UNIVERSIDAD NACIONAL DE TUCUMÁN

FACULTAD DE CIENCIAS EXACTAS Y TECNOLOGÍA

ELECTRÓNICA II para Ingeniería Electrónica (2025)

Actividad 9: Implementación de una computadora capaz de ejecutar instrucciones RV32I

Autor: Chaves, Gonzalo Efraín - Estudiante de Ing. Electrónica

Resumen

El presente trabajo detalla el diseño, simulación e implementación en hardware físico de un microprocesador de 32 bits compatible con la arquitectura RISC-V (conjunto de instrucciones RV32I) utilizando lenguaje VHDL.
El sistema desarrollado integra un datapath completo, diseñado tomando como guía los esquemas conceptuales de la cátedra, una Unidad de Control basada en una Máquina de Estados Finitos (FSM), memoria SRAM y un Crossbar para la gestión de periféricos de entrada/salida. Tras verificar su correcta arquitectura mediante testbenches, el diseño fue sintetizado exitosamente e implementado en una placa EDU-CIAA-FPGA, donde demostró su total capacidad operativa al decodificar y ejecutar un programa en código máquina que exhibe una cuenta hexadecimal en un display de 7 segmentos

Introducción

Arquitectura de Computadora

Según lo que menciona Harris & Harris(2012) una Arquitectura de Computadora es la visión que tiene el programador de la máquina.
La misma está definida de manera estricta por su conjunto de instrucciones (que funciona como el lenguaje o vocabulario de la computadora) y por las ubicaciones de los operandos (que incluye los registros y la memoria disponible para almacenar variables temporalmente).
La arquitectura representa un nivel de abstracción, no define ningún tipo de hardware subyacente, por lo que de la misma puede haber múltiples implementaciones físicas diferentes que la representen.

Arquitectura RISC-V

RISC-V es una arquitectura de conjunto de instrucciones de estándar abierto basada en los principios de RISC (Reduced Instruction Set Computer). Sigue los principios de diseño fundamentales:

- Simplicidad favorece la regularidad: Está diseñada para que las instrucciones tengan formatos consistentes,lo que simplifica la decodificación por parte del hardware.
- Hacer el caso común rápido: Incluye solo instrucciones simples y de uso frecuente, permitiendo que la microarquitectura sea pequeña y rápida.

Instrucciones RV32I

El RV32I es el conjunto básico de enteros de 32 bits. Es el “vocabulario” fundamental que permite a la computadora realizar operaciones aritméticas (sumar, restar), lógicas (comparaciones or, and) y de control de flujo (saltos).

RISC-V es un conjunto de registros donde se definen 32 registros de propósito general, cada uno de 32 bits:

- x0 (zero): está conectado físicamente a tierra y siempre contiene el valor 0.. NO se puede modificar lo cual es fundamental para simplificar operaciones comunes (como copiar un registro sumándole 0).

- x1 - x31: se utilizan para almacenar variables temporales, direcciones de retorno y argumentos de funciones.

Para mantener la regularidad en el hardware, RV32I utiliza formatos de longitud fija (32 bits) con campos en posiciones predecibles:

- Tipo R (Register): Para operaciones entre registros (add, sub) Utiliza tres campos de registros dos de fuente (rs1, rs2) y un destino (rd).
  Campos: opcode (7bits), rd (5), funct3 (3), rs1 (5), rs2 (5), funct7 (7)
- Tipo I (immediate): Para operaciones con constantes o cargas de memoria. Incluye un valor inmediato de 12 bits
  Campos: opcode, rd, funct3, rs1, imm[11:0].
- Tipo S (Store): Utilizado para instrucciones de guardado en memoria. El valor inmediato se divide en dos partes para que los registros fuente siempre estén en la misma posición que en el Tipo R.
- Tipo B (Branch): Variante del Tipo S para saltos condicionales. El inmediato representa un desplazamiento relativo a la dirección actual.
- Tipo U (Upper immediate): Para cargar constantes grandes de 20 bits en la parte superior de un registro.
- Tipo J (Jump): Utilizado para saltos incondicionales permitiendo desplazamientos largos.

Tomando los distintos tipos de instrucciones y observando que las mismas tienen distintas codificaciones, necesitamos entender cuáles de los 32 bits de la instrucción representan un valor de inmediato.

- Tipo I: es directo, los bits de la instrucción forman el inmediato de 12 bits
- Tipo S: Los bits (7 bits) y [11:7] (5 bits) se concatenan para formar los 12 bits del inmediato
- Tipo B: El inmediato de 13 bits se forma con: inst[31](signo), inst[7], ins[30:25], inst[11:8] y un 0 implícito al final.
- Tipo U: Los bits se cargan directamente en la parte superior, rellenando con 12 ceros a la derecha.
- Tipo J: El Inmediato de 21 bits se forma con inst[31], inst[19:12], inst[20], inst[30:21] y un 0 implícito.

Algunos Opcodes decimales son 51 (Tipo R), 19 (Tipo I) y 99 (Tipo J)
Ahora indicamos cómo sabe la ALU que operaciones hacer dependiendo de cada Opcode recibido

- Opcode decimal 51 (Tipo R - Operaciones entre Registros)
  La operación específica no depende solo del opcode. Se define mediante el campo funct3 y el campo funct7.
  Ejemplo: Si funct3 es 000 y funct7 es 0000000, la ALU hace una SUMA. Si funct7 cambia a 0100000, la ALU hace una RESTA.
- Opcode decimal 19 (Tipo I - Operaciones con constantes)
  Utiliza el campo funct3 para definir la operación (ej. addi, slti, andi). Como solo hay una fuente de registro, funct7 generalmente no se usa (excepto en desplazamientos)
- Opcode decimal 99 (Tipo J - Saltos condicionales)
  Para instrucciones de salto como beq(salto si igual), se realizan entre operaciones sucesivas.

Operación de la ALU: El hardware generalmente realiza una resta entre los registros rs1 y rs2.
Condición de Cero: Si el resultado de esa resta es 0, significa que ambos registros son iguales. La ALU activa una señal interna (bandera de cero o Zero flag).
Toma del Salto: La unidad de control verifica esta bandera de cero junto con el funct3 (que indica si buscamos igualdad o desigualdad). Si la condición se cumple, el PC se actualiza con la dirección calculada sumando el inmediato.

Microarquitectura de Computadora

De acuerdo a lo que menciona Harris & Harris (2012) la microarquitectura es la organización específica del hardware para implementar un conjunto de instrucciones. Se compone de un datapath, que incluye los elementos de almacenamiento, procesamiento y cómputo como la ALU y los registros, y una unidad de control, que dirige el flujo de datos mediante señales lógicas, descomponiendo la ejecución de una instrucción en pasos temporales. La microarquitectura también define cómo el núcleo de procesamiento se comunica con el mundo exterior, como interfaces de entrada y salida y los buses del procesador a las memorias.

Un aspecto fundamental es que diferentes microarquitecturas pueden ejecutar el mismo programa, pero variarán en su velocidad, costo, rendimiento, complejidad de hardware.

Resultados

Datapath

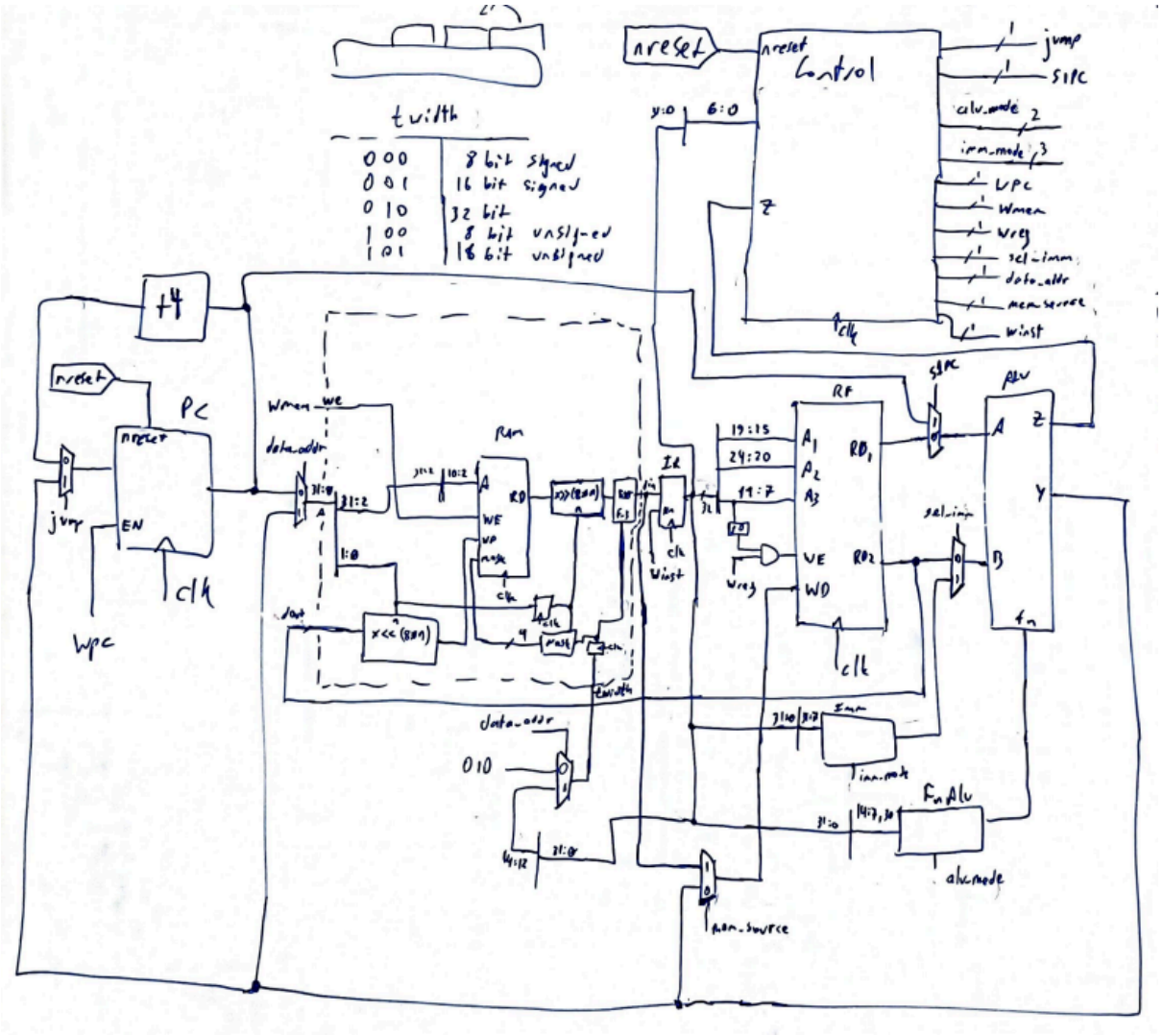

Figura 2.1 - Foto de una pizarra de la Cátedra, donde se muestra el datapath de un microcontrolador, compatible con RV32I

Para el diseño y desarrollo de la arquitectura interna de la CPU, se tomó como guía principal el esquema provisto por la cátedra, ilustrado en la Figura 2.1. A partir de esta base, se implementó un datapath que cuenta con los siguientes módulos

El Datapath de la Figura 2.1 cuenta con los siguientes módulos:

- Contador de programa
  Es un contador que indica la dirección de la próxima instrucción a ejecutar, el mismo siempre avanza de 4 en 4 lugares, pues cada instrucción de memoria es de 32 bits y cada dirección sólo guarda 8 bits, así que se necesitan 4 direcciones para guardar cada instrucción, por lo que la siguiente instrucción estará en la dirección actual +4.

- Registro de instrucción.

Es un registro donde se almacena la instrucción actual que se encuentra realizando el microprocesador.

- Conjunto de registros.
  Es donde el procesador guarda los datos que está manipulando activamente en un momento dado.

- Unidad aritmética lógica de 32 bit.
  Una ALU es el componente combinacional central del datapath encargado de realizar operaciones aritméticas (como suma y resta) y lógicas (como AND, OR y XOR) sobre datos binarios. En el caso de RV32I, la ALU opera con dos operadores de entrada de 32 bits y produce un resultado de 32 bits.

- Interfaz de memoria de 32 bit.
  Es el conjunto de conexiones y señales lógicas que permiten al procesador comunicarse con el sistema de almacenamiento externo.

- Unidad de Control.
  La misma es quien recibe las instrucciones de los registros, los decodifica, separa en pasos y dirige el flujo de datos a través de distintas señales digitales.

DIseño Máquina Control

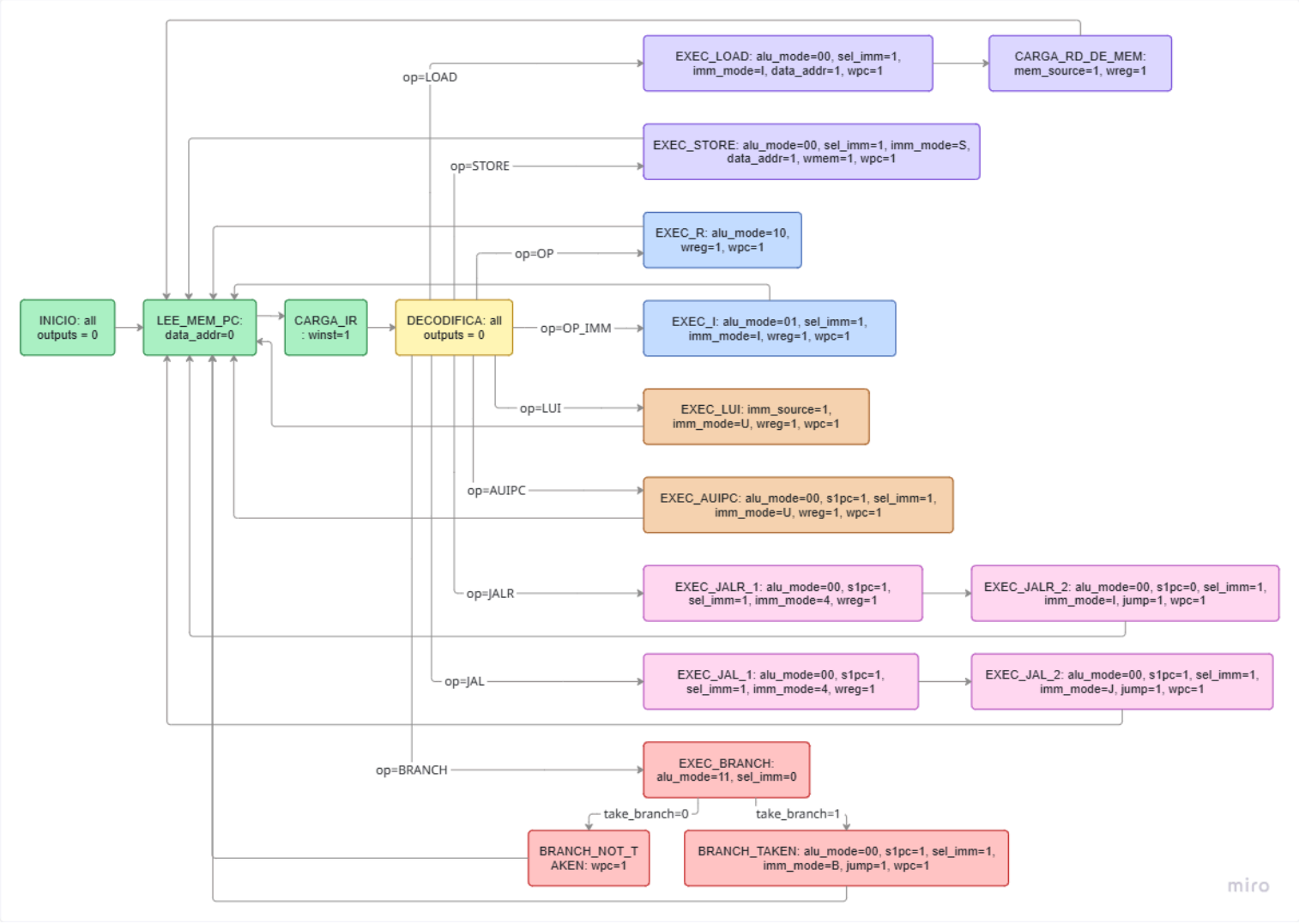

Figura 2.2 - Diseño de una Máquina de Estado Finito capaz de buscar en memoria y decodificar RV32I

En el diseño de la Figura 2.2 se muestran una Máquina de Estado Finito que lee un dato del registro de instrucciones, según el opcode del mismo, decodifica y separa la señales de salida según el tipo de operación requerida, las envía hacia la ALU donde le dice que hacer y acciona los distintos multiplexores para el correcto flujo de datos, luego de ser necesario guarda el dato en el banco de registro o en la SRAM segun sea necesario.De esta manera dirige u orquesta el flujo de datos a través del datapath.

Top Level

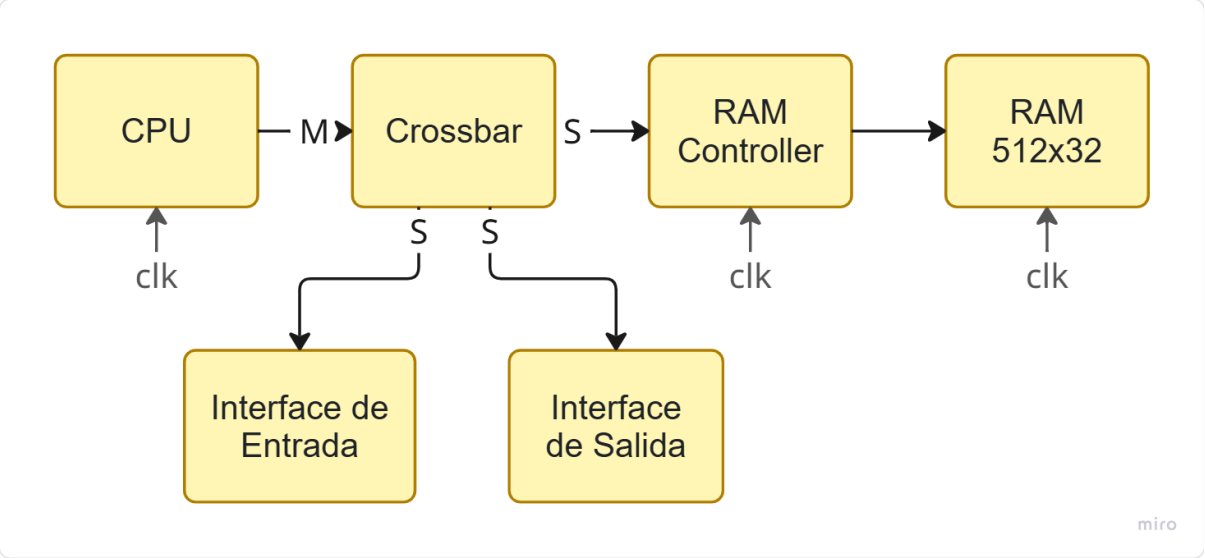

Figura 2.3 - Diagrama final de diseño, del microprocesador compatible con RV32I, con una interfaz de entrada y salida, de 8 bits cada una.

Top Level: Es el nivel de mayor jerarquía del diseño, encargado de integrar y conectar todos los módulos del sistema. Aquí se instancian la CPU, la memoria (Controlador de RAM y RAM de 512 x 32), las interfaces de entrada/salida y el Crossbar, la interconexión del mismo se hizo según lo visto en la Figura 2.3.

Crossbar: Es el módulo de interconexión responsable de administrar el tráfico de información entre el módulo Maestro (la CPU) y los múltiples módulos Esclavos (Memoria e Interfaces). Debido a que la CPU posee un único bus de datos y de direcciones, el Crossbar actúa como un enrutador: decodifica la dirección solicitada por la CPU, habilita únicamente al esclavo correspondiente para evitar colisiones de datos, y multiplexa la respuesta del esclavo activo de regreso hacia el procesador.

Simulaciones y Testbenches

CPU - Unidad Central de Proceso

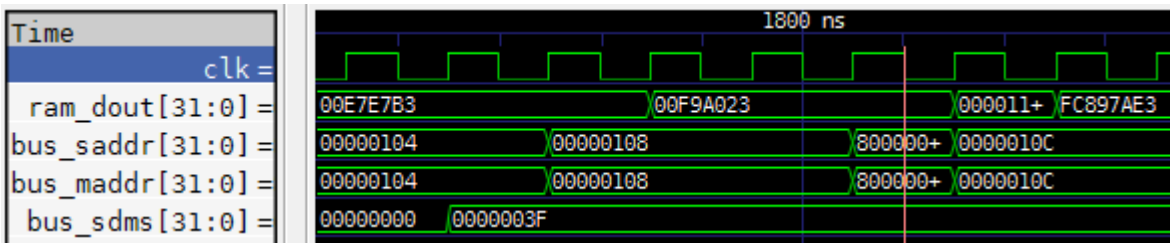

Figura 2.4 - Testbench de la cpu utilizada en el microprocesador

En la Figura 2.4 se muestra una simulación a partir de un testbench hecho para la CPU, donde podemos ver:
La lectura del registro de instrucciones ram_dout, luego el programa avanzando bus_maddr, y el momento donde la CPU escribe en el display, bus_saddr toma el valor de la dirección de lectura en 0x8000_0000, donde toma la lectura el periférico de salida y bus_sdms muestra el valor del display. que sería 3F el valor referente al '0' en RV32I, por lo que la cuenta descrita por el registro de instrucciones se encuentra en funcionamiento.

También se verificó el correcto conexionado de datos simulando el crossbar_tb.vhd y la lectura/escritura alineada con ram_controller_tb.vhd

Top y salida del display

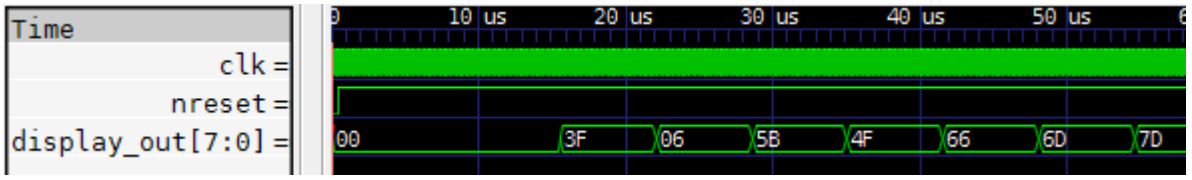

Figura 2.5 - Testbench del top

En la Figura 2.5 se expone el resultado de la simulación correspondiente al testbench del módulo de mayor jerarquía (top_tb.vhd). Esta prueba integra la totalidad del microprocesador hasta el punto de su implementación física en la placa EDU-CIAA-FPGA.

El entorno de simulación ejecuta un programa de prueba compuesto por instrucciones de la arquitectura RV32I, el cual procesa una cuenta y expone el resultado en un display de 7 segmentos.

La simulación permite verificar la evolución temporal de los valores de salida del display, validando así el correcto funcionamiento de todo el sistema antes de proceder a la síntesis y configuración de la FPGA.

Durante la etapa de desarrollo, se presentaron inconvenientes relacionados con la síntesis y el consumo de recursos. Inicialmente, la herramienta de síntesis no lograba mapear el banco de registros a los bloques de memoria RAM dedicados de la FPGA (BRAM), e implementarlos en su lugar mediante Flip-Flops y otras celdas. Esto provocaba un consumo excesivo de recursos lógicos y aumentaba considerablemente el tiempo de compilación. En

este escenario, el reporte indicaba el uso de solo 5 de los 9 bloques de memoria dedicados esperados (utilizando 4 para la memoria RAM de 512 x 32 y 1 para el circuito de Reset, pero omitiendo los 4 correspondientes al banco de registros).

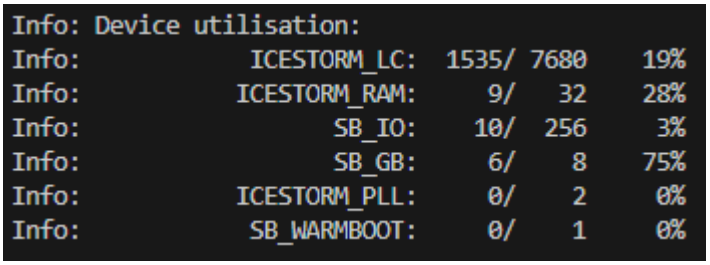

Figura 2.6 - Tabla resumen de Yosys sobre la utilización de recursos en la síntesis

Tras la corrección de estos errores de inferencia y la revisión de la descripción en VHDL de los registros, el diseño se logró sintetizar de manera óptima y eficiente, como se muestra en la Figura 2.6 (interpretar correctamente los 9 bloques de memoria). Finalmente, al realizar la prueba en el hardware físico (EDU-CIAA-FPGA), se comprobó el éxito del diseño: el sistema salió correctamente del estado de reset y el display comenzó a mostrar la cuenta en formato hexadecimal exactamente de acuerdo a lo esperado.

Conclusiones

Como conclusión, se logró integrar exitosamente los conceptos teórico-prácticos abordados a lo largo de Electrónica II. El proyecto partió desde el nivel más alto de abstracción, definiendo la arquitectura de la computadora y estudiando el conjunto de instrucciones (ISA) RV32I.

Durante el desarrollo, se aplicaron los principios de integración y diseño modular. Separar la CPU en bloques dedicados (como la ALU, el banco de registros y el crossbar) no solo facilitó la comprensión y el orden del código VHDL, sino que permitió aislar y solucionar errores de manera más sencilla. La microarquitectura también se integró al interconectar estas señales lógicas y recursos de hardware, donde el control del flujo de datos fue resuelto exitosamente mediante una Máquina de Estados Finitos (FSM). Esta FSM demostró ser fundamental para decodificar las instrucciones, coordinar a la ALU y administrar los tiempos del procesador.

Se recordaron los conocimientos sobre sistemas de almacenamiento, implementando desde estructuras básicas con Flip-Flops tipo D para los registros internos, hasta la comprensión de memorias SRAM complejas en los bloques dedicados (BRAM) de la FPGA.

El microprocesador resultante destaca por ser un sistema completamente funcional y sintetizable en hardware real (EDU-CIAA-FPGA). Al ser compatible con el estándar RISC-V (RV32I), posee la gran capacidad de ejecutar código máquina generado a partir de lenguajes de alto nivel. Además, tener el módulo de Crossbar le permite agregar y mapear nuevos periféricos de entrada/salida en memoria sin necesidad de alterar el núcleo de la CPU.

Referencias

- Harris, D. M., y Harris, S. L. (2012). Digital Design and Computer Architecture (2.ª ed.). Morgan Kaufmann.