**Technical Documentation on usage of C Qualifier** “**Volatile**”

This document illustrates answers to following questions.

1. What is Compiler Optimization ?
2. Why does Compiler do Optimization ?
3. What is the need of Volatile Qualifier ?
4. What happens when we use Volatile qualifier ?
5. Practical Applications of Volatile qualifier.

1)What is Compiler Optimization ?

Optmization is a technique where compiler tries to minimize some attributes of an executable computer program. This can be done in 2 ways.The most common way is to minimize the time taken to execute a program; a less common one is to minimize the amount of memory occupied

Execution time = Operation count \* Machine cycles per operation

***To reduce the Operation count:***

Compiler reduces the no of arithmetic operations as done below:

Int x=1+2;

x++;

To reduce the arithmetic operations compiler does below,

Int x=3;

x++;

This is a simple example to demonstrate how operations count can be reduced.

***To reduce the Machine cycles :***

In fact ,to reduce machine cycles per operation,one has to alter the underlying Hardware of processor like reducing the transistor size which in turn decrease clock time but increases power consumption or reducing Clocks per instruction (CPI) which is an effective average, is affected by instruction-level parallelism and by instruction complexity.If digged ,it takes you to into other ocean of embedded systems wing called “CPU performance equation”,which is not much required to explain Volatile qualifier behavior.

It is definitely not possible to reduce machine cycles per operation,rather we reduce no of operations required to access memory in-directly reducing the no of Machine cycles per operation for that particular variable by storing variable value in the CPU registers.

Here an intermediate question triggered is,”Why does it take less machine cycles to access registers compared to Memory ?”

Registers are a core part of the CPU ,these are circuits which are literally wired directly to the ALU, which contains the circuits for arithmetic, and much of the instruction set of a CPU will be tailored for working against registers rather than memory locations. Accessing a register's value will typically require very few clock cycles (likely just 1), but as soon as memory is accessed, things get more complex and cache controllers / memory buses get involved and the operation is going to take considerably more time.  Keeping data in a register means no data transfer overhead. So,obviously accessing a register takes relatively less time.

2) Why does Compiler do Optimization ?

Straight forward answer is, to save time i,e make a compiled program much faster by doing optimizations that can only be done at the assembly (machine) language level for the target hardware.

3) What is the need of Volatile Qualifier ?

Interestingly, by adding Volatile qualifier to a variable,we ask the compiler to switch off Optimization for that particular variable.We are instructing the compiler to read value of that variable from memory only,each time it is encountered in the program.

4) What happens when we use Volatile qualifier ?

Once compiler sees “volatile” qualifier,the corresponding variable’s value is not cached in the Processor register.Below codes depict the effect well.

***Demonstration using C equivalent Assembly code.***

Let’s consider a 8-bit status register that is memory mapped at address 0x1234(say);

As per requirement ,compiler need to poll the status register(Uint8\_t \* P\_Register) until it becomes Non-Zero.

Uint8\_t \* P\_Register =(Uint\_t\*) 0x1234;

Do

{

-----some operation---

}while(0==\*P\_Register) i.e wait for register to read Non-Zero.

The complier will generate ALP(Here for an 16-Bit x86 processor) as

Mov P\_Register,#0x1234

Mov a,@P\_Register

Loop:

Bz loop

This code fails as the value is read into accumulator before the loop begin.

The solution is using Volatile qualifier as below

Uint8\_t Volatile \* P\_Register =(Uint\_t\*) 0x1234;

The equivalent ALP is now

Mov P\_Register,#0x1234

Loop:

Mov a,@P\_Register

Bz loop

This makes the compiler to read value everytime from Memory after the Loop begin,thus the expected polling is done.

5)What are the Practical Applications of Volatile qualifier ?

A variable should be declared Volatile,when there are chances for that variable value to be changed outside the program where it is declared.Strictly classified as below.

1. If in your embedded system,a variable’s value declared in your code depends on the Memory-Mapped peripheral registers,then Volatile qualifier is a must.
2. By chance, there are any Global variables to be modified by an Interrupt service routine,those global variables need Volatile qualifier in-evitably.
3. In case your program consists of Global variables accessed by multiple tasks within a Multi-threaded application,then Volatile qualifier is a must for those Global variables.