PSoC™ Design Challenge 2002
Cypress MicroSystems Contest

CIRCUIT CELLAR®

CYPRESS MICROSYSTEMS
A Subsidiary of Cypress Semiconductor

AUTOMATED TEST SYSTEM

By Lindsay Meek

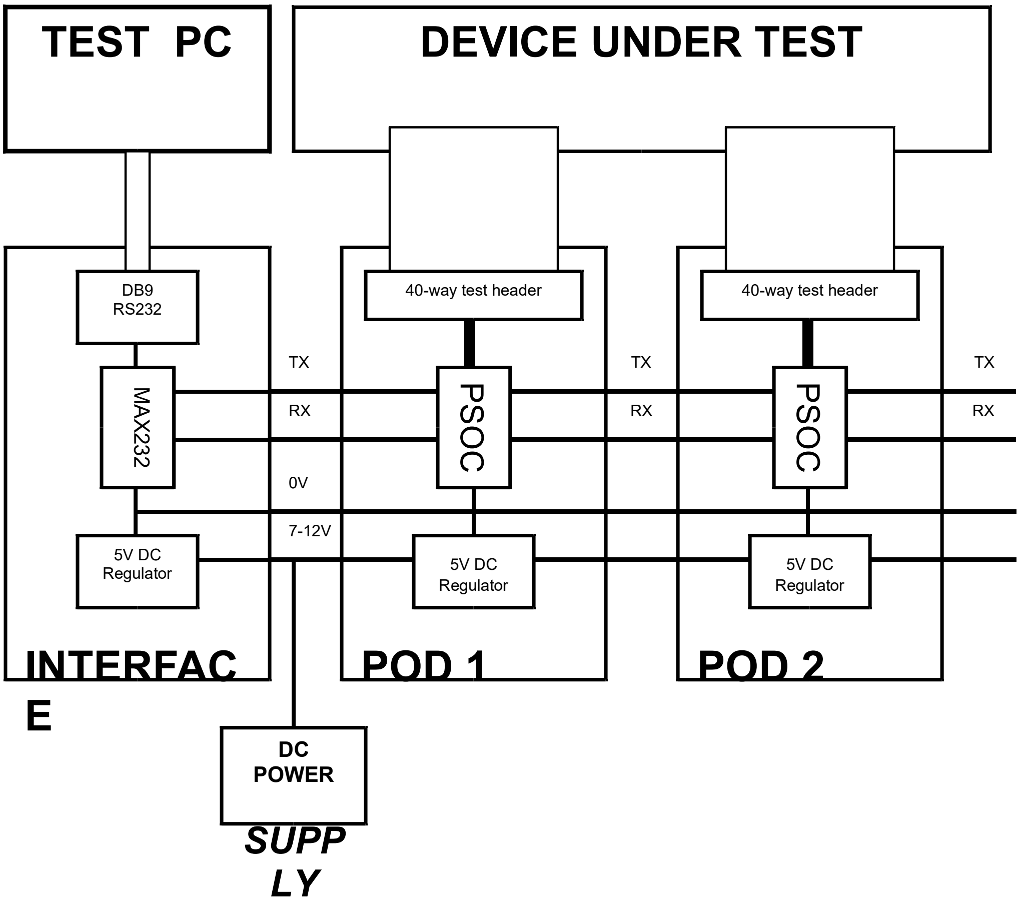
1. ABSTRACT

This project describes an automated test system that has been constructed using the PSOC controller as the main building block. The test system consists of a string of test pod units, each with a PSOC controller. These test pods are daisy-chained together in a bus arrangement to create enough analog and digital I/O to exercise a device-under-test. The pods connect to the device-under-test using a HP logic analyser compatible 40-way connector.

The test pod bus is interfaced to a PC, which interprets a test-vector scripting language, and translates it to low level commands. These commands are then sent to the test pods, and any measured values checked against the expected values to verify that the device-under-test is operating as expected.

The scripting language supports dynamic reconfiguration of pins of the PSOC devices, as either digital input, strong digital output, pulled-up digital output, pulled-down digital output, analog output, or analog input. The pins configurations may be altered from one test vector to the next, increasing the depth of tests that are possible, and reducing the need for custom hardware to be developed for each new test procedure.

2. BLOCK DIAGRAM



3. PSOC ARCHITECTURAL NOTES

- The low-speed 12-bit integrating ADC is used to sample input, rejecting high frequencies and overcoming the problem of a lack of input filtering.
- SampleCLK is to drive ADC data clock, and is adjustable between 8 and 480 Hz.
- A configurable number of samples are taken and averaged to give the output sample.
- A UART block commands from upstream test unit in the daisy chain.
- A UART block relays commands to downstream test unit the daisy chain.
- Four 6-bit DAC block are reconfigured to drive analog outputs on P02..P06
- The continuous time blocks are dynamically reconfigured to achieve four different input configurations: single-ended, double-ended, single-ended inverted, double-ended inverted. Single-ended configurations are sampled with respect to analog ground.

3.1 Analog Input configurations

3.1.1 Double-ended (V+=P0[0,2,4,6] V-=P0[1,3,5,7])

ACA00, ACA01=Instrumentation Amplifier
        Gain=2
        ACA01 = V+ Input = AinMux1
        ACA00 = V- Input = AinMux0
ACA02=Programmable Gain Amplifier
            Gain=User Selectable
        Input = ACA01
ASA12=Incremental ADC
        Input = ACA02

3.1.2 Single-ended (V+=P0[0..7])

ACA02=Programmable Gain Amplifier
        Gain=User Selectable
        Input = Analog Column Input Select 2
        AinMux2 is used for P0[1,3,5,7]
        AinMux2 is used for P0[0,2,4,6]
ASA12=Incremental ADC
        Input = ACA02

3.1.3 Double-ended inverted (V-=P0[0,2,4,6] V+=P0[1,3,5,7])

ACA00, ACA01 = Instrumentation Amplifier
        Gain = 2
        ACA01 = V+ Input = AinMux1
        ACA00 = V- Input = AinMux0
ACA02 = Programmable Gain Amplifier
            Gain = User Selectable
        Input = ACA01
ACA03 = Inverting Amplifier

Gain = -1
Input = ACA02
ASA12=Incremental ADC
Input = ACA03

3.1.4 Single-ended inverted (V-=P0[0..7])

ACA02=Programmable Gain Amplifier
Gain=User Selectable
Input = Analog Column Input Select 2
AinMux2 is used for P0[1,3,5,7]
AinMux2 is used for P0[0,2,4,6]
ACA03 = Inverting Amplifier
Gain = -1
Input = ACA02
ASA12=Incremental ADC
Input = ACA03

3.2 Analog Output Configurations

3.2.1 Analog Output P02

ASA23 = 6-bit Voltage Output DAC
Data Clock = Analog Clock 1 (Baud Clock 153.846 kHz)
Output = Analog Output Bus 3
Buf3 = P02

3.2.2 Analog Output P03

ASB20 = 6-bit Voltage Output DAC
Data Clock = Analog Clock 1 (Baud Clock 153.846 kHz)
Output = Analog Output Bus 0
Buf0 = P03

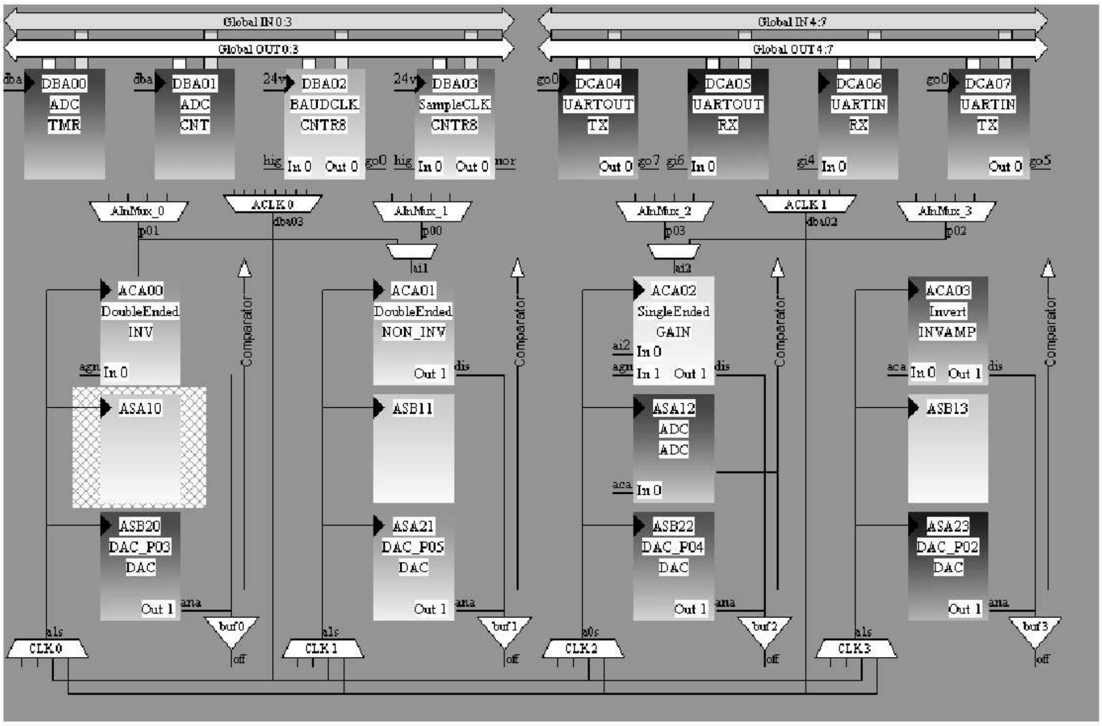
3.2.3 Analog Output P04

ASB22 = 6-bit Voltage Output DAC
Data Clock = Analog Clock 0 (Sample Clock 133kHz..1MHz)
Note: The maximum ADC rate = 60 Hz with this DAC enabled
Output = Analog Output Bus 2
Buf2 = P04

3.2.4 Analog Output P05

ASA21 = 6-bit Voltage Output DAC
Data Clock = Analog Clock 1 (Baud Clock 153.846 kHz)
Output = Analog Output Bus 1
Buf1 = P05

3.2.5 PSOC Designer Device Configuration



4. SERIAL COMMAND LANGUAGE

A simple text based command language is used by the test PC to communicate with the test pods. The general form of a command is an exclamation mark, followed by the destination test pod address (0 is the pod closest to the PC), then either a read or write command to a particular port. The test pods are daisy chained together using the concept of an ‘upstream’ UART and a ‘downstream’ UART. Commands are received from the upstream UART, and the destination address is checked. If it is zero, then the command is processed, else the command is relayed to the downstream unit with the address decremented by one. The serial parameters used by the UART blocks are:

- 19200 bps
- 8 data bits
- 1 stop bit
- No parity

4.1 Write Analog Output

!xWPGG[CR]

Where

x	Test pod address (0=first)
P	Port 0 bit 2,3,4 or 5
GG	Gain 00..3D

Example

!0W21E	Drive 50% voltage on P02

4.2 Write Digital Output

!x=PBMS[CR]

Where

x	Test pod address (0=first)
PB	Port 0..3 Bit 0..7
M	Drive Mode 0= 5.6K pulldown 1=CMOS 2=HighZ 3=5.6K pullup

S	State 0=Low 1=High

Example

!0=1010	Drive P1[0] CMOS low

4.3 Read Digital Input

!x?PBM[CR]

Where

x	Test pod address (0=first)
P	Port 0..3
B	Bit 0..7
M	Force pin to input first (1=Yes 0=No)

Example

!0?101	read P1[0], forcing to input first

Return Value

*N[CR]

Where

N	0 if input is low
	1 if input is high

4.4 Read Analog Input

!xR+-GGNNDD[CR]

Where

x	Test pod address (0=first)
+	Positive input, Port 0 even or odd bits. Specify 8 for AGND
-	Negative input, Port 0 odd or even bits. Specify 8 for AGND
GG	Gain constant where

Gain Constant	Gain	Gain Constant	Gain
00	0.0625	10	16.00
01	0.1250	11	8.000
02	0.1875	12	5.333
03	0.2500	13	4.000
04	0.3125	14	3.200
05	0.3750	15	2.667
06	0.4375	16	2.286
07	0.5000	17	2.000
08	0.5625	18	1.778
09	0.6250	19	1.600
0A	0.6875	1A	1.455
0B	0.7500	1B	1.333
0C	0.8125	1C	1.231
0D	0.8750	1D	1.143
0E	0.9375	1E	1.067
0F	1.0000	1F	1.000

Note for double-ended sampling, the instrumentation amplifier has a gain of 2.0 in addition to the gain specified above.

NN Number of samples
DD Sample clock divider (03-B8)
Sample rate = 24M/(Divider*65*256)

Example

!0R080FFF03 sample P0[0] – AGND. Gain=1 No Samples=255 Freq=480 Hz

Return value

*NNNN[CR]

Where

NNNN 2's compliment 11-bit measurement (-100%,0%,100% = F801,0000,07FF)

5. TEST SCRIPTING LANGUAGE

A simple scripting language is used to specify tests to be performed on the device-under-test. This is text based, and consists of a series of test configuration directives followed by a series of test vectors. A windows application forms the core of the interpreter, which sends simple serial commands to the test pods, and analyzes the response.

Comments may be inserted into the test file using a preceding semi-colon.

5.1 Directive Syntax

Directives have the general form '$directive {value}'. The following directives are supported.

5.1.1 $EXIT

Terminate the tests at this point. This can be used to short-circuit test scripts for debugging purposes.

e.g.

…test vectors being developed and debugged…
$EXIT
…test vectors we don't want to test yet…

5.1.2 $MSG Text

Dumps a text string message to the console.

e.g.

$MSG Now about to test supply voltage..

5.1.3 $VAR NewVariableName=PORT

Creates a symbolic variable assigned to the nominated port. This is useful for improving the readability of the test output data.

e.g.

$VAR Vcc=P01

Creates the symbolic variable 'Vcc' and assigns it to pod 0, port 0, bit 1.

Note that the system supports a maximum of 8 test pods, 4 test ports per pod, and 8 bits per port. To reference test pods, the 'P' port prefix is modified as per the table below.

Port label prefix	Test pod referenced
P	Test pod 0. The first test pod.
Q	Test pod 1.
R	Test pod 2.

S	Test pod 3.
T	Test pod 4.
U	Test pod 5.
V	Test pod 6.
W	Test pod 7. The last test pod.

5.1.4 $TRACE Level

Sets the amount of test information to dump to the test output. Level is an integer from 0 to 2, and defaults to 2. The trace levels are:

0	Only reports test failures
1	Dumps test failures & test vector output
2	Dumps test vector failures, input & output (default)
3	Dumps diagnostic information

e.g.

```
$TRACE 2
```

5.1.5 $REPEAT N..$ENDR

These two directives allow a series of tests to be repeated N times, where N is an integer starting from 1.

e.g.

```
$REPEAT 5
$MSG Hello
$ENDR
```

Generates

```
Hello
Hello
Hello
Hello
Hello
```

5.1.6 $RATE N

This directive controls the delay between successive test vectors, specified as N milliseconds. Note that the communications delays may prevent very fast rates (N < 100) from being achieved. If this is the case, the tests will be conducted as fast as possible.

e.g.

```
$RATE 100
```

Sets the test execution rate to one test vector per 100 ms.

5.1.7 $TIMEOUT N

This directive controls how long to wait for a test response, specified as N milliseconds. e.g.

$TIMEOUT 100

Directs the test system to wait a maximum of 100 ms for a test response, before generating a test fault.

5.1.8 $ORDER [PORT SPECIFICATION]

This directive controls the formatting of the test vectors, specifically the order in which test vectors are expected.

e.g.

$ORDER Vcc P02 P21

Directs the test system that the test vectors will be of the form, Vcc [Defined earlier as P0[1]], P0[2] and P2[1].

There is also support for double-ended ports and inverted ports, typically used for analog inputs. The double-ended ports are specified as two ports separated by a hyphen, with the positive port specified first. The inverted ports are specified with a hyphen prefix. E.g.

$ORDER Vcc-P02 -P21

Directs the test system that, in addition to inverted port P21, there is a double-ended test vector, with Vcc as the positive leg and P02 as the negative leg.

5.1.9 $SAMPLES N

This directive specifies how many samples to average for analog input measurements. This defaults to 1. The maximum value is 255 samples.

e.g.

$SAMPLES 100

Directs the test system to average 100 samples before testing the result.

5.1.10 $FREQ N

This directive specifies the sampling frequency for the analog input measurements. Note that the 12-bit integrating ADC on the PSOC controller discards frequencies above the sampling frequency. The range of N is 8 to 480 Hz. The default value is 57 Hz.

e.g.

$FREQ 100

Directs the test system to sample at 100 Hz, and reject frequencies above this level.

5.2 Test Vector Syntax

The test vectors specify the actual test operations to be performed by the automated test system. A vector consists of a series of outputs to be driven and the expected inputs that the device-under-test should generate in response. A vector is completely specified on one line of the text file, and **each test element is separated by a space**. The order of test elements needs to be specified with the $ORDER directive before test vectors can be entered. See section 5.1.8 for more information on this directive. As a port pin may be configured as a digital output, digital input, analog output or analog input, minor syntax variations are used to imply the test to be performed.

5.2.1 Drive Digital Output

0	Drive pin as CMOS, low
1	Drive pin as CMOS, high
0U	Drive pin as resistive pull-up, low
1U	Drive pin as resistive pull-up, high
0D	Drive pin as resistive pull down, low
1D	Drive pin as resistive pull down, high
C	Drive pin as CMOS, clocked low, high, low
CU	Drive pin as resistive pull-up, clocked low, high, low
CD	Drive pin as resistive pull down, clocked low, high, low
K	Drive pin as CMOS, clocked high, low, high
KU	Drive pin as resistive pull-up, clocked high, low, high
KD	Drive pin as resistive pull down, clocked high, low, high
X	Drive pin as high impedance
R	Drive pin as CMOS, pseudo random
RU	Drive pin as resistive pull-up, pseudo random
RD	Drive pin as resistive pull down, pseudo random

5.2.2 Test Digital Input

L	Test digital input is low
H	Test digital input is high
Z	Test digital input is high impedance (Pull-up and pull-down, check state is present)
*	Don’t care – read input value and substitute in vector

5.2.3 Drive Analog Output

A.B	Where A.B is a real number between 0 and 1. e.g. 0.5 Note that only P0[2],P0[3],P0[4] and P0[5] can be analog outputs

5.2.4 Test Analog Input

A.B-C.D	Where A.B and C.D are real numbers between -1 and 1. Note that only P0[0]..P0[7] can measure analog inputs. Note also that the gain defaults to 1.0 if not specified.

E.F,A.B-C.D	Same as above, but specify an input gain of E.F, between 0 and 16

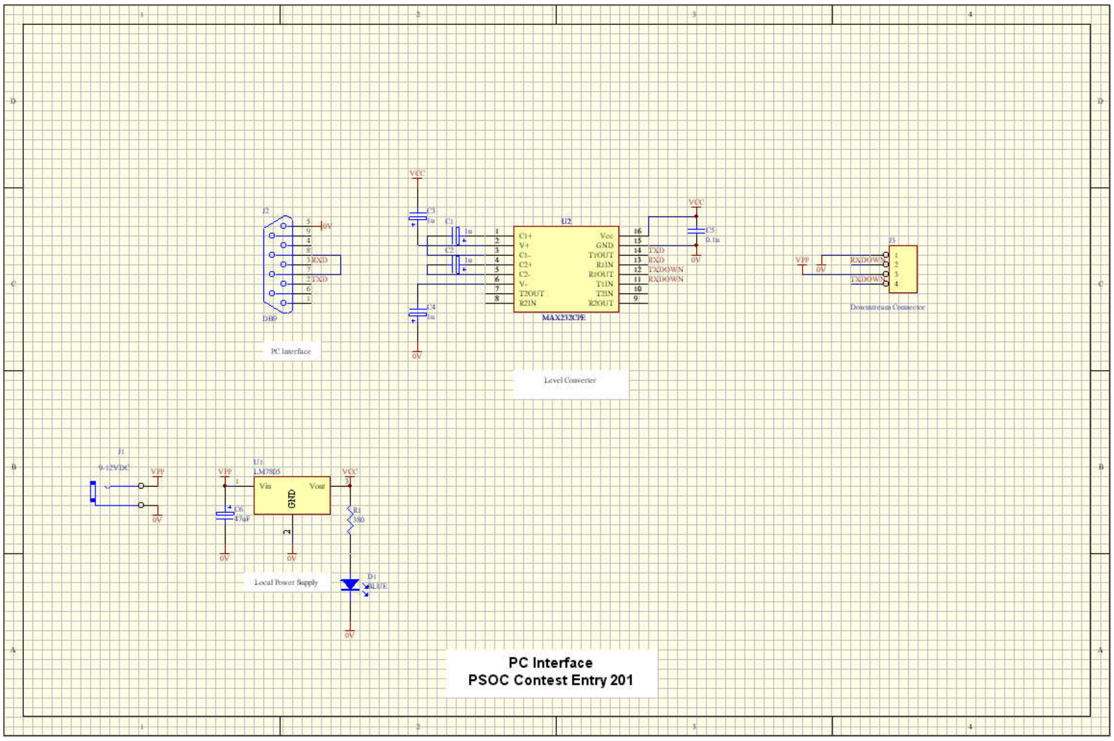
e.g.

2.0,0.5-1.0	Input gain 2.0, test analog voltage is between 50% and 100%.

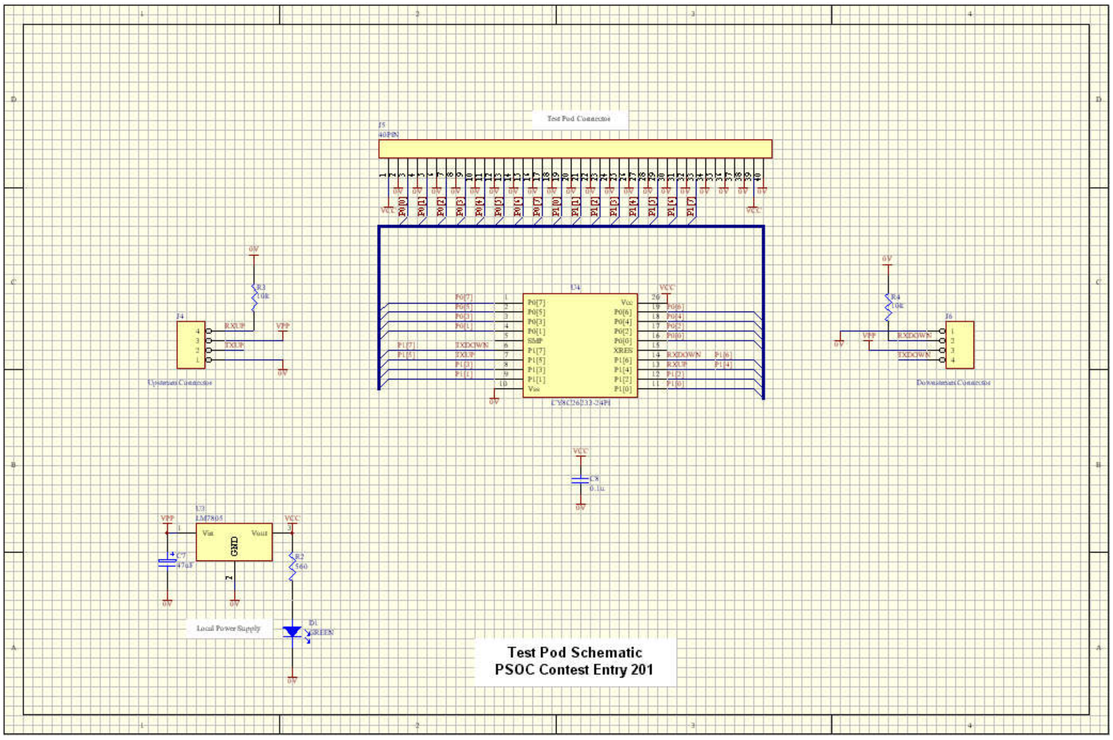
5.3 Example

```
;
;Example automated test script
;
;Tests a 50% resistive divider with IN and OUT
;
$VAR IN=P01 OUT=P00
$RATE 100
$TIMEOUT 1000
$MSG Applying Forward Voltage Test
$ORDER IN OUT
$REPEAT 10
1 0.1-0.3
$ENDR
```

6. INTERFACE SCHEMATIC



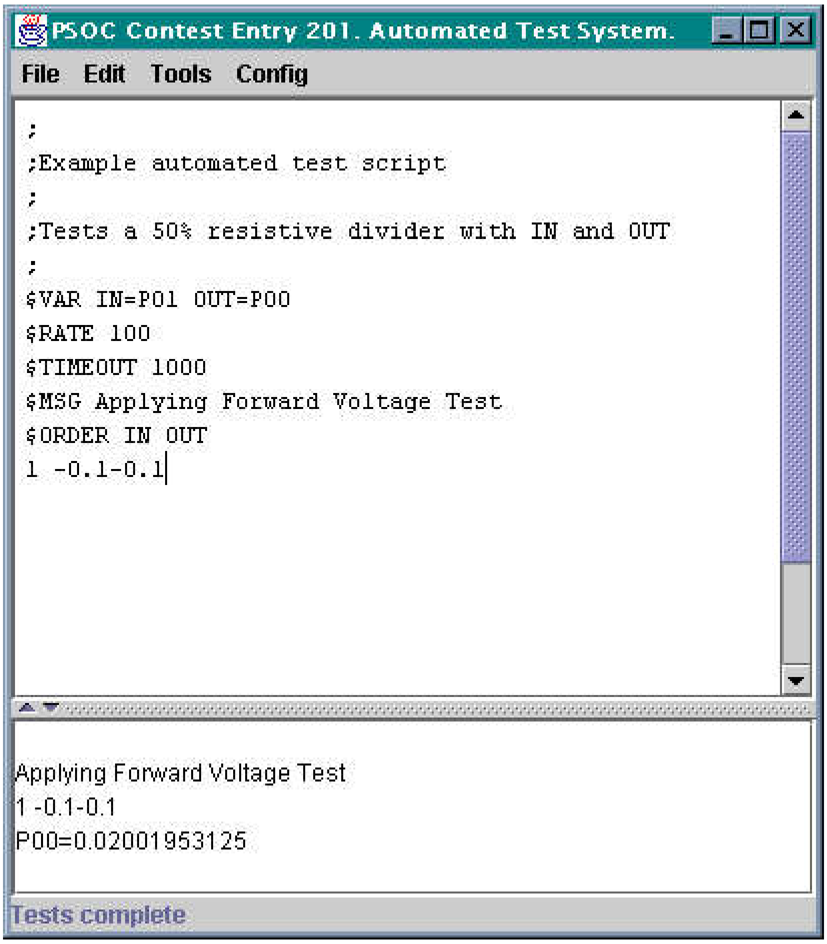
7. TESTPOD SCHEMATIC



8. BILL OF MATERIALS

Component	Qty	Cost	Total	
7805 1A 5V voltage regulator	3	$0.60	$1.80	
PCB Mount 2.5mm DC Power Plug	1	$0.50	$0.50	
DB9 Female	1	$1.00	$1.00	
10uF 16V electro	3	$0.40	$1.20	
IDC 40 way vertical header	2	$0.90	$1.80	
20 pin DIL socket	2	$0.20	$0.40	
MAX232	1	$3.00	$3.00	
Vero Board	3	$2.75	$8.25	
IDC 40 way cable	1	$3.35	$3.35	
Hookup wire	1	$1.25	$1.25	
LED	3	$0.75	$2.25	
Wire wrap strip 36 way	1	$2.50	$2.50	
Steel Screws 25 6x3mm	1	$0.95	$0.95	
Steel Spacers 8 10x3mm	2	$1.95	$3.90	
				$32.15

9. SCREEN CAPTURE



```
;
;Example automated test script
;
;Tests a 50% resistive divider with IN and OUT
;
$VAR IN=P01 OUT=P00
$RATE 100
$TIMEOUT 1000
$MSG Applying Forward Voltage Test
$ORDER IN OUT
1 -0.1-0.1
```

```
Applying Forward Voltage Test
1 -0.1-0.1
P00=0.02001953125
```

Tests complete

10. PSOC SOURCE LISTING

```
;
;Circuit Cellar Cypress PSOC Design Contest
;Contest Entry 201
;
;Title:                 Reconfigurable Test System
;Version:               21/5/02
;Description:           This is the main code for reconfiguring and
;                       operating the PSOC as a cascadable test POD
;

export _main

include "m8c.inc"       ;include m8c specific declarations

;
;Some useful macro definitions
;


;
;load effective address of operand into A:X
;
macro       lea
            mov         a,#>@0
            mov         x,#<@0
endm

;
;and/or ABF_CR register bits (write only register)
;
macro       loadabf
            push  a
            mov         a,[ABF_TMP]
            M8C_SetBank1
            mov         reg[ABF_CR],a
            M8C_SetBank0
            pop         a
            endm

macro       andabf
            and         [ABF_TMP],#@0
            loadabf
endm

macro       orabf
            or          [ABF_TMP],#@0
            loadabf
endm

;
;DDR loading macros (write only register)
;
macro       loadddr0
            push  a
            mov         a,[PRTDM0_T+0]
            mov         reg[PRT0DM0],a
            mov         a,[PRTDM1_T+0]
            mov         reg[PRT0DM1],a
            pop         a
endm

macro       loadddr1
            push  a
            mov         a,[PRTDM0_T+1]
            mov         reg[PRT1DM0],a
            mov         a,[PRTDM1_T+1]
            mov         reg[PRT1DM1],a
            pop         a
endm

macro       loadddr2
            push  a
            mov         a,[PRTDM0_T+2]
            mov         reg[PRT2DM0],a
            mov         a,[PRTDM1_T+2]
            mov         reg[PRT2DM1],a
            pop         a
endm

;
;Carry set/clear
;
macro       clc
            and         F,#~4
endm

macro       sec
            or          F,#4
endm

;
;Echo a character upstream
;
macro       echo
            push  a
            mov         a,#@0
            call  TxIN
            pop         a
endm

MAXLINE:    equ         16                      ;number of characters in a line

;
;Register memory usage
;
area bss(RAM)
      TMP:        blk   1                       ;working register
      TMP2:       blk   1                       ;working mask register
      TMP3:       blk   1                       ;working index register
      ABF_TMP:    blk   1                       ;stores current state of ABF_CR
      PRTDM0_T:   blk   3                       ;stores current state of PRT0..2DM0
      PRTDM1_T:   blk   3                       ;stores current state of PRT0..2DM1
      PORT:       blk   2                       ;active port(s)
      NOSAMPLES:  blk   1                       ;number of samples to take
      GAINK:      blk   1                       ;gain constant
      RESULT:     blk   3                       ;averaging accumulator
      SAMPLECLK:  blk   1                       ;clock divider for ADC
      RX_STATUS:  blk   1                       ;temp uart rx status
      COMMBUF:    blk   MAXLINE                 ;incoming command storage
      RAMPADDING: blk   64-(19+MAXLINE)         ;skip RX corruption area

;
;Code memory
;
area text(ROM,REL)


;
;Analog input section
;
include "doubleended.inc"
include "invert.inc"
include "singleended.inc"
include "adc.inc"

;
;Convert bit number to bit mask
;
MASK1LUT:
            db          1,2,4,8,16,32,64,128                ;OR operation
MASK0LUT:
            db          ~1,~2,~4,~8,~16,~32,~64,~128        ;AND operation
```

```
;
;Configure ADC for single ended input and initialise
;
;A=Input select P0[0]..P0[7]
;GAINK = gain
;
SingleEndedPort:
            and         a,#7
            mov         [TMP],a
            rrc         a                             ;check if port is odd or even
            jc          OddPort
EvenPort:
            mov         a,[TMP]
            index       MASK0LUT
            mov         [TMP2],a
            M8C_SetBank1
            mov         a,[PRTDM0_T+0]
            and         a,[TMP2]
            mov         reg[PRT0DM0],a                ;select high Z on that pin
            mov         [PRTDM0_T+0],a                ;update local latched value
            mov         a,[TMP]
            index       MASK1LUT
            mov         [TMP2],a
            mov         a,[PRTDM1_T+0]
            or          a,[TMP2]
            mov         reg[PRT0DM1],a
            mov         [PRTDM1_T+0],a

            orabf       40h                           ;select mux 3 (even ports)
            rlc         [TMP]
            rlc         [TMP]
            rlc         [TMP]
            rlc         [TMP]
            rlc         [TMP]
            and         [TMP],#128+64
            M8C_SetBank0
            mov         a,reg[AMX_IN]
            and         a,#~(128+64)
            or          a,[TMP]
            mov         reg[AMX_IN],a                 ;select even port
            jmp         DonePort
OddPort:
            andabf      ~40h                          ;select mux 2 (odd ports)
            rlc         [TMP]
            rlc         [TMP]
            rlc         [TMP]
            and         [TMP],#32+16
            M8C_SetBank0
            mov         a,reg[AMX_IN]                 ;select odd port
            and         a,#~(32+16)
            or          a,[TMP]
            mov         reg[AMX_IN],a
DonePort:

            mov         a,reg[ACA02CR1]
            and         a,#~(2+4+16+8)
            or          a,#1+32                       ;select column input
            mov         reg[ACA02CR1],a

            mov         a,[GAINK]
            call        SingleEnded_SetGain           ;user selected gain
            mov         a,#SingleEnded_MEDPOWER
            call        SingleEnded_Start

            lcall       Invert_Stop                   ;shut down inverting amp
            lcall       DoubleEnded_Stop              ;shut down balanced amp

            M8C_SetBank0
            and         reg[ASA12CR1],#~(128+64+32)   ;select ASA12 input for ACA02

            clc
            ret
```

```
;
;Configure ADC for single ended inverted input and initialise
;
;A=Input select P0[0]..P0[7]
;GAINK = gain
;
SingleEndedInvPort:
            and         a,#7
            mov         [TMP],a
            rrc         a                      ;check if port is odd or even
            jc          OddPort2
EvenPort2:
            mov         a,[TMP]
            index       MASK0LUT
            mov         [TMP2],a
            M8C_SetBank1
            mov         a,[PRTDM0_T+0]
            and         a,[TMP2]
            mov         reg[PRT0DM0],a         ;select high Z on that pin
            mov         [PRTDM0_T+0],a
            mov         a,[TMP]
            index       MASK1LUT
            mov         [TMP2],a
            mov         a,[PRTDM1_T+0]
            or          a,[TMP2]
            mov         reg[PRT0DM1],a
            mov         [PRTDM1_T+0],a

            orabf       40h                    ;select mux 3 (even ports)
            rlc         [TMP]
            rlc         [TMP]
            rlc         [TMP]
            rlc         [TMP]
            rlc         [TMP]
            and         [TMP],#128+64
            M8C_SetBank0
            mov         a,reg[AMX_IN]
            and         a,#~(128+64)
            or          a,[TMP]
            mov         reg[AMX_IN],a          ;select even port
            jmp         DonePort2
OddPort2:
            andabf      ~40h                   ;select mux 2 (odd ports)
            rlc         [TMP]
            rlc         [TMP]
            rlc         [TMP]
            and         [TMP],#32+16
            M8C_SetBank0
            mov         a,reg[AMX_IN]          ;select odd port
            and         a,#~(32+16)
            or          a,[TMP]
            mov         reg[AMX_IN],a
DonePort2:

            mov         a,reg[ACA02CR1]
            and         a,#~(2+4+16+8)
            or          a,#1+32                ;select column input
            mov         reg[ACA02CR1],a

            mov         a,[GAINK]
            call        SingleEnded_SetGain    ;user selected gain
            mov         a,#SingleEnded_MEDPOWER
            call        SingleEnded_Start

            mov         a,#Invert_MEDPOWER
            lcall       Invert_Start           ;start inverting amp
            mov         a,#Invert_G1_00        ;gain = -1.0
            lcall       Invert_SetGain
            lcall       DoubleEnded_Stop       ;shut down balanced amp

            M8C_SetBank0
            and         reg[ASA12CR1],#~(64+32)      ;select ASA12 input for ACA03
            or          reg[ASA12CR1],#128

            clc
            ret
```

```
;
;Configure ADC for double ended input and initialise
;
;A.MSN      = V+ input P0[0]..P0[7]
;A.LSN  = V- input P0[0]..P0[7]
;GAINK  = Gain
;
;Carry is clear if configuration is possible
;
DoubleEndedPort:

            mov         [TMP],a

            and         a,#7
            index       MASK0LUT
            mov         [TMP2],a
            M8C_SetBank1
            mov         a,[PRTDM0_T+0]
            and         a,[TMP2]
            mov         reg[PRT0DM0],a    ;select high Z on that V- input
            mov         [PRTDM0_T+0],a
            mov         a,[TMP]
            and         a,#7
            index       MASK1LUT
            mov         [TMP2],a
            mov         a,[PRTDM1_T+0]
            or          a,[TMP2]
            mov         reg[PRT0DM1],a
            mov         [PRTDM1_T+0],a

            mov         [TMP],a
            rrc         a
            rrc         a
            rrc         a
            rrc         a
            and         a,#7
            index       MASK0LUT
            mov         [TMP2],a
            M8C_SetBank1
            mov         a,[PRTDM0_T+0]
            and         a,[TMP2]
            mov         reg[PRT0DM0],a    ;select high Z on that V+ input
            mov         [PRTDM0_T+0],a
            mov         a,[TMP]
            rrc         a
            rrc         a
            rrc         a
            rrc         a
            and         a,#7
            index       MASK1LUT
            mov         [TMP2],a
            mov         a,[PRTDM1_T+0]
            or          a,[TMP2]
            mov         reg[PRT0DM1],a
            mov         [PRTDM1_T+0],a

            tst         [TMP],#1          ;check if V+ and V- are even/odd or odd/even
            jnz         NegIsOdd
NegIsEven:

            tst         [TMP],#16
            jz          InvalidConfig

            or          [TMP2],#1         ;use inverter

            mov         a,[TMP]
            rlc         [TMP]
            rlc         [TMP]
            rlc         [TMP]
            rlc         [TMP]
            and         [TMP],#240
            rrc         a
            rrc         a
            rrc         a
            rrc         a
            and         a,#15
            or          [TMP],a           ;swap nibbles
            jmp         ValidConfig

NegIsOdd:

            and         [TMP2],#~1        ;don't use invert
            tst         [TMP],#16
            jz          ValidConfig
InvalidConfig:
            sec
            ret                           ;both odd.. error!

ValidConfig:
            M8C_SetBank0
            mov         a,reg[AMX_IN]
            and         a,#~(1+2)         ;mask column 0 (V-) mux select
            tst         [TMP],#2
            jz          NoSet0
            or          a,#1
NoSet0:
            tst         [TMP],#4
            jz          NoSet1
            or          a,#2
NoSet1:
            and         a,#~(4+8)         ;mask column 1 (V+) mux select
            tst         [TMP],#32
            jz          NoSet2
            or          a,#4
NoSet2:
            tst         [TMP],#64
            jz          NoSet3
            or          a,#8
NoSet3:
            mov         reg[AMX_IN],a

            mov         a,#DoubleEnded_MEDPOWER       ;power up differential amp
            lcall       DoubleEnded_Start
            mov         a,#DoubleEnded_G2_00          ;nominal gain of 2.0
            lcall       DoubleEnded_SetGain

            M8C_SetBank0
            and         reg[ACA02CR1],#~(1+2+4+16+8)  ;select ACA02 input to ACA01/AGND
            or          reg[ACA02CR1],#32

            mov         a,[GAINK]
            call        SingleEnded_SetGain           ;user selected gain
            mov         a,#SingleEnded_MEDPOWER
            call        SingleEnded_Start             ;power up cascade opamp

            tst         [TMP2],#1                     ;invert enabled?
            jnz         InvertOn                      ;yes, turn it on

InvertOff:

            M8C_SetBank0
            and         reg[ASA12CR1],#~(128+64+32)   ;select ASA12 input for ACA02


            call        Invert_Stop                   ;shut down unused opamp
            jmp         InvertDone

InvertOn:

            M8C_SetBank0
            or          reg[ASA12CR1],#128
            and         reg[ASA12CR1],#~(64+32)       ;select ASA12 input to ACA03


            mov         a,#Invert_MEDPOWER
            call        Invert_SetGain                ;nominal gain of 1.0
            mov         a,#Invert_G1_00
            call        Invert_Start

InvertDone:

            clc
            ret
```

```
;
;Initialise ADC and sample clock
;
;Sample rate = Data clock / (65*256)     must be between 7.8 and 480
;
InitADC:
            mov         [NOSAMPLES],#1
            mov         [SAMPLECLK],#25                    ;57 hz
            call        SampleCLK_DisableInt
            lcall       ADC_Stop
            call        Invert_Stop
            call        SingleEnded_Stop
            lcall       DoubleEnded_Stop
            ret

;
;Initialise ADC and take some samples, apply averaging filter
;
;Sample rate = Data clock / (65*256)     must be between 7.8 and 480
;
;Carry is set if sample rate is invalid
;
;Returns data in X:A
;
RunADC:
            cmp         [NOSAMPLES],#0
            jz          ADCInvalid
            mov         a,[SAMPLECLK]
            cmp         a,#3
            jc          ADCInvalid
            mov         a,[SAMPLECLK]
            cmp         a,#184
            jc          ADCOK
ADCInvalid:
            sec
            ret
ADCOK:

            call        SampleCLK_WritePeriod

            mov         a,[SAMPLECLK]
            clc
            rrc         a                            ;compare = 50% duty
            call        SampleCLK_WriteCompareValue
            call        SampleCLK_Start

            mov         a,#ADC_HIGHPOWER
            lcall       ADC_Start

            call        SettleDelay

            mov         [RESULT],#0
            mov         [RESULT+1],#0
            mov         [RESULT+2],#0                ;reset accumulator

            mov         a,[NOSAMPLES]                ;how many samples
            mov         [TMP2],a
            lcall       ADC_GetSamples               ;start sampler

LoopADC:
            ADC_ISDATA                               ;poll flag
            jz          LoopADC                      ;wait for ADC
            ADC_CLEARFLAG                            ;reset flag

            ADC_GETDATA                              ;get data
            swap        a,x
            add         a,8                          ;convert to unsigned
            swap        a,x
            add         [RESULT],a
            swap        a,x
            adc         [RESULT+1],a
            adc         [RESULT+2],#0
            dec         [TMP2]
            jnz         LoopADC

            lcall       ADC_Stop
            cmp         [NOSAMPLES],#1
            jz          SkipDiv                      ;any need to divide?

;
;Divide 24-bit result by number of 8-bit number of samples
;
            mov         [TMP2],#24
            mov         [TMP],#0
            clc
DivLoop:
            rlc         [RESULT]
            rlc         [RESULT+1]
            rlc         [RESULT+2]
            rlc         [TMP]
            jc          DivOver
            mov         a,[TMP]
            sub         a,[NOSAMPLES]
            jc          DivUnder
            mov         [TMP],a
            dec         [TMP2]
            jz          DivDone
            sec
            jmp         DivLoop

DivOver:
            mov         a,[NOSAMPLES]
            sub         [TMP],a
            dec         [TMP2]
            jz          DivDone
            sec
            jmp         DivLoop

DivUnder:
            dec         [TMP2]
            jz          DivDone
            clc
            jmp         DivLoop

DivDone:

            rlc         [RESULT]
            rlc         [RESULT+1]
            rlc         [RESULT+2]
SkipDiv:
            sub         [RESULT+1],8
            mov         a,[RESULT]
            mov         x,[RESULT+1]
            clc
            ret


;
;Digital input/output section
;

;
;Set data direction on pin for strong drive
;
;Inputs:    A=Port / Bit
;
;Outputs:   X=Port Offset
;           A=Port / Bit
;
SetDDROut:
            mov         [TMP],a
            rrc         a
            rrc         a
            rrc         a
            and         a,#3                   ;port number
            swap        a,x
            mov         a,[TMP]
            and         a,#7
            index       MASK1LUT               ;lookup mask for OR'ing
            mov         [TMP2],a
            M8C_SetBank1
            mov         a,[X+PRTDM0_T]
            or          a,[TMP2]
            mov         reg[X+PRT0DM0],a
            mov         [X+PRTDM0_T],a
            mov         a,[TMP]
            mov         a,#7
            index       MASK0LUT
            mov         [TMP2],a
            mov         a,[X+PRTDM1_T]
            and         a,[TMP2]
            mov         reg[X+PRT0DM1],a        ;strong drive
            mov         [X+PRTDM1_T],a
            mov         a,[TMP]
            M8C_SetBank0
            ret


;
;Set data direction on pin for pulldown
;
;Inputs:    A=Port / Bit
;
;Outputs:   X=Port Offset
;           A=Port / Bit
;
SetDDRPulldown:
            mov         [TMP],a
            rrc         a
            rrc         a
            rrc         a
            and         a,#3                    ;port number
            swap        a,x
            mov         a,[TMP]
            and         a,#7
            index       MASK0LUT                ;lookup mask for AND'ing
            mov         [TMP2],a
            M8C_SetBank1
            mov         a,[X+PRTDM0_T]
            and         a,[TMP2]
            mov         reg[X+PRT0DM0],a
            mov         [X+PRTDM0_T],a
            mov         a,[X+PRTDM1_T]
            and         a,[TMP2]
            mov         reg[X+PRT0DM1],a        ;pulldown
            mov         [X+PRTDM1_T],a
            mov         a,[TMP]
            M8C_SetBank0
            call        SettleDelay
            ret

;
;Set data direction on pin for pullup
;
;Inputs:    A=Port / Bit
;
;Outputs:   X=Port Offset
;           A=Port / Bit
;
SetDDRPullup:
            mov         [TMP],a
            rrc         a
            rrc         a
            rrc         a
            and         a,#3                    ;port number
            swap        a,x
            mov         a,[TMP]
            and         a,#7
            index       MASK1LUT                ;lookup mask for OR'ing
            mov         [TMP2],a
            M8C_SetBank1
            mov         a,[X+PRTDM0_T]
            or          a,[TMP2]
            mov         reg[X+PRT0DM0],a
            mov         [X+PRTDM0_T],a
            mov         a,[X+PRTDM1_T]
            or          a,[TMP2]
            mov         reg[X+PRT0DM1],a        ;pullup
            mov         [X+PRTDM1_T],a
            mov         a,[TMP]
            M8C_SetBank0
            call        SettleDelay
            ret

;
;Set data direction on pin for high Z input
;
;Inputs:    A=Port / Bit
;
;Outputs:   X=Port Offset
;           A=Port / Bit
;
SetDDRIn:
            mov         [TMP],a
            rrc         a
            rrc         a
            rrc         a
            and         a,#3                    ;port number
            swap        a,x
            mov         a,[TMP]
            and         a,#7
            index       MASK0LUT                ;lookup mask
            mov         [TMP2],a
            M8C_SetBank1
            mov         a,[X+PRTDM0_T]
            and         a,[TMP2]
            mov         reg[X+PRT0DM0],a
            mov         [X+PRTDM0_T],a
            mov         a,[TMP]
            mov         a,#7
            index       MASK1LUT
            mov         [TMP2],a
            mov         a,[X+PRTDM1_T]
            or          a,[TMP2]
            mov         reg[X+PRT0DM1],a  ;high Z
            mov         [X+PRTDM1_T],a
            mov         a,[TMP]
            M8C_SetBank0
            call        SettleDelay
            ret

;
;Drive digital output high
;
;A    P0[0]..P2[7]
;
SetDigitalOutput:
            and         a,#7
            index       MASK1LUT
            mov         [TMP2],a                ;lookup bit
            M8C_SetBank0
            mov         reg[X+PRT0DR],a
            or          a,[TMP2]
            mov         a,reg[X+PRT0DR]         ;set bit
            ret

;
;Drive digital output low
;
;A    P0[0]..P2[7]
;
ClrDigitalOutput:
            and         a,#7
            index       MASK0LUT
            mov         [TMP2],a                ;lookup bit
            M8C_SetBank0
            mov         reg[X+PRT0DR],a
            and         a,[TMP2]
            mov         a,reg[X+PRT0DR]         ;clear bit
            ret

;
;Read digital input
;
;A    P0[0]..P2[7]
;
```

```
;Returns result in Z (Z=1 if bit is clear)
;
ReadDigitalInput:
            and         a,#7
            index       MASK1LUT
            mov         [TMP2],a
            M8C_SetBank0
            mov         a,reg[X+PRT0DR]
            and         a,[TMP2]
            ret





;
;Analog output section (DAC6)
;
include "dac_p02.inc"
include "dac_p03.inc"
include "dac_p04.inc"
include "dac_p05.inc"

;
;Control signal routing for DAC outputs
;
InitDAC:
            mov         [ABF_TMP],#0            ;turn off all outputs, reset latch
            loadabf

            call  DAC_P03_Off
            call  DAC_P05_Off
            call  DAC_P04_Off
            call  DAC_P02_Off
            ret

;
;Turn off DAC indicated by bit in A
;
DisableDAC:
            sub         a,#2
            clc
            rlc         a
            jacc        DisableLUT

DisableLUT:
            jmp         DAC_P02_Off
            jmp         DAC_P03_Off
            jmp         DAC_P04_Off
            jmp         DAC_P05_Off

DAC_P03_On:
            M8C_SetBank1
            or          [PRTDM1_T+0],8          ;high-Z mode for P0[3]
            and         [PRTDM0_T+0],~8
            loadddr0
            M8C_SetBank0
            mov         a,#DAC_P03_MEDPOWER
            call        DAC_P03_Start

            orabf       8                       ;enable output buffer
            ret

DAC_P03_Off:
            M8C_SetBank1
            and         [PRTDM1_T+0],~8         ;pulldown mode for P0[3]
            and         [PRTDM0_T+0],~8
            loadddr0
            M8C_SetBank0
            andabf      ~8                      ;disable output buffer

            call        DAC_P03_Stop
            ret

DAC_P05_On:
            M8C_SetBank1
            or          [PRTDM1_T+0],20h        ;high-Z mode for P0[5]
            and         [PRTDM0_T+0],~20h
            loadddr0
            M8C_SetBank0
            mov         a,#DAC_P05_MEDPOWER
            call        DAC_P05_Start

            orabf       20h                     ;enable output buffer
            ret

DAC_P05_Off:
            M8C_SetBank1
            and         [PRTDM1_T+0],~20h       ;pulldown mode for P0[5]
            and         [PRTDM0_T+0],~20h
            loadddr0
            M8C_SetBank0
            andabf      ~20h                    ;disable output buffer

            call        DAC_P05_Stop
            ret

DAC_P04_On:
            M8C_SetBank1
            or          [PRTDM1_T+0],10h        ;high-Z mode for P0[4]
            and         [PRTDM0_T+0],~10h
            loadddr0
            M8C_SetBank0
            mov         a,#DAC_P04_MEDPOWER
            call        DAC_P04_Start

            orabf       10h                     ;enable output buffer
            ret

DAC_P04_Off:
            M8C_SetBank1
            and         [PRTDM1_T+0],~10h       ;pulldown mode for P0[4]
            and         [PRTDM0_T+0],~10h
            loadddr0
            M8C_SetBank0
            andabf      ~10h                    ;disable output buffer

            call        DAC_P04_Stop
            ret

DAC_P02_On:
            M8C_SetBank1
            or          [PRTDM1_T+0],4          ;high-Z mode for P0[2]
            and         [PRTDM0_T+0],~4
            loadddr0
            M8C_SetBank0
            mov         a,#DAC_P02_MEDPOWER
            call        DAC_P02_Start

            orabf       4                       ;enable output buffer
            ret

DAC_P02_Off:
            M8C_SetBank1
            and         [PRTDM1_T+0],~4         ;pulldown mode for P0[2]
            and         [PRTDM0_T+0],~4
            loadddr0
            M8C_SetBank0
            andabf      ~4                      ;disable output buffer

            call        DAC_P02_Stop
            ret

;
;Mainline
;
_main:

            call        InitComms               ;Initialise subsystems
            call        InitDAC
            call        InitADC

            M8C_EnableGInt
            call        SettleDelay
            call        Signon                  ;Print signon message

main_loop:
            call        GetLine                 ;Get command line
            jz          main_loop               ;Skip length=0
            call        ProcessCmd              ;Process command line
            jmp         main_loop               ;Loop forever


;
;Communications section
;
include "baudclk.inc"
include "uartin.inc"
include "uartout.inc"

;
;Initialise both UARTs
;
InitComms:

            mov         [PRTDM0_T+0],#0
            mov         [PRTDM1_T+0],#0
            mov         [PRTDM0_T+1],#0
            mov         [PRTDM1_T+1],#0
            mov         [PRTDM0_T+2],#0
            mov         [PRTDM1_T+2],#0         ;initialise latch storage registers

            or          [PRTDM0_T+1],#128+32    ;P1[7],P1[5]=Strong out (TX)
            or          [PRTDM1_T+1],#64+16     ;P1[6],P1[4]=High Z     (RX)

            M8C_SetBank1
            loadddr0
            loadddr1
            loadddr2
            M8C_SetBank0                              ;load DDRs

            mov         a,#UART_PARITY_NONE
            call        UARTIN_Start
            mov         a,#UART_PARITY_NONE
            call        UARTOUT_Start
            call        BAUDCLK_Start
            ret

;
;Transmit A upstream
;
TxIN:
            tst         reg[UARTIN_TX_CONTROL_REG],UART_TX_BUFFER_EMPTY
            jz          TxIN
            call        UARTIN_SendData
            ret

;
;Transmit a newline upstream
;
NewLine:
            mov         a,#10
            call        TxIN
            mov         a,#13
            jmp         TxIN

;
;Transmit a 16-bit hex digit upstream
;
;X:A=Value
;
DumpHex16:
            push        a
            swap        a,x
            call        DumpHex
            pop         a
            jmp         DumpHex


;
```

```
;Transmit a 8-bit hex digit upstream
;
;A=Value
;
DumpHex:
            mov         [TMP2],a
            rrc         a
            rrc         a
            rrc         a
            rrc         a
            and         a,#15
            index       HEXLUT
            call        TxIN
            mov         a,[TMP2]
            and         a,#15
            index       HEXLUT
            jmp         TxIN

;
;Hex digit codes
;
HEXLUT:
            ds          "0123456789ABCDEF"

;
;Dump null terminated string pointed to by A:X
;
DumpString:
            mov         [TMP],a
            romx
            jz          DumpDone
            call        TxIN
            mov         a,[TMP]
            inc         x
            jnz         DumpString
            inc         a
            jnz         DumpString
DumpDone:
            ret

;
;Transmit A downstream
;
TxOUT:
            tst         reg[UARTOUT_TX_CONTROL_REG],UART_TX_BUFFER_EMPTY
            jz          TxOUT
            call        UARTOUT_SendData
            ret
;
;Receive A from upstream controller (blocking)
;If a byte is received from downstream controller,
;     echo it to upstream controller for daisy chaining
;
RxIN:
            call        bUARTOUT_ReadRxStatus
            mov         [RX_STATUS],a
            and         a,#UART_RX_COMPLETE
            jz          NoRxOUT
            tst         [RX_STATUS],#UART_RX_NO_ERROR
            jnz         NoRxOUT
            call        bUARTOUT_ReadRxData                ;fetch data
            cmp         a,#0
            jz          NoRxOUT                            ;screen nulls
            call        TxIN                               ;pass it up
NoRxOUT:
            call        bUARTIN_ReadRxStatus
            mov         [RX_STATUS],a
            and         a,#UART_RX_COMPLETE
            jz          RxIN
            tst         [RX_STATUS],#UART_RX_NO_ERROR
            jnz         RxIN
            call        bUARTIN_ReadRxData
            cmp         a,#0
            jz          RxIN                               ;screen nulls
            ret

;
```

```
;Print the signon message
;
Signon:
            lea         Signon_MSG
            call        DumpString
            call        NewLine
            ret

Signon_MSG:
            asciz       "Pod Online"
```

```
;
;Get a LF terminated line from the upstream controller
;Strip LF,CR and NULL terminate it
;
;Returns string length in A (not including NULL)
;
LF:         equ         10
CR:         equ         13

GetLine:

            mov         [TMP3],#COMMBUF              ;set up storage pointer
GetSyncLoop:
            call        RxIN                         ;get a char (blocking)
            cmp         a,#'!'                       ;scan for start character
            jnz         GetSyncLoop                  ;ignore everything
GetCmdLoop:
            call        RxIN                         ;get a char (blocking)
            cmp         a,#CR                        ;terminate character?
            jz          GetDone
            cmp         a,#31                        ;disregard control chars
            jc          GetCmdLoop
            mvi         [TMP3],a                     ;store char, advance ptr
            cmp         [TMP3],COMMBUF+MAXLINE-1     ;out of storage space?
            jnz         GetCmdLoop                   ;no, next char
GetDone:
            mov         a,#0
            mvi         [TMP3],a                     ;null terminate string
            sub         [TMP3],#COMMBUF+1            ;work out length
            mov         a,[TMP3]
            ret

;
;Process a command
;
ProcessCmd:
            mov         [TMP3],#COMMBUF              ;initialise pointer

            mvi         a,[TMP3]                     ;fetch address byte
            jz          ProcessDone                  ;unexepected NULL

            cmp         a,#'0'                       ;command for this pod?
            jz          ProcessMatch                 ;yes, interpret command

ProcessEcho:
            jc          ProcessDone                  ;invalid address - discard

            dec         a                            ;consume one hop
            push        a
            mov         a,#'!'                       ;command header
            call        TxOUT
            pop         a
            call        TxOUT                        ;address
EchoLoop:
            mvi         a,[TMP3]                     ;echo command downstream
            jz          EchoDone
            call        TxOUT
            cmp         [TMP3],#COMMBUF+MAXLINE
            jnz         EchoLoop
EchoDone:
            mov         a,#CR
            call        TxOUT                        ;terminate
ProcessDone:
            ret

;
;Command destination is this unit -
;process command and generate appropriate response
;
ProcessMatch:

            mvi         a,[TMP3]                     ;Fetch command
            jz          ProcessDone                  ;Unexpected NULL - discard
            cmp         a,#'W'                       ;Write DAC
            jz          ProcessAnaOut
            cmp         a,#'='                       ;Write digital output
            jz          ProcessDigOut
            cmp         a,#'?'
            jz          ProcessDigIn                 ;Read digital input
            cmp         a,#'R'
            jnz         ProcessDone                  ;Unsupported command - discard

;
;Read analog input command
;
;Command syntax !0R+-GGNNDD
;
;+          Positive Input Port 0 (0..7). Note that 8 is the ground.
;-          Negative Input Port 0 (0..7)
;GG         Gain constant
;NN         Number of samples
;DD         Sampling frequency
;
ProcessAnaIn:
            call        GetHexDigit                  ;Scan + input
            jc          CmdError
            cmp         a,#9
            jnc         CmdError
            mov         [PORT],a
            call        GetHexDigit                  ;Scan - input
            jc          CmdError
            cmp         a,#9
            jnc         CmdError
            mov         [PORT+1],a

            call        GetHexByte
            jc          CmdError
            mov         [GAINK],a
            call        GetHexByte
            jc          CmdError
            mov         [NOSAMPLES],a
            call        GetHexByte
            jc          CmdError
            mov         [SAMPLECLK],a

            cmp         [PORT+1],#8                  ;Single ended +VE input
            jz          Single
            cmp         [PORT],#8                    ;Single ended -VE input
            jz          SingleInverted
Double:
            mov         a,[PORT]
            rlc         a
            rlc         a
            rlc         a
            rlc         a
            and         a,#240
            or          a,[PORT+1]
            call        DoubleEndedPort
            jc          CmdError                     ;Not possible?
            mov         a,#'*'
            call        TxIN
            call        RunADC
            jc          CmdError
            call        DumpHex16
            call        NewLine
            ret

Single:
            mov         a,[PORT]
            cmp         a,#8
            jz          CmdError
            call        SingleEndedPort
            mov         a,#'*'
            call        TxIN
            call        RunADC
            jc          CmdError
            call        DumpHex16
            call        NewLine
            ret

SingleInverted:
            mov         a,[PORT+1]
            cmp         a,#8
            jz          CmdError
            call        SingleEndedInvPort
            mov         a,#'*'
            call        TxIN
            call        RunADC
            jc          CmdError
            call        DumpHex16
            call        NewLine
            ret

;
;Read digital input command
;
;
;Command syntax !0?PBM        P=Port 0..1 B=Bit 0..7 M=Set DDR (1=Yes 0=No)
;
ProcessDigIn:

            call        GetPortBit              ;Scan port & bit settings
            jc          CmdError

            call        GetHexDigit             ;Scan mode setting
            jc          CmdError
            cmp         a,#0                    ;Don't force DDR?
            jz          SkipDDR                 ;Yes, skip it
            mov         a,[PORT]
            call        SetDDRIn
SkipDDR:
            mov         a,#'*'
            call        TxIN
            mov         a,[PORT]
            call        ReadDigitalInput
            jnz         DigIsHigh
DigIsLow:
            mov         a,#'0'
            call        TxIN
            call        NewLine
            ret
DigIsHigh:
            mov         a,#'1'
            call        TxIN
            call        NewLine
            ret

;
;Interpret the port/bit combination
;
GetPortBit:

            call        GetHexDigit
            jc          PortBitErr
            cmp         a,#2                    ;Port 0 or 1
            jnc         PortBitErr
            rlc         a
            rlc         a
            rlc         a
            and         a,#8                    ;Shift up
            mov         [PORT],a                ;Save
            call        GetHexDigit
            jc          PortBitErr
            cmp         a,#8
            jnc         PortBitErr              ;Bit 0..7
            or          [PORT],a
            clc
            ret
PortBitErr:
            sec
            ret

;
;Drive digital output command
;
;
;Command syntax !0=PBMS      P=Port 0..1 B=Bit 0..7 M=Mode S=State
;
;
ProcessDigOut:

            call        GetPortBit
            jc          CmdError

            mov         a,[PORT]
            rrc         a
            rrc         a
            rrc         a
            and         a,#3              ;Port 0?
            jnz         SkipDACOFF        ;No, skip
            mov         a,[PORT]
            and         a,#7              ;P02..P05?
            cmp         a,#2
            jc          SkipDACOFF
            cmp         a,#6
            jnc         SkipDACOFF
            call        DisableDAC

SkipDACOFF:

            call        GetHexDigit       ;Mode
            jc          CmdError

            and         a,#3
            clc
            rlc         a
            jacc        ModeLUT

ModeLUT:
            jmp         SetPulldownMode
            jmp         SetStrongMode
            jmp         SetHighZMode
            jmp         SetPullupMode

SetHighZMode:
            mov         a,[PORT]
            call        SetDDRIn
            jmp         DoneMode

SetPulldownMode:
            mov         a,[PORT]
            call        SetDDRPulldown
            jmp         DoneMode

SetPullupMode:
            mov         a,[PORT]
            call        SetDDRPullup
            jmp         DoneMode

SetStrongMode:
            mov         a,[PORT]
            call        SetDDROut

            ;fall thru

DoneMode:
```

```
            call        GetHexDigit       ;State (Clear/Set)
            jc          CmdError
            cmp         a,#0
            jz          ClrDig
SetDig:
            mov         a,[PORT]
            call        SetDigitalOutput
            ret
ClrDig:
            mov         a,[PORT]
            call        ClrDigitalOutput
            ret

CmdError:
            mov         a,#'?'
            call        TxIN
            ret

;
;Command syntax !0WPGG       P=Port 2,3,4 or 5 GG=Gain 00..3D
;
ProcessAnaOut:
            call        GetHexDigit             ;port
            jc          CmdError
            mov         [PORT],a
            sub         a,#2
            jc          CmdError
            cmp         a,#4                    ;make sure it lies within 2..5
            jnc         CmdError
            clc
            rlc         a
            jacc        DACLUT

DACLUT:
            jmp         Ctrl_DAC_P02
            jmp         Ctrl_DAC_P03
            jmp         Ctrl_DAC_P04
            jmp         Ctrl_DAC_P05

Ctrl_DAC_P02:
            call        GetHexByte
            jc          CmdError
            cmp         a,#$3e
            jnc         CmdError
            call        DAC_P02_WriteStall
            call        DAC_P02_On
            ret

Ctrl_DAC_P03:
            call        GetHexByte
            jc          CmdError
            cmp         a,#$3e
            jnc         CmdError
            call        DAC_P03_WriteStall
            call        DAC_P03_On
            ret

Ctrl_DAC_P04:
            call        GetHexByte
            jc          CmdError
            cmp         a,#$3e
            jnc         CmdError
            call        DAC_P04_WriteStall
            call        DAC_P04_On
            ret

Ctrl_DAC_P05:
            call        GetHexByte
            jc          CmdError
            cmp         a,#$3e
            jnc         CmdError
            call        DAC_P05_WriteStall
            call        DAC_P05_On
            ret

;
;Interpret next two character as a hex byte and return in A
```

```
;
GetHexByte:
            call        GetHexDigit
            jc          GetHexDone
            rlc         a
            rlc         a
            rlc         a
            rlc         a
            and         a,#240
            mov         [TMP],a
            call        GetHexDigit
            jc          GetHexDone
            or          a,[TMP]
GetHexDone:
            ret






;
;Interpret next character as a hex digit and return in A
;
;Carry is set if digit was invalid
;
GetHexDigit:
            mvi         a,[TMP3]
            jz          InvalidDigit
            cmp         a,#'0'
            jc          InvalidDigit
            sub         a,#'0'
            cmp         a,#10
            jc          ValidDigit
            sub         a,#7
            cmp         a,#16
            jnc         InvalidDigit
ValidDigit:
            clc
            ret
InvalidDigit:
            sec
            ret

;
;Wait for pin to settle before sampling
;
SettleDelay:
            mov         x,#104                  ;10 ms
            mov         a,#0
SettleLp:
            inc         a                       ;96 us per iteration (2313 cyc)
            jnz         SettleLp
            dec         x
            jnz         SettleLp
            ret
```

11.PC TEST ENGINE SOURCE LISTING

11.1 ContestEntry201.java

```
/** @(#)ContestEntry201.java  1.0 02/28/04 Entry201

Title:                  PSOC Contest Entry 201
Description:            Windows front-end interface for automated test system
Development Tools:      Sun JDK 1.3, Sun JavaCOMM API

*/

import java.awt.*;
import java.awt.event.*;
import java.util.Hashtable;
import java.io.*;
import javax.swing.*;
import javax.swing.text.*;
import javax.swing.event.*;
import javax.swing.undo.*;

public class ContestEntry201 extends JFrame {
      private JTextPane textPane;
      private LimitedStyledDocument lsd;
      JTextArea statusLog;
      private String newline = "\n";
      private static final int MAX_CHARACTERS = 10000;
      private Hashtable actions;
      JLabel statusLabel;
      private String file,comport;
      //undo helpers
      private UndoAction undoAction;
      private RedoAction redoAction;
      private SaveAction saveAction;
      private LoadAction loadAction;
      private RunAction  runAction;
      private CommAction com1Action,com2Action,com3Action,com4Action;
      private  UndoManager undo = new UndoManager();

/**
  * Constructs a <code>ContestEntry201</code> GUI
  * using the nominated test script.
  *
  * @param   file      The name of the test script file
  */
    public ContestEntry201(String file) {
        //some initial setup
        super("PSOC Contest Entry 201. Automated Test System.");

            this.file=file;
            comport="COM1";

        //Create the document for the text area.
        lsd = new LimitedStyledDocument(MAX_CHARACTERS);

        //Create the text pane and configure it.
        textPane = new JTextPane(lsd);  //All right! No 60's jokes.
        textPane.setCaretPosition(0);
        textPane.setMargin(new Insets(5,5,5,5));
        JScrollPane scrollPane = new JScrollPane(textPane);
        scrollPane.setPreferredSize(new Dimension(400, 300));

        //Create the text area for the status log and configure it.
        statusLog = new JTextArea(5, 30);
        statusLog.setEditable(false);
        JScrollPane scrollPaneForLog = new JScrollPane(statusLog);

        //Create a split pane for the change log and the text area.
        JSplitPane splitPane = new JSplitPane(
                                       JSplitPane.VERTICAL_SPLIT,
                                       scrollPane, scrollPaneForLog);
        splitPane.setOneTouchExpandable(true);

        //Create the status area.
        JPanel statusPane = new JPanel(new GridLayout(1, 1));
        statusLabel =
                new JLabel("Idle");
        statusPane.add(statusLabel);

        //Add the components to the frame.
        JPanel contentPane = new JPanel(new BorderLayout());
        contentPane.add(splitPane, BorderLayout.CENTER);
        contentPane.add(statusPane, BorderLayout.SOUTH);
        setContentPane(contentPane);

        //Set up the menu bar.
        createActionTable(textPane);
        JMenu fileMenu = createFileMenu();
        JMenu editMenu = createEditMenu();
        JMenu toolsMenu = createToolsMenu();
        JMenu configMenu = createConfigMenu();
        JMenuBar mb = new JMenuBar();
        mb.add(fileMenu);
        mb.add(editMenu);
        mb.add(toolsMenu);
        mb.add(configMenu);
        setJMenuBar(mb);

        // Load the test file into memory
        loadDocument();

        //Start watching for undoable edits and caret changes.
        lsd.addUndoableEditListener(new MyUndoableEditListener());

    }

    /* Listen for edits that can be undone. */
    private class MyUndoableEditListener
                    implements UndoableEditListener {
        public void undoableEditHappened(UndoableEditEvent e) {
            //Remember the edit and update the menus.
            undo.addEdit(e.getEdit());
            undoAction.updateUndoState();
            redoAction.updateRedoState();
        }
    }

    /* Create the edit menu. */
    private JMenu createEditMenu() {
        JMenu menu = new JMenu("Edit");

        undoAction = new UndoAction();
        menu.add(undoAction);

        redoAction = new RedoAction();
        menu.add(redoAction);

        menu.addSeparator();

        //These actions come from the default editor kit.
        menu.add(getActionByName(DefaultEditorKit.cutAction));
        menu.add(getActionByName(DefaultEditorKit.copyAction));
        menu.add(getActionByName(DefaultEditorKit.pasteAction));

        menu.addSeparator();

        menu.add(getActionByName(DefaultEditorKit.selectAllAction));
        return menu;
    }

    //Create the file menu.
    private JMenu createFileMenu() {
        JMenu menu = new JMenu("File");

        loadAction = new LoadAction();
        menu.add(loadAction);
```

```
        menu.addSeparator();

        saveAction = new SaveAction();
        menu.add(saveAction);

        return menu;
    }

    //Create the tools menu.
    private JMenu createToolsMenu() {
        JMenu menu = new JMenu("Tools");

        runAction = new RunAction();
        menu.add(runAction);

        return menu;
    }

    //Create the config menu.
    private JMenu createConfigMenu() {
        JMenu menu = new JMenu("Config");

        com1Action = new CommAction("COM1");
        menu.add(com1Action);
        com2Action = new CommAction("COM2");
        menu.add(com2Action);
        com3Action = new CommAction("COM3");
        menu.add(com3Action);
        com4Action = new CommAction("COM4");
        menu.add(com4Action);

        highlight();

        return menu;
    }

      //
      // Highlight the active com port
      //
      private void highlight()
      {
            if(comport.equals("COM1"))
                  com1Action.putValue(AbstractAction.NAME,"COM1*");
            else
                  com1Action.putValue(AbstractAction.NAME,"COM1");

            if(comport.equals("COM2"))
                  com2Action.putValue(AbstractAction.NAME,"COM2*");
            else
                  com2Action.putValue(AbstractAction.NAME,"COM2");

            if(comport.equals("COM3"))
                  com3Action.putValue(AbstractAction.NAME,"COM3*");
            else
                  com3Action.putValue(AbstractAction.NAME,"COM3");

            if(comport.equals("COM4"))
                  com4Action.putValue(AbstractAction.NAME,"COM4*");
            else
                  com4Action.putValue(AbstractAction.NAME,"COM4");

      }

    //
    // Load the script into memory
    //
    private void loadDocument() {
            FileInputStream f;
            AttributeSet attr=new SimpleAttributeSet();
            StringBuffer str=new StringBuffer(128);
            int i,x;

            try {
                  f = new FileInputStream(file);

                  try {
```

```
                  // erase document
                  lsd.clear();
            } catch(BadLocationException ble) {
                        statusLabel.setText("Couldn't erase document");
                        try { f.close(); } catch(IOException ioe) { }
                        return;
                  }

                  x=0;
                  while(x >= 0)
                  {

                  // erase string contents
                  if(str.length() > 0)
                        str.delete(0,str.length());

                  // scan a line in to EOL
                  for(i=0;i<128;i++)
                  {
                        try {
                              x=f.read();
                        } catch(IOException ioe) {
                              statusLabel.setText("I/O error on read");
                              break;
                        }
                        if(x < 0 || x == 10)
                              break;
                        if(x > 31)
                              str.append((char)x);

                  }

                  try {

                        lsd.insertString(lsd.getLength(),str.toString()+newline,attr);
                        } catch(BadLocationException ble) {
                              statusLabel.setText("Couldn't insert text
"+str.toString()+"'");
                              break;
                        }
                  } // while
                  try {
                        f.close();
                  } catch(IOException ioe)
                  {

                  }
                  if(x < 0)
                        statusLabel.setText("Loaded "+file);


            } catch(FileNotFoundException fnfe) {
                  statusLabel.setText("Couldn't load "+file);

            }
    }

    //The following two methods allow us to find an
    //action provided by the editor kit by its name.
    private void createActionTable(JTextComponent textComponent) {
        actions = new Hashtable();
        Action[] actionsArray = textComponent.getActions();
        for (int i = 0; i < actionsArray.length; i++) {
            Action a = actionsArray[i];
            actions.put(a.getValue(Action.NAME), a);
        }
    }

    private Action getActionByName(String name) {
        return (Action)(actions.get(name));
    }


    class UndoAction extends AbstractAction {
        public UndoAction() {
            super("Undo");
            setEnabled(false);
        }

        public void actionPerformed(ActionEvent e) {
            try {
                undo.undo();
            } catch (CannotUndoException ex) {
                statusLabel.setText("Unable to undo: " + ex);
                ex.printStackTrace();
            }
            updateUndoState();
            redoAction.updateRedoState();
        }

        protected void updateUndoState() {
            if (undo.canUndo()) {
                setEnabled(true);
                putValue(Action.NAME, undo.getUndoPresentationName());
            } else {
                setEnabled(false);
                putValue(Action.NAME, "Undo");
            }
        }
    }

    class RedoAction extends AbstractAction {
        public RedoAction() {
            super("Redo");
            setEnabled(false);
        }

        public void actionPerformed(ActionEvent e) {
            try {
                undo.redo();
            } catch (CannotRedoException ex) {
                statusLabel.setText("Unable to redo: " + ex);
                ex.printStackTrace();
            }
            updateRedoState();
            undoAction.updateUndoState();
        }

        protected void updateRedoState() {
            if (undo.canRedo()) {
                setEnabled(true);
                putValue(Action.NAME, undo.getRedoPresentationName());
            } else {
                setEnabled(false);
                putValue(Action.NAME, "Redo");
            }
        }
    }

      //
      // Save script to disk
      //
      private void saveDocument()
      {
            int i;
                  FileOutputStream f;
                  String str ;

              try {
               str = lsd.getText(0,lsd.getLength());
              } catch(BadLocationException ble) {
                  statusLabel.setText("Unable to access document");
                  return;
              }

              try {
                  f = new FileOutputStream(file);
              } catch(IOException ioe) {
                  statusLabel.setText("Unable to open "+file+" for write");
                  return;
              }


               for(i=0;i<str.length();i++)
```

```
             {
                try {
                f.write(str.charAt(i));
                } catch(IOException ioe) {
                      statusLabel.setText("Unable to write "+file);
                      break;
                }
             }

            try {
                f.close();
                statusLabel.setText("Saved "+file);

                } catch(IOException ioe) {
                      statusLabel.setText("Unable to close "+file);
                }


           }

                //
                // Handler for save menu item
                //
                class SaveAction extends AbstractAction {

                      public SaveAction() {
                      super("Save");
                      setEnabled(true);
                  }

                  public void actionPerformed(ActionEvent e) {

                            saveDocument();

                  }

              }

    //
    // Handler for load menu item
    //
    class LoadAction extends AbstractAction {

          public LoadAction() {
          super("Load");
          setEnabled(true);
      }

      public void actionPerformed(ActionEvent e) {

                loadDocument();

      }

   }

    //
    // Handler for comm port menu item
    //
    class CommAction extends AbstractAction {

          public CommAction(String port) {
          super(port);
          setEnabled(true);
      }

      public void actionPerformed(ActionEvent e) {
                comport=e.getActionCommand();
                statusLabel.setText("Comport "+comport);
                highlight();
      }

   }


/**
  *   The standard main method.
  *
  *   @param      args  Arguments [script name]
 **/
    public static void main(String[] args) {
      String name;
      if(args.length < 1)
            name="ate.scr";
      else
            name=args[0];

      final ContestEntry201 frame = new ContestEntry201(name);

      frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
            public void windowActivated(WindowEvent e) {
                frame.textPane.requestFocus();
            }
        });

        frame.pack();
        frame.setVisible(true);
    }

      //
      // Handler for 'run' menu item. This executes the test script.
      //
      class RunAction extends AbstractAction {

            public RunAction() {
            super("Run");

            setEnabled(true);
            }

        public void actionPerformed(ActionEvent e) {

                  try {
                              AutomatedTestEngine ate = new

                              AutomatedTestEngine(lsd,statusLabel,statusLog,comport);

                              if(false==ate.run())
                                    statusLabel.setText("Tests incomplete");
                              else
                                    statusLabel.setText("Tests complete");


                  } catch(Exception exc) {
                              statusLog.append(exc.getMessage()+newline);
                  }


        }
    }

}
```

11.2 AutomatedTestEngine.java

```
/** @(#)AutomatedTestEngine.java   1.0 02/18/05 Entry201

Title:                  PSOC Contest Entry 201
Description:            Automated test interpreter and runtime engine
Development Tools:      Sun JDK 1.3, Sun JavaCOMM API

*/

import javax.swing.*;
import javax.swing.text.*;

public class AutomatedTestEngine  extends Object {

      private LimitedStyledDocument lsd;
      private JLabel statusLabel;
      private JTextArea statusLog;
      private String newline = "\n";
      private boolean abort;
      private int rate;
      private int line;
      private int trace;
      private int order_index;
      private int orderp[];
      private int ordern[];
      private int element;
      private int clock=0;
      private boolean used_clock;
      private int cycle=0;
      private int no_tests;
      private int
test_portp[],test_portn[],test_type[],test_state[],test_upper[],test_lower[],test_gain
k[];
      final int RPT_HEAPSIZE=100;
      final int VAR_HEAPSIZE=500;
      final int MAX_VECTOR=24;
      String rpt_heap[];
      String var_heap[];
      int var_value[];
      int rpt_idx,rpt_cnt,var_idx;
      TestBus bus;
      boolean open;

/**
**    Constructs an <code>AutomatedTestEngine</code> object
**
**  @param lsd                The document containing the script
**  @param statusLabel        The label at the bottom of the window
**                            used for status information
**  @param statusLog          The text window for displaying test results
**                            and diagnostics
**  @param comport            The name of the comport that the test ports
**                            are connected to
**  @exception                If the comport could not be opened/configured
*/
      public AutomatedTestEngine(LimitedStyledDocument lsd,
                                         JLabel statusLabel,
                                         JTextArea statusLog,
                                         String comport)
                  throws TestBusException
      {
                  int i;

                  open=false;
                  this.statusLabel = statusLabel;
                  this.statusLog = statusLog;
                  this.lsd = lsd;

                  rate=100;

                  trace=2;
```

```
            rpt_cnt=0;
            rpt_heap = new String[RPT_HEAPSIZE];
            var_heap = new String[VAR_HEAPSIZE];
            var_value = new int[VAR_HEAPSIZE];
            var_idx=0;
            cycle=0;
            orderp = new int[MAX_VECTOR];
            ordern = new int[MAX_VECTOR];
            // free array
            for(i=0;i<VAR_HEAPSIZE;i++)
                  var_value[i]=-1;

            test_portp = new int[MAX_VECTOR];
            test_portn = new int[MAX_VECTOR];
            test_type = new int[MAX_VECTOR];
            test_state = new int[MAX_VECTOR];
            test_upper = new int[MAX_VECTOR];
            test_lower = new int[MAX_VECTOR];
            test_gaink = new int[MAX_VECTOR];

            bus = new TestBus(comport);

            open=true;
}

//    Set the trace level 0..3
private void set_trace(int n)
      throws NumberFormatException
{
      if(trace < 0 || trace > 3)
         throw new NumberFormatException("Trace level must be between 0 and 3");

      trace=n;
}

// start a new test vector specification
private void init_order()
{
      order_index=0;
}

//
// process a test digital input request [queued for later processing]
//
private void test_digital_input(int portp,int portn,int state)
{
      test_portp[no_tests]=portp;
      test_portn[no_tests]=portn;
      test_type[no_tests]=0;
      test_state[no_tests]=state;

      no_tests++;
}

//
// process a test analog input request [queued for later processing]
//
private void test_analog_input(int portp, int portn, double fvalue_lower,
                  double fvalue_upper, double gain)
{
      // allowable gains
      final double gains[]={16.0,8.0,5.333,4.0,3.2,2.667,2.286,2.0,1.778,
                              1.6,1.455,1.333,1.231,1.143,1.067,1.0,

                              0.0625,0.125,0.1875,0.25,0.3125,0.375,0.4375,0.5,
                              0.5625,0.625,0.6875,0.75,0.8125,0.875,0.9375,1.0};

      double diff,d;
      int lower,upper,gaink,i,idx;

      //
      // only supports ADC routing on port 0
      //
      if(portp >= 0 && ((portp>>3)&3)>0)
            err(10,value2name(portp));
      else if(portn >= 0 && ((portn>>3)&3)>0)
            err(10,value2name(portn));
```

```
else
{
 // are we just reading the input pin or conducting a test
 if(fvalue_lower >= 0 && fvalue_upper >= 0)
 {
 // test.. is it a double-ended input? (using the instrumentation amp)
      if(portp >= 0 && portn >= 0)
      {
            // yes, need to compensate for a x2 gain
            fvalue_lower *= 2.0;
            fvalue_upper *= 2.0;

            if(trace > 2)
                  echo(" GAIN*2 ");
      }


// has gain been specified for the vector element?
if(gain >=0.0)
{
 // yes, find the closest supported gain to the requested gain
 idx=-1;
 diff=32.0;
 for(i=0;i<32;i++)
 {
      d=Math.abs(gains[i]-gain);
      if(d<diff)
      {
            diff=d;
            idx=i;
      }
 }
}
else
{
//
// gain not specified.. pick gain the gives the lowest quantisation error
// with both limits in range
//
      idx=-1;
      diff=0.0;
      gain=1.0;
      for(i=0;i<32;i++)
      {
            lower=(int)(2047.0*fvalue_lower*gains[i]);
            upper=(int)(2047.0*fvalue_upper*gains[i]);


            if(Math.abs(upper) > 2047 || Math.abs(lower) > 2047 )
                  continue;

            d=upper-lower;

            if(d >= diff)
            {
                  diff = d;
                  idx=i;
            }
      }

}

 // map to PSOC hardware gain constant
gaink=(idx & 15) << 4;
if((idx & 16)==0)
      gaink|=8;

 //
 // quantise thresholds to 12-bit levels
 //
 fvalue_lower *= gains[idx]/gain;
 fvalue_upper *= gains[idx]/gain;
 lower=(int)(2047.0*fvalue_lower);
 upper=(int)(2047.0*fvalue_upper);

 //
 // queue test until after outputs have been driven
```

```
       //
      test_portp[no_tests]=portp;
      test_portn[no_tests]=portn;
      test_type[no_tests]=1;
      test_lower[no_tests]=lower;
      test_upper[no_tests]=upper;
      test_gaink[no_tests]=gaink;

      no_tests++;

      if(trace > 2)
            echo("TEST ANALOG GAIN="+gains[idx]+"/"+gaink+
            " "+fvalue_lower+"/"+lower+"-"+fvalue_upper+"/"+upper+
            " PORT "+value2name(portp,portn)+newline);

      }
      else
      {
            // no, is it a double-ended input? (using the instrumentation amp)
            if(portp >= 0 && portn >= 0)
                  // yes, need to compensate for a x2 gain
                  gain=0.5;
            else
                  gain=1.0;

            // map gain to a constant
            for(idx=0;idx<32;idx++)
            {
                  if(gains[idx]==gain)
                        break;
            }

            // map to hardware constant
            gaink=(idx & 15) << 4;
            if((idx & 16)==0)
                  gaink|=8;

            //
            // queue read until after outputs have been driven
            //
            test_portp[no_tests]=portp;
            test_portn[no_tests]=portn;
            test_type[no_tests]=2;
            test_gaink[no_tests]=gaink;

            no_tests++;

            if(trace > 2)
                  echo("READ ANALOG GAIN="+gains[idx]+"/"+gaink+
                        " PORT "+value2name(portp,portn)+newline);

            }
      }


}

//
// start a new test vector
//
private void start_vector()
{
      element=0;
      used_clock=false;
      no_tests=0;

      cycle++;
}

//
// test a port for high impedance
//
private boolean test_high_impedance(int port)
      throws TestBusException
{
      boolean ok;
```

```
        ok=true;

        // drive pin resistive pulldown, low (pull down using 5.6K resistor)
        bus.drive_digital_output(port, 0, 0);
        // wait for pin to settle
        rate_delay();
        if(0!=bus.read_digital_input(port,false))
              ok=false;
        // drive pin resistive pullup, high (pull up using 5.6K resistor)
        bus.drive_digital_output(port, 1, 3);
        // wait for pin to settle
        rate_delay();
        if(1!=bus.read_digital_input(port,false))
              ok=false;
        // reset pin to high impedance
        bus.drive_digital_output(port, 0, 2);

        return ok;
  }

  //
  // terminate test vector and execute any queued tests
  //
  private void end_vector()
  {
        int i,p,n,v;
        boolean ok;

        if(element > 0)
        {
              if(trace > 2)
                    echo(newline+"VECTOR END"+newline);
              else if(trace > 1)
                    echo(newline);

              // implement settle delay here
              rate_delay();

              // run tests

              for(i=0;i<no_tests;i++)
              {
              switch(test_type[i]) {
                    case 0:

                    if(trace >= 1)
                          echo(value2name(test_portp[i],test_portn[i])+"=");

                    //
                    // processing for logic high, logic low, read state
                    //
                    if(test_state[i] < 2)
                    {
                    try {
                     p=n=0;
                     if(test_portp[i] >= 0)
                     {
                          p=bus.read_digital_input(test_portp[i],true);
                          if(trace >= 1 || test_state[i] < 0)
                                echo(""+p);
                     }
                     if(test_portn[i] >= 0)
                     {
                          n=bus.read_digital_input(test_portn[i],true);

                          if(trace >= 1 || test_state[i] < 0)
                                echo(""+n);
                     }

                     if(trace >= 1 || test_state[i] < 0)
                          echo(" ");

                     if(test_state[i] >= 0)
                     {
                          ok=true;
                          if(test_portp[i] >= 0 && p!=test_state[i])
                                ok=false;
```

```
            if(test_portn[i] >= 0 && n!=(test_state[i]^1))
                  ok=false;
            if(!ok)
            {
                  if(trace < 1)

                        echo("FAIL:"+value2name(test_portp[i],
                              test_portn[i]));
                  else
                        echo("(FAIL)");

                  echo(" ");
                  abort=true;
            }
       }

       } catch(TestBusException tbe) {
            echo("TestBus failed on read_digital_input "
                        +tbe.getMessage()+"\n");
            abort=true;
            i=no_tests;
       }
      }
      else // processing for high impedance test
      {
            if(trace >= 1)
                  echo("HiZ ");

            try {
                        if(test_portp[i] >= 0)
                        {

                              if(false==test_high_impedance(
                                          test_portp[i]))
                                    abort=true;
                        }
                        if(test_portn[i] >= 0 && !abort)
                        {
                              if(false==test_high_impedance(
                                          test_portn[i]))
                                    abort=true;
                        }
                        if(abort && trace >= 1)
                              echo("(FAIL) ");

                  } catch(TestBusException tbe) {
                        echo("TestBus failed on test_high_impedance "
                              +tbe.getMessage()+"\n");
                        abort=true;
                        i=no_tests;
                  }


      }

break;

      case 1:
      case 2:
       if(trace >= 1 || test_type[i]==2)
            echo(value2name(test_portp[i],test_portn[i])+"=");

      try {
            v=bus.read_analog_input(test_portp[i],
                        test_portn[i],test_gaink[i]);

            if(trace >= 1 || test_type[i]==2)
                  echo(new Double(v/2048.0).toString()+" ");

            if(test_type[i]==1)
            {
                  if(v < test_lower[i] || v > test_upper[i])
                  {
                        if(trace >= 1)
                              echo("(FAIL)");
                        else
```

```
                                          echo("FAIL:"+value2name(test_portp[i],
                                          test_portn[i]));
                                    echo(" ");
                                    abort=true;
                              }
                        }
                  } catch(TestBusException tbe) {
                              echo("TestBus failure on read_analog_input "+
                                    tbe.getMessage()+"\n");
                              abort=true;
                              i=no_tests;
                  }

            break;

            }

            }

            if(used_clock)
                  clock ^= 1;
      }
}

//
// set the test execution rate in milliseconds
//
private void set_rate(int n)
      throws NumberFormatException
{
      if(rate < 1)
            throw new NumberFormatException("Rate must be greater than 1");

      rate=n;
}

// delay the current thread by rate ms
private void rate_delay()
{
      try
      { // Snoozing a bit.
          Thread.sleep(rate);
      }
      catch (Exception e)
      {
          statusLabel.setText("Couldn't sleep! "+e.getMessage());
      }
}

// return ascii representation of single-ended port name
private String value2name(int value)
{
      if(value >= 0)
            return ""+(char)('P'+((value>>5)&7))+
                  (char)(((value >> 3)&3)+'0')+(char)((value & 7)+'0');
      else
            return "AGND";
}

// return ascii representation of double-ended port name
private String value2name(int valuep,int valuen)
{
      if(valuep >= 0)
      {
            if(valuen >= 0)
                  return value2name(valuep)+"-"+value2name(valuen);
            else
                  return value2name(valuep);
      }
      else
            return value2name(valuen);
}

// insert a test vector element definition
private void add_order(int valuep,int valuen)
{
      int i;
```

```
        // scan for duplicates
        for(i=0;i<order_index;i++)
        {
            if(valuep >= 0)
            {
                if(orderp[i]==valuep || ordern[i]==valuep)
                {
                        err(5,value2name(valuep));
                    break;
                }
            }
            if(valuen >= 0)
            {
                if(orderp[i]==valuen || ordern[i]==valuen)
                {
                        err(5,value2name(valuen));
                    break;
                }
            }
        }

        if(order_index >= MAX_VECTOR)
            err(6,value2name(valuep,valuen));
        else
        {
            orderp[order_index]=valuep;
            ordern[order_index++]=valuen;
        }
    }

    //
    // evaluate test vector element
    //
    private void eval_vector_element(String p)
    {
        int portp,portn,mode;
        int s,s2,s3,len;
        double val1,val2,gain;

        if(element == 0)
        {
            if(trace > 2)
                echo("VECTOR START"+newline);
        }

        try {

        if(element >= order_index)
            err(6,p);
        else
        {
            portp=orderp[element];
            portn=ordern[element];

            if(trace > 2)
            echo("["+value2name(portp,portn)+"]="+p+" ");
            else if(trace > 1)
                echo(p+" ");

            len=p.length();

            //
            // extract resitive pullup / pulldown modifiers
            //
            mode=1;             // Strong/CMOS by default

            if(len == 2)
            {
                switch(Character.toLowerCase(p.charAt(1))) {
                    case 'u':
                        len--;
                        mode=3;
                        break;
                    case 'd':
                        len--;
                        mode=0;
                        break;
                }

            }

        //
        // parse test vector element operation
        //
        if(len==1)
        {


            switch(Character.toLowerCase(p.charAt(0))) {
                default:
                err(4,p);
                break;
            case '0':
            case '1':
                bus.drive_digital_output(portp, portn, p.charAt(0)-'0', mode);
                break;
            case 'c':
                bus.drive_digital_output(portp,portn,clock,mode);
                used_clock=true;
                break;
            case 'k':
                bus.drive_digital_output(portp,portn,clock ^ 1,mode);
                used_clock=true;
                break;
            case 'r':
                bus.drive_digital_output(portp,portn,
                        Math.random() > 0.5 ? 1:0,mode );
                break;
            case 'x':
                bus.drive_digital_output(portp,portn,0,2);
                break;
            case 'l':
                test_digital_input(portp,portn, 0);
                break;
            case 'h':
                test_digital_input(portp,portn, 1);
                break;
            case 'z':
                test_digital_input(portp,portn, 2);
                break;
            case '*':
                if(portp < 0 || ((portp>>3)&3)==0)
                {
                    if(portn < 0 || ((portn>>3)&3)==0)
                        test_analog_input(portp,portn,-1,-1,-1);
                    else
                        test_digital_input(portp,portn,-1);
                }
                else
                    test_digital_input(portp,portn,-1);
                break;
            } // switch
        } // strlen
        else // is an analog element
        {
            s=0;
            // scan for test, or driven value
            while(s < p.length())
            {
                if(p.charAt(s)=='-' || p.charAt(s)==',')
                    break;
                s++;
            }

            if(s >= p.length())
            {
                // driven DAC value
                    try {
                        val1 = Double.parseDouble(p);

                        if(val1 < 0.0)
                            val1=0.0;
                        if(val1 > 1.0)
```

```
                        val1=1.0;

                  if(trace > 2)
                      echo("DRIVE ANALOG "+val1+
                              " on PORT "+value2name(portp,portn)+
                              " ");

                  bus.drive_analog_output(portp,portn, val1);

            } catch(NumberFormatException nfe) {
                  err(4,p);
            }
}
else // - or , encountered
{
      gain=-1.0;
      s3=0;

      // extract gain parameter if found
      if(p.charAt(s)==',')
      {
            s2=s+1;
            while(s2 < p.length())
            {
                  if(p.charAt(s2)=='-')
                  break;
             s2++;
            }
            if(p.charAt(s2)!='-')
                  err(4,p);
            else
            {
                  try {
                        gain=Double.parseDouble(
                              p.substring(0,s-1));

                        s3=s+1;
                        s=s2;
                  } catch(NumberFormatException nfe) {
                         err(4,p.substring(0,s-1));
                  }
            }
      }
      else
            {
                  // leading negative
                  if(s==0)
                  {
                        s++;

                        while(s < p.length())
                        {
                              if(p.charAt(s)=='-')
                              break;
                              s++;
                        }
                        if(p.charAt(s)!='-')
                              err(4,p);

                  }
            }

            try {
                  val1=Double.parseDouble(p.substring(s3,s));

                  if(val1 < -1.0) val1=-1.0;
                  if(val1 > 1.0) val1=1.0;

                  val2=Double.parseDouble(p.substring(s+1));

                  if(val2 < -1.0) val2=-1.0;
                  if(val2 > 1.0) val2=1.0;

                  test_analog_input(portp,portn,val1,val2,gain);

                  } catch(NumberFormatException nfe) {
                        err(4,p);
                        }

                }
        } // strlen(p) == 1

        element++;

        } // element < order_index

        } catch(TestBusException tbe) {
              statusLog.append("TestBus Error "+tbe.getMessage()+"\n");
              abort=true;
        }
}

//
// string printer
//
private void echo(String str)
{
        statusLog.append(str);
}


//
// interpreter error reporter
//
private void err(int no,String arg)
{
        final String errs[] = { "Invalid Command",                   // 0
                                "Internal",                          // 1
                          "Unknown Variable",                        // 2
                          "Out of Variable Heap Space",              // 3
                          "Syntax Error",                            // 4
                          "Duplicated Test Vector Element",          // 5
                          "Too Many Test Vector Elements",           // 6
                          "Out of Repeat Heap Space",                // 7
                          "Too Many Nested Repeats",                 // 8
                          "Analog Output Not Supported",             // 9
                          "Analog Input Not Supported"};             // 10


        echo("ERROR "+no+" "+errs[no]);

        if(arg != "")
              echo(" "+arg);

        echo(newline);
        abort=true;
}

//
// scan for reserved names p00..07, p10..17, p20..27, q,r,s,t,u,v
//
// returns -1 if not reserved, otherwise index number 0..31
//
private int is_reserved(String name)
{
        int port,bit,pod;

        if(name.length()==3)
        {
              pod = Character.toLowerCase(name.charAt(0)) - 'p';

              if(pod >= 0 && pod < 8)
              {
                    port = name.charAt(1)-'0';
                    if(port >= 0 && port < 4)
                    {

                          bit = name.charAt(2)-'0';
                          if(bit >= 0 && bit < 8)
                          {

                                return (pod << 5)+(port<<3)+bit;
                          }
                    }
```

```
            }
      }
      return -1;
}

//
// scan for a variable, return value or -1 if not found
//
private int is_var(String name)
{
      int i;

      for(i=0;i<var_idx;i++)
      {
            if(var_value[i] >= 0)
            {
             if(var_heap[i].equalsIgnoreCase(name))
                  return var_value[i];
            }
      }

      return -1;
}

//
// add a variable to the heap, replacing duplicates
//
private void add_var(String name, int value)
{
      int i;

      if(trace > 2)
            echo("ADD_VAR "+name+" = "+value2name(value)+newline);

      // remove duplicates
      for(i=0;i<var_idx;i++)
      {
            if(var_value[i] >= 0)
            {
                  if(var_heap[i].equalsIgnoreCase(name))
                  {
                        var_value[i]=value;
                        return;
                  }
            }
      }

      if(var_idx < VAR_HEAPSIZE)
      {
       var_heap[var_idx] = new String(name);
       var_value[var_idx++] = value;
      }
      else
            err(3, name);

}

//
// map variable/port name to port value
//
// return -1 if not found
//
private int resolve_var(String name)
{
      int x=is_reserved(name);

      if(x >= 0)
            return x;

      x=is_var(name);

      if(x >= 0)
            return x;

      err(2,name);

      return -1;
```

```
}

//
// evaluate a variable=value assignment
//
private void eval_var(String expr)
{
      int p;
      int value;

      p=0;
      while(p < expr.length())
      {
            if(expr.charAt(p)=='=')
                  break;
            p++;
      }
      if(p >= expr.length())
            err(4,expr);
      else
      {
            value=resolve_var(expr.substring(p+1));
            if(value >= 0)
            {

                  add_var(expr.substring(0,p), value);
            }
      }
}

//
// process an incoming line with respect to a ‘repeat’ block
//
private void process_repeat(String line, int rpt_req)
{
      if(rpt_req != 0)
      {
            // repeat already active (error)
            if(rpt_cnt != 0)
                  err(8, "");

            rpt_idx = 0;
            rpt_cnt = rpt_req;
      }
      else
      {
      // line capture active?
      if(rpt_cnt != 0)
      {
                  if(rpt_idx  >= RPT_HEAPSIZE)
                        err(7,line);
                  else
                  {
                        rpt_heap[rpt_idx++] = new String(line);
                  }
      }
      }
}

//
// execute a repeat block
//
private void end_repeat()
{
      int idx,cnt=rpt_cnt;

      rpt_cnt=0;  // cancel repeat line capture

      while(--cnt > 0)
      {
            idx=0;

            while(idx < rpt_idx)
            {
                  process(rpt_heap[idx++]);

            }
```

```
         }
   }


   //
   // script line parser.. this is the top level of the interpreter
   //
   // apologies for any ravoli code
   //
   private void process(String line)
   {
         int p,pe;
         int i,j,state=0;                    // parser state
         // directives
         final String cmds[] =
         { "exit", "msg", "order", "var", "rate", "repeat",
           "endr", "timeout", "trace", "samples", "freq",
           "-" };
         boolean msg_flag=false;
         int rpt_req=0;
         boolean eol;

         p=0;

         start_vector();

         // strip leading spaces and comments
         while(p < line.length())
         {
               if(line.charAt(p)==';')
                     return;
               if(line.charAt(p)!=' ')
                     break;
               p++;
         }
         if(p >= line.length())
               return;

         if(trace > 2)
                echo("--> "+line.substring(p)+" <--"+newline);

// process to eol or space
eol=false;
do
{
 pe=p;
 while(pe < line.length())
 {
   if(line.charAt(pe)==' ' || line.charAt(pe)==';')
         break;
   pe++;
 }

 if(p != pe)
 {
   if(pe >= line.length())
         eol=true;

   // process sub arguments
   //    echo("["+line.substring(p,pe)+"]"+newline);


  switch(state) {
   case 0:           // top level scan
      if(line.charAt(p)=='$') // command
      {
         i=0;
         while(!cmds[i].equalsIgnoreCase("-"))
         {
            if(cmds[i].equalsIgnoreCase(line.substring(p+1,pe)))
            {
             switch(i) {
               case 0: // exit
                     abort=true;
                     break;
               case 1: // msg
```

```
                  msg_flag=true;
               state=1;
               break;
            case 2: // order
                  init_order();
               state=2;
               break;
            case 6: // endr
                  end_repeat();
               return;
            case 3: // var
            case 4: // rate
            case 5: // repeat
            case 7: // timeout
            case 8: // trace
            case 9: // samples
            case 10: // freq
               state=i;
               break;
            default:
                  err(1,line.substring(p));
               break;
          }
          break;
         }
         i++;
      }
      if(cmds[i].equalsIgnoreCase("-"))
            err(0,line.substring(p,pe));
   }
   else // test vector
   {
      eval_vector_element(line.substring(p,pe));
   }
   break;
case 1:    // msg
       echo(line.substring(p,pe)+" ");
   break;
case 2: // order
      j=line.substring(p,pe).indexOf('-');      // scan for a double-ended pair
                                                     or inverted single-ended
      if(j >= 0)
      {
            if(j==0)
            {
                  // single-ended inverted specification
                  i=resolve_var(line.substring(p+1,pe));
                  if(i >= 0)
                        add_order(-1,i);
            }
            else
            {
                  i=resolve_var(line.substring(p,p+j));
                  j=resolve_var(line.substring(p+j+1,pe));
                  if(i >= 0 && j >= 0)
                        add_order(i,j);
            }
      }
      else
      {
       // single ended order specification
       i=resolve_var(line.substring(p,pe));
       if(i >= 0)
            add_order(i,-1);
      }
      break;
case 3: // var
      eval_var(line.substring(p,pe));
      break;
case 4: // rate
   set_rate(Integer.parseInt(line.substring(p,pe)));
   state=0;
   break;
case 5: // repeat
      i=Integer.parseInt(line.substring(p,pe));
   if(i>=0)
      rpt_req=i;
```

```
         state=0;
            break;
      case 7: // timeout
            i=Integer.parseInt(line.substring(p,pe));
         if(i>=0)
            bus.set_timeout(i);
         state=0;
         break;
      case 8: // trace
         set_trace(Integer.parseInt(line.substring(p,pe)));
         state=0;
         break;
      case 9: // samples
         bus.set_samples(Integer.parseInt(line.substring(p,pe)));
         state=0;
         break;
      case 10: // freq
         bus.set_freq(Integer.parseInt(line.substring(p,pe)));
         state=0;
         break;
      default:
       err(4,line.substring(p));
       break;
     }


     if(!eol)
      p=pe+1;
     else
      p=pe;

    }
   } while(pe != p);

   if(msg_flag)
      echo(newline);

   end_vector();

   if(!abort)
      process_repeat(line, rpt_req);


}


/**
** Execute the test script
** @return boolean indicating whether the test script was run to completion
*/
public boolean run()
{
            int i,j;

            j=0;
            line=1;

            echo(newline);

            try {
             for(i=0;i<lsd.getLength();i++)
             {
                  if(lsd.getText(i,1).equals(newline))
                  {
                        if(j != i)
                        {
                         process(lsd.getText(j,i-j));
                         if(abort)
                         {
                              // position the cursor at the start of the line

                              break;
                         }
                        }

                        line++;
                        j=i+1;
                        }
                 }
                } catch(BadLocationException ble) {
                        statusLabel.setText("Unable to access document at line "+line);
                }

                // perform cleanup operations
                if(open==true)
                        bus.clean();

                open=false;

                return !abort;
        }

}
```

11.3 TestBus.java

```
/* @(#)TestBus.java     1.0 02/28/04 Entry201

Title:                  PSOC Contest Entry 201
Description:            Automated test system serial bus interface object
Development Tools:      Sun JDK 1.3, Sun JavaCOMM API

*/

import javax.comm.*;          // communcation's api plug-in from sun
import java.io.*;
import javax.swing.*;

public class TestBus extends Object {

      private OutputStream streamo;
      private InputStream streami;
      private SerialPort sPort;
      private int timeout,samples,freq,dividek;
      private CommPortIdentifier portId;
      private boolean open;
      private byte buffer[];
      private String endcmd = ""+(char)10+(char)13;
      final String hex="0123456789ABCDEF";

    /**
     * Constructs a <code>TestBus</code> interface
     * on the nominated serial port
     *
     * @param   portname      The serial port name eg. COM1
     * @exception TestBusException if the port could not be opened/configured
     */
      public TestBus(String portname)
            throws TestBusException
      {
            set_timeout(1000);
            set_samples(1);
            set_freq(57);
            open=false;
            buffer = new byte[16];

      //Uncomment if using SerialIO library instead of Sun’s JavaCOMM API
      //CommPortIdentifier.addPortName("COM3", CommPortIdentifier.PORT_SERIAL, null);
      //CommPortIdentifier.addPortName("COM4", CommPortIdentifier.PORT_SERIAL, null);

      // Obtain a CommPortIdentifier object for the port you want to open.
            try {
        portId =
         CommPortIdentifier.getPortIdentifier(portname);
      } catch (NoSuchPortException e) {
            throw new TestBusException(e.getMessage());
      }

      // Open the port represented by the CommPortIdentifier object. Give
      // the open call a relatively long timeout of 30 seconds to allow
      // a different application to reliquish the port if the user
      // wants to.
      try {
          sPort = (SerialPort)portId.open("ContestEntry201", 10000);
      } catch (PortInUseException e) {
          throw new TestBusException(e.getMessage());
      }

      // Set connection parameters, if set fails return parameters object
      // to original state.
      try {
            sPort.setSerialPortParams(19200,SerialPort.DATABITS_8,
                              SerialPort.STOPBITS_2,SerialPort.PARITY_NONE);
            sPort.setFlowControlMode(SerialPort.FLOWCONTROL_NONE );

      } catch (UnsupportedCommOperationException e) {
            sPort.close();
            throw new TestBusException("Unsupported parameter");
      }

      // Open the input and output streams for the connection. If they won't
            // open, close the port before throwing an exception.
      try {
            streamo = sPort.getOutputStream();
            streami = sPort.getInputStream();
      } catch (IOException e) {
            sPort.close();
            throw new TestBusException("Error opening I/O streams");
      }

        sPort.setDTR(true);
        sPort.setRTS(true);

      open = true;

      }

      /**
       *    Perform any cleanup operations
       **/
      public void clean()
      {
            if(open==true)
                  sPort.close();

            open=false;
      }

      /**
       * set the response timeout to wait for a pod to response
       *
       * @param n       The timeout in milliseconds
       */
      public void set_timeout(int n)
      {
            if(n < 1)
                  n=1;
            timeout=n;
      }


      /**
       * set the number of samples to take when measuring an analog input
       *
       * @param n       The number of samples (1..255)
       * @exception     If the value is out of range
       */
      public void set_samples(int n)
            throws NumberFormatException
      {
            if(n < 1 || n > 255)
                  throw new NumberFormatException("Samples is out of range");

            samples=n;
      }


      /**
       * set the sampling frequency of analog inputs
       *
       * @param hz      The sampling frequency (8..480 Hz)
       * @exception     If the value is out of range
       */
      public void set_freq(int hz)
            throws NumberFormatException
      {
            if(hz < 8 || hz > 480)
                  throw new NumberFormatException("Frequency is out of range");

            //
            // sampling freq = data clock / 65*256
            // data clock = 24M / divider
            // ie. divider = 24M / 65*256*sampling freq
            //
```

```
            dividek = (int)(24.0e6 / (65*256*hz));

            //System.out.println("Divider = "+dividek);

            freq=hz;
      }


/**
** Set a single-ended digital output to a particular state
**
** @param   port  The port to drive (B5..7=POD[0..7] B3..4=PORT[0..3] B0..2=BIT[0..7])
** @param   state 0=LO,1=HI
** @param   mode  0=5.6K pulldown 1=CMOS/Strong 2=HighZ 3=5.6K pullup
** @exception TestBusException if there was a problem with the bus
*/
public void drive_digital_output(int port, int state, int mode)
            throws TestBusException
{
            String cmd;

            cmd = "!"+(char)(((port>>5)&7)+'0')+"="+((port>>3)&3)+(port&7);

            if(state < 0 || state > 1)
                  throw new TestBusException("Invalid state in drive_digital_output");

            if(mode < 0 || mode > 3)
                  throw new TestBusException("Invalid mode in drive_digital_output");

            cmd = cmd + (char)(mode+'0') + (char)(state+'0') + endcmd;

            System.out.print(cmd);

            try {
                  streamo.write(cmd.getBytes(),0,cmd.length());
                  streamo.flush();
            } catch(IOException ioe) {
                  throw new TestBusException(ioe.getMessage());
            }
}

/**
** Set a double-ended digital output to a particular state
**
** @param   portp The positive port to drive
**                (B5..7=POD[0..7] B3..4=PORT[0..3] B0..2=BIT[0..7])
** @param   portn The positive port to drive
**                (B5..7=POD[0..7] B3..4=PORT[0..3] B0..2=BIT[0..7])
** @param   state 0=LO,1=HI
** @param   mode  0=Open drain 1=CMOS 2=HighZ 3=Open collector
** @exception TestBusException if there was a problem with the bus
*/
public void drive_digital_output(int portp,int portn, int state, int mode)
            throws TestBusException
{
            if(portp >= 0)
                  drive_digital_output(portp,state,mode);
            if(portn >= 0)
                  drive_digital_output(portn,state ^ 1,mode);
}

/**
** Set a single-ended analog output into a particular state
**
** @param   port  The port to drive (B5..7=POD[0..7] B3..4=0 B0..2=BIT[0..7])
** @param   value The value to drive 0..1
** @exception TestBusException if there was a problem with the bus
*/
public void drive_analog_output(int port, double value)
            throws TestBusException
{
            String cmd;
            int ivalue = (int)(value*62.0);

            cmd = "!"+(char)(((port>>5)&7)+'0')+"W"+
                  (port&7)+(char)(hex.charAt(ivalue>>4))+
                  (char)(hex.charAt(ivalue&15))+endcmd;
            System.out.print(cmd);

            try {
                  streamo.write(cmd.getBytes(),0,cmd.length());
                  streamo.flush();
            } catch(IOException ioe) {
                  throw new TestBusException(ioe.getMessage());
            }

   }

/**
** Set a double-ended analog output into a particular state
**
** @param   portp The positive port to drive (B5..7=POD[0..7] B3..4=0 B0..2=BIT[0..7])
** @param   portn The positive port to drive (B5..7=POD[0..7] B3..4=0 B0..2=BIT[0..7])
** @param   value The value to drive 0..1
** @exception TestBusException if there was a problem with the bus
*/
public void drive_analog_output(int portp, int portn, double value)
            throws TestBusException
{
            if(portp >=0)
                  drive_analog_output(portp,value);
            if(portn >=0)
                  drive_analog_output(portn,1.0-value);

}

//
// Wait for data to be present on the input stream
//
private boolean waitData(int bytes)
            throws TestBusException
{
      int gotbytes=0;
      int to=timeout;

    while((gotbytes < bytes) && (to > 0))
    {
            try {
              gotbytes = streami.available();
            } catch(IOException ioe) {
                  throw new TestBusException(ioe.getMessage());
            }

     if(gotbytes < bytes)
     {
            //System.out.println("Got "+gotbytes+" Expecting "+bytes+" Sleep "+to);

            try
            { // Snoozing a bit.
                Thread.sleep(10);
            }
            catch (Exception e)
            {
                        throw new TestBusException("Couldn't sleep "+e.getMessage()
            }
        to -= 10;
     }
    }

    return (gotbytes >= bytes);
}

/**
** Read the state of a single-ended digital input
**
** @param port The port to test (B5..7=POD[0..7] B3..4=PORT[0..3] B0..2=BIT[0..7])
** @param force_input Whether to force the pin to an input first
** @exception TestBusException if there was a problem with the bus
** @return int indicating whether the input was high or low
*/
public int read_digital_input(int port,boolean force_input)
            throws TestBusException
{
```

```
            String cmd;

             cmd = "!"+(char)(((port>>5)&7)+'0')+"?"+((port>>3)&3)+(port&7);

             if(force_input)
                  cmd = cmd + "1";
             else
                  cmd = cmd + "0";

             cmd=cmd+endcmd;

             System.out.print(cmd);

             try {
                  streamo.write(cmd.getBytes(),0,cmd.length());
                  streamo.flush();

                  while(true==waitData(1))
                  {
                              streami.read(buffer,0,1);
                              if(buffer[0]=='*')
                              {
                                    if(true==waitData(1))
                                    {
                                          streami.read(buffer,1,1);
                                          return buffer[1]-'0';
                                    }
                              }
                  }
                  throw new TestBusException("Timeout");

            } catch(IOException ioe) {
                  throw new TestBusException(ioe.getMessage());

            }

}

/**
** Read the state of a single-ended or double-ended analog input
**
** @param port_p The positive port to test (B5..7=POD[0..7] B3..4=PORT[0..3]
**                                              B0..2=BIT[0..7]) (0=AGND)
** @param port_n The negative port to test (B5..7=POD[0..7] B3..4=PORT[0..3]
**                                              B0..2=BIT[0..7]) (0=AGND)
** @param gaink The gain setting for the signal conditioning op-amp
** @exception TestBusException if there was a problem with the bus
** @return int analog value between -2047 and 2047
*/
public int read_analog_input(int port_p,int port_n, int gaink)
            throws TestBusException
{
            String cmd,resp;
            int retval,unit=0;
            char cport_p,cport_n;

            // some sanity checks
            if(port_p >= 0 && port_n >= 0)
            {
             if((port_p >> 5) != (port_n >> 5))
                  throw new TestBusException(
                  "Analog positive and negative inputs must be on the same pod");
            }

            // determine port names
            if(port_p < 0)
                  cport_p = '8';
            else
            {
                  cport_p = (char)((port_p & 7)+'0');
                  unit=(port_p >> 5)&7;
            }

            if(port_n < 0)
                  cport_n = '8';
            else
            {
```

```
			cport_n = (char)((port_n & 7)+'0');
			unit=(port_n >> 5)&7;
		}

		cmd = "!"+(char)(unit+'0')+"R"+cport_p+cport_n ;

		cmd = cmd + hex.charAt(gaink>>4) + hex.charAt(gaink&15) ;
		cmd = cmd + hex.charAt(samples>>4) + hex.charAt(samples&15);
		cmd = cmd + hex.charAt(dividek>>4) + hex.charAt(dividek&15);
		cmd = cmd + endcmd;

		System.out.print(cmd);

		try {
			streamo.write(cmd.getBytes(),0,cmd.length());
			streamo.flush();

			while(true==waitData(1))
			{
					streami.read(buffer,0,1);

					if(buffer[0]=='*')
					{

						if(true==waitData(4))
						{
							streami.read(buffer,1,4);

							resp = ""+(char)buffer[1]+(char)buffer[2]+
										(char)buffer[3]+(char)buffer[4];

							try {
								retval = Integer.parseInt(resp, 16);
							} catch(NumberFormatException nfe) {
								throw new TestBusException("Garbled
response "+nfe.getMessage());
							}

							if(retval > 32767)
								retval = retval-65536;

							return retval;
						}
					}


			}
			throw new TestBusException("Timeout");

		} catch(IOException ioe) {
			throw new TestBusException(ioe.getMessage());

		}

	}

}
```

11.4 LimitedStyledDocument.java

```
/* @(#)LimitedStyledDocument.java  1.0 02/28/04 Entry201

Title:                  PSOC Contest Entry 201
Description:            Text Editor Document Class
Development Tools:      Sun JDK 1.3, Sun JavaCOMM API

*/


import javax.swing.*;
import javax.swing.text.*;
import java.awt.Toolkit;

public class LimitedStyledDocument extends DefaultStyledDocument {
    int maxCharacters;

/**
 * Constructs a <code>LimitedStyledDocument</code>
 * with the specified maximum size.
 *
 * @param   maxChars  The maximum number of characters that the document can contain.
 */

public LimitedStyledDocument(int maxChars)
{
        maxCharacters = maxChars;
}

/**
 * Erases the contents of the document
 **/

public void clear()
              throws BadLocationException
{
            if(super.getLength() > 0)
                 super.remove(0,super.getLength());
}

/**
 * Inserts a text string into the document
 *
 * @param offs    The character offset within the document
 * @param str     The string to be inserted
 * @param a       The font attributes associated with the string
 */

public void insertString(int offs, String str, AttributeSet a)
        throws BadLocationException
{

        //This rejects the entire insertion if it would make
        //the contents too long. Another option would be
        //to truncate the inserted string so the contents
        //would be exactly maxCharacters in length.
        if ((getLength() + str.length()) <= maxCharacters)
            super.insertString(offs, str, a);
        else
            Toolkit.getDefaultToolkit().beep();
    }
}
```

11.5 TestBusException.java

```
/* @(#)TestBusException.java  1.0 02/28/04 Entry201

Title:                  PSOC Contest Entry 201
Description:            Exception for Serial Interface to Test Pods
Development Tools:      Sun JDK 1.3, Sun JavaCOMM API

*/

public class TestBusException extends Exception {

    /**
     * Constructs a <code>TestBusException</code>
     * with the specified detail message.
     *
     * @param   str   the detail message.
     */
    public TestBusException(String str) {
        super(str);
    }

    /**
     * Constructs a <code>TestBusException</code>
     * with no detail message.
     */
    public TestBusException() {
        super();
    }
}
```

12. PROJECT PROTOTYPE PHOTOS