AES-128 Encryption Using HPS–FPGA Co-Design

Presented by: Husam Aldulaimi

Email:haa190002@utdallas.edu, enghusam1977@gmail.com

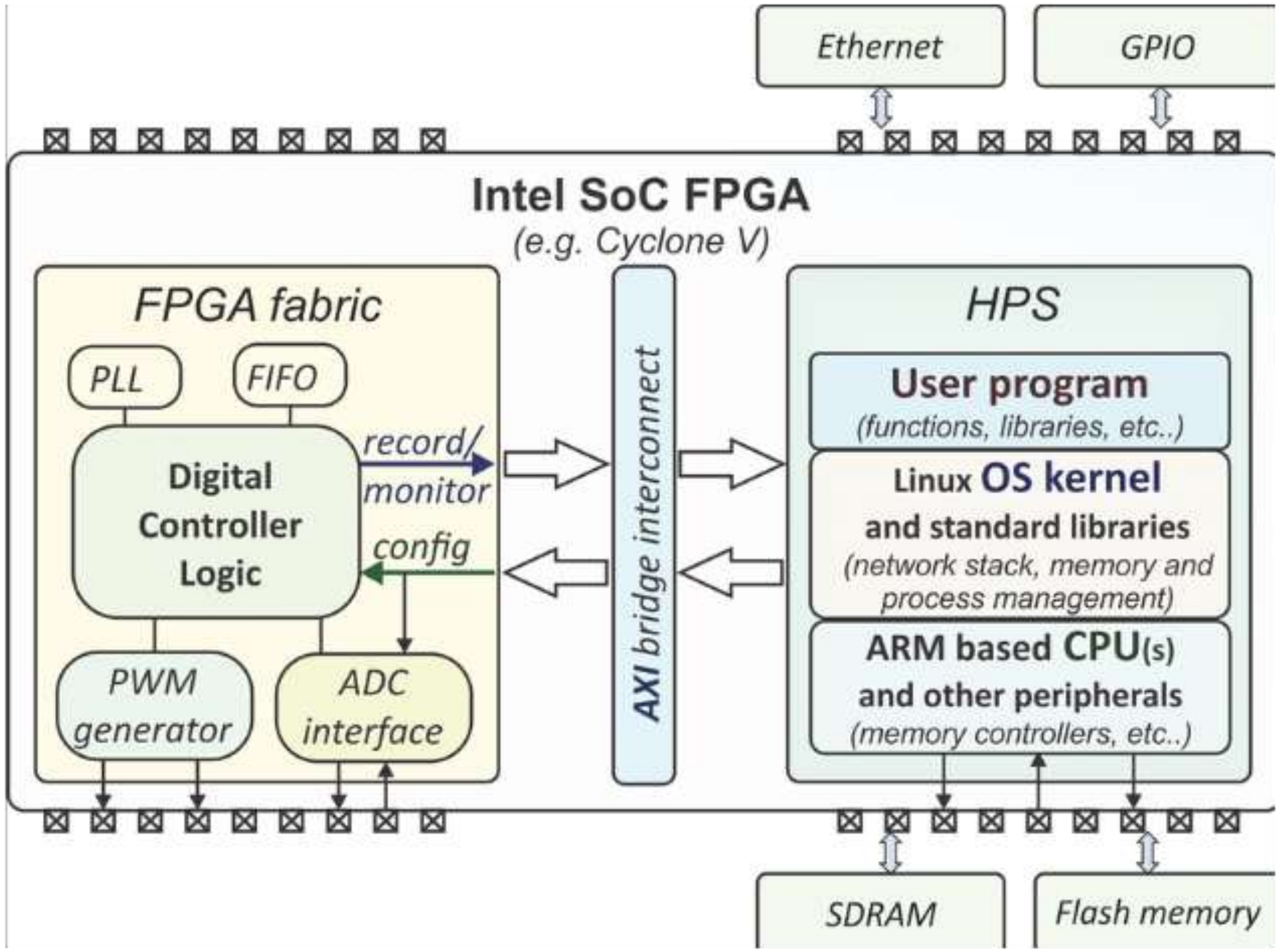
Course: EEDG 6370 – Design & Analysis of Reconfigurable Systems
Toolchain: Quartus Prime, Platform Designer, Altera SoC EDS(Embedded Design Suite), Linux (HPS)

Project Motivation

- AES-128 is a widely used symmetric encryption standard
- Hardware acceleration provides:
  - Higher performance
  - Deterministic latency
  - Lower CPU load
- Cyclone V SoC enables **tight integration** between:
  - ARM Cortex-A9 (HPS)
  - FPGA fabric
- **Goal:** Implement AES-128 encryption in FPGA and control it from HPS using memory-mapped registers.

System Architecture Overview

- **Main Components:**
- **HPS (ARM Cortex-A9)**
  - Generates Random 128 bit Key
  - Read the input plaintext
  - Controls encryption
  - Reads FPGA ciphertext
  - Decrypt FPGA ciphertext
- **FPGA Fabric**
  - AES-128 encryption core
  - Control & status logic
- **HPS–FPGA Lightweight AXI Bridge**
  - Memory-mapped communication
- **Communication Style:**
  Register-based (PIO peripherals)



Hardware Architecture (FPGA Side)

FPGA Modules:

- AES-128 encryption core (128-bit data path)
- Input registers (4*32 bits plaintext)
- Key registers (4*32 bits key)
- Output registers (4*32 bits ciphertext)
- Control register (START)
- Status register (BUSY, DONE)

Clock Domain:

- CLOCK_50 (50 MHz)

HPS ↔ FPGA Memory Map

- **Bridge:**

  -HPS Lightweight AXI → Avalon-MM

  - IN0_addr-IN3_addr: plaintext

  - KEY0_addr-KEY3_addr: Key

  - ENC0_addr-ENC3_addr: Ciphertext

  - CTRL_addr: Control

  only bit[0] used for START signal

  wire start = ctrl_conduit[0];

  - STAUTS_addr: STATUS

  bit[0] used for DONE, bit[1] used for BUSY

  assign status_conduit = {30'd0, done, busy};

Register	**Address Offset**	**Width**	**Direction**
IN0_addr–IN3_addr	INx_BASE	32-bit	HPS → FPGA
KEY0_addr – KEY3_addr –	KEYx_BASE	32-bit	HPS → FPGA
ENC0_addr – ENC3_addr	ENCx_BASE	32-bit	FPGA → HPS
CTRL_addr	CTRL_BASE	32-bit	HPS → FPGA
STATUS_addr	STATUS_BASE	32-bit	FPGA → HPS

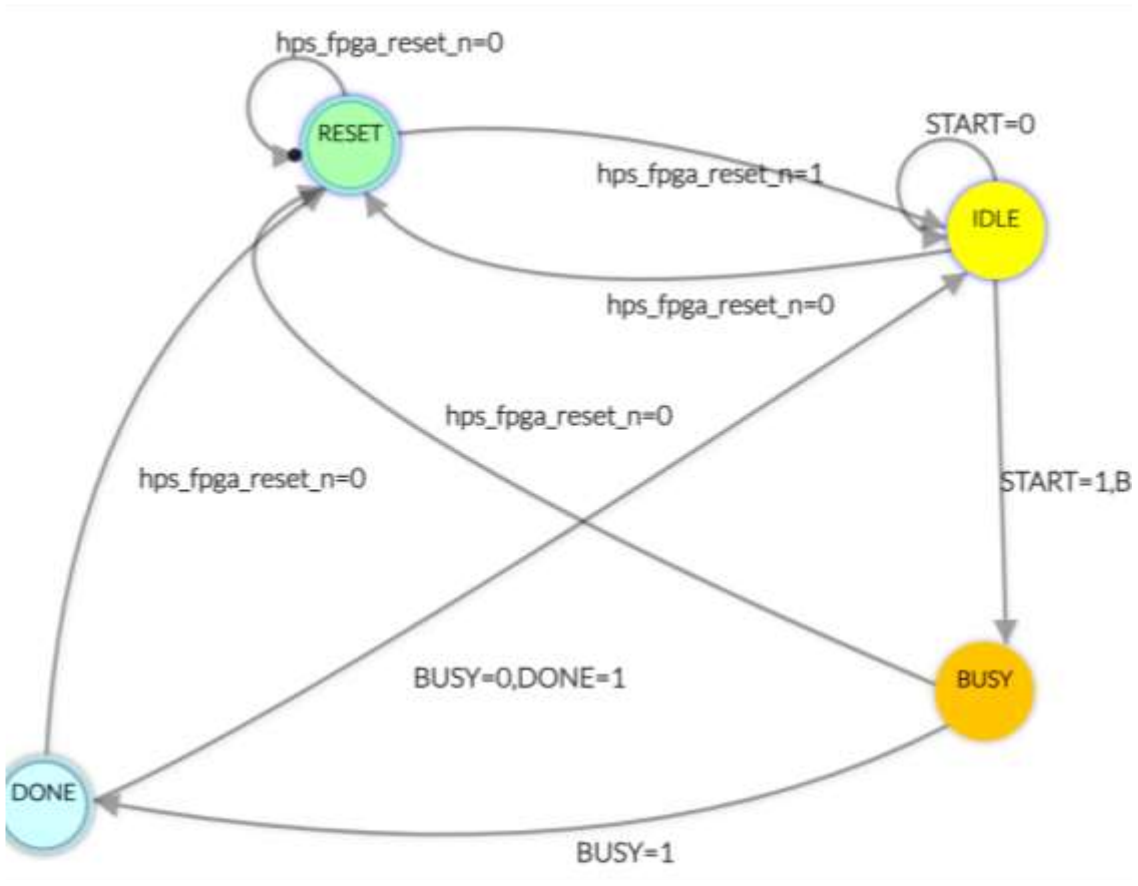
Control & Status Register Definition

CTRL Register Bit	Name	Function
0	START	Start encryption

STATUS Register Bit	Name	Function
0	BUSY	Encryption in progress
1	DONE	Encryption completed

FPGA Control Logic (FSM Behavior)

```
always @(posedge CLOCK_50 or negedge hps_fpga_reset_n) begin //RESET state
 if (!hps_fpga_reset_n) begin //Idle state
   pt_latched  <= 128'd0;
   key_latched <= 128'd0;
   enc_latched <= 128'd0;
   busy <= 1'b0;//not busy
   done <= 1'b0;//
 end else begin  if (start && !busy) begin//Busy state
    pt_latched  <= in;
    key_latched <= key128;
    busy <= 1'b1;
    done <= 1'b0;
   end
if (busy) begin//Done state
    enc_latched <= enc_comb;
    busy <= 1'b0;
    done <= 1'b1;
   end
 end
end
```



Software Workflow (HPS Side)

C Program Steps:

1. Map FPGA registers using /dev/mem
2. Generate random 128-bit AES key
3. Read plaintext from user
4. Pack plaintext & key into 32-bit words Registers
5. Write KEY and plaintext registers to FPGA
6. Clear DONE flag
7. Issue START pulse
8. Poll STATUS until DONE = 1
9. Read ciphertext from FPGA
10. Decrypt the FPGA ciphertext

Start / Handshake Sequence

```
*CTRL_addr = 0x0;   // start=0
BARRIER();
*INx_addr  = plaintxt
*KEYx_addr = Generated Key
BARRIER();
*CTRL_addr = 0x1;   // start =1
BARRIER();
*CTRL_addr = 0x0;
```

Polling

- while(((*STATUS_addr) & 0x2) == 0);

FPGA_ciphertext= *ENCx_addr

Timing Behavior

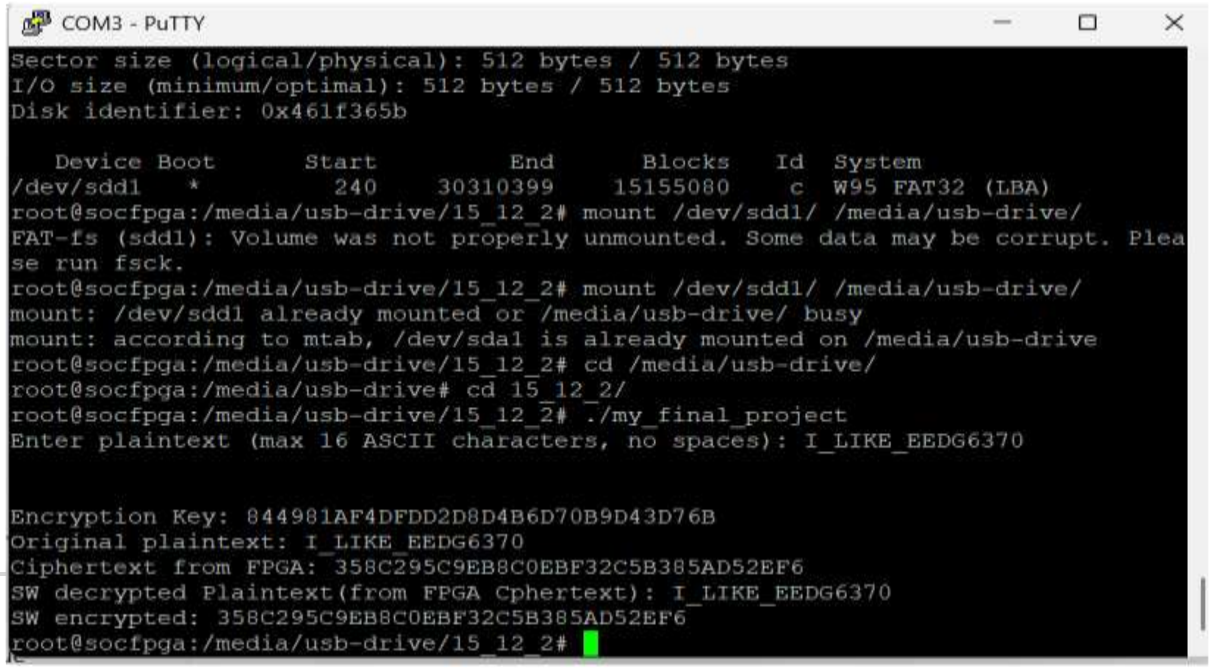
START asserted → inputs latched

Compute : AES core runs inside FPGA)

AES combinational output captured

DONE : write on HPS

– Experimental Results



```
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x461f365b

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1   *         240    30310399    15155080    c  W95 FAT32 (LBA)
root@socfpga:/media/usb-drive/15_12_2# mount /dev/sdd1/ /media/usb-drive/
FAT-fs (sdd1): Volume was not properly unmounted. Some data may be corrupt. Plea
se run fsck.
root@socfpga:/media/usb-drive/15_12_2# mount /dev/sdd1/ /media/usb-drive/
mount: /dev/sdd1 already mounted or /media/usb-drive/ busy
mount: according to mtab, /dev/sda1 is already mounted on /media/usb-drive
root@socfpga:/media/usb-drive/15_12_2# cd /media/usb-drive/
root@socfpga:/media/usb-drive# cd 15_12_2/
root@socfpga:/media/usb-drive/15_12_2# ./my_final_project
Enter plaintext (max 16 ASCII characters, no spaces): I_LIKE_EEDG6370


Encryption Key: 844981AF4DFDD2D8D4B6D70B9D43D76B
Original plaintext: I_LIKE_EEDG6370
Ciphertext from FPGA: 358C295C9EB8C0EBF32C5B385AD52EF6
SW decrypted Plaintext(from FPGA Cphertext): I_LIKE_EEDG6370
SW encrypted: 358C295C9EB8C0EBF32C5B385AD52EF6
root@socfpga:/media/usb-drive/15_12_2#
```

- Original plaintext : **I_LIKE_EEDG6370**
- Encryption Key: **844981AF4DFDD2D8D4B6D70B9D43D76B**
- Ciphertext from FPGA: **358C295C9EB8C0EBF32C5B385AD52EF6**
- SW decrypted plaintext(from FPGA ciphertext)/Plaintext from FPGA: **I_LIKE_EEDG6370**
- SW encrypted: **358C295C9EB8C0EBF32C5B385AD52EF6**

•**Verification:**
SW encryption : ✓ returned FPGA encryption
FPGA Decryption (SW): ✓ returns original plaintext

Debug & Validation Techniques

Printed CTRL and STATUS registers

Verified BUSY/DONE transitions

Printed raw FPGA registers (IN, KEY, OUT)

Compared FPGA ciphertext vs software AES

Identified and fixed endianness mismatch

Challenges & Solutions

Challenge	Solution
Incorrect ciphertext	Fixed 128-bit word ordering
Start misbehavior	Used start pulse (0→1→0)
Debug difficulty	Status register instrumentation

Key Learnings

HW/SW co-design requires **strict data ordering discipline**

Avalon PIOs expose **register interfaces**, not variables

Non-blocking assignments affect control timing

Clear handshake protocols are essential

Debug visibility (STATUS/CTRL) saves time

Future Work:

DMA-based data transfer

Performance benchmarking