Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Industriales

Sistemas Microprocesadores

Memoria del Trabajo

Programación mediante Microchip Studio de un Túnel de Lavado Automático

María del Mar Martín Diaz - 18207

Celia Ramos Ramírez - 18295

Juan Galvañ Hernando - 18125

Mario Gómez Pozo - 18151

Gonzalo Quirós Torres - 17353

Josep Mª Barberá Civera - 17048

21 de mayo de 2022

Índice

I Introducción

La presente memoria recoge los aspectos principales sobre la programación mediante *Microchip Studio* de un túnel de lavado automático. El control de dicha maqueta se ha realizado con un microcontrolador de 8 bits con arquitectura AVR, en concreto el ATmega640.

Como controlador de versiones se ha empleado *Github*, puede encontrarse el código empleado en este repositorio. Además de *Microchip Studio*, también se ha empleado *VSCode* para el desarrollo del software.

II Particionado

Enunciamos ahora cada una de las partes en las que está dividido el trabajo y una tabla adjunta con la distribución de estas entre los integrantes del grupo.

- **Parte 1 (P1):** Control de los rodillos de lavado horizontal y secado para seguir los perfiles de los vehículos.
- **Parte 2 (P2):** Control de la barrera de entrada, rodillo de lavado vertical y de la luz de indicación de estado (luz periódica -de medio segundo- con destellos cada 10 segundos o intermitente cada segundo).
- **Parte 3 (P3):** Control de la cinta de arrastre, semáforo, parada de emergencia e integración del sistema

Parte 1	Parte 2	Parte 3
Mario Gómez (18151)	María del Mar Martín(18207)	Gonzalo Quirós (17353)
Juan Galvañ (18125)	Celia Ramos (18295)	Josep Mª Barberá (17048)

Tabla 1: Distribución del trabajo

Puede verse en figura 1 las zonas anteriormente mencionadas en referencia a la maqueta.

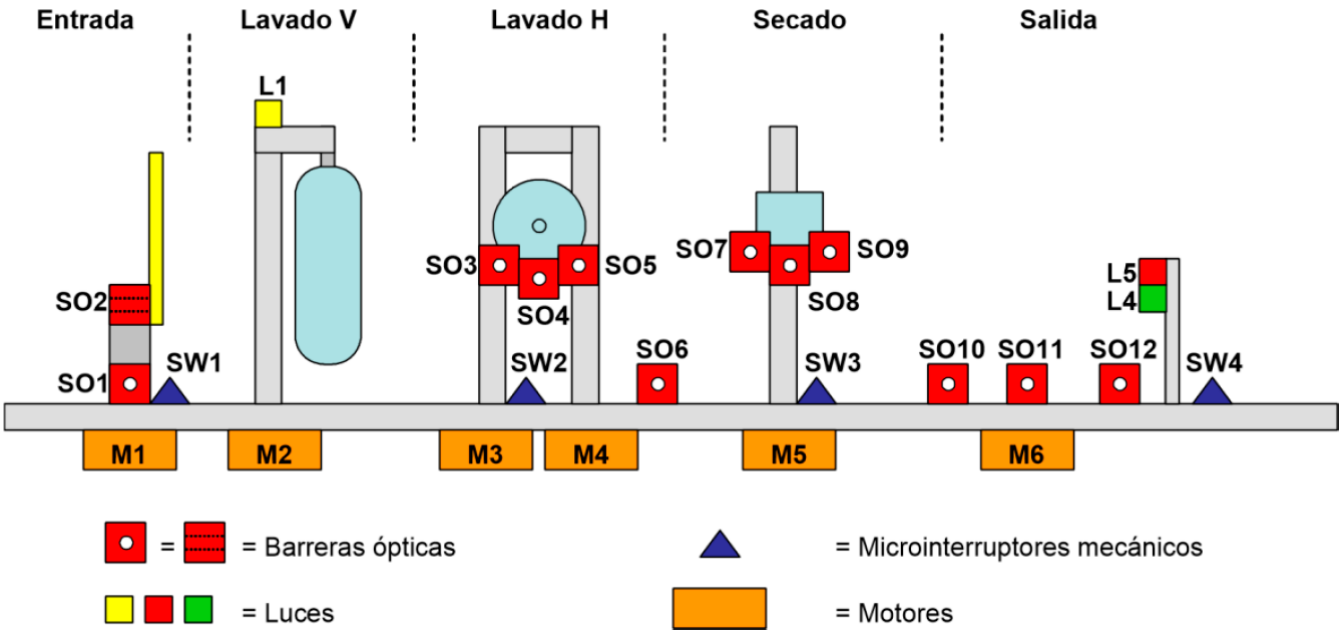

Figura 1: Zonas de la maqueta. Imagen cortesía del CEI.

III

Planteamiento de la solución

3.1 Entrada

- **Descripción:** control de la barrera de entrada.
- **Pareja:** Mar Martín, Celia Ramos.
- **Tiempo empleado:** 7.5 horas.
- **Archivos fuente:** *entrada.c*, *entrada.h*.

La entrada incluye los siguientes dispositivos: el sensor óptico SO1 (activo por nivel bajo -al igual que todos los demás sensores-), el sensor SO2, el microinterruptor SW1 y el motor M1. Puede verse un resumen de los actuadores por cada sección de la maqueta en la parte superior de la tabla 2. La gestión de cada uno de estos sensores y actuadores se ha realizado del siguiente modo:

- **SO1:** se le ha asignado una interrupción externa tipo INT. En concreto la INT1. Puede verse un ejemplo de su uso e implementación en el código 1.
- **SO2:** se consulta de forma periódica para comprobar si la barrera está cerrada o abierta.
- **SW1:** se le ha asignado una interrupción externa tipo INT. En este caso la INT0. Puede verse la implementación en 2.
- **M1:** se enciende y apaga mediante el puerto correspondiente (ver tabla 3).

Para mayor claridad se ha realizado un flujograma de la interrupción del sensor SO1 (ver Fig. 2).

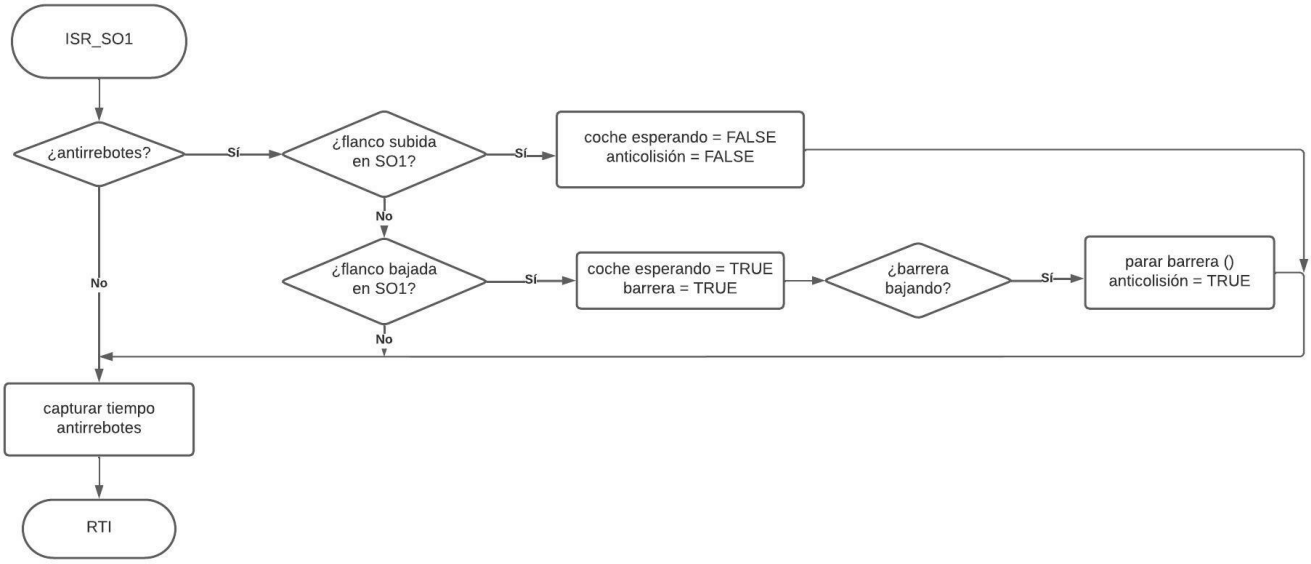

Figura 2: Interrupción del sensor óptico SO1 para el control de la barrera de entrada.

```
ISR (INT1_vect)
{
   if (miliseconds - antireb_SO1 > SENSOR_DELAY) // antirrebotes del sensor
   optico
   {
      if (SO1_f) // Acaba de dejar de detectar (flanco de subida)
      {
         carWaiting = FALSE;
         antiCollision = FALSE;
      }
      else if(!SO1_f) // Empieza a detectar (flanco de bajada)
      {
         carWaiting = TRUE;
         barrier = TRUE;
         if (barrierPulseCounter > 3) // si la barrera esta bajando
         {
            // y el coche intenta entrar de nuevo la barrera se para
            barrierStop();
            antiCollision = TRUE;
         }
      }
      antireb_SO1 = miliseconds; // se captura el tiempo  para el antirrebotes
   }
}
```

Código 1: Interrupción para el sensor óptico SO1

```
ISR(INT0_vect)
{   // antirrebotes del microinterruptor
    if ((miliseconds - antireb_SW1) > BOTON_DELAY)
    {   // se lleva la cuenta de los pulsos de SW1
        barrierPulseCounter++ ;
        antireb_SW1 = miliseconds;  // se captura el tiempo  para el antirrebotes
    }

    if (barrierPulseCounter == 4)   // barrera esta levantada
    {   // se controla el estado de la barrera con una bandera
        barrierUp = TRUE;
      // capturamos el tiempo para el inicio del Lavado Vertical
      secondsLV = half_second;
    }
    else
    {
        barrierUp = FALSE;
    }
   if (barrierPulseCounter == 5)
   {   // capturamos el tiempo para apagar el Lavado Vertical
      secondsLVOff = half_second;
   }
    if (SO2_f == FALSE)
    {
        barrierDown = TRUE;
      secondsLVOff = half_second;
      barrierPulseCounter = 0;
    }
}
```

Código 2: Interrupción para el microinterruptor SW1

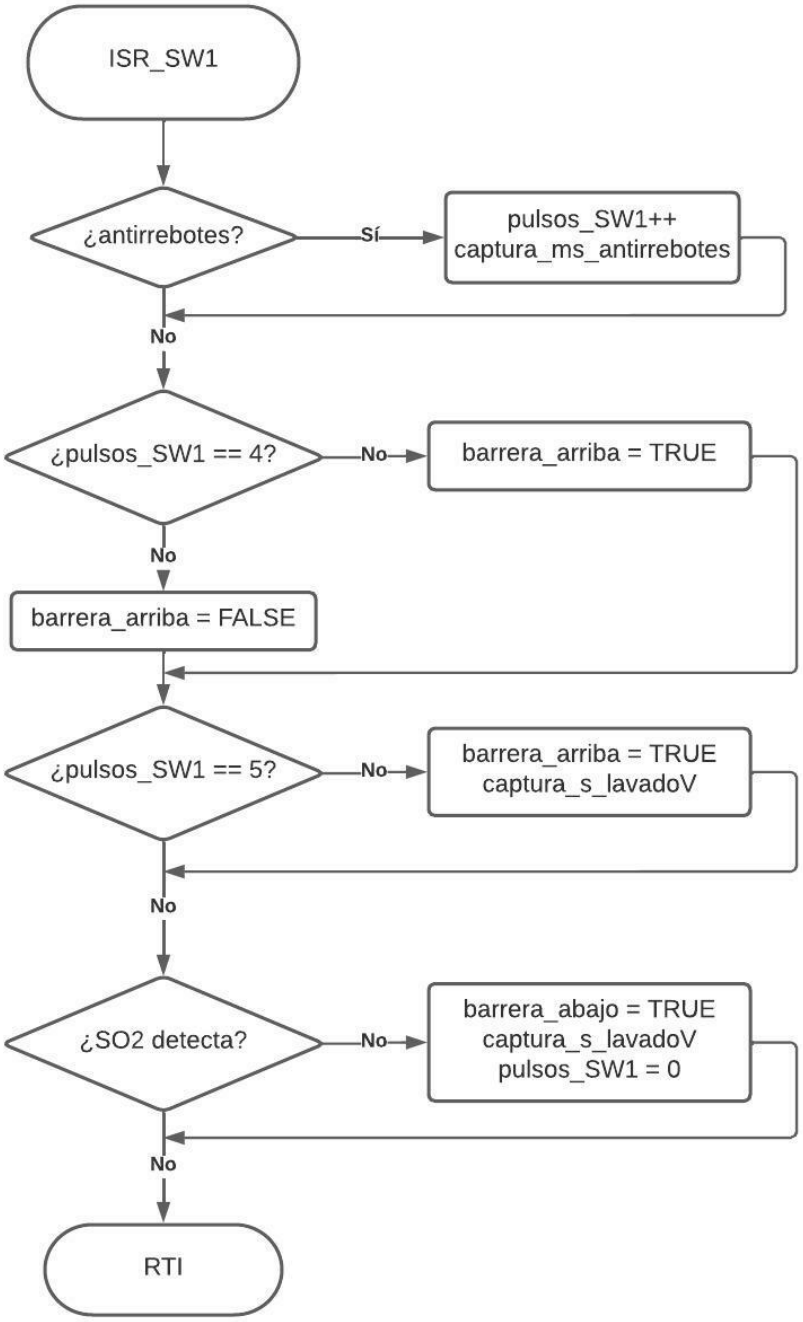

Figura 3: Interrupción del microinterruptor SW1 para el control de la barrera de entrada mediante la cuenta de pulsos.

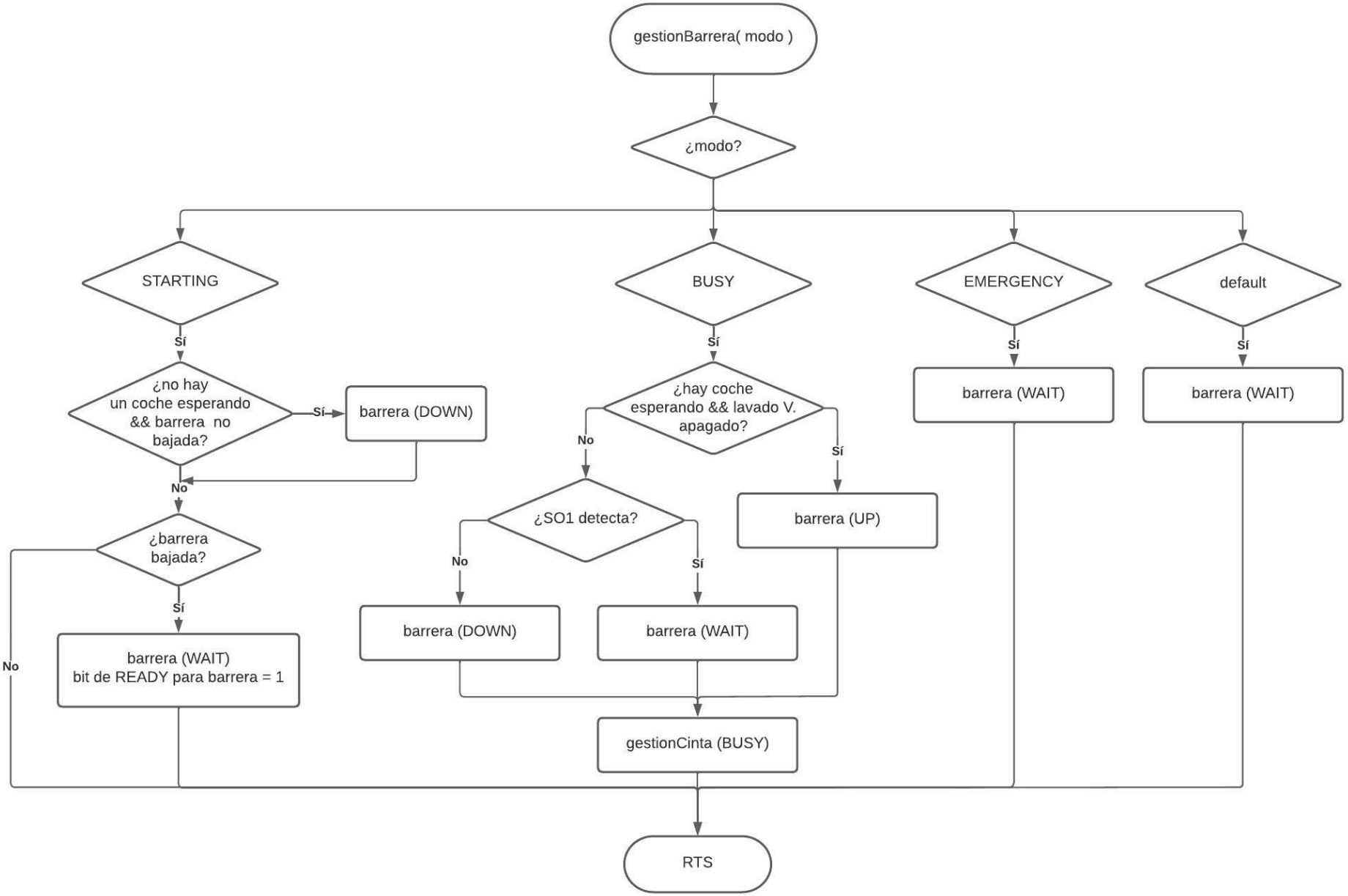

Figura 4: Función *gestionBarrera()*. Se encarga de establecer los estados de funcionamiento de la barrera.

```
void gestionBarrera(mode_t modo)
{
    switch (modo)
   {
      case STARTING:
         if (carWaiting == FALSE && SO2_f == 1)
         {
            barrera(DOWN);
         }

         if (SO2_f == 0 || barrierDown == TRUE)
         {
            barrera(WAIT);
            ready |= (1 << ENTRY_MOD);
         }
         break;

      case EMERGENCY:
         barrera(WAIT);
         break;

      case BUSY:
         if (carWaiting == TRUE && LV == FALSE)
         {
            barrera(UP);
         }
         else
         {
            SO1_f ? barrera(DOWN) : barrera(WAIT);
         }
         gestionCinta(BUSY);
         break;

      default:
         barrera(WAIT);
         break;
   }
}
```

Código 3: Implementación de la función gestionBarrera().

```
void barrera(barrier_status_t estado)
{
   switch (estado)
   {
      case UP:
         if (barrierPulseCounter < 4)
         {
            barrierMove();
         }
         else
         {
            barrierStop();
         }
         barrier = TRUE;
         break;

      case DOWN:
         if (barrierDown)
         {
            barrierStop();
            barrier = FALSE;
         }
         else
         {
            if (antiCollision == FALSE)
            {
               barrierMove();
               barrier = TRUE;
            }
            else
            {
               barrierStop();
            }
         }
         break;

      case WAIT:
         barrierStop();
         barrier = FALSE;
         break;

      default:
         break;
   }

    if(SO2_f)
   {
      barrierDown = FALSE;
   }
   else
   {
      barrierDown = TRUE;
      barrier = FALSE;
      barrierPulseCounter = 0;
   }
}
```

Código 4: Implementación de la función BARRERA().

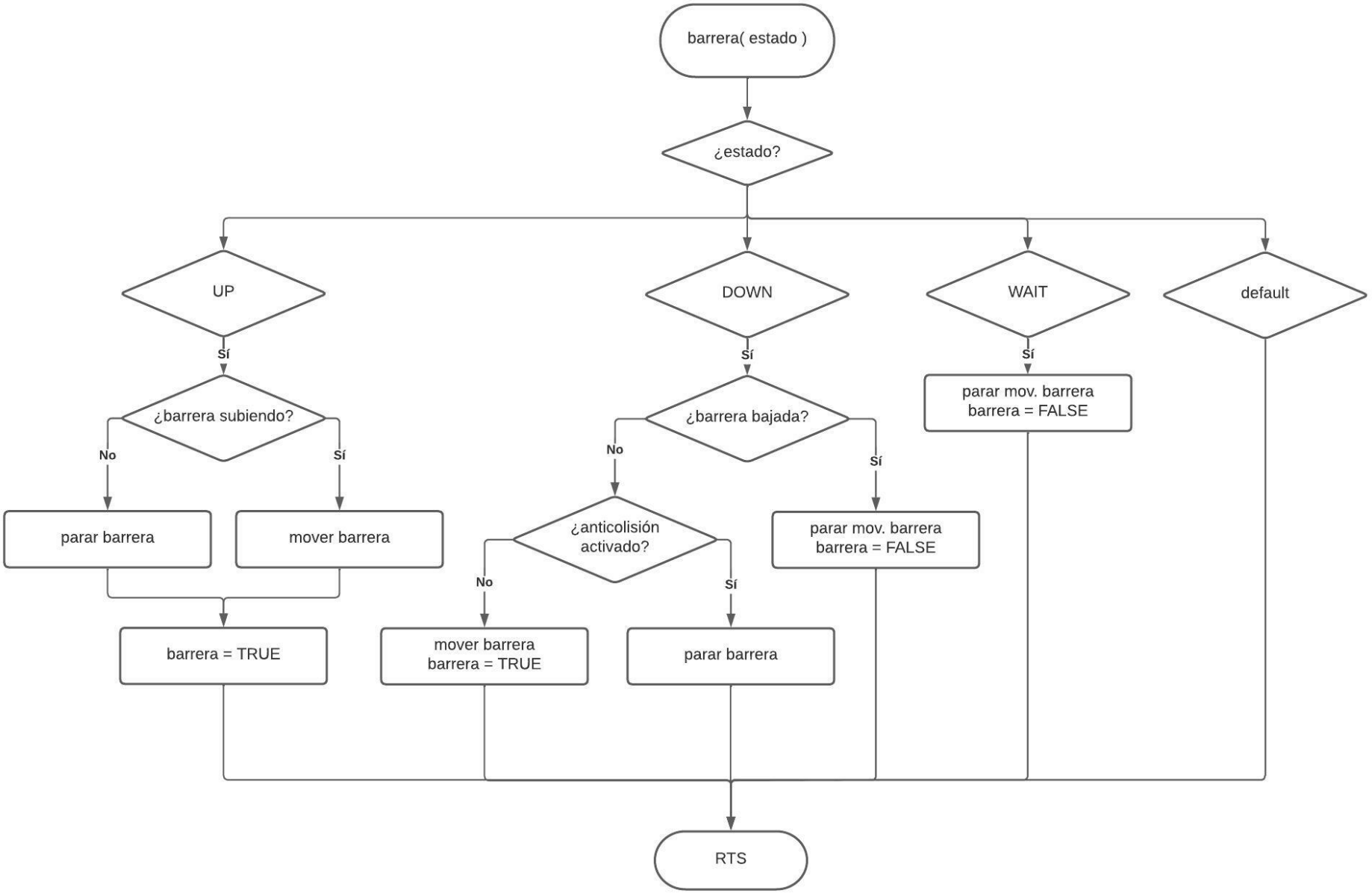

Figura 5: Función *barrera()*. Según las órdenes de *gestionBarrera()* mueve la barrera comprobando internamente las posibles restricciones.

3.2 *Lavado vertical y luz de indicación de estado*

- **Descripción:** rodillo de lavado vertical.
- **Pareja:** Mar Martín, Celia Ramos.
- **Tiempo empleado:** 10 horas.
- **Documentos fuente:** *lavado_vertical.c*, *lavado_vertical.h*, *luz.c* y *luz.h*.

El lavado vertical incluye los siguientes dispositivos: el motor M2 y la luz L1. Un resumen de los actuadores puede encontrarse en la tabla 2. La gestión de estos dos actuadores se ha realizado del siguiente modo:

- **M2:** se enciende y apaga mediante el puerto correspondiente (ver tabla 3) .
- **L1:** de la misma forma que los motores se controla mediante su puerto.

Se incluye a continuación flujograma de la interrupción de la función LAVADO_VERTICAL, de la función GESTIONLV y de PARPADEO (ver FIGS. 6, 7 y 8). Además también se incluye junto a estos los trozos de código que los implementan (ver CÓDS. 5, 6 y 7).

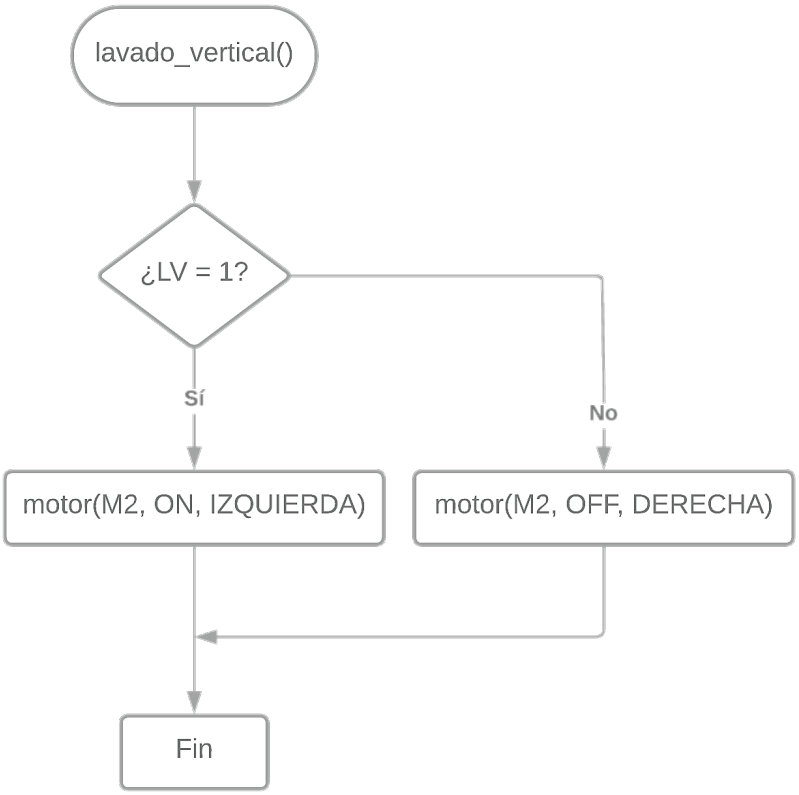

Figura 6: Diagrama de flujo de la **función** lavado vertical.

```
1  void lavado_vertical()
2  {
3      if(LV == 1 ){ //LV valdra 1 despues de unos segundos tras abrirse la barrera
4
5          motor(M2,ON,IZQUIERDA);
6
7      }
8      else if(LV == 0){ //LV se pone a 0 cuando pasa un tiempo de que la barrera se
       apaga
9
10         motor(M2,OFF,DERECHA);
11
12     }
13 }
```

Código 5: Implementación de la función LAVADO_VERTICAL().

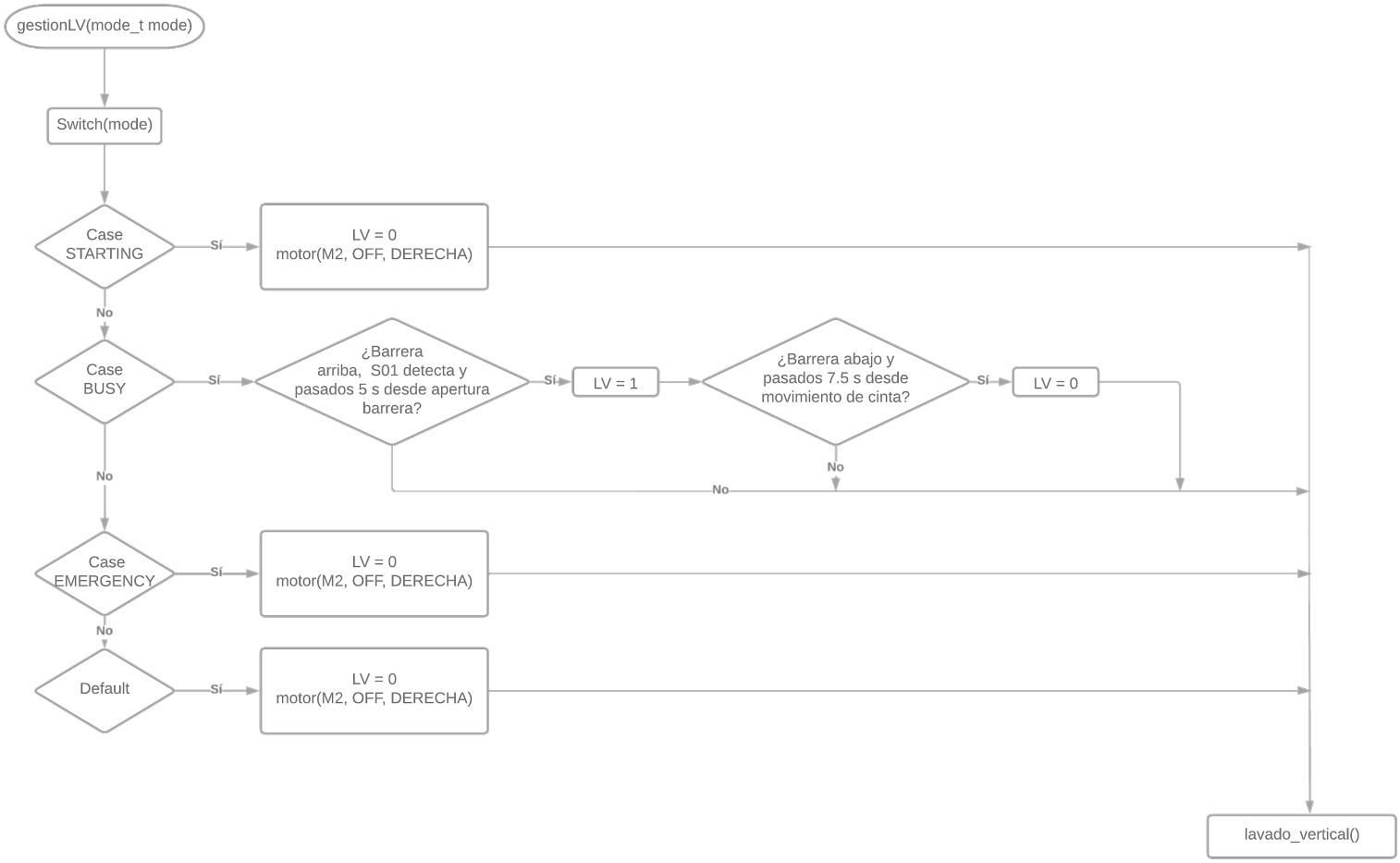

Figura 7: Diagrama de flujo de la **gestión** del lavado vertical.

```
void gestionLV(mode_t mode)
{
   switch(mode)
   {
      case STARTING:
         LV = 0;
         motor(M2,OFF,DERECHA);
         ready |= (1<<LV_MOD);
         break;

      case BUSY:
         if (barrierUp && !SO1_f && ((secondsLV + T_LV) < half_second))
         {
            LV = 1;
         }
         if (!cinta)
         {
            timeOff = timeOff-(half_second - secondsLVOff);
            secondsLVOff = half_second;
         }
         if (barrierDown && ((secondsLVOff + timeOff) < half_second))
         {
            LV = 0;
            timeOff = TLV_Off;
         }

         break;

      case EMERGENCY:
         LV = 0;
         motor(M2,OFF,DERECHA);
         break;

      default:
         LV = 0;
         motor(M2,OFF,DERECHA);
   }
   lavado_vertical();
}
```

Código 6: Implementación de la función GESTIONLV().

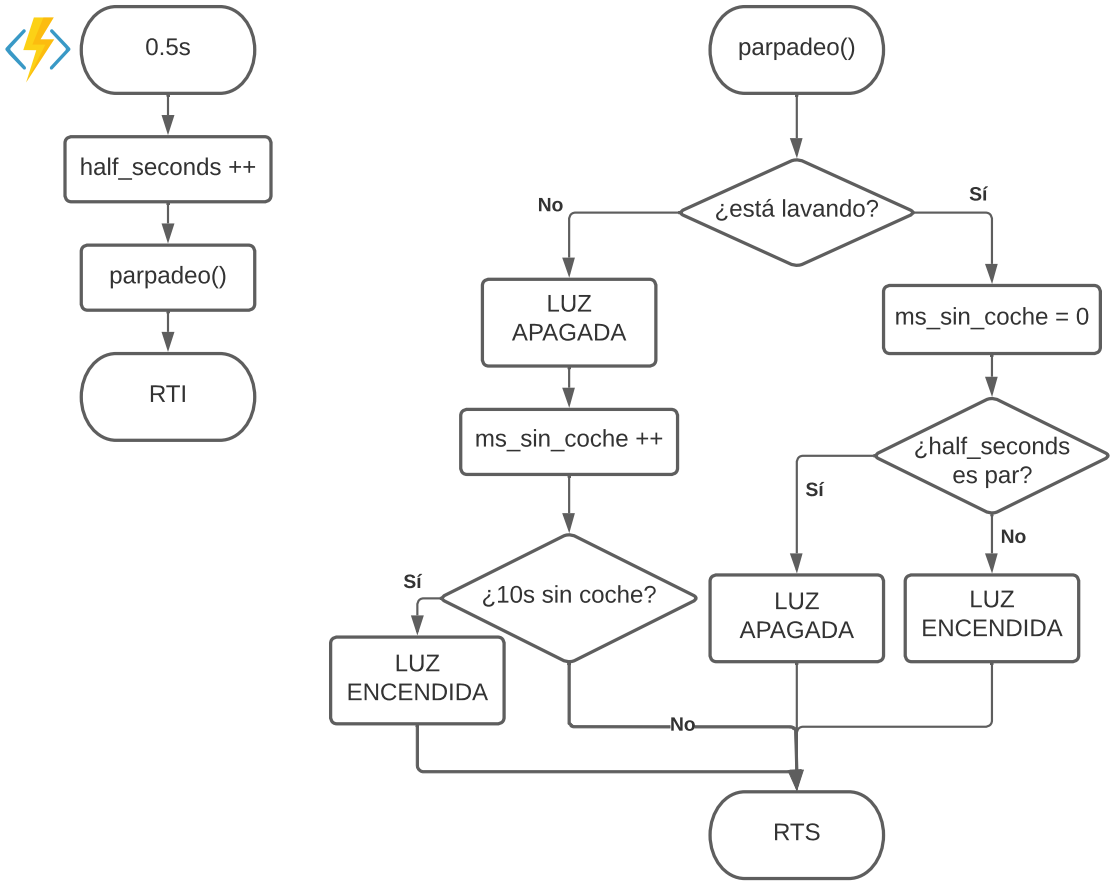

Figura 8: Diagrama de flujo de la **gestión** de la luz de estado mediante la función PARPADEO y de la interrupción periódica mediante el *Timer4* de 16 bits cada 0.5 segundos.

```
void parpadeo(seconds_t ms, bool coche)
{
    if (coche)
    {
       if(ms % 2)  // en los medios segundos "pares" esta encendida
         {
             luz(L1, ON);
         }
         else luz(L1, OFF);
    }
    else
    {
         luz(L1, OFF);
       s_sin_coche++;
         // si han pasado 20 medios de segundo (10s), se enciende
         if ((s_sin_coche % 20) == 0)
         {
             luz(L1, ON);
             s_sin_coche = 0;
         }
    }
}
```

Código 7: Implementación de la función PARPADEO().

3.3 *Lavado horizontal*

- **Descripción:** control de los rodillos de lavado horizontal.
- **Pareja:** Mario Gómez y Juan Galvañ.
- **Tiempo empleado:** 14.5 horas.
- **Archivos fuentes:** *lavado_horizontal.c* y *lavado_horizontal.h*.

El lavado horizontal es la zona de la maqueta con más sensores y actuadores. Incluye los siguientes dispositivos: el motor M3 y M4, los sensores ópticos SO3, SO4 y SO5 y el microinterruptor SW2. De nuevo se recomienda ir a la tabla 2 para mayor claridad y detalle. La gestión de estos dispositivos se ha realizado del siguiente modo:

- **M3 y M4:** se encienden y apagan mediante el puerto correspondiente (ver tabla 3), esta actuación sobre los puertos de los motores se ha generalizado mediante una función llamada MOTOR() implementado en el archivo *actuators.c*.
- **SO3, SO4 y SO5:** para estos tres sensores ópticos se ha asignado una interrupción externa tipo PCINT. En concreto la PCINT2 (ver TAB. 3). Además de estos sensores otros también están asignados a esta interrupción pero se han enmascarados para anular su efecto sobre el control del lavado horizontal.
- **SW2:** corresponde al final de carrera inferior del lavado horizontal, se emplea para saber cuándo ha llegado el lavado abajo del todo y que se inicie el ascenso a la posición de reposo.

Se incluyen los trozos de código que implementan el control del lavado (ver CÓDS. 8 y 9).

```
1  void lavado_horizontal_ISR()
2  {  // Bandera que indica si el lavado horizontal esta activo
3     if(LH == 1)
4     {  // Se comprueba si SO3 y SO5 estan detectando, si detectan, el lavado sube
5        if(SO3_f && SO5_f)
6        {  // Se comprueba si SO4 detecta algo, si no detecta, el lavado baja
7           if(SO4_f)
8           {
9              M3_state = ON;
10             M3_dir = IZQUIERDA;
11          }
12          else  // Si SO4 detecta y SO3 y SO5 tambien, entonces...
13          {     // ...el lavado esta en la posicion adecuada
14             M3_state = OFF;
15          }
16       }
17       else
18       {
19          M3_state = ON;
20          M3_dir = DERECHA;
21       }
22    }     // Si la bandera LH esta a 0 y hay interrupcion,
23    else  // se activa la bandera y se encienden los motores.
24    {
25       LH=1;
```

```
26          M4_state = ON;
27          M3_state = ON;
28          M3_dir = DERECHA;
29      }
30  }
```

Código 8: Implementación de la función LAVADO_HORIZONTAL_ISR() que se ejecuta cada vez que algún sensor (SO3, SO4 o SO5) interrumpe.

Los flujogramas de la interrupción por la PCINT2 y la posterior consulta periódica para la gestión del lavado se incluyen en las figura 9 y 10.

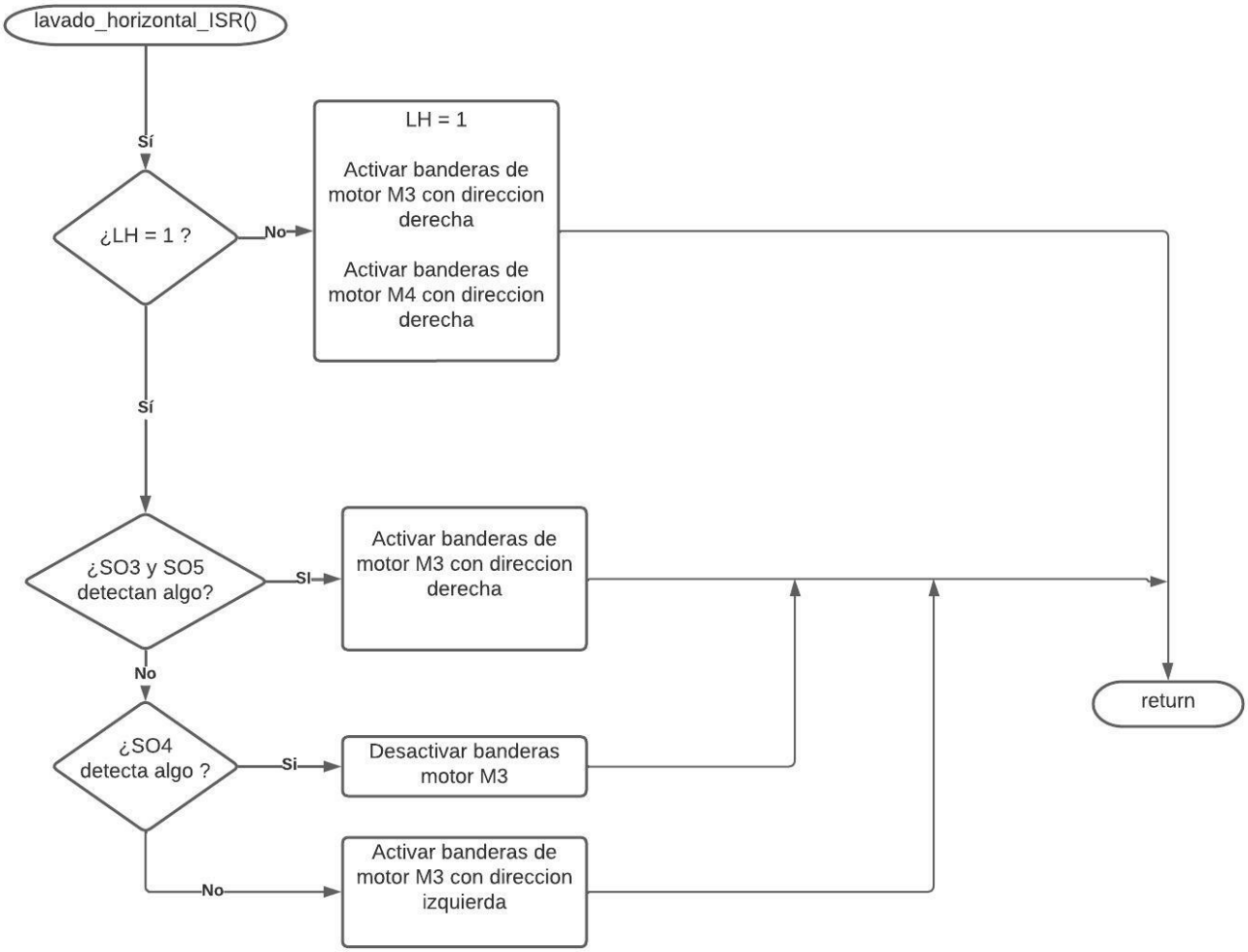

Figura 9: Diagrama de flujo de la interrupción lavado horizontal.

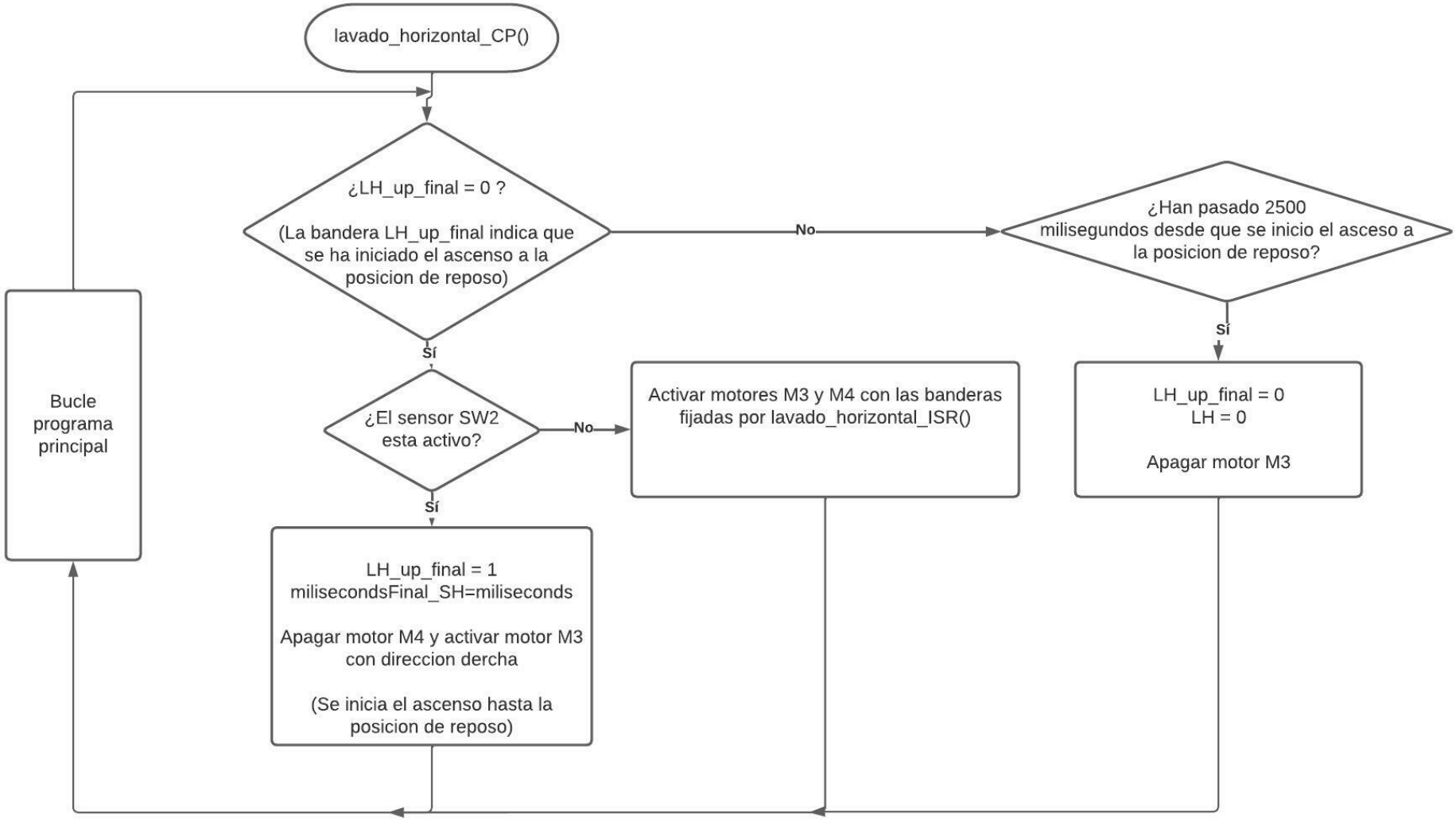

Figura 10: Diagrama de flujo de la gestión del lavado horizontal por consulta periódica.

```
void lavado_horizontal_CP()
{
   if(LH_up_final == 0) //Bandera de que se esta subiendo a la posicion final
   {  //Mientras no se detecte nada SW2 los motores funcionan con las baderas...
      if((PINK & (1 << 6)) == (1 << 6))   //...de lavado_horizontal_ISR
      {
         motor(M4,M4_state,DERECHA);
         motor(M3,M3_state,M3_dir);
      }
      else  //En cuanto detecta algo SW2 se inicia el ascenso a la posicion de reposo
      {
         M3_state = OFF;
         M4_state = OFF;
         //Se pone a 1 la bandera de que se esta subiendo a la posicion final
         LH_up_final = 1;
         milisecondsFinal_LH = miliseconds;
         motor(M4,OFF,DERECHA);
         motor(M3,ON,DERECHA);
      }
   }
   else
   {  //Cuando el lavado lleva 2500ms subiendo hacia la posicion final se paran los motores
      if(milisecondsFinal_LH + 2500 < miliseconds) //... y se baja la bandera LH_up_final
      {
         LH_up_final = 0;
         LH = 0;
         motor(M3,OFF,DERECHA);
      }
   }
}
```

Código 9: Implementación de la función LAVADO_HORIZONTAL_CP(). CP hace referencia a *Consulta Periódica.*

3.4 Secado

- **Descripción:** control del secado para seguir los perfiles de los vehículos.
- **Pareja:** Mario Gómez y Juan Galvañ.
- **Tiempo empleado:** 15 horas.
- **Archivos fuente:** *secado_horizontal.c* y *secado_horizontal.h*.

El secado horizontal es casi totalmente análogo al lavado horizontal. La única diferencia es que solamente tiene un motor, el M5. Igual que el lavado, también incluye los sensores ópticos SO7, SO8 y SO9 y el microinterruptor SW3 (que tampoco se emplea). En la tabla 2 puede verse esta asignación en mayor detalle. La gestión de estos dispositivos se ha realizado del siguiente modo:

- **M5:** se encienden y apagan mediante el puerto correspondiente (ver tabla 3).
- **SO7, SO8 y SO9:** igual que en lavado horizontal, se ha asignado una interrupción externa tipo PCINT. Ahora la PCINT0 (ver Tab. 3).
- **SW3:** corresponde al final de carrera inferior del secado horizontal, se emplea igual que el SW2 para saber cuándo ha llegado el secado a bajo del todo y que se inicie el ascenso a la posición de reposo.

```
1  void secado_horizontal_ISR()
2  {  // Bandera que indica si el secado horizontal esta activo
3     if(SH==1)
4     {  // Se comprueba si SO7 y SO9 estan detectando algo,
5        if(SO7_f && SO9_f)   // si detectan algo el lavado tiene que subir
6        {  // Se comprueba si SO8 detecta algo, si no detecta el lavado
7           if(SO8_f)   // esta muy alto y tiene que bajar
8           {
9              M5_state = ON;
10             M5_dir = IZQUIERDA;
11          }
12          else  // Si SO8 detecta y SO7 y SO9 tambien el
13          {     // lavado esta en la posicion adecuada
14             M5_state = OFF;
15          }
16       }
17       else
18       {
19          M5_state = ON;
20          M5_dir = DERECHA;
21       }
22    }
23    else     // Si el la bandera SH esta a 0 y hay interrupcion
24    {     // se activa la bandera y se encienden los motores
25       SH = 1;
26       M5_state = ON;
27       M5_dir = DERECHA;
28    }
29 }
```

Código 10: Implementación de la función SECADO_HORIZONTAL_ISR() que se ejecuta cada vez que algún sensor (SO7, SO8 o SO9) interrumpe.

Los flujogramas son análogos a los del lavado horizontal y se incluyen en las figura 11 y 12. Además también se incluye junto a estos los trozos de código que los implementan (ver Cód. 10 y 11).

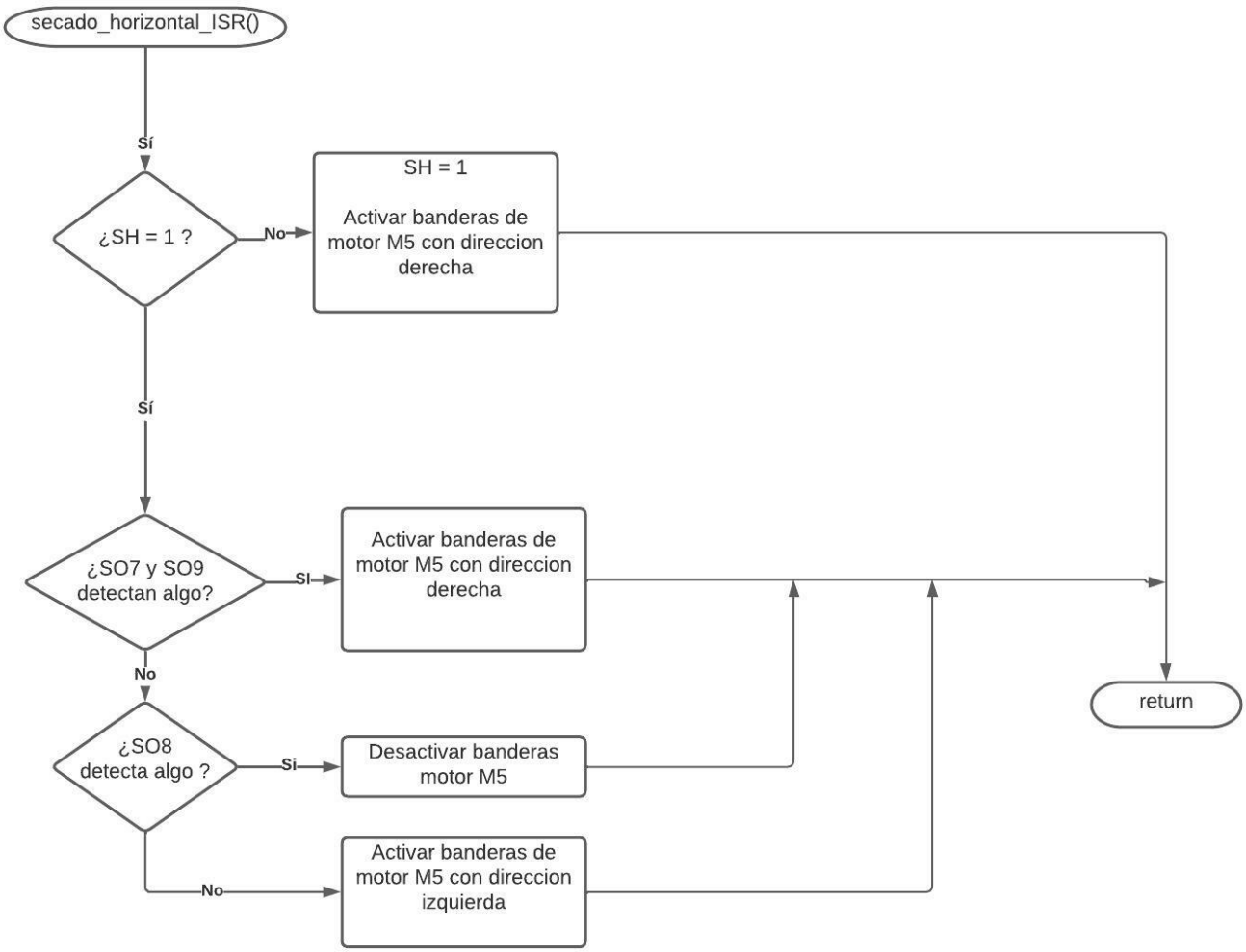

Figura 11: Diagrama de flujo de la interrupción del secado horizontal.

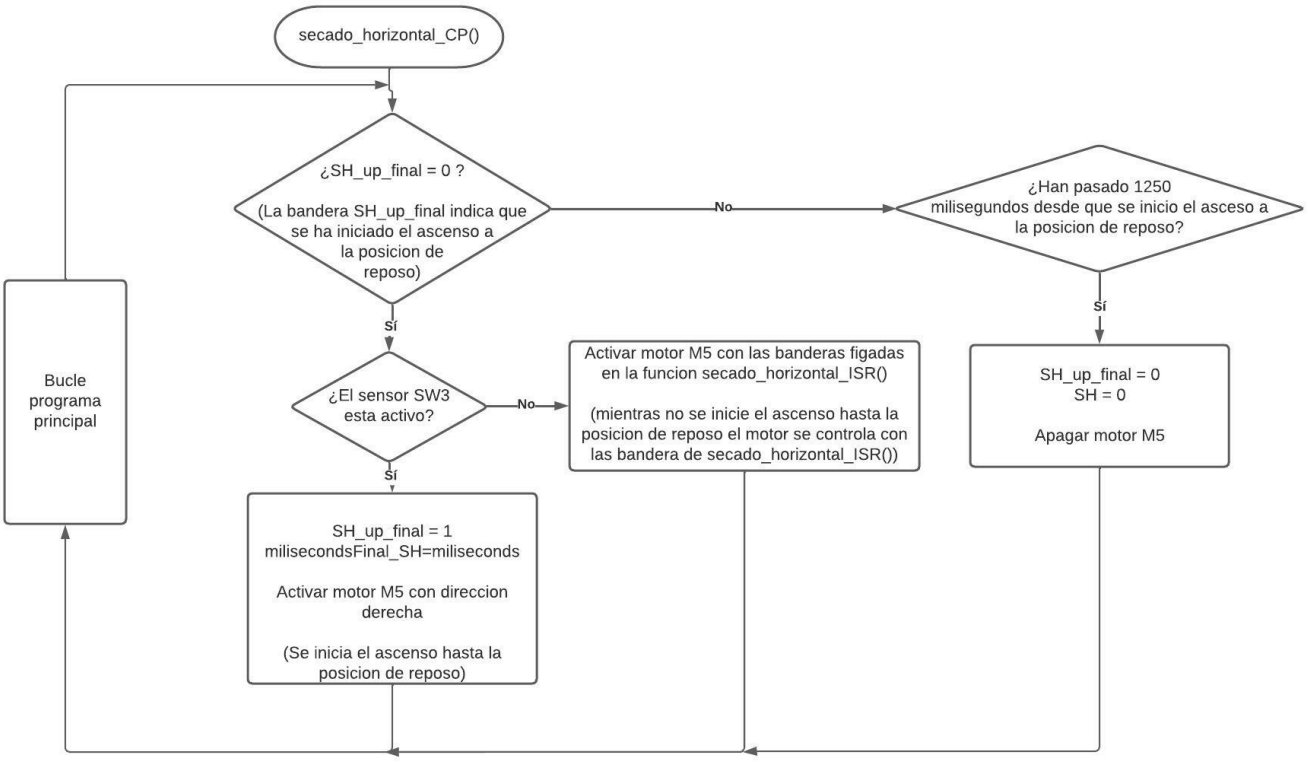

Figura 12: Diagrama de flujo de la gestión del secado horizontal por consulta periódica.

```
void secado_horizontal_CP()
{  // Bandera de que se esta subiendo a la posicion final
   if(SH_up_final == 0)
   {  // Mientras no se detecte nada en SW3 los motores funcionan con las
      if((PINK & (1 << 7)) == (1 << 7))   // banderas de secado_horizontal_ISR
      {
         motor(M5,M5_state,M5_dir);
      }
      else // En cuanto detecta SW3 se inicia el ascenso a la posicion de reposo
      {
         M5_state = OFF;
         // Se pone a 1 la bandera de que se esta subiendo a la posicion final
         SH_up_final = 1;
         milisecondsFinal_SH = miliseconds;
         motor(M5,ON,DERECHA);
      }
   }
   else  // Cuando el secado lleva 1250ms subiendo hacia la posicion final
   {     // se paran los motores y se baja la bandera SH_up_final
      if(milisecondsFinal_SH + 1250 < miliseconds)
      {
         M5_state = OFF;
         SH_up_final = 0;
         SH = 0;
         motor(M5,OFF,DERECHA);
      }
   }
}
```

Código 11: Implementación de la función SECADO_HORIZONTAL_CP().

3.5 Salida, cinta de arrastre y parada de emergencia

- **Descripción:** control de la cinta de arrastre, semáforo y parada de emergencia.
- **Pareja:** Gonzalo Quirós y Josep Mª Barberá.
- **Tiempo empleado:** 8 horas.
- **Archivos fuente:** *salida.c*, *salida.h*, *cinta.c*, *cinta.h* y *emergencia.h*.

La salida comprende los sensores SO10, SO11 y SO12, además de las luces L4 y L5. La cinta de arrastre sólo funciona en una dirección mediante el motor M6 y la parada de emergencia sólo debe atender la pulsación del microinterruptor SW4. De esta forma la gestión de dichos dispositivos se ha realizado del siguiente modo:

- **Salida**
  - **SO10, SO11 y SO12:** para el SO10 se ha asignado la interrupción externa INT2. Para el resto, tienen asignada la PCINT2 pero están enmascarados para que no afecten, pues se comprobaran por consulta periódica.
  - **L4 y L5:** al igual que los motores se trata de cambiar un bit de su correspondiente puerto de salida. Se ha empleado una función genérica llamada LUZ que gestiona la asignación de pines, y otra con un nivel de abstracción todavía mayor para poner a verde o a rojo el semáforo (luces L4 y L5 respectivamente).
- **Cinta**
  - **M6:** se enciende y apaga mediante el puerto correspondiente (ver tabla 3). La lógica está basado en el estado de una variable booleana global llamada *cinta* que recoge el estado de todos módulos del túnel de lavado y se enciende en función de que al menos uno de ellos esté funcionando.
- **Parada de emergencia**
  - **SW4:** está configurado con la interrupción externa INT3. De esta forma cuando el botón de emergencia es pulsado se entra en una interrupción bloqueante que pone en estado de emergencia a todos los sistemas del túnel de lavado (ver CÓD. 16).

Los flujogramas se incluyen en las figura 13, 14 y 15. Además también se incluye junto a estos los trozos de código que los implementan (ver CÓDS. 12, 13, 14 y 15).

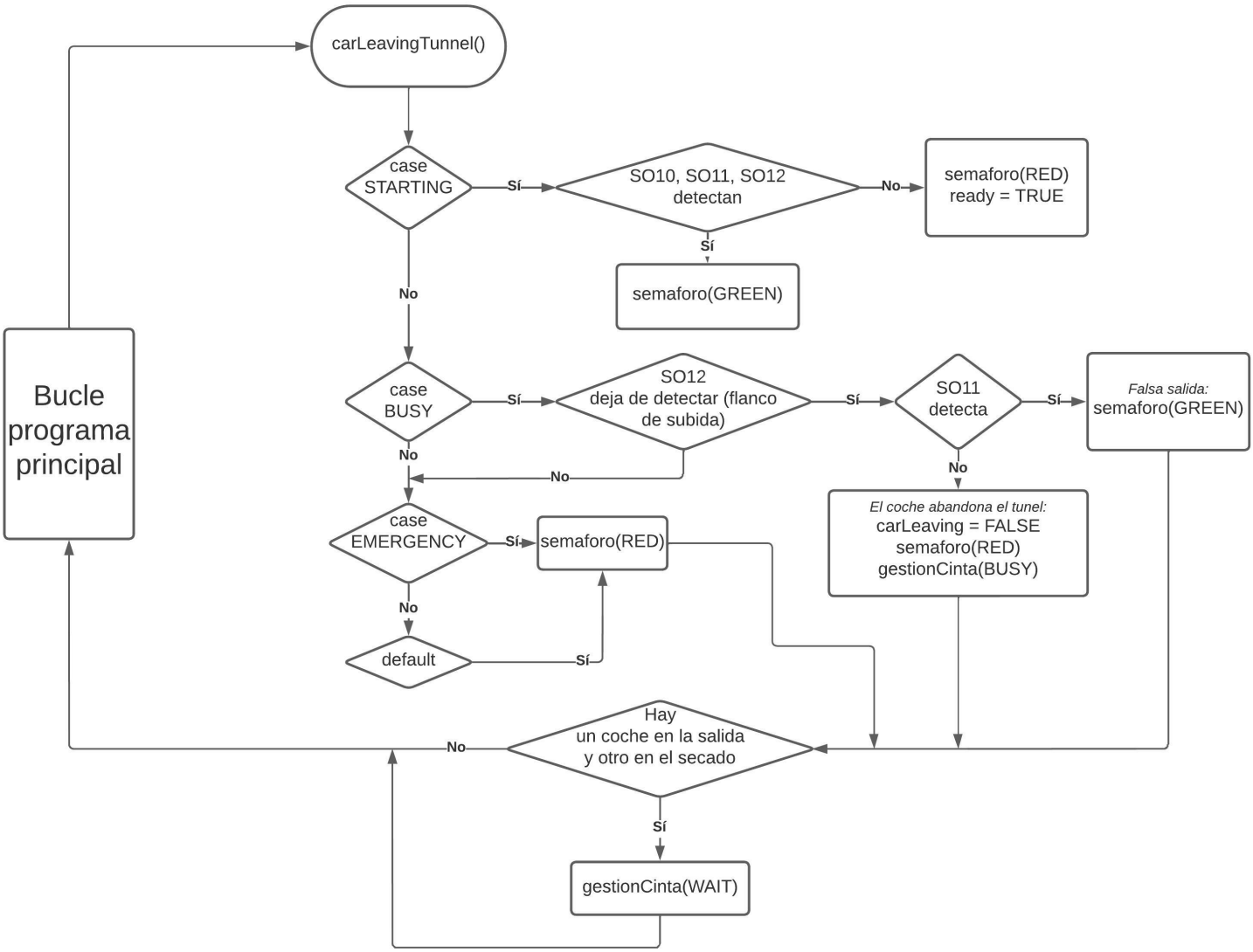

Figura 13: Diagrama de flujo de la gestión de la salida del túnel de lavado.

```
void carLeavingTunnel (mode_t mode)
{
    switch (mode)
    {
        case STARTING:
            if (SO12_f && SO11_f && SO10_f)
            {
                semaforo(RED);
                ready |= (1 << OUT_MOD);
            }else
            {
                semaforo(GREEN);
                ready &= ~(1 << OUT_MOD);
            }
            break;
        case WAITING:
        case BUSY:
            if (SO12_f)  // No hay nada cortando el sensor. No esta detectando
            {
                if (prevState)
                {   // No detecta nada
                    prevState = TRUE;
                }else if (!prevState)   // Esto es un flanco de subida.
                {                       // Antes detectaba y ahora no.
                    if (!SO11_f)    // Se produce un flanco de subida,
                    {   // pero sigue habiendo coche dentro, estan dando
                        prevState = TRUE;   // marcha atras en la salida
                    }else if (SO11_f)   // el coche esta saliendo de verdad
                    {
                        carLeaving = FALSE;
                        semaforo(RED);
                        gestionCinta(BUSY);
                        prevState = TRUE;
                    }
                }
            }else if (!SO12_f)
            {
                prevState = FALSE;
            }
         if (!SO11_f && !SO10_f )
         {
            semaforo(GREEN);
         }
            if (!SO10_f && !SO11_f && !SO12_f)
            {   // Si detecto
                carLeaving = TRUE;
            }   // Si hay un coche en el secado y otro en la salida...
            if (carLeaving && SH) // paramos la cinta para evitar choques
            {
                gestionCinta(WAITING);
            }
            break;
        case EMERGENCY:
            semaforo(RED);
            break;
    }
}
```

Código 12: Implementación de la función carLeavingTunnel().

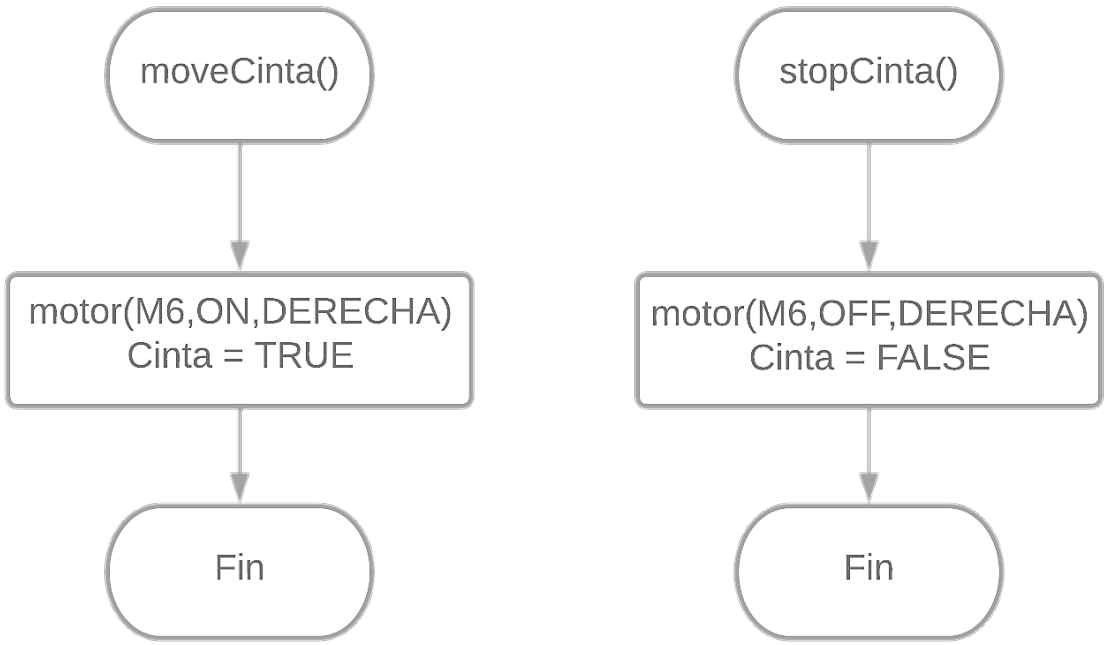

Figura 14: Diagrama de flujo de la función para el encendido y apagado de la cinta.

```
1  void moveCinta()
2  {
3      motor(M6,ON,DERECHA);
4      cinta = TRUE;
5  }
6
7  void stopCinta()
8  {
9      motor(M6,OFF,DERECHA);
10     cinta = FALSE;
11 }
```

Código 13: Implementación de las funciones moveCinta() y stopCinta().

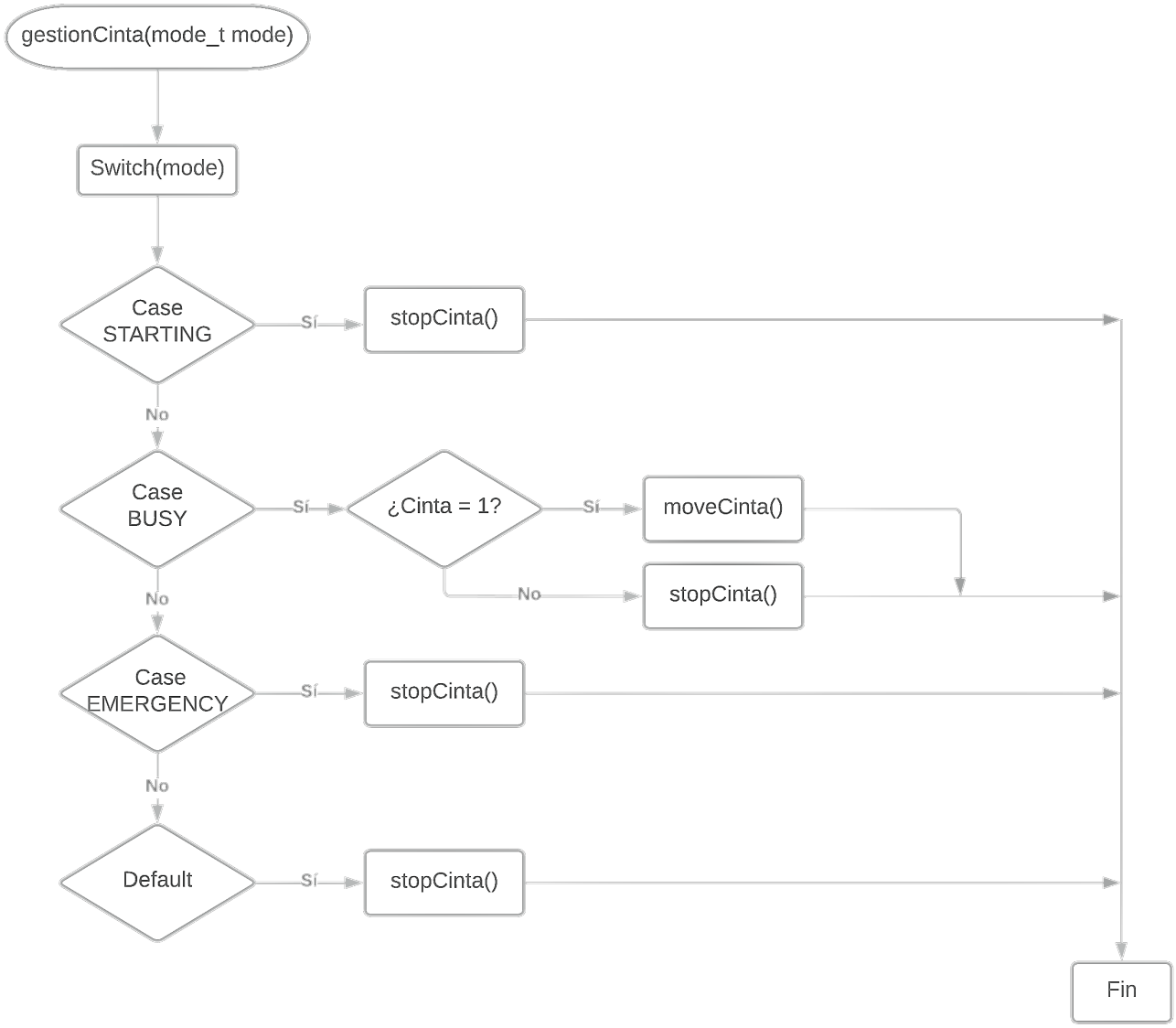

Figura 15: Diagrama de flujo de la gestión de la cinta.

```
void gestionCinta (mode_t modo)
{
    switch (modo)
    {
        case STARTING:
            stopCinta();
            ready |= (1<<BELT_MOD);
            break;

        case WAITING:
         stopCinta();
         break;
        case BUSY:
            cinta ? moveCinta() : stopCinta();
            break;

        case EMERGENCY:
            stopCinta();
            break;

        default:
            stopCinta();
            break;
    }

}
```

Código 14: Implementación de la función GESTIONCINTA().

```
ISR(INT3_vect)
{   // Espera bloqueante, sera necesario resetear el micro.
    // Se entiede que el operario encargado del tunel de lavado
    // ha tenido que parar todo por un problema grave.
    while (1)
    {
        gestionBarrera(EMERGENCY);
        gestionCinta(EMERGENCY);
      gestionLV(EMERGENCY);
      gestionLH(EMERGENCY);
      gestionSH(EMERGENCY);
      carLeavingTunnel(EMERGENCY);
    }

}
```

Código 15: Implementación de la interrupción EMERGENCIA().

3.6 Integración del sistema

- **Descripción:** gestión de cada sección e integración entre módulos.
- **Pareja:** Gonzalo Quirós y Josep Mª Barberá.
- **Tiempo empleado:** 10 horas.
- **Archivos fuentes:** *timers.c*, *setup.c*, *timers.h*, *commonstuff.h* y *main.c*.

Para la integración del sistema se ha empleado una variable global tipo char llamada *ready*, de forma que en sus 8 bits almacenaba el estado de cada una de las partes del tunel de lavado. Dicha variable se inicializaba a cero y en la primera parte del bucle principal cada una de las partes de la maqueta comenzaban en modo STARTING, cuyo fin tenía acabar cambiando el bit correspondiente de la variable *ready* de 0 a 1.

Puede verse en el código 16 el bit asignado para cada zona de la maqueta. Estas definiciones están incluidas al final del archivo *commonstuff.c*, el cual ha sido clave para programar de forma más abstracta y funcional. De los ocho bits solo se emplean 7 y de esos, el correspondiente al modo de funcionamiento de la luz (LIGHT_MODE) no ha sido necesario finalmente. Por en el código 17 puede verse la variable *ready* comparada al número binario 0*b*01101111.

```
/* Posicion de cada "modo" en el byte de Modos */
#define ENTRY_MOD    0
#define LV_MOD       1
#define LH_MOD       2
#define DRYER_MOD    3
#define LIGHT_MOD    4 // finalmente no se emplea
#define BELT_MOD     5
#define OUT_MOD      6
```

Código 16: Definición de los pines de *ready* para cada zona de la maqueta.

```
if (ready != 0b01101111)
     {
        gestionBarrera(STARTING);
        gestionLV(STARTING);
        gestionLH(STARTING);
        gestionSH(STARTING);
        gestionCinta(STARTING);
        carLeavingTunnel(STARTING);
     }
     else if (ready == 0b01101111)
     {
        gestionBarrera(BUSY);
        gestionLV(BUSY);
        gestionLH(BUSY);
        gestionSH(BUSY);
        carLeavingTunnel(BUSY);
     }
```

Código 17: Bucle principal del main.

Ha sido muy importante el uso de *timers* para llevar la cuenta del tiempo. Para esto hemos programado dos interrupciones temporales con *timers de 16 bits*, en concreto el Timer3 y el Timer4. Puede verse en el código 18 el *setup* de los *timers*.

```
void timerSetup()
{
    // setup del contador 3
    cli ();
    TCCR3B |= (1<<WGM32);              // Modo ctc
    TCCR3B |= (1<<CS31)|(1<<CS30);     // Preescalado clk/64 (periodo de 8 uS)
    OCR3A = ANTIREB_TIME;              // Contamos COUNTER_TIME periodos (= 1 ms)
    TIMSK3 |= (1<<OCIE3A);             // Habilitamos la interrupcion por compare
   match
    // setup del contador 4
    TCCR4B |= (1<<WGM42);              // Modo ctc
    TCCR4B |= (1<<CS42);               // Preescalado clk/256 (periodo de 32 uS)
    OCR4A = REAL_TIME;                 // Contamos REAL_TIME periodos (=0.5 s)
    TIMSK4 |= (1<<OCIE4A);             // Habilitamos la interrupcion por compare
   match
    sei();
}
```

Código 18: Setup de los *timers*.

IV

Comentario y propuestas de mejora

Sobre la relación entre el módulo de entrada y el lavado vertical:

- El encendido y apagado del lavado vertical se efectúa mediante una gestión temporal, debido a que el lavado vertical (LV) carece de sensores. Para efectuar un control robusto de la activación y desactivación del LV, es necesario supeditarlos a la detección de la presencia de los vehículos por parte de sensores en otros módulos. De esta forma podemos evitar encendidos innecesarios (falsas entradas), y apagar antes el LV ante rectificaciones de los clientes (salen marcha atrás antes de ser enganchados por la cinta).
- El tamaño de los vehículos es variable, y por lo tanto el tiempo que permanece encendido el LV también debe serlo. Para tener en cuenta esta variabilidad, no se ha prefijado un tiempo de funcionamiento, sino que se han fijado los tiempos de encendido y apagado de los rodillos del LV, que serán invariables porque la velocidad de arrastre de la cinta es constante. El tiempo de encendido se ha establecido empíricamente como el tiempo en segundos que tarda un vehículo en llegar al LV desde que se le permite la entrada al túnel de lavado. Si transcurrido ese tiempo, que llamaremos T_LV, el sensor de entrada (SO1) sigue detectando, entonces la entrada resulta ser una entrada normal de vehículo, en caso contrario sería una rectificación y se procedería a un apagado anticipado de los rodillos para ahorrar energía. Por otro lado el tiempo de apagado, que llamaremos TLV_Off, es el tiempo transcurrido desde que la parte trasera del vehículo abandona el sensor de entrada (SO1) hasta que abandona los rodillos del LV.

Limitaciones y líneas futuras de la relación entre la entrada y el lavado vertical:

- La principal limitación del diseño de éstos módulos tiene que ver con el tamaño de los vehículos. Como se desprende de la explicación anterior, si un vehículo no es los suficientemente largo como para seguir cortando el sensor de entrada (SO1) después de transcurridos T_LV segundos, su entrada real será tomada como rectificación (salida marcha atrás) y los rodillos se apagarán prematuramente. Por esta razón, el diseño está limitado a vehículos que guarden la misma escala que con los que se ha probado y para los cuales se han diseñado todos los módulos.
- En cuanto a las líneas futuras, la principal característica que no se ha podido implementar a tiempo debido a los problemas experimentados en el lavado horizontal (ruido en sensores debido al PWM), es la consulta del estado del LH para mejorar el apagado del LV ante rectificaciones de los clientes. Actualmente, la forma de comprobar si un cliente ha rectificado es consultar el estado del sensor de entrada (SO1) en el momento de encendido de los rodillos del LV. Si no detecta vehículo, se toma como rectificación. Veamos ahora el principal error que puede ocurrir:
    - Si el vehículo rectifica después del encendido del LV: para detectar esta situación, sería necesario consultar el estado del LH, si pasado cierto tiempo desde la entrada en el túnel, el LH sigue sin activarse (se activa por sensores, no por tiempo) entonces podemos tomarlo como una rectificación tardía del cliente. En este caso, el funcionamiento adecuado sería levantar la barrera para permitir la salida hacia atrás del cliente arrepentido y apagar de inmediato el LV.

Sobre la relación entre el módulo de salida y el secado:

- La salida de la cinta es un momento crítico, pues el vehículo pasa de estar controlado por la cinta a estar controlado por el cliente, luego hay que hacer que el sistema sea lo suficientemente robusto para evitar accidentes provocados por conductas inadecuadas de los clientes en esta zona. El principal problema que puede surgir es que un vehículo se quede bloqueando la salida (por avería del vehículo, por que se cale o porque el cliente esté a otra cosa) mientras otro vehículo avanza por la cinta, lo que puede provocar un choque entre ambos vehículos.
- Una forma de evitarlo sería que el operario activara la parada de emergencia antes de que el segundo alcance al primero, pero esto arruinaría el lavado completo del segundo y habría que volver a introducirlo en el túnel una vez reseteado. Esto es inviable por el gasto que conllevaría. La solución por la que se ha optado es habilitar un modo automático de pausa subordinado al movimiento de la cinta. Si hay un vehículo aún en la salida y el siguiente llega al secado, la cinta se para, imponiendo un estado de pausa en LV (si estaba en funcionamiento, sigue en funcionamiento y si estaba parado sigue parado). Este estado no es necesario en la barrera, en LH, ni en el secado, pues como ya se ha dicho, funcionan por sensónica, de modo que reaccionan a los estímulos recibidos directamente por sus sensores (si detectan funcionan y si no, no).
- Cuando el vehículo que bloqueaba la salida al fin la abandona, la cinta vuelve a ponerse en funcionamiento y el conteo de tiempo para el apagado o encendido del LV se reanuda si fuera necesario.

v

Tabla de conexiones

Para la asignación de los pines, hemos partido de las diagramas de

Parejas		Motores	S. Ópticos	S. Mecánicos	Luces
P2	**Entrada**	M1	SO1, SO2	SW1	—
P2	**L. Vertical**	M2	—	—	—
P1	**L. Horizontal**	M3, M4	SO3, SO4, SO5	SW2	—
P1	**Secado**	M5	SO7, SO8, SO9	SW3	—
P3	**Salida**	—	SO10, SO11, SO12	—	L4, L5
P3	**Cinta**	M6	—	—	—
P2	**Luz**	—	—	—	L1
P3	**B. Emergencia**	—	—	SW4	—
Pines Maqueta		M1(9, 11) M2(13, 15) M3(2, 4) M4(6, 8) M5(10, 12) M6(14, 16)	SO1(19), SO2(21) SO3(23), SO4(25) SO5(27), SO6(29) SO7(31), SO8(33) SO9(28), SO10(30) SO11(32), SO12(34)	SW1(1) SW2(3) SW3(5) SW4(7)	L1(20) L4(22) L5(24)
Pines Micro		M1(PK0, PK2) M2(PK4, PK6) M3(PB1, PB3) M4(PB5, PB7) M5(PK1, PK3) M6(PK5, PK7)	SO1(PL0), SO2(PL2) SO3(PL4), SO4(PL6) SO5(PD0), SO6(PD2) SO7(PD4), SO8(PD6) SO9(PD1), SO10(PD3) SO11(PD5), SO12(PD7)	SW1(PB0) SW2(PB2) SW3(PB4) SW4(PB6)	L1(PL1) L4(PL3) L5(PL5)
Pines reales		M1(PL0, PL1) M2(PL2, PL3) M3(PL4, PL5) M4(PL6, PL7) M5(PD4, PD5) M6(PD6, PD7)	SO1(PD1), SO2(PK1) SO3(PK2), SO4(PK3) SO5(PK4), SO6(PK5) SO7(PB0), SO8(PB1) SO9(PB2), SO10(PD2) SO11(PB4), SO12(PB5)	SW1(PD0) SW2(PK6) SW3(PK7) SW4(PD3)	L1(PB3) L4(PB6) L5(PB7)

Tabla 2: Distribución de pines

Motores	**Dispositivo**	**Puerto**	**Interrupciones**
Barrera entrada	M1_en	PL0	—
	M1_dir	PL1	—
Lavado vertical	M2_en	PL2	—
	M2_dir	PL3	—
Lavado horizontal	M3_en	PD4	—
	M3_dir	PD5	—
Giro lavado horizontal	M4_en	PL6	—
	M4_dir	PL7	—
Secado	M5_en	PL4	—
	M5_dir	PL5	—
Cinta arrastre	M6_en	PD6	—
	M6_dir	PD7	—
Sensores			
Entrada de vehículos	SO1	PD1	INT1
Cierre de barrera	SO2	PK1	PCINT17 (no usar)
Lavado horizontal inicio	SO3	PK2	PCINT18
Lavado horizontal medio	SO4	PK3	PCINT19
Lavado horizontal final	SO5	PK4	PCINT20
Paso intermedio	SO6	PK5	PCINT21 (no usar)
Secado inicio	SO7	PB0	PCINT0
Secado medio	SO8	PB1	PCINT1
Secado final	SO9	PB2	PCINT2
Abandono de cinta	SO10	PD2	INT2
Paso final	SO11	PB4	PCINT4 (no usar)
Abandono de estación	SO12	PB5	PCINT5 (no usar)
Interruptores			
Contador pulsos barrera	SW1	PD0	INT0
Fin de carrera L. horizontal	SW2	PK6	PCINT22 (no usar)
Fin de carrera Secado	SW3	PK7	PCINT23 (no usar)
Parada de emergencia	SW4	PD3	INT3
Luces			
Luz funcionamiento	L1	PB3	—
Luz salida (luz verde)	L4	PB6	—
Luz salida (luz roja)	L5	PB7	—

Tabla 3: Distribución de los actuadores según su puerto y cuando corresponde su interrupción asociada.

VI

Anexos

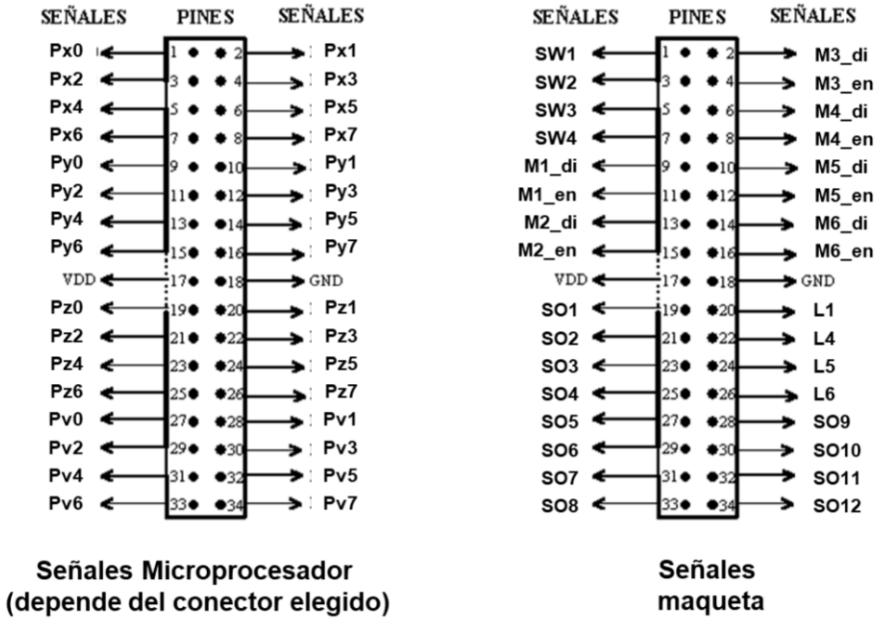

Figura 16: Conexión entre el microprocesador y la maqueta. Esquema de pines.

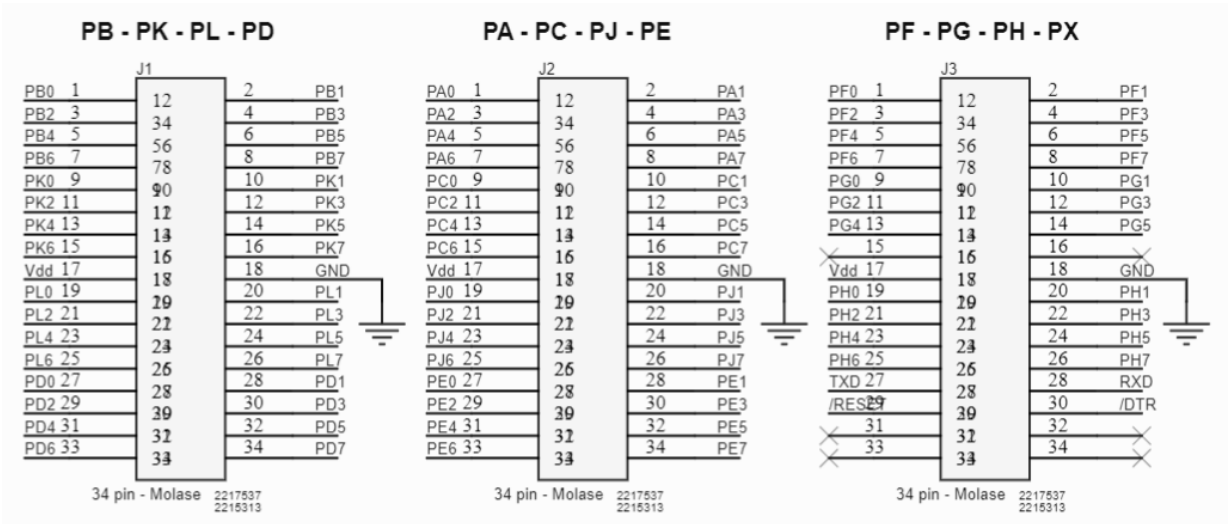

Figura 17: Los tres posibles conectores con el microprocesador. Determinan los pines del micro a emplear y por tanto los recursos que estos ofrecen (PWM, INT, PCINT, etc.). Nuestro caso ha supuesto la elección del conector J1.

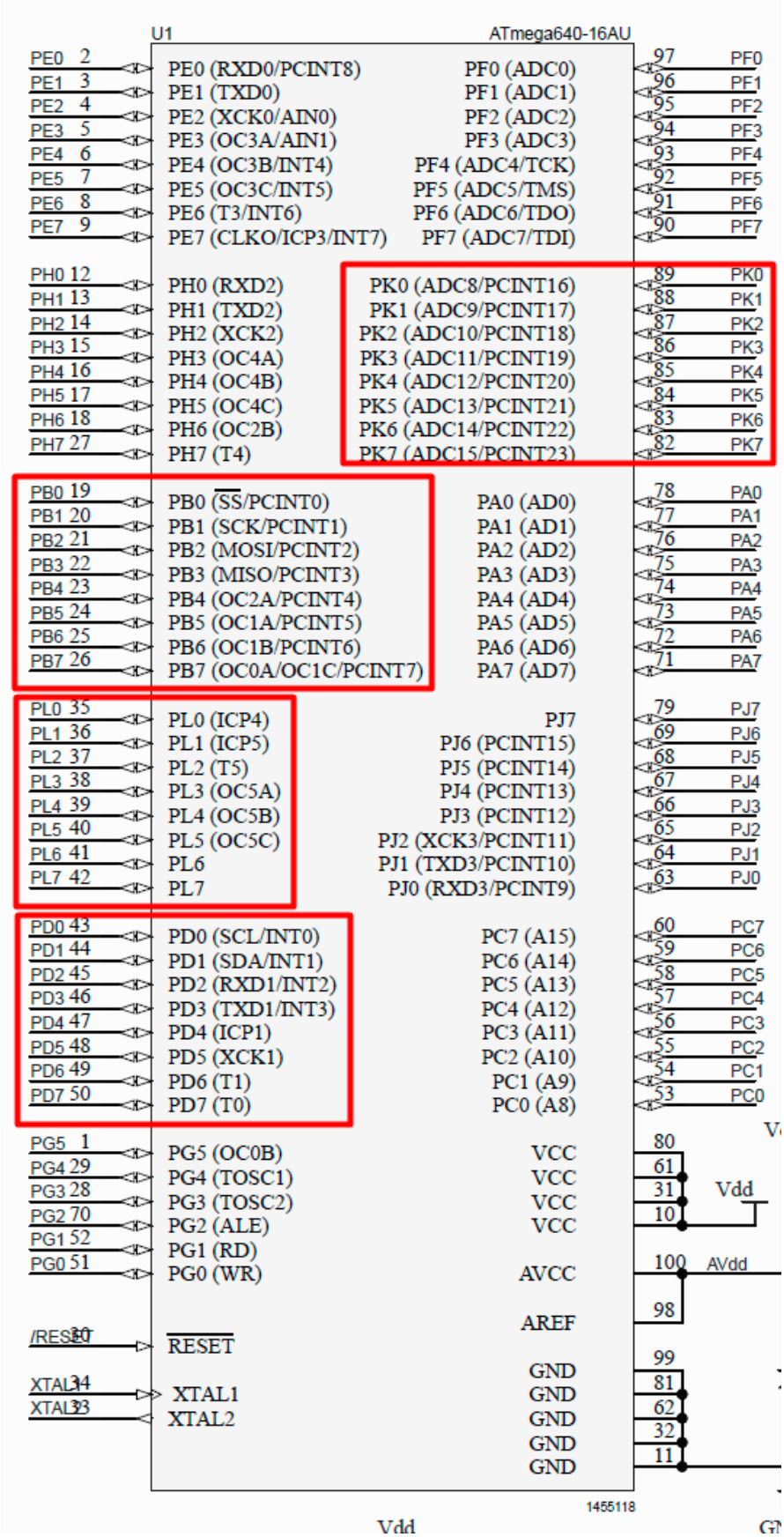

Figura 18: Esquema del ATmega640. Se ha señalado en rojo los puertos que empleamos.