2013

DESIGN AND IMPLEMENT DYNAMIC PROGRAMMING BASED DISCRETE POWER LEVEL SMART HOME SCHEDULING USING FPGA

Xin Yang
michigan technological

DESIGN AND IMPLEMENT DYNAMIC PROGRAMMING BASED DISCRETE POWER LEVEL SMART HOME SCHEDULING USING FPGA

By

Xin Yang

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Electrical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2013

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Electrical Engineering

Department of Electrical and Computer Engineering

Report Advisor: *Shiyan Hu*

Committee Member: *Sumit Paudyal*

Committee Member: *Chaoli Wang*

Department Chair: *Daniel R. Fuhrmann*

Contents

Ⅰ. Abstract:

With the development and capabilities of the Smart Home system, people today are entering an era in which household appliances are no longer just controlled by people, but also operated by a Smart System. This results in a more efficient, convenient, comfortable, and environmentally friendly living environment. A critical part of the Smart Home system is Home Automation, which means that there is a Micro-Controller Unit (MCU) to control all the household appliances and schedule their operating times. This reduces electricity bills by shifting amounts of power consumption from the on-peak hour consumption to the off-peak hour consumption, in terms of different “hour price”. In this paper, we propose an algorithm for scheduling multi-user power consumption and implement it on an FPGA board, using it as the MCU. This algorithm for discrete power level tasks scheduling is based on dynamic programming, which could find a scheduling solution close to the optimal one. We chose FPGA as our system’s controller because FPGA has low complexity, parallel processing capability, a large amount of I/O interface for further development and is programmable on both software and hardware. In conclusion, it costs little time running on FPGA board and the solution obtained is good enough for the consumers.

Ⅱ. Introduction

1. History

Over a century ago, Nikola Tesla proposed the architecture of the power grid and people followed the design and developed it. During that time, electricity was a luxury resource that was used for lighting. Today, the power grid is used in almost all fields and people may not survive without it. Following the rapid technology development and no longer satisfying the present situation of the power grid, people are now concerned about issues such as greenness, efficiency, sustainability and reliability. The power grid could have the capability to become “smarter”, a smarter grid [1] [2].

2. Today’s power grid

There are many ways to make the power grid become smarter, such as improving the efficiency and reliability, developing environmentally friendly generators, managing the power consumption for consumers and so on so forth [3]. A basic and important part of the smart grid is the smart home. Specifically, if household tasks or daily power consumption could be arranged and scheduled, a more efficient system could be developed.

3. Daily life power consumption

The following figure demonstrates the inefficiency power consumption that would occur during daily life. This is a plot of the power consumption for the province Ontario, Canada, on a normal weekday in April 27, 2009 [4].

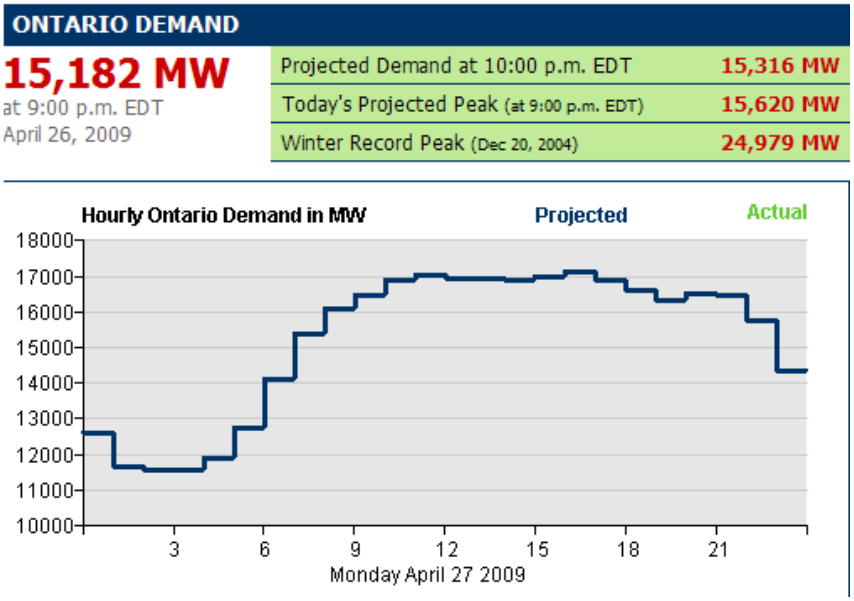


Figure 1. Power consumption in Ontario on a normal day

We could easily find out that the peak power consumption occurred during the afternoon and low power consumption during the late night when most people were asleep [5]. The inefficiency occurs when the power companies must build the power plant which could support this high demand. These power plants with more capacity need more maintenance and repair cost. Hence, utility companies have to purchase power at higher rate. Furthermore, consumers typically purchase power with same price during different time periods. As the result, consumers would not consider adjusting their power consumption.

4. Solutions

In order to improve this inefficient situation, utility companies would like to choose a “time of use” policy, which means that the power rate could be variable due to the total amount of power consumption. For example, higher power rate occurs during the afternoon and lower power rate occurs during the late night. Hence, for our individual consumers, we would like to have a smart system to help us schedule our household tasks in a reasonable way to reduce our electricity bill. The result could be that the controller for the system assists us to shift parts of

soft loads from the peak demand hours to the low demand hours. Soft loads are the loads can be scheduled, like washing clothes. During the day, the consumer may not need a precise washing time for clothes but ask for an end time to do this job. So, he/she can put the clothes there and let the smart washing machine decide when the clothes should be washed. These kinds of jobs are called soft loads, and hard loads are exactly the opposite. As previously stated, due to the smart scheduling, in terms of the individual the electricity bill is surely reduced when the hourly power rate offered by the utility company remains the same. These kinds of smart shifting behaviors can not only affect our electricity bill, but can also help the environment. For instance, in a typical week in October, one needs to do laundry. Thursday night during the high power demand hours, and Friday morning during the low power demand hours could be chosen as the time to start the laundry. The difference is, the power plants will need to add immediate and long-term generation capacity to support the electricity grid on Friday, which results in an increasing fuel levels in the plants by burning more coal for that kind of capacity buildup. Hence, due to the help of the smart scheduling, the emissions could be reduced [6].

However, if all the consumers in this community considered this reasonable scheduling by using the same smart system, they would all like to shift their loads from the peak demand hours to the low demand hours simultaneously. This would result in a bad situation that the original low demand hours become the high demand hours. Hence, if we need to solve this problem, communication with the other users is one of the prerequisites. Good news is that the smart system has a good capacity of communication. Thus, the controller could schedule the tasks based on the other's arrangement in an optimal way. This solution has two major advantages. On one hand, it has less complexity and results in a time saving strategy. On the other hand, it would allow the users to change the schedule optionally, since whenever the consumer inputs the newest schedule information to the controller in real time, the controller would immediately calculate the best scheduling option based on the other users' schedule.

This method is basically a dynamic programming strategy, which solves complex problems by dividing them into some simpler “sub-problems” and each time solves the sub-problem only once; the current solution depended on the previous computed optimal solution [7]. In this report, our proposed method is dynamic programming-based and focuses on the scheduling for the multi-user-community.

Since we need a Micro-Controller Unit (MCU) for the carrier of this proposed algorithm, FPGA is chosen due to its low complexity, convenient re-programmability both on software and hardware and the technology trend. This is preliminary work for the controller of the Smart Home system, because in our work, the controller can only schedule the tasks but not perform other smart behaviors like real life wireless remote controls, controlling the power level of the household appliances based on the temperature and so on so forth. Fortunately, it is not hard to do further developments on the FPGA controller and that is exactly its critical advantage.

Ⅲ. Algorithm

1. Dynamic programming method

Since our proposed algorithm is dynamic programming method based, I would like to introduce the dynamic programming strategy first. Dynamic programming is more like a divide and conquer algorithm, which divides the problems into pieces of sub-problems and then conquers them step by step. But, the difference is that these sub-problems are usually not mutually independent. If we solve this problem using the divide and conquer algorithm, some of the sub-problems may be calculated many times. Thus, if we could store the solutions of previous sub-problems and use them to solve the next one, an abundance of repeated work would be avoided. For example, we could use a table to record all the solutions of the sub-problems we have already solved, no matter whether these sub-problems would or would not be used for the future. This is the basic thinking of the dynamic programming algorithm. Keeping in mind, dynamic programming is not a typical algorithm that has a standard mathematical expression or clear structure. It is a way, a ladder, of solving the optimization problem. For different decision processes, there is different dynamic programming method-based algorithm [7].

2. Proposed algorithm

Now, I would like to present our proposed algorithm in the following paragraph. First of all, individual scheduling is presented. Assume that there are ***n*** hours in the time period, and there are ***m*** tasks that need to be scheduled during the time period. In our case, each task has 5 factors, which are the start time ***Task [i] [0]***, the end time ***Task [i] [1]***, power level 1 ***Task [i] [2]***, power level 2 ***Task [i] [3]*** and the total power consumption ***Task [i] [4]***. For an easy simulation, we only divide time into hourly periods and only have two options of the power levels for each task. Furthermore, each task can only be operated continuously and can

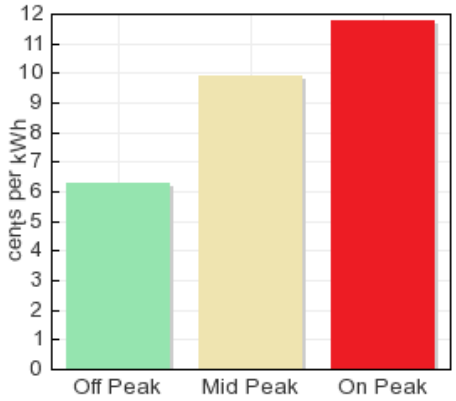
also change its power level during the operating time period. Hence, the total power consumption ***Task [i] [4]*** for one task would be,

$$\text{Task [i][4]} = \sum_{\text{t=Task [i] [0]}}^{\text{Task [i] [1]}} \text{Energy}_{\text{task[i],t}} \qquad (1)$$

Since we are on the premise that the utility company is using the “time of use” strategy, different power consumption at different times has different power rates. These prices for the different hours are offered to the customers before the day starts and the prices would not change during that day. For example, Table 1, which is shown below, indicates the price related with certain hours in Ontario from May, 2012 to Apr, 2013. [8] In our case, we assume each hour has 3 factors, ***hours [j] [0]***, ***hours [j] [1]*** and ***hours [j] [2]***. ***hours [j] [0]*** represents the threshold value. If the total power consumption of the community in this hour is larger than the threshold value, the power rate would be ***hours [j] [1]***. Otherwise, if the community total power consumption is less than the threshold value, the power rate would become ***hours [j] [2]***. Then, we could start our scheduling algorithm in this condition for seeking the best or one of the best solutions to schedule all of the ***n*** tasks.

From	To	Summer Rate (May - Oct)	Winter Rate (Nov - Apr)
7:00 AM	8:00 AM	mid-peak rate 9.9 cents/kWh	on-peak rate 11.8 cents/kWh
8:00 AM	9:00 AM		
9:00 AM	10:00 AM		
10:00 AM	11:00 AM		
11:00 AM	12:00 PM	on-peak rate 11.8 cents/kWh	mid-peak rate 9.9 cents/kWh
12:00 PM	1:00 PM		
1:00 PM	2:00 PM		
2:00 PM	3:00 PM		
3:00 PM	4:00 PM		
4:00 PM	5:00 PM		
5:00 PM	6:00 PM	mid-peak rate 9.9 cents/kWh	on-peak rate 11.8 cents/kWh
6:00 PM	7:00 PM		
7:00 PM	8:00 PM	off-peak rate 6.3 cents/kWh	off-peak rate 6.3 cents/kWh
8:00 PM	9:00 PM		
9:00 PM	10:00 PM		
10:00 PM	11:00 PM		
11:00 PM	Midnight		
Midnight	1:00 AM		
1:00 AM	2:00 AM		
2:00 AM	3:00 AM		
3:00 AM	4:00 AM		
4:00 AM	5:00 AM		
5:00 AM	6:00 AM		
6:00 AM	7:00 AM		

Comparison of Time-of-Use Rates



Note
Off peak rates are charged on weekends and holidays. The rates shown to the left apply Monday to Friday.

Based on a small sample of 152 households, we calculate that, on average, time-of-use consumption typically breaks down as follows:

Off-peak: 63.4%
Mid-peak: 16.0%
On-peak: 20.5%

This results in a blended rate of **8.0 cents/kWh.**

Help us improve our sample by participating in our Ontario Electricity Usage study.

First 600 Kilowatt Hours per month:	7.4 cents/kWh
Above 600 kWh:	8.7 cents/kWh

Table 1. 2012-2013 Ontario Time of Use Electricity Rates

The easiest way for scheduling these tasks that comes to mind would usually be the enumeration algorithm. However, since we have hundreds of tasks for multi-users and need to schedule them on at least 24 hours, the enumeration algorithm would cost a large amount of time to accomplish the job. This is due to the number of repeating sub problems that would grow exponentially while the size of the input is increasing. Fortunately, our proposed dynamic programming based algorithm is especially useful to deal with this kind of problems. Hence, it is very easy for us to abandon the way using the enumeration algorithm, even though using the enumeration algorithm can get the global optimal solution.

Back to the point, let ***Sched (E, C)*** be the main function of our scheduling. ***E*** denotes the current total energy consumption, while ***C*** denotes the current total cost. Thus, we obtained Table 2 for the optimization process for each task, assuming the start time is 12:00am.

Hour / Ways	12:00am	1:00am	2:00am	3:00am	……
1	(0,C1)	(0,C4)	(0,C11)	……	……
2	(E1,C2)	(E1,C5)	(E1,C12)	……	……
3	(E2,C3)	(E2,C6)	(E2,C13)	……	……
4		(E1+E1,C7)	(E1+E1,C14)	……	……
5		(E1+E2,C8)	(E1+E2,C15)	……	……
6		(E2+E1,C9)	(E2+E1,C16)	……	……
7		(E2+E2,C10)	(E2+E2,C17)	……	……
8			(E1+E1+E1,C18)	……	……
9			(E1+E1+E2,C19)	……	……
10			(E1+E2+E1,C20)	……	……
11			(E1+E2+E2,C21)	……	……
12			(E2+E1+E1,C22)	……	……
13			(E2+E1+E2,C23)	……	……
……			…….	……	……

Table 2. Scheduling the tasks without pruning

E1 and **E2** denote the power level choices of the task respectively. For the first hour, we could schedule the task in 3 ways: no schedule, schedule it in power level 1, or schedule it in power level 2. And each of these schedules would cause a cost named $\mathbf{C_i}$. Since each schedule is based on the previous solution, we have a cost when this task does not schedule on this hour, or $(0, C_x)$. We could easily find out that the solution numbers for ***j***th hour is 3^{2*i}+1, and this number we obtained without simplification. With simplification, some of the solution could be abandoned due to its lower energy consumption and higher cost in the same row. For example, looking at the second row, at this hour, there are 2 outputs of cost for total power consumption **E1**+**E2**. So, if $\mathbf{C_8}$ is bigger than$\mathbf{C_7}$, which means the

5th way costs more, compared to the 6th way, while consuming the same power at this time. Thus, the 5th way at 1:00am should be abandoned. For the result, the solution following this abandoned solution at the next hour should all be deleted, since for those child solutions, their parent solution is no longer an optimal one. Then, we do the same comparison for all the solutions in the same column mutually as long as no higher cost with lower energy consumption solutions remain. That is a huge simplification for this table, and the new one, as an example, is shown below in Table 3.

Hour / Ways	12:00am	1:00am	2:00am	3:00am	……
1	(0,C1)	(0,C4)	(0,C11)	……	……
2	(E1,C2)	(E1,C5)	(E1,C12)	……	……
3	(E2,C3)	(E2,C6)	(E2,C13)	……	……
4		(E1+E1,C7)	(E1+E1,C14)	……	……
5		(E2+E1,C9)	(E1+E2,C15)	……	……
6		(E2+E2,C10)	(E2+E2,C17)	……	……
7			(E1+E1+E1,C18)	……	……
8			(E1+E1+E2,C19)	……	……
9			(E2+E1+E2,C20)	……	……
10			(E2+E2+E1,C21)	…	…..
11				…..	……
…..				…..	…..

Table 3. Scheduling the tasks with pruning

For ***j***th hour, j∈[1,24], the combination of energy is (***a****E1+***b****E2), and ***a*** <= ***j*** , ***b***<= ***j***. Hence, we could find out that the solution numbers for ***j***th hour become no more than$(\mathrm{j}+1)^2$. Assume that in the real life, the power level would usually be

an integer number. Let C equals to the maximum value in the E set, then the solution number for *j*th hour becomes no more than C*j.

After one task scheduling, the next task scheduling which is based on the previous one, would be easy and we would like to schedule it in the same way as the previous one was done. After finishing all the tasks for individual customers, we would like to re-schedule the tasks from the beginning for better scheduling. The advantage of doing so is that more precise optimization would occur. What we need to do is just removing one task from the scheduler, recalculating the total energy consuming for each of hours the task used to schedule on, and re-schedule the task into the scheduler just like a new task based on the rest tasks. Multi-user tasks scheduling is almost the same as the individual user task scheduling. In multi-user tasks scheduling, however, we just add the remaining tasks to the previous user's scheduler and repeat the work, which is shown through the example below.

3. Example

Let us assume that we have 9 tasks, and each of them has 5 factors (start time=0, end time=6, power level 1=1, power level 2=2, total energy=4). The hourly price for all hours is $1 if the total energy consumption for each hour is below or equal to 1 and the hourly price would be $2 if the total energy consumption for each hour is above 1. Then, the scheduling for the first task is shown below.

*j*th hour / ways	1st(0)	2nd(1)	3rd(2)	4th(3)	5th(4)	6th(5)	7th(6)
1	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
2	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
3	(2,4)	*(2,4)	*(2,4)	*(2,4)	*(2,4)	*(2,4)	*(2,4)
4		(2,2)	(2,2)	(2,2)	(2,2)	(2,2)	(2,2)
5		*(3,5)	*(3,5)	*(3,5)	*(3,5)	*(3,5)	*(3,5)
6		(3,5)	(3,3)	(3,3)	(3,3)	(3,3)	(3,3)
7		(4,8)	*(4,6)	*(4,6)	*(4,6)	*(4,6)	*(4,6)
8			(4,6)	(4,4)	(4,4)	(4,4)	(4,4)
9							

* need to be pruned

Table 4. Example of pruning for task 1

The red ones are the final schedule for task 1 which is the optimum solution as there is no other task scheduled. Now, we need to schedule the next task depending on the task 1's scheduling.

*j*th hour / ways	1st(0) Prev 1	2nd(1) Prev 1	3rd(2) Prev 1	4th(3) Prev 1	5th(4) Prev 0	6th(5) Prev 0	7th(6) Prev 0
1	(0,1)	(0,2)	(0,3)	(0,4)	(0,4)	(0,4)	(0,4)
2	(1,4)	(1,5)	(1,6)	(1,7)	(1,5)	(1,5)	(1,5)
3	(2,6)	(2,7)	(2,8)	(2,9)	*(2,8)	*(2,8)	*(2,8)
4		*(2,8)	*(2,9)	*(2,10)	(2,8)	(2,6)	(2,6)
5		*(3,10)	*(3,11)	*(3,12)	*(3,11)	*(3,9)	*(3,9)
6		(3,10)	(3,11)	(3,12)	(3,10)	(3,9)	(3,9)
7		(4,12)	(4,13)	(4,14)	*(4,13)	*(4,12)	*(4,12)
8			*(4,14)	*(4,15)	(4,13)	(4,11)	(4,10)
9							

* need to be pruned

Table 5. Example of pruning for task 2

Prev n means in this hour period, ***n*** units' power has already been scheduled due to previous scheduling. After scheduling, we figure out that the total cost is lowest when we arranged the task 2 uniformly in power level 1 on the last four time periods. Repeat the scheduling using our proposed algorithm for the rest tasks. After one time full-tasks scheduling, we arrange all the tasks in the way shown in the following table.

*j*th hour / tasks	1st(0)	2nd(0)	3rd(2)	4th(3)	5th(4)	6th(5)	7th(6)
1	1	1	1	1			
2				1	1	1	1
3	2	2					
4	2	2					
5	2	2					
6	2	2					
7	2	2					
8	2	2					
9	2	2					

Table 6. All tasks scheduling

In order to optimize the arrangement, we prefer to run it again. First, we should "delete" task 1 from the table. Then, depending on the existing arrangement, reschedule task 1 and reschedule the rest after it. Finally, we would obtain the following arrangement.

*j*th hour / tasks	1st(0)	2nd(0)	3rd(2)	4th(3)	5th(4)	6th(5)	7th(6)
1	2	1	1				
2				1	1	1	1
3	2	2					
4	2	2					
5	2	2					
6	2	2					
7	2	2					
8	2	2					
9	2	2					

Table 7. All tasks rescheduling

Then, we met equilibrium, no matter how many further rescheduling we run. For this simple example, this arrangement is one of the optimum solutions. Most of the time, we may not obtain the optimum solution using this algorithm, but we could get a solution very close to the optimum one after rescheduling λ times. In order to determine this λ, we could compare the lowest cost in λth time and in $(\lambda-1)$th time. If the difference is below a certain value, like 1%, which the consumers may accept, the controller shall stop rescheduling the tasks. Then we could do this progress for all the users in the community, just treating multi-users as multi-tasks. Finally, equilibrium would be met and all the users in the community would satisfy the result [9].

4. Nash equilibrium

Our proposed algorithm is to find Nash equilibrium. “In game theory, the Nash equilibrium is a solution concept of a non-cooperative game involving two or more players, in which each player is assumed to know the equilibrium strategies of the other players, and no player has anything to gain by changing only his own strategy unilaterally” [10]. In the system, like our “smart community”, where

multiple independent consumers are trying to optimize their own utility unilaterally, Pareto Optimality is very hard to achieve. Pareto optimality or Pareto efficiency means that in the system, “no one can be made better off without making at least one individual worse off [11].” Hence, Nash equilibrium is fair and reasonable among consumers since everyone in the system is choosing the individual optimal strategy depending on others’ choices. Following is the normal math definition of Nash equilibrium. In a game set G={S1,…,Sn: u1,...un}, there is a Strategy Profile (s1*,…,sn*), where si*∈Si(Si is the strategy set of player i). If any si* is the best strategy to the combination of others’ strategies (s1*,…s*i-1,s*i+1,…,sn*), or ui(s1*,…,s*i-1,s*i,s*i+1,…,sn*) ≥ui((s1*,…,s*i-1,sij*,s*i+1,…,sn*) where sij∈Si, then the strategies(s1*,…,sn*) is a Nash equilibrium of the game set G [12].

Apparently, our algorithm keeps finding the Nash equilibrium until the improvement of the two scheduling solutions for the whole community is less than a certain value. For instance, our solution is implemented in a community with 100 customers. The controller would do the scheduling for the first customer, and repeat the procedure until the last one. The strategy for each scheduling depends on what we have scheduled for the previous ones. After one complete scheduling for these 100 customers, we could obtain a game set G1= $\{S1_1, S2_1,\ldots, S100_1: c1_1, c2_1,\ldots, c100_1\}$, where Si_1 denotes the strategy of customer ***i*** and cj_1denotes the cost of customer ***j*** using its strategy in the 1st scheduling. Then in the 2nd scheduling for the whole community, we would change everyone’s scheduling strategy one by one, depending on the Strategy Profile of the whole community $(S1_2, S2_2, \ldots, Si-1_{2,}Si_1, Si+1_1, \ldots, S100_1)$, to obtain a new Strategy Profile $\{S1_2, S2_2, \ldots, Si-1_{2,}Si_2, Si+1_1, \ldots, S100_1\}$. After a Strategy Profile $\{S1_2, S2_2, \ldots, Si-1_{2,}Si_2, Si+1_2, \ldots, S100_2\}$ is obtained, the controller shall start the 3rd scheduling for the community. Since each individual scheduling strategy is set depending on all the other strategies in the community, the controller must be able to find a game set G_k, in which any Si_k is the best strategy to the combination of others’ strategies (s1,…si-1,si+1,…,sn).

Additionally, the best strategy means the strategy with the lowest cost. It is important to remember that finding the Nash equilibrium does not mean that we should find out Minimum(c1+c2+…+c100). However, this ***k*** value is hard to find since the tasks' number in a community may be very large. In fact, we only need to go through ***λ*** times, like mentioned in the earlier paragraph. After ***λ*** times scheduling, the difference of the total cost between two complete scheduling for the community is less than a very small value. Then, we would obtain a solution that is very close to the Nash equilibrium and the customers would be satisfied.

The algorithm part is completed and therefore I would like to present the hardware and software setup in the next sections. Due to the complexity of the algorithm, I decided to use the CPU core embedded on the FPGA board to do the calculation, or scheduling. The following are flow charts of the whole procedure of scheduling.

Ⅳ. Flow Charts

1. Procedure

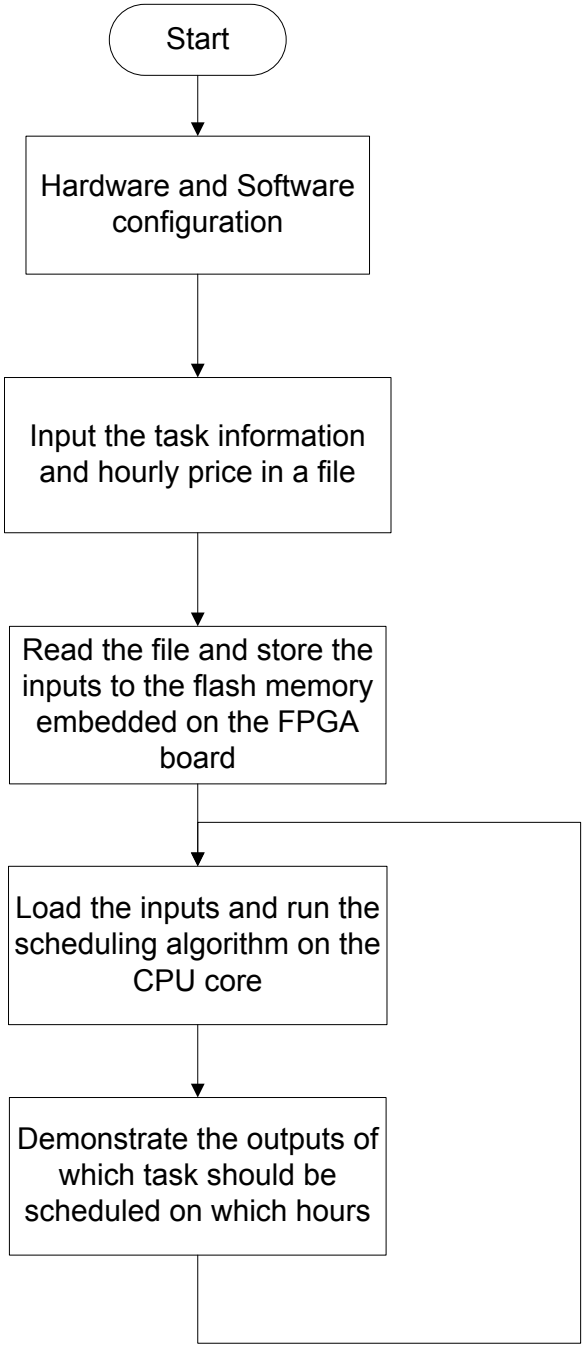


Figure 2a. Flow chart of the procedure

2. Individual user scheduling

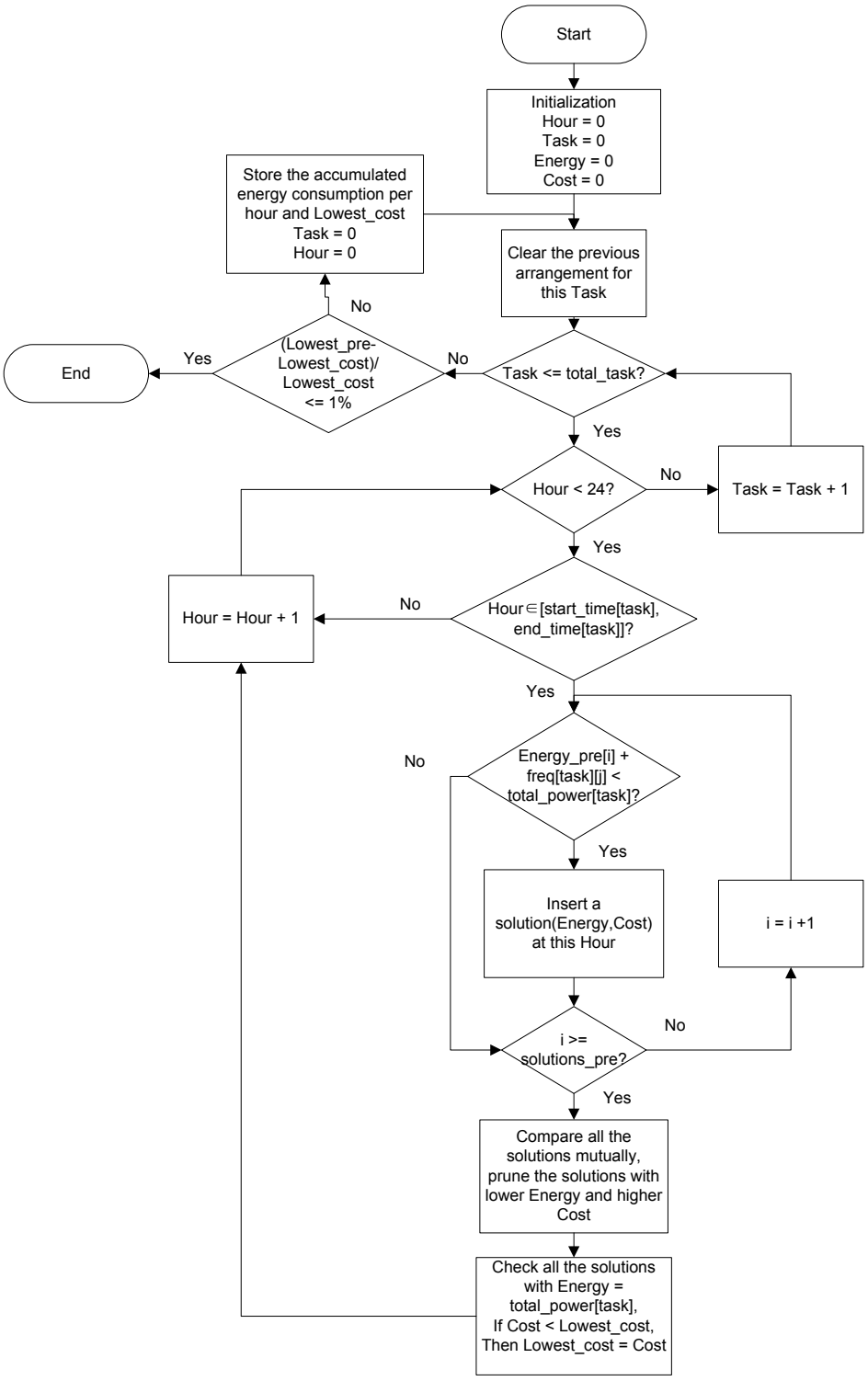


Figure 2b. Flow chart of the algorithm for individual user scheduling

3. Multiple users scheduling

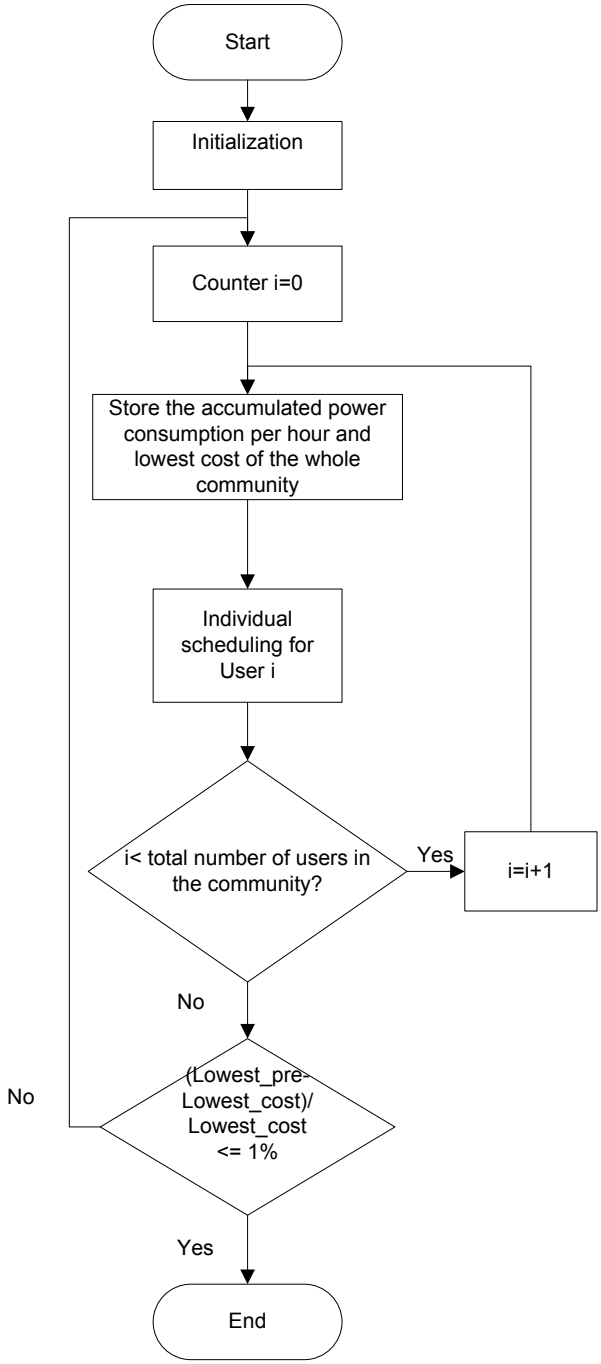


Figure 2c. Flow chart of the algorithm for multiple users scheduling

Ⅴ. FPGA Structure

1. Choosing FPGA board

At the beginning of the project, I chose Altera's DE0 Nano board, which is shown below, as my controller due to its small size of body, good portability, and enough logic elements and comparatively large memory storage. Cyclone IV EP4C22 FPGA, which has 150,000 logic elements, is embedded in the DE0-Nano board. This FPGA education board also has 32MB SDRAM [13]. However, since it has no Flash Memory embedded on the board, it is kind of difficult to load the task information and hourly electricity price, which needs to be input by the operators from the keyboard or from a file. Hence, I finally abandoned this board and used Altera's DE2 board as the MCU. Even though this board is about 10 times larger than DE0-Nano board and only has Cyclone II FPGA core with 68,416 logic elements and 8MB SDRAM, it still fits this project [14].

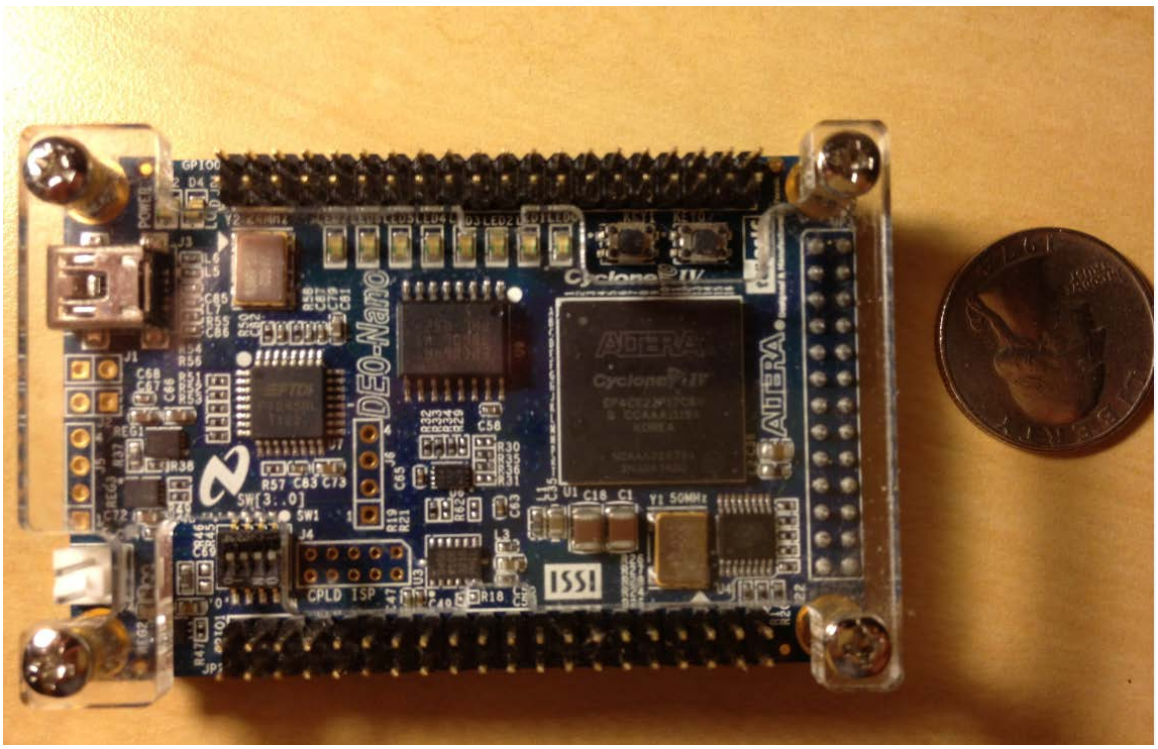


Figure 3a. DE0-Nano board

A photograph of the DE2 board is shown below in Figure 3. It demonstrates the location of the key component. The components with a yellow square frame are used in our project.

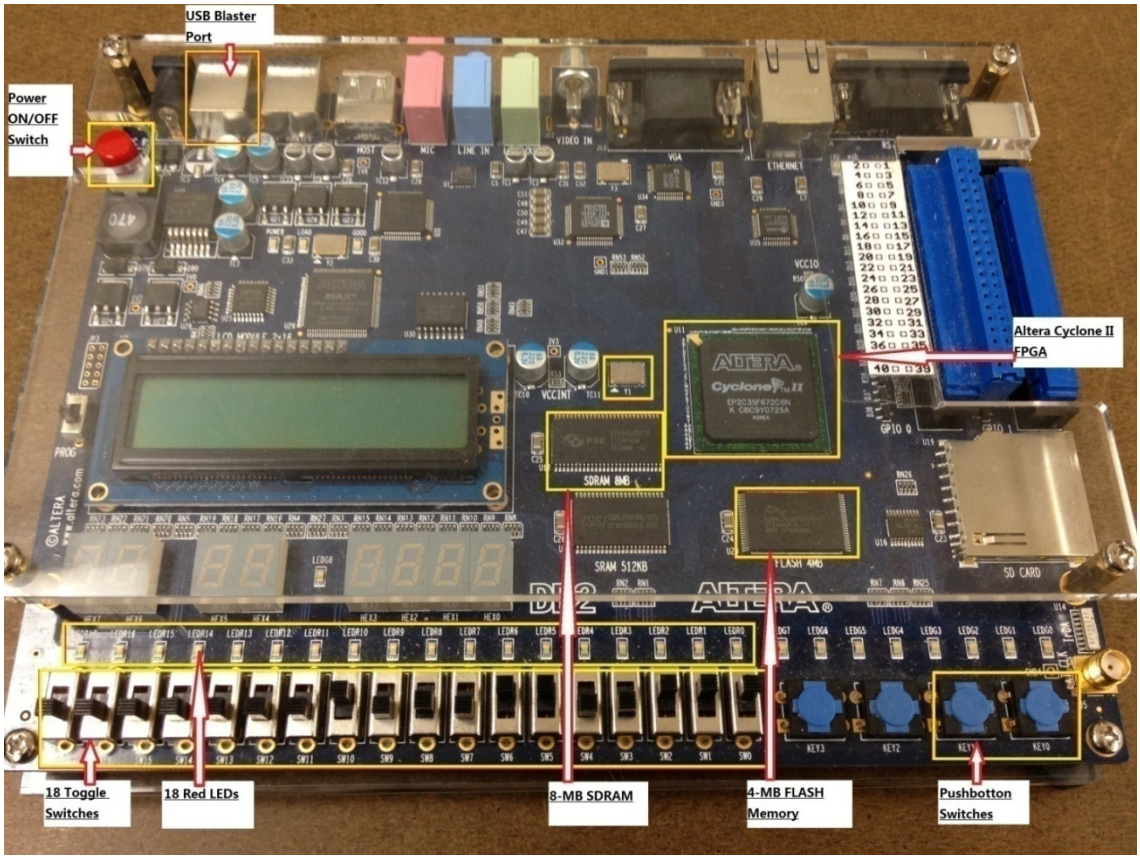


Figure 3b.DE2 board

2. NIOS II processor

In order to implement the comparatively complex algorithm, the NIOS II Processor embedded in Cyclone II FPGA is used in this project. Figure 4, obtained from Altera, shows an example of the architecture of a NIOS II Processor System [15].

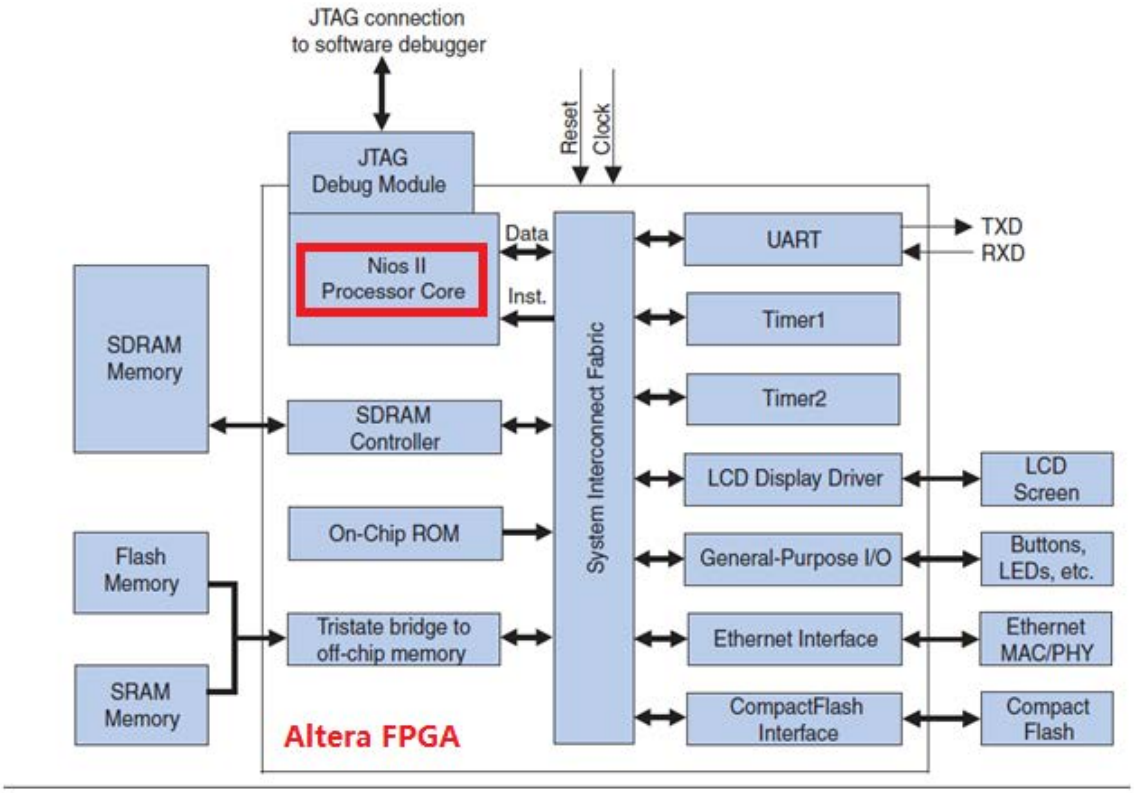


Figure 4.Example of a NIOS II Processor System

This architecture is very important for us to build a System on Programmable Chip (SOPC). I will review this figure later when I use the SOPC builder in Quatus II.

Ⅵ. Instruction of Implementing in Quartus II

1. Schematic

First of all, we need to create the hardware configuration in Quartus II. A new project should be built and after we will create a schematic file as our project's top entity. The overview of the hardware schematic is shown below as the Figure 5a and Figure 5b.

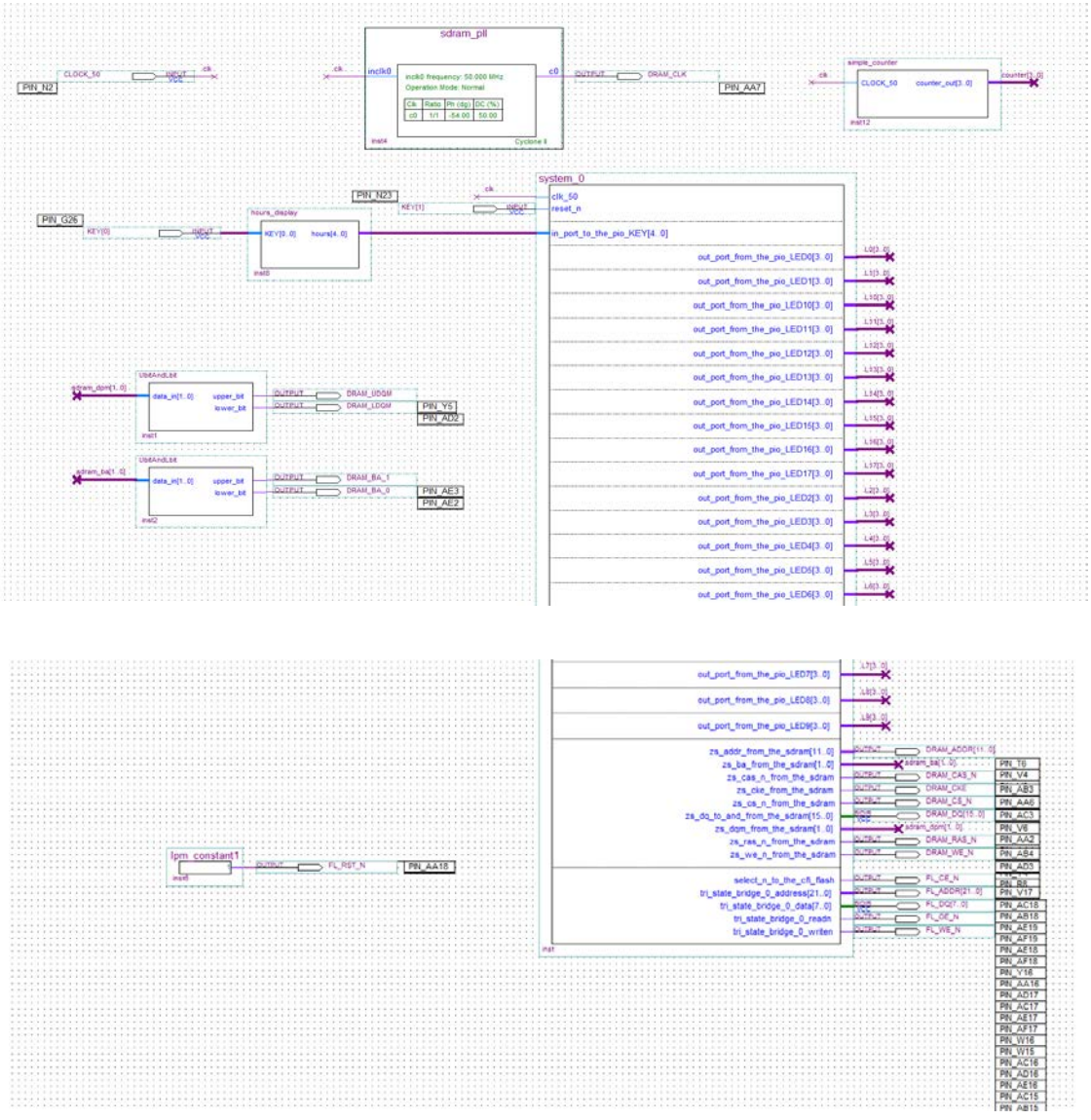


Figure 5a. Overview of the schematic1

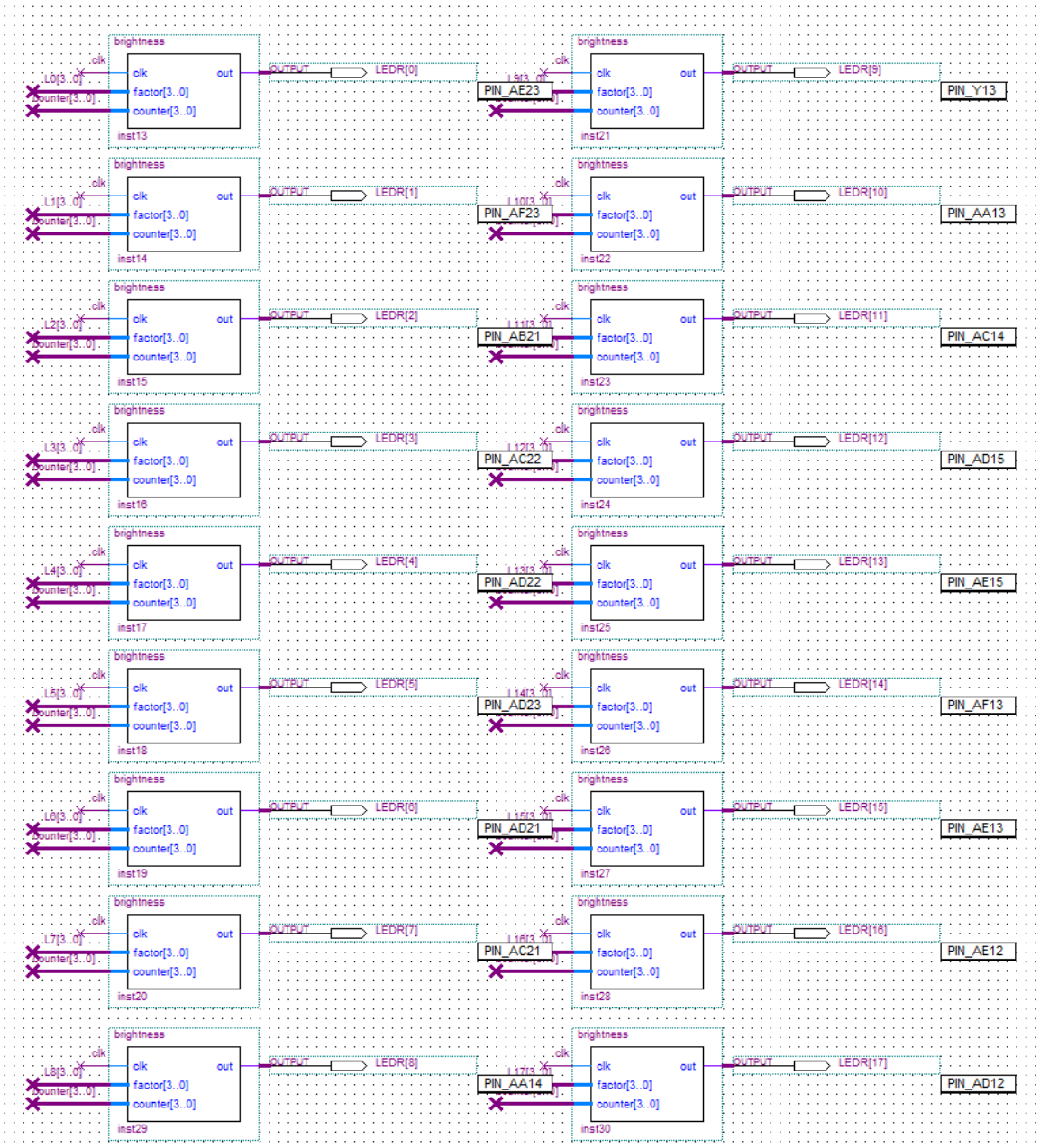


Figure 5.Overview of the schematic2

A system clock is needed, so we will create an input connecting to *CLOCK_50,* which has 50Mhz power level, followed by the wire named clk. This wire could connect to any modules, which need clock sources.

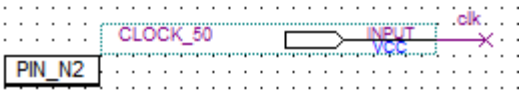


Figure 6.System Clock Source

Use KEY[1] as the *reset* input and KEY[0] to choose the particular hours to demonstrate the tasks scheduled on them.

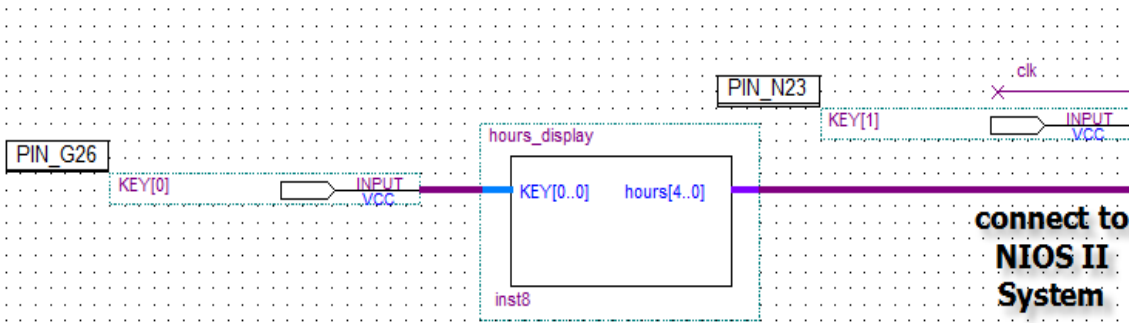


Figure 7.KEY modules

In this case, if KEY[0] is not pushed, the tasks scheduled on *hour[0]* would be displayed on the board by turning on the LEDs. If KEY[0] is pushed one time, the tasks scheduled on *hour[1]* would be displayed and so on so forth. In this simulation, I set the number of hours up to 24 and the output has 5 bits. The Verilog code of this KEY controlling module is shown below.

```
module hours_display(
                                    KEY,
                                    hours
                            );
input [0:0]KEY;
output reg [4:0]hours;
always @(negedge KEY[0])
        if (hours>5'd22) hours<=0;
        else hours<=hours+1;
endmodule
```

Following is the demonstrated used LEDs output module and its brightness control used counter module. Each LED represents a task. If the power level of the running task is high, the LED would become brighter, otherwise it would become fainter. In our project, I assume that the power level range of all the tasks is between 1 and 15. *Counter* is used to count the clock. Since it only has 4 bits, *counter* will keep increasing between 0 and 15 in 50 Mhz frequency as the same as the system clock. Hence, the brightness of the LED would be set proportionally depending on the power level of the corresponding task, since during each 16 units time period the LED would be turned on in *factor* number unit time period and be turned off in (16 - *factor*) number unit time period.

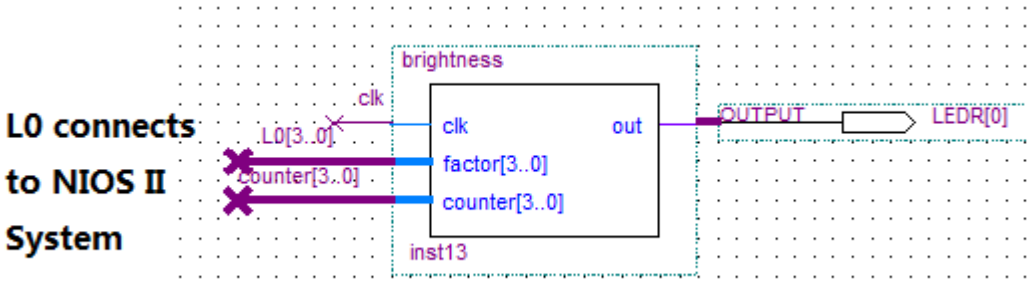


Figure 8.LEDs output module

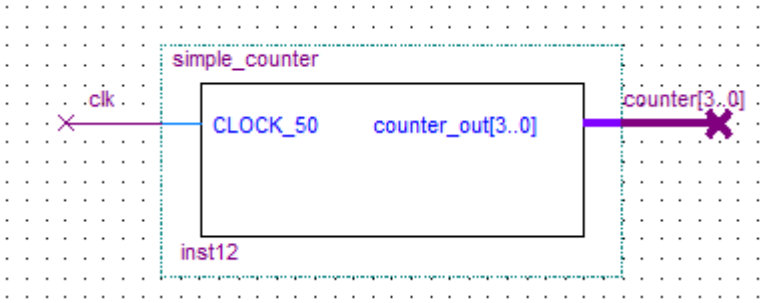


Figure 9.Brightness control used counter module

The Verilog codes for these two module are shown below.

```
module simple_counter (
                                    CLOCK_50,
                                    counter_out
                                );
input           CLOCK_50;
output reg  [3:0]  counter_out;
always @ (posedge CLOCK_50)
        begin
                counter_out <= #1 counter_out + 1;
        end
endmodule
```

```
module brightness (
                                        clk,
                                        factor,
                                        counter,
                                        out
                                    );
input           clk;
input           [3:0]factor;
input       [3:0]counter;
output reg  out;
always @ (posedge clk)
        if (factor==0) out<=0;
        else if (counter<=factor) out <=1; else out <=0;
endmodule
```

2. SOPC builder

The most important part is the CPU core of this system. We will use *SOPC builder* to create an on chip system, which is shown below.

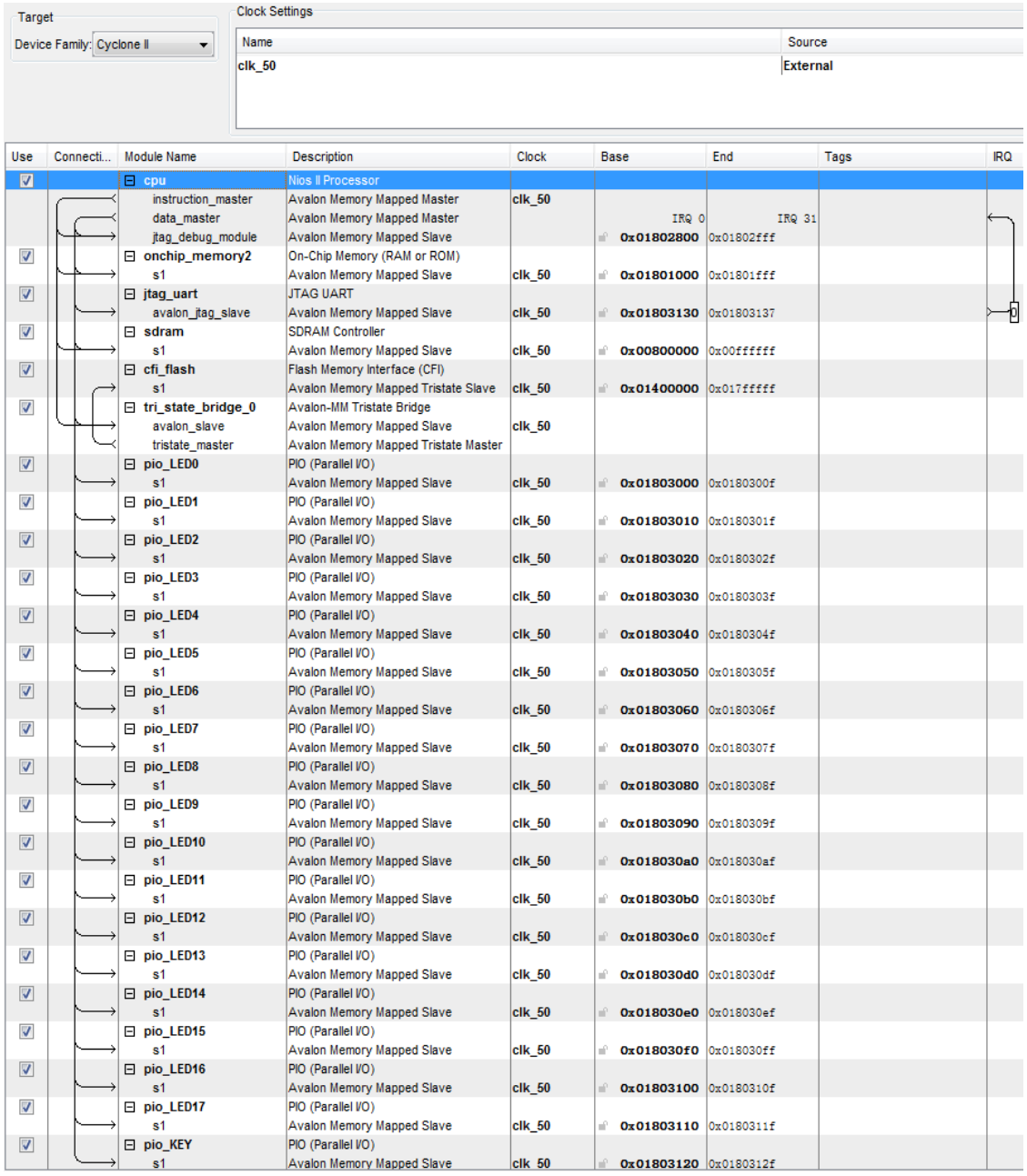


Figure 10.SOPC builder setup

CPU is the core and it connects to all of the other parts of system. *Jtag_uart* is used for the FPGA board to communicate with the PC. For example, through the *jtag_uart* interface, the input file could be stored from the PC to the Flash Memory embedded on the FPGA board. Then, we created several I/O interfaces like *pio_key* and *pio_ledx* to connect the CPU core to the I/O pins on the FPGA board. It's painless to quickly add a System ID component, named *sysid*, to keep track of whether the BSP driver package currently used is still compatible with the hardware we are trying to run it on. It is also painless to add an Onchip_memory. Actually we don't need to use it since I decided to store all the instructions, library, etc. in the *SDRAM*. Why I would like to use *SDRAM* as my CPU memory is that the memory space in onchip_memory is too small to support the full c library. I could only use the full c library to use file operation in NIOS II. In Figure 4, it shows that we need an *SDRAM* controller to connect the real *SDRAM* memory and to control its behavior. For each particular FPGA chip, we should consider the configuration of the *SDRAM* controller. Datasheet of the target board is needed here.

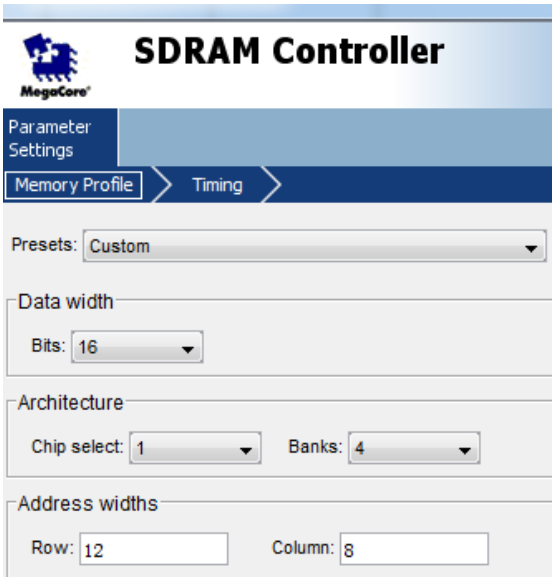


Figure 11.SDRAM Controller Memory Profile configuration

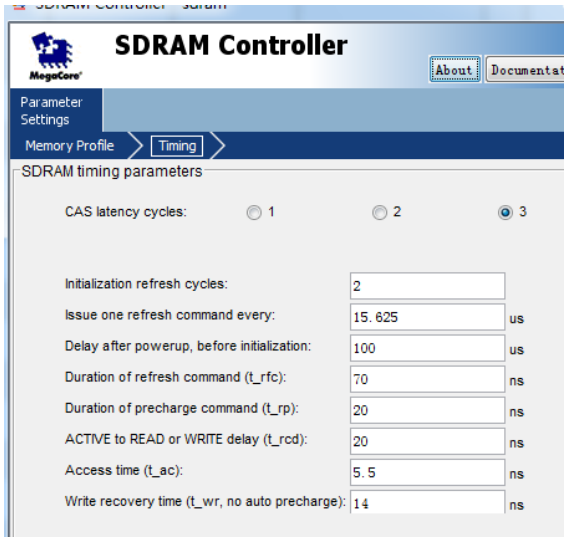


Figure 12.SDRAM Controller Timing Configuration

Flash Memory Interface, named cfi_flash, is also added in the SOPC. In order to correctly use Flash Memory, indicating in Figure 4, tri_state_bridge is also required, and the slave side should connect to the CPU and the master side should connect to the Flash Memory. Flash Memory Interface's setup is shown below.

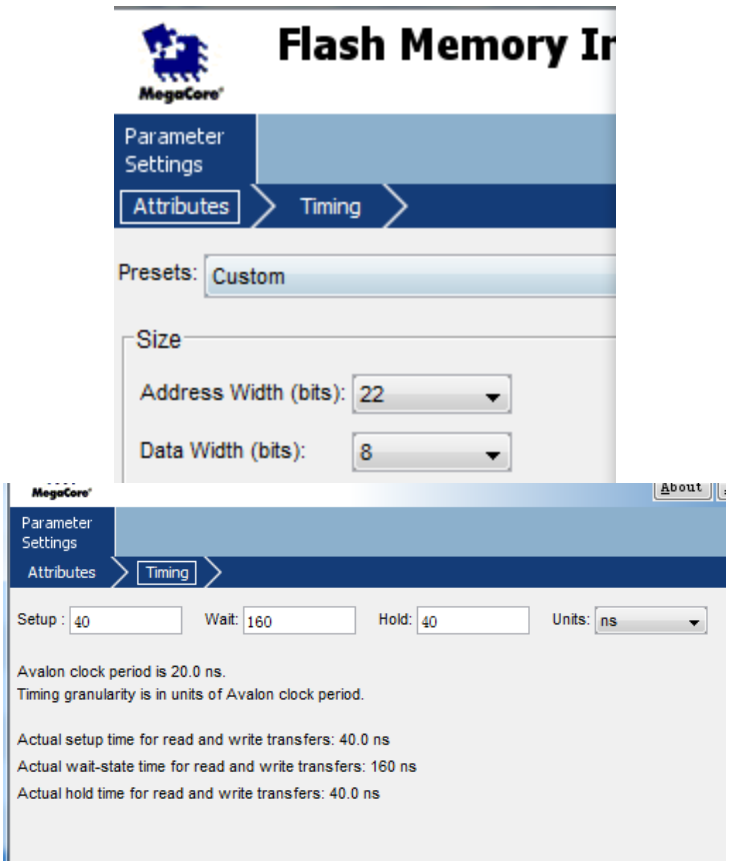


Figure 13.Flash Memory Interface Configuration

The NIOS II CPU core is setup as shown below, notice that *SDRAM* is chosen.

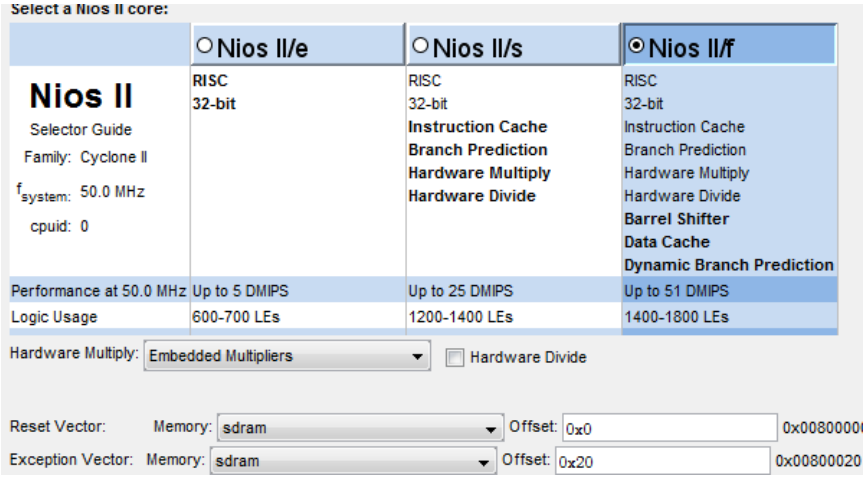


Figure 14.NIOS II core overview

After we created all of the components of the system and added a system 50Mhz clock, we need to refresh the system and Auto-Assign Base Addresses for all of the components and then generate it.

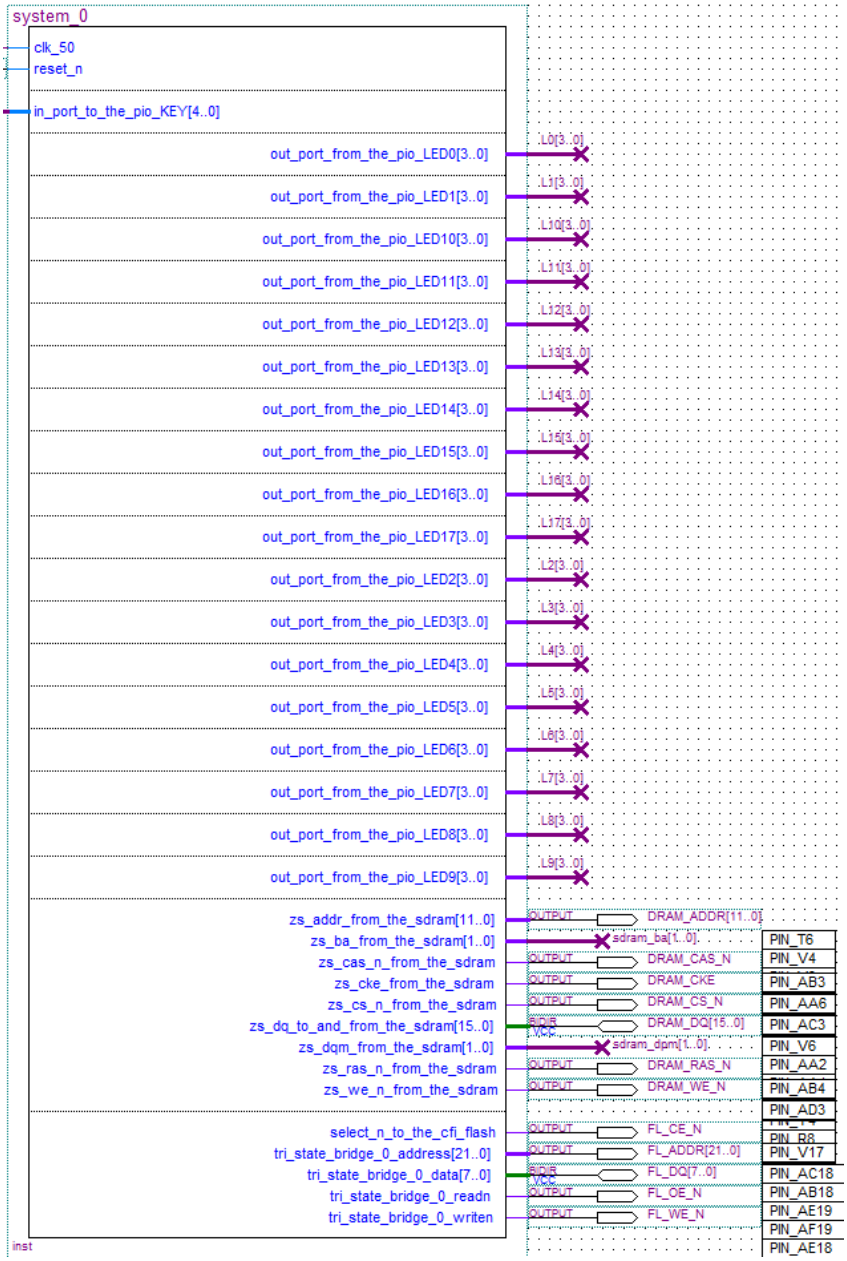


Figure 15.SOPC module

The figure above is the quick look of the SOPC module. Since we only create a SDRAM controller in the SOPC, we have to connect the controller pins to the

input pins of the SDRAM embedded on the FPGA board. Followed by the controller pins of *SDRAM* is the flash controller pins. Just a reminder, the Flash Memory reset pin must connect to the reset pin or simply a constant 0.

3. Clock source and Phase Lock Loop(PLL)

In order to make sure all the signals are stable on the clock edge, a Phase Lock Loop (PLL) module is needed to create a second clock signal into SDRAM. We should also consider the setup of the PLL module, since that may cause an error when building the project if some values are incorrect for each particular FPGA board. The setup for DE2 board, or Cyclone II EP2C35F672C6 FPGA chip, is shown below.

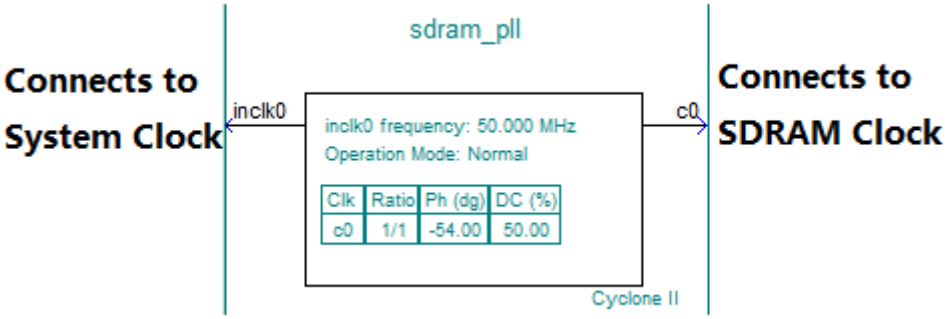


Figure 16.PLL module

Ⅶ. Hardware Testing

1. Example 1

I would like to have two tests to demonstrate my work. First one is the example I analyzed in the part **Algorithm.** Assume there are 10 tasks and we are supposed to assign them in 7 hours. First, since we need to input our information into the FPGA board using txt format file, a Zip file named files.zip should be created under scheduling_0_syslib folder, including 2 txt format files named file1.txt and file2.txt storing the tasks information and hourly price information respectively. Due to the software work, the names of these files cannot be modified and the txt files cannot be compressed. See the following figure.

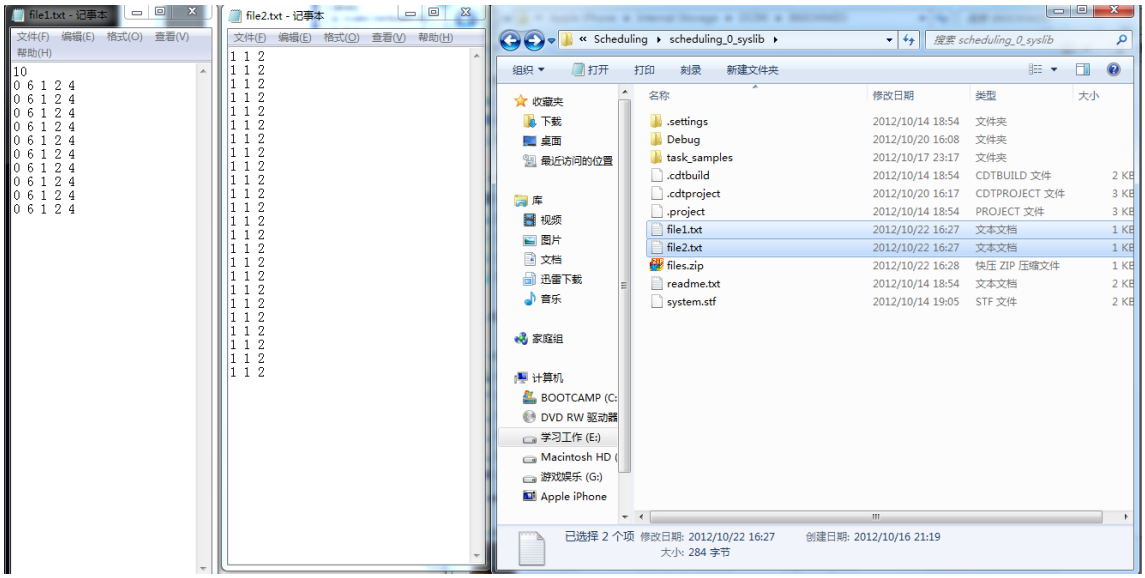


Figure 17.Input information of Example 1

Then the result run in FPGA shows below from hour 0 to hour 6. We could see that the tasks with higher power level would be brighter.

Figure 18. Hourly power consumed of Example 1

2. Example 2

Another example is shown below

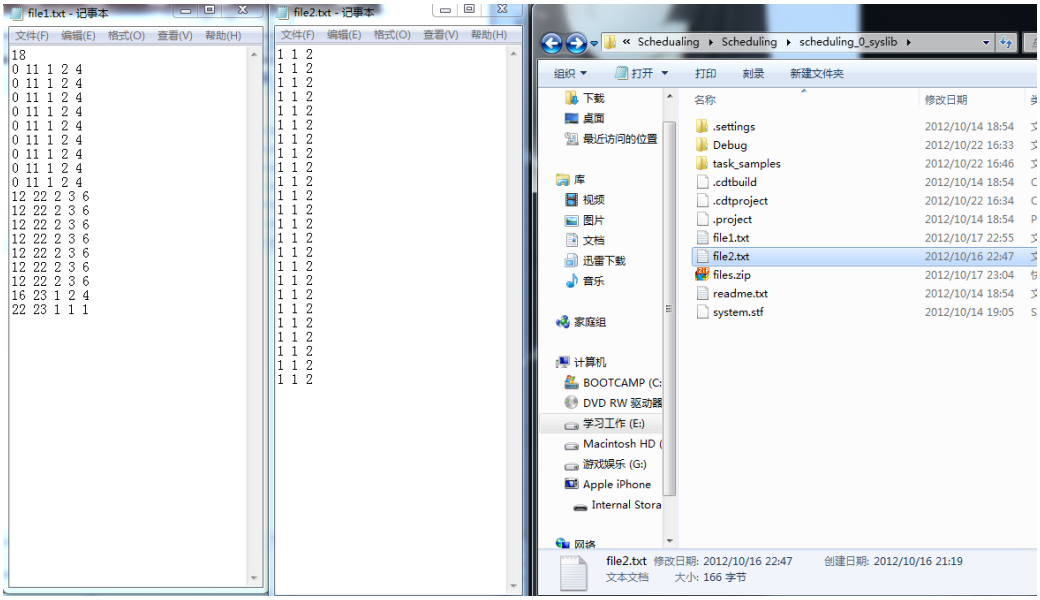
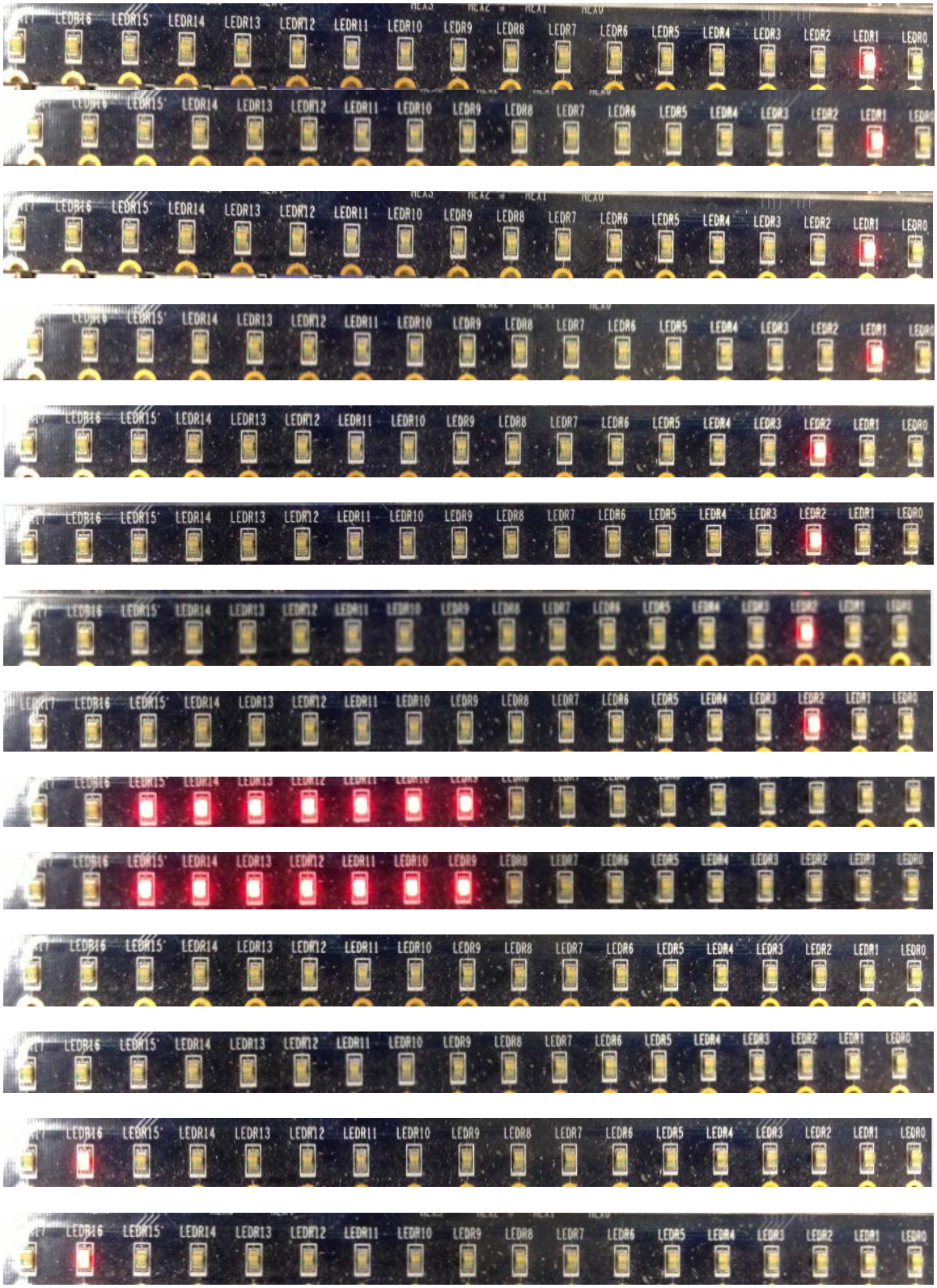


Figure 19. Input information of Example 2

And the hardware testing result is shown below from hour 0 to hour 23.



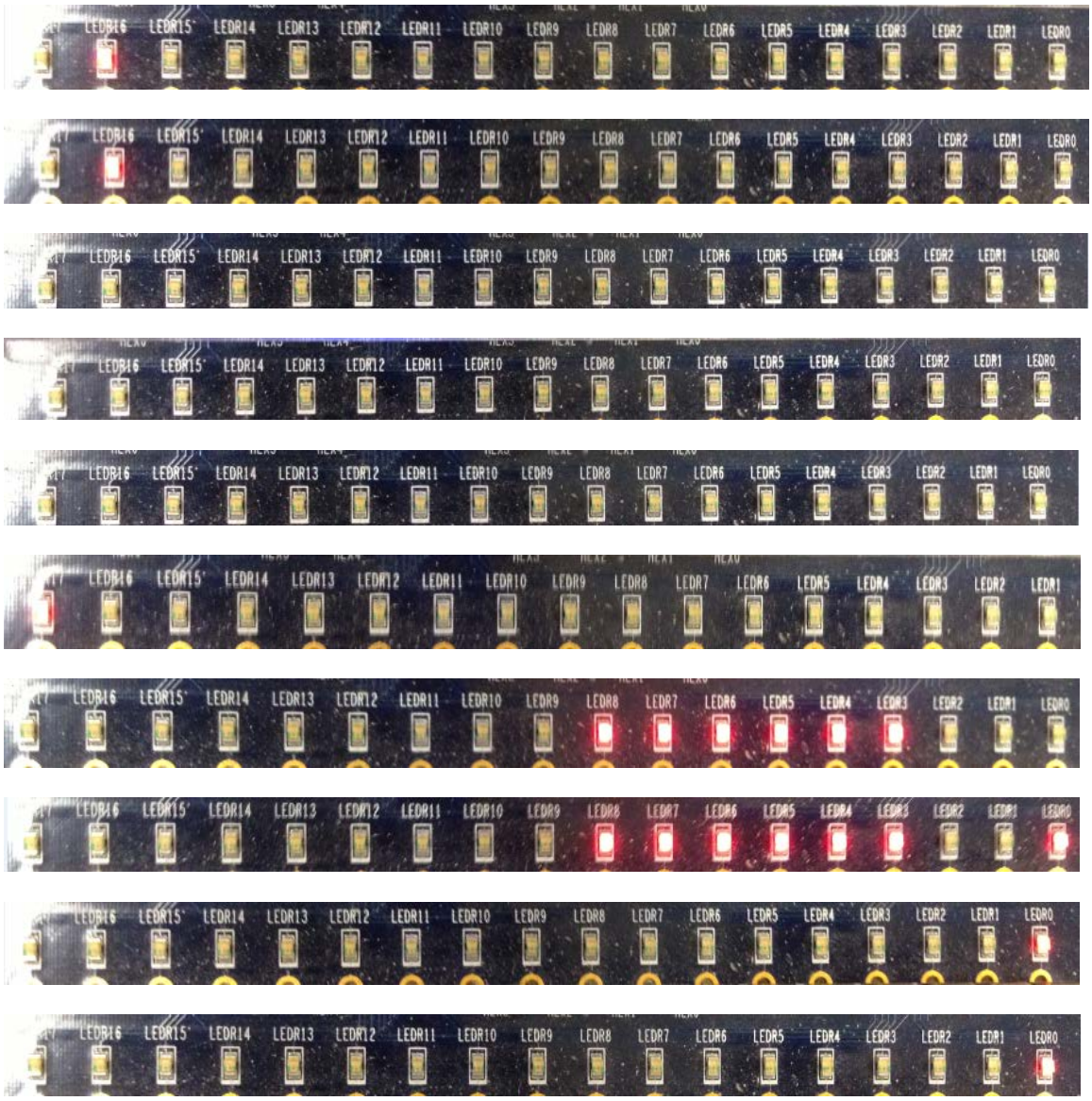


Figure 20. Hourly power consumed of Example 2

Since there are only 18 onboard LEDs, I could not demonstrate the tasks with more than 18 in this way. I would like to show the debug result when there are more than 18 tasks, like 99 tasks, and multiple users below.

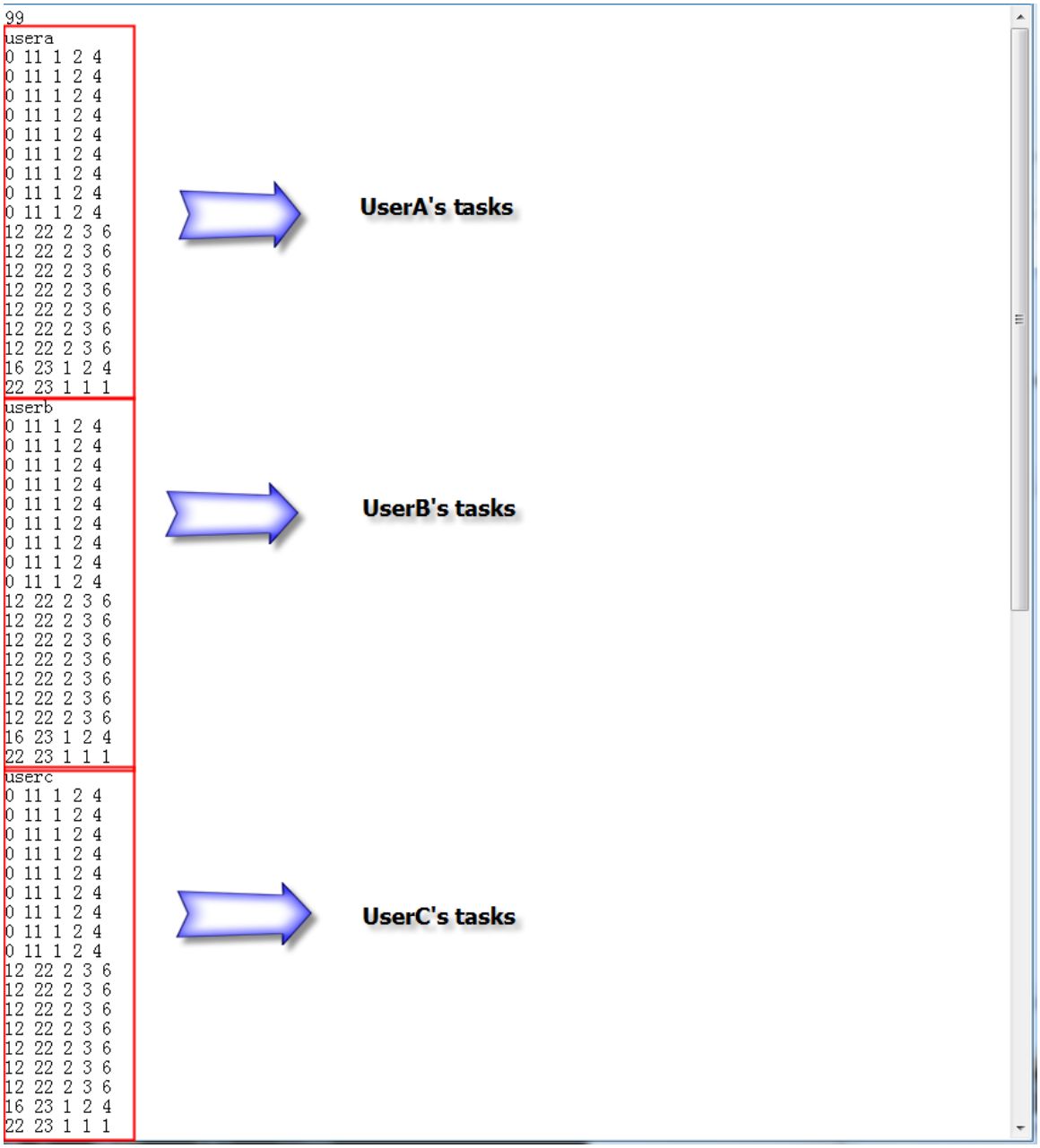


Figure 21. Multiple users_1

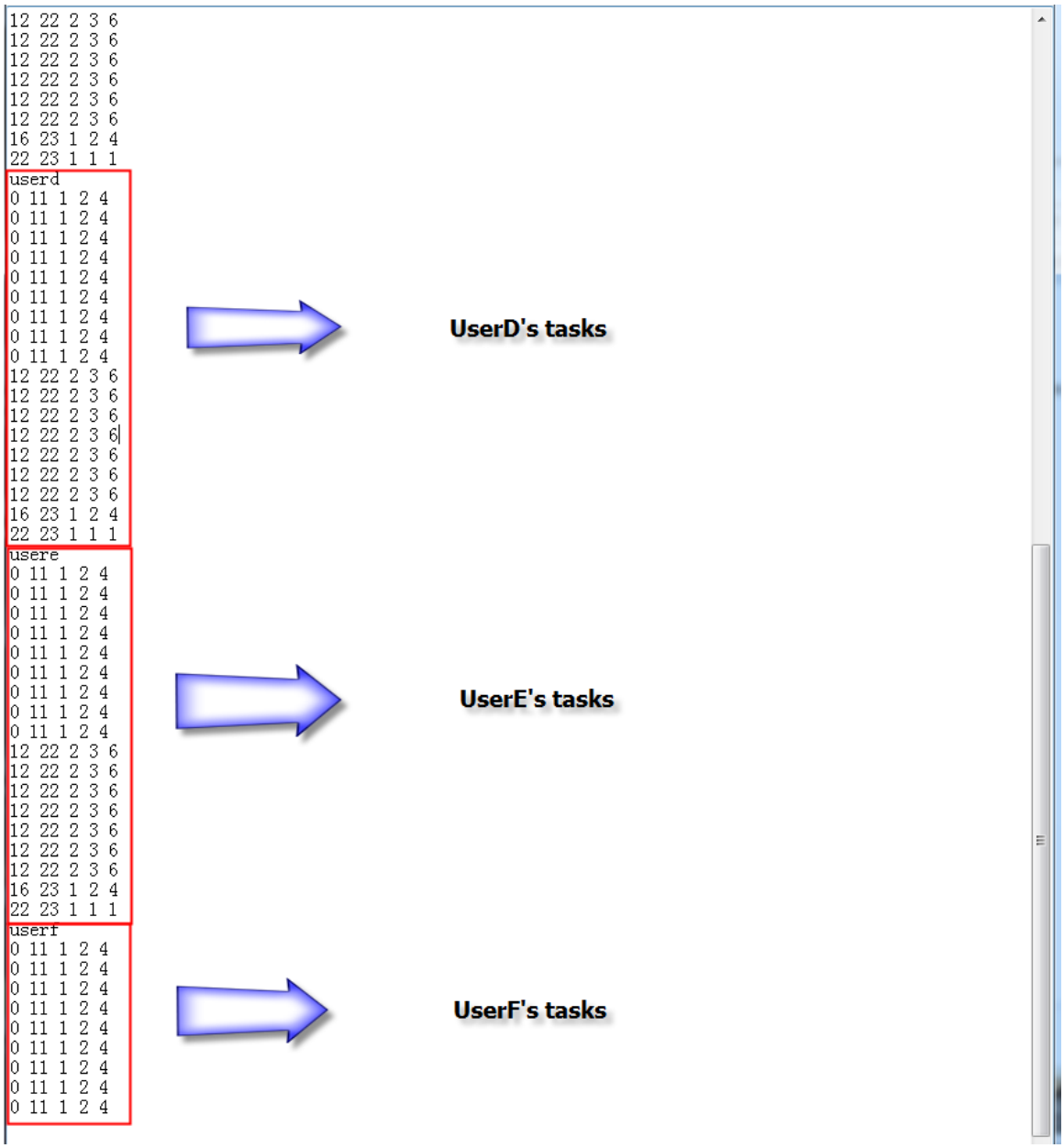


Figure 21. Multiple users_2

power	
power[0]	102
power[1]	104
power[2]	1
power[3]	1
power[4]	1
power[5]	1
power[6]	1
power[7]	1
power[8]	1
power[9]	1
power[10]	1
power[11]	1
power[12]	105
power[13]	105
power[14]	0
power[15]	0
power[16]	1
power[17]	1
power[18]	1
power[19]	1
power[20]	1
power[21]	1
power[22]	8
power[23]	11

Figure 22. Accumulated power level for each hours

cost_total	886

Figure 23. The total cost

It cost about 2 seconds to obtain the result if the total task number is 99.

3. Example 3

To demonstrate the effectiveness, I would like to present the simulation of household appliances. The power use for each smart appliance is consulted from the manuals [16][17][18][19].

Appliance	Start time	End time	Power level 1 (Watt)	Power level 2 (Watt)	Total power Consumption (KJ)
Mini Oven	10:00	12:15	900	1800	1620
Rice Cooker	11:00	12:15	400	600	1080
Clothes Washer	8:00	14:30	400	600	1800
Clothes Dryer	14:30	17:00	2650	5300	19080
Refrigerator	0:00	23:45	720	720	25920
Vacuum Cleaner	0:00	18:45	1000	1200	3600
Dishwasher	13:15	18:45	1200	2400	2160
Water Pump	0:00	23:45	250	1000	2700

Table 5. Input information of household appliances

About the price scheme for a day, we could use the Ontario Time of Use Electricity Rates for Winter I mentioned in **Algorithm** part, which is shown below. (For better calculation and demonstration, I will round the decimal price to the nearest integer)

From	To	Winter Rate(Nov-Apr)
7:00 AM	8:00 AM	12 cents/kwh
8:00 AM	9:00 AM	12 cents/kwh
9:00 AM	10:00 AM	12 cents/kwh
10:00 AM	11:00 AM	12 cents/kwh
11:00 AM	12:00 PM	10 cents/kwh
12:00 PM	1:00 PM	10 cents/kwh
1:00 PM	2:00 PM	10 cents/kwh
2:00 PM	3:00 PM	10 cents/kwh
3:00 PM	4:00 PM	10 cents/kwh
4:00 PM	5:00 PM	10 cents/kwh
5:00 PM	6:00 PM	12 cents/kwh
6:00 PM	7:00 PM	12 cents/kwh
7:00 PM	8:00 PM	6 cents/kwh
8:00 PM	9:00 PM	6 cents/kwh
9:00 PM	10:00 PM	6 cents/kwh
10:00 PM	11:00 PM	6 cents/kwh
11:00 PM	Midnight	6 cents/kwh
Midnight	1:00 AM	6 cents/kwh
1:00 AM	2:00 AM	6 cents/kwh
2:00 AM	3:00 AM	6 cents/kwh
3:00 AM	4:00 AM	6 cents/kwh
4:00 AM	5:00 AM	6 cents/kwh
5:00 AM	6:00 AM	6 cents/kwh
6:00 AM	7:00 AM	6 cents/kwh

Table 6. 2012-2013 Ontario Time of Use Electricity Rates for winter

Since we measured the day as 96 quarter hours in this simulation, we have 96 time periods. For example, time period 0 represents the period from midnight to 12:15 AM, or time period 16 represents the period from 4:00 AM to 4:15 AM. In Ontario, “in the winter months, the higher electricity price is charged for consumption above 1,000 kwh.” [8] As shown in Table 1, usually the higher electricity price is about 1 cent/kwh higher than the lower price. Hence, in our simulation, I will consider that if the quarterly power use was over

$$\frac{1{,}000\frac{kwh}{month}}{30\frac{days}{month}\times 24\frac{hours}{day}\times 4\frac{quarters}{hour}} \approx \frac{0.35kwh}{quarter} = \frac{1260kJ}{quarter}$$

the price would be 1 more cent plus the original price. The unit of power level shown in Table 5 is Watt, and each of the time period is a quarter. Hence, we need to convert $^{J}/_{S}$ to $^{J}/_{quarter}$, which means each power level need to multiply by 60×15 $^{S}/_{quarter}$. Therefore, the input information for this simulation is shown below.

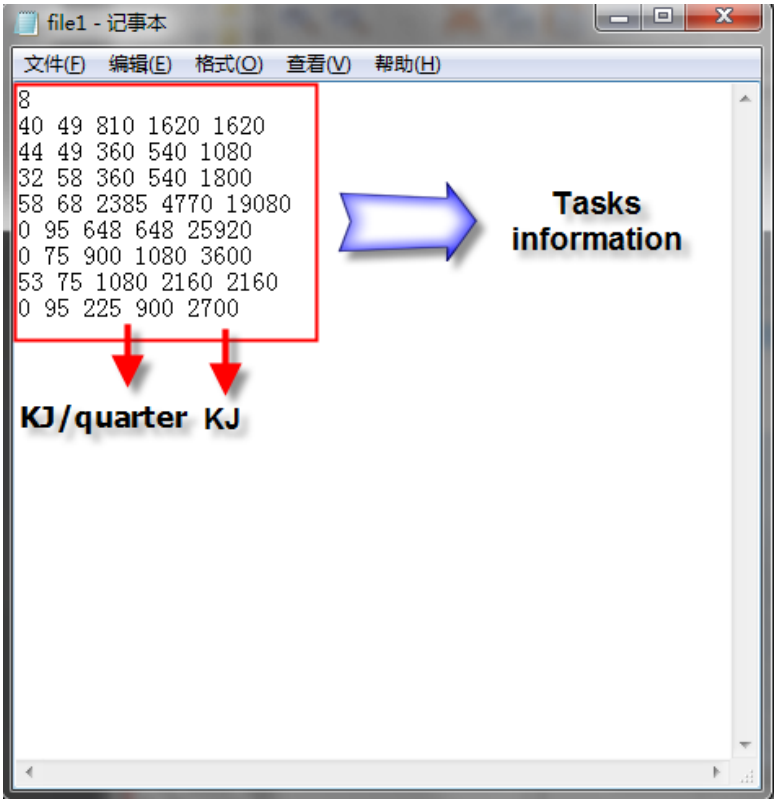


Figure 24. Input tasks information

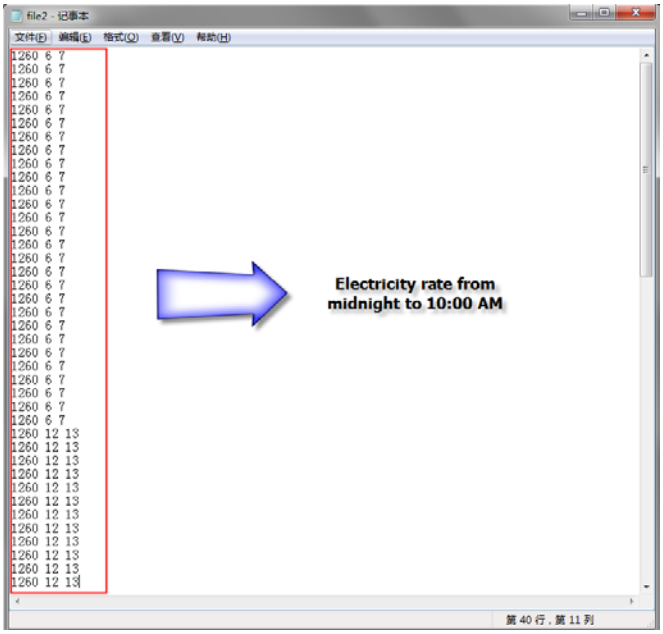


Figure 25a. Input price information

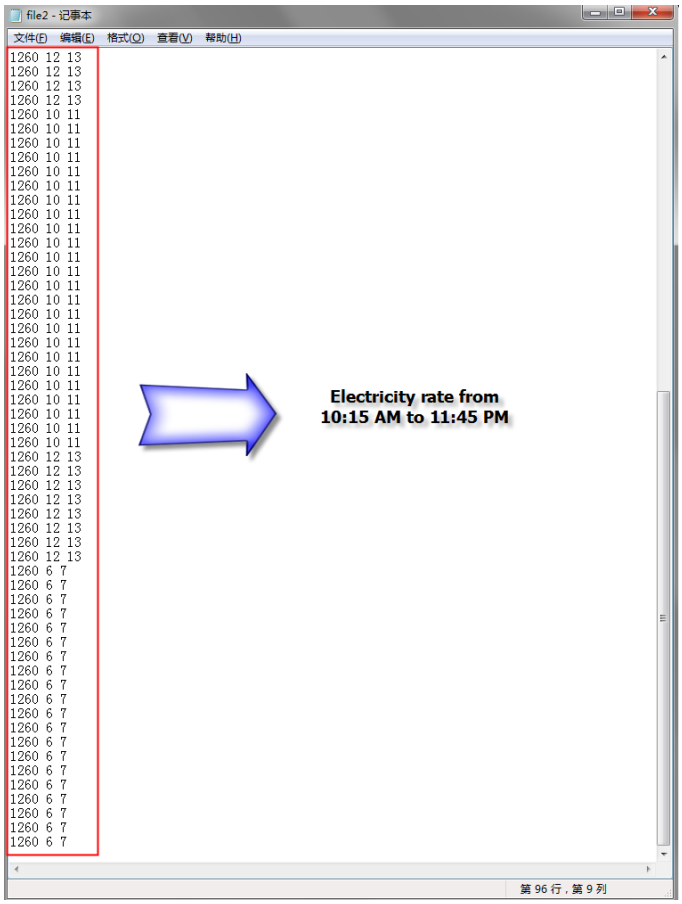


Figure 25b. Input price information

For clearer demonstration, it is better to add a time display module in Quartus II, which is shown below.

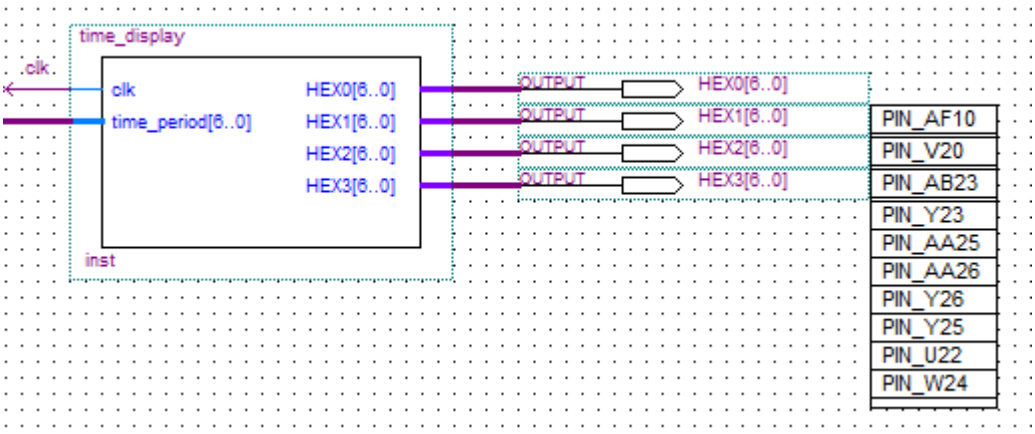


Figure 26. time_display module

And the Verilog code for this module is shown below.

```
module time_display (
                                    clk,
                                    time_period,
                                    HEX0,
                                    HEX1,
                                    HEX2,
                                    HEX3
                                );
input clk;
input [6:0]time_period;
output [6:0]HEX0,HEX1,HEX2,HEX3;
reg [3:0]t1,t2,t3,t4;
reg [4:0]tm,ts; //tm: min; ts: sec

always @ (posedge clk)
```

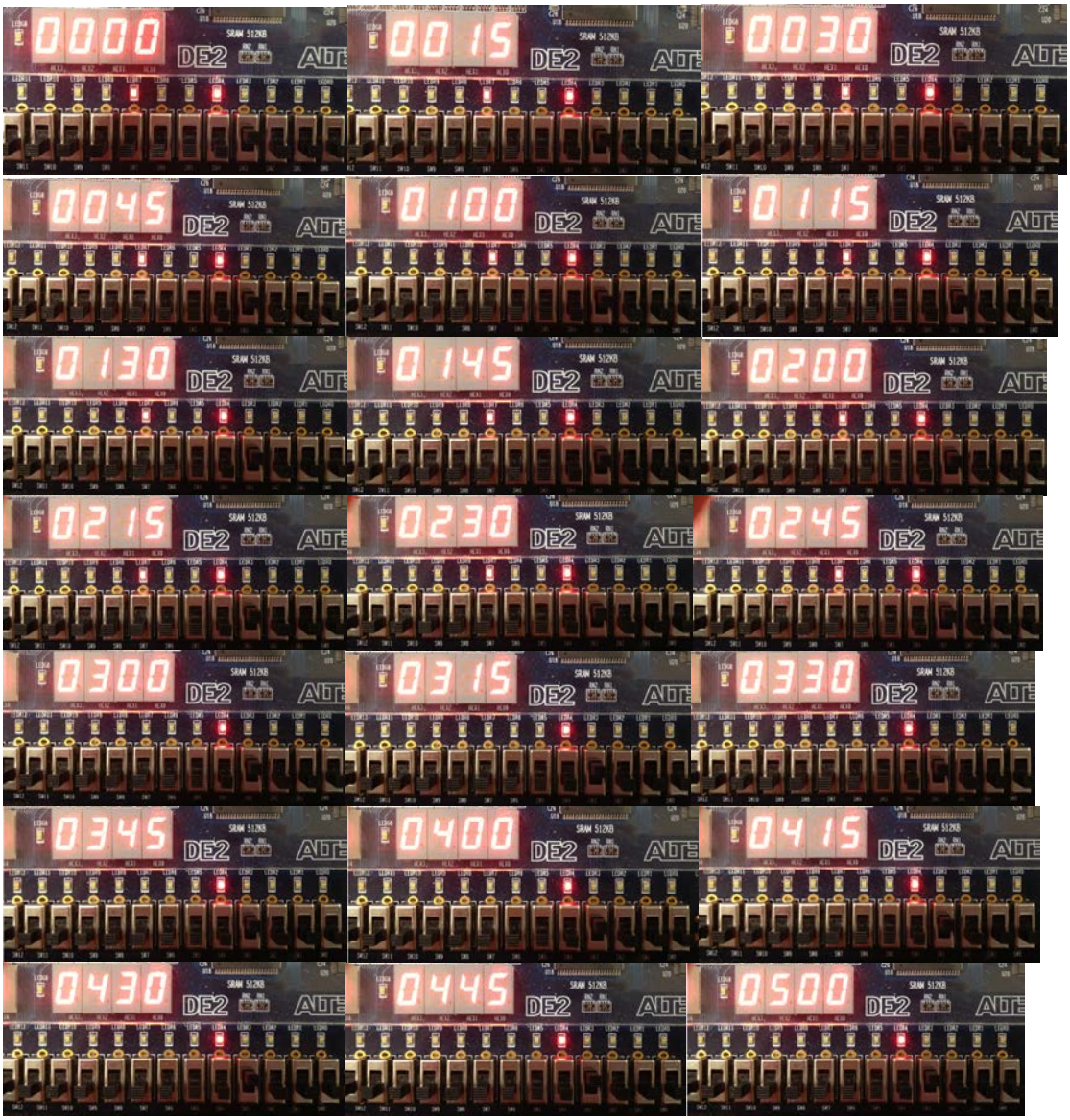
```
    begin
        tm<=time_period/4;
        ts<=time_period%4;
        t1<=tm/10;
        t2<=tm%10;
        case (ts)
        0: begin
            t3<=0;
            t4<=0;
          end
        1: begin
            t3<=1;
            t4<=5;
          end
        2: begin
            t3<=3;
            t4<=0;
          end
        3: begin
            t3<=4;
            t4<=5;
          end
        endcase
    end

HEX_display h1(t1,HEX3);
HEX_display h2(t2,HEX2);
HEX_display h3(t3,HEX1);
HEX_display h4(t4,HEX0);
```

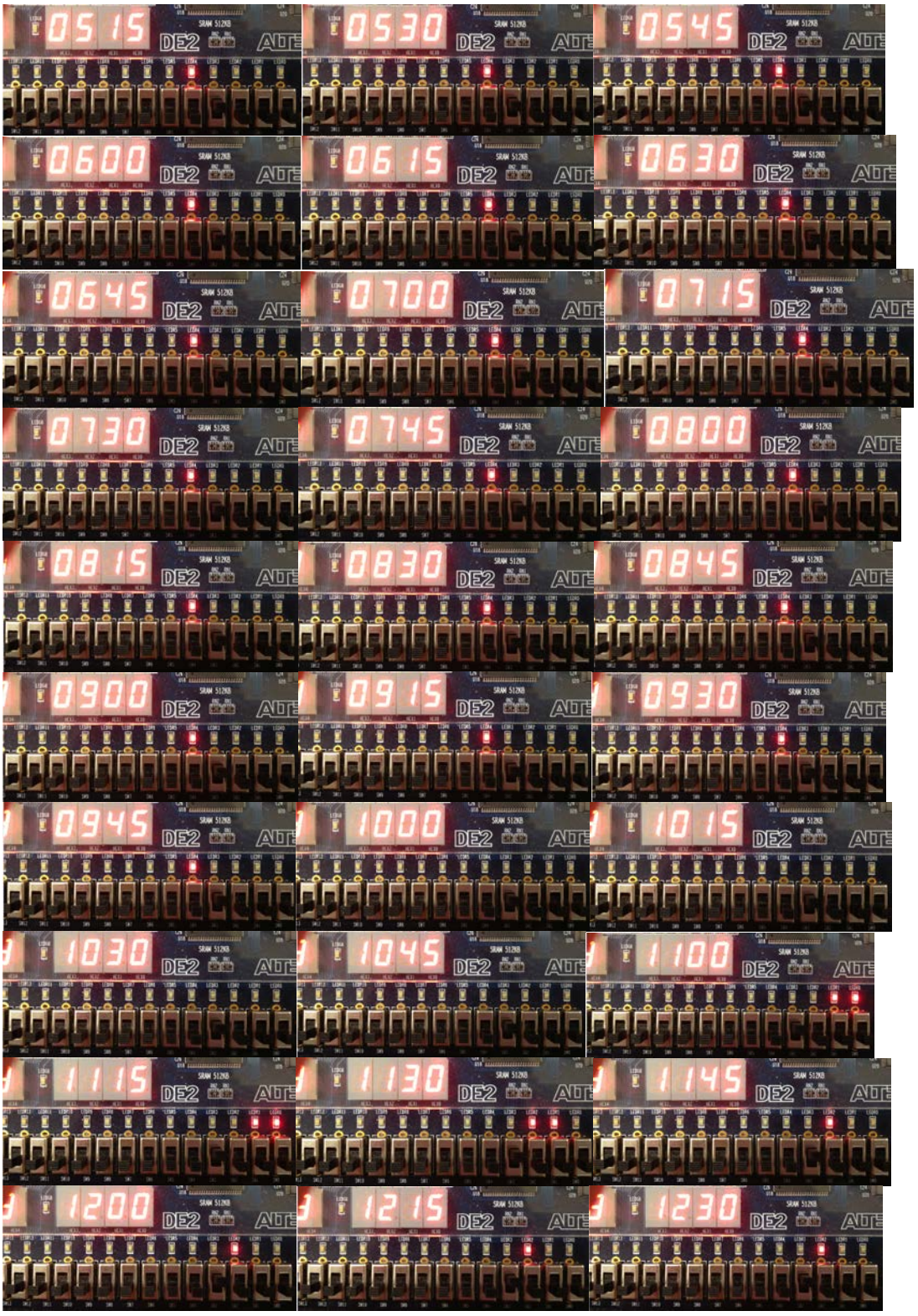
```
endmodule

module HEX_display(t,q);
input [3:0]t;
output reg [7:0]q;

always begin
        case (t)
        0:  q<=7'b1000000;
        1:  q<=7'b1111001;
        2:  q<=7'b0100100;
        3:  q<=7'b0110000;
        4:  q<=7'b0011001;
        5:  q<=7'b0010010;
        6:  q<=7'b0000010;
        7:  q<=7'b1111000;
        8:  q<=7'b0000000;
        9:  q<=7'b0010000;
        default q<=7'b1111111;
        endcase
    end
endmodule
```

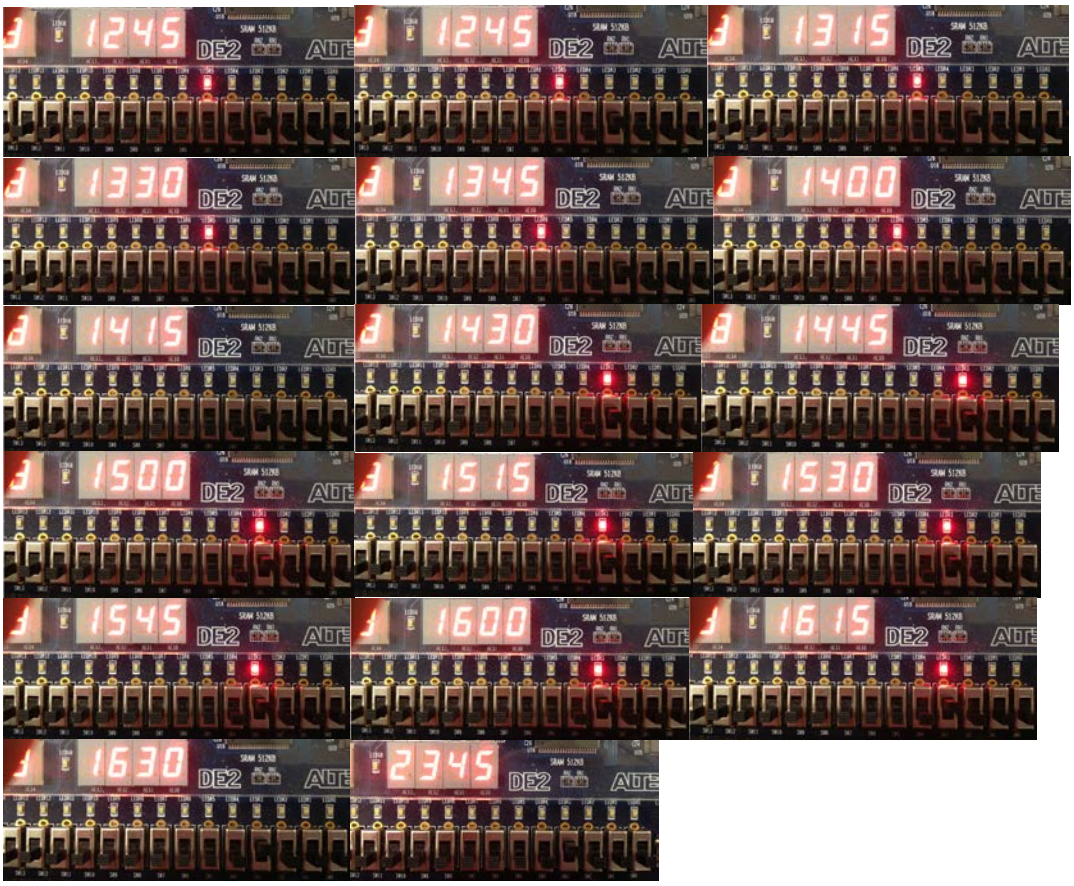
Since it is better to assign one task in more time periods to flat the power consumption curve, I will add another factor that record how many time periods this task is assigned in. If there are many solution with the same lowest cost for one task, the controller would choose the one with the most time periods. Therefore, the result for this simulation is shown below

Time period	Task and Power Level
0:00->3:00	**Refrigerator***(720Watt)*, **Water Pump***(250Watt)*
3:00->10:00	**Water Pump***(250Watt)*
11:00->11:30	**Mini Oven***(900Watt)*, **Rice Cooker***(400Watt)*
11:30->11:45	**Rice Cooker***(400Watt)*, **Clothes Washer***(400Watt)*
11:45->12:45	**Clothes Washer***(400Watt)*
12:45->13:45	**Vacuum Cleaner***(1000Watt)*
13:45->14:15	**Dishwasher***(1200Watt)*
14:30->16:30	**Clothes Dryer***(2650Watt)*
otherwise	No tasks assigned

Table 7. Scheduling




0515
0530
0545
0600
0615
0630
0645
0700
0715
0730
0745
0800
0815
0830
0845
0900
0915
0930
0945
1000
1015
1030
1045
1100
1115
1130
1145
1200
1215
1230
DE2



The total cost is $1.42, which is exactly the same value with the result obtained from the enumeration method. This is the optimal solution, which means that, in this case, any other scheduling cannot get lower price than $1.42. Customers can benefit from our scheduling.

4. Example 3

Above is the simulation for individual user, and I will present the simulation for multi-user below. Here is the input information.

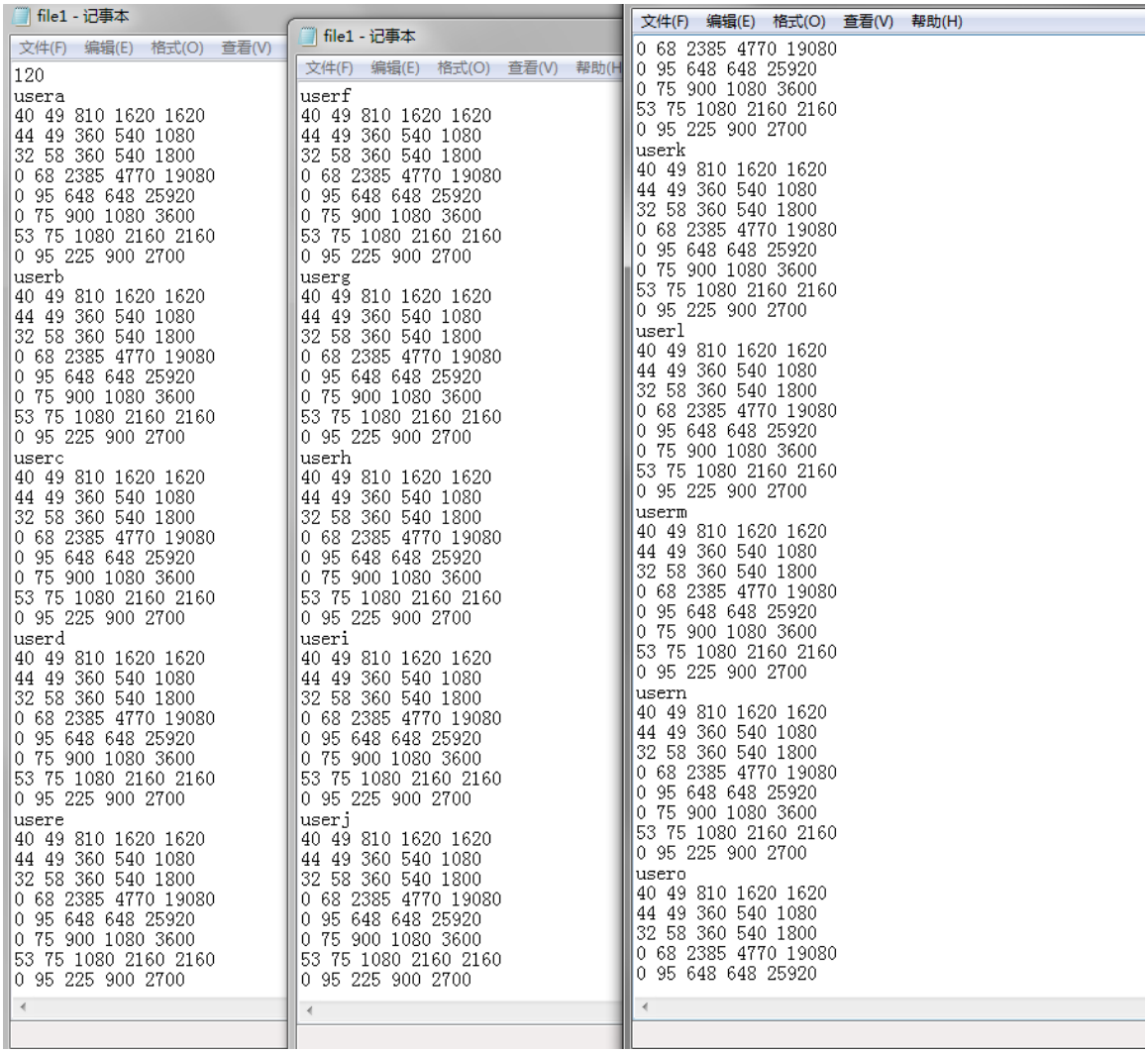


Figure 27. Multiple users' tasks

Then, the simulation result is shown below.

power	
power[0]	25182
power[1]	26082
power[2]	26982
power[3]	32652
power[4]	17442
power[5]	16542
power[6]	20412
power[7]	21312
power[8]	25182
power[9]	25182
power[10]	20412
power[11]	20412
power[12]	16542
power[13]	20412
power[14]	20412
power[15]	20412
power[16]	21312
power[17]	21312
power[18]	21312
power[19]	16542
power[20]	20412
power[21]	16542
power[22]	20412
power[23]	20412
power[24]	26082
power[25]	25182
power[26]	20412
power[27]	19512
power[28]	9072
power[29]	9072
power[30]	9072
power[31]	9072
power[32]	9072
power[33]	9072
power[34]	9072
power[35]	9072
power[36]	9072
power[37]	9072
power[38]	9072
power[39]	9072
power[40]	0
power[41]	0
power[42]	0
power[43]	0
power[44]	1170
power[45]	14670
power[46]	1260
power[47]	1170
power[48]	15570
power[49]	17100
power[50]	6660
power[51]	7020
power[52]	1080

Outline | Make Targets | Variables | Disassembly

Name	Value
power[53]	1080
power[54]	1080
power[55]	30600
power[56]	1008
power[57]	1188
power[58]	1188
power[59]	648
power[60]	648
power[61]	648
power[62]	648
power[63]	648
power[64]	648
power[65]	648
power[66]	648
power[67]	648
power[68]	648
power[69]	648
power[70]	648
power[71]	648
power[72]	648
power[73]	648
power[74]	648
power[75]	648
power[76]	10548
power[77]	10548
power[78]	10548
power[79]	648

Outline | Make Targets | Variables | Disassembly

Name	Value
power[80]	648
power[81]	648
power[82]	648
power[83]	648
power[84]	648
power[85]	648
power[86]	648
power[87]	648
power[88]	648
power[89]	648
power[90]	648
power[91]	648
power[92]	648
power[93]	648
power[94]	648
power[95]	648
lowest	32767000
taskprice	0
j	95
m	8
i	96
n	4
ii	120
cost_total	7168536
tasknumber	120
sched	

Figure 28. Simulation result

If we don’t use any optimized way to assign these tasks, the power consumption curve would become very sharp in the high-demand time periods. After using our proposed algorithm, the curve becomes flatter. The total cost in this case is 7168536/3600=1991.26cents.

The scheduling time is about 10 minutes in this case for 120 tasks. This is for the first scheduling at the beginning of the day. During the day, if the consumer want to change several tasks or add some new tasks to the loads, the controller will only do the scheduling for these tasks, not all the tasks for the whole community. The scheduling time for several tasks is less than 1 minutes. The speed is acceptable.

Ⅷ. Problem Solution

1. If the onchip memory is insufficient, use SDRAM to substitute it.

2. When build project in NIOS II.

```
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Pausing target processor: OK
Initializing CPU cache (if present)
OK

Downloading 00800000 ( 0%)
Downloading 00810000 (83%)
Downloaded 77KB in 1.3s (59.2KB/s)

Verifying 00800000 ( 0%)
Verify failed between address 0x800000 and 0x80FFFF
Leaving target processor paused
```

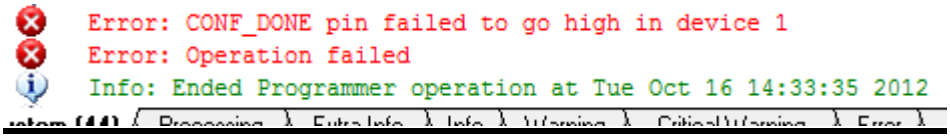
   First of all, check the address in SOPC builder to see which component has error, most of the time the errors occur in memory component.

   Then check the design in Quartus especially the pin assignment to see if all the pins are correctly connected. Data bus must be bidirectional, and it is very easy to forget it.

   If it is the sdram, pll module is needed and the phase shift in pll module should be corrected.

3. The data would be lost in sdram after shut down the FPGA. Altera's DE0 Nano board has NO flash memory.

4.



Switch the button  from **PROG** to **RUN**

5. Because of the compatibility issues, the following problem would occur very often in Win7 system.

   *[main] ? (5680) D:\altera\80\quartus\bin\cygwin\bin\sh.exe: *** fatal error - couldn't allocate heap, Win32 error 487, base 0x870000, top 0x890000, reserve_size 126976, allocsize 131072, page_const 4096*
   2 [main] sh 3244 fork: child -1 - died waiting for longjmp before initialization, retry 0, exit code 0x100, errno 11

   It is not enough to change the compatibility in properties. The reason to cause it is that the capacity of *Cygwin*'s heap is not big enough and Windows does not add its capacity automatically. We need to open the Registry, under HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER section, add a DWORD key named *heap_chunk_in_mb* under the *Cygwin* folder. Change the value of it to 1024 in decimal to limit the capacity of heap to 1024 Mb. Even though the problem would still occur sometimes, the probability becomes much lower.

Ⅸ. Conclusion

Using our proposed dynamic programming based algorithm could significantly reduce the time complexity to schedule tasks for multi-users, compared with the classical algorithm like the method of enumeration. On the other hand, I am satisfied with the accuracy of finding a solution set that is close to the global optimal solution.

Altera's DE2 FPGA board we used fits our task. Furthermore, Smart Home system consists of a lot of applications to provide improved comfort, convenience and efficiency. Due to FPGA's large amount of I/O interfaces and low complexity, it is very easy for us to do the further development of smart behavior on FPGA.

Ⅹ. Coding in NIOS II

```
/*
 * Project: DESIGN AND IMPLEMENT DYNAMIC PROGRAMMING BASED DISCRETE
POWER LEVEL SMART HOME SCHEDULING
 *
 * Name   : XIN YANG
 *
 * Advisor: SHIYAN HU
 *
 * This project consists of 3 parts:
 * 1: Read user information from the computer and save it into flash
memory in FPGA
 * 2: Schedule the tasks use our proposed dynamic programming based
algorithm
 * 3: Demonstrate the results using the onboard LEDs
 *
 */

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include "system.h"
#include "io.h"
#include "altera_avalon_pio_regs.h"

#define hours 96
#define task_max_number 200
#define repeat_times 10
#define value_size 6      //maxium size of each input value
#define BUF_SIZE (30*task_max_number)



// [0] start time [1] end time [2] power level 1 [3] power level 2 [4]
total time
int task[task_max_number][5];

// [0] Accumulated power threshold [1] unit power level price <=
threshold [2] unit power level price > threshold
int price[hours][3];

// [0] power consumed [1] cost [2] power level [3] path
int sched[hours][task_max_number][4];
int best[hours][task_max_number];
int power[hours] = {0};


int lowest = 9999999;
volatile int taskprice = 0;
int delay;
int condition;
```

```
volatile int j=0;
volatile int m=0;
volatile int i=0;
volatile int n=0;
volatile int ii=0;
volatile int cost_total=0;

volatile int tasknumber;



/*****************************************************************
*  Function: taskread
*
*  Purpose: Read the tasks information from the file input from
*           the consumers
*
*****************************************************************/
void taskread(FILE* fp)
{
  char buffer[BUF_SIZE] = {0x20};
  char value[value_size];
  int read_size;
  int i,j,p;
  int ptr;

  read_size = fread (buffer, 1, BUF_SIZE, fp); //the total size of the
file

  for(p=0; p<value_size; p++) value[p] = 0;

  for(i=0; i<value_size; i++)
  {
    if ((buffer[i] == 0xa) | (buffer[i] == 0x20) | (buffer[i] == 0xd))
break;     // if it is a newline or space or Enter, then break
    else value[i] = buffer [i];
  }

  ptr = i+2;
  tasknumber = atoi(value);

  for(i=0; i<tasknumber; i++)
  {
    for(j=0; j<5; j++)
    {
        for(p=0; p<value_size; p++) value[p] = 0;
        for(p=ptr; p<ptr+value_size; p++)
        {
            if ((buffer[p] == 0xa) | (buffer[p] == 0x20) | (buffer[p]
== 0xd)) break;  // if it is a newline or space or Enter, then break
            else value[p-ptr] = buffer[p];
        }
        ptr = p+1;
        task[i][j] = atoi(value);
        if (buffer[p] == 0xd) {ptr++; break;}
    }
```

```
  }

}

/*****************************************************************
*  Function: hoursread
*
*  Purpose: Read the hourly price information from the file input from
*           the utility company
*
*****************************************************************/
void hoursread(FILE* fp)
{
  char buffer[BUF_SIZE] = {0x20};
  char value[value_size];
  int read_size;
  int i,j,p;
  int ptr=0;

  read_size = fread (buffer, 1, BUF_SIZE, fp); //the total size of the
file

  for(p=0; p<value_size; p++) value[p] = 0;

  for(i=0; i<hours; i++)
  {
    for(j=0; j<3; j++)
    {
        for(p=0; p<value_size; p++) value[p] = 0;
        for(p=ptr; p<ptr+value_size; p++)
        {
            if ((buffer[p] == 0xa) | (buffer[p] == 0x20) | (buffer[p]
== 0xd)) break;  // if it is a newline or space or Enter, then break
            else value[p-ptr] = buffer[p];
        }
        ptr = p+1;
        price[i][j] = atoi(value);
        if (buffer[p] == 0xd) {ptr++; break;}
    }
  }

}

/*****************************************************************
*  Function: Initialization
*
*  Purpose: Open the users' task information file and utility
*           company's power rates information file in the
*           flash memory
*
*****************************************************************/
void Initialization()
{
       FILE *fp;
```

```
  fp    =   fopen   ("/mnt/rozipfs/file1.txt", "r");
    if (fp ==   NULL)
    {
        printf ("Cannot open file.\n");
        exit (1);
    }

  taskread(fp);

    fclose (fp);

    fp    =   fopen   ("/mnt/rozipfs/file2.txt", "r");
    if (fp ==   NULL)
    {
        printf ("Cannot open file.\n");
        exit (1);
    }

  hoursread(fp);

    fclose (fp);
}



/*****************************************************************
*  Function: cost
*
*  Purpose: Return the cost. If the accumulated power level is larger
*           or equal to the threshold value, use the 1st power rate.
*           Otherwise, use the 2nd power rate.
*
*****************************************************************/
int cost(int hour_now, int power_consume)
{
   int sum;
   if (power_consume<=price[hour_now][0])
      sum = price[hour_now][1] * power_consume;
   else
      sum = price[hour_now][2] * power_consume;
   return sum;
}



/*****************************************************************
*  Function: Schedule the tasks
*
*  Purpose: Use our proposed algorithm to schedule the tasks
*
*
*****************************************************************/
void go(int task_now)
{
   int good;
   int cost_cach;
```

```
   int lowest=9999999;
   int count;
   int lowest_time;  // it means the last time we found the lowest cost

   int Fst_hour=task[task_now][0];   // First hour

   lowest_time = 0;

   // for repeatly schedualing
   for (j=Fst_hour; j<=task[task_now][1]; j++)
        {
            cost_total=cost_total-cost(j,power[j])+cost(j,power[j]-
best[j][task_now]);
            power[j]=power[j]-best[j][task_now];
            best[j][task_now]=0;
        }

    //sched[j][0][0] means the number of situation in hour j
   sched[Fst_hour][0][0]=3;
   sched[Fst_hour][1][1]=cost_total;
   sched[Fst_hour][2][0]=task[task_now][2];

sched[Fst_hour][2][1]=cost_total+cost(Fst_hour,power[Fst_hour]+task[tas
k_now][2])-cost(Fst_hour,power[Fst_hour]);
   sched[Fst_hour][2][2]=task[task_now][2];
   sched[Fst_hour][3][0]=task[task_now][3];

sched[Fst_hour][3][1]=cost_total+cost(Fst_hour,power[Fst_hour]+task[tas
k_now][3])-cost(Fst_hour,power[Fst_hour]);
   sched[Fst_hour][3][2]=task[task_now][3];

  // j means hours
   for (j=Fst_hour; j<task[task_now][1]; j++)
   {
    sched[j+1][0][0]=1;
    sched[j+1][1][3]=1;
    sched[j+1][1][1]=sched[j][1][1];

    for (m=1; m<=sched[j][0][0]; m++)
    if ((m==1) | (sched[j][m][0]!=0))
      if (sched[j][m][0]<task[task_now][4])
         //n means solutions
         for (n=2; n<=3; n++)
         if (sched[j][m][0]+task[task_now][n]<=task[task_now][4])
         {
            i=1;
            count=0;
            good=0;

cost_cach=sched[j][m][1]+cost(j+1,power[j+1]+task[task_now][n])-
cost(j+1,power[j+1]);
            while (i<=sched[j+1][0][0])
            {
              if (((sched[j][m][0]+task[task_now][n]>sched[j+1][i][0])
& (cost_cach<=sched[j+1][i][1])) |
((sched[j][m][0]+task[task_now][n]>=sched[j+1][i][0]) &
```

```
(cost_cach<sched[j+1][i][1])))
                {
                if (good==0)
                {
                 sched[j+1][i][0]=sched[j][m][0]+task[task_now][n];
                 sched[j+1][i][1]=cost_cach;
                 sched[j+1][i][2]=task[task_now][n];
                 sched[j+1][i][3]=m;
                 good=i;
                }
                else {sched[j+1][i][0]=0; sched[j+1][i][1]=0;}
                }
               if ((sched[j][m][0]+task[task_now][n]>sched[j+1][i][0]) &
(cost_cach>sched[j+1][i][1]))
                  count=count+1;
              i=i+1;
            }
            if (count>=sched[j+1][0][0])
            {
                sched[j+1][0][0]=i;
                sched[j+1][i][0]=sched[j][m][0]+task[task_now][n];
                sched[j+1][i][1]=cost_cach;
                sched[j+1][i][2]=task[task_now][n];
                sched[j+1][i][3]=m;
                good=count+1;   // good is the new solution added in
the schedule
            }

            if ((sched[j][m][0]+task[task_now][n]==task[task_now][4]) &
((cost_cach<lowest) | ((cost_cach==lowest) &
(power[lowest_time]>power[j+1]))))
                {
                  lowest=cost_cach;
                  lowest_time=j+1;
                  cost_total=cost_cach;
                  for (i=0; i<hours; i++)
                    best[i][task_now]=0;
                  best[j+1][task_now]=task[task_now][n];
                  for (i=j+1; i>0; i--)
                    {
                        best[i-1][task_now]=sched[i-
1][sched[i][good][3]][2];
                        good=sched[i][good][3];
                    }
                }
        }
   }
   for (i=0; i<hours; i++)
        power[i]=power[i] + best[i][task_now];
}


/*****************************************************************
*  Function: LEDs
*
*  Purpose: Give the outputs of the NIOS II core specific values
```

```
*           to toggle the LEDs
*
*
*************************************************************/
void led(int hour)
{
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED0_BASE, best[hour][0]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED1_BASE, best[hour][1]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED2_BASE, best[hour][2]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED3_BASE, best[hour][3]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED4_BASE, best[hour][4]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED5_BASE, best[hour][5]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED6_BASE, best[hour][6]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED7_BASE, best[hour][7]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED8_BASE, best[hour][8]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED9_BASE, best[hour][9]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED10_BASE,
best[hour][10]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED11_BASE,
best[hour][11]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED12_BASE,
best[hour][12]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED13_BASE,
best[hour][13]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED14_BASE,
best[hour][14]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED15_BASE,
best[hour][15]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED16_BASE,
best[hour][16]);
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED17_BASE,
best[hour][17]);

}

/*****************************************************************
*  Function: main
*
*  Purpose : Just do it
*
*****************************************************************/

int main()
{
    Initialization();

    int repeat;
    for (repeat=repeat_times; repeat>0; repeat--)
        for (ii=0; ii<tasknumber; ii++)
            go(ii);

    while(1)
    {
        int tmp;
        tmp = IORD_ALTERA_AVALON_PIO_DATA(PIO_KEY_BASE);
        switch (tmp)
```

```
        {
            case 0x00: led(0);break;
            case 0x01: led(1);break;
            case 0x02: led(2);break;
            case 0x03: led(3);break;
            case 0x04: led(4);break;
            case 0x05: led(5);break;
            case 0x06: led(6);break;
            case 0x07: led(7);break;
            case 0x08: led(8);break;
            case 0x09: led(9);break;
            case 0xa: led(10);break;
            case 0xb: led(11);break;
            case 0xc: led(12);break;
            case 0xd: led(13);break;
            case 0xe: led(14);break;
            case 0xf: led(15);break;
            case 0x10: led(16);break;
            case 0x11: led(17);break;
            case 0x12: led(18);break;
            case 0x13: led(19);break;
            case 0x14: led(20);break;
            case 0x15: led(21);break;
            case 0x16: led(22);break;
            case 0x17: led(23);break;
        }
    }

    return 0;









}
```

XI. Appendix

In this appendix, I would like to describe the operating instructions about using Quatus II and NIOS II.

I start with design entry using schematics since all the modules are visible and the project becomes more intuitively clear than using Verilog Code as my top level entity.

First of all, create a project in Quatus II and assign the device. In the Quartus II software, select **File** -> **New Project Wizard**. Create a working directory for the project and then type a name for the top-level design entity. Assign a specific FPGA device in **Family & Device Settings**. In this project, since DE2 board is choosing, we need to change the Device family to **Cyclone II** and select **EP2C35F672C6** as our device. Then **Finish**.

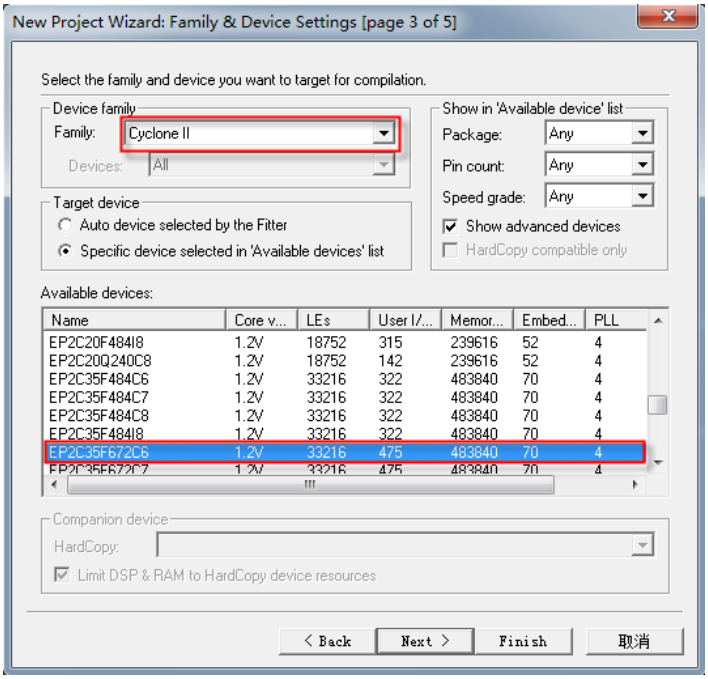


Figure 29. Specify the device

Create a top-level schematic design file Scheduling.bdf by selecting **File** -> **New**, and choose **Block Diagram/Schematic File**, then **OK**.

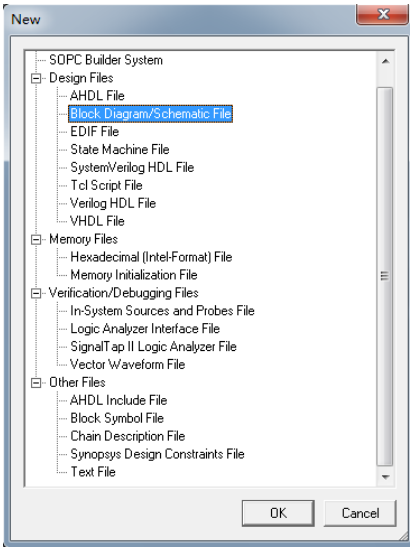


Figure 30.New BDF

Select **Tools** ->**SOPC Builder** to configure the NIOS II on chip system.

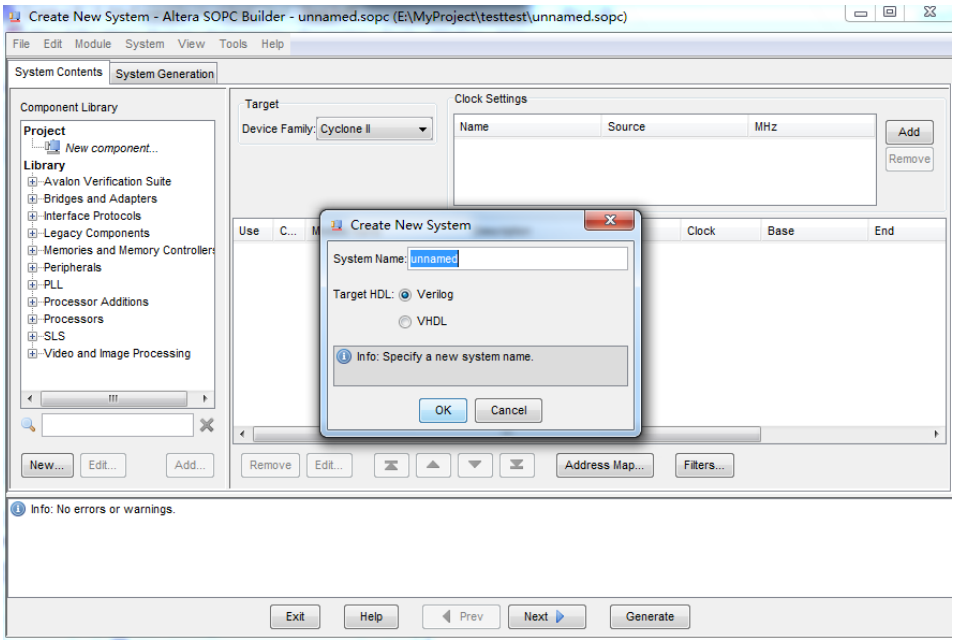


Figure 31.SOPC System

Select **Library** -> **Processors** -> **NIOS II Processor** to open the wizard as shown below to configure our CPU component. Click **Finish**

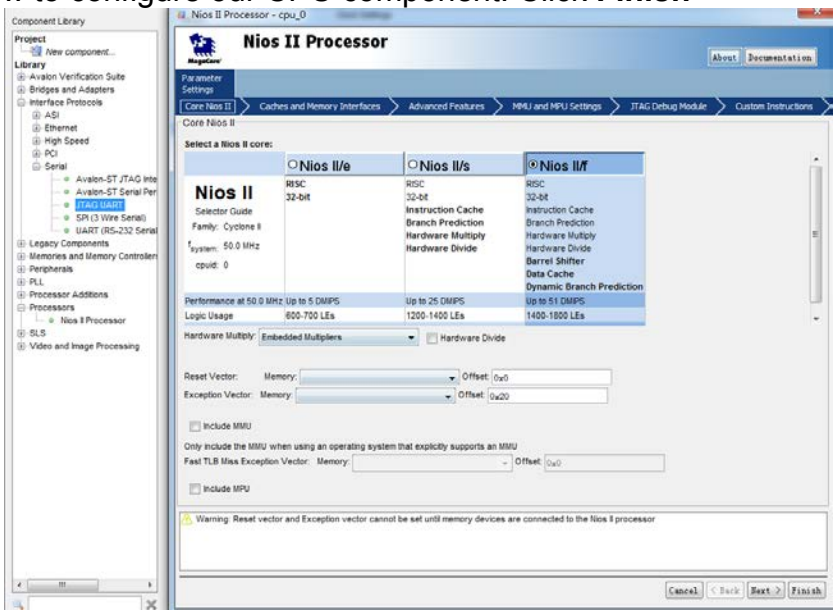


Figure 32.NIOS II Processor

The NIOS II/f we chose is with the best performance compared to the NIOS II/e and NIOS II/s. Even though we do not need this high performance in our project, I decide to use SDRAM, which has 8MB, as our memory and we do not really care about the extra memory space by choosing the high performance. In some other cases, when we use on chip memory, which has very small memory space like several KB, high performance processors may not be able to be used since there may not be enough memory space to support it.

Choose **Library** -> **Interface Protocols** -> **Serial** -> **JTAG UART** to open wizard and add JTAG UART by clicking **Finish**. See the figure below.

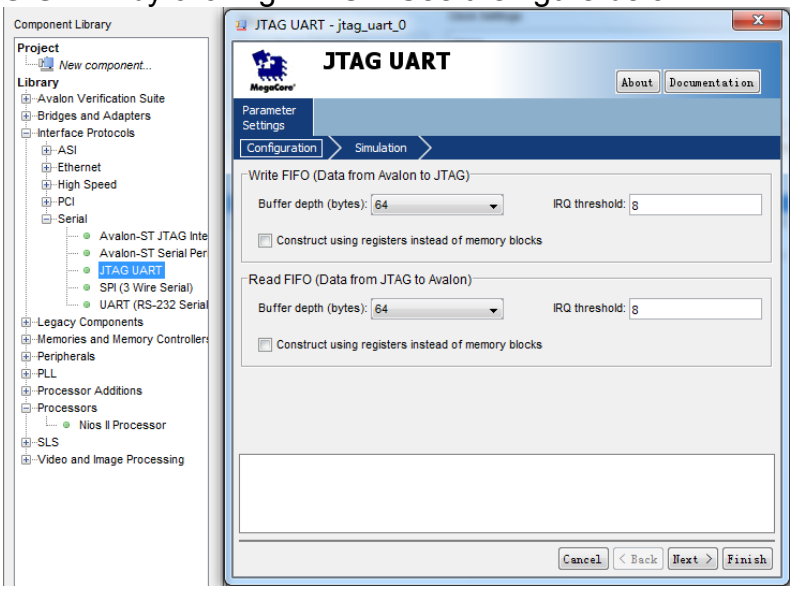


Figure 33. JTAG UART

Choose **Library** -> **Memories and Memory Controllers** -> **SDRAM** -> **SDRAM Controller** to open the wizard as shown below and add SDRAM to our system.

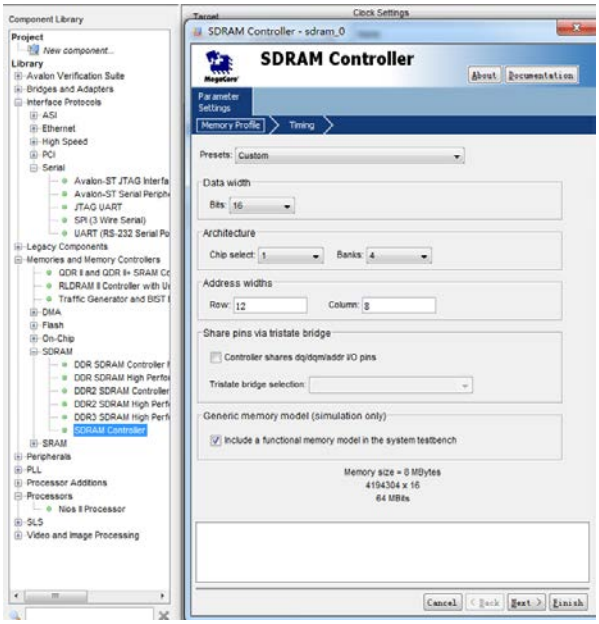


Figure 34. SDRAM Controller

The configuration of this SDRAM Controller including Timing Setting is described in the report at page 22 in details.

Click module *CPU* in the component list we have already built, update Reset Vector and Exception Vector as shown in figure below. Then click Finish.

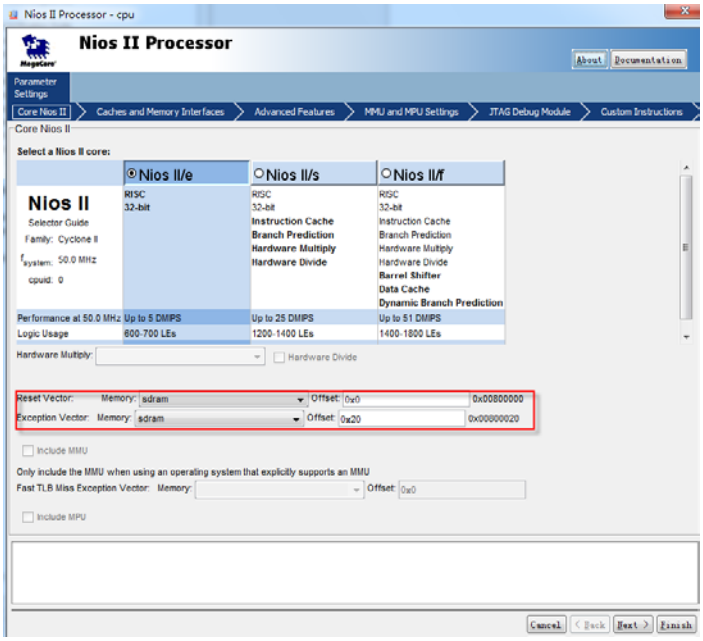


Figure 25. Nios II Processor Memory configuration

Choose **Library** -> **Memories and Memory Controllers** -> **Flash** -> **Flash Memory Interface (CFI)** to open the wizard to add the Flash Memory Interface as shown below.

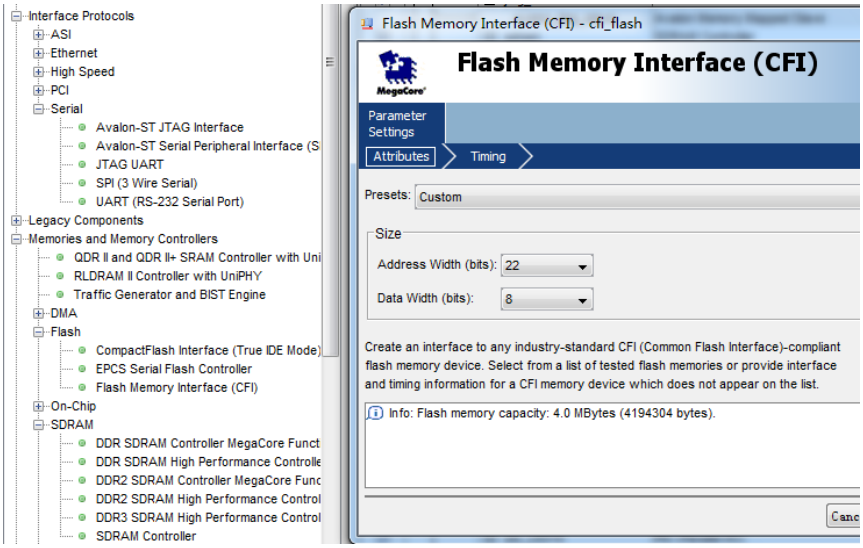


Figure 26. Flash Memory Interface (CFI)

The configuration of this Flash Memory Interface (CFI) including Timing Setting is also described in the report at page 22 in details.

Look back to Figure 4, the example of a NIOS II Processor System, Tristate Bridge is needed for the system to connect off-chip memory, like Flash Memory. Choose **Library** -> **Bridges and Adapters** -> **Memory Mapped** -> **Avalon-MM Tristate Bridge** to open the wizard as shown below to add the Tristate Bridge.

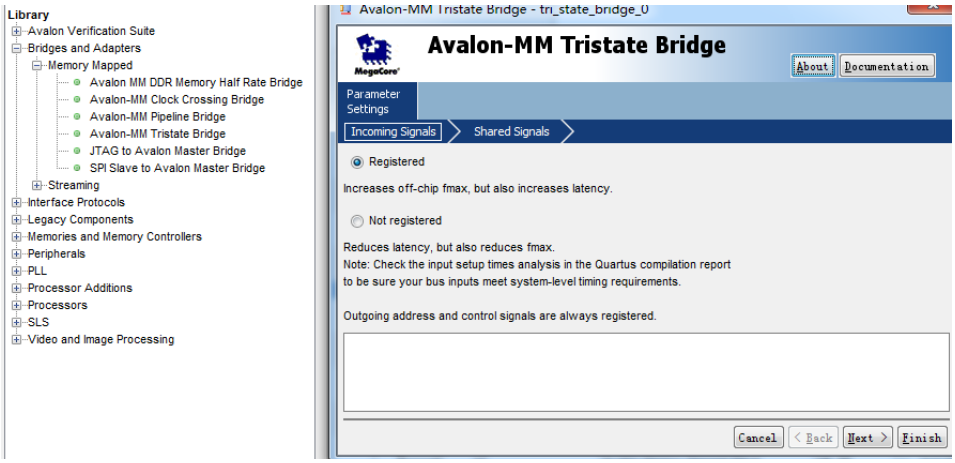


Figure 37. Tristate Bridge

Remember to connect the Tristate Bridge from the Tristate Master side to the Tristate Slave side of Flash Memory. See figure below.

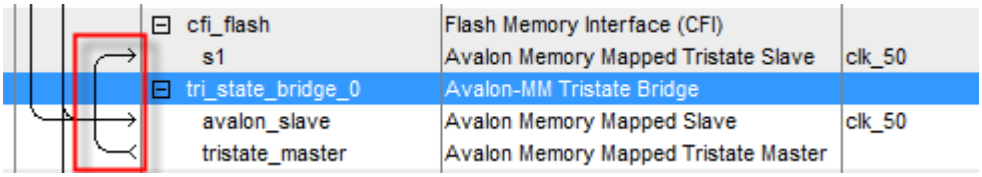


Figure 38. Connection between Tristate Master and Tristate Slave

Choose **Library** -> **Peripherals** -> **Microcontroller Peripherals** -> **PIO (Parallel I/O)** to open the wizard to add the Parallel inputs and outputs as shown below.

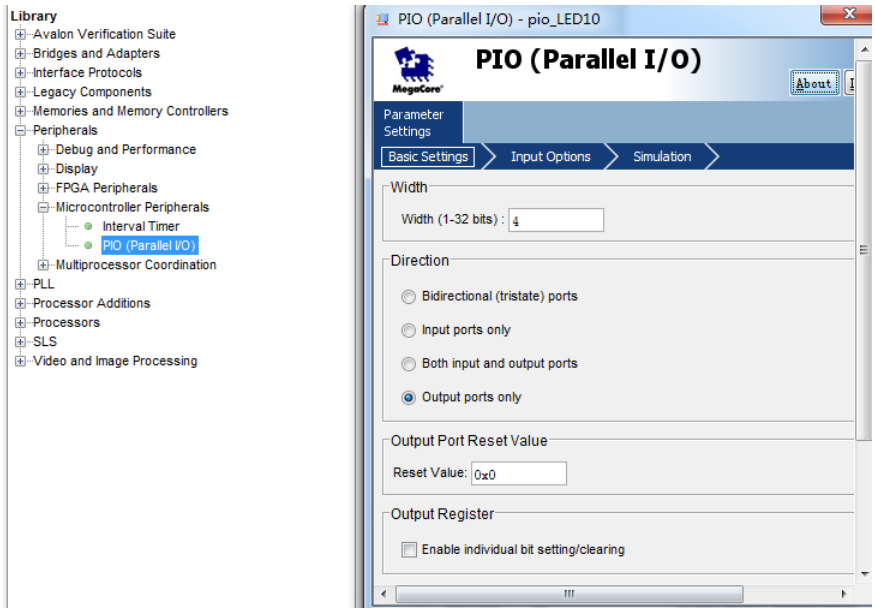


Figure 39. Parallel I/O

Since we need to use 18 LEDs to demonstrate the arrangement of tasks and a KEY input to control the hours, 18 4-bits wide output and 1 5-bits wide input are created in SOPC as shown below.

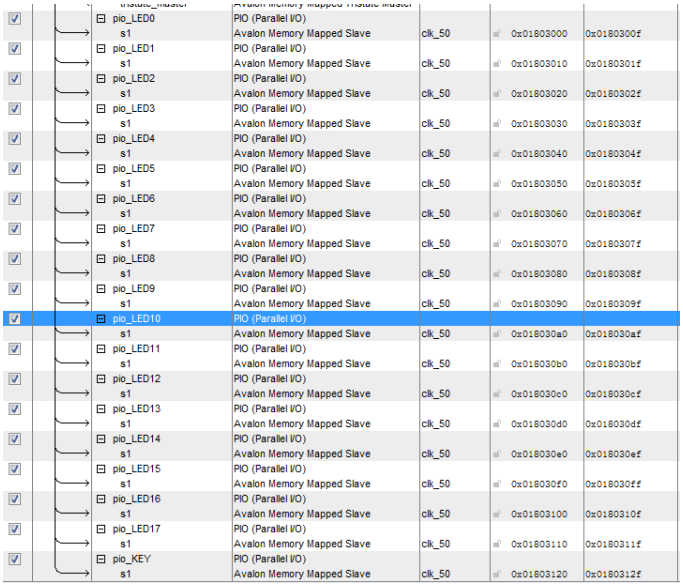


Figure 40. Parallel I/O setup

Choose **System** -> **Auto-Assign Base Addresses**. After these base addresses are assigned, choose **File** -> **Refresh System**. No errors would be displayed in the message window as shown in the figure below

Figure 41. No errors

Generate the system and **exit** the SOPC builder after the system is successfully generated.

In Quatus II, choose **Edit** -> **Insert Symbol…** and you will find the system module we just created in **Libraries**.

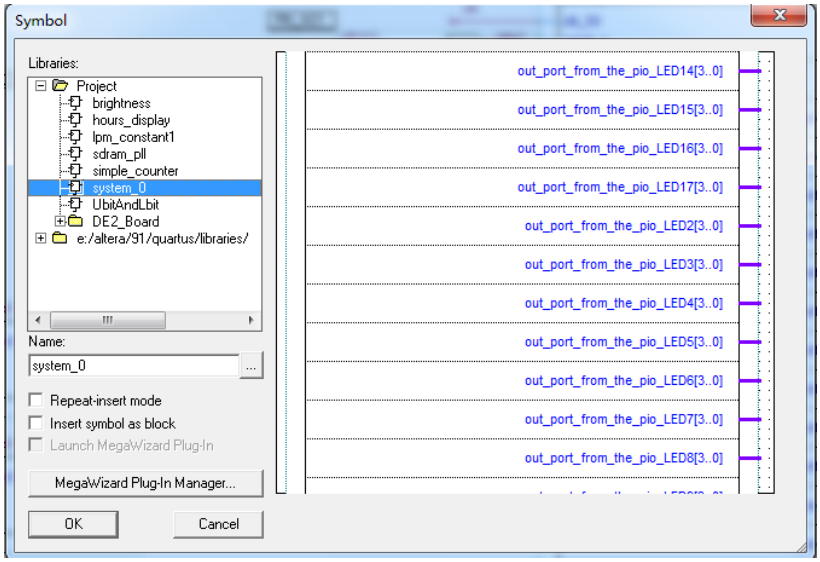


Figure 42. Insert Symbol

Click **OK** and put the module at anywhere in the schematic file we created.

Next we need to use Quartus to add a Phase Lock Loop (PLL) Megafunction.

Choose Edit -> Insert Symbol.

Click Megawizard Plug-In Manager and following window would appear.

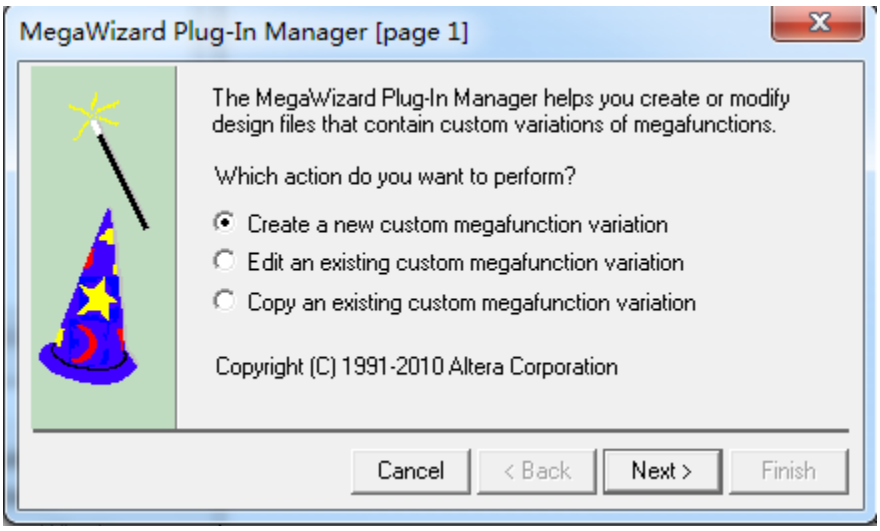


Figure 43. Megawizard Plug-In Manager Page1

Click **Next**.

Choose **I/O** -> **ALTPLL**. Under "**Which device family will you be using**", choose **Cyclone II** since we are using DE2 FPGA development, which is embedded with Altera's Cyclone II device. Choose **Verilog HDL** and give this PLL module a new name like **sdram_pll**. Then click **Next**.

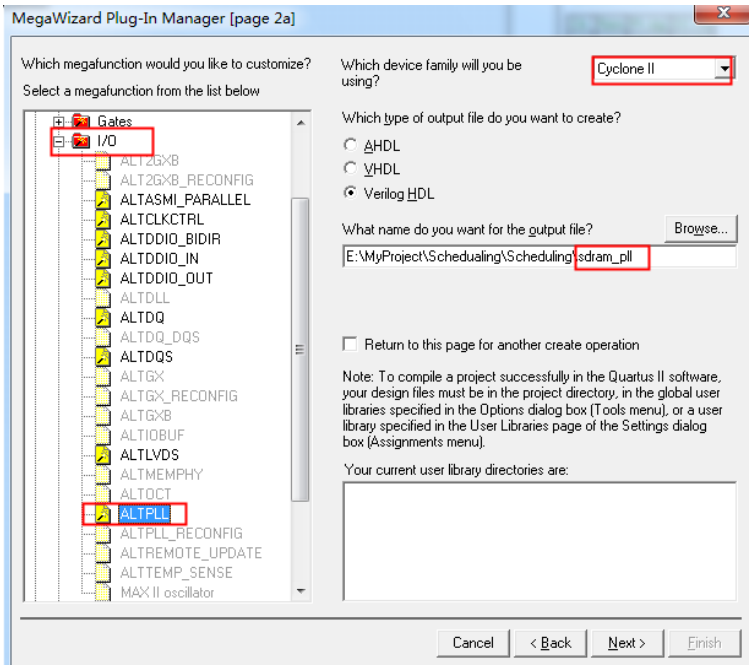


Figure 44. Megawizard Plug-In Manager Page2

Configure the ALTPLL followed by the figures shown below.

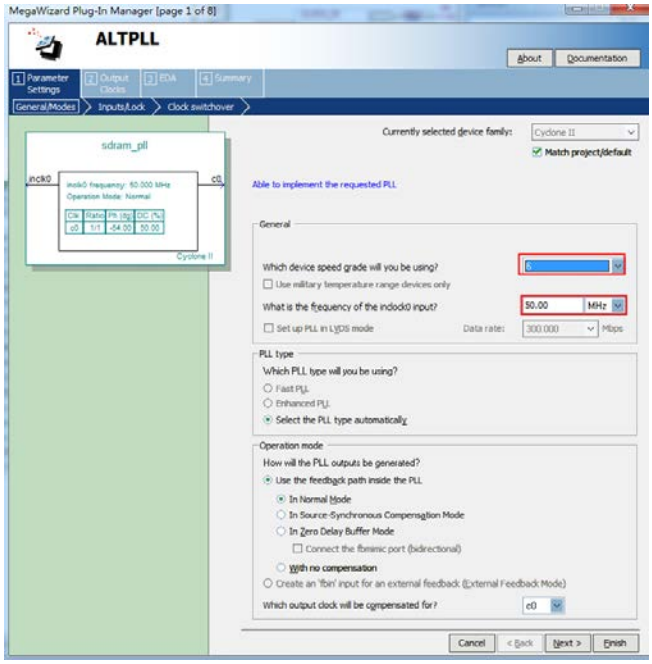


Figure 45. Step 1 PLL configuration

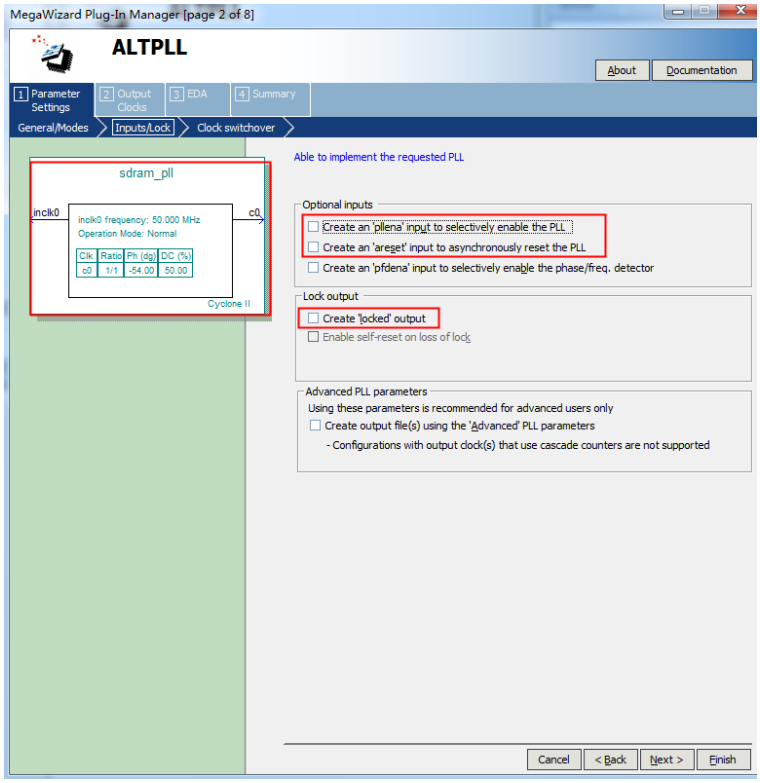


Figure 46. Step 2 PLL configuration

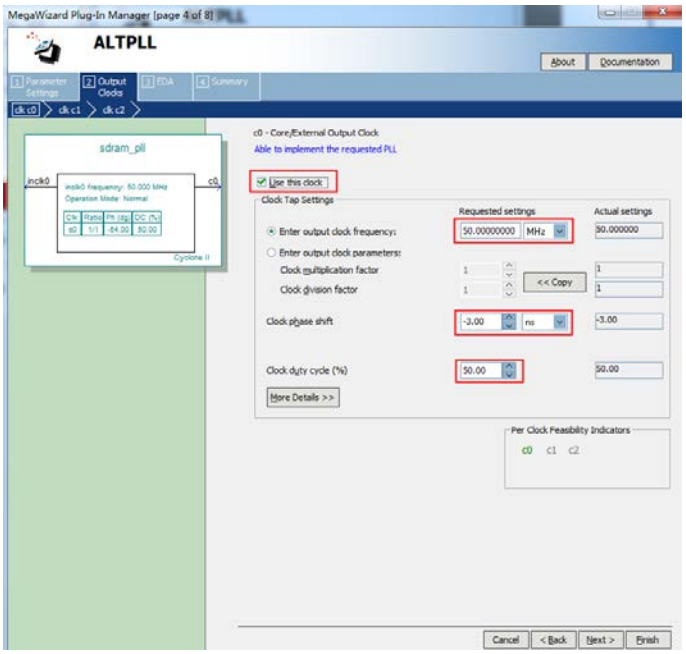


Figure 47. Step 3 PLL configuration

Then **Finish**.

Then insert this module in the schematic like we did for adding the system module.

Create other components modules and name the connecting wires between them, and then we could obtain the schematic in Figure 5. After assigning the pin assignment, compiling the project and downloading the sof file to the board, we finish the hardware design part. We could close the Quartus II Programmer or leave it open in the background.

Open **Nios II IDE**.

Choose **File** -> **Switch Workspace** to set the workspace to your project folder.

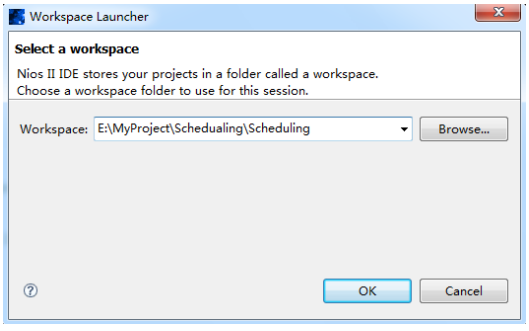


Figure 48. Switch Workspace

Choose File -> New -> NIOS II C/C++ Application to open the New Project Wizard.

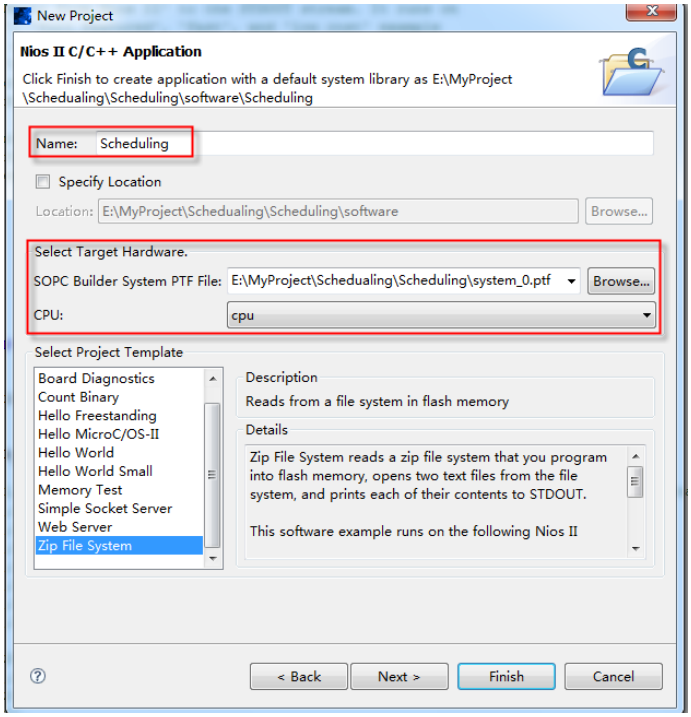


Figure 49. New Project

Then fill the code in the part “Coding in NIOS II”.

Choose **Tools** -> **Flash Programmer**. Under Flash Programmer, create a new flash programmer and configure it. See the figures below.

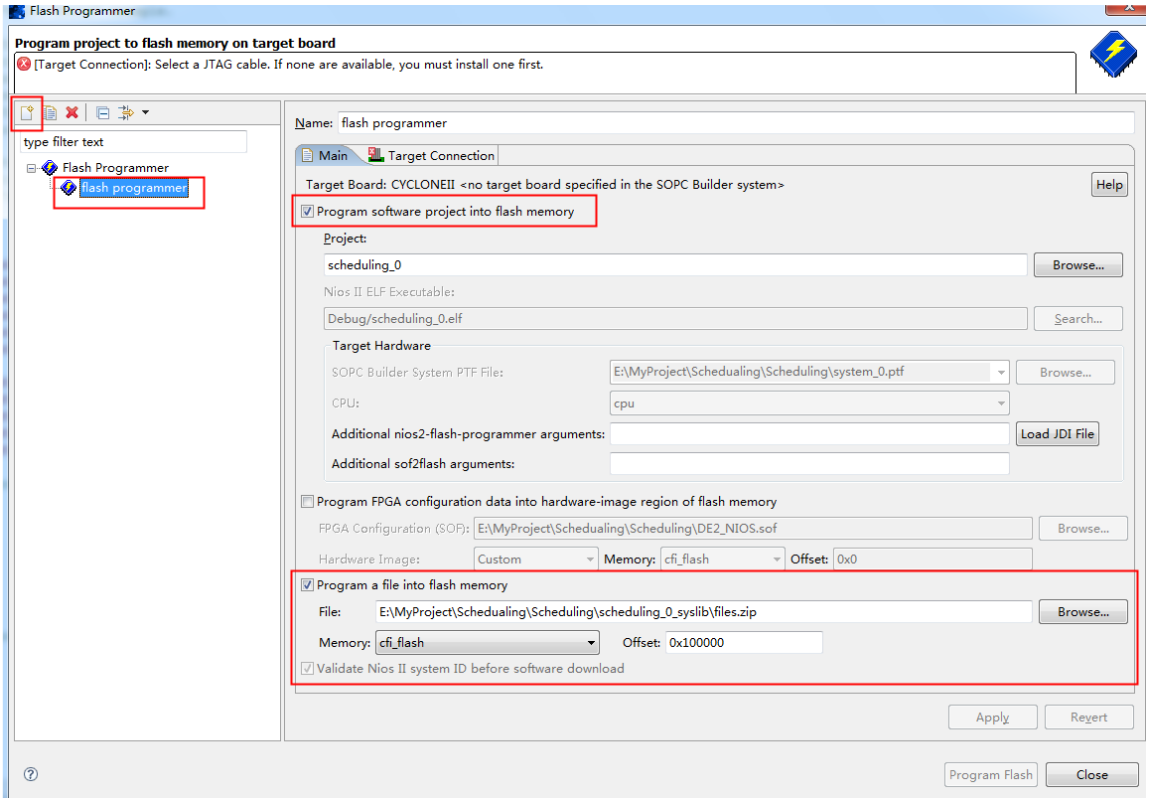


Figure 50. Flash Programmer

Offset must be 0x100000 in this case. Then click **Program Flash.** Following messages would appear and that is fine.

Choose **Project** -> **Properties** to open the wizard and choose **Associated System Library**, then click **System Library Properties..**

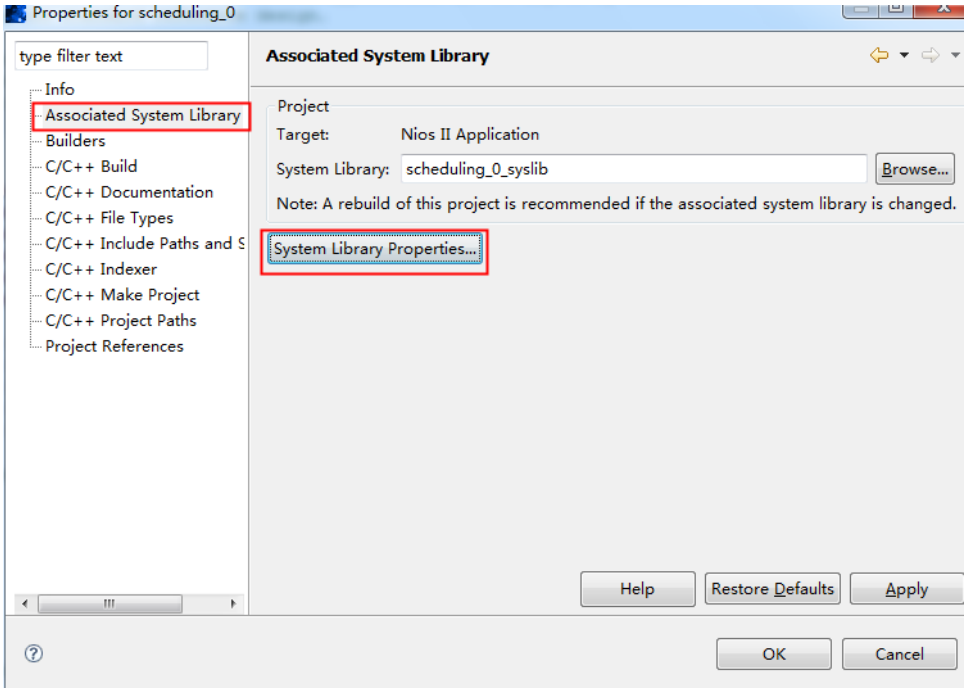


Figure 51. Properties of the project

Remember to set the memory location to ***SDRAM*** and uncheck **Small C library.** Small C library has no file relate operation command. See the figure below.

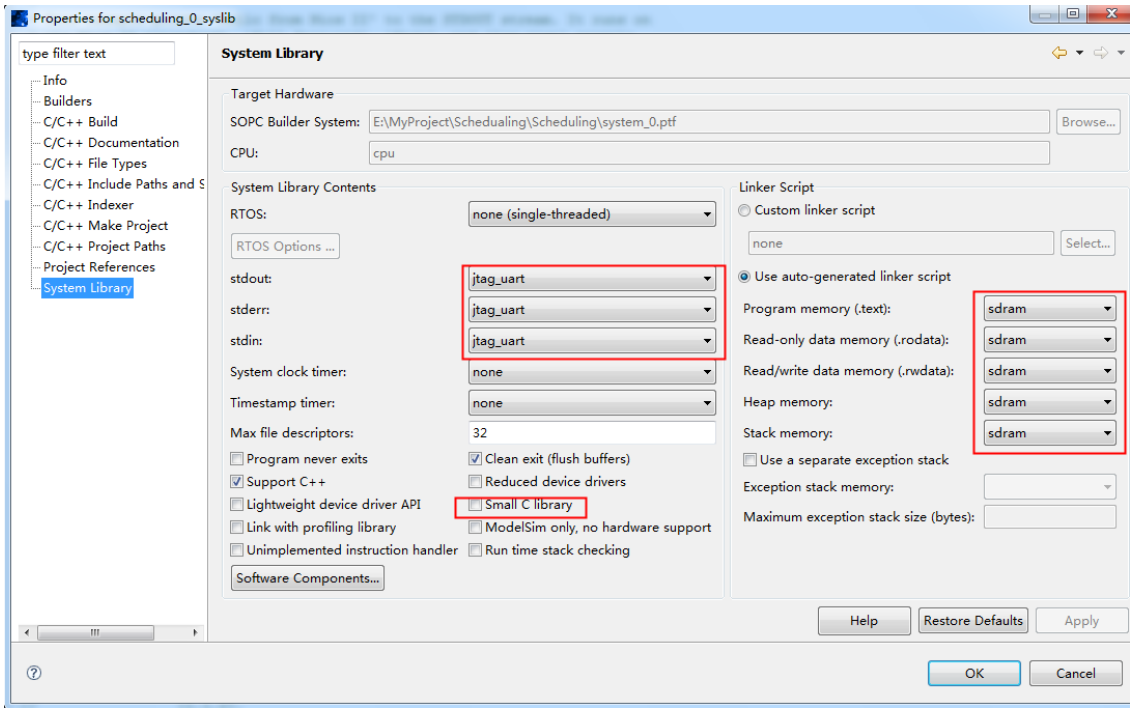


Figure 52. System Library

Then click Software Components, check the following information to make sure it is right.

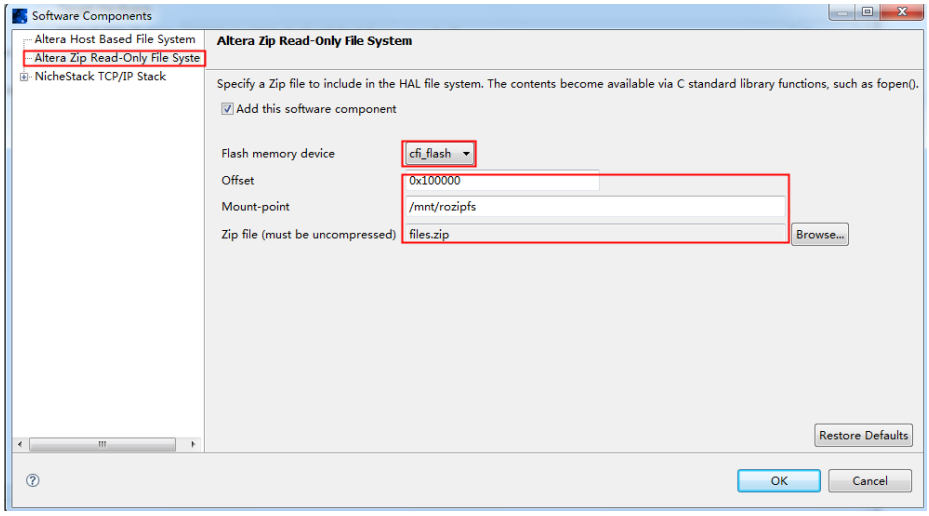


Figure 53. Software Components

Finally, build up our project and Debug it and we could see the results on FPGA.

REFERENCES

[1] “The smart grid: An introduction,” U.S. Dept. Energy, 2009.

[2] M. Shinwari, "A Water-Filling Based Scheduling Algorithm for the Smart Grid." *Smart Grid, IEEE Transactions on.,* vol. 3(2): 710-719, Feb. 2012

[3] R. Krishnan, “Meters of tomorrow,” *IEEE Power Energy Mag.*, vol. 6, pp. 92–94, Mar. 2008.

[4] (2009). "Ontario's MicroGeneration Feed-Tarriff." Available: http://renaud.ca/wordpress/?tag=cost-of-electricity

[5] “Ontario demand and market prices,” Independent Electricity System Operator, Sep. 2011 [Online]. Available: http://www.ieso.ca

[6] (2012). “How does Power Stoplight help the environment?” Available: http://www.powerstoplight.com/?page_id=10

[7] A. Lew and H. Mauch, *Dynamic Programming: A Computational Tool.* Springer-Verlag Berlin Heidelberg 2007

[8] (2013) “Ontario Hydro Rates: Time-of-use Pricing”, Apr. 2013, Available: http://www.ontario-hydro.com/index.php?page=current_rates

[9] M. J. Osborne and A. Rubinstein*, A Course in Game Theory*. Cambridge, MA: MIT, 1994.

[10] Osborne, Martin J., and Ariel Rubinstein. *A Course in Game Theory*. Cambridge, MA: MIT, 1994. Print

[11] Barr, N. (2004). *Economics of the welfare state*. New York, Oxford University Press (USA)

[12] Nash, J. F. "Non-Cooperative Games." *Ann. Math.* 54, 286-295, 1951.

[13] “DE0-Nano Development and Education Board,” Altera, Available: http://www.altera.com/education/univ/materials/boards/de0-nano/unv-de0-nano-board.html

[14] “Cyclone II Device Handbook,” Altera, Available: http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf

[15] “Nios II Processor Reference Handbook,” Altera, Available: http://www.altera.com/literature/lit-nio2.jsp?GSA_pos=1&WT.oss_r=1&WT.oss=NIOS%20II%20Processor%20System

[16] Manual for Breville the Smart Oven, Available: http://www.brevilleusa.com/media/mediaappearance/4539/BOV800XL.pdf

[17] Manual for Breville the Risotto Plus, Available: http://www.brevilleusa.com/media/mediaappearance/12563/BRC600XL_IB_A12_FA_LowRes.pdf

[18] Features for Samsung 7.5 cu. Ft. King-size Capacity, Electric Touch Screen LCD Front-Load Dryer, Available: http://www.samsung.com/us/appliances/washers-dryers/DV457EVGSGR/AA-specs

[19] “Estimating Appliance and Home Electronic Energy Use”, Energy.gov, August 31, 2012, Available: http://energy.gov/energysaver/articles/estimating-appliance-and-home-electronic-energy-use