ARM. Учебный курс. Тактовый генератор STM32  
<http://easyelectronics.ru/arm-uchebnyj-kurs-taktovyj-generator-stm32.html>

Нам разум дал стальные руки-крылья,А вместо сердца — пламенный мотор

На прошлых занятиях, мы научились создавать проекты, настраивать их и даже научились моргать светодидом напрямую и с использованием функций стандартной библиотеки работы с периферией SPL и CMSIS. Все это, конечно, хорошо но хочется чего-то большего… Но прежде чем приступить к более сложным примерам, хотелось бы поговорить об одной очень важной системе микроконтроллера. Речь идет о *системе генерации тактовых частот*, которая, выражаясь метафорически, является сердцем нашего микроконтроллера. Думаю, ни для кого не секрет, что работа всех остальных систем микроконтроллера зависит от того, получают они тактовые сигналы или нет и какой частоты эти сигналы. Если какие-то блоки не будут получать тактовых сигналов, то они просто не будут работать. Ну, а от частоты этих сигналов зависит скорость работы этих блоков и (между прочим) – электропотребление микроконтроллера.

В нашей программе управления светодиодом, мы практически не касались вопросов генерации тактовой частоты и микроконтроллер вроде бы работал, а программа выполнялась, из сего некоторые могут сделать опрометчивый вывод, что раз работает, так не стоит туда и лезть.  
  
Однако, это не так! Если внимательно посмотреть на ассемблеровский файл начальной низкоуровневой инициализации микроконтроллера (STM32F10x.s/startup\_stm32f10x\_hd.s или подобный, что присутствует в проекте), то можно увидеть, что перед вызовом функции **main()** нашей программы, идет вызов CMSIS’овской функции **SystemInit()**:

**Заголовок для Keil из поставки STM32**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | ; Reset handler  Reset\_Handler PROC  EXPORT Reset\_Handler [WEAK]  IMPORT \_\_main  IMPORT SystemInit  LDR R0, =SystemInit  BLX R0  LDR R0, =\_\_main  BX R0  ENDP |

**IAR**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | PUBWEAK Reset\_Handler  SECTION .text:CODE:REORDER(2)  Reset\_Handler  LDR R0, =SystemInit  BLX R0  LDR R0, =\_\_iar\_program\_start  BX R0 |

А вот в том файле инициализации, что создает Keil’овский визард проекта при создании нового проекта вызова SystemInit нет!

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | ; Reset Handler    Reset\_Handler PROC  EXPORT Reset\_Handler [WEAK]  IMPORT \_\_main  LDR R0, =\_\_main  BX R0  ENDP |

И его надо вручную дернуть из main. В общем, смотри в оба что там да как :) Я (Di Halt) же предпочитаю вообще выпилить CMSISовский System Init и накатать свой. А то очень уж там все заморочено и заверчено. Тем более есть с чего взять пример ;)

Наш пытливый ум, может посмотреть на исходники функции **SystemInit** и увидеть, что она в сообществе с дополнительными функциями вида **SetSysClock / SetSysClockХХХ** как раз и настраивает для нас генераторы тактовых частот. Если мы решили заниматься программированием микроконтроллеров всерьез, то нам не пристало полагаться на какие-то там готовые функции и оставаться в неведении относительно их поведения, поэтому в данной статье я и постараюсь сорвать покровы тайны со столь важной системы нашего микроконтроллера :-)

Для лучшего усвоения информации приведу структурную схему системы формирования тактовых частот, из документации на микроконтроллер (раздел reset and clock control (RCC)).

|  |
| --- |
| http://easyelectronics.ru/img/ARM_kurs/PLL/1.png |

Как можно увидеть из данной схемы, практически все блоки микроконтроллера затактовываются от линии **SYSCLK**, что расшифровывается как System Clock (в моем вольном переводе это будет звучать как системная тактовая частота). Исключение составляют блоки USB, RTC и IWDG. Т.к. системная тактовая частота очень важна, то разберемся сначала с ее генерацией.

Как следует из документации и схемы, источниками для системной тактовой частоты могут служить 3 генератора:

* Генератор HSI — внутренний высокоскоростной.
* Генератор HSE — внешний высокоскоростной.
* Внутренний PLL — система фазовой автоподстройки частоты (ФАПЧ)) В нашем случае можно сказать, что это умножитель частоты с управляемым коэффициентом умножения.

На самом деле, IMHO, более правильно сказать, что источниками могут быть только HSI и HSE частота которых может умножаться, а может и нет.  
Разберемся с ними поподробнее.

**Встроенный RC генератор (HSI)**  
Встроенный в микропроцессор генератор HSI вырабатывает тактовую частоту 8 МГц. Генератор автоматически запускается при появлении питания Vcc и при выходе в нормальный режим работы выставляет флаг HSIRDY в регистре RCC\_CR. Первоначально процессорное ядро запускается на тактовой частоте HSI.  
К преимуществам относится быстрое время начала генерации тактовой частоты после подачи питания и отсутствие необходимости в использовании дополнительных электронных компонентов для работы микроконтроллера.

Недостаток – низкая стабильность частоты генерируемого сигнала, а при умножении на PLL погрешность тоже умножается. Например, частота HSI при разных температурных условиях плавает от 7.3 до 8.7МГц. При множителе в 9 на выходе будет разброс уже от 65.7 до 78.3 МГЦ.

Производитель гарантирует стабильность частоты в 1% при температуре ядра 25C, для чего в процессе производства в регистр RCC\_CR записываются биты HSICAL[7:0], отвечающие за его калибровку. Если этого недостаточно или необходимо откалибровать микроконтроллер для работы при другой температуре, то можно воспользоваться битами HSITRIM[4:0] этого же регистра. После ресета там находится число 16, т.е. середина диапазона регулировки.  
Изменение одного младшего бита дает подстройку приблизительно в 40кГц. Тактовая частота с генератора HSI может подаваться на прямую как источник системной частоты, либо поступать в блок умножителя частоты предварительно деленная на 2.

Генератор HSI может быть включен/выключен управлением бита HSION регистра RCC\_CR.

**Внешний генератор (HSE)**  
В качестве внешнего генератора могут выступать:

* Внешний тактовый сигнал не превышающий 25МГц, поданный на ножку OSC\_IN в то время, как нога OSC\_OUT находится в высокоимпендансном состоянии.
* Внешний кварцевый резонатор подключенный на ножки OSC\_IN и OSC\_OUT. Внешний кварцевый резонатор должен быть в диапазоне от 4 до 16МГц и при его использовании достигается очень высокая стабильность частоты работы генератора.

Внешний генератор HSE по умолчанию выключен и его включение/выключение управляется битом HSEON регистра RCC\_CR. После включения HSE и его выхода в рабочий режим устанавливается бит HSERDY кроме этого, может быть сгенерировано прерывание. Также как и сигнал с генератора HSI, сигнал HSE может быть подан напрямую в качестве системного тактового сигнала либо поступать в блок умножения. Но в отличие от HSI в блок умножения он может поступать напрямую, либо пройдя через делитель на 2.

**PLL**  
Внутренний умножитель частоты может умножать вошедший тактовый сигнал с одного из трех источников, на выбор

* HSI/2
* HSE
* HSE/2

на множитель от 2-х до 16-ти. По умолчанию умножитель выключен и его включение/выключение управляется битом PLLON регистра RCC\_CR. После включения PLL и его выхода в рабочий режим устанавливается бит PLLRDY кроме этого, может быть сгенерировано прерывание.

Работа умножителя конфигурируется через регистр RCC\_CFGR.

*Все манипуляции над его режимами работы должны проводиться только при выключенном PLL!*

В данном регистре:

* Бит PLLSRC задает источник умножения, либо HSI либо HSE.
* Бит PLLXTPRE задает будет ли сигнал с HSE предварительно делиться на 2 или нет.
* Биты PLLMUL[3:0] задают коэффициент умножения от 2-х до 16-ти.

С источниками системной тактовой частоты вроде разобрались. Теперь, прежде чем пройти дальше, куда там эта частота подается, отойдем немного в сторону и посмотрим, что у нас есть еще… :-)

А есть у нас во первых блок USB. Особенностью его является то, что работать он может только на частоте 48МГц! Если частота будет отличной от 48МГц, то другие устройства на шине USB нас не поймут а мы их. Из схемы видно две вещи:

1. Тактовые сигналы на блок USB идут с PLL.
2. Перед блоком USB Стоит делитель с управляемым коэффициентом деления 1 или 1.5.

Из этого следует, что если нам нужна работа USB, то обязательно должен работать PLL и его выходная тактовая частота должна быть либо 48 (при делителе USB = 1) либо 72 (при делителе USB = 1.5) МГц! И все! Больше никаких вариантов.

Во вторых, есть у нас еще RTC (Real Time Clock – часы реального времени). Что такое часы реального времени, думаю знают все. Они могут работать, от батарейного питания, при выключенном микроконтроллере. Источником тактовой частоты для них могут являться:

1. Тактовые сигналы генератора HSE деленные на 128.
2. Тактовые сигналы генератора LSI (Low Speed Internal – внутренний низкоскоростной), который генерирует 40кГц.
3. Тактовые сигналы генератора LSE (Low Speed External – внешний низкоскоростной), для его работы нужен внешний «часовой» кварц на 32.768кГц.

Ну и в третьих, есть еще IWDG (Independent Watchdog – независимая сторожевая собака :-) Что это такое в данной статье вдаваться не будем). IWDG тактуется сугубо от LSI описанном чуть выше.

Ну вроде все, теперь можно перейти и к остальным блокам, которые тактуются от системной частоты SYSCLK.

Дальше системная тактовая частота попадает в делитель шины AHB, который делит ее на делитель от 1 до 512, и все остальные блоки получают на вход уже эту деленную частоту (некоторые блоки имеют еще дополнительные делители). По схеме все видно, какой блок имеет делитель а какой нет, какие коэффициенты деления у делителей, и какая периферия в итоге какую частоту получает.

Например, блоки SDIO и FSMC работают на частоте шины AHB. Шины APB1 и APB2 имеют собственные делители с коэффициентами от 1 до 16. Блок ADC имеет делитель от 2 до 8 и т.д.

Так же на схеме подписаны максимальные допустимые частоты для блоков. Все делители имеющие управляемые коэффициенты могут быть настроены, что в целом позволяет довольно гибко варьировать частоты отдельных блоков в различных пределах.  
Кроме того, на схеме это не показано, очень важно знать и помнить 2 вещи:

1. Какие блоки на какой шине сидят. Например, порты ввода вывода сидят на шине APB2, контроллеры шины I2C сидят на шине APB1, контроллера прямого доступа к памяти на шине AHB а, например, таймеры, частью сидят на APB2 а частью на APB1.
2. Каждый блок имеет свой вход тактовых сигналов и бит управления этим входом. **И по умолчанию практически все блоки отключены от тактовых сигналов!!!** Di Halt на эти грабли уже наступал. И мы тоже, в программе моргания светодиодом разрешали вход тактовых сигналов на блок GPIOA т.е. порт А.

Т.е., если нужна работа какого-то блока: GPIO, DMA, ADC, DAC, таймеров и т.д., то не забываем подавать на них тактовые сигналы, т.е. устанавливать соответствующие биты! Иначе работать ничего не будет!

Зачем же нужно такое разнообразие делителей, умножителей, источников синхронизации и т.д? А нужно это все для гибкой и точной настройки микроконтроллера под задачу, которую ему предстоит решать в том или ином проекте. Если не нужны какие-то блоки, то просто отключаем их от источников синхронизации, нечего им энергию потреблять вхолостую (что по умолчанию и сделано). Если не требуется в проекте высокая производительность, то можно сконфигурировать микроконтроллер на работу на частоте, например, 48, 36, 24 и т.д. МГц, что опять же понизит его энергопотребление и т.д.

**Алгоритм ручной настройки частоты**

* Определяемся с чего будем работать (HSI или HSE) и с какой частотой.
* Если HSE, то выбираем внешний кварц. Гнаться за частотой не надо — чем медленней кварц, тем надежней он работает т.к. менее требователен к качеству разведения платы. Оптимально 8…12МГц. Но тут надо учитывать, что мы должны «попасть» в частоту USB (опять же если USB нужна, если нет то пофигу)
* Затем определяем каким путем пойдет тактовый сигнал через все эти мультиплексоры/делители/множители попутно вычисляем нужные величины делителей.
* Начинаем разглядывать битики в RCC регистрах, определяясь что нам и каким образом нужно выставить
* Выставляем!

Но не все так просто. По крайней мере в STM32. Тут мало взять и навтыкать конфигураций, а вдруг мы переключились на кварц, а его нету? Или на дорожки попала влага и он не завелся? Или банально забыли включить? Есть система защиты, по правилам которой мы можем переключиться на другой источник тактирования только в том случае, когда точно знаем, что с ним все в порядке. Сделано это на флагах. Т.е. , например, включили мы PLL и ждем пока появится флаг готовности PLL и только тогда переключаемся. Иначе все повиснет нафиг.

Стоит рассказать еще об одном блоке, который сам в генерации тактовых сигналов участия не принимает, но иногда может оказаться весьма полезным. На схеме он обозначен как CSS Из схемы видно, что на вход ему подается тактовая частота с HSE а выход из этого блока заведен на блок выбора источника системного тактового сигнала. Итак, что же это за блок какой-то защиты? Назначение этого блока — следить за тактовыми сигналами поступающими с HSE и если с ними случится какой-то косяк, например они пропадут (кварц там у нас отвалился или еще что произошло), то этот блок может вмешаться в ситуацию и тогда происходит следующее:

1. Автоматически отключается HSE.
2. Посылается событие на останов работы расширенных таймеров TIM1 и TIM8.
3. Генерируется прерывание CSSI, которое внутри процессора заведено на немаскируемое прерывание NMI, т.е. пропустить его невозможно.
4. Если в качестве источника системной частоты использовался HSE напрямую либо через PLL, то источник системной частоты автоматом переключается на HSI.
5. Если PLL был активен и входом ему служил сигнал с HSE, то PLL отключается.

Таким образом, микроконтроллер сохранит свою работоспособность, пусть и на пониженной и менее стабильной частоте, и проинформирует выполняющуюся программу об этом. А теперь представьте, что бы было, если бы этот микроконтроллер стоял в какой-то критически важной системе и просто бы остановился при проблеме с кварцем.

Ну и для того, чтобы эта прелесть заработала ее надо включить программно через установку бита CSSON регистра RCC\_CR. По умолчанию он сброшен.

**Код**  
Теперь, собственно, вернемся к коду настройки генераторов.  
Вот как это сделано в CMSIS для STM32:

*Сразу оговоримся, что это справедливо только для простой линейки STM32 для Connectivity Line все чуток по другому. Но очень похоже.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | \_\_IO uint32\_t StartUpCounter = 0, HSEStatus = 0;    */\* Конфигурацяи SYSCLK, HCLK, PCLK2 и PCLK1 \*/*  */\* Включаем HSE \*/*  RCC->CR |= ((uint32\_t)RCC\_CR\_HSEON);    */\* Ждем пока HSE не выставит бит готовности либо не выйдет таймаут\*/*  do  {  HSEStatus = RCC->CR & RCC\_CR\_HSERDY;  StartUpCounter++;  }  while( (HSEStatus == 0) && (StartUpCounter != HSEStartUp\_TimeOut));    if ( (RCC->CR & RCC\_CR\_HSERDY) != RESET)  {  HSEStatus = (uint32\_t)0x01;  }  else  {  HSEStatus = (uint32\_t)0x00;  }    */\* Если HSE запустился нормально \*/*  if ( HSEStatus == (uint32\_t)0x01)  {  */\* Включаем буфер предвыборки FLASH \*/*  FLASH->ACR |= FLASH\_ACR\_PRFTBE;    */\* Конфигурируем Flash на 2 цикла ожидания \*/*  */\* Это нужно потому, что Flash не может работать на высокой частоте \*/*    FLASH->ACR &= (uint32\_t)((uint32\_t)~FLASH\_ACR\_LATENCY);  FLASH->ACR |= (uint32\_t)FLASH\_ACR\_LATENCY\_2;      */\* HCLK = SYSCLK \*/*  RCC->CFGR |= (uint32\_t)RCC\_CFGR\_HPRE\_DIV1;    */\* PCLK2 = HCLK \*/*  RCC->CFGR |= (uint32\_t)RCC\_CFGR\_PPRE2\_DIV1;    */\* PCLK1 = HCLK \*/*  RCC->CFGR |= (uint32\_t)RCC\_CFGR\_PPRE1\_DIV2;    */\* Конфигурируем множитель PLL configuration: PLLCLK = HSE \* 9 = 72 MHz \*/*  */\* При условии, что кварц на 8МГц! \*/*  */\* RCC\_CFGR\_PLLMULL9 - множитель на 9. Если нужна другая частота, не 72МГц \*/*  */\* то выбираем другой множитель. \*/*  RCC->CFGR &= (uint32\_t)((uint32\_t)~(RCC\_CFGR\_PLLSRC | RCC\_CFGR\_PLLXTPRE | RCC\_CFGR\_PLLMULL));  RCC->CFGR |= (uint32\_t)(RCC\_CFGR\_PLLSRC\_HSE | RCC\_CFGR\_PLLMULL9);    */\* Включаем PLL \*/*  RCC->CR |= RCC\_CR\_PLLON;    */\* Ожидаем, пока PLL выставит бит готовности \*/*  while((RCC->CR & RCC\_CR\_PLLRDY) == 0)  {  *// Ждем*  }    */\* Выбираем PLL как источник системной частоты \*/*  RCC->CFGR &= (uint32\_t)((uint32\_t)~(RCC\_CFGR\_SW));  RCC->CFGR |= (uint32\_t)RCC\_CFGR\_SW\_PLL;    */\* Ожидаем, пока PLL выберется как источник системной частоты \*/*  while ((RCC->CFGR & (uint32\_t)RCC\_CFGR\_SWS) != (uint32\_t)0x08)  {  *// Ждем*  }    }  else  {  */\* Все плохо... HSE не завелся... Чего-то с кварцем или еще что...*  *Надо бы както обработать эту ошибку... Если мы здесь, то мы работаем*  *от HSI! \*/*  } |

В CMSIS@STM32 в файле system\_stm32f10x.c есть штука вида:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | #if defined (STM32F10X\_LD\_VL) || (defined STM32F10X\_MD\_VL)    */\* #define SYSCLK\_FREQ\_HSE HSE\_Value \*/*  #define SYSCLK\_FREQ\_24MHz 24000000    #else    */\* #define SYSCLK\_FREQ\_HSE HSE\_Value \*/*  */\* #define SYSCLK\_FREQ\_24MHz 24000000 \*/*  */\* #define SYSCLK\_FREQ\_36MHz 36000000 \*/*  */\* #define SYSCLK\_FREQ\_48MHz 48000000 \*/*  */\* #define SYSCLK\_FREQ\_56MHz 56000000 \*/*  #define SYSCLK\_FREQ\_72MHz 72000000    #endif |

Как видишь, тут уже готовые макросы для разных частот. Остается только раскомментировать нужную строку.  
Надо заметить, что CMSIS для STM32 имеет все макроопределения заточенны под кварц на 8МГц. Под другой кварц уже не покатит — придется пересчитывать заново все делители.

**Пример**  
Скреативим программку, которая будет моргать диодиком и на ходу менять значения умножителя PLL каждые десять миганий. Также там будет активирована защита от сбоев тактового генератора. Сбить с копыт кварцевый генератор несложно, достаточно наслюнявить палец и коснуться обоих выводов кварца :) Ну или, например, припаять проводочки к нему и коротнуть его выводы на резистор в 100ом.

См. файл. Пример работы с PLL STM32 для Keil uVision 4.docx

**З.Ы.**  
На последок, хотел еще чуть чуть затронуть такую тему как «разгон» или работа на повышенных частотах. Была у меня одна задача, когда надо было получать и обрабатывать данные по одной достаточно высокоскоростной параллельной шине (порядка 40МГц). Так вот «родной» частоты в 72МГц не хватало для нормальной работы, что и не удивительно. И тогда я решил разогнать немного микроконтроллер. В общем, мой опыт показал, что частоту в 144МГц (т.е. в 2 раза!!! больше стандартной) они переваривают легко. Проверял на 3-х имеющихся в наличии контроллерах STM32F103RB, STM32F103RE и STM32F103VE. Все 3 работают стабильно, не греются. Частоту в 160МГц, уже не переваривают… Зависают через 10-15 секунд работы, но при этом также не греются. Для сравнения, имеющийся в наличии NXP LPC1768 с «родной» частотой 100МГц на 120 уже не завелся вообще.

Я никого ни к чему не призываю, это просто хинт, может быть кому-то когда-то будет полезен, как и мне в свое время… Ну и в даташите черным по белому написано НЕ ПРЕВЫШАЙТЕ максимально допустимых производителем частот работы блоков. Все, что выше — на ваш страх и риск, без гарантий и т.д. и т.п… Ну да на заборе тоже вон чего написано, а на самом деле там дрова :-D

*Основной автор статьи — Владимир aka RtxOnAir  
Редактирование и всякие дополнения уточнения — DI HALT*