

```

package nmc.sidekick;
import com.ajile.jem.rawJEM;
import com.ajile.drivers.spi.*;

public class Adc {

    /**
     * Starts the application.
     * @param args an array of command-line arguments
     */
    public static void main(java.lang.String[] args) {

        Adc adc = new Adc();

        adc.initSpiForAdcTest(); // duplicates other code in full sidekick

        adc.initAdcCsN(); // but this needs to be done in real code

        int r = 0;
        while (true) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println("got InterruptedException: " + e);
            }
            r = adc.getReading(2);
            System.out.println("adc Reading was " + r + " = 0x"
                               + Integer.toHexString(r));
        }

        static final int ADCCSN = (1<<4);

        private void initSpiForAdcTest() {
            spi.setClkDivider(spi.SPI_CLOCK_DIVIDER_64);
            spi.setSlaveSelect(spi.SPI_SLAVE_SELECT_NONE);
        }

        // The following is NOT THREAD-SAFE
        // If anything else is doing a read-modify-write of regE at the same time,
        // Heisenbugs are likely to appear
        //
        public void initAdcCsN() {
            // set the value of AdcCsN to 1 (chip select not asserted)
            int regEOut = rawJEM.getInt(AjileAddr.GPIOE_OR);
            regEOut |= ADCCSN;
            rawJEM.set(AjileAddr.GPIOE_OR, regEOut);

            // now enable the output driver on the pin connected to AdcCsN (E4)
            int regEEn = rawJEM.getInt(AjileAddr.GPIOE_OER);
            regEEn |= ADCCSN;
            rawJEM.set(AjileAddr.GPIOE_OER, regEEn);
        }

        protected SpiMaster spi = SpiMaster.getInstance();

        // There should be some mutual exclusion between this and the PtSensor
        // (and anything else that does a read-modify-write on REG_E, especially
        // in an interrupt handler, as the PtSensor does).

```

```

    int getReading(int channel) throws IllegalArgumentException {
        int dataIn;
        if (channel > 2) {
            throw new IllegalArgumentException("Channel " + channel
                                              + " doesn't exist on the adc");
        }
        channel |= 0x8; // set the EN bit; must be set to take reading
        int retval;
        int high;
        int low;
        int regE;
        synchronized (spi) {
            spi.setSlaveSelect(spi.SPI_SLAVE_SELECT_NONE); // just in case
            spi.writeReadData(channel); // with AdcCsN high
            regE = rawJEM.getInt(AjileAddr.GPIOE_OR);
            regE &= ~ADCCSN; // set E4 low, leave everything else alone
            rawJEM.set(AjileAddr.GPIOE_OR, regE);

            high = spi.writeReadData(0); // data sent to ADC is ignored
            low = spi.writeReadData(0);
            // remember the high bit of low is sign-extended to 32 bits
        } // we now let go of the spi so someone else can use it

        // we could do all the stuff below in one line, but this is
        // easier to read
        //
        retval = (0x1f & high);
        retval <= 8;
        retval |= (0xff & low);
        retval >>= 1;

        regE = rawJEM.getInt(AjileAddr.GPIOE_OR);
        regE |= ADCCSN; // set E4 high again
        rawJEM.set(AjileAddr.GPIOE_OR, regE);

        return retval;
    }
}

```

```
package nmc.sidekick;

// This intentionally has package scope. Unless there is a good reason
// to allow another package to become dependent on the internal implementation
// of the ajile processor, this should be invisible to anything outside of
// the sidekick package.
//
class AjileAddr {
    public static final int IOCR = 0xffff00a0;

    public static final int GPTC_GMR      = 0x09001000;
    public static final int GPTC_IOMR     = 0x09001004;
    public static final int GPTC_IOPR     = 0x09001008;
    public static final int GPTC_ISR      = 0x09001010; // Interrupt Status
    public static final int GPTC_PRLR     = 0x09001014;

    public static final int GPTC0_MR      = 0x09001018;
    public static final int GPTC0_RLR     = 0x0900101c;
    public static final int GPTC0_STICR    = 0x09001028;

    public static final int GPIOA_OER     = 0x09002000;
    public static final int GPIOA_OR      = 0x09002004;
    public static final int GPIOA_IR      = 0x09002008;
    public static final int GPIOA_POLARITY = 0x09002010;
    public static final int GPIOA_LEVEL_EDGE = 0x09002014;
    public static final int GPIOA_EDGE_MODE = 0x09002018;
    public static final int GPIOA_INT_ENABLE = 0x0900201c;
    public static final int GPIOA_PENDING = 0x09002020;
    public static final int GPIOA_CLEAR_INT = 0x09002024;

    public static final int GPIOB_OER     = 0x09002100;
    public static final int GPIOB_OR      = 0x09002104;
    public static final int GPIOB_IR      = 0x09002108;

    public static final int GPIOC_OER     = 0x09002200;
    public static final int GPIOC_OR      = 0x09002204;
    public static final int GPIOC_IR      = 0x09002208;

    public static final int GPIOD_OER     = 0x09002300;
    public static final int GPIOD_OR      = 0x09002304;
    public static final int GPIOD_IR      = 0x09002308;

    public static final int GPIOE_OER     = 0x09002400;
    public static final int GPIOE_OR      = 0x09002404;
    public static final int GPIOE_IR      = 0x09002408;

    public static final int ADCCSN        = (1<<4); // GPIO bank E bit 4
}
```

sidekick\CommandParser.java

```

package nmc.sidekick;

// 200205011630

import java.io.*;
import javax.microedition.io.Connection;
import javax.microedition.io.Connector;
import javax.microedition.io.Datagram;
import javax.microedition.io.DatagramConnection;
import com.ajile.drivers.spi.*;
import javax.comm.SerialPort;
import javax.comm.CommPortIdentifier;

//import java.io.DataOutputStream;
//import java.io.ByteArrayOutputStream;
//import java.io.ByteArrayInputStream;
//import java.io.DataInputStream;

import nmc.common.*;

public class CommandParser {

    SystemHardwiredInfo settings = SystemHardwiredInfo.Current;

    public IoResource[] channel;

    /**
     * Starts the application.
     * @param args an array of command-line arguments
     */
    public static void main(java.lang.String[] args) {
        CommandParser cp = new CommandParser();
        cp.loadAndGo(); // this is still the main thread;
        // we just transition to working on an instance
        // I am not sure whether the main thread should be
        // treated specially or not... at this point,
        // I am not treating it specially, and from here
        // it will go on to wait for a command, look it
        // up, and do it.
    }

    // This does not inherit from Runnable, so it is not a "real" run...
    public void loadAndGo() {

        channel = new IoResource[60]; // initial values are nulls

        initializePlatform();

        //      byte[] configOut =
        //      "first line\r\nsecond line\r\nthird line".getBytes();

        //      ConfigurationCmd cfgCmd1
        //      = new ConfigurationCmd(0, ConfigurationType.INSTRUMENT, configOut);

        //      System.out.println("ConfigurationCmd originally=" + cfgCmd1.toString());

        //      try {
        //          byte[] cfgWheatiesCmd = cfgCmd1.cerealize();
        //          ByteArrayInputStream cfgBisCmd
        //          = new ByteArrayInputStream(cfgWheatiesCmd);

        //          DataInputStream cfgDisCmd = new DataInputStream(cfgBisCmd);
        //          int msgTypeCmd = cfgDisCmd.readInt();
        //          System.out.println("message type is " + msgTypeCmd);
        //          ConfigurationCmd cfgCmd2 = new ConfigurationCmd(cfgDisCmd);
        //          System.out.println("ConfigurationCmd Post Wheaties="
        //              + cfgCmd2.toString());
        //      } catch (Exception e) {
        //          System.out.println("in loadAndGo (Cmd), caught exception=" + e);
        //      }

        //      try {
        //          byte[] configIn =
        //          "Three\r\ntwo\r\nnonezerofire\r\n".getBytes();
        //          System.out.println("ONE");
        //          TimestampedData td = new TimestampedData(configIn);
        //          System.out.println("TWO");
        //          ConfigurationReply cfgReply1 = new ConfigurationReply(cfgCmd1, td);
        //          System.out.println("cfgReply1=" + cfgReply1.toString());

        //          byte[] cfgWheatiesReply = cfgReply1.cerealize();
        //          System.out.println("THREE");
        //          ByteArrayInputStream cfgBisReply
        //          = new ByteArrayInputStream(cfgWheatiesReply);
        //          System.out.println("FOUR");
        //          DataInputStream cfgDisReply = new DataInputStream(cfgBisReply);
        //          System.out.println("FIVE");
        //          int msgTypeReply = cfgDisReply.readInt();
        //          System.out.println("message type is " + msgTypeReply);
        //          ConfigurationReply cfgReply2 = new ConfigurationReply(cfgDisReply);
        //          System.out.println("SIX");
        //          System.out.println("ConfigurationReply Post Wheaties="
        //              + cfgReply2.toString());

        //      } catch (Exception e) {
        //          System.out.println("in loadAndGo (Reply), caught exception=" + e);
        //      }

        OutboundMessageQ outbound = null;
        InboundMessageQ inbound = null;
        // Set up the outbound and inbound queues
        try {
            System.out.println("Creating outbound message queue...");
            System.out.println("    sidearmIpAddr="
                + settings.sidearmIpAddr
                + "; sidearmRcvPort="
                + settings.sidearmRcvPort);
            outbound = new OutboundMessageQ(settings.sidearmIpAddr,
                settings.sidearmRcvPort);
            outbound.startOutboundMessaging();

            System.out.println("Creating inbound message queue...");
            System.out.println("    sidekickRcvPort="
                + settings.sidekickRcvPort);
            inbound = new InboundMessageQ(settings.sidekickRcvPort);
            inbound.startInboundMessaging();

        } catch (Exception e) {
            System.out.println("Caught exception while opening connection:"

```

```

        + e);
    }
    return;
}

int j = 0;
while(true) {
    Message m = inbound.blockingGet();
    System.out.println("Just returned from blocking get");
    int msgType = m.getMessageType();
    switch (msgType) {
    case Message.POLL_CMD: {
        System.out.println("detected poll cmd");
        PollCmd p = (PollCmd)m;
        IoResource ior = channel[p.channelNumber];
        Instrument dataSrc =
            (Instrument) ior.getResourceType(IoResourceType.INSTRUMENT);
        TimestampedData dataSample = null;
        try {
            dataSample = dataSrc.poll();
        } catch (Exception e) {
            // We have to create an error return to the
            // sidearm. The error return should contain a
            // string that explains something about the error,
            // and an error code, that puts the error in one
            // of a predefined set of categories so that the
            // sidearm can better deal with it.
            //
            // Error categories would include things such as
            // bufferOverflow, timeout, tooManyRetries, and
            // a catchall Unknown or Other.
            //
            System.out.println("Error during poll, error text=" + e);
        }
        PolledDataReply pdr = new PolledDataReply(m,dataSample);
        outbound.put(pdr);
        /*
        byte[] b2 = null;
        Message m2 = null;
        try {
            b2 = pdr.cerealize();
            m2 = DecerealizingFactory.parse(b2);
        } catch (IOException e) {
            System.out.println("bad cereal! " + e);
        }
        System.out.println("Reconstruction: " + m2.toString());
        */
    }
    break;
    case Message.INITIALIZE_INSTRUMENT_CMD: {
        System.out.println("detected INITIALIZE_INSTRUMENT_CMD");

        InitializeInstrumentCmd iic = (InitializeInstrumentCmd)m;
        System.out.println("    for channel number "
            + iic.channelNumber);

        // We cannot use that yet, but in the real system
        // the DPA board index is channelNumber >> 1 and the
        // low-order bit (channelNumber & 1) is the left/right
        // discriminator

        IoResource ior = channel[iic.channelNumber];

```

```

        DpaBoard.DpaChannel aChannel =
            (DpaBoard.DpaChannel)
            ior.getResourceType(IoResourceType.DPA_BOARD);
        System.out.println("Got the dpa channel, initializing it");
        aChannel.initializeChannel();

        InitializeInstrumentReply iir =
            new InitializeInstrumentReply(m, 90);
        outbound.put(iir);
        System.out.println("Just did outgoing "
            + "InitializeInstrumentReply"
            + iir.toString());
    }
    break;
    case Message.CONFIGURATION_CMD:
        System.out.println("detected CONFIGURATION_CMD");

        ConfigurationCmd cc = (ConfigurationCmd)m;
        // start
        IoResource ior = channel[cc.channelNumber];
        Instrument dataSrc =
            (Instrument) ior.getResourceType(IoResourceType.INSTRUMENT);
        TimestampedData cfgResults = null;
        try {
            cfgResults = dataSrc.configure(cc.data);
        } catch (Exception e) {
            //
            // Need to return a real exception message.
            // See issues noted above in comments under PollCmd.
            //
            System.out.println("Error during poll, error text=" + e);
        }
        ConfigurationReply pdr = new ConfigurationReply(m,cfgResults);
        outbound.put(pdr);
        // end

        try {
            byte[] configIn =
                "Three\r\ntwo\r\nnonezerofire\r\n".getBytes();
            System.out.println("ONE");
            TimestampedData td = new TimestampedData(configIn);
            System.out.println("TWO");
            ConfigurationReply cfgReply1 = new ConfigurationReply(m, td);
            System.out.println("cfgReply1=" + cfgReply1.toString());
            outbound.put(cfgReply1);
        } catch (Exception e) {
            System.out.println("in loadAndGo (Reply), caught exception=" +
                e);
        }

    }
    break;
    case Message.WAKEUP_CMD:
    {
        System.out.println("detected WAKEUP_CMD");

        WakeupCmd wc = (WakeupCmd)m;
        AckReply reply = new AckReply(wc);
        outbound.put(reply);
    }
    break;

```

```

        case Message.GET_ABSOLUTE_TIME_CMD:
        {
            System.out.println("detected GET_ABSOLUTE_TIME_CMD");
            GetAbsoluteTimeCmd g = (GetAbsoluteTimeCmd)m;
            long t = System.currentTimeMillis();
            System.out.println("\tcurrentTimeMillis() gives " + t);
            AbsoluteTimeReply r = new AbsoluteTimeReply(m, t);
            outbound.put(r);
        }
        break;
    default:
        System.out.println("unknown message received");
    }
    System.out.println("Around the loop " + j + " time(s).");
}

protected void initializePlatform() {
    /*
     * This is all the platform-specific configuration stuff.
     * Everything except the association between the instruments
     * and the port is likely to remain hardwired. The association
     * of instruments to serial lines must be configurable.
     */

    System.out.println("Starting 'initializePlatform()'");
    DpaBoard board = new DpaBoard(SpiMaster.SPI_SLAVE_SELECT_0);
    // the next line should not work
    board.getLeftChannel();
    DpaBoard.DpaChannel aChannel = board.getLeftChannel();
    System.out.println("Finished creating left dpa channel");
    aChannel.initializeChannel();
    System.out.println("Finished initializing left dpa channel");
    SerialPort cereal = null;
    try {
        cereal = openSerialPort("com1");
        aChannel.setSerialPort(cereal);
        System.out.println("Finished creating the serial port on com1");
        Instrument guitar = new MicroCat();
        guitar.setSerialPort(cereal);
        aChannel.setInstrument(guitar);
    } catch (IOException e) {
        System.out.println("could not open 'com1': " + e);
    }

    channel[0] = aChannel;
    System.out.println("Finished setting up left channel");

    aChannel = board.getRightChannel();
    System.out.println("Finished creating right dpa channel");
    aChannel.initializeChannel();
    System.out.println("Finished initializing right dpa channel");

    channel[1] = aChannel;
    System.out.println("Finished setting up right channel");
}

// this seems like it should go in some class of utility routines

```

```

protected SerialPort openSerialPort(String portName) throws IOException {
    CommPortIdentifier commPortId;
    SerialPort serialPort = null;
    try {
        commPortId = CommPortIdentifier.getPortIdentifier(portName);
        System.out.println("Whew, we got the port identifier for '"
            + portName + "'!");
        if (!commPortId.isCurrentlyOwned()) {
            System.out.println("Whew, no one owned " + portName);
            if (commPortId.getPortType() ==
                CommPortIdentifier.PORT_SERIAL) {
                System.out.println("Whew, com1 turns out to be a "
                    + "serial port...");
                try {
                    // the name seems to be completely meaningless.
                    // we use "xyz"
                    serialPort = (SerialPort)commPortId.open("xyz",
                        0);
                    // outStream = serialPort.getOutputStream();
                    // inStream = serialPort.getInputStream();
                } catch (javax.comm.PortInUseException e) {
                }
            }
        }
    } catch (javax.comm.NoSuchPortException e) {
        System.out.println("javax.comm.NoSuchPortException on 'com1'");
    }
    // This really should throw an exception if it cannot
    // create a fully functioning serial port.

    return serialPort;
}

```

06/13/02
20:06:58

sidekick\DataPort.java

1

```
package nmc.sidekick;  
  
public interface DataPort {  
    public void sendData(byte[] data);  
}
```

sidekick\DpaBoard.java

```

package nmc.sidekick;
import javax.comm.SerialPort;
import com.ajile.drivers.spi.*;
import java.lang.Thread;

// The DPA board looks like the following

/*
 * Major things not yet tested:
 * - taking a reading from the ADC
 * - taking all readings from the ADC
 * - turning each relay on, power cycle, first write is turn it off
 * - turning each relay off, power cycle, first write is turn it on
 * - turning each relay of, power cycle, first write is turn it off
 * - turning each relay on, power cycle, first write is turn it on
 * - I presume that interactions between relays are not significant
 * - getting an enabled interrupt
 * - getting an enabled interrupt and disabling it
 * - getting a disabled interrupt
 * - tripping the digital circuit breaker
 * - tripping the digital circuit breaker and resetting it
 * - repeatedly trip/reset the digital circuit breaker, deal with short
 * - various combinations of drive modes (unipolar/bipolar, RS-485, etc.)
 */

/*
- DPA
- Notes: In the listing below, single-bit register fields are only
  listed by their values for a '1' in that position. If a '0' in
  that position means something other than the simple negation of
  a '1', then the meaning of '0' is listed in the same line.
  There are only four items that are not single-bit fields: (1) the
  command (in all cases), (2) the power-down (or simply 'power') mode for
  an ADC Register Write, (3) the channel value for an ADC Register
  Write, and (4) the 'count', or number of clicks to move the
  potentiometer for a Dpot
  (Current Limit) Register Write. There are individual names for
  each possible value for the cmnd field and for the ADC power
  mode. The ADC Channel and the Dpot Count field are simply
  unsigned numbers, so only a mask value is specified for those.
- Defaults: Most of the values do not have meaningful defaults.
  Those that do have a '*' or a '!' before the value. An '*'
  means that the asterisked value is the default. A '!' means
  that a 0 in that bit position is the default.
- Summary of registers
  - ADC
  - Interrupt
  - Relays
  - Dpot
  - Channel Control
- DPA board-level commands (cmnd mask 0xf000, values:)
  - ADC
    - Notes: Programmer writes a command to the ADC and waits for
      the DPA's status to become not busy, which indicates
      that the command has been accepted and the conversion
      has been completed. After the DPA status becomes
      not busy, the programmer can read the ADC. If the power
      mode is power on, the command is also a conversion
      request; all bits are always meaningful, so every
      conversion request must include power mode, acquisition
      mode and drive types.
    - 0x4000 ADC Write (aka take reading)
      - Power Mode (mask 0x00c0, values:)
        - 0x0000: full power down [default when not sampling]
        - *0x0040: standby
        - 0x0080: power on / use internal clock [default when sampling]
        - 0x00c0: power on / use external clock
      - Acquisition Mode
        - !0x0020: External (0 means Internal)
      - SGL/DIF (drive type)
        - *0x0010: Single-ended (0 means Differential)
      - Uni/Bi (drive polarity)
        - *0x0008: Unipolar (0 means Bipolar)
      - Channels (mask 0x0007, values:)
        - 0x0007: Battery Supply Voltage (Vbat = 5.8*Vin)
        - 0x0006: Right Channel voltage sense (Vinstr=6*Vin)
        - 0x0005: Right Channel trip level sense
        - 0x0004: Right Channel current sense
        - 0x0003: Heat sink temperature
        - 0x0002: Left Channel voltage sense (Vinstr=6*Vin)
        - 0x0001: Left Channel trip level sense
        - 0x0000: Left Channel current sense
    - 0xb000 ADC Read (mask for values: 0x0fff)
- Interrupt Register
  - Notes: The DPA is capable of generating an interrupt to the
    SideKick processor ONLY on overcurrent conditions. If a
    channel has an overcurrent condition (is drawing more
    current than allowed by the SetCurrentLimit cmd), the
    electronic circuit breaker will trip. An interrupt will
    be asserted iff the electronic circuit breaker for that
    channel has tripped (and has not been reset), AND the
    interrupt enable for that channel is set, AND the global
    interrupt enable has been set. The assertion of an
    interrupt will cause three things to be true: the
    interrupt flag for that channel will be set, the global
    interrupt flag will be set, and the DPA board will
    assert an interrupt to the SideKick processor. The
    overcurrent condition itself is asserted by the digital
    circuit breaker, and that breaker is cleared by turning
    it off. (Reading the interrupt register has no effect
    on the interrupt flags.)
  - Commands
    - 0x5000 Write
    - 0xb000 Read
  - Register bits (mask 0x003f, RO=WriteOnly, RW=Read/Write)
    - 0x0020: (RW)Left Channel overcurrent intr en
    - 0x0010: (RW)Right Channel overcurrent intr en
    - 0x0008: (RW)Global intr en
    - 0x0004: (RO)Left Channel overcurrent intr flag
    - 0x0002: (RO)Right Channel overcurrent intr flag
    - 0x0001: (RO)Global intr flag
- DPA Channel-level commands (cmnd mask 0xf000, values:)
  - Relays
    - Commands
      - 0x2000: Write left channel register
      - 0x3000: Write right channel register
      - 0xa000: Read left channel register
      - 0xb000: Read right channel register
    - Register bits
      - 0x0004: Connect 485 Terminators

```

sidekick\DpaBoard.java

```

- 0x0002: Isolate communications ground
- 0x0001: Isolate instrument power
- Dpot (Digital Circuit Breaker Current Limit)
  - Commands
    - 0x0000: Write left channel
    - 0x1000: Write right channel
  - Register bits and fields
    - 0x0100: Save position
    - 0x0080: Up (move potentiometer UP)
    - 0x007f: mask for counter value (0 through 127)
- Channel Control
  - Commands
    - 0x6000: Write left channel
    - 0x7000: Write right channel
    - 0xd000: Read left channel
    - 0xe000: Read right channel
  - Register bits
    - 0x0020: Serial tx power on
    - 0x0010: Comm fast (0 means Slew Rate Controlled)
    - 0x0008: Comm Mode RS-485 (0 means RS-232)
    - 0x0004: Comm Half Duplex (0 means full duplex)
    - 0x0002: Comm Power On
    - 0x0001: Instrument Power On (this is the Digital Circuit Breaker)

```

```

*/

// Implementation notes: when a board is created (ctor) it should
// instantiate all the left and right components, because it should
// initialize them. From there on out, when a user of a channel
// manipulates any of its components, including components that are
// specialized to the left or right channel, they will do the right
// thing, since (a) they have access to the DPA board resources (shared
// between the channels) and (b) have their own control when that's
// appropriate. Since the operation will be specified in the (abstract)
// superclass while the implementation lives in the subclass, the
// user can safely call superclass operations and they will either
// do the right thing because they are identical for both channels, or
// they will do the right thing because the implementation that gets
// invoked has been correctly specialized to the correct channel.
// Probably the only thing that really needs to handle the left and
// right channels as such is the board itself, and it can do so.
// E.g., if it needs to initialize the current limit by setting both
// channels to zero, it can do { leftChannel.adc.setCurrentLimit(0);
// rightChannel.adc.setCurrentLimit(0); }.
//

```

```

public class DpaBoard {

    private DpaChannel dpaLeftChannel; // auto-initialized to null
    private boolean dpaAvailChecked;    // have we looked for a DPA?
    private boolean dpaHardwareIsPresent; // is it there?

    private DpaChannel dpaRightChannel; // auto-initialized to null
    private int[] adcReg = new int[1];
    private int[] leftRelayReg = new int[1];
    private int[] rightRelayReg = new int[1];
    private int[] leftChannelCtrlReg = new int[1];
    private int[] rightChannelCtrlReg = new int[1];
    private int[] interruptReg = new int[1];

```

```

{
    adcReg[0] = 0;
    leftRelayReg[0] = 0;
    rightRelayReg[0] = 0;
    leftChannelCtrlReg[0] = 0;
    rightChannelCtrlReg[0] = 0;
    interruptReg[0] = 0;
}

/*
public void printShadowRegisters() {
    System.out.println("adcReg[0] = " + adcReg[0]
        + "; leftRelayReg[0] = " + leftRelayReg[0]
        + "; rightRelayReg[0] = " + rightRelayReg[0]);
}
*/

abstract public class HdwrReg {
    protected int[] regVal;
    protected boolean changed = true; // power-up value uncertain

    HdwrReg() {
        System.out.println("HdwrReg(void) ctor invoked");
    }
    HdwrReg(int[] regVal) {
        System.out.println("HdwrReg(int[]) ctor invoked");
        this.regVal = regVal;
    }
    protected void set(int mask, int val) {
        if ((regVal[0] & mask) == val)
            return;
        regVal[0] &= ~mask;
        regVal[0] |= val;
        changed = true;
    }
    protected void set(int bitmask) {
        set(bitmask, bitmask);
    }
    protected void clear(int mask) {
        set(mask, 0);
    }
};

abstract private class Adc extends HdwrReg {

    public final static int Wr = 0x4000;
    public final static int Rd = 0xb000;

    private final static int PowerMask = 0x00c0;
    private final static int PowerOff = 0x0000;
    private final static int PowerStby = 0x0040;
    private final static int PowerOnIntClk = 0x0080;
    //private final static int PowerOnExtClk = 0x00c0;

    private final static int AcqExt = 0x0020;
    private final static int SglDrive = 0x0010;
    private final static int Unipolar = 0x0008;
    private final static int BatterySupplyVoltage = 0x0007;
    protected final static int RightVoltage = 0x0006;
    protected final static int RightTripLevel = 0x0005;
    protected final static int RightCurrent = 0x0004;
    private final static int HeatSinkTemp = 0x0003;

```



```

protected final static int LeftVoltage      = 0x0002;
protected final static int LeftTripLevel    = 0x0001;
protected final static int LeftCurrent      = 0x0000;

private Adc(int[] regVal) {
    super(regVal);
    System.out.println("Adc ctor invoked");
    // These values will ALWAYS be this way.
    setSglDrive();
    setUnipolar();
}

public void setPowerOff() {
    set(PowerMask, PowerOff);
}

public void setPowerStby() {
    set(PowerMask, PowerStby);
}

public void setPowerOnIntClk() {
    set(PowerMask, PowerOnIntClk);
}

// PowerOnExternalClock will NEVER be used. Here only for reference.
//public void setPowerOnExtClk() {
//    set(PowerMask, PowerOnExtClk);
//}

public void setAcqExternal() {
    set (AcqExt, AcqExt);
}

public void setAcqInternal() {
    set (AcqExt, 0);
}

public void setSglDrive() {
    set (SglDrive, SglDrive);
}

public void setDifferentialDrive() {
    set (SglDrive, 0);
}

public void setUnipolar() {
    set (Unipolar, Unipolar);
}

public void setBipolar() {
    set (Unipolar, 0);
}

// To take a reading, put the ADC into the on/ext state.
// Wait 100msec.
// Put it into the stby state.
// Wait for the board to become nonbusy.
// Read the value.
void initialize() {
    setAcqInternal();
    setPowerStby();
    writeReadSpi(Wr | regVal[0] | 0); // channel is don't-care here
    stallOnBusyBit();
}

protected int getReading(int adcChannel) {
    setAcqExternal();
    setPowerOnIntClk();
    writeReadSpi(Wr | regVal[0] | adcChannel);
    try {
        Thread.sleep(100);

```

```

    } catch (java.lang.InterruptedExceptio e) {
        System.out.println("ADC sleep got interrupted: " + e);
    }
    stallOnBusyBit();

    setAcqInternal();
    setPowerStby();
    writeReadSpi(Wr | regVal[0] | adcChannel);
    stallOnBusyBit();

    int retval = writeReadSpi(Rd); // other 12 bits are don't-cares
    stallOnBusyBit(); // read cannot possibly make it busy, though
    return retval;
}

public int getBatterySupplyVoltage() {
    return getReading(BatterySupplyVoltage);
}

public int getHeatSinkTemp() {
    return getReading(HeatSinkTemp);
}

abstract public int getVoltage();
abstract public int getTripLevel();
abstract public int getCurrent();
};

private class AdcLeft extends Adc {
    private AdcLeft(int[] regVal) {
        super(regVal);
        System.out.println("AdcLeft ctor invoked");
    }
    public int getVoltage() {
        return getReading(LeftVoltage);
    }
    public int getTripLevel() {
        return getReading(LeftTripLevel);
    }
    public int getCurrent() {
        return getReading(LeftCurrent);
    }
}

};

private class AdcRight extends Adc {
    private AdcRight(int[] regVal) {
        super(regVal);
        System.out.println("AdcRight ctor invoked");
    }
    public int getVoltage() {
        return getReading(RightVoltage);
    }
    public int getTripLevel() {
        return getReading(RightTripLevel);
    }
    public int getCurrent() {
        return getReading(RightCurrent);
    }
}

};

Adc adc; // Adc is abstract, a real DpaBoard must be
// created with a AdcLeft or an AdcRight, but
// the user does not need to know which

// To read the intr register, first do a read (otherwise you may get

```

```
// a stale value
//

abstract public class RelayReg extends HdwrReg {
    private final static int Connect485Terminators = 0x0004;
    private final static int IsolateCommunicationsGround = 0x0002;
    private final static int IsolateInstrumentPower = 0x0001;
    RelayReg(int[] regVal) {
        super(regVal);
        System.out.println("RelayReg ctor invoked");
    }
    public void connect485Terminators() {
        set(Connect485Terminators);
    }
    public void disconnect485Terminators() {
        clear(Connect485Terminators);
    }
    public void isolateCommunicationsGround() {
        set(IsolateCommunicationsGround);
    }
    public void connectCommunicationsGround() {
        clear(IsolateCommunicationsGround);
    }
    public void isolateInstrumentPower() {
        set(IsolateInstrumentPower);
    }
    public void connectInstrumentPower() {
        clear(IsolateInstrumentPower);
    }
    abstract public void write();
};

private class LeftRelayReg extends RelayReg {
    LeftRelayReg(int[] regVal) {
        super(regVal);
        System.out.println("LeftRelayReg ctor invoked");
    }
    private final static int Wr = 0x2000;
    public void write() {
        writeReadSpi(Wr | regVal[0]);
        stallOnBusyBit();
    }
};

private class RightRelayReg extends RelayReg {
    RightRelayReg(int[] regVal) {
        super(regVal);
        System.out.println("RightRelayReg ctor invoked");
    }
    private final static int Wr = 0x3000;
    public void write() {
        writeReadSpi(Wr | regVal[0]);
        stallOnBusyBit();
    }
};

abstract public class ChannelCtrlReg extends HdwrReg {
    private final static int TxHiPowerOn = (1<<5);
    private final static int SlewRateNotLimited = (1<<4);
    private final static int CommMode485 = (1<<3);
    private final static int CommHalfDuplex = (1<<2);
    private final static int CommPowerOn = (1<<1);
    private final static int InstrumentPowerOn = (1<<0);

    ChannelCtrlReg(int[] regVal) {
        super(regVal);
    }
    public void setTxHiPower() {
        set(TxHiPowerOn);
    }
    public void setTxLoPower() {
        clear(TxHiPowerOn);
    }
    public void setSlewRateNotLimited() {
        set(SlewRateNotLimited);
    }
    public void setSlewRateLimited() {
        clear(SlewRateNotLimited);
    }
    public void setCommModeRs485() {
        set(CommMode485);
    }
    public void setCommModeRs232() {
        clear(CommMode485);
    }
    public void setCommHalfDuplex() {
        set(CommHalfDuplex);
    }
    public void setCommFullDuplex() {
        clear(CommHalfDuplex);
    }
    public void setCommPowerOn() {
        set(CommPowerOn);
    }
    public void setCommPowerOff() {
        clear(CommPowerOn);
    }
    public void setInstrumentPowerOn() {
        set(InstrumentPowerOn);
    }
    public void setInstrumentPowerOff() {
        clear(InstrumentPowerOn);
    }
    abstract public void write();
};

private class LeftChannelCtrlReg extends ChannelCtrlReg {
    private LeftChannelCtrlReg(int[] regVal) {
        super(regVal);
        System.out.println("LeftChannelCtrlReg ctor invoked");
    }
    private final static int Wr = 0x6000;
    public void write() {
        System.out.println("doing a LeftChannelCtrlReg write()");
        writeReadSpi(Wr | regVal[0]);
        stallOnBusyBit();
    }
};

public class RightChannelCtrlReg extends ChannelCtrlReg {
    private RightChannelCtrlReg(int[] regVal) {
        super(regVal);
        System.out.println("RightChannelCtrlReg ctor invoked");
    }
    private final static int Wr = 0x7000;
```

```

        public void write() {
            System.out.println("doing a RightChannelCtrlReg write()");
            writeReadSpi(Wr | regVal[0]);
            stallOnBusyBit();
        }
    };

    abstract private class InterruptReg extends HdwrReg {
        public static final int Wr = 0x5000;
        public static final int EnableMask          = 0x0038;
        public static final int LeftOvercurrentEnable = 0x0020;
        public static final int RightOvercurrentEnable = 0x0010;
        public static final int GlobalOvercurrentEnable = 0x0008;
        public static final int FlagMask              = 0x0007;
        public static final int LeftOvercurrentIntrFlag = 0x0004;
        public static final int RightOvercurrentIntrFlag = 0x0002;
        public static final int GlobalOvercurrentIntrFlag = 0x0001;

        public InterruptReg(int[] regVal) {
            super(regVal);
        }

        public void setGlobalOvercurrentEn() {
            set(GlobalOvercurrentEnable);
        }
        public void clearGlobalOvercurrentEn() {
            clear(GlobalOvercurrentEnable);
        }
        public boolean isGlobalOvercurrentIntrFlagSet() {
            return ((regVal[0] & GlobalOvercurrentIntrFlag) != 0);
        }
        abstract public void setOvercurrentIntrEn();
        abstract public void clearOvercurrentIntrEn();
        abstract public boolean isOvercurrentFlagSet();
        public void write() {
            int retval = writeReadSpi(Wr | regVal[0]);
            regVal[0] = ((regVal[0] & EnableMask) | (retval & FlagMask));
            stallOnBusyBit();
        }
        //abstract public void read(); // actually, we can implement this here
    };

    private class LeftInterruptReg extends InterruptReg {
        private LeftInterruptReg(int[] regVal) {
            super(regVal);
        }
        public void setOvercurrentIntrEn() {
            set(LeftOvercurrentEnable);
        }
        public void clearOvercurrentIntrEn() {
            clear(LeftOvercurrentEnable);
        }
        public boolean isOvercurrentFlagSet() {
            return ((regVal[0] & LeftOvercurrentIntrFlag) != 0);
        }
    };

    private class RightInterruptReg extends InterruptReg {
        private RightInterruptReg(int[] regVal) {
            super(regVal);
        }
        public void setOvercurrentIntrEn() {
            set(RightOvercurrentEnable);
        }
    };

```

```

        public void clearOvercurrentIntrEn() {
            clear(RightOvercurrentEnable);
        }
        public boolean isOvercurrentFlagSet() {
            return ((regVal[0] & RightOvercurrentIntrFlag) != 0);
        }
    };

    protected boolean doDebug = false;

    protected int spiSlaveSelectValue;
    protected SpiMaster spi = SpiMaster.getInstance();
    protected boolean boardInitialized = false;

    public DpaBoard (int spiSlaveSelectValue) {
        System.out.println("DpaBoard ctor invoked");
        this.spiSlaveSelectValue = spiSlaveSelectValue;
    }

    // NOTE: The create-if-null strategy will NOT work in the presence
    // of contention by multiple threads
    //
    public DpaChannel getLeftChannel() {
        if (dpaLeftChannel == null)
            dpaLeftChannel = this.new DpaLeftChannel();

        return dpaLeftChannel;
    }

    public DpaChannel getRightChannel() {
        if (dpaRightChannel == null)
            dpaRightChannel = this.new DpaRightChannel();

        return dpaRightChannel;
    }

    protected int writeReadSpi(int dataOut) {
        int dataIn;
        synchronized (spi) {
            spi.setSlaveSelect(spiSlaveSelectValue);
            dataIn = spi.writeReadData(0xff & (dataOut >> 8));
            dataIn <<= 8;
            dataIn |= (0xff & spi.writeReadData(0xff & dataOut));
            spi.setSlaveSelect(spi.SPI_SLAVE_SELECT_NONE);
        }
        /**/
        System.out.println("    writeReadSpi(0x"
            + Integer.toHexString(dataOut)
            + ") returns 0x"
            + Integer.toHexString(dataIn));
        /**/

        // if (dataIn == 0xffff) or
        // (dataIn & 0x7fff) == 0x7fff) throw exception
        return dataIn;
    }

    public synchronized void stallOnBusyBit() {
        /**/
        int regVal;
        if (dpaHardwareIsPresent) {
            do {

```

sidekick\DpaBoard.java

```

        regVal = writeReadSpi(0xf << 12);
    } while ((regVal & 0x7fff) == 0x7fff);
}
/**
/**
System.out.println("Spi bus cycle complete");
/**
}

/*
 * The DpaChannel contains everything associated with the
 * channel, and also with its DPA board.
 */
public abstract class DpaChannel implements IoResource {

    protected boolean channelInitialized = false;
    protected boolean currentCtrlInitialized = false;
    protected Instrument instrument;
    protected javax.comm.SerialPort instrumentPort;
    protected int presentPosition;
    protected static final int MOVE_UP = 1;
    protected static final int MOVE_DOWN = 0;
    protected RelayReg relayReg;
    protected ChannelCtrlReg channelCtrlReg;
    protected InterruptReg interruptCtrlReg;

    public DpaChannel() {
        System.out.println("DpaChannel ctor invoked");
    }

    void initializeChannel() {
        if (! channelInitialized) {
            initializeBoard(); // no-op if board already initialized

            //      1) Initialize the ADC. Done in initializeBoard();

            //      2) Initialize the relays
            relayReg.disconnect485Terminators();
            relayReg.isolateCommunicationsGround();
            relayReg.isolateInstrumentPower();
            relayReg.write();

            //      3) Set Current Limit -- this is done by the
            //          SideArm (or some other configurator) later.

            setCurrentLimit(1400); // sets trip value for
                                   // digital circuit breaker
                                   // 1400 is 11 steps (rounded down)

            //      4) Configure the comm channel -- this is also done
            //          by the configurator.
            relayReg.connectInstrumentPower();
            relayReg.write();
            System.out.println("initializing channelCtrlReg");
            channelCtrlReg.setTxHiPower();
            channelCtrlReg.setSlewRateNotLimited();
            channelCtrlReg.setCommModeRs232();
            channelCtrlReg.setCommFullDuplex();
            channelCtrlReg.setCommPowerOn();
            channelCtrlReg.setInstrumentPowerOn();
            channelCtrlReg.write();

```

```

        // should remember global setting and other channel's setting
        interruptCtrlReg.setOvercurrentIntrEn();
        interruptCtrlReg.write();

        channelInitialized = true;
    }
}

protected void checkOnceForDpaHardware() {
    if (! dpaAvailChecked) {
        // this is just like stallOnBusyBit()
        int spiVal;
        spiVal = writeReadSpi(0xf << 12);
        dpaAvailChecked = true;

        // 0x7fff is the busy indication. If we get that on
        // the first read, we know that the real hardware,
        // which would definitely not be busy, is not there
        // and the circuitry that is there will assert 'busy'
        // forever. Note that sometimes no hardware will
        // return 0xffff, hence the mask.

        dpaHardwareIsPresent = ((spiVal & 0x7fff) != 0x7fff);
        System.out.print("checkOnceForDpaHardware() called...");
        if (dpaHardwareIsPresent) {
            System.out.println("DPA is available");
        } else {
            System.out.println("found no DPA hardware.");
        }
    }
}

protected void initializeBoard() {
    System.out.println("first line of initializeBoard()");
    if (! boardInitialized) {
        System.out.println("Doing DpaChannel.initializeBoard()");
        spi.setClkDivider(spi.SPI_CLOCK_DIVIDER_64);

        spi.setSlaveSelect(spi.SPI_SLAVE_SELECT_NONE);
        spi.writeReadData(0); // Do a write with no chip select.
                               // This forces the DPA into a known
                               // state (just in case it had seen
                               // spurious spi clock edges with a
                               // spurious CS).

        // Check for DPA hardware before any normal SPI cycles.
        //
        checkOnceForDpaHardware();

        adc.initialize();
        interruptCtrlReg.setGlobalOvercurrentEn();
        interruptCtrlReg.write();
        boardInitialized = true;
    }
}

public void setSerialPort(SerialPort aPort) {
    // if aPort is null or instrumentPort isn't, throw exception...
    this.instrumentPort = aPort;
}

```

```

public Object getResourceType(IoResourceType aType) {
    if (aType == IoResourceType.DPA_BOARD)
        return this;
    else if (aType == IoResourceType.INSTRUMENT)
        return instrument;
    else
        return null;
}

private void initializeCurrentCtrl() {
    System.out.println("in InitializeCurrentCtrl()");

    movePot(MOVE_DOWN, 99);
    stallOnBusyBit();
    presentPosition = 0;
    currentCtrlInitialized = true;
}

public void setCurrentLimit(int milliamps) {
    System.out.println("In setCurrentLimit(int milliamps)");
    if (!currentCtrlInitialized) {
        System.out.println("    about to initializeCurrentCtrl()...");
        initializeCurrentCtrl();
        System.out.println("    initializeCurrentCtrl() done.");
    }

    // if milliamps < 0 or > ?? throw exception

    // For safety, we always round DOWN (i.e., std integer division)
    // The device can be set in increments of 120 milliamps,
    // from 0*120 to 99*120 = 11880 (i.e., from 0 to 11.88 Amps)
    int newPosition = milliamps / 120;
    int diff = newPosition - presentPosition;
    if (diff > 0)
        movePot(MOVE_UP, diff);
    else if (diff < 0)
        movePot(MOVE_DOWN, -diff);

    stallOnBusyBit();

    this.presentPosition = newPosition;
    /*END*/
}

int getCurrentLimitCtrlValue() {
    return presentPosition * 120;
}

protected abstract void movePot(int direction, int nStepsUp);

public void setInstrument( Instrument anInstrument ) {
    instrument = anInstrument;
}
};

public class DpaLeftChannel extends DpaChannel {
    DpaLeftChannel() {
        System.out.println("DpaLeftChannel ctor invoked");
        adc = new AdcLeft(adcReg); // both channels share the adc reg
        relayReg = new LeftRelayReg(leftRelayReg); // but not relay reg
        channelCtrlReg = new LeftChannelCtrlReg(leftChannelCtrlReg);
        interruptCtrlReg = new LeftInterruptReg(interruptReg);
    }
}

```

```

protected void movePot(int direction, int nStepsUp ) {
    System.out.println("DpaLeftChannel movePot(dir="
        + direction
        + "; steps=" + nStepsUp );

    int regValue =
        (0 << 12)           // CMND = 0 is left channel Dpot cmdnd
        | (direction << 8)  // up is 1 in bit 8, down is 0
        | (0 << 7)         // do not save position to flash
        | (nStepsUp & 0x7f); // the count is in the low order 7 bits

    writeReadSpi(regValue);
}

};

public class DpaRightChannel extends DpaChannel {
    DpaRightChannel() {
        System.out.println("DpaRightChannel ctor invoked");
        adc = new AdcRight(adcReg); // both channels share the adc reg
        relayReg = new RightRelayReg(rightRelayReg); // but not relay reg
        channelCtrlReg = new RightChannelCtrlReg(rightChannelCtrlReg);
        interruptCtrlReg = new RightInterruptReg(interruptReg);
    }

    protected void movePot(int direction, int nStepsUp ) {
        int regValue =
            (1 << 12)           // CMND = 1 is right channel Dpot cmdnd
            | (direction << 8)  // up is 1 in bit 8, down is 0
            | (0 << 7)         // do not save position to flash
            | (nStepsUp & 0x7f); // the count is in the low order 7 bits

        writeReadSpi(regValue);
    }
}
};
}

```

```
package nmc.sidekick;

public class DumbSleepTest {

    /**
     * Starts the application.
     * @param args an array of command-line arguments
     */
    public static void main(java.lang.String[] args) {
        int i = 0;
        while (true) {
            try {
                Thread.sleep(1000);
                i++;
                if ( (i % 10) == 0 ) {
                    System.out.println("i=" + i);
                }
            } catch (InterruptedException e) {
                System.out.println("InterruptedException: " + e);
            }
        }
    }
}
```

06/13/02
20:06:58

sidekick\Instrument.java

1

```
package nmc.sidekick;
import nmc.common.TimestampedData;

import javax.comm.SerialPort;

public interface Instrument {
    public void setSerialPort(SerialPort aPort) throws java.io.IOException;
    public TimestampedData poll() throws Exception;
    public TimestampedData configure(byte[] cfgData) throws Exception;
}
```

06/13/02
20:06:58

sidekick\IoResource.java

1

```
package nmc.sidekick;  
  
// this is just a test edit  
  
public interface IoResource {  
    public Object getResourceType(IoResourceType aType);  
}
```


06/13/02
20:06:58

sidekick\IoResourceType.java

1

```
package nmc.sidekick;

public class IoResourceType {
    private String name;
    private IoResourceType(String aString) {
        name = aString;
    }
    static final IoResourceType DPA_BOARD = new IoResourceType("DPA_CHANNEL");
    static final IoResourceType INSTRUMENT = new IoResourceType("INSTRUMENT");
}
```

```
package nmc.sidekick;
import javax.comm.SerialPort;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.lang.Exception;
import nmc.common.TimestampedData;

public class MicroCat extends StdInstrumentDriver {

    MicroCat ( ) {

        _name      = "TheMicroCatDriverRev2.1";

        _enter      = "\r"      .getBytes();
        _prompt     = "\r\nS>"  .getBytes();
        _sampleRequest = "TS\r"  .getBytes();
        _sampleRequestEcho = "TS\r\n" .getBytes();
        _cfgEol      = "\r\n"    .getBytes();

        _timeout     = 40000;
        _maxTries     = 3;
        _maxAttnBytes = 250;
        _maxSampleBytes = 1000;
        _initMaxTries = 3;
        _maxSkipBytes  = 200;
        _cfgBufSize    = 200;

    }
}
```

```
package nmc.sidekick;

import java.io.IOException;
import com.ajile.util.PropUtils;
import com.ajile.jem.rawJEM;

/**
 * Example program which demonstrates how to use the RAMTest instruction to test a
 * block of read/write RAM memory. Copied from Ajile distribution, with a few
 * minor changes.
 */
public class Ramtest {

    private static int RAMStart;
    private static int RAMSize;

    public static void main(String[] args) {

        Ramtest app = new Ramtest();
    }

    // RAM starts at 0x10, default is 1MB, but allow for non-default
    // RAM starts and sizes if the properties are set
    public Ramtest() {

        int RAMSize = 0;
        int RAMStart = 0x0;
        int TEST_PATTERN_1 = 0x55555555;
        int TEST_PATTERN_2 = 0xAAAAAAAA;

        System.out.println("Starting test of SRAM^_^_^_^_^_^_^");

        boolean loopRAMTest = System.getProperty("repeat.test") != null;

        RAMStart = PropUtils.getIntProperty("ram.start", 0x10);

        System.out.println("RAMStart= 0x" + Integer.toHexString(RAMStart));

        RAMSize = PropUtils.getIntProperty("ram.size", 0x120000);

        System.out.println("RAMSize= 0x" + Integer.toHexString(RAMSize));

        int RAMTestStart;

        // Don't start the RAM test at location 0x0
        if (RAMStart == 0x0) {
            RAMTestStart = 0x10;
        } else {
            RAMTestStart = RAMStart;
        }

        int numBytes = (RAMSize - RAMTestStart);

        do {

            System.out.println("-----");
            System.out.println("RAM test starting at address 0x" + RAMTestStart);

            // Run a non-destructive pattern test pattern1 and pattern2 on physical RA
            M memory
            int failword = rawJEM.ramTest(RAMTestStart, numBytes/4, TEST_PATTERN_1, TE
            ST_PATTERN_2);

            System.out.println(numBytes + " bytes written");

            if (failword != 0)
                System.out.println("Failed at word 0x" + Integer.toHexString(failword)
            );
            else
                System.out.println("RAM test passed");

            System.out.println("-----");

        } while (loopRAMTest);
    } // Ramtest
}
```

sidekick\Rtc.java

```

package nmc.sidekick;

import com.ajile.jem.rawJEM;
import com.ajile.drivers.spi.*;
import java.util.Calendar;
import com.ajile.drivers.irq.InterruptController;
import com.ajile.drivers.irq.InterruptEventListener;

class RtcRunner extends Thread {
    public RtcRunner() {
        rawJEM.setCeiling(this, 31);
    }
    public void run() {
        System.out.println("This is the standalone Rtc test");
        Calendar rn = Calendar.getInstance();
        System.out.println("Sunday=" + Calendar.SUNDAY);
        System.out.println("Monday=" + Calendar.MONDAY);
        System.out.println("Tuesday=" + Calendar.TUESDAY);
        System.out.println("Wednesday=" + Calendar.WEDNESDAY);
        System.out.println("Thursday=" + Calendar.THURSDAY);
        System.out.println("Friday=" + Calendar.FRIDAY);
        System.out.println("Saturday=" + Calendar.SATURDAY);

        Rtc rtc = new Rtc();
        rtc.initSpiForRtcTest();
        rtc.initRtcCsN();

        rtc.runTest();
    }
}

public class Rtc implements InterruptEventListener {

    private SpiMaster spi = SpiMaster.getInstance();
    private int erruptsProcessed = 0;
    private int WErruptsProcessed = 0;
    private int DErruptsProcessed = 0;

    /**
     * Starts the application.
     * @param args an array of command-line arguments
     */
    public static void main(java.lang.String[] args) {
        RtcRunner r = new RtcRunner();
        r.start();
    }

    public void runTest() {
        System.out.println("reading control registers");

        int c1 = get(CTRL_1_REG);
        c1 &= 0xc0;
        set(CTRL_1_REG, c1);
        int c2 = get(CTRL_2_REG);
        System.out.println("ctrl1=0x" + Integer.toHexString(c1)
            + "; ctrl2=0x" + Integer.toHexString(c2)
            + ";");

        c2 &= ~3;
        set(CTRL_2_REG, c2);

        c1 |= 0x20;

        set(CTRL_1_REG, c1);
        c1 = get(CTRL_1_REG);
        System.out.println("ctrl1=0x" + Integer.toHexString(c1)
            + "; ctrl2=0x" + Integer.toHexString(c2)
            + ";");

        RtcTime torig = getTime();
        System.out.println("Before setting, the time (by the rtc) is: "
            + torig);
        setTime(2002, Calendar.MAY, 23, 10, 07, 50);
        int r = 0;
        installInterruptHandler();
        enableInterrupts();
        setAlarmD(18, 58);
        while (true) {

            RtcTime t = getTime();
            System.out.println("The current time (by the rtc) is: "
                + t);

            System.out.println("interrupts processed = " + erruptsProcessed);
            if (erruptsProcessed != r) {
                System.out.println("got a new interrupt");
                r = erruptsProcessed;
                c2 = get(CTRL_2_REG);
                if ((c2 & 3) != 0) {
                    set(CTRL_2_REG, (c2 & ~3));
                }
                System.out.println("ctrl reg 2 = 0x"
                    + Integer.toHexString(c2));
                if (++t.minute >= 60) {
                    t.minute = 0;
                    if (++t.hour == 24) {
                        t.hour = 0;
                    }
                }

                setAlarmD(t.hour, t.minute);
                reenableInterrupts();
            }
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                System.out.println("got InterruptedException: " + e);
            }
        }
    }

    public void installInterruptHandler() {
        InterruptController
            addInterruptListener(this,
                InterruptController.GPIOA_BIT5_INTERRUPT);
    }

    public void enableInterrupts() {
        InterruptController
            enableInterrupt(InterruptController.GPIOA_BIT5_INTERRUPT);
    }

    public void reenableInterrupts() {
        bp3(0x21);
    }
}

```

```

rawJEM.set(AjileAddr.GPIOA_CLEAR_INT, (1<<5)); // GPIOA5
rawJEM.setField(1, 0x0900201c, 5, 1);
bp3(0x31);
}

public void interruptEvent() {
    // this is called during the 'rupt, from the interrupt controller
    bp2(0x11);
    erruptsProcessed++;
    rawJEM.setField(0, 0x0900201c, 5, 1);
}

public int bp2(int i) {
    return i++;
}

public int bp3(int i) {
    return i++;
}

int toBcd(int b) {
    int b0 = b % 10;
    int b1 = b / 10;
    return (b1<<4) | b0;
}

int fromBcd(int bcd) {
    return (bcd & 0xf + (bcd>>4)*10);
}

public void setAlarmD(int hour, int min) {
    int c1 = get(CTRL_1_REG);
    c1 &= ~DALE;
    set(CTRL_1_REG, c1);

    set(ALARM_D_HOUR_REG, toBcd(hour));
    set(ALARM_D_MIN_REG, toBcd(min));

    int c2 = get(CTRL_2_REG);
    c2 &= ~DAFG;
    set(CTRL_2_REG, c2);
    c1 |= DALE;
    set(CTRL_1_REG, c1);
}

class RtcTime {
    int year;
    int month;
    int day;
    int hour;
    int minute;
    int second;

    public String toString() {
        return "RctTime: "
            + year
            + "; "
            + month
            + "; "
            + day
            + "; "
            + hour
            + "; "
            + minute

```

```

            + "; "
            + second
            + ";";
        }
    }

    public RtcTime getTime() {
        RtcTime t = new RtcTime();
        int mccb = get(MCCB_REG);
        int y = get(YEAR_CTR_REG);
        t.year = (y & 0xf) + (y >> 4)*10;
        if ((mccb & 0x80) == 0)
            t.year += 1900;
        else
            t.year += 2000;

        System.out.println("in getTime, mccb = 0x" + Integer.toHexString(mccb));
        t.month = (0xf & mccb) + ((0x70 & mccb) >> 4)*10;

        int d = get(DAYOFMONTH_CTR_REG);
        t.day = (0xf & d) + (d >> 4)*10;

        //          t.hour = get(HOUR_CTR_REG);
        //          if ((t.hour & 0x20) != 0) {
        //              t.hour &= ~0x20;
        //          } else if (t.hour == 12) {
        //              t.hour = 0;
        //          }

        int h = get(HOUR_CTR_REG);
        t.hour = (h & 0xf) + (h>>4)*10;

        int min = get(MINUTE_CTR_REG);
        t.minute = (0xf & min) + (min>>4)*10;
        int sec = get(SECOND_CTR_REG);
        t.second = (0xf & sec) + (sec>>4)*10;
        return t;
    }

    public void setTime(RtcTime t) {
        setTime(t.year, t.month, t.day, t.hour, t.minute, t.second);
    }

    public void setTime(int year, int month, int day, int hour, int minute, int second
) {
        int centuryBit;
        int y, yy0, yy1;
        if (year >= 2000) {
            centuryBit = 0x80;
            y = year - 2000;
            yy0 = y % 10;
            yy1 = y / 10;
            // if (yy1 > 9)
            //     throw new IllegalArgumentException("cannot support years beyond 209
9");
        } else {
            centuryBit = 0;

```

sidekick\Rtc.java

```

// if (year < 1900)
//     throw new IllegalArgumentException("cannot support years before 190
0");
    y = year - 1900;
    yy0 = y % 10;
    yy1 = y / 10;
}
set(YEAR_CTR_REG, (yy1<<4 | yy0));
int mm0, mm1;
month += 1; // In java JANUARY==0, in the rtc, january == 1
if (month > 9) {
    mm1 = 1;
    mm0 = month - 10;
} else {
    mm1 = 0;
    mm0 = month;
}
set(MCCB_REG, centuryBit | (mm1 << 4) | mm0);
int dd0 = day % 10;
int dd1 = day / 10;
set(DAYOFMONTH_CTR_REG, dd1<<4 | dd0);
//
//     switch (hour) {
//     case 0:
//         hour = 12;
//         pm = 0;
//         break;
//     case 1:
//     case 2:
//     case 3:
//     case 4:
//     case 5:
//     case 6:
//     case 7:
//     case 8:
//     case 9:
//     case 10:
//     case 11:
//         pm = 0;
//         break;
//     case 12:
//         pm = 0x20;
//         break;
//     case 13:
//     case 14:
//     case 15:
//     case 16:
//     case 17:
//     case 18:
//     case 19:
//     case 20:
//     case 21:
//     case 22:
//     case 23:
//         pm = 0x20;
//         hour -= 12;
//         break;
//     default:
//         throw new IllegalArgumentException("hour ("
//             + hour
//             + ") is negative or too high");
//         // break; // compiler complains that break is unreachable code

```

```

//     }
//     int hh0, hh1;
//     hh0 = hour % 10;
//     hh1 = (hour > 9) ? 1 : 0;
//     set(HOUR_CTR_REG, pm | (hh1<<4) | hh0);

    int hh0, hh1;
    hh0 = hour % 10;
    hh1 = hour / 10;
    set(HOUR_CTR_REG, (hh1<<4) | hh0);
    int min0, min1;
    min0 = minute % 10;
    min1 = minute / 10;
    set(MINUTE_CTR_REG, (min1<<4) | min0);
    int ss0, ss1;
    ss0 = second % 10;
    ss1 = second / 10;
    set(SECOND_CTR_REG, (ss1<<4) | ss0);
}

// duplicates other code in full sidekick
public void initSpiForRtcTest() {
    spi.setClkDivider(spi.SPI_CLOCK_DIVIDER_64);
    spi.setSlaveSelect(spi.SPI_SLAVE_SELECT_NONE);
}

// The following is NOT THREAD-SAFE
// If anything else is doing a read-modify-write of regE at the same time,
// Heisenbugs are likely to appear
//
public void initRtcCsN() {
    // set the value of AdcCsN to 1 (chip select not asserted)
    bp();
    rawJEM.setField(0, AjileAddr.GPIOC_OR, 7, 1);

    bp();

    rawJEM.setField(1, AjileAddr.GPIOC_OER, 7, 1);

    bp();
}

private void assertCs() {
    rawJEM.setField(1, AjileAddr.GPIOC_OR, 7, 1);
}

public void negateCs() {
    rawJEM.setField(0, AjileAddr.GPIOC_OR, 7, 1);
}

public void bp() {
    System.out.println("set breakpoint here");
}

static final int SECOND_CTR_REG      = 0x00;
static final int MINUTE_CTR_REG      = 0x10;
static final int HOUR_CTR_REG        = 0x20;
static final int DAYOFWEEK_CTR_REG   = 0x30;
static final int DAYOFMONTH_CTR_REG  = 0x40;

```

```
static final int MCCB_REG          = 0x50;
static final int YEAR_CTR_REG      = 0x60;
static final int OSC_ADJ_REG       = 0x70;
static final int ALARM_W_MIN_REG   = 0x80;
static final int ALARM_W_HOUR_REG  = 0x90;
static final int ALARM_W_DAYOFWEEK_REG = 0xa0;
static final int ALARM_D_MIN_REG   = 0xb0;
static final int ALARM_D_HOUR_REG  = 0xc0;
static final int CTRL_1_REG        = 0xe0;
static final int CTRL_2_REG        = 0xf0;

static final int READ  = 0xc;
static final int WRITE = 0x8;

static final int WALE = 0x80; // C1
static final int DALE = 0x40; // C1
static final int WAFG = 0x02; // C2
static final int DAFG = 0x01; // C2

public void set (int reg, int arg) {
    int cmd = reg | WRITE;
    System.out.println("SPI WRITE 0x"
        + Integer.toHexString(reg)
        + " "
        + Integer.toHexString(arg));
    synchronized (spi) {
        assertCs();
        rawJEM.setField(1, 0x09001804, 4, 1);
        spi.writeReadData(cmd); // with AdcCsN high
        spi.writeReadData(arg);
        negateCs();
        rawJEM.setField(0, 0x09001804, 4, 1);
    }
}

public int get(int reg) {
    int cmd = reg | READ;
    int retval;
    synchronized (spi) {
        rawJEM.setField(1, 0x09001804, 4, 1);
        assertCs();
        spi.writeReadData(cmd); // with AdcCsN high
        retval = spi.writeReadData(0);
        negateCs();
        rawJEM.setField(0, 0x09001804, 4, 1);
    }
    return retval;
}
}
```

```

package nmc.sidekick;
import javax.comm.SerialPort;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.lang.Exception;
import nmc.common.TimestampedData;

class TimeoutException extends Exception {
    TimeoutException() {super();}
    TimeoutException(String s) {super(s);}
}

abstract public class StdInstrumentDriver implements IoResource, Instrument {

    String _name;

    byte[] _enter;
    byte[] _prompt;
    byte[] _sampleRequest;
    byte[] _sampleRequestEcho;
    byte[] _cfgEol;

    int _timeout;
    int _maxTries;
    int _maxAttnBytes;
    int _maxSampleBytes;
    int _initMaxTries;
    int _maxSkipBytes;
    int _cfgBufSize;

    /** Device communications port */
    SerialPort _commPort;

    /** Input from device */
    public InputStream _input;    // pulled from _commPort

    /** Output to device */
    public OutputStream _output; // pulled from _commPort

    /** Unique sensor ID */
    long _sensorID = 9999; // does this ID apply to, say, MicroCat,
                          // or to MicroCat serial# 149236 ?

    public void setSerialPort(SerialPort aPort) throws IOException {
        _commPort = aPort;
        _input     = aPort.getInputStream(); // may throw exception
        _output    = aPort.getOutputStream(); // may throw exception
    }

    /** Read a data sample from device communications interface. */
    public TimestampedData poll() throws Exception {
        int tries = 0;
        TimestampedData retVal;

        while (tries < _maxTries) {

```

```

            tries++;
            try {
                byte[] theSample = getSample();
                TimestampedData retVal = new TimestampedData(theSample);
                return retVal;
            } catch (TimeoutException e) {
                System.out.println("Driver: '"
                                   + _name
                                   + "' got timeout");
            } // other Exceptions fly past into the caller

            throw new Exception("Retry limit exceeded (retries=" + tries + ")");
        }

    private byte[] getSample() throws Exception {

        write( _output, _enter );

        skipUntil( _input, _prompt );

        write( _output, _sampleRequest );

        skipUntil( _input, _sampleRequestEcho );

        byte[] sampleBuf = new byte[_maxSampleBytes];
        int bytesCaptured = readUntil( _input, sampleBuf, _prompt );

        // copy the sample into the exactly-right-sized byte array
        byte[] retVal = new byte[bytesCaptured];
        for (int i = 0; i < bytesCaptured; i++) {
            retVal[i] = sampleBuf[i];
        }
        return retVal;
    }

    protected void write ( OutputStream ostream, byte[] src )
        throws IOException {

        write( ostream, src, src.length );

    }

    protected void write ( OutputStream ostream, byte[] src, int valid )
        throws IOException {

        ostream.write( src, 0, valid );

    }

    protected void skipUntil( InputStream istream, byte[] terminator )
        throws Exception {

        if (terminator == null)
            return; // no terminator means don't look for this

        ////////////////////////////////////////////
        // If we find that we are skipping huge amounts of bytes,
        // we can rewrite this -- for now, it's useful to reuse the
        // code in readUntil(...)
        //

```



```

byte[] buf = new byte[ _maxSkipBytes ];
readUntil( istream, buf, terminator );
}

public TimestampedData configure(byte[] cfgData) throws Exception {
    System.out.println("We are in the driver \"
        + _name
        + "\" with configuration data="
        + new String(cfgData));

    byte[] tempBuf = new byte[ _cfgBufSize ];
    ByteArrayOutputStream cfgRecord = new ByteArrayOutputStream();
    ByteArrayInputStream cfgSource = new ByteArrayInputStream( cfgData );

    // In the code below, we assume that the instrument is
    // full duplex, and that it is using end-of-line characters
    // that are acceptable. Instead of copying the configuration
    // requests from the source, we know that the instrument will
    // echo the source request with its notion of eol and then
    // send the result(s) (if any) with its notion of eol, and then
    // send the "prompt characters". The code's notion of prompt
    // includes the end-of-line character(s) that immediately precede
    // the prompt (so if the human sees "S>" the code's notion of
    // the prompt is something like "\r\nS>". Dumb terminals and
    // DOS are going to want to see \r\n, while Unix variants will
    // want to use \n as end-of-line.

    int count;
    while ( cfgSource.available() > 0 ) {
        count = readUntil( cfgSource, tempBuf, _cfgEol );
        System.out.println("forming string from tempBuf contents...");
        String s = new String( tempBuf, 0, count );
        System.out.println("returned (1) from readUntil with tempBuf=" + s);
        write( _output, tempBuf, count);
        System.out.println("wrote tempBuf to _output");
        write( _output, _enter );
        System.out.println("wrote _enter to _output");
        count = readUntil( _input, tempBuf, _prompt );
        String s2 = new String(tempBuf, 0, count);
        System.out.println("readUntil (2) returned with tempBuf=" + s2);
        write( cfgRecord, tempBuf, count + _prompt.length );
    }
    byte[] result = cfgRecord.toByteArray();
    cfgRecord.close();
    cfgSource.close();
    TimestampedData retval = new TimestampedData( result );
    return retval;
}

private void checkReadUntilArgs(InputStream input,
                                byte[] output,
                                byte[] terminator)
                                throws Exception {

    if (input == null) {
        throw new NullPointerException("readUntil called with "
            + "null InputStream");
    }

    if (output == null) {
        throw new NullPointerException("readUntil called with "

```

```

        + "null output buffer");
    }

    if (terminator == null) {
        throw new NullPointerException("readUntil called with "
            + "null terminator");
    }
}

public boolean foundTerminator(byte[] buffer,
                                int totalBytesRead,
                                byte[] terminator) {

    int termSz = terminator.length; // cache value for readability
    if (totalBytesRead < termSz)
        return false; // we can't match if we have too few items

    int offset = totalBytesRead-termSz; // starting point is termSz back
    int matches = 0;
    for (int i = 0; i < termSz; i++) {
        if (buffer[i+offset] != terminator[i]) {
            System.out.println("mismatch at i="
                + i
                + ", buffer="
                + buffer[i]
                + "terminator[i]="
                + terminator[i]);

            return false;
        }
    }
    return true;
}

public int readUntil(InputStream instream,
                    byte[] outbuf,
                    byte[] terminator )
    throws Exception {

    System.out.println("entering readUntil(terminator: "
        + (new String(terminator))
        + " )" );

    try {

        // throw exception if any bad args
        checkReadUntilArgs( instream, outbuf, terminator );

        // Total number of bytes read
        int bytesRead = 0;

        long t0 = System.currentTimeMillis();

        byte lastbyte = terminator[terminator.length - 1];

        // Read until we receive terminator
        while (true) {

            //////////////////////////////////////////

```

```
// Read one byte at a time - this might be VERY
// inefficient, depending on whether input is buffered!
//

if (instream.available() > 0) {

    int c = instream.read();
    System.out.println("read got: " + (char)c + "=" + c);
    outbuf[bytesRead++] = (byte)c;

    if (c == lastbyte) {
        if (foundTerminator(outbuf, bytesRead, terminator)) {
            System.out.println("found terminator: "
                               + terminator);
            return bytesRead - terminator.length;
        } else {
            System.out.println("matched last char but "
                               + "mismatched terminator: "
                               + terminator);
        }
    }
}

// if end of buffer, we missed last chance to
// recognize the terminator

if (bytesRead >= outbuf.length) {
    throw new Exception("Output buf in instrument driver "
                        + "exceeded (buffer max length is "
                        + outbuf.length
                        + ")");
}

}

long elapsed = System.currentTimeMillis() - t0;

if (elapsed > _timeout) {
    throw new TimeoutException("Timed out");
}
}
finally {
    System.out.println("leaving readUntil");
}
}

/** Read and discard all characters in serial input. */
public void flushInput() throws Exception {
    _input.skip(_input.available());
}

public Object getResourceType(ioResourceType aType) {
    if (aType == ioResourceType.DPA_BOARD)
        return null;
    else if (aType == ioResourceType.INSTRUMENT)
        return this;
    else
        return null; // probably should throw exception
}
```

}

```
package nmc.sidekick;

import com.ajile.io.BufferedInputStream;
import com.ajile.jem.rawJEM;

public class TestRunner {

    public static int a1;
    public static int a2;
    public static int a3;
    public static int a4;

    public static int loopsRun = 0; // Really just so compiler doesn't
                                   // optimize TestRunnerBkptFn out of
                                   // existence.

    public static int param0 = 0;
    public static int param1 = 1;
    public static int param2 = 2;
    public static int param3 = 3;
    public static int param4 = 4;

    /**
     * Starts the application.
     * @param args an array of command-line arguments
     */
    public static void main(java.lang.String[] args) {
        System.out.println("Hello from the TestRunner");

        int done = 0;

        while (done == 0) {
            TestRunnerBkptFn();
            done = runTest(param0, param1, param2, param3);
            System.out.println("Completed test, done=" + done);
        }

        public static void TestRunnerBkptFn() {
            loopsRun++;
        }

        public static int runTest( int testNum, int arg1, int arg2, int arg3) {
            Ramtest r = null;
            switch(testNum) {
                case 1:
                    // ramtest
                    r = new Ramtest();
                    break;
                case 2:
                    // flashtest
                    System.out.println("flashtest not implemented yet");
                    break;
                case 3:
                    // enetloopback
                    System.out.println("enetloopback not implemented yet");
                    break;
                case 4:
                    // uartloopback
                    System.out.println("doing loopback for com" + arg1);
                    UartLoopbackTest.doInternalLoopback(arg1);
                    break;
                case 5:
                    // adc
                    break;
                case 6:
                    // setdate
                    break;
                case 7:
                    // settime
                    break;
                case 8:
                    // getdate
                    break;
                case 9:
                    // gettime
                    break;
                case 10:
                    // setwakeup
                    break;
                case 11:
                    // pressure
                    break;
                case 12:
                    // temperature
                    break;
                case 13:
                    // dpaspi
                    break;
                case 14:
                    // dpaspiadc
                    break;
                case 15:
                    // quit
                    System.out.println("Quit requested, TestRunner exiting...");
                    return 1; // done
                default:
                    System.out.println("unrecognized operation number: " + testNum);
                    break;
            }

            return 0; // meaning not done yet
        }

        public static void fn1() {
            System.out.println("fn1 is running");
            Ramtest ramtest = new Ramtest();
        }

        public static void fn2() {
            System.out.println("fn2 is running");
        }
    }
}
```

sidekick\UartLoopbackTest.java

```
package nmc.sidekick;

// 200205161514
// 200206121652

import java.io.*;
import javax.microedition.io.Connection;
import javax.microedition.io.Connector;
import javax.microedition.io.Datagram;
import javax.microedition.io.DatagramConnection;
import com ajile.drivers.spi.*;
import javax.comm.SerialPort;
import javax.comm.CommPortIdentifier;
import com ajile.jem.rawJEM;

public class UartLoopbackTest {

    public static int UartBkpts = 0;
    public static void UartBkptHere() {
        UartBkpts++;
    }

    protected IoResource channel[] = new IoResource[64];
    /**
     * Starts the application.
     * @param args an array of command-line arguments
     */
    public static void main(java.lang.String[] args) {

        System.out.println("This is the UartLoopbackTest running.");
        UartLoopbackTest ult = new UartLoopbackTest();

        ult.initializeLoopbackPlatform();
        ult.doLoopback();

        ult.cutTxPower(0);
        ult.cutTxPower(1);

    }

    static int[] mcrAddrs = {
        0, // 0
        0x09000010, // 1
        0x09000810, // 2
        0x00900010, // 3
        0x00a00010, // 4
        0, // 5
        0, // 6
        0, // 7
        0, // 8
        0, // 9
        (0x00800010 + 0*0x40), // 10
        (0x00800010 + 1*0x40), // 11
        (0x00800010 + 2*0x40), // 12
        (0x00800010 + 3*0x40), // 13
        (0x00800010 + 4*0x40), // 14
        (0x00800010 + 5*0x40), // 15
        (0x00800010 + 6*0x40), // 16
        (0x00800010 + 7*0x40), // 17
    };

    static protected void doInternalLoopback(int ch) {

        UartBkptHere();
        System.out.println("Trying doInternalLoopback(" + ch + ")");
        if (ch < 0 || ch > mcrAddrs.length) {
            System.out.println("no such port as com" + ch);
            return;
        }
        if (mcrAddrs[ch] == 0) {
            System.out.println("no such port as com" + ch);
            return;
        }

        CommPortIdentifier cpi = null;
        SerialPort serialPort = null;
        OutputStream ostream = null;
        InputStream istream = null;
        String cpiString = "com" + ch;
        System.out.println("cpiString = \"" + cpiString + "\"");
        try {
            cpi = CommPortIdentifier.getPortIdentifier(cpiString);
        } catch (javax.comm.NoSuchPortException e1) {
            System.out.println("Exception on opening port: " + e1);
            return;
        }
        try {
            serialPort = (SerialPort)cpi.open("xy", 0);
            ostream = serialPort.getOutputStream();
            istream = serialPort.getInputStream();

        } catch (javax.comm.PortInUseException e2) {
            System.out.println("PortInUseException: " + e2);
            return;
        } catch (java.io.IOException e3) {
            System.out.println("Exception on opening port: " + e3);
            return;
        }

        int addr = mcrAddrs[ch];
        UartBkptHere();
        rawJEM.setField(1, addr, 4, 1);
        rawJEM.setField(1, addr, 4, 1);
        UartBkptHere();

        try {

            byte[] outgoing = "hubbahubba".getBytes();
            ostream.write(outgoing);
            int bytesWritten = outgoing.length;
            System.out.println("Seem to have succeeded in writing byte(s)");

            int a = 0;
            int loopctr = 0;
            while (a < bytesWritten) {
                a = istream.available();
                if (loopctr % 500 == 0) {
                    System.out.println("Tried " + loopctr + " times, "
                        + a
                        + " available for reading");
                }
            }
        }
    }
}
```

```
        }
        loopctr++;
    }
    System.out.println("Just got " + a + " bytes to read");

    byte[] b = new byte[a];
    istream.read(b);
    String s = new String(b, 0, a);
    System.out.println("Seem to have succeeded in reading " + s);
} catch (java.io.IOException e) {
    System.out.println("doLoopback got IOException: " + e);
}

    System.out.println("DONE with loopback test");
}

protected void doLoopback() {

    DpaBoard.DpaChannel dc =
        (DpaBoard.DpaChannel)channel[0].getResourceType(IoResourceType.DPA_BOARD);
    SerialPort sp = dc.instrumentPort;
    if (sp != null) {
        System.out.println("GOT sp");
    } else {
        System.out.println("BADBADBAD");
    }

    try {

        InputStream istream = sp.getInputStream();
        OutputStream ostream = sp.getOutputStream();
        byte[] outgoing = "hubbahubba".getBytes();
        ostream.write(outgoing);
        int bytesWritten = outgoing.length;
        System.out.println("Seem to have succeeded in writing byte(s)");

        int a = 0;
        int loopctr = 0;
        while (a < bytesWritten) {
            a = istream.available();
            if (loopctr % 500 == 0) {
                System.out.println("Tried " + loopctr + " times, "
                                   + a
                                   + " available for reading");
            }
            loopctr++;
        }
        System.out.println("Just got " + a + " bytes to read");

        byte[] b = new byte[a];
        istream.read(b);
        String s = new String(b, 0, a);
        System.out.println("Seem to have succeeded in reading " + s);
    } catch (java.io.IOException e) {
        System.out.println("doLoopback got IOException: " + e);
    }

    System.out.println("DONE with loopback test");
}
```

```
    }

    private void cutTxPower(int n) {
        if (channel[n] == null) {
            System.out.println("channel[" + n + "] is null!");
            return;
        }
        DpaBoard.DpaChannel dc =
            (DpaBoard.DpaChannel)channel[n].getResourceType(IoResourceType.DPA_BOARD);
        dc.channelCtrlReg.setTxLoPower();
        dc.channelCtrlReg.write();
    }

    protected void initializeLoopbackPlatform() {

        /*
         * This is all the platform-specific configuration stuff.
         * Everything except the association between the instruments
         * and the port is likely to remain hardwired. The association
         * of instruments to serial lines must be configurable.
         */

        System.out.println("Starting 'initializeLoopbackPlatform()'");
        DpaBoard board = new DpaBoard(SpiMaster.SPI_SLAVE_SELECT_0);
        // the next line should not work
        DpaBoard.DpaChannel aChannel = board.getLeftChannel();
        System.out.println("Finished creating left dpa channel");
        aChannel.initializeChannel();
        System.out.println("Finished initializing left dpa channel");
        SerialPort cereal = null;
        String serialPortName = "com10";
        try {
            cereal = openSerialPort(serialPortName);
            aChannel.setSerialPort(cereal);
            System.out.println("Finished creating the serial port on "
                               + serialPortName);
        } catch (IOException e) {
            System.out.println("could not open '"
                               + serialPortName
                               + "': " + e);
        }

        channel[0] = aChannel;
        System.out.println("Finished setting up left channel");
        System.out.println("Finished setting up left channel"
                           + " and wiring it to "
                           + serialPortName);

        aChannel = board.getRightChannel();
        System.out.println("Finished creating right dpa channel");
        aChannel.initializeChannel();
        System.out.println("Finished initializing right dpa channel");

        channel[1] = aChannel;
        System.out.println("Finished setting up right channel");
    }

    protected SerialPort openSerialPort(String portName) throws IOException {
        CommPortIdentifier cpi;
        SerialPort serialPort = null;
    }
}
```

```
try
{
    cpi = CommPortIdentifier.getPortIdentifier(portName);
    serialPort = (SerialPort)cpi.open("xyz", 0);

    System.out.println( "Succeeded in opening port named: "
        + portName);
}
catch ( javax.comm.NoSuchPortException e )
{
    System.out.println( "Unable to locate " + portName );
}
catch ( javax.comm.PortInUseException e )
{
    System.out.println( "Port " + portName + " is already in use!" );
}
catch ( Exception e )
{
    System.out.println( "Some other exception happened to "
        + portName
        + ". Exception: "
        + e );
}
return serialPort;
}
```

```
package nmc.sidekick;

import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;

import javax.comm.CommPort;
import javax.comm.CommPortIdentifier;
import javax.comm.NoSuchPortException;
import javax.comm.PortInUseException;
import javax.comm.SerialPort;

import com.ajile.jem.rawJEM;

public class xr16c2850 {

    public static int bkpts = 0;
    public static void BkptHere() {
        bkpts++;
    }

    /**
     * Starts the application.
     * @param args an array of command-line arguments
     */
    public static void main(java.lang.String[] args) {
        System.out.println("Hello, this is a test for the xr16c2850");
        try {
            CommPortIdentifier commPort3 = CommPortIdentifier.getPortIdentifier("com3");

            SerialPort serialPort3 = (SerialPort)commPort3.open("xy3", 0);
            OutputStream ostream3 = serialPort3.getOutputStream();
            InputStream istream3 = serialPort3.getInputStream();

            CommPortIdentifier commPort4 = CommPortIdentifier.getPortIdentifier("com4");

            SerialPort serialPort4 = (SerialPort)commPort4.open("xy4", 0);
            OutputStream ostream4 = serialPort4.getOutputStream();
            InputStream istream4 = serialPort4.getInputStream();

            System.out.println("opened com3 and com4");
            System.out.println("setting XR16C2850 port into internal loopback");
            BkptHere();
            rawJEM.setField(1, 0x00900010, 4, 1);
            rawJEM.setField(1, 0x00a00010, 4, 1);
            BkptHere();

            System.out.println("now sleeping 2 seconds");
            Thread.sleep(2000);

            String s = "howdy, partner";
            String t = "It's a lovely day in the neighborhood.";
            System.out.println("Sending string: " + s);
            int totLen3 = s.length();
            int totLen4 = t.length();
            ostream3.write(s.getBytes());
            ostream4.write(t.getBytes());
            int a = 0;
            int lasta = 0;
            int b = 0;
            int lastb = 0;

            while ((lasta < totLen3) || (lastb < totLen4)) {
                a += istream3.available();
                b += istream4.available();
                if (a != lasta) {
                    System.out.println("Just got " + (a - lasta) + " new bytes on com3");
                }
                if (b != lastb) {
                    System.out.println("Just got " + (b - lastb) + " new bytes on com4");
                }
                lasta = a;
                lastb = b;
            }
            byte[] rcvd3bytes = new byte[totLen3];
            byte[] rcvd4bytes = new byte[totLen4];
            int r3 = istream3.read(rcvd3bytes);
            int r4 = istream4.read(rcvd4bytes);
            String rcvd3str = new String(rcvd3bytes);
            String rcvd4str = new String(rcvd4bytes);
            System.out.println("Received string: " + rcvd3str + " on com3");
            System.out.println("Received string: " + rcvd4str + " on com4");

        } catch (NoSuchPortException e) {
            System.out.println("got NoSuchPortException exception: " + e);
        } catch (PortInUseException e) {
            System.out.println("got PortInUseException exception: " + e);
        } catch (IOException e) {
            System.out.println("got IOException exception: " + e);
        } catch (InterruptedException e) {
            System.out.println("got InterruptedException exception: " + e);
        } catch (Exception e) {
            System.out.println("got exception: " + e);
        }
    }
}
```