Verifying In-situ Embedded Software using Coverage Driven Verification

Dr David Robinson
Verilab GmbH

ClubV Europe 2005

david.robinson@verilab.com
Proof of concept project

Novel features

- verifying embedded software
- this was running on its target hardware
- the testbench and the interface to the software were on different machines
- passers by became unwitting participants in the verification!
Bluetooth based system
- Bluetooth devices automatically connected to system
- hands free connection
- data connection
- voice control

The customer was looking for new ways of verifying their embedded software
- would coverage driven verification work?

Four weeks to prove it could be done
The DUT and its Environment

CPU
- System Interface
- Bluetooth Interface
- HandsFree Interface
- SerialPort Interface
- Audio Interface
- SIM Interface
- Other Interfaces

Bluetooth

Requests
UART
Response
Events

Background
The Verification Environment
Results
The parameter field length and structure depends on the command and itself

- e.g. when the Bluetooth stack is initialised, you have to tell it how many services it will support. The size and format of the rest of the parameter is based on this field
struct packet_t like any_sequence_item {
    packet_type : packet_types_t;
    interface   : interface_type_t;
    command     : command_type_t;
    parameter_length: uint;
    keep soft packet_type == REQUEST;
};

extend I_BT_CONTROL REQUEST packet_t {
    when BT_INITIALISE packet_t{
        SecurityMode: security_mode_t;
        NoOfServices: uint;
        ServiceList: list of service_list_s;
    }
};

struct service_list_s{
    profile   : service_profile_t;
    security_level: uint;
    service_name : list of byte;
};
Verification Requirements

- Single Request Handling
  - check each request can be accepted by the DUT
  - a sequence of requests may be required to test one

- Scenario Handling
  - check that each request has the correct effect

- Error Handling
  - check that the software deals with errors correctly

- Stress Testing
  - check that it can cope with anything
Overview of Verification Environment

- **Linux**
  - SpecmanElite Testbench

- **Windows**
  - Windows Client Software

- **CPU**
  - Bluetooth

- **Stimulus**

- **Response**

- **Background**

- **The Verification Environment**

- **Results**
Windows Client Software

Requests

Windows Client Software

Response

Events

CVL

Background

The Verification Environment

Results
The eVC

Background
The Verification Environment
Results

System Agent
BTControl Agent
HandsFree Agent
SerialPort Agent

Channel Interface

From Agents
Multiplexor
To Agents
Router

Output Channel
Input Channel

Requests
Response
Events
The CVL Interface

eVC for board 0

From eVCs

Multiplexor

CVL Interface

To eVCs

Router

CVL Link (send)

CVL Link (receive)

Background

The Verification Environment

Results

Requests

Events

Response
The Verification Environment

Specman Testbench

Channel Interface

CVL Interface

CVL

Requests

Response

Events

Background

The Verification Environment

Results

Windows application

Copyright © 2005 Verilab GmbH
Using the CVL: From e to c

- **e method prototype**

  ```c
  cvl_callback CVL_msg_to_windows_client(
    msg_type : int, card_id : int,
    msg_data : list of byte ) @sys.any
  is C routine CVL_msgToWindowsClient;
  ```

- **e method call**

  ```c
  CVL_msg_to_windows_client ( 
    raw_message.get_type(),
    raw_message.get_board_number(),
    raw_message.get_data());
  ```

- **c function**

  ```c
  CVL_msgToWindowsClient ( 
    int msgType, int target,
    char* msg, int msgLength){
  }```
c function call

CVL_msgFromWindowsClient(msgType, msg, msgLength);

e method prototype

cvl method CVL_msg_from_windows_client(
    msg_type : int,
    msg_length : int,
    msg_data : list of byte) @sys.any
is C routine CVL_msgFromWindowsClient;

e method

CVL_msg_from_windows_client(
    msg_type : int,
    msg_length : int,
    msg_data : list of byte) @sys.any is{
};
The CVL was designed to allow HW/SW cosimulation

- This assumes that the c program is the master
- In this testbench it wasn't

We had to turn the c application into the master

- it sent synchronisation packets when it had nothing else to send
- this allowed the testbench advance time
Certain use-cases required manual interaction

- new versions of the testbench will allow the user to specify their intended action via the Windows application

The testbench was more interactive than we planned

- people nearby became an integral part of the verification environment
Project was a success
- Specman can be used to test in-situ embedded software

All of the standard features are available
- coverage, random generation, self-checking, etc

Stimuli and response can be routed through any interface
- RS232, USB, CANbus, etc

Five issues with the DUT found