Digital Signal Processing with Dsch 3.5

Etienne SICARD
Professor
INSA-Dgei, 135 Av de Rangueil
31077 Toulouse – France
www.microwind.org
email: Etienne.sicard@insa-toulouse.fr

Vinay SHARMA
Ni2designs
vinay@ni2designs.com
www.ni2designs.com

This document describes basic concepts on Digital Signal Processing (DSP), including coding of numbers, fixed point principles, adder and multiplier structure, and a 4-bit implementation of a Multiply-Add-Accumulate Structure.

1. Coding of numbers

1.1 Hexadecimal, Integer, Binary

The correspondence between hexadecimal values, integer and binary values is reported in Table xxx.

Hexadecimal	**Integer**	**Binary**
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Table 1: Hexadecimal vs. integer and binary values

Load “Dsp/UInt4bit.sch” as shown in Fig. xxx. The schematic diagram includes an hexadecimal keyboard. When clicking “A”, the outputs “8” and “2” are on, leading to “1010” which is displayed as “A” with the hexadecimal digit and 10 with the unsigned digit.

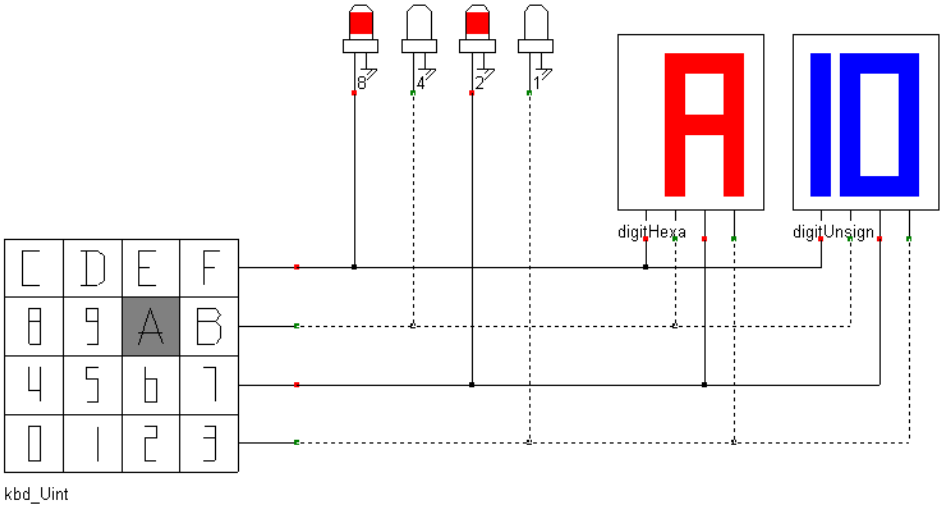

Figure 1 : A 4-bit hexadecimal input "A", its binary, hexa and unsigned integer interpretation (Dsp/UInt4bit.sch)

The hexadecimal display (digit.sym) can be reconfigured as hexadecimal, unsigned or signed integer by a double click in the symbol.

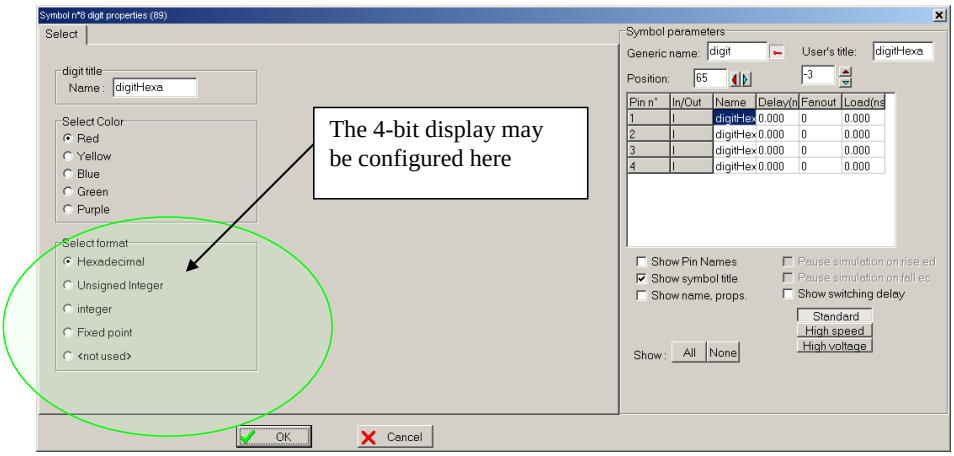

Figure 2: configuring the display in various formats (digit.sym)

1.2 Unsigned Integer Format

A summary of integer formats is reported in table xxx. The signification of each bit is given in figure xxx. The unsigned integer format is simply the series of power of 2.

Type	Size (bit)	Usual name	Range
Unsigned integer	4		0..15
	8	Byte	0..255
	16	Word	0..65535
	32	Long word	0..4294967295

Table 2: Size and range of usual unsigned integer formats

2^3	2^2	2^1	2^0

Unsigned integer, 4 bit

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Unsigned integer, 8 bit

2^{15}	2^{14}	2^{13}	..	..	2^4	2^3	2^2	2^1	2^0

Unsigned integer, 16 bit

2^{31}	2^{30}	2^{29}	..	..	..	2^4	2^3	2^2	2^1	2^0

Unsigned integer, 32 bit

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
....
$2^{10} = 1024$
$2^{15} = 32768$
$2^{20} = 1048576$
$2^{30} = 1073741824$
$2^{31} = 2147483648$

Table 3: Unsigned integer format

As an example, the number 01101011 (0x6B) corresponds to 107, as detailed in equation xxx. The illustration of the coding in binary, hexadecimal and unsigned integer format is proposed in the schematic diagram of Figure 4.

$$01101011 = 2^6 + 2^5 + 2^3 + 2^1 + 2^0 = 64 + 32 + 8 + 2 + 1 = 107 \qquad \text{(Equ. xxx)}$$

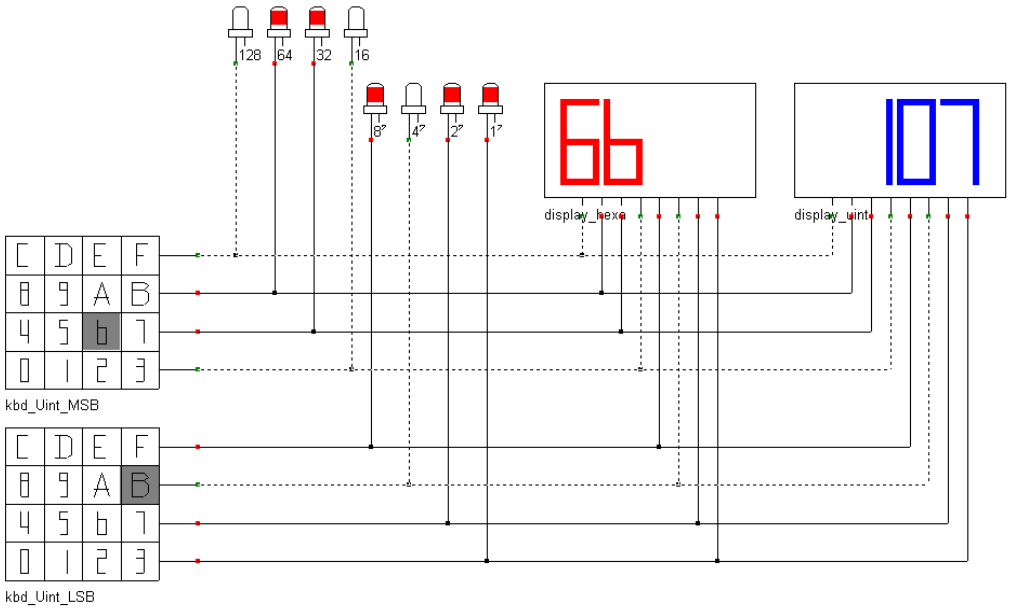

Figure 3: coding of an 8-bit value in binary, hexadecimal and unsigned integer format (dsp/ UInt8bit.sch)

1.3 Signed Integer Format

Signed integer	4		-8..+7
	8	Short integer	-128..+127
	16	Integer	-32768..+ 32767
	32	Long integer	-2147483648..+ 2147483647

Table 4: Signed integer format

The main difference between "signed" and "unsigned" integer is the interpretation of the most significant bit (MSB). The coding of the data works as for the unsigned integer, except that the left-most bit accounts for a negative number. In 4-bit format, a 1 in the sign bit equals to adding -8; in 16-bit format, a 1 in the sign bit equals to adding -32768.

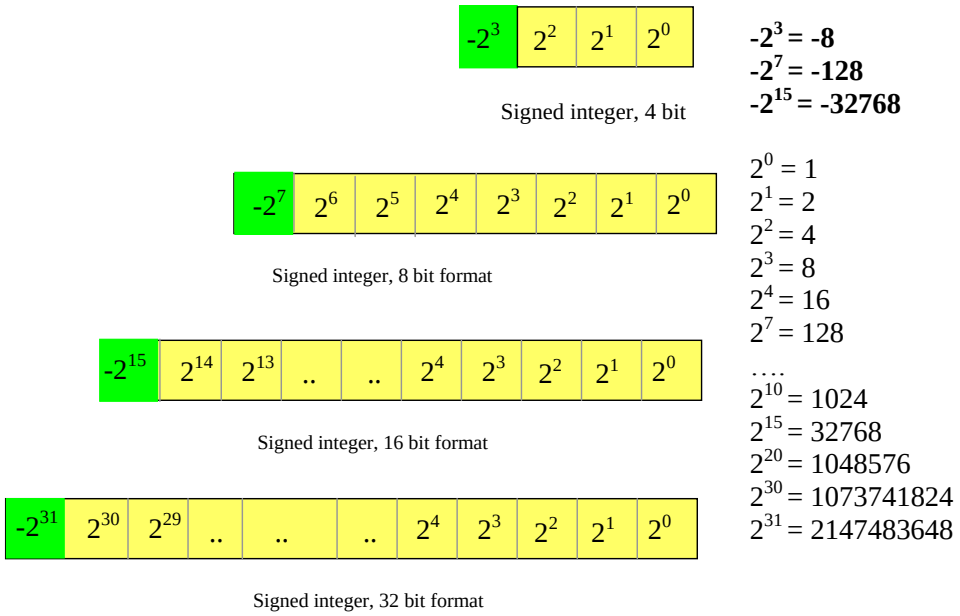

Figure 4: Signed integer format

Load "Dsp/Int4bit.sch" as shown in Fig. xxx. The schematic diagram includes a specific keyboard "kbdsigned.sym" available in the symbol library. This specific keyboard encodes 4-bit signed integers into 4 logic outputs. As an example, the 4-bit signed number 1111 is detailed in the equation below. The sign bit appears in the sum as -8, all the other bit remaining positive.

$1111 = -2^3 + 2^2 + 2^1 + 2^0 = -8 + 4 + 2 + 1 = -1$ (Equ. xxx)

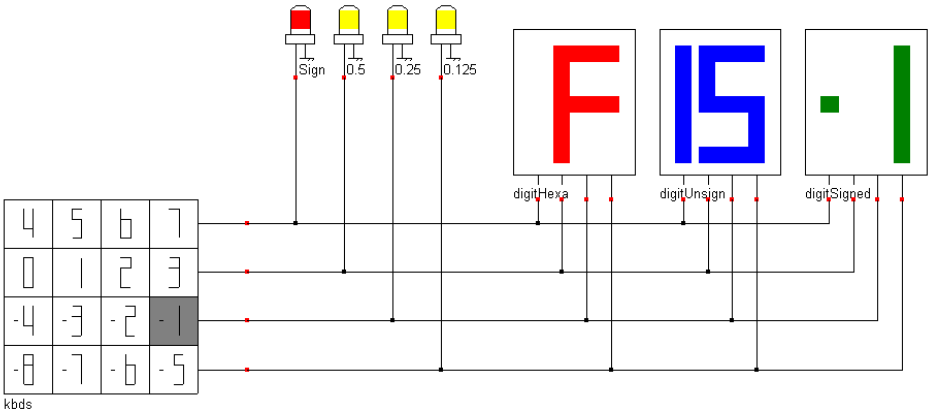

Figure 5: the integer "-1" and its binary, hexadecimal, unsigned and signed interpretation.(dsp/Int4bit.sch)

1.4 Fixed Point Format

In DSP, the most important class of numbers is the real format. Low cost and low power digital signal processors use a specific real format called "fixed point". The key idea is to restrict the real numbers within the range [-1.0..+1.0] and to use a simple arithmetic hardware that is compatible with integer hardware. A summary of fixed-point real formats is reported in table xxx.

Type	Size (bit)	Resolution	Range
Fixed point	4	0.125	-1.0..+1.0
	8	0.0078125	-1.0..+1.0
	16	0.00003	-1.0..+1.0
	32	0.00000000046	-1.0..+1.0

Table 5: Size and range of usual real formats

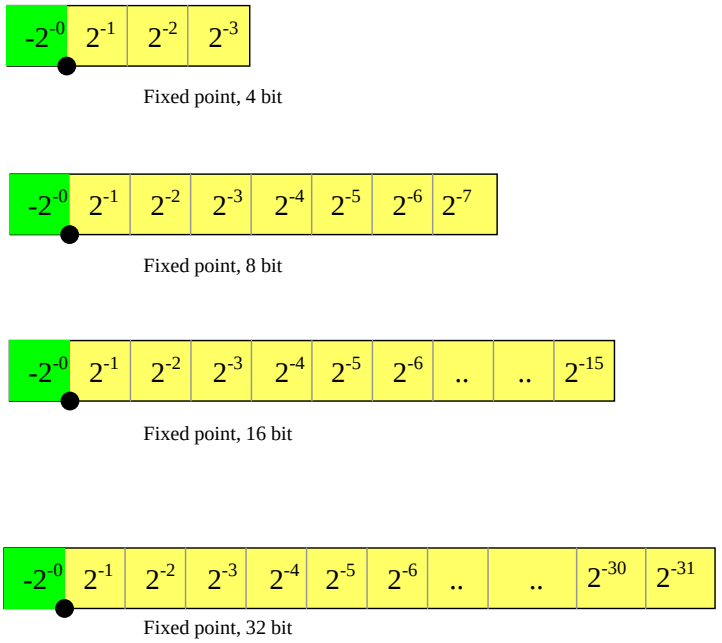

Figure 6: Fixed point numbers in 16 and 32 bit format

In the case of fixed point arithmetic, we read bits as fractions in negative power of 2 (Fig. xxx). When the left-most bit is set to 1, it accounts for -1.0 in the addition (Equation xxx). The main limitation of this format is its limited range from -1.0 to 1.0. Its main advantage is a hardware compatibility with integer circuits, leading to low power computing, a particularly attractive feature for embedded electronics. Most digital signal processing of mobile phones work in fixed point arithmetic, in 12-16 bit format. Two examples of encoding of 8-bit fixed-point numbers are reported in Figs. 7 and 8.

$$01100100 = 2^{-1} + 2^{-2} + 2^{-5} = 0.5 + 0.25 + 0.03125 = 0.78125$$

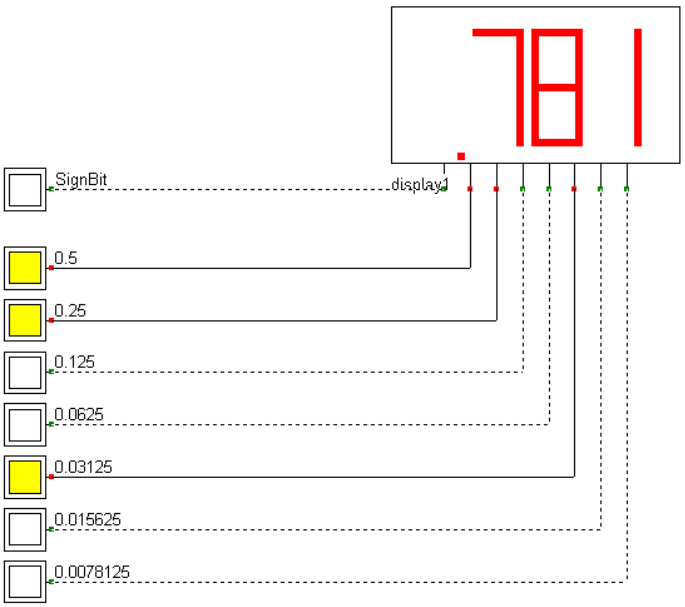

Figure 7: Encoding of 0x64 (0110 0100) using dsp/FixedPointBasics.sch

$$11100100 = -2^{0} + 2^{-1} + 2^{-2} + 2^{-5} = -1.0 + 0.5 + 0.25 + 0.03125 = -0.21875$$

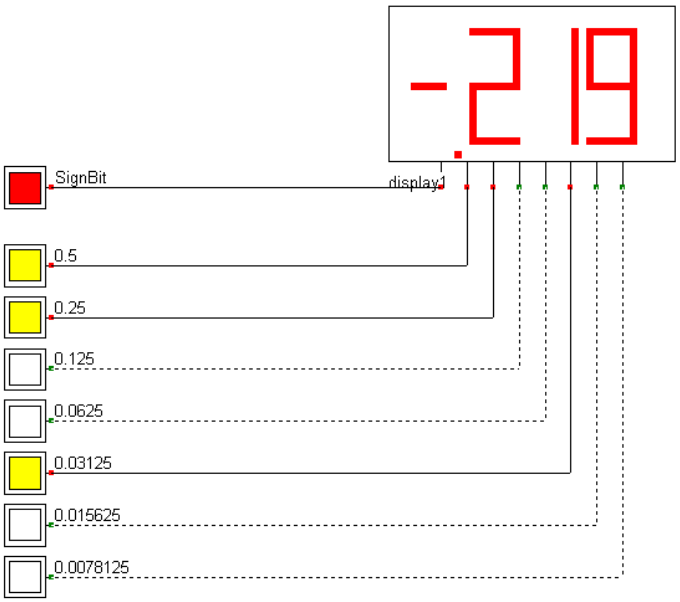

Figure 8: Encoding of 0xE4 (1110 0100) using dsp/FixedPointBasics.sch

1.5 Fixed point source

A convenient symbol called "fixedpoint.sym" enables to produce an 8-bit fixed-point signal. The symbol may be found in the symbol library, by a single click on the button "symbol list" in the palette, or through the command "File > Insert User Symbol". Double click the symbol and change the value, which should be within the validity range [-1.0..+1.0].

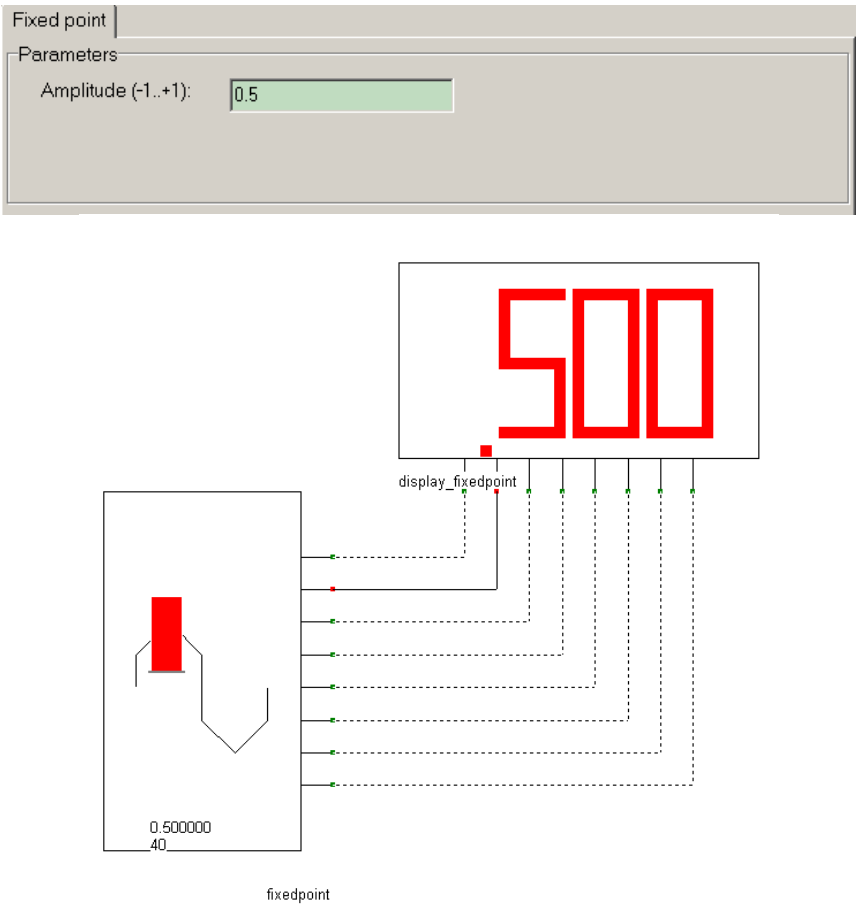

Figure 9 : Generating an 8-bit data from a fixed-point real value using the symbol "fixedpoint.sym" (dsp/fixedPointSource.sch)

2. DSPs in very short

In very short, Digital Signal Processors are able to transform a digital signal x[n] into a signal y[n] (Fig. xxx). In signed format, x[n] and y[n] range from -1.0 to 1.0. One of the most common transformations is filtering. For example, a very-low complexity low-pass filter is described by its recurrence equation as follows [MentorDSP].

$$y[n] = 0.39 \times x[n] + 0.61 \times x[n-1] + 0.39 \times x[n-2] \quad \text{Eq. 1}$$

Low pass means that if x[n] is a low frequency sinusoidal wave, y[n] is close to x[n], in other word, the DSP let the signal goes through without attenuation (Fig. 10-a). If x[n] is a high frequency sinusoidal wave, y[n] is an attenuated version of x[n] (Fig. 10-b).

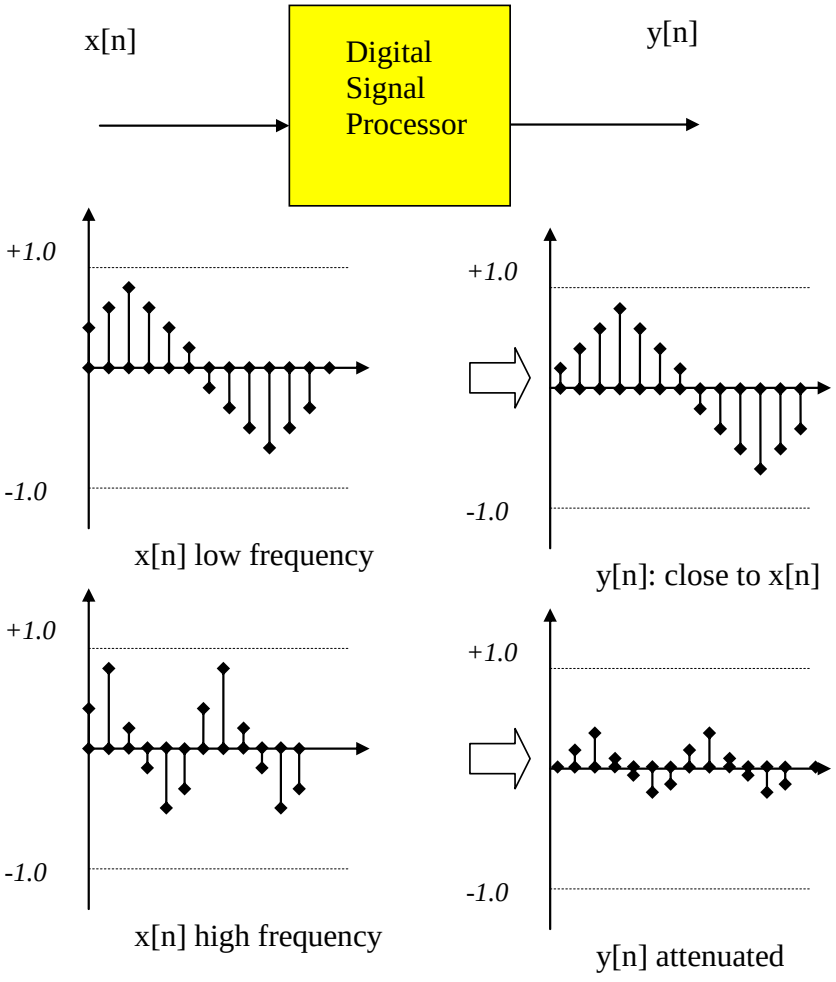

Figure 10: a DSP used as a low-pass filter

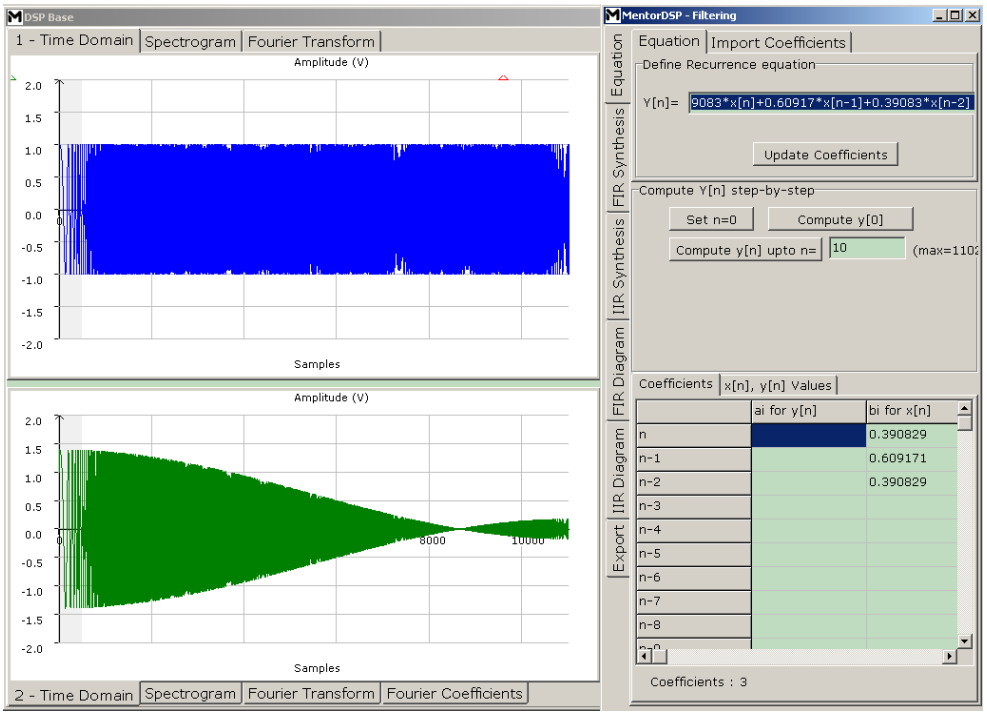

Figure 11: time domain response of Equ.1 to a chirp signal x[n] (top), showing that y[n] bottom is unchanged at low frequencies and attenuated at high frequencies [MentorDSP]

Using MentorDSP, we may see the effect of the filter (equ. 1) on a chirp signal that has a fixed amplitude of 1.0V and varying frequency. It clearly appears that low frequencies are unchanged, while high frequencies are attenuated (Fig. 11).

A generic DSP structure that can implement and execute equation 1 is proposed below. It consists of memory elements, multiply and add structures. The equation 1 concerns a second order (because of n-1) finite-Impulse-response filter (because it only includes x[n-i] elements). If we identify variables, $a_0=0.39$, $a_1=0.61$, $a_2=0.39$. We highlight the basic structure consisting of an adder (yellow), a multiplier (red) and a memory (blue).

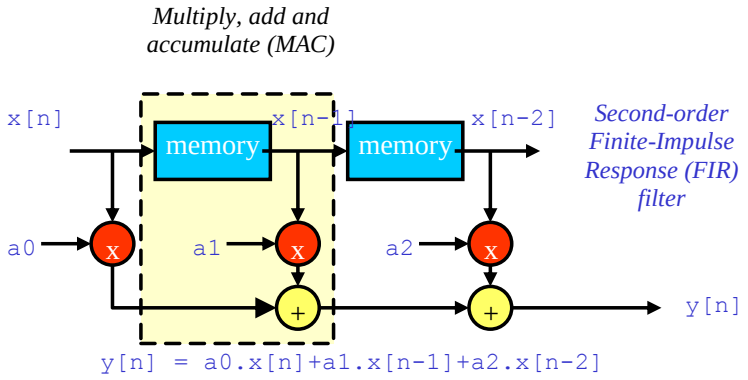

Figure 12: generic DSP structure implementing the filter, and illustration of the Multiply-Add-Accumulate structure.

3. MAC structure

The Multiply-Add-Accumulate (MAC) is the generic component of a DSP. In this paragraph, we illustrate the fixed-point addition and fixed-point multiplication.

3.1 Fixed point addition

One of the most important circuits to be build the MAC structure (Fig. 12) is the adder, in signed format. The figure below shows a 4-bit fixed point adder. It can be seen that the adder structure is exactly the same as for the 4-bit unsigned integer addition. In other words, the fixed-point arithmetic's consists in interpreting in different way unsigned integer arithmetic, without any heavy structural change.

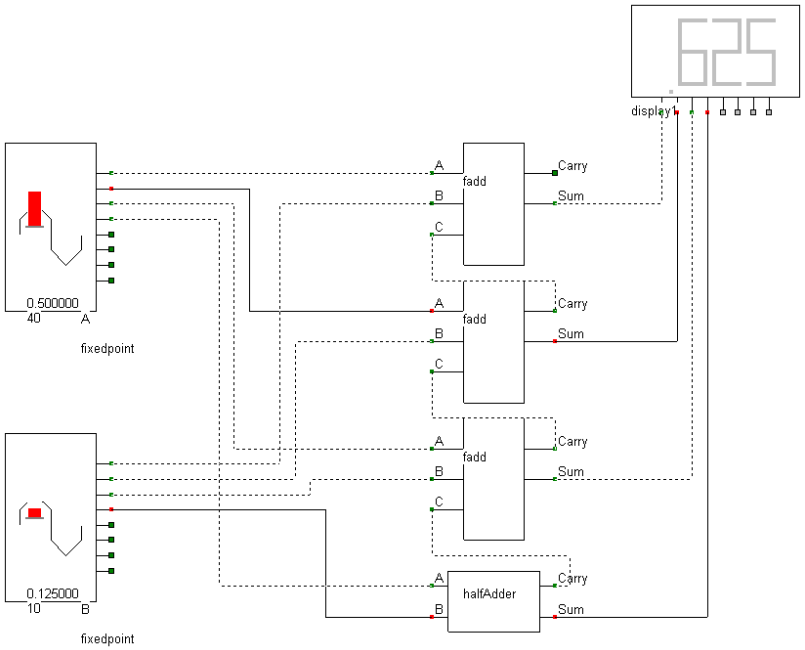

Figure 13 : Adding 0.5 and 0.125 using an adder hardware based on unsigned integers (dsp/fixedAdd4.sch)

3.2 Fixed point multiplication

The next step is now to build a multiplier, the second important circuit of the MAC mathematical block. We reuse the existing *mul44* circuit, add the "fixedpoint.SYM" symbol instead of hexadecimal keyboard, and configure the display in fixed mode. The result is shown Fog. 14. As an illustration, we configure the "A" data to 0.5 the B data to 0.25 and observe the result. Note that the sign wires of A and B are disconnected and the sign bit of the display is stuck at 0. A simple adder circuit (as for Fig.13) may be added to complete the multiplier function.

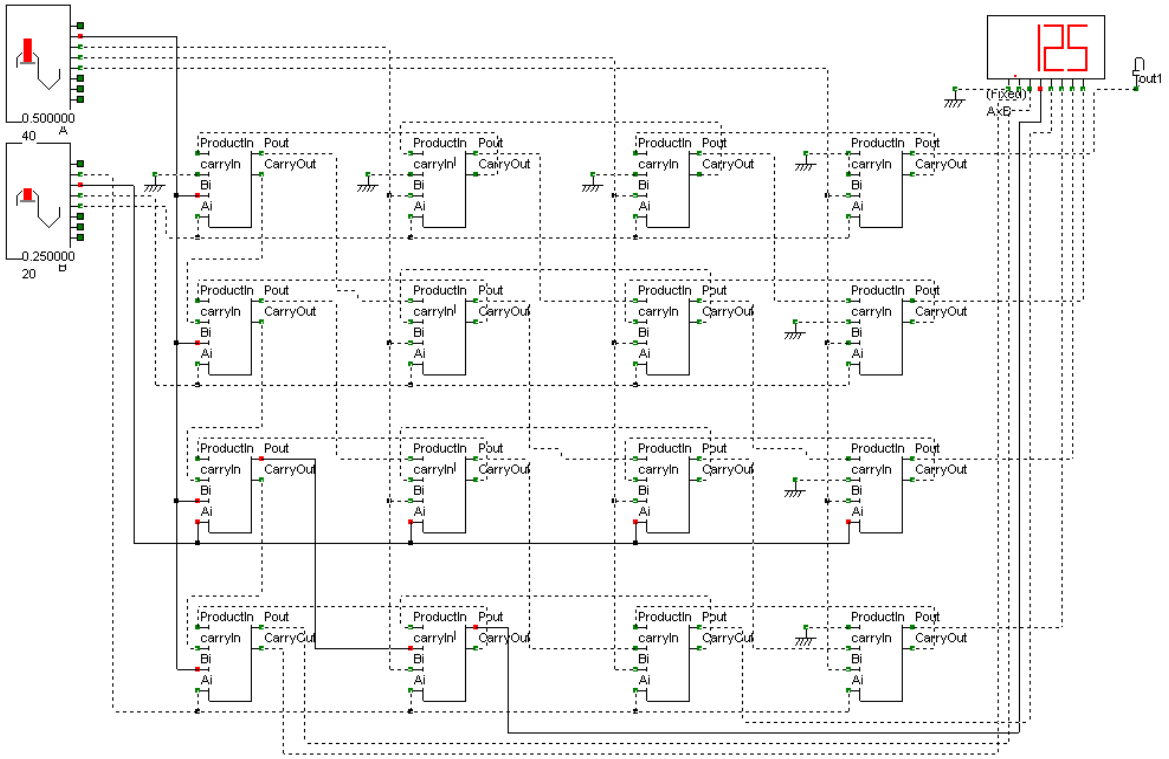

Figure 14: Multiplying 0.5 with 0.25 using an unsigned integer multiplier circuit (dsp/fixedMul44.sch.sch)

3.3 Memory

Any latch may be used to construct the "memory" function. In the implementation of Fig. 15, we simply instantiate 4 1-bit memory cells based on edge-trigged D-latches (see the chapter "Sequential circuits" of Microwind/Dsch user's manual [Microwind]). After a reset, a fall edge of the clock transfers the data to the display. Doing so, we transform x[n] data into x[n-1].

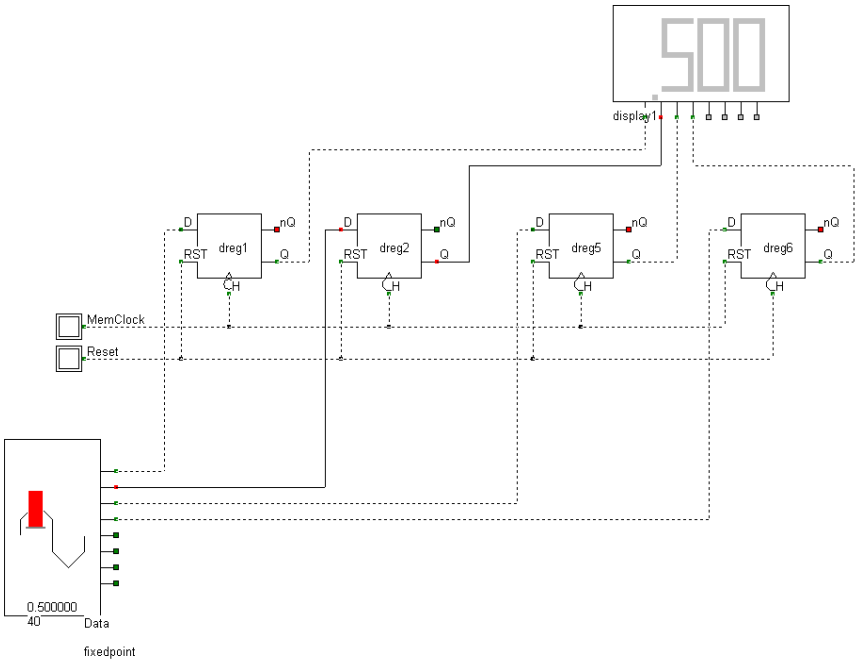

Figure 15: Accumulator based on 4 memory devices (D-reg) to store a 4-bit fixed point data on a fall edge of the clock "MemClock" (dsp/ mem4.sch)

4. References

[MentorDSP] www.etienne-sicard.fr/mentordsp

[Microwind] The user's manual and lite version are available at http://www.microwind.net/