Mini_CT Pascal Code Functions

Why change the program? Perhaps you need to find a value that is not present such as RMS voltage or time in the XT display mode. You want to add a script command to set the cursor index to the peak current value. You don't need a curve tracer but want to build a low frequency oscilloscope or make a photo spectrometer.

This document is by no means comprehensive nor is the code much to be proud of. After all it's a hobby, not a job. My style is to use global variables as much as possible. You will find the functions and procedures have access to all global variables needed for their operation. A minimal number of parameters are passed to the procedures.

To start programming, open Project1.dpr. Almost all the program uses Form1 and UnitCT. The other forms and units are for multiple plot displays, pop up messages and about information. When the project compiles, the executable is project1.exe.

Procedures and functions are defined in use order. This eliminates the need for forward declarations. To use procedures and functions, place your code at the end of the program.

The program is a single process interrupted by Timers or by Form Component events. Timers are used extensively to perform background sequential operations. Timer **tmrchstatus** handles receipt of serial data. In addition, it scales received data and plots it to the screen. Other timers handle continuous acquisition and script program commands.

It is most likely you will want to manipulate data to extract information. For insight into where data is stored and how to locate it, look at the **doredraw()** function description.

<u>Timer Procedures:</u>

1. **tmrchkstatus**
    a. Period 2ms
    b. Always enabled
    c. Calls **checkforbyte()**
        i. Check for received serial data charcters
            1. Save in character array **cline[]**
            2. If no characters present, exit
        ii. On end of record char(129), process **cline[]**
            1. Extract 14-bit integer values for Collector Voltage and Current
            2. Save values in integer array **addata[sample, type]**
                a. sample = data point 0 to 255
                b. type 0: Collector-V, type 1: Collector-I, type 2: Base-V, type 3: Base-I

3. Call **fixdata()**
    a. Offset and scale Collector **addata[]** and save in **V[]** array
    b. Obtain Base voltage and Base current from Form settings then save values in **V[] array**
    c. Extract switch setting data from end of record and update Form
    d. If selected on Form, Filter and Average data in **V[]** array
    e. If acquisition, filtering and averaging are completed, copy **V[]** array to **traces[]** array
    f. Call **doredraw()**
        i. Plot data from selected **traces[]** array on the Form
4. If there are less than 256 data points, then convert the data to a DC measurement
    a. Sum and average data to generate DC values
    b. Update DC values on Form
    c. If **calnum** != 0, use DC values for calibration of gains and offsets

2. timer7

a. Triggers **Run Sweep** event to implement continuous trace acquisition
b. Enabled when Live is checked
c. Period is equal to **delay1** or **delay2** values set by Rate menu
    i. **delay1** is for the AC sweep mode of acquisition: 200ms, 100ms, 25ms
    ii. **delay2** is for the slow sweep mode of acquisition: 400ms, 200ms, 25ms

3. timer5

a. Period is set by **Script Control Step Time** of 10ms, 100ms or 1000ms
b. Enabled by **Run Script** button event. Also causes **timer6** to become disabled
c. Processes Script program one line per pass
    i. Increments program line pointer each pass
    ii. Exits upon **stop** script command
d. Coding new script commands:
    i. The command string is all lower case followed by a space
        1. The interpreter converts all program code to lower case
    ii. An integer value may follow the command string
        1. Extract the integer value with a **copy** function
            a. **i** is the pointer to the program line string position matching the command
            b. **s** contains the script program string and is parameter 1
            c. **n** is the length of the program line
            d. Use **i+length** for parameter 2
                i. length('setcollector ') = 13
                ii. There is a space after the r
                iii. Parameter 2 = **i+13**
            e. Use **n-i-(length-1)** for parameter 3
                i. length('setcollector ')-1 = 12

ii. Parameter 3 = **n-i-12**

iii. In this example, the spin edit box **CollectorV** value is changed to the integer following the command string 'setcollector ' and the DAC0 voltage is updated.

1. A procedure call to **dosectcollector** changes the collector voltage.
2. To set the collector to 5V, the script command is:
   setcollector 500
3. Use unique names for the command strings

Example Pascal Code for adding a Script command:

```
i := Pos('setcollector ', s);    // Test for the command string setcollector
 if i = 1 then begin             // If it matches at the beginning, process it
  es := 'Numeric Input Error';   //es is the error string
  form1.CollectorV.value :=  strtoint(trim(copy(s,i+13,n-i-12)));  //extract integer
  dosetcollector;              //call the procedure to set the collector voltage
  exit;                        //exit the timer5 procedure, all done
end;
```

4. **timer6**
   a. Queries switch settings to update Form when Live mode is inactive
   b. Period 500ms
   c. Enabled except when running Script program or disabled by menu

Display Procedures and Functions:

1. **doredraw()**
   a. Plots traces onto form
      i. Single Trace
      ii. Multiple Traces
      iii. Comparison Traces
      iv. Sets Trace color depending on plot type
   b. Uses Form control variables to identify trace to plot
      i. Part traces
         1. **Form1.SweepStep.Value** = pin+
         2. **Form1.TraceNumber.Value** = pin-
         3. **Form1.SpinSweepSteps .Value**
      ii. Compare traces
         1. **Form1.spinPC.Value** = pin+
         2. **Form1.spinNC.Value** = pin-
   c. Loads data into **V[]** array from **traces[]** array
      i. Locates trace at pin+ and pin- using function **hash()**
         1. **index** = **hash(trace, pin-, pin+, RW)**
            a. **trace** = 0 part, **trace** = 1 compare

i. Part is the normal trace storage for measurement
ii. Compare is secondary storage for comparing devices

b. **pin+** = Collector positive pin number
c. **pin-** = Collector negative (GND) pin number
d. **RW** = 0 locate trace, RW = 1 save trace
e. **index** = Integer pointer to trace data in traces[] array
f. hash table is stored in **pins[trace, index, pin]** array
    i. **trace** = 0 part, **trace** = 1 compare
    ii. **index** = Integer pointer to trace data in traces[] array
    iii. **pin** = 0 array location for **pin+** number
    iv. **pin** = 1 array location for **pin-** number

2. Single Array **V[sample, type]**
    a. Holds calibrated data
    b. Is used for trace plotting
        i. Place new data in this array to plot your own values
    c. **sample** = data point 0 to 255
    d. **type** = 0: Collector-V, **type** = 1: Collector-I, **type** = 2: Base-V, **type** = 3: Base-I
3. Single Array **traces[trace, index, sample, type]**
    a. Storage array for part traces and compare traces
    b. **trace** = 0 part, **trace** = 1 compare
        i. Part is the normal trace storage for measurement
        ii. Compare is secondary storage for comparing devices
    c. **index** = Integer pointer obtained from hash() function
    d. **sample** = data point 0 to 255
    e. **type** = 0: Collector-V, **type** = 1: Collector-I, **type** = 2: Base-V, **type** = 3: Base-I

d. Calls procedure **plotdata()**, after each trace is loaded into array **V[]**
e. Calls procedure **paintdib()** to put plot on form
    i. Uses **myimage**
        1. 24 bit color
        2. 513x513 pixels

2. Plotdata()

a. Generates BitMap image **myimage**
    i. Makes screen reticule if **flagmult**i = false.
        1. **flagmulti** = true suppressed reticule overwrite to allow multiple trace plots
    ii. Plots V-I data or V-t, I-t data
        1. **PlotType** radio button determines XY or XT plot
        2. Uses Form parameters to scale and offset plots

a. **UpDownVO** and **UpDownIO** spin buttons set the number of **Voffset** and **Ioffset** divisions
b. **SpinVolts** and **SpinMA** buttons set **vperdiv** and **iperdiv** plot scales

3. Places optional cursor on trace plot
    a. The **SamplePoint** spin edit value selects **V[sample, 0]** voltage and **V[sample, 1]** current values for screen placement

iii. Places text on plot

1. Base step information
    a. **V[0,2]** is the Base voltage step value
    b. **V[0,3]** is the Base current step value
2. Trace information
    a. Plot Scaling and Offset values
    b. Cursor voltage and current
    c. Trace pin+ and pin- numbers
    d. Serial Number
    e. Calculated Parameters

Example Pascal Code to assign data to V[] array then plot it:

```
//On Form1, make control ButtonExample property visible equal to true

procedure TForm1.ButtonExampleClick(Sender: TObject);
var
 j, i : integer;
begin
 flagmulti := false;                  // Force drawing of reticule
 plotcolor := $00a700;               // Initial plot color is green
 for j := 0 to maxpoints-1 do begin    // save the data to a trace plot array
   v[j,1] := 10.0*sin(j*0.05);
   v[j,2] := 0.01*cos(j*0.2+ 0.75);
   v[j,3] := 0;
   v[j,4] := 0;
 end;
 i := hash(0,form1.TraceNumber.value,form1.SweepStep.value,1);  //find a table entry
 if i <> -1 then begin
  pins[0,i,0] := form1.TraceNumber.value;    //Update pin numbers
  pins[0,i,1] := form1.SweepStep.value;
  for j := 0 to maxpoints-1 do begin       // save the data to a traces[] array
   traces[0,i,j,1] := v[j,1];                  //Image is not persistent unless saved to traces[] array
   traces[0,i,j,2] := v[j,2];
   traces[0,i,j,3] := v[j,3];
   traces[0,i,j,4] := v[j,4];
```

```
  end;
 end;
 plotdata;                    //generate the graph
 paintdib;                    //paint graph on the Form
end;
```

Control and Acquisition Procedures:

1. **DoRunSweep()**
    a. Runs a trace sweep acquisition
    b. If **sweeptype** radio itemindex = 0, run a "SWP0256" command
    c. If **sweeptype** radio itemindex = 1, run a "MEA0256" command
2. **DoSetCollector()**
    a. Converts the **CollectorV** spin edit value to a DAC integer
    b. Sends command "DA0" + 4-digit string of DAC integer
3. **DoSetBaseV()**
    a. Converts the **BaseVolts** spin edit value to a DAC integer
    b. Sends command "DA1" + 4-digit string of DAC integer
4. **DoSetBaseI()**
    a. Converts the **BaseuA** spin edit value to a DAC integer
    b. Sends command "DA1" + 4-digit string of DAC integer
5. **DoSetValues()**
    a. Converts the **CollectorStart** spin edit value to a DAC integer
    b. Sends command "STA" + 4-digit string of DAC integer
    c. Converts the CollectorStep spin edit value to a DAC integer
    d. Sends command "STP" + 4-digit string of DAC integer
6. **DoReadVoltages(*N*)**
    a. 1 <=*N* <= 250
    b. Does a DC measurement read returning *N* values that are averaged to a single measurement point
    c. Sends command "DCM" + 4-digit string of *N* integer
7. **SetInteger('*c1*', '*c2*', '*c3*', *N*)**
    a. Sends three character command string *c1+c2+c3*
    b. Followed by 4-digit string of integer *N*