Project Presentation

Embedded Linux - custom hardware

OPERATING SYSTEMS FOR EMBEDDED SYSTEMS 2023/2024

Badas Lorenzo
Bevilacqua Piero
Delbosco Fabio
Zavatta Chiara

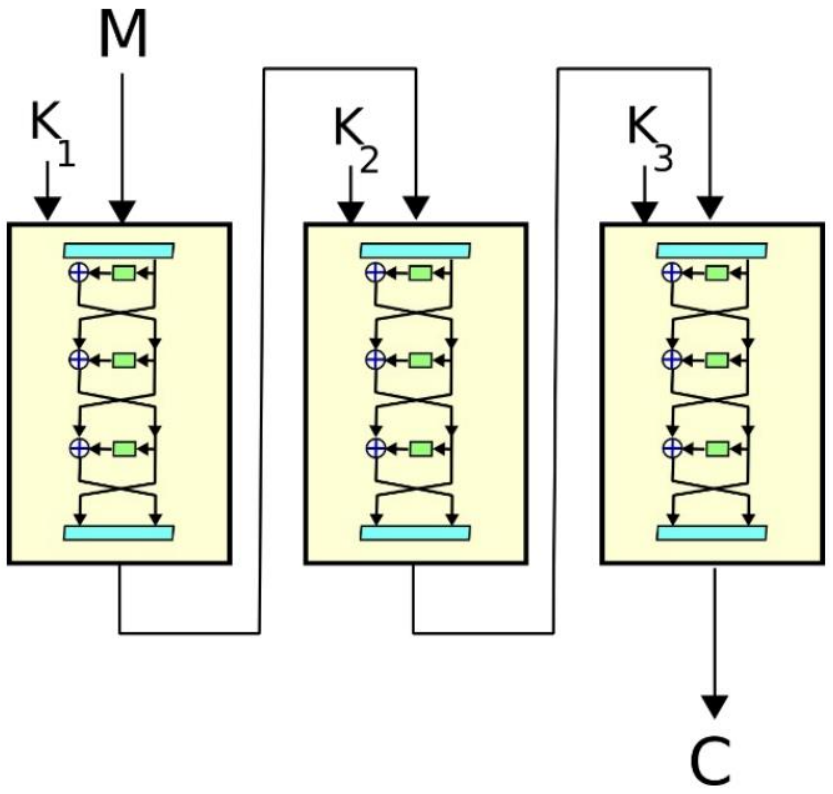
Outline

- DES3 insight
- Modification of test component DES3
- Baremetal testing
- Setup Petalinux
- Device driver development
- Application development

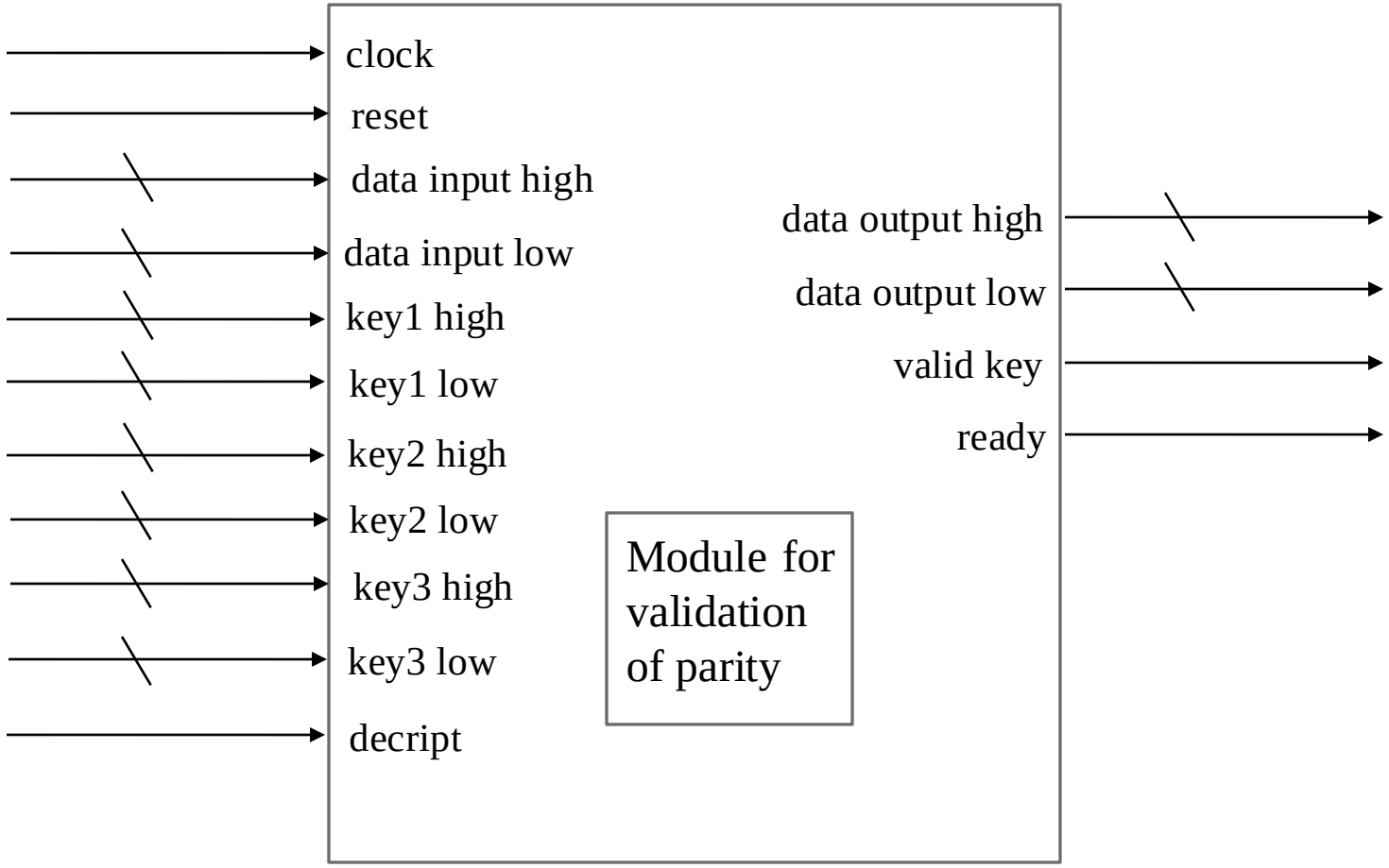
DES3

Triple Data Encryption Standard

- Symmetric algorithm
- Repeats DES 3 times, with 64 bits keys (56 bits used for keys and 8 bit for parity checking)
- 2 or 3 different keys



The DES3 core



Modifications of Test Component

Des3 Test Component Modification:

- Implemented parity bit checking (odd parity) and transformation of the keys from 56 bit to 64 bits
- Addition of a module implementing the validation of the parity
- Transformation of 3 inputs of 64 bits into 1 input of 192 bits
- Development and verification of a bare-metal application using Vitis for thorough testing

Testbench

```
module tb;

    reg clk, decrypt;
    reg [63:0] desIn;
    reg [63:0] key1, key2, key3;
    wire [63:0] desOut;
    wire ready;
    wire valid_key;

    des3 dut(
            .clk(clk),
            .valid_key(valid_key),
            .ready(ready),
            .desIn(desIn),
            .decrypt(decrypt),
            .key1_64(key1),
            .key2_64(key2),
            .key3_64(key3),
            .desOut(desOut)
            );
```

```
    initial begin
        clk = 0;
        dut.ready = 0;
        dut.desOut_reg = 0;
//        dut.key1_reg = 0;
//        dut.key2_reg = 0;
//        dut.key3_reg = 0;
        decrypt = 0;
        key1 = 64'h6d6e73646e7a6e62;
        key2 = 64'h6164617331323234;
        key3 = 64'h3437373838313838;
        desIn = 64'h6369616F6369616F;
        #510
        key1 = 64'h6d6e73646e7a6e63;
        key2 = 64'h6164617331323234;
        key3 = 64'h3437373838313838;
        #510
        $stop;
    end
    always #5 clk = ~clk;
endmodule
```

Baremetal testing

```c
int main()
{
    init_platform();

    DES3_AXI_mWriteReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_KEY1_HI_OFFSET, 0x6d6e7364);
    DES3_AXI_mWriteReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_KEY1_LO_OFFSET, 0x6e7a6e62);

    DES3_AXI_mWriteReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_KEY2_HI_OFFSET, 0x61646173);
    DES3_AXI_mWriteReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_KEY2_LO_OFFSET, 0x31323234);

    DES3_AXI_mWriteReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_KEY3_HI_OFFSET, 0x34373738);
    DES3_AXI_mWriteReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_KEY3_LO_OFFSET, 0x38313839);

    uint32_t ready = DES3_AXI_mReadReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_READY_OFFSET);

    DES3_AXI_mWriteReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_DES_IN_HI_OFFSET, 0x6369616F);
    DES3_AXI_mWriteReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_DES_IN_LO_OFFSET, 0x6369616F);

    while (DES3_AXI_mReadReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_READY_OFFSET) == ready);

    xil_printf("desOut_HI: 0x%X\n", DES3_AXI_mReadReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_DES_OUT_HI_OFFSET));
    xil_printf("desOut_LO: 0x%X\n", DES3_AXI_mReadReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_DES_OUT_LO_OFFSET));
    xil_printf("valid_key: 0x%X\n", DES3_AXI_mReadReg(XPAR_DES3_AXI_0_S00_AXI_BASEADDR, DES3_AXI_VALID_KEY_OFFSET));


    cleanup_platform();
    return 0;
}
```

Setup Petalinux

Main Steps:

- Create Project
- Configure Hardware
- Create Kernel Module
- Create Application
- Package Petalinux Image
- Booting PetaLinux Image on Hardware with an SD Card

Device driver development

Driver Implementation:

- Utilization of necessary function calls for device access.
- Implementation of key functionalities such as read, write, open, and close.
- Mapping of AXI registers to virtual memory.
- Integration with the Linux kernel for seamless interaction with the hardware.

Open and release functions

```c
/* This is called whenever a process attempts to open the device file */
static int device_open(struct inode *inode, struct file *file)
{
    pr_info("device_open(%p)\n", file);

    try_module_get(THIS_MODULE);
    return SUCCESS;
}

static int device_release(struct inode *inode, struct file *file)
{
    pr_info("device_release(%p,%p)\n", inode, file);

    module_put(THIS_MODULE);
    return SUCCESS;
}
```

```c
/* This function is called whenever a process which has already opened the
 * device file attempts to read from it.
 */
static ssize_t device_read(struct file *file, /* see include/linux/fs.h   */
                           char __user *buffer, /* buffer to be filled  */
                           size_t length, /* length of the buffer     */
                           loff_t *offset)
{
    int i;
    unsigned int result[13];
    pr_info("Starting read from AXI Registers\n");
    result[ 0] = ioread32(axi_slv_regs + DES3_AXI_KEY1_HI_OFFSET);
    result[ 1] = ioread32(axi_slv_regs + DES3_AXI_KEY1_LO_OFFSET);
    result[ 2] = ioread32(axi_slv_regs + DES3_AXI_KEY2_HI_OFFSET);
    result[ 3] = ioread32(axi_slv_regs + DES3_AXI_KEY2_LO_OFFSET);
    result[ 4] = ioread32(axi_slv_regs + DES3_AXI_KEY3_HI_OFFSET);
    result[ 5] = ioread32(axi_slv_regs + DES3_AXI_KEY3_LO_OFFSET);
    result[ 6] = ioread32(axi_slv_regs + DES3_AXI_DES_IN_HI_OFFSET);
    result[ 7] = ioread32(axi_slv_regs + DES3_AXI_DES_IN_LO_OFFSET);
    result[ 8] = ioread32(axi_slv_regs + DES3_AXI_DECRYPT_OFFSET);
    result[ 9] = ioread32(axi_slv_regs + DES3_AXI_DES_OUT_HI_OFFSET);
    result[10] = ioread32(axi_slv_regs + DES3_AXI_DES_OUT_LO_OFFSET);
    result[11] = ioread32(axi_slv_regs + DES3_AXI_READY_OFFSET);
    result[12] = ioread32(axi_slv_regs + DES3_AXI_VALID_KEY_OFFSET);

    pr_info("Read from AXI Registers completed\n");
    pr_info("des_in  : 0x%08X%08X\n", result[6], result[7]);
    pr_info("decrypt : 0x%08X\n",     result[8]);
    pr_info("key1    : 0x%08X%08X\n", result[0], result[1]);
    pr_info("key2    : 0x%08X%08X\n", result[2], result[3]);
    pr_info("key3    : 0x%08X%08X\n", result[4], result[5]);
    pr_info("des_out : 0x%08X%08X\n", result[9], result[10]);
    pr_info("ready   : 0x%08X\n",     result[11]);
    pr_info("validkey: 0x%08X\n\n",   result[12]);
```

Read function

```c
    // checks
    if (length < 52) {
        pr_info("lenght is %d\n", length);
        pr_alert("length < 52\n Nothing done\n");
        return 0;
    }

    copy_to_user(buffer, result, sizeof(result));

    pr_info("Read 52 bytes\n");
    return 52;
}
```

```c
/* called when somebody tries to write into our device file. */
static ssize_t device_write(struct file *file, const char __user *buffer,
                            size_t length, loff_t *offset)
{
    char kernel_buffer[36] = {0};

    pr_info("dentro device_write");

    // checks
    if (length != 36) {
        pr_info("lenght is %d\n", length);
        pr_alert("Tried to write length != 36\nNothing done\n");
        return 0;
    }
    if (*offset != 0) {
        pr_info("Offset != 0\nOffset was ignored\n");
    }

    // write operation
    for (int i = 0; i < length; i++) {
        get_user(kernel_buffer[i], buffer + i);
    }
    pr_info("Starting write to AXI Registers\n");
    iowrite32(*((unsigned int*)(kernel_buffer   )), axi_slv_regs + DES3_AXI_KEY1_HI_OFFSET);
    iowrite32(*((unsigned int*)(kernel_buffer+ 4)), axi_slv_regs + DES3_AXI_KEY1_LO_OFFSET);
    iowrite32(*((unsigned int*)(kernel_buffer+ 8)), axi_slv_regs + DES3_AXI_KEY2_HI_OFFSET);
    iowrite32(*((unsigned int*)(kernel_buffer+12)), axi_slv_regs + DES3_AXI_KEY2_LO_OFFSET);
    iowrite32(*((unsigned int*)(kernel_buffer+16)), axi_slv_regs + DES3_AXI_KEY3_HI_OFFSET);
    iowrite32(*((unsigned int*)(kernel_buffer+20)), axi_slv_regs + DES3_AXI_KEY3_LO_OFFSET);
    iowrite32(*((unsigned int*)(kernel_buffer+24)), axi_slv_regs + DES3_AXI_DES_IN_HI_OFFSET);
    iowrite32(*((unsigned int*)(kernel_buffer+28)), axi_slv_regs + DES3_AXI_DES_IN_LO_OFFSET);
    iowrite32(*((unsigned int*)(kernel_buffer+32)), axi_slv_regs + DES3_AXI_DECRYPT_OFFSET);
    pr_info("Write to AXI Registers completed\n");
    /* Return the number of input characters used. */
    return 36;
}
```

Write function

Application development

Application Features:

- Interfacing with the driver for encryption/decryption operations.
- Reading keys and input data from files.
- Writing encrypted/decrypted data to an output file.
- Command-line argument parsing for flexible operation modes.
- Padding of the text

Des3app - Padding

PKCS#5, PKCS#7, SSL, TLS:

- Always add padding, even when unnecessary.
- Each byte of padding is equal to the length of the padding.

```
length = get_file_length(datainfile);
mod_length = length % 8;
padding = 8 - mod_length;
```

```
fread(buffer, 1, mod_length, datainfile);
for (i = 0; i < padding; i++) {
    buffer[mod_length+i] = padding;
}
```

Code used for padding during encryption

Des3app - Padding

- To know how much padding there is, read the value of the last byte

```
uint8_t ref = buffer[7];
uint8_t found_different = 0;
if (ref <= 8 && ref > 0) {
    for (i = 1; i < ref; i++) {
        if (buffer[7-i] != ref) {
            found_different = 1;
        }
    }
    if (!found_different && ref != 8) {
        fwrite(buffer, 1, 8-ref, dataoutfile);
        fclose(dataoutfile);
        return;
    }
    else if (!found_different && ref == 8) {
        fclose(dataoutfile);
        return;
    }
}
```

Code used for padding during decryption

Demo time