Ayudantía N°7

Ayudantes:

Andreas Krumpoeck

Juan Santiago

Resumen de Código C primera parte:

Este resumen se asocia al uso de un Arduino UNO. Sin embargo, la idea es que sea general para todos los microcontroladores de arquitectura AVR. Para esto se mostrará el diagrama PINOUT del mismo Arduino y se explicará lo principal para poder empezar a programar.

También cabe destacar que Arduino tiene su propio lenguaje de programación, sin embargo, no se verá este, ya que no es general para todos los microcontroladores, por lo que se profundizará solamente en el Código C.

Primero se muestra el esquema PINOUT:

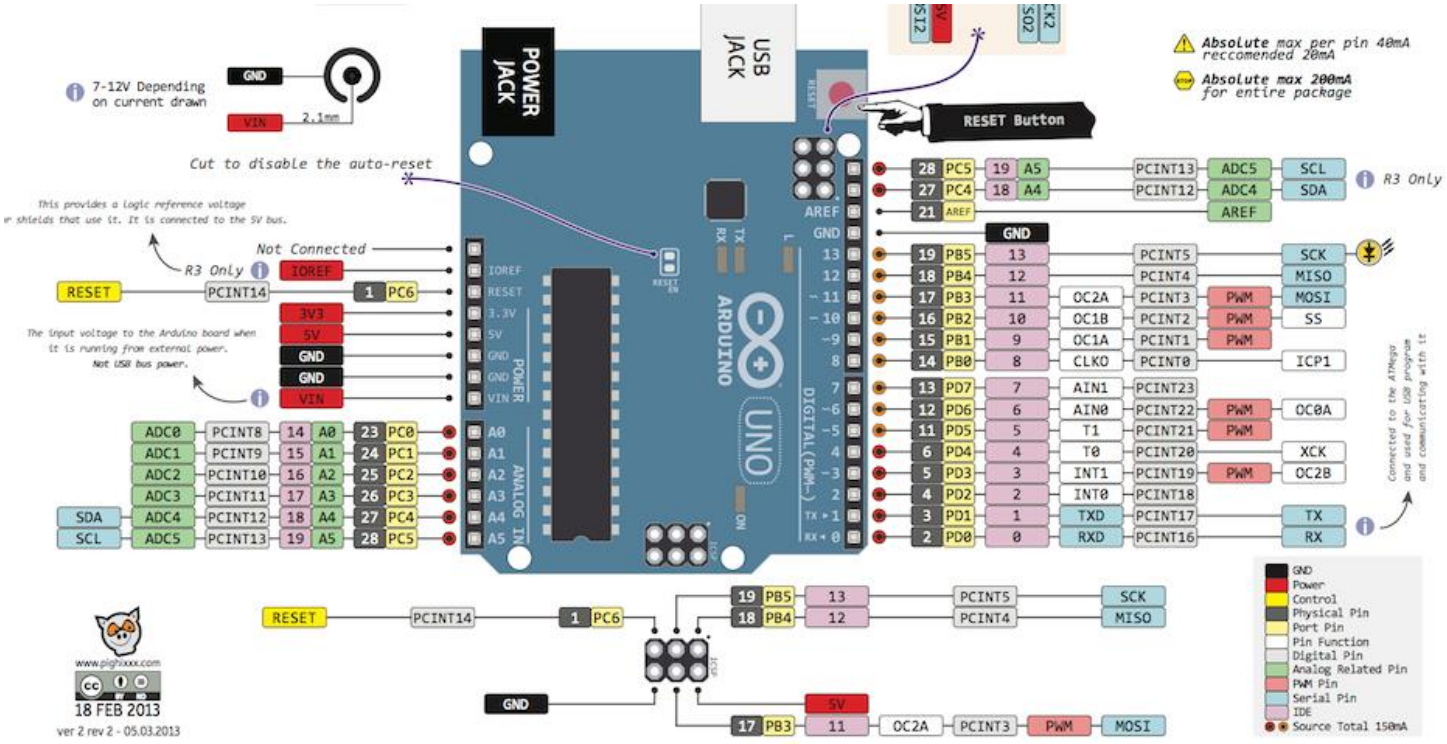


Imagen N°1: Pinout Arduino UNO

En esta primera parte solo se le prestará atención al port PIN. Si se observa en la imagen anterior en la zona inferior derecha se encuentra que significa cada color en el que esta cerrado cada función de cada uno de los pines.

Imagen N°2: Simbología Pin out

Ahora que ya se tiene visto que la estructura de un Arduino, ahora se puede hablar sobre su microcontrolador, ya que el Arduino es una extensión del microcontrolador. El microcontrolador que ocupa el Arduino es un ATMEGA328P. Este es un microcontrolador desarrollado por ATMEL, por esto que es utilizamos el ATMEL Studio para programar y no la plataforma Arduino. Estos microcontroladores son de 8 bits. Si se observa bien los pines están distribuidos por letras. En este caso C, B y D. Cada uno tiene a lo más 8 pines, sin embargo, algunos puertos llegan a 7 pines. Esto quiere decir que por cada puerto (Conjunto de pines que pertenecen a una misma letra en este caso) puede realizar todas las combinaciones posibles, es decir 2^8 o 256 posibilidades, para el caso del puerto D que tiene 8 pines.

Lo siguiente es explicar que se puede hacer con cada pin. Un pin es una forma que tiene el microcontrolador de comunicarse con lo que uno desee. Este puede tener dos estados lógicos. Estos serían o que esté pasando corriente o que no este pasando corriente. Solamente que esto será mediante un 0 (no pasa corriente), o un 1 (pasa corriente). O bien puede ser al revés, depende de cómo se defina. Además, pueden ser considerados como entradas o salidas, es decir, como entrada va a estar leyendo si entra o no entra corriente, y como salidas va a estar dejando pasar corriente o no según se necesite. Un ejemplo de entrada puede ser un botón, si se deja pasar corriente a través del botón esta llegará al PIN correspondiente y ahora esa información será procesada y se podrá realizar ejecuciones como por ejemplo prender un LED. Este último puede ser considerado como salida. Esto ya que el es prendido por el microcontrolador.

A continuación, se explicará cómo relacionar toda esta idea e información el Código en sí.

Registros de un microcontrolador ATMEGA328P:

Para empezar en esta ayudantía solamente utilizaremos Led's y botones, para esto utilizaremos tres tipos de registros. El **DDRX**, **PORTX** y **PINX**. Donde en la **X** dependerá de que puerto ocupemos. Por ejemplo, si usamos el puerto D estos tres registros quedarán como DDR**D**, PORT**D** y PIN**D**.

¿Dónde encontrar la información de cada registro y que hace? En la página https://mecanica-usach.mine.nu/15167/ (página del curso) se encuentra el Datasheet del ATMEGA328P. Aquí se encuentra como configurar todo este microcontrolador y cada Registro existente. Aquí se ira tomando un resumen de lo que utilizaremos, diciendo cada página donde se podrá encontrar la información extraída.

Se empezará por explicar el registro **DDRX:**

13.4.9 DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
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	

Imagen N°3: Registro DDRD. Pág 73 del Datasheet

Aquí se observa el registro DDRD, se tomará el puerto D para todas las explicaciones.
¿Qué hace este registro? Este registro permite establecer las entradas y salidas del puerto D.
Si se enciende un pin se establece como Salida, de lo contrario, como entrada. Es por esto cuando se programará es que se omite la escritura de las entradas, ya que el valor inicial es cero en este registro.

En código C se escribiría así para definir salidas:

```
DDRD |= _BV(PD0) | _BV(PD1) |_BV(PD2); // Se define como salida del primer pin hasta el numero tres.  Observar que dentro del _BV() se escribe PDX, donde P es para decir pin, D para decir que es del puerto D y el número hace referencia al número del Pin.
```

Ahora bien, ¿por qué se utiliza el '' | ''? es por es el comando '' or '' en este código, entonces así puede encender estos tres registros, sin embargo, no altera el estado lógico de los demás.

```
DDRD = 255; // Aquí estaríamos definiendo que todos los pines del puerto D son salidas, ya que escribiendo 255 estamos diciéndole en binario el siguiente número: 11111111, esto quiere decir que todos los pines tendrán valor uno.
```

```
DDRD= 0xb11111111; // Aquí se escribe lo mismo que el caso anterior, pero de forma binaria.
```

Para este curso lo más común será utilizar la macro BIT VALUE o escrito en el primer ejemplo _BV()

Registro **PORTD**:

13.4.8 PORTD – The Port D Data Register

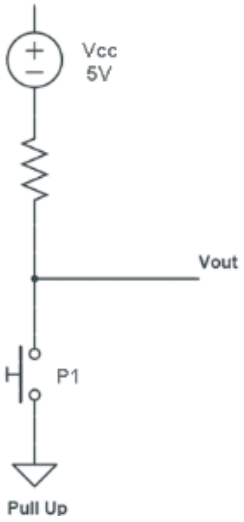
Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Imagen N°4: Registro PORTD. Pág 73 del Datasheet

Este registro nos permite establecer varias cosas. Primero el estado lógico inicial de las salidas del programa. Por ejemplo, si se establece que el pin 0 es una salida, su valor inicial será cero, por lo cual si se tuviera un led este partiría apagado hasta que cambie el estado lógico. Sin embargo, si le cambio su valor inicial a uno este partiría encendido hasta que yo lo cambie de su estado lógico. Para poder realizar este último caso se haría de la siguiente manera:

```
DDRD |= _BV(PD0); // Definimos como salida el pin cero
PORTD |= _BV(PD0);  //Hacemos que se encienda dicho pin.
PORD &= ~_BV(PD0); //Con esto podemos apagar el pin cero, mediante la operación ‘’ & ‘’ o mejor dicho ‘’ y ‘’, junto con la operación de negación ‘’ ~ ‘’.
```

Por otra parte, para las entradas cambie su uso. Si se prende un pin que haya sido definido como entrada, se activarán las resistencias PULL-UP, de lo contrario no se activa este tipo de resistencia. Pero,
¿Qué es una resistencia PULL-UP? Aquí se hará una pequeña pausa para explicar esto.



- Resistencias **PULL-UP.**

Imaginar que se tiene un botón, este cuando se activa se deja pasar corriente al microcontrolador y éste dice “*OH, llega corriente hare algo”,* y cuando dejas de apretar, este simplemente dice *“Ya no me llega nada, así que dejare de hacer lo que estaba haciendo”.* Bueno, el problema es que no existe el concepto de que no llegue nada. Si no llega nada, no se puede establecer un estado lógico a partir de esto. Por esto es que se utilizan las resistencias PULL-UP. Se muestra un esquema a continuación se muestra la idea de una resistencia PULL-UP.

Imagen N°5: Esquema de resistencias Pull-up

Que es lo que hace esto. Continuamente se esta enviando 5volt´s al voltaje de salida, con una resistencia entremedio para no generar un cortocircuito. Sin embargo, cuando el botón uno es apretado, deja de pasar corriente hacia al voltaje de salida y este desvía la corriente hacia un cable a tierra. Con esto se evita el problema anteriormente mencionado.

Un pequeño paréntesis antes de continuar es que los botones tienen un problema de DEBOUNCE. Esto traducido al español se denomina como rebote. Cuando se presiona el botón uno creo que esta pasando lo que debe pasar, sin embargo, es tan sensible el botón que en el tiempo que uno aprieta este deja pasar corriente y además la va cortando, generando un problema en la lectura. Esto es solucionado en esta ayudantía con un DELAY. Ya se profundizará mas en esto.

Ahora que ya se entiende mejor este concepto, es que se puede pasar a explicar que hacer con el PORTD cuando se definen como entradas al problema.

Entonces, como ya se mencionó, para activar la resistencia PULL-UP solamente bastará con prender este pin, por ejemplo, si se quisiera dejar el pin 7 como entrada, con una resistencia PULL-UP activada bastaría con realizar lo siguiente:

```
DDRD &= ~_BV(PD7); //Esto se puede omitir como ya habíamos mencionado, ya que el valor inicial es de cero.

PORTD |= _BV(PD7); //Aquí queda activada la resistencia PULL-UP.
```

Entonces este pin estará constantemente marcando un uno, ya que estará pasando 5 volts continuamente y cuando se apriete el botón este marcará cero. Importante mantener esto en cuenta cuando se decida programar.

Registro **PIND:**

13.4.10 PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
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	

Imagen N°6: Registro PIND. Pág 73 del Datasheet

Este registro básicamente permite saber en que estado lógico se encuentra cada pin, independiente si ha sido configurado como entrada o salida. Mas adelante se vera su aplicación. Muy importante para por ejemplo saber si cambia el estado lógico de una entrada.

Finalmente se hablará de algunas cosas a utilizar en los ejercicios a realizar.

Por ejemplo, como preguntarse en que estado lógico esta un botón. Como se mencionó cuando se apreté un botón el estado lógico del pin cera cero. Es por esto que debemos preguntarle al pin cuando el valor cambie de uno a cero. Para esto se utiliza la comando bit_is_clear(PINX,PXY), donde la X dependerá del puerto a ocupar, la Y dependerá del número del pin. Si es apretado el botón, el estado lógico será cero, si es cero, el comando bit_is_clear dará como resultado un uno, ya que si, se encuentra limpio, de lo contrario marcará cero. Es muy útil para la aplicación que se busca darle.

El uso del delay. Este comando se utilizará como _delay_ms(tiempo). El delay es un comando que permite derechamente detener el funcionamiento completo del microcontrolador. ¿Cuándo tiempo? El tiempo que se especifique. Al utilizar ms se esta diciendo que el tiempo estará dado en milisegundos. Y el número que ingrese dentro estará medido en milisegundos. Por este lapso se detendrá el programa. La aplicación que se puede dar para esta ayudantía es poder hacer que LED's de prendan y apaguen en un tiempo determinado y además poder reparar el DEBOUNCE de los botones. No hay que dejar de lado el hecho que detenga todo el microcontrolador no es bueno, ya que, si se estuvieran haciendo otras tareas en segundo plano, estas también se verían afectadas. Por esto es que se evita el uso de delay comúnmente, la parte buena es que ayuda a empezar a introducirse en el mundo de la programación en Código C, ya que es una herramienta muy simple de usar en comparación con lo que realmente se debe hacer.

Se verá ahora a grandes rasgos la estructura que debe llevar un programa en general.

```
#define F_CPU 12000000 // Aquí se define la frecuencia del oscilador del microcontrolador.
//Lo siguiente es incluir las librerías que se van a ocupar
//Ejemplo librería del delay
#include <util/delay.h> // Aquí se le dice al programa que se le deben incluir el delay, si no el programa
// no reconocerá cuando ocupemos este comando.

//Podemos para hacer más fácil el entendimiento del código asignarle nombre a los puertos, entradas
o salidas que vayamos a utilizar.
//Ejemplo si utilizamos el puerto D

#define puerto_D PORTD; // Aquí en vez de utilizar el nombre PORTD se utiliza puerto_D, así se
// entiende mejor cuando se programa y cuando otra persona lea el código.
//Otro ejemplo, imaginar que el pin 7 del puerto D va a ser para un boton.
#define boton PD7; //Con esto si se entenderá mejor que se esta haciendo cuando se programe

//Luego de forma opcional se o según amerite, se pueden crear funciones en código C, estas se podrán
//llamar en cualquier parte del programa, por ejemplo, se pueden definir las entradas y salidas
//mediante una función y luego llamarla en alguna parte del código.
```

```
void configuracion_pines()
{                              //El corchete sirve para abrir la función
    DDRD |= _BV(PD1) |_BV(PD2); //Aquí se define el pin 1 y 2 como salidas
    puerto_D |= _BV(boton); //utilizando la definición anterior se puede definir la resistencia pull up
//del boton
    PORTD |= _BV(PD7); //Lo mismo de la línea anterior pero esta vez sin utilizar las definiciones
} //También sirve para cerrarla, se debe abrir y cerrar la función para que el compilador entienda bien

int main(void)
{
//Código ejecutar
}
```

Las líneas anteriores sirven para decirle al código que desde aquí se empezará a ejecutar el programa, todo lo que este dentro del int main(void) será ejecutado por el microcontrolador.

Y ya dentro de este se puede hacer lo que se plazca. Lo demás se ira explicando en el ejercicio de la ayudantía.

Problema 1:

Se desea hacer que dos leds prendan de forma alternada cada dos segundos.

Solución problema 1:

Primero se monta el siguiente esquema en un Arduino UNO:

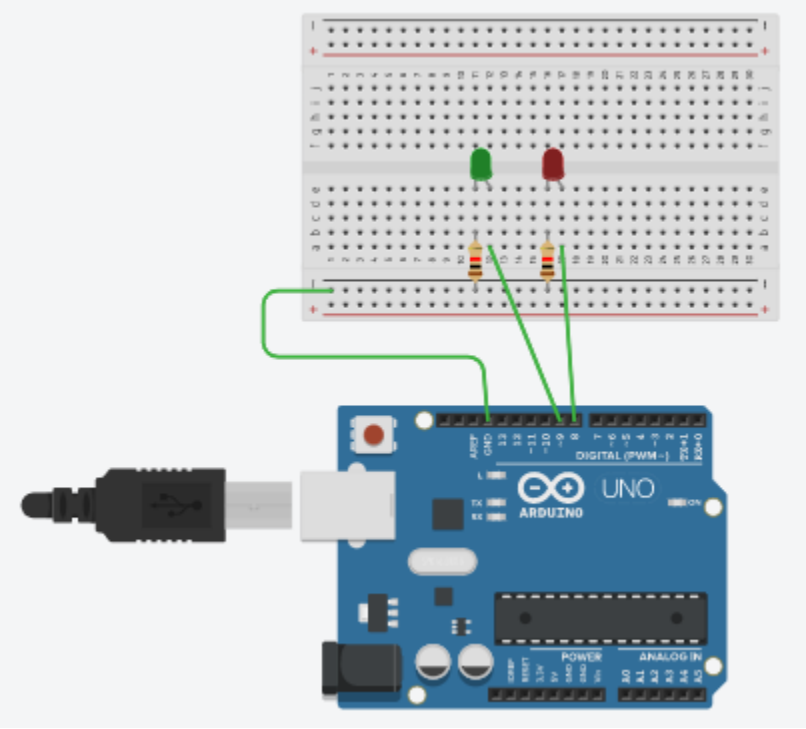


Imagen N°7: Esquema en un Arduino UNO.

Se puede observar según el PIN OUT mostrado al comienzo que el puerto utilizado es el puerto B. Desde este puerto se utilizaron los pines PB0 y PB1. El código de programación se muestra a continuación:

```
#define F_CPU 12000000 // Se define la frecuencia del Oscilador
#include <avr/io.h> // Libreria de I/O
#include <util/delay.h> //Libreria del delay

//Definimos nombres para los puertos, direcciones, pin.
#define puerto_B PORTB //Definimos un nombre para el puerto B
#define direcciones_B DDRB //Definimos un nombre para las direcciones B
#define led_1 PB0 //Definimos un nombre para el primer led
#define led_2 PB1 //Definimos un nombre para el segundo led

//Hacemos una función para definir la funcion de cada pin
void config_puerto_B()
{ //Abrimos la funcion
	direcciones_B |= _BV(led_1) |_BV(led_2); //Se define el pin cero y uno como salidas
} //Cerramos la funcion

int main(void) //Se empieza a ejecutar el programa
{ //Abrimos el main void
	config_puerto_B(); //Llamamos a nuestra funcion que configura los pines
	const int desfase = 2000; //Establecemos el tiempo de encendido y apagado, dos segundos son 2000
//milisegundos

	while (1) //Realizamos un ciclo while que sea eternamente verdadero para que haga esto siempre
	{ //Abrimos el ciclo While
		puerto_B |= _BV(led_1); //Prendemos el primer led
		puerto_B &= ~_BV(led_2); //Apagamos el segundo led
		_delay_ms(desfase); //Aplicamos el desfase de dos segundos
		puerto_B |= _BV(led_2); //Prendemos el segundo led
		puerto_B &= ~_BV(led_1); //Apagamos el primer led
		_delay_ms(desfase); //Aplicamos el desfase de dos segundos
	} //Cerramos el ciclo while
} //Cerramos el main(void)
```

Problema 2:

Se desea hacer funcionar dos secuencias de leds. Una que cuatro leds´s vayan parpadeando cada 1 segundo todos juntos. La segunda que se prendan todos, pase medio segundo, se prenda uno, pase medio segundo, y se prenda el otro y así hasta que se pase por todos.
Estas secuencias deben ser accionadas mediante un botón.

Para esto se muestra el siguiente esquema realizado en un Arduino UNO.

Solución problema 2:

Se puede apreciar que los led's fueron conectados, según el pin out del Arduino Uno, a los pines PB0, PB1, PB2 y PB3. Por otro lado, el botón fue conectado al pin PD7.

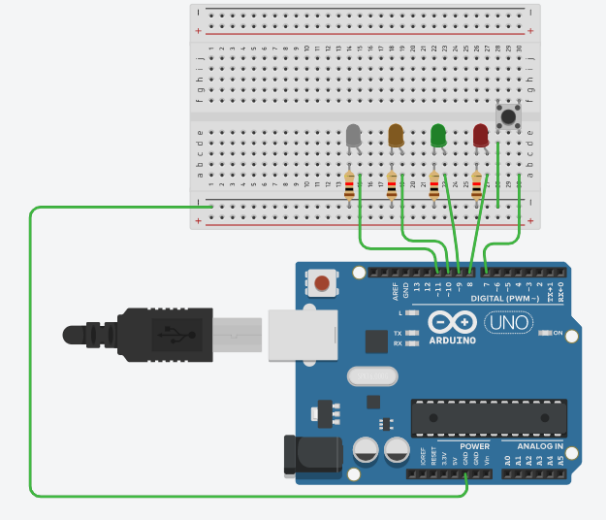


Imagen N°8: Esquema en un Arduino UNO.

El código de programación se presente a continuación, se omitirán comentarios ya realizados en el ejercicio anterior:

```
#define F_CPU 12000000
#include <avr/io.h>
#include <util/delay.h>

//Definimos nombres
#define puerto_B PORTB
#define direcciones_B DDRB
#define puerto_D PORTD
#define direcciones_D DDRD
#define pines_D PIND
#define led_1 PB0
#define led_2 PB1
#define led_3 PB2
#define led_4 PB3
#define boton PD7

//Funcion de configuracion de puertos
void config_puertos()
{
	direcciones_B |= _BV(led_1) | _BV(led_2) | _BV(led_3) | _BV(led_4); //Salidas del puerto
	puerto_D |= _BV(boton); //Activamos la resistencia pull-up
}

int main(void)
{
    config_puertos();
	unsigned char tarea = 1;
    while (1)
    {
		if (bit_is_clear(pines_D,boton))
		{
			_delay_ms(1000); //Para contrarrestar el rebote
			if (tarea == 1)
			{
				tarea = 2;
			} if (tarea == 2)
			{
				tarea = 1;
			}
```

```
            }
            if (tarea == 1)
            {
                puerto_B = 255; // Prende todo el puerto
                _delay_ms(1000);
                puerto_B = 0; // Apaga todo el puerto
                _delay_ms(1000);
            } if (tarea == 2)
            {
                puerto_B = 255;
                _delay_ms(500);
                puerto_B |= _BV(led_1);
                puerto_B &= ~_BV(led_2) & ~_BV(led_3) & ~_BV(led_4);
                _delay_ms(500);
                puerto_B |= _BV(led_2);
                puerto_B &= ~_BV(led_1) & ~_BV(led_3) & ~_BV(led_4);
                _delay_ms(500);
                puerto_B |= _BV(led_3);
                puerto_B &= ~_BV(led_1) & ~_BV(led_2) & ~_BV(led_4);
                _delay_ms(500);
                puerto_B |= _BV(led_4);
                puerto_B &= ~_BV(led_1) & ~_BV(led_2) & ~_BV(led_3);
                _delay_ms(500);
            }
    }
}
```