Application Note: AN10022

How to use inline assembly

This application note is a short how-to on programming/using the xTIMEcomposer tools. It shows how to use inline assembly.

Required tools and libraries

This application note is based on the following components:

- xTIMEcomposer Tools - Version 14.0.0

Required hardware

Programming how-tos are generally not specific to any particular hardware and can usually run on all XMOS devices. See the contents of the note for full details.

1 How to use inline assembly

The asm statement can be used to embed assembly code inside a C or XC function. The basic syntax for doing this is:

```
asm("assembly template" : output operands : input operands)
```

The following example calls the add assembly instruction to add 5 to the value y, storing the result in x:

```
int x, y = 5;

asm("add %0, %1, %2" : "=r"(x) : "r"(y), "r"(5));
```

Each operand is described by an operand constraint string followed by an expression in parentheses. The "r" in the operand constraint string indicates that the operand must be located in a register. The "=" indicates that the operand is written. Each output operand must have "=" in its constraint. Operands can be referred to in the assembler template using an escape sequence of the form %num where num is the operand number.

If the assembly code overwrites specific registers, this can be described by using a third colon after the input operands, followed by the names of the clobbered registers as a comma-separated list of strings:

```
asm("get r11, id; mov %0, r11" : "=r"(x) : /* no inputs */ : "r11");
```

If an asm statement has output operands, the compiler assumes the statement has no side effects apart from writing to the output operands. This may result in the compiler removing the asm statement if the output operands are are unused. To mark an asm statement as having side effects use volatile:

```
asm volatile("in %0, res[%1]" : "=r"(x) : "r"(p));
```

If the asm statement accesses memory, add "memory" to the list of clobbered registers:

```
asm volatile("stw %0, dp[a]" : /* no outputs */ : "r"(x) : "memory");
```

This prevents the compiler caching memory values in registers around the asm statement.