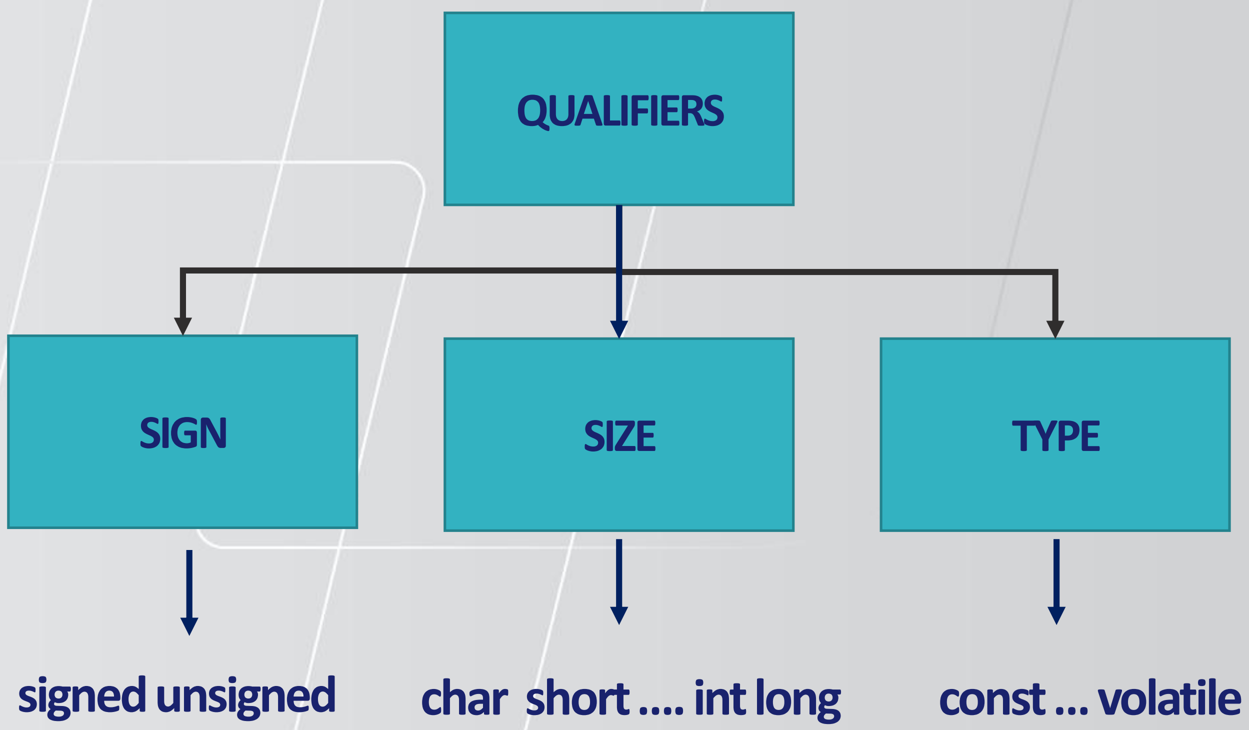
QUALIFIERS

19 Aug 2024



OUTLINE

1. volatile Type Qualifier

2. Optimization Option

1. volatile Type Qualifier

What volatile means:

- ➢ **volatile** keyword is a type qualifier that is applied to a variable when it is declared.

- ➢ The declaration of a variable as volatile tells the compiler that the variable can be modified at any time by another entity that is external to the implementation.

  For example:
  - By the operating system
  - By hardware

- ➢ This declaration ensures that the compiler does not optimize any use of the variable on the assumption that this variable is unused or unmodified.

1. volatile Type Qualifier

How to use volatile:

- Declare a variable volatile, include the keyword volatile before or after the data type in the variable definition.

```
volatile int var;                  //Declare var to be a volatile integer
int volatile var;
volatile unsigned char *p;        //Pointer to a volatile unsigned char
int * volatile p;                  //Volatile pointer to signed integer
int volatile * volatile p;        //volatile pointer to a volatile variable
```

1. volatile Type Qualifier

When to use volatile:

- Memory-mapped peripheral registers.
- Global variables modified by an interrupt service routine.
- Global variables accessed by multiple tasks within a multi-thread application.
- Inline assembly code.

```
__asm__ volatile ("cpsid i");
```

1. volatile Type Qualifier

Potential problems when not using volatile:

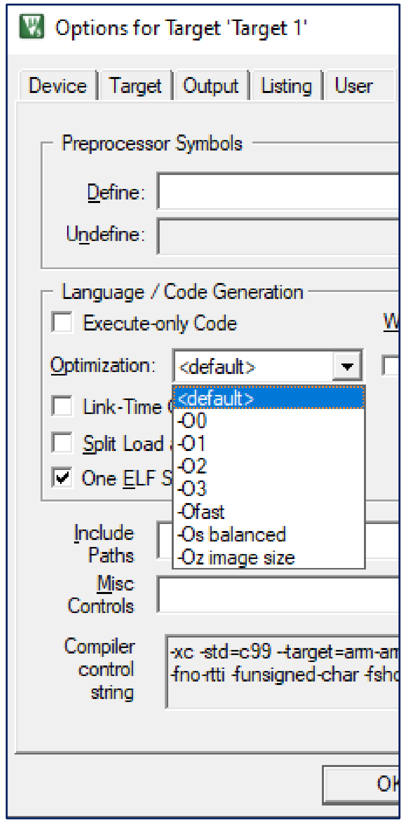
- The compiler assumes that its value cannot be modified from outside the scope that it is defined in. Therefore, the compiler might perform unwanted optimizations and generate unintended code or remove intended functionality.

- This problem can manifest itself in various ways:

  + Code might become stuck in a loop while polling hardware.

  + Optimization might result in the removal of code that implements deliberate timing delays.

2. Optimization Option



Options for Target 'Target 1'

Device | Target | Output | Listing | User

Preprocessor Symbols

Define:

Undefine:

Language / Code Generation

☐ Execute-only Code

Optimization: <default>

<default>
-O0
-O1
-O2
-O3
-Ofast
-Os balanced
-Oz image size

☐ Link-Time (

☐ Split Load

☑ One ELF S

Include Paths

Misc Controls

Compiler control string: -xc -std=c99 -target=arm-ar -fno-rtti -funsigned-char -fshc

No	Optimize level	Description
1	-O0	Does not perform optimization
2	-O1 | -O2 | -O3	Specify the level of optimization to be used when compiling source files. A higher number implies a higher level of optimization for performance.
3	-Ofast	Enables all the optimizations from -O3 together with other aggressive optimizations that might violate strict compliance with language standards.
4	-Os	Performs optimizations to reduce the code size at the expense of a possible increase in execution time. This option aims for a balanced code size reduction and fast performance.
5	-Oz	Optimizes for smaller code size.

Thank you