



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES CARRERA DE TELECOMUNICACIONES



“Diseño e implementación de aceleradores en hardware para funciones de seguridad y cifrado, utilizando bibliotecas Vitis y plataformas SoC-FPGA, y evaluando su rendimiento computacional.”

**TRABAJO DE INTEGRACIÓN CURRICULAR PREVIO A LA OBTENCIÓN DEL
TÍTULO DE INGENIERO EN TELECOMUNICACIONES**

Autor: Rodríguez Sevilla Paolo Estefano

Director del Proyecto: Ing. Navas Viera Byron Roberto, PhD.

Director de Carrera: Ing. Altamirano Carlos Daniel, PhD.



El Problema Real En Telecomunicaciones

Contexto de aplicación: 5G · IoT Masivo · Sistemas Críticos

- El procesamiento en tiempo real satura los sistemas convencionales
- Incremento de la latencia
- Consumo energético
 - La seguridad criptográfica



"El problema ya no es solo proteger los datos, sino hacerlo sin detener la red."



La Solución:

Arquitectura Propuesta: Aceleración con SoC-FPGA

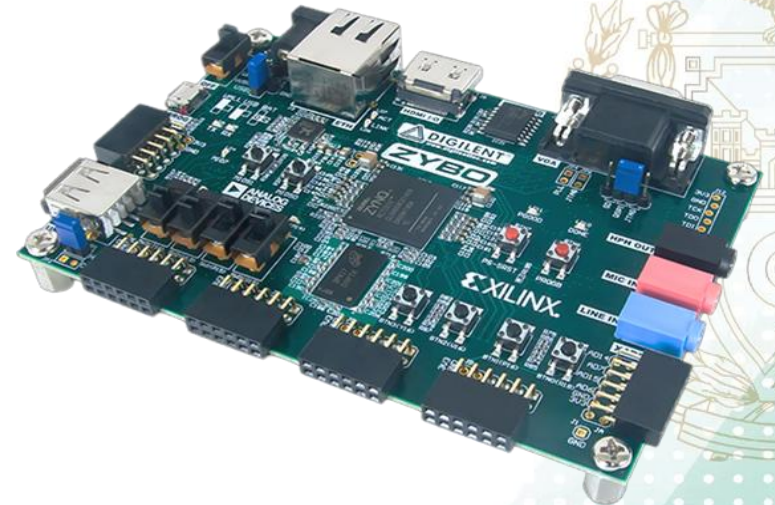
Hardware dedicado vs. Procesamiento General

CPU (GENERALISTA)

- Alta frecuencia (GHz), pero ineficiente por ciclo para tareas específicas.
- Flexible pero secuencial.
- Alto consumo energético bajo carga.

SOC - FPGA (HARDWARE PERSONALIZADO)

- Paralelismo masivo a nivel de hardware.
- Alta eficiencia por ciclo de reloj.
- Reconfigurable según la aplicación.
- Bajo consumo energético.



Objetivo y Actividades Del Proyecto

Objetivo General:

Diseño e implementación de aceleradores en hardware para funciones de seguridad y cifrado, utilizando bibliotecas Vitis y plataformas SoC-FPGA, y evaluando su rendimiento computacional.

Actividades:



Investigación y Modelado

Selección de algoritmos criptográficos y verificación con modelos en Python/C



Integración en Plataforma

Implementación de los módulos en la SoC Zynq bajo el entorno PetaLinux.



Diseño y Experimentación

Desarrollo de los aceleradores mediante Síntesis de Alto Nivel (C/C++).



Validación y Documentación

Pruebas de desempeño vs. CPU y documentación de la metodología e implementación.

Fases de Desarrollo

Modelamiento:



FASE I: MODELO DE REFERENCIA
Software en Python validado con vectores NIST.



FASE II: GENERACIÓN DE IPS
Uso de Vitis Libraries, síntesis de Alto Nivel (HLS) y empaquetado AXI4.

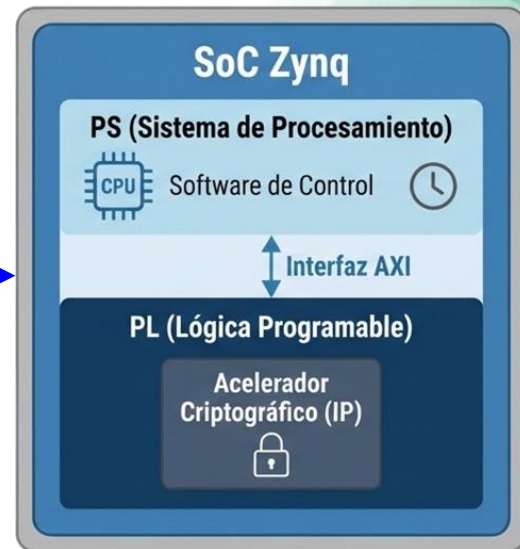
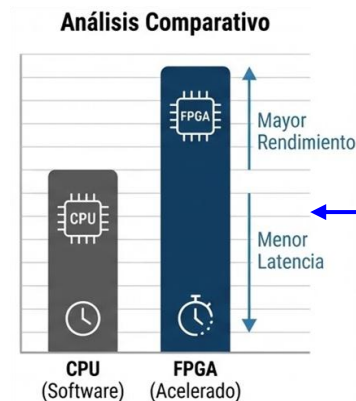


FASE III: INTEGRACIÓN & DESPLIEGUE
Conexión de las IPs, diseño en Vivado y ejecución sobre PetaLinux.



FASE IV: EVALUACIÓN DE RENDIMIENTO
Medición y análisis del desempeño computacional final.

Metodología:



Arquitectura del Sistema: Zynq-7000

Processing System (PS)

-  CPU ARM Cortex-A9.
-  Linux embebido: Gestión de archivos y plataforma de desarrollo.
-  Control de aceleradores hardware: Configuración y transferencia de datos (AXI)

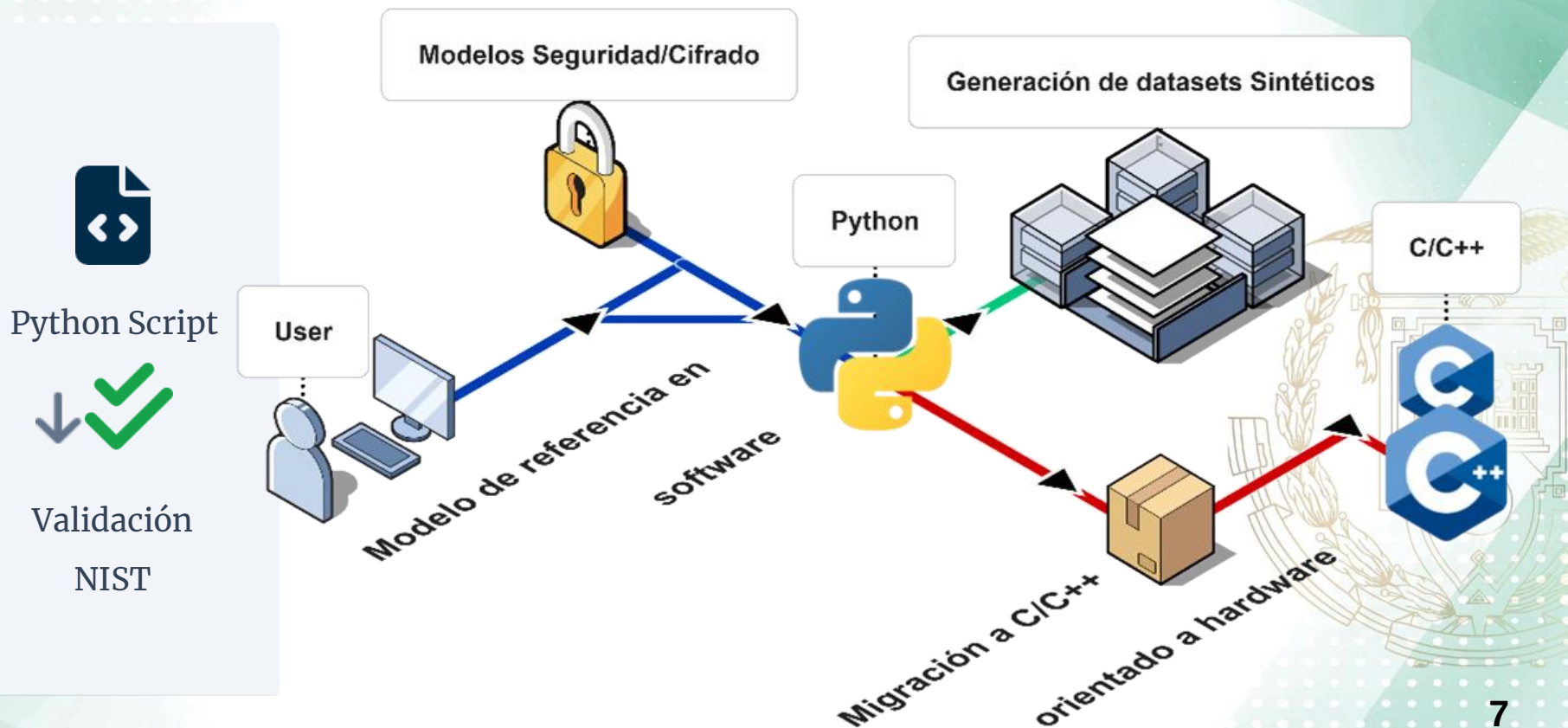
Programmable Logic (PL)

-  Aceleradores IP core:
Núcleos AES-128 y SHA-256.
-  Bloques lógicos configurables
-  Hardware personalizado:
Hardware creado a partir de VHDL y Verilog.

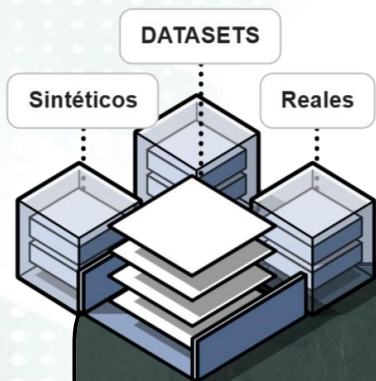


Modelamiento:

FASE I - Validación Funcional

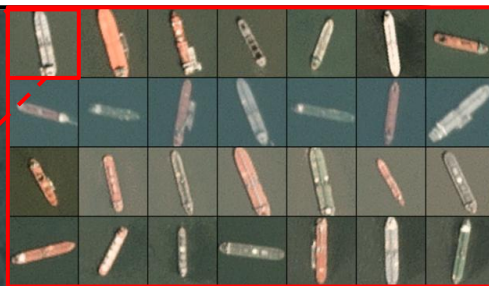


Modelamiento: FASE I - DATASETS



Ships in Satellite Imagery

Classify ships in San Francisco Bay using Planet satellite imagery



dataset_test_512MB.bin	524,288 KB
dataset_test_256MB.bin	262,144 KB
dataset_test_128MB.bin	131,072 KB
dataset_test_64MB.bin	65,536 KB
dataset_test_32MB.bin	32,768 KB
dataset_test_16MB.bin	16,384 KB
dataset_test_8MB.bin	8,192 KB
dataset_test_4MB.bin	4,096 KB
dataset_test_2MB.bin	2,048 KB



Modelamiento:

FASE I - SHA-256 Función Hash Criptográfica

Preprocesado

Añade bits al mensaje para alcanzar múltiplos de 512 bits

Motor de Compresión

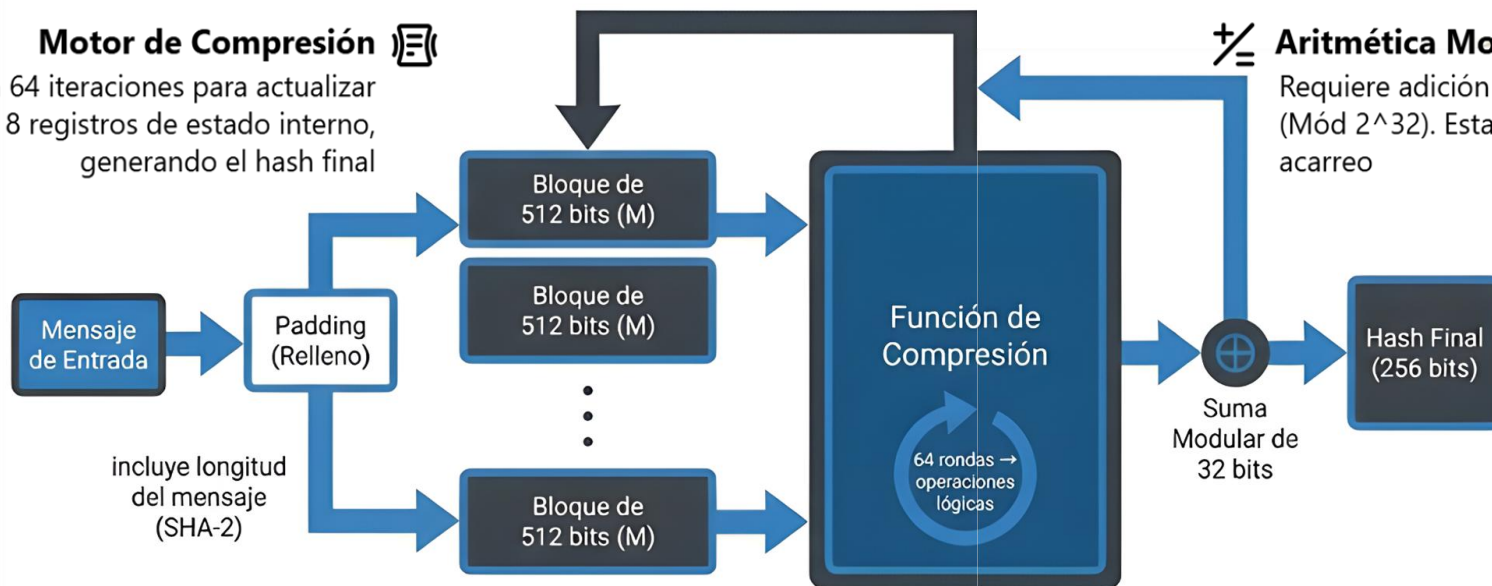
Realiza 64 iteraciones para actualizar los 8 registros de estado interno, generando el hash final

Operaciones Lógicas

Utiliza funciones lógicas y rotaciones de bits

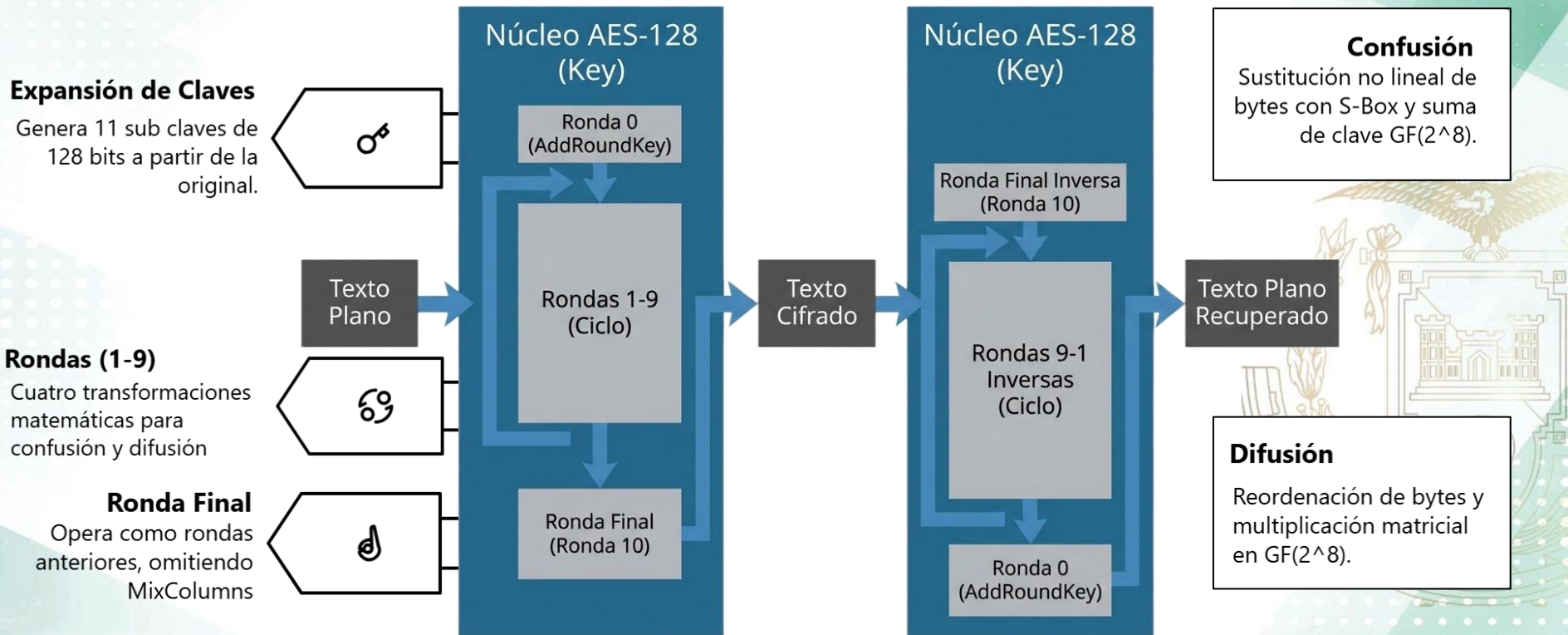
Aritmética Modular

Requiere adición binaria estándar (Mód 2^{32}). Estas sumas propagan el acarreo



Modelamiento:

FASE I - AES 128 ECB Cifrado simétrico por bloques

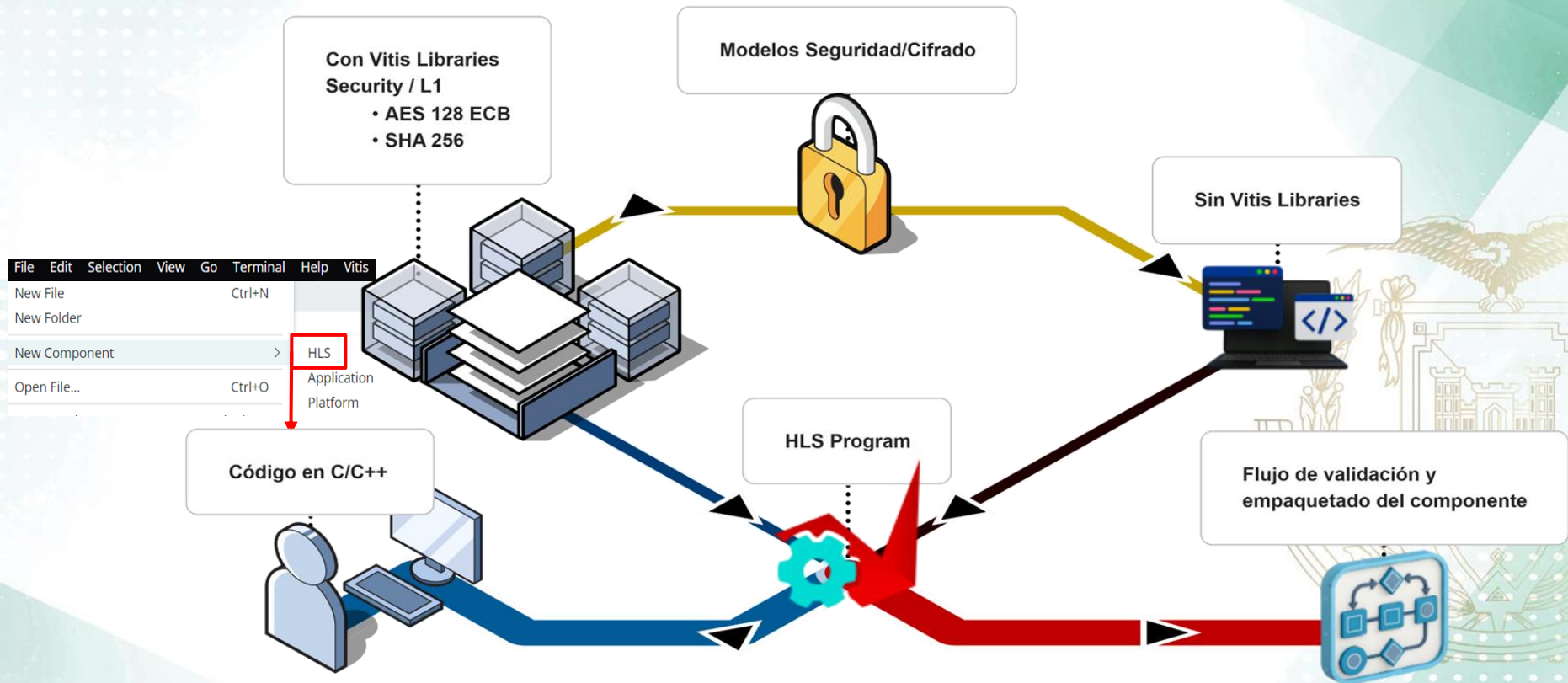


Metodología:

FASE II – Vitis HLS



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA



Metodología:

FASE II – Vitis Library



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Vitis Libraries

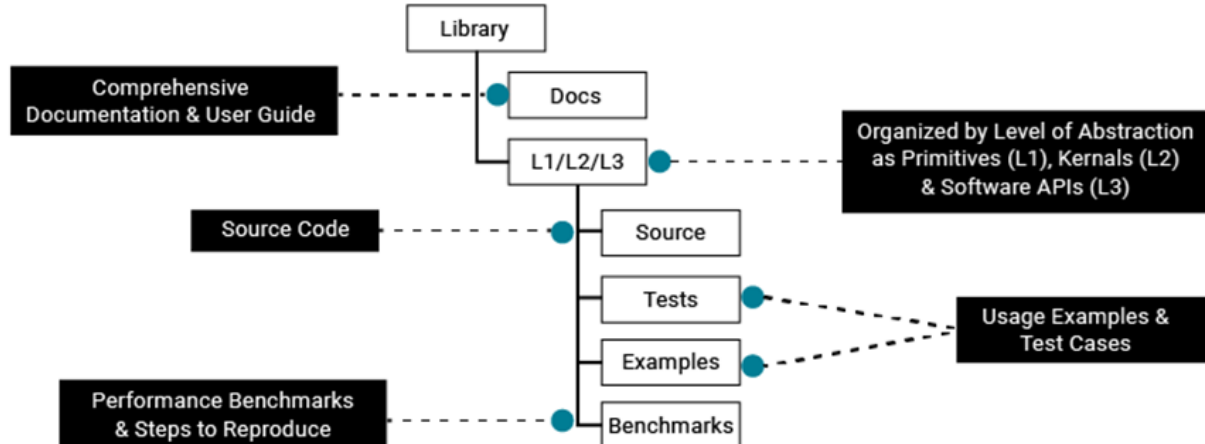
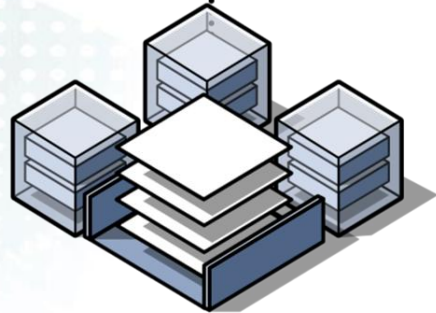
Security / L1

- AES 128 ECB
- SHA 256

```
LIB_AES_ENCRYPT > C++ hls_aes128_enc.cpp > .  
1  #include <ap_int.h>  
2  #include <hls_stream.h>  
3  #include <string.h>  
4  #include "xf_security/ecb.hpp"
```

```
LIB_SHA256_HASH > C++ hls_sha256.cpp > ...  
1  #include <ap_int.h>  
2  #include <hls_stream.h>  
3  #include <stdint.h>  
4  #include "xf_security/sha224_256.hpp"
```

Apache 2.0 License



Metodología:

FASE II – Implementación en VITIS HLS



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

```
// Depth ajustado para absorber latencias sin consumir mucha BRAM
```

```
#pragma HLS STREAM variable=plainStrm depth=64
#pragma HLS STREAM variable=cipherStrm depth=64
#pragma HLS STREAM variable=endPlainStrm depth=64
#pragma HLS STREAM variable=endCipherStrm depth=64
#pragma HLS STREAM variable=keyStrm depth=4
```

```
#pragma HLS PIPELINE II=1
```

```
    // Escribimos en Big Endian (estándar SHA)
    // range(high, low) selecciona 32 bits del total de 256
    out_mem[i] = digest.range(32 * (7 - i) + 31, 32 * (7 - i));
}
endif = end_strm.read();
}
```

```
// Interfaces de Control (AXI Lite)
```

```
#pragma HLS INTERFACE s_axilite port=key_w0 bundle=control
#pragma HLS INTERFACE s_axilite port=key_w1 bundle=control
#pragma HLS INTERFACE s_axilite port=key_w2 bundle=control
#pragma HLS INTERFACE s_axilite port=key_w3 bundle=control
#pragma HLS INTERFACE s_axilite port=length_bytes bundle=control
#pragma HLS INTERFACE s_axilite port=return bundle=control
```

```
// Usamos LUTRAM (Registros) para ser rápidos y ahorrar BRAM.
```

```
#pragma HLS STREAM variable=len_strm      depth=16
#pragma HLS STREAM variable=end_len_strm  depth=16
#pragma HLS STREAM variable=hash_strm     depth=16
#pragma HLS STREAM variable=end_hash_strm depth=16
```

Test Bench

```
// Datos de prueba (Vector NIST)
// Plaintext: 6bc1bee22e409f96e93d7e117393172a
// Key:       2b7e151628aed2a6abf7158809cf4f3c
// Expected:  3ad77bb40d7a3660a89ecaf32466ef97
```

```
unsigned char test_bytes[16] = {0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
                                0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a};
```

Metodología:

FASE II – Flujo de Validación



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

FLOW

Component ⚡ hls_aes128_enc ⚙️

✓ C SIMULATION

▷ Run ✓

⚙️ Debug

> REPORTS

✓ C SYNTHESIS

▷ Run ✓

> REPORTS

✓ C/RTL COSIMULATION

▷ Run ✓

> REPORTS

✓ PACKAGE

▷ Run ✓

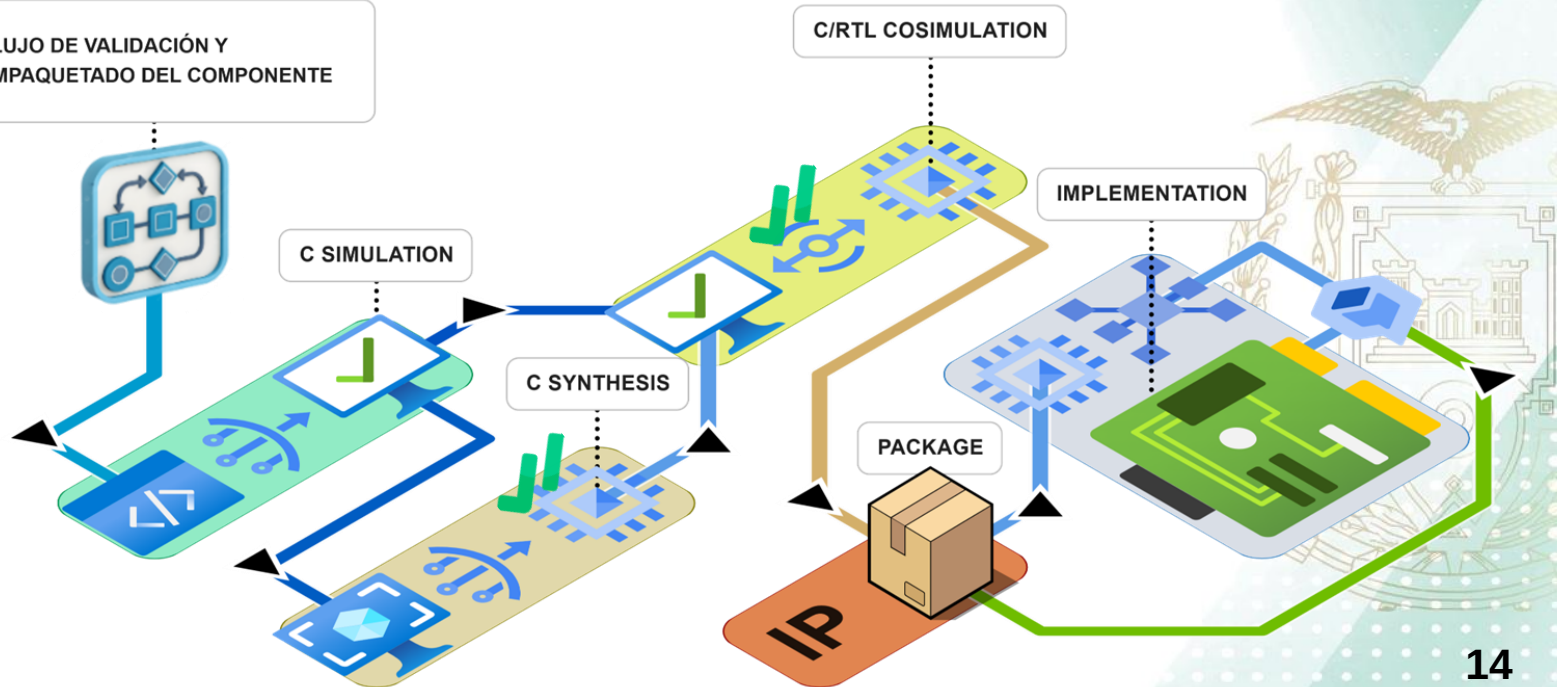
> REPORTS

✓ IMPLEMENTATION

▷ Run ✓

> REPORTS

FLUJO DE VALIDACIÓN Y
EMPAQUETADO DEL COMPONENTE



Metodología:

FASE II – C Simulation & C Synthesis

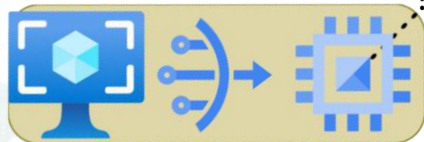


ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA



C SIMULATION → La simulación en Vitis HLS verifica que el algoritmo descrito en C o C++ funciona correctamente antes de convertirlo en hardware.

C SYNTHESIS



AES-128

TARGET	ESTIMATED	UNCERTAINTY
10.00 ns	7.882 ns	2.70 ns

SHA-256

TARGET	ESTIMATED	UNCERTAINTY
20.00 ns	14.600 ns	5.40 ns

▼ Performance & Resource Estimates

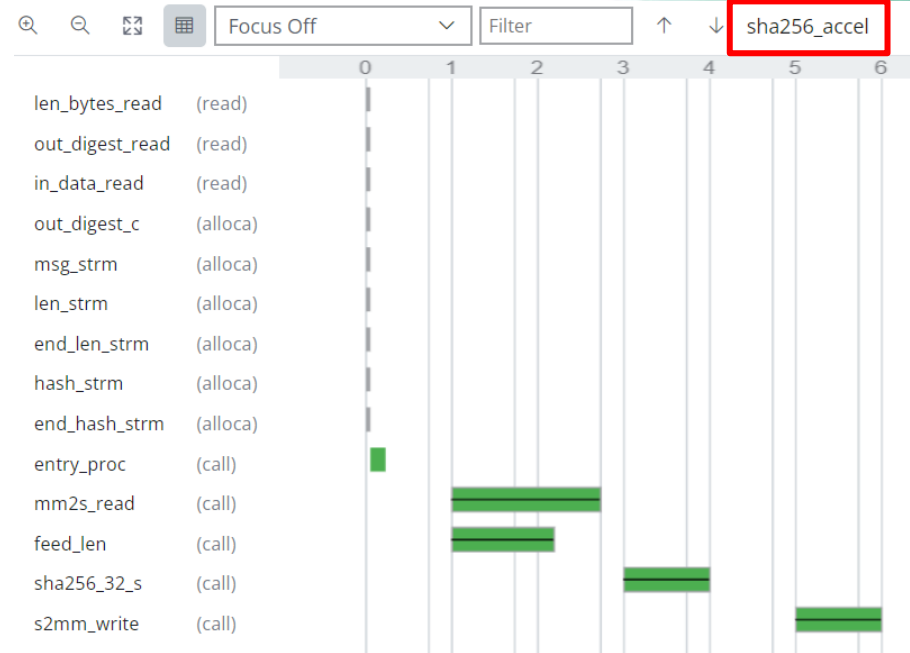
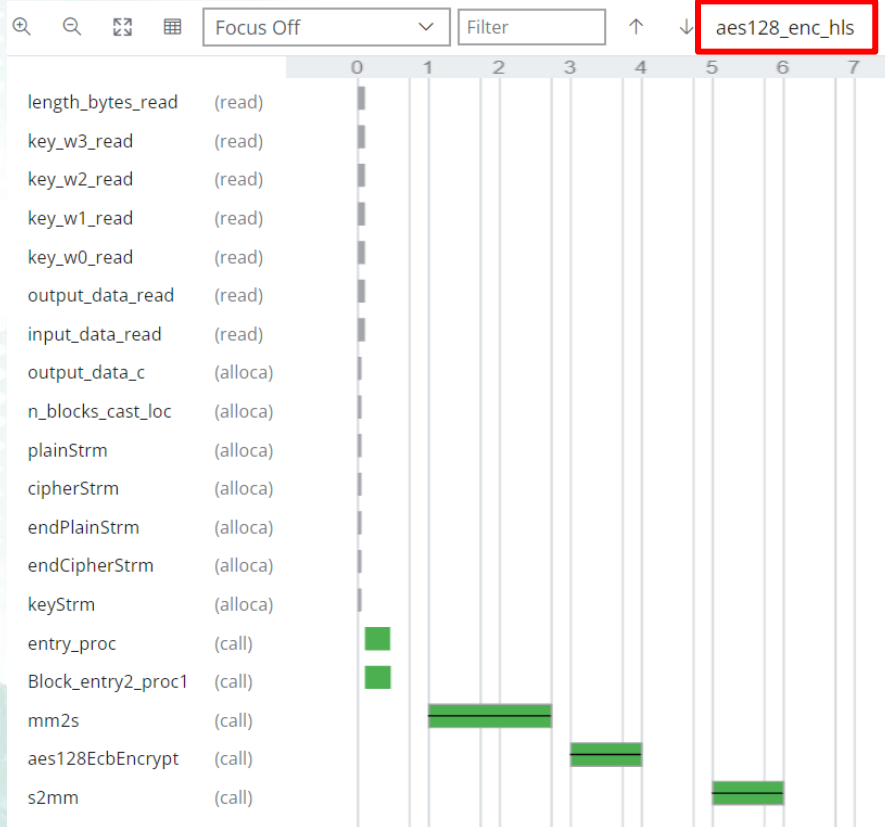
								☒ Modules ☒ Loops ☒ Hide empty columns					
MODULES & LOOPS	ISSUE TYPE	SLACK	LATENCY(CYCLES)	LATENCY(NS)	ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
▼ aes128_enc_hls (5)		-0.58	-	-	-	-	-	dataflow	52	0	7096	12523	0
entry_proc		-	0	0.0	-	0	-	no	0	0	3	29	0
MODULES & LOOPS	LATENCY(CYCLES)		LATENCY(NS)		ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
▼ sha256_accel (5)	-		-		-	-	-	dataflow	16	0	8761	14590	0
entry_proc	0		0.0		-	0	-	no	0	0	3	29	0

Metodología:

FASE II – C Simulation & C Synthesis



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA



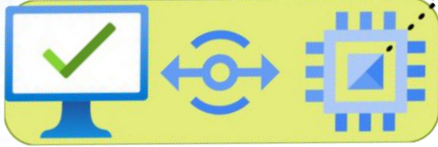
Metodología:

FASE II – C/RTL Co-Simulation

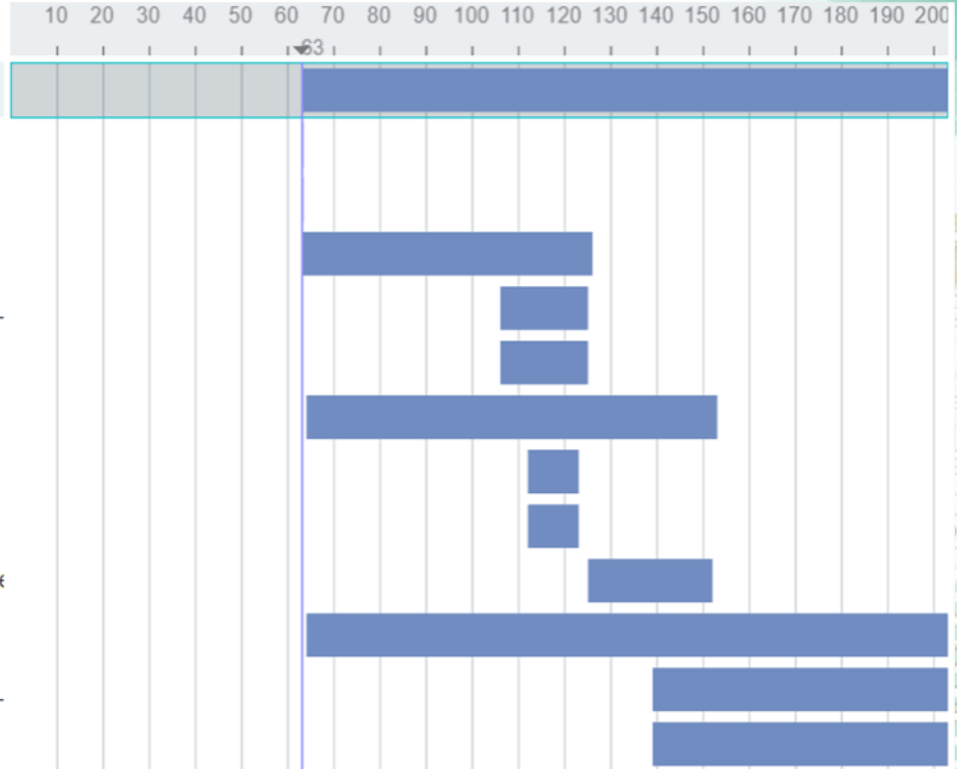


ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

C/RTL COSIMULATION



- aes128_enc_hls
 - entry_proc
 - Block_entry2_proc1
 - ✓ ● mm2s
 - ✓ ● mm2s_Pipeline_VITIS_LOOP_
 └─┐ VITIS_LOOP_18_1
 - ✓ ● aes128EcbEncrypt
 - ✓ ● updateKey
 - └─┐ VITIS_LOOP_418_1
 - > ● aes128EcbEncrypt_Pipeline_ε
 - ✓ ● s2mm
 - ✓ ● s2mm_Pipeline_VITIS_LOOP_
 └─┐ VITIS_LOOP_37_1

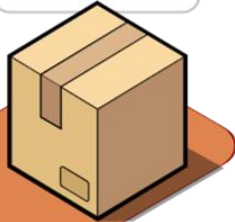


Metodología:

FASE II – Package

Un IP Core es un bloque hardware reutilizable que implementa una función específica dentro de la FPGA.

PACKAGE

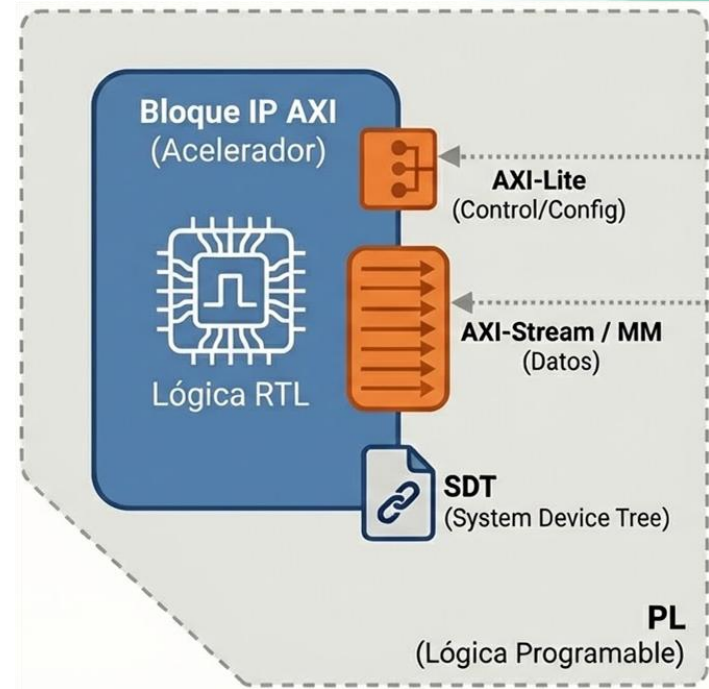


- Implementado en lógica digital (RTL)
- Integrable mediante interfaces estándar AXI
- Puede funcionar como acelerador hardware

No es software: es un circuito digital real dentro del chip



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

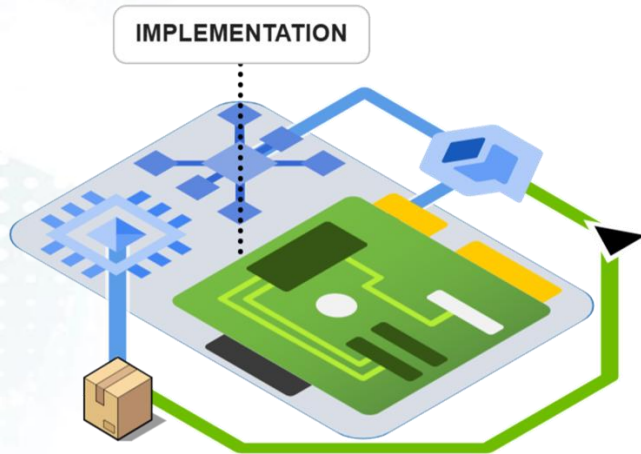


Metodología:

FASE II – Implementation



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA



AES-128 Encrypt

▼ Resource Usage

NAME	VERILOG
SLICE	2631
LUT	9493
FF	5379
DSP	0
BRAM	22

SHA-256

▼ Resource Usage

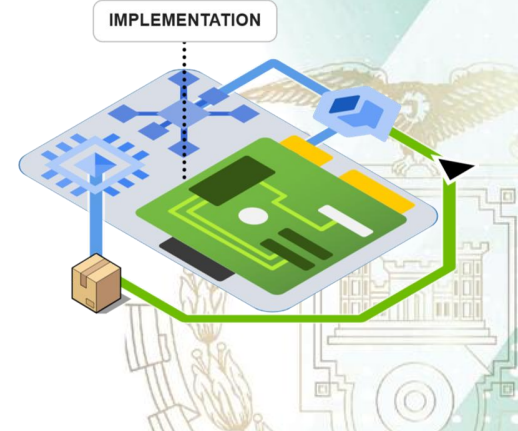
NAME	VERILOG
SLICE	2589
LUT	5562
FF	8528
DSP	0
BRAM	9

Costo del Rendimiento: Uso de Recursos en AES-128

Recurso / AES-128 Encrypt	Vitis Library IP	Custom IP
LUT (Logic)	9493	7018
FF (Flip-Flops)	5379	6275
BRAM (Memory)	22	74

AES-128 Encrypt

✓ Resource Usage	
NAME	VERILOG
SLICE	2631
LUT	9493
FF	5379
DSP	0
BRAM	22



i **Análisis:** El aumento significativo en BRAM en el diseño Custom fue una *decisión arquitectónica deliberada* para implementar buffers internos más grandes, lo que reduce drásticamente el overhead de comunicación con el bus AXI.

Manejo e Integración del Hardware



Vivado Design Suite

Integración del IP
generado con el Zynq
Processing System.

Generación del Bitstream
final (.bit) y archivo de
plataforma hardware

(.xsa)



PetaLinux

Construcción de la
plataforma de desarrollo.

Configuración del Device
Tree para reconocer los
aceleradores en el PL.



Drivers UIO

Implementación de
drivers udmabuf y UIO
para acceso directo a
memoria desde espacio de
usuario.

Metodología:

FASE III - Implementacion de la IP en Vivado

C:/Vitis_projects/MIC_HW_RE/MINE_AES128_ENC/VIVADO

Tools Reports Window Layout View

☒ Validate Design F6

Create and Package New IP...

Create Interface Definition...

Enable Dynamic Function eXchange...

Run Tcl Script...

Property Editor Ctrl+J

Associate ELF Files...

Generate Memory Configuration File...

Compile Simulation Libraries...

Vivado Store...

Custon Commands

Launch Vitis IDE

Language Templates

Settings...

Settings

Project Settings

General

Simulation

Elaboration

Synthesis

Implementation

Bitstream

IP

Repository

Package

Tool Settings

Project

IP Defaults

Vivado Store

Source File

Display

Help

Text Editor

3rd Party Simulators

Colors

Selection Rules

Shortcuts

Strategies

Window Behavior

IP > Repository

Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

IP Repositories

+ - ↑ ↓

C:/Vitis_projects/MIC_HW_RE/MINE_AES128_ENC/VIVADO_AES128_ENC/aes128_custom_enc (Pr

Refresh All

IP INTEGRATOR

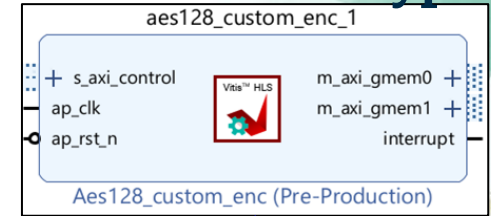
Create Block Design

Open Block Design

Generate Block Design

OK Cancel Apply Restore...

AES-128 Encrypt



BLOCK DESIGN - vivado_aes128_enc *

Diagram x Address Editor x

Search: Q: aes (2 matches)

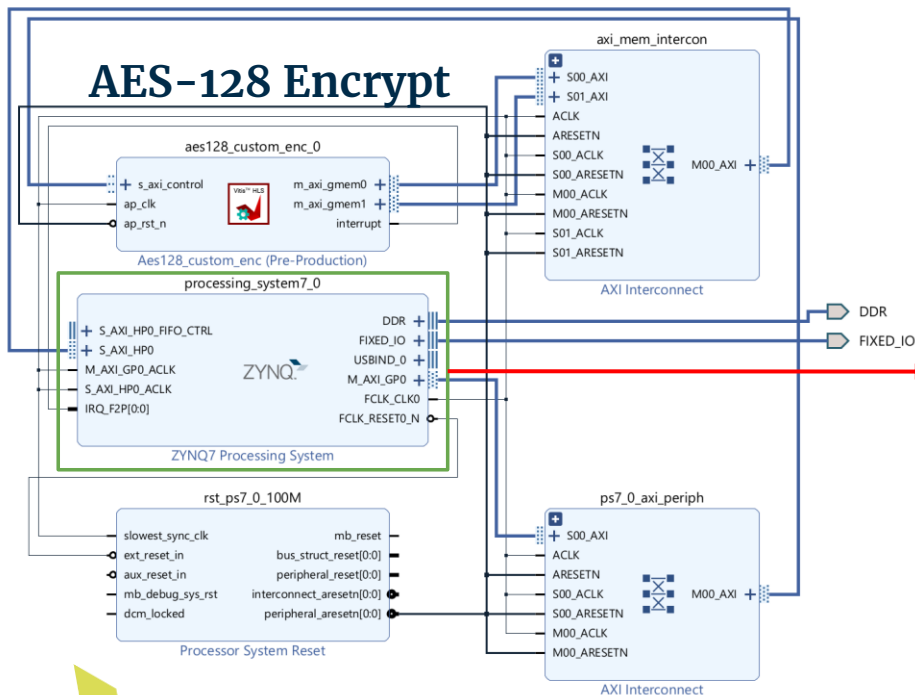
Aes128_custom_enc

SPDIF/AES3



Metodología:

FASE III - Configuración del Zynq PS



Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

Page Navigator

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

Name	Select	Description
HP Slave AXI Interface		
> S AXI HP0 interface	☒	Enables AXI high performance slave interface 0
> S AXI HP1 interface	☐	Enables AXI high performance slave interface 1
> S AXI HP2 interface	☐	Enables AXI high performance slave interface 2
> S AXI HP3 interface	☐	Enables AXI high performance slave interface 3

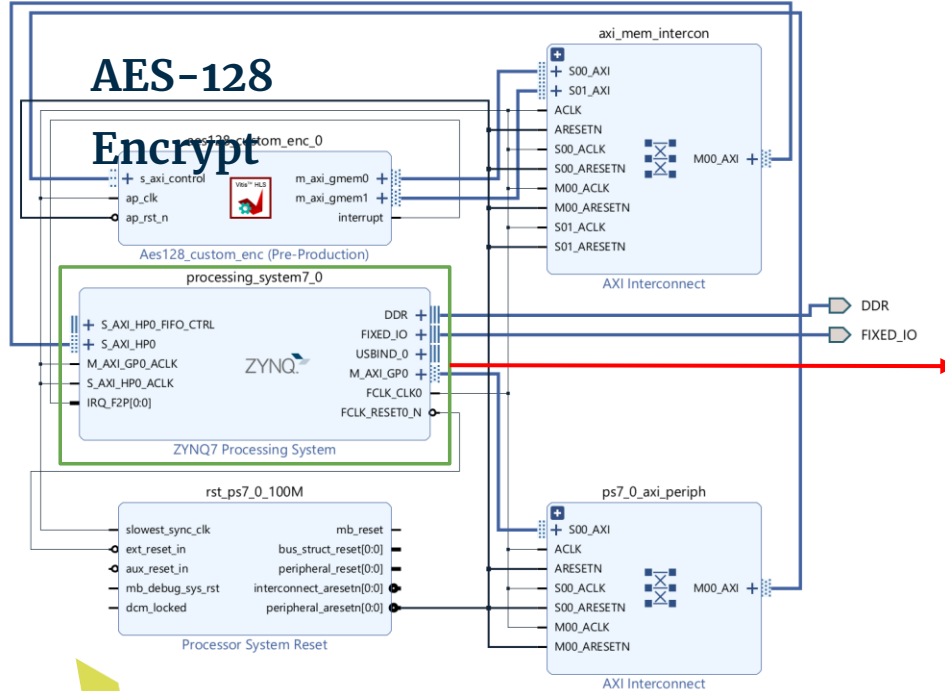
Component	Clock S...	Requested...	Actual Fre...	Range(MHz)
PL Fabric Clocks				
☒ FCLK_CLK0	IO PL	100	100.000000	0.100000 : 250.000000
☐ FCLK_CLK1	IO PLL	50	10.000000	0.100000 : 250.000000
☐ FCLK_CLK2	IO PLL	50	10.000000	0.100000 : 250.000000
☐ FCLK_CLK3	IO PLL	50	10.000000	0.100000 : 250.000000

Interrupt Port	ID	Description
Fabric Interrupts		
Enable PL Interrupts to PS and vice versa		
PL-PS Interrupt Ports		
☒ IRQ_F2P[15:0]	[91:84]	Enables 16-bit shared interrupt port from the PL. MSB is a
☐ Core0_nFIQ	28	Enables fast private interrupt signal for CPU0 from the PL
☐ Core0_nIRQ	31	Enables private interrupt signal for CPU0 from the PL

OK Cancel

Metodología:

FASE III - Comunicación AXI



AXI-Lite:

Canal de Bajo ancho de banda para el control, configuración de registros y estado de los aceleradores.

AXI HP (High Performance):

Canal de Alto rendimiento para la **transferencia masiva de datos (DMA)** directamente hacia/desde la memoria DDR.



Metodología:

FASE III - SYNTHESIS DEL IP SHA-256



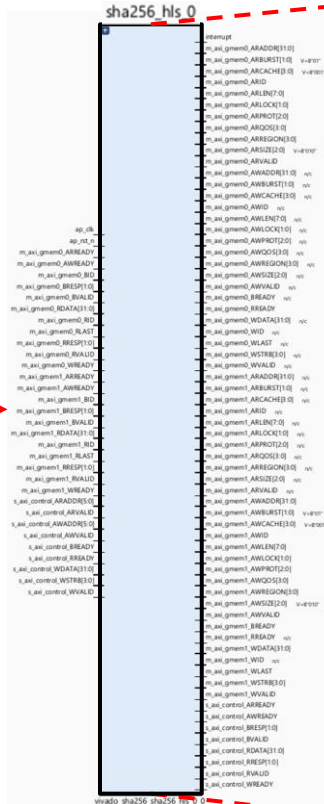
- ✓ SYNTHESIS
 - ▶ Run Synthesis
 - Open Synthesized Design

SHA-256

AES-128 Encrypt

DDR_cas_n	+	DDR_addr[14:0]
DDR_ck_n		DDR_ba[2:0]
DDR_ck_p		DDR_dm[3:0]
DDR_cke		DDR_dq[31:0]
DDR_cs_n		DDR_dqs_n[3:0]
DDR_odt		DDR_dqs_p[3:0]
DDR_ras_n		FIXED_IO_ddr_vrp
DDR_reset_n		FIXED_IO_mio[53:0]
DDR_we_n		FIXED_IO_ps_clk
FIXED_IO_ddr_vrn		FIXED_IO_ps_porb
		FIXED_IO_ps_srstb

vivado_aes128_enc



• vivado_sha256_sha256_hls_0_0(vivado_sha256_sha256_hls_0_0_arch) (vivado_sha256_sha256_hls_0_0_vhd) (1)

✓ U0: sha256_hls[behav] (sha256_hls.vhd) (33)

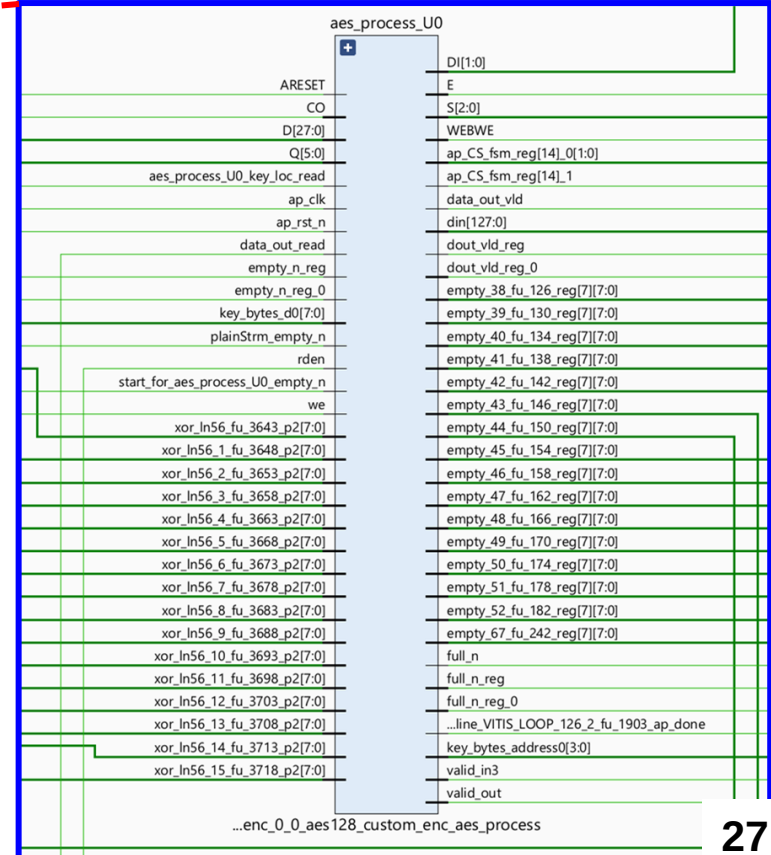
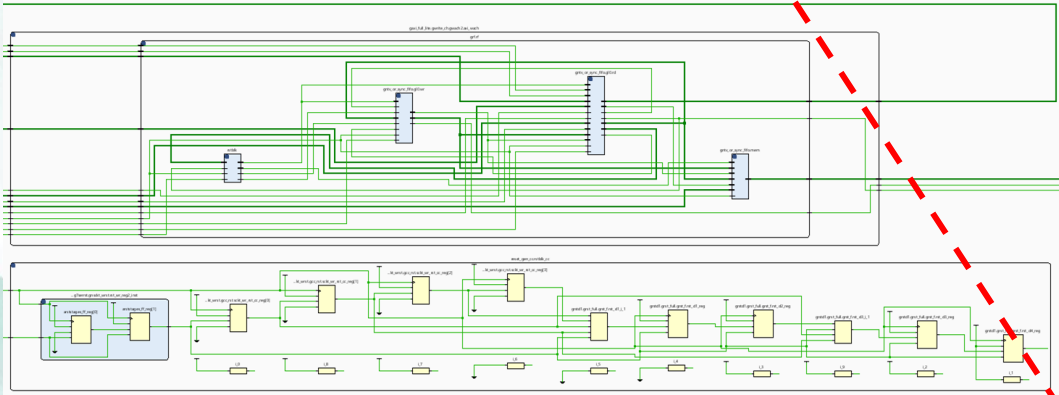
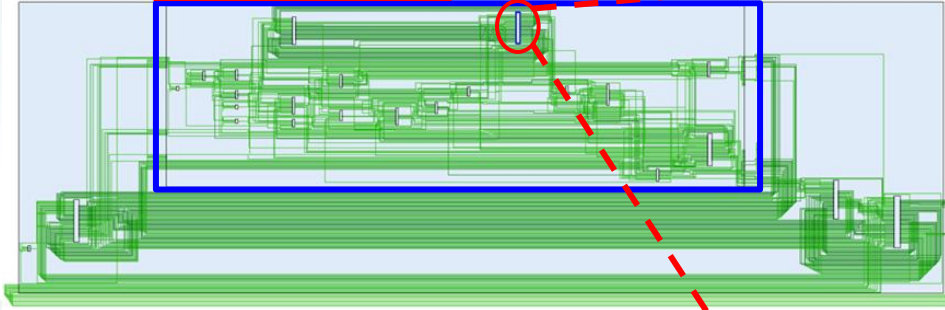
- K_U: sha256_hls_K_ROM_AUTO_1R(rtl) (sha256_hls_K_ROM_AUTO_1R.vhd)
- w_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_1_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_2_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_3_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_4_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_5_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_6_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_7_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_8_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_9_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_10_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- w_11_U: sha256_hls_w_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_w_RAM_1WNR_AUTO_1R1W.vhd)
- block_U: sha256_hls_block_RAM_1WNR_AUTO_1R1W(rtl) (sha256_hls_block_RAM_1WNR_AUTO_1R1W.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_84_2_fu_660: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_84_2(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_84_2.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_32_1_fu_670: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_32_1(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_32_1.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_41_2_fu_678: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_41_2(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_41_2.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_48_3_fu_685: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_48_3(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_48_3.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_92_3_fu_710: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_92_3(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_92_3.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_32_14_fu_719: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_32_14(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_32_14.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_41_25_fu_727: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_41_25(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_41_25.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_112_4_fu_734: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_112_4(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_112_4.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_48_36_fu_739: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_48_36(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_48_36.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_32_17_fu_764: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_32_17(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_32_17.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_41_28_fu_772: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_41_28(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_41_28.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_48_39_fu_779: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_48_39(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_48_39.vhd)
- > grp_sha256_hls_Pipeline_VITIS_LOOP_32_11_fu_804: sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_32_11(behav) (sha256_hls_sha256_hls_Pipeline_VITIS_LOOP_32_11.vhd)

Metodología:

FASE III - Vivado RTL del AES-128 Encrypt



aes128_custom_enc_0





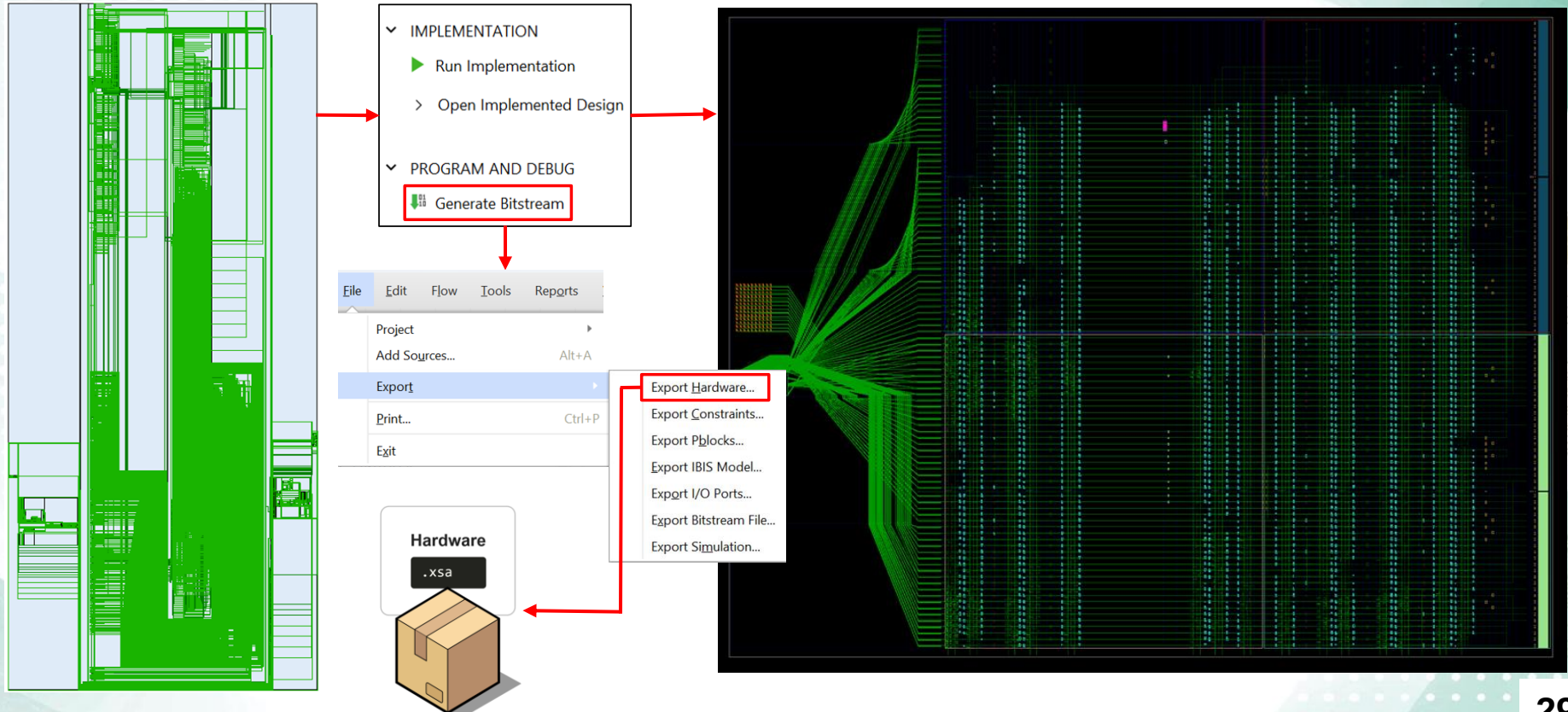
28

Metodología:

FASE III- IMPLEMENTATION y Bitstream

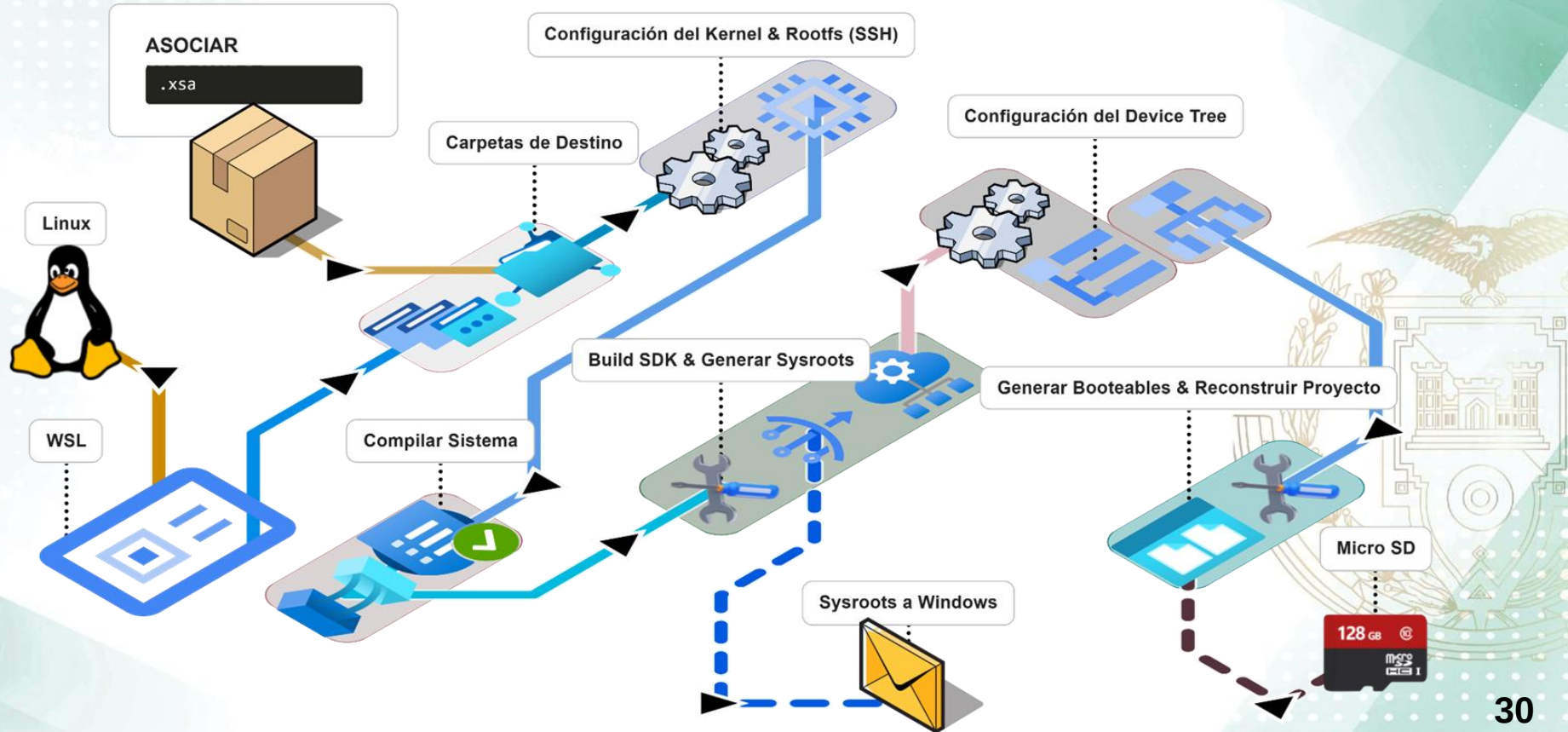


ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA



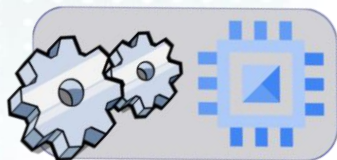
Metodología:

FASE IV – Linux Embebido – WSL



Metodología:

FASE IV - Configuración del kernel - Rootfs (SSH)

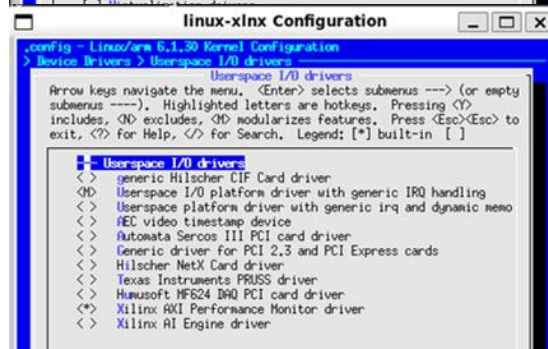
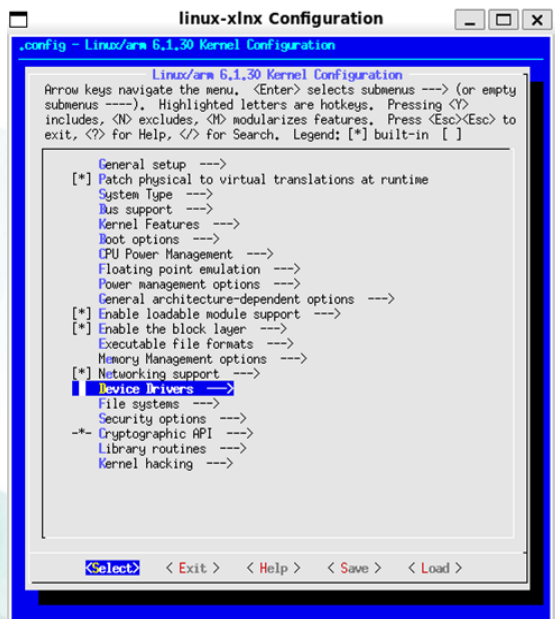


CONFIGURAR KERNEL Y ROOTFS (SSH, LOGIN ROOT, SD)

```
cd ~/petalinux_acelerador_projects/hw_aes128_enc/aes128_enc_linux  
petalinux-config -c kernel
```

Configurar rootfs y resolver conflicto openssl

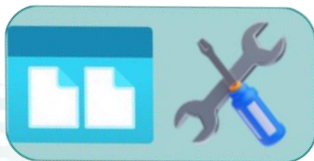
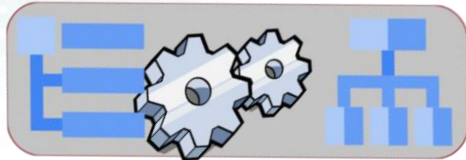
```
petalinux-config -c rootfs
```



```
[*] openssl  
[ ] openssl-misc  
[ ] openssl-dbg  
[*] openssl-sshd  
[*] openssl-keygen  
[ ] openssl-ssh  
[ ] openssl-dev  
[*] openssl-sftp  
[*] openssl-sftp-server  
[ ] openssl-scp
```


Metodología:

FASE IV – Configuración del Buffer, el SDT y Generar Booteables



Crear el módulo (Buffer)

```
cd ~/petalinux_acelerador_projects/hw_aes128_enc/aes128_enc_linux  
petalinux-create -t modules --name u-dma-buf --enable
```

Sobre escribir el Buffer

Vamos a usar el comando `cat` para sobre escribir el archivo `u-dma-buf.bb` con la configuración correcta que descarga el driver desde GitHub.

```
cat <<EOF > project-spec/meta-user/recipes-modules/u-dma-buf/u-dma-buf.bb  
SUMMARY = "Recipe for building an external u-dma-buf Linux kernel module"  
SECTION = "PETALINUX/modules"  
LICENSE = "BSD-2-Clause"  
LIC_FILES_CHKSUM = "file://LICENSE;md5=bebf0492502927bef0741aa04d1f35f5"  
  
inherit module  
  
SRC_URI = "git://github.com/ikwzm/udmabuf.git;protocol=https;branch=master"  
SRCREV = "${AUTOREV}"  
  
S = "${WORKDIR}/git"  
  
RPROVIDES:${PN} += "u-dma-buf"  
EOF
```

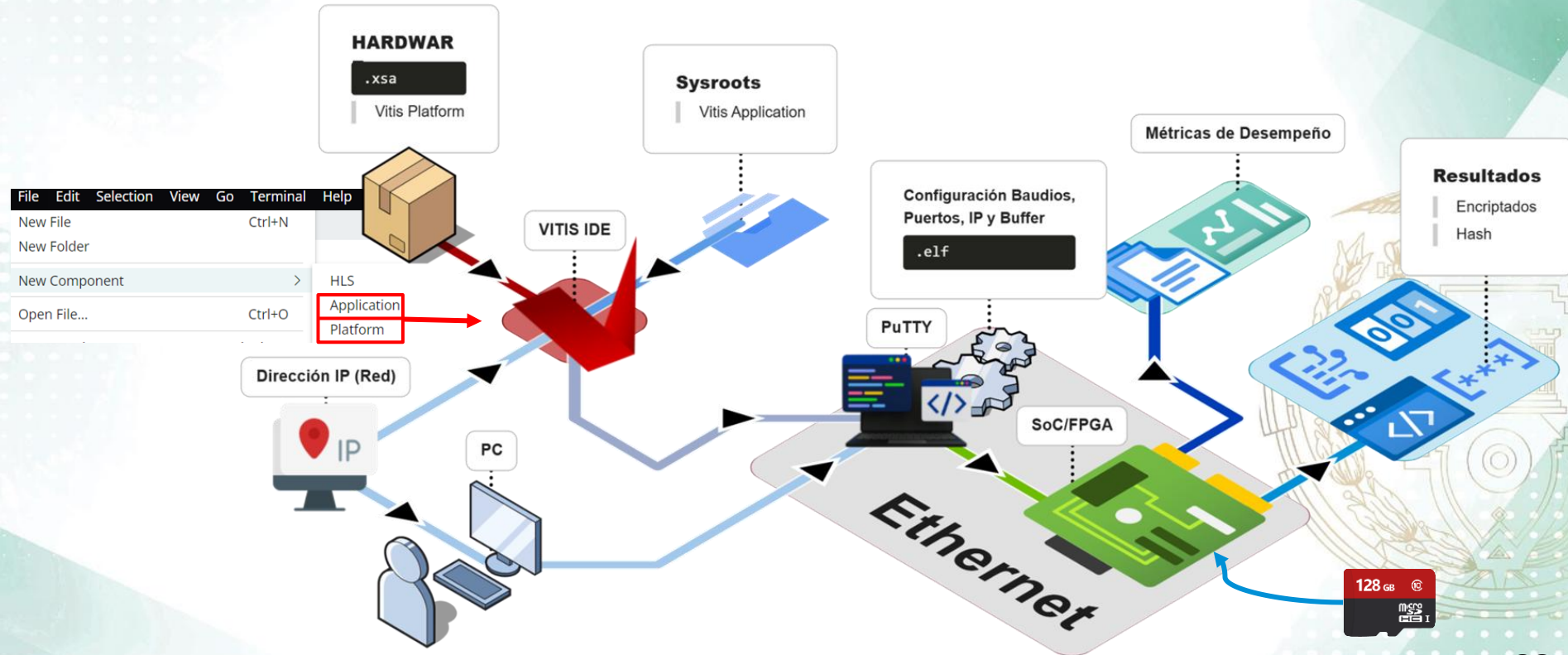
Reconstruimos el divece tree

```
petalinux-build -c device-tree -x clean  
petalinux-build -c device-tree  
petalinux-build  
petalinux-package --boot --force \  
--fsbl images/linux/zynq_fsbl.elf \  
--fpga images/linux/system.bit \  
--u-boot
```



Metodología:

FASE IV – Implementación en Vitis IDE & PuTTY

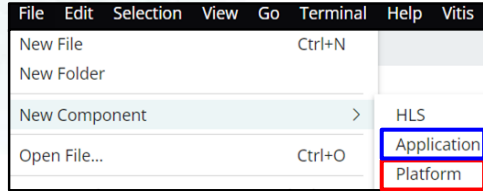


Metodología:

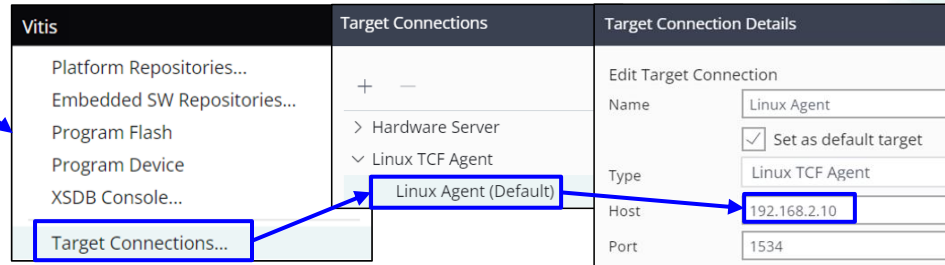
FASE IV - Vitis IDE

Red e Internet > Ethernet

Asignación de IP: Manual
Dirección IPv4: 192.168.2.2
Máscara IPv4: 255.255.255.0



VITIS IDE



Other debug flags

Other flags related to debugging

```
--sysroot=C:/hw_my_sdk_aes128_enc/sysroots/cortexa9t2hf-neon-xilin
```

Linker flags

Add any linker options that are not covered by the above variables, they will be added as extra linker options.

```
-mfloat-abi=hard -mfpv=neon
```

// Offsets de registros AXI-Lite del IP (según Vitis HLS).

```
#define REG_CTRL      0x00  
#define REG_GIE       0x04  
#define REG_IER       0x08  
#define REG_ISR       0x0C
```

```
#define REG_PTR_INPUT  0x10  
#define REG_PTR_OUTPUT 0x18  
#define REG_KEY_W0     0x20  
#define REG_KEY_W1     0x28  
#define REG_KEY_W2     0x30  
#define REG_KEY_W3     0x38  
#define REG_LEN        0x40
```

Metodología:

FASE IV - Configuración en el PuTTY

PuTTY Configuration

Category:

- Session
- Logging
- Terminal
- Keyboard
- Bell
- Features
- Window
- Appearance
- Behaviour
- Translation
- Selection
- Colours
- Connection
- Data
- Proxy
- Telnet
- Rlogin
- SSH
- Serial

About

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM9 Speed: 115200

Connection type:
☐ Raw ☐ Ielnet ☐ Rlogin ☐ SSH ☒ Serial

Options controlling the effects of keys

Change the sequences sent by:

The Backspace key
☐ Control-H ☒ Control-? (127)

The Enter key
☐ CR ☒ CR LF

Options controlling local serial lines

Select a serial line

Serial line to connect to: COM9

Configure the serial line

Speed (baud): 115200

Data bits: 8

Stop bits: 1

Parity: None

Flow control: None

Configuración en el PuTTY

```
sudo ifconfig eth0 192.168.2.10 netmask 255.255.255.0 up
```

UDMABUF + UIO

```
sudo su
cd /lib/modules/$(uname -r)/

# Cargar udmabuf de 8MB (cambia si quieres 1MB: 1048576)
rmmod u_dma_buf 2>/dev/null
insmod ./extra/u-dma-buf.ko udmabuf0=8388608
chmod 666 /dev/udmabuf0 # permisos
```

```
# Cargar UIO y bindear la IP
modprobe uio_pdrv_genirq 2>/dev/null || insmod ./kernel/drivers/uio/uio_pdrv_genirq.ko

# Confirma el device (cambia 40000000.aes128_enc_hls si tu base cambia)
ls -l /sys/bus/platform/devices | grep -i 40000000
ls -l /sys/bus/platform/devices | grep -i sha

# Fuerza override y bind
echo "" > /sys/bus/platform/devices/40000000.aes128_enc_hls/driver_override
echo uio_pdrv_genirq > /sys/bus/platform/devices/40000000.aes128_enc_hls/driver_override
echo 40000000.aes128_enc_hls > /sys/bus/platform/drivers/uio_pdrv_genirq/bind

chmod 666 /dev/uio0 # permisos
```


Ejecución y Envío de Resultados

```
18 int main() {
19     int fd = open("/dev/mem", O_RDWR | O_SYNC);
20     if (fd < 0) return 1;
21
22     // Mapeo de la IP
23     volatile uint32_t *ip = mmap(NULL, IP_RANGE, PROT_READ|PROT_WRITE, MAP_SHARED, fd,
24
25     // Obtener datos de udmabuf
26     FILE *f_addr = fopen("/sys/class/u-dma-buf/udmabuf0/phys_addr", "r");
27     uint64_t phys_addr;
28     fscanf(f_addr, "%llx", &phys_addr);
29     fclose(f_addr);
30
31     int fd_udma = open("/dev/udmabuf0", O_RDWR | O_SYNC);
32     uint8_t *buffer = mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, fd_udma, 0);
33
34     // Preparar mensaje
```

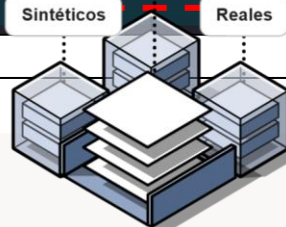
OUTPUT x DEBUG CONSOLE x PROBLEMS 1 x DATASETS

Uso: /home/petalinux/app_sha256.elf

Cargar los Datasets desde el WSL y Ejecutar

```
ssh-keygen -R 192.168.2.10
```

```
scp /mnt/c/Vitis_projects/Datasets/Pruebas/dataset_test_4MB.bin petalinux@192.168.2.10:/home/
petalinux/datasets/
```



Correr programa

```
../app_sha256.elf dataset_test_4KB.bin
```

Traer los Resultados de la FPGA a tu WSL

```
scp petalinux@192.168.2.10:/home/petalinux/datasets
/custom_metricas_sha256_fpga.csv
~/metricas_rendimiento/
```

Pasarlo de WSL a Windows

```
cd ~/metricas_rendimiento
cp custom_metricas_sha256_fpga.csv /mnt/c
/Vitis_projects/Metricas_Rendimiento/FPGA_Mine/
cd
```

Métricas Clave de Rendimiento



Tiempo End-to-End (E2E)

Visión real del usuario desde Linux. Incluye el tiempo de procesamiento + el overhead de movimiento de datos

(DMA).



Throughput Efectivo

Velocidad de procesamiento real en

$$T = \frac{\text{Data Size}}{\text{Time}}$$



Eficiencia de Recursos

Relación costo-beneficio. Consumo de LUTs, BRAM y Flip-Flops vs. la aceleración obtenida.

Reglas de Comparación:

- ✓ **Misma Plataforma:** Misma placa Zynq-7000.
- ✓ **Mismos Datasets**
- ✓ **Escenarios Reales:** Incluyendo latencia de transferencia de datos.

Resultados AES 128



COM9 - PuTTY

```
my_aes128_enc_linux:/home/petalinux/datasets# ls -l
-rwxr-xr-x 1 petalinu petalinu 46192724 Mar 9 12:40 Vitis_Unified_IDE_Reference_Manual_ENC.pdf
-rw-r--r-- 1 root root 46192736 Mar 9 12:42 custom_Vitis_Unified_IDE_Reference_Manual_ENC.pdf
-rw-r--r-- 1 root root 277 Mar 9 12:42 custom_metricas_aes128_enc_fpga.bin
my_aes128_enc_linux:/home/petalinux/datasets#
my_aes128_enc_linux:/home/petalinux/datasets# cat custom_Vitis_Unified_IDE_Reference_Manual_ENC.pdf
r!b' o q yq C<
2N K j, EJ sM & B2r C| bt b f-2 W O= I 2 y $ _ ) H mi vd o
7 m j ? F A t & T szj ( ' x q > OA 81
z';c.8hC#K '#4x 3 Jj.; z eyg j ū1{ 3' . 8RqUp t<M_a[]JW"Y@c|ky'd 2'b t>o_o Ny <n.
I * 5( (&f) w 7yL
c w1 ~#bX-4<N: x ~r e>A I
z3- /5" T]q[D*
[= 數 ^' b OS] Ha j j 4 P Q ; qwQ /=]6| cL x Z z 61 0r' u p o i g s #
4 W[ [ MV Qd b <8 Y @ ' J ' r n 5
u J; IYc / / M X5S
t~J*+-RZ k06
f I Q; 3o ] ZN:= I , b hRR 2 7 G P% h GQ| Yq| ! h p YUG4
, =, YY[ kGc < rnt+ R;4+ v 8> s05 v g5Up|a G 9 GN2D i
*x8 Z>[(Y4gw(1, l o - 60 .) R;g ~@ gI Jw v b1L"R H V )
E 7 X [ , ^ t< / pm 7 @ Q ^ 8 Hu 4 |a Wv & b 1AtDQ Xy
5 X x \ c - 8p + { = $ / k ? nll G =9 1^ > O o #< Q \ #YS
0p @ ' b W f K Lix7R : B wj k< hm s y t c W / h , r (11ws rr) , @ :
^ 8? X R s I 4 ' z H ; & rgr qt 8 N) b Q s
> \ / O tw m c + t c 租 U [ o + ~ ^ i 5_w ? & $ 2M O ' r u = b
0F ? ( R 2 u = b ) ( x w @ f1p/r 8
h4 S ,
```

custom_dataset_test_256MB_ENC.bin (0 ~ 5MB)

```
1 y09K>B<<iX6àY EOT à \[HSOHc; ūi I "āhm EOT UDÀ×20 EM, 5-I j ^ H <-v~Gc1 FS "XŬi i & DLE DEL o NUL | ACK% vfa$ ENQ SUB
( ūūv . ^ ^ c X ū UFS 2 ū . 0c-1 fñ51 tēk "A . " f F A ^ c t b E T X : Ā { 0ē SI pēā% } ACK " D F S ņ # x 0 { t 6 E M ū / @ x > R A H O P ± I 2 p ^ C
, æ " 0 D E L C ā ( - 0 D C 4
2 ^āBāG+ DEL + DCS • 0Ā • pBS SUB eē | " m d I ē P SOH _um & v> rē = ē j ā B ū C ^ 54B (ō g < 0z Z . ūē G , y , q$
3 ^aē ESC É ē : $ a ~ i G n K " G t ^ ; VT
4 e " " " i g + DCS ē ESC - 0° : " < 44 j 5 t - i 0 DCS + + φ (4ā4 SYN r s Bē SOH w , ō t E T B 7 > < ņ C ō ā ā D C 3 ^ , , ACK p I A i ~ CAN Q$ DEL
( ō r ) ^ , M x 0 S " E M ō i J R ē , g m Q P 3 j 0 ^ R S y ^ 4 ^ ESC S O 9 H c < 0 b N ā q , + ū Y φ j Ū . S T X ō ō ^ c ē p b o " w ō s
5 M P Y
6 ē 2 X CAN 4 DEL " E _ " Š Ā i " 0 0 L e 6 b i " _ ē ō % Z ^ ē z ; SYN , " g ū f 3 x " 0 c < k c C " DCS ~ Ž k A ā " 0 d Ā
7 È Xē DC 4 ē 0 DC 3 ō Ā & < 0 p i ē GS + S S 3 : ° 0 t & ē ē 0 F F S O 0 " " . k . " ē EOT - , , DEL H t ; l ā A ē ā p X " a ; r ō s CAN ° - % ō I ā
Y # w Ō R I VT D C 3 $ ESC X L D C 1 ū ā EOT O S C " F Y , s ō I O E M " ; : P D C 1 7 ^ z : M ^ ō i i S O H ā Y S O H s ō I Ō 7 ē 4 ō I i j Y h > ^ S - h ē j ū ā ā -
# 4 ^ i m ^ DEL ē ENQ D C 4 ; BEL : Ū x c z y Ū ā H i c 6 EOT D C 1 i s ( R E M a N A K S D L E J N U L ō ^ N 7 s d 1 1 R S ^ D C 4 S ō _ ā R I Ū , R S B r " 3 c D C 4 ō
$ w B " F Z H X , Ā C F S w s ā K ^ ē s S " m ^ ū E N Q . ° I F F f z ; ā x ē ā h ^ ! S D C 2 R I i o / I j ē ž Ā " GS @ - R S ā p S I & C i ^ U S \ ž ē y ° { $ i D C 2 F S ē ^ -
ž A C K y ō x ā Y B E L O c X H , X & ž { { S O H ū D C 2 D C 3 ō ^ E N Q s c : J R S " ž ū ē E N Q ū o E T B ē i o o d F F ņ y ē ū m s j u A A ē ^ & s m B S ņ d ō c ā D 2 E T B
ē y V T φ G ; 0 r ^ ; , o ° = g r a " i " c ; ā = 3 , z Ū d _ 7 ; D C 1 E S C - j " φ ē D C 3 O m ° ū ū ō L D L E ō 0 8 R " D C 2 i ō E T B ō D C 1 Ž P f e , m ) = ^ i " w V V M ^ T ~ †
, Ō A K " n Ō k c ^ 2 r F S z ā B C O S C ā ^ 6 S g - 0 Y Ā . B = ž - ō ō % X ? D C 3 < g - V S ē S I @ ū ō ū y L ō , ō i " i i " V Ā " ; < X " E J W a S Q EOT DEL D C 1
X ō i ō EOT 3 8 F p i g + G | P D L E v Ō E T X 8 x T c + DEL o φ | : 8 p ^ 2 d ā i ē c z / Ō M N U L + ū F i ā ē R - q R I D C 2 + B S X : [ DCS ~ X i j B S
Q ā j X 7 Ū ^ N A K c h x + " Ū 8 ° / H S O H " - = / O - L D ō m + U D C 2 8 2 P p ē S Y Ā i 4 ō ū v ) ō ņ DEL ē E S L 2 H O P E T B " X " 0 ā v ~ ž S I ē S ē Z \ ^ T °
Ū DEL NUL - SYN S ō S .
```


Resultados SHA 256



```
=====
EJECUTANDO SHA-256 EN FPGA (Zynq PL)
=====
Archivo      : dataset_test 4MB.bin
Tamano procesado : 4.00 MB (4194304 bytes)
=====
```

```
=====
Hash (SHA-256)      d9b5532296bd468e570ad278519ded700f7bc9d34496e89fb9221726590124b5
=====
```

METRICAS DE RENDIMIENTO (SHA-256 - FPGA)

```
=====
[1] Tiempo total End-to-End      : 0.315899457 s
[2] Throughput End-to-End        : 12.66 MB/s
=====
[3] Tiempo de CPU del proceso (ARM) : 0.315777000 s
[4] Uso de CPU del proceso (ARM)    : 99.96 %
=====
[5] Tiempo de ejecucion del acelerador (PL) : 0.266740251 s
[6] Throughput del acelerador (PL)   : 15.00 MB/s
=====
[7] Tiempo de transferencia de datos : 0.048399792 s
[8] Overhead de control y sincronizacion : 0.000015768 s
=====
```

```
>> Metricas_agregadas a: metricas_sha256_fpga.csv
```

```
18 int main() {
19     int fd = open("/dev/mem", O_RDWR | O_SYNC);
20     if (fd < 0) return 1;
21
22     // Mapeo de la IP
23     volatile uint32_t *ip = mmap(NULL, IP_RANGE, PROT_READ|PROT_WRITE, MAP_SHARED, fd,
24
25     // Obtener datos de udmabuf
26     FILE *f_addr = fopen("/sys/class/u-dma-buf/udmabuf0/phys_addr", "r");
27     uint64_t phys_addr;
28     fscanf(f_addr, "%llx", &phys_addr);
29     fclose(f_addr);
30
31     int fd_udma = open("/dev/udmabuf0", O_RDWR | O_SYNC);
32     uint8_t *buffer = mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, fd_udma, 0);
33
34     // Preparar mensaje
```

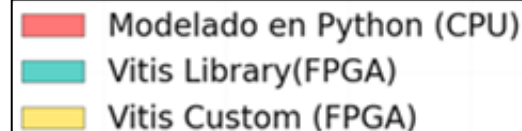
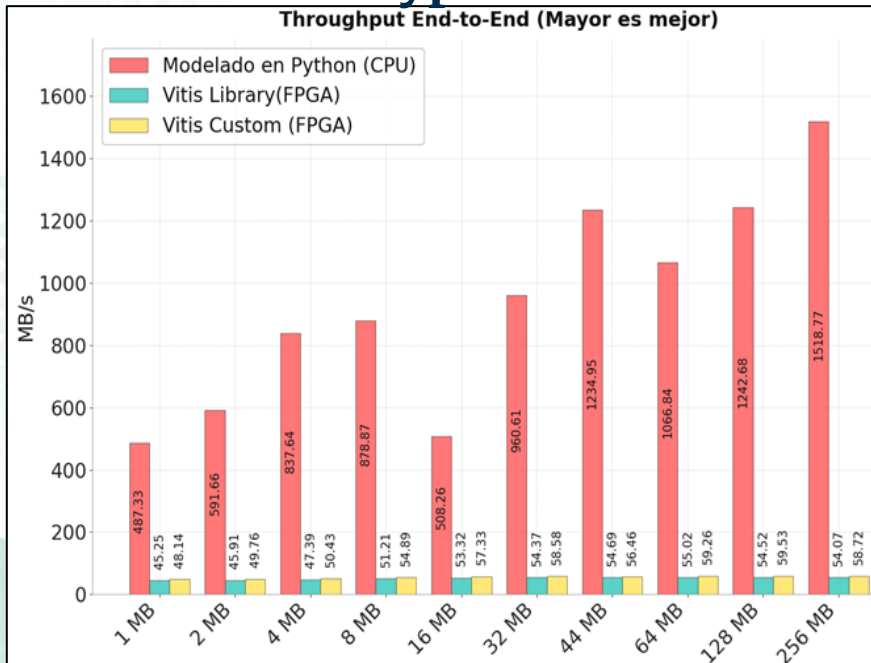
OUTPUT x DEBUG CONSOLE x PROBLEMS 1 x

Uso: /home/petalinux/app_sha256.elf

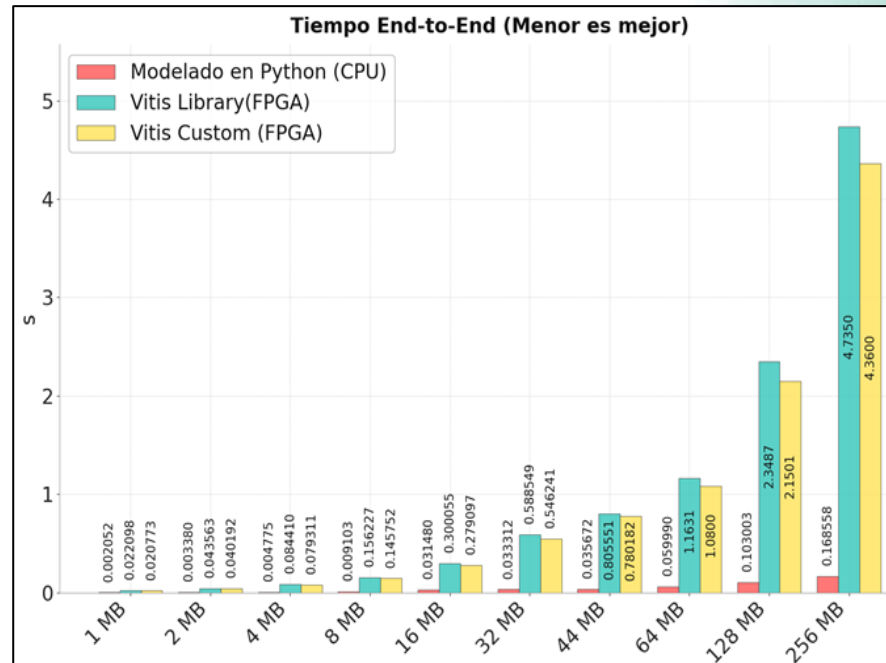
Resultados AES-128: Comparativa de Rendimiento

AES-128 Encrypt

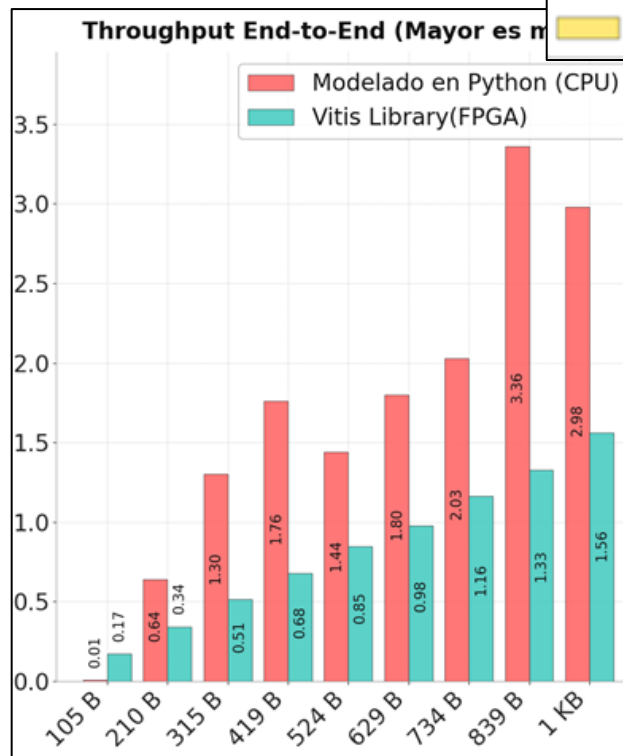
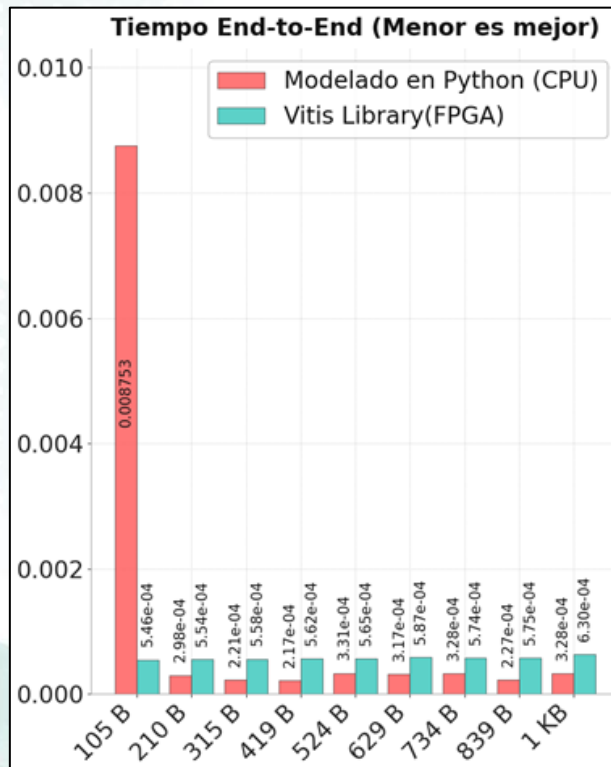
Throughput End-to-End (Mayor es mejor)



Tiempo End-to-End (Menor es mejor)



Resultados SHA-256: Comparativa de Rendimiento



Modelado en Python (CPU)
Vitis Library(FPGA)
Vitis Custom (FPGA)

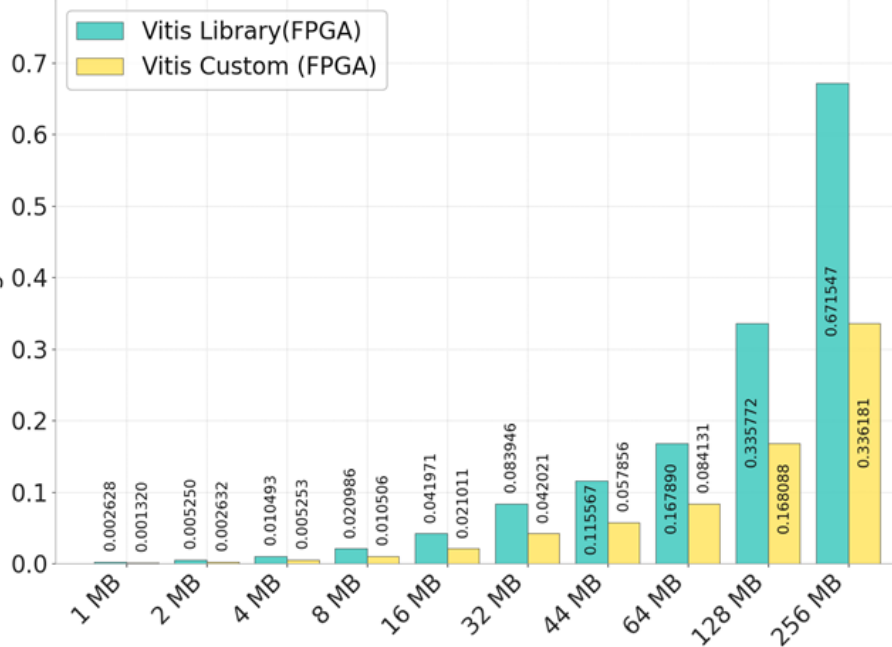


El Cuello de Botella Real: Overhead

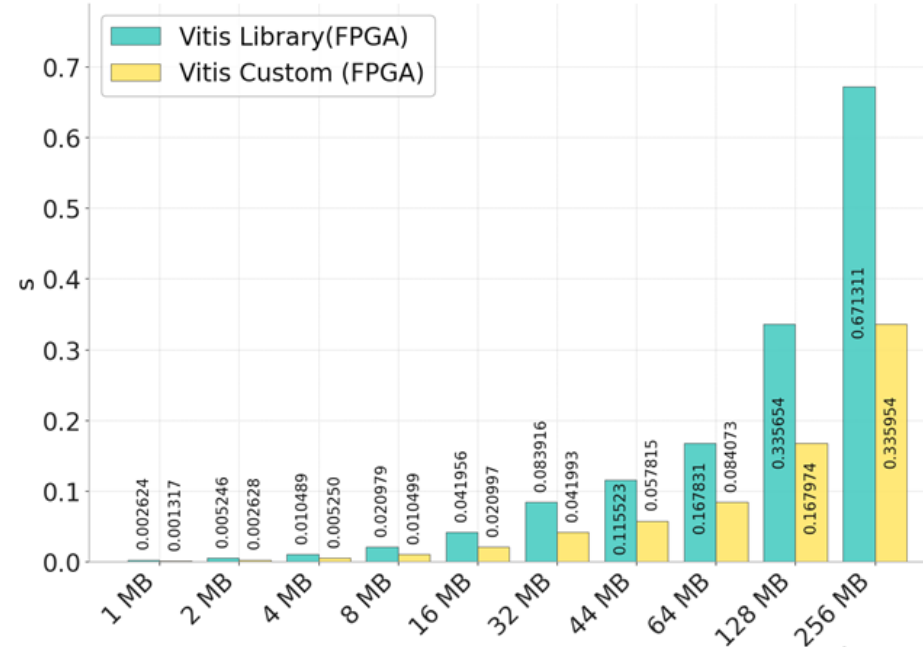
AES-128 Encrypt

Vitis Library(FPGA)
Vitis Custom (FPGA)

Overhead de control/sincronización (Menor es mejor)

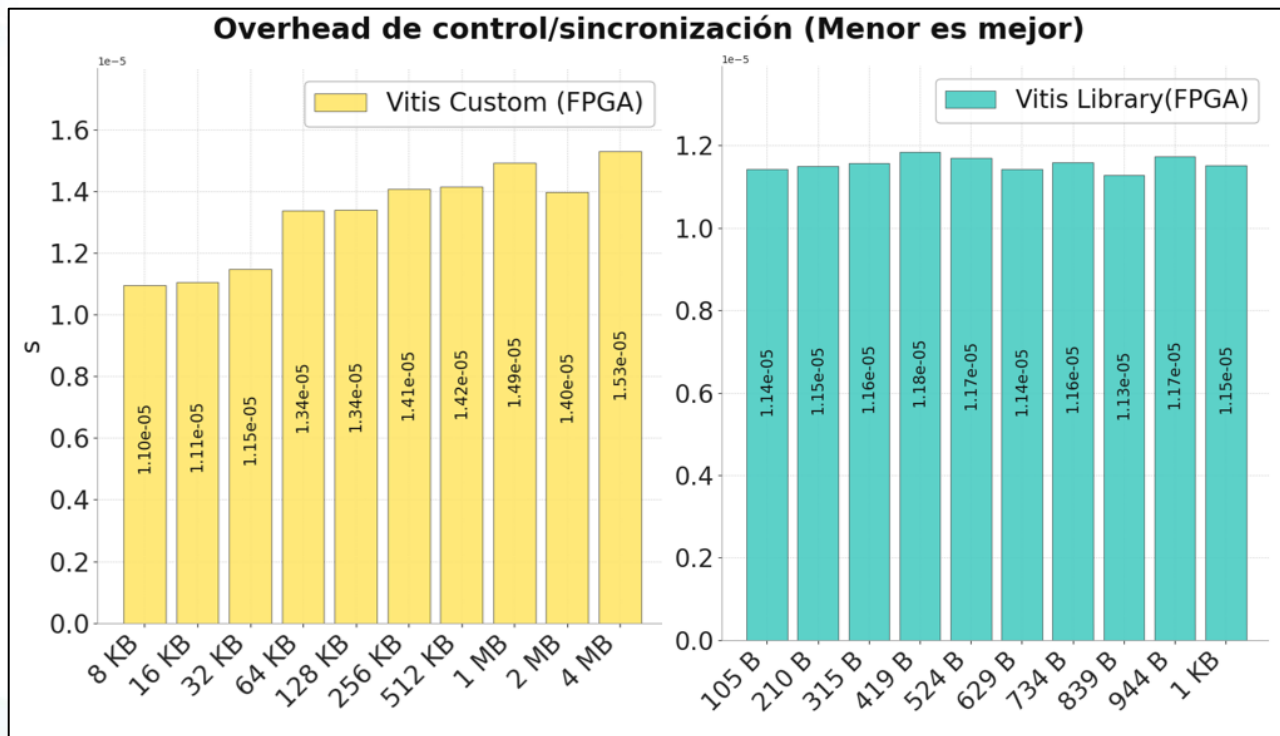


Tiempo de ejecución del acelerador (PL) (Menor es mejor)



Overhead de control y tiempo de transferencia

Vitis Library(FPGA)
Vitis Custom (FPGA)



Conclusiones



HLS > VHDL/RTL

High-Level Synthesis permite una aceleración real reduciendo drásticamente el tiempo de desarrollo frente a RTL puro, sin sacrificar rendimiento crítico.



Estabilidad Zynq-7000

La plataforma Zynq-7000 demuestra ser robusta para offloading criptográfico, validando su uso en seguridad embebida industrial.



Personalizado > Librería

En entornos de recursos limitados (IoT/Edge), las soluciones a medida superan a las IPs "caja negra" en eficiencia de área y latencia.

Recomendaciones y Trabajo Futuro

Fase 1: Optimización

Implementación de DMA Scatter-Gather para permitir streaming continuo de datos y eliminar tiempos muertos de transferencia.

Fase 2: Seguridad Real

Evolución a modos de operación AES-GCM o CTR.
Implementación de contramedidas contra ataques de canal lateral.

Fase 3: Migración

Salto a arquitectura UltraScale+ para romper los límites de frecuencia de reloj y ancho de banda del bus AXI actuales.

***¡MUCHAS
GRACIAS!***

