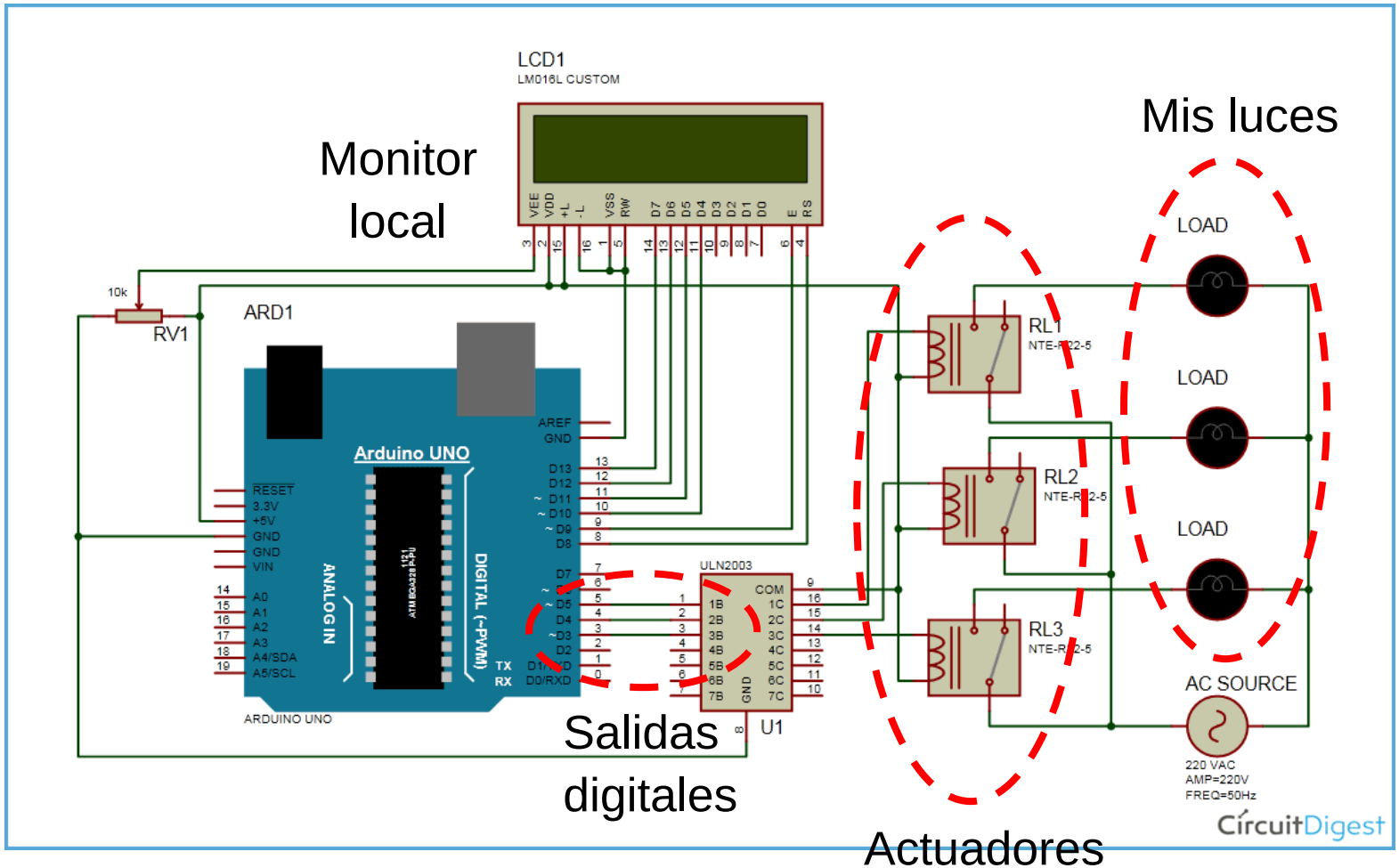
CIRCUITOS DIGITALES Y MICROCONTROLADORES 2022

Facultad de Ingeniería
UNLP

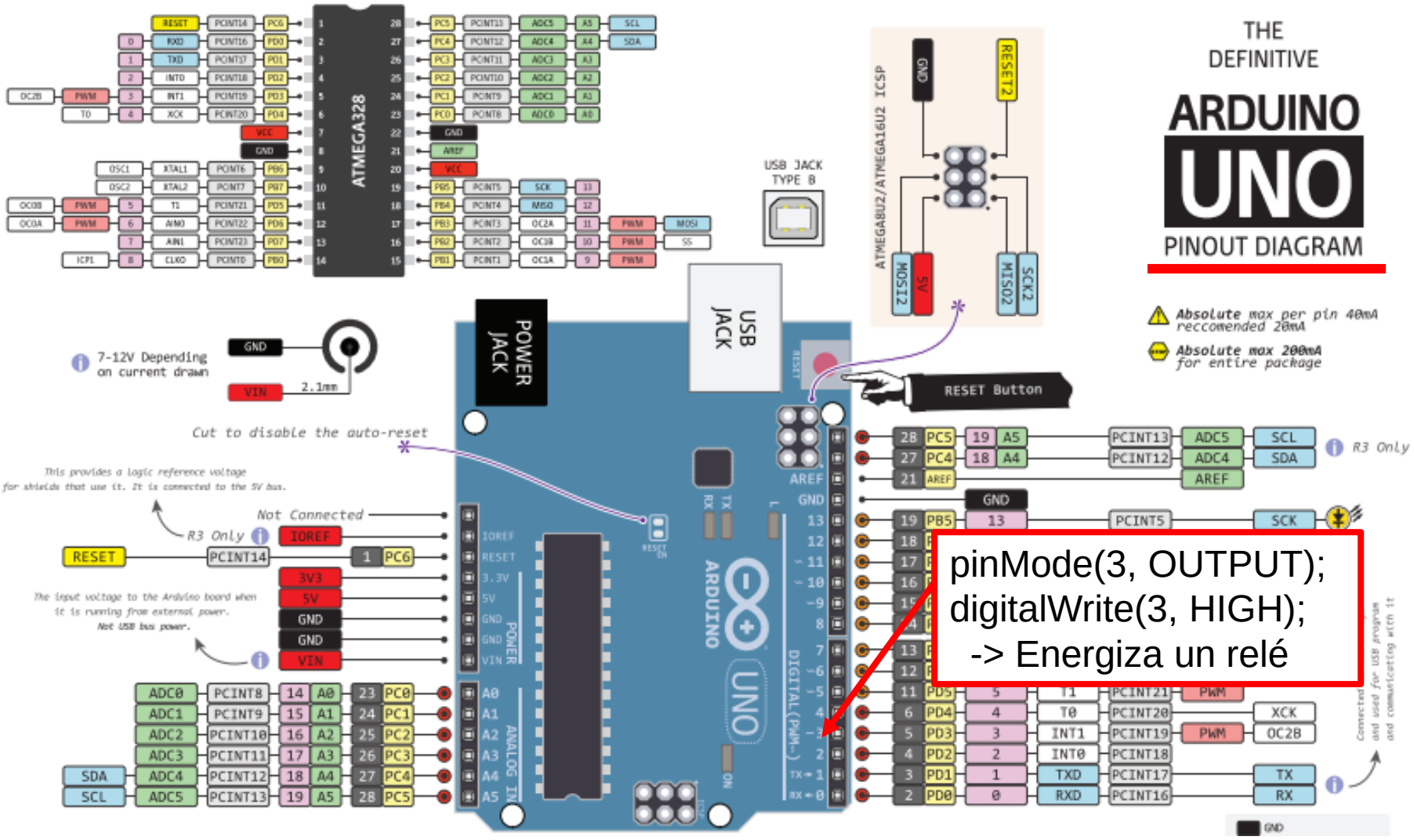
Programación de puertos de Entrada/Salida - TP1

Un ejemplo motivador
Domótica Fácil

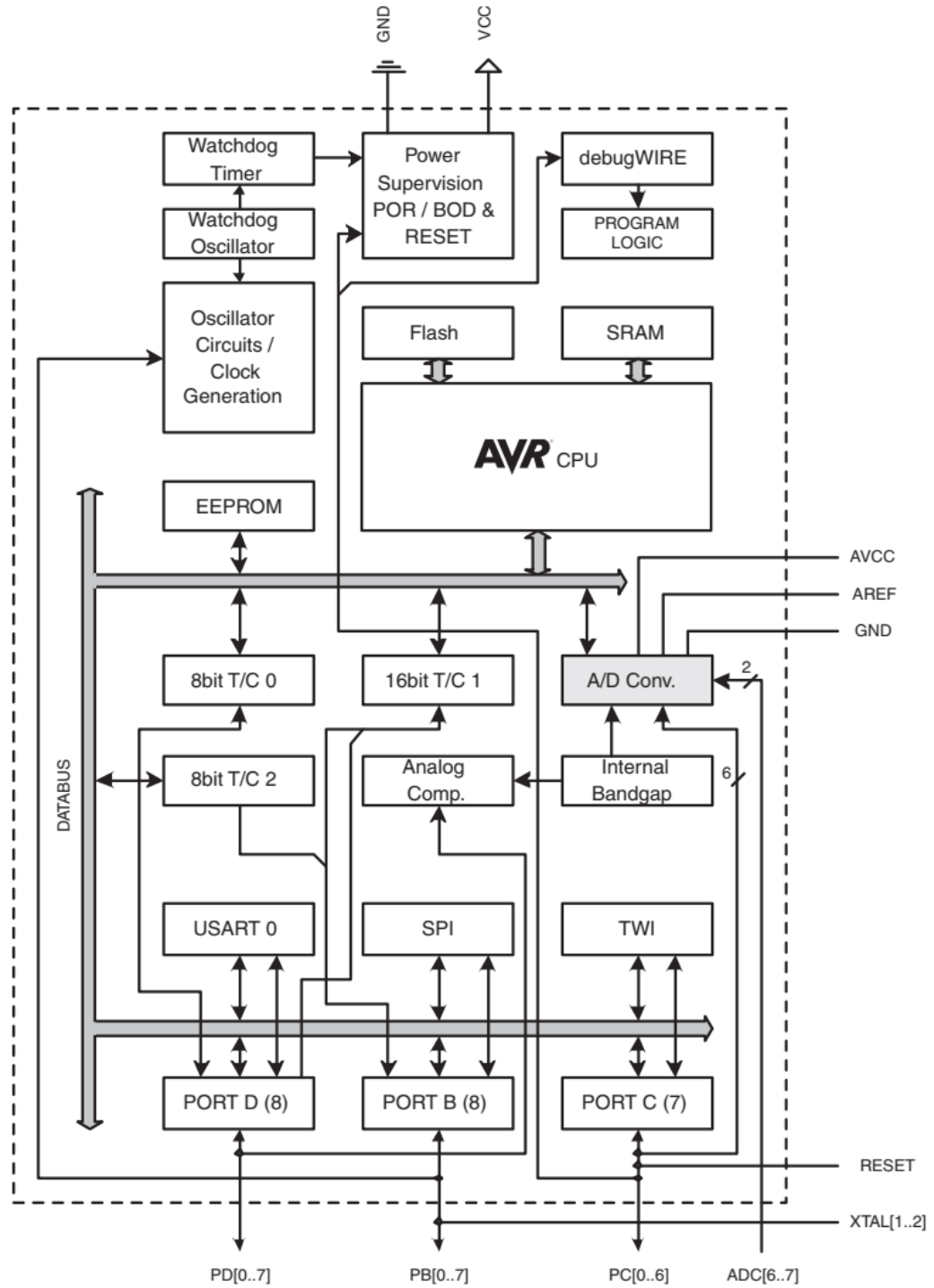


Arduino

https://www.youtube.com/watch?v=BWhup75svIk



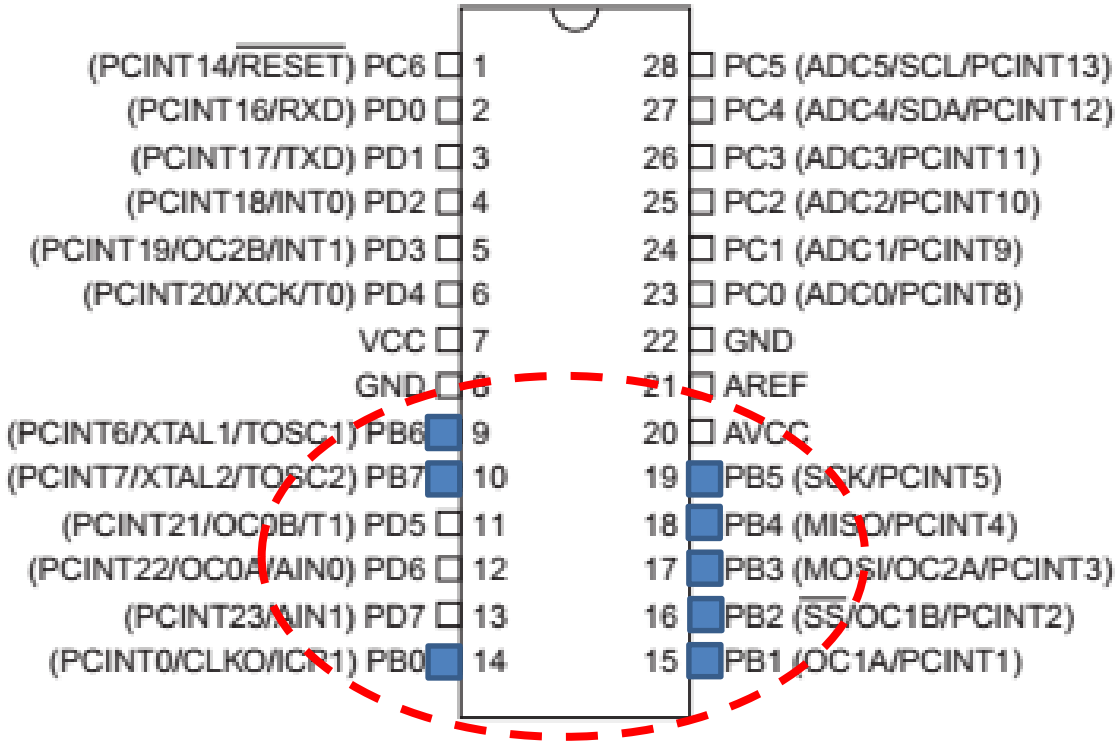
- AVR - ATMEGA 328



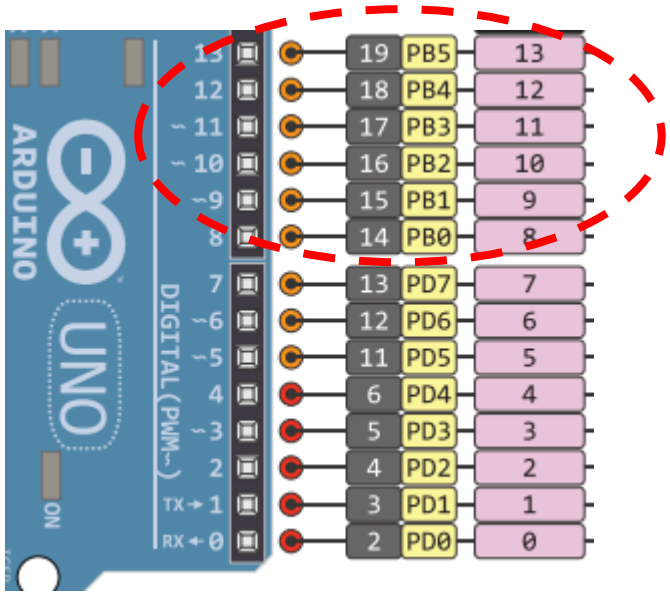
ATMEGA - AVR

¿Donde están los terminales I/O del puerto B?

Pinout ATmega48PA/88PA/168PA/328P



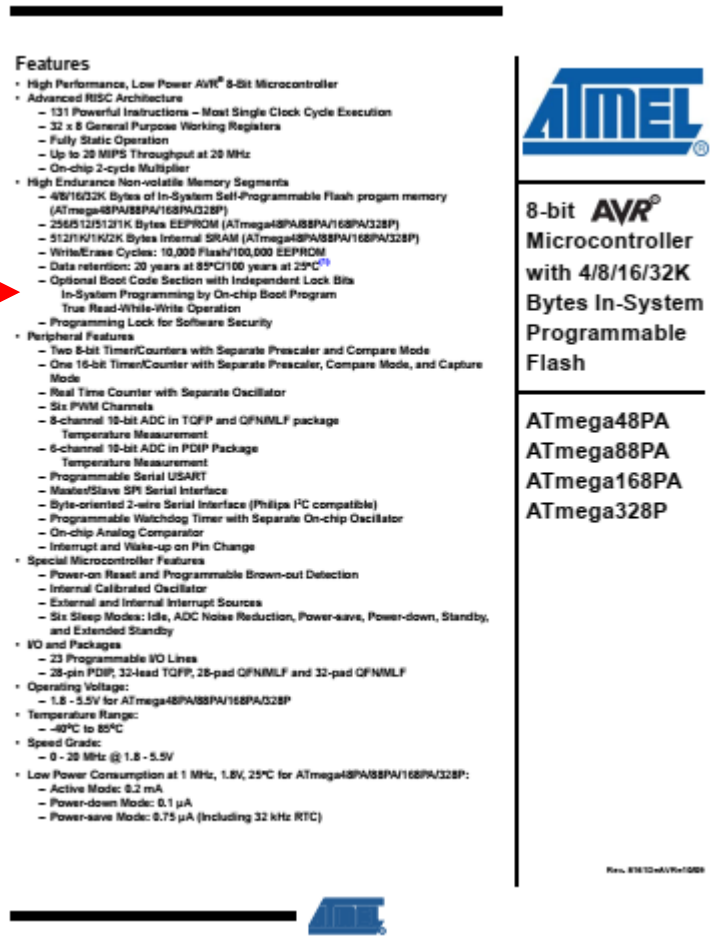
No confundir el número de pin de Arduino con el número de terminal en el chip



Registros de los Puertos I/O

¿Donde están los registros I/O del puerto B ?

Hoja de datos
(data sheet)



Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash progam memory (ATmega48PA/88PA/168PA/328P)
  - 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  - 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I²C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 20 MHz @ 1.8 - 5.5V
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

ATMEL

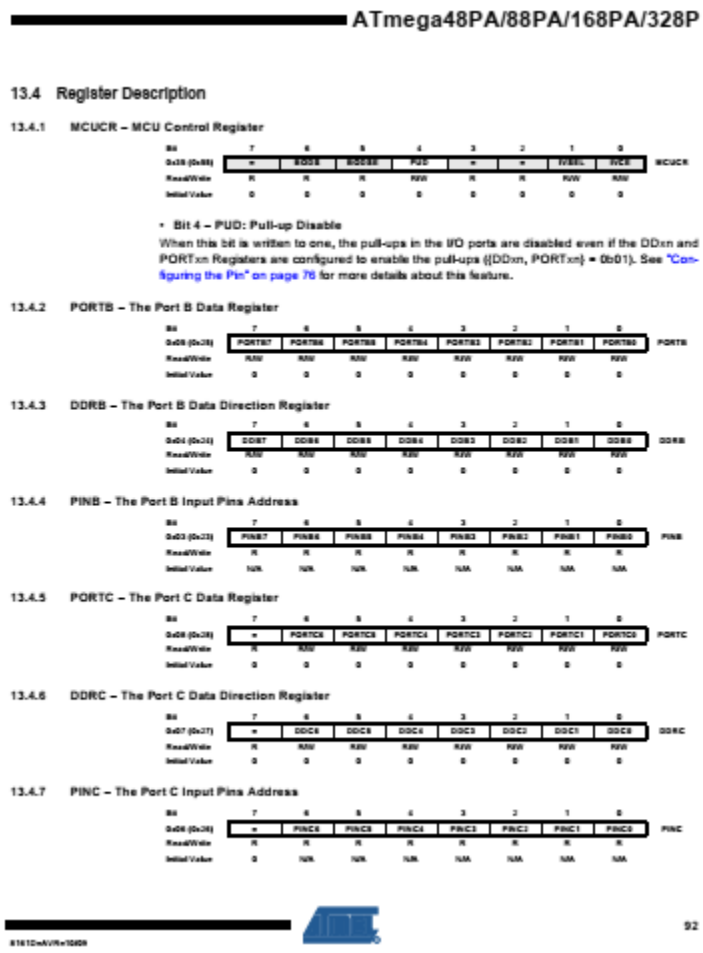
8-bit AVR® Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash

ATmega48PA
ATmega88PA
ATmega168PA
ATmega328P

Registros de los Puertos I/O

¿Donde están los registros I/O del puerto B ?

Capítulo 13
I/O ports



ATmega48PA/88PA/168PA/328P

13.4 Register Description

13.4.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	BODS	BODSE	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 4 – PUD: Pull-up Disable

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See "Configuring the Pin" on page 76 for more details about this feature.

13.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

13.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

13.4.4 PINB – The Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

13.4.5 PORTC – The Port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

13.4.6 DDRC – The Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

13.4.7 PINC – The Port C Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

92

Registros de los Puertos I/O

¿Donde están los registros I/O del puerto B ?

PB0

descripción →

PORTB – Port B Data Register Address I/O: 0x05

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRB – Port B Data Direction Register Address I/O: 0x04

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

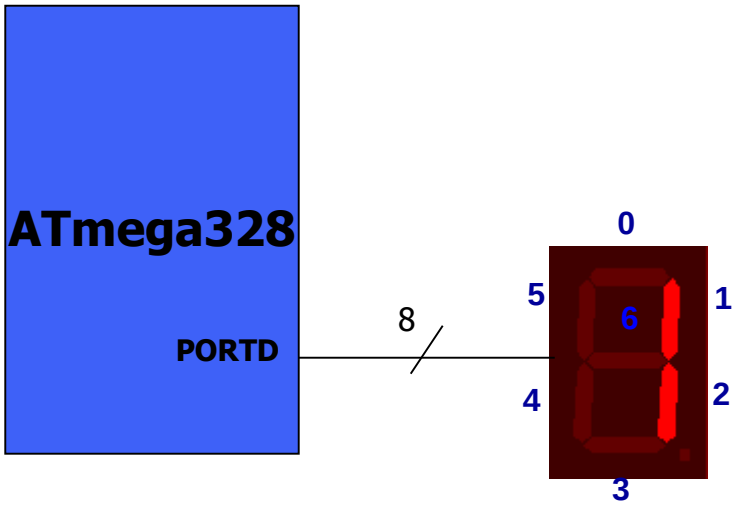
PINB – Port B Input Pins Address Address I/O: 0x03

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Registros de los Puertos I/O

Mostrar un “1” en un display 7-segmentos

DDRD:	1	1	1	1	1	1	1	1
PORTD:	0	0	0	0	0	1	1	0

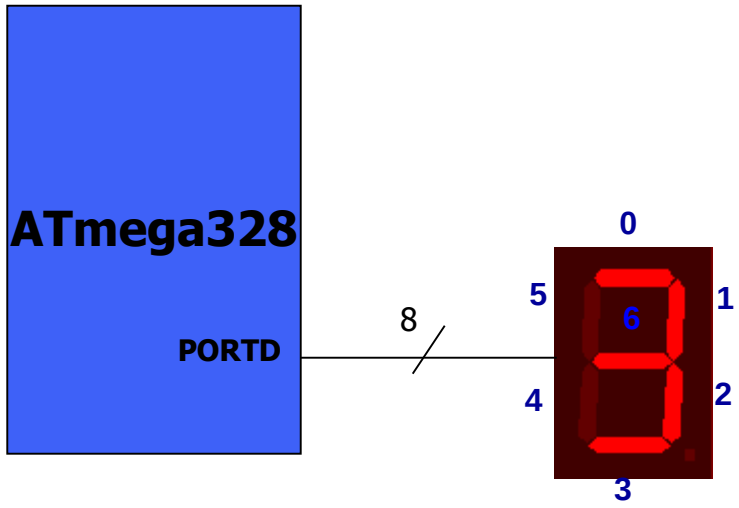


PORTx / DDRx	0	1
0	high impedance	Out 0
1	pull-up	Out 1

Registros de los Puertos I/O

Mostrar un “3” en un display 7-segmentos.

DDRD:	1	1	1	1	1	1	1	1
PORTD:	0	1	0	0	1	1	1	1

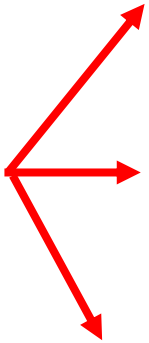


PORTx \ DDRx	0	1
0	high impedance	Out 0
1	pull-up	Out 1

Programando PORTS en C

¿Cómo se programan en C?

variables uint8 que llevan los mismos nombres que los registros



DDRB= 0b01010110;

PORTB= 0xC3;

mi_var =PINB;

La asociación entre las variables y los registros se realiza en el archivo de cabecera `io.h`

Programando PORTS en C

Ejemplo: enviar un valor 0xAA al PORTD

```
#include <avr/io.h>

int main ()
{
   DDRD = 0xFF;          DDRD=11111111
   PORTD = 0xAA;         PORTD=10101010

   while (1);            Loop
   return 0;
}
```

Programando PORTS en C

Ejemplo: Leer, modificar y escribir un PORT

```
#include <avr/io.h>
int main(void)
{
   DDRB = 0xFF;
   while (1)
   {
      PORTB = PORTB + 1;
   }
   return 0;
}
```

Incluye cabecera andard AVR header

Función principal

Inicialización '/Port B is output

Bucle infinito

Operaciones: Leer-sumar-escribir

Fin función principal
(nunca se alcanza)

Programando PORTS en C

Ejemplo: transferir entradas a salidas

```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
   unsigned char temp;

   DDRB = 0x00;                        //Port B is input
   DDRC = 0xFF;                        //Port C is output

   while(1)
   {
      temp = PINB;
      PORTC = temp;
   }
   return 0;
}
```

Programando PORTS en C

Ejemplo: calcular PINB + PINC y escribir el resultado en PORTD.

```
#include <avr/io.h>

int main ()
{
   DDRB = 0x00;          Todos los bits del PORTB como entradas
   DDRC = 0x00;          Todos los bits del PORTC como entradas
   DDRD = 0xFF;          Todos los bits del PORTD como salidas

   while (1)
     PORTD = PINB + PINC;
   return 0;
}
```

Programando PORTS en C

Ejemplo: visualizar un contador de 8 bits en un PORT

```
#include <avr/io.h>                    //standard AVR header

int main(void)
{
   unsigned char z;
   DDRB = 0xFF;                        //PORTB is output
   for(z = 0; z <= 255; z++)
     PORTB = z;
   while(1);
   return 0;
}
```

¿qué valores toman los bits del PORTB?

¿cuanto tiempo tarda en ejecutarse?

Resultado:

PORTB = 0x00, 0x01, 0x02, … ,0xFF

Programando PORTS en C

Ejemplo: escribir caracteres ASCII en el PORTB

```
#include <avr/io.h>                    //standard AVR header

int main(void)                         //the code starts from here
{
      unsigned char myList[]= "012345ABCD";
      unsigned char z;
      DDRB = 0xFF;                     //PORTB is output
      for(z=0; z<10; z++)              //repeat 10 times and increment z
            PORTB = myList[z];         //send the character to PORTB

      while(1);                        //needed if running on a trainer
      return 0;
}
```

¿qué codifican los bits en el PORTB?

´0´=48=0b00110000

Programando PORTS en C

Ejemplo: Trabajando con números con signo (ca2)

```
#include <avr/io.h>                    //standard AVR header

int main(void)
{
   char mynum[ ]  = { -4,-3,-2,-1,0,+1,+2,+3,+4} ;
   unsigned char z;

   DDRB = 0xFF;                        //PORTB is output

   for(z=0; z<=8; z++)
     PORTB = mynum[ z] ;

   while(1);                           //stay here forever
   return 0;
}
```

Resultado:

PORTB = 0xFC, 0xFD, 0xFE, 0xFF, 0x00, 0x01, 0x02, 0x03, 0x04

Operaciones aritméticas

Ejemplo: Conversión Binario-Decimal

```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
     unsigned char x, binbyte, d1, d2, d3;
     DDRB = DDRC = DDRD =0xFF;         //Ports B, C, and D output
     binbyte = 0xFD;                   //binary (hex) byte
     x = binbyte / 10;                 //divide by 10
     d1 = binbyte % 10;                //find remainder (LSD)
     d2 = x % 10;                      //middle digit
     d3 = x / 10;                      //most-significant digit (MSD)
     PORTB = d1;
     PORTC = d2;
     PORTD = d3;
     while(1);
     return 0;
}
```

dato

Unidad

decena

centena

0xFD =253

Centena = 2 = 0b00000010

Decena = 5 = 0b00000101

Unidad = 3 = 0b00000011

Operadores lógicos bit a bit

Table 7-3: Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	Y=~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

AND

```
  1110 1111
& 0000 0001
-------------
  0000 0001
```

OR

```
  1110 1111
| 0000 0001
-------------
  1110 1111
```

NOT

```
~ 1110 1011
-------------
  0001 0100
```

Operadores de desplazamiento

- data >> number of bits to be shifted right
- data << number of bits to be shifted left

```
 1110 0000 >> 3
--------------
 0001 1100
```

```
 0000 0001 <<2
--------------
 0000 0100
```

Operaciones Lógicas con bits

Ejemplo: Trabajando con operadores lógicos a nivel de bits

```
#include <avr/io.h>                 //standard AVR header

int main(void)
{
   DDRB = 0xFF;                     //PORTB is output
   DDRC = 0x00;                     //PORTC is input
   DDRD = 0xFF;                     //PORTB is output

   while(1)
   {
      if (PINC & 0b00100000)        //check bit 5 (6th bit) of PINC
        PORTB = 0x55;
      else
        PORTB = 0xAA;
   }

   return 0;
}
```

PINC XXXX XXXX

& 0010 0000

00x0 0000

¿resultados posibles?

Operaciones Lógicas con bits

Ejemplo: Trabajando con operadores lógicos a nivel de bits

```
#include <avr/io.h>                          //standard AVR header

int main(void)
{
   DDRB = 0xFF;                              //PORTB is output

   while(1)
   {
      PORTB = PORTB | 0b00010000;            //set bit 4 (5th bit) of PORTB
      PORTB = PORTB & 0b11101111;            //clear bit 4 (5th bit) of PORTB
   }

   return 0;
}
```

Simplificaciones Lógicas con I/O

Establecer un 1 lógico en el bit 4 del PORTB

```
    xxxx xxxx
|   0001 0000
    -----------
    xxx1 xxxx
```

OR

```
    xxxx xxxx
|   1 << 4
    -----------
    xxx1 xxxx
```

```
PORTB |= (1<<4); //set bit 4 (5th bit) of PORTB
```

Notar que de esta manera no se modifican los otros bits del mismo registro

Simplificaciones Lógicas con I/O

Establecer un 0 lógico en el bit 4 del PORTB

```
      xxxx  xxxx
&     1110 1111
      ----------
      xxx0  xxxx
```

OR

```
      xxxx xxxx
&     ~(1 << 4)
      ----------
      xxx0 xxxx
```

```
PORTB &= ~(1<<4); //clear bit 4 (5th bit) of PORTB
```

Simplificaciones Lógicas con I/O

¿está el bit 5 de PINC en 1?

```
        xxxx xxxx
&       0010 0000  == (1 << 5)
        -----------
        00x0 0000
```

x puede ser 0 ó 1

```
if( ((PINC & (1<<5)) != 0)  //check bit 5 (6th bit)
```

o directamente:

```
if(((PINC & (1<<5)))  //check bit 5 (6th bit)
```

Operaciones sobre registros I/O

- **Resumiendo: operaciones con "<< "**

```
X |= (1 << Bitnumber);       // setear un bit de la variable x, SIN AFECTAR EL RESTO
X &= ~(1 << Bitnumber) ;     // borrar un bit de la variable x, SIN AFECTAR EL RESTO
```

- **Ejemplo:**

```
#include <avr/io.h>
...
PORTA |= (1 << 2);           //setear bit 2 del port A
PORTA &= ~(1 << 2);          //borrar bit 2 del port A
```

- **Utilizando los macros definidos en <avr/io.h>**

```
#include <avr/io.h>
...
DDRA &= ~( (1<<PORTA0) | (1<<PORTA3) );        /* Setear PA0 y PA3 como inputs */
PORTA |= (1<<PORTA0) | (1<<PORTA3);            /* Habilitar sus Pull UP s internos */
```

Operaciones sobre registros I/O

- **Implementación y Optimización del compilador:**

```
PORTA |= (1<<PORTA0);                                  //set bit 0 del port A
PORTA |= (1<<PORTA2) | (1<<PORTA3) | (1<<PORTA4);      //set bits 2, 3 y 4 del port A
```

- **En Assembler queda:**

```
   PORTA |= (1<<PORTA0);
6c:  d8 9a       sbi         0x1b, 0

   PORTA |= (1<<PORTA2) | (1<<PORTA3) | (1<<PORTA4);
6e:  8b b3       in          r24, PORTA
70:  8c 61       ori         r24, 0x1C
72:  8b bb       out         PORTA, r24
```

Utilizando AVR LIBC

Evaluar un bit de registro

- Puede usarse la expresión : **PINC & (1<<PINC1)**
- O puede realizarse utilizando las “avr libc” de la siguiente manera:
  - **bit_is_set (PINC, PINC1)**
  - **Devuelve 1 si el el bit 1 de PINC está en alto**
- De la misma manera: **!(PINB & (1<<PINB2))**
- Puede realizarse utilizando las “avr libc” así:
  - **bit_is_clear (PINB, PINB2)**
  - **Devuelve 1 si el el bit 2 de PINB está en bajo**

Utilizando AVR LIBC

Esperar el resultado de un flag (sondeo o polling):

```
while( !(ADCSRA&(1<<ADIF)))
        { ; }                          //esperar fin de conversión
```

Puede realizarse con las libc de la siguiente manera:

```
loop_until_bit_is_set (ADCSRA, ADIF);
```

O también su contraparte:

```
loop_until_bit_is_clear (ADCSRA, ADIF);
```

VER todas las funciones en: avr-libc-user-manual-2.0.0.pdf

Utilizando AVR LIBC

- Un retardo de tiempo puede implementarse simplemente así:

…pero es impreciso y difícil de ajustar

```
void delay(void)
{
  unsigned short i;
  for(i = 0; i < 42150; i++)
  { ; }
}
```

- Biblioteca de retardos:
  - Se debe especificar ANTES la constante F_CPU con la frecuencia de reloj [MHz]

```
#define F_CPU 8000000UL
#include <util/delay.h>

        _delay_us(200);   //200 microseconds
        _delay_ms(100);   //100 milliseconds
```

  - Ejemplo: lo veremos en la práctica

Utilizando AVR LIBC

- Bibliotecas de funciones matemáticas (math.h)

Function	Avr2	Avr4 (Hardware MUL)
__addsf3 (1.234, 5.678)	113	108
__mulsf3 (1.234, 5.678)	375	138
__divsf3 (1.234, 5.678)	466	465
acos (0.54321)	4411	2455
asin (0.54321)	4517	2556
atan (0.54321)	4710	2271
atan2 (1.234, 5.678)	5270	2857
cbrt (1.2345)	2684	2555
ceil (1.2345)	177	177
cos (1.2345)	3387	1671
cosh (1.2345)	4922	2979
exp (1.2345)	4708	2765
fdim (5.678, 1.234)	111	111
floor (1.2345)	180	180
fmax (1.234, 5.678)	39	37
fmin (1.234, 5.678)	35	35
fmod (5.678, 1.234)	131	131
frexp (1.2345, 0)	42	41
hypot (1.234, 5.678)	1341	866
ldexp (1.2345, 6)	42	42
log (1.2345)	4142	2134
log10 (1.2345)	4498	2260
modf (1.2345, 0)	433	429
pow (1.234, 5.678)	9293	5047
round (1.2345)	150	150
sin (1.2345)	3353	1653
sinh (1.2345)	4946	3003
sqrt (1.2345)	494	492
tan (1.2345)	4381	2426
tanh (1.2345)	5126	3173
trunc (1.2345)	178	178

Accessing Flash

```
#include <avr/pgmspace.h>

const unsigned char PROGMEM lookup[] ={5,6,7,4};

int main(void)
{
  unsigned char a;
  a = pgm_read_byte(&lookup[i]);

  while (1);
}
```

Puede ser también pgm_read_word, lword o float

Accessing EEPROM

```
#include <avr/io.h>
#include <avr/eeprom.h>

unsigned char EEMEM myVar; //reserve a location in EEPROM

int main(void)
{
  DDRC = 0xFF;

  if((PINB&(1<<0)) != 0)  //if PB0 is HIGH
    eeprom_write_byte(&myVar,'G'); //read from EEPROM
  else
    PORTC = eeprom_read_byte(&myVar); //write to EEPROM

  while (1);
}
```

o sus variantes:
_word
_lword
_float

Bibliografía

Libros:

- *Los Microcontroladores AVR de ATMEL*. Felipe Espinoza. 2012. (CH3)
- *The AVR Microcontroller and Embedded System*. Mazidi. 2011. (CH7 y APENDICE D)

Manuales:

- *Microchip Atmel-Studio-7-User-Guide (pdf)*

Apuntes:

- *CProgrammingInAtmelStudio7 (pdf)*. Naimi. 2017.
- *UsingArduinoBoardsInAtmelStudio7 (pdf)*. Naimi. 2017.

Bibliotecas:

- *AVR Libc*
  - *AVR Libc Reference Manual* (avr-libc-user-manual-2.2.0.pdf)
  - *AVR Libc Reference Manual* (On-line)