μC

LEARN HARDWARE FIRMWARE AND SOFTWARE DESIGN

BY

O G POPA

5TH EDITION [REVISED, VERSION ED5RV07EN–15]

THE PUBLISHING HOUSE OF

Complement Control

http://www.corollarytheorems.com

Attention: in edition 5 of LHFSD we recommend that the SDx programs developed in Software Design are studied only as "programming techniques/methods". This means, some readers may use VB6 to build the SDx programs, naturally, and each program is guaranteed to work without problems. However, other readers may use ANY SOFTWARE COMPILER THEY WANT in order to build *programs that are similar to the SDx ones*. It is perfectly possible, and everything should work perfectly well.

Please be aware that, lately (well, after 2007), Laptop PCs do not have DB9 connectors anymore: they work only with USB ports. *If that is your case*, you need to buy an "*USB-to-Serial*" cable adapter. Commonly, there are two types of USB/RS232 adaptors:

1. built with Prolific® chipsets (used in ed. 1/2/3/4 of LHFSD book);
2. built with FTD® chipsets (used in ed. 5 of LHFSD book).

Attention: it is possible there are (or they are going to be) more chipsets on the market, than the two ones listed above. Regardless, you should implement *similar adjustments* as they are presented in details here. The main aspect to note is, *the buffers of the USB driver need to be as small as possible.*

Now, the thorny aspect is that each USB/RS232 chipset behaves differently.

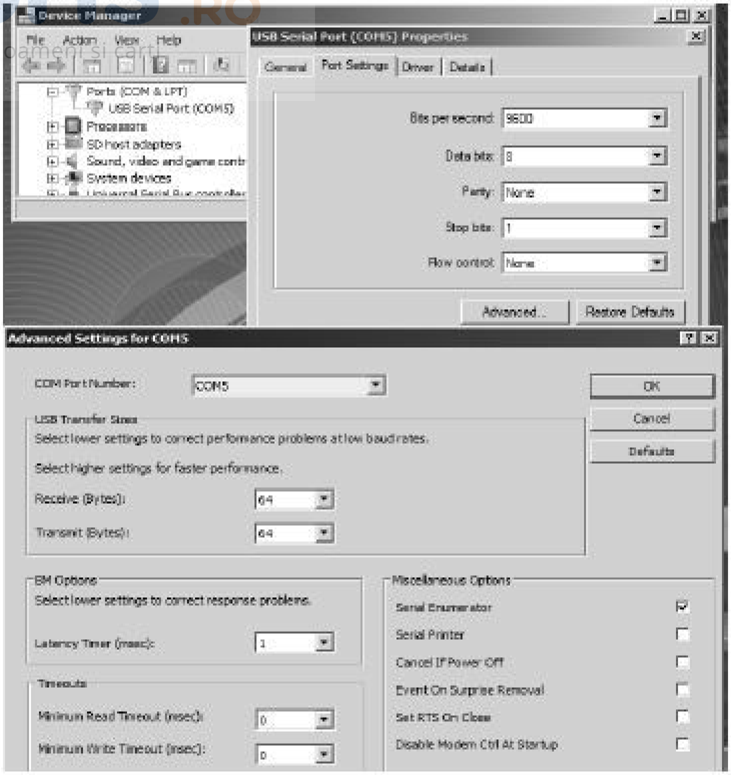
Fig R1 A Delock® USB/RS232 cable adapter employing a FTD chipset.

Each adaptor costs around 10–15 EURO. However, each has some restrictions/limitations, therefore the readers need to be aware about them.

Fig R2 A Sabrent® USB/RS232 cable adapter using a Prolific chipset.

Naturally, there are way more types of USB/RS232 adaptors on the market. However, I want to present the troubles I have with my adaptors, so that you know to what kind of problems to look for.

First of all, Prolific chipsets do not have drivers for Windows 8 at this time (October 2014); however, the FTD ones do work well an all Windows versions (XP/Vista/7/8). On the other hand, once you install the FTD driver (on any Windows version) you need to navigate to "*Control Panel>Device Manager>Ports (COM & LPT)>USB Serial Port (COMx)*": highlight, right-click, and then select "*Properties>Port Settings>Advanced . . .*"



Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

COM Port Number: COM5

USB Transfer Sizes
Select lower settings to correct performance problems at low baud rates.
Select higher settings for faster performance.
Receive (Bytes): 64
Transmit (Bytes): 64

BM Options
Select lower settings to correct response problems.
Latency Timer (msec): 1

Timeouts
Minimum Read Timeout (msec): 0
Minimum Write Timeout (msec): 0

Miscellaneous Options
Serial Enumerator ✓
Serial Printer ☐
Cancel If Power Off ☐
Event On Surprise Removal ☐
Set RTS On Close ☐
Disable Modem Ctrl At Startup ☐

Fig R3 Driver settings of Delock® (FTD chipset) USB/RS232 cable adapter.

If you work with a FTD chipset cable adaptor, your settings need to be as they are presented in the picture above; otherwise the adapter is not going to work well. In general, you need to *reduce the size of the receive and transmit buffers to minimum possible*. In the picture above, the minimum buffer size is set to 64—which is too big, but that IS the minimum possible—1 byte sized buffers would have been far better.

Prolific chipsets have smaller buffers, though it is a very good idea to reduce them as well. The procedure to follow is identical. Navigate to "*Control Panel>Device Manager>Ports (COM & LPT)>USB Serial Port (COMx)*": highlight, right-click and select "*Properties>Port Settings>Advanced . . .*" You should see something similar to picture R4.

Attention: please pay particular attention when installing your USB/RS232 drivers on your PC—do it according to manufacturer specifications. In addition, please do customize your drivers as it is explained in this chapter, or in a similar manner.

Failing to customize you USB/RS232 adaptor appropriately may result in an erratic behavior—therefore your FDx/SDx applications may not function correctly. Again, all FDx/SDx programs will function, though not correctly, as they should.

abstruse principles have been copiously denigrated and made a laughingstock by many of his contemporary peers, although we wouldn't have PCs and cell-phones today without his extraordinary work. On the other hand, it is true that he was also highly esteemed by a few mathematician-scientists who did understand his work. Among them, the preeminent British mathematician *Mr. Augustus de Morgan* [1806-1871] has sympathized with Boolean Algebra, and he brought along substantial contributions to it.

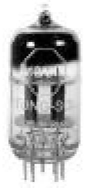


Fig 1.3 Vacuum tube/valve type EEC82.

Boolean algebra marks the apparition of *logic hardware* theoretically, in principle, in 1854. Note that the first *electronic vacuum tube/valve* was officially created in 1884 by Mr. Thomas Alva Edison [1847-1931, USA]. Unfortunately, Mr. Edison did not understand the physics behind his invention, therefore he simply patented it as the "*Edison effect*".

The vacuum tube was first implemented as *an electronic demodulating diode of AM radio signals* in 1904, by *Mr. John Ambrose Fleming* [1849-1945, UK]. In 1907, *Mr. Lee de Forest* [1873-1961, USA] is credited with the invention of the *amplification triode* (still a vacuum tube), which marks the beginning of the great, extraordinary, *Electronic Revolution*. Vacuum tubes, as electronic components in TVs, radios, tape-recorders, vinyl-disc players etc. have lasted romantically up to the end of the 20th century (up to around 1990) and everybody has been charmed by their faint, specific, intimate "hum".

The point to note is the fact that, regarding *logic hardware*, it is only the principle (the Boolean algebra) that matters; the hard-part implementation is too little important since it may be achieved in many different ways. Accordingly, logic hardware may be built using switches, mechanical relays, vacuum tubes, transistors, integrated components, FPGAs or ASICs, and so on. The physical "*form*" is utterly not important in logic hardware; what matters is only the "*function*" performed [according to Boolean algebra]. This is the reason logic hardware has appeared, in fact, in 1854, once the book of Mr. Boole was published. It was perfectly possible, at that time, to implement a complex hardware module using only mechanical switches. Of course, nobody did it, but the possibility was right there. Now, the Electronic Revolution started at the beginning of the 20th century, with the first vacuum tubes: *diodes* and *triodes*. In the other side, considering *logic hardware*, things have evolved, historically, as follows.

In 1904, the logic (hardware) principles of the Boolean algebra have been included into *Edward Huntington*'s—an American mathematician—book, "*Postulates for the Algebra of Logic*". In this way, British Boolean algebra has crossed the Atlantic Ocean in order to grow vigorous roots right into the USA university grounds. However, the first practical implementation of Boolean algebra is considered to be the 1937 MIT Master's Thesis of *Mr. Claude Shannon* [a mathematician, an engineer, and a cryptographer, the father of the "Information Theory"], published as "*A Symbolic Analysis of Relay and Switching Circuits*". That brilliant work brought the Nobel prize award to Mr. Claude Shannon in 1939, and it is considered *the Master's Thesis of the century in USA!* In addition, it is rumored that Mr. Claude Shannon is also credited with the apparition of the "*binary code*": the very essence of HFS [Hardware Firmware and Software]!

🖹 LH2: OF FIRMWARE

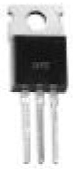
In 1937 *Mr. George Stibiz*—an engineer at Bell Laboratories—built an ingenious summing computer prototype named "*Model K*" ["K" stands for "*kitchen*"] using relays and the binary code.

Well, we should also mention the exceptional work of *Mr. Konrad Zuse* (1910-1995), a mechanical engineer at the Henschel-Berlin aircraft factory: he invented the *world's first programmable computer in 1941*, using telephony relays and *a perforated 35 mm celluloid film*. The programming language he had developed in 1945/46, named "*Plankalkül*" (from "Plan calculus"), is considered the first algorithmic (firmware) programming language. However, due to the WW2 crisis his work became acknowledged only decades later. Most patents of Mr. Konrad Zuse have been bought (and later used) in 1960s, by IBM™.

Back to the proper timeline, on USA grounds, *John Atanasoff* and *Clifford Berry* built in 1939 the first computer, named "ABC", using vacuum tubes, relays, and a rotary capacitor working as an "*electrostatic memory*". However, again due to the troublesome WW2 historical catastrophe, the ABC computer was never upgraded; in fact, it was even disassembled in 1942 to gain additional storing space for war supplies.

Beginning in 1944, *Howard Aiken* and *Grace Hopper* from Harvard University build the "*Mark*®" series of computers. *Mark 1* finished in 1946 weighted about 5 tons, and it had approximately 760 000 components manually assembled! Nicknamed "*The Manchester Baby*", Mark 1 has been used by the USA Marine for (big) guns ballistic computations up to 1959. Particularly interesting was that Mark 1 used *a punched paper tape* in order to input the data, while the output was redirected to an electronic office typewriter. In addition, Mark 1 used a special vacuum tube, named "*Williams-Kilburn Tube*", working as data memory: that was the first RAM [Random Access Memory]. The second prototype, "Manchester Mark 1", has been implemented into factory production by Ferranti Ltd.™ company, and it became the first commercial computer (re-named "*Ferranti Mark*® *1*").

Fig 2.1 A BJT transistor in a TO220 package.



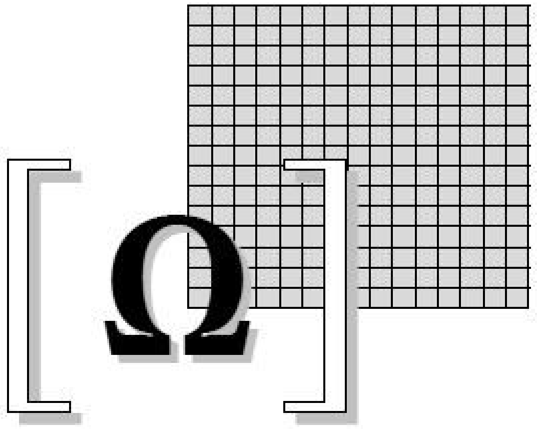
Between 1947–1948, a monumental invention has shifted the entire Electronic Revolution: the *BJT transistor* [Bipolar Junction Transistor]. A group of researchers from *Bell Telephone Laboratories*™ have discovered (entirely by accident, during their very last attempt) the amplifying property of the germanium crystals.

This landmark invention brought the Nobel prize to *John Bardeen*, *William Shockley*, and *Walter Brattain*, while the entire World was going to change radically. As a note a little bit out of context, we can present now the major "ages" in computer timeline.

CHART LH2.2: MAJOR AGES IN COMPUTER EVOLUTION

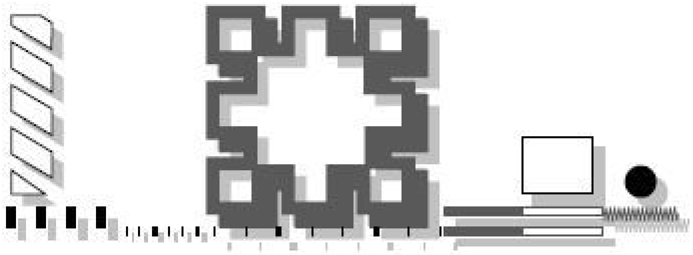
COMPUTER EVOLUTION	1ST GENERATION – VACUUM TUBES
	2ND GENERATION – TRANSISTORS
	3RD GENERATION – INTEGRATED CIRCUITS
	4TH GENERATION – MICROPROCESSORS

In 1951, *John Presper Eckert* and *John Mauchly* set a new landmark in computer evolution with their UNIVAC® [Universal Automatic Computer] commercial computer—this is, after they abandoned their engineering studies, and after five years of dire financial straits. The UNIVAC computer was in competition with similar equipments produced by the giant IBM™ organization, though it proved to be superior in operation (way faster) since it used *a magnetic tape*, instead of "*punched cards*" used by the IBM computers. The IBM organization [International Business Machine™ Corp.] is a famous name in computer history. IBM



PART 1: HARDWARE DESIGN

BUILDING THE LHFSD-HCK PCB



H1: ABOUT HARDWARE DESIGN

Amazingly, the "hardware" term is so general, and it names so many "things", that only a few hardware specialists understand the proper intended meaning in our day-to-day messages. Personally, I have been through a couple of interviews in which the professional interviewer was asking me questions about "hardware", although he/she had absolutely no idea of what that "hardware" thing really was!

The general "*hardware*" term refers to "*any physical means needed to work with/on*". Consequently, we can have "*house hardware*", "*office hardware*", "*maintenance hardware*", "*electronics hardware*", "*kitchen hardware*", "*chemicals hardware*", and so on. It is clear that, when we use the term "*hardware*", we need to explain its meaning in details. Now, "*electronics hardware*" refers to the electronic components; however, electronics hardware can be "*digital/logic hardware*" and "*analog electronics*"—the difference between these two major domains is best explained in the coming subchapter H1.1.

The term "*hardware design/development work*" [explained in H1.3], related to "*logic hardware*", refers to a few specific "*hardware subdomains*", as it is presented in the following chart.

CHART H1.1: MAJOR SUBDOMAINS IN HARDWARE DEVELOPMENT

LOGIC HARDWARE	***MICROCONTROLLER DESIGN*** [PRESENTED IN THIS BOOK]
	PGA DESIGN [VIA HDL PROGRAMMING LANGUAGES]
	ASIC DESIGN [MOST ADVANCED/COMPLEX OF ALL HARDWARE ACTIVITIES]
	PLC DESIGN [VIA "RELAY LOGIC" PROGRAMMING]

The focus in this book is on "*microcontroller hardware design*"; the other remaining domains, way more complex, are only described briefly—for orientation—in the following subchapters. Note that all different hardware development subdomains are commonly referred as "*hardware design*"; however, the differences in terms of "*the skills require*" are enormous. Just one example, the hardware design subdomain of ASIC [Application Specific Integrated Circuits] may refer to *the hardware design of an Intel processor*, which requires an entire team (a couple hundred) of ultra highly specialized engineers.

Anyway, the interest in our book is to look at such things globally, in a relaxed manner, and only one step at a time—baby steps.

H1.1 COMPARISON: LOGIC HARDWARE VS. ANALOG ELECTRONICS

The first aspect to mention about *analog electronics* and *logic hardware* is the fact that the users need to study a lot, theoretically and practically, both domains of *electronic design*. The good news is, there is a tremendous amount of documentation on the Internet about electronics; the reverse of the coin is, beginners are simply bewildered, not knowing where to start from—now, now, do not worry, dear friends, 'cause this book is going to direct you on the right path to follow.

CHART H2.6.2: TYPES OF TRANSISTORS

TRANSISTORS	*UJT* [UNIJUNCTION TRANSISTORS]
	BJT [BIPOLAR JUNCTION TRANSISTORS]
	FET [FIELD-EFFECT TRANSISTORS]

Unijunction transistors are little used [in analog electronics only], mostly as *pulse generator*. However, BJT transistors are employed a lot in hardware electronics, both in analog and in logic schematics. Now, particular to BJT transistors employed in logic hardware circuits is the fact that they are "(*2nd order*) *current controlled devices/relays/switches*".

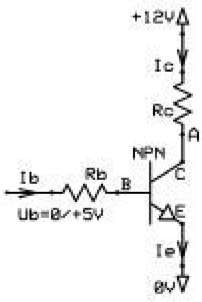


Fig H2.6.3 The schematic needed to calculate transistor equation.

The main formula used with BJT transistors is:

$I_e = I_b + I_c$ [transistor equation; saturation]
$I_b = 0$; => $I_e=0$; => $I_c=0$; [the cutoff state]

Despite its simplicity, the general formulas above allow for a few interesting observations—related strictly to the logic functions transistors perform. First of all, we have the "*cutoff*" state, when the base current, Ib, is "0": this happens while the base voltage, Ub, is smaller than 0.7 V (for NPN silicon transistors only).

Specific to the "cutoff" state is the fact that there is no Ie, therefore, there is no Ic also. Consequently, the voltage in point A is +12V. This "cutoff" state represents a logic "0", "OFF", or "false".

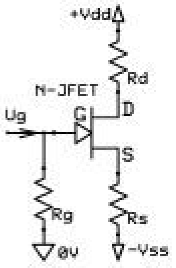
Now, if we ramp up the current Ib, calculated as Ub/Rb, past a certain limit (named the "*saturation base current*"), our NPN transistor enters the "*saturation*" state. Specific to the "saturation" state is the fact that the voltage in point A becomes 0V, therefore this state represents a logic "1", "ON, or 'true"—there is Ic and Ie current flow.

All we need to do to an NPN transistor, in order to make it work as an ON/OFF switch/relay, is to wire a Rb resistor that has a sufficiently low value (say, of 1kΩ), needed to push the transistor into the saturation mode—into the "ON" state.

Note that the base current, Ib, is the only thing that controls the ON/OFF states (again, named "cutoff"/"saturation") the transistor is in. However, these two functioning modes/states happen only when we use transistors in logic hardware circuits.

In analog circuits, transistors also work as "amplifiers" (of current, of voltage, and of frequency) which are far more difficult to calculate, to analyze, and to embed into analog circuits—again, performing analog functions. Working with analog circuits is not quite a child's play.

Fig H2.6.4 Biasing the N-channel Junction FET (having a grounding gate protection Rg).



FET transistors are different from the BJT ones, in that they are "*(3rd order) voltage controlled devices/switches/relays*". Of course, it is far more difficult to work with FET transistors; just an example, they can be easily destroyed by our body's static voltage!

However, FET transistors are the optimum technological solution for building millions of transistor circuits inside ICs [Integrated Circuits]. In fact, the mentioned FET transistors built inside ICs are named MOSFET [Metal-Oxide-Semiconductor (technology of) Field Effect Transistors]. Now, if we can ignore their different fabrication technology (of MOS type), plus their different schematic (of FET type), and of course their different required biasing/polarization circuits, then we can see that FET transistors work, logically, exactly as the BJT ones!

The thing to remember about FET transistors is, we have a "gate voltage", Ug, that controls the "ON/OFF" states the transistor is in.

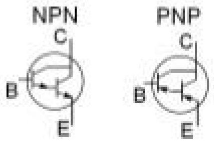
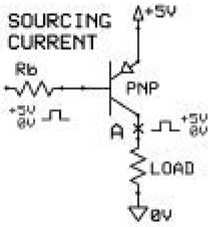


Fig H2.6.5 Schematics of Darlington transistors.

One particular type of BJT transistor is named "Darlington": this is a pair of transistors specifically calculated to work together as a single transistor. One low power transistor triggers the second one, of high power.

However, regardless the new name, and of its complex schematics, a Darlington pair works exactly as a single BJT transistor.

Fig H2.6.6. Using a PNP transistor in a "sourcing" circuit.



Now, all we care in logic circuits is the "ON/OFF" state the transistor (and the circuit) is in. However, this "ON/OFF" state can be achieved using BJT transistors in two different ways: as "sourcing (current)" or as "sinking (current)".

The "sourcing" schematic is not much used, though it is the principle that matters most. In the other side, the "sinking" schematic is used a lot. Once again, it needs to be highlighted that BJT transistors are "current controlled devices"; however, in order to achieve current-flow in one direction or another, we need the adequate *voltage polarization*—this is named "*biasing the transistor*".

As for the latest achievements in hardware design, they are waiting to be discovered, one glorious day, someplace right inside your own mind, my dear reader!

H5.1.4 SUGGESTED TASKS

H5.1.4.1 STUDY THE FAMILIES OF TRANSISTORS BASED ON THEIR PACKAGE

Use the Internet and try to discover the TO-220 package transistors capable of delivering the most current. Next, try finding a similar part of type TO-220F having an isolated collector.

H5.1.4.2 COMPARE THROUGH-HOLE AND SURFACE MOUNT RS232 DRIVER ICS

It is very important to know the size of the SM components because the PCB manufacturing price is still very high. The savings in PCB area are great when using SM components; however, populating the PCB with SM components could increase your costs dramatically.

Sometimes a mixed design could be the right solution. These considerations, and the size of the components you intend to use, need to be very well weighted before starting a new design.

H5.1.4.3 ESTIMATE THE DISSIPATED HEAT OF THE STANDARD RS232 DRIVER IC

Try to estimate the dissipated heat in mW of the MAX232N and MAX232A ICs running at 9600 Baud rate, then at 57.6 KBaud. Do not bother with the exact formula because this is only an estimation; instead, use the graphs provided in datasheets.

H5.2 SPI MODULE

The "*Serial to Peripheral Interface*" [SPI] is one of the simplest communications protocols—in terms of both hardware and firmware implementation—and also one of the most useful. *SPI is used on the PCB board only*, where it allows for extremely fast serial data exchange between various ICs—note that all ICs, plus the SPI module, form together "*the SPI Bus*". There are no clear specifications for the SPI protocol—in fact, there is no SPI protocol. At first, SPI was developed by Motorola™, then it was quickly adopted by everybody. The implementation of the SPI protocol, however, is more or less particular to each IC.

The transmission rate is limited only by the hardware characteristics of the slowest component on the SPI Bus, or by the firmware application. That data-speed could be 1, 20, 80 MBPS [Mega Bits Per Second] or even more. In terms of hardware, SPI works with many positive DC logic voltage levels. The firmware protocol specifies only *a set of serial pulses*—their number is "*as many as they are needed*". Note that communications do not have to be complex, in order to be extremely efficient.

Unfortunately, microcontroller ports 25 and 26 used for ICD3 programming interface have many other useful functions which we are not going to use. In addition to UART1, on those two pins there are SDO1 [SPI Data-Out 1] and SDI1 [SPI Data-In 1] employed by the built-in SPI driver. We managed to use UART2 instead of UART1, but there is no second SPI Bus. That is bad, though not too much because we are going to build a custom hardware and firmware SPI driver.

It is my strong belief that, if you do understand our custom SPI Bus, it will be a lot easier to understand and use later the built-in SPI driver. Fact is, using the custom SPI driver is more a matter of firmware; therefore, I am going to explain it in adequate details at firmware design-time, than I do it now.

Even more, the next chapters deal with serialized data transfers between ICs, therefore we are going to continue working on the SPI Bus hardware module. For now, I describe only the basics of building a custom SPI Bus hardware driver.

H5.2.1 SPI BUS

Please take a good look at the following picture.

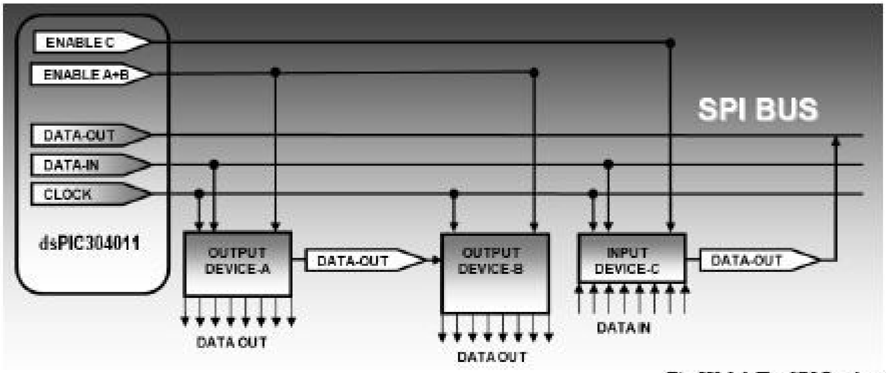


Fig H5.2.1 The SPI Bus layout.

The SPI Bus has three lines: "*Clock*", "*Data-In*", and "*Data-Out*". Very many peripheral devices can be connected to the SPI Bus, but not all of them need to have connections to all three Bus lines. In Fig H5.2.1, each peripheral device is an example of a particular case. Peripheral DEVICE-A has direct connections to the CLOCK and DATA-IN lines; its DATA-OUT port is connected serially to DEVICE-B as a DATA-IN line. DEVICE-B is connected to the CLOCK line, and it takes its DATA-IN line from DEVICE-A. Since DEVICE-A and DEVICE-B are output devices, they are not connected to the DATA-OUT Bus line.

DEVICE-C is of the input type, therefore it has direct connections to CLOCK and DATA-IN lines, then it sends its input information on the DATA-OUT line. You should note that there are many other configurations possible, based on the examples illustrated. For example, one output device of type A may enable/disable eight independent input/output devices connected to the Bus lines.

Now, the SPI Bus works this way. We send messages on the SPI Bus for each peripheral device, one at a time. Note that each of the three devices (or more) needs a dedicated ENABLE line in order to receive the message: otherwise, it cannot do it. When we send a message for DEVICE-C, we set first its ENABLE-C line high, then we send the message addressed specifically to DEVICE-C. In the same time, DEVICES A and B do exactly nothing because they are not enabled. The true beauty is, we have only three common Bus lines, plus an enable one for each DEVICEx, and this is excellent news if you take into account that one peripheral DEVICEx could have eight I/O drive lines, or even more.

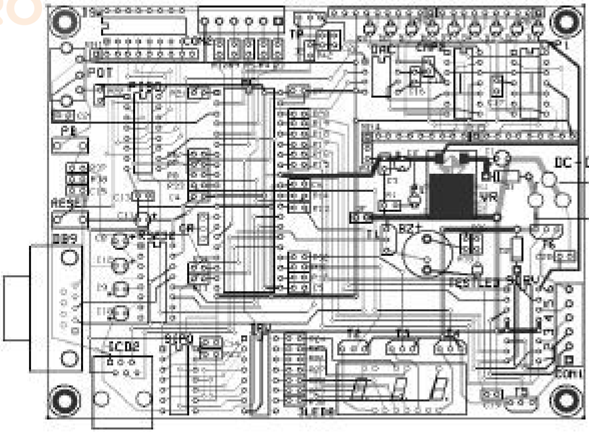


Fig H10.1.2 LHFSD-HCK V3.2 PCB with Silkscreen, plus Top and Bottom copper layers.

I am not very proud with my work in this picture, therefore it is not an example of a good practice PCB design. As you can see, I used many square corners, and there are no ground planes.

It happens that I am not quite the personification of patience—I worked all PCB traces in one 14 hrs workday—and the truth is I do not savor much repetitive work. Please find better PCB design examples than mine.

My interest in Fig H10.1.2 was to build *a functional board without mistakes*. In that respect, the Top Copper layer (darker) was drawn in horizontal traces only, and the Bottom one (gray) in vertical traces. That method of manual tracing has resulted in very many "vias", therefore I had to bend my horizontal and vertical rules a little, and take out as many vias as I was able to.

The last part of PCB design is "*Testing*". That was the most important design phase, and I allocated three full working days for that. As I mentioned before, I discovered a few mistakes not only on the PCB, but also in schematic.

Unfortunately, this is another sad reality: when we become too confident in our professional level we work leisurely, without paying much attention to small details because we know we are able to correct, later, any mistake. That is, of course, not very good, therefore *some laborious testing time is always quite necessary*.

Attention: Corollary Theorems has sold the kit LHFSD-HCK V2.2 ... V3.2 from 2006 up to 2009 (up to Edition 4 of this book). We had to cease that activity because the shipping costs became prohibitive in 2009, more than the cost of the kit itself!

Anyway, we sold our kits throughout the World, from Australia, Malaysia, India, Chile, to Europe and N. America, and we had to fight permanently with "delivery-time", "customs delays" and, naturally, with impatient customers. It was way too much trouble, therefore we had to quit.

To compensate, we decided to provide in Edition 5 the electrical schematics of LHFSD-HCK V4.00 in "ExpressSCH" format. ["ExpressPCB" program is available for free download; just search the Internet for "ExpressPCB".] Unfortunately, some readers are never "totally" satisfied: they asked us for the PCB layout of LHFSD-HCK V4.00 also.

First of all, dear readers, you cannot use the PCB layout of LHFSD-HCK V4.00 without the precise "parts list" we used, which is, in turn, dependant on particular distributors you may not be able to contact. Now, if you cannot procure a single component, then the entire PCB layout of LHFSD-HCK V4.00 is useless.

Secondly, the PCB layout of LHFSD-HCK V4.00 is subjected to the Copyright Laws [all PCBs, schematics, and technical drawings are], and we do not intend to delegate that particular right of ours. Please do not be discouraged by this aspect, because designing the PCB is *a mandatory lesson for any hardware designer*. Start with discovering the PDF drawings of all parts you need, then build your PCB accordingly.

Note that *building the LHFSD-HCK PCB may take you only a few days of work* since this is, in fact, the easiest part of hardware design (far easier than schematic design). We have provided the right schematics for you, the only trouble left is discovering the available parts in your region of the World.

Attention: we suspect that many of the kits we sold previously may be purchased as a "second hand" from eBay, or from some other places on the Internet. However, if you build the LHFSD-HCK, and you use it properly to work with this book, then you could sell the kit to other beginner designers—to recover some of costs.

Just write to us , and we are going to display your offer in a dedicated page of our Internet site. In addition, you could use your LHFSD-HCK PCB and this book in a far more constructive manner, to actually *teach the knowledge in class/tutoring activities*. Both the LHFSD book and the LHFSD-HCK have been particularly designed to be used as "pedagogical instruments". Note that education will always remain "in fashion"!

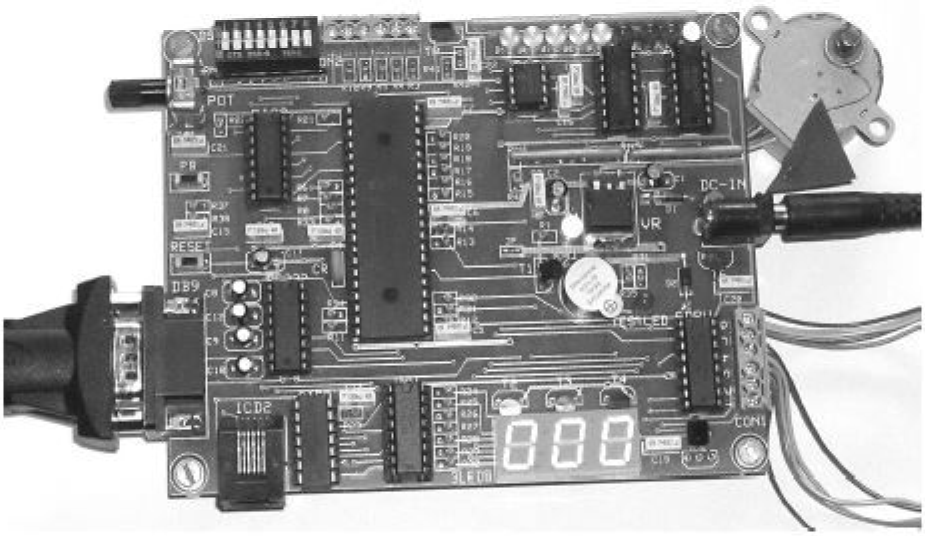


Fig H10.1.3 Finished LHFSD-HCK V2.4

In Fig H10.1.3 our LHFSD-HCK V2.4 is finished, and working. Note the connected stepper (it has a tiny red triangular flag attached), and also the USB/RS232 adaptor and the DC-IN power plug. We are ready now to start firmware and software design. However, there are still a few aspects related to the hardware design work which we do need to cover; one of the most important is BOM.