Система тактирования

Первое что стоит сделать в нашем устройстве — настроить систему тактирования, дабы каждый периферийный блок мог работать корректно. Её обзор был сделан в разделе «Микроконтроллер под микроскопом»). Мы не станем разбирать каждый регистр модуля RCC (Reset and Clock Control), т.к. это сильно отвлечёт от более интересных задач, работы с периферией. Код для настройки будет предоставлен как есть, по завершению курса имеет смысл вернуться к этому разделу и разобраться с остальными регистрами самостоятельно.

В устройстве не предусмотрено внешнего генератора, а значит по умолчанию МК запускается от RC-цепочки и работает на частоте 8 МГц. Этого не достаточно, чтобы корректно работать с датчиком температуры, поэтому частоту нужно увеличить используя PLL, скажем, до 64 МГц.

> К слову, 64 МГц довольно высокая частота для реализации часов, но т.к. часы не единственная функция, ваша фантазия не ограничена, а оптимальность написанного вами кода не гарантирована, то остановимся на этом значении.

Алгоритм переключения на PLL следующий:

1. сброс регистров RCC;
2. запуск высокочастотного встроенного источника HSI (плюс ожидание выхода в рабочий режим);
3. настройка и запуск PLL (плюс ожидание выхода в рабочий режим);
4. настройка и запуск буфера flash-памяти (задержки);
5. установка предделителей шин (APB1, APB2, AHB);
6. переключение `SYSCLK` на PLL (плюс ожидание выхода в рабочий режим);
7. обновление переменной `SystemCoreClock`.

При настройке предделителей шин, нужно помнить о том, что не все они могут работать на той же частоте `SYSCLK`, так APB1 может работать максимум на 36 МГц. Вернитесь в раздел «Микроконтроллер под микроскопом» и рассмотрите диаграмму внимательно.

Ниже приведён код для SPL (для других библиотек обратитесь к репозиторию на github).

```
void pll_init(void) {
    // сброс регистров RCC (выбирается HSI)
    RCC_DeInit();
    // включение и ожидания HSI
    RCC_HSICmd(ENABLE);
    while(!RCC_GetFlagStatus(RCC_FLAG_HSIRDY));
    // настройка PLL, множетель 16, частота 4 * 16 = 64 МГц
    RCC_PLLConfig(RCC_PLLSource_HSI_Div2,RCC_PLLMul_16);
    // включение и ожидания PLL
    RCC_PLLCmd(ENABLE);
    while(!RCC_GetFlagStatus(RCC_FLAG_PLLRDY));
    // настройка и вллючение буфера flash-памяти
    FLASH_SetLatency(FLASH_Latency_2);
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
    // установка предделителей шин
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
```

```
    RCC_PCLK1Config(RCC_HCLK_Div2); // 36 MHz максимум!
    RCC_PCLK2Config(RCC_HCLK_Div1);
    // переключение SYSCLK на PLL
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
    while(RCC_GetSYSCLKSource()!=0x08);
    // обновление SystemCoreClock
    SystemCoreClockUpdate();
}
```

Скопируйте данный код в файл `utils.c` и объявите прототип в `utils.h`. При необходимости изменить частоту, вам нужно лишь поменять множитель.

Если у вас имеется осциллограф или логический анализатор, то вы можете вывести тактовый сигнал на ножку MCO. При помощи приведённого ниже кода вы можете посмотреть частоту SYSCLK или PLL/2.

```
// utils.c
void mco_init(void) {
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN; // включаем тактирование порта A

    GPIOA->CRH &= ~GPIO_CRH_CNF8;       // режим push-pull
    GPIOA->CRH |= GPIO_CRH_CNF8_1;      // alternative output
    GPIOA->CRH &= ~GPIO_CRH_MODE8;      // max speed
    GPIOA->CRH |= GPIO_CRH_MODE8_1 | GPIO_CRH_MODE8_0; // max speed

    RCC->CFGR &= ~(RCC_CFGR_MCO);         // Обнуляем MCO
    RCC->CFGR |= RCC_CFGR_MCO_SYSCLK;     // системная шина
    //RCC->CFGR |= RCC_CFGR_MCO_PLL;        // PLL/2
}
```

Подцепите крокодил к разъёму USB (он подключён к земле), а щуп приложите на медный пяточок (JP_1) рядом с МК. Обратите внимание,что максимальная частота, которую способна выдавать ножка равняется 50 МГц, т.е. при максимальной частоте вы увидите шум. Меняйте умножители PLL и посмотрите как меняется сигнал.

> Систему тактирования удобно настраивать в STM32CubeMX.

Вернёмся к понятию «регистр». По сути это просто ячейка памяти в которую можно записывать значения. В зависимости от состояния ячеек он активируют или деактивируют некоторую функциональность в системе. Блоком RCC отвечает за тактирование, следовательно через него можно включать или отключать тактирование портов ввода/вывода.

Звучит не сложно, но... усложним и провернём финт ушами. Это поможет вам осознать глубину происходящего. Откройте **Reference Manual** и найдите в оглавлении пункт «**3.3 Memory map**». В этой таблице записаны адреса периферийных блоков, нас интересует пункт «**Reset and clock control RCC**» (модуль системы тактирования).

Boundary address	Peripheral	Bus	Register map
0xA000 0000 - 0xA000 0FFF	FSMC	AHB	*Section 21.6.9 on page 564*
0x5000 0000 - 0x5003 FFFF	USB OTG FS		*Section 28.16.6 on page 911*
0x4003 0000 - 0x4FFF FFFF	Reserved		-
0x4002 8000 - 0x4002 9FFF	Ethernet		*Section 29.8.5 on page 1067*
0x4002 3400 - 0x4002 7FFF	Reserved		-
0x4002 3000 - 0x4002 33FF	CRC		*Section 4.4.4 on page 65*
0x4002 2000 - 0x4002 23FF	Flash memory interface		-
0x4002 1400 - 0x4002 1FFF	Reserved		-
0x4002 1000 - 0x4002 13FF	Reset and clock control RCC		*Section 7.3.11 on page 121*
0x4002 0800 - 0x4002 0FFF	Reserved		-
0x4002 0400 - 0x4002 07FF	DMA2		*Section 13.4.7 on page 289*
0x4002 0000 - 0x4002 03FF	DMA1		
0x4001 8400 - 0x4001 FFFF	Reserved		-
0x4001 8000 - 0x4001 83FF	SDIO		*Section 22.9.16 on page 621*

Таблица из Reference Manual, *Table 3. Register boundary addresses*

Он имеет адрес `0x40021000`.

Снова перейдите к оглавлению и в «**7 Low-, medium-, high- and XL-density reset and clock control (RCC)**» перейдите к пункту «**7.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)**».

7.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

Note: *When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Описание регистра `APB2ENR` из Reference Manual, *7.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)*

Данный регистр позволяет включать и отключать тактирование блоков периферии: портов ввода/вывода, SPI, USART и т.д. Записав `1` в четвёртую позицию включается тактирование порта B, и наоборот, записав туда `0`, тактирование будет убрано.

В строке «**Address**» у казано не абсолютное значение, а смещение относительно блока RCC, т.е. адрес регистра `0x40021000` + `0x18` = `0x40021018`. Записав `1` на место четвёртого бита можно включить тактирование порта B. Как это сделать? Очень просто!

```
*((u32 *)0x40021018) = 0x00000008;
```

Но, конечно же, никто так код не пишет (21 век, как-никак). Для каждой периферии заведена структура (CMSIS, `stm32f10x.h`), в которой хранятся переменные, связанные с соответствующими регистрами. В нашем случае это структура `RCC`. `0x00000008` писать долго, проще использовать битовую операцию смещения:

```
RCC->APB2ENR |= 1 << 3; // 1 смещается на 3 позиции, т.е. будет хранится в 4 ячейке
```

Такой стиль очень плох. Почему? Допустим, вы передали кому-то код (или еще проще, сами сели за этот проект спустя некоторое время), и для того, чтобы человек понял, что вы конкретно включили этой строчкой кода, ему потребуется залезть в документацию и посмотреть, за что отвечает третий (с нуля) бит в этом регистре. Неудобно, не правда ли? Можно написать комментарий, а можно сделать так, чтобы код говорил сам за себя. Если вы снова откроете `stm32f10x.h`, то сможете найти макросы масок для всех случаев жизни.

```
RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;
```

Так намного лучше. Самое время перейти к порта ввода/вывода.

Код урока можно найти на github: **CMSIS**.

Назад | **Оглавление** | **Дальше**