PROLOGUE

NOTHING IS GOING TO MAKE SENSE WITHOUT THIS

WHAT IS A COMPUTER? HOW DO THEY WORK?

- A Computer is any machine that can be programmed to complete tasks
- Computers can be mechanical not just electronic
- Digital computers use logic to execute tasks
- At its most basic level a computer uses binary to compute actions

WHAT'S BINARY

- Just like counting by tens binary is a counting system
- Instead of counting 1, 2, 3, 4 binary counts as 00, 01, 10, 11
- A rolling count
- Binary can be used to communicate values for example in electronic logic 1's are power and 0 are ground
- One binary number is called a bit a group of 8 is called a byte

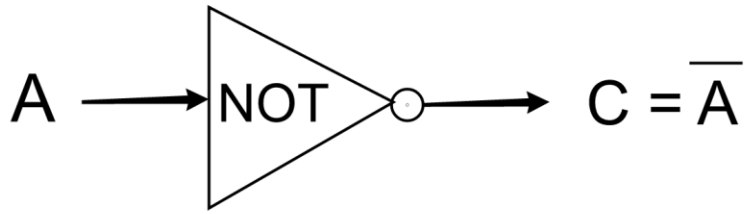
WHAT'S A LOGIC GATE?

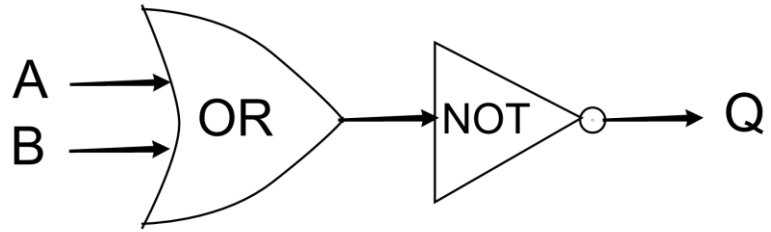
- Many types with different functions however the three most basic are the not gate, the and gate, and the or gate

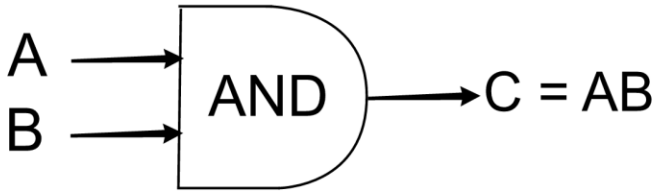
A	Q
0	1
1	0

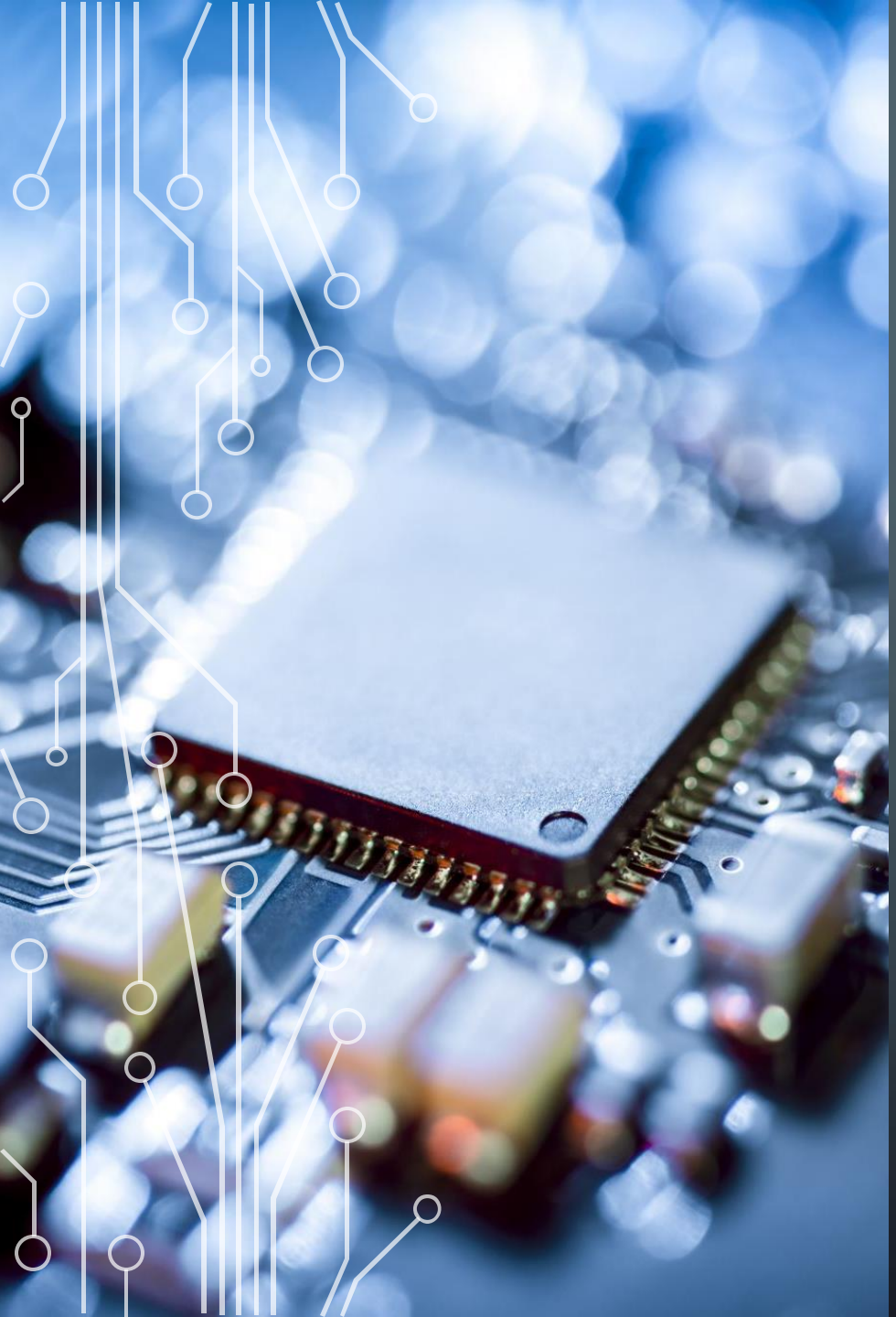
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1







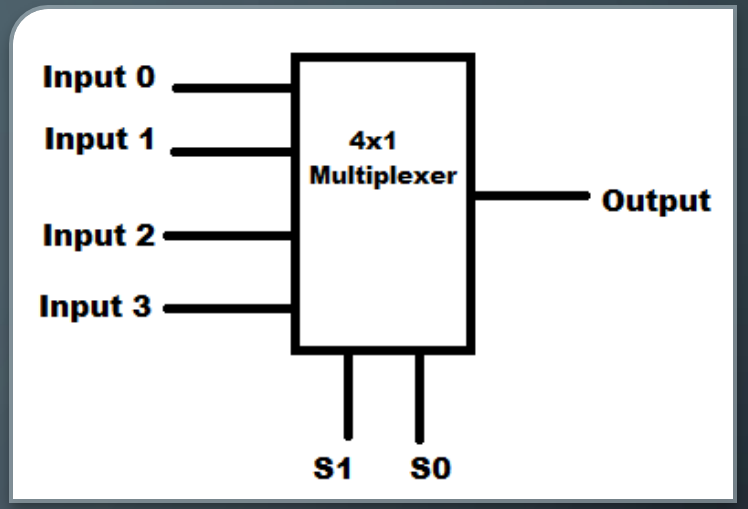


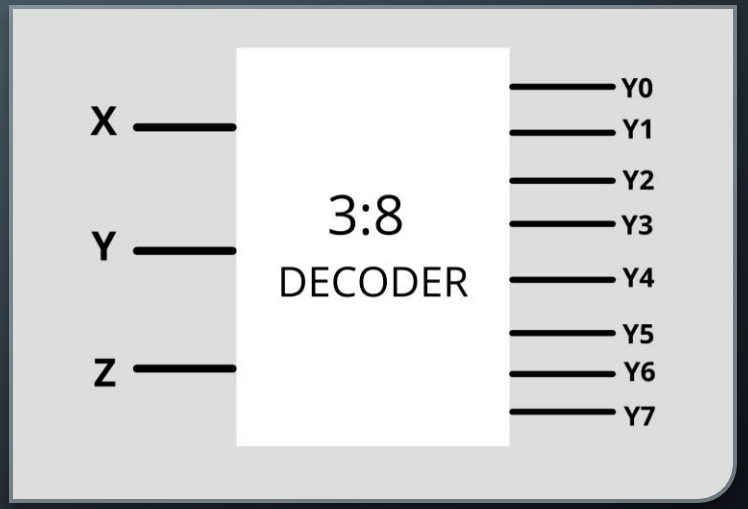
BASIC COMPUTING CHIPS

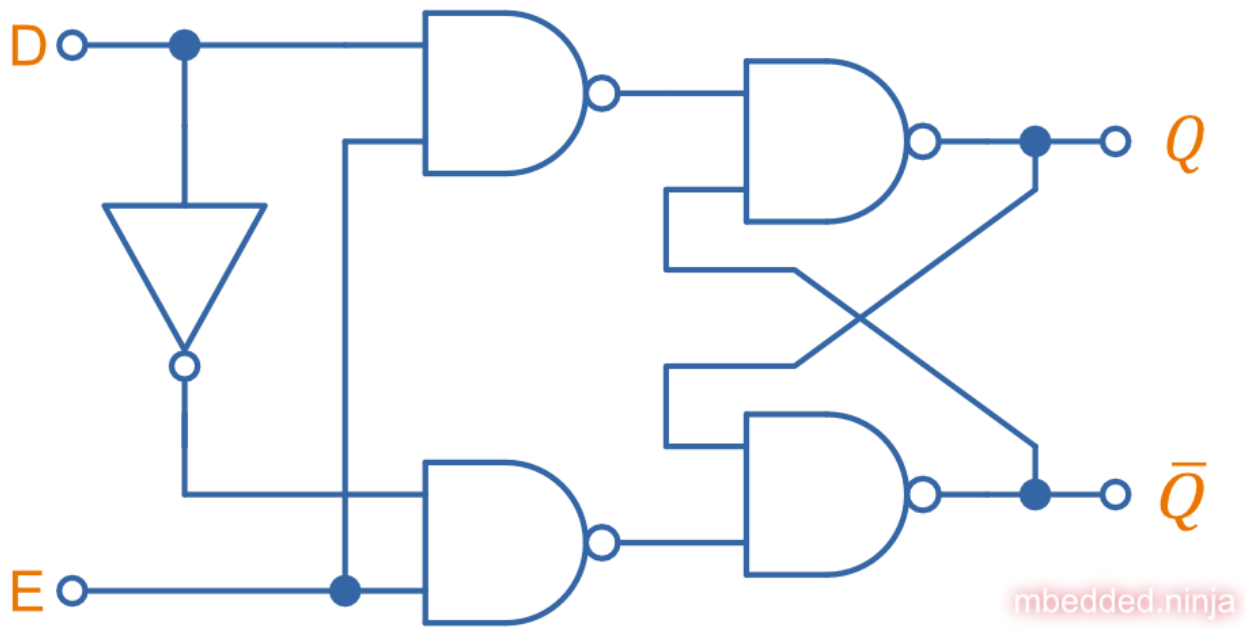
ITS ABOUT TO GET COMPLICATED

PLEXERS

- Two types Multiplexer, and Decoders
- Multiplexers Take multiple signals and decide what signal to output (like switching the input on a tv)
- Decoders take an input signal and activate a specific output based on it



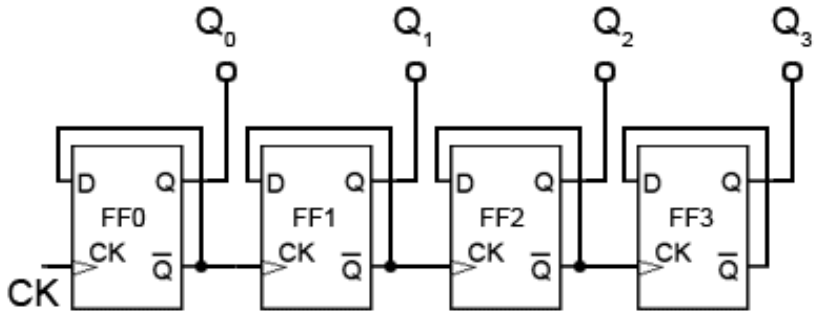


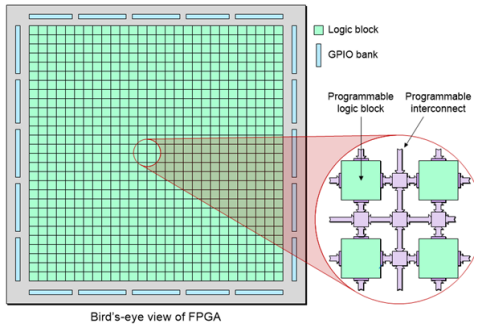


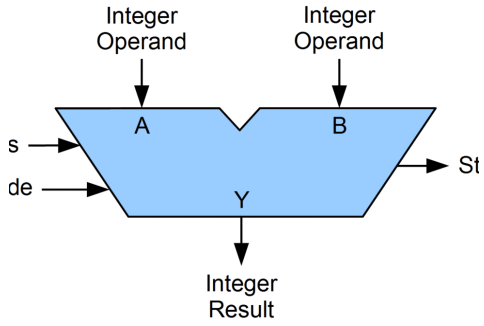
MEMORY

- Many Forms the simplest is called a Latch (stores one bit)
- Registers store data near a processor
- RAM (Random Accesses Memory can quickly change values as need be)
- ROM (Read Only Memory is programed once and keeps these values often used to store instruction steps)

LOGICAL PROCESSORS



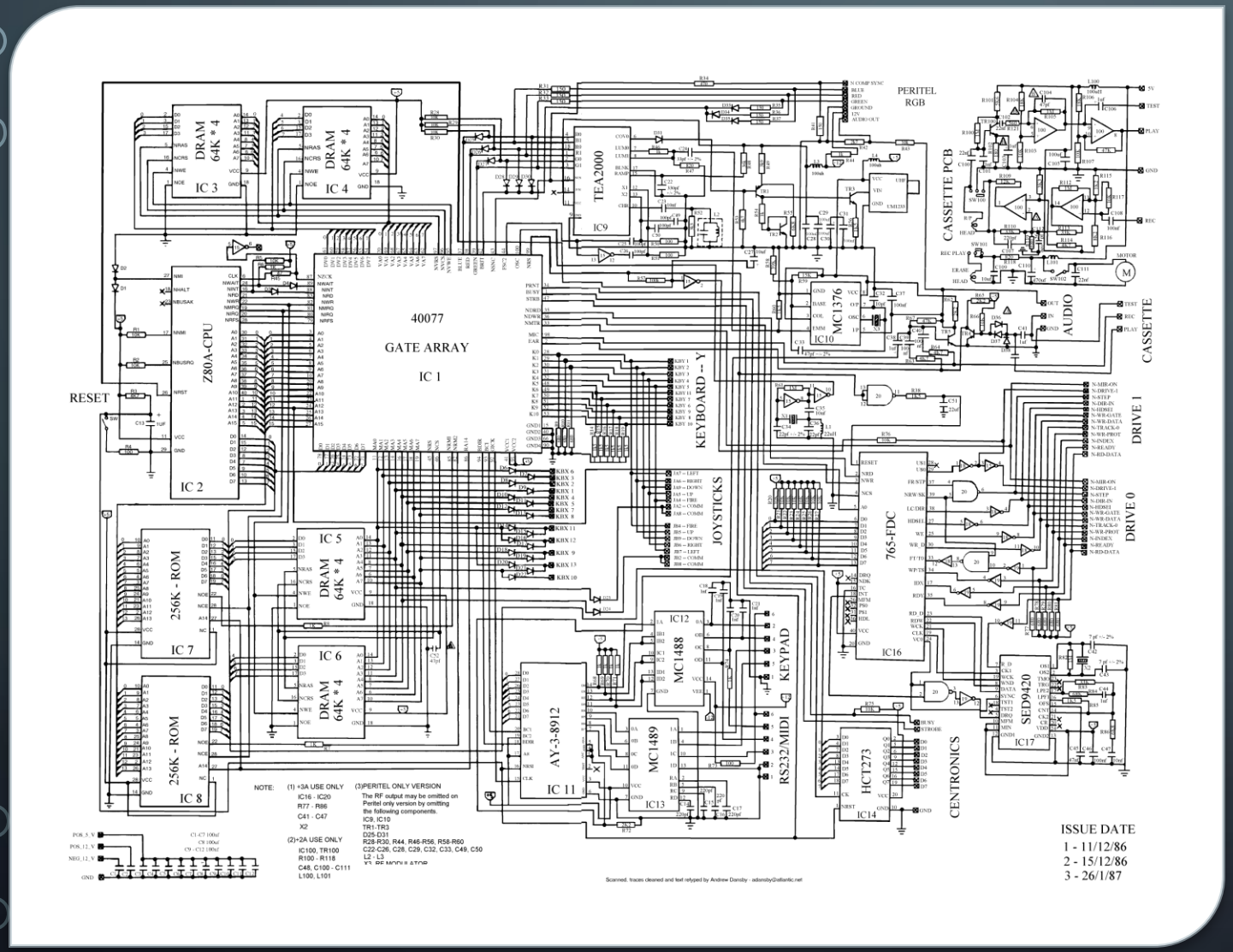




- Gate Arrays - Reads the current data set and decides what actions are necessary
- ALU (Arithmetic Logic Unit) - Preform basic operations on given data add, subtract, multiply and divide and comparisons <, > and bitwise operations Shifting bits left or right and finally ANDing, ORing, XORing values)
- Program Counter - Keeps track of which instructions need to be executed next

NOW THE REAL THING

BUCKLE UP

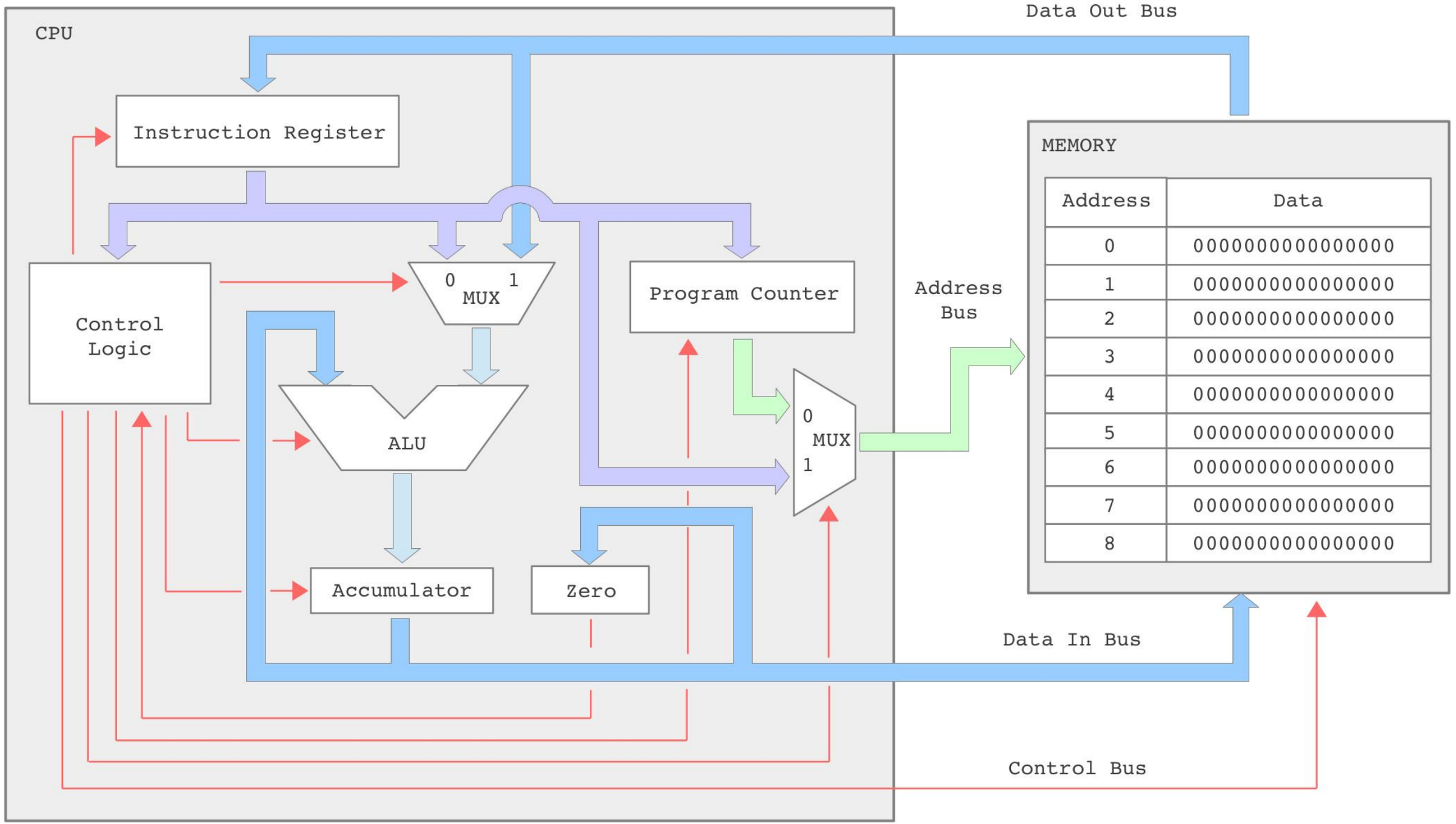


HOW DOES A COMPUTER WORK

- So Basically…
- Nah I'm Just kidding let's simplify it a bit

THE SIMPLE EXPLANATION

(THANK ME LATER)


CPU
Instruction Register
Control Logic
0 MUX 1
ALU
Program Counter
0 MUX 1
Accumulator
Zero
Data Out Bus
Address Bus
Data In Bus
Control Bus
MEMORY
Address
Data
0 0000000000000000
1 0000000000000000
2 0000000000000000
3 0000000000000000
4 0000000000000000
5 0000000000000000
6 0000000000000000
7 0000000000000000
8 0000000000000000

NOW YOU KNOW THE BASICS

THANKS TO MWAH

HOW TO BUILD A SMART PARKING GARAGE

BY: WILLIAM CONNER AND ADI LEVI

DESIGN PROCESS

Originally the plan was to use exclusively 7400 logic (one floor nine spaces)

Decided we wanted to add the extra functionality that a PLD offers (two floors 9 per floor 18 total and full lot indicators)

Decided to highlight spots remaining to drivers because that is more useful information

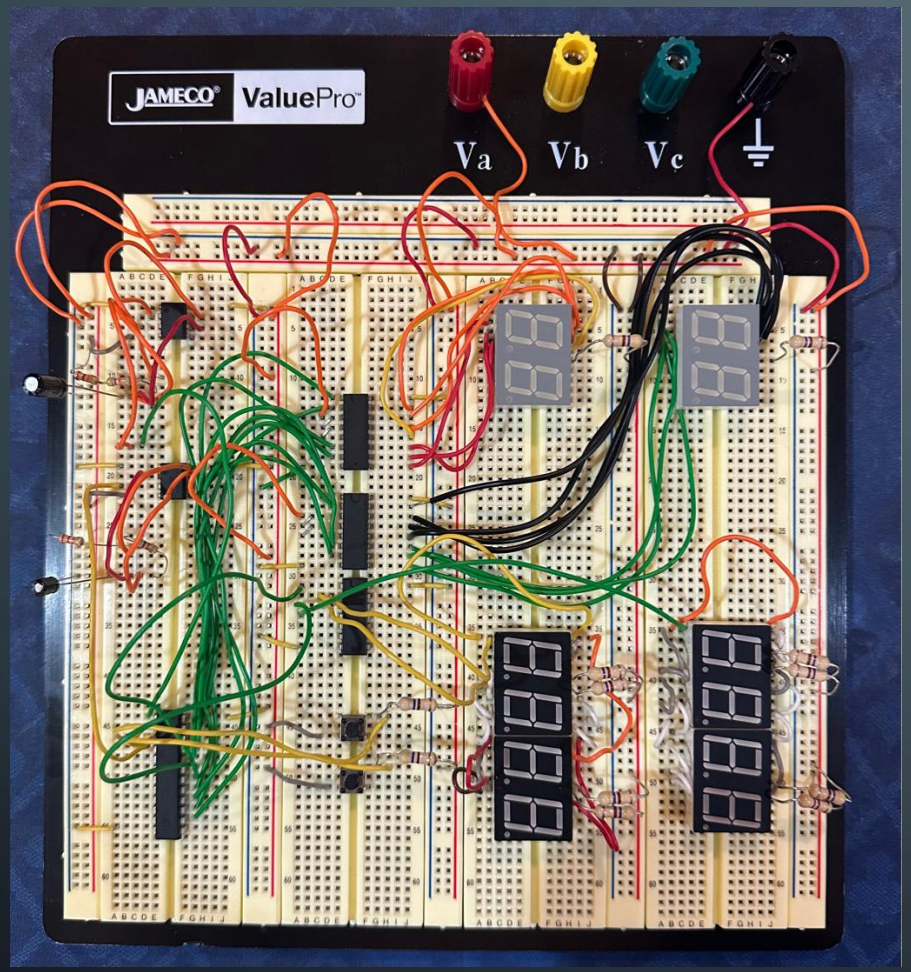
Introduced random floor selection

ABOUT OUR PROJECT

- comprised of 6 IC's
- Two separate counters 0000 to 1001are kept on the PLD
- The PLD also detects when each floor is full and displays an indicator
- Floor randomization is created using a second 555 timer set to a different pulse

PART LIST

# of parts	Part name
2	555 timers
2	button
1	Gal22v10c PLD
2	7448 decoders
6	seven segment display
1	7404 inverter
12	470 Ω resistor
2	47k Ω resistor
2	22k Ω resistor

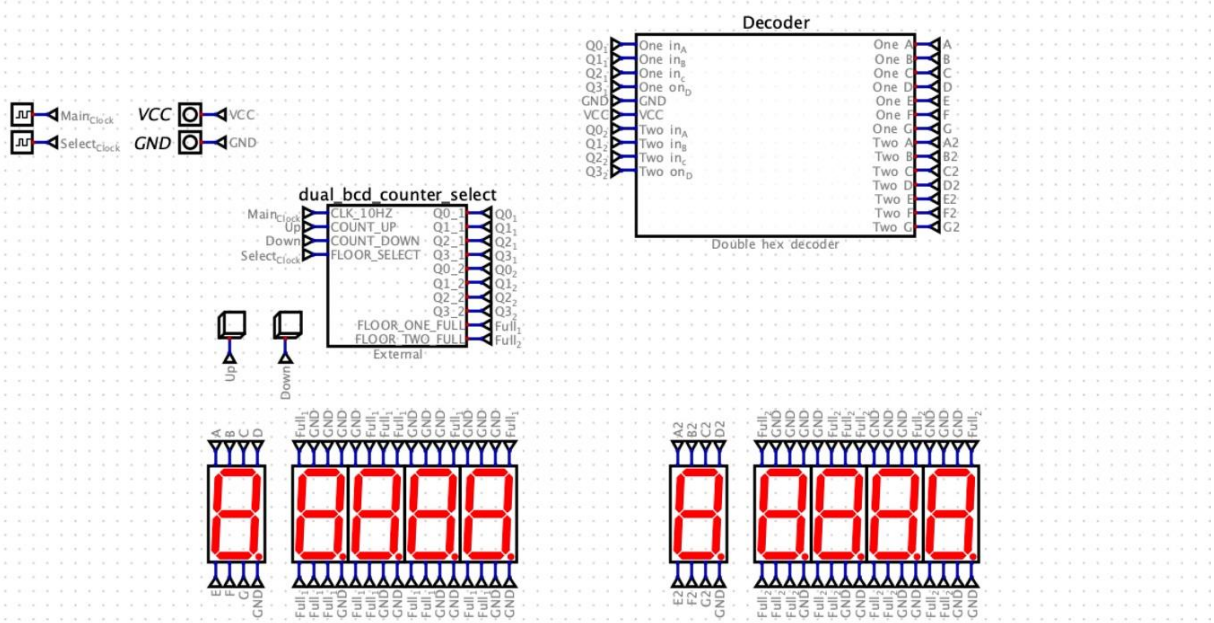


HOW OUR PROJECT LOOKS LIKE AND HOW IT WORKS

Two clock signals are given to the PLD one used to drive "randomness."

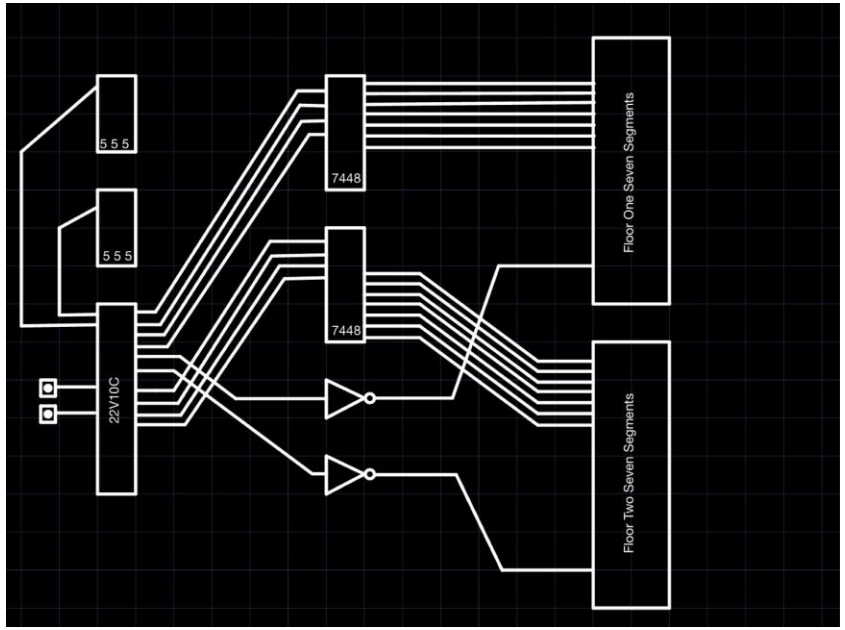
The PLD keeps an individual count of each floor and drives a BDC output to decoders.

One extra pin per floor to drive the “full” signal



WIRING DIAGRAM

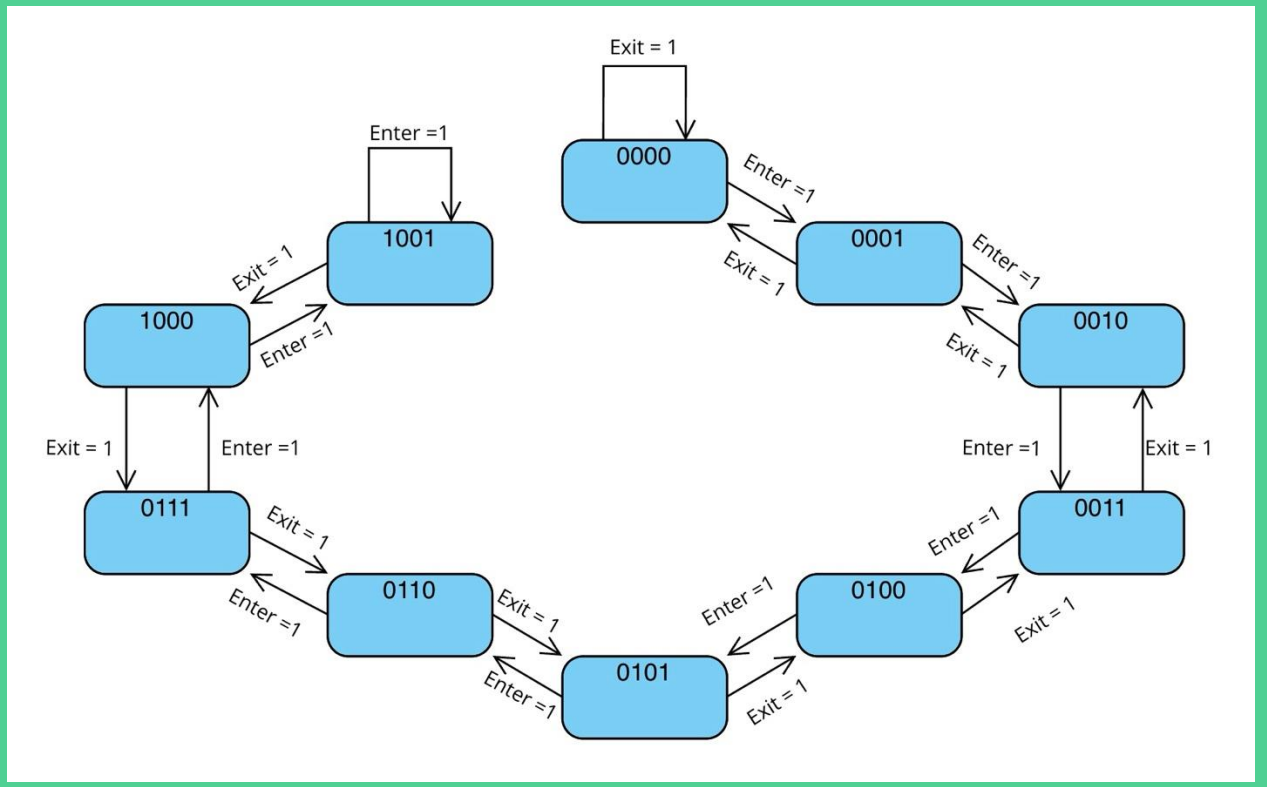
- Designed in Digital
- Tunnels with the same names are connected (used to keep the design organized).
- Double Hex is an embedded design of two BDC to seven segment decoders.



BLOCK DIAGRAM

STATE TABLE AND STATE DIAGRAM

Current State	Car Enter	Car Leave	Next State Floor	Output
1001	1	X	1000	1000
1000	1	X	0111	0111
0111	1	X	0110	0110
0110	1	X	0101	0101
0101	1	X	0100	0100
0100	1	X	0011	0011
0011	1	X	0010	0010
0010	1	X	0001	0001
0001	1	X	0000	0000
0000	1	X	0000	0000
0000	X	1	0001	0001
0001	X	1	0010	0010
0010	X	1	0011	0011
0011	X	1	0100	0100
0100	X	1	0101	0101
0101	X	1	0110	0110
0110	X	1	0111	0111
0111	X	1	1000	1000
1000	X	1	1001	1001
1001	X	1	1001	1001



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dual_bcd_counter_select
    Port (
        CLK_10HZ        : in  STD_LOGIC;   -- 10 Hz System Clock -> Pin 1
        CAR_LEAVE       : in  STD_LOGIC;   -- Button to count up (Car leaves) -> Pin 2
        CAR_ENTER       : in  STD_LOGIC;   -- Button to count down (Car enters) -> Pin 3
        FLOOR_SWITCH    : in  STD_LOGIC;   -- Switch to choose which floor to control -> Pin 4
        FLOOR1_BIT0     : out STD_LOGIC;   -- Bit 0 of floor one counter -> Pin 12
        FLOOR1_BIT1     : out STD_LOGIC;   -- Bit 1 of floor one counter -> Pin 13
        FLOOR1_BIT2     : out STD_LOGIC;   -- Bit 2 of floor one counter -> Pin 14
        FLOOR1_BIT3     : out STD_LOGIC;   -- Bit 3 of floor one counter -> Pin 15
        FLOOR2_BIT0     : out STD_LOGIC;   -- Bit 0 of floor two counter -> Pin 16
        FLOOR2_BIT1     : out STD_LOGIC;   -- Bit 1 of floor two counter -> Pin 17
        FLOOR2_BIT2     : out STD_LOGIC;   -- Bit 2 of floor two counter -> Pin 18
        FLOOR2_BIT3     : out STD_LOGIC;   -- Bit 3 of floor two counter -> Pin 19
        FLOOR1_FULL     : out STD_LOGIC;   -- Light when floor one  full -> Pin 20
        FLOOR2_FULL     : out STD_LOGIC    -- Light when floor two  full -> Pin 21
    );
end dual_bcd_counter_select;

architecture Simplified of dual_bcd_counter_select
    signal counter1          : STD_LOGIC_VECTOR(3 downto 0) := "1001";  -- Counter for floor
one signal counter2          : STD_LOGIC_VECTOR(3 downto 0) := "1001";  -- Counter for floor
two signal car_leave_prev    : STD_LOGIC := '0';  -- Previous state of CAR_LEAVE
    signal car_enter_prev    : STD_LOGIC := '0';  -- Previous state of CAR_ENTER
begin
```

VHDL CODE EXPLAINED (BORING)

- Inputs and outputs assigned
- On start counters are initialized at 1001 (not functional)
- Detects if car leave or enter is high
- Increment when a car leaves
- Decrement when a car enters
- Floor detect 0 for floor 1 and 1 for floor 2

```
    -- Counting logic
    process(CLK_10HZ)
        variable car_leave_edge : STD_LOGIC := '0';  -- Detect rising edge of
CAR_LEAVE
        variable car_enter_edge : STD_LOGIC := '0';  -- Detect rising edge of
CAR_ENTER
    begin
        if rising_edge(CLK_10HZ) then
            -- Detect button press
            car_leave_edge := CAR_LEAVE and not car_leave_prev;
            car_enter_edge := CAR_ENTER and not car_enter_prev;
            car_leave_prev <= CAR_LEAVE;
            car_enter_prev <= CAR_ENTER;

            -- Check for button presses and update counters
            if car_leave_edge = '1' and car_enter_edge = '0' then
                if FLOOR_SWITCH = '0' then
                    -- Count up floor one
                    case counter1
                        when "0000" => counter1 <= "0001";
                        when "0001" => counter1 <= "0010";
                        when "0010" => counter1 <= "0011";
                        when "0011" => counter1 <= "0100";
                        when "0100" => counter1 <= "0101";
                        when "0101" => counter1 <= "0110";
                        when "0110" => counter1 <= "0111";
                        when "0111" => counter1 <= "1000";
                        when "1000" => counter1 <= "1001";
                        when others => counter1 <= "1001";  -- Max value
                    end case;
                else
                    -- Count up floor two
                    case counter2
                        when "0000" => counter2 <= "0001";
                        when "0001" => counter2 <= "0010";
                        when "0010" => counter2 <= "0011";
                        when "0011" => counter2 <= "0100";
                        when "0100" => counter2 <= "0101";
                        when "0101" => counter2 <= "0110";
                        when "0110" => counter2 <= "0111";
                        when "0111" => counter2 <= "1000";
                        when "1000" => counter2 <= "1001";
                        when others => counter2 <= "1001";  -- Max value
                    end case;
                end if;
```

CONTINUED

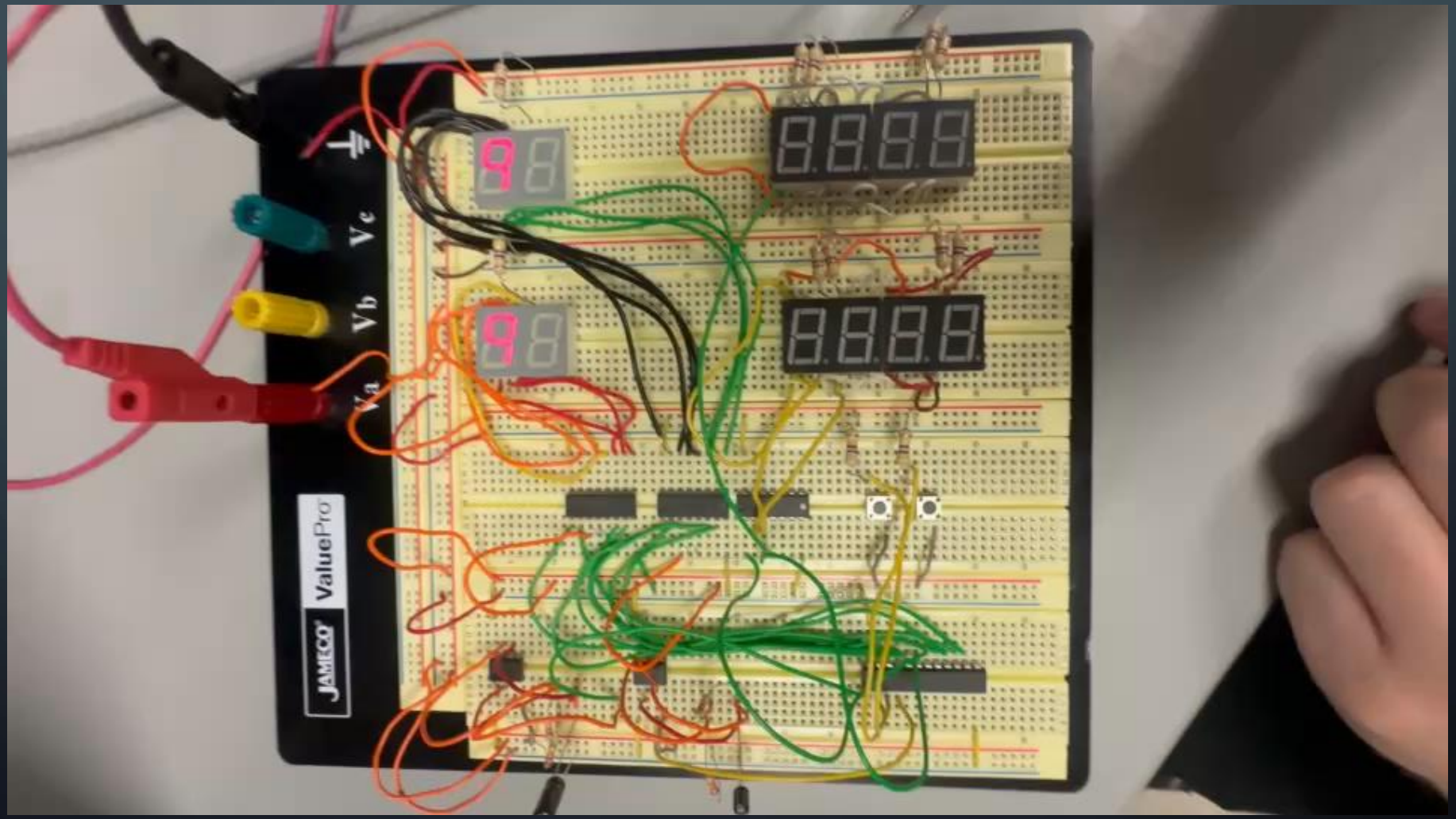
- Detect a button press if car leave is 1 and was 0 in the last check
- Store the last state
- Is a car leaves and no car enters than check the floor indicator check the current value for floor one than increase it one
- Else if floor is not 0 aka 1 than do the same but for floor 2

```
        elsif car_enter_edge = '1' and car_leave_edge = '0' then
            if FLOOR_SWITCH = '0' then
                -- Count down floor one
                case counter1
                    when "1001" => counter1 <= "1000";
                    when "1000" => counter1 <= "0111";
                    when "0111" => counter1 <= "0110";
                    when "0110" => counter1 <= "0101";
                    when "0101" => counter1 <= "0100";
                    when "0100" => counter1 <= "0011";
                    when "0011" => counter1 <= "0010";
                    when "0010" => counter1 <= "0001";
                    when "0001" => counter1 <= "0000";
                    when others => counter1 <= "0000";  -- Min
value
                end case;
            else
                -- Count down floor two
                case counter2
                    when "1001" => counter2 <= "1000";
                    when "1000" => counter2 <= "0111";
                    when "0111" => counter2 <= "0110";
                    when "0110" => counter2 <= "0101";
                    when "0101" => counter2 <= "0100";
                    when "0100" => counter2 <= "0011";
                    when "0011" => counter2 <= "0010";
                    when "0010" => counter2 <= "0001";
                    when "0001" => counter2 <= "0000";
                    when others => counter2 <= "0000";  -- Min
value
                end case;
            end if;
        end if;
    end if;
end process;

-- Lights for when floors are full
FLOOR1_FULL <= '1' when counter1 = "0000" else '0';
FLOOR2_FULL <= '1' when counter2 = "0000" else '0';

-- Assign counter outputs
FLOOR1_BIT0 <= counter1(0);
FLOOR1_BIT1 <= counter1(1);
FLOOR1_BIT2 <= counter1(2);
FLOOR1_BIT3 <= counter1(3);
FLOOR2_BIT0 <= counter2(0);
FLOOR2_BIT1 <= counter2(1);
FLOOR2_BIT2 <= counter2(2);
FLOOR2_BIT3 <= counter2(3);

end;
```

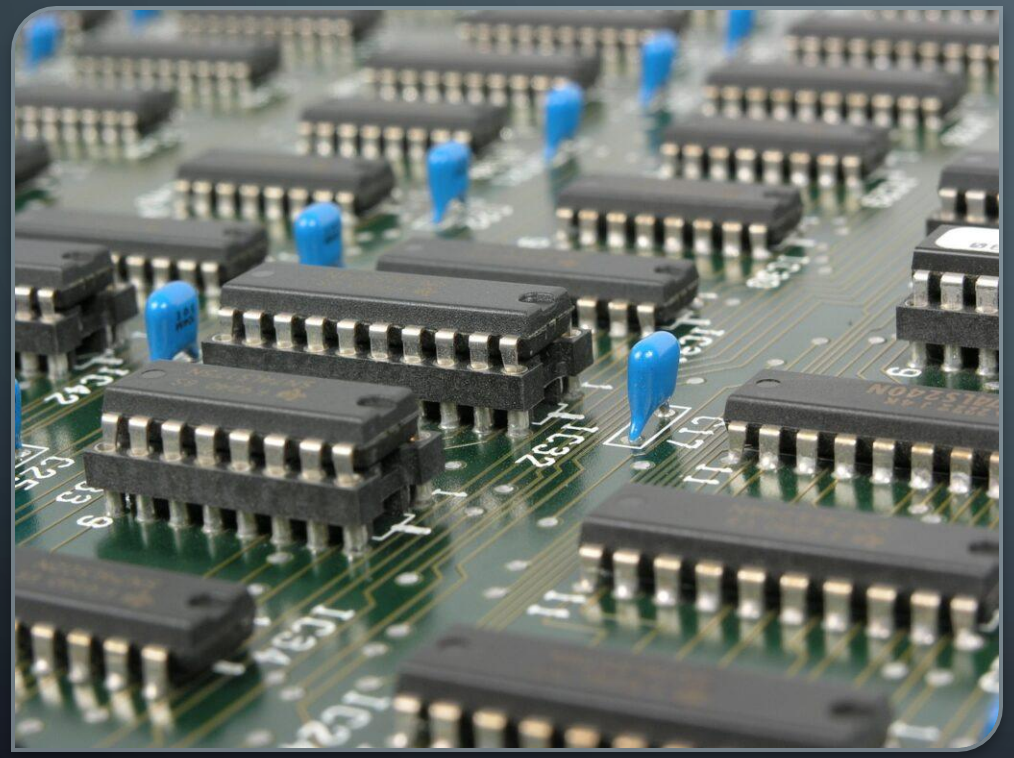
STILL GOING AND STILL BORING

- Similar but the opposite
- If a car enters and no car leaves than check the current state and count down one state
- When counter is 0000 than output 1 for that full signal
- Assign the counter bits to pins

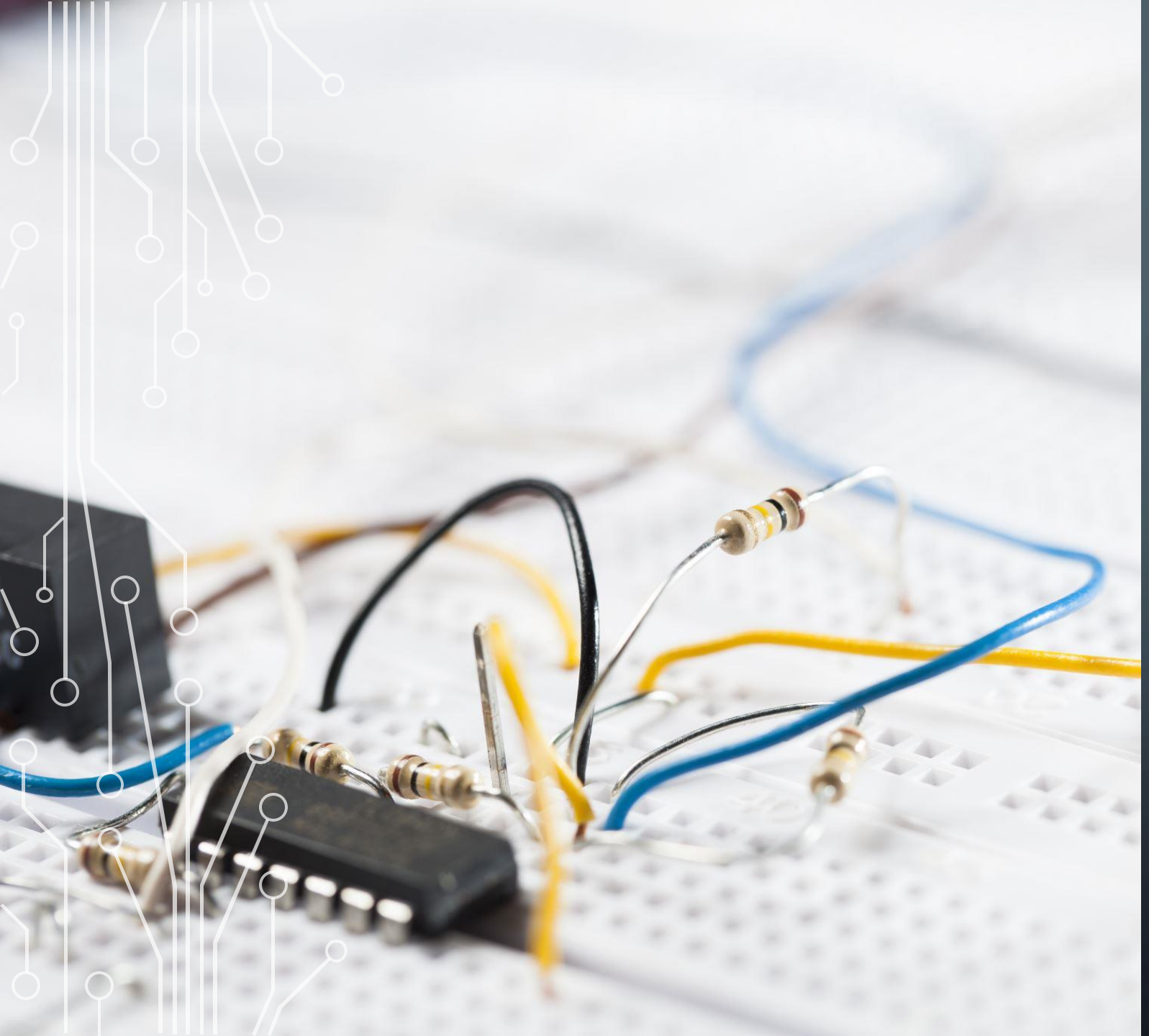
VIDEO DEMO



OPPORTUNITIES FOR IMPROVEMENT



1. If you hold the button it will continue to count
2. Full signal is programed as active high on PLD
3. Starts at 0 instead of 9 when counting
4. Limited to single digits so limited only to 9 per floor



SOLUTIONS TO DESIGN OVERSIGHTS

- 1. Using Schmitt trigger combined with a resistor and capacitor, we could smooth out the signal to clean the output or use flip-flops to store the previous states.
- 2. Change the code to have an output of 0 instead of 1 so we would not need the inverter.
- 3. Preload a 9 on the 7-segment display using a register.
- 4. We could have used 2 PLDS, one per floor to drive two 4 bit counters one for each digit