Third Prize

Intelligent Solar Tracking Control System Implemented on an FPGA

Institution: **Institute of Electrical Engineering, Yuan Ze University**

Participants: **Zhang Xinhong, Wu Zongxian, Yu Zhengda**

Instructor: **Professor Huang Yingzhe**

Design Introduction

In today's high-tech environment, energy has become the impetus for socio-economic development. Since the Industrial Revolution, humans have used fossil fuels as their primary energy source. However, the amount of fossil fuels on the earth is limited, and their use has caused unprecedented changes to the global ecological environment and climate. Gases from burning fossil fuels can build up in the atmosphere, becoming thicker and thicker to produce greenhouse effects such as rising global temperature and sea level. These effects will dramatically alter our living environment. Fortunately, humans are becoming more conscious of environmental protection, and are seeking new energy sources that cause less pollution and do not threaten the environment. As a free, nonpolluting, inexhaustible energy, solar energy is ideal for generating electricity. Currently, generating electricity by solar energy is inefficient, so our project focuses on how to improve its efficiency.

A solar panel receives the most sunlight when it is perpendicular to the sun's rays, but the sunlight direction changes regularly with changing seasons and weather. Currently, most solar panels are fixed, i.e., the solar array has a fixed orientation to the sky and does not turn to follow the sun. To increase the unit area illumination of sunlight on solar panels, we designed a solar tracking electricity generation system. The design mechanism holds the solar panel and allows the panel to perform an approximate 3-dimensional (3-D) hemispheroidal rotation to track the sun's movement during the day and improve the overall electricity generation. This system can achieve the maximum illumination and energy concentration and cut the cost of electricity by requiring fewer solar panels, therefore, it has great significance for research and development.

Solar Tracking Control System

Our high-performance solar tracking system has multiple functions and uses two motors as the drive source, conducting an approximate hemispheroidal 3-D rotation on the solar array (see Figure 1). The two drive motors are decoupled, i.e., the rotation angle of one motor does not influence that of the other motor, reducing control problems. Additionally, the tracker does not have the problems common to two-axis mechanical mechanisms (that one motor has to bear the weight of the other motor). This implementation minimizes the system's power consumption during operation and increases efficiency and the total amount of electricity generated.

Figure 1. Solar Tracking Array Architecture Scheme

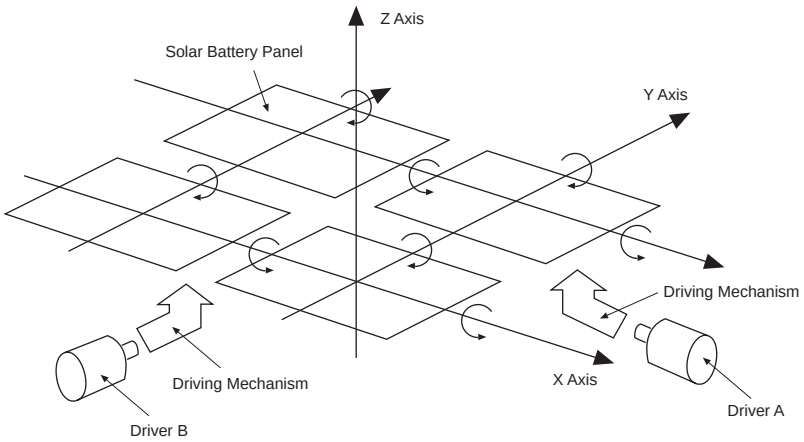


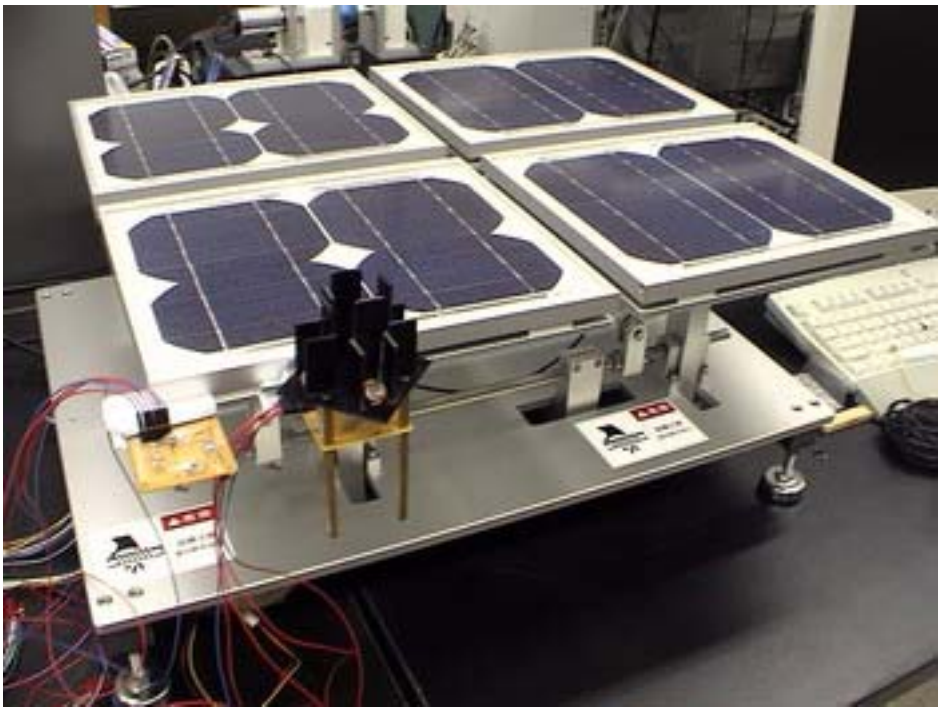
Figure 2 shows the solar tracking system we designed based on the considerations described previously. The mechanism must support the solar panel and allow the panel to conduct 3-D rotation within a certain amount of space. The array-type mechanism has two advantages:

- *High photoelectric conversion efficiency*—Because the flexible panel of the solar tracker array can conduct 3-D rotation, tracking the sun in real time, the system efficiently performs photoelectric conversion and production.

- *Simple, energy-saving controls*—The two rotational dimensions of the array solar tracker are controlled by two independent drive sources. The rotation angles are decoupled and neither one has to bear the weight of the other one. Additionally, the overall movement inertia is dramatically reduced.

We used the Altera® Nios® II processor to perform solar tracking. The design combines a Nios II processor with a two-axis motor tracking controller to integrate peripherals such as microprocessor, memory, and I/O into one Altera FPGA based on system-on-a-programmable-chip (SOPC) concepts. This integration accelerates development while maintaining design flexibility, reduces the circuit board costs with a single-chip solution, and simplifies product testing.

Figure 2. Complete Solar Tracking Control Platform



Function Description

Our design includes three modes: balance positioning, automatic mode, and manual mode.

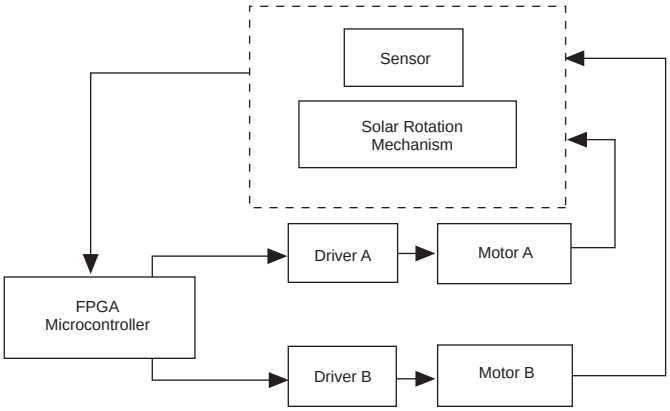
- *Balance positioning*—When setting the solar platform default, we used a mercury switch for balance positioning. The switch sets the four boundaries of the platform and prevents the solar panels's four tilting boundaries from hitting the mechanism platform and damaging it or the motor.

- *Automatic mode*—In this mode, the system receives sunlight onto the cadmium sulphide (CdS) photovoltaic cells and the CdS acts as the main solar tracking sensor. The sensor feeds back to the FPGA controller through an analog-to-digital (A/D) device. The Nios II processor is the main control core and adjusts the two-axis motor so that the platform is in the location for optimal, efficient electricity generation.

- *Manual mode*—If the system has a fault or needs to be maintained, we can switch the system to manual mode. In this mode, we can adjust the system's position to check it or perform repairs.

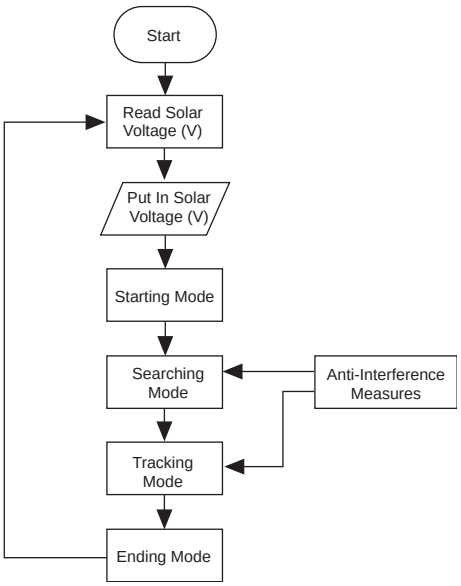
Figure 3 shows the block diagram of the solar tracking system.

Figure 3. Solar Tracker Control Block Diagram



We implemented the system's logic low design using the Nios II processor control circuit. Figure 4 shows the tracking control flow chart. The system starts when we turn on the tracking control circuit's power supply switch. The tracking control circuit performs system tracking, energy saving, and system protection, as well as a designed control mode and external anti-interference measures. External interference includes weather influences, such as wind, sand, rain, snow, hail, salt damage (i.e., salt erosion on the mechanism), etc.

Figure 4. Tracking Control Flow Chart



Performance Parameters

We used the following modules to build our solar tracking system:

- *Balance sensor module*—At the initial system reset, the four switches of the balance sensor are all powered-on at horizontal balance status.
- *A/D module*—We used the AD0804 A/D converter, which has a 100-µs conversion time and 8-bit resolution.

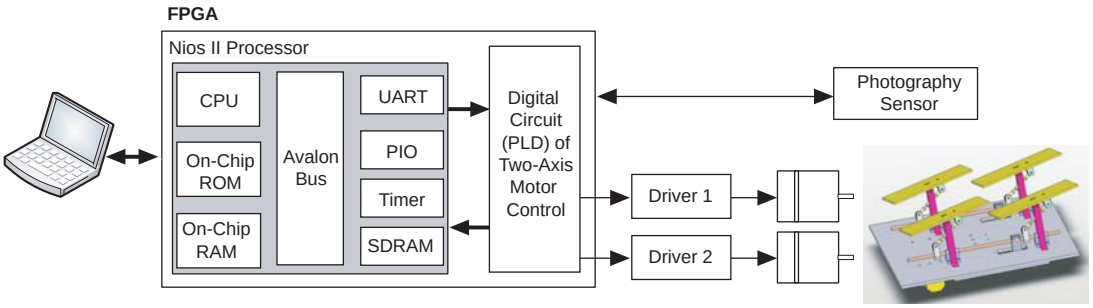
- *Motor control module*—We used a 100-µs circle step motor to control the motor's speed.
- *CPU*—We created a tailor-made 32-bit RISC-based CPU that has the capacity to control the solar system.

For precision, the fuzzy control time should not be more than 0.1 second. In addition to the reset balance (positioning level of 0 degree = 262,144), the balance sensor can also bound the X and Y axes.

Design Architecture

As shown in Figure 5, the Nios II processor is the control center and integrates our two-axis control chip. The system determines which data is fed back to the FPGA using a photography sensor. It conducts the tracking control rule operation to calculate the angle required by the motor and adjusts motor's current angle. It also moves the solar panel to achieve optimal power.

Figure 5. System Architecture



For the hardware design, we first used a balance sensor to set the system's zero point. Then, we designed a tracking sensor to determine the orientation of the solar light source. The signals fed back by the sensor form the basis of the controller input. The control design outputs the signals to control the two-axis step motor and the solar tracking control system. The following sections introduce the hardware.

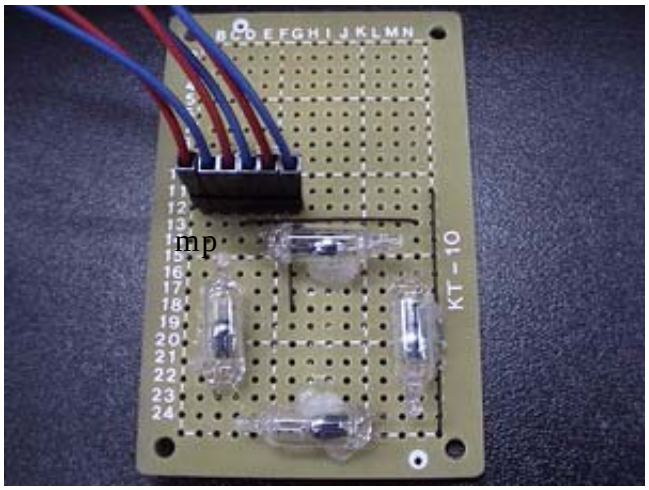
Balance Sensor

For the initial reset balance, we used a mercury switch (also called a tilt switch), which is a kind of circuit switch. Its main body is a mini container that is connected with electrodes and contains a drop of mercury, and usually the container is a vacuum or infused with inert gas (see Figure 6). Because of gravity, the mercury bead flows towards the lower position in the container. If it contacts the two electrodes simultaneously, the circuit closes and the power-on switch opens. According to our design principle, we set four switches (east, west, south, and north) in the mechanism design and fixed the mechanism. If the four switches are all powered-on, the mechanism balances. Figure 6 also shows the balance sensor stereogram.

Figure 6. Mercury Switches and Stereogram

Mercury Switches

Balance Sensor Stereogram



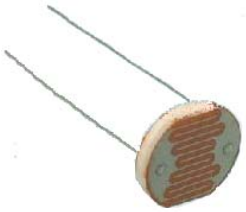
Sensor Design

One of our key modules is the sensor. Because the sensor tracks the solar light source's orientation, selecting the right tracking sensor is very important. CdS sensors (see Figure 7) are cheap, reliable, and photo-sensitive. In our design, the CdS sensor provides the following advantages:

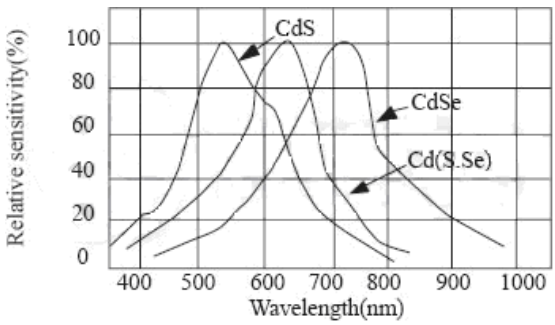
- Without polarity (ohmic structure), the CdS sensor is easy to use.
- CdS sensors have a photo-variable resistor in which the internal impedance changes with the intensity of light energy.
- When the ambient light brightens, the CdS sensor's internal impedance reduces.
- The CdS sensor's photo sensitivity (i.e., spectral characteristics) is 0.4 to 0.8 mm, which is close to the wavelength scope of visible solar light (0.38 to 0.76 mm), as shown in Figure 7.

Figure 7. CdS Stereogram and Sensitivity Scope

CdS Stereogram



CdS Sensitivity Scope



Tracking Sensor Design

The tracking sensor is composed of four similar CdS sensors, which are located at the east, west, south, and north to detect the light source intensity in the four orientations. The CdS sensor forms a 45° angle with the light source. At the CdS sensor positions, brackets isolate the light from other orientations to achieve a wide-angle search and quickly determine the sun's position (see Figure 8). The four sensors are divided into two groups, east/west and north/south. In the east/west group, the east and west CdS sensors compare the intensity of received light in the east and west. If the light source intensity received by the sensors are different, the system obtains signals from the sensors' output voltage in the two orientations. The system then determines which sensor received more intensive light based on the sensor output voltage value interpreted by voltage type A/D converter (ADC) and ADC0804 device. The system drives the step motor towards the orientation of this sensor. If the output values of the two sensors are equal, the output difference is zero and the motor's drive voltage is zero, which means the system has tracked the current position of the sun. The north/south sensors track the position of the sun similarly. Figure 9 shows the sensor stereogram.

Figure 8. Tracking Sensor Internal Design

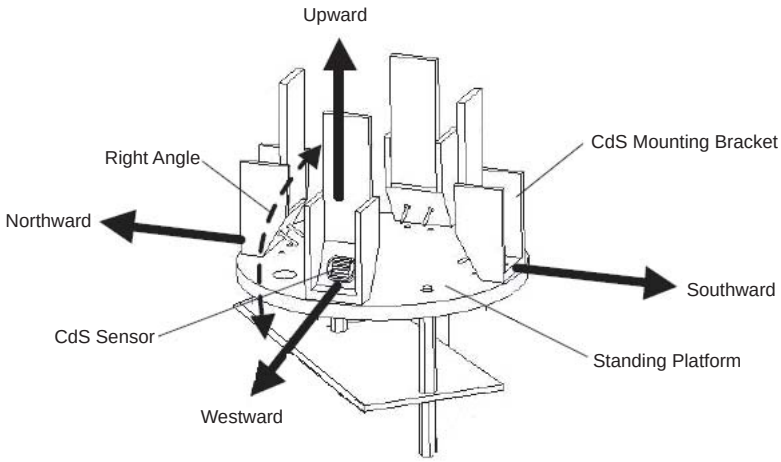
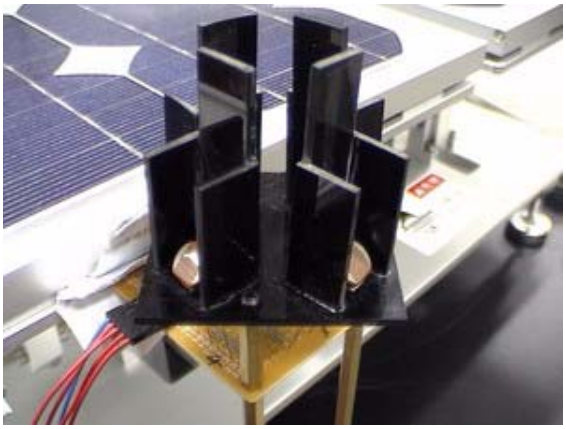


Figure 9. Tracking Sensor Stereogram



ADC

Generally, measured continuous signals such as voltage or current are analog signals. An ADC converts analog signals to digital signals. Digital signals can minimize noise interference during signal

transmission and debug noise interference with encoding technology. Additionally, digital signals are easy to store.

The ADC0804 ADC is a 20-pin device with 8-bit resolution and a single channel operating with a 5-V single power supply. Its analog input voltage scope is 0 V to 5 V. The power consumption is 15 mW and the conversion time is 100 µs. Because the resolution is 8 bits, there is a 256-step quantization. If the reference voltage is 5 V, each step is 5/256 = 0.01953 V. 00000000 (00H) represents 0.00 V and 11111111 (FFH) represents 4.9805 V. The unadjusted error of the ADC0804 device is 1 least significant bit (LSB), i.e., 0.01953 V, including the full-scale error, offset error, and nonlinear error.

Figure 10 shows the ADC0804 pins. `D0` through `D7` is the 8-bit output port. When both $\overline{CS}$ and $\overline{RD}$ are low, digital data is sent to the output port. When $\overline{CS}$ or $\overline{RD}$ is high, `D0` through `D7` float. $\overline{WR}$ is the control signal for initiating conversion. When $\overline{CS}$ and $\overline{WR}$ are low, the ADC0804 device performs deletion; when $\overline{WR}$ goes high, the device performs conversion. `CLK IN` is the time sequence input with a frequency scope of 100 to 800 KHz. $\overline{INTR}$ is high during conversion and changes to low when conversion ends. `Vin(+)` and `Vin(-)` are differential analog signal inputs, usually single-ended inputs, and `Vin(-)` is grounded. The ADC0804 device has two ground ends, `A GND` and `D GND`. `Vref/2` is half of the reference voltage input value if the overhead connection, `2Vref`, is equal to the power supply voltage, V_{CC}. The ADC0804 device is embedded with a Schmitt trigger as shown in Figure 11. If resistance and capacity are added to `CLK R` and `CLK IN`, the time sequence, which is required by operating the ADC, is generated with the following frequency:

$$f_{CLK} \approx \frac{1}{1.1RC}(Hz) \quad (1)$$

In the equation, the time sequence signal is decided by R and C and the signal does not need to be added with `CLK IN`. Figure 12 shows the ADC0804 circuit diagram. The input analog signal is controlled by variable resistance `VR2` and input from `Vin(+)` end with `Vin(-)` being short. `2Vref` is provided by `R1`, `R2` and `VR1`; `C1` and `R3` control the time sequence of the circuit. $\overline{CS}$ and $\overline{RD}$ are grounded to create the chip enable. $\overline{WR}$ and $\overline{INTR}$ receive the `SW1` switch to emulate control signals.

Figure 10. ADC0804 Pin Function Diagram

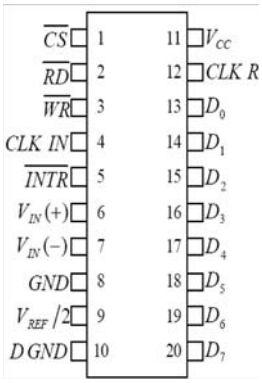


Figure 11. ADC0804 Internal Time Sequence Generation Circuit

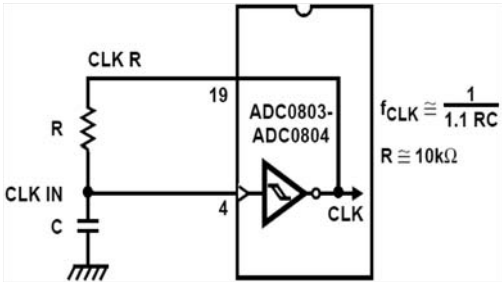


Figure 12. ADC0804 Circuit Diagram

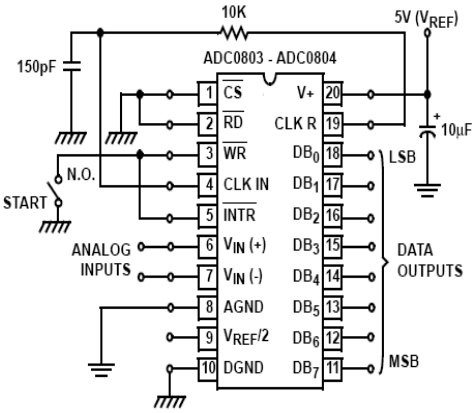


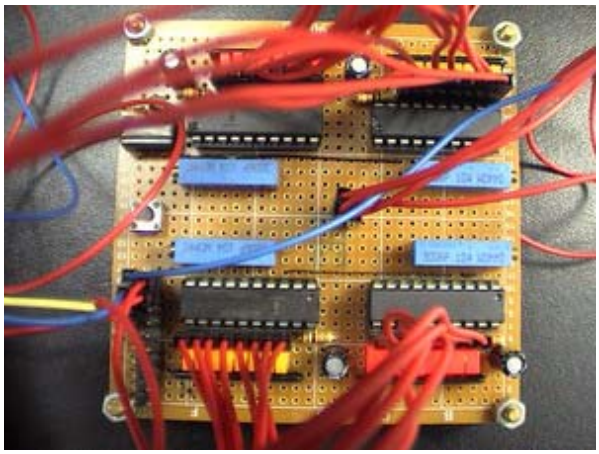
Table 1 shows the ADC specifications.

Table 1. ADC Specifications

Operating voltage	+5 V DC
Analog voltage input scope	$0 \le V_{in} \le +5$ V DC
Resolution	1/256
Conversion output value	0 to 255
Conversion frequency	f_{ck} = 1/(1.1 x R x C)
Conversion error	± 1 LSB
Reference voltage	+ 2.5 V DC

Figure 13 shows the completed ADC circuit stereogram.

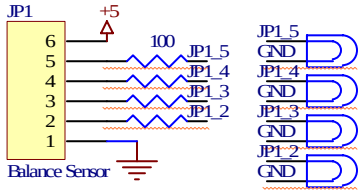
Figure 13. Complete ADC Circuit Stereogram



Sensor Production

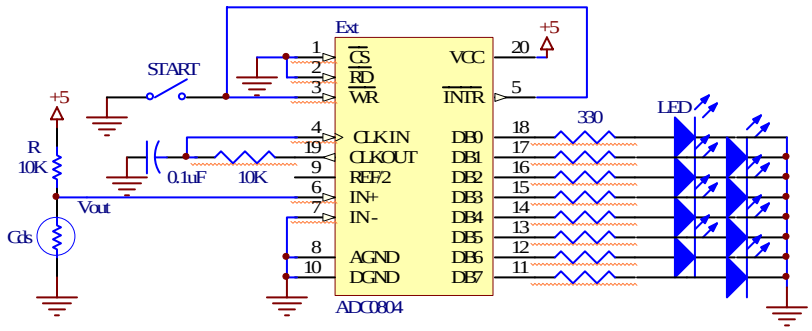
Figure 14 shows the balance sensor circuit diagram for the default mercury switch.

Figure 14. Balance Sensor Circuit Diagram



The CdS sensor's output signals generated by the solar light source are the input signals for the ADC chip's sixth pin. They are converted into analog signals and 8-bit output signals via the eleventh through eighteenth pin. Then, the signals are sent to the FPGA as input signals. Figure 15 shows the complete circuit diagram.

Figure 15. ADC Circuit Diagram



Design Methodology

Based on our experience, we know that if we make our solar panel perpendicular to sunlight, the illumination is strongest and electricity efficiency is highest. When we cannot adjust the actual position with accurate instruments, we measure, recognize, and determine the position with our eyes. Fuzziness might not be bad; sometimes we must accept information using a fuzzy method because we cannot know everything fully. General objects under control require a large number of complex mathematical formulas. To master controls with fuzzy features, scientists developed fuzzy theory. In application, the theory gives high importance to human experience and master degree for the properties of things, but does not advocate resolving problems with complex mathematical analysis and modes. In the following sections, we introduce fuzzy theory and describe how we use it in our Nios II control system.

Fuzzy Logic Control Design

Since professor L. A. Zadeh of the University of California at Berkeley proposed the concept of fuzzy sets in the academic journal, *Information and Control* in 1965, fuzzy theory has boomed. The theory emphasizes that most knowledge can be expressed with language, i.e., we can fuzzify all knowledge fields. Implementing the theory provides a wider application scope and error tolerance and is suitable for nonlinear systems in real life.

Early in 1974, professor E. H. Mamdani of Queen Mary, University of London, successfully applied fuzzy control to the automatic operation of a steamer. In 1980, a Danish company, F.L. Smidth, used fuzzy controls on a cement kiln. The corporation transformed the operation statuses of the cement kiln and their solutions into language-type control rules and controlled the cement kiln by computers. Additionally, the Sendai Municipal Subway of Japan, which was launched in July 1987, successfully used fuzzy controls to automatically operate trains. Some consumer electronics also use the theory.

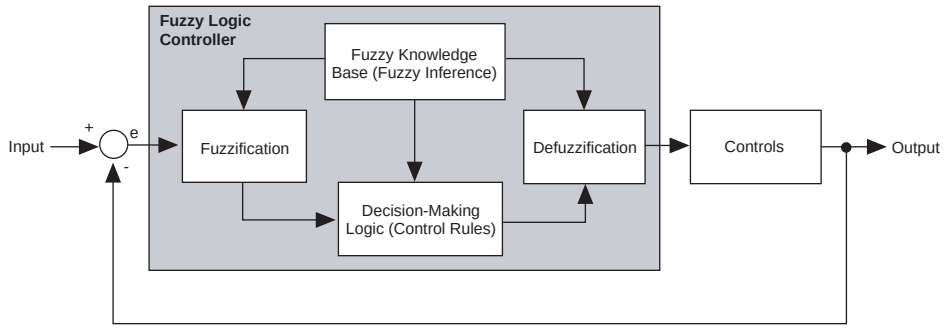
Fuzzy theory is a science closely related to our lives. Because it describes things with language, it is easy to accept. In real life, most descriptions are fuzzy. For example, when we say “sweet fruit” or “drive fast,” sweet and fast are not accurate values but simply a description of the degree. However, people can easily understand the meaning from the description. During nearly 40 years of development, achievements in fuzzy theory have been recognized and the application of the theory has extended to all scientific fields, including:

- *Control engineering*—Intelligent controls, vehicle electronics, Sendai Municipal subway, etc.
- *Image identification*—Image processing, voice identification, signal processing, etc.
- *Consumer electronics*—Washing machines, refrigerators, coolers, etc.
- *Other*—Data management, teaching appraisals, financial management, etc.

Fuzzy Logic Controller Structure

Figure 16 shows the fuzzy logic controller (FLC) structure. The FLC is composed of a fuzzification interface, knowledge base, inference engine, and defuzzification interface.

Figure 16. Fuzzy Logic Controller Basic Structure



Fuzzification Interface

The input of a common controller is a specific numeric value, but the knowledge base for fuzzy control is expressed with language. The system must turn numeric values into language and corresponding domains to allow the fuzzy inference engine to inference. This transformation is called fuzzification.

Knowledge Base

Knowledge base is the inference basis for fuzzy control. It defines all relevant language control rules and parameters. The knowledge base (including the database and rules base) is the core of a fuzzy control system.

Database

Fuzzy control rules have two types of presentations. The first is a state evaluation type that evaluates the system state at time (t) and calculates the fuzzy control action at certain point in time using the control rule and database input variables. The second is an object evaluation type that predicts current and future control actions. It determines whether the control object is achieved and then decides whether to output a control command. The database is usually built based on the following sources:

- *Working and expert experience*—The fuzzy control rule is based on information obtained by a controlled system. Experience rules are the most important part of fuzzy control.
- *System self-learning*—The system has an off-line learning method and builds a rules base with the help of other algorithms; however, this method requires a system model to be established already.
- *Predication evaluation*—This is a traditional method. It uses mass tests to verify the result and updates the rules base with the latest results. This method takes a long time.

Rules Base

The rules base contains many rules presented as language. The following rule is presented as a simple conditional statement:

Rt IF x is At THEN y is Bt (2)

where:

- t is the statement number.
- IF is the statement antecedent proposing the conditions to determine whether the statement is true.
- THEN is the statement consequence representing the inference result according to the conditions.

- Antecedent x is the input variable of the fuzzy system and is used to measure the system state.
- Consequent y is the output variable of the fuzzy system and is used to control the system.

The actual variable number can be increased or reduced according to the status of controlled system. A_t and B_t are the fuzzy concepts presented by language. For example, the definitions of tall, short, fast, and slow are different due to human subjective opinions and are difficult to present with data; instead they are defined using membership functions.

Fuzzy Inference Engine

As the most important part of fuzzy control, the fuzzy inference engine performs the actual decision-making process. The basic theory of the fuzzy inference engine is an approximate inference. The engine has two key inference methods: generalized modus pones (GMP) and generalized modus tollens (GMT). GMT is object-oriented inverse fuzzy theory, but GMP is forwarding linking inference modus. In GMP, when data is input, the output can be inferred according to rules; therefore, GMP is applicable for a fuzzy control inference mechanism. Its operation includes the following three calculations:

- Perform an AND operation for all the propositions of the antecedent of the triggered rule to obtain the antecedent fit.
- Perform an AND operation for all the propositions of the consequent corresponding to the antecedent fit of the triggered rule to determine how strongly true the rule is.
- Perform an OR operation for all consequents of all triggered rules.

Defuzzification

The reverse of fuzzification, defuzzification transforms the fuzzy inference engine's output values into equivalent assured values, making the assured value comply with the input signals of the controlled system. This process gives output control signals to the controlled system.

Solar Energy Controller Production

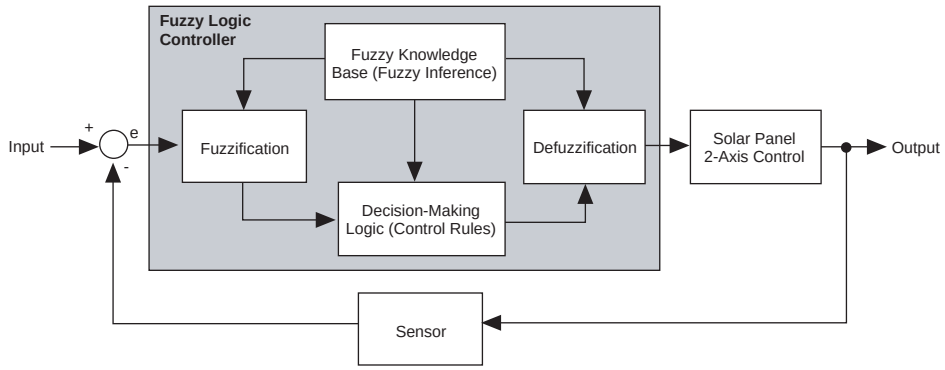
Although the solar tracking system's two drive motors can independently rotate without the problem of coupling, they inevitably have nonlinear phenomena in the moment of inertia (this is a common problem for 3-D rotation mechanisms). Therefore, the motors require a closed loop control. Although nonlinear phenomena exist in the moment of inertia control, it is not necessary for the solar tracking system to rotate very quickly due to the speed of the sun's movement. Therefore, we can use fuzzy control rules to control the motor operation while ensuring the system control mechanism's adjustability and fast response time.

We use fuzzy control theory as the control basis of motor driver. When implementing the hardware control circuit, we used a hardware description language such as VHDL and Verilog HDL to load the control program into the Nios II processor, which is the control center. Then, we created the sensor, decoder, and other devices to form a complete control loop, ensuring optimal electricity efficiency of the system.

Fuzzy Logic Controller Implementation

Our controller designed takes the measured value of the light strength received by the sensor as the feedback and implements control using many rounds of modifications. Figure 17 shows the basic fuzzy control system structure. The CdS sensor resistance changes with the light strength. It is converted by the ADC to obtain a partial pressure voltage. Fuzzy control takes the errors of the two groups in the vertical (southern and northern) and horizontal (eastern and western) axis as the fuzzy control input.

Figure 17. Solar Energy Fuzzy Control System Structure



The whole fuzzy controller can be designed in five steps.

Step 1: Definitions

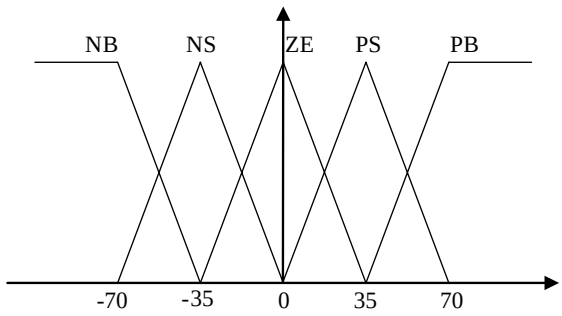
We define the input variables, output variables, and linguistic variables. Linguistic variables include the choice of input and output variables. In this study, for an axis we choose *e* as the input variable and *e* as the output variable, and define five triangle membership functions for one chosen linguistic variable.

- The input variable is Error e = b_pio - a_pio (3)
- The output variable is the number of seconds between the rotation and reversion adjustment of the step motor.
- For the domain of the output and input variables we used normalized discrete domain to set domain scope as [-75, 75].
- For the linguistic items, we defined five linguistic items for each linguistic variable, i.e., *e* ={NB, NS, ZE, PS, PB}. The linguistic items are defined as:
  - NB: Negative Big
  - NS: Negative Small
  - ZE: Zero
  - PS: Positive Small
  - PB: Positive Big

Step 2: Build Membership Input Functions

Based on the defined linguistic variables, we built the input membership functions. Figure 18 shows the membership function of errors in a horizontal or vertical orientation.

Figure 18. Fuzzy Control System Membership Function



Step 3: Set Up Fuzzy Rules Base

The setup of the fuzzy rules base is crucial because all states must be operated based on the rules defined in the rules base. To set up a fuzzy rules base smoothly, we adopted five fuzzy control rules expressed with IF THEN statements.

- *Rule 1*—If e is PB, then U_f is PB.
- *Rule 2*—If e is PS, then U_f is PS.
- *Rule 3*—If e is ZE, then U_f is ZE.
- *Rule 4*—If e is NB, then U_f is NB.
- *Rule 5*—If e is NS, then U_f is NS.

Step 4: Define Fuzzy Inference Engine

There are many fuzzy inference methods and different results are inferred with different methods. In this project we use the center of gravity method proposed by Mamdani as the defuzzification tool because the method is easy and reliable.

Step 5: Defuzzify Result

We defuzzify the result inferred by the fuzzy inference engine to convert the information into precise numbers. There are many methods for defuzzification. In this project we use center of gravity for defuzzification to obtain actual operation. The formula is shown in equation 4.

If the inference result is a fuzzy single value, the weighted mean method is most widely applicable. For n rules, w_i is the initiating strength of number i fuzzy rule, r_i is the inference result of number i fuzzy rule, and $\hat{U}_f$ is the output operation, the formula is:

$$\hat{U}_f = \frac{\sum_{i=1}^{n} w_i r_i}{\sum_{i=1}^{n} w_i} \tag{4}$$

Here we use the five fuzzy rules as follows:

$$\hat{U}_f = \frac{\sum_{i=1}^{5} w_i r_i}{\sum_{i=1}^{5} w_i} = w_1 r_1 + w_2 r_2 + w_3 r_3 + w_4 r_4 + w_5 r_5 \quad (5)$$

This defuzzification method can also be easily implemented with a digital circuit.

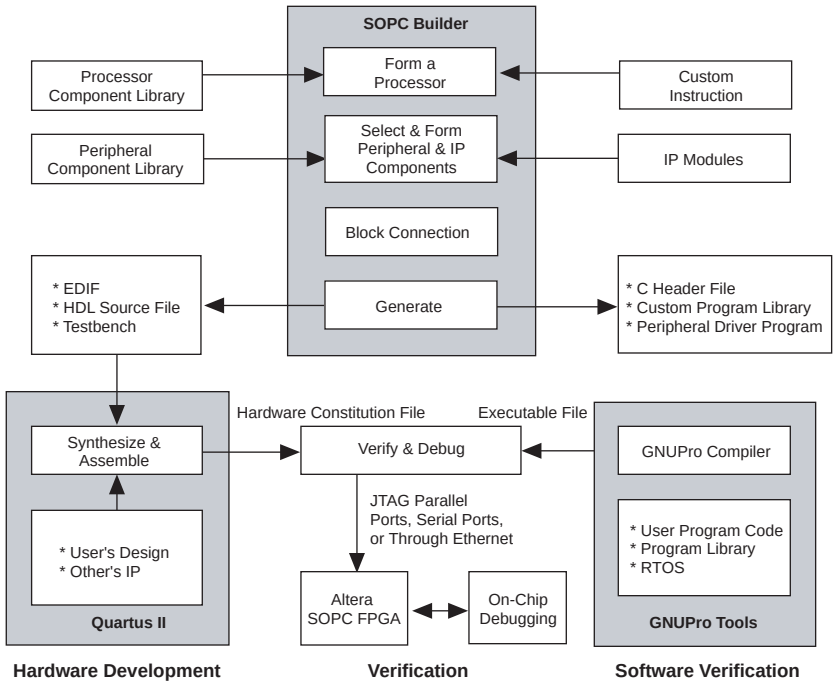
FPGA Program Design

As FPGAs have evolved in recent years, a single FPGA can accommodate more and more logic circuits. Building of a whole digital electronic circuit on a single chip provides a big edge in speed and power consumption, making system-on-a-programmable-chip (SOPC) designs gradually become the design tendency. As an emerging systematic design technology, SOPC design can incorporate the hardware system (including processor, memory, peripheral interface circuit, and user logic circuit) and software design on a single programmable chip.

SOPC System Design

Figure 19 shows the SOPC system development flow. First we define the system, including the processor, memory interface, peripherals, arbitrator, custom instructions, etc. Next, we generate the system with SOPC Builder. Then we perform the hardware and software design. During hardware design, we used the Quartus® II software to compile the logic circuit of the HDL programs and EDIF files. During software design, we used the GNUPro software development tool and software resources such as header files, library, monitors, and peripheral drivers to generate and edit application code. We debugged programs using Debug/Profile. To ensure the correctness of the hardware and software design, we used the ModelSim software for simulation. When we found an error, we went back to system generation so that SOPC Builder can modify and generate the system until it is right. Finally, we downloaded the hardware and software design into development board and prototyping kit for circuit verification.

Figure 19. SOPC System Development Flow



Hardware Design Solution

With Altera FPGAs, we could use the soft-core Nios II processor or the ARM ARM922T hard core. Altera provides the soft-core Nios II processor, which is a 16- or 32-bit RISC-based configurable embedded processor. For peripherals, Altera provides on-chip ROM, on-chip RAM, memory interfaces (such as SDRAM, SSRAM, and DMA controllers), series I/O (such as UART and Ethernet), parallel I/O (such as an input/output/two-way port, or PCI interface) as well as timers (such as a simple timer, frequency timer, and watchdog timer). For intellectual property (IP), Altera provides a PCI 32/33 bridge and Ethernet MAC. For the bus, Altera provides the Avalon® bus.

For the EDA hardware development tool, we used Altera's Quartus II software for place and route, design entry, compilation, and programming of the FPGA. The Quartus II software provides design entry methods such as VHDL, Verilog HDL, Altera Hardware Description Language (AHDL), and block/schematic entry. We used the LeonardoSpectrum software for circuit synthesis and the ModelSim® software for system simulation.

Software Design Solution

Altera provides the following relevant software development tools:

- Compiler
- Assembler
- Linker
- Debugger
- Monitor

- Libraries
- Utilities

Nios II Embedded Processor

Embedded systems were first designed for industrial computers. With the boom of information products and digital consumer electronics (CE), embedded systems became more popular. Embedded systems are used in a variety of applications, from information products, CE, and network products to portable devices. Because the ARM processor does not include FPGA-based IP, Altera embeds the ARM922T microprocessor hardware circuit into the FPGA to improve the system design integrity. In this project, integrating an FPGA control chip, it is convenient and feasible to use the Nios II processor to develop an SOPC embedded system. Using the SOPC Builder Integrated Development Environment (IDE) of Altera as an example, we can plan CPU types using the ARM core and Nios II IP (embedded into the CPU as VHDL or Verilog HDL) as the priorities. The main advantage of the SOPC Builder IDE is that it integrates and records the required circuits and peripherals to an FPGA to create a small circuit and maintainable hardware and software while accelerating product development and keeping a scalable design.

In this project we used the Altera Development and Education (DE1) board and implemented the Nios II embedded processor in the Cyclone II EP2C20F484C7 FPGA on the board. Figure 20 and Table 2 show the development board specifications. The resulting processor is a low-cost, high-performance FPGA and its system performance can be configured by customers as required, including:

- Three types of Nios II processors: fast (Nios II/f), standard (Nios II/s), and economical (Nios II/e). They are 32-bit instruction set structural systems.
- Complete peripheral hardware settings such as a timer, bridge, and counter, which SOPC Builder uses to integrate a complete microprocessor structure.
- Avalon switch structure, which can simultaneously process multiple units to improve system bandwidth with minimum FPGA resources.

Figure 20. DE1 Development Board

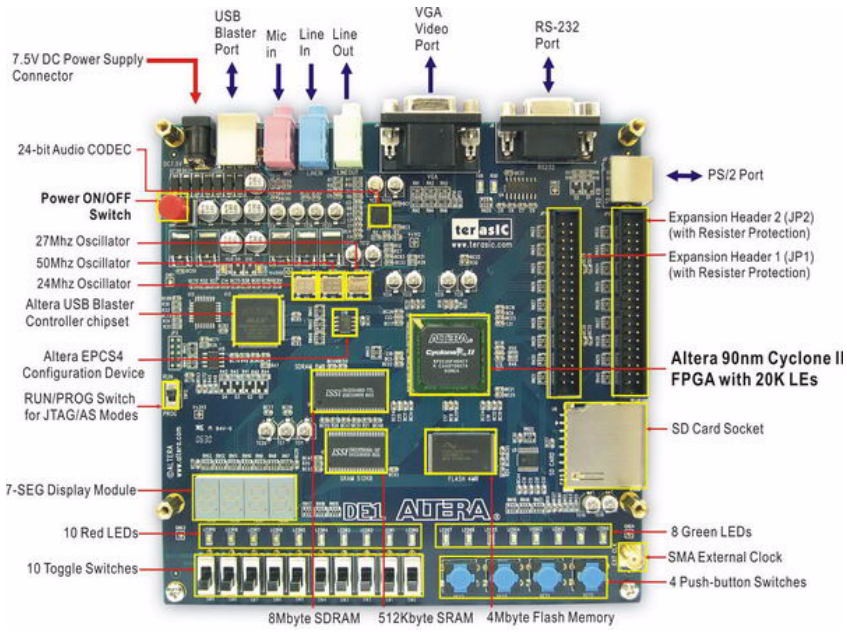


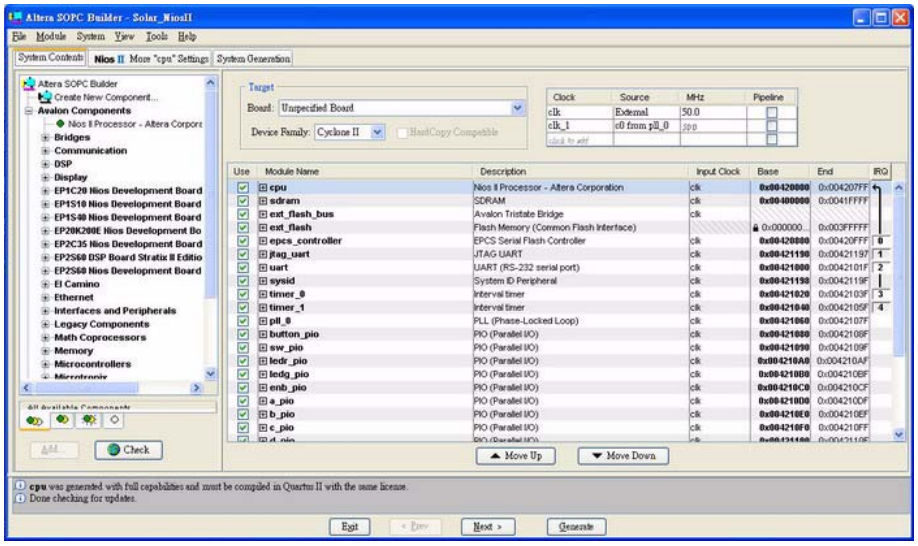
Table 2. DE1 Development Board Specification

Specification	Value
Total logic elements (LEs)	18,752
M4K RAM blocks	52
Total RAM bits	239,616
Embedded multipliers	26
PLLs	4
Maximum user I/O pins	315
FineLine BGA package	484

To design a Nios II system, we need to design the software and hardware. We used the Quartus II software and SOPC Builder to design the hardware. We used the Block Editor to generate the upper Block Design File (**.bdf**) and used the Text Editor to create VHDL or Verilog HDL hardware files. We also used the embedded MegaWizard® Plug-in Manager to generate low-order VHDL design files.

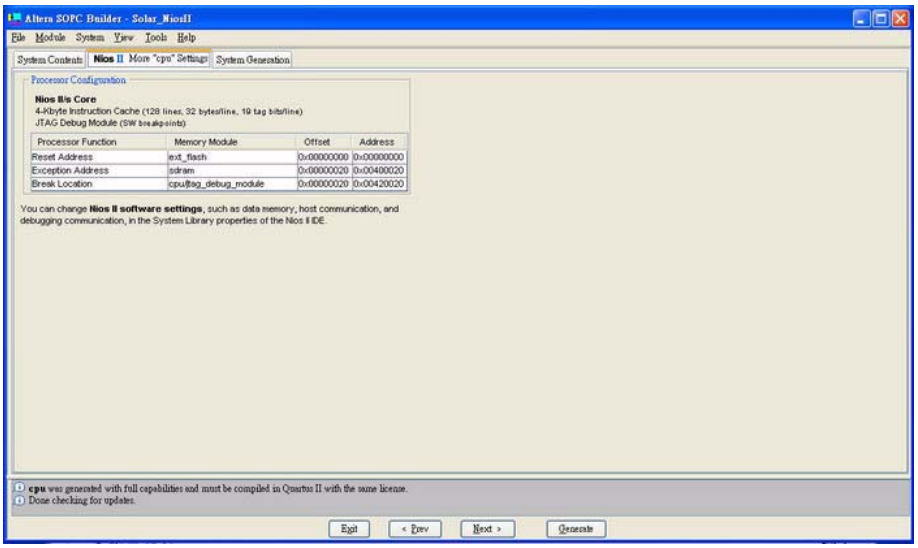
In the hardware design, we used SOPC Builder to establish Nios II system modules including the CPU, memory, and peripheral circuits. We selected the needed modules and set their parameters to designate the base address, interrupt request (IRQ), and system frequency as shown in Figure 21.

Figure 21. SOPC Builder System Content



Next, we designated the locations of the memory, boot chip, and interrupt vector table for the design and other system module settings as shown in Figure 22.

Figure 22. Nios II System Settings



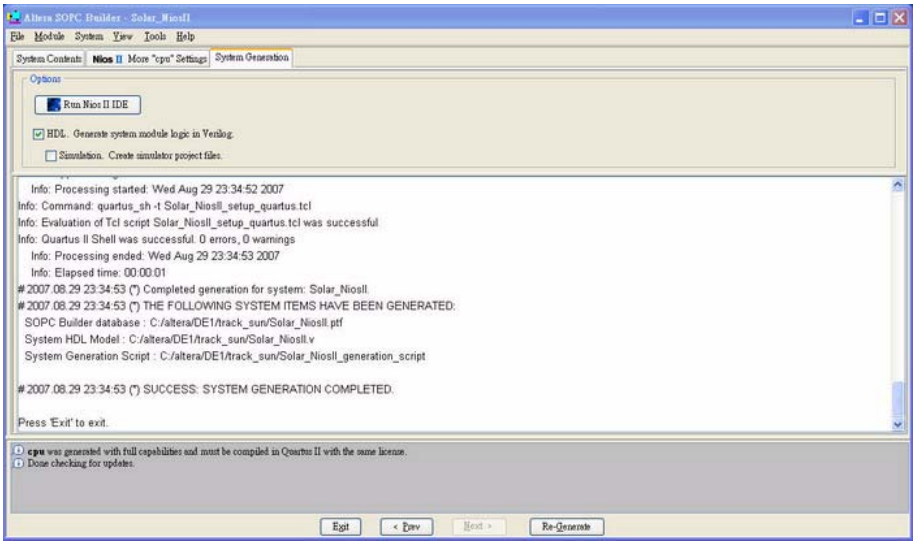
After completing the system design, we turned on the HDL option and turned on the chip model to be configured if the ModelSim simulation software has been installed. Then we clicked **Generate** to begin generation. This step performs the following actions:

- Generates the HDL file.
- Generates the simulation items and source files.

- Generates the C and Assembly language headers and source files.
- Compiles the system library.

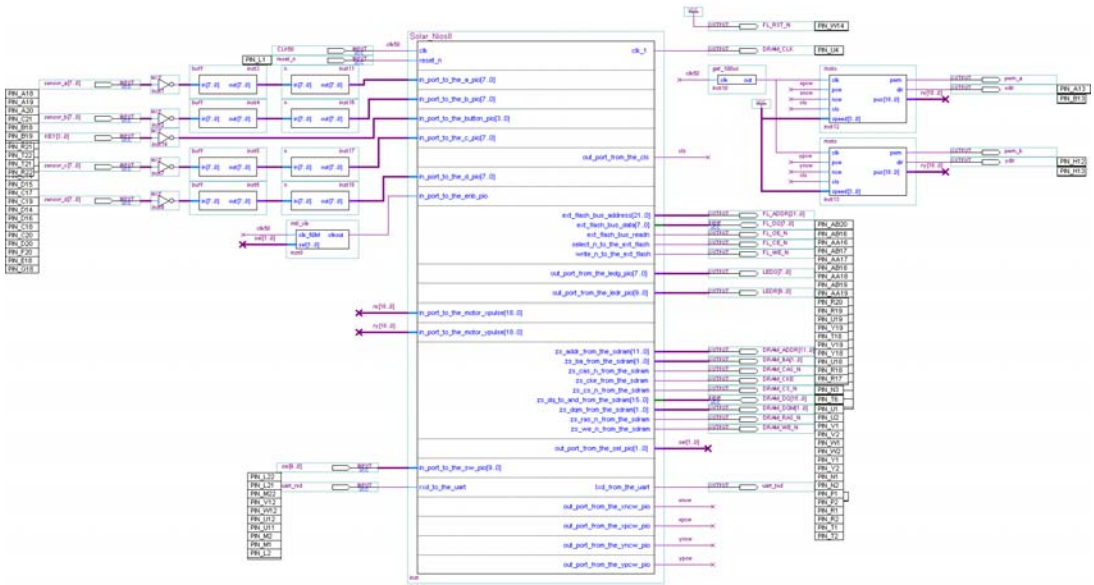
When system generation completes, we clicked **Exit** to go back to the Symbol Editor. See Figure 23.

Figure 23. SOPC Builder System Generation



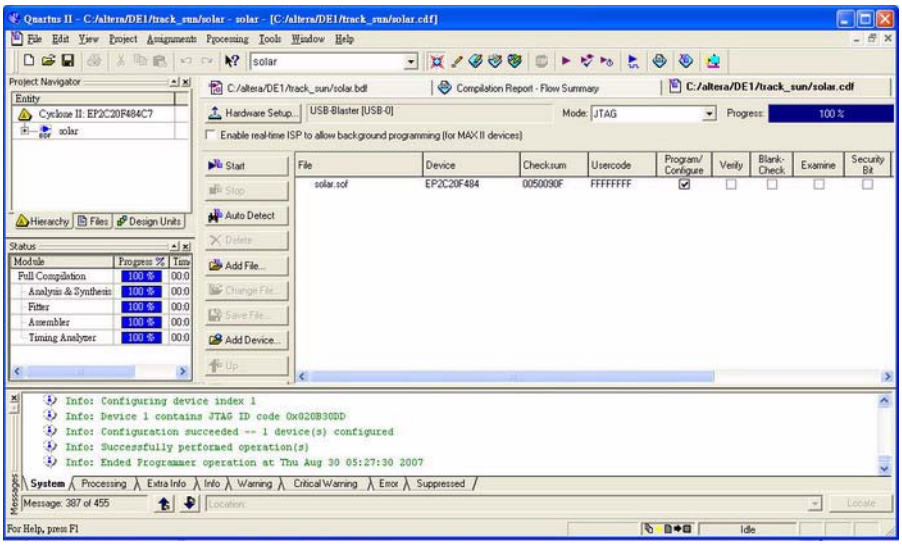
SOPC Builder generates a symbol of system module. Then, we added the circuit symbol into the BDF. See Figure 24. We added the input, output, and bidirectional pins, and then named each pin and other basic device symbols.

Figure 24. Quartus II BDF



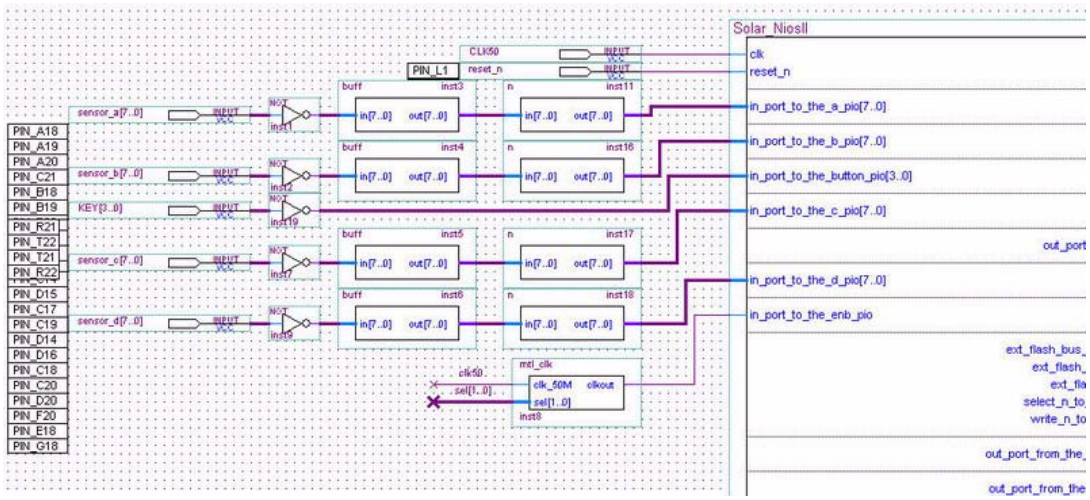
Finally, we downloaded the designed circuit to the development board using the Quartus II Programmer as shown in Figure 25.

Figure 25. Quartus II Programmer



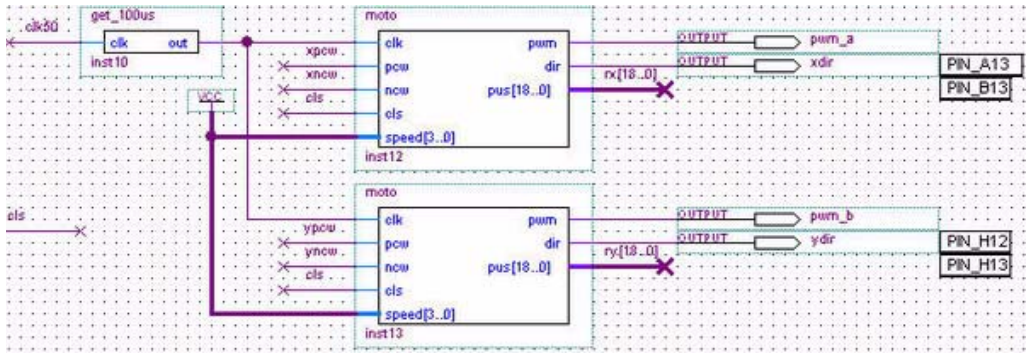
We input the decoding circuit for the CdS sensor using Verilog HDL. The signal is sent to the Nios II CPU, which controls it. See Figure 26.

Figure 26. Input Decoder Circuit using the CdS Sensor



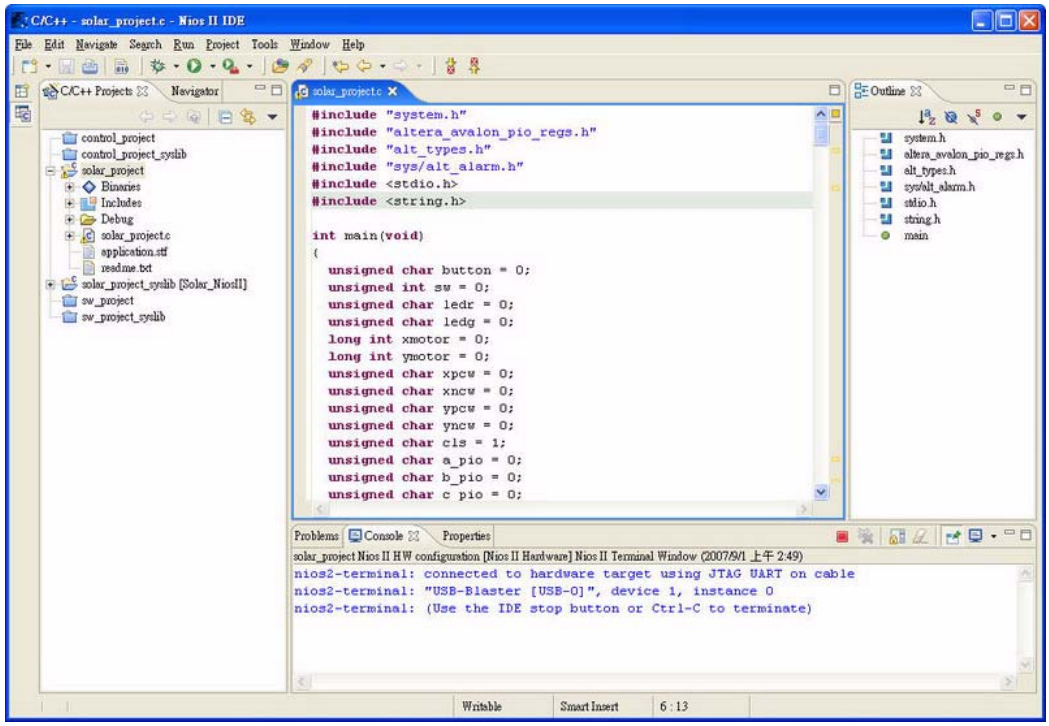
After determining the angle that the solar panels should reach, the Nios II CPU gives a signal to the motor to drive it. As shown in Figure 27, there are two motor modules controlling the system's X and Y axis.

Figure 27. Step Motor Module



After the required system hardware is generated, we can implement the controller using hardware or software. We used the Nios II 32-bit CPU to accelerate our fuzzy control rules. The Nios II IDE is shown in Figure 28.

Figure 28. Nios II IDE



Design Features

The Nios II processor helped us implement the design in the following ways:

- The major difference between our design and traditional, single-chip designs (such as the 8051 or PIC device) is that we added a fuzzy control rule to the circuit. Traditional chips cannot write VHDL. If we used traditional devices, we might need external logic circuits to implement the fuzzy controller, increasing the controller design volume and cost burdens. Alternatively, if we

used a microprocessor (MCU) plus an FPGA (excluding the Nios II processor), we would need to use a two-device assembly.

- For complex logic circuits, we can create a design using an FPGA and the Nios II processor with the Nios II IDE. The advantages of this method are that we can use the C language to write fuzzy algorithms and incorporate them into the Nios II CPU and we can compile the VHDL code into the FPGA to control the step motor. This implementation allows us to process algorithm operations and I/O control in parallel, improve integrated efficiency, and quickly implement and verify our hardware circuits.

Finally, we took the solar tracking control system for an outdoor test. For each 24-hour day, we used the step motor, FPGA development board, and control and sensing circuit to track the sunlight for about 30 seconds/hour. In one day, the solar panel is charged for about 8 hours, and in for the rest of the time it does not consume power (i.e., there is no standby mode to consume power). Therefore, we can calculate the energy data as shown in Table 3.

Table 3. Collected 24-Hour Solar Energy Radiation (Cloudy)

Test Method Measured Data	Fixed Solar Current Collection System	Smart Solar Current Collection System
Average 24-hour accumulated electricity generation of the solar panel P_S (J).	276,480	345,600
24-hour accumulated current consumed by step motor and control and sensing circuits P_C (J).	0	9,216
Average 24-hour accumulated net electricity generation $P_{total} = P_S - P_C$ (J).	275,480	336,384

Comparing the total net electricity generation of the fixed elevation angle control and smart solar tracking control, we found that smart system is superior to the fixed system.

Outdoor Measurement

In the study, we moved the solar platform to the top of the school building to test the fixed and smart systems (the results are shown in Table 3). Outdoor fixed and smart solar current collection system simulations are shown in Figures 29 and 30, respectively.

Figure 29. Outdoor Fixed Solar Current Collection System Test

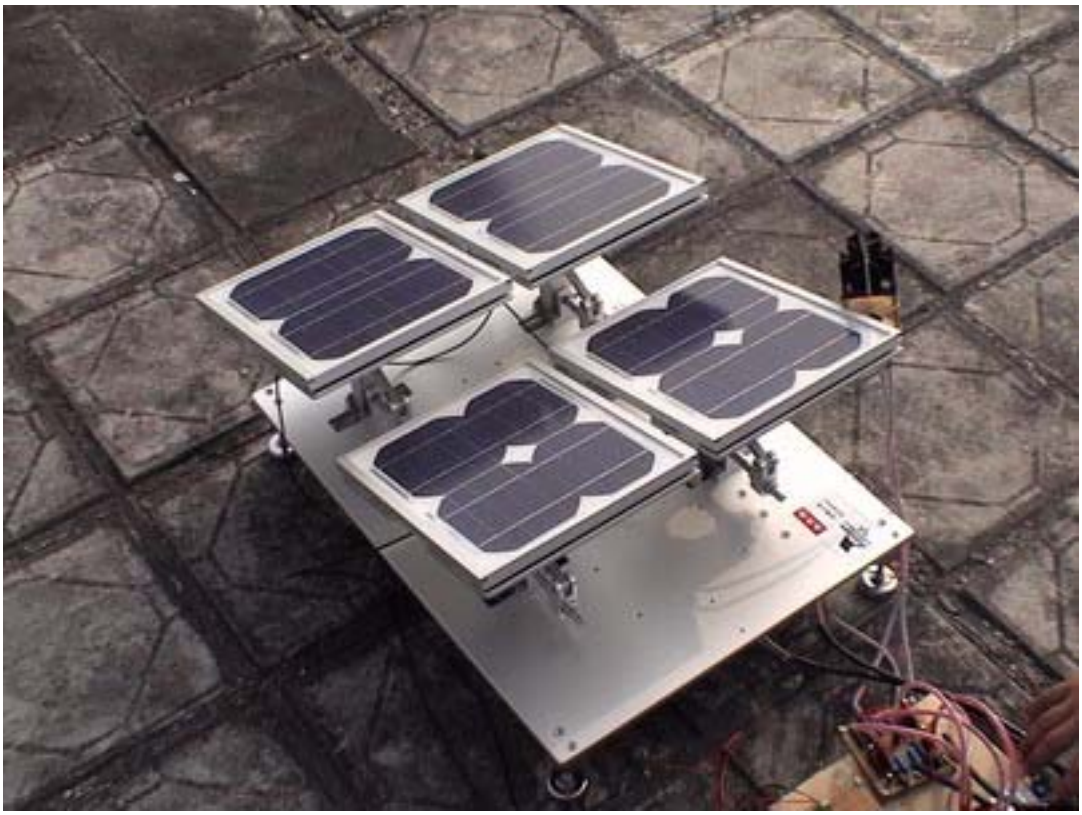
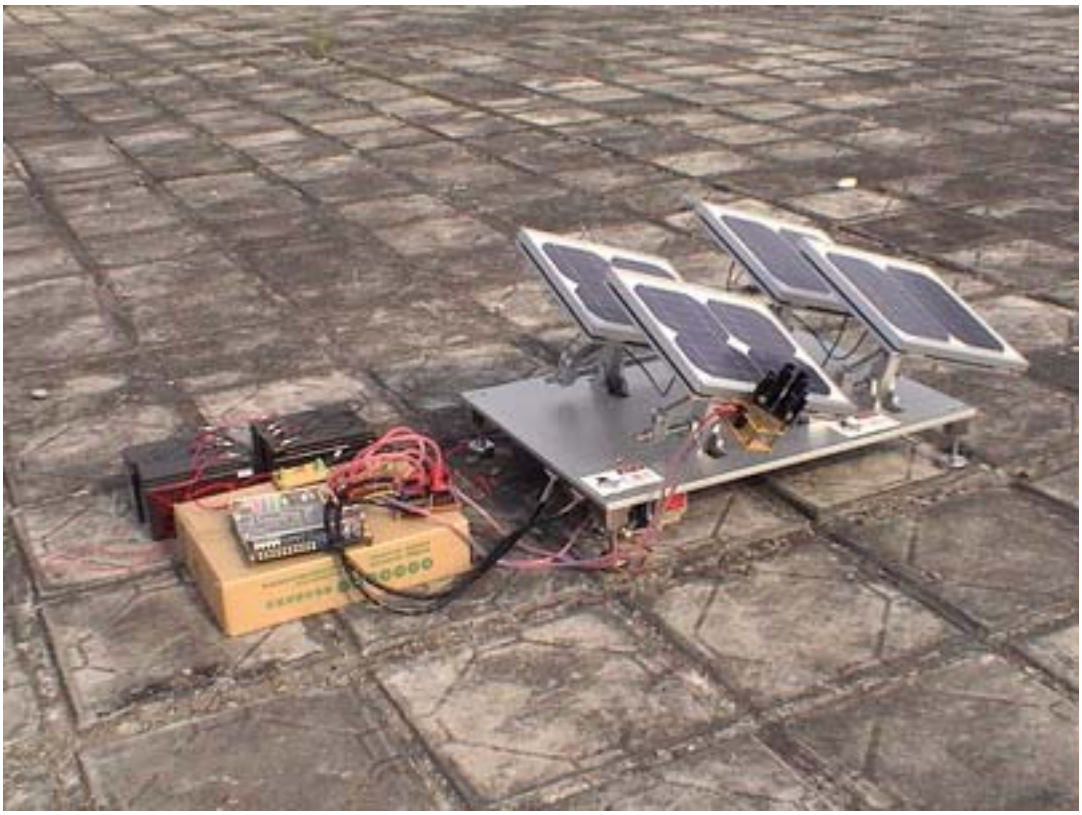


Figure 30. Outdoor Smart Solar Current Collection System Test



Indoor Measurement with Searchlight Simulating Sunlight

In this test, we used a searchlight as a simulated sunlight source, established a fixed and smart simulated sun running orbit, and used Visual Basic (VB) to transmit the measured voltage to notebook (NB) computer to measure the actual voltage through the RS-232 port. Figures 31 and 33 show the fixed and intellectual solar current collection system simulation, respectively. They show the different electricity generation efficiencies at the same angle. Figures 32 and 34 show the indoor fixed and smart solar current collection system voltages, respectively.

Figure 31. Indoor Fixed Solar Current Collection System Test

Figure 32. Indoor Fixed Solar Current Collection System Voltage

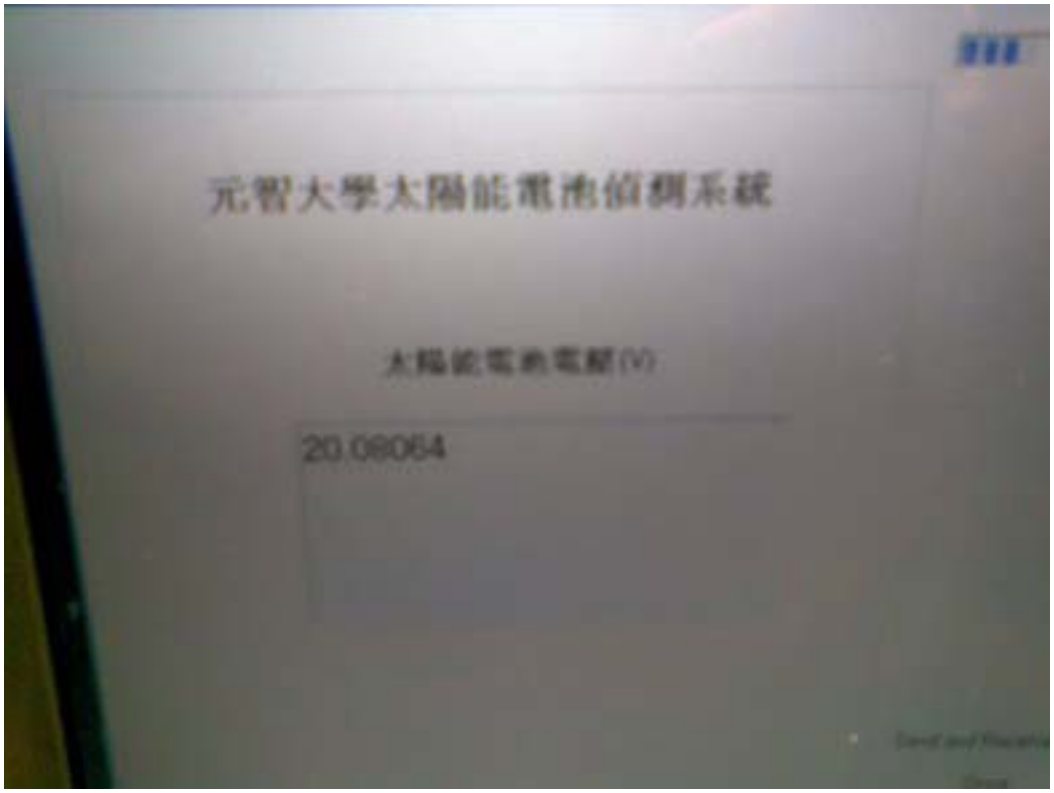


Figure 33. Indoor Smart Solar Current Collection System Test

Figure 34. Indoor Smart Solar Current Collection System Voltage

From Figures 32 and 34, we can see the voltage of the fixed solar current collection system is less than that of the smart solar current system. Therefore, the smart system is superior to the fixed system.

From the experimental results, we reached the following conclusions:

- By using the CdS sensor, the step motor, and Nios II processor in the FPGA to write and create different sunlight processing solutions (such as cloud intervention processing and reset solutions), we could establish a smart elevation angle control system to track a solar light source and improve the electricity generation efficiency of the solar batteries.
- The data in Table 3 shows that the smart elevation control system is a solar battery electricity generation system deserving wide promotion.
- Used with an FPGA, Altera's Nios II processor simplifies research and development and product testing and reduces circuit board costs, making it far superior to commonly used single devices.

Conclusion

We had used Altera FPGAs before we participated in the contest, so we had heard about the first-generation Nios processor. However, we only knew the Nios II processor from Altera's web site and collateral. This Nios II design contest gave us a good opportunity to enhance our ability in Nios II processor design and helped us thoroughly understand the Nios II processor's brand-new design concepts and features.

In this design, we used Altera's SOPC-based FPGAs. SOPC design represents a new system design technology, and SOPC Builder and the Nios II processor helped us see the powerful design technologies of software and hardware systems. Most traditional circuit designs are composed of hardware components building on a printed circuit board (PCB). If errors are found or the system needs to be improved or upgraded, the PCB must be redesigned. Adjusting and modifying the PCB is very inconvenient and increased the design cost and development period. In this contest, we accomplished our goals before the deadline. From concept design to system implementation, we only needed to model on the PC because Altera provides complete tools—including SOPC Builder, the Nios II IDE, and Quartus II development environment—to accelerate the software and hardware development. The Nios II processor is greatly improved over the Nios processor, and is more efficient, compact, and stable. Finally, integrating Nios II development, test environment, and C language compiler provides great convenience for users.