



PROJECT M254 : XY-Plotter

A high performance LCD-based XY plotter

***or how to drive high-resolution graphic LCD
panels without expensive LCD controller chips...***

The graphic LCD challenge!

The use of a graphical LCD in a project is an excellent way to drastically change its look & feel, from a technician-oriented classic 2-lines text LCD to a much more user friendly device. Unfortunately graphic LCDs are real resource-hungry devices, both in terms of memory and CPU power. So the embedded-systems designer is forced to leave its lovely minimalist designs and either select an intelligent LCD display (with on-board LCD controller and memory), or swap its usual micro-controller for a more classic microprocessor/memory/display-controller set.

Both options are expensive, but there are unfortunately no other solution, right ? For example the 320x240 pixels display used in this project eats one 4-bits nibble each 677ns, and needs a minimum of 10KB of RAM just to store the displayed bitmap, so it is impossible to directly drive it with a high-end Microchip PIC controller providing a cycle of 100ns when clocked at 40MHz, and just 1536 bytes of RAM in total, right ? Nothing useful could be done in less than seven assembly instructions per nibble, right ?

As you might expect this project shows that impossible is possible with a very optimized firmware design, and that it's even possible to use this minimalist concept to do something useful !

XY-Plotter : Overall description

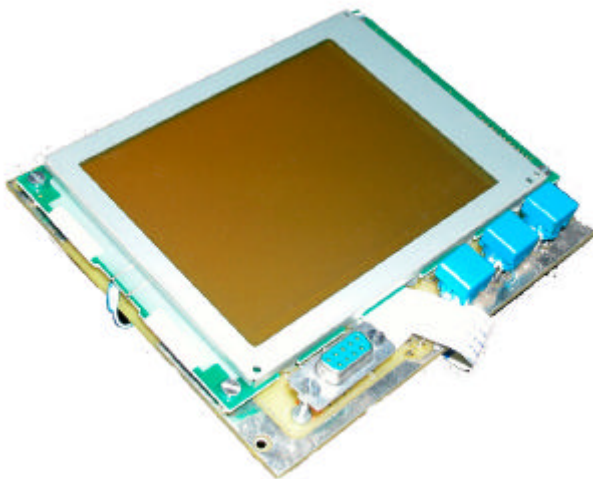


Photo 1 : The XY-Plotter prototype, front side. The large 320x240 back-light LCD is directly fixed on the PCB. The three control push-buttons and the screen-dump RS-232 connector are implanted on the bottom side too.

The XY-Plotter idea came from an old and heavy spectrum analyzer sleeping in my garage. The radio parts were still working but the CRT was dead long time ago. I used it from time to time with an oscilloscope as an output device but this arrangement was very cumbersome and uncomfortable to use. So I decided to repackage this spectrum analyzer in a prettier and smaller enclosure, and to design an LCD alternative to the CRT display. In order to have a design reusable for other projects I soon decided to develop a quite generic display subsystem : XY-Plotter (photo 1).

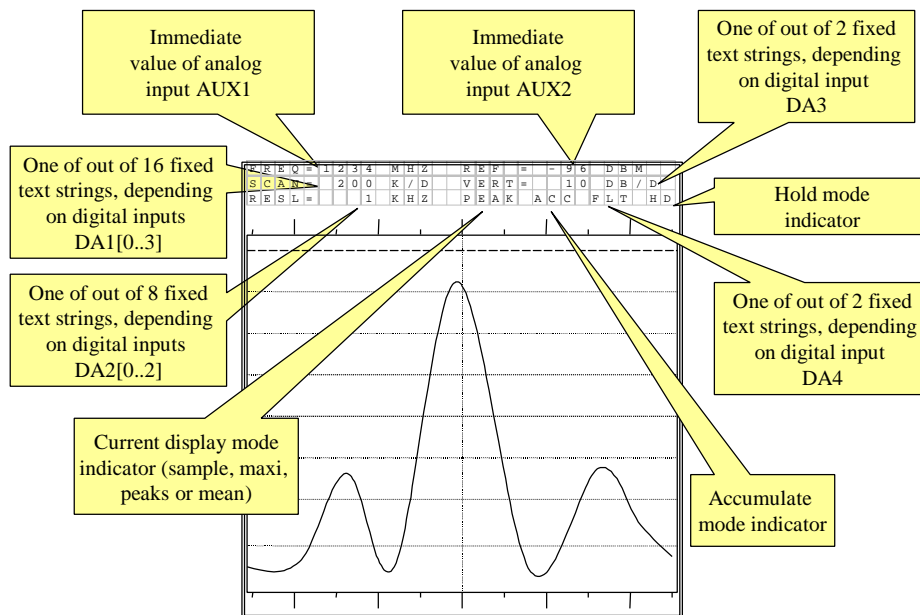


Figure 1 : The XY-Plotter screen displays a real-time X/Y graph as well as three lines of configurable textual status informations and real-time measurements.

XY-Plotter is an autonomous analog-like display, with two main X and Y inputs. It continuously scans the two inputs and displays them on a real-time X/Y graph with configurable modes : sample, maximum, peaks or average, with « accumulate » and « hold » controls. Moreover both analog and digital auxiliary inputs allows to display configurable information's on the screen like in my case center frequency, reference level, scan time, etc (figure 1). Lastly an RS-232 port is provided in order to dump hard-copies of the screen to a host computer.

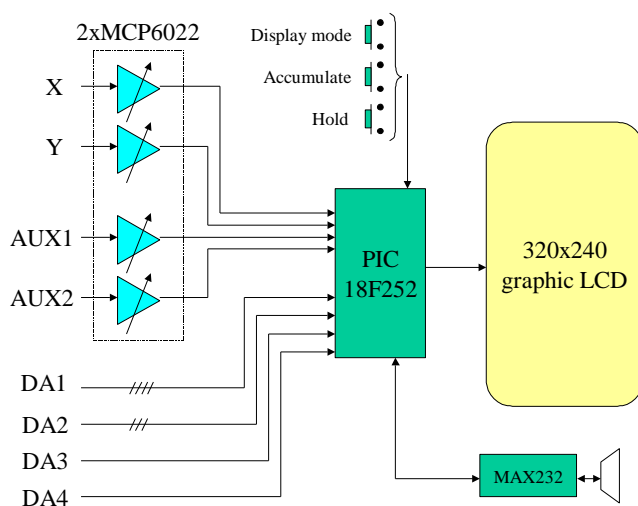


Figure 2 : An high level diagram of the XY-Plotter project highlight its simplicity, at least on the hardware side. The PIC 18F252 is managing everything including the LCD pixel generation in real time. A couple of Microchip MCP6022 rail-to-rail operational amplifiers are used to scale the analog inputs.

As illustrated on figure 2 the overall architecture of XY-Plotter is indeed simplistic : nothing more than a PIC18F252, some MCP6022 analog amplifiers, a low-cost dumb LCD display and a couple of other low-cost components. An integrated power supply is built using a MCP1541 precision voltage reference.

LCD timing requirements

Some precision's on the LCD used : It is an FTN reflective EPSON ECM-A0635-2 display, providing a 320x240 pixels black and white screen (figure 3). This type of display is quite « dumb » and needs to be supplied with the required pixels in real time : The host controller must send a new frame each 13ms, each frame including 240 lines, and each line including 320 pixels grouped into 4-bits nibbles. In addition to the 4-bits data input port the controller must also supply 3 clocks (frame, line and nibble clock). So one new nibble must be delivered each $13\text{ms}/240/(320/4)=677\text{ns}$. It should be indicated that here this LCD is used in vertical mode (320 pixels height, 240 wide), the scan lines being vertical.

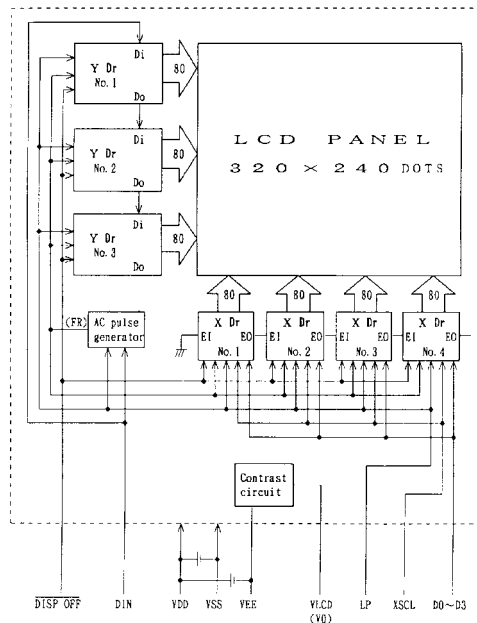


Figure 3 : Architecture of the EPSON ECM-A0635-2 display. The display doesn't include anything else than lines and columns registers and drivers. The host controller must send pixels with strict timing requirements and supply frame, line and pixel clock signals.

Micro-controller selection

Regarding the choice of the micro-controller to use the criteria were quite critic : First I needed speed. The more instructions possible in these bloody 677ns the best it is... Then I also needed a significant amount of RAM memory. As described later I decided not to store the full bitmap but at least to store min, max and sample values for each column, requiring already 768 bytes. I also needed a precision A/D converter on-board. Lastly I needed a quite large program memory in order to store the huge tables used in this design (including characters bitmaps), and Flash memory was also a must in order to configure the display for each application.

These requirements were probably impossible to fulfill one or two years ago, but thanks to suppliers like Microchip they are now easily satisfied, in particular with the PIC18F product line. Here I selected the PUC 18F252 device, with 1.5KB of ram and plenty of flash (32KB).

Detailedled schematics

The full schematics of the XY-Plotter design is given on figures 4 and 5.

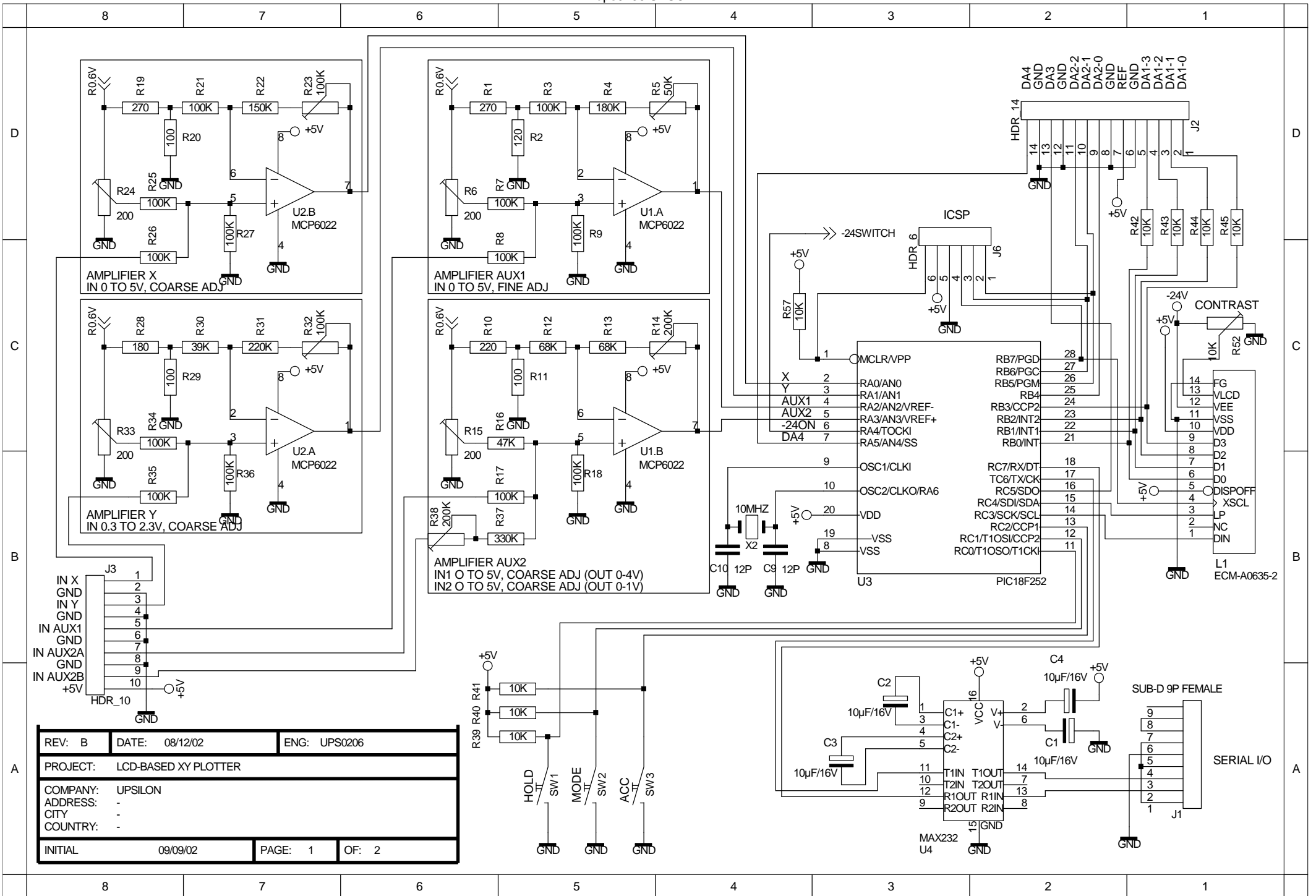


Figure 4 (previous page) : The full schematics of the XY-Plotter system, power supply excepted. Each analog input is scaled by a MCP6022 analog amplifier with both scale and offset controls. Some I/O lines of the micro-controller are multiplexed to limit the I/O count requirement.

Firstly each analog input (X, Y, AUX1 and AUX2) are conditioned thanks to half of a Microchip MCP6022 dual rail-to-rail operational amplifier. Two 20-turns trimmers per input allow to easily adjust the full scale deviation as well as the DC offset independently for each channel. One of the channels (AUX2) even includes two inputs, summed by the analog amplifier. The values of the resistors used for each amplifier stage can of course be adjusted for each specific application, to accommodate different input ranges and adjustment precision. It is not obvious to design an amplifier stage with +/- offset adjustment without any negative power supply. Here is the trick used here : a fixed positive voltage derived from a +0.6V reference is first subtracted to the input signal, then a twice variable positive voltage is added to the input signal, providing an offset either positive or negative. An excel spreadsheet was used to easily calculate all these resistors.

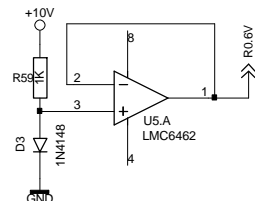
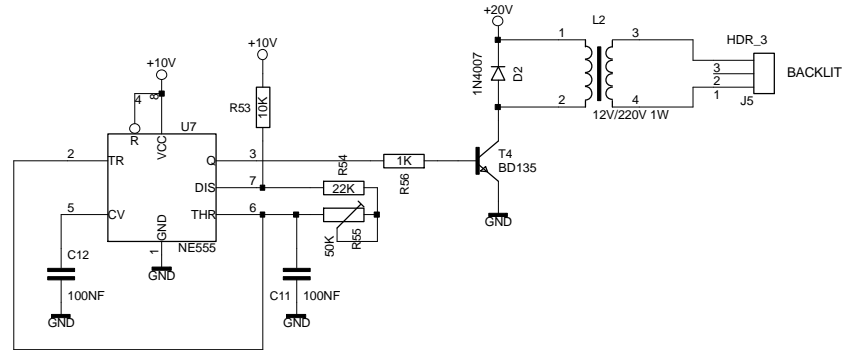
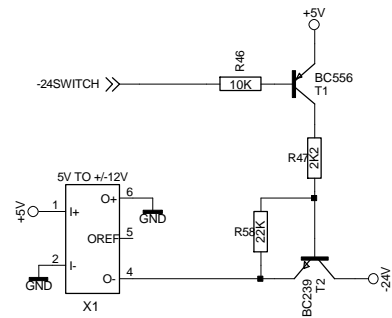
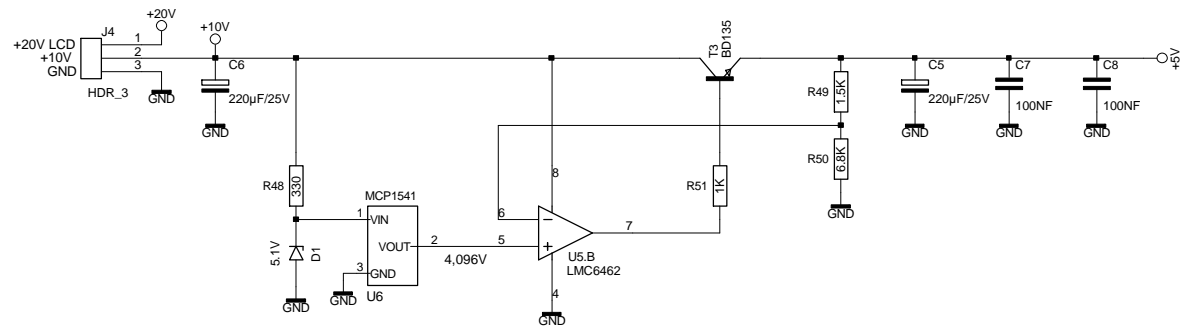
The PIC is clocked by a 10MHz crystal, up-converted to 40MHz thanks to the on-board PLL. The LCD is directly connected to the PIC I/O lines, while the auxiliary digital inputs, used to dynamically select the text to be displayed on the screen, are either direct inputs of the PIC or are multiplexed with LCD data lines (thanks to a firmware reconfiguration on the fly). Lastly an ubiquitous MAX232 does what it is intended to do. It should be noted that I included "just in case" an in-circuit programming header, that I didn't had to use thanks to the boot loader firmware provided by Microchip : all programming where done though the serial port.

Power supplies

The power supply is not really trivial (figure 5). First I needed a clean 5V. As I was already using all analog inputs of the PIC I was not able to configure it in "external reference" mode, but I still needed a very stable reference for the analog to digital conversions. After some headaches I decided to just use the PIC in its 0/5V reference mode, but to provide a very well stabilized 5V. I used a high-precision MCP1541 voltage reference and built a discrete power supply around a low-drift LMC6462 operational amplifier. The second part of the LMC6462 is used to get the 0.6V reference used by the offset circuitry.

The LCD was also quite difficult to deal with : it needs both a -24V DC input (for the display itself) and a 100V AC power for the back-light... In order to limit the requirements I used a small 5V to +/-12V DC converter to generate the -24V, commuted through two transistors under PIC control. I was not able to find a ready-made DC/AC converter for the back-light in time, but this is not an issue : I built a pretty one with a small 220V/12V transformer backward, driven by a NE555. Done.

Figure 5 (next page) : The power supply schematics includes 4 independent subsystems. First the main 5V regulator, built using a high-precision Microchip MCP1541 reference, then a 5V to -24V converter for the LCD display. The third part is a home-built converter to supply the back-light voltage (100V AC). Last a 0.6V reference is provided for offset control.



REV: A	DATE: 09/09/02	ENG: UPS0206
PROJECT: LCD-BASED XY PLOTTER		
COMPANY: UPSILON		
ADDRESS: -		
CITY: -		
COUNTRY: -		
INITIAL	09/09/02	PAGE: 2 OF: 2

Prototype assembly

I built a quite simple PCB for this project (photo 2). All the components largely fit as I wanted the size of this PCB to be identical to the LCD in order to fix it simply. It should be noted that the front panel components (push-buttons and RS232 connector) are soldered on the bottom side. All trimmers are easily accessible with a screw driver, as they are laterally shifted from one to the other.

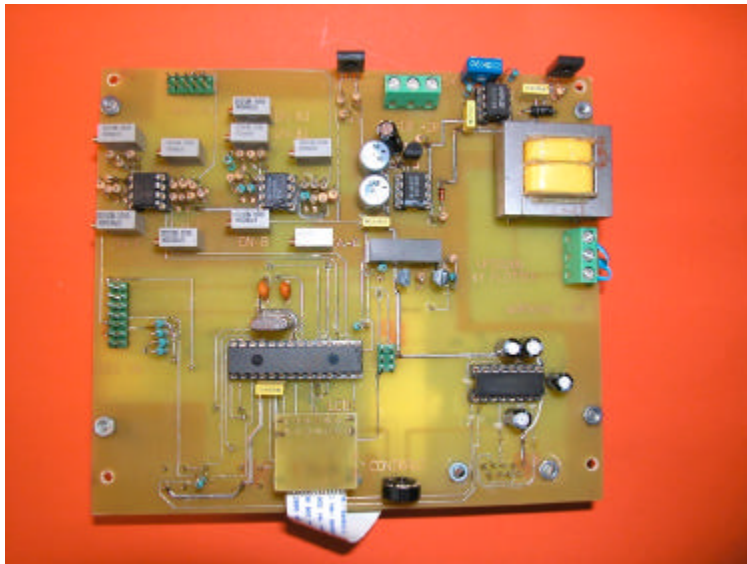


Photo 2 : The assembled XY-Plotter PCB (the LCD is on the bottom side, see photo 1). The analog front end is on the upper left with its nine trimmers, the power supplies on the top right... and the PIC in the middle. This PCB has plenty of empty space as its size was dictated by the LCD dimensions.

Firmware design

The hardware was straightforward, so you will imagine that the firmware is not and you will be right. Figure 6 illustrates its overall architecture. In order to comply with the 7 instructions per nibble requirement I didn't use any interrupt and built a fully sequential program flow.

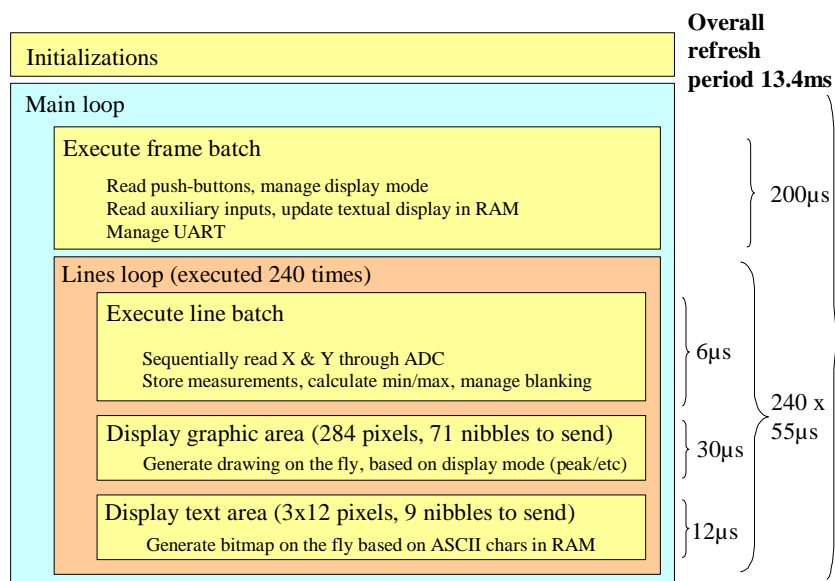


Figure 6 : Here is a high-level chart showing the architecture and timing of the firmware. The most critical section is the graphic display routine : 71 nibbles must be sent in 30µs, giving 422ns per nibble or just four PIC instructions per nibble even at 40MHz !

A main loop is executed each 13.4ms, and starts with a frame-batch routine which manages the push-buttons and more importantly reads all auxiliary inputs,

analog and digital, and generates the text that will be displayed in the first 3 lines. This text is stored as ASCII characters in RAM, using 90 (3x30) bytes. A binary to decimal conversion routine was grabbed on the Internet (see reference 4). This frame-batch routine also manages the UART using a simple protocol.

Then a loop is executed for each of the 240 columns of the display. At each iteration the X and Y analog values are read and managed (storing Y min, max and sample values for each value of X in three 256-bytes RAM areas) and managing the display « blanking » (Y is not stored when X is reducing).

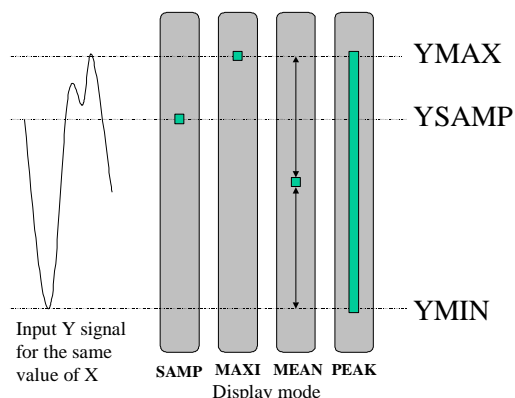
The tricky part is the last one : for each line the firmware must generate the nibbles to send to the LCD on the fly, firstly the nibbles corresponding to the graphic area (back of the screen, see figure 1), then the ones for the three text lines at the top.

Let's discuss it into more details.

Graphic display

How to generates on the fly the graphic display ? The fixed parts (borders, scales, etc) are easily sent to the LCD with the proper timing but the graph itself should be built from the min/max/sample values and depending on the display mode selected by the user (figure 7).

Figure 7 : Four display modes are supported by XY-Plotter : The sample mode just plot the first Y value acquired for each value of X, while the maximum mode plots the highest Y for a given X. The mean mode is in fact currently more a median mode (average of min and max values). Last but surely not least the peak mode display a line showing all Y values measured for a given X.



As the LCD is used in vertical mode (320 pixels height) the scan is also vertical. That means that the successive nibbles to send to the LCD correspond to successive vertical blocks of 4 pixels. In order to generate then a very optimized algorithm is implemented, based on one trick : for each column there is in this application only one « black » line surrounded by whites. First the two extremities (ystart and ystop) of this black line are calculated based on the operating mode. A loop then sends an optimal number of « fully blank » nibbles followed depending on the ystart/ystop values by pre-calculated bitmaps corresponding to the different situation and stored in a pre-calculated table as well as full black or full white nibbles in good quantity. This table, flash-based, is cached in a RAM page at startup for efficiency.

Figure 8 (here after) gives a more visual description of this algorithm.

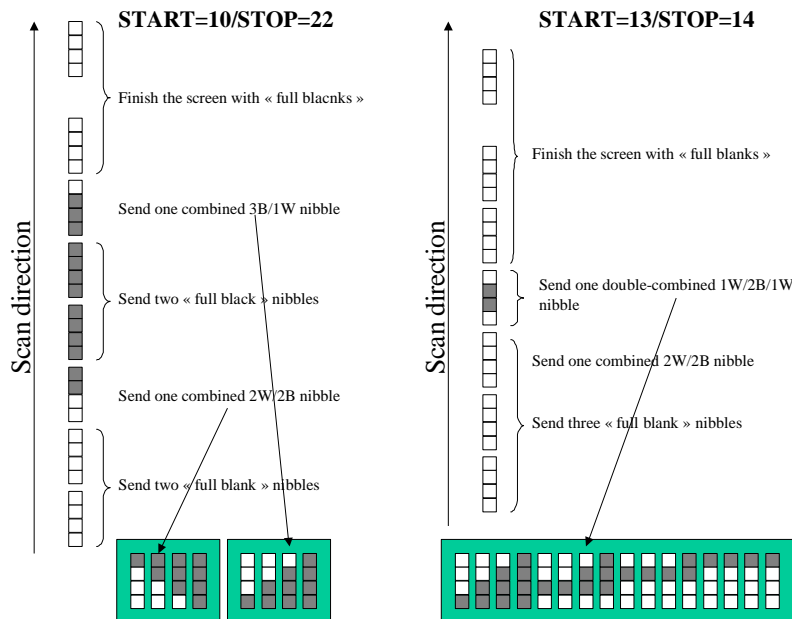


Figure 8 : This figure demonstrate how the 4-bits nibbles are generated for each scan line of the LCD. The firmware first calculates how many « 0000 » must be sent, then either one « white/black » combined nibble from a table followed by full backs if any and « black/white » combined nibble, or a combined « white/black/white » if all the black pixels to draw are included in the same nibble.

Test display

The three text lines are also generated on the fly based on the ASCII characters to display that are stored in RAM. For this purpose a specific character bitmap was pre-calculated and stored in flash. This table gives in fact the successive nibbles to send to the display for each character, from back to top and from left to right. Each character is encoded into a 12x8 pixel bitmap, giving $240/8=30$ characters per line. A significant overhead is needed at the start of each character (first scan line out of the 8) in order to pre-calculate the different pointers needed. I have built this unusual character bitmap table with an Excel spreadsheet, starting from a standard 12x8 bitmap found on the Internet.

Line batch

The line-batch routine manages the acquisition of the X and Y analog values and the storage of the min/max/sample values into RAM. This routine is build as a 5-stages step machine (figure 9), each step corresponding to a different acquisition sequence.

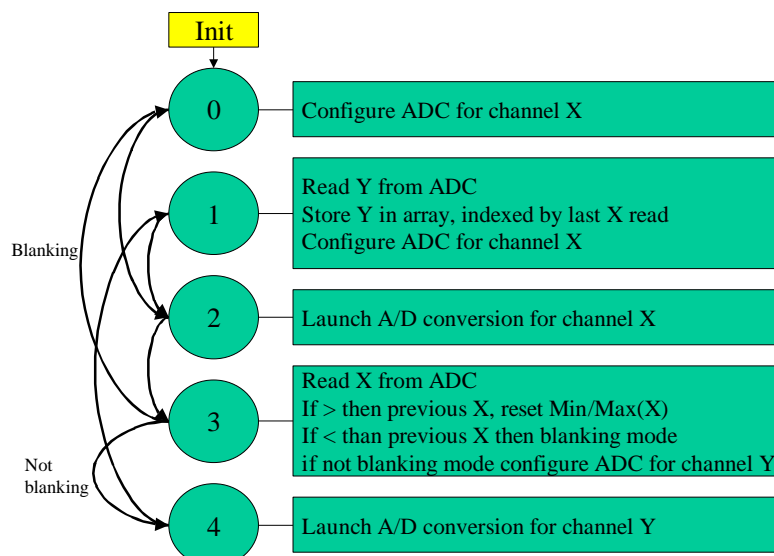


Figure 9 :The acquisition of the X and Y analog values is managed thanks to a 5-states machine executed each time the line-batch routine is called (each $55\mu s$). This allows to comply with the PIC ADC timing (precharge, then conversion, then read) without any lost time.

With this architecture no time is lost, and a full couple of X/Y values is acquired each $4 \times 55 \mu\text{s} = 220 \mu\text{s}$, giving a very satisfactory update rate of 4.5KHz. Depending on the scan rate applied, i.e. the frequency of the saw-tooth applied on the X input, two modes are automatically executed : If the scan rate is lower than $4.5\text{KHz}/256 = 17\text{Hz}$ (or scan time higher than $1/17 \times 10 = 6\text{ms/div}$ using the usual scope vocabulary) then more than one Y value is acquired for each X value each scan, enabling all functionality's like min/max/peaks. If the scan speed is higher (reasonably up to 500Hz) then an equivalent-time sampled display is generated, and min/max/peak measurements are then only available in accumulate mode (no reset of the min/max between scans).

Optimization tips

The efforts to optimize the cycles count of such a firmware are huges. For example one of its basic tasks is to send N pulses to the nibble clock input of the LCD. An usual loop already needs 5 cycles to do it, and just remember we have globally time for less than 7 instructions per nibble, and the firmware has other things to do than just sending clock pulses ! So far more optimized techniques are needed, like code expansion and calculated goto (see listing 1).

```

;-----
; Send W pulses to the XSCK line (W=0 to 60)
; Execution duration : 100ns x (2xW + 20) for W<60
; Average with W=20 (worst case) giving 3 instructions/pulse (300ns)
;-----

send_upto60_pulses          ; Limited to 60 due to page boundary
    ; input in tmp_send_w_pulses
    movf    tmp_send_w_pulses,W
    sublw   .60              ; calculate 2x(60-w)
    rlncf   WREG
    rlncf   WREG
    movwf   tmp_send_w_pulses

    goto    pulsesaligned
pulsesnotaligned
    org     (1 + high pulsesnotaligned)*.256      ; MUST START ON A PAGE BOUNDARY
pulsesaligned

    movlw   high pulsesaligned
    movwf   PCLATH          ; high byte of new PC should be defined
    movf    tmp_send_w_pulses,W
    addwf   PCL,F           ; jump to next instr if W=0 (80 pulses)
    bsf     PORTB, RB_LCDXSCL_BIT ; pulse 60
    bcf     PORTB, RB_LCDXSCL_BIT
    bsf     PORTB, RB_LCDXSCL_BIT ; pulse 59
    bcf     PORTB, RB_LCDXSCL_BIT
    bsf     PORTB, RB_LCDXSCL_BIT ; pulse 58
    bcf     PORTB, RB_LCDXSCL_BIT
    ...
    bsf     PORTB, RB_LCDXSCL_BIT ; pulse 02
    bcf     PORTB, RB_LCDXSCL_BIT
    bsf     PORTB, RB_LCDXSCL_BIT ; pulse 01
    bcf     PORTB, RB_LCDXSCL_BIT
    retlw   0                ; Must be in the same page than the first one !

```

Listing 1 : This code snippet illustrate the kind of coding techniques used to achieve the strict timing requirements of this project. Here this routine sends a configurable number of pulses to the LCD clock input, with less than 3 PIC instructions per pulse on average! Try to do it with a classic loop...

The calculated goto technique is extensively used in this project. Basically we are manually "unrolling" the code, like an optimized compiler does or tries to do. For example a long calculated goto table is used to select the specific line generation

algorithm to be used for each column of the display (graduations, plain line, ordinary curve column, etc). This gives a quite strange assembly listing to read, but quite interesting to write !

Another optimization I used is to copy the combined pixels table from Flash to RAM at startup, and an indirect access to RAM is far quicker than a table read from Flash.

Memory requirements

Of course these firmware optimizations are memory hungry, fortunately with 32KB of flash this is not an issue : my firmware currently uses "only" 10KB.

On the RAM side quite all the RAM of the PIC18F252 is used for this design : three pages of 256 bytes each are used to store the respective minimum, maximum and sampled Y value for each value of X. Then one page is devoted to the storage of the ASCII text, even if only 90 bytes are actually needed. One last 256-bytes page is also fully used to store the bitmap patterns (more on that later). 256 bytes are remaining for all general purpose variables. That's 1536 bytes in total.

Development tools & development process

This project was fully developed under the very good Microchip's MPLAB environment and simulator. I have also exclusively used Microchip's boot loader firmware (AN851) for flash burning, a quite interesting feature even if some improvements to this firmware will be welcome. In particular no on-chip debug facility is currently provided (breakpoints, etc), but I'm sure they will be there in the next version, isn't it dear Microchip folks ? It should also be noticed that this boot loader doesn't provide an automatic reentry facility : as soon as an applicative firmware is downloaded and activated there is no way to activate back the bootloader without specific user-supplied application code (here pressing simultaneously the three keys at power-up). This is well documented in the documentation but more secure solutions exists (timeout for example).

Some words on the development process I have used for this project. As I am not lucky enough to have a full-featured ICE for this processor I wanted to avoid hundreds of burn&test cycles. So I started by developing all critic code (like pixel generation algorithm) on a PC in C, just to validate the algorithm itself. Then I developed the full firmware under MPLAB, keeping a very structured approach to facilitate the validation. Then, still under MPLAB and without yet a target system, I simulated really 100% of the software with small stub routines to execute each routine individually in a bottom-up approach. I even kept a source listing and just ticked all assembly lines to be sure to go across each of them. Timings were also verified at that stage using the MPLAB stop-watch. When and only when everything seemed fine under the simulation framework I then went to the target processor. And this approach proved to be a successful one : my first burned firmware was of course not free from bugs, but I got a working display with the first burned file !

One feature also helped a lot in the final debugging steps : the RS232 interface. In fact rather than developing a specific protocol for each project I am used to adopt a very easy and powerful method : The UART firmware just dump ALL content of the RAM over the RS232 port when requested from the host. Then a

simple software on the PC side is able to grab any interesting information, like rebuilding something like a screen hardcopy, based on this RAM content. But the interesting point is that the same feature is invaluable during the debugging steps !

Wrap-up

We are December 15th and the contest dead line is this evening... I have currently spent only around 100 hours on this project, and the XY-Plotter prototype is fully operational, clearly demonstrating that the concept actually works (photo 3). The screen is refreshed 70 times per second and is free from any flicker. The A/D management and graph generation are perfect in all modes as well as the textual display.

I have of course still a couple of remaining bugs to correct, but nothing that seems critic. One of the first on the list is the fact that different columns of the display have a quite different contrast. The root cause is a variation in the execution time of each column generation routine, that will be easily corrected thanks to one of the on-board timers. A couple of more nights and this project will be ready to be embedded into my new spectrum analyzer !

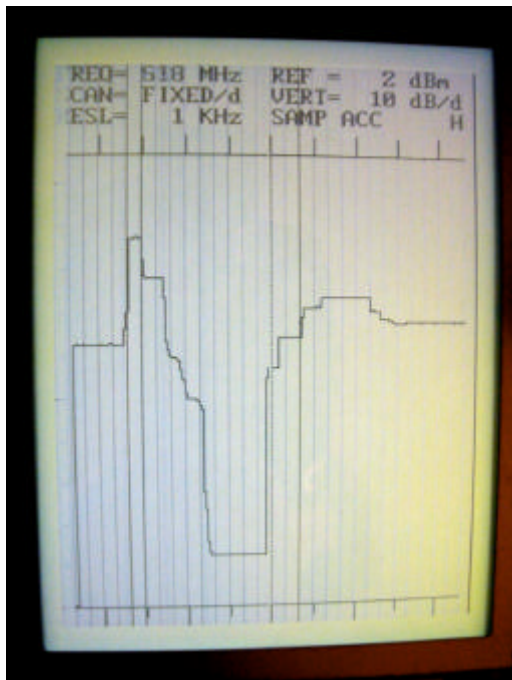


Photo 3 : XY-Plotter in action ! The prototype is currently fully operational, clearly demonstrating that the concept is a working one ! One of the bugs to correct is the slightly different contrast from scan line to scan line, due to variation in execution time. Just some firmware adjustments and that's it.

On the lessons learned side this project once again demonstrated the power of low-cost micro-controllers as well as the very efficient debugging possible with a good simulator. I also learned that LCD back-light high voltage generators are harmful but that's another story...

I have also a full list of future improvements for this project, starting with a PC-based configuration software in order to customize the display for any new application (modification of the textual information's, etc). This will be very easy thanks to the Flash based PIC used.

Globally developing this useful project was really fun, I hope reading about it was fun too !

References

- [R1] PIC18FXX2 data sheet
Microchip Technology Inc
DS39564A
- [R2] Application note AN851
"A flash bootloader for PIC16 and PIC18 devices"
Microchip Technology Inc
DS00851A
- [R3] ECM-A0635-2 LCD specification
Seiko-Epson corporation – LCD division
SC-900063501
- [R4] PIC Code for Binary to Decimal Conversion
Part of the Binary to Decimal Conversion Tutorial
www.cs.uiowa.edu/~jones/bcd/decimal.html
Douglas W. Jones
THE UNIVERSITY OF IOWA, Department of Computer Science
- [R5] 8x12 charset
www.sxlist.com/techref/datafile/charset/8x12.htm

Sources

PIC18F252
MCP6022
MCP1541
Microchip Technology Inc
www.microchip.com

MAX232
Maxim
www.maximic.com

LMC6462
National Semiconductors
www.national.com