О volatile замолвлю я слово

Глядя на полемику по поводу использования ключевого слова volatile, возникшую в https://radiokot.ru/forum/viewtopic.php?f=62&t=37190&start=7140, я, признаться, был обескуражен той лёгкостью подхода, с которой некоторые авторы навязывают свою точку зрения и учат молодёжь дурному: не использовать это слово, использовать какие-то «ключи оптимизации» вместо него и т.д. и т.п.

В скобках отмечу, что исходя из своего опыта преподавания программирования микроконтроллеров, я знаю, что ровно 100% студентов налетают на «грабли» с **volatile**. И это - неспроста.

Всё дело в непонимании для чего оно нужно и зачем его использовать.

Итак, глянем, что говорится о ключевом слове **volatile** в документации к языкам C и C++. Я не буду приводить километровых цитат, а только приведу основной смысл. Итак:

> Ключевое слово **volatile** информирует компилятор, что значение переменной может меняться извне.

Что это значит на практике? А именно то, что сказано. То есть, если переменная обозначена как **volatile**, то её значение может произвольно меняться в любой момент времени.

Что из этого следует? А следует из этого простой вывод: мы не можем гарантировать, что переменная в следующий момент времени будет иметь то же значение, что и в предыдущий.

Поэтому, компилятор, встречая **volatile**-переменную «понимает», что любое действие с ней (а это лишь два возможных действия - чтение и запись) - следует производить в точности как указал программист, без всяких оптимизаций.

Преимущества слова **volatile** перед многомудрыми советами «использовать ключи оптимизации», «прагмы» и «атрибуты» и прочими «а у меня и так всё работает» - очевидны:

1. Слово volatile - это стандарт. То есть его обязан понимать и использовать любой компилятор C/C++. Вывод - программа более переносима.
2. Ключи оптимизации распространяются на отдельный файл. Слово **volatile** - распространяется на отдельную переменную или поле структуры. То есть, с точки зрения оптимизации, за которую боролись сторонники «ключей оптимизации, прагм и атрибутов», именно слово **volatile** позволяет выполнить более точную оптимизацию программы.

3. Ключи компиляции привязаны к конкретному компилятору. Отсюда непереносимость программы. Кроме того, выставлять в программе из десятков, а то и сотен файлов индивидуальные ключи оптимизации для каждого из файлов – это очень на любителя извращений.
4. Прагмы и атрибуты ещё более, чем ключи оптимизации, привязаны к конкретному компилятору. Отсюда ещё большая непереносимость программы.

Итак, **volatile** – это стандарт и переносимость плюс облегчение себе жизни в плане экспериментов с ключами оптимизации.

Разумеется, что и прагмы и атрибуты и ключи оптимизации – **вещи нужные и важные**. Но не надо пытаться членом дрова рубить – и член угробишь и печь не растопишь. Каждой задаче – свой инструмент.

А теперь – слайды! Ну то есть примеры – зачем и для чего нужно использовать **volatile**.

Запись в аппаратный регистр

Итак, предположим, некто В.Пупкин начал писать программу. И есть у него байтовый регистр UART FIFO DATA, выглядящий для микроконтроллера как ячейка памяти, расположенная по адресу 0x010A Запись в этот регистр приводит к записи байта в FIFO UART, а чтение из него, соответственно к чтению из FIFO UART.

В.Пупкин, начитавшийся советов о том, что **volatile** есть зло, уверенно пишет тест-программу (инициализацию UART опустим):

```
unsigned char* udr = (unsigned char*)0x010A;
*udr = 'A';
```

и с любовью во взоре наблюдает буковку 'A' на экране терминала. Затем, В.Пупкин, как истинный говноайтишнег решает вывести своё имя на экран:

```
unsigned char* udr = (unsigned char*)0x010A;
*udr = 'B';
*udr = 'a';
*udr = 'c';
*udr = 'я';
```

И что он видит? А видит он только букву 'я' – последнюю букву. И начинает рвать волосы на теле, если таковые имеются, ругать землю, небо, страну, террана и вообще все на свете. Но, как обычно, виноват сам В.Пупкин.

Рвёт волосищи и ругается он до тех пор, пока не докладывается, что надо бы почитать умных людей и понять – почему от его имени осталась последняя буква.

Объяснение просто. Многие компиляторы, даже при отключённой оптимизации, кое-какую оптимизацию всё же производят. И компилятор, абсолютно законно, решает, что если в одну и ту же переменную производится несколько записей подряд, то он вправе выкинуть все операции записи, кроме последней. А что? Результат то не меняется. Для обычной переменной, разумеется, не меняется. Но у нас то переменная необычная.

Совершенно меняется результат выполнения кода, если написать:

```
volatile unsigned char* udr = (unsigned char*)0x010A;
*udr = 'В';
*udr = 'а';
*udr = 'с';
*udr = 'я';
```

На экране появится имя 'Вася' и наш программист будет удовлетворён полностью. Почему так произойдёт – понять не сложно. Раз переменная помечена как **volatile** – то запись в неё будет производиться каждый раз, вне зависимости от того, что там кажется компилятору.

Пример этот не такой надуманный как кажется. Буфера FIFO в UART и не только в UART – это вещь распространённая.

Чтение из аппаратного регистра

Предположим, что наш В.Пупкин опрашивает состояние входа GPIO и ждёт нажатия кнопки. Пусть регистр состояния GPIO байтовый и расположен как ячейка памяти по адресу 0x0507. Проверять будем бит 3. Пусть при нажатой кнопке в бит 3 выставляется «нуль», а если кнопка не нажата, то в бите 3 будет «единица».

```
unsigned char* gpio = (unsigned char*)0x0507;
while( (*gpio) & (1<<3) ) {} // Ждём нажатия кнопки
```

Тут сложно точно сказать, что произойдёт. Но многие компиляторы соптимизируют этот цикл до однократного чтения из регистра gpio и затем уйдёт в бесконечный цикл, если в момент опроса кнопка не нажата.

И опять же – запись:

```
volatile unsigned char* gpio = (unsigned char*)0x0507;
while( (*gpio) & (1<<3) ) {} // Ждём нажатия кнопки (бит 3 == 0)
```

решит проблему опроса регистра. Опрос регистра **gpio** будет производиться каждую итерацию цикла.

Переменные, модифицируемые в обработчиках прерывания

Помимо обработки аппаратных регистров, ключевое слово **volatile** очень полезно при использовании переменных, которые изменяются в прерывании, а обрабатываются вне прерываний. Пример:

```
// Переменная хранит посленее считанное значение из UART
char udr_data;

// Прерывание по приходу байта в UART
ISR_UART_RECV(){
	udr_data = UDR; /* читаем данные в переменную */
}

// Какойто основной цикл
..
udr_data=0;
while(1) {
	switch(udr_data) {
		case '1':
			// Пришло 1 - что-то делаем
		break;
		case '2':
			// Пришло 2 - что-то делаем
		break;
	}
}
```

То есть идёт постоянная проверка значения переменной **udr_data** и в зависимости от значения в ней производится какое-то действие. Всё логично. С первого взгляда.

Но... работать такой код часто не будет. Многие компиляторы вообще выкинут все тело **switch-case** потому как решат, что если переменной присвоен 0 - то она никогда не станет равной '1' или '2'. И, следовательно, зачем лишний код? Мёртвый цикл лучше. И короче.

Точно так же тут поможет объявление переменной как **volatile**:

```
// Переменная хранит посленее считанное значение из UART
volatile char udr_data;
```

В этом случае **switch-case** будет работать как надо - каждый раз проверяя значение переменной **udr_data**.

Заключение

Разумеется, что во всех приведённых примерах можно было бы добиться работоспособности кода и без **volatile**.

Но только для этого надо было бы написать примеры для каждого конкретного компилятора: GCC, IAR, KEIL и других.

А вот ключевое слово **volatile** – универсально и бьёт точно в цель – то есть в ту переменную, которая особенная и не задевает код, не относящейся к другим объектам.

Если вы загляните в исходник библиотек для микроконтроллеров, то увидите, что все регистры там объявлены как **volatile**. И это не с проста.

Проще всегда делать правильно, чем сэкономить десяток байт кода, но постоянно рисковать нарваться на грабли. Адепты мантры – «а у меня и так работает» – часто нарываются на очень неожиданное поведение кода в самый неподходящий момент времени. Удачи им в скачке по граблям.

Разумеется, что использовать ключевое слово **volatile** нужно с умом, то есть применять его только к тем переменным, которые либо изменяются аппаратно (как на чтение, так и на запись), либо модифицируются в прерывании, а проверяются вне его.

Конечно, бывают и более сложные случаи. Но нельзя объять необъятного и потому думайте сами, где надо ставить **volatile**, а где оно лишнее.

Счастливой разработки!