Cost effective replacement of analog parts by DSP software

by Jean-Marie ORY

Centre de Recherche en Automatique de Nancy
ESSTIN
Université Henri Poincaré, Nancy, France

ory@esstin.uhp-nancy.fr

1 Introduction

1.1 Analog to DSP: why ?

Many industrial control applications require sophisticated signal processing. In the past, most controllers were designed using analog components. Later, low cost single chip conventional micro-controllers were used in those tasks requiring sequencing, accurate timing, man-machine interfaces etc... but still lot of analog components had to be used.
The emergence of new generations of fast, low cost Digital Signal Processors (primarily designed for the communication industry) allows reducing the analog parts to only sensors, AD converters, power outputs, and actuators.

1.2 Example: the linear inverting integrator

In order to get an opinion about how things can be done, let's look at a simple example: the linear inverting integrator (Fig. 1) .

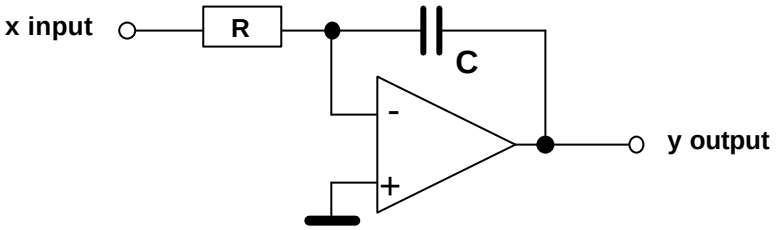


Fig. 1 The analog inverting integrator

The performed transfer function is

$$\frac{Y(s)}{X(s)} = -\frac{1}{RCs}$$

or, in the time domain:

$$y(t) = y_0 - \frac{1}{RC}\int_{t0}^{t} x(\tau)d\tau$$

If we want to translate this function into DSP software, in most cases, we would approximate the integration process by a digital accumulation to be executed at each sampling period T (Fig. 2).

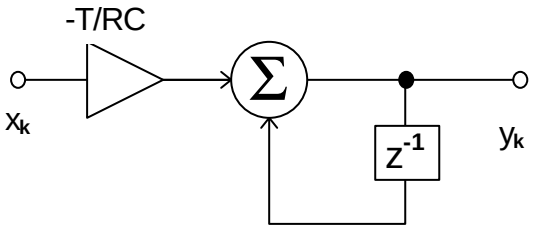


Fig. 2 Simple digital integrator

Digital inverting integrator:

$$y_k = y_{k-1} - \frac{T}{RC}x_k$$

or in z transform:

$$\frac{Y(z)}{X(z)} = \frac{-T}{RC(1-z^{-1})}$$

which can be translated in DSP56300 assembly language by following code segment:

```
integ move  yk,a
      move  xk,x0
      macri #-T/(R*C),x0,a
      move  a,yk
```

Execution time is 40ns per sample at 100MHz

Now, it is often desirable to force the initial output y_0 to a defined value. On an analog design, this will oblige the designer to add a relay in the following manner (Fig. 3):

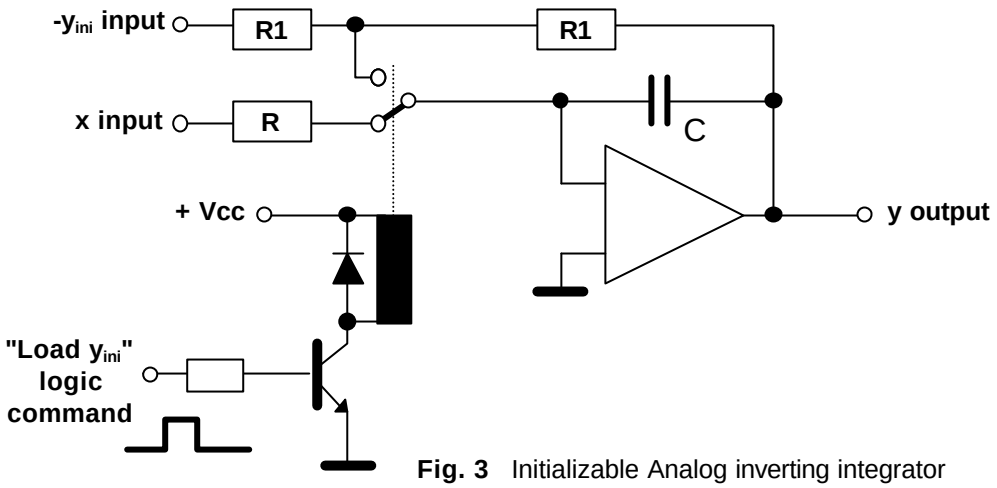


Fig. 3 Initializable Analog inverting integrator

Obviously, the digital version is straightforward: at initialization time, only 2 lines have to be added:

```
move   #yini,a
move   a,x:yk
```

Since analog components are not accurate, sometimes, the value of R would have to be made adjustable. This is not necessary in a DSP environment: calculations are made with a precision (< 10^{-6} for 24bits) not comparable with analog components precision (about 1% in best cases).
This example shows that **a digital realization of a single function will most often be much simpler than an analog version for the same function**.

Hereunder are some advantages of the DSP technology.

1.3 Modifications made easy

The most obvious advantage of software versus analog hardware is the ease of modifications. Thus, applications for which working conditions have not been well defined at the beginning of the study, will be easily maintained by applying software updates.

1.4 Better precision, better stability, no trimming

Every analog designer knows that most common analog components have precision tolerances about ±5 to ±10%. Components with tolerances <= to ±1% are rather expensive. This implies frequent adjunction of adjustment potentiometers, which causes additional cost, due to time spent for trimming. Furthermore, component values are dependant to temperature. This is dramatic when a capacitor and an inductor define the resonance frequency of a filter.
A DSP application needs 2 precise components: the clock oscillator and the AD converter.
Clock oscillators have commonly a precision and a stability over temperature of about 10-100ppm.
AD converters can be chosen with a great precision, but in most industrial applications, 12 bits will be enough. Resolution is therefore 1/4096, and the precision of the reference voltage source will be about 1%. It's drift with temperature will be about 0.001% / K.
In these conditions, calculation errors due to finite length numbers within DSPs (16, 24, 32, 48 or 56 bits) will appear negligible in most cases.

1.5 New functions

Some signal processing functions are very difficult to implement on analog hardware, whereas these are easy in software, for example:

- Pure delay lines
- Convolvers, Hilbert transformers
- Adaptive filters
- Arbitrary non-linear function generators
- Neural networks
- Fourier Transforms

1.6 Speed of development

The designer who has a DSP card available with a library of configurable bloc functions will be able to design a new product by just assembling a few software blocs, adjusting their parameters, and testing the prototype. This will take a few hours in most cases.

In the case of an analog design, the designer would have to simulate the circuit, design the printed circuit, and cable the prototype before testing it. This can take a couple of weeks.

1.6 Lower cost

In terms of production cost, an analog application will have a price roughly proportional to its complexity, whereas a DSP application will have a non negligible initial price, but this price is much less dependent on complexity (Fig. 4).

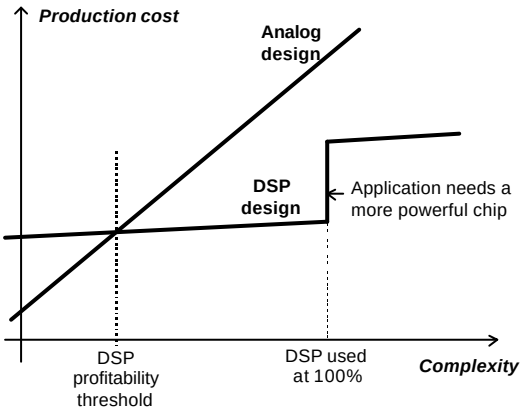


Fig. 4 Production costs: Analog and DSP

Thus there exists a threshold level of complexity over which DSP design should be used in order to minimize cost. When complexity level meets the maximum DSP performance, then, a more powerful chip or a second DSP chip has to be used, producing a step in the DSP cost curve.

1.7 Analog or DSP ? Comparing costs

Production cost of an industrial analog card:

Let's try to establish a rough measure of a printed circuit's cost, that is expected to be produced in quantities of N pieces:

Cost of design (simulation, CAD, routing) / N
+ Sum of components prices
+ (price per pin to solder) * (number of pins or holes)

According to Arnatronic, a company which designs and produces a great number of custom industrial control electronic systems, following rule of the thumb can be applied to get orders of magnitude:

Cost of a design **2000 to 5000 Euro**, (mostly depending on printed circuit surface)
Cost of production: **~ 3 Euro per analog IC** . This also includes associated passive low precision components, and their implantation.

Production cost of an industrial DSP card

Example A
Processor DSP56309 80MHz, 1Mb E²PROM, 2x ADC 12b 800Ks/s, 2x DAC 12b, 36 logic I/O, RS232, JTAG, OnCE
Production cost: **120 Euro** for 100 pieces

Example B
Processor DSP56002 40MHz, 1Mb E²PROM, 1x ADC 12b 100Ks/s + Mux 8 ch, 16 logic I/O, RS232, OnCE
Production cost: **75 Euro** for 100 pieces

Let's apply the previous rule for a given 100 pieces fabrication:
Analog version: Design cost = (3000/100) +3 x (number of IC's) Euro per card
Thus,
"Example A" DSP card (120 Euro) is profitable to replace an analog design over **30 analog ICs**,
"Example B" DSP card (75 Euro) is profitable to replace an analog design over **15 analog ICs**.

2 Converting analog designs into DSP software

2.1 From s to z

Designs with analog components are mostly used to create linear time transfer functions. These are expressed in their Laplace Transform equivalent (variable s).
In a DSP application, signals are sampled at a frequency F_s before being converted to digital. F_s should be chosen in order to avoid aliasing, which implies:

$$\mathbf{F_s > 2F_{max}}$$

where F_{max} is the highest frequency component of the signal spectrum. The DSP will have to process several streams of digital input samples, and generate several streams of digital output samples. In order that digital behavior meets the equivalent continuous time (analog) behavior, several methods are commonly used:

- Impulse invariant transform method:

The digital system must have the same impulse response as the analog model at the sampling times kT. This transform can be described by:

$$h_d(k) = h_a(kT)$$

$$\frac{1}{s+a} \rightarrow \frac{1}{1-e^{-aT}z^{-1}}$$

- The backward difference method:

Recalling that s is the derivation operator, an approximation of the signal derivative at sample k is

$$\frac{x_k - x_{k-1}}{T} \quad \text{thus:} \quad s \rightarrow \frac{1-z^{-1}}{T}$$

- The Bilinear Transform method:

$$s \rightarrow \frac{2}{T}\frac{z-1}{z+1}$$

The Bilinear Transform is usually the preferred method. Care has to be taken that the frequencies are not linearly related in the digital and the analog model. If, for instance, a resonance occurs at frequency f_a in the analog model, then, it will occur at frequency fd in the digital model with following relationship:

$$f_a = \frac{\mathrm{Tan}(\pi f_d T)}{\pi T}$$

This obliges the designer to execute an operation named "pre-warping of the frequency axis" if a given frequency response has to be respected.

2.2 Creating linear transfer functions

Two types of linear transfer functions can be easily programmed on DSPs:

Non recursive or **FIR** (Finite Impulse Response) which is a digital convolution (Fig. 5)

$$Y(z) / X(z) = \sum h_n.z^{-n}$$

or: $$y_k = \sum h_n.x_{k-n}$$

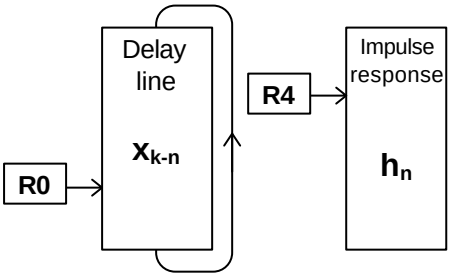


Fig. 5 FIR structure

The program uses 2 buffers, the first one as a digital delay line, the second one holding the sampled impulse response. The DSP56xxx code for the sample time execution task is given by:

```
move   x0,x:(r0)
clr    a            x:(r0)-,x0    y:(r4)+,y0
rep    #N-1
mac    x0,y0,a                    x:(r0)-,x0
y:(r0)+,x0
macr   x0,y0,a                    (r0)+
```

Recursive or **IIR** (Infinite Impulse Response)

$$Y(z) / X(z) = \sum b_m.z^{-m} / \ (1 - \sum a_n.z^{-n})$$

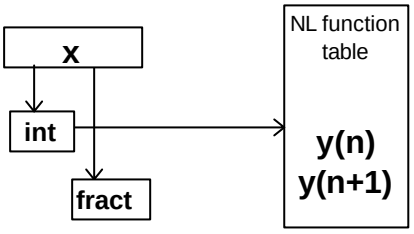
or: $$y_k = \sum a_n.y_{k-n} + \sum b_m.x_{k-m}$$

Several forms of implementation are commonly used:

- Direct form (one delay line for the xk's one for the yk's)
- Canonic form (single delay line)
- Canonic transposed (single delay line)

2.3 Nonlinear functions

In the analog world, nonlinear functions are approximated by complicated diode resistors networks which are temperature sensitive.
In the DSP world, nonlinear functions are generated by reading and interpolating within a table of constants in memory. The input signal is converted to a table entry address. The integer part indexes the table, and the fractional part allows interpolating within 2 consecutive points (Fig. 6).



y = f(x)
Table read with linear interpolation:
y = Table[int(x)]
+ fract(x). (Table[int(x)+1] - Table[int(x)])

Fig. 6 Generating nonlinear functions

2.4 Timing functions

Analog timers are done with slope generators followed by comparators. Since time constants are depending on a capacitor value, analog timers aren't accurate and are temperature dependent.
In the digital world, timing is done by counting a given number of clock cycles. Fast timings such as high frequency Pulse Width Modulation (PWM) will be done in hardware, using an onboard timer counter, whereas lower frequencies can be generated in software by counting the sampling clock cycles.

2.5 Interface and communication functions

Traditional man-machine analog interfaces are potentiometers and voltmeters.
On a DSP system, internal information is digital, thus digital interfaces are welcome. Most often a serial RS232 interface will be very useful for communicating with a computer.
Potentiometer equivalents are rotational optical encoders. Since these components are still expensive, a couple of increment / decrement buttons are often preferred. Potentiometers are replaced by digital displays.
In systems which have very complicated front panels including a great number of LED indicators, displays and push-buttons, a method of minimizing wiring consists of using a small local micro-controller which communicates via a serial line with the DSP. This communication can be done in background, using a DMA channel of the DSP. Therefore the DSP software will just have to exploit the I/O data image which is mapped in it's own memory.

3 Designing DSP software

3.1 Software architecture of a general real time control application

A general industrial real time control application can be split in 4 functional groups:

- Initializations and system management tasks:

The purpose of this module is to configure the machine hardware at startup, load programs and data in internal memory, initialize variables and launch the main loop execution.

- Timing and sequencing tasks:

Combination of hardware timers (generating interrupts) and software timers usually based on the sampling frequency. The aim of a software timer is to set a flag at timeout. This flag is tested and cleared by low priority tasks waiting for execution.

- Tasks to be executed at the sampling frequency

These tasks actually do the real time signal processing job. Each task can be considered as an independent bloc taking data samples present on the inputs and processing them to new data transferred to outputs. Thus each bloc simulates an analog signal processing circuit with a given transfer function. Blocs are connected between each other by virtual wires which consist in transferring data from bloc outputs to inputs of other blocs. Most often, AD converters constitute the initial inputs and DA converters the final outputs.

- Asynchronous I/O and communication tasks

What characterizes asynchronous I/O and communications is the no relationship between their occurrence times and other timings available on the machine. Thus producer-consumer protocols with FIFO management have to be installed in order to manage different data rates.

3.2 Writing modules in C language

C language programming is the most common and fastest way for generating DSP applications. However, C compilers generate non optimal code.
On DSP56xxx, reasons for this are:

- Parallel memory fields not supported.
- Fractional saturation arithmetic not supported .
- Modulo addressing not supported.

Thus time sensitive routines should be written in assembly language and embedded within the C program.

3.3 Writing modules in assembly language

Writing modules in assembly language requires a good knowledge of the machine's architecture and instructions. Long programs must absolutely be fractionated into several short modules in order to be understandable. Fortunately, the Motorola DSP assembler includes several features that permit efficient programming in assembly language, in particular:

- Macros with arguments
- Conditional assembly
- Mathematical functions
- Relocatable sections with attributes

3.4 Using macros

Macros allow to generate a sequence of instructions or directives with a single identifier. Using arguments allow variations in the generated code. The assembler must have read the macro definition prior to using the new identifier. The macro is parsed by a preprocessor before being expanded. Following example shows some features found on macros and conditional assembly:

Example:
Vector multiplication

```
vmul          macro         size,vector1,vector2,vector3

       if            (@cnt()>4)|(@cnt()<3)                          ; if syntax not correct,
       fail   'Syntax: vmul size,vector1,vector2[,vector3]'         ; generate error
       else
              if     @cnt()=3                                  ; if 3 arguments, destination is vector 3
              vmul   size,vector1,vector2,vector2 ; self nested macro call
              else
              section exec                                     ; Section to be executed at sampling time
              move   #vector1,r0                               ; initialize pointers ...
              move   #vector2,r4
              move   #vector3,r5
              move   #size-1,x1
              jsr    vecmul                                    ; call function
              endsec
if     !@def(vecmul)                                  ; check if code has already been implemented
                     section       routines          ; (do not generate code, if it already exists)
vecmul               move   x:(r0)+,x0    y:(r4)+,y0
                     do     x1,vecmul1
                     mpy    x0,y0,a       x:(r0)+,x0    y:(r4)+,y0
                     move   a,y:(r5)+
vecmul1
                     rts
                     endsec
                     endif
              endif
       endif
```

3.5 Creating a library of software modules

For a company which designs many applications of the same type (example: industrial control), it will be valuable to create a software library of ready-to-use modules. These modules have to be written as ASM macros in a way that makes their usage transparent to non specialists.

In following example (Fig. 7), a DSP is initialized to trigger the AD converters at 10KHz; AD1 input is applied to a first order low-pass filter, and the result applied to a 100 sample delay line and to DA1 output; The delay line output is applied to DA2 output.

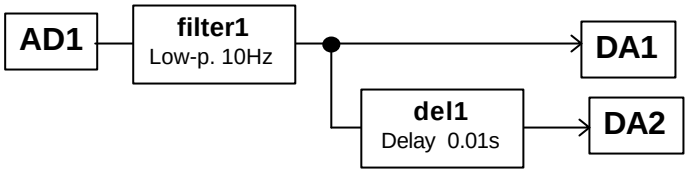


Fig. 7 Bloc programming example

This Real Time application is split into two parts: the **Data Flow** which describes the signal paths and the **Program Flow** which gives the processing order:

```
; Demo
; DATA FLOW:
; Define connections (say which variables are common)
        cn      ad1,filter1_in              ; transmit information
        cn      filter1,da1                 ; between blocs
        cn      filter1,del1_in             ; ...
        cn      del1,da2                    ; ...

; PROGRAM FLOW
loop    ada     10000.                      ; wait for AD conversion
        lp1     filter1,10.0,abs            ; Execute low-pass at 10Hz
        delay   del1,100                    ; Execute 100 sample delay
        goto    loop                        ; Wait for next sample
```

This code is all that the user has to write. Assembly language syntax is respected, but each mnemonic is actually a macro. Each macro contains several sections. These describe code implementation, initialization and execution. These sections are memory mapped during the linking process.
The assembly / linking batch file adds all required system initialization files at compilation, therefore the generated machine code is directly ROMable.

4 Bloc components

4.1 Identification

Each bloc instance must have a unique name (ASCII string). This name is concatenated to every internal block variable name, thus generating unique identifiers.

4.2 Processing code

The processing code is an executable machine code segment which performs the desired signal processing function between inputs and outputs. It is often a subroutine which is called from the execution entry. Several instances from a same bloc type with different parameters will generate only one routine segment.

4.3 Initialization code

The initialization code is a machine code segment which is executed during system startup, or eventually on demand. If no INIT segment has been defined in the bloc, the system will clear it's state variables to zero at startup.

4.4 Real time segment

This segment executes at sampling times. It is a code segment which usually initializes pointers to variables and data, and launches the processing code routine with adequate parameters.

4.5 Variables

Each bloc instance has variables which actually carry the information to process. Inputs and outputs are considered as variables. On DSP56xxx variables are generally implemented in the X: field.

4.6 Data

Most blocs require constant values such as gains, filter parameters ... Since these data are often associated to variables within 2-operands instructions, on a DSP56x, they will be located in the Y: field. This allows the use of efficient parallel move instructions.
Some data can be considered as variables, for instance coefficients of an adaptive filter, or neural network weights.

4.7 Signal properties

Each physical problem handles data representing real measurements with associated units. A DSP bloc processes one or more input data streams and generates one or more output streams. Each data stream has an associated unit and scaling factor. These values (compilation time variables) are passed to bloc inputs by the Connect (CN) macros. Depending on the realized function, new values are available on the bloc's outputs, which will propagate further, and so on. A units / scaling factors report is generated at the end of compilation which will help to point out some design errors .

4.8 Execution time

When the DSP application has to simulate an analog diagram, each bloc will have to execute once at each sampling period. Thus, the sum of all bloc execution times is calculated and compared to the sampling period. Execution times can be calculated during the compilation process, using the "cumulative cycle count" option. At the end of compilation, an error message will signal if total blocs execution time is longer than 90% of the sampling period .

4.9 Resources

Each bloc occupies some room in memory. Some blocs use part of the DSP chip's hardware. At end of compilation, an error message report signals to the programmer if more memory than available has been reserved, or if some exclusive hardware resources have been allocated more than once.

5 Application example

5.1 Bloc diagram of a TIG (Tungsten Inert Gas) welding machine

This example shows a typical use of a DSP in order to reduce components count and cost. The original analog version of this TIG welding machine was extremely complex, thus hardly valuable.
The use of a small DSP card has allowed to reduce dramatically the components count, and even to add some functionality not available on the analog version.

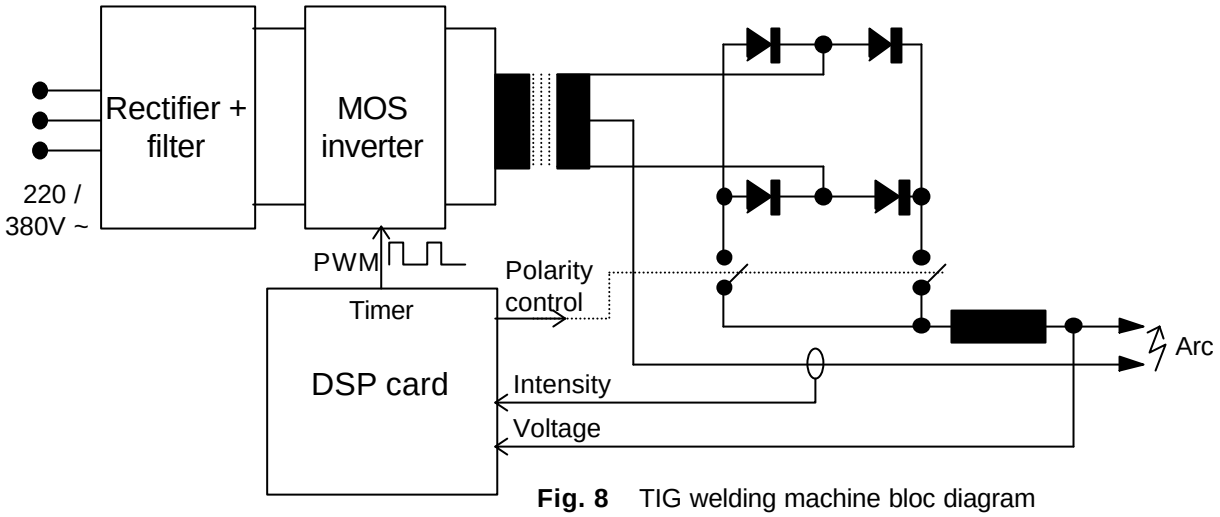


Fig. 8 TIG welding machine bloc diagram

Here, (Fig. 8) the job of the DSP is to generate a 80KHz Pulse Width Modulated (PWM) rectangular waveform which is used to drive the MOS inverter. The aim is to get an arc with a controlled variable current. This current can be direct or alternative with an arbitrary periodic waveform. For an alternative output, a control on IGBT switches will decide which half rectifier bridge will be used. Voltage measurement just serves for getting the arc's dynamic impedance.

5.2 Arbitrary waveform generator

One aspect which was very complicated (thus not implemented) in the analog version was the arbitrary waveform generator which is the reference control signal for output current (Fig. 9).

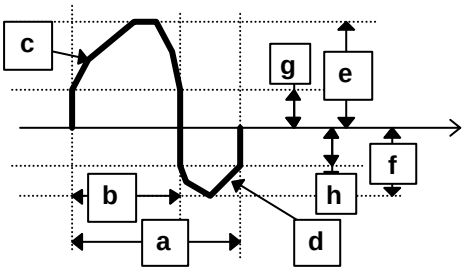


Fig. 9 Arbitrary waveform

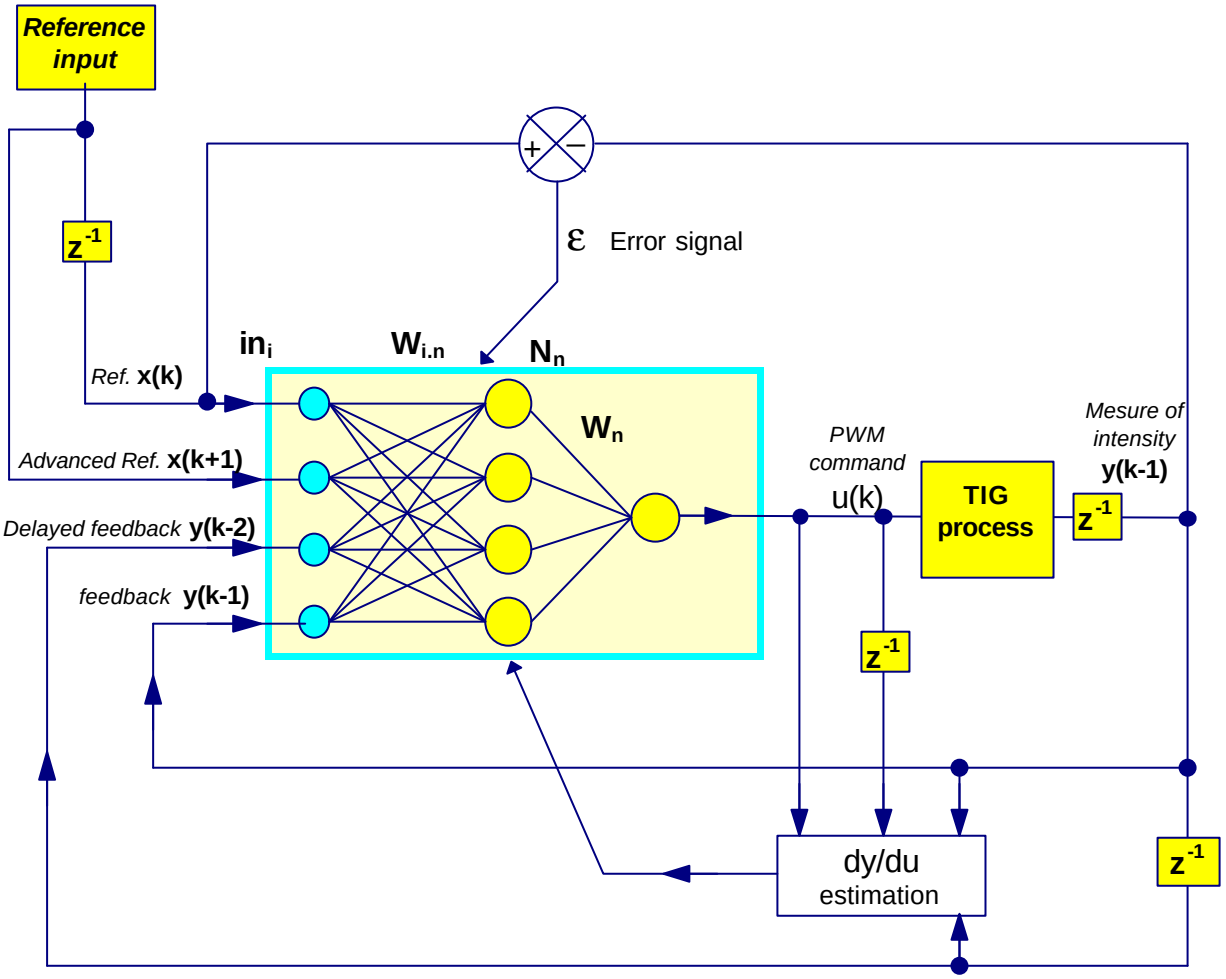
a Wave frequency (50 .. 250Hz)
b Duty cycle (0..100 %)
c Positive waveform (4 wave tables in memory)
d Negative waveform (4 wave tables in memory)
e Positive amplitude
f Negative amplitude
g Positive pedestal
h Negative pedestal

From data (a) and (b), the DSP will compute the phase increments per sample for the positive and the negative waves. Integrating these values modulo N will generate addresses within N point waveform tables. Interpolation between points will give an accurate result. This result is then multiplied by (e)-(g) and added to (g) in order to get the required positive amplitude and pedestal. When passing from positive to negative half wave, a new set of parameters is loaded.

5.3 Neural control module

The waveform described in 5.2 serves as an arc current reference. The description of the system function between the PWM input and the arc current output is not straightforward. Variable time constants are produced by the saturable inductor associated to the variable arc dynamic resistor. Even in DC mode, the system command is highly non linear. On the analog version (constant DC input reference), a nonlinear PID regulator was used.

In order to adapt dynamically the controller to the best behavior in approaching the desired output waveform, we used a small neural network with online training as described in Fig 10 :



Neural network control algorithm:

$N_n = \text{Tanh}\,[\alpha * (w_{0,n} + \sum in_i * w_{i,n})]$ $\quad i = 1..4$

$out = \beta * (w_0 + \sum N_n * w_n)$ $\quad n = 1..5$

Training algorithm:

$W_n(k+1) = W_n(k) + a1 \,.\, \varepsilon \,.\, N_n \,.\, dy/du$

$W_{i,n}(k+1) = W_{i,n}(k) + a2 \,.\, \varepsilon \,.\, in_i \,.\, W_n \,.\, g(N_n) \,.\, dy/du$

Fig. 10 On line training neural network for controlling the arc current

Experimentation has shown that the dy/du term never changes in sign. Therefore, this term can be taken equal to 1 in the training algorithm. In these conditions, implemented on DSP56309 at 64MHz, the controller takes less than 8µs per sample for both executing and training. Since one decision is taken every two PWM pulses, the actual sampling period is 25µs, which leaves a comfortable 17µs per sample window for executing all other tasks.

6 Conclusions

Some companies are still designing electronic control applications using operational amplifiers, precision resistors and capacitors. Reasons invoked are mainly the cost in terms of development time and DSP components price. In this paper, we have tried to show that above a determined level of complexity, DSP solutions are cheaper than analog ones, and bring many other advantages (new functions, precision, easy product updating). By investing in the design of a set of most used macro modules library, software development becomes fast and easy.

Our experience of designing a DSP controlled TIG welding machine shows that in practice, a great number of complicated tasks (such as neural network control) can be implemented in real time at high sampling rates, on a single small and cheap DSP chip.