Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Programa de Licenciatura en Ingeniería Electrónica

# Implementación de algoritmos de aprendizaje automático en procesadores softcores basados en ARM

Informe de Trabajo Final de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

José Pablo Castro Valverde

Cartago, 23 de noviembre de 2021

Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.

José Pablo Castro Valverde

Cartago, 23 de noviembre de 2021

Céd: 02-0559-0110

# Resumen

El presente informe describe la implementación de un sistema mínimo con un procesador softcore ARM Cortex-M1 en un FPGA de la familia Artix-7 utilizando la tarjeta de desarrollo Nexys-A7 y ejecutando un modelo de aprendizaje automático entrenado empleando TensorFlow Lite Micro.

Se muestra así mismo un comportamiento consistente del modelo en tres escenarios distintos: simulación en QEMU de un SoC ARM Cortex-M3, una tarjeta de desarrollo con un SoC Cortex-M4F y en un diseño con procesador softcore en un FPGA.

**Palabras clave:** SoC, softcore, TensorFlow Lite Micro, FPGA, Cortex-M1

# Abstract

The present work describe the implementation of a minimal system with an ARM Cortex-M1 softcore processor in an Artix-7 Family FPGA using the Nexys-A7 development board and executing a machine learning model trained employing TensorFlow Lite Micro.

Likewise, a consistent behavior of the model is shown in three different scenarios: simulation in QEMU of an ARM Cortex-M3 SoC, a development board with a Cortex-M4F SoC and in a design with a softcore processor in an FPGA.

**Keywords:** SoC, softcore, TensorFlow Lite Micro, FPGA, Cortex-M1

# Índice general

**Índice de figuras** III

**Índice de tablas** V

**Lista de abreviaciones** VI

**1. Introducción** 1
- 1.1. Aprendizaje automático utilizando FPGA . . . 1
- 1.2. Sistema mínimo de aprendizaje automático en un FPGA de gama baja . . 3
- 1.3. Objetivos y estructura del documento . . . 3

**2. Marco teórico** 5
- 2.1. Procesador softcore ARM Cortex M . . . 5
  - 2.1.1. Memoria de programa . . . 8
  - 2.1.2. Características de trazabilidad de la familia de procesadores Cortex-M 10
- 2.2. Tarjetas FPGA . . . 11
  - 2.2.1. Familia Xilinx Artix-7 . . . 11
  - 2.2.2. Tarjeta Nexys A7 (100T) . . . 12
- 2.3. Tensorflow Lite Micro . . . 12
  - 2.3.1. Aprendizaje automático en microcontroladores . . . 13
  - 2.3.2. Plataformas soportadas . . . 13
  - 2.3.3. Ejemplos básicos de TensorFlow Lite Micro . . . 14
  - 2.3.4. Flujo de trabajo . . . 14
  - 2.3.5. Limitaciones . . . 14
- 2.4. Proyecto Zephyr . . . 15
- 2.5. Nordic Semiconductor nRF52840DK . . . 15
- 2.6. Desarrollo de aplicaciones para procesadores softcore en FPGA utilizando las herramientas de Xilinx . . . 16
  - 2.6.1. Diseño de bloques en Vivado . . . 17
  - 2.6.2. Generación de controladores (BSP) . . . 18
  - 2.6.3. Actualización del bitstream . . . 18
  - 2.6.4. Actualización de aplicación genérica del software para un procesador softcore ARM Cortex-M . . . 19

2.6.5. Actualización de aplicación de aprendizaje automático del software para un procesador softcore ARM Cortex-M . . . . . 19

**3. Implementación de algoritmos entrenados con TensorFlow Lite Micro en SoC de arquitectura ARM Cortex-M y softcores en FPGA 22**

3.1. Diseño de plataforma de hardware con procesador softcore utilizando un FPGA de Xilinx . . . . . 22

3.1.1. Diseño m1_nexys_a7_gpio . . . . . 23

3.1.2. Diseño m1_nexys_a7_uart . . . . . 25

3.1.3. Diseño m1_nexys_a7_io . . . . . 28

3.1.4. Diseño m1_nexys_a7_sdram . . . . . 32

3.1.5. Diseño m1_nexys_a7 . . . . . 32

**4. Resultados y análisis 38**

**5. Conclusiones 41**

5.1. Trabajos derivados . . . . . 41

**Bibliografía 43**

# Índice de figuras


1.1. Diagrama de flujo de alto nivel para el desarrollo de aplicaciones de aprendizaje automático en FPGA . . . 2

2.1. Procesador softcore del Cortex-M1 . . . 5
2.2. Diagrama de bloques internos del softcore del procesador Cortex-M1 . . . 7
2.3. Diagrama de bloques en externo en Vivado del softcore del procesador Cortex-M1 sin el uso de depuración . . . 7
2.4. Mapa de memoria del procesador Cortex-M1 . . . 8
2.5. Tarjeta de desarrollo nRF5284DK . . . 16
2.6. Diagrama de flujo general de desarrollo FPGA del fabricante Xilinx. . . . 17
2.7. Diagrama de flujo para la actualización del archivo Bitstream. . . . 19
2.8. Diagrama de flujo para la actualización de una aplicación general. . . . 20
2.9. Árbol de directorio, parcial, de la biblioteca TensorFlow Lite para arquitectura ARM Cortex-M. . . . 20
2.10. Diagrama de flujo para la actualización de una aplicación utilizando TensorFlow Lite. . . . 21

3.1. Diagrama de bloques del diseño m1_nexys_a7_gpio. . . . 23
3.2. Diagrama de bloques del diseño m1_nexys_a7_gpio. Vista de interfaces. . . . 24
3.3. Mapa de memoria del diseño m1_nexys_a7_gpio. . . . 24
3.4. Árbol de directorio del BSP para el sistema de la figura 3.1 desarrollado en Vitis IDE. . . . 25
3.5. Diagrama de flujo simplificado de la aplicación “Blinky” de prueba para el diseño mostrado en la Figura 3.1. . . . 25
3.6. Diagrama de flujo simplificado de la aplicación “BlinkyInt” de prueba para el diseño mostrado en la Figura 3.1. . . . 26
3.7. Diagrama de bloques del diseño m1_nexys_a7_uart. . . . 26
3.8. Diagrama de bloques del diseño m1_nexys_a7_uart vista de interfaces. . . . 27
3.9. Mapa de memoria del diseño m1_nexys_a7_uart. . . . 27
3.10. Árbol de directorio del BSP para el sistema de la figura 3.7 desarrollado en Vitis IDE. . . . 27
3.11. Diagrama de flujo de la aplicación de prueba para el diseño mostrado en la Figura 3.7. . . . 28
3.12. Diagrama de bloques del diseño m1_nexys_a7_io. . . . 28

3.13. Diagrama de bloques del diseño m1_nexys_a7_io vista de interfaces. . . . . 29
3.14. Mapa de memoria del diseño m1_nexys_a7_io. . . . . 29
3.15. Árbol de directorio del BSP para el sistema de la figura 3.12 desarrollado en Vitis IDE. . . . . 30
3.16. Diagrama de flujo de la aplicación de prueba para el diseño mostrado en la Figura 3.12. . . . . 31
3.17. Diagrama de bloques del diseño m1_nexys_a7_sdram. . . . . 32
3.18. Diagrama de bloques del diseño m1_nexys_a7_sdram vista de interfaces. . . 33
3.19. Mapa de memoria del diseño m1_nexys_a7_sdram. . . . . 33
3.20. Árbol de directorio del BSP para el sistema de la figura 3.17 desarrollado en Vitis IDE. . . . . 34
3.21. Diagrama de bloques del sistema final. . . . . 34
3.22. Diagrama de bloques del sistema final m1_nexys_a7 vista de interfaces. . . 35
3.23. Mapa de memoria del sistema final m1_nexys_a7. . . . . 35
3.24. Árbol de directorio del BSP para el sistema de la figura 3.21 desarrollado en Vitis IDE. . . . . 36
3.25. Diagrama de flujo de la aplicación de prueba para el diseño mostrado en la Figura 3.21. . . . . 37

4.1. Comparación entre la función senoidal ideal y los resultados del modelo entrenado con TensorFlow Lite Micro ejecutado en los tres escenarios utilizados (qemu_arm_cortex_m3, nRF52840 y procesador softcore en un FPGA) . . . . . 40

# Índice de tablas

2.1. Señales entrada/salida del softcore Cortex-M1 . . . . . 8
2.2. Regiones de la memoria del procesador . . . . . 9
2.3. Compatibilidad con funciones de depuración y seguimiento en los procesadores Cortex-M . . . . . 11
2.4. Muestra de dispositivos de la familia Artix-7 . . . . . 12
2.5. Módulos AXI IP Block disponibles para los periféricos de la tarjeta Nexys-A7 18
2.6. Módulos AXI IP Block disponibles para uso interno del diseño de bloques . 18

4.1. Huella de las aplicaciones de prueba para cada diseño de hardware . . . . . 38
4.2. Resultados del modelo entrenado en TensorFlow Lite. . . . . 38
4.3. Comparación del error cuadrático medio y error absoluto medio del ejemplo de TensorFlow Lite Micro en los tres escenarios probados . . . . . 39

# Lista de abreviaciones

## Abreviaciones

| | |
|---|---|
| AHB-Lite | Advanced High-perfomance Bus reduced version |
| AHB | Advanced High-perfomance Bus |
| AMBA | Advanced Microcontroller Bus Architecure |
| API | Application Programming Interface |
| ARM | Advanced RISC Machines |
| AXI | Advanced eXtensible Interface |
| BRAM | Block RAM |
| BSP | Board Support Package |
| CMSIS | Common Microcontroller Software Interface Standard |
| DDR2 | Double Data Rate 2 |
| DSP | Digital Signal Processing |
| DTCM | Data Tightly-Coupled Memory |
| DWT | Data Watchpoint and Trace |
| ELF | Executable and Linkable Format |
| FPGA | Field-Programmable Gate Array |
| IIC | Inter-Integrated Circuit |
| IoT | Internet of Things |
| ITCM | Instruction Tightly-Coupled Memory |
| LC | Logic Cell |
| LC | Logic Cells |
| LED RGB | 3 Color Light Emitting Diode: Red, Green and Blue |
| LED | Light Emitting Diode |
| MCU | Micro Controller Unit |
| ML | Machine Learning |
| MMCM | Mixed-Mode Clock Manager |
| MMI | Memory Map Information |
| NMI | Non Maskable Interruption |
| NVIC | Non Vectorial Interruptions Controller |
| QEMU | Quick EMUlator |
| RAM | Random Access Memory |
| RTOS | Real Time Operating System |
| SDK | Software Development Kit |
| SDRAM | Synchronous Dynamic Random-Access Memory |

| | |
|---|---|
| SoC | Sistem on Chip |
| SPI | Serial Peripheral Interface |
| UART | Universal Asynchronous Receiver/Transmitter |
| XSA | Xilinx Shell Archive |

# Capítulo 1

# Introducción

## 1.1. Aprendizaje automático utilizando FPGA

La utilización de arreglos de compuertas programables en campo, *Field-Programmable Gate Array* (FPGA), para la implementación de aplicaciones complejas de aprendizaje automático en dispositivos en el borde (*edge computing devices*), abre la posibilidad para desarrollar soluciones de hardware reconfigurable de bajo costo. Esto es deseable ya que el desarrollo de aplicaciones de aprendizaje automático cambia rápidamente, con nuevas topologías y redes utilizando un ciclo de trabajo que permite la mejora continua. Así, las FPGA presentan un escenario apropiado para la puesta en marcha de aceleradores desarrollados para aumentar el rendimiento y reducir el consumo de potencia y energía de estas aplicaciones. Sin embargo, las FPGA de gama baja no cuentan con un procesador hardcore, requerido para ejecutar aplicaciones de aprendizaje automático utilizando marcos de trabajo especializados, como por ejemplo, TensorFlow Lite, con la ayuda de los aceleradores que, de manera automática, se puedan diseñar e implementar en dichas FPGA.

En este momento, no se cuenta con un ejemplo funcional de un sistema mínimo con un procesador softcore capaz de ejecutar modelos entrenados con herramientas de aprendizaje automático utilizando un FPGA de gama baja. Se requiere entonces elaborar una guía con los pasos a seguir para replicar y/o ampliar el ejemplo básico.

Para desarrollar dicho protocolo se utilizará el flujo de trabajo general mostrado en la Figura 1.1.

A continuación una breve descripción de cada etapa del flujo de trabajo general:

**Diseño de Hardware:** En esta etapa se diseña el hardware que cumpla con los requisitos de la aplicación específica utilizando un procesador softcore como núcleo del diseño.

**Controladores:** En esta etapa se desarrollan los controladores necesarios para utilizar el hardware previamente diseñado.

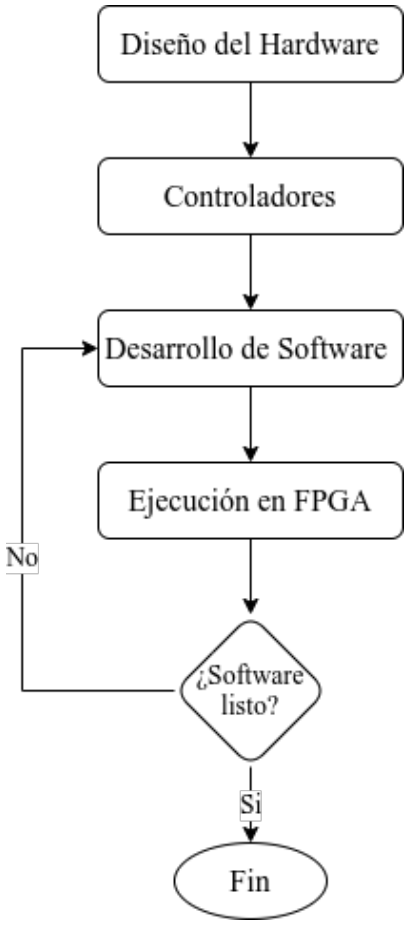


**Figura 1.1:** Diagrama de flujo de alto nivel para el desarrollo de aplicaciones de aprendizaje automático en FPGA

**Desarrolo de software:** En esta etapa se desarrolla la aplicación de software que se desplegará en la plataforma de hardware diseñado.

**Ejecución en FPGA:** En esta etapa se programa la FPGA con la plataforma de hardware y con la aplicación de software.

**¿Software listo?:** Si la aplicación de software cumple con los requisitos del proyecto entonces termina el flujo, caso contrario es necesario actualizar el software y continuar con el bucle hasta cumplir con los requisitos de la aplicación de software.

Una vez que se ha desarrollado la plataforma de hardware, el flujo de trabajo del desarrollo de software es el mismo al utilizado con cualquier tarjeta de desarrollo con un SoC Sistema en chip (System on Chip), con procesadores hardcore disponibles en el mercado y de distintas arquitecturas.

## 1.2. Sistema mínimo de aprendizaje automático en un FPGA de gama baja

Siguiendo el flujo general descrito en el apartado anterior, en este informe de proyecto de graduación se presenta un sistema mínimo, tal que en la etapa del desarrollo de hardware se utiliza un procesador softcore de ARM Cortex-M1 como núcleo conectado a puertos de entrada/salida de propósito general (LED, Push Buttons, Switch Bottons, LED RGB, UART, entre otros), además de un módulo para memoria de programa y conexión a memoria de datos (RAM), externa. Así mismo, se utilizarán las herramientas provistas por el fabricante del FPGA para implementar los controladores. En la etapa de desarrollo de software se implementará un ejemplo básico de TensorFlow Lite Micro, el cual cuenta con un modelo previamente entrenado. Esto servirá de ejemplo base para establecer el protocolo de desarrollo desde cero para futuros proyectos más complejos con algoritmos diseñados y previamente entrenados en un sistema con mayor potencia.

## 1.3. Objetivos y estructura del documento

El objetivo general del trabajo es entonces implementar un sistema mínimo con un procesador softcore ARM Cortex-M1 y ejecutar en él un ejemplo de aplicación de aprendizaje automático, que funcione como ejemplo básico para futuros desarrollos así como una metodología básica a seguir para replicar y ampliar el presente proyecto.

El ejemplo de `hello_world` del TensorFlow Lite Micro, utiliza un modelo entrenado para replicar una función sinusoidal, es decir, toma un solo número como entrada y genera el valor sinusoidal correspondiente. Este resultado no es exacto, es una aproximación.

Para comprobar el correcto funcionamiento de un modelo de aprendizaje automático se requiere ejecutar el ejemplo `hello_world` de TensorFlow Lite Micro en tres escenarios distintos:

1. En un computador de escritorio con un sistema operativo GNU/Linux.
2. En una tarjeta de desarrollo con un SoC con un MCU ARM Cortex-M.
3. En un sistema mínimo con un procesador softcore de ARM Cortex-M en un FPGA.

Para verificar el modelo entrenado en los tres escenarios mencionados, se comparará el resultado obtenido en los tres escenarios con el valor ideal de la función sinusoidal.

Para cumplir con la ejecución en un computador de escritorio así como en una tarjeta de desarrollo con un MCU ARM Cortex-M es requisito utilizar el sistema operativo del proyecto Zephyr con el fin de simplificar el proceso de prueba y ahorrar tiempo útil para la etapa en el FPGA la cual requiere más trabajo de prueba y depuración.

Así mismo se requiere de una guía con los pasos necesarios para replicar el flujo de trabajo realizado en este proyecto. Esta guía debe incluirse en la documentación del repositorio en github del proyecto para futuros desarrollos y actualizaciones de la misma documentación como un documento vivo.

En el Capítulo 2 se exponen los conocimientos mínimos necesarios para entender la solución.

En el Capítulo 3 se expone el procedimiento a seguir para ejecutar el ejemplo de TensorFlow Lite Micro en los dos primeros escenarios. El desarrollo en la FPGA se implementan 5 diseños de hardware, utilizando el flujo de trabajo general descrito anteriormente.

En el Capítulo 4 se muestran los resultados de la ejecución del ejemplo `hello_world` de TensorFlow Lite Micro en los tres escenarios.

En el Capítulo 5 se resumen las conclusiones del presente proyecto así como una lista de posibles proyectos derivamos al presente trabajo.

# Capítulo 2

# Marco teórico

## 2.1. Procesador softcore ARM Cortex M

Los procesadores softcore de ARM Cortex-M1 y M3 disponibles en el sitio web de la compañía como parte del programa DesignStart FPGA, están diseñados para su uso en FPGA como módulos IP cerrados que se instancian en un diagrama de bloques y se configuran según las necesidades del proyecto a desarrollar.

Dependiendo de la configuración del módulo se agregan o eliminan componentes de hardware interno y se agregran o eliminan señales externas de entrada y/o salidas.

El módulo IP de pruebas del procesador Cortex-M1 proporcionado por ARM en su programa DesignStart FPGA, cuenta con las siguientes características básicas mostradas en la figura 2.1

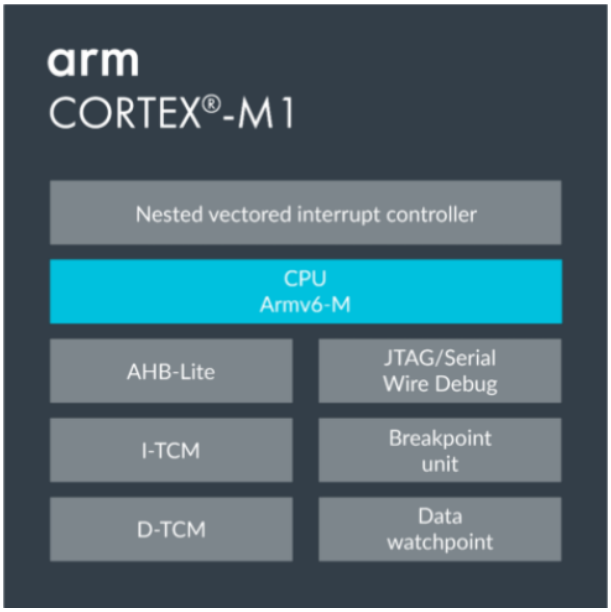


**Figura 2.1:** Procesador softcore del Cortex-M1 (tomado de [1])

Estas son las principales características del núcleo Cortex-M1:

1. Hasta 32 interrupciones como máximo.
2. Interfaces ITCM (Instruction Tightly Coupled Memory), DTCM (Data Tightly Coupled Memory), ambos de hasta 1 MB para memoria de programa y datos respectivamente. El propósito de la memoria estrechamente acoplada (TCM) es proporcionar memoria de baja latencia que el procesador puede usar sin la imprevisibilidad que es una característica de los cachés.
3. Depuración por línea serial (SW), JTAG o una combinación de ambos. También es posible deshabilitar la depuración del todo.
4. Extensiones configurables: OS, formato Little-endian, soporte de depuración integrado.

Internamente el Cortex-M1 utiliza un bus de comunicación AMBA (*Advanced Microcontroller Bus Architecure*) AHB-Lite (*Advanced High-perfomance Bus reduced version*). Para comunicarse con los módulos IP provistos por Xilinx, que utilizan un puerto AXI (*Advanced eXtensible Interface*), el softcore cuenta con un puente entre AHB (*Advanced High-perfomance Bus*) y AXI.

Entre algunas de las funcionalidades del Cortex-M1 se encuentran las siguientes:

**Instrucciones:** Utiliza la arquitectura ARMV6-m RISC. Utiliza el set de instrucciones Thumb de 16 bits y el Thumb2 de 32 bits.

**Registros:** Contiene trece registros de 32 bits divididos en 2 bloques. Los registros bajos (R0-R7) y los registros altos (R8-R12). Los registros más utilizados son: stack pointer (SP), registro de enlace (LR), contador de programa (PC) y el registro de estado del programa (xPSR)

**Interfaz de memoria:** Cuenta con dos interfaces de memoria: una para memoria de programa, ITCM, y la otra para memoria de datos DTCM.

**NVIC:** Tiene la capacidad de utilizar 1, 16 y hasta 32 interrupciones gestionadas por el controlador de interrupciones vectoriales (NVIC, Non Vectorial Interruptions Controller).

**NMI:** Interrupción no enmascarable o NMI. Esta es la interrupción de mayor prioridad y en caso de no utilizarse, se debe conectar a la señal GND para evitar que el sistema se bloquee en un intento por atender una interrupción inexistente.

**Depuración:** El procesador tiene 4 modos de operación para la depuración:

1. JTAG.
2. Serial Wire.

3. JTAG and Serial Wire.
4. No Debug. Esta opción elimina el hardware requirido para la depuración.

El diagrama de bloques del microprocesador sin depurador se muestra en la Figura 2.2 el cual está configurado para ocho interrupciones externas.

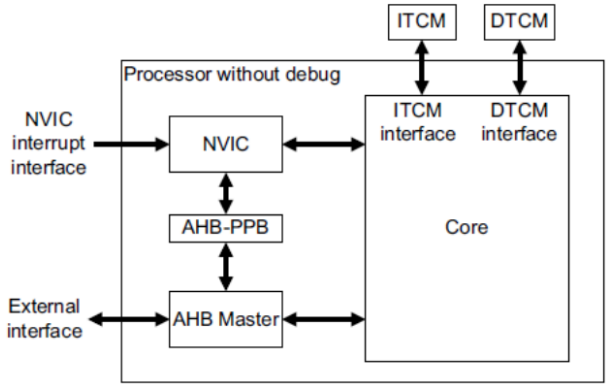


**Figura 2.2:** Diagrama de bloques internos del softcore del procesador Cortex-M1 (tomado de [1]).

La Figura 2.3 muestra el módulo IP en la aplicación Vivado, del fabricante Xilinx, sin el uso del módulo interno de depuración.

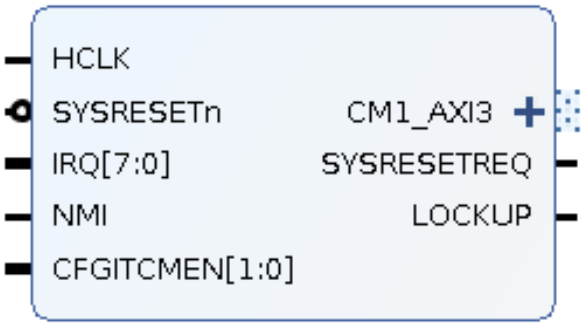


**Figura 2.3:** Diagrama de bloques en externo en Vivado del softcore del procesador Cortex-M1 sin el uso de depuración

El puerto CFGITCMEN[1:0] activa o desactiva el alias inferior y alias superior del ITCM. Este puerto debe conectar ambos bits a 0 para el caso de no utilizar la interfaz interna de memoria. En el caso de desactivar el ITCM y el DTCM es necesario proveer una memoria externa para el programa y los datos.

Xilinx provee un módulo de memoria externa BRAM con el que es posible proveer de la memoria de programa y datos en un solo bloque, dividiendo dicha memoria en dos. El archivo del enlazador es configurado de manera que se mapee adecuadamente la memoria de programa y datos.

La Tabla 2.1 resume los puertos de entrada y salida del procesador softcore Cortex-M1 con la configuración sin depuración, la cual es la configuración más simple disponible.

**Tabla 2.1:** Señales entrada/salida del softcore Cortex-M1

| Nombre | Tipo | Descripción |
|---|---|---|
| HCLK | Entrada | Señal de reloj principal del procesador |
| SYSRESETn | Entrada | Señal de reset del sistema |
| IRQ | Entrada | Señales de interrupciones vectoriales |
| NMI | Entrada | Interrupción no enmascarable |
| CFGITCMEN[1:0] | Entrada | Habilita el Alias ITCM. [UpperAlias:LowerAlias] |
| CM1_AXI3 | Salida | Puerto maestro AXI |
| SYSRESETREQ | Salida | Señal para solicitar al sistema controlador de reset que reinicie al núcleo. Se mantiene no conectada |
| LOCKUP | Salida | Señal que indica que el sistema ha entrado en un estado de LOCKUP y por tanto el procesador softcore está bloqueado |

### 2.1.1. Memoria de programa

El procesador softcore cuenta con un mapa de memoria bien definido para cada uno de los componentes internos y externos, tal como se muestra en la Figura 2.4.

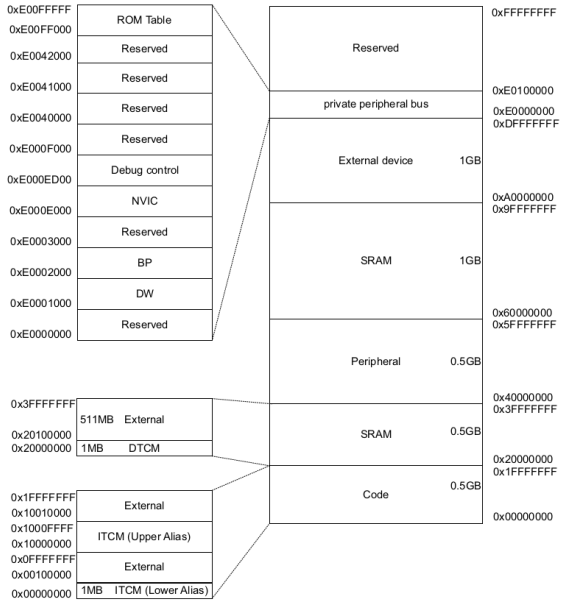


**Figura 2.4:** Mapa de memoria del procesador Cortex-M1 (tomado de [1])

La Tabla 2.2 resume el detalle de los permisos de las regiones de memoria del procesador. Nota: "**XN**: Execute Never". Una región está marcada como XN para prevenir que se obtengan las instrucciones desde esa región.

**Tabla 2.2:** Regiones de la memoria del procesador (tomado de [1])

| Región | Nombre | Tipo de dispositivo | XN | Interfaz accesada |
|---|---|---|---|---|
| `0x00000000` `0x000FFFFF` | Code, ITCM, Lower Alias | Normal | | Si el Alias inferior del ITCM está habilitado, la búsqueda de instrucciones se realiza en el ITCM. |
| `0x00100000` `0x0FFFFFFF` | Code, external | Normal | | Las búsquedas de instrucciones y los accesos a datos se realizan en el bus del sistema externo. |
| `0x10000000` `0x1000FFFF` | Code, ITCM, Upper Alias | Normal | | Si el alias superior de ITCM está habilitado, se realizan búsquedas de instrucciones y accesos de datos a ITCM. |
| `0x10010000` `0x1FFFFFFF` | Code, external | Normal | | Las búsquedas de instrucciones y los accesos a datos se realizan en el bus del sistema externo. |
| `0x20000000` `0x200FFFFF` | SRAM, DTCM | Normal | XN | Recuperación de instrucciones fallida. Los accesos a datos se realizan en la DTCM. |
| `0x20100000` `0x3FFFFFFF` | SRAM, external | Normal | | Las búsquedas de instrucciones se realizan en el bus del sistema externo. Los accesos a datos se realizan al bus del sistema externo. |
| `0x40000000` `0x5FFFFFFF` | Peripheral | Device | XN | Los accesos a datos se realizan al bus del sistema externo. |
| `0x60000000` `0x9FFFFFFF` | SRAM | Normal | | Los accesos de instrucciones y datos se realizan en el bus del sistema externo. |
| `0xA0000000` `0xDFFFFFFF` | External Device | Device | XN | Los accesos a datos se realizan al bus del sistema externo. |

*Continúa en la siguiente página*

| Región | Nombre | Tipo de dispositivo | XN | Interfaz accesada |
|---|---|---|---|---|
| `0xE0000000` `0xE00FFFFF` | Private Peripheral Bus | SO | XN | Los accesos a los datos se realizan a través del PPB. |
| `0xE0100000` `0xFFFFFFFF` | System | | XN | Segmento del sistema. Los accesos a las instrucciones están impedidos y con fallas. Esta región está reservada para la búsqueda de datos. |

### 2.1.2. Características de trazabilidad de la familia de procesadores Cortex-M

Las características de trazabilidad en los procesadores Cortex-M incluyen:

**Micro Trace Buffer (MTB):** es una solución de rastreo de instrucciones de bajo costo que permite al sistema asignar una pequeña parte de la SRAM del sistema para el rastreo de instrucciones. El resultado del seguimiento se puede recopilar a través de la conexión de depuración.

**Embedded Trace Macrocell (ETM):** es una solución de rastreo de instrucciones en tiempo real que transmite información de ejecución de instrucciones para depurar el anfitrión a través de una conexión de rastreo.

**Trazabilidad de eventos:** Trazamiento en tiempo real de eventos de excepción generados por el módulo DWT (Data Watchpoint and Trace).

**Traza de perfiles:** es la generación de carreras para el análisis del rendimiento del sistema generado por el módulo DWT.

**Trazamiento de datos selectivo:** es un trazamiento en tiempo real de una pequeña selección de datos generados por el módulo DWT. Los comparadores de puntos de observación de datos se utilizan para detectar accesos a la ubicación de la dirección específica, y si se realiza una transferencia a dicha ubicación, la información sobre la misma se puede exportar en la interfaz de seguimiento.

**Traza de instrumentación:** un seguimiento en tiempo real que genera el software para proporcionar mensajes de depuración y soporte de reconocimiento del sistema operativo. Por ejemplo, es posible dirigir los mensajes printf ("Hola mundo") a la macrocélula de seguimiento de instrumentación (ITM) para mostrar el mensaje en una consola de depuración en el anfitrión de depuración.

**Soporte de marca de tiempo:** la mayoría de las fuentes de seguimiento (excepto MTB) admiten la marca de tiempo y eso permite que el anfitrión de depuración reconstruya la sincronización de varios eventos.

Debido a limitaciones de área y energía, no todos los procesadores Cortex-M son compatibles con todas estas funciones. La Tabla 2.3 enumera las funciones de depuración y rastreo compatibles con los procesadores Cortex-M actuales.

**Tabla 2.3:** Compatibilidad con funciones de depuración y seguimiento en los procesadores Cortex-M (tomado de [12]).

| | Comparadores de puntos de interrupción | Comparadores de puntos de observación | MTB | ETM | Seguimiento de datos selectivo de DWT | ITM |
|---|---|---|---|---|---|---|
| Cortex-M0 | Hasta 4 | Hasta 2 | | | | |
| Cortex-M1 | Hasta 4 | Hasta 2 | | | | |
| Cortex-M0+ | Hasta 4 | Hasta 2 | X | | | |
| Cortex-M3 | Hasta 6 (instrucción +2 literal) | Hasta 4 | | X | X | X |
| Cortex-M4 | Hasta 6 (instrucción +2 literal) | Hasta 4 | | X | X | X |
| Cortex-M7 | Hasta 8 | Hasta 4 | | X | X | X |
| Cortex-M23 | Hasta 4 | Hasta 4 | X | X | | |
| Cortex-M33 | Hasta 8 | Hasta 4 | X | X | X | X |
| Cortex-M35P | Hasta 8 | Hasta 4 | X | X | X | X |

## 2.2. Tarjetas FPGA

### 2.2.1. Familia Xilinx Artix-7

La familia de FPGA Artix-7 está optimizada para aplicaciones de baja potencia y alto rendimiento para aplicaciones de DSP (*Digital Signal Processing*). Cuenta con soporte para memorias DDR2 y DDR3.

Las tarjetas de desarrollo Arty A7 (Artix-7 35T o Artix-7 100T) se han utilizado para generar el proyecto de ejemplo provisto por ARM. La Tabla 2.4 muestra algunos dispositivos de la familia Artix-7 comparando el número de (celdas lógicas), BRAM de 36kB, DSP slices y MMCM.

**Tabla 2.4:** Muestra de dispositivos de la familia Artix-7

| Dispositivo | Número de LC | Número de BRAM de 36 kb | BRAM bits | Número de DSP slices | Número de MMCM |
|---|---|---|---|---|---|
| XC7A15T | 16640 | 25 | 900k | 45 | 5 |
| XC7A35T | 33280 | 50 | 1800k | 90 | 5 |
| XC7A50T | 52160 | 75 | 2700k | 120 | 5 |
| XC7A75T | 75520 | 105 | 3780k | 180 | 6 |
| XC7A100T | 101440 | 135 | 4860k | 240 | 6 |
| XC7A200T | 215360 | 365 | 13140k | 740 | 10 |

### 2.2.2. Tarjeta Nexys A7 (100T)

Ladino [2] hace uso de una tarjeta Nexys4, antecesora de la Nexys A7, para implementar un sistema con múltiples núcleos compartiendo un sistema de interconexión AXI. En dicho trabajo la implementación final utilizó 4 núcleos softcore ARM Cortex-M1 con la configuración sin depuración, demostrando así las capacidades de la tarjeta Nexys4 para implementar un sistema de estas características.

Para el desarrollo del presente proyecto se cuenta con la tarjeta Digilent Nexys A7 en su modelo de 100T con espacio suficiente para implementar una plataforma de hardware con el procesador softcore Cortex-M1 como su núcleo central. El presente proyecto únicamente utilizará un núcleo pero implementará un ejemplo de `hello_world` de la herramienta TensorFlow Lite Micro.

El nombre del dispositivo tal y como lo identifica Vivado es Nexys A7-100T (xc7a100tcsg324-1) y requiere instalar los archivos provistos por Digilent con los que facilitan la conexión de los dispositivos incluídos en la tarjeta dentro de un diseño de bloques en Vivado permitiendo la asignación automática de los pines externos del FPGA con sus respectivos puertos de entrada y/o salida según corresponda.

De esta manera solo es necesario arrastrar el bloque que se necesita incluir en el diagrama de bloques y Vivado se encargará de asignar de manera automática los pines correspondientes.

## 2.3. Tensorflow Lite Micro

TensorFlow [4] es una interfaz para expresar algoritmos de aprendizaje automático y una implementación para ejecutar dichos algoritmos. Un cálculo expresado con TensorFlow se puede ejecutar con poco o ningún cambio sistemas, que van desde dispositivos móviles como teléfonos y tabletas hasta sistemas distribuidos a gran escala de cientos de máquinas y miles de dispositivos computacionales como tarjetas GPU. El sistema es flexible y se

puede usar para expresar incluidos algoritmos de entrenamiento e inferencia para modelos de redes neuronales profundas, y se ha utilizado para realizar investigaciones y para implementar sistemas de aprendizaje automático en producción en el reconocimiento de voz, la visión por computadora, la robótica, la recuperación de información, el procesamiento del lenguaje natural, la extracción de información geográfica y el descubrimiento computacional de fármacos, entre otros. La API de TensorFlow y una implementación de referencia se lanzaron como un paquete de código abierto bajo la licencia de Apache 2.0 en noviembre de 2015 y están disponibles en `www.tensorflow.org`.

El Tensorflow Lite for Microcontroller o simplemente Tensoflow Lite Micro [10] es una implementación del TensorFlow diseñado para ejecutar modelos de aprendizaje automático en microcontroladores y sistemas empotrados con pocos kilobytes de memoria disponible.

### 2.3.1. Aprendizaje automático en microcontroladores

Un microcontrolador es un circuito integrado programable. Está compuesto de varios bloques funcionales que cumplen una tarea específica. Un microcontrolador incluye tres unidades funcionales principales: unidad central de procesamiento, memoria y periféricos de entrada/salida.

Al llevar el aprendizaje automático a pequeños microcontroladores es posible aumentar la inteligencia de miles de millones de dispositivos utilizados para el consumo diario, incluidos electrodomésticos y dispositivos de Internet de las cosas, sin depender de hardware costoso o conexiones confiables a Internet, que a menudo están sujetas al ancho de banda y restricciones de energía y da como resultado una alta latencia. Así mismo, ayuda a preservar la privacidad, ya que ningún dato sale del dispositivo.

### 2.3.2. Plataformas soportadas

Tensoflow Lite Micro está escrito en lenguaje orientado a objetos C++ 11 y requiere de una plataforma de hardware de 32 bits. Así mismo se ha probado en microcontroladores basados en la arquitectura ARM Cortex-M, así como otras arquitecturas incluyendo la RISC-V [3]

Para interés del presente trabajo interesa el soporte a:

**QEMU ARM Cortex-M3** Ejecución en una simulación en un sistema en chip con un procesador ARM Cortex-M3 sobre un sistema operativo de escritorio como por ejemplo GNU/Linux.

**nRF52840-DK** Tarjeta de desarrollo con un SoC del fabricante Nordic Semiconductor con un procesador ARM Cortex-M4F.

### 2.3.3. Ejemplos básicos de TensorFlow Lite Micro

**Hello World:** Demuestra los conceptos básicos absolutos del uso de TensorFlow Lite para microcontroladores.

**Micro speech:** Captura audio con un micrófono para detectar las palabras en inglés "yes" y "no".

**Magic wand:** Captura datos de un acelerómetro para clasificar diferentes tipos de gestos físicos.

**Person detection:** Captura datos de una cámara y detectar la presencia o ausencia de una persona.

### 2.3.4. Flujo de trabajo

Los siguientes pasos son requeridos para deplegar y ejecutar un modelo de TensorFlow en un microcontrolador:

1. Entrenar un modelo:
   1.1. Generar un modelo pequeño de TensorFlow que se ajuste al dispositivo destino y a la vez contenga las operaciones soportadas.
   1.2. Convertir a un modelo de TensorFlow Lite utilizando el convertidor para TensorFlow Lite.
   1.3. Convertir a una matriz de bytes en lenguaje C utilizando herramientas estándar para almacenar en memoria de programa en el dispositivo donde se va a desplegar.
2. Ejecutar la inferencia en el dispositivo utilizando la biblioteca de C++ y procesar los resultados.

### 2.3.5. Limitaciones

TensorFlow Lite para microcontroladores está diseñado para las limitaciones específicas presentes en los microcontroladores. Deben tenerse en cuenta las siguientes limitaciones:

1. Compatibilidad con un subconjunto limitado de operaciones de TensorFlow.
2. Soporte para un conjunto limitado de dispositivos.
3. API de C++ de bajo nivel que requiere administración de memoria manual.
4. El entrenamiento en el dispositivo no es compatible.

## 2.4. Proyecto Zephyr

El proyecto Zephyr [5] es un sistema operativo de tiempo real (RTOS), escalable de código abierto el cual soporta múltiples arquitecturas de hardware incluidas ARC, ARM, RISC-V y X86.

El modelo de desarrollo del Proyecto Zephyr[9] es diferente del que se usa típicamente en el software propietario, porque el Zephyr *RTOS* es un proyecto de código abierto bajo el paraguas de la Fundación Linux. Como tal, se desarrolla de manera independiente por múltiples colaboradores, y está gobernado por el Comité Directivo Técnico y la Junta de Zephyr.

Zephyr OS es modular, los desarrolladores pueden desarrollar una solución según sus necesidades.

La lista de tarjetas de desarrollo soporta es extensa y se encuentra en [7]. Como parte del proyecto Zephyr se incluye el soporte a Tensorflow Lite Micro así como los ejemplos básicos mencionados anteriormente en la Sección 2.3.3. El proyecto Zephyr cuenta con soporte de la tarjeta de desarrollo Digilent Arty para ambos softcore ARM Cortex-M1 y Cortex-M3, para mayor información revisar [6].

## 2.5. Nordic Semiconductor nRF52840DK

El nRF52840 DK [8] es un kit de desarrollo de placa única versátil para Bluetooth Low Energy, Bluetooth mesh, Thread, Zigbee, 802.15.4, ANT y aplicaciones patentadas de 2.4 GHz en el SoC nRF52840 (ARM Cortex-M4F).

La tarjeta facilita el desarrollo de aplicaciones aprovechando todas las características del SoC nRF52840. Incluye una antena NFC que permite el uso del periférico de etiqueta NFC-A. Todos los GPIO están disponibles a través de conectores de borde y encabezado. Cuatro botones y cuatro LED simplifican la entrada y salida hacia y desde el SoC. Una memoria externa incorporada está conectada al periférico QSPI en el SoC nRF52840.

Esta plataforma es compatible con Arduino Uno Revisión 3, lo que permite conectar tarjetas de expansión de terceros con facilidad. Cuenta con un depurador SEGGER J-Link integrado que permite programar y depurar tanto el SoC integrado como los objetivos externos a través del encabezado de depuración. También interactúa directamente con el Power Profiler Kit II.

La tarjeta de desarrollo generalmente se alimenta con USB, pero puede ser alimentado por una amplia gama de fuentes, dentro del rango de suministro de 1,7 a 5,0 V. Además de USB, puede alimentarse con una fuente externa, pero también incluye un soporte de batería CR2032 y un conector de batería Li-Po, para pruebas en campo. El consumo de corriente se puede medir utilizando los pines de medición de corriente dedicados.

En la Figura 2.5 se muestra la tarjeta de desarrollo nRF52840DK.

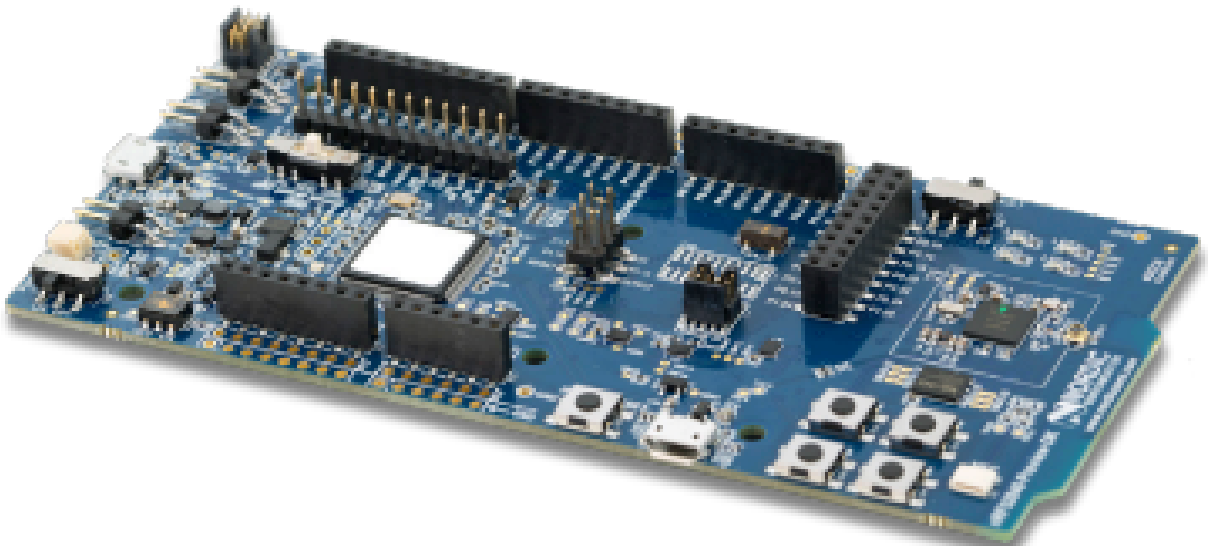

**Figura 2.5:** Tarjeta de desarrollo nRF5284DK (tomado de [8]).

## 2.6. Desarrollo de aplicaciones para procesadores softcore en FPGA utilizando las herramientas de Xilinx

En el Capítulo 1 se introdujo el diagrama de flujo general para el desarrollo de aplicaciones de aprendizaje automático en FPGA (Ver figura 1.1). En esta sección se detalla el flujo de trabajo utilizado con las herramientas del fabricante Xilinx.

En la Figura 2.6 se muestra el diagrama de flujo general utilizando las herramientas de Xilinx. El flujo comienza con el diseño de bloques para generar la plataforma de hardware con un procesador softcore como núcleo, utilizando además los bloques IP disponibles por parte del fabricante y, de ser necesario, bloques desarrollados a la medida utilizando un lenguaje de descripción de hardware. Por ejemplo, Verilog, VHDL, entre otros.

Una vez que se ha definido la plataforma de hardware, es necesario generar el archivo XSA ("Xilinx Shell Archive"), el cual contiene el bitstream con la síntesis del diseño programable en un FPGA. El archivo XSA es un insumo para la generación de los archivos del BSP ("Board Support Package"), en el cual se encuentran los controladores necesarios para la programación del software cuyo destino es la plataforma de hardware diseñada. Este BSP contiene únicamente los controladores de aquellos componentes presentes en el diseño de hardware.

En la etapa de desarrollo de software se utilizan los controladores del BSP, bibliotecas específicas de la arquitectura del procesador softcore, además de las bibliotecas estándar de C/C++ y el código fuente de la aplicación en específico. Se actualiza el bitstream con el software generado en el banco de memoria destinado para el programa de software.

Una vez generado un nuevo archivo bitstream con la actualización del software, se pro-

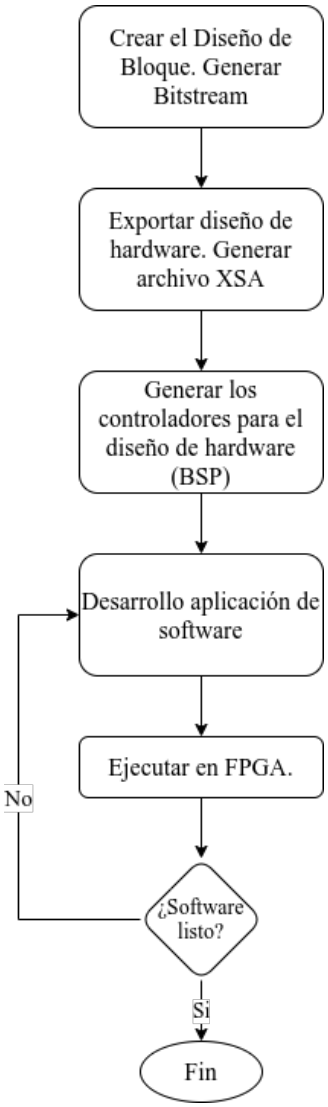


**Figura 2.6:** Diagrama de flujo general de desarrollo FPGA del fabricante Xilinx.

grama el FPGA y se prueba la aplicación de software. En el caso de cumplir con los requisitos del software, se termina el bucle; caso contrario, se itera a partir del desarrollo del software sin alterar la plataforma de hardware y/o modificar el BSP.

### 2.6.1. Diseño de bloques en Vivado

En la Tabla 2.5 se muestran los módulos IP necesarios para el desarrollo del presente trabajo con los periféricos disponibles en la tarjeta Nexys-A7. En la Tabla 2.6 se muestran los módulos IP internos necesarios para conectar al procesador softcore ARM Cortex-M1, independientemente de la tarjeta de desarrollo utilizada.

**Tabla 2.5:** Módulos AXI IP Block disponibles para los periféricos de la tarjeta Nexys-A7

| AXI IP Block | Descripción |
|---|---|
| AXI UARTLITE | Interfaz UART |
| AXI GPIO | Puertos de Entrada/Salida de propósito General |
| AXI IIC | Puerto IIC |
| AXI QUAD SPI | Puerto SPI simple y quad |
| MIG_7SERIES | Controllador para memoria SDRAM |

**Tabla 2.6:** Módulos AXI IP Block disponibles para uso interno del diseño de bloques

| AXI IP Block | Descripción |
|---|---|
| AXI BRAM | Block RAM |
| AXI Interconnect | Módulo de interconexión entre el procesador softcore y los bloques IP |
| AXI Smart Connect | Módulo similar al AXI Interconnect pero mas compacto |
| Procesor System Reset | Módulo para generar múltiples señales de reset del sistema |
| Clock Wizard | Módulo para generar múltiples señales de reloj |
| Constant | Módulo para conectar un valor constante a un puerto simple o múltiple. |
| Concat | Módulo que concatena múltiples señales simples en una sola. Utilizada con las señales de interrupción externas. |

### 2.6.2. Generación de controladores (BSP)

Una vez que se han conectado los módulos necesarios y se verifica el diseño en la herramienta Vivado del fabricante Xilinx, se exporta el diseño en un archivo XSA el cual se utiliza junto a la herramienta Vitis IDE de Xilinx para generar los controladores necesarios para el diseño de hardware específico.

### 2.6.3. Actualización del bitstream

El proceso para actualizar el archivo bitstream se describe con más detalle el flujo mostrado la Figura 2.7.

En esta etapa el diseño de hardware y los controladores se mantienen sin cambios. Únicamente se crea y modifica la aplicación de software en lenguaje C/C++.

En la memoria del programa se almacena en los bloques BRAM de 36kB disponibles en el FPGA y mapeados según la síntesis del diseño. Para actualizar el contenido de los bloques BRAM es necesario un archivo MMI ("Memory Map Information").

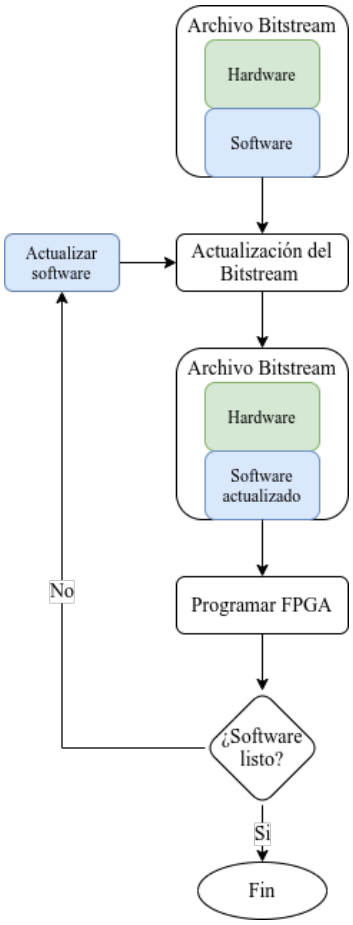


**Figura 2.7:** Diagrama de flujo para la actualización del archivo Bitstream.

### 2.6.4. Actualización de aplicación genérica del software para un procesador softcore ARM Cortex-M

Para desarrollar una aplicación de software genérica para un procesador softcore de ARM Cortex-M se necesitan los controladores del BSP, la biblioteca CMSIS ("Common Microcontroller Software Interface Standard") provisto por ARM, las bibliotecas estándar de C/C++ para la arquitectura ARM Cortex-M y el código fuente de la aplicación como tal en el archivo main.c

### 2.6.5. Actualización de aplicación de aprendizaje automático del software para un procesador softcore ARM Cortex-M

Para desarrollar una aplicación utilizando un modelo entrenado con el TensorFlow Lite Micro es necesaria la biblioteca generada con los componentes mínimos necesarios para la plataforma de hardware destino, tal y como se muestra en la Figura 2.9. Los pasos para crear dicha biblioteca se encuentran en [11] (agregar mención al repositorio) junto con el código fuente del presente trabajo. La biblioteca del TensorFlow Lite incluye el CMSIS como se muestra en la Figura 2.10

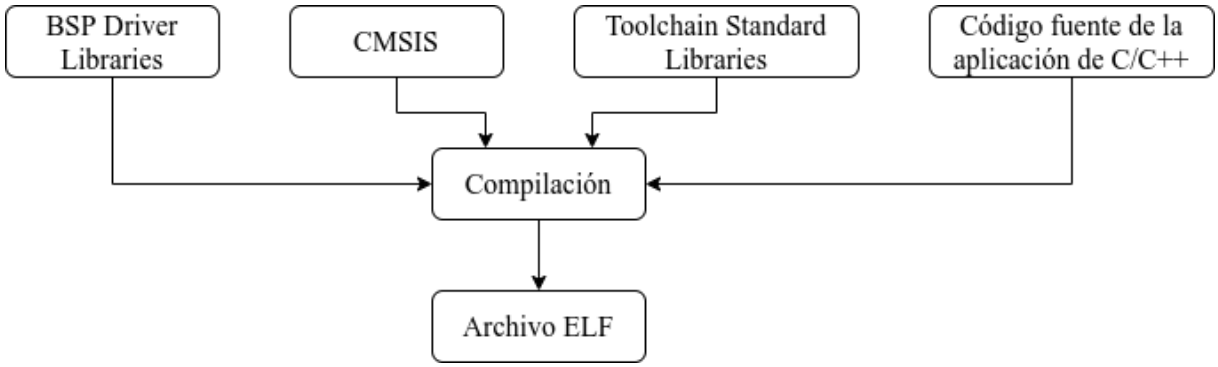


**Figura 2.8:** Diagrama de flujo para la actualización de una aplicación general.

```
tflite-micro-cmsis/
├── examples
│   ├── hello_world
│   ├── magic_wand
│   ├── micro_speech
│   └── person_detection
├── lib
├── tensorflow
│   └── lite
│       ├── c
│       ├── core
│       ├── experimental
│       ├── kernels
│       ├── micro
│       └── schema
└── third_party
    ├── cmsis
    ├── flatbuffers
    ├── gemmlowp
    ├── kissfft
    └── ruy
```

**Figura 2.9:** Árbol de directorio, parcial, de la biblioteca TensorFlow Lite para arquitectura ARM Cortex-M.

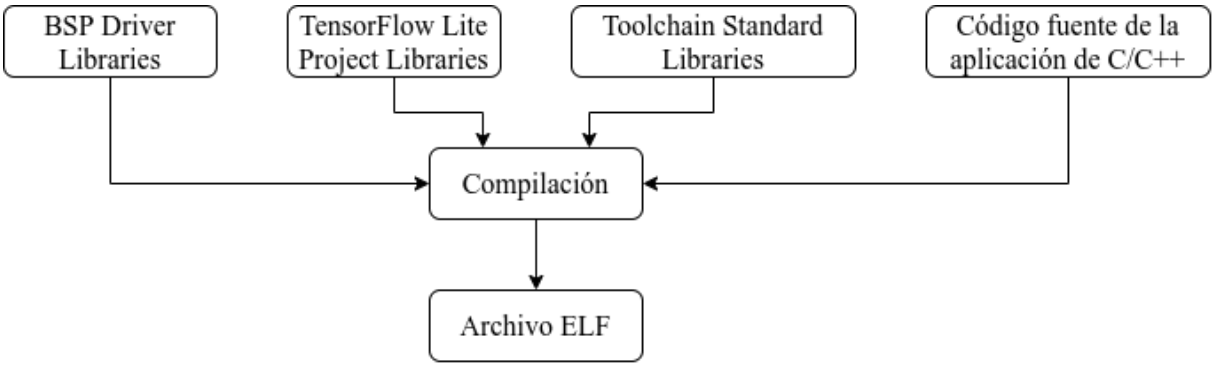


**Figura 2.10:** Diagrama de flujo para la actualización de una aplicación utilizando TensorFlow Lite.

# Capítulo 3

# Implementación de algoritmos entrenados con TensorFlow Lite Micro en SoC de arquitectura ARM Cortex-M y softcores en FPGA

## 3.1. Diseño de plataforma de hardware con procesador softcore utilizando un FPGA de Xilinx

Utilizando la metodología descrita en la Sección 2.6, y utilizando un FPGA del fabricante Xilinx, se diseñan 5 plataformas de hardware con un procesador softcore de la arquitectura ARM Cortex-M1 como núcleo.

Cada diseño incrementa la complejidad comenzando con un sistema con un periférico hasta obtener un diseño con todos los componentes periféricos de la tarjeta Nexys-A7 que cuentan con una bloque AXI IP soportado en el repositorio de Xilinx. Los componentes periféricos que no cuentan con un bloque AXI IP deben ser desarrollados a la medida en un lenguaje de descripción de hardware como Verilog o VHDL. Este desarrollo está fuera del alcance del presente proyecto y se recomienda como un posible trabajo derivado agregar el soporte a los periféricos de la tarjeta Nexys-A7 faltantes.

A continuación se presenta un breve resumen de los cinco diseños de hardware que se implementan:

**m1_nexys_a7_gpio:** Este diseño contiene un sistema mínimo con solo un componente externo GPIO y una memoria AXI BRAM dividida en memoria de programa y datos.

**m1_nexys_a7_uart:** Este diseño contiene un sistema mínimo con solo un componente externo UART y una memoria AXI BRAM dividida en memoria de programa y

datos.

**m1_nexys_a7_io:** Este diseño fusiona los dos anteriores y agrega los sensores de temperatura y acelerómetro (comunicación por protocolo SPI), disponibles en la tarjeta de desarrollo.

**m1_nexys_a7_sdram:** Este proyecto amplía el anterior agregando el módulo IP de comunicación con el DDR2 SDRAM externo. A partir de este diseño la memoria de datos se asigna en el SDRAM y se utiliza el AXI BRAM únicamente como memoria de programa.

**m1_nexys_a7:** Este diseño amplía el anterior agregando la conexión a un visualizador de siete segmentos, y a la memoria flash, además de incrementar el tamaño de la memoria de programa (AXI BRAM), al máximo posible en el *FPGA* de 512kB.

### 3.1.1. Diseño m1_nexys_a7_gpio

La Figura 3.1 muestra el diseño de bloques, para este primer diseño, el cual cuenta con un control de la señales de reloj;, un controlador de las señales de reset; el procesador softcore ARM Cortex-M1; dos bloques IP que generan una señal constante de manera que el puerto NMI del procesador está conectado a la señal de tierra (GND o señal en bajo); y el puerto CFGITCMEN[1:0] está configurado para deshabilitar el ITCM alias superior e inferior; un módulo AXI BRAM para controlar un bloque generador de memoria utilizado como memoria de programa y datos; dos instancias del bloque AXI GPIO conectando los LED, RGB LED, pulsadores e interruptores tipo DIP; por último un bloque AXI Interconnect el cual conecta el procesador softcore con los periféricos.

La Figura 3.2 muestra el mismo diseño pero resaltando únicamente las conexiones con las interfaces de entrada/salida. En esta imagen se aprecian los bloques AXI que cambian entre diseños. Por otra parte la Figura 3.3 muestra el mapa de memoria del sistema. El bloque AXI BRAM se asigna a la dirección `0x0000_0000` para que el procesador softcore cargue el programa almacenado en la memoria BRAM.

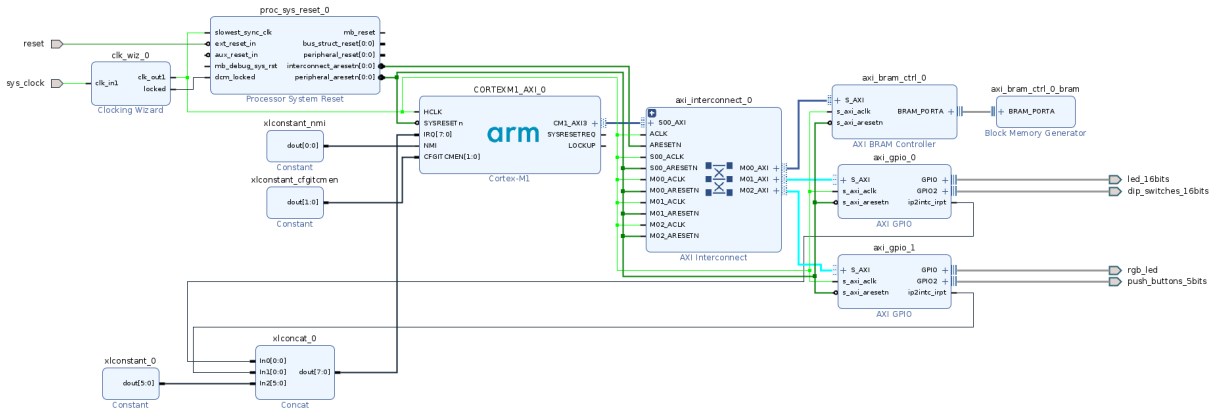


**Figura 3.1:** Diagrama de bloques del diseño m1_nexys_a7_gpio.

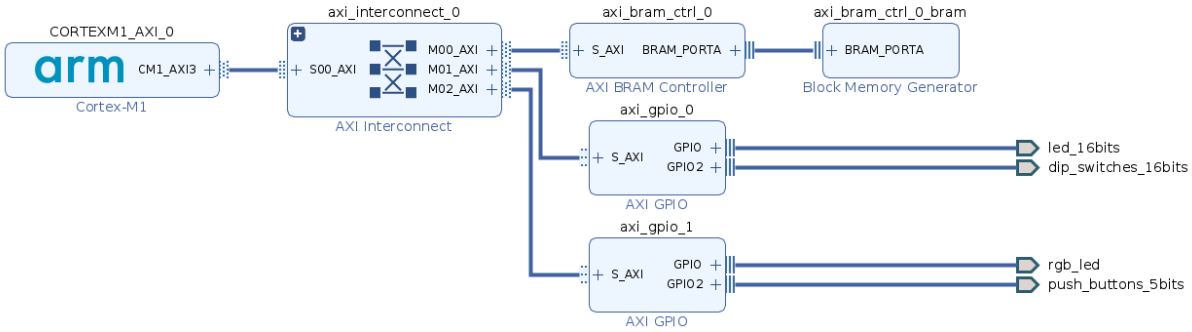


**Figura 3.2:** Diagrama de bloques del diseño m1_nexys_a7_gpio. Vista de interfaces.

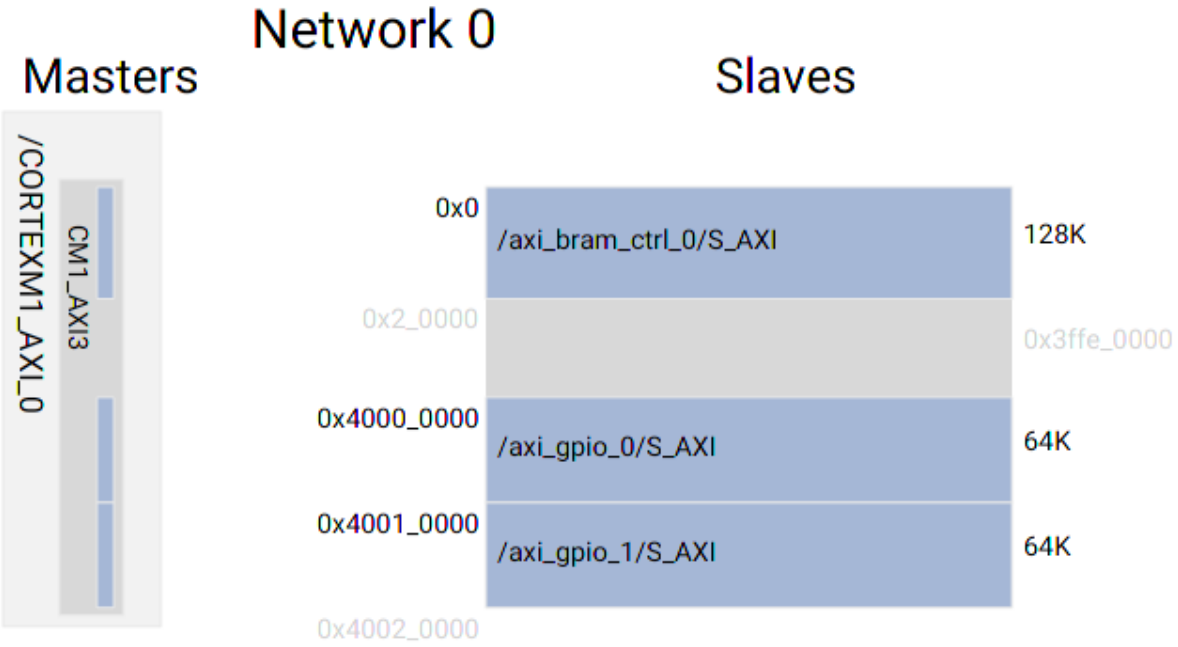


**Figura 3.3:** Mapa de memoria del diseño m1_nexys_a7_gpio.

A partir de este diseño de bloques se generan los controladores, utilizando el programa Vitis IDE. En la Figura 3.4 se muestra el árbol de directorios con los controladores necesarios para el presente diseño.

Es común en todos diseños presentados en este documento los directorios bram_v4_5, cpu_cortexm1_v1_1 y standalone_v7_3.

En la Figura 3.5 se muestra el diagrama de flujo simplificado de una aplicación que hace un barrido en los puertos LED sin utilizar interrupciones. En la Figura 3.6 se muestra el diagrama de flujo de una aplicación que extiende la anterior y le agrega soporte a las interrupciones externas de los conmutadores DIP y de los pulsadores.

```
m1_nexys_a7_gpio_hw/CORTEXM1_AXI_0/standalone_domain/bsp/CORTEXM1_AXI_0/
├── include
└── libsrc
    ├── bram_v4_5
    │   └── src
    ├── cpu_cortexm1_v1_1
    │   └── src
    ├── gpio_v4_7
    │   └── src
    └── standalone_v7_3
        └── src
```

**Figura 3.4:** Árbol de directorio del BSP para el sistema de la figura 3.1 desarrollado en Vitis IDE.

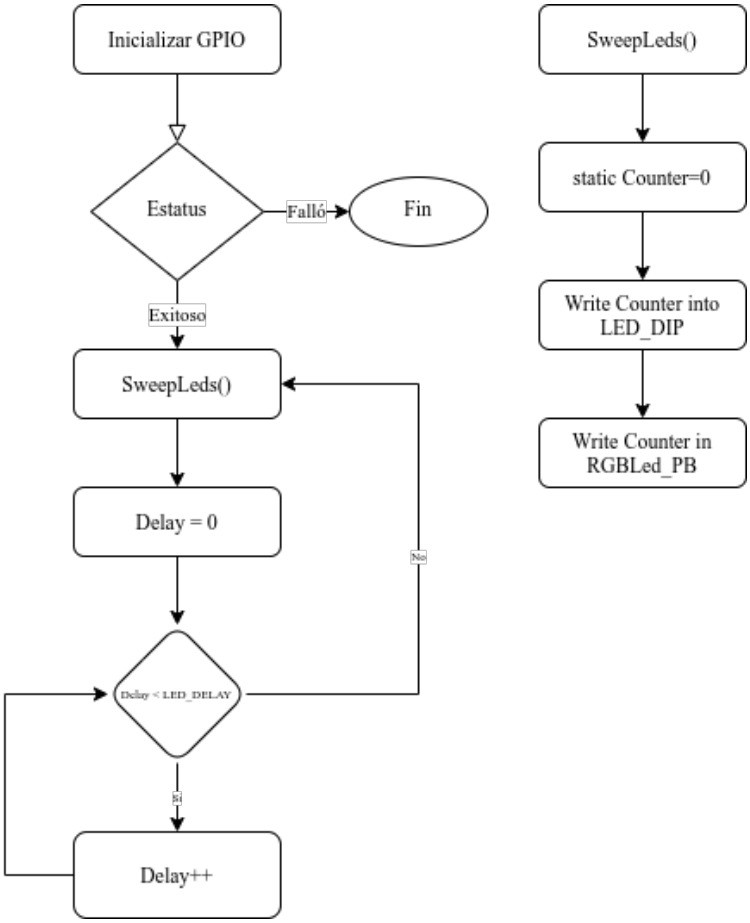


**Figura 3.5:** Diagrama de flujo simplificado de la aplicación "Blinky" de prueba para el diseño mostrado en la Figura 3.1.

### 3.1.2. Diseño m1_nexys_a7_uart

El presente diseño parte del anterior eliminando los puertos GPIO y conectando en su lugar un puerto UART tal y como se muestra en la Figura 3.7. En la Figura 3.8 se muestra el mismo diseño con la vista de interfaces.

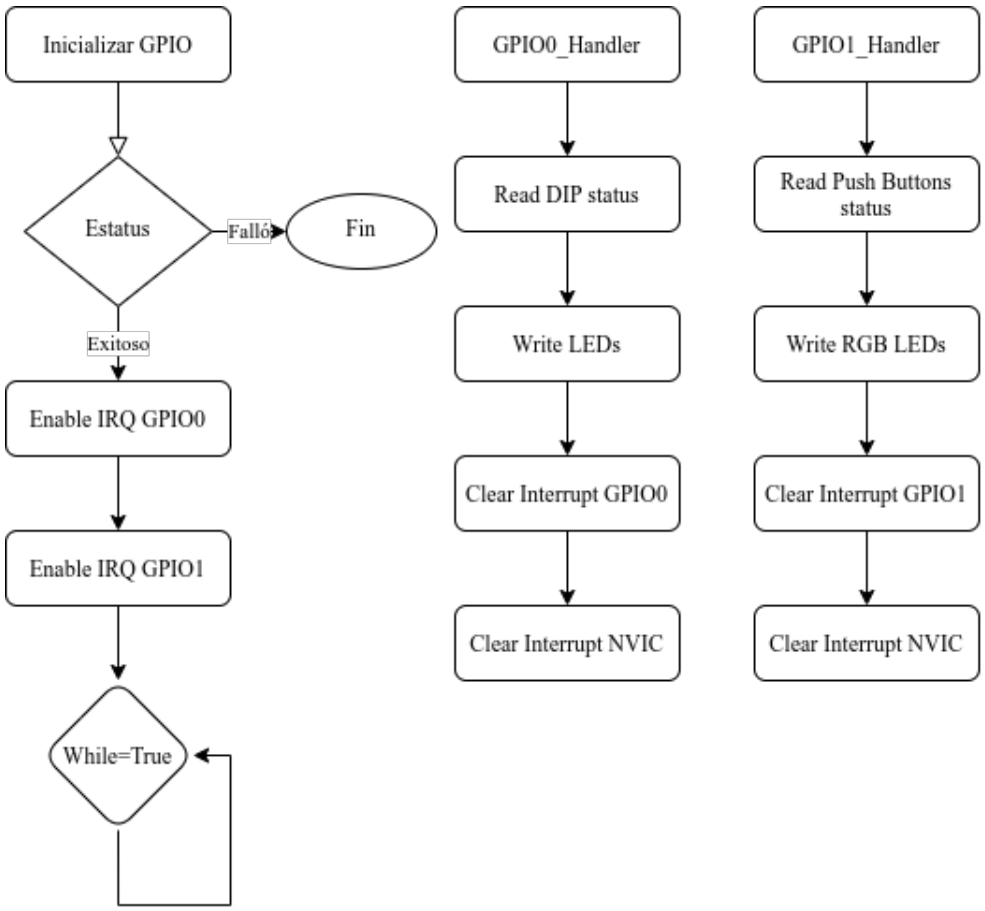


**Figura 3.6:** Diagrama de flujo simplificado de la aplicación "BlinkyInt" de prueba para el diseño mostrado en la Figura 3.1.

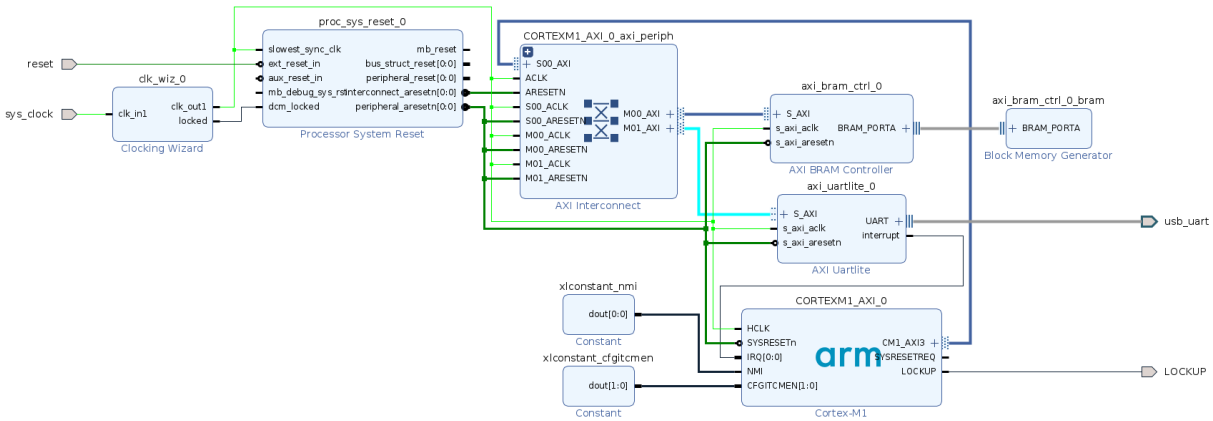


**Figura 3.7:** Diagrama de bloques del diseño m1_nexys_a7_uart.

En la Figura 3.9 se muestra el mapa de memoria para este diseño de bloques. A partir de este diseño de bloques se generan los controladores utilizando el programa Vitis IDE. En la Figura 3.10 se muestra el árbol de directorios con los controladores necesarios para el presente diseño en el cual se elimina el directorio con el controlador del GPIO y se en su lugar está el controlador del bloque AXI UART Lite v3_5.

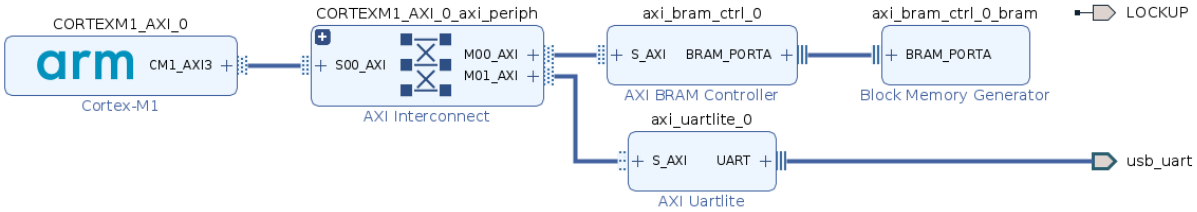


**Figura 3.8:** Diagrama de bloques del diseño m1_nexys_a7_uart vista de interfaces.

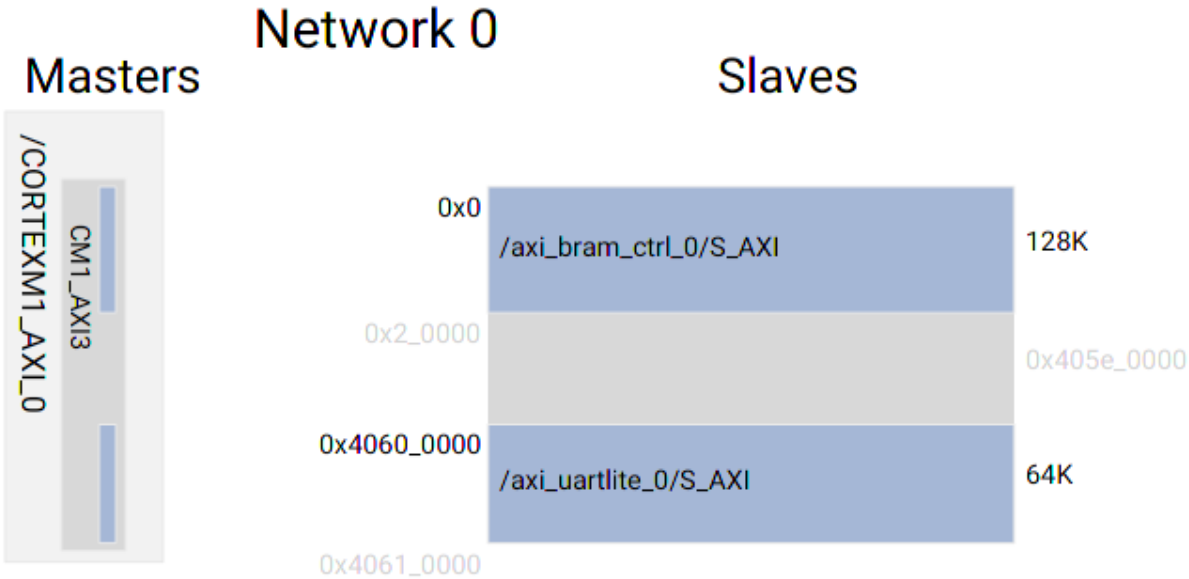


**Figura 3.9:** Mapa de memoria del diseño m1_nexys_a7_uart.

```
m1_nexys_a7_uart_hw/CORTEXM1_AXI_0/standalone_domain/bsp/CORTEXM1_AXI_0/
├── include
└── libsrc
    ├── bram_v4_5
    │   └── src
    ├── cpu_cortexm1_v1_1
    │   └── src
    ├── standalone_v7_3
    │   └── src
    └── uartlite_v3_5
        └── src
```

**Figura 3.10:** Árbol de directorio del BSP para el sistema de la figura 3.7 desarrollado en Vitis IDE.

La Figura 3.11 muestra el diagrama de flujo simplificado del programa de software utilizado para probar el correcto funcionamiento del puerto UART.

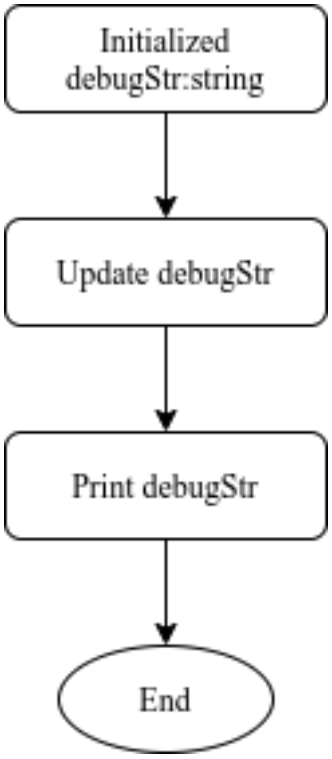


**Figura 3.11:** Diagrama de flujo de la aplicación de prueba para el diseño mostrado en la Figura 3.7.

### 3.1.3. Diseño m1_nexys_a7_io

Este diseño combina los dos diseños anteriores además de agregar los boques AXI IP para conectar el sensor de temperatura y el acelerómetro tal y como se muestra en la Figura 3.12

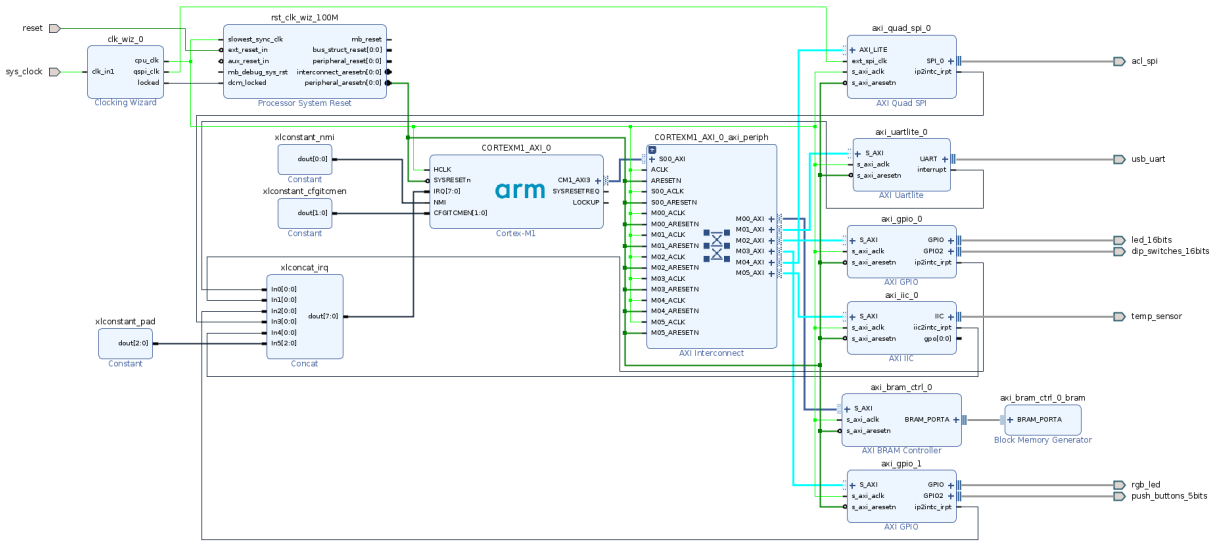


**Figura 3.12:** Diagrama de bloques del diseño m1_nexys_a7_io.

En la Figura 3.13 se muestra el diseño de bloques con las interfaces combinadas de los dos diseños anteriores además de los sensores de temperatura y acelerómetro. La Figura 3.14 muestra el mapa de memoria con la memoria BRAM, los puertos GPIO0, GPIO1, UARTLITE, IIC y QUAD_SPI.

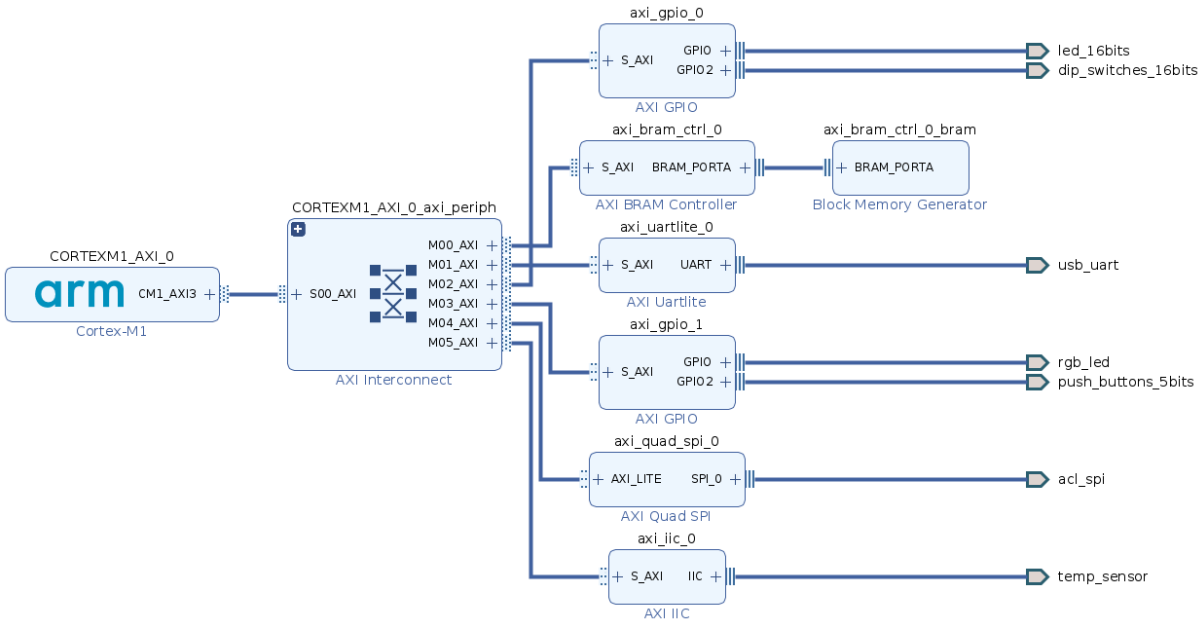


**Figura 3.13:** Diagrama de bloques del diseño m1_nexys_a7_io vista de interfaces.

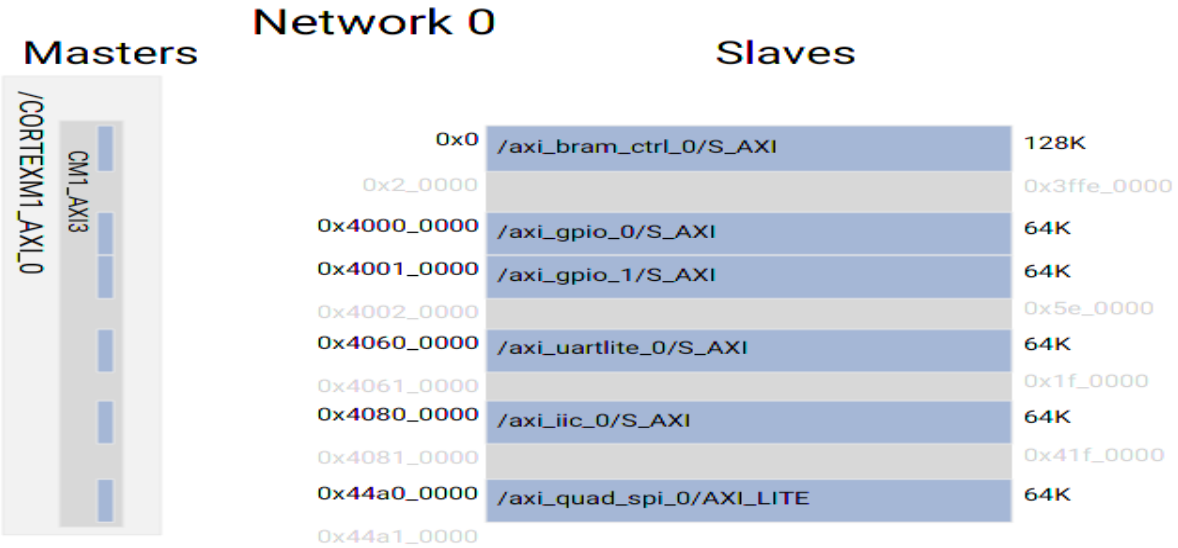


**Figura 3.14:** Mapa de memoria del diseño m1_nexys_a7_io.

En la Figura 3.15 se muestra el árbol de directorios del BSP con los controladores generados para el presente diseño agregando el soporte a los puertos IIC y SPI

En la Figura 3.16 se muestra el diagrama de flujo simplificado de la aplicación de prueba para este diseño de bloques.

```
ml_nexys_a7_io_hw/CORTEXM1_AXI_0/standalone_domain/bsp/CORTEXM1_AXI_0/
├── code
├── include
├── lib
└── libsrc
    ├── bram_v4_5
    │   └── src
    ├── cpu_cortexm1_v1_1
    │   └── src
    ├── gpio_v4_7
    │   └── src
    ├── iic_v3_7
    │   └── src
    ├── spi_v4_7
    │   └── src
    ├── standalone_v7_3
    │   └── src
    └── uartlite_v3_5
        └── src
```

**Figura 3.15:** Árbol de directorio del BSP para el sistema de la figura 3.12 desarrollado en Vitis IDE.

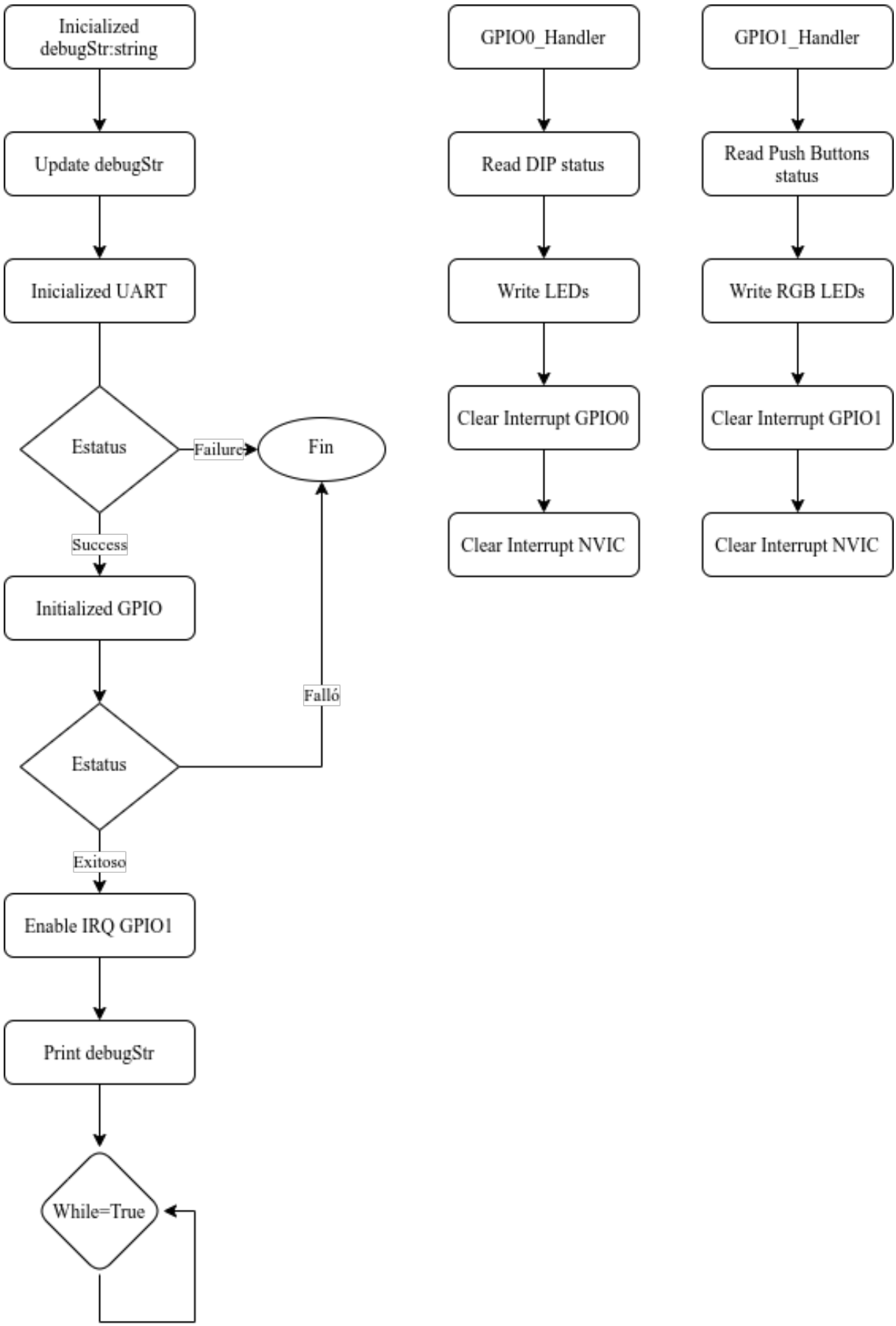


**Figura 3.16:** Diagrama de flujo de la aplicación de prueba para el diseño mostrado en la Figura 3.12.

### 3.1.4. Diseño m1_nexys_a7_sdram

El diseño mostrado en la Figura 3.17 extiende el diseño mostrado en la sección anterior agregando el bloque AXI MIG para conectar la memoria SDRAM externa al FPGA y disponible en la tarjeta de desarrollo Nexys-A7. La Figura 3.18 muestra el presente diseño en vista de interfaces.

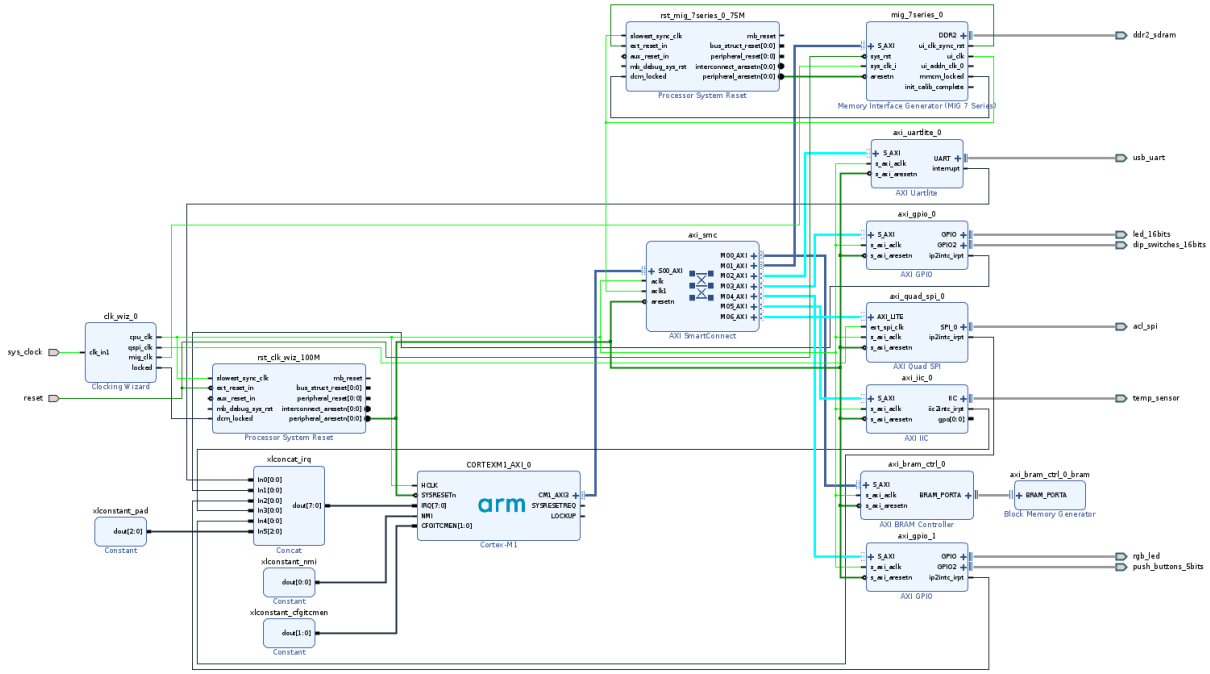


**Figura 3.17:** Diagrama de bloques del diseño m1_nexys_a7_sdram.

El mapa de memoria se extiende para agregar la memoria SDRAM en la dirección 0x6000_0000 como memoria de programa tal y como se muestra en la Figura 3.19. La Figura 3.20 muestra el árbol de directorios del BSP con los controladores del presente diseño el cual agrega soporte para el bloque mig_7series_v2_2.

La diferencia en la aplicación utilizada en el diseño **m1_nexys_a7_io** y la aplicación utilizada en el diseño **m1_nexys_a7_sdram** está en el archivo del enlazador donde se define el puntero de la pila y el espacio de memoria para HEAD y STACK en la memoria SDRAM, en lugar de dividir la memoria BRAM. Así la memoria BRAM se utiliza exclusivamente para instrucciones del programa.

### 3.1.5. Diseño m1_nexys_a7

El sistema de bloques del diseño final agrega un bloque GPIO conectando el puerto de visualizador LED de 7 segmentos tal y como se muestra en la figura 3.21. Para este diseño se amplia la memoria BRAM a un valor de 512kB, el cual es el tamaño máximo que soporta la FPGA de la tarjeta Nexys-A7(ver en la Figura 3.23).

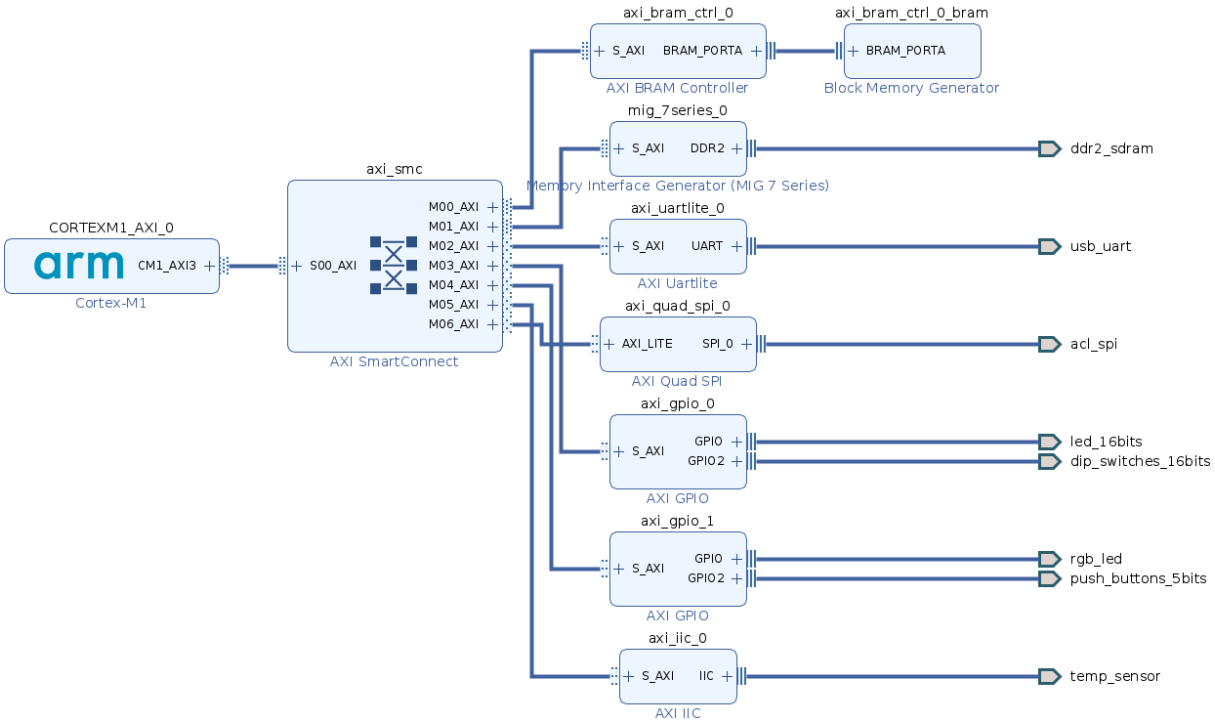


**Figura 3.18:** Diagrama de bloques del diseño m1_nexys_a7_sdram vista de interfaces.

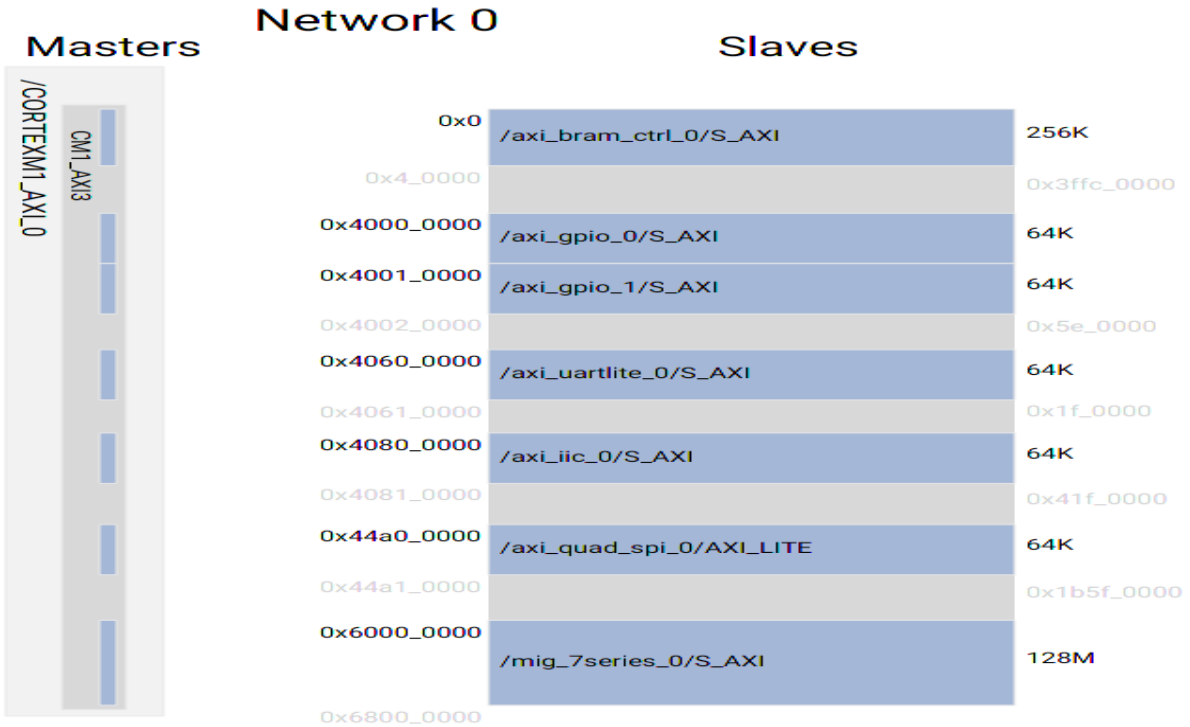


**Figura 3.19:** Mapa de memoria del diseño m1_nexys_a7_sdram.

```
ml_nexys_a7_sdram_hw/CORTEXM1_AXI_0/standalone_domain/bsp/CORTEXM1_AXI_0/
├── include
└── libsrc
    ├── bram_v4_5
    │   └── src
    ├── cpu_cortexm1_v1_1
    │   └── src
    ├── gpio_v4_7
    │   └── src
    ├── iic_v3_7
    │   └── src
    ├── mig_7series_v2_2
    │   └── src
    ├── spi_v4_7
    │   └── src
    ├── standalone_v7_3
    │   └── src
    └── uartlite_v3_5
        └── src
```

**Figura 3.20:** Árbol de directorio del BSP para el sistema de la figura 3.17 desarrollado en Vitis IDE.

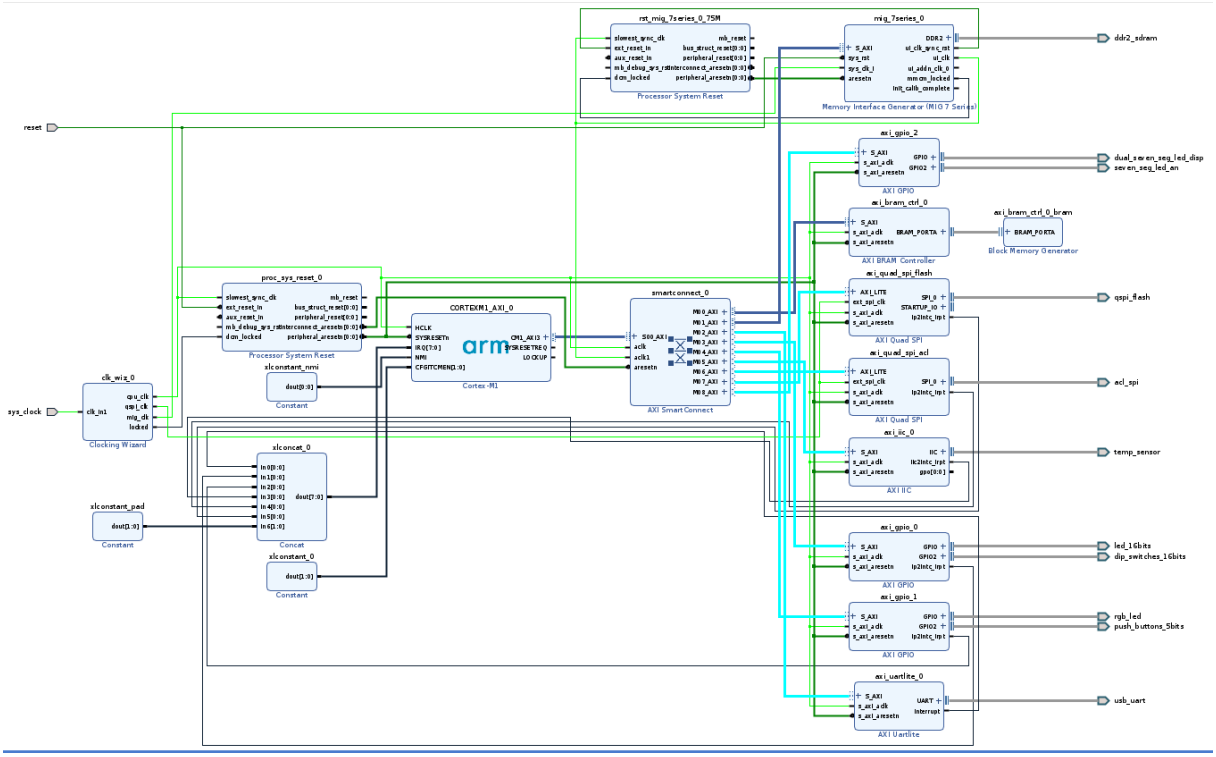


**Figura 3.21:** Diagrama de bloques del sistema final.

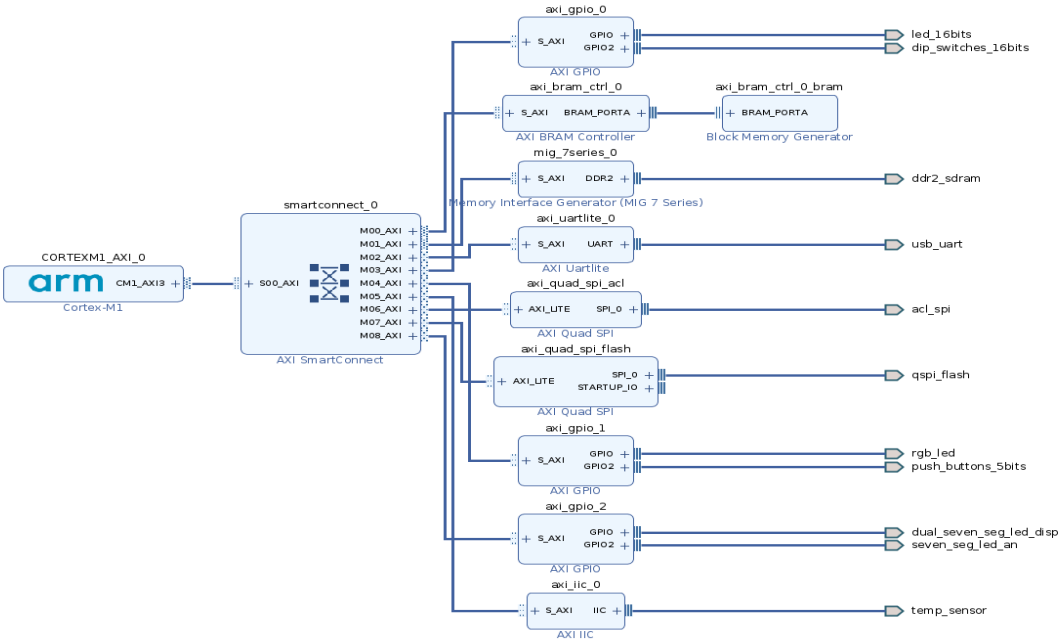


**Figura 3.22:** Diagrama de bloques del sistema final m1_nexys_a7 vista de interfaces.

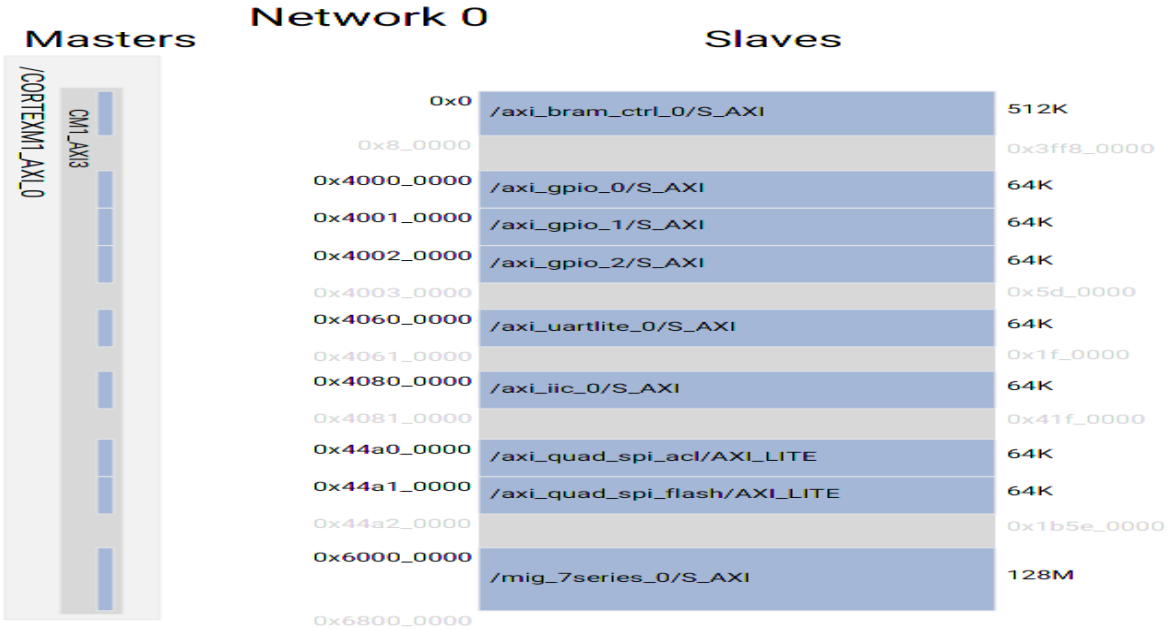


**Figura 3.23:** Mapa de memoria del sistema final m1_nexys_a7.

```
ml_nexys_a7_hw/CORTEXM1_AXI_0/standalone_domain/bsp/CORTEXM1_AXI_0/
├── include
└── libsrc
    ├── bram_v4_5
    │   └── src
    ├── cpu_cortexm1_v1_1
    │   └── src
    ├── gpio_v4_7
    │   └── src
    ├── iic_v3_7
    │   └── src
    ├── mig_7series_v2_2
    │   └── src
    ├── spi_v4_7
    │   └── src
    ├── standalone_v7_3
    │   └── src
    └── uartlite_v3_5
```

**Figura 3.24:** Árbol de directorio del BSP para el sistema de la figura 3.21 desarrollado en Vitis IDE.

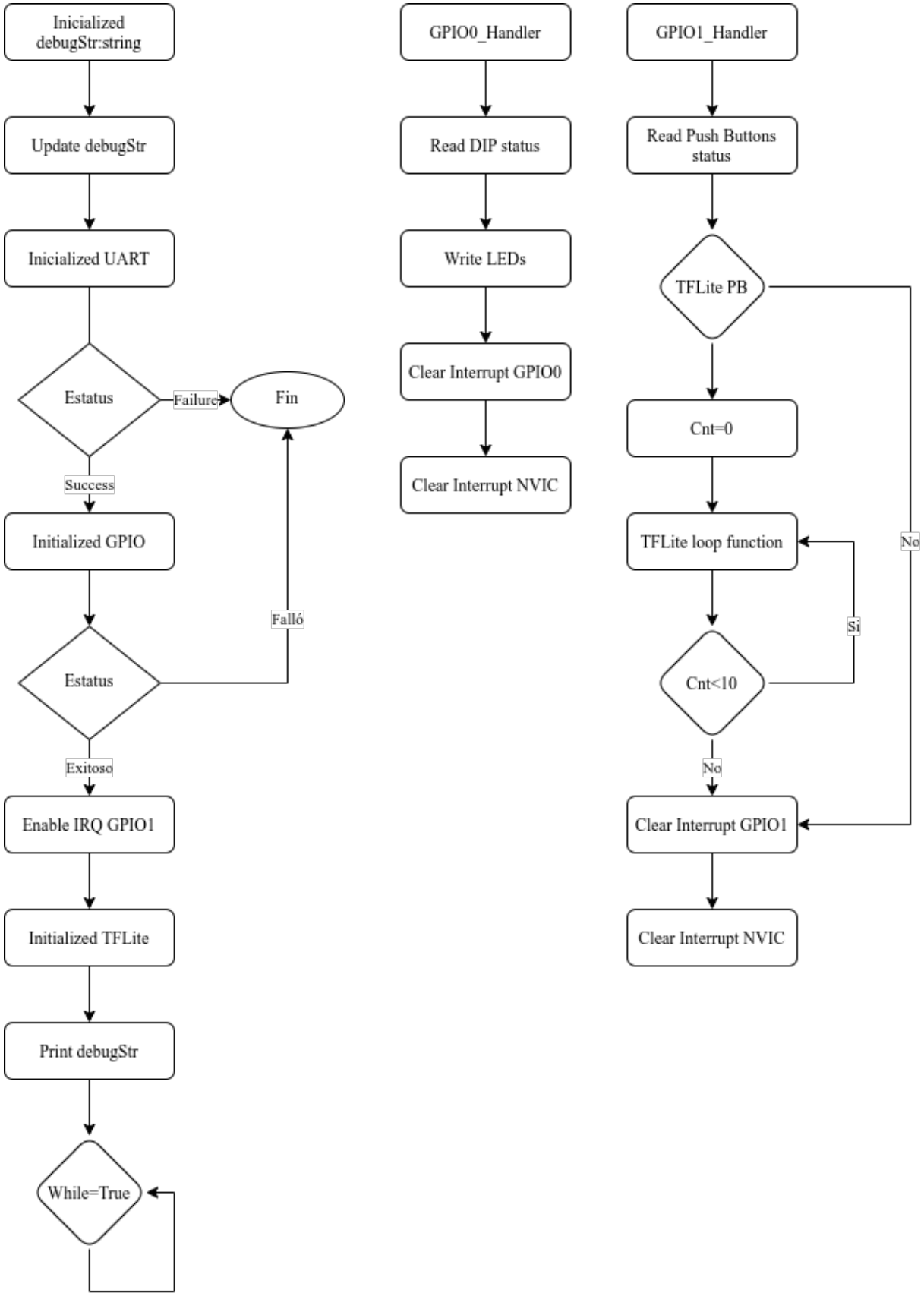


**Figura 3.25:** Diagrama de flujo de la aplicación de prueba para el diseño mostrado en la Figura 3.21.

# Capítulo 4

# Resultados y análisis

La Tabla 4.1 muestra la comparación de la huella de las aplicaciones de prueba para cada diseño de hardware descrito en el capítulo 3.1.

La huella de la aplicación hello_world del TensorFlow Lite Micro es mayor debido al tamaño de los datos del modelo y la biblioteca del TensorFlow Lite Micro.

**Tabla 4.1:** Huella de las aplicaciones de prueba para cada diseño de hardware

| Diseño de hardware | Nombre de la aplicación | text (kB) | data (kB) | bss (kB) | dec (kB) | hex (kB) |
|---|---|---|---|---|---|---|
| m1_nexys_a7_gpio | BlinkInt | 2520 | 160 | 212 | 2892 | b4c |
| m1_nexys_a7_uart | HelloWorld_Int | 8484 | 144 | 304 | 8932 | 22e4 |
| m1_nexys_a7_io | HelloWorldIO | 7968 | 176 | 332 | 8476 | 211c |
| m1_nexys_a7_sdram | HelloWorldSDRAM | 10232 | 176 | 332 | 10740 | 29f4 |
| m1_nexys_a7 | Hello_World_TFLM | 232988 | 600 | 7844 | 241432 | 3f18 |

La Tabla 4.2 muestra los resultados del modelo ejecutado en cada uno de los tres escenarios: QEMU, nRF52840-DK y con el procesador softcore en un FPGA. Además muestra valor ideal de la función sinusoidal.

**Tabla 4.2:** Resultados del modelo entrenado en TensorFlow Lite.

| **Valor de entrada** | **Resultado provisto por el modelo** | | | **Resultado ideal** |
|---|---|---|---|---|
| | QEMU | nRF52840 | FPGA | |
| 5.877D-39 | 5.877D-39 | 5.877D-39 | 0.0 | 5.877D-39 |
| 0.3141591 | 0.3727693 | 0.3727693 | 0.372768 | 0.3090169 |
| 0.6283183 | 0.5591539 | 0.5591539 | 0.559152 | 0.5877851 |
| 0.9424775 | 0.838731 | 0.838731 | 0.838729 | 0.8090168 |
| 1.2566366 | 0.9658115 | 0.9658115 | 0.965809 | 0.9510564 |
| 1.5707957 | 1.0420598 | 1.0420598 | 1.042057 | 1. |

*Continúa en la siguiente página*

| Valor de entrada | Resultado provisto por el modelo | | | Resultado ideal |
|---|---|---|---|---|
| 1.8849551 | 0.9573396 | 0.9573396 | 0.957337 | 0.9510567 |
| 2.1991134 | 0.8217871 | 0.8217871 | 0.821785 | 0.8090179 |
| 2.5132732 | 0.533738 | 0.533738 | 0.533736 | 0.587786 |
| 2.8274318 | 0.2372169 | 0.2372169 | 0.237216 | 0.3090185 |
| 3.1415914 | 0.008472 | 0.008472 | 0.008472 | 0.0000013 |
| 3.4557506 | -0.3049931 | -0.3049931 | -0.304992 | -0.3090157 |
| 3.7699102 | -0.533738 | -0.533738 | -0.533736 | -0.5877845 |
| 4.0840684 | -0.7794269 | -0.7794269 | -0.779425 | -0.8090158 |
| 4.3982268 | -0.9658115 | -0.9658115 | -0.965809 | -0.9510556 |
| 4.7123864 | -1.1098361 | -1.1098361 | -1.109833 | -1. |
| 5.0265464 | -0.9827555 | -0.9827555 | -0.982753 | -0.9510571 |
| 5.3407052 | -0.7455386 | -0.7455386 | -0.745537 | -0.8090184 |
| 5.6548636 | -0.533738 | -0.533738 | -0.533736 | -0.5877878 |
| 5.9690232 | -0.3558253 | -0.3558253 | -0.355824 | -0.3090197 |

La Tabla 4.3 muestra el error cuadrático medio (ver ecuación 4.1), y el error medio absoluto (ver ecuación 4.2), de los resultados mostrado en la Tabla 4.2 mencionada anteriormente.

$$ECM = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{4.1}$$

$$EAM = \frac{1}{n}\sum_{i=1}^{n}\left|\hat{Y}_i - Y_i\right| \tag{4.2}$$

**Tabla 4.3:** Comparación del error cuadrático medio y error absoluto medio del ejemplo de TensorFlow Lite Micro en los tres escenarios probados

| Escenario | Error Cuadrático Medio | Error Absoluto Medio |
|---|---|---|
| qemu_cortex_m3 | 0.0021173 | 0.0370284 |
| nRF52840-DK | 0.0021173 | 0.0370284 |
| FPGA Softcore Cortex-M1 | 0.0021173 | 0.0370281 |

El modelo entrenado en TensorFlow Lite Micro presenta un margen de error consistente en los tres escenarios en los que se probó el ejemplo, tal y como se muestra en la Figura 4.1, donde se sobreponen los resultados obtenidos. Estos resultamos demuestran la factibilidad de ejecutar modelos de aprendizaje máquinas en procesadores softcore de arquitectura ARM Cortex-M1.

Inicialmente se planteo medir con una marca de tiempo (*timestamp*), el número de ciclos del procesador utilizados para obtener el resultado del modelo entrenado en TensorFlow Lite. Sin embargo, tal y como se muestra en la Tabla 2.3 esta funcionalidad no está

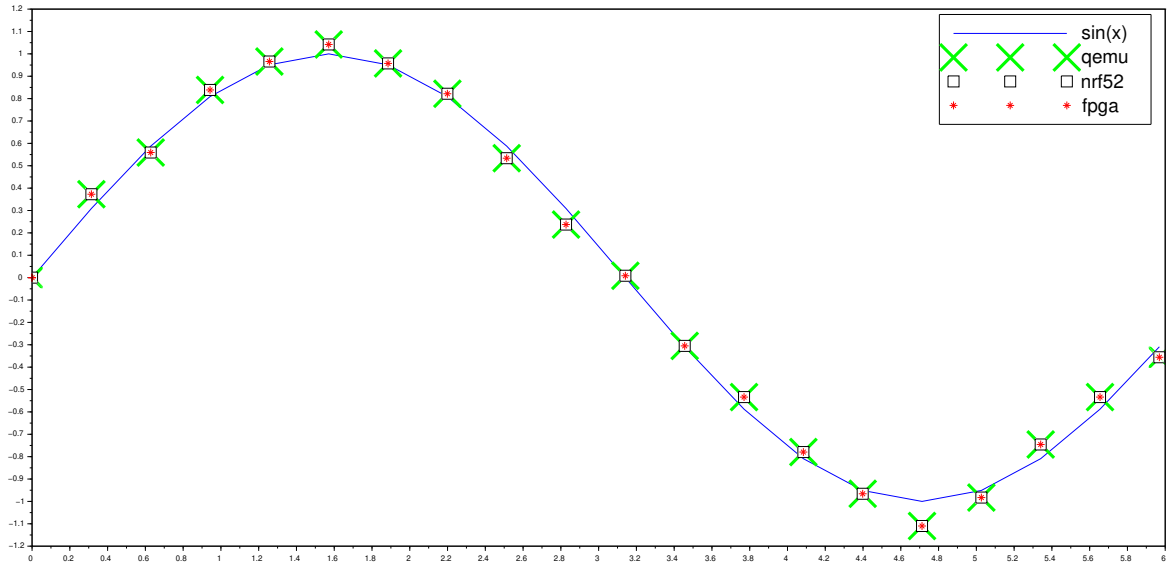


**Figura 4.1:** Comparación entre la función senoidal ideal y los resultados del modelo entrenado con TensorFlow Lite Micro ejecutado en los tres escenarios utilizados (qemu_arm_cortex_m3, nRF52840 y procesador softcore en un FPGA)

implementada en el Cortex-M1 así que no es posible obtener la medición de la marca de tiempo. Se recomienda para un trabajo derivado la implementación de un sistema con un procesador softcore ARM Cortex-M3 el cual sí tiene implementada la funcionalidad de marca de tiempo para medir el tiempo que le toma al procesador softcore obtener un resultado del modelo probado en este trabajo.

# Capítulo 5

# Conclusiones

Este trabajo propone una hoja de ruta para desarrollar aplicaciones de software en C/C++ y desplegar dichas aplicaciones en un procesador softcore ARM Cortex-M1 en modalidad de baremetal sin un sistema operativo de tiempo real. Así mismo se planteó la ejecución de un modelo de aprendizaje automático entrenado con el marco de trabajo TensorFlow Lite Micro.

Para comprobar el comportamiento del modelo de aprendizaje automático se ejecutó en tres escenarios distintos: QEMU simulando la arquitectura ARM Cortex-M3, una tarjeta de desarrollo nRF52840-DK y en un procesador softcore en un FPGA. En los tres escenarios el resultado obtenido fue consistente.

Inicialmente se planteó medir el número de ciclos de reloj necesarios para la ejecución del modelo sin embargo, la funcionalidad necesaria para tal fin no está implementada en el modelo Cortex-M1 de la familia de procesadores ARM Cortex-M. Para medir esta variable es necesario utilizar el procesador softcore ARM Cortex-M3.

## 5.1. Trabajos derivados

A continuación se muestra un listado de posibles trabajos derivados del presente proyecto.

**Agregar soporte a Serial Wire con un TAG Connector:** Crear un diseño de PCB para conectar un puerto PMOD a un dispositivo TAG Connector utilizado principalmente en el proceso de manufactura de tarjetas de desarrollo para instalar el firmware de los SoC.

**Agregar soporte a JTAG:** Ampliar el diseño de bloques para conectar el puerto JTAG del softcore ARM Cortex-M1 a un puerto PMOD de manera que se utilice un dispositivo JTAG externo para depuración del sistema.

**Probar sensores:** Crear aplicaciones de software que prueben los sensores de la tarjeta Nexys A7 como el sensor de temperatura y el acelerómetro.

**Agregar diseño a Zephyr:** Implementar el soporte necesario para una tarjeta de desarrollo a partir de un procesador softcore en un FPGA como parte del proyecto Zephyr. Agregando un programador Serial Wire (Por ejemplo un TAG Connector).

**Probar el ejemplo Magic Wand:** Probar el ejemplo del Tensorflow Lite Micro llamado Magic Wand, tanto en su versión baremetal como utilizando el proyecto Zephyr.

**Implementar un SoC con el Cortex-M3:** Replicar, ya sea solo el proyecto final o cada uno de los sistemas mínimos desarrollados, con el softcore ARM Cortex-M3.

**Implementar la medición de marca de tiempo con el Cortex-M3:** Replicar el ejemplo hello_world con un procesador softcore Cortex-M3 y medir el tiempo que consume el cálculo del modelo entrenado en TensorFlow Lite Micro.

**RISC-V:** Replicar el proyecto con un softcore de arquitectura RISC-V, como por ejemplo el E31 y E51 de SiFive o cualquier otra implementación de código abierto o cerrado que permita su ejecución en una tarjeta Nexys A7.

# Bibliografía


[1] *Cortex-M1 Technical Reference Manual*, ARM DDI0413D, ARM Limited, 2006-2008.

[2] F. K. C. Ladino, «Implementación de dispositivos ARM sobre FPGAs de Xilinx,» Tesis de mtría., Universidad de Alcalá, 2020. dirección: http://hdl.handle.net/10017/44989.

[3] M. S. Louis, Z. Azad, L. Delshadtehrani, S. Gupta, P. Warden, V. J. Reddi y A. Joshi, *Towards Deep Learning using TensorFlow Lite on RISC-V*, 2019.

[4] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu y Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. dirección: http://tensorflow.org/.

[5] Z. Project, *Zephyr*, nov. de 2021. dirección: https://www.zephyrproject.org.

[6] ——, *Zephyr Board Arty*, nov. de 2021. dirección: https://docs.zephyrproject.org/latest/boards/arm/arty/doc/index.html.

[7] ——, *Zephyr Boards*, nov. de 2021. dirección: https://docs.zephyrproject.org/latest/boards/index.html.

[8] N. Semiconductor, *nRF52840 DK Product Brief Version 1.3*, nov. de 2021. dirección: https://www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/nRF52840-DK-product-brief.pdf.

[9] ——, *Zephyr Bluetooth Low Energy Controller*, nov. de 2021. dirección: https://infocenter.nordicsemi.com/pdf/nwp_029.pdf.

[10] TensorFlow, *TensorFlow Lite Micro*, nov. de 2021. dirección: https://www.tensorflow.org/lite/microcontrollers.

[11] J. P. C. Valverde, *Repositorio del proyecto*, nov. de 2021. dirección: https://github.com/ECASLab/ARM-CortexM-NexysA7.

[12] J. Yiu, *System-on-Chip Design with Arm Cortex-M Processors*. arm Education Media, 2019, ISBN: 978-1-911531-19-7.