Arduino

ez Serial/Parallel IC Library (SIPO8) ~ Tutorial 3

A Tutorial to Consolidate Understanding
& Use of the ez Serial/Parallel IC Library
(SIPO8)

Tutorial 3
~ Using Timers ~

Author: R D Bentley, Stafford, UK.
Date: April 2021
Version: 1.00

Warranties & Exceptions

Change Record

Version	Date	Change
1.00	April 2021	Initial version published

Audience

Whilst it is not necessary to and be familiar with 8bit serial/parallel ICs, such as the 74HC595 IC, some understanding with wiring these ICs and driving them with suitable Arduino code, for example the standard Arduino `shiftOut` function, would provide an excellent primer on which to build and develop sophisticated and innovative solutions through the use of the `<ez_SIPO8_lib>` library.

Contents

Introduction to the Tutorial

In this third tutorial we will build on our Tutorial 1 and 2 experiences and look at how we can use the SIPO8 library timers features to control the status of SIPO IC output pins. We shall make use of the same physical SIPO IC circuit we set up in Tutorial 1.

The tutorial will configure eight SIPO8 timers, one for each of the SIPO output pins and use these to control the rate at which we wish to flash the connected LEDs on/off. The approach will be flexible allowing us to easily vary flash rates and output pin ordering.

(If you have not already done so, download the SIPO8 User Guide from github.)

Objectives

We shall concern ourselves with creating a virtual SIPO environment in which we can apply further basic principles. In particular we shall learn:

1. how we can wire up and connect a single 74HC595 IC (SIPO) to a microcontroller (see Tutorial 1, if you have not already done so)
2. how we can use inherent SIPO8 library timers to control and drive physical SIPO outputs
3. the further use of SIPO8 library functions
4. witness the outputs of the sketch – variable flashing LEDs and serial monitor SIPO data

Steps

If you ran through Tutorials 1 and 2 and still have your components set up as for that tutorial, then skip to The Code/Sketch, otherwise continue as below.

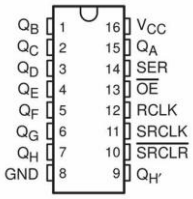
Kit List

Gather together the following components:

Components	Number	Comment
Arduino UNO	1	The design uses an Arduino UNO, but any suitable microcontroller (Arduino or clone, e.g. Elegoo) will do, providing it is able to support the pin out requirements for drving the SIPO interface and power requirements
Breadboard	1	Small or large, whatever you have to hand
74HC595 IC	1	8bit SIPO IC, or other clone providing it is genuinely 'plug-compatible'
LEDs	8	Whatever you have around
Restistors, 220 ohm	8	One per LED. Use 220 ohm resisters and ignore the suggested 180 ohm values in the wiring diagrams
Connecting wires	Lots	Short/long or breadboard wire connectors, whatever suits

74HC595 Orientation and Pin Outs

Which end is which? Well, notice that the 74HC595 has a notch at one end, here at the top of the diagram. Pin numbering starts at 1 at the top left and continues down the left hand side and the around the bottom of the IC rising to the top right hand side to pin 16:



Pin Outs 1 - 74HC595

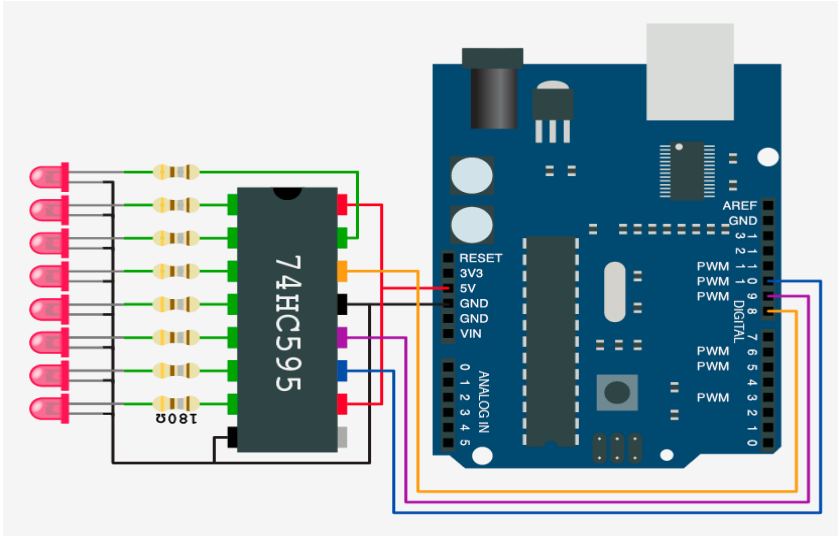
Microcontroller / SIPO Interface Pin Configuration

Every bank of SIPO ICs you connect to the microcontroller requires a 3-wire digital interface. The table below suggests pin mapping between the microcontroller and the SIPO IC for this tutorial, but you may choose what microcontroller digital pins you wish. If you do choose different pins then be sure to alter the sketch `create_bank` call.

UNO Pin	SIPO Pin(s)	Comment
8	14	SIPO Data Pin
9	12	SIPO Latch Pin
10	11	SIPO Clock Pin
+5v	10, 16	Power to the SIPO
GND	8, 13	Return ground (0v)

Connecting It All Together

Using the following diagram, wire up all of the components, taking care to get the output/input connections correct:



Wiring Scheme 1 - Single SIPO IC, 8 outputs

Notice that the only 74HC595 IC pin not to have anything connected is pin 9, Q_H'. This is used as the serial output pin to connect to the serial input pin, 14 SER, of the next SIPO 74HC595 in a cascade.

The Code/Sketch

Now that's done, let's look at a sketch using timers to control variable flashing LEDs.

Using the Arduino IDE, start with a new sketch and enter the following (download from github):

```
//   Tutorial 3 - use of ez_SPI8 library,
//   1x physical SIPO, and use of SIPO8 timers to control SIPO outputs with time
//
//   Ron D Bentley, Stafford, UK
//   April 2021
//
//   This example and code is in the public domain and
//   may be used without restriction and without warranty.
//

#include <ez_SIPO8_lib.h>

#define Max_SIPOs  1  // one virtual SIPO for this tutorial
#define Max_timers 8  // 8 timers required to map a complete 8bit SIPO

// initiate the class for Max SIPOs/timers required
SIPO8 my_SIPOs(Max_SIPOs, Max_timers);

int bank0_id;  // used to keep the SIPO bank id
//
// setup pin/LED flash data
uint8_t timer;
uint32_t timer_intervals[Max_timers] = {
  300, 400, 500, 600, 700, 800, 900, 1000 // millisecond elapse timer values
};
uint8_t timer_pins[Max_timers] = {
  0, 1, 2, 3, 4, 5, 6, 7 // SIPO output pin absolute addresses
};

void setup() {
  Serial.begin(9600);
  // create bank, params are:
  // data pin, clock pin, latch pin, number of SIPOs this bank
  bank0_id = my_SIPOs.create_bank(8, 10, 9, 1);
  if (bank0_id == create_bank_failure) {
    Serial.println(F("\nfailed to create bank"));
    Serial.flush();
    exit(0);
  }
  // print the bank data for confirmation/inspection
  my_SIPOs.print_SIPO_data();
}
void loop() {
  // start by setting all SIPO outputs to low (off)
  my_SIPOs.set_all_array_pins(LOW);// set all declared virtual output pins LOW/off
  my_SIPOs.xfer_array(LSBFIRST);// move virtual pin statuses to real SIPO o/p pins
  //
  // start all timers
  for (timer = 0; timer < Max_timers; timer++) {
    my_SIPOs.SIPO8_start_timer(timer);
  }
  timer = 0; // start checking at first timer
  do {
    // check each timer for elapse and, if elapsed, invert the timer's output pin
    // and reset the timer
```

```
    if (my_SIPOs.SIPO8_timer_elapsed(timer, timer_intervals[timer]) == elapsed) {
      // invert the pin associated with this timer
      my_SIPOs.invert_array_pin(timer_pins[timer]);
      my_SIPOs.xfer_array(MSBFIRST);       // update physical SIPO outputs
      my_SIPOs.SIPO8_start_timer(timer);   // reset/restart the current timer
    }
    timer++;  // move on to next timer
    if (timer >= Max_timers) timer = 0;    // wrap around to beginning
  } while (true);
}
```

(all SIPO8 functions/methods are shown against a darker grey background)

Notice:

1. that we automatically configure the eight timers we need for our purposes as a part of the SIPO8 class initiation (second parameter, here 8). We can create up to 255 timers if desired, but that's a lot!
2. we declare and preset two arrays to define the flash rate intervals and associated absolute output pin numbers we wish to control –

   ```
   uint32_t  timer_intervals[Max_timers]  and
   uint8_t   timer_pins[Max_timers]
   ```

   These arrays allow us to easily vary flash rates and change the order of output pins if we desire

3. there are three SIPO8 library functions that allow us to utilise and control SIPO8 timers, these being `SIPO8_start_timer`, `SIPO8_timer_elapsed` and `SIPO8_stop_timer`. Each timer function takes and initial parameter which is the timer number we wish to reference. However, in the case of `SIPO8_timer_elapsed` then this function has a second parameter which is the elapsed time interval we wish to check against for elapsed time. Note that this second parameter is an unsigned 32 bit variable, so watch how you define these values in your sketch.

   The sketch does not use the `SIPO8_stop_timer` function, but it is documented in the User Guide.

4. the code controlling the flash cycle is succinct and simply structured

Check the serial monitor, it should look like this:

```
SIPO global values:
pins_per_SIPO   = 8
max_SIPOs       = 1
bank_SIPO_count = 1
num_active_pins = 8
num_pin_status_bytes = 1
next_free bank =  all SIPOs used
Number timers  =  8

Bank data:
bank = 0
  num SIPOs = 1
  latch_pin = 9  clock_pin =      10  data_pin  =     8
  low_pin   = 0  high_pin  =      7
```

Before finishing this tutorial, have a go at varying the flash rates and/or output pin ordering.

That is the end of Tutorial 3, which I hope you found instructional.