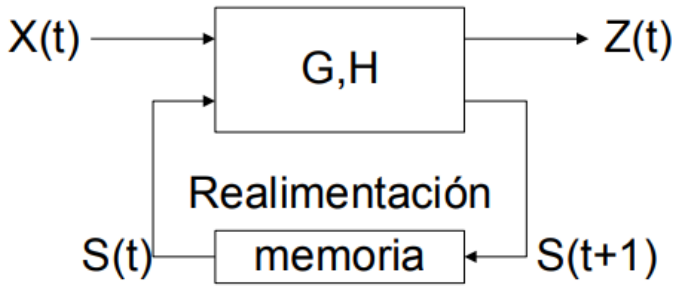
MÁQUINAS DE ESTADOS FINITOS

En los sistemas secuenciales la salida Z en un determinado instante de tiempo ti depende de X (la entrada) en ese mismo instante de tiempo ti y en todos los instantes temporales anteriores. Para ello es necesario que el sistema disponga de elementos de memoria que le permitan recordar la situación en que se encuentra (estado).



$X(t)$: $entrada\ actual$
$Z(t)$: $salida\ actual$
$S(t)$: $estado\ actual$
$S(t+1)$: $estado\ próximo$

G: $función\ de\ salida$
$Z(t) = G(X(t), S(t))$

H: $función\ de\ transición$
$S(t+1) = H(X(t), S(t))$

Como un sistema secuencial es finito, tiene una capacidad de memoria finita y un conjunto finito de estados posibles máquina finita de estados (FSM: finite state machine)

Modelado, ¿qué es un modelo?

Modelo es la representación simplificada de un sistema que contempla las propiedades importantes del mismo desde un punto de vista.
Los modelos de mayor utilidad son:

- Abstractos: realzan aspectos importantes y elimina aquellos innecesarios de un sistema.
- Comprensibles: se expresan en forma sencilla.
- Precisos: representan fielmente el sistema modelado.
- Predictivos: pueden usarse para responder cuestiones sobre el sistema modelado.
- Económicos: son más baratos de construir que el propio sistema.

Modelado en Sistemas Embebidos

Se utilizan como elementos de entrada para generación de código y además sirven para lograr un conocimiento más profundo del sistema.
Para esto se debe:

- Utilizar una representación gráfica del sistema a desarrollar.

- Describir el sistema con un cierto grado de abstracción.
- Generar código a partir del propio modelo.

Máquina de estados finitos (MEF)

¿Qué es?

- Es una herramienta gráfica usada para modelar sistemas.
- Describe el comportamiento de sistemas que dependen de eventos actuales y pasados.
- En cada instante, la máquina permanece en un estado determinado y dependiendo de las entradas actuales y pasadas, la máquina puede o no cambiar de estado y realizar acciones que a su vez influyen en el ambiente.
- Es un modelo de desarrollo muy utilizado.
- Un programa puede estructurarse de acuerdo a los estados propuestos.

¿Cómo se modela?

Para modelar una MEF se pueden usar diagramas de estado o tablas de estado. Cada cambio de estado implica diferentes respuestas del sistema, existen reglas bien definidas para que se produzcan y estas se van produciendo de acuerdo a un orden determinado.

Aplicaciones y características

- El uso clásico de MEF es para interactuar con un usuario.
- Una MEF es fácil de mantener ya que se pueden quitar o agregar estados sin modificar el resto.

Definiciones:

Entrada o Evento

Es el cambio de valor de una variable en un determinado instante. A veces se lo denomina “mensaje”. En general se tratan de varias variables. El conjunto de valores que puede tomar se lo denomina “alfabeto”.

Máquina abstracta

Es un modelo de computación (computación visto como un general) que establece el cómo se generan los eventos o mensajes de salida según sean los eventos o mensajes de entrada.

Estado

Es el conjunto de atributos que representan las propiedades de un sistema en un determinado instante de tiempo.

Transición

Es el cambio de estado del sistema e indica cómo se pasa de un estado a otro.

Tipos de Máquina de Estado:

Existen dos posibles implementaciones: Mealy y Moore.

Mealy

Estado

El estado siguiente depende de la entrada y del estado actual.

Salida

La salida del sistema depende tanto de la entrada como del estado actual. Las diferentes salidas son necesarias para el cambio de estado.

Moore

Estado

El estado siguiente depende de la entrada del estado actual.

Salida

La salida del sistema depende exclusivamente del estado actual, puede haber muchos estados con la misma salida, pero cada uno de ellos es diferente.

Toda máquina de Moore es un caso particular de una máquina de Mealy.

Metodología de trabajo

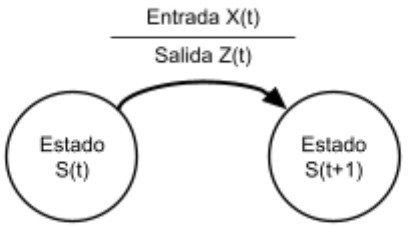
Identificación de entradas y salidas

- Determinar las señales que entran y salen del circuito que se quiere diseñar.
- En los sistemas de “control”, los sensores son considerados como entradas al circuito y los actuadores son considerados salidas.
- Si el sistema tiene etapas sincrónicas, el reloj (clock) es considerado una entrada.

Diagrama de transición de estados

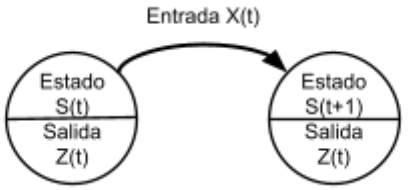
- Comenzar por el estado de reposo o inicio o reset (al encender el sistema)
- De cada estado deben salir 2^e transiciones ($e = números\ de\ entradas$)

Representación diagrama de transición de una Máquina Mealy



SALIDA Z(t) depende del **ESTADO** y de la **ENTRADA** X(t).

Representación diagrama de transición de una Máquina Moore



SALIDA Z(t) sólo depende del **ESTADO**, por eso se puede dibujar dentro del **ESTADO**

Comprobación y reducción del diagrama

Comprobación

Se hace para cada estado y se debe comprobar las condiciones de tránsito, todas las transiciones deben tener valores distintos de entrada.

Reducción

Dos estados son iguales si a partir de ellos la evolución es la misma, ante una misma combinación de las entradas el sistema, se va a los mismos estados y con las mismas salidas. En este caso, las transiciones se unen en un mismo estado.

Ejemplos

Lavarropas

Se dispone de un lavarropas que tiene 3 botones. “Encender”, “Detener”, “Lavar”, un indicador luminoso, un Motor, una traba puerta y un interruptor de Puerta.

Identificación de entradas y salidas

Entradas

- Botón “**E**ncender”, enciende el lavarropa.
- Botón “**D**etener”, detiene el lavarropa si se encuentra lavando.
- Interruptor “**P**uerta”, 1 Cerrada, 0 Abierta : apaga el lavarropa, solo si este está detenido.
- Botón “**L**avar”, inicia el lavado si el lavarropa se encuentra encendido y con la puerta cerrada.

Salidas

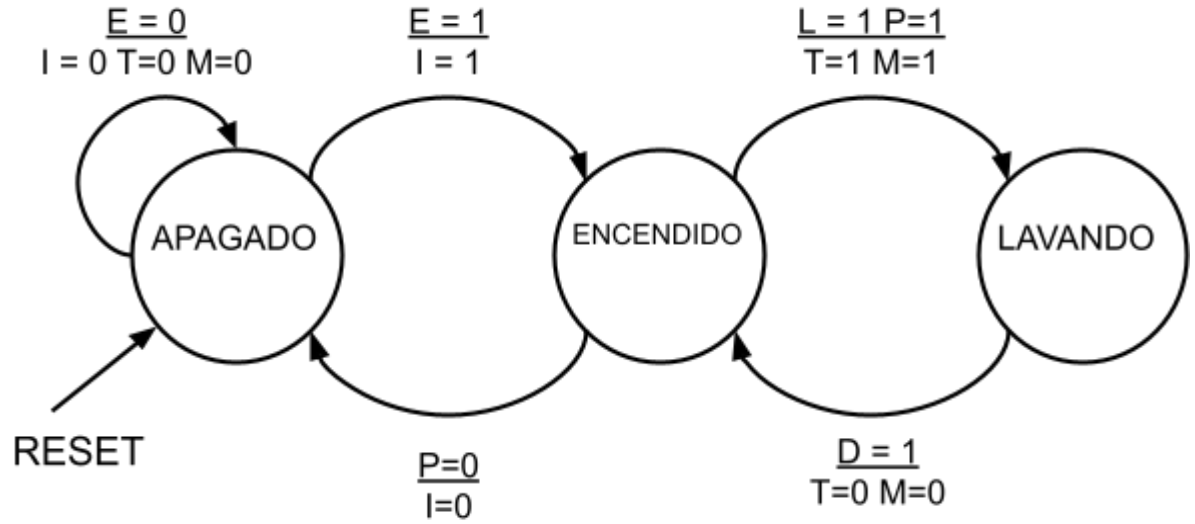
- **“I**ndicador luminoso”, se enciende si el lavarropa se energiza y se apaga si lo hace el lavarropa.
- “**M**otor”, comienza el ciclo de lavado si lavarropa se encuentra detenido
- “**T**raba Puerta”, bloquea la apertura si está lavando

Transición de estados

Podemos distinguir 3 estados posibles del lavarropa.

- Apagado. (RESET)
- Encendido.
- Lavando.

Diagrama de transición de estados



Intermitencia (regida por clock)

Se dispone de un indicador luminoso “L” el cual debe ejecutar siempre la misma secuencia, permanecer encendido durante 2 segundos y apagado 3 segundos. (La secuencia se debe ejecutar por siempre).

Identificación de entradas y salidas

Entradas

Este sistema a simple vista no cuenta con entradas definidas, analizando, el único evento que podemos identificar como una entrada al sistema es el reloj (clock)

- Reloj “**C**ontador”, variable interna al sistema que hace las veces de entrada.

Salidas

- **“I**ndicador luminoso”, se enciende durante 2 segundos, se apaga durante 3 segundos (el ciclo se repite indefinidamente).

Estados

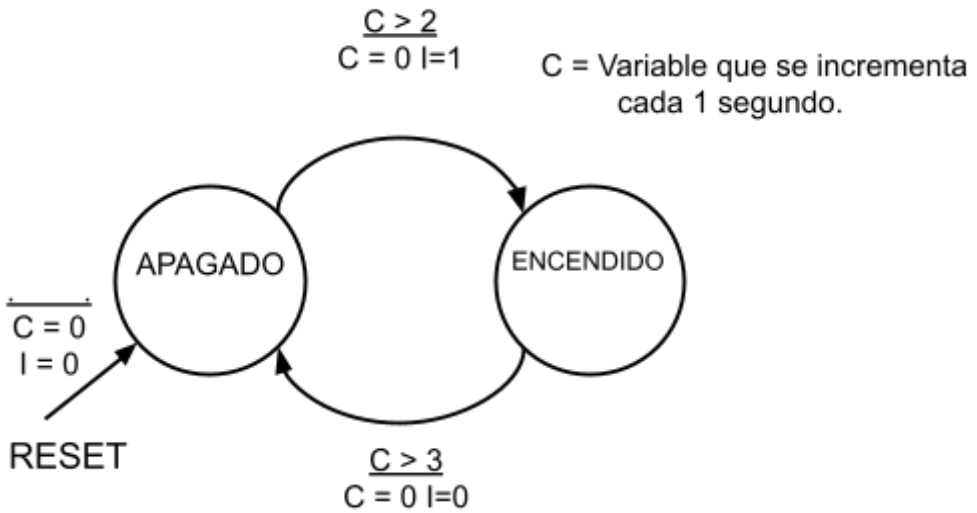
Podemos distinguir 2 estados posibles para el sistema.

- S1 1 **APAGADO**. (RESET)
- S2 2 **ENCENDIDO**.

Transición de estados

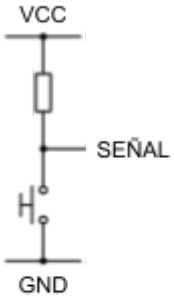
- S1 – **APAGADO**→ Transición a S2 cuando C > 2 seg.
- S2 – **ENCENDIDO** → Transición a S1 cuando C > 3 seg.

Diagrama de transición de estados

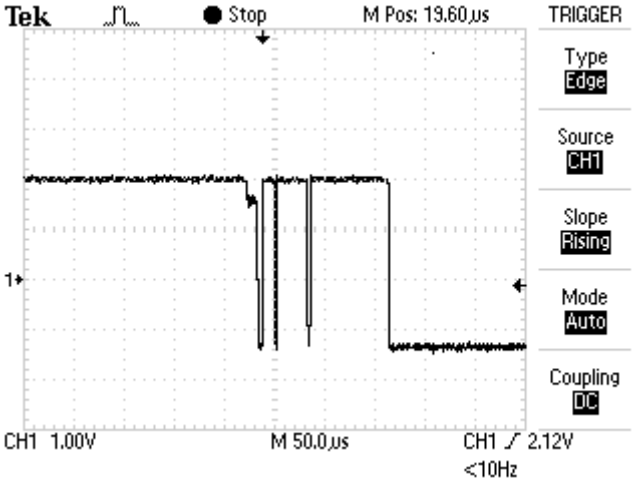


Detección de cambio de estado en un PULSADOR

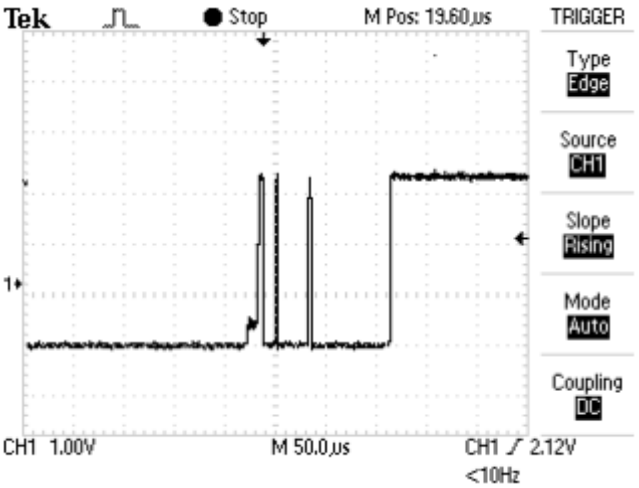
Los pulsadores son dispositivos micromecánicos a los cuales se les aplica presión externa que produce la deflexión de un circuito que se encuentra en su interior. Existen pulsadores que en estado de reposo tienen el circuito abierto y otros que lo tienen cerrado.
Al ser dispositivos mecánicos, estos no son ideales, y como tales generan un rebote técnico en la transición de un estado al otro.
Veamos el siguiente circuito de implementación de un pulsador:



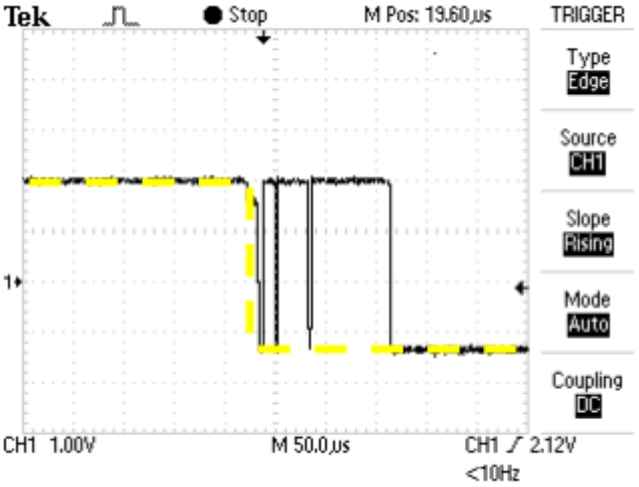
En la imagen vemos la aplicación típica para conectar un pulsador a la entrada de un microcontrolador. Midamos con un osciloscopio sobre la pata de señal.



Como se observa en la imagen capturada de un osciloscopio, en estado de reposo podemos medir un valor de tensión igual a VCC (llamado estado ALTO, 1, HIGH). Cuando se aplica presión sobre el pulsador, el valor deseado a medir es 0V(GND) (llamado estado BAJO, 0, LOW). El inconveniente, se puede apreciar en la transición de un estado al otro, donde se observa el llamado REBOTE (Bounce en Inglés).
Si la entrada a nuestro sistema fuese esa señal, y como evento usamos los cambios de flancos, se estarían generando 7 eventos, 4 corresponden a cambios de “1” a “0” (conocido como flanco descendente - falling) y los 3 restantes a cambios de “0” a “1” (conocido como flanco ascendente - rising).
Lo mismo se observa cuando se libera el pulsador:



Para resolver este “inconveniente físico” se propone diseñar un sistema que elimine ese rebote (sin usar electrónica adicional), para lo cual al sistema se le debe adicionar una entrada o evento que será un “tiempo”, un tiempo de inmunidad entre la detección del primer flanco descendente y la determinación del estado que tomó la señal. Un ruido (señal de poca duración) puede generarnos un falso evento, este sistema evita que se sucedan falsas salidas.
La salida esperada del sistema sería:



Para lograr dicha salida al sistema, como se mencionó antes, se debe agregar una entrada o evento de tiempo que será usando de tiempo de inmunidad a los cambios.

Resolución por método de MOORE

Identificación de entradas y salidas

Entradas

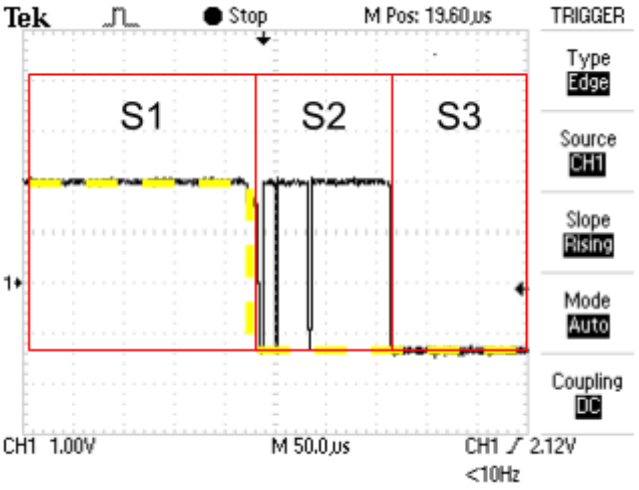
- **P**ulsador, entrada externa al sistema
- **T**iempo, inmunidad interna del sistema.

Salidas

- **E**stado **P**ulsador, variable que nos indica el estado del pulsador

Estados del sistema

Analizando la imagen, podemos definir 3 estados:



Estados

- Estado 1 (S1): **LIBRE**, No hay pulsador presionado.
- Estado 2 (S2): **DEMORA**, Demora o inmunidad.
- Estado 3 (S3): **PULSADO**, Hay pulsador presionado

Transición de estados

- S1 – **LIBRE** → Transición a S2 cuando P = 0 (Pulsador presionado)
- S2 – **DEMORA** → Transición a S3 cuando P = 0 y t > 30 mSeg (Demora o inmunidad)
- S3 – **PULSADO** → Transición a S1 cuando P = 1 (Pulsadora no presionado)

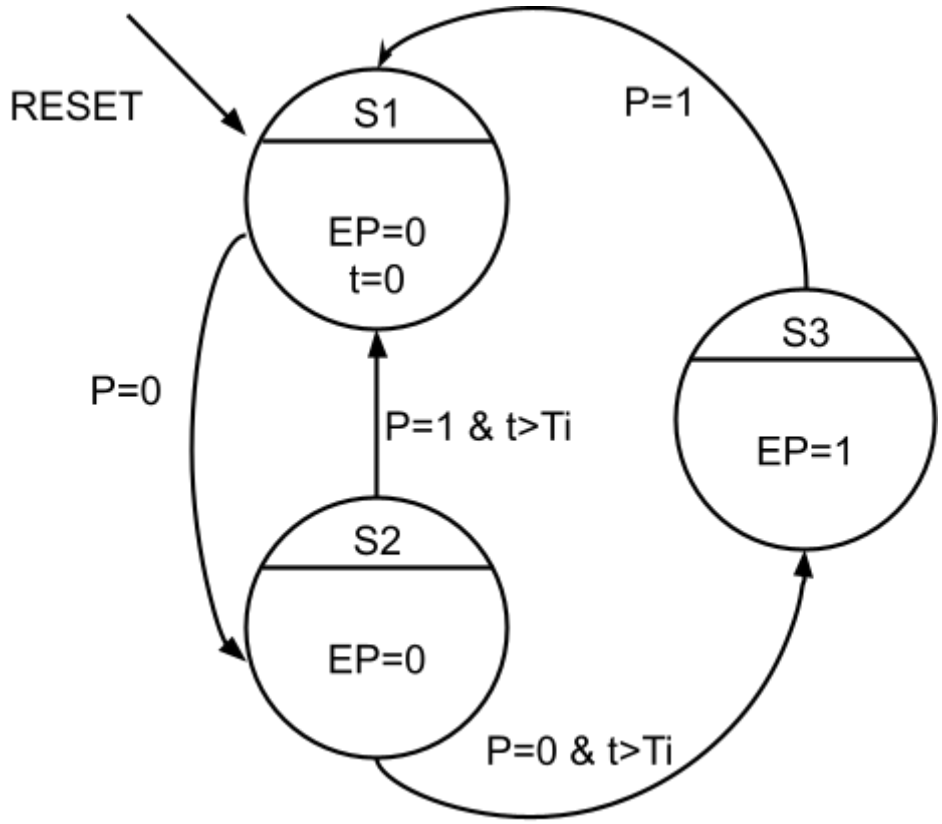
Salidas de cada Estado

- S1 – **LIBRE** → **E**stado **P**ulsador = 0 (NO HAY PULSADOR PRESIONADO)
- S2 – **DEMORA** → **E**stado **P**ulsador = 0 (NO HAY PULSADOR PRESIONADO)
- S3 – **PULSADO** → **E**stado **P**ulsador = 1 (HAY PULSADOR PRESIONADO)

Análisis

- La salida solo depende del estado actual (condición de Moore).
- El estado siguiente depende del estado actual y las entradas (pulsador y tiempo) (condición de Moore y Mealy).

Diagrama de transición de estados



Resolución por método de MEALY

Identificación de entradas y salidas

Entradas

- **P**ulsador, entrada externa al sistema
- **T**iempo, inmunidad interna del sistema.

Salidas

- **E**stado **P**ulsador, variable que nos indica el estado del pulsador

Estados del sistema

- Estado 1 (S1): **LIBRE**, No hay pulsador presionado.
- Estado 2 (S2): **PULSADO**, Hay pulsador presionado

Transición de estados

- S1 – **LIBRE** → Transición a S2 cuando P = 0 y t > 30 mSeg (Demora o inmunidad)
- S2 – **PULSADO** → Transición a S1 cuando P = 1 y t > 30 mSeg

Salidas de cada Estado

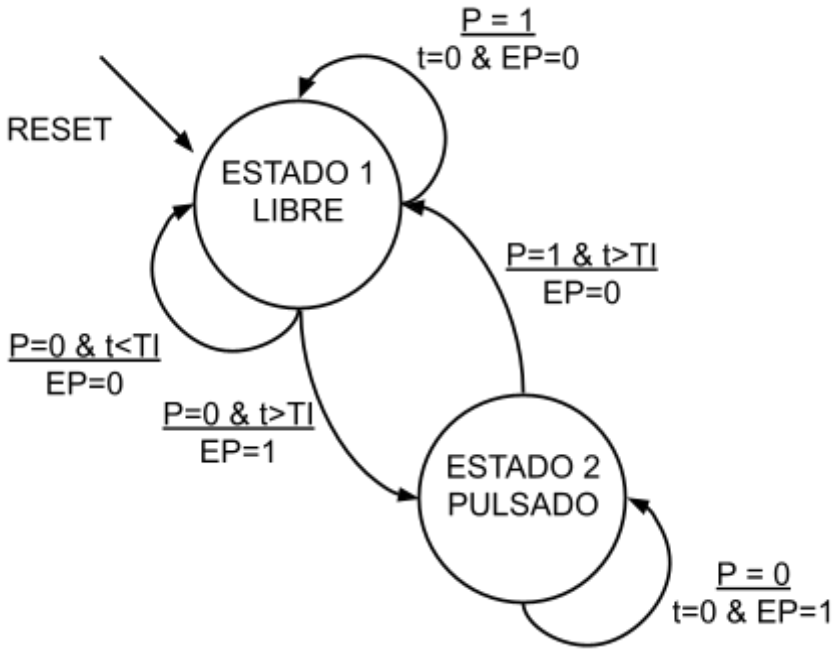
- S1 – **LIBRE** → **E**stado **P**ulsador = [**“0”**→ P=1 o (P=0 t < Ti)] [**“1”** si (P=0 y t > Ti)]

- S2 – **PULSADO** → **E**stado **P**ulsador = [**“1”**→ si P=0 o (P=1 t < Ti)] [**“0”**→ (P=1 t > Ti)]

Análisis

- La salida solo depende del estado actual y la entrada (condición de Mealy).
- El estado siguiente depende del estado actual y las entradas (pulsador y tiempo) (condición de Moore y Mealy).

Diagrama de transición de estados

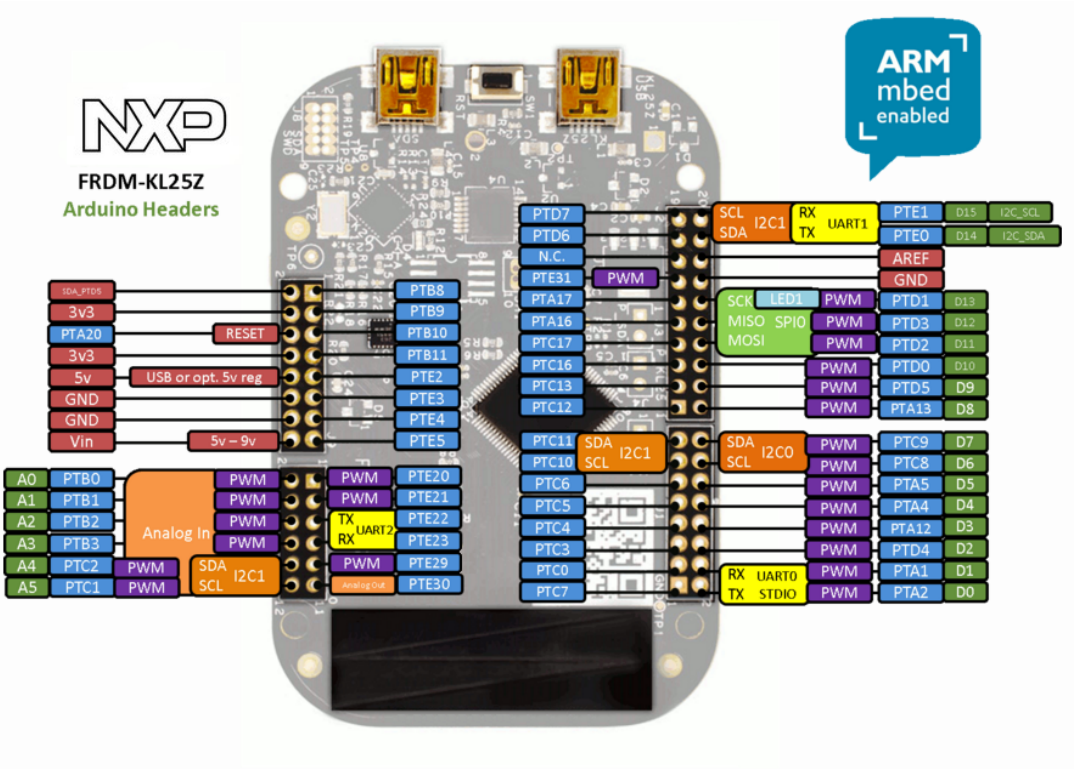


IMPLEMENTACIÓN EN MBED https://ide.mbed.com/compiler/

La biblioteca mbed proporciona la plataforma de software C / C ++ y las bibliotecas para construir sus aplicaciones.
En el siguiente enlace pueden encontrar todas las definiciones del API (inglés Application Programming Interface) https://os.mbed.com/handbook/Homepage

Placa de desarrollo

Vamos a trabajar usando una placa de desarrollo de NXP, FRDM-KL25.
Todas las plataformas FREEDOM son Arduino Header compatibles. Circuito Esquemático



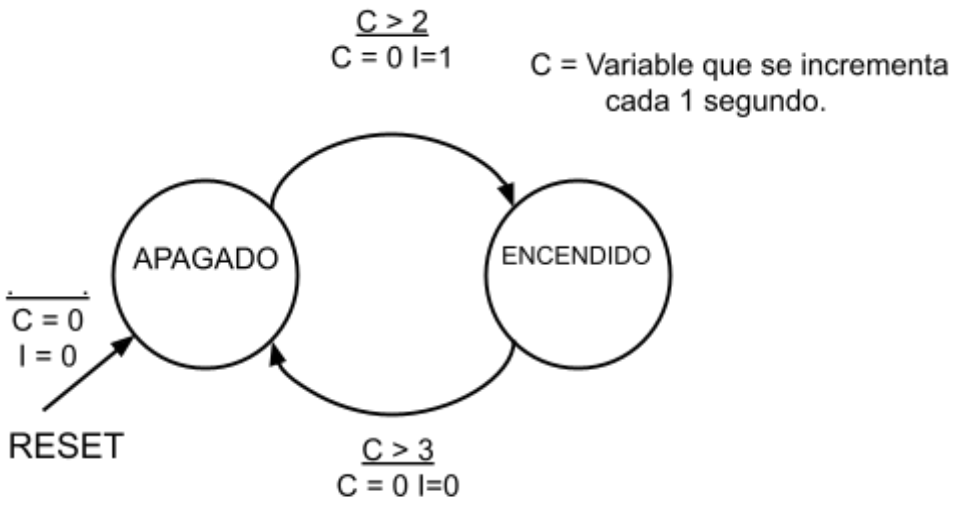
Así como mbed nos provee de una API para construir las aplicaciones, también nos provee las definiciones con nombres y asociaciones a los pines físicos del microcontrolador. El listado completo lo pueden encontrar: frdm kl25z pinnames -

Algunas definiciones generales a tener en cuenta

- DigitalOut - Configure and control a digital output pin.
- Ticker - Repeatedly call a function

Intermitencia de una señal luminosa

Basándonos en el ejemplo de más arriba vamos a trabajar con el diagrama propuesto para la intermitencia.



Entradas

En este caso la entrada está definida por una variable..
La definimos de tipo `unsigned char` dado que el valor máximo esperado es 3

```
3 //Entradas o eventos del sistema
4 unsigned char C = 0;
```

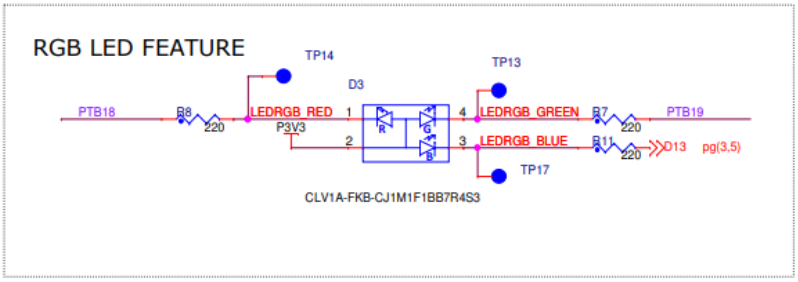
La variable C, es la variable que vamos a usar para contar tiempo. Para contar tiempo mbed cuenta con varias opciones, en este caso vamos a elegir el Ticker

Salidas

Creamos la salida del sistema. En este caso, la salida del sistema está dada por uno de los diodos leds que están montados en la placa de trabajo (FRDM-KL25Z)

```
6 //Salida del sistema
7 DigitalOut led_rojo(LED1);
```

Del Circuito Esquemático se puede observar que los led's montados en la placa se los enciende cuando ponemos un estado LOW (GND) en el pin asociado.



Uso del ticker

La interfaz Ticker (haciendo click los lleva a un ejemplo) se usa para configurar una interrupción recurrente para llamar repetidamente a una función a una velocidad especificada.

Creamos el objeto:

```
//Interrupcion recurrente
Ticker nombre_del_ticker;
```

Inicializamos el objeto:

Ya dentro de nuestro programa, en la sección de inicializaciones (aquellas cosas que solo se van a ejecutar una sola vez) creamos el objeto y lo configuramos:

```
int main() {
    //Inicializaciones
    nombre_del_ticker.attach(&funcion, 1.0);

    //lazo o loop infinito
    while(1){

    }
}
```

El primer parámetro que se pasa, en este caso **`&funcion`** , es la dirección (**&** delante del nombre de una función o variable retorna la dirección) de la función a la cual se va a llamar cada, segundo parámetro, `1.0` segundos.
Recordar: La función **`funcion`** debe estar declarada por encima de donde la vamos a utilizar.
Dentro **`funcion`** de vamos a incrementar en 1 la variable C.

```
//funcion con llamado recurrente
void funcion (){
    C++; //C = C + 1; C += C;
}

int main() {
    //Inicializaciones
    nombre_del_ticker.attach(&funcion, 1.0);

    //lazo o loop infinito
    while(1){

    }
}
```

Función máquina de estado:

Ahora vamos a crear la función máquina de estado. Por el momento vamos a trabajar con switch-case, donde cada `case` será un estado.

Definición de estados:

Para definir los nombres de los estados vamos a usar la directiva de pre-compilación `#define` de manera de mantener orden en la creación de estados futuros. Recordemos que este sistema tenemos 2 estados posibles.

```
3 //declaracion de estados
4 #define APAGADO 0
5 #define ENCENDIDO 1
```

Creación de la función máquina de estados:

Creamos una función, la cual no va recibir parámetros ni devolver valores, dentro de la función creamos una variable que vamos a llamar `estado`

```
21 void maquina_estado(){
22     static unsigned char estado = APAGADO;
23 
24 }
```

La variable estado fue definida del tipo `static` lo que significa que la variable se comportara como una “variable global” pero solamente será accesible en el ámbito de la función `maquina_estado` , lo que significa que nadie fuera de la función podrá cambiar el valor.

> *Si una variable estática está declarada en una función, sólo será accesible desde esa función y mantendrá su valor entre ejecuciones de la función. Este comportamiento es contra intuitivo porque estas variables se declaran en el mismo lugar que el resto de variables en una función, pero mientras que estas adquieren valores nuevos con cada ejecución, las estáticas conservan estos valores entre ejecuciones.*

Ahora codificamos cada uno de los case.

```
21 void maquina_estado(){
22     static unsigned char estado = APAGADO;
23 
24     switch(estado)
25     {
26         case APAGADO:
27 
28         break;
29 
30         case ENCENDIDO:
31 
32         break;
33     }
34 }
```

Nos queda definir de qué manera se hace el inicio o RESET de la máquina de estado, observando en el diagrama que creamos, el RESET nos deja posicionados en el estado APAGADO. Para definir el “estado” de RESET, vamos a usar el caso `default` de los

switch-case, lo que significa que se va a ejecutar el estado `default` si no es ninguno de los definidos. Cada vez que entremos al caso `default` vamos a ir al estado APAGADO.

```
21 void maquina_estado(){
22     static unsigned char estado = APAGADO;
23 
24     switch(estado)
25     {
26         default:
27             estado = APAGADO;
28             break;
29 
30         case APAGADO:
31             break;
32 
33         case ENCENDIDO:
34             break;
35     }
36 }
37 
```

case APAGADO:

Cómo definimos con anterioridad S1 – **APAGADO**→ Transición a S2 cuando C > 2 seg.

```
30         case APAGADO:
31             //entrada
32             if( C > 2 )
33             {
34                 //salidas
35                 led_rojo = 0;    //pin a GND = LED ENCENDIDO
36                 C = 0;
37                 //transicion
38                 estado = ENCENDIDO;
39 
40             }
41             break;
42 
```

case ENCENDIDO:

Cómo S2 – **ENCENDIDO** → Transición a S1 cuando C > 3 seg.

```
43         case ENCENDIDO:
44             //entrada
45             if( C > 3 )
46             {
47                 //salidas
48                 led_rojo = 1;    //pin a VCC = LED APAGADO
49                 C = 0;
50                 //transicion
51                 estado = APAGADO;
52 
53             }
54             break;
```

Llamado (invocar) función máquina de estados:

Ahora lo que nos resta hacer, es hacer que nuestro programa principal invoque a la función de la máquina de estados que queremos ejecutar. Para hacer esto vamos a colocar un llamado a la función en la sección de loop infinito.

```
58 int main() {
59     //Inicializaciones
60     nombre_del_ticker.attach(&funcion, 1.0);
61 
62     //lazo o loop infinito
63     while(1){
64         maquina_estado();
65     }
66 }
67 
```

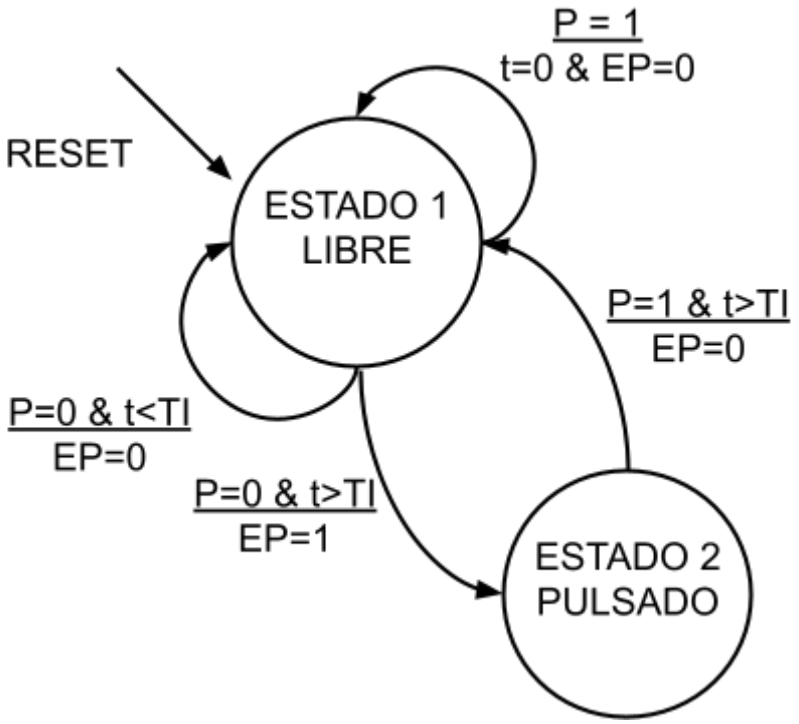
Compilamos

y probamos.

Link al ejemplo.

Detección de cambio de estado en un PULSADOR eliminando rebote

Basándonos en el ejemplo de más arriba vamos a trabajar con el diagrama propuesto para la detección del cambio de estado de un pulsador e implementar la programación del código.
Para la codificación vamos a usar la resolución por el método de Mealy. El diagrama propuesto fue el siguiente:



Entradas

La entrada está definida por un factor externo **P**, un pin usado como entrada digital (DigitalIn) al cual se le conecta un pulsador con la configuración más comúnmente usada (Pullup) y una variable de tiempo que llamaremos **t**.

```
3 //Entradas o eventos del sistema
4 DigitalIn P(PTA5);  //Entrada digital conectada a PTA5
5 unsigned char t = 0;
```

Al igual que en ejemplo anterior vamos a definirla como `unsigned char`.

Salidas

A diferencia del ejemplo anterior, en este, para la salida del sistema vamos a usar una variable, la cual vamos a nombrar **EP** (**E**stado **P**ulsador)

```
7 //Salida del sistema
8 unsigned char EP = 0;
```

Al igual que en ejemplo anterior, dado que la variable t la vamos a usar para contar tiempo, debemos crear e inicializar un objeto de ticker.
En este caso el contador de tiempo lo vamos a utilizar para filtrar el efecto de rebote que se produce en la entrada digital de nuestro sistema, un tiempo en el orden de los 30 a 50 milisegundos esta correcto para filtrar. (Para nuestro ejemplo vamos a usar 30).

Selección del periodo del Ticker

No es algo simple de explicar, la idea es buscar el tiempo máximo común a todos los contadores de tiempos que van a ser regidos por el mismo ticker, pero a su vez buscar tener precisión o resolución en la cuenta mínima que vamos a usar.
Nuestro caso va tener un ticker para contar 30 milisegundos, vamos a elegir como base de tiempo 10 milisegundos, de forma de tener un resolución de 3 cuentas para el tiempo buscado.

```
10 //Interrupcion recurrente
11 Ticker ticker_debounce;
12
13 //funcion con llamado recurrente
14 void timer_debounce (){
15     t++;
16 }
17
18 int main() {
19     //Inicializaciones
20     ticker_debounce.attach(&timer_debounce, 0.01);
21
```

Función máquina de estado:

Ahora vamos a crear la función máquina de estado. Por el momento vamos a trabajar con switch-case, donde cada `case` será un estado.

Definición de estados:

Para definir los nombres de los estados vamos a usar la directiva de pre-compilación `#define` de manera de mantener orden en la creación de estados futuros. Recordemos que este sistema tenemos 2 estados posibles.

```
3 //declaracion de estados
4 #define PULS_LIBRE 0
5 #define PULS_PRESIONADO 1
```

Creación de la función máquina de estados:

Creamos una función, la cual no va recibir parámetros ni devolver valores, dentro de la función creamos una variable que vamos a llamar `estado`

case PULS_LIBRE:

Cómo definimos con anterioridad S1 – **LIBRE**→ *Transición a S2* cuando P = 0 y t > 30mSeg (Demora o inmunidad), Salida **EP** = [**"0"**→ P=1 o (P=0 t < Ti)] [**"1"** si (P=0 y t > Ti)]

```
36          case PULS_LIBRE:
37              if( P==0 & t>3 )
38              {
39                  //transicion
40                  estado = PULS_PRESIONADO;
41                  //salida
42                  EP = 1;
43                  printf("PULSADOR PRESIONADO\r\n");
44              }
45              else
46              {
47                  //nos quedamos en PULS_LIBRE (S1)
48                  //salida
49                  EP = 0; //ambas salidas al S1 Hacen EP = 0
50                  if( P == 1)
51                      t = 0;  //Solo hacemos t=0 cuando P=1
52              }
53              break;
```

case PULS_PRESIONADO:

Como vimos antes el S2 – **PULSADO** → Transición a S1 cuando P = 1 y t > 30mSeg, Salida **EP** = [**"1"**→ si P=0 o (P=1 t < Ti)] [**"0"**→ (P=1 t > Ti)]

```
55          case PULS_PRESIONADO:
56              if( P==1 & t>3 )
57              {
58                  //transicion
59                  estado = PULS_LIBRE;
60                  //salida
61                  EP = 0;
62                  printf("PULSADOR LIBRE\r\n");
63              }
64              else if( P == 0 )
65              {
66                  //nos quedamos en PULS_PRESIONADO (S2)
67                  //salida
68                  EP = 1;
69                  t = 0;
70              }
71              break;
```

Codificamos y compilamos

y probamos, la variable EP es quien se queda con el valor del estado del pulsador.

Definiciones para la máquina de estados

Así como hemos definido los valores de los estado con la directiva de pre-compilación #define, podemos hacer lo mismo para los tiempos.

```
7 #define DEBOUNCE_TIEMPO_BASE 0.01   //10mseg
8 #define DEBOUNCE_TIEMPO_FILTRO 0.03   //10mseg
9 #define DEBOUNCE_CONTADOR_FILTRO (DEBOUNCE_TIEMPO_FILTRO / DEBOUNCE_TIEMPO_BASE)
```

Donde **DEBOUNCE_TIEMPO_BASE** es el tiempo base para el ticker (10mSeg), **DEBOUNCE_TIEMPO_FILTRO** es el tiempo de filtrado de la señal digital de entrada (30mSeg) y **DEBOUNCE_CONTADOR_FILTRO** es la cuenta que tiene que llegar la variable t para determinar que pasaron los 30mSeg deseados para el filtro.

NOTA

No se olviden de conectar en el PTA5 (o el pin que deseen) el circuito propuesto. Si cambian de PIN deben definirlo acá