Hardware Adventures

Running LED's using the MCP3008 analog-to-digital converter and SN74HC595 shift register

Jan Fojtík
4178068

ICT - Open Learning
Fontys University of Applied Sciences
2023 November

Contents

Revision History

Revision	Date	Author(s)	Description
0.1	3rd November 2023	J. Fojtík	Initial document setup and problem description
0.2	14th November 2023	J. Fojtík	Fixing formatting and addition of explanations for diagrams
0.3	16th November 2023	J. Fojtík	Addition of technical drawings and function table of the shift register.
1.0	17th November 2023	J. Fojtík	Addition of conclusion and preparation of the document for submission.

1 Context

During my 7th semester, I decided to focus on developing my skills in the realm of embedded programming. I want to acquire knowledge of electronic circuits and the skills necessary to assemble them. Given that my prior knowledge of embedded systems only consisted of Semester 1 knowledge of Arduino and, at best, an understanding of the difference between analog and digital input, the plan was to start with bare-bones, low-level, and accessible concepts. After consulting Eric Slaats, my mentor, we have determined what steps I should take to acquire the desired understanding and knowledge and develop a confident level of autonomy in electrical engineering.

I wanted to learn how to utilize Arduino and other microcontroller boards and integrated circuits. Thus, the first assignment was to learn what an MCP3008 analog-to-digital converter and the SN74HC595 shift register are. Since the only reliable information about such integrated circuits is almost exclusively accessible only in the data sheets provided by the manufacturers, this task meant learning how to read such data sheets.

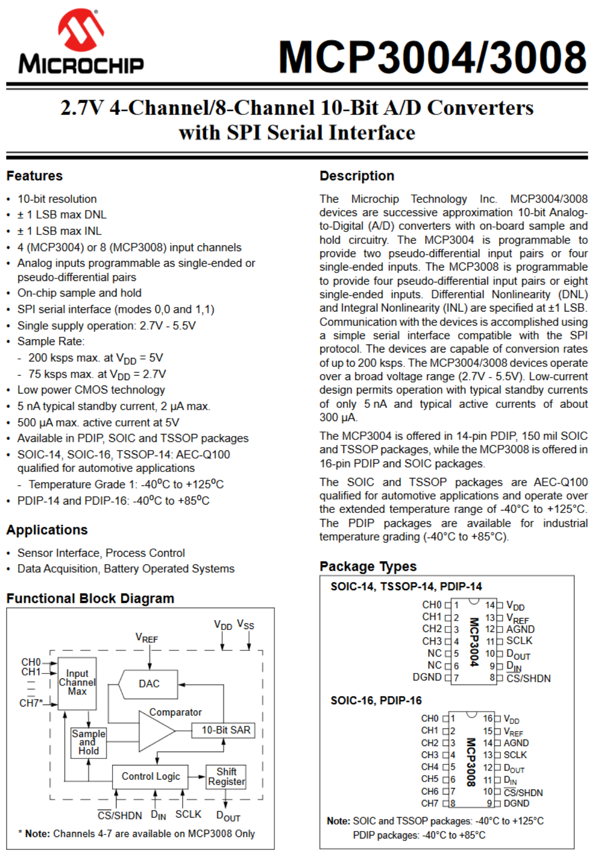
In the following section of the document, I will cover the baby steps of my first read-through of a data sheet. I believe noting these steps will help in the future, similarly to commenting on one's code, which helps understanding when viewing the code after a significant time has passed.

2 Reading the MCP3008 data sheet

2.1 Introductory page

It was immediately extremely overwhelming after opening the data sheet for the first time. Clearly but fairly, the authors of such data sheets assume extensive prior electrical engineering knowledge. While the introductory section of the data-sheet is clearly indexed and relevant information is easily identified, the contents of each section become puzzling quickly, and it is important to remember that it is impossible to process and absorb all the information immediately.

To provide an example in the given context, i.e., learning what an MCP3008 is while also developing the skill of reading a data sheet and extracting relevant information from it, let's have a look at the first page of the data sheet and identify relevant information.



MICROCHIP MCP3004/3008

2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface

Features

- 10-bit resolution
- ± 1 LSB max DNL
- ± 1 LSB max INL
- 4 (MCP3004) or 8 (MCP3008) input channels
- Analog inputs programmable as single-ended or pseudo-differential pairs
- On-chip sample and hold
- SPI serial interface (modes 0,0 and 1,1)
- Single supply operation: 2.7V - 5.5V
- Sample Rate:
    - 200 ksps max. at V_{DD} = 5V
    - 75 ksps max. at V_{DD} = 2.7V
- Low power CMOS technology
- 5 nA typical standby current, 2 μA max.
- 500 μA max. active current at 5V
- Available in PDIP, SOIC and TSSOP packages
- SOIC-14, SOIC-16, TSSOP-14: AEC-Q100 qualified for automotive applications
    - Temperature Grade 1: -40°C to +125°C
- PDIP-14 and PDIP-16: -40°C to +85°C

Applications

- Sensor Interface, Process Control
- Data Acquisition, Battery Operated Systems

Functional Block Diagram

* **Note:** Channels 4-7 are available on MCP3008 Only

Description

The Microchip Technology Inc. MCP3004/3008 devices are successive approximation 10-bit Analog-to-Digital (A/D) converters with on-board sample and hold circuitry. The MCP3004 is programmable to provide two pseudo-differential input pairs or four single-ended inputs. The MCP3008 is programmable to provide four pseudo-differential input pairs or eight single-ended inputs. Differential Nonlinearity (DNL) and Integral Nonlinearity (INL) are specified at ±1 LSB. Communication with the devices is accomplished using a simple serial interface compatible with the SPI protocol. The devices are capable of conversion rates of up to 200 ksps. The MCP3004/3008 devices operate over a broad voltage range (2.7V - 5.5V). Low-current design permits operation with typical standby currents of only 5 nA and typical active currents of about 300 μA.

The MCP3004 is offered in 14-pin PDIP, 150 mil SOIC and TSSOP packages, while the MCP3008 is offered in 16-pin PDIP and SOIC packages.

The SOIC and TSSOP packages are AEC-Q100 qualified for automotive applications and operate over the extended temperature range of -40°C to +125°C. The PDIP packages are available for industrial temperature grading (-40°C to +85°C).

Package Types

SOIC-14, TSSOP-14, PDIP-14

SOIC-16, PDIP-16

Note: SOIC and TSSOP packages: -40°C to +125°C
PDIP packages: -40°C to +85°C

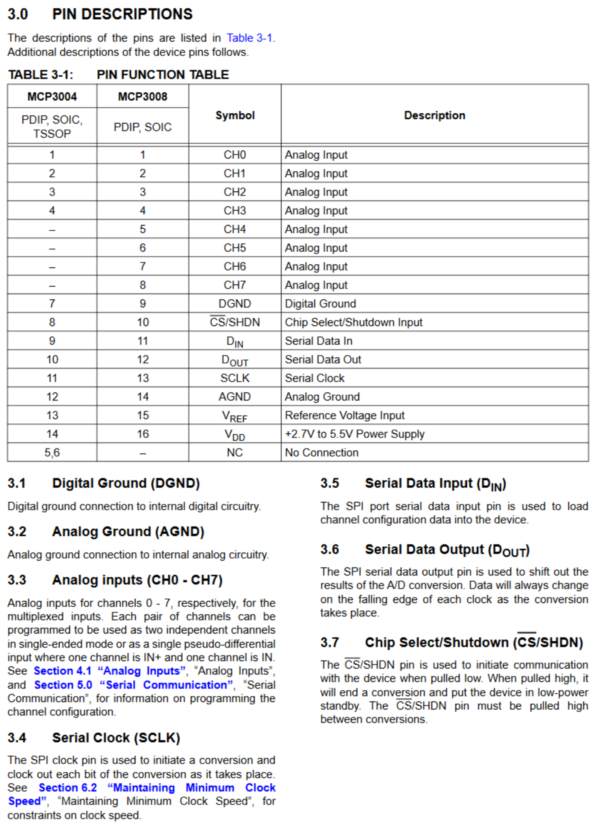
Figure 1: Introduction page

I was interested in discovering what MCP3008 was and its applications, so reading the introductory page was very straightforward. Ignoring the "Features" and most of the "Description" part.

I was interested in the fact that MCP3008 is an analog-to-digital converter and that typical applications usually implement the 3008 as a sensor interface, possibly an input extension for a microcontroller with limited pins. Lastly, the pin-out in the bottom right corner of Figure 1 interested me; this was, however, covered in finer detail further in the data sheet.

2.2 Pin descriptions

Following the introduction page was a plethora of electrical characteristics. These were unimportant to me, as my use case would refrain from taking the capabilities of the 3008 to its limits. For this reason, I have moved forward to the pin definition. As stated before, the pin-out has already been shown on the introductory page. However, no descriptions and detailed information were provided.



3.0 PIN DESCRIPTIONS

The descriptions of the pins are listed in Table 3-1. Additional descriptions of the device pins follows.

TABLE 3-1: PIN FUNCTION TABLE

MCP3004	MCP3008	Symbol	Description
PDIP, SOIC, TSSOP	PDIP, SOIC		
1	1	CH0	Analog Input
2	2	CH1	Analog Input
3	3	CH2	Analog Input
4	4	CH3	Analog Input
–	5	CH4	Analog Input
–	6	CH5	Analog Input
–	7	CH6	Analog Input
–	8	CH7	Analog Input
7	9	DGND	Digital Ground
8	10	$\overline{CS}$/SHDN	Chip Select/Shutdown Input
9	11	D_{IN}	Serial Data In
10	12	D_{OUT}	Serial Data Out
11	13	SCLK	Serial Clock
12	14	AGND	Analog Ground
13	15	V_{REF}	Reference Voltage Input
14	16	V_{DD}	+2.7V to 5.5V Power Supply
5,6	–	NC	No Connection

3.1 Digital Ground (DGND)

Digital ground connection to internal digital circuitry.

3.2 Analog Ground (AGND)

Analog ground connection to internal analog circuitry.

3.3 Analog inputs (CH0 - CH7)

Analog inputs for channels 0 - 7, respectively, for the multiplexed inputs. Each pair of channels can be programmed to be used as two independent channels in single-ended mode or as a single pseudo-differential input where one channel is IN+ and one channel is IN. See **Section 4.1 "Analog Inputs"**, "Analog Inputs", and **Section 5.0 "Serial Communication"**, "Serial Communication", for information on programming the channel configuration.

3.4 Serial Clock (SCLK)

The SPI clock pin is used to initiate a conversion and clock out each bit of the conversion as it takes place. See **Section 6.2 "Maintaining Minimum Clock Speed"**, "Maintaining Minimum Clock Speed", for constraints on clock speed.

3.5 Serial Data Input (D_{IN})

The SPI port serial data input pin is used to load channel configuration data into the device.

3.6 Serial Data Output (D_{OUT})

The SPI serial data output pin is used to shift out the results of the A/D conversion. Data will always change on the falling edge of each clock as the conversion takes place.

3.7 Chip Select/Shutdown ($\overline{CS}$/SHDN)

The $\overline{CS}$/SHDN pin is used to initiate communication with the device when pulled low. When pulled high, it will end a conversion and put the device in low-power standby. The $\overline{CS}$/SHDN pin must be pulled high between conversions.

Figure 2: Pin descriptions

Clearly visible from TABLE 3-1 in Figure 2, column MCP3008, pins 1 through 8 are channel pins. These are used as input extensions, i.e., they can have analog sensors connected to them, further converting the readings from these to digital signals to be processed by an MC. Further pins that I had to familiarize myself with, which, as I came to find later, were very rudimentary and essential in the context of electrical engineering and understanding circuits, were the "control" pins such as the chip select ($\overline{CS}$), clock ($SCLK$), and serial data pins (D_{IN} and D_{OUT}). Finally, there were the power delivery pins, ground (GND), and 5V (VDD).

2.3 Serial communication

Figure 3 describes how the serial communication bits are constructed. This will be important later when I explain how the code was written without using libraries for the MCP3008, as individual bits need to be precisely set; otherwise, the functionality of the 3008 would be hindered.

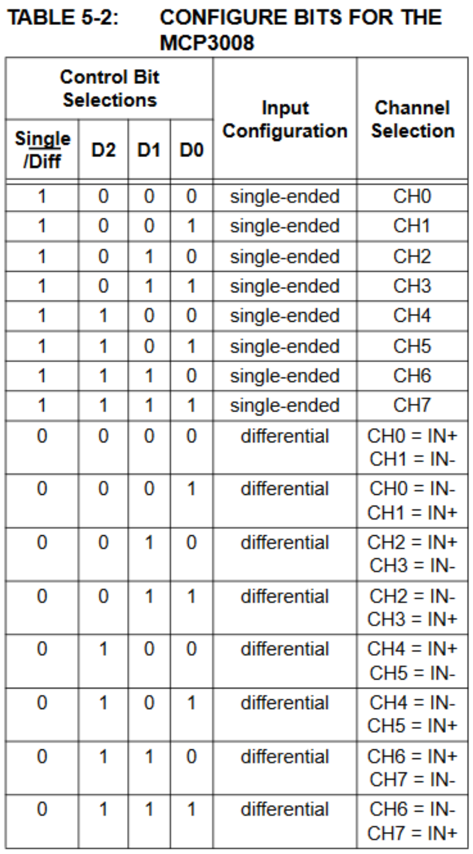


TABLE 5-2: CONFIGURE BITS FOR THE MCP3008

Control Bit Selections				Input Configuration	Channel Selection
Single /Diff	D2	D1	D0		
1	0	0	0	single-ended	CH0
1	0	0	1	single-ended	CH1
1	0	1	0	single-ended	CH2
1	0	1	1	single-ended	CH3
1	1	0	0	single-ended	CH4
1	1	0	1	single-ended	CH5
1	1	1	0	single-ended	CH6
1	1	1	1	single-ended	CH7
0	0	0	0	differential	CH0 = IN+ CH1 = IN-
0	0	0	1	differential	CH0 = IN- CH1 = IN+
0	0	1	0	differential	CH2 = IN+ CH3 = IN-
0	0	1	1	differential	CH2 = IN- CH3 = IN+
0	1	0	0	differential	CH4 = IN+ CH5 = IN-
0	1	0	1	differential	CH4 = IN- CH5 = IN+
0	1	1	0	differential	CH6 = IN+ CH7 = IN-
0	1	1	1	differential	CH6 = IN- CH7 = IN+

Figure 3: Serial communication

In the table above, it is important to take note of the "Control Bit Selections" column, which defines how the four control bits should be set in order to use either a single-ended or differential input configuration on any given channel. Yet again, this will make more sense when the code is explained later.

2.3.1 Single-ended v. differential input configurations

Single-ended input configuration the ADC measures the voltage of a single input channel relative to the *GND* reference; this provides a simpler and more common configuration where the voltage on a single input is compared to the ground. Meanwhile *differential input configuration*, the ADC measures the voltage difference between two input channels, which is useful when you want to measure the voltage across a component or the voltage difference between two points in a circuit; however, this also means that two pins are thus taken up.

2.4 Application information

The following figure illustrates how bytes of data should be transferred to the MCP3008 and what response can be expected based on those. This diagram is already very close to what the implementation looks like. In order to draw a clear parallel, I will start with the chip select pin, the first "row" in the diagram. Note that the pin is set to LOW (denoted by the line next to the $\overline{CS}$ symbol). For now, we can ignore the *SCLK* row. The D_{IN} and D_{OUT} are, however, related to the transmission and dictate the byte structure.

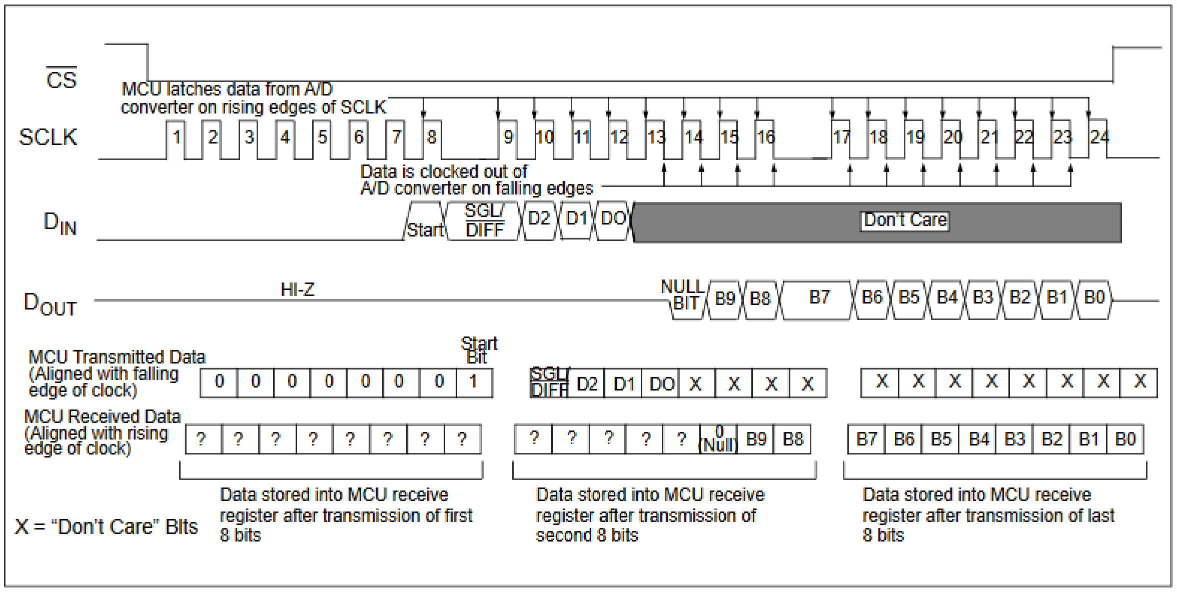

CS
SCLK
MCU latches data from A/D converter on rising edges of SCLK
Data is clocked out of A/D converter on falling edges
DIN
Start
SGL/DIFF
D2
D1
D0
Don't Care
DOUT
HI-Z
NULL BIT
B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
MCU Transmitted Data (Aligned with falling edge of clock)
Start Bit
MCU Received Data (Aligned with rising edge of clock)
0 (Null)
Data stored into MCU receive register after transmission of first 8 bits
Data stored into MCU receive register after transmission of second 8 bits
Data stored into MCU receive register after transmission of last 8 bits
X = "Don't Care" Bits

FIGURE 6-1: *SPI Communication with the MCP3004/3008 using 8-bit Segments (Mode 0,0: SCLK idles low).*

Figure 4: Application information

This section will now be concluded and I will move on to the reading of the data sheet of the so-called "595 shift register".

3 Reading the SN74HC595 data sheet

Along with the pin out in Figure 5, the essential piece of information about the 595 is that it is an 8-bit, meaning it can store eight states at once; serial-in, meaning it cannot read values on its pins, only write to them, parallel-out shift register.

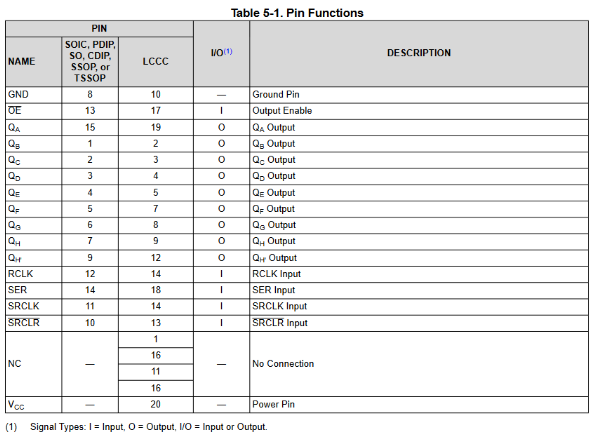


Table 5-1. Pin Functions

PIN			I/O(1)	DESCRIPTION
NAME	SOIC, PDIP, SO, CDIP, SSOP, or TSSOP	LCCC		
GND	8	10	—	Ground Pin
$\overline{OE}$	13	17	I	Output Enable
Q_A	15	19	O	Q_A Output
Q_B	1	2	O	Q_B Output
Q_C	2	3	O	Q_C Output
Q_D	3	4	O	Q_D Output
Q_E	4	5	O	Q_E Output
Q_F	5	7	O	Q_F Output
Q_G	6	8	O	Q_G Output
Q_H	7	9	O	Q_H Output
$Q_{H'}$	9	12	O	$Q_{H'}$ Output
RCLK	12	14	I	RCLK Input
SER	14	18	I	SER Input
SRCLK	11	14	I	SRCLK Input
$\overline{SRCLR}$	10	13	I	$\overline{SRCLR}$ Input
NC	—	1	—	No Connection
		16		
		11		
		16		
V_{CC}	—	20	—	Power Pin

(1) Signal Types: I = Input, O = Output, I/O = Input or Output.

Figure 5: Introduction page

Following this information on connecting the 595 to an MC or another IC, the next important thing regarding its operation is the timing diagram, which explains how data should be written and outputted by the 595.

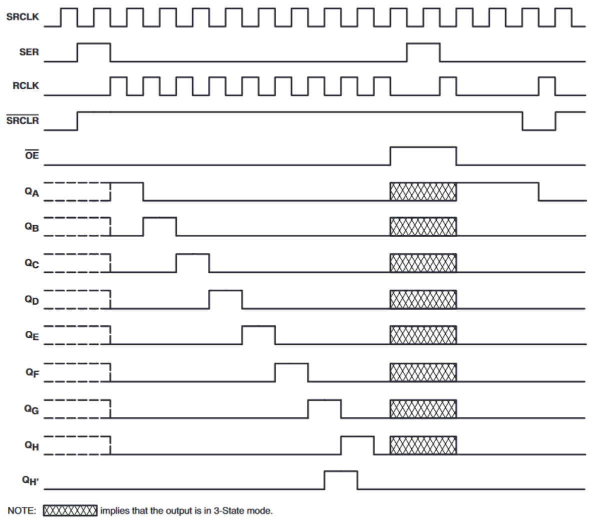


Figure 6-1. Timing Diagram

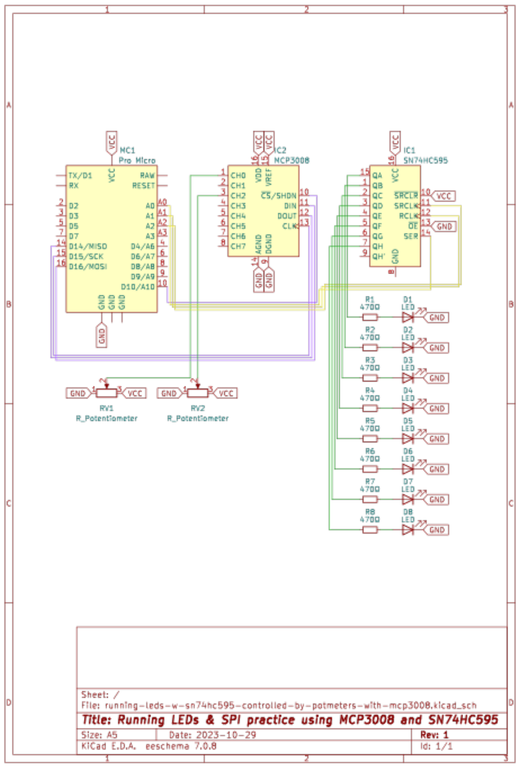
Figure 6: Timing diagram

As can be seen from the diagram, every time the 595 receives a clock pulse, the bits contained in the register are shifted. What is not visible in the diagram is that this is a binary left shift by one position. This means that the 0^{th} bit value is pushed into the 1^{st} , 1^{st} bits value is pushed into the 2^{nd} , etc. Afterward, the 0^{th} bit gets set to whatever the current value on the *SER* (data) pin in. On the rising edge of the pulse, either 1 or 0 is pushed into the shift register.

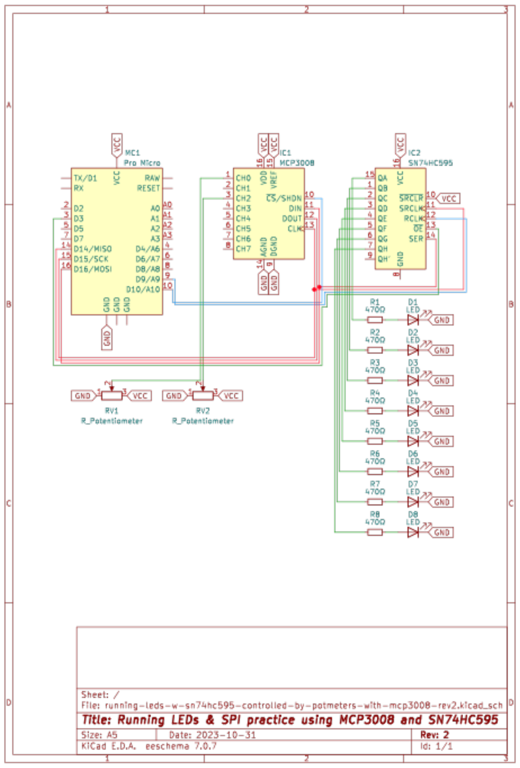
4 Realization

4.1 SPI implementation assembly

As noted before, I was interested in understanding the control pins of each integrated circuit I investigated. This was because of understanding how to implement SPI - Serial Peripheral Interface. I have, however, graciously managed to fail to execute this in the very first attempt. As shown in Figure 7(a), the 595 and 3008 have their own separate *CLK* and *DATA* pins. The 3008's D_{IN}, D_{OUT} and *CLK* are connected to the Pro Micro's *D16/MOSI*, *D14/MISO* and *D15/SCK* respectively. The $\overline{CS}$ pin is wired to the Pro Micro's *D10*. Meanwhile, the 595's control pins *SRCLK*, *RCLK* and *SER* are wired to the Pro Micro's *A2*, *A1* and *A0* respectively. The $\overline{OE}$ and $\overline{SRCLR}$ pins are connected to *GND* and *VCC* of the Pro Micro, as they are active low pins. The $\overline{OE}$ pin needs to be pulled LOW; otherwise, the shift register output would be disabled. Meanwhile, the $\overline{SRCLR}$ must be pulled HIGH; otherwise, the shift register's content would be rest to zeroes.


Sheet: /
File: running-leds-w-sn74hc595-controlled-by-potmeters-with-mcp3008.kicad_sch
Title: Running LEDs & SPI practice using MCP3008 and SN74HC595
Size: A5
Date: 2023-10-29
Rev: 1
KiCad E.D.A. eeschema 7.0.8
Id: 1/1

(a) Revision 1 of the circuit


Sheet: /
File: running-leds-w-sn74hc595-controlled-by-potmeters-with-mcp3008-rev2.kicad_sch
Title: Running LEDs & SPI practice using MCP3008 and SN74HC595
Size: A5
Date: 2023-10-31
Rev: 2
KiCad E.D.A. eeschema 7.0.7
Id: 1/1

(b) Revision 2 of the circuit

Figure 7: Comparison of circuit revisions

With such a setup, it is impossible to use SPI as the 595 has no chip select line by which it could be addressed. According to Figure 8, the *RCLK* pin of the 595 being pulled HIGH causes the data stored in the shift-register to be pushed into the storage-register, effectively "writing" that data to the output pins.

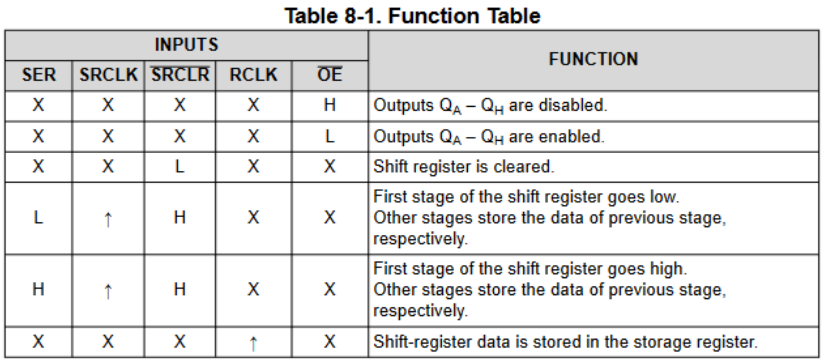


Table 8-1. Function Table

INPUTS					FUNCTION
SER	SRCLK	$\overline{\text{SRCLR}}$	RCLK	$\overline{\text{OE}}$	
X	X	X	X	H	Outputs Q_A – Q_H are disabled.
X	X	X	X	L	Outputs Q_A – Q_H are enabled.
X	X	L	X	X	Shift register is cleared.
L	↑	H	X	X	First stage of the shift register goes low. Other stages store the data of previous stage, respectively.
H	↑	H	X	X	First stage of the shift register goes high. Other stages store the data of previous stage, respectively.
X	X	X	↑	X	Shift-register data is stored in the storage register.

Figure 8: Rev. 1.0 of the circuit

This change was crucial for making SPI implementation possible, as in the second revision, for which please refer to Figure 7(b), both the 3008 and 595 each have their own chip select line, for the 3008 this being the $\overline{CS} \leftrightarrow D10$ connection and for the 595 it is the $RCLK \leftrightarrow D9$ connection.

4.2 SPI programmatic implementation

Note: In the actual code implementation, the methods used within *void loop()* are declared after the scope of *void loop()* is executed, for demonstration purposes, in the explanation further down, they are covered first to ease the reading experience and accessibility. The following subsection will focus on the software side of the implementation. As mentioned, this will be carried out without using libraries except for *<SPI.h>* and *<math.h>*.

Firstly, importing the necessary libraries, defining the pins for both ICs and defining the *START_BYTE*, *DIFFMODE* constants and *LED_PATTERN* variable used for controlling pattern displayed by the LEDs, and lastly, the SPI communication protocol settings:

```
#include <SPI.h>
#include <math.h>

const int MCP3008_CS_PIN = 10;

const int SN74HC595_SER_PIN = 16;
const int SN74HC595_RCLK_CS_PIN = 9;
const int SN74HC595_SRCLK_PIN = 15;
const int SN74HC95_OE_PIN = 3;

const byte START_BYTE = 1;
const byte DIFFMODE = 128;

byte LED_PATTERN = 1;

SPISettings mySettings(2000000, MSBFIRST, SPI_MODE0);
```

Referring back to Figure 3, which describes the serial communication, note the constant *DIFFMODE* is set to decimal 128, i.e. 1000 0000 binary. This is because I need to use the 3008 in a single-ended input configuration. This, in combination with the constant *START_BYTE*, which is declared as decimal 1, i.e., 0000 0001 binary, will later construct the first instruction that will be sent from the master MC to the 3008; in this relation, the slave.

Following is the setup, a very standard setting of pin modes. Notice that even though I use the 3008 as an input device, the chip select pin is declared as an *OUTPUT* pin as it will be used to address the 3008, and further communication will be carried out using SPI rather than reading data directly from the pin.

```
void setup()
{
  SPI.begin();

  pinMode(SN74HC595_SER_PIN, OUTPUT);
  pinMode(SN74HC595_RCLK_CS_PIN, OUTPUT);
  pinMode(SN74HC595_SRCLK_PIN, OUTPUT);
  pinMode(SN74HC95_OE_PIN, OUTPUT);

  pinMode(MCP3008_CS_PIN, OUTPUT);
  digitalWrite(MCP3008_CS_PIN, LOW);
  digitalWrite(MCP3008_CS_PIN, HIGH);
}
```

Next up, the functions for communication via SPI are declared. The *readPot(byte channel)* function expects a byte parameter that specifies the channel on which the data will be read.

```
int readPot(byte channel)
{
  SPI.beginTransaction(mySettings);

  digitalWrite(MCP3008_CS_PIN, LOW);

  SPI.transfer(START_BYTE);

  byte MSB = SPI.transfer(DIFFMODE + channel << 4) & 0x03;
  byte LSB = SPI.transfer(0);

  digitalWrite(MCP3008_CS_PIN, HIGH);

  SPI.endTransaction();

  return MSB << 8 | LSB;
}
```

Following is the function for updating the shift register values. Note how both in the *updateShiftRegister(byte pattern)* and in *readPot(byte channel)*, the chip select pin of both the 3008 and 595 is first pulled low after beginning the SPI communication.

Afterwards, the actual data is transferred.

```
void updateShiftRegister(byte pattern)
{
  SPI.beginTransaction(mySettings);

  digitalWrite(SN74HC595_RCLK_CS_PIN, LOW);

  SPI.transfer(pattern);

  digitalWrite(SN74HC595_RCLK_CS_PIN, HIGH);

  SPI.endTransaction();
}
```

Using PWM - Pulse Width Modulation, it is possible to set the brightness of the LEDs by writing an analog value between 0 and 255 to the *OE* pin of the 595.

```
void setBrightness(byte brightness)
{
  analogWrite(SN74HC95_OE_PIN, 255-brightness);
}
```

Eventually, all methods come together inside the *void loop()* to control the pattern displayed through the LEDs connected to the shift register.

```
void loop()
{
  byte bitPotChannel = 0;
  byte brightPotChannel = 2;

  int bitPotValue = map(readPot(bitPotChannel), 0, 1000, 0, 255);
  int brightPotValue = map(readPot(brightPotChannel), 0, 1023, 0, 255);

  LED_PATTERN =
  setBrightness(brightPotValue);
  for (int i = 0; i < 255; i++)
  {
    updateShiftRegister(i);
    delay(50);
  }
}
```

5 Conclusion

The document shares my adventure into embedded programming and electrical engineering, mainly focusing on learning about MCP3008 and SN74HC595 by diving into data sheets and hands-on projects.

Key Points:

- *Reading data sheets*: The author breaks down the intimidating data sheets of MCP3008 and SN74HC595. They highlight the crucial sections for beginners, like pin descriptions and application info.
- *Understanding components*: The document explains the practical side of MCP3008 (analog-to-digital converter) and SN74HC595 (shift register). It touches on concepts like single-ended vs. differential inputs and how a shift register works.
- *Real-world challenges*: The author shares the ups and downs of building actual circuits. They faced issues, like the lack of a chip select line, and show how they tackled these challenges.
- *SPI implementation*: The author aims to use the Serial Peripheral Interface (SPI) for communication. The document covers setting up hardware, defining pin modes, and implementing SPI in code.
- *Learning by doing*: The document reflects on the learning journey. It emphasizes the value of hands-on experience and problem-solving in grasping embedded programming and electronics.