REFERENCES

Abou-Elfotouh,F.A., L.L.Kazmerski, R.J.Matson, D.J.Dunlavy, and T.J.Coutts, J. Vac. Sci. Technol. **A8**, 3251 (1990).

Andrews,J.M., and J.C. Phillips, Phys. Rev. Lett. **35,** 56 (1975).

Bardeen, J., Phys. Rev. **71,** 717 (1947).

Bowron,J.W., S.D.Damaskinos, and A.E.Dixon, Solar Cells. **31,**159 (1991).

Braicovich,L., C.M.Garner, P.R.Skeath, C.Y.Su, P.W.Chye, I.Lindaue, and W.E.Spicer, Phys. Rev. **B20**, 5131 (1979).

Braun, F., Pogg. Ann. **153**, 556 (1874).

Brillson, L.J., Phys. Rev. Lett. **40,** 260 (1978).

_____.,L.J., J. Vac. Sci. Tecnol. **16,** 1137 (1979).

Card, H.C. and E.H.Rhoderick, J. Phys. D., Appl. Phys. **4,** 1589 (1971).

Chye,P.W., I.Linday, P.Pianetta, C.M.Garner, C.Y.Su, and W.E.Spicer, Phys. Rev. **B18**,5545 (1978).

Cox,R.H. and H.Strack, Solid-State Electron. **10**, 1213 (1967).

Damaskinos,S., J.D.Meakin, and J.E.Phillips, *in Proceedings of the 19th IEEE Photovoltaic Specialists Conference* ,(IEEE, New York, 1987), p.1299.

Fahrenbruch,A.L. and H.Bube, *Fundamentals of Solar Cells*, Academic Press. New York, 1983.

Fowler,R.H., *Statistical Mechanics*, Cambridge Univ. Press, London, 1955.

Freeouf, J.L. and J.M.Woodall, Appl. Phys. Lett. **39,** 727 (1981).

Hein,V., Phys. Rev. **A 138,** 1689 (1965).

Hiraki,A., M.A.Nicolet, and J.W. Mayer, Appl. Phys. Lett. **18**, 178 (1971).

Komatsu,S., M.Nakahashi, and Y.Koike, Japan. J. Appl. Phys. **20**, 549 (1981).

Kupka,R.K.,W.A.Anderson, J. Appl. Phys. **69**, 3623 (1991).

Look, D.C., *Electrical Characterization of GaAs Materials and Dedices*, John Wiley & Sons, New York, 1989.

Matson,R.J., O.Jamjoum,A.D.Buonaquisti, P.E.Russell, L.L.Kazmerski, P.Sheldon, and R.K.Ahrenkiel, Solar Cells. **11**,301 (1984).

Mattheiss,L.F., Phys. Rev.**134**, A970 (1964).

Mickelsen,R.A. and W.S.Chen, *in Proceedings of the 16th IEEE Photovoltaic Specialists Conference* ,(IEEE, New York, 1982), p.781.

Moons, E., T. Engelhard, and D. Cahen, J. Electron. Mater. **22,** 275 (1993).

Nelson, A. J., S. Gebhard, L.L. Kazmerski, E. Colavita, M. Engelhardt, and H. Hochst, J. Vac. Sci Technol. **A9,** 978 (1991).

Neuman, H., Sol. Cells. **16,** 317 (1986).

Padovani, F.A. and R. Strattron, Solid-State Electron. **9,** 695 (1966).

Phillips, J.C., J.Vac. Sci. Technol. **11,** 947 (1974).

Raud, S. and M.A. Nicolet, *Thin Solid Films.* **361** (1991).

Rhoderick, E.H., *Metal-Semiconductor Contacts*, Clarendon Press. Oxford, 1978.

Rockett, A. and R.W.Birkmire, Appl. Phys. **70, R 81** (1991).

Russell, P.E., O.Jamjoum, R.K.Ahrenkiel, L.L.Kazmerski, R.A.Mickelsen, And W.S.Chen, Appl. Phys. Lett. **40,** 995 (1982).

Schottky, W. Z.Phys. **113,** 367 (1939).

_____ . W. Naturwiss. **26,** 843 (1938).

Schroder,D.K. and D.L.Meier, IEEE Trans.ED **,ED 31,**637 (1984).

Schroder,D.K., *Semiconductor Material and Device Characterization*, John Wiley & Sons, New York, 1990.

Sinha,A.K. and J.M.Poate, Appl. Phys. Lett. **23**, 666 (1973).

Stolt,L., J.Hedstrom, J.Kessler, M.Ruckh, K.O.Velthaus, and H.W.Schock, Appl. Phy. Lett. **62**, 597 (1993).

Spicer, W.E., P.W.Chye, P.R. Skeath, C.Y. Su, and I.Lindau, J.Vac. Sci. Technol. **16,** 1422(1979).

Spicer,W.E., I.Lindau, P.R.Skeath, and C.Y. Su, J.Vac.Sci.Technol. **17,**1019 (1980).

Sze,S.M., *Physics of Semiconductor Devices*, John Wiley & Sons, New York, 1981.

Tantraporn, W., Solid State Electron. **7,** 81 (1964a).

_____.,W., General Electric Technical Information Series. **No.64-RL-3783E** (1964b).

_____.,W., General Electric Technical Information Series. **No.66-C-004** (1966).

_____., W., J. Appl. Phys. **41,** 4669 (1970).

_____.,W., IEEE Trans. E.D. **ED-19,** 331 (1972).

_____.,W., and Stephens, J.M.III, *"Determination of Barrier Height at Apparently Ohmic Contacts", presentation to Electro-chemical Society Meeting* Hollywood, FL, 1980.

Toro, C.R., *Metal Contacts to $CuInSe_2$*. Ph.D. thesis, Brown University, (1986).

Yu, A.Y.C., Solid. State. Electron. **13**,239 (1970).

Wagner,S., J.L.Shay, P.Migliorato, and H.M.Kasper, Appl. Phys. Lett. **25,** 434(1975).

Wasim,S.M., Sol. Cells. **16,** 289 (1986).

Williams, R.H., Contemp. Phys. **23,** 329 (1982).

.....

APENDIX A

DERIVATION OF THE CURRENT DENSITY

The original derivation of forward bias, n-type semoconductor is in the section 4 of Kupka(1991). We apply this derivation for the case of p-type semiconductor under reverse bias, as shown :

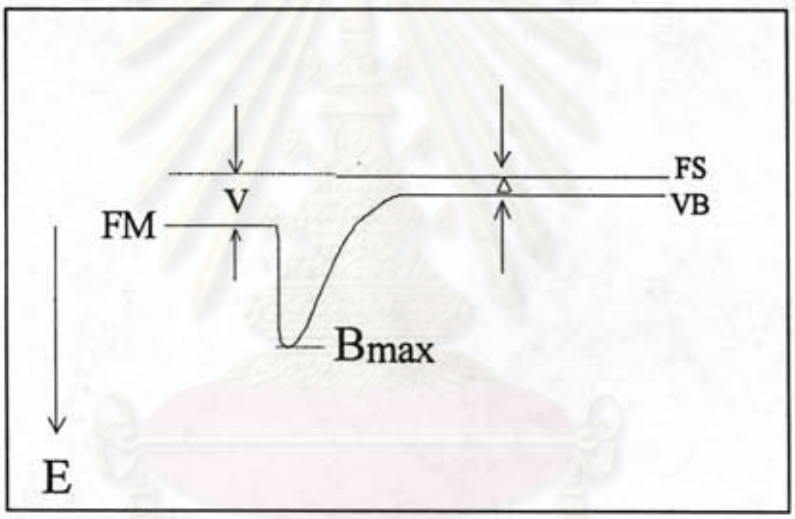


Fig.21 The Schottky barrier of p-type semiconductor under reverse bias. The FM and FS are the metal and the semiconductor Fermi levels, respectively. The VB is the valence band edge. Bmax is the maximum of the barrier shape including image foce lowering. The Δ is the difference between the valence band edge and the semiconductor Fermi level, assigned a plus sign in the case shown in this figure.

For convenience, in p-type semiconductor the energy is measured downward.

At the center of the Brillouin zone of p-$CuInSe_2$, the light and heavy hole band are nearly degenerate. We assume that they are degenerate, and that the current from the two band are additive.

Consider the light holes first. The semiconductor-metal current density for light hole is :

$$J^1_{sm} = q\int_{\vec{p}} v_x(\vec{p})D(\vec{p})[1-R(\vec{p})]H_p(\vec{p})d\vec{p} \qquad (A1)$$

where H_p = the number light holes per unit volume in phase space

v_x = hole velocity in the x direction

D(p) = semiconductor-metal transmission probability

R(p) = reflection probability in metal.

Physically in the above equation, the electron tunnel from the metal to the semiconductor.

Using

$$H(p) = \frac{2f_s}{h^3} = \frac{2}{h^3(1+e^{(E+qV-E_f)/kT})} \qquad (A2)$$

$$v_x = \frac{\partial E_x}{\partial P_x} \qquad (A3)$$

Where E is the hole energy, f_S is the semiconductor Fermi-Dirac distribution , E_f is the Fermi level at equilibrium, V in A2 is plus sign for reverse bias. Thus:

$$J^1_{sm} = \frac{2q}{h^3}\int_{E_x}\int_{P_y}\int_{P_z}\frac{D(E)[1-R(E)]dp_y dp_z}{(1+e^{(E+qV-E_f)/kT})}dE_x \qquad (A4)$$

The reflection probability is given by the metal Fermi-Dirac distribution

$$R(E) = f_m = \frac{1}{[1+e^{(E-E_f)/kT}]} \tag{A5}$$

so that

$$J^1_{sm} = \frac{2q}{h^3}\int_{E_x}\int_{P_y}\int_{P_z} \frac{D(E)e^{(E-E_f)/kT}dp_y dp_z}{[1+e^{(E-E_f)/kT}][1+e^{(E+qV-E_f)/kT}]}dE_x \tag{A6}$$

The metal-semiconductor current density is obtained in the same way. It is

$$J^1_{ms} = \frac{2q}{h^3}\int_{E_x}\int_{P_y}\int_{P_z} \frac{D(E)e^{(E+qV-E_f)/kT}dp_y dp_z}{[1+e^{(E-E_f)/kT}][1+e^{(E+qV-E_f)/kT}]}dE_x \tag{A7}$$

The total reverse current density is

$$J^1_R = J^1_{ms} - J^1_{sm}$$

$$= \frac{2q}{h^3}\int_{E_x}\int_{P_y}\int_{P_z} \frac{D(E)e^{(E-E_f)/kT}[e^{qV/kT}-1]dp_y dp_z}{[1+e^{(E-E_f)/kT}][1+e^{(E+qV-E_f)/kT}]}dE_x \tag{A8}$$

Assuming

$$E = \frac{p^2}{2m_l^*} \quad \text{and} \quad E = E_X + E_y + E_Z \tag{A9}$$

Where m_l^* is the effective mass of light hole at the band edge.

By given

$$Y = \frac{P_y}{(2m_l^* kT)^{1/2}};\ Z = \frac{P_z}{(2m_l^* kT)^{1/2}};\ a = \frac{E_x - E_f}{kT};\ u = \frac{qV}{kT} \tag{A10}$$

So that

$$J_R^l = \frac{4qm_l^* kT}{h^3} \int_{E_x} \int_{Y=-\infty}^{\infty} \int_{z=-\infty}^{\infty} \frac{D(E) e^{(a+Y^2+Z^2)} [e^u - 1] dY dZ}{[1+e^{(a+Y^2+Z^2)}][1+e^{(a+Y^2+Z^2+u)}]} dE_x \quad \text{(A11)}$$

transferring into cylindrical coordinates, the angular integration is 2π.

Thus :

$$J_R^l = \frac{4\pi qm_l^* kT}{h^3} \int_{E_x} \int_{r=0}^{\infty} \frac{D(E_x) e^{(a+r^2)} [e^u - 1] 2rdr}{[1+e^{(a+r^2)}][1+e^{(a+r^2+u)}]} dE_x \quad \text{(A12)}$$

Substituting $t = r^2$ gives

$$J_R^l = \frac{4\pi qm_l^* kT}{h^3} \int_{E_x} \int_{t=0}^{\infty} \frac{D(E_x) e^{(a+t)} [e^u - 1] dt}{[1+e^{(a+t)}][1+e^{(a+t+u)}]} dE_x \quad \text{(A13)}$$

The integration produces

$$J_R^l = \frac{4\pi qm_l^* kT}{h^3} \int_{E_x} D(E_x) \ln \left[\frac{e^{-(E_x - E_f)/kT} + 1}{e^{-(E_x + qV - E_f)/kT} + 1} \right] dE_x \quad \text{(A14)}$$

Similarly, for heavy hole

$$J_R^h = \frac{4\pi qm_h^* kT}{h^3} \int_{E_x} D(E_x) \ln \left[\frac{e^{-(E_x - E_f)/kT} + 1}{e^{-(E_x + qV - E_f)/kT} + 1} \right] dE_x \quad \text{(A15)}$$

The current density : $J_R = J_R^l + J_R^h$

So that :

$$J_R = \frac{4\pi q(m_l^* + m_h^*)kT}{h^3} \int_{E_X} D(E_x) \ln\left[\frac{1+e^{-(E_X-E_f)/kT}}{1+e^{-(E_X+qV-E_f)/kT}}\right] dE_X \quad (A16)$$

Assuming that for holes having energy more than B_{max}, the transmission coefficient $D(E_X) = 1$. The final current density is

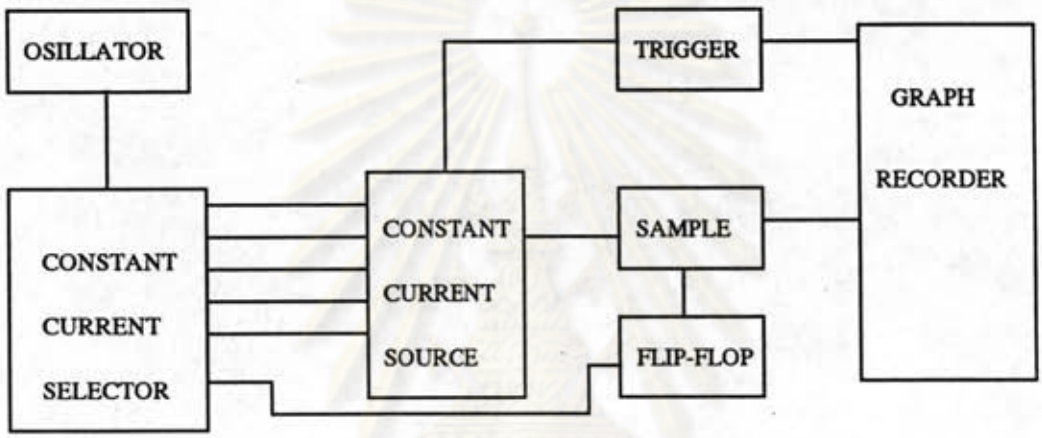
$$J_R = \frac{4\pi q(m_l^* + m_h^*)kT}{h^3} \int_{B\,max}^{\infty} \ln\left[\frac{1+e^{-(E_x-E_f)/kT}}{1+e^{-(E_x+qV-E_f)/kT}}\right] dE_x$$

$$+\frac{4\pi q(m_l^* + m_h^*)kT}{h^3} \int_{-V+\Delta}^{B\,max} D(Ex) \ln\left[\frac{1+e^{-(E_x-E_f)/kT}}{1+e^{-(E_x+qV-E_f)/kT}}\right] dE_x \quad (A17)$$

APPENDIX B

STEP CONSTANT CURRENT

The diagram of step constant current used in DACCT measurement is shown below.



It composes of:

(a) Voltage source, shown in Fig.22a.

(b) Oscillator (clock), the frequency is about 1 Hz, shown in Fig.22b..

(c) Constant current selector, i.e. CD 4017 in Fig.22c. Also shown in this figure are flipflop (in Fig.22f), and relays of constant current source in Fig.22d.

(d) Constant current source, shown in Fig.22d. Also shown in this figure are the resistance values used in shift the zero voltage that sustaining on the sample of each constant current.

(e) time delay trigger, shown in Fig.22e, the time delay used is about 3/4 s.

(f) Flip-flop used in change polarity of constant current, shown in Fig.22f.

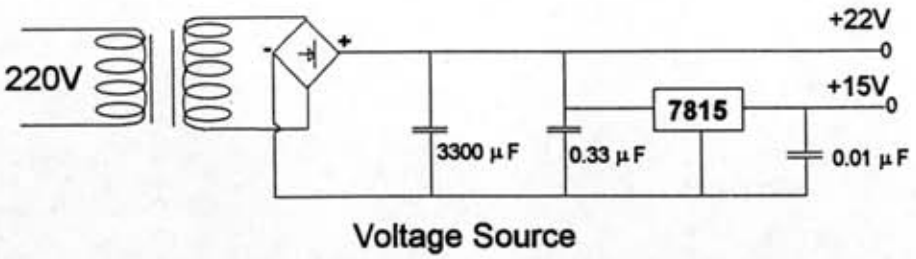


Fig.22a Voltage source +22V and regulated +15V

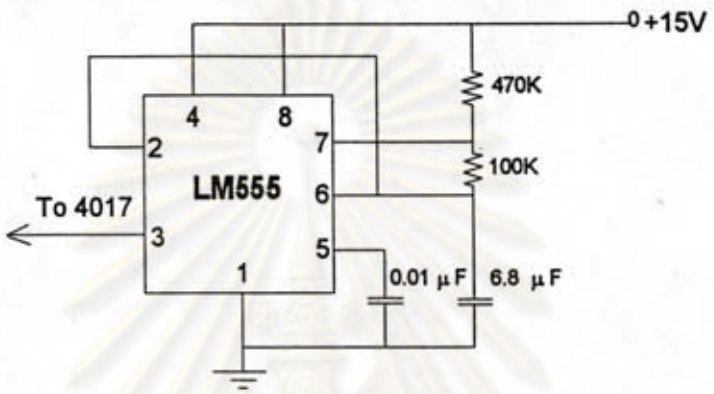


Fig.22b Oscillator 1 Hz

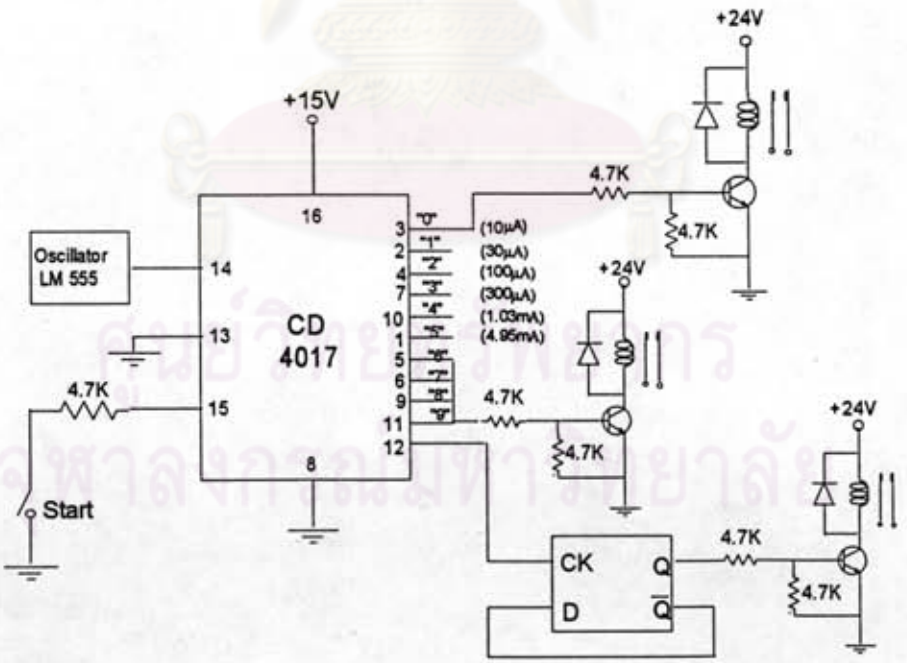


Fig.22c Constant current selector, oscillator, flip-flop, and the relays of constant current source.

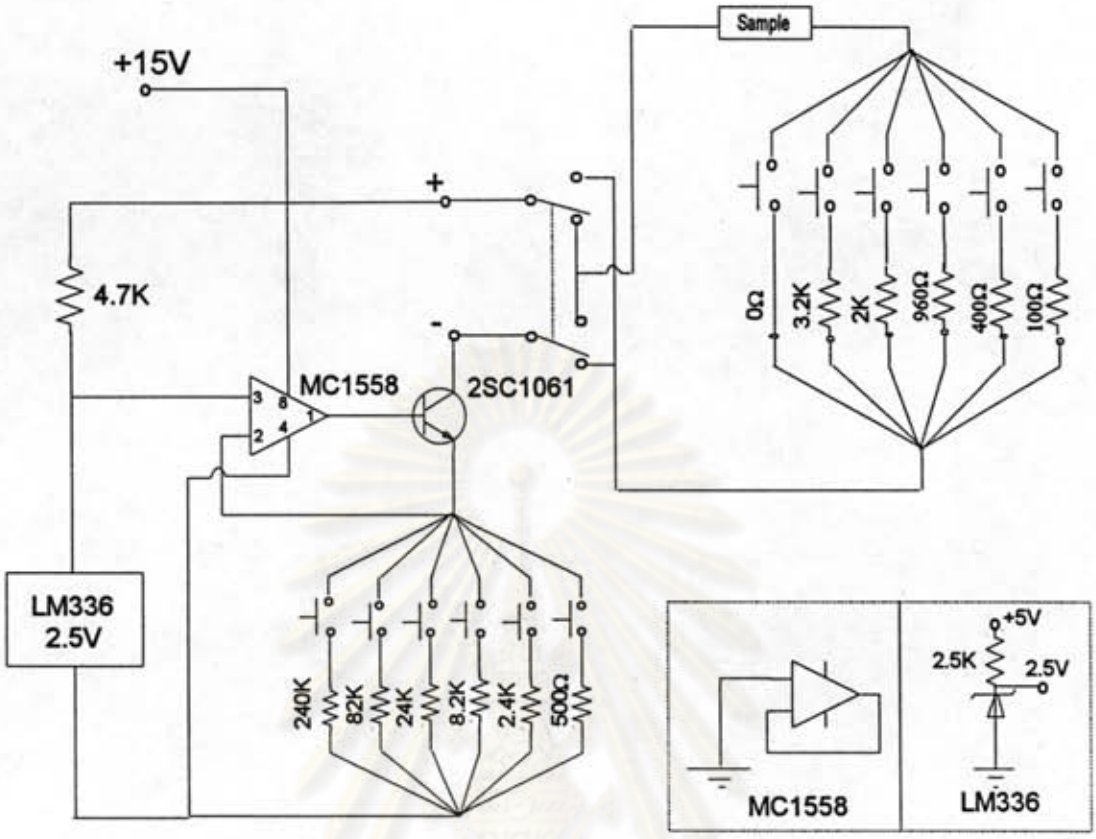


Fig.22d Constant current source, at the top right hand are the resistance used in shift zero voltage. The inset are the remainder ports of MC1558, and the detail of LM336.

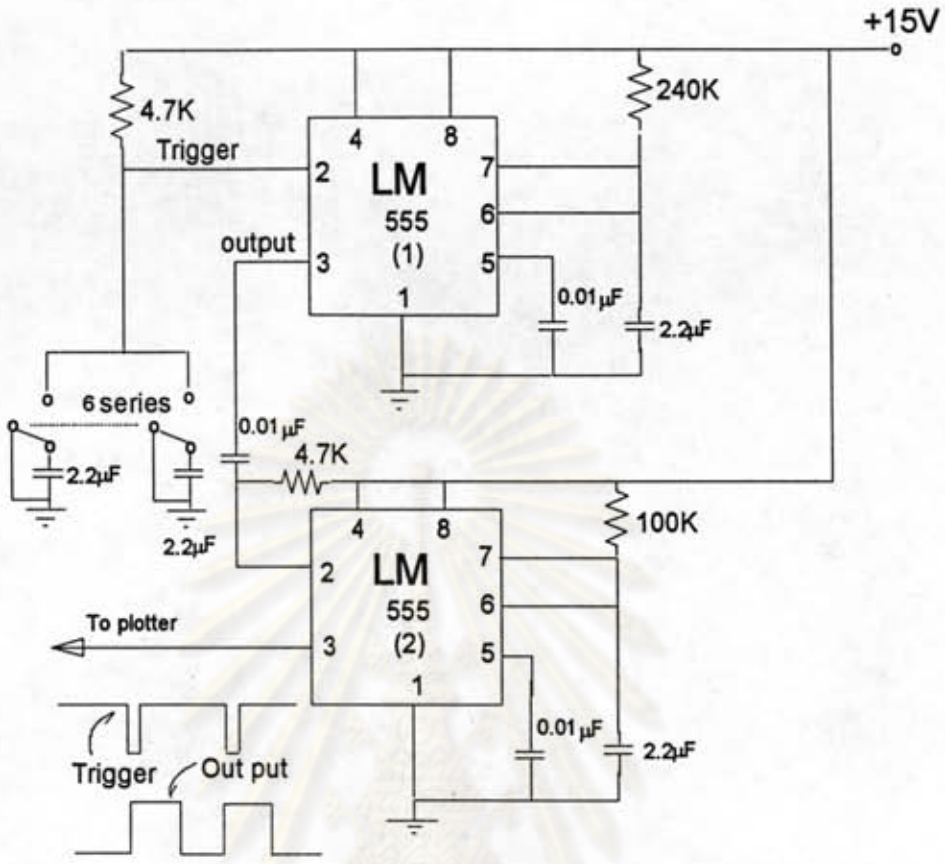


Fig.22e Trigger which compose of two LM555. Inset at the bottom left hand show the input (trigger) and out put of LM555.

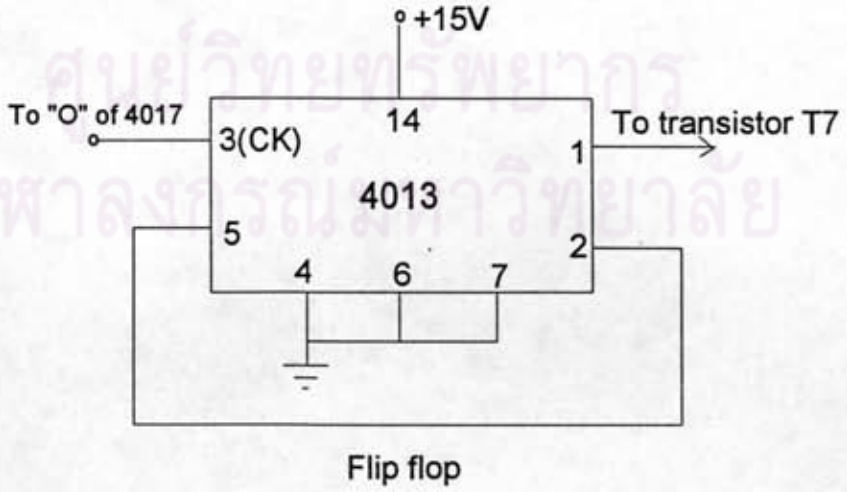


Fig.22f Show the detail of flip-fop.

The time interval that the sample sustains a constant current is limited by oscillator (clock), about 1 s, for each constant current value. At about 3/4 s, the trigger sends a pulse to the graph recorder to record the voltage sustained across the sample. After this, the constant current selector selects a higher constant current value and the cycle repeats. After the highest current value is reached, the sample is allowed to relax to zero with no apply current for about 5 s, then the flip-flop changes polarity and the current selector starts to select the lowest constant current value, and so on.

APPENDIX C

COMPUTER PROGRAM FOR FITTING

Referring to the illustration in Fig.4 for the three-dimensional parameter space, a beginning point O on the ε_1 equi-error surface is selected. At point O, for each parameter, upon testing for the value of error on either side of O, like "gradient", the direction toward smaller errors can be found. A small step in the appropiate direction of the "gradient" thus leads to the smaller error surface ε_2. Further steps can be taken until the error number begin to increase, such as at the point P. The gradient scheme can again be applied at P and similarly thereafter until a minimum error point can be localized at Q within a small volume.

The technique can be applied directly to the m-dimensional parameter space.

The computer program below, i.e. "program fitting", is based on the iterative technique in FORTRAN language of Tantraporn (1966). Here, it was written in PASCAL language. The data was in "data file", at the end of "program fitting".

This computer program in the present form is not yet to be regarded perfect, but can be effectively used.

```
Program Fitting ;
{$N+,E+}
Uses    Crt, Dos, Printer;
Type
Parameter   =   Record
                Para  :   Array[1..6] of Extended;
                Err :Extended;
                Mode :    String[5];
                End;
FullSet =       Array[1..150] of Parameter;
ExpData     =   Record
                X     :   Array[1..5] of Extended;
                Y     :   Extended;
                End;
LineIn  =       String[200];
TemData     =   Record
                TD :Array[1..50] of Extended;
                End;
TemTheo     =   Record
                TT : Array[1..50] of Extended;
                End;
Slab    =       Record
                Energy: Extended;
                  Llimit  : Extended;
                  Ulimit  : Extended;
                  End;
```

```
Var
TDat    :    TemData;              { There are NumPoint - 1 TDat}
NumPoint     : Integer;            { NL in Fortran              }
ErrMinPara   : Parameter;          { EMIN set                   }
OldEMin      : Parameter;          { Old EMIN set used in extra }
ErrLimit     : Extended;           { Limit for min error        }
Found   :    Boolean;              { Find solution or not       }
NewPoint     : Parameter;          { New point found            }
RealNewPoint : Parameter;
NumData      : Integer;            { Number of experimental set }
Data    :    Array[1..50] of ExpData;    { Experimental data    }
ParaStep     : Parameter;                { Small stepping factor }
StepLength   : Extended;                 { Size of ParaStep     }
Adjust  :    Extended;                   { Some adjustable factor}
MaxParaNum   : Integer;                  { Number of parameter  }
Cause   :    String[80];
OutFile :    text;
SteppingNum  : Integer;          { First Parameter to be step   }
PerErrStep   : Extended;{Percent of error's increasing from stepping}
GradNo:      Integer;
S       :    Extended;           { barrier width                }
Bmax    :    Extended;           { maximum barrier shape        }
DBmax :      Extended;      { position of Bmax from interface }
Const
K       = 13.6;    {low frequency dielectric constant of CISe   }
Mden    = 0.73;    {density of state effective mass for p-CISe   }
```

```
dE      = 0.005;{eV}
kB      = 0.0000861733;          {eV/Kelvin}
Procedure InitData ;
Var
  I     : Integer;
Begin
  For I :=    NumData Downto 1 Do
    Begin
      TDat.TD[I] := Data[I].Y;
      If I < NumData Then
      TDat.TD[I] := TDat.TD[I] / TDat.TD[NumData];
    End;
End;

Function tanh ( X : Extended) : Extended;
Var
  tempo : Extended;
Begin
  If X = 0 Then tempo := 0
  Else
  If X > 5 then tempo := 1
  Else
  If X <( - 5)  Then tempo := - 1
  Else tempo := (exp(X) - exp(-X)) / (exp(X) + exp(-X));
  tanh := tempo;
End;
```

```
Procedure Add (A, B: Parameter; Var C : Parameter);
Var
  I : Integer;
Begin
  For I := 1 to MaxParaNum Do
  C.Para[I] := A.Para[I] + B.Para[I];
  C.Err  := 9999;
  C.Mode := 'ADD';
End;

Procedure Sub (A, B: Parameter; Var C : Parameter);
Var
  I : Integer;
Begin
  For I := 1 to MaxParaNum Do
  C.Para[I] := A.Para[I] - B.Para[I];
  C.Err  := 9999;
  C.Mode := 'SUB';
End;

Procedure Mult (K : Extended; A : Parameter; Var C : Parameter);
Var
  I : Integer;
Begin
  For I := 1 to MaxParaNum Do
  C.Para[I] := A.Para[I] * K;
```

```
  C.Err  := 9999;
  C.Mode := 'MUL';
End;

Procedure Transform (P:Parameter; Var RealP:Parameter );
Var
  I :  Integer;
Begin
  RealP.Para[1] := P.Para[1] *0.1;
  RealP.Para[2] := P.Para[2] *0.01;
  RealP.Para[3] := P.Para[3] *0.1;
  RealP.Para[4] := P.Para[4] *1E-7;
  RealP.Para[5] := P.Para[5] *1E-5;
  RealP.Para[6] := P.Para[6] *0.03;
  RealP.Err := 9999;
  RealP.Mode := 'TRANS';
End;

Function TheoryFunc ( RP: Parameter ; X: ExpData ) : Extended;
Var
  OldSlab    : Slab ;
  NewSlab    : Slab ;
  SubA       : Extended ;
  SubTun     : Extended ;
  MaxSubTun  : Extended ;
  ThermioFlux  : Extended ;
```

```
  TunnelFlux     : Extended ;
  StopTunnel     : Boolean ;
Procedure Barwidth ;
Var
  Tempo : Extended;
Begin
  Tempo := ((RP.Para[1] + X.X[1] - RP.Para[2])*K/90.47635/RP.Para[4])
        - (2*RP.Para[5]/RP.Para[4]/RP.Para[6]/RP.Para[6]);
  If Tempo < 2500 Then S:= 999999
  Else
    S  :=Sqrt(Tempo);
End;

Function Barshape (d:Extended) :Extended;
Var
  Tempo : Extended;
Begin
  If d > 1 Then
  Begin
    Barshape :=  (180.95270/K) * ((RP.Para[4]*S*S/2) - (RP.Para[4]*S*d)
      + (RP.Para[4]*d*d/2) + (RP.Para[5]/RP.Para[6]/RP.Para[6])
      * (exp(-RP.Para[6]*d))) - X.X[1] + RP.Para[2]-(3.599940/K/d);
  End
  Else
  Begin
```

```
    Tempo :=    (180.95270/K) * ((RP.Para[4]*S*S/2) - (RP.Para[4]*S)
                + (RP.Para[4]/2) + (RP.Para[5]/RP.Para[6]/RP.Para[6])
                *(exp(-RP.Para[6]))) - X.X[1] + RP.Para[2] - (3.599940/K);
    Barshape := Tempo * d;
  End;
End;

Procedure BarMax;
Var
  d,y,z : Extended;
Begin
  d:= 0.1;
  y:= -9999;
  Repeat
    z:= y;
    d:= d + 0.5;
    y:= Barshape(d);
    If d > 550 Then
      Begin
       y := 99;
       z := 999;
      End;
  Until y < z;
  d:= d - 1.0;
  y:= -9999;
  Repeat
```

```
   z:= y;
   d:= d + 0.025;
   y:= Barshape(d);
   If d > 540 Then
   Begin
      y := 99;
      z := 999;
   End;
Until y < z;
d:= d - 0.05;
y:= -9999;
Repeat
z:= y;
d:= d + 0.001;
y:= Barshape(d);
If d > 530 Then
   Begin
      y := 99;
      z := 999;
   End;
Until y < z;
d:= d - 0.0005;
If d > 500 Then Bmax := 999999
Else
Bmax := Barshape(d);
DBmax := d;
```

```
End;

Procedure ThermioFluxCal;
Var
   y,a,b :Extended;
Const
   kB1  = 1.380662E-16; {erg/Kelvin}
   h1   = 6.626176E-27; {erg*sec}
   m0   = 0.9109534E-27;  {gram}
Begin
   a:= exp(-Bmax/kB/X.X[2]);
   b:= exp((-Bmax - X.X[1])/kB/X.X[2]);
   y:=(4 *Pi *m0 * Mden *kB1 *kB1 *X.X[2] *X.X[2]/h1/h1/h1) * (a - b);
   ThermioFlux := y;
   GotoXY(30,14);Write('                    ');
End;

Procedure TunnelFluxCal;
Var
   dxL,dxU,LstackA        : Extended;
   UstackA,TunCoef,E0     : Extended;
   y,z,a,b,c,d,R,L,Tempo  :Extended;
   M,I   :Integer;
Begin
   SubA      := 0;
   SubTun    := 0;
```

```
Tempo     := 0;
TunCoef   := 0;
StopTunnel:= False;
a  := 0;
b:= 0;
NewSlab.Energy      := NewSlab.Energy - dE;
If NewSlab.Energy > 0  Then
Begin
   Repeat
      NewSlab.Llimit := NewSlab.Llimit - 0.2;
      y:= Barshape(NewsLab.Llimit);
      Until y < NewSlab.Energy ;
      NewSlab.Llimit := NewSlab.Llimit + 0.2;
      Repeat
      NewSlab.Llimit := NewSlab.Llimit - 0.01;
      y:= Barshape(NewSlab.Llimit);
   Until y < NewSlab.Energy;
   NewSlab.Llimit := NewSlab.Llimit + 0.005;
End
Else
NewSlab.Llimit := 0;
dxL :=(DBmax - NewSlab.Llimit)/5;
LstackA := 0;
For I := 1 to 5 Do
   Begin
      Tempo := (NewSlab.Llimit + (I *dxL) - (dxL/2));
```

```
    Tempo := Barshape(Tempo);
    Tempo := Tempo - NewSlab.Energy;
    Tempo := Sqrt(Tempo);
    LstackA := LstackA + (dxL * Tempo);
End;
SubA := SubA + LstackA;
Repeat
    NewSlab.Ulimit := NewSlab.Ulimit + 0.5;
    y:= Barshape(NewSlab.Ulimit);
Until y < NewSlab.Energy;
NewSLab.Ulimit := NewSlab.Ulimit - 0.5;
Repeat
    NewSlab.Ulimit := NewSlab.Ulimit + 0.025;
    y:= Barshape(NewSlab.Ulimit);
    Until y < NewSlab.Energy;
    NewSlab.Ulimit := NewSlab.Ulimit - 0.0125;
    dxU := (NewSlab.Ulimit - DBmax)/30;
    UstackA := 0;
    For I := 1 to 30  Do
      Begin
          Tempo := NewSlab.Ulimit - (I *dxU) + (dxU/2);
          Tempo := Barshape(Tempo);
          Tempo := Tempo - NewSlab.Energy;
          Tempo := Sqrt(Tempo);
          UstackA := UstackA + (dxU * Tempo) ;
      End;
```

```
SubA := SubA + UstackA;
Tempo := SubA * Sqrt(RP.Para[3]);
TunCoef := exp(-1.02463 * Tempo);
E0 := NewSlab.Energy + (dE/2);
a  := Ln(1 + exp(-E0/kB/X.X[2]));
b  := Ln(1 + exp(-(E0 +X.X[1])/kB/X.X[2]));
If E0 < 0  Then
  SubTun := (8.7040E24)* Mden * X.X[2] * a  * TunCoef * dE
Else
  Begin
      R := ((8.7040E24) * Mden * X.X[2]) * a  * TunCoef *dE;
      L := ((8.7040E24) * Mden * X.X[2]) * b  * TunCoef * dE;
      SubTun := R - L;
  End;
If MaxSubTun < SubTun Then
MaxSubTun := SubTun;
TunnelFlux := TunnelFlux + SubTun;
OldSLab.Energy := NewSlab.Energy;
OldSlab.Llimit := NewSLab.Llimit;
OldSlab.Ulimit := OldSlab.Ulimit;
If (10000 * SubTun) < MaxSubTun
  Then StopTunnel := True
Else
If (NewSlab.Energy - dE) < (-X.X[1] + RP.Para[2])
Then StopTunnel := True;
GotoXY(30,12); Write('                    ')
```

```
  End;
Begin{Main theoryfunc}
  OldSlab.Energy     := 0;
  OldSlab.Llimit     := 0;
  OldSlab.Ulimit     := 0;
  NewSlab.Energy     := 0;
  NewSlab.Llimit     := 0;
  NewSlab.Ulimit     := 0;
  SubTun             := 0;
  MaxSubTun          := 0;
  SubA               := 0;
  Bmax               := 0;
  DBmax              := 0;
  S            := 0;
  ThermioFlux        := 0;
  TunnelFlux         := 0;

  Barwidth;
  If (S =999999) Or (S > 10000) Then
    TheoryFunc := 999999
  Else
  Begin
    Barmax ;
    OldSlab.Energy   := Bmax;
    OldSlab.Llimit   := DBmax;
    OldSlab.Ulimit   := DBmax;
```

```
      NewSlab.Energy  := Bmax;
      NewSlab.Llimit   := DBmax;
      NewSlab.Ulimit   := DBmax;
      If (S = 999999) Or (Bmax = 999999)  Or ((Bmax - dE) < 0) Then
        TheoryFunc    := 999999
      Else
      Begin
        ThermioFluxCal;
        Repeat
        TunnelFluxCal;
        Until StopTunnel;
        TheoryFunc := 1.6021892E-19 *(ThermioFlux +TunnelFlux);
      End
    End;
End;

Procedure ErrorCalc (Var  A: Parameter);
Var
   I        : Integer;
   Error    : Extended;
   TTheo    : TemTheo;
   ErrFlag  : BooLean;
Begin
   ErrFlag := False;
   For I    := NumData  Downto 1 Do
   Begin
```

```
      TTheo.TT[I] := TheoryFunc(A,Data[I]);
      If(TTheo.TT[I] = 999999)        Then
        Begin
          ErrFlag := True;
          I:= 1;
        End
      Else
      If I < NumData  Then
        TTheo.TT[I] := TTheo.TT[I] / TTheo.TT[NumData];
    End;
    If Not ErrFlag  Then
    Begin
      Error := 0 ;
      For I := 1 to NumData - 1 Do
      Error := Error + Sqr( (TTheo.TT[I] - TDat.TD[I]) / TDat.TD[I]);
      Error := Error /( NumData - 1 );
      A.Err := Sqrt( Error);
    End
    Else
      A.Err := 999999;
End;

Procedure Initialize;
Var
  I        : Integer;
  FileN    : Text;
```

```
  InPutLine     : LineIn;
  FirstParaSet  :  Parameter;

Function Extract (Var StrLine : LineIn) : Extended;
Var
  I          : Integer;
  Strtempo   : String;
  Len        : Integer;
  Code       :  Integer;
  Tempo      :  Extended;
Begin
  I := 0;
  Len := Length ( StrLine);
  Repeat
    I := I + 1;
  Until (Copy( StrLine, I, 1) = ',') or (I > Len);
  Strtempo := Copy ( StrLine, 1, I - 1);
  StrLine := Copy ( StrLine, I + 1, Len - I);
  Val ( StrTempo, Tempo, Code);
  Extract := Tempo;
End;

Procedure AcceptData ( Var X : ExpData);
Var
  Code, J    : Integer;
Begin
```

```
  ReadLn ( FileN, InPutLine);
  For J := 1 to 5 Do
     X.X[J] := Extract ( InPutLine);
     X.Y := Extract ( InPutLine);
End;

Procedure AcceptSet ( Var P : Parameter);
Var
  Code, J    : Integer;
Begin
  ReadLn ( FileN, InPutLine);
  For J := 1 to 6 Do
  P.Para[J] := Extract ( InPutLine);
  P.Mode := 'SAMP';
End;

Procedure ReadData;
Var
  Num   : Integer;
Begin
  Assign ( FileN,'A:\C3_20_01.DAT');
  Reset ( FileN);
  ReadLn ( FileN, NumData);
  For Num := 1 to NumData Do
  AcceptData (Data[Num]);
  ReadLn ( FileN, MaxParaNum);
```

```
  ReadLn ( FileN, NumPoint);
  ReadLn ( FileN, ErrLimit);
  AcceptSet ( ParaStep);
  ReadLn ( FileN, Adjust);
  Close ( FileN);
End;

Procedure PrintSampling;
Var
  I :  Integer;
Begin
  WriteLn ( OutFile);
  For I := 1 to NumPoint Do
  With ParaSet[I] Do
  WriteLn ( OutFile,Para[1]:9:4,Para[2]:9:4,Para[3]:9:4,Para[4]:9:4,Para[5]:9:4,
            Para[6]:9:4,Err:5,Mode:6);
End;

Procedure PrintData;
Var
  I :  Integer;
Begin
  WriteLn ( OutFile);
  WriteLn ( OutFile);
  WriteLn ( OutFile, ' Stepping Vector :');
  WriteLn ( OutFile,'Para 1':9,'Para 2':9,'Para 3':9,'Para 4':9,'Para 5':9,'Para 6':9);
```

```
    With ParaStep Do
     WriteLn (OutFile,Para[1]:9:4,Para[2]:9:4,Para[3]:9:4,
                 Para[4]:9:4,Para[5]:9:4,Para[6]:9:4);
    WriteLn ( OutFile);
End;

Begin { Main control of INITIALIZE }
    SteppingNum := 0 ;
    ReadData;
    ErrMinPara := ParaSet[1];
    If ErrMinPara.Err <= ErrLimit Then
    Begin
       Found := True;
       Cause := 'Sampling point error number lower than error limit. ';
    End;
    GotoXY (10,3); Write ('                    ');
End;
{ End of INITIALIZE }

Procedure FitParameter;
Var
    Success, GradEnd      : Boolean;
    StopExtra, ErrorLess  : Boolean;
    OldPiVot : Parameter;

Procedure GradFind;
```

```
Var
   FinalPoint :  Parameter;
   Point      :  Parameter;
   RealP      :  Parameter;
   I,Sign,D   :  Integer;
   N1,N2,L    :  Extended;
Begin
   GotoXY(3,7); WriteLn('GRADIENT');
   FinalPoint := NullVector;
   GradNo  := GradNo + 1;
   For I := 1 to MaxParaNum Do
   Begin
      L     := 1;
      Point := ErrMinPara;
      If GradNo = 1  Then  D := 10
      Else
      If GradNo = 2  Then  D := 2
      Else
      D := 1;
      Point.Para[I] := ErrMinPara.Para[I] + (ParaStep.Para[I]*D);
      Transform(Point,RealP);
      ErrorCalc(RealP);
      N1  := RealP.Err;
      If N1 > ErrMinPara.Err  Then
      Begin
         Point.Para[I] := ErrMinPara.Para[I] - (ParaStep.Para[I]*D);
```

```
    Transform(Point,RealP);
    ErrorCalc(RealP);
    N2 := RealP.Err;
    If (N2 < ErrMinPara.Err)  Then  L := -1;
  End;
  PerErrStep:= (RealP.Err - ErrMinPara.Err) * 100 / ErrMinPara.Err;
  If RealP.Err = 999999 Then
    FinalPoint.Para[I] := 0
  Else
  If (N1 > ErrMinPara.Err) And (N2 > ErrMinPara.Err)  Then
    FinalPoint.Para[I] := 0
  Else
  If ErrMinPara.Err = RealP.Err  Then
    FinalPoint.Para[I] := 0
  Else
    FinalPoint.Para[I] := (ParaStep.Para[I]*D)* L ;
End;
If VSize(FinalPoint) = 0  Then
  Success := False
Else
Begin
  Add (FinalPoint, ErrMinPara, NewPoint);
  NewPoint.Mode := 'GRAD';
  Transform(NewPoint,RealNewPoint);
  ErrorCalc(RealNewPoint);
  NewPoint.Err := RealNewPoint.Err;
```

```
      If NewPoint.Err > ErrMinPara.Err  Then
         Begin
            Mult (0.3, FinalPoint,FinalPoint);
            Add (FinalPoint, ErrMinPara,NewPoint);
            NewPoint.Mode := 'GRAD';
            Transform(NewPoint,RealNewPoint);
            ErrorCalc(RealNewPoint);
            NewPoint.Err := RealNewPoint.Err;
         End
      Else
      If NewPoint.Err < ErrMinPara.Err Then
         Begin
         OldEMin := ErrMinPara;
         ErrMinPara := NewPoint;
         Success := True;
         End;
   End;
   GotoXY(3,7); WriteLn('                              ');
End;

Procedure SmallStepping;
Var
   Sign    : Integer;
   K       : Integer;
   P       : Extended;
Begin
```

```
GotoXY (3,9); Write ('SMALL STEPPING');
ErrorLess := False;
For K :=(SteppingNum + 1) to (SteppingNum + MaxParaNum)  Do
For Sign := 1 to 2 Do
   If Not ErrorLess    Then
   Begin
      If K < (MaxParaNum + 1 ) Then
         SteppingNum := K
      Else
         SteppingNum := K - MaxParaNum;
      NewPoint := ErrMinPara;
      NewPoint.Para[SteppingNum] := ErrMinPara.Para[SteppingNum] +
      ( Sign * 2 - 3 ) * ParaStep.Para[SteppingNum];
      NewPoint.Mode := 'STEP';
      GotoXY(15,9);
      Write('No:',SteppingNum,' ',Sign*2-3,'*',ParaStep.Para[SteppingNum]);
      ErrorCalc(RealNewPoint);
      NewPoint.Err := RealNewPoint.Err;
      P := (NewPoint.Err - ErrMinPara.Err) * 100 / ErrMinPara.Err;
      If NewPoint.Err < ErrMinPara.Err Then
        Begin
           ErrorLess := True;
           OldEMin   := ErrMinPara;
           ErrMinPara := NewPoint;
           K   := SteppingNum + MaxParaNum;
           Sign  := 2;
```

```
            End;
        End;
    GotoXY(3,9);Write ('                                        ');
End;

Procedure Extrapolate;
Var
    I     : Integer;
    Factor, EFactor   : Boolean;
    AFactor,Valve   : Extended;
    ErrFlag :  Boolean;
    ExAdjust  :  Extended;
Begin
    GotoXY (3,5); Write ('EXTRAPOLATE');
    If GradNo = 1  Then  ExAdjust  := Adjust * 5
    Else
    If GradNo = 2  Then  ExAdjust  := Adjust
    Else
        ExAdjust := Adjust / 3 ;
    NewPoint := NullVector;
    If OldEMin.Err = ErrMinPara.Err  Then
        StopExtra := True
    Else
    Begin
        EFactor := ErrMinPara.Err /(OldEMin.Err - ErrMinPara.Err);
        For I := 1 to MaxParaNum Do
```

```
  Begin
    ErrFlag:= False;
    If ErrMInPara.Para[I] = 0 Then
      ErrFlag := True
    Else
    Begin
    AFactor := (ErrMinPara.Para[I] - OldEMin.Para[I])/ErrMinPara.Para[I];
      AFactor := ABS(AFactor);
      Factor := tanh(EFactor * AFactor);
    End;
  End;
NewPoint.Mode := 'EXTRA';
Transform(NewPoint,RealNewPoint);
ErrorCalc(RealNewPoint);
GotoXY(15,17);Write('Error Now :',NewPoint.Err *100:15:10,' %');
If NewPoint.Err >= ErrMinPara.Err Then
  StopExtra := True
Else
Begin
  OldEMin := ErrMinPara;
  ErrMinPara := NewPoint;
 If Newpoint.Err < ErrMinPara.Err  Then
Begin
    StopExtra := True;
    Found := True;
    Cause := ' Error less than Error Limit : ';
```

```
        End;
      End;
    End;
    GotoXY (3,5); Write ('                    ');
End;

Begin { Main Control of FITPARAMETER }
    Success := False;
    If GradNo  < 4  Then
    Begin
    GradFind;
    If Success Then
    Begin
      StopExtra := False;
      Repeat
        Extrapolate;
      Until StopExtra;
    End
    End
    Else
    Begin
      ErrorLess := False;
      SmallStepping;
      If ErrorLess Then
      Begin
        StopExtra := False;
```

```
        Repeat
          Extrapolate;
        Until StopExtra;
      End
      Else
      Begin
        Found := True;
        Cause := 'Finding minimun but greater than ErrLimit';
      End;
    End;
End;

Procedure PrintResult;
Var
  I        : Integer;
  Par      : Parameter;
  Tempo,A,J    : Extended;
  C : ExpData;
Begin
  WriteLn ( OutFile,' #C3/15(0-1) Ni/p-CIS');
  WriteLn ( OutFile,'Para 1':9,'Para 2':11,'Para 3':11,'Para 4':10,'Para 5':9,
            'Para ':10,'Error':8);
  With ErrMinPara Do
  WriteLn ( OutFile,Para[1]:5:4,'    ' ,Para[2]:5:4,'    ',Para[3]:6:5,'    ',
    Para[4]:5:4,'  ',Para[5]:5:4,'  ', Para[6]:6:5,'  ',(Err*100):4:2,' %');
  Transform(ErrMinPara,Par);
```

```
  Tempo := 0;
  WriteLn (OutFile);
  WriteLn (OutFile,NumData,' data points');
  WriteLn (OutFile);
  For I := 1 to NumData  Do
  Begin
    J := TheoryFunc(Par,Data[I]);
    A := Data[I].Y / J;
    WriteLn(OutFile,I:2,'  ',Data[I].X[1]:4:3,'   ',Data[I].X[2]:3:1,'   ',
    Data[I].Y:6:5,'    ',J:12:6,'      ',A:12:11);
    Tempo := Tempo + A;
  End;
End;
Procedure InitPrinter;
Begin
  Write ( OutFile, #27,'@',#27,'M',#27,'0');
End;

{************************************************************}
 {*                    MAIN  CONTROL                        *}
{************************************************************}
Begin
  Clrscr;
  Write('Input Filename to save : '); ReadLn(fn);
  Assign  (OutFile, fn);
  ReWrite (OutFile);
```

```
    WriteLn (OutFile, fn);
    InitPrinter;
    ClrScr;
    Found := False;
    Initialize;
    GradNo   := 0;
    Repeat
      GotoXY(15,17); Write ('Error  Now: ', ErrMinPara.Err*100:15:10,' %');
      FitParameter;
      GotoXY(3,19);
      Write ('ErrMinPara=',ErrMinPara.Para[1],' ',ErrMinPara.Para[2],' ',
                ErrMinPara.Para[3],' ',ErrMinPara.Para[4],' ',
                ErrMinPara.Para[5],' ',ErrMinPara.Para[6]  );
    Until Found;
    PrintResult;
    Sound(2000); Delay(100); NoSound; Delay(100);
    Sound(2000); Delay(100); NoSound; Delay(100);
    Sound(2000); Delay(100); NoSound; Delay(100);
    Sound(1500); Delay(300); NoSound;
    GotoXY(15,25); Write (' FOUND !!!');
    Close (OutFile);
End.
```

DATA FILE.

15

0.283,107.1,0,0,0,0.00001,

0.255,137.5,0,0,0,0.00001,

0.174,164.0,0,0,0,0.00001,

0.070,188.0,0,0,0,0.00001,

0.363,107.1,0,0,0,0.00003,

0.341,137.5,0,0,0,0.00003,

0.267,164.0,0,0,0,0.00003,

0.148,188.0,0,0,0,0.00003,

0.486,107.1,0,0,0,0.00010,

0.456,137.5,0,0,0,0.00010,

0.393,164.0,0,0,0,0.00010,

0.283,188.0,0,0,0,0.00010,

0.144,210.6,0,0,0,0.00010,

0.591,107.1,0,0,0,0.00030,

0.567,137.5,0,0,0,0.00030,

0.519,164.0,0,0,0,0.00030,

0.285,210.6,0,0,0,0.00030,

0.138,232.2,0,0,0,0.00030,

0.720,107.1,0,0,0,0.00103,

0.705,137.5,0,0,0,0.00103,

0.657,164.0,0,0,0,0.00103,

0.577,188.0,0,0,0,0.00103,

0.472,210.6,0,0,0,0.00103,

0.316,232.2,0,0,0,0.00103,

0.162,252.9,0,0,0,0.00103,

0.948,107.1,0,0,0,0.00495,

0.896,137.5,0,0,0,0.00495,

0.833,164.0,0,0,0,0.00495,

0.770,188.0,0,0,0,0.00495,

0.704,210.6,0,0,0,0.00495,

0.624,232.2,0,0,0,0.00495,

0.484,252.9,0,0,0,0.00495,

0.322,273.0,0,0,0,0.00495,

0.155,292.6,0,0,0,0.00495,

0.422,188.0,0,0,0,0.00030,

6

1

5,-20,1,5,10,5,

0.03

0.005,0.05,0.004,0.05,0.1,0.01,

3

Curriculum Vitae

Mr. Teerapunt Santitewegul was born on February 10, 1954 in Ubolratchatani province. He received his B. Sc. from Srinakarinwirot University at Prasarnmit in 1981 and M.Sc.from Chulalongkorn University in 1984.