Advanced Digital Electronics

# COURSE MINI-PROJECT

---

# Root function implementation

---

**Done by:**
**Muhammad Abdelnaby**

**Supervised by:**
**M. H.MATHIAS**

## Problem definition

In this project, we are addressing the idea of architecture choice when it comes to size, throughput, and complexity. In this sense, the root function is implemented in different algorithms and architectures.

## Architecture one.

**Newton Raphson method** The first architecture is implemented sequentially using iteration with the Newton method to find the function's zero. A root of a given input A is transformed into the zero of the function

$$y = x^2 - A$$

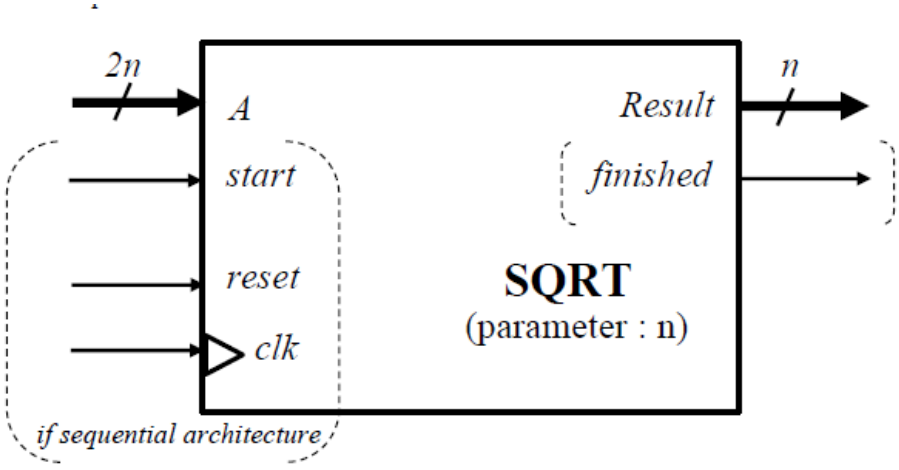
Newton's method starts with an initial guess, and a tangent to the function is generated following the equation of a straight line:

$$\frac{dy}{dx} = \frac{y - y_0}{x - x_0}$$

For the straight line, its zero, or intersection with the x-axis, is given by y = 0, by rearranging the equation, we get:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Since the intersection of that line is closer to the zero compared to the initial guess, iterating with this formula gives the zero and hence the root of A.
The design will take a top-level view as follows:



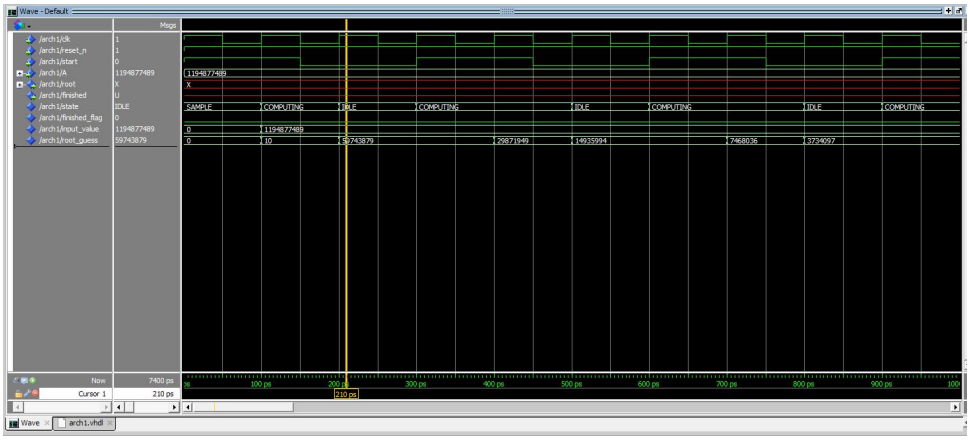

The input is of size 2N with reset, clk, and a signal to control whether the computations inside the block are allowed to continue. An n-bit unsigned vector is displayed as output, associated with a "finished" signal to indicate a flag of ready output.
The design is implemented using an FSM with 4 states. The circuit enters the "SAMPLE" state by default to initiate the internal variables, then moves to the "IDLE" state if the start is not valid. When valid, the "COMPUTE" state begins, and when finished, it enters the "DONE" state, where the output is assigned. In the "DONE" state, the circuit checks for a change in input; if so, it goes back to sample the internal signals again and repeats. The computations are sequential due to the nature of the algorithm, but a full iteration of Newton's method is done in one clock cycle.

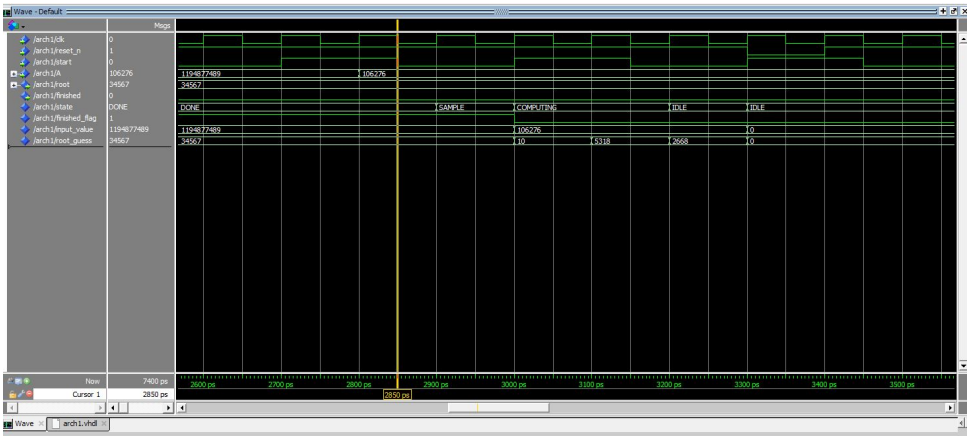
An initial guess is chosen to be A/2; this results in fewer computations compared to starting with a fixed number. However, it can be enhanced by adjusting the initial guess to the closest $2^x$ to A. This could reduce the number of iterations but would also increase the area and complexity of calculating the logarithm of a given integer. Therefore, the A/2 initial guess is sufficient to start with.

This snippet represents the computation part, which takes an initial guess and iterates each time a rising edge of the clock arrives, until the iterative formula gives the same value.

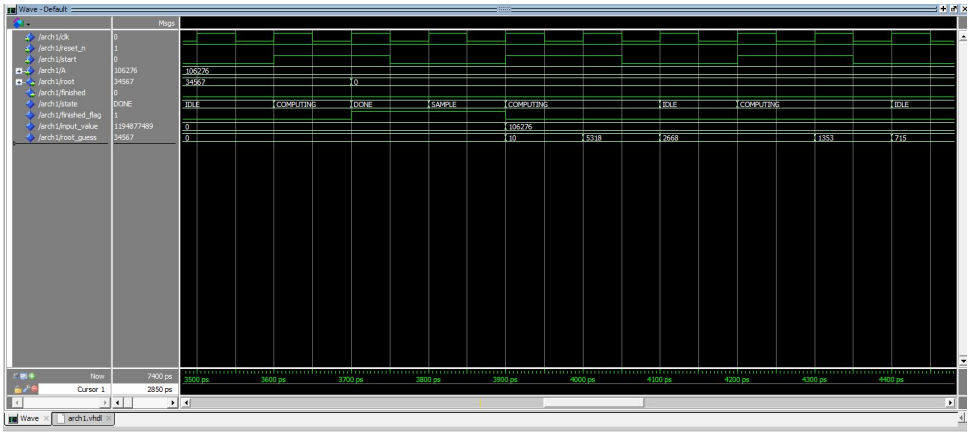
```
when COMPUTING =>
    if (input_value = 0) then
        next_guess := 0;
        root_guess <= 0;
    else
        next_guess := (root_guess + (input_value / root_guess)) / 2;
    end if;
    if next_guess = root_guess then
        finished_flag <= '1';
        state <= DONE;
        root <= to_unsigned(next_guess, n);
        finished <= finished_flag;
    else
        root_guess <= next_guess;
        if start = '0' then
            state <= IDLE;
            end if;
    end if;
```

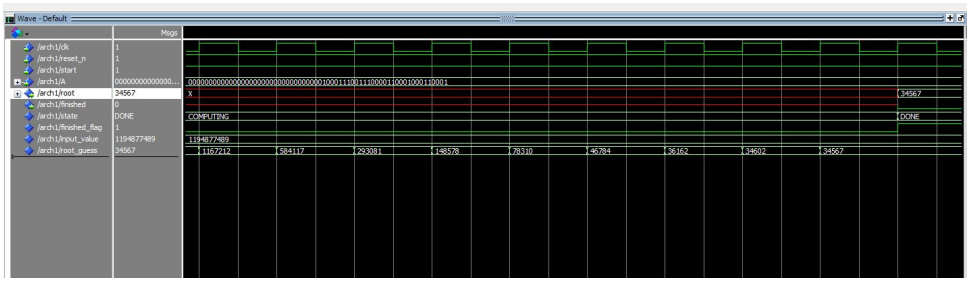
The simulation was done to test both computations and functionality of control signals:





Testing the states functionality and control of reset and start signals over the combinational logic





Handling different test cases including zero and large numbers.

## Synthesis and analysis

As a rule of thumb, the circuit finishes with around 5-6 iterations, and the order of iterations is around $\log_2(n)$, but it increases for larger inputs since the initial guess increases linearly while the function's order is higher.
Using Quartus, the utilization and maximum frequency are reported as follows:

```
Analysis & Synthesis Summary
------------------------------------+-------------------------------------------------
Analysis & Synthesis Status         ; Successful - Thu Dec 26 22:50:14 2024
Quartus II 64-Bit Version           ; 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name                       ; arch1
Top-level Entity Name               ; arch1
Family                              ; Cyclone II
Total logic elements                ; 1,503
    Total combinational functions   ; 1,471
    Dedicated logic registers       ; 101
Total registers                     ; 101
Total pins                          ; 100
Total virtual pins                  ; 0
Total memory bits                   ; 0
Embedded Multiplier 9-bit elements ; 0
Total PLLs                          ; 0
```

| Slow Model Fmax Summary | | | |
|---|---|---|---|
| Fmax | Restricted Fmax | Clock Name | Note |
| 5.86 MHz | 5.86 MHz | myclock | |

Using the Cyclone II family, a number of 1,503 logic elements were used with a maximum frequency of 5.86 MHz achieved.

## Architecture two.

In this design, the top-level module and main functionality remain the same, but the computing algorithm is changed to a non-restoring algorithm to save computational complexity by using shifters instead of divisions. The algorithm is based on computing the output starting from the most significant bit by shifting the input by two bits for each two operations and deciding through an internal signal R if it results in 1 at that bit or 0.

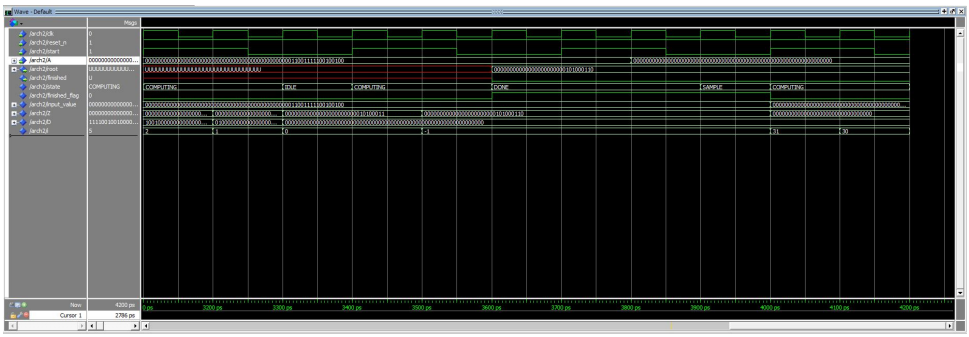
```
D = A ;
R = 0 ;
Z = 0 ;
For i = n-1 downto 0 do
        If R>=0 then
                R = R * 4 + D / 2^(2n-2) – ( 4 * Z + 1 ) ;
        Else
                R = R * 4 + D / 2^(2n-2) +( 4 * Z + 3 ) ;
        End If ;
        If R>=0 then
                Z = 2 * Z + 1 ;
        Else
                Z = 2 * Z ;
        End If ;
        D = D*4 ;
End For ;
Result = Z;
```

The design takes the same frame of states done in architecture one but the computations part is adjusted as follows:

```
when COMPUTING =>
    if i >= 0 then
        i <= i-1;
        if (signed(R)>= 0)then
            R := shift_left(R, 2) + shift_right(D, 2*n-2) - shift_left(Z, 2) - to_unsigned(1, 2*n);
        else
            R := shift_left(R, 2) + shift_right(D, 2*n-2) + shift_left(Z, 2) + to_unsigned(3, 2*n);
        end if;
        if signed(R)>=0 then
            Z <= shift_left(Z,1)+ to_unsigned(1, n);
            else
            Z <= shift_left(Z, 1);
        end if;
        D <= shift_left(D, 2);
        if start = '0' then
            state <= IDLE;
        end if;
    else
        finished_flag <= '1';
        state <= DONE;
        root <= Z;
        finished <= finished_flag;
    end if;
```

An integer iterates the output bits from MSB and decrements each cycle when the bit is calculated until done.

The simulation was done to test both computations and functionality of control signals:



The state transitioning is the same as the first architecture.

## Synthesis and analysis

Since the design computes an output bit for each cycle, an N number of cycles is needed for input of size 2N. the number of cycles is fixed nevertheless, the value of the input is within the input size.
Using Quartus for the same kit as in Architecture One:

```
Analysis & Synthesis Summary
-----------------------------------+-------------------------------------------------
Analysis & Synthesis Status        ; Successful - Thu Dec 26 23:51:50 2024
Quartus II 64-Bit Version          ; 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name                      ; arch2
Top-level Entity Name              ; arch2
Family                             ; Cyclone IV GX
Total logic elements               ; 693
    Total combinational functions  ; 596
    Dedicated logic registers      ; 293
Total registers                    ; 293
Total pins                         ; 100
Total virtual pins                 ; 0
Total memory bits                  ; 0
Embedded Multiplier 9-bit elements ; 0
Total GXB Receiver Channel PCS     ; 0
Total GXB Receiver Channel PMA     ; 0
Total GXB Transmitter Channel PCS  ; 0
Total GXB Transmitter Channel PMA  ; 0
Total PLLs                         ; 0
```

```
Slow 1200mV 85C Model Fmax Summary
------------+-----------------+------------+------
Fmax        ; Restricted Fmax ; Clock Name ; Note
------------+-----------------+------------+------
125.55 MHz ; 125.55 MHz       ; clk        ;
```

A number of 693 logic elements were used with a maximum frequency of 125.55 MHz achieved. This is much optimized in comparison to architecture one since the division complexity is high, compatible to the complexity of finding a root itself.

## Architecture Three.

In this design, the same non-restoring algorithm is also used since it showed highly optimized performance compared to the Newton method.

Archecitrue 3 is a modified combinatorial form of architecture 2. That is done by combining the iterations in one clock cycle through a for loop running sequentially in one process filling the output from MSB to LSB.

A modification is done as follows:

```
for i in n-1 downto 0 loop
    if (signed(R)>= 0)then
        R := shift_left(R, 2) + shift_right(D, 2*n-2) - shift_left(Z, 2) - to_unsigned(1, 2*n);
    else
        R := shift_left(R, 2) + shift_right(D, 2*n-2) + shift_left(Z, 2) + to_unsigned(3, 2*n);
    end if;
    if signed(R)>=0 then
        Z := shift_left(Z,1)+ to_unsigned(1, n);
        else
        Z := shift_left(Z, 1);
    end if;
    D := shift_left(D, 2);
end loop;
```

The simulation was done to test the logic:

The test vector includes corners such as 0, 1, and bigger squares all controlled by the negative reset.

## Synthesis and analysis

Since the design is combinational, the square is calculated each cycle, this will increase the path delay and size of the circuit.
Using Quartus for the same kit as in Architecture One:

| Flow Summary | |
|---|---|
| Flow Status | ; Successful - Fri Dec 27 01:12:12 2024 |
| Quartus II 64-Bit Version | ; 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition |
| Revision Name | ; arch3 |
| Top-level Entity Name | ; arch3 |
| Family | ; Cyclone IV GX |
| Total logic elements | ; 7,547 / 21,280 ( 35 % ) |
| Total combinational functions | ; 7,547 / 21,280 ( 35 % ) |
| Dedicated logic registers | ; 0 / 21,280 ( 0 % ) |
| Total registers | ; 0 |
| Total pins | ; 97 / 167 ( 58 % ) |
| Total virtual pins | ; 0 |
| Total memory bits | ; 0 / 774,144 ( 0 % ) |
| Embedded Multiplier 9-bit elements | ; 0 / 80 ( 0 % ) |
| Total GXB Receiver Channel PCS | ; 0 / 4 ( 0 % ) |
| Total GXB Receiver Channel PMA | ; 0 / 4 ( 0 % ) |
| Total GXB Transmitter Channel PCS | ; 0 / 4 ( 0 % ) |
| Total GXB Transmitter Channel PMA | ; 0 / 4 ( 0 % ) |
| Total PLLs | ; 0 / 4 ( 0 % ) |
| Device | ; EP4CGX22CF19C6 |
| Timing Models | ; Final |

Slow 1200mV 85C Model Fmax Summary

| Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|
| 2.34 MHz | 2.34 MHz | myclock | |

A total of 7547 logic elements were used, with a maximum frequency of 2.34 MHz achieved. This represents a much larger area compared to Architecture 2, as all iterations are combinational without shared resources for the shifter and ALUs. Additionally, this is assumed to be the lowest working frequency possible since it has the longest path among the other architectures and any other possible design.

## Architecture Four.

Due to the low frequency in architecture 3, pipelining is proposed to increase the throughput. The loop is broken into 2 main loops each calculating half the bits of the out. The modification is done as follows:

```
-- first stage
R := (others => '0');
Z := (others => '0');
D := A;
for i in n-1 downto n/2 loop
    if (signed(R)>= 0)then
        R := shift_left(R, 2) + shift_right(D, 2*n-2) - shift_left(Z, 2) - to_unsigned(1, 2*n);
    else
        R := shift_left(R, 2) + shift_right(D, 2*n-2) + shift_left(Z, 2) + to_unsigned(3, 2*n);
    end if;
    if signed(R)>=0 then
        Z := shift_left(Z,1)+ to_unsigned(1, n);
        else
        Z := shift_left(Z, 1);
    end if;
    D := shift_left(D, 2);
```

For the first stage, an iterative looping is done to set the most significant n/2 bits of the output.

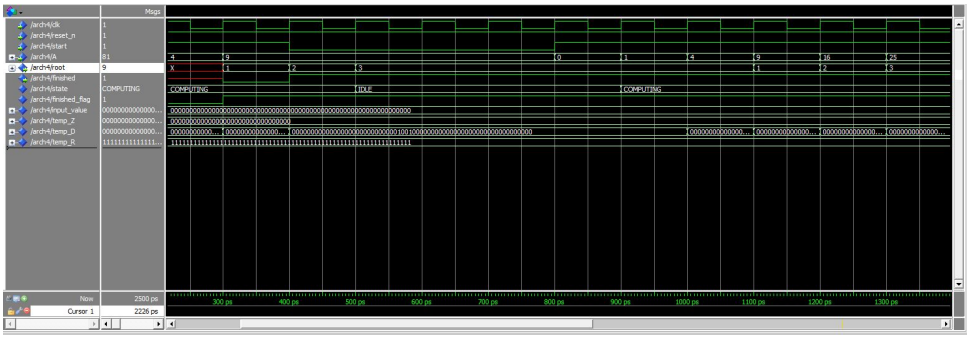
```
temp_R <= R;
temp_Z <= Z;
temp_D <= D;
R_dash := temp_R;
Z_dash := temp_Z;
D_dash := temp_D;
```

A registers state to save the output state from the first stage in registers.

```
for i in (n/2)-1 downto 0 loop
    if (signed(R_dash)>= 0)then
        R_dash := shift_left(R_dash, 2) + shift_right(D_dash, 2*n-2) - shift_left(Z_dash, 2) - to_unsigned(1, 2*n);
    else
        R_dash := shift_left(R_dash, 2) + shift_right(D_dash, 2*n-2) + shift_left(Z_dash, 2) + to_unsigned(3, 2*n);
    end if;
    if signed(R_dash)>=0 then
        Z_dash := shift_left(Z_dash,1)+ to_unsigned(1, n);
        else
        Z_dash := shift_left(Z_dash, 1);
    end if;
    D_dash := shift_left(D_dash, 2);
end loop;
```

A second stage operates on the previous cycle input to calculate the n/2-1 to 0 bits of the output.

The simulation was done to test both computations and functionality of control signals:



Handling different test cases including zero and large numbers controlled by start signal changes and their effect on the state, once the start is 1, the design computes the root of the last 2 inputs.

## Synthesis and analysis

Since the design is pipelined, the latency has increased a bit the working frequency is divided due to the pipelining stages. Using Quartus for the same kit as in Architecture One:

```
Flow Summary
-------------------------------------+-------------------------------------------------
Flow Status                          ; Successful - Fri Dec 27 00:36:19 2024
Quartus II 64-Bit Version            ; 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name                        ; arch4
Top-level Entity Name                ; arch4
Family                               ; Cyclone IV GX
Total logic elements                 ; 7,555 / 21,280 ( 36 % )
    Total combinational functions    ; 7,541 / 21,280 ( 35 % )
    Dedicated logic registers        ; 147 / 21,280 ( < 1 % )
Total registers                      ; 147
Total pins                           ; 100 / 167 ( 60 % )
Total virtual pins                   ; 0
Total memory bits                    ; 0 / 774,144 ( 0 % )
Embedded Multiplier 9-bit elements   ; 0 / 80 ( 0 % )
Total GXB Receiver Channel PCS       ; 0 / 4 ( 0 % )
Total GXB Receiver Channel PMA       ; 0 / 4 ( 0 % )
Total GXB Transmitter Channel PCS    ; 0 / 4 ( 0 % )
Total GXB Transmitter Channel PMA    ; 0 / 4 ( 0 % )
Total PLLs                           ; 0 / 4 ( 0 % )
Device                               ; EP4CGX22CF19C6
Timing Models                        ; Final
```

```
Slow 1200mV 85C Model Fmax Summary
----------+-----------------+------------+------
Fmax      ; Restricted Fmax ; Clock Name ; Note
----------+-----------------+------------+------
8.36 MHz  ; 8.36 MHz        ; clk        ;
----------+-----------------+------------+------
```

A number of 7555 logic elements were used with a maximum frequency of 8.36 MHz achieved. This is almost the same utilization as in Architecture 3 but the frequency is much increased due to deciding the critical path.

## Architecture Five.

In this architecture, a new design is proposed to minimize the area using resource sharing. The design is composed of one ALU that is accessed once each cycle, and an FSM associated with a decoder to control the enable and control signals for each circuit block.
A structural approach is used in this design to gain more control over the synthesized circuit.

To design the finite state machine, a truth table is generated:

| | A | B | C | D | E | F | | | | A | B | C | D | E | F | | | | A | B | C | D | E | F | | | | A | B | C | D | E | F | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | i=0 | reset | S1 | S0 | start | input | S1 | S0 | | i=0 | reset | S0 | S1 | start | input | S0 | S1 | | i=0 | reset | S0 | S1 | start | input | S0 | S1 | | i=0 | reset | S0 | S1 | start | input | S0 | S1 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 32 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 48 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 17 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 33 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 49 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 18 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 34 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 50 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 19 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 35 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 51 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 20 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 36 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 52 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 21 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 37 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 53 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 22 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 38 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 54 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 7 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 23 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 39 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 55 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 24 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 40 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 56 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 9 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 25 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 41 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 57 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 10 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 26 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 42 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 58 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 11 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 27 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 43 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 59 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 12 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 28 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 44 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 60 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 13 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 29 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 45 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 61 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 14 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 30 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 46 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 62 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 15 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 31 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 47 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 63 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

The inputs of the FSM are as:: check if the counter is 0 and computations are done, the negative reset, the two bits of the current state, start signal, and check signal if the input has changed.

Using the Karnaugh map, a function for S0 and S1 are calculated as follows:

$$s_0 = BC'E + BD'E + A'BCD' + BCDF'$$

$$s_1 = BC'E + BD'E + A'BCD' + BCDF'$$

Where A, B, C, D, E, and F are the inputs in the same order above for simplicity.

The implementation in VHDL was done using an impure function as follows:

```
impure function FSM(A: integer; B : std_logic; C : std_logic; D: std_logic; E: std_logic; F: std_logic) return std_logic_vector is
--FSM(i_end: integer; reset : std_logic; state : std_logic, state : std_logic; start: std_logic; input_chage: boolean)
variable outp : std_logic_vector(1 downto 0);
variable sig : std_logic;
begin
if A = 0 then
    sig:= '0';
    i <= n-1;
else
    i <= i-1;
    sig:= '1';
end if;
outp(0) := (B and not(C) and not(E)) or (B and not(D) and not(E)) or (not(sig) and B and C and not(D)) or (B and C and D and not(F));
outp(1) := (B and not(C) and E) or (B and not(D) and E) or (not(sig) and B and C and not(D)) or (B and C and D and not(F));
return outp;
end function;
```

Then designed elementary blocks such as mux and shifter and a special shifter for Z in a way to reduce the usage of the ALU:

```
function zshifter(a: unsigned; lsbit : std_logic) return unsigned is
    variable outp : unsigned(2*n-1 downto 0);
    begin
    outp := shifter(a,1, '0');
    outp(0) := (outp(0) and not(lsbit)) or (not outp(0)  and lsbit);
    return outp;
end function;
```

And a design of a special ALU dedicated to the R iterative equation

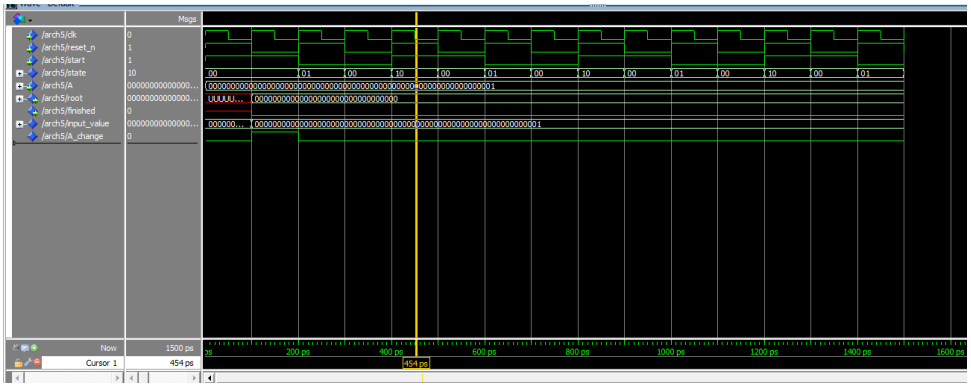
```
function ALU(R: unsigned; D: unsigned; Z: unsigned; direction : std_logic) return unsigned is
variable outp : unsigned(2*n-1 downto 0);
begin
outp := mux(shifter(R,2, '0') + shifter(D,2*n-2, '1') - shifter(D,2*n-2, '0') - to_unsigned(1, 2*n),
            shifter(R,2, '0') + shifter(D,2*n-2, '1') + shifter(D,2*n-2, '0') + to_unsigned(3, 2*n),
            direction);
return outp;
end function;
```

The main process is responsible for updating all signals using the defined pure functions as follows:

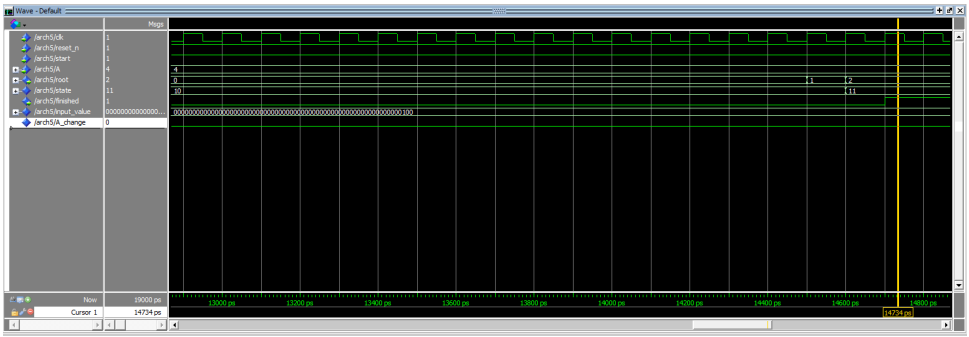
```
if rising_edge(clk) then
    finished <= state(0) and state(1);
    state <= FSM(i, reset_n, state(1),state(0),start,A_change); --i=0   reset   S1  S0  start   input_change
    R := mux(mux(R, ALU(R, D, Z, R_sign ), state(0)), mux(R, to_unsigned(0, 2*n), state(0)), state(1));
    R_sign  :=ALU_sign(R);
    Z := mux(mux(Z, zshifter(Z, R_sign), state(0)), mux(Z, to_unsigned(0, 2*n), state(0)), state(1));
    i := to_integer(mux(mux(to_unsigned(i, 2*n), to_unsigned(i-1, 2*n), state(0)), to_unsigned(n-1, 2*n), state(1)));
    D := mux(mux(D, shifter(D,2,'0'), state(0)), mux(D, A, state(0)), state(1));
    input_value <= mux(input_value, A, state(1) or state(0));
    root <= Z(n-1 downto 0);
    if input_value /= A then
        A_change <= '1';
    else
        A_change <= '0';
    end if;
end if;
```

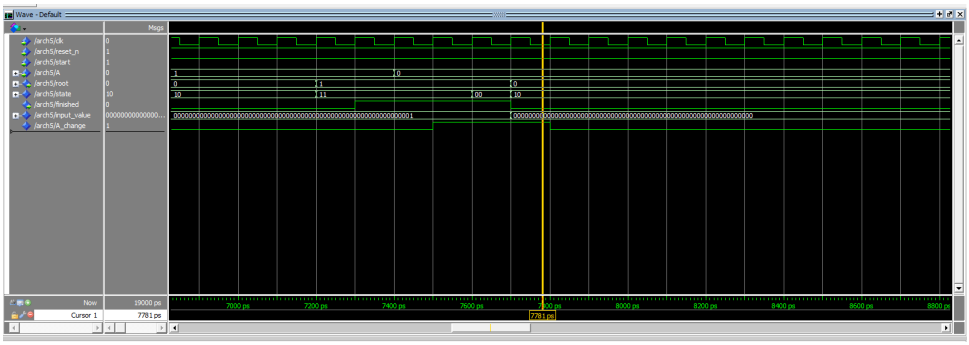
**ARCH 5 Declaration:**
I declare my understanding of the significance of working with the code at a lower level, giving instructions and controlling each bit while avoiding any pre-existing functions when possible. The procedure I used started with building the design to navigate from one state to another properly, which was tested and verified. The second step was to add the root iterations and test them; this part is not reliable yet, as will be seen in the simulation section. The last part involved adjusting each snippet of the code, which was done modularly to enable adjustments in each part. The third stage was not tackled as planned due to logical errors in the second step. Therefore, the code can be simplified into more detailed instructions to control all possible signals, thus allowing for more control over the synthesis.

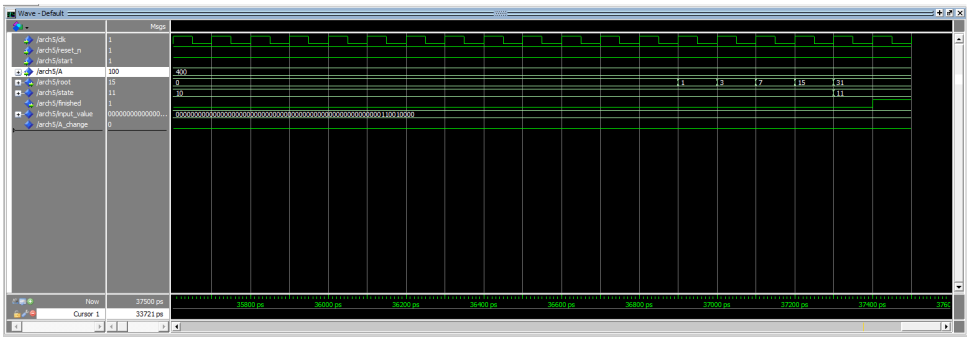
The simulation was done to test both computations and functionality of control signals:




The FSM works properly.





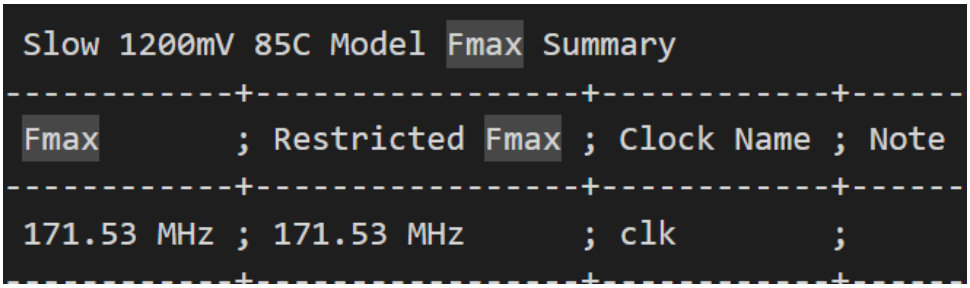


The results are not always right, due to logical error, some results are calculated properly while others are not.

## Synthesis and analysis

The design has some logical errors in assigning the controlling signals, so no additional functionality will be needed. An analysis of the synthesis of this design will show that it has almost the same functionality as the data path, and the level of complexity for deriving the controllers of the decoder and FSM is almost the same.
Since the design performs an addition each clock cycle, and hence iterations, the design needs N cycles to finish a root for an input of size 2N.

```
Flow Summary
-------------------------------------+-----------------------------------------------
Flow Status                          ; Successful - Fri Dec 27 00:39:23 2024
Quartus II 64-Bit Version            ; 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name                        ; arch5
Top-level Entity Name                ; arch5
Family                               ; Cyclone IV GX
Total logic elements                 ; 360 / 21,280 ( 2 % )
    Total combinational functions    ; 327 / 21,280 ( 2 % )
    Dedicated logic registers        ; 227 / 21,280 ( 1 % )
Total registers                      ; 227
Total pins                           ; 100 / 167 ( 60 % )
Total virtual pins                   ; 0
Total memory bits                    ; 0 / 774,144 ( 0 % )
Embedded Multiplier 9-bit elements   ; 0 / 80 ( 0 % )
Total GXB Receiver Channel PCS       ; 0 / 4 ( 0 % )
Total GXB Receiver Channel PMA       ; 0 / 4 ( 0 % )
Total GXB Transmitter Channel PCS    ; 0 / 4 ( 0 % )
Total GXB Transmitter Channel PMA    ; 0 / 4 ( 0 % )
Total PLLs                           ; 0 / 4 ( 0 % )
Device                               ; EP4CGX22CF19C6
Timing Models                        ; Final
```



Slow 1200mV 85C Model Fmax Summary

| Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|
| 171.53 MHz | 171.53 MHz | clk | |

A number of 360 logic elements were used with a maximum frequency of 171.53 MHz achieved. This is around 5.3 MHz throughput. The utilization is the most optimized among the rest of the designs due to sharing one ALU.