A BASIC ADF4110 LOCK-BOX

This is an experimental local-oscillator locking project that, while it has a well-tried topology, differs in the chips and the PIC programming language used. Code listings for such items are often supplied as a hexadecimal dump together with the assembly source code. These are often not very clear to an experimenter who might wish to use them as a basis for their own code in another project. It was thought then that a higher-level language would be more transparent and so it was programmed using the "MikroBasic PRO" compiler **[1]**.

This note's purpose then is to not only to encourage constructors to try a PIC Basic but also to point to the unlicensed version of this particular software as an example of what is available. Though having a compiled file length of only 2kb, it is full-featured and it is quite surprising what may be achieved within this single limitation. For those who may be interested in Pascal or C compilers, they are also available from the same site along with one for the AVR micro-controller. Obviously, just like any other commercial organisation, MikroElektronika are in the business of selling their products that include development boards as well as software.

The schematic shows the 10MHz-referenced lock-box with four user-selectable, rational-numbered, local oscillator frequencies. The ADF4110 is a low noise, 550MHz, 3-wire programmable, synthesizer with flexible dividers, a phase-frequency detector whose reference depends on the divisors used (see the simulator at **[2])**, a precision charge pump with no 'dead zone' and a digital 'in-lock' circuit. Individual frequency and housekeeping data is loaded from the 8-pin 12F629 PIC and selected by a pair of DIL switches. The loop filter employs a LM6211 op-amp; a chip designed for this particular purpose which is powered by 9V to give a greater pull-in range and output voltage swing from the VT terminal for an estimated Kv of 100Hz/V. If only one frequency is desired the DIL switches may be eliminated and just one set of data entered at 0x00 in the EEPROM. A 'reset' switch is provided in case of 'hang-ups' and for 'hot-switching'. The PIC has six port pins (GPIO) of which three are used to transfer data to the synthesizer by the 'Load Enable', 'Clock' and 'Data' lines. The 'Counter Reset Method' uses 4 x latch loads (excluding 'Initialisation') to achieve this as is explained on pages 14 and 20 of the data sheet **[2]**. So on power-up or reset -

- LE is taken low and a burst of 24 CLK pulses is generated.
- On the falling edge of each of these, one bit of data is transferred until...
- LE is taken high again to finish the loading of an EEPROM block.
- This is then repeated three times to finish the DATA load.

There are 12 words (double-bytes) used in each operation and, being timed by the PIC's 4MHz internal RC clock, no external crystal is required. EEPROM editing is via a built-in utility in the compiler and consists of the entry of data into a simple table. To align them for ease of viewing, they are padded out with a single 'FF' and a numerical reminder of the frequency; neither of which are sent to the ADF4110. As all code streams have several bytes in common with only the central ones defining the frequency; this allows any differences to stand out.

Here are two sample strings with those parts underlined -

1F 80 16 70 01 90 00 7D 09 1F 80 12 FF 10 02 00 (100.2MHz)

1F 80 16 70 01 90 00 9B 15 1F 80 12 FF 12 45 00 (124.5MHz)

This may all seem like gobbledygook but it is nothing more than a series of calculated bit patterns that trigger internal switches to set up the chip and, with the exception of the division ratios, all are much the same. A very few programmers will not load the EEPROM directly from the hex code. For these alone the BASIC listing must be modified to do this at start-up by including the strings and their "EEPROM_Write" statements. For brevity these are not shown here.

Most components are 1206 SMDs and their values, being just what I had at the time, are not critical except perhaps for the 2.6:1 ratio of R1/2. The MSA-0885 has so much gain that just a simple resistive load, limiting the current to 20mA, was used. The 10k (R12) pull-up resistor was included because the GP0/LE combination tends to float a little when set high. The cost of the principal chips was around £10 **[3]** and the unit was built RDDS-style in one of G3NYK's **[4]** 1000105 (74 x 55.5 x 30mm) boxes. The compiler's BASIC project files, with its source code and programming hex-dump, are available on the "Software" page of this web site, as are pointers to the full packages and viewers for the "sPlan 70" schematic and "Sprint-Layout 50" PCB software used in this project.

Integer-N systems usually have a higher phase-noise than fractional-N and so are probably better employed on the lower frequency microwave bands; even so it is the intention of a fellow amateur to try this out on 24GHz. Division numbers and loop filter values for the project were generated by the "ADsimPLL" utility also found at reference **[2]**. This project was not particularly intended for duplication (although it can be) but was simply meant as a pointer to other things. Bye-the-way, I do not claim much originality for the basic circuit concept, as there have been several notable precursors, especially one by DF9IC **[5]**. This is in German but do look for "PL-VCXO" in his archives.

Finally you may wish to look at another PLL chip - National's LMX2306 **[6]** which has a similar architecture and, it would be also remiss of me not to mention another piece of useful (though not free-of-charge) software, the Oshonsoft PIC Simulator IDE **[7]**. It is a one-man effort, emanating from Serbia (as does MikroBasic Pro), and has a dedicated Yahoo group for support. The hex code generated by both the mikroElektronika and OshonSoft products will also operate in the Microchip **[8]** MPLAB IDE environment should one need to. Suffice to say that the usual personal disclaimers apply with respect to all the companies mentioned above.

[1] http://www.mikroe.com

[2] http://www.analog.com

[3] http://uk.rs-online.com

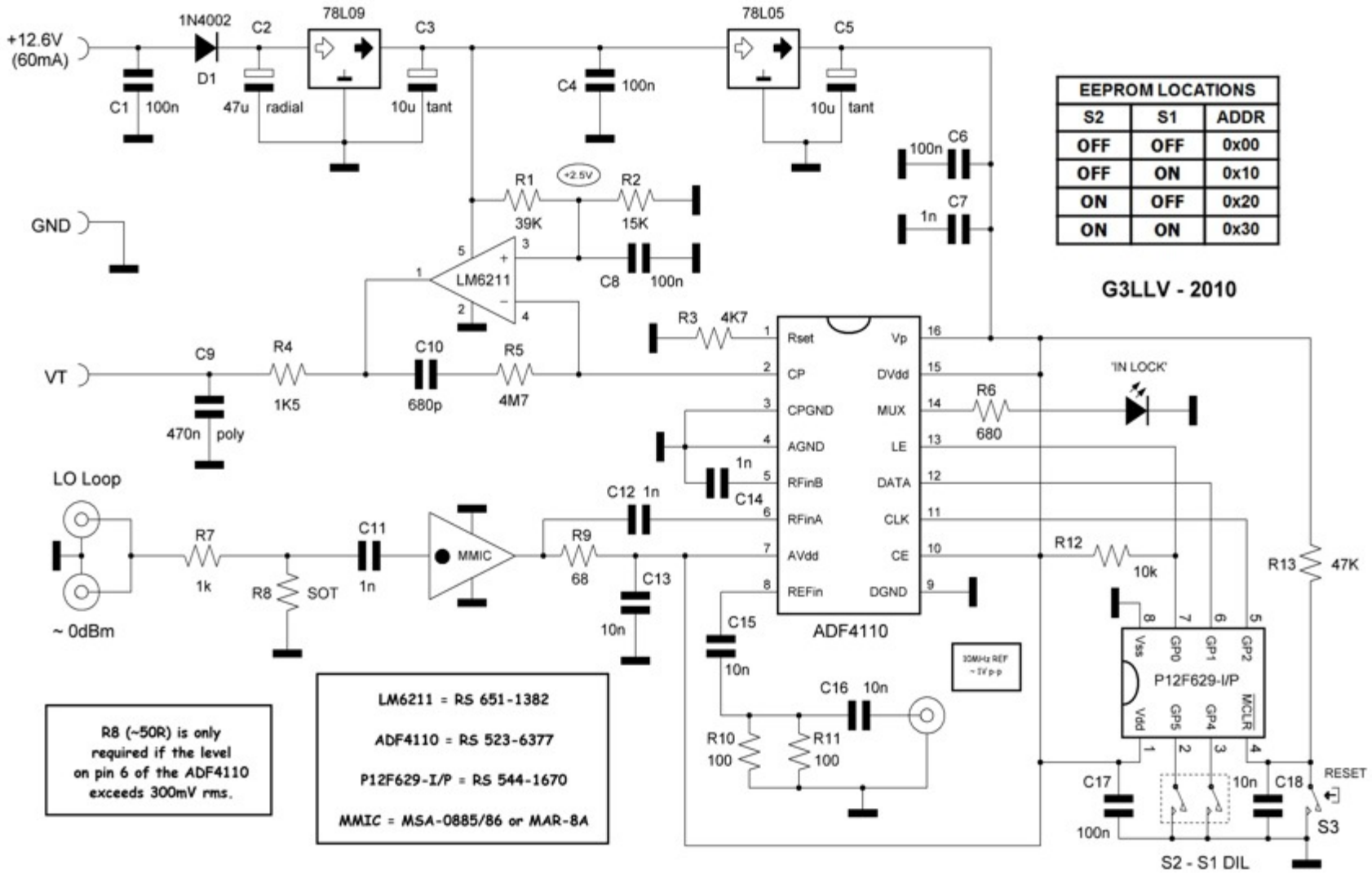
[4] alan.melia(at)btinternet(dot)com

[5] http://www.df9ic.de

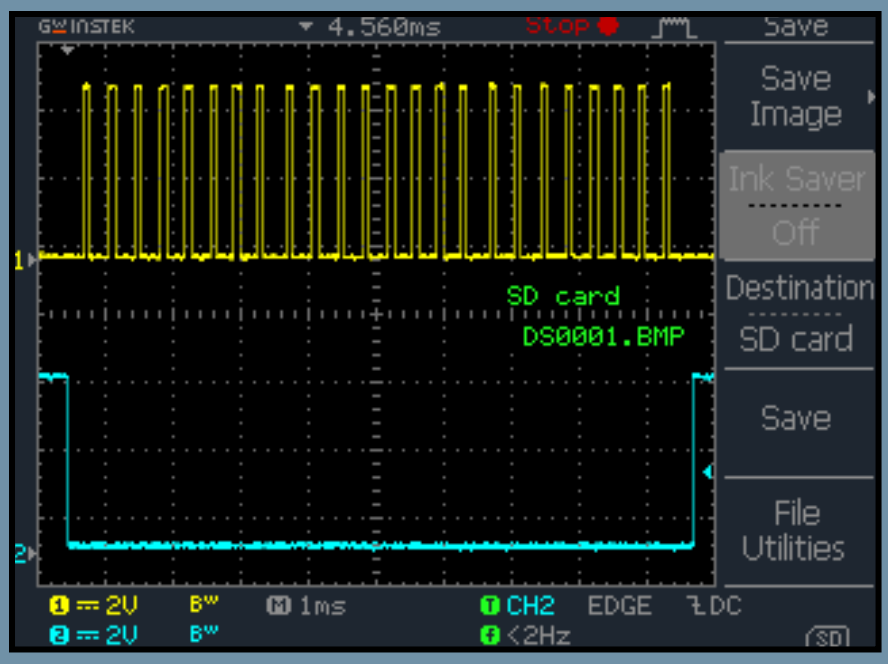
[6] http://www.national.com

[7] http://www.oshonsoft.com

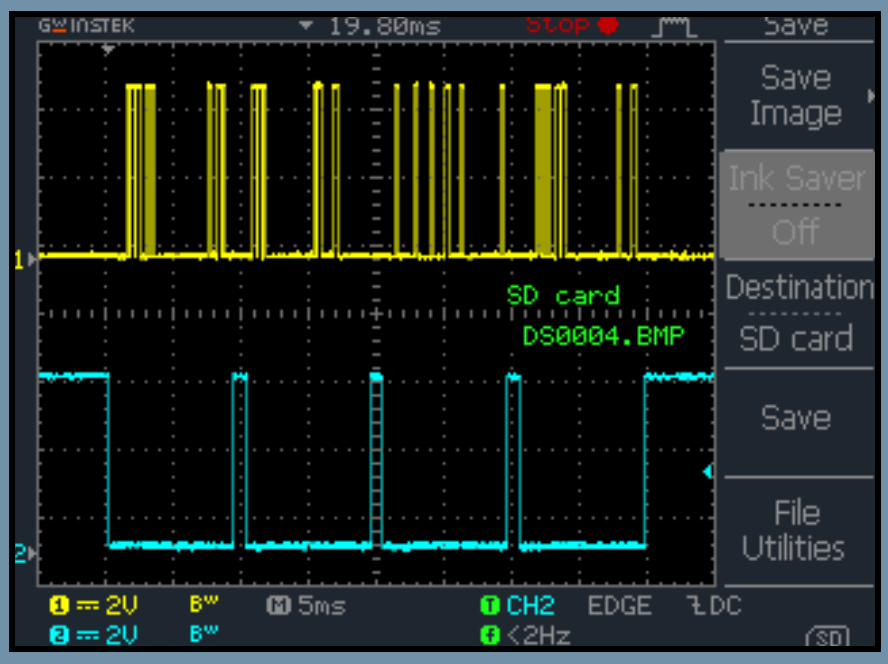
[8] http://www.microchip.com



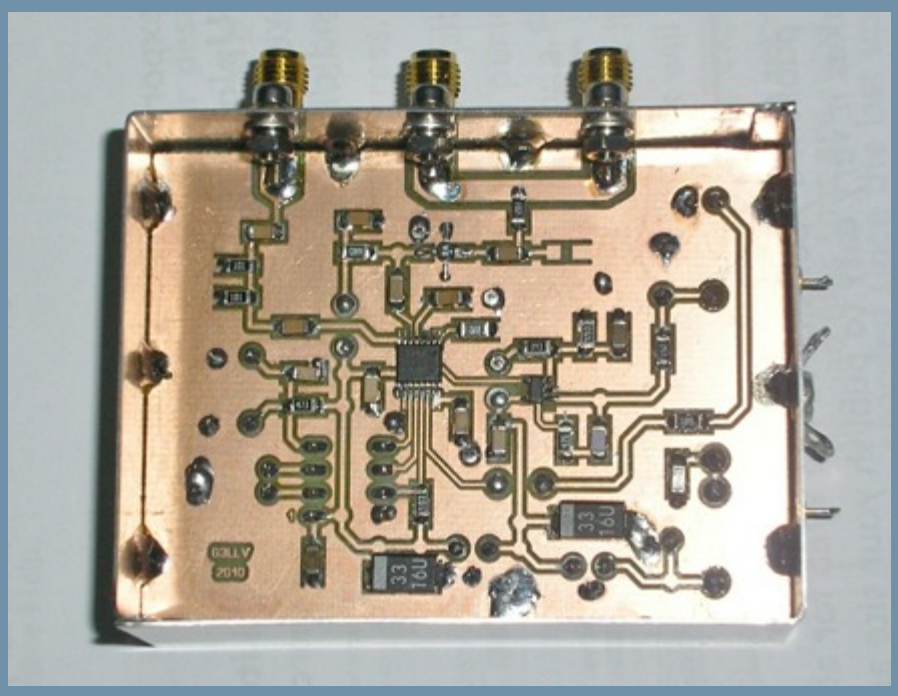
Home : Description : Layout : Software

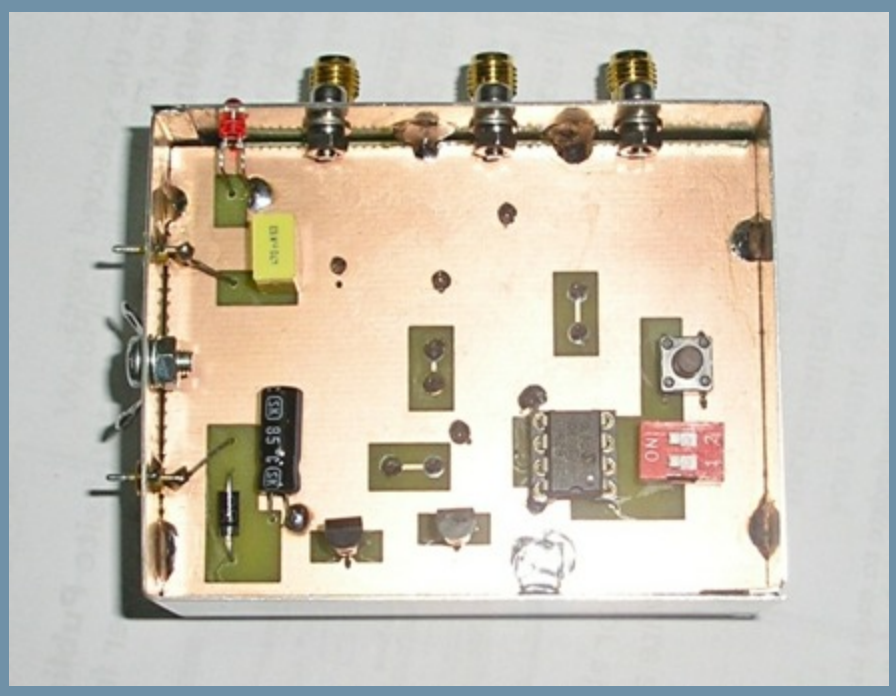


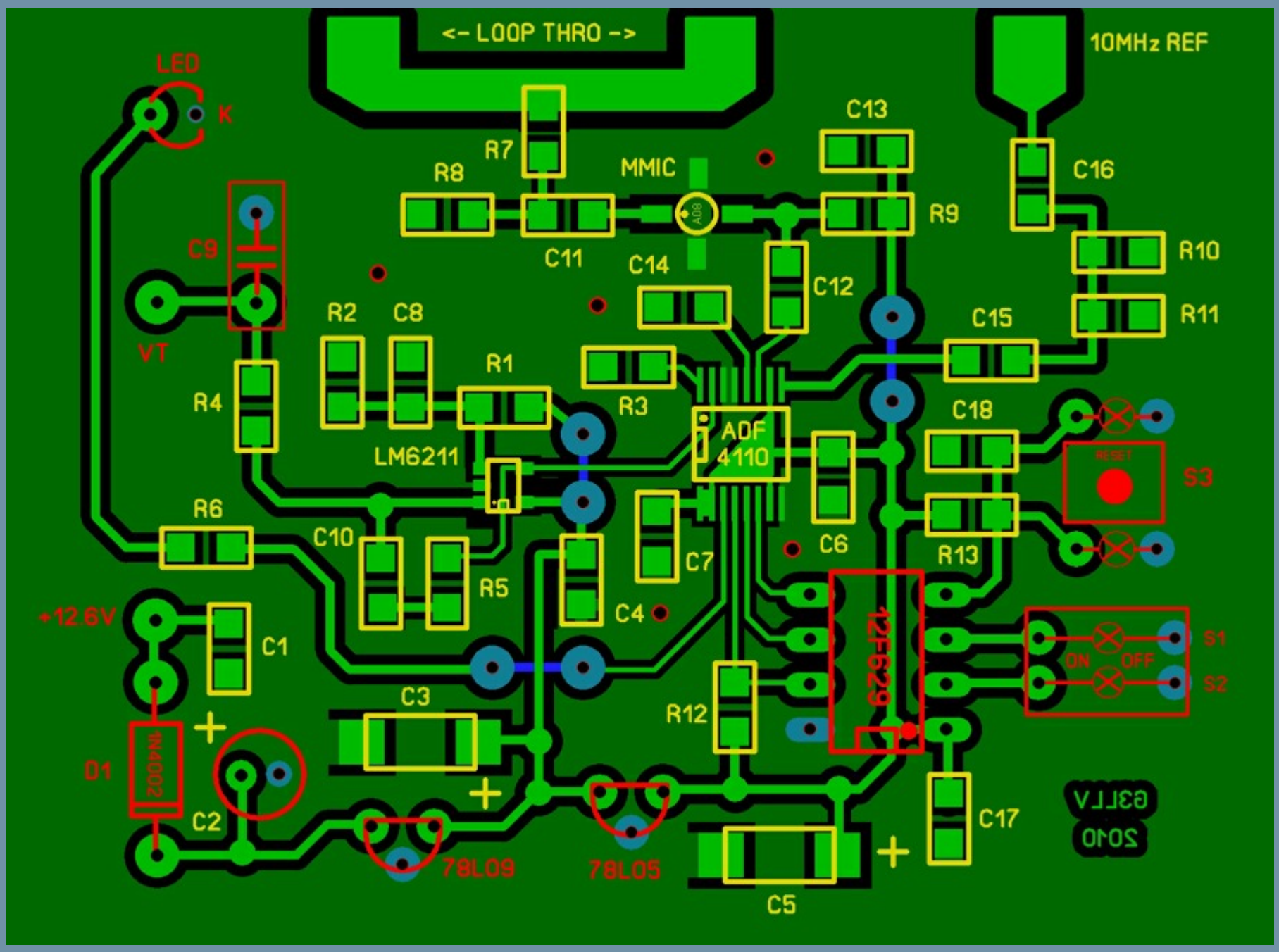
The CLK and LE waveforms for each EEPROM-to-latch load.



The DATA and four LE waveforms for a complete EEPROM block read.







The double-sided PCB layout seen from above with the YELLOW SMD and GREEN track layers underneath. The RED through-hole components and BLUE pads/links are on top together with five small RED circles indicating riveted vias. Pin-8 of the 12F629 is soldered on both sides of the board. A viewer and print utility for this layout is available on the “Software” page.

Software

MikroElektronika	-	MikroBasic 5.40 Compiler Site
Lock Box Files	-	ADF4110_Loader Project Files
	-	PCB Layout File and “Sprint” Viewer
	-	Schematic File and “sPlan” Viewer
	-	Lock_Box Spreadsheet
	-	Example ADIsimPLL File
	-	Project File List
ADF4110 Project	-	Lock Box Pages PDF
ADIsimPLL	-	PLL Simulator Request Form

Home : Schematic : Description : Layout

– Postscript –

Here is a ‘paper’ method of calculating values for the various latches of the ADF4110. The ADIsimPLL utility does not handle awkward decimal frequencies conveniently although, by rounding them off, loop filter values may be derived quite easily.

Given: Fxtal = [(P x B) + A] x Fref / R

If R is made equal to the reference frequency times the oscillator chain multiplier then [(P x B) + A] must equal the injection frequency. From there, after arbitrarily choosing P, it is simple arithmetic to work out B and A. As the lower the value of R means the higher the phase-detector reference frequency, and possibly the lower the phase noise, the results may be plugged into the spreadsheet and optimised as described there by dividing it down as far as possible with A still remaining an integer or else equal to zero.

An example calculation:

F = 5760MHz (using 144MHz transceiver)
Finj = 5760 – 144 = 5616MHz
Multiplier = 48
Fxtal = 5616 / 48 = 117.0MHz
Fref = 10MHz
Putting R = 10 x 48 = 480
So then (P x B) + A = 5616
Choosing P = 8
B = INT(5616 / P) = 702
A = P x FRACT(5616 / P) = 0

Finally: [(8 x 702) + 0] x 10 / 480 = 117.0 but, although here A = 0, R may still be divided further.