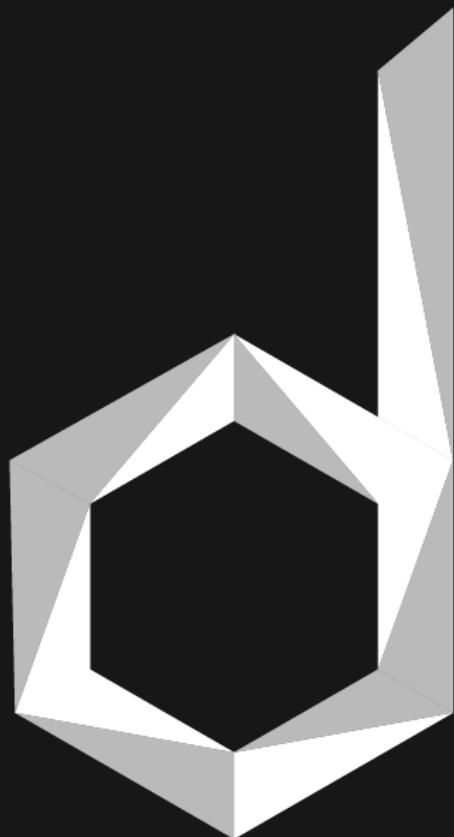


INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

LENGUAJE C, PROGRAMACIÓN MICROCONTROLADORES: ATMEGA328P (ARDUINO)

MICROCHIP STUDIO

Introducción a
Microchip Studio

Contenido

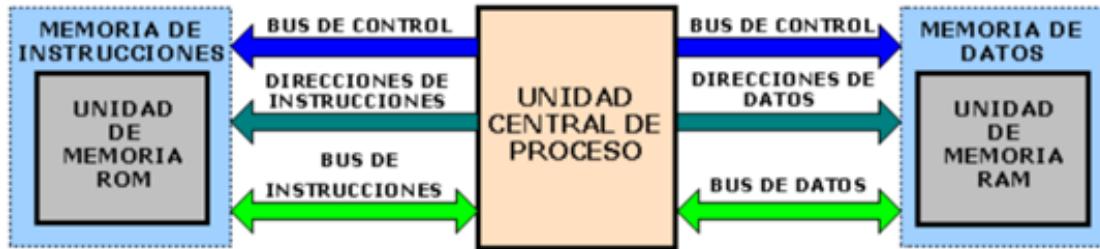
Estructura de un Microcontrolador	2
Pines del Microcontrolador ATMEGA328	4
Configuración Arduino – Microchip Studio.....	8
Nuevo proyecto - Microchip Studio.....	13
Simulación de un Programa en Microchip Studio.....	23
Puertos B, C y D Físicos en el Arduino.....	25
Conexión Física para Insertar Datos a los Pines del Arduino UNO	26
Programar Físicamente la Placa Arduino UNO con el Programa de Microchip Studio.....	27
LCD (Liquid Crystal Display).....	28
Máscara AND	28
Máscara OR y XOR	28



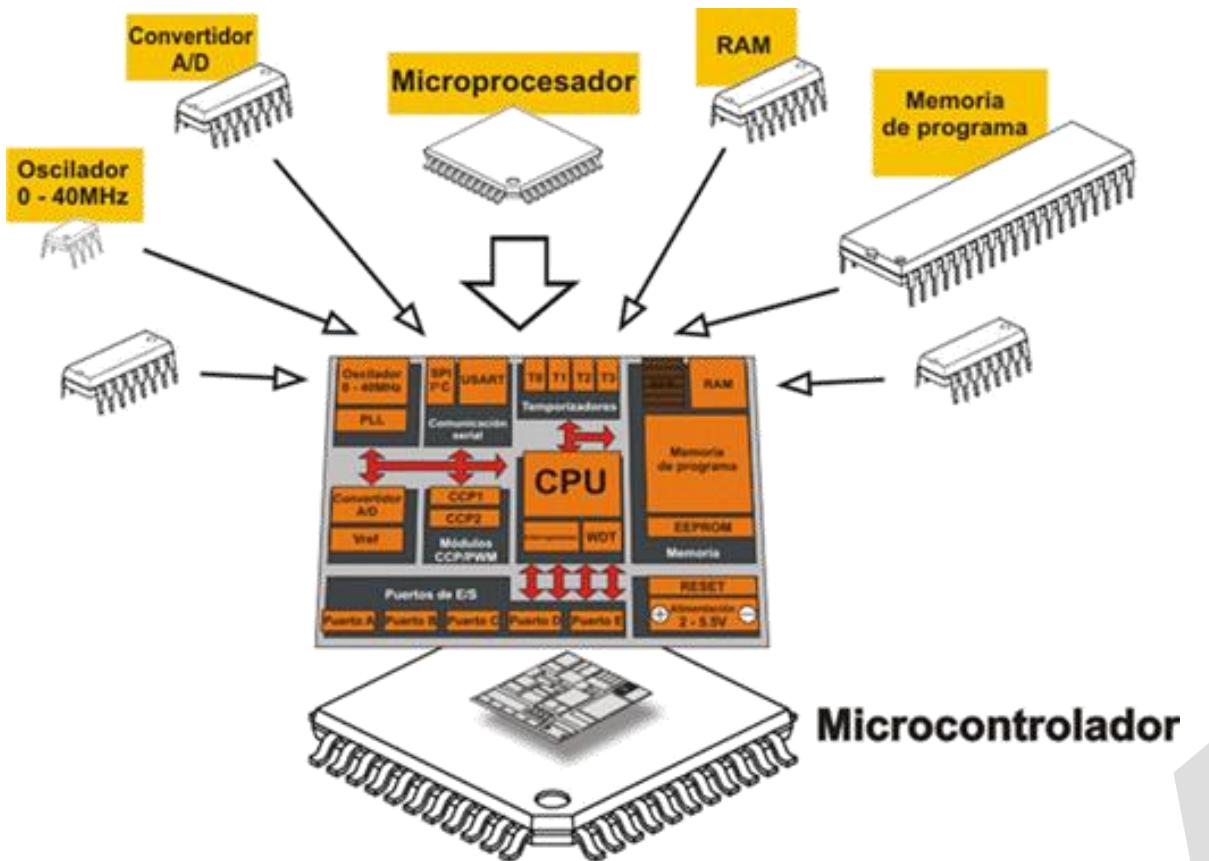
Estructura de un Microcontrolador

Se utilizará el microcontrolador ATMEGA328P el cual utiliza una arquitectura Harvard en su composición:

ARQUITECTURA HARVARD



Los requerimientos básicos necesarios para que un microcontrolador funcione son su alimentación, un oscilador y una memoria (dentro de la cual existen muchos tipos de registros). Aunque además consta de otros elementos.



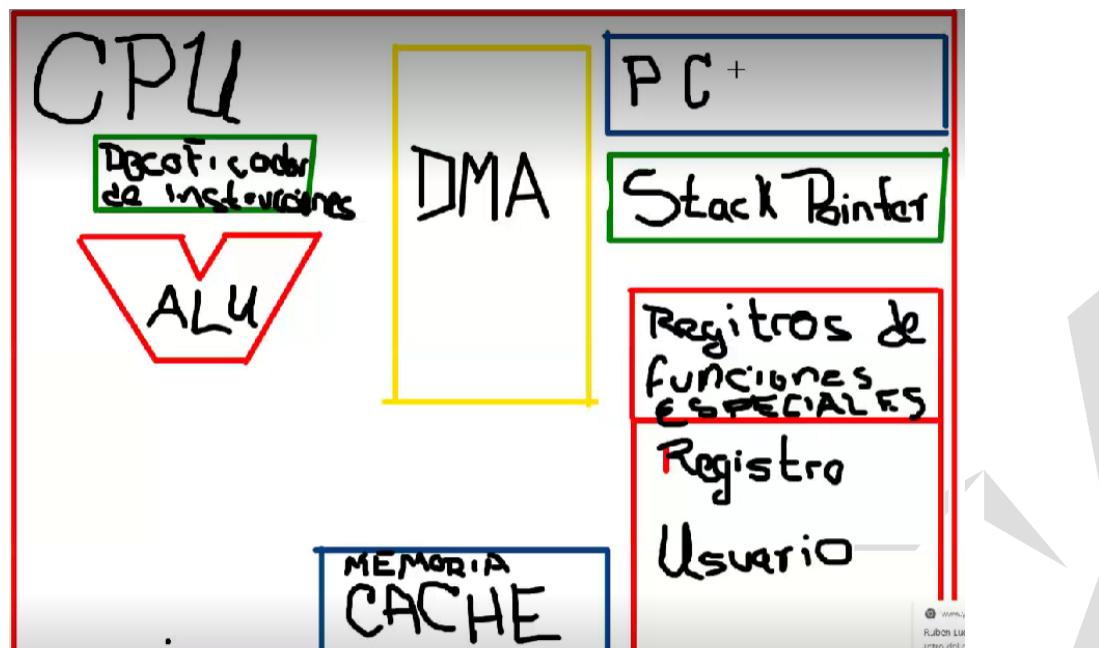
El CPU o microprocesador se conforma de las siguientes unidades:

- **ALU (Unidad lógica aritmética):** Esta unidad sirve para realizar las operaciones matemáticas y lógicas básicas.
- **Contador de programa o Program Counter (PC):** Es un registro de memoria de corto plazo que contiene la dirección de la memoria del programa (memoria FLASH) de la siguiente instrucción a ejecutar, osea que accede a la siguiente línea de código que se va a correr, es un contador

interno que empieza a correr desde cero después de un reset aplicado al ATMEGA328P y se incrementa de 1 en 1 siguiendo el paso de la señal de RELOJ o CLK.

El contador de programa (en inglés Program Counter o PC), también llamado Puntero de instrucciones (Instruction Pointer), es un registro del CPU que indica la posición donde está el procesador en su secuencia de instrucciones.

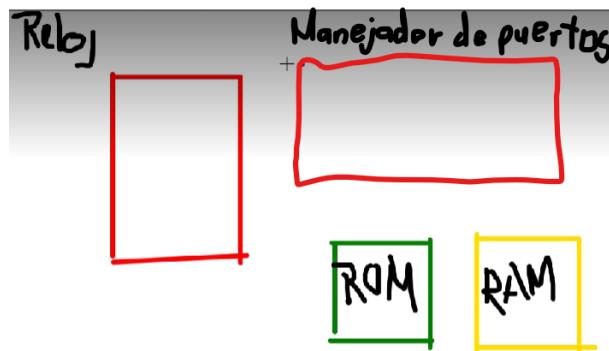
- **Pila (Stack Pointer):** Fragmento de la RAM diseñado para guardar el contenido del contador de programa (direcciones de la memoria FLASH en el microcontrolador) y sirve para regresar el programa principal a una línea de código específica accediendo a cierto valor del PC (Program Counter o Contador de Programa) cuando se usan subrutinas o interrupciones.
- **Memoria caché:** Memoria de uso general que sirve para almacenar datos temporales.
- **Registro de propósito específico:** Registro de la RAM que sirve para configurar el microcontrolador y recibir información de resultados y eventos generados por el procesamiento o por eventos acontecidos dentro o fuera del dispositivo.
 - Son registros que se utilizan para una tarea determinada, están asociados a las unidades funcionales del microcontrolador y cumplen tareas específicas como activar el contador, registros de direcciones de memoria, registro de instrucciones, registro de estado, watchdog, etc.
- **Registro de estado (STATUS) o chismoso:** Es un registro con banderas que dejan constancia de algunas condiciones que se dieron en la última operación realizada, son los que toman en cuenta acarreos en la suma o resta binaria.
 - El registro de STATUS me permite pasar de un banco de registros a otro.
- **Watchdog:** Es un temporizador que irá continuamente incrementando un contador para evitar que ocurran errores inesperados cuando el micro esté en marchas forzadas o entornos dañinos.
 - En electrónica, un perro guardián (en inglés watchdog) es un mecanismo de seguridad que provoca un reset del sistema en caso de que este se haya bloqueado. Consiste en un temporizador que irá continuamente decrementando un contador, inicialmente con un valor relativamente alto.
- **DMA (Direct Memory Access):** Es una herramienta que permite mover datos de una memoria a otra, sin necesidad de usar la ALU.



Pines del Microcontrolador ATMEGA328

(PCINT14/RESET)	PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD)	PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD)	PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0)	PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1)	PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0)	PD4	6	23	PC0 (ADC0/PCINT8)
VCC		7	22	GND
GND		8	21	AREF
(PCINT6/XTAL1/TOSC1)	PB6	9	20	AVCC
(PCINT7/XTAL2/TOSC2)	PB7	10	19	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1)	PD5	11	18	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0)	PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1)	PD7	13	16	PB2 (SS/OC1B/PCINT2)
(PCINT0/CLKO/ICP1)	PB0	14	15	PB1 (OC1A/PCINT1)

Existen registros de propósito específico para activar ciertas funciones, como declarar los puertos como entradas o salidas y se hace de la siguiente manera, usando el manejador de puertos.



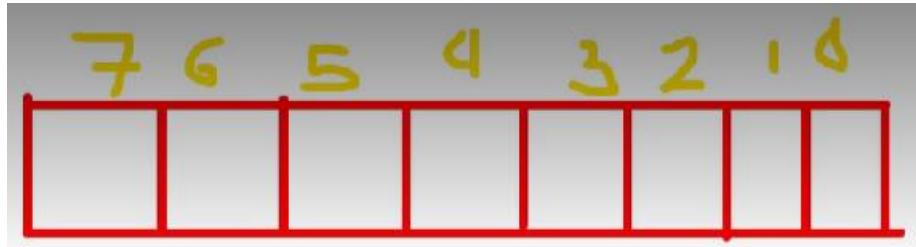
Manejador de Puertos: Es un registro de 8 bits que sirve para indicar la dirección de memoria donde se encontrarán los datos que ingresan o salen del micro, el registro en específico se llama DDR (Data Direction Register) y depende los puertos del microcontrolador, en el microcontrolador Atmega328P solo existen 3 familias de puertos: **B**, **C** y **D**, por lo que solo existen los registros **DDRB**, **DDRC** y **DDRD**.

En un inicio todos los registros de funciones especiales se encuentran como entradas, hasta que se indique lo contrario para cada uno de los pines de las 3 familias de puertos:

- Con un **1** se declara un pin del puerto como **salida**.
- Con un **0** se declara un pin del puerto como **entrada**.

A cada grupo de 4 bits se le llama nibble, los 8 bits del registro de dirección de datos (DDR) se componen de dos conjuntos de nibbles, el nibble alto y el nibble bajo, descritos a continuación:

- **Nibble alto:** Se conforma de los bits 7, 6, 5 y 4.
- **Nibble bajo:** Se conforma de los bits 3, 2, 1 y 0.



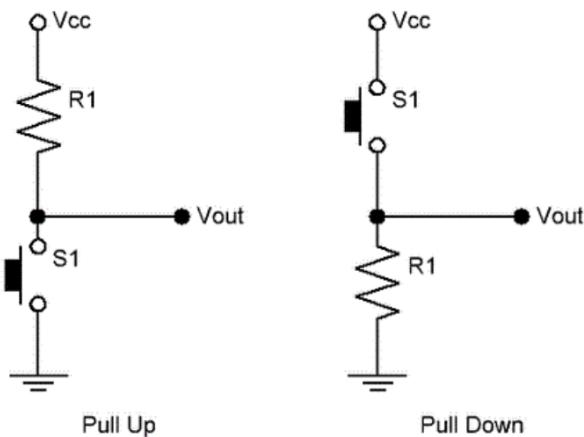
Si por ejemplo si en el registro **DDRB** se declara un pin del puerto como **entrada** asignándole un **0** a su bit y luego en el registro **PORTB** se pone un **1**, lo que pasa es que se activa la resistencia de pull-up, que marca un 1 lógico cuando esté abierto el circuito y 0 lógico cuando esté cerrado, osea que leerá un inverso de la señal de entrada.

Pull-up:

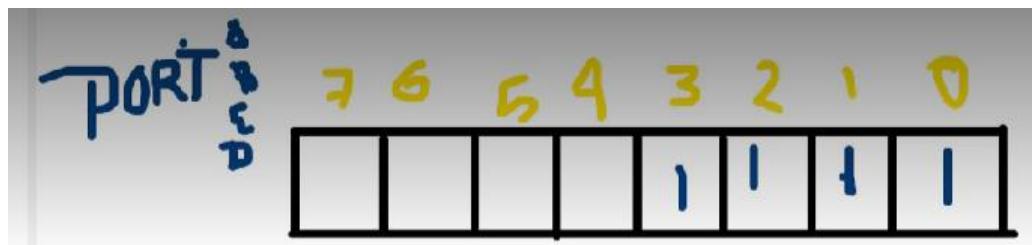
Es una configuración de resistencias que cuando el switch está abierto, entrega un 1 lógico y cuando está cerrado entrega un 0 lógico en el nodo Vout.

Pull-down:

Es una configuración de resistencias que cuando el switch está abierto, entrega un 0 lógico y cuando está cerrado entrega un 1 lógico en el nodo Vout.



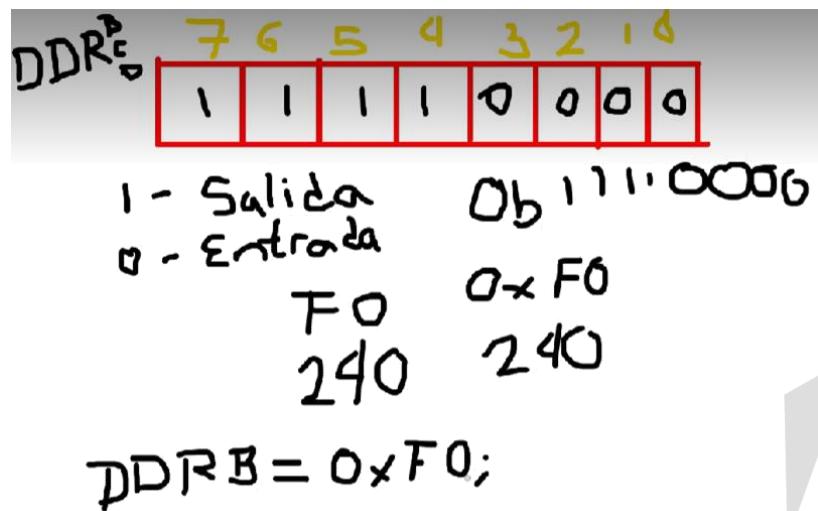
Por lo tanto, en el registro **DDR** se declara si es **entrada** con **0** o **salida** con **1** y en la variable **Port** se indica que va a haber en la salida, con 0 se saca un 0 lógico y con 1 se saca un 1 lógico del pin, pero si en DDR se indicó que es **entrada** con **0** lógico y luego un **1** en la variable **PORT**, se activa la **configuración pull-up**, por lo que la **señal llegará inversa**, si se pone un **0** en **PORT**, se activa la **configuración pull-down** y la señal entrará normal.

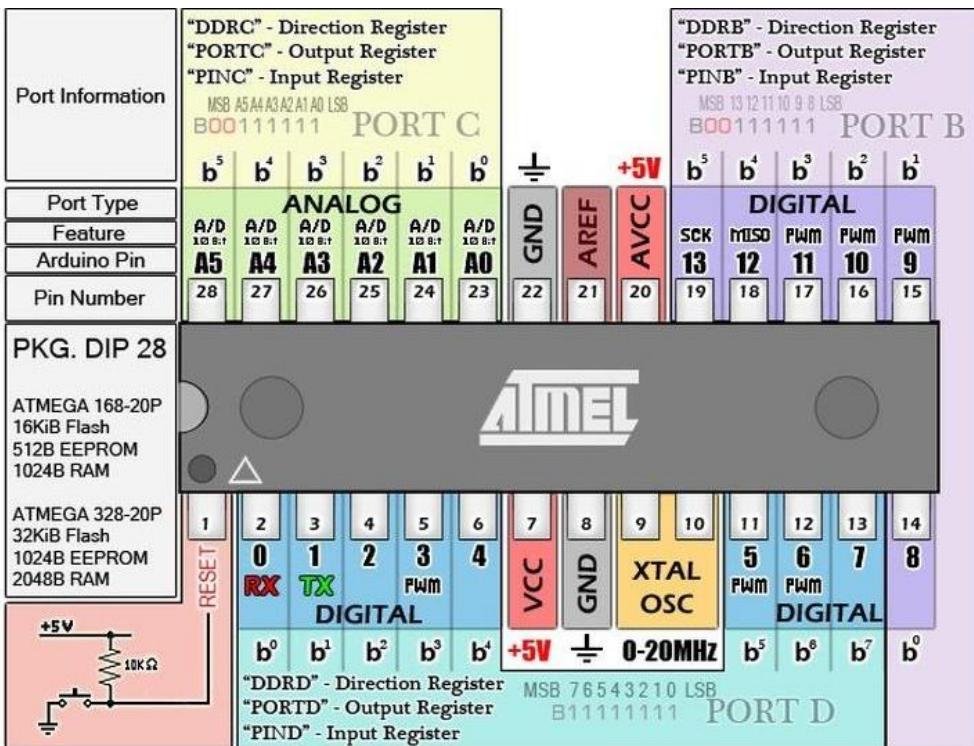


Además de esto, para leer los datos ingresados en los puertos, se usa la variable **PIN** que accede a cada pin de cada puerto **B**, **C** o **D** perteneciente al microcontrolador Atmega328P, en donde se recibirán los bits de la señal de entrada dependiendo de qué se haya puesto en la variable **DDR** y **PORT** anteriormente. El valor de las variables se puede declarar de forma binaria bit por bit o hexadecimal.

Los pines de las 3 familias de puertos del microcontrolador ATMEGA328P son manejados individualmente con su propio bit, cada una de las familias de puertos tienen su propia variable de 8 bits:

- **DDRB**: Con este registro los 6 Pines del puerto **B** se declaran entradas o salidas.
 - **Pines como Entradas**: Declarando los bits del puerto **DDR** como **0**.
 - **PORTB**: Registro que recibe la señal de entrada de una forma o de otra dependiendo del bit que se ponga en cada pin:
 - Si se pone un **1** (Resistencia pull-up): Recibe la señal como su inversa, cuando ingrese un **0** lo convertirá a **1** y cuando ingrese un **1** lo convertirá a **0**.
 - Si se pone un **0** (Resistencia pull-down): Simplemente recibe la señal como viene.
 - **PINB**: Registro que se usa para recibir los datos de entrada, ya habiendo declarado la forma en la que se van a recibir por medio del registro **PORT**.
 - **Pines como Salidas**: Declarando los bits del puerto **DDR** como **1**.
 - **PORTB**: Saca los valores de tensión **0** o **1** en los pines del microcontrolador.
- **DDRC**: Con este registro los 6 Pines del puerto **C** se declaran entradas o salidas.
 - **Pines como Entradas**: Declarando los bits del puerto **DDR** como **0**.
 - **PORTC**: Elige la resistencia **pull-up** o **down** para recibir la señal.
 - **PINC**: Recibe los datos de la señal de entrada.
 - **Pines como Salidas**: Declarando los bits del puerto **DDR** como **1**.
 - **PORTC**: Saca valores de tensión **0** o **1**.
- **DDRD**: Con este registro los 8 Pines del puerto **D** se declaran entradas o salidas.
 - **Pines como Entradas**: Declarando los bits del puerto **DDR** como **0**.
 - **PORTD**: Elige la resistencia **pull-up** o **down** para recibir la señal.
 - **PIND**: Recibe los datos de la señal de entrada.
 - **Pines como Salidas**: Declarando los bits del puerto **DDR** como **1**.
 - **PORTD**: Saca valores de tensión **0** o **1**.



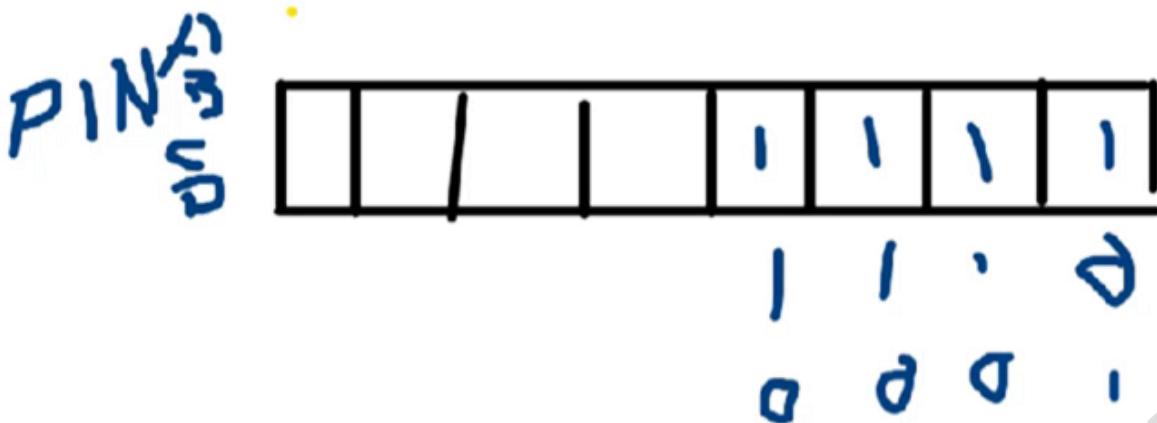


ATMEL - ATMEGA168/328 - 20P

Robin Farrell 2010

En resumen, los registros utilizados para la declaración de puertos son los siguientes:

- **DDR**: Indica cual pin es entrada o salida.
- **PORT**: Saca datos.
- **PIN**: Lee los datos recopilados.



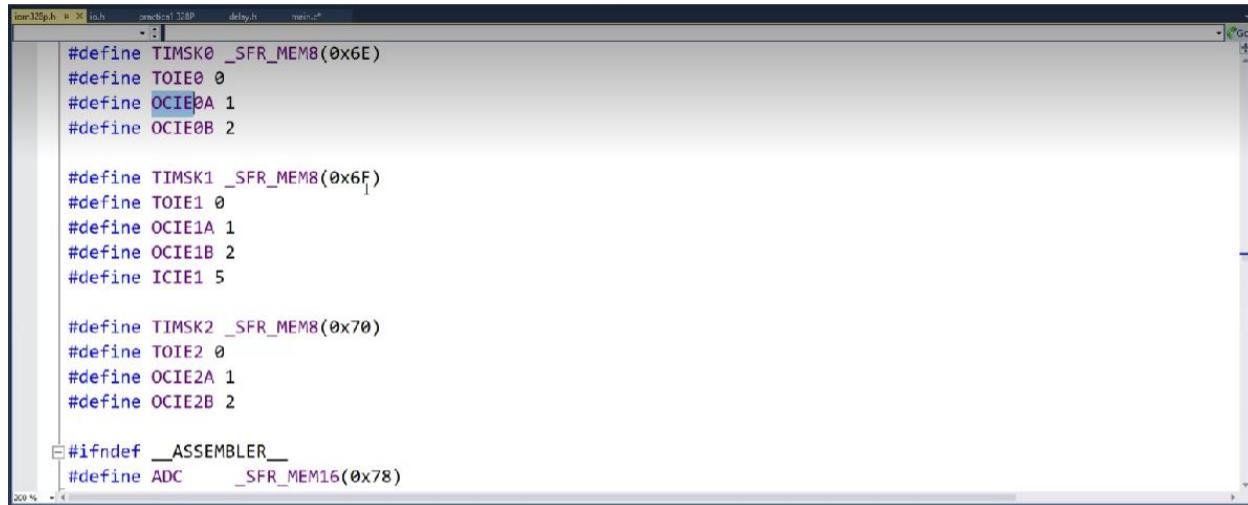
Cada puerto tiene su nombre específico y para usarlos se debe usar la siguiente librería:

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

Que obtiene todos los nombres de los registros de funciones especiales de la librería iom328p, para acceder a los registros del microcontrolador ATMEGA328P, contenida dentro de la librería io (que significa input output), por eso se deben usar los nombres reservados de las variables **DDRB**, **DDRC**, **DDRD**, **PORTB**, **PORTC**, **PORTD**, **PINB**, **PINC** y **PIND**:

```
#include <avr/iom328p.h>
```

Nota: Cuando en la instrucción `#define` existe el nombre de una variable que sea antecedido por un guion bajo es porque esa función escrita con el lenguaje de programación C está mandando a ejecutar un archivo creado con el lenguaje de programación Ensamblador para correr la función, si no está antecedido por un guion antes del nombre de la función es porque ejecuta un programa en C directamente.



```

icon32ip.h  io.h  sometech328P  delay.h  main.c*
[...]
#define TIMSK0 _SFR_MEM8(0x6E)
#define TOIE0 0
#define OCIE0A 1
#define OCIE0B 2

#define TIMSK1 _SFR_MEM8(0x6F)
#define TOIE1 0
#define OCIE1A 1
#define OCIE1B 2
#define ICIE1 5

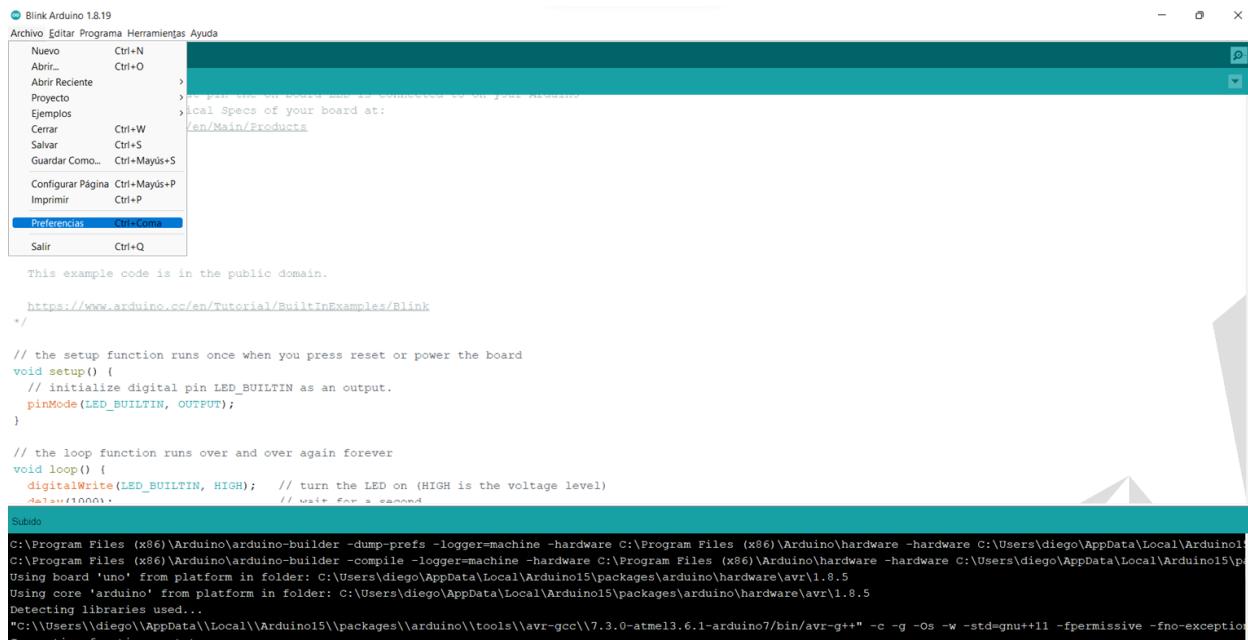
#define TIMSK2 _SFR_MEM8(0x70)
#define TOIE2 0
#define OCIE2A 1
#define OCIE2B 2

#ifndef __ASSEMBLER__
#define ADC    _SFR_MEM16(0x78)
[...]

```

Configuración Arduino – Microchip Studio

Para poder ejecutar cualquier programa en Microchip Studio y que este se pueda cargar al microcontrolador ATMEGA328P del Arduino, primero nos debemos introducir al IDE de Arduino y correr el microcontrolador una vez ahí para así extraer cierta información del microcontrolador que se pasará después al programa de Microchip Studio.



Blink Arduino 1.8.19

Archivo Editar Programa Herramientas Ayuda

- Nuevo Ctrl+N
- Abrir... Ctrl+O
- Abrir Reciente
- Proyecto
- Ejemplos
- Cerrar Ctrl+W
- Salvar Ctrl+S
- Guardar Como... Ctrl+Mayús+S
- Configurar Página Ctrl+Mayús+P
- Imprimir Ctrl+P
- Preferencias Ctrl+Coma**
- Salir Ctrl+Q

This example code is in the public domain.

<https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink>

```

/*
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off (LOW is the voltage level)
  delay(1000);                      // wait for a second
}

```

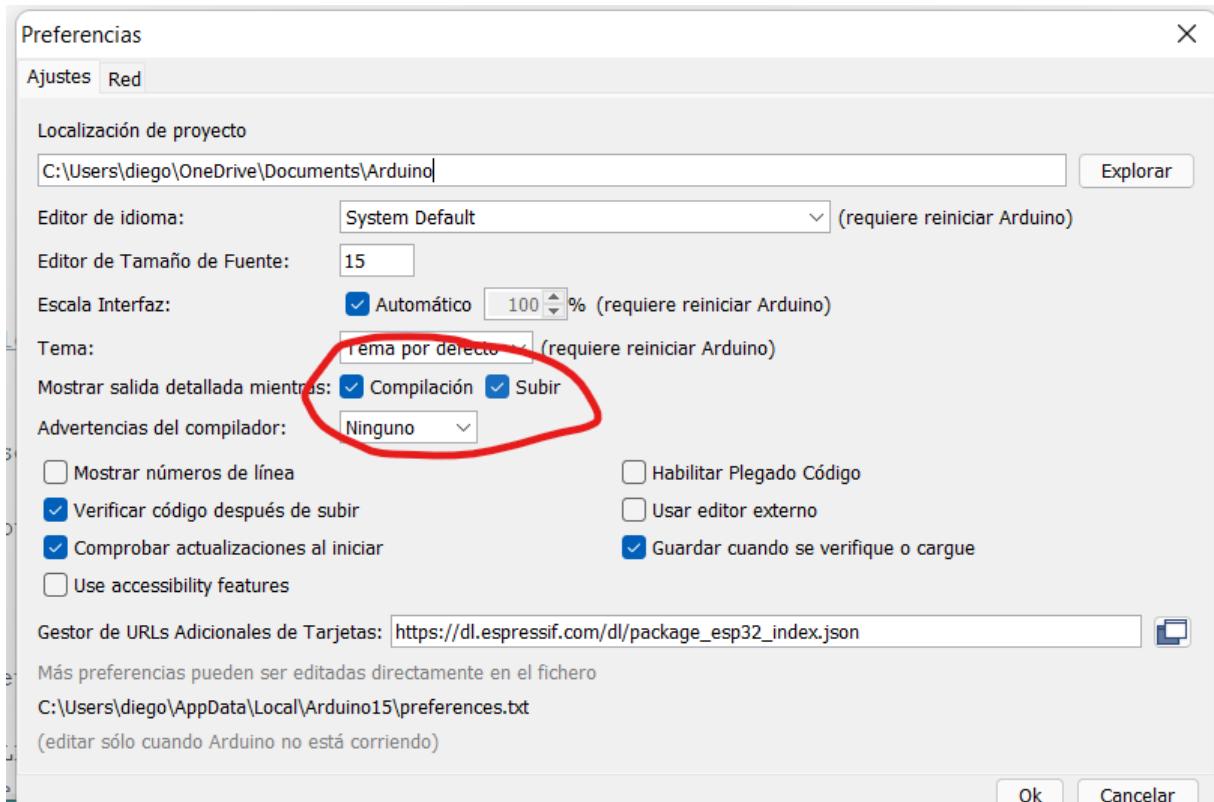
Subido

```

C:\Program Files (x86)\Arduino\arduino-builder -dump-prefs -logger=none -hardware C:\Program Files (x86)\Arduino\hardware -hardware C:\Users\diego\AppData\Local\Arduino15\hardware
C:\Program Files (x86)\Arduino\arduino-builder -compile -logger=none -hardware C:\Program Files (x86)\Arduino\hardware -hardware C:\Users\diego\AppData\Local\Arduino15\hardware
Using board 'uno' from platform in folder: C:\Users\diego\AppData\Local\Arduino15\packages\arduino\hardware\avr\1.8.5
Using core 'arduino' from platform in folder: C:\Users\diego\AppData\Local\Arduino15\packages\arduino\hardware\avr\1.8.5
Detecting libraries used...
"C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7\bin\avr-g++" -c -g -Os -w -std=gnu++11 -fpermissive -fno-exceptions
[...]

```

Las Preferencias del IDE de Arduino deben ser cambiadas de la siguiente manera para que se pueda extraer la información necesaria cuando se ejecute el programa en el ATMEGA328P.



Al hacerlo aparecerá el siguiente mensaje en consola cuando se corra el programa en la placa Arduino:

```

Blink Arduino 1.8.19
Archivo Editar Programa Herramientas Ayuda
Blink
** You must connect your Arduino board and be connected to an USB port. If your model, check the Technical Specs of your board at:
https://www.arduino.cc/en/Main/Products

Subido
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 2039 bytes para las variables locales. El máximo es 2048 bytes.
C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrdude\6.3.0-arduino17\bin\avrdude -CC:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrdude\6.3.0-arduino17/etc/avrdude.conf

avrdude: Version 6.3-20190619
Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
Copyright (c) 2007-2014 Joerg Wunsch

System wide configuration file is "C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrdude\6.3.0-arduino17/etc/avrdude.conf"

Using Port : COM7
Using Programmer : arduino
Overriding Baud Rate : 115200
AVR Part : ATmega328P
Chip Erase delay : 9000 us
PAGE[EL : PDI
BS2
RESET disposition : dedicated
RETRY pulse : SCK
serial program mode : yes
parallel program mode : yes
Timeout : 200
StabDelay : 100
CmdexeDelay : 25
SyncLoops : 32
ByteDelay : 0
PollIndex : 3
PollValue : 0x53
Memory Detail :

```

Ahora debemos copiar la dirección que viene indicada cuando se selecciona la opción de Mostrar salida detallada mientras se sube el programa a la placa Arduino, solo para esto es que se necesita usar una vez el Arduino IDE:

```

Blink Arduino 1.8.19
Archivo Editar Programa Herramientas Ayuda
Subido
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 2039 bytes para las variables locales. El máximo es 2048 bytes.
C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrduude\6.3.0-arduino17\bin\avrduude -C:C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrduude\6.3.0-arduino17/etc\avrduude.conf

avrduude: Version 6.3-20190619
Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
Copyright (c) 2007-2014 Joerg Wunsch

System wide configuration file is "C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrduude\6.3.0-arduino17/etc\avrduude.conf"

Using Port           : COM7
Using Programmer    : arduino
Overriding Baud Rate: 115200
AVR Part             : ATmega328P
Chip Erase delay   : 9000 us
FPGEL               : FD7
BS2                 : FC2
RESET disposition  : dedicated
RETRY pulse         : SCK
serial program mode: yes
parallel program mode: yes
Timeout              : 200
StabDelay            : 100
CmdexeDelay          : 25
SyncLoops            : 32
ByteDelay            : 0
PollIndex            : 3
PollValue             : 0x53
Memory Detail        :

```

Los pasos por seguir para extraer las direcciones del mensaje de consola del IDE de Arduino son:

1. Se extrae este mensaje de la consola al ejecutar el programa del Arduino y subirlo a la placa:

```
C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrduude\6.3.0-
arduino17\bin\avrduude
CC:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrduude\6.3.0-
arduino17/etc\avrduude.conf -v -patmega328p -carduino -PCOM7 -b115200 -D -
Uflash:w:C:\Users\diego\AppData\Local\Temp\arduino_build_963165/Blink.ino.hex:i
```

2. De este solo necesitaremos la última parte de la dirección copiada, donde al final de la parte que dice `avrduude` que es el compilador que utiliza Arduino IDE, se coloca la extensión `.exe`.

```
C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrduude\6.3.0-
arduino17\bin\avrduude.exe
```

3. Ahora agarraré la segunda parte del mensaje inicial, le agregaré un `-` al inicio y añadiré unas comillas dobles, cerrándolas hasta donde dice `.conf`.

```
-C"C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrduude\6.3.0-
arduino17/etc\avrduude.conf" -v -patmega328p -carduino -PCOM7 -b115200 -D -
Uflash:w:C:\Users\diego\AppData\Local\Temp\arduino_build_963165/Blink.ino.hex:i
```

4. En esta última dirección tenemos que tener cuidado ya que se indica el microcontrolador que se está utilizando `-patmega328p` y el puerto COM donde está conectado `-PCOM7`, lo demás se debe dejar tal cual como está, pero borrando lo que viene después de `w:` y añadiendo la línea `$(TargetDir) $(TargetName).hex:i`.

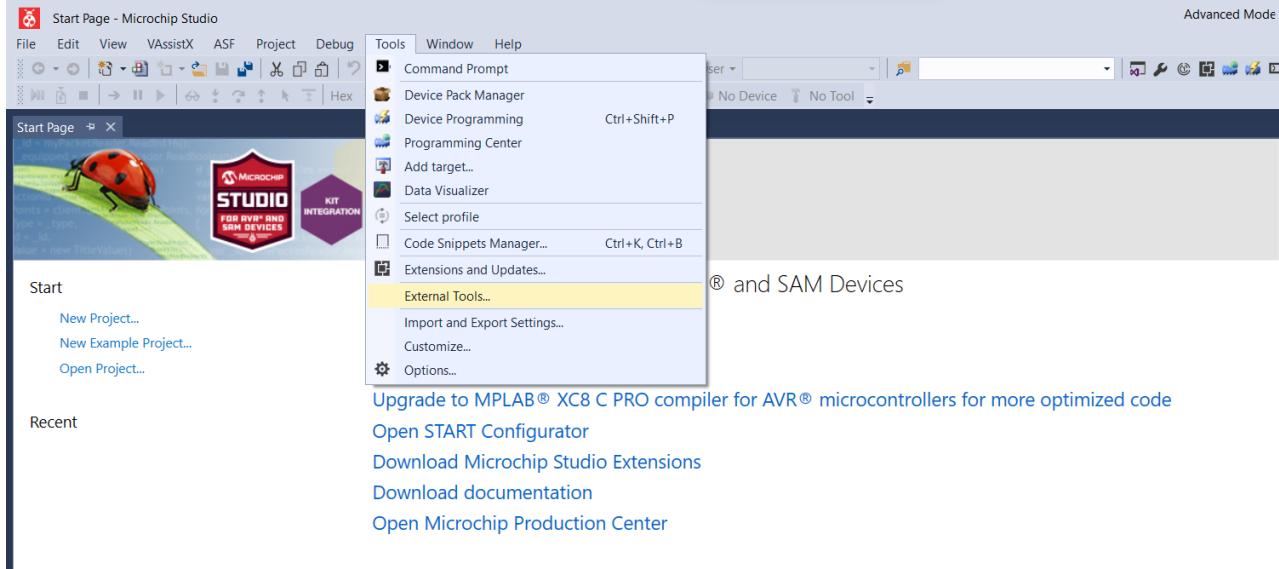
```
-C"C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrduude\6.3.0-
arduino17/etc\avrduude.conf" -v -patmega328p -carduino -PCOM7 -b115200 -D -Uflash:w: $(TargetDir)
$(TargetName).hex:i
```

5. Finalmente, las direcciones con las que terminamos y que se deben subir al editor de código de Microchip Studio son las siguientes:

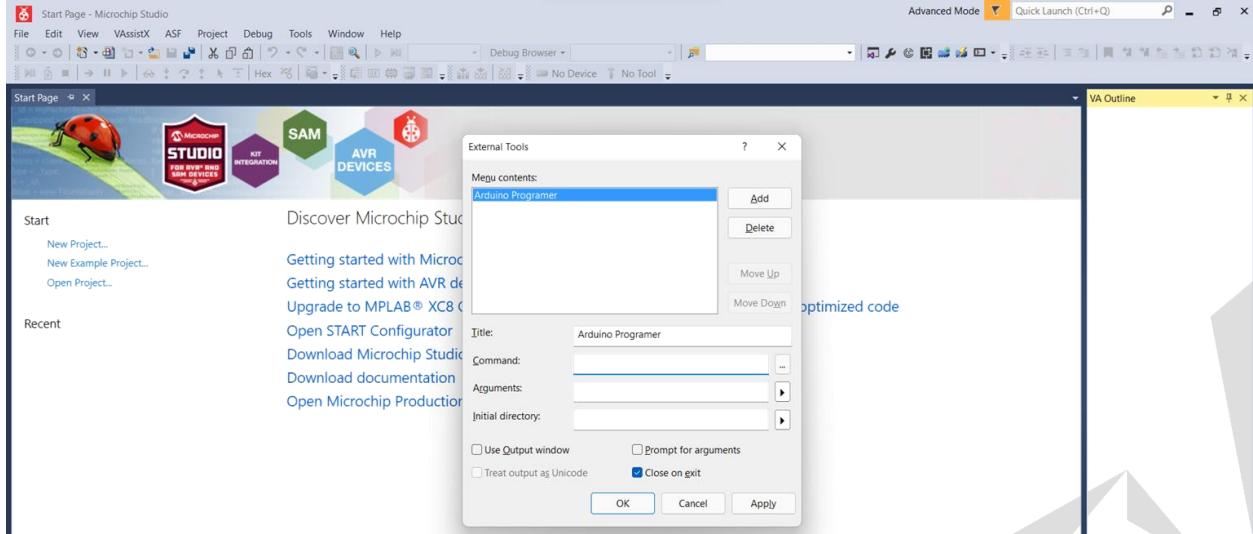
C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrdude\6.3.0-arduino17\bin\avrdude.exe

-C"C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrdude\6.3.0-arduino17/etc/avrdude.conf" -v -patmega328p -carduino -PCOM7 -b115200 -D -Uflash:w: \$(TargetDir) \$(TargetName).hex:i

6. Y estas se deben introducir en la siguiente parte de Microchip Studio:

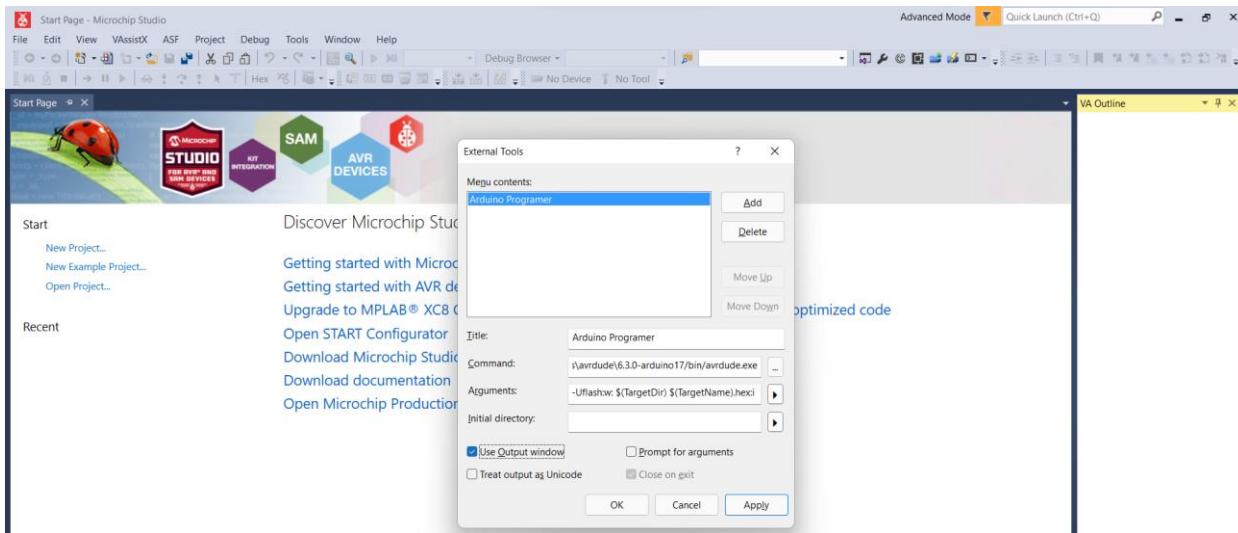


En la parte donde dice **Command** se pone la primera ruta y en la parte donde dice **Arguments** se coloca la segunda:

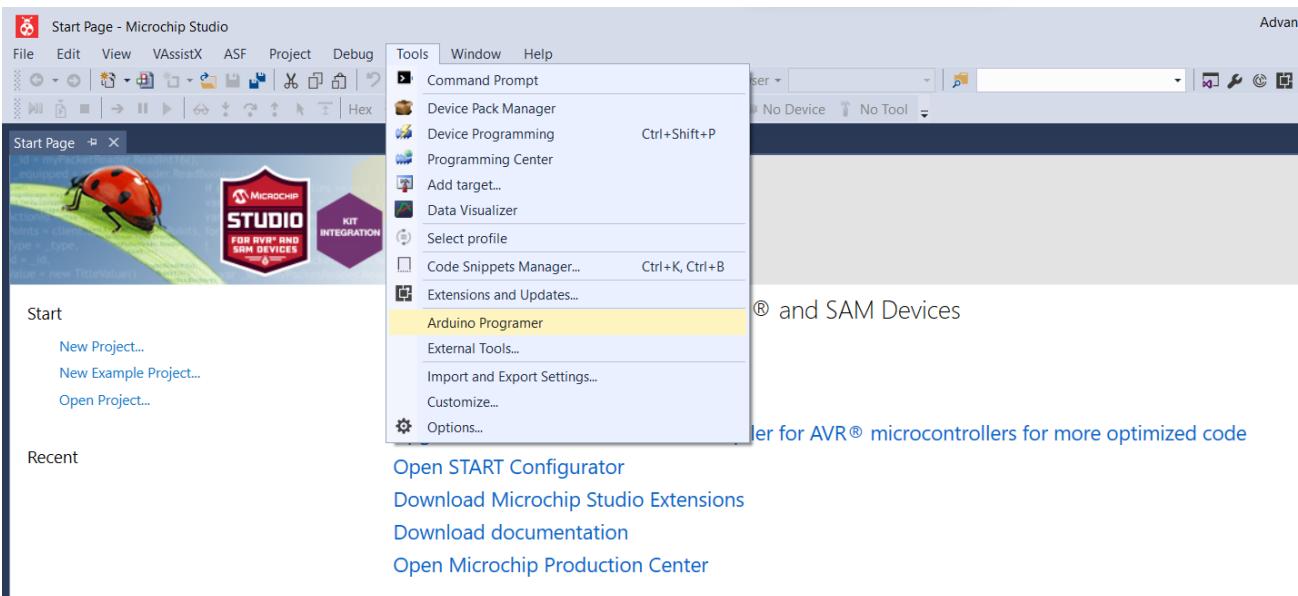


Al final solo se debe seleccionar el checkbox que dice **Use Output Window** y daremos clic primero en el botón de **Apply** y posteriormente en el de **Ok** para que quede configurado correctamente Microchip

Studio, creando así un nuevo programador que me permita subir mi programa al microcontrolador ATMEGA328P, para reconocer ese programador se le asigna un nombre en la parte donde dice **Title**:



Todo lo anterior se realizó para que finalmente dentro del mismo menú de Tools se pueda seleccionar el nuevo programador creado y se pueda programar el Arduino a través de Microchip Studio.



Al ejecutar el programa lo que pasará es que en la consola debe aparecer el puerto que se está usando, que es el mismo que se había puesto en la segunda dirección asignada a la parte que dice **Arguments**, específicamente donde dice **-PCOM7**.

Además, deberá aparecer en consola el microcontrolador que se está programando, que es el mismo que se había puesto en la segunda dirección asignada a la parte que dice **Arguments**, específicamente donde dice **-patmega328p**.

Para que la configuración de Microchip Studio funcione se debe copiar a fuerza las direcciones antes mencionadas desde el bloque de notas, si se copia desde Word habrá un error.

The screenshot shows the Microchip Studio interface. At the top, there's a toolbar with various icons for file operations, search, and debugging. Below the toolbar is a banner with the Microchip logo and text: "FOR AVR® AND SAM DEVICES", "KIT INTEGRATION", "SAM", and "AVR DEVICES". The main area is titled "Discover Microchip Studio for AVR® and SAM Devices". On the left, there's an "Output" window showing the configuration for "avrduude.exe". The configuration includes details like port (COM7), programmer (arduino), baud rate (115200), and AVR part (ATmega328P). The "Output" window also indicates "Ready".

```

avrduude.exe: Version 6.3-20190619
Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
Copyright (c) 2007-2014 Joerg Wunsch

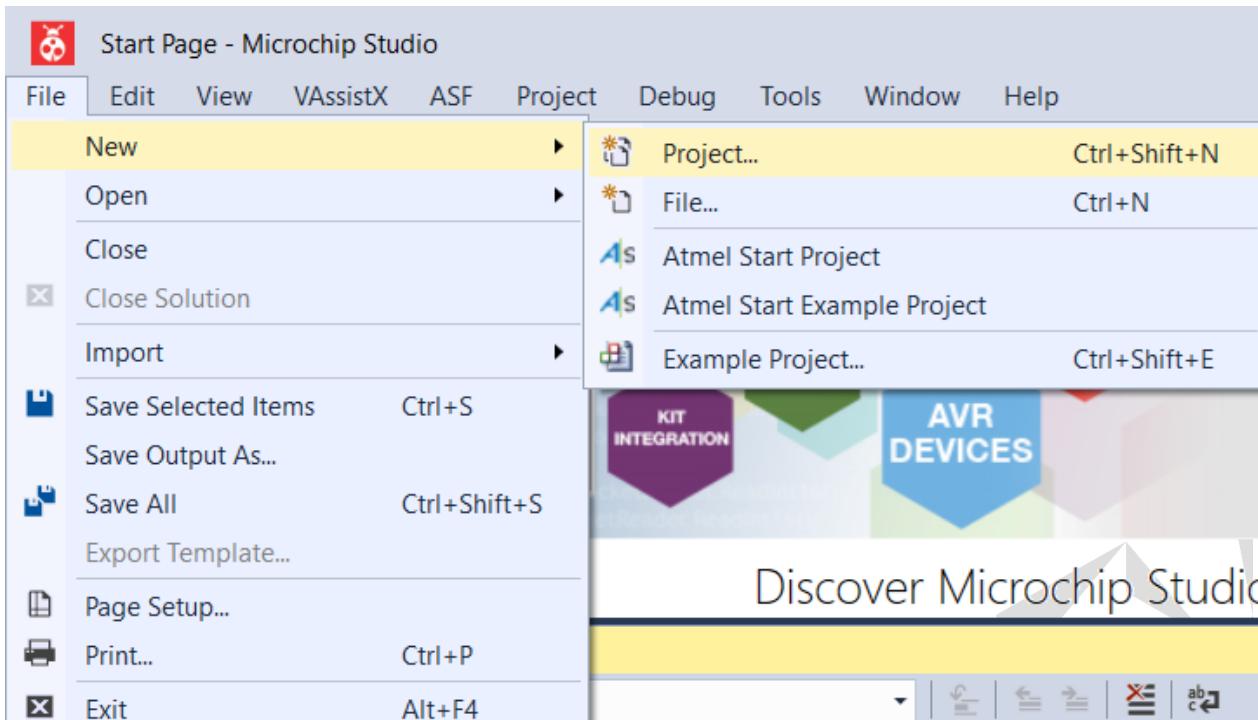
System wide configuration file is "C:\Users\diego\AppData\Local\Arduino15\packages\arduino\tools\avrduude\6.3.0-arduino17/etc/avrduude.conf"

Using Port : COM7
Using Programmer : arduino
Overriding Baud Rate : 115200
AVR Part : ATmega328P
Chip Erase delay : 9000 us
PAGEL : P07
BS2 : PC2
RESET disposition : dedicated
RETRY pulse : SCK
serial program mode : yes
parallel program mode : yes
Timeout : 2000
StrobeDelay : 100
CedexDelay : 25
SyncLoops : 32
ByteDelay : 0
PollIndex : 3
PollValue : 0x53
Memory Detail :

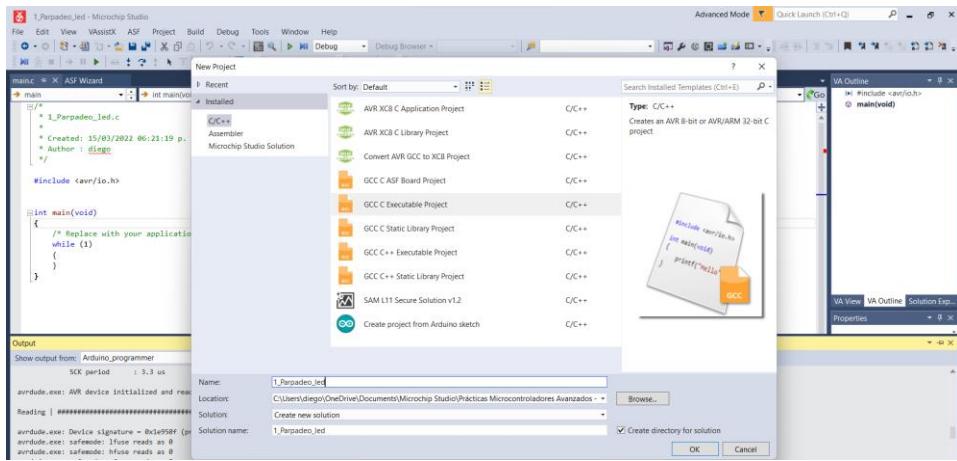
```

Esa es solo la configuración para que se pueda programar el Arduino desde Microchip Studio, pero ya que se cree un archivo con el lenguaje de programación C para programar el Arduino, se deberá seleccionar el programador creado para que se cargue el Programa a la placa Arduino UNO.

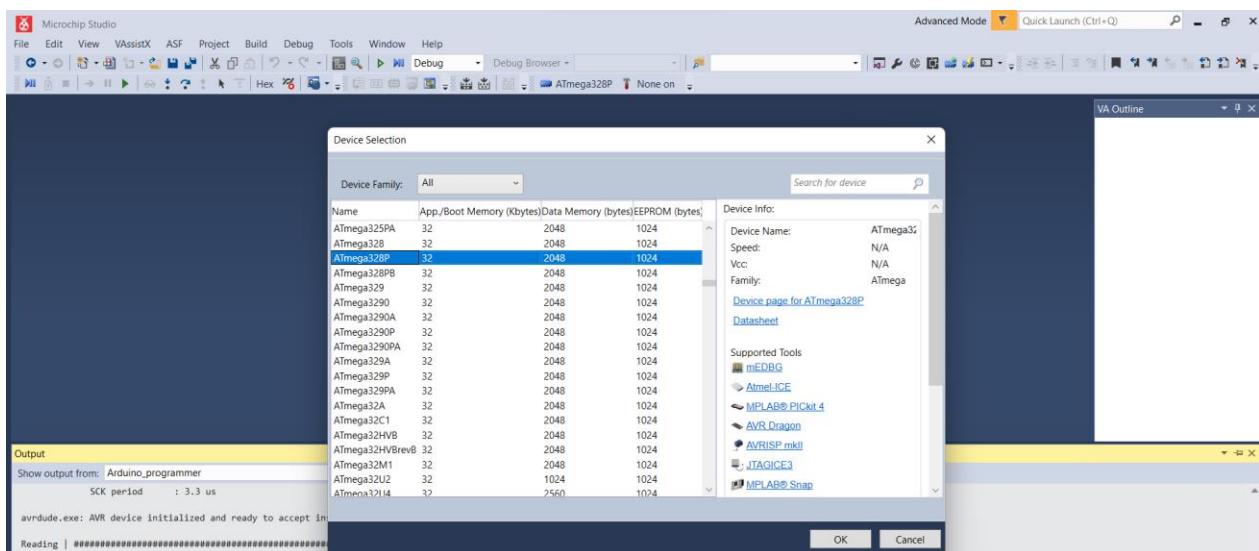
Nuevo proyecto - Microchip Studio



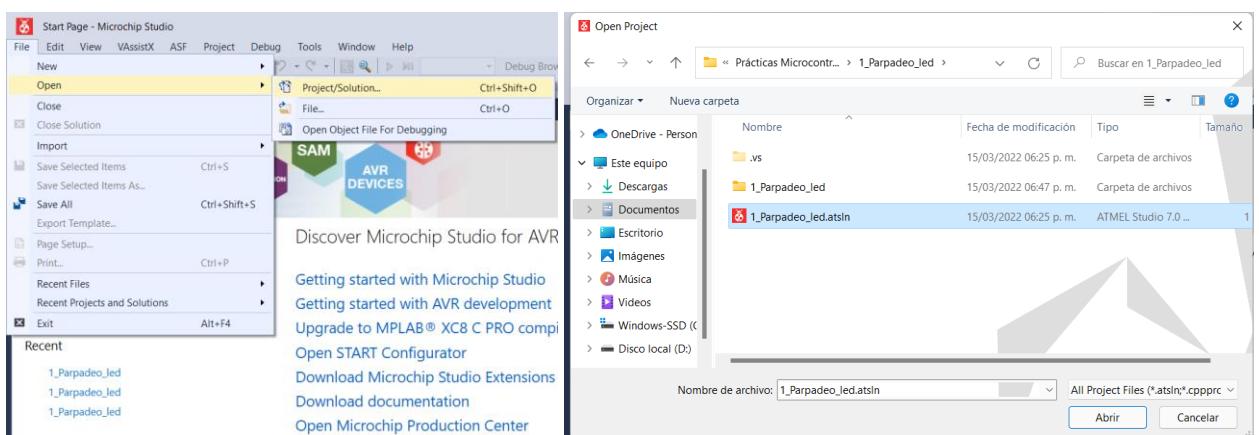
*Nota: Si se quisiera crear una librería personalizada usando el lenguaje de programación C, se seleccionaría la opción **GCC C Static Library Project**.*



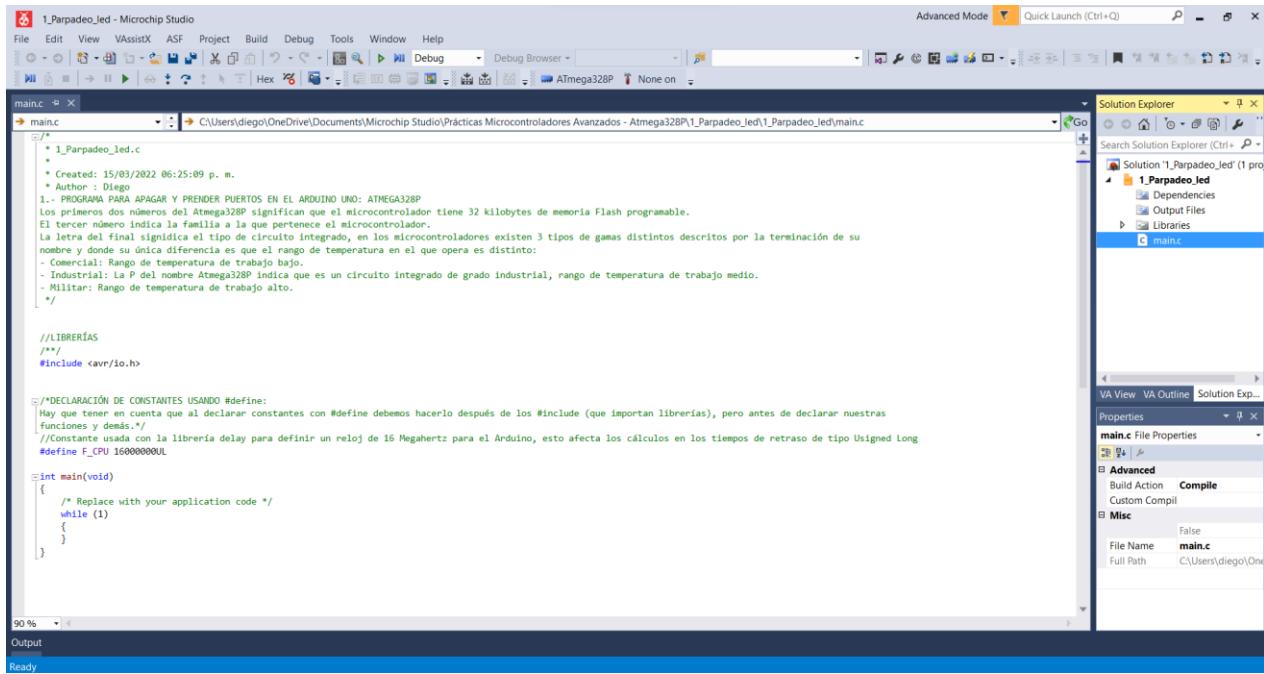
Siempre al crear un nuevo proyecto se debe seleccionar el microcontrolador en el cual se quiere utilizar, que en este caso es el ATmega328P.



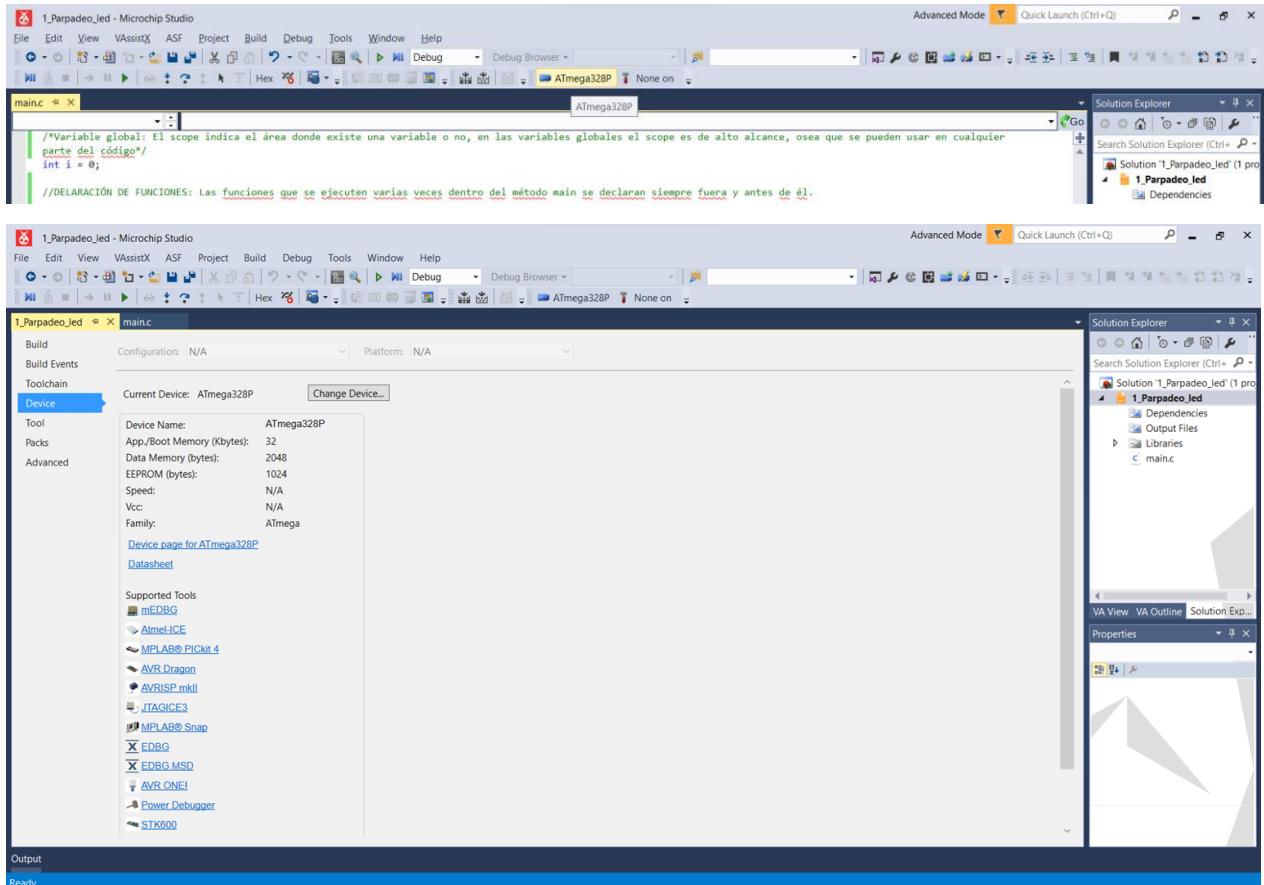
Para abrir proyectos existentes el proceso es el siguiente:



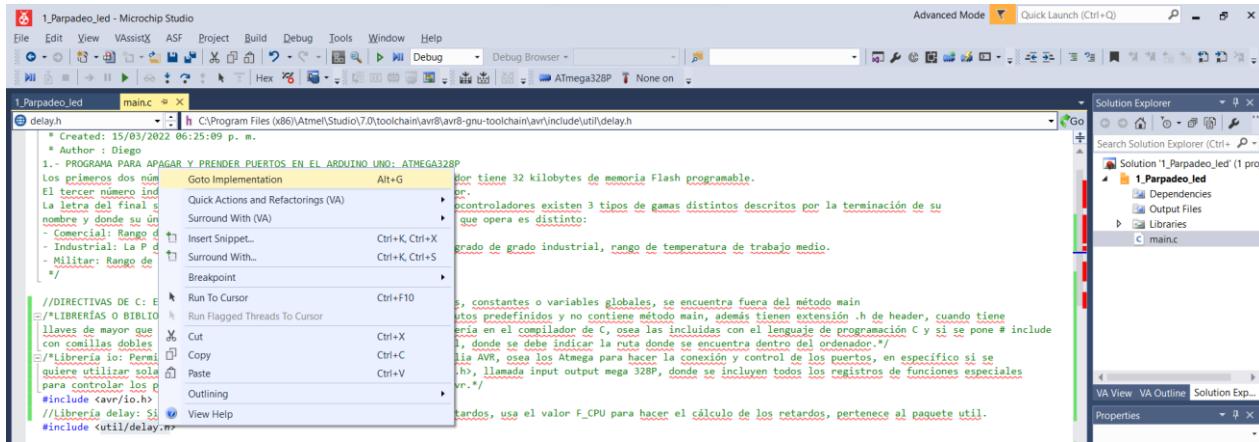
Esta es el área de trabajo del IDE de Microchip Studio.



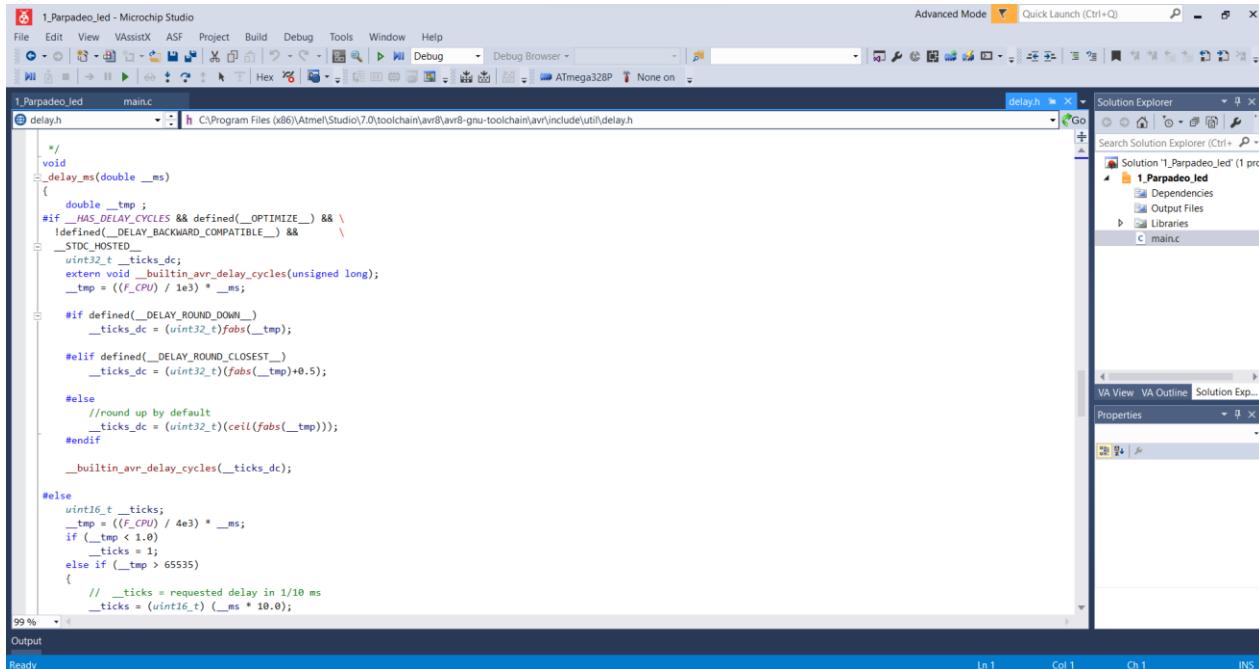
En la parte de arriba de Microchip Studio se indica que microcontrolador se está utilizando y al dar clic sobre ese botón sale una explicación detallada de él.



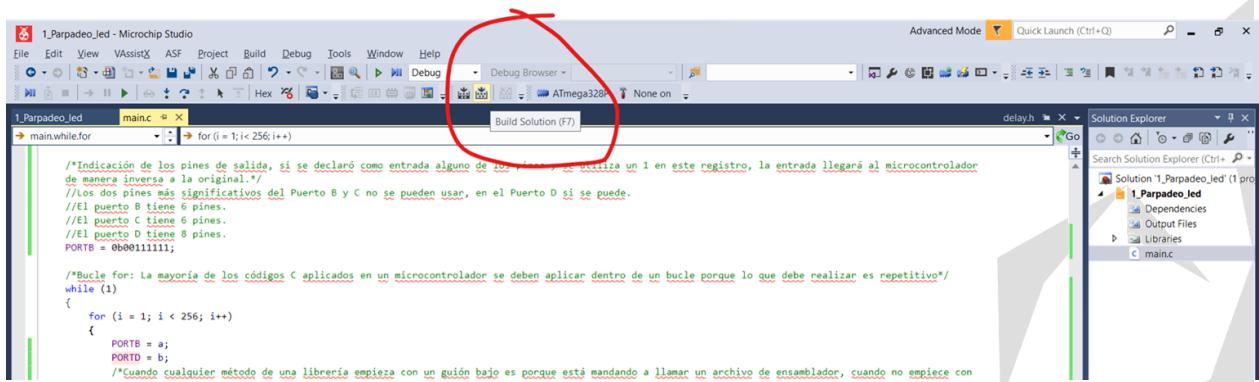
Nota: Para acceder al contenido de una librería utilizada en el código se debe seleccionar el nombre de la librería, dar clic derecho sobre ella y seleccionar la opción de **Goto Implementation**:



Dentro del contenido se podrán ver los distintos atributos y métodos que incluye la librería.



Para compilar el código y ver que este no contenga errores se usa el botón de Build solution.



Y al compilar se podrá ver un mensaje en la consola de abajo que se construyó correctamente el código.

The screenshot shows the Microchip Studio interface. The code editor displays C code for a project named '1_Parpadeo_led'. The Solution Explorer shows the project structure with files like 'delay.h' and 'main.c'. The Output window at the bottom shows a successful build with no errors or warnings.

```

delay.h
main.c

for (i = 1; i < 256; i++)
{
    PORTB = a;
    PORTD = b;
    /*Cuando cualquier método de una librería empieza con un guión bajo es porque está mandando a llamar un archivo de ensamblador, cuando no empiece con
    //delay_ms(): Retardo de tiempo indicado en milisegundos, usa la variable F_CPU
    _delay_ms(500);
    PORTB = b;
    PORTD = a;
    _delay_ms(500);
    PORTB = c;
    PORTD = c;
    //delay_us(): Retardo de tiempo indicado en microsegundos, usa la variable F_CPU.
    _delay_us(500);
}
}

Error List
Output
Build succeeded

```

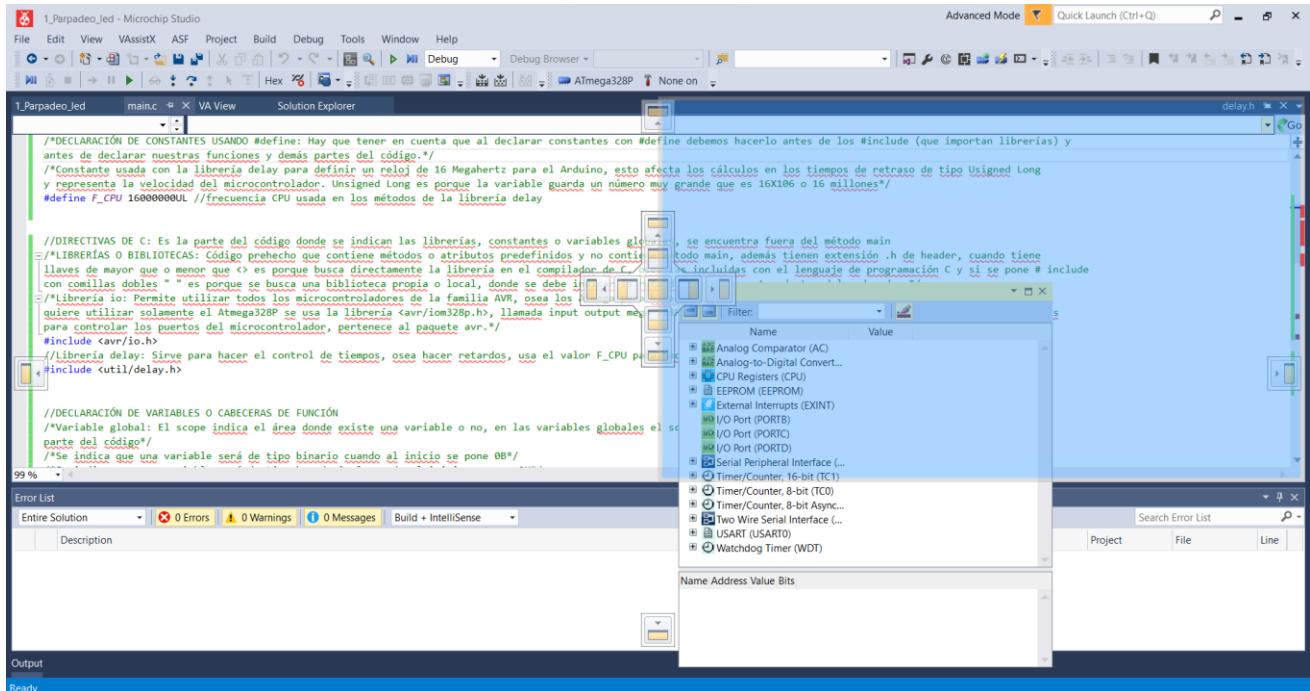
Dando clic en la parte izquierda de la pantalla de color gris se pueden crear Breaking Points, en estos se detiene la ejecución del código para analizarlo por partes, dando clic de nuevo sobre el break point este se quita.

The screenshot shows the Microchip Studio interface with a red circle highlighting a break point (a small red dot) in the code editor. The code editor displays the same C code as the previous screenshot, with the break point set on the line 'DORB = 0x00011111;'. The Solution Explorer is visible on the right.

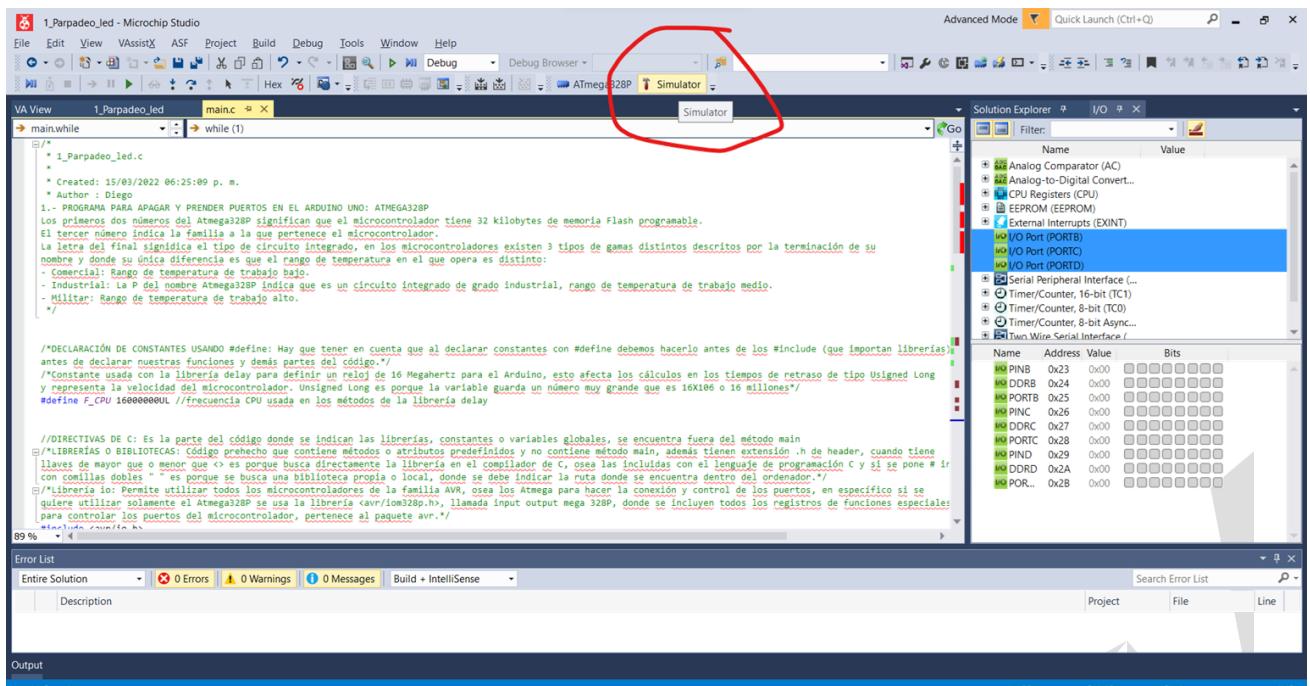
En la parte superior también se tienen varios controles, uno muy importante es el de *I/O*, este se refiere a las entradas o salidas que puede tener el código y se usa para ejecutar simulaciones, medir tiempos de ejecución, analizar señales de entrada, etc.

The screenshot shows the Microchip Studio interface with a red circle highlighting the 'I/O' button in the toolbar. The code editor displays the same C code as the previous screenshots. The Solution Explorer is visible on the right.

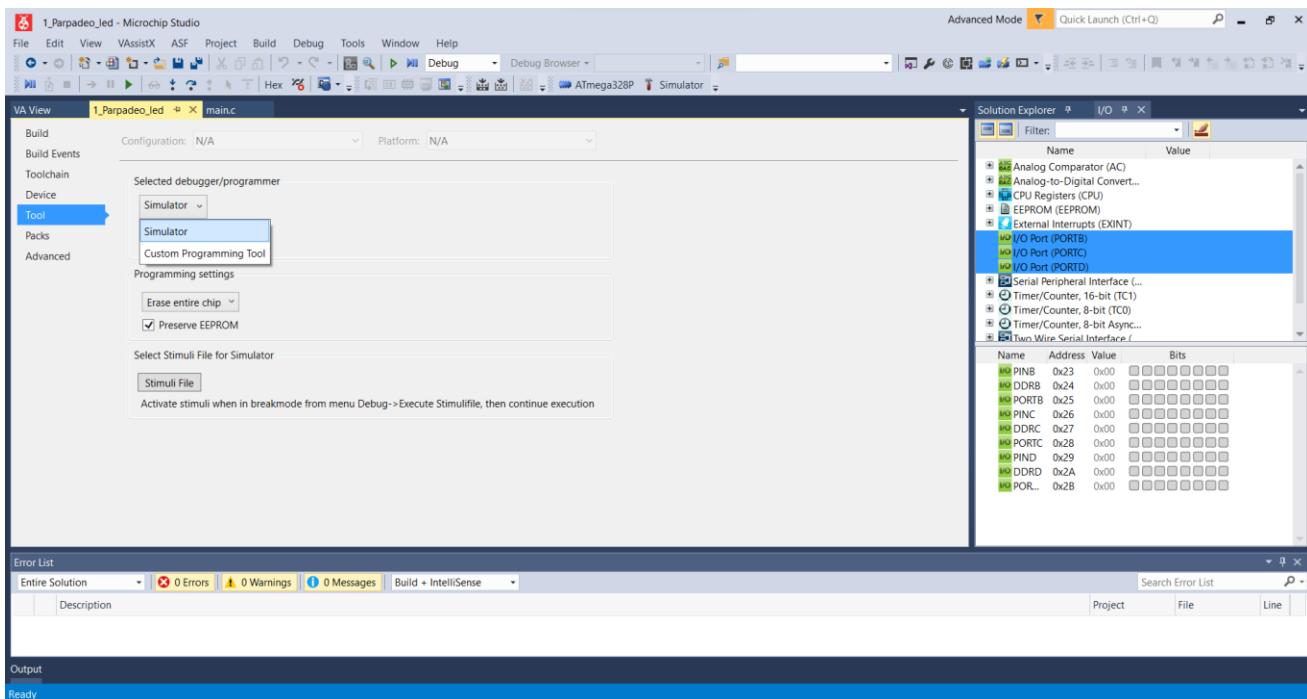
Este puede ser acomodado alado para verlo mejor arrastrándolo y poniéndolo sobre el área de trabajo de la siguiente manera:



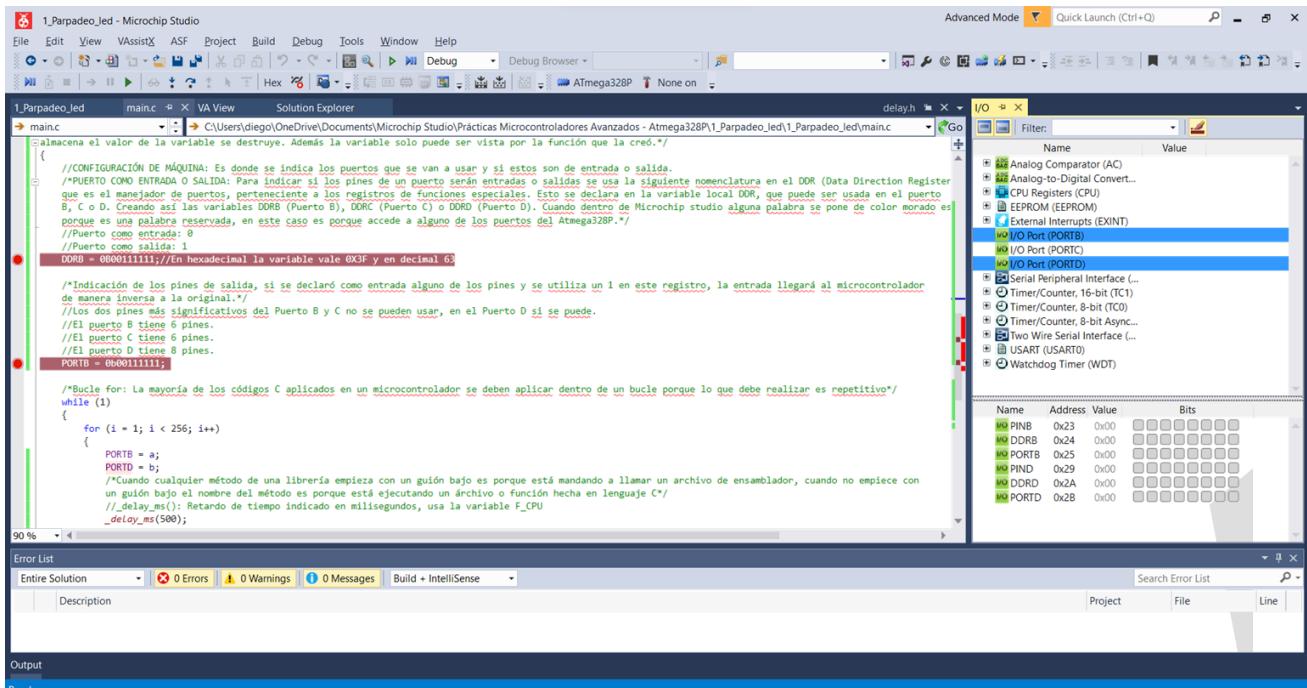
Antes de correr la simulación del programa se debe seleccionar la opción de Simulator y seleccionar el debugger Simulator.



También es importante mencionar que, si en algún momento nos salimos de la pestaña del código C, podemos regresar a ella a través del Solution Explorer, el cual es una pestaña que se encuentra del lado derecho del área de trabajo, buscando el archivo que diga **main.c** y dando clic sobre él.



Con la pestaña perteneciente a la herramienta de **I/O** se pueden ver los Puertos y registros que se están utilizando en el programa, por ejemplo, el Puerto B, C y D, para seleccionarlos se debe dar clic en cada uno y presionar la tecla CTRL si quiero seleccionar más de un registro para que se vea su interacción con el programa en la parte de debajo de la parte derecha del área de trabajo:



Cuando se dé clic ahora en el botón de **Start Debugging and Break** que se encuentra en la parte superior izquierda del área de trabajo, se ejecutará el programa y de los puertos y registros

seleccionados en la pestaña de **I/O** se verán datos en la parte de abajo, además de pausar su ejecución en los puntos donde se haya colocado **Breaking Points** dentro del código:

The screenshot shows two instances of Microchip Studio. The top instance has the 'I/O' tab selected in the Solution Explorer, displaying a table of port pins (PINB, DDRB, PORTB, PINC, DDRC, PORTC, PIND, DDRD, POR...) with their current values (0x00). The bottom instance also has the 'I/O' tab selected, showing the same table. Below the I/O tabs, there are memory dump windows for 'Memory 3' and 'Memory 4'. The 'Memory 3' window shows memory starting at address 0xFD6C with data values like 0x94, 0x34, 0x00, etc. The 'Memory 4' window shows memory starting at address 0x0000 with data values like 0x00, 0x00, 0x00, etc. Both memory windows have a 'prog FLASH' memory type.

```

main.c
main() {
    // Variables locales: el scope de este tipo de variables solo existe mientras se este ejecutando la función, cuando se deja de utilizar, el espacio de memoria donde se almacena el valor de la variable se destruye. Además la variable solo puede ser vista por la función que la creó.*/
    {
        //CONFIGURACIÓN DE MÁQUINA: Es donde se indica los puertos que se van a usar y si estos son de entrada o salida.
        /*PUERTO COMO ENTRADA O SALIDA: Para indicar si los pinos de un puerto serán entradas o salidas se usa la siguiente nomenclatura en el DDR (Data Direction Register), que es el manejador de puertos, perteneciente a los registros de funciones especiales. Esto se declara en la variable local DDR, que puede ser usada en el puerto B, C o D. Creando así las variables DDRB (Puerto B), DDRC (Puerto C) o DDRD (Puerto D). Cuando dentro de Microchip studio alguna palabra se pone de color morado es porque es una palabra reservada, en este caso es porque accede a alguno de los puertos del Atmega328P.*/
        //Puerto como entrada: 0
        //Puerto como salida: 1
        DDRB = 0x00000000; //En hexadecimal la variable vale 0XF y en decimal 63

        /*Indicación de los pinos de salida, si se declaró como entrada alguno de los pinos y se utiliza un 1 en este registro, la entrada llegará al microcontrolador de manera inversa a la original, si se pone un 0 llegará de forma normal.*/
        //Los dos pinos más significativos del Puerto B y C no se pueden usar, en el Puerto D si se puede.
        //El puerto B tiene 6 pinos.
        //El puerto C tiene 6 pinos.
        //El puerto D tiene 8 pinos.
        PORTB = 0x00000000;

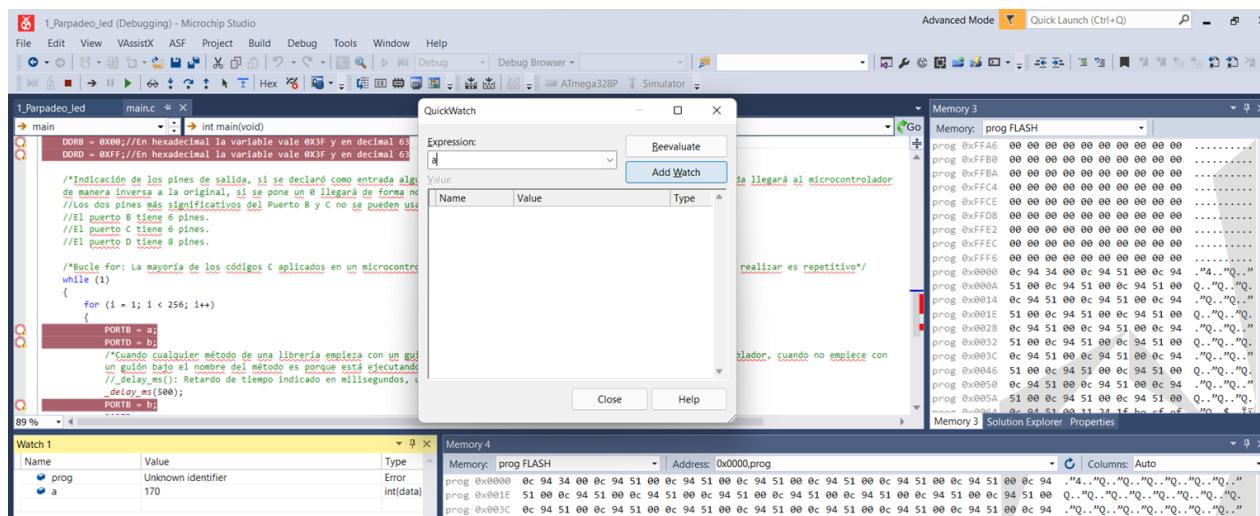
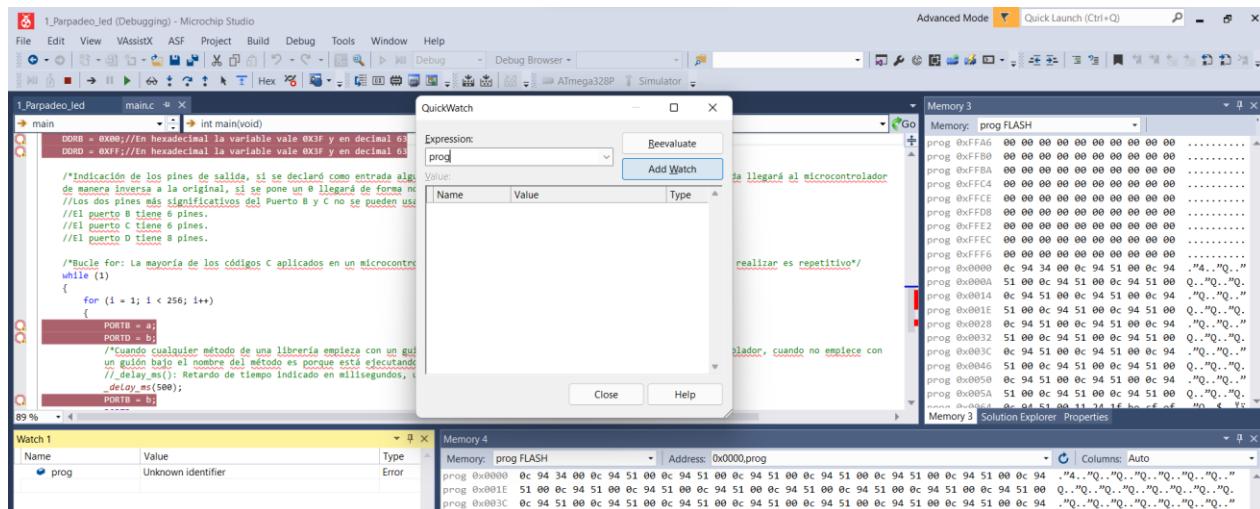
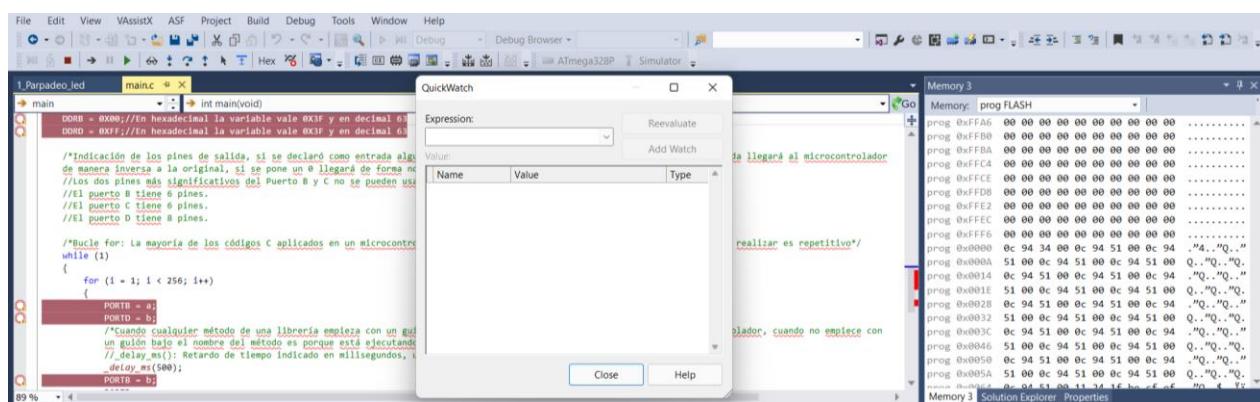
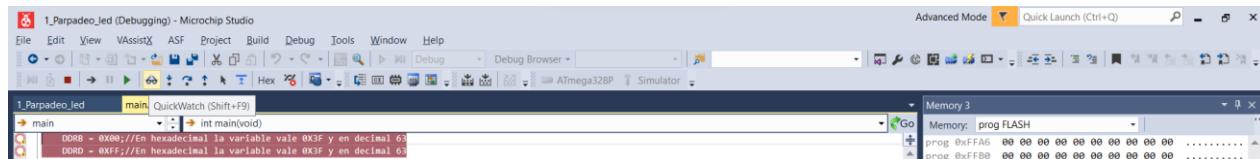
        /*Bucle for: La mayoría de los códigos C aplicados en un microcontrolador se deben aplicar dentro de un bucle porque lo que debe realizar es repetitivo*/
        while (1)
        {
            for (i = 1; i < 256; i++)
            {
                PORTB = a;
                PORTD = b;
                /*Cuando cualquier método de una librería emplea con un guion bajo es porque está mandando a llamar un archivo de ensamblador, cuando no emplee con un guion bajo el nombre del método es porque está ejecutando un archivo o función hecha en lenguaje C*/
                //delay_ms(): Retardo de tiempo indicado en milisegundos, usa la variable F_CPU
                delay_ms(500);
            }
        }
    }
}

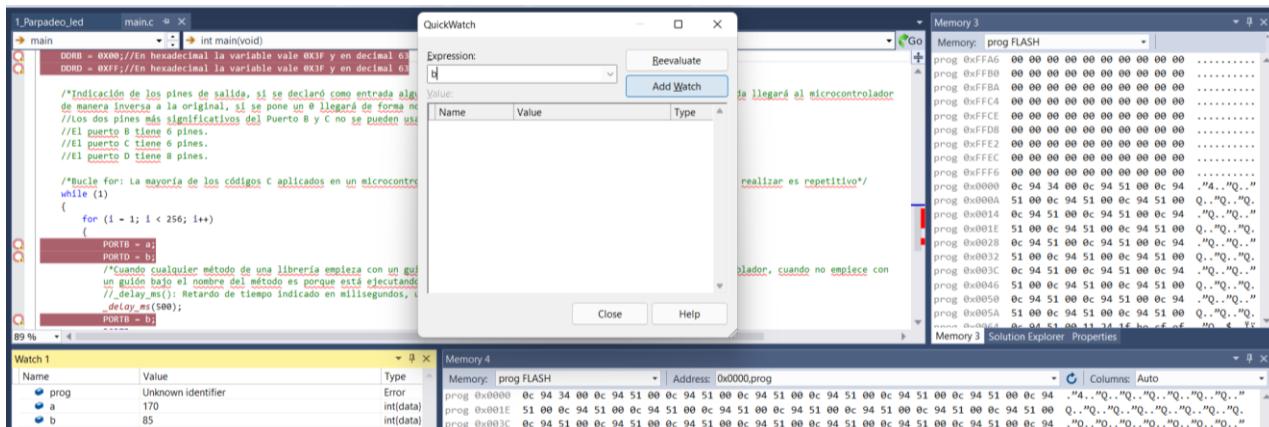
//Bucle for: La mayoría de los códigos C aplicados en un microcontrolador se deben aplicar dentro de un bucle porque lo que debe realizar es repetitivo*/
while (1)
{
    for (i = 1; i < 256; i++)
    {
        PORTB = a;
        PORTD = b;
        /*Cuando cualquier método de una librería emplea con un guion bajo es porque está mandando a llamar un archivo de ensamblador, cuando no emplee con un guion bajo el nombre del método es porque está ejecutando un archivo o función hecha en lenguaje C*/
        //delay_ms(): Retardo de tiempo indicado en milisegundos, usa la variable F_CPU
        delay_ms(500);
    }
}

```

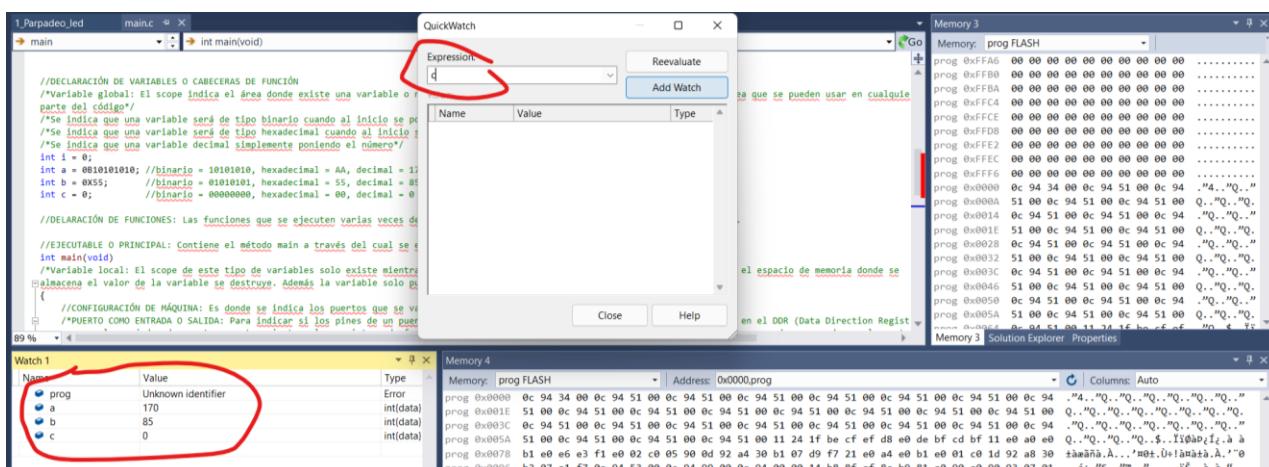
QuickWatch: Lo que hace esta herramienta es abrir una ventana donde en la parte que dice Expression se debe poner el nombre de la variable que se quiere observar y después dar clic en el botón de Add Watch, al hacerlo en la parte inferior izquierda aparecerá una ventana con el nombre de **Watch**, lo que

hace esto es observar y mostrar el valor de una variable en específico, para ir viendo cómo se comporta en el transcurso de la ejecución del programa.

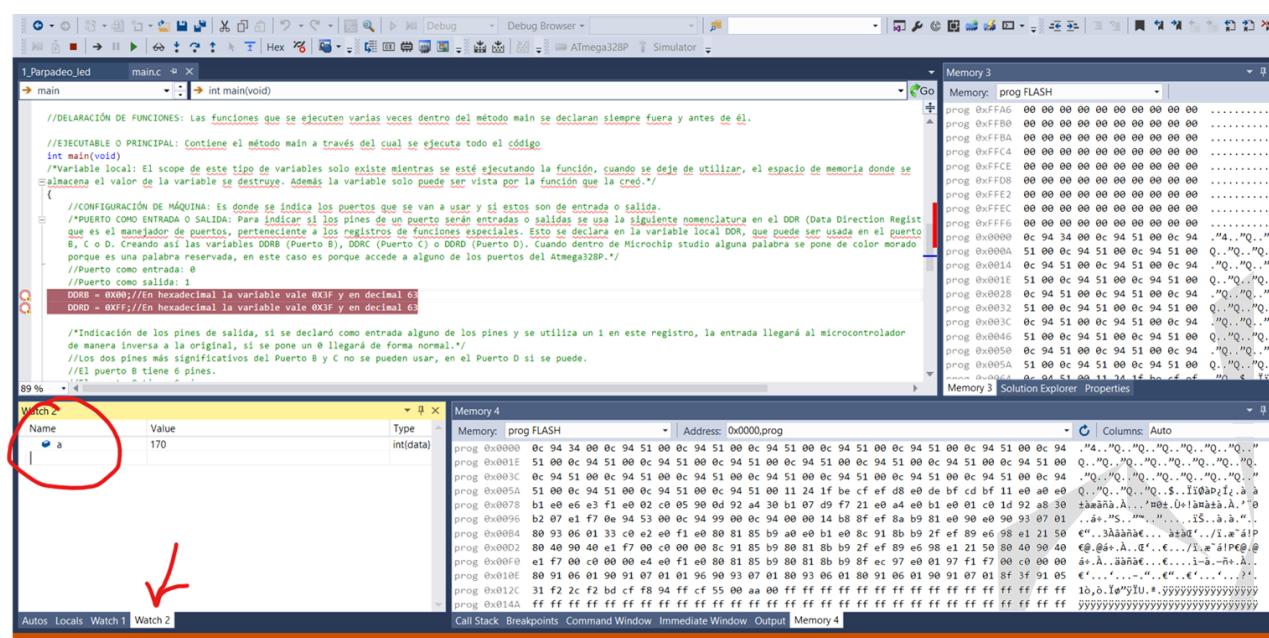




Todos los **Watch** que se agreguen aparecerán en la parte inferior derecha del área de trabajo.

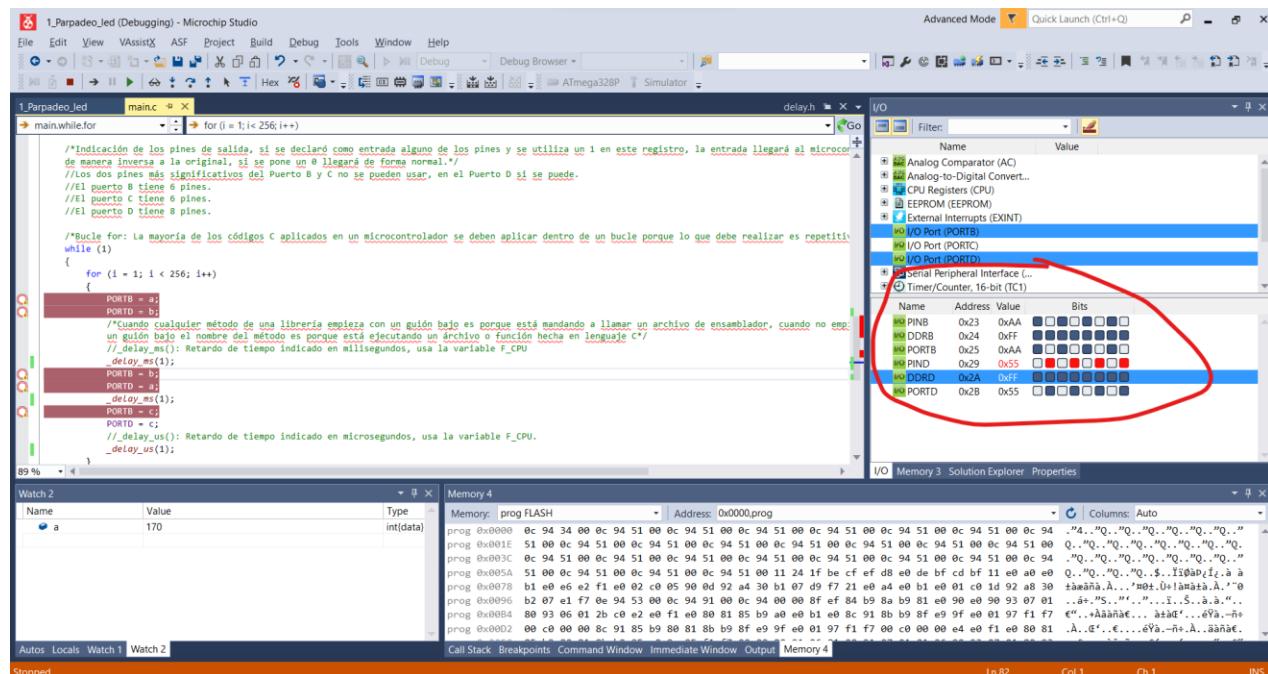


Además, se pueden agregar manualmente en la parte de abajo las variables de los **Watch** que se quiere ver, existen los **Watch 1, 2, ...**, en cada uno se pueden ver distintas variables o las mismas.



Simulación de un Programa en Microchip Studio

Cuando se quiera simular el programa se debe poner un número 1 en el delay, porque tarda mucho en simular esto Microchip Studio, en este caso es importante saber que se muestran los 3 registros de función específica de cada puerto, ya sea el **DDR** para indicar si es **entrada con un 1**, una **salida con un 0**, el registro de Puerto y el de Pin de todos los puertos seleccionados en la pestaña de **I/O**.

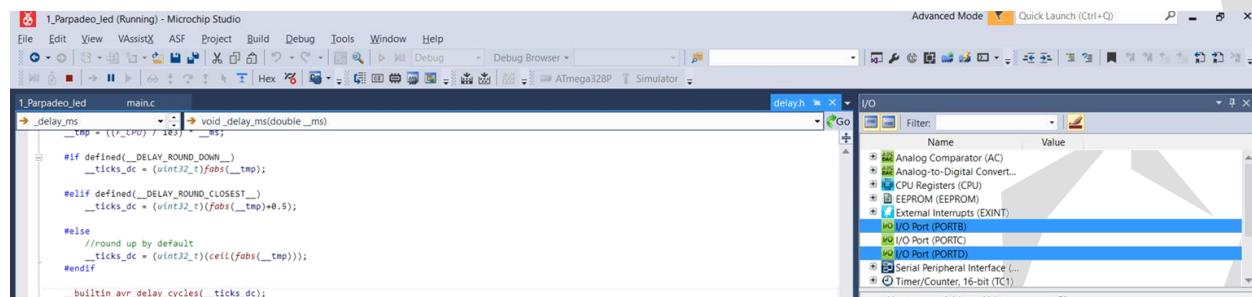


En el menú superior se cuenta con 3 opciones de simulación que realizan lo siguiente:

- Step Into:** Ejecuta solo las líneas de código donde haya breaking points, saltándose lo demás, empezando desde el contenido de la función main y saltándose la ejecución de librerías.
- Step Over:** Ejecuta línea por línea el código, empezando desde el contenido de la función main.
- Step Out:** Lo que hace esto es ejecutar todas las líneas de código hasta que se encuentre con un breaking point (el puntito rojo de la izquierda del código).

Cuando los registros de **DDR (Data Direction Register)** que indican si un puerto es entrada o salida sean de salida, los bits del registro PIN serán igual que los del registro PORT, sino serán distintos. Además, se pueden ejecutar intercalados los comandos anteriores para ejecutar la simulación. Se recomienda ejecutar el comando **Step Over** cuando el programa brinque a librerías.

Nota: Al llegar a la función delay, el programa se detendrá un momento en lo que la ejecuta, por eso se le debe poner un 1.



The screenshot shows the VASSISTX IDE interface. In the center, there is a code editor with C code for a microcontroller. On the right, there is a 'Watch' window titled 'I/O' which displays the state of various pins. A red circle highlights the 'PORTB' row in the watch window, specifically the columns for PINB, DDRB, PORTB, and PIND.

```

//Indicación de los pines de salida, si se declaró como entrada alguno de los pines y se utiliza un 1 en este registro, la entrada llegará al microcontrolador de manera inversa a la original, si se pone un 0 llegaría de forma normal.
//Los dos pines más significativos del Puerto B y C no se pueden usar, en el Puerto D si se puede.
//El puerto C tiene 8 pines.
//El puerto D tiene 8 pines.
//El puerto D tiene 8 pines.

//Bucle for: La mayoría de los códigos C aplicados en un microcontrolador se deben aplicar dentro de un bucle porque lo que debe realizar es repetitivo.
while (1)
{
    for (i = 1; i < 256; i++)
    {
        PORTB = a;
        PORTD = b;

        //Cuando cualquier método de una librería empieza con un guión bajo es porque está mandando a llamar un archivo de ensamblador, cuando no empieza con un guión bajo el nombre del método es porque está ejecutando un archivo o función hecha en lenguaje C
        //delay_ms(); Retardo de tiempo indicado en milisegundos, usa la variable _F_CPU
        _delay_ms(1);
        PORTB = b;
        PORTD = a;
        delay_ms(1);
        PORTB = c;
    }
}

```

Además, en el **Watch** se puede declarar el valor de alguna variable de manera manual durante la simulación para ver como interactúa con el programa al cambiar de valor.

This screenshot shows the VASSISTX IDE with the 'Watch' window open. A red circle highlights the 'Value' column for the variable 'val', which is currently set to 37. The code in the editor shows declarations for ports B, C, and D, and a main loop with a call to the 'delay_ms' function.

```

//DECLARACIÓN DE CONSTANTES USANDO #define: Hay que tener en cuenta que al declarar constantes con #define debemos hacerlo antes de los #include (que importan librerías), pero antes de declarar nuestras funciones y demás. No se usa el signo de igual para asignar un valor, solo se declara el nombre y luego el valor.
#define F_CPU 16000000UL //frecuencia del reloj, es necesario declararla porque la librería delay de manejo de tiempos la utiliza

//IMPORTACIÓN DE LIBRERÍAS, DIRECTIVAS
#include <avr/io.h> //Librería para todo el manejo de puertos y demás aspectos del microcontrolador AVR Atmega328P
#include util/delay.h //Librería de manejo de tiempos, como retardos principalmente

//DECLARACIÓN DE VARIABLES GLOBALES (CABECERAS DE FUNCIÓN)
int i = 0; //Variable que almacenará el contenido proveniente del PINB
int val = 0; //Variable que almacenará el contenido proveniente del PINB
//int i = 0, val = 0; Se puede declarar también de esta manera

//MÉTODO MAIN O EXECUTABLE: Desde este método se ejecuta todo el código y es lo que diferencia la librería de un ejecutable.
int main(void)
{
    //PROGRAMACIÓN DE LA MÁQUINA: En este se indica por medio de los registros de función específica, los pines de los puertos B, C y D que serán entradas o salidas.
    //0: Pin como entrada
    //1: Pin como salida
    DDRA = 0x00; //Todos los pines del puerto B son entradas
    DDRB = 0x00; //Todos los pines del puerto B son salidas

    //Indicación de los pines de salida, si se declaró como entrada alguno de los pines y se utiliza un 1 en este registro, la entrada llegará al microcontrolador de manera inversa a la original (resistencias pull-up), si se pone un 0 llegaría de forma normal(resistencias pull-down). Funciona más o menos como una compuerta NOT.
    //Los dos pines más significativos del Puerto B y C no se pueden usar, en el Puerto D si se puede.
    //El puerto C tiene 8 pines.
    //El puerto D tiene 8 pines.
    //El puerto D tiene 8 pines.

    PORTB = 0xFF; //La entrada llegará al microcontrolador de manera inversa a la original (Resistencias pull-up)

    //Los registros DDx indican si el puerto es entrada o salida, los registros PORT manejan las salidas y los registros PIN reciben las entradas de los puertos B, C y D.
    while (1)
    {
        val = PINB; //Los bits provenientes del puerto B se almacenan en la variable val
        /*Se recomienda en la bibliografía que cuando se hace una asignación al PIN, se le da un tiempo de 2 NOP, osea 2 microsegundos, para que el puerto se regule.*/
        //Cuando se usen delays en las simulaciones, se debe poner un 1 dentro del paréntesis sino se queda mucho tiempo trabado el simulador.
        //delay_us(); //delay_us(): Método de la librería delay, creando un retardo de tiempo en microsegundos de lo que se ponga en su paréntesis
        PORTD = val; //Los bits guardados en la variable val salen por el puerto D
        _delay_ms(); //delay_ms(): Método de la librería delay, creando un retardo de tiempo en milisegundos de lo que se ponga en su paréntesis
    }
}

```

También se puede cambiar el valor de los bits en un puerto si se da clic en cada uno de ellos dentro de la ventana I/O.

This screenshot shows the Microchip Studio IDE. A red circle highlights the 'Value' column for the variable 'val', which is currently set to 37. The code in the editor shows declarations for ports B, C, and D, and a main loop with a call to the 'delay_ms' function. The 'Watch' window is also visible at the bottom left.

```

//1.Parpadeo_led_version2 (Debugging) - Microchip Studio
File Edit View VAssistX ASF Project Build Debug Tools Window Help
delay.h Makefile main.c
mainwhile.c
while(1)
{
    //PROGRAMACIÓN DE LA MÁQUINA: En este se indica por medio de los registros de función específica, los pines de los puertos B, C y D que serán entradas o salidas.
    //0: Pin como entrada
    //1: Pin como salida
    DDRA = 0x00; //Todos los pines del puerto B son entradas
    DDRB = 0x00; //Todos los pines del puerto B son salidas

    //Indicación de los pines de salida, si se declaró como entrada alguno de los pines y se utiliza un 1 en este registro, la entrada llegará al microcontrolador de manera inversa a la original (resistencias pull-up), si se pone un 0 llegaría de forma normal(resistencias pull-down). Funciona más o menos como una compuerta NOT.
    //Los dos pines más significativos del Puerto B y C no se pueden usar, en el Puerto D si se puede.
    //El puerto C tiene 8 pines.
    //El puerto D tiene 8 pines.
    //El puerto D tiene 8 pines.

    PORTB = 0xFF; //La entrada llegará al microcontrolador de manera inversa a la original (Resistencias pull-up)

    //Los registros DDx indican si el puerto es entrada o salida, los registros PORT manejan las salidas y los registros PIN reciben las entradas de los puertos B, C y D.
    while (1)
    {
        val = PINB; //Los bits provenientes del puerto B se almacenan en la variable val
        /*Se recomienda en la bibliografía que cuando se hace una asignación al PIN, se le da un tiempo de 2 NOP, osea 2 microsegundos, para que el puerto se regule.*/
        //Cuando se usen delays en las simulaciones, se debe poner un 1 dentro del paréntesis sino se queda mucho tiempo trabado el simulador.
        //delay_us(); //delay_us(): Método de la librería delay, creando un retardo de tiempo en microsegundos de lo que se ponga en su paréntesis
        PORTD = val; //Los bits guardados en la variable val salen por el puerto D
        _delay_ms(); //delay_ms(): Método de la librería delay, creando un retardo de tiempo en milisegundos de lo que se ponga en su paréntesis
    }
}

```

Puertos B, C y D Físicos en el Arduino

PORTB*

11111111



PORTC*

11111111

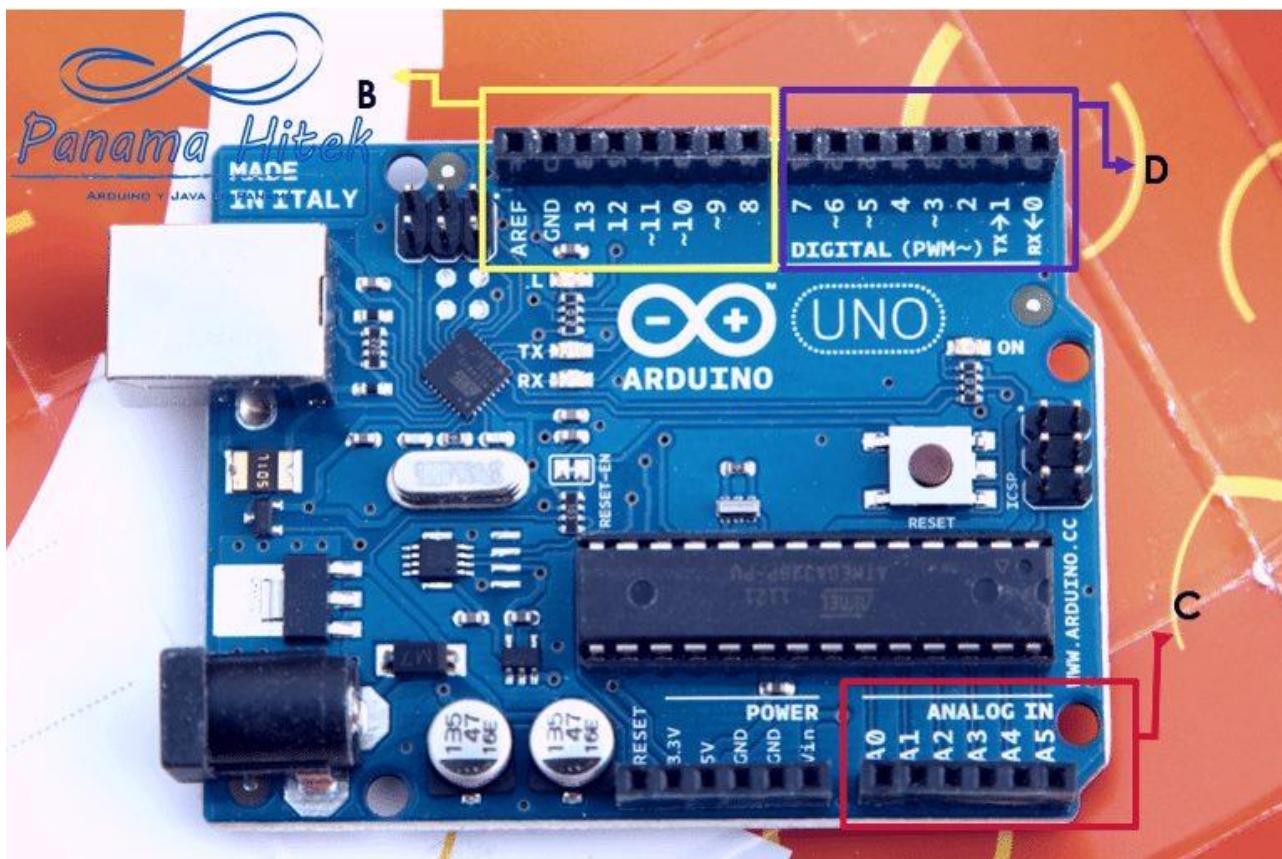


PORTD

11111111

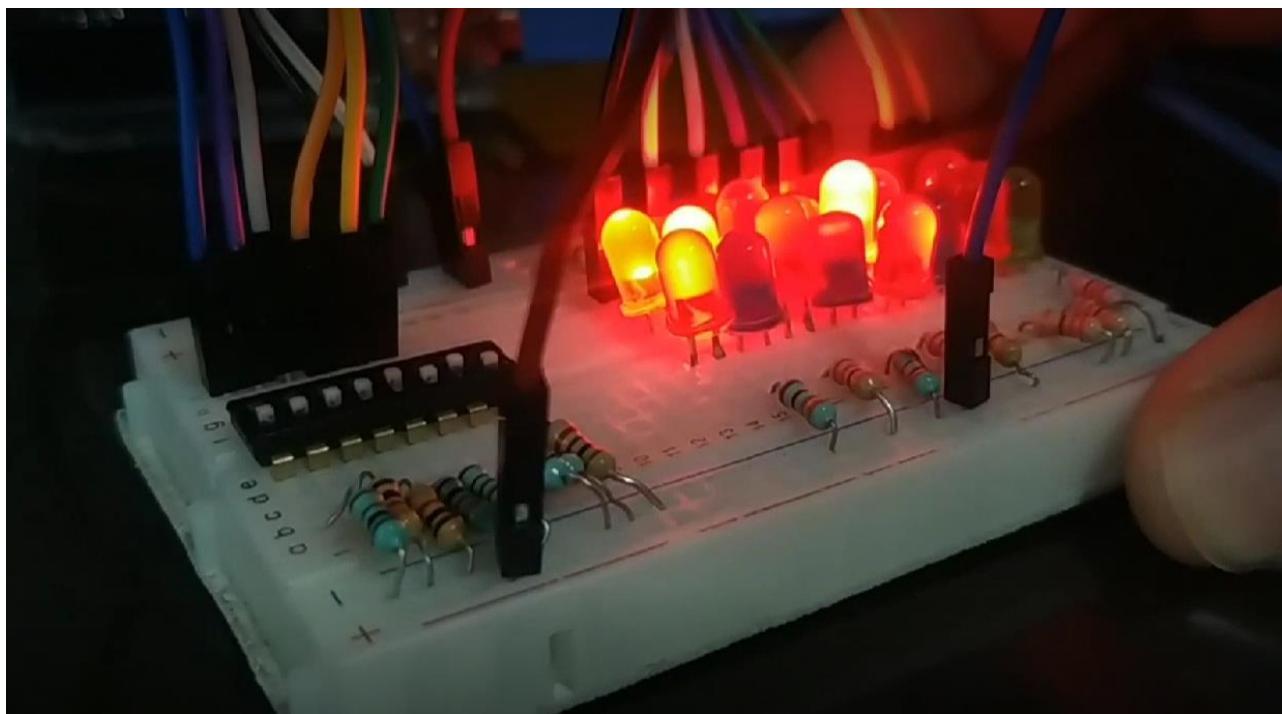
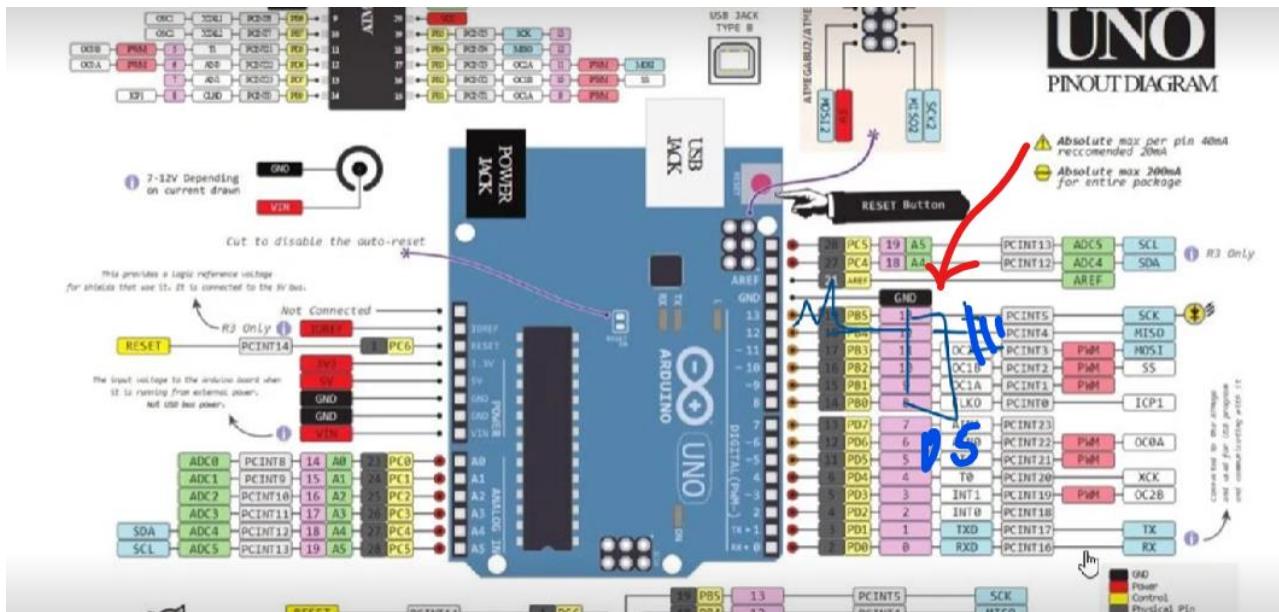


Los pines de mayor peso de los puertos **B** y **C** no se usan, por lo cual en los puertos **B** y **C** existen **6 pines** y en el puerto **D** existen **8 pines**, los pines de cada una de las 3 familias de puertos de la placa Arduino UNO se muestran en la siguiente imagen.



Conexión Física para Insertar Datos a los Pines del Arduino UNO

La manera de insertar datos al Arduino UNO se realiza a través de la conexión de un Dip Switch conectado con a una resistencia de 100Ω que a su vez se conecte directo al puerto y el otro extremo se conecte a tierra como se puede ver en la siguiente imagen.

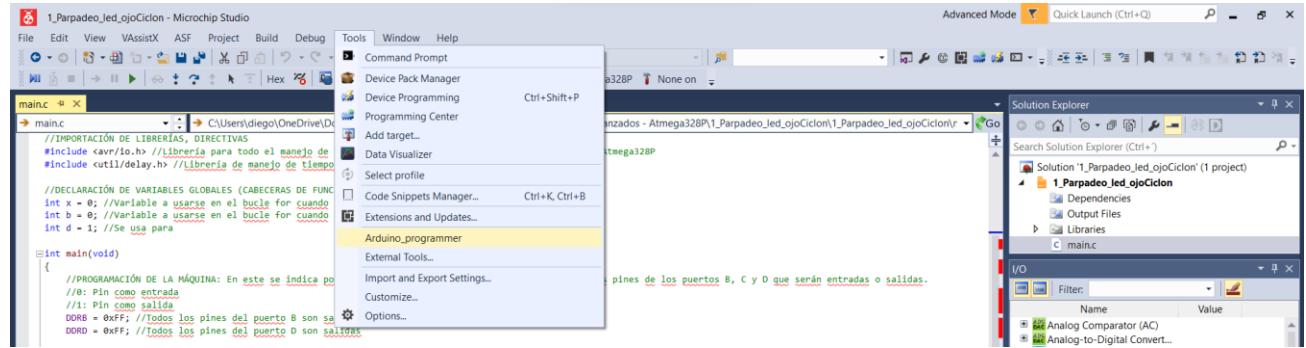


Programar Físicamente la Placa Arduino UNO con el Programa de Microchip Studio

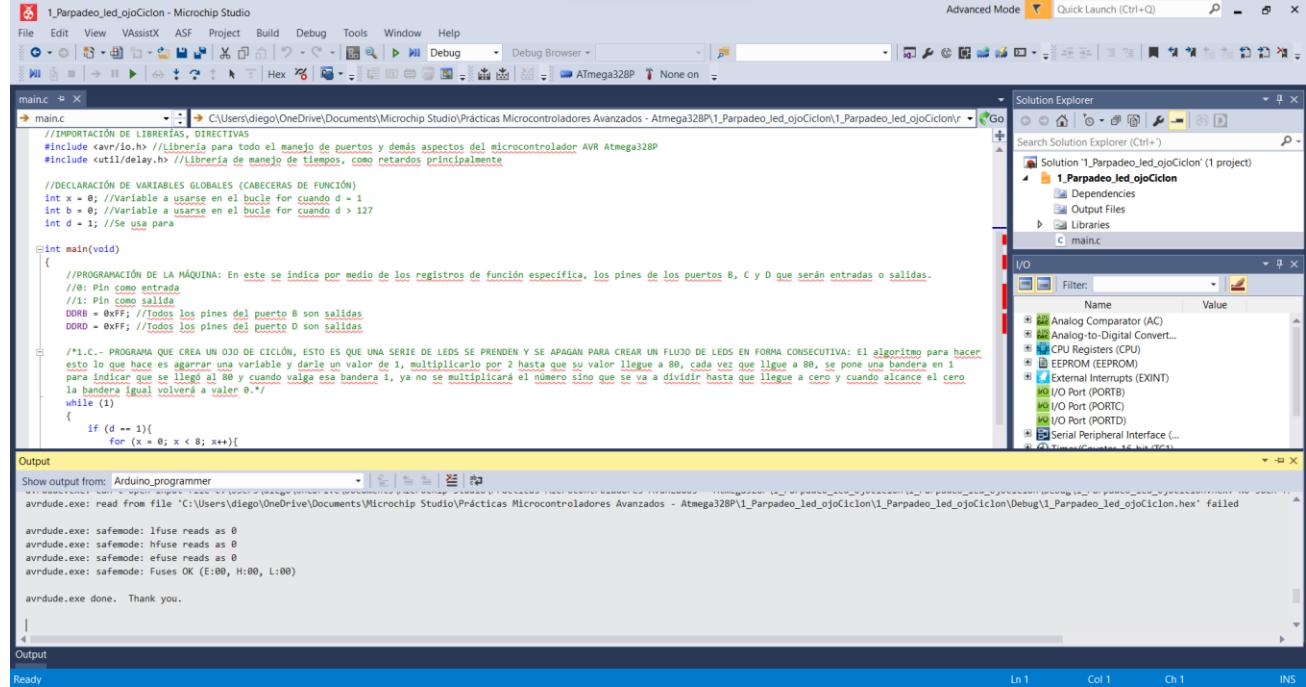
Recordemos que la forma de programar la placa del Arduino UNO, ya habiendo realizado la configuración de su programador en el programa de Microchip Studio es seleccionando la opción de:

Tools → Arduino programmer (opción creada y nombrada cuando se configuró el Arduino).

Es muy importante mencionar que cuando se carga un código a una placa Arduino, debemos tener cuidado que no haya nada conectado a los pines **TX** y **RX** del **Puerto D** porque si no Microchip Studio indicará un error cuando tratemos de subir el programa, ya habiéndolo subido podemos utilizar esos pines de nuevo y no habrá problema.



The screenshot shows the Microchip Studio interface with the 'Arduino_programmer' option highlighted in the Tools menu. The code editor displays a C program named 'main.c' for an Atmega328P microcontroller. The code includes comments explaining pin usage: pins B, C, and D are used as inputs or outputs, while pins 0 and 1 are used as inputs. The I/O configuration table in the bottom right shows various peripherals like Analog Comparator (AC), CPU Registers (CPU), and External Interrupts (EXINT) listed under the I/O tab.



The screenshot shows the Microchip Studio interface with the AVRdude command displayed in the Output window. The command reads: 'avrdude.exe: read from file 'C:\Users\diego\OneDrive\Documents\Microchip Studio\Prácticas Microcontroladores Avanzados - Atmega328P\1_Parpadeo_led_ojoCiclon\1_Parpadeo_led_ojoCiclon\Debug\1_Parpadeo_led_ojoCiclon.hex' failed'. This indicates that the fuse settings were incorrect, leading to a failure during the programming process.

LCD (Liquid Crystal Display)

Cada uno de los puntos que se ven en el LCD se llaman DOTS, cada DOT permite ver luz o no para mostrar un punto del carácter porque dentro tiene un resorte que al ser electrificado se contrae, haciendo que no pueda pasar luz a través de él y se pueda ver el punto del carácter.



Para entender el funcionamiento y pasos para programarlo, debemos leer el archivo:

2.- LCD Alfanumérico

Máscara AND

- **Máscara AND:** Es una máscara que al aplicarse a un número binario cualquiera:
 - **Donde tenga 1:** Deja pasar el número tal cual como viene originalmente.
 - **Donde tenga 0:** Convierte el número original a 0.

Máscara OR y XOR

- **Máscara OR:** Es una máscara que al aplicarse a un número binario cualquiera:
 - **Donde tenga 1:** Convierte el número original a 1.
 - **Donde tenga 0:** Deja pasar el número tal cual como viene originalmente.
- **Máscara XOR:** Es una máscara que al aplicarse a un número binario cualquiera:
 - **Donde tenga 1:** Convierte el número original a su inverso, si era 0 lo hace 1 y si era 1 lo hace 0.
 - **Donde tenga 0:** Deja pasar el número tal cual como viene originalmente.