Prime Number

Problem statement: The problem statement on which this report is done is "finding out whether the given number is prime number or not without using any divider".

Prime numbers: The numbers that are only divisible by '1' and itself are known as prime numbers.

There are many algorithms to check for the prime number, some of them are

1. **Trial Division Method:**
    - This method involves checking if the number is divisible by any integer from 2 to the square root of the number.
    - If it's not divisible by any integer in this range, it's considered prime.
    - This method is straightforward but not the most efficient for very large numbers.
2. **Sieve of Eratosthenes:**
    - The Sieve of Eratosthenes is a classic algorithm for generating all prime numbers up to a certain limit.
    - You can use this algorithm to generate a list of primes and then check if the given number is in that list.
    - It's efficient for checking primarily up to a predefined limit.
3. **Fermat's Little Theorem:**
    - Fermat's Little Theorem states that if p is prime, then for any integer a**, a ^ (p-1) ≡ 1 (mod p)** unless 'a' is divisible by p.
    - You can use this theorem to test the primarily of a number.
    - If the theorem doesn't hold for a number, it's composite.
4. **Miller-Rabin Primality Test:**
    - The Miller-Rabin primality test is a probabilistic algorithm used to determine if a number is likely prime.
    - Similar to Fermat's test, but more reliable.
    - It uses modular exponentiation and is probabilistic.
    - Multiple iterations with different a values increase accuracy.

As mentioned above there are more number of algorithms to find out whether the given number is prime or not.

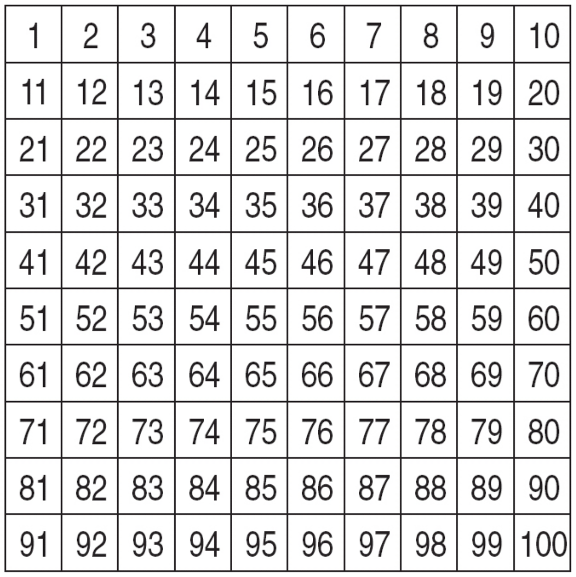
From the above mentioned algorithms, the algorithm that suits our problem statement is **Sieve of Eratosthenes.**

Sieve of Eratosthenes:

The Sieve of Eratosthenes is a time-honored and efficient algorithm for identifying prime numbers within a specified range. Named after the ancient Greek mathematician Eratosthenes and the word sieve means filtering, this method operates by iteratively sieving out the multiples of each prime number, beginning with 2 as the first prime. It systematically eliminates composite numbers, leaving behind a list of prime numbers.

Steps:

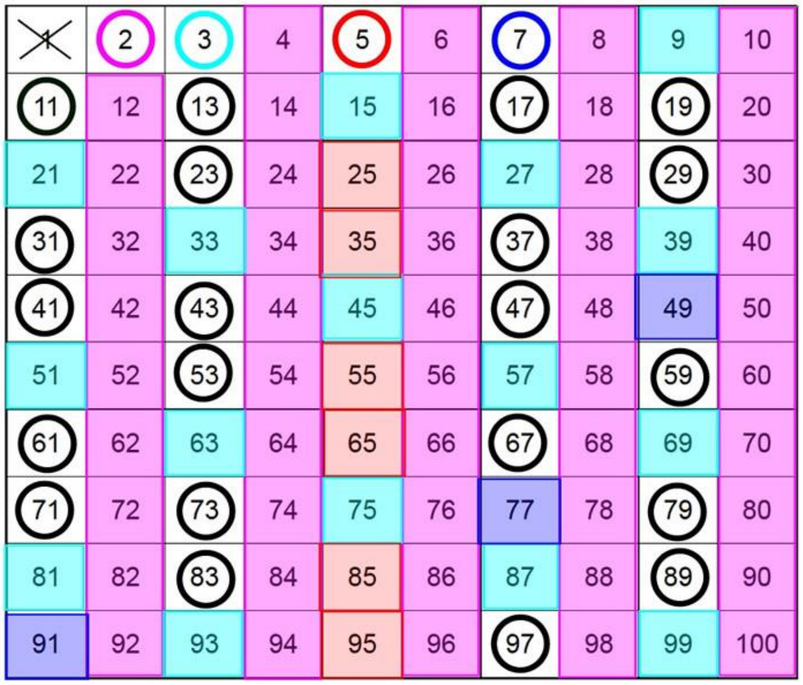
1. Create a list or an array of Boolean values, often called the "sieve" or "flags," with indices from 2 to 'n.' Initially, set all values to 'true' to indicate that all numbers are potentially prime.



1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

2. Begin with the first prime number, which is 2. Mark 2 as prime and then mark all its multiples of 2 as non-prime. To do this, iterate through the array, starting from 2, and for each multiple of 2, mark the corresponding index as 'false.'

3. Find the next unmarked number in the list, which will be the next prime number, and mark it as prime. Then, mark all its multiples as non-prime.
4. Continue this process of finding the next unmarked number, marking it as prime, and marking its multiples as non-prime until you reach the square root of 'n.' The reason to stop at the square root is that all non-prime factors of a number are less than or equal to its square root.
5. Once you've iterated through all numbers up to the square root of 'n,' the remaining unmarked numbers in the list are prime numbers.



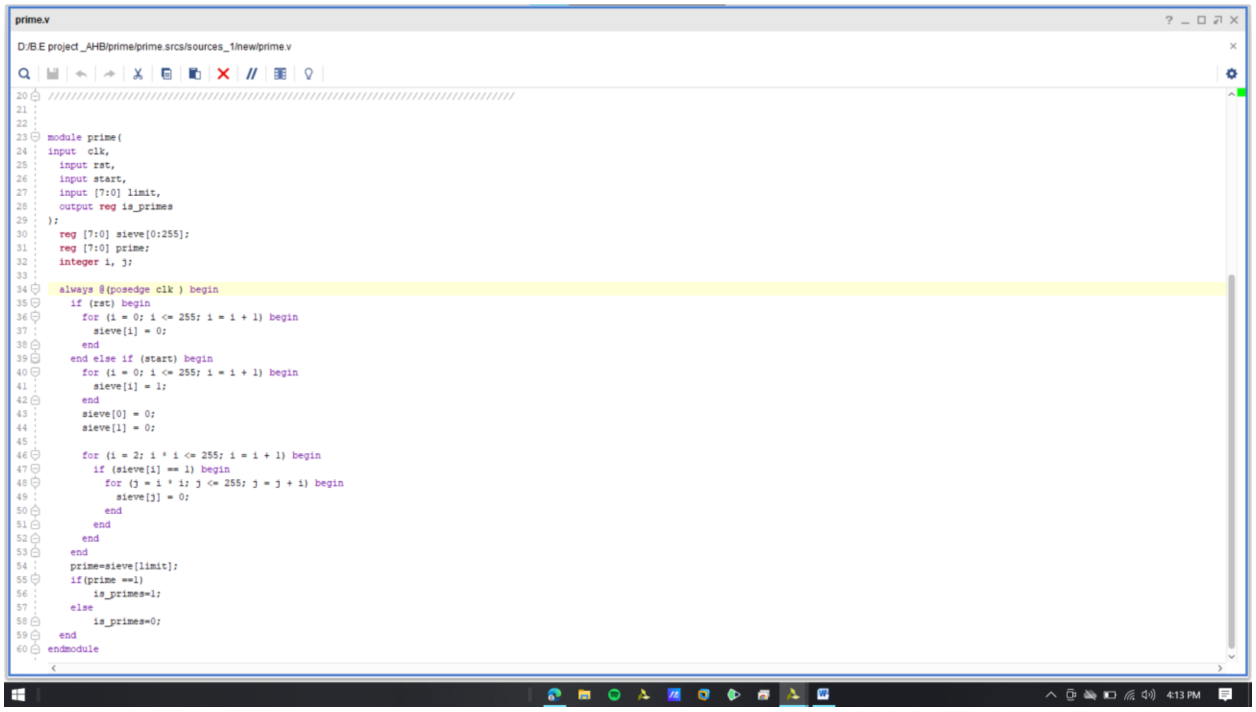
~~1~~	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

in the above diagram all the rounded numbers are the prime numbers

6. All this unmarked numbers are set to the value true, now we take the number which is need to be checked if it's a prime number or not.
7. The inputted number is prime if its position in the array is set to the value true or else the number is not a prime number.

Implementation of Sieve of Eratosthenes to check primality

1. Code

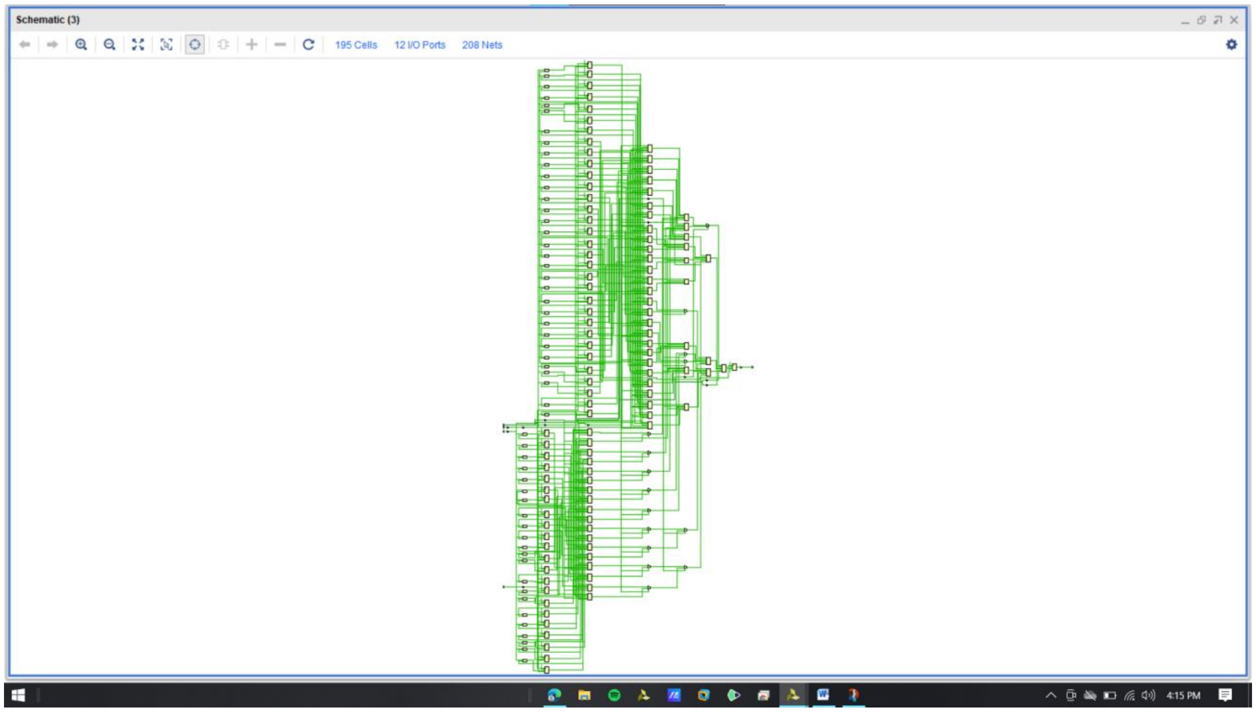


prime.v

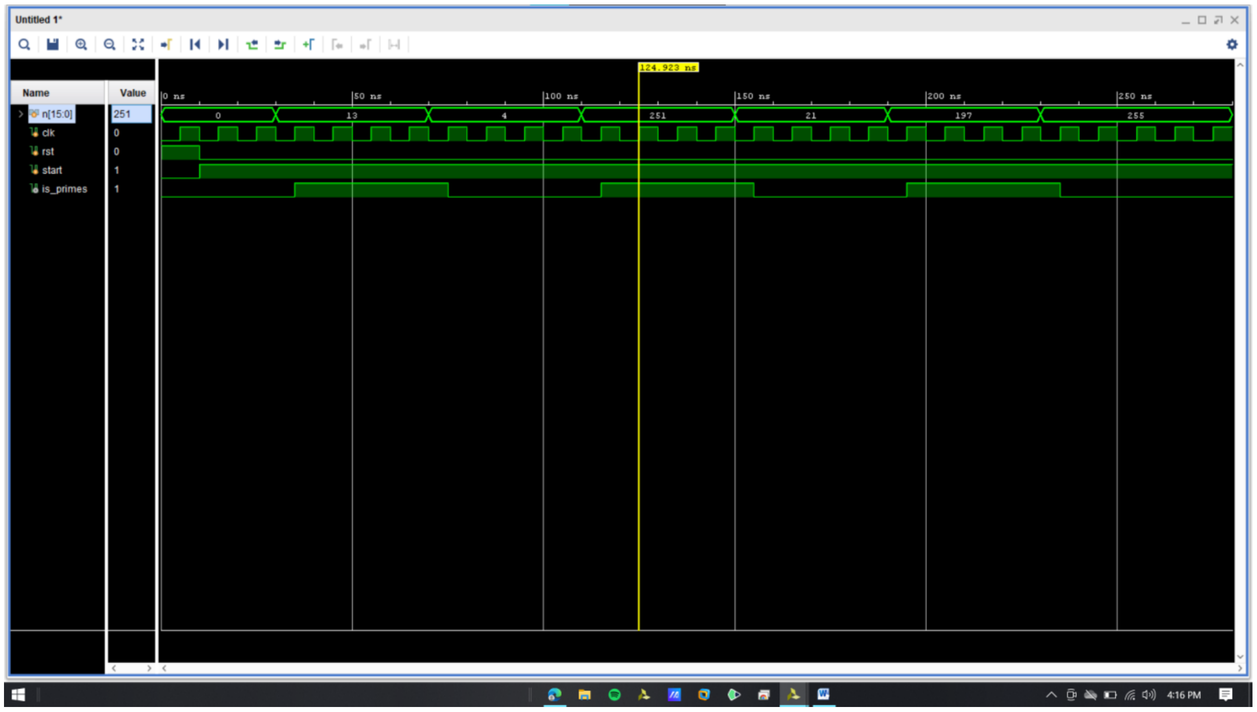
D:/B.E project _AHB/prime/prime.srcs/sources_1/new/prime.v

```verilog
//////////////////////////////////////////////////////////////////////////////////


module prime(
input  clk,
  input rst,
  input start,
  input [7:0] limit,
  output reg is_primes
);
  reg [7:0] sieve[0:255];
  reg [7:0] prime;
  integer i, j;

  always @(posedge clk ) begin
    if (rst) begin
      for (i = 0; i <= 255; i = i + 1) begin
        sieve[i] = 0;
      end
    end else if (start) begin
      for (i = 0; i <= 255; i = i + 1) begin
        sieve[i] = 1;
      end
      sieve[0] = 0;
      sieve[1] = 0;

      for (i = 2; i * i <= 255; i = i + 1) begin
        if (sieve[i] == 1) begin
          for (j = i * i; j <= 255; j = j + i) begin
            sieve[j] = 0;
          end
        end
      end
    end
    prime=sieve[limit];
    if(prime ==1)
        is_primes=1;
    else
        is_primes=0;
  end
endmodule
```

2. Schematic:

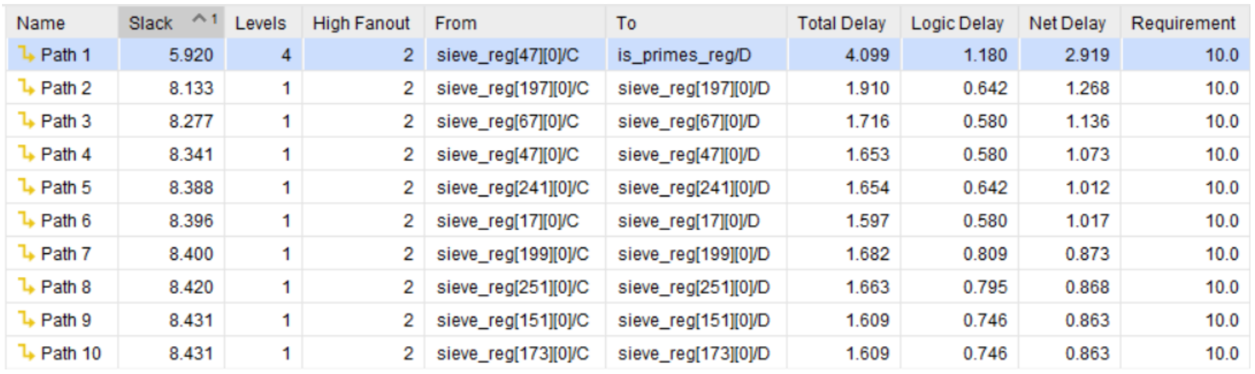


3. **Waveform:**



1.1. Timing report:

The clock used in this design has a time period of 10ns. There are a total of 20 timing paths in this design, comprising 10 setup timing paths and 10 hold timing paths. The following table lists 10 setup timing paths, including their source flip-flops, destination flip-flops, and respective slack values.



Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	5.920	4	2	sieve_reg[47][0]/C	is_primes_reg/D	4.099	1.180	2.919	10.0
Path 2	8.133	1	2	sieve_reg[197][0]/C	sieve_reg[197][0]/D	1.910	0.642	1.268	10.0
Path 3	8.277	1	2	sieve_reg[67][0]/C	sieve_reg[67][0]/D	1.716	0.580	1.136	10.0
Path 4	8.341	1	2	sieve_reg[47][0]/C	sieve_reg[47][0]/D	1.653	0.580	1.073	10.0
Path 5	8.388	1	2	sieve_reg[241][0]/C	sieve_reg[241][0]/D	1.654	0.642	1.012	10.0
Path 6	8.396	1	2	sieve_reg[17][0]/C	sieve_reg[17][0]/D	1.597	0.580	1.017	10.0
Path 7	8.400	1	2	sieve_reg[199][0]/C	sieve_reg[199][0]/D	1.682	0.809	0.873	10.0
Path 8	8.420	1	2	sieve_reg[251][0]/C	sieve_reg[251][0]/D	1.663	0.795	0.868	10.0
Path 9	8.431	1	2	sieve_reg[151][0]/C	sieve_reg[151][0]/D	1.609	0.746	0.863	10.0
Path 10	8.431	1	2	sieve_reg[173][0]/C	sieve_reg[173][0]/D	1.609	0.746	0.863	10.0

As mentioned above there are different values of slacks for different paths. The path which is having the lowest value of slack will be the critical path, because the value of slack is the time that is left out with us and if this time is less it means that the time taken by this path is more which means it the critical path.

Critical path:

Source : sieve_reg[47][0]/C

Destination : is_prime_reg/D

Source clock path :clock rise edge(0ns) + net(0ns) + IBUF(0.938ns) + net(1.972ns) + BUFG(0.096ns) + net(1.638) **=4.644ns**

Data path : FDRE(0.456ns) + LUT5(0.15ns) + LUT5(0.326ns) + LUT6(0.124ns) + LUT6(0.124ns) **=4.099ns**

Destination clock path : clock rise edge(10ns) + net(0ns) + IBUF(0.805ns) + net(1.868ns) + BUFG(0.091ns) + net(1.519ns) + clock pessimism(0.336ns) + clock uncertainty (-0.035ns) + FDRE(0.079ns) **=14.663ns**

Arrival Time = source clock path + Data path
= 4.644 + 4.099
= 8.743ns.

Required Time = Destination clock path
= 14.663ns.

Slack = Required time – Arrival time
= 14.633 – 8.743
= 5.890ns.

1.2. Maximum frequency of operation

To find the maximum frequency of operation we need to know the minimum time required by this circuit to produce the output, which can nothing but the total delay in the critical path.

We know time period and the value of slack for the critical path, using these both we can calculate minimum time required and the formula is

Min time = time period – slack

Min time = 10 – 5.890 = 4.110ns

Max frequency of operation $= \frac{1}{\min time\ required} = \frac{1}{4.110*10^{-9}}$ **= 243.30MHz**