TEC | Tecnológico de Costa Rica

Área Académica de Ingeniería en Computadores

Curso: CE-5303 Introducción a los sistemas embebidos

Proyecto 2: Documento de diseño

Profesor: M. Luis Chavarría Zamora

Integrantes:

Calderón Yock Sebastián - 2018161630

Acevedo Rodríguez Kevin - 2018148661

Camacho Hernández Julián - 2019201459

Cartago 2023

Requisitos

Tabla 1. Requisitos de integración con la Raspberry Pi

Código	Requerimiento
RRP-01	El sistema debe ser Bare-Metal, no debe usar un OS.
RRP-02	El sistema debe usar Uboot como bootloader.
RRP-03	El sistema debe leer un sensor, de escogencia propia, en todo su rango de tensión.
RRP-04	El sistema debe contar con algún tipo de interfaz para interpretar los datos.
RRP-05	El sistema debe usar el **timer** (no usar NOPs) para el manejo temporal.
RRP-06	La comunicación de dispositivos del sistema debe ser serial (UART, SPI, I2C, etc)
RRP-07	No se pueden conectar periféricos para iniciar el sistema.

Tabla 2. Requisitos de la aplicación bare-metal

Código	Requerimiento
RAPP-01	Debe implementarse en Rust o C y representar la interfaz por medio de LEDs o en C++, pero representar la interfaz gráfica en una pantalla.

Tabla 3. Requisitos no funcionales

Código	Requerimiento
RNF-01	Debe utilizarse la plataforma EVM Raspberrypi 2 o 3.
RNF-02	El sistema debe integrarse con otro sistema empotrado que tenga ADC.
RNF-03	El sistema debe ser a la medida (recursos limitados a la aplicación).
RNF-04	La interfaz de usuario debe ser intuitiva y fácil de entender por el usuario.
RFN-05	No se debe conectar el teclado para iniciar el sistema.

Diseño

Diseño del sistema

El sistema se compone de cuatro partes (ver figura 1):

- **Raspberry Pi con Aplicación Bare-Metal**
  El núcleo del sistema reside en una Raspberry Pi 2, que ejecuta una aplicación bare-metal desarrollada en C++. Esta aplicación se compila mediante desarrollo cruzado y se agrega como un archivo *kernel7.img*. La inicialización del sistema se realiza desde una tarjeta SD que contiene los archivos *bootcode.bin*, *start.elf*, *kernel7.img* y un *config.txt*. La aplicación bare-metal tiene la tarea de leer una señal UART proveniente de un Arduino UNO, interpretarla y presentar la información de manera intuitiva en la pantalla. Este enfoque sin sistema operativo proporciona una eficiencia y rendimiento óptimos para la aplicación específica.

- **Arduino UNO como Interfaz de Lectura de Sensor de Proximidad**
  Un Arduino UNO actúa como interfaz para la lectura de una señal proveniente de un sensor de proximidad. Dado que el Arduino UNO cuenta con un convertidor analógico a digital (ADC), la Raspberry Pi 2, a través de la aplicación bare-metal, lee la señal del sensor a través de un pin UART en el Arduino. Esta comunicación permite la transferencia eficiente de datos entre el sensor de proximidad y la Raspberry Pi, aprovechando las capacidades de ambos dispositivos.

- **Circuito para Acondicionar un Sensor de Proximidad**
  Para mejorar la lectura del sensor ultrasónico, se implementa un circuito de acondicionamiento. Este circuito se encarga de preparar la señal del sensor ultrasónico para su lectura por parte del Arduino UNO. La información procesada se transmite luego a la Raspberry Pi a través de la comunicación UART, contribuyendo así a una lectura precisa y confiable del sensor de proximidad.

- **Bootloader con U-Boot para la Inicialización del Sistema**
  El sistema incorpora un bootloader creado con U-Boot, utilizando la configuración *rpi_2_defconfig*. Este bootloader se desarrolla mediante herramientas de desarrollo cruzado *arm-none-eabi-*. Su función principal es permitir el arranque de la aplicación bare-metal en la Raspberry Pi 2. La elección de U-Boot garantiza una inicialización eficiente y confiable del sistema, ofreciendo flexibilidad y adaptabilidad al entorno específico de la Raspberry Pi.

Resumen

Este diseño integral se centra en la interconexión armoniosa de componentes, desde la interfaz de lectura del sensor en el Arduino UNO hasta la presentación intuitiva de datos en la pantalla de la Raspberry Pi, todo ello respaldado por un sistema de arranque confiable implementado mediante U-Boot. Este enfoque modular y cuidadosamente planificado asegura un rendimiento óptimo y una integración sin problemas en el contexto de la aplicación propuesta.

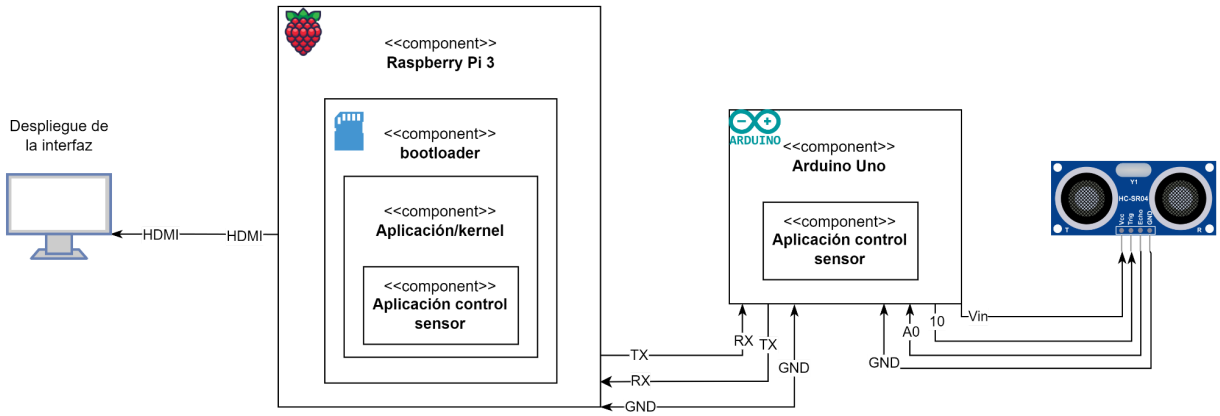

Figura 1. Diagrama de componentes

Descripción detallada de Métodos y Bibliotecas en la Aplicación Bare-Metal

1. **Métodos utilizados**

   a. **CKernel::Initialize():**
   Este método realiza la inicialización completa de las variables de la clase CKernel. Se encarga de configurar y poner en marcha los componentes esenciales del sistema, como la pantalla HDMI, la interfaz serie (UART), el sistema de interrupciones, y el temporizador del sistema.

   b. **clearBuffer(void pBuffer, size_t nSize):**
   Una función auxiliar que recorre un búfer y establece cada byte en 0, proporcionando una limpieza del búfer.

   c. **my_strlen(const char str):**
   my_strlen(const char str): Otra función auxiliar que itera sobre una cadena para determinar su longitud.

d. **CKernel::Run():**
   Este método es el núcleo de la aplicación. Se ejecuta continuamente, leyendo datos desde el Arduino a través del protocolo UART. Luego, presenta de manera intuitiva la distancia medida desde el sensor de proximidad en la pantalla HDMI. Utiliza el temporizador para introducir retardos y espera antes de realizar nuevas mediciones.

2. **Bibliotecas utilizadas:**

   a. **circle/string.h:**
      Se utiliza para operaciones relacionadas con cadenas de caracteres.

   b. **circle/debug.h:**
      Proporciona funciones de depuración y registro de información para seguimiento durante el desarrollo.

   c. **assert.h:**
      Utilizada para la aserción de condiciones específicas durante el desarrollo y ejecución.

   d. **circle/actled.h:**
      Ofrece acceso al LED de actividad de la Raspberry Pi para indicar la inicialización del kernel.

   e. **circle/koptions.h:**
      Se utiliza para manejar las opciones de configuración del kernel.

   f. **circle/devicenameservice.h:**
      Permite el mapeo de nombres de dispositivos, facilitando la gestión de dispositivos.

   g. **circle/screen.h:**
      Proporciona acceso y funcionalidad relacionada con la pantalla HDMI.

   h. **circle/serial.h:**
      Utilizada para inicializar y gestionar la interfaz serie (UART) para la comunicación con el Arduino.

   i. **circle/exceptionhandler.h:**
      Gestiona excepciones y abortos durante la ejecución del programa.

   j. **circle/interrupt.h:**
      Se utiliza para la inicialización y manejo de interrupciones del sistema.

k. **circle/timer.h:**
   Ofrece funcionalidades relacionadas con el manejo del tiempo y temporizadores del sistema.

l. **circle/logger.h:**
   Facilita el registro de eventos y mensajes del sistema.

Descripción detallada de Métodos en el Arduino UNO

1. **Definiciones de pines y constantes**

   a. **ECHO_PIN y TRIGGER_PIN:**
      Define los pines utilizados para la entrada del sensor ultrasónico (ECHO_PIN) y la salida del pulso ultrasónico (TRIGGER_PIN).

   b. **SECOND:**
      Representa la duración de un segundo en milisegundos y se utiliza para establecer los intervalos de medición.

   c. **SOUND_SPEED:**
      La velocidad del sonido en el aire, necesaria para calcular la distancia basada en el tiempo de vuelo del pulso ultrasónico.

2. **Método setup():**
   Configura la comunicación serial y los pines del Arduino para el sensor ultrasónico. Inicializa la comunicación serial a 115200 baudios y configura TRIGGER_PIN como salida y ECHO_PIN como entrada.

3. **Método loop():**
   Ejecuta repetidamente el proceso de generación de pulsos ultrasónicos, medición de distancia y envío de datos por la comunicación serial.

4. **Función generate_ultrasonic_pulse():**
   Genera un pulso ultrasónico para activar el sensor de proximidad.

5. **Función measure_distance():**
   Mide la distancia basada en el tiempo de vuelo del eco del pulso ultrasónico.

Verificación

Tabla 1. Verificación de la integración con la Raspberry Pi

Código	Requerimiento
RRP-01	Se verificará que no haya sistema operativo presente en la Raspberry Pi al ejecutar el sistema. Esto se puede comprobar revisando la configuración de inicio y la ausencia de un sistema operativo en el proceso de arranque.
RRP-02	Se confirmará que U-Boot sea utilizado como bootloader al revisar la configuración de inicio y los archivos de configuración de la Raspberry Pi.
RRP-03	Se verificará que el sistema sea capaz de leer datos del sensor seleccionado en todo su rango de tensión. Esto se puede realizar alimentando al sensor con diferentes tensiones y observando las lecturas resultantes.
RRP-04	Se comprobará que la interfaz de usuario implementada en Rust, C, o C++ sea capaz de interpretar y mostrar los datos provenientes del sensor. Esto se puede verificar observando la salida en la pantalla.
RRP-05	Se verificará que el sistema utilice el timer para gestionar los intervalos temporales, en lugar de depender de instrucciones NOP (no operation). Esto se puede comprobar revisando el código fuente del sistema y observando la precisión temporal.
RRP-06	Se confirmará que la comunicación entre dispositivos del sistema se realice a través de interfaces seriales como UART, SPI o I2C. Esto se puede verificar revisando la configuración de comunicación en el código fuente.
RRP-07	Se comprobará que el sistema no requiera la conexión de periféricos adicionales para iniciar. Esto se puede verificar revisando los requisitos de inicio y asegurándose de que no se dependa de dispositivos externos.

Tabla 2. Verificación de la aplicación bare-metal

Código	Requerimiento
RAPP-01	Se verificará que la aplicación esté implementada en Rust o C y que represente la interfaz mediante LEDs o en C++. Además, se comprobará que la interfaz gráfica sea visible en la pantalla de la Raspberry Pi. Esto se puede realizar revisando el código fuente y observando la salida en la pantalla.

Tabla 3. Verificación de los requisitos no funcionales

Código	Requerimiento
RNF-01	Se confirmará que el sistema esté configurado y ejecutándose en la plataforma Raspberry Pi 2 o 3. Esto se puede verificar revisando la configuración del sistema y la plataforma de hardware utilizada.
RNF-02	Se comprobará que el sistema se integre de manera efectiva con otro sistema empotrado que cuente con un convertidor analógico a digital (ADC). Esto se puede verificar revisando la comunicación entre sistemas y asegurándose de que los datos del sensor sean correctamente interpretados.
RNF-03	Se verificará que el sistema esté diseñado para operar con recursos limitados y que no utilice más recursos de los necesarios para la aplicación específica. Esto se puede comprobar revisando el código fuente y asegurándose de que no haya redundancia o uso innecesario de recursos.
RNF-04	Se comprobará que la interfaz de usuario implementada en Rust, C, o C++ sea intuitiva y fácil de entender para el usuario. Esto se puede verificar realizando pruebas de usuario o evaluando el diseño de la interfaz.
RFN-05	Se comprobará que el sistema no requiera la conexión de un teclado para iniciar. Esto se puede verificar revisando los requisitos de inicio y asegurándose de que no se dependa de la entrada de un teclado externo.