Laboratorio 4 Microcontrolador

Carlos Hurtado Villalobos
Tecnológico de Costa Rica
carlosh@estudiantec.cr

Luis Pablo Vargas Muñoz
Tecnológico de Costa Rica
pablovargas@estudiantec.cr

Jean Paul Dobrosky Espionoza
Tecnológico de Costa Rica
2021079467@estudiantec.cr

Francisco Javier Suarez Sarmiento
Tecnológico de Costa Rica
chiscojavi2000@estudiantec.cr

Víctor Sánchez Cáceres
Tecnológico de Costa Rica
vicsanca19@estudiantec.cr

***Abstract*—This document presents the problem statement and the research questions associated with the design and implementation of a digital system based on a microcontroller with the RV32I instruction set, synthesizable on an FPGA and aimed at temperature monitoring through memory-mapped peripherals.**

***Resumen*—Este documento presenta el planteamiento del problema y las preguntas de investigación asociadas al diseño e implementación de un sistema digital basado en un microcontrolador con el conjunto de instrucciones rv32i, sintetizable en FPGA y orientado a la monitorización de temperatura mediante periféricos mapeados en memoria.**

***Index Terms*—RISC-V, microcontroller, FPGA, SystemVerilog, memory-mapped peripherals.**

***Palabras clave*—RISC-V, microcontrolador, FPGA, SystemVerilog, periféricos mapeados en memoria.**

I. Planteamiento del problema

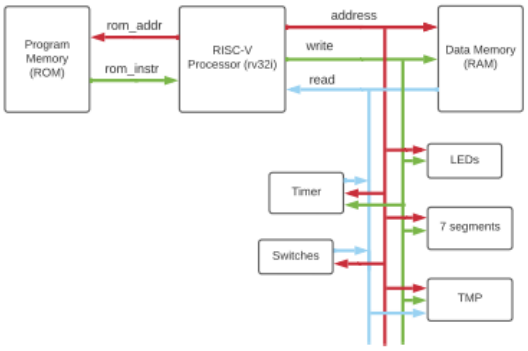

Figura 1: Diagrama de bloques del microcontrolador a desarrollar.

Figura 1. Diagrama de bloques del microcontrolador a desarrollar.

El laboratorio plantea el diseño de un sistema digital cuyo núcleo es un microcontrolador basado en el subconjunto de instrucciones rv32i. En la Figura 1 se observa que el procesador RISC-V debe coordinar una memoria de programa (ROM), una memoria de datos (RAM) y varios periféricos mapeados en memoria (temporizador, LEDs, switches, módulo de 7 segmentos y módulo de temperatura). Esta arquitectura obliga a definir buses de direcciones y datos bien estructurados, así como señales de lectura y escritura que garanticen la transferencia correcta de información entre el procesador y cada bloque [1], [4]. El problema no se limita a conectar módulos, sino a asegurar que la ruta de datos y la ruta de control funcionen de forma coherente en todo el sistema.

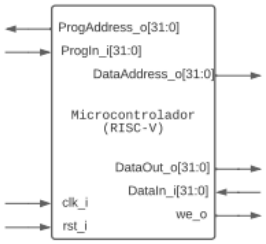

Figura 2: Diagrama de alto nivel del *core* para el microcontrolador.

Figura 2. Diagrama de alto nivel del *core* para el microcontrolador.

La Figura 2 detalla la interfaz del *core* RISC-V, donde se definen las señales `ProgAddress_o`, `ProgIn_i`, `DataAddress_o`, `DataOut_o`, `DataIn_i` y `we_o`, además de las entradas de reloj y reinicio. Esta descripción evidencia que el procesador debe operar con buses de 32 bits y con un único reloj de 10 MHz, lo que impone restricciones temporales claras sobre todos los módulos conectados [2]. Cualquier error en la generación de direcciones, en la activación de `we_o` o en la sincronización con el reloj puede causar lecturas o escrituras incorrectas en memoria y, en consecuencia, en un funcionamiento no determinista del sistema.

El mapa de memoria mostrado en la Figura 3 especifica la distribución de ROM, RAM y registros de periféricos dentro del espacio de direcciones. La ROM se ubica desde 0x0000, la RAM a partir de 0x1000 y los periféricos entre 0x2000 y 0x2034, incluyendo registros de datos y de control para switches, LEDs, 7 segmentos, temporizador y sensor de temperatura. Bajo este esquema, el microcontrolador debe

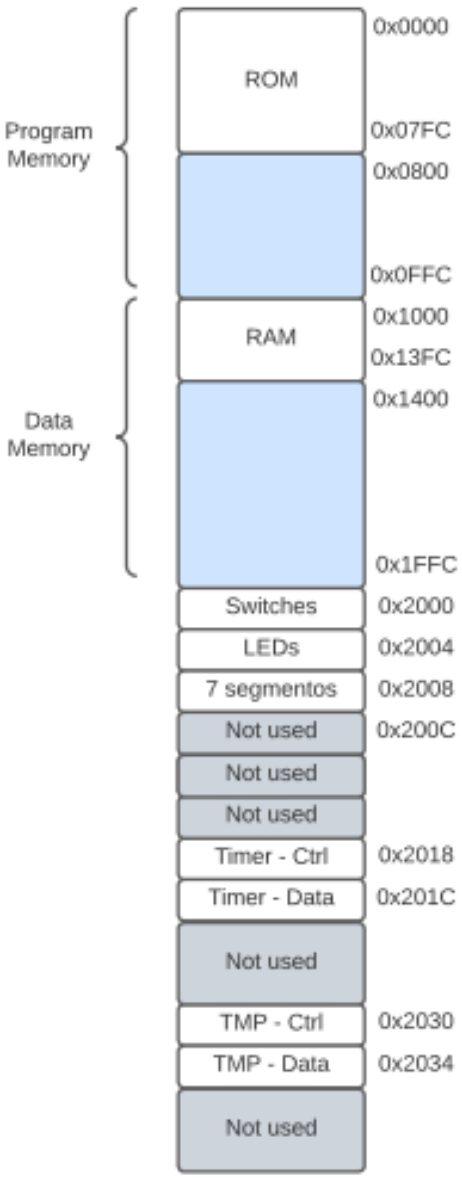

Figura 3: Mapa de memoria.

Figura 3. Mapa de memoria del sistema basado en RISC-V.

interpretar correctamente las direcciones generadas por el programa ensamblador y dirigirlas al bloque correspondiente mediante un módulo de decodificación y multiplexación de buses [3]. Además, la aplicación de monitoreo de temperatura exige que el programa lea periódicamente el sensor, convierta el resultado a representación decimal y lo despliegue en el módulo de 7 segmentos, ajustando el intervalo de muestreo según el estado de los switches.

En síntesis, el problema se centra en desarrollar un sistema digital modular, basado en un *core* RISC-V rv32i, capaz de interactuar correctamente con un mapa de memoria definido y con periféricos mapeados, operando a 10 MHz y ejecutando una aplicación en lenguaje ensamblador para la monitorización de temperatura. La solución requiere integrar decisiones de arquitectura, diseño digital sintetizable y programación de bajo nivel de forma consistente y verificable en FPGA [1], [4].

II. Preguntas de investigación

Con base en el problema planteado, se establecen las siguientes preguntas de investigación para guiar el desarrollo del sistema:

1. ¿Qué instrucciones del subconjunto rv32i son indispensables para implementar un microcontrolador sintetizable capaz de ejecutar un programa de adquisición y despliegue de temperatura?

   La implementación de un microcontrolador sintetizable capaz de ejecutar un programa de contol de la temperatura utilizando el subconjunto rv32i (RISC-V 32-bit integer), necesita de un conjunto básico de instrucciones que cubran operaciones aritméticas, de control de flujo y acceso a memoria, dado que por esto se puede realizar las funciones de la ALU, los registros y los controladores. Las instrucciones fundamentales para realizar este sistema son:
   - Instrucciones de carga y almacenamiento (Load/Store): LW (Load Word) y SW (Store Word) las cuales se encargan de cargar y almacenar datos en la memoria.
   - Instrucciones de aritmética y lógica: ADD, SUB, AND, OR, XOR, SLL y SRL los cuales son necesarias para realizar operaciones aritméticas y lógicas en los datos de la temperatura y otros cálculos asociados.
   - Instrucciones de comparación: BEQ, BNE, BLT, BGE: que son decisiones basadas en el valor de la temperatura.
   - Instrucción de salto como JAL que se implementa en subrutinas, especialmente si se requiere un ciclo de adquisición de datos y procesamiento.
   - Instrucción de interrupciones: en caso de necesitar un temporizador o interrupciones.

2. ¿Cómo debe estructurarse el mapa de memoria para garantizar el acceso ordenado a la memoria de programa, memoria de datos y periféricos mapeados?

   Para garantizar un acceso ordenado y eficiente a la memoria de programa, memoria de datos y periféricos mapeados en un microcontrolador basado en el subconjunto rv32i, el mapa de memoria debe estar bien organizado para que los diferentes espacios de memoria y periféricos estén claramente definidos en rangos específicos y así no se sobreescriban unos a otros. La estructura a utilizar es la siguiente:

- Memoria de programa (ROM): Se encuentra en el rango más bajo de la memoria.
  La dirección base normalmente usada en los sistemas RISC-V comienza en la dirección 0x0000 y se extiende hasta 0x0FFF.
  Es por esto que se mantendrá la convención y se utilizará este espacio para albergar el código del programa a ejecutar en el microcontrolador.
- Memoria de datos (RAM): Esta memoria se encontrará justo después de la memoria de programa. El espacio asignado en este proyecto para guardar los datos de esta memoria va desde 0x1000 hasta 0x1FFF, iniciado desde 0x1000.
- Periféricos mapeados a memoria: Se apartó un espacio en memoria para los diferentes periféricos a usar (LEDs, Display de 7 segmentos, Timers, entre otros), permitiendo que sean accesibles mediante simples instrucciones de carga y almacenamiento, facilitando la implementación y uso de estos.

Dirección	Descripción
0x0000 - 0x0FFF	Memoria de programa
0x1000 - 0x1FFF	Memoria de datos (RAM)
0x2000 - 0x2034	Periféricos mapeados a memoria

Cuadro I
MAPEOS GENERALES DE LAS MEMORIAS

3. ¿Qué mecanismos de multiplexación y control permiten administrar de forma correcta los buses de datos, de direcciones y las señales de control entre los diferentes módulos?

   La administración eficiente de los buses de datos, direcciones y señales de control en un microcontrolador o procesador se logra mediante varios mecanismos de multiplexación y control:
   - **Multiplexación de buses:** La multiplexación permite que múltiples señales compartan un mismo bus. En muchos sistemas, los buses de direcciones y datos utilizan las mismas líneas físicas en diferentes momentos. Esto se logra mediante un multiplexor de control, que activa las señales adecuadas según si el procesador está leyendo, escribiendo o accediendo a un periférico.
   - **Señales de control:** Las señales de control coordinan las operaciones entre memoria y periféricos, asegurando un funcionamiento correcto del sistema. Entre las señales más comunes se encuentran:
     - **Señales de lectura/escritura:** Indican el tipo de operación realizada.
     - **MEM_READ:** Indica una operación de lectura desde memoria.
     - **MEM_WRITE:** Indica una operación de escritura hacia memoria.
     - **CS_MEM:** Selección del módulo de memoria.
     - **CS_UART:** Selección del periférico UART.
     - **IRQ:** Señal de interrupción generada por un periférico.
   - **Bus de direcciones y bus de control:**
     - **Bus de direcciones:** Especifica la ubicación de memoria o el registro del periférico al cual se realizará el acceso.
     - **Bus de control:** Gestiona las señales que coordinan las operaciones de lectura y escritura, activando las líneas necesarias para direccionamiento y control.
   - **Arbitraje del bus:** Cuando varios módulos requieren acceso al bus, el sistema de arbitraje determina cuál obtiene prioridad. En sistemas con prioridades fijas, un módulo de mayor prioridad accede primero, aunque esto puede generar bloqueos en módulos con baja prioridad.
   - **Buffers y latches:** Se utilizan para almacenar temporalmente señales de datos o direcciones durante las transiciones entre fases del ciclo del bus (lectura, escritura), evitando pérdida de información.
   - **Multiplexores (MUX):** Permiten seleccionar entre distintas fuentes para el bus de datos, facilitando la integración de múltiples módulos que comparten recursos.

4. ¿Qué implicaciones tiene la programación *bare-metal* sobre el diseño del software ensamblador y la interacción directa con los periféricos?

   La programación bare-metal implica escribir código para ejecutar directamente sobre el hardware sin la intervención de un sistema operativo. De manera que puede influir en varios aspectos del diseño del software ensamblador y en la interacción con los periféricos, por ejemplo:
   - Control del hardware: en un entorno bare-metal, el programador tiene acceso directo y completo a todos los recursos del sistema como lo son el registro del procesador, la memoria y los periféricos.
   - Manejo de interrupciones: Como no hay un sistema operativo que administre las interrupciones, el programador debe configurar y manejar las interrupciones de manera explícita. Como lo puede ser la comunicación UART.
   - Acceso directo a los periféricos: la interacción con los periféricos debe ser gestionada directamente a través de registros específicos, sin las abstracciones que ofrece un sistema operativo.
   - Restricciones de hardware: al estar tan cerca de los sistemas, el bare-metal a menudo se utiliza en

aplicaciones de tiempo real, donde los requisitos de latencia y sincronización son estrictos. Esto requiere que el programador tenga un profundo conocimiento de los ciclos de reloj del procesador y de cómo los periféricos interactúan con la CPU.

Es decir que la programación bare-metal otorga un control completo sobre el hardware, pero también impone la responsabilidades avanzadas en programación de bajo nivel y un entendimiento profundo de la arquitectura del sistema.

5. ¿Qué métodos de validación de pre–síntesis, post–síntesis y post–implementación son necesarios para asegurar que el microcontrolador ejecute correctamente las instrucciones y que los periféricos respondan como se espera?

   Para asegurar que un microcontrolador ejecute correctamente las instrucciones y que los periféricos responden como se espera, se deben aplicar una serie de métodos de validación a lo largo de las diferentes fases del diseño ya sea en pre o post síntesis, donde cada fase se enfoca en detectar posibles errores o inconsistencias en distintas etapas del diseño y la implementación. De esta forma podemos ver que la validación pre-síntesis en esta fase se puede diseñar en una implementación física, antes de que se convierta en una descripción de hardware que pueda ser implementada en la FPGA, sin dejar de lado la verificación formal, donde se usan métodos matemáticos para demostrar que el diseño cumple con las especificaciones sin necesidad de simulaciones exhaustivas.
   En la fase de validación post-síntesis esta se encarga del diseño sintetizado, con el objetivo de asegurar de que el funcionamiento sea correcto y que el mapeo físico no ha introducido errores.
   Y por último, en la etapa de validación de post-implementación, se ha colocado el diseño dentro de la FPGA para realizar la validación final para verificar que el diseño funcione correctamente en el hardware real.

6. ¿Qué requisitos temporales y funcionales deben cumplir los periféricos (ADC, temporizador, LEDS, *switches* y display de 7 segmentos) para garantizar un monitoreo estable y confiable de la temperatura ambiente?

   Para realizar un monitoreo estable y confiable de la temperatura ambiente en una FPGA como la Nexys, es fundamental comprender los requisitos temporales y funcionales de cada periférico involucrado:

   - **Convertidor Analógico–Digital (ADC):**
     - **Resolución:** Debe contar con suficiente resolución para representar con precisión la señal analógica del sensor de temperatura. Una resolución de **10 bits** es adecuada para obtener mediciones precisas en aplicaciones de monitoreo ambiental.
     - **Tasa de muestreo:** Debe ser capaz de capturar la variación de la señal del sensor sin pérdida de información. Se recomienda un rango entre **1 Hz y 10 Hz** para mediciones de temperatura, que suelen variar lentamente.
   - **Temporizador:**
     - Debe ofrecer buena **precisión y estabilidad**, ya que controla el intervalo entre lecturas del ADC.
     - Para un sistema de monitoreo ambiental, un rango típico de temporización entre **1 ms y 100 ms** permite gestionar adecuadamente el muestreo y la actualización de los datos.
   - **LEDs:** Utilizados como indicadores del estado del sistema.
     - **Indicadores de estado:** Permiten señalar condiciones como temperatura en rango seguro, alerta o peligro.
     - **Control de brillo:** Puede utilizarse para representar variaciones de temperatura de manera visual o para indicar niveles de intensidad.
   - **Switches:** Funcionan como entradas del usuario y permiten interactuar directamente con el hardware.
     - Pueden utilizarse para realizar ajustes, cambiar rangos de operación o seleccionar la unidad de medida.
   - **Display de 7 segmentos:**
     - Permite brindar una visualización clara y legible de la temperatura.
     - Según el diseño, se pueden requerir **hasta 5 dígitos** para representar temperaturas con mayor rango o precisión.
     - Puede incluirse un dígito adicional para indicar la unidad utilizada (**°C**, **°F** o **K**).

Referencias

[1] D. Patterson and J. Hennessy, *Computer Organization and Design RISC-V Edition*. Morgan Kaufmann, 2017.
[2] RISC-V Foundation, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.2," 2019.
[3] Xilinx, *Vivado Design Suite User Guide*. Xilinx Inc., 2020.
[4] M. M. Mano and C. R. Kime, *Digital Design*, 5th ed. Pearson, 2012.
[5] BENAVIDES ALBESIANO, Ricardo Augusto. Multiplexores. 2013.
[6] DE ORDENADORES, Arquitectura. Definicion de Memoria. Tesis Doctoral. Universidad Carlos III de Madrid.