Arduino

ez Serial/Parallel IC Library (SIPO8) ~ Tutorial 6

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

Tutorial 6
~ Q & As ~

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

Warranties & Exceptions

Change Record

Version	Date	Change
1.00	May 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 tutorial we will deal with a number of Questions & Answers (Q&As), looking at what is going on behind the scenes and how we can use some of the SIPO8 library's inner features to good effect.

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

Objectives

We shall assume that if you have covered previous tutorials you will already have setup a test harness comprising a single SIPO IC and connected LEDs, but if not and you wish to practise some of the features covered by this tutorial, then setup the components as outlined below Steps/Kit List/etc.

In this tutorial we shall explore some obvious questions and provide, hopefully, helpful answers.

Steps

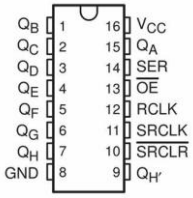
Kit List

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

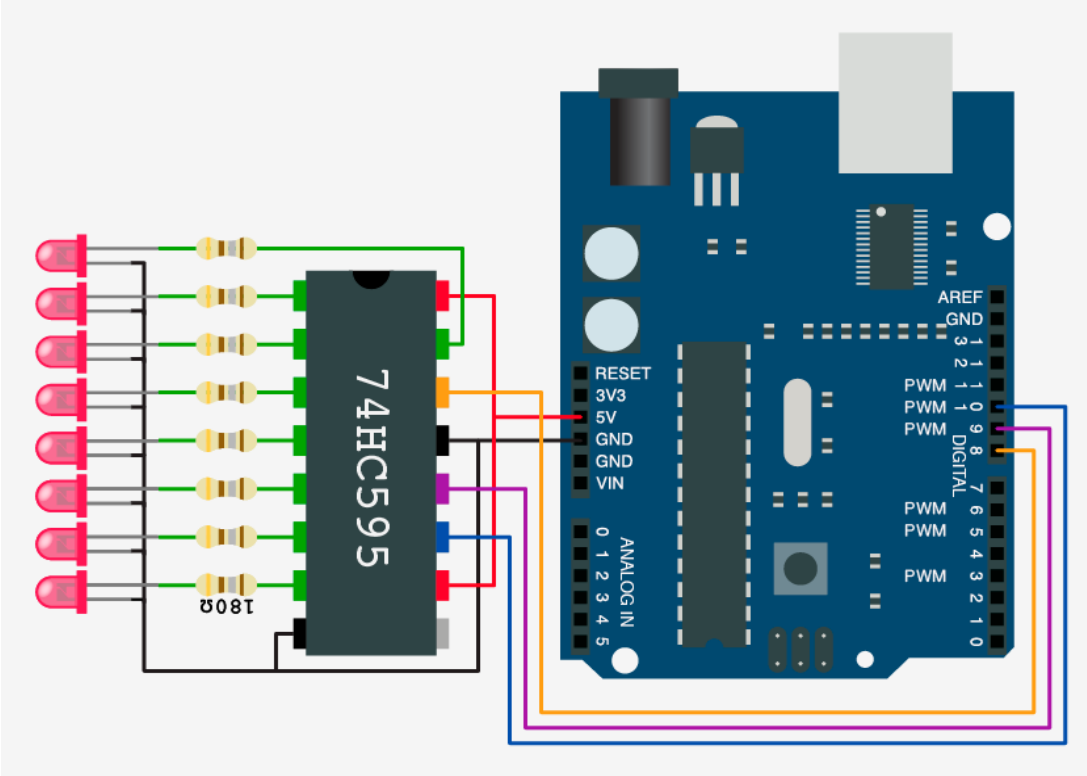
Every bank of SIPO ICs you connect to the microcontroller requires a 3-wire digital interface. The table below suggests pin mapping between the microcontroller and the SIPO IC for this tutorial, but you may choose what microcontroller digital pins you

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.

Q & As

What is/How Can I/How Do I...

Q1. What is the virtual array pool of output pins?

A1. The SIPO8 library will create and reserve space for every SIPO IC output pin you request when you initiate the library's class. This is determined by the first parameter of class initiation statement, e.g. `SPIO my_SIPOs(4, 0)`, here the request is to create a virtual SPIO output pin environment of 4 x single 8bit SPIO ICs, or 4 x 8 = 32 output pins.

The entire virtual set of SIPO output pins are referred to as the virtual array or array pool, with an address range from 0 to the (8 x number of SIPO ICs) – 1. In the above example the address range is 0 to 31.

It is important to understand that, once created, the virtual output pin array is not yet addressable. This is because that the library does not yet know how these virtual output pins are to be associated with/linked to the physical world, i.e. physical SIPO ICs. This cannot happen until we allocate the array pool to one or more banks. See question 2.

Q2. How do I link the virtual output pins in the array pool to physical SPIO ICs?

Having defined/created our virtual array pool we now need to chop it up/allocate it into banks of grouped output pins. A bank will:

- comprise a unique set of virtual output pins allocated from the virtual array pool
- map a consecutive and contiguous absolute address range of virtual output pins
- define the 3-wire digital microcontroller pins used to drive the bank
- make the bank's associated output pins active and therefore addressable, absolutely and relatively (see question 3)

For example, using our example above (`SPIO my_SIPOs(4, 0)`), we wish split up our 32 virtual output pins into 2 banks, 1 bank of 8 output pins and the other bank the remaining 24 output pins, and allocate these to microcontroller digital pins 3, 4, 5 and A0, A1, A2 respectively. The banks are created using the `create_bank` function so:

```
bank1_id = my_SIPOs.create_bank(3, 4, 5, 1);     // 8 outputs
bank2_id = my_SIPOs.create_bank(A0, A1, A2, 3); //24 outputs
```

The first `create_bank` call allocates array pool pins 0 – 7 to the first bank (`bank1_id`) and makes the associated output pins active and addressable. The second `create_bank` allocates array pool pins 8 – 31 to the second bank (`bank2_id`) and makes the associated output pins active and addressable.

Now the array pool is fully allocated to our banks and all output pins active so we can address them in two ways – by absolute addressing and by relative addressing. See question 3.

Q3.What is the difference between absolute addressing and relative addressing?

A3. The answer is very simple:

Absolute addresses relate only to the active output pins in the virtual array pool with an inclusive address range from 0 to (the number of output pins) - 1 in the array pool. There are few functions (but sufficient) that use absolute addressing, these are:

- `set_all_array_pins(status)` implicit absolute addressing
- `invert_all_array_pins()` implicit absolute addressing
- `set_array_pin(pin_address)` explicit absolute addressing
- `invert_array_pin(pin_address)` explicit absolute addressing
- `read_array_pin(pin_address)` explicit absolute addressing
- `xfer_array(LSBFIRST_or_MSBFIRST)` implicit absolute addressing

where `pin_address` is the absolute address of the pin in the array pool, i.e. an address range from 0 to the (8 x number of defined SIPO ICs) – 1. In the above example, the absolute address range is from 0 – 31 inclusive. (See the User Guide for a full description of each of these functions.)

Relative addresses relate only to the active output pins in a bank with an inclusive address range from 0 to (the number of output pins in a bank) - 1. There many bank related functions that use relative addressing, these are:

- `set_bank_pin(bank_id, pin_address, status)` explicit relative addressing
- `invert_bank_pin(bank_id, pin_address)` explicit relative addressing
- `read_bank_pin(bank_id, pin_address)` explicit relative addressing
- `set_bank(bank_id, status)` implicit relative addressing
- `set_banks(from_bank, to_bank, status)` implicit relative addressing
- `set_banks(status)` implicit relative addressing
- `invert_bank(bank_id)` implicit relative addressing
- `invert_banks(from_bank, to_bank)` implicit relative addressing
- `invert_banks()` implicit relative addressing
- `set_bank_SIPO(bank_id, SIPO_num, SIPO_value)` implicit relative addressing
- `invert_bank_SIPO(bank_id, SIPO_num)` implicit relative addressing
- `read_bank_SIPO(bank_id, SIPO_num)` implicit relative addressing
- `xfer_bank(bank_id, LSBFIRST_or_MSBFIRST)` implicit relative addressing
- `xfer_banks(bank_from, bank_to, LSBFIRST_or_MSBFIRST)` implicit relative addressing
- `xfer_banks(LSBFIRST_or_MSBFIRST)` implicit relative addressing

where
`pin_address` is the relative address of the pin in the bank, i.e. an address range from 0 to (number of output pins in the bank) – 1.
`SIPO_num` is the relative address of an 8bit SIPO define by the bank, i.e. ranges from 0 to (number of SIPOs defined for the bank) – 1.

In the above example, the respective addressing ranges are:

entity	address range	SIPO address range
array pool	0 - 31	n/a
`bank1_id`	0 - 7	0 - 0
`bank2_id`	0 - 23	0 - 2

(See the User Guide for a full description of each of these functions.)

Q4. How do I know how many virtual SIPO pins there are in the array pool?

A4. The library provides a number of user accessible variables that are helpful in decision support and control. In this particular case, the library variable to use is `max_pins`*, and you would use it by prefixing it with the name you gave for the SIPO8 class when you initiated it. For example, if you named your class '`my_SIPOs`' then the use would be `my_SIPOs.max_pins`.

Q5. How do I know how many virtual active SIPO pins there are in the array pool?

A5. Again, we would use a library variable, in this instance the library variable `num_active_pins`*, and you would use it by prefixing it with the name you gave for the SIPO8 class when you initiated it. For example, `my_SIPOs.num_active_pins`.

* *Note that there is a distinction to be made between how many SIPO pins an array pool is sized for (`max_pins`) and the number of active pins (`num_active_pins`) in an array. When the SIPO8 class is initiated, the array pin pool is created to be as large as required by the provided class parameter. However, at this point none of the array pins are accessible. Before they can be used/accessed they need to be assigned into banks using the `create_bank` function, which groups consecutive pins into groups of a multiple of eight pins. (See question 1).*

Q6. I know the absolute address of a SIPO pin, how do I work out which bank it is in?

A6. The library has a function specifically for this need – `get_bank_from_pin`. It takes a single parameter (absolute pin address) and returns the id of the bank it is mapped by. If the specified pin does not belong to any bank then it will return a value of -1 (macro names of `bank_not_found`). Look at the following example which shows how we can work from an absolute active pin address to its associated bank and then draw together a number of useful attributes:

```
int bank_id = my_SIPOs.get_bank_from_pin(97);
if (bank_id != bank_not_found){
  // ok, work on this bank
  uint16_t low_pin_add  = SIPO_banks[bank_id].bank_low_pin;
  uint16_t high_pin_add = SIPO_banks[bank_id].bank_high_pin;
  uint8_t  num_SIPOs    = SIPO_banks[bank_id].bank_num_SIPOs;
  // we now have number of SIPOs in this bank and the
  // absolute pin address range of all pins in the bank
  ...
}
```

Q7. How can I find out the absolute output pin addresses a bank is assigned/owns?

A7. Central to the library's design is a control struct(ure) called `SIPO_banks` (you saw an example of this in question 6). It is fully documented in the User Guide, but for convenience it looks like:

```
struct SIPO_control {
  uint8_t  bank_data_pin;
  uint8_t  bank_clock_pin;
  uint8_t  bank_latch_pin;
  uint8_t  bank_num_SIPOs;
  uint16_t bank_low_pin;
  uint16_t bank_high_pin;
} *SIPO_banks;
```

For a given bank id, any of the above bank attributes may be accessed/referenced simply by giving the bank_id, for example:

```
unit16_t abs_start = my_SIPOs.SIPO_banks[bank_id].bank_low_pin;
unit16_t abs_last  = my_SIPOs.SIPO_banks[bank_id].bank_high_pin;
```

and, if we wished to find out how many output pins a bank owns/maps, we can do this in several way:

```
1. uint8_t num_pins = my_SIPOs.SIPO_banks[bank_id].bank_high_pin -
                      my_SIPOs.SIPO_banks[bank_id].bank_low_pin + 1;

2. uint8_t num_pins = my_SIPOs.SIPO_banks[bank_id].bank_num_SIPOs
                      * pins_per_SIPO;

3. uint8_t num_pins = my_SIPOs.num_pins_in_bank(bank_id);
```

Note that, although bank pin addressing is relative addressing, the low/high pin addresses held in a bank are absolute pin address values.

Q8. What other things should I know about that may be useful?

A8. A bit of an open question, but let's look at what other library variables that are accessible and which you may find helpful. The table below highlights several of these, but more precise information can be found in the User Guide.

Other user accessible SIPO8 library variables you may find helpful are:

SIPO8 library variables	Description

`uint8_t * pin_status_bytes`	Every virtual, and therefore physical, SIPO output pin is mapped by a single bit in the `pin_status_bytes` array. This array is sized sufficient to map all required SIPO output pins, starting at pin 0.
	Each 8bit byte of the array corresponds to 8 output pins. Therefore byte 0 holds output pins 0 – 7, byte 1 holds output pins 8 – 15, and so on. It may be accessed directly:
	
	`status_byte = pin_address / pins_per_sipo;`
	`status_pin  = pin_address % pins_per_sipo;`
	`pin_status  =`
	`  bitRead(pin_status_bytes[status_byte],`
	`  status_pin);`
	
	Much better to use the library's explicit functions?

SIPO8 library variables	Description

my_SIPOs.num_pin_status_bytes	This provides the total number of pin status bytes mapping the entire array pool. For example:
	
	`// clear down all status byte pins`
	`uint8_t num_bytes, byt;`
	`num_bytes = my_SIPOs.num_pin_status_bytes;`
	`for(byt = 0; byt < num_bytes; byt++){`
	`  pin_status_bytes[byt] = 0;`
	`}`

my_SIPOs.num_banks	This variable defines the number of active banks created using the create_bank function. For example, as banks start at bank 0 and run consecutively:
	
	`uint8_t num_banks, bank;`
	`num_banks = my_SIPOs.num_banks;`
	`for(bank = 0; bank < num_banks; bank++){`
	`  my_SIPOs.xfer_bank(bank, LSBFIRST_or_MSBFIRST);`
	`}`
	
	The above code is equivalent any one of the following:
	`my_SIPOs.xfer_banks(LSBFIRST_or_MSBFIRST);`
	`my_SIPOs.xfer_banks(0, my_SIPOs.num_banks-1, LSBFIRST_or_MSBFIRST);`
	`my_SIPOs.xfer_array(LSBFIRST_or_MSBFIRST);`

my_SIPOs.max_SIPOs	This is the number of 8bit SIPOs the SIPO8 class was initiated with. For example: `SPIO my_SIPOs(4, 0);` Here my_SIPOs.max_SIPOs would be equal to 4.
my_SIPOs.bank_SIPO_count	The total number of 8bit SIPOs allocated to banks. When all class defined SIPOs are allocated to banks then: `my_SIPOs.bank_SIPO_count==my_SIPOs.max_SIPOs`
my_SIPOs.max_timers	The number of SIPO8 timers that class has been initiated for. Note that this value can be 0 to 255.

`struct timer_control {`
`  bool  timer_status;`
`  uint32_t start_time;`
`} *timers;`
	This structure is used to control each of the defined timers with each timer entry being referenced by the timer_number, from 0 to my_SIPOs.max_timers -1.
	
	For example:
	
	`if(my_SIPOs.timers[3].timer_status==active){`
	`  ..`
	`}`

That is the end of this tutorial which I trust you have found to be helpful?