
EXAMPLE 4

Description:
Using a function without a "prolog" and "epilog".

Notes:
GCC allows to generate a function without parameter transfer on entry and exit using __naked__ attribute. Proper entry and exit handling is the responsibility of the application programmer.

IAR relies on __task extended keyword. This works fine for standard functions. However, it seems to be neglected by IAR compiler for interrupt functions.

This is **not for beginners** - neither in GCC nor in IAR !

Danger warning for IAR code: **HIGH**

GCC code example:

```
void TestStdTask(void)   __attribute__ ((naked));

void TestStdTask(void)
{
          static long Var32;

          Var32 += 1;
}
```

GCC assembler output:

```
          .align    2
          .global   TestStdTask
          .type     TestStdTask,function
TestStdTask:
          @ Naked Function: prologue and epilogue provided by programmer.
          @ args = 0, pretend = 0, frame = 0
          @ frame_needed = 1, current_function_anonymous_args = 0
          ldr       r3, .L6
          ldr       r3, [r3, #0]
          add       r2, r3, #1
          ldr       r3, .L6
          str       r2, [r3, #0]
.L7:
          .align    2
.L6:
          .word     Var32.0
.Lfe5:
          .size     TestStdTask,.Lfe5-TestStdTask
```

Acknowledgment: Ivan Paulik @ PASAT, Bratislava

Equivalent IAR C code:

```
__task __irq __arm void TestIntTask (void);

__irq __arm __task void TestIntTask (void)
{
          static long Var32;

          Var32 += 1;
}

__task void TestStdTask (void)
{
          static long Var32;

          Var32 += 1;
}
```

IAR assembler output:

```
     77          __task __irq __arm void TestIntTask (void);
     78

   \                                In section .text, align 4, keep-with-next
     79          __irq __arm __task void TestIntTask (void)
     80          {
   \                        TestIntTask:
   \   00000000   04E04EE2           SUB      LR,LR,#+4
   \   00000004   03402DE9           PUSH     {R0,R1,LR}
     81              static long Var32;
     82
     83              Var32 += 1;
   \   00000008   0C009FE5           LDR      R0,??TestIntTask_0  ;; ??Var32
   \   0000000C   001090E5           LDR      R1,[R0, #+0]
   \   00000010   011081E2           ADD      R1,R1,#+1
   \   00000014   001080E5           STR      R1,[R0, #+0]
     84          }
   \   00000018   0380FDE8           LDM      SP!,{R0,R1,PC}^  ;; return
   \                        ??TestIntTask_0:
   \   0000001C   ........           DC32     ??Var32

   \                                In section .bss, align 4
   \                        ??Var32:
   \   00000000                     DS8 4
     85

   \                                In section .text, align 4, keep-with-next
     86          __task void TestStdTask (void)
     87          {
     88              static long Var32;
     89
     90              Var32 += 1;
   \                        TestStdTask:
   \   00000000   0248               LDR      R0,??TestStdTask_0  ;; ??Var32_1
   \   00000002   0168               LDR      R1,[R0, #+0]
   \   00000004   491C               ADDS     R1,R1,#+1
   \   00000006   0160               STR      R1,[R0, #+0]
     91          }
   \   00000008   7047               BX       LR               ;; return
   \   0000000A   C046               Nop
   \                        ??TestStdTask_0:
   \   0000000C   ........           DC32     ??Var32_1

   \                                In section .bss, align 4
   \                        ??Var32_1:
   \   00000000                     DS8 4
```

Acknowledgment: Ivan Paulik @ PASAT, Bratislava