Quick evaluation of technical skills and research aptitude - Answer

AmirrezaGh

Abstract— **In this report, one method is proposed to each problem described in the Quick evaluation of technical skills and research aptitude documents, minimum hamming distance and GCD VHDL hardware. Also, the methods are evaluated against previous works.**

Keywords— **Hamming distance – GCD hardware**

I. Introduction

As it was mentioned before, in this report, minimum hamming distance is studied, and minimum hammering distance of the elements of an array is calculated in C++ language. Also, the RTL hardware of the Euclidean GCD algorithm is written in VHDL.

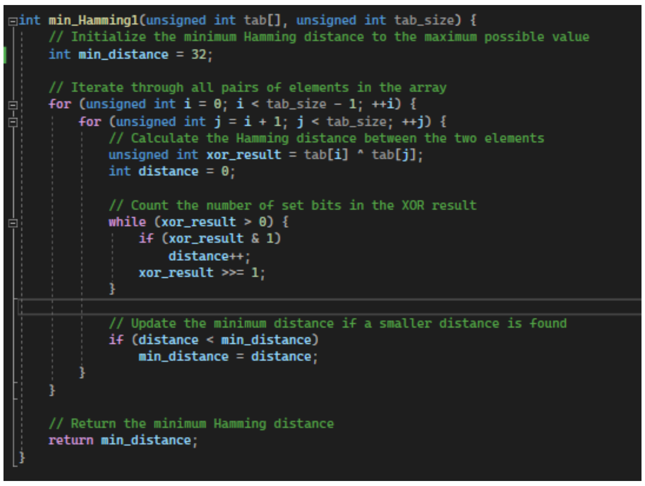
II. Min Hamming Distance

Hamming distance is a concept used in information theory and computer science to measure the difference between two strings of equal length. It quantifies the minimum number of substitutions required to change one string into another by flipping individual bits.

It has various applications in areas such as error detection and correction, data clustering, cryptography, and DNA sequence analysis.

To calculate the Hamming distance, you compare each corresponding pair of bits in the two strings and count the number of positions where they differ. This count represents the Hamming distance between the strings. The strings must be of the same length for a valid comparison.

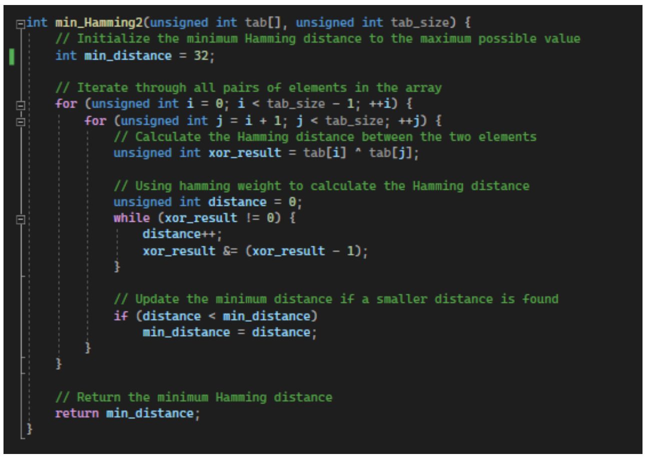
To calculate the minimum hamming distance in an array of unsigned integer values, there are two famous approaches. First use shift operator when the number of ones is calculated.



```cpp
int min_Hamming1(unsigned int tab[], unsigned int tab_size) {
    // Initialize the minimum Hamming distance to the maximum possible value
    int min_distance = 32;

    // Iterate through all pairs of elements in the array
    for (unsigned int i = 0; i < tab_size - 1; ++i) {
        for (unsigned int j = i + 1; j < tab_size; ++j) {
            // Calculate the Hamming distance between the two elements
            unsigned int xor_result = tab[i] ^ tab[j];
            int distance = 0;

            // Count the number of set bits in the XOR result
            while (xor_result > 0) {
                if (xor_result & 1)
                    distance++;
                xor_result >>= 1;
            }

            // Update the minimum distance if a smaller distance is found
            if (distance < min_distance)
                min_distance = distance;
        }
    }

    // Return the minimum Hamming distance
    return min_distance;
}
```

Fig. 1 Min Hamming Method 1

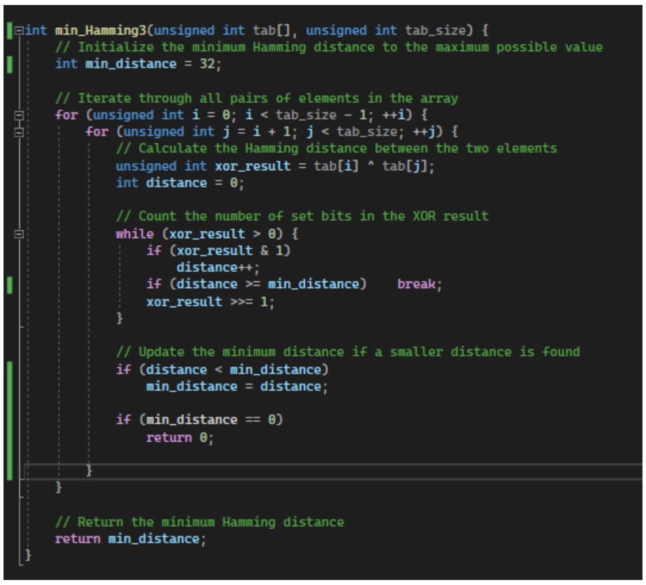
The second approach is to use hamming weight in order to count the number of dissimilarities between two elements.



```cpp
int min_Hamming2(unsigned int tab[], unsigned int tab_size) {
    // Initialize the minimum Hamming distance to the maximum possible value
    int min_distance = 32;

    // Iterate through all pairs of elements in the array
    for (unsigned int i = 0; i < tab_size - 1; ++i) {
        for (unsigned int j = i + 1; j < tab_size; ++j) {
            // Calculate the Hamming distance between the two elements
            unsigned int xor_result = tab[i] ^ tab[j];

            // Using hamming weight to calculate the Hamming distance
            unsigned int distance = 0;
            while (xor_result != 0) {
                distance++;
                xor_result &= (xor_result - 1);
            }

            // Update the minimum distance if a smaller distance is found
            if (distance < min_distance)
                min_distance = distance;
        }
    }

    // Return the minimum Hamming distance
    return min_distance;
}
```

Fig. 2 Min Hamming Method 2

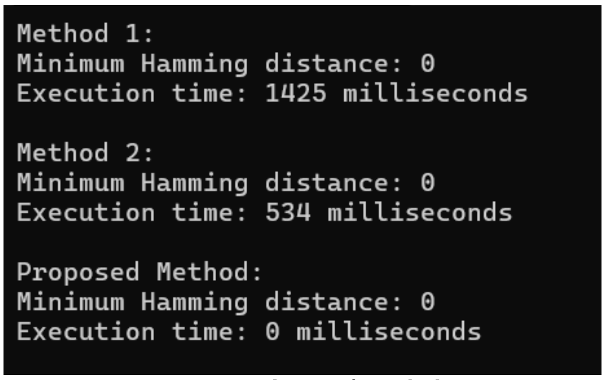
The proposed method is to terminate the dissimilarity count loop early using the temporary minimum distance is less than the computed distance until the current state of calculation of dissimilarities between the two elements. Also, if the minimum distance is zero, it terminates the process and returns zero.



```c
int min_Hamming3(unsigned int tab[], unsigned int tab_size) {
    // Initialize the minimum Hamming distance to the maximum possible value
    int min_distance = 32;

    // Iterate through all pairs of elements in the array
    for (unsigned int i = 0; i < tab_size - 1; ++i) {
        for (unsigned int j = i + 1; j < tab_size; ++j) {
            // Calculate the Hamming distance between the two elements
            unsigned int xor_result = tab[i] ^ tab[j];
            int distance = 0;

            // Count the number of set bits in the XOR result
            while (xor_result > 0) {
                if (xor_result & 1)
                    distance++;
                if (distance >= min_distance)    break;
                xor_result >>= 1;
            }

            // Update the minimum distance if a smaller distance is found
            if (distance < min_distance)
                min_distance = distance;

            if (min_distance == 0)
                return 0;

        }
    }

    // Return the minimum Hamming distance
    return min_distance;
}
```

Fig. 3 Min Hamming Proposed Method

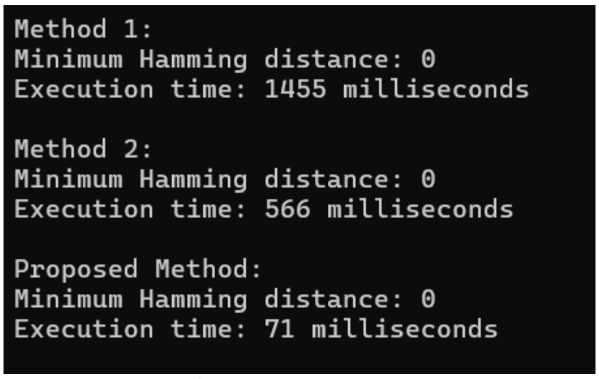
The result obtained from running three methods on a table of 10000 integer numbers are shown below.



```
Method 1:
Minimum Hamming distance: 0
Execution time: 1425 milliseconds

Method 2:
Minimum Hamming distance: 0
Execution time: 534 milliseconds

Proposed Method:
Minimum Hamming distance: 0
Execution time: 0 milliseconds
```

Fig. 4 Evaluation of 3 methods

Because of early termination of the whole function using the condition whether minimum is zero or not, the execution time gets very small. But in order to show the impact of early termination in count loop, the condition gets commented. So, the result is shown below.

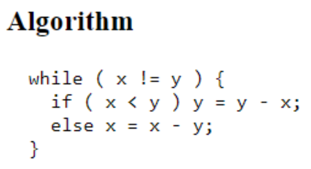


```
Method 1:
Minimum Hamming distance: 0
Execution time: 1455 milliseconds

Method 2:
Minimum Hamming distance: 0
Execution time: 566 milliseconds

Proposed Method:
Minimum Hamming distance: 0
Execution time: 71 milliseconds
```

Fig. 5 Evaluation of 3 methods without the condition on minimum distance

III. GCD VHDL Code

In this report, the Euclidean algorithm is used to compute greatest common divisor (GCD). The algorithm is based on the principle that the GCD of two numbers does not change if the smaller number is subtracted from the larger number repeatedly until the two numbers become equal. At this point, the common value is the GCD of the original two numbers.



Algorithm

```
while ( x != y ) {
  if ( x < y ) y = y - x;
  else x = x - y;
}
```

Fig. 6 Euclidean Algorithm

First of All, the following behavioural finite state machine is implemented in the previous works in order to show the correctness of the algorithm and behaviour of the next implementations.

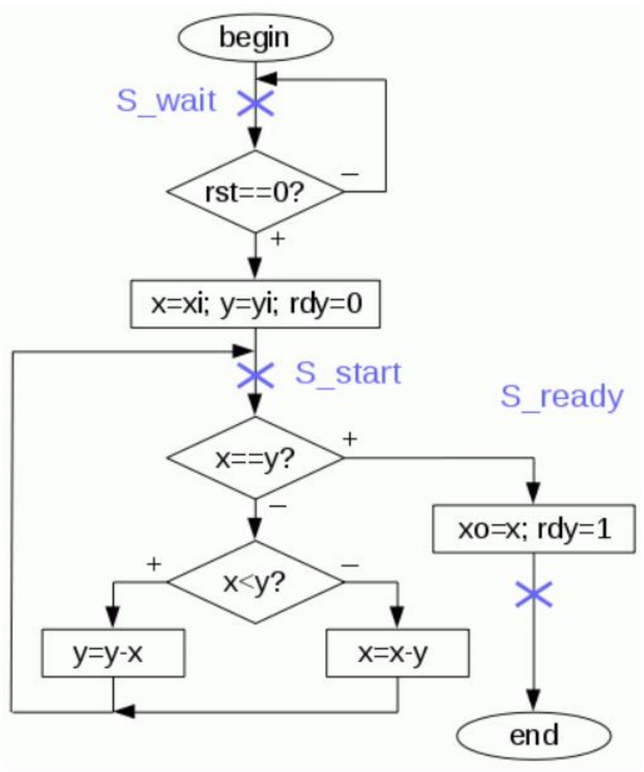
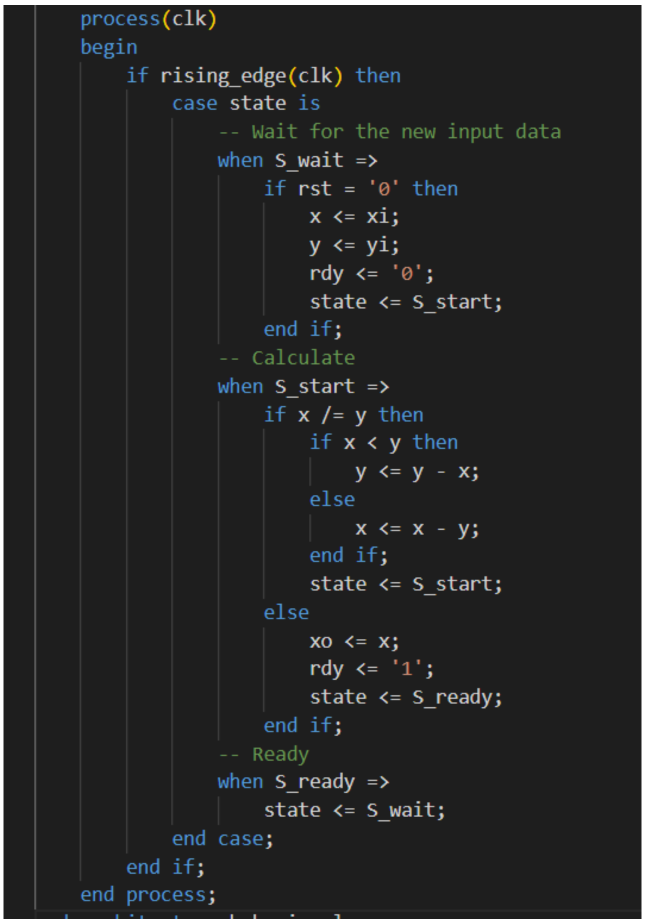


Fig. 7 behavioural FSM



```
process(clk)
begin
    if rising_edge(clk) then
        case state is
            -- Wait for the new input data
            when S_wait =>
                if rst = '0' then
                    x <= xi;
                    y <= yi;
                    rdy <= '0';
                    state <= S_start;
                end if;
            -- Calculate
            when S_start =>
                if x /= y then
                    if x < y then
                        y <= y - x;
                    else
                        x <= x - y;
                    end if;
                    state <= S_start;
                else
                    xo <= x;
                    rdy <= '1';
                    state <= S_ready;
                end if;
            -- Ready
            when S_ready =>
                state <= S_wait;
        end case;
    end if;
end process;
```

Fig. 8 Behavioural FSM code

The first method of RTL implementation of GCD uses operation reuse using only one ALU for all operations, such as subtraction and comparison. The DataPath is shown below.

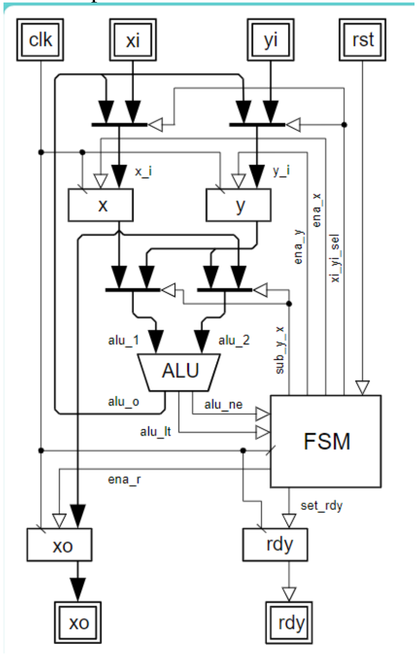


Fig. 9 DataPath of RTL method 1

The most simple FSM for the DataPath is as follows.

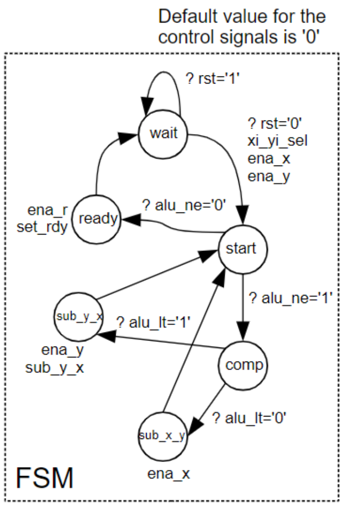


Fig. 10 FSM 1 of RTL method 1

Then the better FSM with less states and clocks is as follows using mealy state machine

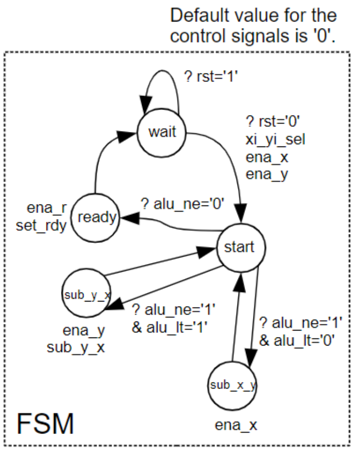


Fig. 11 FSM 2 of RTL method 1

The second method is using a separate comparator besides ALU in order to achieve better speed.

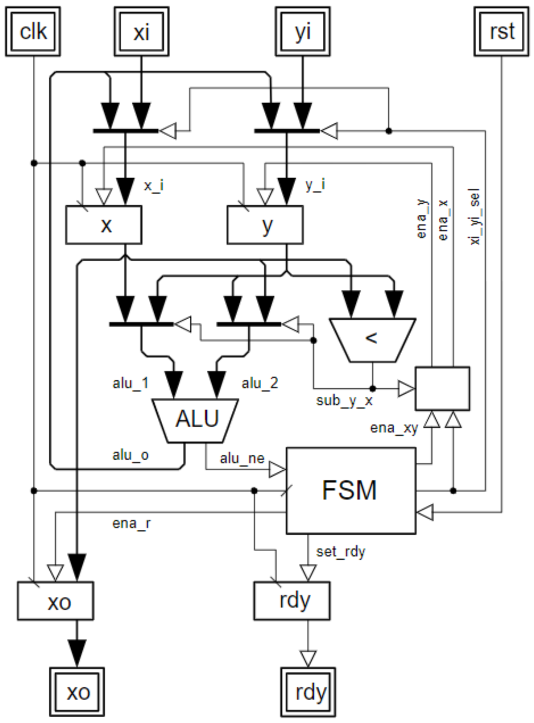


Fig. 12 DataPath of RTL method 2

The third method is using a separate subtractor besides ALU in order to achieve better speed.

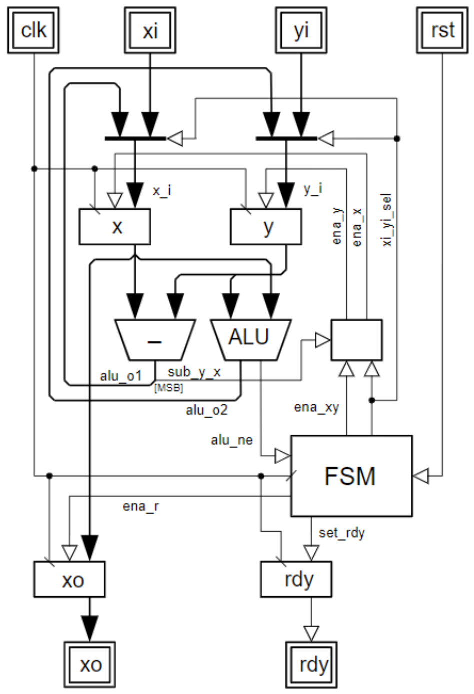


Fig. 13 DataPath of RTL method 3

The forth method in previous works is using two subtractor and a comparator instead of ALU.

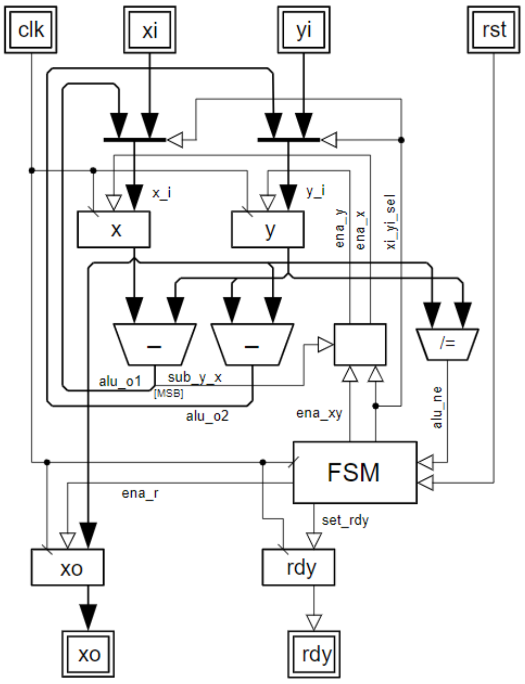


Fig. 14 DataPath of RTL method 4

The proposed method is using less-than, not-equal comparators and two subtractor in order to maximize the speed.

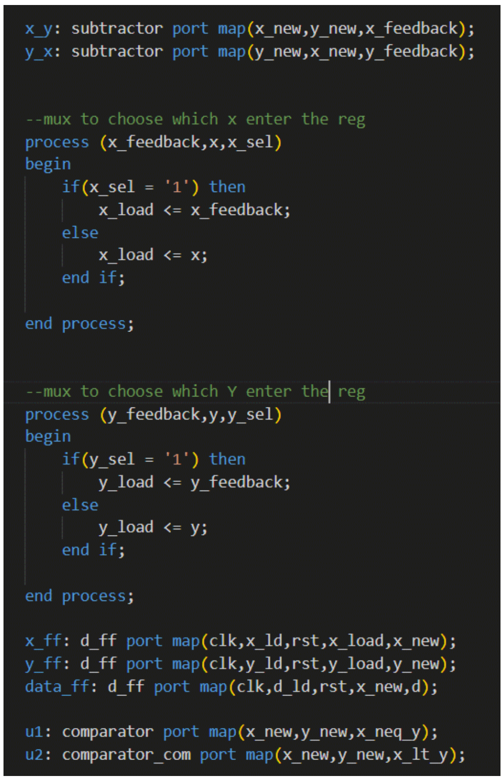
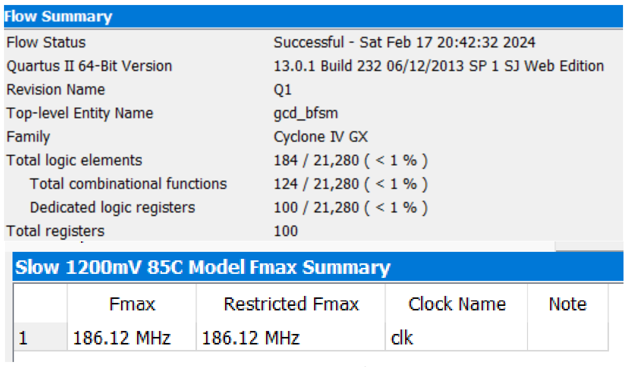

```
x_y: subtractor port map(x_new,y_new,x_feedback);
y_x: subtractor port map(y_new,x_new,y_feedback);


--mux to choose which x enter the reg
process (x_feedback,x,x_sel)
begin
    if(x_sel = '1') then
        x_load <= x_feedback;
    else
        x_load <= x;
    end if;

end process;


--mux to choose which Y enter the reg
process (y_feedback,y,y_sel)
begin
    if(y_sel = '1') then
        y_load <= y_feedback;
    else
        y_load <= y;
    end if;

end process;

x_ff: d_ff port map(clk,x_ld,rst,x_load,x_new);
y_ff: d_ff port map(clk,y_ld,rst,y_load,y_new);
data_ff: d_ff port map(clk,d_ld,rst,x_new,d);

u1: comparator port map(x_new,y_new,x_neq_y);
u2: comparator_com port map(x_new,y_new,x_lt_y);
```

Fig. 15 Proposed Method code

The Evaluation Results of the above methods using Quartus software to implement on Cyclone IV GX are as follows.

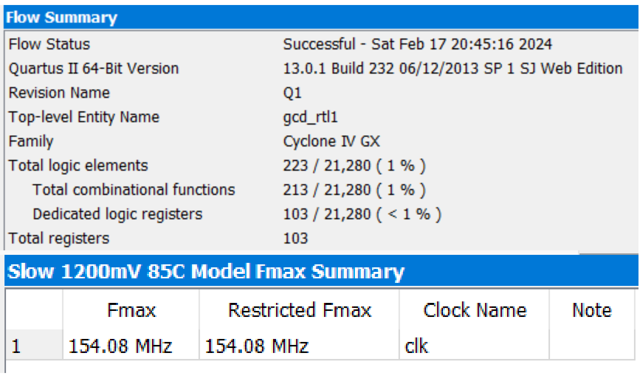


Flow Summary	
Flow Status	Successful - Sat Feb 17 20:42:32 2024
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Q1
Top-level Entity Name	gcd_bfsm
Family	Cyclone IV GX
Total logic elements	184 / 21,280 (< 1 %)
Total combinational functions	124 / 21,280 (< 1 %)
Dedicated logic registers	100 / 21,280 (< 1 %)
Total registers	100

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	186.12 MHz	186.12 MHz	clk	

Fig. 16 Synthesis Result of behavioural FSM

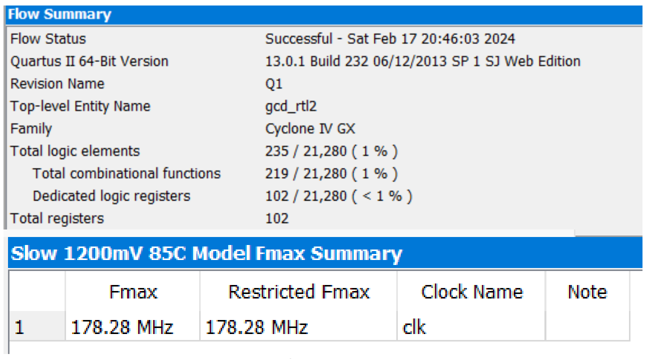


Flow Summary	
Flow Status	Successful - Sat Feb 17 20:45:16 2024
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Q1
Top-level Entity Name	gcd_rtl1
Family	Cyclone IV GX
Total logic elements	223 / 21,280 (1 %)
Total combinational functions	213 / 21,280 (1 %)
Dedicated logic registers	103 / 21,280 (< 1 %)
Total registers	103

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	154.08 MHz	154.08 MHz	clk	

Fig. 17 Synthesis Result of second method with first FSM

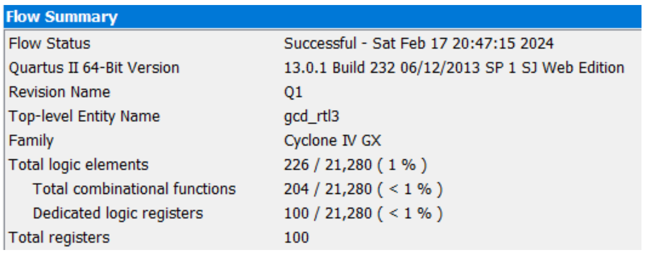


Flow Summary	
Flow Status	Successful - Sat Feb 17 20:46:03 2024
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Q1
Top-level Entity Name	gcd_rtl2
Family	Cyclone IV GX
Total logic elements	235 / 21,280 (1 %)
Total combinational functions	219 / 21,280 (1 %)
Dedicated logic registers	102 / 21,280 (< 1 %)
Total registers	102

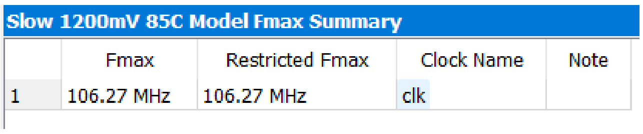
Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	178.28 MHz	178.28 MHz	clk	

Fig. 18 Synthesis Result of second method with the second FSM



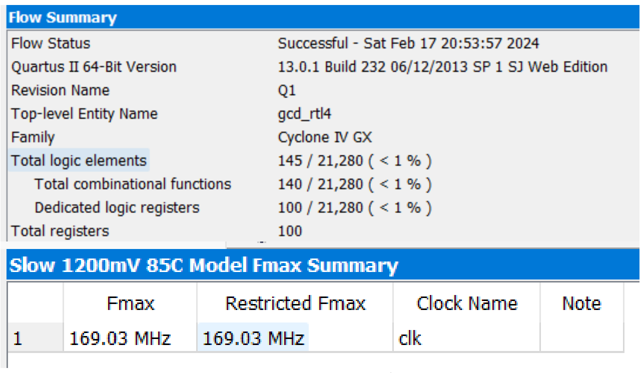
Flow Summary	
Flow Status	Successful - Sat Feb 17 20:47:15 2024
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Q1
Top-level Entity Name	gcd_rtl3
Family	Cyclone IV GX
Total logic elements	226 / 21,280 (1 %)
Total combinational functions	204 / 21,280 (< 1 %)
Dedicated logic registers	100 / 21,280 (< 1 %)
Total registers	100



Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	106.27 MHz	106.27 MHz	clk	

Fig. 19 Synthesis Result of third method with first FSM

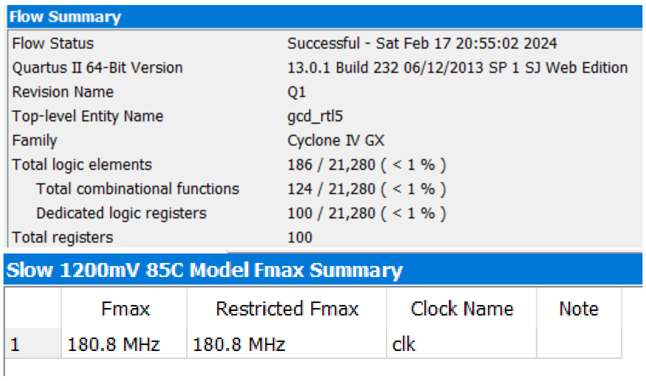


Flow Summary	
Flow Status	Successful - Sat Feb 17 20:53:57 2024
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Q1
Top-level Entity Name	gcd_rtl4
Family	Cyclone IV GX
Total logic elements	145 / 21,280 (< 1 %)
Total combinational functions	140 / 21,280 (< 1 %)
Dedicated logic registers	100 / 21,280 (< 1 %)
Total registers	100

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	169.03 MHz	169.03 MHz	clk	

Fig. 20 Synthesis Result of third method



Flow Summary	
Flow Status	Successful - Sat Feb 17 20:55:02 2024
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Q1
Top-level Entity Name	gcd_rtl5
Family	Cyclone IV GX
Total logic elements	186 / 21,280 (< 1 %)
Total combinational functions	124 / 21,280 (< 1 %)
Dedicated logic registers	100 / 21,280 (< 1 %)
Total registers	100

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	180.8 MHz	180.8 MHz	clk	

Fig. 21 Synthesis Result of forth method

The result of calculation of GCD(14, 161) using the proposed method is as follows.

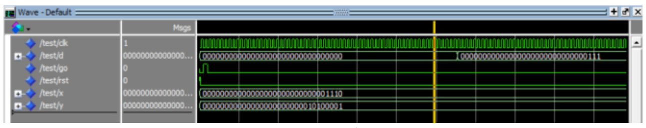
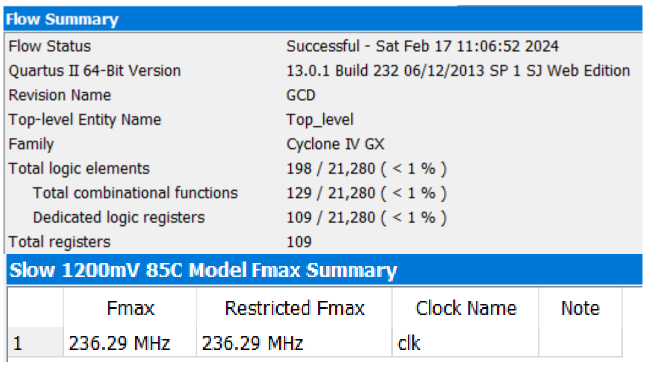


Fig. 22 GCD Result of proposed method



Flow Summary	
Flow Status	Successful - Sat Feb 17 11:06:52 2024
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	GCD
Top-level Entity Name	Top_level
Family	Cyclone IV GX
Total logic elements	198 / 21,280 (< 1 %)
Total combinational functions	129 / 21,280 (< 1 %)
Dedicated logic registers	109 / 21,280 (< 1 %)
Total registers	109

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	236.29 MHz	236.29 MHz	clk	

Fig. 23 Synthesis Result of proposed method

IV. CONCLUSION

To put it in a nutshell, in the first problem, the evaluations are as follows.

Tab. 1 Conclusion of Problem 1

Method	Execution time(ms)
Simple method	1455
Weight hamming	566
Proposed method	71

In the second problem, the following table concludes every method result.

Tab. 2 Conclusion of Problem 1

Method	Area(Logic Block)	Fmax(MHz)
Behavioural	184	186
ALU1	223	154
ALU2	235	178
ALU + comp	226	106
ALU + sub	145	169
2Sub + comp	186	180
2 Sub + 2 comp (Proposed Method)	198	236

Therefore, the best method in terms of area is the forth method (2 subtractor + 1 comparator), and the best method in terms of speed is the proposed method (2 subtractor + 2 comparator).

References:

- ihabadly/FSMD-GCD: Basic FSMD example to calculate the Greatest Common Divisor of two 8-bit numbers (github.com)