JUMP STARTING SOFTWARE DEVELOPMENT TO MINIMIZE DEFECTS (BUGS)

Jacob Beningo | President
THE LECTURER
Jacob Beningo Principal Consultant

Social Media / Contact

✉️ jacob@beningo.com
📞 810-844-1522
🐦 Jacob_Beningo
⭐️ Beningo Engineering
🔗 JacobBeningo
💡 Embedded Basics

Consulting
• Secure Bootloaders
• Code Reviews
• Architecture Design
• Real-time Software
• Expert Firmware Analysis
• Microcontroller Systems

Embedded Training
• RTOS Workshop
• Bootloader Design
• Debugging Techniques
• Security Fundamentals
• Micro Python

Newsletters
Embedded Bytes

http://bit.ly/1BAHYXm

www.beningo.com
SESSION OVERVIEW

TOPICS

1. Bugs, errors and defects
2. Debugging techniques
3. Defect management
4. The 9 jump start phases
5. Best practices
6. Time for action
7. Going Further

OBJECTIVE

Explore minimizing software defects by properly starting a project.
DEMONSTRATION MATERIALS

STM32L475 Discovery Kit

Keil ULINKplus

Percepio Tracealyzer
INTRODUCTION

How much time do you spend debugging?

40%

Developers on average spend 40% of the development cycle debugging!

On a year long project, this is anywhere from 2.5 – 4.8 months!
INTRODUCTION

No other industry on the planet accepts error rates this high!
WHAT DEBUGGING TECHNIQUES DO YOU USE?

Evaluate Yourself – Scale 1 – 10

• 10 – No Improvement Necessary

• 0 – I don’t even use this technique

0 – 40 Stumbling in the dark ages

40 – 60 Crawling out of the abyss

60 – 80 Fast, Efficient Bug Squasher
Errors are mistakes made by the programmer in implementing the software design.

Defects are mistakes that result from unanticipated interactions or behaviors that occur when implementing the software design.

The developer is responsible for preventing errors and defects not just removing them!
PHASE 1: PROJECT SETUP

1. Setup revision control system
2. Create an empty project
3. Create the project directory structure
4. Set the white tab spacing
5. Identify the coding standards that will be used to develop the software
PHASE 2: DOCUMENTATION FACILITY SETUP

1. Add Doxygen code templates
2. Configure Doxygen wizard
3. Import skeleton HAL’s and API’s
4. Create a version log
5. Create hardware configuration modules

```c
/*
 * Function : Uart0_ISR( )
 */

/*
 * \section Description Description:
 * This function is the Uart0 Channel interrupt service routine.
 * \param None.
 * \return None.
 * \section Example Example:
 * \endcode
 */

/*
 * \endcode
 */

/*
 * \see Uart_Init
 */

- HISTORY OF CHANGES -

<table>
<thead>
<tr>
<th>Date</th>
<th>Software Version</th>
<th>Initials</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04/02/14</td>
<td>0.1.0</td>
<td>JWB</td>
<td>Function Created.</td>
</tr>
</tbody>
</table>
PHASE 3: CODE ANALYSIS

1. Setup static code analysis tool
2. Setup software metrics analyzers
3. Setup dynamic code analysis tools
PHASE 4: SCHEDULER SETUP
### PHASE 5: SETUP RTOS AWARE DEBUGGING

<table>
<thead>
<tr>
<th>Profile</th>
<th>Thread</th>
<th>Stack</th>
<th>MessageQueue</th>
<th>CountingSemaphore</th>
<th>Mutex</th>
<th>EventFlag</th>
<th>MemoryBlockPool</th>
<th>MemoryBytePool</th>
<th>Timer</th>
<th>System</th>
<th>ReadyQueue(No.:Priority)</th>
<th>Status</th>
<th>SuspendedFactor(ControlBlock*)</th>
<th>OwnedTX_MUTEX*(top)</th>
<th>Priority</th>
<th>RunCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
<td>Name</td>
<td>Entry</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Blinky Thread</td>
<td>blinky_thread_func</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SLEEP</td>
<td></td>
<td></td>
<td>1</td>
<td>79</td>
</tr>
<tr>
<td>2</td>
<td>IO_Thread</td>
<td>IO_Thread_func</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>COMPLETED</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not created</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not created</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not created</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not created</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not created</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not created</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not created</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not created</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not created</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Profile</th>
<th>Thread</th>
<th>Stack</th>
<th>MessageQueue</th>
<th>CountingSemaphore</th>
<th>Mutex</th>
<th>EventFlag</th>
<th>MemoryBlockPool</th>
<th>MemoryBytePool</th>
<th>Timer</th>
<th>System</th>
<th>Status</th>
<th>Priority</th>
<th>MaxStackUsage(bytes)</th>
<th>StackSize(bytes)</th>
<th>StackStart</th>
<th>StackEnd</th>
<th>StackPointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
<td>Name</td>
<td>Entry</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Blinky Thread</td>
<td>blinky_thread_func</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1024</td>
<td>1ffce2ba8</td>
<td>1ffce2c3f</td>
<td>1ffce2ba8</td>
</tr>
<tr>
<td>2</td>
<td>IO_Thread</td>
<td>IO_Thread_func</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1024</td>
<td>1ffce27e0</td>
<td>1ffce283f</td>
<td>1ffce27e0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1024</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PHASE 6: SETUP DEBUG MESSAGES & TRACE

A trace channel

- Serial
- TCP/IP
- RTT

Trace tool

- SystemView (SEGGER)
- Tracealyzer (Percepio)
PHASE 6: SETUP DEBUG MESSAGES & TRACE

- printf
- UART
- ITM
- Configure assert
- Real-time data graphing
- Watch points on critical variables
PHASE 7: RECORD A BASELINE
DEMONSTRATION
PHASE 8: SOFTWARE IMPLEMENTATION

```c
void IO_Thread_entry(void)
{
    /* Define the units to be used with the threadx sleep function */
    const uint32_t threadx_tick_rate_Hz = 100;

    /* Set the blink frequency (must be <= threadx_tick_rate_Hz */
    const uint32_t freq_in_hz = 2;

    /* Calculate the delay in terms of the threadx tick rate */
    const uint32_t delay = threadx_tick_rate_Hz / freq_in_hz;

    ioport_level_t level = IOPORT_LEVEL_HIGH;
    g_ioport.p_api->pinWrite(IOPORT_PORT_01_PIN_04, IOPORT_LEVEL_HIGH);

    while (1)
    {
        if (level == IOPORT_LEVEL_HIGH)
        {
            g_ioport.p_api->pinWrite(IOPORT_PORT_01_PIN_04, IOPORT_LEVEL_HIGH);
            level = IOPORT_LEVEL_LOW;
        }
        else
        {
            g_ioport.p_api->pinWrite(IOPORT_PORT_01_PIN_04, IOPORT_LEVEL_LOW);
            level = IOPORT_LEVEL_HIGH;
        }
        tx_thread_sleep (delay);
    }
}
```
PHASE 9: VERIFICATION

Instruction Tracing

Function Analysis

Code Profiling

Instruction Tracing
EXPECTED RESULTS

The end of bugs!

Dramatically decreased time spent managing software errors and defects
  • Can we decrease from 40% to 20%?
  • This could be 2 months less time per year troubleshooting!

Decreased developer stress

Working fewer hours to meet deadlines

More time to focus on product features

Improved robustness and reliability

Meeting project deadlines

Detailed understanding on how the system is behaving and performing
JUMP START BEST PRACTICES

1. Don’t wait until there is a problem to setup defect management tools
2. Start your project with all the tools in place to
   • Detect a defect immediately
   • Visualize and understand the software as it develops
3. Use the ITM for printf messages
4. Use trace tools to not just debug but baseline your software
5. Feedback timing measurements into model assumptions such as RMA
6. Don’t think of bugs as something that is happening to you, they are CAUSED by you!
7. Take the time up front to design and instrument the software
8. Be disciplined in using the tools and not skipping any steps!
9. Don’t let today’s management fires and priorities distract you and make you lose your discipline
WHAT ARE YOU GOING TO DO?

**Time Frame**

- Next 30 days?
- Next 90 days?
- Next 365 days?

**Actions**

- Dedicate 2 lunches per weeks to growing skills
- Read a book on debugging
- Examine and experiment with new techniques
- Take an online course
GOING FURTHER

Resources from beningo.com

- C Doxygen templates
- Jacob’s Blog
- Embedded Bytes Newsletter
- RTOS Best Practices Guide
- White Papers
- Jump start project checklist
- Books
- Other resources for developers and managers
UPCOMING EVENTS

• Getting Started with Amazon FreeRTOS
• Amazon FreeRTOS Behind the Scenes

$100 off VIP, Gold, and Main Conference Passes or a Free Expo Hall Pass

Code: BENINGO100
THE NEXT WEBINAR WILL BEGIN SHORTLY

Debugging and profiling your STM32 device using Atollic TrueSTUDIO for STM32

As any builder, handyman, or software developer knows, the right tools make all the difference in meeting deadlines, working efficiently and delivering a quality product. In embedded development the quality of your tools often determines the length and difficulty of the project schedule, particularly when it comes to debugging, test, and software optimization. In this webinar we will show a number of tools in Atollic TrueSTUDIO for STM32 that can be used to debug and profile your code.

Key Take-A-Ways:

- Build analysis, visualize how code and data is placed in the build output memory layout
- Stack analysis, visualize the stack usage of your functions and entire application
- Fault analysis, a simple way of examining CPU fault conditions
- Printf() re-direction to ITM ports, software tracing with minimal overhead
- Statistical profiling, understand which portions of code consume most CPU time
- Exception/Interrupt tracing, examine exception and interrupt behavior
- Data tracing, non-intrusive monitoring of memory accesses

Register at: https://bit.ly/2k9YSun
THANK YOU!

beningo.com