# ECE 385 – Digital Systems Laboratory

## Lecture 2 – Lab 2 Introduction, TTL Debugging

Zuofu Cheng

Spring 2019

Link to Course Website

ECE ILLINOIS

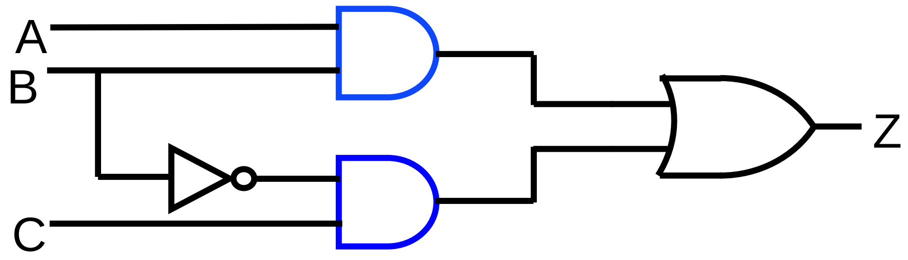
ILLINOIS

## Experiment 1 – Introduction to TTL

- Goal – Design a 2 to 1 multiplexor using only quad 2-input NAND gate chips (7400)
- The naïve design will have glitches, redesign circuit to remove possibility of static-1 hazard
- Things to do in Lab 1 during session
  - Find lab partner in same section
  - Check out lab kit (one per group)
  - Build both naïve and glitch-free circuits
  - Capture operation using oscilloscope for comparison in lab report
- After your Lab 1 session
  - Write up lab report (this lab report is to be done **individually**)
  - Upload onto Compass before start of next lab session
- This will be the **only** time this semester will you should have enough time to complete the entire lab without preparation.

## Experiment 1 – 2 to 1 MUX

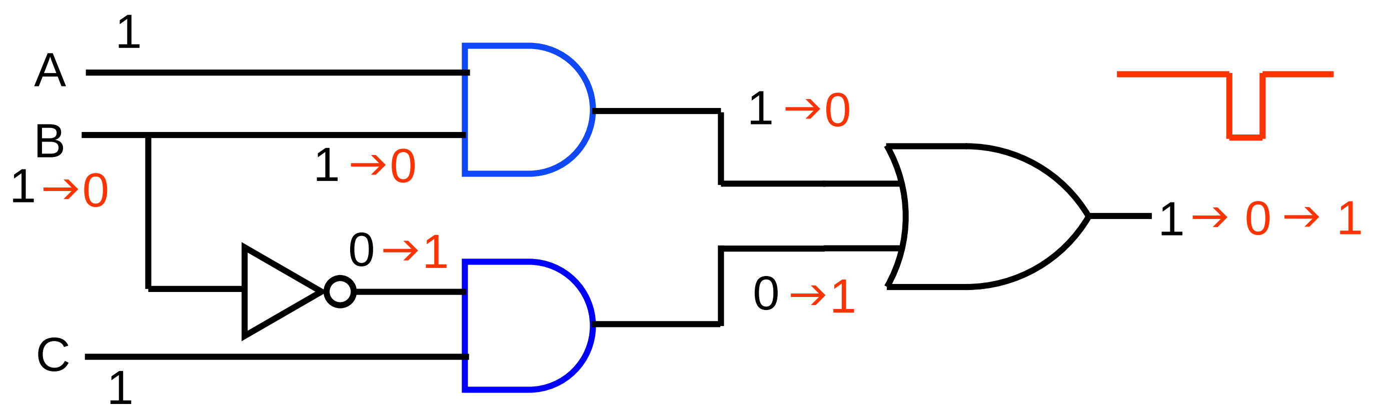
- What are inputs, outputs?
  - MUX can be thought of as digital switchbox, 2-1 means two inputs (A, C), one output (Z) and select (B)
- What is Boolean function for operation (in SOP form)?
  - Z = BA + B'C
- How do we implement using only NAND gates? How many 7400s do we need?

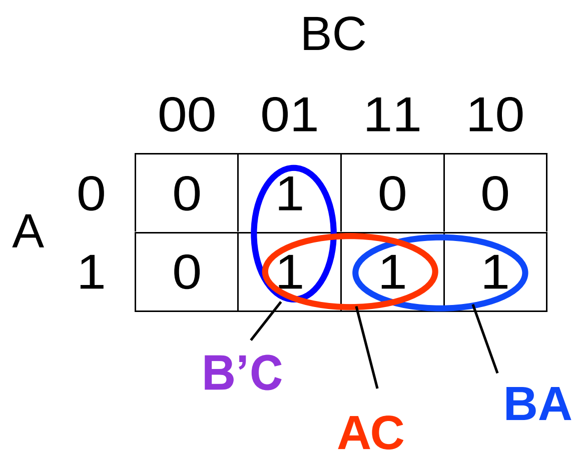


## Static Hazards

- Boolean Algebraic analysis of a logic circuit gives us steady state (DC) behavior of that circuit. For example for a two-input OR gate, input of 10 and 01 both produce 1 output.
  - However, when the input 10 is changed to 01, both inputs may not change exactly at the same time in a physical world. There exists a possibility of both inputs assuming 0 values for a short time. This could lead to a short duration 0 output. This is called a glitch.
- A Static-1 Hazard is the possibility of producing a 0 glitch (pulse) when the static analysis of the logic tells us that the output should stay at logic 1.
  - Formal Definition: A Static-1 Hazard is a pair of input combinations that (a) Differ in only one input variable and (b) both give a 1 output; such that it is possible for momentary 0 output to occur during transition in the differing input variable.
- A Static-0 Hazard is the possibility of producing a 1 glitch (pulse) when the static analysis of the logic tells us that the output should stay at logic 0.
  - Formal Definition: A Static-0 Hazard is a pair of input combinations that (a) Differ in only one input variable and (b) both give a 0 output; such that it is possible for momentary 1 output to occur during transition in the differing input variable.

# Static Hazards – Example with 2 to 1 MUX





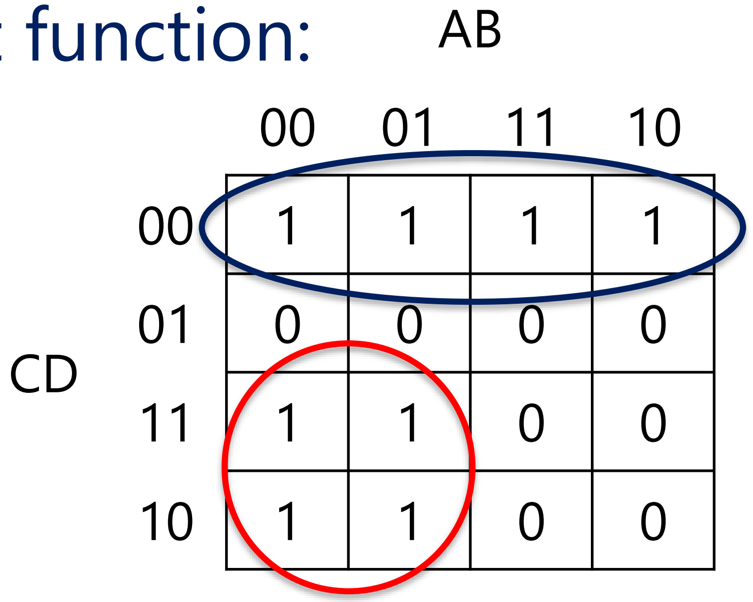
| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |

To avoid Static-1 hazard in an AND-OR (SOP) circuit, cover all adjacent min-terms in the K-map.
In this example, add the term AC

- What is the final Boolean function for glitch-free circuit?
  - Z = B’C + BA + AC

## Static Hazards – Another Example

- Another example, consider 4 input function:
- What is minimal representation?
- Z = C′D′+A′C



| CD \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 1 | 1 | 1 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | 1 | 1 | 0 | 0 |
| 10 | 1 | 1 | 0 | 0 |

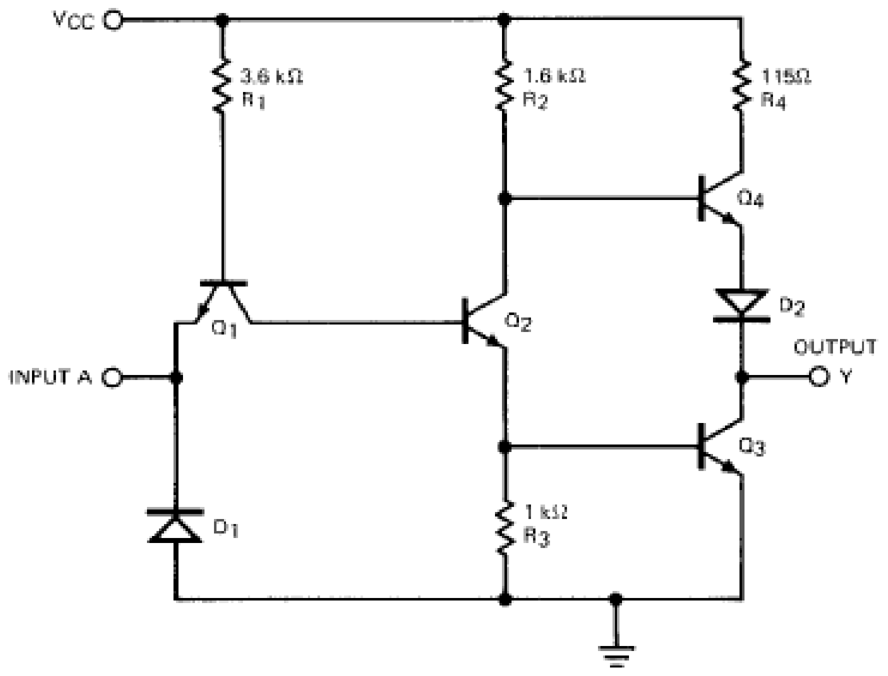
- Are there any static hazards?
  - Adjacent product terms without overlapping terms?
  - Remember, K-maps wrap around
- How do we solve static hazard? (what term(s) should we add?)

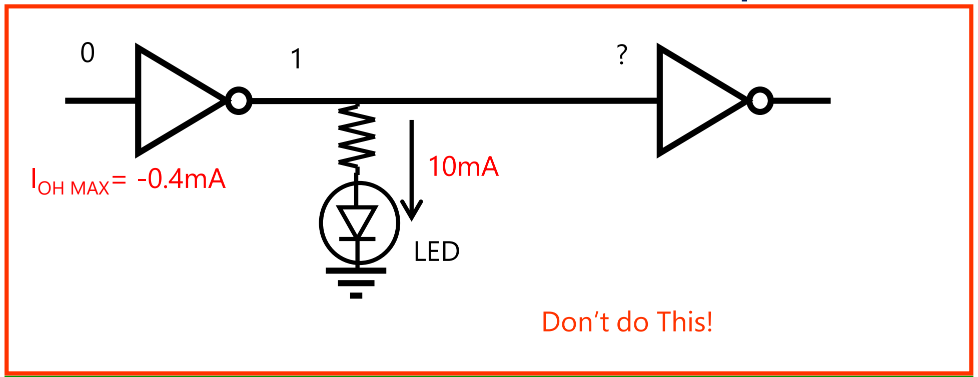
## Debugging TTL at Home

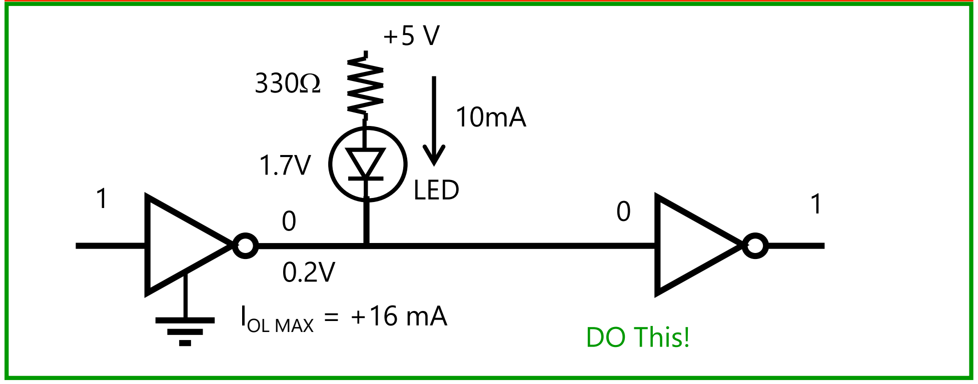
- You will be expected to come into lab with mostly working circuit
  - Oscilloscope / function generator is available in open-lab sessions
  - Lab kits include LEDs and switches for seeing outputs and inputs
  - LEDs must be driven with current limiting (with current sink driver)
    - Use inverter as driver
    - Use current sink topology to avoid inversion of LED
  - Switches must be de-bounced (or glitches will occur)
    - Only necessary for clock signal (if design is strictly synchronous)

## Debugging TTL - Driving LEDs

- TTL ICs have very limited current source capability (why? take a look at TTL inverter schematic below)
- Resistor and gate to pull **down** (note LED is on when output is

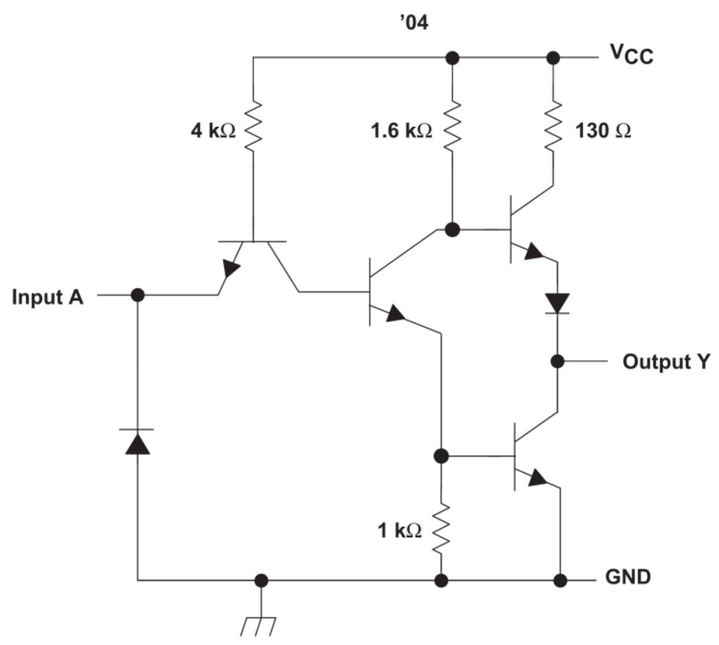


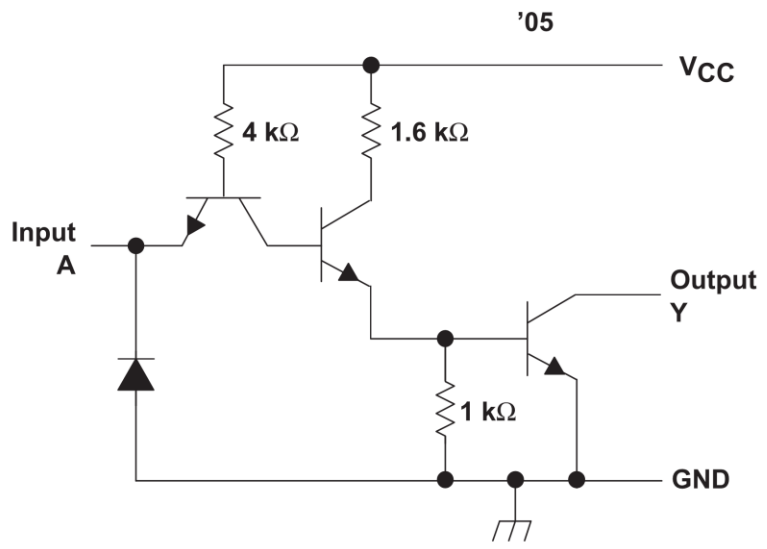




## Open Collector

- General guide recommends using only open collector (7405) chips to drive LEDs
- Not strictly necessary, but may be good practice for other reasons
- What are some advantages of open collector (disadvantages)?





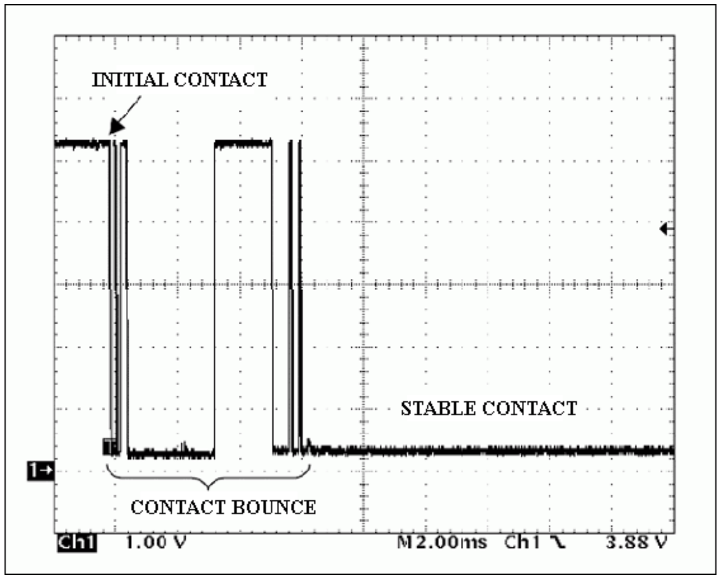
## Recommended Tools for Lab Report

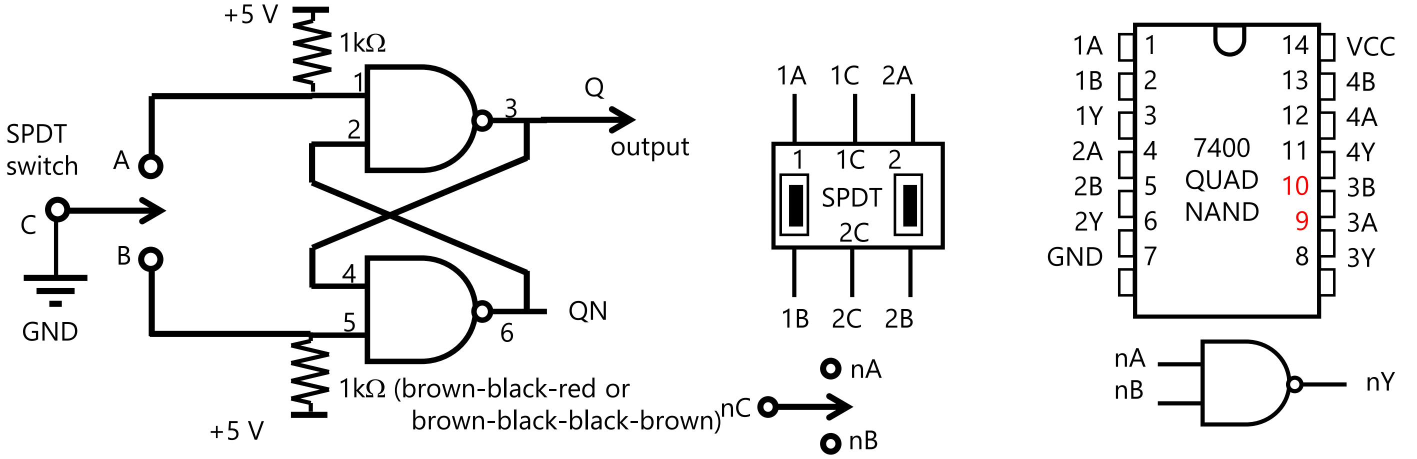
- Upload system takes PDF or DOC file (among others)
- Should use CAD program for logic diagrams (CircuitLab, EasyEDA, SchemeIT, Fritzing, LTSpice)
- Use Visio (Microsoft) or Inkscape (Open Source) for block diagrams
- Timing diagrams may be done by hand or with tools e.g. Wavedrom (can also use Quartus Prime to draw if it is already installed)
- Pictures (scans) are acceptable for layout charts, or you can import blank from PDF of General Guide and draw using Visio / Inkscape

## Switches for Debugging

## De-bouncing Switches

- Mechanical switches will cause glitches during switch due to contact bounce



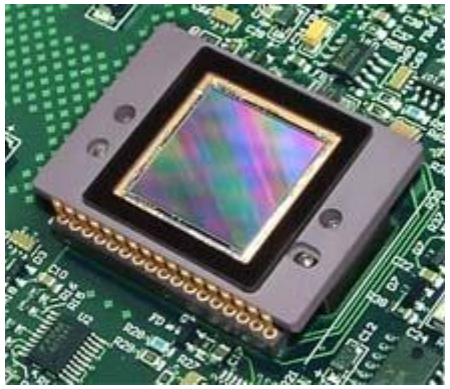


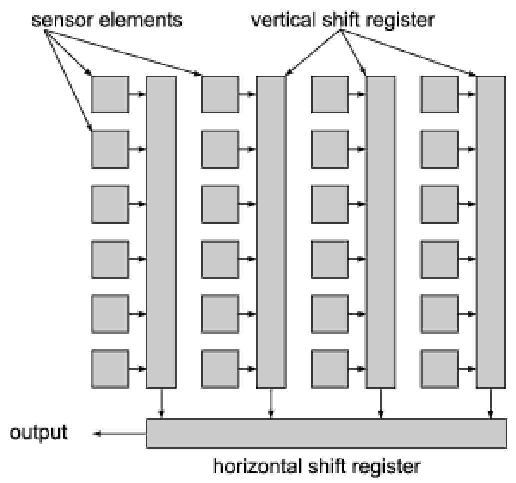
## Experiment 2 – Data Storage with TTL

- Design a Shift Register Memory (4 x 2 bit)
  - Serial Input and Output
  - Circulating data
  - Use a shift-register chip (e.g. 74LS194A) and other logic
  - Data sheet of the chip is posted online (under Labs)
  - **Must not use parallel load or parallel output (only serial input and right-most output may be used)**
  - **Data must also be continuously shifting**

## Experiment 2 – Motivation

- Why use shift register to create RAM?
  - Shift registers are simpler to layout and wire (no address pins, fewer data pins)
  - Some devices are inherently organized as shift registers (example CCD image sensor)

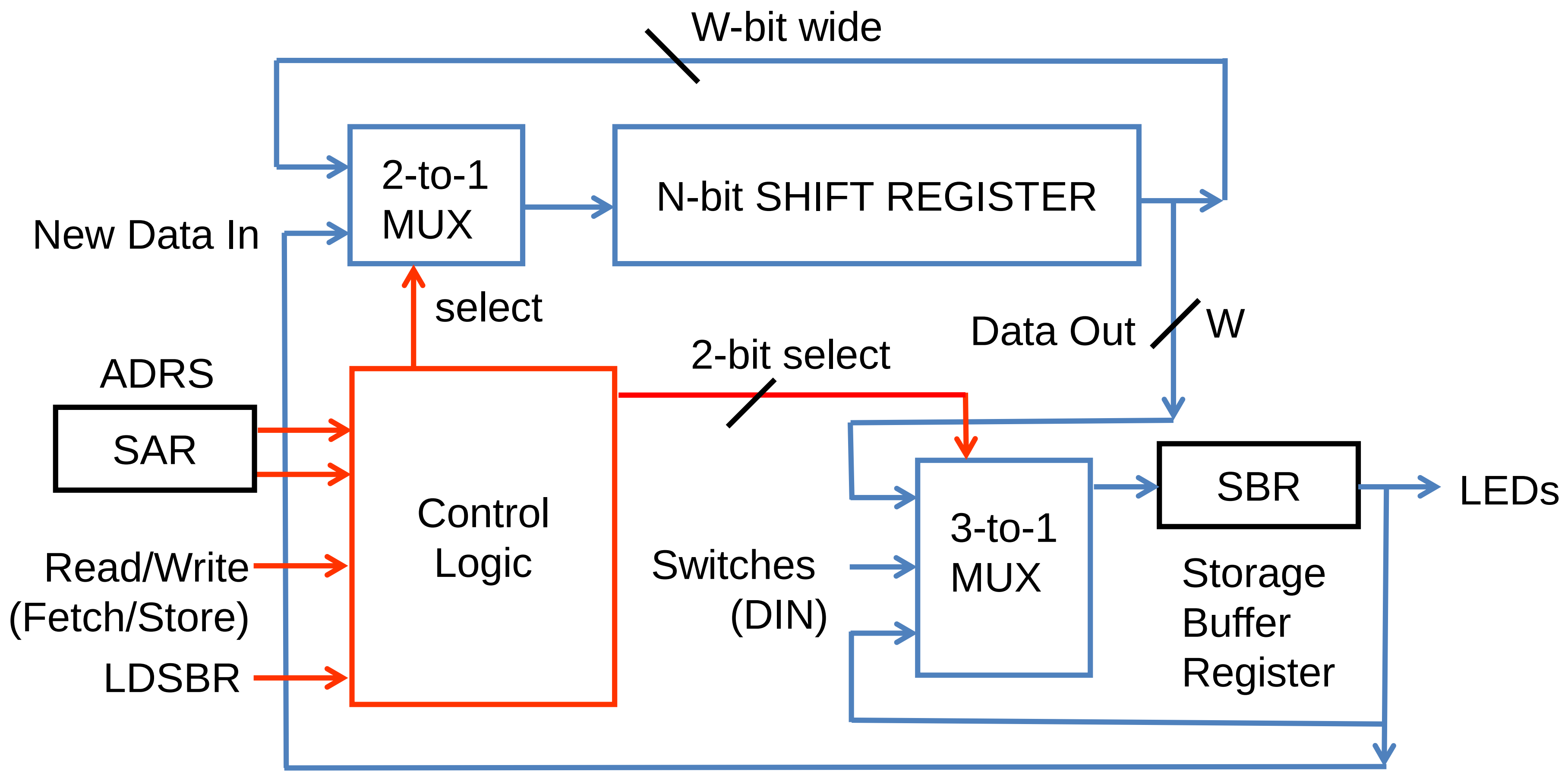




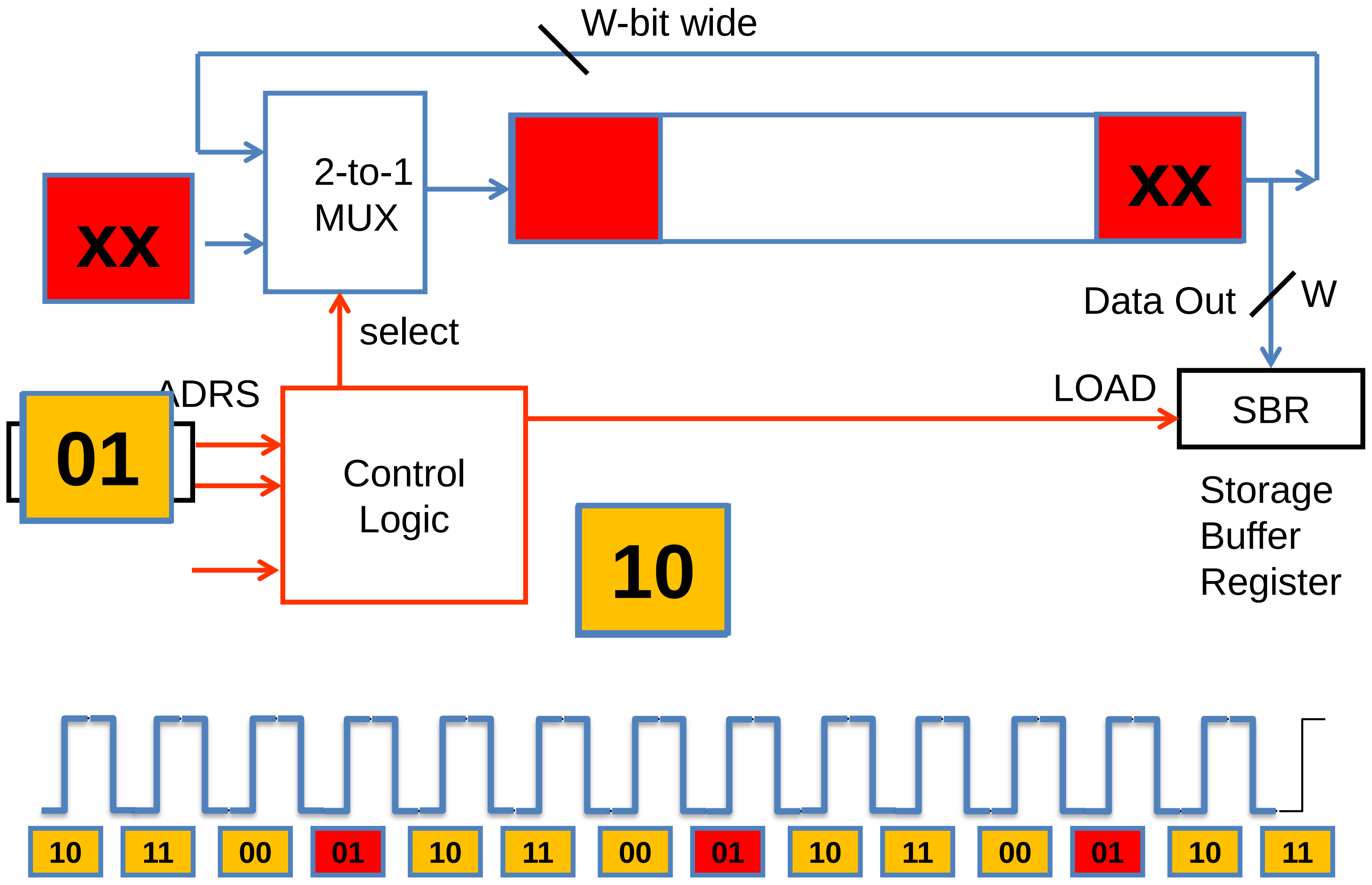
  - Note that any type of memory (shift register, parallel RAM, FIFO) may be used to emulate any other type, given additional control logic

# Experiment 2 – Data Storage with TTL

Use two 74LS194 shift-registers
Serial Input and Output
Circulating data



# Experiment 2 – Operation of Circuit



## Experiment 2 – Inputs and Outputs

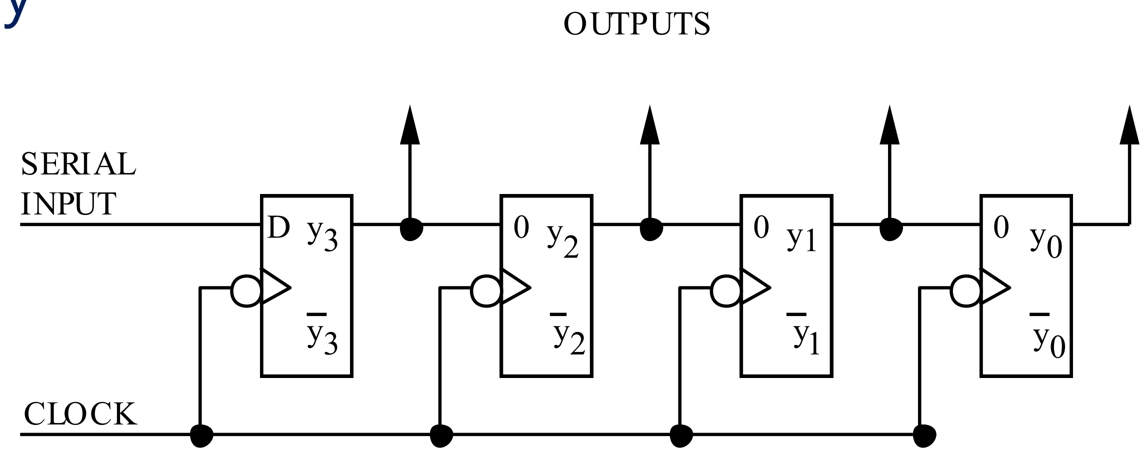
- FETCH (Switch)
  - When FETCH is high, the value in the data word specified by the SAR is read into the SBR.
- STORE (Switch)
  - When STORE is high, the value in the SBR is stored into the word specified by the SAR.
- SBR1, SBR0 (Flip-flops)
  - The data word in the SBR; either the most recently fetched data word or a data word loaded from switches
- SAR1, SAR0 (Switches)
  - The address, in the SAR, of a word in the storage
- DIN1, DIN0 (Switches)
  - Data word to be loaded into SBR for storing into storage
- LDSBR (Switch)
  - When LDSBR is high, the SBR is loaded with the data word DIN1, DIN0

## Experiment 2 – Demo

- FETCH (3 points)
  - M[SAR] -> SBR (multi-cycle operation)
- STORE (1 point)
  - SBR -> M[SAR] (multi-cycle operation)
- LOAD (1 point)
  - DIN -> SBR (single-cycle operation)
- NOP
  - Do nothing, contents of memory is preserved (shift register must continue to shift – **do not use clock inhibit on shift register)**
- You can assume that only one of the FETCH/STORE/LDSBR switches will be set at any given time (behavior is undefined if more than one is set at one time)
- You can get 1 demo point for demonstrating proper shift operation (not necessary to independently demo if FETCH is completely working)

## 74194 – Shift Register

- Always shift right (right shift on falling edge)
- Make sure you disable parallel load and clear signals
- Type of shift determined by what goes into leftmost input (logical, barrel, arithmetic)
- Remember: you may use parallel outputs for debugging purposes, but your circuit may only use the rightmost functionally

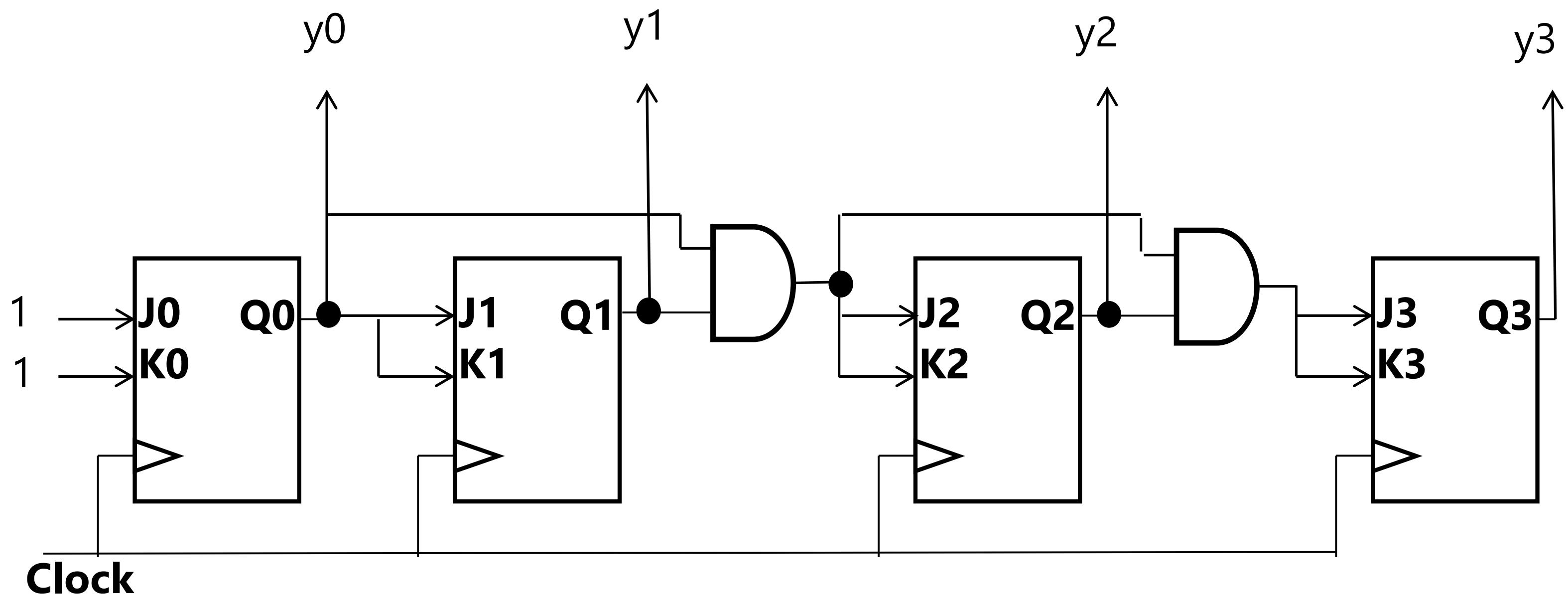


## Synchronous vs. Ripple Counter

- You may use either ripple (7493) or synchronous (74169 or 74193) counters
- Ripple counter
  - Asynchronous design (each clock is derived from previous output)
  - Simple to build and use (can easily extend without additional hardware)
  - Potentially glitch in operation

## Synchronous vs. Ripple Counter

- Synchronous counter (shown with J-K flip flop)
  - Synchronous design (each flip flop gets same clock)
  - More complex design – harder to extend to more bits
  - Removes possibility of glitch in outputs



## Why Avoid Clock Gating?

- In Lab 1, we used redundant terms to remove glitches
- In practice – wasteful and tedious to remove possibility of glitches in all cases
  - May not be possible
  - May require lots of extra circuitry
- Instead, use synchronous design to avoid problems associated with glitches
  - Glitches may still exist, but we don’t care
  - Why?

## Experiment 2 – Hints

- Do
  - Shift registers need to be continuously shifting
  - Inputs go to control logic, which then sends out signals to other parts of the circuit
  - To distinguish each storage location, each location needs to be assigned to a unique address
  - To identify the correct location to write / fetch, the address of the locations need to be compared against the desired address (SAR)
  - A single inverter to flip clock edge is ok
- Don't
  - Don't put logic on clock (do not gate clock – poor practice in synchronous designs)
  - Don't create full state machine (you will end up duplicating counter)

## Experiment 2 – Hints

- Chips to consider using (you may use others from kit) 7474 (dual D-flip flop), 7493, 74169, or 74193 (counter), 74153 and 74157 (multiplexors), 7485 (comparator) others…
- Will at least need to use 2 x 74194 (4-bit shift register) for memory
- Although your final design may not use parallel I/O in shift register, you may use them for debugging purposes (only need 8 LEDs to see entire contents of memory)
- **Be aware that some chips have additional letters in the part number, e.g. 74169<u>A</u> which may change pinout**
- Will probably need some additional ICs for debugging
- Main circuit will need to be disassembled at end of demo, but portion relating the mini-switchboard (debugging sections with switches and LED drivers) may be kept