LAB REPORT: 5

Name: Arghya Roy

Roll Number: 2021115008

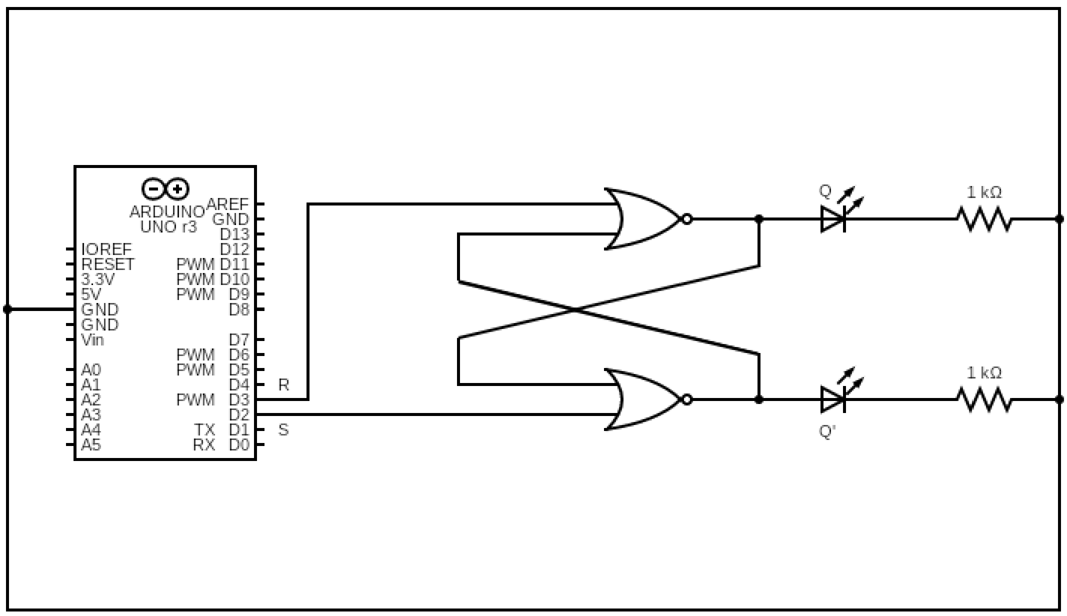
Group: 8

Part A: SR Latch

Aim/Objective of the experiment: To design an SR latch

Electronic components used: 1 Arduino board, two 1 kilo ohm resistors, 2 LEDs, 1 breadboard, 2 NOR gates(74HC02), 1 pushbutton, wires

Reference Circuit:



Procedure:

1. A NOR latch is assembled using two NOR gates, as shown in the reference figure above, on the breadboard.
2. R and S are taken as inputs from the user.
3. Outputs Q and Q' are displayed on two LEDs.
4. An Arduino code is written to give different combinations of inputs as input.
5. The observed outputs of the latch are tabulated.

The code:

```
int r,s;

void setup()
{
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  Serial.begin(9600);
}

void loop()
{

  if(Serial.available()>0)
  {
    s=Serial.read();
    s=s-'0';

    digitalWrite(2,s);
  }

  if(Serial.available()>0)
  {
    r=Serial.read();
    r=r-'0';

    digitalWrite(3,r);
  }
  delay(100);

}
```

Conclusion:

S	R	Q	Q’
0	1	0	1
0	0	0	1
1	0	1	0
0	0	1	0
0	1	0	1
1	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0
0	0	0	1
1	0	1	0
1	1	0	0
0	0	0	1
0	1	0	1
1	1	0	0
0	0	0	1

Q. When given the above inputs, explain till when the latch can be expected to operate correctly and why?

Ans. The latch can be expected to operate correctly until both S and R inputs are 1. The 11 input is a forbidden input for NOR implementation of SR latch since if it followed by a 00 input, we may not get the previous state though that is expected.

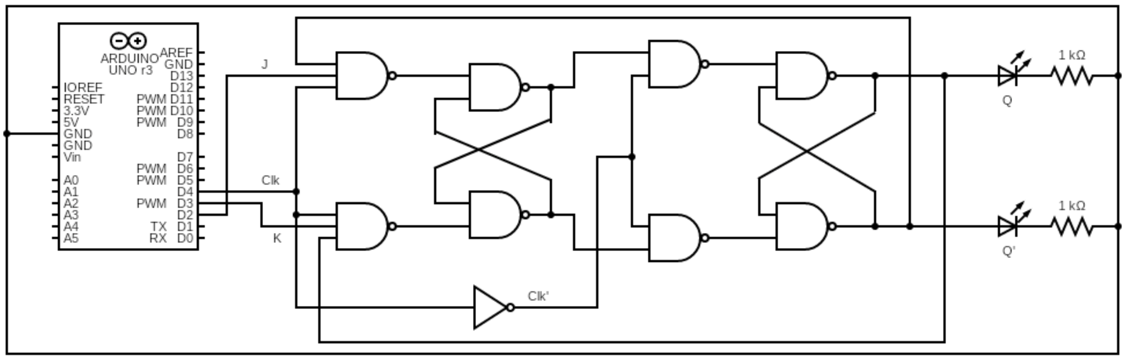
TinderCAD simulation: https://www.tinkercad.com/things/5K34xg6Fmvq-arghya-lab-5-part-a-sr-latch/

Part B: JK Master-Slave Flip-Flop

Aim/Objective of the experiment: To design a JK Master-Slave Flip-Flop

Electronic components used: 1 Arduino board, two 1 kilo ohm resistors, 2 LEDs, 3 breadboards, 1 triple 3-input NAND gate (74HC10), 1 hex inverter(74HC04), 5 Quad NAND gates(74HC00), 1 push button, wires

Reference Circuit:



Procedure:

1. The circuit is set up, as shown in the reference figure above, on the breadboard.
2. Power supply is added to it.
3. An Arduino code is written to give different combinations of inputs as input.
4. The observed outputs of the JK flip-flop are tabulated.

   The code used:

```
int j,k,c;

void setup()
{
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  c=1;
  if(Serial.available()>0)
  {
    j=Serial.read();
    j=j-'0';

    digitalWrite(2,j);
  }

  if(Serial.available()>0)
  {
    k=Serial.read();
    k=k-'0';

    digitalWrite(3,k);

     digitalWrite(4,c);
     delay(100);
     c=0;
     digitalWrite(4,c);

  }

  delay(100);
}
```

Conclusion:

J	K	Q	Q’
1	0	1	0
0	0	1	0
0	1	0	1
1	0	1	0
0	1	0	1
0	0	0	1
1	1	1	0
0	0	1	0
1	0	1	0
1	1	0	1
0	0	0	1
0	1	0	1
1	1	1	0
0	0	1	0

Thus, the outputs are tabulated and verified.

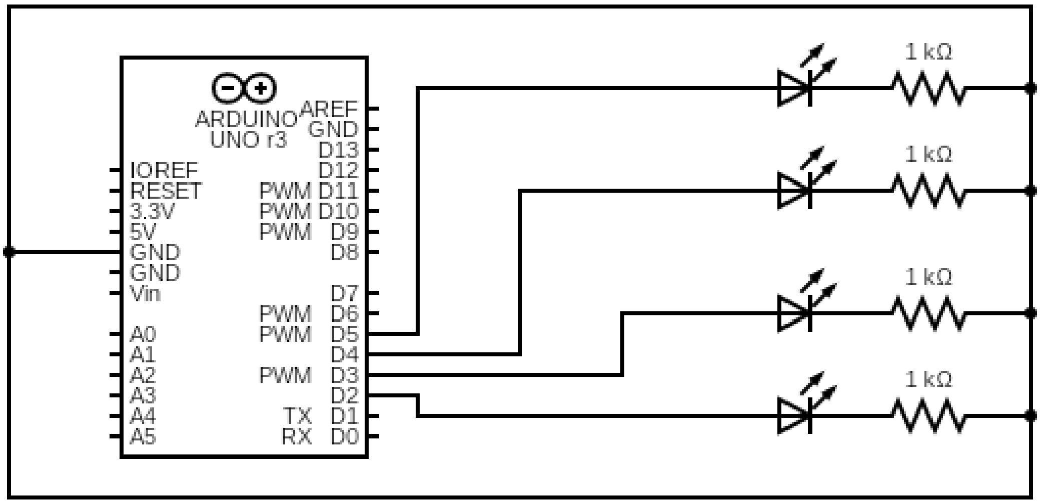
TinkerCAD Simulation: https://www.tinkercad.com/things/2pjyaPqrPTw-arghya-lab-5-part-b-jk-flip-flop/

Part C: 4-bit Up-Down Counter

Aim/Objective of the experiment: To implement a 4-bit counter

Electronic components used: 1 Arduino board, four 1 kilo ohm resistors, 4 LEDs, 1 breadboard, wires

Reference Circuit:



Procedure:

1. The circuit is set up, as shown in the reference figure above, on the breadboard.
2. An Arduino code is written to implement a 4-bit counter with LEDs using the Timer library.
3. Each of the bit outputs are represented by an LED.

The code used:

```
/*
 *     This program is free software; you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation; either version 2 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program; if not, write to the Free Software
 *     Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 *     MA 02110-1301, USA.
 */

/*  * * * * * * * * * * * * * * * * * * * * * * * * * * *
 Code by Simon Monk
 http://www.simonmonk.org
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef Event_h
#define Event_h

#include <inttypes.h>

#define EVENT_NONE 0
#define EVENT_EVERY 1
#define EVENT_OSCILLATE 2

class Event
{

public:
  Event(void);
  void update(void);
  void update(unsigned long now);
  int8_t eventType;
  unsigned long period;
  int repeatCount;
  uint8_t pin;
  uint8_t pinState;
  void (*callback)(void);
  unsigned long lastEventTime;
  int count;
};

#endif
```

```
#ifndef Timer_h
#define Timer_h

#include <inttypes.h>


#define MAX_NUMBER_OF_EVENTS (10)

#define TIMER_NOT_AN_EVENT (-2)
#define NO_TIMER_AVAILABLE (-1)

class Timer
{

public:
  Timer(void);

  int8_t every(unsigned long period, void (*callback)(void));
  int8_t every(unsigned long period, void (*callback)(void), int repeatCount);
  int8_t after(unsigned long duration, void (*callback)(void));
  int8_t oscillate(uint8_t pin, unsigned long period, uint8_t startingValue);
  int8_t oscillate(uint8_t pin, unsigned long period, uint8_t startingValue, int repeatCount);

  /**
   * This method will generate a pulse of !startingValue, occuring period after the
   * call of this method and lasting for period. The Pin will be left in !startingValue.
   */
  int8_t pulse(uint8_t pin, unsigned long period, uint8_t startingValue);

  /**
   * This method will generate a pulse of pulseValue, starting immediately and of
   * length period. The pin will be left in the !pulseValue state
   */
  int8_t pulseImmediate(uint8_t pin, unsigned long period, uint8_t pulseValue);
  void stop(int8_t id);
  void update(void);
  void update(unsigned long now);

protected:
  Event _events[MAX_NUMBER_OF_EVENTS];
  int8_t findFreeEventIndex(void);

};
#endif

//// YOUR CODE STARTS HERE

Timer t;
/*
int pin0 = 2;
int pin1 = 3;
```

```
int pin2 = 4;
int pin3 = 5;
*/
int flag = 0;
int eventId1,eventId2,eventId3,eventId4;
void setup() {
   Serial.begin(9600);
   pinMode(2, OUTPUT);
     pinMode(3, OUTPUT);
     pinMode(4, OUTPUT);
     pinMode(5, OUTPUT);
     eventId1 = t.oscillate(2, 500, flag);

  //if (eventId1<0){Serial.println("Could not initialize timer");}
     eventId2 = t.oscillate(3, 1000, flag);
   //  if (eventId2<0){Serial.println("Could not initialize timer");}
     eventId3 = t.oscillate(4, 2000, flag);
     //if (eventId3<0){Serial.println("Could not initialize timer");}
     eventId4 = t.oscillate(5, 4000, flag);
    // if (eventId4<0){Serial.println("Could not initialize timer");}

     t.after(8000, after);
}
void after()
{
     t.stop(eventId1);
     t.stop(eventId2);
     t.stop(eventId3);
     t.stop(eventId4);
     flag = !flag;
     //Serial.println(flag);
     eventId1 = t.oscillate(2, 500, flag,8);

  //if (eventId1<0){Serial.println("Could not initialize timer");}
     eventId2 = t.oscillate(3, 1000, flag,4);
   //  if (eventId2<0){Serial.println("Could not initialize timer");}
     eventId3 = t.oscillate(4, 2000, flag,2);
     //if (eventId3<0){Serial.println("Could not initialize timer");}
     eventId4 = t.oscillate(5, 3000, flag,1);
    // if (eventId4<0){Serial.println("Could not initialize timer");}

     t.after(8000, after);
}
// 1 unit of your timer = 500ms in real time
void loop() {
   t.update();
}


// "every" X milliseconds
```

```
//// YOUR CODE ENDS HERE

// For Arduino 1.0 and earlier
// #if defined(ARDUINO) && ARDUINO >= 100
// #include "Arduino.h"
// #else
// #include "WProgram.h"
// #endif

Event::Event(void)
{
	eventType = EVENT_NONE;
}

void Event::update(void)
{
   unsigned long now = millis();
   update(now);
}

void Event::update(unsigned long now)
{
	if (now - lastEventTime >= period)
	{
		switch (eventType)
		{
			case EVENT_EVERY:
				(*callback)();
				break;

			case EVENT_OSCILLATE:
				pinState = ! pinState;
				digitalWrite(pin, pinState);
				break;
		}
		lastEventTime = now;
		count++;
	}
	if (repeatCount > -1 && count >= repeatCount)
	{
		eventType = EVENT_NONE;
	}
}


Timer::Timer(void)
{
}

int8_t Timer::every(unsigned long period, void (*callback)(), int repeatCount)
{
```

```
	int8_t i = findFreeEventIndex();
	if (i == -1) return -1;

	_events[i].eventType = EVENT_EVERY;
	_events[i].period = period;
	_events[i].repeatCount = repeatCount;
	_events[i].callback = callback;
	_events[i].lastEventTime = millis();
	_events[i].count = 0;
	return i;
}

int8_t Timer::every(unsigned long period, void (*callback)())
{
	return every(period, callback, -1); // - means forever
}

int8_t Timer::after(unsigned long period, void (*callback)())
{
	return every(period, callback, 1);
}

int8_t Timer::oscillate(uint8_t pin, unsigned long period, uint8_t startingValue, int repeatCount)
{
	int8_t i = findFreeEventIndex();
	if (i == NO_TIMER_AVAILABLE) return NO_TIMER_AVAILABLE;

	_events[i].eventType = EVENT_OSCILLATE;
	_events[i].pin = pin;
	_events[i].period = period;
	_events[i].pinState = startingValue;
	digitalWrite(pin, startingValue);
	_events[i].repeatCount = repeatCount * 2; // full cycles not transitions
	_events[i].lastEventTime = millis();
	_events[i].count = 0;
	return i;
}

int8_t Timer::oscillate(uint8_t pin, unsigned long period, uint8_t startingValue)
{
	return oscillate(pin, period, startingValue, -1); // forever
}

/**
 * This method will generate a pulse of !startingValue, occuring period after the
 * call of this method and lasting for period. The Pin will be left in !startingValue.
 */
int8_t Timer::pulse(uint8_t pin, unsigned long period, uint8_t startingValue)
{
	return oscillate(pin, period, startingValue, 1); // once
}
```

```
/**
 * This method will generate a pulse of startingValue, starting immediately and of
 * length period. The pin will be left in the !startingValue state
 */
int8_t Timer::pulseImmediate(uint8_t pin, unsigned long period, uint8_t pulseValue)
{
	int8_t id(oscillate(pin, period, pulseValue, 1));
	// now fix the repeat count
	if (id >= 0 && id < MAX_NUMBER_OF_EVENTS) {
		_events[id].repeatCount = 1;
	}
	return id;
}


void Timer::stop(int8_t id)
{
	if (id >= 0 && id < MAX_NUMBER_OF_EVENTS) {
		_events[id].eventType = EVENT_NONE;
	}
}

void Timer::update(void)
{
	unsigned long now = millis();
	update(now);
}

void Timer::update(unsigned long now)
{
	for (int8_t i = 0; i < MAX_NUMBER_OF_EVENTS; i++)
	{
		if (_events[i].eventType != EVENT_NONE)
		{
			_events[i].update(now);
		}
	}
}
int8_t Timer::findFreeEventIndex(void)
{
	for (int8_t i = 0; i < MAX_NUMBER_OF_EVENTS; i++)
	{
		if (_events[i].eventType == EVENT_NONE)
		{
			return i;
		}
	}
	return NO_TIMER_AVAILABLE;
}
```

Conclusion:

Time	A	B	C	D
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	1	1	0
8	0	1	1	1
9	1	0	0	0
10	1	0	0	1
11	1	0	1	0
12	1	0	1	1
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0
16	1	1	1	1
17	1	1	1	1
18	1	1	1	0
19	1	1	0	1
20	1	1	0	0
21	1	0	1	1
22	1	0	1	0
23	1	0	0	1
24	1	0	0	0
25	0	1	1	1
26	0	1	1	0
27	0	1	0	1
28	0	1	0	0
29	0	0	1	1
30	0	0	1	0
31	0	0	0	1
32	0	0	0	0

We observe that the ripple counter first goes up from 0 (0000) to 15 (1111), then goes down from 15 to 0, then goes up again, and this cycle keeps on repeating until the simulation is stopped.

TinkerCAD Simulation: https://www.tinkercad.com/things/9303yjXZhLE-arghya-lab-5-part-c-4-bit-counter/