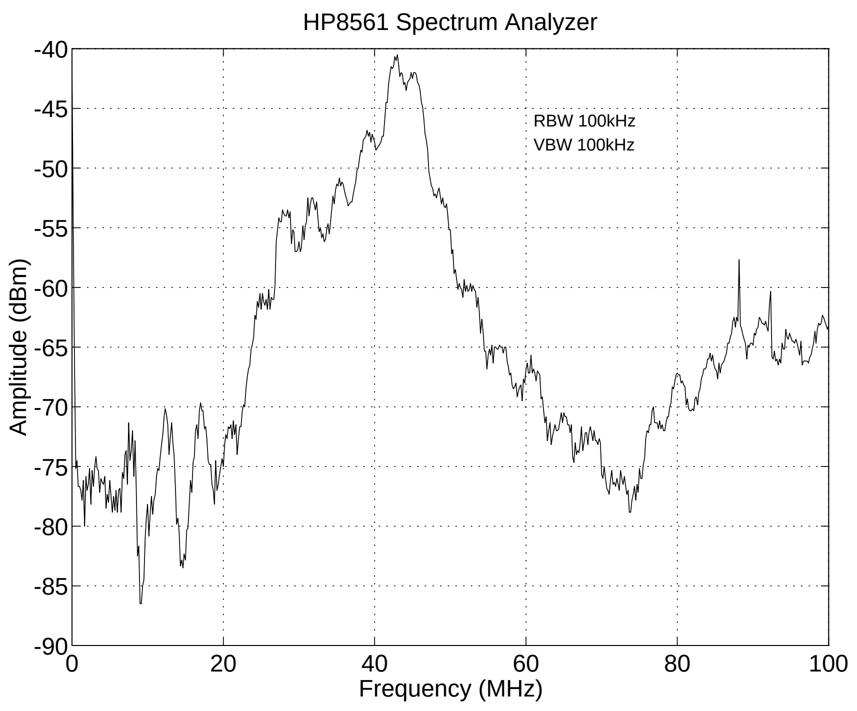


Figure 38: Noise spectra of DSP CARD 4 when in metallic box.

After we bypassed all the connections, we measured the resulting spectra when the box was closed, the result is shown in figure 39. As it is seen, no measurable emissions were detected. After those shielding and bypassing operations, it was possible to operate the R100 receiver when the DSP CARD 4 was in the box under the receiver.

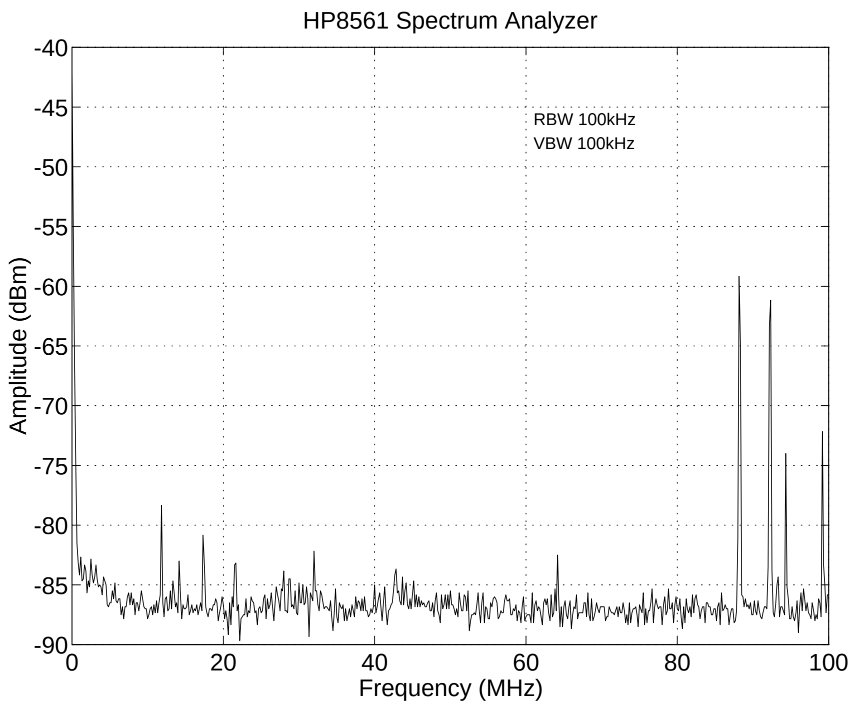


Figure 39: Noise spectra of DSP CARD 4 when in metallic box, and all the wires are bypassed.

Programming with the DSP CARD 4

“Oi sielu, rohkenetko
lähteä matkaani kohti tuntematonta seutua,
missä ei ole maata astella, ei polkua seurata?”

“Darest Thou Now, O Soul”, Walt Whitman

Devising and then implementing signal processing algorithms is a little different from ‘normal’ programming. First of all, *think analogwise when coding*. It very tempting, especially for ‘real programmers’, to use discrete like methods to accomplish the various needs for modems. For example, in order to remove the DC-level from the signal we first calculated the mean and then subtracted it from the samples. This is the common way of doing when coding usual applications; devising a simple sequential algorithm. But in digital signal processing world, there is much more powerful method—a simple high-pass filter [5]. Using first order IIR section and by selecting appropriate coefficients one can simply synthesise a filter that has infinity attenuation at the zero frequency (zero at the +1 position) and beautiful pass-band characteristics (pole at -1), so the filter actually removes the DC-component and is very easy to implement with DSP processors.

DSP56001

The 56001 User’s manual [40] and application note considering the 56001 arithmetic [41] are essential when doing serious work with the 56001. This section is not a complete introduction to the 56001 However, in this section we present only some curious facts (and troubles) we have had when programming the 56001.

DSP56001 has a very good resemblance to ordinary microcomputers, therefore the shift from microprocessors to Motorola signal processor is not so huge. In addition, the multiply and accumulate instruction `MAC` is a single instruction type that performs all the necessary data transferring, address pointer incrementations and multiplication and accumulation operations in a single instruction (and a single cycle too).

General

Relationship	*Signed*	*Unsigned*
$a < x0$	BLT	BLO
$a \le x0$	BLE	—
$a = x0$	BEQ	BEQ
$a \ne x0$	BNE	BNE
$a > x0$	BGT	—
$a \ge x0$	BGE	BHS

Table 5: This table shows which jump instructions will result in a jump taken when testing for a given relationship of a to x0 after a `CMP x0,a` instruction.

The syntax of two-operand instructions may be reversed from other machines' one is used to. For instance, the 56001 instruction `move #1,a1` is equivalent to `ld a1,#1` on some other machines. On the 56001, the destination register—the one affected by the instruction—is always second. The operand order for `CMP` instructions is also reversed from some other machines, so `cmp x0,a` means "compare a to x0".

Comparison operations

The 56001 provides the comparison operations shown in table 5. These include all six possible relationships between two signed or unsigned numbers[12]. The distinction between signed and unsigned comparison comes up rarely, since they are the same unless one of the values involved has the high-order bit (the sign bit) set.

Addressing modes

Due to design difficulties and implementation trade-offs, the 56001 doesn't have very rich addressing mode arsenal. There are only three main modes; register direct, (address) register indirect and special (actually an immediate and direct addressing modes) modes. There is no indexed mode, and one of the operands must be register. The special addressing mode is very curious, it contains both immediate and direct addressing modes. Actually one of us first believed that the 56001 doesn't have direct addressing mode at all, and always coded as follows

```
move    #$1234,r0
move    #0.34,x0
move    x0,x:(r0)
```

when with direct addressing can be coded like this

```
move    #0.34,x0
move    x0,x:$1234
```

In the special addressing mode group there are two important addressing modes, namely short forms of absolute address and immediate data. When the address or data is short enough (data 8-bit, address 6-bit, jump address 12-bit), it can be included in the program op-code and it doesn't need an additional word. This saves program memory and speeds up the

[12]Comparing the word values 000006 and fffffe hexadecimal depends on how the numbers are interpreted. If they're signed numbers, 6 is greater than -2. But if they're addresses they are unsigned, and 000006 is lower address than fffffe.

execution considerably. Often it is needed to force the compiler to use the special short forms by '<' prefix, e.g. `jmp <loop`.

Jumps

The (conditional) jump instruction on the 56001 executes in two instruction cycles. The execution timing is the same whether the jump is taken or not. The program address generator fetches both jump targets (taken and not-taken addresses) but only decodes the chosen part[13].

Using parallelism

In order to make the 56001 really singing, it is very important to utilise its parallelism fully. Sometimes this is very easy, data movement goes side by side with ALU calculations, as is seen in the following example (CRC-calculation)

```
    move            #>1,x0
    and     x0,a    y:<rem,x0
    eor     x0,a
    lsr     a       #>poly,x0
    jcc     <_c1
    eor     x0,a
_c1 move            a1,y:<rem
```

Sometimes it is advantageous to load register with parallel move even if it is not needed, as in the following

```
    cmp     x1,a    #>@cvi(@pow(2,15-1)),x1
    jne     <str4b
; yes, gain scaling (shift left 15 bits)
    mpy     x0,x1,a
```

Arithmetic and data representation

The most curious thing on the 56001 is its support for fractional arithmetic. Fractional numbers have a little different binary representation than the 'normal' integer numbers, and this may cause a little confusion. It is very important to note that 56001 assembler tries to use short immediate addressing mode whenever it can, and depending on the destination register 56001 interprets this data as fractional and therefore stores it left-side justified. For example `move #56,x0` stores 380000 hexadecimal to register x0 that is clearly incorrect. If the immediate long force operator '>' is used, an extension word will be used, and the data will be right-side justified; `move #>56,x0` stores 38 hexadecimal to register x0 which is correct. When the destination register is other than x0,x1,y0,y1,a or b the 56001 doesn't interpret the data as fractional, so `move #3,a1` is correct, and in fact it is quite beneficial to use short forms of immediate data whenever it is possible.

It is possible to enter both integer form and fractional form data to 56001 assembler. When entering the number -1 it is very important to specify what kind of number it is, because fractional -1 is 800000 hexadecimal and integer -1 is fffffe hexadecimal. So whenever using integer -1, enter -1, and whenever using fractional -1, enter -1.0.

When multiplying two **integer** numbers together, it is important to note that the result must be right sifted one bit to the left, and the result is in the lowest order word.

One interesting fact about signal processors is that the multiply instruction doesn't take very long time. So when doing multiple shifts, it is advantageous to use one multiply instruction rather than many shift operations.

[13]This is called *multiple prefetching*.

Some methodology

The most common way to program the DSP56001 is with assembler language. Of course there are other high level utilities, C compilers and code generators, but today high-performance program code must be written by hand in assembler language. The assembler provided by Motorola, ASM56000, and the accompanying linker, LNK56000, and simulator, SIM56000 by Motorola, are in public domain.

Use LEONID monitor functions to ease your interface with the DSP CARD 4 hardware. There are functions for serial line input/output, codec handling and output lines controlling. There is no need in most cases for direct hardware controlling. Beware that LEONID reserves some memory space for its own use and do not use the r3 address register because LEONID wants to use it on its own.

The complete program can be made from separate modules that are linked together. This ensures modularity and maintainability of the program. This saves also compile time, because we need compile only the user's own modules. Most often the signal processing software is so sort that it can be placed in one source file, e.g. our 9600 bit/s G3RUH-modem fits nicely in single source file.

When the editing, compiling and linking phase is over and there are no syntax errors around, we are ready to test the program. We can do it with the simulator program. It is often quite beneficial to use the simulator, because it has very powerful utilities to test and debug the programs.

After the simulator phase we can finally transfer the program to the DSP CARD 4. This is made with the `DL -g filename` command. Often the program does not work at the first phase, and the the 'trial and error' debugging method starts. Fortunately downloading does not last very long, which means that the new software code versions can be tested quickly.

So you want to build a DSP application

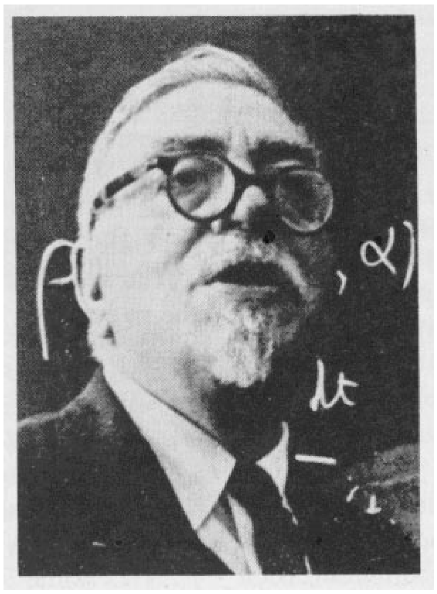
We have tried to warn you not to build a DSP application by your own. But we know that somewhere there are beings who don't make what they can, but what they want to[14]. So, when you want to build a completely new DSP application, the first thing to do is to study some theory because they don't teach us stuff like digital communication techniques in an elementary school. Do some theoretical framework to find out if there are any theoretical base for your new application construction and how to possibly implement those ideas. After devising the needed algorithms it is time to simulate (if possible) those algorithms to ensure their performance. There are plenty of algorithmic simulation tools available (we used Comdisco's SPW, but Matlab can do also, for example). After the algorithms are OK, then it is time to code them to the DSP56001. Beware the aspects mentioned in section about 56001 programming. Then simulate your code with `SIM56000`—you can use `SHOW` to display important parameters, timing error for example to detect symbol syncros performance. After all those steps it is the time to port your code to the modems skeleton and testing it with real signals.

[14]And that's good.

Alef Null ideology—how to think, how to react

"Muuan mies Sysmästä,
tunnettu oli taidosta.
Signaalinselittäjä hän lie,
ilosanomaa evankelioiden vie."

$\aleph_0$ Alef Null *Group, 1991*



Our hero

The mainstream of $\aleph_0$ Alef Null ideology is *minimalism*. Minimalism first emerged in art on 1960's. Mostly those works were three dimensional, contained extremely simple modules, were aestethically simple and were anonymous in a sense that there were no worker identification.

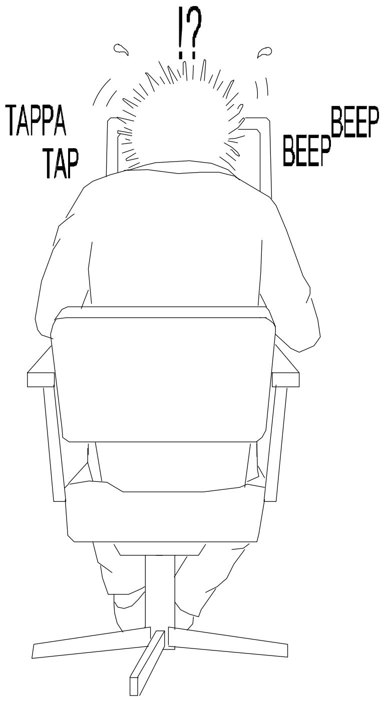
One of the most impressive musical composition is John Cage's 4'33'' that contains only thirteen pauses. In general those minimalist compositions contain no beginning and no end, they are only meditative.

In literature the most famous novel is [52]. One interesting ensemble [53] (Zero Point) contains only empty pages.

Minimalism in this DSP CARD 4 is clearly shown by its simplicity; the total component count is small and there are no bells and whistles features. In spite of this simplicity, it is although quite versatile, or at least we hope so.

"Don't ask what Alef Null can do for You,
but what You can do for Alef Null."

$\aleph_0$ Alef Null *Group, 1992*



History

Year	*Actions*
1980	first general purpose DSP chip introduced
1983	DSP technology in widespread use
1986 spring	first Alef Null speculations of DSP
1986 autumn	TMS32010 studying
1987	DSP CARD 1: M68HC11 & TMS32020
1988	DSP CARD 2: TMS320C25
1989	DSP CARD 3: DSP56001 designing begins
1990 spring	First DSP-program working (complex correlator)
1990 summer	DSP CARD 3 Rev 1.2 working
1990 autumn	DSP CARD 3 Rev 1.3 working (beta test version)
1991 summer	G3RUH with LMS algorithm working
1991 autumn	DSP CARD 4 Rev 1.0 schematics ready
1992 spring	2400 bit/s LPC Codec working
1992 autumn	DSP CARD 4 Rev 1.0 working
1993 spring	DSP CARD 4 Rev 2.2 working
1994 winter	first batch of DSP4 100 boards available
1994 autumn	DSP CARD 4 Rev 2.3 working

Table 6: Alef Null history of DSP development.

For those interested, this section explains the history of DSP-dreams of the $\aleph_0$ Alef Null Group, see table 6.

The first general purpose DSP chip was μPD7720 that appeared on the market on 1980. Intel’s 2920, introduced as early as 1979, lacked a hardware multiplier, so the 2920 did not qualify as a modern programmable DSP. In 1982, Texas Instruments introduced the

TMS32010 as the first member of what was to become the most popular DSP family. In 1983 there were several articles in EDN magazine that discussed several topics in digital signal processing.

After six years the first general purpose DSP was introduced, and three years after the DSP methods were in widespread use, we heard first time the word DSP from some amateur sources and from those old magazines like EDN. So in 1986 we knew nothing about digital signal processing, but intensive and fairly fanatic DSP research began. First we read several books on the DSP-theory and designed (but we never implemented any of them) various systems with Texas chips. In 1989 we heard some noise from USA where hams were using Motorola DSP56001 processor. We studied it, and immediately liked the beautiful architecture and easy interfacing. So the first version we actually realised was DSP CARD 3 with Motorola DSP56001. First DSP-program we created was a complex correlator which with simulator was found to be working at 04/18/1990. At 06/18/1990 11:30pm the first alpha version (revision 1.2) of DSP56001 based card was found working. We made some modifications to it (MIDI-interface, E1 size board, etc.), and the beta series (revision 1.3) was completed. After struggling several difficulties with noisy high-speed static RAM, we got the Rev 1.3 working at 12/10/1990.

This revision 1.3 was the card that has been distributed outside the Alef Null. After reviewing the comments of our beta testers we found that there are a lot of things to improve. At autumn 1991 we read from magazines [50] that Crystal Semiconductor has released a marvellous stereo Codec chip. We immediately ordered data sheets and started tinkering our design. At the same time on the software side we implemented 2400 bit/s LPC vocoder that immediately got a success as a special DSP-application.

It was found that four layer boards are very common technology in contemporary industry and that the price in large quantities is not prohibitively high. At autumn 1991 we had a new design with that Crystal stereo Codec, but no money for the PCBs and components. However we had some luck because the Laboratory of Space Technology at our university needed some DSP-devices for their satellite project and therefore funded the DSP CARD 4 project. We designed and assembled the boards and they gave the necessary funding.

At summer 1992 we got PCBs for the new DSP CARD 4 and immediately assembled one board. There were disappointedly many bugs in our design and the resulting hacked board was not very beautiful, but during the autumn 1992 we got it working!

The alpha series had some very annoying bugs and the board was populated with ‘full satisfaction boards’ as we call them in Alef Null (thanks to Rob, PE1CHL, who located one of the bugs just by looking carefully enough the schematic!). Furthermore because of limitations of the CAD software, the board was divided in two files and full electrical rule check was therefore not possible. (In fact Alef Null’s Mr. Router decided that he will never again touch his fingers on that %&Ï#$œ@ software. Now AN is experimenting with Ultiboard, which is not very good either, sic!.)

February 11, 1993 we assembled and tested the latest version, Rev 2.2, of DSP CARD 4. This version started operating immediately we switched it on. After the successful start we started to make software for this beautiful board. Monitor program needed more functions, amateur modem software had to be ported from the older DSP board versions. After we have the limited amount of software available, our local amateur club, called as ‘Radioamatööritekniikan seura’, RATS r.y., made a small batch (100 boards) of this revision of the board available for the public. These boards were sold to enthusiastics all over the world.

After the great success of the DSP4 Rev 2.2 our club made the decision to make more of those boards. At that time we had a workstation version of the our PCB CAD software available. This new version had much improved copper area fillings capabilities, so we wanted to make a new revision of the DSP4 board, called Rev 2.3. This revision had improved ground and power planes, codec layout conforming the latest specs from the Crystal Semiconductor and smaller layout improvements. January 4, 1995 one board of this new revision was connected to the power supply, and it worked ok immediately (yes, we are wondering that too!). The codec input SNR of this new layout improved 10 dB and finally conforms the Crystal specs (84 dB).

Conclusions and Future Speculation

Currently it seems that the chosen design paradigm—minimalism—has been quite successful. The resulted board is very useful and quite cheap. There is only one processor on the board, and the selected processor, Motorola DSP56001, has shown to be very powerful and easy to program. Especially the orthogonal instruction set (only 62 distinct instructions), powerfull bit-handling capabilities and non-pipelined single-cycle MAC unit has made the life of programmer more lenient. We have even programmed low-level AX25 protocol handling for DSP56001, thus DSP CARD 4 can be used, for example, as a KISS TNC replacement.

The DSP56001 has the advantage of being a relative late comer to the DSP processing ranks. Introduced in 1986, the 24-bit fixed-point DSP56001 had the luxury of smaller silicon processes and increased silicon budgets. The Motorola engineers who designed the processor integrated microcontroller buses and architectural concepts with a DSP MAC core and X and Y memory blocks. Thus, unlike many early fixed-point DSPs, the 56001 has a versatile external memory bus, standard bit-manipulation capabilities, and the ability to execute directly from external memory with single-cycle accesses.

Motorola is the third largest DSP chip seller (market share 15%) after Texas Instruments (46.7%) and AT&T (18.7%) [36] and it seems that Motorola is continuously developing its processors. Motorola has introduced DSP56002 and DSP56004 chips for the DSP56XXX family. Those new chips have same capabilities and performance as the Analog Devices ADSP-2111 and Texas Instruments TMS320C50. It seems that the DSP56001 will be the cheapest version of Motorola DSP56XXX series, and there will be no changes to the base architecture of Motorola signal processors [37].

Just now we are perceiving a tremendous increase in DSP applications. It is expected that the general-purpose DSP-chip market is growing about 30% in a year, just after ten years of TMS32010, the first popular general-purpose DSP-chip. There has been some contemplation if those DSP chips are obsolete [35] but just now there are no signs of the death of general-purpose DSP processors.

At this moment there are not so many application programs for the DSP CARD 4, but it seems that new DSP CARD 4 users are enthusiastic enough to program new applications for this board.

"Olen johtanut sinut harhaan ja
seisot talvisen vuoren äärellä
ja koet tähystää oksien halki,
keisaria jota ei olekaan."

"Synnyinmaa", Paavo Haavikko, 1955