ECLAIR considers the compiler: Why is this valuable?

Frank Büchner, 001 – 2020-09-28

Contents

1 Introduction

This demonstration shows that ECLAIR takes the compiler into account and considers it. This is important, because implementation-defined behavior of a compiler might decide about the output of a program and/or if e.g. a MISRA guideline is violated or not.

2 Prerequisites

This demonstration assumes that the following programs are in the path:

- ECLAIR (i.e. eclair_env.exe and eclair_report.exe)
- GCC (a compiler where int has 32 bits)
- Icc430 (a cross compiler for embedded systems from IAR, where int has 16 bits).

3 Application Files

3.1 Source File: main.c

There is only one source file and this source file is not changed during the demonstration.

```
int main(void)
{
    unsigned int a = 0x100;
    unsigned int b = 0x100;
    return (((a * b) + 2UL) == 2UL) ? 1 : 0;
}
```

Fig. 1 The contents of main.c

The parentheses in the return statement are for better readability and understandability and to avoid a violation of rule 12.3 of the MISRA C:2012 guidelines. The result is identical with or without parentheses; letting the standard operator precedence would do its job would give the same result.

3.2 Two makefiles

Two makefiles are used during this demonstration. One makefile (makefile_gcc) invokes gcc to build the executable; the other makefile (makefile_iar) invokes icc430 to build the executable.

3.2.1 Makefile_gcc

```
VERSION = 8.2.0
CC      = gcc
CFLAGS  = -Wall -g -std=c99
LDFLAGS =

EXE = prog.exe
OBJ = main.o

prog: $(OBJ)
        $(CC) $(CFLAGS) -o prog $(OBJ) $(LDFLAGS)

%.o: %.c
        $(CC) $(CFLAGS) -c $<

.PHONY: clean
clean:
        rm -rf $(EXE) $(OBJ)
```

Fig. 2 The contents of makefile_gcc

This makefile compiles main.c to main.o using gcc as compiler. Then it links main.o to the form the executable prog.exe.

The makefile target “clean” removes main.o and prog.exe, should they exist.

The command line option “-Wall” enables all warnings.

The command line option “-std=c99” sets the C language version to C99.

3.2.2 Makefile_iar

```
CC      = icc430
LINK    = xlink
CFLAGS  =
LDFLAGS = -S -f lnk430.xcl

PROG = aout.a43
OBJ = main.r43

prog: $(OBJ)
        $(LINK) $(OBJ) $(LDFLAGS)

%.r43: %.c
        $(CC) $(CFLAGS) $<

.PHONY: clean
clean:
        rm -rf $(PROG) $(OBJ)
```

Fig. 3 The contents of makefile_iar

This makefile compiles main.c to main.r43 using icc430 from IAR as compiler. Then it links main.r43 to the form the executable aout.a43, using the linker file lnk430.xcl.

The makefile target “clean” removes aout.a43and main.r43, should they exist.

By default, the C language version of the IAR compiler is C99.

4 Files Related to ECLAIR

4.1 Batch File: analyze.bat

Usually a batch file “analyze.bat” conducts analysis with ECLAIR.

4.1.1 First Part

The first part of this batch file defines some environment variables.

```
set CC_ALIASES=gcc icc430
```

Fig. 4 The environment variable CC_ALIASES directs ECLAIR to intercept gcc and icc430

4.1.2 Middle Part

In the middle part of this batch file conducts the analysis with ECLAIR.

```
eclair_env "-eval_file=analyze.ecl" -- make
```

Fig. 5 The command “eclair_env” conducts the actual analysis

The file “analyze.ecl” controls the analysis. In this file, ECLAIR is set up to check for violations of the MISRA C:2012 guidelines.

The part of the command line after “--“ is the command that is used to build the project that shall be analyzed by ECLAIR. In our case, make.exe will execute the commands present in the file “makefile” in the current directory.

4.1.3 Last Part

The last part of this batch file generates the ECLAIR reports.

5 Demonstration

5.1 Using makefile_gcc

We make the contents of “makefile” identical to the contents of makefile_gcc. Then we execute analyze.bat to start the analysis.

```
c:\Projects\ECLAIR\Hitex\GCC-IAR_MISRA-10-7>analyze.bat
```

Fig. 6 Start the analysis

```
c:\Projects\ECLAIR\Hitex\GCC-IAR_MISRA-10-7>make clean
rm -rf prog.exe main.o
```

Fig. 7 Remove existing files

Because makefile currently is identical to makefile_gcc, prog.exe and main.o are removed.

```
c:\Projects\ECLAIR\Hitex\GCC-IAR_MISRA-10-7>eclair_env "-eval_file=analyze.ecl" -- make
gcc -Wall -g -std=c99 -c main.c
gcc -Wall -g -std=c99 -o prog main.o
```

Fig. 8 The effects of the command “eclair_env”

The command ECLAIR_env uses the settings in analyze.ecl for analysis and calls make to build the project. Because makefile currently is identical to makefile_gcc, main.c is compiled to main.o and then it is linked to prog.exe.

service	violation	caution	information
B.BUGFIND.deadcode.UnreachableCode		1	
B.EXPLAIN			1
MC3.D1.1	6	1	
MC3.D4.1	27		
MC3.D4.6	2		

Fig. 9 The result of the analysis of ECLAIR (case gcc)

Please note that no rules of MISRAC C:2012 are violated (but some directives).

5.2 Using makefile_iar

We make the contents of “makefile” identical to the contents of makefile_iar. Then we execute analyze.bat to start the analysis.

```
c:\Projects\ECLAIR\Hitex\GCC-IAR_MISRA-10-7>analyze.bat
```

Fig. 10 Start the analysis

```
c:\Projects\ECLAIR\Hitex\GCC-IAR_MISRA-10-7>make clean
rm -rf aout.a43 main.r43
```

Fig. 11 Remove existing files

Because makefile currently is identical to makefile_iar, aout.a43 and main.r43 are removed.

```
c:\Projects\ECLAIR\Hitex\GCC-IAR_MISRA-10-7>eclair_env "-eval_file=analyze.ecl" -- make
icc430  main.c

   IAR C/C++ Compiler V7.12.1.987/W32 for MSP430
   Copyright 1996-2018 IAR Systems AB.
   PC-locked license - IAR Embedded Workbench for Texas Instruments MSP430, 8K KickStart Edition 7.12

 34 bytes of CODE memory

Errors: none
Warnings: none
xlink main.r43 -S -f lnk430.xcl
```

Fig. 12 The effects of the command “eclair_env”

The command ECLAIR_env uses the settings in analyze.ecl for analysis and calls make to build the project. Because makefile currently is identical to makefile_iar, icc430 is used to compile main.c to main.r43 and xlink is used to link main.r43 to aout.a43.

Please note: The IAR compiler reports neither warnings nor errors.

service	violation	caution	information
B.BUGFIND.deadcode.UnreachableCode		1	
B.EXPLAIN			1
MC3.D1.1	9	4	
MC3.D4.1	27		
MC3.D4.6	2		
MC3.R10.7	1		

Fig. 13 The result of the analysis of ECLAIR (case IAR)

Please note that in case the IAR compiler is used, a violation of rule 10.7 from the MISRA C:2012 guidelines is reported.

service MC3.R10.7: If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type (1 violation)

☰ violation for MC3.R10.7
main.c:24.18-24.24: 🔎 `*' multiplication operator has essential type 16-bit unsigned integer and standard type `unsigned'
main.c:24.26: 🔎 `+' addition operator expects a 32-bit unsigned integer

Fig. 14 Details of the violation of rule 10.7

```
20 int main(void)
21 {
22         unsigned int a = 0x100;
23         unsigned int b = 0x100;
24         return (((a * b) + 2UL) == 2UL) ? 1 : 0;
☰ MC3.R10.7 `*' multiplication operator has essential type 16-bit unsigned integer and standard type `unsigned'
`*' multiplication operator has essential type 16-bit unsigned integer and standard type `unsigned'
`+' addition operator expects a 32-bit unsigned integer 🔎
25 }

20 int main(void)
21 {
22         unsigned int a = 0x100;
23         unsigned int b = 0x100;
24         return (((a * b) + 2UL) == 2UL) ? 1 : 0;
☰ MC3.R10.7 `*' multiplication operator has essential type 16-bit unsigned integer and standard type `unsigned'
`*' multiplication operator has essential type 16-bit unsigned integer and standard type `unsigned'
`+' addition operator expects a 32-bit unsigned integer 🔎
25 }
```

Fig. 15 The source code where rule 10.7 is violated

6 Explanation

As we have seen, just using a different compiler causes changes in the report of ECLAIR about violated MISRA rules. The fundamental reason of this lies in the fact that the two compilers are intended for different microcontroller architectures. In the case at hand, the size of an integer is different: The gcc compiler features 32 bit integers; whereas the IAR compiler targets the MSP430 microcontroller architecture from Texas instruments, where the integer size is 16 bit. Because the size of an integer is inherent to the compiler, this is implement-defined behavior.

This different integer size causes different behavior of the program.

6.1 Insights from the Debugger

The different behavior of the program is demonstrated by stepping through the code using a debugger.

6.1.1 Insights from CDT

We can debug the code compiled by gcc using the CDT debugger in Eclipse.

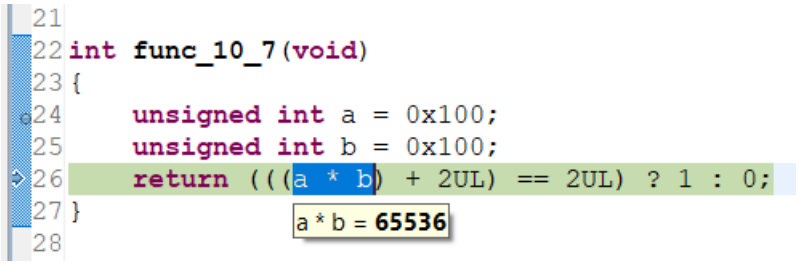

Fig. 16 Insights from CDT, part 1

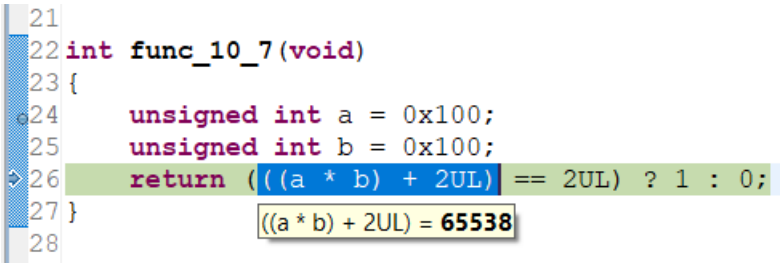

Fig. 17 Insights from CDT, part 2

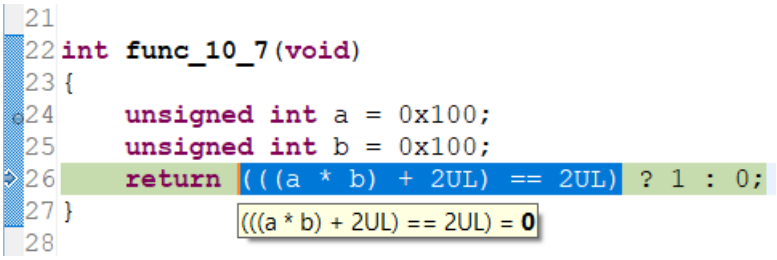

Fig. 18 Insights from CDT, part 3

It should be clear that the return value using the gcc compiler is 0.

6.1.2 Insights from C-Spy

We can debug the code compiled by icc430 using the C-Spy debugger in the Embedded Workbench form IAR.

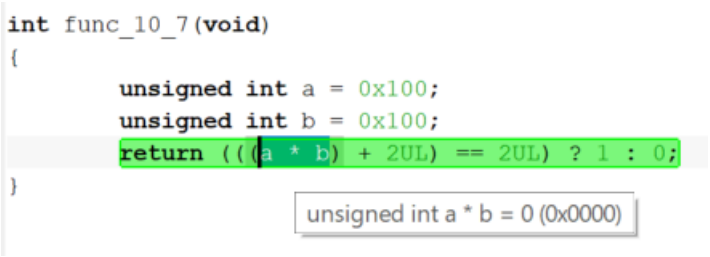

Fig. 19 Insights from C-Spy, part 1

The figure above shows the difference between 32 bit integer size (where a*b = 65536) and 16 bit integer size (where a*b = 0).

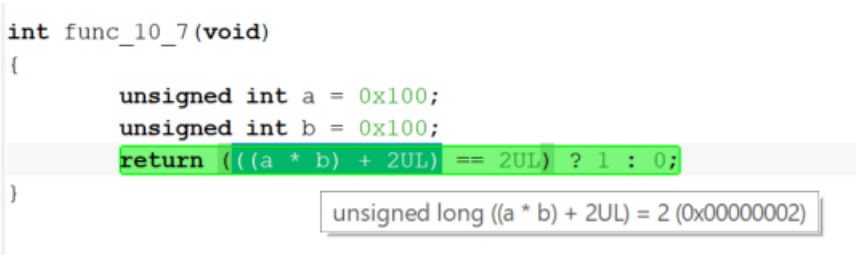

Fig. 20 Insights from C-Spy, part 3

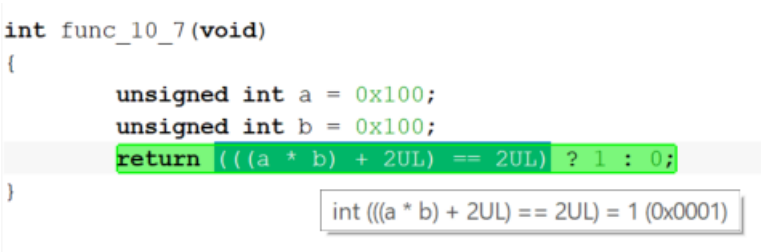

Fig. 21 Insights from C-Spy, part 3

It should be clear that the return value using the icc430 compiler is 1.

6.2 More Insights from ECLAIR

For both compilers, ECLAIR reports possible unreachable code. If we take a closer look, we find out that the location of the unreachable code also depends on the compiler that is used to build the project.

```
20 int main(void)
21 {
22         unsigned int a = 0x100;
23         unsigned int b = 0x100;
24         return (((a * b) + 2UL) == 2UL) ? 1 : 0;
☰ B.BUGFIND.deadcode.UnreachableCode [1] This statement is never executed
25 }
```

Fig. 22 Location of unreachable code using gcc

```
20 int main(void)
21 {
22         unsigned int a = 0x100;
23         unsigned int b = 0x100;
24         return (((a * b) + 2UL) == 2UL) ? 1 : 0;
☰ B.BUGFIND.deadcode.UnreachableCode [1] This statement is never executed
25 }
```

Fig. 23 Location of unreachable code using icc430

Hence: Using gcc, always 0 is returned; using IAR, always 1 is returned.

7 Conclusion

Neither the source code, nor the analysis setup of ECLAIR (i.e. the analyze.bat and analyze.ecl) nor the report generation of ECLAIR has changed. The only thing that has changed was how the project was build, i.e. the contents of the makefile, and in consequence, the compiler that was used.

In consequence, considering just the source code is not enough to decide if a potential bug or MISRA violation is present or not. Considering just the source code can lead to false positives (or false negatives).

8 Version Used

This document was created using

- ECLAIR V3.8.1
- gcc V8.2
- IAR icc430 V7.12.1.

9 Author

Frank Büchner
Dipl.-Inform.
Hitex GmbH
Greschbachstr. 12
D-76229 Karlsruhe
Tel.: +49-721-9628-125
frank.buechner@hitex.de

Any comments are appreciated.