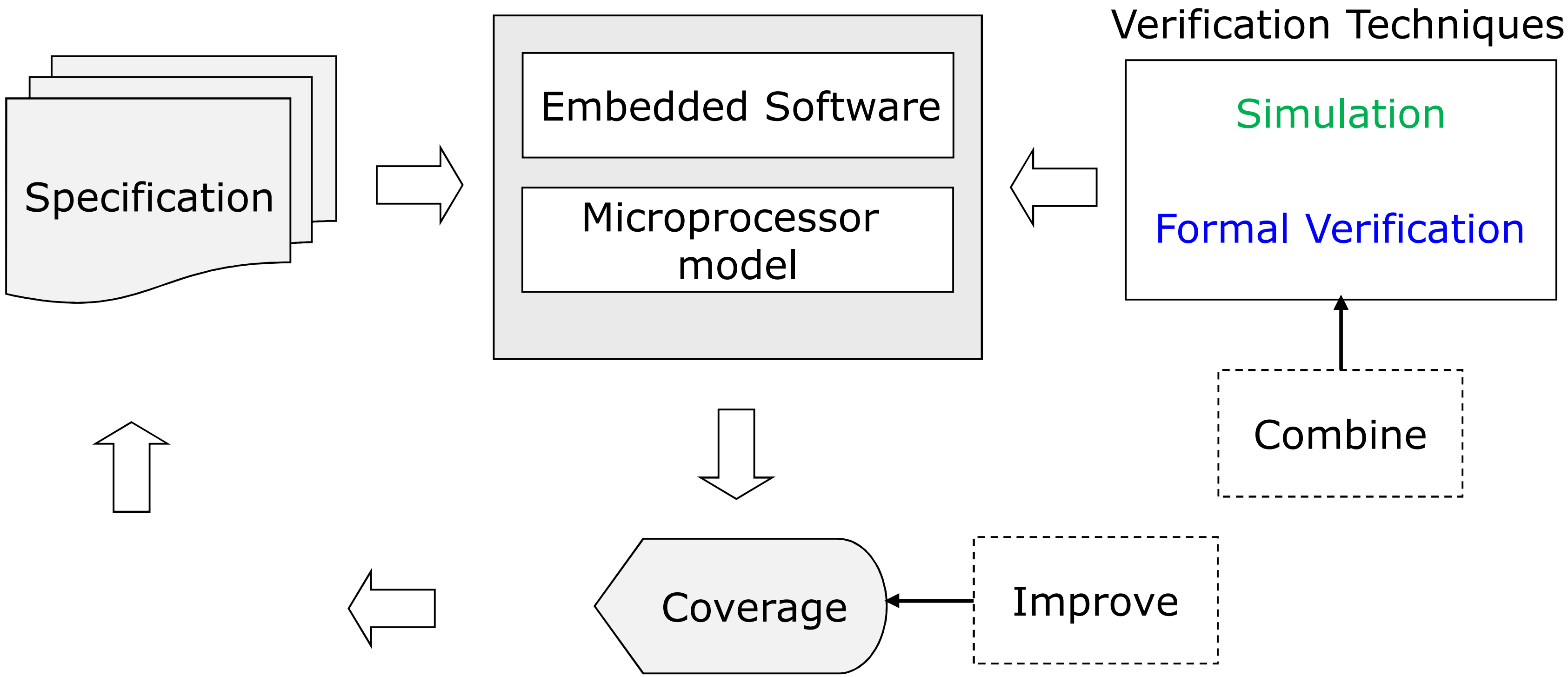
i. Generate test vectors (with constraints)
ii. Use assertion-based verification
iii. Use the high-level processor model during simulation

- Verification of embedded systems raises some additional challenges:

i. Meet the timing constraints
ii. Handle software concurrency
iii. Platform-dependent software
iv. Legacy designs (written in low-level languages)

Objective of this work

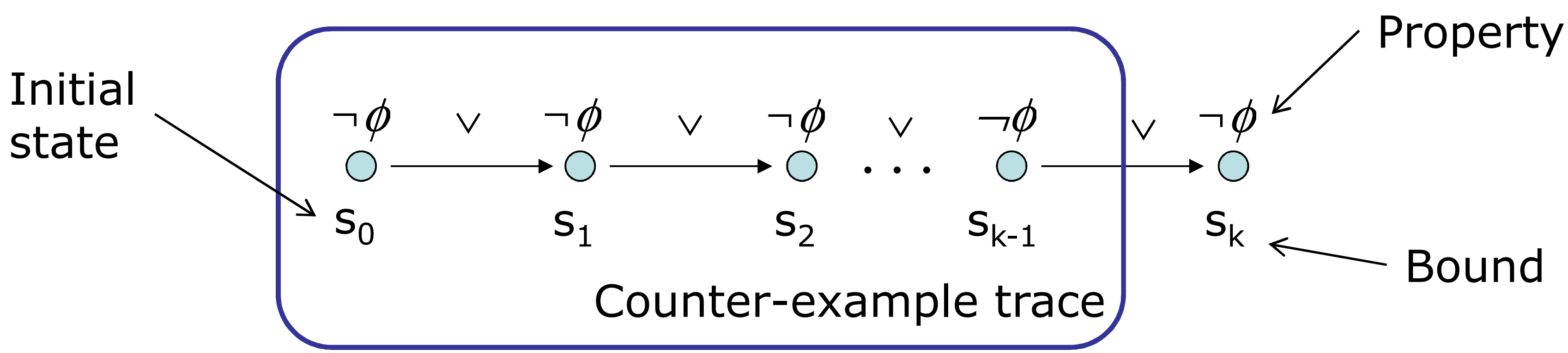
• Improve coverage and reduce verification time by combining static and dynamic verification.



Bounded Model Checking

- The basic idea of BMC is to check the negation of a given property ϕ at a given depth.

- Given a transition system M, a property ϕ and a bound k:

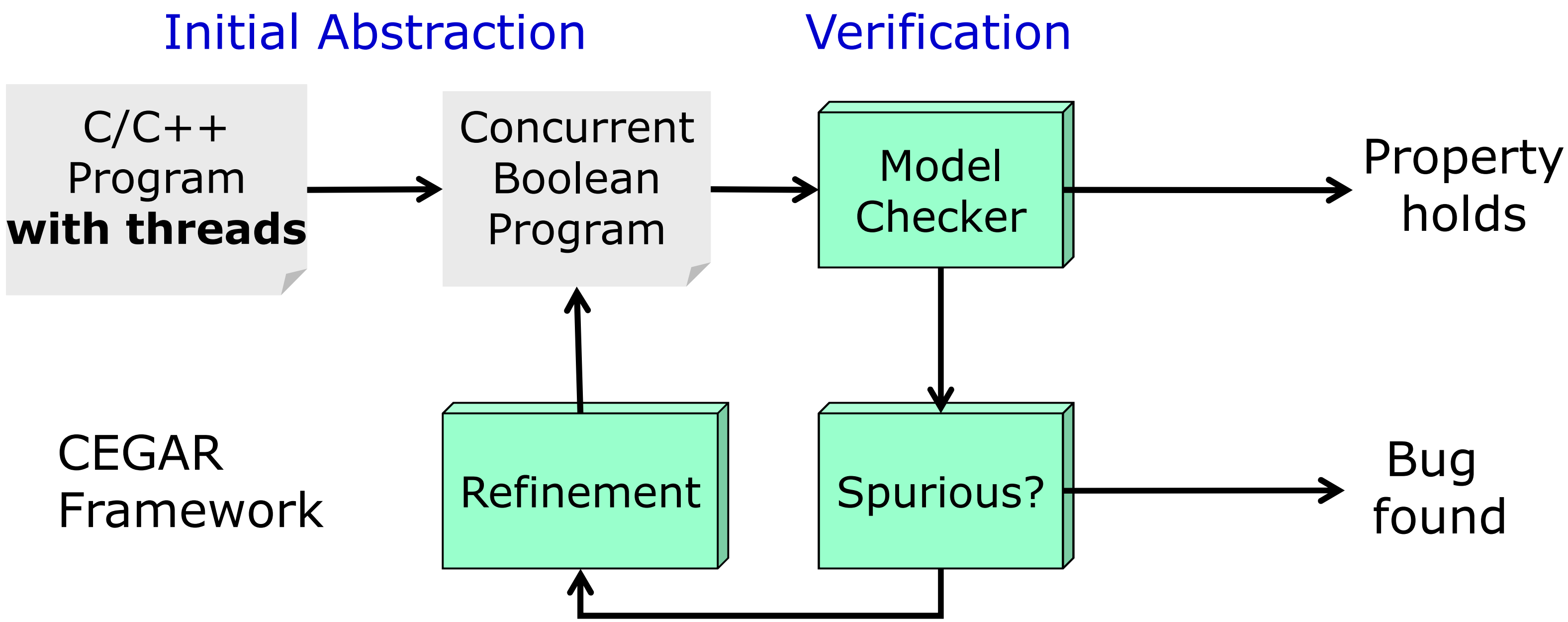


- BMC unrolls the design *k* times and translates it into a verification condition ψ such that ψ is satisfiable *iff* ϕ has a counter-example of depth less than or equal to *k*.

Predicate Abstraction

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- It abstracts data by only keeping track of certain predicates to represent the data.



- Conservative approach reduces the state space, but generates spurious counter-examples.

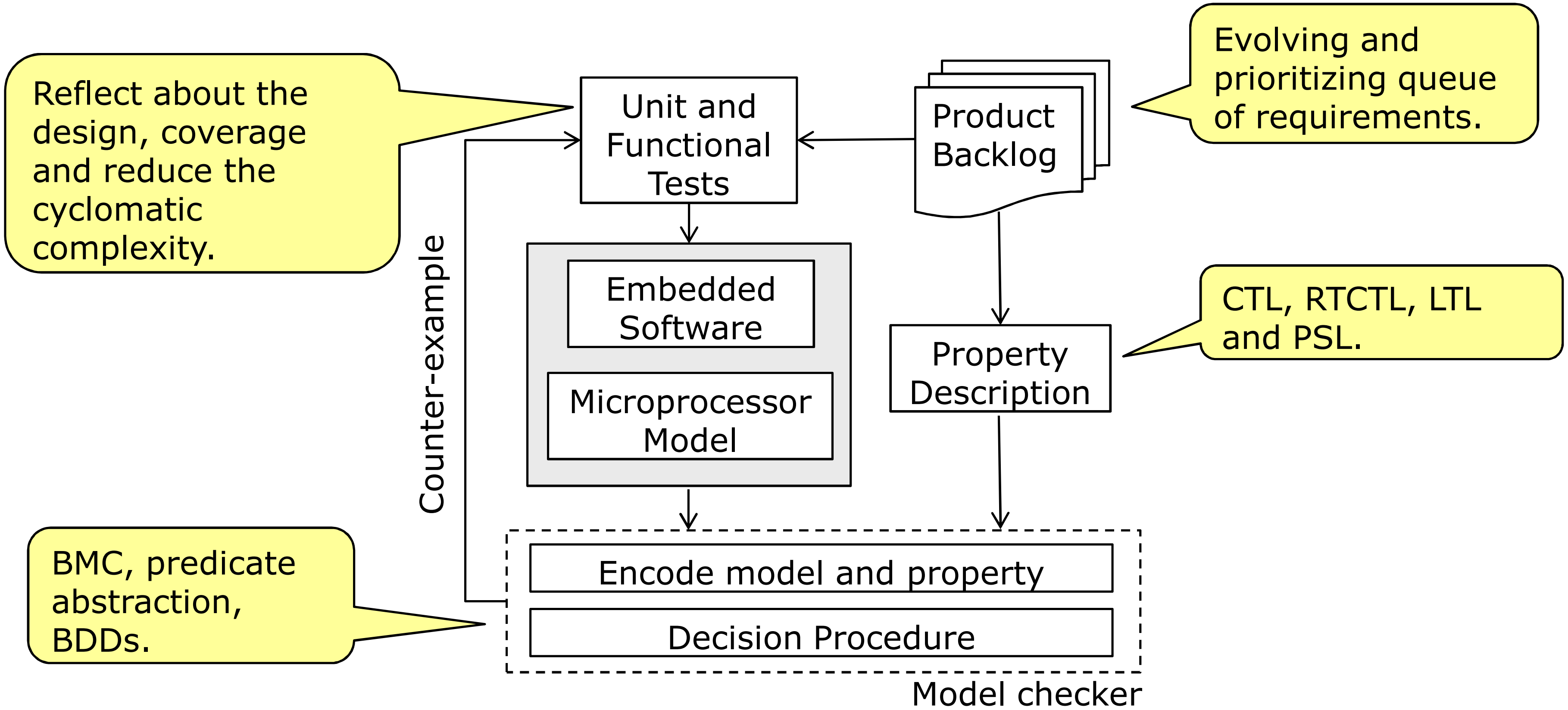
Agenda

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Introduction
- Formal Verification Methodology
- Case Study and Experimental Results
- Conclusions and Future Work

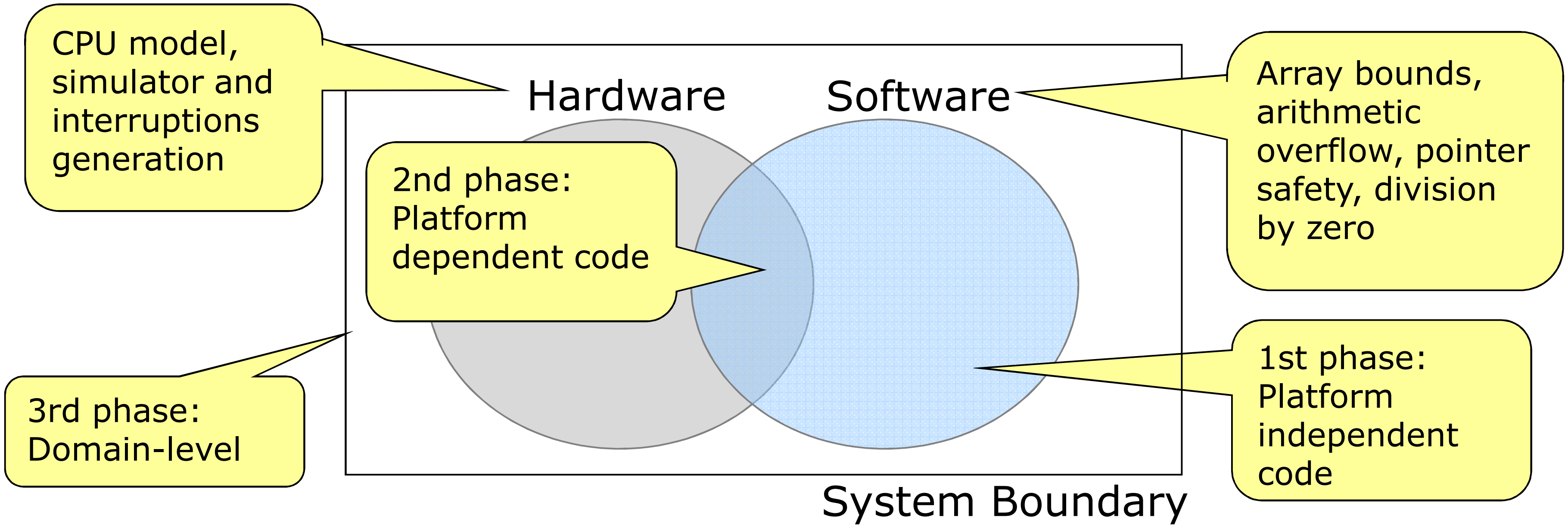
Verification Methodology

Consider not only higher levels of abstraction, but also the HW/SW interface.



Proposed Approach

- In complex embedded systems, there will be modules that depend on the hardware and others that do not.



- To reason about temporal properties to assure the *correctness* and *timeliness* of the design.

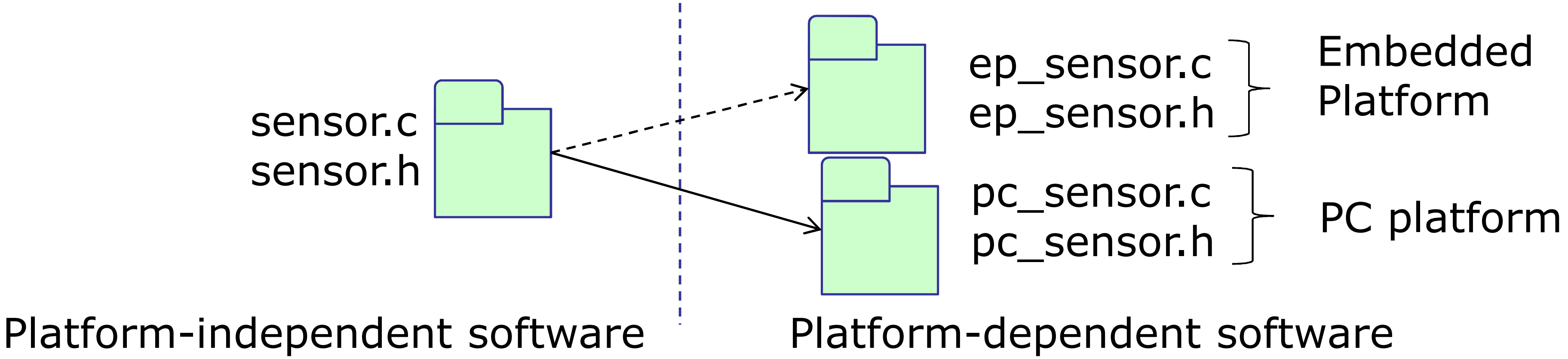
Platform-Independent Software Verification

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Implement small changes in the ESW to be able to:

i. Use model checkers;
ii. Perform automated unit tests;
iii. Run the ESW on the target platform.

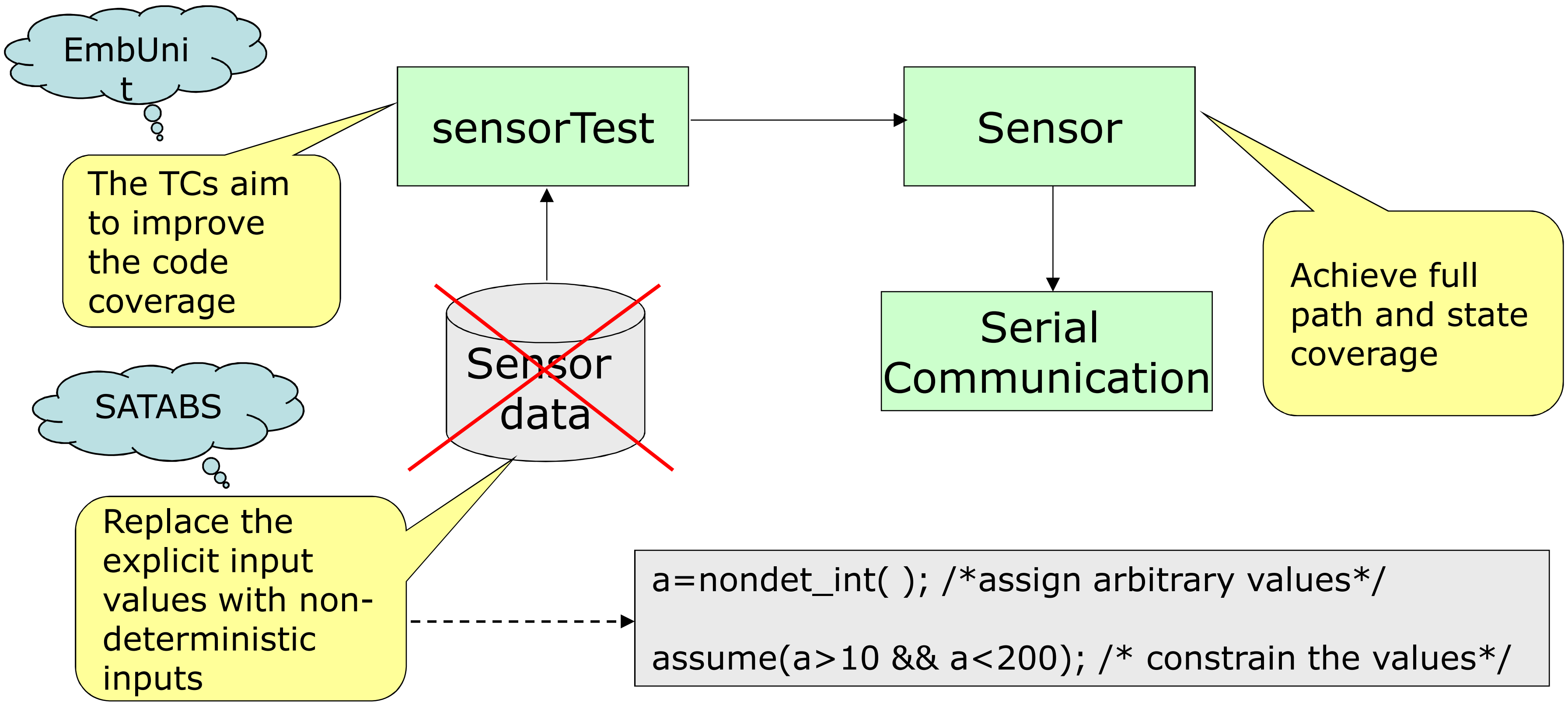
- Include the platform-dependent software in lower level driver files:



Platform-Independent Software Verification

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- We separate into two software classes: pure and driven by the environment.



Platform-Dependent Software Verification

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Specify properties based on C's *assert* macro using the microprocessor model.

```
Fml ::= Fml con Fml | ~Fml | Atm
con  ::= AND | OR | XOR
Atm ::= Trm rel Trm | true | false
rel   ::= < | <= | > | >= | = | !=
Trm := var | const
```

Examine the call stack and interpret the counterexample

CBMC

```
struct module_tc {
   unsigned int tl0;
}
extern struct module_tc oc8051_tc;
oc8051_tc.tl0=TLOW;
for(cycle=0; cycle<n; cycle++)
   next_timeframe();
assert(oc8051_tc.tl0==Y);
…
```

Hold the value of tl0 register

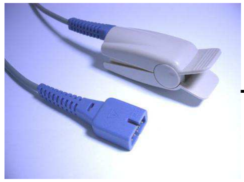
Load timer register

Change the state of the registers in the verilog model

Check user-specified assertions

Domain-Level Verification

We use RTCTL to specify properties that involve time bounds.



Sensor → Compute HR and SpO2 →

m $\longrightarrow$ n

$$T = \{ t \in \aleph \mid m \le t \le n \}$$

compute_expr :: **MIN [rtctl_expr , rtctl_expr]** (shortest path)
| **MAX [rtctl_expr , rtctl_expr]** (longest path)

rtctl_expr :: **EBF** m..n p| **ABF** m..n p| **EBG** m..n p| **ABG** m..n p
| **E** [p **U** m..n q] | **A** [p **U** m..n q**]**

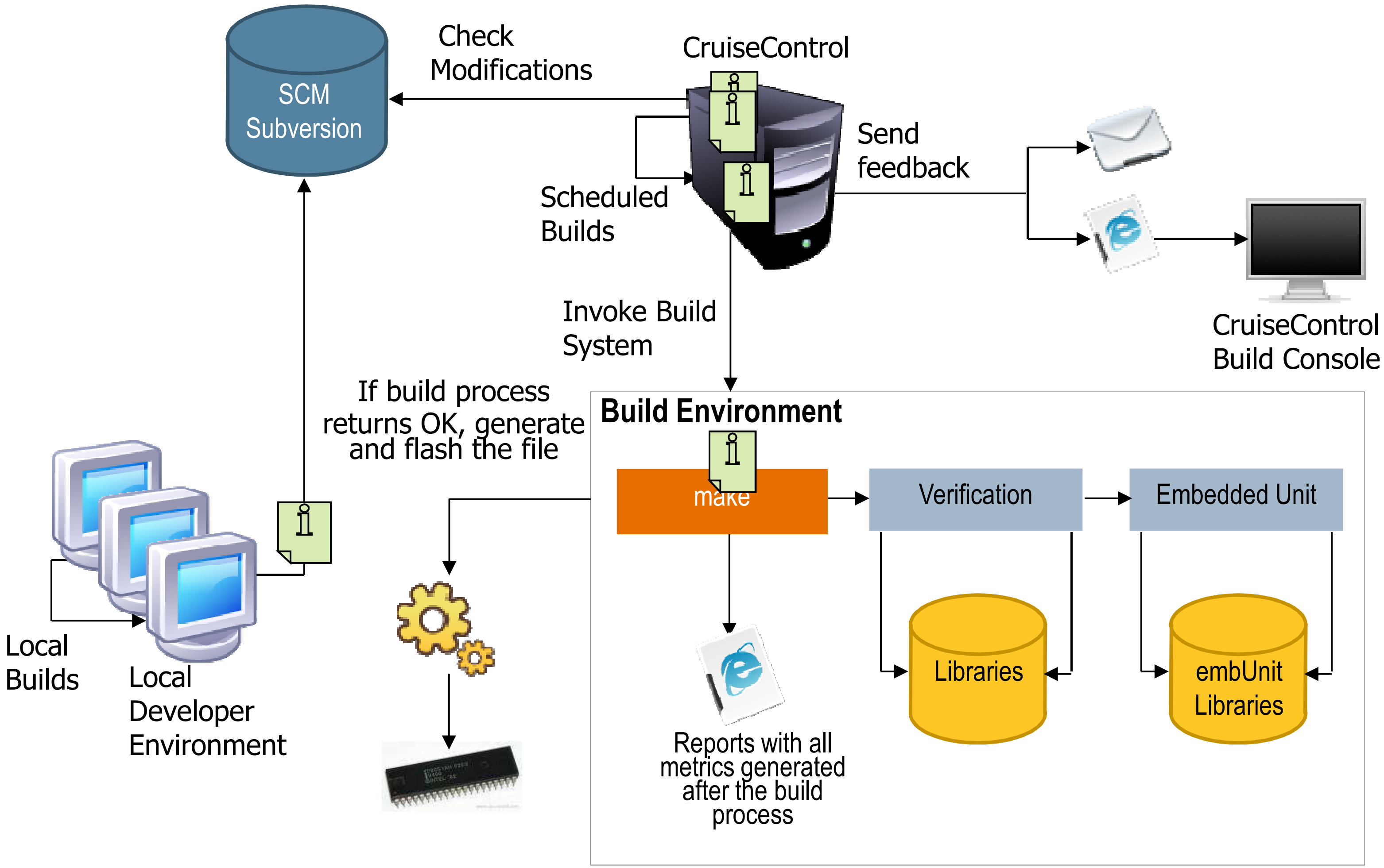
NuSMV2

Simulation

Log system

```
Timer   Component   Function:Filename(line)
12320   c_LCD   ->  LCD_Driver_InitModule: lcd_class_driver.c(85)
12789   c_LCD   ->  LCD_WriteData: lcd_class_driver.c(90)
13452   c_LCD   ->  LCD_InterfaceDescriptor: lcd_class_interface.c(102)
14216   c_LCD   ->  LCD_InterfaceContext_Create: lcd_class_interface.c(18)
14834   c_LCD   ->  LCD_initialize: lcd_class_interface.c(80)
```

Infrastructure



Agenda

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Introduction
- Formal Verification Methodology
- Case Study and Experimental Results
- Conclusions and Future Work

Medical Device Case Study

University of Southampton
School of Electronics and Computer Science

- The pulse oximeter measures the oxygen saturation and cardiac frequency.

  i. Show SpO2 and HR on each second.

  ii. Change the alarm configuration.

  iii. User interface (keyboard and a graphical display).

  iv. The design is highly optimized for life-cycle cost and effectiveness.

- Typical of many embedded real-time systems.

Formal Verification using Model Checking

University of Southampton
School of Electronics and Computer Science

- How many bugs can you find in this ANSI-C code fragment? (the compiler compiles it without errors)

```
#define BUFFER_MAX 6400

typedef int Data8;
typedef unsigned int uData8;

static char buffer[BUFFER_MAX];
static Data8 next=0;
static uData8 buffer_size=BUFFER_MAX;

void insertLogElement(Data8 b) {
    if (next < buffer_size) {
        buffer[next] = b;
        next = (next+1)%buffer_size;
}
```

First bug: the array
buffer is a char data
type
is a
type

Second bug: there is a
division by zero in
(next+1)%buffer_size

(pre-production code)

Model Checking with NuSMV2

UNIVERSITY OF Southampton
School of Electronics and Computer Science

NuSMV2 accepts models in NuSMV language and system properties in CTL, Real-Time CTL, LTL and PSL.

Property *(a)*: ensure that the buffer does not overflow.

```
MODULE log
VAR
   buffer_size : 0..255;
   nextptr : 0..255;
DEFINE
   nextptr_condition := nextptr < buffersize;
ASSIGN
   init(nextptr) := 0;
   next(nextptr) := case
   nextptr = nextptr_condition & buffer_size > 0
               :((nextptr+1) mod buffer_size);
   1 : nextptr;
   esac;
PSLSPEC AG (nextptr <= buffer_size)
```

NuSMV2 found a division by zero and a typecast overflow.

Ensure that on all paths, at all states on each path the formula holds

Specifying Complex Properties in CBMC and SATABS

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- We specified property (b) in LTL and translated it into Buechi Automata.

Property *(b)*: check the data flow to compute the HR value provided by the pulse oximeter sensor hardware.

- Property (b) can be expressed as:

$$AG\left(p \rightarrow Fr\right)$$

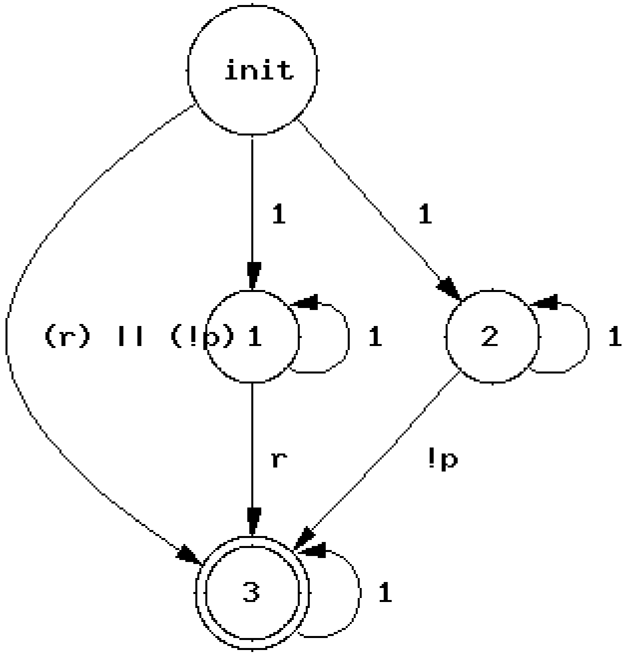
- Let *p* denote the state that the buffer contains HR. Let *r* denote the state that defines the respective HR value.

- Any state containing the HR raw data is eventually followed by a state representing the respective HR value.

Specifying Complex Properties in CBMC and SATABS

Example:

$AG(p \rightarrow Fr)$




```
…
switch (state) {
   case T0_init:
      …
      break;
   case accept_S1:
      …
      break;
   …
}
…
```

Property in LTL *Buechi Automata* *C code*

Experimental Results

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- The pulse oximeter ESW contains approximately 3500 lines of ANSI-C c .

First phase: one bug related

First phase: one bug related to pointer safety
Third phase: one bug related to timing constraints

First phase: one bug related to division by zero and another bug related to typecast overflow

							Dynamic Verification	
					SMV2		EmbUnit	
Module	P				erties	V.T.(s)	Test Cases	V.T. (μs)
MenuApp					53	5.4	62	130
Sensor	224				10	3.7	42	403
LCD	22				8	4.1	6	6
Serial	5				8	4.6	5	8
Timer	12				7	4.9	7	12
Keyboard	1	0.02	1	0.02	16	4.8	10	18
Log	14	2	6	1	4	5.4	10	48
Total	**310**	**33.04**	**662**	**134.06**	**106**	**32.9**	**139**	**625**

- The most relevant related work verified dynamically ESW from automotive domain with approximately 3000 lines of C code in 34388 seconds (~9 h) using SystemC models [Lettnin'08].

Conclusions and Future Work

- We have combined static and dynamic verification for "pure" and hardware-related embedded software.

- Test driven development helps reduce the cyclomatic complexity and alleviates the state explosion problem.

- The proposed methodology allowed us to find undiscovered bugs.

- We intend to verify formally ANSI-C and SystemVerilog using SAT Modulo Theories solvers.

- We aim at defining a subset of Real-Time CTL and PSL to verify more complex properties in embedded software.