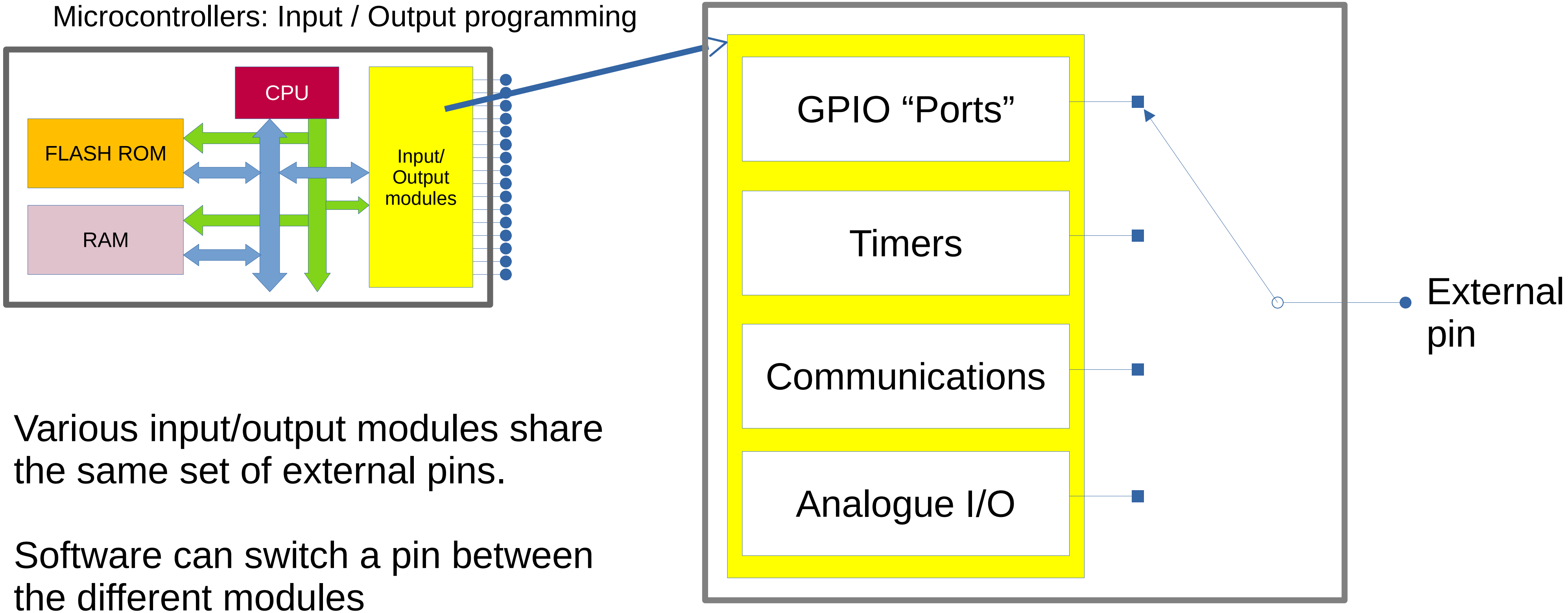
Input/Output programming

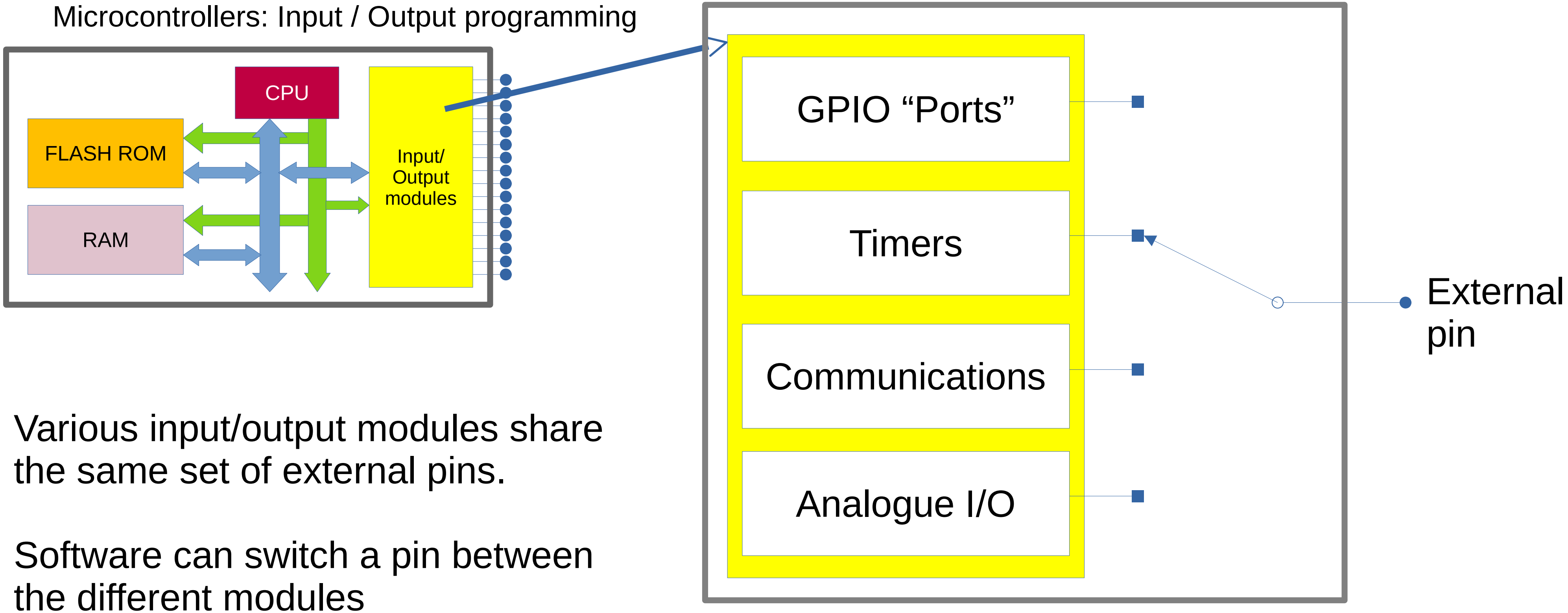
Microcontrollers: Input / Output programming



Various input/output modules share the same set of external pins.

Software can switch a pin between the different modules

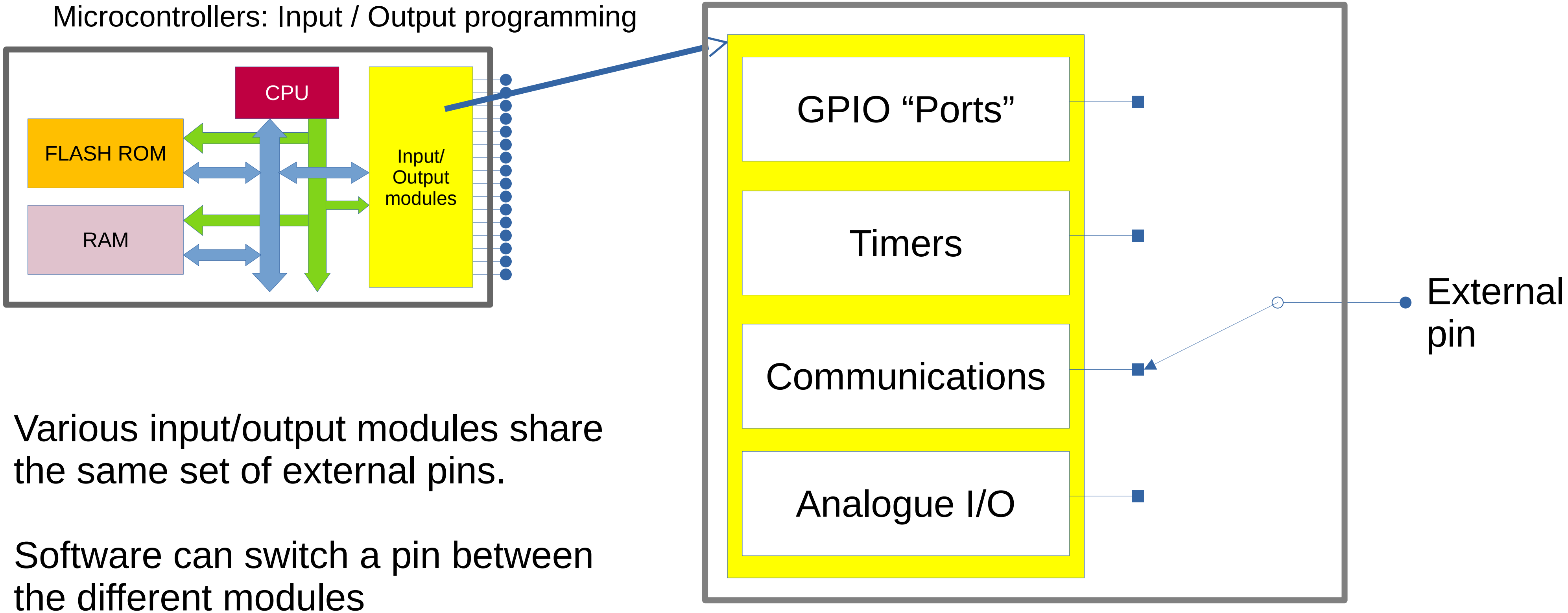
Microcontrollers: Input / Output programming



Various input/output modules share the same set of external pins.

Software can switch a pin between the different modules

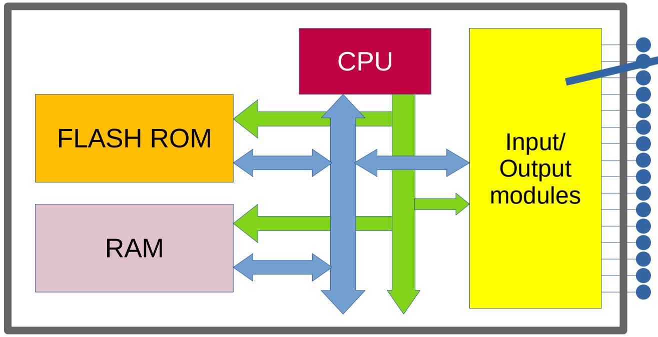
Microcontrollers: Input / Output programming

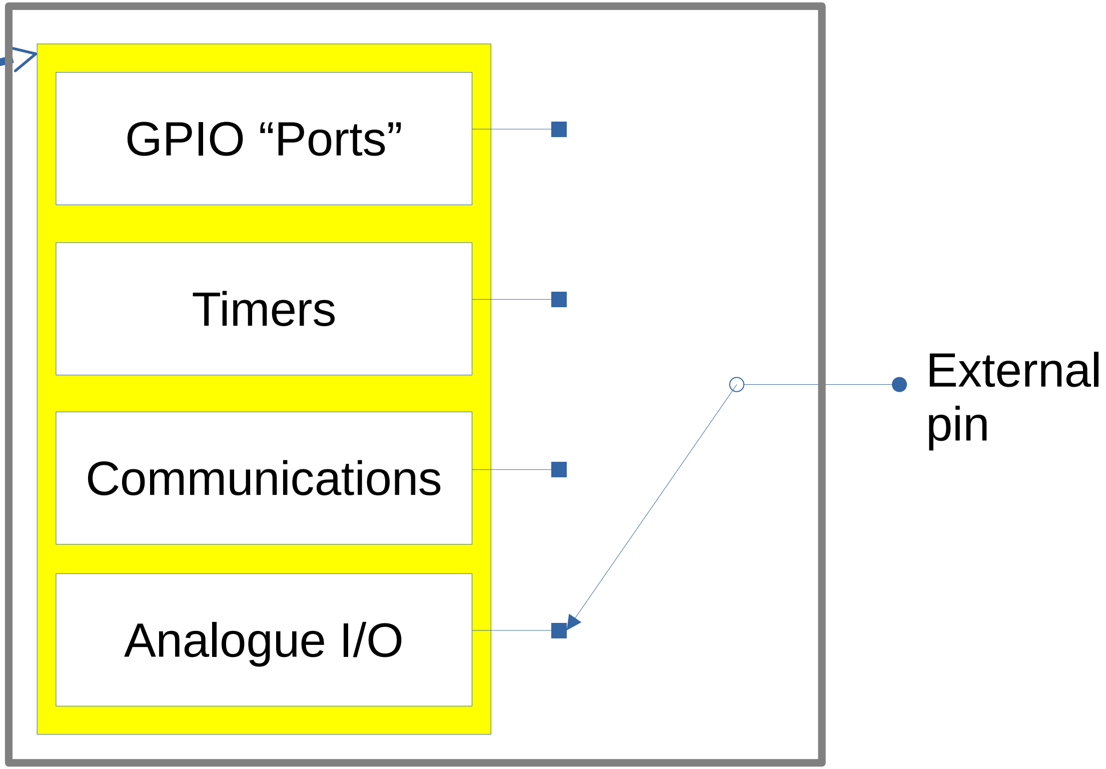


Various input/output modules share the same set of external pins.

Software can switch a pin between the different modules

Microcontrollers: Input / Output programming





Various input/output modules share the same set of external pins.

Software can switch a pin between the different modules

Microcontrollers: Input / Output programming

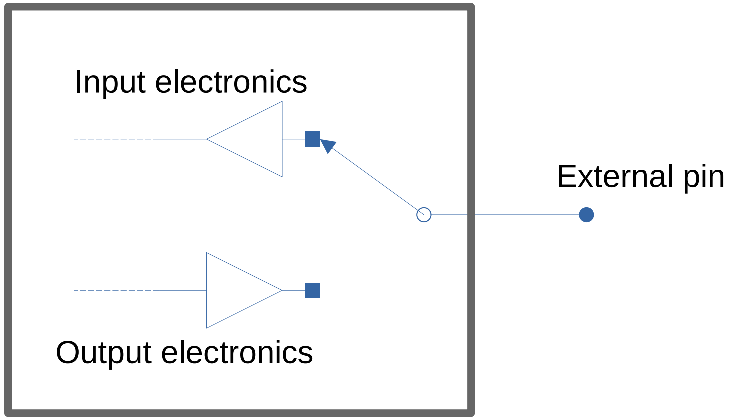
GPIO “Ports”

Timers

Communications

Analogue I/O

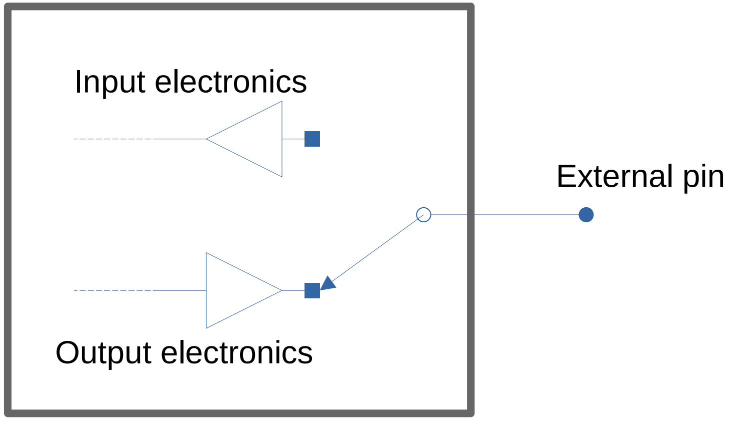
GPIO pins can be configured to be digital inputs (the default case)



Microcontrollers: Input / Output programming

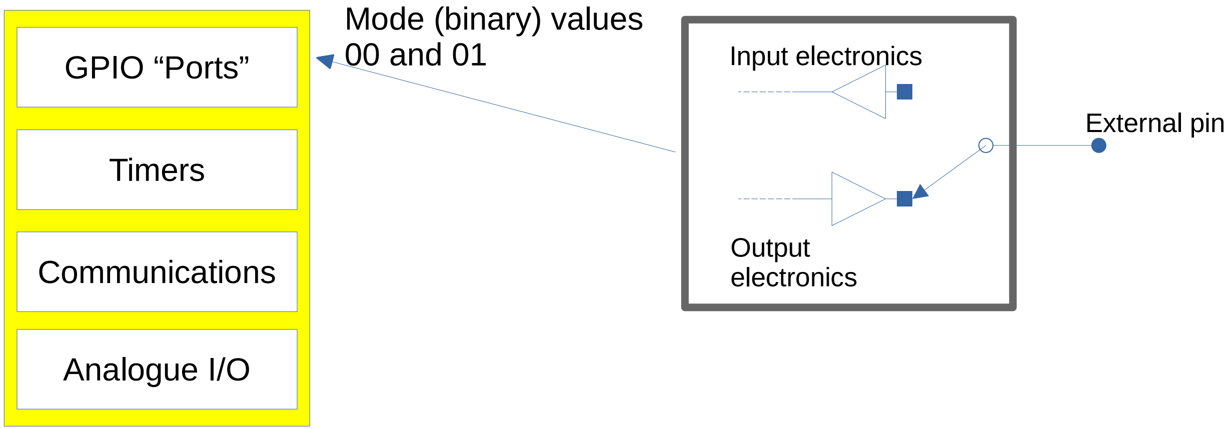
- GPIO “Ports”
- Timers
- Communications
- Analogue I/O

Or as outputs



Microcontrollers: Input / Output programming

Or as outputs



GPIO “Ports”

Timers

Communications

Analogue I/O

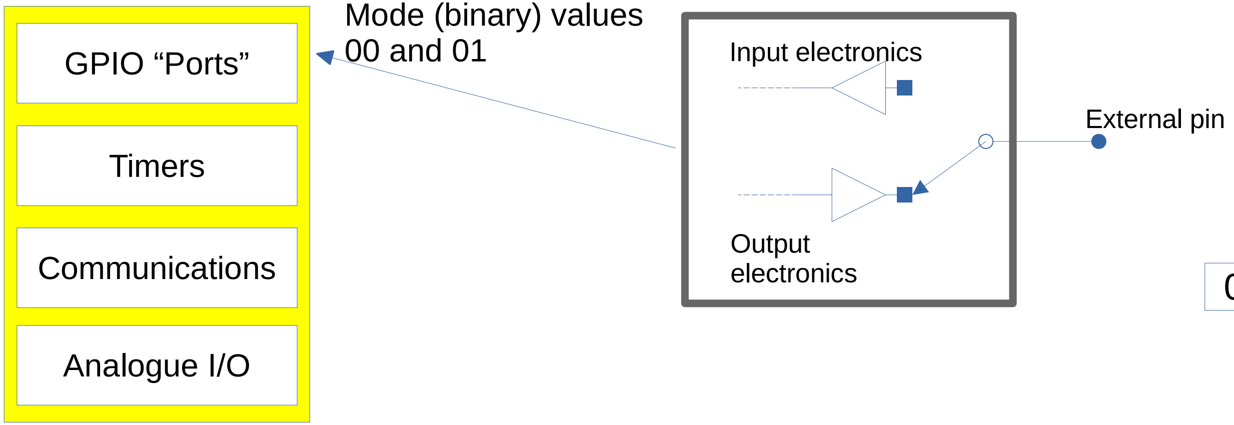
Mode (binary) values
00 and 01

Input electronics

External pin

Output
electronics

Microcontrollers: Input / Output programming



GPIO “Ports”

Timers

Communications

Analogue I/O

Mode (binary) values
00 and 01

Or as outputs

Mode register for GPIOA

0	0	0	0		0	0	0	0	0	0	0	0

Two bits are used to configure each pin. This gives 4 possibilities for each pin of GPIOA:

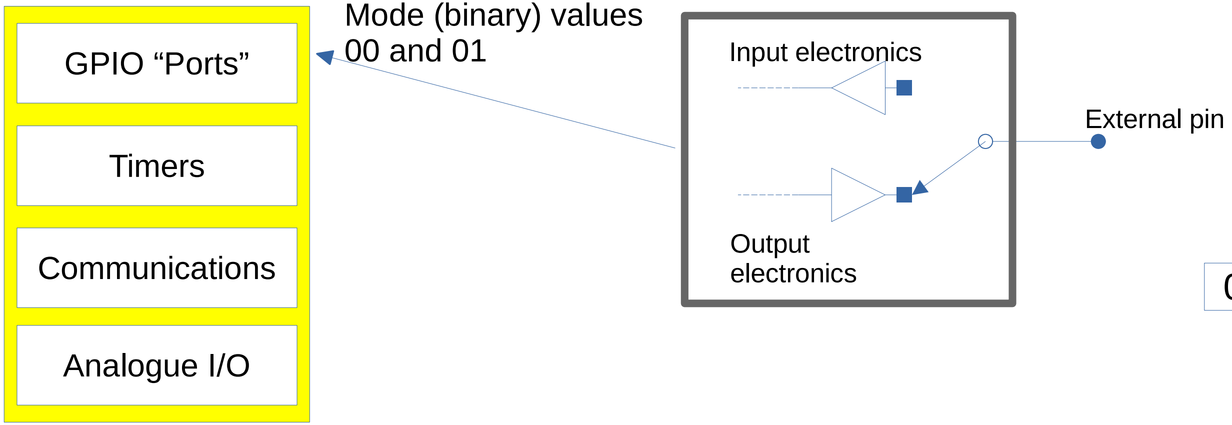
00 = simple digital input

01 = simple digital output

10 = alternative function

11 = analogue input

Microcontrollers: Input / Output programming



Or as outputs

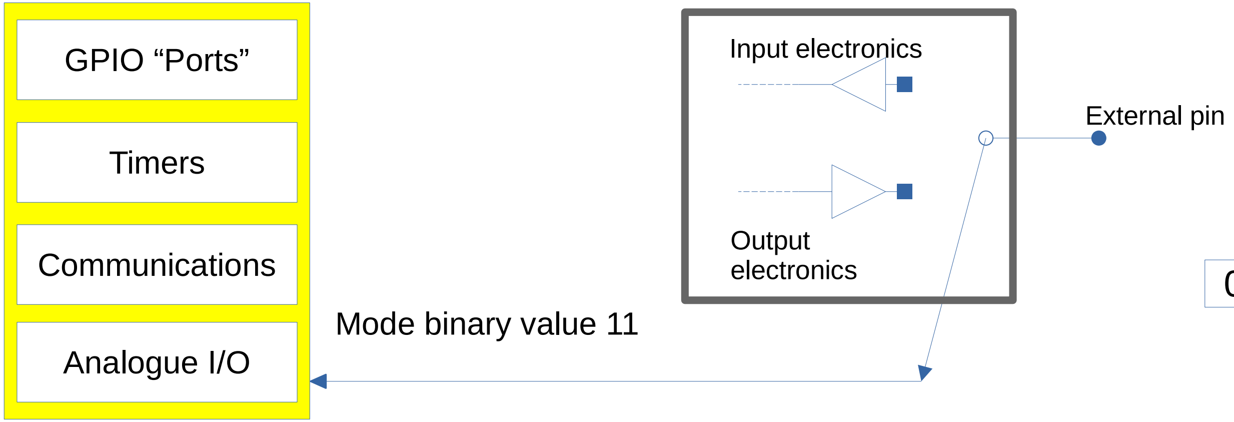
Mode register for GPIOA

0	0	0	0		0	0	0	0	0	0	0	1

Two bits are used to configure each pin. This gives 4 possibilities for each pin of GPIOA:

00 = simple digital input
01 = simple digital output
10 = alternative function
11 = analogue input

Microcontrollers: Input / Output programming



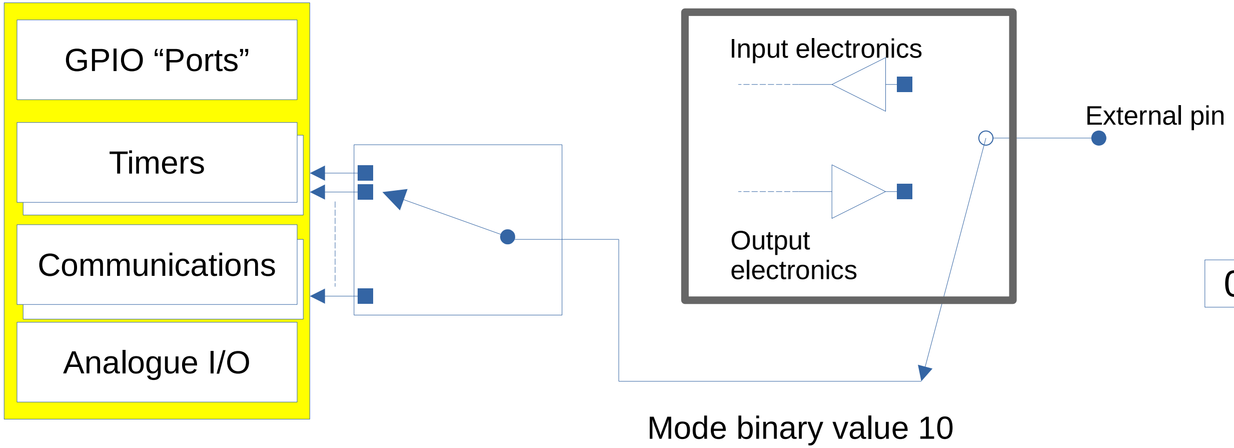
Or as outputs

Mode register for GPIOA

0	0	0	0		0	0	0	0	0	0	1	1

Two bits are used to configure each pin. This gives 4 possibilities for each pin of GPIOA:
00 = simple digital input
01 = simple digital output
10 = alternative function
11 = analogue input

Or as outputs



Mode register for GPIOA

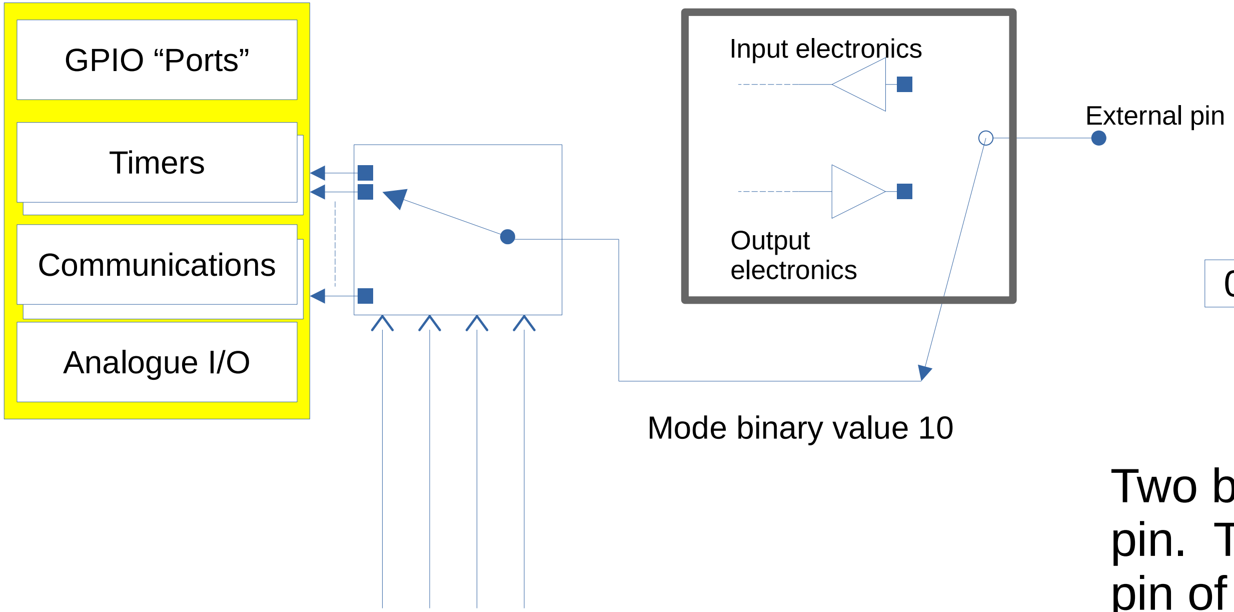
0	0	0	0		0	0	0	0	0	0	1	0

Two bits are used to configure each pin. This gives 4 possibilities for each pin of GPIOA:

00 = simple digital input
01 = simple digital output
10 = alternative function
11 = analogue input

Microcontrollers: Input / Output programming

Or as outputs



4 Further control bits for each pin in AFR[0],AFR[1]

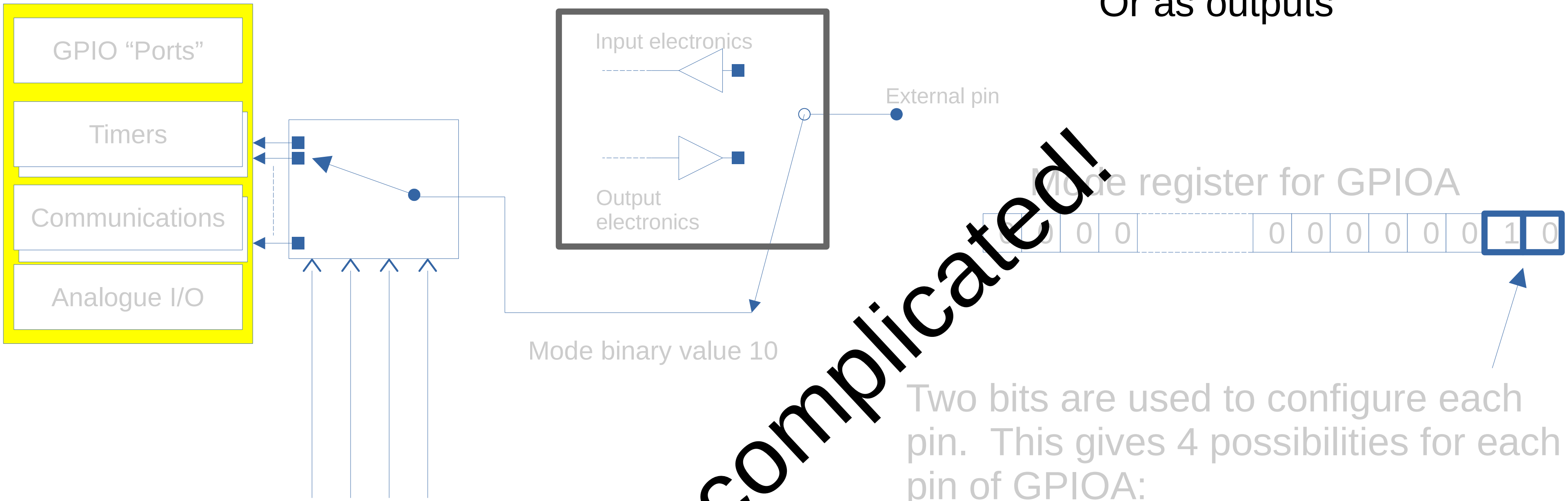
Mode register for GPIOA

0	0	0	0		0	0	0	0	0	0	1	0

Two bits are used to configure each pin. This gives 4 possibilities for each pin of GPIOA:
00 = simple digital input
01 = simple digital output
10 = alternative function
11 = analogue input

Microcontrollers: Input / Output programming

Or as outputs



4 Further control bits for each pin in AFR[0],AFR[1]

Two bits are used to configure each pin. This gives 4 possibilities for each pin of GPIOA:

00 = simple digital input

01 = simple digital output

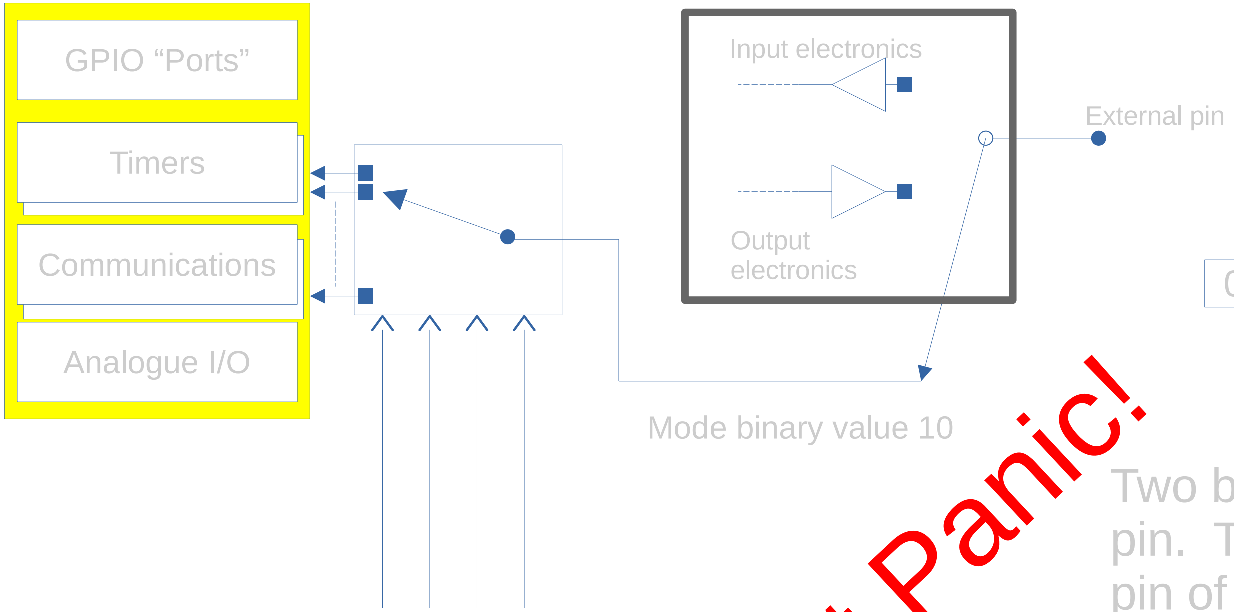
10 = alternative function

11 = analogue input

Very complicated!

Microcontrollers: Input / Output programming

Or as outputs



4 Further control bits for each pin in AFR[0],AFR[1]

Don’t Panic!

Mode register for GPIOA

0	0	0	0		0	0	0	0	0	0	1	0

Two bits are used to configure each pin. This gives 4 possibilities for each pin of GPIOA:

00 = simple digital input
01 = simple digital output
10 = alternative function
11 = analogue input

We can remove the complexity and reduce the probability of error by using the helper function:

```
void pinMode(GPIO_TypeDef *Port, uint32_t BitNumber, uint32_t Mode)
{
    /*    Mode values :      0b00 = Digital input
                             0b01 = Digital output
                             0b10 = Alternative function
                             0b11 = Analog input
    */
    uint32_t mode_value = Port->MODER;
    Mode = Mode << (2 * BitNumber);
    mode_value = mode_value & ~(3u << (BitNumber * 2));
    mode_value = mode_value | Mode;
    Port->MODER = mode_value;
}
```

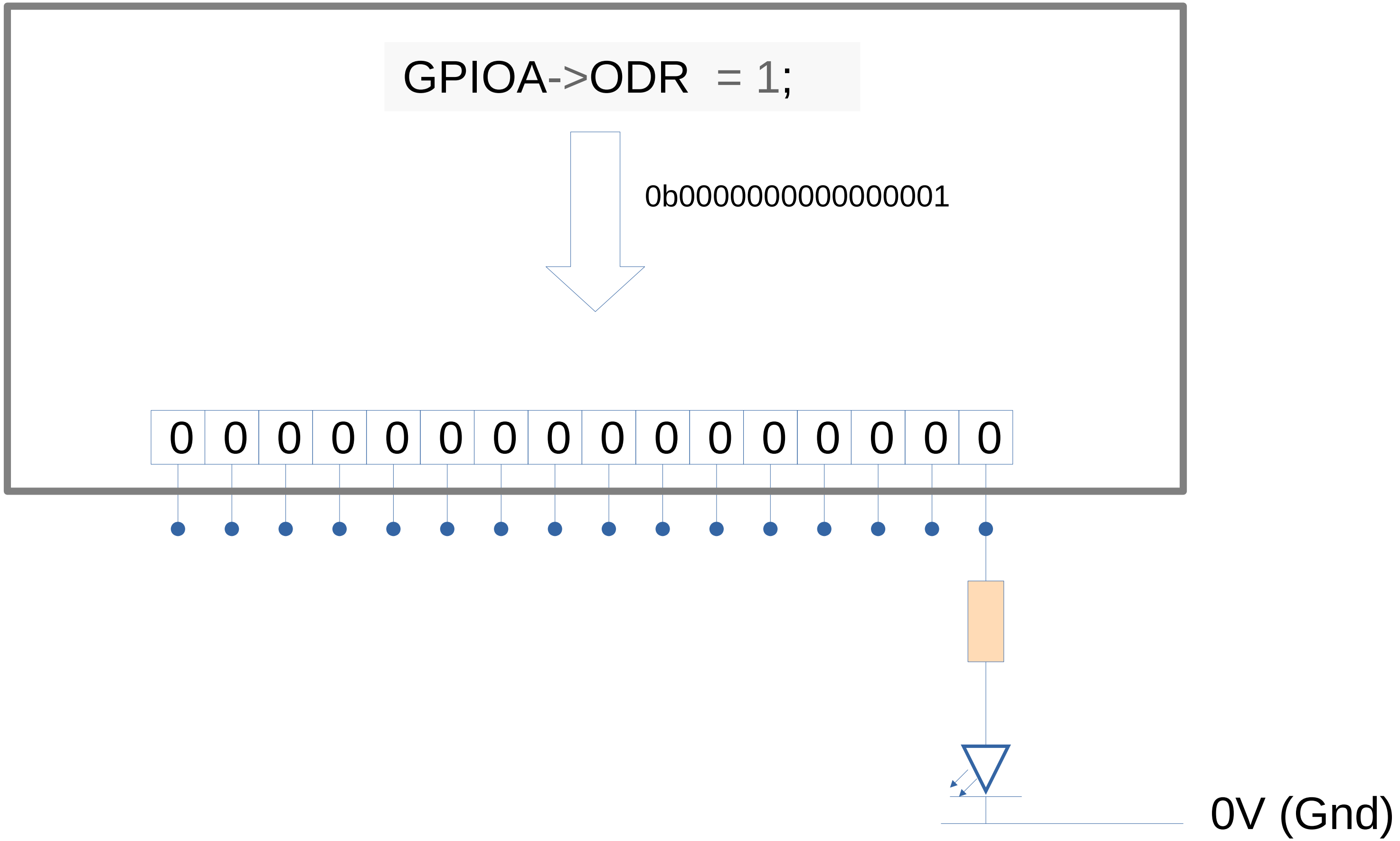
```
pinMode(GPIOA,4,1); // make PA2 a digital output

pinMode(GPIOB,7,0); // make PB7 a digital input

pinMode(GPIOC,14,2);  // assign one of the Alternate functions to PC14
                      // requires a subsequent write to AFR register
```

We will be mostly only use pins as digital inputs, outputs and analogue inputs so mostly we will use mode values of 0,1, and 3.

Controlling individual output bits

```
GPIOA->ODR  = 1;
```

Controlling individual output bits

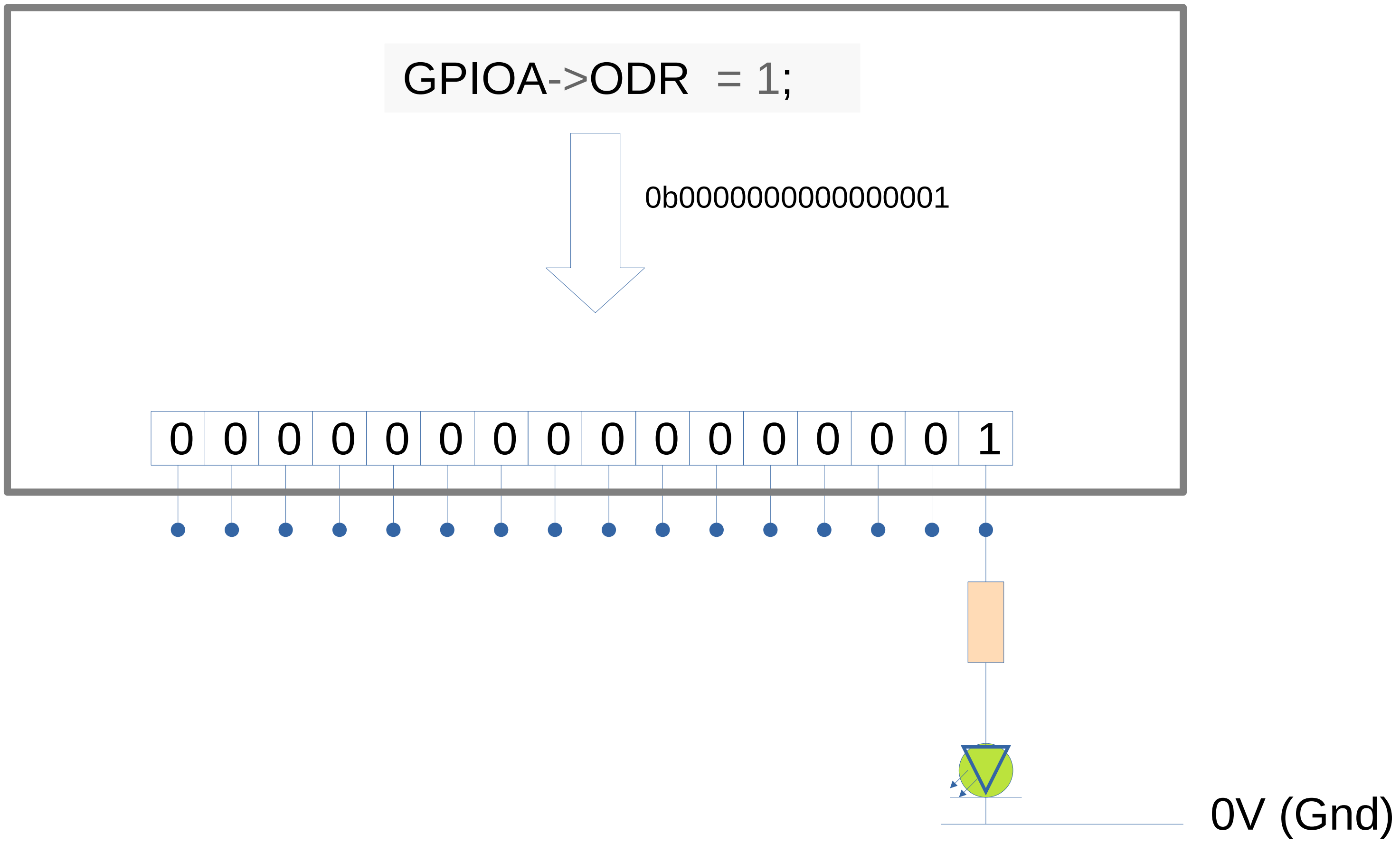


```
GPIOA->ODR  = 1;
```

0b0000000000000001

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Controlling individual output bits

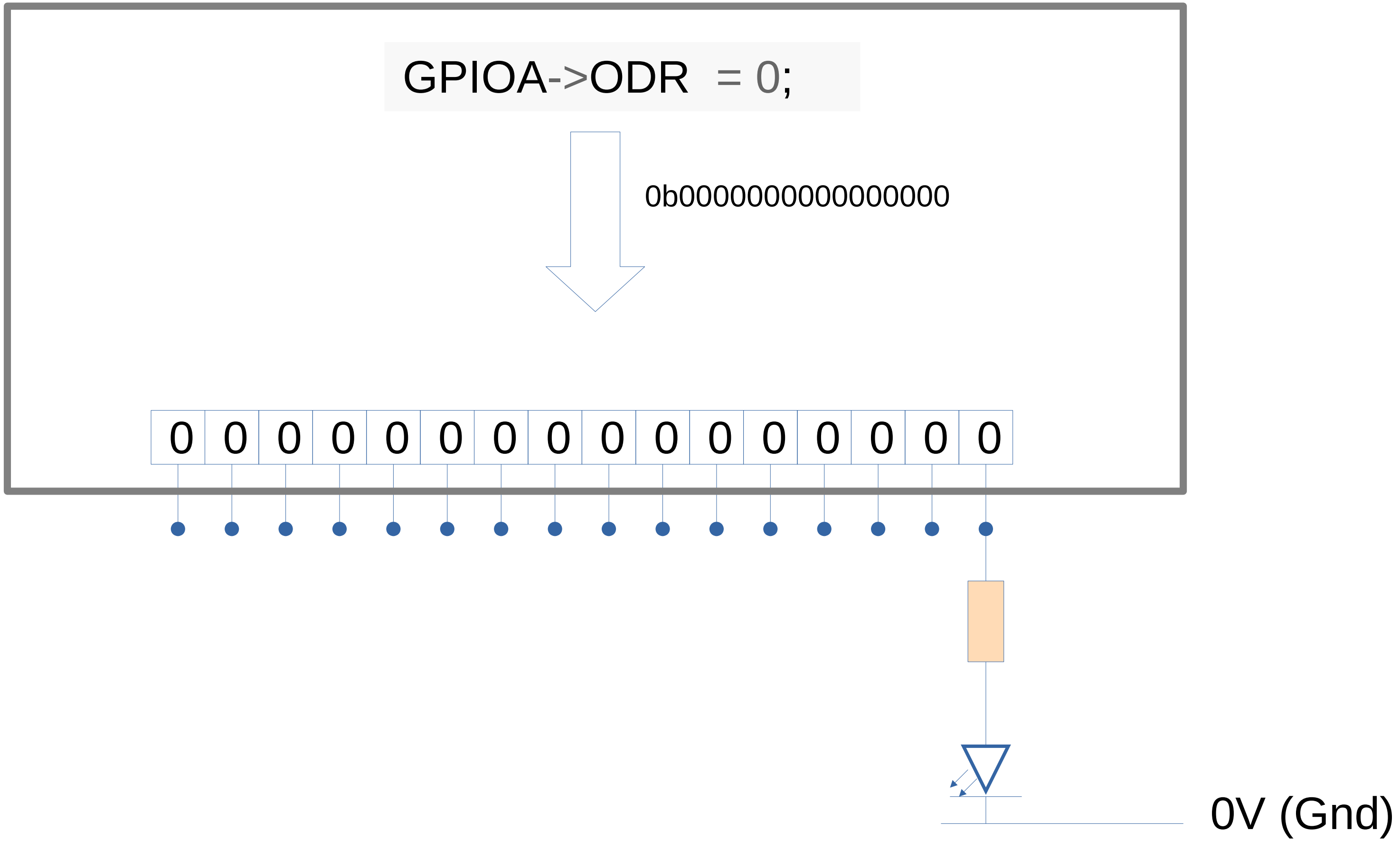


```
GPIOA->ODR  = 1;
```

0b0000000000000001

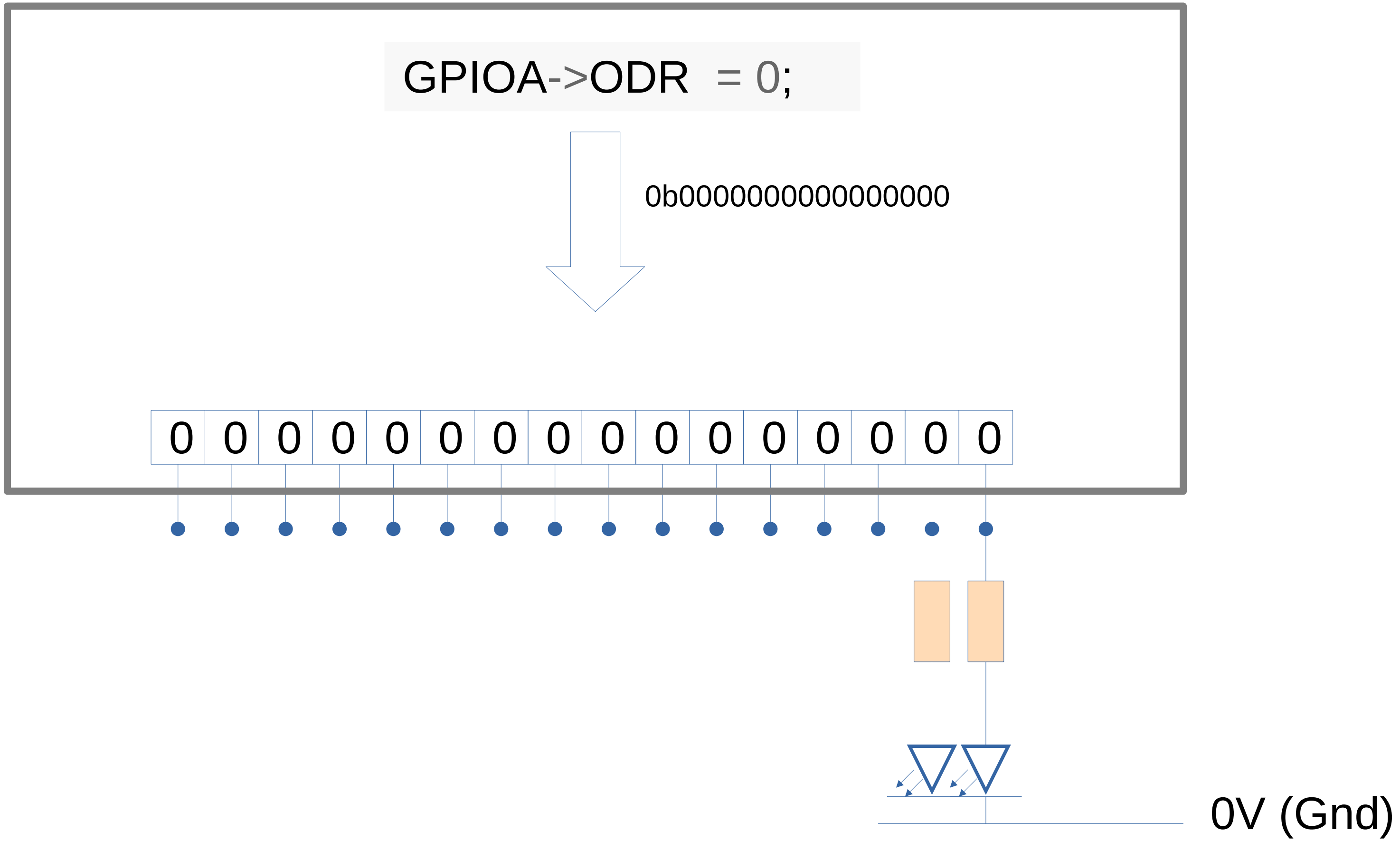
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Controlling individual output bits



```
GPIOA->ODR  = 0;
```

Controlling individual output bits

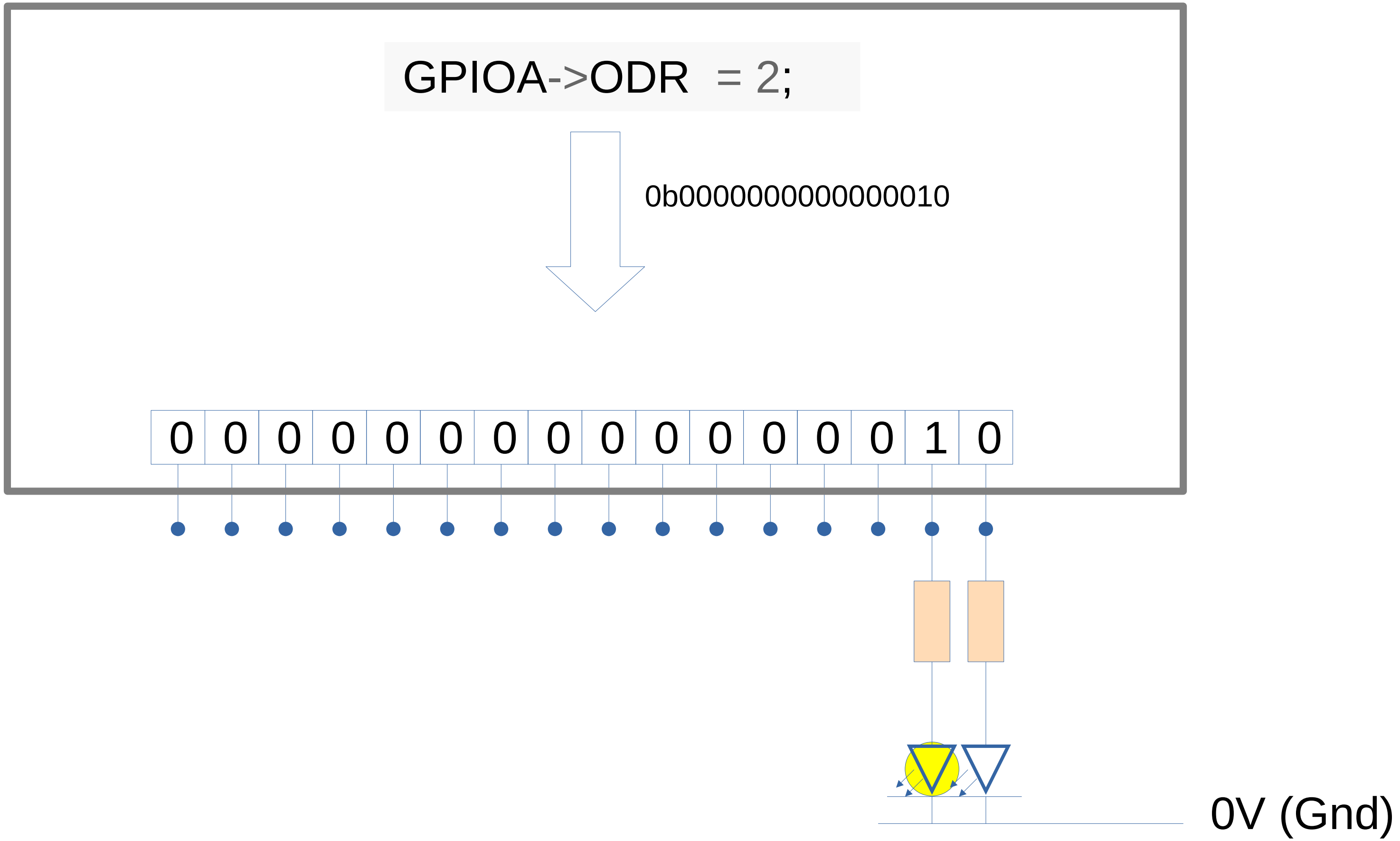


```
GPIOA->ODR  = 0;
```

0b0000000000000000

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Controlling individual output bits



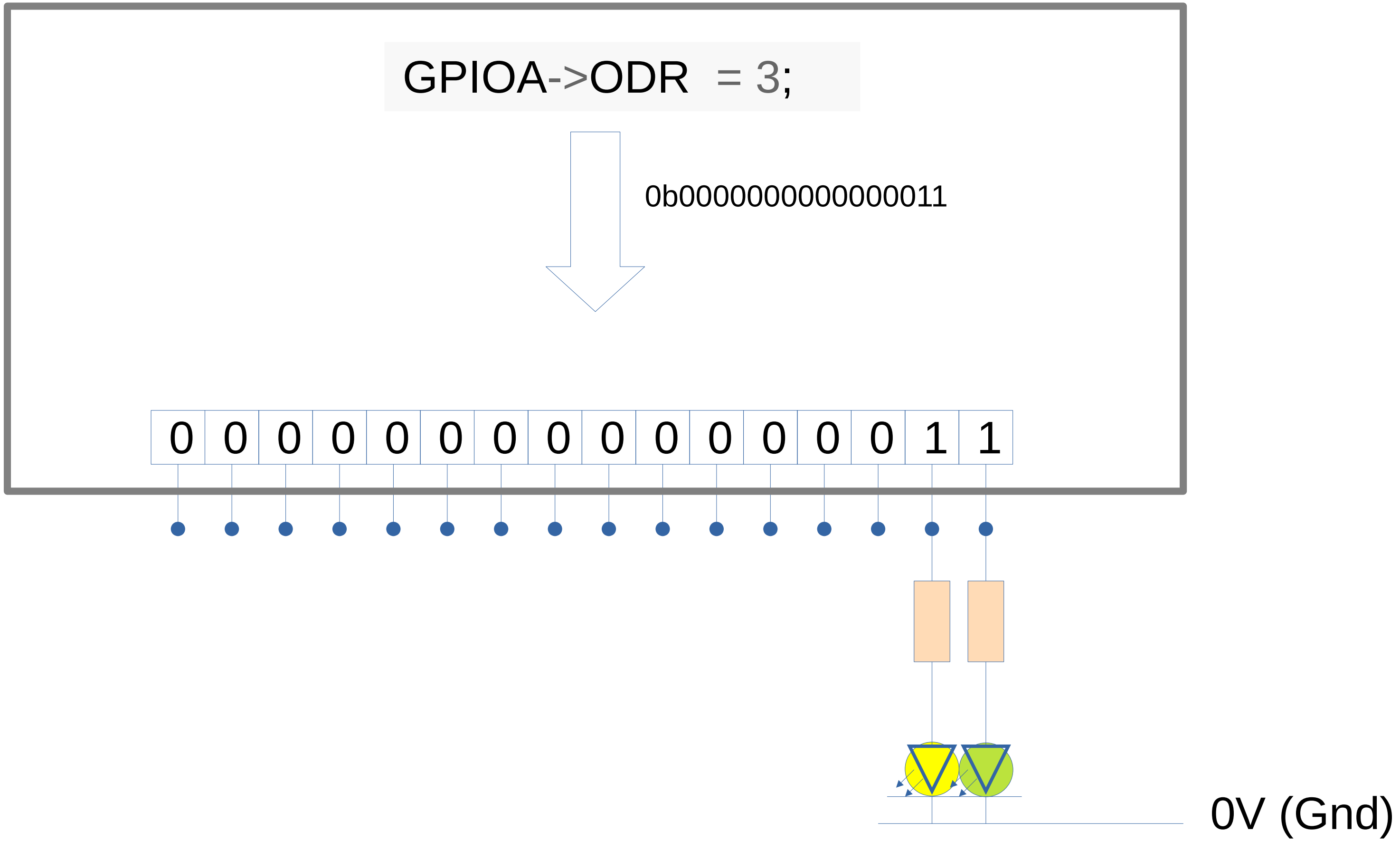
```
GPIOA->ODR  = 2;
```

0b0000000000000010

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

0V (Gnd)

Controlling individual output bits



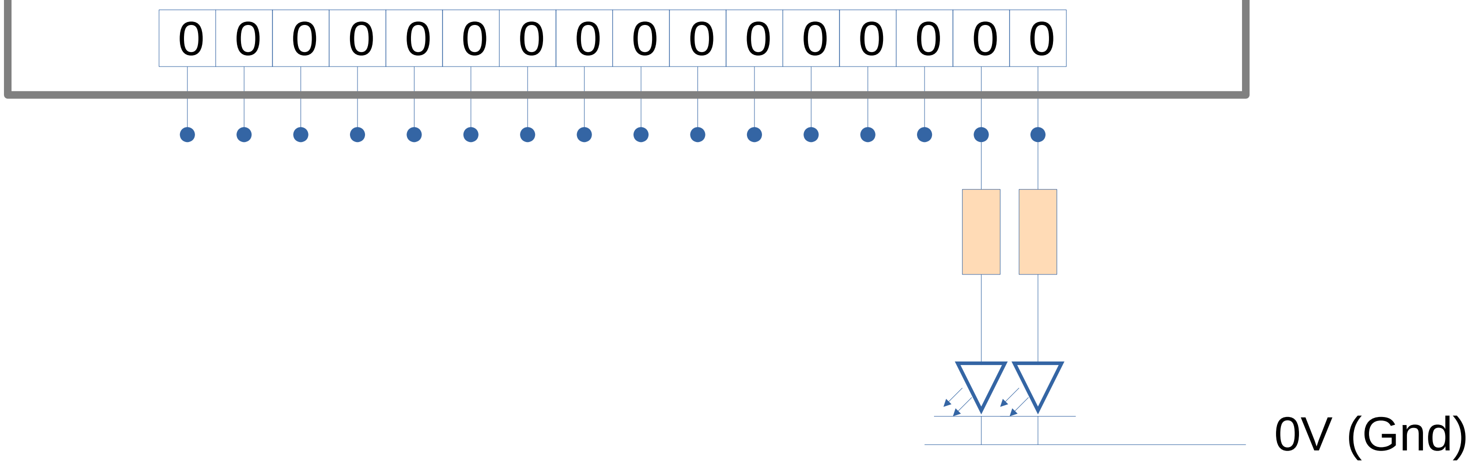
```
GPIOA->ODR  = 3;
```

0b0000000000000011

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1

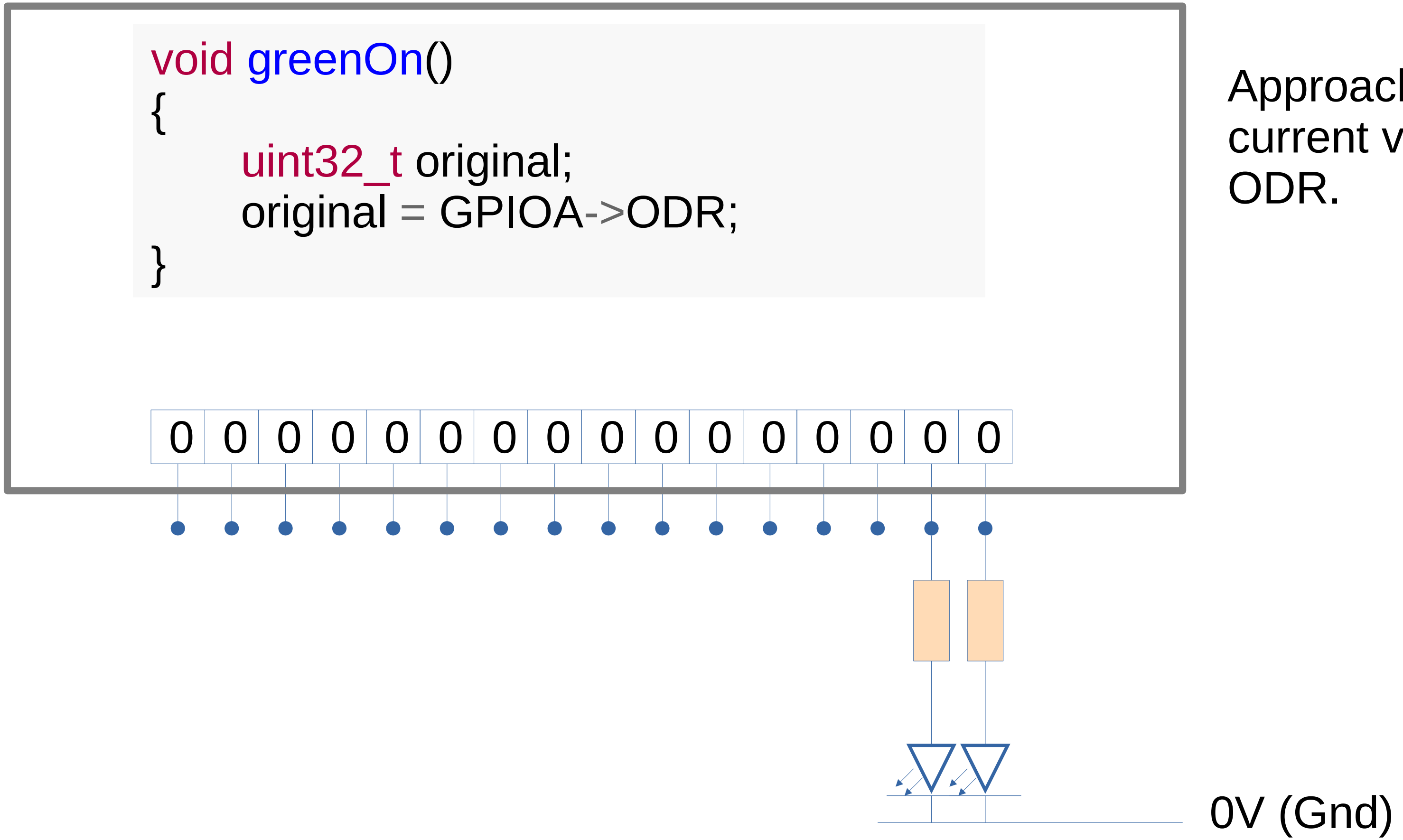
Controlling individual output bits

```
void greenOn()
{

}
```



Can we write a function greenOn that will turn on the green LED without affecting other bits in the ODR?

Controlling individual output bits



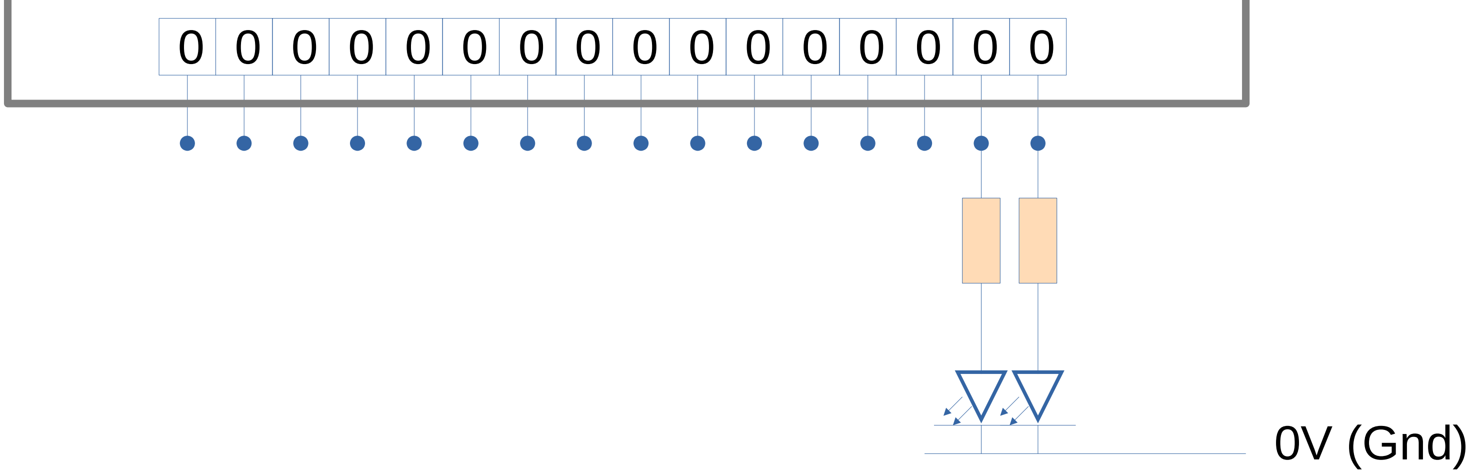
```
void greenOn()
{
        uint32_t original;
        original = GPIOA->ODR;
}
```

Approach: Read the current value in ODR.

Controlling individual output bits

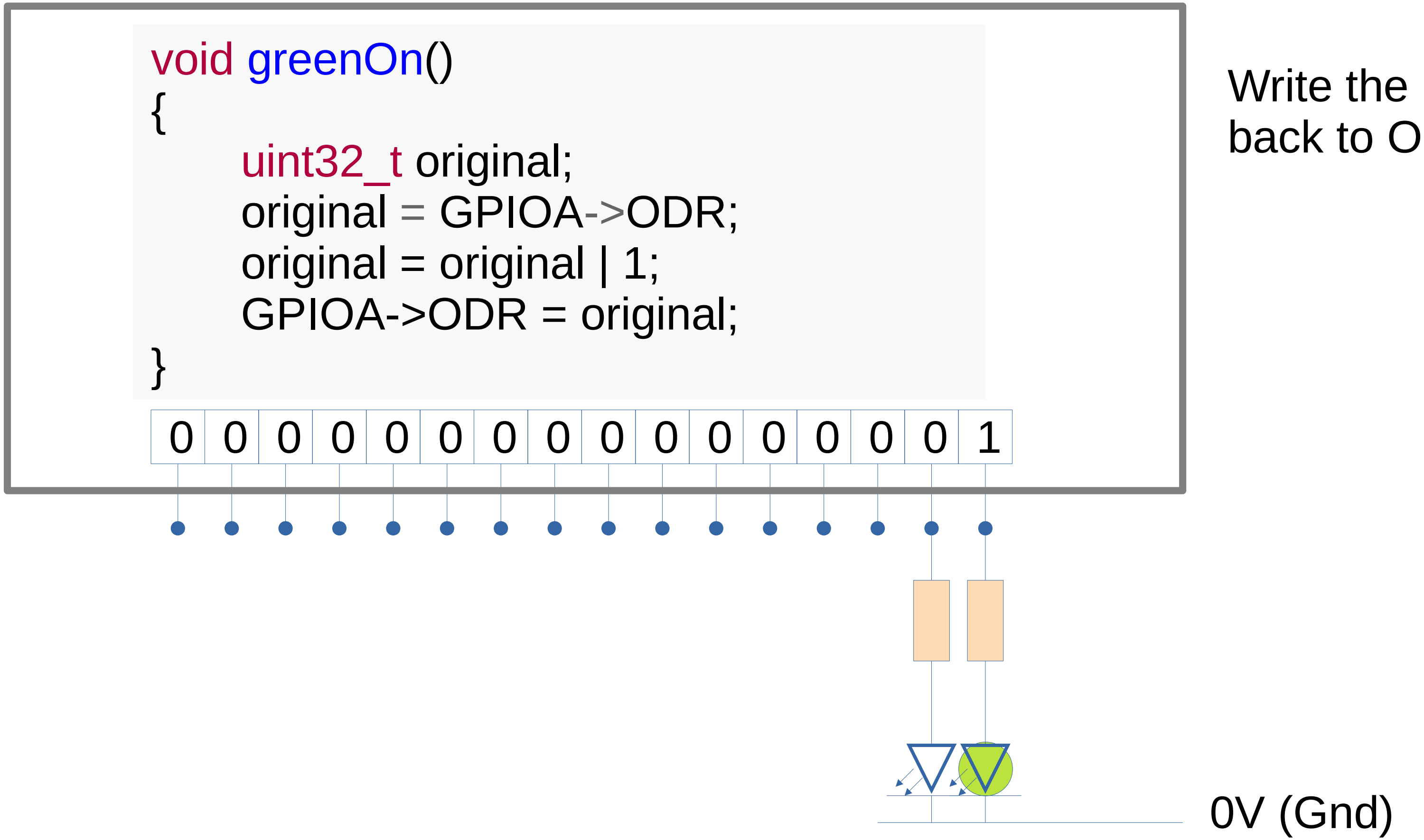
```
void greenOn()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original | 1;
}
```

Perform a bitwise OR



0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Controlling individual output bits



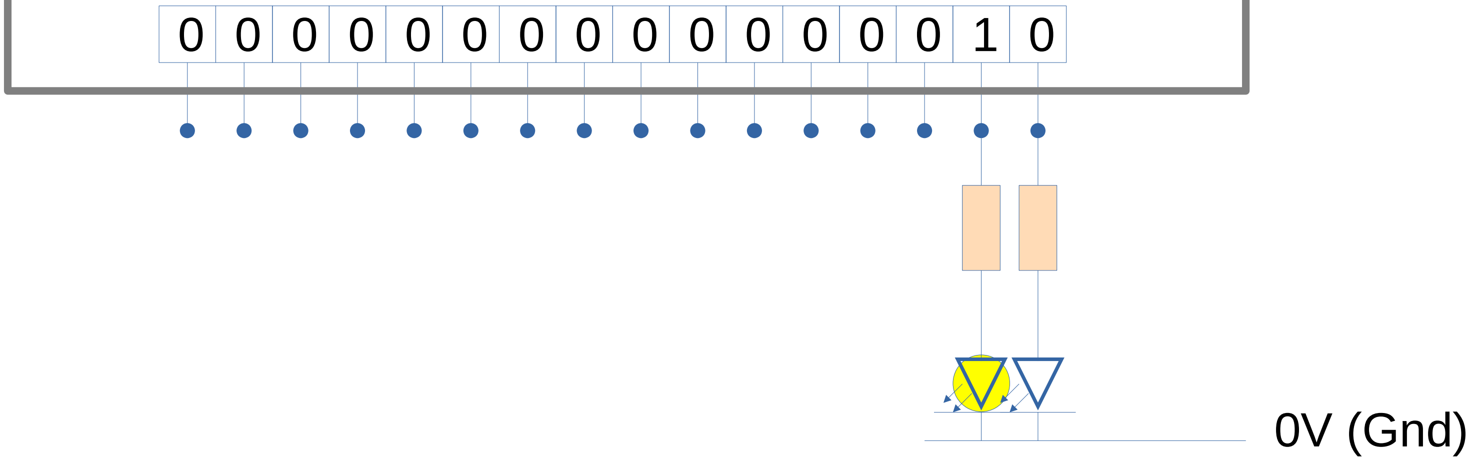
```
void greenOn()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original | 1;
        GPIOA->ODR = original;
}
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Write the result back to ODR

Controlling individual output bits

```
void greenOn()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original | 1;
        GPIOA->ODR = original;
}
```



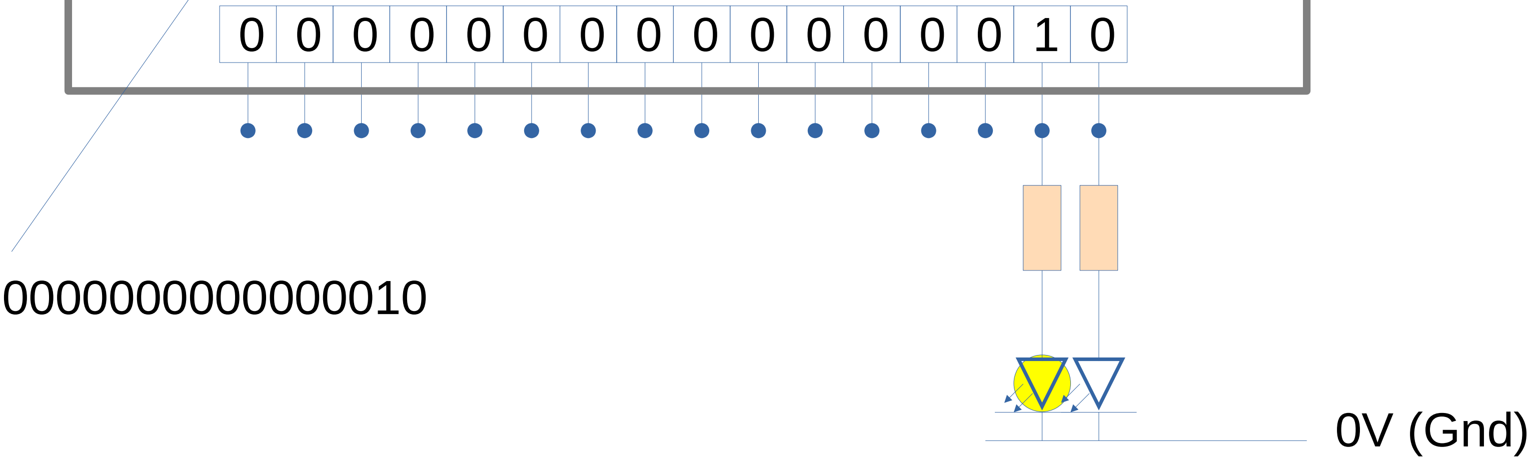
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

What if the yellow LED was previously on?

Controlling individual output bits

```
void greenOn()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original | 1;
        GPIOA->ODR = original;
}
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

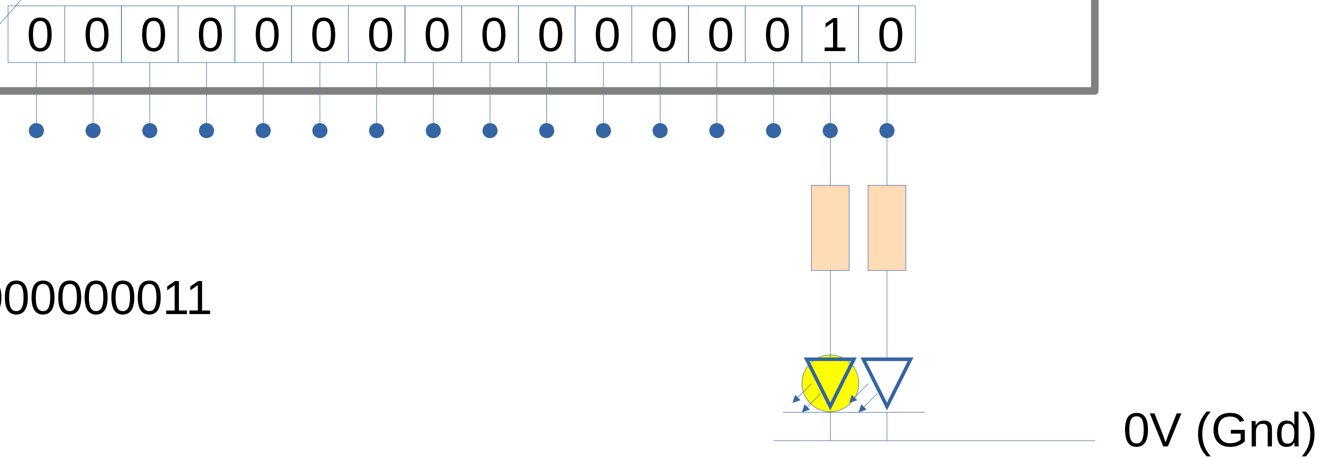


What if the yellow LED was previously on?

Controlling individual output bits

```
void greenOn()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original | 1;
        GPIOA->ODR = original;
}
```

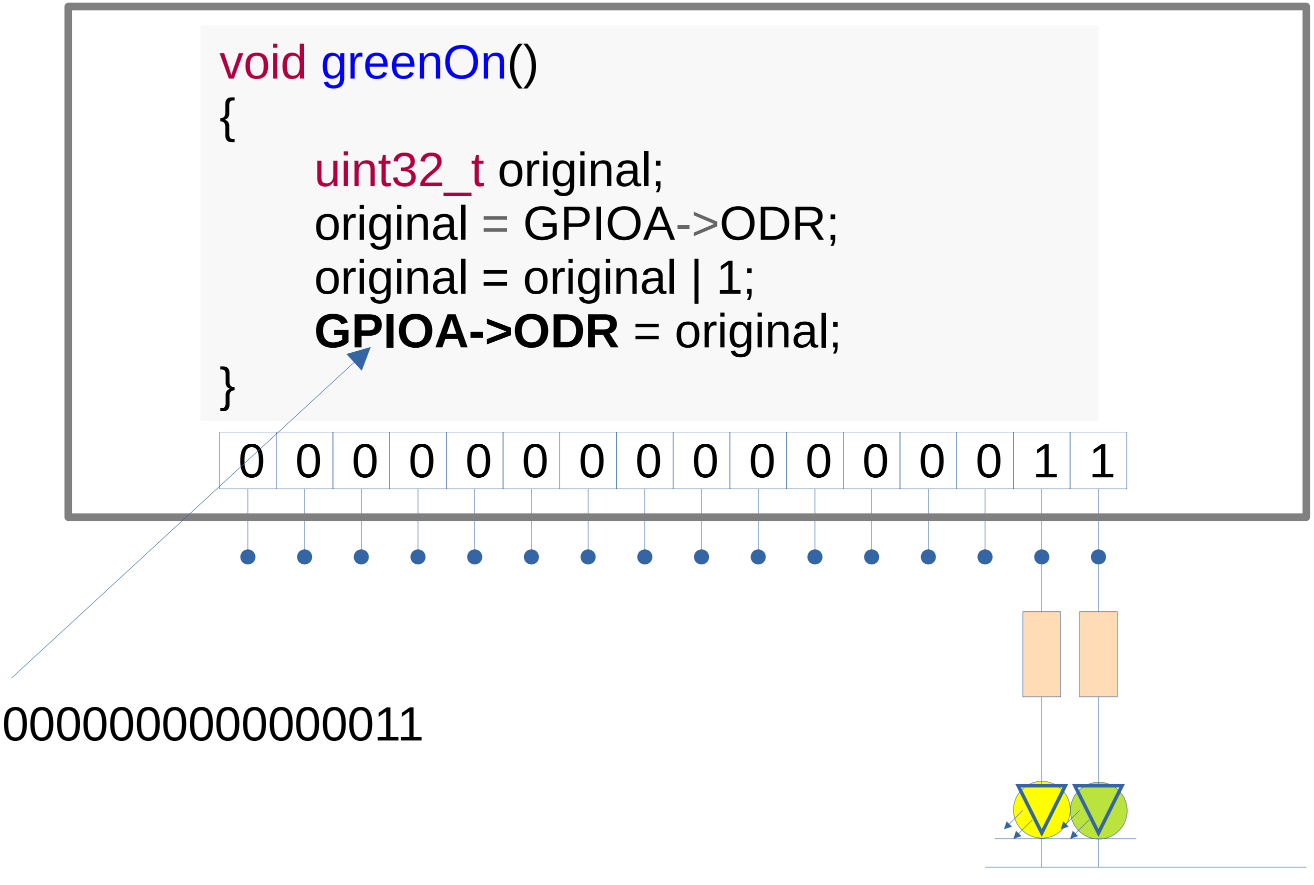
What if the yellow LED was previously on?



0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

0000000000000011

Controlling individual output bits



```
void greenOn()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original | 1;
        GPIOA->ODR = original;
}
```

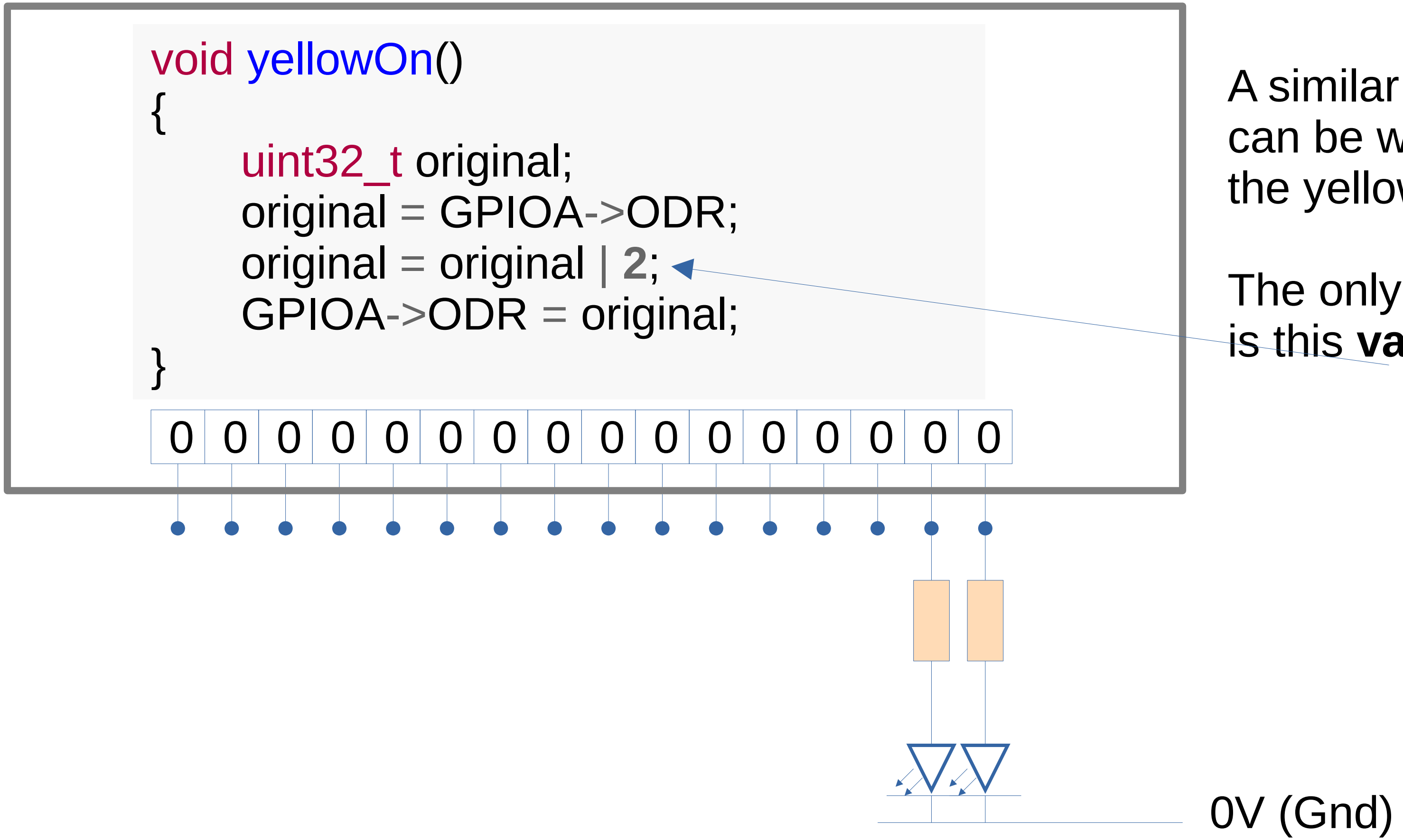
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

0000000000000011

0V (Gnd)

We can turn the green LED on without affecting the yellow LED or other outputs.

Controlling individual output bits



```
void yellowOn()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original | 2;
        GPIOA->ODR = original;
}
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

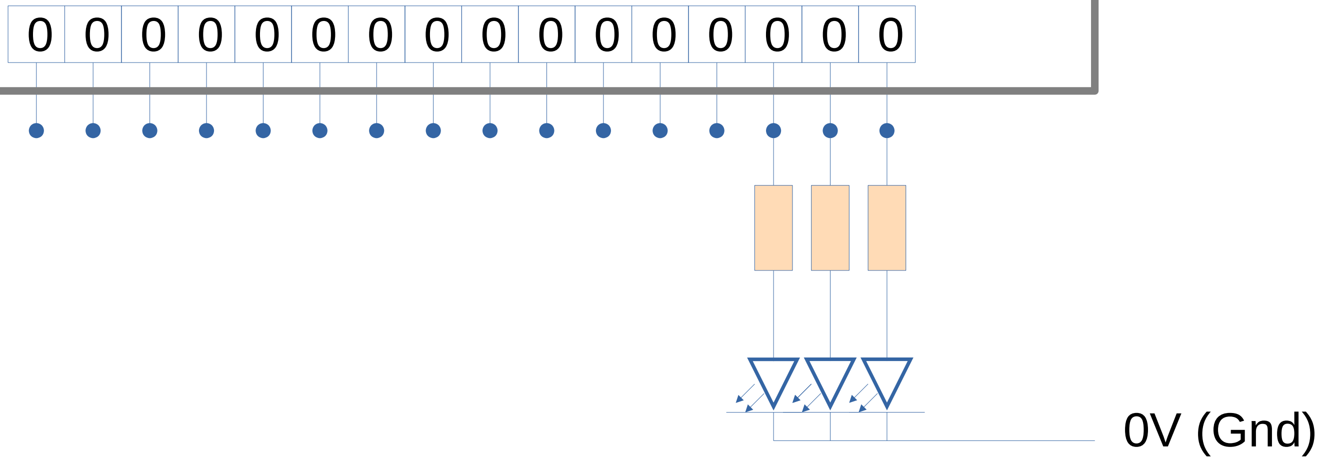
A similar function can be written for the yellow LED.

The only difference is this **value**

Controlling individual output bits

```
void redOn()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original | ??;
        GPIOA->ODR = original;
}
```

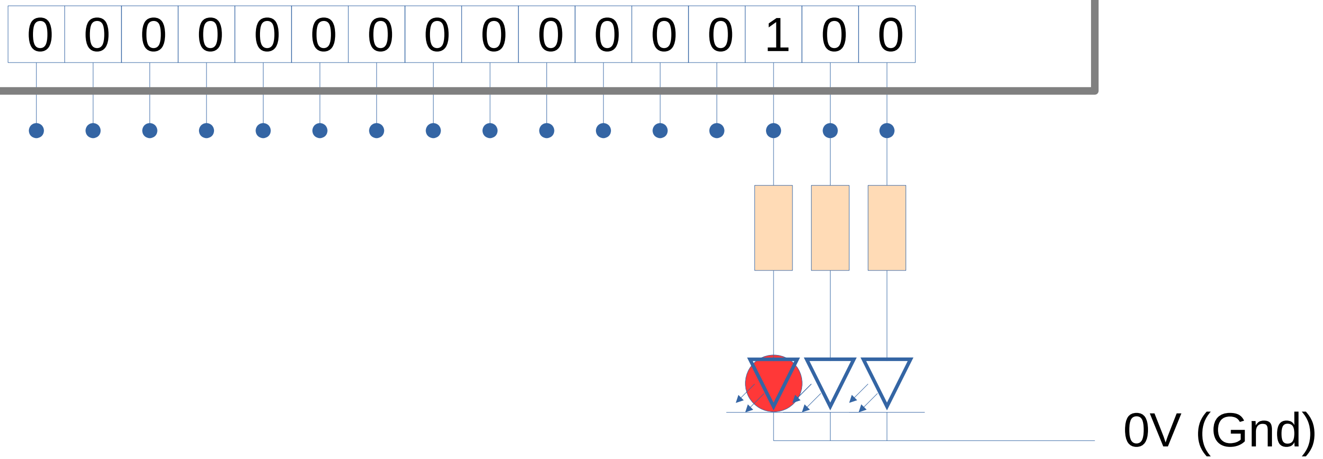
What value do we use for the next LED?



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Controlling individual output bits

```
void redOn()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original | 4;
        GPIOA->ODR = original;
}
```

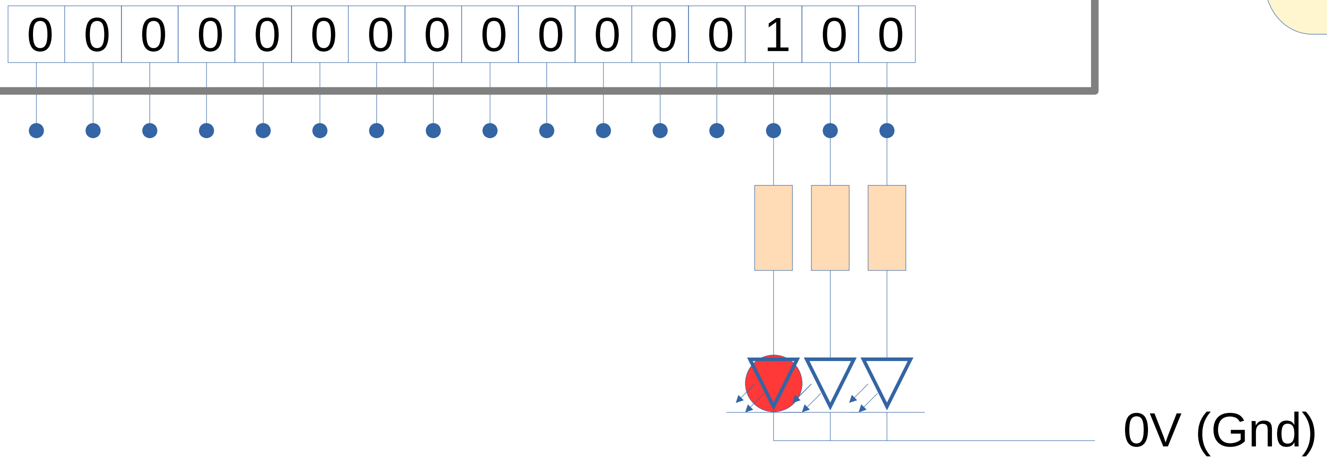


```
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
```

Controlling individual output bits

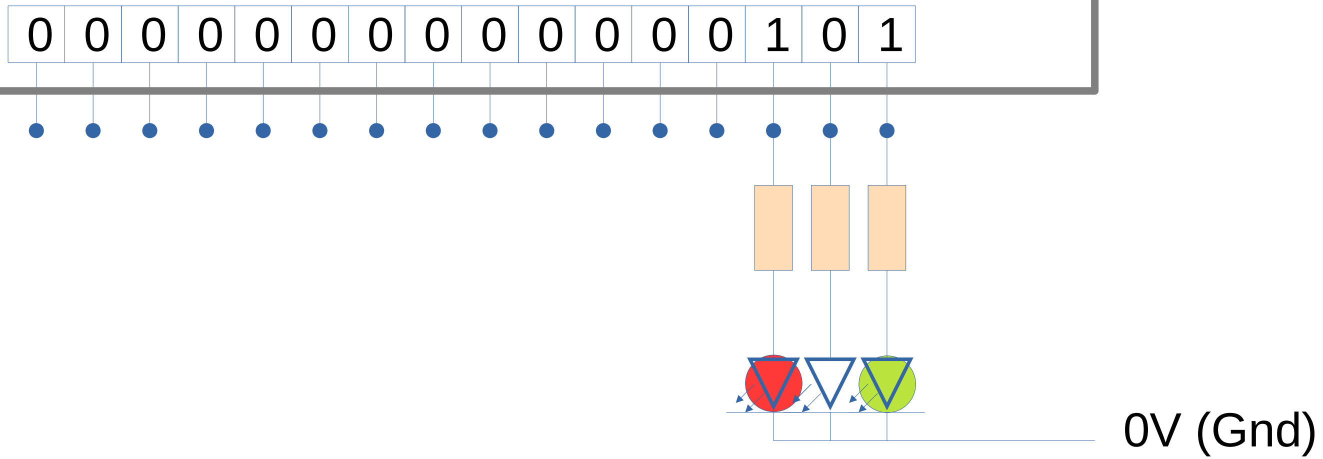
```
void redOn()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original | 4;
        GPIOA->ODR = original;
}
```

These values are usually called **masks**



Controlling individual output bits

```
void redOff()
{
        uint32_t original;
        original = GPIOA->ODR;
        ??????
        GPIOA->ODR = original;
}
```

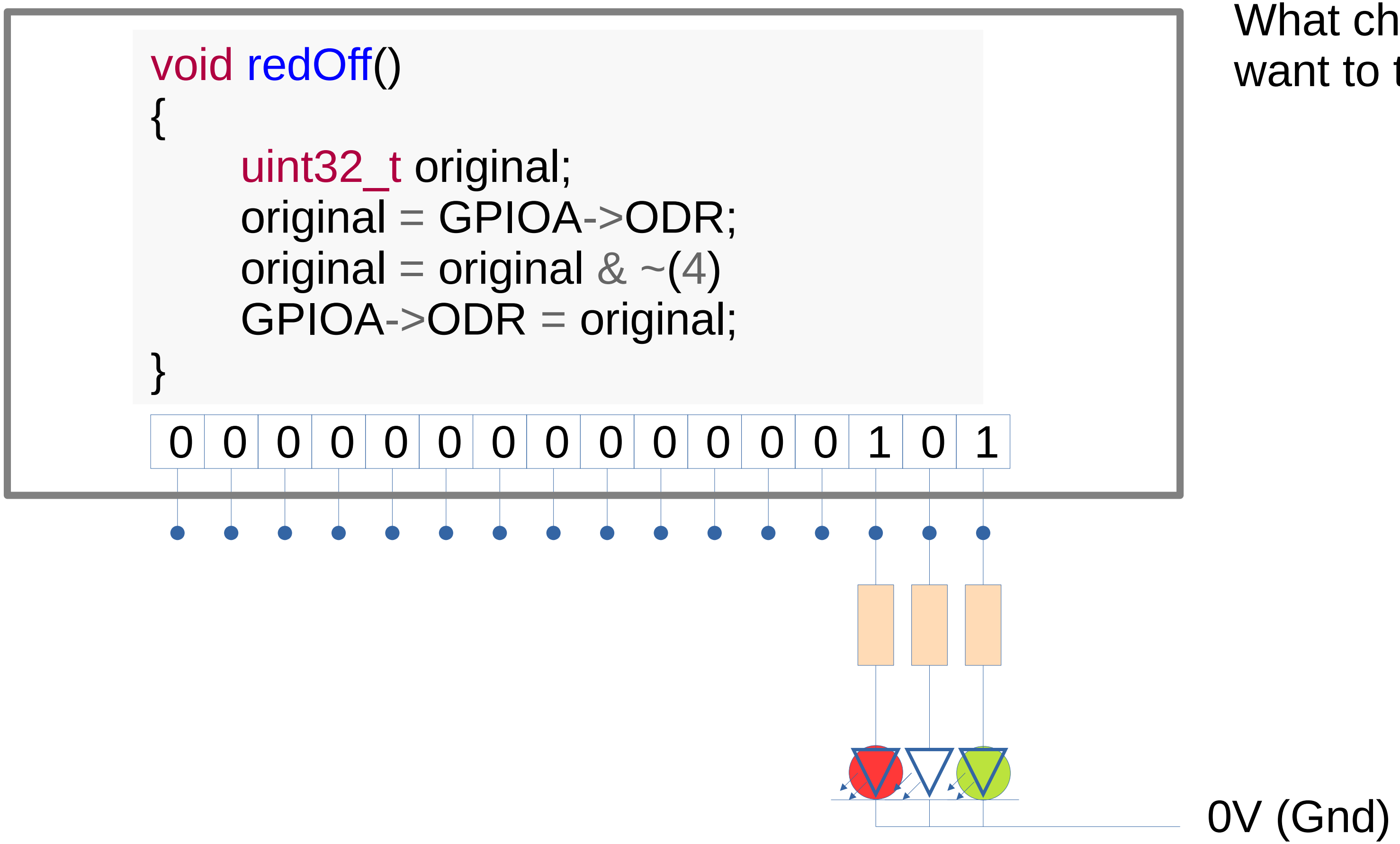


0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

What changes is we want to turn off an LED?

Controlling individual output bits

What changes is we want to turn off an LED?



```
void redOff()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original & ~(4)
        GPIOA->ODR = original;
}
```

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

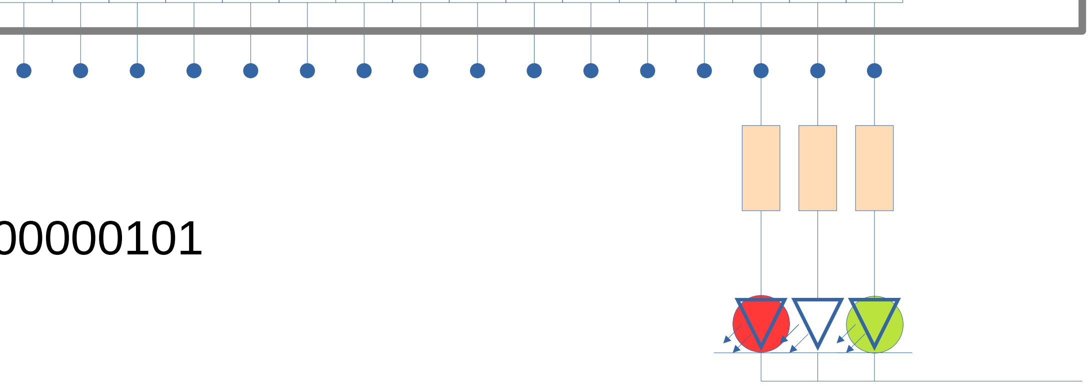
Controlling individual output bits

```
void redOff()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original & ~(4)
        GPIOA->ODR = original;
}
```

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

What changes is we want to turn off an LED?

0000000000000101

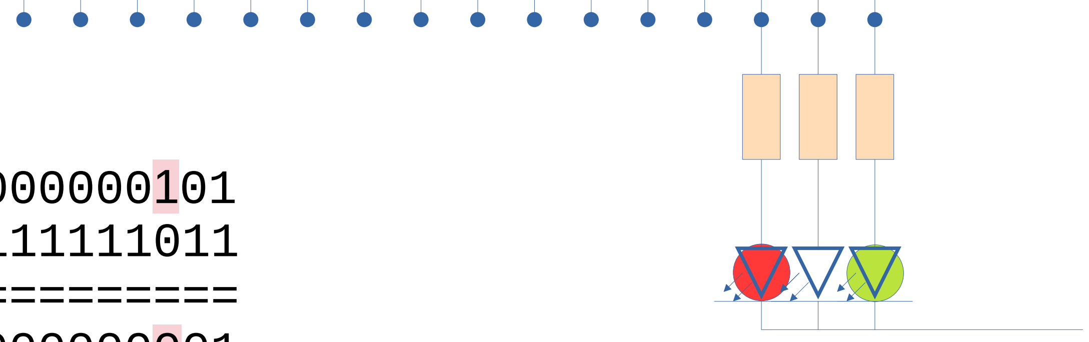


Controlling individual output bits

```
void redOff()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original & ~(4)
        GPIOA->ODR = original;
}
```

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

What changes is we want to turn off an LED?

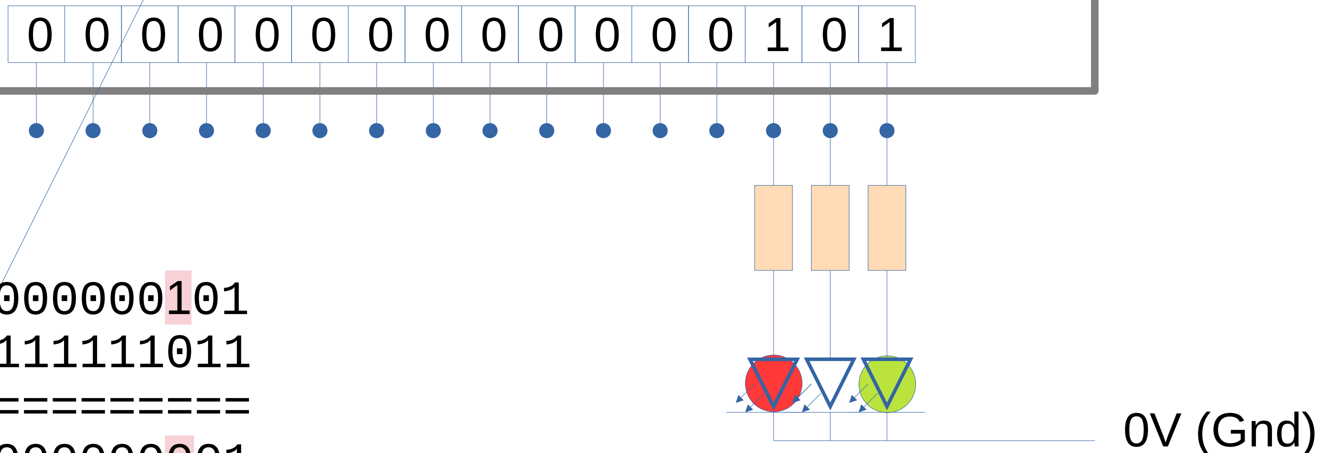


```
&  0000000000000101
   1111111111111011
   ================
   0000000000000001
```

Controlling individual output bits

```
void redOff()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original & ~(4)
        GPIOA->ODR = original;
}
```

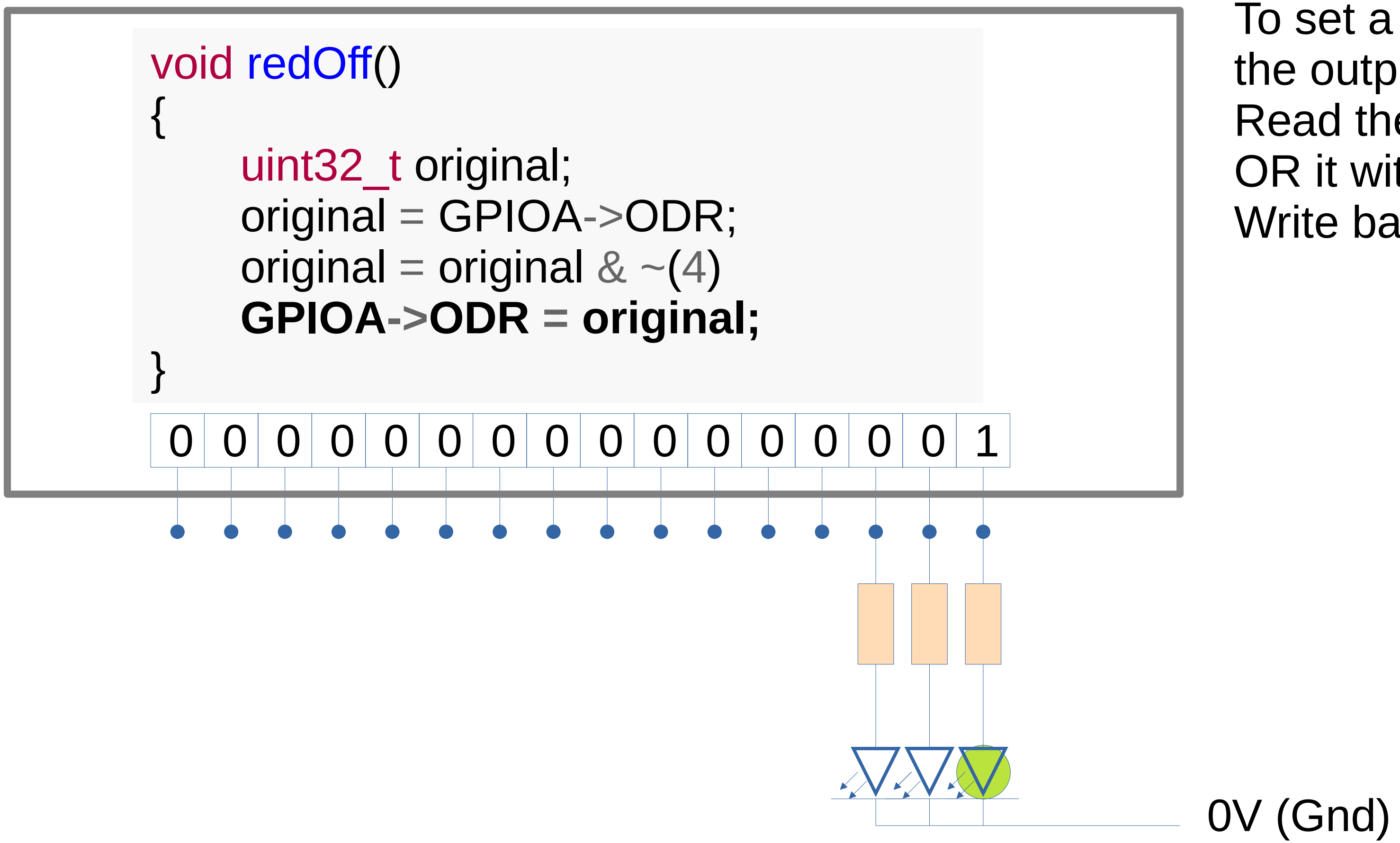
What changes is we want to turn off an LED?



0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1

```
&   0000000000000101
    1111111111111011
    ================
    0000000000000001
```

Controlling individual output bits

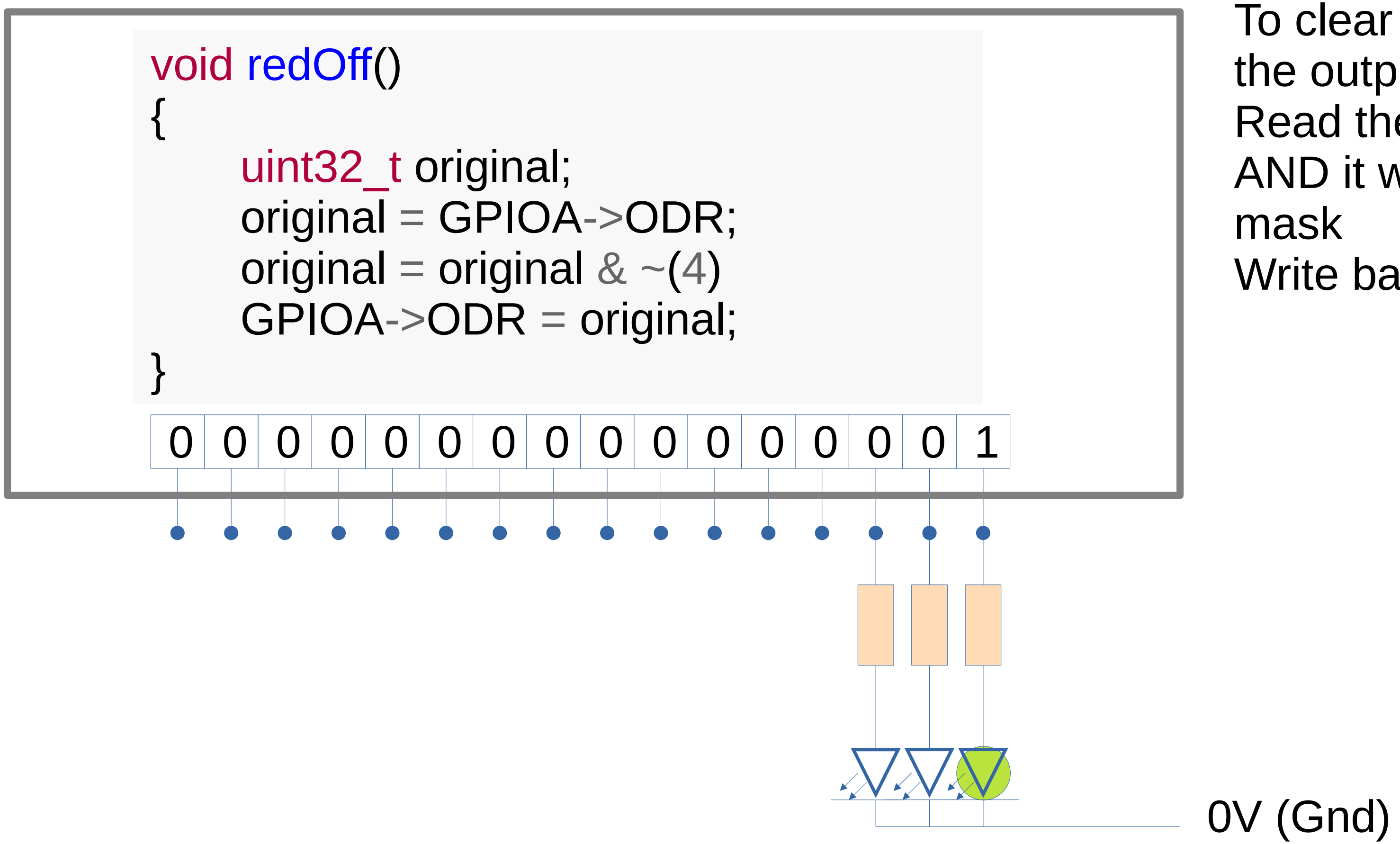


```
void redOff()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original & ~(4)
        GPIOA->ODR = original;
}
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

To set a particular bit in the output register:
Read the ODR
OR it with a mask
Write back to ODR

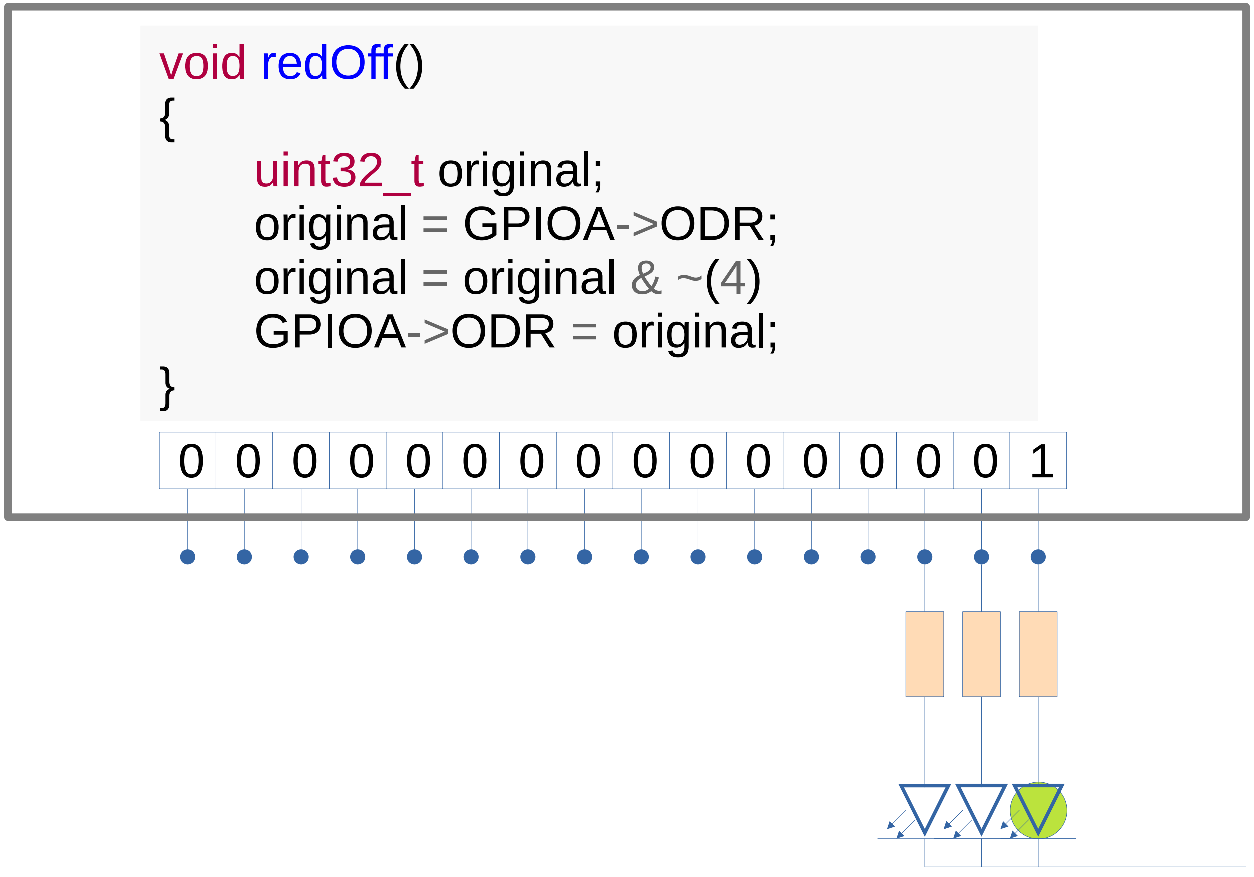
Controlling individual output bits



```
void redOff()
{
	uint32_t original;
	original = GPIOA->ODR;
	original = original & ~(4)
	GPIOA->ODR = original;
}
```

To clear a particular bit in the output register:
Read the ODR
AND it with an inverted mask
Write back to ODR

Controlling individual output bits



```
void redOff()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original & ~(4)
        GPIOA->ODR = original;
}
```

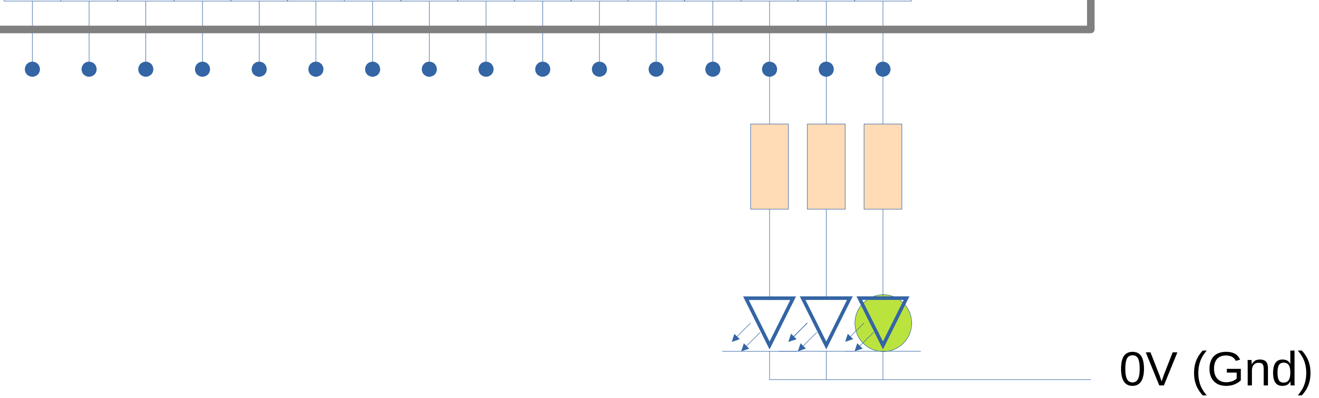
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Green LED mask : 1
Yellow LED mask : 2
Red LED mask : 4

Controlling individual output bits

```
void redOff()
{
        uint32_t original;
        original = GPIOA->ODR;
        original = original & ~(4)
        GPIOA->ODR = original;
}
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1



The mask value for a bit N is:

2^N

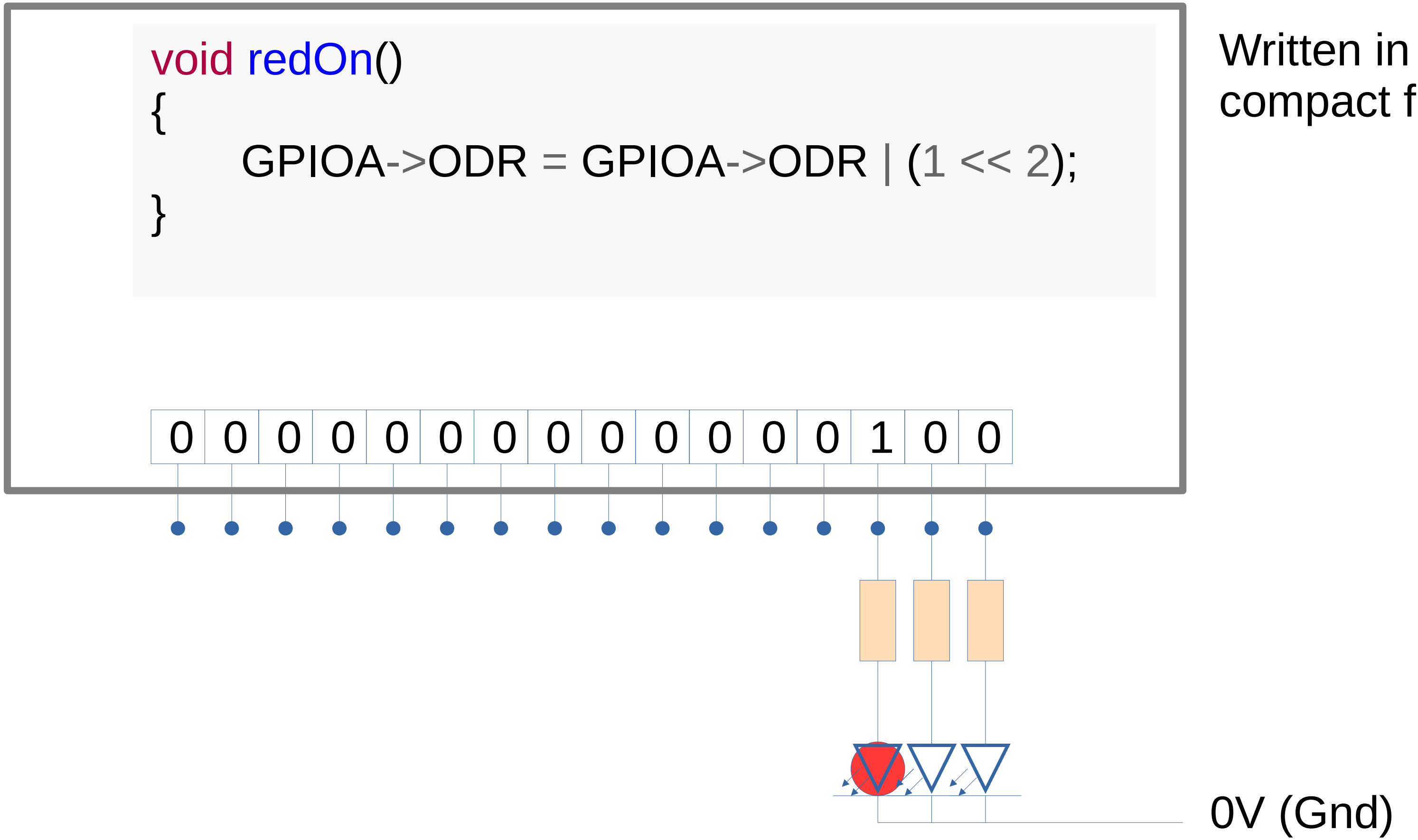
We can express this in C most efficiently as

(1 << N)

Controlling individual output bits

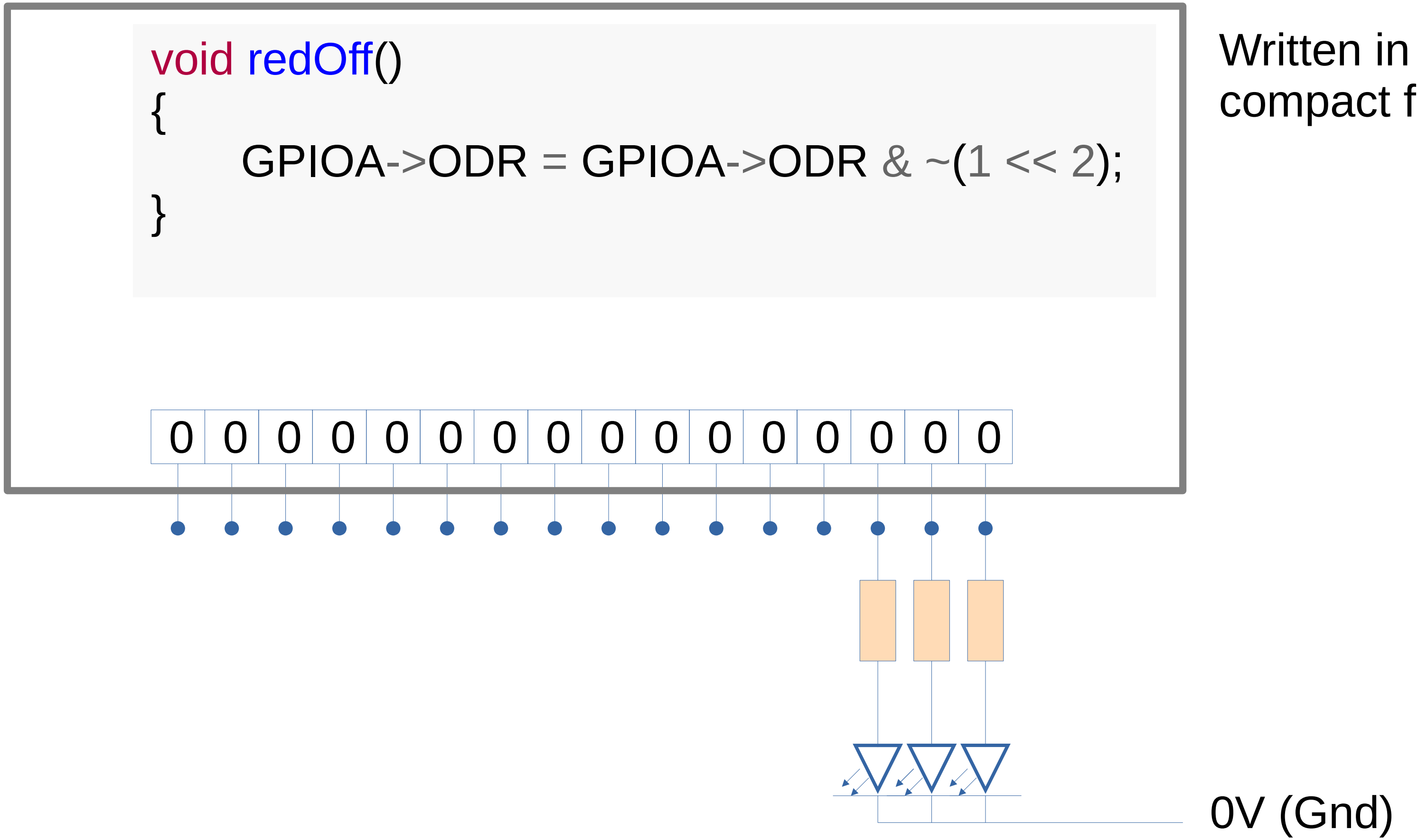
```
void redOn()
{
        GPIOA->ODR = GPIOA->ODR | (1 << 2);
}
```

Written in a more compact form



0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Controlling individual output bits



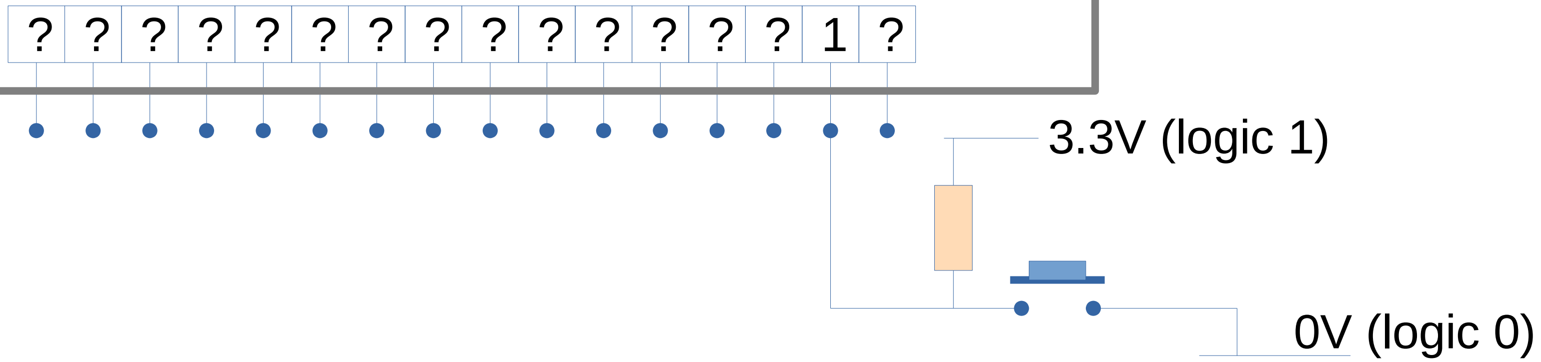
```
void redOff()
{
      GPIOA->ODR = GPIOA->ODR & ~(1 << 2);
}
```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Written in a more compact form

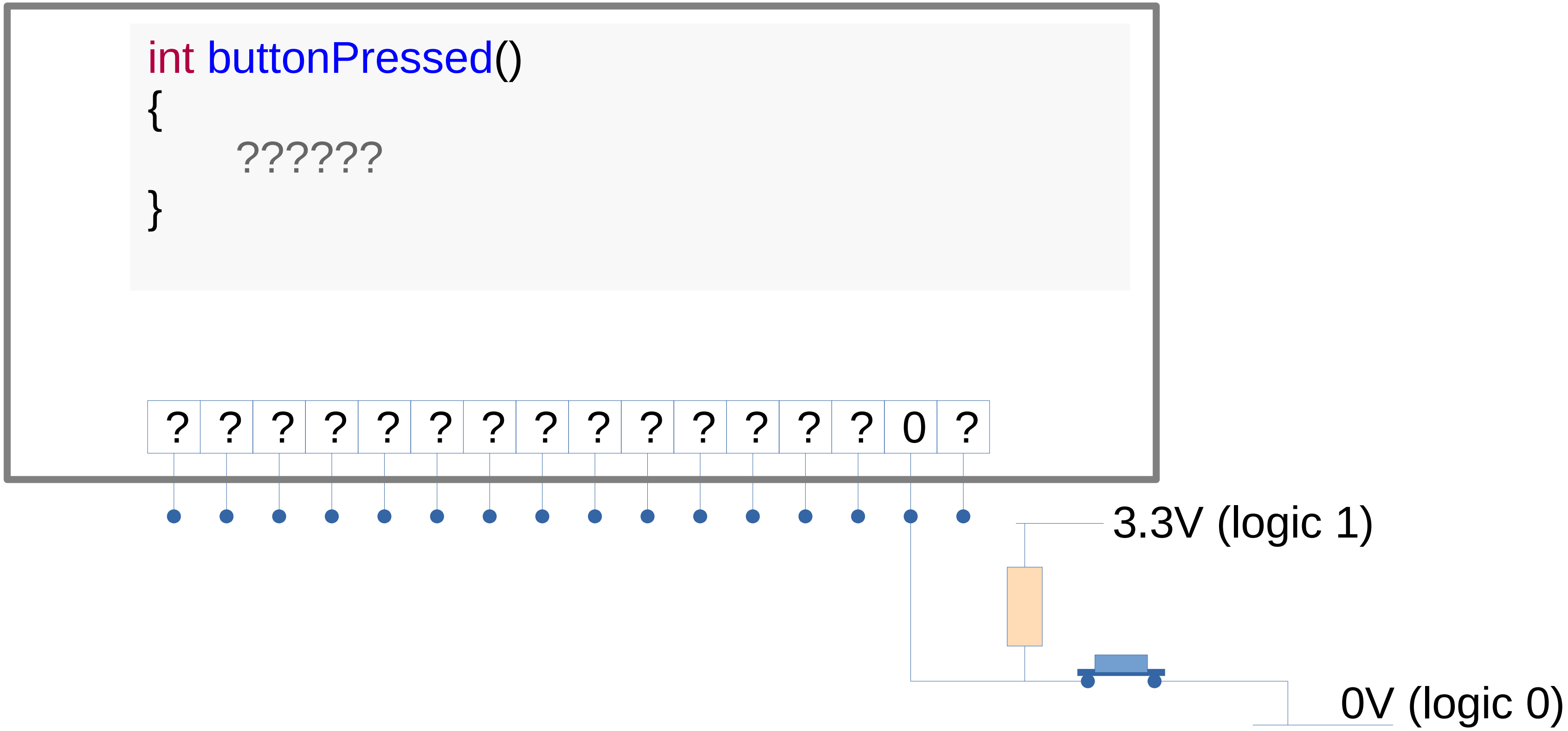
Reading individual output bits

```
int buttonPressed()
{
      ??????
}
```



?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	?

Reading individual output bits



```
int buttonPressed()
{
        ??????
}
```

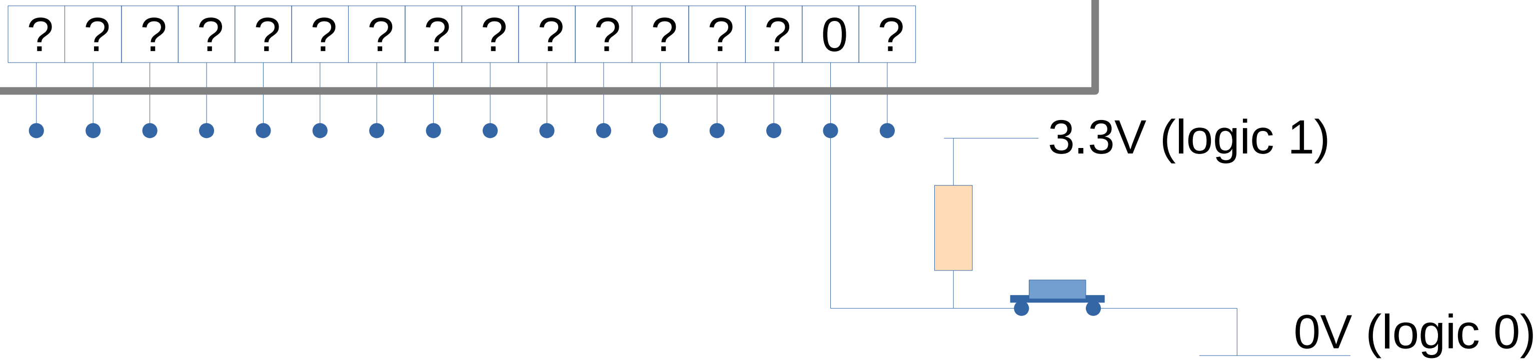
?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	?

Reading individual output bits

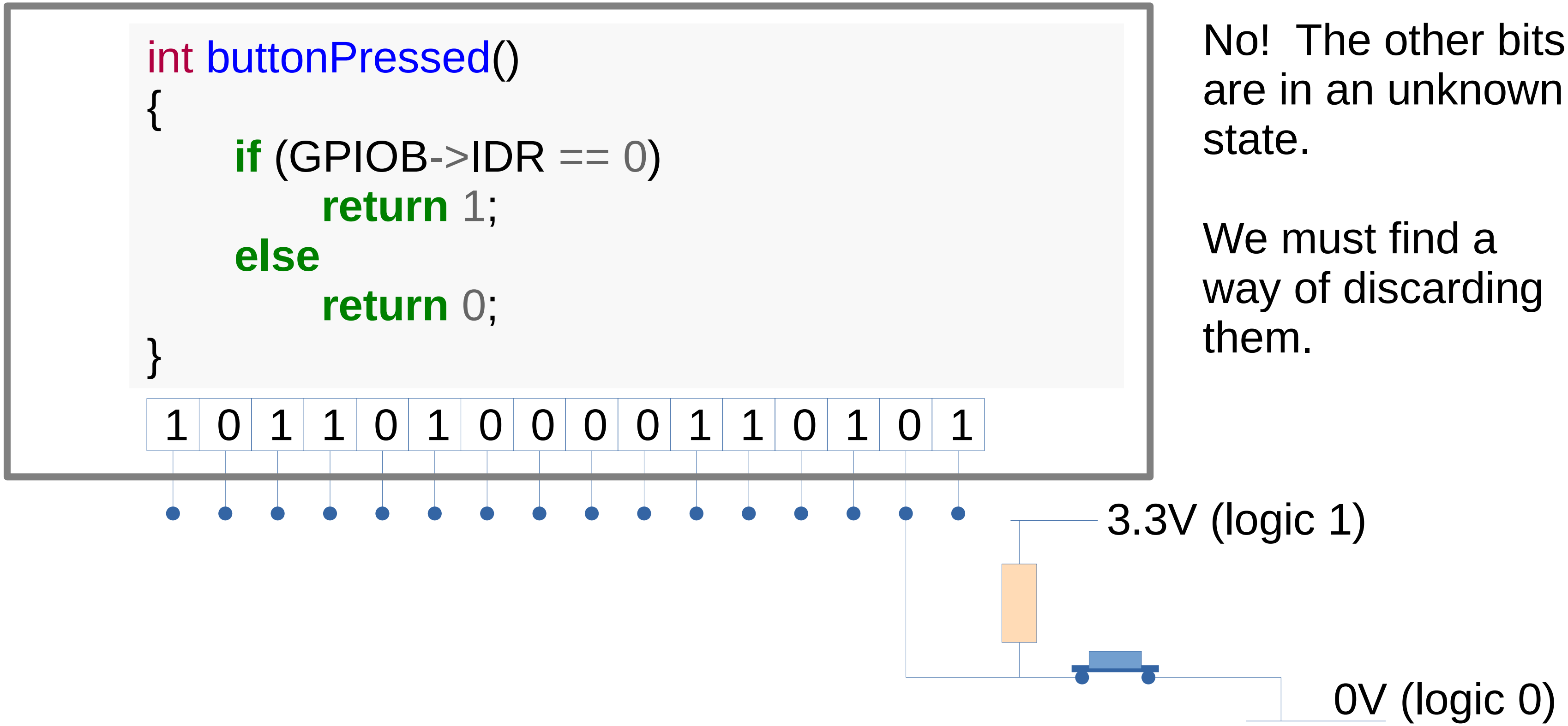
```
int buttonPressed()
{
        if (GPIOB->IDR == 0)
                return 1;
        else
                return 0;
}
```

Would this work?

?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	?



Reading individual output bits



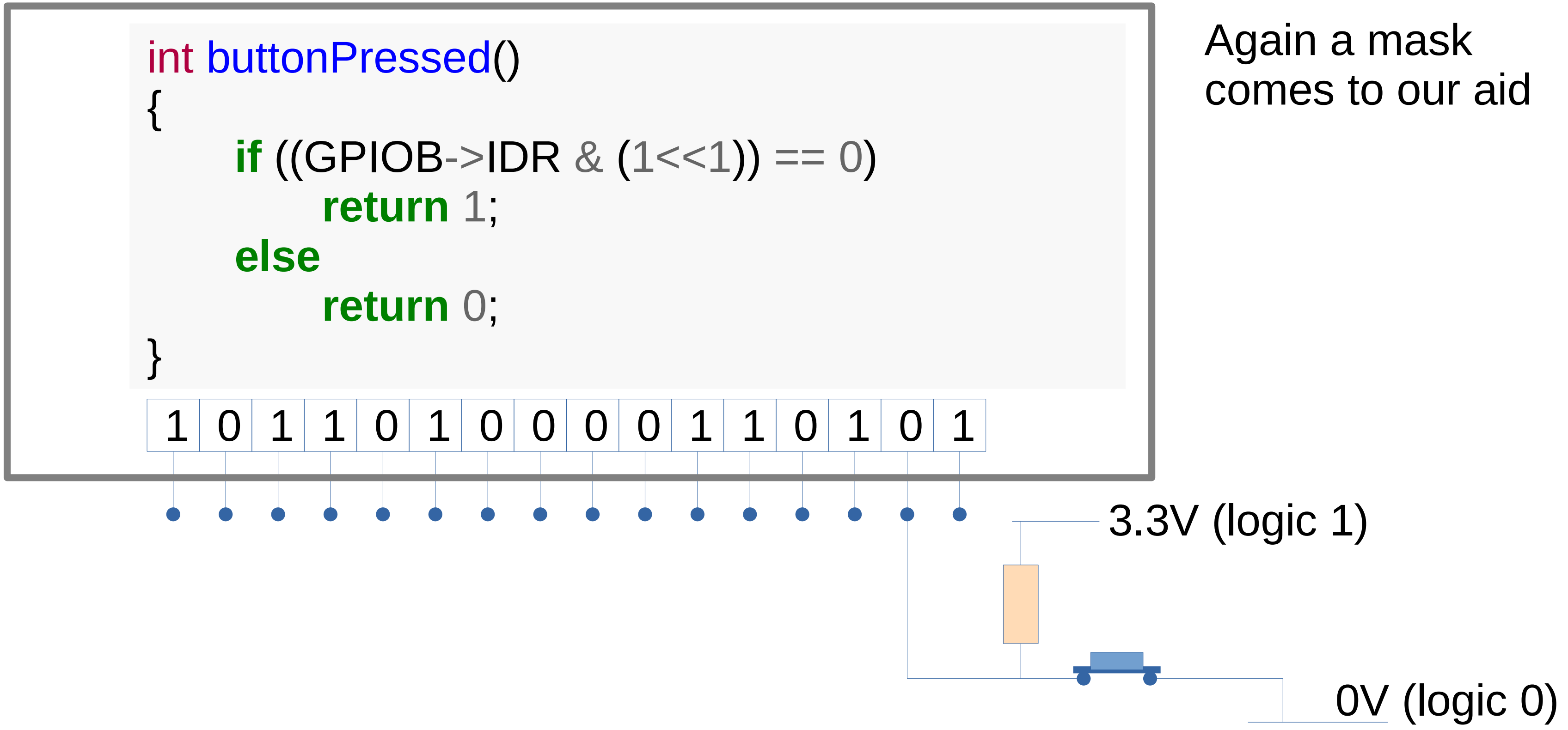
```
int buttonPressed()
{
        if (GPIOB->IDR == 0)
                return 1;
        else
                return 0;
}
```

1	0	1	1	0	1	0	0	0	0	1	1	0	1	0	1

No! The other bits are in an unknown state.

We must find a way of discarding them.

Reading individual output bits

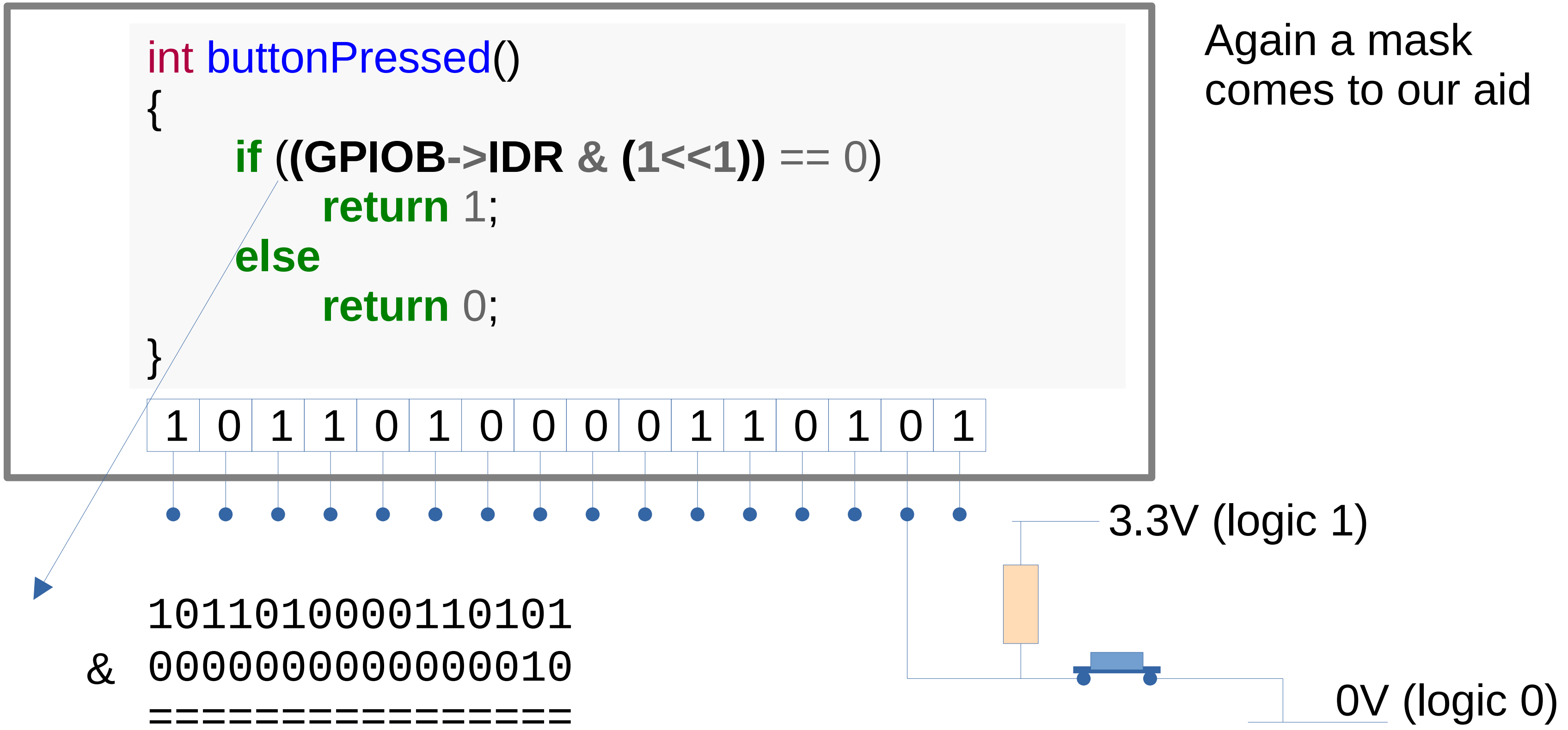


```
int buttonPressed()
{
        if ((GPIOB->IDR & (1<<1)) == 0)
                return 1;
        else
                return 0;
}
```

1 0 1 1 0 1 0 0 0 0 1 1 0 1 0 1

Again a mask comes to our aid

Reading individual output bits

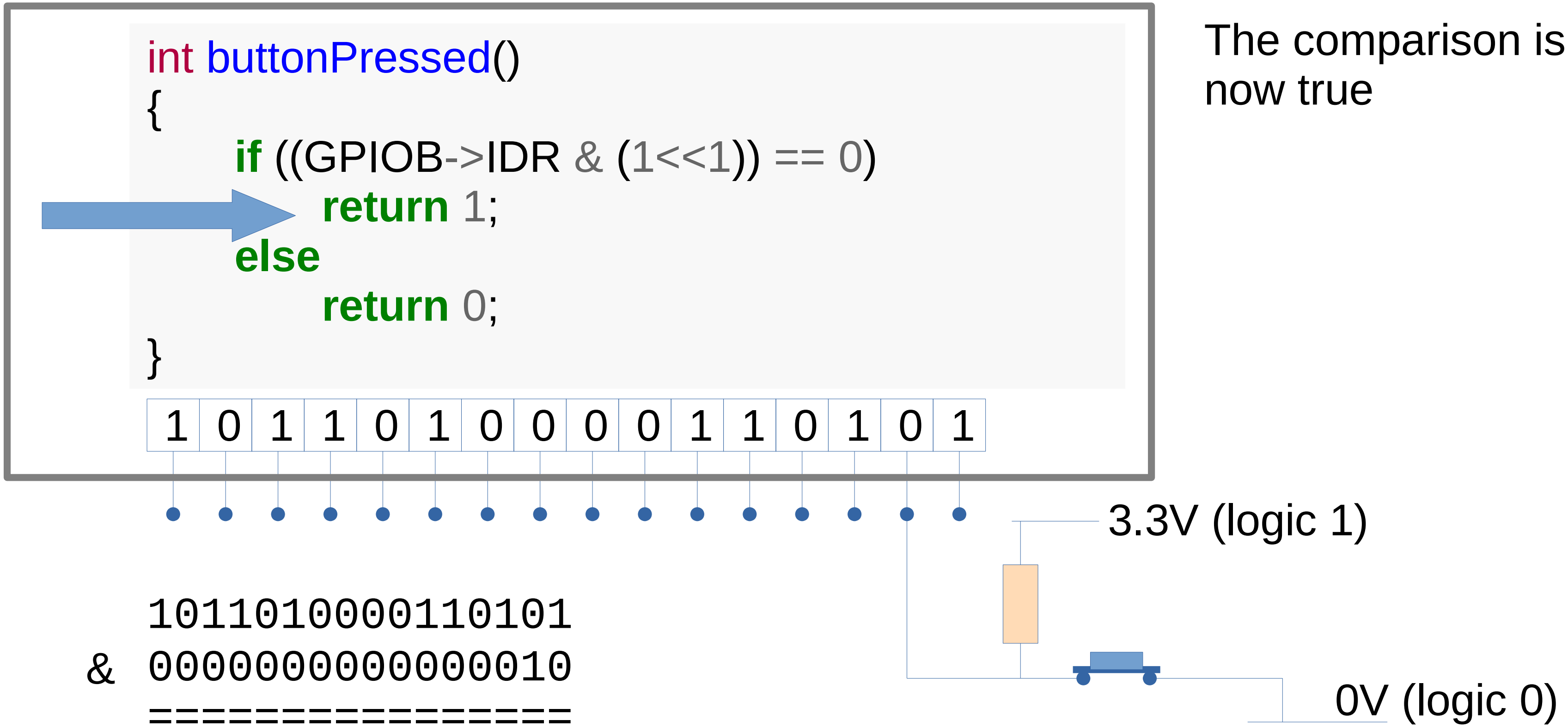


```
int buttonPressed()
{
        if ((GPIOB->IDR & (1<<1)) == 0)
                return 1;
        else
                return 0;
}
```

Again a mask comes to our aid

```
  1011010000110101
& 0000000000000010
  ================
  0000000000000000
```

Reading individual output bits

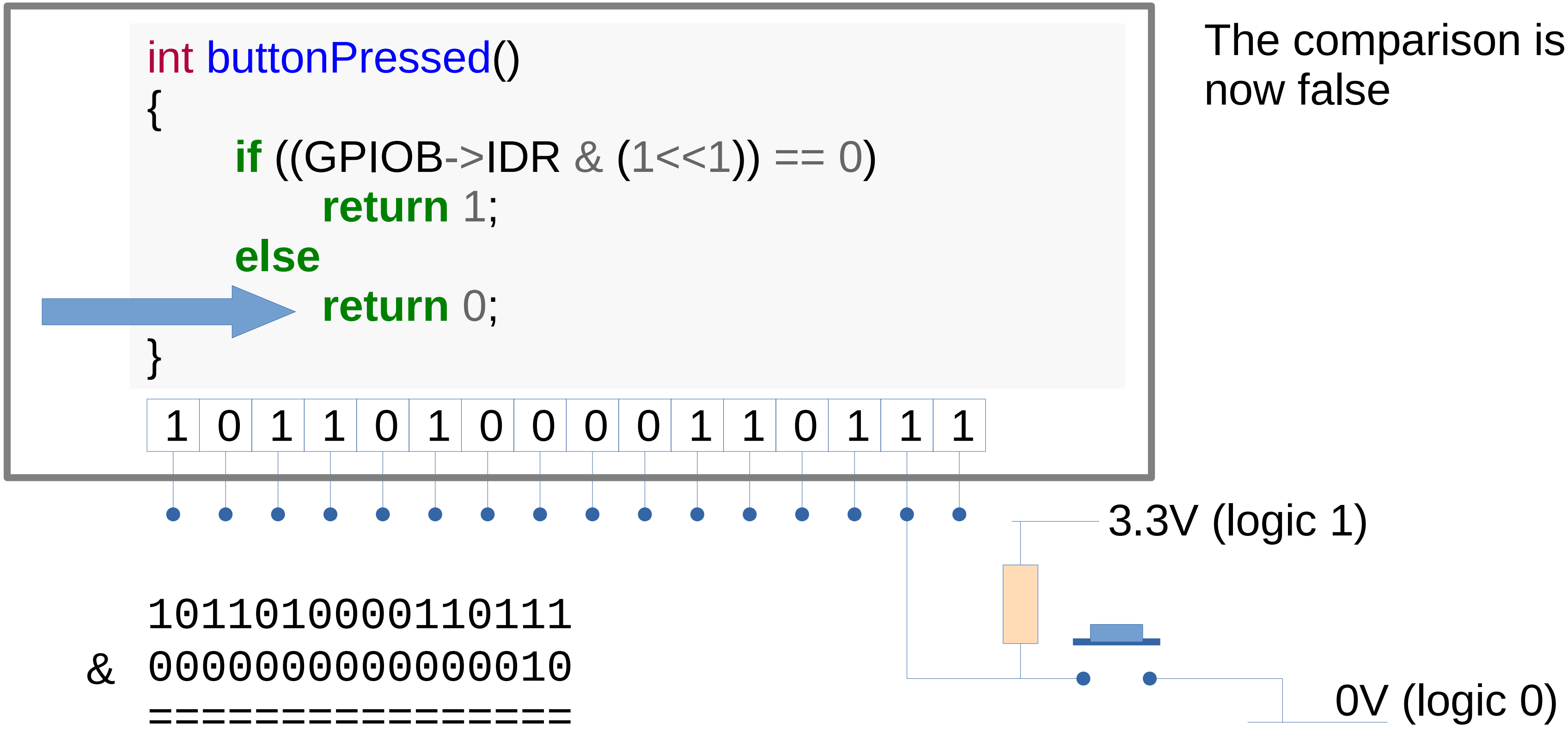


```
int buttonPressed()
{
        if ((GPIOB->IDR & (1<<1)) == 0)
                return 1;
        else
                return 0;
}
```

The comparison is now true

```
  1011010000110101
& 0000000000000010
  ================
  0000000000000000
```

Reading individual output bits



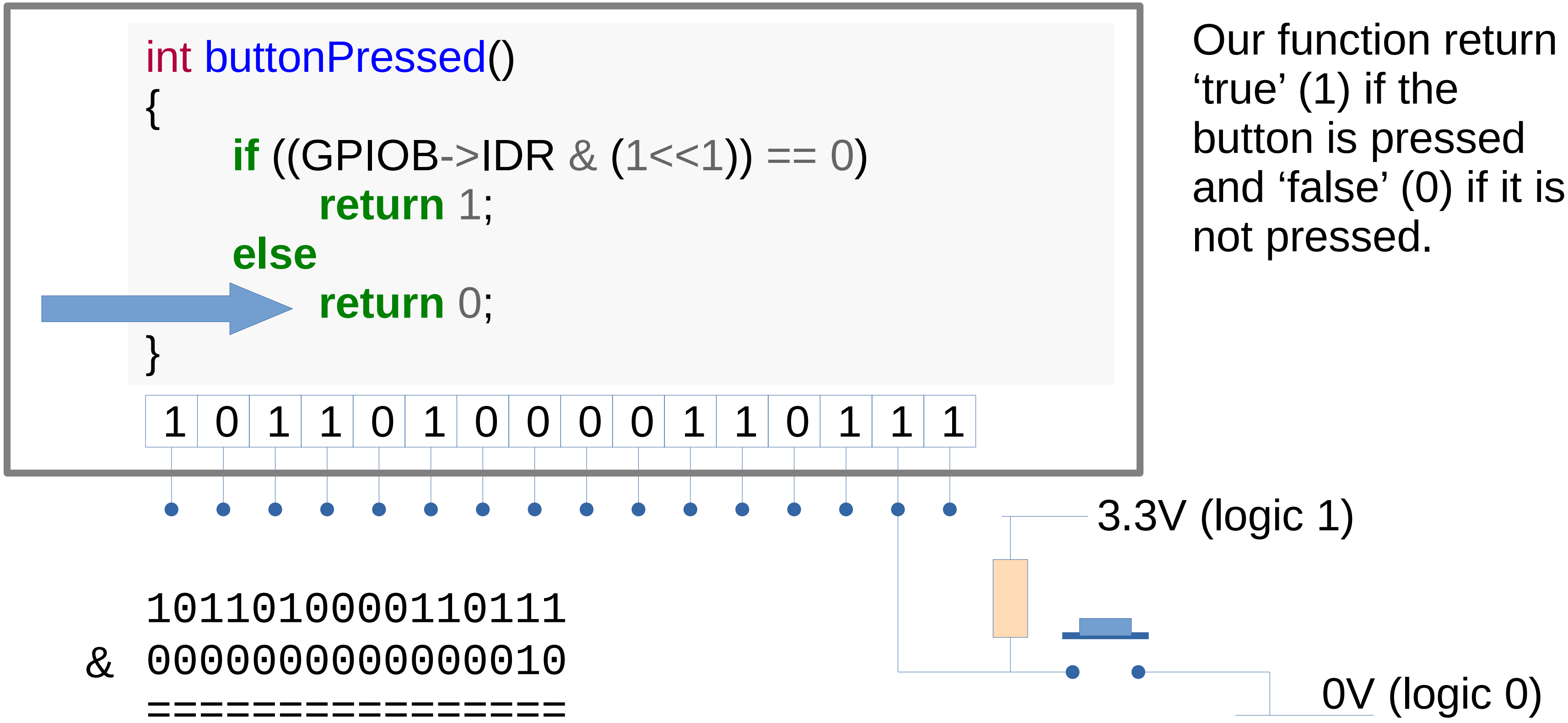
```
int buttonPressed()
{
        if ((GPIOB->IDR & (1<<1)) == 0)
                return 1;
        else
                return 0;
}
```

1	0	1	1	0	1	0	0	0	0	1	1	0	1	1	1

The comparison is now false

```
  1011010000110111
& 0000000000000010
  ================
  0000000000000010
```

Reading individual output bits



```
int buttonPressed()
{
    if ((GPIOB->IDR & (1<<1)) == 0)
        return 1;
    else
        return 0;
}
```

1 0 1 1 0 1 0 0 0 0 1 1 0 1 1 1

Our function return 'true' (1) if the button is pressed and 'false' (0) if it is not pressed.

```
  1011010000110111
& 0000000000000010
  ================
  0000000000000010
```

A program could use these helper functions as follows:

```
if ( buttonPressed() )
{
        redOn();
}
else
{
        redOff();
}
```