

CHS Engineering
Av. Fusion 170
1920 MARTIGNY
027 7236003
email : info@chs-eng.ch

DAVID system reference

DSP-based development system



Revision 1.4

info@chs-eng.ch

Table of contents

1.	<i>WARNING</i> _____	5
2.	<i>Introduction</i> _____	6
3.	<i>First use</i> _____	7
4.	<i>Help monitor</i> _____	8
5.	<i>Interfaces use</i> _____	9
	5.1 Introduction _____	9
	5.2 A/D converters _____	9
	5.3 MACROS _____	9
	5.4 D/A converter _____	9
	5.4.1 Connector _____	9
	5.5 Binary bi-directional port _____	10
6.	<i>Use of the FPGA</i> _____	11
	6.1 Programming of the FPGA from the user program _____	11
	6.2 Program PWM_CNT _____	11
	6.2.1 Description _____	11
	6.2.2 Connections _____	11
	6.2.3 Registers _____	11
	6.3 Program BUS _____	12
	6.3.1 Description _____	12
	6.3.2 Connections _____	12
	6.4 User configuration for the FPGA _____	13
7.	<i>Memory space</i> _____	15
8.	<i>Software organization</i> _____	17
	8.1 Variables for peripherals access _____	17
	8.2 Communication functions _____	17
	8.2.1 Send a string _____	17
	8.2.2 Messages reception _____	17
	8.2.3 Send a floating number _____	18
	8.2.4 Send an integer number _____	18
	8.3 Use of FLASH memory _____	18
	8.3.1 Introduction _____	18
	8.3.2 Write data _____	18
	8.3.3 Read and clear _____	18
	8.4 Timers _____	19
	8.4.1 Introduction _____	19
	8.4.2 Functions for the use of the timers _____	19
9.	<i>System functions</i> _____	20
10.	<i>Analog signals shaping circuit</i> _____	23
	10.1 Shaping for the measurement of a voltage _____	24
	10.1.1 Limitations _____	25
	10.1.2 Problem induced by default values of resistors _____	25
	10.1.3 Choice of normalized values resistors. _____	25

10.1.4	Example	25
10.2	Shaping for a current measurement.	26
10.3	Position of resistors on the board	26
10.4	Analog connector	28
11.	Using FPGA interrupt instead of internal timer	29
11.1	Introduction	29
11.2	What is changed	29
11.2.1	FPGA preparation	29
11.2.2	IRQ dispatcher	29
11.2.3	Setup interrupt routine	29
11.2.4	Call system processes	29
11.2.5	Restore previous state	30
11.3	Example	30
11.4	Special cases	30
11.4.1	Timer interrupt may be kept	30
11.4.2	Priority of IRQ2	31
12.	Extension cards	32
13.	Questions/Answers	33
13.1	Where and how to specify the sampling rate.	33
13.2	How to discard the monitor	33
13.3	How to make run the program in a stand alone mode.	33
13.4	How to change the flow rate of the serial interface.	33
13.5	How to get two different sampling period	33
13.6	How to measure the execution time of regulation procedures.	33
13.7	How to know which is the minimal possible sampling time.	33
13.8	The bandwidth of the A/D converter is insufficient.	33
13.9	The communication with the board is often looked, or many communication errors appears.	34
13.10	The card is not starting.	34
13.11	The communication does not work or halt often.	34

Software Release 1.7

1. WARNING

The board DAVID is designed for research and development of regulation systems. The implementation of the system has to take into account the fact that interaction with the PC may perturb the regulation algorithm. That condition may result in serious damage to the controlled system. The controlled system must be protected against eventual hardware or software failure of the DAVID board.

CHS Engineering don't assume any responsibility for damages or injuries induced to person or hardware by a dysfunction of the board or the software. The user has the full responsibility of the safety of his system.

Many information given in this document are for the advanced user. Access to configuration registers or changes to the architecture file by example may disturb the handling of the development system.

The information included in this document may change without advertisement.

SUPPORT

Mail to

support@chs-eng.ch

2. Introduction

This document describes the software and hardware related to the DSP board. Follow the First Use Steps of next chapter to start with DAVID. See also the Goliath Reference to startup with the Goliath program.

This board has been designed to comply with most of the regulation requirements. Indeed, some limitations are introduced into the software to keep good efficiency and ease of use. Ask technical support for adaptation to special applications.

3. First use

Power-up the board :

Power :

**Supply the board with a 5 V= /1A power supply.
Use the enclosed cable, RED cable for positive supply.**

The easiest way to start is to define the five next functions in the file user.c. The file has to include the files dspmon.h, dspcom.h and user.h like this

```
#include « dspmon.h »
#include « dspcom.h »
#include « user.h »
#include « io.h »
```

In the file « io.h » are defined macros for peripherals access. The three other files defines prototypes for system function and variables references.

- int UserInit()** This function do the initialization of user variables. This function is called only once when the DSP come from reset. The return value tell how the system start up. The typical tasks are by example initialization of variables, programation of FPGA, computation of constant array, and so on. The return value has to one of the following
- startNever : The user process never start, except if the command is given through the host program
 - startIfAlone : The user process start only if no host program is running.
 - startAlways : The user process start always. This function is provided to let the card function in stand alone mode.
- void UserStart()** This procedure is called when the user process is started. It allows the user to take special action on the peripherals and to initialize variables. The user may start the PWM modulator by example. The sampling time has to be specified in this procedure. (function SetSamplingTime())
- void UserInterrupt()** This procedure is called every sampling time when the process is started. It has to implement the control algorithm..
- Void UserIdle()** This procedure is called in background. There is no sampling time, and the user has to exit often, else the system may not respond to the host commands.
- Void UserStop()** These procedure is called when the process is stopped. It let the user turn off any external device.
- int UserCommand(char * string)** These procedure is called each time a user command is issued from GOLIATH. It allows the user to send actions to his program on DAVID from GOLIATH. See GOLIATH reference for more information.

4. Help monitor

At startup, the DSP load automatically a loader code from the FLASH memory, which will initialize variables and load the program. This code has been modified to let start two different programs, depending on the position of the SW button state at reset. The first page of the FLASH memory is programmed with an help monitor, which is loaded when SW is pressed. Else, the normal application, containing the user program, is loaded from the two next pages. When the user download a program from the PC, it is stored in the two dedicated pages of the FLASH memory. The help monitor is never erased, so it is always possible to restart the board and get communication with it.

The J-TAG interface is available for use of the debugger. The debugger must be acquired from Analog Devices or other third parties.

5. Interfaces use

5.1 Introduction

The peripherals presents on the board are :

- A/D converters
- FPGA
- Binary port
- DA converter

These devices are used through functions or macros defined further. The use of macros is the powerful mean for the use of the interface. It allows to include directly the code in the program. The access to peripheral is often done with 1 or 2 instructions. The call to a function take much time than the access itself.

5.2 A/D converters

The A/D converters are very fast. It is not useful to use an interrupt system to handle the end of conversion. The end of conversion is polled by the program. As the conversion time is equivalent to a few instructions, a little computing task can be done before reading of the result. The converter take 600ns to do the conversion. This mean that conversion time is equivalent to 18 assembler instructions. The bus access take 100ns per A/D.

5.3 MACROS

StartADconv(p).

These macro start the conversion. The parameter p correspond to the converters to be run.

Converters are grouped by two, each bit corresponding to two consecutive A/D. It's a good practice to set p to 0xff value for compatibility with further versions.

IsADready()

These macro allows the test of end of conversion. Test return true if conversion is finished.

The test is done with the first converter, so it must be used to check the end of conversion.

ReadADvalue(channel)

These macro return the conversion result on the specified channel. The result is given in Volt, for an unary gain shape circuit. To change the gain, the macro can be changed in the io.h file.

5.4 D/A converter

The D/A converter used has 10 bits 4 channels. Each channel is constituted with 2 converters, one primary and one secondary. The function access to the primary converter. The system set the secondary to the maximum value. The advanced user can refer to the documentation of the manufacturer for further details.

DA(v1,v2,v3,v4)

The four outputs are updated to binary values v1 through v4. The update is done at the same time. Binary values goes from 0 to 1024.

5.4.1 Connector

The pinning of the connector J14 for the D/A converter is displayed on table 1 :

Table 1 : pinning of connector D/A

Pin	Function	Pin	Function
1	VCC	6	GND
2	V1	7	VCC
3	V2	8	REF IN
4	V3	9	REF OUT
5	V4	10	GND

5.5 Binary bi-directional port

A binary port is at the disposition of the user. The 8 bits of the port are accessible through the binary value of each bit. The port is accessed through the digital port J13. The position of the bits is described in table 4.

The function available for binary port handling are the described below :

BinDir(unsigned mask)

Parameter *mask* give the direction of each bit. A value of 1 set the bit as output. Default setup is input.

BinO(unsigned mask)

Set output state for bit that are set as output.

BitSet(unsigned mask)

Set to one output bits specified by *mask*. Other values are unchanged

BitClear(unsigned mask)

Reset output bits specified by *mask*. Other values are unchanged.

BinI()

Read the state of input pins.

6. Use of the FPGA

The FPGA can be used either with standard programs, either with user programs. The standards programs are described in the next paragraphs. It is possible to design a program for a specific application. This task require the development system from XILINX, and a minimum of know-how.

WARNING : The use of a bad program may destroy the board. When the user try a new program, it has to check that the board is still working. If not, turn off the power immediatly. As the FPGA is a SRAM-based device, the program has to be downloaded at each power-up of the board. The program can be stored in the FLASH memory.

6.1 Programmation of the FPGA from the user program

For stand alone applications, the FPGA can be programmed at startup. This is done by a call to the next function.

int XILINXload() Load the FPGA program from the FLASH. If the function is successful, it return 0.

NOTE : When the user is developing a new program, this function not be used. Prefer the command line command. This way, if the program is not working, the board is restarting without problems.

6.2 Program PWM_CNT

6.2.1 Description

The program PWM_CNT implement a 3-phased PWM modulator, with delay generator and polarity selector. The modulator is build with a triangle generator and three comparators. The switching frequency is selected by the maximum values at which the generator can go before it return to zero. The generator is clocked at 32MHz and is build with a 16bits counter.

The two quadrates impulses encoders interface allows the measurement of a position of a mechanical system. An index allows the recognition of a reference position. The index reset the counter. Three signals are used, referred as A,B and Z. The counter is able to measure the time between two transitions of signal A for accurate speed computation.

6.2.2 Connections

The connections of the program PWM_CNT are displayed on the next table :

Pin	Type	Function	Pin	Type	Function
3	OUT	PWM1_DIR	9	IN	CP1_A
4	OUT	PWM1_INV	10	IN	CP1_B
5	OUT	PWM2_DIR	12	IN	CP1_Z
6	OUT	PWM2_INV	16	IN	CP2_A
7	OUT	PWM3_DIR	17	IN	CP2_B
8	OUT	PWM3_INV	18	IN	CP2_Z

6.2.3 Registers

The registers allows the control of the modulator and access to counters. The global variable XILINX_reg[] give an easy way to access these registers. The next table give the description of each register :

Offset	Write	Read
0	Ratio of modulator 1	Counter 1
1	Ratio of modulator 2	Time counter 1
2	Ratio of modulator 3	Counter 2
3	Maximum value of generator	Time counter 2
4	Bits [7..0] Delay time, bit 15 : 1=enable, bit 14 : 1=reverse polarity	
5	Load a value in counters	
6	Latch counters (write 1 to latch, 0 to unlatch)	
7		Unlatch of counters

6.2.4 Functions

The access to this program can be done through the functions defined in `pwm.c` and `pwm.h` modules. These functions are described below.

<code>void PWMmode(enum t_output to)</code>	Setup mode of the modulator. Parameter <i>to</i> can have values <i>Direct</i> or <i>Inverse</i>
<code>int PWMinit(float Frequency,float Antichev,enum t_output to)</code>	Initialize modulators with following parameters : Frequency : Frequence at which the modulator will run Antichev : Dead time in second to : polarity, <i>Direct</i> or <i>Inverse</i>
<code>int PWMstop()</code>	Stop modulator and put outputs into OFF state. This state is depending on the polarity
<code>static inline void PWMcons(float r1,float r2,float r3)</code>	Give a new reference to the modulator. Ratio <i>r1</i> , <i>r2</i> and <i>r3</i> must be in the range [0..1]
<code>static inline void PWMconsR(r1, r2, r3)</code>	Give a reference to the modulator in integer. The counter limit defines the maximum value.
<code>int ReadCounter()</code>	Read incremental interface counter
<code>void WriteCounter(unsigned v)</code>	Write new value into interface counter

6.3 Program BUS

6.3.1 Description

The program BUS let extend the processor bus to the digital connector. The bus is reduced to 16 bits and 5 address lines. The address space is divided in four parts by the CS[0..3] signals. Each part may use the 3 remaining addresses.

Access to the bus is characterized by the signal CS which goes low while address and data becomes valid. Then one of the signals RD or WR goes low to read or write data.

6.3.2 Connections

The bus is specified in table 2 :

Table 2 : Specification of bus program for the FPGA

Pin	Signal	Pin	Signal
1	GND	26	IRQ

2	VCC	27	LOAD
3	D0	28	CLEAR
4	D1	29	GND
5	D2	30	CS0
6	D3	31	CS1
7	D4	32	CS2
8	D5	33	CS3
9	D6	34	RD
10	D7	35	WR
11	GND	36	GND
12	D8	37	B0
13	D9	38	B1
14	D10	39	B2
15	NC	40	B3
16	D11	41	B4
17	D12	42	B5
18	D13	43	B6
19	D14	44	B7
20	GND	45	GND
21	D15	46	GND
22	A0	47	VCC
23	A1	48	VCC
24	A2	49	GND
25	A3	50	GND

6.4 User configuration for the FPGA

The design of a specific program for the FPGA allows the user to get the maximum power and efficiency from the DAVID board. Nevertheless, some know-how is needed to design such a circuit. The user must get the development system from XILINX to achieve route the digital circuit and produce the configuration file.

WARNING : A bad circuit may destroy the card. The use of a user circuit void warranty of the board. After the programmation, if something is not working properly, turn off the power of the board quickly. The reset button don't reset the FPGA, so it's not sufficient to pressed it.

The interface with the DSP is done like a microprocessor interface. The FPGA is connected to 16 lower bits of the data bus, 5 address lines, RD and WR signal, and a CS which comes from the PLD.

Table 3 : Connection of the FPGA to the system BUS.

Signals	Pins FPGA	Description
A0..A4	25 26 27 28 29	Address lines
D0..D15	10 9 8 7 6 5 4 3 84 83 82 81 80 79 78 77	Data bus
CS	66	Selection of FPGA
RD	14	Signal RD from DSP
WR	13	Signal WR from DSP
ECLK	35	External oscillator (may be mounted next to the

		FPGA)
CLKIN	51	System clock (32MHz)
CLK	72	Clock signal through the PLD (32MHz)
LINK	65	Link with PLD PLD
INT	68	Interrupt request. This signal is routed to the PLD. The Irq dispatcher of the PLD must be configured to route this signal to one of the DSP IRQs inputs. The IRQ is active when this signal is high (PLD does an inversion)

The bus access is characterized by a CS going low, followed by one of the signals RD or WR.

The clock of the FPGA is the same as the one of the DSP. Transfer can be synchronous.

The digital connector is specified in table 4.

Table 4 : Connection of the FPGA on the connector J13. Xn specifies FPGA pin number n.

Pin	Description	FPGA	Pin	Description	FPGA
1	GND		26	X21	57
2	VCC		27	X22	58
3	X0	50	28	X23	59
4	X1	49	29	GND	
5	X2	48	30	X24	60
6	X3	47	31	X25	61
7	X4	46	32	X26	69
8	X5	45	33	X27	70
9	X6	40	34	X28	67
10	X7	39	35	X29	62
11	GND		36	GND	
12	X8	38	37	B0	
13	X9	37	38	B1	
14	X10	36	39	B2	
15	NC	nc	40	B3	
16	X12	24	41	B4	
17	X13	23	42	B5	
18	X14	20	43	B6	
19	X15	19	44	B7	
20	GND		45	GND	
21	X16	18	46	GND	
22	X17	17	47	VCC	
23	X18	16	48	VCC	
24	X19	15	49	GND	
25	X20	56	50	GND	

7. Memory space

The use of the RAM memory by the compiler is specified in the architecture file 062.ach. The DSP uses internal memory starting at address 0x20000 (hex). There is no difference between data memory and program memory. The DSP user 48 bits for program memory, and 32 bits for data memory. The address of memory accessed as data memory is not the same as program memory. The RAM is organized in two blocks. Each of them is constituted of column of 16 bits. Data access result in a two columns organization, and program access result in a 3 column organization. Good practice is to use a block for program, and the other for data. In this configuration, the program is 21800 word long, and data memory is 32768 word long (ADSP 21062). Please refer to DSP documentation for further details.

The peripherals are accessed through the MS0 through MS2 signals. DSP decode external memory access with these signals according to the configuration written in a the systems registers. Environment configure the memory space as following.

MS0	400000h
MS1	480000h
MS2	500000h

MS0 is used for PLD registers and interface access, MS1 give access to ADs, MS2 give access to the FLASH memory.

The peripherals are accessed according to table 5. The functionality of each register is described in chapter 7

Table 5 : Peripherals memory space

400000h	Binary port
400020h	Direction of binary port (1=OUT)
400040h	Start A/D conversion
400060h	IRQ dispatcher.
400080h	LED control
4000A0h	UART access
4000C0h	FPGA control
4000E0h	D/A space. Refer to manufacturer documentation for details
400100h	Latch control of D/A converter. Write to this register to activate the latch command of the converter.
400120h	FPGA memory space. Depends on the program loaded in the FPGA
	The A/D converters are accessed through the following addresses
480000h	AD0
480020h	AD1
480040h	AD2
Etc	
4801A0h	AD13

The FLASH memory is mapped in the space going to 500000hex to 580000h. The size is 80000h. Data is accessed by 8 bits bus on the lower part of the bus. The use of the FLASH is the described in table 6.

Table 6 : FLASH memory organization

0-0xFFFF	Help monitor
0x10000-0x2FFFF	User program
0x30000-0x4FFFF	FPGA configuration
0x50000-0x5FFFF	User data
0x60000-0x6FFFF	Reserved

8. Software organization

8.1 Variables for peripherals access

Peripherals may be accessed by a C program through the next variables or arrays.

Table 7 : Variables for access to peripherals

Unsigned BinIO_reg	R/W	Read or Write to the binary port
Unsigned BinDIR_reg	W	Write of the binary port bits direction
Unsigned AD_reg	W	Start conversion of A/D. write 0xff.
Unsigned IrqDisp_reg	W	IRQ dispatcher. There are 3 IRQ sources : End of A/D conversion, UART and FPGA. Each input can be mapped to one of the 3 DSP interrupt lines, according to the next table : bits 0,1 Select source for IRQ2. 0 : no IRQ, 1 : AD, 2 : UART, 3 : XILINX bits 2,3 Same for IRQ 1 bits 4,5 Same for IRQ 0 Bit 6 Enable start of conversion with IRQ
Unsigned LED_reg	W	Configuration of LEDs. 1 turn off the LED, 0 turn on the LED
Unsigned UART_reg[15]	R/W	Access to the UART. Refer to manufacturer documentation for further details (PC 15550)
Unsigned XILINX_reg[0x20]	R/W	Access to FPGA registers. The meaning depend on the FPGA program.
Unsigned XILCTRL_reg	R/W	FPGA programming control
Unsigned FLASH[0x1]	R/W	FLASH memory. Only the lower 8 bits are used. When reading, the value must be anded with 0xff to void other bits. The real size is 0x80000. The chip used is the AM29F040.
Unsigned DAval_reg[8]	W	Access to DA converter. Refer to manufacturer documentation for further details.
Unsigned DALatch_reg	W	Latch of D/A values
Unsigned AD_val[0x200]	R	Access to A/D conversion result. The offset between each A/D is 0x20

8.2 Communication functions

8.2.1 Send a string

A string may be sent to the host with the next function `msgSendString(char *)`.

The call has to be done from the `UserIdle` function.

8.2.2 Messages reception

Host may send messages to DSP. These messages can be retrieved with a call to the next function :

`Int MsgGetUserMessage(char * str)`.

If the function return true, a message is available on the string str.

8.2.3 Send a floating number

Simple floating number message can be send with the function

Void MsgSendUserFloat(char *ch,float f).

Ch is a 4 chars string

8.2.4 Send an integer number

Simple integer number message can be send with the function

Void MsgSendUserFloat(char *ch, int i).

Ch is a 4 chars string

8.3 Use of FLASH memory

8.3.1 Introduction

FLASH memory can hold user data. The system provides some functions to handle data stored in the FLASH memory. This system work like a reduced file system. An ID serves as file identifier.

8.3.2 Write data

Data may be written through the following function :

int FLASHWriteData(unsigned len, const void * ptr, int ID);

write len data from the pointer position ptr. The ID serves as reference to these data. If a block exists with this is, it is erased. The function return following results

0	if operation success.,
1 or 3	if there is a problem with data in the FLASH,
2	if there is not enough room remaining n the FLASH. Flash must be erased to refresh all datas.

8.3.3 Read and clear

Reading of previously stored data is possible with the following function.

int FLASHReadData(unsigned len,const void * ptr,int ID);

len must be the same as when writing.

All data in the FLASH memory may be erased with the following function.

The function returns following results

0	if operation success,
1 or 3	if data in the FLASH are corrupted,
2	if the ID don't exist,
4	if the length is not the same.

int FLASHEraseData(unsigned magic);

magic value must be 0x1234.

8.4 Timers

8.4.1 Introduction

Timers are used for time managing. They allows multiple time dependent tasks to be accomplished. Basic sampling time is given by the sampling rate specified by SetSamplingTime function call. Timers period are specified in seconds. A divider is computed when SetSamplingTime is called. This is an integer divider, rounded to the next value of desired sampling time.

There are two ways to use timers.

1. Polling it to know when timer is out
2. Assign a function to the timer, which is called when timer is out.

8.4.2 Functions for the use of the timers

A timer is initialized by a call to function

int GetTimerHandle(int * hh,float TimerBase);

value of hh must be -1 before the call, and the handle is returned into hh. Function return true if success. Function may fail is there is not enough timers available.

int IsTimerOut(int hh);

This function returns true if timer is out.

int ResetTimer(int hh)

This function resets the timer. The period is restarted.

int ReleaseTimer(int * hh);

This function release timer to the system. The handle hh is reset to -1.

int SetTimerProc(int * hh,float TimerBase,void(*Func()));

Set function Func on a timer. This function will be called each time the period is out. To have a function which is called only once. The function may release the timer with a call to ReleaseTimer(&hh).

9. System functions

The system functions are briefly described below. This is for information only, provided without support.

void msgHandler()	Handle a received message
void msgInit()	Initialization of message system
void msgTimeOut()	Reset of message receiver
void msgIdle()	Message receiver
void UART_init(int div);	UART and buffer initialization
void UART_handler(int signal);	Interrupt procedure. Handle the interrupt of the UART. Send data in the output buffer, and put incoming data in the input buffer.
int UART_isdata();	Check if there is data in the input buffer
char UART_get();	Retrieve a char from the input buffer
void UART_send(char ch);	Send a char to the serial port
void UART_sendstr(char*str,int l);	Send a string to the serial port. Parameter l specifies the length of the string. Any char may be in the string
int UART_BufferFree();	Return free space of output buffer
float GetAD(int ch);	Start a conversion on the A/D converter, wait for the result, and read the specified channel.
float ReadADValue(int ch);	Read the specified A/D converter
int IsADReady();	Test end of conversion. Return true if conversion is finished.
void StartADconv(int channel);	Start a conversion on specified channels
void AD_handler(int);	Interrupt procedure for A/D converter. Obsolete
Void DA(unsigned v1,unsigned v2, unsigned v3, unsigned v4) ;	Output of 4 values to D/A converter
int GetTimerHandle(int * hh,float *TimerBase);	Get a Timer handle for time process. Timer period is specified in seconds.
int IsTimerOut(int hh);	Return true if timer is out
int ResetTimer(int hh) ;	Reset timer count
int ReleaseTimer(int *hh);	Give back the timer to the system
int InitTimerHandler();	Timer system initialization.
int SetSamplingTime(float);	Change sampling time.
void TimerHandler(int signal);	SHARC timer interrupt handler.
int SetTimerProc(int * hh,float Te,void(*Func)());	Setup of a procedure on a given timer. Procedure will be called at specified period. Keep handle for further reference.
void Trigger(void (*Func)()) ;	Add a procedure to the list of function to call. Functions are called as low priority procedures.
Void TimerProcStarter(int signal) ;	Manage the list of functions to call.
void BinO(unsigned mask);	Set binary port output
void BitSet(unsigned mask);	Set specified bits according to the mask
void BitClear(unsigned n);	Clear specified bits, according to the mask
void BinDir(unsigned mask);	Select direction of binary port, according to the mask (1=OUT)
unsigned BinI();	Read of binary port.
void LEDon(int mask);	Turn on the LEDS specified by mask
void LEDoff(int mask);	Turn off the LEDS specified by mask
void msgSendString(const char	Send a string to the host (C string)

```

*ch);
void msgSendStringC(int          Send a string with a specific message code
code,const char *ch);
void msgSendInt(const int v);    Send an integer
void msgSendIntC(int           Send an integer with a specific code
code,const int v);
void msgSendFloat(const float   Send a float
v);
void msgSendFloatC(int         Send a float with a specific code
code,const float v);
void msgSendVector(const int    Send the acquisition vector
id);
void msgSendCode(int n,int     Send a message
c,...) ;
void msgHallo();              Say 'hello' to the host
int msgDefineVector(int        Define an acquisition vector. Pointers are given as parameters
size,float * v1,...);
int msgDefineVectorTab(int     Define an acquisition vector from an array of pointers
size,float**);
int msgVersion(char * ch);     Test and decode a message of type Version. next functions does
                                the same process for each message the monitor can handle.
int msgUserStart(char * ch);   idem Start
int msgADconv(char * ch);      idem A/D conversion
int msgSetFloatValue(char *    idem set a float value
ch) ;
int msgSetIntValue(char * ch) ; idem set an integer value
int msgGetFloatValue(char *    idem send a float value
ch) ;
int msgGetIntValue(char *      idem send an integer value
ch) ;
int msgChangeLED(char *        idem turn on/off the LED
ch) ;
int msgSetVector(char * ch) ;  idem vector
int msgAcqControl(char * ch) ; idem acquisition control
int msgDefTrigger(char * ch) ; idem trigger definition
int msgTe(char * ch) ;         idem return sampling period
int msgXILINXctrl(char * ch) ; idem control of FPGA
int msgXILINXdata(char *      idem data for FPGA
ch) ;
int msgUserPgm(char * ch) ;    idem send user program
int msgPresence(char * ch) ;   idem remote test presence
int AcqSetupMemory() ;         Setup acquisition memory
void AcqInit();                Initialize acquisition system. Acquisition is set in stop mode
int AcqStoreVector() ;         Store acquisition vector
int AcqGetAndSendVector() ;    Send a stored vector to the serial port
void msgHandler();             Decode a message. Call successively all the decoders to find the
                                right one. Give back an error string if no decoder want the
                                message.
int UserInit();                User initialization

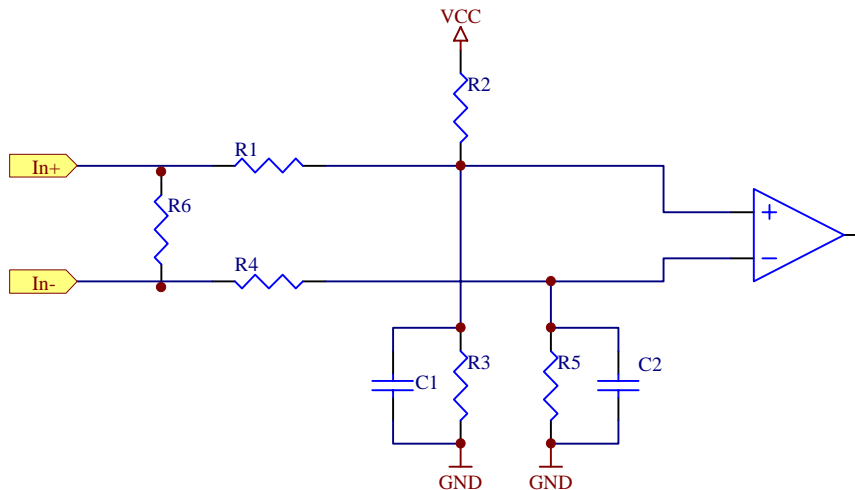
```

void UserIdle();	User background task
void UserStop();	User process stop
void UserAD();	User AD interrupt
void UserInterrupt();	User interrupt procedure
void ClearSection(unsigned section,unsigned magic);	Clear a page of FLASH memory. Section must be the first address of the page, magic must be 0x1234
int DataPolling(unsigned Ad,unsigned V);	FLASH write process procedure. Test end of write
int FLASHWriteByte(unsigned address,unsigned value);	Write an octet to the FLASH memory
void FLASHWriteWord32(unsigned Waddress,unsigned value);	Write a word to the FLASH. Waddress is multiplied by 4 to give the address of the first FLASH address.
unsigned FLASHReadWord32(unsigned Waddress);	Read a word from the FLASH memory
int FLASHWriteData(unsigned len, const void * ptr, int ID);	Write a bloc of data in the FLASH memory. ID is like the name of the block. It is possible to write on previous data with the same ID. The function return 0 if operation success., 1 or 3 if there is a problem with data in the FLASH, 2 if there is not enough room remaining n the FLASH.
int FLASHReadData(unsigned len,const void * ptr,int ID);	Read a block of data from the FLASH. The length len must be the same as the value given for the write process. The function returns 0 if operation success, 1 or 3 if data in the FLASH are corrupted, 2 if the ID don't exist, 4 if the length is not the same.
int FLASHEraserData(unsigned magic);	Clear the page dedicated to the data. All data are lost
int XILINXreset();	Clear FPGA program
int XILINXsendBlock(char *);	Send a block of data to the FPGA
int XILINXrun();	Start FPGA program
int XILINXload();	Load and start FPGA from the FLASH

10. Analog signals shaping circuit

The shaping circuit of the analog signals is build with 6 resistors which allows the adaptation of the input range of the A/D converters. As it is a passive circuit, the circuit has some limitations. The impedance of the sensor must be taken into account and it is not possible to do an amplification.

The range of the converter is 0 to 4.096V. The shaping circuit is shown on the next picture.



At factory, the card is mounted with resistors R1 and R4 with a value of 100ohms, and capacitors C1 and C2 with a value of 1000pF. These values allows the full bandwidth use of the A/D converter with a very good accuracy.

IMPORTANT : It is possible to add or to modify the value of the components to fit the shaping circuit to a specific application. The soldering and unsoldering of the components must be made with care to avoid damaging the printed board. A force to high on the pads may destroy it. The solder should be removed with adequate tools, or both side of resistors must heated simultaneously. For soldering a resistor, first place solder on one pad, fix the resistor with this pad, and sold the second pad. As there is already solder on the pads from the mounting process of the board, first remove the solder of one of the pads before mounting a resistor.

The choice of capacitors must take into account the impedance of capture circuit of A/D converter. A minimal value of 1000pF must be kept for accuracy. The total resistance of resistor and sensor resistor are added to compute the bandwidth of shaping circuit. A resistance value of 100ohms allows full bandwidth usage.

The purpose of resistors is described in the next table :

Tableau 8 : Purpose of resistors

	Purpose	Default value
R1	Make a first order filter with the input capacitor. A voltage divider may be done with the resistor R3	100
R2	This resistor allows the introduction of an offset from the supply voltage.	
R3	May be used to do a voltage divider with R1	
R4	This resistor as the same purpose as R1, but for the negative input.	100
R5	Same purpose as R3, May be used to bindle the reference to the ground of the board.	
R6	This resistor may be used to control the impedance of the input circuit. Use with a current output sensor.	

The shaping circuit allows to fit requirements of most of the situations. The A/D can't handle negative inputs. For bipolar signals, an offset must be introduced with resistor R2.

The resistors may affect the precision of the converter. Shaping circuit should be calibrated before use. The introduction of an offset from the power supply may be hazardous if the power supply is not accurate. It is possible to use one of the converters to measure the power supply voltage.

10.1 Shaping for the measurement of a voltage

The most general application consists to use the shaping circuit to add an offset and do an attenuation of the signal. In this case, the problem can be divided in two parts :

1. Establishment of an offset on the positive input of the A/D converter,
2. Choice of a gain G less than 1

The voltage at input of the A/D converter can be computed by the following relation :

$$V_+ = \frac{R_1 R_3}{R_1 R_3 + R_2 R_3 + R_1 R_2} V_{CC} + \frac{R_2 R_3}{R_1 R_3 + R_2 R_3 + R_1 R_2} V_{in+}$$

The first term gives the offset, and the resistors factor give the gain. Taking R1 as reference, the relative value of the others resistors is given by the following relation :

$$r_2 = \frac{G \cdot V_{CC}}{V_{offset}} \quad (1)$$

$$r_3 = \frac{G \cdot V_{CC}}{V_{CC}(1 - G) - V_{offset}} \quad (2)$$

The values of the resistors are computed by multiplying the value of R1 by the factors.

The value of R2 must take into account a filtering resistor of 10 ohms on the power supply.

The effective value must be chosen 10 ohm less than the computed value.

The values of the resistors is then :

$$R_3 = r_3 R_1$$

$$R_2 = r_2 R_1 - 10$$

The circuit can accept a common mode of both entries if this one is positive. The voltage at the input must not exceed 5V. Diodes protect the A/D against voltage excess, but the current must not exceed 200mA.

The value of R4 and R5 must be chosen to satisfy the selected gain G. Taking R4 as reference, the value of r5 is simply :

$$r5 = G/(1-G) \quad (3)$$

The final value of R5 is given by the following relation :

$$R5 = r5R4$$

10.1.1 Limitations

Given the offset you choose, you may not succeed to find a value for both resistors. If the value is negative, this means that you cannot do the shaping you want. If you have a given offset, the gain is limited by the relation :

$$G < 1 - \frac{V_{offset}}{VCC}$$

If the gain is fixed, the offset must satisfy this relation :

$$V_{offset} < VCC(1 - G)$$

10.1.2 Problem induced by default values of resistors

The default values of the input resistors may request important current from the power supply. As showed after, a conversion of a +/- 10V current produced an equivalent resistor of 100ohms on the power supply. This value induced a current of 50mA by channel. The power supply of the converter is lowered by 0.5V.

10.1.3 Choice of normalized values resistors.

The values given by the computation is not necessarily available in normalized values. The resulting gain is changed, so the effective gain must be computed according to the selected values..

To keep a good accuracy, the resistors chosen must be precision resistors. It is possible to calibrate the circuit and take the effective gain in account in the software.

10.1.4 Example

To get a bipolar entry that transform a voltage in the range +/- 10V to voltage in the range of the A/D converter, one should process like that :

The gain is the relation between the input range and the range of the converter. For simplification, the input range of the converter is rounded to 4V. This give the this relation for the gain :

$$G = \frac{4}{20} = \frac{1}{5}$$

To compute the offset, the input voltage has to be the half of the converter range, 2V. The resistors value is given by (1) and (2) :

$$r2 = \frac{\frac{1}{5} \cdot 5}{2} = 1/2$$

$$r_3 = \frac{\frac{1}{5} \cdot 5}{5(1 - \frac{1}{5}) - 2} = 1/2$$

For R1=100 ohms, R3=50 ohms and R2=40 ohms.

The ratio r5 is found from (3)

$$r_5 = 1/4.$$

For R4=100, the value of resistor R5 is 25ohms.

These values lower the voltage supply of the converter. It may be preferable to use greater values, by multiplying all the resistors values by a factor 5 or 10. The bandwidth is affected in relationship.

10.2 Shaping for a current measurement.

The measurement of the current uses shunt resistor R6 to convert the current into a voltage. A value of 0 ohms for R5 fixes the reference for the converter. If an offset is needed, it may be introduced with R2. The offset is dependent of the value of R6. The equivalent shunt resistor is then the value of R6 and R2 mounted in parallel. If desired value of shunt is Rsh, the values of R6 and R2 is given by :

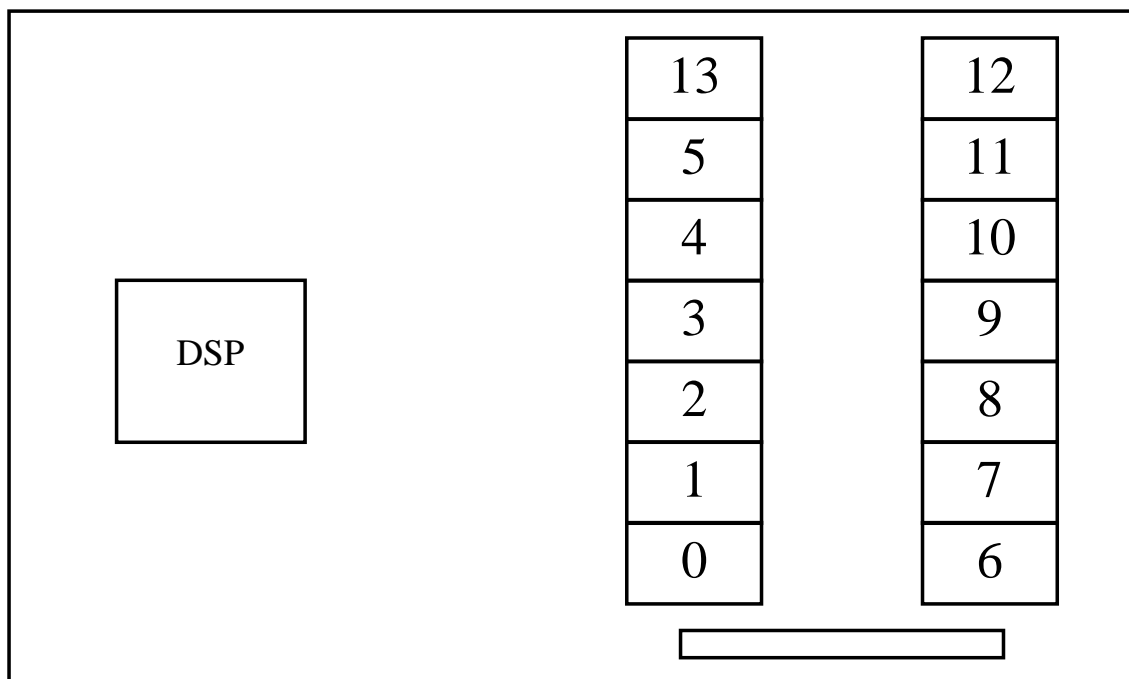
$$R_6 = \frac{R_{sh}VCC}{VCC - V_{offset}}$$

$$R_2 = \frac{R_{sh}VCC}{V_{offset}} - 10$$

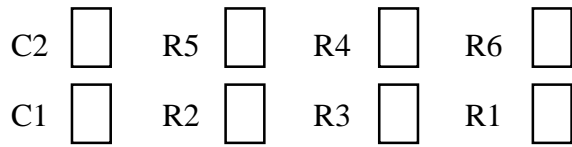
The value of R2 is lowered by 10 to take into account the power supply resistor.

10.3 Position of resistors on the board

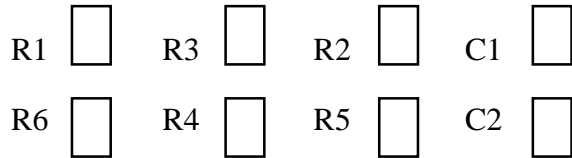
The configuration of the resistors is the same for each channel. The channels are disposed as shown on the next figure.



The disposition of the 6 resistors and the 2 capacitors is showed on the next picture for the channels 0 to 5 and 13.



For the others channels, located to the right



The relation between the designators of the resistors in the schematic and the serigraphy print of the board is the showed in table 9.

Table 9 : Relation between schematic designators and board serigraphy.

Sch	0	1	2	3	4	5	6	7	8	9	10	11	12	13
R1	R12	R17	R22	R38	R43	R48	R88	R83	R78	R73	R68	R63	R58	R53
R2	R15	R20	R25	R30	R46	R51	R91	R86	R81	R76	R71	R66	R61	R56
R3	R13	R18	R23	R39	R44	R49	R89	R84	R79	R74	R69	R64	R59	R54
R4	R11	R16	R21	R37	R42	R47	R87	R82	R77	R72	R67	R62	R57	R52
R5	R14	R19	R24	R40	R45	R50	R90	R85	R80	R75	R70	R65	R60	R55
R6	R7	R92	R93	R94	R95	R96	R104	R103	R102	R101	R100	R99	R98	R97
C1	C43	C45	C47	C95	C97	C99	C115	C113	C111	C109	C107	C105	C103	C101
C2 :	C44	C46	C48	C96	C98	C100	C116	C114	C112	C110	C108	C106	C104	C102

10.4 Analog connector

Table 10 : Analog connector

Pin	Signal	Pin	Signal
1	GND	21	Vin7-
2	VCC	22	GND
3	Vin0+	23	Vin8+
4	Vin0-	24	Vin8-
5	Vin1+	25	Vin9+
6	Vin1-	26	Vin9-
7	GND	27	GND
8	Vin2+	28	Vin10+
9	Vin2-	29	Vin10-
10	Vin3+	30	Vin11+
11	Vin3-	31	Vin11-
12	GND	32	GND
13	Vin4+	33	Vin12+
14	Vin4-	34	Vin12-
15	Vin5+	35	Vin13+
16	Vin5-	36	Vin13-
17	GND	37	GND
18	Vin6+	38	GND
19	Vin6-	39	GND
20	Vin7+	40	GND

11. Using FPGA interrupt instead of internal timer

11.1 Introduction

The use of the FPGA as interrupt source is very useful in power electronics. This feature gives the possibility to synchronize the regulator with the modulator by example.

In addition, the interrupt may start A/D conversion. This feature has many advantages :

- The A/D conversion start at a very precise time
- The A/D conversion is finished when the DSP start user routine.
- The A/D conversion measure values when no switch is working (in most of conditions)

11.2 What is changed

To install an interrupt routine on an IRQ driven by the FPGA, the procedure is the following :

11.2.1 FPGA preparation

Program the FPGA with a program which use the INT connection (see FPGA chapter). If the program is not loaded before, the DSP may hang in the interrupt routine

11.2.2 IRQ dispatcher

The IRQ dispatcher is used to direct interrupt sources to IRQ lines of the DSP. Setup the IRQ dispatcher to use either IRQ2 or IRQ0. The IRQ1 is used by the UART.

11.2.3 Setup interrupt routine

The interrupt routine can be easily set up with the interrupt() function. This is part of Analog Devices compiler run time library. See ADI references for more information.

In some cases, it may be useful to remove normal timer interrupt routine. This is done the same way with interrupt() function

The sampling period must be given to the new timer process...

11.2.4 Call system processes

There are two system processes used with timer interrupt.

11.2.4.1 Acquisition process

Acquisition process is synchronized with regulation process to guaranties that each call of the regulation algorithm is acquired once and only once.

This call is done in the original timer handling procedure. If the user write his own procedure, this function has to be called. The function handling the acquisition is the following :

```
sysProcAcquisition();
```

11.2.4.2 Timers handling process

Timers are used to manage other sampling periods with lower frequencies as the main regulation process. One of these timers make the LED blink.

To make the timers system work, a function call must be issued at a given frequency. The function call is the following :

```
sysProcTimer();
```

11.2.5 Restore previous state

To restore initial state, the interrupt has to be restored to timer with a call to `interrupt()`.

11.3 Example

Following example shows how to trigger interrupt with FPGA. The new function is called `PWMUserInterrupt()`. It will be called by modulator interrupt.

```
void UserStart()
{
    // set internal IRQ register to drive IRQ2 with modulator
    interrupt
    IrqDisp_reg = 0x4b;

    // tell the system the new interrupt frequency
    SetSamplingTime(0.5/PWM_FREQ); // adapt timer handling of
    monitor program

    // interrupt setup
    interruptf(SIG_TMZ0,SIG_IGN);
    interruptf(SIG_IRQ2,PWMUserInterrupt); // setup the procedure
    to PWM interrupt

}

void UserStop()
{
    // Interrupt restoration
    interruptf(SIG_IRQ2,SIG_IGN);
    interruptf(SIG_TMZ0,TimerHandler);
}

void PWMUserInterrupt(int signal)
{
    LEDon(0x4); // used for time measurement with an
    oscilloscope

    // enter your code here
    ...

    sysProcAcquisition();
    sysProcTimer();

    LEDoff(0x4); // used for time measurement with an
    oscilloscope
}
```

11.4 Special cases

11.4.1 Timer interrupt may be kept

If the timer interrupt can be kept, but if it is not part of the regulation algorithm, the monitor will call `sysProcAcquisition()` with timer interrupt. In this case, the acquisition is not

synchronized with regulation process. The user should install a special timer interrupt handler which simply call `sysProcTimer()`.

11.4.2 Priority of IRQ2

The IRQ2 priority is lower than TMZ0 interrupt. If timer interrupt should have lower priority than the IRQ2, install the timer handler routine (`TimerHandler()`) on TMZ interrupt. This interrupt is also driven by the timer, with a lower priority.

12. Extension cards

Some specific applications need special signal conditioning. These cards may be plugged under the processor board. The actually available cards are listed below.

1. Interface card for power electronic

This card has 16 fiber optic emitters for power electronics devices, 14 analog differential inputs, 4 fiber optic inputs and 4 opto-coupler inputs.

Reference : DC-POWER

2. Acquisition card

This card uses a shaping circuit for global shaping of analog signals. It may be used to measure voltage from $\pm 10\text{V}$ range to $\pm 10\text{mV}$ range. A shield bus is available to connect many cards together. Memory has been extended to match large acquisition needs.

Reference : DC-ACQ

3. D/A card

The D/A card provides 2 D/A converters. Each of them provides 8 channels, 13 bit accuracy. The voltage range is $\pm 4.096\text{V}$.

Reference : DC-DA16

13. Questions/Answers

13.1 *Where and how to specify the sampling rate.*

Use SetSamplingTime (Te) in the UserStat() procedure. The sampling time is reset when user application is halted

13.2 *How to discard the monitor*

Simply compile the program without the monitors objects. Once downloaded, the card will start it. To get the communication working again, reset the card while button SW is pressed. (or write your own GOLIATH)

13.3 *How to make run the program in a stand alone mode.*

Return startAlways from the function UserInit()

13.4 *How to change the flow rate of the serial interface.*

Call the function UART_init(div) with the right divider. This divider is computed like that:

$$div = \frac{22118400}{16 \cdot baudrate}$$

The default value is 12 for 115kbs.

This function call must be placed in UserInit().

13.5 *How to get two different sampling period*

Choose a period which is a multiple divider of both sampling rates for the timer frequency. Use SetTimerProc to define which functions should be called at which sampling rate. The shortest possible sampling rate is in the range of 20us.

13.6 *How to measure the execution time of regulation procedures.*

The best way is to use binary outputs. Use BitSet and BitClear to modify the bits, setting 1 at beginning of the procedure, and 0 at the end. Use a scope to measure the sampling rate.

13.7 *How to know which is the minimal possible sampling time.*

Measure the execution time of the regulation procedure, and add 10 to 15us for the monitor execution. The time used by the monitor may vary with the number of acquisition variables selected.

13.8 *The bandwidth of the A/D converter is insufficient.*

Verify the impedance of the sensor. A capacitor of 1000pF is placed at the input of the A/D converter. The added values of the input resistor and the sensor impedance make a filter. To get the maximum bandwidth, this value should be less than 100 ohms. If the value is

insufficient, an operational amplifier must be added. The value of the capacitor must not be lowered, because the accuracy of the converter may be changed.

13.9 The communication with the board is often looked, or many communication errors appears.

The PC cannot memorize all data coming from the card. This problem may be solved by lowering the thresholds of the FIFO of the serial interface of the PC. (advanced parameters)

13.10 The board is not starting.

Restart the card on the help monitor by holding the SW2 button while resetting the card. If a bad FPGA program has been downloaded, turn off the power, and restart the board while pressing the SW2 button.

13.11 The communication does not work or halt often.

A uncompleted message block the Goliath program. To solve the problem, go in the RS232 setup panel, and click OK to reinitialize the communication port. This probability of this condition may be reduced by avoiding the use of continuous acquisition. Prefer single or trigger acquisition.