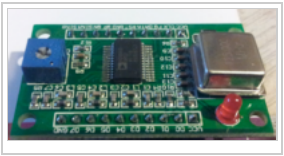
AD9850 Arduino

Contents

Using the Analog Devices AD9850 DDS with an Arduino board



There is a fair bit of information regarding the AD9851 DDS (as used on the NJQRP DDS-60 daughter card) chip with Microchip PICs and Arduino development boards, but not much for the slightly cheaper and lower spec (but still good) AD9850. The AD9850 has no internal reference clock multiplier, so it requires a faster reference clock then the AD9851. Not having an internal multiplier means that code written for the AD9851 will not work directly on the AD9850 - the code will attempt to set registers that do not exist in the AD9850.

A no-frills AD9850 DDS module can be obtained from eBay for around £4 at the time of writing (August 2012) and will produce a decent signal from a few Hz to over 30 MHz. A read of the datasheet provides all the information needed to drive the DDS chip and get some RF out.

Initialising and Resetting the AD9850

These code fragments are written in the C++ like language the Arduino development board uses.

```
#define DDS_CLOCK 125000000
#define CLOCK 8 //pin connections for DDS
#define LOAD 9
#define DATA 10
#define RESET 11




void AD9850_init()
{
digitalWrite(RESET, LOW);
digitalWrite(CLOCK, LOW);
digitalWrite(LOAD, LOW);
digitalWrite(DATA, LOW);
}



void AD9850_reset()
{
//reset sequence is:
// CLOCK & LOAD = LOW
// Pulse RESET high for a few uS (use 5 uS here)
// Pulse CLOCK high for a few uS (use 5 uS here)
// Set DATA to ZERO and pulse LOAD for a few uS (use 5 uS here)

// data sheet diagrams show only RESET and CLOCK being used to reset the device, but I see no output unless I also
// toggle the LOAD line here.

digitalWrite(CLOCK, LOW);
digitalWrite(LOAD, LOW);

digitalWrite(RESET, LOW);
delay(5);
digitalWrite(RESET, HIGH); //pulse RESET
delay(5);
digitalWrite(RESET, LOW);
delay(5);

digitalWrite(CLOCK, LOW);
delay(5);
digitalWrite(CLOCK, HIGH); //pulse CLOCK
delay(5);
digitalWrite(CLOCK, LOW);
delay(5);
digitalWrite(DATA, LOW); //make sure DATA pin is LOW

digitalWrite(LOAD, LOW);
delay(5);
digitalWrite(LOAD, HIGH); //pulse LOAD
delay(5);
digitalWrite(LOAD, LOW);
// Chip is RESET now
}
```

All so far so simple, pretty much the same as you'd see with AD9851 drivers. I'm still not sure why the datasheet suggests you only need to pulse RESET and CLOCK to reset the chip, when I find in practice I need to pulse the LOAD line too.

Setting the output frequency

The output frequency is set by calculating a 40-bit turning word and loading it to the DDS, either via a 3-wire serial bus or an 8-bit parallel bus. The 40-bit word is comprised of 32-bits of phase and frequency information and a further 8-bit, 3-bits that set specific operating (and factory test) modes of the DDS - it is these 3-bits that cause problems when trying to use AD9851 code with the AD9850- and 5-bits of phase information. For simplicity, I set all of these bits to 0. The datasheet gives the following equation to calculate the turning word

$$f_{OUT} = \frac{\Delta Phase \times REFCLOCK}{2^{32}}$$

This translates into C++ as

```
unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_CLOCK;
```

using floating point maths. Or as

```
Unsigned long tuning_word = (frequency * 4294967296LL) / DDS_CLOCK;
```

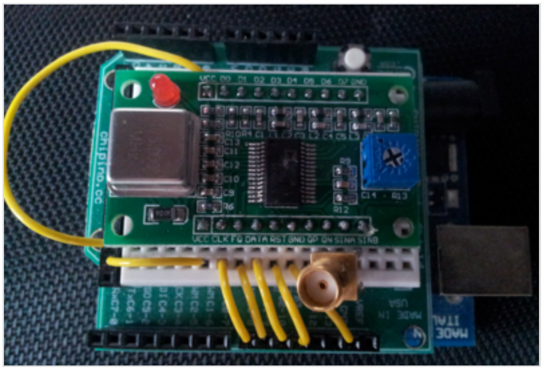
using integer maths. In theory, integer maths should be slightly faster and more accurate as the required frequency increases, in practice I find either method fast enough and accurate enough up to 30MHz, frequency errors are due to the poor stability of the reference oscillator on the DDS module. Calculating the turning word and writing it to the DDS module can them be wrapped up in a single function taking the required frequency as its sole parameter

```
void SetFrequency(unsigned long frequency)
{
unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_CLOCK;
digitalWrite (LOAD, LOW);

shiftOut(DATA, CLOCK, LSBFIRST, tuning_word);
shiftOut(DATA, CLOCK, LSBFIRST, tuning_word >> 8);
shiftOut(DATA, CLOCK, LSBFIRST, tuning_word >> 16);
shiftOut(DATA, CLOCK, LSBFIRST, tuning_word >> 24);
shiftOut(DATA, CLOCK, LSBFIRST, 0x0);

digitalWrite (LOAD, HIGH);
}
```

Wiring up & example code download

The code assumes that Arduino pins 8,9,10 & 11 are connected to the DDS CLOCK, DATA, LOAD and RESET lines respectively. Signal output can be taken from pin 21 of the DDS chip.

An example project that just initialises the DDS and sets the output frequency to 10 MHz can be downloaded here. Do what you want with the code except claim it as your own and try to prevent anyone else using it.



Categories: C++ | Experiments | Projects | Radio | Arduino