```
        cpkp  = 2*Dp*Nv / (y_top + y_bottom) / y_top;
        xpkp  = (jpl.phi - phi) / VT+(jpl.chi-chi)/q/VT+(jpl.Eg-Eg)/q/VT;
        bpkp  = fabs(xpkp) < 1e-10 ? 1+0.5*xpkp : xpkp / (exp(xpkp) - 1);
        aijpl = cpkp * bpkp;

        cpkm  = 2*Dp*Nv / (y_top + y_bottom) / y_bottom;
        xpkm  = (jml.phi - phi) / VT+(jml.chi-chi)/q/VT+(jml.Eg-Eg)/q/VT;
        bpkm  = fabs(xpkm) < 1e-10 ? 1+0.5*xpkm : xpkm / (exp(xpkm) - 1);
        aijml = cpkm * bpkm;

        aij = aiplj + aimlj + aijpl + aijml;

        r = 1. / (Tn*ni*(Nv*V*exp(-phi/VT)*exp(-(chi+Eg)/q/VT)+1)
                + Tp*ni*(Nc*U*exp(phi/VT)*exp(chi/q/VT)+1));
        G = ni * U * r/ephp;
        f = -ni * r/ephp;
        F = aiplj*ipl.V + aimlj*iml.V + aijpl*jpl.V + aijml*jml.V - f;

        V = F / (aij+G);

        p =ni* Nv* V * exp(-phi/VT)*exp(-(chi+Eg)/q/VT);

        return V;
}

// method to calculate the electron quasi-Fermi level
double material::ret_Fermi_leveln (double px) {

        double Efi, Ef;
        double me = (0.067 + 0.083*px) * 0.91e-30;
        double mlh = 0.087 + 0.063*px;
        double mhh = 0.62 + 0.14*px;
        double templ = sqrt(mlh*mlh*mlh) + sqrt(mhh*mhh*mhh);
        double mh = cbrt(templ*templ) * 0.91e-30;

        const double CHI = 4.07*q ;
        const double EGAP = 1.4224*q;

        // mh=0.45*9.108e-31;
        // me=0.067*9.108e-31;

        VT=0.0259;
        q = 1.6e-19;
        Efi=-phi*q-(chi-CHI)-(Eg+EGAP)/2+3/4*VT*q*log(mh/me);
        Ef=Efi+q*(phi+chi/q+VT*log(Nc)+VT*log(U));

        return Ef;

}

// method to calculate the hole quasi-Fermi level
double material::ret_Fermi_levelp (double px) {

        double Efi,Ef;


        double me = (0.067 + 0.083*px) * 0.91e-30;
        double mlh = 0.087 + 0.063*px;
        double mhh = 0.62 + 0.14*px;
        double templ = sqrt(mlh*mlh*mlh) + sqrt(mhh*mhh*mhh);
        double mh = cbrt(templ*templ) * 0.91e-30;

        const double CHI = 4.07*q ;
        const double EGAP = 1.4224*q;

        // mh=0.45*9.108e-31;
        // me=0.067*9.108e-31;

        VT=0.0259;
        q = 1.6e-19;
        Efi=-phi*q-(chi-CHI)-(Eg+EGAP)/2+3/4*VT*q*log(mh/me);
        Ef=Efi+q*(phi+chi/q+Eg/q-VT*log(Nv)-VT*log(V));

        return Ef;
}

// method to calculate the electron density in the quantum well
```

```
material material::cal_qwn (double Efermi,double size,double perc) {

        int num,i;
        double tmp = 0;
        VT=0.0259;
        q = 1.6e-19;
        double V0, gam, me, htagl, mh, lf, rg, a_temp, y;
        double Energ[30];
        double a[30];

        const double CHI = 4.07*q;
        const double EGAP = 1.4224*q;

        mh=0.45*9.108e-31;
        me=0.067*9.108e-31;
        htagl=6.625e-34/(2*3.14159);
        V0=0.7482*(perc)*q;
        gam=sqrt(2*(me/htagl)*V0*size*size/htagl);

        num=0;

        do {
            lf=0;
            rg=1;
            num++;

            do {
                a_temp=(lf+rg)/2;
                y=gam*a_temp-((num-1)*3.14159+2*acos(a_temp));

                if (y<0)
                    lf=a_temp;
                else
                    rg=a_temp;
            } while((fabs(y)>1e-5)&&((1-lf)>1e-8));

            Energ[num]=V0*a_temp*a_temp;
            a[num]=a_temp;

        } while(((1-lf)>1e-8 ) && (num<30));

        n=0;

        for(i=1;i<num;i++) {
            tmp = log(1+exp((Efermi+phi*q+(chi+CHI)-Energ[i])/(VT*q)));
            if (tmp==0)
                tmp = exp((Efermi+phi*q+(chi+CHI)-Energ[i])/(VT*q));
            n +=  7.232e+15*tmp;
        }

        U = exp(-phi/VT)*exp(-chi/q/VT)* n /(ni*Nc);

        return *this;
}

// method to calculate the hole density in the quantum well
material material::cal_qwp (double Efermi,double size,double perc) {

        int num, i;
        double tmp;

        VT = 0.0259;
        q = 1.6e-19;

        double V0, gam, me, htagl, mh, lf, rg, a_temp, y;
        double Energ[30];
        double a[30];

        const double CHI = 4.07*q;
        const  double EGAP = 1.4224*q;

        mh = 0.45*9.108e-31;
        me = 0.067*9.108e-31;
        htagl = 6.625e-34/(2*3.14159);
        V0 = (1.247-0.7482)*(perc)*q;
        gam = sqrt(2*mh*V0*size*size/(htagl*htagl));
        num = 0;
```

```
        do {
            lf=0;
            rg=1;
            num++;

            do {
                a_temp=(lf+rg)/2;
                y=gam*a_temp-((num-1)*3.14159+2*acos(a_temp));

                if (y<0)
                    lf=a_temp;
                else
                    rg=a_temp;

            } while((fabs(y)>1e-5)&&((1-lf)>1e-8));

            Energ[num]=V0*a_temp*a_temp;
            a[num]=a_temp;

        } while(((1-lf)>1e-8)&&(num<30));

        p = 1.0e-10;

        for(i=1;i<num;i++) {
            tmp=log(1+exp((-phi*q-(chi+CHI)-(Eg+EGAP)-Energ[i]-Efermi)/(VT*q)));
            if (tmp==0)
                tmp = exp((-phi*q-(chi+CHI)-(Eg+EGAP)-Energ[i]-Efermi)/(VT*q));

            p +=  4.857e+16*tmp;
        }

        V = p/(ni* Nv) * exp(phi/VT) * exp((chi+Eg)/q/VT);

        return *this;
}

// method to calculate the filed
material material::cal_field(material& ip1, material& im1, material& jp1,
                             material& jm1) {

        double dfx,dfy;
        dfx=-(ip1.phi-phi)/x_right/2-(phi-im1.phi)/x_left/2;
        dfy=-(jp1.phi-phi)/y_top/2-(phi-jm1.phi)/y_bottom/2;
         field=sqrt(dfx*dfx + dfy*dfy);
         return *this;
}

// method to set the density of the material
material material::setDensity( double denst) {

        double temp;

        density = denst;

        if (density < 0) {
            temp = fabs(density) / (2 * ni);
            phi0 = -log(temp + sqrt((temp*temp) + 1));
        }
        else {
            temp = density / (2 * ni);
            phi0 = log(temp + sqrt((temp*temp) + 1));
        }

        phi = phi_applied + phi0*VT;
        n   = ni* Nc * U * exp(phi/VT) * exp(chi/VT/q);
        p   = ni* Nv * V * exp(-phi/VT) * exp(-(chi +Eg)/VT/q);

        return *this;
}

// method to set the applied voltage
material material::setAppliedVolt(double phi_appl) {

        phi_applied = phi_appl;
        phi         = phi_applied + phi0*VT;
        U           = exp(-phi_applied/VT);
```

```
        V           = exp(phi_applied/VT);

        n = ni* Nc * U * exp(phi/VT) * exp(chi/VT/q);
        p = ni* Nv * V * exp(-phi/VT) * exp(-(chi +Eg)/VT/q);

        return *this;
}

// method to set the lifetime
material material::set_time( double tau) {

        Tp=tau;
        Tn=tau;
        return *this;
}

// method to set the diffusion coefficients
material material::set_Diffusivity() {

        double mobn0,mobp0;
        mobn0 = (7200/(pow(1+5.51e-17*fabs(density),0.233)))
                *pow(300/Temperature, 2.3);
        mobp0 = (380/(pow(1+3.17e-17*fabs(density),0.266)))
                *pow(300/Temperature, 2.7);
        Dn=VT*(mobn0+7.7e06*pow(field,3)/2.56e14)/(1+pow(field/4e03,4));
        Dp=VT*mobp0/(1+mobp0*field/1.5e07);

        return *this;
}

// method to set the steps (h, k)
material material::set_step( double size1, double size2) {

        x_left = size1;
        x_right = size1;
        y_top = size2;
        y_bottom = size2;

        return *this;
}

// declaration for the equal sign
material material::operator=(const material &input) {

        phi         = input.phi;
        U           = input.U;
        V           = input.V;
        phi0        = input.phi0;
        phi_applied = input.phi_applied;
        density     = input.density;
        ni          = input.ni;
        n           = input.n;
        p           = input.p;
        q           = input.q;
        eps         = input.eps;
        Eg          = input.Eg;
        chi         = input.chi;
        VT          = input.VT;
        Temperature = input.Temperature;
        x_right     = input.x_right;
        x_left      = input.x_left;
        y_top       = input.y_top;
        y_bottom    = input.y_bottom;
        Tp          = input.Tp;
        Tn          = input.Tn;
        Dp          = input.Dp;
        Dn          = input.Dn;
        Nc          = input.Nc;
        Nv          = input.Nv;

        return *this;
}

// declaration for the object when used with the cout function.
ostream& operator<<(ostream &stream, const material& output) {

        stream << "Potential = " << output.phi << "\n";

          :
```

```
        stream << "U = " << output.U << "\n";
        stream << "V = " << output.V << "\n";

        return stream;
}
```

- **Main Program**

```
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <stdlib.h>
#include "material.h"

double eps_AlGaAs(double x=0);          // return eps of AlGaAs
double ni_AlGaAs(double x=0);           // return ni of AlGaAs
double chi_AlGaAs(double x=0);          // return chi of AlGaAs
double Nc_AlGaAs(double x=0);           // return Nc of AlGaAs
double Nv_AlGaAs(double x=0);           // return Nv of AlGaAs
double Eg_AlGaAs(double x=0);           // return Eg of AlGaAs

void main(void) {

        //**** declarations ****//
        const int max_i = 122;
        const int max_j = 3;
        const double density_of_n = 5e16;
        const double density_of_i = 1e15;
        const double density_of_p_contact = -1e17;
        const double density_of_barrier = 5e17;
        const double density_of_QW = 5e14;
        const double q = 1.6e-19;
        const double eps_GaAs = 13.18 * 8.85e-14;
        const double CHI =  4.07*q ;
        const double EGAP = 1.4224*q;
        const double NCR =2.5e19*0.067;
        const double NVR =2.5e19*0.48;
        const double ni_GaAs = 5915256.5520340;
        const double VT = 0.026;

        double Ni_AlGaAs;
        int i,j;
        double vectx[max_i+2];
        double x;
        double vn,vp;
        double vj;
        material mat[max_i+2][max_j+2];
        ofstream ddmdata;

// initialise the GaAs
        material *temp_GaAs = new material(0,0,ni_GaAs,eps_GaAs,0,0,1,1,300);

        vn = 0.0;       // applied voltage at n-doped region
        vp = 0.0;       // applied voltage at p-doped region

        for (i=1; i<=max_i; i++)              // looping for the whole device
            for (j=1; j<=max_j; j++) {

                if (i>=1 && i<=40) {   // set the dimension and parameters of n region

                    x = 0.1;
                    vectx[i]=x;        // Al composition in AlGaAs of n region

        // calculate the ni
                     Ni_AlGaAs = sqrt(Nc_AlGaAs(x)*Nv_AlGaAs(x))
                                 * exp(-((Eg_AlGaAs(x)/2)/q)/VT);

        // initialise the dummy
                     material *temp_AlGaAs = new material(0,0,Ni_AlGaAs,
                                                          eps_AlGaAs(x),
                                                          Eg_AlGaAs(x)-EGAP,
                                                          chi_AlGaAs(x)-CHI,
                                                          Nc_AlGaAs(x)/NCR,
                                                          Nv_AlGaAs(x)/NVR);
                     mat[i][j] = *temp_AlGaAs; // assign the material
                     mat[i][j].setDensity(density_of_n);
```

```
            mat[i][j].setAppliedVolt(vn + i*(vp-vn)/max_i+1);
            mat[i][j].set_Diffusivity();

            delete temp_AlGaAs;         // remove the dummy

        }

        if ((i>=41) && (i<=122))        // set the rest of the device
            if ((i>=61) && (i<=62)) { //set quantum well
                vectx[i]=0;
                mat[i][j] = *temp_GaAs;
                mat[i][j].setDensity(density_of_QW);
                mat[i][j].setAppliedVolt(vn + i*(vp-vn)/max_i+1);
                mat[i][j].set_Diffusivity();
            }
            else {                      // set i region
                x =0.3;
                vectx[i]=x;
                Ni_AlGaAs = sqrt(Nc_AlGaAs(x)*Nv_AlGaAs(x))
                            * exp(-((Eg_AlGaAs(x)/2)/q)/VT);

                material *temp_AlGaAs = new material(0,0,Ni_AlGaAs,
                                                     eps_AlGaAs(x),
                                                     Eg_AlGaAs(x)-EGAP,
                                                     chi_AlGaAs(x)-CHI,
                                                     Nc_AlGaAs(x)/NCR,
                                                     Nv_AlGaAs(x)/NVR);
                mat[i][j] = *temp_AlGaAs;
                if ((i>=83) && (i<=122))        // set p region
                        mat[i][j].setDensity(density_of_p_contact);
                else
                        mat[i][j].setDensity(density_of_i);
                mat[i][j].setAppliedVolt(vn + i*(vp-vn)/max_i+1);
                mat[i][j].set_Diffusivity();

                delete temp_AlGaAs;
            }
    }

// set ohmic contacts at n and p regions
x =0.1; // n-region
Ni_AlGaAs = sqrt(Nc_AlGaAs(x)*Nv_AlGaAs(x))
            * exp(-((Eg_AlGaAs(x)/2)/q)/VT);
material *temp_AlGaAs1 = new material(0,0,Ni_AlGaAs,eps_AlGaAs(x),
                                      Eg_AlGaAs(x)-EGAP,
                                      chi_AlGaAs(x)-CHI,
                                      Nc_AlGaAs(x)/NCR,
                                      Nv_AlGaAs(x)/NVR);
x =0.3; // p-region
Ni_AlGaAs = sqrt(Nc_AlGaAs(x)*Nv_AlGaAs(x))
            * exp(-((Eg_AlGaAs(x)/2)/q)/VT);
material *temp_AlGaAs = new material(0,0,Ni_AlGaAs,eps_AlGaAs(x),
                                     Eg_AlGaAs(x)-EGAP,
                                     chi_AlGaAs(x)-CHI,
                                     Nc_AlGaAs(x)/NCR,
                                     Nv_AlGaAs(x)/NVR);
for (j=0; j<max_j+2; j++) {   // reset boundaries in y direction
    mat[0][j] = *temp_AlGaAs1;
    mat[0][j].setDensity(density_of_n);
    mat[0][j].setAppliedVolt(vn);
    mat[0][j].set_Diffusivity();

    mat[max_i+1][j] = *temp_AlGaAs;
    mat[max_i+1][j].setDensity(density_of_p_contact);
    mat[max_i+1][j].setAppliedVolt(vp);
    mat[max_i+1][j].set_Diffusivity();
}

delete temp_GaAs;
delete temp_AlGaAs;
delete temp_AlGaAs1;

for (i=1; i<=max_i; i++) {    // reset boundaries in x direction

    mat[i][0] = mat[i][2];
    mat[i][max_j+1] = mat[i][max_j-1];
```

```
}

// main loop

double old_sum;
double sum = 0;
int  round = 1;
double temp_fermi=0;

for (i=1; i<=max_i; i++)
    for(j=1; j<= max_j; j++)
                    sum += mat[i][j].potential();

do {    // loop until converge

    old_sum = sum;
    sum = 0;

    for (i=1; i<=max_i; i++) {          // reset boundaries
        mat[i][0] = mat[i][2];
        mat[i][max_j+1] = mat[i][max_j-1];
    }

    for (i=1; i<=max_i; i++)  // computing phi
        for(j=1; j<= max_j; j++)
                sum += mat[i][j].cal_phi(mat[i+1][j],mat[i-1][j],
                                         mat[i][j+1],mat[i][j-1]);

    for (i=1; i<=max_i; i++)  // computing U
        for(j=1; j<=max_j; j++) {
            mat[i][j].cal_U(mat[i+1][j],mat[i-1][j],mat[i][j+1],
                            mat[i][j-1]);
            mat[max_i-i+1][max_j-j+1].cal_U(mat[max_i-i][max_j-j+1],
                                            mat[max_i-i+2][max_j-j+1],
                                            mat[max_i-i+1][max_j-j],
                                            mat[max_i-i+1][max_j-j+2]);
        }

    for (i=1; i<=max_i; i++)  // computing V
        for(j=1; j<=max_j; j++) {
            mat[i][j].cal_V(mat[i+1][j],mat[i-1][j],mat[i][j+1],
                            mat[i][j-1]);
            mat[max_i-i+1][max_j-j+1].cal_V(mat[max_i-i][max_j-j+1],
                                            mat[max_i-i+2][max_j-j+1],
                                            mat[max_i-i+1][max_j-j],
                                            mat[max_i-i+1][max_j-j+2]);
        }

    for (i=1; i<=max_i; i++)  // computing field
        for(j=1; j<=max_j; j++) {
            mat[i][j].cal_field(mat[i+1][j],mat[i-1][j],mat[i][j+1],
                                mat[i][j-1]);
            mat[i][j].set_Diffusivity();
        }

    for (i=61; i<=62; i++)     // computing quantum wells
        for(j=1; j<=max_j; j++) {
            temp_fermi=mat[i][j].ret_Fermi_leveln(0.3);
            mat[i][j].cal_qwn(temp_fermi,15e-8,0.3);
            temp_fermi=mat[i][j].ret_Fermi_levelp( 0.3);
            mat[i][j].cal_qwp(temp_fermi,15e-8,0.3);
        }

    if ((round%5000) == 0) {  // intermediate writing data every 5k iterations

        ddmdata.open("d60nobi_2.dat");
        if (!ddmdata) {
            cout << "Error open 'd60nobi_2.dat'.\n";
            exit(1);
        }

        // write header
        ddmdata << "# this is data of round : " << round << "\n";
        ddmdata << "# |old_sum - sum| = " << fabs(old_sum-sum) << "\n"
                << "\n";

        // write data
```

```
            for (i=1; i<=max_i; i++) {
                for(j=1; j<=max_j; j++)
                    ddmdata << i << "  "
                            << j << "  "
                            << -mat[i][j].potential()
                               -CHI/q-mat[i][j].CHI()/q << "  "
                            << -mat[i][j].potential()
                               -CHI/q-mat[i][j].CHI()/q
                               -mat[i][j].EG()/q-EGAP/q << "  "
                            << mat[i][j].ret_n() << "  "
                            << mat[i][j].ret_p() << "  "
                            << mat[i][j].potential() << "  "
                            << mat[i][j].ret_Fermi_leveln(vectx[i])/q
                            << "  "
                            << mat[i][j].ret_Fermi_levelp(vectx[i])/q
                            << "  "
                            << -mat[i][j].potential()
                               -(mat[i][j].CHI()+CHI)/q
                               -(mat[i][j].EG()+EGAP)/q/2
                               +3/4*VT*log(0.45*9.108e-31/0.067*9.108e-31)
                            <<"\n";
                ddmdata << "\n";
            }
            ddmdata << "\n\n";
            ddmdata.close();
        }

        round++;

    } while(fabs(old_sum-sum) > 1e-5);    // within the error?

    //**** end cycle ****

    ddmdata.open("d60nobi_2.dat");        // writing data

    if (!ddmdata) {
        cout << "Error open 'd60nobi_2.dat'.\n";
        exit(1);
    }

    // header
    ddmdata << "# Finished running, round = " << round << "\n";
    ddmdata << "# |old_sum - sum| = " << fabs(old_sum-sum) << "\n\n";

    // data
    for (i=1; i<=max_i; i++) {
        for(j=1; j<=max_j; j++)
            ddmdata << i << "  "
                    << j << "  "
                    << -mat[i][j].potential() - CHI/q-mat[i][j].CHI()/q
                    << "  "
                    << -mat[i][j].potential() - CHI/q-mat[i][j].CHI()/q
                       -mat[i][j].EG()/q - EGAP/q << "  "
                    << mat[i][j].ret_n() << "  "
                    << mat[i][j].ret_p() << "  "
                    << mat[i][j].potential() << "  "
                    << mat[i][j].ret_Fermi_leveln(vectx[i])/q << "  "
                    << mat[i][j].ret_Fermi_levelp(vectx[i])/q << "  "
                    << -mat[i][j].potential() - (mat[i][j].CHI()+CHI)/q
                       - (mat[i][j].EG()+EGAP)/q/2
                       + 3/4*VT*log(0.45*9.108e-31/0.067*9.108e-31) <<"\n";

        ddmdata << "\n";
    }

    ddmdata << "\n\n";
    ddmdata.close();
}

// function declaration

double eps_AlGaAs( double x) {

    if ((x>=0) && (x<=1))
        return 8.85e-14 * (13.18 - 3.12*x);
    else {
        cout << "x must be between 0.0-1.0\n";
```

```
            exit(1);
        }
}

double Eg_AlGaAs( double x) {

        double Eg;
        if ((x >= 0) && (x <= 1)) {
            if ( x <=0.45 )
                Eg = (1.4224 - 1.247*x) * 1.6e-19;
            else
                Eg = (1.900 + 0.125*x + 0.143*x*x)*1.6e-19;
        }
        else {
            cout << "x must be between 0.0-1.0\n";
            exit(1);
        }

        return Eg;

}

double chi_AlGaAs( double x) {

        double chi;
        if ((x>=0) && (x<=0.45))
            chi = (4.07 -0.7482 * x)* 1.6e-19;
        else {
            cout << "x must be between 0.0-0.45\n";
            exit(1);
        }

        return chi;

}

double Nc_AlGaAs(double x) {

        double Nc;
        if ((x>=0) && (x<=0.45))
            Nc = 2.5e19*(0.067+0.083*x);
        else {
            cout << "x must be between 0.0-0.45\n";
            exit(1);
        }

        return Nc;

}

double Nv_AlGaAs(double x) {

        double Nv;
        if ((x>=0) && (x<=0.45))
            Nv = 2.5e19*(0.48+0.31*x);
        else {
            cout << "x must be between 0.0-0.45\n";
            exit(1);
        }

        return Nv;

}
```

ภาคผนวก ง.

Class และ Main Program สำหรับโปรแกรมบน PC

เนื่องจากโปรแกรมบน PC เป็นโปรแกรมชนิด Graphical User Interface (GUI) ซึ่งประกอบด้วยส่วนที่เป็น Resources ต่างๆ จำนวนมาก ดังนั้นจึงขอแสดงเฉพาะส่วนของ Class ที่เกี่ยวข้องกับการหาผลเฉลย

- **Class header (Material.h)**

```
// Material.h: interface for the CMaterial class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_MATERIAL_H__DFEF2D5F_B71A_4201_B479_C270EABE9B2A__INCLUDED_)
#define AFX_MATERIAL_H__DFEF2D5F_B71A_4201_B479_C270EABE9B2A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMaterial
{
protected:
    double     phi;                    //      potential
        double  field;                  //      electric field
        double  U;                              //      special variable for electrons
        double  V;                              //      special variable for holes
        double  n;                              //      electron concentration
        double  p;                              //      hole concentration

        double  density;                //      density of the material
        double  phi0;                   //      initial potential
        double  phi_applied;    //      applied potential
        double  ni;                             //      ni of the material
        double  q;                              //      electron charge
        double  eps;                    //      material permitivity
        double  Eg;                             //      material bandgap
        double  chi;                    //      electron affinity of the material
        double  VT;                             //      kT/q
        double  Temperature;    //      Temperature
        double  x_right;                //      hi,j
        double  x_left;                 //      hi-1,j
        double  y_top;                  //      ki,j
        double  y_bottom;               //      ki,j-1
        double  Nc;                             //      effective density of states in
 conduction band
        double  Nv;                             //      effective density of states in
 valence band
        double  Tp;                             //      hole lifetime
        double  Tn;                             //      electron lifetime
        double  Dn;                             //      electron diffusion coefficient
        double  Dp;                             //      hole diffusion coefficien
        double  me;                             //      electron mass
        double  mh;                             //      hole mass

 public:
        // default constructor
        CMaterial();

        // main constructor
        CMaterial(double density_p, double phi_applied_p = 0.0,
                double ni_p = 5915256.5520340, double eps_p = 1.16643e-12,
                double Eg_p = 0,double chi_p =0,
                double Nc_p = 1, double Nv_p = 1,
                double Temperature_p = 300,  double q_p = 1.602e-19,
                double VT_p = 0.026 , double x_right_p = 5e-7,
                double x_left_p = 5e-7, double y_top_p = 5e-7,
                double y_bottom_p = 5e-7, double Tp_p = 1e-8,
                double Tn_p = 1e-8, double Dp_p = 10.4,double Dn_p = 221,
                double me_p = 6.103365e-32, double mh_p = 4.099275e-31);
```

```
        //Set value for variable  in inputdata file.
        void SetValue(double density_p, double phi_applied_p = 0.0,
                double ni_p = 5915256.5520340, double eps_p = 1.16643e-12,
                double Eg_p = 0,double chi_p =0,
                double Nc_p = 1, double Nv_p = 1,
                double Temperature_p = 300,  double q_p = 1.602e-19,
                double VT_p = 0.026 , double x_right_p = 5e-7,
                double x_left_p = 5e-7, double y_top_p = 5e-7,
                double y_bottom_p = 5e-7, double Tp_p = 1e-8,
                double Tn_p = 1e-8, double Dp_p = 10.4,double Dn_p = 221,
                double me_p = 6.103365e-32, double mh_p = 4.099275e-31);

        double potential(void) {return phi;}                          //      return
potential
        double u(void) {return U ;}                                                 //
        return U
        double v(void) {return V ;}                                   //
        return V
        double EG(void) {return Eg;}                                  //
        return Eg
        double CHI(void){return chi;}                                 //
        return chi
        double ret_n(void) {return n; }                                             //
        return n
        double ret_p(void) {return p; }                                             //
        return p
        double ret_Ni(void) {return ni;}                              //
        return ni
        double ret_Nc(void) {return Nc;}                              //
        return Nc
        double ret_Nv(void) {return Nv;}                              //
        return Nv
        double ret_VT(void) {return VT;}                              //
        return VT
        double ret_Dn(void) {return Dn;}                              //
        return Dn
        double ret_Dp(void) {return Dp;}                              //
        return Dp
        double ret_me(void) {return me;}                              //
        return me
        double ret_mh(void) {return mh;}                              //
        return mh
        double ret_x_left(void) {return x_left;}                      //      return
 x_left
        double ret_x_right(void) {return x_right;}                    //      return
 x_right
        double ret_y_top(void) {return y_top;}                                      //
        return y_top
        double ret_y_bottom(void) {return y_bottom;}          //      return y_bottom
        double ret_eps(void) {return eps;}                                          //
        return eps
        double ret_Field(void) {return field;}                                      //
        return field
        double ret_Density(void) {return density;}                    //      return
 density
        double ret_PhiApplied(void) {return phi_applied;}     //      return phi_applied

        // calculate phi from the neighbour's objects
        double cal_phi (CMaterial &ipl, CMaterial &iml, CMaterial &jpl,
                        CMaterial &jml, double error_limit = 1e-7,int m = 1000);

        //calculate U from the neighbour 's objects
        double cal_U (CMaterial &ipl, CMaterial &iml, CMaterial &jpl, CMaterial &jml);


        //calculate V from the neighbour 's objects
        double cal_V (CMaterial &ipl, CMaterial &iml, CMaterial &jpl, CMaterial &jml);

        // calculate electron quasi-Fermi level
        double cal_fermi_level_n(double ref_chi, double ref_Eg);

        // calculate hole quasi-Fermi level
        double cal_fermi_level_p(double ref_chi, double ref_Eg);

        // calculate electron density in QW
        void cal_qwn(double width, double depth, double ref_Eg, double ref_chi);
        // width is in meter and depth is in Joule
```

```
        // calculate hole density in QW
        void cal_qwp(double width, double depth, double ref_Eg, double ref_chi);
        // width is in meter and depth is in Joule

        //calculate field from the neighbour's objects
        void cal_field (CMaterial &ip1, CMaterial &im1, CMaterial &jp1, CMaterial
&jm1);

        void set_Density(double denst);                                          //
set material density
        void set_AppliedVolt(double phi_appl);                          // set the
appled voltege
        void set_time(double tau);                                               //
set lifetime
        void set_x_left(double size1) {x_left = size1;}             // set x_left
        void set_x_right(double size1) {x_right = size1;}    // set x_right
        void set_y_top(double size1) {y_top = size1;}               // set y_bottom
        void set_y_bottom(double size1) {y_bottom = size1;} // set y_bottom
        CMaterial operator= (const CMaterial &input);        // declare equal sige
output

        virtual ~CMaterial();

};

#endif // !defined(AFX_MATERIAL_H__DFEF2D5F_B71A_4201_B479_C270EABE9B2A__INCLUDED_)
```

- **Class Implementation (Material.cpp)**

```
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"

#include <iostream.h>
#include <math.h>
#include <stdlib.h>

#include "Self_Consistence.h"
#include "Material.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CMaterial::CMaterial()
{
        phi                     = 0;
        field                   = 0;
        phi0                    = 0;
        phi_applied             = 0;
        U                       = 0;
        V                       = 0;
        n                       = 0;
        p                       = 0;
        density                 = 0;
        ni                      = 0;
        q                       = 0;
        eps                     = 0;
        Eg                      = 0;
        chi                     = 0;
        VT                      = 0;
        Temperature             = 0;
        x_right                 = 0;
        x_left                  = 0;
        y_top                   = 0;
        y_bottom                = 0;
        Nc                      = 0;
```

```
        Nv                      = 0;
        Tp                      = 0;
        Tn                      = 0;
        Dp                      = 0;
        Dn                      = 0;
}

CMaterial::CMaterial(double density_p, double phi_applied_p,
                                        double ni_p, double eps_p, double Eg_p,
                                        double chi_p, double Nc_p, double Nv_p,
                                        double Temperature_p,  double q_p,
                                        double VT_p, double x_right_p,
                                        double x_left_p, double y_top_p,
                                        double y_bottom_p, double Tp_p ,
                                        double Tn_p, double Dp_p,double Dn_p,
                                        double me_p, double mh_p)
{
        double temp;
        density                 = density_p;
        phi_applied             = phi_applied_p;
        ni                      = ni_p;
        q                       = q_p;
        eps                     = eps_p;
        Eg                      = Eg_p;
        chi                     = chi_p;
        VT                      = VT_p;
        Temperature             = Temperature_p;
        x_right                 = x_right_p;
        x_left                  = x_left_p;
        y_top                   = y_top_p;
        y_bottom                = y_bottom_p;
        Nc                      = Nc_p;
        Nv                      = Nv_p;
        Tp                      = Tp_p;
        Tn                      = Tn_p;
        Dp                      = Dp_p;
        Dn                      = Dn_p;
        field                   = 0;
        me                      = me_p;
        mh                      = mh_p;

        if (density<0) {
                temp = -density / (2 * ni );
                phi0 = -log(temp + sqrt((temp*temp) + 1));
        }
        else {
                temp = density / ( 2 * ni);
                phi0 = log(temp + sqrt((temp * temp ) + 1 ));
        }

        phi = phi_applied + phi0*VT;
        U   = exp(-phi_applied/VT);
        V   = exp(phi_applied/VT);
        //n   = ni * Nc * U * exp(phi/VT) * exp(chi/q/VT);
        n   = ni * Nc * U * exp((phi+chi/q)/VT);
        //p     = ni * Nv * V * exp(-phi/VT) * exp(-(chi+Eg)/q/VT);
        p       = ni * Nv * V * exp(-(phi+(chi+Eg)/q)/VT);
}

void CMaterial::SetValue(double density_p, double phi_applied_p,
                double ni_p, double eps_p,
                double Eg_p, double chi_p,
                double Nc_p, double Nv_p,
                double Temperature_p, double q_p,
                double VT_p, double x_right_p,
                double x_left_p, double y_top_p,
                double y_bottom_p, double Tp_p,
                double Tn_p, double Dp_p, double Dn_p,
                double me_p, double mh_p)
{
        double temp;
        density                 = density_p;
        phi_applied             = phi_applied_p;
        ni                      = ni_p;
        q                       = q_p;
        eps                     = eps_p;
        Eg                      = Eg_p;
```

```
        chi                     = chi_p;
        VT                      = VT_p;
        Temperature             = Temperature_p;
        x_right                 = x_right_p;
        x_left                  = x_left_p;
        y_top                   = y_top_p;
        y_bottom                = y_bottom_p;
        Nc                      = Nc_p;
        Nv                      = Nv_p;
        Tp                      = Tp_p;
        Tn                      = Tn_p;
        Dp                      = Dp_p;
        Dn                      = Dn_p;
        field                   = 0;
        me                      = me_p;
        mh                      = mh_p;

        if (density<0) {
                temp = -density / (2 * ni );
                phi0 = -log(temp + sqrt((temp*temp) + 1));
        }
        else {
                temp = density / ( 2 * ni);
                phi0 = log(temp + sqrt((temp * temp ) + 1 ));
        }

        phi = phi_applied + phi0*VT;
        U   = exp(-phi_applied/VT);
        V   = exp(phi_applied/VT);
        n   = ni * Nc * U * exp((phi+chi/q)/VT);
        p       = ni * Nv * V * exp(-(phi+(chi+Eg)/q)/VT);
}

double CMaterial::cal_phi(CMaterial &ipl, CMaterial &iml, CMaterial &jpl,
                                                CMaterial &jml, double error_limit, int
m)
{
        double y, dy, dxeps, dyeps;
        double aij, aiplj, aimlj, aijpl, aijml;
        double F, f;
        double G1, G2;
        double dfx, dfy;

        if (m <= 0)
        {
                cout << "It's not funny" << m << ", set 'm' again.\n";
                exit(1);
        }

        dxeps = ipl.eps/x_right/2 - iml.eps/x_left/2 - eps*(1/x_right-1/x_left)/2;
        dyeps = jpl.eps/y_top/2 - jml.eps/y_bottom/2 - eps*(1/y_top-1/y_bottom)/2;
        aiplj = eps*2./(x_right+x_left)/x_right + dxeps/x_right/2;
        aimlj = eps*2./(x_right+x_left)/x_left - dxeps/x_left/2;
        aijpl = eps*2./(y_top+y_bottom)/y_top + dyeps/y_top/2;
        aijml = eps*2./(y_top+y_bottom)/y_bottom-dyeps/y_bottom/2;
        aij       = aiplj + aimlj + aijpl + aijml;

        f  = -q*density;
        F  = aiplj*ipl.phi + aimlj*iml.phi + aijpl*jpl.phi + aijml*jml.phi - f;
        G1 = q*ni*(Nc*U*exp((phi+chi/q)/VT)
                - Nv*V*exp(-(phi+(chi+Eg)/q)/VT));
        y  = aij*phi + G1 - F;

        while ((fabs(y) > error_limit) && (m > 0))
        {
                m--;
                G2      = q*ni*(Nc*U/VT*exp((phi+chi/q)/VT)
                          + Nv*V/VT*exp(-(phi+(chi+Eg)/q)/VT));
                dy      = aij + G2;
                phi     -= y/dy;

                G1      = q*ni*(Nc*U*exp((phi+chi/q)/VT)
                          -Nv*V*exp(-(phi+(chi+Eg)/q)/VT));
                y       = aij*phi + G1 -F;
        }
```

```
        if ( m < 0 )
        {
                        cout << "dy = " << dy <<"\n";
                        cout << "1/dy = " << 1/dy <<"\n";
                        cout << "Function is not convergent , exit program. \n";
                        exit(1);
        }

        n    = ni * Nc * U * exp((phi+chi/q)/VT);
        p       = ni * Nv * V * exp(-(phi+(chi+Eg)/q)/VT);

        dfx       = -(ipl.phi-phi)/x_right/2 - (phi-iml.phi)/x_left/2;
        field = sqrt(dfx*dfx);


        dfx = -(ipl.phi-phi)/x_right/2 - (phi-iml.phi)/x_left/2;
        dfy = -(jpl.phi-phi)/y_top/2 - (phi-jml.phi)/y_bottom/2;
        field = sqrt(dfx*dfx + dfy*dfy);

        return phi;
}

double CMaterial::cal_U(CMaterial &ipl, CMaterial &iml, CMaterial &jpl,
                                                CMaterial &jml)
{
        double aij, aiplj, aimlj, aijpl, aijml;
        double cnhp, cnhm, cnkp, cnkm, bnhp, bnhm, bnkp, bnkm;
        double enhp, xnhp, xnhm, xnkp, xnkm;
        double F, f, G, r;

        cnhp  = 2*Dn*Nc / (x_right+x_left) / x_right;
        xnhp  = (phi-ipl.phi)/VT + (chi-ipl.chi)/q/VT;
        bnhp  = fabs(xnhp) < 1e-10 ? 1-0.5*xnhp : xnhp/(exp(xnhp)-1);
        aiplj = cnhp * bnhp;

        cnhm  = 2*Dn*Nc / (x_right+x_left) / x_left;
        xnhm  = (phi-iml.phi)/VT + (chi-iml.chi)/q/VT;
        bnhm  = fabs(xnhm) < 1e-10 ? 1-0.5*xnhm : xnhm/(exp(xnhm)-1);
        aimlj = cnhm * bnhm;

        cnkp  = 2*Dn*Nc / (y_top+y_bottom) / y_top;
        xnkp  = (phi-jpl.phi)/VT + (chi-jpl.chi)/q/VT;
        bnkp  = fabs(xnkp) < 1e-10 ? 1-0.5*xnkp : xnkp/(exp(xnkp)-1);
        aijpl = cnkp * bnkp;

        cnkm  = 2*Dn*Nc / (y_top+y_bottom) / y_bottom;
        xnkm  = (phi-jml.phi)/VT + (chi-jml.chi)/q/VT;
        bnkm  = fabs(xnkm) < 1e-10 ? 1-0.5*xnkm : xnkm/(exp(xnkm)-1);
        aijml = cnkm * bnkm;

        aij = aiplj + aimlj + aijpl + aijml;

        enhp = exp((phi+chi/q)/VT);

        r = 1. / (Dn*ni*(Nv*V*exp(-(phi+chi/q)/VT)+1)
                        + Dp*ni*(Nc*U*exp((phi+chi/q)/VT)+1));

        G = ni*V*r/enhp;
        f = -ni*V enhp;
        F = aiplj*ipl.U + aimlj*iml.U + aijpl*jpl.U + aijml*jml.U - f;

        U = F / (aij+G );

        n = ni*Nc*U*exp((phi+chi/q)/VT);

        return U;
}

double CMaterial::cal_V(CMaterial &ipl, CMaterial &iml, CMaterial &jpl,
                                                CMaterial &jml)
{
        double aij, aiplj, aimlj, aijpl, aijml;
        double cphp, cphm, cpkp, cpkm, bphp, bphm, bpkp, bpkm;
        double ephp, xphp, xphm, xpkp, xpkm;
        double F, f, G, r;

        cphp  = 2*Dp*Nv / (x_right+x_left) / x_right;
```

```
        xphp  = (ip1.phi-phi)/VT + (ip1.chi-chi)/q/VT + (ip1.Eg-Eg)/q/VT;
        bphp  = fabs(xphp) < 1e-10 ? 1+0.5*xphp : xphp/(exp(xphp)-1);
        aiplj = cphp * bphp ;

        cphm  = 2*Dp*Nv / (x_right+x_left) / x_left;
        xphm  = (im1.phi-phi)/VT + (im1.chi-chi)/q/VT + (im1.Eg-Eg)/q/VT;
        bphm  = fabs(xphm) < 1e-10 ? 1+0.5*xphm : xphm/(exp(xphm)-1);
        aim1j = cphm * bphm ;

        cpkp  = 2*Dp*Nv / (y_top+y_bottom) / y_top;
        xpkp  = (jp1.phi-phi)/VT + (jp1.chi-chi)/q/VT + (jp1.Eg-Eg)/q/VT;
        bpkp  = fabs(xpkp) < 1e-10 ? 1+0.5*xpkp : xpkp/(exp(xpkp)-1);
        aijp1 = cpkp * bpkp;

        cpkm  = 2*Dp*Nv / (y_top+y_bottom) / y_bottom;
        xpkm  = (jm1.phi-phi)/VT + (jm1.chi-chi)/q/VT + (jm1.Eg-Eg)/q/VT;
        bpkm  = fabs(xpkm) < 1e-10 ? 1+0.5*xpkm : xpkm/(exp(xpkm)-1);
        aijm1 = cpkm * bpkm;

        aij = aiplj + aim1j + aijp1 + aijm1;

        ephp  = exp(-(phi+chi/q+Eg/q)/VT);

        r = 1. / (Tn*ni*(Nv*V*exp(-(phi+(chi+Eg)/q)/VT)+1)
                         + Tp*ni*(Nc*U*exp((phi+chi/q)/VT)+1));

        G = ni*U*r/ephp;
        f = -ni*r/ephp;
        F = aiplj*ip1.V + aim1j*im1.V + aijp1*jp1.V + aijm1*jm1.V - f;
        V = F / (aij+G);

        p = ni*Nv*V*exp(-phi/VT)*exp(-(chi+Eg)/q/VT);

        return V;
}

void CMaterial::cal_field(CMaterial &ip1, CMaterial &im1, CMaterial &jp1,
                                                  CMaterial &jm1)
{
        double dfx, dfy;

        dfx = -(ip1.phi-phi)/x_right/2 - (phi-im1.phi)/x_left/2;
        dfy = -(jp1.phi-phi)/y_top/2 - (phi-jm1.phi)/y_bottom/2;

        field = sqrt(dfx*dfx + dfy*dfy);
}

double CMaterial::cal_fermi_level_n(double ref_chi, double ref_Eg)
{
        double Efi, Ef;

        Efi = -phi*q - (chi+ref_chi) - (Eg+ref_Eg)/2 + 3/4*VT*q*log(mh/me);

        Ef = Efi + q*(phi + chi/q + VT*log(Nc) + VT*log(U));
        return Ef;
}

double CMaterial::cal_fermi_level_p(double ref_chi, double ref_Eg)
{
        double Efi,Ef;

        Efi = -phi*q - (chi+ref_chi) - (Eg+ref_Eg)/2 + 3/4*VT*q*log(mh/me);

        Ef = Efi + q*(phi + chi/q + Eg/q - VT*log(Nv) - VT*log(V));

        return Ef;
}

void CMaterial::cal_qwn(double width, double depth, double ref_Eg, double ref_chi)
// width is in meter and depth is in Joule
{

        int num,i;
        double tmp = 0;
        double gam, hbar, lf, rg, a_temp, y, Ef;
        double Energ[30];
        double a[30];
```

```
        Ef = cal_fermi_level_n(ref_chi, ref_Eg);

        hbar = 6.625e-34/(2*3.14159);
        gam = sqrt(2*(me/hbar)*depth*width*width/hbar);

        num=0;

        do
        {
            lf=0;
            rg=1;
            num++;

            do
                {
                        a_temp = (lf+rg)/2;
                        y = gam*a_temp-((num-1)*3.14159+2*acos(a_temp));

                        if (y<0)
                            lf=a_temp;
                        else
                                rg=a_temp;
            } while((fabs(y)>1e-5)&&((1-lf)>1e-8));

            Energ[num]=depth*a_temp*a_temp;
            a[num]=a_temp;

        } while(((1-lf)>1e-8) && (num<2));

        n=0;

        for(i=1;i<num;i++)
        {
            tmp = log(1+exp((Ef+phi*q+(chi+ref_chi)-Energ[i])/(VT*q)));
            if (tmp==0)
                        tmp = exp((Ef+phi*q+(chi+ref_chi)-Energ[i])/(VT*q));
            n +=  7.232e+15*tmp;
        }

        U = exp(-(phi+chi/q)/VT)* n/(ni*Nc);
}

void CMaterial::cal_qwp(double width, double depth, double ref_Eg, double ref_chi)
// width is in meter and depth is in Joule
{
        int num, i;
        double tmp;

        double gam, hbar, lf, rg, a_temp, y, Ef;
        double Energ[30];
        double a[30];

        Ef = cal_fermi_level_p(ref_chi, ref_Eg);

        hbar = 6.625e-34/(2*3.14159);
        gam = sqrt(2*mh*depth*width*width/(hbar*hbar));
        num = 0;

        do
        {
            lf = 0;
            rg = 1;
            num++;

            do
                {
                        a_temp = (lf+rg)/2;
                        y = gam*a_temp - ((num-1)*3.14159+2*acos(a_temp));

                        if (y<0)
                            lf=a_temp;
                        else
                                rg=a_temp;

            } while((fabs(y)>1e-5)&&((1-lf)>1e-8));
```

```
            Energ[num] = depth*a_temp*a_temp;
            a[num] = a_temp;

        } while(((1-lf)>1e-8)&&(num<2));

        p = 1.0e-10;

        for(i=1;i<num;i++) {
            tmp=log(1+exp((-phi*q-(chi+ref_chi)-(Eg+ref_Eg)-Energ[i]-Ef)/(VT*q)));
            if (tmp>=0)
                        tmp       =       exp((-phi*q-(chi+ref_chi)-(Eg+ref_Eg)-Energ[i]-
Ef)/(VT*q));
            p += 4.857e+16*tmp;
        }

        V = p/(ni*Nv) * exp((phi+(chi+Eg)/q)/VT);
}

void CMaterial::set_Density(double denst)
{
        double temp;

        density = denst;

        if (density<0)
        {
                temp = -density/2*ni;
                phi0 = -log(temp + sqrt((temp*temp) + 1));
        }
        else
        {
                temp = density/2*ni;
                phi0 = log(temp + sqrt((temp*temp) + 1 ));
        }

        phi = phi_applied + phi0*VT;
        n       = ni*Nc*U*exp((phi+chi/q)/VT);
        p       = ni*Nv*V*exp(-(phi+(chi+Eg)/q)/VT);
}

void CMaterial::set_AppliedVolt(double phi_appl)
{
        phi_applied     = phi_appl;
        phi                     = phi_applied + phi0*VT;
        U                       = exp(-phi_applied/VT);
        V                       = exp(phi_applied/VT);

        n       = ni*Nc*U*exp((phi+chi/q)/VT);
        p       = ni*Nv*V*exp(-(phi+(chi+Eg)/q)/VT);
}

void CMaterial::set_time(double tau)
{
        Tp                      = tau;
        Tn                      = tau;
}

CMaterial CMaterial::operator=(const CMaterial &input)
{
        phi                     = input.phi;
        field                   = input.field;
        U                       = input.U;
        V                       = input.V;
        phi0                    = input.phi0;
        phi_applied             = input.phi_applied;
        density                 = input.density;
        ni                      = input.ni;
        n                       = input.n;
        p                       = input.p;
        q                       = input.q;
        eps                     = input.eps;
        Eg                      = input.Eg;
        chi                     = input.chi;
        VT                      = input.VT;
        Temperature             = input.Temperature;
        x_right                 = input.x_right;
        x_left                  = input.x_left;
```

```
        y_top                   = input.y_top;
        y_bottom                = input.y_bottom;
        Tp                      = input.Tp;
        Tn                      = input.Tn;
        Dn                      = input.Dn;
        Dp                      = input.Dp;
        Nc                      = input.Nc;
        Nv                      = input.Nv;
        me                      = input.me;
        mh                      = input.mh;

        return *this;
}

CMaterial::~CMaterial()
{

}
```

ภาคผนวก จ.
ผลงานในการประชุมวิชาการ

ผลงานในการประชุมวิชาการ ณ งาน วทท. 28 ในเดือนตุลาคม 2545

การประยุกต์ใช้ปรากฏการณ์ควอนตัมคอนไฟน์สตาร์ก ใน แหล่งกำเนิดแสงหลายความยาวคลื่นชนิดใหม่

AN APPLICATION OF THE QUANTUM CONFINED STARK'S EFFECT IN A NOVEL MULTIPLE WAVELENGTH LIGHT EMITTER.

ภัทร อัยรักษ์[1*], บุญเหลือ พงศ์ดารา[1], แอนโทนี วิคเกอร์ส[2]

Pattara Aiyarak[1*], Boonlua Phongdara[1], Anthony Vickers[2]

Department of Physics, Faculty of Science, Prince of Songkla University, Hatyai, Songkla 90112, Thailand; [2]Department of Electrical System Engineering, University of Essex, Colchester, CO4 3SQ, United Kingdom; e-mail address: apattara@ratree.psu.ac.th

บทคัดย่อ: แหล่งกำเนิดแสงหลายความยาวคลื่นชนิดสารกึ่งตัวนำซึ่งประกอบด้วยหลุมควอนตัม[1] ได้ถูกออกแบบขึ้นเพื่อศึกษาความเป็นไปได้โดยใช้คอมพิวเตอร์เป็นเครื่องมือในการศึกษา โครงสร้างของสิ่งประดิษฐ์สารกึ่งตัวนำนี้ใช้ $Al_xGa_{1-x}As$-GaAs เป็นโครงสร้างดังแสดงในรูปที่ 1 โดยมีกำแพงของระดับพลังงานที่รอยต่อของ n-doped และ intrinsic ซึ่งใน intrinsic นี้มีหลุมควอนตัมอยู่ตรงกลาง เมื่อถูก forward bias โฮลที่ p-doped จะเคลื่อนที่มารออยู่ที่หลุมควอนตัม อย่างไรก็ตามอิเล็กตรอนไม่สามารถเข้ามารวมกับโฮลได้เนื่องจากการเคลื่อนที่ของอิเล็กตรอนถูกกันโดยกำแพงพลังงานที่รอยต่อของ n-doped และ intrinsic อิเล็กตรอนจะเคลื่อนที่เข้ามาได้ก็ต่อเมื่อบริเวณ n-doped มีสนามไฟฟ้าแบบขวางมาให้พลังงาน หรือ "heat" อิเล็กตรอนทำให้อิเล็กตรอนมีพลังงานมากพอที่จะข้ามกำแพงพลังงานแล้วเข้าไปที่หลุมควอนตัมเพื่อรวมตัวกับโฮล แล้วให้แสงออกมา ขนาดของการ forward bias เป็นตัวควบคุมความยาวคลื่นที่จะออกมาเนื่องจากขนาดของการ forward bias ที่ต่างกันจะทำให้มีสนามไฟฟ้าต่างกันเกิดขึ้นที่บริเวณหลุมควอนตัม และเนื่องจากปรากฏการณ์ควอนตัมคอนไฟน์สตาร์ก ระดับพลังงานในหลุมควอนตัมจึงต่างกันเป็นผลให้ได้แสงที่มีความยาวคลื่นที่ต่างกัน คอมพิวเตอร์ได้ถูกนำมาศึกษาความสัมพันธ์ระหว่าง forward bias ที่ขนาดต่างๆ กับความยาวคลื่นที่ของแสงที่ออกมา

Abstract: A quantum well semiconductor light emitter[1] was designed and simulated by a computer in order to realize a possibility of being a multiple wavelength light emitter. The structure, as shown in Figure 1, is an $Al_xGa_{1-x}As$-GaAs system with an energy barrier at the interface between the n-doped region and the intrinsic region. The GaAS quantum well, in addition, is situated at the middle of the intrinsic region. When the device is forward biased, the holes from the p-doped region drift into the quantum well and wait for the electrons from the n-doped region; however, the electrons from the n-doped region is prevented from entering the quantum well because of the energy barrier. Once the n-doped region is applied with a transverse electric field, the electrons are so-called heated and able to overcome the energy barrier. The electrons, therefore, enter the quantum well and recombine with the holes; hence the light. The wavelength of the light can be altered by changing the forward bias voltage. The difference in the forward bias voltage gives the different electric field across the quantum well. Because of the quantum confined Stark's effect, the energy levels in the quantum well are different which correspond to the different emitting wavelength. In order to estimate the emitting wavelength with a curtain forward bias, a computer simulation is required.

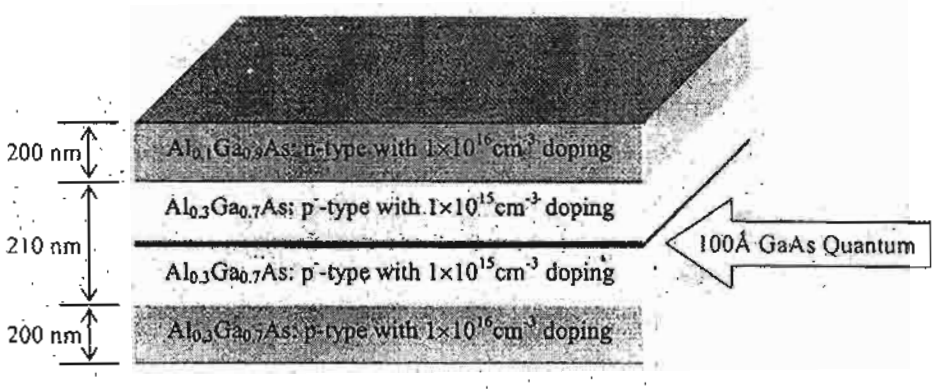

Figure 1: The structure of the device.

Methodology: A computer program was written in order to simulate the device. Poisson and Schrödinger's equations were solved self-consistently in the quantum well while the bulk material used the drift-diffusion model to simulate. Ultimately, the program will be used to study the relationship between the forward bias and the emitting wavelength after the n-doped region is heated. The program was written with the object-oriented concept so that the program is easy to modify and upgrade in the future.

Results, Discussion and Conclusion: The program is divided into two parts, i.e. drift-diffusion and quantum model. The first part is finished and verified by simulating a simple device, e.g. p-n junction diode, and the result fits with the analytical result. However, once the models were combined, the convergent problem occurred. The program and parameters are being studied and modified in order to fix this problem.

References: (1) Vickers, A. J., Mashayekhi, H. R., Aiyarak, P. and Oriato, D. (1999) *Turkish Journal of Physics*. **23**, 649.

Keywords: quantum confined Stark's effect, multiple wavelength light emitter, simulation, self-consistence

ภาคผนวก ฉ.

Manuscript

A Computational Search for the Optimized Structure of a Novel Light Emitter

P. Aiyarak[1], A. J. Vickers[2] and B. Phongdara[1]

[1]Department of Physics, Faculty of Science, Prince of Songkla University, Hatyai, Songkhla, 90112, Thailand
[2]Department of ESE, University of Essex, Colchester, CO4 3SQ, UK.

Abstract
A computer simulation is required when a novel idea of the semiconductor device was proposed. The device, which is called Hot Electron Barrier Light Emitter or HEBLE, being simulated here is a light source with the potential for being a multi-wavelength light emitter in the WDM system. The structure of the device is based on the $GaAs/Al_xGa_{1-x}As$ material system because it is well understood. The novelty of this device is in the way it operates. The difference in this device is the energy barrier at the interface between the n-doped region and the intrinsic region. The GaAS quantum well, in addition, is situated at the middle of the intrinsic region. When the device is forward biased, the holes from the p-doped region drift into the quantum well and wait for the electrons from the n-doped region; however, the electrons from the n-doped region is prevented from entering the quantum well because of the energy barrier. Once the n-doped region is applied with a transverse electric field, the electrons are so-called heated and able to overcome the energy barrier. The electrons, therefore, enter the quantum well and recombine with the holes; hence the light. The wavelength of the light can be altered by changing the forward bias voltage. The difference in the forward bias voltage gives the different electric field across the quantum well. Because of the quantum confined Stark's effect, the energy levels in the quantum well are different which correspond to the different emitting wavelength. To find an appropriate structure for the device, varieties of structures were put into the simulation. The best solution for the device is discussed.

1. Introduction

As multiple wavelength emitters are required in the WDM system, research in this area is very intensive. The tunable lasers can be roughly divided into three groups[1]. The first group is the mechanically tuned lasers. This type of laser is mainly a Fabry-Perot cavity that is adjacent to the lasing medium (i.e. external cavity) to filter out the unwanted wavelengths. Tuning is achieved by adjusting the distance between two mirrors. This type of tunable laser provides a wide range of tunability but the tuning time is limited. The second type of tunable laser is an acousto-optically and electro-optically tuned laser. These are tuned by applying sound waves or electrical current into the external cavity in order to change the refractive index. The tuning time is limited by the time required for light to build up in the cavity at the new frequency. The tuning time of an acousto-optic laser is a lot quicker than the mechanically tuned laser but still slower than the electro-optic laser. The tuning range is limited by the range of frequencies generated by the laser. The vertical cavity surface emitting laser (VCSEL) coupled with a micromachined deformable-membrane has been used to make an electro-optic laser. The wavelength tuning range of the laser is rather narrow, 19.1nm, but it can be tuned continuously[2]. A more recent example of an electro-optic laser is a micro-electromechanical tunable VCSEL laser (MEM-VCSEL)[3].

The device has three terminals where the first two are for biasing the laser and the third one is for changing the wavelength by moving the top mirror; hence the laser cavity. The last group of tunable lasers is totally different from the previous two. Instead of changing the external parameter, this type of tunable laser is tuned by changing the internal parameters of the laser. The parameters can be changed by injecting current into parts of laser or simply changing the bias. This group of tunable lasers has a wider range of wavelength than the second group and the tuning time is quicker than the first group. This type of laser will be discussed, as it is very popular and intensively studied. Several examples of this type of lasers are being studied worldwide. The examples of the laser in this group are the three-terminal bias induced color-tunable emitter (BICE)[4], the three-branch laser or Y-laser[5], the sampled grating distributed Bragg reflector (SG-DBR)[6], the super structure grating distributed Bragg reflector (SSG-DBR)[7], and etc. Here we propose another device in this category. The novelty of the device is the way of changing the wavelength. The device uses the quantum confined Stark's effect (QCSE) to change the wavelength. The structure, as shown in Figure 1, is an $Al_xGa_{1-x}As$-GaAs system with an energy barrier at the interface between the n-doped region and the intrinsic region. The GaAS quantum well, in addition, is situated at the middle of the intrinsic region. When the device is forward biased, the holes from the p-doped region drift into the quantum well and wait for the electrons from the n-doped region; however, the electrons from the n-doped region is prevented from entering the quantum well because of the energy barrier. Once the n-doped region is applied with a transverse electric field, the electrons are so-called heated and able to overcome the energy barrier. The electrons, therefore, enter the quantum well and recombine with the holes; hence the light. The wavelength of the light can be altered by changing the forward bias voltage. The difference in the forward bias voltage gives the different electric field across the quantum well. Because of the QCSE, the energy levels in the quantum well are different which correspond to the different emitting wavelength. A self-consistence between the drift-diffusion and Schrödinger equations are required to study the device.

2. Computer model

Our aim was to discover a device that does not turn on when forward biased, unless a transverse electric field is also applied. The simulation was undertaken on devices based on the AlxGa1-xAs-GaAs p-i-n structure with a quantum well within the i-region. The simulation uses a drift-diffusion transport model which is widely used in solving semiconductor device problems. The model consists of the well-known classical Poisson's equation and current continuity equations for both holes and electrons. The structure of interest contains a quantum well which cannot be solved by classical equations, only a solution of Schrödinger equation gives a correct estimation of electrons and holes in the quantum well. Therefore, a numerical solution of the Schrödinger equation is also required. In order to solve the derivative equations numerically, a particular definition of derivative is used. All main equations in the simulation were solved using the finite difference method and in some cases Newton's method were used in order to solved the non-linear algebraic equations[8]. The simulation is implemented in the C++ language and run under a UNIX environment. Using the technique of object oriented programming provided by C++; the program was written and modified easily[9].

The results from the simulation are the band edge diagram and carrier concentrations across the device at different forward bias voltages. Figure 2 shows the template of

the structure which will be simulated. To find the appropriate structure, parameters were altered and the solutions were compared and discussed.

3. Numerical results and discussion

After the parameters for the first device have been set, the device has been simulated without any voltage applied, i.e. in equilibrium. The band edge diagram can be seen in Figure 3. The electron and hole concentrations at different locations of the device are plotted against the applied voltages in Figure 4. Increasing the applied voltage increases the carrier concentrations in the quantum well and also increases the hole population in the n-doped region. The hole concentration in the quantum well is approximately 10^7 times higher at one-volt forward bias compared to the equilibrium. When the forward bias is increased to two volts, the hole concentration in the quantum well increases as expected. However, it is also high in the n-doped region, which will lead to unwanted recombination and light from the n-doped region.

It can also be seen that the barrier at the interface of the n- and p$^-$-doped regions does prevent electrons from the n-doped region flooding into the quantum well as the electron concentration in the quantum well is very low with every applied voltage. Many parameters, such as the position of the quantum well, the doping and the dimension of each region, were altered to see whether an operational device could be achieved.

A desired device is a device that does not turn on, i.e. no light emission, when forward biased, unless a transverse electric field is also applied to the n-doped region. In the other words, when the device is only forward biased, the electron concentration in the quantum well must be low while the hole concentration in it must be high. However, to avoid bulk emission, i.e. the emission at the n-doped region, the hole concentration in the n-doped region must be low.

The first parameter to be altered is the percentage of aluminium in the p$^-$- and p-doped regions. Actually the first device with $Al_{0.3}Ga_{0.7}As$ gives a reasonable result. However, it is a good idea to look at the effect of the composition of aluminum and the result is shown in Figure 5. A lower composition gives a lower barrier, so that the electrons can be heated easier. The main effect of this parameter is on the height of the heterojunction at the p-n interface and also the confinement of the quantum well. As the composition of Al in the p$^-$- and p-doped regions varies from 0.1 to 0.3, we can see that the hole concentration in the quantum well increases, i.e. a better confinement in the quantum well, while it decreases in the n-doped region. At the same time the increasing aluminium concentration produced a dramatic decrease of the electron population in the quantum well. So with reasons, the program gives appropriate results and an idea of how much the aluminium composition in p$^-$-and p-doped region should be. Clearly in terms of our requirements we should choose a high aluminium concentration. Therefore $Al_{0.3}Ga_{0.7}As$ was chosen for the p$^-$- and p-type regions.

The next parameter to be looked at is the position of the quantum well. The test devices will have the quantum well from 100nm away from the p-doped region to 300nm. Figure 6 shows the carrier concentrations in the quantum well and n-doped region. It can be seen that the closer the quantum well is to the p-doped region the higher the hole concentration in the quantum well at the same bias. The electron concentration in the quantum well is also lower when the quantum well is closer to the p-doped region. The hole concentration in n-doped region is, however, almost constant throughout the variation of the position of the quantum well. It is clear that the optimized device would have the quantum well at 100nm from the p-doped region.

The next parameter to be looked at is the size of the p^--doped region. As said earlier, the ideal device would have holes populating the quantum well when it is forward biased alone. If the distance from the n-doped region to the quantum well is long, it might take a longer time for electrons to arrive when the device is electrically heated limiting the response of the device, also losses due to the recombination. The length of the p^--doped region was varied in order to find the optimum. The quantum well was fixed at 100nm from the p-doped region.

It has been found, as seen in Figure 7, from the simulation that the results are almost identical in all sizes. As mentioned earlier, the smaller size of p^--doped region makes the device more responsive, and, as the simulation shows nothing different between the different sizes, the size of 200nm is chosen.

The next concern is the hole concentration in the quantum well. It seems to be low as the doping of the p-doped region is low. We are trying to have the highest possible concentration of holes in the quantum well by increasing the doping concentration in the p-doped region. However, the drawback is that the hole concentration in n-doped region is going to be higher as well. The variation of the doping concentration in the p-doped region will be from 1×10^{16} to $1\times10^{18}cm^{-3}$. Figure 8 shows the concentration of holes and electrons in the quantum well and n-doped region. There is a big improvement of the hole concentration in the quantum well from the doping of 1×10^{16} to $1\times10^{17}cm^{-3}$; however, it seems not to change much between the doping of 1×10^{17} and $1\times10^{18}cm^{-3}$. The doping of $1\times10^{17}cm^{-3}$ gave an acceptable result as the hole concentration is approximately $10^{17}cm^{-3}$ in the quantum well, which is high, but in the n-doped region is approximately $10^{15}cm^{-3}$, which is still low. The doping of $1\times10^{17}cm^{-3}$, therefore, was chosen for the doping of the p-doped region.

The next parameter to be looked at in this simulation is the percentage of the aluminium in the n-doped region. Previously, the n-doped region was GaAs. Increasing the aluminium composition would results in lowering the barrier. The result is shown in Figure 9. The hole concentration in the quantum well does not change with different materials for the n-doped region; however, the hole concentration in the n-doped region changes dramatically. It can also be seen that the electron concentration is getting higher in the quantum well as the aluminium composition of the n-doped region's material is higher, i.e. the barrier is lower. We have chosen $Al_{0.1}Ga_{0.9}As$ to be the material for the n-doped region as it gives a lower hole concentration in n-doped region. The reason that $Al_{0.2}Ga_{0.8}As$ was not chosen is that the heating might be difficult. The mobility of the material will be lower when the aluminium composition is higher, which makes it difficult to heat electrically. Moreover, the electron concentration in the quantum well is getting higher as the barrier is lower.

Last but not least, the doping concentration of the n-doped region was investigated. The reason we used $1\times10^{16}cm^{-3}$ for the doping of the n-doped region is that we might have encountered problems when heating a high doping region. The higher doping of the n-doped region results in the lower resistance, which requires more power for the same heating field. However, if the doping concentration is too low, the carrier population will not be enough. Our aim is to raise the doping to $5\times10^{16}cm^{-3}$. The simulation results, Figure 10, show that there is no difference for hole concentrations in both quantum well and the n-doped region when the doping of the n-doped region is changed to $5\times10^{16}cm^{-3}$.

One may notice a slightly higher, but still less than $10^{12}cm^{-3}$, of electron concentration in the quantum well. This higher electron concentration is basically from the higher doping of the n-doped region but as it is very low, it shows that the barrier is working

perfectly. Therefore we can safely change the doping concentration of the n-doped region to $5\times10^{16}cm^{-3}$.
So far, it seems that all parameters have been obtained. However, the middle part of the structure is the p^--doped region, which is doped at $1\times10^{15}cm^{-3}$. This doping level might not be easy to achieve when growing the wafer. Therefore, we decided to leave that region undoped. When the $Al_xGa_{1-x}As$ is undoped, the built-in doping level is $1\times10^{15}cm^{-3}$ n-type. Another simulation was performed to see if the change of the doping would affect any results we previously have. Figure 11 shows the concentration of carriers at an applied voltage of 2V and indicates that the results are insensitive to the doping in the central region where the range of doping is from p-doped of $1\times10^{15}cm^{-3}$ to n-doped of $1\times10^{15}cm^{-3}$. This is reasonable as the doping is very low and the middle region is almost totally depleted. We also found that all results at every bias are almost identical.

4. Conclusion

From the results, the appropriate structure was found as shown in Figure 12. However, according to the result shown in Figure 9, the material of the n-doped region will be $Al_{0.1}Ga_{0.9}As$. As mentioned earlier, the mobility of $Al_{0.1}Ga_{0.9}As$ is lower than GaAs, which makes the device more difficult to heat. The suggestion is that there should be two testing wafers with different materials in the n-doped region, i.e. $Al_{0.1}Ga_{0.9}As$ and GaAs.

Acknowledgements

The first author would like to specially thank to Thailand Research Fund (TRF) for funding and supporting. Special thanks also go to the Department of Physic, Faculty of Science, Prince of Songkla University, Thailand for allowing the first author to work on this project.

References

[1] M. S. Borella, J. P. Jue, D. Banerjee, B. Ramamurthy, and B. Mukherjee, "Optical Components for WDM Lightwave Networks", *Proceedings of the IEEE*. **85**, 1274 (1997).

[2] F. Sugihwo, M. C. Larson, and J. S. Harris, Jr., "Low threshold continuously tunable vertical surface-emitting lasers with 19.1 nm wavelength range", *Applied Physics Letters* **70**, 547 (1997).

[3] P. Wang, P. Tayebati, D. Vakhshoori, C. Lu, "Half-symmetric cavity microelectromechanically tunable vertical cavity surface emitting lasers with single spatial mode operating near 950 nm", *Applied Physics Letters* **75**, 897 (1999).

[4] F. E. Reed, D. Zhang, T. Zhang, and R. M. Kolbas, "Three-terminal bias induced dual wavelength semiconductor light emitter", *Applied Physics Letters* **65**, 570 (1994).

[5] M. Kuznetsov, "Design of Widely Tunable Semiconductor Three-Branch Lasers", *Journal of Lightwave Technology* **12**, 2100 (1994).

[6] H. Ishii, H. Tanobe, F. Kano, Y. Tohmori, Y. Kondo, and Y. Yoshikuni, "Broad-range wavelength coverage (62.4nm) with superstructure-grating DBR laser", *Electronics Letters* **32**, 454 (1996).

[7] R. C. Alferness, U. Koren, L. L. Buhl, B. I. Miller, M. G. Young, T. L. Koch, G. Raybon, and C. A. Burrus, "Broadly tunable InGaAsP/InP laser based on a

vertical coupler filter with 57-nm tuning range", *Applied Physics Letters* **60**, 3209 (1992).

[8] V. Comincioli, *Numeric Analysis*, McGraw Hill, New York, 1990.

[9] H. Schildt, *C++ from the Ground Up*, 2nd edition, Osborne McGraw-Hill, Berkeley, USA, 1998.

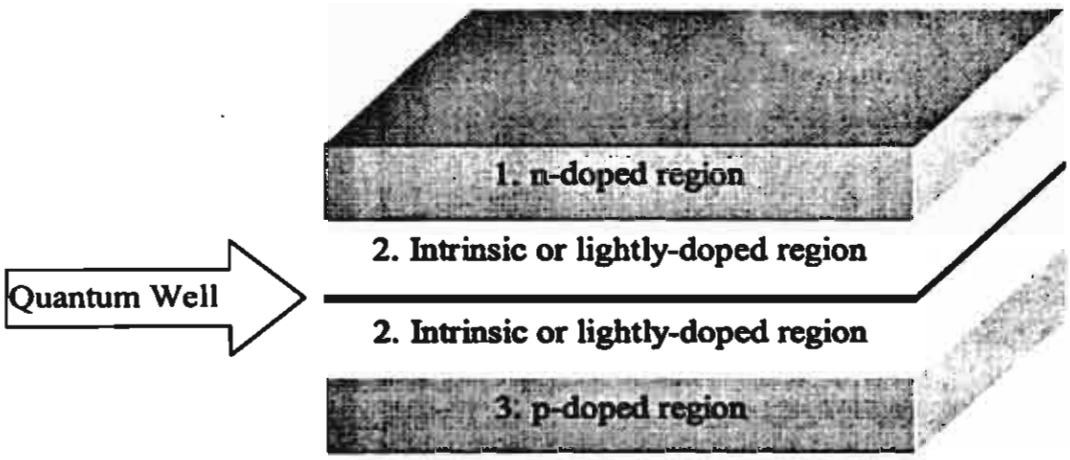

Figure 1. The outline of the device structure whose doping concentration, percentage of aluminium in $Al_xGa_{1-x}As$ of all layers, size and position of the quantum well were altered and simulated.

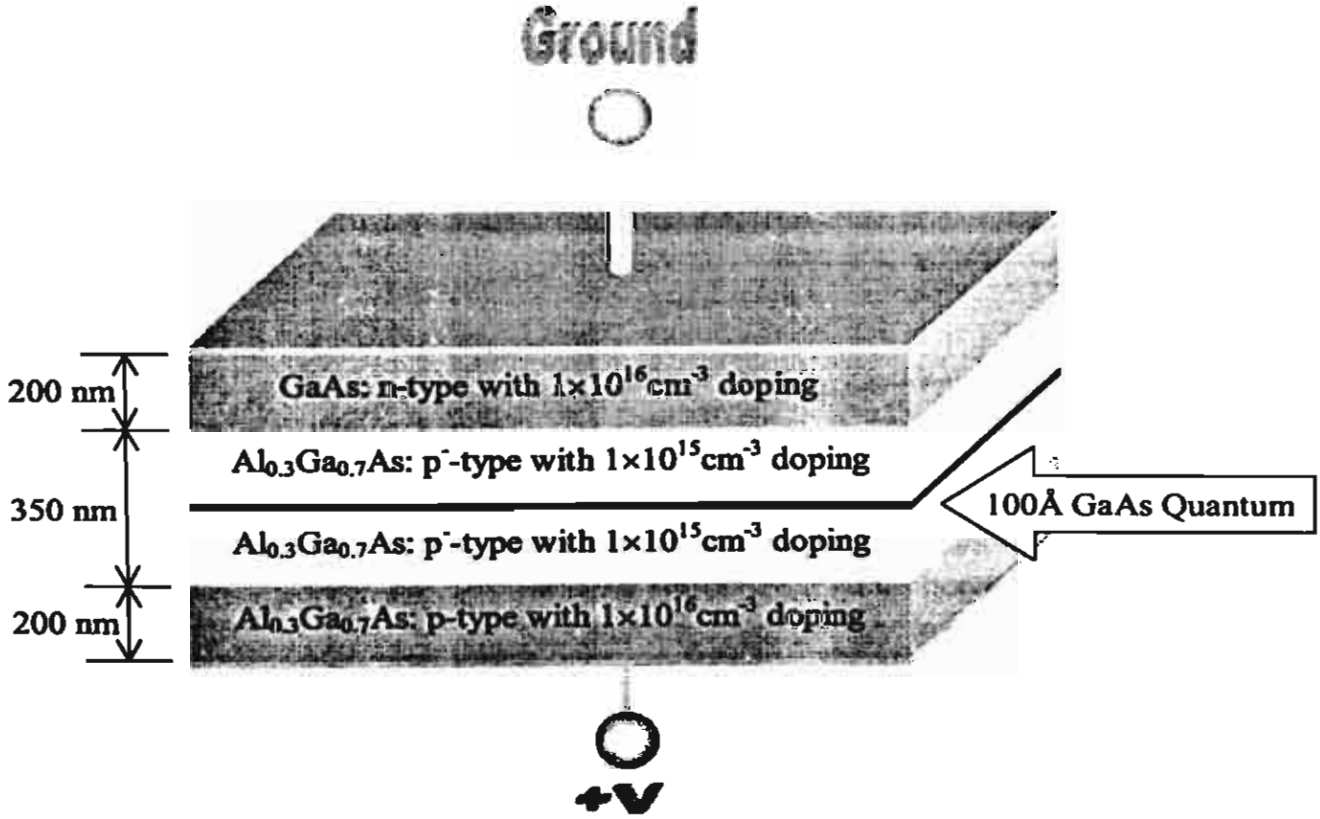

Figure 2. The first structure for simulation is shown with the dimensions and doping concentrations. The quantum well is in the middle of the p^--doped region.

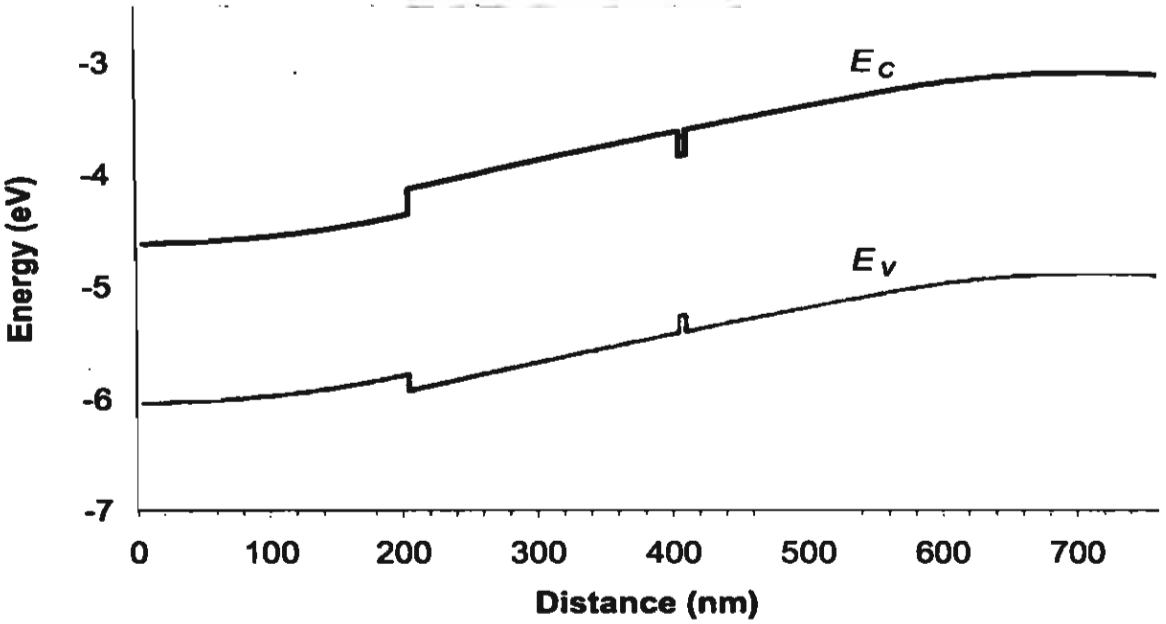

Figure 3. The band edge diagram of the first device in equilibrium.

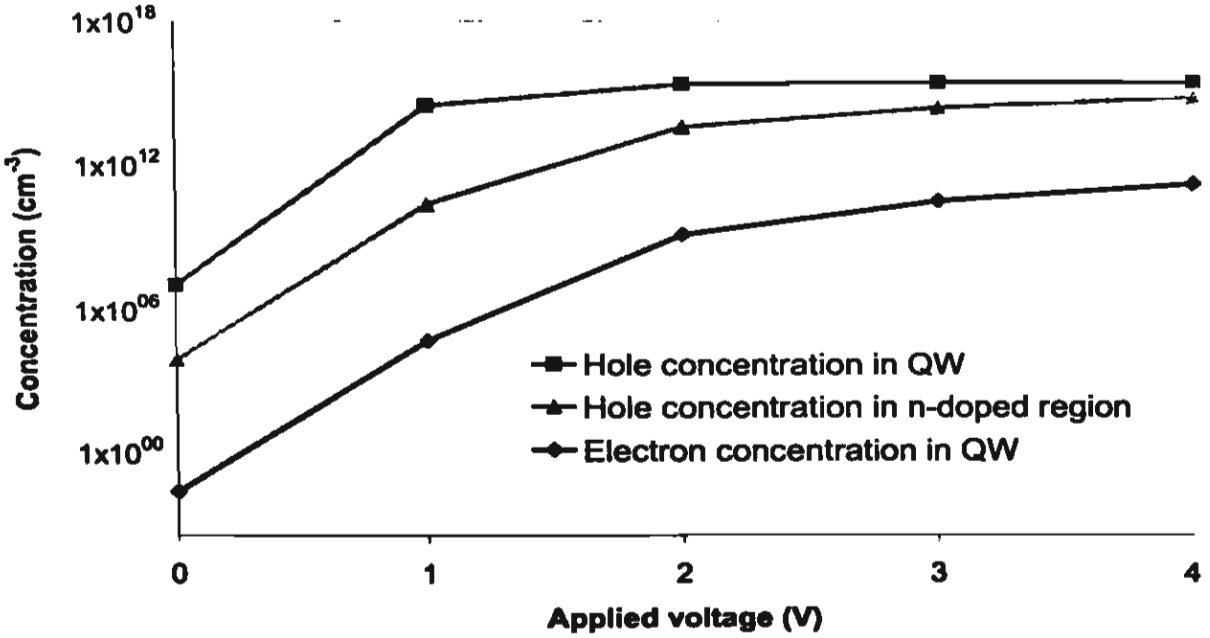

Figure 4. Carrier concentrations with the different applied voltages of the first device.

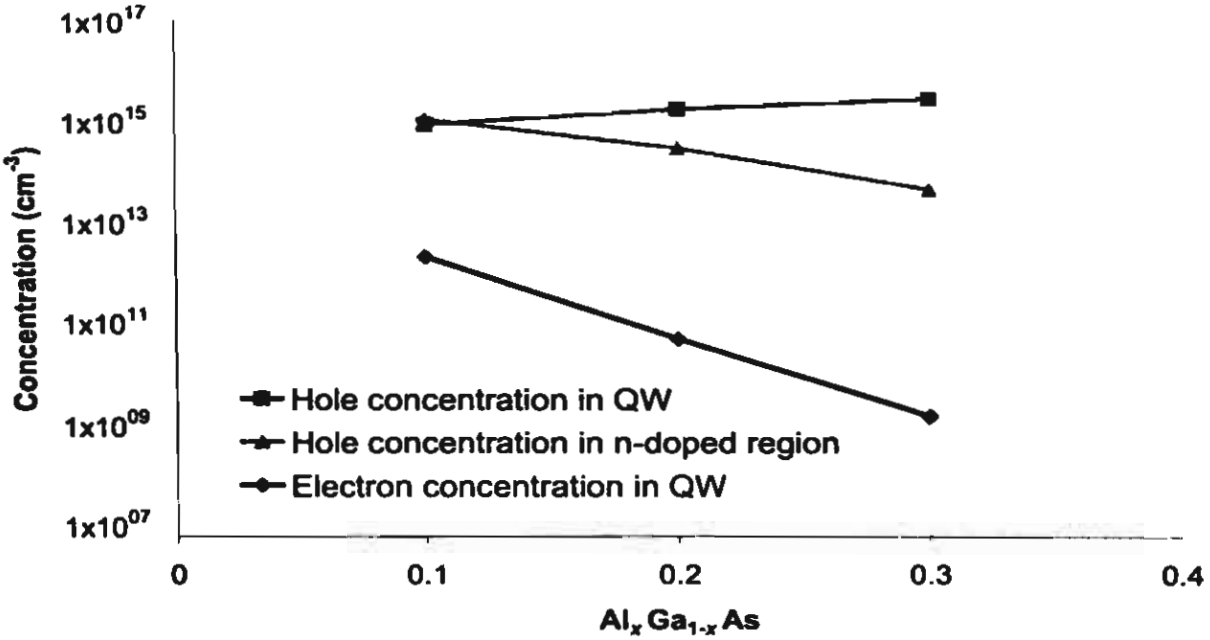

Figure 5. Concentrations in the quantum well and n-doped region of the devices with different Al composition in the p⁻- and p-doped regions. The devices are two-volt forward biased.

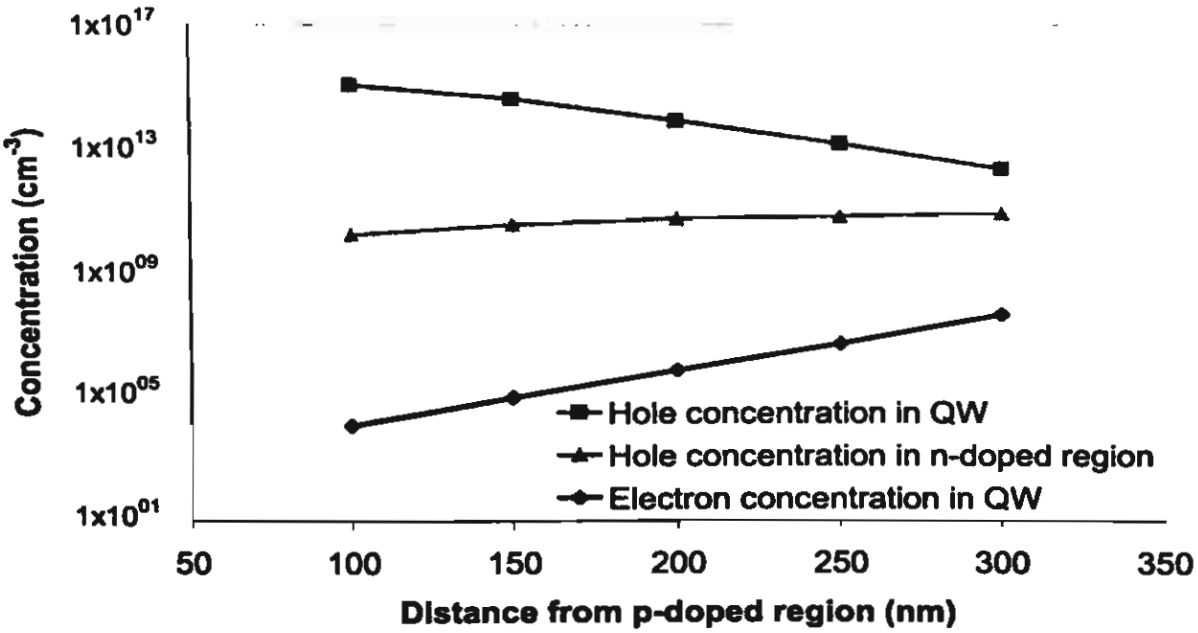

Figure 6. Concentrations in the quantum well and n-doped region of the devices with different position of the quantum well. The devices are one-volt forward biased.

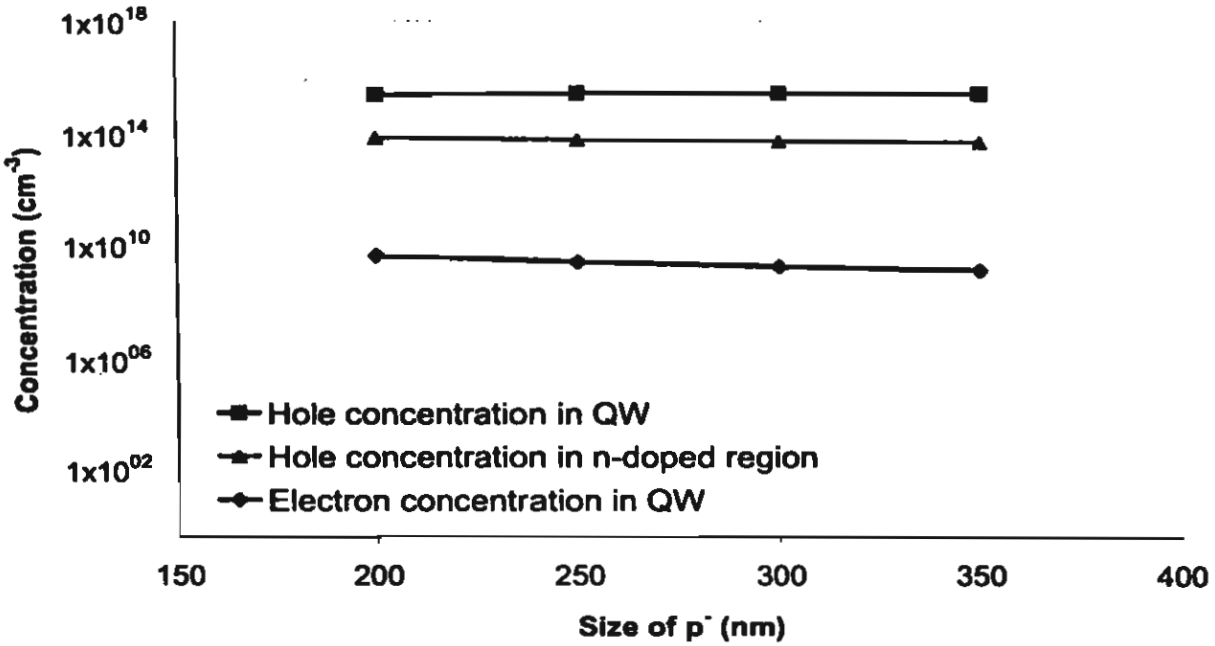

Figure 7. Concentrations in the quantum well and n-doped region of the devices with different sizes of the p--doped region. The devices are two-volt forward biased.

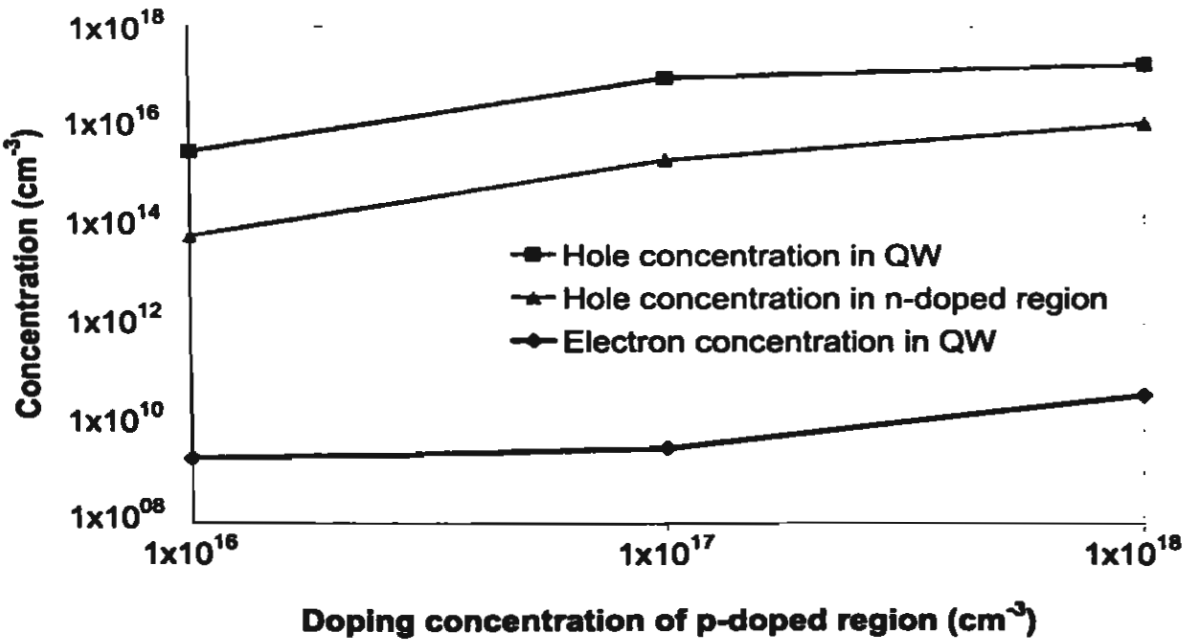

Figure 8. Concentrations in the quantum well and n-doped region of the devices with different doping of the p-doped region. The devices are two-volt forward biased.

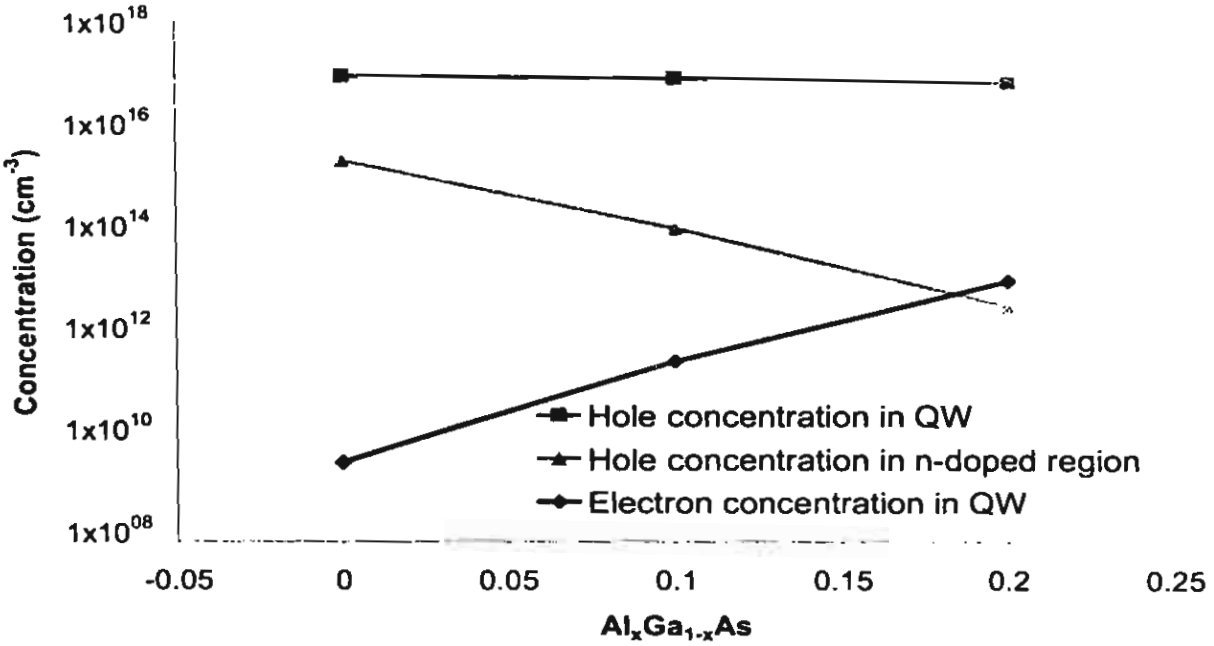

Figure 9. Concentrations in the quantum well and n-doped region of the devices with different Al composition in n-doped region. The devices are two-volt forward biased.

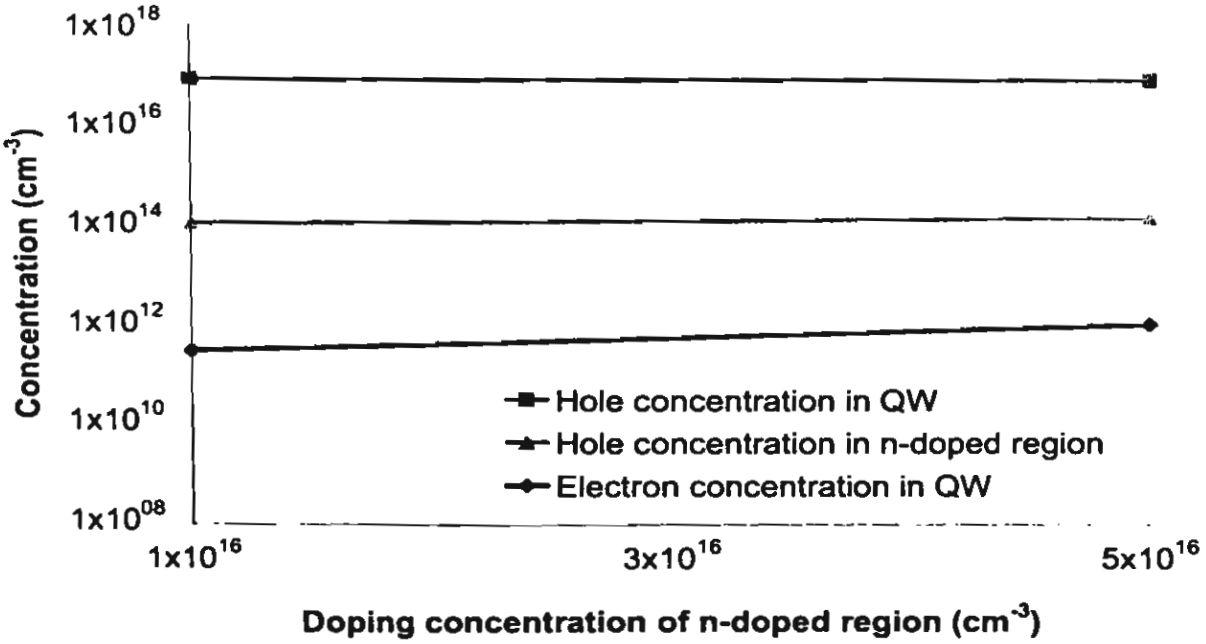

Figure 10. Concentrations in the quantum well and n-doped region of the devices with different doping of the n-doped region. The devices are two-volt forward biased.

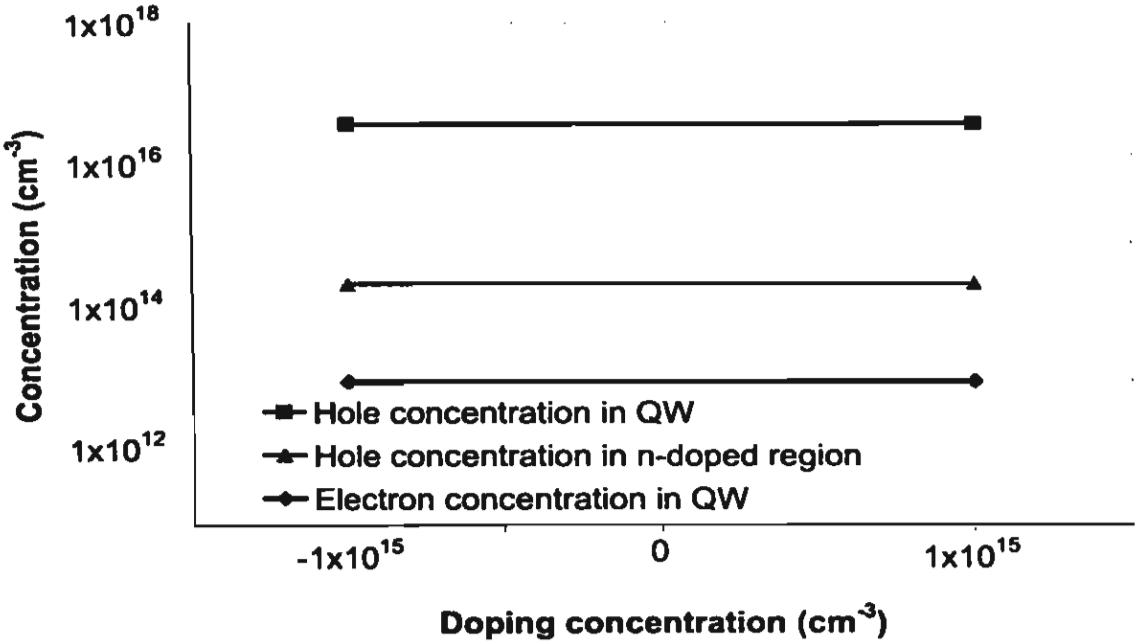

Figure 11. Concentrations in the quantum well and n-doped region of the devices with different doping of the middle region, where the negative doping means p-typed doping and the positive means n-typed doping. The devices are two-volt forward biased.

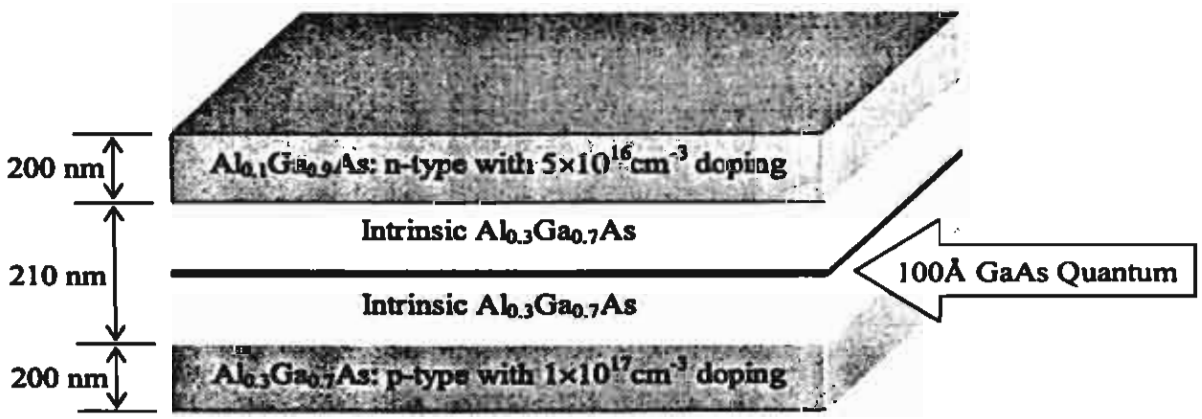

Figure 12. The optimized structure for the device. The quantum well is in the middle of the intrinsic region.