

Appendix A

LazyFish Technical Documentation

A.1 LEDs

The LazyFish has two sets indicator of LEDs, one set on the sensor subsection, and one set for the RS-232 interface. These can be very helpful in debugging Lazyfish applications. On both boards, the amber LED shows that power is present, the green indicates incoming data from the host, and the red LED is for outgoing data. The data LEDs are tied directly to the communication lines and flash on when a one bit is present on the line. When the board is being polled by a computer, the red LED will not typically flash as brightly as the green, because the poll command is a single byte, while the Lazyfish's response is many bytes. Thus the resulting duty cycle for the red LED is much lower.

If either portion of the Lazyfish is being mounted in a case, the surface mount indicator LEDs may be removed, and panel or case-mounted LEDs used instead. Underneath each LED is a pair of holes for soldering wires leading to off-board panel-mounted indicator LEDs.

A.2 Connectors

A.2.1 Interface board

The RS-232 portion of the Lazyfish has a DB-9 serial connector, power jack, and stereo audio jack. The stereo audio jack carries two TTL level data lines (for data headed to the host and from the host), plus ground. When the sensing portion of the board is split from the RS-232 interface, a stereo cable can be used to connect the two portions. There are six holes (with standard .1" inter-hole spacing) on the RS-232 portion and 6 corresponding holes on the sensing portion that can be stuffed with a 6 pin header. When the two boards are mounted in the "sandwich" configuration, the header can be used to join the two boards. When the sensing board is in the "remote" configuration, rather than using the stereo cable to connect the two boards, a ribbon cable connecting these headers could be used. The advantage over the stereo jack is that the interface board can also supply power to the sensing board this way, for cases in which there is no local power supply for the sensing board. The power switch on the interface board cuts the power to both boards when the sensor board is getting its power from the interface unit. If the sensor board is being powered from its own battery, you will probably want to add an additional, external power switch for the sensor board.

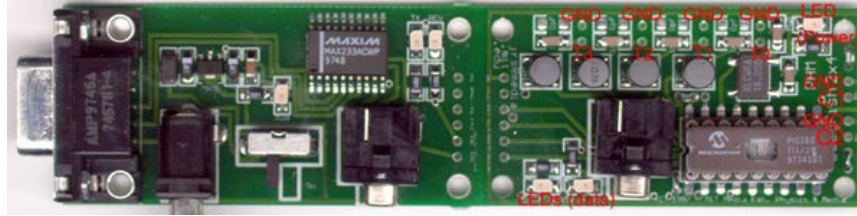


Figure A-1: This figure shows the board connectors.

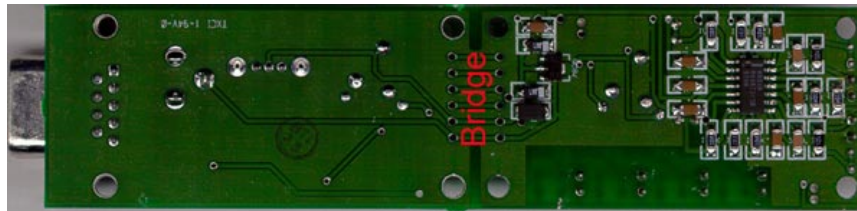


Figure A-2: The bottom of the Lazyfish board (integrated configuration), showing the bridge connecting the two halves.

A.2.2 Bridge

The bridge is the set of six lines connecting the interface board to the sensing board. There is a hole on each side of each bridge line, for soldering wires or headers. When the bridge is broken to separate the boards, these holes may be used to connect the two halves. The six bridge lines are: power (unregulated), ground, data TX, data RCV, DIG1, and DIG2. The power line is the raw, unregulated power supplied to the RS-232 board. The sensing board has its own voltage regulation and power conditioning so that it can be run from a local battery if desired. The labels TX and RCV are from the computer's perspective...if you consider the RS-232 interface to be a part of the computer, you could argue that this is a feature. Even if you disagree, that's the way it is. Two unused bi-directional PIC digital lines are broken out for easy access on the DIG1 and DIG2 lines. These lines do not connect to anything on the RS-232 board, but the holes on the RS-232 side of the bridge were included for mechanical purposes: a six pin header can be used to connect the boards in the sandwich configuration.

A.2.3 Sensing board

Two of the PIC pins connected to ADC channels are unused, and these pins have also been broken out to their own holes. These two pins can also be configured as configured as bi-directional digital lines if desired. See figure [] for the location of these holes.

Four wire loops, intended for attaching scope probes to monitor the behavior of the analog front end, are also included. There are two front end channels, each with two gain stages, and a wire loop for each gain stage. The plated-through screw holes in the corners of the board are all grounded and may be used as attachment points for the oscilloscope probe ground. Ordinarily, one would check the output of the second gain stage to determine whether the front end is clipping. Checking the output of the first gain stage would be rarer, probably only for investigating possible hardware problems.

There are a variety of holes for attaching transmit and receive electrodes. Each electrode hole has an associated ground hole spaced .1" away for connecting a shield. If the Lazyfish

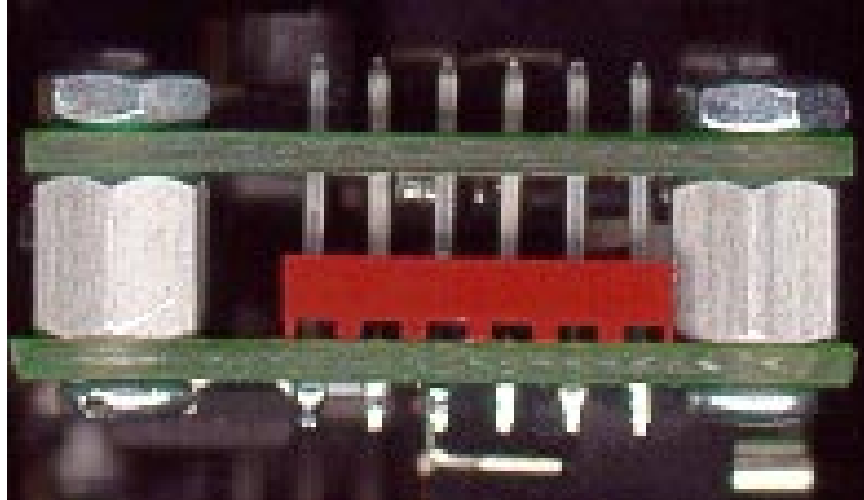


Figure A-3: The two halves of the Lazyfish connected by a header in the stacked configuration.

is being deployed in such a way that the electrode cables may move, or need to be changed occasionally, I recommend soldering in a two-pin molex header in each, as shown in figure []. The transmit and receive electrode locations are labeled in figure []. There is enough clearance on the board to use the locking Molex connectors for the receive electrodes. The locking connectors can be used for the transmit electrodes too, though the connectors will sit on top of some surface mount components.

A.3 PIC pin assignments

- Pin 1 PIN A2 R2—front end RCV chan 2
- Pin 2 PIN A3 R3—uncommitted ADC channel 3
- Pin 3 VREF
- Pin 4 MCLR
- Pin 5 VSS (GND)
- Pin 6 PIN B0 DATARCV—serial data RCV
- Pin 7 PIN B1 DATATX—serial data TX
- Pin 8 PIN B2 DIG2—uncommitted digital pin
- Pin 9 PIN B3 DIG1—uncommitted digital pin
- Pin 10 PIN B4 TX3—resonant TX 3
- Pin 11 PIN B5 TX2—resonant TX 2
- Pin 12 PIN B6 TX1—resonant TX 1
- Pin 13 PIN B7 TX0—resonant TX 0
- Pin 14 VDD (+5V)
- Pin 15 OSC2
- Pin 16 OSC1
- Pin 17 PIN A0 R0—uncommitted rcv chan 0
- Pin 18 PIN A1 R1—front end RCV chan 1

A.4 Communications protocol and commands (code version LZ401)

The default communication protocol shipped with the LazyFish is 38400 baud, 8 bits, no parity, 1 stop bit (38.4K baud, 8N1). To poll the LazyFish, send either an ASCII W or ASCII R. After an R, the LazyFish will respond with a 48-byte string representing 8 decimal numbers between 0 and 65535. These 8 numbers are all the measurements the LazyFish can make: there are 4 transmitters and 2 receivers, and the eight numbers are all the transmit-receive pairs. The W command returns 4 numbers (24 bytes), representing the signal on the currently selected receive channel due to each of the four transmitters.

Protocol: 38.4K baud, 8N1

Your Command	Meaning	LazyFish Response (e.g.)
R	Read all 8 chans	00000 11111 22222 33333[CR] 44444 55555 66666 77777[CR]
W	Read 4 chans	00000 11111 22222 33333[CR]
S	Read Single Chan	65535[CR]
T	Test command...get raw samples	00255 00255 00255 00255[CR]
C0	Read from unused ADC chan 0	None
C1	Change to RCV chan 1	None
C2	Change to RCV chan 2	None
C3	Read from unused ADC chan 3	None
Iyz	Change integration parameters	None
X0	Change to TX chan 0	None
X1	Change to TX chan 1	None
X2	Change to TX chan 2	None
X3	Change to TX chan 3	None
U	Read 4 chans, return binary result on DIG2	<i>aabbccdd</i> (4 two-byte values)
Y	Read 4 chans, return binary result	<i>aabbccdd</i> (4 two-byte values)
V	Get code version	LZ401

A.4.1 R command

Read all eight channels (all measurements made by pairing the 4 transmitters with 2 receivers). This command is not affected by the C or X commands: it automatically manages assignment of the transmit and receive channels. The values are 5 byte decimal ASCII representations of 16 bit numbers, so the range of values is 0 to 65535. Each value is followed by a space, except for the 4th and 8th value, which are followed by carriage returns.

A.4.2 W command, C command

Read 4 channels (each of the 4 transmitters paired with the currently selected receive channel). This command is not affected by the X command since it automatically chooses the transmitters, but is affected by the C command, which chooses the receive channel. The LazyFish has two analog front end receive channels, which can be selected using C1 or C2. The commands C0 and C3 select two uncommitted ADC channels (with no op-amp front

ends). To use the LazyFish as a data acquisition board (to measure a DC voltage to and return it to the computer), you'd use C0 or C3, followed by the T command.

A.4.3 S command

Read a single channel. Uses the currently selected transmitter or transmit mask, and the currently selected receive channel. The transmit channel (or channels) is selected by the X command (whichever was issued last). The receive channel is determined by the C command.

A.4.4 T command

The main purpose of the T command is to check whether the front end is clipping. This command returns 4 raw ADC samples ($0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}$). The range of values is 0 to 255. For consistency with other commands, these are 5 byte decimal ASCII numbers (so the first two digits of each number are 0). If any of the 4 values returned by this command are 0 or 255, the front end is clipping. To eliminate clipping, the transmit burst length parameter z can be decreased, or the electrode geometry can be changed. To eliminate clipping by changing the electrode geometry, the electrode sizes can be decreased, the distance between the electrodes can be increased, or a grounded shield can be placed in the vicinity of the electrodes.

A.4.5 I command

The parameters y and z are each single bytes. The default values of y and z are 20 and 7. The parameter y specifies the number of measurements to be integrated each time the LazyFish receives a poll command. The z value is the length (in periods) of the transmit burst used to excite the resonant transmitter. If the receive front end is clipping (which can be determined by connecting an oscilloscope to test points on the board, or using the T command), this value should be decreased. The shorter excitation burst will prevent the transmitter from ringing all the way up to its maximum 80V peak-to-peak swing, and thus should prevent the receive front end from clipping.

The total measurement time is proportional to the product $y * z$. Increasing y will decrease the measurement noise, as will increasing z (up to the point that the front end clips). But the increased measurement time means that the sensor update rate is lower. The default parameter values represent a good tradeoff between update rate and measurement noise.

A.4.6 X command

This selects a transmitter. X0 picks transmitter 0, X1 picks transmitter 1, and so on. Since the R and W commands reset the transmit channel automatically, X does not affect R or W. Modifying the selected transmit channel does affect S and T, however.

A.4.7 Y command

Same as the W command, but the data is returned in binary rather than ASCII form. Since it returns a 16 bit value for each of the 4 channels, it returns 8 bytes (high byte, low byte, high byte, low byte, high byte, low byte, high byte, low byte).

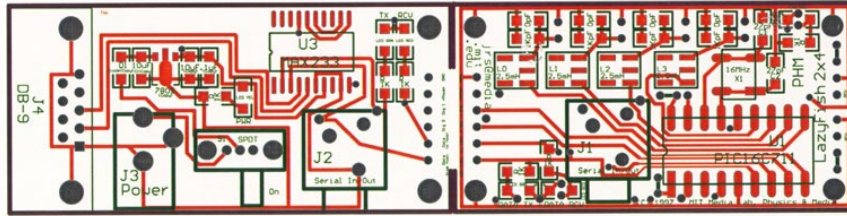


Figure A-4: Top of LazyFish.

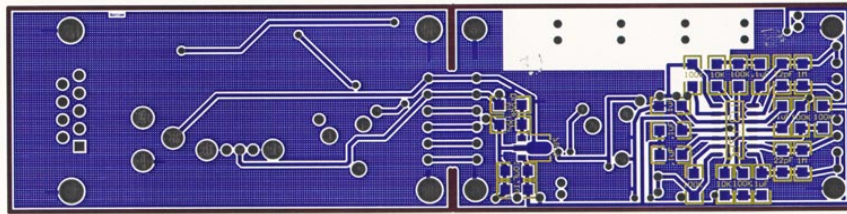


Figure A-5: Bottom of LazyFish

A.4.8 U command

Same as the Y command (4 channels returned in binary format), but the data is returned on the DIG2 line rather than the usual line used for serial communication. This function can be useful when the LazyFish is attached to both a PC and another microcontroller. The binary format is typically easier for another microcontroller to interpret (since it doesn't have to bother converting from ASCII). With the DIG2 line, the PC doesn't have to "hear" the data that the LazyFish is sending back to the other device.

A.4.9 V command

Returns code version. Current code version is LZ401.

A.5 LazyFish Front End Application

The LazyFish front end application can be used exercise most of the LazyFish commands, demonstrate the basic capabilities of the device, and debug electrode geometries. The main interface elements are the sliders, which are used to display sensor values. Each slider is labelled with the combination of transmitter and receiver it represents. For example, the first slider is labelled "TOR1," for transmit 0 and receive 1.¹

The controls are grouped into various logically related frames. Interface objects that correspond closely to a LazyFish command are labelled with the command name, so that one can use the front end program to learn the commands. The frame labelled "Sensing Mode," for example, contains a set of radio buttons that determine which one of the various LazyFish Sensing commands should be used to update the display. In "Off" mode, the

¹The LazyFish transmit channels start at 0, while the receive channels start at 1. This numbering system reflects the PIC's hardware (receive channel 1 corresponds to analog input 1). Having forgotten entirely about the hardware and confused myself many times while writing software, I consider this inconsistency of numbering schemes to be a dreadful mistake.

LazyFish is not being operated at all, and the display is static. In “1 Channel” mode, the S command is used to poll a single channel. Which channel is polled is determined by the parameters in the frame labelled “One Channel:” the contents of the “TX Chan” text box determine the transmit channel, and the “RCV Chan” box choose the receive channel. The sensor value is plotted on the appropriate slider. The “4 Channels” mode uses the W command to read out the bank of 4 channels associated with the currently selected receiver (as determined by the “RCV Chan” text box). The “8 Channels” mode makes no reference to the “TX Chan” or “RCV Chan” text boxes, because it reads out all the channels.

The “1 chan test” command is somewhat different than the others. It uses the “T” command, which returns raw in-phase and quadrature samples, with no integration. In the usual sensing modes, the LazyFish takes the magnitude in software. Also in these modes, multiple samples are taken and summed before returning the value. (The number of samples taken is specified by the “Int Time” text box in the “LazyFish Integration Parameters” frame.) In the T mode, the magnitude is not taken in the firmware, and the measurement is not repeated. The 4 raw values (0, 90, 180, and 270 degrees) are displayed as text in the Diagnostic frame, and also plotted as an XY plot. One purpose of the XY plot (and of the T mode) is to allow the user to see whether the sensor values are clipping, without using a scope. Basically it lets the LazyFish function as a scope. If any of the values are clipped (which occurs when a displayed vector touches the outer borders of the box), then the electrode geometry may be changed, or the transmit amplitude may be reduced by shortening the transmit square wave burst, as explained in Chapter 2.

The “Tx time” text box in the “LazyFish Integration Paramters” frame determines the length of the transmit burst. Decreasing this value should decrease the magnitude of the vectors plotted in the Diagnostic scope display, which can be used to eliminate clipping. The other text box in the Integration frame, “Int Time,” has no effect on the T command, only on the usual commands that operate the sliders because these other commands integrate the results of multiple measurements before returning a sensed value.

On startup, and whenever the integration parameters change, one should measure baseline values for all the channels by clicking the “Get Baseline” button several times. Nothing should be in the sensing field when the baseline is taken. The Get Baseline button measures all the channels once using the current integration parameters, and subtracts future sensed values from this baseline. The reason this button should be clicked several times is that the VB front end performs some low-pass filtering of the sensed values (including those used to take the baseline), so a few samples are needed for the baseline to converge correctly. Once a baseline has been established, the sensor value should be zero for each channel when nothing is in the vicinity of the sensors.

The filtering mentioned above is controlled by the Alpha text box in the “Host Display Parameters” frame. When Alpha is 1, the sensed values are not low-pass filtered at all. When Alpha is 0, new data does not affect the displayed values at all. The filter is of the form $y = \alpha x + (1 - \alpha)y'$, where x is the latest data, y is the new displayed value, and y' is the previous displayed value. The default value of Alpha is .5.

The “Host Display Parameters” frame also contains the “Slider Min” and “Slider Max” text boxes, which allow the lowest and highest value on the slider to be set arbitrarily (with certain restrictions, like the Max has to be greater than the Min). Typically, Slider Min is left at its default value of 0, since if a baseline has been taken, the minimum sensor value is 0. It is usually necessary to adjust Slider Max when the integration parameters are changed, because increasing the integration time increases the size of the maximum possible deviation from the baseline (since each measurement is the sum of more numbers).

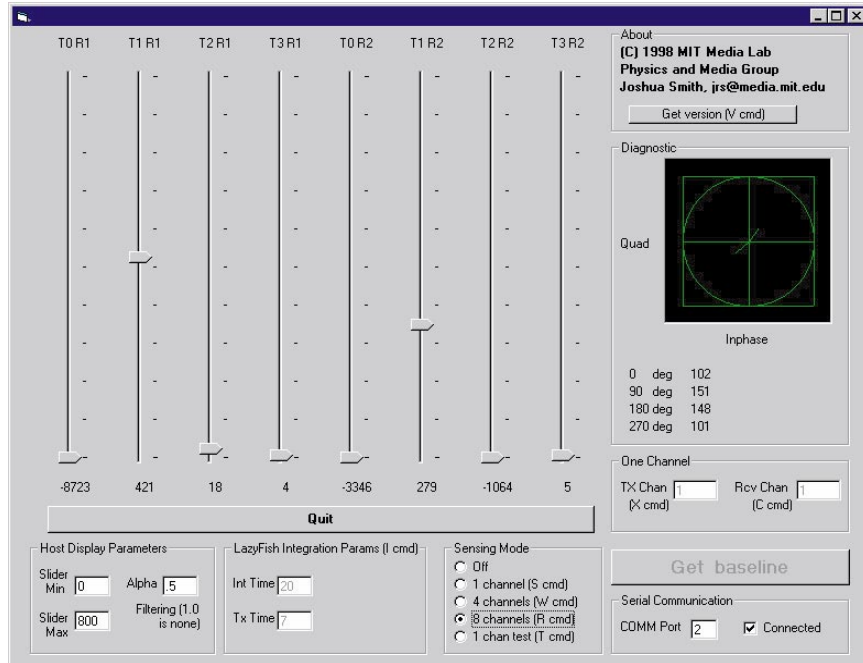


Figure A-6: The LazyFish front end application, oct.exe.

A.6 LazyFish Firmware (version LZ401)

```
// Joshua R. Smith
// MIT Media Lab (c) 1998
//
// LZ401      Some opts to shrink code
// LZ39C711   Fixed TRIS on ADC indicator pin
// LZ38C711   Try waiting for TX to die down. Got rid of outer int loop.
// LZ37C711   Use sense to implement test;
//            do abs(sum(i))+abs(sum(q)) instead of
//            sum(abs(i))+sum(abs(q)) [faster, OK if is phase const]
// LZ36C711   Added binary output mode
// LZ35C711   Fixed bad sensing problem. Made sense() consistent
//            with test()
// LZ34C711   Added U (url) and V (version) cmds. Changed naming convention.
// lazy33     Added single channel read cmd;
//            Improved timing in main sense command as lazy32 did for test cmd.
// lazy32     In test mode, added spikes on dig1 to indicate when sampling occurs
//            Improved timing in test cmd by using DELAY_CYCLES instead of DELAY_US
// lazy31     switch to C711
// lazy30     added code to switch TX, and code to give user direct control over TXMASK
// lazy29     added test code to detect saturation / measure phase
// lazy28     added cmd to read all 8 chans; moved sense code from include into this file
// lazy27     Use all 4 TXs, by compressing code, by not using OUTPUT_HI on tx
// lazy26     Fix multiple rcv channels
// lazy25     Delayed powerup message so it doesn't get stomped on by FishFace
// lazy24     Change default ADC chan; reverse order of txs
// lazy23     Fixed bug in 20: clear registers before sensing
// lazy20     Don't use U as cmd
// lazy19     Made tx burst length a variable
//            Switched to 711
//            Made sense routine an include; took out software filtering (not enough ram)
// lazy18     Switch to using longs; do software filtering
// lazy17     Add inphase & quad together
// lazy16     Made mode w/ no spike on TX2; added delay
//            between bursts 2 and 3 to keep phase consistent;
//            added hex output; changes sign of hex quad output to\
//            make it positive
//            100kHz This is debug version: makes spike on TX2,
//            outputs raw measured values
// lazy15     four channels
// lazy14     quadrature
// lazy13     Separate inner and outer integration loop
// lazy12
// lazy3.c    Data out serial port
// lazy2.c
// synch3.c   Made integration time adjustable param
// synch2.c   Split fish read into separate routine
// based on
```

```

// synchl.c First working Laz-i-fish code

#byte portb = 6

#include <16C711.h>
//#include <stdio.h>
#include <math.c>

#fuses HS,NOWDT,NOPROTECT,BROWNOUT,PUT
#USE Delay(Clock=16000000)
//#USE FAST_IO(a)
//#USE FAST_IO(b)

// pin 17
#define R0 PIN_A0
// pin 18
#define R1 PIN_A1
// pin 1
#define R2 PIN_A2
// pin 2
#define R3 PIN_A3

// Pin 6
#define RX PIN_B0
// Pin 7
#define TX PIN_B1
//Pin 8 binary data to eg FishFace
#define DIG2 PIN_B2
//Pin 9 ADC indicator pin
#define DIG1 PIN_B3

// pin 10
#define TX3 PIN_B4
// pin 11
#define TX2 PIN_B5
// pin12
#define TX1 PIN_B6
// pin 13
#define TX0 PIN_B7

#define TX3MASK 0x10
#define TX2MASK 0x20
#define TX1MASK 0x40
#define TX0MASK 0x80

#USE RS232 (Baud=38400, Parity=N, Xmit=TX,Rcv=RX)

#define hi(x) (*(x+1))
#define lo(x) (*(x))

byte G_inttime,G_txttime;
byte G_it_old;
long i_adc;
long q_adc;
byte cmd;
byte txmask;
byte chanoff;
//byte binmode;

long i2_adc;
long q2_adc;
long i1_adc;
long q1_adc;

long chan[8];

void initialize() {
    OUTPUT_HIGH(TX);

    SET_TRIS_A(0xFF); // WAS 0X0F
    //SET_TRIS_B(0xF6); // TXs, RX, TX and DIGIN are tristated
    //SET_TRIS_B(0xFF); // everything tristated
    SET_TRIS_B(0xFB); // everything tristated but DIG2
    SETUP_PORT_A(ALL_ANALOG);
    SETUP_ADC(ADC_CLOCK_DIV_32);
    SET_ADC_CHANNEL(1);

    i_adc = 0;
    q_adc = 0;
    chanoff = 0;
    G_inttime = 20;
    G_txttime = 7;
    OUTPUT_HIGH(DIG2);
    // binmode = 0;
}

void putdecl(long n) {
    putchar( (n/10000) +'0');
    putchar( ((n/1000) % 10) +'0');
    putchar( ((n/100) % 10) +'0');
}

```

```

putchar( ((n/10) % 10) + '0');
putchar( (n % 10) + '0');
}

outd4() {
byte c;
for (c=chanoff; c<chanoff+3; c++) {
putdecl(chan[c]);
putc(' ');
}
putdecl(chan[c]); // no space
putc('\r');
}

outb4d2() {
byte c;
#USE RS232 (Baud=38400, Parity=N, Xmit=DI02,Rcv=RX)
for (c=chanoff; c<chanoff+4; c++) {
putc(hi(chan[c]));
putc(lo(chan[c]));
}
#USE RS232 (Baud=38400, Parity=N, Xmit=TX,Rcv=RX)
}

outb4() {
byte c;
for (c=chanoff; c<chanoff+4; c++) {
putc(hi(chan[c]));
putc(lo(chan[c]));
}
}

sense() {
byte i, j, k;
byte adcval;
i_adc = 0; // Erase old value of adc...cut if we're doing low pass filter
q_adc = 0;
for (j=0; j < G_inttime; j++) {
SET_TRIS_B(0x02); // UNTRIS TXs, UNTRIS DI02, TRIS TX
for (k=0; k < G_txtime; k++) {
portb = txmask;
DELAY_CYCLES(18);
portb = 6; // 4+2
DELAY_CYCLES(10);
}
//DELAY_CYCLES(10);
DELAY_CYCLES(6); // reduced by 4 to allow for ADC indicator
portb = 14; // ADC indicator 8+4+2
portb = 6; // 4+2
adcval=READ_ADC(); // phase pi/2
SET_TRIS_B(0xF3); // RE-TRIS TXs to discharge tank
q1_adc = adcval;
q_adc = q_adc+q1_adc;
DELAY_US(15); // time for tank to discharge

SET_TRIS_B(0x02); // UNTRIS TXs, UNTRIS DI02, TRIS TX
for (k=0; k < G_txtime; k++) {
portb = txmask;
DELAY_CYCLES(18);
portb = 6; // 4+2
DELAY_CYCLES(10);
}
//DELAY_CYCLES(20);
DELAY_CYCLES(16); // reduced by 6 to allow for ADC indicator
portb = 14; // ADC indicator 8+4+2
portb = 6; // 4+2
adcval=READ_ADC(); // phase pi
SET_TRIS_B(0xF3); // RE-TRIS TXs to discharge tank
i2_adc = adcval;
i_adc = i_adc-i2_adc;
DELAY_US(15); // time for tank to discharge

SET_TRIS_B(0x02); // UNTRIS TXs, UNTRIS DI02, TRIS TX
for (k=0; k < G_txtime; k++) {
portb = txmask;
DELAY_CYCLES(18);
portb = 6; // 4+2
DELAY_CYCLES(10);
}
//DELAY_CYCLES(30);
DELAY_CYCLES(26); // reduced by 4 to allow for ADC indicator
portb = 14; // ADC indicator 8+4+2
portb = 6; // 4+2
adcval=READ_ADC(); // phase 3pi/2
SET_TRIS_B(0xF3); // RE-TRIS TXs to discharge tank
q2_adc = adcval;
q_adc = q_adc-q2_adc;
DELAY_US(15); // time for tank to discharge

SET_TRIS_B(0x02); // UNTRIS TXs, UNTRIS DI02, TRIS TX

```

```

    for (k=0; k < G_txtime; k++) {
        portb = txmask;
        DELAY_CYCLES(18);
        portb = 6;           // 4+2
        DELAY_CYCLES(10);
    }
    //DELAY_CYCLES(40);
    DELAY_CYCLES(36);      // reduced by 4 to allow for ADC indicator
    portb = 14;           // ADC indicator 8+4+2
    portb = 6;           // 4+2
    adcval=READ_ADC();    // phase 2pi
    SET_TRIS_B(0xF3);     // RE-TRIS TXs to discharge tank
    i1_adc = adcval;
    i_adc = i_adc+i1_adc;
    //DELAY_US(5);
}
//SET_TRIS_B(0xF6);     // RE-TRIS TXs
SET_TRIS_B(0xFB);      // RE-TRIS TXs, but don't TRIS DIG2

if (i1_adc < i2_adc) { // need to invert to take abs?
    i_adc = 65535-i_adc;
}
if (q1_adc < q2_adc) { // need to invert to take abs?
    q_adc = 65535-q_adc;
}

i_adc = i_adc+q_adc;
}

read4_ef() {
byte i;
byte c;
byte txmaskind;
txmaskind = TXOMASK;
txmask = txmaskind+6; //4+2

for (c=chanoff; c<chanoff+4; c++) {
    sense();
    chan[c] = i_adc;
    txmaskind = (txmaskind >> 1);
    txmask = txmaskind+6; //4+2
}
}

readoutd4() {
    read4_ef();
    outd4();
}

rc1() {
    SET_ADC_CHANNEL(1);
    chanoff = 0;
}

rc2(){
    SET_ADC_CHANNEL(2);
    chanoff = 4;
}

void main () {
    initialize();
    while(1) {
        cmd = getch();
        if (cmd == 'U') { // Read current bank of 4 in binary on DIG2
            read4_ef();
            outb4d2();
        }
        if (cmd == 'Y') { // Read current bank of 4 in binary
            read4_ef();
            outb4();
        }
        if (cmd == 'W') { // Read current bank of 4
            readoutd4();
            //printf("\x");
        }
        if (cmd == 'R') { // Read all 8
            rc1();
            readoutd4();
            //putc(' ');
            rc2();
            readoutd4();
            //putc('\x');
            chanoff = 0;
        }
        if (cmd == 'S') {
            sense();
        }
    }
}

```

```

    putdecl(i_adc);
    putc('\r');
}
if (cmd == 'C') {
    cmd = getc();
    if (cmd == '0') {
        SET_ADC_CHANNEL(0);
    }
    if (cmd == '1') {
        rc1();
    }
    if (cmd == '2') {
        rc2();
    }
    //if (cmd == '3') {
    //    SET_ADC_CHANNEL(3);
    //}
}
if (cmd == 'X') {
    cmd = getc();
    if (cmd == '0') {
        txmask = TX0MASK+6; //4+2
    }
    if (cmd == '1') {
        txmask = TX1MASK+6; //4+2
    }
    if (cmd == '2') {
        txmask = TX2MASK+6; //4+2
    }
    if (cmd == '3') {
        txmask = TX3MASK+6; //4+2
    }
}
//if (cmd == 'K') {
//    txmask = getc();
//}
if (cmd == 'T') {    // Test to check for ADC saturation
    G_it_old = G_inttime;
    G_inttime = 1;
    sense();
    chan[chanoff+0] = q1_adc;
    chan[chanoff+1] = i2_adc;
    chan[chanoff+2] = q2_adc;
    chan[chanoff+3] = i1_adc;
    outd4();
    //putc('\r');
    G_inttime = G_it_old;
}
if (cmd == 'I') {
    G_inttime = getc();
    G_txttime = getc();
}
if (cmd == 'V') {
    printf("LZ401\r");
}
//if (cmd == 'B') {
//    binmode = 1;
//}
//if (cmd == 'A') {
//    binmode = 0;
//}
//if (cmd == 'U') {
//    printf("http://jrs.www.media.mit.edu/~jrs/lazydrv\r");
//}
SET_TRIS_B(0xFB);
}
}

```