Teaching embedded software design with radSUITE

Franco Fummi (University of Verona, EDALab, Italy)
Cristina Marconcini (STM Products, Italy)
Graziano Pravadelli (University of Verona, EDALab, Italy)

October 3rd, 2013 - Montreal, Canada
Outline

• Motivations
• Training program and HW kit
• Application to Embedded SW life-cycle
• Case study
• Concluding remarks
Motivations

- There is the feeling that university courses are uselessness with respect to knowledge required in industry.
Motivations

- University courses focus on a single aspects of Embedded Systems (ES) design flow (modeling, verification and deployment flow)

- Case studies used in laboratory are too simple to allow students to experiment real-case issues

- EDA tools used for laboratory exercises are academic prototype, very useful for didactic purposes but far enough from commercial tools

Students cannot capture in practice the real complexity of managing the ES design flow

- Students work on the same case study throughout all steps, it should be complex enough to allow experiment the most ES design related issues

- Methodologies with theoretical basis that must have a clear correspondence with the functionality of tools used during laboratory activities

- Use of interoperable tools to automatize the entire design flow for ES
Motivations - industry training

- Technology changes are persistent in EDA with new theories, methodologies and tools
  - Senior employees are unwilling to accept changes in their modus operandi with new tools
  - Adoption of new formalisms and new methodologies is difficult

✓ A new framework for ESW modeling, verification and deployment
  - Efficacy of a commercial tool
  - Easiness of use typical of didactic prototypes
University Program (UP)

• What is
  – Software tools for MDD and Functional Verification (Educational free license)
  – HW starter kit: Vulcano G20 ARM9 with eCos
  – Training material with PPT presentations
    • MDD and Functional verification main concepts
    • How to perform MDD and Functional Verification with radSUITE
    • radSUITE training
    • Case study

• The UP has been recently adopted by courses taught at the University of Naples, Florence, L’Aquila and Trieste
University Courses Program

1. Introduction to embedded SW
2. Introduction to real-time operating systems
3. Embedded SW modeling
   a) Modeling alternatives
   b) Model-driven design
   c) Automatic synthesis of the application
4. Embedded SW verification
   a) Simulation and test pattern generation
   b) Assertion-based verification
   c) The Property Specification Language
5. Embedded SW deployment
   a) Platform-based design
   b) Embedded platform architecture
   c) HW abstraction layer
   d) Cross-compilation tool chain

Courses of the graduate degree in Computer Science and Engineering at University of Verona

*Embedded System Design*
*Design Automation of Embedded System*
Industry Courses Program

1. Model-based design
   - OOP approach / Hierarchy
   - Modularity and reuse

2. Model Based Design and UML
   - State machines

3. Modeling
   - Application structure
   - Application behavior with methods and State machine

4. Guidelines for a Quality oriented development

5. Target integration
   - Scheduling of the generated code
   - Typical scenarios: with / without OS
   - Data communication and Data usage
   - Time resolution
   - HAL approach

5. Functional Verification, preliminary concepts of ABV

6. The Property Specification Language

7. Requirements formalization
   - Learn-by-doing approach
   - PSL templates

8. Verification and debugging
   - Unit testing
   - Integration testing
   - System testing

9. Refinement based approach
Starter Kit contents - HW

Breakout-G20 evaluation board

Olimex ARM-USB-TINY-H JTAG interface

Vulcano-G20 CPU module

• Low cost, general purpose single-board computer, suitable for a wide range of applications
• MCU Atmel ARM 9 - AT91SAM9G20 400 Mhz
Starter Kit contents - SW

radCASE Template

eCos tool chain

Case study application
Starter Kit contents – target tool chain

- **Yagarto** GCC and Tools: cross-compiler
- **OpenOCD**: GDB debugger
- **Drivers:**
  - Olimex ARM-USB-TINY-H JTAG
  - Breakout-G20 USB serial
- **eCos**: operating system
  - Binary image and headers in the Template project
- **Eclipse** IDE (for advanced API debugging)
  - Preconfigured workspace
Case study - Application

- This project is a simplified concept of a typical home bread maker machine, with a recipe program that controls a heater and a mixer motor.
- The application uses some of the Vulcano-G20 Input / Output devices, and exploits the USER1 button for start and stop, and the three LEDs to show the status.

<table>
<thead>
<tr>
<th>Input / Output</th>
<th>Device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start / Stop</td>
<td>USER1</td>
<td>Push button</td>
</tr>
<tr>
<td>Active indicator</td>
<td>LED (D1)</td>
<td>Main activity status</td>
</tr>
<tr>
<td>Probe</td>
<td>ADC 0 (J14 pin 2)</td>
<td>Temperature probe</td>
</tr>
<tr>
<td>Heating indicator</td>
<td>LED (D7)</td>
<td>Heating status</td>
</tr>
<tr>
<td>Heating command</td>
<td>D0 (J14 pin 5)</td>
<td>Heating command</td>
</tr>
<tr>
<td>Mixer indicator</td>
<td>LED (D8)</td>
<td>Mixer working status (clockwise or counter-clockwise)</td>
</tr>
<tr>
<td>Motor CW</td>
<td>D1 (J14 pin 7)</td>
<td>Mixer motor clockwise command</td>
</tr>
<tr>
<td>Motor CCW</td>
<td>D2 (J14 pin 6)</td>
<td>Mixer motor counter-clockwise command</td>
</tr>
<tr>
<td>Motor Fast</td>
<td>D3 (J14 pin 15)</td>
<td>Mixer motor fast command</td>
</tr>
</tbody>
</table>

- Used for typical exercises performed during university to practically learn the embedded SW design flow.
ESW design

Automatic code generator

ANSI C code

Microsoft VisualC++ Express

PC Simulator

Debugging Framework

Target compiler

Target
ESW design: MBD

C / C++ code

UML 2.0

Module

Library

Object Oriented approach

Components reuse

Project as single central repository

Configurable and portable project

Settings

Target

Timing

Code gen.

Model
ESW design: MBD

• Every logical/technical unit of a system can be implemented by a MODULE

• A MODULE is very similar to a C++ class
  – It contains data, metadata, functionality and configurations for environment interaction
  – It can be **instantiated** and used in another MODULE
  – It can send/receive information to/from a child
ESW design: MBD
ESW simulation

Module View

Design model debugging
ESW deployment: model synthesis

- INIT functional items
  - execute_init()

- PERM functional items
  - execute_perm()

- User defined type <TaskName>
  - This method is a flexible instrument for distributing the calculation tasks and for prioritizing procedures

- Procedures are collectively called using a central execute function with the same processing type name
  - execute INT1 {
    proc_modA1(data_modA1);
    proc_modA2(data_modA2);
  }

executeApplication()
{
  init_HW();
  execute_INIT();
  ...
  while(true)
  {
    read_input();
    execute_PERM();
    write_output();
  }
}
ESW deployment: project structure

Application / Model
- radCASE Design

Generated Code
- Functional Code
- Structure, Data, Tables

Binaries
- Fonts, Texts
- Bitmaps, HMI

Runtime Library (mainly for HMI)
- rC-Lib

HAL Hardware/System Abstraction Layer

Low Level API
- Hardware Port IO Routines
- HMI Routines
- Data Storage Routines
- Buffered Communication Routines
- Scheduling, Timing, Interrupts

Board Support
- OS or Scheduler
- Services, Update

Target-Middleware
- HW- and Bus-assignment

Product Framework radFRAME
- PC-Software
  - with emulation of the API
  - (for single and multiple targets)

User Code.c

includes:
- process visualization
- complete functional simulation
- complete HMI simulation
- distributed communication

only for Target

only for PC-Simulation
Hardware Abstraction Layer

$Output1 = \$HIGH$

do_write(Controller_Output1, BIT_HIGH);

Embedded application

at91_set_gpio_value(do_pin1, 1);

Integration API

#define HAL_ARM_AT91_GPIO_SET(_pin_)
  CYG_MACRO_START
  HAL_ARM_AT91_GPIOX_SET(_pin_, 0, AT91_PIO);
  CYG_MACRO_END

#define HAL_ARM_AT91_GPIO_SET(_pin_)
  CYG_MACRO_START
  HAL_ARM_AT91_GPIOX_SET(_pin_, 0, AT91_PIO);
  CYG_MACRO_END

RTOS

Device drivers

Target hardware
ESW deployment: test on target

Concurrent SW & HW development
Functional verification is the process of checking that the intent of a design is preserved during its implementation.

We adopt Assertion-based Verification (ABV) for functionally verifying embedded software.

- ABV provides a unified methodology for unambiguously specifying design intents by using formal specifications.
Requirements formalization

- Properties and specifications defined in natural languages are imprecise, long and hard to understand, ...
  - What does this mean: “every request must remain asserted until a grant is received?”
  - Which signals are meant? What does “until” mean? ...

- Assertions are mathematically precise

Property Specification Language (PSL)

\[ \text{psl A1: assert never GntA and not (Req A);} \]
Effective formalization of specs requires high skills?

The interface of a parametric template

$P$ occurs at least once in between $Q$ and $R$

The PSL parametric definition

always ((Q & (!R))! ((!R) until! (P & (!R))))
ESW Requirements: Templates examples

• **responsiveness:** some events occur in response to other under certain conditions
  Ex : P causes S to happen
  PSL : always ($P \rightarrow \text{eventually!} (S))

• **absence:** an event does not occur under certain conditions
  Ex : P is continually false before R
  PSL : !$P \text{ until! } $R

• **universality:** a specific condition is true under certain conditions
  Ex : P holds continuously after Q
  PSL : always ($Q \rightarrow \text{next (always $P))}$

• **precedence:** it defines the chronological order of the events under certain conditions
  Ex : P precedes R globally
  PSL : always ($P \rightarrow ($P \text{ before } $R))

• **existence:** an event takes place at least once under certain conditions
  Ex : P holds at least once since Q up to R
  PSL : always ((Q & !$R) \rightarrow (!$R \text{ until! } $P))
Dynamic Assertion Based Verification (ABV)

- Dynamic ABV allows to verify the design and not the model
- Automatic TPGs allow to overcame a still human-based activity
- Portability of solution

Formal ABV

Simulation
- Design
- Checker
- Stimuli

Model checking
- Model
- Property
- true/false

Dynamic ABV

true/false
Checker generation for ABV

Requirements

radCASE

Assertions

Design

Test cases

Checkers

Functional Verification

SIL on PC

PIL on target

HIL on target

True / False

Validate
Test generation for ABV

• Fully automatic

• Semi-automatic (i)
  – The human tester provide an *initialization test sequence* to radCHECK to lead the design in an otherwise hard-to-reach state
  – radCHECK will exploit this suggestion and it will start to automatically generate the test sequences

• Semi-automatic (ii)
  – The human tester specify the inputs and provides some *constraints* on them
  – radCHECK will generate the test using these constraints

• Manual
  – The user provides his own test sequences stored in a file or recorded with radCASE environment
ESW Functional Verification

1) running a batch execution of the test case to identify which input sequence causes the assertion failure, determining also the exact instant in which this occurs.

2) selecting sequences to re-simulate only sequences that cause the failure.

3) inserting breakpoints inside an input sequence to reach a specific configuration from which the user can start the analysis.

4) executing a step-by-step simulation to understand which is the exact behavior that causes the assertion failure.
ESW Functional Verification on target

Solution for checking if the ESW executed on its target board behaves correctly respect to its specifications
Case study lessons

1. Creation of a new Project from the SK-A20-E Template
2. Design the application model with OO approach
3. Model the functionalities with UML state machines

Modules

**System**: the radCASE application entry point

**Program**: manage 4 phases of mixing, kneading, proving and baking.

**Thermostat**: control the Heating to obtain the Temperature set by the Program.

**Mixer**: control the mixer motor according to the mode set by the Program.

**VulcanoG20**: physical input / output model
Case study - lessons

4. Build and debug with the Simulator
5. Deploy on the Vulcano G20 HW kit
   - Build the target eCos application
   - Connect the evaluation board
   - Download and debug the application
6. Start the radCASE Visualization and test the application on Vulcano-G20
Case study - lessons

1. Functional verification of implemented application

2. Write PSL properties
   - Define temporal assertions describing the Breadmaker functionality starting from the design specification
   - The application consists of functionalities that pre- and post-conditions to be validated that has been expressed by means of PSL invariants

3. Synthesized the assertions into checkers by using the checker generator

4. Generate test case
   - Perform automatic test case generation
   - Record initialization sequences to verify specific cases

5. Perform verification session
   - Step-by-step sequence execution to debug the design
Case study - lessons

• Examples of assertions for the breadmaker

**Thermostat**

The heating coil is switched off when temperature is greater than the set point
always ((Thermostat.temperature > SETPOINT) ->
next (Thermostat.heating = OFF))

**Mixer**

The clockwise and counter-clockwise signal must not be active at the same time
never ((Mixer.cw = ON) & (Mixer.ccw = ON))

**Breadmaker**

To change the paddle mixer direction from clockwise to counterclockwise
direction a 2 seconds pause is necessary
always {Mixer.cw = ON ; Mixer.cw = OFF} |->
{ { Mixer.cw = OFF & Mixer.ccw = OFF } [*] ; Breadmaker.timer = 2;
Mixer.ccw = ON }!

Thermostat: The heating coil is switched off when temperature is greater than the set point

Mixer: The clockwise and counter-clockwise signal must not be active at the same time

Breadmaker: To change the paddle mixer direction from clockwise to counterclockwise direction a 2 seconds pause is necessary
ESW lifecycle

Requirements Analysis → High Level Design

High Level Design → Low Level Design

Low Level Design → Implementation

Implementation → Unit Testing

Unit Testing → Integration Verification

Integration Verification → System Verification

System Verification → Requirements Analysis
Concluding remarks

• Cover all phases of ESW development lifecycle
  – Starting from informal specification up to the HW deployment

• radSUITE: two off-the-shelf tools
  – Implement state of art methodologies for design and functional verification of ESW

• User friendly functionalities
  – Easy to use in university laboratory and industrial training courses
History

• Companies involved and role
  – STM Products s.r.l.
    • Development of radCASE Editor and radCHECK
  – EDALab s.r.l.
    • Development of radCHECK and its verification methodologies
  – University of Verona (ESD Group)
    • Research and teaching activities on verification methodologies
  – IMACS GmbH
    • Development of radCASE code generator, product framework (simulator) and target integration
References

Publications
Franco Fummi; Cristina Marconcini; Graziano Pravadelli, Teaching embedded software design with radSUITE, “Workshop on Embedded and Cyber-Physical Systems Education (WESE)”, October 3, 2013, Montreal, Canada


Di Guglielmo G.; Di Guglielmo L; Franco F.; Graziano P., Enabling dynamic assertion-based verification of embedded software through model-driven design, Proceedings of "ACM/IEEE Design, Automation and Test in Europe (DATE)", Dresden, Germany, 12-16 March, 2012, pp. 212-217

Application solutions
radCASE  http://www.radCASE-UX.com
radCHECK  http://www.verificationsuite.com

Contact
cristina.marconcini@stm-products.com