SN74HC595(N) Shift Register

1. About the Module

As the monitoring station should be more or less self-sustainable later on, an important factor to consider is the power consumption of the devices. There are different sources of power losses that drain the batteries of the station over time. One such loss is the current that flows through the VDD pins to GND in the sensors while they are not used. This power loss could be lowered if the power supply would be just activated when the sensor is actually in use.

This could be done using a simple transistor. However, with a transistor, one GPIO pin of the MCU would be used for each device that is powered individually. When the number of sensors increase, the available GPIOs rapidly decrease. The same would be true for directly using GPIOs to power the sensors which should always be avoided; one reasons for that is, that the GPIOs can only supply a low current and do not have as stable voltage levels as the 3.3V pin.

Another method is to use a shift register, such as the SN74HC595 (figure 1). That is a small integrated circuit (IC) which transforms a digital serial input into a digital parallel output. The SN74HC595 contains a shift register and a storage register of the size of 1 byte and possesses 8 parallel output pins. To operate the device 3 GPIO pins of the MCU are needed, one pin for transmitting the data (data, DS, SER), one pin for the clock signal of the shift register (clock, SHCP, SRCLK) and another one as clock signal for the storage register (latch, STCP, RCLK). The advantage of the shift register is that with only 3 GPIO pins, 8 output pins can be controlled at the same time individually. Furthermore, the IC has another output for serial data which can be connected to the serial input (data pin) of another shift register. This allows for the cascading, that means connecting many ICs in series, of the SN74HC595; so, with only 3 pins a large number of outputs can be managed.

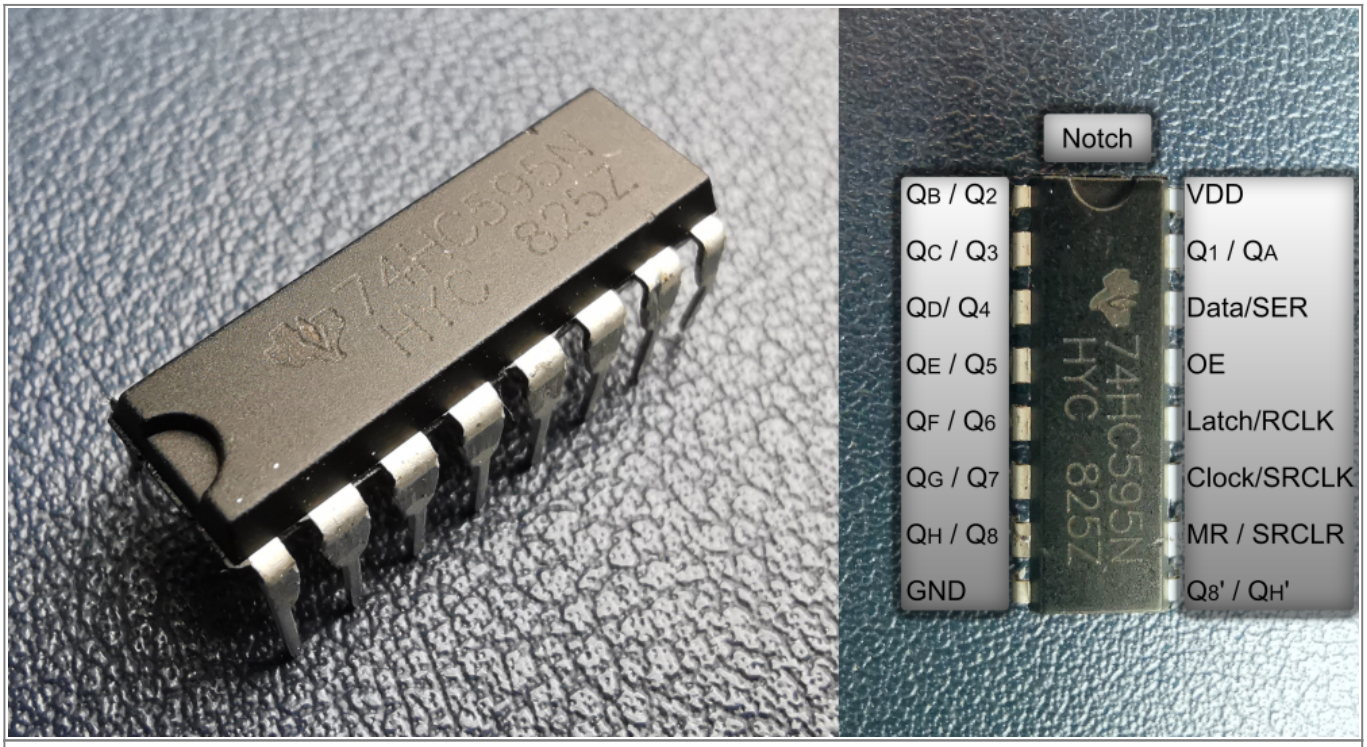

Figure 1 SN74HC595(N) Shift Register and pinout.

2. Working Principle

The MCU issues a clock signal (SRCLK) to the shift register and sends the data (SER) bitwise in 8-bit groups to the shift register (figure 2). If afterwards another byte is sent, the shift register outputs the first byte through the serial output pin (QH'), which may be connected to another shift register receiving the data. So, the shift register always stores the latest byte and if it receives a new byte, it shifts the old one to the next shift register.

The byte stored in the shift register is transferred to the storage when a high voltage level signal is sent to the storage register through the storage register clock input (RCLK). The byte stored in the storage register defines which output pins should be active and which ones should be inactive. When the output enable pin (OE) is set to GND, the byte from the storage register is used as output on the output pins. When the output enable pin is high, the three-state outputs are in high impedance state, blocking all current through the output pins.

It can happen that the outputs from the shift register are random when for example uploading code or resetting the MCU. To reset the pins and stop any high output, the master reset pin (MR, SRCLR) must be shorted to ground. This can be done for example by connecting a >1 uF capacitor between GND and SRCLR and a 10kΩ resistor between SRCLR and VDD. After powering up, the capacitor acts as a short circuit to ground resetting the outputs and as soon as it is charged, it acts as an open circuit and the potential of the master reset is high.

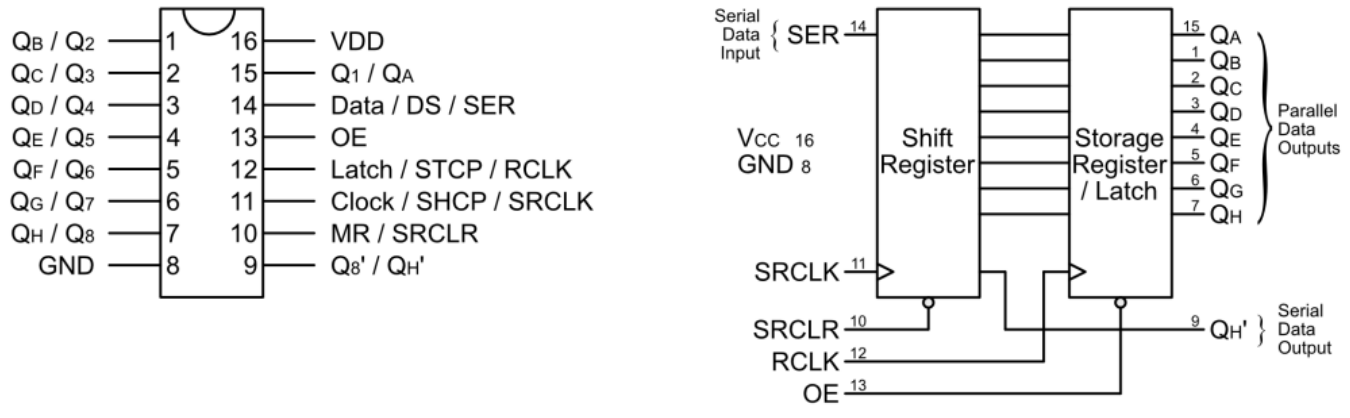

Figure 2 SN74HC595(N) pinout diagram (left) and logic diagram (right), there are different names and abbreviations for the pins depending on the source; the labels in the logic diagram are the official ones from the datasheet (also see table 2).

3. Technical Specifications and Setup of the Module

The ESP32 can supply 3.3 V, which is in the recommended operating range of the SN74HC595 (table 1). The shift register consumes a maximum of 80 µA which occurs at a supply voltage of 6 V while the shift register is actively used; that means during data transmission or transfer from shift register to storage register. Normally the module consumes much less current. The DHT-22 alone consumes 50 µA during idle state continuously which is much higher than the current in the idle shift register. The more sensors are connected to the shift register, the higher the efficiency of the sensor/shift register combination and the more energy can be saved.

It is important to note that the continuous output current should be limited to a maximum of 70 mA.

As the voltage drops when more current is drawn, the current should be limited to about 5 mA per pin. As both the DHT-22 and the DS18B20 only use about 1.5 mA during measuring, this is not an issue.

An overview over the pins and what they do can be found in table 2. When setting up the module the notch helps orienting the shift register in the right direction. The pins were connected the following way:

- VDD to 3.3V
- GND to GND
- OE to GND
- SRCLR to 3.3V
- SRCLK to GPIO 4
- RCLK to GPIO 2
- SER to GPIO 0

Furthermore, a small ceramic capacitor (104 pF) was added between VDD and GND with the legs of the capacitor as close to the pin as possible to reduce the inductance of the wire as much as possible. As mentioned before, the shift register draws most current in very short pulses when the data are shifted, or the byte is transferred to the storage register. In this situation it can be advantageous to have a small capacitor which can supply additional current to keep the voltage from dropping too low.

The parallel outputs can be connected to any device operated at 3.3 V which draws less than 5 mA. To test the working principle of the shift register, the outputs can be connected to red LEDs with a sufficient resistor in series:

$$R=\frac{V_{DD}-U_F}{I_{LED}}=\mathrm{\frac{3.3V-1.7V}{0.005A}=320\Omega}$$

The next bigger resistor (usually 330Ω) should be used. After testing, in the real application, the VDD pin of the DHT-22 and the VDD pins of the DS18B20s were connected to the output pins QB and QC. The power for the sensors was thus supplied through the output pins of the shift register.

Table 1 **SN74HC595(N) specifications**

Module	**SN74HC595(N) Shift Register**
Supply Voltage VDD	2 – 6 V
Operating Temperature	-40 – 125 °C
Power consumption	80 µA (max at 6V)

Table 2 **SN74HC595(N) pin overview and description**

Pin	**Label**	**Alternative Labels**	**Description**
1	QB	Q2	Parallel Output 2
2	QC	Q3	Parallel Output 3
3	QD	Q4	Parallel Output 4
4	QE	Q5	Parallel Output 5
5	QF	Q6	Parallel Output 6
6	QG	Q7	Parallel Output 7
7	QH	Q8	Parallel Output 8
8	GND	-	Ground (0 V)
9	QH’	Q8’	Serial Data Output
10	SRCLR	MR	Shift Register Clear / Master Reset
11	SRCLK	Clock / SHCP	Shift Register Clock

Table 2 **SN74HC595(N) pin overview and description**			
Pin	**Label**	**Alternative Labels**	**Description**
12	RCLK	Latch / STCP	Storage Register Clock
13	OE	-	Output Enable
14	SER	Data / DS	Serial Data Input
15	QA	Q1	Parallel Output 1
16	VDD	-	Power Supply (3.3V)

4. The Code

The following code is to test the function of the shift register with LEDs to see if it is working properly. To see the sketch working, the shift register needs to be connected as described above and red LEDs have to be connected with a 330Ω resistor in series to the pins QA, QB and QC.

4.1 Code for the Shift Register

ShiftRegisterFunction.ino

```
//ESP32 74HC595 Shift Register Test with LEDs

//ESP32 PINS
const int LATCHPIN = 2;                                          //1
const int CLOCKPIN = 4;
const int DATAPIN = 0;

byte sensor_numbers[9] = {                                       //2
  B00000000,  //all off
  B00000001,  //Pin 1
  B00000010,  //Pin 2
  B00000100,  //Pin 3
  B00001000,  //Pin 4
  B00010000,  //Pin 5
  B00100000,  //Pin 6
  B01000000,  //Pin 7
  B10000000,  //Pin 8
};

void setup() {
  pinMode(LATCHPIN ,OUTPUT);                                     //3
  pinMode(CLOCKPIN ,OUTPUT);
  pinMode(DATAPIN ,OUTPUT);
}

void loop() {
  powerSwitch(0);                                                //4
  delay(1000);
```

```
  powerSwitch(1);
  delay(1000);
  powerSwitch(2);
  delay(1000);
  powerSwitch(3);
  delay(1000);
}

//Power Supply Function
void powerSwitch(byte pin){                                    //5
  digitalWrite(LATCHPIN, LOW);                                 //6
  shiftOut(DATAPIN, CLOCKPIN, MSBFIRST, sensor_numbers[pin]); //7
  digitalWrite(LATCHPIN, HIGH);                                //8
}
```

4.2. Explanation of the Code

1. The GPIO for latch, clock and data input were chosen as described before.
2. The array sensor_numbers is used to configure different configurations for which pins should be high. the pattern of 0s and 1s in the binary numbers directly shows which pins are powered and which are not. A 1 means that it is switched on, a 0 means it is switched off. The most significant bit refers to pin QH/Q8 and the least significant bit refers to QA/Q1. Any combination of pins to be powered can be chosen by adjusting the binary numbers in the array accordingly. The elements in an array are zero indexed, so the first element has the index 0, the second has index 1 and so on. Here the elements were chosen such that the index 0 put all outputs low and the index 1 to 8 put only the corresponding pin high.
3. The shift register receives latch, data and clock as input, so the GPIOs need to be configured as outputs.
4. Here the function powerSwitch(), defined at the bottom, is used to switch off all pins and then switch on pin 1, then pin 2, then pin 3 in 1 second intervals and repeat.
5. The function powerSwitch() expects a byte type number as argument which represents the parallel output pin to be switched on.
6. Putting the latch pin at a low voltage level results in the storage register not being actualized while the new byte is shifted into the shift register; this prevents a disturbance of the output pins.
7. The function shiftOut() is already included in the Arduino library and is used for shift registers. As argument, it expects the data pin, the clock pin, the bit order and a value. The pins are given in the int variables in (1). The bit order defines in in which order the bits are shifted out. It can be either MSBFIRST, so the most significant bit is shifted out first, or LSBFIRST, so the least significant bit is shifted out first. When MSBFIRST is chosen, the MSB is pin QH/Q8 and the LSB is pin QA/Q8. Using LSBFIRST switches the order around. The value needs to below or equal to 255 because the function can only shift out data byte-wise. Here the element with the index given in the powerSwitch() argument from the array sensor_number initialized in (2) is used as value.
8. Giving a high signal on the latch pin transmits the fully shifted out byte to the storage register which adjusts the three-state outputs accordingly causing the LEDs/sensors to be turned on.

4.3 Combining the Sketches

This code can be combined with the code from the DHT-22 and the DS18B20s. This way the sensors are only powered when they are supposed to measure and are switched off again afterwards.

The sensors and the shift register need to be connected like explained in their pages, respectively. Additionally, the VDD pin of the DHT-22 must be connected to QB of the shift register and the VDD of both DS18B20s must be connected to QC. Furthermore, if a red LED with a 330Ω resistor in series is connected to QA, it indicates when the sensors finish their measurements with a short blink.

4.3.1 The Code combined

The individual sections of the code are explained in detail in the pages of the DHT-22 and the DS18B20, respectively.

ESP32_Sensors_Combined.ino

```
DHT dht(DHTPIN, DHTTYPE);      //Represents sensor

//Variables
float dht22AirTem = 0;         //takes up new readings + final average
float dht22AirTemSum = 0;      //takes up the sum of readings for
averaging
String dht22Temperature = ""; //gives back Temperature as a String for
MQTT Transmission

float dht22RelHum = 0;         //takes up new readings + final average
float dht22RelHumSum = 0;      //takes up the sum of readings for
averaging
String dht22Humidity = "";     //gives back Humidity as a String for
MQTT Transmission

const uint8_t AveragingNumberDHT22 = 5; //Number of measurements to be
averaged

/*=====================================================================
==================================
 * DS18B20
*=====================================================================
==================================
*/
//Definitions and Pins
const int ONE_WIRE_BUS = 14;      //=GPIO 14 of the ESP32 as 1-Wire Bus

//Libraries and Objects
#include <OneWire.h>            //Library for the 1-Wire protocol
#include <DallasTemperature.h>  //Library for sending commands and
```

```
receiving data

DeviceAddress bottomSensorAddress =
{0x28,0xC4,0xA0,0x51,0x38,0x19,0x01,0xC2};    //Address bottom sensor
(sensor 1, Tape)
DeviceAddress surfaceSensorAddress =
{0x28,0x0B,0xDB,0x60,0x38,0x19,0x01,0xA3};   //Address surface sensor
(sensor 2, w/o Tape)
OneWire oneWire(ONE_WIRE_BUS);          //Object representing the 1-
Wire bus
DallasTemperature DS18B20(&oneWire);  //Object representing the sensors

//Variables
float bottomTem = 0.00;          //for Temperature readings
String bottomTemperature = "";  //for the Sring to be sent using MQTT

float surfaceTem = 0.00;
String surfaceTemperature = "";

const uint8_t AveragingNumberDS18B20 = 5; //Number of measurements to
be averaged

int RESOLUTION = 12;      //Resolution of the sensor
int TCONV = 750;          //Conversion time for resolution = 12
int delayTime = (TCONV/pow(2, (12-RESOLUTION))) + 10; //Necessary delay
time after measuring to get new values, depending on resolution

/*====================================================================
==================================
 * Shift Register
*====================================================================
==================================
*/
//Shift Register Pins
const int LATCHPIN = 2;
const int CLOCKPIN = 4;
const int DATAPIN = 0;

//Shift Register Pin Array
byte sensor_numbers[9] = {
  B00000000,  //all off
  B00000001,  //Pin 1
  B00000010,  //Pin 2
  B00000100,  //Pin 3
  B00001000,  //Pin 4
  B00010000,  //Pin 5
  B00100000,  //Pin 6
  B01000000,  //Pin 7
  B10000000,  //Pin 8
};
```

```
/*=========================================================================
=====================================
 * Setup
*=========================================================================
=====================================
*/

void setup() {
  pinMode(LATCHPIN ,OUTPUT);
  pinMode(CLOCKPIN ,OUTPUT);
  pinMode(DATAPIN ,OUTPUT);

  Serial.begin(115200);
  delay(5000);
  Serial.println("Measurement is starting ...");
  Serial.println("==============================");
}

/*=========================================================================
=====================================
 * Loop
*=========================================================================
=====================================
*/

void loop() {
  Serial.println("Measuring ...");
  powerSwitch(2); //Turn DHT-22 ON
  measureDHTTemHum(AveragingNumberDHT22);
  powerSwitch(1); //LED Blink
  delay(200);
  powerSwitch(3); //Turn DS18B20s ON
  measureDS18B20Tem(AveragingNumberDS18B20);
  powerSwitch(1); //LED Blink
  delay(200);
  Serial.println("DHT-22 Measurement Results: ");
  Serial.println("Temperature:       " + dht22Temperature + " °C");
  Serial.println("Relative Humidity: " + dht22Humidity + " %");
  Serial.println("DS18B20 Measurement Results: ");
  Serial.println("Sensor 1 (Bottom) Measurement:  " + bottomTemperature
+ " °C");
  Serial.println("Sensor 2 (Surface) Measurement: " +
surfaceTemperature + " °C");
  Serial.println("========================================");
}

/*=========================================================================
=====================================
 * Sensor and Shift Register Functions
*=========================================================================
```

```
==================================
*/

//DS18B20 Measurement function
void measureDS18B20Tem (const uint8_t AveragingNumber){
  DS18B20.begin();
  bottomTem = 0;
  surfaceTem = 0;
  bottomTemperature = "";
  surfaceTemperature = "";

  for(byte i = 0; i < AveragingNumber; i++)
  {
    DS18B20.requestTemperatures();
    delay(delayTime);
    bottomTem += DS18B20.getTempC(bottomSensorAddress);
    surfaceTem += DS18B20.getTempC(surfaceSensorAddress);
  }
  bottomTem /= AveragingNumber;
  surfaceTem /= AveragingNumber;

  if(bottomTem<10)
    bottomTemperature = "0";
  bottomTemperature += bottomTem;

  if(surfaceTem<10)
    surfaceTemperature = "0";
  surfaceTemperature += surfaceTem;
}

//DHT22 Measurement function
void measureDHTTemHum (const uint8_t AveragingNumber)
{
  dht.begin();                                          //activate
Sensor
  delay(1000);                                          //to prevent
unstable status (see Datasheet)
  dht22AirTemSum = 0;
  dht22RelHumSum = 0;
  dht22Temperature = "";
  dht22Humidity = "";
  for (byte i = 0; i < AveragingNumber; i++)            //take n
measurements for both
  {
    do {                                                //execute this
at least once
      dht22AirTem = dht.readTemperature();              //measure
Temperature
      dht22RelHum = dht.readHumidity();                 //measure
Humidity
      if (!isnan(dht22AirTem) && !isnan(dht22RelHum))   //if results
```

```
are valid, add them to the sum
      {
        dht22AirTemSum += dht22AirTem;
        dht22RelHumSum += dht22RelHum;
      }
      else
        delay(2000);                                    //if
measurement is invalid, it has to be repeated. Sensor must not heat up
    } while (isnan(dht22AirTem) || isnan(dht22RelHum)); //if results
were invalid, repeat the loop
    if (i < (AveragingNumber - 1))                      //after the
last measurement, there is no delay needed anymore
      delay(2000);
  }
  dht22AirTem = dht22AirTemSum / AveragingNumber;       //averaging the
measurements to get more accurate results
  if(dht22AirTem<10)                                    //if the
Temperature is below 10°C, add a 0 to the front of the String
    dht22Temperature = '0';
  dht22Temperature = dht22Temperature + dht22AirTem;

  dht22RelHum = dht22RelHumSum / AveragingNumber;
  if(dht22RelHum<10)
    dht22Humidity = '0';
  dht22Humidity = dht22Humidity + dht22RelHum;
}

//Shift Register Power Switch
void powerSwitch(byte pin){
  digitalWrite(LATCHPIN, LOW);
  shiftOut(DATAPIN, CLOCKPIN, MSBFIRST, sensor_numbers[pin]);
  digitalWrite(LATCHPIN, HIGH);
}
```

4.3.2 Results

The results of the sensor measurements are printed to the serial monitor:

```
18:42:13.297 -> Measurement is starting ...
18:42:13.297 -> ==============================
18:42:13.330 -> Measuring ...
18:42:27.159 -> DHT-22 Measurement Results:
18:42:27.159 -> Temperature:       27.72 °C
18:42:27.159 -> Relative Humidity: 46.22 %
18:42:27.159 -> DS18B20 Measurement Results:
18:42:27.159 -> Sensor 1 (Bottom) Measurement:  27.85 °C
18:42:27.159 -> Sensor 2 (Surface) Measurement: 27.42 °C
```

```
18:42:27.159 -> ========================================
18:42:27.159 -> Measuring ...
18:42:44.147 -> DHT-22 Measurement Results:
18:42:44.147 -> Temperature:       27.80 °C
18:42:44.147 -> Relative Humidity: 45.06 %
18:42:44.147 -> DS18B20 Measurement Results:
18:42:44.147 -> Sensor 1 (Bottom) Measurement:  27.87 °C
18:42:44.147 -> Sensor 2 (Surface) Measurement: 27.54 °C
18:42:44.147 -> ========================================
```

During this test, all sensors were in equilibrium, measuring the room temperature during a rather hot summer day, and all results are close together, indicating that the sensors are working properly.