



Trabajo Fin de Grado

Diseño y desarrollo de un simulador de comunicaciones RS-232 y RS-422 basado en Microcontrolador.

Alumno: Carlos I. Ayllón Fernández

Consultor: Aleix López Antón

Junio de 2023





Contenido

1. Introducción y objetivos
2. Estado del Arte
3. Arquitectura del Sistema
4. Diseño y desarrollo Hardware
5. Diseño y desarrollo Software
6. Verificación del Sistema
7. Resultados y Conclusiones

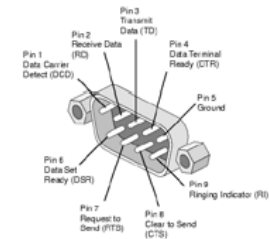


- Trabajo Fin de Grado



Introducción y objetivos

- Los estándares RS-232 y RS-422 son antiguos (60 años de historia), pero no están obsoletos, se siguen utilizando, diseñando y fabricando equipos que usan estos protocolos a cada vez mas velocidad de bits.
- Los PC's y herramientas disponibles no han seguido esta línea: Los puertos serie físicos han desaparecido de los PC's, abundan los adaptadores USB que adolecen de latencias, o *jitter* y son poco deterministas.



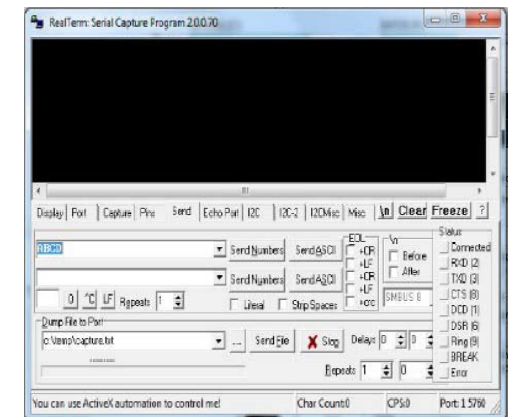
Desarrollar un prototipo funcional capaz de simular los estándares RS232 y RS422

- Con requerimientos técnicos específicos,
- Que funcione de manera autónoma y portátil,
- Totalmente configurable, preciso y determinista,
- Generando mensajes largos, en bucle repetitivo, a alta velocidad, con contador, etc.
- Aplicando todos los conocimientos adquiridos durante el grado



Estado del Arte

- Hardware: Convertidores USB a Serie
 - Basados en varios chips
 - Con conversores integrados que no suelen superar los 256Kbps
 - Buffers pequeños
- Software: Programas de terminal
 - Mas o menos evolucionados
 - Poca capacidad para generar retardos o bucles
- Abunda el software de análisis o captura (*sniffers*)
 - **No transmiten**, solo reciben, por lo que no son solución





- Soluciones completas Hardware/Software
 - Sistemas modulares: CompactRIO
 - Requiere al menos 2 módulos: RS232 y RS422
 - Coste elevado: Solo al alcance de profesionales
 - Requiere desarrollo LabView a medida
 - Poco portable



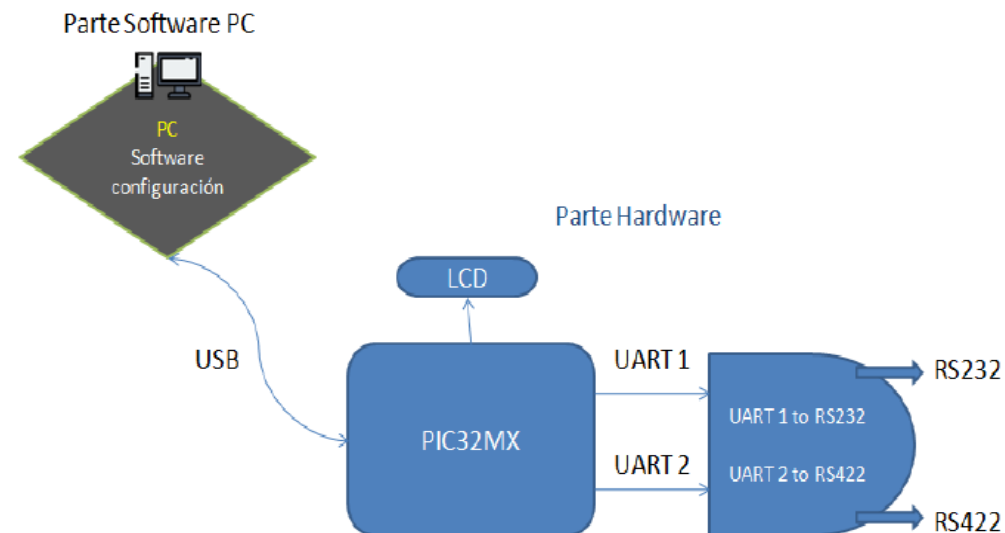
Sistema CompactRIO de National Instruments





Arquitectura del sistema

- Hardware que genera las señales
 - PIC32MX
 - LCD con status y configuración
 - Conversión TTL a RS232 o 422
- Software que configura el Hardware con las opciones deseadas



- Arquitectura del sistema

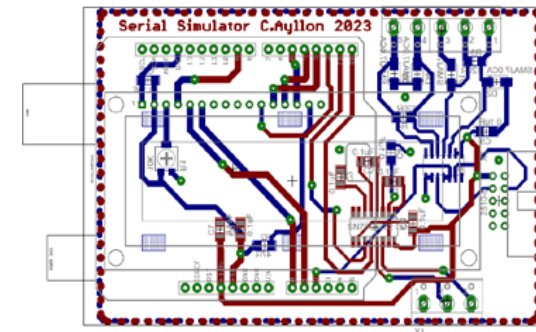
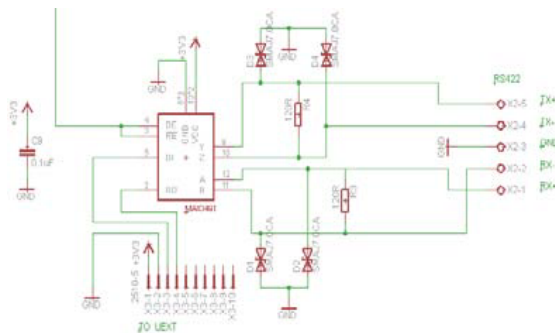




Parte Hardware → Electrónica y PCB

Desarrollo

- Diseño de la electrónica-esquema
 - Selección de componentes
 - Cálculos
 - Conversiones TTL a Serie y Pantalla LCD
- Diseño de la PCB estilo *shield*
 - Reglas de diseño y electromagnéticas (EMI)
 - Ruteado de pistas a doble cara

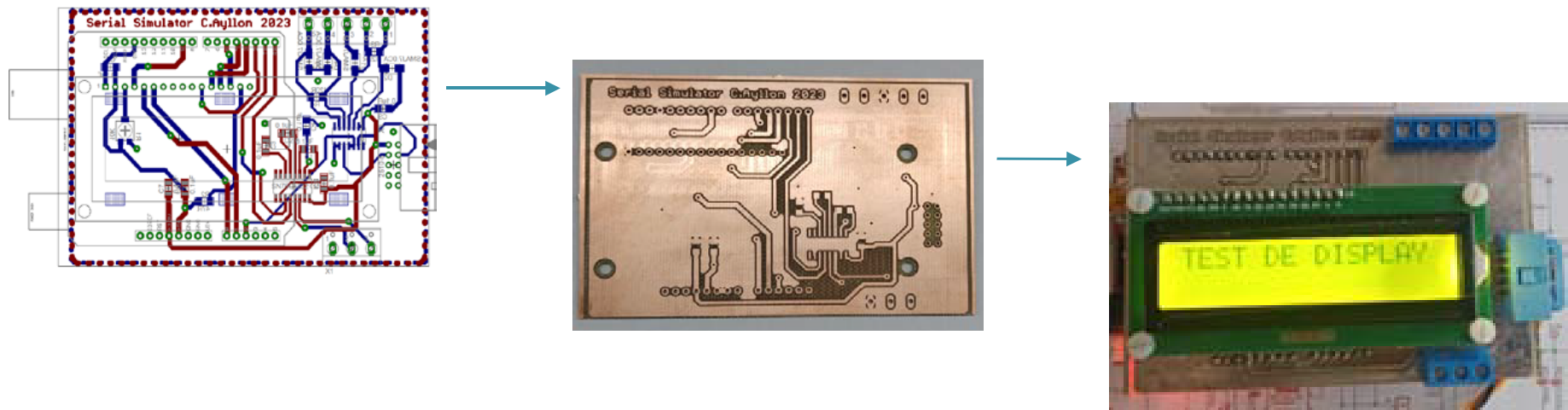




Parte Hardware → Electrónica y PCB

Fabricación:

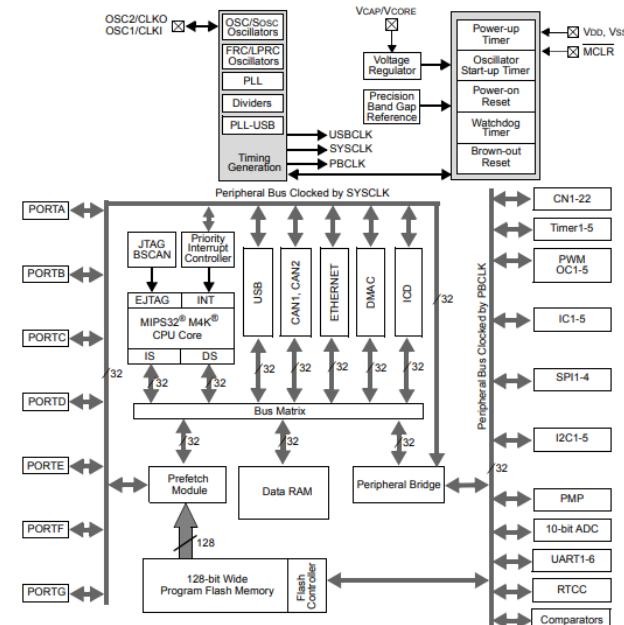
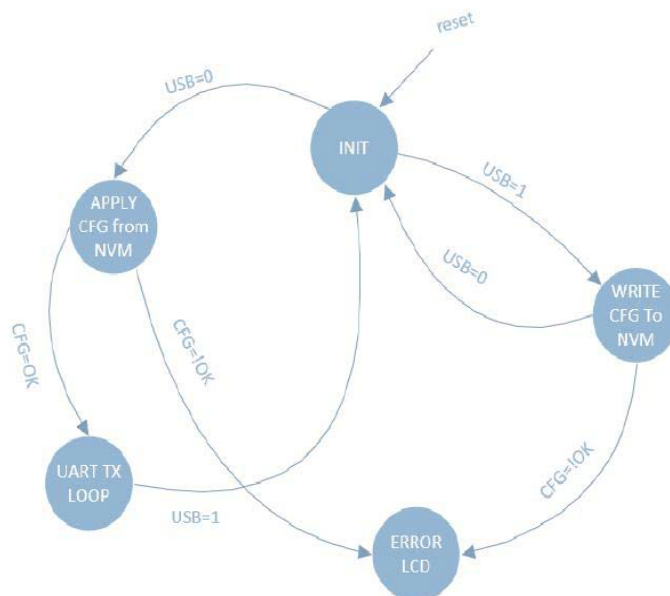
- Reproducción de la PCB en cobre FR4
 - Vías y taladros
 - Montaje y soldadura de componentes
 - Test y comprobaciones preliminares





Parte Hardware → Placa PIC32MX y Firmware

- Diseño y programación del Firmware
 - Diagrama de estados → Programa en lenguaje C
 - Estudio de datasheet y registros del microcontrolador
 - Cálculos de osciladores, timers y demás



- Diseño y desarrollo Hardware





Parte Hardware → Placa PIC32MX y Firmware

- Diseño y programación del Firmware
 - Programación MPLAB en lenguaje C
 - Registros y configuraciones del Microcontrolador
 - Conexión USB con PC, Mensajes en LCD
 - Non-Volatile Memory, UARTS, Timers , GPIO's
 - Control de errores
 - Depuración y pruebas preliminares

```

1190 // Write bytes starting at Row Address NVM_PROGRAM_PAGE
1191 DelayMs(2);
1192 NVMProgram (((void *)NVM_PROGRAM_PAGE), (const void*) Uart_Msg_Hex, size_msg, (void *) 0xA0002000);
1193 DelayMs(2);
1194 // Verify if data matches
1195 if(memcmp(Uart_Msg_Hex, (void *)NVM_PROGRAM_PAGE, size_msg)) // si error al comparar
1196 {
1197     // If not include
1198     LCD_SetPosition(1, 1);
1199     printf("Error ");
1200     Address = 0xA0008BC, Uart_Msg_Hex[0] = 0xA4A3A2A1;
1201     Address = 0xA0008C0, Uart_Msg_Hex[1] = 0xA8A7A6A5;
1202     Address = 0xA0008C4, Uart_Msg_Hex[2] = 0xA4E3E2E1;
1203     Address = 0xA0008C8, Uart_Msg_Hex[3] = 0x00000000;
1204     Address = 0xA0008CC, Uart_Msg_Hex[4] = 0x00000000;
1205     Address = 0xA0008D0, Uart_Msg_Hex[5] = 0x00000000;
1206     Address = 0xA0008D4, Uart_Msg_Hex[6] = 0x00000000;
1207     Address = 0xA0008D8, Uart_Msg_Hex[7] = 0x00000000; // escribe ok en direccion 3_c420
1208     Address = 0xA0008DC, Uart_Msg_Hex[8] = 0x00000000; // escribe en nvm tamaño de msg
1209     Address = 0xA0008E0, Uart_Msg_Hex[9] = 0x00000000;
1210     Address = 0xA0008E4, Uart_Msg_Hex[10] = 0x00000000;
1211     Address = 0xA0008E8, Uart_Msg_Hex[11] = 0x00000000;
1212     Address = 0xA0008EC, Uart_Msg_Hex[12] = 0x00000000;
1213     Address = 0xA0008F0, Uart_Msg_Hex[13] = 0x00000000;
1214     Address = 0xA0008F4, Uart_Msg_Hex[14] = 0x00000000;
1215     break;
1216 case CODE_CONF292: //
            
```



- Diseño y desarrollo Hardware



Parte Software → Aplicación de PC para configurar el HW

- Flujo de la aplicación → *Processing*
- Interacción con el PIC32 por USB
- Envío de mensajes y parámetros
- Control de errores
- Pruebas preliminares



```

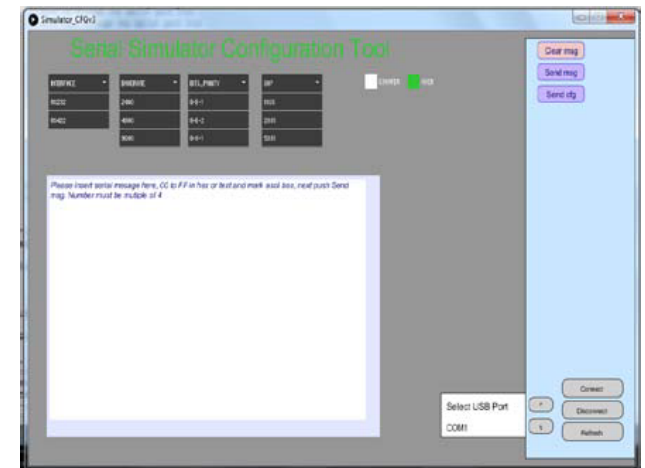
    #include <Arduino.h>
    #include <Serial.h>

    // Configuración de pines
    const int LED_PIN = 13;
    const int SERIAL_PIN = 2;

    // Variables globales
    bool led_on = false;
    bool config_ok = false;

    // Funciones de inicialización
    void setup() {
      pinMode(LED_PIN, OUTPUT);
      pinMode(SERIAL_PIN, INPUT);
      Serial.begin(9600);
    }

    void loop() {
      // Leer configuración por serial
      if (Serial.available()) {
        char c = Serial.read();
        if (c == 'C') {
          // Configuración de LED
          digitalWrite(LED_PIN, led_on);
          led_on = !led_on;
          Serial.println("LED: " + (led_on ? "ON" : "OFF"));
        } else if (c == 'F') {
          // Configuración de pines
          Serial.println("Pines: OK");
          config_ok = true;
        }
      }
    }
  
```

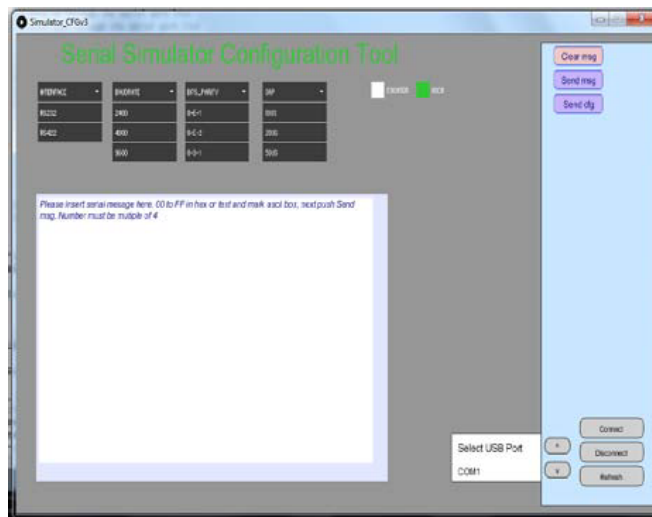




Verificaciones del Sistema

Pruebas de comunicación SW/HW

- Interacción Software de PC → Prototipo HW
- Comunicación con el PIC32 por USB
- Envío de mensajes y parámetros
- Control de errores
- Pruebas preliminares



USB



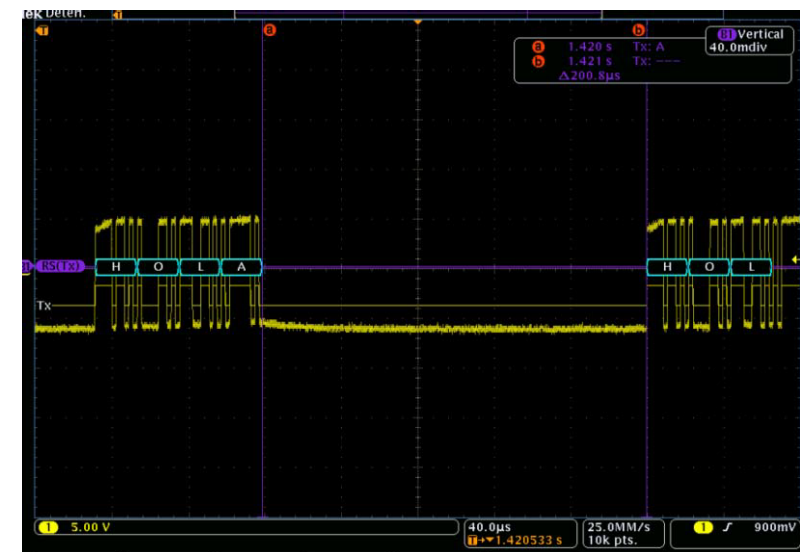
- Verificación del sistema





Pruebas de transmisión **RS-232**

- Configuración y generación de señales
- Comprobación de niveles de tensión
- Medidas de tiempos con osciloscopio
- Verificación del mensaje transmitido



Comprobado con todas las combinaciones de velocidad y configuración

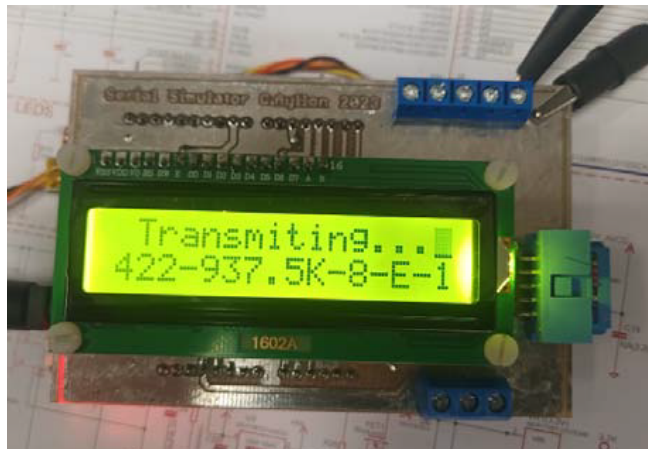
- Verificación del sistema





Pruebas de transmisión **RS-422**

- Configuración y generación de señales diferenciales
- Comprobación de niveles de tensión
- Medidas de tiempos con osciloscopio
- Verificación del mensaje transmitido



Comprobado con todas las combinaciones de velocidad y configuración





Resultados y Conclusiones

- Requisitos técnicos cumplidos y verificados en todo el rango de operación de velocidad y carga de pago.
- Prototipo plenamente funcional y utilizable.
- Diseño escalable y reutilizable
- Cumplimiento de la planificación prevista, presupuesto y ejecución del proyecto.
- Aplicando todos los conocimientos del grado y adquiriendo nuevas competencias





ii **GRACIAS** !!

Carlos Ayllón Fernández
cayllon@uoc.edu

