User's manual of the GOLIATH program

1. Introduction

The DAVID board and the GOLIATH program have been developed at the ***Industrial Electronics Laboratory (LEI)*** of the *Swiss Federal Institute of Technology* in Lausanne (EPFL). The development and commercialisation of these products are now supported by *CHS*, a spin-off company of the LEI lab.

GOLIATH is the supervisor program that allows an easy control of the DAVID board, based on the *SHARC* digital signal processor. The program communicates with the DAVID board through the RS-232 serial link of the computer ; no special interface board is required. Therefore, the program works on any PC operating under (Windows 95 / Windows NT).

The computer requirements are :

Operating system :	Windows 95 / Windows NT
Processor :	486 / 66 MHz and faster
Disk space :	10 MB
RAM memory :	16 MB minimum

The program uses a compiler from Analog Devices to create the executable file. The user has to buy this compiler separately (about 800.- SFR).

2. Application example

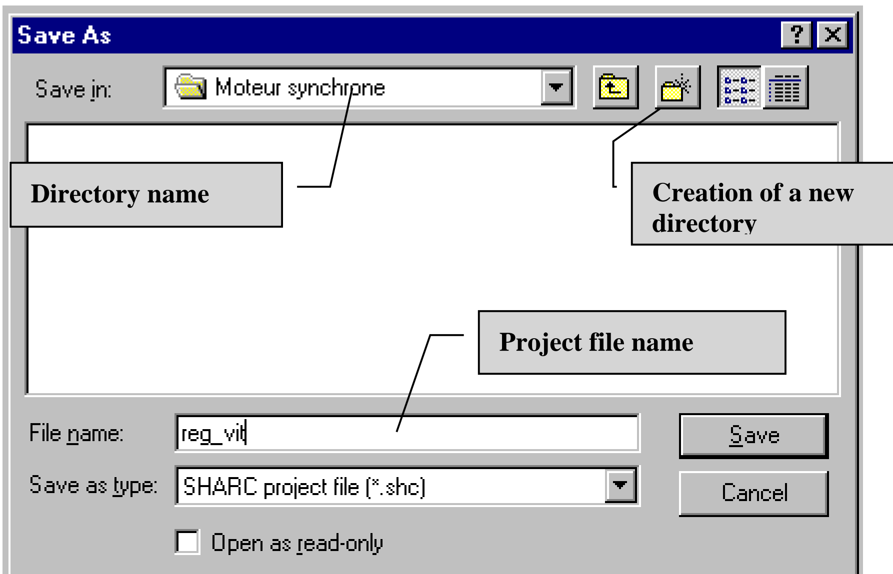
As a tutorial example, we consider the control of a small 3 phase brushless DC motor.

2.1. Creation of the project

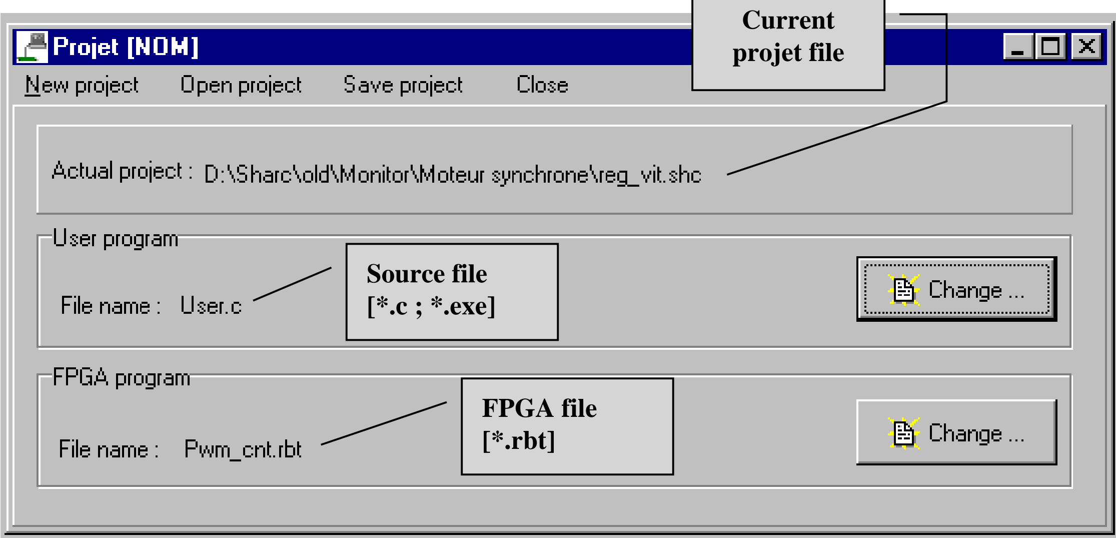
The program controls the user's configuration using a project file. In particular, the project file include the following information :

- Application filename (source code in C / SHARC executable file)
- FPGA configuration filename
- Options of the Analog Devices compiler.

The access to the project manager is through the icon of the toolbar, or from the menu Project➔Open. Choose [New project], create the directory [Moteur synchrone] and choose [reg_vit] as the name of the project file.
NOTE : for compatibility problems with the Analog Devices compiler, the name of the project file, respectively the name of the user's file must use the DOS convention (8 characters for the name + 3 characters for the extension).



The new window in the project manager looks then like this :



The file in the section ***[User program]*** is the source file of the application. There are two options :

- The user's file is written in the standard C language (*.c). In this case, GOLIATH carries out the compilation of the source code and the link editing as well as the creation of the downloadable file and the variable file.
- The user's file is a SHARC executable file. In this case, GOLIATH simply create the downloadable file and the variable file.

The file in the section ***[FPGA program]*** contains the information for the configuration of the programmable logic circuit. Normally, the system is delivered with the standard configuration [PWM_CNT], which realises a 3 phase PWM modulator with dead time control as well as an interface for an incremental encoder with 2 signals in quadrature.
The user's file, respectively the FPGA file associated to the project can be changed with the two buttons [Change …].

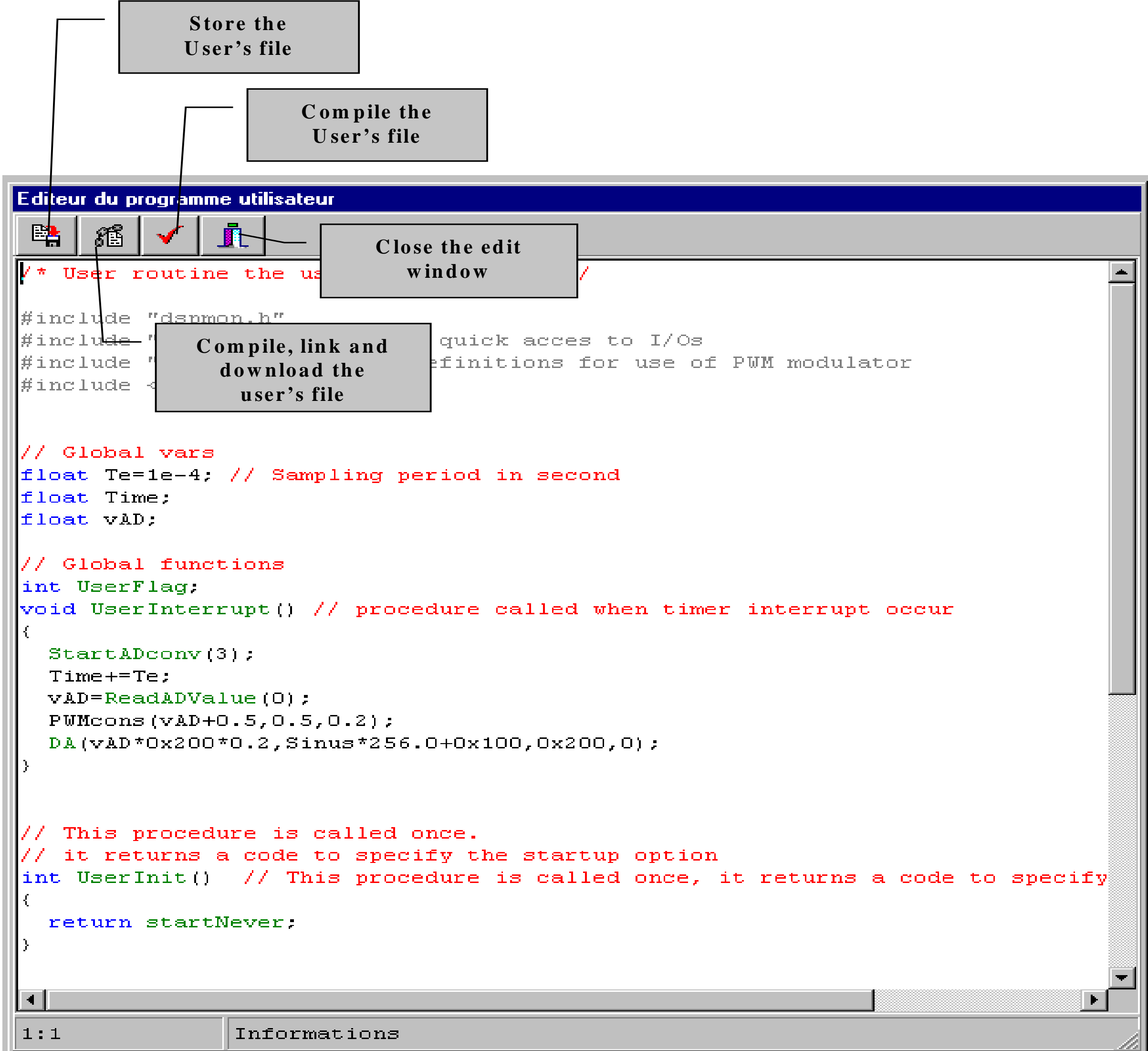
2.2. FPGA code

As soon as the user has chosen a project, several menu choices appear in the menu [Project]. In particular, the last choice is the menu ***[Download FPGA]***.

With the sub-menu *[Download FPGA → direct]* we realise the direct programming of the FPGA circuit; the programmable logic circuit implements the new functionality since the acknowledgement of the last line of the RBT file.
With the sub-menu *[Download FPGA → FLASH]* we do not realise the direct programming of the FPGA circuit, but the program is stored in the FLASH memory. The user can then program the FPGA from the FLASH by typing the command line [xilinx eeprom] in the user's command line. The storage of the FPGA program in the FLASH memory allows the FPGA configuration to be stored on the DAVID board. The user's program can then download the FPGA configuration at the start of the user's program when working without supervisor program.

2.3. Editing the user's code

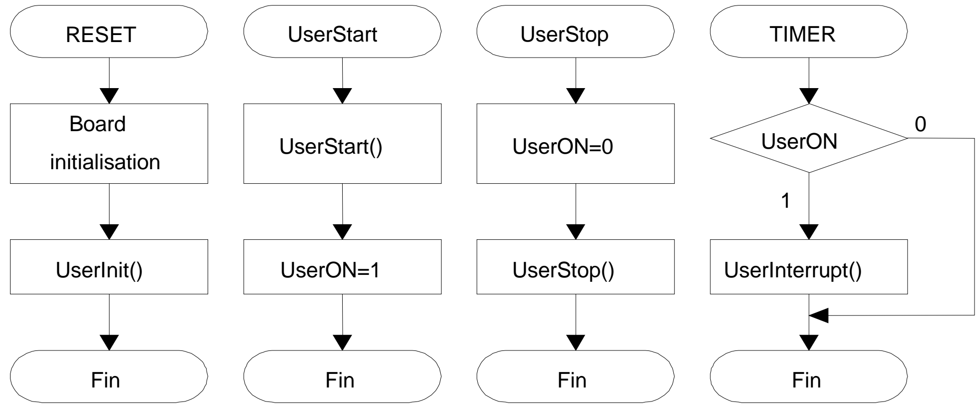
When the user generates a new project, GOLIATH creates the file *[USER.C]*, containing the minimal application code. To edit the file, just choose the menu *[Project → Editor]* or click on the icon .



The user's code is controlled through the 5 following functions :

- UserInit()
- UserStart()
- UserInterrupt()
- UserStop()

- UserIdle()



The user can also add personal functions in order to have a good program structure.

2.3.1. Global declarations

The user's file starts by indicating the following include files

- dspmon.h basic program for the whole DSP control (including the control of the user's program)
- io.h macros to access the DSP board peripherals
- math.h mathematical library.

In the example project we also use the macros (MIN, MAX) that are predefined in the file [macros.h]. We have then to add the definition : `#include "macros.h"` .

The next step is to define all the global variables. For our application example, we define the following variables:

```
float Te=1e-4; // Sampling period in second
float Ucma,Ucmb;
float PU,PV,PW;
float Ucmu,Ucmv,Ucmw;
float Theta,Theta0,ThetaOld,Speed,SpeedRef,ThetaU;
float Amplitude;
float IU,IV,IW,Ia,Ib,Id,Iq;
float Ki,Kpi,Erreur,Xr,U, Up;
char MyMessage[128];
float Time, MyTime;
int Pos;
float DPos,fPos;
int Phase;
int TotalCnt;
```

2.3.2. Initialisation at the power on → UserInit()

The function UserInit() is called after the board initialisation phase (i.e. after a DSP RESET or a power on). This function must return to the DSP a code determining the working mode at the power on or after a RESET. There are three starting modes:

- startNever ➔ The user's program is started after a specific command from the host (PC). This avoid an anomalous starting of the control procedure for the applications under development.
- startAlways ➔ The user's program starts spontaneously, no matter if the host is present or not. This allows DAVID to work in stand-alone.
- startIfAlone ➔ The user's program is started if no host is present at the power on. The monitor program must be present and active.

On the previous flowcharts we see that the function UserInit() is called at the DSP power on (i.e. following a board reset). On the other hand, the function UserStart() is called at every starting of the user's program. Due to this difference, the variables initialised in the section UserInit() can be modified through the program

GOLIATH and are not reinitialised at every starting of the user's program. For example, the variables of a PI controller can be initialised in the zone UserInit() and modified online with the program GOLIATH. This way, when the user's process is started, the controller variables are already initialised at their correct values.
The variables initialised in the function UserStart() are initialised at every starting of the control program (typically, state vector variables, time integrals, etc).
The next operation is to program the FPGA with the code stored in the EEPROM memory.
The code of the function UserInit is listed here:

```
int UserInit()
{
  Amplitude = 0.2;
  PWMmode(Inverse);
  DPos = 0.001;
  Kpi=0.1;
  Ki=0.05;
  XILINload();
  return startNever;
}
```

2.3.3. Starting of the user's program → UserStart()

At the previous section, we have shown the differences between the functions UserInit() and UserStart(). Along with the process variables we also have to initialise the three phase PWM modulator with the following parameters:

- Switching frequency : 20 kHz
- Dead time : 1 µs
- Output polarity : negative (the transistor is on when the output signal is low)

The last function sets the sampling time Te (100 µs in our example)

```
void UserStart() // User start to run after this procedure
{
  Time=0;
  MyTime=0;
  TotalCnt=0;
  ThetaU=0;
  Phase=0;
  Xr=0;
  PWMinit(20000,1e-6,Inverse);
  SetSamplingTime(Te);
}
```

2.3.4. Interruption procedure → UserInterrupt()

This function is the heart of your control procedure. GOLIATH calls the interruption procedure at every sampling period (according to the value specified in the function UserStart()).
Usually, the digital control of a system is implemented in the following way:

```
void UserInterrupt() // procedure called at every timer interrupt
{
  StartADconv(0xff);
```

Start the A/D conversion(s). The parameter specifies the converters to start..
Example: 0x01 → A/D 1; 0x05 → A/D 1 et A/D 5; 0xff → all the A/D

```
Time+=Te;
```
Integral of time

```
// compute position
  Pos=ReadCounter();
  if(Pos>=0x8000) Pos = Pos-0x10000;
```
counter overflow
```
  fPos=Pos;
```

```
  Theta = ((float)Pos * 3.1416 / 1000.0)-Theta0 ;
```
Compute the new angle Theta (used to generate the sinusoidal voltages)
```
  if (Theta-ThetaOld>1) Speed = (Theta-ThetaOld-6.2832)/Te;
  else if (Theta-ThetaOld<-1) Speed = (Theta-ThetaOld+6.2832)/Te;
  else Speed = (Theta-ThetaOld)/Te;
```
The speed is computed keeping into account the periodicity of 2π, respectively the rotation direction of the motor.

```
  ThetaOld=Theta;

// read AD values
  IU=ReadADValue(0);
  IV=ReadADValue(1);
  IW=ReadADValue(2);

// Compute Id, Iq
  Ia=IU;
  Ib=(IV-IW)/sqrt(3);
```
It is possible to optimise the computing time by substituting the last instruction by:
Ib = (IV-IW)*INVSQRT3; INVSQRT3 is a variable computed in the function UserInit(); INVSQRT3 = 1/sqrt(3);

```
  Id=cos(Theta)*Ia+sin(Theta)*Ib;
  Iq=cos(Theta)*Ib-sin(Theta)*Ia;
```
Compute the current values in the Park representation

```
// Control of Iq
  Erreur=(SpeedRef-Speed);
  U=Kpi*Erreur+Xr;
  if (U>0.45) U=0.45;        Limitation of the integral term
  if (U<-0.45) U=-0.45;
  Xr+=Ki*(Erreur)-(Kpi*Erreur+Xr-U);

// compute orientation
  switch (Phase) {
  case 0 :
    ThetaU=ThetaU+DPos;
    if(ThetaU<21)   { Look for the index of the incremental encoder
      PU = cosf(ThetaU);
      PV = cosf(ThetaU+2.09);
    } else {              We assign a fixed three phase voltage system (Theta = 0)
      Phase=1; // end of initialisation
      Time=0;
      PU = cosf(0);
      PV = cosf(2.09);
    }
    break;
  case 1 :
    if(Time>=1) {
```
The reading of the sensor allows Theta0 to be computed. Theta0 is the steady state angle resulting from the application of a fixed three phase voltage system (Theta = 0)
```
      Theta0=((float)Pos * 3.1416 / 1000.0);
      Phase=2;
}
    break;
  case 2 :
    ThetaU=Theta+3.14/2.0;
    if(ThetaU>6.2832) ThetaU-=6.2832;
    PU = cosf(ThetaU);
    PV = cosf(ThetaU+2.09);
    Amplitude=U;
```
The amplitude is the voltage computed by the PI controller as a function of the speed error.

```
      break;
    }

  PW = -PU-PV;  Compute the third phase
// compute references
  Ucmu = MIN(Amplitude*PU+0.5,0.95);
  Ucmv = MIN(Amplitude*PV+0.5,0.95);
  Ucmw = MIN(Amplitude*PW+0.5,0.95);

// output references
  PWMcons(Ucmu,Ucmv,Ucmw);    Output the PWM references to the FPGA

// visualisation of the variables
  Ucma=Ucmu;
  Ucmb=MIN(Amplitude*sin(Theta+90)+0.5,0.95);


// output to DAC
  DA(Ia*0x200*0.2+0x200,Ib*0x200*0.2+0x200,
     ThetaU*0x80+0x80,Theta*0x80+0x200);
}
```

2.3.5. Stop the user's program → UserStop()

When the user's process is stopped (USERON == 0), GOLIATH calls the function UserStop() which causes the transistor command pulses to be stopped (PWMstop()). If the user allocates dynamic variables in the function UserStart(), he must dispose the allocated memory in the function UserStop().

```
void UserStop() // Stop user, used to shut down critical I/O
{
  PWMstop();
}
```

2.3.6. Background task → UserIdle()

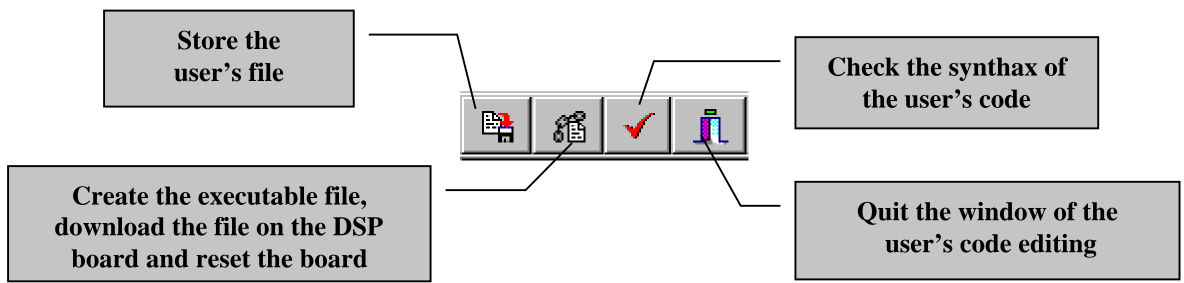
The last function available to the user is called as a background task (i.e. when the processor is not used). For example, this function can be used to send messages to the user, to execute auxiliary computations, etc.
Warning: it is mandatory that this function does not monopolise the processor with an active waiting.

```
void UserIdle()              // Background process.
{                            // the function must exit after execution
  if ((Time - MyTime) > 2) {
    MyTime = Time;
    msgSendUserInt("ToCo",TotalCnt++);
  }
}
```

The function ***msgSendUserInt(Ident, Valeur)*** allows the programmer to communicate with a custom application. In this case, the processor send an integer value identified by the four characters [ToCo].

2.4. Compilation, linking and downloading

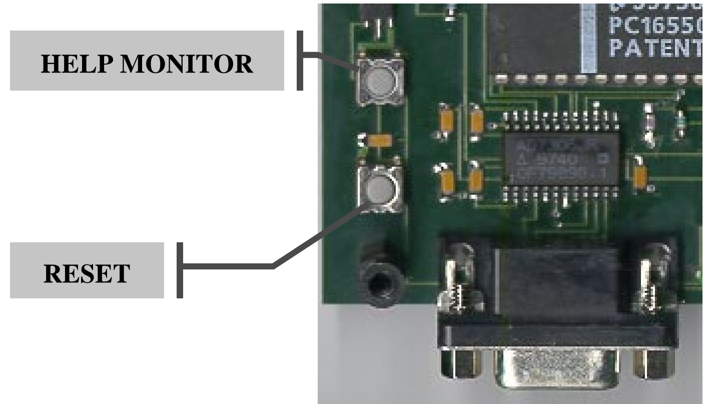
Once the control program code is finished, the code has to be compiled (creation of the objet code), linked (creation of the executable file) and downloaded on the DSP board.
On the window for the user's code editing, we find the four following icons:



The downloading of the code on the DSP board can also be done from the menu ***[Project → Download program …]***. Note that if the user's file is a SHARC executable file, the editing window does not become visible; the only way to download the code is from the menu ***[Project → Download program …]***.

2.5. Control of the user's program

Once the executable code is downloaded on the EEPROM memory, a RESET must be done on the board in order for the DSP to start with the newly downloaded code. The RESET can be executed by pressing on the board RESET button or through the GOLIATH program, by typing the word [reset] on the command line.



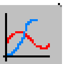
If the user's program performs an illegal operation, the DSP may crash. As a consequence, the DSP control program can no more communicate with the GOLIATH program and the user cannot download any program. To recover from this situation, the board is delivered with a standard DSP control software that can be activated by holding the button [HELP MONITOR] pressed while giving a pulse on the RESET button. The first LED remains ON to indicate that you are running on the rescue monitor.

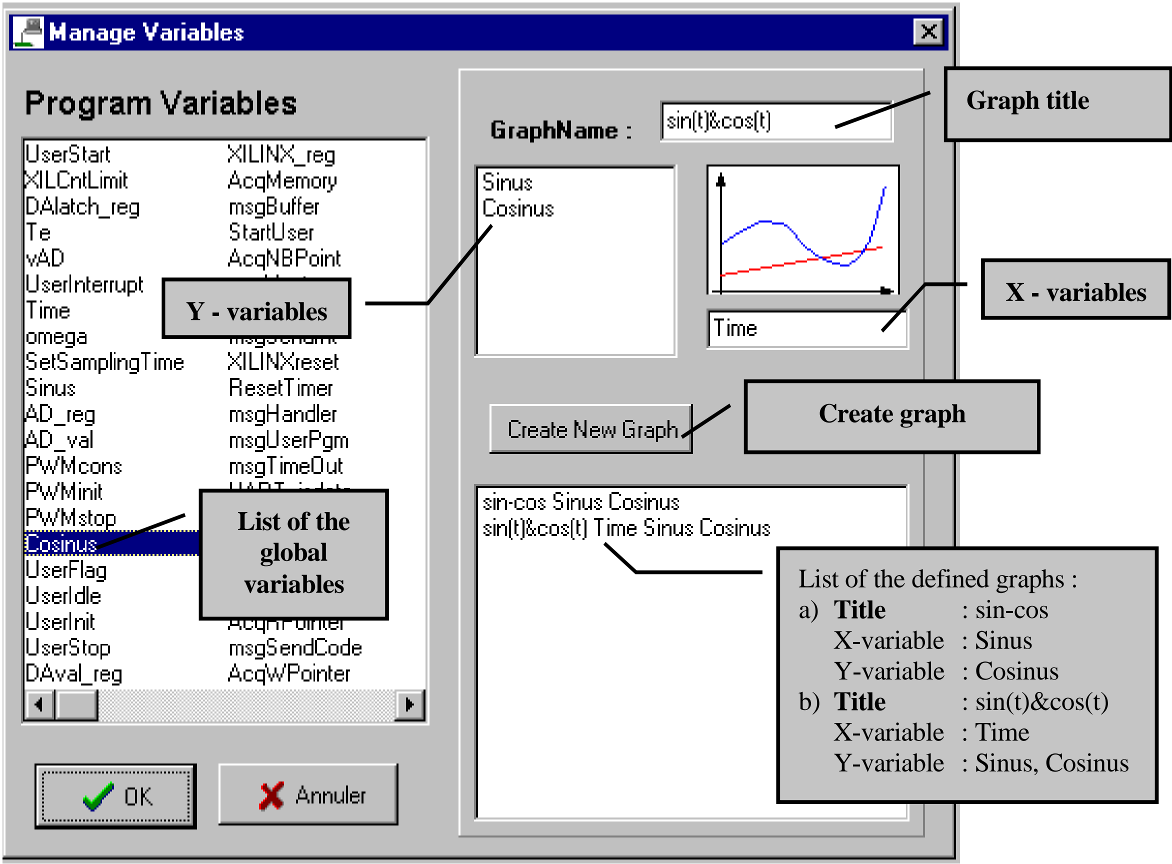
The user can easily control his program with the two buttons . The second button realises the operation USERON=1, which starts the user's process. The first button realises the inverse operation, i.e. USERON=0 and the user's process stops.
These two operations can also be done from the command line:

- [user stop] : stops the user's process
- [user start] : starts the user's process

2.6. Acquisition control

To monitor the process, the user can visualise the global variables of his application on the PC screen. This operation is completely transparent to the user as no additional code has to be written; the GOLIATH supervisor program does all what is required for a non intrusive acquisition.

The user accesses the graphical manager through the menu ***[Project → Graphics]*** or with the button .



On the left we find the complete list of the global variables. At present, this list includes the user's variables and the variables of the DSP control program. The user selects the variables in the left list and using the Windows properties ***slide / move***, he can fill the field X-variable and the list of Y-variables. He then specifies the graph title and he create it by clicking on the button ***[Create New Graph]***.
The bottom left list shows the graphs created by the user. A graph can be easily cleared by selecting it on this list and clicking on the button [DELETE]; the program still asks to confirm the operation.
The program stores the graphical configuration in the file [Graph.cfg] in the same directory than the actual project file. This way the user has not to define his graphs each time he enters the supervisor program.
The confirmation of the graphical configuration is carried out when the user click on the ***[OK]*** button. This operation is very important because the supervisor program determines the data vector to be transmitted and send this information to the DSP.

2.7. Acquisition modes

Once the graphical manager is quitted, the main window of the supervisor program contains a child window for each of the graphs defined. In any graphic window we have two buttons :

- AUTOSCALE → the scale is chosen automatically depending on the limits of the data.
- UPDATE → the scale is assigned according to the values of the fields Xmin, Xmax, Ymin, Ymax.

The windows can be manipulated according to the usual Windows rules ; the menu [Window] allows the control of the child windows (graphs), of the list of the DSP answers as well as of the list of the keywords reserved for the command line.
To start the acquisition, the user has to start the process and to select one of the three following acquisition modes :

- ***continue acq.***: The DSP checks the RS-232 interface at every sampling time. If the interface is free, the DSP send the "actual" data vector, otherwise the latter is lost.
- ***single acq.***: The DSP stores all the data specified by the acquisition vector at every sampling time until its RAM memory (2 or 4 MB, depending on the model) is completely filled. Once the memory is filled, the DSP send the data to the supervisor program which deals with the visualisation task.

- ***trigger acq.***: similar to the ***[single acq.]*** mode. The acquisition start is triggered when the trigger condition is filled. At present, the trigger condition can be defined from the following command line : ***trigger [+/-] [Variable Name] [trigger level]***. For example, to store the data starting when the ***Sinus*** variable is rising and crosses the value 0.56, the following command line must be typed: ***trigger + Sinus 0.56***.

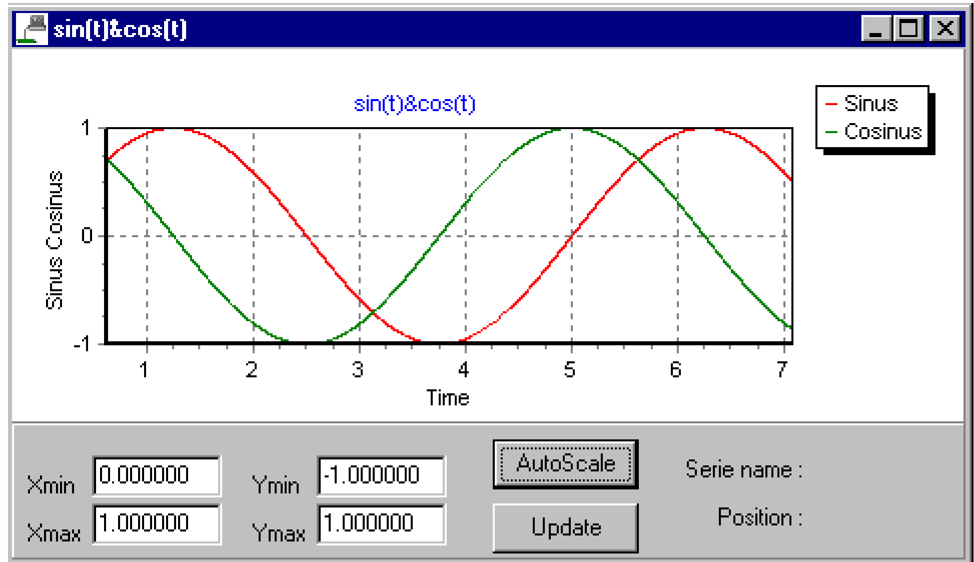


Figure 1 : Graphical window

The user can zoom a particular area of the graph in the following way :

- place the cursor at the left top of the area to zoom
- click and hold the left mouse button
- move on the right bottom of the area to zoom
- release the left mouse button

The program zooms then the selected area.
To come back to the normal dimensions :

- from the right to the left, select any area of the graph
- click on the button **[AutoScale]**

The user can also move the graph by just clicking and holding the right mouse button, moving the graph and finally releasing the button.

The graphical window has also a dedicated menu which appears when the right mouse button is clicked. This menu allows the graph to be printed, his parameters to be changed and also, the graph to be exported in the desired format (Bitmap, Metafile or Enhanced metafile).

3. Addendum

*3.1. Use a *.exe file*

In special cases the user may want to directly download a SHARC executable file, created by the Analog Devices compiler. In this case, the supervisor program still needs the project file to control the environment. The user must simply check that the executable file is located in the same directory than the project file. Also, he must select the executable file as ***[User Program]*** in the project manager.

3.2. The user's procedures

The user can create his program by subdividing it in as many functions as desired. For example, if the user wants to control two processes with two different sampling times, he can write the program as described in the code below. For this example, we consider the sampling times :

- Process A : $T_{eA} = 100\mu s$
- Process B : $T_{eB} = 500\mu s$

```
float Te;
int Controller;

void UserStart()
{
  Te = 1e-4;
  Controller = 0;
  // user code
  SetSamplingTime(Te);
}

void ControlProcessA();
{
  // user code
}

void ControlProcessB();
{
  // user code
}


void UserInterrupt()
{
  ControlProcessA();
  Controller++;
  if (Controller >= 5) {
    ControlProcessB();
    Controller = 0;
  }
}
```

In this example, the user has defined two functions (ControlProcessA and ControlProcessB). In order for the compiler to establish all the links, the user has still to input the definition of the two functions in the file ***[user.h]***.

4. The command line

Through the command line the user can execute his commands like in a DOS/UNIX environment. In principle, all the commands can be executed from the command line which is very useful when developing an application. When the user places the cursor in the command line (at the left bottom of the screen), he can access the 20 last commands simply using the keys ***[UP]*** and ***[DOWN]*** . By selecting the menu ***[Window → User Commands]*** the list of all the possible line commands is displayed. For example, by double-clicking on [trigger] and hitting then on ***[RETURN]***, the following line appears at the right hand side of the command line :

```
trigger [+/-] [NomVariable] [Valeur Trigger].
```

When the user enters an erroneous command (incomplete or with too many parameters), the program displays the original definition as an help.

Command list:

- **set [VarName] [Value]**
  Changes the value of a variable in memory
  VarName : variable name (!! case sensitive !!)

Value : value to assign to the variable

- **get [VarName]**
  Return the actual value of the variable
  VarName : variable name (!! case sensitive !!)

- **user [start/stop]**
  Controls the user's process
  start : starts the user's process
  stop : stops the user's process

- **ad [Numero canal]**
  Reads the value of the specified A/D channel and displays the result
  Numero canal : channel number to read

- **acq [stop/start/single/trigger]**
  Select an acquisition mode of the data (see chapter 0)
  stop : stops the acquisition of data
  start : starts the acquisition in the continuous mode
  single : starts the acquisition in the single mode
  trigger : starts the acquisition in the trigger mode

- **version**
  Returns the version number of the actual DSP monitor program

- **vecteur [VarName_1, VarName_2, …]**
  Defines the variables to transfer during the acquisition from the supervisor program
  VarName_X : the names of the variables to transfer

- **led [off/on/pr] [n]**
  Controls the 4 leds of the DSP board
  off : switches off the selected leds
  on : switches on the selected leds
  pr : monitor priority (not implemented)
  n : led selection (examples: 5 → leds 1 and 3; 11 → leds 1, 2 and 4)

- **xilinx [eeprom/reset]**
  Xilinx control
  eeprom : programs the FPGA with the code stored in the EEPROM
  reset : resets the FPGA (all the outputs in a high impedance state)

- **xilload [prog/cmp] [dir/flash/ram] [FileName]**
  Downloads the program file for the FPGA circuit
  prog : downloads the program from the supervisor at the specified location
  cmp : compares the program of the supervisor with the one at the destination location (not implemented)
  dir : the destination is the FPGA circuit
  flash : the destination is the eeprom memory location reserved for the FPGA program
  ram : the destination is the DSP RAM (not implemented)
  FileName : source file to download

- **download [flash/flash_reset] [FileName]**
  Downloads the executable file (or user's program) in the EEPROM
  flash : stores the program in the flash
  flash_reset : stores the program in the flash and order the board reset
  FileName : specifies the name of the file to download

- **reset**
  Sends the reset command to the DSP board; the DSP starts with the user's program

- **trigger [+/-] [VarName] [Trigger Value]**
  Defines the trigger condition for the corresponding acquisition mode
  +/- : rising / falling variable
  VarName : variable to monitor as a trigger condition
  Trigger Value : trigger value

5. Configuration

The configuration panel is composed by 3 pages (at present, the third is not used).

5.1. Configuration RS-232

The configuration of the RS-232 interface is very simple. The user has to specify the transmission rate as well as the port to use.

5.2. Compiler options

This page allows the command line of the Analog Devices compiler to be specified. Normally, the user creates the project file without any modification. The compiler command line looks then like the following line :

```
g21k [userfile.c] dspcomm.o dspmon.o -a 062.ach -O2 -nomem –o [prjfile.exe]
```

With these instructions, the compiler compiles the file [userfile.c], links the userfile, the dspcomm file and the dspmon file, uses the architecture file 062.ach and finally, creates the file prjfile.exe.
This command line is very useful when a user wants to create his own code (without the standard DSP monitor program). By a proper modification of the command line, the user can compile his own program from the supervisor and download it on the board (with the Help monitor).

6. Complements to the DSP board

The peripherals of the DSP board can be interfaced with the external world through three connectors, described in the following sections.

6.1. Digital connector

The digital connector outputs to the user the I/O pins of the FPGA circuit, respectively of the MACH circuit. There is a substantial difference between the two circuits :

- To program the MACH, a logic circuit programmer must be available. On the other hand, when the circuit is fed, the MACH pins are operational.
- The FPGA program can be modified very easily (downloading of the *.RBT file). However, the FPGA pins are in a high impedance state unless the user decides to program the circuit.

The digital I/O connector is named J13. The pin configuration is given in Table 1, page 16.

6.2. A/D connector

The connector J17 is directly connected to the A/D converters. The Protel drawing of a A/D channel is reported in Figure 2. The 12 bits converter (LTC1415) has an internal reference voltage of 4.096 V which allows a resolution of 1 mV to be obtained. To protect the converter against possible overvoltages, four diodes (BAV99) have been added to eliminate all the overvoltages greater than the supply voltages.

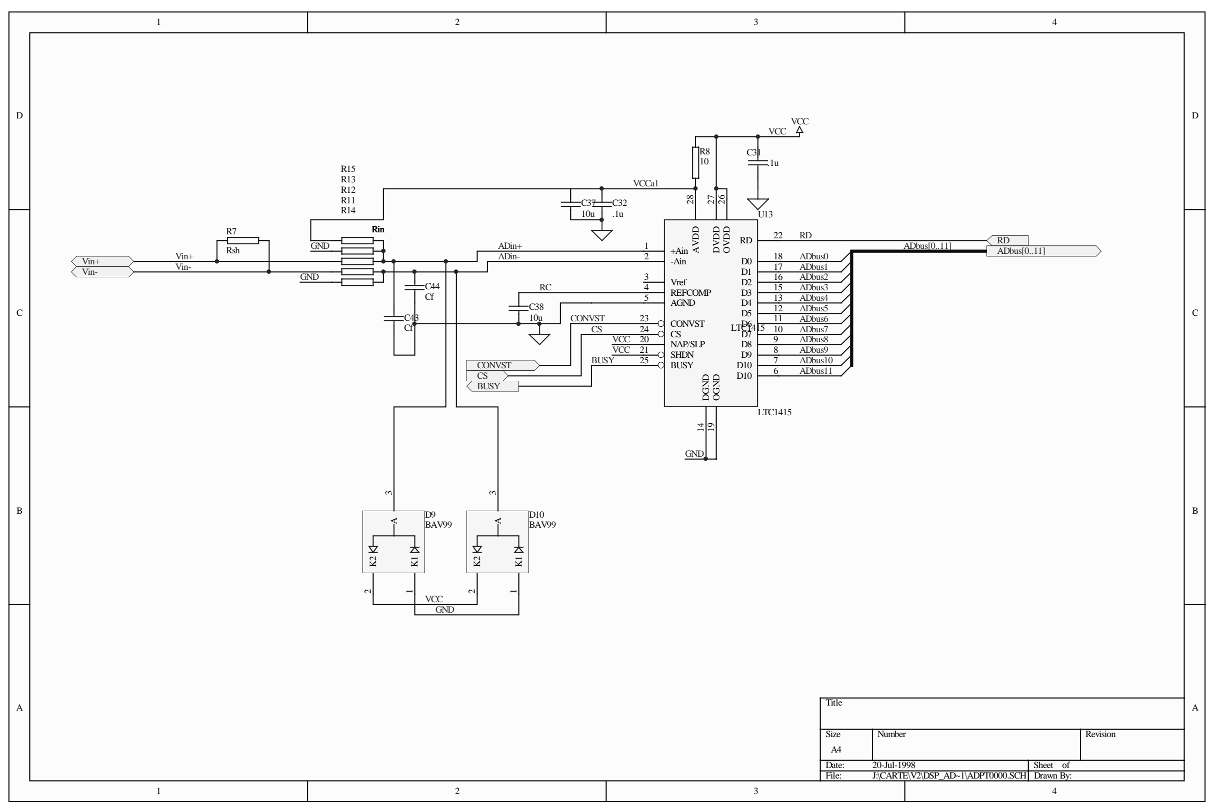


Figure 2 : connection details for an A/D channel

Before the A/D converter there is a shaping circuit that can be configured in the three following manner:

1. ***Unipolar voltage*** (for example a DC voltage)
   The user connects the points Adin- and Vin- to GND, by installing a 10Ω resistor on R14 and R11. The voltage to measure (that must be between the limits 0 V and 4.096 V) is directly applied on the point Vin+.
2. ***Bipolar voltage*** (for example an alternating voltage)
   The user installs four 1 kΩ resistors R15, R12, R11 and R14. With this configuration it is possible to have a voltage offset, determined by the equation:

$$[Adin+]-[Adin-]=\frac{([Vin+]-[Vin-])}{2}+\frac{[VCCa1]}{2}$$

   With this set up, it is possible to have a bipolar low impedance (2kΩ) differential input.
3. ***Current configuration*** (for example, LEM current sensors)
   In this case, we have to add 2 resistances of value Rm at locations R15 and R7, plus three 0Ω resistances at locations R14, R11 and R12 . With this configuration, the voltage between the points Adin+ and Adin- can be computed using the following equation:

$$[Adin+]-[Adin-]=i(R12)\cdot Rm+\frac{[VCCa1]}{2}$$

The pins of the connector J17 are described in Table 1, page 16.

6.3. D/A connector

The J14 connector is used to interface to the external world the four D/A converter outputs. The interfacing is easy: all the outputs VoutX are referred to the GND of the SHARC board.
In the program, the user applies the function [DA(v1, v2, v3, v4)], which output the four values simultaneously.
In the tutorial example, the D/A was used this way:

```
DA(Ia*0x200*0.2+0x200,Ib*0x200*0.2+0x200,
  ThetaU*0x80+0x80,Theta*0x80+0x200);
```

The range of the D/A converter is 10 bits, equivalent to 0x400 (hexadecimal); when the user writes the maximal value, the converter outputs the full VCC voltage. Hence, the desired output value in [V] must be multiplied by the conversion factor (converter range/VCC [V]).

3.67*0x400*0.2 → D/A output value: 3.67 V
((sin(t)+1)*0x200)*0.2 → D/A output value: sinusoidal signal between 0 and 5V
(sin(t)*0x100+0x200)*0.2 → D/A output value: sinusoidal signal between 1.25 and 3.75V

The pins of the J14 connector are described in the Table 1, page 16.

Table 1 : description of the I/O connectors

Conn I/O		
J3	NAME	Xilinx/ Mach
1	GND	
2	VCC	
3	X I/O0	50
4	X I/O1	49
5	X I/O2	48
6	X I/O3	47
7	X I/O4	46
8	X I/O5	45
9	X I/O6	40
10	X I/O7	39
11	GND	
12	X I/O8	38
13	X I/O9	37
14	X I/O10	36
15	N.C.	
16	X I/O12	24
17	X I/O13	23
18	X I/O14	20
19	X I/O15	19
20	GND	
21	X I/O16	18
22	X I/O17	17
23	X I/O18	16
24	X I/O19	15
25	X I/O20	56
26	X I/O21	57
27	X I/O22	58
28	X I/O23	59
29	GND	
30	X I/O24	60
31	X I/O25	61
32	X I/O26	69
33	X I/O27	70
34	X I/O28	67
35	X I/O29	62
36	GND	
37	BIN0	66
38	BIN1	67
39	BIN2	68
40	BIN3	69
41	BIN4	70
42	BIN5	71
43	BIN6	72
44	BIN7	73
45	GND	
46	GND	
47	VCC	
48	VCC	
49	GND	
50	GND	

Conn A/D	
J17	NAME
1	GND
2	VCC
3	AD0+
4	AD0-
5	AD1+
6	AD1-
7	GND
8	AD2+
9	AD2-
10	AD3+
11	AD3-
12	GND
13	AD4+
14	AD4-
15	AD5+
16	AD5-
17	GND
18	AD6+
19	AD6-
20	AD7+
21	AD7-
22	GND
23	AD8+
24	AD8-
25	AD9+
26	AD9-
27	GND
28	AD10+
29	AD10-
30	AD11+
31	AD11-
32	GND
33	AD12+
34	AD12-
35	AD13+
36	AD13-
37	GND
38	GND
39	GND
40	GND

Conn D/A	
J14	NAME
1	VCC
2	VoutA
3	VoutB
4	VoutC
5	VoutD
6	GND
7	VCC
8	Ref_IN
9	Ref_OUT
10	GND