



Manual de Usuario

**Módulos IP AXI-Slave para Sistemas en Chip basados en RISC-V
(MicroBlaze V)**

Información General

Este manual de usuario corresponde al proyecto “Diseño y validación de módulos IP AXI-Slave para un sistema en chip basado en RISC-V (MicroBlaze V)”, desarrollado por Emanuel Marín Gutiérrez (carné 2019067500) como parte del curso CE-1114 Proyecto de Aplicación de la Ingeniería en Computadores del Instituto Tecnológico de Costa Rica, específicamente de la Escuela de Ingeniería en Computadores.

El proyecto cuenta con la supervisión del Dr. Jorge Castro Godínez como supervisor principal y del Dr. Jeferson González Gómez como co-supervisor, bajo la dirección del curso a cargo del Ing. Diego Noguera Mena.

Para el desarrollo e implementación del proyecto, se utilizó la plataforma de hardware Digilent Basys 3 con FPGA Artix-7, empleando las herramientas de desarrollo AMD Vivado Design Suite 2025.1 y AMD Vitis Unified Software Platform 2025.

Todo el código fuente, documentación y recursos del proyecto están disponibles en el repositorio público: <https://github.com/MarinGE23/mbv-axi-slave-ip-repo>

El proyecto se ejecutó durante el período de Agosto a Noviembre de 2025, como requisito parcial para la obtención del grado de Licenciatura en Ingeniería en Computadores.

Descripción Breve: Este proyecto consiste en el diseño, implementación y validación de módulos IP AXI-slave compatibles con el procesador AMD MicroBlaze V (RISC-V) para la plataforma Basys 3. Los módulos desarrollados incluyen periféricos para displays de 7 segmentos, generadores de números aleatorios (PRNG/TRNG), generadores de pulsos, un perceptrón binario y un sistema integrado de autenticación segura.

Índice

| | |
|---|-----------|
| 1. Introducción | 3 |
| 1.1. Propósito del documento | 3 |
| 1.2. Audiencia objetivo | 3 |
| 1.3. Organización del documento | 3 |
| 2. Requisitos Previos | 4 |
| 2.1. Hardware requerido | 4 |
| 2.2. Software requerido | 4 |
| 2.2.1. Vivado Design Suite | 5 |
| 2.2.2. Vitis IDE | 5 |
| 2.3. Conocimientos previos recomendados | 5 |
| 2.4. Obtención del repositorio del proyecto | 6 |
| 3. Creación de Proyectos desde cero | 8 |
| 3.1. Introducción al script mbv_axi_base.tcl | 8 |
| 3.2. Proyecto AXI 7-Segment Display | 11 |
| 3.2.1. Crear proyecto Vivado (vivado-axi-7seg) | 13 |
| 3.2.2. Crear proyecto Vitis (vitis-axi-7seg) | 26 |
| 3.3. Proyecto AXI PRNG (Pseudo Random Number Generator) | 39 |
| 3.3.1. Crear proyecto Vivado (vivado-axi-prng) | 43 |
| 3.3.2. Crear proyecto Vitis (vitis-axi-prng) | 43 |
| 3.4. Proyecto AXI TRNG (True Random Number Generator) | 46 |
| 3.4.1. Crear proyecto Vivado (vivado-axi-trng) | 52 |
| 3.4.2. Crear proyecto Vitis (vitis-axi-trng) | 52 |
| 3.5. Proyecto AXI Pulse Generator | 55 |
| 3.5.1. Crear proyecto Vivado (vivado-axi-pulsegen) | 57 |
| 3.5.2. Crear proyecto Vitis (vitis-axi-pulsegen) | 58 |
| 3.6. Proyecto AXI Perceptron | 60 |
| 3.6.1. Crear proyecto Vivado (vivado-axi-perceptron) | 62 |
| 3.6.2. Crear proyecto Vitis (vitis-axi-perceptron) | 62 |
| 3.7. Proyecto AXI Secure Authentication | 66 |
| 3.7.1. Crear proyecto Vivado (vivado-axi-secure-auth) | 71 |
| 3.7.2. Crear proyecto Vitis (vitis-axi-secure-auth) | 71 |
| 4. Ejecución de Proyectos Existentes | 75 |
| 5. Solución de Problemas | 76 |
| 5.1. Errores comunes en Vivado | 76 |
| 5.2. Errores comunes en Vitis | 77 |
| 5.3. Problemas de compatibilidad | 78 |

1. Introducción

1.1. Propósito del documento

Este manual de usuario tiene como propósito guiar al lector en el proceso de creación, configuración y ejecución de proyectos basados en IP Cores AXI personalizados para sistemas embebidos en FPGA. El documento proporciona instrucciones detalladas para trabajar con cinco IP Cores individuales (AXI 7-Segment Display, AXI PRNG, AXI TRNG, AXI Pulse Generator y AXI Perceptron), así como con el proyecto integrador AXI Secure Authentication que combina múltiples componentes.

El manual está diseñado para facilitar tanto la creación de proyectos desde cero utilizando el script automatizado `mbv_axi_base.tcl`, como la apertura y ejecución de proyectos previamente configurados. Se cubren tanto el flujo de diseño de hardware en Vivado Design Suite como el desarrollo de software embebido en Vitis IDE, proporcionando una visión completa del proceso de desarrollo para sistemas basados en procesadores MicroBlaze V.

1.2. Audiencia objetivo

Este documento está dirigido a:

- **Estudiantes de ingeniería:** Estudiantes de Ingeniería Electrónica e Ingeniería en Computadores, que requieran implementar proyectos en FPGA utilizando procesadores soft-core.
- **Investigadores:** Personal académico e investigadores que trabajen en el desarrollo de sistemas embebidos, procesamiento digital de señales, criptografía en hardware o machine learning en dispositivos FPGA.
- **Desarrolladores e ingenieros:** Profesionales con experiencia básica en diseño digital que deseen implementar o evaluar los IP Cores desarrollados en aplicaciones reales.

1.3. Organización del documento

El presente manual está organizado en las siguientes secciones:

Sección 2 – Requisitos Previos: Describe los requisitos de hardware y software necesarios para trabajar con los proyectos, incluyendo las versiones específicas de las herramientas de AMD-Xilinx y los conocimientos previos recomendados.

Sección 3 – Creación de Proyectos desde cero: Proporciona instrucciones paso a paso para crear nuevos proyectos en Vivado y Vitis utilizando el script automatizado `mbv_axi_base.tcl`. Se cubre la creación de proyectos para cada uno de los cinco IP Cores individuales y el proyecto integrador de autenticación segura.

Sección 4 – Ejecución de Proyectos Existentes: Presenta la opción de abrir, compilar y ejecutar proyectos ya configurados que se encuentra en la branch develop del repositorio.

Sección 5 – Solución de Problemas: Presenta algunos errores que pueden surgir durante el flujo de diseño en Vivado y Vitis, junto con sus soluciones y recomendaciones para evitar problemas de compatibilidad.

Sección 6 – Referencias: Contiene enlaces y referencias a documentación oficial de AMD-Xilinx, especificaciones del protocolo AXI y recursos adicionales de consulta.

2. Requisitos Previos

2.1. Hardware requerido

Para la correcta ejecución de los proyectos presentados en este manual, se requiere el siguiente hardware:

- **Tarjeta de desarrollo:** Digilent Basys3 Artix-7 FPGA
 - **FPGA:** Xilinx Artix-7 XC7A35T-1CPG236C
 - **Recursos lógicos:** 33,280 celdas lógicas, 1,800 Kbits de Block RAM
 - **Interfaz de usuario:** 16 switches, 16 LEDs, 5 botones, display de 7 segmentos (4 dígitos)
 - **Conectividad:** Puerto USB para programación y comunicación UART
- **Computadora host:**
 - **Procesador:** Intel Core i5 o superior / AMD Ryzen 5 o superior (recomendado: i7/Ryzen 7)
 - **Memoria RAM:** Mínimo 4 GB (recomendado: 8 GB o más)
 - **Almacenamiento:** Mínimo 100 GB de espacio libre en disco
 - **Sistema operativo:** Windows 10/11 (64-bit) o Linux (Ubuntu 20.04/22.04)
- **Cables y accesorios:**
 - Cable USB tipo A a Micro-USB para conexión con la tarjeta Basys3
 - Fuente de alimentación USB (puerto USB 2.0 o superior)

2.2. Software requerido

El entorno de desarrollo requiere las siguientes herramientas de software de AMD-Xilinx:

2.2.1. Vivado Design Suite

Versión requerida: AMD Vivado Design Suite 2025.1

Espacio en disco requerido: Aproximadamente 60 GB

Descarga: Disponible en el sitio web oficial de AMD-Xilinx:

<https://www.xilinx.com/support/download.html>

Nota importante: Es fundamental utilizar la versión 2025.1 para garantizar la compatibilidad con los proyectos y scripts proporcionados en este manual. El script `mbv_axi_base.tcl` fue generado específicamente para esta versión y utiliza características del procesador MicroBlaze V introducidas en versiones recientes.

2.2.2. Vitis IDE

Versión requerida: AMD Vitis Unified Software Platform 2025.1

Espacio en disco requerido: Aproximadamente 40 GB (adicional a Vivado)

Descarga: Generalmente incluido en la instalación de Vivado 2025.1 o disponible por separado en:

<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis.html>

Integración: Vitis trabaja en conjunto con Vivado para el desarrollo de software embebido. Vitis importa los archivos de hardware generados por Vivado (archivos .xsa) para crear aplicaciones en C/C++ que se ejecutan en el procesador MicroBlaze V.

2.3. Conocimientos previos recomendados

Para aprovechar al máximo este manual y trabajar eficientemente con los proyectos, se recomienda que el lector posea los siguientes conocimientos y habilidades:

- **Diseño digital y arquitectura de computadoras:**
 - Conceptos fundamentales de lógica digital y sistemas digitales
 - Arquitectura de procesadores y sistemas embebidos
 - Jerarquías de memoria y organización de sistemas computacionales
- **Lenguajes de descripción de hardware (HDL):**
 - VHDL o Verilog a nivel básico-intermedio
 - Comprensión de síntesis y simulación de diseños digitales
 - Familiaridad con testbenches y verificación funcional
- **Protocolo de comunicación AXI:**
 - Arquitectura del bus AMBA AXI4 y AXI4-Lite

- Transacciones de lectura y escritura en AXI
- Concepto de memoria mapeada (memory-mapped I/O)

■ Programación en lenguaje C:

- Sintaxis y estructuras de control de C
- Manejo de punteros y direccionamiento de memoria
- Programación de bajo nivel y acceso a registros de hardware

■ Herramientas AMD-Xilinx:

- Navegación básica en el entorno Vivado Design Suite
- Conceptos de IP Integrator y Block Design
- Familiaridad con el flujo de diseño: síntesis, implementación y generación de bitstream
- Uso básico de Vitis para desarrollo de aplicaciones embebidas

■ Sistemas embebidos y procesadores soft-core:

- Concepto de procesador soft-core (MicroBlaze/MicroBlaze V)
- Diferencias entre arquitecturas Harvard y von Neumann
- Interfaz UART/JTAG para depuración

■ Conocimientos adicionales útiles:

- Scripts en Tcl (Tool Command Language) para automatización en Vivado
- Lectura e interpretación de diagramas de bloques de sistemas digitales
- Uso de terminal/consola y comandos básicos del sistema operativo

Nota: Aunque estos conocimientos son recomendados, el manual está diseñado para ser lo suficientemente detallado como para permitir a usuarios con diferentes niveles de experiencia completar los proyectos exitosamente. Se proporcionan explicaciones paso a paso y referencias adicionales cuando es necesario.

2.4. Obtención del repositorio del proyecto

Para comenzar a trabajar con el proyecto, primero debe descargar el repositorio que contiene todos los archivos necesarios: IP Cores, scripts TCL, código de aplicaciones para Vitis y archivos de constraints.

Descarga del repositorio

El repositorio está disponible públicamente en GitHub. Se recomienda seguir estos pasos:

Requisito previo: Tener Git instalado en su sistema.

Pasos para clonar el repositorio:

1. Abra una terminal o línea de comandos.
2. Navegue al directorio donde desea guardar el proyecto, por ejemplo:

```
cd C:/Users/[usuario]/Desktop
```

3. Ejecute el comando de clonación:

```
git clone https://github.com/MarinGE23/mbv-axi-slave-ip-repo.git
```

4. Acceda al directorio del proyecto:

```
cd mbv-axi-slave-ip-repo
```

Recomendaciones importantes:

- **Evite rutas con espacios o caracteres especiales** (como acentos, eñes, símbolos).
- **Ejemplos de rutas seguras:**
 - Windows: C:\Users\[usuario]\Desktop\mbv-axi-slave-ip-repo
 - Linux/macOS: /home/[usuario]/projects/mbv-axi-slave-ip-repo

Nota importante: Descargar y extraer el ZIP del repositorio no es recomendable para este proyecto debido a que el archivo ZIP no incluye los submódulos y dependencias esenciales del repositorio.

Estructura del repositorio

Una vez descargado usando git clone, el repositorio contiene la siguiente estructura de directorios:

- **ip_repo/**: Contiene los 5 IP Cores desarrollados
 - axi_7seg_1_0/: IP Core de display de 7 segmentos
 - axi_prng_1_0/: Generador pseudo-aleatorio XorShift32
 - axi_trng_1_0/: Generador true random con ring oscillators
 - axi_pulsegen_1_0/: Generador de pulsos configurables
 - axi_perceptron_1_0/: Perceptrón binario con entrenamiento
- **tcl_script/**: Contiene el script base de creación de proyectos
 - mbv_axi_base.tcl: Script principal para generar diseños con el procesador MicroBlaze V

- **vitis_code/**: Código fuente de aplicaciones de ejemplo
 - main_7seg_1.c, main_7seg_2.c: Demos del display
 - main_prng.c: Demo del generador pseudo-aleatorio
 - main_trng.c: Demo del generador true random
 - main_pulsegen.c: Demo del generador de pulsos
 - main_perceptron.c: Demo de entrenamiento del perceptrón
 - main_secure_auth.c: Sistema completo de autenticación
- **vivado_constraints/**: Archivos XDC de constraints físicos
 - 7seg.xdc: Constraints para display de 7 segmentos
 - prng.xdc, trng.xdc: Constraints para generadores
 - pulsegen.xdc: Constraints para salida de pulsos
 - perceptron.xdc: Constraints para entradas/salida del perceptrón
 - secure_auth.xdc: Constraints para el sistema integrado

3. Creación de Proyectos desde cero

3.1. Introducción al script mbv_axi_base.tcl

El script `mbv_axi_base.tcl` es un script Tcl automatizado generado por Vivado 2025.1 que permite recrear de manera reproducible un proyecto de hardware base para sistemas embebidos en FPGA. Este script sirve como punto de partida para todos los proyectos de IP Cores presentados en este manual, proporcionando una infraestructura común de procesador, memoria y comunicación.

Propósito del script

El script recrea un proyecto de Vivado configurado para la tarjeta de desarrollo Digilent Basys3 con un sistema embebido completo basado en el procesador MicroBlaze V. Este sistema base permite la integración de IP Cores personalizados a través del bus AXI4-Lite, facilitando el desarrollo de aplicaciones de hardware/software co-diseñadas.

Arquitectura del sistema

El sistema embebido implementado por el script comprende los siguientes componentes principales:

- **MicroBlaze RISC-V (`microblaze_riscv_0`)**: Procesador soft-core de 32 bits con arquitectura RISC-V que actúa como unidad central de procesamiento del sistema. Configurado con:

- Capacidades de depuración habilitadas (C_DEBUG_ENABLED=1)
 - Interfaz AXI4-Lite para acceso a periféricos (C_D_AXI=1)
 - Buses locales de memoria para instrucciones y datos (C_I_LMB=1, C_D_LMB=1)
- **Memoria Local (Local Memory Bus - LMB):** Sistema de memoria de baja latencia implementado mediante Block RAM (BRAM) dual-port. Proporciona 32 KB de memoria compartida para almacenar tanto instrucciones como datos del programa ejecutable.
 - **Clock Wizard (clk_wiz_0):** Generador y administrador de reloj que toma como entrada el reloj de sistema de 100 MHz de la tarjeta Basys3 y produce las señales de reloj necesarias para el funcionamiento del procesador y periféricos.
 - **Processor System Reset (rst_clk_wiz_0_100M):** Módulo de gestión de señales de reset que garantiza la secuencia correcta de inicialización del sistema, sincronizando los resets con las señales de reloj estables.
 - **MicroBlazeV Debug Module - MDM (mdm_1):** Interfaz de depuración que permite la comunicación UART/JTAG entre la computadora host y el procesador MicroBlaze V para programación, depuración y control del sistema.
 - **AXI UART Lite (axi_uartlite_0):** Controlador de comunicación serial conectado al puerto USB de la tarjeta Basys3. Permite la transmisión y recepción de datos entre el sistema embebido y una terminal en la computadora host a través de la interfaz usb_uart.
 - **AXI SmartConnect (axi_smc):** Interconector inteligente AXI4-Lite que actúa como bus de comunicación entre el procesador MicroBlaze V y los periféricos. Facilita las transacciones de lectura/escritura hacia direcciones de memoria mapeadas, permitiendo la expansión del sistema con IP Cores adicionales.

Mapa de memoria

El sistema define el siguiente espacio de direcciones:

- **0x00000000 - 0x00007FFF:** Memoria local (BRAM) - 32 KB para código y datos
- **0x40600000 - 0x4060FFFF:** AXI UART Lite - Registros de comunicación serial (64 KB)
- **Direcciones adicionales:** Reservadas para IP Cores personalizados (asignadas durante la configuración específica de cada proyecto)

Configuración del proyecto

El script configura automáticamente:

- **Flujos de síntesis e implementación:** Define las estrategias de optimización “Vivado Synthesis Defaults” y “Vivado Implementation Defaults” con sus respectivos reportes de análisis.
- **Reportes de análisis:** Genera múltiples reportes durante el proceso de diseño, incluyendo:
 - Utilización de recursos (LUTs, FFs, BRAM, DSP)
 - Análisis de timing (setup, hold, clock skew)
 - Design Rule Check (DRC) y Methodology Check
 - Análisis de consumo de potencia
 - Estado de ruteo y control sets

Uso del script

Para crear un nuevo proyecto utilizando este script, se debe ejecutar el siguiente comando en la consola Tcl de Vivado:

```
source <ruta>/mbv_axi_base.tcl
```

El script acepta parámetros opcionales para personalizar la creación del proyecto:

- **-origin_dir <ruta>:** Define el directorio de origen para las rutas relativas de archivos
- **-project_name <name>:** Especifica un nombre personalizado para el proyecto (por defecto: project_1)
- **-help:** Muestra la información de ayuda del script

Ejemplo de uso en el Tcl Console de Vivado:

```
cd C:/Users/usuario/<ruta del tcl_script>/
set ::user_project_name "mi_proyecto"
set ::origin_dir_loc "<ruta relativa o completa del proyecto>"
source mbv_axi_base.tcl
```

Extensibilidad

El diseño base creado por este script está preparado para ser extendido mediante la adición de IP Cores personalizados al Block Design. Los proyectos presentados en las siguientes secciones utilizan este sistema base y agregan componentes específicos (AXI 7-Segment Display, AXI PRNG, AXI TRNG, etc.) conectándolos al bus AXI SmartConnect y asignándoles direcciones de memoria únicas en el espacio de direccionamiento del procesador.

En esencia, `mbv_axi_base.tcl` proporciona un sistema de procesamiento embebido minimalista, completamente funcional y listo para:

- Desarrollo de software embebido en lenguaje C mediante Vitis IDE
- Comunicación con la computadora host a través de UART/USB
- Integración de periféricos personalizados mediante el bus AXI4-Lite
- Depuración en tiempo real mediante interfaz UART/JTAG

3.2. Proyecto AXI 7-Segment Display

El proyecto AXI 7-Segment Display implementa un controlador de hardware para el display de 7 segmentos de 4 dígitos integrado en la tarjeta Basys3. Este IP Core personalizado permite al procesador MicroBlaze V controlar el display mediante operaciones de escritura en registros mapeados en memoria, facilitando la visualización de números decimales, contadores y valores de depuración en aplicaciones embebidas.

Descripción del IP Core

El axi_7seg_1_0 es un IP Core desarrollado en Verilog que implementa una interfaz AXI4-Lite Slave y se conecta directamente a las señales físicas del display de 7 segmentos de la tarjeta Basys3. El módulo incorpora las siguientes características principales:

- **Interfaz AXI4-Lite:** Permite la comunicación con el procesador mediante el bus AXI utilizando transacciones estándar de lectura y escritura de 32 bits.
- **Decodificación automática:** Convierte valores numéricos (0-9) en patrones de 7 segmentos de forma transparente, eliminando la necesidad de que el software realice la conversión.
- **Multiplexación por hardware:** Implementa control de multiplexación temporal para los 4 dígitos del display, garantizando una visualización estable sin intervención del procesador.
- **Control independiente de dígitos:** Permite actualizar individualmente cada uno de los 4 dígitos o establecer los 4 simultáneamente en una sola operación.
- **Señales activas en bajo:** Compatible con la lógica activa-baja de los ánodos y segmentos del display de la Basys3.

Arquitectura del módulo

El IP Core está compuesto por dos módulos principales:

1. Módulo AXI4-Lite Slave (axi_7seg_slave_lite_v1_0_S00_AXI):

- Implementa el protocolo de comunicación AXI4-Lite completo (canales de escritura, lectura y respuesta)
- Gestiona 4 registros internos de 32 bits (slv_reg0 a slv_reg3)
- Maneja las señales de handshaking (AWVALID, AWREADY, WVALID, WREADY, etc.)

- Espacio de direccionamiento de 4 bits (16 direcciones)

2. Módulo Display Decoder (display_decoder):

- Decodifica valores decimales (0-9) a patrones de 7 segmentos
- Implementa lógica de multiplexación para los 4 dígitos
- Genera señales de control de ánodos (an_out[3:0]) y segmentos (seg_out[7:0])
- Formato de segmentos: DP-G-F-E-D-C-B-A (donde DP es el punto decimal, no utilizado)

Mapa de registros

El IP Core utiliza dos registros principales para su operación:

| Offset | Nombre | Descripción |
|--------|--------------|--|
| 0x00 | ANODE_CTRL | Control de ánodos (habilita/deshabilita dígitos) |
| 0x04 | DIGIT_VALUES | Valores de los 4 dígitos (8 bits por dígito) |

Registro ANODE_CTRL (offset 0x00):

- Bits [3:0]: Control de habilitación de los 4 dígitos (activo en bajo)
- Valor típico: 0x0F (todos los dígitos habilitados)

Registro DIGIT_VALUES (offset 0x04):

- Bits [7:0]: Valor del dígito 0 (dígito más a la derecha)
- Bits [15:8]: Valor del dígito 1
- Bits [23:16]: Valor del dígito 2
- Bits [31:24]: Valor del dígito 3 (dígito más a la izquierda)
- Rango válido por dígito: 0-9

Driver de software

El IP Core incluye un driver en lenguaje C que simplifica el desarrollo de aplicaciones. Las funciones principales son:

- Seg7_Initialize(): Inicializa el driver y habilita todos los dígitos
- Seg7_SetDigit(): Establece el valor de un dígito específico (0-3)
- Seg7_SetAllDigits(): Establece los valores de los 4 dígitos simultáneamente
- Seg7_SetNumber(): Muestra un número decimal (0-9999) en el display
- Seg7_ClearDisplay(): Limpia el display (establece todos los dígitos en 0)
- Seg7_Reset(): Reinicia el display a su estado por defecto

Aplicaciones típicas

Este IP Core es útil en diversas aplicaciones de sistemas embebidos:

- **Contadores y temporizadores:** Visualización de valores incrementales o tiempo transcurrido
- **Depuración de hardware y Monitores de sistema:** Mostrar valores de registros, estados o códigos de error. Visualización de métricas como frecuencia, temperatura o voltaje
- **Interfaces de usuario:** Presentar resultados de cálculos o lecturas de sensores
- **Aplicaciones educativas:** Demostración de conceptos de multiplexación y decodificación digital

Las siguientes subsecciones detallan el proceso completo de creación de un proyecto en Vivado y Vitis utilizando este IP Core, desde la configuración inicial hasta la ejecución de aplicaciones de prueba en hardware real.

3.2.1. Crear proyecto Vivado (vivado-axi-7seg)

Paso 1: Ejecutar el `mbv_axi_base.tcl`

Abrir Vivado y ejecutar los siguientes comandos y presionar Enter en el Tcl Console (ver Figura 1).

```
cd "C:/Users/<usuario>/Desktop/mbv-axi-slave-ip-repo/tcl_script/"
set ::user_project_name "vivado-axi-7seg"
set ::origin_dir_loc "C:/Users/<usuario>/Desktop/mbv-axi-slave-ip-repo/"
source mbv_axi_base.tcl
```

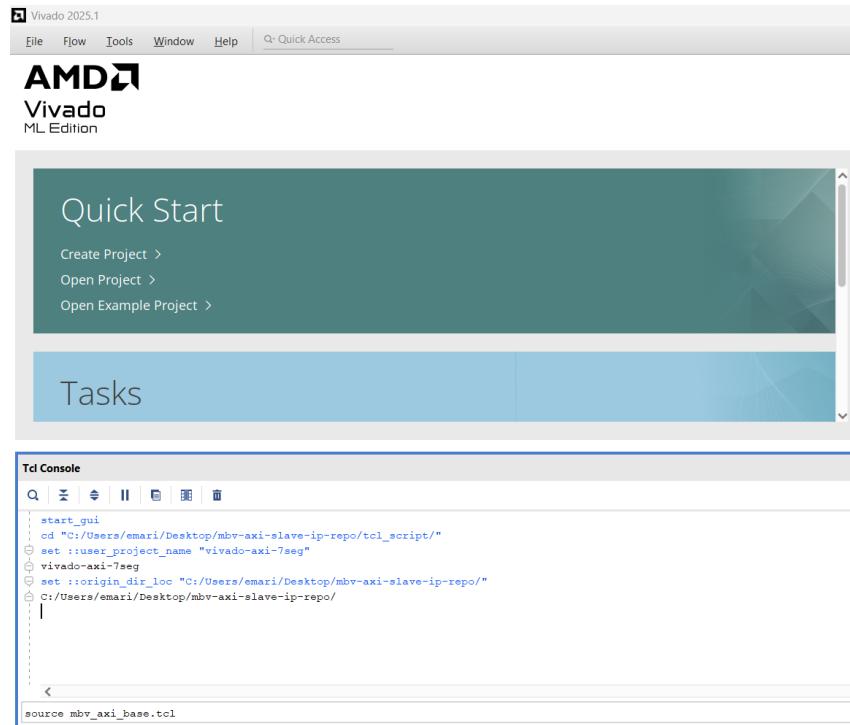


Figura 1: Crear proyecto vivado-axi-7seg

Una vez creado el proyecto, dar click en Open Block Design para ver el diseño generado por el `mbv_axi_base.tcl`. Figura 2 y 3.

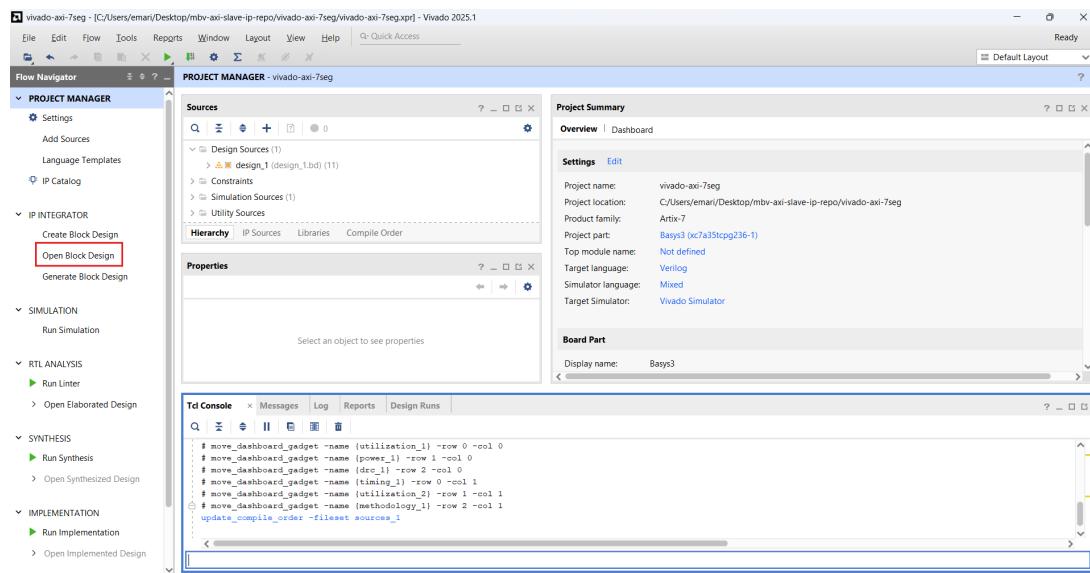


Figura 2: Click en Open Block Design

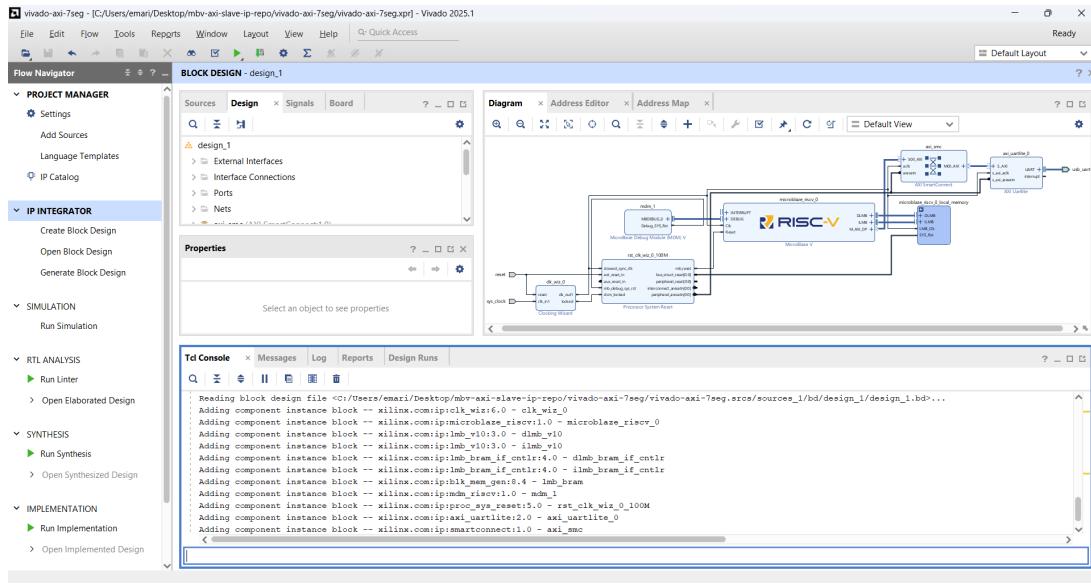


Figura 3: Vista Block Design

En la Figura 4, se puede observar el diseño generado con mayor detalle.

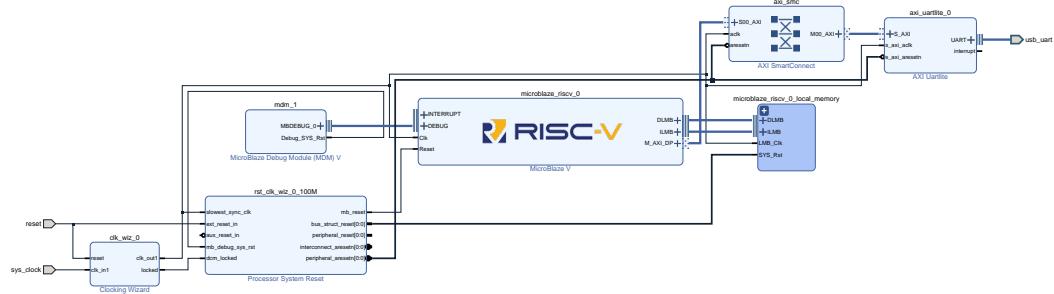


Figura 4: Block Design generado por el mbv_axi_base.tcl

Paso 2: Agregar el repositorio con los IP Cores desarrollados

Para empezar hay que seleccionar Tools/Settings... (Figura 5), luego seleccionar IP/Repository y dar click en el botón + (Figura 6), en la venta IP Repositories se debe seleccionar la carpeta ip_repo y presionar Select (Figura 7). En la ventana Add Repository se puede observar que el repositorio fue agregado al proyecto y que el mismo contiene los cinco IP Cores desarrollados, dar click en OK (Figura 8). Finalmente, se debe dar click en Apply y OK para guardar la configuración (Figura 9).

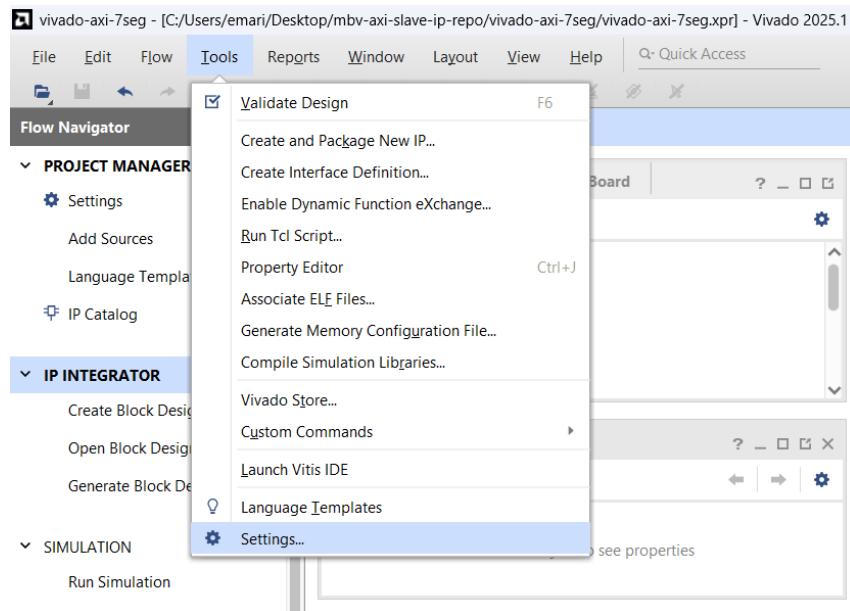


Figura 5: Click en Tools/Settings...

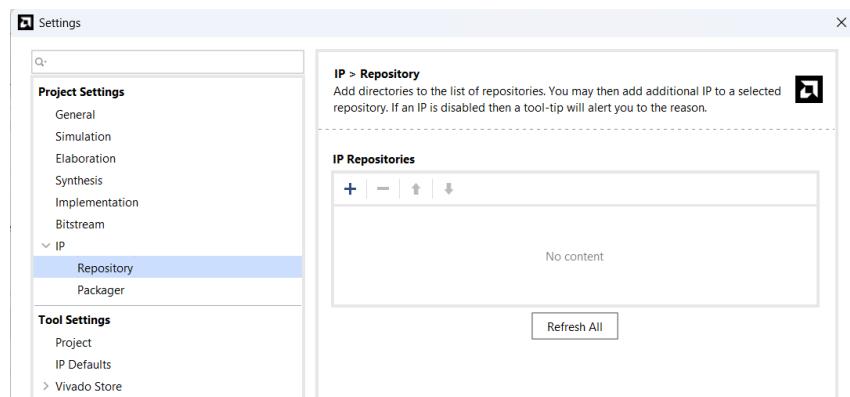


Figura 6: Click en IP/Repository/+

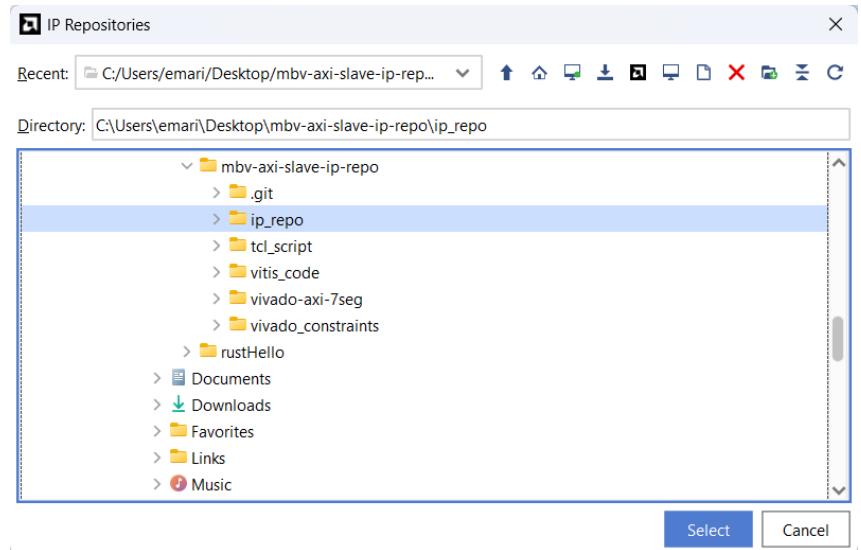


Figura 7: Seleccionar carpeta ip_repo

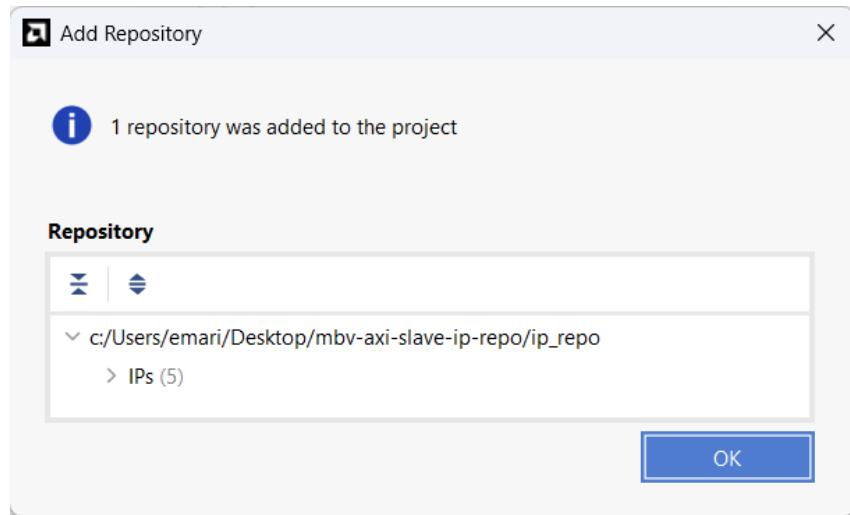


Figura 8: Repositorio agregado al proyecto

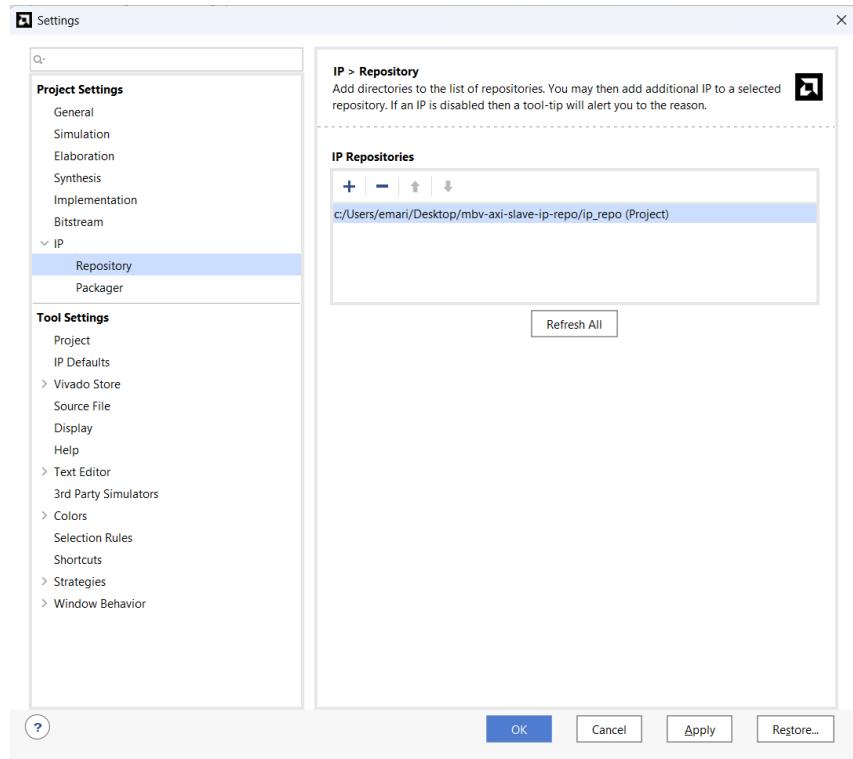


Figura 9: Guardar configuración de Tools/Settings...

Paso 3: Agregar y conectar el IP axi_7seg_v1.0

Para empezar hay que dar click en el botón + del Tab Diagram y escribir en el Search el nombre del módulo: `axi_7seg_v1.0`, al encontrarlo dar doble click izquierdo para seleccionarlo (ver Figura 10). Una vez agregado el módulo, dar click en Run Connection Automation (Figura 11) y presionar OK en la ventana con el mismo nombre (Figura 12).

El módulo tiene dos puertos de salida (`an_out[3:0]` y `seg_out[7:0]`), hay que seleccionar cada uno y con click derecho escoger la opción Make External (Figura 13).

En la Figura 14, se puede dar click en el botón marcado en rojo de la derecha para reorganizar el layout del diagrama y se puede dar click en el botón marcado en rojo de la izquierda para validar el diseño actual.

En la Figura 15 se observa el diseño actual con más detalle.

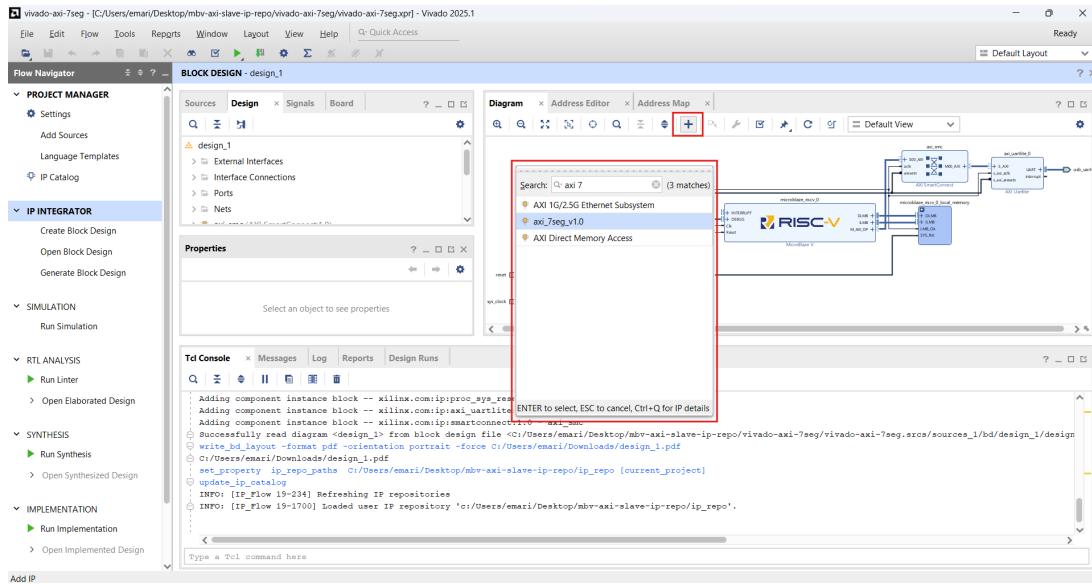


Figura 10: Agregar módulo axi_7seg_v1.0

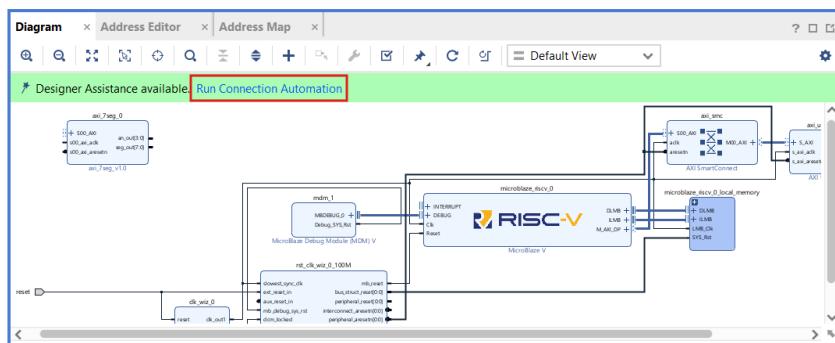


Figura 11: Click en Run Connection Automation

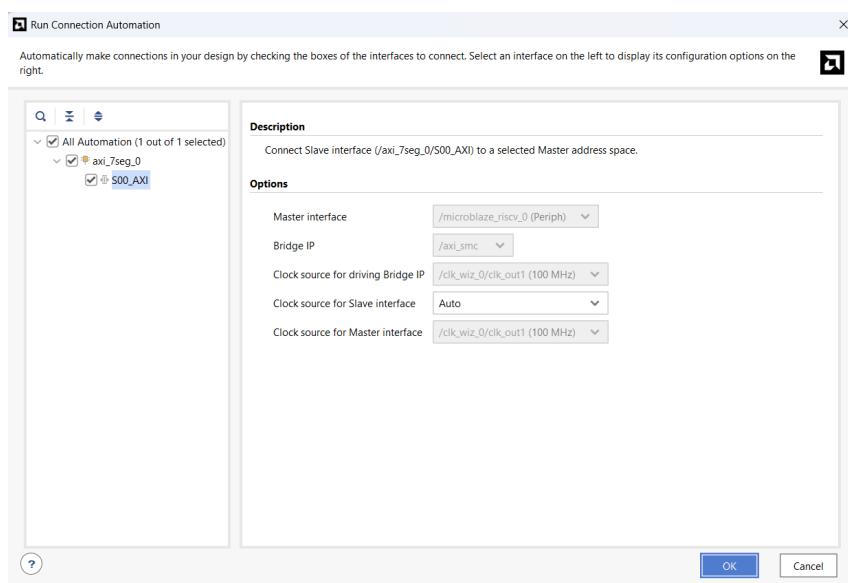


Figura 12: Click OK en la ventana Run Connection Automation

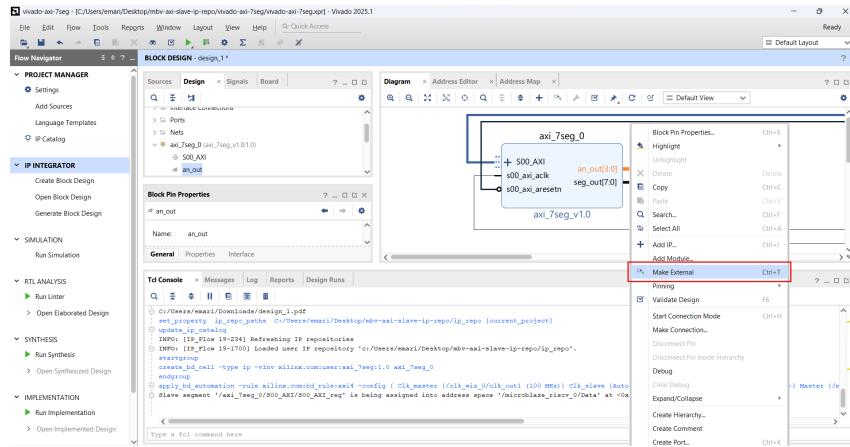


Figura 13: Click en la opción Make External

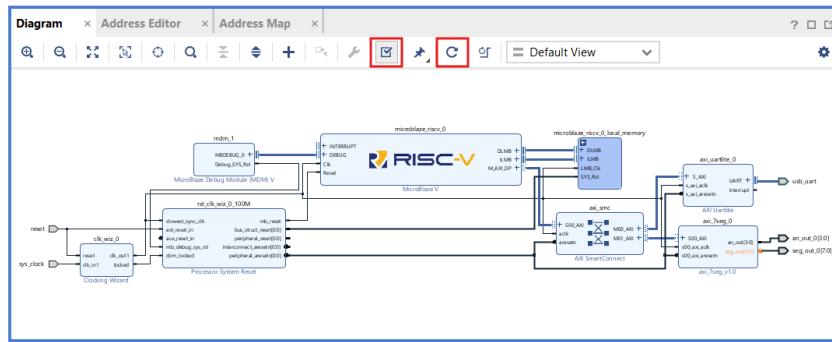


Figura 14: Click en Regenerate Layout y Validate Design

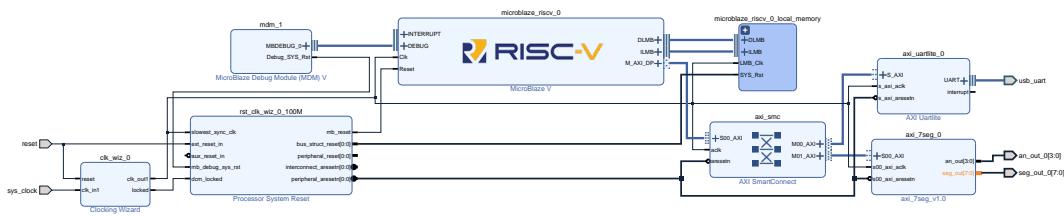


Figura 15: Block Design después de agregar el módulo axi_7seg_v1.0

Paso 4: Agregar constraints al proyecto

De acuerdo con la Figura 16, se debe dar click derecho en la carpeta Constraints y seleccionar Add Sources..., en la ventana Add Sources dar click en Next (Figura 17) y luego click en Add Files (Figura 18).

En la ventana Add Constraint Files se debe buscar la carpeta vivado_constraints dentro de mbv-axi-slave-ip-repo, seleccionar el archivo 7seg.xdc y presionar OK (Figura 19) y finalmente dar click en Finish.

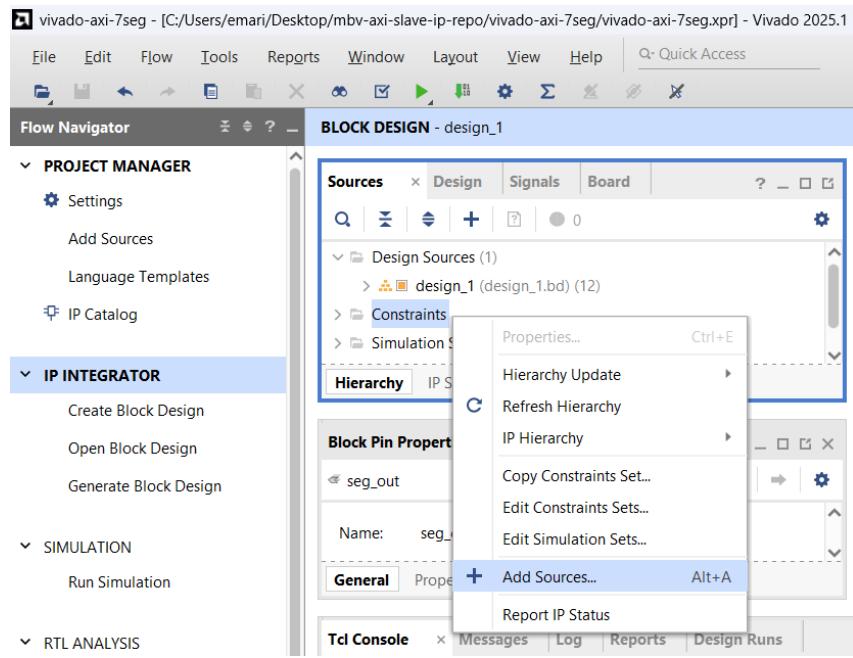


Figura 16: Click derecho en la carpeta Constraints y seleccionar Add Sources...

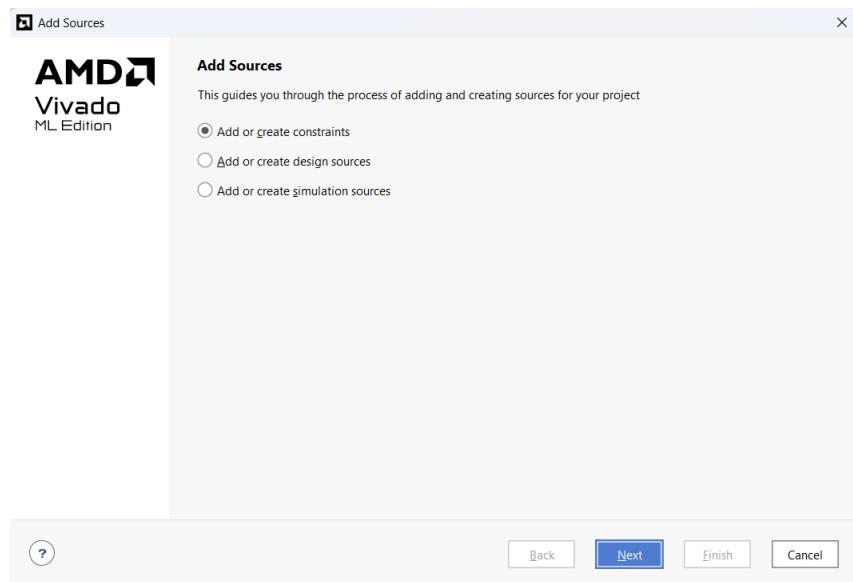


Figura 17: Click Next en la ventana Add Sources

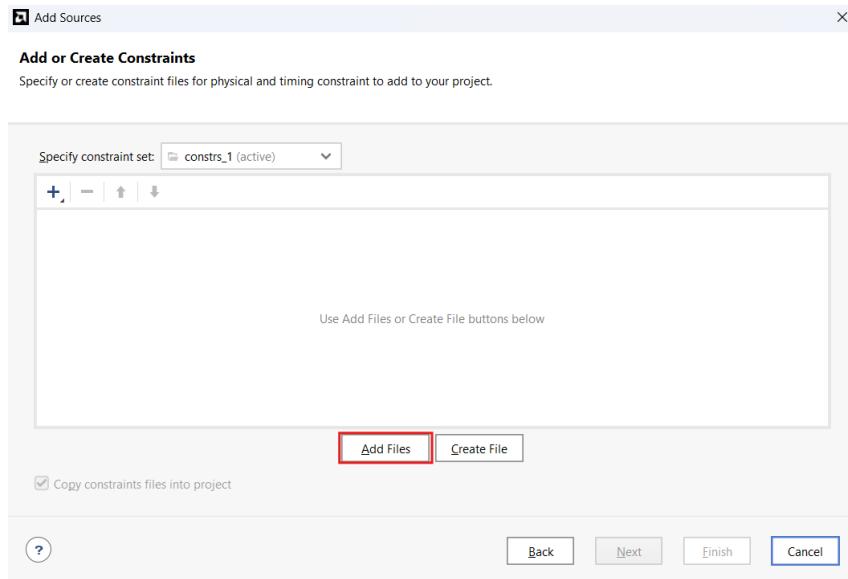


Figura 18: Click Add Files en la ventana Add Sources



Figura 19: Seleccionar el archivo 7seg .xdc

Paso 5: Crear HDL Wrapper y generar bitstream

De acuerdo con la Figura 20, se debe dar click derecho en el archivo `design_1` y seleccionar `Create HDL Wrapper...`, luego dar click en `OK` en la ventana `Create HDL Wrapper` (Figura 21).

Una vez que el Wrapper esté listo, dar click en `Generate Bitstream` (Figura 22), luego dar click en `Yes` en la ventana `No Implementation Results Available` (Figura 23) y click `OK` en la ventana `Launch Runs` (Figura 24).

La generación del bitstream puede tardar varios minutos de acuerdo a la capacidad de procesamiento de la computadora host, al terminar el proceso aparecerá la siguiente ventana (ver Figura 25), dar click en `Cancel`.

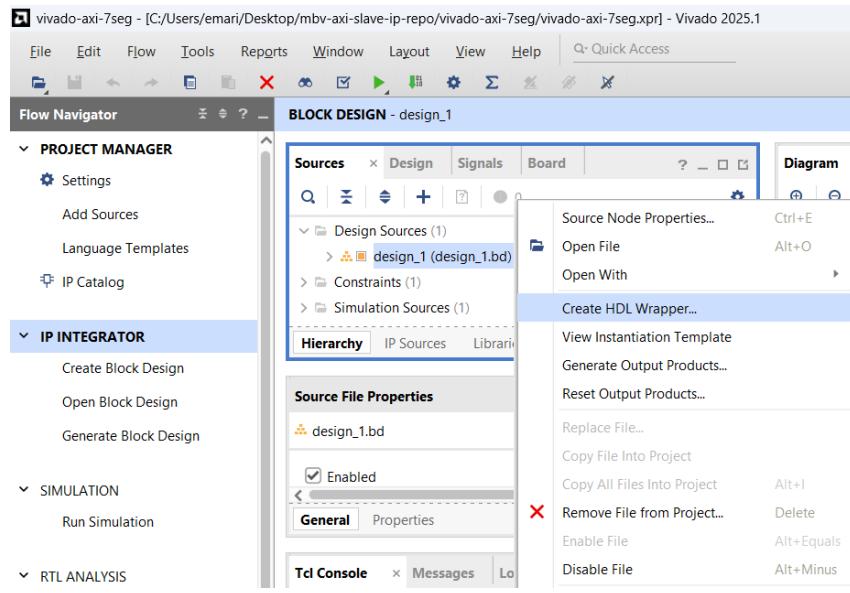


Figura 20: Click derecho en el archivo design_1 y seleccionar Create HDL Wrapper...

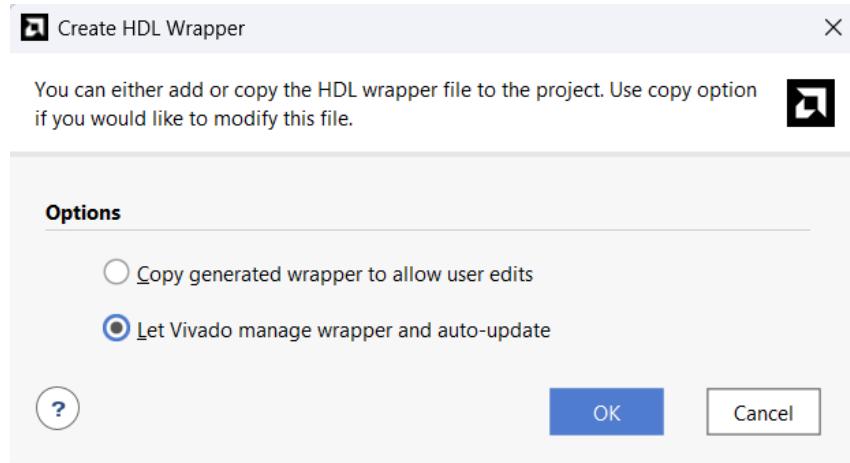


Figura 21: Click OK en la ventana Create HDL Wrapper

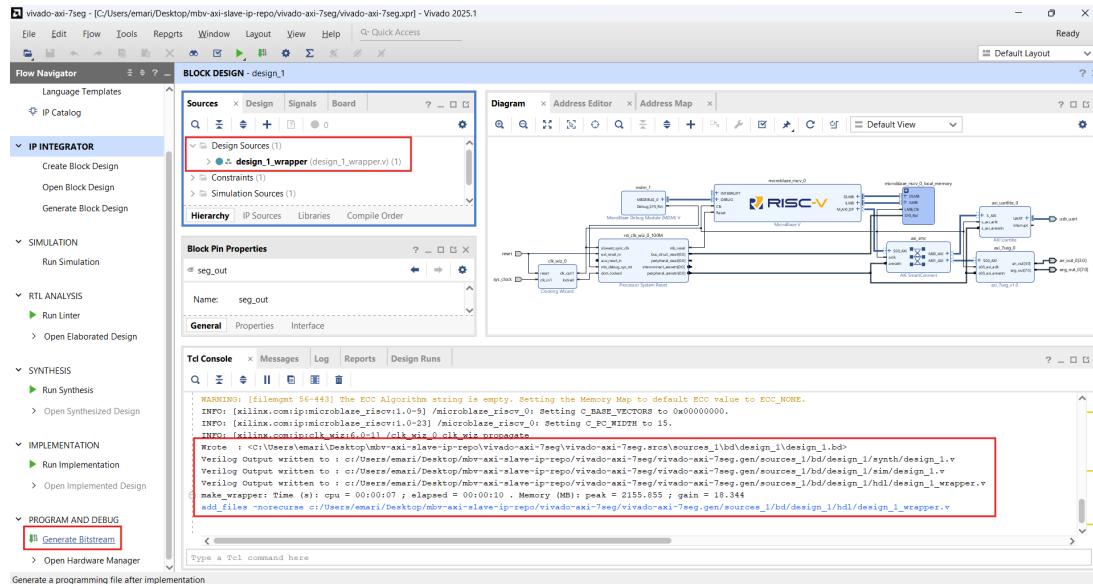


Figura 22: Click en Generate Bitstream

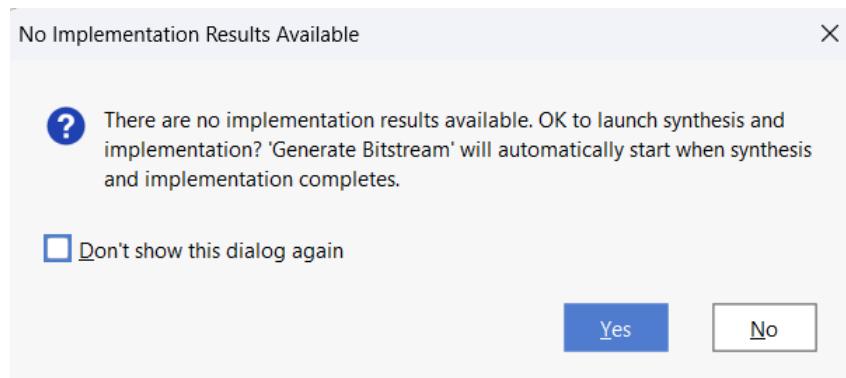


Figura 23: Click Yes en la ventana No Implementation Results Available

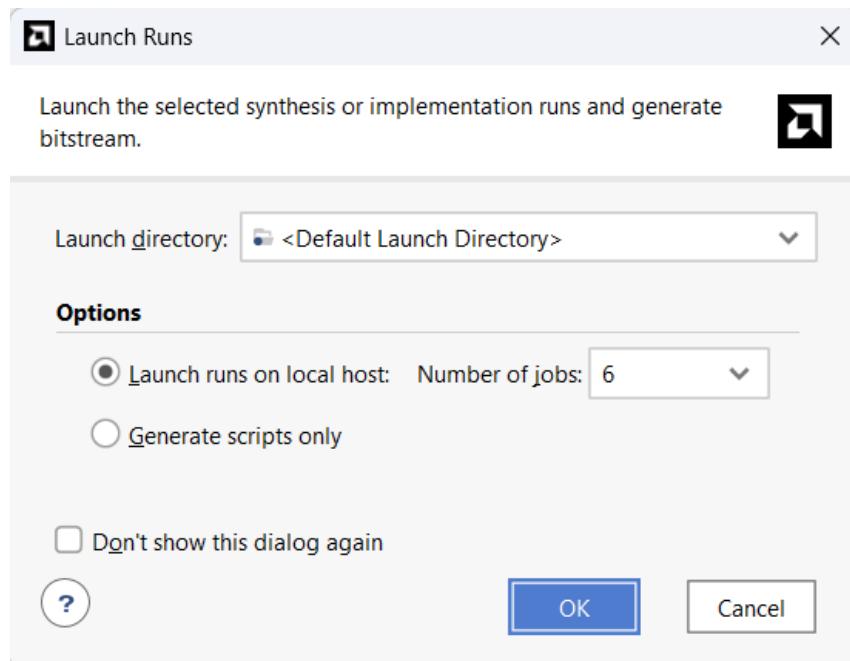


Figura 24: Click OK en la ventana Launch Runs

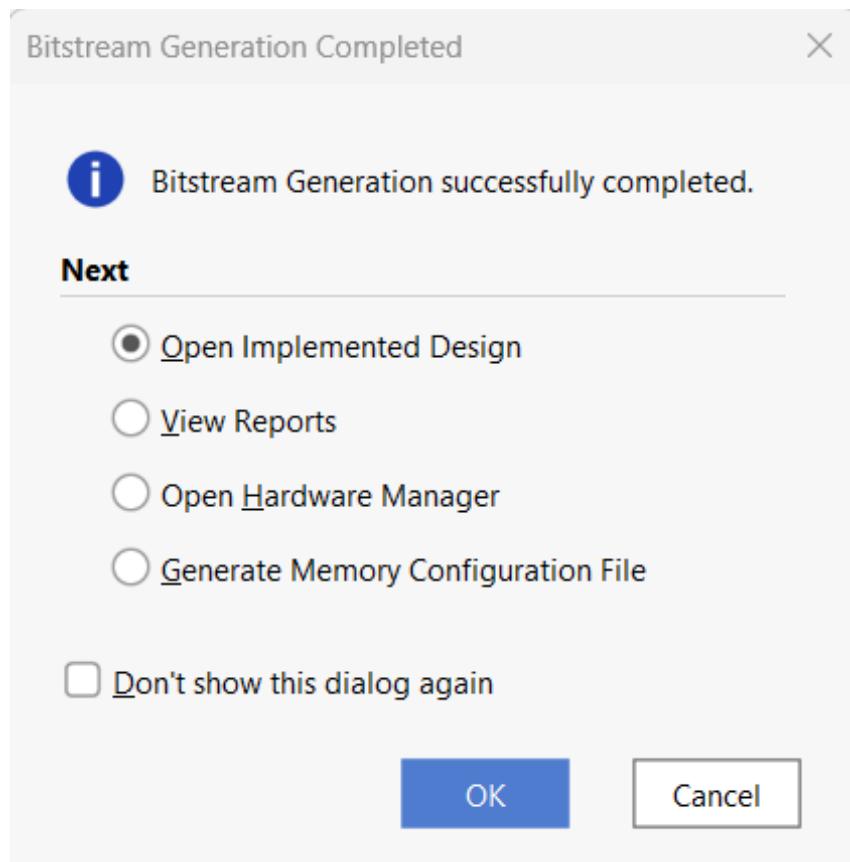


Figura 25: Click Cancel en la ventana Bitstream Generation Completed

3.2.2. Crear proyecto Vitis (vitis-axi-7seg)

Paso 1: Exportar hardware del proyecto vivado-axi-7seg

Antes de crear el proyecto en Vitis, es necesario exportar el hardware del proyecto en Vivado tras la generación del bitstream. Para ello se debe seleccionar **Files/Export/Export Hardware . . .** (ver Figura 26).

Cuando aparezca la ventana **Export Hardware Platform**, dar click en **Next** (Figura 27), luego seleccionar la opción **Include bitstream/binary** (Figura 28), dar click en **Next** dos veces más y finalmente dar click en **Finish**. Esto generará el archivo **design_1_wrapper.xsa**.

Nota: Un archivo .xsa en Vivado es un archivo contenedor (Xilinx Support Archive) que almacena toda la información de hardware de una plataforma para un dispositivo específico. Contiene detalles de la configuración, como la del reloj, la memoria, los periféricos conectados, la definición de los pines físicos y los datos de programación de la FPGA. Este archivo es crucial para el desarrollo de software y se utiliza en herramientas como Vitis para crear y enlazar aplicaciones a la plataforma de hardware.

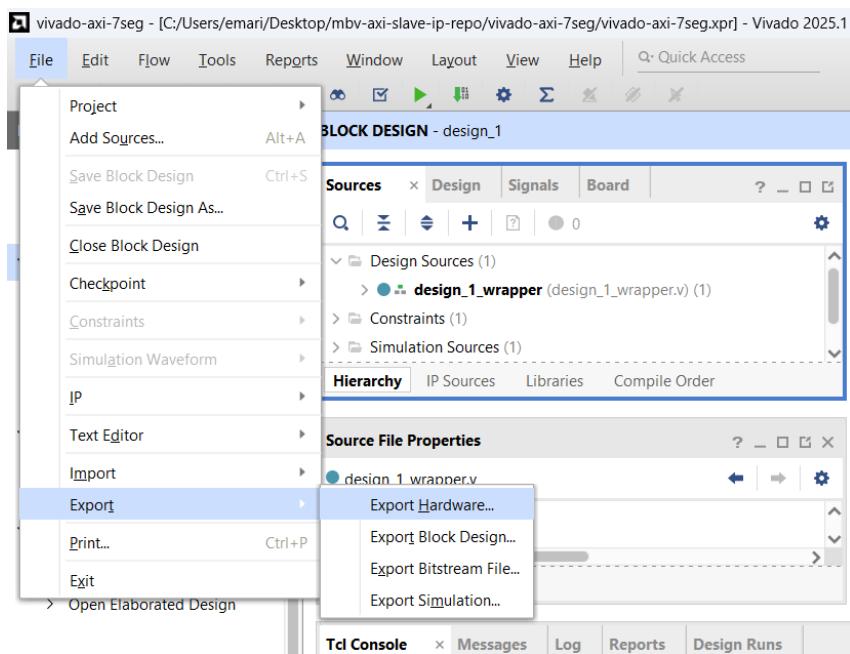


Figura 26: Exportar hardware del proyecto de Vivado

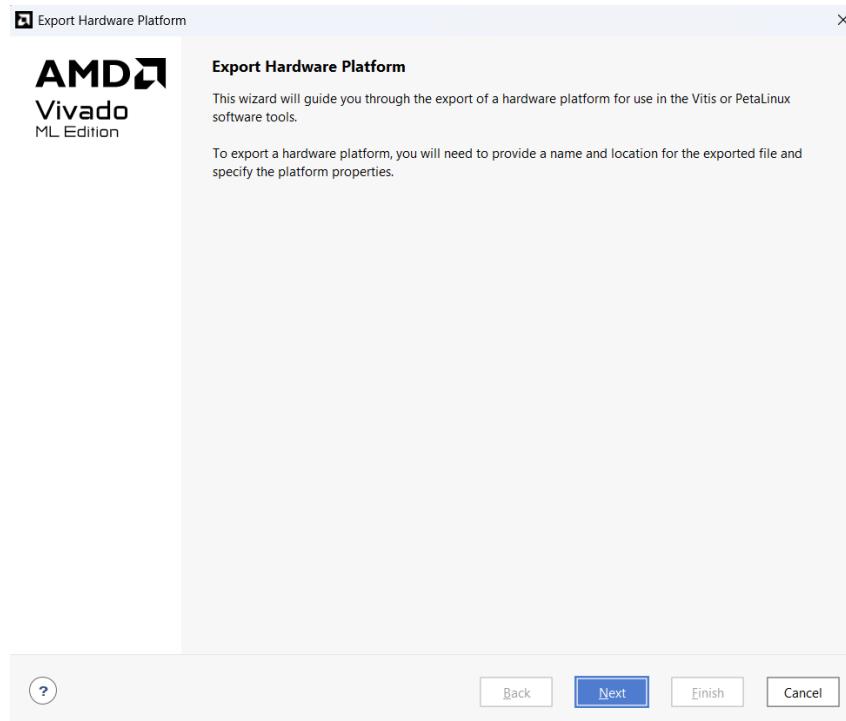


Figura 27: Click Next en la ventana Export Hardware Platform

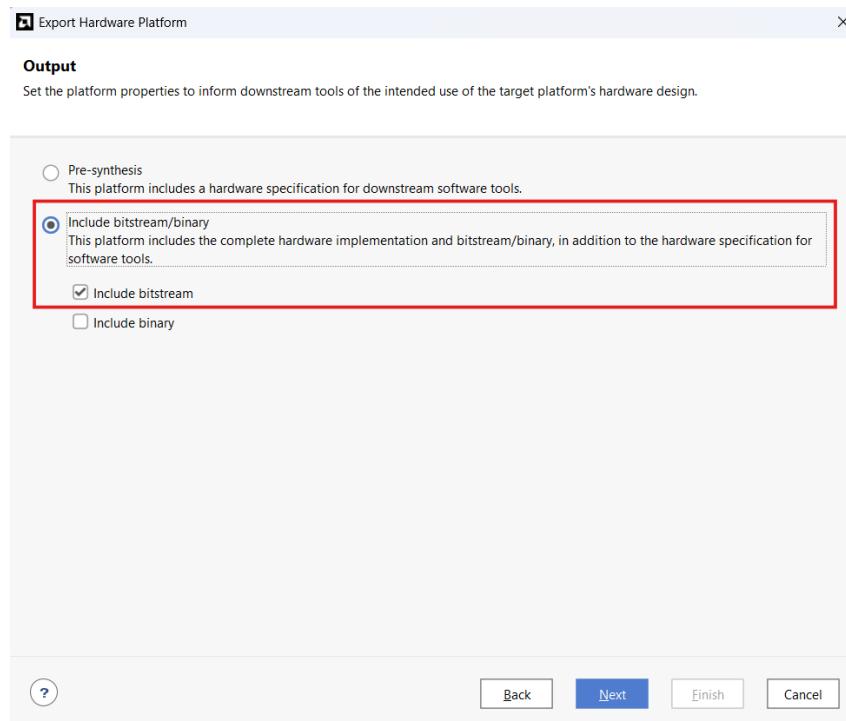


Figura 28: Seleccionar Include bitstream/binary en la ventana Export Hadware Platform

Paso 2: Crear plataforma de desarrollo embebido

Abrir Vitis y presionar Set Workspace, en mbv-axi-slave-ip-repo crear un nuevo folder llamado vitis-axi-7seg y seleccionarlo (Figura 29 y 30).

Luego dar click en Create Platform Component (Figura 31). En la ventana Create Platform Component, dar click en Next, luego dar click en Browse y seleccionar el archivo `design_1_wrapper.xsa` generado en el paso anterior y dar click en Open(Figura 32). Dar click en Next dos veces más y finalmente dar click en Finish.

Una vez que la plataforma de desarrollo embebido está creada, se debe dar click en Build (Figura 33) y cuando este termine, así se debe mostrar (Figura 34).

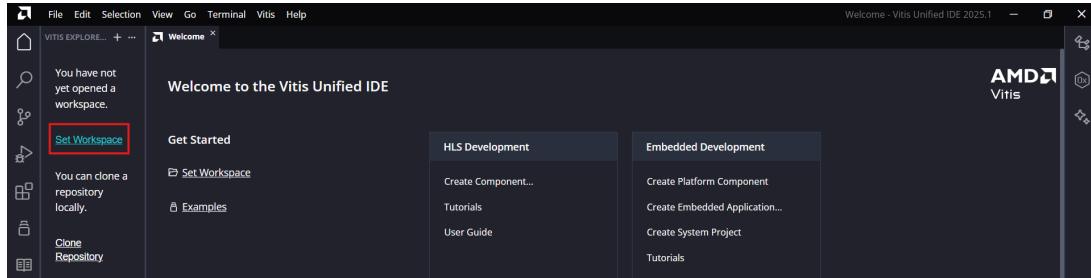


Figura 29: Configurar espacio de trabajo

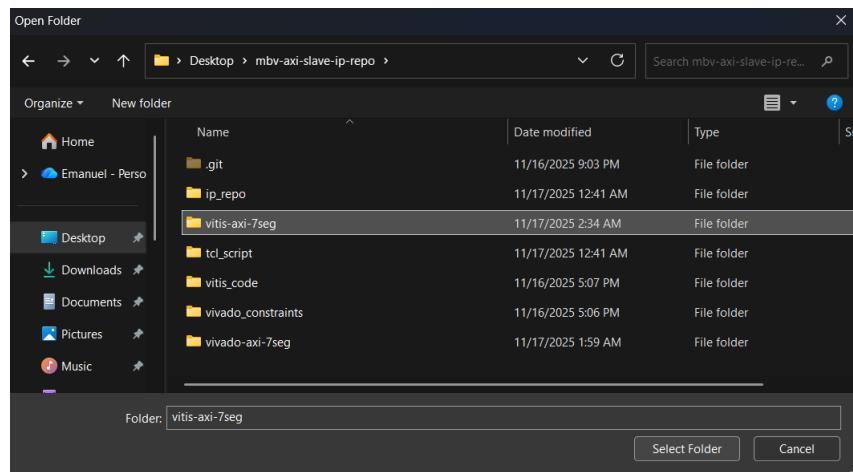


Figura 30: Seleccionar espacio de trabajo

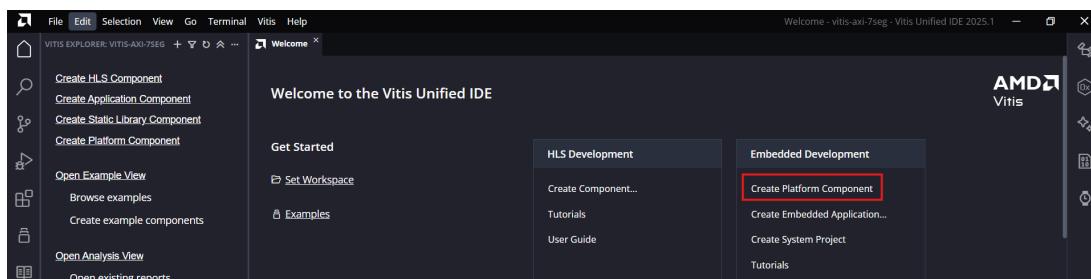


Figura 31: Seleccionar Create Platform Component

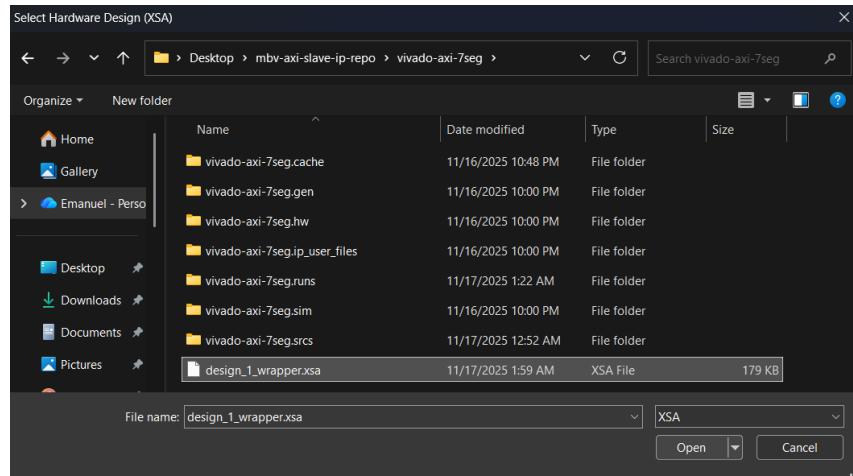


Figura 32: Seleccionar el archivo design_1_wrapper.xsa y presionar Open

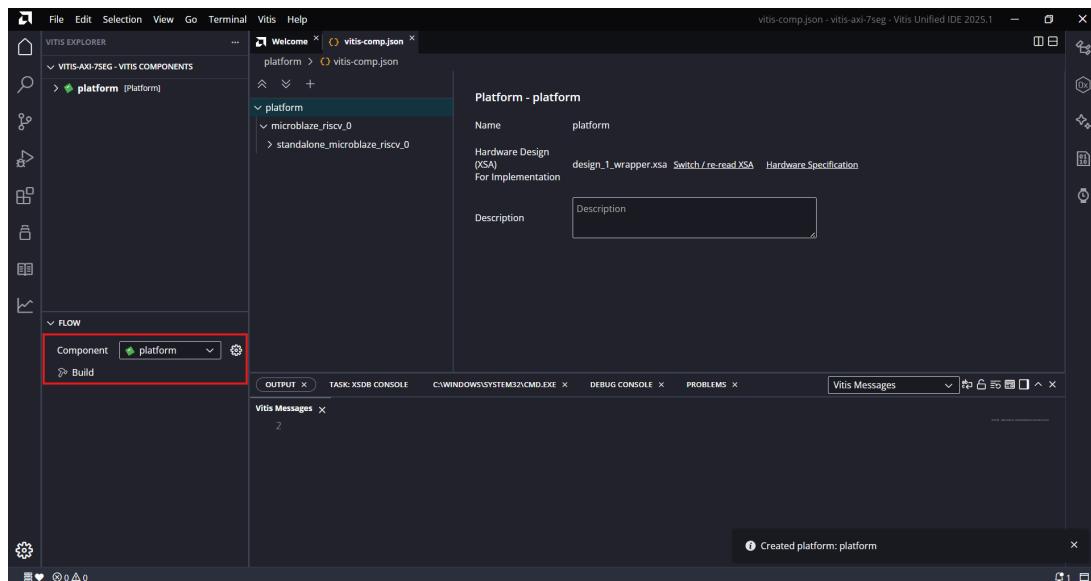


Figura 33: Click en Build

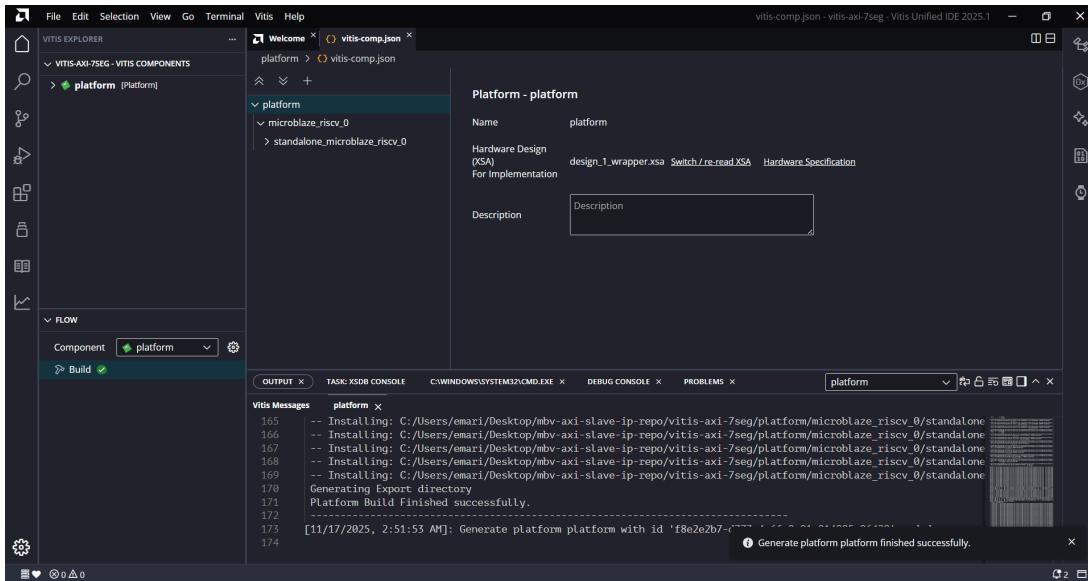


Figura 34: Build finalizado

Paso 3: Crear aplicación a partir de plantilla Hello World

Click en Examples y luego en Hello World (Figura 35). Click en Create Application Component from Template (Figura 36). En Component name poner 7seg_test_app y dar click en Next (Figura 37). Asegurarse de seleccionar la plataforma creada en el paso anterior (Figura 38). Por último dar click en Next y Finish.

Una vez que la aplicación está creada, se debe dar click en Build (Figura 39) y dar click en Save in Workspace Preference (Figura 40). Cuando el build termine así se debe mostrar (Figura 41).

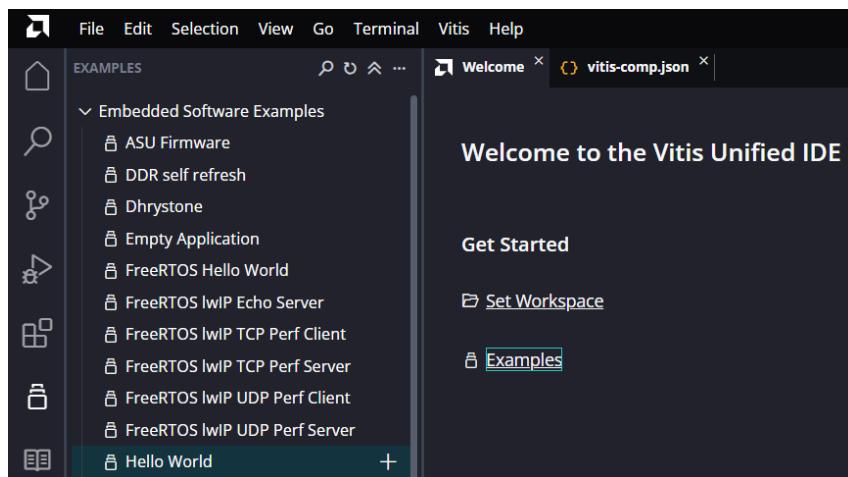


Figura 35: Click en Examples y luego en Hello World

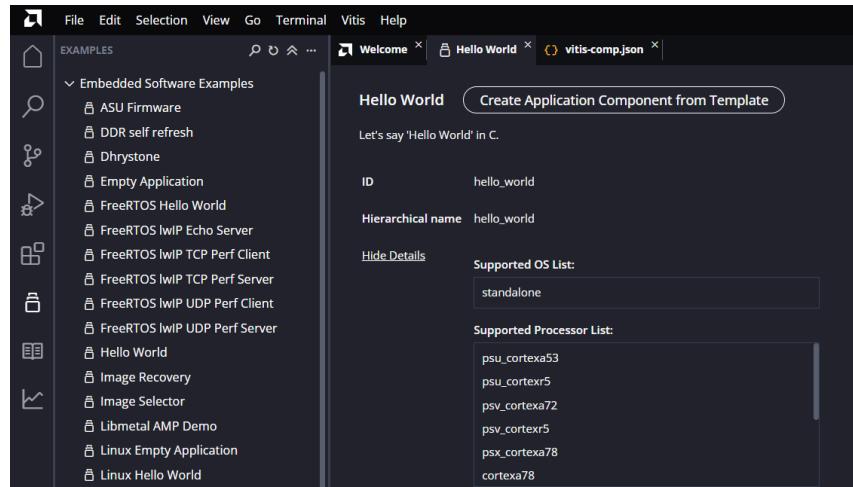


Figura 36: Click en Create Application Component from Template

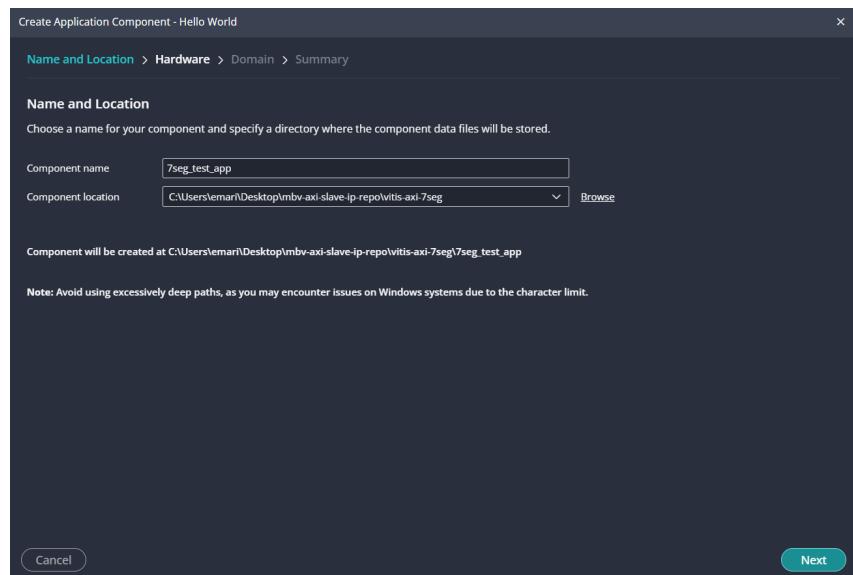


Figura 37: Escribir Component name

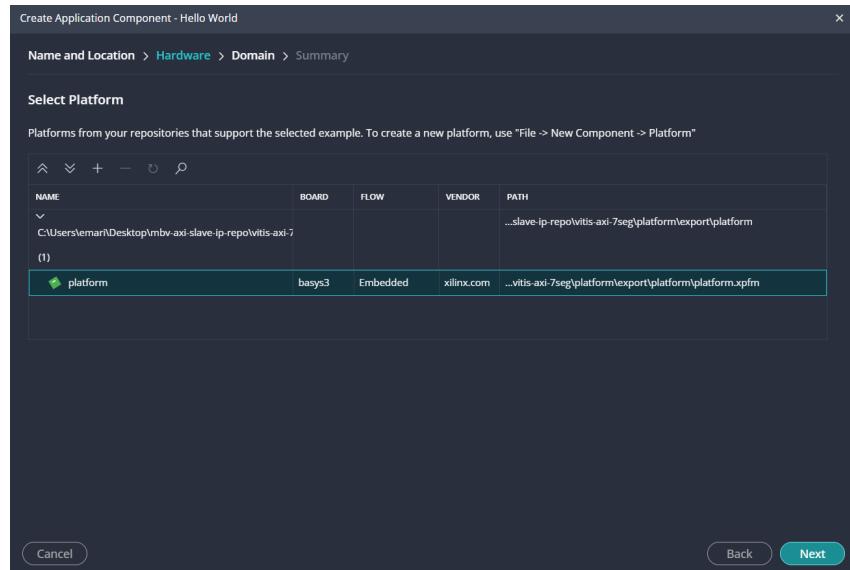


Figura 38: Seleccionar plataforma creada en el paso anterior

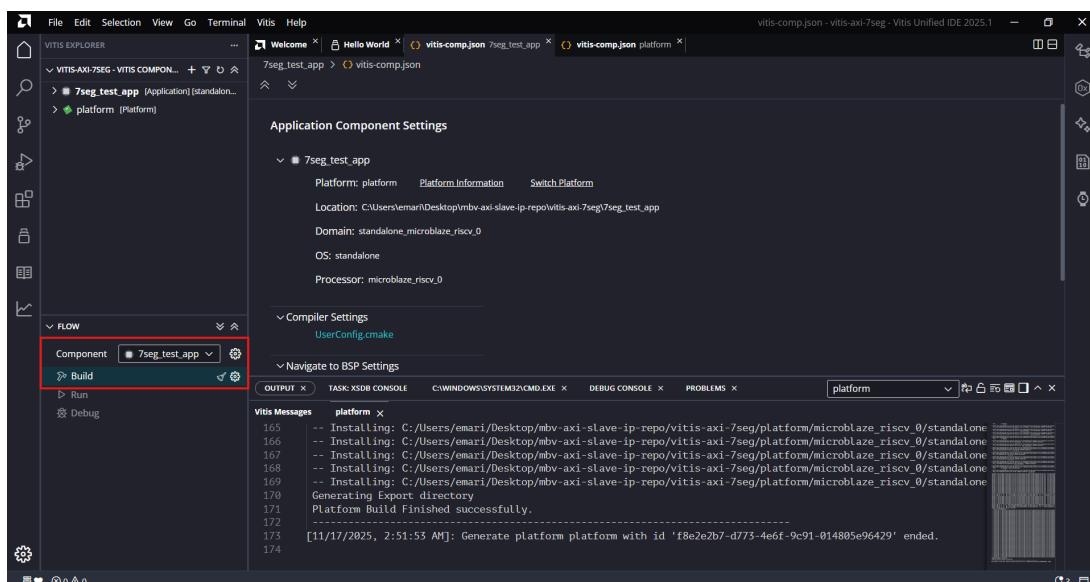


Figura 39: Click en Build

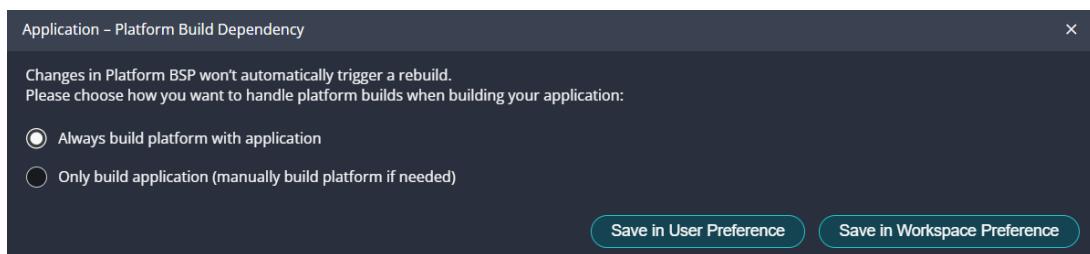


Figura 40: Click en Save in Workspace Preference

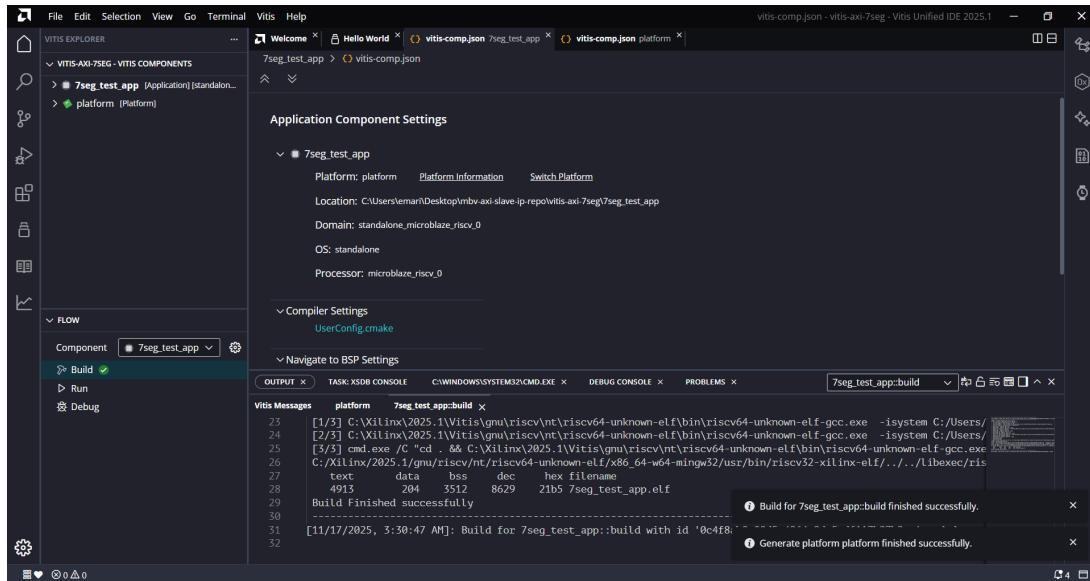


Figura 41: Build finalizado

Paso 4: Editar aplicación

En Sources/src cambiar el nombre de `helloworld.c` por `main.c` (Figura 42). Luego hacer el mismo cambio en la sección Sources de Settings/UserConfig.cmake (Figura 43), cerrar el archivo y dar click en Build.

Para importar el driver del módulo `axi_7seg_v1.0` es necesario dar click derecho en `src`, seleccionar `Import/Files...` (Figura 44). Buscar y seleccionar los archivos `axi_7seg.h` y `axi_7seg.c` (poner atención a la ruta de la Figura 45). Y Verificar que los archivos se hayan agregado a `src`, además que el archivo `axi_7seg.c` se encuentre en la sección Sources de Settings/UserConfig.cmake (Figura 46). Dar click en Build de nuevo.

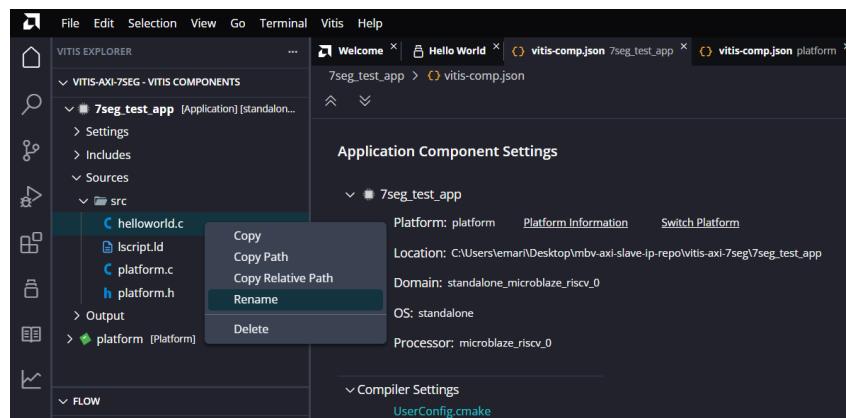


Figura 42: Renombrar archivo en Sources/src

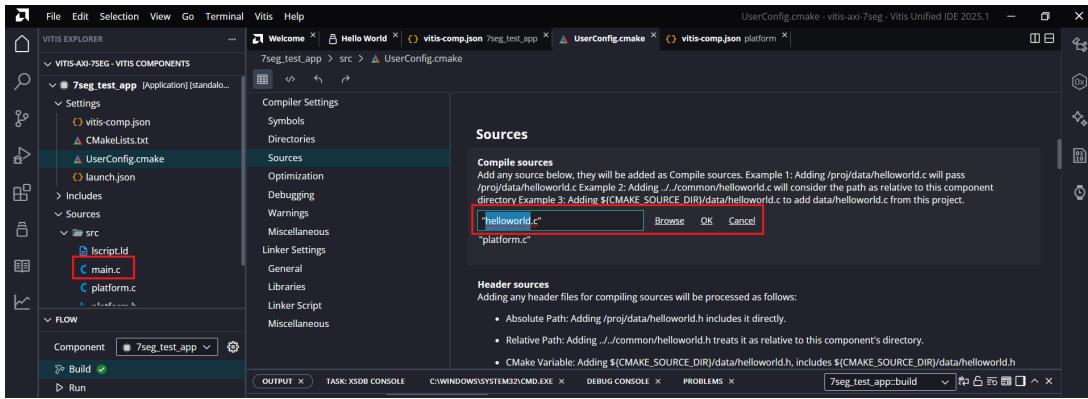


Figura 43: Renombrar archivo en Settings/UserConfig.cmake

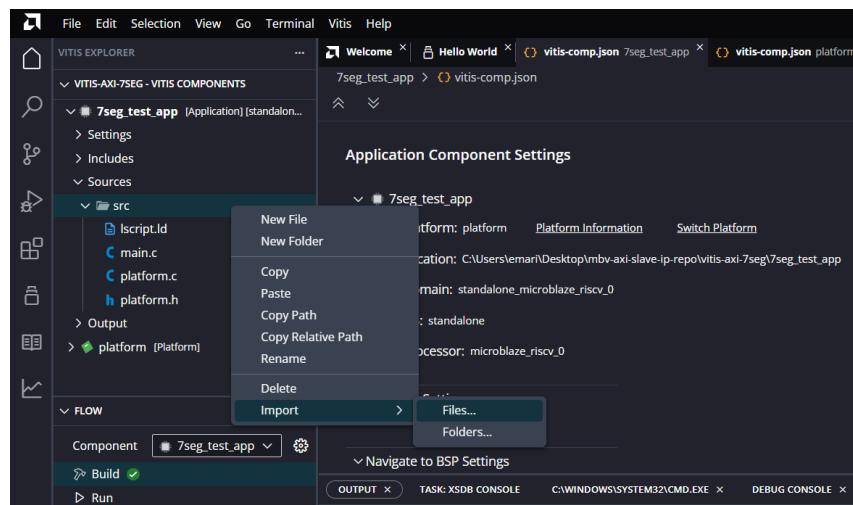


Figura 44: Click derecho en src y seleccionar Import/Files...

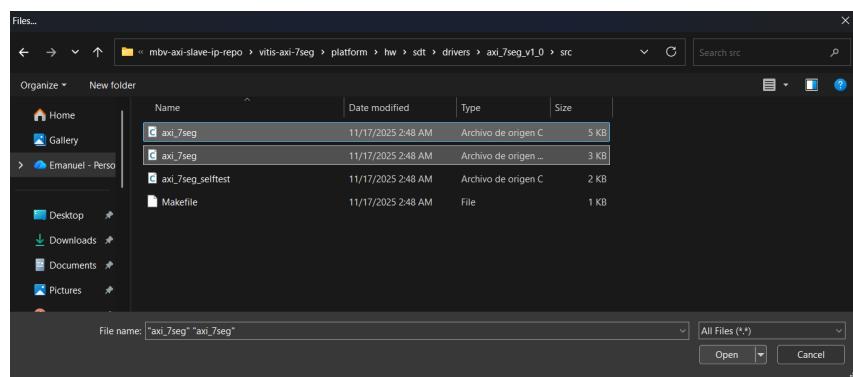


Figura 45: Buscar y seleccionar axi_7seg.h y axi_7seg.c

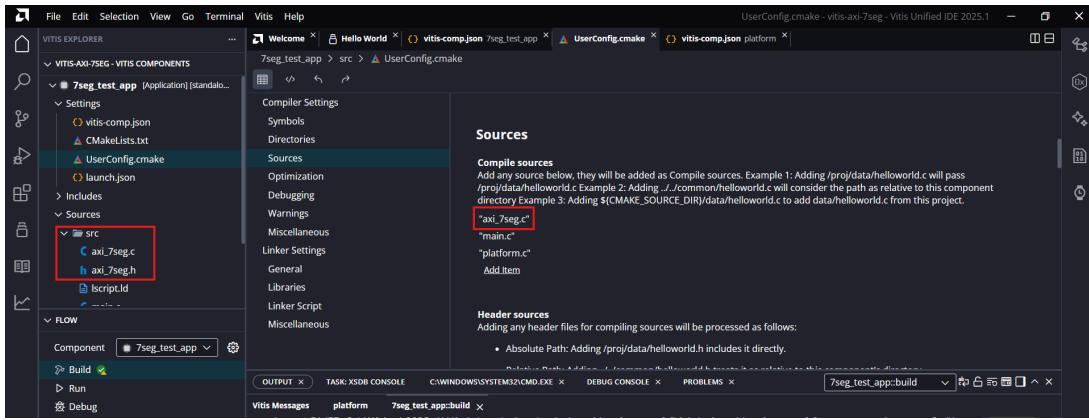


Figura 46: Verificar que el archivo `axi_7seg.c` se encuentre en la sección Sources de Settings/UserConfig.cmake

Paso 5: Cargar y ejecutar código de ejemplo

En primer lugar se debe eliminar el código que tiene `main.c`, luego se debe copiar el contenido del archivo `main_7seg_1.c` (ubicado en `mbv-axi-slave-ip-repo/vitis_code`) y pegarlo en el archivo `main.c`, guardar los cambios y dar click en Build.

En este punto es importante tener conectada la computadora host con la placa Basys3, mediante el cable USB tipo A a Micro-USB.

Para ejecutar o debuguear la aplicación, se debe seleccionar **Vitis/Serial Monitor** (Figura 47), luego seleccionar el puerto de comunicación, en este caso es el COM6 (FTDI) (confirmar en el Device Manager de Windows o su equivalente en Linux) y seleccionar un baud rate de 9600 (Figura 48 y 49).

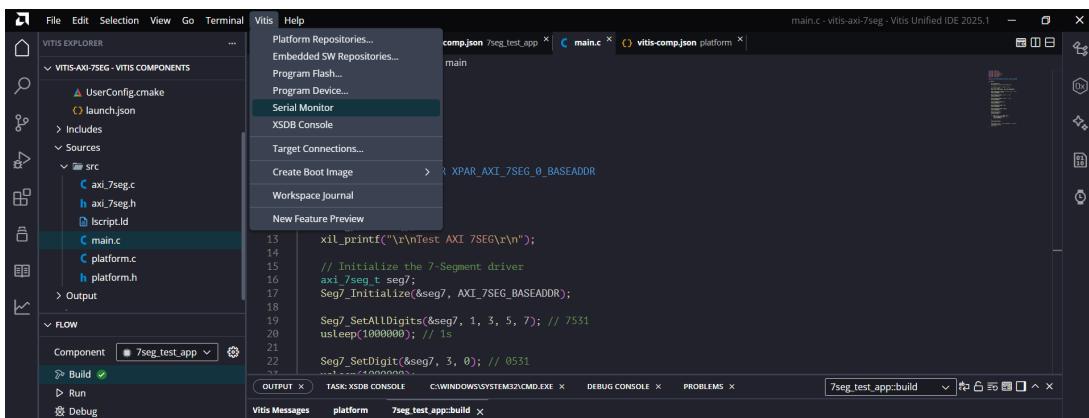


Figura 47: Click en Vitis/Serial Monitor

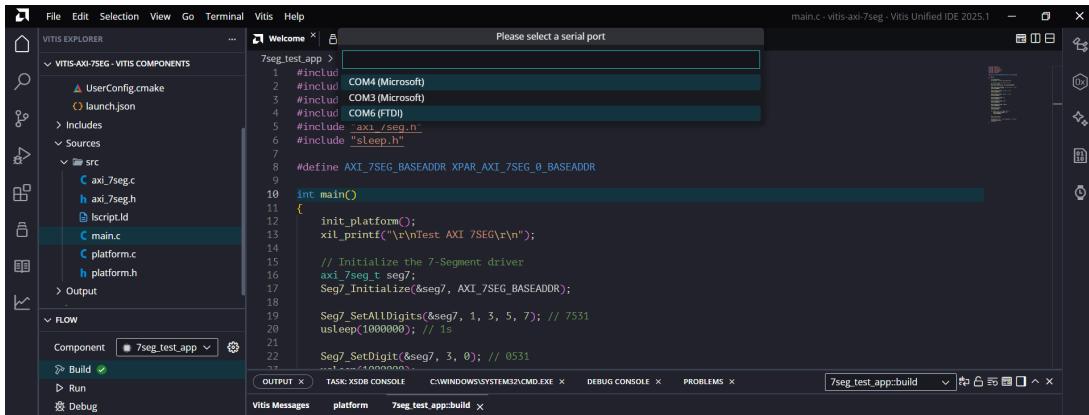


Figura 48: Seleccionar puerto COM6 (FTDI)

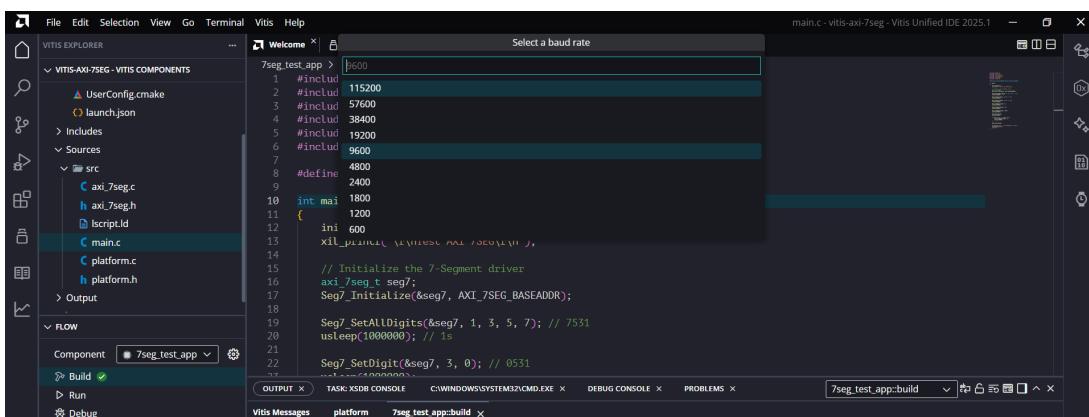


Figura 49: Seleccionar un baud rate de 9600

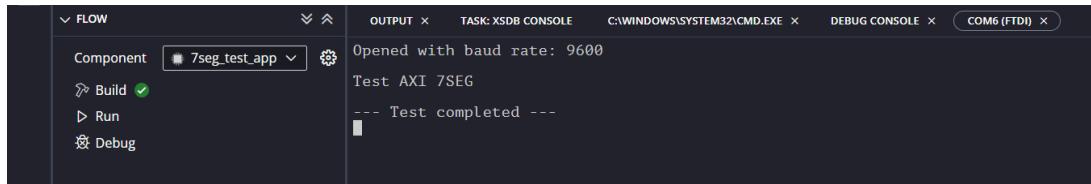
Esta aplicación ejecuta una secuencia de pruebas automáticas que demuestran las diferentes funciones del driver del display de 7 segmentos:

- 1. Configuración individual de dígitos:** Muestra el número 7531 estableciendo cada dígito de forma independiente usando Seg7_SetAllDigits()
- 2. Modificación de dígito específico:** Cambia el dígito más significativo a 0, resultando en 0531, demostrando Seg7_SetDigit()
- 3. Actualización parcial:** Modifica el dígito menos significativo a 0, mostrando 0530
- 4. Visualización de números:** Usa Seg7_SetNumber() para mostrar secuencialmente:
 - El número 3 (se mostrará como 0003)
 - El número 10 (se mostrará como 0010)
 - El número 2025 (año actual como ejemplo)
- 5. Reset del display:** Limpia el display usando Seg7_Reset(), estableciendo todos los dígitos a 0000

6. **Contador automático:** Implementa un contador de 0 a 100 que se incrementa cada 200 ms, demostrando la capacidad de actualización dinámica del display
7. **Finalización:** Limpia el display nuevamente al completar todas las pruebas

Cada transición incluye un retardo de 1 segundo para permitir la visualización clara de cada estado. Esta aplicación es ideal para verificar el funcionamiento básico del IP Core sin requerir interacción del usuario.

En la Figura 50 se muestran los prints en consola tras la ejecución de la aplicación. En la Figura 51 se muestra el funcionamiento del sistema en la FPGA.



The screenshot shows the Xilinx Serial Monitor interface. On the left, there's a sidebar with options like 'FLOW', 'Component' (set to '7seg_test_app'), 'Build' (with a green checkmark), 'Run', and 'Debug'. The main area has tabs for 'OUTPUT', 'TASK: XSDB CONSOLE', 'C:\WINDOWS\SYSTEM32\CMD.EXE', 'DEBUG CONSOLE', and 'COM6 (FTDI)'. The 'OUTPUT' tab is active and displays the following text:
 Opened with baud rate: 9600
 Test AXI 7SEG
 --- Test completed ---

Figura 50: Prints del Serial Monitor de acuerdo con el código de main_7seg_1.c

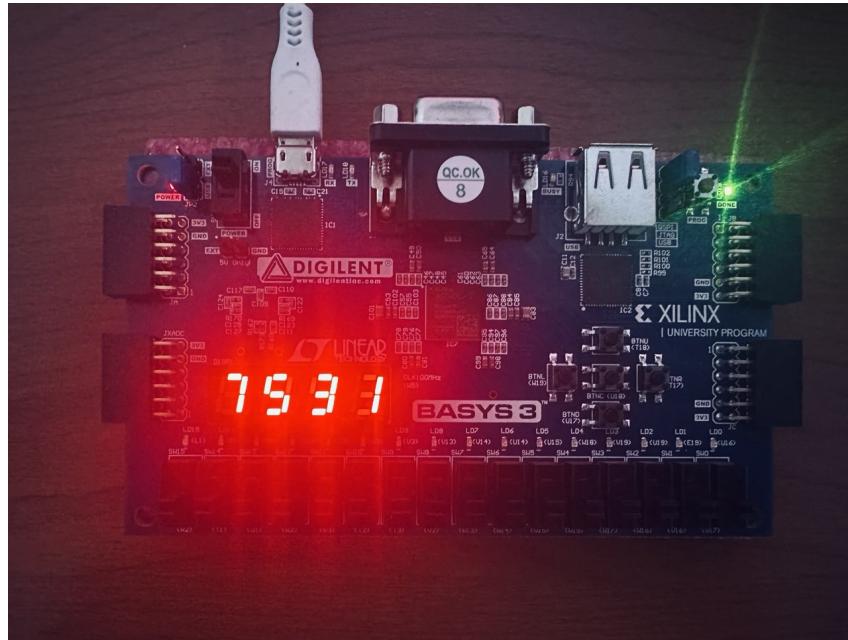


Figura 51: Funcionamiento del sistema en la FPGA de acuerdo con el código de main_7seg_1.c

Si en lugar de copiar y pegar el contenido del archivo `main_7seg_1.c` en `main.c`, se copia y pega el contenido del archivo `main_7seg_2.c`, se tiene que:

Esta aplicación implementa una interfaz interactiva que permite al usuario ingresar dígitos mediante el puerto UART y visualizarlos en tiempo real en el display de 7 segmentos:

1. **Inicialización:** Configura el driver del display y entra en un bucle infinito esperando entrada del usuario

2. **Lectura de dígitos:** Utiliza XUartLite_RecvByte() para leer caracteres ASCII desde el terminal serie
3. **Conversión ASCII a entero:** Resta 48 del valor ASCII para obtener el dígito numérico (convierte '0'-'9' a 0-9)
4. **Validación de entrada:**
 - Si el carácter es un dígito válido (0-9): Se acumula en la variable data mediante desplazamiento de 8 bits (data << 8) y se añade el nuevo dígito
 - Si el carácter no es un dígito (cualquier otra tecla): Se reinicia el contador a 0 y se limpia la línea del terminal
5. **Actualización del display:** Escribe directamente el registro SEG7_REG_DIGIT_VALUES con el valor acumulado, mostrando los últimos 4 dígitos ingresados

Ejemplo de uso: Si el usuario ingresa la secuencia 1, 2, 3, 4, el display mostrará 1234. Si continúa ingresando 5, el display actualizará a 2345 (descartando el dígito más antiguo). Al presionar cualquier tecla no numérica (como Enter), el display se reinicia a 0000.

Esta aplicación demuestra:

- Integración del display con el módulo UART Lite
- Actualización en tiempo real del hardware sin latencia perceptible
- Técnica de acumulación de dígitos mediante operaciones de desplazamiento de bits
- Interfaz simple pero funcional para entrada de usuario

En la Figura 52 se muestran los prints en consola tras la ejecución de la aplicación. En la Figura 53 se muestra el funcionamiento del sistema en la FPGA.

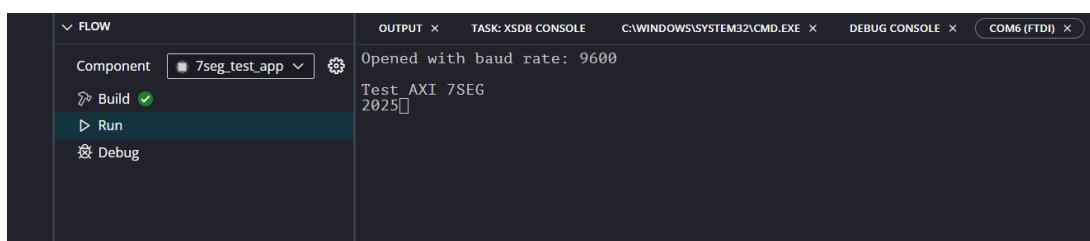


Figura 52: Prints del Serial Monitor de acuerdo con el código de main_7seg_2.c

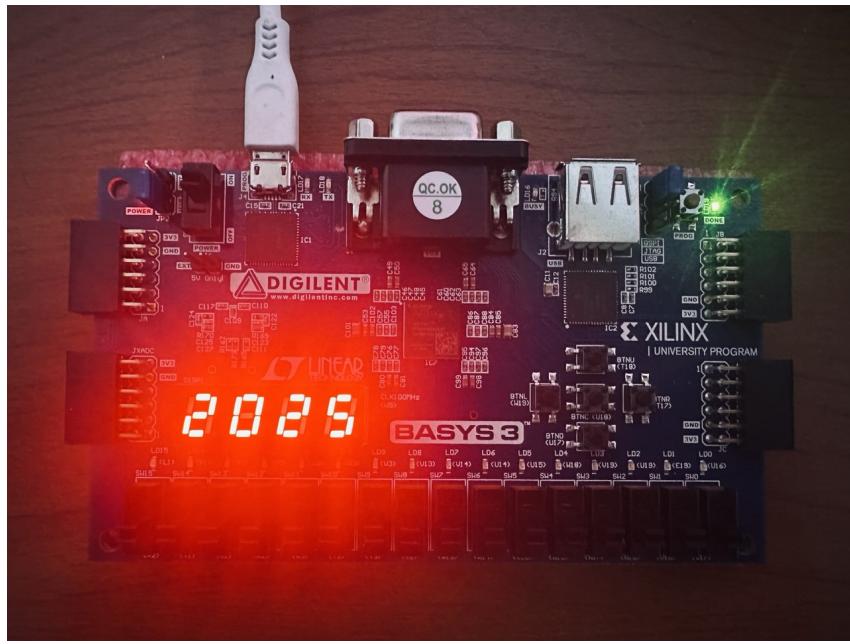


Figura 53: Funcionamiento del sistema en la FPGA de acuerdo con el código de main_7seg_2.c

3.3. Proyecto AXI PRNG (Pseudo Random Number Generator)

El proyecto AXI PRNG implementa un generador de números pseudo-aleatorios de alta velocidad basado en hardware utilizando el algoritmo XorShift32. Este IP Core permite al procesador MicroBlaze V generar secuencias de números aleatorios de 32 bits de forma eficiente, sin consumir ciclos de procesador en cálculos de generación, resultando ideal para aplicaciones de simulación, juegos, criptografía y procesamiento estadístico.

Descripción del IP Core

El axi_prng_1_0 es un IP Core desarrollado en Verilog que implementa el algoritmo XorShift32, conocido por su simplicidad, rapidez y buenas propiedades estadísticas. El módulo proporciona una interfaz AXI4-Lite Slave que permite la configuración dinámica de la semilla (seed) y el rango de salida, además de la lectura de números aleatorios generados por hardware.

Características principales

- **Algoritmo XorShift32:** Implementación en hardware del algoritmo de George Marsaglia que genera números pseudo-aleatorios de alta calidad mediante operaciones XOR y desplazamientos de bits.
- **Generación de 32 bits:** Produce números aleatorios de rango completo (0 a $2^{32} - 1$) en un solo ciclo de reloj.

- **Configuración dinámica de semilla:** Permite cambiar la semilla inicial en tiempo de ejecución para generar diferentes secuencias aleatorias.
- **Escalado automático por hardware:** Mapea los números aleatorios a un rango definido por el usuario [Low, High) sin usar operaciones de división, empleando multiplicación de alta precisión.
- **Doble salida:** Proporciona tanto el número aleatorio sin procesar (random_raw) como el número escalado al rango especificado (random_in_range).
- **Control de habilitación:** Sistema de enable/disable que permite controlar cuándo se generan nuevos valores.
- **Protección contra estado cero:** Garantiza que el generador nunca entre en un estado inválido (state = 0) que produciría una secuencia de ceros.

Algoritmo XorShift32

El algoritmo XorShift32 implementado sigue la siguiente secuencia de operaciones para generar el siguiente estado:

```
x1 = state ^ (state << 13)
x2 = x1      ^ (x1      >> 17)
x3 = x2      ^ (x2      << 5)
next_state = x3
```

Estas operaciones se ejecutan completamente en lógica combinacional, permitiendo la generación de un nuevo número aleatorio en cada ciclo de reloj cuando el generador está habilitado. El periodo de la secuencia es $2^{32} - 1$, cubriendo todos los valores posibles excepto el cero.

Mapeo de rango sin división

Una característica destacada de este IP Core es su capacidad de escalar números aleatorios a un rango arbitrario [Low, High) sin utilizar divisiones, que son costosas en hardware. El método implementado es:

```
span = High - Low
product = next_state * span          (multiplicación 32x32 -> 64 bits)
scaled = Low + product[63:32]        (tomar los 32 bits superiores)
```

Esta técnica, conocida como “multiply-high”, utiliza únicamente la parte alta del producto de 64 bits para obtener una distribución uniforme aproximada en el rango deseado, evitando el uso de divisores que consumirían significativamente más recursos lógicos.

Arquitectura del módulo

El IP Core está compuesto por dos módulos principales:

1. Módulo AXI4-Lite Slave (axi_prng_slave_lite_v1_0_S00_AXI):

- Implementa el protocolo de comunicación AXI4-Lite completo
- Gestiona 8 registros internos de 32 bits para control y lectura
- Interfaz de escritura para configuración (seed, rango, enable)
- Interfaz de lectura para obtener números aleatorios

2. Módulo XorShift32 RNG (xorshift32_rng):

- Implementa el algoritmo XorShift32 en lógica combinacional
- Registros de estado interno (32 bits)
- Registros de configuración de rango (low_reg, high_reg)
- Lógica de escalado sin división
- Señal de validez para indicar datos frescos

Mapa de registros

El IP Core utiliza ocho registros para su operación y configuración:

| Offset | Nombre | Descripción |
|--------|-----------------|---------------------------------------|
| 0x00 | ENABLE | Habilita/deshabilita la generación |
| 0x04 | UPDATE_SEED | Señal de actualización de semilla |
| 0x08 | NEW_SEED | Nuevo valor de semilla |
| 0x0C | UPDATE_RANGE | Señal de actualización de rango |
| 0x10 | NEW_LOW | Límite inferior del rango (inclusivo) |
| 0x14 | NEW_HIGH | Límite superior del rango (exclusivo) |
| 0x18 | RANDOM_RAW | Número aleatorio sin procesar (RO) |
| 0x1C | RANDOM_IN_RANGE | Número aleatorio escalado (RO) |

Parámetros configurables

El IP Core incluye los siguientes parámetros configurables:

- DEFAULT_SEED = 42: Semilla inicial por defecto
- DEFAULT_LOW = 0: Límite inferior por defecto del rango
- DEFAULT_HIGH = 100: Límite superior por defecto del rango

Driver de software

El IP Core incluye un driver en lenguaje C con las siguientes funciones:

- Prng_Initialize(): Inicializa el driver con la dirección base

- Prng_Enable(): Habilita la generación de números aleatorios
- Prng_Disable(): Deshabilita el generador
- Prng_ChangeSeed(): Cambia la semilla del generador
- Prng_ChanceRange(): Configura el rango de salida [Low, High)
- Prng_Random(): Lee un número aleatorio escalado al rango
- Prng_RawRandom(): Lee un número aleatorio de 32 bits sin escalar

Aplicaciones típicas

Este IP Core es útil en diversas aplicaciones que requieren generación rápida de números aleatorios:

- **Simulaciones Monte Carlo:** Generación rápida de grandes cantidades de números aleatorios para análisis estadístico
- **Juegos y entretenimiento:** Dados virtuales, barajas de cartas, eventos aleatorios en juegos
- **Generación de IDs:** Creación de identificadores únicos o tokens temporales
- **Algoritmos de muestreo:** Selección aleatoria de datos o muestreo estadístico
- **Pruebas de hardware:** Generación de patrones de prueba pseudo-aleatorios
- **Seed para TRNG:** Puede usarse como pre-procesador para True Random Number Generators

Consideraciones de uso

- **Determinismo:** Al ser un PRNG, la secuencia es completamente determinista dada una semilla. Para aplicaciones criptográficas se recomienda usar el IP Core TRNG.
- **Calidad estadística:** XorShift32 tiene buenas propiedades estadísticas pero no pasa todas las pruebas de TestU01. Para aplicaciones críticas considere usar el TRNG o combinar múltiples generadores.
- **Semilla cero:** Evite usar semilla = 0, ya que el generador entraría en un estado fijo. El hardware lo previene utilizando la semilla por defecto.
- **Rango inválido:** Si High \leq Low, el hardware ajusta automáticamente a High = Low + 1.

Las siguientes subsecciones detallan el proceso de creación de un proyecto en Vivado y Vitis utilizando este IP Core, desde la configuración inicial hasta la ejecución de aplicaciones que demuestran diferentes casos de uso del generador de números pseudo-aleatorios.

3.3.1. Crear proyecto Vivado (vivado-axi-prng)

Ejecutar los **Pasos** tal y como se describe en 3.2.1. No obstante, para el **Paso 1** se debe setear:

```
set ::user_project_name "vivado-axi-prng"
```

El **Paso 2** es exactamente igual. En el **Paso 3** se debe buscar el módulo `axi_prng_v1_0`. Dicho módulo tiene un puerto de entrada (`prng_reset`) que tiene que ser marcado como `Make External`. En la Figura 54 se observa el diseño actual.

En el **Paso 4**, el archivo que se debe seleccionar es `prng.xdc`. El **Paso 5** es exactamente igual.

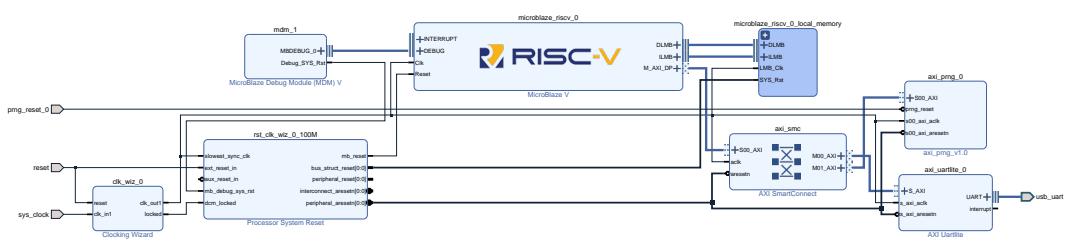


Figura 54: Block Design después de agregar el módulo `axi_prng_v1_0`

3.3.2. Crear proyecto Vitis (vitis-axi-prng)

Ejecutar los **Pasos** tal y como se describe en 3.2.2. El **Paso 1** es exactamente igual. En el **Paso 2** el workspace a crear y seleccionar dentro de `mbv-axi-slave-ip-repo` es `vitis-axi-prng`.

En el **Paso 3**, el `Component name` de la aplicación debe ser `prng_test_app`. Con el **Paso 4** se deben importar los archivos `axi_prng.h` y `axi_prng.c`.

En el **Paso 5** se debe copiar y pegar el contenido del archivo `main_prng.c` en `main.c`

Esta aplicación ejecuta un conjunto pruebas que demuestran las capacidades del generador de números pseudo-aleatorios XorShift32, incluyendo la generación con diferentes semillas y rangos:

Estructura de la demostración

1. Inicialización del sistema:

- Inicializa el driver del PRNG mediante `Prng_Initialize()`
- Configura la dirección base del IP Core (`XPAR_AXI_PRNG_0_BASEADDR`)
- Prepara el sistema para pruebas exhaustivas

2. Configuración de pruebas:

La aplicación define dos conjuntos de parámetros para demostrar la versatilidad del generador:

- **Semillas de prueba** (4 valores):

- 0x000002F1
- 0xDEADBEEF
- 0xCAFEBABE
- 0xFFFFFFFF

- **Rangos de prueba** (5 configuraciones):

- [0, 100]: Rango original por defecto
- [50, 150]: Rango desplazado (offset positivo)
- [1000, 2000]: Rango amplio para valores grandes
- [1, 10]: Rango pequeño
- [999, 1002]: Rango muy estrecho (3 valores posibles)

3. Pruebas sistemáticas:

La aplicación ejecuta un doble bucle anidado que realiza:

1. **Bucle externo** (4 iteraciones): Cambia la semilla del generador
 - Imprime la semilla actual en formato hexadecimal
 - Aplica la nueva semilla usando `Prng_ChangeSeed()`
2. **Bucle intermedio** (5 iteraciones): Cambia el rango de salida
 - Imprime el rango actual [low, high]
 - Configura el nuevo rango usando `Prng_ChanceRange()`
3. **Bucle interno** (5 iteraciones): Genera números aleatorios
 - Genera un número aleatorio escalado usando `Prng_Random()`
 - Lee el valor raw sin escalar usando `Prng_RawRandom()`
 - Imprime ambos valores: el escalado (decimal) y el raw (hexadecimal)

Total de números generados: $4 \times 5 \times 5 = 100$ números aleatorios con diferentes configuraciones.

4. Simulación de datos:

Tras las pruebas sistemáticas, la aplicación demuestra un caso de uso práctico:

- Configura una nueva semilla específica: 0x12345678

- Establece el rango [1, 7) para simular un dado de 6 caras (valores: 1, 2, 3, 4, 5, 6)
- Genera 10 tiradas de dado consecutivas (NUM_RANDOM_VALUES × 2)
- Imprime cada resultado como “Dice roll [n]: [valor]”

En el Serial Monitor de las Figuras 55 y 56 se muestran los prints de ejecución del sistema en la FPGA. Como se puede observar, los resultados que genera el PRNG son idénticos en ambas ejecuciones, esto está alineado con que un generador de números pseudo-aleatorio debe ser determinista.

```

File Edit Selection View Go Terminal Vitis Help
VITIS EXPLORER
  VITIS-AXI-PRNG - VITIS COMPONENTS
    > platform [Platform]
      > prng_test_app [Application] (standalone.m...
        main.c - vitis-axi-prng - Vitis Unified IDE 2025.1
        Welcome | vitis-comp.json platform | vitis-comp.json prng_test_app | main.c
        ...
        Random[3] = 118 (raw: 0x80A6275A)
        Random[4] = 149 (raw: 0x54950720)
        --- Range [1000, 2000] ---
        Random[0] = 1478 (raw: 0x922C516)
        Random[1] = 1052 (raw: 0x514C14F9)
        Random[2] = 1662 (raw: 0x70BE239D)
        Random[3] = 1588 (raw: 0x64600114)
        Random[4] = 1922 (raw: 0xA26F9606)
        --- Range [1, 10] ---
        Random[0] = 3 (raw: 0x2A190F56)
        Random[1] = 5 (raw: 0x8AA576C9E)
        Random[2] = 9 (raw: 0x84B8AAC1)
        Random[3] = 6 (raw: 0xED73A89C)
        Random[4] = 6 (raw: 0x4567F508)
        --- Range [999, 1002] ---
        Random[0] = 999 (raw: 0x8793A42A)
        Random[1] = 1000 (raw: 0xAA576C98)
        Random[2] = 1001 (raw: 0x6E1EDC7C)
        Random[3] = 999 (raw: 0x885E930F)
        Random[4] = 1000 (raw: 0x87BDDC43)
        === Simulating Dice Rolls (1-6) ===
        Dice roll 1: 4
        Dice roll 2: 1
        Dice roll 3: 2
        Dice roll 4: 4
        Dice roll 5: 3
        Dice roll 6: 1
        Dice roll 7: 4
        Dice roll 8: 5
        Dice roll 9: 5
        Dice roll 10: 2
  
```

Figura 55: Ejecución 1. Prints del Serial Monitor de acuerdo con el código de main_prng.c

```

File Edit Selection View Go Terminal Vitis Help
VITIS EXPLORER
  VITIS-AXI-PRNG - VITIS COMPONENTS
    > platform [Platform]
      > prng_test_app [Application] (standalone.m...
        main.c - vitis-axi-prng - Vitis Unified IDE 2025.1
        Welcome | vitis-comp.json platform | vitis-comp.json prng_test_app | main.c
        ...
        Random[3] = 118 (raw: 0x80A6275A)
        Random[4] = 149 (raw: 0x54950720)
        --- Range [1000, 2000] ---
        Random[0] = 1478 (raw: 0x922C516)
        Random[1] = 1052 (raw: 0x514C14F9)
        Random[2] = 1662 (raw: 0x70BE239D)
        Random[3] = 1588 (raw: 0x64600114)
        Random[4] = 1922 (raw: 0xA26F9606)
        --- Range [1, 10] ---
        Random[0] = 3 (raw: 0x2A190F56)
        Random[1] = 5 (raw: 0x8AA576C9E)
        Random[2] = 9 (raw: 0x84B8AAC1)
        Random[3] = 6 (raw: 0xED73A89C)
        Random[4] = 6 (raw: 0x4567F508)
        --- Range [999, 1002] ---
        Random[0] = 999 (raw: 0x8793A42A)
        Random[1] = 1000 (raw: 0xAA576C98)
        Random[2] = 1001 (raw: 0x6E1EDC7C)
        Random[3] = 999 (raw: 0x885E930F)
        Random[4] = 1000 (raw: 0x87BDDC43)
        === Simulating Dice Rolls (1-6) ===
        Dice roll 1: 4
        Dice roll 2: 1
        Dice roll 3: 2
        Dice roll 4: 4
        Dice roll 5: 3
        Dice roll 6: 1
        Dice roll 7: 4
        Dice roll 8: 5
        Dice roll 9: 5
        Dice roll 10: 2
  
```

Figura 56: Ejecución 2. Prints del Serial Monitor de acuerdo con el código de main_prng.c

Conceptos demostrados

Esta aplicación ilustra varios conceptos importantes:

1. **Reproducibilidad:** La misma semilla siempre produce la misma secuencia de números, permitiendo debugging y verificación determinista
2. **Independencia de rangos:** El escalado de rango mantiene las propiedades estadísticas del generador subyacente
3. **Versatilidad de configuración:** Cambios dinámicos de semilla y rango sin reiniciar el sistema
4. **Visualización dual:** Mostrar tanto valores raw (hexadecimal) como escalados (decimal) ayuda a comprender el proceso de mapeo
5. **Aplicación práctica:** La simulación de dados demuestra un caso de uso real común en juegos y simulaciones
6. **Distribución en rangos estrechos:** Los rangos pequeños (como [999, 1002]) muestran la capacidad de generar valores en espacios limitados

Análisis de resultados esperados

Propiedades observables:

- **Uniformidad:** Los números generados deben distribuirse uniformemente en el rango especificado (observable con suficientes muestras)
- **Cobertura:** Para rangos pequeños, todos los valores deberían aparecer eventualmente
- **No correlación aparente:** Los valores consecutivos no deben mostrar patrones obvios
- **Rango correcto:** Todos los valores deben estar dentro de [low, high], nunca fuera

3.4. Proyecto AXI TRNG (True Random Number Generator)

El proyecto AXI TRNG implementa un generador de números verdaderamente aleatorios basado en hardware utilizando osciladores en anillo (ring oscillators) como fuente de entropía física. A diferencia del PRNG que genera secuencias deterministas, este IP Core produce números aleatorios no reproducibles basados en variaciones térmicas y de jitter intrínsecas del silicio, haciéndolo adecuado para aplicaciones criptográficas, generación de claves y aplicaciones de seguridad donde se requiere entropía real.

Descripción del IP Core

El axi_trng_1_0 es un IP Core desarrollado en Verilog que implementa un TRNG completo con las siguientes características distintivas:

- **Fuente de entropía física:** Utiliza variaciones de jitter y ruido térmico inherentes a la FPGA
- **No determinismo:** Las secuencias generadas no son reproducibles ni predecibles
- **Health tests integrados:** Monitoreo continuo de calidad de entropía mediante RCT y APT
- **Postprocesamiento:** Incluye debiasing Von Neumann para mejorar uniformidad estadística
- **Cumplimiento de estándares:** Diseño basado en recomendaciones de NIST SP 800-90B y AIS-31

Características principales

- **Banco de ring oscillators:** 8 osciladores independientes de 5 etapas cada uno como fuentes de entropía paralelas
- **Mezcla XOR:** Combinación de todas las salidas de osciladores mediante XOR para maximizar entropía
- **Muestreo decorrelacionado:** Divisor de frecuencia programable para controlar el intervalo de muestreo y optimizar recolección de entropía
- **Debiasing Von Neumann:** Algoritmo clásico que elimina sesgos estadísticos produciendo bits equiprobables
- **Health tests en línea:** Monitoreo continuo mediante Repetition Count Test (RCT) y Adaptive Proportion Test (APT)
- **Acumulación a 32 bits:** Pipeline que recolecta bits individuales hasta formar palabras completas de 32 bits
- **Escalado automático:** Mapeo sin división a rangos arbitrarios [Low, High] usando multiplicación de alta precisión
- **Señales de estado:** Reporta salud del generador, ocupación del pipeline y fallas de tests

Arquitectura del sistema

La arquitectura del TRNG está compuesta por múltiples etapas de procesamiento:

1. Fuente de Entropía - Ring Oscillators (RO):

Los ring oscillators son cadenas de inversores retroalimentados que oscilan a frecuencias indeterminadas debido a:

- Variaciones de proceso (PVT: Process, Voltage, Temperature)
- Jitter de propagación en compuertas lógicas

- Ruido térmico del transistor
- Interferencias electromagnéticas del entorno

Cada RO está implementado como una cadena de 5 inversores (número impar obligatorio para oscilación):

```
n[0] = ~n[4]  (con enable gating)
n[1] = ~n[0]
n[2] = ~n[1]
n[3] = ~n[2]
n[4] = ~n[3]
ro_out = n[4]
```

Advertencia importante: Los sintetizadores tienden a optimizar bucles combinacionales. Es crítico usar constraints XDC para preservar la estructura del RO.

2. Mezcla XOR:

Las 8 salidas de los ROs se combinan mediante una operación XOR-reduce:

```
ro_mix = ro_bits[0] ^ ro_bits[1] ^ ... ^ ro_bits[7]
```

Esto maximiza la entropía al combinar múltiples fuentes independientes de aleatoriedad.

3. Divisor de Muestreo (Sample Divider):

Un contador programable controla la frecuencia de muestreo del bit mezclado. Valores típicos:

- **Divider = 1:** Muestreo en cada ciclo de reloj (máxima velocidad, menor entropía por bit)
- **Divider = 1000 (default):** Muestreo cada 1000 ciclos (balance velocidad/entropía)
- **Divider >1000:** Muestreo más lento (mayor independencia entre bits, mejor entropía)

4. Sincronización y Debiasing Von Neumann:

El bit muestreado se sincroniza al dominio de reloj principal y pasa por el algoritmo de Von Neumann que:

- Procesa bits en pares: (0,0), (0,1), (1,0), (1,1)
- Descarta pares idénticos: (0,0) y (1,1)
- Emite el primer bit de pares diferentes: (0,1) → 0, (1,0) → 1
- Eficiencia: 25 % (descarta 75 % de bits pero garantiza equiprobabilidad)

5. Health Tests:

Repetition Count Test (RCT):

- Detecta fallas tipo “stuck-at” (bits que no cambian)

- Cuenta bits consecutivos idénticos
- Alarma si run length >64 (configurable vía parámetro RCT_MAX_RUN)
- Indica problemas de sincronización o fallas de hardware

Adaptive Proportion Test (APT):

- Monitorea balance estadístico de ceros y unos
- Ventana de 512 bits (APT_WIN_SIZE)
- Cuenta número de unos en cada ventana
- Rango aceptable: 192-320 unos (37.5 %-62.5 %)
- Alarma si cuenta fuera de [APT_LOW_THRESH, APT_HIGH_THRESH]
- Detecta sesgos estadísticos o problemas de entropía

6. Acumulador y Escalado:

Los bits validados se acumulan en un registro shift de 32 bits. Al completarse:

- Se emite random_raw (número completo de 32 bits)
- Se calcula random_in_range mediante multiply-high
- Se activa la señal valid

Mapa de registros

El IP Core utiliza siete registros para operación y monitoreo:

| Offset | Nombre | Descripción |
|--------|-----------------|-----------------------------------|
| 0x00 | CONTROL | Control: enable, clear alarms |
| 0x04 | SAMPLE_DIV | Divisor de muestreo (velocidad) |
| 0x08 | RANGE_LOW | Límite inferior del rango |
| 0x0C | RANGE_HIGH | Límite superior del rango |
| 0x10 | RANDOM_RAW | Número aleatorio crudo (RO) |
| 0x14 | RANDOM_IN_RANGE | Número escalado al rango (RO) |
| 0x18 | STATUS | Estado: health, busy, alarms (RO) |

Registro CONTROL (0x00):

- Bit [0]: ENABLE - Habilita generación de números aleatorios
- Bit [1]: CLR_ALARMS - Pulso para limpiar alarmas de health tests

Registro STATUS (0x18):

- Bit [0]: HEALTH_OK - 1 = ambos tests (RCT y APT) pasando
- Bit [1]: BUSY - 1 = generador activo

- Bit [2]: RCT_FAIL - 1 = Repetition Count Test falló
- Bit [3]: APT_FAIL - 1 = Adaptive Proportion Test falló

Parámetros configurables

El IP Core incluye los siguientes parámetros configurables:

- DEFAULT_LOW = 0: Límite inferior por defecto del rango
- DEFAULT_HIGH = 100: Límite superior por defecto del rango

Driver de software

El IP Core incluye un driver completo en C con funciones de control y monitoreo:

Funciones de inicialización y control:

- Trng_Initialize(): Inicializa el driver con valores por defecto
- Trng_Enable(): Habilita la generación de números aleatorios
- Trng_Disable(): Deshabilita el generador (ahorro de energía)
- Trng_Reset(): Reinicia configuración a valores por defecto

Funciones de configuración:

- Trng_SetSampleDivider(): Ajusta velocidad de muestreo
- Trng_SetRange(): Configura rango de salida [Low, High)
- Trng_ClearAlarms(): Limpia alarmas de health tests

Funciones de lectura:

- Trng_Random(): Lee número aleatorio escalado al rango
- Trng_RawRandom(): Lee número aleatorio de 32 bits completo

Funciones de monitoreo:

- Trng_GetStatus(): Lee registro de estado completo
- Trng_IsHealthy(): Verifica si ambos tests pasan (recomendado)
- Trng_IsBusy(): Verifica si el generador está activo
- Trng_IsRctFailed(): Verifica falla en RCT
- Trng_IsAptFailed(): Verifica falla en APT

Aplicaciones críticas

Este IP Core es apropiado para aplicaciones de seguridad y criptografía:

- **Generación de claves criptográficas:** Producción de claves AES, RSA, ECC
- **Nonces y IVs:** Vectores de inicialización para cifrados de flujo y bloque
- **Generación de desafíos:** Protocolos challenge-response de autenticación
- **Padding aleatorio:** OAEP, PSS para esquemas de firma digital
- **Generación de salt:** Para hashing de contraseñas (bcrypt, scrypt)
- **Números aleatorios de sesión:** Tokens de sesión, CSRF tokens
- **Seed para PRNG:** Inicialización de generadores deterministas con entropía real
- **Protocolos TLS/SSL:** Generación de randoms en handshake

Consideraciones de diseño y uso

- **Constraints XDC requeridas:** Es *crítico* incluir constraints para preservar los ring oscillators y deshabilitar análisis de timing en bucles combinacionales. Sin estas constraints, el sintetizador optimizará los ROs y el diseño no funcionará.
- **Velocidad vs Entropía:** Existe un trade-off fundamental. Sample dividers más altos (muestreo más lento) producen bits con mayor independencia y mejor entropía, pero reducen el throughput.
- **Monitoreo continuo:** Siempre verifique Trng_IsHealthy() antes de usar números aleatorios en aplicaciones críticas. Las alarmas indican problemas de hardware o entropía insuficiente.
- **Variación entre FPGAs:** Diferentes chips tendrán frecuencias de RO ligeramente diferentes debido a variaciones de proceso. Esto es deseable y aumenta la entropía en sistemas multi-chip.
- **Validación estadística:** Para aplicaciones criptográficas, valide la salida con test suites como NIST SP 800-22 (dieharder, TestU01).
- **No determinismo garantizado:** A diferencia del PRNG, dos ejecuciones idénticas del código producirán secuencias completamente diferentes.

Recomendación: Use PRNG para simulaciones, juegos y aplicaciones no críticas. Use TRNG para criptografía, seguridad y generación de claves.

Las siguientes subsecciones detallan el proceso de creación de un proyecto en Vivado y Vitis utilizando este IP Core, desde la configuración inicial hasta la ejecución de aplicaciones que demuestran la generación de números verdaderamente aleatorios con monitoreo de salud del sistema.

3.4.1. Crear proyecto Vivado (vivado-axi-trng)

Ejecutar los **Pasos** tal y como se describe en 3.2.1. No obstante, para el **Paso 1** se debe setear:

```
set ::user_project_name "vivado-axi-trng"
```

El **Paso 2** es exactamente igual. En el **Paso 3** se debe buscar el módulo axi_trng_v1.0. En la Figura 57 se observa el diseño actual.

En el **Paso 4**, el archivo que se debe seleccionar es trng.xdc. El **Paso 5** es exactamente igual.



Figura 57: Block Design después de agregar el módulo axi_trng_v1.0

3.4.2. Crear proyecto Vitis (vitis-axi-trng)

Ejecutar los **Pasos** tal y como se describe en 3.2.2. El **Paso 1** es exactamente igual. En el **Paso 2** el workspace a crear y seleccionar dentro de mbv-axi-slave-ip-repo es vitis-axi-trng.

En el **Paso 3**, el Component_name de la aplicación debe ser trng_test_app. Con el **Paso 4** se deben importar los archivos axi_trng.h y axi_trng.c.

En el **Paso 5** se debe copiar y pegar el contenido del archivo main_trng.c en main.c

Esta aplicación demuestra el funcionamiento completo del generador de números verdaderamente aleatorios (TRNG) basado en ring oscillators, incluyendo generación de números, configuración de rangos y monitoreo del estado de salud del sistema mediante los tests RCT y APT.

Estructura de la demostración

1. Inicialización y configuración:

- Inicializa el driver del TRNG mediante Trng_Initialize()
- Configura la dirección base del IP Core (XPAR_AXI_TRNG_0_BASEADDR)
- Opcionalmente permite ajustar dos parámetros críticos (comentados por defecto):
 - **Sample divider:** Trng_SetSampleDivider() - controla la velocidad de muestreo de los ring oscillators (valor por defecto: 1000, mínimo: 1)

- **Rango de salida:** Trng_SetRange() - define los límites del escalado (por defecto: [0, 100])
- Habilita el generador mediante Trng_Enable()

2. Generación de números aleatorios:

La aplicación genera una serie de números aleatorios con la configuración por defecto:

- **Cantidad:** NUM_RANDOM_VALUES (5 números por defecto)
- **Método de lectura dual:**
 - Trng_Random(): Lee el número escalado al rango configurado
 - Trng_RawRandom(): Lee el número sin procesar de 32 bits completos
- **Visualización:** Imprime ambos valores en cada iteración. Ejemplo:

```
Random[0] = 67 (raw: 0xAB12C3D4)
Random[1] = 23 (raw: 0x5E7F8901)
```

3. Simulación de dados con TRNG:

Tras la generación inicial, la aplicación demuestra un caso de uso práctico:

- Reconfigura el rango a [1, 7) para simular un dado de 6 caras
- Utiliza Trng_SetRange(&trng, 1, 7) para establecer el nuevo rango
- Genera 10 tiradas de dado (NUM_RANDOM_VALUES × 2)
- Cada tirada produce un valor entre 1 y 6 (inclusivo)
- Imprime cada resultado: "Dice roll [n]: [valor]"

Importante: A diferencia del PRNG, estas tiradas de dado son *verdaderamente aleatorias* y no reproducibles, incluso si se ejecuta el programa múltiples veces en condiciones idénticas.

4. Apagado del generador:

- Deshabilita el TRNG usando Trng_Disable() para ahorrar energía
- Los ring oscillators dejan de oscilar cuando el generador está deshabilitado

5. Monitoreo de salud del sistema:

Una característica distintiva de esta aplicación es la función print_status(), que proporciona información detallada sobre el estado del generador.

Esta función verifica:

- **Status completo:** Registro de estado raw (32 bits)
- **Health OK:** Indicador global de salud (1 = ambos tests pasando)

- **Busy:** Estado de ocupación del pipeline de generación
- **RCT Fail:** Estado del Repetition Count Test (1 = fallo detectado)
- **APT Fail:** Estado del Adaptive Proportion Test (1 = fallo detectado)

La función se invoca al final de la ejecución para diagnosticar el estado del hardware tras uso intensivo.

En el Serial Monitor de las Figuras 58 y 59 se muestran los prints de ejecución del sistema en la FPGA. Como se puede observar, los resultados que genera el TRNG son completamente diferentes, esto está alineado con lo que se espera de un generador de verdaderos aleatorios.

```

File Edit Selection View Go Terminal Vitis Help
VITIS EXPLORER
  VITIS-AXI-TRNG - VITIS COMPONENTS
    > platform [Platform]
    < trng_test_app [Application] [standalone...]
      > Settings
      > Includes
      > Sources
        > src
          C axi_trng.c
          h axi_trng.h
          lscript.ld
          C main.c
          C platform.c
          h platform.h
      > FLOW
        Component: trng_test_app
        Build > Run > Debug
      > Build
      > Run
      > Debug

OUTPUT X TASK:XSDB CONSOLE DEBUG CONSOLE X PROBLEMS X
Opened with baud rate: 9600
Test AXI TRNG
--- Generate 5 random numbers---
Random[0] = 26 (raw: 0x86239F93)
Random[1] = 43 (raw: 0xE03328AF)
Random[2] = 34 (raw: 0xAE4A45FB)
Random[3] = 81 (raw: 0xA520D140)
Random[4] = 88 (raw: 0xC5AB4B6C9)
== Simulating Dice Rolls (1-6) ===
Dice roll 1: 2
Dice roll 2: 2
Dice roll 3: 3
Dice roll 4: 2
Dice roll 5: 5
Dice roll 6: 5
Dice roll 7: 4
Dice roll 8: 3
Dice roll 9: 5
Dice roll 10: 2
TRNG Status: 0x00000001
Health OK: 1
Busy: 0
RCT Fail: 0
APT Fail: 0
--- Test completed ---

```

Figura 58: Ejecución 1. Prints del Serial Monitor de acuerdo con el código de main_trng.c

```

File Edit Selection View Go Terminal Vitis Help
VITIS EXPLORER
  VITIS-AXI-TRNG - VITIS COMPONENTS
    > platform [Platform]
    < trng_test_app [Application] [standalone...]
      > Settings
      > Includes
      > Sources
        > src
          C axi_trng.c
          h axi_trng.h
          lscript.ld
          C main.c
          C platform.c
          h platform.h
      > FLOW
        Component: trng_test_app
        Build > Run > Debug
      > Build
      > Run
      > Debug

OUTPUT X TASK:XSDB CONSOLE DEBUG CONSOLE X COM4 (FTDI) X PROBLEMS X
Opened with baud rate: 9600
Test AXI TRNG
--- Generate 5 random numbers---
Random[0] = 82 (raw: 0xA47B2C87)
Random[1] = 6 (raw: 0x20F86602)
Random[2] = 19 (raw: 0x62C4992A)
Random[3] = 19 (raw: 0x62C4992A)
Random[4] = 59 (raw: 0xC9474C65)
== Simulating Dice Rolls (1-6) ===
Dice roll 1: 2
Dice roll 2: 3
Dice roll 3: 4
Dice roll 4: 3
Dice roll 5: 6
Dice roll 6: 6
Dice roll 7: 5
Dice roll 8: 1
Dice roll 9: 4
Dice roll 10: 4
TRNG Status: 0x00000001
Health OK: 1
Busy: 0
RCT Fail: 0
APT Fail: 0
--- Test completed ---

```

Run session completed for trng_test_app.app_hw_1 [trng_test_app]

Figura 59: Ejecución 2. Prints del Serial Monitor de acuerdo con el código de main_trng.c

3.5. Proyecto AXI Pulse Generator

El proyecto AXI Pulse Generator implementa un generador de pulsos digitales configurable basado en hardware que produce señales periódicas con periodo y ancho programables. Este IP Core es esencial para aplicaciones de control de timing, generación de señales PWM, sincronización de sistemas y control de actuadores.

Descripción del IP Core

El axi_pulsegen_1_0 es un IP Core desarrollado en Verilog con las siguientes características:

- **Temporización determinista:** Basado en contador hardware, garantiza precisión independiente de la carga del procesador
- **Configuración dinámica:** Permite modificar periodo y ancho en tiempo de ejecución
- **Salida física directa:** La señal de pulso se conecta directamente a pines de I/O de la FPGA
- **Bajo overhead:** Opera de forma autónoma una vez configurado

Características principales

- **Contador de 32 bits:** Permite periodos desde 20 ns hasta ~43 segundos (@ 100 MHz)
- **Periodo programable:** Configurable desde 2 ciclos hasta $2^{32} - 1$ ciclos
- **Ancho de pulso programable:** Duty cycle ajustable desde 1 ciclo hasta periodo-1
- **Control start/stop:** Señal de habilitación para iniciar/detener la generación
- **Actualización sincronizada:** Cambios de configuración aplicados mediante señal de validación

Principio de funcionamiento

El generador opera mediante un contador de ciclos de reloj que se compara con dos umbrales configurables:

1. El contador inicia en 0 y se incrementa en cada ciclo de reloj
2. Mientras `counter < width_reg`, la salida `pulse_out = 1`
3. Cuando `counter >= width_reg`, la salida `pulse_out = 0`
4. Al alcanzar `counter >= period_reg`, el contador se reinicia
5. El ciclo se repite continuamente mientras `start` esté activa

Arquitectura del módulo

Módulo AXI4-Lite Slave:

- Implementa el protocolo de comunicación AXI4-Lite
- Gestiona 4 registros de configuración y control
- Maneja transacciones de lectura/escritura desde el procesador

Módulo Pulse Generator Core:

- Implementa el contador de 32 bits y lógica de comparación
- Registros de configuración interna: period_reg y width_reg
- Generación de la señal de salida pulse_out

Mapa de registros

| Offset | Nombre | Descripción |
|--------|--------------|--------------------------------|
| 0x00 | CONTROL | Control: start/stop, reset |
| 0x04 | CONFIG_VALID | Aplicación de configuración |
| 0x08 | PERIOD | Periodo en ciclos de reloj |
| 0x0C | WIDTH | Ancho del pulso alto en ciclos |

Registro CONTROL (0x00):

- Bit [0]: START - 1 = iniciar generación, 0 = detener
- Bit [1]: RESET - Pulso para reiniciar el generador

Registro CONFIG_VALID (0x04):

- Bit [0]: VALID - Pulso para aplicar nueva configuración

Registro PERIOD (0x08):

- Periodo en ciclos de reloj (mínimo: 2 ciclos)
- Valor por defecto: 100,000,000 (1 segundo @ 100 MHz)

Registro WIDTH (0x0C):

- Ancho del pulso alto en ciclos (rango: 1 a PERIOD-1)
- Valor por defecto: 50,000,000 (0.5 segundos - 50 % duty cycle @ 100 MHz)

Driver de software

Funciones principales:

- PulseGen_Start(), PulseGen_Stop(): Control de generación
- PulseGen_SetPeriod(), PulseGen_SetWidth(): Configuración de parámetros

- `PulseGen_ApplyConfiguration()`: Aplica los cambios de configuración
- `PulseGen_GetPeriod()`, `PulseGen_GetWidth()`: Lectura de configuración actual

Aplicaciones típicas

- **Control PWM**: Modulación de ancho de pulso para motores, LEDs
- **Generación de clock personalizado**: Relojes derivados con frecuencias específicas
- **Sincronización temporal**: Triggers periódicos para captura de datos
- **Control de actuadores**: Señales para servomotores, relés, solenoides
- **Generación de interrupciones periódicas**: Base de tiempo para sistemas en tiempo real

Consideraciones de diseño

- **Constraints de pines**: La señal `pulse_out` debe conectarse a un pin de I/O mediante archivo `.xdc`
- **Actualización sincronizada**: Siempre llamar a `PulseGen_ApplyConfiguration()` después de modificar parámetros
- **Validación de parámetros**:
 - $PERIOD \geq 2$ (mínimo 20 ns @ 100 MHz)
 - $1 \leq WIDTH < PERIOD$
- **Rangos prácticos**:
 - Frecuencias altas: hasta 50 MHz (periodo = 2 ciclos)
 - Frecuencias bajas: hasta 0.023 Hz (periodo \approx 43 segundos)

Las siguientes subsecciones detallan el proceso de creación de un proyecto en Vivado y Vitis utilizando este IP Core, desde la configuración inicial hasta la ejecución de aplicaciones que demuestran generación de pulsos con diferentes frecuencias y duty cycles.

3.5.1. Crear proyecto Vivado (vivado-axi-pulsegen)

Ejecutar los **Pasos** tal y como se describe en 3.2.1. No obstante, para el **Paso 1** se debe setear:

```
set ::user_project_name "vivado-axi-pulsegen"
```

El **Paso 2** es exactamente igual. En el **Paso 3** se debe buscar el módulo `axi_pulsegen_v1_0`. Dicho módulo tiene un puerto de entrada (`pulsegen_reset`) y un puerto de salida

(pulse_out), que tienen que ser marcados como Make External. En la Figura 60 se observa el diseño actual.

En el **Paso 4**, el archivo que se debe seleccionar es pulsegen.xdc. El **Paso 5** es exactamente igual.



Figura 60: Block Design después de agregar el módulo axi_pulsegen_v1.0

3.5.2. Crear proyecto Vitis (vitis-axi-pulsegen)

Ejecutar los **Pasos** tal y como se describe en 3.2.2. El **Paso 1** es exactamente igual. En el **Paso 2** el workspace a crear y seleccionar dentro de mbv-axi-slave-ip-repo es vitis-axi-pulsegen.

En el **Paso 3**, el Component name de la aplicación debe ser pulsegen_test_app. Con el **Paso 4** se deben importar los archivos axi_pulsegen.h y axi_pulsegen.c.

En el **Paso 5** se debe copiar y pegar el contenido del archivo main_pulsegen.c en main.c

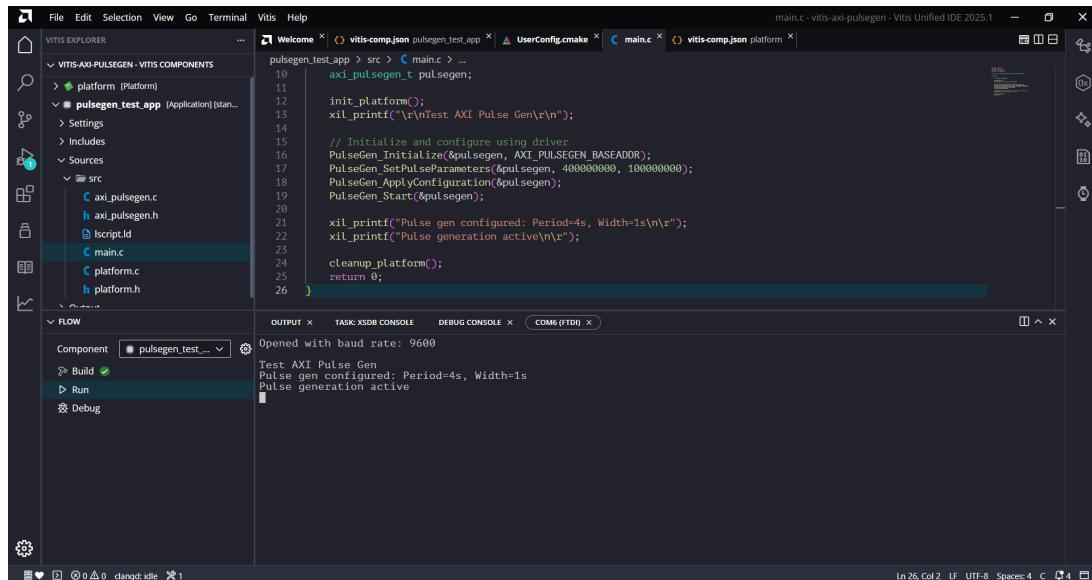
Este archivo contiene un programa de prueba que demuestra el uso del IP core AXI Pulse Generator implementado en hardware. El programa realiza las siguientes operaciones:

1. **Inicialización:** Inicializa la plataforma y configura una instancia del generador de pulsos con su dirección base en el bus AXI.
2. **Configuración de parámetros:** Establece los parámetros del pulso:
 - Período del pulso: 4 segundos (400000000 ciclos de reloj)
 - Ancho del pulso: 1 segundo (100000000 ciclos de reloj)
3. **Aplicación de configuración:** Aplica la configuración al hardware mediante el registro de control correspondiente.
4. **Inicio de generación:** Inicia la generación continua de pulsos con los parámetros configurados.

En la Figura 61 se muestran los prints en consola tras la ejecución de la aplicación donde se puede apreciar que el módulo quedó configurado con un periodo de 4s y un ancho de 1s.

En la Figura 62 se muestra el funcionamiento del sistema en la FPGA, actualmente el led verde de la derecha enciende por un 1s y permanece apagado por 3s. Al presionar

el botón A marcado en rojo, se vuelve a ejecutar la aplicación desde cero, quedando el módulo con la misma configuración (periodo=4s y ancho=1s). Si se presiona el botón B marcado en rojo, sin necesidad de que exista comunicación entre el procesador MicroBlaze V y el módulo AXI Pulse Generator por medio de AXI, el módulo se resetea a sus valores por defecto (periodo=1s y ancho=0.5) y se puede apreciar que el led verde parpadea más rápido.



```

File Edit Selection View Go Terminal Vitis Help
VITIS EXPLORER
VITIS-AXI-PULSEGEN - VITIS COMPONENTS
> platform [Platform]
-> pulsegen_test_app [Application] [stan...
  > Settings
  > Includes
  > Sources
    > src
      C axi_pulsegen.c
      h axi_pulsegen.h
      lscript.ld
      C main.c
      C platform.c
      h platform.h
  < FLOW
  Component: pulsegen_test...
  >> Build
  > Run
  & Debug
  OUTPUT X TASK: XSDM CONSOLE DEBUG CONSOLE X COM6 (FTDI) X
  Opened with baud rate: 9600
  Test AXI Pulse Gen
  Pulse gen configured: Period=4s, Width=1s
  Pulse generation active
  Line 26, Col 2 LF UTF-8 Spaces: 4 C 4

```

Figura 61: Prints del Serial Monitor de acuerdo con el código de `main_pulsegen.c`

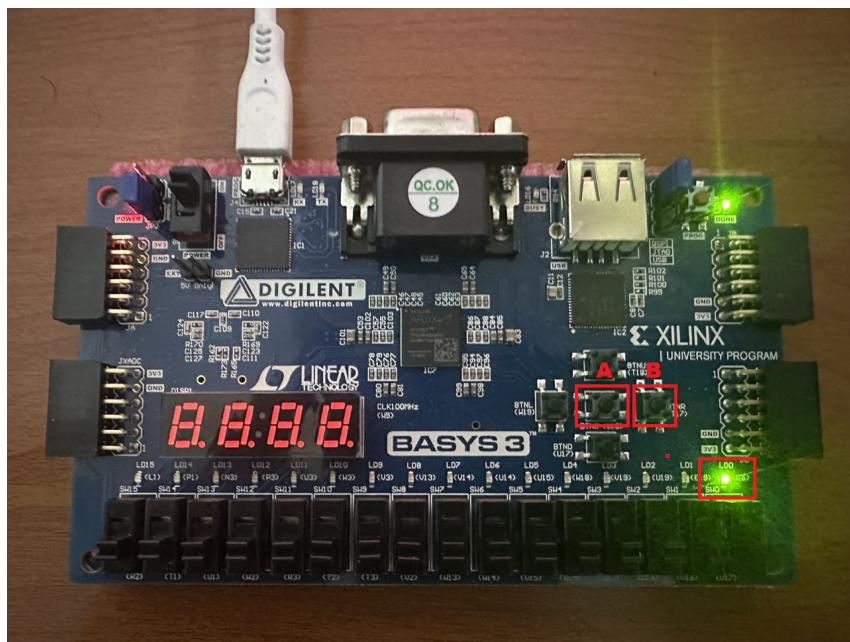


Figura 62: Funcionamiento del sistema en la FPGA de acuerdo con el código de `main_pulsegen.c`

3.6. Proyecto AXI Perceptron

El proyecto AXI Perceptron implementa en hardware un perceptrón binario de 2 entradas con capacidad de entrenamiento autónomo mediante el algoritmo clásico del perceptrón. Este IP Core demuestra conceptos fundamentales de machine learning en FPGA.

Descripción del IP Core

El axi_perceptron_1_0 implementa el modelo matemático del perceptrón de Rosenblatt (1957), incluyendo tanto inferencia como entrenamiento en hardware dedicado.

Características principales

- **Perceptrón binario de 2 entradas:** Dos entradas booleanas y una salida binaria
- **Entrenamiento en hardware:** Algoritmo de aprendizaje implementado en lógica digital
- **Aritmética Q4.4:** Pesos y bias representados con 4 bits enteros y 4 bits fraccionarios
- **Funciones lógicas predefinidas:** AND, OR, NAND, NOR, XOR, XNOR
- **Detección de convergencia:** Hardware detiene entrenamiento cuando todos los patrones se clasifican correctamente

Fundamentos teóricos

Función de activación:

$$y = \text{step}(s) = \begin{cases} 1 & \text{si } s > 0 \\ 0 & \text{si } s \leq 0 \end{cases}$$

Suma ponderada:

$$s = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

Algoritmo de entrenamiento: Para cada patrón (x_1, x_2, t) :

1. Calcular predicción: $\hat{y} = \text{step}(w_1x_1 + w_2x_2 + b)$
2. Calcular error: $e = t - \hat{y}$
3. Si $e \neq 0$, actualizar: $w_i \leftarrow w_i + \eta \cdot e \cdot x_i, b \leftarrow b + \eta \cdot e$

Arquitectura del módulo

Módulo AXI4-Lite Slave:

- Protocolo de comunicación AXI4-Lite
- Gestión de 8 registros de configuración y control

Módulo Binary Perceptron Core:

- Banco de pesos: w_1, w_2, b (registros Q4.4)
- Máquina de estados para entrenamiento
- Lógica aritmética Q4.4

Mapa de registros

| Offset | Nombre | Descripción |
|--------|-----------|------------------------------------|
| 0x00 | CONTROL | Control y targets de entrenamiento |
| 0x04 | MAXEPOCHS | Número máximo de épocas |
| 0x08 | ETA | Tasa de aprendizaje Q4.4 |
| 0x0C | INIT | Inicialización de pesos |
| 0x10 | STATUS | Estado de convergencia (RO) |
| 0x14 | W12 | Pesos aprendidos (RO) |
| 0x18 | BIAS | Bias aprendido (RO) |
| 0x1C | RESULT | Suma y predicción actuales (RO) |

Targets predefinidos:

- 0x8: AND, 0xE: OR, 0x7: NAND, 0x1: NOR
- 0x6: XOR, 0x9: XNOR (no convergen)

Driver de software

Funciones principales:

- Perceptron_StartTraining(): Inicia entrenamiento
- Perceptron_SetTrainingParams(): Configura parámetros
- Perceptron_ReadWeights(): Obtiene pesos aprendidos
- Perceptron_ReadStatus(): Lee estado de convergencia

Aplicaciones

- **Enseñanza de machine learning:** Visualización física del aprendizaje neuronal
- **Demostración de limitaciones:** Problemas linealmente separables vs no separables
- **Clasificación binaria hardware:** Decisiones en tiempo real
- **Laboratorios interactivos:** Entrenamiento en hardware real

Consideraciones de diseño

- **Separabilidad lineal:** Solo AND, OR, NAND, NOR convergen
- **Tasa de aprendizaje:** Valor por defecto 0.5 generalmente óptimo
- **Convergencia:** Funciones separables convergen en 5-20 épocas
- **Entradas físicas:** Conectables a switches y LEDs para demostración en tiempo real

Las siguientes subsecciones detallan el proceso de creación de un proyecto en Vivado y Vitis utilizando este IP Core, desde la configuración inicial hasta la ejecución de aplicaciones que demuestran el entrenamiento de diferentes funciones lógicas y la exploración de los límites del aprendizaje lineal.

3.6.1. Crear proyecto Vivado (vivado-axi-perceptron)

Ejecutar los **Pasos** tal y como se describe en 3.2.1. No obstante, para el **Paso 1** se debe setear:

```
set ::user_project_name "vivado-axi-perceptron"
```

El **Paso 2** es exactamente igual. En el **Paso 3** se debe buscar el módulo `axi_perceptron_v1.0`. Dicho módulo tiene dos puertos de entrada (`x1` y `x2`) y un puerto de salida (`y`), que tienen que ser marcados como `Make External`. En la Figura 63 se observa el diseño actual.

En el **Paso 4**, el archivo que se debe seleccionar es `perceptron.xdc`. El **Paso 5** es exactamente igual.

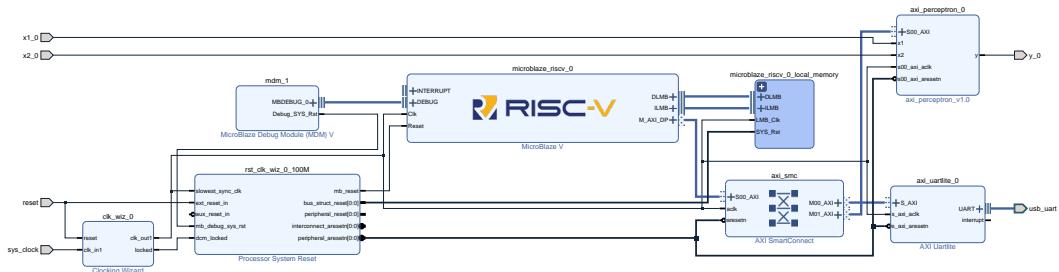


Figura 63: Block Design después de agregar el módulo `axi_perceptron_v1.0`

3.6.2. Crear proyecto Vitis (vitis-axi-perceptron)

Ejecutar los **Pasos** tal y como se describe en 3.2.2. El **Paso 1** es exactamente igual. En el **Paso 2** el workspace a crear y seleccionar dentro de `mbv-axi-slave-ip-repo` es `vitis-axi-perceptron`.

En el **Paso 3**, el `Component name` de la aplicación debe ser `perceptron_test_app`. Con el **Paso 4** se deben importar los archivos `axi_perceptron.h` y `axi_perceptron.c`.

En el **Paso 5** se debe copiar y pegar el contenido del archivo `main_perceptron.c` en `main.c`

Este archivo contiene un programa de prueba que demuestra el uso del IP core AXI Perceptron implementado en hardware. El programa realiza las siguientes operaciones:

1. **Inicialización:** Inicializa la plataforma y configura una instancia del perceptrón con su dirección base en el bus AXI.
2. **Configuración de parámetros de entrenamiento:** Establece los parámetros necesarios para el entrenamiento:
 - Máximo número de épocas (64 por defecto)
 - Tasa de aprendizaje (η) en formato Q4.4
 - Pesos iniciales y bias (inicializados en cero)
3. **Entrenamiento:** Entrena el perceptrón para aprender una función lógica booleana (por defecto AND, pero puede configurarse para OR, NAND, NOR, XOR, XNOR o funciones personalizadas).
4. **Visualización de resultados:** Muestra en consola:
 - El estado del entrenamiento (incluyendo si converge)
 - Los pesos finales (w_1, w_2) y el bias (b) en formato Q4.4
 - El nombre de la función lógica entrenada

En la Figura 64 se muestran los prints en consola tras la ejecución de la aplicación para la función AND, se puede observar que la función converge ya que es linealmente separable. En la Figura 65 se muestra el funcionamiento del sistema en la FPGA, tras el entrenamiento se prueba que con $x_1=1$ y $x_2=1$, la salida $y=1$, por eso enciende el led verde.

En la Figura 66 se muestran los prints en consola tras la ejecución de la aplicación para la función NOR, se puede observar que la función converge ya que también es linealmente separable. En la Figura 67 se muestra el funcionamiento del sistema en la FPGA, tras el entrenamiento se prueba que con $x_1=0$ y $x_2=0$, la salida $y=1$, por eso enciende el led verde.

Finalmente en la Figura 68 se muestran los prints en consola tras la ejecución de la aplicación para la función XOR, se puede observar que la función no converge ya que la misma no es linealmente separable. Haciendo pruebas en la FPGA se observa que con diferentes combinaciones de x_1 y x_2 , la salida y es errónea.

The screenshot shows the Vitis Unified IDE interface. In the top right, there's a code editor window with the file `main.c` open. The code is a C program for an AXI Perceptron. Below the code editor is a terminal window titled "COMS(FTDI)" showing the serial monitor output. The output text is as follows:

```

Welcome > C main.c > vitis-comp.json perception_test_app > vitis-comp.json platform >
perception_test_app > src > C main.c > ...
43 int main(void) {
44     init_platform();
45     xil_printf("Test AXI Perception\r\n");
46
47     // Perceptron instance
48     axi_perceptron_t perceptron;
49
50     // Target logic function to train
51     u8 targets = PERCEPTRON_TARGETS_AND;
52
53     // Maximum number of training epochs
54     u16 max_epochs = 64;
55
56     // Learning rate (eta) in Q4.4 format
56     s8 eta_q4_4 = (S8)18;
Test AXI Perception
Initializing Perceptron...
Configuring training parameters...
Max epochs: 64
Learning rate (eta): 0.5000
Loading initial weights and bias...
Status: 0x00000000 (conv=0)
w1=0.0000, w2=0.0000, b=0.0000
Starting training...
Training: AND function...
Training completed.
Status: 0x00000001 (conv=1)
w1=1.0000, w2=0.5000, b=-1.0000
--- Test completed ---

```

The bottom left of the IDE shows a flowchart component labeled "perceptron_test_app". The "Run" option is selected in the dropdown menu.

Figura 64: Prints del Serial Monitor para la función AND de acuerdo con el código de `main_perceptron`

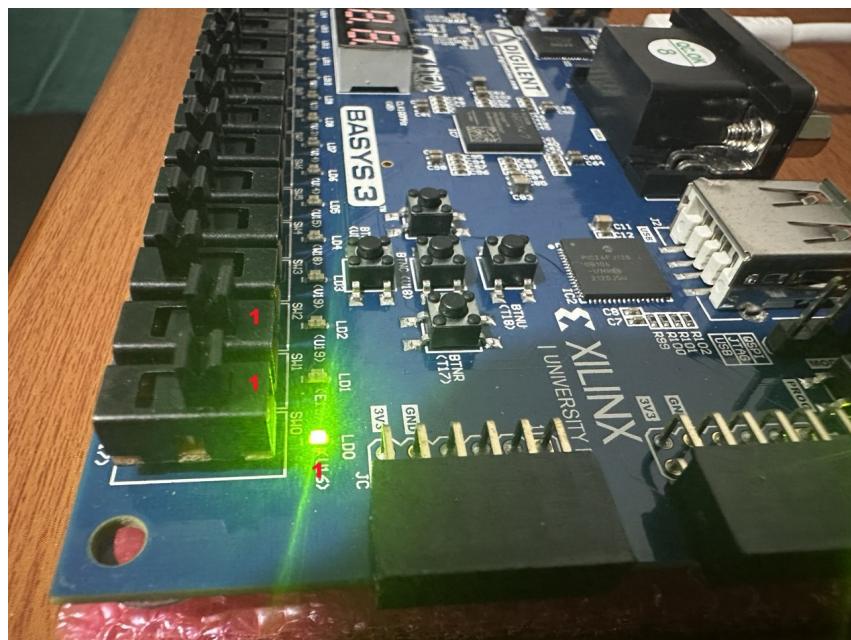


Figura 65: Funcionamiento del sistema en la FPGA para la función AND de acuerdo con el código de `main_perceptron`

```

43 int main(void) {
44     init_platform();
45     xil_printf("\r\nTest AXI Perceptron\r\n");
46
47     // Perceptron instance
48     axi_perceptron_t perceptron;
49
50     // Target logic function to train
51     u8 targets = PERCEPTRON_TARGETS_NOR;
52
53     // Maximum number of training epochs
54     u16 max_epochs = 64;
55     // Learning rate in Q4.4 format
56     s8 eta_q4 = (s8)8;

```

OUTPUT X TASK_XSDB CONSOLE DEBUG CONSOLE X COM6 (FTDI) X PROBLEMS X

Opened with baud rate: 9600

Test AXI Perceptron

Initializing Perceptron...

Configuring training parameters...

Max epochs: 64

Learning rate (eta): 0.5000

Loading initial weights and bias...

Status: 0x00000000 (conv=0)

w1=0.0000, w2=0.0000, b=0.0000

Starting training...

Training: NOR function...

Training completed.

Status: 0x00000001 (conv=1)

w1=-0.5000, w2=-0.5000, b=0.5000

--- Test completed ---

Figura 66: Prints del Serial Monitor para la función NOR de acuerdo con el código de `main_perceptron`

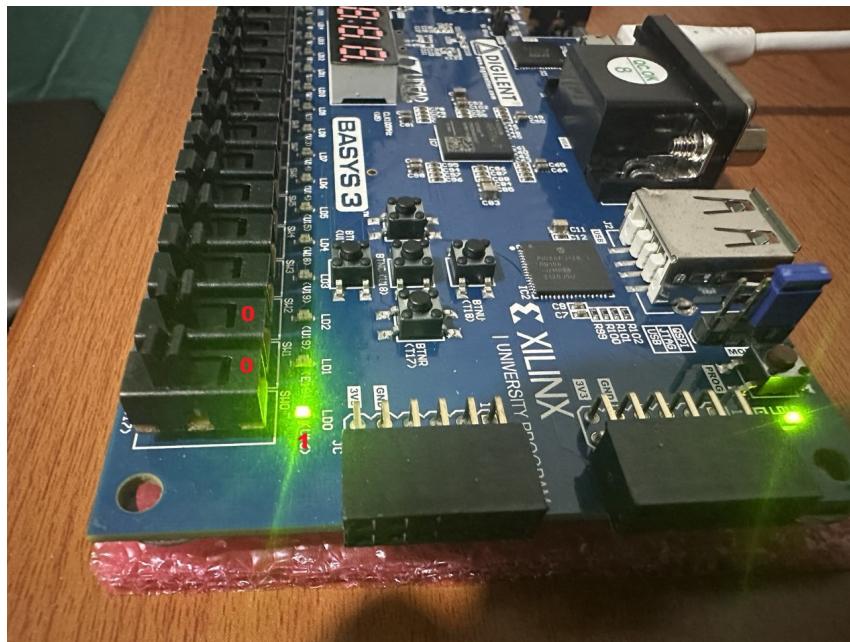


Figura 67: Funcionamiento del sistema en la FPGA para la función NOR de acuerdo con el código de `main_perceptron`

```

43 int main(void) {
44     init_platform();
45     xil_printf("Test AXI Perception\r\n");
46
47     // Perceptron instance
48     axi_perceptron_t perceptron;
49
50     // Target logic function to train
51     u8 targets = PERCEPTRON_TARGETS_XOR;
52
53     // Maximum number of training epochs
54     u16 max_epochs = 64;
55     // Learning rate in Q4.4 format
56     s8 eta_q4 = (s8)18;

```

Test AXI Perception
Initializing Perceptron...
Configuring training parameters...
Max epochs: 64
Learning rate (eta): 0.5000
Loading initial weights and bias...
Status: 0x00000000 (conv=0)
w1=0.0000, w2=0.0000, b=0.0000
Starting training...
Training: XOR function...
Training completed.
Status: 0x00000000 (conv=0)
w1=-0.5000, w2=0.0000, b=0.5000
--- Test completed ---

Figura 68: Prints del Serial Monitor para la función XOR de acuerdo con el código de main_perceptron

3.7. Proyecto AXI Secure Authentication

El proyecto AXI Secure Authentication es un sistema integrado de autenticación de contraseñas que combina dos IP Cores desarrollados (AXI TRNG y AXI 7-Segment Display) con algoritmos criptográficos en software (SHA-256) para demostrar un caso de uso real de seguridad en sistemas embebidos. Este proyecto representa la culminación del manual, mostrando cómo los IP Cores individuales pueden integrarse para crear una aplicación funcional y práctica que implementa mejores prácticas de seguridad moderna.

Descripción del sistema

El sistema de autenticación implementa un flujo completo de gestión segura de contraseñas que incluye:

- **Configuración inicial:** Registro de contraseña maestra del usuario
- **Generación de salt aleatorio:** Utiliza el TRNG para producir salt criptográfica
- **Hashing seguro:** Aplica SHA-256 con salt para almacenar contraseñas
- **Autenticación:** Verifica credenciales mediante comparación constant-time
- **Visualización de estados:** Usa display de 7 segmentos para feedback visual
- **Protección contra ataques:** Límite de intentos y delays anti fuerza bruta

Arquitectura del sistema

El proyecto integra tres componentes principales:

1. IP Cores de hardware:

- **AXI TRNG (axi_trng_v1.0)**: Genera números verdaderamente aleatorios para la salt criptográfica de 128 bits (16 bytes)
- **AXI 7-Segment Display (axi_7seg_v1.0)**: Proporciona feedback visual del estado del sistema
- **AXI UART Lite**: Interfaz para entrada/salida de contraseñas y mensajes

2. Algoritmos criptográficos en software:

- **SHA-256**: Implementación completa del algoritmo de hash criptográfico estándar FIPS 180-4
- **Salt generation**: Función que utiliza TRNG para producir 16 bytes de entropía
- **Password hashing**: Combina password + salt antes de aplicar SHA-256

3. Lógica de aplicación:

- Máquina de estados para configuración, autenticación y gestión de sesión
- Sistema de intentos limitados (máximo 3 intentos)
- Comparación constant-time para prevenir timing attacks
- Delays deliberados entre intentos para mitigar fuerza bruta

Flujo del sistema

Fase 1: Configuración inicial (primera ejecución)

1. El sistema detecta que no hay contraseña configurada
2. Display muestra 1111 (modo INIT)
3. Usuario es solicitado crear una contraseña maestra
4. TRNG genera 16 bytes aleatorios como salt
5. Se calcula: hash = SHA-256(password || salt)
6. Hash y salt se almacenan en estructura PasswordStore
7. Display confirma configuración exitosa

Fase 2: Autenticación (ejecuciones subsecuentes)

1. Display muestra 2222 (modo WAITING)
2. Usuario ingresa contraseña vía UART
3. Sistema calcula: hash_test = SHA-256(input || salt_stored)
4. Comparación constant-time: result = $\bigoplus_{i=0}^{31} (\text{hash_test}[i] \oplus \text{hash_stored}[i])$
5. Si result = 0: Autenticación exitosa

- Display muestra 8888 (AUTHENTICATED)
 - Sistema muestra perfil de usuario simulado
 - Sesión activa hasta logout manual
6. Si result $\neq 0$: Autenticación fallida
- Display muestra 0000 (DENIED)
 - Contador de intentos se incrementa
 - Delay de 2 segundos antes de reintentar
7. Tras 3 intentos fallidos: Sistema bloqueado
- Display parpadea 9999 (ERROR)
 - Requiere reset físico del dispositivo

Características de seguridad implementadas

1. Salting criptográfico:

- Salt de 16 bytes generado por TRNG (entropía física real)
- Salt único previene ataques de rainbow tables
- Concatenación password + salt antes de hashing

2. Hashing con SHA-256:

- Algoritmo estándar de la industria (FIPS 180-4)
- Hash de 256 bits (32 bytes) de salida
- Proceso unidireccional (no reversible)
- Resistente a colisiones (prácticamente imposible encontrar dos inputs con mismo hash)

3. Comparación constant-time:

Para prevenir timing attacks, la verificación no usa comparación simple:

```
// INSEGUR0 (timing attack vulnerable):
if (memcmp(hash1, hash2, 32) == 0) { ... }

// SEGUR0 (constant-time):
u8 result = 0;
for (int i = 0; i < 32; i++) {
    result |= (hash1[i] ^ hash2[i]);
}
if (result == 0) { ... }
```

La versión segura siempre ejecuta las 32 iteraciones, independientemente de dónde difieren los hashes, eliminando la fuga de información temporal.

4. Mitigación de fuerza bruta:

- Límite de 3 intentos antes de bloqueo
- Delay de 500 ms tras cada intento (legítimo o no)
- Delay adicional de 2 s tras fallo
- Sistema bloqueado requiere intervención física (reset)

5. Feedback visual seguro:

- Display no revela información sobre la contraseña
- Estados claros pero informativamente mínimos
- Indicador de bloqueo visible (parpadeo de 9999)

Implementación de SHA-256

La implementación sigue el estándar FIPS 180-4:

Constantes:

- 64 constantes $K[i]$: Primeros 32 bits de las partes fraccionarias de las raíces cúbicas de los primeros 64 primos
- 8 valores iniciales de hash H_0 a H_7 : Primeros 32 bits de las partes fraccionarias de las raíces cuadradas de los primeros 8 primos

Funciones lógicas:

- $\text{Ch}(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$
- $\text{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
- $\Sigma_0(x) = \text{ROTR}^2(x) \oplus \text{ROTR}^{13}(x) \oplus \text{ROTR}^{22}(x)$
- $\Sigma_1(x) = \text{ROTR}^6(x) \oplus \text{ROTR}^{11}(x) \oplus \text{ROTR}^{25}(x)$
- $\sigma_0(x) = \text{ROTR}^7(x) \oplus \text{ROTR}^{18}(x) \oplus \text{SHR}^3(x)$
- $\sigma_1(x) = \text{ROTR}^{17}(x) \oplus \text{ROTR}^{19}(x) \oplus \text{SHR}^{10}(x)$

Algoritmo:

1. Padding: Agregar bit 1, seguido de ceros, seguido de longitud del mensaje en 64 bits
2. Dividir mensaje en bloques de 512 bits
3. Para cada bloque:
 - Expandir 16 palabras de 32 bits a 64 palabras (message schedule)

- Inicializar variables de trabajo a, b, c, d, e, f, g, h
 - 64 rondas de compresión aplicando funciones lógicas
 - Actualizar valores de hash acumulativos
4. Concatenar $H_0 \parallel H_1 \parallel \dots \parallel H_7$ como hash final de 256 bits

Códigos de display

El sistema utiliza el display de 7 segmentos para indicar estados:

| Estado | Código | Significado |
|---------------|--------|---------------------------------|
| INIT | 1111 | Inicialización del sistema |
| WAITING | 2222 | Esperando entrada de contraseña |
| AUTHENTICATED | 8888 | Autenticación exitosa |
| DENIED | 0000 | Acceso denegado |
| ERROR | 9999 | Error o bloqueo del sistema |

Estructura de datos

PasswordStore:

```
typedef struct {
    u8 password_hash[32]; // SHA-256 hash
    u8 salt[16];          // 128-bit random salt
    u8 is_configured;     // Setup flag
} PasswordStore;
```

SHA256_CTX:

```
typedef struct {
    u8 data[64];        // Buffer de bloque actual
    u32 datalen;        // Bytes en buffer
    u64 bitlen;         // Total bits procesados
    u32 state[8];       // Hash acumulativo (H0-H7)
} SHA256_CTX;
```

Lecciones educativas

Este proyecto integrador enseña conceptos clave de seguridad en sistemas embebidos:

1. **Importancia de entropía de calidad:** TRNG vs PRNG para aplicaciones criptográficas
2. **Salting contra rainbow tables:** Por qué salt aleatoria es esencial
3. **Hashing unidireccional:** Almacenar hashes, nunca contraseñas en texto plano

4. **Timing attacks:** Cómo comparaciones ingenuas pueden filtrar información
5. **Mitigación de fuerza bruta:** Delays y límites de intentos
6. **Integración hardware-software:** Combinar aceleradores HW con algoritmos SW
7. **Trade-offs de seguridad:** Balance entre usabilidad y protección

Las siguientes subsecciones detallan el proceso de creación de este proyecto integrador en Vivado y Vitis, mostrando cómo agregar múltiples IP Cores al diseño base y cómo desarrollar la aplicación completa de autenticación segura.

3.7.1. Crear proyecto Vivado (vivado-axi-secure-auth)

Antes que nada se debe modificar el archivo `mbv_axi_base.tcl` para aumentar el tamaño de la memoria local del procesador MicroBlaze V, ya que la aplicación que va a ejecutar requiere de más espacio. Para ello se debe cambiar el range de la línea 458 y 459 de `0x00008000` a `0x00020000`. Guardar dichos cambios.

Ejecutar los **Pasos** tal y como se describe en 3.2.1. No obstante, para el **Paso 1** se debe setear:

```
set ::user_project_name "vivado-axi-secure-auth"
```

El **Paso 2** es exactamente igual. En el **Paso 3** se deben buscar los módulos `axi_trng_v1.0` y `axi_7seg_v1.0`. El segundo módulo tiene dos puertos de salida (`an_out[3:0]` y `seg_out[7:0]`) que tienen que ser marcados como `Make External`. En la Figura 69 se observa el diseño actual del sistema.

En el **Paso 4**, el archivo que se debe seleccionar es `secure_auth.xdc`. El **Paso 5** es exactamente igual.

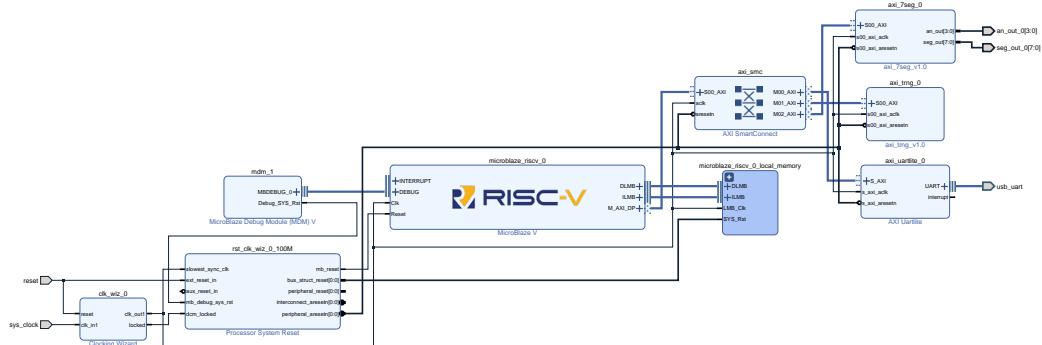


Figura 69: Block Design después de agregar los módulos `axi_trng_v1.0` y `axi_7seg_v1.0`

3.7.2. Crear proyecto Vitis (vitis-axi-secure-auth)

Ejecutar los **Pasos** tal y como se describe en 3.2.2. El **Paso 1** es exactamente igual. En el **Paso 2** el workspace a crear y seleccionar dentro de `mbv-axi-slave-ip-repo` es

vitis-axi-secure-auth.

En el **Paso 3**, el Component name de la aplicación debe ser secure_auth_app. Con el **Paso 4** se deben importar los archivos axi_trng.h, axi_trng.c, axi_7seg.h y axi_7seg.c.

En el **Paso 5** se debe copiar y pegar el contenido del archivo main_secure_auth.c en main.c

Esta aplicación representa el proyecto integrador más completo del repositorio, combinando múltiples IP Cores (AXI TRNG y AXI 7-Segment Display) con algoritmos criptográficos en software (SHA-256) para implementar un sistema de autenticación de contraseñas que demuestra mejores prácticas de seguridad moderna en sistemas embebidos.

Flujo de ejecución

Fase 1: Inicialización del sistema (arranque)

1. Inicializa los drivers de los IP Cores:
 - Display de 7 segmentos: Seg7_Initialize()
 - TRNG: Trng_Initialize()
2. Configura el TRNG para generación de entropía:
 - Rango completo: Trng_SetRange(0, 0xFFFFFFFF)
 - Sample divider rápido: Trng_SetSampleDivider(100) para reducir latencia
 - Habilita el generador: Trng_Enable()
3. Verifica salud del TRNG:
 - Espera 10 ms para estabilización de ring oscillators
 - Comprueba Trng_IsHealthy()
 - Si falla: Muestra 9999 en display y aborta ejecución
4. Muestra banner de bienvenida:
 - Display: 1111 (código INIT)
 - Terminal: Banner con características de seguridad implementadas
 - Espera 2 segundos para que el usuario lea la información

Fase 2: Configuración inicial (primera ejecución)

Si no existe contraseña configurada (g_password_store.is_configured == 0):

1. Detecta ausencia de contraseña y solicita configuración inicial
2. Muestra requisitos de contraseña:

- Longitud máxima: 32 caracteres
 - Será protegida con SHA-256 y salt aleatorio
3. Lee contraseña del usuario vía UART:
- Función `read_password()`: Lee caracteres uno a uno
 - Soporta backspace para corrección
 - Acepta caracteres ASCII imprimibles (32-126)
 - Termina con Enter
4. Valida entrada:
- Si está vacía: Error y terminación
5. Ejecuta `store_password()`:
- **Generación de salt:** Llama `generate_salt()` que usa TRNG para producir 16 bytes aleatorios (128 bits de entropía)
 - **Hashing con salt:** Calcula $\text{hash} = \text{SHA-256}(\text{password} \parallel \text{salt})$ usando `hash_password_with_salt()`
 - **Almacenamiento:** Guarda hash y salt en `g_password_store`
 - **Debug:** Imprime salt y hash en hexadecimal para verificación
6. Confirma configuración exitosa y espera 2 segundos

Fase 3: Bucle de autenticación (ejecuciones subsecuentes)

El sistema entra en un bucle con límite de 3 intentos:

1. **Estado de espera:**
 - Display: 2222 (código WAITING)
 - Muestra contador de intentos: "Attempt [n] of 3"
2. **Entrada de contraseña:**
 - Lee contraseña con `read_password()`
 - Delay anti fuerza bruta: 500 ms obligatorio
3. **Verificación con `verify_password()`:**
 - Calcula: $\text{hash_test} = \text{SHA-256}(\text{input} \parallel \text{salt_stored})$
 - **Comparación constant-time:**

```
u8 result = 0;
for (int i = 0; i < 32; i++) {
    result |= (hash_test[i] ^ hash_stored[i]);
}
return (result == 0);
```

Esta implementación previene timing attacks al ejecutar siempre las 32 iteraciones

4. Caso: Autenticación exitosa:

- Display: 8888 (código AUTHENTICATED)
- Imprime mensaje de bienvenida
- Llama `display_user_profile()`:
 - Muestra perfil simulado con nivel de acceso
 - Animación del display
- Sesión activa: Espera tecla para logout
- Al logout: Vuelve a 2222 y sale del bucle

5. Caso: Autenticación fallida:

- Incrementa contador de intentos
- Display: 0000 (código DENIED)
- Imprime mensaje de error
- Delay punitivo: 2 segundos antes de permitir reintento
- Si alcanza 3 intentos fallidos:
 - Imprime mensaje de bloqueo
 - Display parpadea 9999 cinco veces (300 ms on/off)
 - Queda bloqueado mostrando 9999
 - Requiere reset físico del dispositivo para recuperar

Fase 4: Finalización

- Deshabilita el TRNG: `Trng_Disable()` para ahorrar energía
- Limpia el display: `Seg7_ClearDisplay()`
- Muestra mensaje de despedida
- Libera recursos de la plataforma

En las Figuras 70 y 71 se muestran los prints generados tras ejecutar la aplicación.

```

[OUTPUT X COM4 (FTDI) X TASK_XSDB CONSOLE DEBUG CONSOLE X PROBLEMS X
Opened with baud rate: 9600
=====
SECURE PASSWORD AUTHENTICATION SYSTEM
=====
Hardware Security Features:
- True Random Number Generator (TRNG)
- SHA-256 Hashing
- Salt-based Protection
=====

[SETUP] No password configured. Initial setup required.
[INFO] Please create a secure password.
[INFO] Password requirements:
- Maximum length: 20 characters
- Will be hashed using SHA-256
- Random salt will be generated using TRNG
Enter password: MySecurepassword123
[SECURITY] Generating random salt...
[SECURITY] Hashing password with SHA-256...
[SECURITY] Password stored securely.
[DEBUG] Hash (hex): A9E83D000F9222AD0C2C22BA9180356B00BF0B93D09E79AE95D8A9ABFC0C810D1
[INFO] You can now use this password to authenticate.

=====
AUTHENTICATION
=====
Attempt 1 of 3
Enter password: MySecurepassword123
[SECURITY] Computing password hash...
[DEBUG] Computed hash: A9E83D000F9222AD0C2C22BA9180356B00BF0B93D09E79AE95D8A9ABFC0C810D1
[SUCCESS] Authentication successful!
[ACCESS] Welcome back, user!

=====
SECURE USER PROFILE ACCESS
=====
User: admin
Access Level: Administrator
Security Level: HIGH
=====

[SESSION] Session active. Press any key to logout...
[SECURITY] Logging out...
=====

SYSTEM SHUTDOWN
=====
[INFO] Thank you for using the secure authentication system.

```

Figura 70: Ejecución 1. Prints del Serial Monitor de acuerdo con el código de main_secure_auth

```

[OUTPUT X COM4 (FTDI) X TASK_XSDB CONSOLE DEBUG CONSOLE X PROBLEMS X
Opened with baud rate: 9600
=====
SECURE PASSWORD AUTHENTICATION SYSTEM
=====
Hardware Security Features:
- True Random Number Generator (TRNG)
- SHA-256 Hashing
- Salt-based Protection
=====

[SETUP] No password configured. Initial setup required.
[INFO] Please create a secure password.
[INFO] Password requirements:
- Maximum length: 20 characters
- Will be hashed using SHA-256
- Random salt will be generated using TRNG
Enter password: MySecurepassword123
[SECURITY] Generating random salt...
[SECURITY] Hashing password with SHA-256...
[SECURITY] Password stored securely.
[DEBUG] Hash (hex): ECF93EE450515A0FF7D51F84925324CF660839A4F59F8D2A91290EA578961D0
[INFO] You can now use this password to authenticate.

=====
AUTHENTICATION
=====
Attempt 1 of 3
Enter password: Password
[SECURITY] Computing password hash...
[DEBUG] Computed hash: ECF93EE450515A0FF7D51F84925324CF660839A4F59F8D2A91290EA578961D0
[DENIED] Authentication failed!
[SECURITY] Invalid password.

=====
SECURE USER PROFILE ACCESS
=====
User: admin
Access Level: Administrator
Security Level: HIGH
=====

[SESSION] Session active. Press any key to logout...
[SECURITY] Logging out...
=====

SYSTEM SHUTDOWN
=====
[INFO] Thank you for using the secure authentication system.

```

Figura 71: Ejecución 2. Prints del Serial Monitor de acuerdo con el código de main_secure_auth

4. Ejecución de Proyectos Existentes

Para no tener que crear desde cero los proyectos descritos en la sección 3, existe la alternativa de clonar el repositorio desde la branch develop en lugar de la branch main. Para ello:

1. Abra una terminal o línea de comandos.
2. Navegue al directorio donde desea guardar el proyecto, por ejemplo:

```
cd C:/Users/[usuario]/Desktop
```

3. Ejecute el comando de clonación especificando la branch develop:

```
git clone --branch develop  
https://github.com/MarinGE23/mbv-axi-slave-ip-repo.git
```

4. Acceda al directorio del proyecto:

```
cd mbv-axi-slave-ip-repo
```

Estructura del repositorio

Una vez descargado usando git clone y especificando la branch develop, el repositorio contiene la siguiente estructura de directorios:

- **ip_repo/**: Contiene los 5 IP Cores desarrollados
- **tcl_script/**: Contiene el script base de creación de proyectos
- **vitis_code/**: Código fuente de aplicaciones de ejemplo
- **vivado_constraints/**: Archivos XDC de constraints físicos
- **vitis-axi-x**: Todos los proyectos de Vitis creados en la sección 3
- **vivado-axi-x**: Todos los proyectos de Vivado creados en la sección 3

Nota importante: Se debe tener en cuenta que clonar el repositorio desde la branch develop requiere aproximadamente 1.2 GB de espacio, mientras que clonar desde la branch main ocupa solo alrededor de 400 MB. Se recomienda ejecutar los proyectos de Vitis realizando primero el build de la plataforma y luego el de la aplicación.

5. Solución de Problemas

5.1. Errores comunes en Vivado

Durante el flujo de diseño en Vivado Design Suite, pueden surgir varios errores que afectan la síntesis, implementación o generación del bitstream. A continuación se presentan los más frecuentes y sus soluciones:

Error: [BD 41-1273] No interface ports matched

Este error ocurre cuando se intenta ejecutar Run Connection Automation pero Vivado no puede encontrar puertos compatibles para conectar automáticamente.

Solución:

1. Verificar que el IP Core tenga una interfaz AXI4-Lite Slave

2. Asegurarse que el AXI SmartConnect esté presente en el diseño
3. Si la conexión automática falla, conectar manualmente

Error: [Place 30-574] Poor placement for routing between an IO pin and BUFG

Problema de colocación de pines de entrada/salida que afecta el ruteo de señales de reloj.

Solución:

1. Verificar que el archivo de constraints (.xdc) tenga definiciones correctas de pines
2. Asegurar que las señales de reloj estén conectadas a buffers globales (BUFG)
3. Revisar que los pines físicos en Basys3 estén correctamente mapeados en el XDC

Error: [DRC 23-20] Rule violation

Violación de niveles lógicos debido a pull-ups/pull-downs conflictivos en pines de I/O.

Solución:

1. Revisar el archivo de constraints para pines con conflictos
2. Verificar que no haya múltiples definiciones para el mismo pin
3. Asegurar compatibilidad entre voltajes de I/O de la Basys3 (3.3V)

5.2. Errores comunes en Vitis

Durante el desarrollo de software embebido en Vitis IDE pueden presentarse errores de depuración y comunicación con la placa. A continuación se detallan los más comunes:

Error: Could not find ARM device on the board for connection 'Local'. Check if the target is in...

Este error indica que Vitis no puede conectarse al procesador MicroBlaze V en la tarjeta Basys3. Las causa puede estar relacionada con la conexión UART/JTAG.

Soluciones (en orden de prioridad):

1. **Verificar conexión física del cable USB:**
 - Desconectar y reconectar el cable USB tipo A a Micro-USB
 - Utilizar puertos USB 2.0 o superior (no USB 1.1)
 - Probar con un cable USB diferente para descartar falla de cable
 - Verificar visualmente que el conector no está dañado
2. **Verificar driver FTDI en el sistema:**
 - En Windows: Abrir Device Manager y buscar la tarjeta bajo Ports (COM & LPT). Debe aparecer como *USB Serial Port (COMx)* o similar.

- Si aparece con símbolo de exclamación: Desinstalar y reinstalar drivers desde <https://ftdichip.com/drivers/>
- En Linux: Ejecutar `lsusb` y verificar que aparezca *Future Devices Unlimited, Ltd*

3. Verificar configuración del bitstream:

- Asegurar que el bitstream se generó correctamente en Vivado sin errores DRC o implementación
- En Vitis, verificar que la plataforma de hardware importada (`design_1_wrapper.xsa`) contiene el bitstream compilado
- Si es necesario, regenerar el bitstream en Vivado: *Generate Bitstream*

4. Reiniciar el servicio: Cerrar Vitis completamente, esperar 10 segundos, reabrirlo

Error: Invalid target. Use “connect” command to connect to hw_server/TCF agent

Este error indica que Vitis no tiene un destino (target) de hardware configurado para la ejecución del programa.

Soluciones:

1. Conectar el hardware manualmente:

- En Vitis: Menú *Vitis* → *Program Device...*
- Click en *Select* → *Device* → *Auto Connect*
- Debe detectarse automáticamente la tarjeta Basys3

2. Reconstruir la plataforma y la aplicación:

- Dar click en *Build* a la plataforma y luego a la aplicación, verificar que no haya ningún error de compilación.

3. Verificar puerto serial UART:

- Es posible que el software haya programado correctamente el bitstream pero falta conectar el puerto UART
- Verificar que está abierto un Serial Monitor en el puerto correcto (COM6 o el que corresponda según Device Manager)
- Baud rate debe ser 9600 (configurado en el proyecto)
- Si no aparece el puerto en el Serial Monitor, reinstalar drivers FTDI

5.3. Problemas de compatibilidad

El script `mbv_axi_base.tcl` requiere compatibilidad específica con versiones de software y configuraciones de hardware para funcionar correctamente. Los problemas de compatibilidad son la causa más frecuente de fallos en la creación de proyectos.

Requisitos de compatibilidad críticos:

Versión de Vivado

El script mbv_axi_base.tcl fue generado específicamente para **AMD Vivado Design Suite 2025.1**. Esta versión es obligatoria por las siguientes razones:

- **Procesador MicroBlaze V:** Las características de RISC-V y configuración de depuración utilizadas en el script fueron introducidas en versiones recientes de Vivado. Versiones anteriores no reconocerán los parámetros del procesador y fallarán.
- **IP Cores Xilinx:** Algunos IP Cores como Clock Wizard y AXI SmartConnect tienen cambios en interfaces y parámetros entre versiones
- **Block Design automation:** El comando Run Connection Automation puede no conectar correctamente los periféricos en versiones anteriores

Verificación de versión: Abrir Vivado y en la ventana de bienvenida verificar que aparezca Vivado 2025.1. Si es diferente, desinstalar y descargar la versión correcta desde <https://www.xilinx.com/support/download.html>.

Tarjeta de desarrollo Basys3

El script está diseñado específicamente para la tarjeta Digilent Basys3 con FPGA Xilinx Artix-7 (modelo XC7A35T). Otras tarjetas no son compatibles por:

- Diferentes números de pines de I/O (16 LEDs, 16 switches, 5 botones, display de 7 segmentos)
- Reloj del sistema a 100 MHz (algunas tarjetas utilizan 50 MHz u otras frecuencias)
- Conectividad USB a través del chip FTDI (otras tarjetas pueden usar diferentes controladores)
- Archivo de constraints (.xdc) específico para esta tarjeta

Limitación importante: Intentar ejecutar el script en tarjetas como Nexys, Zybo o Zynq resultará en errores de síntesis o pinout.

Rutas de archivos sin espacios ni caracteres especiales

El script Tcl puede ser sensible a rutas que contengan espacios o caracteres acentuados.

Rutas seguras (recomendadas):

```
C:\Users\usuario\Desktop\mbv-axi-slave-ip-repo  
C:\Proyectos\FPGA\proyecto  
/home/usuario/projects/fpga
```

Memoria local del procesador MicroBlaze V

El script configura por defecto 32 KB de memoria local (BRAM) mediante el componente `microblaze_riscv_0_local_memory`. Esto es suficiente para aplicaciones simples.

Para proyectos que requieren más memoria (como *AXI Secure Authentication*):

Es necesario modificar el script antes de ejecutarlo. En el archivo `mbv_axi_base.tcl`, localizar las líneas 458-459:

Cambiar `0x00008000` (32 KB) por `0x00020000` (128 KB)

Guardar los cambios antes de ejecutar el script.

Nota: Aumentar la memoria local consume más recursos de BRAM de la FPGA.

Compatibilidad entre Vivado y Vitis

Los archivos generados por Vivado (especialmente `design_1_wrapper.xsa`) deben ser compatibles con la versión de Vitis utilizada.

Recomendación: Utilizar versiones coincidentes de Vivado y Vitis (ambas 2025.1).

Problemas de versiones desacopladas:

- Vivado 2025.1 + Vitis 2024.1: Posibles incompatibilidades en plataformas
- Vivado 2024.1 + Vitis 2025.1: El archivo `.xsa` puede no ser reconocido

Referencias

- [1] ARM Limited, "AMBA AXI and ACE Protocol Specification," [En línea]. Disponible en: <https://developer.arm.com/documentation/ihi0022/h>.
- [2] AMD Xilinx, "MicroBlaze V Embedded Design," User Guide UG1711, 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug1711-microblaze-v-embedded-design/>.
- [3] AMD Xilinx, "MicroBlaze V User Guide," User Guide UG1629, 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug1629-microblaze-v-user-guide/>.
- [4] AMD Xilinx, "Vivado Design Suite AXI Reference Guide", User Guide UG1037, 2025. [En línea]. Disponible en: <https://docs.amd.com/v/u/en-US/ug1037-vivado-axi-reference-guide>.
- [5] AMD Xilinx, "Vivado IP Subsystems", User Guide UG994, 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug994-vivado-ip-subsystems>.
- [6] AMD Xilinx, "Vitis Accelerated Embedded Documentation", User Guide UG1701, 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug1701-vitis-accelerated-embedded/>.
- [7] AMD Xilinx, "Vitis Embedded Software Development", User Guide UG1400, 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug1400-vitis-embedded>.
- [8] E. Barker and J. Kelsey, "Recommendation for the entropy sources used for random bit generation", NIST Special Publication 800-90B, Jan. 2018. [En línea]. Disponible en: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf>
- [9] National Institute of Standards and Technology, "Secure hash standard (SHS)," Federal Information Processing Standards Publication 180-4, Aug. 2015. [En línea]. Disponible en: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>
- [10] Digilent, "Basys 3 Reference Manual", [En línea]. Disponible en: <https://digilent.com/reference/programmable-logic/basys-3/reference-manual>.
- [11] G. Marsaglia, "Xorshift RNGs," *Journal of Statistical Software*, vol. 8, no. 14, pp. 1–6, 2003. [En línea]. Disponible en: <https://www.jstatsoft.org/article/view/v008i14>
- [12] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain", *Psychological Review*, vol. 65, no. 6, pp. 386-408, 1958.