|  |  |  |
| --- | --- | --- |
| **Nombre y Apellidos** | **Nº Mat.** | **Horas** |
| Alejandro Pérez Vicente | 16335 | 16 |
| Guillermo Moreno Vázquez | 16289 | 20 |

Equipo: **35**

**DETECTOR DE MONEDAS**

Se ha dividido el funcionamiento del del detector de monedas en dos grandes partes:

1. Detectar cuando entra una moneda e interpretarla: En estas operaciones intervienen los siguientes archivos del código:

* **sensores.c:** Se asignan los sensores ópticos 1 y 2, y el sensor electromagnético al input capture correspondientes a los timers de 16 bits 1,3 y 4 respectivamente. De esta forma se consigue una mayor precisión al obtener los tiempos cuando entra una moneda. Los timers son configurados en modo normal, con cuenta hasta overflow y un preescalado de 8. Con el preescalado se consigue que cada cuenta sea de 1µs, lo cual facilitará cálculos posteriores y mantendrá la precisión de los tiempos capturados por los sensores. Según la tabla de datos experimentales, los tiempos máximos son de unos 40ms que equivale a una cuenta de 40000 en los timers lo cual no supone ningún problema al ser menor que un overflow (65536).

Las interrupciones por input capture de los sensores ópticos registran los tiempos cada flanco de subida o de bajada, y cuando tienen los datos necesarios calculan t1, t2 y t3.

La interrupción por input capture del sensor electromagnético registra únicamente los flancos de subida para calcular el periodo y con este la frecuencia. Cuando se detecte una frecuencia inferior a 1800, considerando ideal el sensor, significa que ha entrado una moneda. Se filtrará el primer periodo distinto por si es erróneo y se cogerán los siguientes como la frecuencia **f** de la moneda que ha entrado.

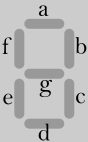
No se ha distinguido entre que sensor está situado antes y cuál después, considerando posibles ambas situaciones. Se impone la doble condición de que la moneda haya pasado tanto por los sensores ópticos como por el electromagnético para poder identificarla. Esta condición se consulta periódicamente al final de la interrupción del sensor electromagnético. Una vez la moneda a pasado por todos los sensores se calcula su ratio **r**, se identifica si es válida o no, y si es válida se almacena.

Una moneda es válida si, por ejemplo, tiene un ratio dentro del intervalo de las monedas de 20cent y además su frecuencia también está dentro del intervalo correspondiente a las frecuencias de monedas de 20cent. Los límites de estos intervalos se eligen de acuerdo con los máximos y mínimos de los datos experimentales aplicando un cierto margen. Cuando no cumple con ninguna relación de intervalos la moneda es no válida. Si la moneda es válida, se suma su cantidad al *depósito* donde se lleva la cuenta del dinero total y se almacena.

* **almacenar.c:** Se encarga de controlar la apertura y cierre de la compuerta. Si la moneda ha sido identificada válida abre la compuerta, reinicia un contador que llevará el tiempo que permanece abierta y activa las interrupciones por OCRA=TCNT1+1000 del timer 1. Es decir, coge el tiempo actual cuando se abrió la compuerta y programa una interrupción para dentro de 1ms (porque las cuentas son de 1µs). En la interrupción por OCRA se actualiza el OCRA=OCRA+1000 para que suceda nuevamente al pasar 1ms, y se suma uno al contador. De esta forma el contador indica el numero de ms que la compuerta lleva abierta. Al llegar al tiempo especificado se cierra la compuerta y se desactivan las interrupciones por OCRA.

1. Control de los actuadores: utiliza los siguientes archivos:

* **main.c:** En el main del programa se realizará inicialmente todos los setup necesarios, luego la comunicación con la API (explicada más adelante) y el control de los actuadores en función del estado de la moneda entrante dentro del bucle while(1). Si entra una moneda válida enciende el led verde y suena un pitido, pero si no es válida enciende el led rojo y suenan 3 pitidos. Si no entra ninguna moneda no se modifica ninguno de estos actuadores. En el caso de entrar varias monedas tan deprisa que no da tiempo a terminar de ejecutar todo el bucle del main, entonces solo se activarán los actuadores para la primera moneda que entró. Sin embargo, todas las monedas serán identificadas y almacenadas con criterio pues, como ya se ha visto estas operaciones se realizan por interrupción.
* **leds.c**: El funcionamiento para controlar los tiempos en los leds, buzzer y display es muy similar al utilizado en la compuerta. En el caso de los leds, se establecen interrupciones también cada ms pero esta vez por OCRB del timer 1. Mientras no entra una moneda el led rojo parpadea. Inicialmente se encuentra apagado y el contador contará el tiempo que permanece así apagado. Cuando cumple este tiempo, se enciende y se reinicia el contador que ahora contará hasta un tiempo de encendido. Si entra una moneda, desde el main se llama a la función que enciende el led correspondiente al estado de la moneda. Esta función enciende el led, activa una bandera (entraMoneda) y reinicia el contador que ahora llegará hasta un nuevo tiempo de encendido. La bandera sirve para diferencia entre el funcionamiento cuando entra una moneda y cuando no (parpadeo).
* **display.c**: Para controlar la frecuencia de refresco del selector se utilizan interrupciones por OCRC del timer 1. En este caso se han programado interrupciones con una frecuencia de 4kHz. El display encedido se almacena en una variable decimal, *display* (de 0 a 3), y se modifica al final de cada interrupción. Para encender cada señal del selector se convierte esta variable a binario y se asigna el bit correspondiente a cada señal del selector. Según cual sea el display encendido en el momento de la interrupción se coge la cifra correspondiente que tiene *deposito* en ese momento (cabe destacar que la variable deposito también es decimal, de 4 cifras que corresponden con el número que debería aparecer en el total de los displays). Finalmente se seleccionan los segmentos necesarios para dicha cifra.

****

* **buzzer.c**: Para generar los pulsos del pitido se ha utilizado un PWM en un nuevo timer de 16 bits. Se ha configurado el timer 5 nuevamente con preescalado de 8 pero en esta ocasión con modo Fast PWM hasta OCRA. Se conectará el buzzer al OCRB y se programará con “set en bottom” y “clear en OCRB”. De esta forma con el OCRA establecemos el periodo y a partir del OCRB elegimos el duty cycle. Desde el main se llama a una función indicando el numero de pitidos que deben sonar. Esta función activa la generación del PWM para hacer sonar el pitido y activa también las interrupciones por OCRA. Al igual que con los otros actuadores, en la interrupción se lleva la cuenta del tiempo que lleva sonando el buzzer en un contador. Cuando el contador alcanza el tiempo especificado se silencia el buzzer y se reinicia el contador hasta un nuevo tiempo de silencio. Si solo era un pitido, en cuanto se pone en silencio se desactivan las interrupciones por OCRA, luego el buzzer permanecerá en silencio. Pero si son más de un pitido, al cumplirse el tiempo de silencio se vuelve a poner en sonido el buzzer reiniciando el contador y repitiendo nuevamente el proceso.

Finalmente, se han introducido en el main del programa las funciones que permiten la comunicación con la API.

|  |  |  |
| --- | --- | --- |
| SEÑALES | PUERTO | PIN |
| SO1 | D | 4 |
| SO2 | E | 7 |
| EM1 | L | 0 |
| RXD0 | E | 0 |
| TXD0 | E | 1 |
| B1 | L | 4 |
| E1 | D | 0 |
| L1 | A | 0 |
| L2 | A | 1 |
| SEL0 | B | 0 |
| SEL1 | B | 1 |
| S0 | C | 0 |
| S1 | C | 1 |
| S2 | C | 2 |
| S3 | C | 3 |
| S4 | C | 4 |
| S5 | C | 5 |
| S6 | C | 6 |

* **api.c** : Para establecer una comunicación entre el PC y el uC se ha utilizado el protocolo de comunicación serial UART. El microcontrolador transmite al PC en cada iteración del bucle el dinero total acumulado en la maquina. El dinero total guarda en la variable de tipo entero “deposito”. El modo de transmisión TX de datos será asíncrono, es decir, la velocidad de transmisión de datos no está fijada por un reloj. En vez de eso, mediante los bits “Start bit”, “Stop bit” y “Parity bit” controlaremos el flujo de datos. Con “Start bit” alertamos al receptor que una secuencia de caracteres va a ser enviada. Después se envían los cuatro dígitos de “deposito” en formato ASCII. Es necesario realizar una conversión previa del tipo de dato. Para ello, usamos la función “sprintf()” de la librería “stdio.h” que nos permitirá almacenar valor entero de “deposito” en formato ASCII en una cadena de caracteres. La transmisión de datos de realiza caracter por caracter (byte por byte). Después del bit “Start bit” enviamos los bits individuales que componen el carácter(8 bits). El “BAUD” o bits/segundo enviados en la comunicación serial se tomaran 9600 baudos(es lo que frecuentemente se usa). Al encender(reset) la maquina se quiere recibir el dinero total almacenado desde el PC, por lo que también se tendrá que configurar el puerto de recepción RX. El microcontrolador recibirá el valor mediante una cadena de caracteres en formato ASCII. Al igual que en la transmisión de datos, en la recepción de datos se usará la función “atoi()”, que permite almacenar en la variable “deposito” el equivalente entero de una cadena de caracteres en formato ASCII. Dado que la herramienta de simulación de Atmel Studio no permite simular comunicaciones seriales USART, se ha decidido integrar dentro de las API’s un delay de 50ms con el fin de simular el tiempo que tardaría tanto recibir como transmitir los datos por el bus serial.

**Tabla de conexiones:**