Atmega 8 Connection to 74HC595

DEEPAK GUPTA

Sagar Mundada

When selecting the microcontroller of choice for your robot, one parameter will be the number of I/O channels available. However, it is quite likely that the ideal processor in terms of special functions may not have enough I/O for the purposes required. The solution to this is to use a shift register chain to expand the I/O potential.

74HC595 is a Shift Register IC take a signal from one wire and output that information to many different pins. In this case, there is one data wire that takes in the data and 8 pins that are controlled depending on what data has been received. To make things better, there is an outpin for each shift register that can be connected to the input pin of another shift register. This is called cascading and thus we can use it to expand limited Microcontroller output pin. We can use minimum of 3 output pin from Microcontroller to control the Shift Register IC (DS, SHCP and STCP) and we can have total of 8 output pin from from the Shift Register IC. In the other words, it is a serial (3 pin) to parallel (8 pin) converter that allows us to expand output pins from 3 pin to unlimited output pin. Connecting this Shift Register IC in series we can have more that 8 pin output.

Circuit Diagram

The circuit diagram shows the connection between atmega 8 controller and 74HC595 shift register.The control lines connected between controller and shift registers are

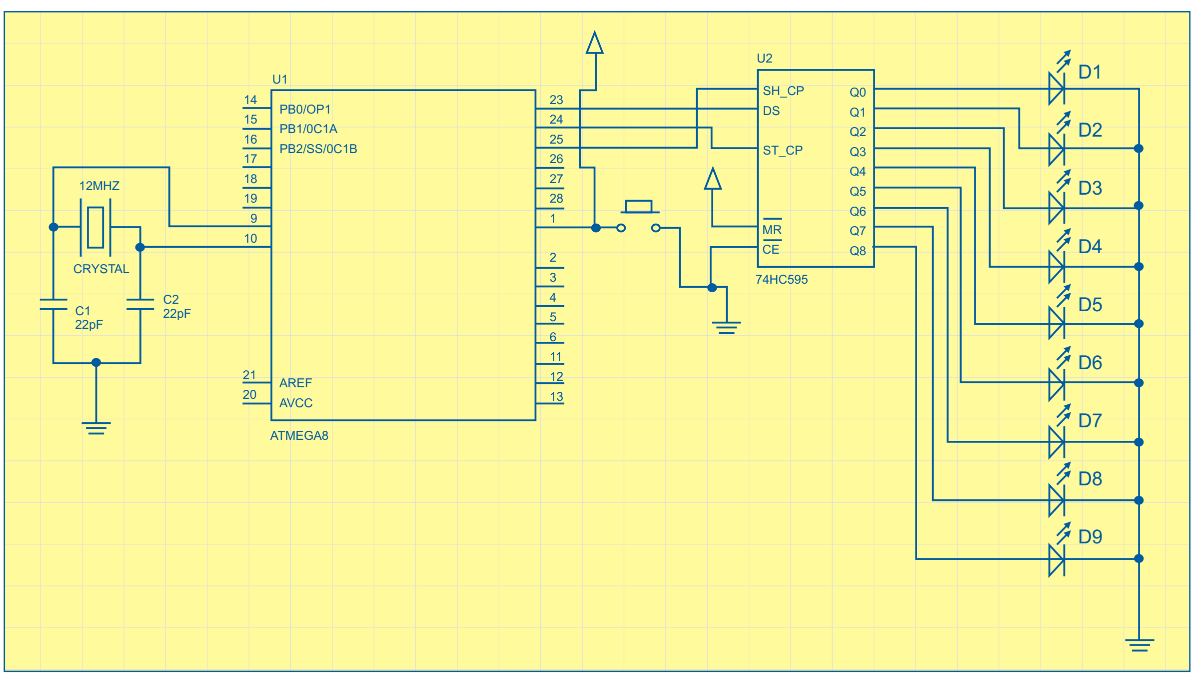
- PC0 to DS
- PC1 to ST_CP
- Pc2 to SH_CP

LEDs are connected at the output of the shift register.The circuit is built and simulated on Proteus. Data is sent from PC0 to the Ds pin of the shift register.The program can be easily altered to change to LED pattern.

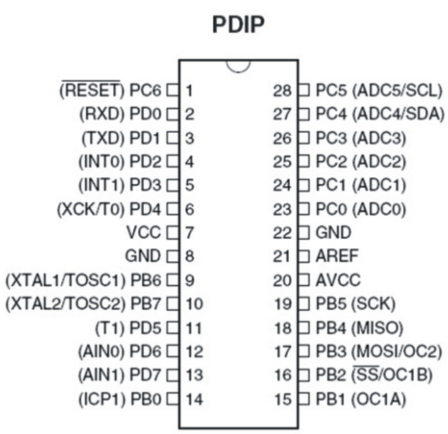
Component Used

ATMEGA 8A

The ATmega8A is a low-power CMOS 8-bit



microcontroller based on the AVR RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega8A achieves throughputs approaching 1 MIPS per MHz, allowing the system designer to optimize power consumption versus processing speed.The ATmega8 provides the following features: 8 Kbytes of In-System Programmable Flash with Read-While-Write capabilities, 512 bytes of EEPROM, 1 Kbyte of SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte oriented Two wire Serial Interface, a 6-channel ADC (eight channels in TQFP and QFN/MLF packages) with 10-bit accuracy, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and five software selectable power saving modes.



The Atmega8 has 23 I/O ports which are organized into 3 groups:

- Port B (PB0 to PB7)
- Port C (PC0 to PC6)
- Port D (PD0 to PD7)

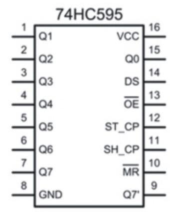
74HC595

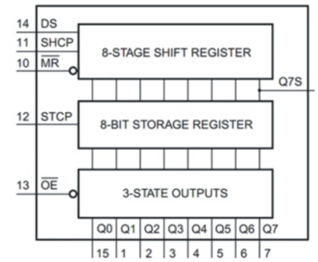
The IC has an 8 bit storage register and an 8 bit shift register. Data is written to the shift register serially and then latched onto the storage register. The storage register then controls 8 output lines.

Pin 14 (DS) is the Data pin. On some datasheets it is referred to as "SER".

When pin 11 (SH_CP or SRCLK on some datasheets) goes from Low to High the value of DS is stored into the shift register and the existing values of the register are shifted to make room for the new bit.

Pin 12 (ST_CP or RCLK on some datasheets) is held





Functional Diagram

low whilst data is being written to the shift register. When it goes High the values of the shift register are latched to the storage register which are then outputted to pins Q0-Q7.

You can cascade multiple 595's together to create more output pins.The operation of the 595 is very simple. You shift in each bit MS bit first. This is done by setting the Serial in pin high or low then taking the clock pin from a low to a high state. Once 8 bits are shifted you latch the data to the output lines by taking the latch pin from low to high. If a pins was already high or low and its new state is the same. There will be not change presented to that pin.

The main advantage of this method is that it allows an unlimited number of I/O pins to be used, even if the processor has a very small number. The main disadvantage is that all these inputs and outputs are going to be accessed a lot slower than normal I/O ports.

Program Code

```
#include <avr/io.h>
#include <util/delay.h>
#define DS_PORT    PORTC
#define DS_PIN    0
#define ST_CP_PORT PORTC
#define ST_CP_PIN  1
#define SH_CP_PORT PORTC
#define SH_CP_PIN  2
#define  DS_low     DS_PORT&=~_BV(DS_PIN)
//BV=Bit Value..It change the state of bit passed
#define DS_high() DS_PORT|=_BV(DS_PIN)
#define ST_CP_low() ST_CP_PORT&=~_BV(ST_CP_PIN)
#define ST_CP_high() ST_CP_PORT|=_BV(ST_CP_PIN)
#define SH_CP_low() SH_CP_PORT&=~_BV(SH_CP_PIN)
#define SH_CP_high() SH_CP_PORT|=_BV(SH_CP_PIN)
//Define functions
```

```
//======================
void ioinit(void);
void output_led_state(unsigned char __led_state);
//======================
 int main (void)
{
  ioinit(); //Setup IO pins and defaults
   while(1)
  {
              for (int i=7;i>=0;i--)
    {
      output_led_state(_BV(i));
      _delay_ms(100);
    }
          for (int i=1;i<7;i++)
    {
      output_led_state(_BV(i));
      _delay_ms(100);
    }
  }
}
 void ioinit (void)
{
     DDRC   = 0b00000111; // set PC0,PC1,PC2
output pins
   PORTC = 0b00000000;
}
```

```
void output_led_state(unsigned char __led_state)
{
  SH_CP_low();
  ST_CP_low();
  for (int i=0;i<8;i++)
  {
    if (bit_is_set(__led_state, i))
      DS_high();
    else          DS_low();
      SH_CP_high();
     SH_CP_low();
  }
// Timer/Counter 2 initialization
// Clock source: TOSC1 pin
// Clock value: PCK2/128
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x08;
TCCR2=0x05;
TCNT2=0x00;
OCR2=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x40;
lcd_init(16);
#asm("sei") // Global enable interrupts
while (1);
}
```