UNIVERSIDAD DE COSTA RICA

Escuela de Ingeniería Eléctrica

IE-0623 Microprocesadores

Proyecto Final

RADAR 623

Realizado por

Stuart Leal Quesada B53777

San José, Costa Rica 2019

Índice general

Índice de figuras

Índice de tablas

1 Resumen

El presente proyecto final del curso IE-623 Microprocesadores, de la Universidad de Costa Rica, pretende realizar el diseño y la implementación de una aplicación de software, que simule el funcionamiento de un Radar vehicular. La implementación se realizará sobre la tarjeta de desarrollo Dragon12, la cuál utiliza un microprocesador 9S12.

El diseño se pretende discutir en el presente documento a fondo. La implementación de este diseño se realizará en lenguaje ensamblador.

La aplicación pretende realizar un diseño real, de una aplicación que posee dos sensores, a 40 metros uno de otro. Cuando un vehículo pase por el primer sensor, el sistema empezará a contar el tiempo que le toma al vehículo alcanzar el segundo sensor. Dado este tiempo, y la separación entre los sensores, la aplicación calcula la velocidad del vehículo.

Con la velocidad del vehículo, El sistema enciende la pantalla informativa, con la velocidad límite y la velocidad del vehículo una vez que el vehículo esté 100 metros antes de la pantalla (Esto se determina asumiendo que la velocidad del vehículo permanece constante durante todo el trayecto). Posteriormente, el mensaje desaparecerá una vez que el vehículo pase por debajo de la pantalla.

Adicionalmente, el sistema tiene un sistema de ALERTA, que le informa al conductor que está por encima del límite de velocidad especificado. Estos son 5 leds que se barren de manera circular cuando se da esta condición. Finalmente, la aplicación tiene tres modos, los cuales son seleccionables con unos interruptores en la tarjeta de entrenamiento. Está el modo medición (modo normal de operación), modo configuración (el modo donde se configura la velocidad límite de la aplicación) y modo libre.

La implementación se realizó en un periodo aproximadamente de tres semanas. Se logró completar exitosamente todas las funcionalidades de la aplicación, y se pasó por un periodo de pruebas en donde se probaron las diferentes funcionalidades de la aplicación.

El proyecto consiste de 21 subrutinas, 5 de interupción y el resto subrutinas generales. Adicionalmente, está la rutina de MAIN, la cuál se detallará más adelante.

2 Desarrollo

2.1. Explicación general

A continuación, se presentarán todas las subrutinas diseñadas para la solución del problema planteado, tal y como lo describe el enunciado del proyecto de Delgado [3].

Primeramente, en la tabla 2.1 se detallan todas las estructuras de datos utilizadas para este diseño. Adicionalmente, la tabla 2.2 detalla el nombre de cada uno de los bits del registro de Banderas. Cada bit de este registro representa una bandera, que será utilizada más adelante en la implementación de las subrutinas.

2.1.1. Estructuras de datos

Tabla 2.1: Estructura de datos utilizada en la aplicación. Elaboración propia.

Subrutina	Nombre	Descripción	Dirección
	Banderas	**Variable tipo word**	$1000
MODO_CONFIG	V_LIM	Variable tipo byte	$1002
TAREA_TECLADO	MAX_TCL	**Constante tipo byte**	$1003
	Tecla	Variable tipo byte	$1004
	Tecla_in	Variable tipo byte	$1005
	Cont_reb	Variable tipo byte	$1006
	Cont_TCL	Variable tipo byte	$1007
	Patron	Variable tipo byte	$1008
	Num_Array	**Arreglo**	$1009
ATD_ISR	BRILLO	Variable tipo byte	$100b
	POT	Variable tipo byte	$100c
PANT_CTRL	TICK_EN	**Variable tipo word**	$100d
	TICK_DIS	**Variable tipo word**	$100f
CALCULAR	VELOC	Variable tipo byte	$1011
TCNT_ISR	TICK_VEL	Variable tipo byte	$1012

Table 2.1: Estructura de datos utilizada en la aplicación. Continuación.

Subrutina	Nombre	Descripción	Dirección
CONV_BIN_BCD	BIN1	Variable tipo byte	$1013
	BIN2	Variable tipo byte	$1014
	BCD1	Variable tipo byte	$1015
	BCD2	Variable tipo byte	$1016
BIN_BCD	BCD_L	Variable tipo byte	$1017
	BCD_H	Variable tipo byte	$1018
BCD_7SEG	DISP1	Variable tipo byte	$1019
	DISP2	Variable tipo byte	$101a
	DISP3	Variable tipo byte	$101b
	DISP4	Variable tipo byte	$101c
PATRON_LEDS	LEDS	Variable tipo byte	$101d
OC4_ISR	CONT_DIG	Variable tipo byte	$101e
	CONT_TICKS	Variable tipo byte	$101f
	DT	Variable tipo byte	$1020
	CONT_7SEG	**Variable tipo word**	$1021
	CONT_200	**Variable tipo word**	$1023
Subrutinas LCD	Cont_Delay	Variable tipo byte	$1025
	D2ms	**Constante tipo byte**	$1026
	D260us	**Constante tipo byte**	$1027
	D40us	**Constante tipo byte**	$1028
	Clear_LCD	**Constante tipo byte**	$1029
	ADD_L1	**Constante tipo byte**	$102a
	ADD_L2	**Constante tipo byte**	$102b
FORMAR_ARRAY	Teclas	**Tabla**	$1030
BCD_7SEG	SEGMENT	**Tabla**	$1040
LCD	iniDISP	**Tabla**	$1050
	CONFIG_L1	**Tabla**	$1060
	CONFIG_L2	**Tabla**	$106e
	MED_L1	**Tabla**	$107d
	MED_ESP_L2	**Tabla**	$108c
	MED_VEL_L2	**Tabla**	$109b

Table 2.1: Estructura de datos utilizada en la aplicación. Continuación.

Subrutina	Nombre	Descripción	Dirección
	MED_CAL_L2	**Tabla**	$10ab
	MODO_LIB_L1	**Tabla**	$10bb
	MODO_LIB_L2	**Tabla**	$10c9

Tabla 2.2: Nombre de cada bit en la variable Banderas.

Bit	Nombre de la bandera
0	TCL_LISTA
1	TCL_LEIDA
2	ARRAY_OK
3	PANT_FLAG
4	ALERTA
5	STATE_CHANGED
6	PH6_THEN
7	PH7_THEN
8	SEND_CMD || SEND_DATA
9	CALC_TICKS
10	OUT_RANGE
11	–
12	–
13	–
14	OPPOS_DIR
15	PH3_FIRED

2.2. Rutina MAIN

2.2.1. Rutina de inicialización

En esta subrutina, lo primero que se hace es la configuración de todos los periféricos utilizados en la aplicación. Todas las subrutinas de interrupción tienen un apartado en donde se detalla a fondo el porqué de la configuración específica de cada periférico.

Sin embargo, a continuación se detallará la configuración de los periféricos más generales, como el módulo de LCD, el teclado matricial, etc.

Lo **primero** que se hace en la subrutina MAIN (ver figura 2.1), es configurar el puntero de pila, en la dirección $3BFF. Seguidamente, se realiza la configuración del periférico ATD, la cuál será detallada a fondo en la sección correspondiente.

Luego, se realiza la configuración del módulo de timers, la cuál está detallada en la sección de OC4. Y posteriormente se realiza la configuración del teclado matricial.

Para esto, se debe de definir la parte alta del teclado matricial como entradas, y la parte baja del teclado matricial como salidas. Esto se logra con el valor de $F0 en el registro DDRA. Adicionalmente, al ser un puerto *core*, se definen las resistencias de *Pull-up* para todo el puerto, con el

bit menos significativo del registro PUCR.

A continuación, la configuración del módulo de RTI se explica en su respectiva sección, y luego se configura todos los bits del puerto K como salidas. Esta salida es utilizada para enviar comandos al periférico del LCD.

Seguidamente, se realiza la configuración para los LEDS. Recordando que en el puerto B se conectan los ánodos de los leds, entonces habilitamos el puerto B como salidas. Además, el bit 1 del puerto J, se utiliza para habilitar o deshabilitar los LEDS, según lo visto en el material de clase [2]. Entonces habilitamos este bit también como una salida. Finalmente, el puerto P, la parte inferior se utiliza para realizar la multiplexación de los displays de 7 segmentos. Por lo tanto también se habilita como salida.

Posteriormente, se realiza la configuración de los botones, la cuál se explica en la subrutina de CALCULAR. Y luego se habilitan las interrupciones mascarables. Seguido se prepara el valor de TC4 para la primera interrupción del OC4. Y por último se realiza la inicialización de las variables utilizadas en la aplicación.

Todos los contadores se ponen en su valor límite, además la variable V_LIM y VELOC se ponen en cero, para indicar el estado de esperando configuración. Se ponen en cero todas las banderas que hay. Finalmente, se borran las variables de Teclas, y se configuran los valores iniciales para el display LCD, y los displays de 7 segmentos.

La segunda parte del diagrama de flujos (ver figura 2.2) corresponde a la lógica de la rutina principal, la cuál será explicada a continuación.

2.2.2. Diagrama y explicación de la rutina MAIN

Descripción

Este es el programa principal, que maneja el modo en el que se encuentra la aplicación.

Explicación

Esta subrutina corre cíclicamente; Es decir, cuando termina vuelve a empezar. La idea es que constantemente esté ejecutando las acciones que corresponden al modo en que se encuentra la aplicación, sin que se quede 'bloqueado' en ningún modo.

Cuando se cambia de modo MEDICION a cualquier otro modo, se deberán apagar los periféricos que no se usan en esos modos, tales como OC4 y las interrupciones de key-wakeups.

La bandera que indica si cambió el modo o no se calcula con dos bits que se encuentran en la variable de Banderas, que guardan el modo anterior. El modo anterior se entiende como el estado de los interruptores PH7 y PH6.

Por lo tanto, **siguiendo el diagrama de flujos** que se encuentra en la figura 2.2, lo primero que se hace es cargar en el registro R2 el valor actual del puerto H, específicamente de PH7 y PH6 (Se hace la máscara con $C0). Posteriormente, se pregunta si V_LIM está en cero. De ser así, la rutina principal se salta la verificación de si hay un cambio de estado o no, puesto que realmente no importa. Cuando V_LIM está en cero, el sistema se está encendiendo por primera vez, y debemos de realizar la configuración del valor V_LIM a un valor válido.

Seguidamente, se verifica si el valor nuevo del puerto H es estado conocido o no ($40 representa un estado no definido), en cuyo caso, se procederá a ignorar este nuevo estado, y realizar el resto de las verificaciones como si el estado actual es el último estado válido.

A continuación, R1 contiene el estado anterior, y R2 contiene el nuevo estado (o último estado válido como se discutió anteriormente). **De ser ambos iguales**, significa que no hubo un cambio de estado, y por tanto ponemos en cero la bandera de STATE_CHANGED (En caso de que estuviera en 1). De ser diferentes, se pone en alto la bandera de STATE_CHANGED, y se procede a guardar el nuevo estado en el registro de banderas.

Posteriormente, se realiza la verificación de en cuál modo se encuentra el programa. Lo primero que se revisa, es si se está en modo MEDICION. De ser así, se salta a la subrutina de medición y luego se vuelve a iniciar la secuencia del programa principal.

Caso contrario, se verifica si el estado cambio. De ser así, podría ser que estem os transicionando del estado MEDICION a otro estado, por lo tanto se apaga la funcionalidad de key-wakeups y el módulo de timers. Adicionalmente se limpian algunas variables utilizadas en ese modo, y se limpian los displays de 7 segmentos, y algunas banderas utilizadas en ese modo.

Finalmente, se verifica si se encuentra en estado MODO_CONFIG o MODO_LIBRE, en cuyo caso se salta a la subrutina correspondiente al modo en que se encuentra la aplicación.

Diagrama

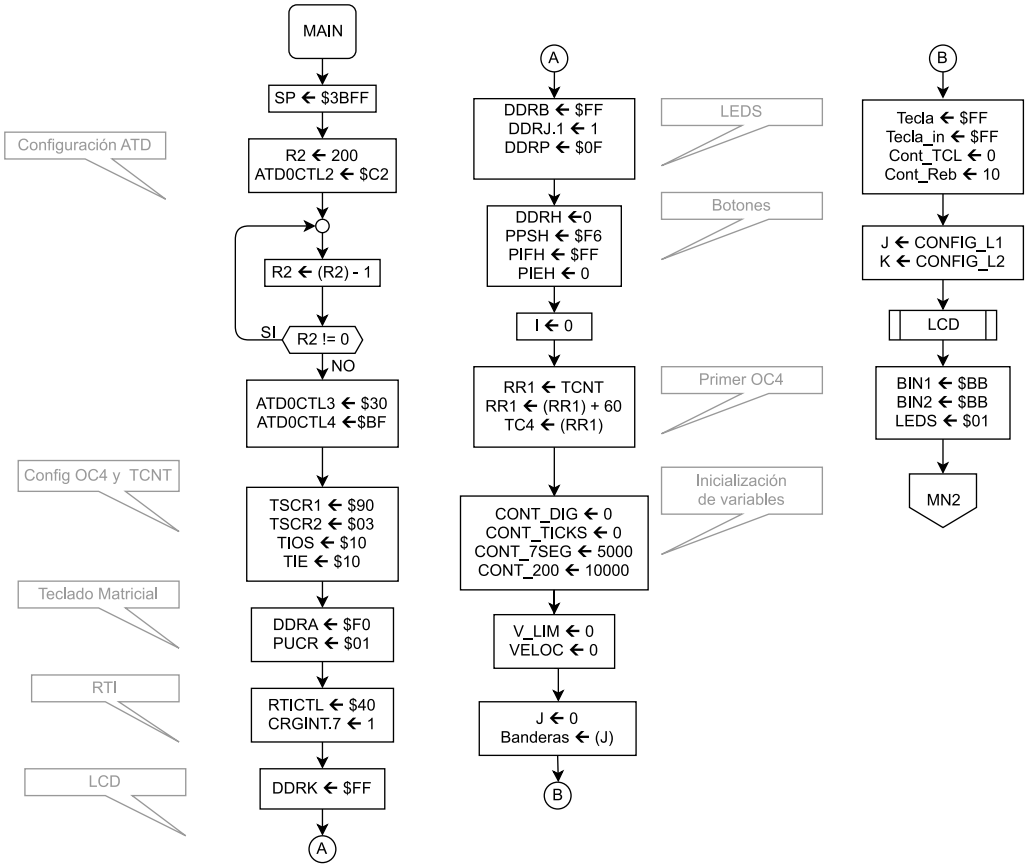

Figura 2.1: Diagrama de flujos para MAIN. Primera parte. Elaboración propia.

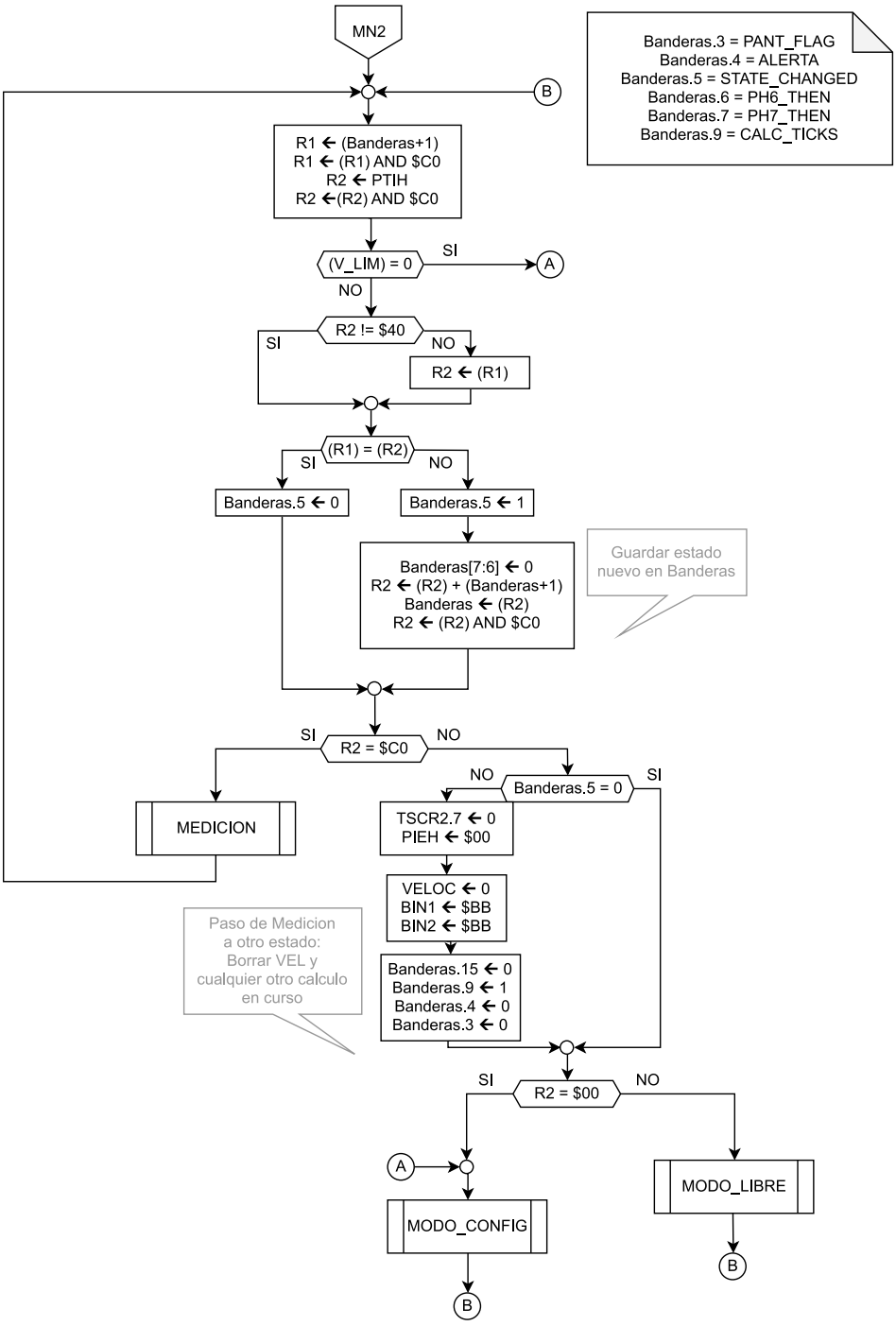

Figura 2.2: Diagrama de flujos para MAIN. Segunda parte. Elaboración propia.

2.3. Subrutina ATD_ISR

2.3.1. Cálculos para interrupción ATD_ISR

En lo que respecta a esta interrupción, debemos realizar los cálculos para encontrar el valor de Presescalador que vamos a utilizar. La fórmula que relaciona la frecuencia con el valor del preescalador es la siguiente:

$$frs = \frac{BUS\_CLK}{2 \cdot (PRS + 1)} \tag{2.1}$$

Para este caso, queremos la frecuencia más baja posible para este periférico, por tanto sería utilizar el valor más alto de PRS posible. Lo cuál corresponde a:

$$PRS = 31 = \$1F \tag{2.2}$$

2.3.2. Configuración de ATD_ISR

Para el primer registro de control $ATD0CTL2$, queremos habilitar el módulo de conversiones con el bit $ADPU = 1$, queremos habilitar la opción de $AFFC$, para que se borra la bandera de interrupción cuando se leen los registros de datos y también, habilitar las interrupciones con el bit $ASCIE = 1$. Esto significa que:

$$ATD0CTL2 = \$C2$$

Para el siguiente registro de control $ATDOCTL3$, queremos 6 conversiones, y además $FIFO = 0$. Entonces tenemos que:

$$ATD0CTL3 = \$30$$

En $ATD0CTL4$, vamos a querer configurar 4 periodos de reloj para el muestreo, y además $SRE8 = 1$ para tener conversiones a 8 bits. Además, vamos a querer el valor del preescalador en 31. Esto significa que:

$$ATD0CTL4 = \$BF$$

Finalmente, tenemos que activar la justificación a la derecha, y las conversiones hacerlas sin signo ($DJM = 1$ y $DSGN = 0$). Además queremos configurar $SCAN = MULT = 0$ para que se muestree sólo la entrada definida, 6 veces y se guarden los valores de $ADR0$ hasta $ADR5$. Y finalmente, queremos seleccionar la entrada 7 con los bits CC, CB y CA. Entonces:

$$ATD0CTL5 = \$87$$

2.3.3. Vector de interrupción ATD_ISR

Finalmente, el vector de interrupción para ATD se encuentra en la dirección $\$3E52$.

2.3.4. Diagrama y explicación de la subrutina ATD_ISR

Descripción

Esta subrutina, atiende la interrupción que se genera cuando se termina un ciclo de conversión. Se calcula el valor de $BRILLO$ a partir del promedio de seis mediciones en el PAD7.

Explicación

El valor de $BRILLO$ está en el rango de $[0, 20]$, sin embargo, como la conversión se realiza a 8 bits, el valor de cada registro de datos está en el rango de $[0, 255]$. Por lo tanto, utilzando la fórmula 2.3, encontramos el valor de $BRILLO$ a partir del promedio de seis valores contenidos en 6 registros de datos.

$$BRILLO = \frac{20 \cdot POT}{BRILLO} \tag{2.3}$$

Entonces, como se puede ver en la figura 2.3, lo **primero** que se hace en esta subrutina de interrupción, es sumar los seis valores de los primeros seis registros de datos, y se guarda el resultado en $RR1$. **Posteriormente**, se divide este valor entre 6, y ese resultado se guarda en J. De acá, tomamos los 8 bits menos significativos (El resultado de la divisón siempre tiene que ser menor o igual que 255), y lo guardamos en la variable POT.

Y **finalmente**, tomamos el resultado de la variable POT (temporalmente, en el DF esta variable todavía se encuentra en el registro $R1$), lo multiplicamos por 20, y luego lo dividimos por 255. El resultado de la división (nuevamente los 8 bits menos significativos), lo guardamos en la variable $BRILLO$.

Parámetros

- **Parámetros de entrada**. Esta subrutina no tiene parámetros de entrada.
- **Parámetros de salida**.
  - **POT**. El valor promedio de la conversión A/D en la variable POT.
  - **BRILLO**. El valor correspondiente de $BRILLO$, en una escala de $[0, 20]$ se guarda en esta variable.

Diagrama

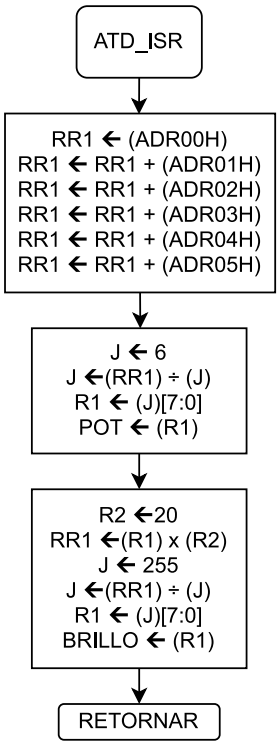

Figura 2.3: Diagrama de flujos de ATD_ISR. Elaboración propia.

2.4. Subrutina TCNT_ISR

2.4.1. Cálculos para interrupción TCNT_ISR

Para esta interrupción, lo que debemos de calcular es el tiempo que toma entre una interrupción y otra. Esto se puede calcular con la siguiente fórmula 2.4.

NOTA: Se usa un Preescalador de 8, puesto que la subrutina $OC4\_ISR$ utiliza este Preescalador la el módulo de TIMERS.

$$T_{TOI} = \frac{2^{16} \cdot 8}{24MHz} = \frac{1024}{46875} seg \tag{2.4}$$

2.4.2. Configuración de TCNT_ISR

Para esta interrupción, la configuración del módulo de reloj se detallará para la subrutina de $OC4\_ISR$.

La configuración propia de la interrupción por rebase, se hace escribiendo un 1 en el bit 7 del registro $TSCR2$.

Para deshabilitar la interrupción, se escribe un 0 en el mismo bit 7 de ese registro.

2.4.3. Vector de interrupción TCNT_ISR

El vector de interrupción para esta subrutina se encuentra en la dirección $\$3E5E$.

2.4.4. Diagrama y explicación de la subrutina TCNT_ISR

Descripción

Esta subrutina atiende la interrupción que se genera cuando se rebasa el registro $TCNT$. La subrutina se encarga de dos tareas: Incrementar el contador $TICK\_VEL$ utilizado para el cálculo de velocidad, y decrementar los contadores utilizados para el control del display LCD y los displays de 7 segmentos.

Explicación

Para el cálculo de velocidad, la subrutina CALCULAR se encargará de que, cuando se presione el botón S1, se ponga en cero el **contador de** $TICK\_VEL$. Este contador **será incrementado** en cada llamada de esta subrutina, la cuál sucede con una cadencia determinada en la fórmula 2.4.

En el diagrama de flujos mostrado en la figura 2.4, vemos que de hecho, **esta es la primera** tarea que se realiza en la subrutina. Se debe considerar el caso límite (Cuando el contador llega a 255), para el cuál se dejará de incrementar el contador. Este valor de tiempo corresponde a la velocidad más lenta que se podría calcular con este contador.

Ahora bien, **la lógica de TICK_EN y TICK_DIS** es la siguiente:

1. Si $TICK\_EN = 0$, pueden haber dos casos.
2. Uno, en donde no es cero, lo cuál, (se sabe que $TICK\_DIS$ siempre va a ser mayor o igual que $TICK\_EN$), $TICK\_DIS$ tampoco es cero, y se decrementan ambos contadores.
3. En el segundo caso, (derecha en el diagrama de la figura 2.4), puede ser que $TICK\_DIS$ ya sea cero, o no.
4. Si $TICK\_DIS$ ya es cero, entonces hay que apagar la bandera de $PANT\_FLAG$ (sólo si la misma no es cero, que sería la primera vez que ambos contadores llegan a cero, después de haber sido cargados con algún valor).
5. Si $TICK\_DIS$ no es cero, y es la primera vez que $TICK\_EN$ es cero, entonces hay que encender la bandera $PANT\_FLAG$. Si ya $PANT\_FLAG$ es 1, entonces quiere decir que estamos esperando sólo a que $TICK\_DIS$ se vuelva cero.

Y **por último**, como se puede ver en el diagrama de la figura 2.4, borramos la bandera de la interrupción, leyendo el registro $TCNT$.

Parámetros

- **Parámetros de entrada**.
  - **TICK_EN**. Esta variable es utilizada para determinar el momento en el que se debe de **mostrar** la velocidad actual y límite al conductor. La variable es decrementada por esta subrutina, y monitoreada por el programa principal, en particular por la subrutina $PANT\_CTRL$.

- **TICK_DIS**. Esta variable es utilizada para determinar el momento en el que se debe de **quitar** la velocidad actual y límite de la pantalla. La variable es decrementada por esta subrutina, y monitoreada por el programa principal, en particular por la subrutina $PANT\_CTRL$.

- **Parámetros de salida**.

  - **PANT_FLAG**. Bandera que, si se encuentra en 1, significa que el programa principal debería de colocar en el display de 7 segmentos la velocidad actual, y la velocidad límite, y además cambiar el contenido del display LCD con el mensaje correspondiente. La transición de 1 a 0 debería de provocar que se vuelva a mostrar el mensaje de esperando, indicando que otro carro podría aparecer.
  - **TICK_VEL**. Contador incrementado en cada entrada, con un tope de 255. Utilizado para el cálculo de velocidad del vehículo.

Diagrama

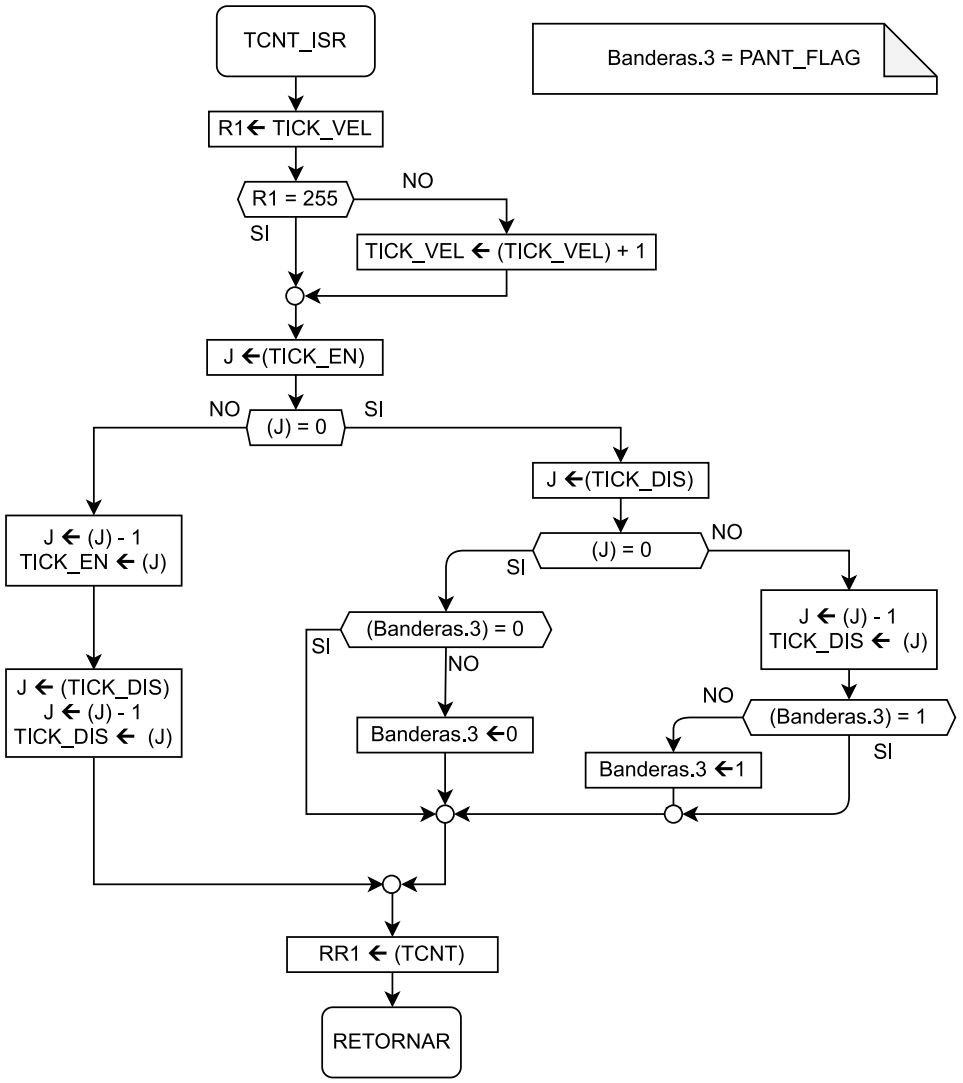

Figura 2.4: Diagrama de flujos para TCNT_ISR. Elaboración propia.

2.5. Subrutina CALCULAR

2.5.1. Cálculos para interrupción CALCULAR

Para realizar el cálculo de la velocidad, usamos la siguiente fórmula:

$$VELOC = \frac{40m}{n_{ticks}} \cdot \frac{tick}{seg} \cdot \frac{seg}{hora} \cdot \frac{km}{metro} \tag{2.5}$$

Desarrollando lo anterior, tenemos que:

$$VELOC = \frac{40}{n_{ticks}} \cdot \frac{46875}{1024} \cdot \frac{3600}{1} \cdot \frac{1}{1000}$$

$$= \frac{25}{n_{ticks}} \cdot \frac{16875}{64}$$

$$= \frac{421875}{n_{ticks} \cdot 64}$$

Entonces, la estrategia para realizar el cálculo de velocidad, será:

- Realizar la multiplicación de $n_{ticks} \cdot 64$ y guardarlo en X.
- Cargar en D $\#200$, luego en Y $\#16785$. Multiplicación queda en $Y : D$.
- Dividir $Y : D$ entre X, y guardar el resultado en VELOC.

NOTA: Hay que verificar que el resultado no sea más grande que 255 (es posible si n es muy pequeño, como por ejemplo $n = 1$ significa que $resultado = 6591$, lo cuál no es una velocidad con sentido. En caso de ser mayor a 255, guardar el máximo valor posible en VELOC (255).

2.5.2. Configuración de CALCULAR

Para hablitar todos los pines del puerto H como entradas, utilizamos el registro $DDRH$, escribiendo cero en todos los bits:

$$DDRH = \$00$$

Para habilitar las interrupciones para $PH3$ y $PH0$, utilizamos el registro $PIEH$:

$$PIEH = \$09 \tag{2.6}$$

Para definir la activación con flanco decreciente, se pone en cero los bits 3 y 0 del registro $PPSH$:

$$PPSH = \$F6$$

Finalmente, para borrar todas las banderas de interrupción, en caso de que haya alguna en cola:

$$PIFH = \$FF$$

2.5.3. Vector de interrupción CALCULAR

El vector de interrupción para esta subrutina se encuentra en la dirección $3E4C$.

2.5.4. Diagrama y explicación de la subrutina CALCULAR

Descripción

Esta subrutina realiza el cálculo de velocidad del vehículo, midiendo con $TICK\_VEL$ el tiempo que pasa entre el accionamiento de S1 y S2. Además, cuando se acciona S2 después de que fué accionado S1 (pasó un vehículo), se cambia el mensaje del LCD a *Calculando...*.

Explicación

Como se puede ver en el diagrama de la figura 2.5, esta subrutina tiene dos casos. Básicamente, se hace dos preguntas: ¿Se accionó S1, o se accionó S2?

Para el caso en que **se acciona S1** (PH3), primero se preguta si un vehículo está pasando en dirección opuesta (Se presionó S2 primero). En caso de ser así, borra la bandera de vehículo en dirección opuesta y retorna.

Ahora, si la bandera de OPPOS_DIR no estaba en alto, se pregunta si ya había sido accionado el botón anteriormente o no. Esto es para contemplar el caso en que pasen dos vehículos por S1, antes de que el primer vehículo llegue a S2, en cuyo caso debe ignorar al segundo vehículo. En caso de que es el primer vehículo ($PH3\_FIRED$ = 0), la subrutina resetea $TICK\_VEL$, y pone la bandera de $PH3\_FIRED$ en 1. Adicionalmente, carga la variable para el contador de rebotes.

La subrutina deberá **suprimir cualquier rebote** del sensor $PH3$, por lo cuál esperará **40ms** para volver a escuchar cualquier otro botón.

Luego de los **40ms**, es posible que se **accione S2**. En cuyo caso, deberá preguntarse si ya algún vehículo accionó previamente S1 ($PH3\_FIRED = 1$). En caso de no ser así, se deberá ignorar este accionamiento, pues significa que es un vehículo pasando en dirección contraria. Adicionalmente, tendrá que poner la bandera OPPOS_DIR en alto, para indicar a S1 que ignore el siguiente accionamiento.

Si previamente un vehículo había accionado S1, entonces, se deberá **cambiar el mensaje** del LCD. Para esto debemos habilitar las interrupciones dentro de la subrutina, para que las subrutinas de LCD puedan utilizar la subrutina DELAY, que depende de $OC4\_ISR$ para decrementar un contador.

Seguidamente, se realizará el **cálculo de la velocidad**, utilizando la fórmula 2.5 (Ver desarrollo). Posterior al cálculo, debemos preguntarnos si la velocidad cálculada es mayor que 255, en cuyo caso debemos limitar el valor de $VELOC$ a 255. Si es menor, guardamos el valor correspondiente en la variable $VELOC$. Y finalmente, debemos de resetar la bandera de $PH3\_FIRED$ para indicar que el vehículo salió de la zona de medición.

Parámetros

- **Parámetros de entrada**. Esta subrutina no tiene parámetros de entrada.
- **Parámetros de salida**.
  - **VELOC**. Esta subrutina realiza el cálculo de la velocidad, y guarda el resultado en la variable $VELOC$.

Diagrama

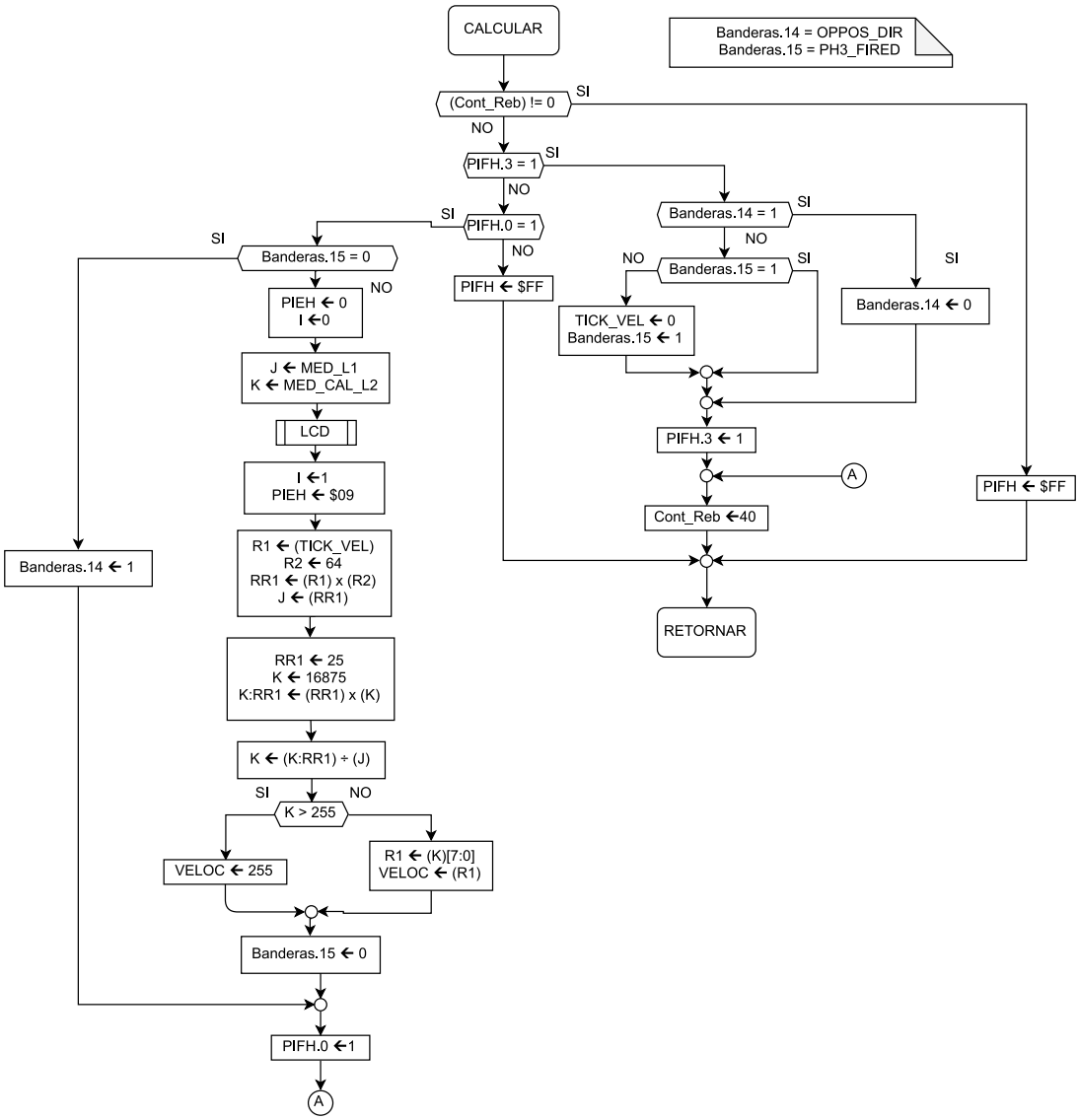

Figura 2.5: Diagrama de flujos para CALCULAR. Elaboración propia.

2.6. Subrutina RTI_ISR

2.6.1. Cálculos para interrupción RTI_ISR

Para esta subrutina, queremos que se ejecute cada 1ms. Entonces, tenemos la siguiente fórmula:

$$T_{RTI} = \frac{(N+1) \cdot 2^{M+9}}{OSC\_CLK} \tag{2.7}$$

Haciendo un poco de retrospección, OSC_CLK tiene un valor de $8MHz$, por lo que en el numerador necesitamos un número muy cercano a este valor.

Recordando que $2^{13} = 8192 \approx 8 \cdot 10^3$, entonces tendríamos que:

$$T_{RTI} = \frac{(0+1) \cdot 2^{4+9}}{8 \cdot 10^6} = 1{,}024ms$$

Entonces, con $N = 1$ y $M = 4$ logramos nuestro objetivo.

2.6.2. Configuración de RTI_ISR

La configuración para este periferico es bastante sencilla en realidad. Básicamente, lo primero es configurar el tiempo de T_{RTI} con los valores calculados anteriormente. Esto sería:

$$RTICTL = \$40$$

Lo segundo, habilitar el puerto de RTI. Esto último se hace con la siguiente configuración:

$$CRGINT = \$80$$

2.6.3. Vector de interrupción RTI_ISR

El vector de interrupción se encuentra en la dirección $\$3E70$ para el Debug12.

2.6.4. Diagrama y explicación de la subrutina RTI_ISR

Descripción

Esta subrutina atiende la interrupción generada por el módulo RTI. Se utiliza esta interrupción para realizar el decremento del contador de rebotes $Cont\_Reb$, cada 1ms.

Explicación

Como se puede observar en la figura 2.6, esta subrutina es bastante simple: Decrementa el contador de $Cont\_Reb$ siempre y cuando el mismo no sea cero. Se utiliza en las subrutinas $CALCULAR$ y $TAREA\_TECLADO$ para realizar la supresión de rebotes de las teclas mecánicas.

Parámetros

- **Parámetro de entrada y salida**.
  - **Cont_Reb**. Esta subrutina toma el valor de esta variable, y lo decrementa siempre que no sea cero.

Diagrama

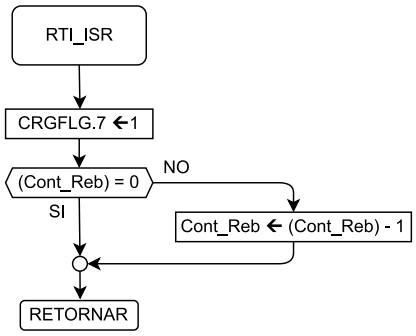

Figura 2.6: Diagrama de flujos para RTI_ISR. Elaboración propia.

2.7. Subrutina OC4_ISR

2.7.1. Cálculos para interrupción OC4_ISR

Tenemos que usar un Preescalador de 8 para la interrpución por overflow de $TCNT\_ISR$. Entonces, haciendo los cálculos para la cantidad de TICKS que tenemos que contar, para que $OC4\_ISR$ se ejecute con una frecuencia de $50kHz$ serían los siguiente:

$$TCS = \frac{20\mu s \cdot 24MHz}{PRS} \tag{2.8}$$

Esto significa entonces que, usando $PRS = 8$, encontramos que:

$$TCS = 60$$

Cada interrupción entonces, debemos cargar $TC4$ con $TCNT + 60$.

NOTA: Para realizar cuentas de $1ms$ por ejemplo, debemos contar hasta 50.

Cuentas de $100ms$ (Para llamar a $CONV\_BIN\_BCD$ y BCD_7SEG) se realizan contando hasta 5000.

Cuentas de $200ms$ (Para llamar a $PATRON\_LEDS$ y $ATD0\_CTL5$) se realizan contando hasta 10000.

2.7.2. Configuración de OC4_ISR

Para el primer registro de configuración $TSCR1$, queremos habilitar el módulo de Timers (bit TEN) y además, queremos habilitar la bandera de $TFFCA$.

Esto quiere decir que, la bandera de $C4F$ se va a borrar cuando se escriba un dato en $TC4$, y además, la bandera de TOF se va a borrar cuando se lea el registro de $TCNT$.

Entonces, para este primer registro, tenemos que:

$$TSCR1 = \$90$$

Luego tenemos el segundo registro de control, $TSCR2$, en donda vamos a configurar el preescalador con el valor de 8. Esto significa guardar un 3 en los bits de PRS. Por lo tanto, tenemos que:

$$TSCR2 = \$03$$

NOTA: Para habilitar o deshabilitar las interrupciones por rebase, la configuración se hace por este mismo registro, en el bit 7 del registro. Sin embargo no están habilitadas por defecto. Sólo se habilitan cuando se está en el modo medición.

Siguiendo con la configuración, para Output Compare, tenemos que habilitar la opción de output compare para el canal 4, en el registro de $TIOS$. Entonces, tenemos que:

$$TIOS = \$10$$

Y además, tenemos que habilitar la interrupción para cuando la bandera de $C4F$ se levanta. Esto se hace en el registro de configuración llamado TIE, en donde tenemos entonces que:

$$TIE = \$10$$

2.7.3. Vector de interrupción OC4_ISR

El vector de interrupción para $OC4\_ISR$ se encuentra en la dirección $\$3E66$.

2.7.4. Diagrama y explicación de la subrutina OC4_ISR

Descripción

Esta subrutina se encarga de controlar los LEDS, desde calcular el brillo periódicamente, realizar la conversión de los valores BIN1 y BIN2 a los códigos que lleva cada display de 7 segmentos, la multiplexación de los displays y realizar la conversión A/D periódicamente. Finalmente, se encarga de llamar periódicamente a la subrutina de $PATRON\_LEDS$.

Explicación

Empezando por la primera parte, como se puede observar en la figura 2.7, lo primero que se hace es volver a cargar el valor para *output compare*. Posteriormente, se decrementa la variable $Cont_Delay$ siempre que la misma no sea cero.

A continuación, se procede a verificar por cuál *tick* (CONT_TICKS) se encuentra **el display actual**. Si se encuentra en el último tick (número 100), se incrementa el dígito actual (display actual), y se realiza el cálculo de la variable DT. En caso de cualquier otro número de tick, se incrementa el valor.

Seguidamente, se revisa si el valor de CONT_TICKS es igual a cero, en cuyo debemos de cambiar el display activo para el ciclo actual.

Además, si no es cero, se verifica si el valor de CONT_TICKS es igual al DT, en cuyo caso se apaga el display actual (para el control del brillo).

La **segunda parte del diagrama** se puede observar en la figura 2.8. En este diagrama se realiza la configuración correspondiente para el dígito en el que se encuentra (0,1,2,3 son DISP y 4 es LEDS).

Y la **tercera parte del diagrama** (figura 2.9), realiza la ejecución de las subrutinas CONV_BIN_BCD y BCD_7SEG, en ese orden cada 100ms. Además, cada 200ms, hace un ciclo de conversión A/D y un llamado a la subrutina PATRON_LEDS.

Diagrama

Parámetros

- **Parámetros de entrada**.
  - **Cont_Delay**. Variable utilizada para realizar un bloqueo por software en la subrutina DELAY.
  - **BRILLO**. Variable utilizada para controlar el duty cycle de cada display.
- **Parámetros de salida**. Esta subrutina no tiene parámetros de salida.

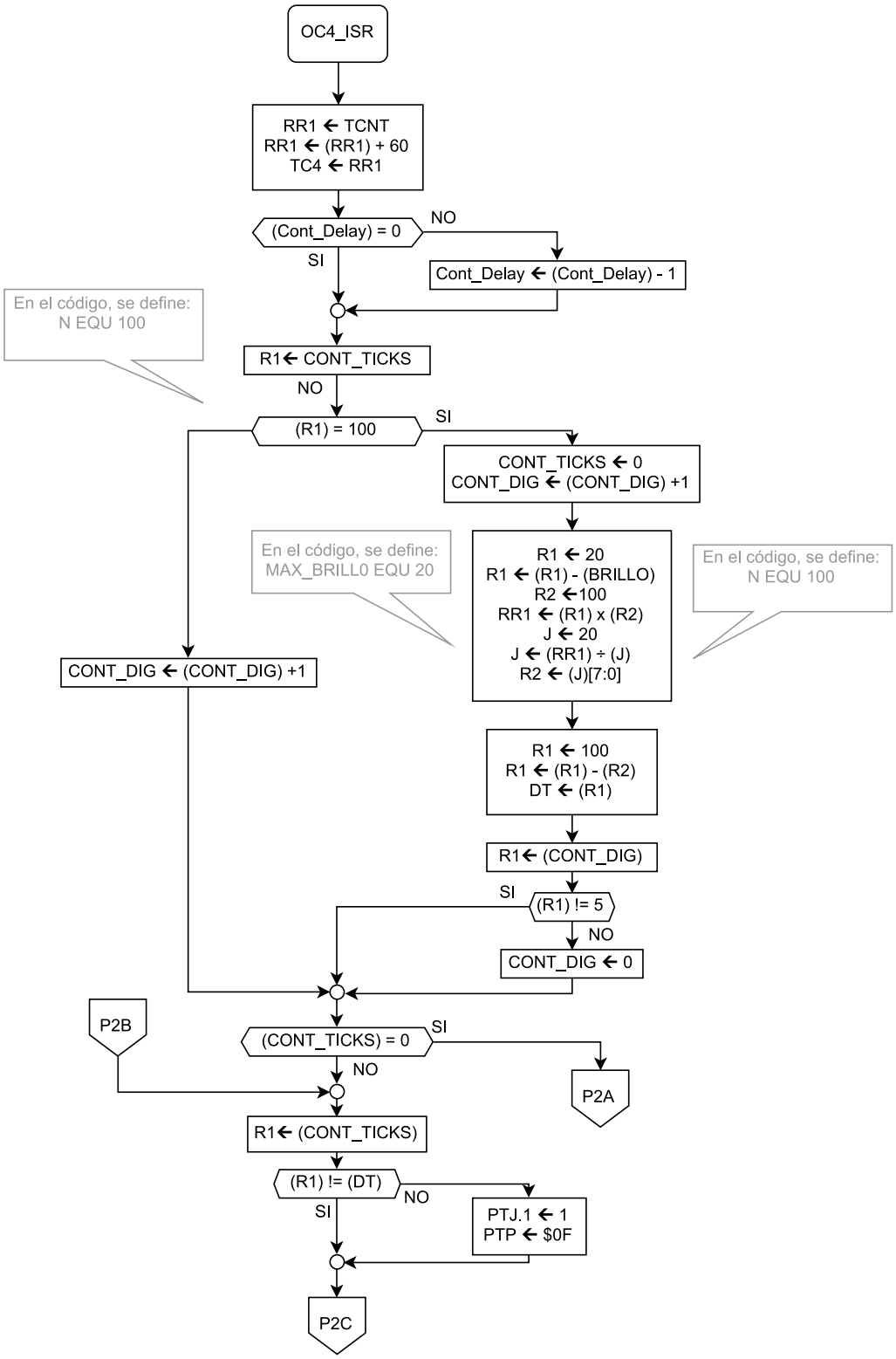

Figura 2.7: Diagrama de flujos para OC4_ISR. Elaboración propia. Primera parte

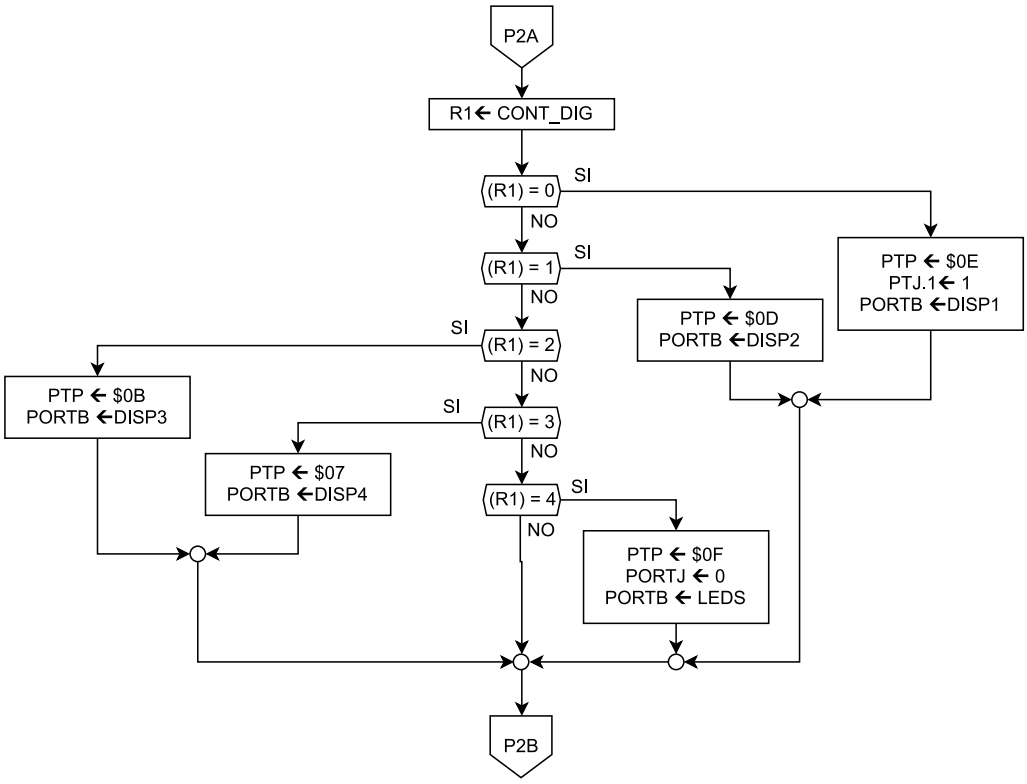

Figura 2.8: Diagrama de flujos para OC4_ISR. Elaboración propia. Segunda parte

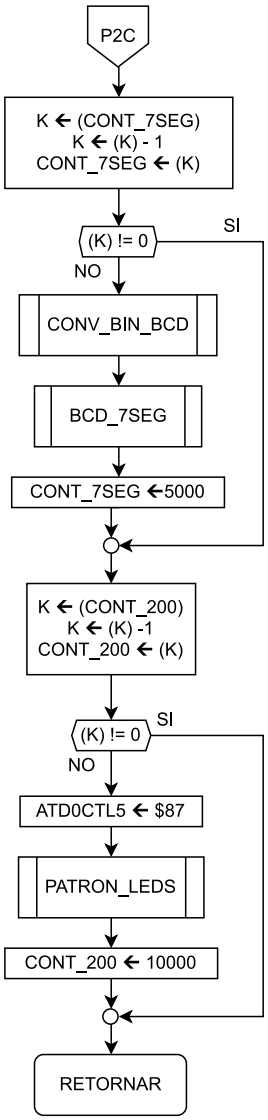

Figura 2.9: Diagrama de flujos para OC4_ISR. Elaboración propia. Tercera parte

2.8. Subrutina MUX_TECLADO

2.8.1. Diagrama y explicación de la subrutina MUX_TECLADO

Descripción

Esta subrutina se encarga de encontrar la tecla presionada del teclado matricial.

Explicación

Como se puede observar en la figura 2.11, esta subrutina es bastante sencilla. Se utiliza la variable Patron para determinar cuál patron se debe escribir. A continuación, dependiendo del valor de Patron, se escribirá en el puerto A (PORTA) el patrón con alguna de las cuatro líneas en 0. La idea, es iterar línea por línea, y ver si hay algún botón presionado en esa línea.

Como se observa en el diagrama de flujos (figura 2.10), **lo siguiente que se hace** justamente es verificar si se presionó algúin botón en alguna de las primeras tres columnas. De ser así, utilizando el valor de Patron, y el contenido del registro K, se determina el **índice** de la tabla Teclas, correspondiente a la tecla presionada. Y por último, con este índice, se guarda el contenido correspondiente en la variable Tecla.

Note que en caso de que Patron esté fuera del rango (Patron = 4), entonces, se guardará un $FF en la variable Tecla.

Parámetros

- **Parámetros de entrada**. Esta subrutina no tiene parámetros de entrada.
- **Parámetros de salida**.
  - **Tecla**. Si hay una tecla presionada, se guardará en la variable Tecla el valor correspondiente a la tecla presionada. Si no hay ningúna tecla presionada, se guardará un $FF en la variable Tecla.

Diagrama

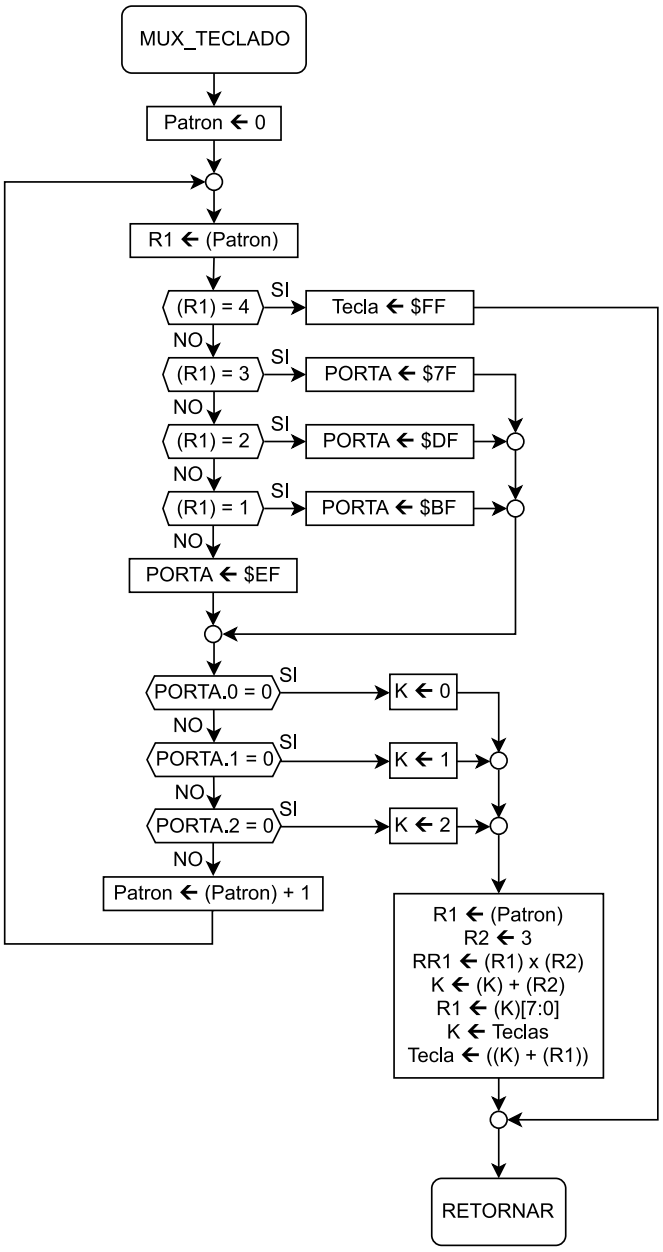

Figura 2.10: Diagrama de flujos para MUX_TECLADO. Elaboración propia.

2.9. Subrutina TAREA_TECLADO

2.9.1. Diagrama y explicación de la subrutina TAREA_TECLADO

Descripción

Esta subrutina se encarga de realizar la función de supresión de rebotes, e implementar la funcionalidad de *tecla retenida*.

Explicación

Como se puede observar en la figura 2.11, esta figura realiza llamados a la subrutina MUX_TECLADO siempre y cuando la variable *Cont_Reb* sea cero (Esto se hace para suprimir rebotes). Ahora, posterior a este llamado, se verifica si la variable *Tecla* contiene un valor váido, o indica que no ha sido presionada ninguna tecla. De ser el primer caso, se realiza la lógica de supresión de rebotes.

La lógica de **supresión de rebotes** consiste en: La primera vez que se detecta una tecla, guarda este valor en la variable Tecla_in, activar la variable *Cont_Reb* para esperarse 10ms, y poner la bandera de TCL_LEIDA. En el caso de que ya se entre a esta lógica por segunda vez (Se usa la bandera de TCL_LEIDA para determinar esto), entonces, se verifica que la Tecla presionada sea la misma que se guardó en Tecla_in, y de ser así se enciende la bandera de TCL_LISTA. De no ser así, se regresan todas las variables a su estado inicial.

La lógica de **verificar que ninguna tecla fué presionada**, verifica antes de retornar, si la banderas de TCL_LISTA está activa. De ser así, llama a la subrutina FORMAR_ARRAY para procesar esta tecla.

Diagrama

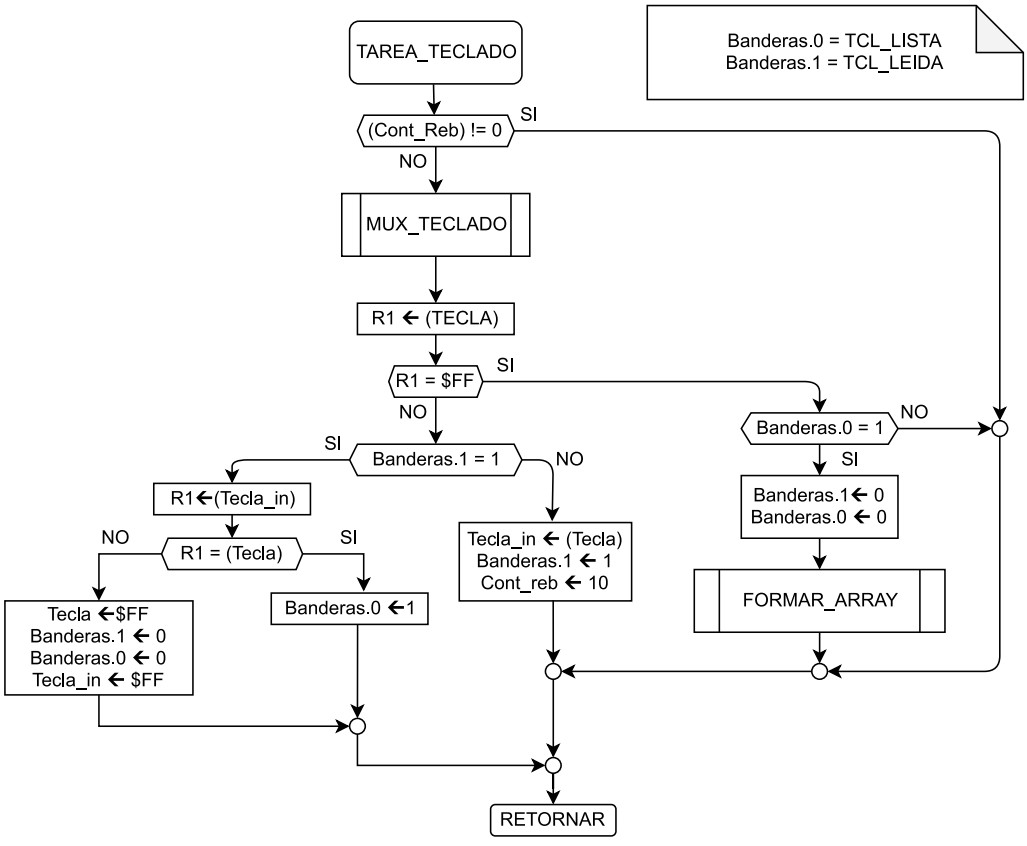

Figura 2.11: Diagrama de flujos para TAREA_TECLADO. Elaboración propia.

2.10. Subrutina FORMAR_ARRAY

2.10.1. Diagrama y explicación de la subrutina FORMAR_ARRAY

Descripción

Esta subrutina se encarga de guardar la secuencia de teclas presionadas en el arreglo Num_Array. Una vez que se termina la secuencia de lectura (Se presiona enter, y una cantidad válida de teclas había sido presionada previamente), entonces esta subrutina enciende la bandera de ARRAY_OK (Se ubica en Banderas.2).

Explicación

Esta subrutina, se encarga de leer el valor de Tecla_in, para averiguar cuál fué la tecla que se presionó recientemente. Ahora, verifica que sucedan varias cosas:

- Que la tecla *enter* sea ignorada, sin antes de haberla presionado, no se presionó alguna tecla válida (Del 0 al 9).
- Que la tecla *borrar*, sea ignorada sin antes de haberla presionado, no se presionó alguna tecla válida (Del 0 al 9).
- Que, si se presiona la tecla *enter* y ya se habían presionado teclas válidas anteriormente, que se levante la bandera de ARRAR_OK.
- Que, si se presiona la tecla *borrar* y ya se habían presionado teclas válidas anteriormente, que se borre la última tecla presionada.
- Que, si se presiona una tecla válida, pero ya se llegó al límite de MAX_TCL, entonces se ignore esa tecla.
- Finalmente, si es una tecla válida, y no se ha llegado al límite, que se registre la tecla, guardandola en el arreglo Num_Array.

Se puede observar en la figura 2.12 que la lógica anteriormente descrita, es la que está implementada en el código. Se utiliza la variable Cont_TCL para determinar la posición actual en el arreglo Num_Array.

Parámetros

- **Parámetros de entrada**. Esta subrutina no tiene parámetros de entrada.
- **Parámetros de salida**.
  - **Num_Array**. En este arreglo se guardarán las teclas válidas que fueron registradas.
  - **ARRAY_OK**. Esta bandera se enciende para indicar que se terminó una secuencia de lectura del teclado. Esta banderas se encuentra en Banderas.2.

Diagrama

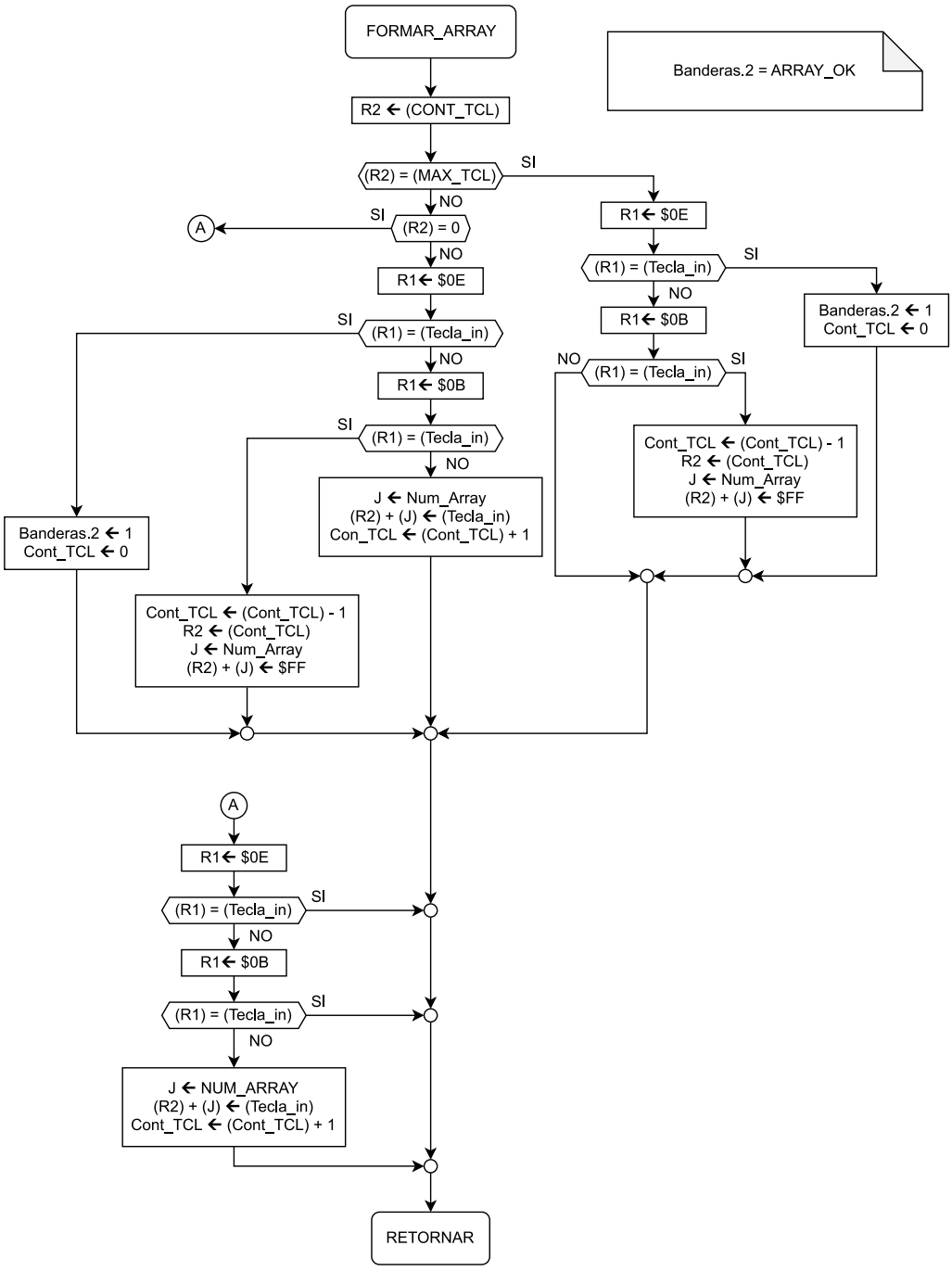

Figura 2.12: Diagrama de flujos para FORMAR_ARRAY. Elaboración propia.

2.11. Subrutina MODO_MEDICION

2.11.1. Diagrama y explicación de la subrutina MODO_MEDICION

Descripción

Esta subrutina implementa el modo Medición del proyecto RADAR 623. Se encarga de cambiar los mensajes LCD, y activar los periféricos necesarios para este modo. Además, cada vez que se detecte que una velocidad no es cero, (Se detectó un vehículo entre S1 y S2), se realizará una llamada a la subrutina de PANT_CTRL.

Explicación

Como se puede observar en la figura 2.13, se utiliza la bandera de STATE_CHANGED, para determinar si esta es la primera vez que se entra en este modo, desde el programa principal o no. En caso de ser así, se realizan las siguientes tareas:

- Se habilita la interrupción por overflow del registro TCNT (TCNT_ISR).
- Se habilita la funcionalidad de key-wakeups en el puerto H para PH3 y PH0.
- Se actualiza el modo actual a desplegar en los LEDS.
- Se borran los displays de 7 segmentos.
- Se cambia el mensaje de la pantalla LCD.
- Se define la bandera CALC_TICKS en 1. Posteriormente, esta bandera en PANT_CTRL es utilizada para verificar si es la primera vez que VELOC != 0, por lo que se debe realizar los cálculos de TICK_EN y TICK_DIS.

Parámetros

- **Parámetros de entrada**.
- **VELOC**. Esta subrutina utiliza el valor de VELOC para determinar si tiene que llamar a la subrutina de PANT_CTRL o no.
- **Parámetros de salida**.
  - **LEDS**. En esta variable, se configura el LED correspondiente que indica que se encuentra en este modo.
  - **BIN1**. Utiliza esta variable para controlar el contenido de uno de los displays de 7 segmentos.
  - **BIN2**. Utiliza esta variable para controlar el contenido de uno de los displays de 7 segmentos.

Diagrama

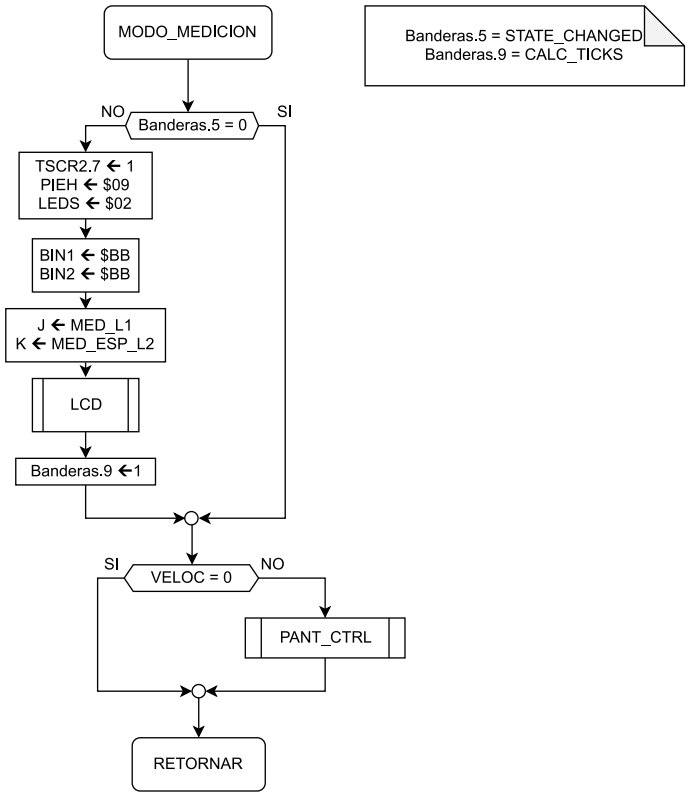

Figura 2.13: Diagrama de flujos para MODO_MEDICION. Elaboración propia.

2.12. Subrutina PANT_CTRL

2.12.1. Cálculos para PANT_CTRL

Lo primero, para lograr que $TICK\_DIS$ llegue a cero en dos segundos, entonces tenemos que cargar:

$$TICK\_DIS = 2seg \cdot \frac{46875}{1024} = 91{,}55 \approx 92$$

Ahora, en general, dado una velocidad $VELOC$, la cantidad de ticks que debemos cargar en contar para que el carro haya avanzado una distancia en metros DISTANCIA, se calcula de la siguiente manera:

$$TICKS = \frac{DISTANCIA}{VELOC} \cdot \frac{1km}{1000m} \cdot \frac{3600s}{1h} \cdot \frac{46875}{1024} \tag{2.9}$$

Ahora, particularmente, para calcular $TICK\_EN$, la $DISTANCIA = 60m$. Entonces tendríamos lo siguiente:

$$TICK\_EN = \frac{100m \cdot 1km \cdot 3600 \cdot 46875}{VELOC \cdot 1000m \cdot 1h \cdot 1024}$$

$$= \frac{16479{,}4921875}{VELOC}$$

$$\approx \frac{16480}{VELOC}$$

Por otro lado, para calcular $TICK\_DIS$, simplemente es el doble de $TICK\_EN$. Es decir:

$$TICK\_DIS = \frac{32958{,}984375}{VELOC}$$

$$\approx \frac{32959}{VELOC}$$

2.12.2. Diagrama y explicación de la subrutina PANT_CTRL

Descripción

Esta subrutina se encarga de orquestrar toda la secuencia de acciones a tomar después de que se detecta un vehículo en los sensores S1 y S2. Desde cambiar los mensajes de la pantalla, encargarse de calcular los TICKS_EN y TICKS_DIS, de encender la alarma, de controlar los displays de 7 segmentos, y de desactivar los botones S1 y S2 mientras se está procesando este vehículo.

Explicación

El diagrama de flujos para este programa se puede observar en la figura 2.14. Lo primero que hace esta subrutina, es desactivar los sensores S1 y S2, para evitar que otro vehículo distorsione la secuencia de el primer vehículo.

Después, se pregunta si ya se realizó el cálculo de *ticks*, o no. **En caso de no haberse realizado**, la subrutina se pregunta si la velocidad está fuera del rango $[30, 99]$ o no. En caso de estar fuera del rango, se encarga de poner una bandera (Bandera OUT_RANGE), para que posteriormente se ponga la velocidad límite y guiones en los displays de 7 segmentos.

Si la velocidad está dentro del rango, verifica si está por encima de la velocidad límite o no. En caso de estar por encima, se enciende la bandera de ALERTA (Banderas.4), y se procede a, en base a la velocidad del vehículo, calcular los valores necesarios de TICK_EN y TICK_DIS.

Finalmente, pone en cero la bandera de CALC_TICKS, para que la lógica de cálculo no se vuelva a ejecutar, hasta que no termine de pasar el vehículo.

En caso de que **ya se realizaron los cálculos** previamente, entonces vamos a verificar si la bandera de PANT_FLAG está encendida o no. Recordando que esta bandera la enciende la subrutina TCNT_ISR cuando TICK_EN llega a 0, vemos que hay varios posibles casos. Para el caso en donde esta bandera **PANT_FLAG NO está encendida**, tenemos en particular dos casos:

- La bandera PANT_FLAG está en cero porque TICK_EN no ha llegado a 0. Lo que significa que debemos esperar.
- La bandera PANT_FLAG está en cero porque TICK_EN y TICK_DIS, ambos ya llegaorn a 0. Lo que significa que debemos volver al estado de esperar un vehículo.

Para diferenciar estos dos casos, se utiliza el contenido de los displays de 7 segmentos. En caso de estar en blanco, estamos en el primer caso, por lo que no debemos hacer nada más que esperar. Si no están vacíos, es porque previamente se escribió algo, por lo que estamos en el segundo caso, y debemos de regresar al estado de esperar.

Esto último significa: Cambiar el mensaje de la pantalla LCD, borrar el contenido de los displays de 7 segmentos, borrar la velocidad, re-habilitar los sensores S1 y S2, y regresar la bandera de OUT_RANGE, CALC_TICKS y ALERTA a su estado pasivo.

Ahora, para el caso en donde **PANT_FLAG está encendida**, tenemos dos posibles casos nuevamente:

- La bandera PANT_FLAG está en uno porque TICK_EN ha llegado a 0 y es la primera vez que entramos en este caso. Lo que significa que debemos poner en pantalla el mensaje correspondiente, y ponerle al usuario en el display de 7 segmentos la velocidad límite y su velocidad (podría ser guiones si está fuera del rango. Esto último se determina con la bandera de OUT_RANGE).
- La bandera PANT_FLAG está en uno porque TICK_EN ha llegado a 0, pero ya actualizamos el contenido del LCD. En este caso, debemos esperar hasta que se encienda la banderas de PANT_FLAG. La manera en que se determina si ya se actualizó el contenido, es verificar el valor $BIN1$.

Finalmente, después de manejar el caso actual, se retorna de la subrutina.

Parámetros

- **Parámetros de entrada**.
  - **VELOC**. Esta subrutina utiliza el valor de VELOC para determinar los valores de TICK_EN y TICK_DIS.
- **Parámetros de salida**.
  - **BIN1**. Utiliza esta variable para controlar el contenido de uno de los displays de 7 segmentos.
  - **BIN2**. Utiliza esta variable para controlar el contenido de uno de los displays de 7 segmentos.
  - **TICK_EN**. Esta variable se utiliza en la subrutina de TCNT_ISR para determinar cuando se tiene que encender la bandera PANT_FLAG.
  - **TICK_DIS**. Esta variable se utiliza en la subrutina de TCNT_ISR para determinar cuando se tiene que apagar la bandera PANT_FLAG.
  - **OUT_RANGE**. Esta bandera indica si la velocidad está fuera del rango o no.

- **CALC_TICKS**. Esta bandera indica si se deben realizar los cálculos de ticks y de ALERTA y OUT_RANGE o no.
- **ALERTA**. Esta bandera indica si se tiene que encender la alera o no. Depende de la velocidad del vehículo.

Diagrama

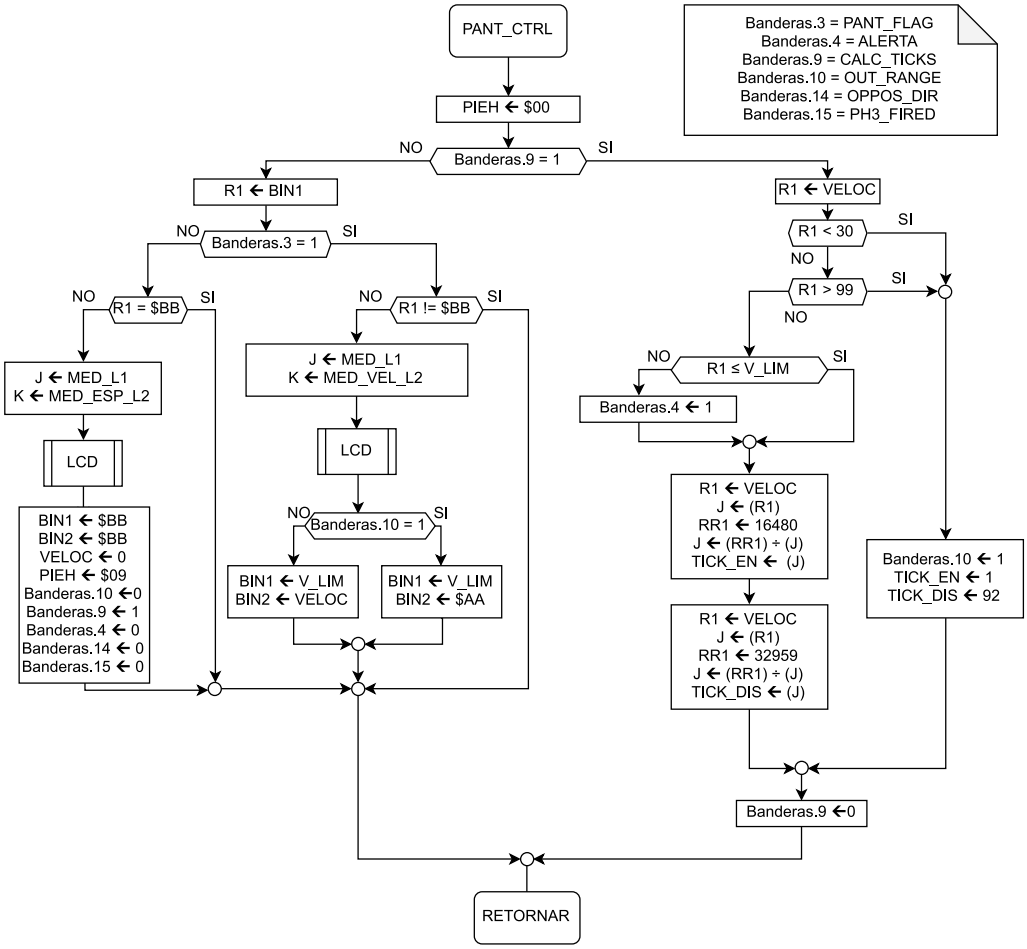

Figura 2.14: Diagrama de flujos para PANT_CTRL. Elaboración propia.

2.13. Subrutina MODO_CONFIG

2.13.1. Diagrama y explicación de la subrutina MODO_CONFIG

Descripción

Esta subrutina implementa el MODO_CONFIG de la aplicación. En este modo se configura el valor de V_LIM, mediante el uso del teclado matricial.

Explicación

De acuerdo al diagram de flujos de la figura 2.15, lo primero que se hace, es **verificar si es la primera vez que se encuentra en este estado o no**.

De ser así, se cambia el mensaje de del LCD, se realiza el cambio de modo en los LEDS, y adicionalmente se pone el valor actual de V_LIM en BIN1. Hay que recordar, que en el programa principal, antes de entrar a este modo se realizan otra serie de configuraciones, como se puede observar en la figura 2.2. Estas configuraciones son comúnes entre MODO_CONFIG y MODO_LIBRE, por esto están fuera de la subrutina. Están relacionadas a desactivar periféricos, borrar los displays de 7 segmentos, y reestablecer banderas.

Seguidamente, esta subrutina se encarga de estar llamando a la subrutina de TAREA_TECLADO, hasta que se detecte que ARRAR_OK esté en 1. Una vez que esto pasa, quiere decir que se **terminó una secuencia de lectura** en el teclado matricial, por lo que se procederá a validar el dato ingresado por el usuario.

El valor que ingresó el usuario, primero se debe convertir a binario, tomándolo del arreglo *Num_Array*. Para esto utilizamos la subrutina BCD_BIN. Seguidamente, se **verificará si el valor ingresado por el usuario** está dentro del **rango** permitido ($[45 - 90]$). Si está dentro del rango, el valor de V_LIM es actualizado. Y en caso de no estar dentro del rango, se descarta este valor ingresado por el usuario, se constituye el valor de V_LIM por el valor antiguo, y finalmente se prepara Num_Array para recibir otra secuencia de lectura del teclado matricial.

Parámetros

- **Parámetros de entrada**.
  - **ARRAY_OK**. Esta es la bandera de ARRAY_OK, y esta subrutina se encarga de modificar esta bandera, para forzar a TAREA_TECLADO a volver a iniciar el proceso de leer teclas. La subrutina FORMAR_ARRAY se encarga de actualizar este valor.
  - **STATE_CHANGED**. Esta bandera indica si el estado cambio. Es decir, esta es la primera vez que se entra a MODO_CONFIG.
- **Parámetros de salida**.
  - **BIN1**. Utiliza esta variable para controlar el contenido de dos de los displays de 7 segmentos. En este display se desplegará el valor de V_LIM.
  - **V_LIM**. Esta subrutina es la encargada de definir V_LIM, y verificar que el valor está en un rango permitido.

Diagrama

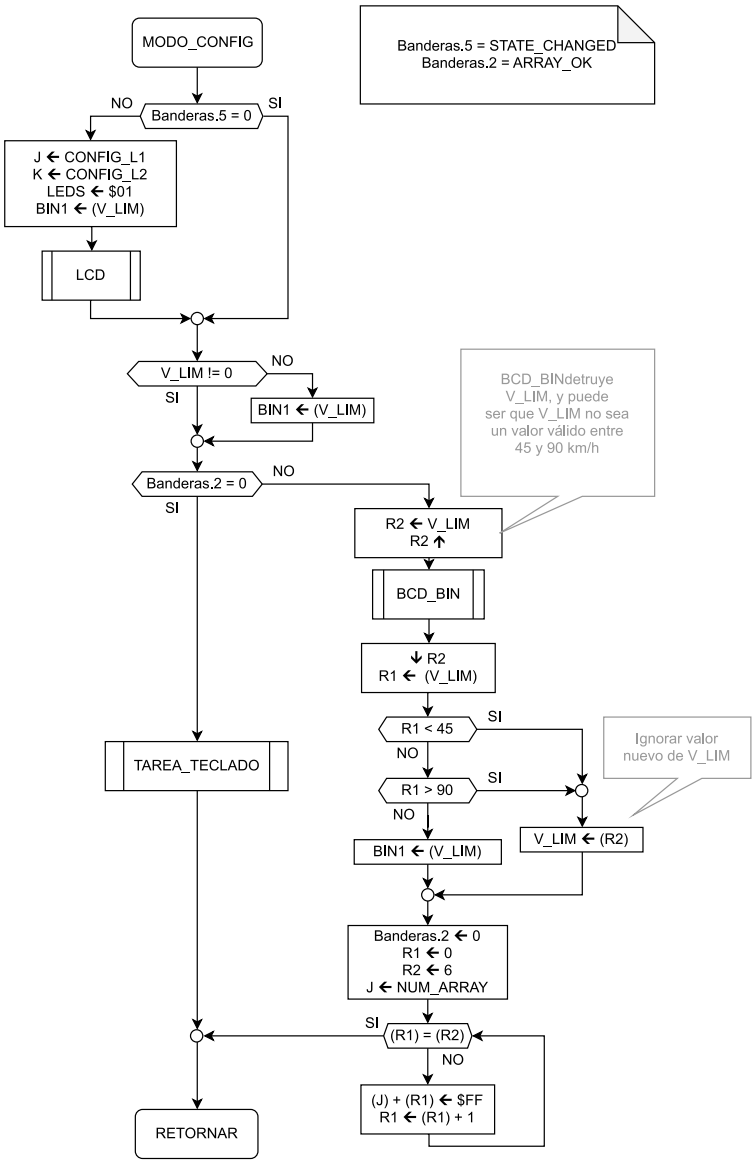

Figura 2.15: Diagrama de flujos para MODO_CONFIG. Elaboración propia.

2.14. Subrutina MODO_LIBRE

2.14.1. Diagrama y explicación de la subrutina MODO_LIBRE

Descripción

Esta subrutina implementa el modo libre de la aplicación.

Explicación

Como se puede observar en la figura 2.16, esta subrutina sólo cumple básicamente una tarea. En caso de que sea la primera vez que se entra en este modo, cambiará el contenido del LCD. Adicionalmente, en el programa principal (ver figura 2.2) se apagan ciertos periféricos, los displays de 7 segmentos, y se restablecen ciertas banderas.

Parámetros

- **Parámetros de entrada**.
  - **STATE_CHANGED**. Esta bandera indica si el estado cambio. Es decir, esta es la primera vez que se entra a MODO_LIBRE.
- **Parámetros de salida**. Esta subrutina no tiene parámetros de salida.

Diagrama

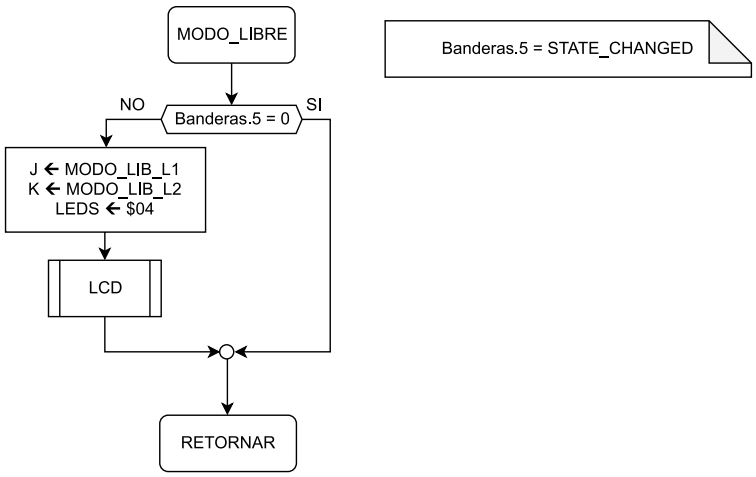

Figura 2.16: Diagrama de flujos para MODO_LIBRE. Elaboración propia.

2.15. Subrutina BCD_BIN

2.15.1. Diagrama y explicación de la subrutina BCD_BIN

Descripción

Esta subrutina convierte el valor de Num_Array y lo guarda en V_LIM.

Explicación

Como se puede observar en la figura 2.17, esta subrutina toma byte por byte que encuentra en el arreglo de Num_Array, y lo convierte a binario, guardándolo en V_LIM. Para esto, la subrutina primero verifica la cantidad de valor válidos que tiene Num_Array. Podría ser 1 valor válido (sólo

un digito) o dos valores válidos (una decena y una unidad). Dependiendo del caso, le asigna un peso distinto a cada posición de Num_Array.

Posteriormente, toma el contenido de la posición, y lo multiplica por el peso correspondiente de esa posición en el arreglo. Con cada iteracción, suma el resultado parcial (Se va guardando en V_LIM) con el nuevo resultado. Una vez que llega al final del arreglo, retorna.

Parámetros

- **Parámetros de entrada**.
  - **Num_Array**. Esta subrutina lee el valor en BCD que está guardado a partir de la dirección de Num_Array.
- **Parámetros de salida**.
  - **V_LIM**. Esta subrutina convierte el valor en BCD que está en Num_Array, y lo guarda en binario en la variable V_LIM.

Diagrama

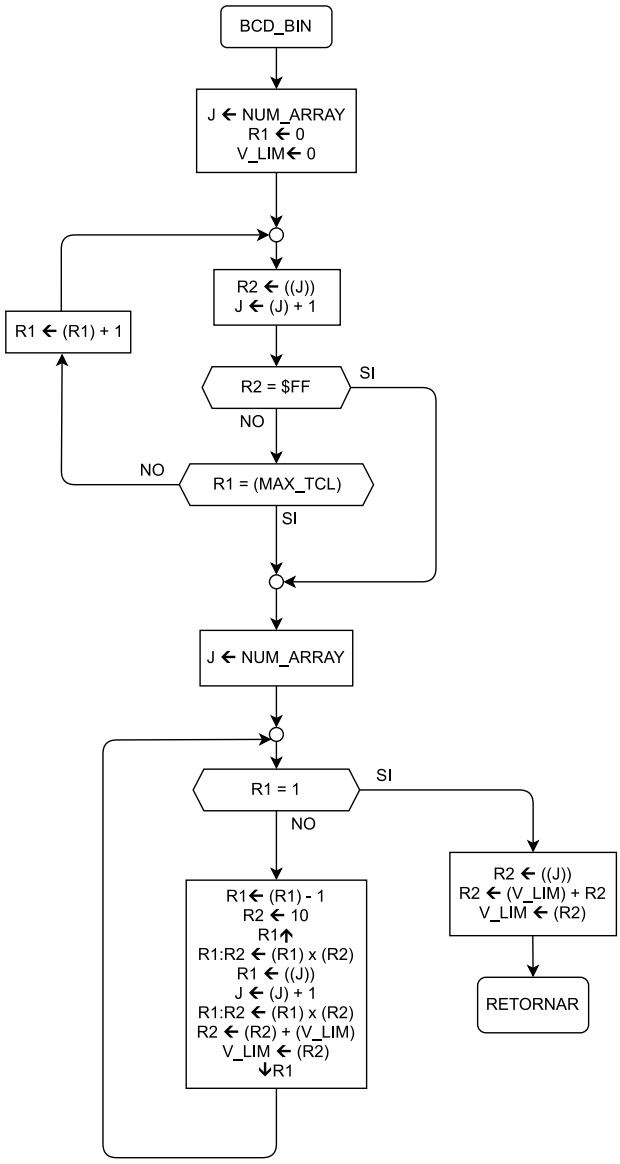

Figura 2.17: Diagrama de flujos para BCD_BIN. Elaboración propia.

2.16. Subrutina CONV_BIN_LCD

2.16.1. Diagrama y explicación de la subrutina CONV_BIN_LCD

Descripción

Subrutina utilizada para convertir dos números binarios dos números en BCD.

Explicación

Esta subrutina (ver figura 2.18) utiliza la subrutina BIN_BCD para convertir dos veces dos números binarios diferentes a BCD. Estos dos números son: BIN1 y BIN2 y son guardados en BCD como BCD1 y BCD2.

Hay dos casos especiales. Cuando la subrutina encuentra que hay un $AA o un $BB en alguna de las dos subrutinas, entonces guardará exactamente el mismo valor en la variable de BCD correspondiente.

Parámetros

- **Parámetros de entrada**.
  - **BIN1**. Primer número binario que se desea cambiar a un número en BCD. El resultado será guardado en BCD1.
  - **BIN2**. Segundo número binario que se desea cambiar a un número en BCD. El resultado será guardado en BCD2.
- **Parámetros de salida**.
  - **BCD1**. Posición en memoria donde se guarda el resultado para el primer número en BCD.
  - **BCD2**. Posición en memoria donde se guarda el resultado para el segundo número en BCD.

Diagrama

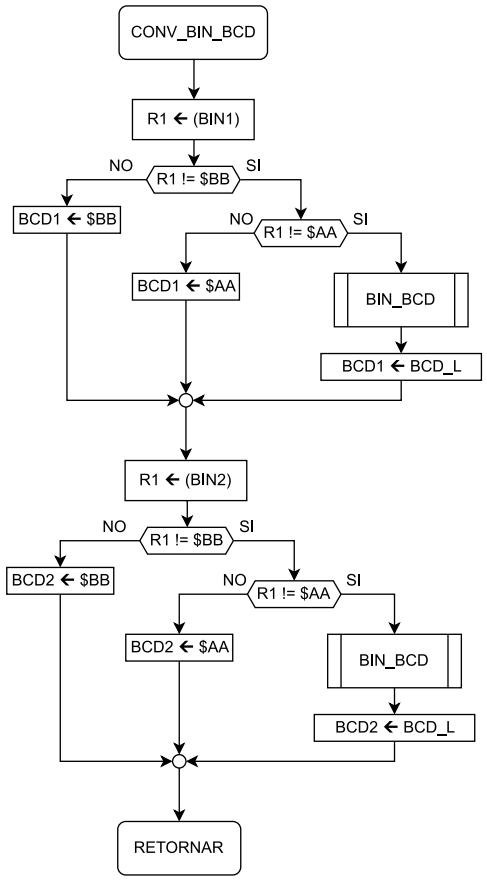

Figura 2.18: Diagrama de flujos para CONV_BIN_LCD. Elaboración propia.

2.17. Subrutina BIN_BCD

2.17.1. Diagrama y explicación de la subrutina BIN_BCD

Descripción

Subrutina general para convertir un número binario a BCD.

Explicación

Esta subrutina (ver figura 2.19), toma un valor en binario por medio del registro R1 (A), y lo convierte a 3 nibbles en BCD. Los dos nibbles inferiores, se guardan en la variable BCD_L, mientras que el nibble superior se guarda en la parte inferior de la variable BCD_H.

El algoritmo de esta subrutina se tomó del curso de microprocesadores, según lo visto en [1].

Parámetros

- **Parámetros de entrada**.

- **R1**. Por medio del registro A se envía el número binario que se desea convertir a BCD.

- **Parámetros de salida**.
  - **BCD_H**. Guarda, en el nibble inferior, el tercer dígito (centenas) del número binario.
  - **BCD_L**. Guarda, en el nibbler inferior, las unidades del número binario. Y en el nibble del medio, guarda las decenas del número binario.

Diagrama

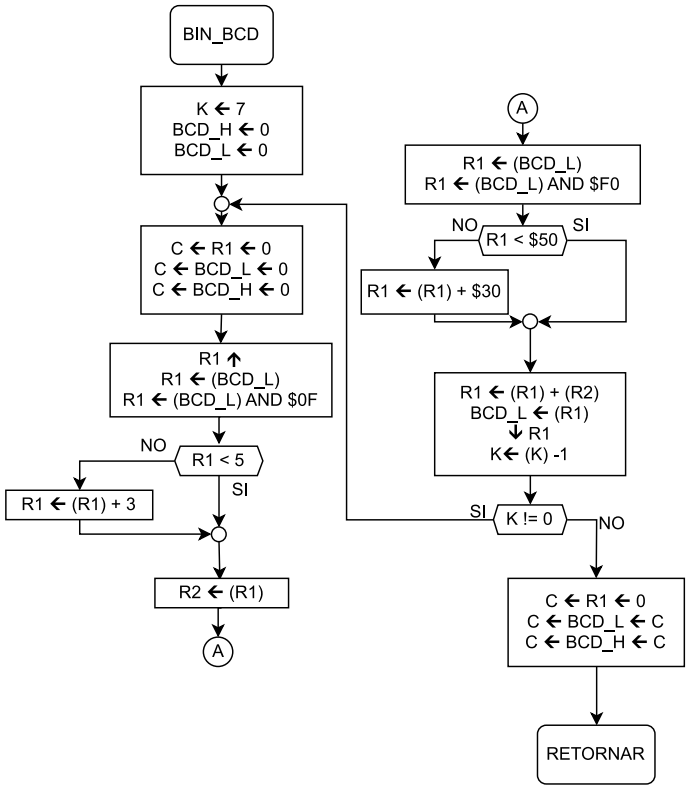

Figura 2.19: Diagrama de flujos para BIN_BCD. Elaboración propia.

2.18. Subrutina BCD_7SEG

2.18.1. Diagrama y explicación de la subrutina BCD_7SEG

Descripción

Subrutina encargada actualizar los valores de los Displays.

Explicación

Esta subrutina (ver figura 2.20), se encarga de tomar los valores que se encuentran en en BCD1 y BCD2 y separarlos en dos (nibble superior e inferior). La subrutina utiliza el valor de los nibbles

como offset en la tabla SEGMENT para encontrar el byte que genera el patron correspondiente para cada DISP.

El resultado se guarda en DISP1 y DISP2 correspondientemente para el BCD1, y en DISP3 y DISP4 correspondientemente para el BCD2.

Parámetros

- **Parámetros de entrada**.
  - **BCD1**. Posición en memoria donde se guarda el resultado para el primer número en BCD.
  - **BCD2**. Posición en memoria donde se guarda el resultado para el segundo número en BCD.
- **Parámetros de salida**.
  - **DISP1**. Display correspondiente al nibble superior de BCD2.
  - **DISP2**. Display correspondiente al nibble inferior de BCD2.
  - **DISP3**. Display correspondiente al nibble superior de BCD1.
  - **DISP4**. Display correspondiente al nibble inferior de BCD1.

Diagrama

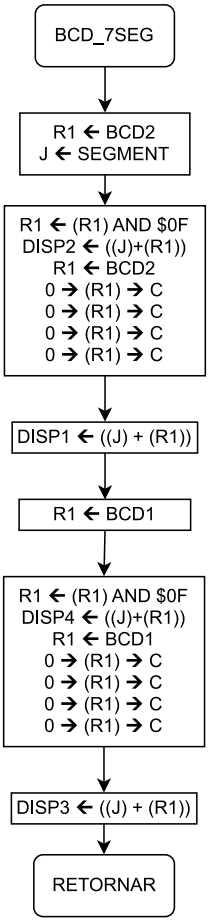

Figura 2.20: Diagrama de flujos para BCD_7SEG. Elaboración propia.

2.19. Subrutina DELAY

2.19.1. Diagrama y explicación de la subrutina DELAY

Descripción

Subrutina que realiza un bloqueo por una cierta cantidad de tiempo.

Explicación

Como se puede observar en la figura 2.21, esta subrutina es bastante simple. Lo único que hace es esperarse hasta que la variable Cont_Delay sea cero. La subrutina Oc4_ISR decrementa el valor de ese contador a una cadencia de $20\mu s$. Con esto, una subrutina que desee crear cierto delay, carga el valor correspondiente en el contador de Cont_Delay, y llama a esta subrutina.

Parámetros

- **Parámetros de entrada**.

- **Cont_Delay**. Esta variable es decrementada por la subrutina OC4_ISR. Esta subrutina se espera hasta que el valor de esta variable sea cero, lo cuál significará que el programa se esperará una cierta cantidad de tiempo.

- **Parámetros de salida**. Esta subrutina no tiene parámetros de salida.

Diagrama

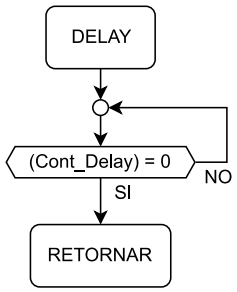

Figura 2.21: Diagrama de flujos para DELAY. Elaboración propia.

2.20. Subrutina SEND

2.20.1. Diagrama y explicación de la subrutina SEND

Descripción

Subrutina utilizada para enviar un comando o un dato al periférico de LCD.

Explicación

El diagrama de flujos de esta subrutina se encuentra en la figura 2.22. Con la bandera de SEND_CMD or SEND_DATA (Banderas.8), esta subrutina envía el contenido de R1 al display LCD, primero envíando el nibble superior, y luego el nibble inferior.

El algoritmo utilizado en esta subrutina fué tomado de la presentación de Perifericos del Curso IE-623 [2].

Parámetros

- **Parámetros de entrada**.
  - **R1**. Por el registro R1, se pasa tanto el nibble superior como el nibble inferior de el dato o el comando que se va a enviar.
  - **Send_CMD || Send_DATA**. Esta bandera se encuentra en el Banderas.8. Indica si se va a enviar un comando o se va a enviar un dato. 0 significa comando, y 1 significa dato.
- **Parámetros de salida**. Esta subrutina no tiene parámetros de salida.

Diagrama

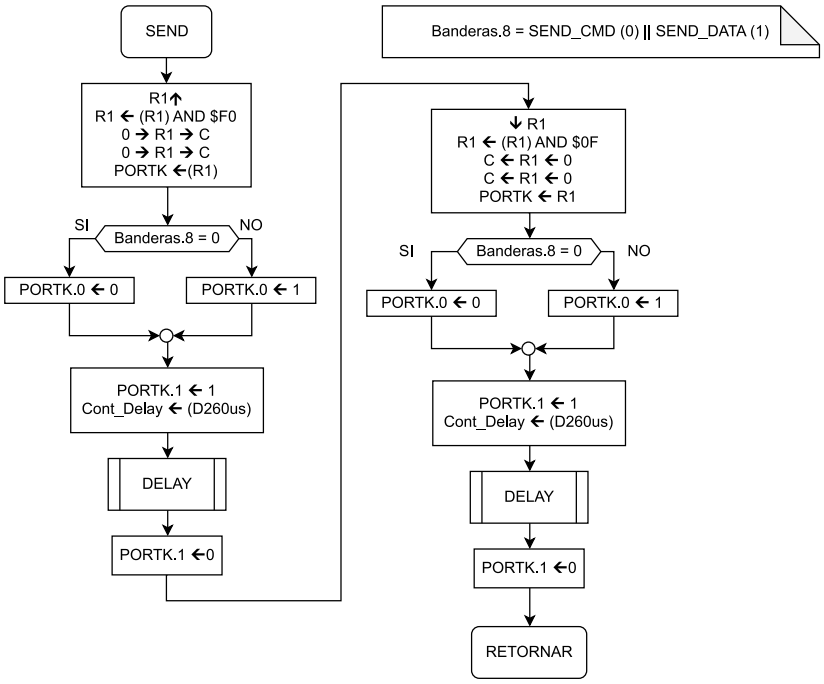

Figura 2.22: Diagrama de flujos para SEND. Elaboración propia.

2.21. Subrutina LCD

2.21.1. Diagrama y explicación de la subrutina LCD

Descripción

Subrutina que se encarga de configurar el periférico de la pantalla LCD, enviando la secuencia de comandos para inicialización.

Explicación

Esta subrutina (ver figura 2.23) se encarga de enviar todos los bytes de la tabla iniDISP, y posteriormente, limpia la pantalla LCD, y pone el mensaje apuntado por los registros J y K. (J para la línea superior y K para la línea inferior).

Nuevamente, el algoritmo de esta subrutina, fué tomado de la la presentación de Perifericos del Curso IE-623 [2].

Parámetros

- **Parámetros de entrada**.
  - **J**. Por medio de este registro se pasa la dirección del la tabla que contiene los caractéres a escribir en la primera línea del LCD.

- **K**. Por medio de este registro se pasa la dirección del la tabla que contiene los caractéres a escribir en la segunda línea del LCD.

- **Parámetros de salida**. Esta subrutina no tiene parámetros de salida.

Diagrama

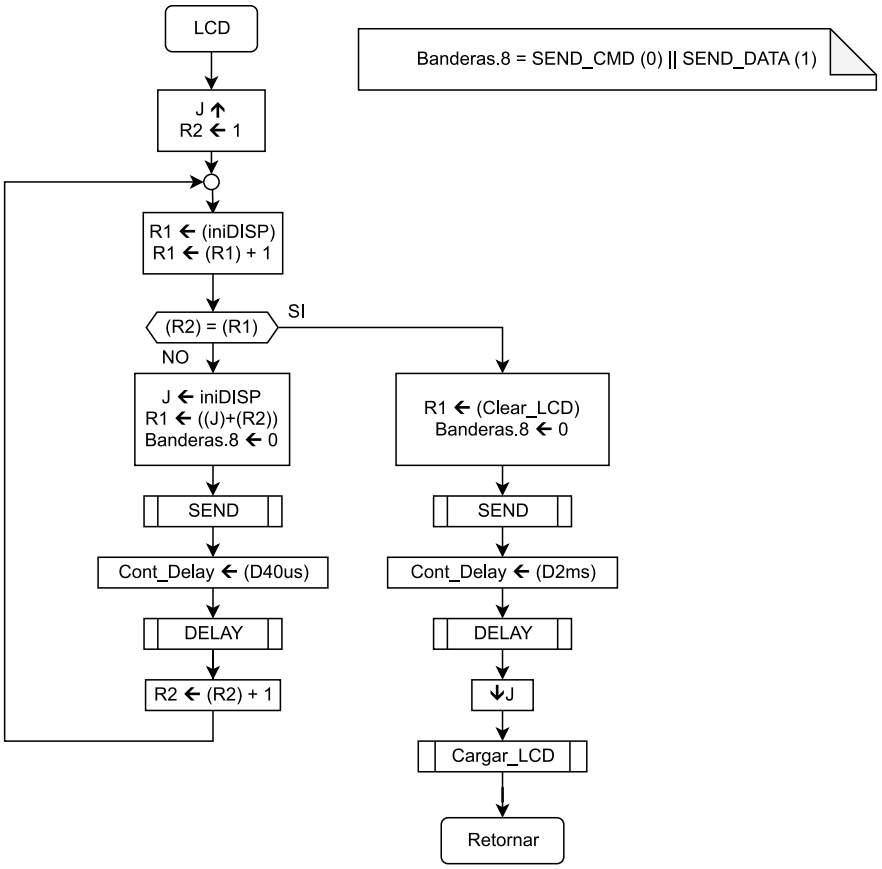

Figura 2.23: Diagrama de flujos para LCD. Elaboración propia.

2.22. Subrutina Cargar_LCD

2.22.1. Diagrama y explicación de la subrutina Cargar_LCD

Descripción

Esta subrutina se encarga de enviar todos los datos de dos tablas al periférico LCD. Las dos tablas corresponden a la primera línea y a la segunda línea.

Explicación

El diagrama de flujos de esta subrutina se puede observar en la figura 2.24.

La subruina se encargará de llamar a la subrutina SEND para enviar un byte de datos a la vez. Además, se encargará de realizar los DELAYs correspondientes para cada envío de datos, y realizar el cambio de línea con las constantes ADD_L1 y ADD_L2, cuando corresponda.

Nuevamente, el algoritmo de esta subrutina, fué tomado de la la presentación de Perifericos del Curso IE-623 [2].

Parámetros

- **Parámetros de entrada**.
  - **J**. Por medio de este registro se pasa la dirección del la tabla que contiene los caractéres a escribir en la primera línea del LCD.
  - **K**. Por medio de este registro se pasa la dirección del la tabla que contiene los caractéres a escribir en la segunda línea del LCD.
- **Parámetros de salida**. Esta subrutina no tiene parámetros de salida.

Diagrama

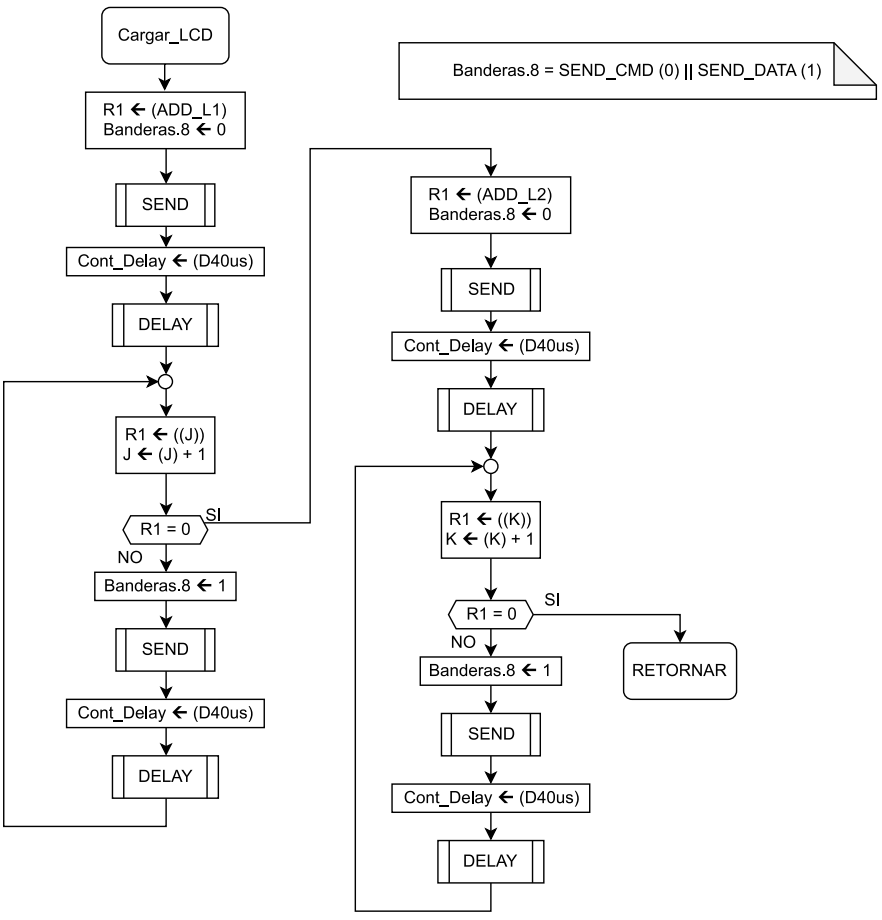

Figura 2.24: Diagrama de flujos para Cargar_LCD. Elaboración propia.

2.23. Subrutina PATRON_LEDS

2.23.1. Diagrama y explicación de la subrutina PATRON_LEDS

Descripción

Esta subrutina se encarga de manejar los LEDS de Alerta, dependiendo de si la bandera de Alerta está encendida o no.

Explicación

Cuando la bandera de ALERTA (bit 4 de Banderas) está en 1, cada vez que se llame a esta subrutina, barrera los leds del 7 al 3, de izquierda a derecha. Sin embargo, si ALERTA está en 0, esta subrutina borrará los LEDS del 7 al 3.

Como se puede ver en la figura 2.25, esto se logra revisando el valor de LEDS. Si **es mayor que 7**, quiere decir que ya se encendió alguno de los LEDS que corresponden a la alerta (Cualquier bit del 4 en adelante, hará que LEDS sea mayor que 7).

Ahora, si esto es así, quiere decir que debemos correr hacia la derecha el leds que está encendido. Para lograr eso, guardamos temporalmente en R1 el contenido de los primeros tres bits (modo), y realizamos un corrimiento a la derecha sobre el registro de LEDS. Borramos los primeros tres bits, y sumamos el contenido del registro R1 para volver a restaurar el modo en el registro de LEDS.

En caso de que **LEDS no sea mayor que 7**, quiere decir que estamos en el caso en donde todos los bits de Alerta están apagados. Por lo que corresponde es poner el bit más significativo en 1 en este ciclo, para volver a realizar el corrimiento bit a bit.

Esto se repite siempre que la bandera de ALERTA esté encendida.

Parámetros

- **Parámetros de entrada**.
  - **ALERTA**. Esta bandera se encuentra en Banderas.4, e indica si la Alerta está encendida o no.
- **Parámetros de salida**. Esta subrutina no tiene parámetros de salida.

Diagrama

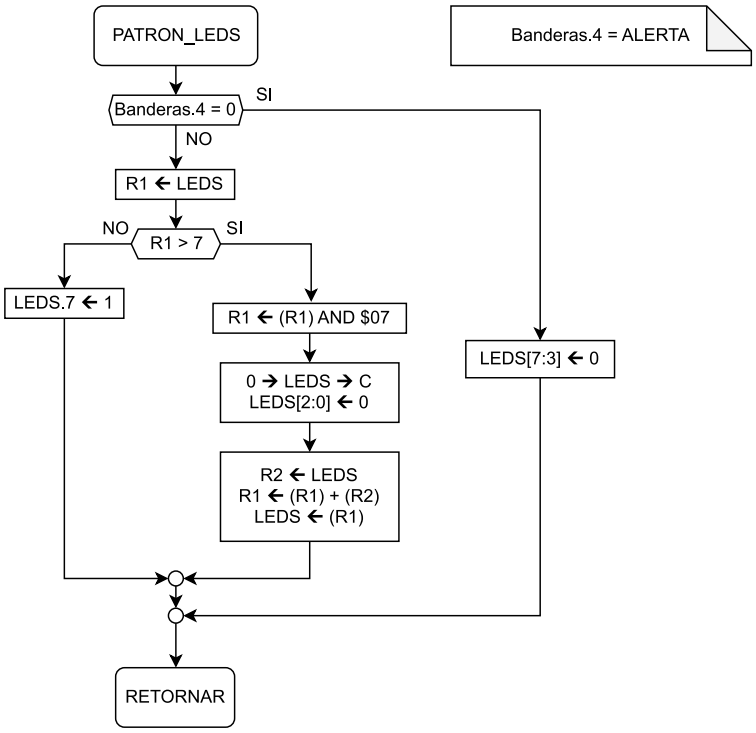

Figura 2.25: Diagrama de flujos para PATRON_LEDS. Elaboración propia.

3 Conclusiones y recomendaciones

3.1. Conclusiones

Algunos de los puntos más importantes concluidos durante el desarrollo de este proyecto:

- Realizar el diseño de una aplicación real es muy complejo. Realizar el desarrollo es un poco menos complicado, aunque encontrar errores en la implementación puede ser difícil si no se tiene una buena metodología para encontrarlos.
- El manejo de la teoría es primordial tanto para el diseño como para realizar *debugging*. Con el conocimiento a mano es muy fácil descartar posibles hipótesis de fuentes de error, y centralizarse en las posibilidades factibles.
- Realizar *debugging* de subrutinas que dependen del tiempo puede volverse muy complicado, puesto que implica que para verificar el funcionamiento real, ir línea por línea no simula las condiciones reales de la subrutina.

3.2. Recomendaciones

Las recomendaciones a partir de cada una de las conclusiones anteriores serían:

- Realizar pruebas de cada subrutina implementada, y mantener un control de versiones por subrutina implementada es una buena estrategia. Devolverse en estas versiones y realizar cambios en versiones anteriores facilita mucho la implementación de la aplicación.
- Repasar la teoría de los periféricos antes de realizar el diseño resulta esencial en desarrollar el diseño eficientemente.
- Para las subrutinas que dependen del tiempo (TAREA_TECLADO, CALCULAR, etc) es útil utilizar variables para guardar un *snapshot* de lo acontecía en ciertos momentos de la subrutina. También utilizar banderas para hacer *debugging* es muy útil (Como poner un contador temporal para contar cuántas veces entra a cierta sección del código en CALCULAR).

Bibliografía

[1] Delgado, G., “Microprocesadores IE-623. I Parte”, Escuela de Ingeniería Eléctrica, San José, San Pedro, Universidad de Costa Rica, 2019.

[2] Delgado, G., “Microprocesadores IE-623. II Parte. Dispositivos Periféricos”, Escuela de Ingeniería Eléctrica, San José, San Pedro, Universidad de Costa Rica, 2019.

[3] Delgado, G., “Proyecto Final. RADAR 623.”, Escuela de Ingeniería Eléctrica, San José, San Pedro, Universidad de Costa Rica, 2019.