Informe de la implementación en un sistema empotrado distribuido del juego Pong

Laboratorio de Sistemas Basados en Microcomputador

Miró Barceló, Oriol
2014-01-07

Índice

1. Introducción

El objetivo consiste en implementar el videojuego Pong (tenis de mesa) en un sistema empotrado y distribuido. Este sistema consistirá de tres microcontroladores dsPic30F4011 y dos ordenadores.

La estructura de este sistema consiste en un maestro, implementado en uno de los microcontroladores, y dos esclavos, implementados en los otros dos microcontroladores, conectados a los ordenadores. Los tres microcontroladores se comunicarán mediante una red CAN y los esclavos se comunicarán con los ordenadores mediante el uso del puerto UART.

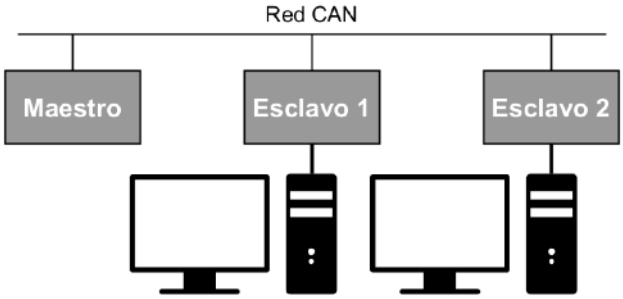

Fig. 2 Arquitectura del sistema

Fig. 1: Arquitectura del sistema

El maestro se encargará de controlar la bola. Monitorizará su estado en todo momento, indicando a los esclavos si ha rebotado contra una pared o contra una paleta o si se ha marcado un punto, y actualizará la posición de la bola, provocando su movimiento acorde a las acciones de los jugadores. También variará la velocidad de la bola acorde con el valor del potenciómetro del microcontrolador.

Los esclavos, que representan un jugador cada uno, responderán a las acciones tomadas por los jugadores en los ordenadores, moviendo las paletas acorde a los comandos introducidos por los jugadores e informando al maestro y al otro esclavo de este movimiento. También será su función pintar la pantalla del ordenador para mostrar el juego.

2. Diseño:

2.1 Pantalla

Empezamos diseñando el aspecto de la pantalla de juego, para así tener claros que elementos intervienen y los tamaños de los elementos visuales para definir correctamente las constantes que los determinarán.

El tamaño de la pantalla de juego será de 40x30 carácteres. En la figura 2 observamos la pantalla de juego mostrada al usuario, donde un cuadrado es el tamaño de la bola, un cuadrado en la figura representa un carácter en la pantalla del ordenador y cada cuadrado negro será mostrado en el ordenador como el carácter #.

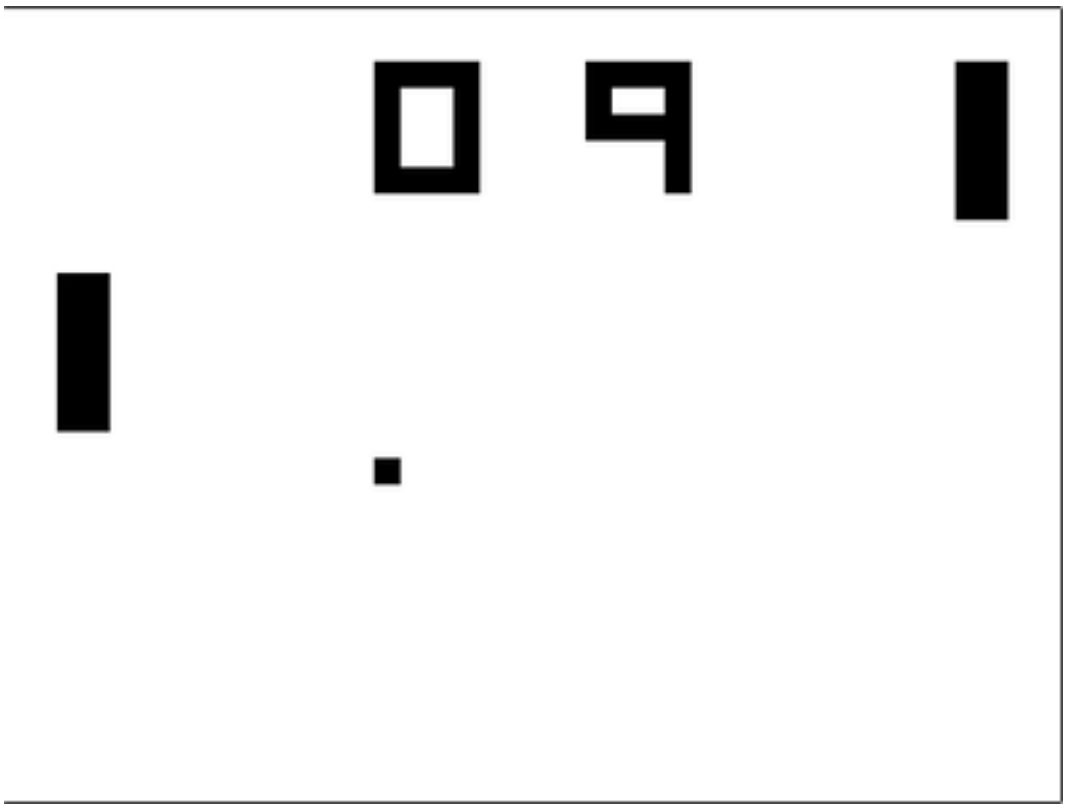

Fig. 2: Visualización del juego

En la siguiente figura observamos como se representarían los número del marcador:

Fig. 3: Visualización del marcador

El diseño de estos dos elementos es interesante ya que nos servirá a la hora de implementar los tamaños de las paletas, la bola y la pantalla, y para tener clara la programación necesaria para representar en la pantalla del ordenador la pantalla de juego.

2.2 Mensajes

Tras diseñar la pantalla de juego, localizamos los elementos que intervienen para saber que cual será la lista de mensajes.

Los elementos que intervienen son la posición de las paletas, la posición de la bola y los límites del campo.

Para indicar la posición de un elemento, lo daremos en coordenadas según el número de carácter donde se encuentran. La posición indicada será su esquina superior izquierda y se podrá saber toda la superficie que ocupan con las constantes que determinarán su tamaño.

Los límites del campo solo será necesario que los conozca el maestro, ya que se encarga de monitorizar si se provoca algún rebote, pero será necesario que el maestro indique cuando ha sucedido este rebote o si se ha marcado un punto.

La posición de la bola es necesaria que la conozcan todos los elementos, el

maestro para controlar sus acciones y actualizar su movimiento y los esclavos para representarla por pantalla. Como el maestro se encarga de su movimiento, este enviará un mensaje a los esclavos para indicar su nueva posición.
La posición de las paletas actúa del mismo modo que la posición de la bola, solo que es controlada por los esclavos en lugar del maestro y que el maestro no necesita estas posiciones para representarlas, sino para controlar si ha sucedido algún rebote o se ha marcado algún punto.
Además, se debe tener en cuenta la acción de saque, que indicarán los esclavos al maestro, ya que es una acción tomada por el jugador.
Atendiendo a todas estas demandas, escribimos la lista de mensajes:

id	Nombre	Acción	Transmisor	Receptor	Datos
00	M_BALL	Posición actual de la bola	Maestro	Esclavos	Coordenada X de la bola (Integer), Coordenada Y de la bola (Integer)
02	M_BOUNCE	Rebote contra paredes O paletas	Maestro	Esclavos	-
04	M_POINT	Marcación de un punto	Maestro	Esclavos	Ganador (1:2)
10	S1_PADDLE	Posición actual de la Paleta del jugador 1	Esclavo1	Maestro, Esclavo2	Coordenada Y de la paleta Del jugador 1 (Integer)
11	S1_SERVICE	El jugador 1 ha realizado El saque	Esclavo1	Maestro	-
20	S2_PADDLE	Posición actual de la Paleta del jugador 2	Esclavo2	Maestro, Esclavo1	Coordenada Y de la paleta Del jugador 2 (Integer)
21	S2_SERVICE	El jugador 2 ha realizado El saque	Esclavo2	Maestro	-

Fig. 4: Tabla de mensajes

El porqué del envío de tan solo la coordenada y de las paletas será explicado en el diseño del maestro y los esclavos.
Nótese que los identificadores siguen una estructura. Los nombres empezados por M indican mensajes enviados por el maestro, y los empezados por Sn indican mensajes enviados por el esclavo n. Luego, los identificadores empezados por 0 indican lo mismo que la M, y los empezados por el 1 o el 2 indican que esclavo los ha enviado. Además, todos los identificadores son pares excepto los de servicio. Usamos este patrón para poder filtrar los mensajes en la red CAN.

2.3 Maestro

El diseño del maestro ha sido realizado en pseudocódigo, mostrando con este las variables que maneja y las funciones que realizará este nodo.

```
constantes {
    ancho_pantalla = 40
    lado_pantalla = 30
    bola_lado = 1
    paleta_largo = 6
    paleta_ancho = 2
}
variables globales {
    bola (bx, by)
    paleta1(p1x, p1y) -> p1x es constante
    paleta2(p2x, p2y) -> p2x es constante
    velocidad
    saque si/no
    posesión_saque
    anugulox, anguloy
```

```
}
```
Nótese que p1x y p2x son constantes, ya que las paletas no avanzarán ni retrocederán horizontalmente, y que se tendrán como variables globales por legibilidad y consistencia. En la fase de diseño se añadirán como constantes y al ser así, no se enviarán en los mensajes de paso de coordenadas de las paletas.

```
inicialización {
    saque = si
    posesión_saque = 1
    bx = p1x + paleta_ancho + 1
    by = p1y + [paleta_largo/2 - bola_lado]
    velocidad = 0
    angulox = 1, anguloy = random{-1, 1}
    p1x = constante
    p1y = lado_pantalla/2 - paleta_largo/2
    p2x = constante
    p2y = lado_pantalla/2 - paleta_largo/2
}
bucle {
    // modo = {0->nada, 1->rebote, 2->punto}
    modo = 0, ganador = 0
    si (saque)
        bx = p1x + paleta_ancho + 1
        by = p1y + [paleta_largo/2 - bola_lado]
    si no
        bx += angulox
        by += anguloy
    ctrlGolpePaletas()
    si (golpe)
        modo = 1
        alterar_angulo()
    ctrlGolpeBordes()
    si (golpe)
        modo = 1
        alterar_angulo()
    ctrlPunto()
    si (punto)
        modo = 2
        ganador = {1, 2}
        actualizar_posición_bola_segun_ganador()
    si (modo = 1) enviar_mensaje_rebote()
    si (modo = 2) enviar_mensaje_punto(ganador)
    enviar_mensaje_posición_bola(bx, by)
    delay(velocidad)
}
ISR_esclavo1 {
    si (id = posición_paletas) actualizar_posición_paletas()
    si (id = saque_realizado) saque = no, angulox = 1, anguloy = random{-1, 1}
}
ISR_esclavo2 {
```

```
    si (id = posición_paletas) actualizar_posición_paletas()
    si (id = saque_realizado) saque = no, angulox = -1, anguloy = random{-1, 1}
}
ISR_potenciómetro {
    velocidad = lógica_potenciómetro
}
```

2.4 Esclavos

```
constantes {
    ancho_pantalla = 40
    lado_pantalla = 30
    bola_lado = 1
    paleta_largo = 6
    paleta_ancho = 2
}
variables globales {
    bola (bx, by)
    paleta1(p1x, p1y) -> p1x es constante
    paleta2(p2x, p2y) -> p2x es constante
    marcador 1, marcador 2
    cursor (cx, cy)
}
inicialización {
    bx = p1x + paleta_ancho + 1
    by = p1y + [paleta_largo/2 - bola_lado]
    p1x = constante
    p1y = lado_pantalla/2 - paleta_largo/2
    p2x = constante
    p2y = lado_pantalla/2 - paleta_largo/2
    marcador1 = 0, marcador2 = 0
}
bucle {
    enviar_mensaje_posicion_paleta(pNy)
    pintar_pantalla()
}
```

Como se ha indicado anteriormente, al ser la coordenada x de las paletas constantes, enviamos tan solo la coordenada y, reduciendo así la complejidad de operación y el peso del mensaje.

```
ISR_PC {
    si (arriba) pNy -= 1 (min.: 0)
    si (abajo) pNy += 1 (max.: ancho_pantalla - paleta_largo)
    si (saque) enviar_mensaje_saque()
}
ISR_esclavoM {
    si (id = posición_paletas) actualizar_posición_paletas()
}
```

En este caso en concreto, al filtrar los mensajes recibidos, no sería necesario identificar el mensaje, ya que del otro esclavo solo puede recibir un mensaje, pero en la implementación, las ISR tanto del otro esclavo como del maestro irán juntas, por lo tanto es correcto ya añadir la comprobación en el pseudocódigo.

```
ISR_maestro {
   si (id = posición_bola) actualizar_posición_bola()
   si (id = rebote) indicar_PC_zumbido()
   si (id = punto) aumentar_marcador(ganador)
}
```

3. Implementación:

La implementación aquí mostrada es la implementación final. Esta ha ido cambiando según se han ido realizando las pruebas. En el apartado Pruebas se indicarán los problemas encontrados y como se han ido solucionando, siendo estas soluciones reflejadas en el código que sigue a continuación.

3.1 Maestro

Usamos el pseudocódigo y la lista de mensajes generada anteriormente para implementar en C el código del nodo maestro. A continuación será dividido y explicado por partes.

```
#define WIDTH       80
#define LENGTH      24

#define BALL_L      1
#define PADDLE_L    5
#define PADDLE_W    2

#define PAD1_X      2
#define PAD2_X      76

#define SERV_NO     0
#define SERV_YES    1

#define M_BALL      00
#define M_BOUNCE    02
#define M_POINT     04
#define S1_PADDLE   10
#define S1_SERVICE  11
#define S2_PADDLE   20
#define S2_SERVICE  21
```

En las constantes podemos ver reflejadas las constantes nombradas en el pseudocódigo y la aparición de nuevas constantes para la implementación. Estas son las posiciones fijas de las paletas en el eje X, los valores de saque y los identificadores de los mensajes, para así aumentar la legibilidad del código.

Notése el cambio de las constantes del tamaño de la pantalla, y de las dependientes a ella como las posiciones de las paletas, para adaptarse mejor a la interfaz de Hyperterminal.

```
void master_init() {
    srand(time(NULL));
    // Initial service
    service = SERV_YES;
    pos_service = 1;
```

```
    // Initial paddle coordinates
    p1x = PAD1_X;
    p1y = (LENGTH/2) - (PADDLE_L/2);
    p2x = PAD2_X;
    p2y = (LENGTH/2) - (PADDLE_L/2);

    // Initial ball coordinates
    bx = p1x + (PADDLE_W) + 1;
    by = p1y + (PADDLE_L/2);

    // Initial ball speed
    speed = 0;

    // Initial ball movement vector
    vector_x = 1;
    vector_y = ((rand() % 2) == 0) ? -1 : 1;
}
```

La inicialización del maestro también sigue la estructura del pseudocódigo. Como detalles de la implementación, notamos el uso de las funciones srand() y rand() para generar números pseudoaleatorios y el uso de un operador condicional para elegir entre el -1 y el 1. Se ha optado por el uso de este operador para evitar usar una estructura condicional para elegir el valor de una sola variable.

```
int main(void) {
    CAN_config();
    ADC_config();

    master_init();

    // mode: 0->nothing, 1->bounce, 2->point
    int mode, winner, i;
    unsigned int ball_coordinates[2];
    while (1) {
        mode = 0;
        winner = 0;

        // Update ball coordinates
        if (service) {
            if (pos_service == 1) {
                bx = p1x + (PADDLE_W) + 1;
                by = p1y + (PADDLE_L/2);
            } else {
                bx = p2x - 2;
                by = p2y + (PADDLE_L/2);
            }
        } else {
            bx += vector_x;
            by += vector_y;
```

```
        }

        // Check bounces
        unsigned char paddle_bounce = check_paddle_hit();
        if (paddle_bounce) {
            mode = 1;
            vector_x = (vector_x == 1) ? -1 : 1;
            if (bx < (WIDTH/2)) {   // If it bounced with
paddle 1
                if (vector_y == 1 && by < (p1y + (PADDLE_L/2)))
vector_y = -1;
                else if (vector_y == -1 && by > (p1y +
(PADDLE_L/2))) vector_y = 1;
            } else {            // If it bounced with paddle 2
                if (vector_y == 1 && by < (p2y + (PADDLE_L/2)))
vector_y = -1;
                else if (vector_y == -1 && by > (p2y +
(PADDLE_L/2))) vector_y = 1;
            }
        }
        unsigned char wall_bounce = check_wall_hit();
        if (wall_bounce) {
            mode = 1;
            vector_y = (vector_y == -1) ? 1 : -1;
        }

        // Check if someone has scored
        winner = check_point_made();
        if (winner) {
            mode = 2;
            if (winner == 1) {
                bx = p2x - 1;
                by = p2y + (PADDLE_L/2) - BALL_L;
                pos_service = 2;
            } else {
                bx = p1x + (PADDLE_W) + 1;
                by = p1y + (PADDLE_L/2) - BALL_L;
                pos_service = 1;
            }
            service = SERV_YES;
        }

        // Send messages
        if (mode == 1) CANSendMsg(M_BOUNCE, 0, NULL);
        else if (mode == 2) CANSendMsg(M_POINT, 1, &winner);
        ball_coordinates[0] = bx;
        ball_coordinates[1] = by;
        CANSendMsg(M_BALL, 2, ball_coordinates);
```

```
        // Wait until next update
        for (i = 0; i < 100-20*speed; i++) Delay5ms();
    }

    return 0;
}
```

El programa principal se inicia con la configuración del CAN y del conversor analógico-digital, junto a la inicialización del maestro.

En el bucle actualizamos las coordenadas de la bola. Aquí hacemos uso de las variables *service* i *pos_service* para actualizarlas correspondientemente. Se puede observar una ligera diferencia entre la actualización de *bx* cuando posee el saque el jugador 1 o cuando lo posee el jugador 2. Esto se debe a que las posiciones de las paletas indican su esquina superior derecha, por lo tanto es necesario o no añadirle su anchura al colocar la bola.

Seguidamente comprobamos si la bola ha rebotado por alguna paleta. En el caso afirmativo, invertimos el sentido de la bola y calculamos si irá hacia arriba o hacia abajo. La norma seguida es si la bola golpea la mitad superior y se dirige hacia abajo, se invierte su sentido vertical, si no, sigue igual. Viceversa si la bola va en sentido contrario. Para saber sobre que paleta ha chocado comprobamos en que mitad del campo se sitúa la bola.

A continuación comprobamos si la bola ha rebotado sobre una pared. En caso afirmativo tan solo se invierte su sentido vertical.

Luego comprobamos si se ha marcado un punto. En caso afirmativo, la función *check_point_made()* nos devuelve el ganador y según el ganador actualizamos la nueva posición de la bola y indicamos que el perdedor tiene la posesión del saque.

Finalmente enviamos los mensajes. Se ha usado la variable modo, como se indica en el pseudocódigo. En caso de haber un rebote tan solo se indica mediante un mensaje sin contenido. Si se ha marcado un punto, se envía quien ha sido el ganador. Y en un caso o en el otro o en ninguna, se envían la posición actual de la bola. Nótese que antes se guardan en un array temporal ya que la función *CANSendMsg()* requiere de un puntero a los datos como argumento.

Y posterior a la siguiente iteración se realiza un *delay* dependiendo de la velocidad. La velocidad tiene 5 niveles, de 0 a 4, y producirá esperas de 500ms, 400ms, 300ms, 200ms y 100ms respectivamente para cada nivel de velocidad.

```
unsigned char check_paddle_hit() {
    unsigned char hit = 0;

    if (bx <= p1x+PADDLE_W && by >= p1y && by < p1y+PADDLE_L)
        hit = 1;
    if (bx >= p2x && by >= p2y && by < p2y+PADDLE_L)
        hit = 1;

    return hit;
}

unsigned char check_wall_hit() {
    unsigned char hit = 0;
```

```
    if (by <= 0 || by >= LENGTH)
        hit = 1;

    return hit;
}

int check_point_made() {
    unsigned char point = 0;

    if (bx <= 0) point = 2;
    else if (bx >= WIDTH) point = 1;

    return point;
}
```

Las tres funciones que comprueban si ha habido rebotes o se ha marcado algún punto son muy sencillas. Tan solo comparan la posición de la bola con la posición de las paletas o de la pared. En el caso de *check_point_made()* se separan las comprobaciones de cada pared para identificar el ganador.

```
void _ISR _ADCInterrupt(void) {
    int ADCValue = ADCBUF0;      // get ADC value
    speed = ((float)ADCValue/1023.0)*4;
    IFS0bits.ADIF = 0;           // restore ADIF
}
```

La lógica del potenciómetro es muy simple. Tan solo obtiene su valor, que está entre 0 y 1023 dada la configuración del conversor, y lo normaliza entre un valor de 0 a 4.

```
void _ISR _C1Interrupt() {
    if (C1INTFbits.RX0IF == 1) {
        unsigned int id = C1RX0SIDbits.SID;
        switch (id) {
            case S1_PADDLE:
                p1y = C1RX0B1;
                break;
            case S1_SERVICE:
                if (pos_service == 1) {
                    service = SERV_NO;
                    vector_x = 1;
                    vector_y = ((rand() % 2) == 0) ? -1 : 1;
                }
                break;
            case S2_PADDLE:
                p2y = C1RX0B1;
                break;
            case S2_SERVICE:
                if (pos_service == 2) {
                    service = SERV_NO;
```

```
                    vector_x = -1;
                    vector_y = ((rand() % 2) == 0) ? -1 : 1;
                }
                break;
        }

        C1RX0CONbits.RXFUL = 0; // Clear reception full status
flag
        C1INTFbits.RX0IF = 0;
    }
    IFS1bits.C1IF = 0;
}
```

El tratamiento de los mensajes de los esclavos es una traducción literal del pseudocódigo, usando un *switch* para elegir entre los distintos identificadores posibles. Aquí se puede ver como se tratan en una misma rutina de servicio lo que eran dos rutinas en el pseudocódigo, ya que son disparadas por la misma interrupción. Nótese la inclusión de un *if* en la recepción de mensajes de saque, para evitar que el jugador que no tenga posesión de la pelota, es decir, el que ha marcado el punto, pueda sacar.

3.2 Esclavo 1

Se explicará la implementación de un esclavo en concreto, en ese caso el 1, ya que los códigos son simétricos.

```
#define WIDTH       80
#define LENGTH      24

#define BALL_L      1
#define PADDLE_L    5
#define PADDLE_W    2

#define PAD1_X      2
#define PAD2_X      76

#define SCORE1_X    31
#define SCORE2_X    44
#define SCORE_Y     1

#define UP          'i'
#define DOWN        'k'
#define SERVICE     'j'

#define FILL        "#"
#define BALL        "O"

#define M_BALL      00
#define M_BOUNCE    02
#define M_POINT     04
#define S1_PADDLE   10
#define S1_SERVICE  11
```

```
#define S2_PADDLE   20
#define S2_SERVICE  21
```
Puede observarse que muchas constantes son las mismas que las del nodo maestro. Se añade la posición de la esquina superior derecha de los marcadores para el jugador 1 y el jugador 2, ya que estos no se moverán. Las constantes de las acciones de la paleta son usadas por legibilidad y para poder cambiar que tecla se usa sin tener que cambiar el código. Por el mismo motivo se usa una constante para el carácter que se usará como relleno en paletas y marcador. Para la bola se usa un carácter distinto para diferenciarla cuando pase por encima del marcador.

```
// Previous versions of ball and paddle variables
unsigned int pre_bx, pre_by, pre_p1y, pre_p2y;
```
Dentro de las variables globales observamos estas variables que nos servirán para saber si se han producido cambios desde la última actualización de la pantalla. Nótese que de las paletas solo se guarda la coordenada y, ya que la coordenada x es constante.

```
void slave1_init() {
    // Initial paddle coordinates
    p1x = PAD1_X;
    p1y = (LENGTH/2) - (PADDLE_L/2);
    p2x = PAD2_X;
    p2y = (LENGTH/2) - (PADDLE_L/2);

    // Initial ball coordinates
    bx = p1x + (PADDLE_W) + 1;
    by = p1y + (PADDLE_L/2);

    // Initial scores
    score[0] = 0;
    score[1] = 0;

    // Initial previous values at same value
    pre_bx = bx;
    pre_by = by;
    pre_p1y = p1y;
    pre_p2y = p2y;
}
```
La inicialización es una traducción literal del pseudocódigo y muy parecida a la del nodo maestro. Cabe destacar la inicialización de los valores previos con los valores actuales.

```
int main(void){
    UARTConfig();
    CAN_config();

    slave1_init();
```

```
    int j;
    for (j = 0; j < 1600; j++) Delay5ms();
    clear_screen();
    draw_screen();
    while (1) {
        if (pre_bx != bx || pre_by != by || pre_p1y != p1y ||
pre_p2y != p2y)
            draw_screen();
    }

    return 0;
}
```

El programa principal tan solo realiza las configuraciones iniciales e inicia el bucle, que es muy sencillo: repinta la pantalla de juego en el caso de que hayan variado las posiciones la bola o de las paletas respecto a su valor previo. Antes de entrar en el bucle se realiza una espera para dar tiempo a los jugadores a configurar el equipo, y se realiza un pintado inicial.

```
    // Configure acceptance mask
    C1RXM0SIDbits.SID = 0b0000000001;    // Mask to check if sid
is odd or even
    C1RXM0SIDbits.MIDE = 1;              // Identifier mode as
determined by EXIDE
    C1RX0CONbits.FILHIT0 = 0;            // Link to acceptance
filter 0

    // Configure acceptance filters
    C1RXF0SIDbits.EXIDE = 0;             // Enable filter for
standard identifier
    C1RXF0SIDbits.SID = 0b000000000;     // Accept messages with
even identifier
```

Cabe destacar la implementación de los filtros en *CAN_config()*. Con la máscara se indica que bits se comprobarán de la id del mensaje, y con el filtro el valor que deben tener los bits indicados para ser aceptados. Como hemos indicados anteriormente, los esclavos tan solo aceptarán los mensajes pares, de aquí el hecho de tan solo aceptarse los mensaje cuyos último bit sea 0.

```
void _ISR _U1RXInterrupt() {
    unsigned char c = ReadUART1();

    if (c == UP) if (p1y > 0) {
        pre_p1y = p1y; p1y -= 1;
        CANSendMsg(S1_PADDLE, 1, &p1y);
    }
    if (c == DOWN) if (p1y < LENGTH-PADDLE_L) {
        pre_p1y = p1y; p1y += 1;
        CANSendMsg(S1_PADDLE, 1, &p1y);
    }
    if (c == SERVICE) CANSendMsg(S1_SERVICE, 0, 0);
```

```
    IFS0bits.U1RXIF = 0;
}
```
En la interrupción de UART identificamos la tecla que ha sido pulsada y actuamos en consecuencia. Si se mueve la paleta, se indica el movimiento con la obtención del valor previo y se obtiene el nuevo valor; si se realiza el saque, se envía un mensaje al maestro.
Se ha optado en añadir los mensajes de transmisión de la posición de las paletas en la interrupción, para tan solo enviar un mensaje en el caso de una actualización.

```
void _ISR _C1Interrupt() {
    if (C1INTFbits.RX0IF == 1) {
        int winner;
        unsigned int id = C1RX0SIDbits.SID;
        switch (id) {
            case M_BALL:
                pre_bx = bx;
                bx = C1RX0B1;
                pre_by = by;
                by = C1RX0B2;
                break;
            case M_BOUNCE:
                WriteUART1(7);          // Send the buzzer
character back to the UART
                while (BusyUART1());    // Wait until the
character is transmitted
                break;
            case M_POINT:
                winner = C1RX0B1;
                score[winner-1] = (score[winner-1] + 1) % 10;
                break;
            case S2_PADDLE:
                pre_p2y = p2y;
                p2y = C1RX0B1;
                break;
        }

        C1RX0CONbits.RXFUL = 0; // Clear reception full status
flag
        C1INTFbits.RX0IF = 0;
    }
    IFS1bits.C1IF = 0;
}
```
En la interrupción CAN se puede observar que no se recibe el mensaje de saque del otro esclavo gracias al filtro. Si se reciben las coordenadas de la bola, se actualizan guardado el valor previo para indicar el repintado de la pantalla. Si se recibe la indicación de rebote se ejecuta el timbre del ordenador mediante la transmisión de un carácter especial por UART. Si se ha realizado un punto, se

obtiene el ganador y se aumenta su marcador, volviendo a 0 en el caso que pasemos de 9 a 10. Si se obtiene un mensaje del otro esclavo, se actualizan las coordenadas de la otra paleta.

```
void draw_screen() {
    int i, j;
    // Clear screen and reset cursor
    clear_screen();

    // Draw player 1 paddle
    position_cursor(p1x, p1y);
    for (i = 0; i < PADDLE_L; i++) {
        for (j = 0; j < PADDLE_W; j++) {
            draw_fill();
        }
        for (j = 0; j < PADDLE_W; j++) {
            move_left();
        }
        move_down();
    }

    // Draw player 2 paddle
    position_cursor(p2x, p2y);
    for (i = 0; i < PADDLE_L; i++) {
        for (j = 0; j < PADDLE_W; j++) {
            draw_fill();
        }
        for (j = 0; j < PADDLE_W; j++) {
            move_left();
        }
        move_down();
    }

    // Draw player 1 scoreboard
    position_cursor(SCORE1_X, SCORE_Y);
    draw_number(score[0]);

    // Draw player 2 scoreboard
    position_cursor(SCORE2_X, SCORE_Y);
    draw_number(score[1]);

    // Draw ball
    position_cursor(bx, by);
    draw_ball();

    //Restore preview values
    pre_bx = bx;
    pre_by = by;
    pre_p1y = p1y;
```

```
    pre_p2y = p2y;
}
```

Esta función tan solo consiste en un seguido de acciones para ir pintando la pantalla, moviendo el cursor a la posición deseada y escribiendo el carácter FILL. No es código complicado, tan solo largo. Pero puede ser conveniente señalar algunos detalles.

Iniciamos la función borrando completamente la pantalla, ya que es más fácil que localizar el punto que ha sido modificado, borrarlo y pintar el nuevo punto. Además, en caso de optarse a borrar-y-pintar, tendríamos un problema al repintar la bola al pasar por encima del marcador, ya que deberíamos de localizar donde se ha borrado el carácter tras pasar la bola y repintar, o repintar de nuevo el marcador. Al borrar la pantalla se reinician a 0 los valores cx y cy, que indican la posición del cursor.

Para posicionarnos en un punto usamos la función *position_cursor()*, que mediante un conjunto de bucles posiciona el cursor según los parámetros dados. En esta función, usamos los carácteres especiales de Hyperterminal para movernos por la pantalla. Con cada movimiento también se actualizan las variables cx y cy.

Para pintar cada paleta, realizamos un doble bucle para ir pintando cada línea de la paleta. Haciéndolo así, podemos modificar los tamaños de las paletas sin tener que reescribir código.

Para pintar el marcador de cada jugador usamos la función *draw_number()*, la cual recibe un número del 0 al 9 por parámetro y lo representa en la posición actual del cursor en un cuadro de 4x5 carácteres.

La bola se pinta tras los marcadores para que esta salga por encima de los números.

Al finalizar reiniciamos los valores previos para evitar que en el bucle principal se siga repintando.

```
void draw_number(unsigned int number) {
    switch (number) {
        // ####
        // #  #
        // #  #
        // #  #
        // ####
        case 0:
            draw_fill();
            draw_fill();
            draw_fill();
            draw_fill();
                move_down();
                move_left();
                move_left();
                move_left();
                move_left();

            draw_fill();
                move_right();
```

```
            move_right();
        draw_fill();
            move_down();
            move_left();
            move_left();
            move_left();
            move_left();

        draw_fill();
            move_right();
            move_right();
        draw_fill();
            move_down();
            move_left();
            move_left();
            move_left();
            move_left();

        draw_fill();
            move_right();
            move_right();
        draw_fill();
            move_down();
            move_left();
            move_left();
            move_left();
            move_left();

        draw_fill();
        draw_fill();
        draw_fill();
        draw_fill();
        break;
    //  ##
    //  ##
    //  ##
    //  ##
    //  ##
    case 1:
            move_right();
        draw_fill();
        draw_fill();
            move_down();
            move_left();
            move_left();

        draw_fill();
        draw_fill();
            move_down();
```

```
        move_left();
        move_left();

    draw_fill();
    draw_fill();
        move_down();
        move_left();
        move_left();

    draw_fill();
    draw_fill();
        move_down();
        move_left();
        move_left();

    draw_fill();
    draw_fill();
    break;
// ####
//    #
// ####
// #
// ####
case 2:
    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();

    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
```

```
        move_down();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
    break;
// ####
//    #
// ####
//    #
// ####
case 3:
    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();

    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();

    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
    break;
// #  #
// #  #
```

```
// ####
//    #
//    #
case 4:
    draw_fill();
        move_right();
        move_right();
    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
        move_right();
        move_right();
    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();

    draw_fill();
        move_down();
        move_left();

    draw_fill();
    break;
// ####
// #
// ####
//    #
// ####
case 5:
    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();
```

```
        move_left();
        move_left();
        move_left();

    draw_fill();
        move_down();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();

    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
    break;
// #
// #
// ####
// #  #
// ####
case 6:
    draw_fill();
        move_down();
        move_left();

    draw_fill();
        move_down();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
```

```
        move_left();

    draw_fill();
        move_right();
        move_right();
    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
    break;
// ####
//    #
//    #
//    #
//    #
case 7:
    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();

    draw_fill();
        move_down();
        move_left();

    draw_fill();
        move_down();
        move_left();

    draw_fill();
        move_down();
        move_left();

    draw_fill();
        move_down();
        move_left();

    draw_fill();
    break;
// ####
// #  #
// ####
// #  #
// ####
```

```
case 8:
    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
        move_right();
        move_right();
    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
        move_right();
        move_right();
    draw_fill();
        move_down();
        move_left();
        move_left();
        move_left();
        move_left();

    draw_fill();
    draw_fill();
    draw_fill();
    draw_fill();
    break;
// ####
// #  #
// ####
```

```
        //    #
        //    #
        case 9:
            draw_fill();
            draw_fill();
            draw_fill();
            draw_fill();
                move_down();
                move_left();
                move_left();
                move_left();
                move_left();

            draw_fill();
                move_right();
                move_right();
            draw_fill();
                move_down();
                move_left();
                move_left();
                move_left();
                move_left();

            draw_fill();
            draw_fill();
            draw_fill();
            draw_fill();
                move_down();
                move_left();

            draw_fill();
                move_down();
                move_left();

            draw_fill();
                move_down();
                move_left();

            draw_fill();
            break;
    }
}
```

Como la anterior, esta función es muy larga pero sencilla. Tan solo consiste en un *switch* para elegir el número a representar a partir del argumento dado. Luego, se van escribiendo carácteres FILL según sea acorde o se avanza el cursor a la derecha. Al terminar la línea, se baja y se retrocede para volver a empezar.

4. Librerías:

En cuanto a librerías tan solo usamos las dadas por el compilador para la placa, p30f4011.h, y para la comunicación UART, uart.h, y para la comunicación CAN, que hemos creado, can.h.

```
#define MAX_MSG 8
```

```
// Transmite message
// DLC = msg's number of bytes
void CANSendBMsg(unsigned int id, unsigned int dlc, unsigned
char *msg);
// DLC = msg's number of integer
void CANSendMsg(unsigned int id, unsigned int dlc, unsigned int
*msg);
```

Podemos distinguir dos funciones que tienen la misma finalidad, enviar un mensaje con el identificador estándar. La diferencia entre estas dos funciones es que la primera transmite un mensaje de bytes y la segunda un mensaje de words, que equivalen al tamaño de un integer en la placa. La constante define el número máximo de bytes que se pueden enviar.

```
void CANSendMsg(unsigned int id, unsigned int dlc, unsigned int
*msg) {
    // Standard Id
    C1TX0SIDbits.SID5_0 = id;
    id = id >> 6;
    C1TX0SIDbits.SID10_6 = id;

    // RTR
    C1TX0DLCbits.TXRTR = 0;      // Normal message

    // IDE
    C1TX0SIDbits.TXIDE = 0;      // Standard identifier

    // DLC
    C1TX0DLCbits.DLC = dlc*2;    // Data Length Code

    int aux;
    if (dlc >= 1) {
        aux = msg[0];
        C1TX0B1 = aux;
    }
    if (dlc >= 2) {
        aux = msg[1];
        C1TX0B2 = aux;
    }
    if (dlc >= 3) {
        aux = msg[2];
        C1TX0B3 = aux;
    }
    if (dlc >= 4) {
        aux = msg[3];
        C1TX0B4 = aux;
    }

    C1TX0CONbits.TXREQ = 1;              // Send message
```

```
    while (C1TX0CONbits.TXREQ == 1);     // Wait until
successfully transmitted
}
```

La función primero inicia el mensaje con el identificador, el tipo de mensaje y el tamaño, que es el doble del dado ya que el argumento nos indica cuantos words tiene el mensaje.
Seguidamente se obtienen los primeros valores del mensaje pasado por argumento, que se guardan en la estructura del mensaje. Por lo tanto, se recibirá el mensaje en el orden original, facilitando así la lógica de recepción.
Para finalizar, se envía el mensaje y se espera en la misma función hasta su correcta transmisión, haciendo más transparente el envío de mensajes.

5. Pruebas:

El conjunto de pruebas realizado está compuesto por tres pruebas: prueba para asegurar que la comunicación UART es correcta, el nodo esclavo pinta la pantalla y se pueden mover las paletas correctamente; prueba para asegurar que la comunicación CAN entre nodo maestro y esclavo funcionan, con una implementación del juego en la cual el nodo esclavo controla los dos jugadores y se juega con el mismo ordenador; y prueba para asegurar que la comunicación CAN entre dos esclavos funciona y comprobar la comunicación UART con dos ordenadores distintos.

En la primera prueba se quiere comprobar que la comunicación UART funciona y que las funciones de pintado por pantalla son correctas. Se consigue mediante la implementación de un esclavo sin comunicación CAN. La interrupción UART quedaría así:

```
void _ISR _U1RXInterrupt() {
    unsigned char c = ReadUART1();

    if (c == UP) if (p1y > 0) {pre_p1y = p1y; p1y -= 1;}
    if (c == DOWN) if (p1y < LENGTH-PADDLE_L) {pre_p1y = p1y;
p1y += 1;}
    if (c == BUZZ) {
        WriteUART1(7);          // Send the buzzer character
back to the UART
        while (BusyUART1());    // Wait until the character is
transmitted
    }

    IFS0bits.U1RXIF = 0;
}
```

Esta prueba sirvió para corregir unos cuantos errores.
Para empezar, sirvió para ajustar las constantes, ya que se comprobó que las primeras constantes, reflejadas en el pseudocódigo, no se ajustaban a la interfaz de Hyperterminal.
El siguiente error fue una confusión en las funciones *move_down()* y *move_up()*, en las cuales el carácter especial estaba invertido.
El último error fue que las acciones no se tomaban hasta el cuarto movimiento, y desde entonces todas las acciones iban con un retraso de tres movimientos. Para

entenderlo mejor, esta es una secuencia de la impresión por pantalla si se imprimía la tecla pulsada por teclado:
[Pulsado] i i i i i j j k i k k
[Impreso] i i i i i j j k
El problema residía que UART tiene un buffer de tamaño 4 carácteres. Aunque por defecto la opción de buffer viene desactivada, con la configuración mediante la función *OpenUART1()* se activa, por lo tanto es necesario una desactivación explícita:

```
U1STAbits.URXISEL = 0;
```

El objetivo de la segunda prueba era comprobar que los mensajes enviados se recibían correctamente (que la configuración de la red y de las interrupciones era la correcta) y que el receptor realizada la acción correspondiente (que la lógica de la interrupción era la que tocaba y el formato del mensaje era el correcto). En esta prueba teníamos dos programas: uno que era el maestro y que actúa como su implementación final, ya que hay dos esclavos, sin distinguir si están en nodos separados o no; y el otro que actúa como los dos esclavos. Para conseguirlo, las principales modificaciones respecto a la implementación final del esclavo se encuentran en las constantes y en la interrupción UART, que hay duplicación de código para cada esclavo:

```
#define UP1         'w'
#define DOWN1       's'
#define SERVICE1    'a'
#define UP2         'i'
#define DOWN2       'k'
#define SERVICE2    'j'

void _ISR _U1RXInterrupt() {
    unsigned char c = ReadUART1();

    if (c == UP1) if (p1y > 0) {
        pre_p1y = p1y; p1y -= 1;
        CANSendMsg(S1_PADDLE, 1, &p1y);
    }
    if (c == DOWN1) if (p1y < LENGTH-PADDLE_L) {
        pre_p1y = p1y; p1y += 1;
        CANSendMsg(S1_PADDLE, 1, &p1y);
    }
    if (c == SERVICE1) CANSendMsg(S1_SERVICE, 0, 0);

    if (c == UP2) if (p2y > 0) {
        pre_p2y = p2y; p2y -= 1;
        CANSendMsg(S2_PADDLE, 1, &p2y);
    }
    if (c == DOWN2) if (p2y < LENGTH-PADDLE_L) {
        pre_p2y = p2y; p2y += 1;
        CANSendMsg(S2_PADDLE, 1, &p2y);
    }
    if (c == SERVICE2) CANSendMsg(S2_SERVICE, 0, 0);
```

```
    IFS0bits.U1RXIF = 0;
}
```
Esta prueba funcionó correctamente y tan solo sirvió como prueba para distinguir que placas funcionaban bien y en que rol (maestro o esclavo) y para implementar la optimización de enviar los mensajes de la posición de las paletas en la interrupción UART, para así evitar el parpadeo a causa de las continuas repintadas de pantalla.

En la tercera prueba se quería comprobar la sincronización entre esclavos. Que la comunicación CAN estuviera configurada correctamente y que los mensajes fueran del formato adecuado. Las principales modificaciones entre la implementación final de esclavo y esta prueba son tan solo la eliminación de la recepción de mensajes del maestro, por lo tanto no se muestra código a continuación.
Esta prueba sirvió para determinar que placas no funcionaban correctamente o que cables de conexión CAN no lo hacían. Una vez, y no puedo expresar suficiente cuanto tiempo dedicado y cuan importante era, se sabían que placas funcionaban correctamente gracias a la anterior prueba, ésta funcionó correctamente y se pudo pasar al pulido final de las implementaciones finales de maestro y esclavo.