Nastavení bitů v registrech

GPIO Output Type Register (OTYPE)

- 16 bits reserved, 16 data bits, 1 bit for each pin

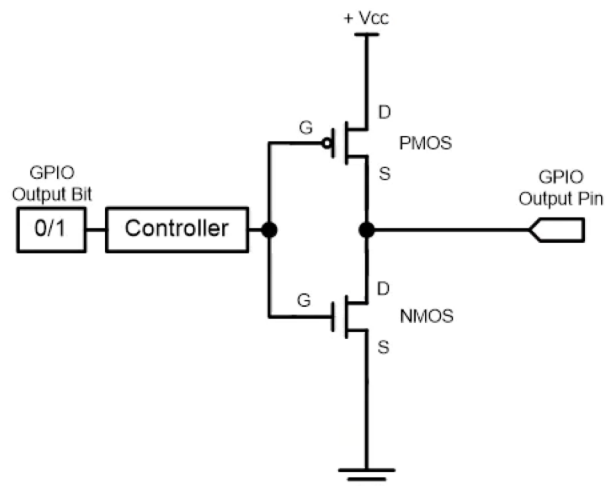
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **OTy:** Port x configuration bits (y = 0..15)

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

0: Output push-pull (reset state)

1: Output open-drain



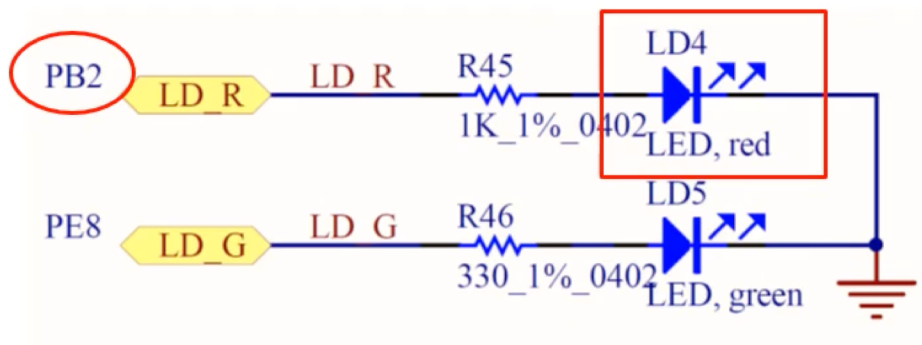

```
GPIOB->OTYPE &= ~(1UL<<2);  // Clear bit 2
```

GPIO Output Data Register (ODR)

- 16 bits reserved, 16 data bits, 1 bit for each pin

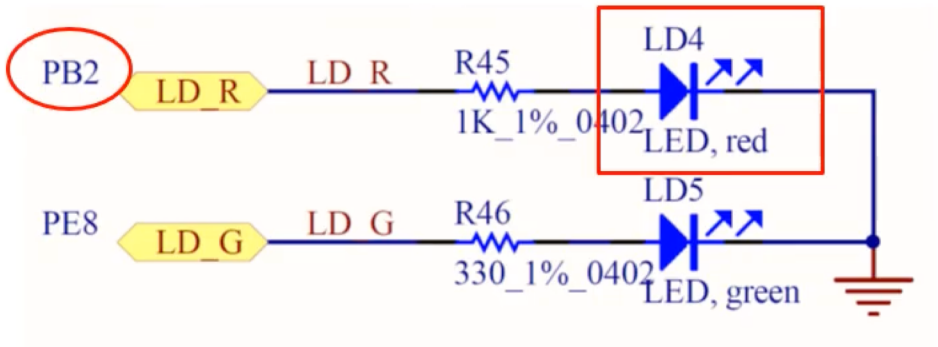
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Pin 2




```
GPIOB->ODR |= 1UL << 2;     // Set bit 2
```

Light up the Red LED (PB.2)



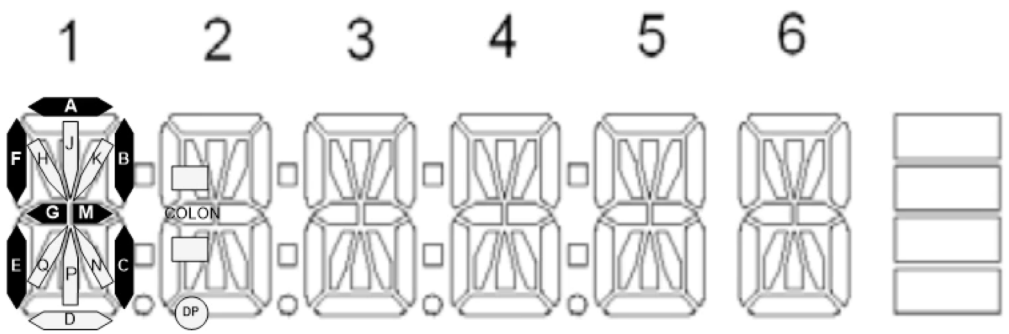

```
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;  // Enable clock of Port B

GPIOB->MODER &= ~(3UL<<4); // Clear mode bits
GPIOB->MODER |=   1UL<<4;  // Set mode to output

GPIOB->OTYPE &= ~(1UL<<2); // Select push-pull output

GPIOB->ODR   |= 1UL << 2;  // Output 1 to turn on red LED
```

Mapping Between Display Memory Bits and Processor Pins




```
LCD->RAM[0] &= ~( 1U << 3 | 1U << 4 | 1U << 22 | 1U << 23 );
LCD->RAM[2] &= ~( 1U << 3 | 1U << 4 | 1U << 22 | 1U << 23 );
LCD->RAM[4] &= ~( 1U << 3 | 1U << 4 | 1U << 22 | 1U << 23 );
LCD->RAM[6] &= ~( 1U << 3 | 1U << 4 | 1U << 22 | 1U << 23 );
LCD->RAM[0] |= 1U << 3 | 1U << 4  | 1U << 22 | 1U << 23;
LCD->RAM[2] |= 1U << 3 | 1U << 22 | 1U << 23;
```

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCD_RAM[0]	4E	4G	3M	3B		6G	5M	5B	1M	1B					6E		3E	3G	2M	2B			6B	6M		2E	2G	1E	1G			
LCD_RAM[1]																													5E	5G	4M	4B
LCD_RAM[2]	4D	4F	3C	3A		6F	5C	5A	1C	1A					6D		3D	3F	2C	2A			6A	6C		2D	2F	1D	1F			
LCD_RAM[3]																													5D	5F	4C	4A
LCD_RAM[4]	4P	4Q	3 Col	3K		6Q	3 Bar	5K	1 Col	1K					6P		3P	3Q	2C ol	2K			6K	1 Bar		2P	2Q	1P	1Q			
LCD_RAM[5]																													5P	5Q	4 Col	4K
LCD_RAM[6]	4N	4H	3 DP	3J		6H	2 Bar	5J	1 DP	1J					6N		3N	3H	2 DP	2J			6J	0 Bar		2N	2H	1N	1H			
LCD_RAM[7]																													5N	5H	4 DP	4J

LCD Driver

- Get the font and modify the display memory: Suppose the font is stored in the array C[4]. The following example controls the display at the first position based on the font.

```
// Clear corresponding bits in the buffer LCD->RAM
// 1G -> Bit 4, 1B -> Bit 23, 1M -> Bit 22, 1E -> Bit 3 in RAM[0]
LCD->RAM[0] &= ~( 1U << 4 | 1U << 23 | 1U << 22 | 1U << 3 );

// 1F -> Bit 4, 1A -> Bit 23, 1C -> Bit 22, 1D -> Bit 3 in RAM[2]
LCD->RAM[2] &= ~( 1U << 4 | 1U << 23 | 1U << 22 | 1U << 3 );

// 1Q -> Bit 4, 1K -> Bit 23, 1Col -> Bit 22, 1P -> Bit 3 in RAM[4]
LCD->RAM[4] &= ~( 1U << 4 | 1U << 23 | 1U << 22 | 1U << 3 );

// 1H -> Bit 4, 1J -> Bit 23, 1DP -> Bit 22, 1N -> Bit 3 in RAM[6]
LCD->RAM[6] &= ~( 1U << 4 | 1U << 23 | 1U << 22 | 1U << 3 );

// Segments: 1G 1B 1M 1E
LCD->RAM[0] |= ((C[0] & 0x1) << 4) | (((C[0] & 0x2) >> 1) << 23) | (((C[0] & 0x4) >> 2) << 22) | (((C[0] & 0x8) >> 3) << 3);

// Segments: 1F 1A 1C 1D
LCD->RAM[2] |= ((C[1] & 0x1) << 4) | (((C[1] & 0x2) >> 1) << 23) | (((C[1] & 0x4) >> 2) << 22) | (((C[1] & 0x8) >> 3) << 3);

// Segments: 1Q 1K 1Col 1P
LCD->RAM[4] |= ((C[2] & 0x1) << 4) | (((C[2] & 0x2) >> 1) << 23) | (((C[2] & 0x4) >> 2) << 22) | (((C[2] & 0x8) >> 3) << 3);

// Segments: 1H 1J 1DP 1N
LCD->RAM[6] |= ((C[3] & 0x1) << 4) | (((C[3] & 0x2) >> 1) << 23) | (((C[3] & 0x4) >> 2) << 22) | (((C[3] & 0x8) >> 3) << 3);
```

KONEC