Общие требования к выполнению лабораторных работ по программированию

Программа работы

1. Работа выполняется с использованием программы Visual Studio (лицензия ТПУ) и NotePad или NotePad++ (бесплатная лицензия). Вместо NotePad и NotePad++ можно использовать стандартный Блокнот ОС Windows.
2. Пишется исходный код в указанной в работе программе. Для случая работ в Visual Studio студент самостоятельно компилирует файл. На проверку, помимо отчета с описанием работы созданного приложения, присылается exe-файл. Для случая написания кода для микроконтроллера, студентом предоставляется только текстовый файл с исходным кодом. Компиляция и программирование микроконтроллера осуществляется преподавателем.
3. Все лабораторные проекты будут выполняться на микроконтроллере STM32F429IIT6.

Требования к отчёту по работе

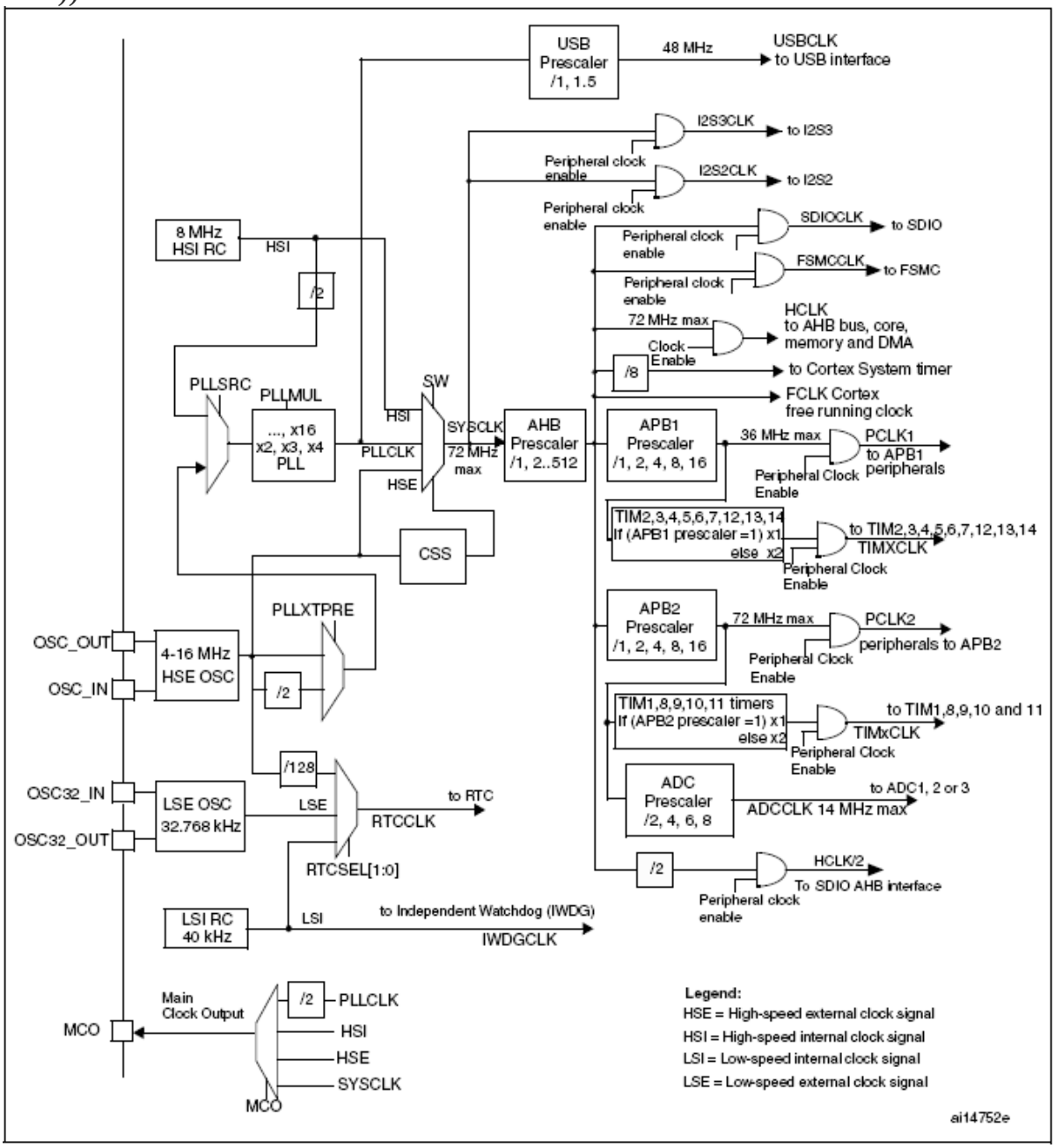
Отчет по лабораторной работе должен содержать:

1. Титульный лист.
2. Цель работы.
3. Этапы выполнения работы.
4. Результаты исследований в виде таблиц и графиков с пояснениями к ним.
5. Выводы по работе, в которых должен содержаться детальный анализ полученных результатов и их интерпретация.

Лабораторная работа № 4. Разработка программы контроллера: управление периферией, обмен данными, таймеры.

Изучение системы тактирования микроконтроллера

Ниже представлена структурная схема системы формирования тактовых частот, из документации на микроконтроллер (раздел reset and clock control (RCC)).



Как можно увидеть из данной схемы, практически все блоки микроконтроллера тактируются от линии **SYSCLK**, что расшифровывается как System Clock (системная тактовая частота).

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 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, например, 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, если частота HSI при разных температурных условиях плавает от 7.3 до 8.7МГц. При множителе PLL в 9 на выходе будет разброс уже от 65.7 до 78.3 МГц.

Генератор HSI может быть включен/выключен управлением бита **HSION** регистра **RCC_CR**.

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

- Внешний тактовый сигнал с частотой не выше 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 25МГц 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 (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 (смотри документацию к МК):

- 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.

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.

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 к какой шине подключены. Например, порты ввода вывода подключены к шине APB2, контроллеры шины I2C – к шине APB1, контроллера прямого доступа к памяти на шине AHB а, например, таймеры, частью подключены к 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!

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

Пример программного кода

```
1
2   __IO uint32_t StartUpCounter = 0, HSEStatus = 0;
3
4   /* Конфигурацяи  SYSCLK, HCLK, PCLK2 и PCLK1 */
5   /* Включаем HSE */
6   RCC->CR |= ((uint32_t)RCC_CR_HSEON);
7
8   /* 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
                }
```

```
71          /* Выбираем PLL как источник системной частоты */
72          RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
73          RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;
74
75          /* Ожидаем, пока PLL выберется как источник системной частоты */
76          while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08)
77                  {
78                  // Ждем
                    }

          }
else
          {
                  /* Не работает */
          }
```

Настройка базового таймера

Таймеры — это периферия контроллера STM32, позволяющая отсчитывать интервалы времени. Следует начать с того, что в контроллерах STM32 существуют таймеры, которые делятся на три группы. Самые простые это ***Basic timers***. Они хороши тем, что очень просто настраиваются и управляются при помощи минимума регистров. Все что они умеют это отсчитывать временные интервалы и генерировать прерывания, когда таймер досчитает до заданного значения. Следующая группа (***general-purpose timers***) более продвинутая – они умеют генерировать ШИМ, умеют считать импульсы, поступающие на определённые ножки и т.д. И самый мощный таймер – это ***advanced-control timer***.

Basic таймеры (TIM6 и TIM7) подключены к шине APB1. Основной регистра – **TIMx_CNT** (здесь и далее **x** — номер basic таймера 6 или 7). Это счётный 16-ти битный регистр, занимающийся непосредственно счётом времени. Каждый раз, когда с шины **APB1** приходит тактовый импульс, содержимое этого регистра увеличивается на единицу. Когда регистр переполняется, все начинается с нуля. У таймера есть предделитель, управлять которым можно при помощи регистра **TIMx_PSC**. Записав в него значение, например, 24000-1 (при частоте шины APB1 24 МГц) мы заставим счётный регистр **TIMx_CNT** увеличивать свое значение каждую милисекунду (Частоту **APB1** делим на число в регистре предделителе и получаем сколько раз в секунду увеличивается счётчик). Единицу нужно вычесть потому, что если в регистре ноль, то это означает, что включен делитель на единицу. Для того чтоб счётчик обнулялся досрочно, а не когда досчитает до предельного значения 0xFFFF, служит регистр **TIMx_ARR**. Записываем в него то число, до которого должен досчитывать регистр **TIMx_CNT** перед тем как обнулиться. Если мы хотим, чтобы прерывание возникало раз в секунду, то нам нужно записать туда 1000. Для включения таймера установливаем бит **CEN** в

регистре **TIMx_CR1**. Этот бит разрешает начать отсчёт, соответственно если его сбросить, то отсчет остановится. В регистре **TIMx_DIER**. Интересен бит **UIE,** который активирует возможность прерывания программы микроконтроллера.

Пример программного кода

```
//----------Base timer TIM6 -> APB1
#define    TIM6_ON           (1 << 4)
#define    TIM6_RESET_FLAG   (0x1)
#define    TIM6_CR1_CLEAR    ((1 << 7) | (1 << 3) | (1 << 2) | (1 << 1)
| (0x1))
#define    TIM6_CR2_CLEAR    ((1 << 6) | (1 << 5) | (1 << 4))
#define    TIM6_DIER_CLEAR   ((1 << 8) | (0x1))

RCC->APB1ENR |= TIM6_ON; // activate TIM6 - it is on APB1 port
TIM6->CR2 &= ~TIM6_CR2_CLEAR;    // Bits 6:4 MMS[2:0]: Master mode
     selection: 000: Reset - the UG bit from the TIMx_EGR register is
     used as a trigger output (TRGO).
TIM6->DIER  &= ~TIM6_DIER_CLEAR;
TIM6->DIER |= 0x00000001;  // UIE: Update interrupt enable: 1: Update
interrupt enabled - allow to interrupt program when timer reached max
(set up in ARR).
TIM6->SR = ~TIM6_RESET_FLAG;    // UIF: Update interrupt. This bit is
     set by hardware on an update event. It is cleared by software. 0
     = reset flag
TIM6->PSC = 0x20CF;        // prescaller: fCK_PSC / (PSC[15:0] + 1),
     fCK_PSC = F_APB1 (42 MHz)* 2 (see Init_RCC function, in default
     it was 16 MHz): 1/0.1 ms = 10 kHz = 84 MHz/(8399 + 1)
TIM6->ARR = 0x3E7;         // number of cycles of rescaller: time =
     TIM6->PSC *  TIM6->ARR: 0.1 s = 0.1ms * (1000 - 1)
TIM6->CR1 &= ~TIM6_CR1_CLEAR;
TIM6->CR1 |= (1 << 7) |    // ARPE: Auto-reload preload enable: 1:
     TIMx_ARR register is buffered.
     (1 << 2) |            // URS: Update request source: 0 - any
     sourse of update, 1- Only counter overflow/underflow generates an
     update interrupt or DMA request if enabled.
      (0x00000001);        // CEN: 1: Counter enabled
```

Настройка портов ввода-вывода

За включение тактирования периферийных блоков отвечают регистры RCC XXX peripheral clock enable register. На месте XXX могут стоять шины AHB1, AHB2, AHB3, APB1 и APB2. Например, тактирование периферийного блока GPIOD включается установкой «1» в третий бит регистра *RCC AHB1 peripheral clock enable register (**RCC_AHB1ENR**).*

7.3.10 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGHS ULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved		DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	Reserved	
	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
Reserved			CRCEN	Reserved			GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 3 **GPIODEN:** IO port D clock enable
Set and cleared by software.
0: IO port D clock disabled
1: IO port D clock enabled

Можно пользоваться готовыми командами:

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN; // включим тактирование порта
```

Затем необходимо установить режим работы пинов порта как *General purpose output mode*, что означает что контроллер GPIO будет управлять состоянием пина МК. Управление режимом работы пинов МК производится помощью регистра *GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)*:

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

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

Bits 2y:2y+1 **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

Пример: **GPIOD->MODER = 0x55000000 = 0b 0101 0101 0000 0000 0000 0000 0000 0000;**

// включим ножки 12,13,14,15 на выход

Если записать все нули, то пин будет сконфигурирован как вход. Входы могут работать в одном из трех режимов:

- no pull up/down resistors
- pull-up – a resistor connected to high
- pull-down – a resistor connected to low

Выходы могут работать в одном из трех режимов:

- open drain – a transistor connects to low and nothing else
- open drain, with pull-up – a transistor connects to low, and a resistor connects to high
- push-pull – a transistor connects to high, and a transistor connects to low (only one is operated at a time)

Входы

Подтягивающие резисторы могут быть заданы с помощью регистра:

8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..I/J/K)

Address offset: 0x0C

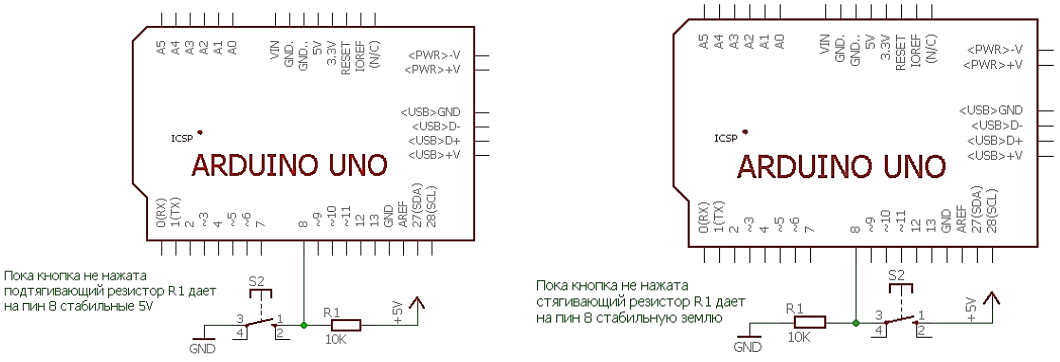
Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[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
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O pull-up or pull-down
00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

Подтягивающий резистор нужен, чтобы гарантировать на логическом входе, с которым соединён проводник, высокий (в первом случае) либо низкий (во втором случае) уровень пока вход разомнут, чтобы не оставлять его в «подвешенном» состоянии.



Выходы

Блок GPIO позволяет применить дополнительные настройки для выходных пинов порта. Данные настройки производятся в регистрах:

- GPIO port output type register (GPIOx_OTYPER) — задается тип выхода push-pull или open-drain. По умолчанию — двухтактный (push-pull) для каждой ножки
- GPIO port output speed register (GPIOx_OSPEEDR) — задается скорость работы выхода

8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..I/J/K)

Address offset: 0x04

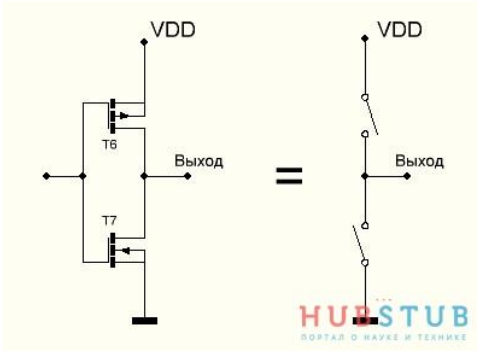
Reset value: 0x0000 0000

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
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

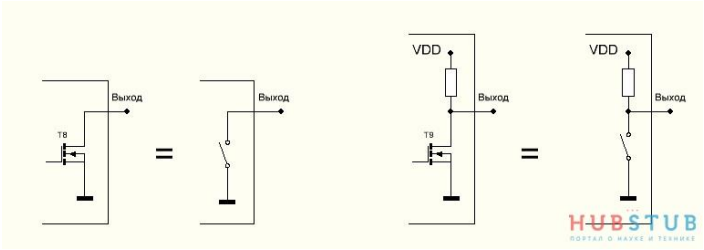
Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)
These bits are written by software to configure the output type of the I/O port.
0: Output push-pull (reset state)
1: Output open-drain

Open drain – используется для подключения нескольких устройств по принципу ИЛИ. В режиме push-pull такое подключение невозможно. Push-pull конфигурация чаще всего используется в интерфейсах, имеющих однонаправленные линии (передача по линии осуществляется только в одном направлении – SPI, UART и т.д.). В Open drain в большинстве случаев используется внешний подтягивающий резистор (есть микроконтроллеры, которые обеспечивают внутренние подтягивающие резисторы для конфигураций с Open drain). Выходы с Open drain чаще всего используются в интерфейсах связи, где несколько устройств подключены к одной линии (например, I2C, One-Wire и т.д.). Когда все выходы устройств, подключенных к линии, находятся в состоянии Hi-Z (high impedance – используется только транзистор), они все «видят» один уровень высокий (для случая open-drain, with pull-up) или низкий.

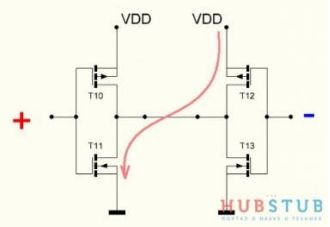


Режим push-pull: по сути, он состоит из двух ключей, один подтягивает вывод к питанию, другой к земле.

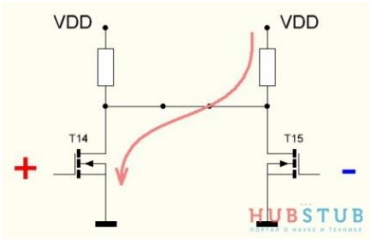


Режим **open-drain**. Справа open-drain, with pull-up

Рассмотрим два выхода push-pull соединенных с разной полярностью. Из-за конфликта уровней один из выходов просто выгорит, так как ток, возникший из-за разности потенциалов, ничем не ограничен.



А теперь так же соединим два вывода **OD**.



Здание скорости. Это задается только для пинов, настроенных как выходы. Использование чрезмерно высокой скорости может вызвать звон и электромагнитные помехи на выходах, поэтому важно использовать минимальную скорость, необходимую для вашего приложения:

8.4.3 GPIO port output speed register (GPIOx_OSPEEDR) (x = A..I/J/K)

Address offset: 0x08

Reset values:

- 0x0C00 0000 for port A
- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [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
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **OSPEEDRy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

00: Low speed

01: Medium speed

10: High speed

11: Very high speed

Note: Refer to the product datasheets for the values of OSPEEDRy bits versus V_{DD} range and external load.

- Low speed- 2 MHz
- Medium speed – 25 MHz
- High speed – 50 MHz
- Very high speed - 100 MHz

Пример программного кода

```
//----------GPIO
#define    PORT_D_ON   (1 << 3)
#define    GPIOD_MODE_MSK    (0x1 << 6) // 01: General purpose output mode, for each pin of port there are 2 bits of register GPIOx->MODER
#define    GPIOD_MODE_CLEAR_MSK   (0x3 << 6)
#define    GPIOD_TYPE_MSK         (0 << 3)       // 0: Output push-pull (reset state), 1: Output open-drain
#define    GPIOD_TYPE_CLEAR_MSK   (1 << 3)
#define    GPIOD_SPEED_MSK        (0x0 << 6)     // 00: Low speed, 01: Medium speed, 10: High speed, 11: Very high speed
#define    GPIOD_SPEED_CLEAR_MSK  (0x3 << 6)
#define    GPIOD_PPUDR_MSK        (0x0 << 6)     // 00: No pull-up, pull-down, 01: Pull-up, 10: Pull-down
#define    GPIOD_PPUDR_CLEAR_MSK  (0x3 << 6)
#define    GPIOD_PD3_ON      (1 << 3)
#define    GPIOD_PD3_OFF     ~(GPIOD_PD3_ON)
#define    GPIOD_LCKR_MSK     (0x0)    // Bits 15:0 LCKy: Port x lock bit y (y= 0..15)
#define    GPIOD_LCKR_CLEAR_MSK   (0x0001FFFF)
#define    GPIOD_AFR1_MSK         (0x0)     // AFRy: Alternate function selection for port x bit y (y = 0..7)
#define    GPIOD_AFR1_CLEAR_MSK   (0xFFFFFFFF)   // 0000: AF0 - SYS
```

```
#define    GPIOD_AFR2_MSK         (0x0)          // AFRy: Alternate
function selection for port x bit y (y = 8..15)
#define    GPIOD_AFR2_CLEAR_MSK   (0xFFFFFFFF)   // 0000: AF0 - SYS

RCC->AHB1ENR |= PORT_D_ON;       // activate controller GPIOD - it is
on AHB1 port
GPIOD->MODER &= ~GPIOD_MODE_CLEAR_MSK;
GPIOD->MODER |= GPIOD_MODE_MSK;
//------------------
GPIOD->OTYPER &= ~GPIOD_TYPE_CLEAR_MSK;
GPIOD->OTYPER |= GPIOD_TYPE_MSK;
//------------------
GPIOD->OSPEEDR &= ~GPIOD_SPEED_CLEAR_MSK;
GPIOD->OSPEEDR |= GPIOD_SPEED_MSK;
//------------------
GPIOD->PUPDR &= ~GPIOD_PPUDR_CLEAR_MSK;
GPIOD->PUPDR |= GPIOD_PPUDR_MSK;
//------------------
GPIOD->LCKR &= ~GPIOD_LCKR_CLEAR_MSK;
GPIOD->LCKR |= GPIOD_LCKR_MSK;
//------------------
GPIOD->AFR[0] &= ~GPIOD_AFR1_CLEAR_MSK;
GPIOD->AFR[1] &= ~GPIOD_AFR2_CLEAR_MSK;
GPIOD->AFR[0] |= GPIOD_AFR1_MSK;
GPIOD->AFR[1] |= GPIOD_AFR2_MSK;
//------------------
GPIOD->ODR &= GPIOD_PD3_OFF;
```

Задание на работу

1. Написать программный код на языке Си настройки системы тактирования RCC (используется внешний тактовый генератор с частотой 25 МГц), базового таймера TIM6 (прерывание каждую секунду) и контроллера ввода-вывода GPIOD (пин 3).
2. Написать программный код на языке Си управления выходным сигналом на третьей ножке (пин 3) контроллера ввода-вывода GPIOD: снятие и установка напряжения на ножке 1 раз в секунду (время должен

отсчитывать таймер TIM6) – эффект мигания лампочки.

Наблюдаемые параметры

1. Контроль появления импульсов на ножке пина 3 GPIOD с помощью цифрового осциллографа.