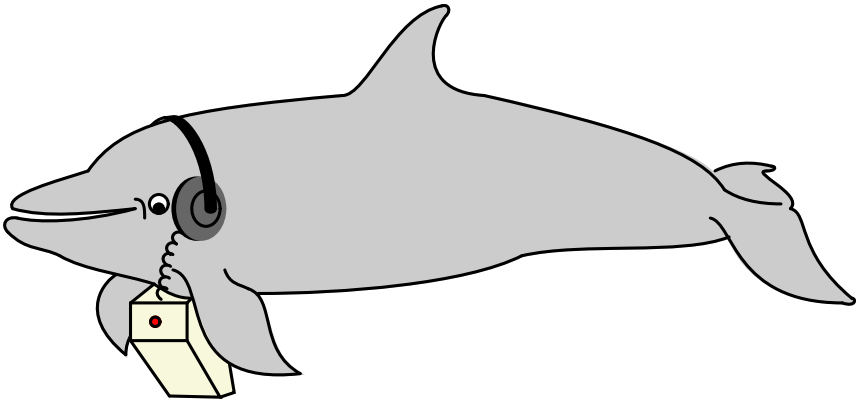
The BioSonar Front-end Signal Processor System Manual

R. Timothy Edwards
Johns Hopkins University Applied Physics Lab
11100 Johns Hopkins Road
Laurel, MD 20723-6099 USA
email: tim@stravinsky.jhuapl.edu



December 28, 2001

Chapter 1

Introduction

Certain animals such as bats and dolphins generate sonar signals (chirps or clicks) for the purpose of echolocation and object detection and classification. "Biomimetic sonar" refers to electrically-generated sonar signals designed to match the properties of the biologically-generated sonar signals. A biomimetic sonar signal processing system attempts to electronically mimic neural processing. This may be for the purpose of biological modeling, or for performing a task such as object recognition.

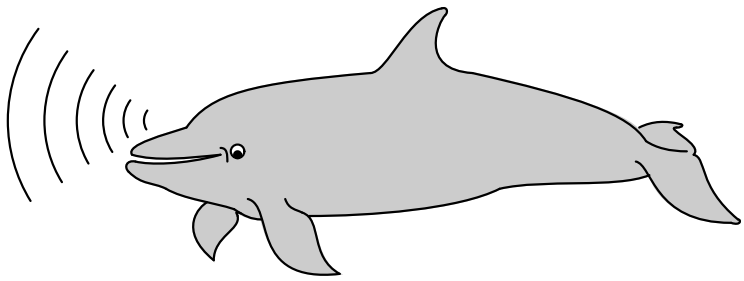


Figure 1.1: Bottlenose dolphin.

The Biomimetic Sonar Front-end Signal Processor System (generally referred to in this manual as the "BioSonar Processor") was designed as a board-level system to investigate architectures for a (future) single-chip integrated biosonar signal processor. At the same time, however, the system is a functional analog signal processing system, capable of performing basic tasks associated with biological cochlear processing, including gain control, filtering through parallel bandpass functions, signal rectification, amplitude envelope extraction, and zero crossing detection. The system was designed using the latest commercial programmable analog and digital technology to make the system architecture flexible. The board relies on In-System Programmable ("ISP") circuits such that the system architecture may be reconfigured on the fly from an external controller (such as a computer with a digital I/O interface). However, the ISP circuits use non-volatile memory to store configuration information, so the biosonar front-end board retains its configuration when powered off, and may be programmed to operate independently of any external controller.

Chapter 2

System Overview

Figure 2.1 shows the system setup as currently expected by the driver program. In this configuration, the board processes digital signals downloaded from the host computer. Main components of the system are as listed below:

A Host computer, Intel or equivalent, running Linux.

B Digital I/O card, PCI-DIO48, CIO-DIO48, or equivalent.

C Bidirectional FIFO buffer board

D BioSonar frontend processor

E DC Power supply capable of delivering 1.2 A at 5 V.

Because the host computer is using a non-real-time operating system (Linux), and also because the 82C55-based digital I/O card cannot keep up with the data rate of the biosonar processor, a bidirectional FIFO must be placed between the computer and the biosonar board. When real-time analog input is available to the board, and the board output directly drives a real-time back-end processor, the bidirectional FIFO buffer is not needed. That situation also allows the board to be configured to run without any connection to the computer; the digital interface is then used for new configuration downloads only.

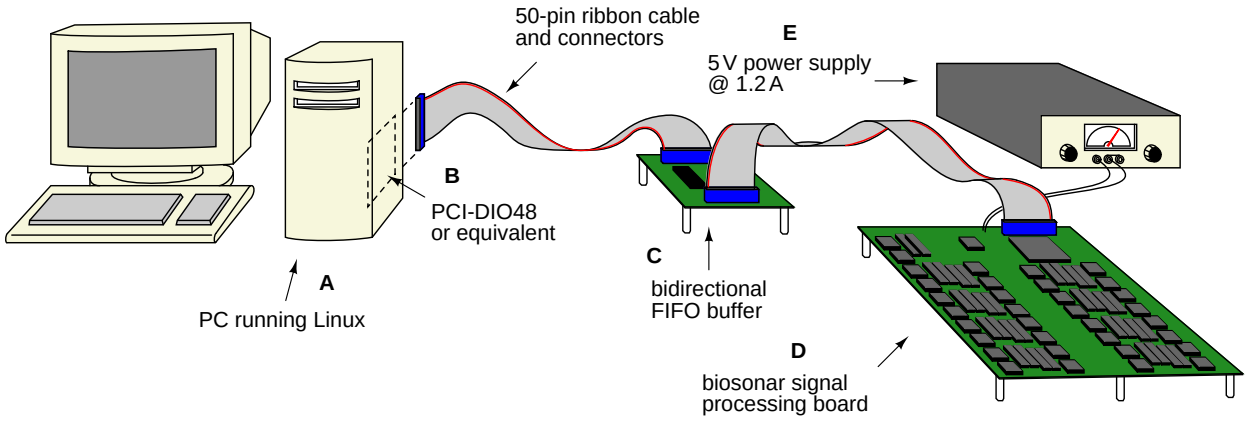


Figure 2.1: Biosonar Signal Processor: System-level diagram.

Figure 2.2 is a depiction of the BioSonar frontend processor board. The board contains a filterbank of 34 channels implemented by Lattice ispPAC programmable analog arrays (one ispPAC-10 and one ispPAC-20

for each filter), one ispPAC-20 chip acting as a preamplifier with digital or analog input selectable, four Maxim MAX1202 12-bit analog-to-digital converter chips, a Lattice ispLSI-6192FF field-programmable gate array (FPGA) chip acting as an I/O interface, and a 20 MHz clock to drive the system. The ispPAC chips contain continuous-time biquad filter sections programmable in a range of 10 kHz to approximately 150 kHz, depending on the programmed Q value and input gain selection. This is contrary to the indications of the Lattice documentation, which declares the upper limit to be 100 kHz. The continuous-time nature of the chips also means that the preamplifier is not subject to the Nyquist frequency of the digital input, as would a digital or discrete-time filter.

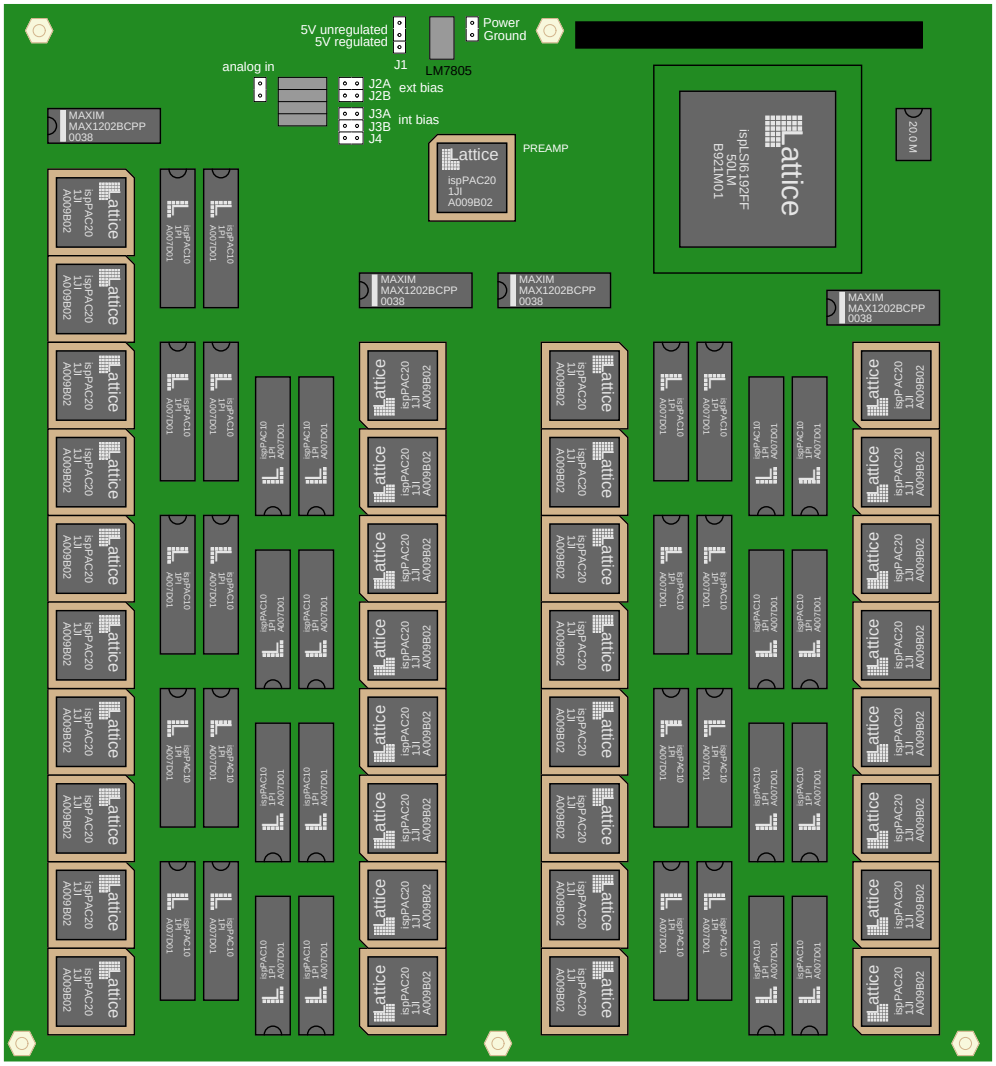


Figure 2.2: Biosonar Signal Processor: Frontend circuit board.

The signal-level block diagram of the frontend board, indicating its major components, is shown in Figure 2.3.

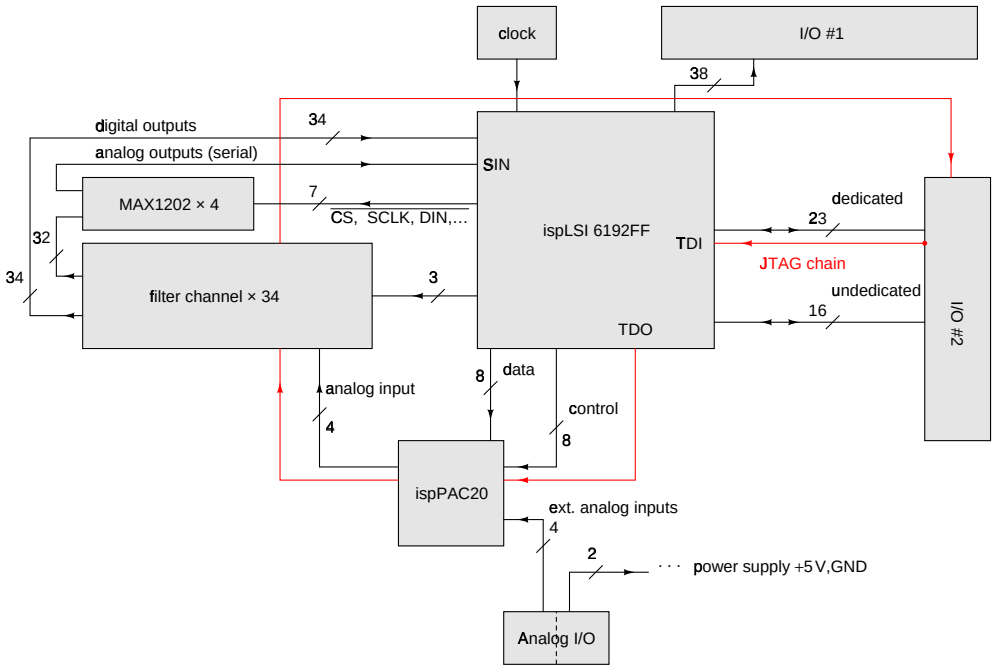


Figure 2.3: Biosonar Signal Processor: Block diagram of the frontend filterbank.

Chapter 3

Hardware

3.1 Jumpered power supply and input configuration

Figure 3.2 shows jumper settings on the biosonar board which are the only configuration options on the board that cannot be set remotely. In addition to the jumpers, the figure shows two of the three input connectors (the 50-pin digital I/O header socket is to the right of the figure and is not shown). All pins are pitched at 0.1 in spacing. The connector labeled "Power" and "Ground" is the main power input. The connector labeled "analog in" is the sonar signal input (microphone) when analog input is selected. Power supply options are selected by jumper $J1$, while signal input options are selected by jumpers $J2$, $J3$, and $J4$, and also by the programmed setting of the preamplifier. Jumper configurations are described below.

Power modes are shown in Table 3.1 and described below:

J1	LM7805	Mode
top	uninstalled	A
top	installed	A
bottom	installed	B

Table 3.1: Jumper configuration for input modes. 'top' indicates that the two pins closest to the top of the board are connected together, and the bottom pin is unconnected. 'bottom' indicates that the bottom two pins are connected together and the top pin is unconnected.

A Connect the top two pins of $J1$ for unregulated input. For this setting, power must be supplied by a regulated power supply ('unregulated' means that the biosonar board is not doing the regulation). 'Power' must be 5 V and 'Ground' must be 0 V. The supply must be able to deliver a sustained DC current of at least 1.2 A at 5 V. Because most of the chips contain continuous-time analog circuits, power consumption is fairly constant.

B Connect the bottom two pins of $J1$ for regulated input. For this setting, an LM7805 5 V regulator must be installed in the indicated position on the board. 'Power' must clear the dropout voltage of the regulator, which usually means about 8 V.

Input modes are shown in Table 3.1 and described below. Figure 3.1 shows the bias circuit between the input pins and the preamplifier.

A Differential signal goes directly from input pins to the 'In1' input of the preamplifer, buffered through 5 kΩ resistors. Differential input must be biased around 2.5 V.

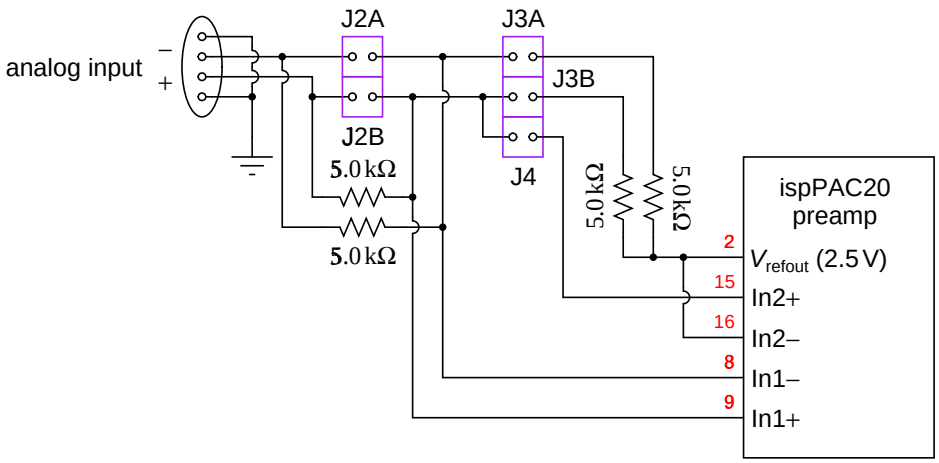


Figure 3.1: Analog input circuit and configuration jumpers.

J2A	J2B	J3A	J3B	J4	Mode
					A
X	X				B
		X	X		C
	X			X	D
				X	E
			X	X	F
X	X	X		X	G
X		X		X	H
X		X	X	X	J

Table 3.2: Jumper configuration for input modes. An ‘X’ indicates that a jumper bridges the two pins (closed connection). No marking indicates an open connection.

B Differential signal goes directly from input pins to the ‘In1’ input of the preamplifier with no buffering of any kind. Differential input must be biased around 2.5 V.

C Differential signal goes from input pins to the ‘In1’ input of the preamplifier. Signal is rebiased to operate at or close to the preamplifer’s reference voltage of 2.5 V.

D Single-ended signal goes directly from input ‘+’ pin to the ‘In2+’ input of the preamplifier, with no buffering. The single-ended input is assumed to be biased to always remain above ground (preferably around 2.5 V).

E Single-ended signal goes from input ‘+’ pin to the ‘In2+’ input of the preamplifier, buffered by a 5 kΩ resistor. Bias requirements are the same as mode ‘D’ above.

F Single-ended signal goes from input ‘+’ pin to the ‘In2+’ input of the preamplifier. Signal is rebiased to operate at or closed to the preamplifier’s reference voltage of 2.5 V.

G Like mode ‘D’ above, but input ‘−’ pin is held at the reference voltage.

H Like mode ’E’ above, but input ‘−’ pin is held at the reference voltage.

J Like mode ’F’ above, but input ‘−’ pin is held at the reference voltage.

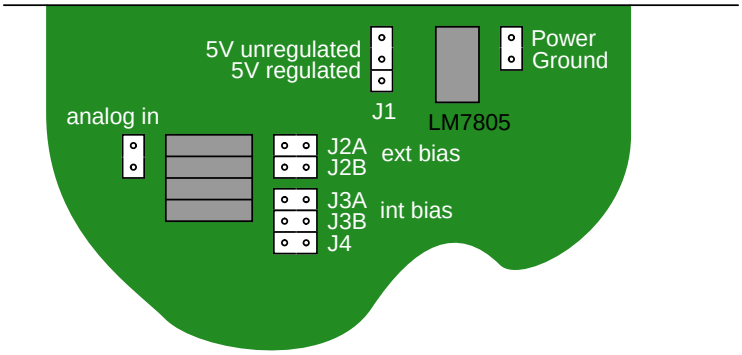


Figure 3.2: Biosonar Signal Processor: Frontend board configuration jumpers.

Chapter 4

Firmware

Much of the BioSonar system's novelty lies in its ability to reconfigure its architecture through nonvolatile, re-writable configuration memory in the filter and interface chips. For the analog chips, filter frequency, filter resonance (Q), and filter type (lowpass, bandpass, highpass) are all programmable, and inputs may be selected from a number of different sources. In addition, the ispPAC20 features two comparators with selectable inputs, and an 8-bit DAC. The ispLSI6192-FF is a full-featured FPGA with general-purpose logic resources. The signal routing on the BioSonar printed circuit board is designed to allow several large-scale filterbank architectures, as well as letting the FPGA handle all routing between the board and any external computer or back-end processor.

4.1 The ISP JTAG protocol

Four I/O pins on the 50-pin connector are dedicated to the ISP JTAG chain. JTAG is a protocol designed to allow boundary scans of programmable chips. Lattice Semiconductor makes dual use of the JTAG protocol to do both the boundary scan and also handle device configuration on the ispLSI-series chips. The ispPAC analog chips have no equivalent boundary scan capability, but they use the JTAG protocol for device configuration.

The BioSonar system is intended to hide the details of the JTAG protocol from the end-user; these protocols are embedded in the software but the end-user is intended to use higher-level calls to erase, program, and verify the devices. However, some details of the JTAG architecture elucidate the use of the software subroutines. JTAG is a serial protocol, requiring for each chip two data bit lines (*TDI* and *TDO*), plus one mode bit (*TMS*) and one clock signal (*TCK*). A well-defined state machine, called the TAP controller and shown in Figure 4.1, allows 5-bit commands, some of which are defined by the protocol, such as "BYPASS" to form a one-bit link between TDI and TDO which bypasses the chip. Lattice defines other commands such as "ERASE," "PROGRAM," and "VERIFY." All chips on the board form one long serial JTAG chain by linking TDO of one chip to TDI of the next chip in the chain, and supplying TMS and TCK to all chips in parallel. The chain, in order, is shown in red in Figure 4.2.

While it is convenient to think of the JTAG chain as a single, very long configuration word, in reality the ispLSI chip must be programmed differently, by supplying 180 separate bit streams, addressed by a different bit. So the actual programming for the board is done in two stages, one for the ispLSI, and one for all of the ispPAC chips. During each stage, the set of chips not being accessed are placed into BYPASS mode.

In addition to being programmed in a different manner, the ispLSI chip is a general-purpose FPGA which requires complicated placement and routing to program. Because the placement and routing algorithms, and the purpose of each configuration bit is Lattice proprietary information, the ispLSI chip must be designed in Lattice software, outside of the BioSonar system, producing a "JDEC" file containing the configuration bit stream for the ispLSI. The purpose of each configuration bit in the ispPAC chips is known, so

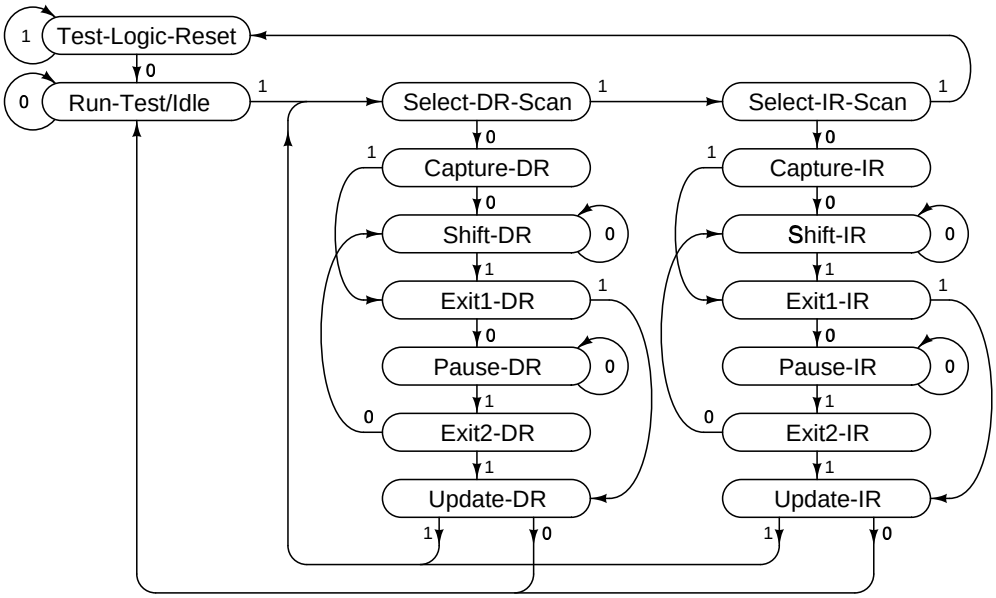


Figure 4.1: The JTAG Test Access Port ("TAP") state machine. Bit values shown represent the state of signal TMS at the time of a rising edge of signal TCK.

all configuration can be determined on the fly by the BioSonar system software (see the section on software).

4.2 The ispLSI-6192FF I/O Interface

The ispLSI-6192FF device is a 207-pin surface-mount device, running at 5 V, containing nonvolatile EEPROM configuration memory, and implementing a JTAG interface compatible with the ispPAC devices. On the biosonar board, the ispLSI chip has been mounted on top of a surface-mount-to-PGA converter board, and plugged into a 17×17 PGA socket.

Unfortunately, Lattice Semiconductor has discontinued the ispLSI-6192FF device (the entire 6000 series has been discontinued), and even more unfortunately, they have made the poor decision to discontinue software support for the device, as well. Although the Lattice Semiconductor software is freely distributed, recent versions of the program will not compile schematics for the 6192 device. The only option is to continue to use an older version of the Lattice "ispDesignExpert" software (version 8.0). A copy of this software version is included with the software package for the BioSonar frontend board. A software license is freely available from Lattice Semiconductor (`http://www.latticesemi.com`), but must be obtained before the software can be used.

In the provided software, the schematic layout corresponding to the latest BioSonar board revision can be found in directory `Lattice\Biosonar\`. The "project" is the file `biosonar.syn` and is configured to launch the ispDesign Expert Project Manager program upon selection.

The schematic for the BioSonar interface (assuming a FIFO board) is drawn in Figure 4.4. Altering the interface requires understanding the interface, so a brief description follows.

Two main circuits in the schematic, FIFO4 and T4R4CPV, are hardwired circuits on the ispLSI6192FF. The T4R4CPV is one of several possible configurations of the on-board register bank. This particular configuration treats the register bank as four 16-bit up-down counters and four 16-bit registers. Each register/counter has its own clock signal, and each counter provides a "terminal count" output (BnTC). Three "select" pins (BS0–BS2) select which register receives input from bus DI and applies its value to output bus

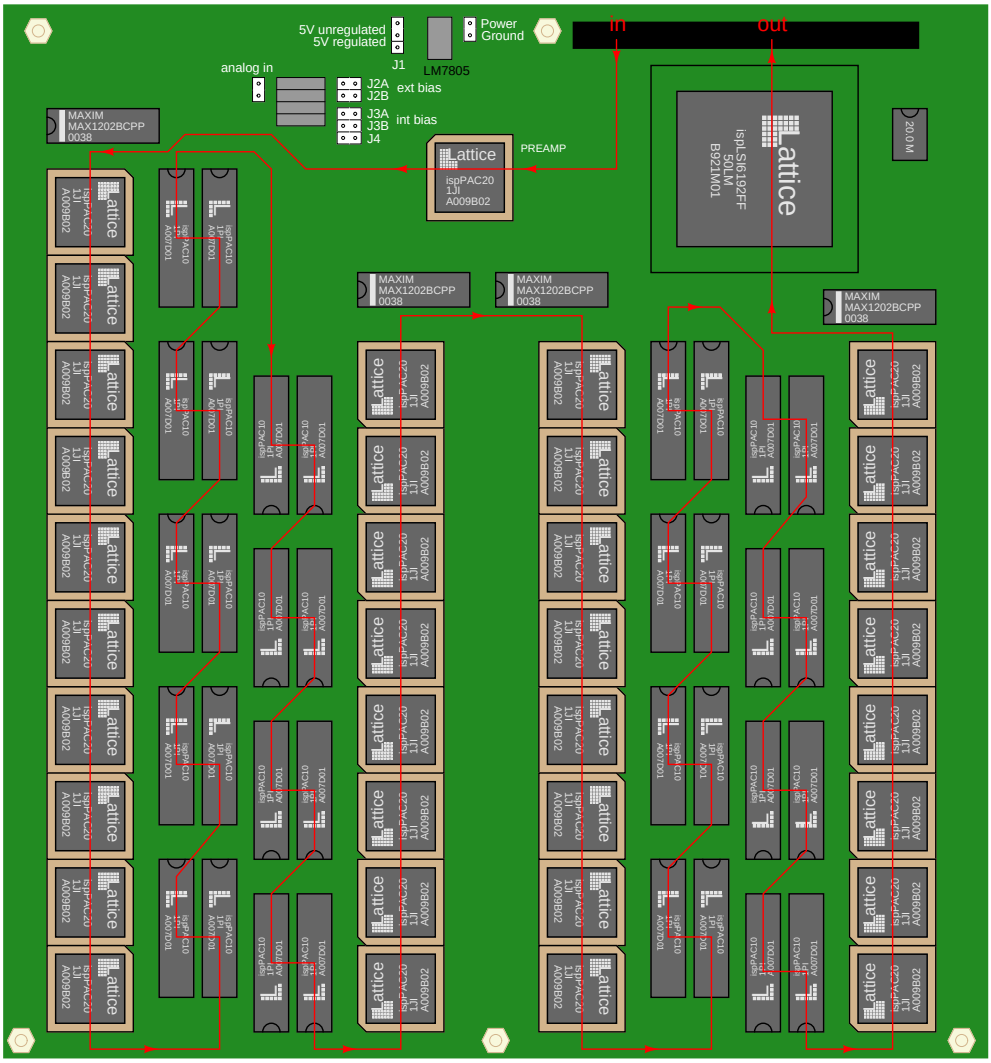


Figure 4.2: The JTAG serial chain links every chip on the board into one long serial bit stream.

DO. The T4R4CPV defines a preset bit sequence for each counter which is loaded on "enable" (BnPLEN) for that counter ($n = \{1, 3, 5, 7\}$). The preset value can be set by editing the T4R4CPV schematic and changing the appropriate inputs. Figure 4.5 shows how the preset values are specified in the ispDesignExpert software: Each bit is determined by the connection of the preset value block's input to either Vdd or GND, to denote a preset value of 1 or 0. No other schematic connections to the inputs are valid. Similarly, the CAOCTRL block indicates the polarity of the carry-out bit of the counter, and the POLCTRL block indicates the polarity of the EN enable bit of the T4R4CPV module.

In the BioSonar system application, the four registers of the T4R4CPV are not used, and the counters are used to divide down the input 20 MHz clock into several different frequencies required by the BioSonar system. Note in particular that the BioSonar board's input and output operate independently; therefore, input- and output-handling circuits are clocked at separate rates. The circuit uses two of the 5 dedicated clock inputs on the ispLSI6192FF for these two rates. The slower clock input is generated by the ispLSI6192 circuit. However, because the ispLSI clock networks are fixed, it is necessary to put the divided-down clock (signal CLK_OUT) on an ispLSI6192 output pin, and route the output back to the second clock input via the printed circuit board. Each counters "terminal count" output is fed directly back to the "enable" input,

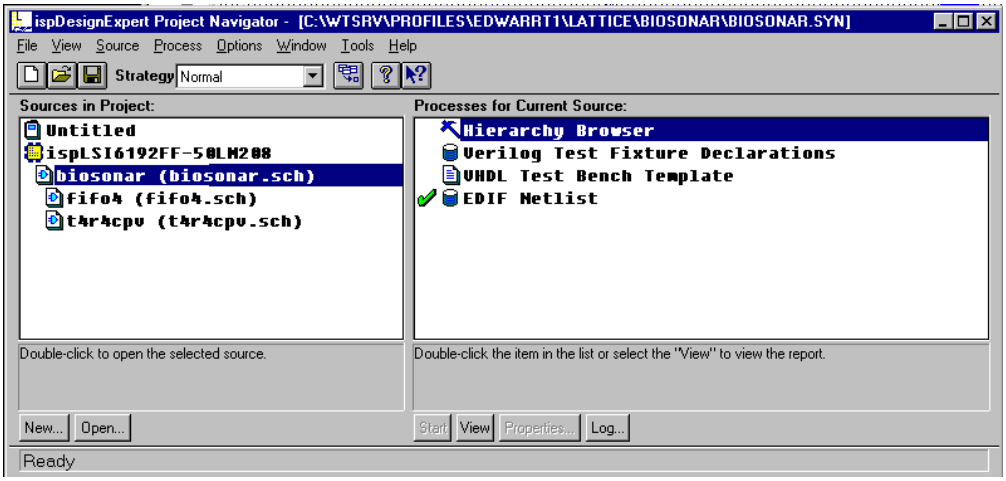


Figure 4.3: ispDesignExpert project manager window.

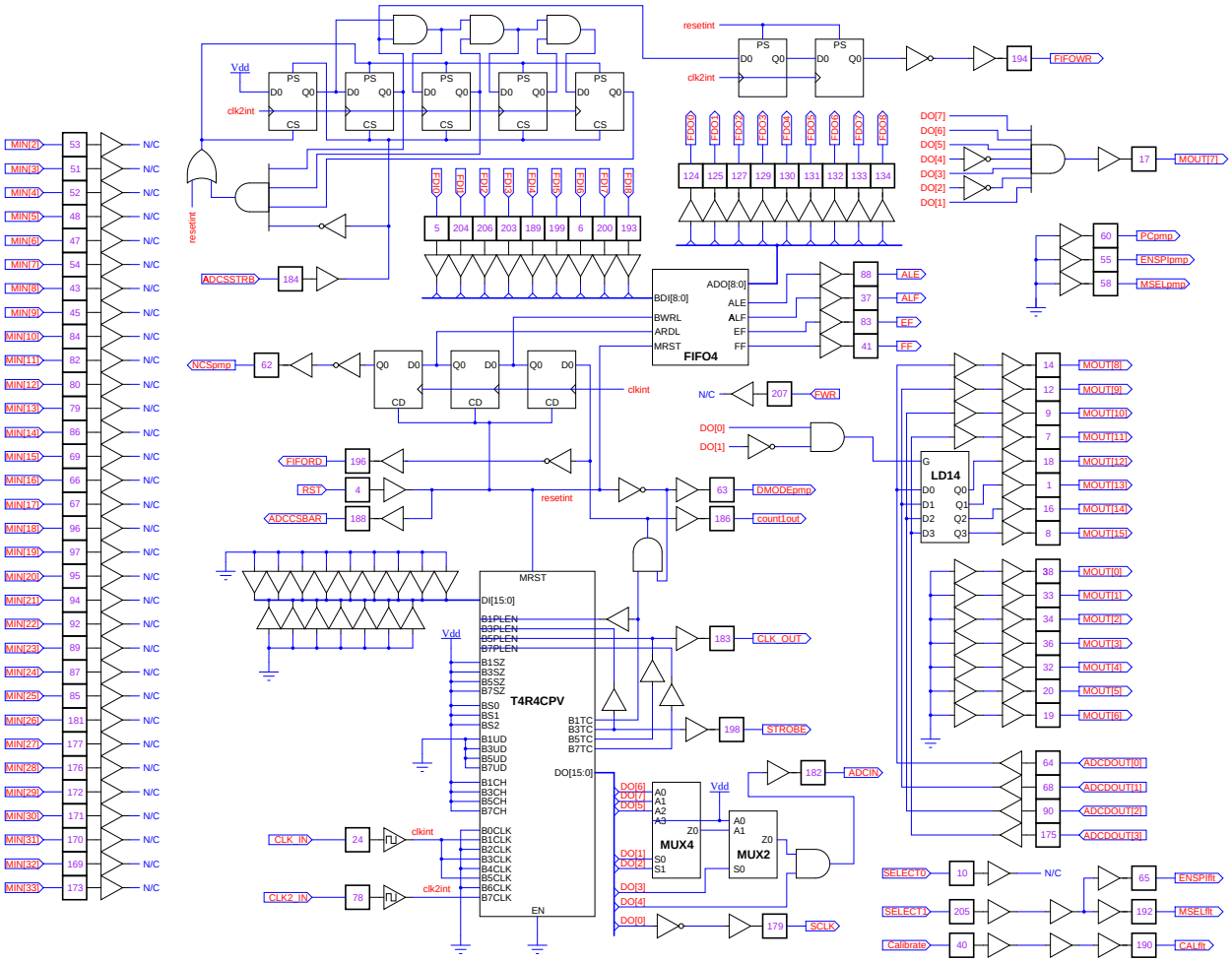


Figure 4.4: Schematic drawing of the interface circuit. This schematic corresponds to the JEDEC file "**`expt.jed`**".

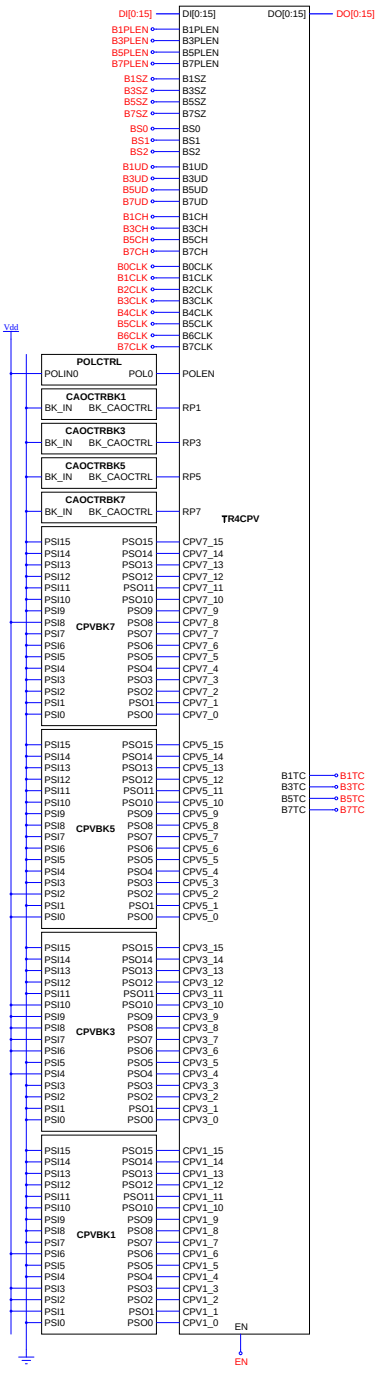


Figure 4.5: Schematic of the T4R4CPV register/counter module, showing register preset value programming.

forming a continuous output. The "select" pins are grounded to enable the continuous monitoring of the value of the first counter on the DO output bus. Mux logic generates various internal and external event signals based on the value of the primary counter's instantaneous value.

The preset values for the four counters, as indicated in Figure 4.5, are:

counter	bit value	hex value	decimal value	time	frequency
1	0000000001001110	0x004e	78	3.9 µs	256 kHz
3	0000011111010000	0x07d0	2000	100 µs	10 kHz
5	0000000000000101	0x0005	5	0.25 µs	4 MHz
7	0000000100000000	0x0100	256	64 µs	15.625 kHz

Table 4.1: Default timer values programmed into the interface. Time and frequency values are measured assuming an input master clock rate of 20 MHz for counters 1, 3, and 5, and assuming that the output of counter 5 becomes the input to counter 7 (connection must be made off-chip).

The FIFO module is effectively replaced by the external bidirectional FIFO board. The arrangement of the BioSonar board requires that the FIFO be put in the "FIFO4" configuration, in which it acts as an input buffer between the I/O connector and the preamplifier's DAC inputs, and has a size of 9-bit words $\times$ 8192 locations. Because the FIFO outputs are hard-wired to specific pins on the ispLSI6192, the FIFO cannot be removed from the system. For use with the external FIFO board, however, it must be bypassed. This is accomplished with a timed sequence of signals to the FIFO's "read" (ARDL) and "write" (BWRL) inputs, ensuring that the FIFO4 module does exactly one write and one read operation in sequence for every input retrieved from the external FIFO board. The external board is queried by the FIFORD signal.

In addition to clock frequency division and input retrieval, the ispLSI6192 interface is responsible for collecting outputs from the BioSonar frontend. There are two separate outputs available from each filterbank channel. One is a single-bit digital output, and one is an analog value. The nature of the output depends on the configuration of the analog filterbank chips (see below). Each of the 34 digital outputs is connected directly to ispLSI6192 inputs. The present schematic does not make use of the digital outputs, so they are shown as not connected. The analog outputs feed to four Maxim MAX1202 ADC chips, each with eight multiplexed inputs (so there are 32 analog outputs available for 34 channels; the analog output of the two highest-frequency channels cannot be accessed, although the digital output is available). The ispLSI6192 is responsible for generating the signals that driver the Maxim chips. ADCIN is the serial bit stream containing configuration information for the ADC chip. SCLK clocks the ADC. The ADC configuration is an 8-bit word, described in the Maxim MAX1202 data sheet, and summarized in Table 4.2:

Bit	Name	Description
7	START	First bit defines beginning of control byte.
6 5 4	SEL2 SEL1 SEL0	Select which channel is read. Numerically, Channel (0–7) = Sel1 (MSB), Sel0, Sel2 (LSB)
3	UNI/$\overline{\text{BP}}$	1 = unipolar (always)
2	SGL/$\overline{\text{DIF}}$	1 = single ended (always)
1 0	PD1 PD0	Power-down mode and clock select PD1 = 1, PD0 = 1 External clock mode (always)

Table 4.2: Bit values of the Maxim MAX1202 configuration word.

All ADC configuration bits are '1' except for the channel selection. The Mux logic provided by the "MUX4" and "MUX2" gates inserts the proper channel number such that each channel is accessed in se-

quence.

ADCSSTRB indicates a completed ADC output, and is used in this schematic to drive the output. FIFOWR is derived directly from the ADCSSTRB signal. The ADC output is 12 bits long and read out serially. Each of the four ADC chips provides a separate output bit stream. The ADC clock (SCLK) has a maximum rate of 2 MHz and the time for an ADC conversion is 16 clocks (the minimum possible time for a conversion is 15 clocks, but the convenience of generating signals based on a cycle of 16 outweighs the advantage of a slightly faster conversion). So the minimum possible time to capture all 32 analog outputs is

$$\frac{1}{2\,\mathrm{MHz}} \times 16\,\mathrm{cycles} \times 8\,\mathrm{channels} = 64\,\mu\mathrm{s}$$

which is an output rate of 15.625 kHz. No attempt should be made to set the output rate (counter) to a cycle frequency greater than this number. It represents the maximum rate at which analog data can be extracted from the BioSonar board. This is the default rate programmed into the output counter as shown in Table 4.1.

Each cycle of output sampling produces $12\,\mathrm{bits} \times 32\,\mathrm{channels} = 384\,\mathrm{bits}$ output. However, in this circuit instantiation, output is generated at the fastest rate using the simplest possible method, in which no attempt is made to organize the output into meaningful words. Instead, the 8 lowest bits of the 9-bit output FIFO are loaded 4 bits at a time with the 4 ADC outputs. The FIFO is written every two clock cycles of the ADC clock. On the first ADC clock, the ADC output is latched into the upper four bits of the FIFO word. On the second ADC clock, the ADC output is directed to the lower four bits of the FIFO word, and all 8 bits are written into the FIFO. After all 12 bits are loaded into the FIFO, the ADC receives configuration information for the next input channel, and the cycle repeats until all 8 channels have been read, converted, and transferred to the output FIFO. In the current instantiation, it is the reponsibility of the back-end system to translate this bit block back into meaningful values, and, if necessary, sort the channels into their sequence in the filterbank.

Because the FIFO has 9 bits, the 9th bit is used to generate a synchronization bit which indicates to the back-end system that a block of data is beginning. This synchronization bit is sent coincident with the first byte written to the FIFO.

On the schematic drawing, each input or output pad contains the number corresponding to the ispLSI pin to which it is attached. Pin numbers and signal names are fixed according to the BioSonar printed circuit board. However, internal signals and all routing are determined by the Lattice ispDesignExpert placement and routing algorithms.

The goal of compiling the circuit in ispDesignExpert is to generate a "JDEC" file containing the configuration bits for the ispLSI6192FF. This file, which has the extension '`.jed`', is transferred from the Windows machine used to run the ispDesignExpert software to the Linux machine running the BioSonar board application (see below). The application program accepts the file exactly as written by ispDesignExpert.

Tables 4.3 through 4.6 give a complete list of pins on the ispLSI6192FF, their signal names as relevant to the BioSonar board, and a description of each signal. The first pin number is for the ispLSI6192FF quad flat pack package, for use in assigning signals to pins in the ispDesignExpert program. The second pin number is the corresponding pin grid array (PGA) pin number, for the purpose of probing signals on the BioSonar frontend circuit board. DIO pin numbers are for the CIO-DIO96 or PCI-DIO96 50-pin connectors, with names corresponding to the BioSonar system software (CIO-DIO96 driver software and **filterbank** application program).

All of the signal names in the table correspond to signal names on the schematic, except for signals which are hardwired to the ispLSI6192FF for the purpose of JTAG programming and are therefore not available as inputs and outputs. These include TDI, TDO, TCK, TMS, BSCAN, and !RST. Pins labeled Y1 and Y2 are dedicated clock network inputs, and pins labeled FIFO_n are dedicated FIFO outputs. Because the FIFO pins are hardwired to the preamplifier DAC inputs on the BioSonar circuit board, the FIFO is required to operate

pin	PGA	ispLSI name	signal name	description
8	C1	I/O_85	MOUT[15]	DIO (pin 25)
16	F2	I/O_91	MOUT[14]	DIO (pin 26)
1	D3	I/O_79	MOUT[13]	DIO (pin 27)
18	F1	I/O_93	MOUT[12]	DIO (pin 28)
7	E4	I/O_84	MOUT[11]	DIO (pin 29)
9	F3	I/O_86	MOUT[10]	DIO (pin 30)
12	F4	I/O_88	MOUT[9]	DIO (pin 31)
14	G3	I/O_89	MOUT[8]	DIO (pin 32)
17	G4	I/O_92	MOUT[7]	DIO (pin 33)
19	H3	I/O_94	MOUT[6]	DIO (pin 34)
20	G2	I/O_95	MOUT[5]	DIO (pin 35)
32	K2	I/O_0	MOUT[4]	DIO (pin 36)
36	L4	I/O_3	MOUT[3]	DIO (pin 37)
34	L3	I/O_2	MOUT[2]	DIO (pin 38)
33	L1	I/O_1	MOUT[1]	DIO (pin 39)
38	M2	I/O_5	MOUT[0]	DIO (pin 40)
200	C5	I/O_72	FDI7	DIO (pin 9)
6	C2	I/O_83	FDI6	DIO (pin 10)
199	A4	I/O_71	FDI5	DIO (pin 11)
189	A7	I/O_64	FDI4	DIO (pin 12)
203	A3	I/O_74	FDI3	DIO (pin 13)
206	A2	I/O_77	FDI2	DIO (pin 14)
204	B3	I/O_75	FDI1	DIO (pin 15)
5	B1	I/O_82	FDI0	DIO (pin 16)
88	P11	I/O_40	ALE	DIO (pin 41)
37	M1	I/O_4	ALF	DIO (pin 42)
83	P10	I/O_35	FE	DIO (pin 43)
41	M4	I/O_7	FF	DIO (pin 44)
196	A5	I/O_69	FIFORD	DIO (pin 5)
194	B6	I/O_68	FIFOWR	DIO (pin 6)
136	H16	TDO	TDO (JTAG out)	DIO (pin 4)
29	K3	TMS	TMS	DIO (pin 18)
28	J2	TCLK	TCK	DIO (pin 46)
99	S15	!RST	!TRST	DIO (pin 23)
26	J4	BSCAN	BSCAN	DIO (pin 24)
72	R7	TOE	TOE	DIO (pin 48)
198	B5	I/O_70	STROBE	DIO (pin 8)
205	C4	I/O_76	SELECT1	DIO (pin 19)
10	D2	I/O_87	SELECT0	DIO (pin 20)
207	D4	I/O_78	FWR	DIO (pin 21)
4	E3	I/O_81	RST	DIO (pin 22)
40	N1	I/O_6	Calibrate	DIO (pin 45)

Table 4.3: Signals connecting the ispLSI6192FF interface chip to the digital I/O (DIO) 50-pin connector, and their corresponding pin numbers.

pin	PGA	ispLSI name	signal name	description
193	A6	I/O_67	FDI8	DIO (B) (pin 48)
186	A8	I/O_61	count1Out	DIO (B) (pin 46)
183	A9	I/O_59	CLK_OUT	DIO (B) (pin 44)
179	B10	I/O_56	SCLK	DIO (B) (pin 42)

Table 4.4: Signals connecting the ispLSI6192FF interface chip to the auxiliary I/O (DIO) area, and the corresponding pin numbers. These pins are expected to be used to probe diagnostic signals; no physical header is connected to the board in this area.

pin	PGA	ispLSI name	signal name	description
27	J1	TDI	TDI	JTAG input from last filter
64	P6	I/O_26	ADCDOUT[0]	ADC #1 (pin15)
68	R6	I/O_30	ADCDOUT[1]	ADC #2 (pin15)
90	R12	I/O_42	ADCDOUT[2]	ADC #3 (pin15)
175	C10	I/O_53	ADCDOUT[3]	ADC #4 (pin15)
188	B8	I/O_63	ADCCSBAR	!CS to all ADCs (pin 18)
184	B9	I/O_60	ADCSSTRB	SSTRB from all ADCs (pin 16)
187	D8	I/O_62	SCLK	SCLK to all ADCs (pin 19)
182	D9	I/O_58	ADCIN	DIN to all ADCs (pin 17)
24	J3	Y1	CLK_IN	clock chip output
78	P9	Y2	CLK2_IN	connected to CLK_OUT
190	C7	I/O_65	CALflt	all ispPAC10 and 20 Calibrate
192	D7	I/O_66	MSELflt	filter ispPAC20 mux select
65	R5	I/O_27	ENSPIflt	filter ispPAC10 and 20 ENSPI
55	S1	I/O_19	ENSPIpmp	preamp ENSPI (pin 4)
58	R3	I/O_21	MSELpmp	preamp MSEL (pin 5)
60	S3	I/O_22	PCpmp	preamp PC (pin 21)
62	R4	I/O_24	NCSpmp	preamp !CS (pin 22)
63	S4	I/O_25	DMODEpmp	preamp DMODE (pin 24)
124	L16	FIFO_9	FDO0	preamp DAC data in lsb (pin 32)
125	L17	FIFO_10	FDO1	preamp DAC data in (pin 33)
127	K16	FIFO_11	FDO2	preamp DAC data in (pin 34)
129	K17	FIFO_12	FDO3	preamp DAC data in (pin 35)
130	J14	FIFO_13	FDO4	preamp DAC data in (pin 36)
131	J17	FIFO_14	FDO5	preamp DAC data in (pin 37)
132	J16	FIFO_15	FDO6	preamp DAC data in (pin 38)
133	H15	FIFO_16	FDO7	preamp DAC data in msb (pin 39)
134	H17	FIFO_17	(unconnected)	
44	N3	I/O_10	(unconnected)	
49	P3	I/O_14	(unconnected)	
201	B4	I/O_73	(unconnected)	
15	E1	I/O_90	(unconnected)	

Table 4.5: Signals connecting the ispLSI6192FF interface chip to the rest of the BioSonar frontend board, and their corresponding pin numbers.

pin	PGA	ispLSI name	signal name	description
61	Q6	I/O_23	MIN[1]	filter channel 1 digital output
56	Q5	I/O_20	MIN[2]	filter channel 2 digital output
53	Q4	I/O_17	MIN[3]	filter channel 3 digital output
51	P4	I/O_15	MIN[4]	filter channel 4 digital output
52	R2	I/O_16	MIN[5]	filter channel 5 digital output
48	Q2	I/O_13	MIN[6]	filter channel 6 digital output
47	Q1	I/O_12	MIN[7]	filter channel 7 digital output
54	Q3	I/O_18	MIN[8]	filter channel 8 digital output
43	P1	I/O_9	MIN[9]	filter channel 9 digital output
45	P2	I/O_11	MIN[10]	filter channel 10 digital output
84	R10	I/O_36	MIN[11]	filter channel 11 digital output
82	S10	I/O_34	MIN[12]	filter channel 12 digital output
80	R9	I/O_33	MIN[13]	filter channel 13 digital output
79	S9	I/O_32	MIN[14]	filter channel 14 digital output
86	Q11	I/O_38	MIN[15]	filter channel 15 digital output
69	P7	I/O_31	MIN[16]	filter channel 16 digital output
66	Q7	I/O_28	MIN[17]	filter channel 17 digital output
67	S5	I/O_29	MIN[18]	filter channel 18 digital output
96	Q13	I/O_46	MIN[19]	filter channel 19 digital output
97	R14	I/O_47	MIN[20]	filter channel 20 digital output
95	S14	I/O_45	MIN[21]	filter channel 21 digital output
94	R13	I/O_44	MIN[22]	filter channel 22 digital output
92	S13	I/O_43	MIN[23]	filter channel 23 digital output
89	S12	I/O_41	MIN[24]	filter channel 24 digital output
87	R11	I/O_39	MIN[25]	filter channel 25 digital output
85	S11	I/O_37	MIN[26]	filter channel 26 digital output
181	A10	I/O_57	MIN[27]	filter channel 27 digital output
177	A11	I/O_55	MIN[28]	filter channel 28 digital output
176	B11	I/O_54	MIN[29]	filter channel 29 digital output
172	B12	I/O_51	MIN[30]	filter channel 30 digital output
171	A13	I/O_50	MIN[31]	filter channel 31 digital output
170	C11	I/O_49	MIN[32]	filter channel 32 digital output
169	B13	I/O_48	MIN[33]	filter channel 33 digital output
173	D11	I/O_52	MIN[34]	filter channel 34 digital output

Table 4.6: Signals connecting the ispLSI6192FF interface chip to the filterbank channel single-bit digital outputs, and their corresponding pin numbers.

in the mode in which these pins are used as FIFO outputs, not inputs. The pin marked “(unused)” was not required by the schematic, although it is available for general purpose output. Pins marked “(unconnected)” are the remaining general-purpose I/O on the ispLSI6192FF. These pins are unconnected due to board layout errors and cuts made for rewiring, and should not be used.

4.3 The DIO Interface

From the standpoint of any back-end hardware, the DIO interface is simply a collection of pins, most of which can be programmed either as input or output on the ispLSI-6192FF for any purpose whatsoever, but with a few pins reserved for use by the JTAG programming. Figure 4.6 shows the 50-pin connector. For use with arbitrary back-end hardware, Figure 4.6 (**a**) highlights which pins (corresponding to Figure 4.3) are available for general-purpose I/O through the ispLSI-6192 interface, and which pins are reserved for the JTAG interface and for power, and which pins have no connections.

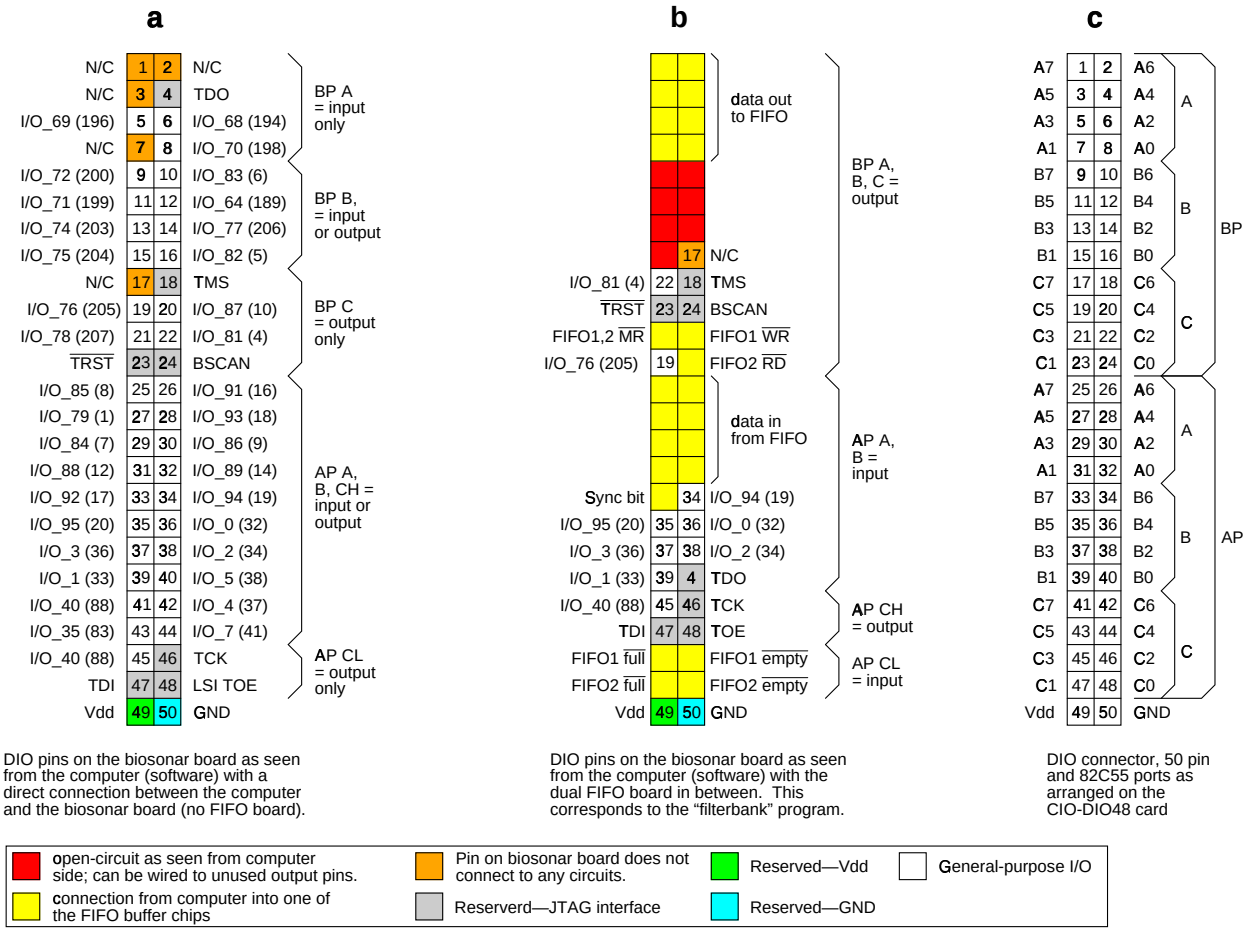


Figure 4.6: The DIO 50-pin interface (generic pin names).

On the computer end of the system, the software driver program accesses the BioSonar board through the 50-pin connector at the input of the dual FIFO board. When the dual FIFO board is connected between the computer and the BioSonar board, many interface pins become reserved for use by the two FIFO chips (data lines and the FIFO read/write controls). Some of the remaining interface lines simply pass through the FIFO board unchanged. However, other lines are rearranged as they pass through the FIFO board to accomodate the vagaries of the 82C55 ports on the CIO-DIO48 interface to the computer. Figure 4.6 (**b**) shows what signals and ispLSI-6192FF general-purpose I/O pins can be addressed from software with the FIFO board in place. The numbering of the pins in this figure match the pin numbers on the BioSonar board, showing how the pins are rearranged by the wire-wrapped jumpers on the FIFO board. The software does not address the interface by pin number, however, but by reference to the two 82C55 interface chips and their ports (A, B, and C, 8 bits each). Figure 4.6 (**c**) maps the 82C55 ports of the CIO-DIO48 interface

to the 50-pin connector. This mapping is reproduced from the Measurement Computing, Inc. (formerly ComputerBoards, Inc.) *PCI-DIO96 Digital Input/Output User's Manual* (Rev. 2, Nov. 2000). The 82C55 ports are grouped in bytes and each byte must be configured in software to be (exclusively) an input port or an output port, except for port C which is split into high and low nybbles CH and CL, respectively, each of which may be selected independently for input or output. Port C can be addressed bit-wise using internal 82C55 commands, making this the proper port to use for output control signals, which must be prohibited from glitching. Ports A and B are more appropriate for data buses, which can be addressed as a single byte or word, and input signals, where glitching is not a problem. These considerations lead to the grouping of the JDEC signals at positions corresponding to port C of the 82C55 devices, similarly for FIFO read/write lines, and the grouping of FIFO data buses to fit in a single port A or B. To the right of each connector in the drawing is marked the port and an indication as to whether that port must be configured as an input or an output port, or may be either. Here, "input" and "output" are relative to the computer: "input" signals are produced by the BioSonar frontend or FIFO board and captured by the computer; "output" signals are produced by the computer and captured by the FIFO board or BioSonar frontend board. The insertion of the FIFO board constrains all of the ports, and therefore constrains the direction of pin signals on all of the accessible I/O pins on the ispLSI6192 interface FPGA.

Figure 4.7 shows the same view as Figure 4.6, except with all the general-purpose I/O labeled according to their designation in the schematic (Figure 4.4) for use with the current instantiation of the JDEC program for the ispLSI-6192 (file **`expt.jed`**), the bidirectional FIFO board, and the application program **filterbank**. Figure 4.7 (**b**) therefore is the view of the BioSonar system as seen by the **filterbank** software. The mapping of signal names in Figure 4.7 (**b**) to 82C55 ports in Figure 4.7 (**c**) exactly matches the definitions in the **filterbank** source code (file `defines.h`). The direction ("input" or "output") required on each port matches the modes set by the **filterbank** program.

It is very important to understand the difference between the pin ordering and labeling as seen by the software, and as seen by the hardware. Admittedly, the rearrangement at the FIFO board makes this especially confusing, but this is a result of switching between interface boards with different pin correspondences for the 82C55 chips, which precipitated a wholesale rearrangement of signals. The rearrangement is done as much as possible in the firmware of the ispLSI-6192, but due to the necessity of having a number of hardwired signals, the rest of the rearrangement is done on the dual FIFO board itself.

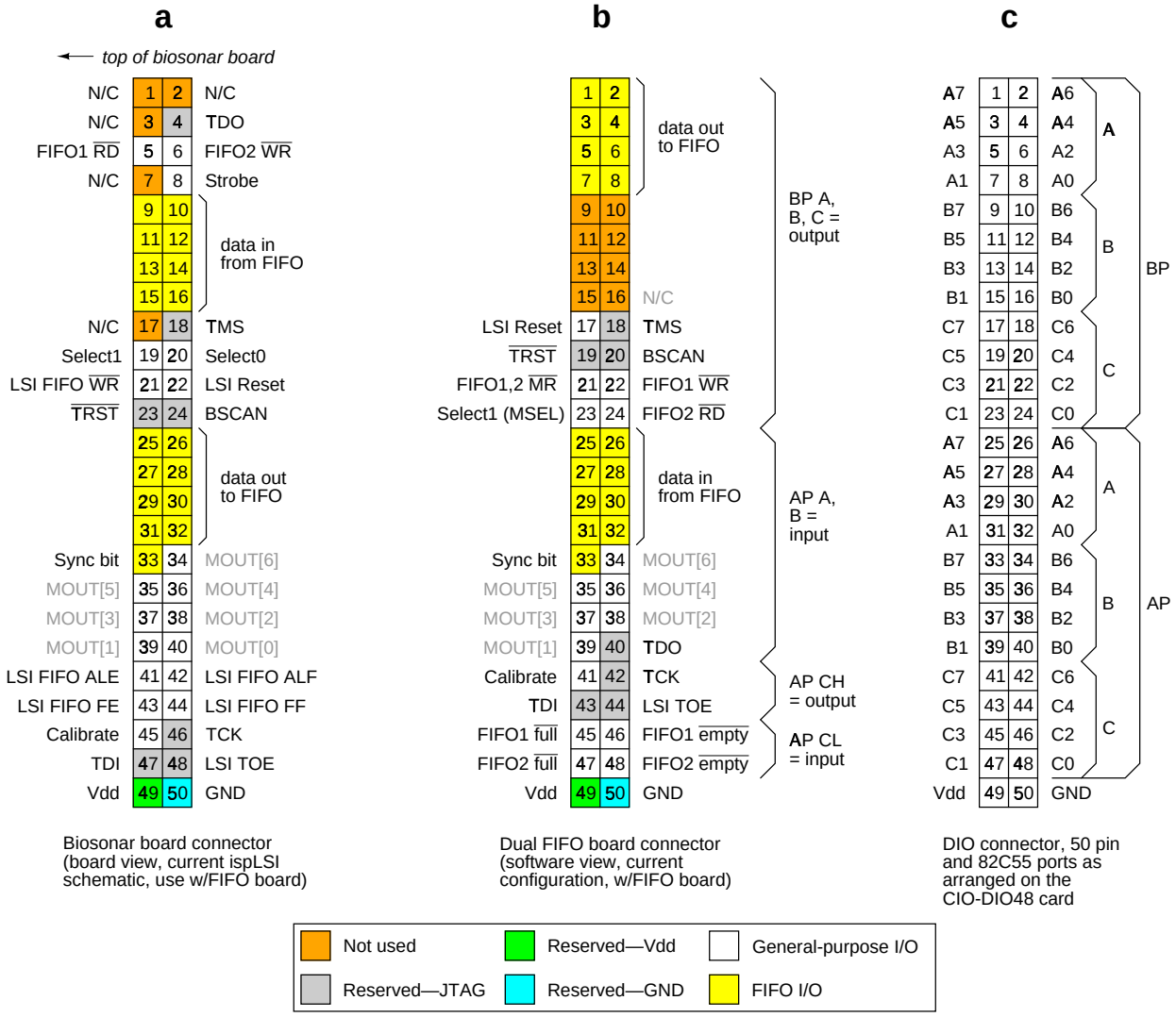


Figure 4.7: The DIO 50-pin interface (signal names specific to the current instantiation of the interface firmware and software).

4.4 The ispPAC-10 filterbank

The filterbank differs from the interface chip in that the configuration bits are known, and all configuration can be determined "on the fly" by the application program. In spite of the ease of programming the various high-level configurations, it is helpful to understand how the frontend printed circuit board is wired and how this affects what can and cannot be programmed into the ispPAC chips' configuration memory.

Internal configuration diagrams in Figures 4.8, 4.10, 4.12, and 4.14 follow the convention of the Lattice PAC-Designer software and the ispPAC chip datasheets. Differential signals are again shown as single wires except where broken out at the input/output pins. Programmable connections are shown in red. Wiring on the printed circuit board, external to the ispPAC chips, is indicated in green.

Figure 4.8 shows the standard implementation of a biquadratic filter in the Lattice ispPAC architecture, using two "PAC blocks." The arrangement produces both the 2nd-order bandpass or the 2nd-order lowpass functions, one at each output, as shown. The exact function produced is

$$\omega_0 = \sqrt{\frac{k_{12}k_{21}}{(C_1 \cdot 250\,\mathrm{k\Omega})(C_2 \cdot 250\,\mathrm{k\Omega})}}$$

with resonance

$$Q = \sqrt{\frac{C_1}{C_2}k_{12}k_{21}}$$

for the bandpass, and unity-gain amplitude

$$A_{\mathrm{DC}} = -\frac{k_{11}}{k_{21}}$$

for the 2nd-order lowpass. See the Lattice Semiconductor application note on biquad filters for details. Both ω_0 and Q are functions not only of the capacitor values, but also of the instrumentation amplifiers' (integer) gains. Knowing the available capacitor values on the Lattice chips, an algorithm which searches for the closest ω_0 and Q values to a given target function over $k_{12} = \{1, 2\}$ and $k_{21} = \{1, 2\}$ will always find a solution within 1% for both parameters.

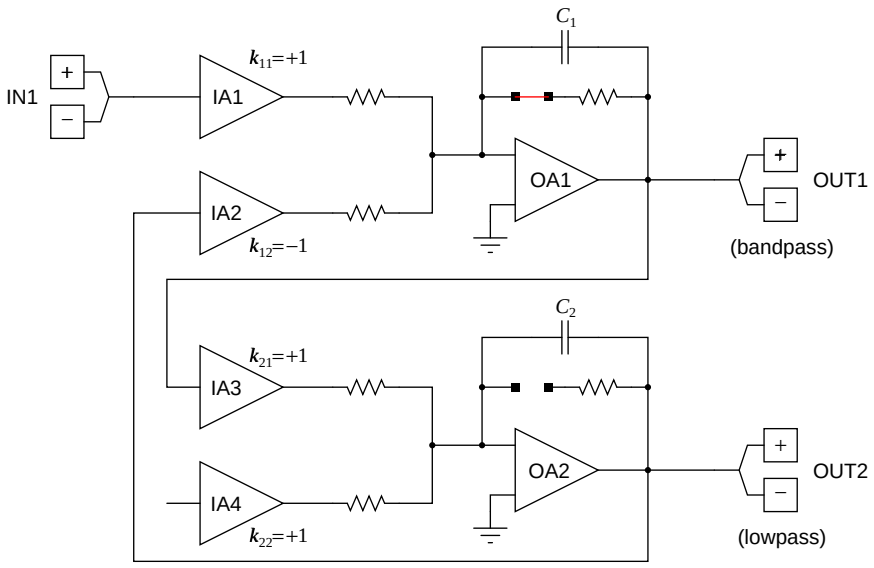


Figure 4.8: ispPAC-10 schematic of a generic biquad filter.

The layout of the BioSonar printed circuit board essentially restricts the use of the ispPAC-10 devices to three main filterbank configurations (others are possible, but unlikely to be of much use):

1. **Parallel**
2. **Cochlear**
3. **Parallel/Cascade**

Each of these configurations is described in detail in the sections below.

4.4.1 Parallel Filterbank Configuration

The primary filterbank configuration, and the only one extensively tested and known to work as advertised, is the parallel bandpass filterbank. In this configuration, each filter channel receives the same input (from the preamplifier output). Each channel is formed from two 2nd-order biquad bandpass filters, in series, each with the same center frequency and same resonance (Q). The series combination of the filters effectively creates a single filter with a 4th-order response, and an effective Q which is the product of the individual filter Q values.

The use of the same value ω_0 and Q for each of the two filters in series is not a necessity, but was done for simplicity. An alternative method using the same configuration might split the frequencies of the two filters, creating a response which is wider but flatter (more uniform through the passband).

4.4.2 Cochlear Filterbank Configuration

The purpose of the cochlear filterbank configuration is to mimic the signal processing of the mammalian cochlea. A simple one-dimensional simplified model of basilar membrane mechanics (described in detail in Carver Mead's *Analog VLSI and Neural Systems* consists of a linear cascade of 2nd-order lowpass filter sections, starting at the high frequency end, with the signal moving through successive filtering operations to the low frequency end. The 2nd-order lowpass function requires a Q which ranges from $1/\sqrt{2} = 0.707$ (maximally flat response) upward. The cumulative effect of cascading the filters is to multiply together the frequency response of each filter between the signal input and the tapped output. Any gain larger than 0 dB causes a "pseudoresonance" which is larger than the Q of any one filter. The size of the pseudoresonance depends on the spacing of the cutoff frequency of each filter. For the pseudoresonance to remain roughly constant over the length of the filterbank, the cutoff frequencies of the filters should be placed on a logarithmic, not linear, spacing.

In this configuration, there are 68 filter sections in the cascade, with an output tap at every other filter output.

4.4.3 Parallel/Cascade Filterbank Configuration

The one-dimensional signal model of basilar membrane mechanics is known to have stability problems, and also lacks a high pass filter function to knock off the lower end of the frequency response (in the above configuration, this can be done in the ispPAC-20 which follows each filter output, but this also restricts the functions which can be implemented in the ispPAC-20). One way to get around the stability problem is to implement the cascade using 1st-order lowpass filters (which are inherently stable, even cascaded in arbitrary numbers), and boost the output at each tap using a bandpass filter, which also knocks out the unity-gain response on the low side of the cutoff frequency. This model retains the advantage from the cochlear model that the high-end cutoff gets multiplied by 6 dB/decade at each stage, creating a very sharp response only a few taps into the filter. The frequency spacing does not need to be on a logarithmic scale. The

important thing is the placement of the bandpass center frequency with respect to the cascade. Because the frequency response of the cascade is the multiplication of the response of each filter going from the tap back to the input, the 3 dB cutoff frequency of the total response is shifted significantly downward with respect to the 3 dB cutoff of the last filter in the cascade before the tap.

This configuration cannot be used as a cochlear model because, to make all three configurations possible, it was necessary to require a single "PAC-block" for the lowpass filter, from which only a 1st-order response can be elicited. The response is actually a first-order cascade, tapped at each segment, with each tap followed by a 2nd-order bandpass function and then followed by one more 1st-order lowpass function. While the simple current implementation sets the cutoff of this final filter as high as possible to have a minimal effect on the signal, it can be used in conjunction with the bandpass filter to create a bandpass response with a slightly steeper cutoff on the high end of the center frequency.

4.4.4 Filterbank Details

Figure 4.15 shows the hardwired connections into the ispPAC-10 and ispPAC-20 which form one of the 34 filterbank channels on the board. All 34 channels are connected in the same way. Pins labeled "1I", "2I", "3I", and "4I" are differential analog inputs, and pins labeled "1O", "2O", "3O", and "4O" are differential analog outputs. Wires are drawn as single lines in keeping with the Lattice PAC-Designer software's notation, but note that each analog signal is differential and carried on two lines. The input labeled "Input" comes from the preamplifier output and is applied to all filters in parallel. All other inputs come from the previous filter and outputs go to the next filter in a serial chain. The sequence of filters in the serial chain is defined by these connections. The numbering of filters is given in Figure 4.16.

The numbering of the filters is unimportant in the parallel configuration, but that it is critical for the cascaded configurations. In addition, cascaded configurations must have the high and low ends of the frequency range in the indicated positions.

4.5 The ispPAC-20 analog signal processors

The filter outputs of each channel (tap) pass to a Lattice ispPAC-20 chip for simple post-processing. The configurability of the ispPAC-20 chip enables numerous useful post-processing functions, including full-wave rectification, half-wave rectification, envelope capture, nearest-neighbor value subtraction, zero crossing detection, thresholding, and offset correction. The channel output can be captured as an analog voltage value converted to a 12-bit word by the Maxim MAX1202 ADC chips, or it can be captured as a single bit (namely, a comparator output, as would be the result of a thresholding or nearest-neighbor comparison), with all 34 channels read in parallel.

Figures 4.10, 4.12, and 4.14 all show the configuration map for the ispPAC-20 chip, in keeping with the notation of the PacDesigner software and the ispPAC-20 data sheet. The mapping shown corresponds to the present instantiation, which is a full-wave rectification and envelope capture producing an analog result.

Full-wave rectification requires the use of one of the comparators and the input multiplexer on the IA4 input amplifier. When the select bit of this multiplexer is logic high, the multiplexer swaps the (differential) inputs into IA4, effectively negating the signal. The ispPAC-20 has a mode called "PC direct," in which this multiplexer is controlled by the output of comparator CP1. By routing the filter output to the positive input of CP1, and connecting the negative input of CP1 to the reference voltage, CP1 computes the sign of the filter output (note that this sign bit can be used directly for zero-crossing information). If the filter output is selected as the input to amplifier IA4, and the CP1 output controls the sign of this input, then IA4 sees a full-wave rectified version of the filter output. The "PAC Block" of which IA4 is a part can then be used as a 1st-order lowpass filter, or both "PAC Blocks" can be used as a 2nd-order lowpass filter, to smooth the rectified signal, producing an estimate of the signal's amplitude envelope. The output of the first

“PAC Block” (OUT1) is the signal supplied to the ADC input on the BioSonar board. If the second “PAC Block” is configured for a 1st-order smoothing filter, then the first “PAC Block” may be used as an envelope comparator by routing the input of amplifier IA1 to IN1, which is the smoothing filter output (OUT2) of the ispPAC-20 in the previous filterbank channel. Then the first “PAC Block” computes the subtraction of the neighboring channel’s amplitude envelope from its own channel’s amplitude envelope. The setup shown in the figures goes one step further by configuring the multiplexer input to IA1 such that the value of the ispPAC-20 pin “MSEL” (signal “MSELflt” in the interface circuit schematic) controls whether the output seen by the ADC is the amplitude envelope of the channel or the difference between the amplitude envelopes of neighboring channels. The interface schematic of Figure 4.4 shows that this control signal is passed from software through the “SELECT1” signal.

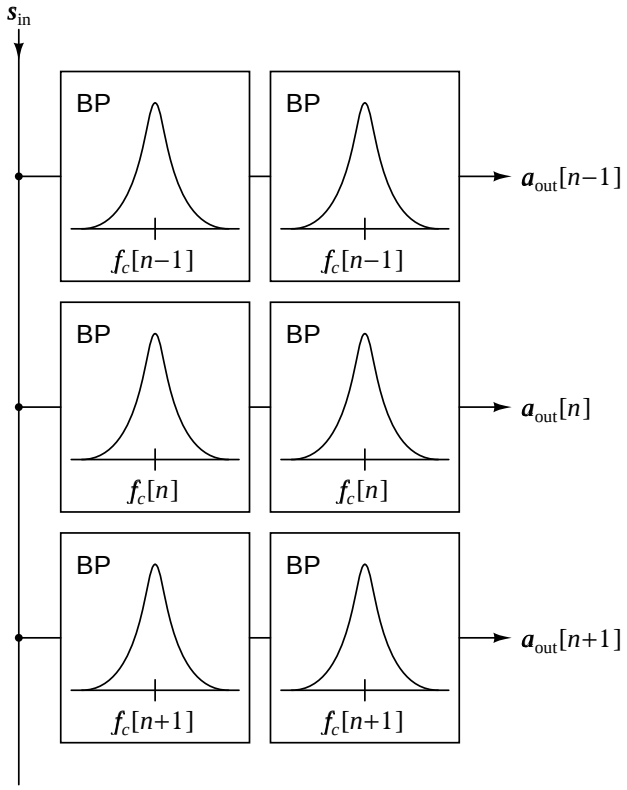


Figure 4.9: Filter signal flow diagram for the parallel filterbank configuration.

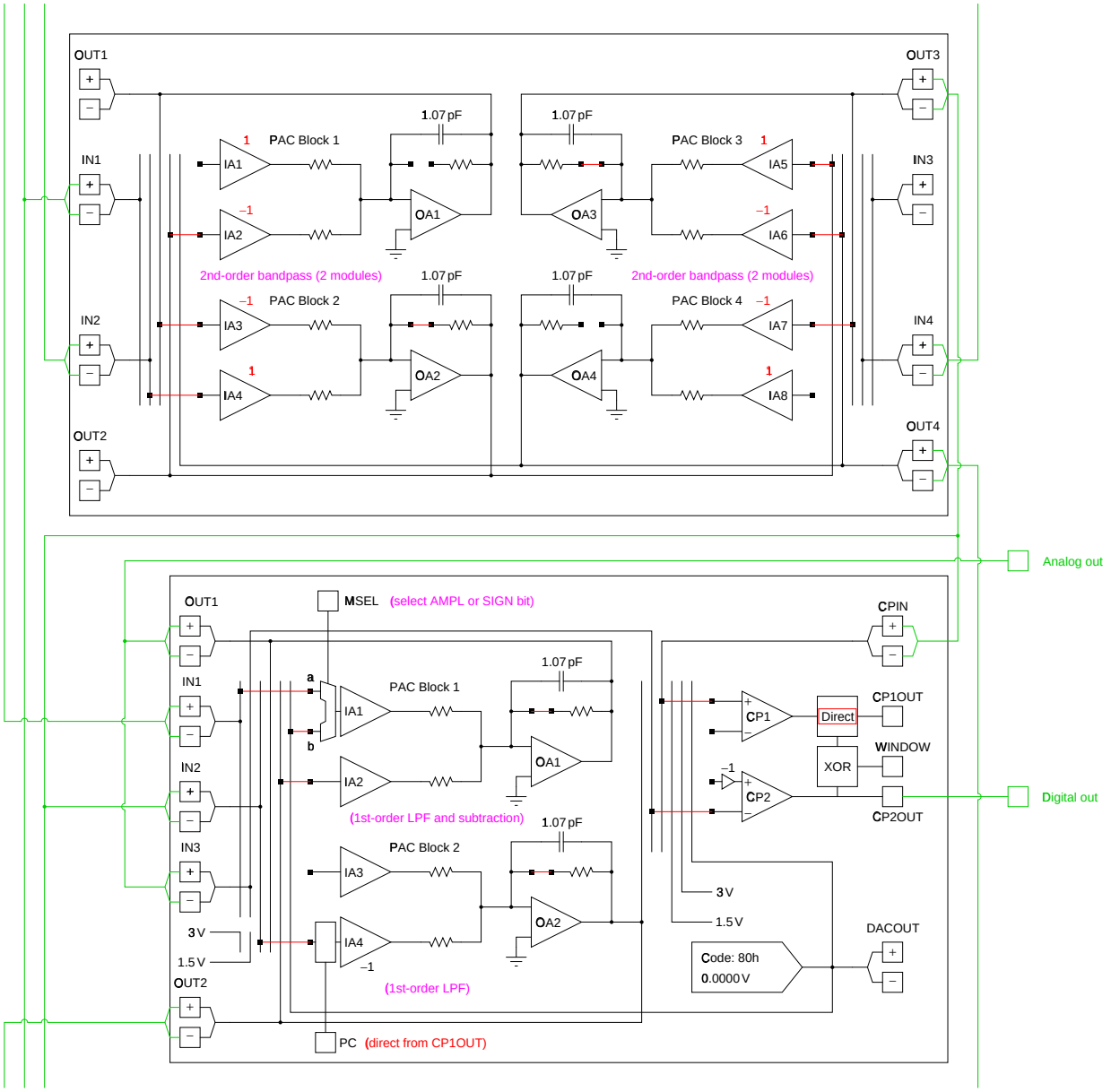


Figure 4.10: ispPAC10 and ispPAC20 internal connections for the parallel filterbank configuration.

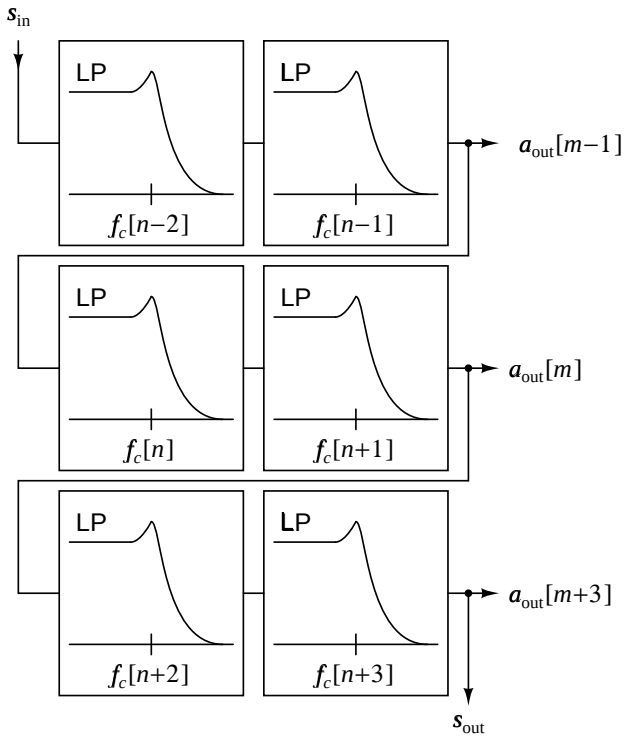


Figure 4.11: Filter signal flow diagram for the cochlear filterbank configuration.

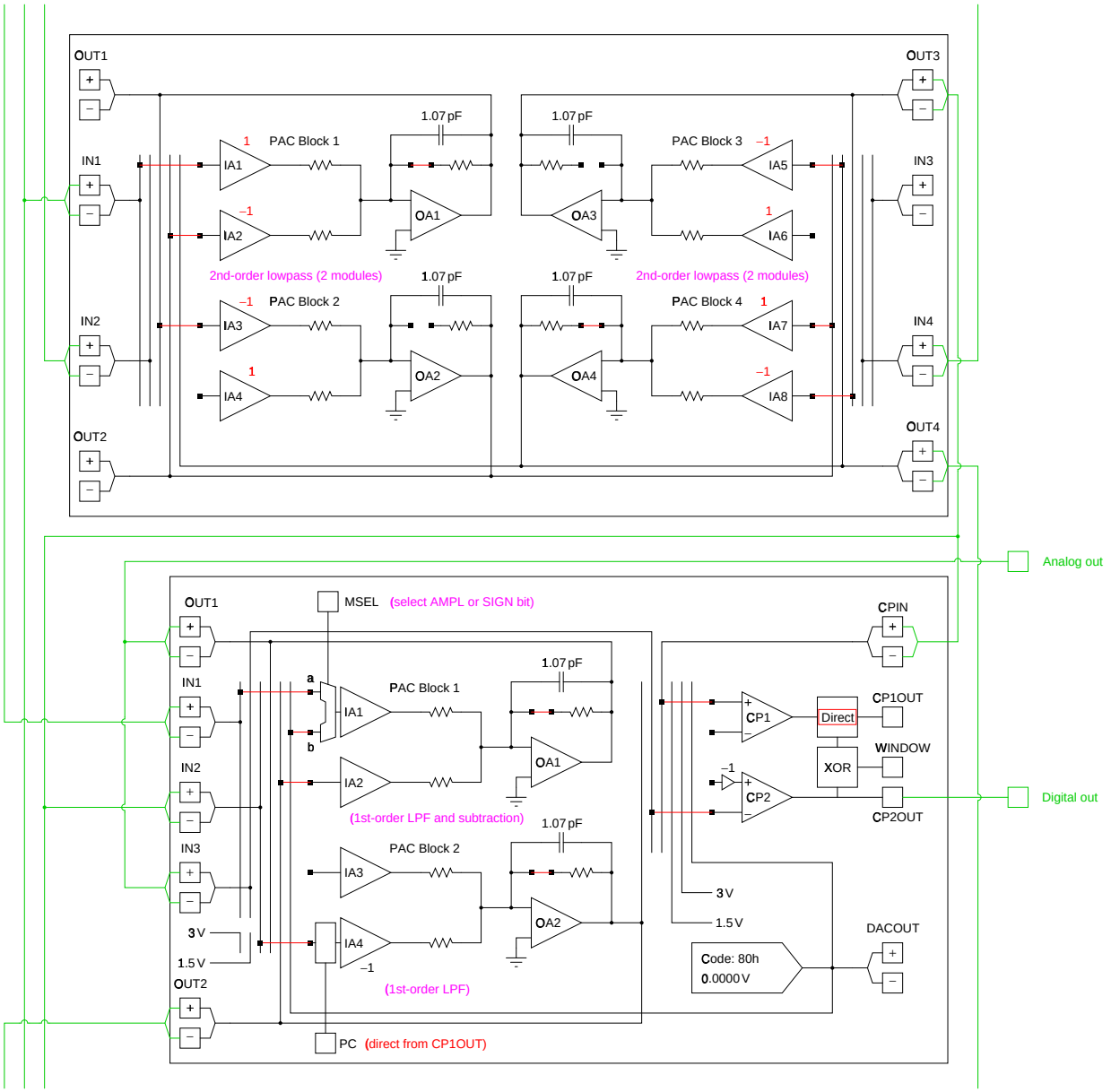


Figure 4.12: ispPAC10 and ispPAC20 internal connections for the cochlear (2nd-order lowpass cascade) filterbank configuration.

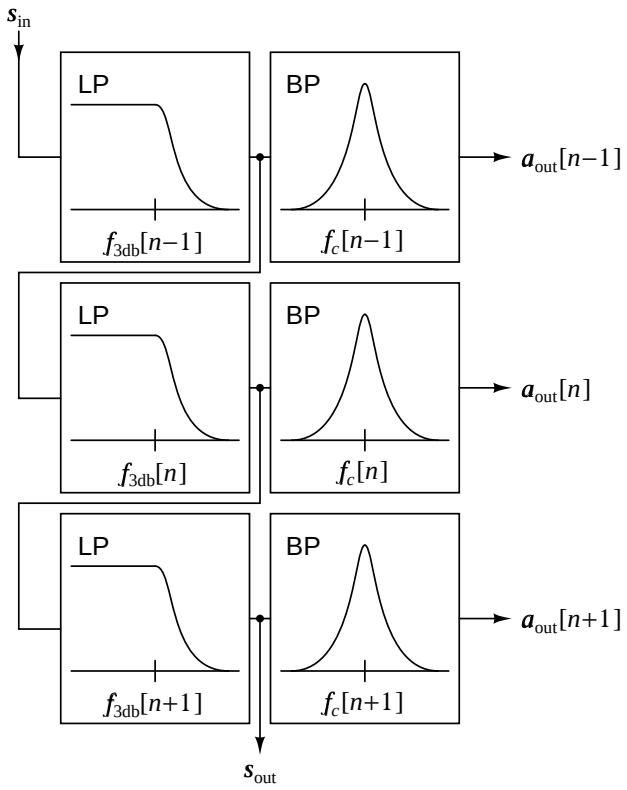


Figure 4.13: Filter signal flow diagram for the cascade-parallel filterbank configuration.

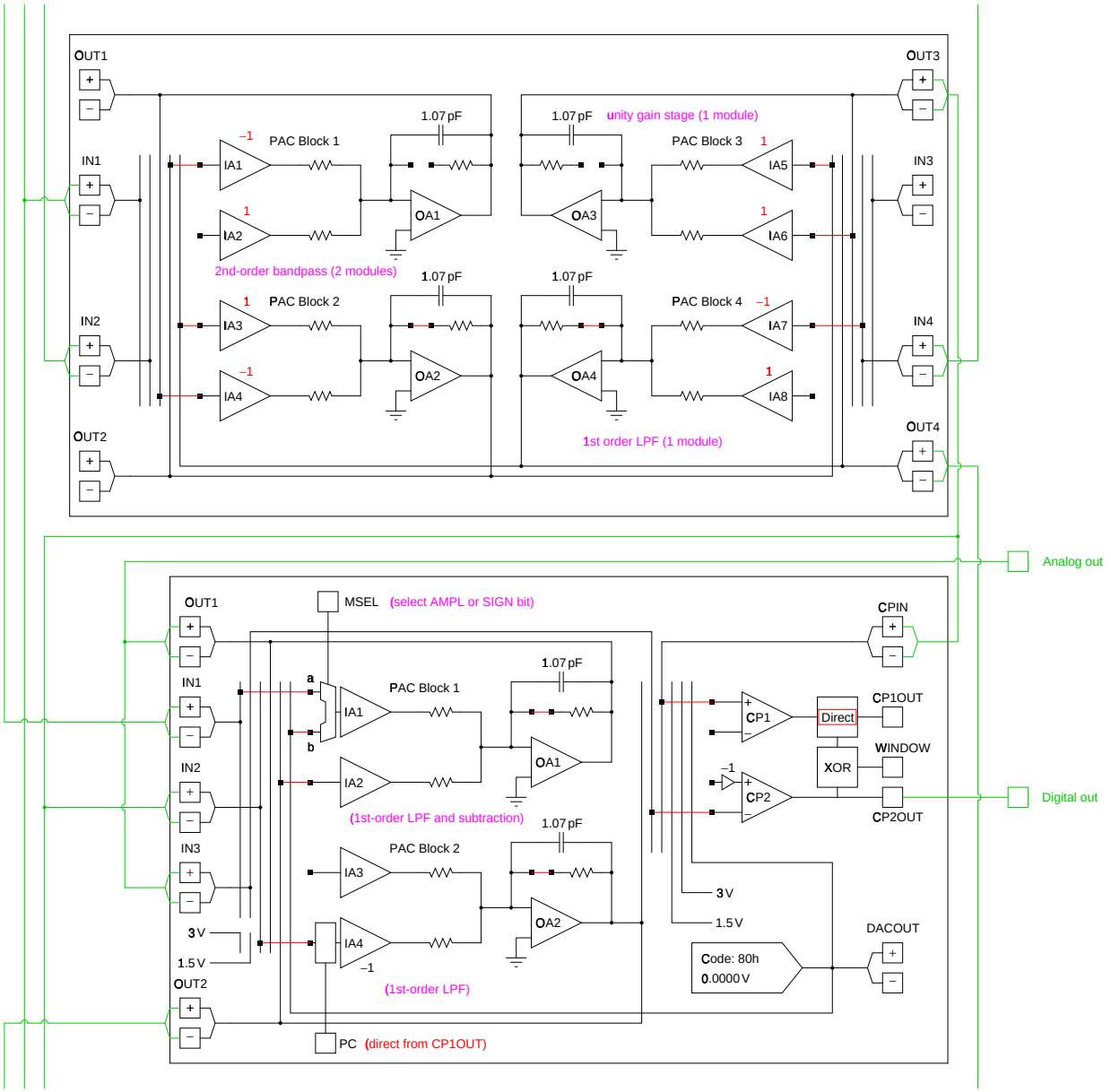


Figure 4.14: ispPAC10 and ispPAC20 internal connections for the cascade-parallel (1st-order lowpass cascade, each tap followed by 2nd-order bandpass) filterbank configuration.

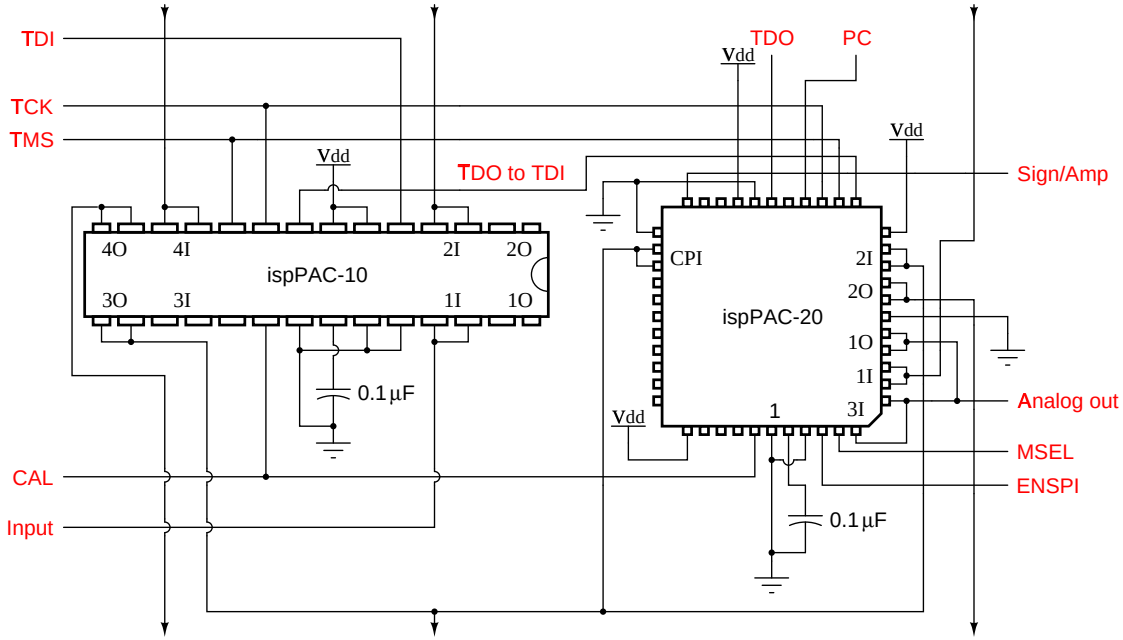


Figure 4.15: Schematic showing one filterbank channel.

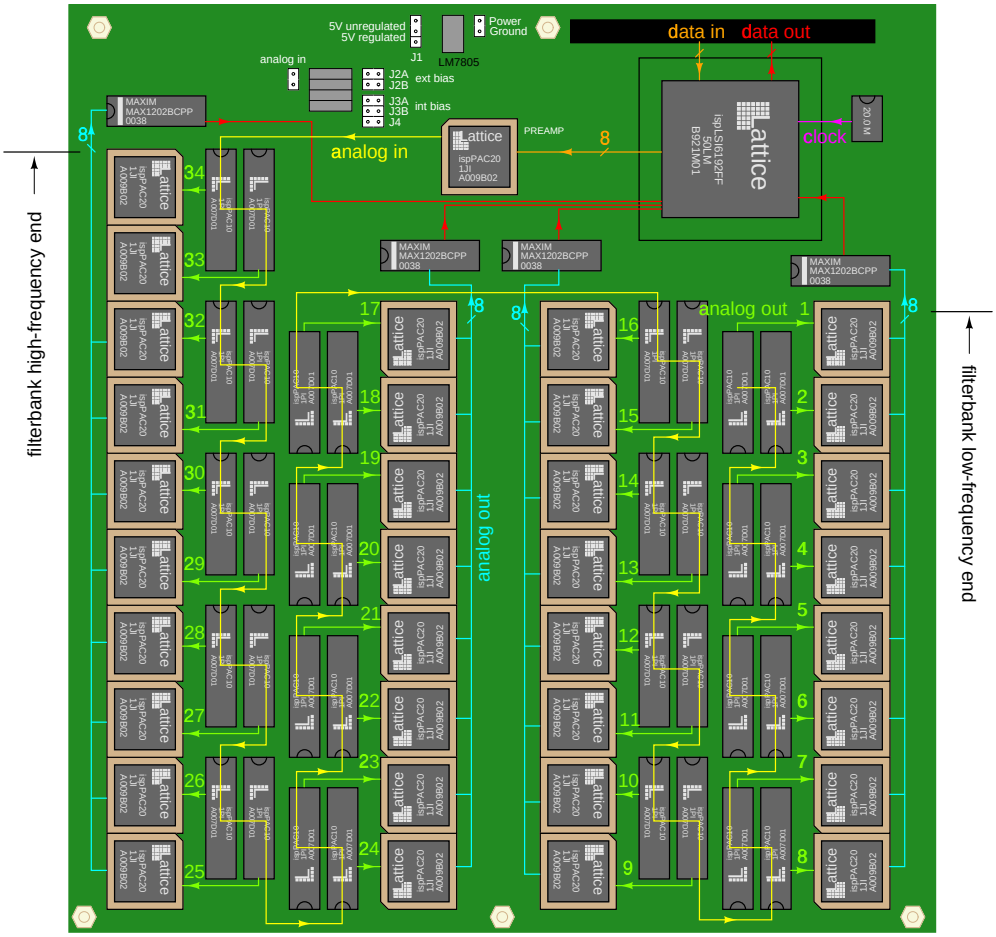


Figure 4.16: Channel numbering and signal flow diagram for the frontend board.

Chapter 5

Software

The main principle behind the term "ISP" (in-system programmable) is that chips with true ISP capability can be reprogrammed without requiring removal from the application system. A typical application is to update firmware with bug fixes or protocol changes and extensions. The BioSonar board takes the ISP principle one step further, to the extent that the system consists almost entirely of programmable chips and can be reconfigured to match the parameters of each experiment. When analyzing sonar data, the BioSonar board can be reconfigured to look at different frequency ranges and spacings, thresholded or sampled output, amplitude envelopes or zero crossings, and different input and downsampling rates, all through ISP downloads. The details of the ISP interface are mostly hidden from the end-user; at the highest level, the end-user can merely specify the range of frequencies to observe, for instance. At a lower level, the end-user can micro-manage the signal-flow by choosing which inputs get routed to which amplifiers inside the ispPAC chips. However, the software setup encourages the use of a hierarchy of subroutines to ensure that the programmable features of the system which are shown to the end-user are on as high a level as possible, and meaningful in the context of the application. Letting the end user choose the low frequency and high frequency bounds of the filterbank is a useful function; requiring the end user to determine capacitor values and figure out which channel is the first and last in the filterbank is not.

The application program supplied with the system hardware is called **filterbank** and serves two major purposes. The first is to act as the programmer for the BioSonar system. **filterbank** can erase, reprogram, and verify every programmable chip on the board. The second function is to operate the board in its digital-input mode, where input to the board is generated from a data file, and output from the board is captured and interpreted by the computer. Because the computer (running Linux) is not a real-time system, it must communicate with the BioSonar board through a bidirectional FIFO buffer; otherwise it cannot keep up with the input and output data rates of the board.

5.1 Using the `filterbank` program to change the firmware

The usage statement for **`filterbank`** reads as follows:

> **`filterbank`** [*jedec_file*`.jed`] [general_options] [program_options — runtime_options]
>
> [*jedec_file*`.jed`] is the name of a JEDEC bitstream file to be loaded to the ispLSI chip; it defaults to "`combo4.jed`" and is ignored if the ispLSI is not programmed or verified according to the options set (see below).
>
> [general_options] can be any of the following:
> **`-diagnostic`** print all the bit streams
> **`-verbose`** print all diagnostic information

[program_options] can be any of the following for programming:

-noLSI	don't do for the ispLSI chip
-noPAC	don't do for the ispPAC chips
-noDAC	don't do for the ispPAC20 onboard DACs
-LSIonly	only do for the ispLSI chip
-PAConly	only do for the ispPAC chips
-DAConly	only do for the ispPAC20 onboard DACs
-verifyonly	don't program, only verify
-setvalues	query new values for ispLSI counters and FIFO

[runtime_options] can be any of the following for data processing:

-run	put the system in run mode
-quiet	don't print to stderr during run
-offset n	start processing at n bytes into the input file
-attenuate n	attenuate input by factor of 2^n
-length n	download only n bytes of the input file
-output *filename*	use *filename* for the output file, rather than the default "data/channels.dat"

The options parser searches the command line for a starting '-' followed by key strings taken from three sets: 1) **no** and **only**; 2) **verify**, **PAC**, **DAC**, and **LSI**, and 3) the remaining single-word options (**run**, **setvalue**, **quiet**, **verbose**, and **output**. Only the presence of these strings in the option word is required, and all other characters are ignored. Thus, **-noLSI**, **-no_LSI**, and **-LSI=no** are all equivalent options. Combination **no** + **verify** is not allowed. Certain options will combine, so **-noLSI -noPAC** is equivalent to **-DAConly**, and **-DAConly -PAConly** does nothing, as these two options are mutually exclusive.

As the usage statement shows, programming is split into three parts. One of these is for programming the ispLSI6192FF interface FPGA chip. This part is separate because of the complexity of the LSI programming method. The second part is for programming all of the ispPAC10 and ispPAC20 chips in parallel. The third part is for programming the DAC values in all of the ispPAC20 chips. Because the current system does not make use of the filter DACs, the third function is not required. However, the ispPAC20 chips could be set up such that the negative input to comparator CP1 was attached to the internal DAC output instead of the reference voltage, and then the DACs could be used to trim offset errors in the channel outputs. The DACs can also be used to set threshold values for amplitude thresholding. Currently, the parameters of the filterbank are hard-coded into the source, so changes to the filterbank setup require changes to the source code and recompilation.

5.1.1 Example Uses

The following are several example uses of the "filterbank" program for firmware programming.

filterbank -PAConly
Program the filterbank and preamplifier as defined in the source code.

filterbank -PAConly -verifyonly
Verify that the filterbank configuration matches the configuration defined in the source code.

filterbank -LSIonly data/expt.jed
Program the ispLSI6192FF interface chip with the bitstream data found in the file "data/expt.jed" (produced by ispDesignExpert).

`filterbank -LSIonly -verifyonly data/expt.jed`
Verify that the current interface configuration matches the data in the file "`data/expt.jed`".

`filterbank -LSIonly -sevalues data/expt.jed`

```
Using JEDEC file "data/expt.jed"
JEDEC file [data/new.jed]:
Set counter/FIFO values:
Input rate [100000.0 Hz]:
Output rate [10000.0 Hz]:
FIFO ALE level [12.3%]:
FIFO ALF level [87.5%]:
Wrote file "data/new.jed" with new values.
```

This is the only command that does not communicate with the BioSonar board. Its purpose is to alter several bit subsquences of the ispLSI6192FF JEDEC file which have a fixed position and purpose. The built-in register and FIFO modules of the ispLSI6192FF architecture, unlike the rest of the FPGA, are not built from generic modules, and their configuration bits occupy a fixed position in the JTAG data stream. Fixed-position configuration bit sequences include the preset counter values for the T4R4CPV register/counter (see Figure 4.5) and positions of the ALE ("almost empty") and ALF ("almost full") flags on the built-in FIFO. Because the counter values represent the input and output data rates of the BioSonar system, it is more convenient to program them as such, and have the computer calculate what counter values are needed. This call to "filterbank" displays the interactive input shown above, prompting for the values of the input data rate, output data rate, and FIFO flag positions. It then generates an output file "data/new.jed" with the altered configuration data, and exits.

5.2 Using the `filterbank` program to process data

The current instantiation of the **filterbank** run-time code is designed for digital input and analog output converted to 12-bit digital words. The input comes from three different BioSonar datasets. One of these is the "LFM" dataset, with a sample rate of 256 kHz, and the other two are the "Expt" and "BOSS" datasets, both with a sample rate of 100 kHz. "LFM" contains data from pinging six buried objects, underwater, with sonar pings produced at 5-degree intervals on a circle around the object, for a total of 72 recordings of sonar backscatter for each of the objects. "Expt" contains five objects with multiple (but nonuniform) recordings taken for each. The recordings have been split arbitrarily into three sets for training, test, and cross-validation. "BOSS" contains two objects with multiple (but nonuniform) recordings taken for each. The recordings have been split arbitrarily into three sets for training, test, and cross-validation.

5.2.1 Matlab Scripts Part 1

The "**`convert*.m`**" scripts were written to convert the data sets as received from Orincon Co, Hawaii, from binary Matlab format into a simple raw file (samples only, no header) for quick downloads to the BioSonar board. Although events in each data set can be grouped together and processed in series in real time for speed and effictiency, the present set of shell scripts assumes one input file per event, which is slower but simpler to implement. The original files are **`rev_pad_LFM2.mat`** for the LFM data set, **`Expt4_5ClassData.mat`** for the Expt data set, and **`OCEANS2001_2Cl_data.mat`** for the BOSS data set. The last two of the three are in a structured format designed for the BioSonar program, defining six fields as follows:

datasetname`.trn`
: Training data matrix, no. events × samples per event

datasetname`.test`
: Test data matrix, no. events × samples per event

datasetname`.cv`
: Cross-Validation data matrix, no. events × samples per event

datasetname`.t_trn`
: Vector (length = no. events) of class types for each event in the training set

datasetname`.t_test`
: Vector (length = no. events) of class types for each event in the test set

datasetname`.t_cv`
: Vector (length = no. events) of class types for each event in the cross-validation set

Output of the "**`convert*.m`**" scripts is a complete set of files named *type*`_m_n.raw`, where *type* can be one of **`train`**, **`test`**, or **`cross`**, where *m* is the class number, and *n* is the event number within the class. These raw files are placed in subdirectories named *`lfm_data`*, *`expt_data`*, and *`boss_data`*, corresponding to the three datasets. Some of the **`convert`** scripts have been misplaced or overwritten, but any of them can be regenerated from the intact example script for the BOSS data, which is **`convert4.m`**.

5.2.2 Runtime Scripts

Although the simplest way to operate the filterbank is to issue the command

`filterbank -run`

the best way to handle large data sets is through scripts. Several shell (csh) scripts are provided for handling the "LFM", "Expt", and "BOSS" data sets. The available scripts are summarized in Table 5.1

Below is an example script for the "Expt" dataset, showing the loop over all raw files in the **`expt_data`** subdirectory.

```
set i = 1
while ($i <= 5)
   set j = 1
   while (-f expt_data/train_${i}_${j}.raw)
      filterbank -run -quiet -attenuate 8 \
                 -output expt_data/train_output_${i}_${j}.dat \
                 expt_data/train_${i}_${j}.raw
      set k = $j
      @ j++
   end
   echo Processed training class ${i}, ${k} members
   @ i++
end
echo "Done with training set"
```

The **`-attenuate 8`** option shifts the 16-bit input data by 8 bits to fit the 8-bit word written to the FIFO and, ultimately, converted to a differential analog voltage and applied to the filterbank input. This is merely a format consideration, not gain control.

Figures 5.1 and 5.2 show sample inputs and outputs written to and read from the BioSonar board by the abovementioned shell script.

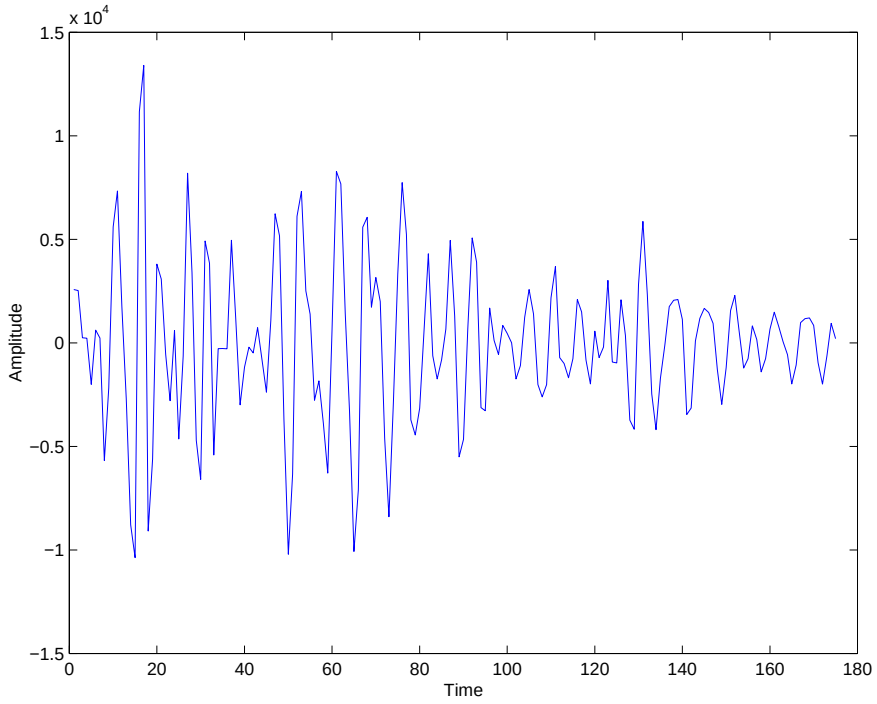


Figure 5.1: Plot of an example sonar backscatter signal from the dataset 'Expt' (original 'Expt4_5ClassData.mat').

The input file format is binary, with 16-bit samples (2 bits per sample, low-order byte first, or "little-endian" format). The output file is ASCII, with output values in the range 0–4096 (the range of the 12-bit ADC outputs), arranged 32 values to a line, one for each filterbank channel. Each line represents a successive time sample of the output. Nominally, the ADC outputs should be centered around the value 2500, corresponding to the common-mode voltage of the ispPAC devices, or a differential value of zero volts. Normally, this mean value should be subtracted out of the data.

5.2.3 Matlab Scripts Part 2

Additional Matlab scripts read the processed (ASCII) output files generated by the shell scripts, and create a Matlab **`.mat`** format (binary) file which is compatible with the Matlab simulations of the backend support-vector machine (SVM) algorithm. Finally, other scripts generate a complete display of all processed data, where each event is a colored 2-dimensional map such as is shown in Figure 5.2. Table 5.1 lists the scripts which handle various datasets.

5.3 Customizing `filterbank`

The **`filterbank`** program is only useful if it can be altered to match the demands of a particular experiment, so it is critical to understand the source code and how to modify it for custom applications.

This section describes the software interface, building up from the lowest-level calls to the high-level description of the application program itself.

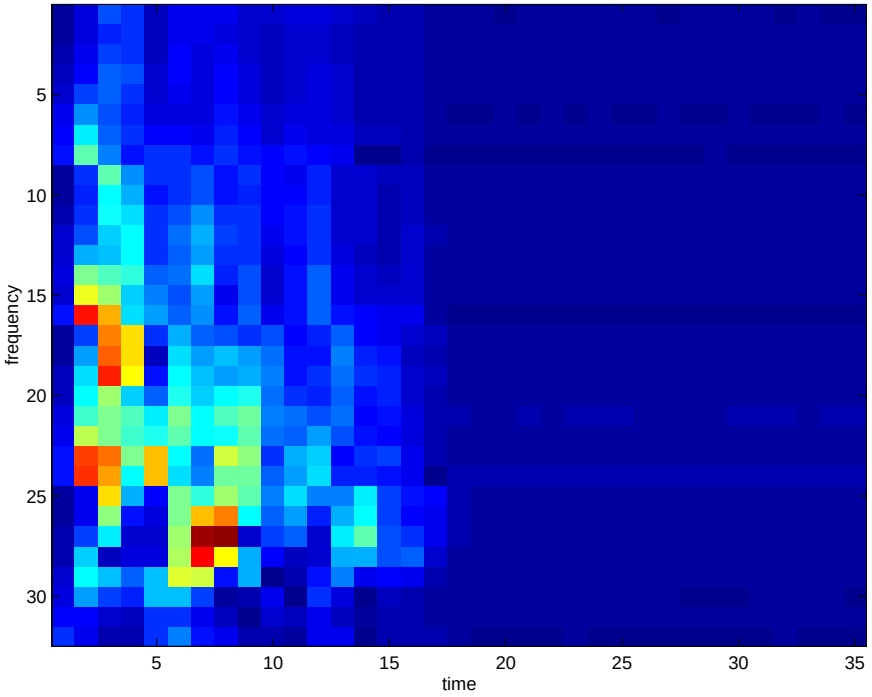


Figure 5.2: Plot of the captured output for a single training event of dataset ‘Expt’.

Binary data file creation		
Script	*Dataset*	*Subset*
`create_frontend`	`LFM2.mat`	N/A
`create_frontend_cv.m`	`Expt4_5ClassData.mat`	`Cross-Validation`
`create_frontend_test.m`	`Expt4_5ClassData.mat`	`Testing`
`create_frontend_train.m`	`Expt4_5ClassData.mat`	`Training`
`create_boss_cv.m`	`OCEANS2001_2Cl_data.mat`	`Cross-Validation`
`create_boss_test.m`	`OCEANS2001_2Cl_data.mat`	`Testing`
`create_boss_train.m`	`OCEANS2001_2Cl_data.mat`	`Training`
Data display		
Script	*Dataset*	*Subset*
`show_frontend.m`	`LFM2.mat`	N/A
`show_frontend2.m`	(others)	(all)

Table 5.1: Matlab scripts

5.3.1 CIO-DIO96 Driver

On the lowest level is the CIO-DIO96 driver. Any of the following list of Measurement Computing, Inc. digital input/output boards are compatible and may be used with the driver and with the application program:

CIO-DIO48	CIO-DIO48H	CIO-DIO96
CIO-DIO96H	CIO-DIO192	CIO-DIO192H
PCI-DIO48H	PCI-DIO48H/CTR15	PCI-DIO96H
CPCI-DIO48H	CPCI-DIO48H/CTR15	CPCI-DIO96H

The following boards are *not* compatible:

(not enough interface pins)	(different pin arrangement on 50-pin connector)
CIO-DIO24	NICB-DIO96
PCI-DIO24	
PCI-DIO24H	
CPCI-DIO24	
CPCI-DIO24H	

The driver program can be found in source directory **`CIO-DIO96`** and consists of a Linux kernel module **`dio96.o`** and a header file to be included in any source using the module, **`dio96.h`**. In addition, executing "make" creates one UNIX character device for each 82C55 chip on the DIO board:

```
\def\dio\AP
\dev\dio\BP
\dev\dio\CP
\dev\dio\DP
```

The devices allow **`read()`** and **`write()`** calls to each device to read the 82C55 ports A, B, and C individually or at once. However, the usual mode of operation is to make **`ioctl()`** calls to the device to set, clear, or read 8-bit ports or single bits without disturbing the other ports. The ioctl calls are defined in the **`dio96.h`** header file.

```
SET_MODE
GET_MODE
SET_p
GET_p
SET_pn
CLR_pn
GET_pn
```

where *p* is one of the three ports "A", "B", or "C", and *n* is a bit number from 0 to 7. Bitwise commands take no argument (argument is NULL). The `SET_MODE` command takes an argument which is defined in the header file:

```
OUTPUT
CNTL_A
CNTL_B
CNTL_CH
CNTL_CL
```

The mode of each port is set to output by default (`OUTPUT = 0`). Modes need to be explicitly declared as input by ORing together the modes for the indicated channels. For example, the C language call

```
ioctl(AP, SET_MODE, CNTL_A | CNTL_CH);
```

Would set the mode of 82C55 chip "AP" for port A input, port B output, port C upper four bits input, and port C lower four bits output.

The driver methods encourage use of preprocessor definitions to make the application program source code more readable. Also, it encourages the use of other practices to increase readability, such as the naming of file pointers to open devices to match the device name:

```
int AP = open("/dev/dio/AP", O_RDWR);
```

Preprocessor definitions are collected in file "`defines.h`". Because the definitions hide the port designations for signals, it is useful to also define the 82C55 mode (input or output direction of each port) within the file:

```
/* Define I/O modes for channels A and B */
#define AP_MODES CNTL_CL | CNTL_A | CNTL_B
```

such that the application program makes a call which hides the details:

```
ioctl(AP, SET_MODE, AP_MODES);
```

Individual bit lines are defined such that the source code calls them by signal name, not by port designation:

```
#define SET_MSEL(a)    ioctl(BP, ((a) == 0) ? CLR_C1 : SET_C1, NULL)
#define GET_TDO(a)     ioctl(AP, GET_B0, &a)
#define GET_FLAGS(a)       ioctl(AP, GET_C, &a); a &= 0x0f
#define COMP_WRITE_DATA(a) ioctl(BP, SET_A, a)
```

The above four calls represent bit-wise set, clear, and read, and byte-wise read and write operations. Hardware changes to the interface which change pin positions are managed by editing the file "`defines.h`" only.

A few definitions combine multiple signal operations into a single command. This ensures that the proper sequence of signals is defined where it is less likely to be accidentally changed by program modifications:

```
#define TCLOCK            ioctl(AP, SET_C6, NULL); \
                          ioctl(AP, CLR_C6, NULL)
#define compwritevalue(v) COMP_WRITE(0); COMP_WRITE_DATA(v); \
                          COMP_WRITE(1)
```

The first definition defines a command "TCLOCK" which pulses the JTAG TCK line (set operation followed by a clear operation). The second defines a FIFO write operation by dropping the $\overline{\text{WR}}$ line, applying 8 bits data, then raising the $\overline{\text{WR}}$ line.

5.3.2 JTAG Calls

The JTAG interface is too complicated to be explained in detail here; a good description of the JTAG interface used for ispPAC chip programming can be found in the Lattice Semiconductor ispPAC-10 and ispPAC-20 datasheets. JTAG operations are built up in hierarchical fashion, starting at the lowest level and working up to high-level chip programming functions. For clarity, these layers are defined and will be referred to as follows:

1. **Hardware Layer**: 82C55 ports; direct addressing to the hardware through machine instruction 'inb' and 'outb' calls. This layer is written into the driver.
2. **Driver Layer**: Low-level UNIX protocol: open(), close(), read(), write(), and ioctl() calls.
3. **Signal Layer**: Definition layer which hides the detail of ioctl() calls behind signal names relevant to the application.
4. **State Machine Layer**: Direct implementation of the JTAG state machine using Signal Layer calls
5. **Instruction Layer**: Implementation of JTAG instructions
6. **Device Layer**: Implementation of the set of instructions for a particular device.
7. **Application Layer**: The overall application calls the JTAG Device Layer commands to program devices in the system to perform specific tasks. The entire JTAG interface is hidden from the end-user by this layer.

The JTAG interface is defined in the C source file `common.c`, containing routines common to all of the BioSonar board programmable chips. Operation of the JTAG state machine is carried out by Signal Layer calls to set and clear the five JTAG control and data bits. The definitions for these bits are as follows:

```
#define SET_TMS(a)   ioctl(BP, ((a) == 0) ? CLR_C6 : SET_C6, NULL)
#define SET_RST(a)   ioctl(BP, ((a) == 0) ? CLR_C5 : SET_C5, NULL)
#define SET_TDI(a)   ioctl(AP, ((a) == 0) ? CLR_C5 : SET_C5, NULL)
#define GET_TDO(a)   ioctl(AP, GET_B0, &a)
#define TCLOCK       ioctl(AP, SET_C6, NULL); ioctl(AP, CLR_C6, NULL)
```

The exact sequence of these signals needed to move around the JTAG state machine are hidden behind the State Machine Layer functions:

`void HardReset()`
hardware-initiated reset using the TRST signal.

`void SoftReset()`
software-initiated reset using the state machine.

`void Idle()`
Move from any state into the idle state.

`void Pause_To_Idle()`
Move from the paused state into the idle state.

`void Init_Data()`
Prepare for data send (data follows).

void Init_Instruction()

Prepare for instruction send on ispPAC device (instruction follows).

void LSI_Init_Instruction()

Prepare for instruction send on ispLSI device (instruction follows).

None of the State Machine Layer calls takes any argument; they are simply sequences of applied Signal Layer calls (TMS and RST signals and pulses of TCK). The State Machine Layer calls are bundled into the Instruction Layer calls for reading and writing the Lattice devices (found in file `common.c` for ispPAC-10 and ispPAC-20 operations, file `ispDAC.c` for operations on the ispPAC-20 DAC configuration, and file `ispLSI.c` for ispLSI6192FF operations).

Most Instruction Layer and Device Layer calls require a parameter *chain_types*. This is an integer array of size **CHAIN_TOTAL** which contains the ID number of each chip in the JTAG chain. For the BioSonar board, **CHAIN_TOTAL** is 70 (34 ispPAC-10 chips, 34 ispPAC-20 chips in the filterbank, 1 ispPAC-20 chip for the preamplifier, and 1 ispLSI6192FF). Chip ID's are the ID numbers returned by each chip on a `QueryID` command (hex 0x100 for the ispPAC-10, hex 0x111 for the ispPAC-20, and hex 0x32 for the ispLSI6192FF). The subroutines use this device array information to address or bypass specific chips in the JTAG chain.

void Instruction(int *chain_types, uchar inst)

Apply ispPAC instruction value *inst*

void DACInstruction(int *chain_types, uchar inst)

Apply ispPAC DAC instruction value *inst*

void LSIInstruction(int *chain_types, uchar inst)

Apply ispLSI-6192 instruction value *inst*

void Capture(int bits, int lastbit, uint *rval)

Read *bits* bits (up to 32) from the JTAG TDO line. If *lastbit* is reached, exit the JTAG read cycle. Result is placed in unsigned integer pointed to by *rval*.

void CaptureStream(uint *bitstream, int ssize)

Read *ssize* bits from the JTAG TDO line, placing the result in the unsigned integer array pointed to by *bitstream*.

void CaptureIDStream(uint *bitstream, int ssize, int LSIcount, int *chain_types)

Read *ssize* bits from the JTAG TDO line, placing the result in the unsigned integer array pointed to by *bitstream*. *LSIcount* is the number of ispLSI chips in the *chain_types* array. This is different from `CaptureStream()` because the ispLSI chips do not provide an ID code in the same manner as the ispPAC chips and must be bypassed.

uint Write(int bits, int lastbit, uint testword, uint *rval)

Write *bits* bits (up to 32) to the JTAG TDI line. Bits are taken from *testword*. If *lastbit* is reached, exit the JTAG write cycle. JTAG write operations pass through the chain and can be captured at TDO for verification, so TDO is read before each bit is written, and the result stored in *rval* as it is done for the `Capture()` instruction.

void WriteStream(uint *bitstream, int ssize, uint *rval)

Write *ssize* bits to the JTAG TDI line, using the bits stored in the unsigned integer array *bitstream*. The TDO result is read before each bit write and the result is returned in the unsigned integer array *rval*.

The full JTAG instruction set used by all of the Lattice chips on the BioSonar board is shown in Table 5.2.

Instruction	Value	Description
Generic JTAG device instruction set		
EXTEST	0	External test. Default to BYPASS
SAMPLE	30	Sample/Preload. Default to BYPASS
BYPASS	31	Bypass (connect TDI to TDO)
ispPAC device valid instruction set		
ADDUSR	1	Address User data register
UBE	2	User bulk erase
VERUSR	3	Verify User data register
PRGUSR	4	Program User data register
IDCODE	13	Read Identification data register
ENCAL	16	Enable Calibration sequence
ispPAC20 device extended instruction set		
DBE	17	DAC bulk erase
VERDAC	18	Verify the DAC register
PRGDAC	19	Program the DAC register
ADDDAC	20	Address the DAC register
ispLSI device valid instruction set		
SHIFT_ADDRESS	1	Enable address shift register
SHIFT_DATA	2	Enable data shift register
LSI_IDCODE	21	Get 8-bit ID Code of device
LSI_BYPASS	14	Bypass (connect TDI to TDO)
ERASE_ALL	16	Enable address shift register
PROGRAM_LOW	8	Program low order bits
PROGRAM_HIGH	7	Program high order bits
VERIFY_LOW	11	Verify low order bits
VERIFY_HIGH	10	Verify high order bits

Table 5.2: Table of all instructions

The Device Layer calls bundle the Instruction Layer command operations into large-scale functions such as programming a chip or verifying a program:

int QueryIDCode(int *_chain_types_, uchar _quiet_mode_)
Check the ID code returned by every ispPAC chip on the board. Returns 0 on success and −1 on failure. If _quiet_mode_ is nonzero, no output is generated.

int QueryLSI_ID(int *_chain_types_ uchar _quiet_mode_)
Same as above, but for the ispLSI6192 only.

void QueryProgram(int *_chain_types_)
Read the current program in all of the ispPAC chips and print the complete bitstream to `stderr`. Returns 0 on success and −1 on failure.

void QueryDACProgram(int *_chain_types_)
Same as above, but reads only the programmed DAC values in the ispPAC20 chips.

int VerifyOnly(int *_chain_types_)
Compare the program in the ispPAC chips to the program described by the bitstream global variable `ispbits[]`. Returns 0 on success and −1 on failure.

int VerifyOnlyDAC(int *_chain_types_)
Same as above, but compares the bitstream containing the values of all the programmed DAC registers of all the ispPAC20 chips against `ispbits[]`.

int VerifyOnlyLSI(char *_filename_, int *_chain_types_)
The program in the ispLSI6192 is compared against the JEDEC bitstream found in JEDEC file _filename_. Returns 0 on success and −1 on failure.

int EraseAndVerify(int *_chain_types_)
Erases the contents of the ispPAC10 and ispPAC20 chips, and verifies that the erase has been successful. Returns 0 on success and −1 on failure.

int EraseAndVerifyDAC(int *_chain_types_)
Same as above, but erases the contents of the ispPAC20 DAC registers only.

int ProgramAndVerify(int *_chain_types_)
Program the ispPAC10 and 20 chips with the bitstream information found in the global array `ispbits[]`. This function calls `EraseAndVerify()` before programming. Returns 0 on success and −1 on failure.

int ProgramAndVerifyDAC(int *_chain_types_)
Same as above, for the DAC registers of the ispPAC20 chips only.

int ProgramAndVerifyLSI(char *_filename_, int *_chain_types_)
Program the ispLSI6192 with the bitstream described by the JEDEC file _filename_. Returns 0 on success and −1 on failure.

int CheckBypass(int *_chain_types_)
Initiates a test operation in which all chips are set to BYPASS mode, and a random bitstream is written to TDI and read out of TDO **CHAIN_TOTAL** clock cycles later. Returns 0 on success and −1 on failure.

void Calibrate(int *_chain_types_)
Initiates a calibration cycle on all of the analog chips (ispPAC10 and ispPAC20) simultaneously.

int ispJTAGenable(int *_chain_types_)
Puts the ispLSI6192 into JTAG mode. This requires a bizarre sequence of instructions and JTAG commands which is not part of the standard JTAG definition. It is required before any call to `ProgramAndVerifyLSI()` or `VerifyOnlyLSI()`.

int ispJTAGdisable(int *_chain_types_)

Puts the ispLSI6192 into normal runtime mode. This routine must be called after every call to `ProgramAndVerifyLSI()` and `VerifyOnlyLSI()`.

5.3.3 ispPAC10 and ispPAC20 Configuration

Application Layer commands are divided into two parts: Setup Commands and Execution Commands. Setup Commands are found mainly in files `ispPAC10.c` and `ispPAC20.c`, whereas the Execution Commands are found mainly in `filterbank.c`. Setup Commands prepare the configuration bitstream to be downloaded to the chip. They do not make any JTAG calls. The filterbank Setup Commands hide the details of which configurations bits in the Lattice chip perform what function, and allow calls to turn certain functions on or off, to select amplifier gain and amplifier input routing by reference to the amplifier "PAC block" by name or number. The Setup Commands rely on a general-purpose routine called "ispstuff()" which merges bit fields into the single long configuration data stream which will be loaded into all of the ispPAC10 and ispPAC20 devices simultaneously.

Properties of the ispPAC10 and ispPAC20 devices are defined as macros for clarity. The applicability of any function should be checked against the configuration map (programming diagram) for each chip (see, for instance, Figure 4.10). Valid input sources for amplifier inputs in the set_amp_route10() and set_amp_route20() subroutines depend on the amplifier chosen, as defined in Table 5.3. As shown in the table, most instrumentation amplifiers are declared by number (1–8 on the ispPAC10 for the input amplifiers, 1–4 for the output amplifiers), but because the ispPAC20 has a multiplexed input on the first instrumentation amplifier, its inputs are named `AMP1A` and `AMP1B` and should use the defined word, not an integer number. The ispPAC20 device can route inputs to the comparators, whose inputs are referenced as listed in Table 5.3, for use with the set_comp_route() and set_comp_hiz() subroutines:

A complete list of configuration subroutines is below. Each of these subroutines takes an integer parameter *`global`*. This parameter is the number of bits into the bitstream (global variable `ispbits[]`) at which the bitstream representing the chip in question begins. The beginning of the **filterbank** program creates the ordering corresponding to the BioSonar circuit board, and creates an array `chip_offsets[]` holding the value of *`global`* to be passed to the configuration subroutines.

`int set_amp10(int global, int number, int route, int gain)`
Declares the input routing and gain for the input (instrumentation) amplifiers in the ispPAC10. *`number`* is 1–8 for the 8 amplifiers. *`gain`* is in the integer range -9 to $+9$ inclusive, but excluding zero. *`route`* is any of the valid inputs to ispPAC10 amplifiers, listed in Table 5.3.

`int set_capacitor10(int global, int number, float value)`
Sets the value of the feedback capacitor for output amplifier *`number`* on an ispPAC10 chip, where *`number`* may be 1–4. *`value`* is the target capacitor value in picoFarads. The routine will find the closest capacitance value supported by the chip, and warn if the value is out of range of capacitor values on the chip.

`int set_feedback10(int global, int number, int value)`
Sets the feedback for output amplifier *`number`* on an ispPAC10 chip, where *`number`* may be 1–4. *value* is 1 for closed circuit (feedback) or 0 for open circuit (no feedback).

`int set_common_mode10(int global, int number, int value)`
Sets the common mode source (external or internal) for each output amplifier of an ispPAC10 chip. *`number`* may be 1–4. *`value`* is 1 for external common-mode source, 0 (default) for internal common-mode source.

`int set_comp_route(int global, int number, int route)`

ispPAC-10 input amplifier sources						
amplifier	**sources**					
1	INPUT_1	INPUT_2	OUTPUT_1	OUTPUT_2	OUTPUT_4	
2	INPUT_1	INPUT_2	OUTPUT_1	OUTPUT_2	OUTPUT_4	
3	INPUT_1	INPUT_2	OUTPUT_1	OUTPUT_2	OUTPUT_4	
4	INPUT_1	INPUT_2	OUTPUT_1	OUTPUT_2	OUTPUT_4	
5	INPUT_3	INPUT_4	OUTPUT_2	OUTPUT_3	OUTPUT_4	
6	INPUT_3	INPUT_4	OUTPUT_2	OUTPUT_3	OUTPUT_4	
7	INPUT_3	INPUT_4	OUTPUT_2	OUTPUT_3	OUTPUT_4	
8	INPUT_3	INPUT_4	OUTPUT_2	OUTPUT_3	OUTPUT_4	
ispPAC-20 input amplifier sources						
amplifier	**sources**					
AMP1A	INPUT_1	INPUT_2	INPUT_3	OUTPUT_1	OUTPUT_2	DAC_OUT
AMP1B	INPUT_1	INPUT_2	INPUT_3	OUTPUT_1	OUTPUT_2	DAC_OUT
2	INPUT_1	INPUT_2	INPUT_3	OUTPUT_1	OUTPUT_2	DAC_OUT
3	INPUT_1	INPUT_2	INPUT_3	OUTPUT_1	OUTPUT_2	DAC_OUT
4	DAC_FULL	DAC_HALF	INPUT_2	OUTPUT_1	OUTPUT_2	DAC_OUT
ispPAC-20 comparator sources						
comparator	**sources**					
CP1_IN1	OUTPUT_2	INPUT_3	CP_IN	DAC_HALF	DAC_FULL	DAC_OUT
CP1_IN2	OUTPUT_2	INPUT_3	CP_IN	DAC_HALF	DAC_FULL	DAC_OUT
CP2_IN1	OUTPUT_2	INPUT_3	CP_IN	DAC_HALF	DAC_FULL	DAC_OUT
CP2_IN2	OUTPUT_2	INPUT_3	CP_IN	DAC_HALF	DAC_FULL	DAC_OUT

Table 5.3: Input source routing choices in the ispPAC10 and ispPAC20 chips.

Sets the origin of an input to one of the comparators on an ispPAC20 chip. *number* is one of the defined types **CP1_IN1**, **CP1_IN2**, **CP2_IN1**, or **CP2_IN2**, as shown in Table 5.3. **IN2** is the positive input, and **IN1** is the negative input.

int set_amp_route20(int *global*, int *number*, int *route*)
Defines the routing source for the ispPAC20 input amplifiers. The source is dependent on the amplifier chosen; valid sources are shown in Table 5.3. Amplifier *number* is referenced by number except for amplifier 1, which has a multiplexed input and therefore defines two routine destinations, defined as **AMP1A** and **AMP1B**.

int set_amp_gain20(int *global*, int *number*, int *gain*)
Sets the gain on each input amplifier (numbered by *number*, 1–4). Valid gains are integers in the range -9 to $+9$ inclusive, excepting value zero.

int set_capacitor20(int *global*, int *number*, float *value*)
Sets the value of the feedback capacitor for output amplifier *number* on an ispPAC20 chip, where *number* may be 1–2. *value* is the target capacitor value in picoFarads. The routine will find the closest capacitance value supported by the chip, and warn if the value is out of range of capacitor values on the chip.

int set_feedback20(int *global*, int *number*, int *value*)
Sets the feedback for output amplifier *number* on an ispPAC20 chip, where *number* may be 1–2. *value* is 1 for closed circuit (feedback) or 0 for open circuit (no feedback).

int set_common_mode20(int *global*, int *number*, int *value*)
Sets the common mode source (external or internal) for each output amplifier of an ispPAC10 chip. *number* may be 1–4. *value* is 1 for external common-mode source, 0 (default) for internal common-mode source.

void set_PC_mode(int *global*, int *value*)
Determines what controls the input inversion on input amplifier 4 of an ispPAC20 chip. *value* may be one of the defined modes **FIXED**, **PC_PIN**, **LATCH**, or **CP1**. See the Lattice Semiconductor datasheet for the ispPAC20 for an explanation of these modes.

The remaining commands control boolean (ON/OFF) properties of the chip, so each takes an argument *value* which is either True (1) or False (0). These commands are only defined for the ispPAC20 chip.

void set_hysteresis(int *global*, int *value*)
Enable/Disable 47 mV hysteresis on comparator inputs.

void set_comp_mode(int *global*, int *value*)
Enable/Disable latching of the output of comparator CP1. Latch is updated by toggling the PC pin.

void set_dsthru(int *global*, int *value*)
Enable/Disable serial addressing of the onboard DAC.

void set_tdo_enable(int *global*, int *value*)
Enable/Disable TDO output. Disabling TDO puts it in high-impedence output. This should not be done on the BioSonar board!

void set_security20(int *global*, int *value*)
Sets the security bit, which prevents readout of the chip program until disabled with a bulk erase. This should not be used on the BioSonar board!

void set_window(int *global*, int *value*)
Set the function of the **Window** output pin to be the XOR of the comparators, or FF (latched comparator output).

void set_comp_hiz(int *global*, int *value*)

Disconnects the output pins from the comparators (and the **Window** pin) so that internal-only use of the comparators generates less noise on the analog components.

void set_slewrate(int *global*, int *value*)

Enable/Disable slewrate enhancement on input amplifier 4.

It is helpful to declare the complete configuration information for an entire device in one subroutine, so one subroutine is dedicated to each of the three types of chips used in the filterbank: The ispPAC10 filter, the ispPAC20 signal processor, and the ispPAC20 preamplifier. The main purpose of these routines is to hide the above device-specific definitions from the application program.

int set_filter_mode10(int *global*, int *mode*, float *bpfreq*, float *Q*)

Configure an ispPAC10 chip for BioSonar filtering. The target filter center frequency is *bpfreq*, in Hz, and the filter resonance Q is parameter *Q*. The *mode* parameter is defined in `filterbank.c` and is one of the defined types **PARALLEL**, **COCHLEAR**, or **TAPPED**, corresponding to the major architectures described above.

int set_filter_mode20(int *global*, float *lpfreq*, int *gain*)

Configure an ispPAC20 chip for BioSonar filter output rectification and smoothing. The smoothing filter cutoff frequency is designated by *lpfreq*, in Hz, and the gain (signal boosting) going into the final output is defined by *gain*.

int set_preamp(int *global*, uchar *source*, uchar *mode*, int *gain*)

Configure the ispPAC20 preamplifier on the BioSonar board. Parameter *source* is one of the defined types **SOURCE_ANALOG** or **SOURCE_DIGITAL**. Parameter *mode* is one of the defined types **SINGLE_ENDED** or **DIFFERENTIAL**.

5.3.4 Filter Parameter Calculation

Because the details of the filter parameter calculations is hidden in the set_filter_mode*() commands, which take a floating-point frequency value as a parameter, the calculation of filter parameters requires only to set up an array of center frequency values for the 34 bandpass channels, and choose a Q value and input gain to the filter. The function

void makecfvalues(float *bpfreq, int *channels*, int *minchan*, float *minfreq*, int *maxchan*, float *maxfreq*, int *mode*)

generates the frequency array, based on a minimum frequency *minfreq* to be set at the low-frequency end of the array, and a maximum frequency *maxfreq* to be set at filter number *maxchan* (because the analog output is only valid for the lowest 32 channels, it is more convenient to declare what the top frequency should be at channel 32, not the highest-frequency channel, which is 34). Parameter *mode* can be either **LINEAR** for linear-spaced frequencies, or **LOGARITHMIC** for logarithmically-spaced frequencies.

5.3.5 BioSonar Board Control

Runtime operation of the BioSonar board is initiated by calling **filterbank -run** and calls the main execution routine

void FilterbankRun(char *datafile, char *output_file)

where *datafile* is the raw input file, and *output_file* is the formatted output.

The first part of this routine sets control lines as necessary to reset the BioSonar board, and then performs a direct hardware reset of the FIFO chips. The "full" flag of the FIFO driven by the BioSonar board is monitored and the system is stopped when the FIFO is full. Writes to a full FIFO are ignored, so the time delay between the FIFO flag signal and the shutdown of the BioSonar board is unimportant.

Most of the hardware-level calls made in the **FilterbankRun()** routine cannot be altered without adversely affecting the operation of the system. However, the interface can be configured to accept certain software signals to alter run-time parameters on the board. One such signal is **Select0** which is toggled by the **SET_MSEL()** macro definition in the runtime code. This controls the value of the **MSEL** pin on the ispPAC20 devices in the filterbank. The **MSEL** pin selects the source of the first input amplifier on the first "PAC-Block" of the ispPAC20. In the configurations described in this document, this pin can be used in one of two ways: When capturing analog output, the command **SET_MSEL(1)** selects non-differential output, in which the ADC captures the amplitude envelope of each channel. The command **SET_MSEL(0)** selects differential output, in which the ADC measures the difference between the amplitude envelope of each channel and that of its neighbor on the higher-frequency side. When capturing the two-bit digital output (there is no corresponding schematic for the interface chip described in this document for two-bit digital output mode), **SET_MSEL()** can be used to toggle between the amplitude and the sign bit.

5.4 Error Messages

filterbank will generate error messages any time it gets erroneous information back in the bitstream from the Lattice chips.

"**IDCODE Query: failed. Manufacturer not Lattice Semiconductor.**"
This is the most likely error to show up, as it is the first test made by the program which reads data off of the BioSonar board. Failure can indicate that the board is not attached or is not powered (most likely), that there is an error with the interface connections (less likely), or that a chip has gone bad (least likely).

"**LSI Query ID failed: Expected 0x32, Read: ...**"
Can indicate the same as "IDCODE Query" if the "-LSIonly" option is chosen, because it is then the first attempt the program makes to communicate with the BioSonar board. Otherwise, it indicates an error with the ispLSI6192FF chip (such as a bad chip).

"**Pass TDI through to TDO: failed. Wrote bits: ...**"
This is a simple test to put all the chips in bypass mode (1 bit register per chip in the JTAG chain) and see if a bit sequence passed into the first chip comes back unaltered 70 clock cycles later. As the second query to the board, this can sometimes fail as a result of an improper JTAG state machine state. It should not occur with the current software version.

"**Fifo length unknown**"
System tried to clock data into the FIFO chip and wait for the "Full" flag to change. Error here indicates that the FIFO board is not present or that the FIFO chip and/or signalling is bad.

"*****Warning: 1st value returned is not 0xf0!**"
The first byte of a block of raw output data should nominally be 0xf0 (MSBs of the resting state of the system—output cannot change instantaneously so this value must always be in range). If it is not, it signifies a glitch in the system which may be either a power transient or a FIFO control line transient error. Data is lost and must be reprocessed.

"**`error opening DIO96 device AP`**"
This is a device-level error and indicates either that the CIO-DIO device driver is not installed in the kernel, is not operating properly (e.g., has the wrong I/O address for an ISA card), or the device has been opened by another application and is in use.

"**`Erase all: failed at bit` *n***"
Indicates that the ispPAC devices failed to erase. This indicates a bad device.

"**`Program: failed. Wrote: ...`**"
Indicates that the ispPAC device failed to verify after a programming step. Indicates a bad device.

"**`Verify: failed. Expected: ...`**"
This is not necessarily an error but simply indicates that the program read off of the chip does not match the program to be verified. Every program is verified before writing, so that if the program already exists on the chips, it will save an erase/rewrite cycle.

"**`Error: Chip count is` *m* `but should be` *n***"
This is a software error and is a heads-up that the device ordering has been altered and is not consistent with the declared number of devices, **`CHAIN_TOTAL`**.

5.5 Source Directory Structure

Figure 5.3 shows the directory hierarchy of the software included with the BioSonar distribution and mentioned in this document.

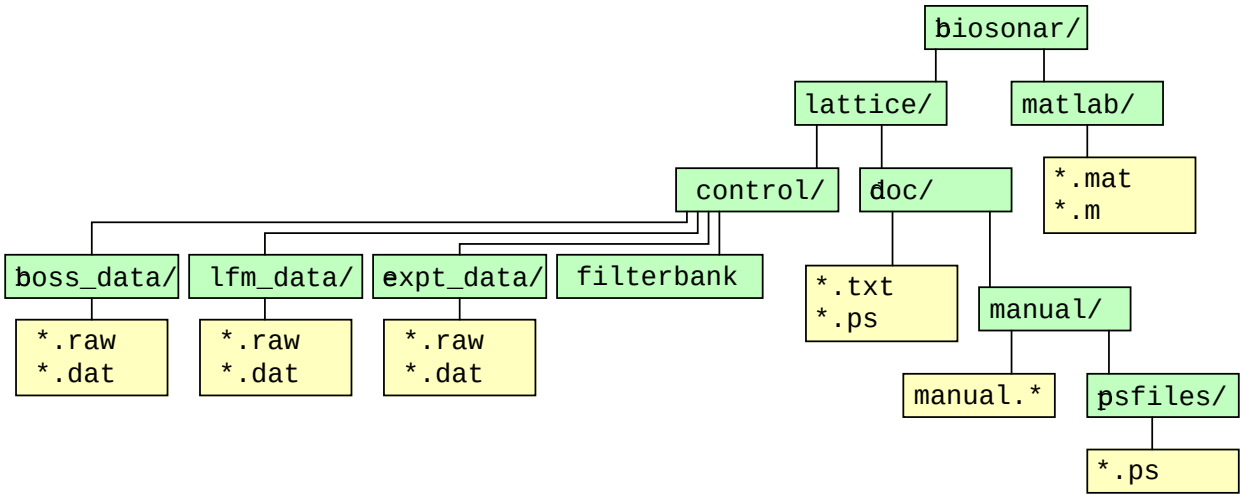


Figure 5.3: Biosonar source code directory hierarchy.