

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

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

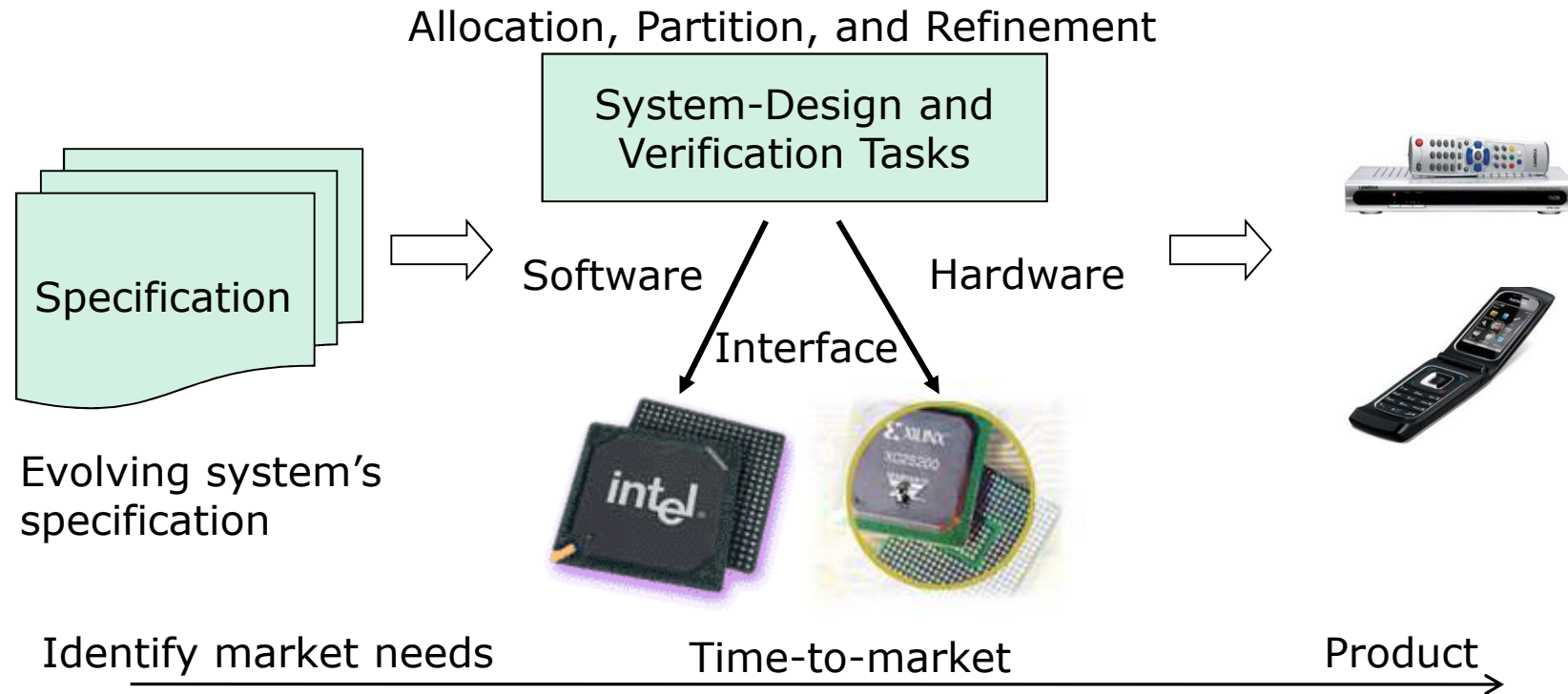
Lucas Cordeiro  
[lcc08r@ecs.soton.ac.uk](mailto:lcc08r@ecs.soton.ac.uk)

# Agenda

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

# Introduction

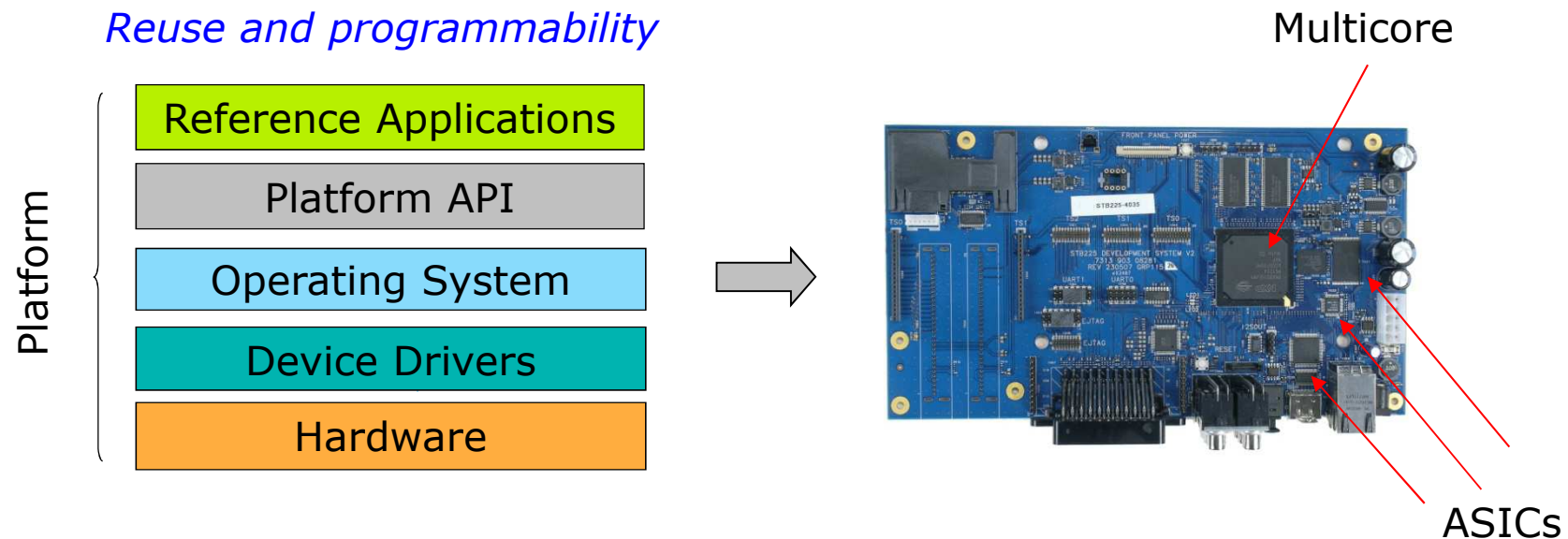
- Design HW/SW that implements **functionalities** and satisfies **constraints**.



- The **complexity** of ESW increased in embedded products

# Platform-Based Design

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



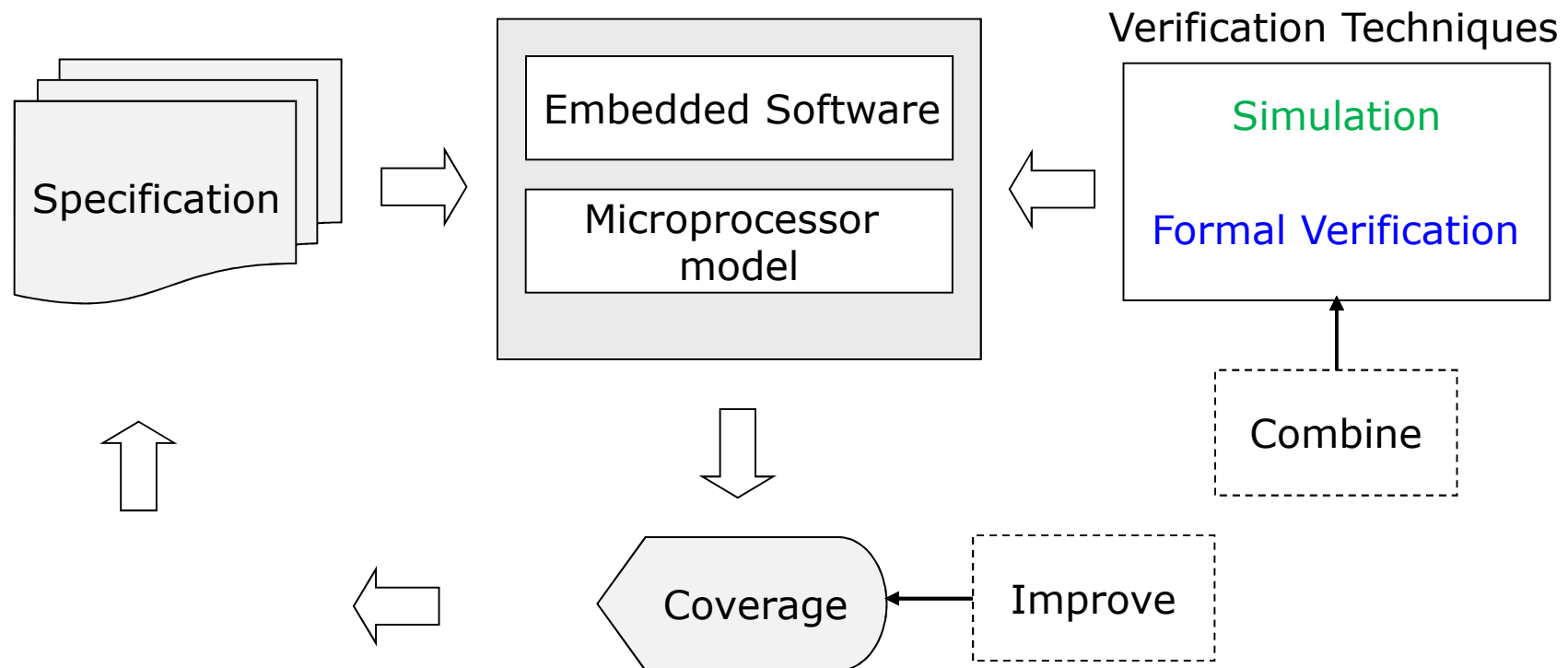
- The **size of ESW** is increasing to millions of LOC.
- Software builds are produced on a **weekly or daily basis**.

# Verification Methodologies and Challenges

- State-of-the-art ESW **verification methodologies** aim to:
  - Generate test vectors (with constraints)*
  - Use assertion-based verification*
  - Use the high-level processor model during simulation*
- Verification of embedded systems raises some **additional challenges**:
  - Meet the timing constraints*
  - Handle software concurrency*
  - Platform-dependent software*
  - Legacy designs (written in low-level languages)*

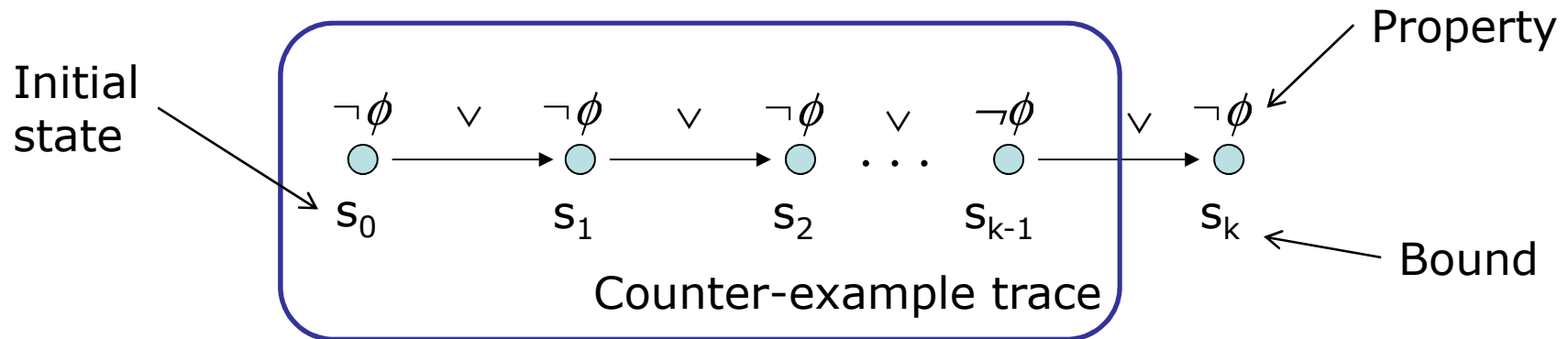
# Objective of this work

- Improve coverage and reduce verification time by combining **static** and **dynamic verification**.



# Bounded Model Checking

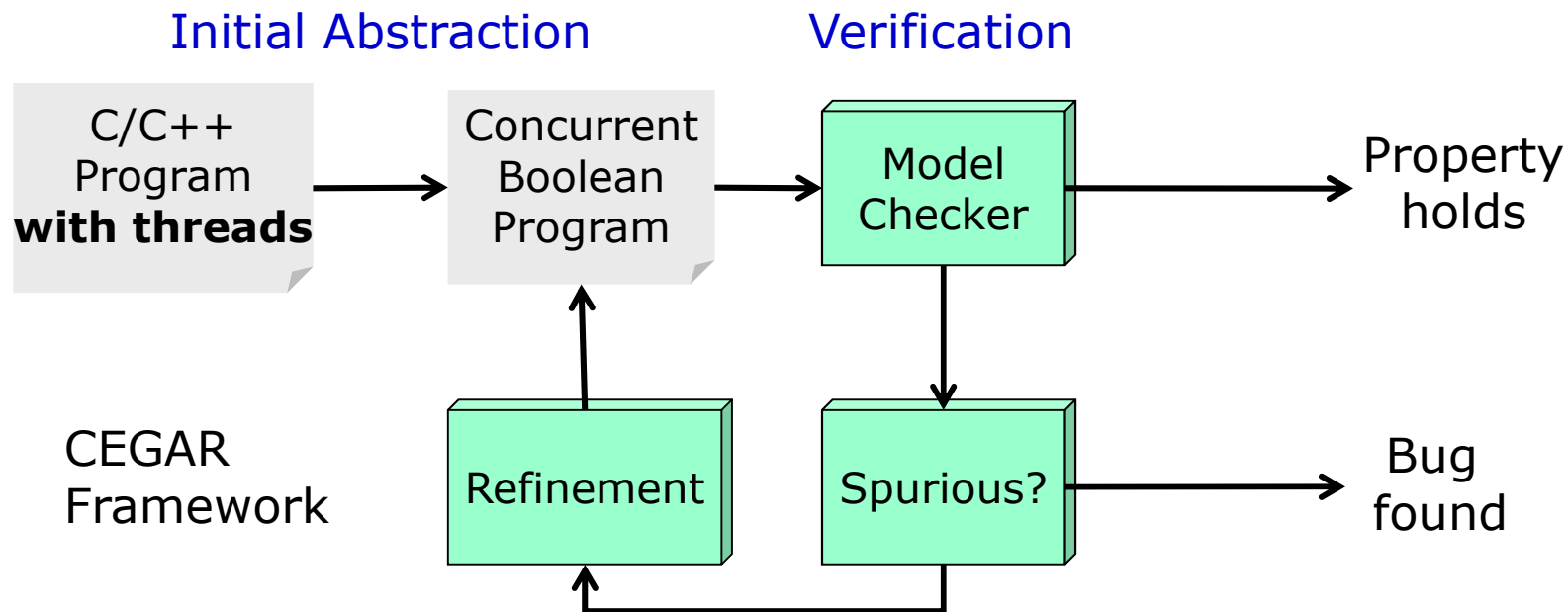
- The basic idea of BMC is to check the **negation** of a given property  $\phi$  at a given depth.
- Given a transition system  $M$ , a property  $\phi$  and a bound  $k$ :



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

# Predicate Abstraction

- 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**.

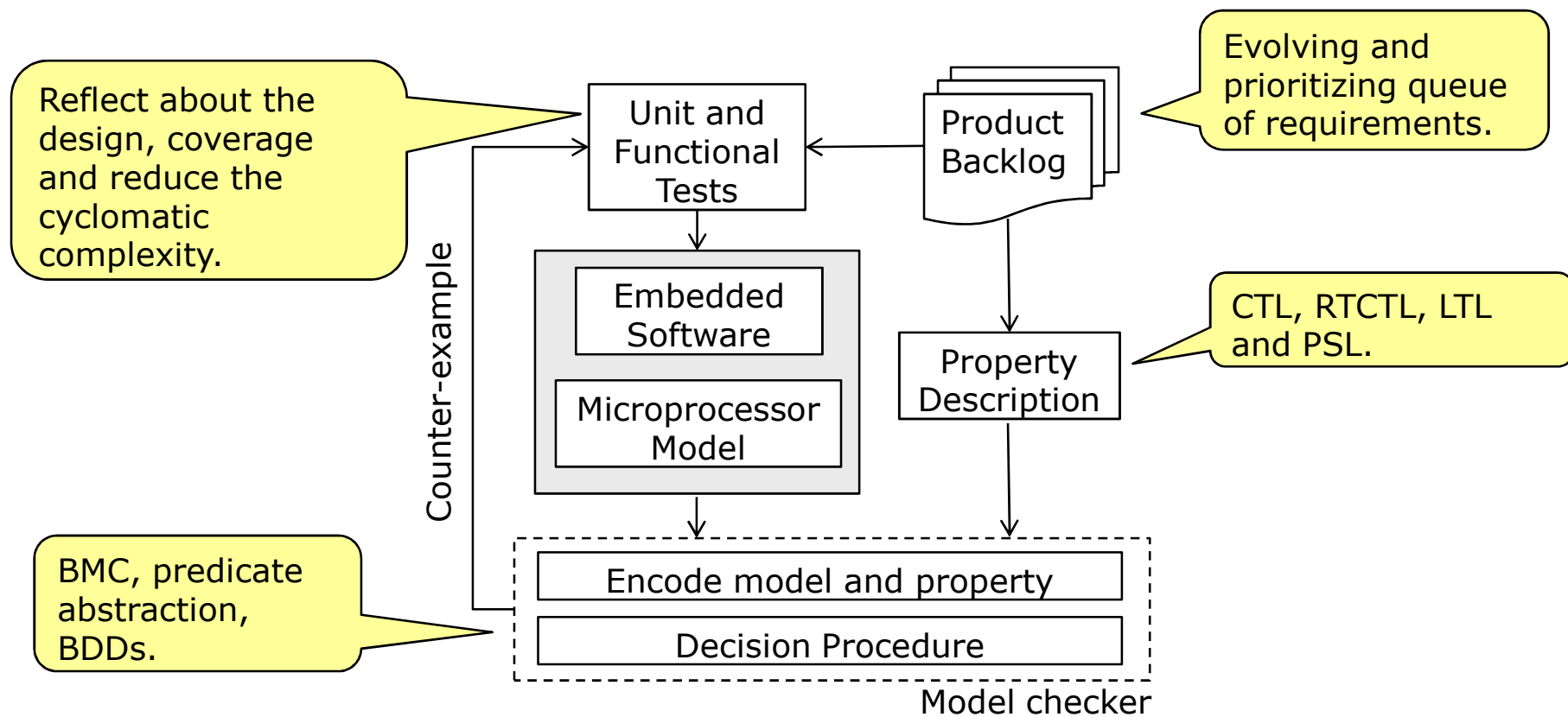


# Agenda

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

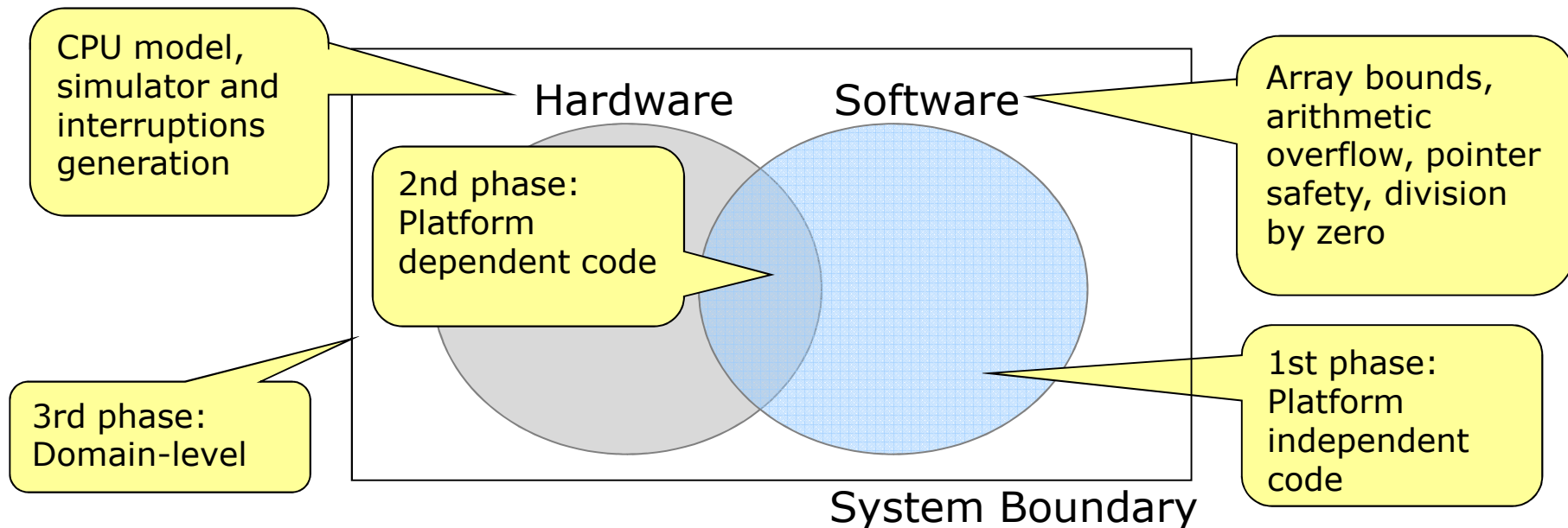
# Verification Methodology

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



# Proposed Approach

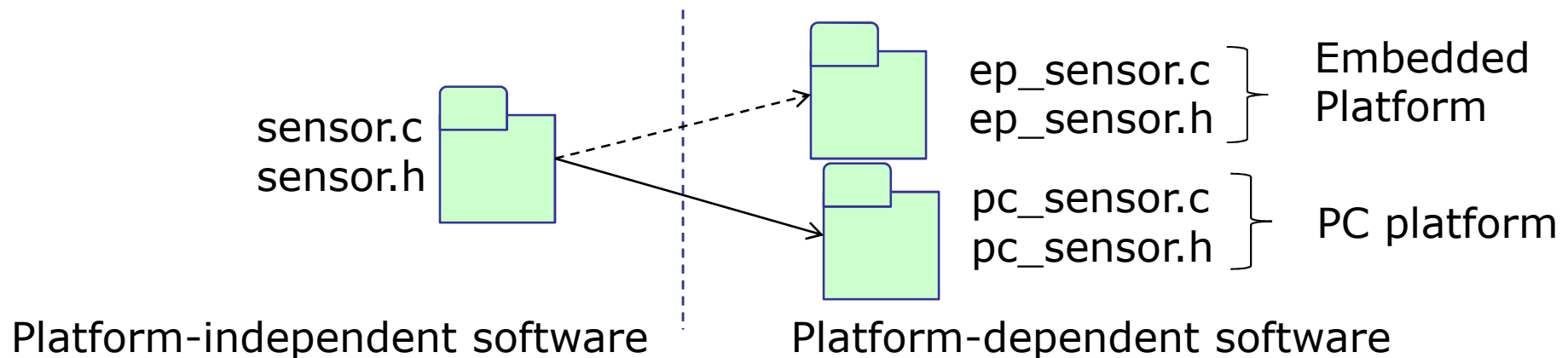
- In complex embedded systems, there will be modules that **depend on the hardware** and others that do not.



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

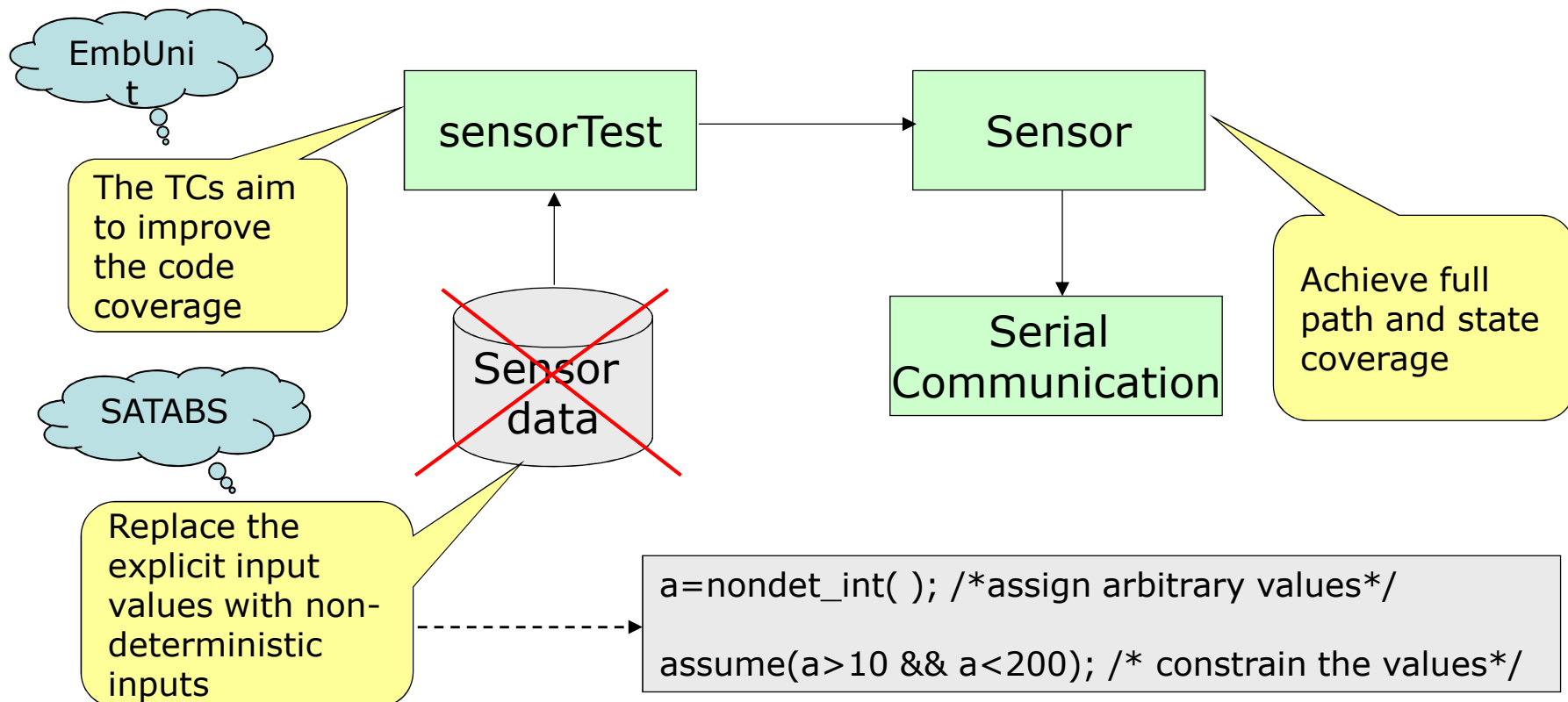
# Platform-Independent Software Verification

- 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

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



# Platform-Dependent Software Verification

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

Examine the call stack and interpret the counterexample

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

CBMC

Hold the value of tl0 register

Load timer register

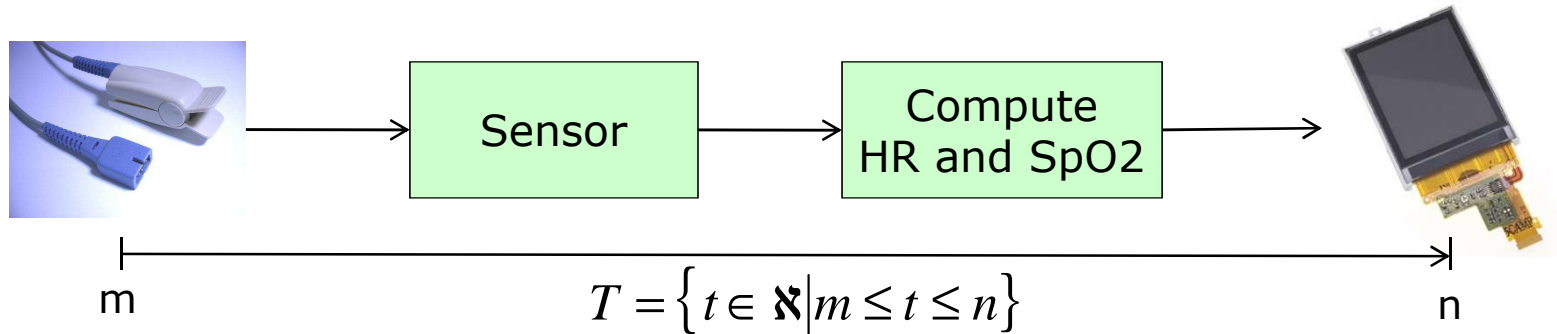
```
struct module_tc {
    unsigned int tl0;
}
extern struct module_tc oc8051_tc;
oc8051_tc.tl0=TLOW;
for(cycle=0; cycle<n; cycle++)
    next_timeframe();
assert(oc8051_tc.tl0==Y);
...
```

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**.



compute\_expr :: **MIN** [ rtctl\_expr , rtctl\_expr ] (shortest path)  
 | **MAX** [ rtctl\_expr , rtctl\_expr ] (longest path)

NuSMV2

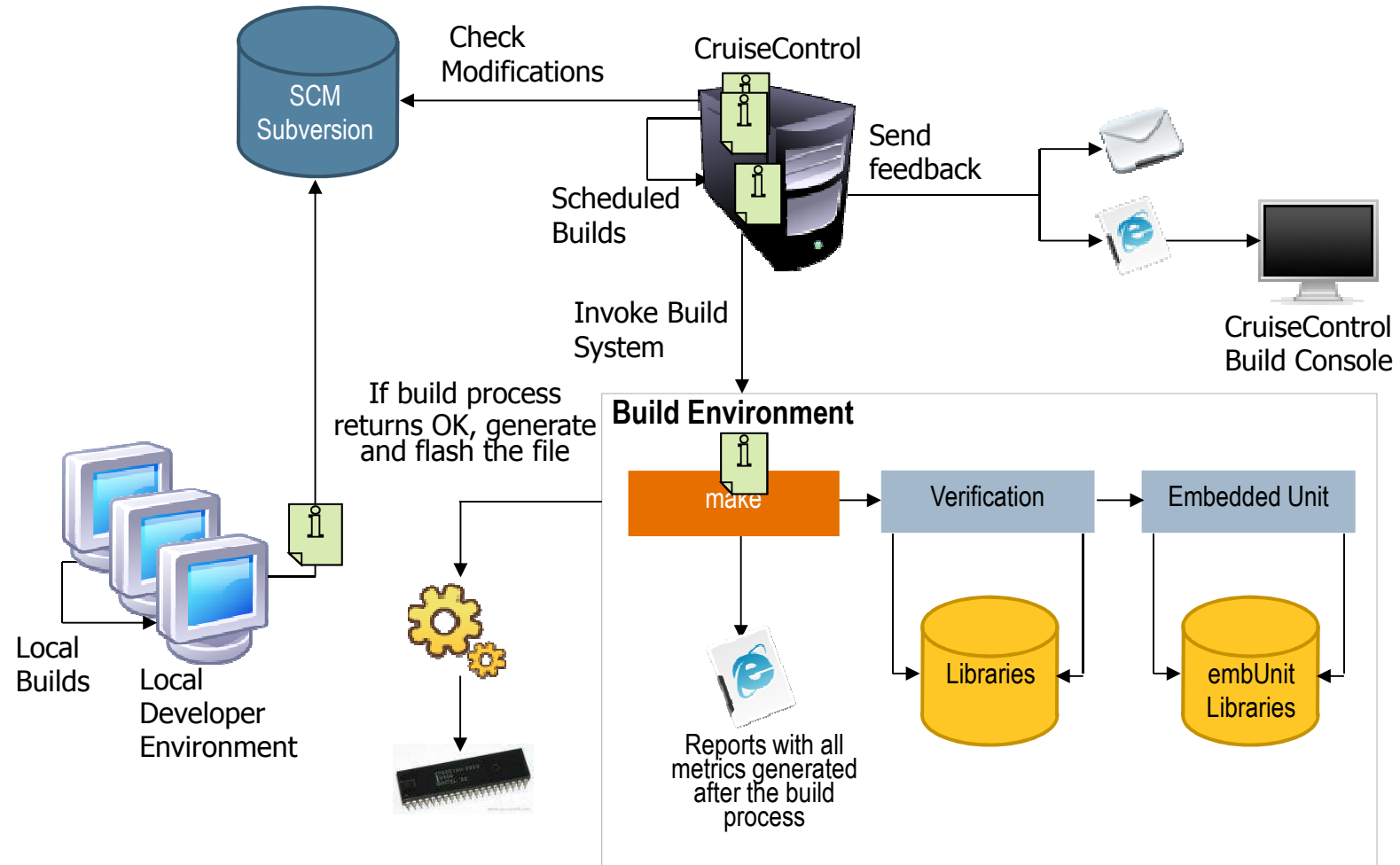
rtctl\_expr :: **EBF** m..n p | **ABF** m..n p | **EBG** m..n p | **ABG** m..n p  
 | **E** [p **U** m..n q] | **A** [p **U** m..n q]

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

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

# Medical Device Case Study

- 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

- 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

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

- 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.

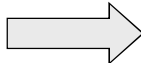
- Property (b) can be expressed as:

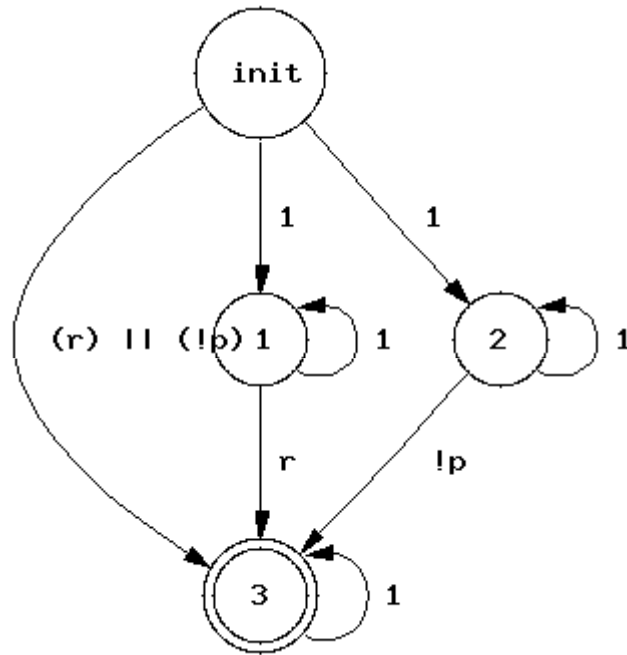
$$AG(p \rightarrow Fr)$$

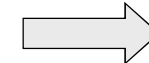
- Let  $p$  denote the state that the buffer contains HR. Let  $r$  denote the state that defines the respective HR value.
- Any state containing the HR raw data is **eventually followed** by a state representing the respective HR value.

# 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

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

Module	Lines	Properties	SMV2 V.T.(s)	Dynamic Verification	
				Test Cases	V.T. ( $\mu$ s)
MenuApp	22	0.02	5.4	62	130
Sensor	224	1	3.7	42	403
LCD	22	0.02	4.1	6	6
Serial	5	1	4.6	5	8
Timer	12	0.02	4.9	7	12
Keyboard	1	1	4.8	10	18
Log	14	6	5.4	10	48
<b>Total</b>	<b>310</b>	<b>33.04</b>	<b>662</b>	<b>139</b>	<b>625</b>

First phase: one bug related to **pointer safety**

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**

- The most relevant related work verified dynamically ESW from automotive domain with approximately **3000 lines of C code in 34388 seconds** ( $\sim 9$  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.