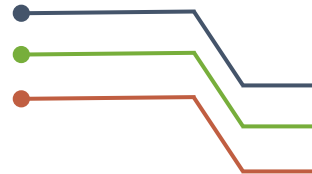
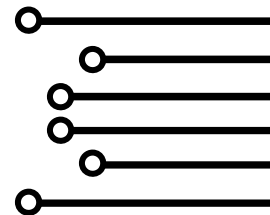




C Embebido



Calificadores

- Se utilizan para modificar la propiedad de una variable.
- Veremos dos palabras claves:

const

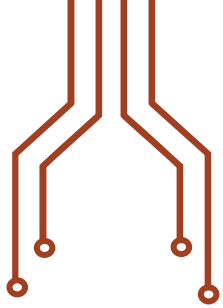
volatile

Const

```
const int dato = 10;  
float const data = 5.1;
```

Const declara una variable no modificable, por lo que sería sólo de lectura.

Hagamos un ejemplo

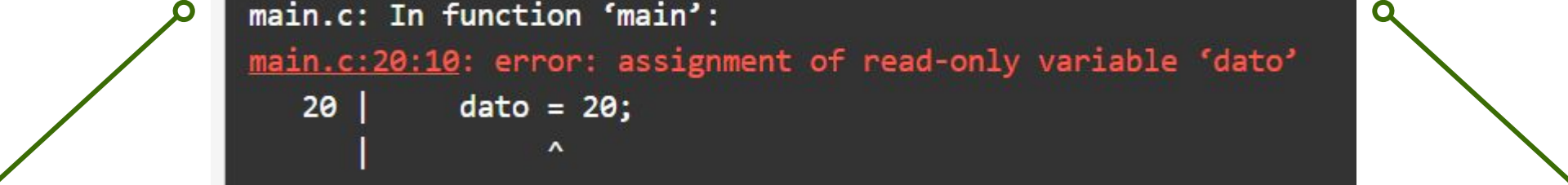


Compilation failed due to following error(s).

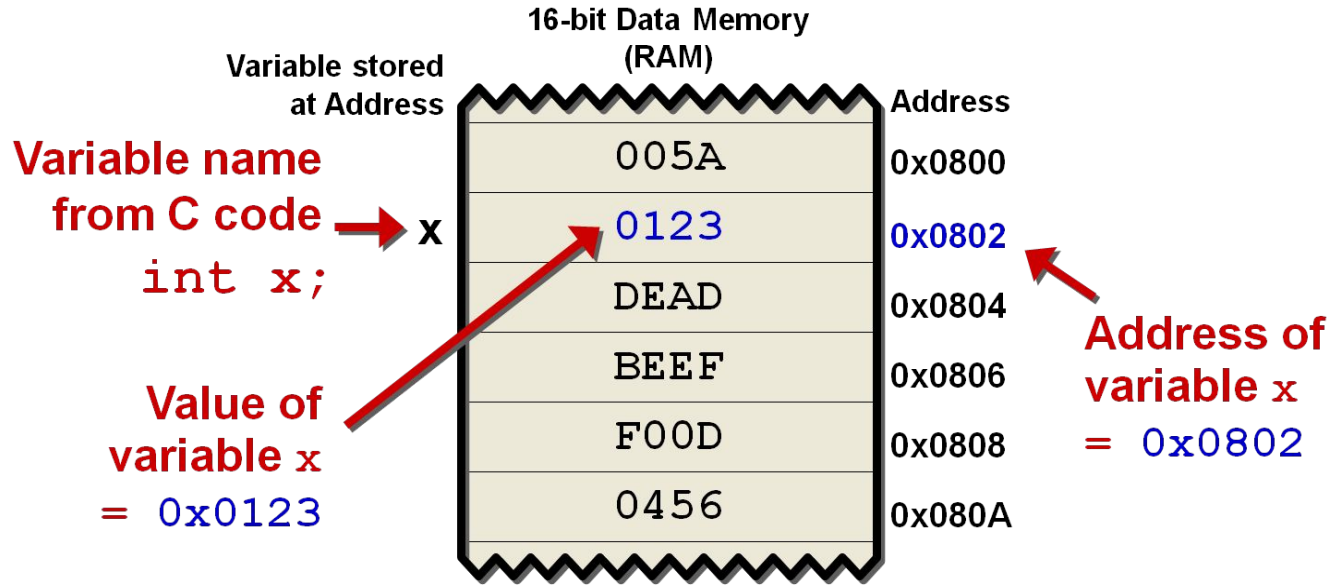
main.c: In function 'main':

main.c:20:10: error: assignment of read-only variable 'dato'

```
20 |     dato = 20;
    |         ^
```

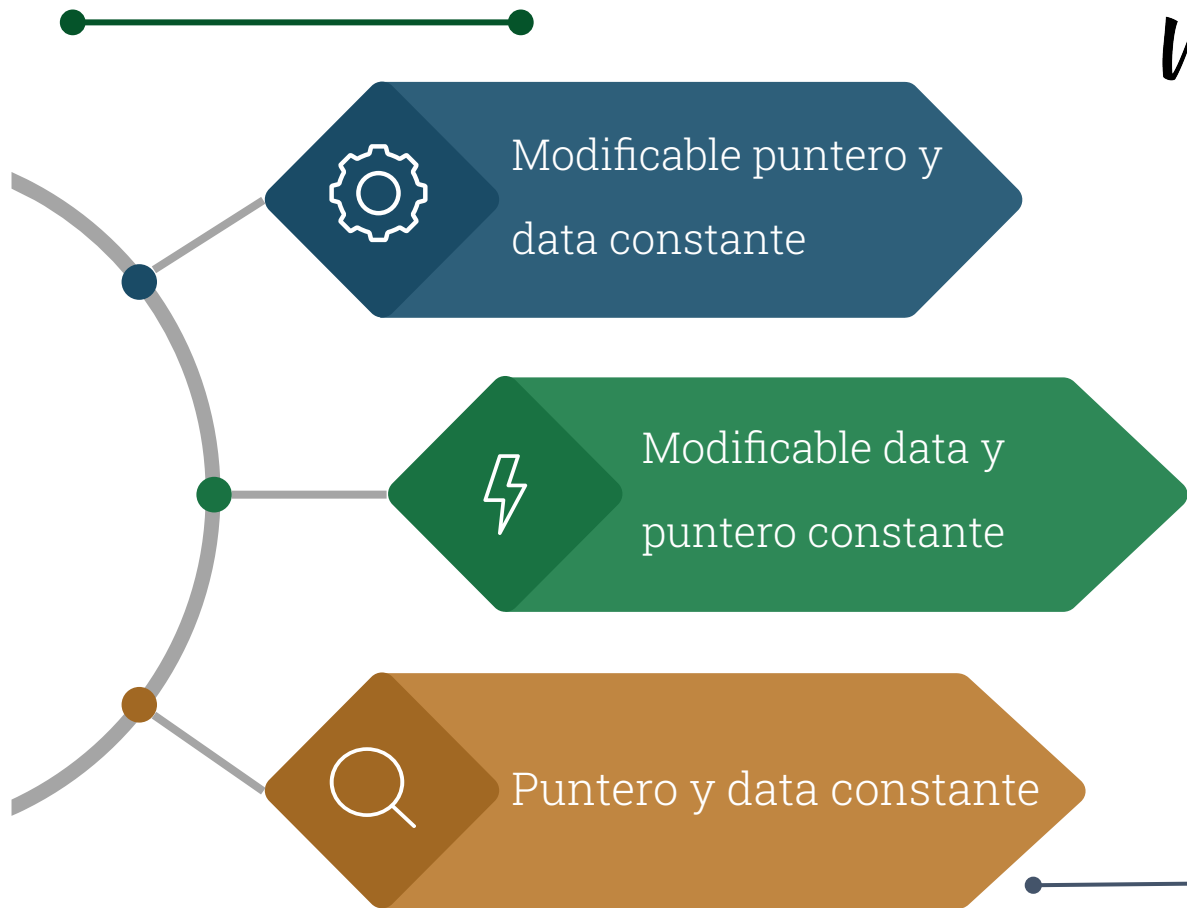


Se puede modificar su valor?



Con Punteros

Const



Modificable puntero y data constante

El puntero se puede modificar pero los
datos punteados no se pueden modificar

```
int const *pData
```

```
uint32_t const *registro_ClkEnableGPIOB = (uint32_t *)0x40020400;
```

¿Podría modificar la data?

Modificable puntero y data constante

El puntero se puede modificar pero los
datos punteados no se pueden modificar

```
int const *pData
```

```
uint32_t const *registro_ClkEnableGPIOB = (uint32_t *)0x40020400;
```

NO SE PUEDE MODIFICAR LA DATA

```
*registro_ClkEnableGPIOB |= (1<<1);
```


Modificable puntero y data constante

El puntero se puede modificar pero los
datos punteados no se pueden modificar

```
int const *pData
```

```
uint32_t const *registro_ClkEnableGPIOB = (uint32_t *)0x40020400;
```

SI PUEDES CAMBIAR REGISTRO

```
registro_ClkEnableGPIOB = (uint32_t *)0x40020400;
```

Modificable data y puntero constante

El dato se puede modificar pero el puntero es constante (no se pueden modificar).

```
int *const pData
```

```
uint32_t *const registro_ClkEnableGPIOB = (uint32_t *)0x40020400;
```

Se puede modificar la data

```
*registro_ClkEnableGPIOB |= (1<<1);
```

Data y puntero constante

Usada para lectura y retorno de
valor de un registro.

```
int const *const pData;
```

```
uint32_t const *const registro_ClkEnableGPIOB = (uint32_t *)0x40020400;
```

Sólo lectura del registro

Const

- Añade mayor seguridad a tu código.
- Mejora las restricciones para el acceso de funciones.

Veamos un ejemplo con Lectura

GPIO Clock

- Debemos habilitar el clock de cada GPIO que utilicemos.
- Se encuentra en el registro **RCC->AHB1ENR**

GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
rw	rw	rw	rw	rw	rw	rw	rw	rw

A-B-C-D-E-F-G-H-I

Wels

- RCC comienza en **0x4002 3800**.
Que llamaremos la dirección base.

- ### 6.3.10 RCC AHB1 peripheral clock register (RCC_AHB1ENR)

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

[illegible]

GPIOs Configuración



Vamos hacer la configuración para el pin 13
del GPIOC

GPIO

Dentro de **GPIOC** existen diferentes registros para configurar

0x4002 0C00 - 0x4002 0FFF	GPIOD
0x4002 0800 - 0x4002 0BFF	GPIOC
0x4002 0400 - 0x4002 07FF	GPIOB

MODER
OTYPER
OSPEEDR
PUPDR
IDR
ODR
BSRR
LCKR
AFRH
AFRL

GPIO

Dentro de **GPIOC** existen diferentes registros para configurar

0x4002 0C00 - 0x4002 0FFF	GPIOD
0x4002 0800 - 0x4002 0BFF	GPIOC
0x4002 0400 - 0x4002 07FF	GPIOB

MODER
OTYPER
OSPEEDR
PUPDR
IDR
ODR
BSRR
LCKR
AFRH
AFRL

Cada registro del periférico tiene un ancho de **32 bits**



MODER

Output

- Con el registro MODER seleccionamos entrada, salida, analógico, función alternativa

MODERy[1:0]: Port x configuration bits ($y = 0..15$)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

GPIOC -> MODER

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

2 bits por cada Pin

MODERy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

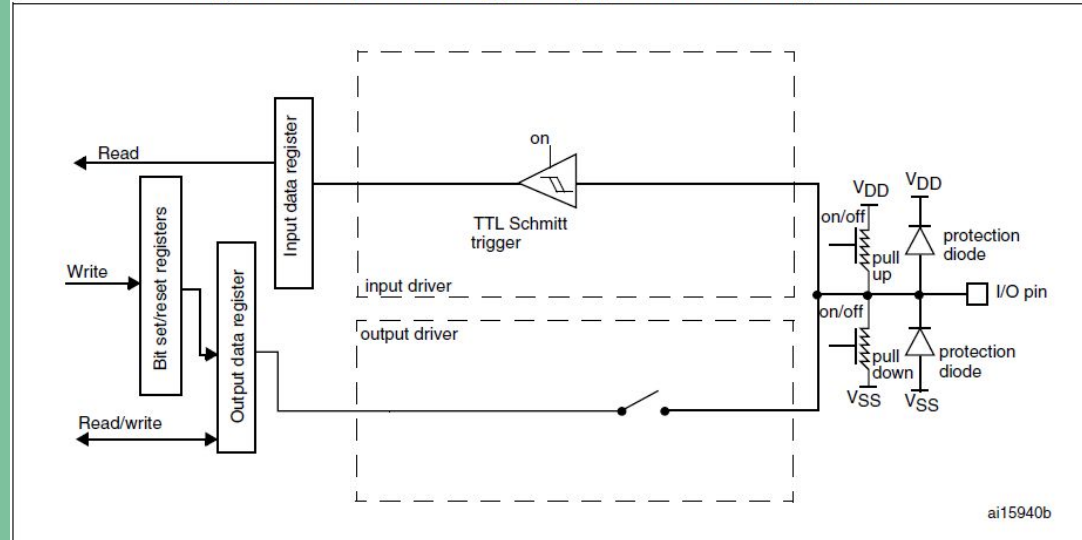
PB0	46	PB0
PB1	47	PB1
PB2	48	PB2
PB3	133	PB3
PB4	134	PB4
PB5	135	PB5
PB6	136	PB6
PB7	137	PB7
PB8	139	PB8
PB9	140	PB9
PB10	69	PB10
PB11	70	PB11
PB12	73	PB12
PB13	74	PB13
PB14	75	PB14
PB15	76	PB15

IDR

Lectura

- Registro de lectura del estado del pin.

Figure 18. Input floating/pull up/pull down configurations



GPIOC -> IDR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Cada bit del registro es para cada Pin.

Wels

- A la dirección base del GPIOC se le suma el address offset: **0x10**

- [illegible]

GPIOC -> IDR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

PB0	46	PB0
PB1	47	PB1
PB2	48	PB2
PB3	133	PB3
PB4	134	PB4
PB5	135	PB5
PB6	136	PB6
PB7	137	PB7
PB8	139	PB8
PB9	140	PB9
PB10	69	PB10
PB11	70	PB11
PB12	73	PB12
PB13	74	PB13
PB14	75	PB14
PB15	76	PB15

Calificadores

- Se utilizan para modificar la propiedad de una variable.
- Veremos dos palabras claves:

const

volatile



Optimización

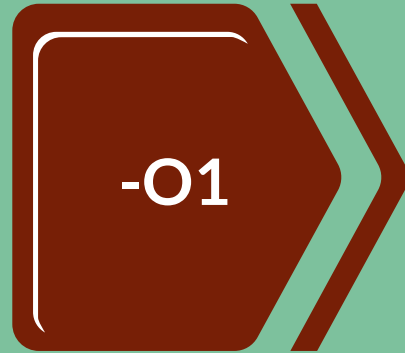
Optimización

Wels



- Cuando decimos **optimización** nos referimos a que el compilador reduce el código del programa.
- Puede reducir el número de instrucciones, el espacio de memoria.
- Por defecto siempre tu programa no se optimiza, tu debes habilitar la opción.

Optimización

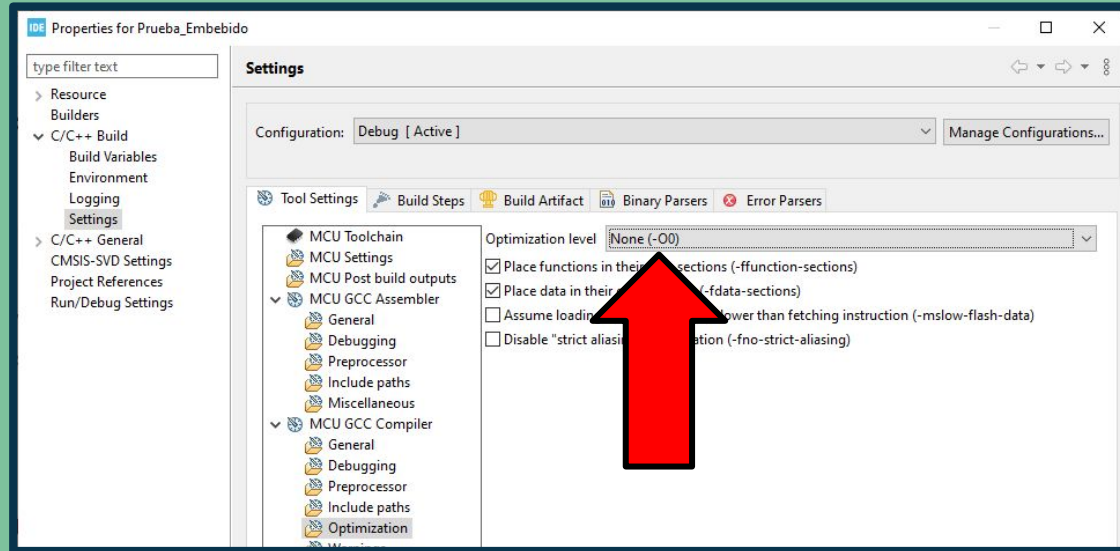


Niveles de optimización



-O0

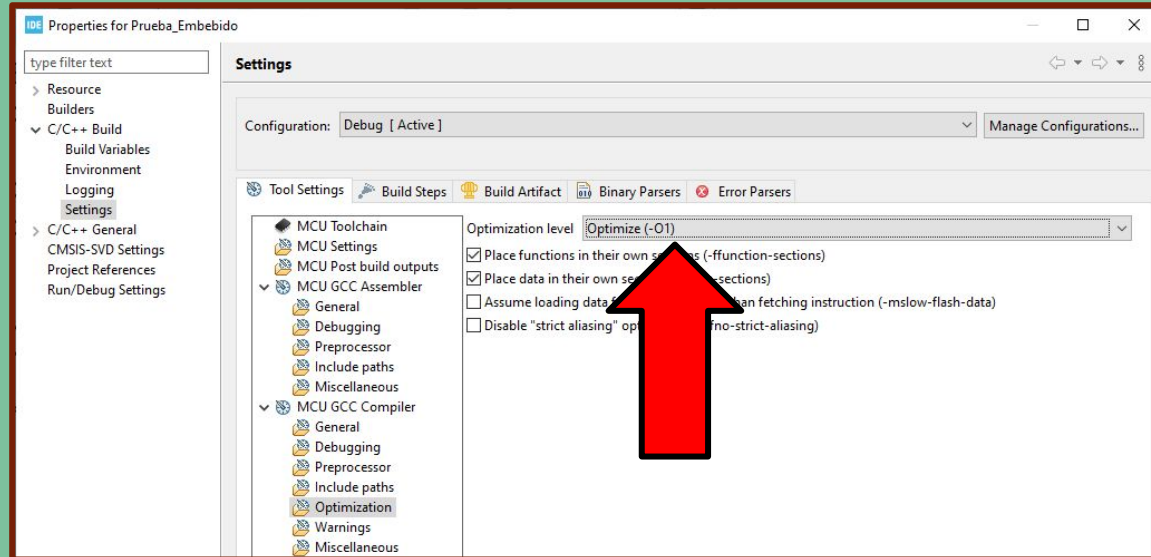
- No hay optimización.
- Su tiempo de compilación es el más rápido.
- Es más accesible de depurar y desarrollar.





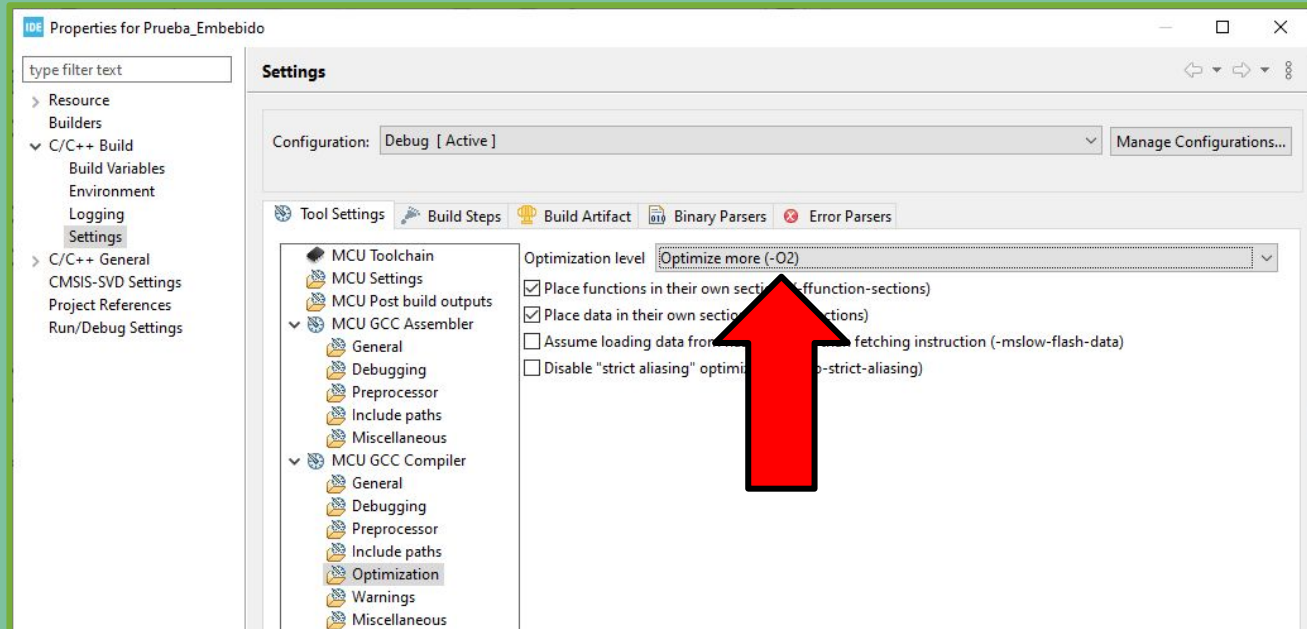
-O1

- Una optimización moderada que decrementa el tiempo de acceso a memoria y espacio de código.



**-O2**

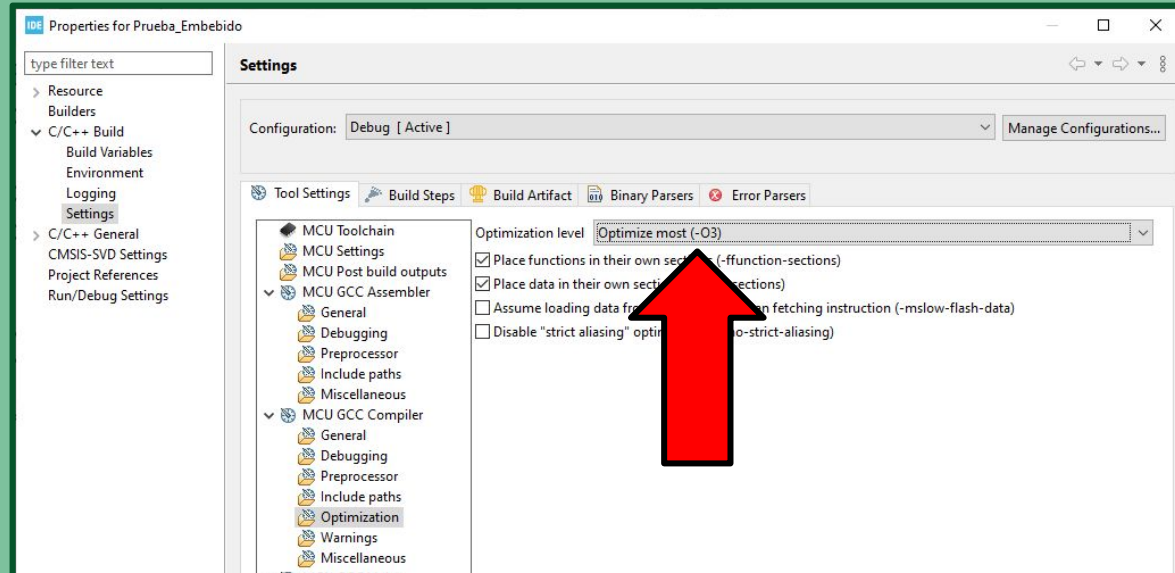
- Full optimización.
- Tiene un lento tiempo de compilación
- No es muy accesible la depuración





-O3

- Es más agresivo que -O2 y toma más tiempo de compilación.
- Puede causar bugs en el programa.
- No es muy accesible la depuración



Volatile

```
int volatile variable = 30;  
volatile float dato = 3.3;
```

Volatile es un tipo de calificador que indica al compilador de no optimice esa variable.

Le indica al compilador que la variable **cambiará** en cualquier tiempo sin el consentimiento del programador.

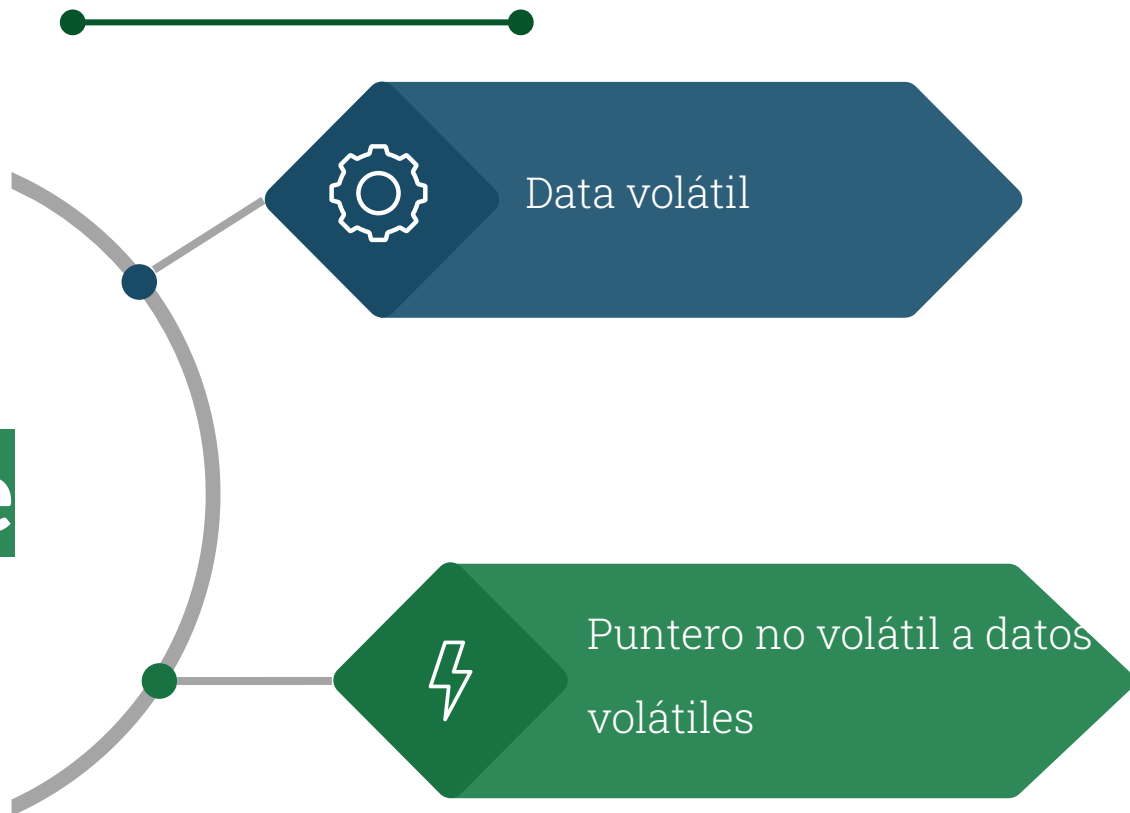
¿Cuándo usar Volatile?

Wels

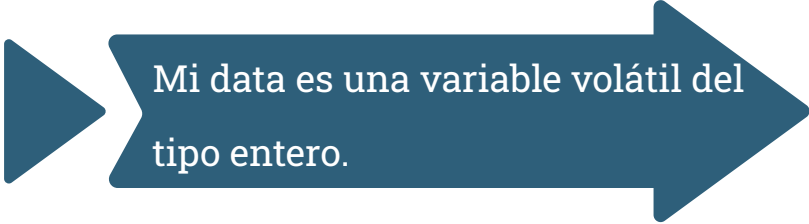


- Cuando se utiliza variables globales que cambia su valor entre el main y ISR code.
- Mapeo de memoria de registros del microcontrolador.

Volatile



Data volátil

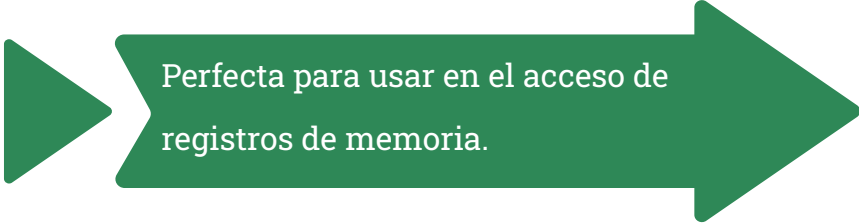


Mi data es una variable volátil del tipo entero.

```
int volatile pData
```

```
uint32_t volatile variable1;
```

Puntero no volátil a datos volátiles



Perfecta para usar en el acceso de registros de memoria.

```
int volatile *pData
```

```
uint32_t volatile *p_ClkEnable = (uint32_t *)0x40023830;
```

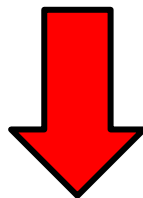
CONST y VOLATILE

Wels

- Se puede usar const y volatile calificadores en una variable.

“volatile *const”

```
uint32_t *const p_ClkEnable = (uint32_t *)0x40023830;
```



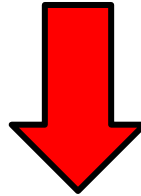
```
uint32_t volatile *const p_ClkEnable = (uint32_t *)0x40023830;
```

CONST y VOLATILE

Wels

- Se puede usar const y volatile calificadores en una variable.

“const volatile *const”



```
uint32_t const volatile *const p_RegIdrGPIOC = (uint32_t *)0x40020810;
```

Que sólo es un registro o memoria de lectura que puede cambiar en cualquier momento.



Gracias

@welstheory

hola@welstheory.com

+51 918 899 684

