Getting More out of your Unit Test Tool
Getting more than a Green Checkmark

Craig Pedersen
VectorCAST Product Specialist
Cary, NC
Craig.Pedersen@vector.com
Terminology & Notes

- **Unit Testing**
  - **In Theory:** Isolate smallest component and test at its interfaces
  - **In Practice:** One or more isolated modules (*.c or *.cpp files)
    > Sometimes crosses over into Integration Testing

- **Unit Testing on Host**
  - **On Development Workstation**
  - **SIL**
  - Using GNU, VisualStudio, or CPU/OS Simulator

- **Unit Testing on Target**
  - **On Physical Hardware**
  - **HIL**
  - Using Target Compiler
## Comparing Certification vs. non-Certification Testing

<table>
<thead>
<tr>
<th></th>
<th>Certified</th>
<th>Non-Certified</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>Green Checkmark</td>
<td>Find Bugs</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>Formal</td>
<td>Informal</td>
</tr>
<tr>
<td><strong>Cost Concerns</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Testing Type</strong></td>
<td>Blackbox</td>
<td>Whitebox &amp; Blackbox</td>
</tr>
<tr>
<td><strong>Tests Performed by</strong></td>
<td>Separate Test Group</td>
<td>Developers and/or Testers</td>
</tr>
</tbody>
</table>
Example Scenario: Design Requirements

Example of a simple function that determines if a company is open for business based upon a specified day & time. The design requirement is that the business is open 8am (800) to 5pm (1700) everyday of the week, but closed on Sunday.

> Function API:  `int open_for_business(int time, enum day);`

The function returns a True/False (0/1) value to indicate if it’s open. The time is in military time (e.g. 1700 is 5pm). The day of the week is an enumerator from Monday thru Sunday.

The objective is to write tests from design requirements, and see if they satisfy certification requirements.

Interesting Points:
> Code is only 3 executable lines
> Lots of interesting tests
> At least 2 bugs in the code
> This code might get thru Certification without finding all the bugs
int open_for_business(int time, t_day_of_week day)
{
    if ((time > OPEN_TIME) && (time < CLOSE_TIME) && (day != SUNDAY))
    {
        return(1);
    }
    else
    {
        return(0);
    }
}
Why use a Formal Unit Test Framework when Certification is not Required?

❖ Disadvantages
❖ Advantages
❖ Suggested Practices
❖ Real World Test Examples
Disadvantages of using a Unit Test Framework for non-Certified Testing

- Cost
- Complexity
- Time/Labor
- Hard to Measure Effectiveness
Advantages of a Unit Test Framework for non-Certified Testing

- Seamlessly Test on Host & Target
  - One copy of the Tests
- Can simulate real-world problem scenarios
  - Hard or impossible to do in System Testing
- Find Bugs Early
- Regression Testing
- Provides Synergy between Development & Test Teams
Suggested Practices

- Develop on Host, Deploy on Target
- Unit Test Framework & Test Environments Created by Development Team
- Developers create Whitebox Tests based on System Knowledge
- Test Team creates Blackbox Tests based on Design Requirements
- Test Team formalizes the process
  - Make both Whitebox & Blackbox Tests regressionable
  - Captures Metrics
  - Continuous Integration
Existing Practices: Testing in Silos

Mostly On-Target

Whitebox Testing
- By Developers
- Using Debugger
- Other Ad-hoc Methods

Blackbox Testing
- By Test Team
- From Requirements
- Regressionable
- Formal Methods
Suggested Practices: Using a Common Frameworks

- **Unit Test Framework**
  - Develop On Host
  - Test Team
    - Blackbox Tests
  - Developer
    - Whitebox Tests
  - Deploy On Target
Contrasting Whitebox vs. Blackbox Testing Concerns

- **API:**
  - Add_Name(Name);
  - Max Name length is 10 characters

- **Test Example:**
  - RetCode = Add_Name(“nametoolong”);

- **Whitebox Test Concern**
  - Buffer Overrun

- **Blackbox Test Concern**
  - Appropriate value for Return Code
Whitebox Testing Concerns

- CPU Stack Usage
- Buffer Overrun/Underrun
- System Call Monitor
- Redefinition of Hardware Constructs on Host
- Interrupt Timing & Interplay
- Loop Counting
- Function Call Trace
- Event Sequencing
- Hardware Anomalies
malloc_front_end()
  ▶ Call Real malloc()
  ▶ Record caller
  ▶ Record Address & Length
  ▶ Add Marker at end of Buffer

free_front_end()
  ▶ Record caller
  ▶ Record Address
  ▶ Validate Marker
  ▶ Call Real free()

Post Mortem (on Host)
  ▶ Run Report from recorded data
Redefinition of Hardware Constructs

Code example: Memory Mapped I/O

- #include “device_driver_registers.h”
- Status = *DEVICE_DRIVER_ADDRESS;
- If (Status & ErrorBit) ...

Test Harness Redefinition:

- #include “device_driver_registers.h”
- #undef DEVICE_DRIVER_ADDRESS
- Unsigned char DeviceDriverBuffer[100];
- #define DEVICE_DRIVER_ADDRESS (& DeviceDriverBuffer)
Simulating Random Interrupts

```c
#include "stdio.h"
#include "string.h"

extern char hardware_input_bfr[100];

int process_packet()
{
    char input_string[100];

    strcpy(input_string, hardware_input_bfr);

    printf("\nInput Packet=%s", input_string);

    return(0);
}
```
Interrupt Insertion

extern void usb_isr(void);
#include "c:\vcast\2019sp2\mingw\include\stdlib.lib"
#include "c:\vcast\2019sp2\mingw\include\string.h"
extern char hardware_input_bfr[100];
int process_packet()
{
    char input_string[100];
    process_packet
    strcpy(input_string, hardware_input_bfr);
    printf("\nInput Packet=%s", input_string);
    return(0);
}

int process_packet()
{
    printf("\nInput Packet=%s", input_string);
    return(0);
    usb_isr();
Summary

- Develop on Host, Deploy on Target
- Unit Test Framework & Test Environments