Clock Configuration and Timers

Lecture 6

Microprocessor-based Systems (E155)

Prof. Josh Brake

Updated Course Schedule

Week	Monday Lecture	Wednesday Lecture	Due
8/24	Intro and Overview	C Programming	(Lab Demos)
8/31	Toolchains	ARM Assembly Programming	Lab 1 - GPIO Blink and GNU Toolchain
9/7	STM32 Datasheet	Clock Configuration and Timers	Lab 2 - ARM Assembly Sort
9/14	Common Digital Structures	Serial Interfaces - Pt. 1	Lab 3 - Digital Audio
9/21	Serial Interfaces - Pt. 2	Analog to Digital and Back Again	Lab 4 - Serial Peripheral Interface (SPI)
9/28	ARM CMSIS	Interrupts	Lab 5 - Pulse-width Modulation (PWM)
10/5	Project Kickoff	Digital Signal Processing	Lab 6 - Serial Temperature Sensor
10/12	How To Pick a MCU	Custom Board Bringup	Lab 7 - The Internet of Things
10/19	PCB Design	Motors and Speakers	Project Proposal
10/26	Graphics and Displays	Bootloaders	Proposal Debriefs
11/2	Digital Business	Advanced MCU Topics	
11/9	Patents and Intellectual Property	TBD	Project Status Report & Demo
11/16	TBD	Presentations	
11/23	Presentations	**Happy Thanksgiving! No class**	
11/30	Presentations	Presentations	Project Checkoffs & Final Demos Due

Outline

- Review of bitfield structures
- Clocks
  - Configuring the clock tree
  - PLL configuration
  - Related considerations
- Timers
  - What is available on our MCU
  - What are they made of?
  - How do we use them?

Bitfield structures

- A convenient way to interact with bits inside individual registers.
- Enables data that doesn’t require a whole byte to be efficiently stored.

```
typedef struct {
volatile uint32_t <name1>  : width_in_bits;
volatile uint32_t <name2>  : width_in_bits;
volatile uint32_t <name3>  : width_in_bits;
...
} <structure_name>;
```

* Compliers and different target platforms may cause differences in how the elements are ordered (e.g., low to high or high to low). For our purposes, GCC for the STM32F401RE orders them low to high.

Bitfield struct example

```
struct mybitfields {
unsigned short a : 4;
unsigned short b : 5;
unsigned short c : 7;
} test;

int main(void) {
    test.a = 2;
    test.b = 31;
    test.c = 0;
}
```

```
0000000111110010
cccccccbbbbbaaaa
```

Bitfield example: Enabling GPIOC Clock

```
typedef struct {
volatile uint32_t GPIOAEN : 1;
volatile uint32_t GPIOBEN : 1;
volatile uint32_t GPIOCEN : 1;
volatile uint32_t GPIODEN : 1;
volatile uint32_t GPIOEEN : 1;
volatile uint32_t         : 2;
volatile uint32_t GPIOHEN : 1;
volatile uint32_t         : 4;
volatile uint32_t CRCEN   : 1;
volatile uint32_t         : 3;
volatile uint32_t         : 5;
volatile uint32_t DMA1EN  : 1;
volatile uint32_t DMA2EN  : 1;
volatile uint32_t         : 9;
} AHB1ENR_bits;
```

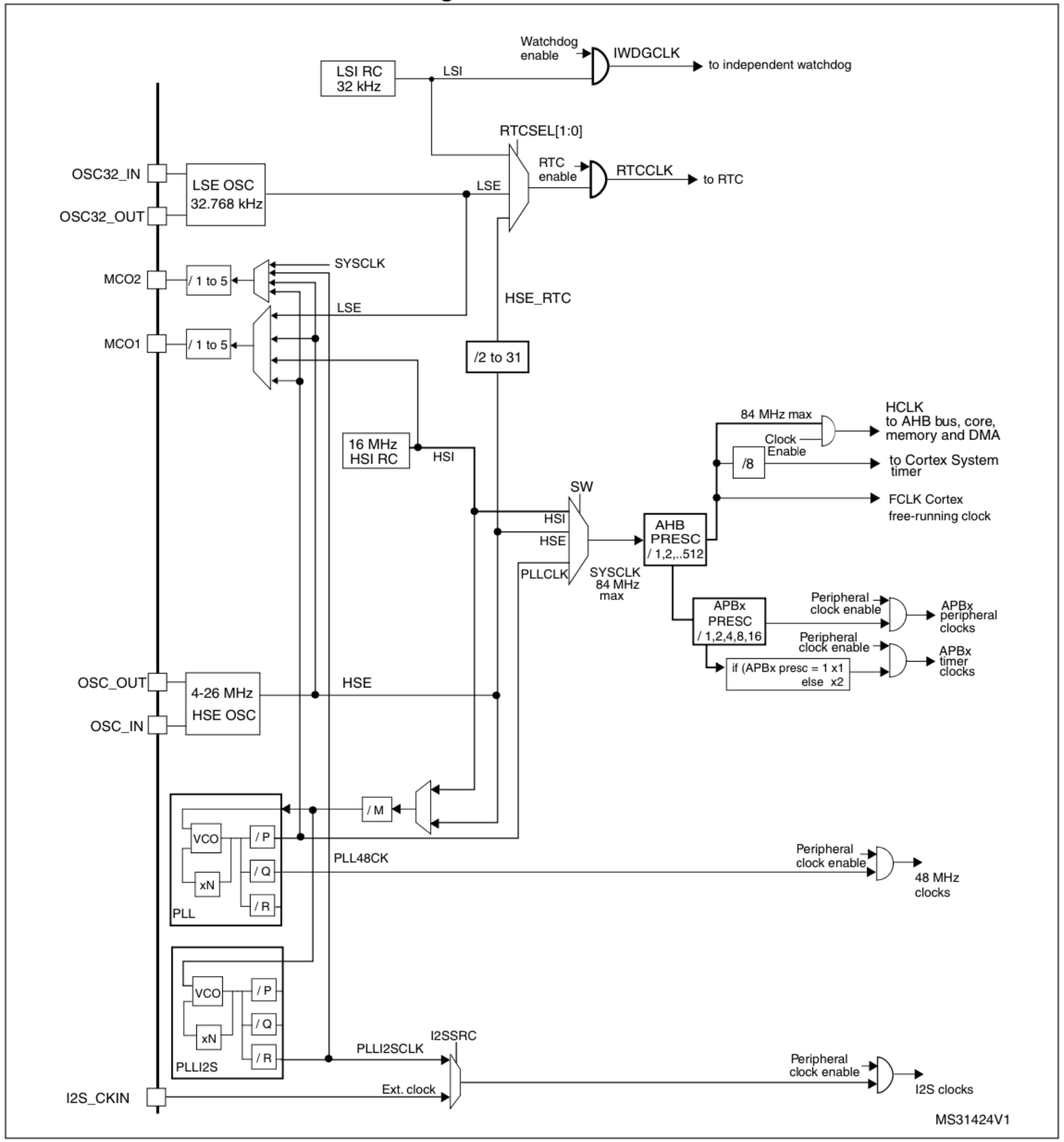
```
#define RCC_BASE (0x40023800UL) // base address of RCC
#define AHB1ENR ((AHB1ENR_bits *) (RCC_BASE + 0x30))

*AHB1ENR |= (1 << 2); // bit shifting way
AHB1ENR->GPIOCEN = 1; // using bitfield structure
```

Clock Configuration

Sources and generation with phase locked loops (PLLs)

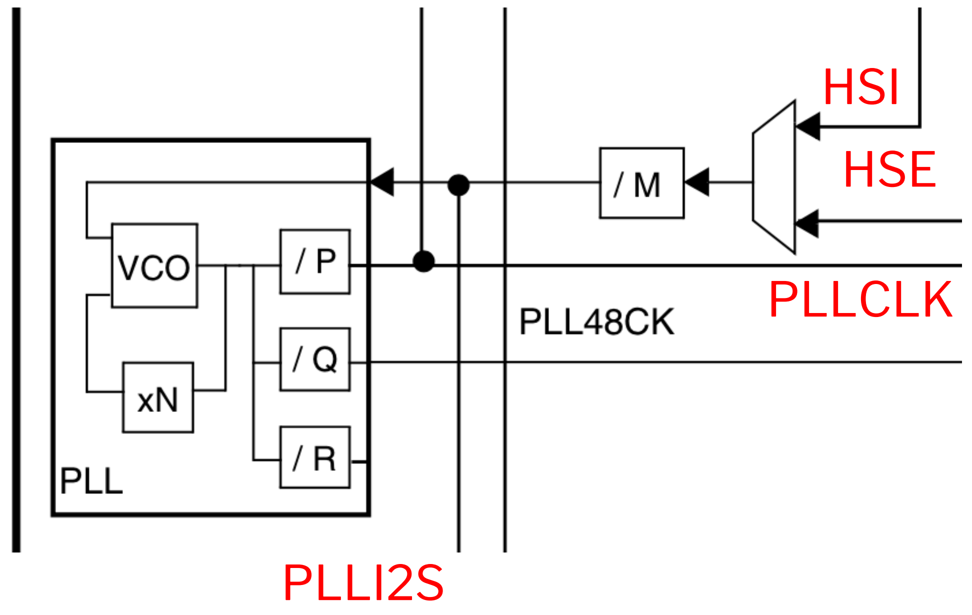
Clock Tree

Figure 12. Clock tree

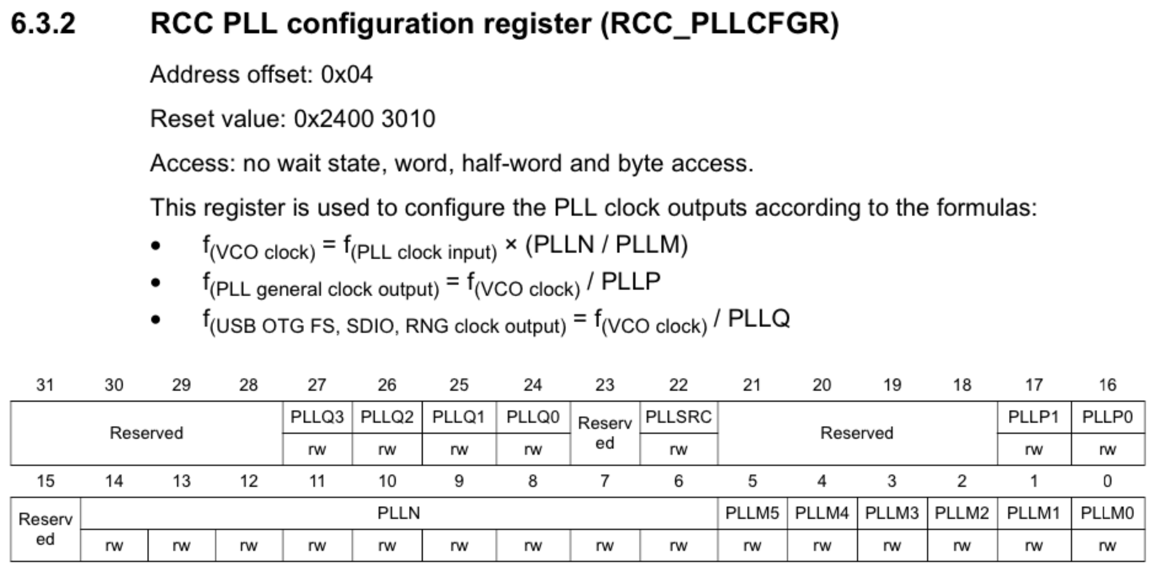


Configuring the PLL

- Configure input clock (HSI/HSE)
- Set main divisor M
- Set multiplier N
- Set divisor P
- Turn on PLL



STM32F401RE RM p.94, Figure 12. Clock tree



6.3.2 RCC PLL configuration register (RCC_PLLCFGR)

Address offset: 0x04

Reset value: 0x2400 3010

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

This register is used to configure the PLL clock outputs according to the formulas:

- $f_{(VCO\ clock)} = f_{(PLL\ clock\ input)} \times (PLLN / PLLM)$
- $f_{(PLL\ general\ clock\ output)} = f_{(VCO\ clock)} / PLLP$
- $f_{(USB\ OTG\ FS,\ SDIO,\ RNG\ clock\ output)} = f_{(VCO\ clock)} / PLLQ$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLLQ3	PLLQ2	PLLQ1	PLLQ0	Reserved	PLLSRC	Reserved				PLLP1	PLLP0
				rw	rw	rw	rw		rw					rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved	PLLN									PLLM5	PLLM4	PLLM3	PLLM2	PLLM1	PLLM0
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Configuring the PLL: Rules for M, N, and P

6.3.2 RCC PLL configuration register (RCC_PLLCFGR)

Address offset: 0x04

Reset value: 0x2400 3010

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

This register is used to configure the PLL clock outputs according to the formulas:

- $f_{\text{(VCO clock)}} = f_{\text{(PLL clock input)}} \times (\text{PLLN} / \text{PLLM})$
- $f_{\text{(PLL general clock output)}} = f_{\text{(VCO clock)}} / \text{PLLP}$
- $f_{\text{(USB OTG FS, SDIO, RNG clock output)}} = f_{\text{(VCO clock)}} / \text{PLLQ}$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLLQ3	PLLQ2	PLLQ1	PLLQ0	Reserved	PLLSRC	Reserved				PLLP1	PLLP0
				rw	rw	rw	rw		rw					rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	PLLN									PLLM5	PLLM4	PLLM3	PLLM2	PLLM1	PLLM0
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 17:16 **PLLP:** Main PLL (PLL) division factor for main system clock

Set and cleared by software to control the frequency of the general PLL output clock. These bits can be written only if PLL is disabled.

Caution: The software has to set these bits correctly not to exceed 84 MHz on this domain.

PLL output clock frequency = VCO frequency / PLLP with PLLP = 2, 4, 6, or 8

00: PLLP = 2
01: PLLP = 4
10: PLLP = 6
11: PLLP = 8

Bits 5:0 **PLLM:** Division factor for the main PLL (PLL) and audio PLL (PLLI2S) input clock

Set and cleared by software to divide the PLL and PLLI2S input clock before the VCO. These bits can be written only when the PLL and PLLI2S are disabled.

Caution: The software has to set these bits correctly to ensure that the VCO input frequency ranges from 1 to 2 MHz. It is recommended to select a frequency of 2 MHz to limit PLL jitter.

VCO input frequency = PLL input clock frequency / PLLM with $2 \leq \text{PLLM} \leq 63$

000000: PLLM = 0, wrong configuration
000001: PLLM = 1, wrong configuration
000010: PLLM = 2
000011: PLLM = 3
000100: PLLM = 4
...
111110: PLLM = 62
111111: PLLM = 63

Bits 14:6 **PLLN:** Main PLL (PLL) multiplication factor for VCO

Set and cleared by software to control the multiplication factor of the VCO. These bits can be written only when PLL is disabled. Only half-word and word accesses are allowed to write these bits.

Caution: The software has to set these bits correctly to ensure that the VCO output frequency is between 192 and 432 MHz. (check also *Section 6.3.20: RCC PLLI2S configuration register (RCC_PLLI2SCFGR)*)

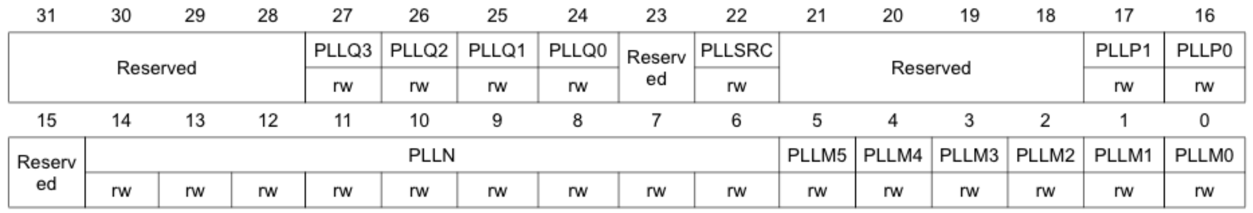
VCO output frequency = VCO input frequency × PLLN with $192 \leq \text{PLLN} \leq 432$

000000000: PLLN = 0, wrong configuration
000000001: PLLN = 1, wrong configuration
...
...
110110000: PLLN = 432
110110001: PLLN = 433, wrong configuration
...
111111111: PLLN = 511, wrong configuration

Bitfield structures: RCC Example

```
typedef struct {
volatile uint32_t PLLM     : 6;
volatile uint32_t PLLN     : 9;
volatile uint32_t          : 1;
volatile uint32_t PLLP     : 2;
volatile uint32_t          : 4;
volatile uint32_t PLLSRC   : 1;
volatile uint32_t          : 1;
volatile uint32_t PLLQ     : 4;
volatile uint16_t          : 4;
} RCC_PLLCFGR_bits;
```

```
typedef struct {
volatile uint32_t          RCC_CR;
volatile RCC_PLLCFGR_bits  RCC_PLLCFGR;
volatile uint32_t          RCC_CFGR;
...
} RCC;
```



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLLQ3	PLLQ2	PLLQ1	PLLQ0	Reserv ed	PLLSRC	Reserved				PLLP1	PLLP0
				rw	rw	rw	rw		rw					rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserv ed	PLLN									PLLM5	PLLM4	PLLM3	PLLM2	PLLM1	PLLM0
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

p. 105 STM32F401RE User Manual

```
#define RCC_BASE (0x40023800UL)        // base address of RCC
#define RCC_PTR ((RCC *) RCC_BASE)     // Pointer to RCC block of memory

RCC_PTR->RCC_PLLCFGR.PLLM = 10;        // Set 6-bit-wide PLLM field to 10
```

Activity: PLL Configuration

Reminder: Some helpful terminal commands

- In your host operating system – run docker container and mount present working directory (pwd) to /usr/project/src
  - MacOS/Linux: `docker run --rm -i -v $(pwd):/usr/project/src -t arm-gcc:10.1`
  - Windows: `docker run --rm -i -v $(pwd):/usr/project/src -t arm-gcc:10.1`
- For use in your Docker container:
  - Check out `gdb.sh` (may have to run `chmod +x gdh.sh` to make it executable. Then run `./gdb.sh` at your terminal to run it.)

```
#/bin/bash

arm-none-eabi-gdb -tui \
 --eval-command="target extended-remote host.docker.internal:3333” \
 -ex "load" -ex "layout asm" -ex "layout reg" blink.elf
```

STM32F401RE_RCC.h

Example Bitfield Struct

```
typedef struct {
volatile uint32_t GPIOAEN : 1;
volatile uint32_t GPIOBEN : 1;
volatile uint32_t GPIOCEN : 1;
volatile uint32_t GPIODEN : 1;
volatile uint32_t GPIOEEN : 1;
volatile uint32_t : 2;
volatile uint32_t GPIOHEN : 1;
volatile uint32_t : 4;
volatile uint32_t CRCEN : 1;
volatile uint32_t : 3;
volatile uint32_t : 5;
volatile uint32_t DMA1EN : 1;
volatile uint32_t DMA2EN : 1;
volatile uint32_t : 9;
} AHB1ENR_bits;
```

RCC Bitfield Struct Templates

```
typedef struct {
volatile uint32_t HSION : 1;
// TODO: Finish filling this out
} CR_bits;

typedef struct {
volatile uint32_t PLLM : 6;
// TODO: Finish filling this out
} PLLCFGR_bits;

typedef struct {
volatile uint32_t SW : 2;
// TODO: Finish filling this out
} CFGR_bits;
```

RCC Register Struct

```
typedef struct {
__IO CR_bits CR; /*!< RCC clock control register, Address offset: 0x00 */
__IO PLLCFGR_bits PLLCFGR; /*!< RCC PLL configuration register, Address offset: 0x04 */
__IO CFGR_bits CFGR; /*!< RCC clock configuration register, Address offset: 0x08 */
__IO uint32_t CIR; /*!< RCC clock interrupt register, Address offset: 0x0C */
__IO uint32_t AHB1RSTR; /*!< RCC AHB1 peripheral reset register, Address offset: 0x10 */
...
uint32_t RESERVED1[2]; /*!< Reserved, 0x28-0x2C */
__IO AHB1ENR_bits AHB1ENR; /*!< RCC AHB1 peripheral clock register, Address offset: 0x30 */
__IO uint32_t AHB2ENR; /*!< RCC AHB2 peripheral clock register, Address offset: 0x34 */
...
} RCC_TypeDef;

#define RCC ((RCC_TypeDef *) RCC_BASE)
```

STM32F401RE_RCC.c

```c
void configurePLL() {
/* TODO: Set clock to 24 MHz
Output freq = (src_clk) * (N/M) / P
(8 MHz) * (N/M) / P = 24 MHz
M:XX, N:XX, P:XX
Use HSE as PLLSRC
*/

RCC->CR.PLLON = 0; // Turn off PLL
while (RCC->CR.PLLRDY != 0); // Wait till PLL is
unlocked (e.g., off)

/* TODO: Load configuration with the values shown above
1. Set PLLSRC
2. Set M
3. Set N
4. Set P
*/

// TODO: Enable PLL and wait until it's locked (i.e.,
until PLLRDY goes high)

}
```

main_blink.c

```c
int main(void) {

/* TODO: Configure APB prescalers
1. Set APB2 (high-speed bus) prescaler to no division
2. Set APB1 (low-speed bus) to divide by 2.
*/

// TODO: Call configureClock() (declared in STM32F401RE_RCC.h)
configureClock();

// TODO: Turn on clock to GPIOA using GPIOA bitfield structure

// TODO: Set pinMode for PA8 to be alternate function
pinMode(GPIO_PA8, GPIO_ALT);

// Set LED_PIN as output
pinMode(LED_PIN, GPIO_OUTPUT);

// Configure LED_PIN for high speed output
GPIOA->OSPEEDR |= (0b11 << 2*LED_PIN);

// Blink LED
while(1) {
ms_delay(DELAY_MS);
togglePin(LED_PIN);
}
return 0;
}
```

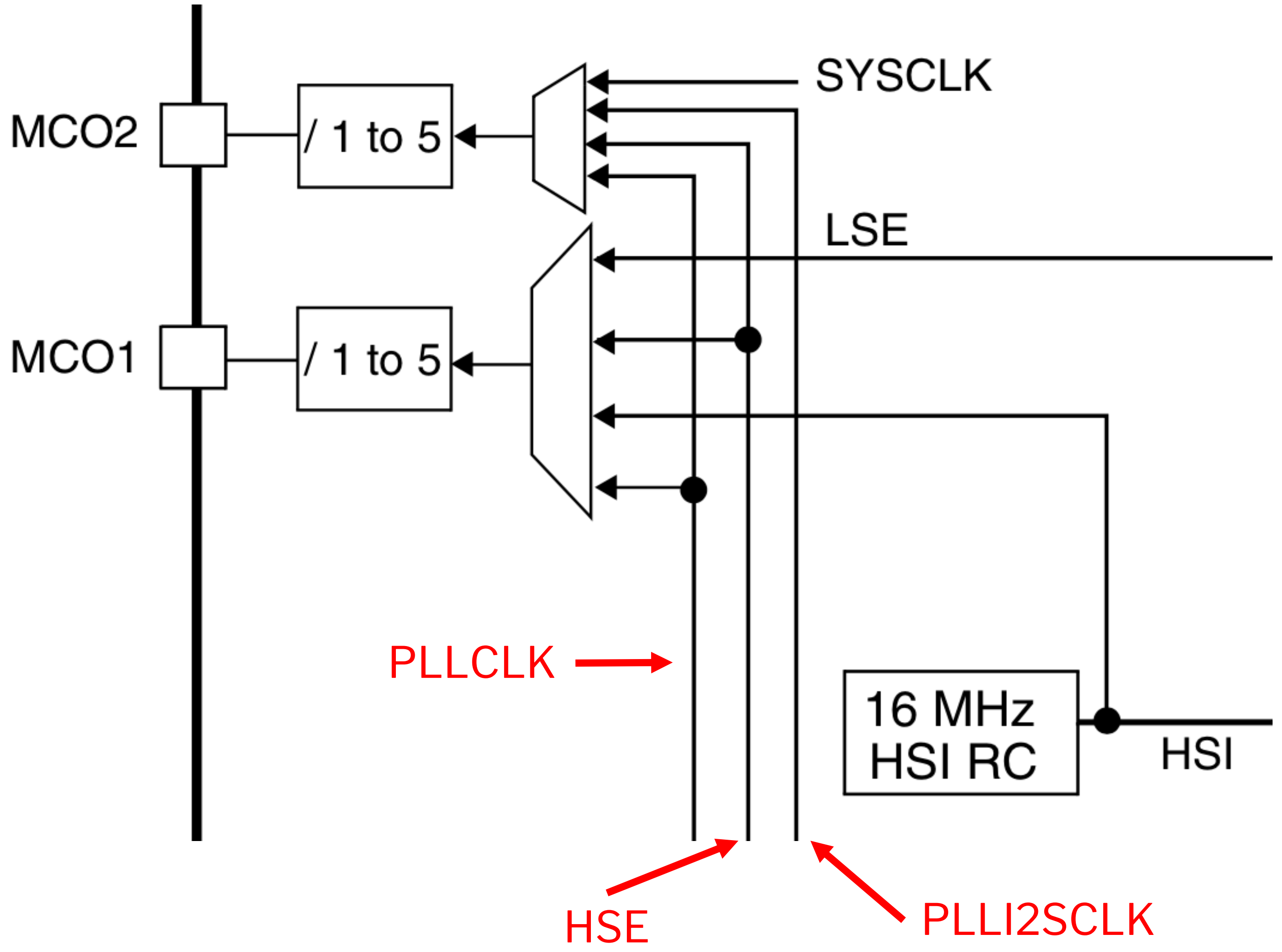
Your turn!

- Download the source files for lecture 6 from Github. You can either:
  - Clone with `git clone https://github.com/joshbrake/E155_FA2020.git`
  - Pull to existing E155_FA2020 repo
  - Download as a .zip
- Work in your breakout rooms to…
  1. In `STM32F401RE_RCC.h` fill out the bitfield structures for
     - CR_bits
     - PLLCFGR_bits
     - CFGR_bits
  2. In `STM32F401RE_RCC.c`, fill out the functions `configurePLL()`. You should configure the PLL to use the external oscillator (HSE) and to run at 24 MHz.
  3. Fill out missing pieces in `main_blink.c`

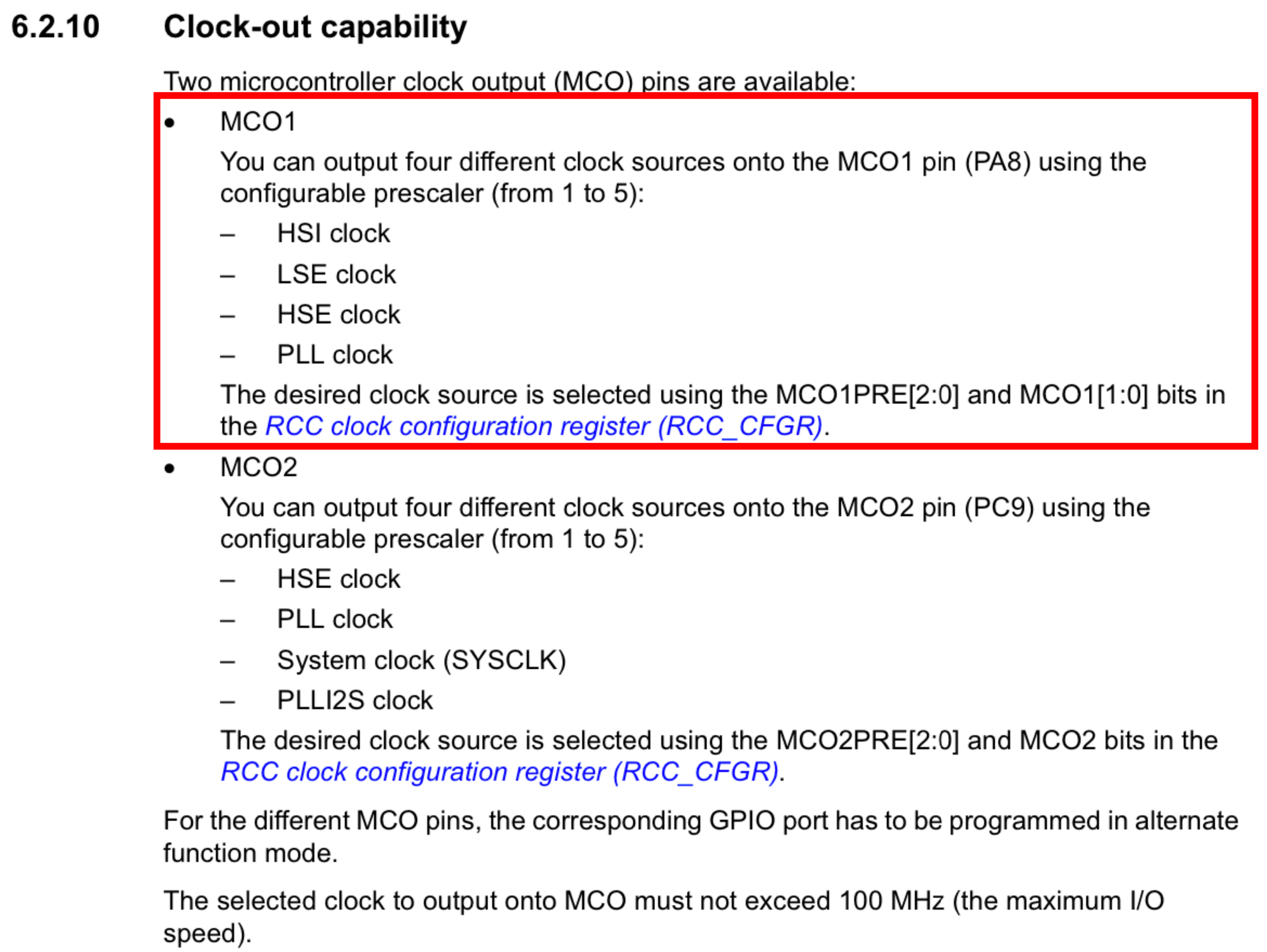
https://docs.google.com/presentation/d/1ShVehgj6aX2Dr44j0UIh4JXsUJaZQ8HdsNg32T6HZcs/edit#slide=id.g95d84e93ee_0_102

Confirming Clock Configuration

RCC Clock-out



RCC Clock-out



6.2.10 Clock-out capability

Two microcontroller clock output (MCO) pins are available:

- MCO1

  You can output four different clock sources onto the MCO1 pin (PA8) using the configurable prescaler (from 1 to 5):
  - HSI clock
  - LSE clock
  - HSE clock
  - PLL clock

  The desired clock source is selected using the MCO1PRE[2:0] and MCO1[1:0] bits in the *RCC clock configuration register (RCC_CFGR)*.
- MCO2

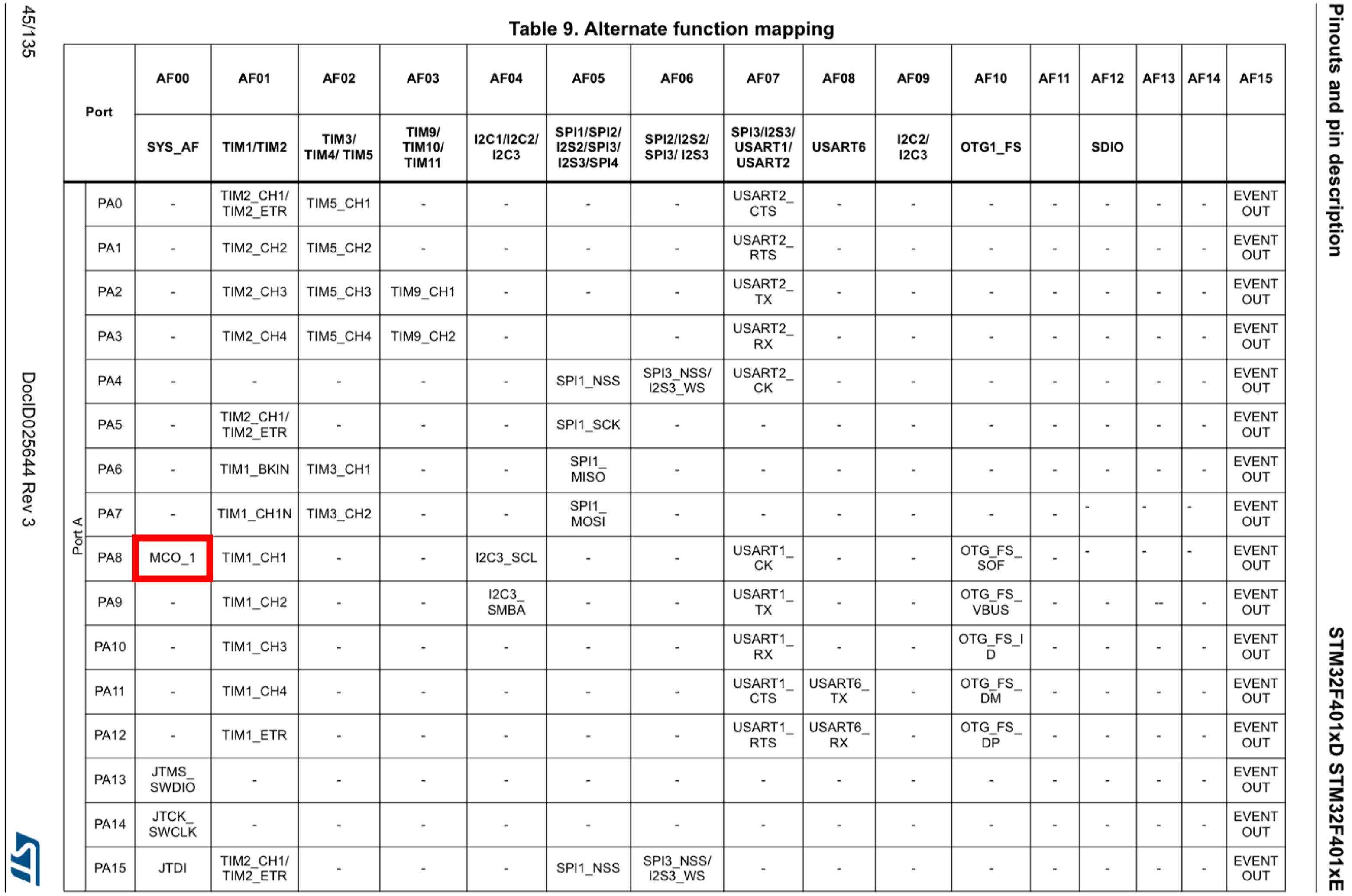
  You can output four different clock sources onto the MCO2 pin (PC9) using the configurable prescaler (from 1 to 5):
  - HSE clock
  - PLL clock
  - System clock (SYSCLK)
  - PLLI2S clock

  The desired clock source is selected using the MCO2PRE[2:0] and MCO2 bits in the *RCC clock configuration register (RCC_CFGR)*.

For the different MCO pins, the corresponding GPIO port has to be programmed in alternate function mode.

The selected clock to output onto MCO must not exceed 100 MHz (the maximum I/O speed).

MCO_1 pin location



45/135

DocID025644 Rev 3

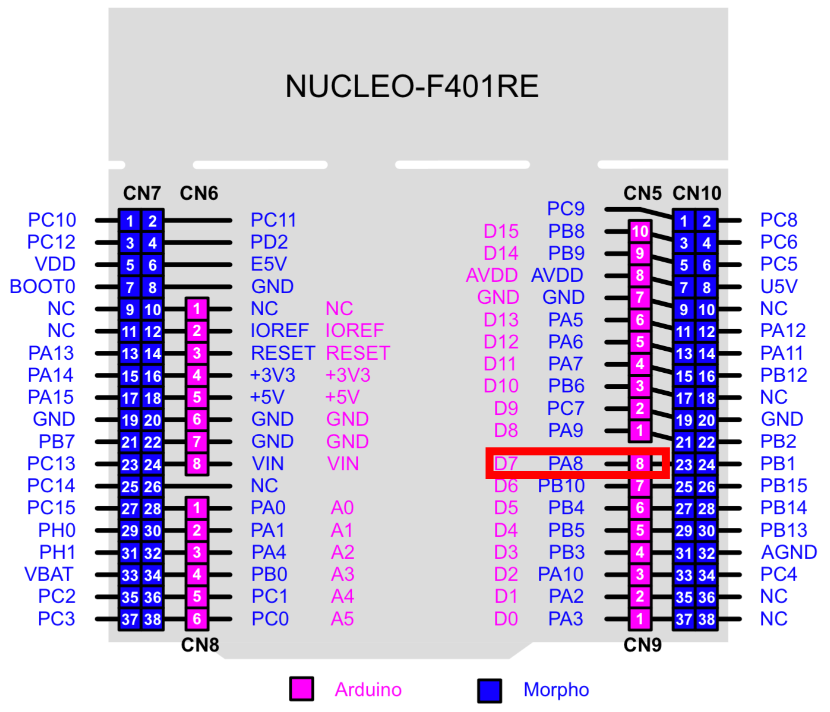
Pinouts and pin description

STM32F401xD STM32F401xE

Table 9. Alternate function mapping

Port		AF00	AF01	AF02	AF03	AF04	AF05	AF06	AF07	AF08	AF09	AF10	AF11	AF12	AF13	AF14	AF15
		SYS_AF	TIM1/TIM2	TIM3/ TIM4/ TIM5	TIM9/ TIM10/ TIM11	I2C1/I2C2/ I2C3	SPI1/SPI2/ I2S2/SPI3/ I2S3/SPI4	SPI2/I2S2/ SPI3/ I2S3	SPI3/I2S3/ USART1/ USART2	USART6	I2C2/ I2C3	OTG1_FS		SDIO			
Port A	PA0	-	TIM2_CH1/ TIM2_ETR	TIM5_CH1	-	-	-	-	USART2_ CTS	-	-	-	-	-	-	-	EVENT OUT
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_ RTS	-	-	-	-	-	-	-	EVENT OUT
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_ TX	-	-	-	-	-	-	-	EVENT OUT
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-		-	USART2_ RX	-	-	-	-	-	-	-	EVENT OUT
	PA4	-	-	-	-	-	SPI1_NSS	SPI3_NSS/ I2S3_WS	USART2_ CK	-	-	-	-	-	-	-	EVENT OUT
	PA5	-	TIM2_CH1/ TIM2_ETR	-	-	-	SPI1_SCK	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA6	-	TIM1_BKIN	TIM3_CH1	-	-	SPI1_ MISO	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA7	-	TIM1_CH1N	TIM3_CH2	-	-	SPI1_ MOSI	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA8	MCO_1	TIM1_CH1	-	-	I2C3_SCL	-	-	USART1_ CK	-	-	OTG_FS_ SOF	-	-	-	-	EVENT OUT
	PA9	-	TIM1_CH2	-	-	I2C3_ SMBA	-	-	USART1_ TX	-	-	OTG_FS_ VBUS	-	-	--	-	EVENT OUT
	PA10	-	TIM1_CH3	-	-	-	-	-	USART1_ RX	-	-	OTG_FS_I D	-	-	-	-	EVENT OUT
	PA11	-	TIM1_CH4	-	-	-	-	-	USART1_ CTS	USART6_ TX	-	OTG_FS_ DM	-	-	-	-	EVENT OUT
	PA12	-	TIM1_ETR	-	-	-	-	-	USART1_ RTS	USART6_ RX	-	OTG_FS_ DP	-	-	-	-	EVENT OUT
	PA13	JTMS_ SWDIO	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA14	JTCK_ SWCLK	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA15	JTDI	TIM2_CH1/ TIM2_ETR	-	-	-	SPI1_NSS	SPI3_NSS/ I2S3_WS	-	-	-	-	-	-	-	-	EVENT OUT

Figure 18. NUCLEO-F401RE

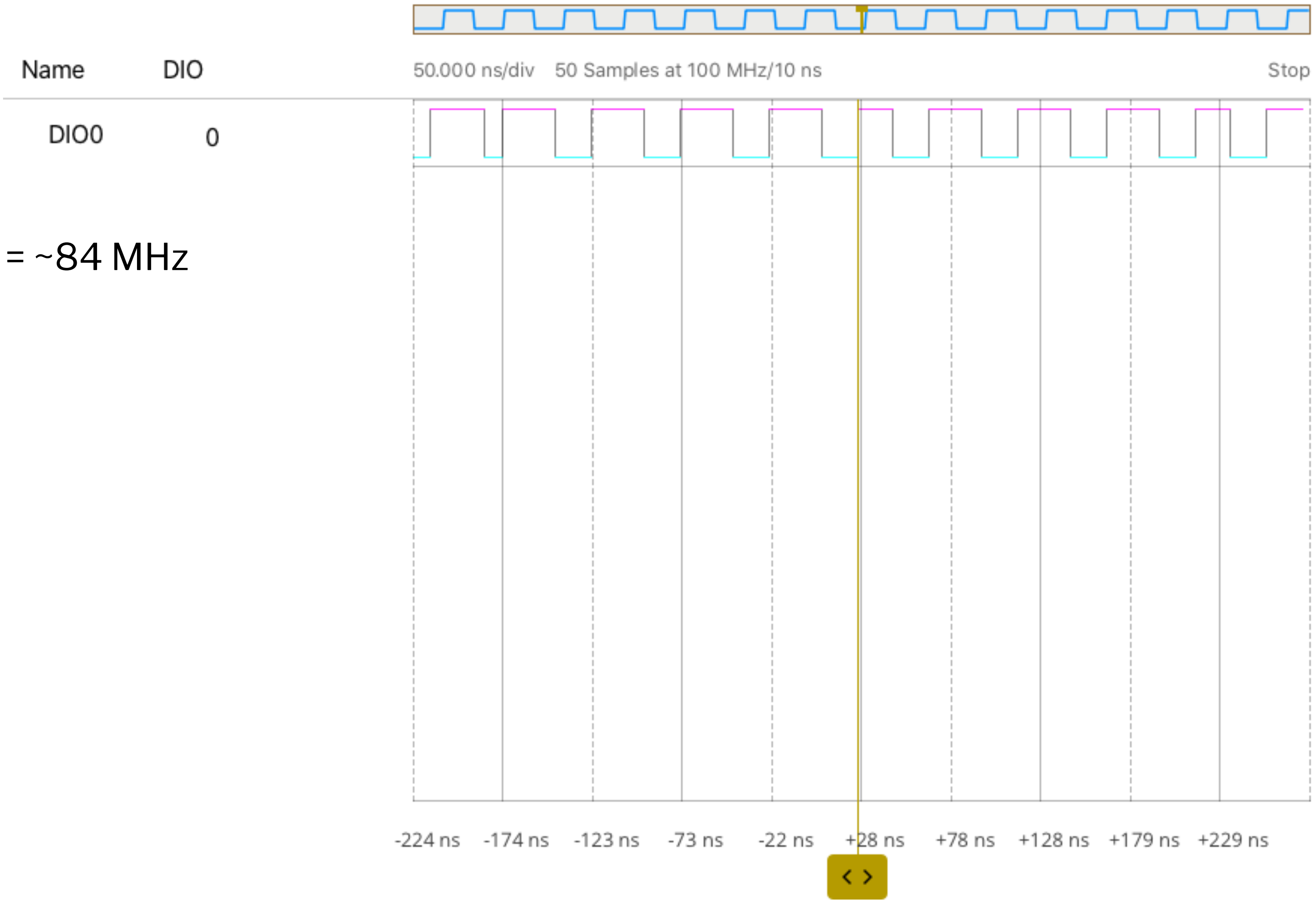


NUCLEO-F401RE
CN7 CN6 CN8 CN5 CN10 CN9
PC10 PC12 VDD BOOT0 NC NC PA13 PA14 PA15 GND PB7 PC13 PC14 PC15 PH0 PH1 VBAT PC2 PC3
PC11 PD2 E5V GND NC IOREF RESET +3V3 +5V GND GND VIN NC PA0 PA1 PA4 PB0 PC1 PC0
NC IOREF RESET +3V3 +5V GND GND VIN A0 A1 A2 A3 A4 A5
D15 D14 AVDD GND D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
PC9 PB8 PB9 AVDD GND PA5 PA6 PA7 PB6 PC7 PA9 PA8 PB10 PB4 PB5 PB3 PA10 PA2 PA3
PC8 PC6 PC5 U5V NC PA12 PA11 PB12 NC GND PB2 PB1 PB15 PB14 PB13 AGND PC4 NC NC
Arduino
Morpho

Scopy Screengrab of MCO_1

PLL/4

f = 1/(~50e-9 s)*4 = ~84 MHz

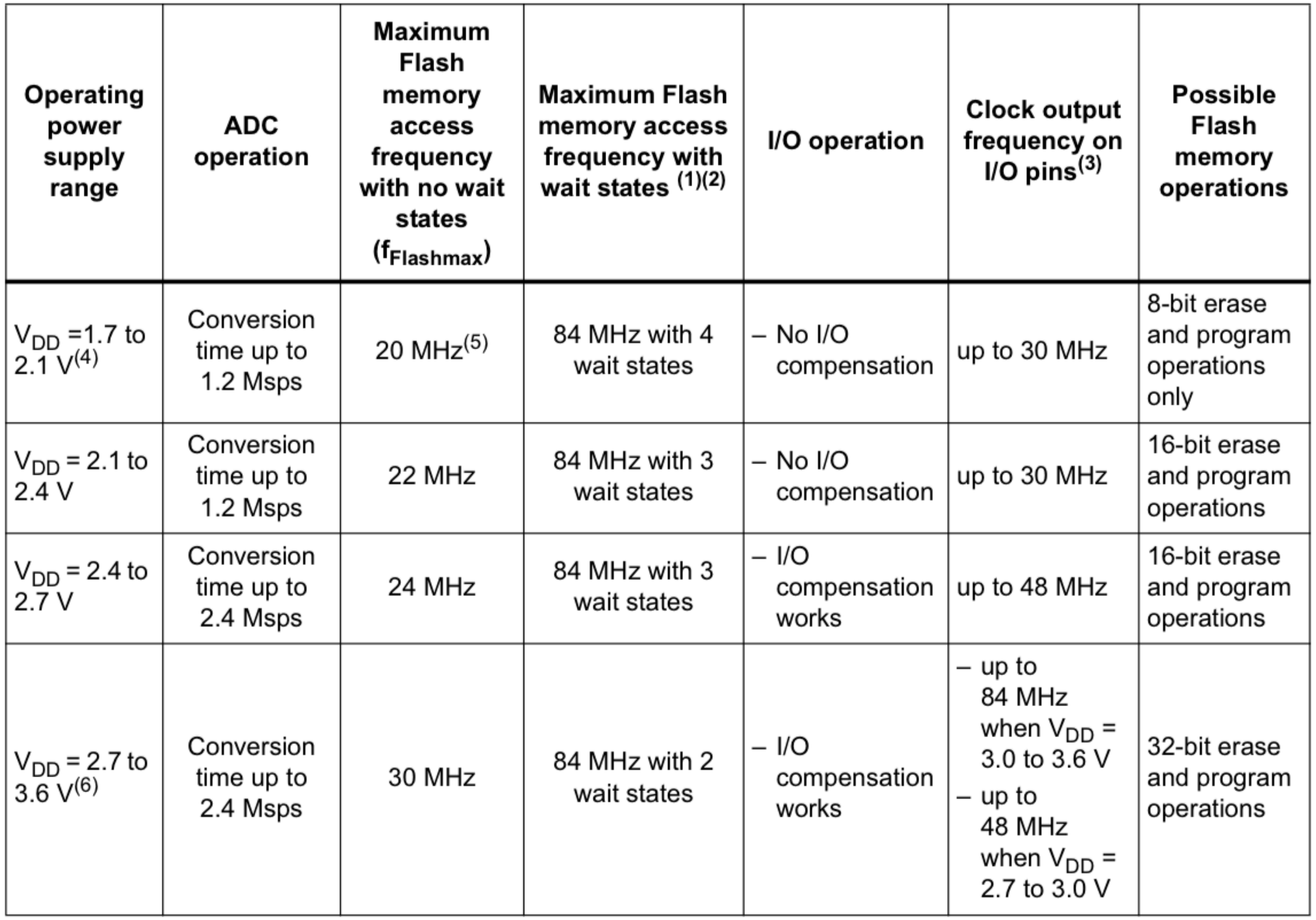


How can we configure the clocks for 84 MHz?

- Turn on HSE (8 MHz source routed from ST-LINK) and HSEBYP (in RCC_CR)
- Configure PLL (RCC_PLLCFGR)
  - M = 8 (VCO_{in} = HSE/8 = 8 MHz/8 = 1 MHz)
  - N = 336 (VCO_{out} = VCO_{in} * 336 = 1 MHz * 336 = 336 MHz)
  - P = 4 (PLLCLK = VCO_{out} /4 = 84 MHz)
- Set the PLL as the clock source (RCC_CFGR)
- Try this out in your demo code

Flash speed

Table 15. Features depending on the operating power supply range

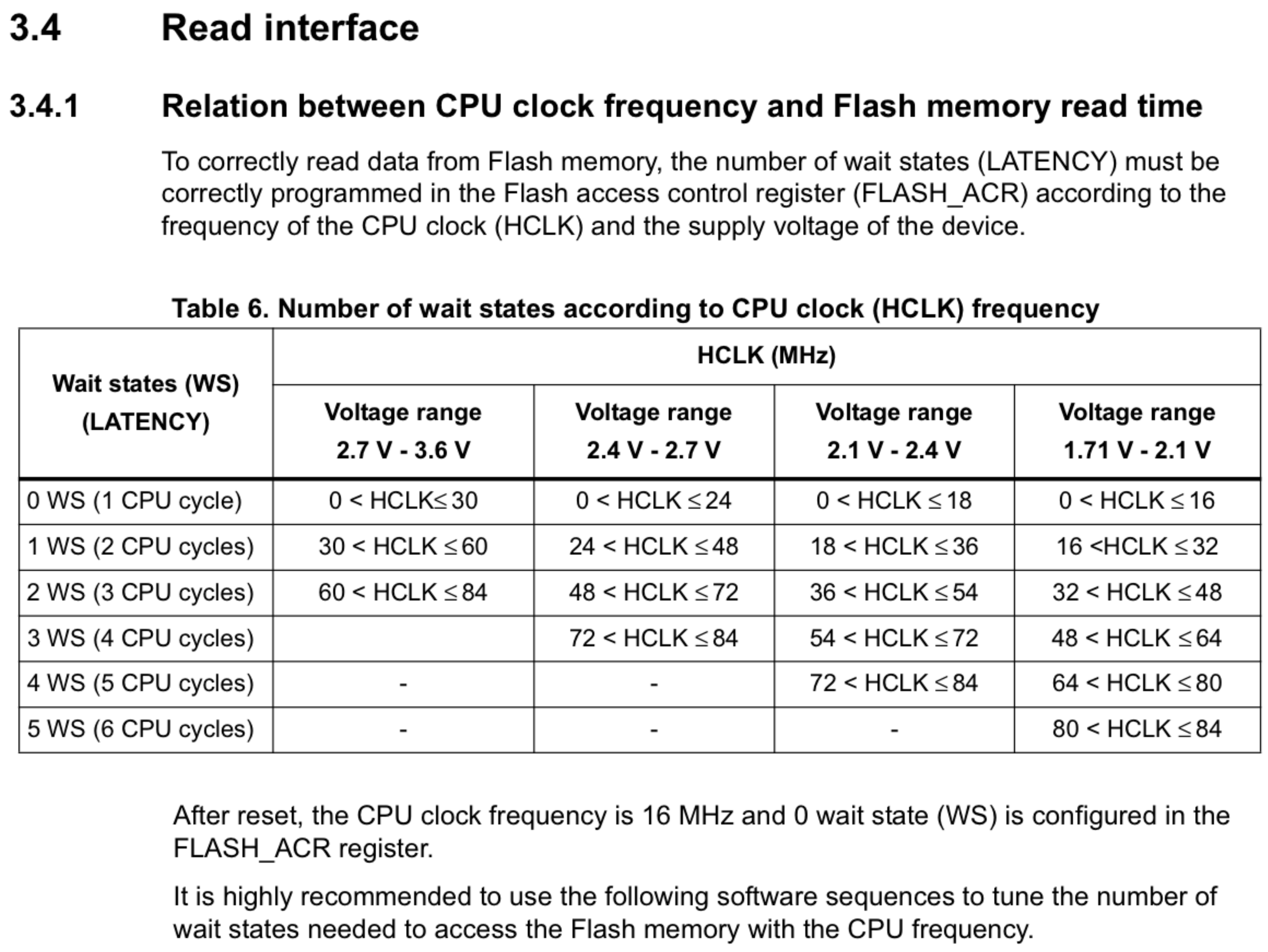


Operating power supply range	ADC operation	Maximum Flash memory access frequency with no wait states ($f_{Flashmax}$)	Maximum Flash memory access frequency with wait states [(1)(2)]	I/O operation	Clock output frequency on I/O pins[(3)]	Possible Flash memory operations
V_{DD} =1.7 to 2.1 V[(4)]	Conversion time up to 1.2 Msps	20 MHz[(5)]	84 MHz with 4 wait states	– No I/O compensation	up to 30 MHz	8-bit erase and program operations only
V_{DD} = 2.1 to 2.4 V	Conversion time up to 1.2 Msps	22 MHz	84 MHz with 3 wait states	– No I/O compensation	up to 30 MHz	16-bit erase and program operations
V_{DD} = 2.4 to 2.7 V	Conversion time up to 2.4 Msps	24 MHz	84 MHz with 3 wait states	– I/O compensation works	up to 48 MHz	16-bit erase and program operations
V_{DD} = 2.7 to 3.6 V[(6)]	Conversion time up to 2.4 Msps	30 MHz	84 MHz with 2 wait states	– I/O compensation works	– up to 84 MHz when V_{DD} = 3.0 to 3.6 V – up to 48 MHz when V_{DD} = 2.7 to 3.0 V	32-bit erase and program operations

VDD = 3.3 V provided from regulator on Nucleo board

STM32F401RE Datasheet p. 61

Flash speed



3.4 Read interface

3.4.1 Relation between CPU clock frequency and Flash memory read time

To correctly read data from Flash memory, the number of wait states (LATENCY) must be correctly programmed in the Flash access control register (FLASH_ACR) according to the frequency of the CPU clock (HCLK) and the supply voltage of the device.

Table 6. Number of wait states according to CPU clock (HCLK) frequency

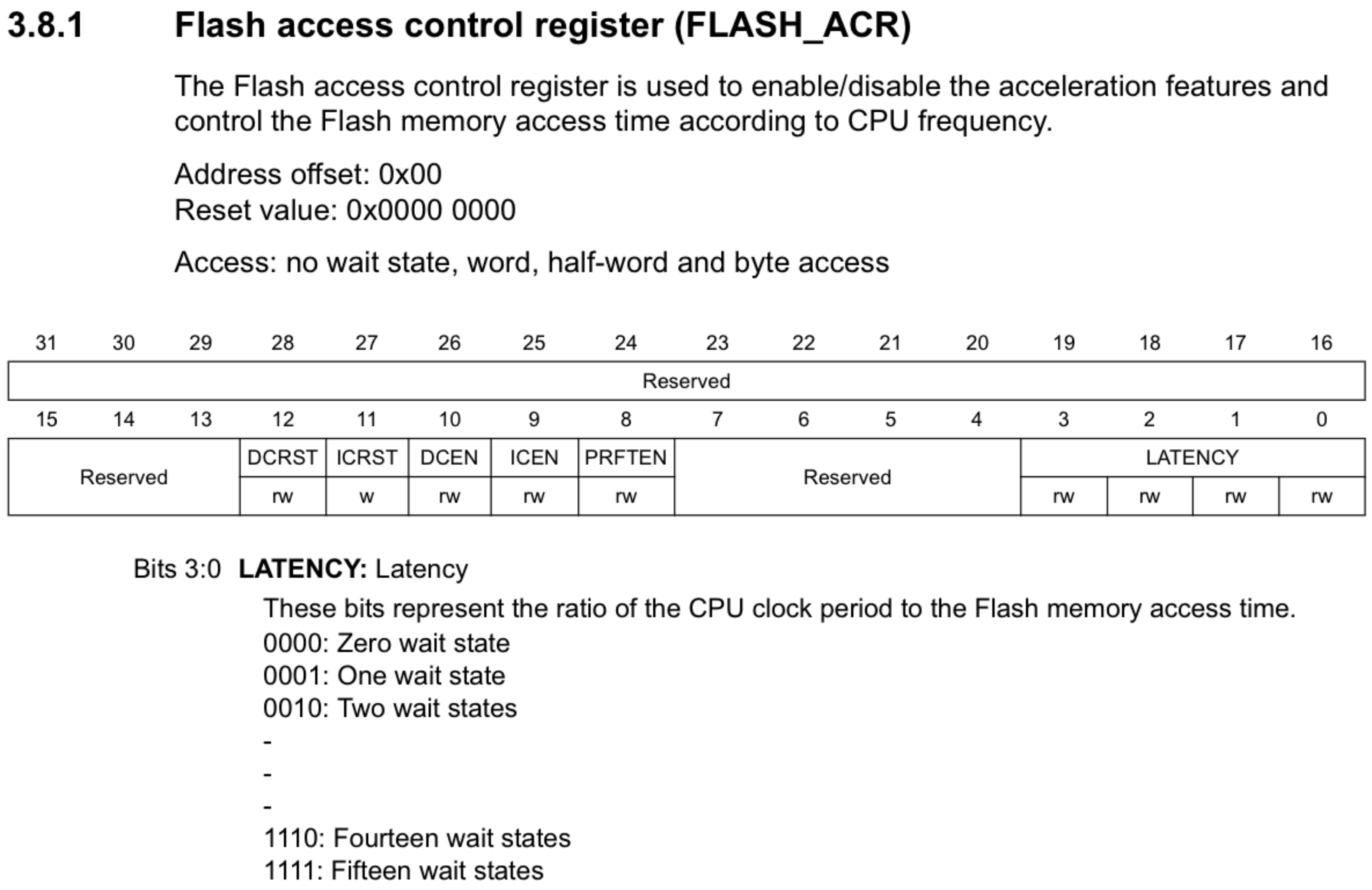
Wait states (WS) (LATENCY)	HCLK (MHz)			
	Voltage range 2.7 V - 3.6 V	Voltage range 2.4 V - 2.7 V	Voltage range 2.1 V - 2.4 V	Voltage range 1.71 V - 2.1 V
0 WS (1 CPU cycle)	0 < HCLK≤ 30	0 < HCLK ≤ 24	0 < HCLK ≤ 18	0 < HCLK ≤ 16
1 WS (2 CPU cycles)	30 < HCLK ≤ 60	24 < HCLK ≤ 48	18 < HCLK ≤ 36	16 <HCLK ≤ 32
2 WS (3 CPU cycles)	60 < HCLK ≤ 84	48 < HCLK ≤ 72	36 < HCLK ≤ 54	32 < HCLK ≤ 48
3 WS (4 CPU cycles)		72 < HCLK ≤ 84	54 < HCLK ≤ 72	48 < HCLK ≤ 64
4 WS (5 CPU cycles)	-	-	72 < HCLK ≤ 84	64 < HCLK ≤ 80
5 WS (6 CPU cycles)	-	-	-	80 < HCLK ≤ 84

After reset, the CPU clock frequency is 16 MHz and 0 wait state (WS) is configured in the FLASH_ACR register.

It is highly recommended to use the following software sequences to tune the number of wait states needed to access the Flash memory with the CPU frequency.

STM32F401RE Reference Manual p. 46

Setting flash latency



3.8.1 Flash access control register (FLASH_ACR)

The Flash access control register is used to enable/disable the acceleration features and control the Flash memory access time according to CPU frequency.

Address offset: 0x00
Reset value: 0x0000 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
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			DCRST	ICRST	DCEN	ICEN	PRFTEN	Reserved				LATENCY			
			rw	w	rw	rw	rw					rw	rw	rw	rw

Bits 3:0 **LATENCY:** Latency

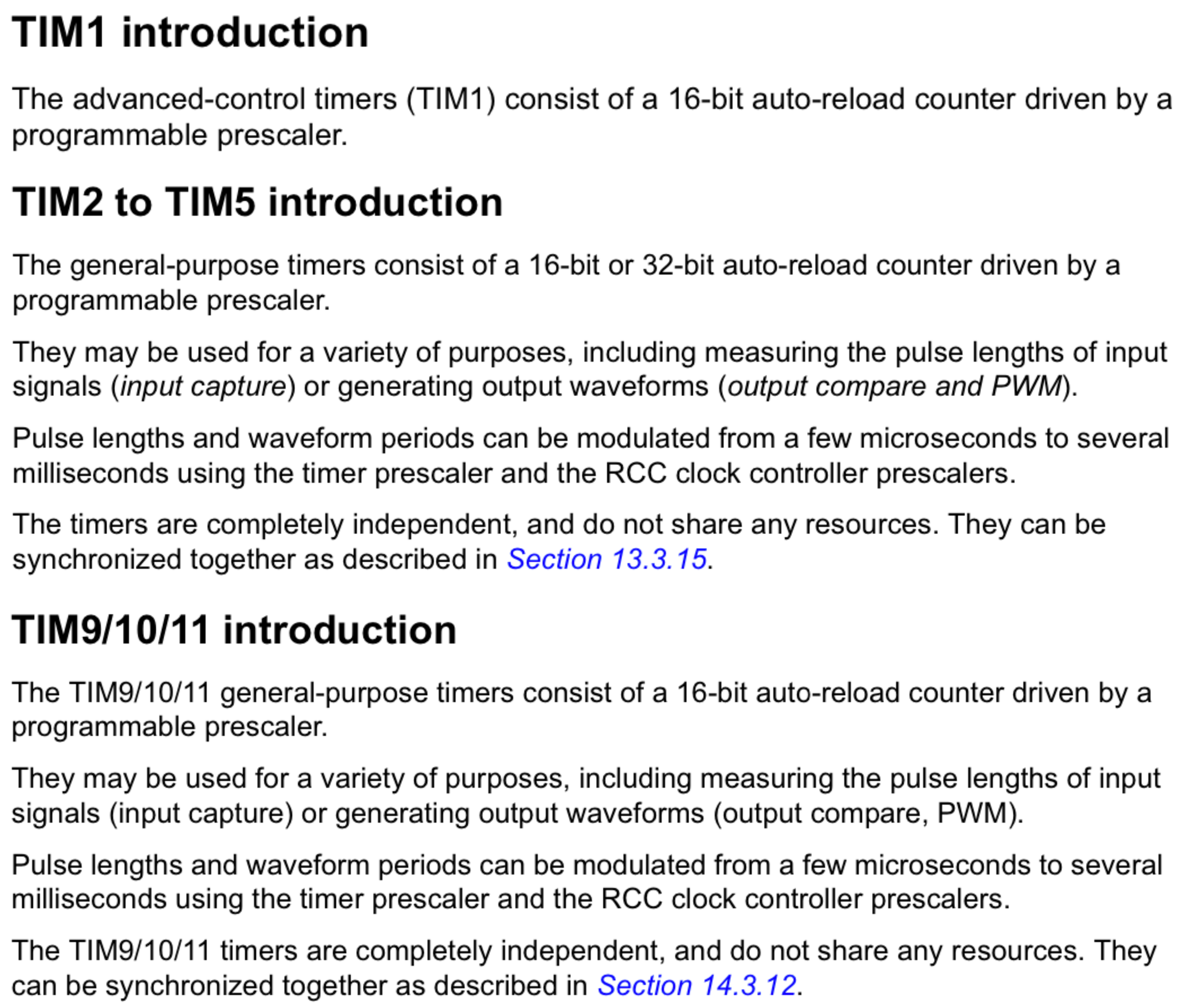
These bits represent the ratio of the CPU clock period to the Flash memory access time.

0000: Zero wait state
0001: One wait state
0010: Two wait states
-
-
-
1110: Fourteen wait states
1111: Fifteen wait states

STM32F401RE Reference Manual p. 60

Timers

Timers on STM32F401RE



TIM1 introduction

The advanced-control timers (TIM1) consist of a 16-bit auto-reload counter driven by a programmable prescaler.

TIM2 to TIM5 introduction

The general-purpose timers consist of a 16-bit or 32-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (*input capture*) or generating output waveforms (*output compare and PWM*).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

The timers are completely independent, and do not share any resources. They can be synchronized together as described in *Section 13.3.15*.

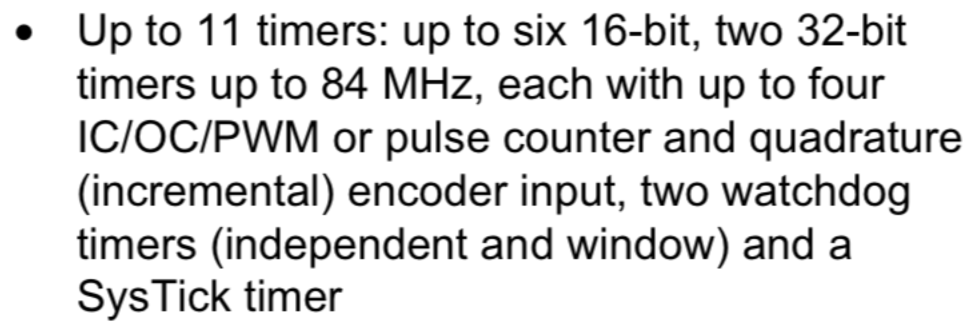
TIM9/10/11 introduction

The TIM9/10/11 general-purpose timers consist of a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, PWM).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

The TIM9/10/11 timers are completely independent, and do not share any resources. They can be synchronized together as described in *Section 14.3.12*.



- Up to 11 timers: up to six 16-bit, two 32-bit timers up to 84 MHz, each with up to four IC/OC/PWM or pulse counter and quadrature (incremental) encoder input, two watchdog timers (independent and window) and a SysTick timer

STM32F401RE Datasheet p. 1

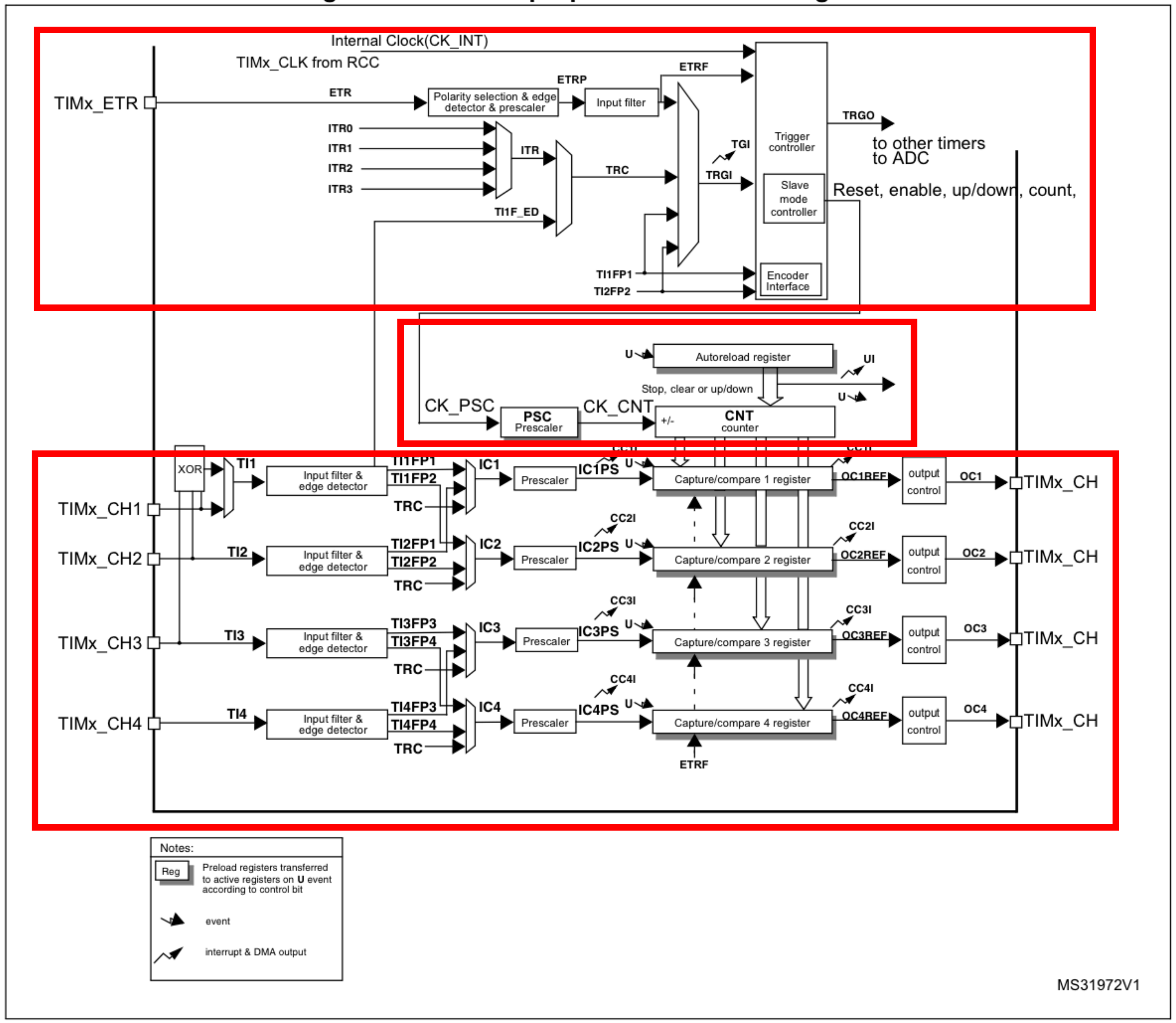
Block diagram

Clock Source and Trigger Controller

Time-base unit

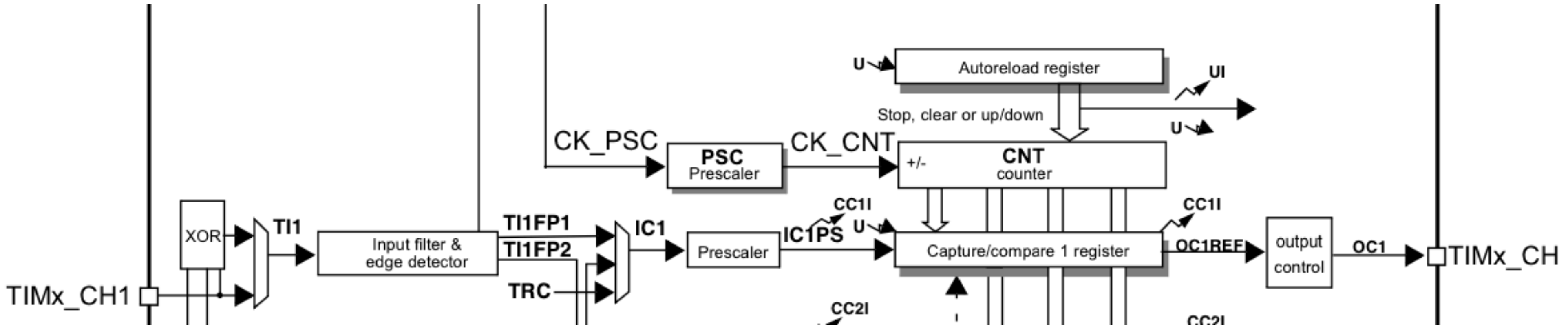
Counter Capture/Compare Channels

Figure 87. General-purpose timer block diagram



STM32F401RE Reference Manual p. 317

Single Timer Channel



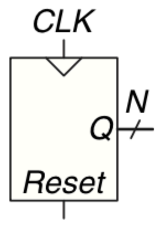


Figure 5.31 Counter symbol

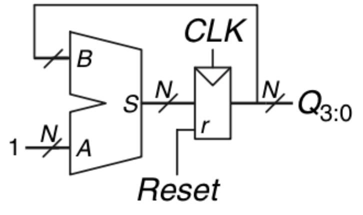
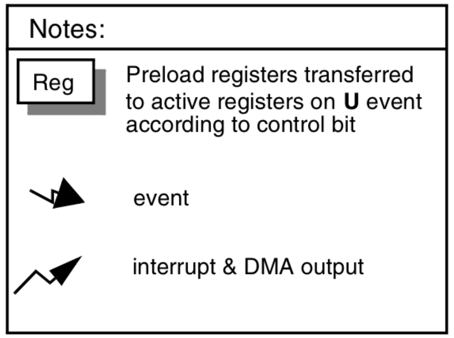


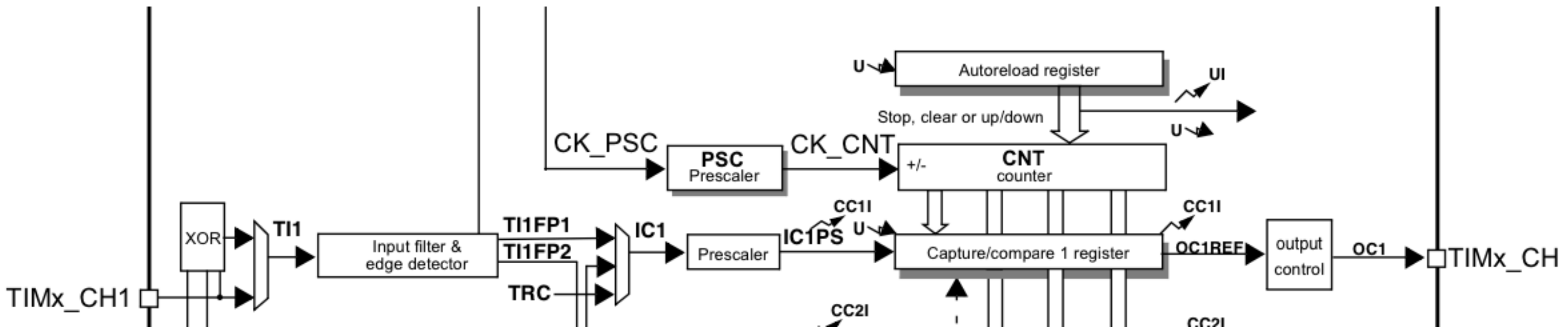
Figure 5.32 *N*-bit counter

DDCA ARMed p. 260



Notes:
Reg
Preload registers transferred to active registers on **U** event according to control bit
event
interrupt & DMA output

Single Timer Channel



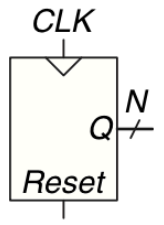


Figure 5.31 Counter symbol

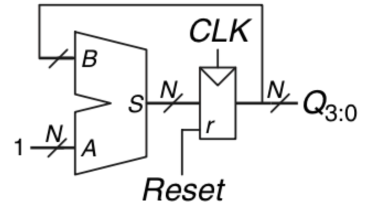


Figure 5.32 *N*-bit counter

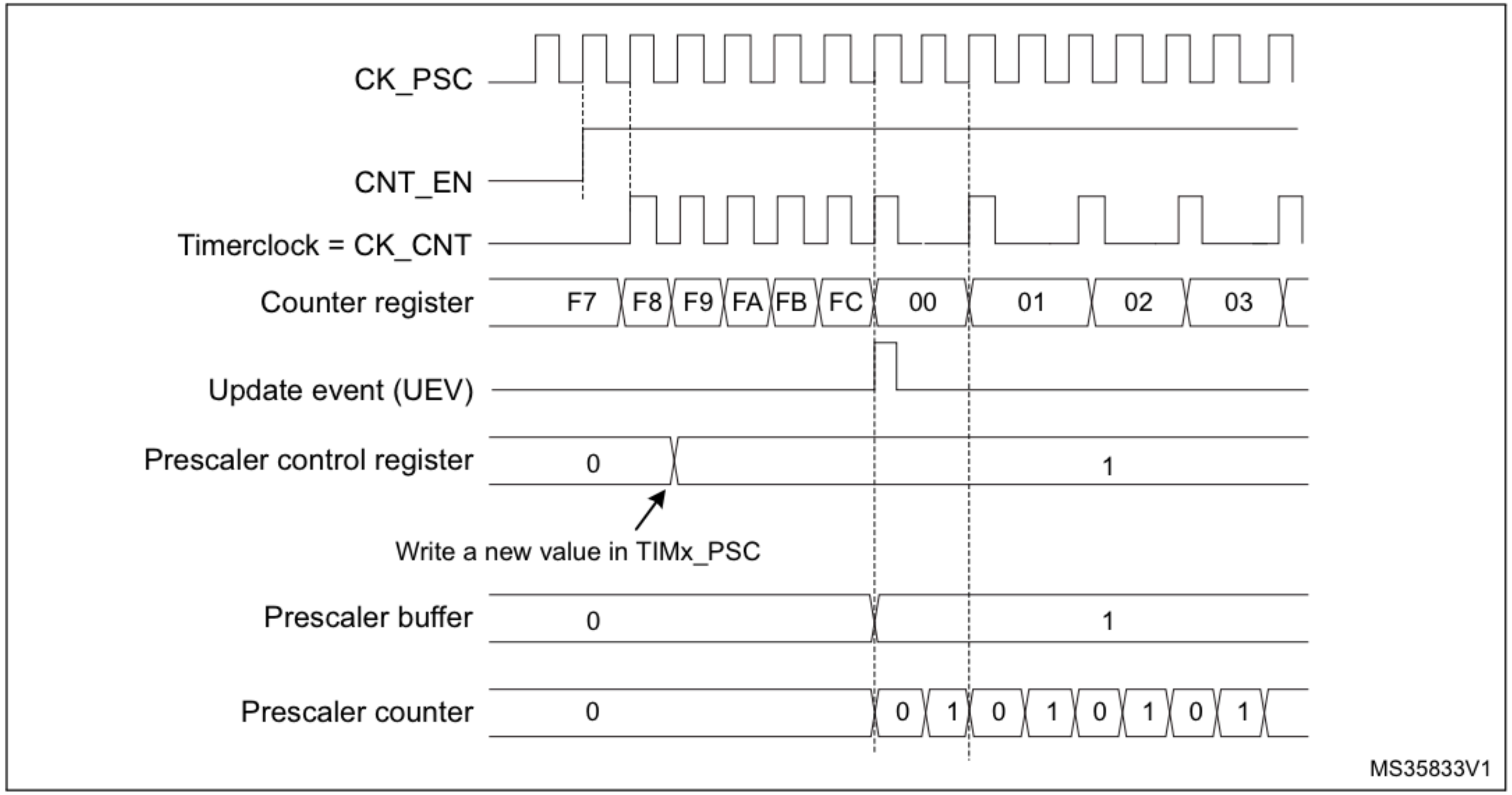
DDCA ARMed p. 260

Main blocks

- Prescaler Register (TIMx_PSC): counter to pre-scale the input clock signal (CK_PSC)
- Counter Register (TIMx_CNT): Counter which holds the current count value. Counts at the rate specified by the prescaled clock (CK_CNT)
- Auto-Reload Register (TIMx_ARR): Holds the max value for the counter
- Capture/compare register (CCR_X): Used to generate a signal that will be sent to the output control block for more advanced operations like pulse-width modulation (PWM)

Timer Example Diagrams: Prescaler

Figure 88. Counter timing diagram with prescaler division change from 1 to 2



Counter: Upcounting mode

TIMx_ARR=0x36

Figure 90. Counter timing diagram, internal clock divided by 1

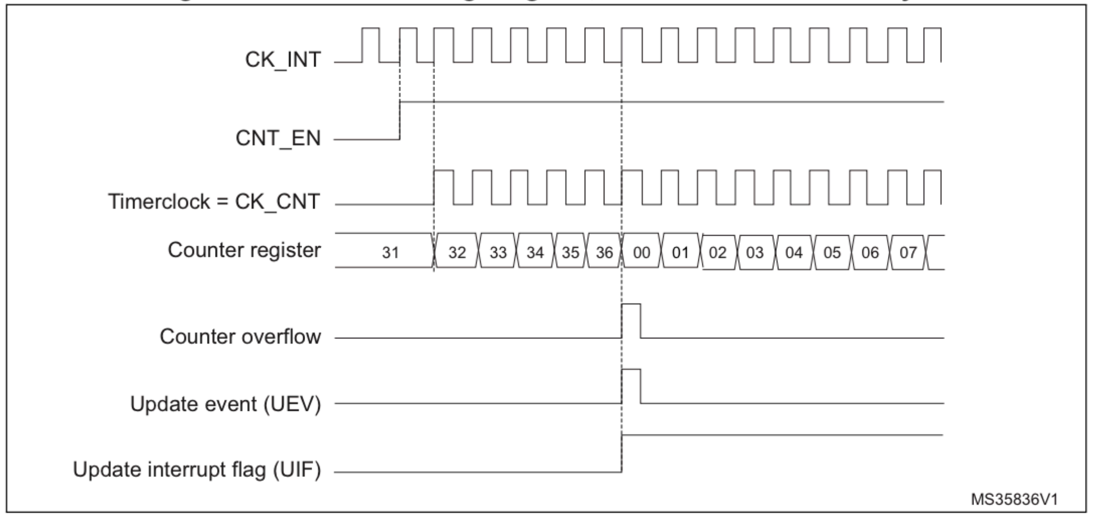
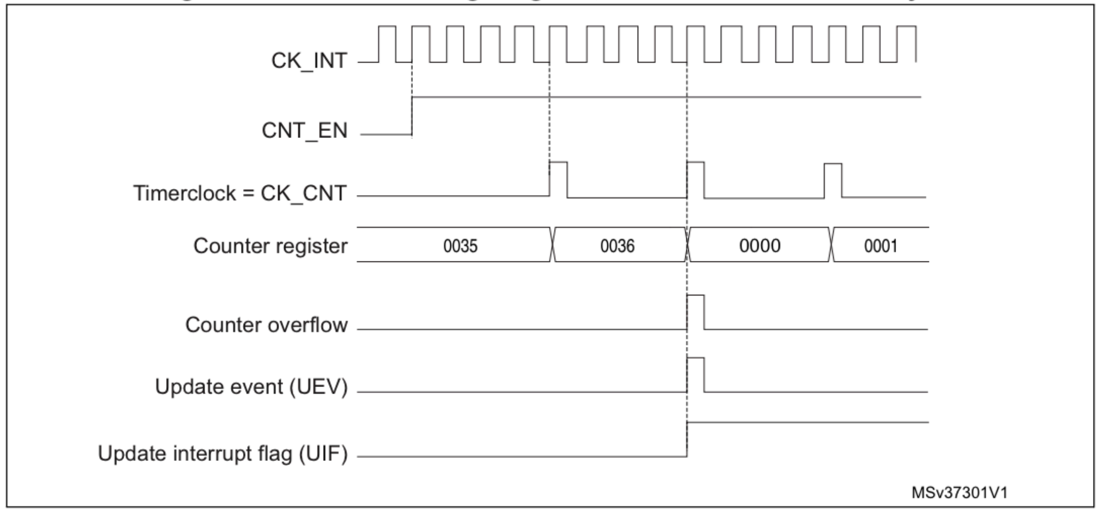


Figure 92. Counter timing diagram, internal clock divided by 4



Counter: Downcounting mode

TIMx_ARR=0x36

Figure 96. Counter timing diagram, internal clock divided by 1

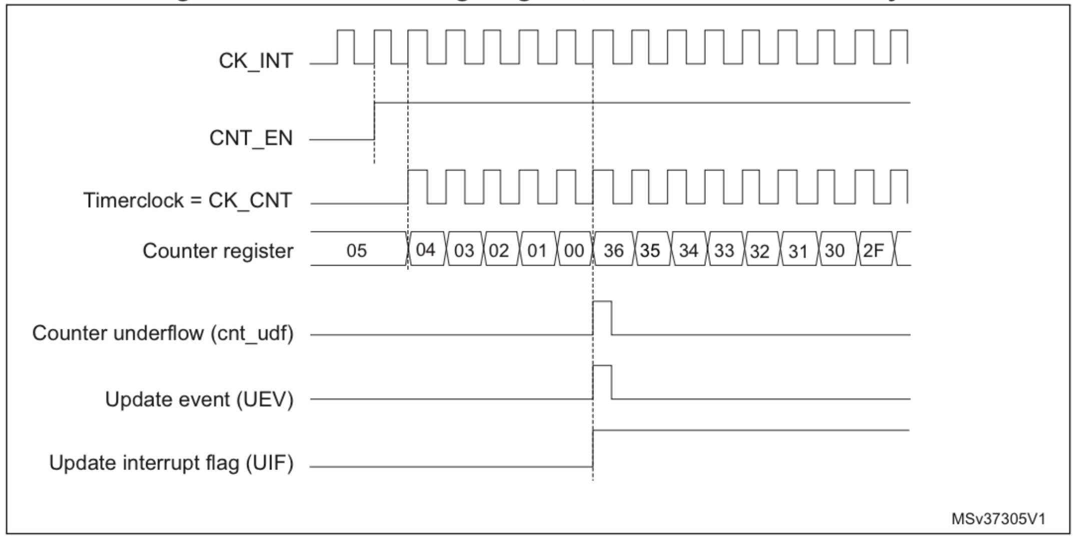
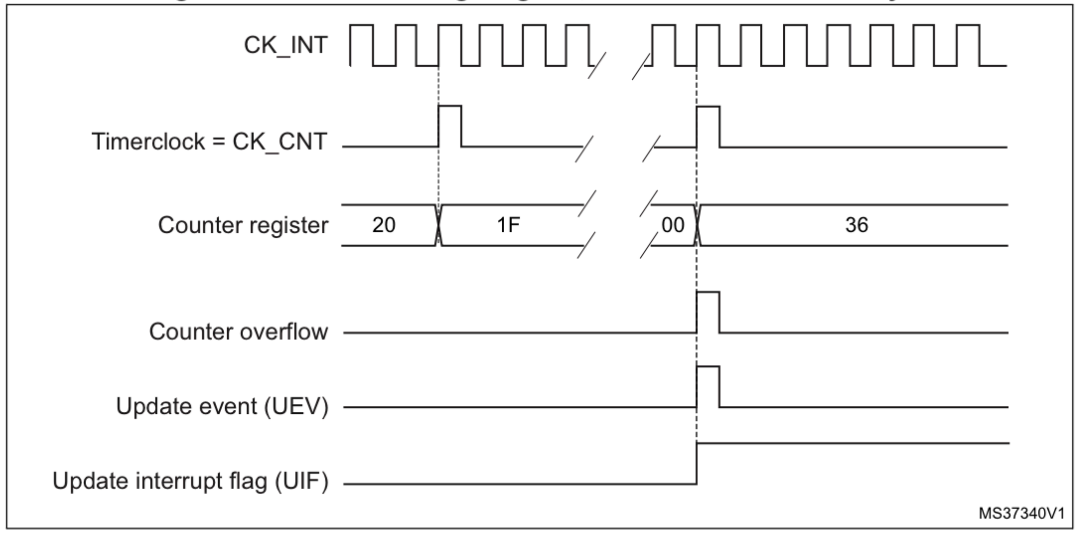


Figure 99. Counter timing diagram, internal clock divided by N



Hints for Lab 3: Digital Audio

- Use timers to generate square waves of a given frequency
- Some hints on timer configuration configuration
  - Configure clocks in RCC (as shown earlier in this lecture; take note of system clock frequency)
  - Turn on clock to timer in RCC
  - Select correct clock source in TIM control (make sure slave mode is disabled)
  - Configure counter
    - Prescaler register (TIMx_PSC)
    - Auto-reload register (TIMx_ARR)
  - Enable counter CEN in TIMx_CR

Summary

- Bitfield structures – convenient ways to control bits within a register
- Clock configuration – operation frequency is customizable, but need to carefully follow constraints on PLL settings.
- Timers - A layer wrapped around a few counters to enable precise timing and generate outputs like PWM signals

Lecture Feedback

- What is the most important thing you learned in class today?
- What point was most unclear from lecture today?

https://forms.gle/Ay6MkpZ6x3xsW2Eb8

Feedback