

Runtime analysis tool C-RUN

Ensuring code quality through runtime analysis

With runtime analysis, you can find actual errors at runtime. Our add-on tool C-RUN is easy to use and has a wide range of features based on customer requests. C-RUN performs runtime analysis by checking application execution directly within the development environment. It checks for arithmetic issues, bounds issues and heap integrity and will tell you what went wrong and where.

Key features

- Analysis of C and C++ code
- Intuitive and easy-to-use settings
- Unique optimizations of test instrumentation minimizes code size overhead
- Comprehensive and detailed runtime error information
- Call stack information provided for each error found
- Code correlation and graphical feedback in editor
- Flexible error filter management
- Bounds checking to ensure accesses to arrays and other objects are within boundaries
- Buffer overflow detection
- Detection of value changes when casting between types
- Checks for overflow and wraparound in computations
- Discovery of bit losses in shift operations
- Heap and memory leaks checking
- Available as an add-on product for:
 - IAR Embedded Workbench for Arm, version 7.20 and forward*
 - IAR Embedded Workbench for RX, version 3.10 and forward*

Arithmetic issues, Bounds issues and Heap checking

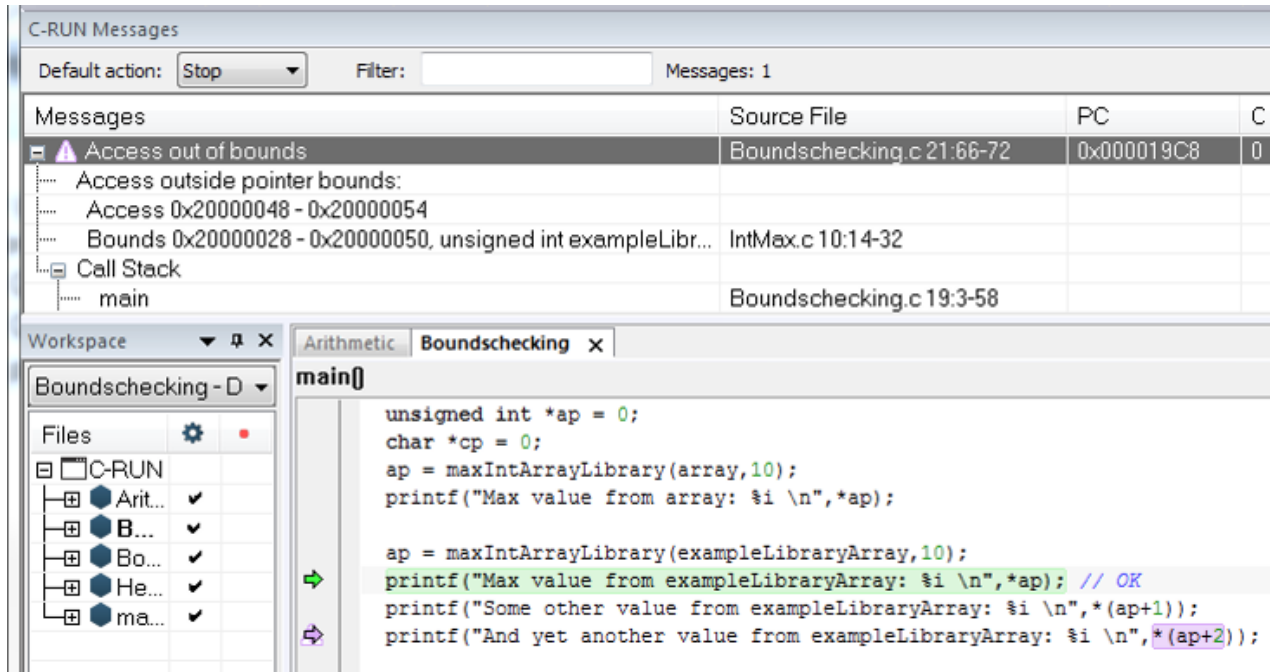
Arithmetic issues includes overflow, wraparound, conversion errors, division by zero and missing default labels in switch statements. Such errors can be detected by inserting specific instrumentation code at all places where a potential error can happen. Source level instrumentation often insert an if-statement or equivalent that checks the condition and either print something to stdout or write a special value to a port to log the issue. Analogously, a compiler can insert instructions to check the condition and somehow report the issue at runtime. Code size will increase more or less linearly with the number of operations to check.

Bounds issues is a very broad category of problems that include typical out-of-bounds issues like writing or reading outside the defined bounds of an array. But the out-of-bounds concept can be generalized to deal with anything that is accessed through a pointer regardless of type or size. This includes things like pointers to scalar objects on the stack, so if you happen to change a pointer, or someone with malicious intent does it for you, to something on the stack, a state-of-the-art bounds checker can detect if the new value of the pointer is within the bounds of a valid object.

Heap checking assures that the heap retains its integrity and does not leak allocated blocks over time. Efficient heap checking is essentially an exercise in library implementation, but knowing the internals of the associated compiler can be beneficial if some functionality can be treated much the same way as other compiler intrinsic functions. Integrity checking is typically done on each call to malloc, free and their friends, both in the C and C++ world. Heap integrity checking can be a real drag on performance if the heap is big, since the checks can entail traversing the full heap, so a way to decide the frequency of checks can be crucial for some applications.

Convenient runtime analysis and error checking with C-RUN

C-RUN is an extension to IAR Embedded Workbench and is available for Arm and RX. C-RUN is designed to be a natural part of your development workflow, when working in a traditional edit/build/debug cycle, running unit tests or doing integration tests. C-RUN provides you with extremely valuable feedback already as soon as the first iteration of code is about to be taken for a test drive. Thanks its tight integration into IAR Embedded Workbench, C-RUN can be part of the daily work for any developer.



C-RUN Messages

Default action: Stop Filter: Messages: 1

Messages	Source File	PC	C
Access out of bounds	Boundschecking.c 21:66-72	0x000019C8	0
Access outside pointer bounds:			
Access 0x20000048 - 0x20000054			
Bounds 0x20000028 - 0x20000050, unsigned int exampleLibr...			
Call Stack			
main			
Boundschecking.c 19:3-58			

Workspace

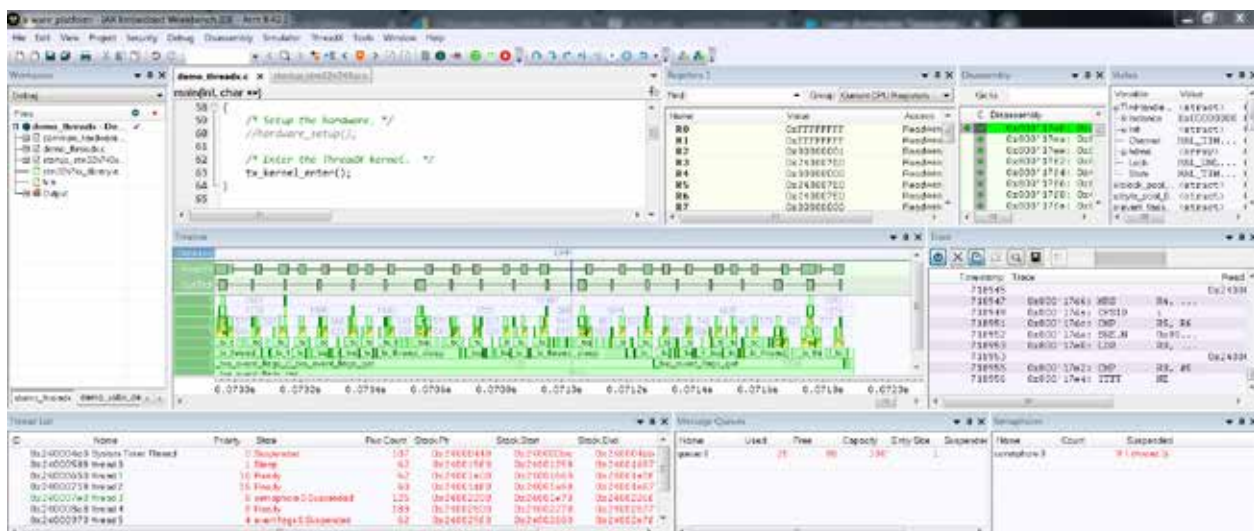
Arithmetic Boundschecking

```
main()
{
    unsigned int *ap = 0;
    char *cp = 0;
    ap = maxIntArrayLibrary(array, 10);
    printf("Max value from array: %i \\n", *ap);

    ap = maxIntArrayLibrary(exampleLibraryArray, 10);
    printf("Max value from exampleLibraryArray: %i \\n", *ap); // OK
    printf("Some other value from exampleLibraryArray: %i \\n", *(ap+1));
    printf("And yet another value from exampleLibraryArray: %i \\n", *(ap+2));
}
```

IAR Embedded Workbench

IAR Embedded Workbench is a complete C/C++ development toolchain for embedded applications. The toolchain offers leading code quality, outstanding optimizations for size and speed, as well as extensive debug functionality with a fully integrated debugger with simulator and hardware debugging support. C-RUN is fully integrated with the IAR Embedded Workbench IDE, which helps developers to ensure their code is safe and of high quality at an early stage, which also aids companies to shorten their time to market as impact of errors further down the line might be very time consuming and expensive.



IAR Embedded Workbench - Arm R42

main() { setup the hardware; // hardware_setup(); // enter the thread kernel; // kernel_entry(); }

Registers

Reg	Value	Access
R0	0x7FFFFFFF	Read/Write
R1	0x7FFFFFFF	Read/Write
R2	0x33333333	Read/Write
R3	0x248E7E0	Read/Write
R4	0x33333333	Read/Write
R5	0x248E7E0	Read/Write
R6	0x248E7E0	Read/Write
R7	0x33333333	Read/Write

Disassembly

PC	Instruction	Comment
718545	0x00000000	0x00000000
718547	0x00000000	0x00000000
718549	0x00000000	0x00000000
718551	0x00000000	0x00000000
718553	0x00000000	0x00000000
718555	0x00000000	0x00000000
718557	0x00000000	0x00000000
718559	0x00000000	0x00000000

Task List

Name	Priority	State	File Count	Stack Top	Stack Bottom
0x24000400 System Timer Thread	0	Ready	187	0x24000440	0x24000400
0x24000500 Thread 0	1	Ready	63	0x24001000	0x24001000
0x24000600 Thread 1	10	Ready	63	0x24001000	0x24001000
0x24000700 Thread 2	15	Ready	63	0x24001000	0x24001000
0x24000800 Thread 3	20	Ready	63	0x24001000	0x24001000
0x24000900 Thread 4	25	Ready	63	0x24001000	0x24001000
0x24000A00 Thread 5	30	Ready	63	0x24001000	0x24001000
0x24000B00 Thread 6	35	Ready	63	0x24001000	0x24001000
0x24000C00 Thread 7	40	Ready	63	0x24001000	0x24001000
0x24000D00 Thread 8	45	Ready	63	0x24001000	0x24001000
0x24000E00 Thread 9	50	Ready	63	0x24001000	0x24001000
0x24000F00 Thread 10	55	Ready	63	0x24001000	0x24001000
0x24001000 Thread 11	60	Ready	63	0x24001000	0x24001000
0x24001100 Thread 12	65	Ready	63	0x24001000	0x24001000
0x24001200 Thread 13	70	Ready	63	0x24001000	0x24001000
0x24001300 Thread 14	75	Ready	63	0x24001000	0x24001000
0x24001400 Thread 15	80	Ready	63	0x24001000	0x24001000
0x24001500 Thread 16	85	Ready	63	0x24001000	0x24001000
0x24001600 Thread 17	90	Ready	63	0x24001000	0x24001000
0x24001700 Thread 18	95	Ready	63	0x24001000	0x24001000
0x24001800 Thread 19	100	Ready	63	0x24001000	0x24001000

Message Queue

Home	Used	Free	Capacity	Entry Size	Suspended	How	Count	Expanded
main	1	15	16	256	1	main	1	1