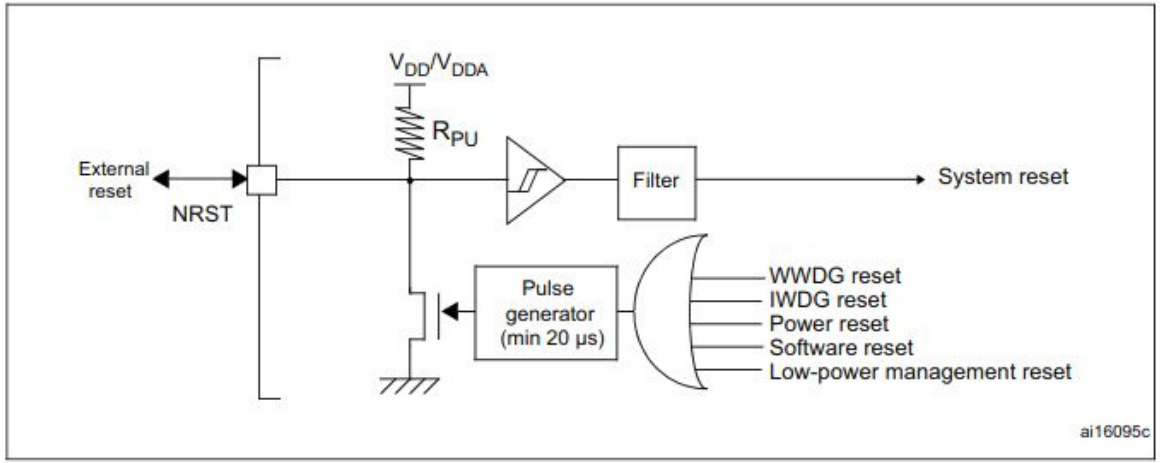
STM32F746g
FROM
SCRATCH

1. Как стартует процессор после сброса/включения питания
2. Memory Layout программы и linker скрипт
3. Собираем проект
4. Тактирование
5. GPIO, регистры
6. USART, регистры



6.4.1 GPIO port mode register (GPIOx_MODER) (x =A to K)

Address offset:0x00

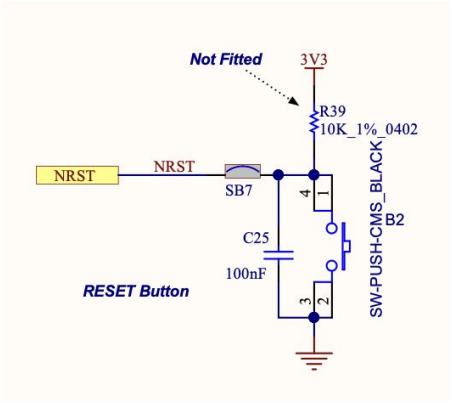
Reset value:

- 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 31:0 **MODER[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

System Reset устанавливает все регистры микроконтроллера в их значение по умолчанию. Значение по умолчанию указано для каждого регистра в reference manual документе.

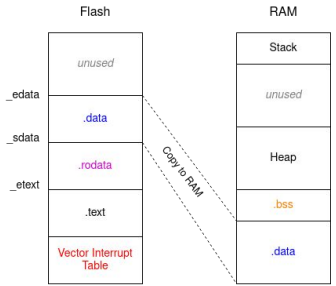


Sofware Reset - SYSRESETREQ bit in Cortex®-M7 Application Interrupt and Reset Control Register

```
bits(32) vectortable = VTOR<31:7>:'0000000';
SP_main = MemA_with_priv[vectortable, 4, AccType_VECTABLE] AND 0xFFFFFFFC<31:0>;
SP_process = ((bits(30) UNKNOWN):'00');
LR = 0xFFFFFFFF<31:0>;          /* preset to an illegal exception return value */
tmp = MemA_with_priv[vectortable+4, 4, AccType_VECTABLE];
tbit = tmp<0>;
APSR = bits(32) UNKNOWN;        /* flags UNPREDICTABLE from reset */
IPSR<8:0> = Zeros(9);           /* Exception Number cleared */
EPSR.T = tbit;                  /* T bit set from vector */
EPSR.IT<7:0> = Zeros(8);        /* IT/ICI bits cleared */
BranchTo(tmp AND 0xFFFFFFFE<31:0>);  /* address of reset service routine */
```

Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
.		.	.
.		.	.
.		.	.
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			
9			Reserved
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

1. Включение питания или Reset сигнал на процессор, приводят к началу исполнения инструкций процессором с адреса расположенного в таблице векторов прерываний со смещением 0x04 – **Reset Vector**. По дефолту таблица прерываний находится во Flash памяти по адресу **0x0000 0000** – который на ваших бордах мапится на **0x0800 0000**. Таблицу векторов прерываний необходимо прошить правильным образом, чтобы прыгнуть на нужное место в коде – ваш загрузчик. А также правильно выставить SP на SRAM память.
2. Код и данные, для инициализации глобальных переменных, лежат во Flash памяти. На старте вам необходимо, инициализировать .data секцию, скопировав данные из .data из Flash в .data в SRAM. Также необходимо заполнить .bss секцию в SRAM нулями.
3. Переустановить SP (если это нужно).
4. Прыгнуть на main, и начать инициализацию периферии (если это нужно).

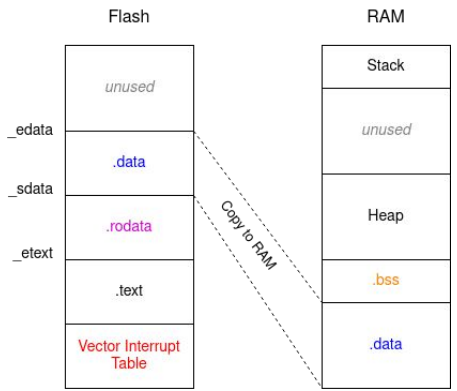


STM32F746NGHx_FLASH.ld:

```
/* Specify the memory areas */
MEMORY
{
RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 320K
FLASH (rx)      : ORIGIN = 0x8000000, LENGTH = 1024K
}

/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >FLASH

  /* The program code and other data goes into FLASH */
  .text :
  {
    . = ALIGN(4);
    *(.text)           /* .text sections (code) */
    *(.text*)          /* .text* sections (code) */
    *(.glue_7)         /* glue arm to thumb code */
    *(.glue_7t)        /* glue thumb to arm code */
    *(.eh_frame)

    KEEP (*(.init))
    KEEP (*(.fini))

    . = ALIGN(4);
    _etext = .;        /* define a global symbols at end of code */
  } >FLASH

  /* used by the startup to initialize data */
  _sidata = LOADADDR(.data);

  /* Initialized data sections goes into RAM, load LMA copy after code */
  .data :
  {
    . = ALIGN(4);
    _sdata = .;        /* create a global symbol at data start */
    *(.data)           /* .data sections */
    *(.data*)          /* .data* sections */

    . = ALIGN(4);
    _edata = .;        /* define a global symbol at data end */
  } >RAM AT> FLASH


  /* Uninitialized data section */
  . = ALIGN(4);
  .bss :
  {
    /* This is used by the startup in order to initialize the .bss secion */
    _sbss = .;         /* define a global symbol at bss start */
    __bss_start__ = _sbss;
    *(.bss)
    *(.bss*)
    *(COMMON)

    . = ALIGN(4);
    _ebss = .;         /* define a global symbol at bss end */
    __bss_end__ = _ebss;
  } >RAM
}
```

startup_stm32f746xx.s:

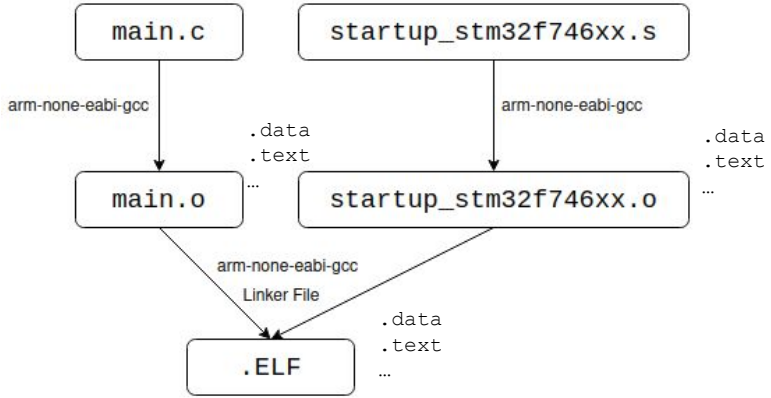
```
/*******************************************************************************
*
* The minimal vector table for a Cortex M7. Note that the proper constructs
* must be placed on this to ensure that it ends up at physical address
* 0x0000.0000.
*
*******************************************************************************/
   .section  .isr_vector,"a",%progbits
  .type  g_pfnVectors, %object
  .size  g_pfnVectors, .-g_pfnVectors


g_pfnVectors:
  .word  _estack
  .word  Reset_Handler

  .word  NMI_Handler
  .word  HardFault_Handler
  .word  MemManage_Handler
  .word  BusFault_Handler
  .word  UsageFault_Handler
  .word  0
  .word  0
  .word  0
  .word  0
  .word  SVC_Handler
  .word  DebugMon_Handler
  .word  0
  .word  PendSV_Handler
  .word  SysTick_Handler

```

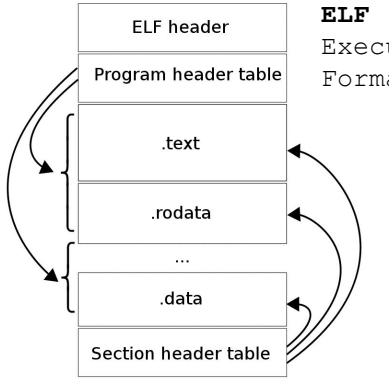



```
/**
 * @brief  This is the code that gets called when the processor first
 *          starts execution following a reset event. Only the absolutely
 *          necessary set is performed, after which the application
 *          supplied main() routine is called.
 * @param  None
 * @retval : None
*/

    .section  .text.Reset_Handler
  .weak  Reset_Handler
  .type  Reset_Handler, %function
Reset_Handler:
  ldr   sp, =_estack      /* set stack pointer */

/* Copy the data segment initializers from flash to SRAM */
  movs  r1, #0
  b  LoopCopyDataInit

CopyDataInit:
  ldr  r3, =_sidata
  ldr  r3, [r3, r1]
  str  r3, [r0, r1]
  adds  r1, r1, #4

LoopCopyDataInit:
  ldr  r0, =_sdata
  ldr  r3, =_edata
  adds  r2, r0, r1
  cmp  r2, r3
  bcc  CopyDataInit
  ldr  r2, =_sbss
  b  LoopFillZerobss
/* Zero fill the bss segment. */
FillZerobss:
  movs  r3, #0
  str  r3, [r2], #4

LoopFillZerobss:
  ldr  r3, = _ebss
  cmp  r2, r3
  bcc  FillZerobss

/* Call the clock system initialization function.*/
  bl  SystemInit
/* Call static constructors */
    bl __libc_init_array
/* Call the application's entry point.*/
  bl  main
  bx  lr
.size  Reset_Handler, .-Reset_Handler
```

https://developer.arm.com/documentation/dui0774/i/armclang-Integrated-Assembler-Directives/Section-directives

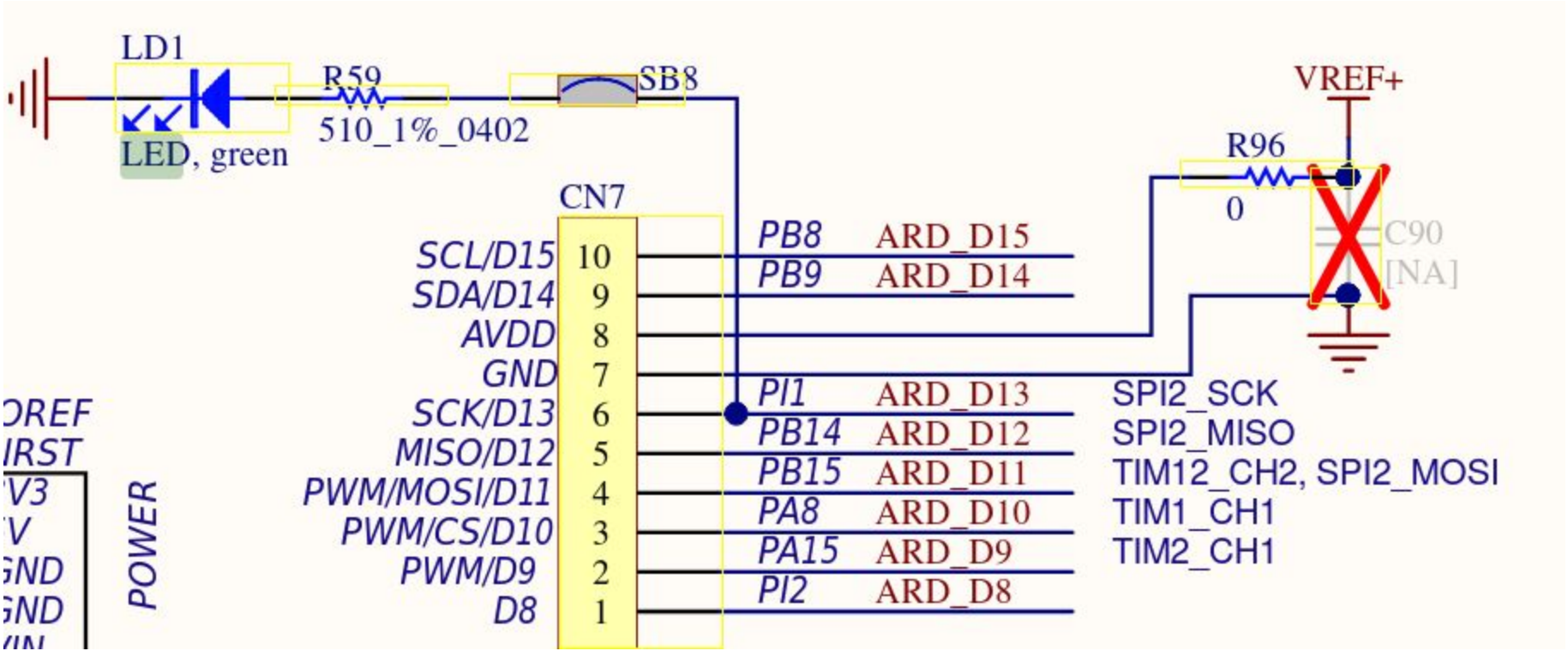
Src/main.c	Наша основная программа
startup_stm32f746xx.s	Описывает вектора прерывания, заполняет bss/data, конструкторы, прыгает в main
Inc/stm32f746xx.h	Define all stm32f746g registers
STM32F746NGHx_FLASH.ld	Linker script



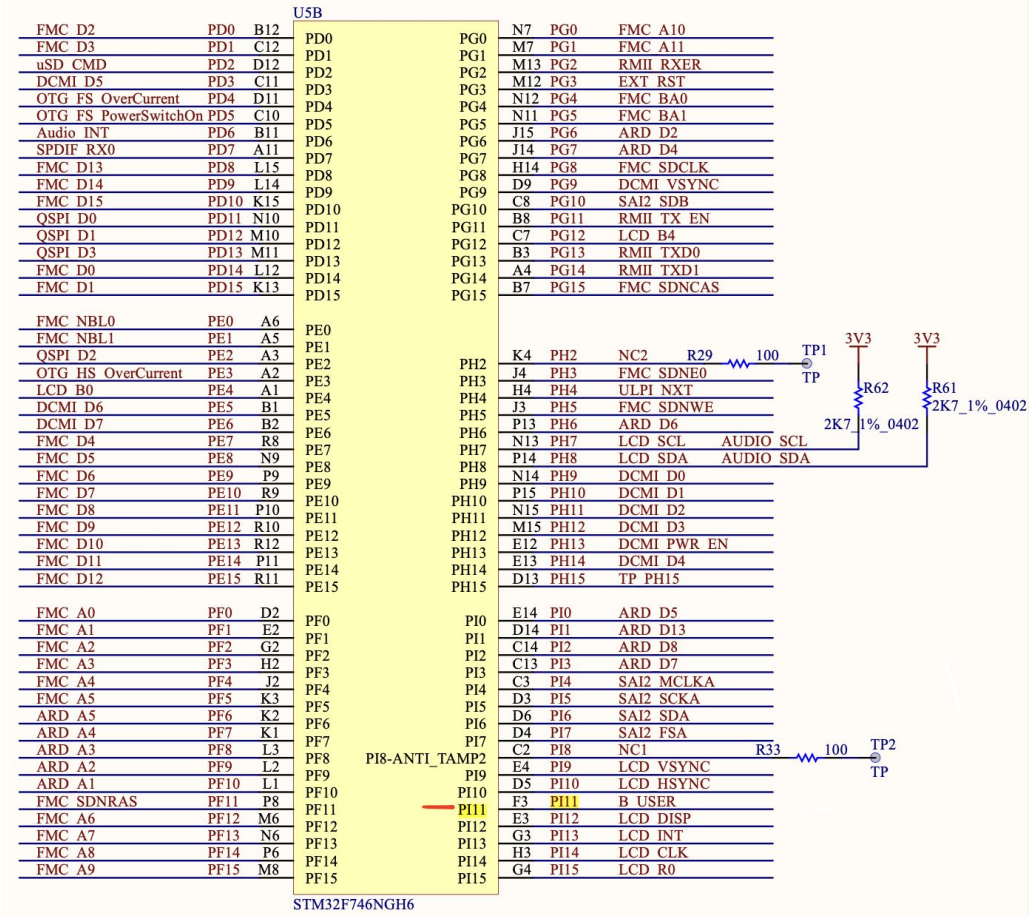


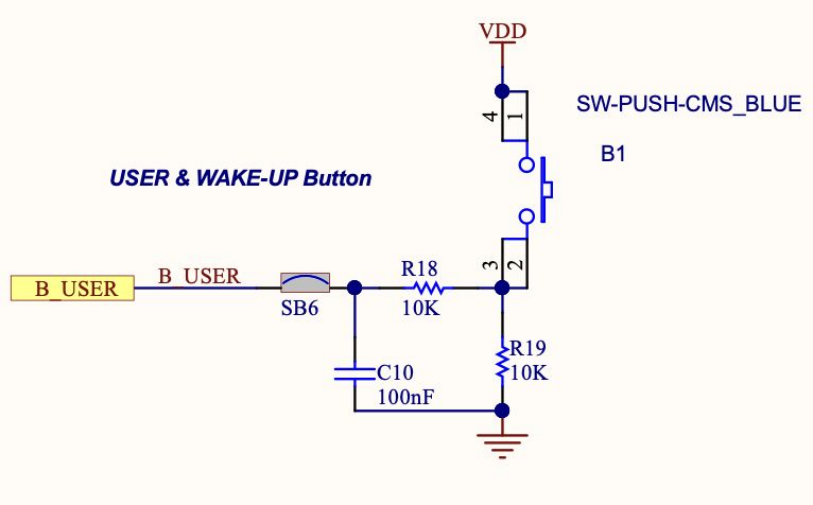
ELF
Executable and Linkable Format

```
arm-none-eabi-gcc -mcpu=cortex-m7 -mlittle-endian -mthumb -g -I./Inc -DSTM32F746xx -c ./Src/main.c -o ./Src/main.o

arm-none-eabi-gcc -mcpu=cortex-m7 -mlittle-endian -mthumb -g -I./Inc -c ./startup_stm32f746xx.s -o
./startup_stm32f746xx.o

arm-none-eabi-gcc -mcpu=cortex-m7 -mlittle-endian -mthumb -T./STM32F746NGHx_FLASH.ld -Wl,--gc-section ./Src/main.o
./startup_stm32f746xx.o -o main.elf
```

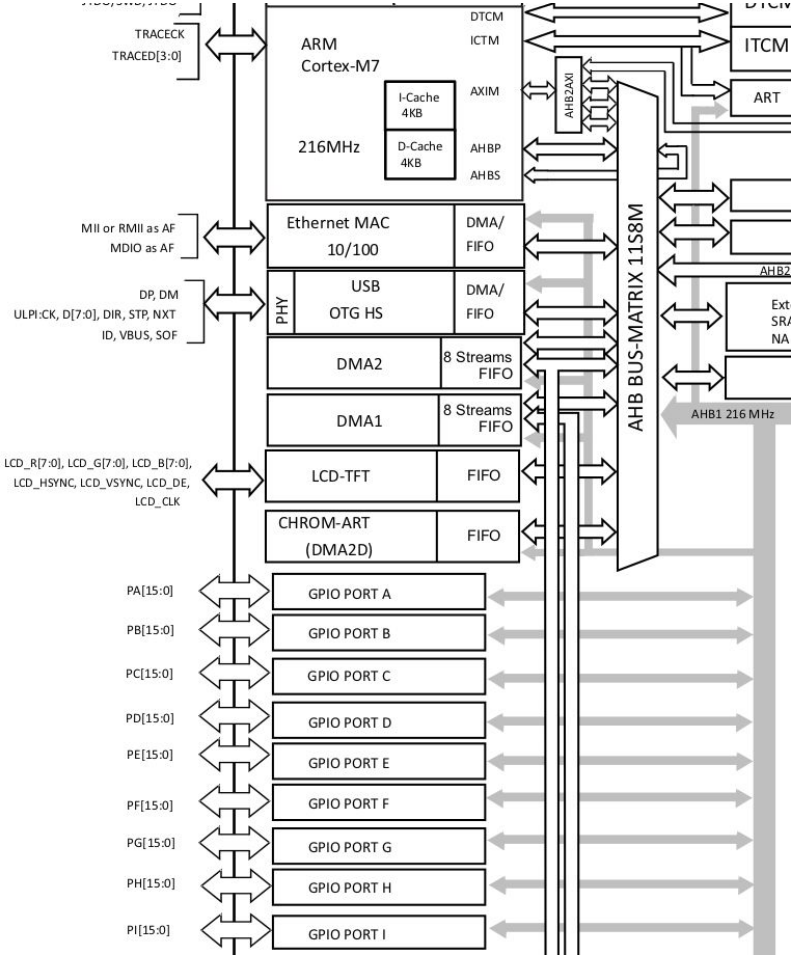


https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group1/ff/cd/ce/2d/f8/fb/40/69/mb1191-F746NGH6-C01_schematic/files/mb1191-F746NGH6-C01_schematic.pdf/jcr:content/translations/en.mb1191-F746NGH6-C01_schematic.pdf





https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group1/ff/cd/ce/2d/f8/fb/40/69/mb1191-F746NGH6-C01_schematic/files/mb1191-F746NGH6-C01_schematic.pdf/jcr:content/translations/en.mb1191-F746NGH6-C01_schematic.pdf



5.3.10 RCC AHB1 peripheral clock register (RCC_AHB1ENR)

Address offset: 0x30

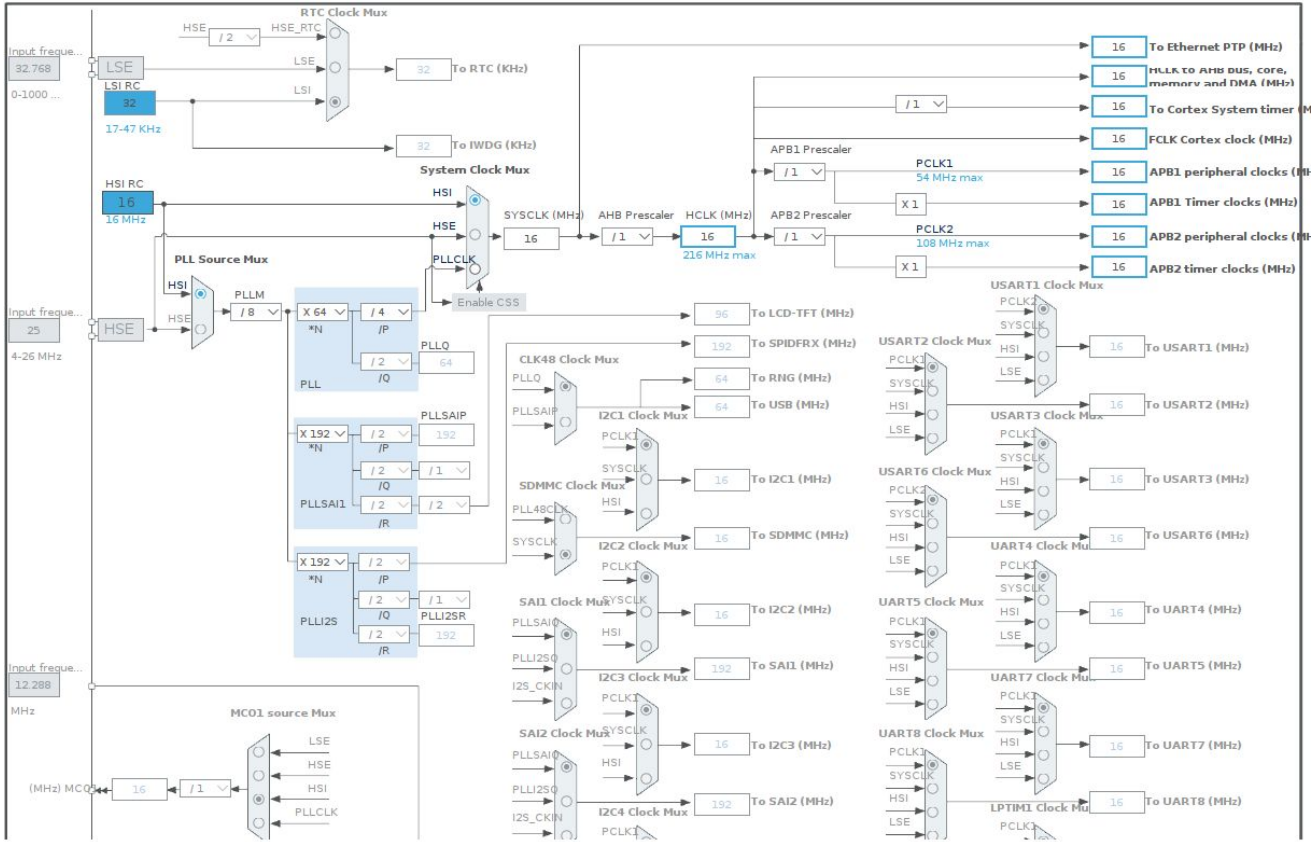
Reset value: 0x0010 0000

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
Res.	OTGHS ULPIEN	OTGHS EN	ETHM ACPTP EN	ETHM ACRX EN	ETHM ACTX EN	ETHMA CEN	Res.	DMA2D EN	DMA2 EN	DMA1 EN	DTCMRA MEN	Res.	BKPSR AMEN	Res.	Res.
	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
Res.	Res.	Res.	CRC EN	Res.	GPIOK EN	GPIOJ EN	GPIOI EN	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOIEN;
```

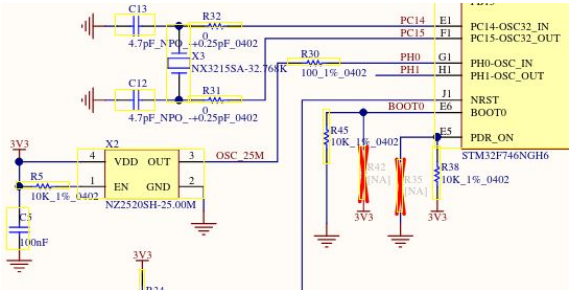

https://www.st.com/resource/en/datasheet/stm32f745ie.pdf page 16

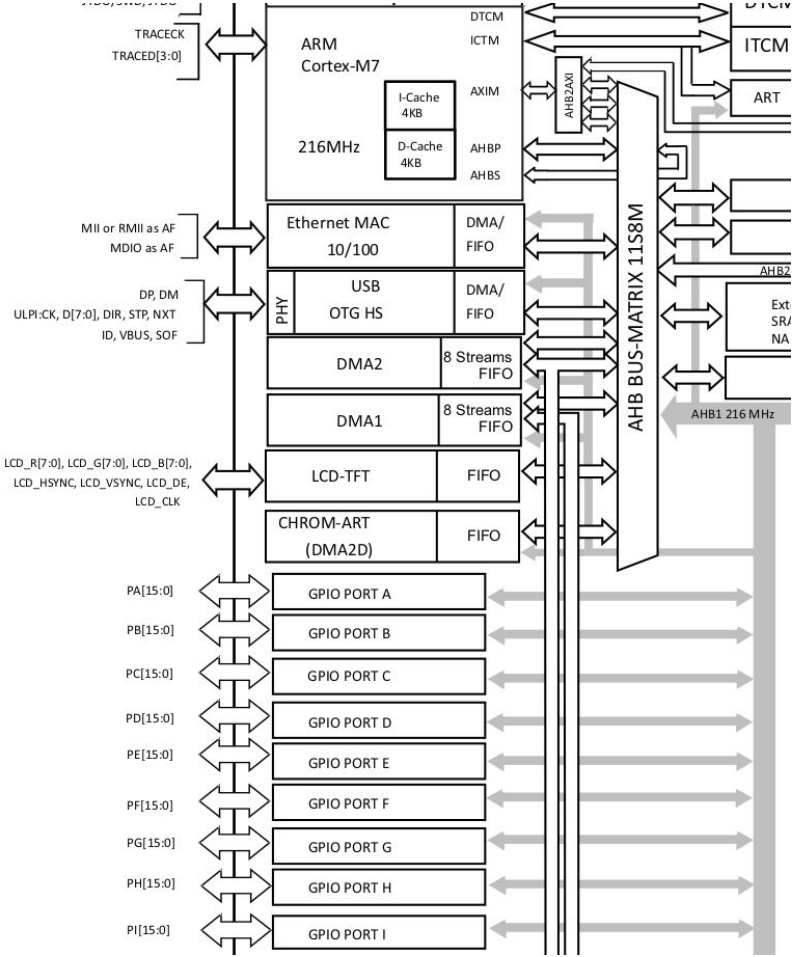


HSI High Speed Internal oscillator clock

HSE High Speed External oscillator clock

PLL Phase Locked Loop clock





5.3.10 RCC AHB1 peripheral clock register (RCC_AHB1ENR)

Address offset: 0x30

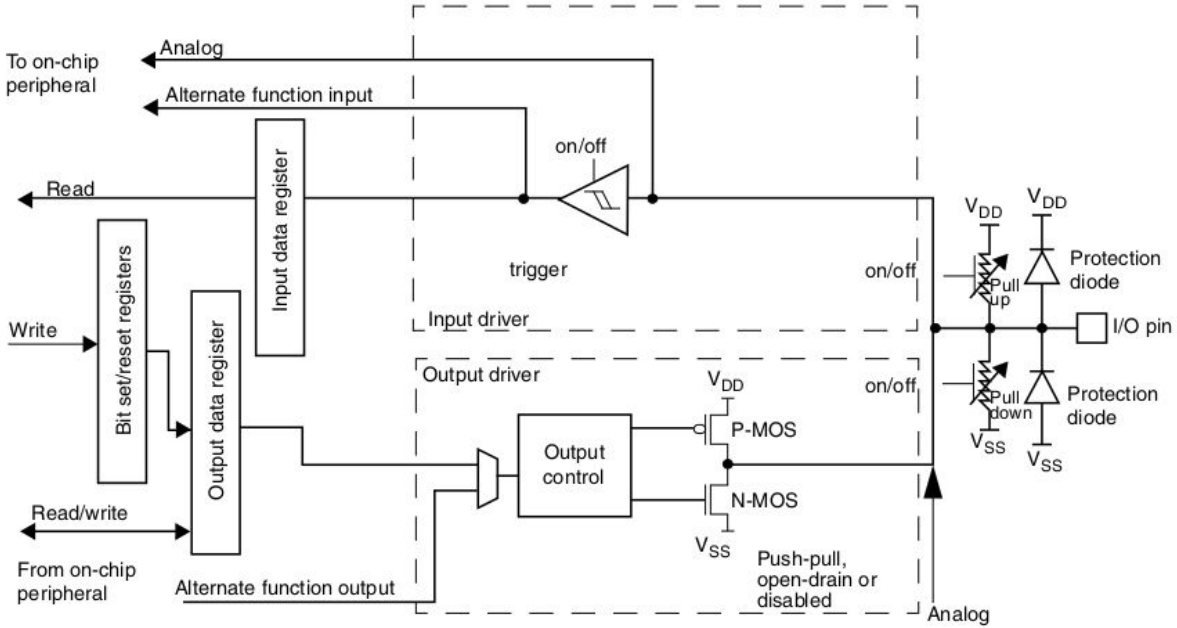
Reset value: 0x0010 0000

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
Res.	OTGHS ULPIEN	OTGHS EN	ETHM ACPTP EN	ETHM ACRX EN	ETHM ACTX EN	ETHMA CEN	Res.	DMA2D EN	DMA2 EN	DMA1 EN	DTCMRA MEN	Res.	BKPSR AMEN	Res.	Res.
	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
Res.	Res.	Res.	CRC EN	Res.	GPIOK EN	GPIOJ EN	GPIOI EN	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOIEN;
```


https://www.st.com/resource/en/datasheet/stm32f745ie.pdf page 16



GPIO MODES:

- Input Floating
- Input pull-up
- Input pull-down
- Analog
- Output open-drain
- Output push-pull
- Alternative functions (push-pull and open-drain): USART, SPI, I2C, Timer, LCD, USB, etc…

https://www.st.com/resource/en/reference_manual/rm0385-stm32f75xxx-and-stm32f74xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

GPIO Block Diagram

GPIOx_MODER:
00 - input mode (R)
01 - GP output mode
10 - Alternative func mode
11 - Analog mode

GPIOx_OTYPER:
0 - Output push-pull (R)
1 - Output open-drain

GPIOx_PUPDR:
00 - No Pull-up/down (R)
01 - Pull-up
10 - Pull-down

GPIOx_IDR:
IDR - input data I/O pin
GPIOx_ODR:
ODR - output data I/O pin

GPIOx_AFRL:
AFR - Alternate function for port0 - port7
example: 0000(R) - AF0, 0010 - AF2, 1111 - AF15

GPIOx_AFRH:
AFR - Alternate function for port8 - port15
example: 0000(R) - AF0, 0010 - AF2, 1111 - AF15

GPIOx_BSRR (Bits 15:0 - Set, Bits 31:16 - Reset):
Atomic set/reset output data I/O pin
To set Port A Bit 5 to a 1 you simply do GPIOA->BSRR = (1<<5)
Without the BSRR you would have to do GPIOA->ODR |= (1<<5)

```
//PI1 GP output mode
GPIOI->MODER |= (0b01<<2);

// set LED PI1
GPIOI->BSRR |= (1 << 1);

// Read Button PI11
while ((GPIOI->IDR & (1 << 11)) != (1 << 11));
```

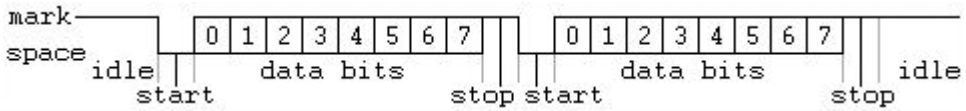
https://www.st.com/resource/en/reference_manual/rm0385-stm32f75xxx-and-stm32f74xx
x-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

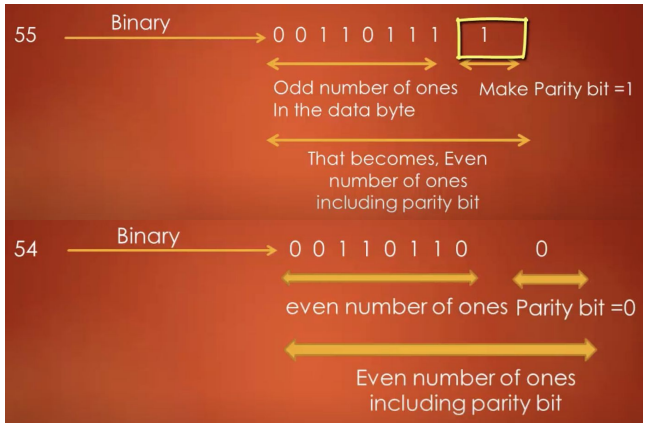
UART

Universal Asynchronous Receiver-Transmitter

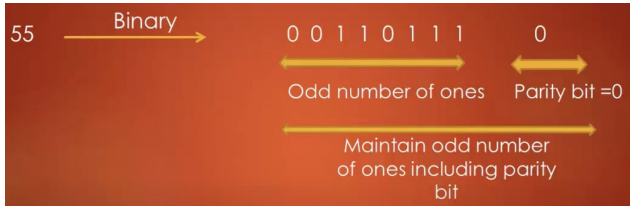
Receiver need to know baudrate of the transmitter before initiation of reception i.e. before communication to be established

It uses start bit (before data word), stop bits (one or two, after data word), parity bit (even or odd) in its base format for data formatting. Parity bit helps in one bit error detection.

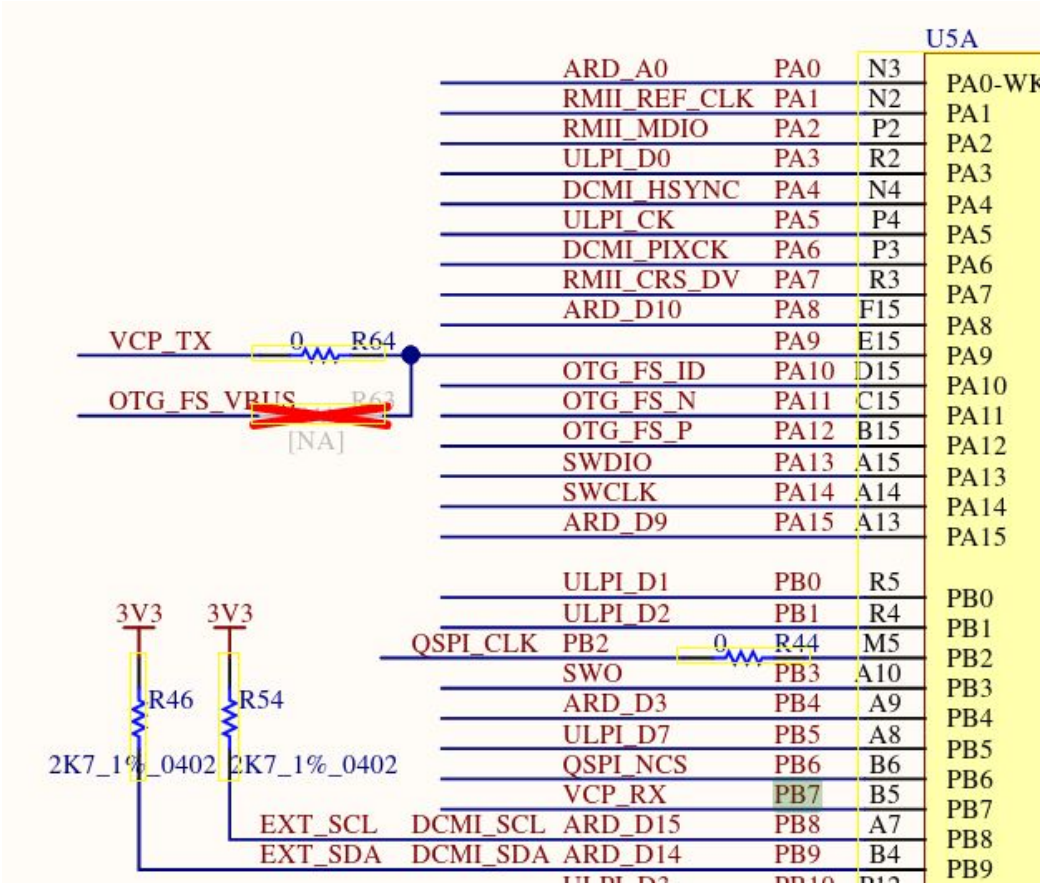




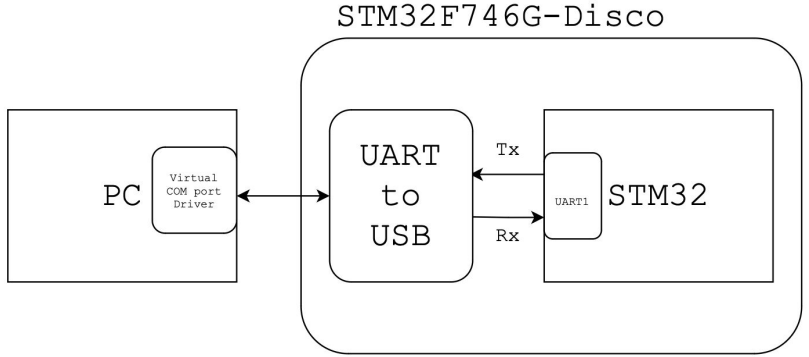
<- Even Parity



<- Odd Parity




```
// Enable Clock
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;

// Alternate Function Mode
GPIOA->MODER |= (0b10 << 9*2); // PA9
GPIOB->MODER |= (0b10 << 7*2); // PB7
```



https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group1/ff/cd/ce/2d/f8/fb/40/69/mb1191-F746NGH6-C01_schematic/files/mb1191-F746NGH6-C01_schematic.pdf/jcr:content/translations/en.mb1191-F746NGH6-C01_schematic.pdf

UART schematic
GPIO for UART

Table 12. STM32F745xx and STM32F746xx alternate function mapping

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11/LPTIM1/CEC	I2C1/2/3/4/CEC	SPI1/2/3/4/5/6	SPI3/SAI1	SPI2/3/USART1/2/3/UART5/SPDIFRX	SAI2/USART6/UART4/5/7/8/SPDIFRX	CAN1/2/TIM12/13/14/QUADSPI/LCD	SAI2/QUADSPI/OTG2_HS/OTG1_FS	ETH/OTG1_FS	FMC/SDMMC1/OTG2_FS	DCMI	LCD	SYS
Port A	PA0	-	TIM2_CH1/TIM2_ETR	TIM5_CH1	TIM8_ETR	-	-	-	USART2_CTS	UART4_TX	-	SAI2_SD_B	ETH_MII_CRS	-	-	-	EVENTOUT
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_RTS	UART4_RX	QUADSPI_BK1_IO3	SAI2_MCK_B	ETH_MII_RX_CLK/ETH_RMII_REF_CLK	-	-	LCD_R2	EVENTOUT
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_TX	SAI2_SCK_B	-	-	ETH_MDIO	-	-	LCD_R1	EVENTOUT
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-	-	-	USART2_RX	-	-	OTG_HS_ULPI_D0	ETH_MII_COL	-	-	LCD_B5	EVENTOUT
	PA4	-	-	-	-	-	SPI1_NSS/I2S1_WS	SPI3_NSS/I2S3_WS	USART2_CK	-	-	-	-	OTG_HS_SOF	DCMI_HSYNC	LCD_VSYNC	EVENTOUT
	PA5	-	TIM2_CH1/TIM2_ETR	-	TIM8_CH1N	-	SPI1_SCK/I2S1_CK	-	-	-	-	OTG_HS_ULPI_CK	-	-	-	LCD_R4	EVENTOUT
	PA6	-	TIM1_BKIN	TIM3_CH1	TIM8_BKIN	-	SPI1_MISO	-	-	-	TIM13_CH1	-	-	-	DCMI_PIXCLK	LCD_G2	EVENTOUT
	PA7	-	TIM1_CH1N	TIM3_CH2	TIM8_CH1N	-	SPI1_MOSI/I2S1_SD	-	-	-	TIM14_CH1	-	ETH_MII_RX_DV/ETH_RMII_CRS_DV	FMC_SDNWE	-	-	EVENTOUT
	PA8	MCO1	TIM1_CH1	-	TIM8_BKIN2	I2C3_SCL	-	-	USART1_CK	-	-	OTG_FS_SOF	-	-	-	LCD_R6	EVENTOUT
	PA9	-	TIM1_CH2	-	-	I2C3_SMBA	SPI2_SCK/I2S2_CK	-	USART1_TX	-	-	-	-	-	DCMI_D0	-	EVENTOUT
	PA10	-	TIM1_CH3	-	-	-	-	-	USART1_RX	-	-	OTG_FS_ID	-	-	DCMI_D1	-	EVENTOUT
	PA11	-	TIM1_CH4	-	-	-	-	-	USART1_CTS	-	CAN1_RX	OTG_FS_DM	-	-	-	LCD_R4	EVENTOUT

```
// <<4=PA9, 0b111=7=AF7
GPIOA->AFR[1] |= (0b111 << (4 * (9-8)));
```

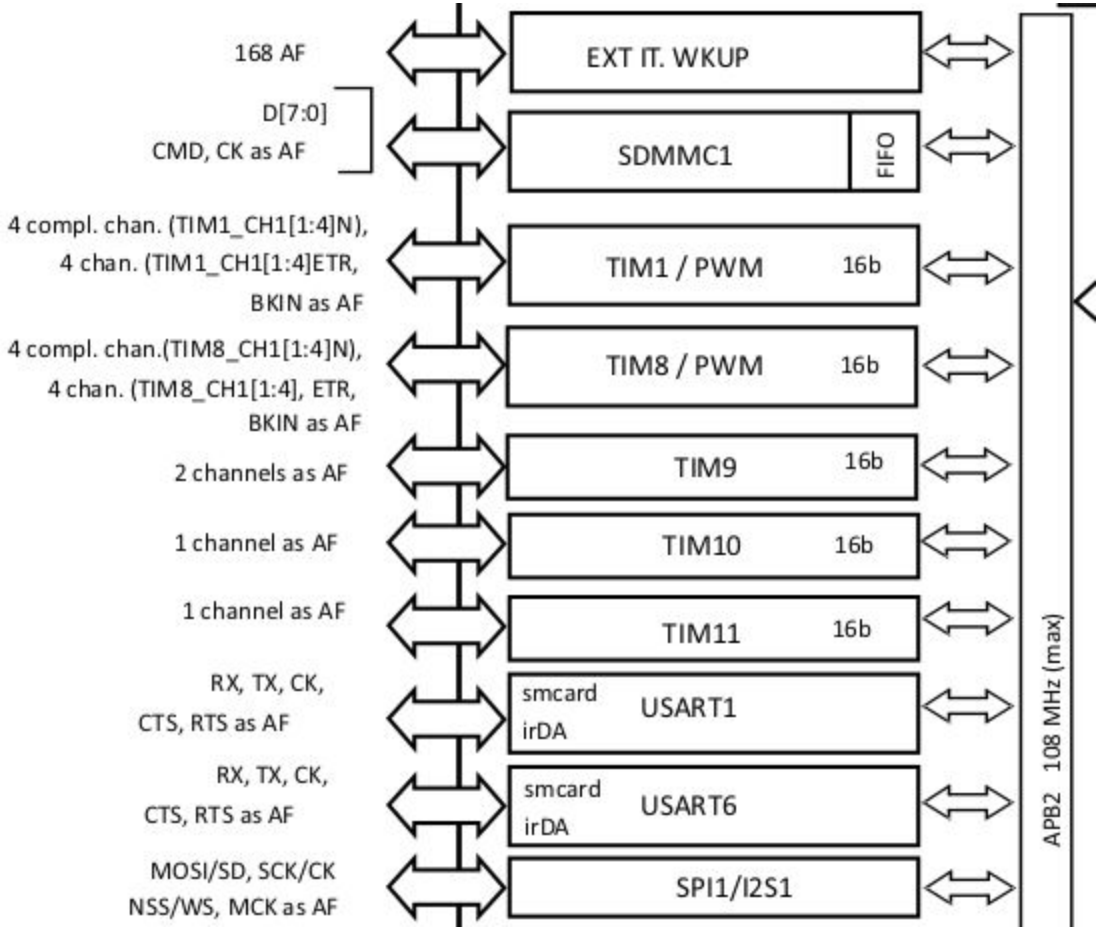
https://www.st.com/resource/en/datasheet/stm32f745ie.pdf page 76

Table 12. STM32F745xx and STM32F746xx alternate function mapping (continued)

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11/LPTIM1/CEC	I2C1/2/3/4/CEC	SPI1/2/3/4/5/6	SPI3/SAI1	SPI2/3/USART1/2/3/UART5/SPDIFRX	SAI2/USART6/UART4/5/7/8/SPDIFRX	CAN1/2/TIM12/13/14/QUADSPI/LCD	SAI2/QUADSPI/OTG2_HS/OTG1_FS	ETH/OTG1_FS	FMC/SDMMC1/OTG2_FS	DCMI	LCD	SYS
Port A	PA12	-	TIM1_ETR	-	-	-	-	-	USART1_RTS	SAI2_FS_B	CAN1_TX	OTG_FS_DP	-	-	-	LCD_R5	EVENTOUT
	PA13	JTMS-SWDIO	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENTOUT
	PA14	JTCK-SWCLK	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENTOUT
	PA15	JTDI	TIM2_CH1/TIM2_ETR	-	-	HDMI-CEC	SPI1_NSS/I2S1_WS	SPI3_NSS/I2S3_WS	-	UART4_RTS	-	-	-	-	-	-	EVENTOUT
Port B	PB0	-	TIM1_CH2N	TIM3_CH3	TIM8_CH2N	-	-	-	-	UART4_CTS	LCD_R3	OTG_HS_ULPI_D1	ETH_MII_RXD2	-	-	-	EVENTOUT
	PB1	-	TIM1_CH3N	TIM3_CH4	TIM8_CH3N	-	-	-	-	-	LCD_R6	OTG_HS_ULPI_D2	ETH_MII_RXD3	-	-	-	EVENTOUT
	PB2	-	-	-	-	-	-	SAI1_SD_A	SPI3_MOSI/I2S3_SD		QUADSPI_CLK	-	-	-	-	-	EVENTOUT
	PB3	JTDO/TRACESWO	TIM2_CH2	-	-	-	SPI1_SCK/I2S1_CK	SPI3_SCK/I2S3_CK	-	-	-	-	-	-	-	-	EVENTOUT
	PB4	NJTRST	-	TIM3_CH1	-	-	SPI1_MISO	SPI3_MISO	SPI2_NSS/I2S2_WS	-	-	-	-	-	-	-	EVENTOUT
	PB5	-	-	TIM3_CH2	-	I2C1_SMBA	SPI1_MOSI/I2S1_SD	SPI3_MOSI/I2S3_SD	-	-	CAN2_RX	OTG_HS_ULPI_D7	ETH_PPS_OUT	FMC_SDCKE1	DCMI_D10	-	EVENTOUT
	PB6	-	-	TIM4_CH1	HDMI-CEC	I2C1_SCL	-	-	USART1_TX	-	CAN2_TX	QUADSPI_BK1_NCS	-	FMC_SDNE1	DCMI_D5	-	EVENTOUT
	PB7	-	-	TIM4_CH2	-	I2C1_SDA	-	-	USART1_RX	-	-	-	-	FMC_NL	DCMI_VSYNC	-	EVENTOUT
	PB8	-	-	TIM4_CH3	TIM10_CH1	I2C1_SCL	-	-	-	-	CAN1_RX		ETH_MII_TXD3	SDMMC1_D4	DCMI_D6	LCD_B6	EVENTOUT

```
// <<28=PB7, 0b111=7=AF7
GPIOB->AFR[0] |= (0b111 << (4 * (7-0)));
```

https://www.st.com/resource/en/datasheet/stm32f745ie.pdf page 77




```
// Enable Clock
RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
```

USART_CR1:

```
M0M1: [00] 1 Start bit, 8 data bits, n stop bits [R]
M0M1: [01] 1 Start bit, 9 data bits, n stop bits
M0M1: [10] 1 Start bit, 7 data bits, n stop bits
PCE:  [0]  Parity Control Disable [R]
PCE:  [1]  Parity Control Enable
PS:   [0]  Even Parity [R]
PS:   [1]  Odd Parity
TE:   [0]  Transmitter Disable [R]
TE:   [1]  Transmitter Enable
RE:   [0]  Receiver Disable [R]
RE:   [1]  Receiver Enable
UE:   [0]  USART Disable [R]
UE:   [1]  USART Enable
```

USART_CR2:

```
MSBFIRST [0]: Less Significant bit first (0bit is first) [R]
MSBFIRST [1]: Most Significant bit first (bit 7/8/9 first)
SWAP     [0]: Tx/Rx as default [R]
SWAP     [1]: Tx/Rx are swapped
STOP    [00]: 1 stop bits [R]
STOP    [01]: 0.5 stop bits
STOP    [10]: 2 stop bits
STOP    [11]: 1.5 stop bits
```

https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group1/ff/cd/ce/2d/f8/fb/40/69/mb1191-F746NGH6-C01_schematic/files/mb1191-F746NGH6-C01_schematic.pdf/jcr:content/translations/en.mb1191-F746NGH6-C01_schematic.pdf

USART_BRR:

USARTDIV: Baudrate divisor register:

In case of oversampling by 16, the equation is:

$$\text{Tx/Rx baud} = \frac{f_{CK}}{\text{USARTDIV}}$$

In case of oversampling by 8, the equation is:

$$\text{Tx/Rx baud} = \frac{2 \times f_{CK}}{\text{USARTDIV}}$$

f_{CK} can be f_{LSE}, f_{HSI}, f_{PCLK}, f_{SYS}.

USART_RDR:

RDR: Contains the received data character

USART_TDR:

TDR: Contains the data character to be transmitted

USART_ISR:

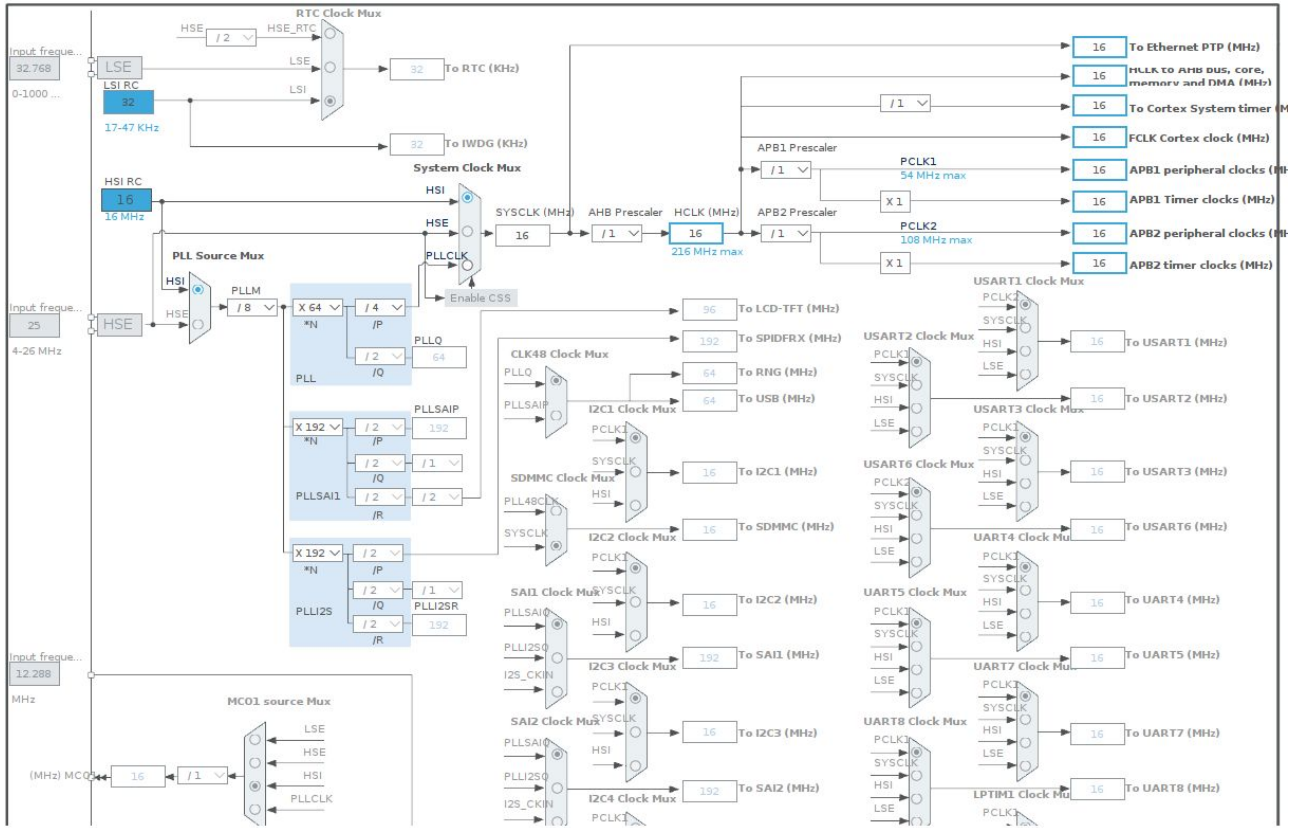
RXNE: [1] Received data is ready to be read

TXE: [1] Data is transferred to the shift register. It's cleared by the writing in **USART_TDR** register.

oversampling is 16 by default, APB2 clock freq = 16MHz (from HSI and without prescallers), so:

USARTDIV (for 9600 baudrate) = 16 000 000 / 9600 = 1667

```
USART1->BRR = 1667;
USART1->CR1 |= USART_CR1_TE;
USART1->CR1 |= USART_CR1_UE;

while (1) {
      USART1->TDR = '3';
      while (!(USART1->ISR & USART_ISR_TXE));
}
```



HSI High Speed Internal oscillator clock

HSE High Speed External oscillator clock

PLL Phase Locked Loop clock

RCC_CFGR:

```
SWS [00]: HSI oscillator used as the system clock
SWS [01]: HSE oscillator used as the system clock
SWS [10]: PLL is used as the system clock

HPRE [0000]: System clocks not divided for AHB clocks
HPRE [1000]: System clocks divided by 2 for AHB clocks
HPRE [1001]: System clocks divided by 4 for AHB clocks
HPRE [1010]: System clocks divided by 8 for AHB clocks
...
HPRE [1111]: System clocks divided by 512 for AHB clocks

PPRE2: [000]: AHB not divided for APB2 clocks
PPRE2: [100]: AHB divided by 2 for APB2 clocks
...
PPRE2: [111]: AHB divided by 16 for APB2 clocks

PPRE1: [000]: AHB not divided for APB1 clocks
PPRE1: [100]: AHB divided by 2 for APB1 clocks
...
PPRE1: [111]: AHB divided by 16 for APB1 clocks

SW[00]: Select HSI oscillator as system clock
SW[01]: Select HSE oscillator as system clock
SW[10]: Select PLL oscillator as system clock
```

RCC_CR:

```
HSION  [0/1]: Off/On HSI
HSIRDY [0/1]: HSI notready/ready
HSEON  [0/1]: Off/On HSE
HSERDY [0/1]: HSE notready/ready
PLLON  [0/1]: Off/On PLL
PLLRDY [0/1]: PLL notready/ready
```

```
RCC->CR |= RCC CR HSEON;
while ((RCC->CR & RCC_CR_HSERDY) != RCC_CR_HSERDY);

RCC->CFGR |= RCC CFGR SW HSE;
while ((RCC->CFGR & RCC_CFGR_SWS_HSE) == 0);

uint32 t sws = 0, hpre = 0, ppre2 = 0, ppre1 = 0;
sws = (RCC->CFGR & RCC CFGR SWS) >>
                                    RCC_CFGR_SWS_Pos;
hpre = (RCC->CFGR & RCC CFGR HPRE) >>
                                    RCC CFGR_HPRE_Pos;
ppre2 = (RCC->CFGR & RCC CFGR PPRE2) >>
                                    RCC CFGR_PPRE2_Pos;
ppre1 = (RCC->CFGR & RCC CFGR PPRE1) >>
                                    RCC_CFGR_PPRE1_Pos;
```

https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group1/ff/cd/ce/2d/f8/fb/40/69/mb1191-F746NGH6-C01_schematic/files/mb1191-F746NGH6-C01_schematic.pdf/jcr:content/translations/en.mb1191-F746NGH6-C01_schematic.pdf

Что вынести из этой лекции:

- Reset signal
- Как стартует процессор
- Основы Linker скрипта и как собирается elf
- Clocks, GPIO, UART registers

Задание:

- Повторить все шаги для настройки (через регистры) UART и передачи данных по нему; залить на github чтобы я мог проверить;
  Вы можете использовать мою заготовку с пустым проектом:
  https://github.com/badembed/STM32F746GDisco-Empty

Задание повышенной сложности:

- Написать код для настройки uart6 или uart7 (не используя библиотек - только регистры) - см. serial6 и serial7 на плате->
  Вы можете перемкнуть serial6 Rx и Serial6 Tx для того чтобы передавать и принимать в тот же самый uart;
  Залить на github чтобы я мог проверить;

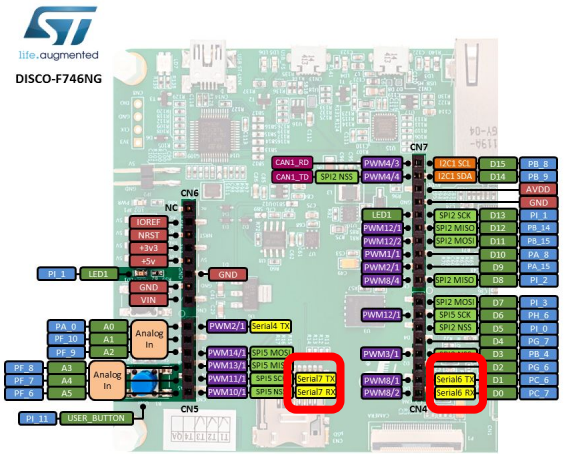
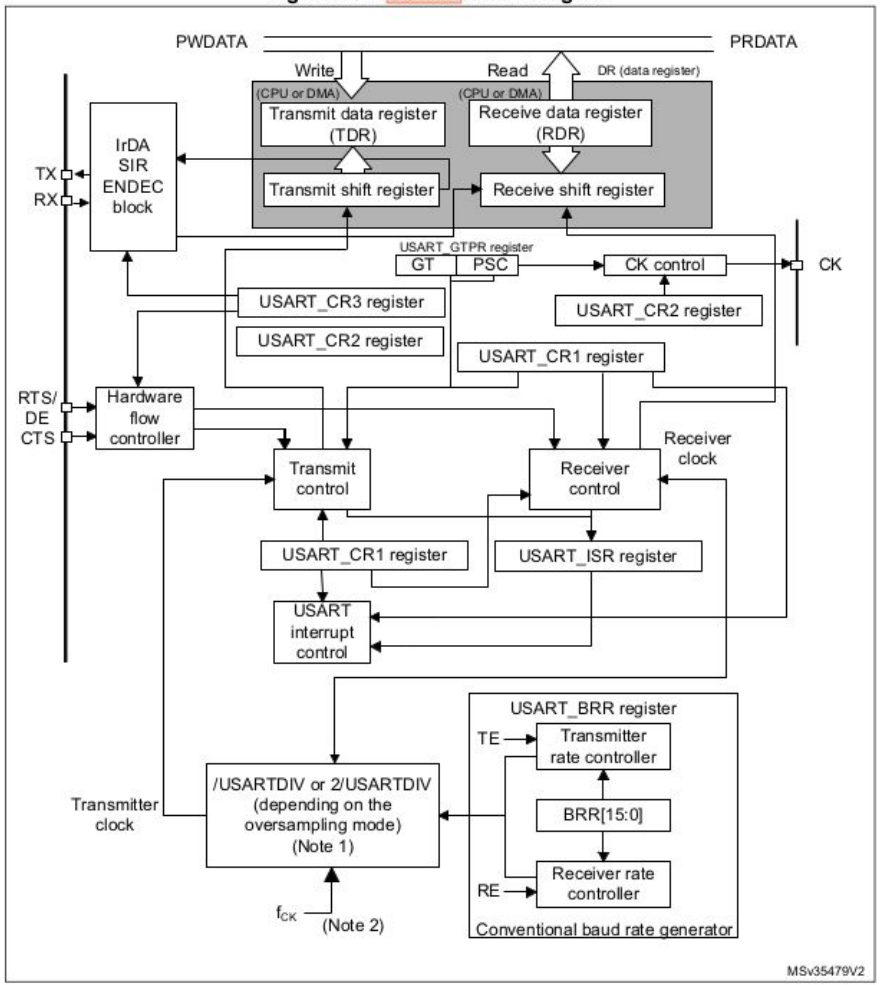


Figure 332. USART block diagram

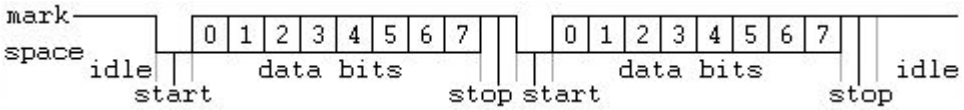


UART

Universal Asynchronous Receiver-Transmitter

Receiver need to know baudrate of the transmitter before initiation of reception i.e. before communication to be established

It uses start bit (before data word), stop bits (one or two, after data word), parity bit (even or odd) in its base format for data formatting. Parity bit helps in one bit error detection.



USART

Universal Synchronous Asynchronous Receiver Transmitter

It generates clocked data (i.e. synchronous clock generation), hence it supports higher data rate.

Receiver need not be required to know the baudrate of the transmitter. This is derived from the clock signal and data line.

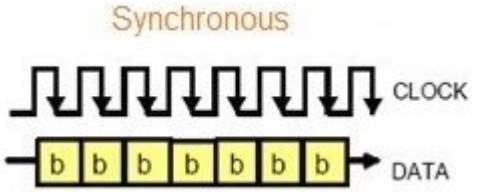


Figure 337. Data sampling when oversampling by 16