Trabajo Práctico 2

16.05.2017

Matías Pan - 783/9
Agustín Sánchez - 939/2
Jorge Stranieri - 917/5

Planteo general

Implementar con el MCU una cerradura electrónica. Para esto se dispone de un display LCD de 2 líneas, un teclado matricial 4x4 y el HS08. La implementación deberá hacerse con máquinas de estados temporizadas con el RTC.

Interpretación

Utilizando el microcontrolador MC9S08SH8 CPJ que provee el kit, una pantalla LCD y un teclado matricial de cuatro por cuatro, realizaremos varios ejercicios donde se explicaran las conexiones, configuración y programación necesaria.

En cada ejercicio procederemos a explicar detalladamente los objetivos, el hardware que se va a utilizar, los esquemas de conexiones y los algoritmos necesarios para resolver lo pedido.

Para la resolución de dichos ejercicios, se optó modularizar el proyecto en librerías independientes que cumplen las funciones requeridas.

Se tomó en cuenta la memoria limitada del microcontrolador, por lo que priorizamos utilizar variables cuyo tamaño sea el menor posible.

Estrategia de resolución

Para llevar a cabo este sistema, implementamos una máquina de estados finita temporizada con el RTC(Real Time Counter), el cual nos permite generar una interrupción periódica.

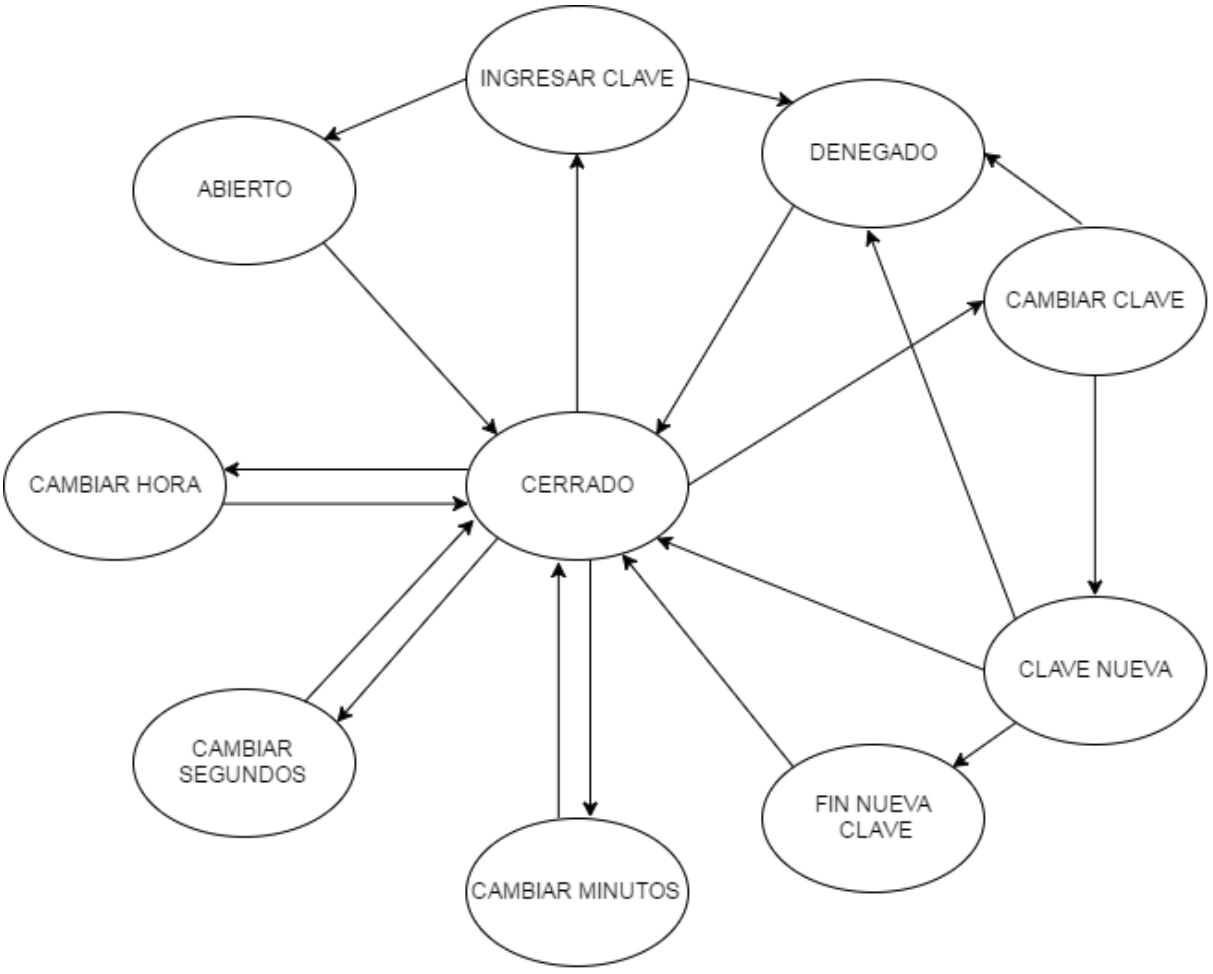
Dado que necesitamos ejecutar varias funciones, como leer de teclado, actualizar la máquina de estados y mantener el reloj y los timeouts, dentro de la rutina de interrupción, se buscó el período máximo requerido para poder atenderlas. Dicho período resultó ser 100 ms (milisegundos).

Como la función encargada de actualizar el reloj se debe ejecutar cada un segundo(a diferencia del resto que es cada 100 ms), utilizamos un contador de 10 con el fin de que dicha función sea llamada con el periodo correspondiente.

Para definir el comportamiento de nuestro sistema, vamos a implementar una máquina de estados finito de tipo Moore, en la cual las transiciones de estados van a depender principalmente de las teclas presionadas y diferentes timeouts. Mientras que las salidas van a ser lo que se imprima en la pantalla LCD.

Para implementar la máquina de estados se utilizó un arreglo de punteros a funciones.

Cada estado y transición serán explicados a lo largo del trabajo práctico y mostramos a continuación una visión general de la máquina de estados diseñada:



Inciso A

Consigna

Cuando el equipo se inicia deberá mostrar en la primer línea del LCD un reloj funcionando con el formato HH:MM:SS(horas, minutos y segundos) y en la segunda línea el estado de la cerradura "CERRADO".

Interpretación

Debemos implementar un reloj que indique la hora en el formato HH:MM:SS, el cual va a ser mostrado en la pantalla LCD provista por la cátedra. Ésta está compuesta de dos filas, donde cada fila tiene 16 caracteres. Para la utilización de dicha pantalla la cátedra nos otorgó una librería, la cual nos abstrae del funcionamiento interno y nos provee un conjunto simple de funciones.

Resolución

Tener en cuenta que para este inciso no es necesario implementar la máquina de estados mencionada en el estrategia de resolución.

Inicialmente configuramos el RTC del microcontrolador para lograr el mecanismo de interrupción previamente mencionado. Se utiliza por defecto el clock interno de 1 kHz. Luego, configuramos el *Prescaler*, el cual actúa como escala con relación a la frecuencia del timer para indicar una unidad de cuenta, y el *RTC Modulo Value* que es el número de unidades que va a tomar para activar la interrupción. En nuestro caso utilizamos un *Prescaler* de 100 y un *RTC Modulo Value* de 0. Entonces como cada ciclo del clock lleva 1 ms el periodo va a ser de 100 * 1 ms = 100 ms. Se podría haber logrado el mismo periodo utilizando por ejemplo un *Prescaler* de 1 y un *RTC Modulo Value* de 99.

Una vez configurado habilitamos las interrupciones, *RTC Interrupt = Enabled*.

Por lo tanto, cada 100 ms se va a ejecutar la función definida dentro del **MCU_init.c:**

__interrupt void isrVrtc(void).

Dentro de esta función estarán los llamados a las rutinas que queremos llamar cada 100 ms. Para el caso del reloj, que sabemos que se debe actualizar cada un segundo, mantuvimos un contador interno que cada diez interrupciones llame a la función correspondiente.

Para implementar el reloj realizamos una librería llamada ***clock.*** Esta contiene tres variables correspondientes a las horas, los minutos y segundos. Para el funcionamiento del reloj implementamos una función denominada ***tick***, la cual será llamada cada un segundo como se explicó previamente, encargada de actualizar el reloj con los chequeos correspondientes a cada campo. Para obtener la hora en el formato pedido, implementamos una función denominada ***get_time_as_str.*** Y para modificar la hora hicimos una función ***cambiar_hora(hh,mm,ss)***, que recibe por parámetro los valores de los campos que se deseen cambiar.

Las terminales de la pantalla LCD se deben conectar a los puertos A y C del microcontrolador como lo especifica la librería ***LCD***.

Como se nos explico en clase, la pantalla LCD utiliza un protocolo de inicialización específico. Para su correcta inicialización llamamos a la función **LCD_init** con los siguientes parámetros:

- DISPLAY8X5 | _2_LINES
- DISPLAY_ON | CURSOR_OFF

El primer parámetro indica que el tamaño de cada carácter va a ser de 8x5 y que la pantalla va a tener un total de dos líneas.

El segundo parámetro indica que el display va a estar prendido y que el cursor no va a ser visible.

Para poder manipular la pantalla LCD realizamos una librería, denominada ***pantalla***, la cual funciona como interfaz entre nuestro programa y la librería ***LCD*** otorgada por la cátedra. Tenemos dos strings de 16 caracteres correspondientes a cada fila de la pantalla. Para inicializar la pantalla realizamos la función ***pantalla_init*** la cual inicializa el LCD con los parámetros correspondientes y limpia la pantalla. Además, implementamos un ***pantalla_update*** que imprime en el LCD los strings almacenados. Finalmente, para solo asignar valores a los strings que serán impresos realizamos una función denominada ***setear_string*** la cual recibe por parámetro la fila correspondiente y el string, el cual será alineado en el centro.

Nuestro pseudo codigo del "main":

```
Inicializar microcontrolador
Inicializar pantalla
setear_string("CERRADO", 1); // 0=primera fila, 1=segunda fila
loop
      get_time_as_str(hora)
      setear_string(hora,0);
      pantalla_update();
end
```

Dado que en este inciso "CERRADO" se muestra de manera estática, lo seteamos una única vez fuera del loop.

Luego, en cada iteración del loop, obtenemos la hora actual(puede que no haya cambiado desde la iteración previa) y la imprimimos en pantalla.

Inciso B

Consigna

El sistema debe tener la clave por defecto 1234 de manera de poder activar o desactivar la cerradura. Si el usuario presiona la clave correcta se mostrará el estado "ABIERTO" durante 5 segundos y luego volverá automáticamente al estado por defecto. Si la clave es incorrecta se mostrará el estado "DENEGADO" durante 2 segundos y luego volverá automáticamente al estado por defecto. No se mostrarán en el LCD las teclas presionadas.

Interpretación

Para este paso de la cerradura electrónica debemos setear por defecto una contraseña, para que posteriormente pueda ser chequeada cuando se ingresen contraseñas por teclado. Debido a esto pueden pasar dos cosas que implica cambiar el estado en el que se encuentra la máquina de estados. Por defecto está en el estado cerrado, y sólo

entrará al estado INGRESAR_CLAVE si se presiona un número del teclado. En este estado se muestra por pantalla la contraseña que se va ingresando representadas por "*", hasta que se indique la finalización del ingreso de la contraseña presionando la tecla "D". El largo de la contraseña ingresada puede tener un máximo de 16 caracteres. Si la contraseña se ingresa correctamente hay una transición de estado de INGRESAR_CLAVE a ABIERTO; Justamente la cerradura electrónica se abriría. Si no se ingresa correctamente hay una transición de estado de INGRESAR_CLAVE a DENEGADO, denegando el acceso.

Resolución

Los 8 terminales del teclado matricial se deben conectar al puerto B del microcontrolador como lo recomendó la cátedra. Cuatro de ellos corresponden para las filas y los restantes cuatro para las columnas.

El método que utilizamos para detectar qué tecla se presiono es el siguiente:

- los terminales correspondientes a las filas se utilizaron como salidas del microcontrolador y los de las columnas se utilizaron como entradas, con las resistencias de pull-up habilitadas.
- Previo a presionar una tecla, las filas deben estar configuradas en 0. En cada interrupción se ejecutará una rutina que comprobará si hay alguna tecla presionada. Cada tecla consiste de un botón que al ser presionado conecta los terminales de su fila y columna, lo cual genera que la columna pase de tener 1(el valor por defecto) a tener 0, lo que indique que fue presionada.
- Una vez detectado el cambio de estado de una columna, debemos realizar un barrido cuyo objetivo es saber cual de todas las filas fue presionada. Para llevar a cabo esto, se ponen todas las filas en 1 a excepción de la que está siendo barrida. Por cada fila se pregunta si la columna en cuestión mantuvo el valor 0, lo cual significa que esa fila y columna corresponden a la tecla presionada.
- Una vez obtenida la tecla presionada, se restablecen los valores por defecto para poder detectar la siguiente tecla.

Para utilizar el teclado implementamos una librería denominada ***teclado***. Ésta define las siguientes funciones: ***teclado_init*** encargada de inicializar el puerto B y ***comprobar_tecla*** la cual será llamada dentro de la rutina de interrupción ***isVrtc***(cada 100 ms), e implementa el algoritmo de barrido mencionado previamente, además de marcar únicamente el evento de presión de tecla una vez que esta haya sido liberada. Esto se implementó así para resolver posibles efectos rebotes.

Para llevar un registro de dichos eventos implementamos un buffer con capacidad para un único evento en una librería llamada ***eventotecla.*** Ésta actúa como interfaz entre el teclado y la entidad que quiera consumir los eventos. En nuestro caso, esta entidad será la máquina de estados la cual utilizará estos eventos para interactuar entre sus diferentes estados y funcionalidades.

Para llevar a cabo lo que pide el inciso vamos a explicar los estados de la MEF correspondientes: **CERRADO, INGRESAR_CLAVE, ABIERTO** y **DENEGADO**, junto con sus transiciones y salidas.

Definimos la máquina de estados finitos en la librería ***mef***, la cual se implementó como un arreglo de punteros a las funciones correspondientes de cada estado y un campo reservado indicando el estado actual. Contamos con una función ***MEF_update*** que llama a la función correspondiente a la del estado actual. Dichas funciones poseen el formato de ***f(nombre_estado)***, por lo que en el caso de este inciso las rutinas a analizar serán ***fCERRADO***, ***fINGRESAR_CLAVE***, ***fABIERTO*** y ***fDENEGADO***.

MEF_update es llamada dentro de la rutina de interrupción de modo que también tiene periodo 100 ms.

Estado cerrado

Cuando el sistema se encuentra en este estado, en la pantalla se verá la hora, en la primer fila, y el string “CERRADO” en la segunda fila.

Para lograr esto se utilizan las funciones ***get_time_as_str***, de la librería ***clock***, y ***setear_string***, de la librería ***pantalla***, para obtener la hora y enviarla al buffer de salida del LCD. De la misma forma se setea “CERRADO” utilizando como parámetro la fila correspondiente (primer fila=0, segunda fila=1).

A partir de este estado, y teniendo en cuenta sólo lo pertinente a este inciso, la única transición del estado CERRADO es hacia el estado INGRESAR_CLAVE. Esta transición ocurre cuando se toca en el teclado cualquier tecla numérica, lo que se reconoce como un intento para ingresar la clave del sistema de alarma.

Estado ingresar clave

El estado INGRESAR_CLAVE seguirá informando en la primer fila de la pantalla la hora actual, como así también un asterisco por cada dígito de la contraseña ingresada.

Como decidimos que la contraseña del sistema de alarma debería ser flexible, de 1 a 16 caracteres, agregamos la funcionalidad de que cuando se termina de ingresar la clave se deberá presionar la tecla D.

El ingreso de la clave lo implementamos con una función llamada ***leer_clave***, la cual irá almacenando internamente las teclas presionadas, con las verificaciones necesarias. Esto lo realiza consumiendo los eventos del buffer, implementado en ***eventotecla***, correspondientes al teclado.

Una vez terminado el ingreso de la clave, osea presionada la tecla D, se compara con la clave del sistema actual almacenada en la misma mef. El resultado de esta comparación será la que determine las transiciones de estado, por lo cual tenemos las transiciones:

- INGRESAR_CLAVE → ABIERTO: cuando la clave ingresada es correcta.
- INGRESAR_CLAVE → DENEGADO: cuando la clave es incorrecta.

Estado denegado y estado abierto

Ambos estados se comportan de manera similar: imprimirán en la primer fila la hora actual, y en la segunda “DENEGADO” o “ABIERTO” según corresponda. Se mantendrán en ese estado por cierto periodo de tiempo, 2 segundos en el caso de DENEGADO y 5 en el caso de ABIERTO. Al finalizar, ambos estados volverán al estado CERRADO.

Para realizar el conteo del tiempo dentro de nuestro programa, y en el caso de este inciso para resolver la espera de 2 y 5 segundos en estos dos últimos estados, implementamos la librería ***timeout.*** Está liberia presenta la siguiente funcionalidad: con la función ***timeout_empezar(parámetro)*** podremos establecer un “timer” con el contador que le asignamos mediante el parámetro con el que es llamado. Una vez inicializado el contador, se utiliza la función ***timeout_termino*** la cual devuelve si ya transcurrió el tiempo con el que fue inicializado el timer. Dentro de la implementación la unidad del parámetro está en décimas de segundos, por lo que si queremos esperar 5 segundos, el valor de parámetro tiene que ser igual a 50 .

Para resolver esto, ***timeout*** posee un contador interno que se incrementará cada 100 ms. Esto lo lleva a cabo realizando una llamada a la función ***timeout_contar*** dentro de la rutina de interrupción ***isVrtc*** programada anteriormente***.***

Contando con esta librería, los estados ABIERTO y DENEGADO inicializan por única vez el timeout con 2 o 5 segundos según corresponda. Luego preguntaran constantemente hasta que el tiempo haya transcurrido para cambiar el estado actual de la máquina de estados a CERRADO.

Inciso C

Consigna

Para modificar la hora del reloj se deberán introducir los campos HH, MM y SS por separado, siguiendo el siguiente procedimiento: Presionar ‘A’ para establecer los dos dígitos de la hora. No se puede borrar (los últimos dos dígitos ingresados serán los válidos). Para terminar, volver a presionar ‘A’. De la misma manera se utilizarán las teclas ‘B’ y ‘C’ para modificar los minutos y los segundos respectivamente.

Interpretación

El sistema de cerradura electrónica dispondrá de un opción para cambiar el horario de la misma. Si se quiere cambiar la hora, se debe apretar la tecla ‘A’, si se quieren cambiar los minutos, se debe apretar la tecla ‘B’ y si se quieren cambiar los segundos se debe apretar la tecla ‘C’. Una vez apretadas alguna de estas teclas, se realizará la transición de estados correspondiente. En cada uno de ellos se hacen las verificaciones necesarias para que no se pueda ingresar un horario incorrecto. Se necesita que se al menos dos dígitos para el cambio de algún valor. Durante el ingreso se ignoraran las teclas que no sean relevantes para este estado.

Resolución

En este inciso implementamos los estados ***CAMBIAR_HH***, ***CAMBIAR_MM*** y ***CAMBIAR_SS.*** Como estos 3 estados se comportan de forma casi idéntica, con la explicación de un solo estado se definen los otros dos estados de forma análoga. Por lo que procederemos a desarrollar el comportamiento solamente del estado CAMBIAR_HH.

Estado cerrado

Ahora al estado cerrado se le agregan 3 transiciones:

- CERRADO → CAMBIAR_HH: al presionar la tecla A.
- CERRADO → CAMBIAR_MM: al presionar la tecla B.
- CERRADO → CAMBIAR_SS: al presionar la tecla C.

Estado cambiar hora

La salida en pantalla de este estado consiste en mostrar en la primer fila la hora actual, y "CAMBIANDO HORA" en la segunda fila. Para el caso de los otros estados sería "CAMBIANDO MIN" y "CAMBIANDO SEG".

Solo se realizará el cambio de estado nuevamente a CERRADO, cuando se vuelva a ingresar la tecla A que indica la finalización del cambio de hora.

La función dentro de ***mef*** encargada de definir el comportamiento de este estado será ***fCAMBIAR_HH***. Ésta utiliza ***setear_string*** para definir las salidas como se define más arriba y utiliza tambien ***eventotecla*** para ir detectando las teclas presionadas. Dichas teclas se verifican para que tengan un valor coherente en la posición que se está cambiando. En el caso del cambio de hora, el primer dígito nunca podrá tener un valor máximo a 2. Si el primer dígito es 0 o 1, el segundo podrá ser de 0 a 9, pero si es 2, el segundo podrá ser de 0 a 3.

Para el caso de los minutos y los segundos, el primero puede ser de 0 a 5, y el segundo de 0 a 9, por lo que tenemos menos chequeos.

El ingreso de los datos se resuelve de manera que solo se toman como válidos los últimos dos dígitos ingresados. Cuando se aprete la tecla para realizar la confirmación del cambio de la hora, se llama a la función ***cambiar_hora***, explicada en incisos anteriores, de la librería ***clock***.

Cabe aclarar que se va permitir la transición al estado CERRADO, cuando se hayan apretado al menos 2 dígitos válidos para modificar la hora, minutos o segundos, según corresponda.

Inciso D

Consigna

Mientras la modificación de algunos de los campos del reloj se esté ejecutando, los dígitos ingresados se mostrarán en el display "parpadeando" una vez cada medio segundo hasta que el comando finalice.

Interpretación

Trabajando sobre lo realizado en el inciso anterior, deberemos implementar algún método de control para el parpadeo regido por un timeout.

El parpadeo se realiza mediante el intercambio entre el número correspondiente a la posición y un espacio en blanco, cada medio segundo. El string resultante será el impreso en la pantalla LCD.

Resolución

Para la resolución de este inciso se hicieron un par de modificaciones a la implementación del inciso anterior.

En primer lugar para mantener el control del "parpadeo" configuramos de la misma forma para los 3 estados CAMBIAR_HH, CAMBIAR_MM y CAMBIAR_SS. El parpadeo lo resolvimos simplemente con un "flag de parpadeo" que indica si el string que se está mostrando actualmente en pantalla se debe encontrar parpadeando o no. El control de este flag lo hicimos estableciendo un timer de 0.5 segundos por cada estado del parpadeo, con nuestra liberia ***timeout***, de forma indefinida hasta que termine el estado del cambio del reloj.

Lo resolvimos de manera que en la pantalla se muestre el reloj funcionando aún cuando se estén realizando por encima modificaciones a sus campos, para lograr una visión más interactiva.

Para que sea más comprensible la utilización del ***timeout*** en este caso, mostraremos el siguiente pseudo-codigo de la función de parpadear:

```
//control de flagParpadeo

timeout_empezar(5);// si ya se empezó el timer y todavia no
                   finalizo, esta funcion internamente no hace nada

If timeout_termino() = TRUE
      Cambiar flagParpadeo al valor negado
End if

//fin control flagParpadeo
```

```
//parpadeo en pantalla
get_time_as_str(hora)
Sobrescribir los dígitos ingresados hasta el momento
If flagParpadeo = TRUE
      Reemplazo con un espacio en la posición que estoy modificando
End if
setear_string(hora_modificado,0)
//fin parpadeo en pantalla
```

Este funcionamiento está dentro de las tres funciones correspondientes a los estados previamente mencionados, ***fCAMBIAR_HH***, ***fCAMBIAR_MM*** y ***fCAMBIAR_SS***, donde para cada uno se considera el campo adecuado.

Inciso E

Consigna

Para modificar la contraseña se deberá presionar la tecla 'D'. En este modo se deberá mostrar el mensaje "clave actual:" y esperar que ingrese la clave actual. Si la clave es incorrecta se mostrará el estado "DENEGADO" durante 2 seg y luego volverá automáticamente al estado por defecto. Si la clave es correcta se deberá mostrar el mensaje "nueva clave:" y leer cada tecla presionada hasta que se presione 'D'. Se mostrará el mensaje "Fin ingreso nueva clave" 5 seg y se volverá al estado por defecto. A partir de aquí está será la nueva clave para abrir la cerradura. Si comete un error al ingresar la nueva clave y desea cancelar se puede presionar '#' lo que interrumpe el ingreso y vuelve al estado por defecto.

Interpretación

Así como se agrego funciones para cambiar la hora, procederemos a implementar una forma en la que se pueda cambiar la contraseña actual de la cerradura.

En primera instancia se debe presionar la tecla "D", luego el usuario deberá autenticarse ingresando la clave actual del sistema para después, en caso de que la clave ingresada sea correcta, se le permitirá ingresar una clave nueva para la cerradura. La lectura de la nueva clave terminará cuando se presiona la tecla "D".

Entonces, las nuevas transiciones de estado necesarias para poder cambiar la clave de la cerradura serían: de CERRADO a CAMBIAR_CLAVE, si la clave ingresada fue incorrecta se pasaría al estado DENEGADO, caso contrario pasaremos al estado CLAVE_NUEVA donde se le permitirá al usuario ingresar la clave nueva para la cerradura, si toma demasiado tiempo en ingresar la nueva clave se ira al estado CERRADO. En caso de que haya ingresado la clave nueva de manera exitosa, se pasará al estado

FIN_NUEVA_CLAVE, donde se informa por pantalla que la clave ha sido cambiada exitosamente.

Resolución

Para la resolución de este inciso tomamos como base lo hecho en el inciso B, debido a que la funcionalidad requerida para el cambio de la clave es similar.

Los nuevos estados que necesitamos agregar son CAMBIAR_CLAVE, CLAVE_NUEVA y FIN_NUEVA_CLAVE. A continuación explicaremos la tarea de cada estado y finalmente mostraremos las transiciones nuevas entre ellos.

Estado cambiar clave

Este estado funciona igual que el estado INGRESAR_CLAVE, con la diferencia de que si la clave ingresada es correcta, el siguiente estado será CLAVE_NUEVA en vez de ABIERTO. Por éstas diferencias es por lo cual decidimos hacer un estado diferente, que como se puede ver en el código al final de informe, la implementación es prácticamente la misma.

Estado nueva clave

En este estado se espera que el usuario proceda a ingresar la clave que desea. Éste se implementó utilizando un timeout entre cada tecla, donde si pasan 3 segundos entre el presionado de una tecla y otra, se volverá al estado CERRADO y la clave actual del sistema quedará intacta. Notar que esta funcionalidad se implementa en el siguiente inciso, por ende la transición de estado se explica en el siguiente inciso también.

Para la finalización del ingreso y el cambio exitoso de la clave, esperamos a que se presione la tecla “D”, lo cual indica el fin de ingreso y pasa al estado FIN_NUEVA_CLAVE. Notar que la longitud de la clave puede ser entre 1 y 16 caracteres. Pero, agregamos una cláusula, mediante el ingreso de la tecla “#”, para que el usuario vuelva al estado CERRADO, cancelando el ingreso y dejando la clave actual del sistema intacta.

Estado fin nueva clave

Este estado es muy simple. Lo único que hace es utilizar un timeout de cinco segundos, durante esto cinco segundos se muestra el mensaje “Fin nueva clave” en la segunda fila y la hora actual en la primer fila(como en la mayoría de los estados). Una vez pasados los cinco segundos se vuelve al estado CERRADO.

Notar que la única forma de acceder a este estado es cambiando la clave de manera exitosa.

Las nuevas transiciones de estado necesarias para este inciso serian:

- CERRADO → CAMBIAR_CLAVE: Presionando la tecla 'D' se hace esta transición.
- CAMBIAR_CLAVE → CLAVE_NUEVA: Se pedirá introducir la clave actual del sistema, en caso de que sea correcta, se realizará la transición para ingresar la clave nueva.
- CAMBIAR_CLAVE → DENEGADO: Si la contraseña ingresada no es igual a la contraseña actual que tiene el sistema, se debe cancelar el cambio de la clave.
- CLAVE_NUEVA → FIN_NUEVA_CLAVE: Una vez que se termina de ingresar la clave nueva y presionando la tecla "D" (donde se actualizará la clave del sistema).
- FIN_NUEVA_CLAVE → CERRADO: Pasados los 5 segundos que se desea mostrar el mensaje "Fin ingreso nueva clave".
- CLAVE_NUEVA → CERRADO: Apretando la tecla "#" por si se cometió algún error y se desea cancelar el cambio de la clave.

Inciso F

Consigna

Las acciones de ingreso de claves por parte del usuario requieren un time-out de manera de cancelar la operación si la misma tarda demasiado y volver al estado por defecto. Agregar esta funcionalidad.

Interpretación

Hasta este momento teníamos la cerradura electrónica completa básicamente pero ¿qué pasa si al ingresar la contraseña pasa demasiado tiempo y nunca apretamos la tecla "D" que indica la finalización de la misma ? Esto produciría que la máquina de estados se quede en el estado INGRESAR_CLAVE y no cambiaría. Entonces, consideramos que si pasa un tiempo entre que se ingresa un dígito de la clave y otro, el cuál si es mayor al esperado, automáticamente se denegará el acceso como si la contraseña estuviera incorrecta, pasando del estado INGRESAR_CLAVE, CAMBIAR_CLAVE a DENEGADO y de CLAVE_NUEVA a CERRADO. El tiempo que consideramos adecuado de espera es de tres segundos.

Resolución

Este inciso se resolvió agregando a la librería ***timeout*** la funcionalidad de poder resetear el timer sin la necesidad de esperar que éste termine y de volver a contar desde cero con la configuración asignada inicialmente. Dicha función se llama ***timeout_reset.***

Habiendo dicho esto, para agregar la funcionalidad de “timeout” se modificaron los estados que cumplian con la tarea del ingreso de la clave por parte del usuario, para que dichos estados sean expulsado si esto tardaba mucho tiempo.

Entonces, dentro de los estados CAMBIAR_CLAVE, INGRESAR_CLAVE y CLAVE_NUEVA, se agregó un timeout de 3 segundos, el cual se reinicia cada vez que el usuario presiona una tecla. Constantemente se estará preguntando si el timeout término, es decir se pasaron los tres segundos desde que se presiono la última tecla, si esto sucede se expulsa del estado actual y se cambia al estado DENEGADO excepto que se esté en el estado CLAVE_NUEVA y por esto pase al estado CERRADO.

El timeout se resetea con la interacción de cada tecla y no desde que se inicializa el estado, con el fin de no tener un tiempo total fijo limitado para ingresar la contraseña.

Las nuevas transiciones de estados serían:

- INGRESAR_CLAVE → DENEGADO: pasados tres segundos de presionar la última tecla
- CAMBIAR_CLAVE → DENEGADO: pasados tres segundos de presionar la última tecla
- CLAVE_NUEVA → CERRADO: pasados tres segundos de presionar la última tecla

Anexo: Archivos de C

Para una visión más detallada del codigo, consultar el repositorio:
https://github.com/trorik23/CDyuC-TP2

MCUinit.c

```c
#include "clock.h"
#include "teclado.h"
#include "mef.h"
#include "timeout.h"
unsigned char contador=0;
const unsigned char RESET_CONTADOR=10;

__interrupt void isrVrtc(void)
{
    if(++contador==RESET_CONTADOR){
            tick();
            contador=0;
    }
    timeout_contar();
    comprobar_tecla();
    MEF_update();
    RTCSC_RTIF=1;
}
/* end of isrVrtc */
```

Main.c

```c
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "pantalla.h"
#include "mef.h"
#include "teclado.h"

#ifdef __cplusplus
extern "C"
#endif

void MCU_init(void); /* Device initialization function declaration */

void main(void) {
    MCU_init();
    teclado_init();
    MEF_init();
    pantalla_init();
    for (;;) {
        pantalla_update();
    }
}
```

Clock.c

```c
#include "clock.h"
char HH = 23;
char MM = 59;
char SS = 55;

//asumimos que es llamado cada 1 segundo
void tick(void) {
    SS++;
    if (SS == 60) {
        SS = 0;
        MM++;
        if (MM == 60) {
            MM = 0;
            HH++;
            if (HH == 24) {
                HH = 0;
            }
        }
    }

}
//el string que pasa por referencia debe ser de minimo 9 caracteres(8+caracter nulo)
void get_time_as_str(char str[]) {
    str[0] = HH / 10 + '0';
    str[1] = HH % 10 + '0';
    str[2] = ':';
    str[3] = MM / 10 + '0';
    str[4] = MM % 10 + '0';
    str[5] = ':';
    str[6] = SS / 10 + '0';
    str[7] = SS % 10 + '0';
    str[8] = '\0';
}

void cambiar_hora(char hh, char mm, char ss) {
    if (hh < IGNORE_HH)
        HH = hh;
    if (mm < IGNORE_MM)
        MM = mm;
    if (ss < IGNORE_SS)
        SS = ss;
}
```

Eventotecla.c

```c
#include "eventotecla.h"
char tecla=CASO_NULO;
char vacio = 1;//vacio=1 --> TRUE

void push_tecla(char t) {
    tecla = t;
    vacio = 0;
}
char pop_tecla() {
    vacio = 1;
    return tecla;
}
char tecla_vacia() {
    return vacio;
}
```

Timeout.c

```c
#include "timeout.h"

const unsigned char flag = '?';
unsigned char timeoutAct = flag;
unsigned char timeout_contador = 0; // en 100 milisegundos

void timeout_contar(void) {
    timeout_contador++;
}

// sec escalado en 0.1 segundos
// ejemplo: si sec=5 -> son 0.5 segundos
void timeout_empezar(char sec) {
    if (timeoutAct != flag){
        return;
    }
    timeout_contador = 0;
    timeoutAct = sec;
}
void timeout_reset(){
    timeout_contador=0;
}

char timeout_termino(void) {
    if (timeout_contador >= timeoutAct) {
        timeoutAct = flag;
        return 1;
    }
    return 0;
}

void timeout_cerrar(void){
    timeoutAct=flag;
}
```

Mef.c

```
1.  #include "clock.h"
2.  #include "eventotecla.h"
3.  #include "pantalla.h"
4.  #include "timeout.h"
5.  #include <string.h>
6.
7.  typedef enum {
8.      CERRADO,
9.      ABIERTO,
10.     DENEGADO,
11.     CAMBIAR_HH,
12.     CAMBIAR_MM,
13.     CAMBIAR_SS,
14.     CAMBIAR_C,
15.     INGRESAR_CLAVE,
16.     CLAVE_NUEVA,
17.     FIN_NUEVA_CLAVE
18. } state;
19. state estadoActual;
20.
21. char leer_clave(void);
22. char valido(char, char);
23. char construir_num(char, char);
24. char obtener_numero(char);
25. char igualdad_strings(char[], char[]);
26.
27. void fCERRADO(void);
28. void fABIERTO(void);
29. void fDENEGADO(void);
30. void fCAMBIAR_HH(void);
31. void fCAMBIAR_MM(void);
32. void fCAMBIAR_SS(void);
33. void fCAMBIAR_C(void);
34. void fINGRESAR_CLAVE(void);
35. void fCLAVE_NUEVA(void);
36. void fFIN_NUEVA_CLAVE(void);
37.
38. void (*MEF[])(void) = { fCERRADO, fABIERTO, fDENEGADO, fCAMBIAR_HH,
39.         fCAMBIAR_MM, fCAMBIAR_SS, fCAMBIAR_C, fINGRESAR_CLAVE, fCLAVE_NUEVA,
    fFIN_NUEVA_CLAVE};
40.
41. char key;
42. char claveActual[16] = { '1', '2', '3', '4' };
43. char claveLeida[16];
44. char hhmmss[9];
45.
46. void MEF_init(void) {
47.     key = CASO_NULO;
48.     estadoActual = CERRADO;
49. }
50.
51. void MEF_update(void) {
52.     (*MEF[estadoActual])();
```

```
53. }
54. 
55. void fCERRADO(void) {
56.     get_time_as_str(hhmmss);
57.     setear_string(hhmmss, 0);
58.     setear_string("CERRADO", 1);
59.     if (tecla_vacia() == 0) {
60.         key = pop_tecla();
61.     }
62.     switch (key) {
63.     case 'A':
64.         estadoActual = CAMBIAR_HH;
65.         break;
66.     case 'B':
67.         estadoActual = CAMBIAR_MM;
68.         break;
69.     case 'C':
70.         estadoActual = CAMBIAR_SS;
71.         break;
72.     case 'D':
73.         estadoActual = CAMBIAR_C;
74.         break;
75.     case '*':
76.         break;
77.     case '#':
78.         break;
79.     case CASO_NULO:
80.         break;
81.         // entra a default cuando key es un numero
82.     default:
83.         estadoActual = INGRESAR_CLAVE;
84.         break;
85.     }
86. }
87. 
88. char valido(char n, char max) {
89.     return ((n - '0' <= max) && (n - '0' >= 0)) ? 1 : 0;
90. }
91. 
92. char construir_num(char pro, char sdo) {
93.     return (((pro - '0') * 10) + (sdo - '0'));
94. }
95. 
96. char i, pro, sdo;
97. 
98. // return:
99. //      1: no hay tecla o no es valido
100.    //      0: todo piola
101.    char obtener_numero(char max) {
102.        if (tecla_vacia() == 1)
103.            return 1;
104.        key = pop_tecla();
105.        if (valido(key, max) == 0)
106.            return 1;
```

```
107.        i = ++i % 2;
108.        return 0;
109.    }
110.
111.    void fABIERTO(void) {
112.        get_time_as_str(hhmmss);
113.        setear_string(hhmmss, 0);
114.        setear_string("ABIERTO", 1);
115.        timeout_empezar(50);
116.        if (timeout_termino() == 1) {
117.            key = CASO_NULO;
118.            // borrar basura
119.            pop_tecla();
120.            estadoActual = CERRADO;
121.        }
122.    }
123.
124.    void fFIN_NUEVA_CLAVE(void) {
125.        get_time_as_str(hhmmss);
126.        setear_string(hhmmss, 0);
127.        setear_string("FIN NUEVA CLAVE", 1);
128.        timeout_empezar(50);
129.        if (timeout_termino() == 1) {
130.            key = CASO_NULO;
131.            // borrar basura
132.            pop_tecla();
133.            estadoActual = CERRADO;
134.        }
135.    }
136.
137.    void fDENEGADO(void) {
138.        get_time_as_str(hhmmss);
139.        setear_string(hhmmss, 0);
140.        setear_string("DENEGADO", 1);
141.        timeout_empezar(20);
142.        if (timeout_termino() == 1) {
143.            key = CASO_NULO;
144.            // borrar basura
145.            pop_tecla();
146.            estadoActual = CERRADO;
147.        }
148.    }
149.
150.    unsigned char flagBlink = 1;
151.    void control_flagBlink() {
152.        timeout_empezar(5);
153.        if (timeout_termino() == 1) {
154.            flagBlink = flagBlink == 0 ? 1 : 0;
155.        }
156.    }
157.
158.    char flagDosTeclas = 0;
159.    void blinkear(char offset) {
160.        get_time_as_str(hhmmss);
```

```
161.
162.        if (i == 1) {
163.            hhmmss[offset] = pro;
164.        }
165.        if (flagDosTeclas == 1) {
166.            hhmmss[offset] = pro;
167.            hhmmss[1 + offset] = sdo;
168.        }
169.        if (flagBlink == 1) {
170.            hhmmss[i + offset] = ' ';
171.        }
172.        setear_string(hhmmss, 0);
173.    }
174.
175.    void fCAMBIAR_HH(void) {
176.        setear_string("CAMBIANDO HORA", 1);
177.        control_flagBlink();
178.        blinkear(0);
179.        // return si es el primer numero y el numero no es valido
180.        if ((i == 0) && (obtener_numero(2) == 0)) {
181.            pro = key;
182.            return;
183.        }
184.        if ((pro == '2') && (obtener_numero(3) == 0)) {
185.            sdo = key;
186.            flagDosTeclas = 1;
187.        } else if ((pro != '2') && obtener_numero(9) == 0) {
188.            sdo = key;
189.            flagDosTeclas = 1;
190.        }
191.
192.        if ((key == 'A') && (flagDosTeclas == 1)) {
193.            flagDosTeclas = 0;
194.            cambiar_hora(construir_num(pro, sdo), IGNORE_MM, IGNORE_SS);
195.            key = CASO_NULO;
196.            i = 0;
197.            timeout_cerrar();
198.            estadoActual = CERRADO;
199.        }
200.    }
201.
202.    void fCAMBIAR_MM(void) {
203.        setear_string("CAMBIANDO MIN", 1);
204.        control_flagBlink();
205.        blinkear(3);
206.        if ((i == 0) && (obtener_numero(5) == 0)) {
207.            pro = key;
208.            return;
209.        }
210.        if (obtener_numero(9) == 0) {
211.            sdo = key;
212.            flagDosTeclas = 1;
213.        }
214.        if ((flagDosTeclas == 1) && (key == 'B')) {
```

```
215.            flagDosTeclas = 0;
216.            cambiar_hora(IGNORE_HH, construir_num(pro, sdo), IGNORE_SS);
217.            timeout_cerrar();
218.            key = CASO_NULO;
219.            i = 0;
220.            estadoActual = CERRADO;
221.        }
222.    }
223.
224.    void fCAMBIAR_SS(void) {
225.        setear_string("CAMBIANDO SEG", 1);
226.        control_flagBlink();
227.        blinkear(6);
228.        if ((i == 0) && (obtener_numero(5) == 0)) {
229.            pro = key;
230.            return;
231.        }
232.        if (obtener_numero(9) == 0) {
233.            sdo = key;
234.            flagDosTeclas = 1;
235.        }
236.        if ((flagDosTeclas == 1) && (key == 'C')) {
237.            flagDosTeclas = 0;
238.            cambiar_hora(IGNORE_HH, IGNORE_MM, construir_num(pro, sdo));
239.            timeout_cerrar();
240.            key = CASO_NULO;
241.            i = 0;
242.            estadoActual = CERRADO;
243.        }
244.    }
245.
246.    char car = 0;
247.    char asteriscos[17] = "                ";
248.    void fCAMBIAR_C(void) {
249.        char j;
250.        setear_string("CLAVE ACTUAL:", 0);
251.        timeout_empezar(30);
252.        if (timeout_termino() == 1) {
253.            estadoActual = DENEGADO;
254.            for (j = 0; j < 16; j++) {
255.                asteriscos[j]=' ';
256.            }
257.            claveLeida[car] = '\0';
258.            car = 0;
259.            return;
260.        }
261.        for (j = 0; j < car; j++) {
262.            asteriscos[j]='*';
263.        }
264.        setear_string(asteriscos, 1);
265.        char aux = leer_clave();
266.        if (aux > 0) {
267.            for (j = 0; j < 16; j++) {
268.                asteriscos[j]=' ';
```

```
269.            }
270.            car = 0;
271.            timeout_cerrar();
272.            if (aux == 1) {
273.                if (igualdad_strings(claveLeida, claveActual) == 1) {
274.                    estadoActual = CLAVE_NUEVA;
275.                } else {
276.                    estadoActual = DENEGADO;
277.                }
278.            }
279.            key = CASO_NULO;
280.        }
281.    }
282.
283.    void fCLAVE_NUEVA(void) {
284.        char j;
285.        setear_string("CLAVE NUEVA:", 0);
286.        timeout_empezar(30);
287.        for (j = 0; j < car; j++) {
288.            asteriscos[j]='*';
289.        }
290.        setear_string(asteriscos, 1);
291.        if (timeout_termino() == 1) {
292.            key = CASO_NULO;
293.            estadoActual = CERRADO;
294.            car = 0;
295.            return;
296.        }
297.        char aux = leer_clave();
298.        if (aux > 0) {
299.            if(aux == 1){
300.                strcpy(claveActual, claveLeida);
301.            }
302.            key = CASO_NULO;
303.            car = 0;
304.            timeout_cerrar();
305.            for (j = 0; j < 16; j++) {
306.                asteriscos[j]=' ';
307.            }
308.            estadoActual = CERRADO;
309.            return;
310.        }
311.    }
312.
313.    char igualdad_strings(char clavel[], char clavea[]) {
314.        char k;
315.        if (strlen(clavel) != strlen(clavea))
316.            return 0;
317.        for (k = 0; k < strlen(clavea); k++) {
318.            if(clavel[k]!=clavea[k])
319.            return 0;
320.        }
321.        return 1;
322.    }
```

```
323.
324.   void fINGRESAR_CLAVE(void) {
325.       char j;
326.       timeout_empezar(30);
327.       if (timeout_termino() == 1) {
328.           estadoActual = DENEGADO;
329.           for (j = 0; j < 16; j++) {
330.               asteriscos[j]=' ';
331.           }
332.           car = 0;
333.           return;
334.       }
335.       get_time_as_str(hhmmss);
336.       setear_string(hhmmss, 0);
337.       for (j = 0; j < car; j++) {
338.           asteriscos[j]='*';
339.       }
340.       setear_string(asteriscos, 1);
341.
342.       if (car == 0)
343.           claveLeida[car++] = key;
344.       char aux = leer_clave();
345.       if (aux > 0) {
346.           for (j = 0; j < 16; j++) {
347.               asteriscos[j]=' ';
348.           }
349.           car = 0;
350.           timeout_cerrar();
351.           if(aux == 1){
352.               if (igualdad_strings(claveLeida, claveActual) == 1) {
353.                   estadoActual = ABIERTO;
354.               } else {
355.                   estadoActual = DENEGADO;
356.               }
357.           }
358.           key = CASO_NULO;
359.       }
360.   }
361.   // return: 0 si todavia no termino
362.   //         1 si termino correctamente la lectura de la clave
363.   //         2 si se cancelo la carga de la clave
364.   char leer_clave(void) {
365.       if (car == 16) {
366.           return 1;
367.       }
368.       if (tecla_vacia() == 0) {
369.           key = pop_tecla();
370.           // pregunta si es numero y es menor que 9
371.           timeout_reset();
372.           if (key == 'D') {
373.               claveLeida[car] = '\0';
374.               return 1;
375.           }
376.           if (key == '#') {
```

```
377.                return 2;
378.            }
379.            if (valido(key, 9) == 1) {
380.                claveLeida[car++] = key;
381.            }
382.        }
383.        return 0;
384.    }
```

pantalla.c

```
1.  #include "pantalla.h"
2.  #include "lcd.h"
3.  #include <string.h>
4.
5.  char pantalla[2][STR_LEN + 1] = { { "                " },
6.          { "                " } };
7.
8.  void setear_string(char str[], char fila) {
9.      char length = strlen(str);
10.     char offset = (STR_LEN - length) / 2;
11.     char i;
12.     //centra el string y completa la linea con espacios
13.     for (i = 0; i < STR_LEN; i++) {
14.         if((i<offset)||((length+offset)<=i)) {
15.             pantalla[fila][i]=' ';
16.         } else {
17.             pantalla[fila][i]=str[i-offset];
18.         }
19.     }
20. }
21. void pantalla_init(void) {
22.     LCD_init(DISPLAY_8X5 | _2_LINES, DISPLAY_ON | CURSOR_OFF);
23.     pantalla_update();
24. }
25. void pantalla_update(void) {
26.     LCD_pos_xy(0, 0);
27.     LCD_write_string(pantalla[0]);
28.     LCD_pos_xy(0, 1);
29.     LCD_write_string(pantalla[1]);
30. }
```

Teclado.c

```
1.  #include <mc9s08sh8.h>
2.  #include "eventotecla.h"
3.  #include "teclado.h"
4.
5.  const char NO_PRESIONADO = 42;
6.  const char TECLA_VACIA = '<';
7.  const char map[4][4] = { { '1', '2', '3', 'A' }, { '4', '5', '6', 'B' }, { '7',
8.          '8', '9', 'C' }, { '*', '0', '#', 'D' } };
9.  char teclaUlt = TECLA_VACIA;
10.
11. void columna_presionada(char *);
12. void fila_presionada(char *, char);
13.
14. void teclado_init(void){
15.     PTBDD = 0x0F;
16.     PTBPE = 0xF0;
17. }
18.
19. void comprobar_tecla(void) {
20.     char filaPresionada;
21.     char columnaPresionada;
22.     columna_presionada(&columnaPresionada);
23.     if (columnaPresionada == NO_PRESIONADO) {
24.         if (teclaUlt == TECLA_VACIA)
25.             return;
26.         push_tecla(teclaUlt);
27.         teclaUlt = TECLA_VACIA;
28.         return;
29.     }
30.     fila_presionada(&filaPresionada, columnaPresionada);
31.     if (filaPresionada == NO_PRESIONADO) {
32.         return;
33.     }
34.     teclaUlt = map[filaPresionada][columnaPresionada];
35. }
36.
37. void fila_presionada(char * fp, char cp) {
38.     char filaAct = 0;
39.     while (filaAct < 4) {
40.         PTBD=0x0F;
41.         switch(filaAct) {
42.         case 0:
43.             PTBD_PTBD0=0;
44.             break;
45.         case 1:
46.             PTBD_PTBD1=0;
47.             break;
48.         case 2:
49.             PTBD_PTBD2=0;
50.             break;
51.         case 3:
52.             PTBD_PTBD3=0;
```

```
53.             break;
54.         }
55.         columna_presionada(&cp);
56.         if(cp!=NO_PRESIONADO) {
57.             *fp=filaAct;
58.             PTBD=0x00;
59.             return;
60.         }
61.         filaAct++;
62.     }
63.     *fp = NO_PRESIONADO;
64. }
65. 
66. void columna_presionada(char * cp) {
67.     if (PTBD_PTBD7 == 0) {
68.         *cp = 3;
69.         return;
70.     }
71.     if (PTBD_PTBD6 == 0) {
72.         *cp = 2;
73.         return;
74.     }
75.     if (PTBD_PTBD5 == 0) {
76.         *cp = 1;
77.         return;
78.     }
79.     if (PTBD_PTBD4 == 0) {
80.         *cp = 0;
81.         return;
82.     }
83.     *cp = NO_PRESIONADO;
84. }
```