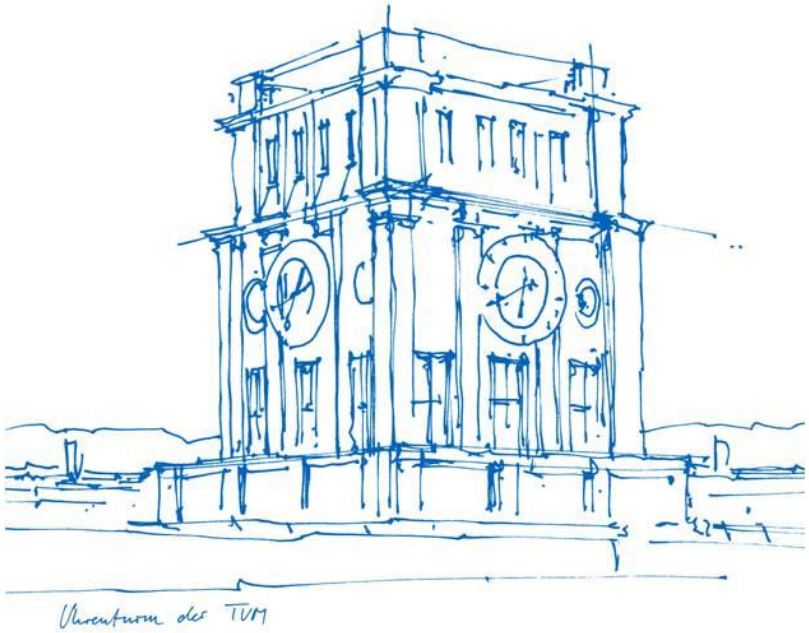
Chair of High-Power Converter Systems
TUM School of Engineering and Design
Technical University of Munich

Gra Projekt: Secure-Memory Unit

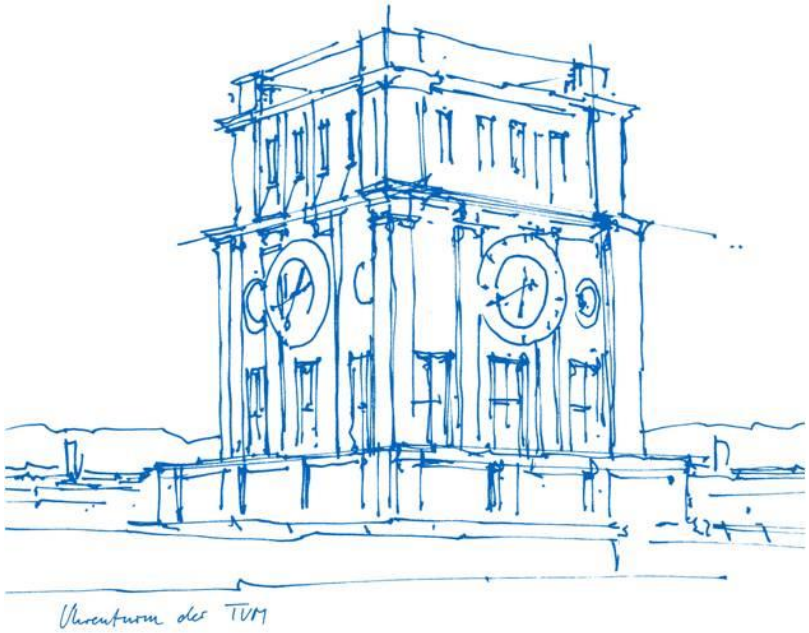
T033

Tianding Hou, Xinrui Zheng, Yuze Ji



Contents

1. Project Introduction
2. Implement Detail
3. Literature Research
4. Integration in TinyCpu
5. Test



1. Project Introduction

Project Introduction

- This project aims to develop a secure-memory unit instead of the simple memory unit, integrating multiple security features to protect data integrity and confidentiality.
- Features:
  - Pseudo Random Number Generator(PRNG),
  - Encryption then Store in preferred Endianness,
  - scrambled logic address,
  - Storage with Parity Check.

2. Implement Details

Implement Details

- Overview

Inputs

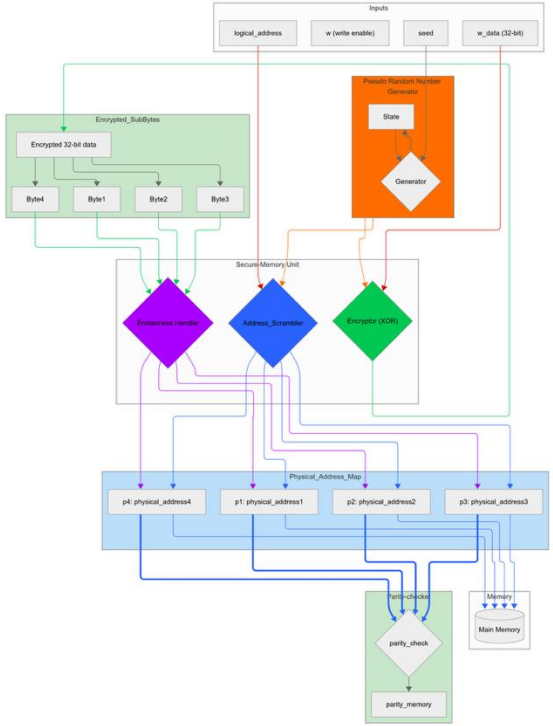
- clk: bool (clock input)
- addr: uint32_t
- wdata: uint32_t
- r: bool
- w: bool
- fault: uint32_t
- faultBit: sc_bv<4>

Outputs

- rdata: uint32_t
- ready: bool
- error: bool

Implement Details

- Overview

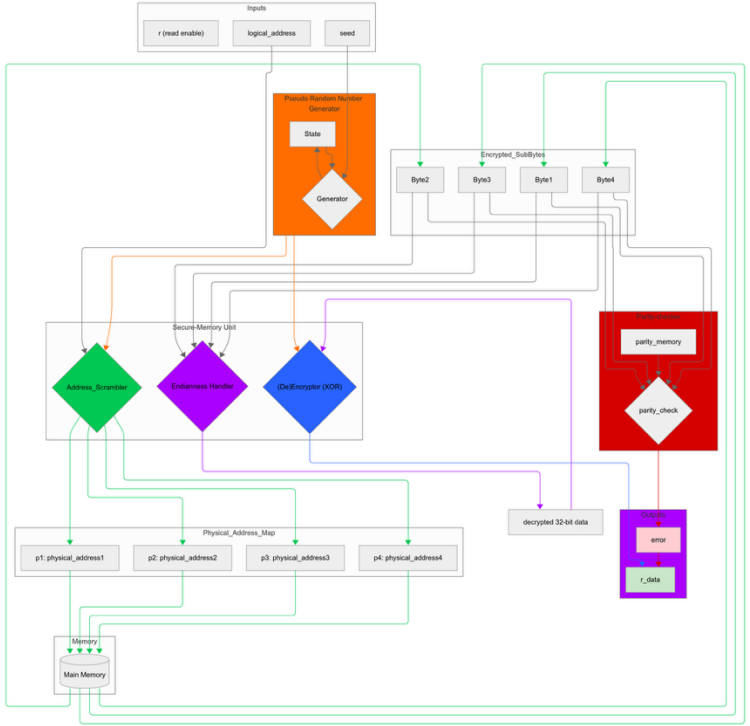


- Writing to Memory

Implement Details

- Overview

- Read from Memory



2. Implement Details

Implement Details

- CLI Implement

- CLI Parameters:
  - Command line parameters allow users to configure max_cycles, endianness, latency settings, seed value, and other simulation parameters.

```
Positional arguments:
  <file>  The path to input file: list of requests
Optional arguments:
  -c, --cycles N                  Number of cycles (uint32_t, default: 1)
  -t, --tf PATH                   Path to the trace file (string)
  -e, --endianness N              Endianness (uint8_t): 0=Little-Endian, 1=Big-Endian
  -s, --latency-scrambling N      Latency for Address Scrambling (uint32_t)
  -n, --latency-encrypt N         Latency for Encryption/Decryption (uint32_t)
  -m, --latency-memory-access N   Latency for Memory Access (uint32_t)
  -d, --seed N                    Seed for PRNG (uint32_t)
  -h, --help                      Show this help message
```

Implement Details

- CLI Implement

- Input Format
  - The input is a formatted CSV file containing a list of memory requests, which are processed by the simulation.(with header in first line)

e.g.:

Type	Address	Data	Fault	Fault-bit
W	0x100	0x15		
R	0x100			
W	0x123	0x15	0x100	8
R	0x100			

Implement Details

- CLI Implement

Result and Request structure

```
struct Result
{
  uint32_t cycles;
  uint32_t errors;
};

struct Request
{
  uint32_t addr;
  uint32_t data;
  uint8_t r;
  uint8_t w;
  uint32_t fault;
  uint8_t faultBit;
};
```

CLI body: Parse_cli function with call of getopt_long

```
int parse_cli(int argc, char *argv[], uint32_t *cycles, char **tracefile, char **input_file,
              uint8_t *endianness, uint32_t *latency_scrambling, uint32_t *latency_encrypt,
              uint32_t *latency_memory_access, uint32_t *seed)
{
    static struct option long_options[] = {
        {"cycles", required_argument, 0, 'c'},
        {"tf", required_argument, 0, 't'},
        {"endianness", required_argument, 0, 'e'},
        {"latency-scrambling", required_argument, 0, 's'},
        {"latency-encrypt", required_argument, 0, 'n'},
        {"latency-memory-access", required_argument, 0, 'm'},
        {"seed", required_argument, 0, 'd'},
        {"help", no_argument, 0, 'h'},
        {0, 0, 0, 0}};
    int opt;
    while ((opt = getopt_long(argc, argv, "c:t:e:s:n:m:d:h", long_options, NULL)) != -1)
```

Implement Details

- CLI Implement

Handle each flag:

```
switch (opt)
{
case 'c':
    *cycles = (uint32_t)strtoul(optarg, NULL, 10);
    break;
case 't':
    *tracefile = optarg;
    break;
case 'e':
    *endianness = (uint8_t)strtoul(optarg, NULL, 10);
    if (*endianness != 0 && *endianness != 1)
    {
        fprintf(stderr, "Invalid endianness value. Use 0 for Little-Endian or 1 for Big-Endian.\n");
        return EXIT_FAILURE;
    }
    break;
case 's':
    *latency_scrambling = (uint32_t)strtoul(optarg, NULL, 10);
    break;
case 'n':
    *latency_encrypt = (uint32_t)strtoul(optarg, NULL, 10);
    break;
case 'm':
    *latency_memory_access = (uint32_t)strtoul(optarg, NULL, 10);
    break;
case 'd':
    *seed = (uint32_t)strtoul(optarg, NULL, 10);
    break;
case 'h':
    print_help(argv[0]);
    return EXIT_SUCCESS;
default:
    print_usage(argv[0]);
    return EXIT_FAILURE;
}
```

Implement Details

- CLI Implement

Handle positional argument:

```
if (optind >= argc)
{
    fprintf(stderr, "No input file specified.\n");
    return 1;
}

while (optind < argc)
{
    if (*input_file != NULL)
    {
        fprintf(stderr, "Too many input files specified.\n");
        return 1;
    }
    *input_file = argv[optind];
    optind++;
}
```

2. Implement Details

Implement Details

- Parsing Requests

Open input .csv File :

```c
FILE *validate_and_open_read(const char *path)
{
    FILE *file;
    if (!(file = fopen(path, "r")))
    {
        return NULL;
    }

    struct stat statbuf;
    if (fstat(fileno(file), &statbuf))
    {
        fclose(file);
        return NULL;
    }
    if (!S_ISREG(statbuf.st_mode) || statbuf.st_size <= 0)
    {
        fclose(file);
        return NULL;
    }
    return file;
}
```

```c
FILE *file = validate_and_open_read(input_file);
if (file == NULL)
{
    fprintf(stderr, "Failed to open input file '%s'.\n", input_file);
    return 1;
}
fseek(file, 0, SEEK_END);
long file_size = ftell(file);
fseek(file, 0, SEEK_SET);

char *input = (char *)malloc(file_size + 1);
if (input == NULL)
{
    fprintf(stderr, "Failed to allocate memory for input file.\n");
    fclose(file);
    return 1;
}
fread(input, 1, file_size, file);
fclose(file);
input[file_size] = '\0';
```

Implement Details

- Parsing Requests

The implementation on parsing is similar to our implementation in Tiny_assembler.c

```
Type,Address,Data,Fault,Fault-bit
W,0x100,0x15,,
R,0x100,,,
W,0x123,0x15,0x38AAA1C8,8
R,0x100,,,
```

(content to parse in this project)

```
beq x1,x2, 0x111
li x2, 255
add x3, x2, x1
#sljcnlan cl
beq x4,x2, 0x111
```

(content to parse in tiny_assembler)

Helper Method:

```
/**
 * @brief Extract a single comma-separated argument from a line.
 *
 * @param line Input string line.
 * @param counter Current character index in the line.
 * @param arg_buf Buffer to store the extracted argument.
 * @return int Updated character index in line after parsing.
 */
int extract_arg(const char *line, int counter, char *arg_buf)
```

```
/**
 * @brief Parse a string as uint32_t, supporting decimal and hexadecimal formats.
 *
 * @param inputString The string to parse.
 * @return uint32_t Parsed integer or -1 (UINT32_MAX) on failure.
 */
static uint32_t parse_uint32(const char *inputString)
```

Implement Details

- Parsing Requests

Helper Method:

```
/**
 * @brief Parse a string as uint8_t.
 *
 * @param inputString The string to parse.
 * @return uint8_t Parsed byte or -1 (UINT8_MAX) on failure.
 */
static uint8_t parse_uint8(const char *inputString)
```

```
/**
 * @brief Parse the next CSV line and extract the request fields.
 *
 * @param code Pointer to input text.
 * @param type Type buffer (e.g., R/W/F).
 * @param address Address buffer.
 * @param data Data buffer.
 * @param fault Fault address buffer.
 * @param faultbit Fault bit index buffer.
 * @return const char* Pointer to start of next line or the end of last line if no more lines are available.
 */
const char *split_next_line(const char *code, char *type, char *address, char *data, char *fault, char *faultbit)
```

Implement Details

- Parsing Requests

Parse_request():

```
/**
 * @brief Parse CSV-formatted memory requests into a Request array.
 *
 * @param input Raw string content of the input file.
 * @param numRequests Pointer to store number of parsed requests.
 * @param requests Pointer to pointer of dynamically allocated Request array.
 * @return int 0 on success, -1 on error.
 */
```

```
while (*new_line == '\n' || *new_line == ' ' || *new_line == '\t')
{
    new_line++;
} // Skip leading whitespace characters
  You, last week • add some debug prints, add Doxygen style commen…
while (*new_line != '\n')
{
    if (*new_line == '\0')
    {
        break;
    }
    new_line++;
} // Skip the first line as it is Header
```

```
while ((new_line = split_next_line(new_line,
                                   type_buf, addr_buf, data_buf,
                                   fault_buf, faultbit_buf)))
{
```

```
char t = (char)tolower((unsigned char)type_buf[0]);
```

```
if (t == 'r')
{
    req.r = 1;
    req.addr = parse_uint32(addr_buf);
    if (req.addr == (uint32_t)-1)
    {
        fprintf(stderr, "Invalid address in read request: %s\n", addr_buf);
        free(arr);
        return -1;
    }
}
```

```
arr[count++] = req;
```

2. Implement Details

Implement Details

- Run_Simulation.cpp
  - Main.c passes arguments and flags by run_simulation() to run_simulation.cpp

```
struct Result run_simulation(
    uint32_t max_cycles,
    const char *tracefile,
    uint8_t endianness,
    uint32_t latency_scrambling,
    uint32_t latency_encryption,
    uint32_t latency_memory_access,
    uint32_t seed,
    uint32_t numRequests,
    struct Request *requests) {
```

(run_simulation.cpp)

```
extern struct Result run_simulation(
    uint32_t cycles,
    const char *tracefile,
    uint8_t endianness,
    uint32_t lat_scramble,
    uint32_t lat_encrypt,
    uint32_t lat_mem,
    uint32_t seed,
    uint32_t numRequests,
    struct Request *requests);
```

(main.c)

Implement Details

- Run_Simulation.cpp

  - Run_simulation uses the parameter to construct our correspondent Component and starts the simulation.

```
sc_clock clk("clk", 1, SC_NS);
sc_signal<uint32_t> addr, wdata, fault;
sc_signal<bool>     r, w;
sc_signal<sc_dt::sc_bv<4>> faultbit;
sc_signal<uint32_t> rdata;
sc_signal<bool>     ready, error;
SecureMemoryUnit smu("smu", endianness, latency_scrambling, latency_encryption, latency_memory_access, seed);
```

```
if(tracefile != NULL) {
    sc_trace_file *tf = sc_create_vcd_trace_file(tracefile);
    sc_trace(tf, clk, "clk");
    sc_trace(tf, addr, "addr");
    sc_trace(tf, wdata, "wdata");
    sc_trace(tf, r, "r");
    sc_trace(tf, w, "w");
    sc_trace(tf, fault, "fault");
    sc_trace(tf, faultbit, "faultbit");
    sc_trace(tf, ready, "ready");
    sc_trace(tf, error, "error");
    sc_trace(tf, rdata, "rdata");
    }
```

```
smu.clk(clk);
smu.addr(addr);
smu.wdata(wdata);
smu.r(r);
smu.w(w);
smu.fault(fault);
smu.faultbit(faultbit);
smu.rdata(rdata);
smu.ready(ready);
smu.error(error);
```

```
fault.write(UINT32_MAX);
faultbit.write("0000");
addr.write(0);
wdata.write(0);
r.write(false);
w.write(false);
sc_start(1, SC_NS);
uint32_t err_count = 0;
uint32_t cyc_count = 0;
```

Implement Details

- Run_Simulation.cpp

  - Main loop body:

```
for (uint32_t i = 0; i < numRequests && cyc_count < max_cycles; ++i){
```

```
if (requests[i].fault != UINT32_MAX) {
    fault.write(requests[i].fault);
    faultbit.write(requests[i].faultBit & 0xF);
} else {
    fault.write(UINT32_MAX);
}
```

```
if (requests[i].r) {
    r.write(true);
    w.write(false);
    addr.write(requests[i].addr);
} else if(requests[i].w) {
    w.write(true);
    r.write(false);
    wdata.write(requests[i].data);
    addr.write(requests[i].addr);
}
```

```
sc_start(1, SC_NS);        You, 3 weeks ago • imple
cyc_count++;
while (!ready.read() && cyc_count < max_cycles) {
    sc_start(1, SC_NS);
    cyc_count++;
}
```

```
r.write(false);
w.write(false);
addr.write(0);
fault.write(UINT32_MAX);
faultbit.write("0000");
wdata.write(0);
```

```
if (error.read()){
    err_count++;
}
```

2. Implement Details

Implement Details

- secure-Memory-Unit.hpp
  - Result and request struct:

```
struct Request {
    uint32_t addr;
    uint32_t data;
    uint8_t  r;
    uint8_t  w;
    uint32_t fault;
    uint8_t  faultBit;
};

struct Result {
    uint32_t cycles;
    uint32_t errors;
};
```

```
extern "C" struct Result run_simulation(
    uint32_t max_cycles,
    const char *tracefile,
    uint8_t endianness,
    uint32_t latency_scrambling,
    uint32_t latency_encryption,
    uint32_t latency_memory_access,
    uint32_t seed,
    uint32_t numRequests,
    struct Request *requests);
```

Implement Details

- secure-Memory-Unit.hpp
  - SC_MODULE

```
SC_MODULE(SecureMemoryUnit) {

  SC_HAS_PROCESS(SecureMemoryUnit);

    sc_in<bool> clk;
    sc_in<uint32_t> addr;
    sc_in<uint32_t> wdata;
    sc_in<bool> r;
    sc_in<bool> w;
    sc_in<uint32_t> fault;
    sc_in<sc_bv<4>> faultbit;

    std::map<uint32_t, uint8_t> memory;
    std::map<uint32_t, bool> parity_memory;
    uint32_t state;
    uint32_t scrambler_key;
    uint32_t encryptor_key;
    uint8_t isBigEndian;
    bool error_flag ;
    uint32_t latency_scrambling;
    uint32_t latency_encryption;
    uint32_t latency_memory_access;
    uint32_t seed;


    sc_out<uint32_t> rdata;
    sc_out<bool> ready;
    sc_out<bool> error;
```

```
SecureMemoryUnit(sc_module_name name,
        uint8_t endianness, uint32_t latency_scrambling, uint32_t latency_encryption,
        uint32_t latency_memory_access,uint32_t seed)
  :
  sc_module(name),
  clk("clk"),
  addr("addr"),
  wdata("wdata"),
  r("r"),
  w("w"),
  fault("fault"),
  faultbit("faultbit"),
  rdata("rdata"),
  ready("ready"),
  error("error"),
  state(seed),
  scrambler_key(generate(seed,state)),
  encryptor_key(generate(seed,state)),
  isBigEndian(endianness),
  latency_scrambling(latency_scrambling),
  latency_encryption(latency_encryption),
  latency_memory_access(latency_memory_access),
  seed(seed),
  error_flag(false)
{

  SC_THREAD(process);
  sensitive << clk.pos();
}
```

Key Security Features

- Address Scrambling
- XOR Encryption
- Parity Checking
- PRNG Generator

SC_MODULE : helper methods

- Address Scrambling

Address scrambling uses a 32-bit key to transform logical addresses into physical addresses, making it difficult for attackers to predict memory locations.

```
//address scrambler
void scramble(uint32_t address, uint32_t key, uint32_t &p0, uint32_t &p1, uint32_t &p2, uint32_t &p3) {
    p0 = (address + 0) ^ key;
    p1 = (address + 1) ^ key;
    p2 = (address + 2) ^ key;
    p3 = (address + 3) ^ key;
}
```

SC_MODULE : helper methods

- XOR Encryption

XOR encryption with an 8-bit key provides a lightweight yet effective method to encrypt to encrypt data bytes, ensuring data confidentiality.

```
//encryptor
uint32_t encrypt(uint32_t data, uint32_t key) {
    uint32_t encrypted_data = 0;
    encrypted_data = data ^ key;
    return encrypted_data;
}
```

Here we use a 32-bit key to encrypt the entire data.
It is more efficient and has the same effect.

Our design:
When reading the encrypted_data, we we separate it into 4 bytes and store them them in an array for future use

```
uint8_t encrypted_data_bytes[4] = {0, 0, 0, 0};
    for (int i = 0; i < 4; i++) {
    uint8_t value = 0;
    uint8_t data_w_value = 0;
    int byteIndex = isBigEndian ? (3 - i) : i;
    value = (encrypted_data >> (i * 8)) & 0xFF;
    encrypted_data_bytes[byteIndex] = value;
}
```

SC_MODULE : helper methods

- Parity Checking

Parity checking adds a parity bit to each byte, allowing the detection of single- and improving data reliability.

```
//parity checker
int calculate_parity(uint8_t data) {
    int parity_bit = 0;
    for (int i = 0; i < 8; ++i) {
      parity_bit += (data >> i) & 1;
    }

    return parity_bit & 1;
    }
```

SC_MODULE：helper methods

- PRNG Generator

Provide random key for scrambling and encryption but meanwhiles controllable with seeds.

```
//PRNG generator
uint32_t generate(uint32_t seed,uint32_t &state) {

    state = (state * 950706376) % 2147483647;

    return state;

}
```

LCG-based Pseudo-Random Number Generator (PRNG) module
Generation function: X_{n+1} = (A * X_n + C) mod M
X_{0} is the seed, which is given from the in_port seed;
Generators recommended by Fishman (1990): M = 2^31 - 1 = 2147483647 A = 950706376 B = 0
Source:https://statmath.wu.ac.at/software/src/prng-3.0.2/doc/prng.html/Table_LCG.html

Implementation Detail

- SC_MODULE: void process()

Fault Injection

```
if(fault.read() != UINT32_MAX){
  uint32_t fault_addresses[4] = {0, 0, 0, 0};
  uint32_t index = faultbit.read().to_uint();
  uint32_t fault_address = fault.read();
  uint32_t f0, f1, f2, f3;
  scramble(fault_address, scrambler_key, f0, f1, f2, f3);
  for (uint32_t i = 0; i < latency_scrambling; ++i) wait();
  fault_addresses[0] = f0;
  fault_addresses[1] = f1;
  fault_addresses[2] = f2;
  fault_addresses[3] = f3;

      for (uint32_t i = 0; i < 4; i++)
      {
       if( index == 8){
      parity_memory[fault_addresses[i]] = !parity_memory[fault_addresses[i]];
       }
       else{
        memory[fault_addresses[i]] = memory[fault_addresses[i]] ^ (1 << index);
       }
      }
}
```

Implementation Detail

- SC_MODULE: void process()

Address Scrambling for read & write

```
uint32_t addresses[4] = {0, 0, 0, 0};

if(r.read() || w.read()) {
// Reset error flag and outputs
error_flag = false;
ready.write(false);
rdata.write(0);

  uint32_t address = addr.read();
  uint32_t p0, p1, p2, p3;
  scramble(address, scrambler_key, p0, p1, p2, p3);
  for (uint32_t i = 0; i < latency_scrambling; ++i) wait();
  addresses[0]=p0; addresses[1]=p1; addresses[2]=p2; addresses[3]=p3;


}
```

Implementation Detail

- SC_MODULE: void process()

Write

```
if(w.read()){
  uint32_t data_w = wdata.read();
  uint32_t encrypted_data = encrypt(data_w, encryptor_key);
  for (uint32_t i = 0; i < latency_encryption; ++i) wait();
  for (int i = 0; i < 4; i++) {
  uint8_t value = 0;
  uint8_t data_w_value = 0;
  int byteIndex = isBigEndian ? (3 - i) : i;
  value = (encrypted_data >> (i * 8)) & 0xFF;
  //for debug Endianess
  data_w_value = (data_w >> (i * 8)) & 0xFF;
  printf("Writing byte %d: %02X\n", byteIndex, data_w_value);
  //debug Endianess end
  encrypted_data_bytes[byteIndex] = value;

}
  for (int i = 0; i < 4; i++)
  {
    int parity = calculate_parity(encrypted_data_bytes[i]);
    parity_memory[addresses[i]] = parity;
    write_memory(addresses[i], encrypted_data_bytes[i]);
}
for(uint32_t j = 0; j < latency_memory_access; ++j) wait();
 error.write(false);
```

XOR encrypt

Separate wdata into 4 bytes, Store in an array

Implementation Detail

- SC_MODULE: void process()

Read

```
if(r.read()) {
  error_flag = false;
  for (int i = 0; i < 4; i++) {
    uint8_t value = 0;
    int byteIndex = isBigEndian ? (3-i) : i;
    value = read_memory(addresses[i]) & 0xFF;

    encrypted_data_bytes[byteIndex] = value;
  }
  for(uint32_t j = 0; j < latency_memory_access; ++j) wait();
  uint32_t data_r = 0;
  for (int i = 0; i < 4; i++)
{
  int parity = calculate_parity(encrypted_data_bytes[i]);

 if (parity_memory.find(addresses[i]) != parity_memory.end()) {
    // Check if the address exists in the parity memory
    // If it exists, compare the stored parity with the calculated parity
    if (parity_memory[addresses[i]] != parity) {
        error_flag = true;
    }
}
  data_r |= (encrypted_data_bytes[i] << (i * 8));
}
```

```
if(data_r != 0){
  data_r = encrypt(data_r, encryptor_key);// Decrypt the data
}//skip the decryption if data_r unmodified
for (uint32_t i = 0; i < latency_encryption; ++i) wait();
if(error_flag){
  error.write(true);
  rdata.write(0);
} else {
  error.write(false);
  rdata.write(data_r);
}
```

Assemble back to data_r

Parity check -> see if it is the data we wrote

3. Literature Research - How safe? What attacks?

3.1 Address Scrambling

3.2 XOR Encryption

3.3 Parity Checking

3.4 PRNG Generator

Literature Research

- Address Scrambling

Advantages:

Results show that data scrambling is possible in any kind of memory system, all with low area overhead, small delay penalty and low power consumption.

Disadvantages:

If key is fixed or predictable, attacker can read original content.

https://www.researchgate.net/publication/269294825_Data_scrambling_in_memories_A_security_measure

Literature Research

- XOR Encryption

By itself, the XOR encryption can be very robust if:

- It is based on a long key that will not repeat itself. (e.g. a key that contains as many bits/characters as the plaintext)
- A new key is randomly generated for any new communication.
- The key is kept secret by both the sender and the receiver.

When a large quantity of text is to be encrypted, a shorter repeating encryption key is used to match the length of the plain text. However re-using the same key over and over, or using a shorter repeating key results in a less secure method where the cipher text could be decrypted using a frequency analysis.

https://www.101computing.net/xor-encryption-algorithm/

Literature Research

- Parity Checking

Can it detect every fault?

NO !

Advantages of Parity Bit Method

- Parity bit method is a promising method to detect any odd bit error at the receiver side.
- Overhead in data transmission is low, as only one parity bit is added to the message bits at sender side before transmission.
- Also, this is a simple method for odd bits error detection.

Disadvantages of Parity Bit Method

- The limitation of this method is that only error in a odd number of bits are identified and we also cannot determine the exact location of error in the data bit, therefore, error correction is not possible in this method.
- If the number of bits in even parity check increase or decrease (data changed) but remained to be even then it won't be able to detect error as the number of bits are still even and same goes for odd parity check.

https://www.geeksforgeeks.org/digital-logic/error-detection-codes-parity-bit-method

Literature Research

- PRNG Generator

Advantages:

- **Deterministic:** PRNGs use algorithms that generate predictable sequences from a given seed.
- **Efficient:** They are computationally less expensive compared to generating true random numbers.
- **Applications:** Widely used in simulations, cryptographic algorithms, and randomized algorithms.
- **Security:** Cryptographic PRNGs ensure unpredictability, essential for secure cryptographic protocols.

Potential issues [edit]

In practice, the output from many common PRNGs exhibit artifacts that cause them to fail statistical pattern-detection tests. These include:

- Shorter-than-expected periods for some seed states (such seed states may be called "weak" in this context);
- Lack of uniformity of distribution for large quantities of generated numbers;
- Correlation of successive values;
- Poor dimensional distribution of the output sequence;
- Distances between where certain values occur are distributed differently from those in a random sequence distribution.

https://en.wikipedia.org/wiki/Pseudorandom_number_generator

https://www.twingate.com/blog/glossary/pseudorandom-number-generator

4. Integration in TinyCpu

Integration in TinyCpu

- Component:

```
PC pc;
REGISTER_FILE rf;
ALU alu;
SecureMemoryUnit smu;
CU cu;

MULTIPLEXER_I32 muxReg1;
MULTIPLEXER_I32 muxReg2;
MULTIPLEXER_I32 muxRegW;
MULTIPLEXER_I32 muxMemAddr;
MULTIPLEXER_I32 muxMemR;
MULTIPLEXER_I32 muxAluRo;
```

Bindings:
all the bindings from MemoryUnit and

```
smu.fault.bind(fault_sig);
smu.faultbit.bind(faultbit_sig);
smu.error.bind(error_sig);
fault_sig.write(UINT32_MAX);
faultbit_sig.write("0000");
```

Integration in TinyCpu

- Main function:

```
std::map<unsigned int, unsigned int> program;


program[0x1000] = 0b00000000000000000110010100110011; // INC x10, x0
program[0x1004] = 0b00000000101001010100010110110011; // ADD x11, x10, x10
program[0x1008] = 0b00000000101110101110001100110011; // ADD x12, x11, x11
program[0x100C] = 0b00000000110001100100011010110011; // ADD x13, x12, x12
program[0x1010] = 0b00000000110101101100011100110011; // ADD x14, x13, x13
program[0x1014] = 0b00000000111001110100011110110011; // ADD x15, x14, x14
program[0x1018] = 0b00000000111110111100100000110011; // ADD x16, x15, x15
program[0x101C] = 0b00000001000010000100100010110011; // ADD x17, x16, x16
program[0x1020] = 0b00000001000110001100100100110011; // ADD x18, x17, x17
program[0x1024] = 0b00000001001010010100100110110011; // ADD x19, x18, x18
program[0x1028] = 0b00000001001110011100101000110011; // ADD x20, x19, x19
program[0x102C] = 0b00000001010010100100101010110011; // ADD x21, x20, x20
program[0x1030] = 0b00000001010110101100101100110011; // ADD x22, x21, x21
program[0x1034] = 0b00000001000010110100001010110011; // ADD x5 , x22, x16
program[0x1038] = 0b00000000110101011100000010110011; // ADD x1 , x11, x13
program[0x103C] = 0b00000000000000000110000110110011; // INC x3 , x0
program[0x1040] = 0b00000000000010010010000100110011; // ADD x4 , x4 , x1
program[0x1044] = 0b00000000001100001101000010110011; // SUB x1, x1, x3
program[0x1048] = 0b00000000000000000101110100011100011; // BLT x1, x0, 8
program[0x104C] = 0b00000000000000010100000000001100011; // J x5
program[0x1050] = 0b00000000001001001000100000100011; // SW  x18 , x4
```

Helper Method:

```
for(auto it = initialMemory.begin(); it != initialMemory.end(); it++) {
  smu.set(it->first, it->second);
}
```

```
void set(uint32_t address, uint32_t value){
 uint32_t addresses[4] = {0, 0, 0, 0};
 uint32_t p0, p1, p2, p3;
 uint8_t encrypted_data_bytes[4] = {0, 0, 0, 0};
 scramble(address, scrambler_key, p0, p1, p2, p3);
 addresses[0] = p0;
 addresses[1] = p1;
 addresses[2] = p2;
 addresses[3] = p3;
 uint32_t encrypted_data = encrypt(value, encryptor_key);
 for (int i = 0; i < 4; i++)
 {
     uint8_t value = 0;
     int byteIndex = isBigEndian ? (3 - i) : i;
     value = (encrypted_data >> (i * 8)) & 0xFF;
     write_memory(addresses[byteIndex], value);
 }
 for (int i = 0; i < 4; i++)
 {
     int parity = calculate_parity(encrypted_data_bytes[i]);
     parity_memory[addresses[i]] = parity;
 }
}
```

Integration in TinyCpu

- Main function output :

```
for (size_t i = 10; i < 23; i++)
{
  std::cout << cpu.rf.registers_data[i] << ' ';
}
std::cout << cpu.rf.registers_data[5] << " x5 ";
std::cout << cpu.rf.registers_data[4] << " x4 ";
std::cout << cpu.rf.registers_data[3] << " x3 ";
std::cout << cpu.rf.registers_data[1] << " x1 ";
std::cout << cpu.getMemoryValue(256) << ' ';
std::cout << std::endl;
```

Expected Output :

```
1 2 4 8 16 32 64 128 256 512 1024 2048 4096 4160 x5 55 x4 1 x3 4294967295 x1 55
```

(running with normal Memory unit)

Our Output :

```
1 2 4 8 16 32 64 128 256 512 1024 2048 4096 4160 x5 55 x4 1 x3 4294967295 x1 55
```

5. Testing（Modular Design）

5.1 Input Parsing Tests

5.2 Functional Verification

- 5.2.1 Read / Write Roundtrip
- 5.2.2 Fault Injection & Parity Check

5.3 CLI Parameter Validation

Input Parsing Tests

- All numeric fields must be in hexadecimal format
- Valid lines must satisfy:
  - Operation type must be one of: R / W / F
  - All required fields must be present
  - Fault-bit (if set) must be within [0, 8]

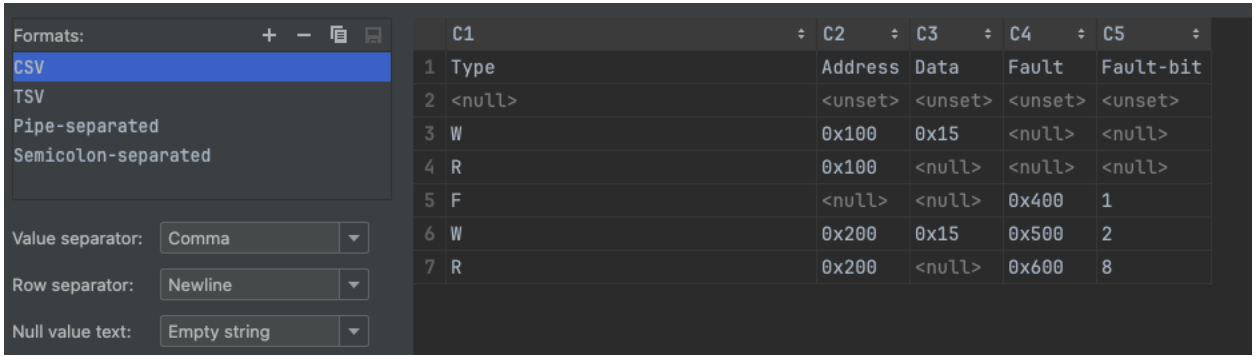
- Valid lines → converted into Request structures
- Invalid lines → rejected with an error message

Valid input test

➢ Test Input File: valid_input.csv

```
Type,Address,Data,Fault,Fault-bit

W,0x100,0x15,,
R,0x100,,,
F,,,0x400,1
W,0x200,0x15,0x500,2
R,0x200,,0x600,8
```



	C1	C2	C3	C4	C5
1	Type	Address	Data	Fault	Fault-bit
2	<null>	<unset>	<unset>	<unset>	<unset>
3	W	0x100	0x15	<null>	<null>
4	R	0x100	<null>	<null>	<null>
5	F	<null>	<null>	0x400	1
6	W	0x200	0x15	0x500	2
7	R	0x200	<null>	0x600	8

➢ Console Output (Parsed Result)

```
(base) jiyuze@jiyuzedeMacBook-Pro eingabe % ./module valid_input.csv
Parsed 5 requests from input file:
Request 0: r=0 w=1 addr=0x00000100 data=0x00000015 fault=4294967295 faultBit=0
Request 1: r=1 w=0 addr=0x00000100 data=0x00000000 fault=4294967295 faultBit=0
Request 2: r=0 w=0 addr=0x00000000 data=0x00000000 fault=1024 faultBit=1
Request 3: r=0 w=1 addr=0x00000200 data=0x00000015 fault=1280 faultBit=2
Request 4: r=1 w=0 addr=0x00000200 data=0x00000000 fault=1536 faultBit=8
```

Invalid input test

➢ Test Input File: Invalid_input.csv

- The parser supports custom test cases with targeted invalid lines
- A dedicated test_case section includes common input errors:
  1、Invalid faultBit values
  2、Illegal characters in numeric fields
  3、Unknown operation types
- Overflowing or negative addresses
- The CSV parser follows a **fail-fast design**:
  → It aborts on the **first detected error**, avoiding simulation of invalid data

```
Type,Address,Data,Fault,Fault-bit

W,0x100,0x15,,
R,0x100,,,
F,,,0x400,1

F,,,abcd,9

W,0x200,0x15,0x500,2
R,0x200,,0x600,3



------------------------
test_case
Q,0x100,0x15,,
W,hello,0x15,,
R,-123,,,
W,0x100000000,0x10,,
W,0x100,xyz,,
W,0x200,-1,,
F,,,0x400000000,2
F,,,abcd,9
W,0x200,0x15,0x500,9
1287%¥6jhdfg
R,0x100,,,,
```

➢ Console Output (Parsed Result)

```
(base) jiyuze@jiyuzedeMacBook-Pro eingabe % ./module Invalid_input.csv
Invalid fault in fault request: abcd
Failed to parse requests from input file.
```

Read / Write Roundtrip

- Overview

```
Request reqs[20];
uint32_t test_data[10] = {
        0x00000000,     // Minimum
        0xFFFFFFFF,     // Maximum
        0x00000001,     // Single bit set low
        0x80000000,     // Single bit set high
        0xAAAAAAAA,     // Alternating bits 1010...
        0x55555555,     // Alternating bits 0101...
        0xFF000000,     // Only high byte
        0x000000FF,     // Only low byte
        0xCAFEBABE,     // Equals to key (example)
        0xA0            // Equals to address
};
uint32_t test_addr[10] = {
        0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80, 0x90, 0xA0
};
```

```
for (int i = 0; i < 10; ++i) {
    // Write request
    reqs[2*i].addr = test_addr[i];
    reqs[2*i].data = test_data[i];
    reqs[2*i].r = 0;
    reqs[2*i].w = 1;
    reqs[2*i].fault = UINT32_MAX;
    reqs[2*i].faultBit = 0;

    // Read request
    reqs[2*i+1].addr = test_addr[i];
    reqs[2*i+1].data = 0;
    reqs[2*i+1].r = 1;
    reqs[2*i+1].w = 0;
    reqs[2*i+1].fault = UINT32_MAX;
    reqs[2*i+1].faultBit = 0;
}
```

```
std::cout << "Testbench Simulation completed.\n";
std::cout << "Cycles: " << result.cycles << std::endl;
std::cout << "Errors: " << result.errors << std::endl;
```

Fault Injection & Parity Check

- Overview

```
int main() {
    Request reqs[13];

    reqs[0].addr = 0x20;
    reqs[0].data = 0xDEADBEEF;
    reqs[0].r = 0;
    reqs[0].w = 1;
    reqs[0].fault = UINT32_MAX;
    reqs[0].faultBit = 0;

    reqs[1].addr = 0x20;
    reqs[1].data = 0;
    reqs[1].r = 1;
    reqs[1].w = 0;
    reqs[1].fault = UINT32_MAX;
    reqs[1].faultBit = 0;

    reqs[2].addr = 0;
    reqs[2].data = 0;
    reqs[2].r = 0;
    reqs[2].w = 0;
    reqs[2].fault = 0x5DE4A452;
    reqs[2].faultBit = 2;
```

```
reqs[3].addr = 0x20;
reqs[3].data = 0;
reqs[3].r = 1;
reqs[3].w = 0;
reqs[3].fault = UINT32_MAX;
reqs[3].faultBit = 0;

reqs[4].addr = 0x30;
reqs[4].data = 0x12345678;
reqs[4].r = 0;
reqs[4].w = 1;
reqs[4].fault = UINT32_MAX;
reqs[4].faultBit = 3;

reqs[5].addr = 0x100;
reqs[5].data = 0x56781234;
reqs[5].r = 0;
reqs[5].w = 1;
reqs[5].fault = 0x5DE4A441;
reqs[5].faultBit = 3;
```

```
reqs[6].addr = 0x30;
reqs[6].data = 0;
reqs[6].r = 1;
reqs[6].w = 0;
reqs[6].fault = UINT32_MAX;
reqs[6].faultBit = 0;

reqs[7].addr = 0x40;
reqs[7].data = 0x23456789;
reqs[7].r = 0;
reqs[7].w = 1;
reqs[7].fault = UINT32_MAX;
reqs[7].faultBit = 0;

reqs[8].addr = 0x40;
reqs[8].data = 0;
reqs[8].r = 1;
reqs[8].w = 0;
reqs[8].fault = 0x5DE4A432;
reqs[8].faultBit = 2;
```

```
reqs[9].addr = 0x50;
reqs[9].data = 0x34567890;
reqs[9].r = 0;
reqs[9].w = 1;
reqs[9].fault = UINT32_MAX;
reqs[9].faultBit = 0;

reqs[10].addr = 0x50;
reqs[10].data = 0;
reqs[10].r = 1;
reqs[10].w = 0;
reqs[10].fault = UINT32_MAX;
reqs[10].faultBit = 0;

reqs[11].addr = 0;
reqs[11].data = 0;
reqs[11].r = 0;
reqs[11].w = 0;
reqs[11].fault = 0x5DE4A423;
reqs[11].faultBit = 8;
```

CLI Parameter Validation

- Overview
- Setup Summary

Element	Description
Request Type	Single Write Request (W)
Fault Config	Set to UINT32_MAX to avoid masking error
Latency Input	0xFFFFFFFF → clearly invalid

```
void test_invalid_latency() {
    Request reqs[1];
    reqs[0].addr = 0x20;
    reqs[0].data = 0xDEADBEEF;
    reqs[0].r = 0;
    reqs[0].w = 1;
    reqs[0].fault = UINT32_MAX;
    reqs[0].faultBit = 0;

    uint32_t invalid_latency = 0xFFFFFFFF;

    std::cout << "\n--- Test with extremely large latency parameter ---" << std::endl;
    Result result = run_simulation(
            10000,            // max_cycles
            "invalid_latency_trace", // tracefile
            0,                // endianness
            invalid_latency,  // latency_scrambling (invalid)
            1,                // latency_encryption
            1,                // latency_memory_access
            1,                // seed
            1,                // numRequests
            reqs
    );
    std::cout << "Cycles: " << result.cycles << std::endl;
    std::cout << "Errors: " << result.errors << std::endl;
}



int main() {
    test_invalid_latency();
    return 0;
}
```

Python Test Driver for C++ Modules

```python
def build_project():
    print("Building with make...")
    result = subprocess.run(["make"], cwd=SCRIPT_DIR, capture_output=True, text=True)
    if result.returncode != 0:
        print("Build failed:\n", result.stderr)
        return False
    print("Build succeeded")
    return True
```

```python
def run_tests():
    for exe in TEST_EXECUTABLES:
        exe_path = os.path.join(SCRIPT_DIR, exe)
        if not os.path.isfile(exe_path):
            print(f"Skipping {exe} — not found.")
            continue

        print(f"\nRunning test: {exe}")
        result = subprocess.run([exe_path], capture_output=True, text=True)
        print("Output:")
        print(result.stdout.strip())
        if result.stderr:
            print("Error:\n", result.stderr.strip())
```

Chair of High-Power Converter Systems
TUM School of Engineering and Design
Technical University of Munich

Thank you for your attention