EEVblog Electronics Community Forum

A Free & Open Forum For Electronics Enthusiasts & Professionals

« previous next »

Pages: [**1**] 2 Next All **Go Down**

PRINT SEARCH

Author **Topic: Macro for efficient stm32 gpio handling (Read 2352 times)**

0 Members and 1 Guest are viewing this topic.

DavidAlfa
Super Contributor
Posts: 2855
Country:

Macro for efficient stm32 gpio handling

« **on:** April 28, 2021, 09:21:28 am »

I must admit HAL it's ok-ish when starting with stm32, otherwise it'll be overwhelming at first.
But after a while you start seeing the issues and limitations everywhere, and start fiddling with registers.

The whole HAL is so badly done! For a simple IO Pin status change, it makes a code jump (adding stack overhead, and likely missing the prefetch cache), an if/else, and returns. It's a total waste of cpu time.
However using the HAL defines is very convenient. So today I made some GPIO macros that works great, compatible with the HAL format.
Most operations use 3 instructions, and no jumps, so it's very efficient..
It's done for the F4 series, others might require some changes, but they're mostly the same.

The target was to not hardcode anything. So any change in CubeMX works right away.
But I found a big issue: HAL doesn't declare the pin numbers anywhere. They're hardcoded as Pin2=0x2, Pin8=0x100, Pin15=0x8000...
Thus, you're not going to get the pin number for "LED_Pin" anywhere for making a bitmask.
I had to do a some bit hacking to overcome that. Crappy HAL!

It might look huge, but since it's a macro, all the calculations are done by the preprocessor at compile time, and the real code will use a constant.

Code: (MACRO CODE) [Select]
```
 *      Author: David
 */

#ifndef INC_GPIO_H_
#define INC_GPIO_H_


/*
*    Returns bit position.
*    0b1->0
*    0b100000->5
*/
#define __get_GPIO_Pos(x)             (15*(1&&(x&(1<<15)))  + \
                                      14*(1&&(x&(1<<14)))   + \
                                      13*(1&&(x&(1<<13)))   + \
                                      12*(1&&(x&(1<<12)))   + \
```

Code: (USAGE) [Select]
```

   #define LED_Pin GPIO_PIN_13
   #define LED_GPIO_Port GPIOC

   volatile uint32_t data = 0;

   setPinMode(LED_GPIO_Port, LED_Pin, MODE_INPUT);
   setPinMode(LED_GPIO_Port, LED_Pin, MODE_OUTPUT);
   setPinMode(LED_GPIO_Port, LED_Pin, MODE_AF);
   setPinMode(LED_GPIO_Port, LED_Pin, MODE_ANALOG);

   setPinSpeed(LED_GPIO_Port, LED_Pin, GPIO_SPEED_FREQ_LOW);
   setPinSpeed(LED_GPIO_Port, LED_Pin, GPIO_SPEED_FREQ_MEDIUM);
   setPinSpeed(LED_GPIO_Port, LED_Pin, GPIO_SPEED_FREQ_HIGH);
   setPinSpeed(LED_GPIO_Port, LED_Pin, GPIO_SPEED_FREQ_VERY_HIGH);
```

Code: (ASSEMBLY, NO OPTIMIZATIONS) [Select]
```

129          setPinMode(LED_GPIO_Port, LED_Pin, MODE_INPUT);
08000c66:    ldr     r3, [pc, #384]  ; (0x8000de8 <main+440>)
08000c68:    ldr     r3, [r3, #0]
08000c6a:    ldr     r2, [pc, #380]  ; (0x8000de8 <main+440>)
08000c6c:    bic.w   r3, r3, #201326592      ; 0xc000000
08000c70:    str     r3, [r2, #0]

130          setPinMode(LED_GPIO_Port, LED_Pin, MODE_OUTPUT);
08000c72:    ldr     r3, [pc, #372]  ; (0x8000de8 <main+440>)
08000c74:    ldr     r3, [r3, #0]
08000c76:    bic.w   r3, r3, #201326592      ; 0xc000000
08000c7a:    ldr     r2, [pc, #364]  ; (0x8000de8 <main+440>)
08000c7c:    orr.w   r3, r3, #67108864       ; 0x4000000
08000c80:    str     r3, [r2, #0]
```
« Last Edit: April 28, 2021, 11:29:55 am by DavidAlfa »

Logged

Hantek DSO2x1x **Drive** **FAQ** **DON'T BUY HANTEK! (Aka HALF-MADE)**
Stm32 Soldering FW **Forum** **Github** **Donate**

JOEBOBSICLE
Regular Contributor
Posts: 62
Country: 🇬🇧

Re: Macro for efficient stm32 gpio handling
« **Reply #1 on:** April 28, 2021, 11:46:14 am »

It's not that inefficient to use the HAL if you use link time optimization which may inline small functions. Functions are cleaner than macros

« Last Edit: April 28, 2021, 11:51:38 am by JOEBOBSICLE »

Logged

ajb
Super Contributor
Posts: 2276
Country: 🇺🇸

Re: Macro for efficient stm32 gpio handling
« **Reply #2 on:** April 28, 2021, 03:42:44 pm »

Quote from: DavidAlfa on April 28, 2021, 09:21:28 am

> But I found a big issue: HAL doesn't declare the pin numbers anywhere. They're hardcoded as Pin2=0x2, Pin8=0x100, Pin15=0x8000...
> Thus, you're not going to get the pin number for "LED_Pin" anywhere for making a bitmask.

I'm not sure I understand this part. Where is LED_Pin defined? Is that a HAL thing for the dev boards that have LEDs on particular pins, or does HAL generate that from what you tell it your pin allocations are? Or is it defined by you, as your example shows, in which case why wouldn't you just define LED_Pin as "13" instead of as "GPIO_PIN_13"? I guess other parts of HAL depend on the pin being defined that way maybe, if you're using other library functions against that pin? Switching from defining LED_Pin as a mask to defining it as an offset would allow you to skip the `__get_GPIO_Pos` macro and replace that `__expand_16to32` macro chain with `(0x03<<(2*x))`

I don't use HAL so don't care about compatibility, but it's possible to make usage even simpler by adding an extra macro layer:

Code: [Select]
```
#define IO_OUT_SET(...)              IO_OUT_SET_SUB(__VA_ARGS__)
#define IO_OUT_SET_SUB(port, pin)       GPIO##port->BSRR = (1<<pin)
```

The extra variadic macro allows multiple arguments to be passed in as a single macro and expands them as the macros are resolved, so you can do this:

Code: [Select]
```
#define LED_PIN     A,6

IO_OUT_SET(LED_PIN);
```

Without the extra macro layer the preprocessor will complain about the wrong number of arguments. Each time I work on a new part family I make a set of macros in this style for it, so everything's consistent, and usually the first code I write in a new project is a list of defines for all of the IO pins transcribed from the schematic.

Quote from: JOEBOBSICLE on April 28, 2021, 11:46:14 am

> It's not that inefficient to use the HAL if you use link time optimization which may inline small functions. Functions are cleaner than macros

That second bit is a rather broad statement. Macros can certainly get very ugly when used improperly, but then so can functions (just look at HAL). A situation like this where you have one to two lines that are very simple presents very little risk of macros getting out of hand. Functions *may* get optimized down to equivalent instructions as the macros, and often they will, but it's not guaranteed. Optimization of functions depends not just on the optimization settings but also on the type and qualification of any arguments. With macros, and defined constants as their arguments, you end up giving the optimizer a set of statements that is unambiguously constant and therefore more likely to get optimized in all cases. Obviously that doesn't mean that EVERYTHING should be macros instead of functions, but for stuff like this where you will use them everywhere across a project and they're so simple to write once, I think macros make a lot of sense.   And even if the macros aren't BETTER than function calls, I don't see how they're especially cleaner in situations like this.

« Last Edit: April 28, 2021, 03:44:21 pm by ajb »

Logged

The following users thanked this post: DavidAlfa

newbrain

Super Contributor

Posts: 1439

Country: 🇸🇪

Re: Macro for efficient stm32 gpio handling
« **Reply #3 on:** April 28, 2021, 03:52:30 pm »

As another alternative, there's the LL library, which mostly consist of static inline functions.
They are very similar to the macros defined above, and for GPIO an init function is also provided, though the HAL one can be used too.

I have yet to see a case where they were not inlined, from -Og onwards.

Logged

Nandemo wa shiranai wa yo, shitteru koto dake.

DavidAlfa

Super Contributor

Posts: 2855

Re: Macro for efficient stm32 gpio handling
« **Reply #4 on:** April 28, 2021, 04:19:25 pm »

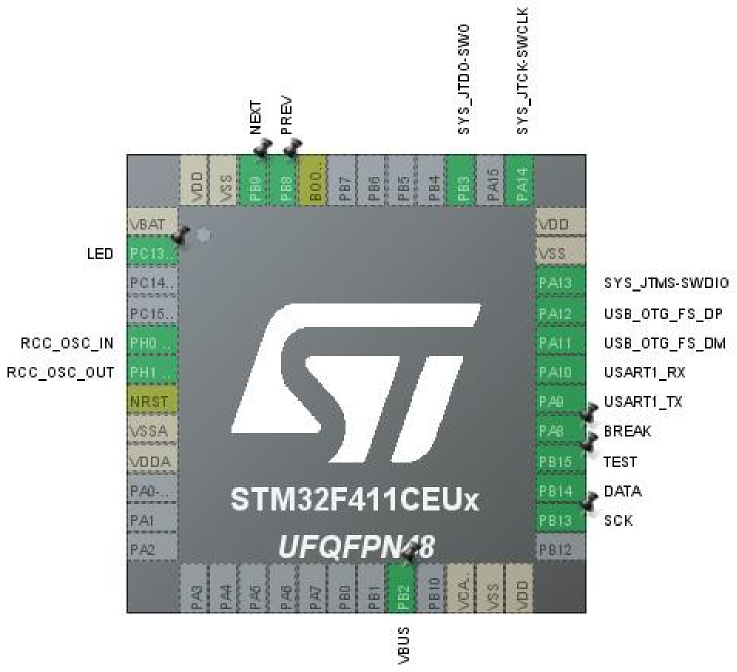
Quote from: ajb on April 28, 2021, 03:42:44 pm

Country: 🇪🇸

> I'm not sure I understand this part. Where is LED_Pin defined? Is that a HAL thing for the dev boards that have LEDs on particular pins, or does HAL generate that from what you tell it your pin allocations are? Or is it defined by you, as your example shows, in which case why wouldn't you just define LED_Pin as "13" instead of as "GPIO_PIN_13"? I guess other parts of HAL depend on the pin being defined that way maybe, if you're using other library functions against that pin? Switching from defining LED_Pin as a mask to defining it as an offset would allow you to skip the `__get_GPIO_Pos` macro and replace that `__expand_16to32` macro chain with `(0x03<<(2*x))`

As I said, the main point is to keep CubeMX for quick peripheral inititalization. It really saves a lot of time.
If I assigned LED to PC3, but I later change it to PA0, I'd like to avoid doing anything else to apply the changes.

About the pins, yes, it's weird. Look at this example:



It will generate this:

Code: (main.h) [Select]

```
#define LED_Pin GPIO_PIN_13
#define LED_GPIO_Port GPIOC
#define VBUS_Pin GPIO_PIN_2
#define VBUS_GPIO_Port GPIOB
#define SCK_Pin GPIO_PIN_13
#define SCK_GPIO_Port GPIOB
#define DATA_Pin GPIO_PIN_14
#define DATA_GPIO_Port GPIOB
#define TEST_Pin GPIO_PIN_15
#define TEST_GPIO_Port GPIOB
#define BREAK_Pin GPIO_PIN_8
#define BREAK_GPIO_Port GPIOA
#define PREV_Pin GPIO_PIN_8
#define PREV_GPIO_Port GPIOB
#define NEXT_Pin GPIO_PIN_9
#define NEXT_GPIO_Port GPIOB
```

And the GPIO declarations (GPIO_PIN_x) are:

Code: (stm32f4xx_hal_gpio.h) [Select]

```
#define GPIO_PIN_0                 ((uint16_t)0x0001)  /* Pin 0 selected    */
#define GPIO_PIN_1                 ((uint16_t)0x0002)  /* Pin 1 selected    */
#define GPIO_PIN_2                 ((uint16_t)0x0004)  /* Pin 2 selected    */
#define GPIO_PIN_3                 ((uint16_t)0x0008)  /* Pin 3 selected    */
#define GPIO_PIN_4                 ((uint16_t)0x0010)  /* Pin 4 selected    */
#define GPIO_PIN_5                 ((uint16_t)0x0020)  /* Pin 5 selected    */
#define GPIO_PIN_6                 ((uint16_t)0x0040)  /* Pin 6 selected    */
#define GPIO_PIN_7                 ((uint16_t)0x0080)  /* Pin 7 selected    */
#define GPIO_PIN_8                 ((uint16_t)0x0100)  /* Pin 8 selected    */
#define GPIO_PIN_9                 ((uint16_t)0x0200)  /* Pin 9 selected    */
#define GPIO_PIN_10                ((uint16_t)0x0400)  /* Pin 10 selected   */
#define GPIO_PIN_11                ((uint16_t)0x0800)  /* Pin 11 selected   */
#define GPIO_PIN_12                ((uint16_t)0x1000)  /* Pin 12 selected   */
#define GPIO_PIN_13                ((uint16_t)0x2000)  /* Pin 13 selected   */
```

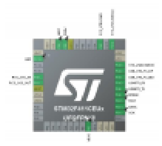
```
#define GPIO_PIN_14                ((uint16_t)0x4000)  /* Pin 14 selected   */
#define GPIO_PIN_15                ((uint16_t)0x8000)  /* Pin 15 selected   */
```

So yes, you can't get the pin index anywhere. I could do a switch, but I wanted to avoid using CPU power when a macro can do it at compilation time.

Quote from: ajb on April 28, 2021, 03:42:44 pm

> **Code:** [Select]
> ```
> #define IO_OUT_SET(...)             IO_OUT_SET_SUB(__VA_ARGS__)
> #define IO_OUT_SET_SUB(port, pin)   GPIO##port->BSRR = (1<<pin)
> ```

This looks very interesting!I'm not a professional coder, learning these little things is great.
But again that will require to change the code if the pin is modified in CubeMX.



cubemx.jpg (48.58 kB, 500x457 - viewed 1062 times.)

« Last Edit: April 28, 2021, 04:42:14 pm by DavidAlfa » Logged

Hantek DSO2x1x **Drive** **FAQ** **DON'T BUY HANTEK! (Aka HALF-MADE)**
Stm32 Soldering FW **Forum** **Github** **Donate**

ataradov
Super Contributor
Posts: 9239
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #5 on:** April 28, 2021, 04:25:00 pm »

Here is what I use for this https://github.com/ataradov/mcu-starter-projects/blob/master/stm32g071/hal_gpio.h

The use is similar, but you get nice type checked functions for each pin, and you don't need to pass port/pin parameters on every call. First define the pins once:

Code: [Select]
```
HAL_GPIO_PIN(LED,      D, 0)
HAL_GPIO_PIN(BUTTON,   B, 0)
```

Then access them though dedicated functions with readable names:
Code: [Select]
```
HAL_GPIO_LED_out();
 HAL_GPIO_LED_set();

HAL_GPIO_BUTTON_in();
HAL_GPIO_BUTTON_pullup();
 if (HAL_GPIO_BUTTON_read()) ....
```

Logged

Alex

The following users thanked this post: laneboysrc, agehall, DavidAlfa

DavidAlfa
Super Contributor
Posts: 2855
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #6 on:** April 28, 2021, 04:31:34 pm »

Looks good! Why nobody taking all these contributions and doing a new library? It would be great to get rid of that Cube crap once for all

« Last Edit: April 28, 2021, 04:43:06 pm by DavidAlfa » Logged

Hantek DSO2x1x **Drive** **FAQ** **DON'T BUY HANTEK! (Aka HALF-MADE)**
Stm32 Soldering FW **Forum** **Github** **Donate**

ataradov

Re: Macro for efficient stm32 gpio handling

Super Contributor

Posts: 9239
Country:

« **Reply #7 on:** April 28, 2021, 04:45:13 pm »

Because a universal library that works for everyone would be bloated again. Everyone who wants things simple just made their own file, others just use Cube. I personally would not switch to any other library when I have one header file that does everything I need.

Logged

Alex

The following users thanked this post: agehall

DavidAlfa
Super Contributor

Posts: 2855
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #8 on:** April 28, 2021, 04:52:49 pm »

The HAL wouldn't be so bad if the guys did things right.

For example, HAL_GPIO_Toggle:
Code: [Select]
```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  uint32_t odr;
  odr = GPIOx->ODR;
  GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
}
```

Instead:
Code: [Select]
```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  GPIOx->ODR^=GPIO_Pin;
}
```

Since the peripheral registers are volatile, the compiler won't optimize these too much.
Really, why so much mess? Makes me thing what people are behind it. Newbies?
Also they could declare it as inline? Eve It will use a little bit more of flash, but also I measured it 3x faster

« *Last Edit: April 28, 2021, 05:30:35 pm by DavidAlfa* »

Logged

Hantek DSO2x1x	Drive	FAQ	**DON'T BUY HANTEK! (Aka HALF-MADE)**
Stm32 Soldering FW	Forum	Github	Donate

SiliconWizard
Super Contributor

Posts: 10119
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #9 on:** April 28, 2021, 04:58:47 pm »

Quote from: newbrain on April 28, 2021, 03:52:30 pm

> As another alternative, there's the LL library, which mostly consist of static inline functions.
> They are very similar to the macros defined above, and for GPIO an init function is also provided, though the HAL one can be used too.
>
> I have yet to see a case where they were not inlined, from -Og onwards.

Yep.

And that said, use what's best for your requirements. Who cares if setting an IO takes a couple more cycles that it would with fully optimized assembly code, unless you actually NEED to save those extra cycles for performance reasons. I'm against execssive bloat, but also against excessive optimization when it's not actually needed. Sometimes it is, but most of the time, if you really need ultra-fast GPIO toggling, then you're probably going to have to use a dedicated peripheral anyway (such as output compare, serial or parallel bus, or whatever else...) rather than bit-banging stuff. Those MCUs are complex stuff and are pretty fast. You can't think of them as you would have when dealing with crippled 8-bit MCUs running at a couple MHz.

So yeah, GPIO-related functions in the HAL are definitely not what causes issues in most cases. There are HAL functions that are easy to use, but definitely NOT efficient though. Functions handling peripheral block access via DMA in just one function, for instance (ADC, DAC, SPI, ...). As they often include the DMA setup code, they can take hundreds of cycles and are good examples of what can be hand optimized.

And as ataradov said, if you think you can replace one general-purpose, one-size-fits-all library with

another, better one, you're probably deluded.
Just use existing libraries when they fit your requirements, and hand-written code when they do not.

Logged

The following users thanked this post: lucazader

ataradov

Super Contributor

Posts: 9239
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #10 on:** April 28, 2021, 05:08:33 pm »

Quote from: DavidAlfa on April 28, 2021, 04:52:49 pm

> Really, why so much mess? Makes me thing what people are behind it. Newbies?

The first code is actually correct if you ever call that from an interrupt. The second code would fail and set the wrong bits if interrupt and main code coincide. Not newbies, quite the opposite.

I use ^ too, but that't the responsibility I'm taking. And I may forget about it and run into bugs that I have to debug. But at least, if it is my bug, I'm the one to blame for time spent debugging.

And that's another argument against universal libraries. In my code I make allowances for my general coding and architecture style. If you were to make a universal code that should just work for everyone, things will get much more complicated.

Also, with a simple code like this that manipulates single pins, you are assuming that you won't need to modify multiple pins. Or you need to make an API for that too. Again, in my case I know that modifications to multiple pins are rarely used by me. Usually it happens in very tightly optimized sections, where completely custom code is justified anyway. So I never bother to implement anything generic for manipulating multiple pins.

« *Last Edit: April 28, 2021, 05:21:37 pm by ataradov* »

Logged

Alex

The following users thanked this post: newbrain

SiliconWizard

Super Contributor

Posts: 10119
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #11 on:** April 28, 2021, 05:23:27 pm »

Quote from: ataradov on April 28, 2021, 05:08:33 pm

> **Quote from: DavidAlfa on April 28, 2021, 04:52:49 pm**
>
> > Really, why so much mess? Makes me thing what people are behind it. Newbies?
>
> The first code is actually correct if you ever call that from an interrupt. The second code would fail and set the wrong bits if interrupt and main code coincide. Not newbies, quite the opposite.

Absolutely. The two versions absolutely DO NOT do the same thing. The ST versions uses the BSRR register to ultimately toggle the IO. This guarantees atomicity. Using the ODR register only instead does not guarantee atomicity.

And that said, compare the assembly code generated for both versions with compiler optimizations enabled. The second one is likely not that much more efficient, and doesn't guarantee atomicity.
You can have a quick look there:
https://electronics.stackexchange.com/questions/392954/understanding-atomic-writes-to-gpio-on-different-arm-cortex-ms

Reminds me that on the PIC32 MCUs, there are SET, RESET and INVERT registers (not sure about the exact names anymore), and thus you can actually toggle IOs in just one instruction writing to the INVERT register. Convenient. I don't think the STM32 MCUs have such "invert" registers (I guess STM would have used them if so), but they are cool.

With that said, as I wrote above, if you really need to "toggle" IOs, and toggle them with strict timing requirements, then it smells like a peripheral such as output compare would definfitely be called for instead of trying to big-bang it.

Logged

ttt

Regular Contributor

Re: Macro for efficient stm32 gpio handling
« **Reply #12 on:** April 28, 2021, 05:28:08 pm »

Quote from: DavidAlfa on April 28, 2021, 04:52:49 pm

Posts: 82
Country:

> The HAL wouldn't be so bad if the guys did things right.
>
> For example, HAL_GPIO_Toggle:
> **Code:** [Select]
> ```
> void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
> {
>   uint32_t odr;
>   odr = GPIOx->ODR;
>   GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
> }
> ```
>
> Instead:
> **Code:** [Select]
> ```
> void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
> {
>   GPIOx->ODR^=GPIO_Pin;
> }
> ```
>
> Since the peripheral registers are volatile, the compiler won't optimize these too much.
> Really, why so much mess? Makes me thing what people are behind it. Newbies?
> Also they could declare it as inline? It will use a little bit more of flash, but also I measured it 3x faster

Your version with using ODR only is not interrupt save, right? If an interrupt modifies one of the other GPIOs on that port you can end up with a race. That's why the use of BSRR is preferred to be the default. Admittedly an edge case since we are talking about a 2 cycle chance of that happening but still possible. And that HAS happened to me in deployed code. Super hard to track down.

As for inlining, -flto makes this a non-issue for practical applications. I take my type safety over using macros any day.

Logged

ataradov
Super Contributor

Posts: 9239
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #13 on:** April 28, 2021, 05:32:31 pm »

Quote from: ttt on April 28, 2021, 05:28:08 pm

> As for inlining, -flto makes this a non-issue for practical applications. I take my type safety over using macros any day.

My macros are type safe, since they generate a bunch of dedicated functions. Those things are marked as inline, but yes, LTO would further hammer that down.

There is a common confusion between simplicity of the code and it being optimal. Optimal code is more often than not is messier.

In my case I go so simplicity of reading and maintenance rather than ultimate optimization. I let the compiler optimize as much as it can, and if that it not enough, I do it by hand. It is rarely needed.

Code that is simple to read is often more optimal than some stuff with a ton of abstraction, but that is not given.

Logged

Alex

The following users thanked this post: newbrain, lucazader

agehall
Frequent Contributor

Posts: 354
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #14 on:** April 28, 2021, 05:49:36 pm »

One also has to remember that the HAL is an abstraction that needs to work across multiple devices - it has to account for differences in a way that doesn't break the APIs it presents. This can lead to some "over complicated" architecture/design in some places.

But from what I've seen of it, it isn't all that bad. Just complicated at times.

Back in the days, I used to write super optimized assembly code to convert data into assembly files that could be compiled into the software I was working on. Completely pointless as that was a one-off thing but I wanted everything optimized 100%. These days, I care much more about code readability and simplicity for the *reader* rather than optimizing everything. I trust the C compiler to do that well enough so that I don't have to care.

And one thing is certain, compared to the steaming pile of crap that is the Arduino framework and related libraries, the STM32 HAL is a beautiful piece of art.

Logged

ttt

Regular Contributor

Posts: 82

Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #15 on:** April 28, 2021, 05:55:53 pm »

Quote from: ataradov on April 28, 2021, 05:32:31 pm

> Code that is simple to read is often more optimal than some stuff with a ton of abstraction, but that is not given.

In principle everybody can agree on that. A secondary goal should also be to make the code more easy to debug. Extensive use of macros and abstractions can make it vastly more difficult to debug code in the debugger. But then I heard there are still are pure printfers out there.

Logged

The following users thanked this post: lucazader

lucazader

Regular Contributor

Posts: 218

Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #16 on:** April 28, 2021, 05:56:53 pm »

I think an important point that has been bought up by a few users here is try not to over optimize things from the beginning, and then only start optimizing if you run into performance issues.

If you want a lighter weight library than the HAL, then using the LL library is probably the easiest way forward.

@ataradov your macro to generate the inline functions is quite nice, I hadn't thought to do this. Thanks for sharing.

Logged

DavidAlfa

Super Contributor

Posts: 2855

Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #17 on:** April 28, 2021, 06:12:23 pm »

Quote from: ttt on April 28, 2021, 05:55:53 pm

> In principle everybody can agree on that. A secondary goal should also be to make the code more easy to debug. Extensive use of macros and abstractions can make it vastly more difficult to debug code in the debugger. But then I heard there are still are pure printfers out there.

Im all ears! What are other possible options for MCUs like STM32? Else thant print or SWV trace? SWV will allow tracing of only 4 variables.

Logged

Hantek DSO2x1x	Drive	FAQ	**DON'T BUY HANTEK! (Aka HALF-MADE)**
Stm32 Soldering FW	Forum	Github	Donate

ataradov

Super Contributor

Posts: 9239

Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #18 on:** April 28, 2021, 06:25:06 pm »

Quote from: DavidAlfa on April 28, 2021, 06:12:23 pm

> Im all ears! What are other possible options for MCUs like STM32?

An actual debugger.

Although I generally don't use debuggers. Somehow I can figure things out with printfs and GPIO toggling faster than with a debugger. But I also have a ton of helpful debug code snippets that I have accumulated over the years.

« Last Edit: April 28, 2021, 06:27:07 pm by ataradov »

Logged

Alex

DavidAlfa

Super Contributor

Posts: 2855

Re: Macro for efficient stm32 gpio handling
« **Reply #19 on:** April 28, 2021, 06:32:08 pm »

Well, that's obvious . I was expecteting some magic! 😁
I've recently seen people claiming that debugging by watching variables, breakpoints and stepping the

Country:

code is something from the past.
So I was curious, I don't know other way.
printf is always a great thing to avoid using 876 breakpoints and opening 51 files!
If certain conditions are supposed to happen, then just print them has they come in.

« Last Edit: April 28, 2021, 06:58:23 pm by DavidAlfa »

Logged

Hantek DSO2x1x **Drive** **FAQ** **DON'T BUY HANTEK! (Aka HALF-MADE)**
Stm32 Soldering FW **Forum** **Github** **Donate**

ataradov
Super Contributor

Posts: 9239
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #20 on:** April 28, 2021, 06:39:31 pm »

There are advantages and disadvantages for both. printfs affect timing and your serial interface may not even have enough bandwidth to get all the information out. Breakpoints and singe stepping has its own gotchas. One of the most common ones is watching the peripheral registers while debugging. If those registers have clear on read flags, then debugger will clear them and the application will not see them.

You need to get comfortable using all the tools available when appropriate. That's part of the deal. There is no one universal tool for everything.

Logged

Alex

DavidAlfa
Super Contributor

Posts: 2855
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #21 on:** April 28, 2021, 07:02:04 pm »

Yeah, but for example, with stm32 you redirect the output to SWO, that's pretty fast (2MHz, or more with better debuggers).
I agree that it will slow down the code a bit but nothing serious if you limit the output instead printing books.
But when something just fails without easy spot, you need to stop and check every register, and for the peripherals, if you're lucky you can use the DBGMCU bits.
But for others like DMA, SPI, you're screwed. I had a lot of trouble with these as I couldn't stop the hardware, the interrupts kepts stacking while halted, fooling myself into thinking there were more bugs .

BTW, Taradov sound funny in spanish, "Tarado" means faulty, crazy, mad, freaked put.

« Last Edit: April 28, 2021, 07:07:44 pm by DavidAlfa »

Logged

Hantek DSO2x1x **Drive** **FAQ** **DON'T BUY HANTEK! (Aka HALF-MADE)**
Stm32 Soldering FW **Forum** **Github** **Donate**

JOEBOBSICLE
Regular Contributor

Posts: 62
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #22 on:** April 28, 2021, 07:02:36 pm »

My point about link time optimization is that I see a lot of developers who don't know about that or understand the tool chain completely.

Writing abstract code in 2021 is fine. Writing macro filled code without a HAL will cause you a whole crap load of problems when your MCU goes out of stock or is obsoleted.

My advice is to never modify peripheral registers in the same file as your business logic.

« Last Edit: April 28, 2021, 07:05:58 pm by JOEBOBSICLE »

Logged

DavidAlfa
Super Contributor

Posts: 2855
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #23 on:** April 28, 2021, 07:10:08 pm »

You would need a slightly different version for every family. Not too hard with a few defines.
GPIO is very similar on all, that's a 10 minute job for every family.

I understand yout thinking from a professional point of view.
You want to develop fast, and avoid losing days in debugging a strange bug without knowing if it's your library or the code itself.
Anyways, HAL itself is not reliable with its bugs waiting to happen under very specific conditions.
I hope there are no HAL-programmed STM32s in airplanes!. Oh, ask Boing for their 737s... Who knows!

« *Last Edit: April 28, 2021, 07:17:56 pm by DavidAlfa* »

Logged

Hantek DSO2x1x **Drive** **FAQ** **DON'T BUY HANTEK! (Aka HALF-MADE)**
Stm32 Soldering FW **Forum** **Github** **Donate**

SiliconWizard
Super Contributor

Posts: 10119
Country:

Re: Macro for efficient stm32 gpio handling
« **Reply #24 on:** April 28, 2021, 07:32:09 pm »

Quote from: ataradov on April 28, 2021, 06:25:06 pm

> **Quote from: DavidAlfa on April 28, 2021, 06:12:23 pm**
>
> Im all ears! What are other possible options for MCUs like STM32?
>
> An actual debugger.
>
> Although I generally don't use debuggers. Somehow I can figure things out with printfs and GPIO toggling faster than with a debugger. But I also have a ton of helpful debug code snippets that I have accumulated over the years.

I practically never use debuggers on embedded targets either. My experience is similar; they actually tend to make you waste a lot of time. Well-written debug code is usually much more effective, and the added benefit is that they leave a lasting trace of your debugging. Of course it requires a bit of thought. Using a debugger won't leave any trace. Usually. I reckon people paid by the hour may prefer having to think less, and spend more time frantically hitting on keys.

Another point is that "real-time" issues can be very hard to track down using a debugger. Toggling IOs can help here. Even sending some info through SWO or even some UART - long gone are the days where you were limited to very low baud rates. You can set up SWO or UART for several Mbits/s, and thus make any logging have mininal overhead.

Now for any "complex" logic that could have a high probably of having bugs, what I sometimes do is just compile it on a PC, writing minimal additional code to test it. It gives you a way of adding automated testing without having to do this on the final target (which can be a pain with MCUs), while being able to occasionally use a debugger if required, with usually many more features than what you get with a MCU. So that's what I usually do for any complex logic, algorithms or even DSP. The benefit of C here (or C++) is that correctly written in a portable way, it can be compiled on a PC unmodified.

Logged

The following users thanked this post: Siwastaja

Pages: [**1**] 2 Next All **Go Up** PRINT SEARCH
« previous next »

Share me

EEVblog Electronics Community Forum » Electronics » Microcontrollers » Macro for efficient stm32 gpio handling

Jump to: => Microcontrollers go