BENINGO EMBEDDED GROUP

Simplifying Concepts, Delivering Success℠

JUMP STARTING SOFTWARE DEVELOPMENT TO MINIMIZE DEFECTS (BUGS)

Jacob Beningo | President

BENINGO EMBEDDED GROUP

arm

Embedded RELATED.com

percepio SENSING SOFTWARE

Serious™

life.augmented

BENINGO EMBEDDED GROUP

THE LECTURER

Jacob Beningo Principal Consultant

www.beningo.com

Social Media / Contact

jacob@beningo.com

810-844-1522

Jacob_Beningo

Beningo Engineering

JacobBeningo

EDN Embedded Basics

ARM Connected Community

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

BENINGO EMBEDDED GROUP

SESSION OVERVIEW

TOPICS

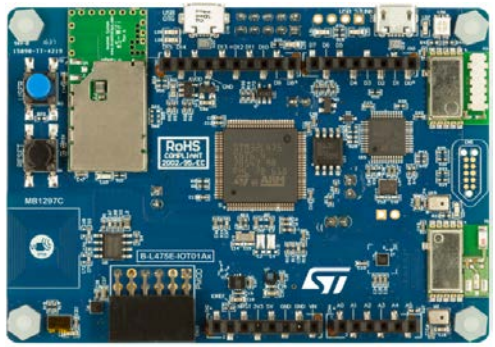
1. Bugs, errors and defects
2. Debugging techniques
3. Defect management
4. The 9 jump start phases
5. Best practices
6. Time for action
7. Going Further

OBJECTIVE

Explore minimizing software defects by properly starting a project.

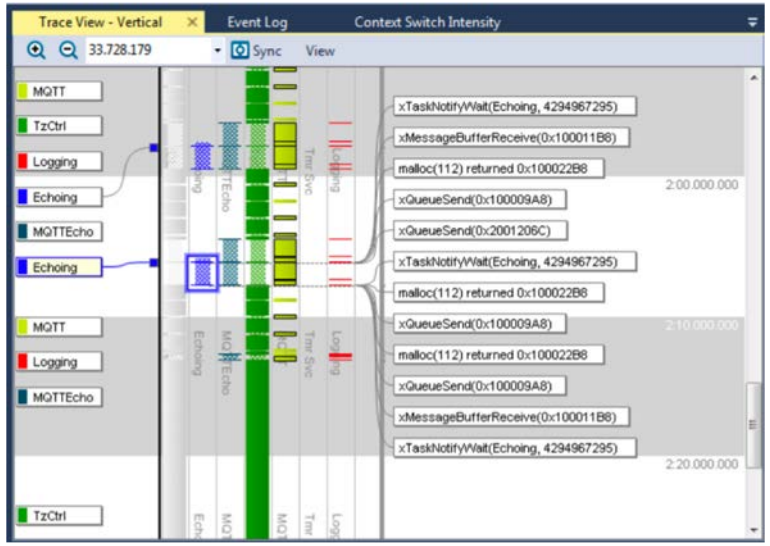
DEMONSTRATION MATERIALS

STM32L475 Discovery Kit



Keil ULINKplus

Percepio Tracealyzer



BENINGO EMBEDDED GROUP

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!

BENINGO EMBEDDED GROUP

INTRODUCTION

No other industry on the planet accepts error rates this high!

BENINGO EMBEDDED GROUP

WHAT DEBUGGING TECHNIQUES DO YOU USE?

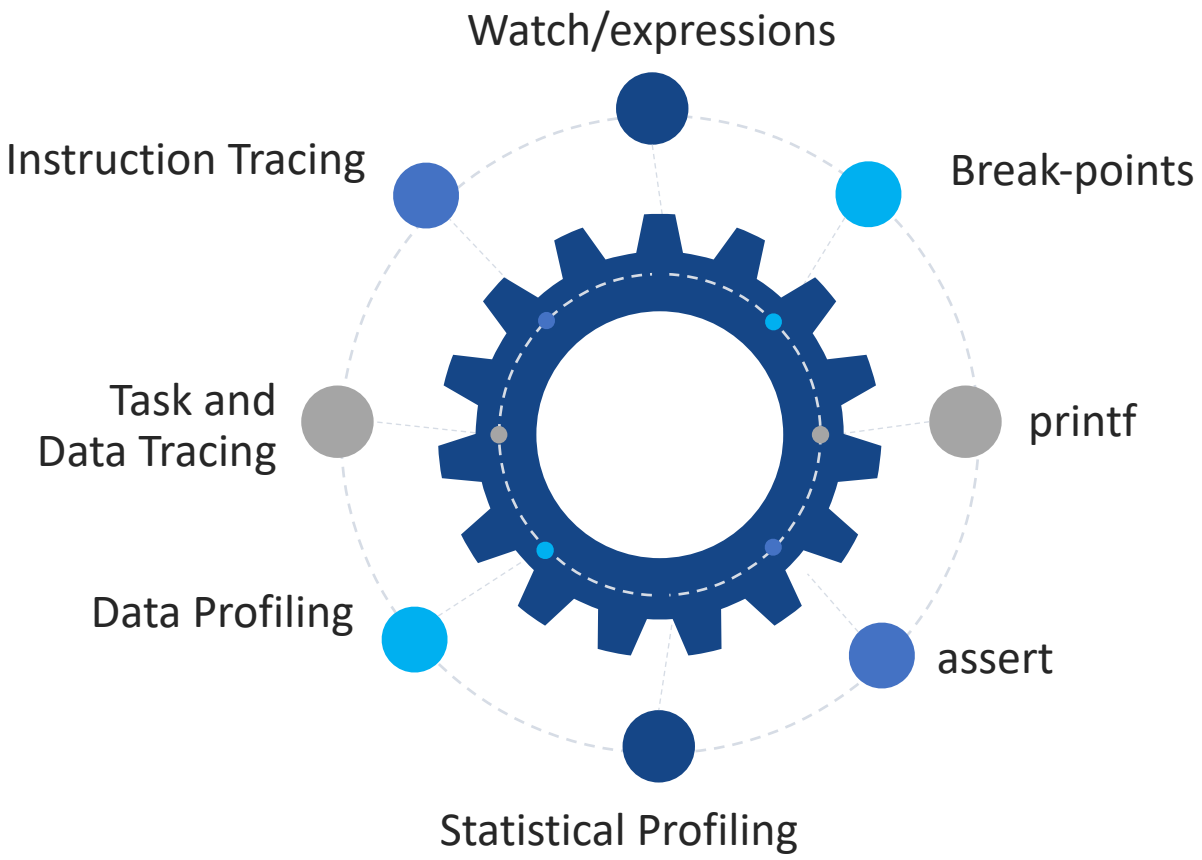
Evaluate Yourself – Scale 1 – 10

- 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



DEFECT MANAGEMENT

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.

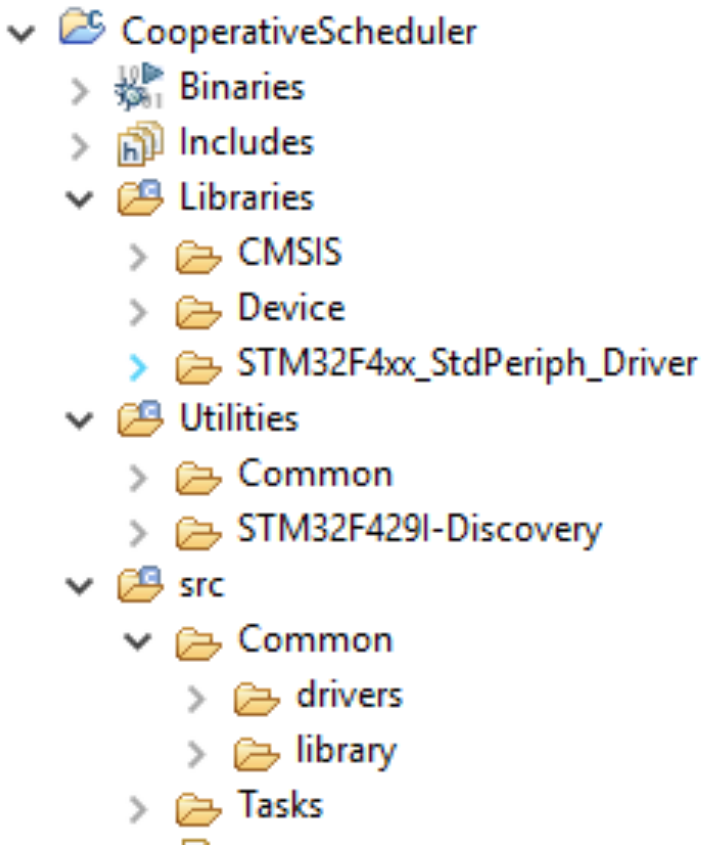
The **developer** is responsible for **preventing** errors and defects not just removing them!

Bugs don't exist in the code!

Defects and Errors Exists

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



BENINGO EMBEDDED GROUP

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

```
/*****************************************************************************
* Function : Uart0_ISR( )
*//**
* \section Description Description:
*
*  This function is the Uart0 Channel interrupt service routine.
*
* \param     None.
*
* \return     None.
*
* \section Example Example:
* \code
*
* \endcode
*
* \see Uart_Init
*
*  ----------------------
*  - HISTORY OF CHANGES -
*  ----------------------
*    Date    Software Version    Initials   Description
*  04/02/14       0.1.0            JWB      Function Created.
*
*****************************************************************************/
```

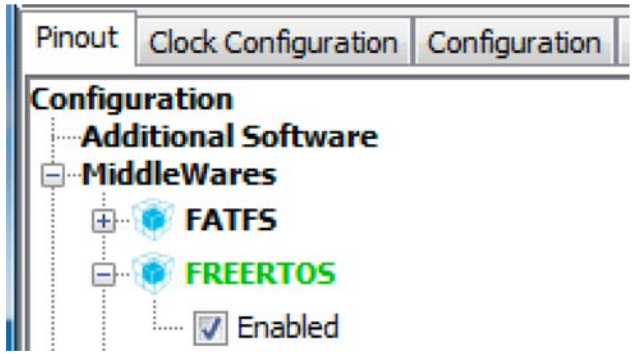
BENINGO EMBEDDED GROUP

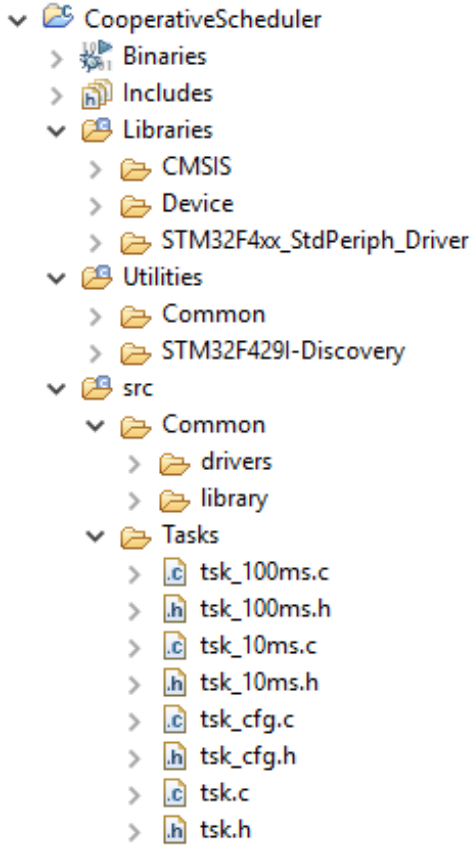
PHASE 3: CODE ANALYSIS

1. Setup static code analysis tool
2. Setup software metrics analyzers
3. Setup dynamic code analysis tools

```
                    ~~ File Functional Summary ~~

File Function Count....:          5
Total Function LOC.....:         35  Total Function Pts LOC :        0.5
Total Function eLOC....:         17  Total Function Pts eLOC:        0.3
Total Function lLOC....:         14  Total Function Pts lLOC:        0.2
Total Function Params .:          7  Total Function Return .:         5
Total Cyclo Complexity :          8  Total Function Complex.:        20
       ------     -----     -----     ------     ------     -----
Max Function LOC ......:         21  Average Function LOC ..:       7.00
Max Function eLOC .....:         11  Average Function eLOC .:       3.40
Max Function lLOC .....:          8  Average Function lLOC .:       2.80
       ------     -----     -----     ------     ------     -----
Max Function Parameters:          2  Avg Function Parameters:       1.40
Max Function Returns ..:          1  Avg Function Returns ..:       1.00
Max Interface Complex. :          3  Avg Interface Complex. :       2.40
Max Cyclomatic Complex.:          4  Avg Cyclomatic Complex.:       1.60
Max Total Complexity ..:          6  Avg Total Complexity ..:       4.00

End of File: C:\SPO2 Module\Common\drivers\src\pwm.c
```

PHASE 4: SCHEDULER SETUP





PHASE 5: SETUP RTOS AWARE DEBUGGING

Profile | Thread | Stack | MessageQueue | CountingSemaphore | Mutex | EventFlag | MemoryBlockPool | MemoryBytePool | Timer | System | ReadyQueue(No.=Priority)

No.	Name	Entry	Status	SuspendedFactor(ControlBlock*)	OwnedTX_MUTEX*(top)	Priority	RunCount
1	Blinky T...	blinky_thread_func	SLEEP			1	79
2	IO_Thread	IO_Thread_func	COMPLETED			1	2
3			Not created				
4			Not created				
5			Not created				
6			Not created				
7			Not created				
8			Not created				
9			Not created				
10			Not created				
11			Not created				

Profile | Thread | Stack | MessageQueue | CountingSemaphore | Mutex | EventFlag | MemoryBlockPool | MemoryBytePool | Timer | Syste

No.	Name	Entry	StackPointer	StackStart	StackEnd	StackSize(bytes)	MaxStackUsage(bytes)
1	Blinky Thread	blinky_thread_func	1ffe2ba8	1ffe2840	1ffe2c3f	1024	152
2	IO_Thread	IO_Thread_func	1ffe27e0	1ffe2440	1ffe283f	1024	128
3							

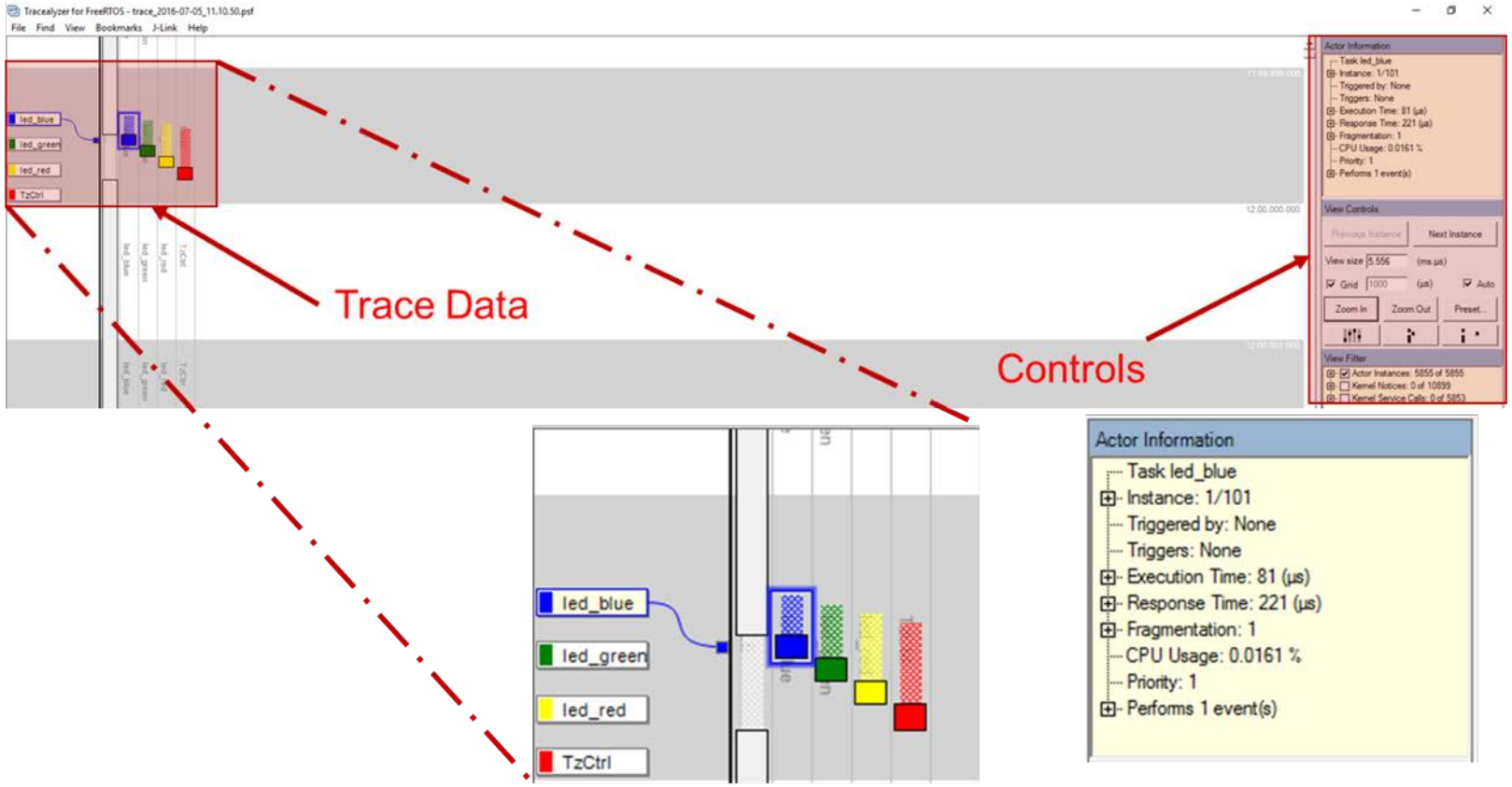
PHASE 6: SETUP DEBUG MESSAGES & TRACE

A trace channel

- Serial
- TCP/IP
- RTT

Trace tool

- SystemView (SEGGER)
- Tracealyzer (Percepio)

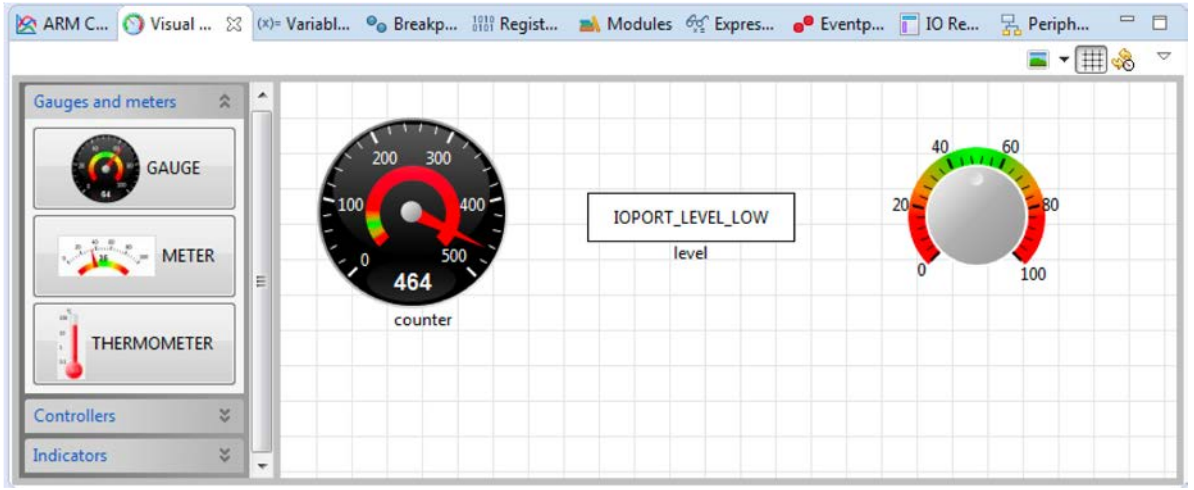


PHASE 6: SETUP DEBUG MESSAGES & TRACE

- printf
- UART
- ITM
- Configure assert
- Real-time data graphing
- Watch points on critical variables

```
int _write(int32_t file, uint8_t *ptr, int32_t len)
{
  /* Implement your write code here, this is used by puts and printf for example */
  int i=0;
  for(i=0 ; i<len ; i++)
  {
    ITM_SendChar((*ptr++));
  }

  return len;
}
```

CMSIS ITM call



PHASE 7: RECORD A BASELINE

Contexts

Name	Type	Stack Information	Run Count	Frequency	Last Run Time	Min Run Time
SysTick	#15		15152	1000 Hz	0.0058 ms	0.0056 ms (#26123)
Scheduler			681	45 Hz	0.0062 ms	0.0056 ms (#9350)
LP Task	@50	512 @ 0x20000310	75	5 Hz	0.0151 ms	0.0141 ms (#11584)
Idle			15152	1000 Hz	0.9941 ms	0.9031 ms (#2195)
HP Task	@100	512 @ 0x20000110	303	20 Hz	0.0148 ms	0.0139 ms (#962)

Max Run Time	Total Run Time	Run Time/s		Min Run Time/s		Max Run Time/s	
0.0457 ms (#14909)	89.4513 ms	5.8930 ms	0.59%	4.5554 ms	0.46%	5.9723 ms	0.60%
0.0324 ms (#959)	13.3393 ms	0.8837 ms	0.09%	0.6754 ms	0.07%	0.8852 ms	0.09%
0.0152 ms (#9349)	1.1158 ms	0.0747 ms	0.01%	0.0454 ms	0.00%	0.0757 ms	0.01%
0.9943 ms (#26124)	15042.564 ms	992.852 ms	99.29%	767.4399 ms	76.74%	992.8548 ms	99.29%
0.0148 ms (#178)	4.4671 ms	0.2964 ms	0.03%	0.2214 ms	0.02%	0.2964 ms	0.03%

SWV Trace Log | SWV Console | SWV Statistical Profiling

Function	% in use	Samples	Start address	Size
prvIdleTask()	41.87%	123322	0x5d6d	0x24
FRTOS1_vApplicationIdle...	28.99%	85399	0x17b1	0xc
prvCheckTasksWaitingTe...	28.99%	85388	0x5ea9	0xc
xTaskIncrementTick()	0.05%	149	0x57a9	0x184
vPortTickHandler()	0.02%	49	0x769d	0x24
xTaskResumeAll()	0.01%	43	0x55c5	0x110
vTaskSwitchContext()	0.01%	35	0x592d	0xcc

Overflow packets: 0 PC Samples: 294553

DEMONSTRATION

PHASE 8: SOFTWARE IMPLEMENTATION

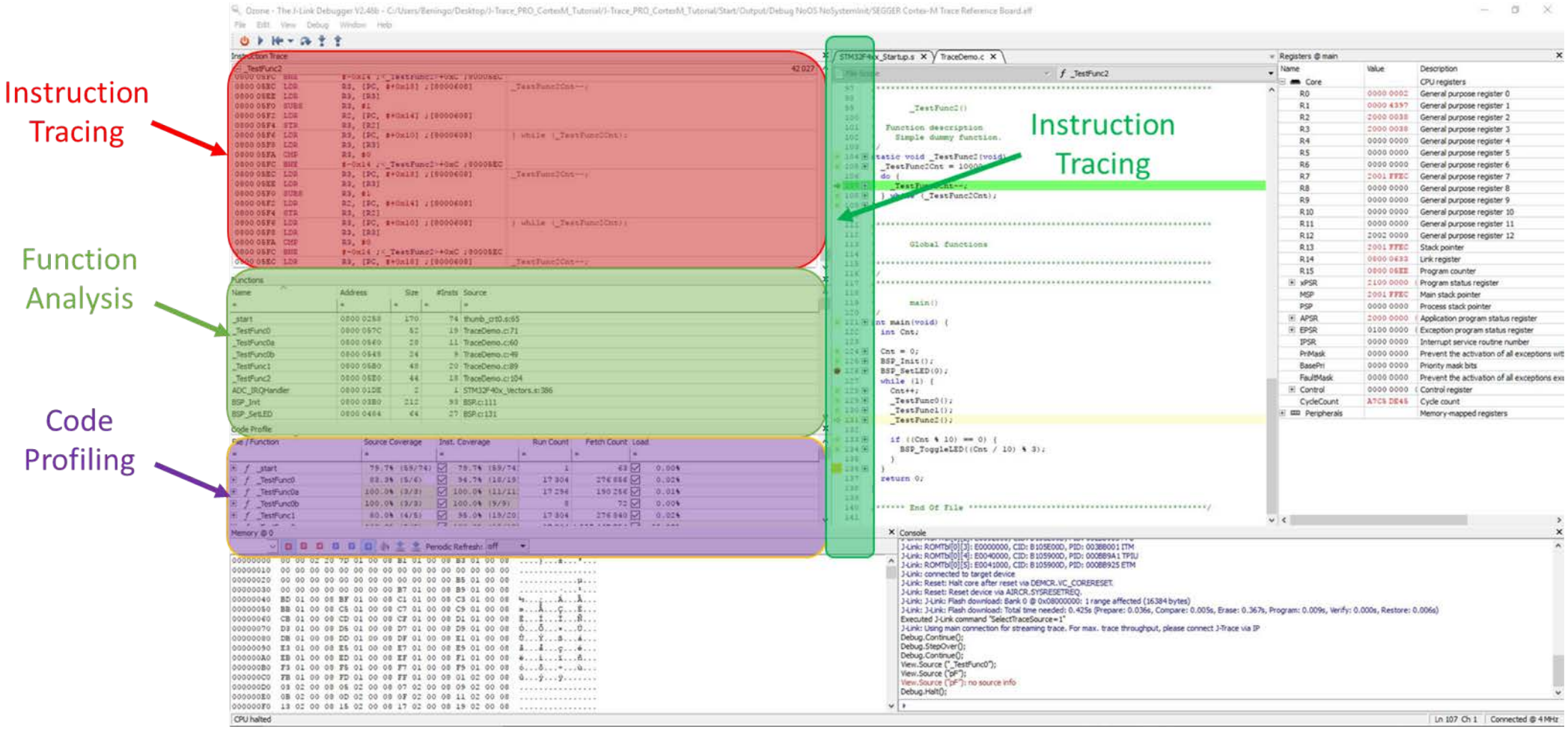
constants

definitions

Thread code

```
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



BENINGO EMBEDDED GROUP

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 todays 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

sensors expo & conference
JUNE 26-28 2018 San Jose

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

Code: BENINGO100

BENINGO EMBEDDED GROUP

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

Simplifying Concepts, Delivering Success℠