4-bit Counter by using JK Flip-Flops

David

Nov. 27, 2025

勞動部勞動力發展署 WORKFORCE DEVELOPMENT AGENCY, MINISTRY OF LABOR
NYCU 國立陽明交通大學
NYCU LASER SYSTEM RESEARCH CENTER 陽明交大雷射系統研究中心

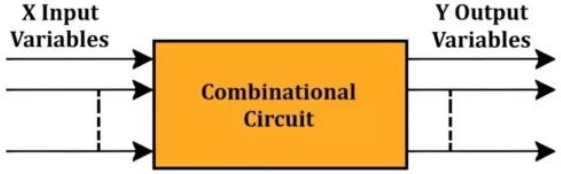
Contents

- **Sequential Logic Circuits**
  - SR Latch and JK Flip-Flop
- **Simulation**
  - LTspice and R-2R DAC
  - ModelSim
- **Experiment**
  - Altera DE2-115 and switch debouncing

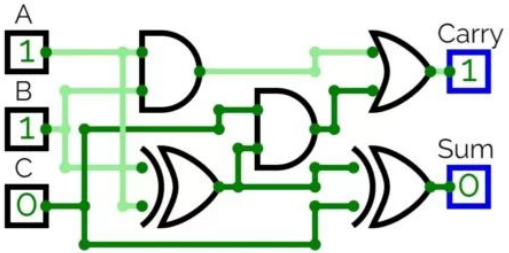
Sequential Logic Circuits

- Sequential Logic
- SR Latch
- JK Flip-Flop

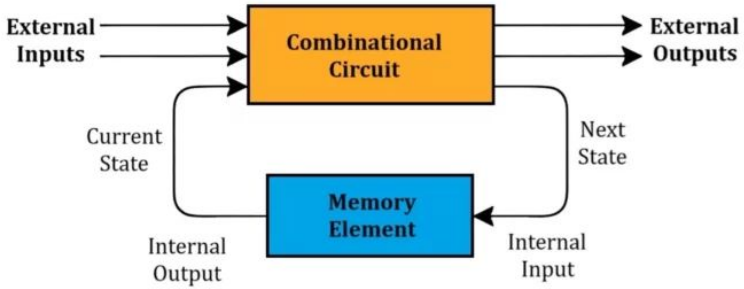
Combinational & Sequential Circuits



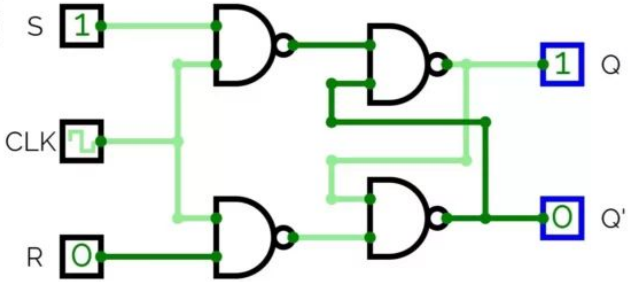
Combinational Circuit



Full Adder



Sequential Circuit



SR Flip Flop

Source:
https://www.hackatronic.com/difference-between-combinational-and-sequential-circuits/

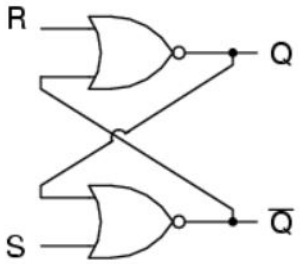
Combinational & Sequential Circuits

Combinational circuits compute functions where the output is a direct function of present inputs only; they contain no storage elements and have no notion of past inputs. **Sequential circuits** include memory elements (flip-flops, latches) so outputs depend on both current inputs and previously stored states, making them inherently time-dependent and stateful.

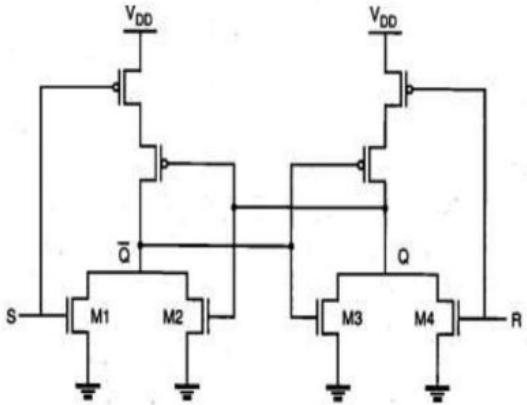
Attribute	Combinational	Sequential
Output depends on	Current inputs only	Current inputs and past state
Contains memory	No	Yes (flip-flops, latches)
Time behavior	Time-independent	Time-dependent (clocked/asynchronous)
Typical use	Arithmetic, logic functions	Counters, controllers, registers
Verification focus	Boolean correctness	State transitions and timing

(NOR-based) SR Latch

- A NOR-based S-R latch is the simplest **bistable** circuit built from two cross-coupled NOR gates; it stores a single binary state and is changed by the **Set (S)** and **Reset (R)** inputs.
- Key behavior:
  - S=1 sets Q=1,
  - R=1 resets Q=0,
  - and S=R=1 is an invalid/forbidden condition.



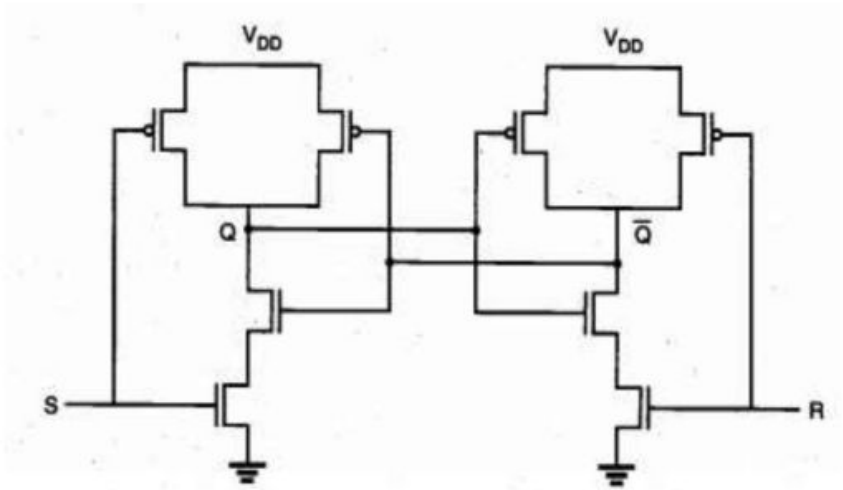
S	R	Q	$\overline{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0



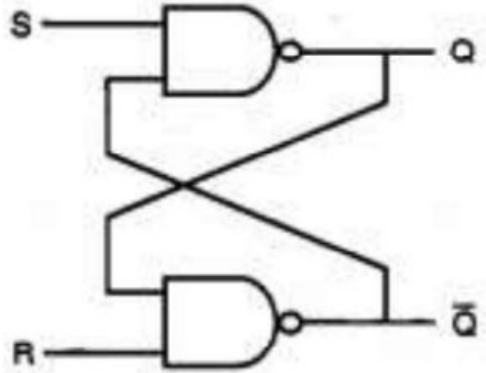
S	R	Q_{n+1}	$\overline{Q_{n+1}}$	Operation
V_{OH}	V_{OL}	V_{OH}	V_{OL}	M1 and M2 on, M3 and M4 off
V_{OL}	V_{OH}	V_{OL}	V_{OH}	M1 and M2 off, M3 and M4 on
V_{OL}	V_{OL}	V_{OH}	V_{OL}	*M*1 and M4 off, M2 on, or
V_{OL}	V_{OL}	V_{OL}	V_{OH}	M1 and M4 off, M3 on

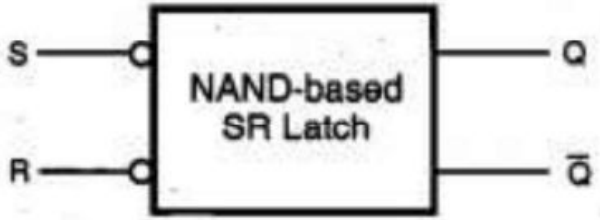
Source:
https://www.allaboutcircuits.com/textbook/digital/chpt-10/s-r-latch/
https://www.tutorialspoint.com/vlsi_design/vlsi_design_sequential_mos_logic_circuits.htm
https://course.cutm.ac.in/wp-content/uploads/2020/06/Lect28_DIC.pdf

(NAND-based) SR Latch



S	R	Q_{n+1}	$\overline{Q_{n+1}}$	Operation
0	0	1	1	not allowed
0	1	1	0	set
1	0	0	1	reset
1	1	Q_n	$\overline{Q_n}$	hold

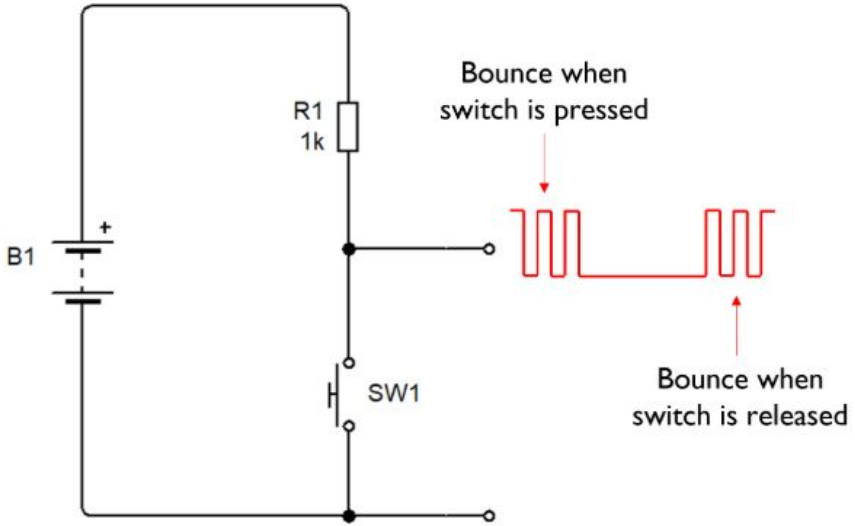




NAND SR latch responds to active low input signals whereas NOR SR latch responds to active high input signals.

Source:
https://course.cutm.ac.in/wp-content/uploads/2020/06/Lect28_DIC.pdf

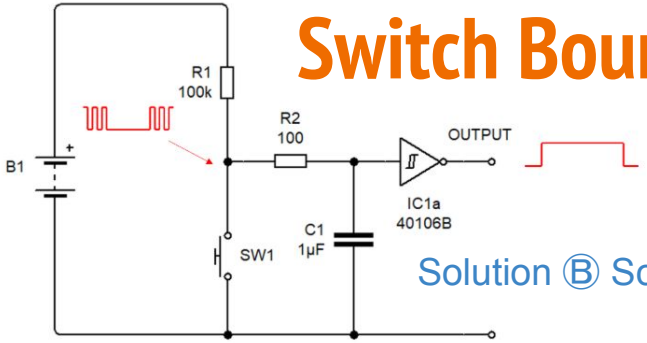
Switch Bounce



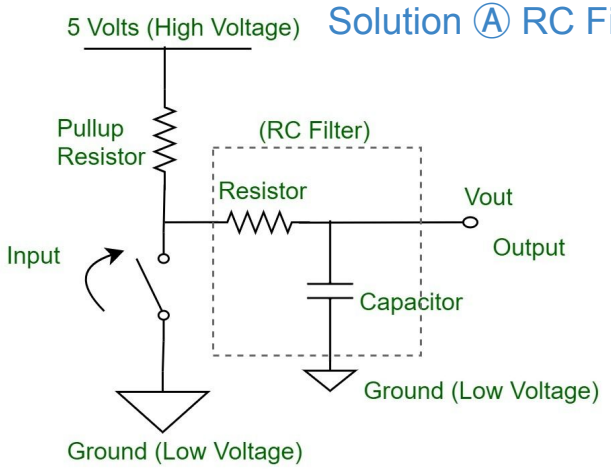
- Mechanical contacts physically bounce when they close or open, producing a train of short pulses that digital logic can interpret as many presses.
- Typical failure modes and what to watch for:
  - **Both S and R briefly asserted** during a transition can produce an invalid or metastable state in some SR implementations (especially simple NAND/NOR cross-coupled latches).
  - **Glitches from slow edges** can still trigger logic if the latch inputs are noisy or if there's no hysteresis.
  - **Power-up undefined state** — an SR latch may power up in either state unless you provide a defined reset.

Source:
https://www.clarvis.co.uk/Electronics/Practical-Circuits/Switch-bounce.html

Switch Bounce: Solution



Solution Ⓑ Schmitt Trigger



Solution Ⓐ RC Filter

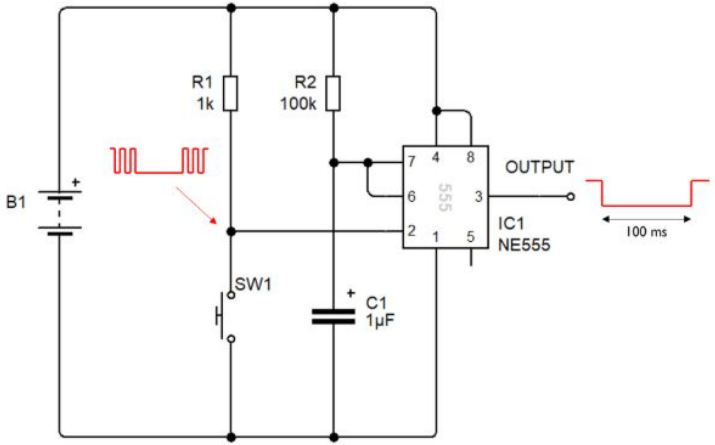
Ⓐ Add simple **RC** timing before the SR latch

Ⓑ Use a **Schmitt** trigger first

Ⓒ Use a **one-shot** or **monostable** to generate a single clean pulse

Ⓓ Replace SR latch with a **D flip-flop** sampled by a periodic clock

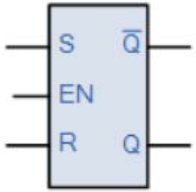
Ⓔ Software debounce: Read the raw input and require it to be **stable** for a **fixed time** (e.g., 5–50 ms) before accepting the change.

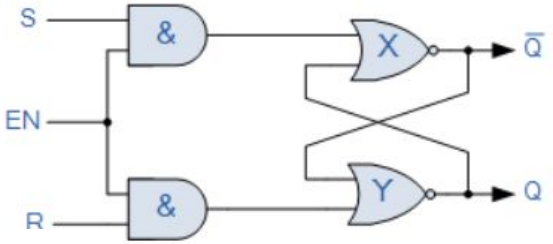


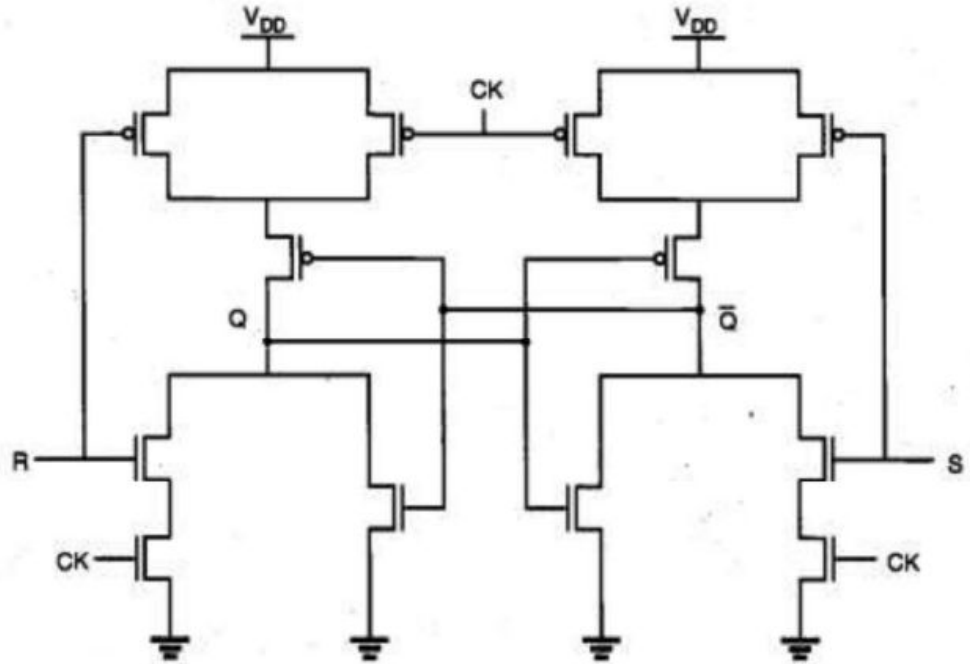
Solution Ⓒ 555 Monostable Circuit

Source:
https://www.geeksforgeeks.org/digital-logic/switch-debounce-in-digital-circuits/
https://www.clarvis.co.uk/Electronics/Practical-Circuits/Switch-bounce.html

(NOR-based) Gated SR Flip-flop





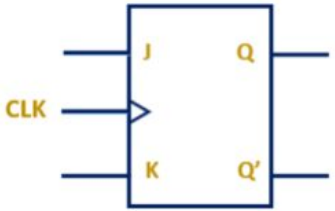


Source:
https://www.electronics-tutorials.ws/sequential/seq_1.html
https://course.cutm.ac.in/wp-content/uploads/2020/06/Lect28_DIC.pdf

(NOR-based) JK Flip-Flop

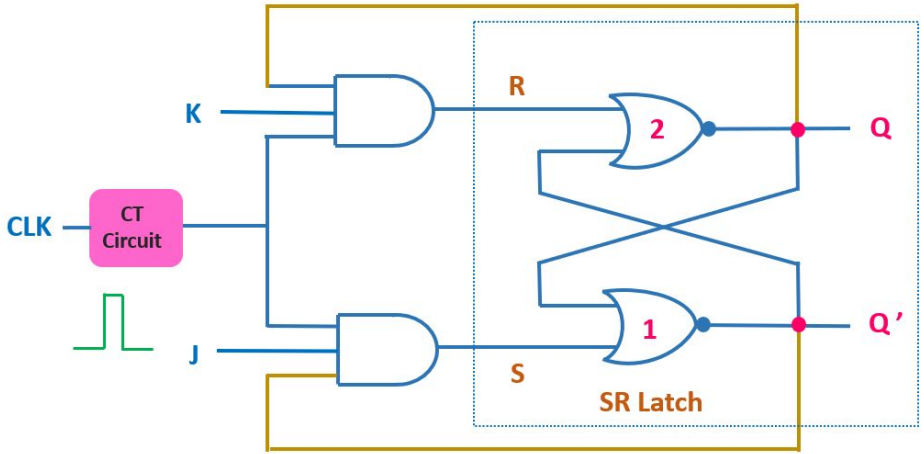
The JK flip-flop is a gated SR flip-flop with an additional clock input circuitry that prevents the invalid output condition when both inputs S and R are equal to logic level “1”.

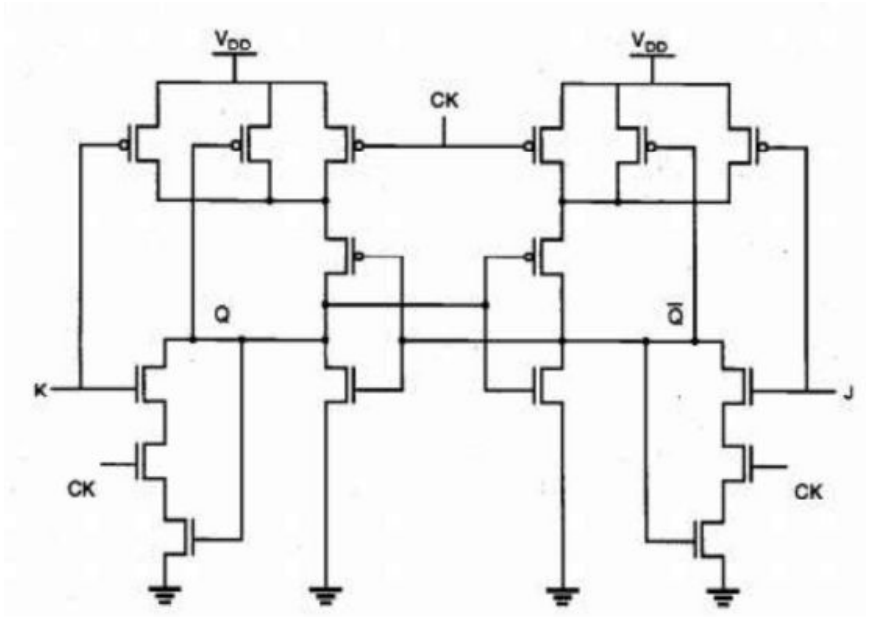
Symbol



Truth Table

CLK	J	K	Q n+1
↑	0	0	Q n
↑	0	1	0
↑	1	0	1
↑	1	1	Q n’



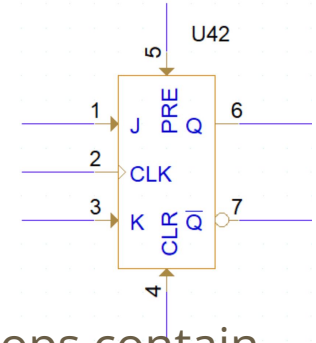


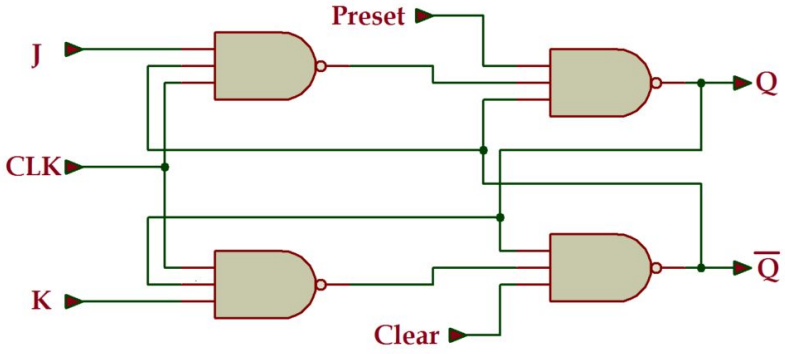
Source:

https://course.cutm.ac.in/wp-content/uploads/2020/06/Lect28_DIC.pdf

https://www.allaboutelectronics.org/jk-flip-flop-explained-race-around-condition-in-jk-flip-flop-jk-flip-flop-truth-table-excitation-table-and-timing-diagram/

JK Flip-Flop



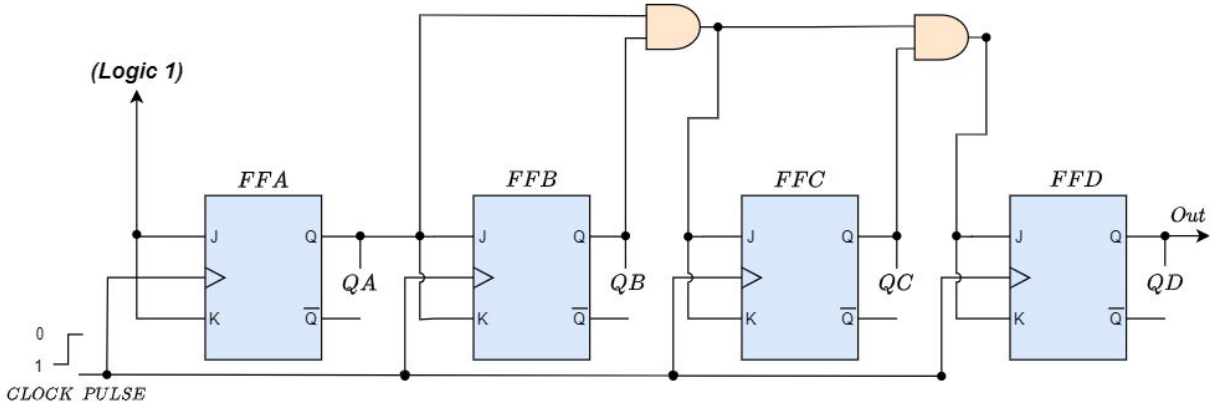


- Sometimes JK flip-flops contain an additional set and reset pin. The set and reset pins work as preset and clear to **initialize** the flip-flop.
- Preset (**PRE**) and Clear (**CLR**): Asynchronous inputs used to directly set or reset the flip-flop regardless of the clock.

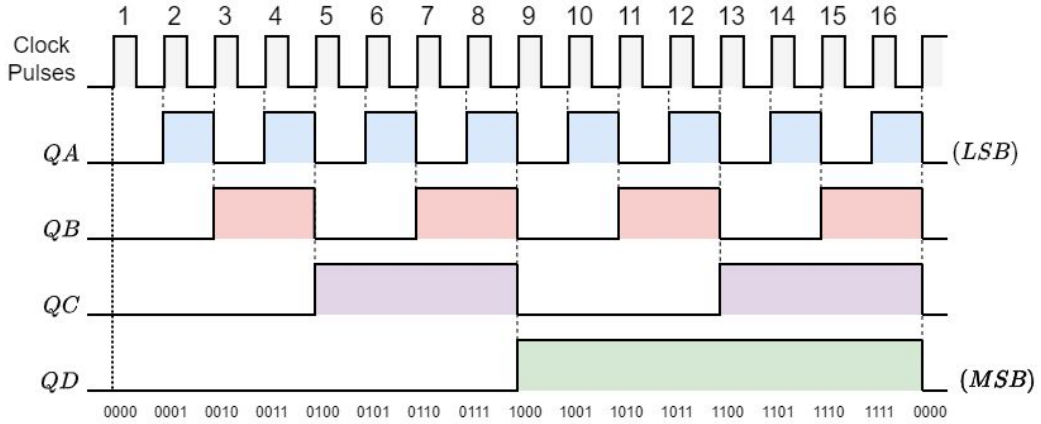
Preset	Clear	CLK	J	K	Q	Q(not)
X	1	X	X	X	0	1
1	X	X	X	X	1	0
0	0	↑	1	0	1	0
0	0	↑	0	1	0	1
0	0	↑	1	1	1*	0
0	0	↑	0	0	1~	0

Source:
https://www.ema-eda.com/ema-resources/blog/jk-flip-flop-spice-model-explained/
https://compraco.com.br/en/blogs/tecnologia-e-desenvolvimento/tutorial-vhdl-17-projete-um-flip-flop-jk-com-preset-e-clear-usando-vhdl

4-bit Synchronous Up Counter



Source: https://www.electronics-lab.com/article/synchronous-counter/



Simulation

- LTspice and DAC
- ModelSim

Simulation

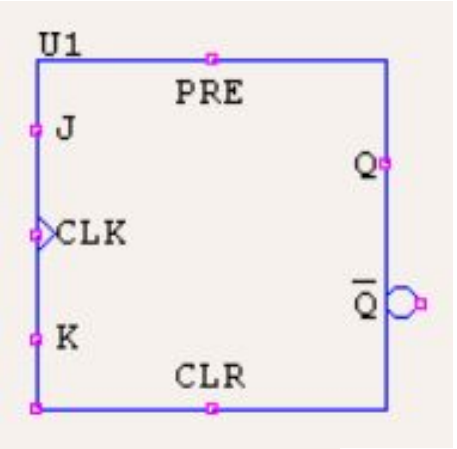
- **LTspice and DAC**
- ModelSim

LTspice

- **SPICE**, standing for "**Simulation Program with Integrated Circuit Emphasis**," is a computer program that simulates the behavior of electronic circuits, enabling engineers to analyze and test them virtually before building them physically.
- **LTspice** is a free **SPICE simulator** software from **Analog Devices** that is used to design, test, and analyze electronic circuits before building physical prototypes.

0101 Library

http://ltspicegoodies.ltwiki.org/0101.php



JKFLOP

J-K flip-flop.

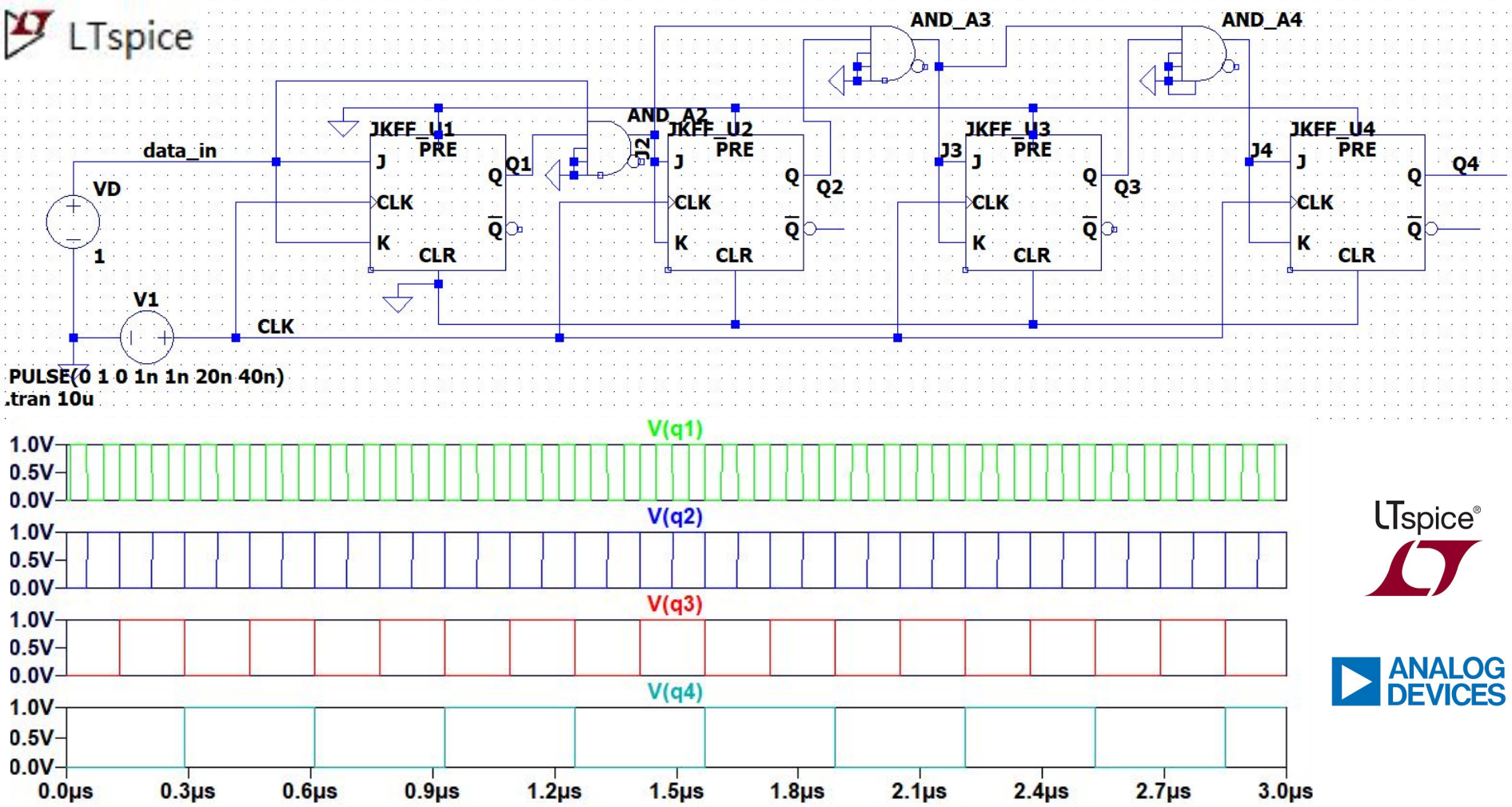
- 'J', 'K', 'CLK' are floating inputs.
- 'PRE', 'CLR' are the set and rese pins, respectively, with *1GΩ* internal pulldown.
- The ground pin is 'RTN' and cannot be left floating!

Parameters:

Vhigh,Vlow	[*V*]	Output logic levels, defaults *<1,0>*

Hidden:

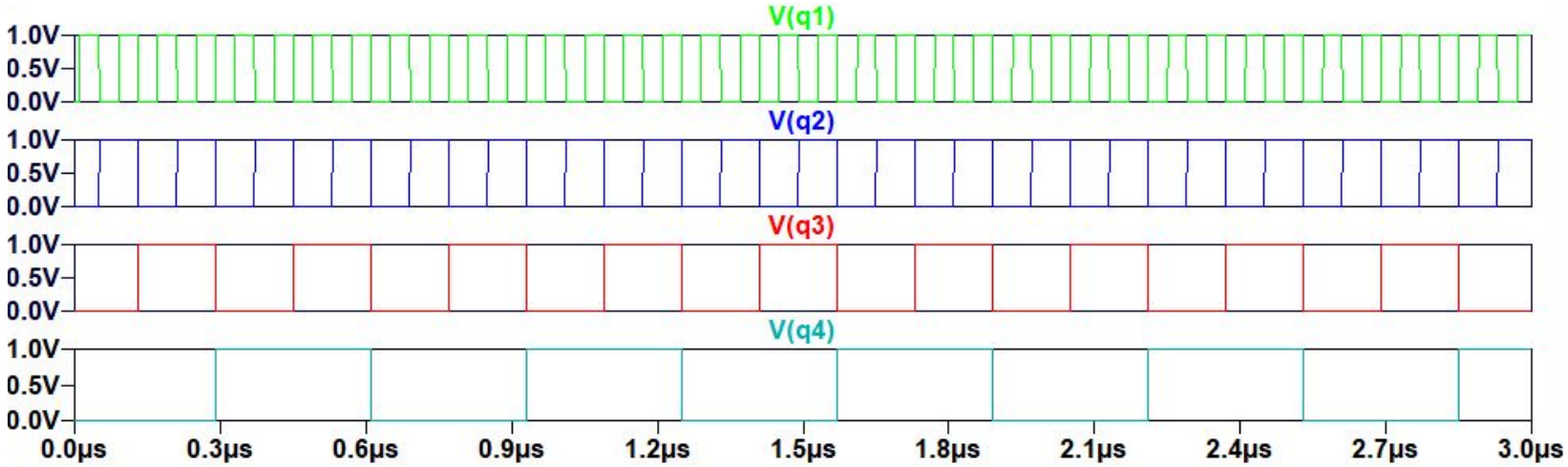
td	[*s*]	LTspice's specific for A-devices (should not be set null!), default *10n*


LTspice
AND_A3
AND_A4
AND_A2
JKFF_U1
JKFF_U2
JKFF_U3
JKFF_U4
PRE
J
CLK
K
CLR
Q
Q1
Q2
Q3
Q4
J2
J3
J4
data_in
VD
1
V1
CLK
PULSE(0 1 0 1n 1n 20n 40n)
.tran 10u
V(q1)
V(q2)
V(q3)
V(q4)
1.0V
0.5V
0.0V
0.0µs
0.3µs
0.6µs
0.9µs
1.2µs
1.5µs
1.8µs
2.1µs
2.4µs
2.7µs
3.0µs
LTspice
ANALOG DEVICES

4-bit Counter in LTspice

Discussion (i)

Flip-Flop Toggling in V(q1):
The least significant bit (LSB) flip-flop toggles its state with every active edge.



LTspice®

ANALOG DEVICES

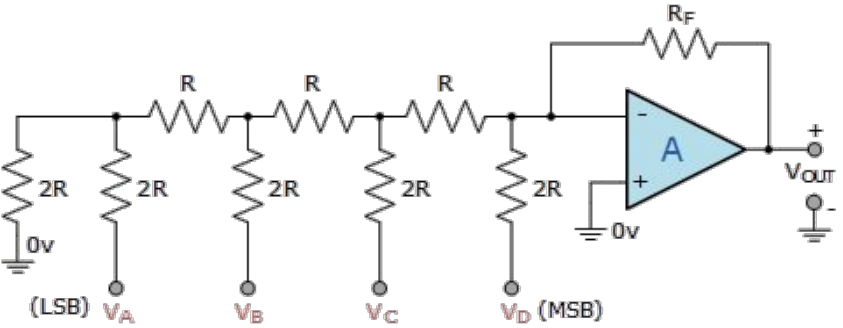
Digital-to-Analog Converter (DAC)

- To observe an analog signal, I input a digital signal (a series of 0s and 1s) into the Digital-to-Analog Converter (DAC). The DAC then converts this digital data into a corresponding analog voltage or current, which is the output signal.
- Types of DACs
  - **R-2R ladder and weighted-resistor DACs**
  - **Current-steering and flash DACs**
  - **Sigma-Delta (ΔΣ) DACs**
  - **PWM (pulse-width modulation) DACs**

Types of DACs

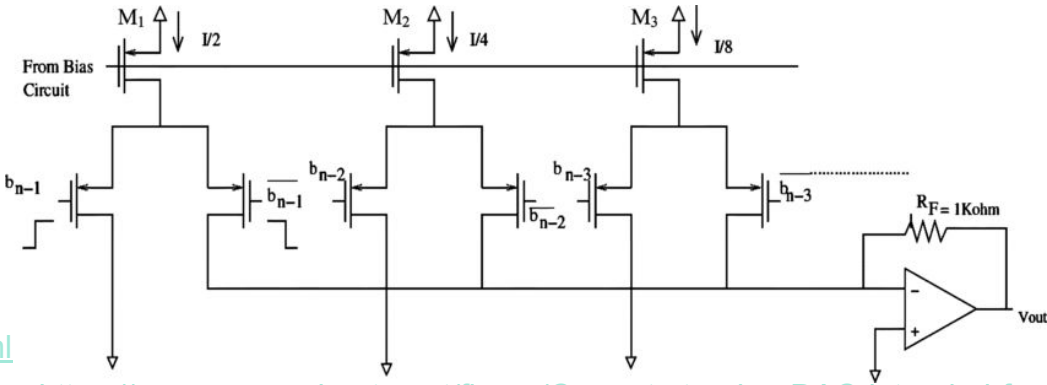
- **R-2R ladders** implement binary weighting with only two resistor values, making them simple and area-efficient for moderate resolutions, but they rely on tight resistor matching and low switch error for accuracy.
- **Current-steering (flash) DACs** use many parallel current sources switched in one clock cycle to achieve very high conversion rates; they are the go-to choice when speed is the primary requirement, at the cost of higher power and silicon area.
- **Sigma-delta (ΔΣ) DACs** trade instantaneous bandwidth for in-band performance by oversampling the signal and shaping quantization noise out of the band of interest; a digital filter and analog reconstruction recover a high-resolution analog output, making them ideal for audio and precision low-frequency applications.
- **PWM DACs** are commonly used in microcontrollers: the desired amplitude is encoded as pulse duty cycle and a simple RC or active low-pass filter smooths the waveform; this approach is inexpensive and flexible but requires careful filtering and clock stability to control ripple and effective resolution.

R-2R DAC

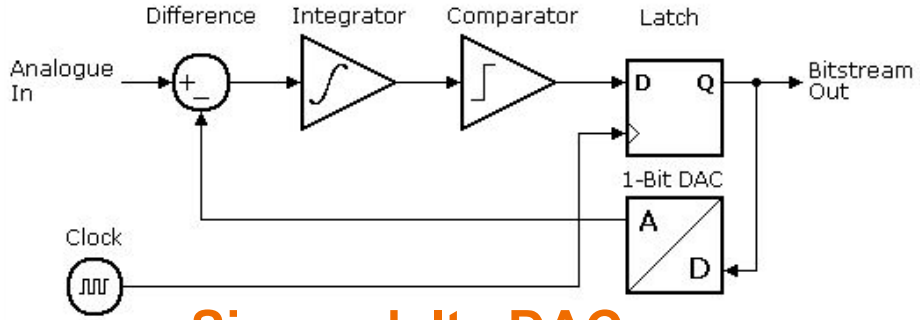


https://www.electronics-tutorials.ws/combination/r-2r-dac.html

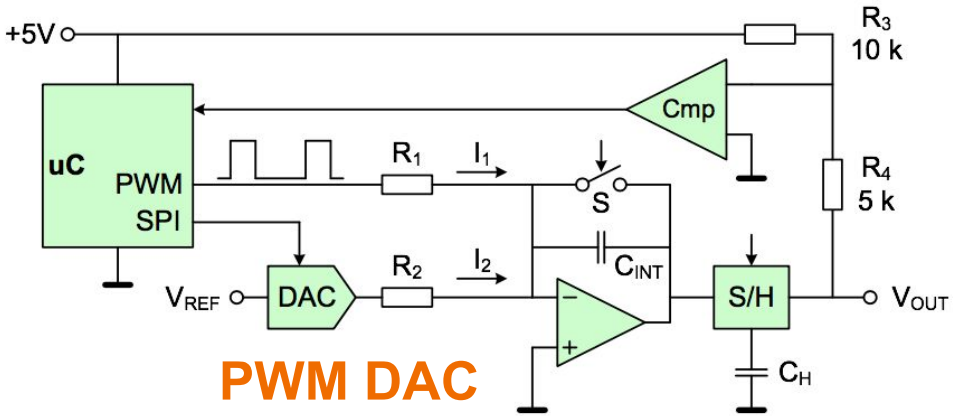
Current steering DAC



https://www.researchgate.net/figure/Current-steering-DAC-intended-for-high-speed-applications-A-six-bit-DAC-has-been_fig12_3074734



Sigma-delta DAC



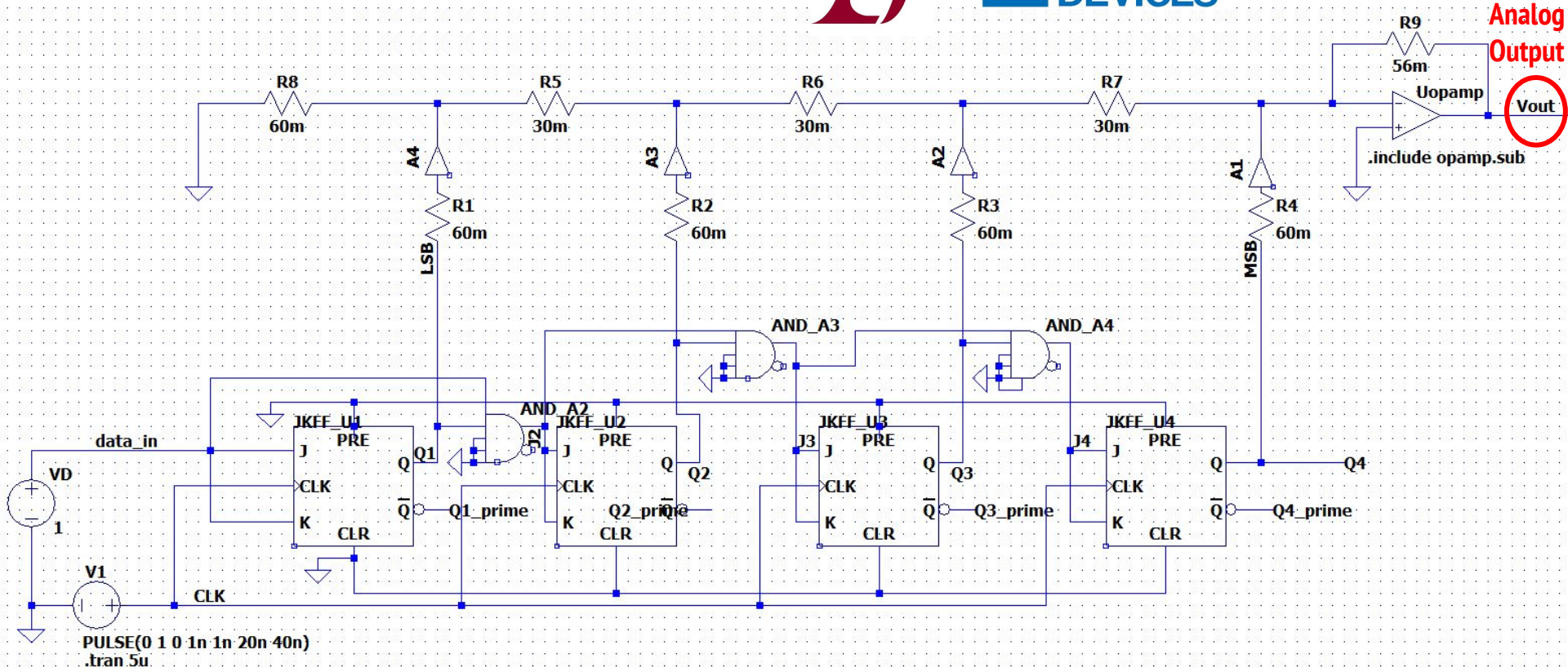
PWM DAC

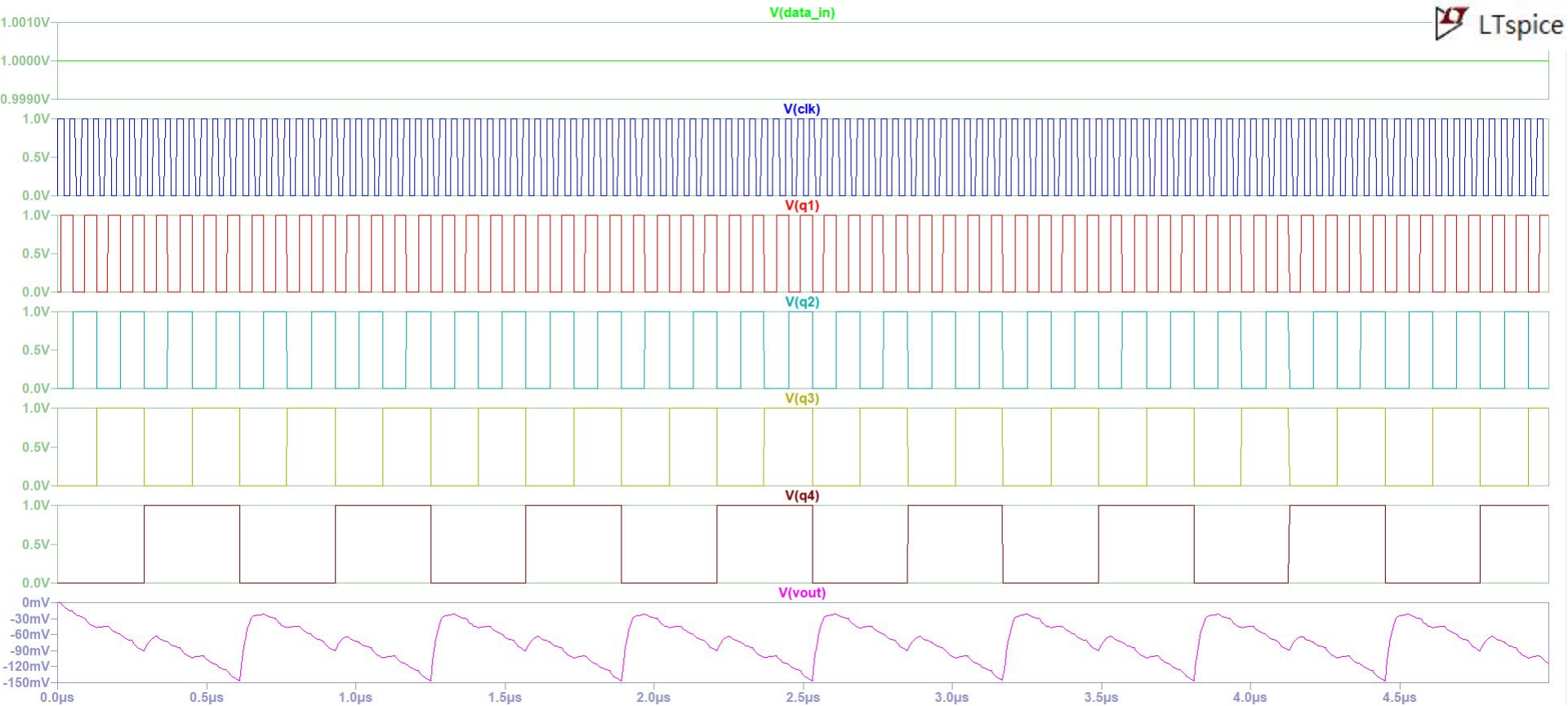
https://www.moon-audio.com/blogs/expert-advice/dac-types-deciphered

https://www.edntaiwan.com/20200210ta31-pwm-dac-settles-in-one-period-of-the-pulse-train/

Architecture	Key concept	Typical speed	Typical resolution	Strengths / Limitations
R-2R ladder	resistor network using only R and 2R values to weight bits	low–moderate	up to ~16 bits (practical)	**Strengths:** simple, compact; **Limitations:** resistor matching limits high precision
Current-steering / Flash	parallel switched current sources produce output in one cycle	very high (GHz class)	low–moderate (6–14 bits common)	**Strengths:** ultra-fast; **Limitations:** area/power scale with resolution
Sigma-Delta (ΔΣ)	oversampling + noise shaping; digital modulator + analog reconstruction	low–moderate (bandlimited)	very high effective resolution (24+ bits in band)	**Strengths:** excellent linearity and SNR in band; **Limitations:** latency, not for wideband RF
PWM DAC	encode amplitude as duty cycle of pulse train; low-pass filter reconstructs analog	low (timer limited)	limited by timer and filter (8–16 bits practical)	**Strengths:** very low cost, MCU-friendly; **Limitations:** needs filtering, switching ripple and jitter

4-bit Counter + R-2R DAC





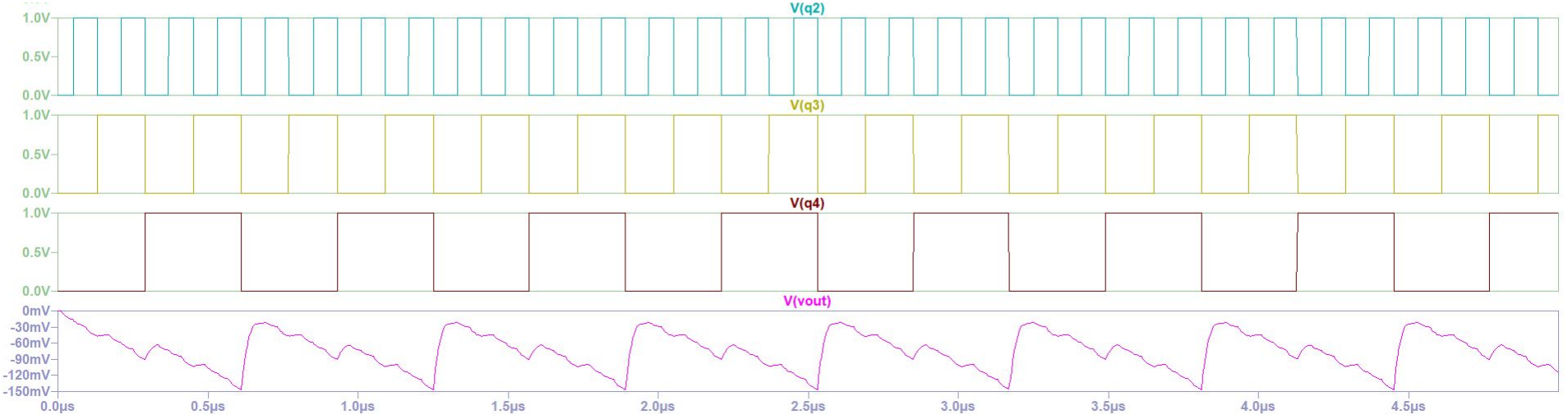
- The binary set {q1, q2, q3, q4} is collected and now represented by the analog output **Vout**.
- When we count from 1 to 15, the reading (**Vout**) will change from -15 to -150 mV.

4-bit Counter in LTspice

LTspice

Discussion (ii)

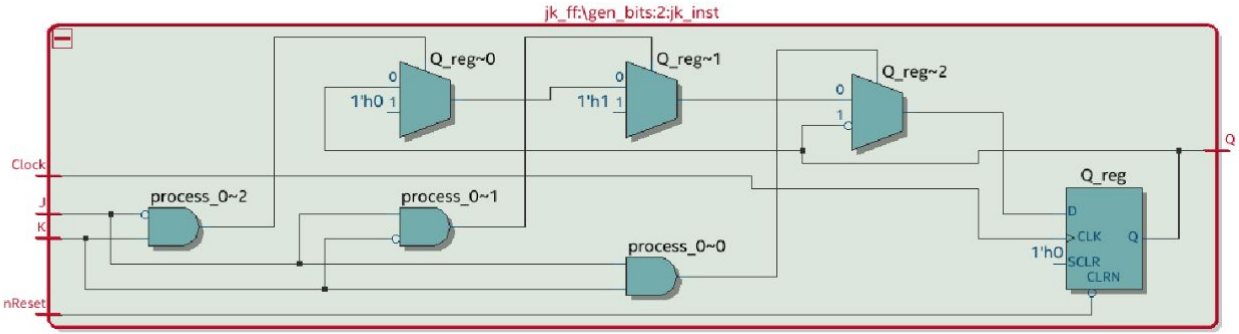
Rollover: When the 4-bit counter reaches its maximum value of 1111 (decimal 15) and receives another increment request, the MSB (the '1' in 10000) is discarded, and the counter rolls over, resetting its value back to 0000.



Simulation

- LTspice and DAC
- **ModelSim**

Write VHDL to implement JK Flip-Flops



Netlist of **jk_ff**

```
24  library ieee;
25  use ieee.std_logic_1164.all;
26
27  entity jk_ff is
28      port (
29          J      : in  std_logic;
30          K      : in  std_logic;
31          Clock  : in  std_logic;
32          nReset : in  std_logic;
33          Q      : out std_logic
34      );
35  end entity jk_ff;
36
37  architecture rtl of jk_ff is
38      signal Q_reg : std_logic;
39  begin
40      process(nReset, Clock)
41      begin
42          if nReset = '0' then
43              Q_reg <= '0';
44          elsif rising_edge(Clock) then
45              if (J = '1' and K = '1') then
46                  Q_reg <= not Q;
47              elsif (J = '1' and K = '0') then
48                  Q_reg <= '1';
49              elsif (J = '0' and K = '1') then
50                  Q_reg <= '0';
51              end if;
52          end if;
53            Q <= Q_reg;
54      end process;
55  end architecture rtl;
```

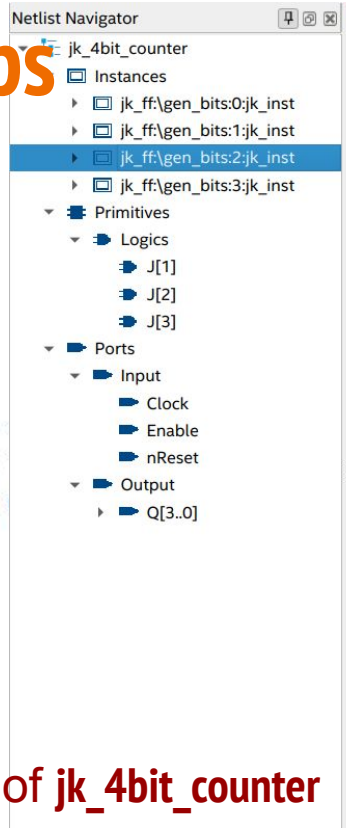
Write VHDL to 4-bit Counter by using JK Flip-Flops

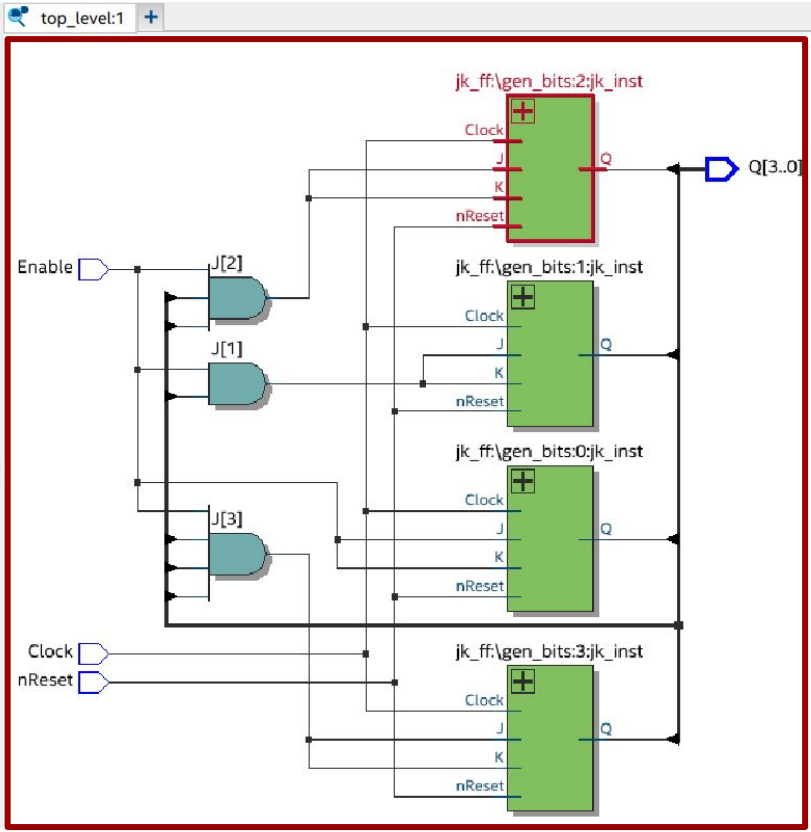
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity jk_4bit_counter is
    port (
        Clock  : in  std_logic;
        nReset : in  std_logic;  -- active low reset
        Enable : in  std_logic;  -- count when '1'
        Q      : out std_logic_vector(3 downto 0)
    );
end entity jk_4bit_counter;
```

```vhdl
architecture rtl of jk_4bit_counter is
    -- internal signals for J and K
    signal q_reg : std_logic_vector(3 downto 0) := (others => '0');
    signal J     : std_logic_vector(3 downto 0);
    signal K     : std_logic_vector(3 downto 0);
begin

    -- J assignments for toggle-style ripple counter
    J(0) <= Enable;
    J(1) <= Enable and q_reg(0);
    J(2) <= Enable and q_reg(0) and q_reg(1);
    J(3) <= Enable and q_reg(0) and q_reg(1) and q_reg(2);

    -- K equals J for toggle behavior
    K <= J;

    -- Instantiate four jk_ff components using a generate loop
    gen_bits : for i in 0 to 3 generate
        jk_inst : entity work.jk_ff
            port map (
                J      => J(i),
                K      => K(i),
                Clock  => Clock,
                nReset => nReset,
                Q      => q_reg(i)
            );
    end generate gen_bits;

    -- output assignment
    Q <= q_reg;

end architecture rtl;
```

Write VHDL to 4-bit Counter by using JK Flip-Flops

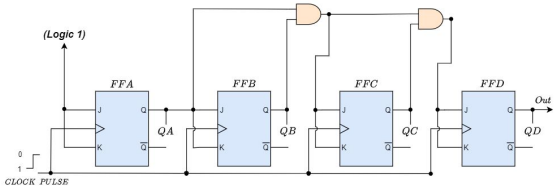
```
28    library ieee;
29    use ieee.std_logic_1164.all;
30
31    entity jk_4bit_counter is
32        port (
33            Clock  : in  std_logic;
34            nReset : in  std_logic;  -- active low
35            Enable : in  std_logic;  -- count when
36            Q      : out std_logic_vector(3 downto
37        );
38    end entity jk_4bit_counter;
```





Netlist of **jk_4bit_counter**

VHDL testbench



Intel® Quartus® Prime
ModelSim

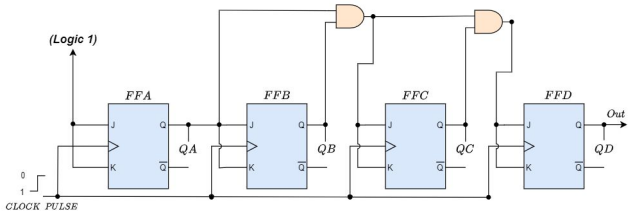
```
35  library ieee;
36  use ieee.std_logic_1164.all;
37  use ieee.numeric_std.all;
38
39  entity tb_jk_4bit_counter is
40  end entity tb_jk_4bit_counter;
41
42  architecture sim of tb_jk_4bit_counter is
43      -- DUT signals
44      signal Clock  : std_logic := '0';
45      signal nReset : std_logic := '1';  -- active low
46      signal Enable : std_logic := '0';
47      signal Q      : std_logic_vector(3 downto 0);
48
49      constant CLK_PERIOD : time := 20 ns;  -- 50 MHz c
50  begin
51
52      -- Instantiate the device under test
53      uut: entity work.jk_4bit_counter
54          port map (
55              Clock  => Clock,
56              nReset => nReset,
57              Enable => Enable,
58              Q      => Q
59          );
60
61      -- Clock generator
62      clk_proc : process
63      begin
64          while true loop
65              Clock <= '0';
66              wait for CLK_PERIOD / 2;
67              Clock <= '1';
68              wait for CLK_PERIOD / 2;
69          end loop;
70      end process clk_proc;
119     -- Monitor process: print Q on each rising edge
120     monitor_proc : process(Clock)
121     begin
122         if rising_edge(Clock) then
123             report "At time " & time'image(now) & " Q = " & integer'image(to_integer
124         end if;
125     end process monitor_proc;
126
127 end architecture sim;
```

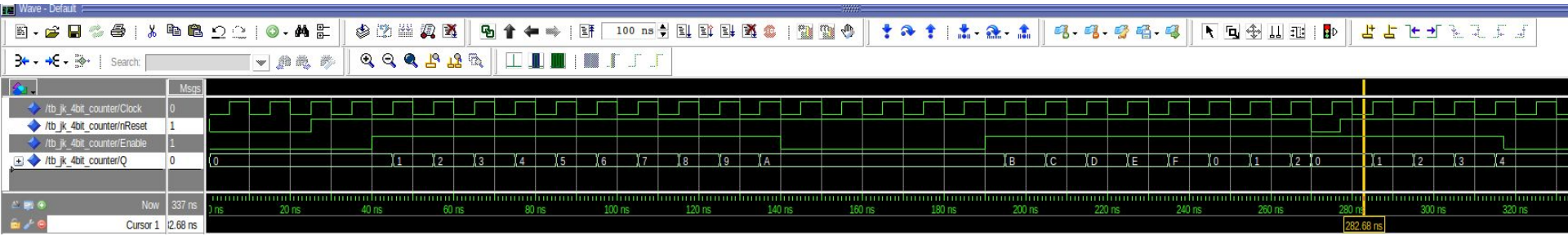
```
72      -- Stimulus process
73      stim_proc : process
74      begin
75          -- initial asynchronous reset asserted
76          nReset <= '0';
77          Enable <= '0';
78          wait for 25 ns;
79
80          -- release reset
81          nReset <= '1';
82          wait for 15 ns;
83
84          -- start counting
85          Enable <= '1';
86          report "Enable asserted, counting should start" severity note;
87
88          -- let it count for a number of clock cycles (observe wrap-around)
89          wait for 10 * CLK_PERIOD;  -- 10 clock cycles
90
91          -- disable counting (hold)
92          Enable <= '0';
93          report "Enable deasserted, counter should hold" severity note;
94          wait for 5 * CLK_PERIOD;
95
96          -- re-enable counting
97          Enable <= '1';
98          report "Enable reasserted, counting resumes" severity note;
99          wait for 8 * CLK_PERIOD;
100
101         -- assert asynchronous reset again to test clear
102         nReset <= '0';
103         report "Asserting asynchronous reset (active low)" severity note;
104         wait for 7 ns;  -- asynchronous, so can be asserted between clocks
105         nReset <= '1';
106         report "Releasing asynchronous reset" severity note;
107         wait for 4 * CLK_PERIOD;
108
109         -- final hold and finish
110         Enable <= '0';
111         wait for 2 * CLK_PERIOD;
112
113         report "End of testbench" severity note;
114         -- stop simulation
115         assert false report "Simulation finished" severity failure;
116         wait;
117     end process stim_proc;
```

twwang97

VHDL Testbench

ModelSim
Intel® Quartus® Prime





Discussion

- **Flip-Flop Toggling**: The least significant bit (LSB) flip-flop toggles its state with every active edge.
- **Rollover**: When the 4-bit counter reaches its maximum value of 1111 (decimal 15) and receives another increment request, the MSB (the '1' in 10000) is discarded, and the counter rolls over, resetting its value back to 0000.
- **n**Reset: An active-high input that, when asserted (time = 283 ns), clears the counter to zero.

Experiment

- DE2-115
- Debouncing

Write VHDL to show the value on 7-segment display

```vhdl
entity hex7seg_display is
  port(
    clk        : in  std_logic;
    rst_n      : in  std_logic;
    sum_number : in  std_logic_vector(3 downto 0);
    hex0_a     : out std_logic;
    hex0_b     : out std_logic;
    hex0_c     : out std_logic;
    hex0_d     : out std_logic;
    hex0_e     : out std_logic;
    hex0_f     : out std_logic;
    hex0_g     : out std_logic
  );
end entity;
```

```vhdl
  with digit select
    seg_pat <=
      "0000001" when "0000", -- 0: a b c d e f ON, g OFF
      "1001111" when "0001", -- 1
      "0010010" when "0010", -- 2
      "0000110" when "0011", -- 3
      "1001100" when "0100", -- 4
      "0100100" when "0101", -- 5
      "0100000" when "0110", -- 6
      "0001111" when "0111", -- 7
      "0000000" when "1000", -- 8: all segments ON
      "0000100" when "1001", -- 9
      "0001000" when "1010", -- A
      "1100000" when "1011", -- b
      "0110001" when "1100", -- C
      "1000010" when "1101", -- d
      "0110000" when "1110", -- E
      "0111000" when "1111", -- F
      "1111111" when others;  -- default: all segments OFF (blank)
```

```vhdl
    hex0_a <= seg_pat(6);
    hex0_b <= seg_pat(5);
    hex0_c <= seg_pat(4);
    hex0_d <= seg_pat(3);
    hex0_e <= seg_pat(2);
    hex0_f <= seg_pat(1);
    hex0_g <= seg_pat(0);
```

Experiment on Altera DE2-115

ISSUE:

The **switch bounce** occurs!

software/hardware debouncing: timer + 2-stage synchronizer

<software> timer filter

- stable0 holds the current debounced level. A counter cnt increments only when the synchronized input sync1 differs from stable0. If sync1 equals stable0, the counter is cleared. When cnt reaches THRESH-1 , stable0 is updated to sync1 and the counter resets.

Timing and THRESH

The constant `THRESH` is computed as:

$$\text{THRESH} = \frac{\text{CLOCK\_FREQ\_HZ}}{1000} \cdot \text{DEBOUNCE\_MS}$$

For the default generics (50 MHz clock, 20 ms debounce) that equals:

$$\text{THRESH} = \frac{50\,000\,000}{1000} \cdot 20 = 1\,000\,000 \text{ clock cycles.}$$

So the input must remain steady for ~1,000,000 clock cycles (~20 ms) before the change is accepted.

<hardware> two-stage synchronizer

- sync0 and sync1 sample raw_btn on consecutive clocks to avoid metastability and bring the asynchronous input into the clock domain.

Write VHDL to implement button_debouncer

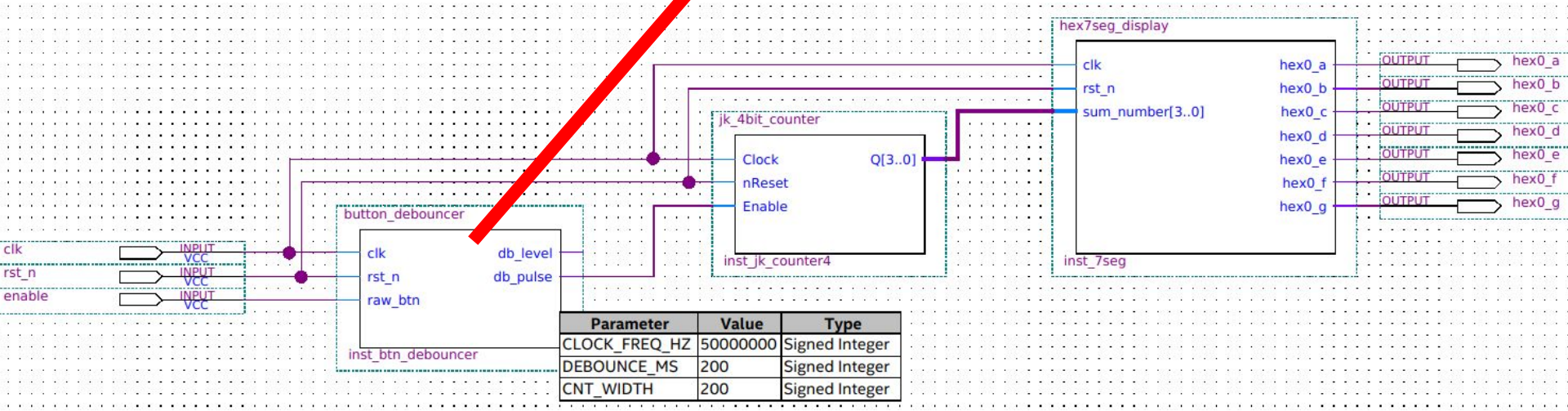
```vhdl
entity button_debouncer is
  generic (
    CLOCK_FREQ_HZ : natural := 50_000_000;
    DEBOUNCE_MS   : natural := 20;
    CNT_WIDTH     : natural := 20
  );
  port (
    clk        : in  std_logic;
    rst_n      : in  std_logic;
    raw_btn    : in  std_logic;  -- asynchr
    db_level   : out std_logic;  -- debounc
    db_pulse   : out std_logic   -- single-
  );
end entity;
--
```

```vhdl
-- 2-stage synchronizer to avoid metastability
process(clk)
begin
  if rising_edge(clk) then
    if rst_n = '0' then
      sync0 <= '0';
      sync1 <= '0';
    else
      sync0 <= raw_btn;
      sync1 <= sync0;
    end if;
  end if;
end process;
-- initial hold-off counter after reset: keep (
process(clk)
begin
  if rising_edge(clk) then
    if rst_n = '0' then
      init_cnt  <= (others => '0');
      init_done <= '0';
    else
      if init_done = '0' then
        if init_cnt = THRESH - 1 then
          init_done <= '1';
          init_cnt  <= (others => '0');
        else
          init_cnt <= init_cnt + 1;
        end if;
      end if;
    end if;
  end if;
end process;
```

```vhdl
-- debounce counter: require sync1 to
process(clk)
begin
  if rising_edge(clk) then
    if rst_n = '0' then
      stable0 <= '0';
      cnt <= (others => '0');
    elsif init_done = '0' then
      -- during initial hold-off keep
      stable0 <= '0';
      cnt <= (others => '0');
    else
      -- normal debounce operation
      if sync1 = stable0 then
        cnt <= (others => '0');
      else
        if cnt = THRESH - 1 then
          stable0 <= sync1;
          cnt <= (others => '0');
        else
          cnt <= cnt + 1;
        end if;
      end if;
    end if;
  end if;
end process;
```

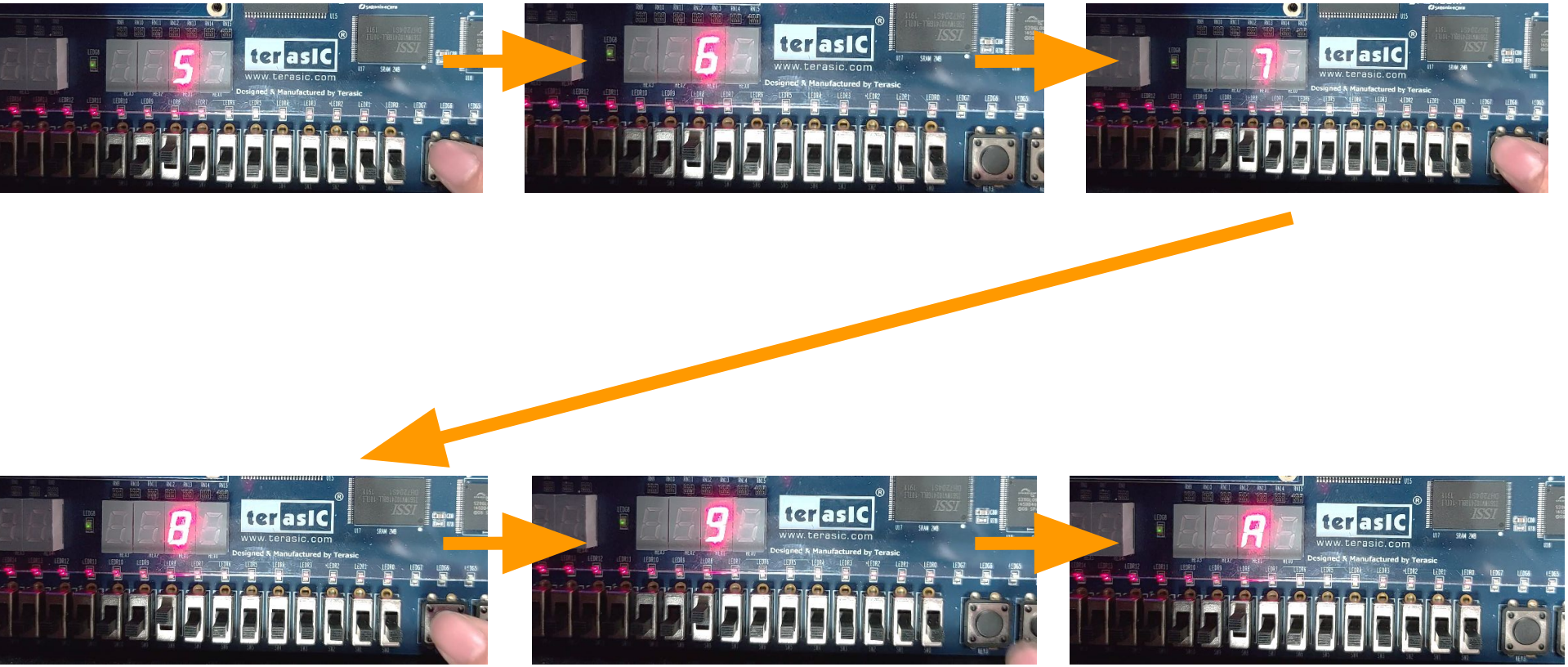
Write VHDL to implement button_debouncer

(block design)



Parameter	Value	Type
CLOCK_FREQ_HZ	50000000	Signed Integer
DEBOUNCE_MS	200	Signed Integer
CNT_WIDTH	200	Signed Integer

Experiment on Altera DE2-115



Conclusions

- Counter (LSB & MSB)
- LTspice & ModelSim
- R-2R DAC
- Switch Debouncing

Conclusions

- The flip-flop toggling and rollover are both observed in our simulators (LTspice and ModelSim) and experiments (DE2-115).
- The R-2R ladder is implemented with two resistor values to convert binary digital signals into an analog voltage.
- A two-stage synchronizer with some timer filtering can debounce a pushbutton.

If you have any feedback, please contact me at

twwang97@gmail.com

GitHub twwang97

Electronics and AI Training Program (3rd session in 2025)

Thanks for your attention

twwang97

nycu
國立陽明交通大學

陽明交大雷射系統研究中心