Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (sequential-logic)

SEQUENTIAL LOGIC

sequential-logic:~# digital systems where the outputs are determined by the previous states and the current states of the circuit

GitHub repo: https://github.com/schwarzschlyle/electronics-and-instrumentation/tree/master/flip-flops

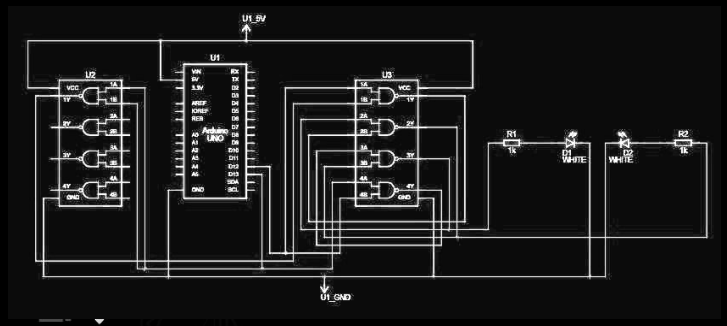
binary-states-convention:~# Throughout the exploration, black-filled LEDs correspond to a HIGH digital state while white-filled LEDs correspond to a LOW digital state

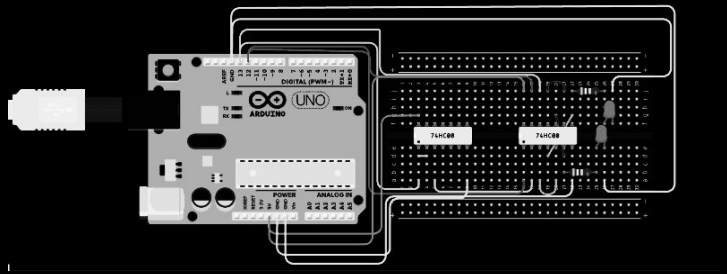
Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (sequential-logic)

d-flip-flops:~# Sequential circuits are capable of “memorizing” data from the previous states. One of the basic sequential circuits is the D Flip flop; such circuit memorizes data input and reproduces the state as an output at a later time triggered by the edge of a clock pulse. To see this, we aim to do the following:

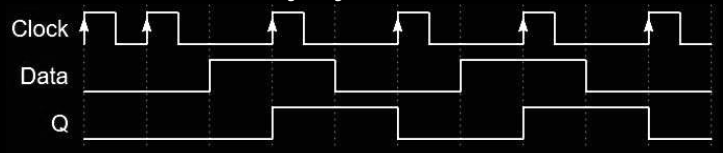
$ construct a D Flip flop circuit from NAND gates,

$ verify the constructed D Flip flop truth values





Timing diagram not to scale



Clock	1	1	1	1	1
Data	0	1	0	1	0
Q	1	1	0	1	0
	phase 1	phase 2	phase 3	phase 4	phase 5

The truth value of a D Flip flop can be summarized as follows: Q follows D when triggered by the rising edge of the clock. We aim to test this via the following phases.

D Flip flop logic test

```
...
digitalWrite(data, LOW); // Phase 1
digitalWrite(clock, HIGH);
delay(1000);
digitalWrite(clock, LOW);
delay(1000);
digitalWrite(clock, HIGH);
delay(1000);
digitalWrite(clock, LOW);
delay(1000);
digitalWrite(data, HIGH); // Phase 2
delay(2000);
digitalWrite(clock, HIGH);
delay(10);
digitalWrite(clock, LOW);
delay(10);
digitalWrite(data, LOW); // Phase 3
delay(2000);
digitalWrite(clock, HIGH);
delay(10);
digitalWrite(clock, LOW);
delay(10);
...
```

```
...
digitalWrite(data, HIGH); // Phase 4
delay(2000);
digitalWrite(clock, HIGH);
delay(10);
digitalWrite(clock, LOW);
delay(10);
digitalWrite(data, LOW); // Phase 5
delay(2000);
digitalWrite(clock, HIGH);
delay(10);
digitalWrite(clock, LOW);
delay(10);
}
…
```

phase-1:~# Clock oscillates as HIGH-LOW-HIGH-LOW. Data is set to LOW; hence, Q is set to LOW.

phase-2:~# Data is set to HIGH and Clock is triggered. Hence, at the rising edge of the Clock pulse, Q rises to a HIGH state.

phase-3:~# Data is set to LOW and Clock is triggered. Hence, at the rising edge of the Clock pulse, Q falls to a LOW state.

phase-4:~# Again, data is set to HIGH and Clock is triggered. Hence, at the rising edge of the Clock pulse, Q rises to a HIGH state.

phase-5:~# Again, data is set to LOW and Clock is triggered. Hence, at the rising edge of the Clock pulse, Q falls to a LOW state.

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (sequential-logic)

d-flip-flops:~# code it up!

codes-used

D Flip flop logic test

```
/*
Tests D Flip flop logic validity by the following state cycle:

Phase 1: Sets data to LOW and clock cycles from HIGH to LOW to HIGH to LOW for
4 seconds with 1 second interval

Phase 2: Set data to HIGH and, after 2 seconds, trigger the clock HIGH then
back to LOW with negligible delay

Phase 3: Set data to HIGH and, after 2 seconds, trigger the clock HIGH then
back to LOW with negligible delay

Phase 4: Repeat phase 2

Phase 5: Repeat phase 3
*/


int data = 13;
int clock = 12;

void setup() {
pinMode(data, OUTPUT);
pinMode(clock, OUTPUT);
digitalWrite(data, LOW);
}


void loop() {
digitalWrite(clock, HIGH); // Phase 1
delay(1000);
digitalWrite(clock, LOW);
delay(1000);
digitalWrite(clock, HIGH);
delay(1000);
digitalWrite(clock, LOW);
delay(1000);
digitalWrite(data, HIGH); // Phase 2
delay(2000);
digitalWrite(clock, HIGH);
delay(10);
digitalWrite(clock, LOW);
delay(10);
```

```
digitalWrite(data, LOW); // Phase 3
delay(2000);
digitalWrite(clock, HIGH);
delay(10);
digitalWrite(clock, LOW);
delay(10);
digitalWrite(data, HIGH); // Phase 4
delay(2000);
digitalWrite(clock, HIGH);
delay(10);
digitalWrite(clock, LOW);
delay(10);
digitalWrite(data, LOW); // Phase 5
delay(2000);
digitalWrite(clock, HIGH);
delay(10);
digitalWrite(clock, LOW);
delay(10);
}
```

Timing diagram construction code

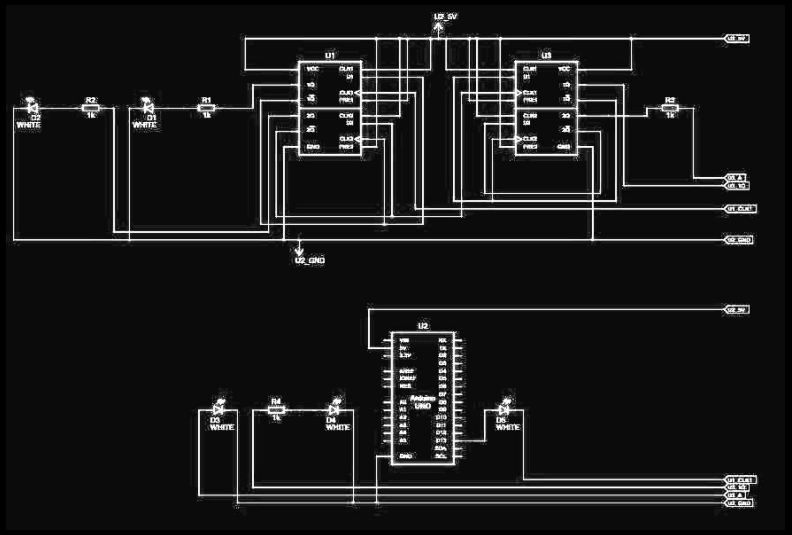
```
{ signal: [
  { name: "Clock", wave: 'P.lPlPlPlP' },
  { name: "Data", wave: 'l.h.l.h.l.' },
  { name: "Q", wave: 'l..h.l.h.l' },
]}


// link: https://wavedrom.com/editor.html
```

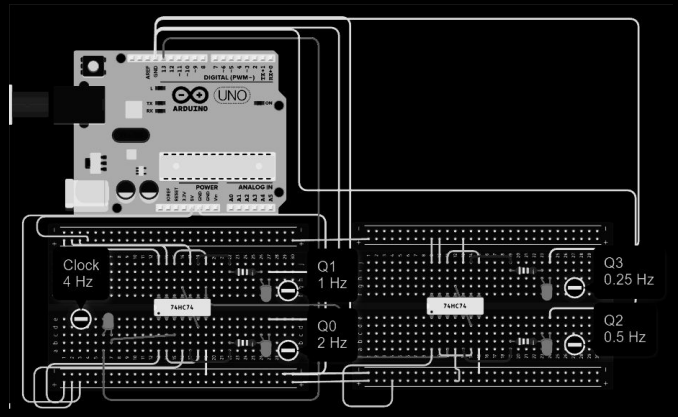
Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (sequential-logic)

frequency-divider:~# One useful applications of sequential circuit is as a counter. A common example is the application to digital watches. Using a crystal oscillator to produce a stable timing waveform of 32,768 Hz, an exact power of 2, the waveform can be counted via a frequency divider. To see how to implement a frequency divider from D flip flops, we aim to do the following:

$ construct a frequency divider circuit from D Flip flops

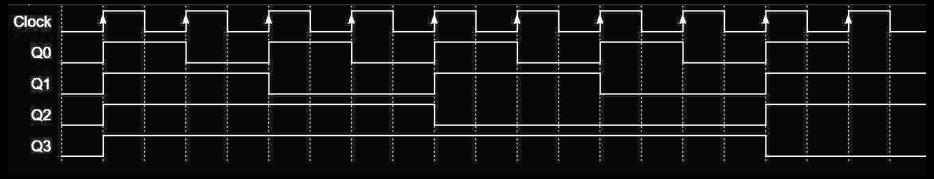
$ verify the constructed frequency divider truth values






```
    Frequency divider logic test
--------------------------------------
...
digitalWrite(clock, LOW);
delay(250);
digitalWrite(clock, HIGH);
delay(250);
...
--------------------------------------
```

The shown circuit is an example of a frequency divider. Observe the the clock pulse fed with delay(250) has a corresponding frequency of 4 Hz - that is, 4 blinks per second. The next output blinks at a frequency of 2 Hz. The following output signal has a frequency of 1 Hz, 0.5 Hz, and finally, 0.25 Hz, respectively.



This is an example of one class of sequential circuits - the counter. Counters are used to count events via clock ticks. Specifically, in this 4-bit counter configuration, the truth table displays a modulo-16 logic counter where the binary number given by the Q0 as the least significant bit and Q3 as the most significant bit ranges from 0 to 15.

This is an essential element of a digital watch. Using a crystal oscillator with timing waveform of 32,768 Hz, a 15-stage frequency divider is used to produce 1 Hz signal for driving a stepper motor for either analogue hand display or a digital display.

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (sequential-logic)

frequency-divider:~# code it up!

codes-used

Frequency divider logic test

```
/*
The input clock waveform is a 4 Hz square wave. Hence, we expect the resulting
array of output from each flip-flop to be 2 Hz, 1 Hz, 0.5 Hz, and 0.25 Hz,
respectively.
*/

int clock = 13;

void setup() {
pinMode(clock, OUTPUT);
}
void loop() {
digitalWrite(clock, LOW);
delay(250);
digitalWrite(clock, HIGH);
delay(250);
}
```

Frequency divider timing diagram code

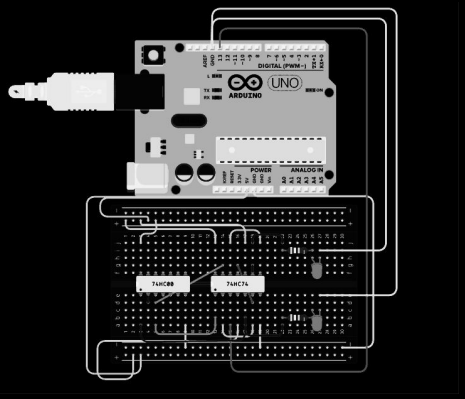
```
{ signal: [
  { name: "Clock", wave: 'lHlHlHlHlHlHlHlHlHlHlH' },
  { name: "Q0", wave: 'lh.l.h.l.h.l.h.l.h.' },
  { name: "Q1", wave: 'lh...l...h...l...h...' },
  { name: "Q2", wave: 'lh.......l.......h.......l........h........'},
  { name: "Q3", wave: 'lh............l........h........l.h.'},
]}
```

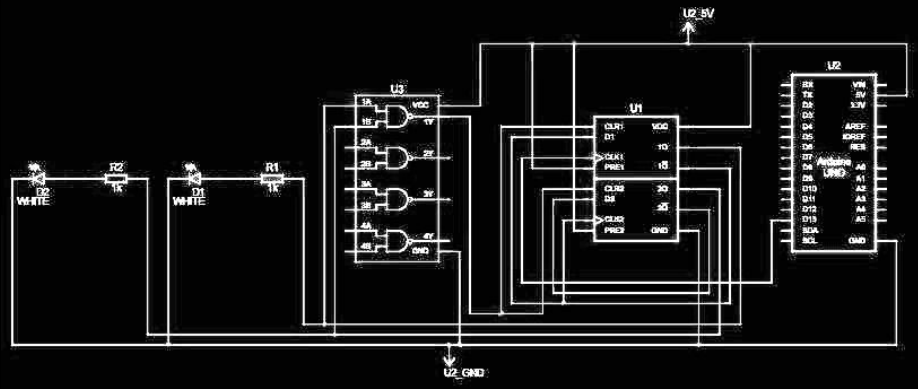
Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (sequential-logic)

2-bit-counter:~# Now, we aim to construct a sequential circuit which is able to track the previous states and “count” up in binary. Using a series of 2 D flip flops with synchronized clocks, the circuit aims to power up a set of 2 LEDs corresponding to the 2-bit data where, at each clock pulse, the binary value increases from 0 to 3. This can be done by doing the following tasks:

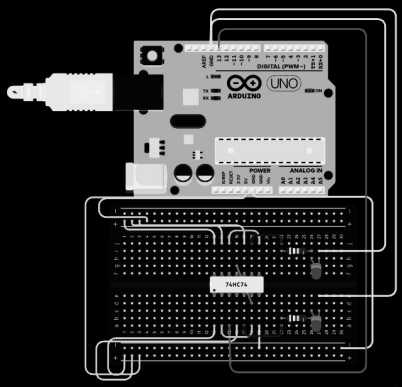
$ construct a 2-bit counter using to D flip flops,

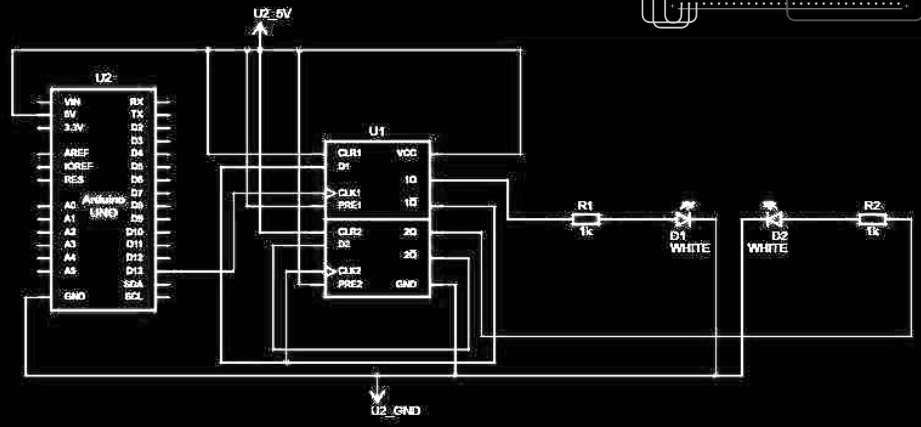
$ check the validity of the truth values of the constructed 2-bit counter






```
2-bit counter logic test
_____________________________________________
...
digitalWrite(clock, LOW);
delay(500);
digitalWrite(clock, HIGH);
delay(500);
...
_____________________________________________
```





Here, we can see another form of a counter - the 2-bit counter. At each rising edge of the clock, the LED display (above circuit) cycles through the following states (00) > (01) > (10) > (11), corresponding to an increasing binary value states at each clock pulse and returning to the original state. Hence, the circuit serves as a counter capable of counter through binary values 0 to 3 then returning to zero.

The circuit can be modified in such a way that the 2-bit counter returns to the original state after reaching the 10 binary state. This can be done by feeding the signal to a NAND gate connected to the master reset pin of the main 2-bit counter circuit. Hence, reaching a state of (11) activates the NAND gate and gives a digital output of LOW. Since the master reset pin is active low, the circuit clears all data and resets to the initial state of (00). Hence, the modified circuit (left circuit) cycles only through the following states (00) > (01) > (10) and the back to (00).

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (sequential-logic)

2-bit-counter:~# code it up!

codes-used

2-bit counter logic test

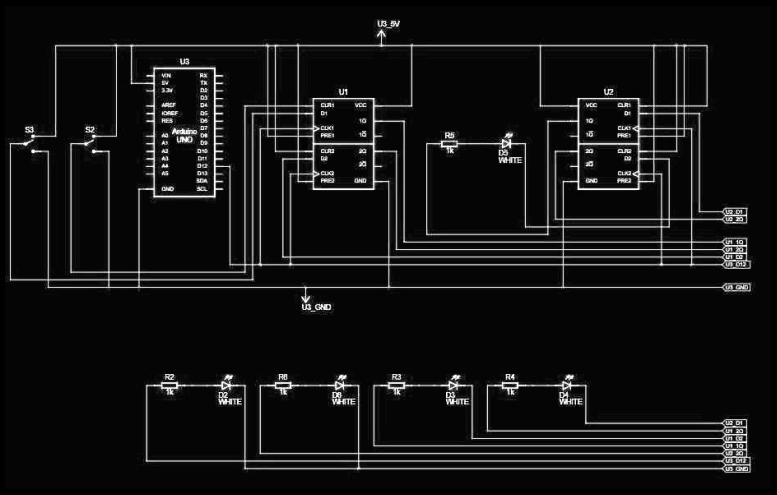
```
/*

*/

int clock = 13;

void setup() {
pinMode(clock, OUTPUT);
digitalWrite(clock, LOW);
digitalWrite(clock, HIGH);
digitalWrite(clock, LOW);
digitalWrite(clock, HIGH);

}
void loop() {
digitalWrite(clock, LOW);
delay(500);
digitalWrite(clock, HIGH);
delay(500);
}
```

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (sequential-logic)

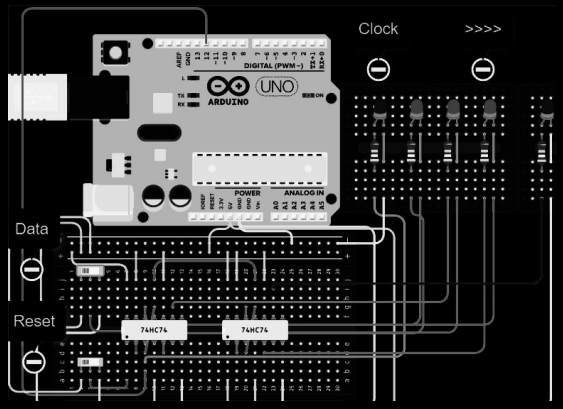
shift-register:~# Finally, sequential circuits are used for long-distance communication where parallel (or serial) words of information are converted to parallel (or serial) to be sent down in a more efficient manner. Here, we can see how a serial data stream can be used to display a parallel data word by shifting the data through the register here as displayed by a series of LEDs. To see this, we aim to do the following tasks:

$ construct a shift register from D flip flops,

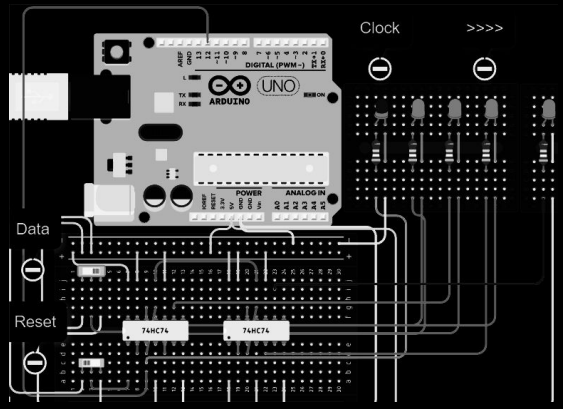
$ verify the constructed shift register truth values by entering the words (1), (11), and (1101)



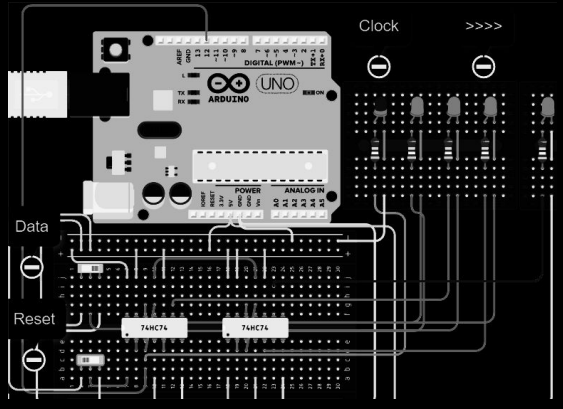
```
Shift register logic test
------------------------
...
digitalWrite(clock,
HIGH);
delay(1000);
digitalWrite(clock, LOW);
delay(1000);
}
...
------------------------
```



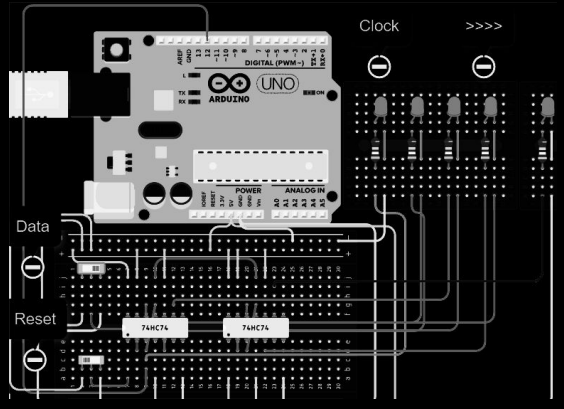
resetting:~# At each clock pulse, the circuit shifts the data as a series of D flip flops output. We start by clearing all data by grounding the active low master reset pin. Resetting clears the register setting the LED to LOW states.



word->11:~# Here, we enter the word (11) through the register by connecting the data to 5V before a clock pulse, maintaining it at 5V until the next clock pulse and turning it off after. Observe that two digital HIGH states “enters” the register at the left and exits at the right.



word->1:~# Now the the register is cleared, we shift the word (1) across the register. This can be done so by connecting the data to 5V before the clock pulse and grounding it after the clock pulse. Observe that the word (1) can be seen shifting through the register. Of course, the shift from data in to Q3 takes four clock pulses.



word->1101:~# By carefully turning off the data states at the right clock pulse timing, one can enter an arbitrary series of binary word to the register. Here, a binary word (1101) was entered which can be seen as digital HIGH states entering from the left and exiting to the right.

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (sequential-logic)

shift-register:~# code it up!

codes-used

Shift register logic test

```
/*

*/

int clock = 12;

void setup() {
pinMode(clock, OUTPUT);


}
void loop() {
digitalWrite(clock, HIGH);
delay(1000);
digitalWrite(clock, LOW);
delay(1000);
}
```