

Gray Codes

Gray Codes

“This is rookie stuff, so I can duck out of this module, get some cookies, and come back later, right?”

History of the Gray Code

- Invented by Emile Baudot (1845-1903)
- Originally called a “cyclic-permuted” code
- Telegraph - 5 bit codes
 - Bits stored on a code wheel in the receiver
 - Wheel connected to the printing disk
 - Matched pattern on wheel and received pattern and then actuated head to print.
- Exhibited at Universal Exposition, Paris (1878)

Forming a Gray Code

- Start with all 0's
- Change the least significant bit that forms a new code word

| a | b | c | d | e |
|----------------|----------------|----------------|----------------|----------------|
| 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 |
| 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 |
| | 0 0 1 1 | 0 0 1 1 | 0 0 1 1 | 0 0 1 1 |
| | | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 |
| | | | 0 1 1 0 | 0 1 1 0 |
| | | | | 0 1 1 1 |

Binary Reflected Gray Code

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 1 |
| 3 | 1 | 0 |

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 |
| 4 | 1 | 1 | 0 |
| 5 | 1 | 1 | 1 |
| 6 | 1 | 0 | 1 |
| 7 | 1 | 0 | 0 |

| | | | | |
|----|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 |
| 5 | 0 | 1 | 1 | 1 |
| 6 | 0 | 1 | 0 | 1 |
| 7 | 0 | 1 | 0 | 0 |
| 8 | 1 | 1 | 0 | 0 |
| 9 | 1 | 1 | 0 | 1 |
| 10 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 | 1 | 0 |
| 12 | 1 | 0 | 1 | 0 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 1 | 0 | 0 | 1 |
| 15 | 1 | 0 | 0 | 0 |

Reflected Gray and Binary Codes

| | Binary | Gray |
|----|---------|---------|
| 0 | 0 0 0 0 | 0 0 0 0 |
| 1 | 0 0 0 1 | 0 0 0 1 |
| 2 | 0 0 1 0 | 0 0 1 1 |
| 3 | 0 0 1 1 | 0 0 1 0 |
| 4 | 0 1 0 0 | 0 1 1 0 |
| 5 | 0 1 0 1 | 0 1 1 1 |
| 6 | 0 1 1 0 | 0 1 0 1 |
| 7 | 0 1 1 1 | 0 1 0 0 |
| 8 | 1 0 0 0 | 1 1 0 0 |
| 9 | 1 0 0 1 | 1 1 0 1 |
| 10 | 1 0 1 0 | 1 1 1 1 |
| 11 | 1 0 1 1 | 1 1 1 0 |
| 12 | 1 1 0 0 | 1 0 1 0 |
| 13 | 1 1 0 1 | 1 0 1 1 |
| 14 | 1 1 1 0 | 1 0 0 1 |
| 15 | 1 1 1 1 | 1 0 0 0 |

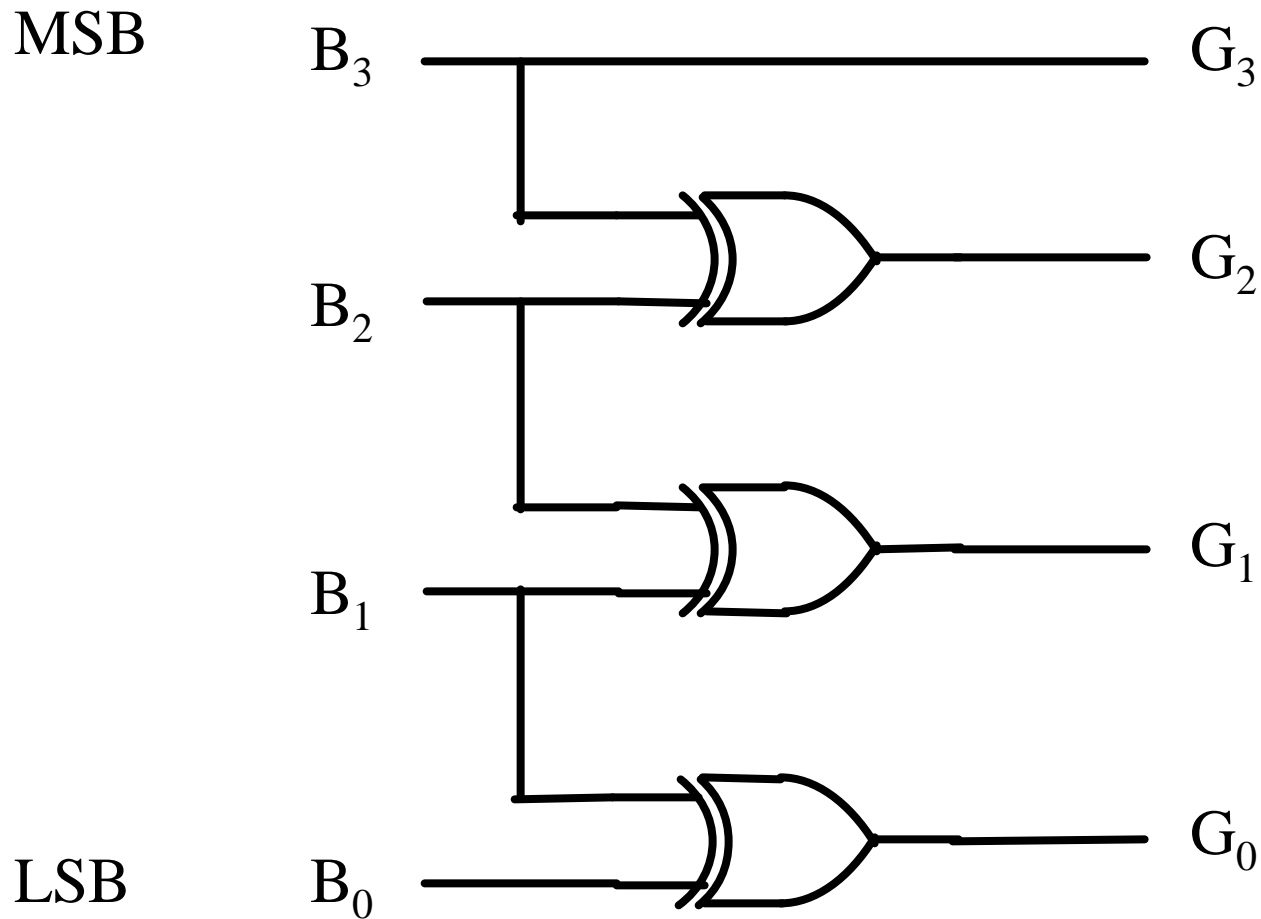
Why Gray Codes?

- Single output changes at a time
 - Asynchronous sampling
 - Permits asynchronous combinational circuits to operate in fundamental mode
 - Potential for power savings
- Multiphase, multifrequency clock generator

Effects of Errors

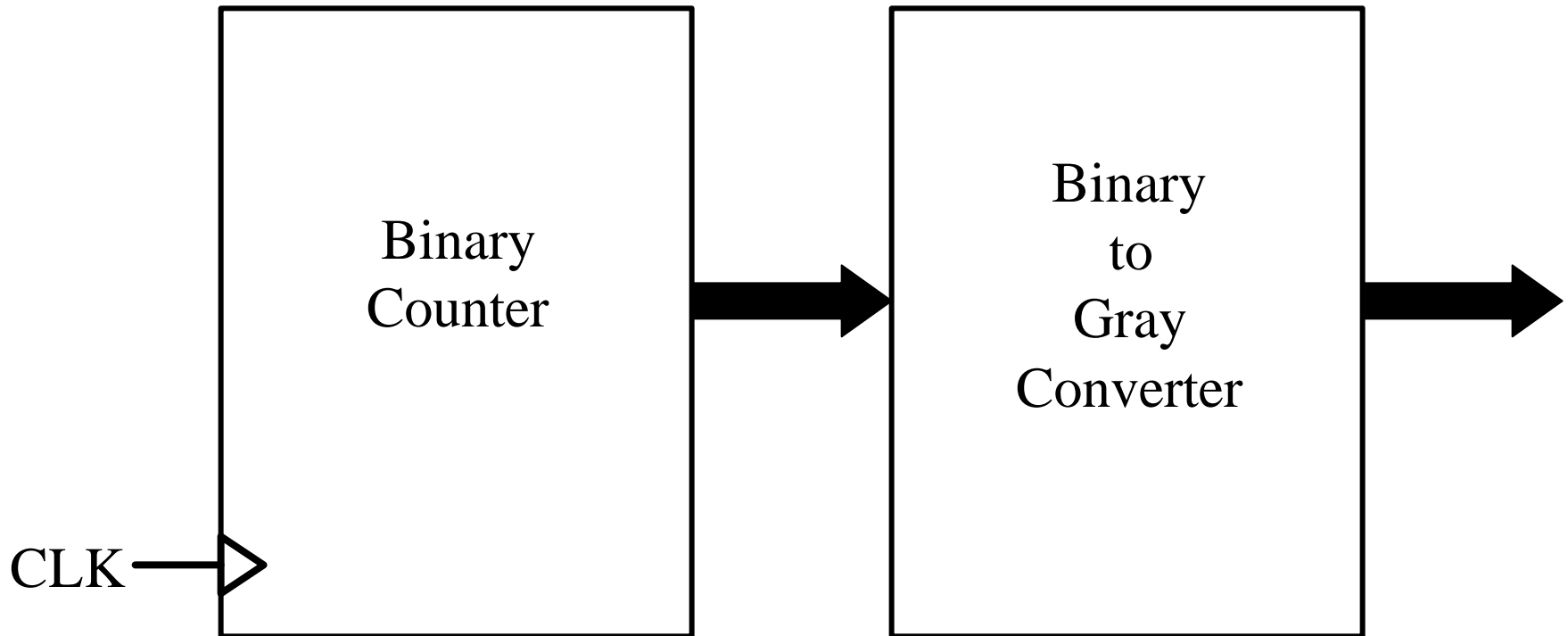
- Lockup States
 - None if all states are used
 - Can't use all states for one-hot

A Parallel Binary to Gray Converter



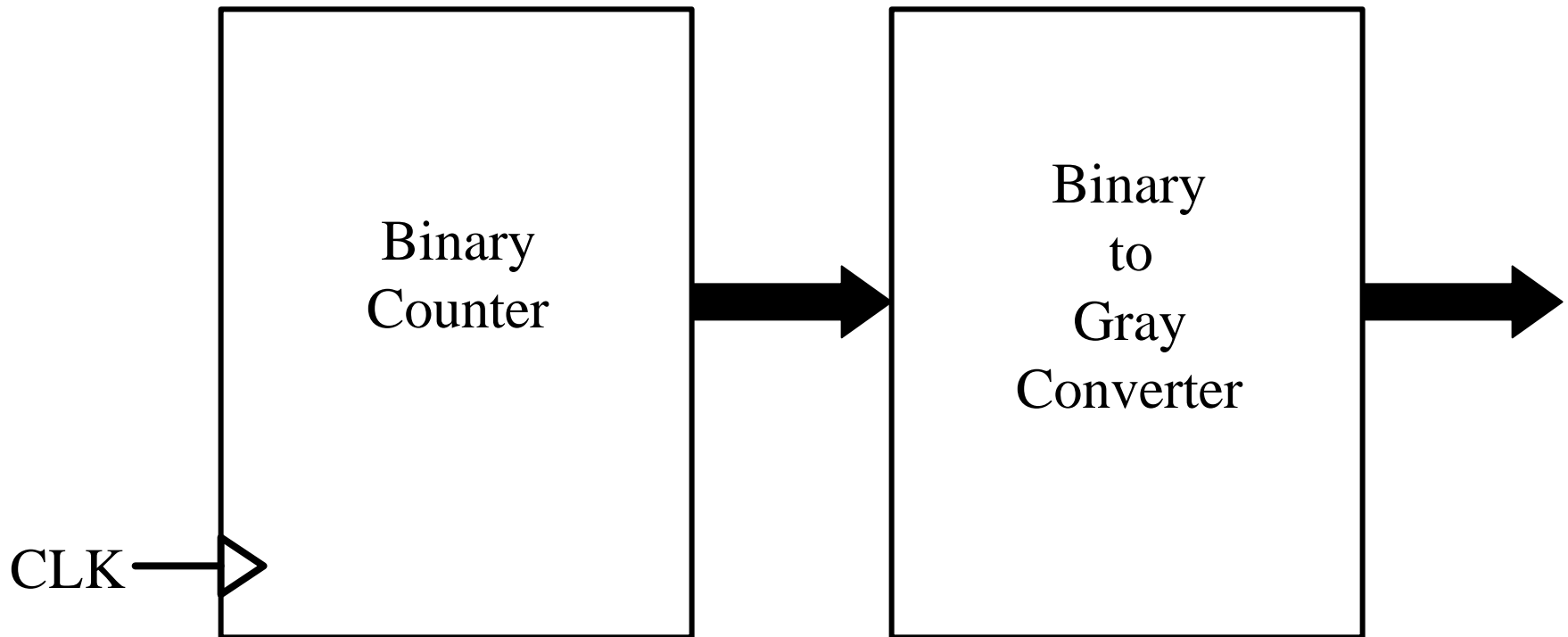
Very fast conversion

Binary Counter + Gray Code Converter: Glitch Free?

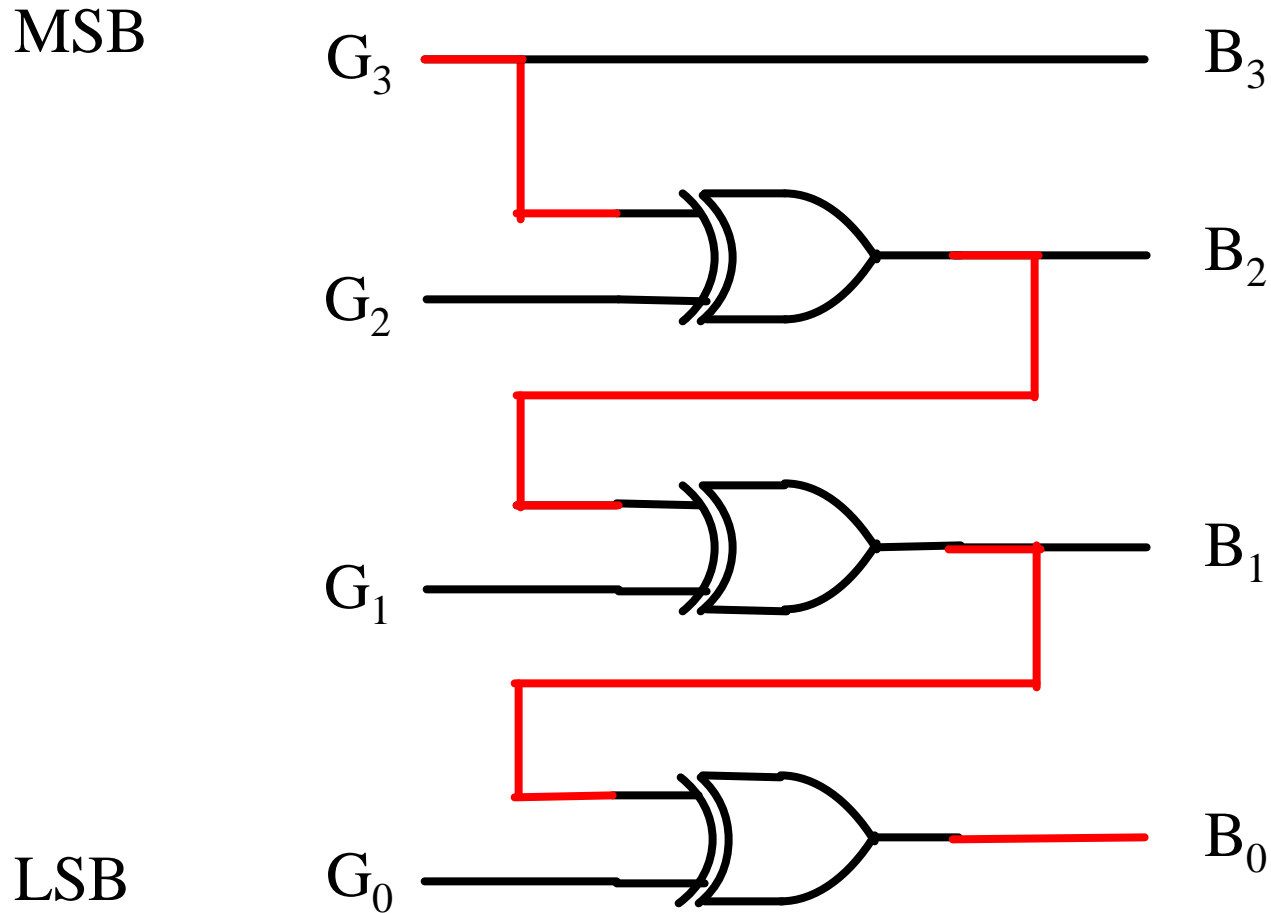


Binary Counter + Converter Glitch Free Decoding?

Discuss



A Parallel Gray to Binary Converter



Very slow conversion

VHDL Code for a 4-Bit Gray Code Sequencer (1)

```
Package Gray_Types Is
```

```
    Type States Is ( s0,    s1,    s2,    s3,  
                    s4,    s5,    s6,    s7,  
                    s8,    s9,    s10,   s11,  
                    s12,   s13,   s14,   s15  );
```

```
End Package Gray_Types;
```

VHDL Code for a 4-Bit Gray Code Sequencer (2)

```

Library IEEE;
  Use IEEE.Std_Logic_1164.all;

Library Work;
  Use Work.Gray_Types.All;

Library synplify;
  Use synplify.attributes.all;

Entity Gray_Code Is
  Port ( Clock      : In  Std_Logic;
        Reset_N    : In  Std_Logic;
        Q           : Out States );
End Entity Gray_Code;

Architecture RTL of Gray_Code Is
  Attribute syn_netlist_hierarchy of RTL : architecture is false;

  Signal IQ : States;
  Attribute syn_encoding of IQ : signal is "gray";

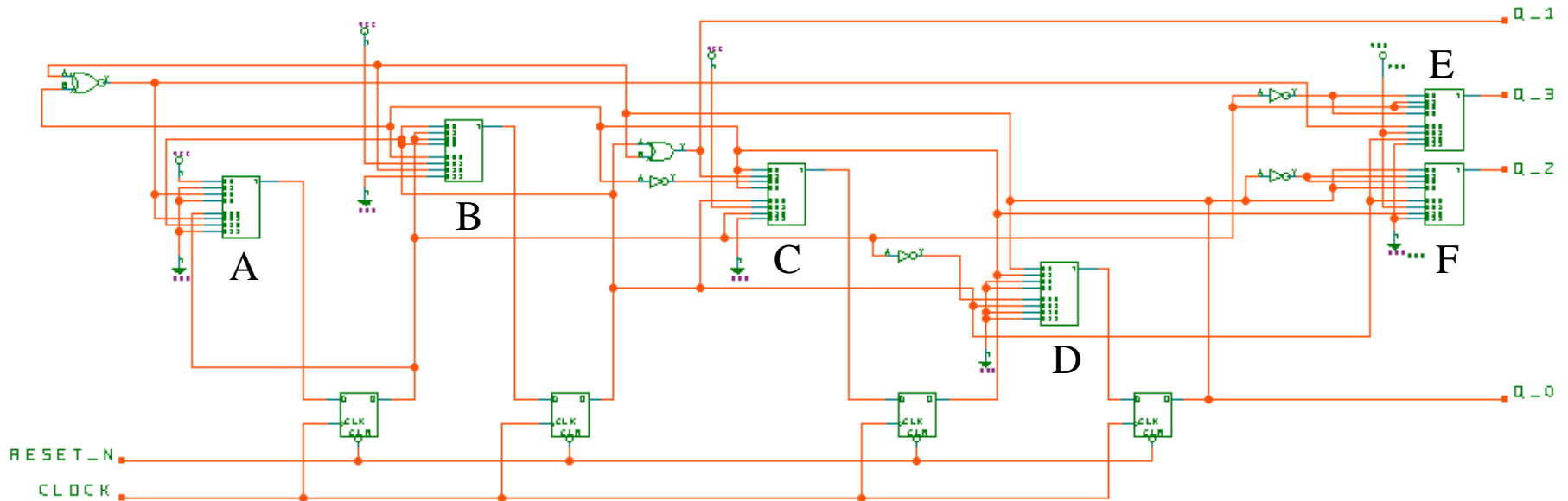
  Begin
    GC: Process ( Clock, Reset_N )
      Begin
        If ( Reset_N = '0' )
          Then IQ <= s0;
        Else If Rising_Edge ( Clock )
          Then Case IQ Is
            When s0      => IQ <= s1;
            When s1      => IQ <= s2;
            When s2      => IQ <= s3;
            When s3      => IQ <= s4;
            When s4      => IQ <= s5;
            When s5      => IQ <= s6;
            When s6      => IQ <= s7;
            When s7      => IQ <= s8;
            When s8      => IQ <= s9;
            When s9      => IQ <= s10;
            When s10     => IQ <= s11;
            When s11     => IQ <= s12;
            When s12     => IQ <= s13;
            When s13     => IQ <= s14;
            When s14     => IQ <= s15;
            When s15     => IQ <= s0;
            When Others => IQ <= s0;
          End Case;
        End If;
      End If;
    End Process GC;

    Q <= IQ;

  End Architecture RTL;

```

Synplicity Output for a Gray Code Sequencer - SX Target



Logic Equations:

$$A: \sim D2 \sim S10 + D2 \sim S00$$

$$B: D0 \sim S00 \sim S10 + D1 S00 \sim S10 + D1 \sim S00 S10 + D0 S00 S10$$

$$C: D0 \sim S00 \sim S10 + D1 S00 \sim S10 + \sim D0 \sim S00 S10 + D0 S00 S10$$

$$D: D0 \sim S01 + D1 \sim S00 S01 + D0 S00$$

$$E: \sim D0 \sim S00 \sim S10 + D0 S00 \sim S10 + D0 \sim S00 S10 + \sim D0 S00 S10$$

$$F: D0 \sim S00 \sim S10 + \sim D0 S00 \sim S10 + \sim D0 \sim S00 S10 + D0 S00 S10$$

Synplicity Output for a Gray Code Sequencer - SX Target

```
net -vsm "D:\designs\  
sequencers\gray_code4.vsm"
```

```
clock clock 1 0  
stepsize 500ns
```

```
vector q q_3 q_2 q_1 q_0  
radix bin q  
watch q
```

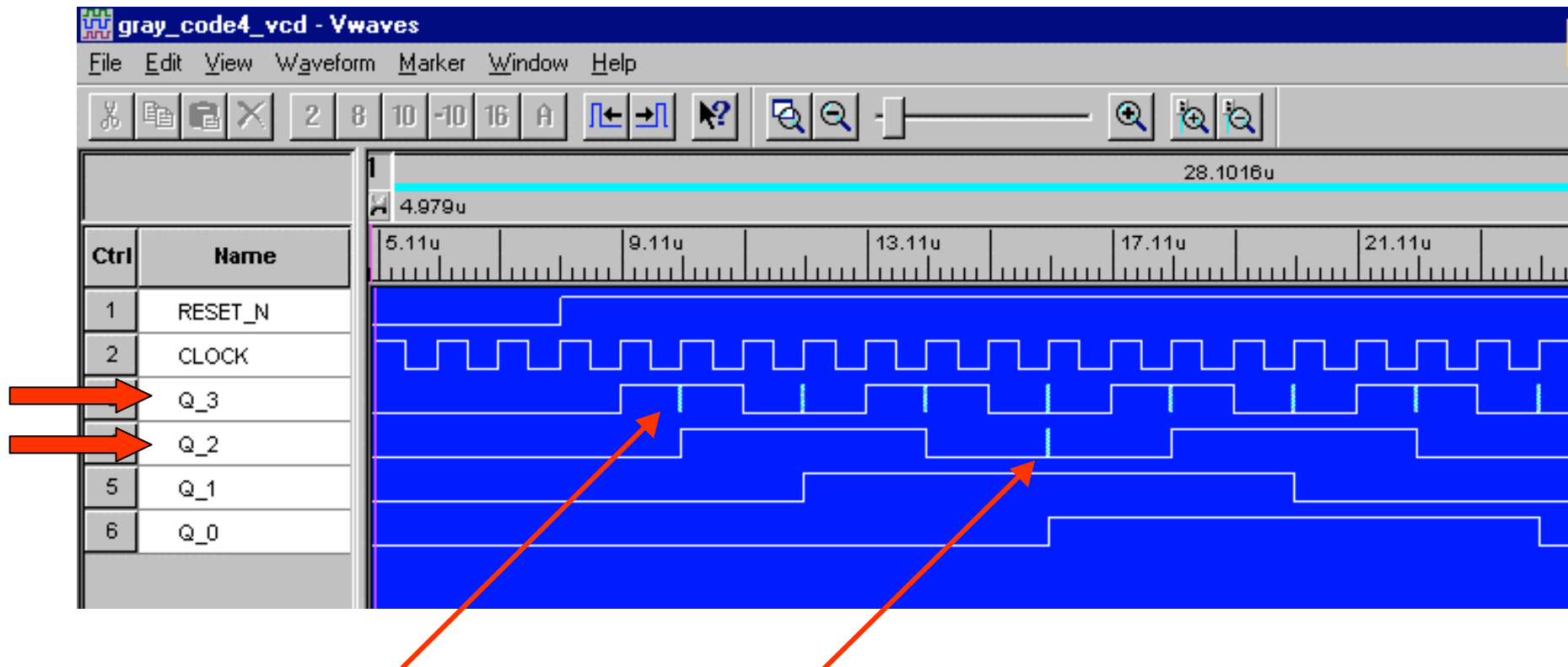
```
l reset_n  
cycle 8
```

```
h reset_n  
cycle 32
```

```
time = 9000.0ns Q=0000  
time = 10000.0ns Q=1000  
time = 11000.0ns Q=1100  
time = 12000.0ns Q=0100  
time = 13000.0ns Q=0110  
time = 14000.0ns Q=1110  
time = 15000.0ns Q=1010  
time = 16000.0ns Q=0010  
time = 17000.0ns Q=0011  
time = 18000.0ns Q=1011  
time = 19000.0ns Q=1111  
time = 20000.0ns Q=0111  
time = 21000.0ns Q=0101  
time = 22000.0ns Q=1101  
time = 23000.0ns Q=1001  
time = 24000.0ns Q=0001  
time = 25000.0ns Q=0000
```

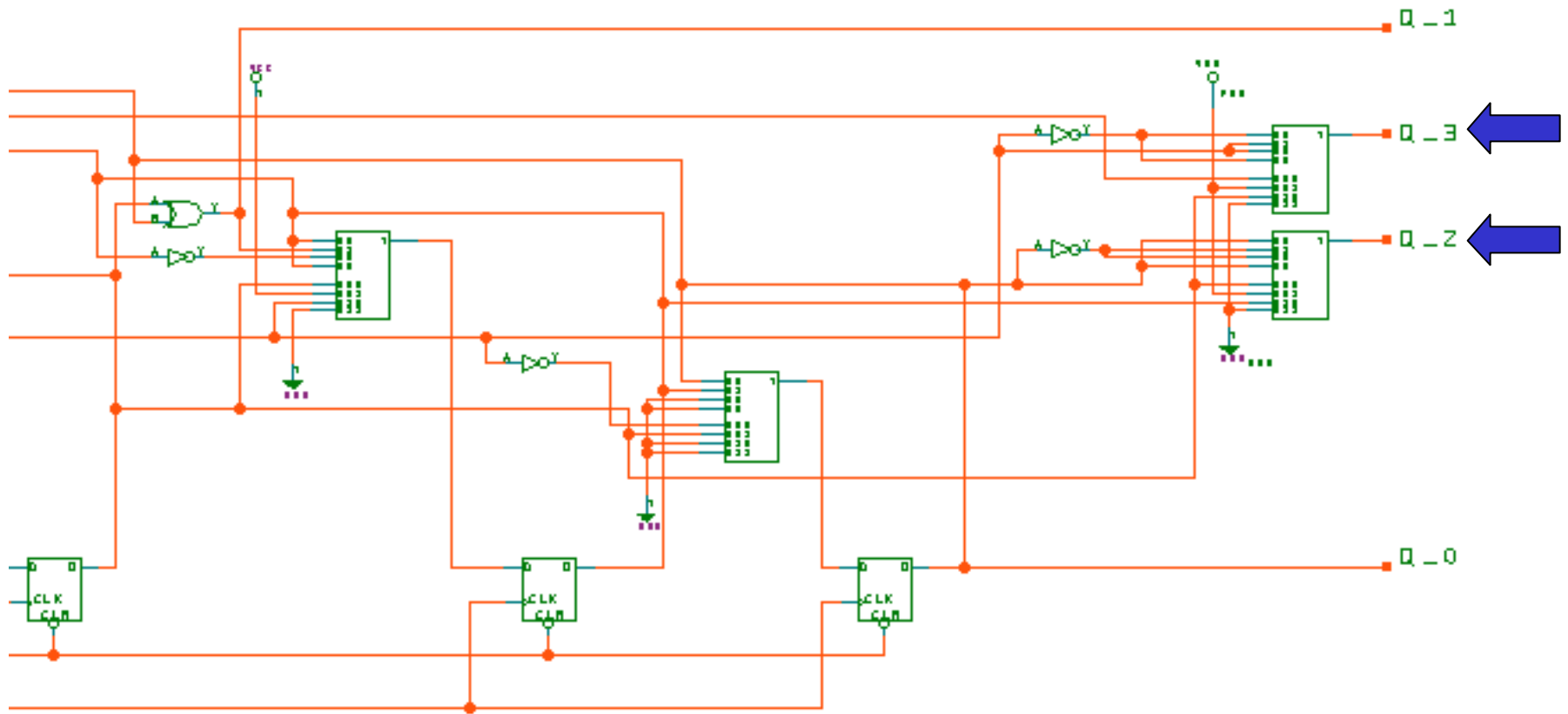


Synplicity Output for a Gray Code Sequencer - SX Target



Continue Discussion

Synplicity Output for a Gray Code Sequencer - SX Target



Outputs are not always driven by a flip-flop

Synplicity¹ Synthesis Issues

- Synthesizer ignored the command to make the state machine a Gray code and decided to make it a one-hot machine. Had to “fiddle” with the VHDL compiler settings for default FSM.
 - `Signal IQ : States;`
 - `Attribute syn_encoding of IQ : signal is "gray";`
- Output glitches!!!!!!!!!!

¹Synplify version 5.1.5

FSM Gray Codes and HDL

The Saga Continues ...

We had another engineer (HDL specialist) run the same Gray coded FSM through his version of Synplicity and what did he get ...

... Yes, as the cynic would expect, a different answer!

FSM Gray Codes and HDL

The Saga Continues ...

Here's the key part of the output listing:

```
Encoding state machine work.Gray_Code(rtl)-  
q_h.q[0:15]
```

```
original code      -> new code
```

```
000000000000000001 -> 0000
```

```
000000000000000010 -> 0001
```

```
...
```

```
100000000000000000 -> 1000
```

... So far so good!

FSM Gray Codes and HDL

The Saga Continues ...

But then ...

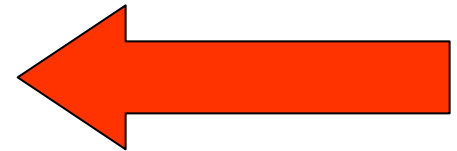
```
Replicating q_h.q[3], fanout 13 segments 2  
Replicating q_h.q[2], fanout 13 segments 2  
Replicating q_h.q[1], fanout 12 segments 2  
Replicating q_h.q[0], fanout 12 segments 2
```

Added 0 Buffers

Added 4 Cells via **replication**

Resource Usage Report of Gray_Code

```
Sequential Cells:      8 of 1080 (1%)  
    dfc1b:              8
```



FSM Gray Codes and HDL

The Saga Continues ...

```
Package Gray_Types Is
```

```
    Type States Is ( s0,  s1,  s2,  s3,  
                    s4,  s5,  s6,  s7,  
                    s8,  s9, s10, s11,  
                    s12, s13, s14, s15);
```

```
End Package Gray_Types;
```

```
library IEEE;  
use IEEE.Std_Logic_1164.all;
```

```
library Work;  
use Work.Gray_Types.all;
```

```
library synplify;  
use synplify.attributes.all;
```

```
entity Gray_Code is
```

```
    port ( Clock    : in  std_logic;  
          Reset_N  : in  std_logic;  
          Q        : out States );
```

```
end entity Gray_Code;
```

```
architecture RTL of Gray_Code is  
    attribute syn_netlist_hierarchy of RTL :  
        architecture is false;
```

```
    signal IQ          : States;  
    attribute syn_encoding of IQ : signal is "gray";
```

```
begin
```

```
    GC : process ( Clock, Reset_N )
```

```
    begin
```

```
        if ( Reset_N = '0' )
```

```
        then IQ <= s0;
```

```
        else if Rising_Edge ( Clock ) then
```

```
            case IQ is
```

```
                when s0    => IQ <= s1;
```

```
                when s1    => IQ <= s2;
```

```
                when s2    => IQ <= s3;
```

```
                when s3    => IQ <= s4;
```

```
                when s4    => IQ <= s5;
```

```
                when s5    => IQ <= s6;
```

```
                when s6    => IQ <= s7;
```

```
                when s7    => IQ <= s8;
```

```
                when s8    => IQ <= s9;
```

```
                when s9    => IQ <= s10;
```

```
                when s10   => IQ <= s11;
```

```
                when s11   => IQ <= s12;
```

```
                when s12   => IQ <= s13;
```

```
                when s13   => IQ <= s14;
```

```
                when s14   => IQ <= s15;
```

```
                when s15   => IQ <= s0;
```

```
                when others => IQ <= s0;
```

```
            end case;
```

```
        end if;
```

```
    end if;
```

```
end process GC;
```

```
Q <= IQ;
```

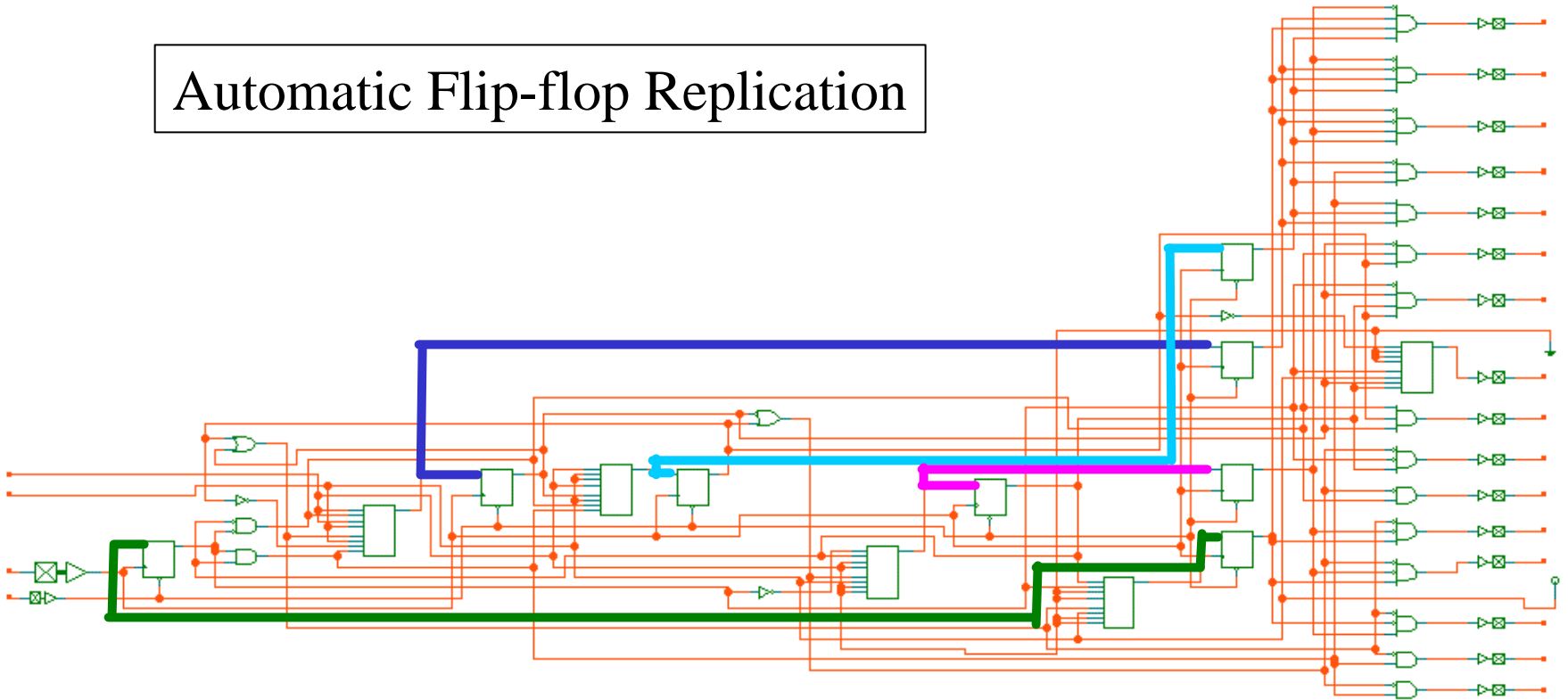
```
end architecture RTL;
```

Synplicity VHDL Compiler, version 6.2.0

FSM Gray Codes and HDL

The Saga Continues ...

Automatic Flip-flop Replication



4-Bit Gray Code

No Enumerations or FSM Optimization (1)

```
library IEEE;
use IEEE.Std_Logic_1164.all;
entity graycntr_lookup is
  port ( Clk      : in  std_logic;
        Reset_N  : in  std_logic;
        Q        : out std_logic_vector(3 downto 0));
end entity graycntr_lookup;

architecture RTL of graycntr_lookup is
  signal IQ : std_logic_vector(3 downto 0);
```

```
begin
  GC : process (Clk, Reset_N)
  begin
    if ( Reset_N = '0' )
    then IQ <= "0000";
    else if Rising_Edge ( Clk ) then
      case IQ is
        when "0000" => IQ <= "0001";
        when "0001" => IQ <= "0011";
        when "0011" => IQ <= "0010";
        when "0010" => IQ <= "0110";
        when "0110" => IQ <= "0111";
        when "0111" => IQ <= "0101";
        when "0101" => IQ <= "0100";
        when "0100" => IQ <= "1100";
        when "1100" => IQ <= "1101";
        when "1101" => IQ <= "1111";
        when "1111" => IQ <= "1110";
        when "1110" => IQ <= "1010";
        when "1010" => IQ <= "1011";
        when "1011" => IQ <= "1001";
        when "1001" => IQ <= "1000";
        when "1000" => IQ <= "0000";
        when others => IQ <= "0000";
      end case;
    end if;
  end if;
end process GC;
Q <= IQ;
end architecture RTL;
```

- No enumerations
- Synplicity VHDL Compiler, version 6.2.0

4-Bit Gray Code

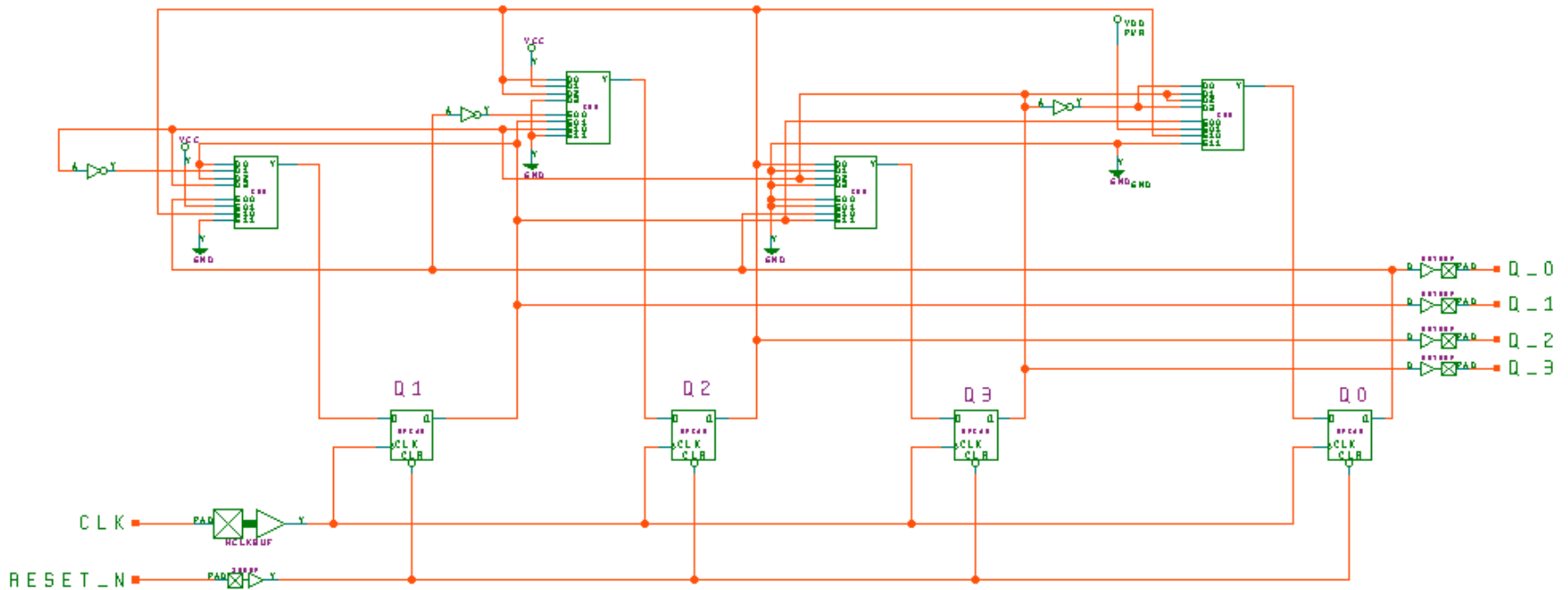
No Enumerations or FSM Optimization (1)

| | | | |
|------------------|-----------|--------|----|
| time = 8000.0ns | RESET_N=0 | Q=0000 | |
| time = 9000.0ns | RESET_N=1 | Q=0000 | 0 |
| time = 10000.0ns | RESET_N=1 | Q=0001 | 1 |
| time = 11000.0ns | RESET_N=1 | Q=0011 | 2 |
| time = 12000.0ns | RESET_N=1 | Q=0010 | 3 |
| time = 13000.0ns | RESET_N=1 | Q=0110 | 4 |
| time = 14000.0ns | RESET_N=1 | Q=0111 | 5 |
| time = 15000.0ns | RESET_N=1 | Q=0101 | 6 |
| time = 16000.0ns | RESET_N=1 | Q=0100 | 7 |
| <hr/> | | | |
| time = 17000.0ns | RESET_N=1 | Q=1100 | 8 |
| time = 18000.0ns | RESET_N=1 | Q=1101 | 9 |
| time = 19000.0ns | RESET_N=1 | Q=1111 | 10 |
| time = 20000.0ns | RESET_N=1 | Q=1110 | 11 |
| time = 21000.0ns | RESET_N=1 | Q=1010 | 12 |
| time = 22000.0ns | RESET_N=1 | Q=1011 | 13 |
| time = 23000.0ns | RESET_N=1 | Q=1001 | 14 |
| time = 24000.0ns | RESET_N=1 | Q=1000 | 15 |
| time = 25000.0ns | RESET_N=1 | Q=0000 | 0 |



4-Bit Gray Code

No Enumerations or FSM Optimization (1)



Flip-flop outputs routed directly to outputs.

References

- “Origins of the Binary Code,” F. G. Heath, *Scientific American*, August 1972, pp. 76-83