How to connect multiple AD985X devices

Introduction

The AD985X is a versatile function generator that can be controlled by a SPI like interface, however it does not have a **SELECT** line.

This makes it hard as a device to share the (hardware) SPI bus as it would clock in any data send to other devices too. The solution for this is to add some extra electronics and a **SELECT** line so the device can be controlled more reliably in a multi-SPI device environment. If that works we can also control multi **AD985X** devices from one processor.

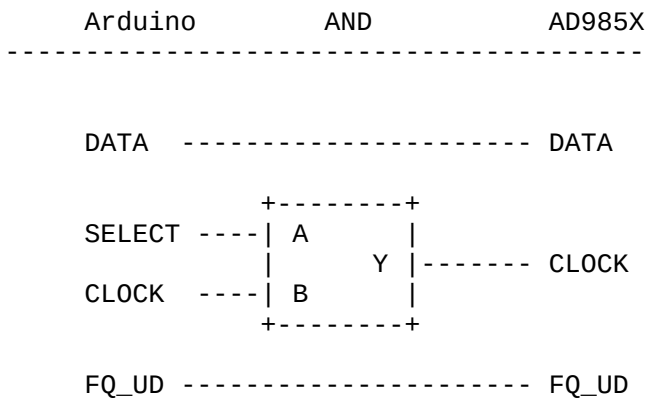
Architecture 0

The simplest way to control multi **AD985X** devices is to give them all a disjunct set of pins. As every device needs typical 4 pins this adds up quickly. In the library this means using the software SPI interface and this works very well.

Drawback is the large amount of pins used, so some other alternatives will be described.

Architecture 1

Insert an **AND** port between **CLOCK** line, controlled by **SELECT** line



By controlling the **CLOCK** line by the **SELECT** signal only the device selected (**SELECT** == **HIGH**) will see the **CLOCK** and will therefor read the bits on the **DATA** line into its frequency register(s). When the **FQ_UD** signal is given only one of the **AD985X** devices will have a new frequency in its register and update it accordingly.

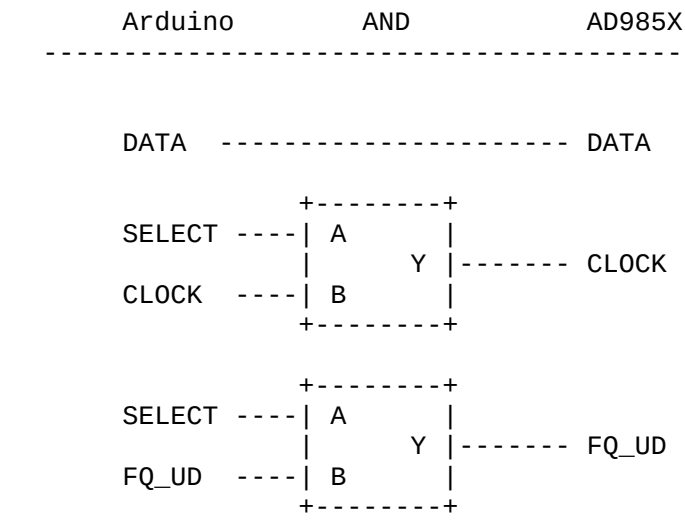
Although this architecture seems to work pretty well, there is a possible problem related to the moment of the update. This moment is not defined and can therefor cause devices without a new frequency register value to reset its output signal. This could possibly result in an sudden shift in phase or in case of a square wave a broken duty cycle.

If the sudden phase shift is no problem for your application. then architecture 1 could work for you. With a single **74HC08** IC you can control up to four **AD985X** devices. All four devices can share the **DATA**, **CLOCK** and **FQ_UD** line, and you need only an unique **SELECT** line per device

If your application does not "like" phase shifts or broken duty cycles, we need to “shield off” the **FQ_UD** line too. Time to investigate architecture 2.

Architecture 2

Insert an **AND** port between **CLOCK** and **FQ_UD** line, controlled by **SELECT** line



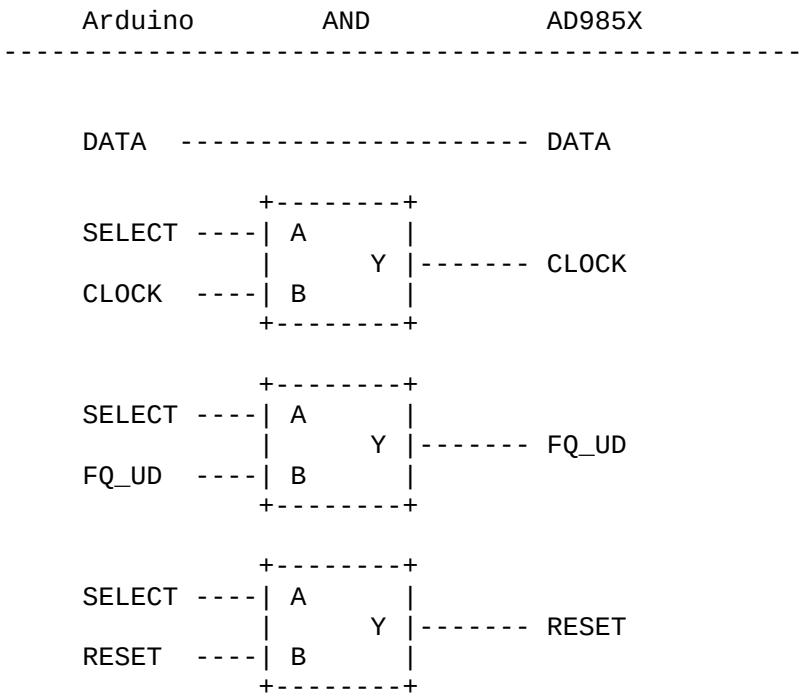
This architecture looks quite similar to architecture 1 and is not that exciting. The only difference is that it uses a second **AND** port for the **FQ_UD** signal. Now the frequency update signal **FQ_UD** is also explicitly controlled by the **SELECT** line. It allows you to selectively update the devices you want, in the order you want.

Only the **AD985X** device selected will now update or refresh its frequency, phase etc. So all the devices not selected will not do anything and therefor will not have a sudden phase-shift or broken duty cycle. Therefor this architecture is more robust and will give you more control than the previous architecture 1.

With a single **74HC08** IC you can control two **AD985X** devices.

Architecture 3

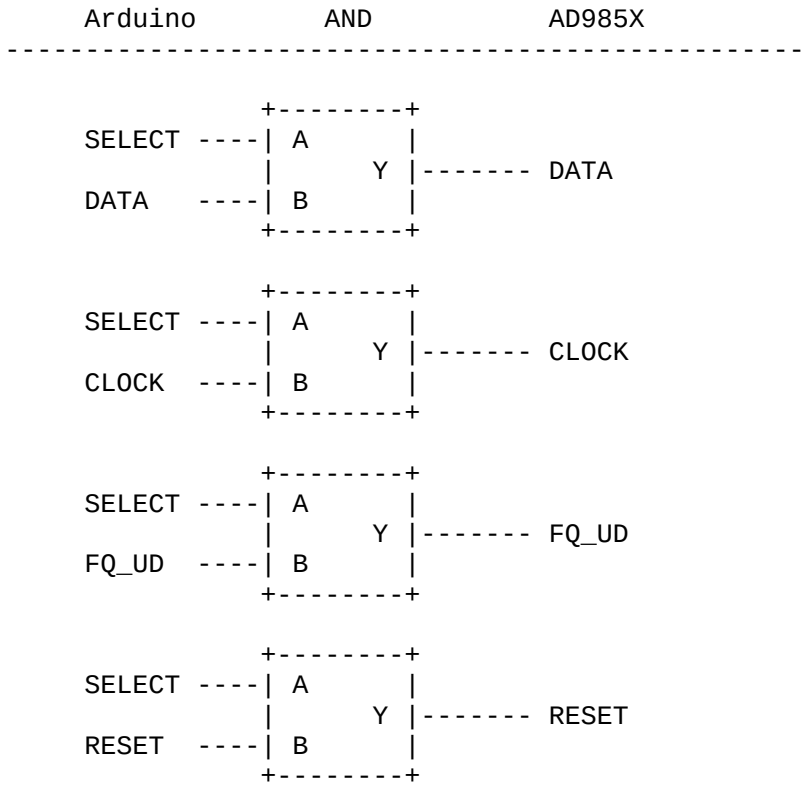
Insert an **AND** port between **CLOCK, FQ_UD** and **RESET** line, controlled by **SELECT** line



Architecture 3 looks is an extension of architecture 2 and it adds an **AND** port to control the (shared) **RESET** pin of the **AD985X** devices. The working is similar to the **FQ_UD** discussed above. It allows you to selectively reset the devices you want, in the order you want.

Architecture 4

Insert an **AND** port between all lines controlled by **SELECT** line



Using three **AND** ports for a device makes us wonder why not add the fourth **AND** port (as they typically come in packages of four) for the **DATA** line. This would indeed be the most robust way to prevent data meant for other devices to 'leak' into an **AD985X**.

Although using this extra **AND** port would not add a functional advantage, it will be good for the electronic design. In the design it will give a one to one relation between the **AND** IC and the **AD985X**, which will give the same schema for any amount of **AD985X**'s used.

Questions

Q1: Why introduce a **SELECT** line, as one could share the **DATA** and **CLOCK** lines and give every device its own **FQ_UD** line?

A good question and yes it could make the architecture simpler if the **AD985X** was the only device on the bus. One of the primary functions of the **SELECT** line is to prevent data from entering the **AD985X** if it was not meant for it. That is the only way to prevent unwanted side effects.

Q2: What is the function of the **autoUpdate()** function for multiple devices?

By setting **autoUpdate(false)** for all devices one can change multiple devices at exact the same moment with a single pulse. For this one something like need the following snippet of code:

```
// assume autoUpdate(false)
dev1.setFrequency(x);
dev2.setFrequency(y);
dev3.setFrequency(z);
// optionally prepare setPhase etc per device.

// prepare simultaneous sync by selecting them all
digitalWrite(SELECT1, HIGH);
digitalWrite(SELECT2, HIGH);
digitalWrite(SELECT3, HIGH);

// update the frequency of all three devices at once
// as all devices share the FQ_UD line
digitalWrite(FQ_UD, HIGH);
digitalWrite(FQ_UD, LOW);

// restore the unselect state for all devices
digitalWrite(SELECT1, LOW);
digitalWrite(SELECT2, LOW);
digitalWrite(SELECT3, LOW);
```

With a similar code construct as above, one could update the frequency of a subset of the devices e.g only two out of four devices at exact the same time, and the other two at a later moment.

Because the class offers the **SELECT** the user can dynamically define when devices are updated simultaneously and when they are updated sequentially.

Note that the **RESET** of all devices or a subset, can be done in the same way, simultaneously or sequentially.

Q3: How can I set multple devices to the same frequency as fast as possible?

One can save some time by disabling the **autoUpdate()** of the devices. Although this sounds like a contradiction the following code will show how it can be done.

```
// assume autoUpdate(false)

// prepare simultaneous writing by selecting them all
digitalWrite(SELECT1, HIGH);
digitalWrite(SELECT2, HIGH);
digitalWrite(SELECT3, HIGH);

// writing to device 1 will automatically also write to device 2 and 3
// as we have selected them all
dev1.setFrequency(x);

// as setFrequency pulls the SELECT line low we need to pull it high again.
digitalWrite(SELECT1, HIGH);

// update the frequency of all three devices at once
// as all devices share the FQ_UD line
digitalWrite(FQ_UD, HIGH);
digitalWrite(FQ_UD, LOW);

// restore the unselect state for all devices
digitalWrite(SELECT1, LOW);
digitalWrite(SELECT2, LOW);
digitalWrite(SELECT3, LOW);
```

A test based on sketch – AD9850_multi_sync – runnning on an Arduino UNO gives the following results and shows it is always substantially faster for multiple devices.
Disclaimer: tested setFrequency() call duration, not with actual devices

Lib version	Mode	# Devices	Sequential	Simultaneous	Factor
0.3.0	HW-SPI	1	72	84	0,86
0.3.0	HW-SPI	2	220	92	2,39
0.3.0	HW-SPI	3	320	100	3,20
0.3.0	HW-SPI	4	424	108	3,93
0.3.0	HW-SPI	5	528	120	4,40
0.3.0	SW-SPI	1	548	572	0,96
0.3.0	SW-SPI	2	1172	584	2,01
0.3.0	SW-SPI	3	1756	588	2,99
0.3.0	SW-SPI	4	2336	600	3,89
0.3.0	SW-SPI	5	2924	600	4,87

Further reading

Datasheets

- https://www.analog.com/media/en/technical-documentation/data-sheets/AD9850.pdf
- https://www.analog.com/media/en/technical-documentation/data-sheets/AD9851.pdf

7400 series ports

- https://en.wikipedia.org/wiki/List_of_7400-series_integrated_circuits

Arduino library

- https://github.com/RobTillaart/AD985X