Manual Control for the ADF5355 Synthesizer G4JNT March 2020

Introduction

This controller allows an ADF5355 synthesizer module to be controlled seamlessly over its complete tuning range of 54 to 13600MHz using a rotary control plus pushbuttons as well as a keypad for direct frequency entry. While various Ebay suppliers do offer various modules that do a similar task, using touch screens etc. as integrated units, there is no flexibility with those and no opportunity for individualisation. You've got what you have, take it or leave it.

The calculations and programming structure for the ADF5355 are described in detail in http://www.g4jnt.com/ADF5355_Synthesizer_Control.pdf , but to summarise :

$$F_{OUT} = F_{COMP} * (N + (F1 + F2/D2) / 2^{24}) / OP_{DIVIDER}$$

F_{COMP} is the comparator frequency, equal to the reference input multiplied by 2 if the reference doubler if used, and divided by the R divider. Unless there's a good reason to do otherwise, set the R divider to 1 for the highest possible F_{COMP}. N is the integer divider and F1, F2 the fractional registers that do the fine frequency setting. D2 is the second fractional denominator with the first one, D1, fixed at 2^{24}. Very fine frequency setting resolution is possible, to within $F_{COMP} / (D2 * 2^{24})$

Hardware

The main controller consists of a 16F628A PIC with rotary encoder, two pushbuttons and a Liquid Crystal Display. A two line by 16 character LCD will suffice, but choosing a four line one has certain advantages, especially when it comes to customising your own PIC code. The circuit diagram is shown in **Figure 1**.

A keypad, (for example Farnell ECO Keypad, Order code 113-0805) is included for direct frequency setting. Rather than making the main controller hardware more complex by using a PIC with more I/O pins to accommodate the keypad, this has its own controller, based around a 16F688 PIC device. It is interfaced to the main controller using a two wire interface that is not the same as, but not totally unlike, an I2C interface . The same keypad hardware, although with different PIC code, appears in these projects http://g4jnt.com/FT817_Keypad.pdf and http://g4jnt.com/FDM-DUO_Keypad.pdf The * key functions as a decimal point and the # key as 'enter' or [rtn]. Frequencies are entered as MHz followed by [rtn]. The circuit for this is shown in **Figure 2**.

Communication with the ADF5355 module is via a three wire SPI interface. As the synthesizer runs at 3.3V and the PIC at 5V, dictated by the LCD, each of these three interfaces needs potting down by 1.6k and 3.3K resistors. I built these onto a header that plugs into the 5x2 way connector on the synthesizer module.

Note that the four way connector shown as 'In Circuit Programming' on both diagrams is my own interface standard and is not compatible with the PICKIT programmer connections. A long story, a legacy thing, don't ask.

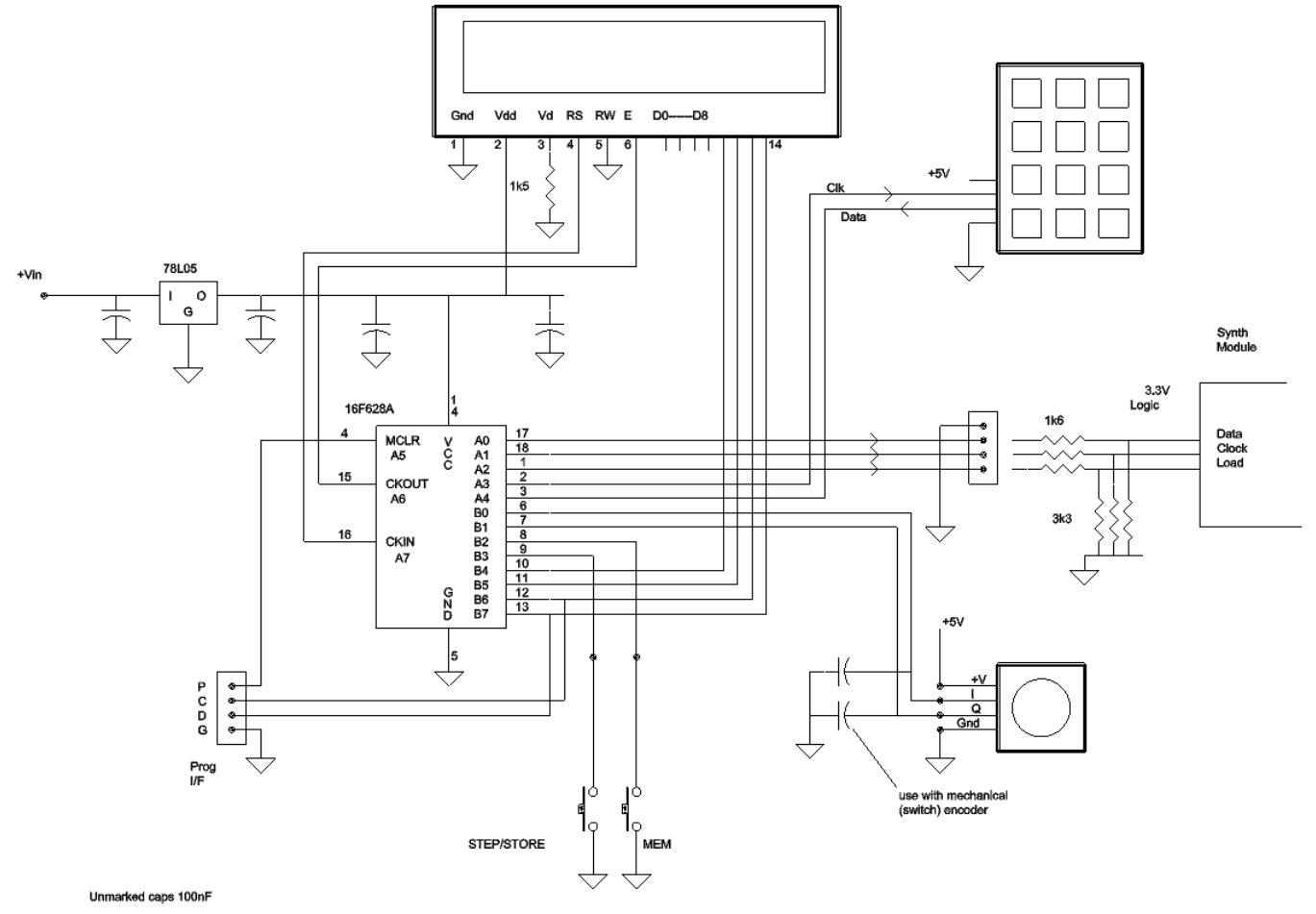
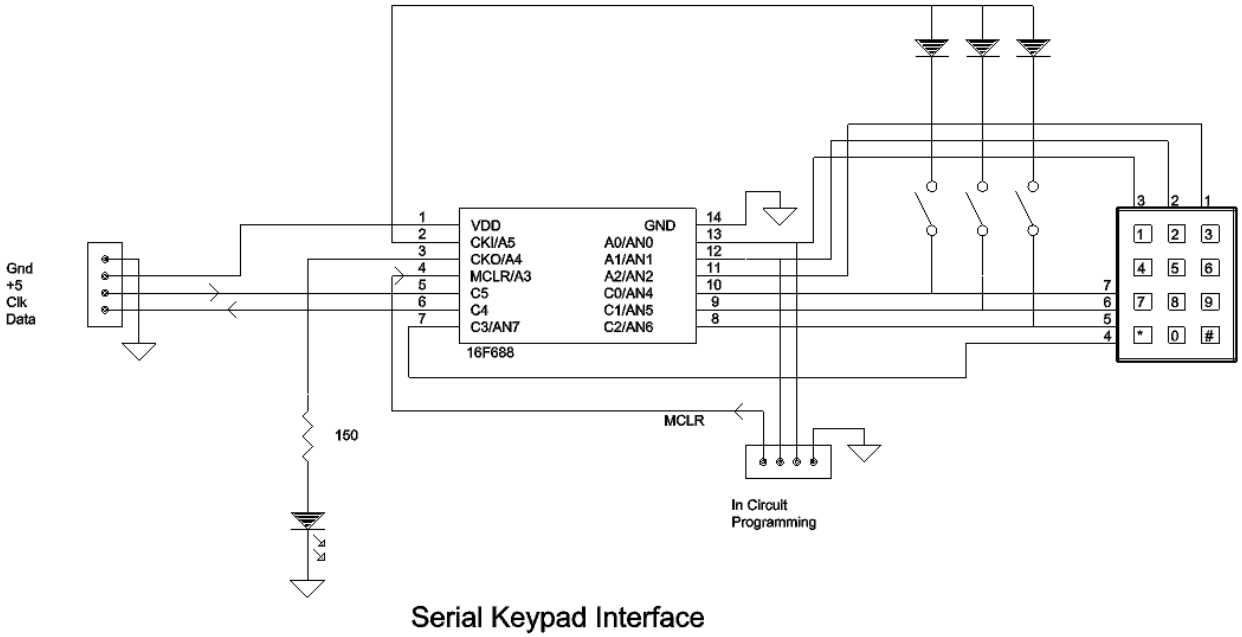


Figure 1 Main Controller



Serial Keypad Interface

Figure 2 Keypad Controller

Functionality

The rotary encoder increments or decrements decimal digits on the display, the digit being altered indicated by a caret, ' ^ ' under it. The digit selection is obtained by repeatedly pressing the [S]tep button to cycle through from 100MHz to 10Hz. Pressing and holding this button until the display shows 'Saved' stores the displayed frequency in the current memory position. That also saves the caret position ready for next time the unit is turned on.

Twenty memories are available, cycled in turn from 1 to 20 by pressing the [M]emory button. Pressing and holding this button saves the current memory to be called up next time the unit is turned on.

Frequency can also be entered directly from the keypad by entering, for example, '*2320*4506#*' for 2320.4506MHz. (A bit of white paint and black marker on the two keypad buttons helps !)

PIC Code and Firmware Customisation

As the PIC code is written in assembler for speed, effortlessness (on my part) and efficiency, floating point arithmetic is not a sensible solution. All you PIC C-code programmers may well gloat, but you'll struggle to get the tuning resolution and speed without using a fast processor and double precision variables. So all the arithmetic within the firmware is done using integers. What follows may look complicated but it isn't. The arithmetic for the entire calculation for each tuning step consists of an addition, BCD conversion, binary scaling and test (a shift register), a single 32 bit integer division then a 32 bit multiplication followed by some logic to split and move registers around.

The desired frequency, set the from rotary encoder or keypad, is stored as a value in 10Hz steps. A 32 bit integer allows frequencies up to 42.9GHz stored in registers Freq3 to Freq0. The rotary encoder increments or decrements this value by an amount equal to the digit indicated. After 4ms of encoder inactivity all calculations are performed and the synthesizer updated. This allows a moderate rate of turn to appear as continuous tuning, but if turned too fast, prevents race conditions where the updating can't keep-up.

For situations where the synthesizer unit is used as an LO, an optional IF offset is stored as a constant in the firmware. This is a signed 32 bit value corresponding to the IF in units of 10Hz. For

normal operation set it to zero. The IF offset is added to the value in Freq3/0, which is then converted to BCD and shown on the display.

The next stage is to calculate the Fractional-N values; the integer divider N, fractional dividers F1, F2 and the output divider. A copy of the required Freq value is made in the V3 – V0 registers. This is tested against the minimum internal VCO frequency (VCOMIN), 3.4GHz for the ADF5355. The value of V3/0 then being shifted (multiplied by 2) repeatedly until it falls into the range covered by the VCO, 3.4 to 6.8GHz. If the frequency starts out already above VCOMAX, V3/0 is set to half the required value and a flag set to enable the RF doubler on the ADF5355. The V3/0 registers now contain the value of F_{VCO} in units of 10Hz.

The integer divider, **N** is obtained directly by integer dividing V by the comparison frequency ie.

$N = INT (F_{VCO} / F_{COMP})$ which leaves a remainder **R** from this division, an integer specifying units of 10Hz which has to be manipulated to give the two fractional values. That requires a bit of skulduggery. We rewrite the original equation as:

$$F_{VCO} = F_{COMP} * N + (F1 * D2 + F2) * F_{COMP} / (2^{24} * D2)$$

This has been done intentionally, for two reasons. Firstly the residual **R**, after **N** has been calculated, is all that to the right of the addition sign. So $R = (F1 * D2 + F2) * F_{COMP} / (2^{24} * D2)$. Also note that the right hand part, after the second multiplication, is a constant since we have defined the second Fract-N denominator **D2** as a fixed value. Rewrite as $R = (F1 * D2 + F2) / K$ where $K = 2^{24} * D2 / F_{COMP}$

By rearranging we now get $(F1 * D2 + F2) = R * K$

R is our residual after the single 32 bit division that generates **N** , while **K** is a constant we specify and don't have to calculate each time. All that remains is to get, separately, **F1** and **F2** and now we get crafty. We can define **D2** as any value we like up to 16383, so let's make it the largest power of two we're allowed, in this case 8192 or 2^{13}.

By doing this we can concatenate the two values for $F2 * 2^{13} + F2$ as a single 37 bit value, the result of multiplying two numbers, one our residual and the other a constant. Being a long binary number we can easily separate out the two values for **F1** and **F2** merely by choosing where in the pattern of '1's and '0's to break into two parts - that's just logic manipulation. One final twist is to scale **K** to get the highest value we can reasonably work with, so the rounding errors inherent to integer arithmetic lead to a minimal final frequency setting error. A scaling factor of 2^{16} is applied so $K = 2^{53} / F_{COMP}$. This is recovered by then just throwing away the lowest two bytes of the final multiplication.

The remaining tasks are administrative. The values in N and F1/F2 have to be placed in the correct position in the words to be sent to the ADF5355 chip. The best way to see this is to examine the source code – look in the routine labelled *ProgSynth*. A few other registers need to be updated based on requested frequency output; those containing the output divider setting, output doubler and output switching. The rest of the registers are fixed in value and will have to have been determined off line, either using AD's design tools or the utility in http://g4jnt.com/ADF5355_Synthesizer_Control.pdf . They are stored as constants in the PIC firmware.

The PIC firmware needs to be customised for any particular hardware configuration with values both for F_{COMP} and $2^{53} / F_{COMP}$, the frequency being specified in units of 10Hz in each case. For 80MHz (reference in = 40MHz with the reference doubler activated) FREF = 8000000 and KCONST = 1125899907 (They have these specific names in the listing. FREF ought really to have a different name as it is NOT necessarily the same as the reference input). While the PIC assembler software

can do calculations on the fly to get constants, the results are only to 32 bit integer precision, so cannot be used here. Bothe FREF and KCONST have to be stored, with the latter derived off line using a calculator.

Odds and Ends

The method of calculation used means there is an inherent uncertainty in the frequency setting due to integer rounding; this is no more than 0.005Hz and has been kept down by the 16 bit prescaling applied to K.

If you want 1Hz steps instead of 10Hz, it will require more than 32 bit resolution, so everything will have to be modified, including all the maths routines – it's really not worth it – I did contemplate the idea, briefly!

When playing with the PIC code, if a 4 line display is used then register values can optionally be displayed. At the beginning of the listing there are two flags, DebugFractN and DebugReg. If the first one is set (= 1, then the code is reassembled) the values for N, OPDIV, F1 and F2 are displayed on lines 3 and 4 of the display. If The second flag is set, the hex values for the Reg6, Reg2, Reg1 and Reg0 are given. These are the ones that are updated as part of the frequency setting, the rest are fixed as prestored values. Don't set both debug flags at the same time – the result is a mess.

There is currently no checking for tuning outside the allowed frequency range – it wasn't worth the effort with a 54 – 13600MHz valid range.

Frequencies above 6.8GHz, where the O/P doubler is used, actually jump in 20Hz steps although the display doesn't reflect this. Only even multiples of 10Hz are accurate. I may get round to fixing it one day, it doesn't seem too important.

This is a copy of the opening section of the source code for my specific case (80MHz F_{COMP}), showing the constants used.

```
VCOMIN          =       d'340000000'            ; 10Hz units
VCOMAX          =       d'680000000'
FMIN            =       d'5400000'              ;Min allowed Freq in 10Hz units
FMAX            =       d'1360000000'           ;Max  "       "     "
DEFAULTFREQ     =       d'50000000'     ;Something to fill the EE with before saving real freqs
FREF            =       d'8000000'      ;Fcomp in units of 10Hz
KCONST          =       d'1125899907'   ;2^53 / FREF  (calculate off line,  needs > 32 bit arithmetic)
FIF             =       d'0000000'      ;IF Offset, units of 10Hz. Add to display frequency (can be -Ve)
NMEMS           =       d'20'           ;Number of memories

;Synth specific constants.  Frequencies and fixed register masks

RegC            =       0x0001041C              ;Get from Design software or data sheet
RegB            =       0x0061300B
RegA            =       0x00C0193A              ;Some Values here depend on Fref
Reg9            =       0x2221BCC9
Reg8            =       0x102D0428
Reg7            =       0x120000E7
Reg6            =       0x3503C076              ;Odiv goes in here bits 21-23
Reg5            =       0x00800025
Reg4            =       0x34009584              ;Ref doubler in here at bit 26 (0x34.. or 0x30..)
Reg3            =       0x00000003
                                                ;Registers  2 - 0 are constructed from first principles

DebugFractN     =       0       ;Display N, F1, F2, Odiv on LCD lines 3/4
DebugReg        =       0       ;Display Synth registers 0,1,2, 6 on LCD lines ¾
```

The PIC source code for both controllers as well as .HEX files (specifically for 40MHz reference input) can be found at http://g4jnt.com/adf5355_rot.zip