

RE: Guarantee of Atomic Statement and Test Compilation

Michael Burns [mjb@cosmic-us.com]

Sent: Friday, January 22, 2010 4:45 PM

To: David Ashley; johnb@cosmic-us.com

Hi David,

There is no easy solution for this topic. We understand the need for such a control in some applications, but this is very target dependant. Some processors cannot support at all atomic operations (RISC machines for instance where any memory modification needs a load and a store) and always need to control the interrupt flag, some others allow atomic accesses while using several instructions (PIC for instance where any operation can target the memory, so a += b, is translated into a load b followed by an addition instruction adding the register to memory value of a and storing the result in memory a. The code produces 2 instructions, but the memory modification is atomic). There are also processors including such a feature in their instruction set (ST10, C166) up to a limited number of instructions. Due to the wide variety of support and lack thereof it becomes difficult to support a common mechanism across the various targets and frankly there hasn't been sufficient interest to justify the effort.

However, one suggestion that might help you to check and optimize. You could tag the critical sequences with ASM comments that could then be parsed in the listing files by a utility or script to determine if the critical sequence produced more than one instruction (in the case of STM8 and HC08). This could be done with a simple macro such as the following.

```
#define ATOMIC(x) {\
    _asm("; atomic start"); \
    x; \
    _asm("; atomic end"); \
}
```

char c;

```
void f(void)
{
    ATOMIC(++c);
}
```

The resulting listing contains then the proper comments:

```
18          ; 11    ATOMIC(++c);
20          ; atomic start
23 0000 725c0000    inc _c
26          ; atomic end
28          ; 12    }
29 0004 81        ret
```

so any external tool (target dependant) can check if the atomicity is verified and there is no need for special compiler options. Such a macro could be complemented by an equivalent called IT_PROTECTED for example, using the same structure but producing the disable and enable interrupt instructions in place of the comments, or this could be even controlled by a second argument of the initial macro.

Hope that helps.

Best Regards,
Mike

Michael Burns
Cosmic Software Inc.
17 Bridge St. STE 101
Billerica MA 01821
Phone (978) 667-2556 Fax (978) 667-2560
Email: mjb@cosmic-us.com
WWW: www.cosmic-software.com

From: David Ashley [mailto:DAshley@cequentgroup.com]
Sent: Saturday, January 16, 2010 5:41 PM
To: mjb@cosmic-us.com; johnb@cosmic-us.com
Subject: Guarantee of Atomic Statement and Test Compilation

Hi Mike and John,

Please let me know if you have any thoughts here:

http://www.dtashley.com/proftopics/general_embedded_toolchain_notes/guarantee_atomic_statement_test_compilation/

We find ourselves all the time doing things like:

```
volatile _Bool x;  
volatile UINT16 z;
```

```
DI();  
x = TRUE;  
EI();
```

```
DI()  
z = 4221;  
EI();
```

In both cases, we've looked at the assembly-language and such statements always seem to be atomic. But we still find ourselves using critical section protocol because we are afraid of what could happen if one day the compiler decides to do it differently.

Any thoughts or ideas on the content of the web page?

Thanks, Dave A.

This message is for the designated recipient(s) only and may contain privileged or confidential information. If you are not the intended recipient, please advise the sender immediately by reply email and delete this message and any attachments without retaining a copy. No representation is made that this email or any attachments are free of viruses.