

# The Joy of Hardware Implementation Using ChatGPT

Simon Mussbacher

31.10.2024

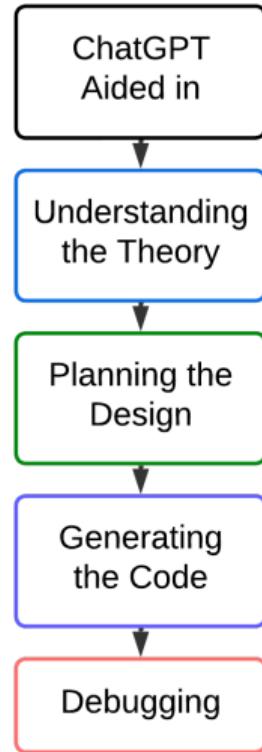
- 1 Background
- 2 What is ChatGPT?
- 3 AES Theory and Implementation
- 4 Evaluation of ChatGPT's Performance

# Why we need Hardware Implementations

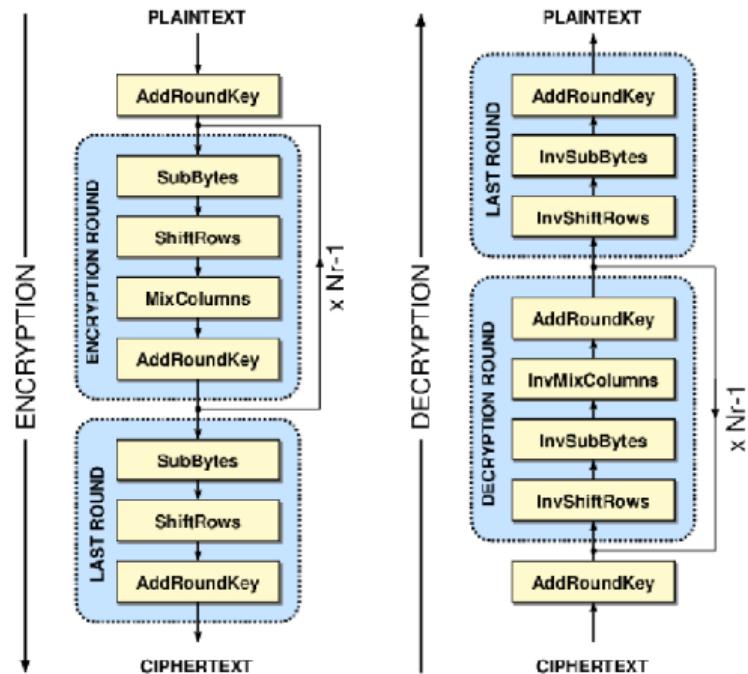
- Hardware solutions (FPGAs, ASICs) provide faster, more secure encryption.
- Require experienced Hardware-Engineers, special tools, a lot of time
- **Benefits:** Real-time processing, energy efficiency, and enhanced security.
- **Drawbacks:** adaptability, cost, steep learning curve

# What is ChatGPT?

**ChatGPT** is an AI language model that generates human-like text for tasks like answering questions and writing code.



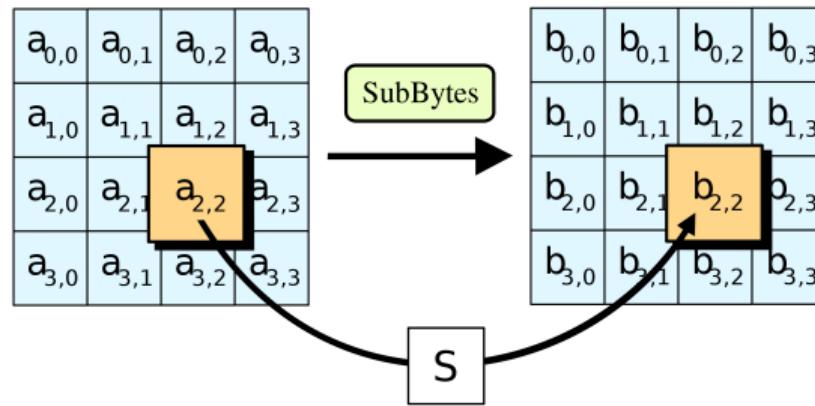
# AES Encryption and Decryption



source: [https://www.researchgate.net/figure/The-basic-AES-128-cryptographic-architecture\\_fig1\\_230853805](https://www.researchgate.net/figure/The-basic-AES-128-cryptographic-architecture_fig1_230853805)

# SubBytes

Non-linear byte substitution using the **S-box** for security through confusion.



[https://en.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard/media/File:AES-SubBytes.svg](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard/media/File:AES-SubBytes.svg)

# SubBytes

⌚:Can you provide me with an implementation for the SubBytes.v module for AES-128, I already have a working SBox implementation

```
// Module to perform the SubBytes operation on a 128-bit input.
module SUB_BYTES(
    input clk,
    input [127:0] IN_DATA,
    output [127:0] SB_DATA_OUT
);

    reg [127:0] SB_DATA_REG;
    wire [127:0] SB_DATA_WORD;

    // Apply SBox transformation on each byte of the input data.
    SBOX SBOX_INST0(.clk(clk), .IN_DATA(IN_DATA[127:120]), .OUT_DATA(SB_DATA_WORD[127:120]));
    SBOX SBOX_INST1(.clk(clk), .IN_DATA(IN_DATA[119:112]), .OUT_DATA(SB_DATA_WORD[119:112]));
    SBOX SBOX_INST2(.clk(clk), .IN_DATA(IN_DATA[111:104]), .OUT_DATA(SB_DATA_WORD[111:104]));
    SBOX SBOX_INST3(.clk(clk), .IN_DATA(IN_DATA[103:96]), .OUT_DATA(SB_DATA_WORD[103:96]));

    SBOX SBOX_INST4(.clk(clk), .IN_DATA(IN_DATA[95:88]), .OUT_DATA(SB_DATA_WORD[95:88]));
    SBOX SBOX_INST5(.clk(clk), .IN_DATA(IN_DATA[87:80]), .OUT_DATA(SB_DATA_WORD[87:80]));
    SBOX SBOX_INST6(.clk(clk), .IN_DATA(IN_DATA[79:72]), .OUT_DATA(SB_DATA_WORD[79:72]));
    SBOX SBOX_INST7(.clk(clk), .IN_DATA(IN_DATA[71:64]), .OUT_DATA(SB_DATA_WORD[71:64]));

    SBOX SBOX_INST8(.clk(clk), .IN_DATA(IN_DATA[63:56]), .OUT_DATA(SB_DATA_WORD[63:56]));
    SBOX SBOX_INST9(.clk(clk), .IN_DATA(IN_DATA[55:48]), .OUT_DATA(SB_DATA_WORD[55:48]));
    SBOX SBOX_INST10(.clk(clk), .IN_DATA(IN_DATA[47:40]), .OUT_DATA(SB_DATA_WORD[47:40]));
    SBOX SBOX_INST11(.clk(clk), .IN_DATA(IN_DATA[39:32]), .OUT_DATA(SB_DATA_WORD[39:32]));

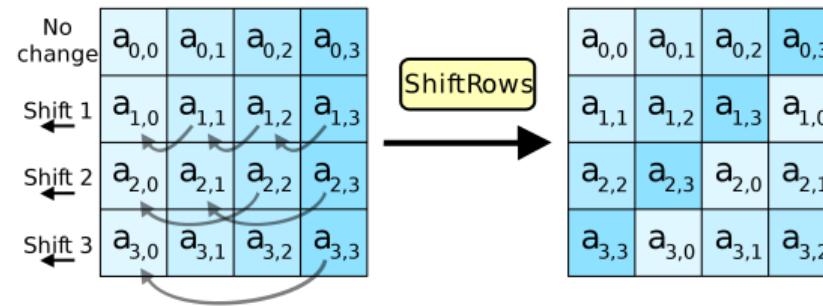
    SBOX SBOX_INST12(.clk(clk), .IN_DATA(IN_DATA[31:24]), .OUT_DATA(SB_DATA_WORD[31:24]));
    SBOX SBOX_INST13(.clk(clk), .IN_DATA(IN_DATA[23:16]), .OUT_DATA(SB_DATA_WORD[23:16]));
    SBOX SBOX_INST14(.clk(clk), .IN_DATA(IN_DATA[15:8]), .OUT_DATA(SB_DATA_WORD[15:8]));
    SBOX SBOX_INST15(.clk(clk), .IN_DATA(IN_DATA[7:0]), .OUT_DATA(SB_DATA_WORD[7:0]));

    // Register the transformed output for stable output data.
    always @(*) begin
        SB_DATA_REG <= SB_DATA_WORD;
    end

    // Assign registered output.
    assign SB_DATA_OUT = SB_DATA_REG;
endmodule
```

# ShiftRows

Circularly shifts rows in the state array.



[https://en.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard/media/File:AES-ShiftRows.svg](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard/media/File:AES-ShiftRows.svg)

# ShiftRows

🔗 Provide me with an implementation for the ShiftRows module for AES-128. It is a circular shift to the left, where the first row stays the same, the second row is shifted by one, the second by two and the third one by three bytes to left

```
// ShiftRows transformation module: reorders rows of the input state matrix.
module SHIFT_ROWS(
    input clk,
    input [127:0] IN_DATA,
    output [127:0] SHIFT_DATA
);

reg [127:0] SHIFT_DATA_REG = 128'b0;

always @(*) begin
    SHIFT_DATA_REG[127:120] = IN_DATA[127:120];
    SHIFT_DATA_REG[119:112] = IN_DATA[87:80];
    SHIFT_DATA_REG[111:104] = IN_DATA[47:40];
    SHIFT_DATA_REG[103:96] = IN_DATA[7:0];

    SHIFT_DATA_REG[95:88] = IN_DATA[95:88];
    SHIFT_DATA_REG[87:80] = IN_DATA[55:48];
    SHIFT_DATA_REG[79:72] = IN_DATA[15:8];
    SHIFT_DATA_REG[71:64] = IN_DATA[103:96];

```

```

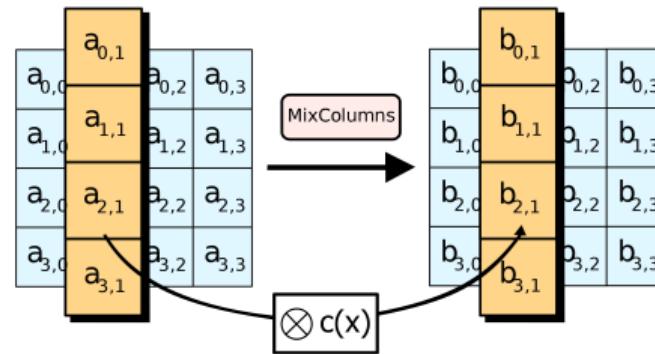
    SHIFT_DATA_REG[63:56] = IN_DATA[63:56];
    SHIFT_DATA_REG[55:48] = IN_DATA[23:16];
    SHIFT_DATA_REG[47:40] = IN_DATA[111:104];
    SHIFT_DATA_REG[39:32] = IN_DATA[71:64];

    SHIFT_DATA_REG[31:24] = IN_DATA[31:24];
    SHIFT_DATA_REG[23:16] = IN_DATA[119:112];
    SHIFT_DATA_REG[15:8] = IN_DATA[79:72];
    SHIFT_DATA_REG[7:0] = IN_DATA[39:32];
end

assign SHIFT_DATA = SHIFT_DATA_REG;
endmodule
```

# MixColumns

Linear mixing of each column using a fixed matrix over GF(2<sup>8</sup>).



[https://en.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard/media/File:AES-MixColumns.svg](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard/media/File:AES-MixColumns.svg)

# MixColumns

⌚ Can you provide me with an implementation for the MixColumns module for AES-128?

```

module MIX_COLUMNS(input clk, input [127:0] IN_DATA, output [127:0] MIXED_DATA;
  reg [127:0] MIXED_DATA_REG;
  always @(*) begin
    MIXED_DATA_REG[127:120] = MIX_COLUMN(IN_DATA[127:120],IN_DATA[119:112],IN_DATA[111:104],IN_DATA[103:96]);
    MIXED_DATA_REG[119:112] = MIX_COLUMN(IN_DATA[119:112],IN_DATA[111:104],IN_DATA[103:96],IN_DATA[127:120]);
    MIXED_DATA_REG[111:104] = MIX_COLUMN(IN_DATA[111:104],IN_DATA[103:96],IN_DATA[127:120],IN_DATA[119:112]);
    MIXED_DATA_REG[103:96] = MIX_COLUMN(IN_DATA[103:96],IN_DATA[127:120],IN_DATA[119:112],IN_DATA[111:104]);
    MIXED_DATA_REG[95:88] = MIX_COLUMN(IN_DATA[95:88],IN_DATA[87:80],IN_DATA[79:72],IN_DATA[71:64]);
    MIXED_DATA_REG[87:80] = MIX_COLUMN(IN_DATA[87:80],IN_DATA[79:72],IN_DATA[71:64],IN_DATA[95:88]);
    MIXED_DATA_REG[79:72] = MIX_COLUMN(IN_DATA[79:72],IN_DATA[71:64],IN_DATA[95:88],IN_DATA[87:80]);
    MIXED_DATA_REG[71:64] = MIX_COLUMN(IN_DATA[71:64],IN_DATA[95:88],IN_DATA[87:80],IN_DATA[79:72]);
    MIXED_DATA_REG[63:56] = MIX_COLUMN(IN_DATA[63:56],IN_DATA[55:48],IN_DATA[47:40],IN_DATA[39:32]);
    MIXED_DATA_REG[55:48] = MIX_COLUMN(IN_DATA[55:48],IN_DATA[47:40],IN_DATA[39:32],IN_DATA[63:56]);
    MIXED_DATA_REG[47:40] = MIX_COLUMN(IN_DATA[47:40],IN_DATA[39:32],IN_DATA[63:56],IN_DATA[55:48]);
    MIXED_DATA_REG[39:32] = MIX_COLUMN(IN_DATA[39:32],IN_DATA[63:56],IN_DATA[55:48],IN_DATA[47:40]);
    MIXED_DATA_REG[31:24] = MIX_COLUMN(IN_DATA[31:24],IN_DATA[23:16],IN_DATA[15:8],IN_DATA[7:0]);
    MIXED_DATA_REG[23:16] = MIX_COLUMN(IN_DATA[23:16],IN_DATA[15:8],IN_DATA[7:0],IN_DATA[31:24]);
    MIXED_DATA_REG[15:8] = MIX_COLUMN(IN_DATA[15:8],IN_DATA[7:0],IN_DATA[31:24],IN_DATA[23:16]);
    MIXED_DATA_REG[7:0] = MIX_COLUMN(IN_DATA[7:0],IN_DATA[31:24],IN_DATA[23:16],IN_DATA[15:8]);
  end
  assign MIXED_DATA = MIXED_DATA_REG;
  // Function to apply the Galois Field operations for Mixcolumns.
  function [7:0] MIX_COLUMN();
    input [7:0] IN1, IN2, IN3, IN4;
    begin
      MIX_COLUMN[7] = IN1[6]^IN2[6]^IN2[7]^IN3[7]^IN4[7];
      MIX_COLUMN[6] = IN1[5]^IN2[5]^IN2[6]^IN3[6]^IN4[6];
      MIX_COLUMN[5] = IN1[4]^IN2[4]^IN2[5]^IN3[5]^IN4[5];
      MIX_COLUMN[4] = IN1[3]^IN1[7]^IN2[3]^IN2[4]^IN2[7];
      MIX_COLUMN[3] = IN1[2]^IN1[7]^IN2[2]^IN2[3]^IN2[7];
      MIX_COLUMN[2] = IN1[1]^IN2[1]^IN2[1]^IN3[2]^IN4[2];
      MIX_COLUMN[1] = IN1[0]^IN1[7]^IN2[0]^IN2[1]^IN2[7];
      MIX_COLUMN[0] = IN1[7]^IN2[7]^IN2[0]^IN3[0]^IN4[0];
    end
  endfunction
endmodule

```

⌚ Use the Galois Field as explained before for the MixColumns module

```

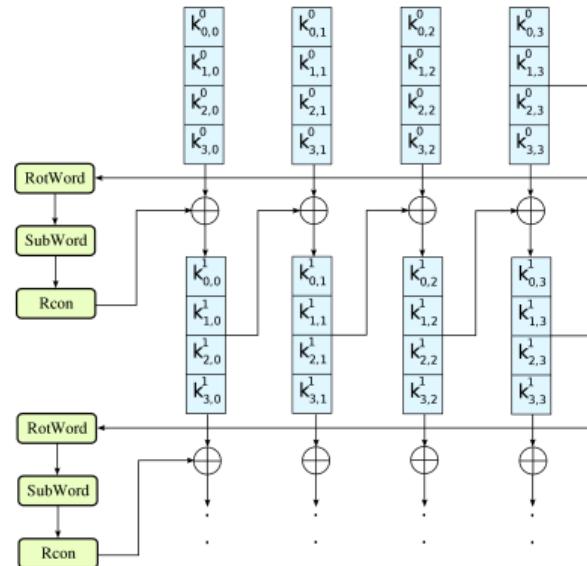
// Galois field multiplication by 2
function [7:0] gm2;
  input [7:0] b;
  begin
    gm2 = (b << 1);
    if (b[7]) gm2 = gm2 ^ 8'h1b; // If the most significant bit was set, reduce by 0x1b.
  end
endfunction
// Galois field multiplication by 3
function [7:0] gm3;
  input [7:0] b;
  begin
    gm3 = gm2(b) ^ b; // Multiplication by 3 is gm2(b) XOR b
  end
endfunction

always @ (posedge clk) begin
  // First column
  MIXED_DATA[127:120] <= gm2(IN_DATA[127:120]) ^ gm3(IN_DATA[119:112]) ^ IN_DATA[111:104] ^ IN_DATA[103:96];
  MIXED_DATA[119:112] <= IN_DATA[127:120] ^ gm2(IN_DATA[119:112]) ^ gm3(IN_DATA[111:104]) ^ IN_DATA[103:96];
  MIXED_DATA[111:104] <= IN_DATA[127:120] ^ IN_DATA[119:112] ^ gm2(IN_DATA[111:104]) ^ gm3(IN_DATA[103:96]);
  MIXED_DATA[103:96] <= gm3(IN_DATA[127:120]) ^ IN_DATA[119:112] ^ IN_DATA[111:104] ^ gm2(IN_DATA[103:96]);
  // Second column
  MIXED_DATA[95:88] <= gm2(IN_DATA[95:88]) ^ gm3(IN_DATA[87:80]) ^ IN_DATA[79:72] ^ IN_DATA[71:64];
  MIXED_DATA[87:80] <= IN_DATA[95:88] ^ gm2(IN_DATA[87:80]) ^ gm3(IN_DATA[79:72]) ^ IN_DATA[71:64];
  MIXED_DATA[79:72] <= IN_DATA[95:88] ^ IN_DATA[87:80] ^ gm2(IN_DATA[79:72]) ^ gm3(IN_DATA[71:64]);
  MIXED_DATA[71:64] <= gm3(IN_DATA[95:88]) ^ IN_DATA[87:80] ^ IN_DATA[79:72] ^ gm2(IN_DATA[71:64]);
  // Third column
  MIXED_DATA[63:56] <= gm2(IN_DATA[63:56]) ^ gm3(IN_DATA[55:48]) ^ IN_DATA[47:40] ^ IN_DATA[39:32];
  MIXED_DATA[55:48] <= IN_DATA[63:56] ^ gm2(IN_DATA[55:48]) ^ gm3(IN_DATA[47:40]) ^ IN_DATA[39:32];
  MIXED_DATA[47:40] <= IN_DATA[63:56] ^ IN_DATA[55:48] ^ gm2(IN_DATA[47:40]) ^ gm3(IN_DATA[39:32]);
  MIXED_DATA[39:32] <= gm3(IN_DATA[63:56]) ^ IN_DATA[55:48] ^ IN_DATA[47:40] ^ gm2(IN_DATA[39:32]);
  // Fourth column
  MIXED_DATA[31:24] <= gm2(IN_DATA[31:24]) ^ gm3(IN_DATA[23:16]) ^ IN_DATA[15:8] ^ IN_DATA[7:0];
  MIXED_DATA[23:16] <= IN_DATA[31:24] ^ gm2(IN_DATA[23:16]) ^ gm3(IN_DATA[15:8]) ^ IN_DATA[7:0];
  MIXED_DATA[15:8] <= IN_DATA[31:24] ^ IN_DATA[23:16] ^ gm2(IN_DATA[15:8]) ^ gm3(IN_DATA[7:0]);
  MIXED_DATA[7:0] <= gm3(IN_DATA[31:24]) ^ IN_DATA[23:16] ^ IN_DATA[15:8] ^ gm2(IN_DATA[7:0]);
end

```

# KeyGeneration

Expands the encryption key into ten separate round keys.



[https://en.wikipedia.org/wiki/AES\\_key\\_schedule/media/File:AES-Key\\_Schedule\\_128-bit\\_key.svg](https://en.wikipedia.org/wiki/AES_key_schedule/media/File:AES-Key_Schedule_128-bit_key.svg)

# KeyGeneration

⌚ Can you provide me with an implementation for the KeyGeneration module for AES-128?

```
module KeyGeneration (
    input clk,
    input [127:0] key,           // Input 128-bit key
    output reg [1407:0] round_keys // Output 11 round keys (128 bits each)
);

    reg [31:0] rcon [0:10];      // RCON values for key expansion
    reg [127:0] keys [0:10];     // Array to store all round keys
    integer i;

    // S-Box Lookup Table (you can reuse the S-Box module or create your own function here)
    function [7:0] SubWord;
        input [7:0] byte;
        // Example S-box substitution using a case statement (partial S-box shown)
        case (byte)
            8'h00: SubWord = 8'h63;
            8'h01: SubWord = 8'h7c;
            8'h02: SubWord = 8'h77;
            // Continue with all S-Box values
            default: SubWord = 8'h00;
        endcase
    endfunction

```

I already had an SBox implementation which was present in the Chat

⌚ Use the SBox implementation I provided to you and the previous information for a new KeyGeneration

```
module KeyExpansion (input wire [127:0] key,          // 128-bit input key
                     output reg [1407:0] w           // Expanded key: 44 words (4 bytes each)
);
    reg [31:0] temp;           // Temporary register to hold intermediate values
    wire [31:0] subword_out;   // Wire for output from the SBox
    reg [31:0] word_in;        // Register to store the word input to SBox
    integer i;
    // Instantiate your SBox module
    SBox sbox_inst (
        .in(word_in),           // 32-bit input to SBox
        .out(subword_out)       // 32-bit output from SBox
    );
    // Rotate word (cyclic permutation of bytes)
    function [31:0] RotWord;
        input [31:0] word;
        begin
            RotWord = {word[23:0], word[31:24]};
        end
    endfunction
    // Key expansion logic
    always @(*) begin
        // First four words are just the original key
        w[127:0] = key;
        // Expand the key
        for (i = 4; i < 44; i = i + 1) begin
            temp = w[32*(i-1) +: 32];
            if (i % 4 == 0) begin
                word_in = RotWord(temp); // Apply RotWord
                temp = subword_out;
            end
            w[32*i +: 32] = w[32*(i-4) +: 32] ^ temp;
        end
    end
endmodule
```

My SBox has 8-bit inputs/output

Missing RCON

# KeyGeneration continued

⌚ The rcon function is missing, apply it, as I explained it to you, and also my SBox uses 8-bit inputs/outputs.

Also I had an issue with the large output reg. Also it should calculate each round key individually to better integrate it in the round iteration and last round

```

reg [31:0] temp;           // Temporary register for key expansion
reg [31:0] rcon_value;    // Round constant
wire [7:0] sbox_out [3:0]; // Output from SBox (8-bit per byte)
reg [7:0] sbox_in [3:0];  // Input to SBox (8-bit per byte)
// Instantiate the 8-bit SBox (handling one byte at a time)
SBox sbox_inst0 (.in(sbox_in[0]), .out(sbox_out[0]));
SBox sbox_inst1 (.in(sbox_in[1]), .out(sbox_out[1]));
SBox sbox_inst2 (.in(sbox_in[2]), .out(sbox_out[2]));
SBox sbox_inst3 (.in(sbox_in[3]), .out(sbox_out[3]));
// Rotate word (cyclic permutation of bytes)
function [31:0] RotWord;
  input [31:0] word;
begin
  | RotWord = {word[23:0], word[31:24]};
end
endfunction
// Round constant (Rcon) function
function [31:0] Rcon;
  input [31:0] round;
  begin
    case (round)
      1: Rcon = 32'h01000000;
      2: Rcon = 32'h02000000;
      3: Rcon = 32'h04000000;
      4: Rcon = 32'h08000000;
      5: Rcon = 32'h10000000;
      6: Rcon = 32'h20000000;
      7: Rcon = 32'h40000000;
      8: Rcon = 32'h80000000;
      9: Rcon = 32'h10000000;
      10: Rcon = 32'h30000000;
      default: Rcon = 32'h00000000;
    endcase
  end
endfunction

```

**Issues:** wrong integration in AES\_128, in/output not working,...

It helped to debug and this solution was the result

```

// Internal signals for the four 32-bit words of the input key.
wire [31:0] KEY0, KEY1, KEY2, KEY3;
wire [31:0] SUBWORD_OUT; // SubWord output after S-Box transformation
assign KEY0 = IN_KEY[127:96];
assign KEY1 = IN_KEY[95:64];
assign KEY2 = IN_KEY[63:32];
assign KEY3 = IN_KEY[31:0];
// Generate the new 128-bit key by XORing the transformed words and round constants.
assign OUT_KEY[127:96] = KEY3 ^ SUBWORD_OUT ^ RCON(ROUND_KEY);
assign OUT_KEY[95:64] = OUT_KEY[127:96] ^ KEY2;
assign OUT_KEY[63:32] = OUT_KEY[95:64] ^ KEY1;
assign OUT_KEY[31:0] = OUT_KEY[63:32] ^ KEY0;
// SBox Instances for SubWord transformation
SBOX_SBOX_INST0(.clk(clk), .IN_DATA(KEY0[23:16]), .OUT_DATA(SUBWORD_OUT[31:24]));
SBOX_SBOX_INST1(.clk(clk), .IN_DATA(KEY0[15:8]), .OUT_DATA(SUBWORD_OUT[23:16]));
SBOX_SBOX_INST2(.clk(clk), .IN_DATA(KEY0[7:0]), .OUT_DATA(SUBWORD_OUT[15:8]));
SBOX_SBOX_INST3(.clk(clk), .IN_DATA(KEY0[31:24]), .OUT_DATA(SUBWORD_OUT[7:0]));
// Store the output key in a register for timing control
always @ (posedge clk)
  OUT_KEY_R <= OUT_KEY;
// Function to provide the round constants (RCON) for each round.
function [31:0] RCON;
  input [31:0] ROUND_KEY;
  begin
    case (ROUND_KEY)
      4'h0: RCON = 32'h01000000;
      4'h1: RCON = 32'h02000000;
      4'h2: RCON = 32'h04000000;
      4'h3: RCON = 32'h08000000;
      4'h4: RCON = 32'h10000000;
      4'h5: RCON = 32'h20000000;
      4'h6: RCON = 32'h40000000;
      4'h7: RCON = 32'h80000000;
      4'h8: RCON = 32'h10000000;
      4'h9: RCON = 32'h30000000;
      default: RCON = 32'h00000000;
    endcase
  end
endfunction

```

# Validating the functionality

```

// Instantiate the AES encryption module
AES_128 AES_128_ENCRYPT(
    .clk(clk),
    .reset(reset),
    .INPUT_DATA(input_data),
    .INPUT_KEY(input_key),
    .OUTPUT_DATA(encrypted_data)
);
// Instantiate the AES decryption module
AES_128_DECRYPT AES_128_DECRYPT(
    .clk(clk),
    .reset(reset),
    .INPUT_DATA(encrypted_data),
    .INPUT_KEY(input_key),
    .OUTPUT_DATA(decrypted_data)
);
initial begin
    clk, reset = 0;
    encryption_cycles = decryption_cycles = 0;
    encryption_done = decryption_done = 0;

    $dumpfile("aes_tb.vcd");
    $dumpvars(0, aes_tb);
    // Test Case 1
    #20
    input_data = 128'h00112233445566778899aabbcdddeeff;
    input_key = 128'h110102030405060708090a0b0c0d0e0f;

```

// Start counting encryption cycles

```

    encryption_cycles, encryption_done = 0;
    while (!encryption_done) begin
        encryption_cycles = encryption_cycles + 1;
        @posedge clk;
        if (encrypted_data == 128'h69c4e0d86a7b0430d8cdb7807b4c55a)
            encryption_done = 1;
    end

    $display("Test Case 1:");
    $display("Input Data: %h", input_data);
    $display("Encrypted Data: %h", encrypted_data);
    $display("Encryption Cycles: %d", encryption_cycles);

    // Start counting decryption cycles
    decryption_cycles, decryption_done = 0;
    while (!decryption_done) begin
        decryption_cycles = decryption_cycles + 1;
        @posedge clk;
        if (decrypted_data == input_data)
            decryption_done = 1;
    end

    $display("Decrypted Data: %h", decrypted_data);
    $display("Decryption Cycles: %d", decryption_cycles);

    if (decrypted_data == input_data)
        $display("Decryption successful!");
    else
        $display("Decryption failed!");

```

// Test Case 2

```

#1000
input_data = 128'h00112233445566778899aabbcdddeeff;
input_key = 128'h110102030405060708090a0b0c0d0e0f;
encryption_cycles, encryption_done = 0;
while (!encryption_done) begin
    encryption_cycles = encryption_cycles + 1;
    @posedge clk;
    if (encrypted_data == 128'haeae180711ab0f6d999bea7418201a81)
        encryption_done = 1;
end

$display("Test Case 2:");
$display("Input Data: %h", input_data);
$display("Encrypted Data: %h", encrypted_data);
$display("Encryption Cycles: %d", encryption_cycles);

decryption_cycles = 0, decryption_done = 0;
while (!decryption_done) begin
    decryption_cycles = decryption_cycles + 1;
    @posedge clk;
    if (decrypted_data == input_data)
        decryption_done = 1;
end

$display("Decrypted Data: %h", decrypted_data);
$display("Decryption Cycles: %d", decryption_cycles);
if (decrypted_data == input_data)
    $display("Decryption successful!");
else
    $display("Decryption failed!");
#100 $finish;
end
always #5 clk = !clk;

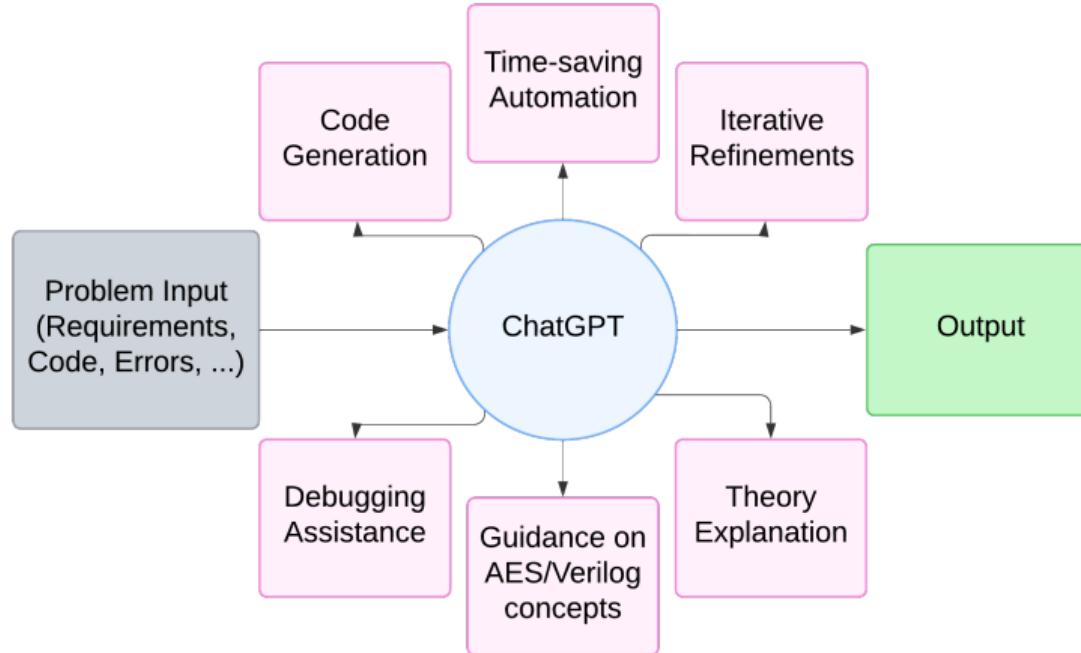
```

Encryption clock cycles: 31

Decryption clock cycles: 30

## Test Setup

- Tested different models: GPT-4, GPT-4o, and Custom GPT-4o.
- Used basic implementations (full adder, AES) for tests
- **Best Performer:** Custom GPT-4o with domain-specific knowledge.
- Utilized ModelSim for Verilog code simulation.



## Challenges and Limitations

- **Complex Design:** Struggled with inter-module connections.
- **Contextual Limitation:** Difficulty integrating multiple files.
- **Knowledge Gaps:** Lacked domain-specific cryptography knowledge.

# Conclusion

- Was it really joyful?

# Conclusion

- **Was it really joyful?** Using ChatGPT for HW implementation was an exciting journey. While not without its challenges, the tool was a great assistant for the whole process.

# Conclusion

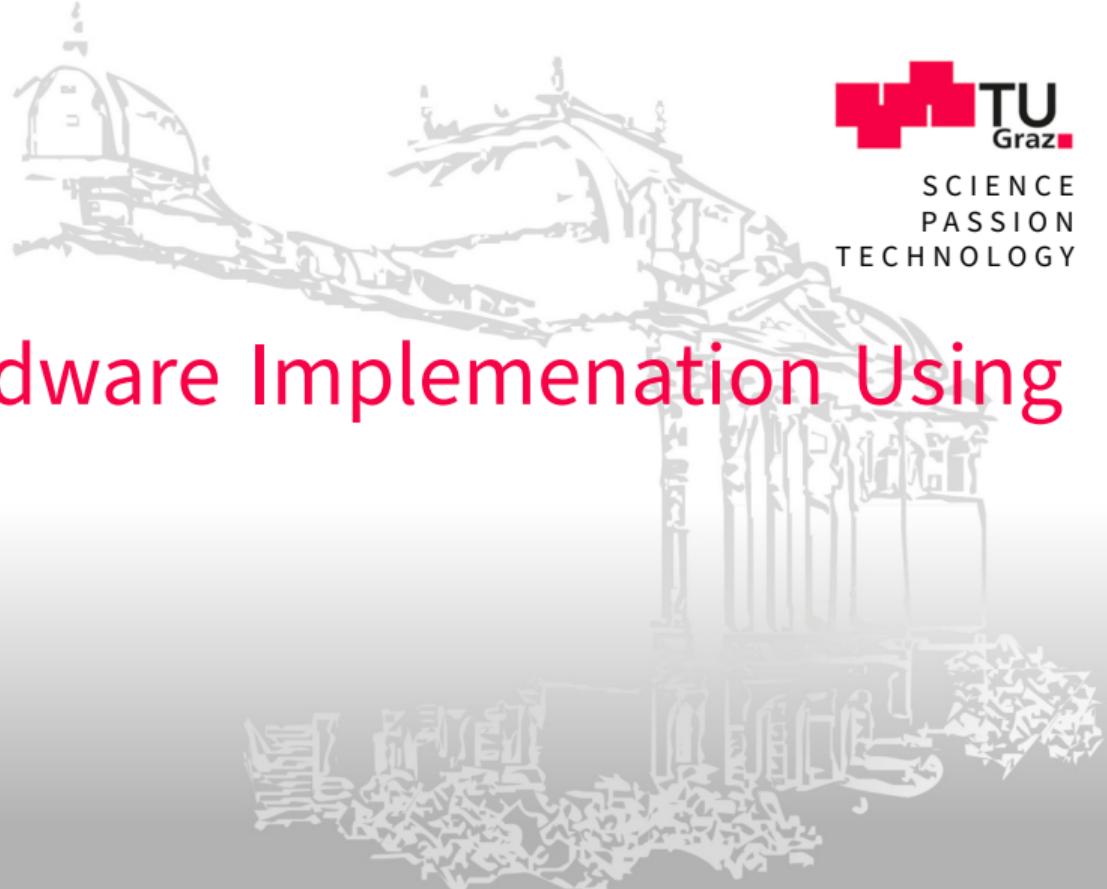
- **Was it really joyful?** Using ChatGPT for HW implementation was an exciting journey. While not without its challenges, the tool was a great assistant for the whole process.
- **Would I use it for different projects?**

# Conclusion

- **Was it really joyful?** Using ChatGPT for HW implementation was an exciting journey. While not without its challenges, the tool was a great assistant for the whole process.
- **Would I use it for different projects?** Absolutely! ChatGPT accelerated the process of understanding the topic, generating code, and simplified debugging.

# Final Thought

*ChatGPT is not perfect yet, but it is a promising tool and can help bridge the expertise gap between Cryptography and Engineering.*



# The Joy of Hardware Implementation Using ChatGPT

Simon Mussbacher

31.10.2024