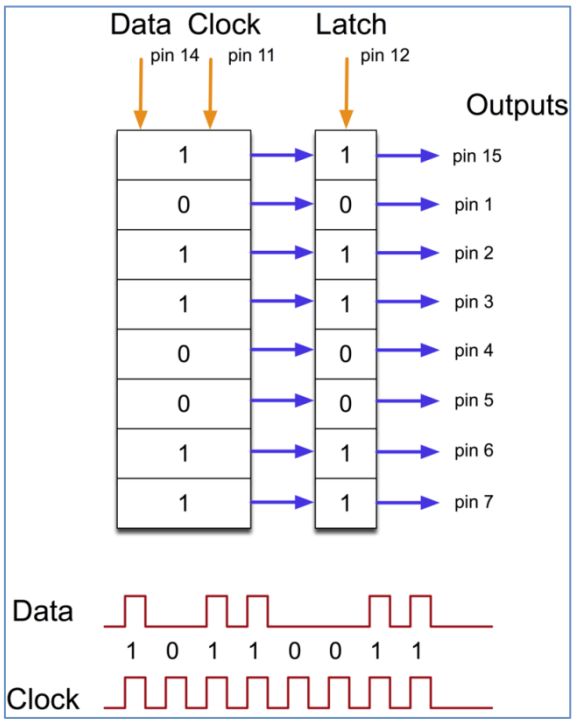
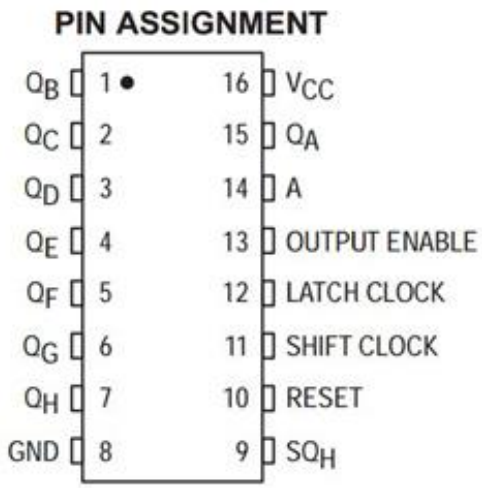
74HC595 8 Bit Schieberegister mit Latch (zu Aufgabenblatt 6)





nichts anschließen
nichts anschließen
Dateneingang (10)
mit GND verbinden
mit storePin verbinden (9)
mit shiftPin verbinden (8)
an +5V anschließen
bzw. mit resetPin verb. (11)
nichts anschließen

Output Enable: Schaltet die Ausgangspins QA bis QH ein (=LOW) oder aus (=HIGH). Dieser Pin sollte also normalerweise mit GND verbunden werden.
Latch Clock: Die im Schieberegister gespeicherten Werte werden mit „HIGH" in das Storage Register übernommen. Dieses Ausgaberegister wird meistens mit **Latch** bezeichnet.
A: Hier stehen die **Daten** (DS) an, die in das Schieberegister übernommen werden sollen.
Shift Clock: Die an Pin A anliegenden Daten werden bei einem Signalwechsel von L auf H in das Schieberegister übernommen.
Reset: Wie der Name schon vermuten lässt wird hier der Inhalt des Schieberegisters mit „LOW" (=GND) zurückgesetzt. Pin 10 muss also normalerweise mit +5V verbunden werden.
SQH: Das tolle an Schieberegistern ist, dass man mehrere davon hintereinander, also in Reihe, schalten kann. Ist das Schieberegister voll so kann man die Daten die sonst sozusagen hinten wieder herausfallen würden in das nächste Schieberegister übergeben.

shiftregister-sketch-1:

Die denkbar einfachste Anwendung ohne Schnick-Schnack ☺.
Wir übernehmen bei jedem Bit der Dateneingabe den Inhalt des Shift-Registers in den Speicher (Latch), damit man den Vorgang beobachten kann. Vorher wird das Shift-Register noch gelöscht.

```
int shiftPin = 8;            // entspricht „clockPin" (=Shift Clock), bei L>H erfolgt Übernahme ins ShiftRegister
int storePin = 9;            // entspricht „latchPin" (=Latch Clock), bei L>H erfolgt Übernahme in den Speicher
int dataPin = 10;            //  die Daten müssen anliegen, bevor die Shift-Clock von L>H geht
int resetPin = 11;           //  zum Rücksetzen des Schieberegisters
int muster[8] = {1,1,0,1,0,0,0,1};    // Dieses Muster soll z.B. ausgegeben werden

void setup() {

pinMode(storePin, OUTPUT);
pinMode(shiftPin, OUTPUT);
pinMode(dataPin, OUTPUT);
pinMode(resetPin, OUTPUT);

digitalWrite(resetPin, LOW);       // die nächsten 4 Anweisungen setzen das Schieberegister.....
digitalWrite(storePin, LOW);       // ......und das Display (LEDs) zurück
digitalWrite(storePin, HIGH);
digitalWrite(resetPin, HIGH);
```

```
for (int i=0; i<8; i++) {              // Array auslesen

delay(1000);                           // damit der Vorgang langsam abläuft
digitalWrite(storePin, LOW);           // storePin wieder auf LOW, Aktion passiert nur bei Wechsel von L auf H
digitalWrite(shiftPin, LOW);           // wie oben
digitalWrite(dataPin, muster[i]);      // Jetzt den Wert der aktuellen Stelle ans Datenpin „A“ (DS) anlegen
digitalWrite(shiftPin, HIGH);          // Übernahme ins Shift-Register
digitalWrite(storePin, HIGH);          // Übernahme in den Speicher
 }                                     // nach 8 Umläufen: Ende der Schleife
 }                                     // Ende Setup

void loop () {
                    // Hier passiert nichts.
}
```

shiftregister-sketch-2:

Im Arduino Sketch 2 läuft ein Zähler von 0 bis 255 und der jeweilige Wert wird in binärer Darstellung auf den 8 LEDs ausgegeben. Zur Übertragung des 8-bit-Wertes in das Schieberegister wird hier der **shiftOut()**-Befehl des Arduino verwendet. Dieser erledigt die Übertragung der einzelnen Bits ins Schieberegister inklusive der notwendigen Flankenwechsel am Shift Pin (=Clock-PIN).
shiftOut() verschiebt ein Daten-Byte ein Bit nach dem anderen. Startet entweder aus dem größten (d. h. dem am weitesten links liegenden) oder aus dem am wenigsten (rechtesten) signifikanten Bit. Jedes Bit wird wiederum auf einen Dateneingang geschrieben, wonach ein Takt gepulst wird (von LOW auf HIGH).
Hinweis: Man muss sicherstellen, dass der Shift-Takt vor dem Aufruf von **shiftOut()** „LOW“ ist. Das geht mit der Anweisung **digitalWrite (clockPin, LOW).**
Syntax: **shiftOut (dataPin, clockPin, bitOrder, value)**
bitOrder: Reihenfolge um die Bits zu verschieben; Entweder **MSBFIRST** oder **LSBFIRST**.
(Das **m**eiste **s**ignifikante **B**it zuerst [**first**]) oder das am wenigsten [**l**etzte] **s**ignifikante **B**it zuerst
value: Wert der zu übertragenden Daten, Angabe als Dezimalzahl oder mit **0b.....** als Binärzahl)

Die Übertragung des Schieberegister-Inhalts an das Latch mit einem Wechsel des Latch-Clock Pin (=storePin) von LOW zu HIGH müssen wir dann noch selbst erledigen, siehe (*) im Sketch.

```
int shiftPin = 8;
int storePin = 9;
int dataPin = 10;
int counter = 0;

void setup() {
 pinMode(storePin, OUTPUT);
 pinMode(shiftPin, OUTPUT);
 pinMode(dataPin, OUTPUT);
}

void loop () {
 digitalWrite(storePin, LOW);     //(*)
 shiftOut(dataPin, shiftPin, MSBFIRST,
                              counter);
 digitalWrite(storePin, HIGH);    //(*)
 delay(500);
 counter ++;
 if (counter > 255) {
 counter = 0;
 }
}
```

```
Binärer Zähler von 0 bis 255.
1 = 0b000000001
2 = 0b000000010
3 = 0b000000011
4 = 0b000000100
.....
usw. Die LEDs zeigen alle Werte von 0
bis 255 an.
```

shiftregister-sketch-3:

Neue Anweisungen: **`bitSet(x,n)`** schreibt eine „1" in einem Bit einer (numerischen) Variablen.
x: Variable, in der ein Bit gesetzt werden soll (hier **`leds`**), n: die Stelle, an der das Bit gesetzt werden soll, beginnend bei 0 für das niedrigstwertige Bit (ganz rechts), siehe auch „Erläuterungen".
Dieser Sketch bewirkt, dass ein Schieberegister „langsam" gefüllt wird und damit immer mehr LEDs leuchten denn ein einmal gesetztes Bit bleibt, es sei denn, es würde mit **`bitClear(x,n)`** wieder gelöscht (auf 0 zurück gesetzt).

```
int clockPin = 8;
int latchPin = 9;
int dataPin = 10;

byte leds = 0;


void setup()
{
 pinMode(latchPin, OUTPUT);
 pinMode(dataPin, OUTPUT);
 pinMode(clockPin, OUTPUT);
}


void loop()
{
  leds = 0;
  updateShiftRegister();   // U-Progr. *)
  delay(500);

  for (int n = 0; n< 8; n++)
  {
    bitSet(leds, n);
    updateShiftRegister();  // U-Progr. *)
    delay(500);
  }
}


void updateShiftRegister()    // *)
{
 digitalWrite(latchPin, LOW);
 shiftOut(dataPin, clockPin, LSBFIRST, leds);
 digitalWrite(latchPin, HIGH);
}
```

Hier ein paar Erläuterungen zum Sketch:

Etwas gewöhnungsbedürftig ist, dass einzelne Bits betrachtet und verschoben werden. Egal in welcher Form wir Zahlen eingeben, unser Rechner wandelt alles in Binärzahlen um.
Wenn die Variable **`leds`** auf Null gesetzt wird, bedeutet das, dass $0_{10} = 00000000_2$ ist, also die dezimale Null dem binären Byte `00000000` entspricht. Wenn kein Index dabei steht ist die Zahl immer im Zehnersystem zu verstehen. Soll es eine Binärzahl sein, müsste man z.B. schreiben: `130 = 0b10000010`. Man notiert `0b` vor der Zahl, damit es hier nicht mit `10`-Millionen-`10` verwechselt werden kann.
Die nebenstehenden Anweisungen bewirken, dass die Variable **`leds`**, die verändert werden soll, zunächst auf Null gesetzt wird. Das Laden und Anzeigen im Shift-Register wird durch ein Unterprogramm (Prozedur) **`updateShiftRegister()`** vorgenommen, das nach unten „ausgelagert" wurde. Das ist hier praktisch, da sie zweimal verwendet wird.
Die Anweisung **`bitSet`** bewirkt, dass in der Schleife nacheinander jeweils ein zusätzliches Bit gesetzt wird. Wenn also die Variable leds vorher den Wert leds= 7 hatte (also **`0b`**`00000111`), dann wird sie nach der Anweisung **`bitSet(leds, 4)`** zu **`0b`**`00001111`, das 4. Bit wurde zusätzlich gesetzt. Nebenbei hat die Variable jetzt den Wert 15.
Im Loop wird also erst das Register gelöscht, so dass alle LEDs aus sind und dann werden durch die Schleife die einzelnen Bits gesetzt und angezeigt.
Man könnte das ausgelagerte Unterprogramm z.B. auch mit „schreibeAufSchiebeRegister();" bezeichnen.