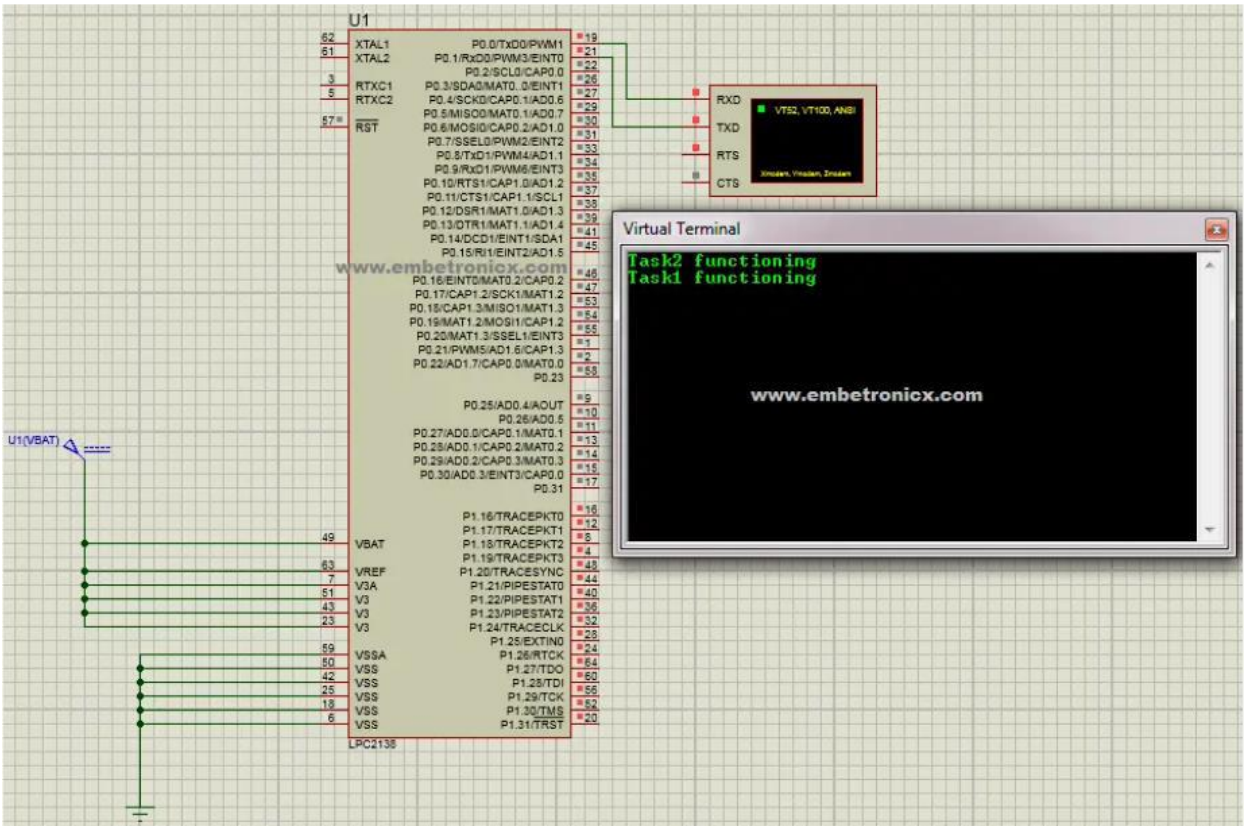
Assignment 4:

Code:

```
#include <lpc214x.h>
#include <stdlib.h>
#include "FreeRTOS.h"
#include "task.h"
#include "uart0.h"
void task1(void *q);
void task2(void *a);
void initpll(void);
int main(void)
{
initpll();
initserial();
xTaskCreate(task1,"task1",128,NULL,1,NULL);
xTaskCreate(task2,"task2",128,NULL,2,NULL);
vTaskStartScheduler();
}
void task1(void *q)
{
while(1) {
sendsserial("Task1 functioning");
sendsserial("\r\n");
vTaskDelay(1000);
}
}
void task2(void *a)
{
while(1) {
sendsserial("Task2 functioning");
```

```
sendsserial("\r\n");
vTaskDelay(1000);
}
}
void initpll(void)
{
PLL0CON=0x01;
PLL0CFG=0x24;
PLL0FEED=0xAA;
PLL0FEED=0x55;
while(!(PLL0STAT&1<<10));
PLL0CON=0x03;
PLL0FEED=0xAA;
PLL0FEED=0x55;
VPBDIV=0x01;
}
```

Output:



Assignment 5:

Code:

```
#include <stdio.h>
void swap(int *a,int *b)
{
int temp=*a;
*a=*b;
*b=temp;
}
int main()
{
int n;
printf("Enter Number of Processes: ");
scanf("%d",&n);
int burst[n],priority[n],index[n];

for(int i=0;i<n;i++)
{
printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
scanf("%d %d",&burst[i],&priority[i]);
index[i]=i+1;
}
for(int i=0;i<n;i++)
{
int temp=priority[i],m=i;
for(int j=i;j<n;j++)
{
if(priority[j] > temp)
{
temp=priority[j];
m=j;
}
}
swap(&priority[i], &priority[m]);
swap(&burst[i], &burst[m]);
swap(&index[i],&index[m]);
}
int t=0;
printf("Order of process Execution is\n");
for(int i=0;i<n;i++)
{
printf("P%d is executed from %d to %d\n",index[i],t,t+burst[i]);
t+=burst[i];

}
printf("\n");
printf("Process Id\tBurst Time\tWait Time\n");
int wait_time=0;
int total_wait_time = 0;
for(int i=0;i<n;i++)
```

```
{
printf("P%d\t\t%d\t\t%d\n",index[i],burst[i],wait_time);
total_wait_time += wait_time;
wait_time += burst[i];
}
float avg_wait_time = (float) total_wait_time / n;
printf("Average waiting time is %f\n", avg_wait_time);
int total_Turn_Around = 0;
for(int i=0; i < n; i++){
total_Turn_Around += burst[i];
}
float avg_Turn_Around = (float) total_Turn_Around / n;
printf("Average TurnAround Time is %f",avg_Turn_Around);
return 0; }
```

output:

```
Enter Number of Processes: 2

Enter Burst Time and Priority Value for Process 1: 5 3
Enter Burst Time and Priority Value for Process 2: 4 2
Order of process Execution is
P1 is executed from 0 to 5
P2 is executed from 5 to 9
Process Id Burst Time Wait Time
P1 5 0
P2 4 5
Average waiting time is 2.500000
Average TurnAround Time is 4.500000
```

Assignment 6:

Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x,n,p[10],pp[10],pt[10],w[10],t[10],awt,atat,i;
printf("Enter the number of process : ");
scanf("%d",&n);
printf("\n Enter process : time priorities \n");
for(i=0;i<n;i++)
{
printf("\nProcess no %d : ",i+1);
scanf("%d %d",&pt[i],&pp[i]);
p[i]=i+1;
}
for(i=0;i<n-1;i++)
{
for(int j=i+1;j<n;j++)
{
if(pp[i]<pp[j])
{
x=pp[i];
pp[i]=pp[j];
pp[j]=x;
x=pt[i];
pt[i]=pt[j];
pt[j]=x;
x=p[i];
p[i]=p[j];
p[j]=x;
}}}
```

```
w[0]=0;
awt=0;
t[0]=pt[0];
atat=t[0];
for(i=1;i<n;i++){
w[i]=t[i-1];
awt+=w[i];
t[i]=w[i]+pt[i];
atat+=t[i];
}
printf("\n\n Job \t Burst Time \t Wait Time \t Turn Around Time Priority \n");
for(i=0;i<n;i++)
printf("\n %d \t\t %d \t\t %d \t\t %d \t\t %d \n",p[i],pt[i],w[i],t[i],pp[i]);
awt/=n;
atat/=n;
printf("\n Average Wait Time : %d \n",awt);
printf("\n Average Turn Around Time : %d \n",atat);
getch();
}
```

Output:

Enter the number of processes: 4
Enter processes: time priorities
Process no 1: 3
1
Process no 2: 4
2
Process no 3: 5
3
Process no 4: 6
4

Job	Burst Time	Wait Time	Turn Around Time	Priority
4	6	0	6	4
3	5	6	11	3
2	4	11	15	2
1	3	15	18	1

Average Wait Time: 8

Average Turn Around Time: 11

Assignment 7:

Code:

```
#include<iostream>
#include<algorithm>
using namespace std;
struct node{
char pname;
int btime;
int atime;
int priority;
int restime=0;
int ctime=0;
int wtime=0;
}a[1000],b[1000],c[1000];
void insert(int n){
int i;
for(i=0;i<n;i++){
cin>>a[i].pname;
cin>>a[i].priority;
cin>>a[i].atime;
cin>>a[i].btime;
a[i].wtime=-a[i].atime+1;
}
}
bool btimeSort(node a,node b){
return a.btime < b.btime;
}
bool atimeSort(node a,node b){
return a.atime < b.atime;
}
bool prioritySort(node a,node b){
return a.priority < b.priority;
}
int k=0,f=0,r=0;
void disp(int nop,int qt){
int n=nop,q;
sort(a,a+n,atimeSort);
int ttime=0,i;
int j,tArray[n];
int alltime=0;
bool moveLast=false;
for(i=0;i<n;i++){
alltime+=a[i].btime;
}

alltime+=a[0].atime;
```

```
for(i=0;ttime<=alltime;){
j=i;
while(a[j].atime<=ttime&&j!=n){
b[r]=a[j];
j++;
r++;
}
if(r==f){
c[k].pname='i';
c[k].btime=a[j].atime-ttime;
c[k].atime=ttime;
ttime+=c[k].btime;
k++;
continue;
}
i=j;
if(moveLast==true){
sort(b+f,b+r,prioritySort);
// b[r]=b[f];
// f++;
// r++;
}
j=f;
if(b[j].btime>qt){
c[k]=b[j];
c[k].btime=qt;
k++;
b[j].btime=b[j].btime-qt;
ttime+=qt;
moveLast=true;
for(q=0;q<n;q++){
if(b[j].pname!=a[q].pname){
a[q].wtime+=qt;
}
}
}
else{
c[k]=b[j];
k++;
f++;
ttime+=b[j].btime;
moveLast=false;
for(q=0;q<n;q++){
if(b[j].pname!=a[q].pname){
a[q].wtime+=b[j].btime; }
}
}
if(f==r&&i>=n)
break;
}
tArray[i]=ttime;
ttime+=a[i].btime;
```

```
for(i=0;i<k-1;i++){
if(c[i].pname==c[i+1].pname){
c[i].btime+=c[i+1].btime;
for(j=i+1;j<k-1;j++)
c[j]=c[j+1];
k--;
i--;
}
}
int rtime=0;
for(j=0;j<n;j++){
rtime=0;
for(i=0;i<k;i++){
if(c[i].pname==a[j].pname){
a[j].restime=rtime;
break;
}
rtime+=c[i].btime;
}
}
float averageWaitingTime=0;
float averageResponseTime=0;
float averageTAT=0;
cout<<"\nGantt Chart\n";
rtime=0;
for (i=0; i<k; i++){
if(i!=k)
cout<<"| "<<'P'<< c[i].pname << " ";
rtime+=c[i].btime;
for(j=0;j<n;j++){
if(a[j].pname==c[i].pname)
a[j].ctime=rtime;
}
}
cout<<"\n";
rtime=0;
for (i=0; i<k+1; i++){
cout << rtime << "\t";
tArray[i]=rtime;
rtime+=c[i].btime;
}
cout<<"\n";
cout<<"\n";
cout<<"P.Name Priority AT\tBT\tCT\tTAT\tWT\tRT\n";
for (i=0; i<nop&&a[i].pname!='i'; i++){
if(a[i].pname=='\0')
break;
cout <<'P'<< a[i].pname << "\t";
cout << a[i].priority << "\t";
cout << a[i].atime << "\t";
cout << a[i].btime << "\t";
cout << a[i].ctime << "\t";
```

```
cout << a[i].wtime+a[i].ctime-rtime+a[i].btime << "\t";
averageTAT+=a[i].wtime+a[i].ctime-rtime+a[i].btime;
cout << a[i].wtime+a[i].ctime-rtime << "\t";
averageWaitingTime+=a[i].wtime+a[i].ctime-rtime;
cout << a[i].restime-a[i].atime << "\t";
averageResponseTime+=a[i].restime-a[i].atime;
cout <<"\n";
}
cout<<"Average Response time: "<<(float)averageResponseTime/(float)n<<endl;
cout<<"Average Waiting time: "<<(float)averageWaitingTime/(float)n<<endl;
cout<<"Average TA time: "<<(float)averageTAT/(float)n<<endl;
}
int main(){
int nop,choice,i,qt;
cout<<"Enter number of processes\n";
cin>>nop;
cout<<"Enter process, priority, AT, BT\n";
insert(nop);
disp(nop,1);
return 0;
}
```

Output:

```
Enter number of processes
5
Enter process, priority, AT, BT
1 6 0 52 4 1 2

3 3 2 4
4 1 3 1
5 2 4 7
Gantt Chart
| P1 | P2 | P3 | P4 | P3 | P5 | P3 | P2 | P1
0 1 2 3 4 5 12 14 15 19
P.Name Priority AT BT CT TAT WT RT
P1 6 0 5 19 19 14 0
P2 4 1 2 15 14 12 0
P3 3 2 4 14 12 8 0
P4 1 3 1 4 1 0 0
P5 2 4 7 12 8 1 1
Average Response time: 0.2
Average Waiting time: 7
Average TA time: 10.8
```

Assignment 8:

Code:

```
#include<iostream>
#include<algorithm>
using namespace std;
struct node{
char pname[50];
int btime;
int atime;
}a[50];
void insert(int n){
int i;
for(i=0;i<n;i++){
cin>>a[i].pname;
cin>>a[i].atime;
cin>>a[i].btime;
}
}
bool btimeSort(node a,node b){
return a.btime < b.btime;
}
bool atimeSort(node a,node b){
return a.atime < b.atime;
}
void disp(int n){
sort(a,a+n,btimeSort);
sort(a,a+n,atimeSort);
int ttime=0,i;
int j,tArray[n];
for(i=0;i<n;i++){
j=i;
while(a[j].atime<=ttime&&j!=n){
j++;
}
sort(a+i,a+j,btimeSort);
tArray[i]=ttime;
```

```
ttime+=a[i].btime;
}
tArray[i] = ttime;
float averageWaitingTime=0;
float averageResponseTime=0;
float averageTAT=0;
cout<<"\n";
cout<<"P.Name AT\tBT\tCT\tTAT\tWT\tRT\n";
for (i=0; i<n; i++){
cout << a[i].pname << "\t";
cout << a[i].atime << "\t";
cout << a[i].btime << "\t";
cout << tArray[i+1] << "\t";
cout << tArray[i]-a[i].atime+a[i].btime << "\t";
averageTAT+=tArray[i]-a[i].atime+a[i].btime;
cout << tArray[i]-a[i].atime << "\t";
averageWaitingTime+=tArray[i]-a[i].atime;
cout << tArray[i]-a[i].atime << "\t";
averageResponseTime+=tArray[i]-a[i].atime;
cout <<"\n";
}
cout<<"\n";
cout<<"\nGantt Chart\n";
for (i=0; i<n; i++){
cout <<"| "<< a[i].pname << " ";
}
cout<<"\n";
for (i=0; i<n+1; i++){
cout << tArray[i] << "\t";
}
cout<<"\n";
cout<<"Average Response time: "<<(float)averageResponseTime/(float)n<<endl;
cout<<"Average Waiting time: "<<(float)averageWaitingTime/(float)n<<endl;
cout<<"Average TA time: "<<(float)averageTAT/(float)n<<endl;
}
```

```
int main(){
int nop, choice, i;
cout<<"Enter number of processes\n";
cin>>nop;
insert(nop);
disp(nop);
return 0;
}
```

Output:

```
Enter number of processes
5
1 0 5
2 1 2
3 2 4
4 3 1
5 4 7
P.Name AT BT CT TAT WT RT
1 0 5 5 5 0 0
4 3 1 6 3 2 2
2 1 2 8 7 5 5
3 2 4 12 10 6 6
5 4 7 19 15 8 8
Gantt Chart
| 1 | 4 | 2 | 3 | 5
0 5 6 8 12 19
Average Response time: 4.2
Average Waiting time: 4.2
Average TA time: 8
```

Assignment 9:

Code:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
float avg_wt, avg_tat;
printf(" Total number of process in the system: ");
scanf("%d", &NOP);
y = NOP;
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t");
scanf("%d", &at[i]);
printf(" \nBurst time is: \t");
scanf("%d", &bt[i]);
temp[i] = bt[i];
}
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{

if(temp[i] <= quant && temp[i] > 0)
{
sum = sum + temp[i];
temp[i] = 0;
count=1;
}
else if(temp[i] > 0)
{
temp[i] = temp[i] - quant;
sum = sum + quant;
}
if(temp[i]==0 && count==1)
{
y--;
printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
wt = wt+sum-at[i]-bt[i];
tat = tat+sum-at[i];
count =0;
}
if(i==NOP-1)
{
i=0;
}
else if(at[i+1]<=sum)
{
```

```
i++;
}
Else

{
i=0;
}
}
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
return 0;
}
```

Output:

```
D:\C Codes\RR3.exe
 Total number of process in the system: 3

 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:       0

Burst time is:  7

 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:       2

Burst time is:  11

 Enter the Arrival and Burst time of the Process[3]
 Arrival time is:       5

Burst time is:  8
Enter the Time Quantum for the process:         4

 Process No             Burst Time             TAT             Waiting Time
Process No[1]           7                              15                      8
Process No[3]           8                              18                      10
Process No[2]           11                             24                      13
 Average Turn Around Time:      10.333333
 Average Waiting Time:  19.000000
--------------------------------
Process exited after 26.78 seconds with return value 0
Press any key to continue . . .
```

Assignment 10:

Code:

```
#include<stdio.h>
#include<string.h>
int gcd(int a,int b){
if(b==0)
return a;
else
gcd(b,a%b);}
int lcm(int a,int b){
return((a*b)/gcd(a,b));}
int hyperperiod(float period[],int n){
int k=period[0];
n--;
while(n>=1){
k=lcm(k,period[n--]);}
return k;}
int edf(float *period,int n,int t,float *deadline){
int i,small=10000.0f,smallindex=0;

for(int i=0;i<n;i++){
if(period[i]<small&&(period[i]-t)<=deadline[i]){
small=period[i];
smallindex=i;}}
if(small==10000.0f)
return -1;
return smallindex;}
int main()
{
int i,n,c,d,k,j,nexttime=0,time=0,task,preemption_count;
float
exec[20],period[20],individual_util[20],flag[20],release[20],deadline[20],instance[
20],ex[20],responsemax[20],responsemin[20],tempmax;
float util=0;
printf("\nEarliest Deadline First Algorithm\n");
FILE *read;
read=fopen("Sampledata.docx","r"); // Sampledata
fscanf(read,"%d ",&n);
for(i=0;i<n;i++)
{
fscanf(read,"%f ",&release[i]);
fscanf(read,"%f ",&period[i]);
fscanf(read,"%f ",&exec[i]);
```

```
fscanf(read,"%f ",&deadline[i]);
}
fclose(read);
for(i=0;i<n;i++)

{
individual_util[i]=exec[i]/period[i];
util+=individual_util[i];
responsemax[i]=exec[i];
deadline[i]=period[i];
instance[i]=0.0f;
}
util=util*100;
if(util>100)
printf("\n Utilisation factor = %0.2f \n\nScheduling is not possible as Utilisation
factor is above 100 \n",util);
else
{
printf("\nUtilisation factor = %0.2f \n\nScheduling is possible as Utilisation factor
is below 100 \n ",util);
printf("\nHyperperiod of the given task set is : %d\n\n",k=hyperperiod(period , n));
c=0;
while(time<k)
{
nexttime=time+1;
task = edf(period,n,time,deadline);
if(task==-1)
{
printf("-");
time++;
continue;
}

instance[task]++;
printf("T%d ",task);
ex[c++]=task;
if(instance[task]==exec[task])
{
tempmax=nexttime-(period[task]-deadline[task]);
if(instance[task]<tempmax)
{
responsemax[task]=tempmax;
}
else
{
```

```
responsemin[task]=instance[task];
}
if(deadline[task]==k)
{
responsemin[task]=responsemax[task];
}
period[task]+=deadline[task];
instance[task]=0.0f;
}
time++;
}
for(i=0;i<n;i++)
{
printf("\n\nMaximum Response time of Task %d = %f",i,responsemax[i

printf("\n\nMinimum Response time of Task %d = %f",i,responsemin[i]);
}
preemption_count=0;
for(i=0;i<k;i=j)
{
flag[i]=1;
d=ex[i];
for(j=i+1;d==ex[j];j++)
flag[d]++;
if(flag[d]==exec[d])
flag[d]=1;
else
{
flag[d]++;
preemption_count++;
}
}
printf("\n\nPreemption Count = %d",preemption_count);
}
return 0;
}
```

Output: