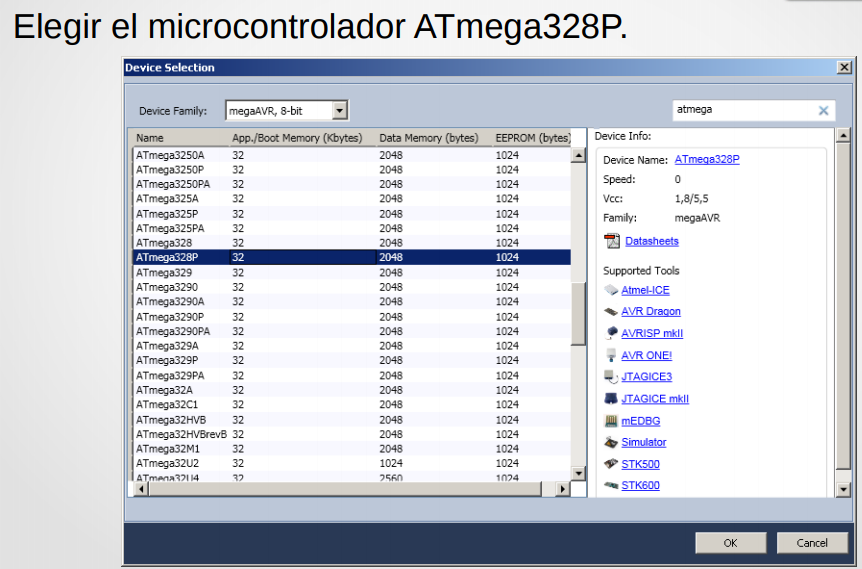
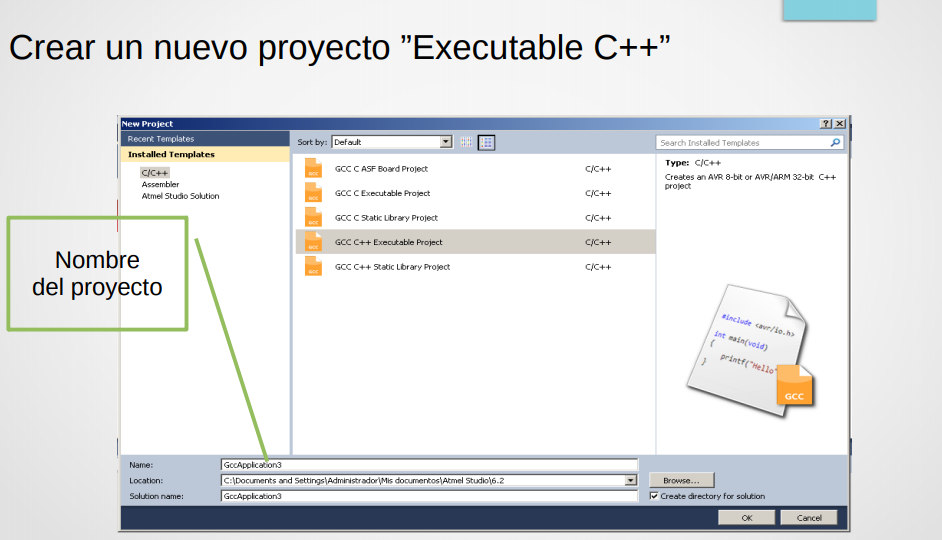
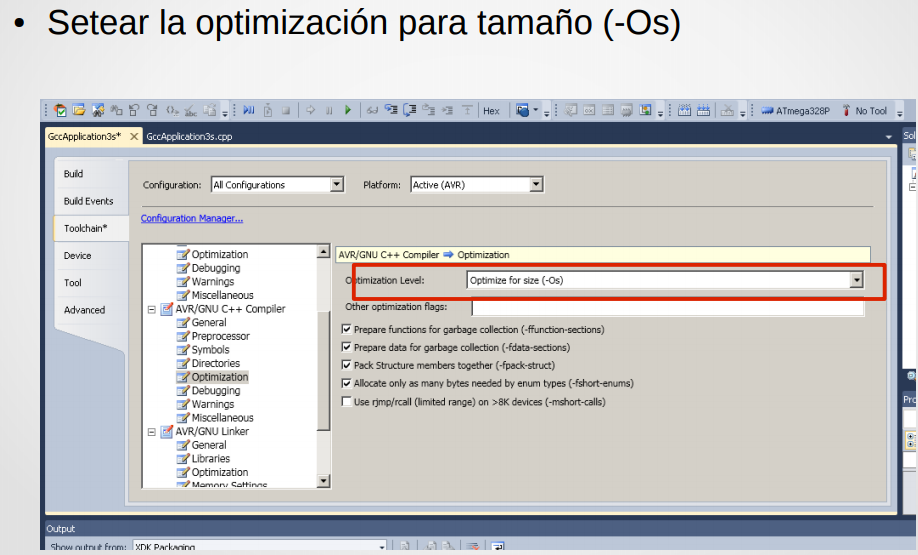
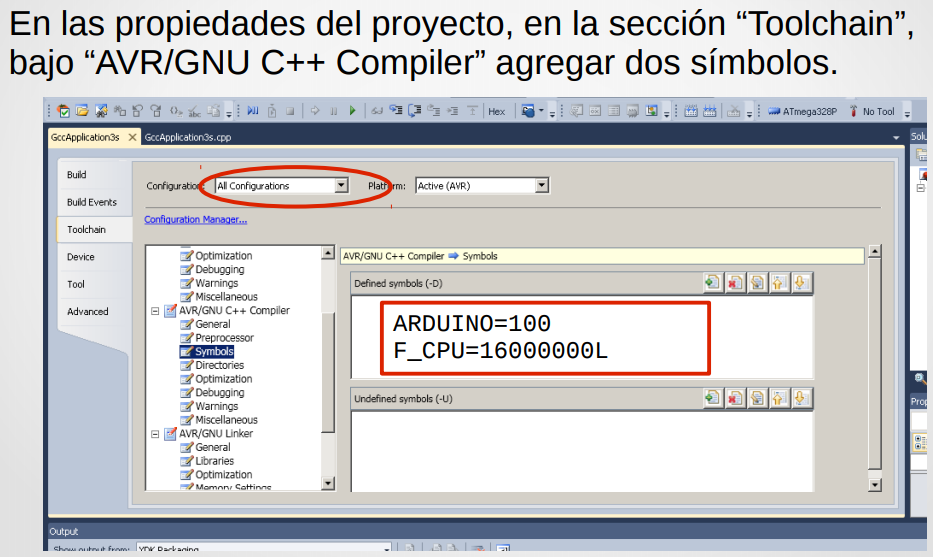
Laboratorio n°1

**Actividad 1: Polling y copia de un puerto a otro**

Instale Atmel Studio. Luego, seguí las instrucciones para crear un nuevo proyecto en Atmel Studio:



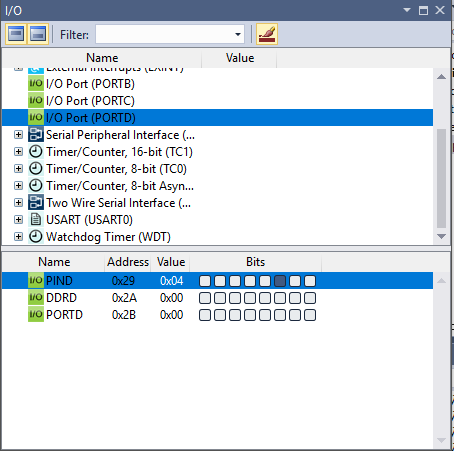
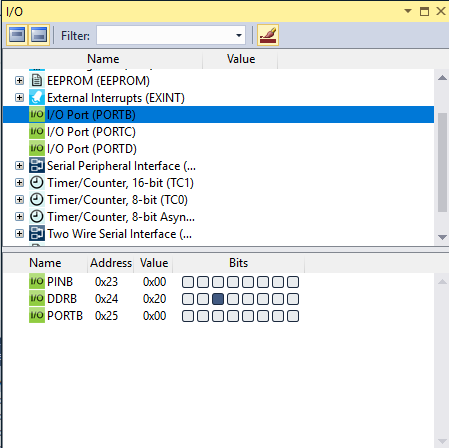


Construir la imagen ejecutable es básicamente compilar y ejecutar.

Sucede que por ahora no grabamos en la memoria flash del arduino el código en HEX, sino que tenemos que simular el funcionamiento del código por software. Para esto:

Tocar Depurar -> Start Debugging and break

Luego toco la opción que dice I/O, me abrirá una ventana así:



Según lo visto en el código, si aprieto y marco en el puerto D, el pin D en el casillero que indica el código (en este caso el bit 2), entonces le estoy suministrando un 1, por lo tanto, sucederá que, por el if, se pasará a prender el led, luego si hago click de nuevo y bajo el 1 a un 0, se apagará el led por el if. Puedo prender y apagar el led sino directamente del puerto B.

Cable aclarar lo siguiente, él PIND nos sirve para poder leer a partir del puerto, tiene 8 entradas, que se corresponde con PD0, PD1, PD2, etc.

El puerto PORTD sirve para escribir, por lo tanto, si queremos escribir algo lo hacemos en uno de los 8 puertos y DDRD sirve para poder marcar el puerto como entrada/salida.

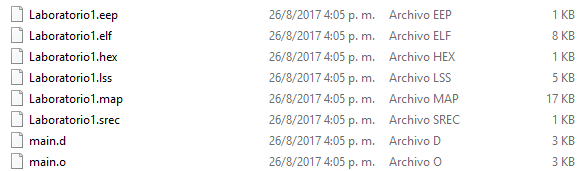
**La duda está en donde ubico PIND, PORTD y DDRD.**

Punto 4:

Con respecto a la compilación, el ensamblado, etc:

Para la construcción del ejecutable, en general, se genera código de otra arquitectura. Una vez se obtuvo la imagen ejecutable, a partir de ella se puede simular en el host, o descargar en el target. El juntar esto y desarrollarlo a la vez se denomina desarrollo cruzado.

Cuando tocamos compilar se generan todos los siguientes archivos:



El archivo .elf y .hex representa la imagen ejecutable

El archivo .map representa un archivo de link map

El archivo .o representa un archivo de objeto

Para la construcción:

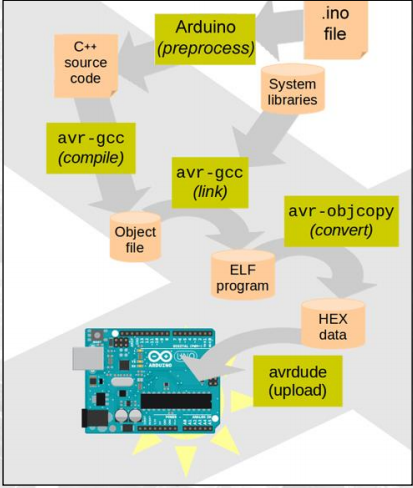
Comienza con el ensamblado y la compilación cruzada, donde los ensambladores y los compiladores traducen el código fuente a código objeto para la arquitectura.

Luego el linker combina el código objeto en referencias externas. Resuelve las ref. simbólicas definiendo locaciones para los símbolos (código absoluto, reubicable, independiente de la posición, ...). Trabaja en conjunto con el locador. Osea que enlaza con posiciones en memoria. Luego el locador inserta dicho código en la memoria

Luego, se hace la locación en memoria, refleja el mapa de memoria del sistema. El locador tiene que ser capaz de generar código que se aloje en cierto espacio de direcciones (ej. ROM), pero que ejecute correctamente en otro (ej. RAM).

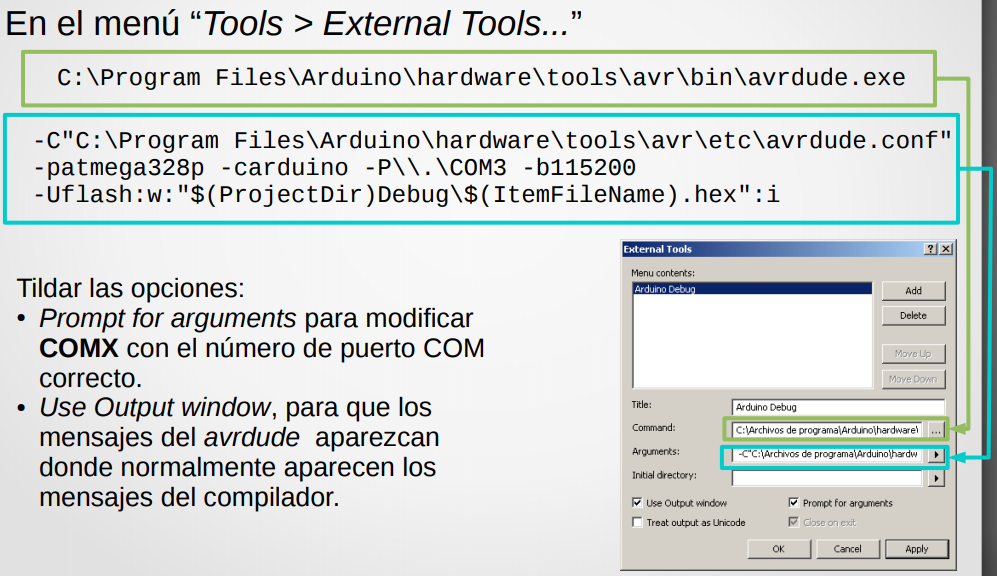
Así se obtiene una imagen ejecutable del sistema en el host.

Por ejemplo en Arduino se hacen los siguientes pasos:  
Realiza un preprocesamiento que genera código C/C++, luego provee el código generado al toolchain de AVR (Atmel).

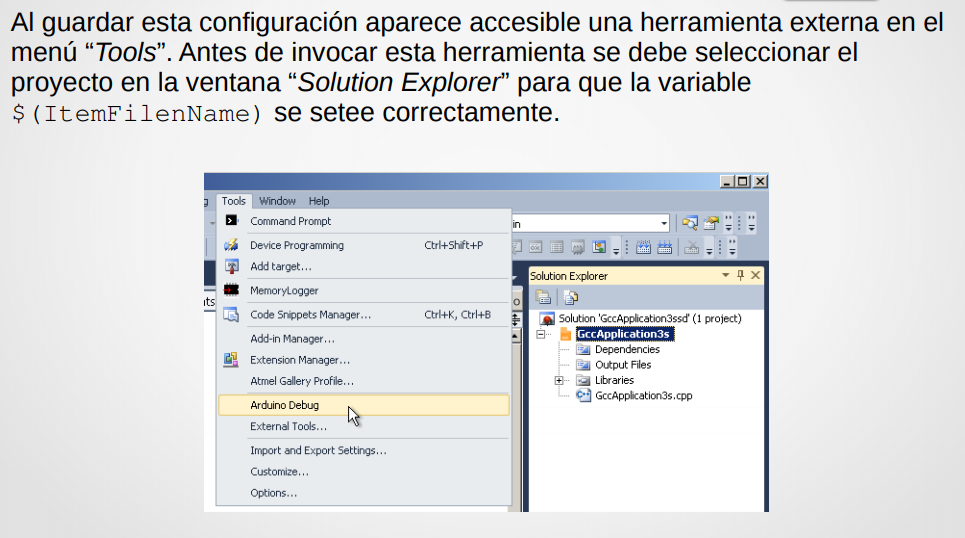


Punto 7

Para descargar la imagen ejecutable que generamos al microcontrolador, hacemos los siguientes pasos:



Agrego la herramienta arduino debug, con la cual podremos grabar el programa en el arduino, en la memoria flash. Para averiguar el puerto COM correspondiente por el cual se transferirá la información, tendremos que mirar el administrador de dispositivos. Todo esto obviamente teniendo conectado por USB el microcontrolador arduino.



Para este ejercicio luego vemos que a medida que presionamos el botón, se prende el led, luego si lo soltamos, se apaga.

Actividad 2: Debouncing

Punto 1: El código de flanco, lo que realiza es que cuando se aprieta el pulsador, y luego se suelta, genera un flanco ascendente, que se mantiene, hasta volver a apretar el botón para generar un flanco descendente. Por eso, este código, guarda el estado anterior almacenado en el sistema.

Sucede que, al probarlo, sufre de rebote, debido al ruido electromagnético presente que modifica la señal. Esto genera que a veces cuando presionemos el pulsador, no se prenda o apague el led respetando el orden correspondiente. Además, si desconectamos la salida de uno de los cables de la protoboard, podemos ver como el led se prende y se apaga rápidamente, concluyendo que el pin está abierto y al estar abierto se ve afectado por todo el ruido electromagnético presente.

Esto se soluciona en el punto 2 añadiendo un capacitor, es la solución por hardware.

Punto 3: El debouncing es la solución por software al rebote. Se lee el estado del pulsador, luego se espera 50 milisegundos para poder esquivar este rebote de la señal y caer en la señal original y de ahí se lee de nuevo. Digamos, se aprieta el pulsador, o se suelta, se checkea que el estado anterior sea diferente al nuevo pasan 50 ms y se vuelve a repetir la secuencia anterior, y ahí está la diferencia.

Supongamos, que el estado anterior esta en 0, se cambia el estado actual a 1 (o sea técnicamente se apretó el botón) pasaron los 50ms y ahora esta suelto, no era una presión del humano, sino ruido (ya que tarda más tiempo el humano), por lo tanto, vuelve a estar en cero el estado actual, y no pasa el ultimo if, por lo tanto, la luz del led no entra a cambiar, y el estado anterior tampoco. Es análogo para cuando estaba prendido el led de antes.

Punto 4: Como ya antes he dicho, la diferencia radica en que, en button el led se mantiene prendido cuando se aprieta el botón y se mantiene apretado, en cambio en flanco, se aprieta una vez y se prende o se apaga el led.

A ambos los afecta el rebote del pulsador (ruido electromagnético), debido a que algunos pulsadores son viejos, aunque, pareciera que no afecta tanto el funcionamiento de “button” debido a que este se mantiene apretado, de todos modos se termina cambiando el estado, se toma el rebote y luego la señal original. **PREGUNTAR.**

Siempre que haya ruido es necesaria la técnica de debouncing.

Punto 5: Los casos ya fueron analizados anteriormente. Respecto a uno u otro, salvar este problema por software es útil cuando no se dispone de elementos de hardware, por ser muy caros o alguna otra razón. Sin embargo, a veces puede suponer una complicación implementar el debouncing a nivel de software, debido a que los componentes pueden ser muy sensibles, resultando en algún tipo de daño en el microcontrolador y dejándolo obsoleto, o el código presente es muy complicado.

**Seguir preguntando ventajas y desventajas**

Actividad 3: Temporizado y retardos por software

3) Determinar a qué pin del microcontrolador ATmega328P está asociado cada uno de los pines digitales 0 a 13 y los analógicos A0 a A5 de la placa Arduino.

Desde los pines digitales IO0 a IO7 están asociados al puerto D, donde corresponden a los pines del microcontrolador 2, 3, 4, 5, 6, 11, 12, 13. Desde los pines digitales IO8 a IO13 (desde IO10 cambian de nombre a SS, MOSI, MISO, SCK) están asociados al puerto B, donde se corresponden a los pines del microcontrolador 14, 15, 16, 17, 18, 19.

Luego, desde los analógicos A0 a A5, se mapean al puerto B, donde corresponden a los pines del microcontrolador 23, 24, 25, 26, 27 y 28.

6) Polling: el procesador consume ciclos de CPU esperando por enviar o recibir un dato del periférico, dicho periférico le avisa seteando un flag en 1, entonces ese tiempo hasta que el flag se setea, se mantiene consumiendo ciclos de CPU el procesador sin hacer nada, luego envía o recibe el dato del o hacia el periférico.

La solución a este consumo, es la interrupción. El procesador continua su ejecución normal, por más que sepa que esta por enviar o recibir un dato, pero cuando se setea el flag, hace una interrupción, y ahí envía el dato o recibe el dato.

El esquema de polling tiene la ventaja de que es muy simple de implementar

7) el retardo por software tiene el problema de que, cuando hay un delay, si se pulsa el botón no se detecta el cambio, cuando el delay, el CPU no hace nada, gira nomas pero no consume nada n ohace nada, por lo tanto, las instrucciones se pierden. La solución a esto es que el botón genere una interrupción, que le avise al micro que hubo una interrupción.

Una solución intermedia es mediante un timer, el cuál está dentro del microcontrolador, que lleva la cuenta.

El registro DDRX lleva cuenta de si es de entrada o salida, depende el bit que se marque

El de PINX depende el bit que se marque indica si esta en high o low el pin

El registro PORTX indica si esta conectada la resistencia de pull up o no.

Actividad 4:

Punto uno: Ambos ejemplos realizan lo mismo. Al apretar el botón pulsador, el led se prende, y al soltarlo el led se apaga. Uno de los otros es más alto nivel.

Sin embargo, los dos trabajan con distintas variables. Se utiliza pinMode para poder inicializar un pin como output o input, y ahí se declara si es el pin del led o del botón.

Luego, utiliza las funciones de más alto nivel digitalWrite y digitalRead, con el que puedo escribir y leer información desde los distintos pines.

// initialize the LED pin as an output:

pinMode(ledPin, OUTPUT);

// initialize the pushbutton pin as an input:

pinMode(buttonPin, INPUT);

}

void loop(){

// read the state of the pushbutton value:

buttonState = digitalRead(buttonPin);

// check if the pushbutton is pressed.

// if it is, the buttonState is HIGH:

if (buttonState == HIGH) {

// turn LED on:

digitalWrite(ledPin, HIGH);

}

else {

// turn LED off:

digitalWrite(ledPin, LOW);

}

}

2. Examinar el código fuente de la librería Arduino.

a) Arduino inicia la ejecución en la función main. ¿Dónde se encuentra declarada esta función en las librerías de Arduino? Analice el flujo de control de dicha función y determine qué tareas llevan a cabo cada una de las funciones invocadas desde ella.

Se encuentra en un archivo separado llamado main.cpp en el core de arduino. Esta función es con la que se inicia una vez se ejecuta el programa especificado. La idea de dicha función, es hacer que haya la menor cantidad posible de código en el IDE de arduino, de modo tal que se entienda más y sea más amigable.

Básicamente, esta función lo que hace es llamar a la función setup(), la cuál estará especificada en el código especificado por el programador en el IDE de arduino. Luego, llama infinitas veces a la función loop() también especificada en el IDE de arduino, de modo tal de seguir todas las sentencias declaradas ahí dentro.

b) Determinar dónde están definidas las constantes OUTPUT, INPUT, HIGH y LOW y qué valores representan.

High 0x1

Low 0x0

Input 0x0

Output 0x1

c) Determinar dónde y cómo se implementa la correspondencia entre la numeración de los pines de Arduino y los pines del ATmega328P.

La correspondencia de los pines de arduino y los pines del atmega se implementan en uno de los archivos de pins\_arduino.h en la carpeta variants de arduino.

**DETERMINAR DONDE**

3) Explique qué tareas realizan las funciones pinMode, digitalRead y digitalWrite. ¿Cómo se logra el mismo resultado en código AVR? Compare ambas alternativas en términos de performance, facilidad de uso, etc.

PinMode configura el pin especificado para comportarse como un input o como un output. Sus parámetros son pin, que es el número del pin al que se quiere setear como input o output, y mode, que indica el modo en que se quiere que este pin actue, en input, output, o input de pullup (de la resistencia)

DigitalRead, es una función que devuelve el valor almacenado dentro del pin especificado, puede devolver High o Low, indicando que hay un 1 o hay un 0 lógico almacenado en el pin. Si no está conectado a nada el pin, por la presencia del ruido, puede devolver un valor High o Low (1 o 0) dependiendo. Como parámetro es necesario que se especifique el número del pin del cuál se quiere leer el valor almacenado.

DigitalWrite, escribe un valor alto o bajo en el pin digital especificado. Si el pin fue configurado como pin de output con la función pinMode(), el voltaje será seteado al correspondiente valor 5V para alto, o 0v para bajo. Si el pin fue configurado como un input, permitirá habilitar (con HIGH) o deshabilitar (con LOW) la resistencia interna de pull up en el pin de input.

En atmel studio se logra el código de la siguiente manera:

PinMode():

//Inicializar el pin del pulsador como entrada

DDRD &= ~(1<<DDD2);

//Inicializar el pin del led como salida

DDRB |= (1<<DDB5);

DigitalRead():

//Leer el estado del pulsador

buttonState = PIND & (1<<PD2);

DigitalWrite():

PORTB &= ~(1<<PB5); apago el led escribiendo un cero

PORTB |= (1<<PB5); prendo el led escribiendo un 1

AVR es mucho más eficiente que utilizar el entorno de arduino, las cosas que programamos son más a bajo nivel, por lo tanto, si se lo programa de manera correcta, los programas resultantes serán mucho más eficientes y más pequeños. Además, dichos programas tendrán mejor desempeño y serán más rápidos.

A diferencia, el entorno de arduino, puede ser muy útil para aplicaciones muy complejas, en los cuales se precise más alto nivel, llevando a una mejoría en la facilidad de la programación. El entorno de arduino es mucho más ameno al usuario, aunque ocupa más espacio que AVR.

El arduino suele tener 32k de espacio

4) Se realiza sobre el toolchain.

5) Estudiar la documentación de la librería “Serial” de Arduino [4]. En particular, analizar los ejemplos “Digital Read Serial” [3] y “Digital Write Serial” [5]. Prestar especial atención al manejo de la librería Serial para lectura y escritura. Analizar qué recursos de hardware son utilizados por la librería y cómo influye ello en el diseño del software embebido.

int SoftwareSerial::read()

{

if (!isListening())

return -1;

// Empty buffer?

if (\_receive\_buffer\_head == \_receive\_buffer\_tail)

return -1;

// Read from "head"

uint8\_t d = \_receive\_buffer[\_receive\_buffer\_head]; // grab next byte

\_receive\_buffer\_head = (\_receive\_buffer\_head + 1) % \_SS\_MAX\_RX\_BUFF;

return d;

}

En la función read utiliza únicamente buffers.

En la función write, toma las variables definidas como locales, y el compilador las pone en regisros antes de deshabilitar las interrupciones y entrar en las secciones críticas de timing definidas más abajo en el código, lo que hace mucho más fácil para poder verificar los timings de los ciclos.

Va jugando con los pines y enviando 0’s y 1’s

By declaring these as local variables, the compiler will put them

// in registers \_before\_ disabling interrupts and entering the

// critical timing sections below, which makes it a lot easier to

// verify the cycle timings

8) El puerto com la maquina lo utiliza para poder comunicarse con el arduino. Cuando no se usa como cargador de código, lo podemos utilizar para mandar información al arduino como puerto COM,

El arduino manda información a la pc por el monitor serie (consola remota) a través del puerto com.

Desde el código arduino, serial pirnt y serial read para mandar información o leer información, inicializar el puerto serie a la velocidad que vayan a usar y el monitor serie tiene que tener la misma velocidad.

Serial bing para setear la velocidad

Buscar función serial print serial write, con arduino, página de arduino CC, aplicación de desarrollo de arduino hay ejemploos

El puerto serie cuando grabamos imagen al micro no se puede enviar información.

You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().