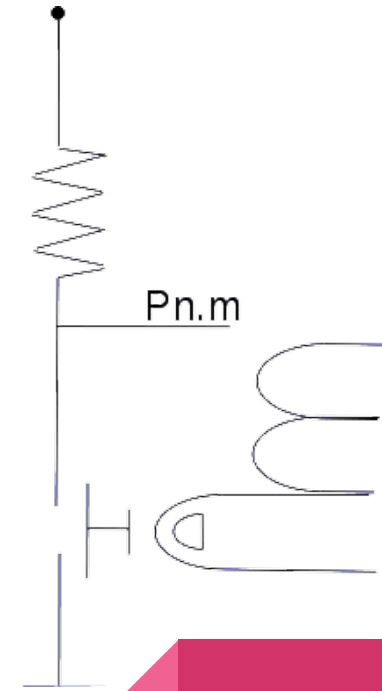
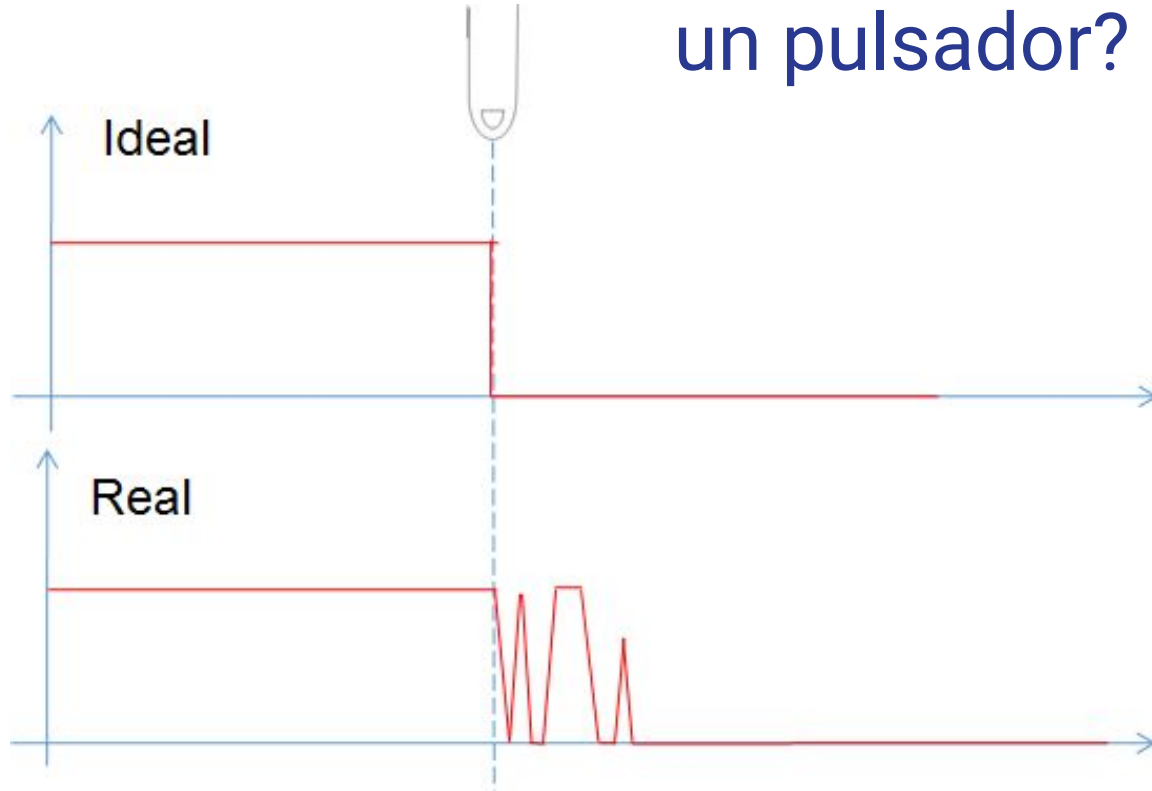


Estrategias de programación para entradas digitales

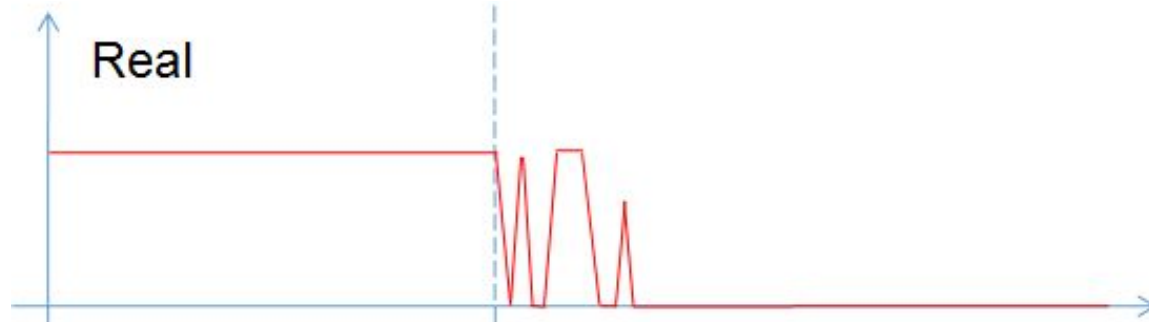
Informática II - R2004
2021

Qué pasa en la señal eléctrica cuando presiono un pulsador?



Al presionar una tecla tenemos ruido asociado al rebote mecánico

Caracterizando la señal de “rebote”

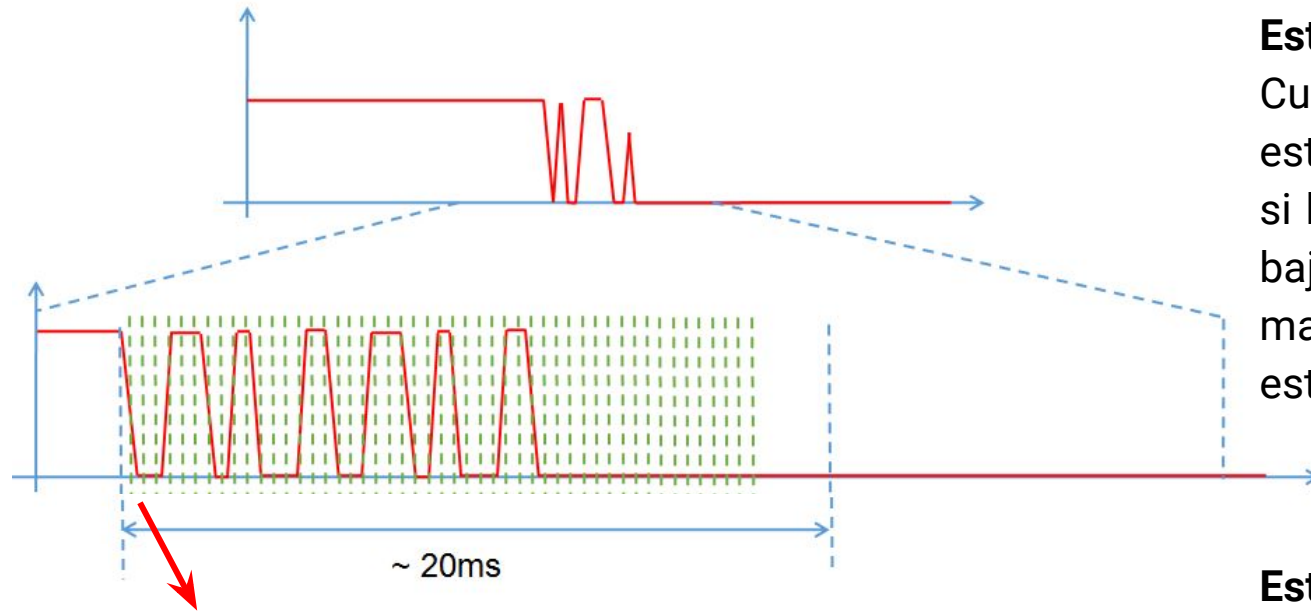


Esta señal puede producirse debido al rebote de los contactos metálicos del pulsador. Su duración es muy variable, ya que depende de muchas características mecánicas y eléctricas, pero se estima que a los 20 milisegundos de la primera detección de un cambio de estado la señal se encuentra estabilizada.

El microcontrolador, que funciona a velocidades en el orden de las décimas o centésimas de microsegundo, puede detectar cada uno de estos rebotes como una opresión independiente, y generar así que una pulsación se convierta en 2 o más

Este ruido se puede eliminar agregando un capacitor en el circuito, que actúa como un filtro para las señales que transicionan muy rápido (conocido como filtro pasabajos), o se puede eliminar por software.

Estrategias para eliminar el “rebote”



Las líneas en verde serían las interrupciones de systick. Si miro la entrada cada vez que se genera esta interrupción, puedo evaluar periódicamente el estado de esta entrada y temporizar el proceso.

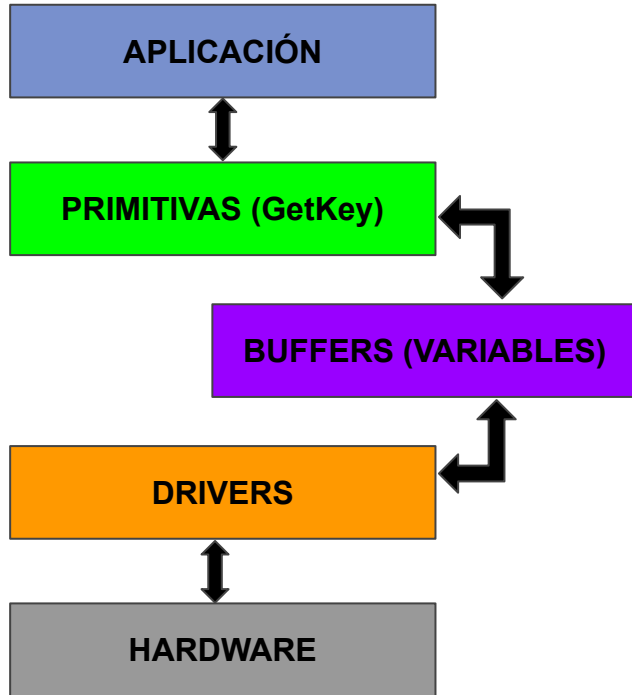
Estrategia 1:

Cuando veo que cambió de estado chequeo periódicamente si la señal está en estado alto o bajo. Cuando cuento que se mantuvo N veces en el mismo estado, la señal está establecida

Estrategia 2:

Desde que detecté que el botón está presionado espero un tiempo adecuado (~20mseg)

Utilización de un driver para “filtrar” las entradas



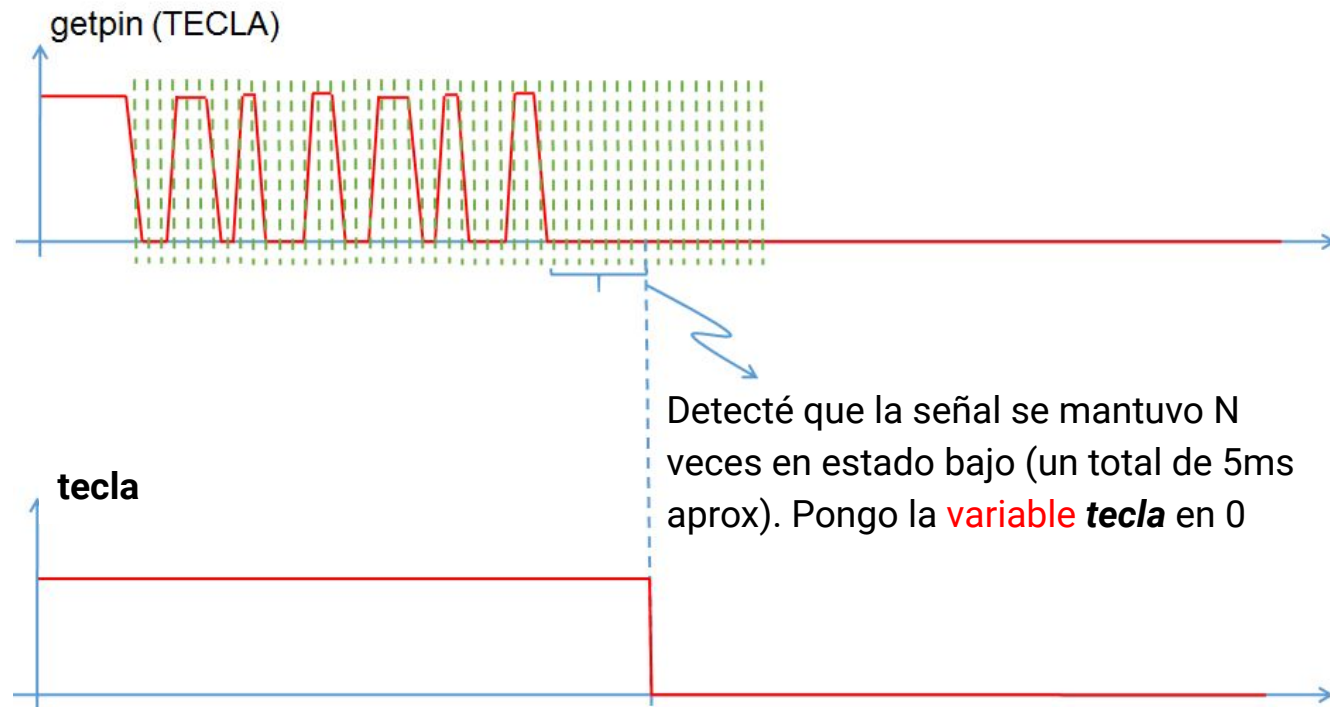
Al usar un **driver** que esté **mirando continuamente** la señal de los **pulsadores** (u otras entradas digitales) y las filtre, voy a necesitar tener en una variable el resultado de este filtrado.

Desde las **primitivas**, voy a estar **mirando esta variable** y no la entrada física. Dado que el proceso de filtrado va a demorar un tiempo (pequeño) en evaluar la entrada, la variable filtrada va a tener un delay con respecto a la entrada física.

La **aplicación invoca** a las **primitivas**.

Utilización de una variable como buffer

Cualquier filtro (de hardware o software) agrega un delay, como puede verse en el gráfico siguiente:



En nuestra aplicación **en lugar de usar:**

```
If ( getpin(TECLA) )  
{  
  ...  
}
```

usaremos una **función primitiva** que vea la **variable tecla**

Invocando el Driver

- Se debe llamar periódicamente.
- La cuenta de los N estados estables debe representar 5ms aproximadamente.
- El teclado lo va a utilizar un ser humano que no presiona las teclas con mayor velocidad que 50ms.

PARA QUE LOS TIEMPOS SEAN ESTABLES Y NO DEPENDAN DEL TAMAÑO DEL CÓDIGO
EL DRIVER DE TECLAS SERÁ INVOCADO EN EL SYSTICK


```
void SysTick_Handler(void)
{
    DebounceEntradas( );
    BarridoDisplay( );
    RefrescoSalidas( );
    AnalizarTimers( );
    EscrituraLCD( );

    DriverTecladoSW ( );
}
```

Driver para eliminar “rebote” y lectura de teclas

El Driver deberá eliminar el rebote y cargar en la variable **Tecla** cuando detecte una tecla.

Hagamos la función (Sabiedo que la misma es invocada cada 1ms en el systick), para ello debemos:

- Chequear el estado del pin donde esté conectado el pulsador, para ello disponemos de la función GetPIN.
 - Contar la cantidad de veces que encontramos el pin en 0 (sabiedo que 0 representa que el pulsador fue presionado)
 - Cuando hayamos contado N veces el pin en 0 se debe cargar en la variable Tecla que detectamos la opresión.
 - En caso que no hayamos llegado a N pero se detecta que el pin no está en 0 se reinicia la cuenta.
- 

Función antirrebote

```
void DriverTecladoSw( void )  
{
```

```
    static uint8_t count = 0;  
    uint8_t key;
```

define que incluye puerto y pin

```
    key = GetPIN( KEY );
```

```
    if( key == 0 && count < DEBOUNCE_COUNT )  
    {
```

```
        count++;
```

```
        if(count == DEBOUNCE_COUNT)
```

```
        {
```

```
            Tecla = key;
```

```
        }
```

Variable buffer global

```
    }
```

```
    else if(key == NO_KEY)
```

```
    {
```

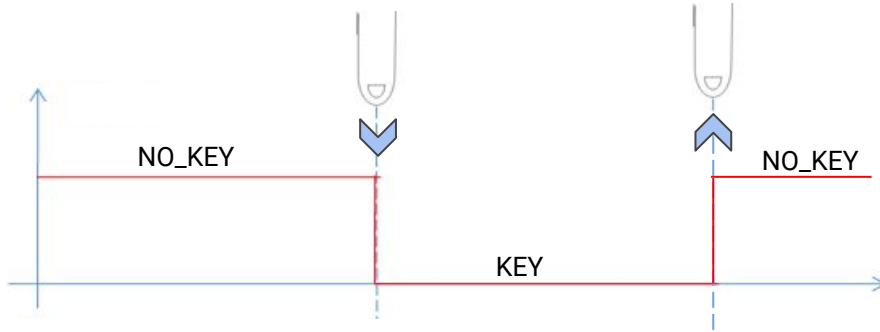
```
        count = 0;
```

```
    }
```

```
}
```

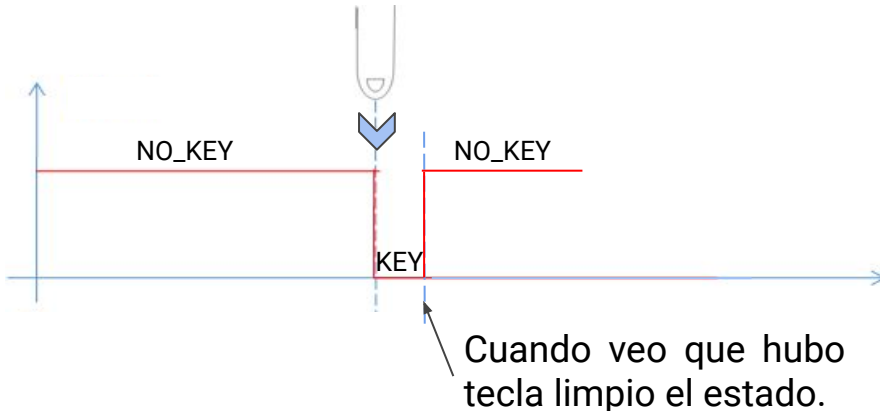
En este caso que
leemos una sola tecla
NO_KEY vale 1

Detección de flanco o de nivel



Nivel: Mientras esté presionado el pulsador la variable Tecla vale KEY después NO_KEY

Tengo que modificar el Driver para que detecte cuando se suelta. Generalmente en teclados de sistemas embebidos se utiliza poco.



Flanco: la variable Tecla vale KEY cuando se presiona y después de ser leída se limpia pasando a valer NO_KEY

La "limpieza" se realiza en la primitiva.

Función Primitiva


La función primitiva es la que luego utilizará la aplicación, la misma leerá la variable global Tecla y la “limpiará”.

```
uint8_t GetKey ( void )  
{  
  
    return Tecla;  
}
```

Función primitiva

```
while(1) {  
    if(GetKey() == SW1)  
    {  
        //Mi código  
    }  
}
```

Aplicación Independiente
variable global y del Hw.



Teclado compuesto por varias Teclas

Realizaremos el driver para un teclado compuesto por varias teclas pero que solamente una de ellas puede estar presionada a la vez.

Para ello debemos:

- Leer de a uno los pines y en caso que alguno esté presionado devolver un código que lo identifique. Esto se suele denominar barrido de las teclas.
- Realizar el antirrebote de la tecla presionada. Para ello hay que verificar que por N veces esté presionada la misma tecla.



Barrido de teclas.

```
uint8_t DriverTecladoHw( void )  
{  
  
    if( !GetPIN( KEY0 ) )  
        return SW1;  
  
    if( !GetPIN( KEY1 ) )  
        return SW2;  
  
    if( !GetPIN( KEY2 ) )  
        return SW3;  
  
    if( !GetPIN( KEY3 ) )  
        return SW4;  
  
    return NO_KEY;  
}
```

Recordemos que en nuestra placa las teclas ponen el PIN a 0 cuando se presionan.

```
#define SW1 0
```

```
#define SW2 1
```

```
#define SW3 2
```

```
#define SW4 3
```

```
#define NO_KEY 255
```

NO_KEY se pone en 255 para poder agregar teclas.



Antirrebote para un conjunto de teclas.

```
void DriverTecladoSw( void )
```

```
{
```

```
    static uint8_t lastKey = NO_KEY;
```

```
    static uint8_t count = 0;
```

```
    uint8_t key;
```

```
    key = DriverTecladoHw();
```

```
    if( lastKey == key && count < DEBOUNCE_COUNT )
```

```
    {
```

```
        count++;
```

```
        if(count == DEBOUNCE_COUNT)
```

```
        {
```

```
            Tecla = key;
```

```
        }
```

```
    }
```

```
    else if(
```

```
    {
```

```
        count = 0;
```

```
    }
```

```
    lastKey = key;
```

```
}
```

Invoco a la función
que hace el barrido.

Verifico que la tecla sea igual a la
última presionada para incrementar
la cuenta.

key != lastKey)

En caso que sea diferente o no
haya ninguna tecla reinicio la
cuenta.

Ejercicios

Ejercicio 0

- Implementar el driver y primitiva para realizar el antirrebote de la tecla PORT0,4. Es la que se encuentra en el stick.

Ejercicio 1

- Implementar el driver y primitiva para utilizar como teclado 4 teclas independientes (que podrían ser un pequeño teclado de selección de opciones). El driver deberá comprender que no se presiona más de una tecla al mismo tiempo, como suele suceder en este tipo de teclados.



Ejercicio Avanzados

- Modificar el driver y la primitiva para que funcione por nivel y no por flanco. Para ello debo cambiar el valor de la variable global Tecla cuando detecto que se suelta y no debe “limpiarse” con la lectura.

