

Embedded System Design Techniques™

Debugging Real-time Embedded Software – Hands-on

Session 2: Foundational Debug Techniques

July 12th, 2016
Jacob Beningo, CSDP

Course Overview

- Introduction to Debugging Real-time Embedded Systems
- **Foundational Debugging Techniques**
- Debugging the ARM Cortex-M Microcontroller
- Utilizing Systems Viewers and Trace tools to Debug Firmware
- Tips and Tricks for Debugging Embedded Systems

Session Overview

- Bug Classification
- Methods for Tracking Bugs
- Effective use of break-points
- Printf
- assert



Bug Classification

Bohrbugs



Heisenbugs



Schroedinbugs



Mandelbugs



Bug Severity

Severity	Severity Level	Severity Description
1	Critical	The module/product crashes or the bug causes non-recoverable conditions. System crashes, GP Faults, or database or file corruption, or potential data loss, program hangs requiring reboot are all examples of this type.
2	High	Major system component unusable due to failure or incorrect functionality. These bugs cause serious problems such as a lack of functionality, or insufficient or unclear error messages that can have a major impact to the user, prevents other areas of the app from being tested, etc. Severity 2 bugs can have a work around, but the work around is inconvenient or difficult.
3	Medium	Incorrect functionality of component or process. There is a simple work around for the bug if it is Severity 3.
4	Minor	Documentation errors or signed off severity 3 bugs

Bug Fix Priority

Priority ID	Priority Level	Priority Description
5	Must Fix	This bug must be fixed immediately; the product cannot ship with this bug.
4	Should Fix	These are important problems that should be fixed as soon as possible. It would be an embarrassment to the company if this bug shipped.
3	Fix When Have Time	The problem should be fixed within the time available. If the bug does not delay shipping date, then fix it.
2	Low Priority	It is not important(at this time) that these bugs be addressed. Fix these bugs after al other bugs have been fixed.
1	Trivial	Enhancements that are good to have features but are currently out of scope.

Tracking Bugs

Best Solution to Track Bugs

- 1) Don't track them (Fix them on the spot)
- 2) Use Bug tracking software

What to look for in a tracking system:

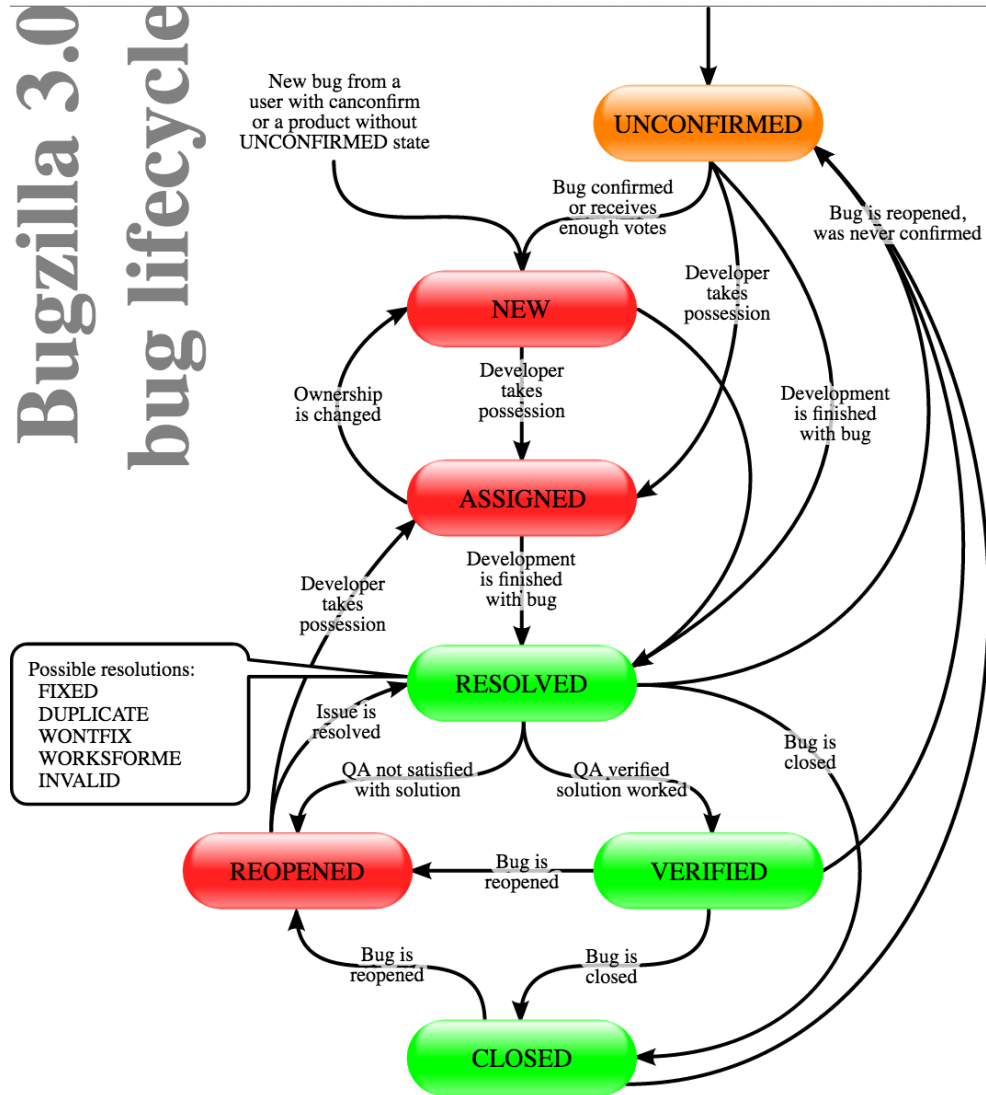
- Bug Categorization
- Bug Prioritization
- Bug Assignment and Status
- Reporting
- Project tracking integration
- Strong eco-system

Examples:

- Bugzilla
- Mantis
- etc

Bug Life Cycle

Bugzilla 3.0
bug lifecycle



[CC BY-SA 3.0](#)

Derivative work:
[CrazyTerabyte](#) (talk)
[Bugzilla Lifecycle color-aqua.png](#):
[User:Nyco](#)

Breakpoint Debugging

Breakpoints – define instruction addresses so that when the processor gets to that instruction, it stops there (enters halt or sometimes referred to as debug state)

- Hardware breakpoints
- Software breakpoints

Watchpoints – allows you to define data addresses so that when the processor accesses this address location, it triggers a debug event that can be used to halt the processor

Breakpoint Debugging

Two different types of breakpoints that can be used

- BKPT is a software breakpoint instruction that can be added to software.
 - BKPT must be used carefully if the debug monitor is used. It cannot be used in tasks and exception handlers with lower priorities. That means no NMI or Hard Fault handler breakpoints!
 - If BKPT is encountered when halt and debug monitor modes are disabled, the processor will raise a hard fault exception.
- Breakpoint using the address comparators in the Flash Patch and Breakpoint Unit (FPB). There are only 8 comparators, 6 of which can be used for address breakpoints.

Debugging w/ printf

printf provides a method for formatting and outputting serial data.

```
#include <stdio.h>
```

```
printf("Hello World!");
```

```
printf("The value of x is %d", x);
```

```
printf("The value of x is %d, y is %d", x, y);
```

```
printf("The value of pi is %3.2f", 3.14);
```

Debugging w/ printf

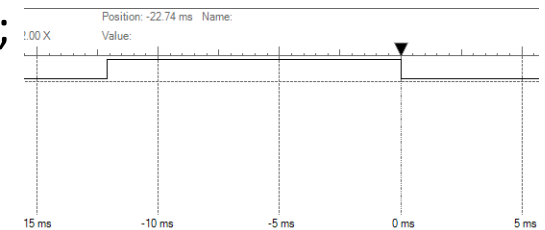
- Have to pull in standard library
- ROM and RAM usage will increase
- Simple printf statements can affect the real-time performance of the system
- Limited number of printf's that can be added to a system
- Requires changing code to get useful data out during a debug session

printf Setup

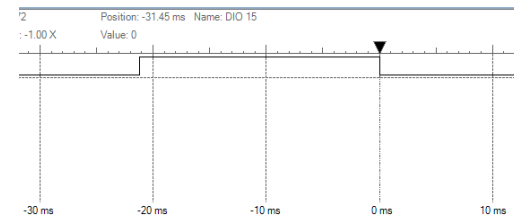
Setup options

- Use system services (libraries)
 - Use UART
 - Driver
 - Application code
 - Use ITM
- Transmission Type
 - Blocking
 - Non-blocking

- `printf("Hello World!");`
 - 12.5ms



- `printf("The system state is %d", state);`
 - 21ms



Assertions

An assertion is a Boolean expression at a specific point in a program that will be true unless there is a bug in the program.

Pros of ASSERT()

- improve testing
- bugs are easier to detect
 - execution stops at them
- can serve as executable comments
- improve code quality
- can be turned on and off

Cons of ASSERT()

- slow down code execution
- commonly misunderstood
- used improperly for error handling
- use string that require RAM/ROM
- require printf and a terminal
- can be turned on and off

What is the difference between a bug and an error condition?

Assertions

- If the expression is true, execution continues normally
- If the expression is false, whatever happens is "undefined"

Proper Use:

```
void CalculateDistance(uint8_t Velocity)
{
    ASSERT(Velocity < 150);
}
```

Improper Use:

```
int result = Open("MyFile.txt", 'r');
ASSERT(result != NULL);
```

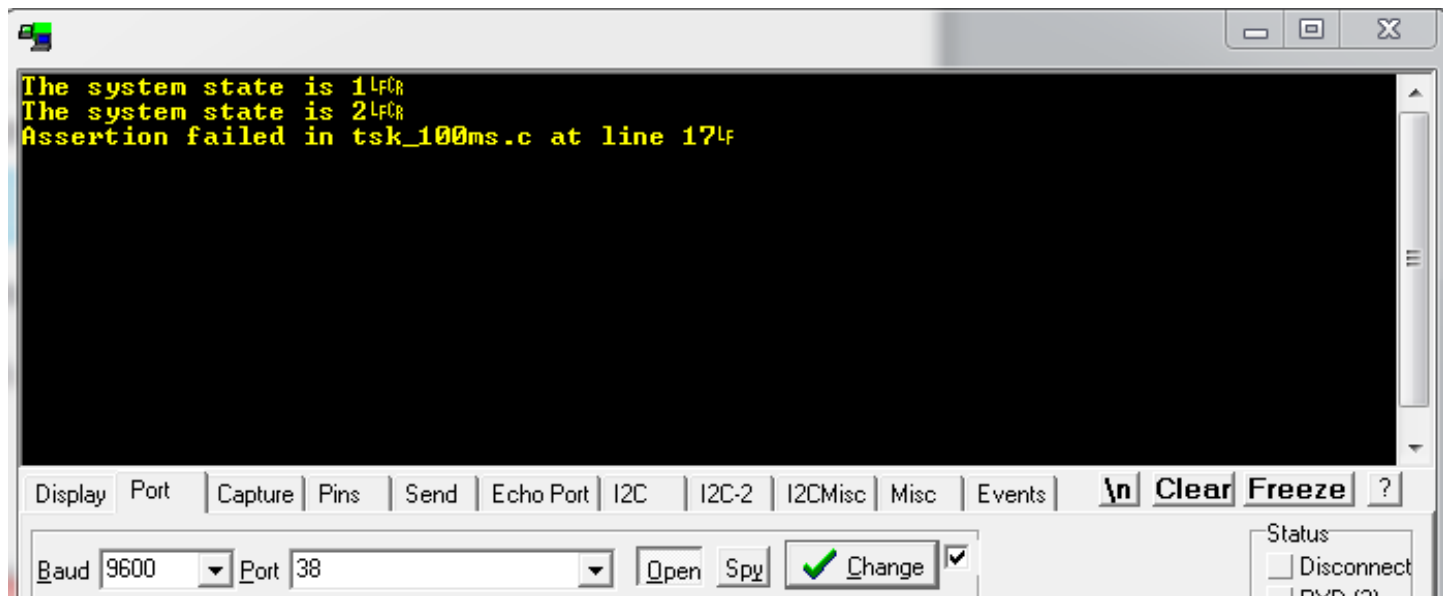
Mistakes:

```
ASSERT(result = 14);
ASSERT(VelocitySet(50) < VELOCITY_MAX);
```

Assertions Setup

Steps to Setup assert

1. Setup serial interface
2. Setup printf
3. Include assert.h
4. Enable assertions
5. Sprinkle assertions through-out code base



The screenshot shows a serial terminal window with a black background and yellow text. The text displays the following output:

```
The system state is 1\r\nThe system state is 2\r\nAssertion failed in tsk_100ms.c at line 17\r\n
```

Below the terminal window is a control panel with various tabs: Display, Port, Capture, Pins, Send, Echo Port, I2C, I2C-2, I2CMisc, Misc, Events, \n, Clear, Freeze, and ?. The Port tab is selected. The Baud rate is set to 9600 and the Port is set to 38. There are buttons for Open, Spy, Change (with a green checkmark), and a checkbox for Status. The Status section includes a checkbox for Disconnect and a checkbox for BVD (2).

Additional Resources

- Download Course Material for
 - Updated C Doxygen Templates (Sept 2015)
 - Example source code
 - Templates
 - YouTube Videos
- Microcontroller API Standard
- EDN Embedded Basics Articles
- Embedded Bytes Newsletter
 - <http://bit.ly/1BAHYXm>



From www.beningo.com under






- Blog > Debugging Realtime Embedded Systems

The Lecturer – Jacob Beningo



Jacob Beningo
Principal Consultant

Social Media / Contact

-  : jacob@beningo.com
-  : 248-719-6850
-  : Jacob_Beningo
-  : Beningo Engineering
-  : JacobBeningo

EDN : Embedded Basics

CONSULTING

- Secure Bootloaders
- Code Reviews
- Architecture Design
- Real-time Software
- Expert Firmware Analysis

EMBEDDED TRAINING



www.beningo.com