TRM0090

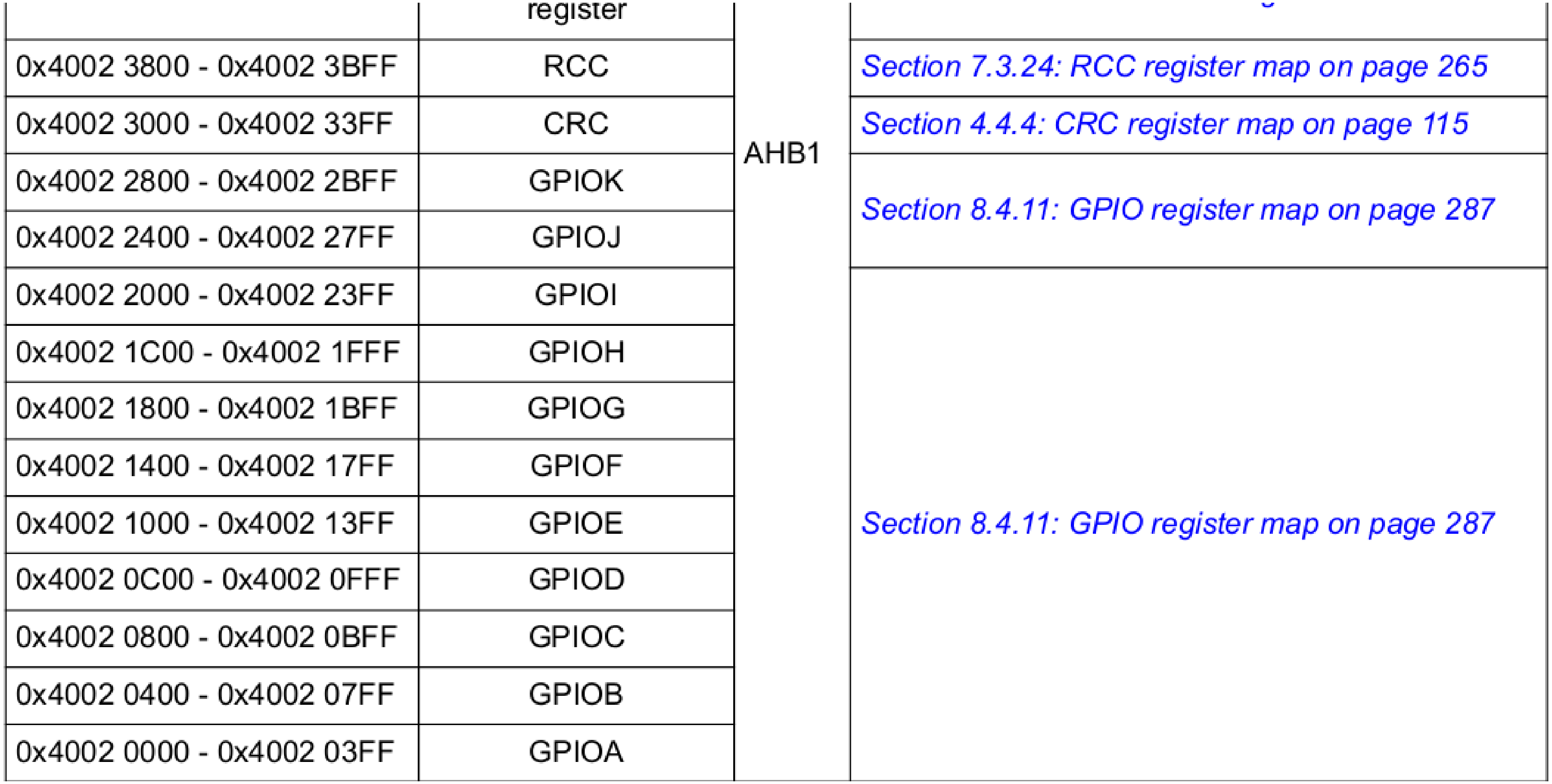
TRM0090

Reference manual

STM32F405/415, STM32F407/417, STM32F427/437 and

STM32F429/439 advanced ARM ® -based 32-bit MCUs

TRM0090 – Table 1. STM32F4xx register boundary addresses



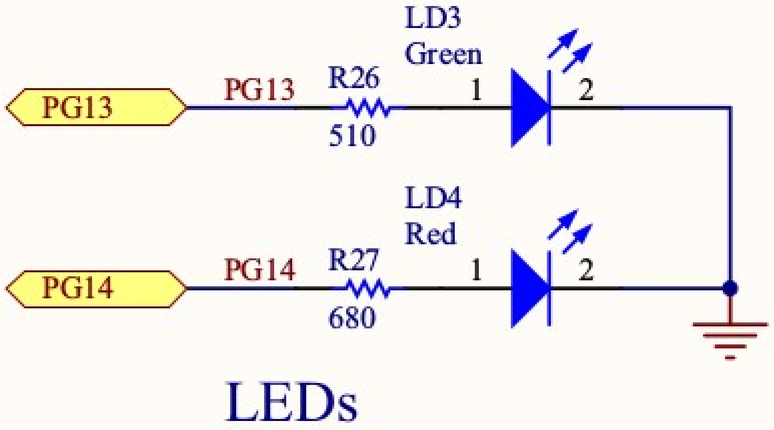
	register	AHB1	
0x4002 3800 - 0x4002 3BFF	RCC		*Section 7.3.24: RCC register map on page 265*
0x4002 3000 - 0x4002 33FF	CRC		*Section 4.4.4: CRC register map on page 115*
0x4002 2800 - 0x4002 2BFF	GPIOK		*Section 8.4.11: GPIO register map on page 287*
0x4002 2400 - 0x4002 27FF	GPIOJ		
0x4002 2000 - 0x4002 23FF	GPIOI		*Section 8.4.11: GPIO register map on page 287*
0x4002 1C00 - 0x4002 1FFF	GPIOH		
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

LEDs STM32F429

Page 33 UM1670

User manual

Discovery kit with STM32F429ZI MCU



General Purpose I/O (GPIO)

Up to 144 GPIO pins, individually configurable

Each port (GPIOA through GPIOI) comprises 16 GPIO pins

Pin options (each pin configurable via GPIO registers):

- Input, Output, Analog, Alternate functions e.g. Uart tx.

Digital data input/output via GPIO registers

enable the clock

```
  ldr     r0, =RCC_AHB1ENR
  ldr     r1, [r0]
```

firstly, to use any peripheral, the clock should be enabled for it.

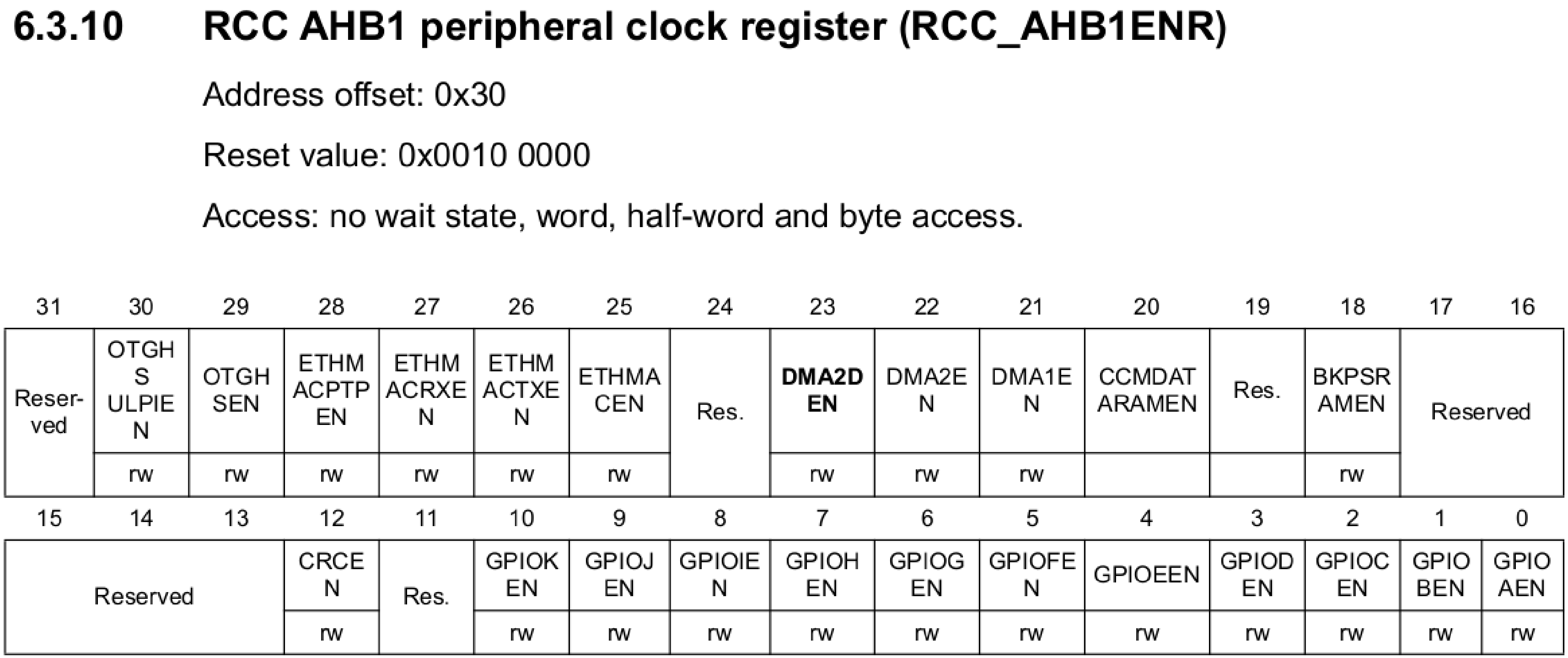
Different registers are responsible for that (see reference manual's "Reset and clock control").

For use of GPIOG it's clock is enabled in RCC_AHB1ENR register

Set the GPIOG ENable bit

```
    orr     r1, GPIOGEN
    str     r1, [r0]
```

enable the clock



6.3.10 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
Reserved	OTGHSULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Res.	**DMA2DEN**	DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	Reserved	
	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw		

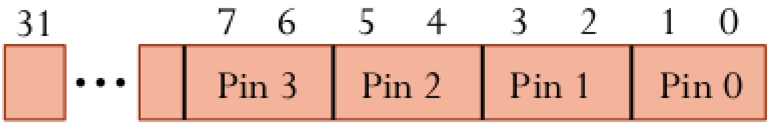
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Res.	GPIOKEN	GPIOJEN	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIO “mode” register

- GPIOx_MODER selects operating mode for each pin

  x = A…I (GPIOA, GPIOB, …, GPIOI)

- 2 bits per pin:



31 … 7 6 5 4 3 2 1 0

… Pin 3 Pin 2 Pin 1 Pin 0

00 – Input mode (reset state):

Pin value captured in IDR every bus clock (through Schmitt trigger)

01 – General purpose output mode:

- Write pin value to ODR
- Read IDR to determine pin state
- Read ODR for last written value

10 – Alternate function mode:

Select alternate function via AF mux/register (see later slide)

11 – Analog mode:

Disable output buffer, input Schmitt trigger, pull resistors

(so as not to alter the analog voltage on the pin)

Setup the Mode register

```
ldr     r0, =GPIOG_MODER
  ldr     r1,[r0]

  orr     r1, r1,#( MODER14_OUT | MODER13_OUT)
  str     r1,[r0]
```

GPIO data registers

- 16-bit memory-mapped data registers for each port GPIOx
  x = A…I (GPIOA, GPIOB, …, GPIOI)
- GPIOx_IDR
  - Data input through the 16 pins
  - Read-only
- GPIOx_ODR
  - Write data to be output to the 16 pins
  - Read last value written to ODR
  - Read/write (for read-modify-write operations)
- C examples:

```
GPIOA->ODR = 0x45;        //send data to output pins
N = GPIOA->IDR;           //copy data from in pins to N
```

Turn on the green led

Load into r2 the address of the GPIO G output data register

```
ldr     r2, =GPIOG_ODR
.Lblink:
```

turn on pin 13 on GPIO G with the next two instructions

```
   movw    r1, LED_GREEN
   str     r1, [r2]
```

do a delay to leave the Green light on

```
   bl      .Ldelay             /* pause */
```

Turn on the red led

turn on pin 14 on GPIO G with the next two instructions

```
movw    r1, LED_RED

    str     r1, [r2]            /* etc  */
    bl      .Ldelay
    bl      .Ldelay
    b       .Lblink
```

Delay function

```
.Ldelay:
    movt    r0, DELAY             /* moving DELAY value
into high halfword of the register  */
loop1:                          /* to make a big number */
     subs     r0, r0, 1             /* and just spend time
substracting */
    bne     loop1
     bx      lr
```

Vector table – linker script

```
MEMORY{
  RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 192K
  ROM (rx)       : ORIGIN = 0x8000000, LENGTH = 2048K
}
SECTIONS
{
  /* The startup code into ROM memory */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >ROM
```