Third Prize

Nios II Processor-Based Fingerprint Identification System

Institution: College of Communication Engineering, Chongqing University

Participants: Ji Wang, Liang Wu, Yong Liu

Instructor: He Wei

Design Introduction

With the boom of information technology represented by computers since the 1960s, computer technology has begun to be used in the fingerprint identification field, bringing new thoughts, implementation methods, and processing approaches for automated fingerprint identification. Authorities, institutions, and universities have begun implementing fingerprint analysis and processing using computers. A computerized system that performs automated fingerprint identification is called an Automated Fingerprint Identification System (AFIS).

People often need to be identified in society. The common ID authentication methods such as keys, password, certificates, and IC cards provide identification using objects, which indirectly identify the object holder. These objects are not very accurate and have significant security risks, including counterfeited certificates and tokens, and decrypted or stolen passwords. With the development of image processing and pattern identification technologies, emerging identification technology based on biometric characteristics has become the focus of research and applications due to its unique reliability, stability, and convenience. As the earliest and most mature biometric identification technology in the pattern identification field, fingerprint identification technology integrates sensors, biometric technology, electronic technology, digital image processing, and pattern identification. Many automatic fingerprint identification systems are used worldwide, but fingerprint identification technology is not yet mature. China is behind in fingerprint collection and algorithm study, so the research of fingerprint identification algorithms and systems will play a significant role in theory and practice.

Systems with fingerprint minutia identification have wide applications in the fields of security, jurisdiction, military, finance and economy, information service, etc. Identification is needed in access control systems and other similar applications. Embedded fingerprint minutia identification systems

fully utilize biometric minutiae and are applicable for determining attendance in schools and enterprises, identifying residents in residential quarters, and so on.

The study and practice of police AFIS for two or three decades has laid a good foundation for civil AFIS. Specifically, the existing civil AFIS is easy to use, accurate, reliable, and affordable, allowing fingerprint identification products to enter our daily life. By replacing a personal identification code and password, fingerprint identification technology can guard against unauthorized access and illegal use of ATMs, cell phones, intelligent cards, desktop PCs, work stations, and computer networks. It can facilitate identification in telephone- or Internet-based financial deals and replace keys, certificates, stamps, and IC card readers in buildings or worksites.

As microelectronics technology advances, programmable logical controllers are becoming more diversified, faster, and more powerful. Today, many FPGAs support embedded soft-core processors to facilitate FPGA-based hardware development. For example, the Altera® Nios® II RISC CPU soft-core processor features pipelining and single instruction flow, can be embedded in an FPGA, and can leverage custom logic to build a FPGA-based on-chip system. Compared to an embedded hard core, a soft core is more flexible. Additionally, the faster FPGA exactly meets the fingerprint identification system's speed requirements.

We chose to use the Nios II soft-core processor in our design for the following reasons:

- The Nios II soft-core processor can cut costs through large-scale system integration and FPGA/CPU optimization.
- The Nios II processor is more flexible, provides shorter design cycles, and can prolong the product lifecycle with upgrades.
- Custom instructions and logic can accelerate complex arithmetic operations and logic.
- The Nios II C-to-Hardware Acceleration (C2H) Compiler allows designs to operate more than 40 times faster than software that is not accelerated.

Function Description

We designed the fingerprint identification system based on Altera's Nios II processor and FPGAs. This system can collect real-time fingerprint image signals, extract finger minutiae, and match minutiae in a database to perform fingerprint identification. The whole system design includes fingerprint image collection, fingerprint image preprocessing, minutia extraction, minutia matching, and a database.

Fingerprint Image Signal Collection

The fingerprint collector serves as the fingerprint collection module. The module's fingerprint sensor is Veridicom's third-generation product, the FPS200 sensor (with 256 x 300 array numbers and 500-DPI resolution). The sensor uses Veridicom's ImageSeek function and high-speed image transmission technology to obtain quality images of all fingerprint types. The fingerprint collector design performs the following functions:

- *Capture stage*—Each column of the capacitor array is connected to two sampling/hold circuits to capture one row of fingerprint images. The capture procedure has two stages: capacitors in the chosen row are charged to U_{DD} and store the charging voltage in the first sampling/hold circuit; next, the capacitors discharge to the current source at a speed proportional to the discharged current. After a discharge period, the second sampling/hold circuit stores the capacitors' final discharge voltage. By measuring the difference (ΔU) between the charging and discharging voltages, we obtain the capacity (C) of each capacitor unit.

- *Analog-to-digital (A/D) conversion*—The analog signals representing the unit capacities of the row go through A/D conversion to generate the digital fingerprint image information for the row. The system puts the information in a register and captures the fingerprint images of the next row.

- *Fingerprint signal transmission*—FPS200 contains three bus interface circuits: USB, serial peripheral interface (SPI), and microcontroller (MCU) interfaces. This design uses the SPI interface to transmit the digital information in the registers to the Development and Education (DE2) board's SDRAM.

Fingerprint Image Preprocessing

During fingerprint image preprocessing, the fingerprint image is enhanced. Accurate fingerprint identification relies on the identification of the fingerprint ridge texture and minutiae. However, due to skin condition, collection conditions, devices, the working and living environment of the fingerprinted person, etc., the raw fingerprint images collected by the fingerprint sensor usually contain noise and degrade dramatically. Therefore, the raw fingerprint images must be preprocessed after being collected. Fingerprint image preprocessing procedures include image normalization, orientation and frequency extraction, filtration, binarization, ridge thinning, etc.

Normalization

Image normalization reduces the diversification degree of the grayscale along the ridges and valleys without changing the raw image's structure or texture information. This process gives the image preset means and variances, and facilitates pattern capture and fingerprint frequency. Nevertheless, normalization can also enhance some hash in the image background. Equation 2.1 is for normalization and equation 2.2 is an improved version based on equation 2.1 for the convenience of hardware implementation.

$$G(i,j) = \begin{cases} M_0 + \sqrt{\dfrac{VAR_0(I(i,j) - M)^2}{VAR}} & I(i,j) > M \\ M_0 - \sqrt{\dfrac{VAR_0(I(i,j) - M)^2}{VAR}} & \text{Others} \end{cases} \quad (2.1)$$

$$G(i,j) = \begin{cases} M_0 + \lambda \times VAR \times |I(i,j) - M| & I(i,j) > M \\ M_0 - \lambda \times VAR \times |I(i,j) - M| & \text{Others} \end{cases} \quad (2.2)$$

Fingerprint Orientation Extraction

The basic concept for pattern extraction is to:

- Calculate certain statistics (like grayscale difference and gradients) of each point (or block) in all orientations of the raw fingerprint grayscale image.

- Decide the orientation of the point (block) according to the difference of these statistics in all orientations to obtain the fingerprint pattern.

The algorithm process is as follows:

1. Suppose f(i,j) is the grayscale value of fingerprint pixel point (i,j).

2. Divide the image into W x W-sized sub-blocks without overlapping. It is better for W to contain the size of a ridge and a valley. Here we use 10 for W.

3. Use the Sobel operator to compute the gradients of image's pixel points (i,j) in X and Y orientations, $G_X(u,v)$ and $G_Y(u,v)$. Figure 1 shows the Sobel operator template coefficient.

Figure 1. Sobel Operator Template Coefficient

X-Axis Orientation

1	2	1
0	0	0
-1	-2	-1

Y-Axis Orientation

1	0	-1
2	0	-2
1	0	-1

4. Calculate the partial orientation Θ(I,j) of the W x W sub-block focus on (I,j).

5. Smooth the resulting fingerprint pattern.

Fingerprint Frequency Extraction

In partial non-singular areas of the fingerprint image, the changing pixel point values of the ridges and valleys constitute an approximate two-dimensional (2-D) sine wave along the orientation vertical to the ridges (i.e., the gradient orientation of the ridges). The fingerprint frequency algorithm procedures are as follows:

1. Divide the normalized fingerprint image into W x W-sized sub-blocks without overlapping. Here we use 16 for W.

2. For each image sub-block focusing on (i,j), use the fingerprint orientation Θ(i,j) of (i,j) for the minor axis and draw a rectangular orientation window of L x W (32 x 16).

3. For each rectangular orientation window, calculate the grayscale discrete signals X[0], X[0],…X [L - 1] vertical to Θ(i,j) (i.e., the gradient orientation of the fingerprint) according to the following method: because X[k] constitutes an approximate 2-D sine wave, calculate the mean distance of the sine crest L_0 to obtain sine wave frequency $f = 1/L_0$.

Generally the frequency scope of 500-DPI fingerprint images is [1/25, 1/3].

Valid Mask Extraction

Valid masks are an important part of fingerprint image preprocessing. Valid mask extraction distinguishes the foreground of the finger image from its background. If we do not extract the valid image masks, we will waste time processing background areas and obtain many false minutiae. Therefore, it is necessary to extract valid masks of the fingerprint to eliminate the background and reduce false minutiae points. A valid mask in the fingerprint image sub-block must satisfy the following conditions:

1. The difference between the maximum crest (G_{MAX}) and minimum trough (G_{MIN}) in the 2-D sine wave along the fingerprint gradient orientation of the sub-block's central point should be more than the threshold (*T*).

2. The frequency of the sub-block's 2-D sine wave should be within a certain scope. Here we use [1/25, 1/3].

3. The amplitude variance of the sub-block's 2-D sine wave should be more than the threshold (*D*).

4. In the sub-blocks, the fingerprint grayscale along the fingerprint orientation and along the orientation vertical to the fingerprint orientation should have distinct differences.

Gabor Directional Filtration

To greatly improve the fingerprint image quality, our design uses an orientation filter enhancement method. The Gabor filter formula is:

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_x}\exp\left(-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_x^2}\right)\right)\exp(-2\pi ifx)$$

Removing the odd component simplifies the formula. Turn to the fingerprint orientation. The design takes each pixel point in the fingerprint image as the central point and W x W as the window to obtain the Gabor enhancement coefficient of each pixel point in the block along the fingerprint's texture.

Binarization and Noise Reduction

The binarization process generates a monochrome fingerprint image from a grayscale image. To implement binarization, we use an adaptive, local threshold scheme, i.e., we adjust the threshold by the global grayscale of a block in the image. The procedures is:

1. Divide the fingerprint image in the valid mask into W x W blocks (we use 8 for W in this design).
2. Calculate the grayscale mean of each sub-block.
3. Take the grayscale mean as the threshold to implement binarization in the sub-block.
4. The binarization process may introduce noise; therefore, after binarization the fingerprint image should be filtered and have noise removed to delete the holes, notches, and other salient features caused by binarization.

Thinning

Ridge thinning transforms distinct but diversely sized binary fingerprint images into a single-pixel central point and thread image. Our design uses the Hilditch algorithm for this operaton. With this method, the whole image must be scanned several times. During each scan, pixel points satisfying given conditions are marked. After scanning, the marked pixel points are deleted and the next scan begins. When no pixel points are marked during scanning, the ridge thinning process finishes.

This functional algorithm has been emulated successfully using the C language. We used different schemes to implement modules according to their operation time. If we directly implemented the normalization, frequency and orientation extraction, and orientation filtration of the fingerprint image in the Nios II processor, the operation time would be unacceptable. Therefore, our system accelerates the hardware logic and instructions with the C2H Compiler. Because binarization and image thinning consume less time, we can implement them directly in the Nios II processor. Additionally, the whole algorithm involves many multiplication, evolution, rotation, and floating-point operations, which greatly slow the processing speed. Therefore, our design uses custom instructions in the Nios II processor to add the custom functions directly into the arithmetic logic units of the Nios II CPU, implementing these complex, time-consuming operations in hardware.

Fingerprint Minutia Extraction

Minutiae extraction involves preprocessing the image to obtain a quality image, and then finding and specifying the minutiae. After a raw fingerprint image goes through orientation filtration, binarization, and thinning, it becomes a thinned image. We then determine the endpoint and bifurcation according to the crossing of each point on the thinned image and extract the useful information of the two minutia points, such as coordinate position, type, and orientation. Fingerprint minutiae fall into many types. From the perspective of probability, 2-bifurcation and ending are the most common.

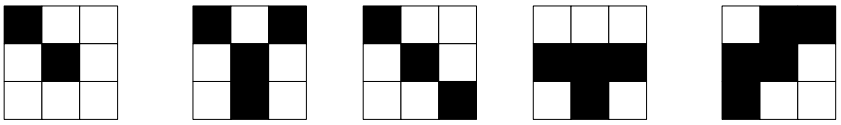
Minutia Point Extraction

For thinned images, the pixel point grayscale value can only be 0 or 1. We set 0 as the background point grayscale and 1 as the foreground point grayscale of the ridge. The crossing number C_N of eight fields of any point P on the thinned image (see Figure 2) is defined as:

$$C_N = \frac{1}{2} \sum_{k=1}^{8} |P_{k+1} - P_k| \text{, here } P_9 = P_1.$$

If $C_N(P) = 1$, point P is an endpoint. If $C_N(P) = 3$, point P is bifurcation. Otherwise, point P is a consecutive point or an isolated point and cannot be calculated into the minutia set.

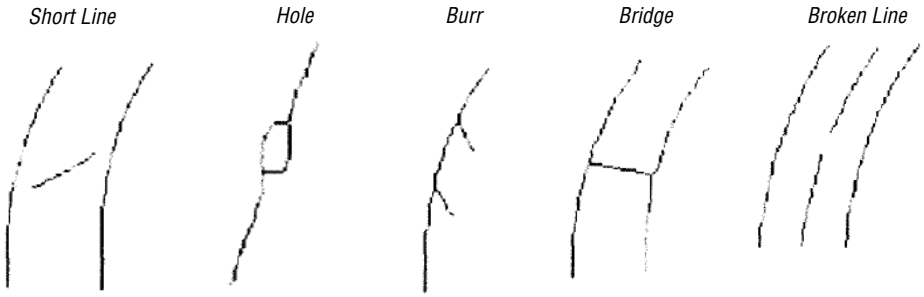
Figure 2. Minutiae Point Block Diagram



False Minutiae Removal

The thinned fingerprint images that are not refined include the false minutiae shown in Figure 3.

Figure 3. Basic False Minutia Structure Types



We remove these different false minutia points using different algorithms:

1. Remove the edge effect of the image.
2. Delete the false minutia point caused by the obscure part of the image.
3. Delete the false minutia point caused by broken ridges.
4. Delete the false minutiae point caused by holes.
5. Delete the false minutiae point caused by burrs.
6. Delete the false minutiae point caused by bridges.

These algorithms have been emulated successfully using the C language. Because the algorithms are comparatively easy, do not have time-consuming operations, and the processing speed meets real-time requirements, we implemented them directly in the Nios II processor.

Minutia Matching

Fingerprint minutia matching is the core of the fingerprint identification algorithm and is an important research subject. The accuracy and speed of minutia matching have a huge influence on the whole algorithm. Minutia matching matches the extracted minutia information set of two given fingerprint images to determine whether they are identical. In our design, we:

- Use a typical point pattern matching algorithm based on a minutia point coordinate mode.
- Utilize the triangle structure composed of three neighboring minutia points to locate a datum mark and to find conversion parameters.
- Conduct matching in the polar coordinate system after coordinate translation.
- Introduce multiple judgement conditions and a variable limit-box matching algorithm to improve the identification rate.

We implemented the algorithm as follows:

1. For any minutia point Q_i in input point set Q and any minutia point P_j in template point set P, search the two minutia points (Q_{i1},Q_{i2}) and (P_{j1},P_{j2}) that are nearest to Q_i and P_j, respectively, in Q and P. Thus, points (P_j,P_{j1},P_{j2}) and (Q_i,Q_{i1},Q_{i2}) form two triangles $\Delta P_j,P_{j1},P_{j2}$ and $\Delta Q_i,Q_{i1},Q_{i2}$.

2. Determine whether the two triangles are identical and have the same rotation orientation. If they do, take the point as a polar point to create polar coordinates, and then match the minutia points using a variable, limit-box method.

3. We believe there is a match only if the two conditions are met:

$$(N_m \geq T_m),\left(\eta = \frac{2 \times N_m}{N + M} \times 100 \quad \geq \quad _s\right)$$

   where N_m is the number of matching point pairs of the two minutia sets, T_m and T_s and are the thresholds under the two conditions, and N and M are the minutia numbers of the two minutia point sets.

These algorithms have also been verified successfully in the C language. For our simple fingerprint authentication service system, we implemented these functions in the Nios II processor.

Database

Due to the limitations of time and conditions, our system implements a simple fingerprint authentication service system. For this system, we loaded the fingerprint database into the flash memory on the DE2 development board. The system stores and accesses data using a linked list.

Liquid Crystal Display (LCD)

The system's LCD module is a complete hardware-control module designed with a hardware description language (HDL). When the CPU is not accessing the SRAM, the image information in the SRAM is displayed on the LCD. Fingerprints are displayed on the left side of the LCD and the names of the participating team and team members, as well as the operation button prompts, are displayed on the right.

The CPU does not need to control the LCD module. Using 1% of the FPGA logic resources, the function can satisfy the real-time system requirements and demonstrate the advantages of combining hardware and software using system-on-a-programmable-chip (SOPC) technology.

Performance Parameters

This section describes the design's performance parameters.

Actual System Performance Parameters

The actual system has the following parameters:

- Power supply: DC, 9 V
- Operating environment
  - Operating temperature: -5° to 45°
  - Relative humidity: 8% to 95%
- Input signal
  - Method: Input by our fingerprint collector with the FPS200 fingerprint sensor
  - Sensor array number: 256 x 300
  - Sensor resolution: 500 DPI
  - Signal transmission mode: SPI
- Display
  - LCD: 16 x 2
  - Display: 320 x 240
- Fingerprint recognition rate
  - Number of tested users: 5
  - Test times: 20 times/person
  - Recognition rate: 99%
  - False rejection rate: 1%
- Consumed time
  - Fingerprint collection time: 2.3207 seconds
  - Fingerprint image preprocessing time: 24.1294 seconds
  - Normalization time: 0.8326 seconds
  - Orientation extraction time: 1.3586 seconds

- Frequency extraction time: 2.8262 seconds
- Gabor filtration time: 12.5535 seconds
- Binarization time: 0.4280 seconds
- Thinning time: 6.1305 seconds
- Minutia extraction time: 1.4634 seconds
- Minutia matching time: subject to the number of fingerprints in the database

■ Fingerprint database storage size—The flash storage device on the DE2 board is only 4 Mbytes. Of the 4, we use 1 Mbyte to store other information, so the fingerprint database is only 3 Mbytes and can store a maximum of 48 fingerprints. If we expand the non-volatile memory, e.g., with another flash device, we can theoretically expand the fingerprint database to any size.

Nios II Processor Functions in the Design

As the core of this system, the Nios II soft-core processor has the following functions:

■ The Nios II processor contains two types of peripherals: standard and custom. It is easy to add standard peripherals. Additionally, Altera and some third-party corporations are providing more and more intellectual property (IP) cores for standard peripherals. Custom peripherals greatly accelerate system operation and save CPU resources. The design's fingerprint image preprocessing normalization module improves the processing speed over 20 times.

■ Custom instructions are a unique feature of the Nios II processor. They can dramatically improve the system performance. The system's Gabor filter takes full advantage of the custom instruction feature and improves the processing speed over 30 times.

■ The C2H hardware acceleration is efficient and easy for designers to use to accelerate system development. Additionally, it highly improves system performance. In this design, we only apply C2H hardware acceleration to some orientation extraction algorithms, which improves the processing speed more than 6 times.

Hardware Resource Usage and Performance Improvement

Table 1 shows the hardware resource usage and Table 2 shows the performance improvement.

Table 1. Resources Used

Resource	Logic Elements (LEs)		Memory Bits	
	Gate	%	Bits	%
Normalization (custom module)	1,394	5	24	<1
Gabor filter (custom instruction)	1,832	6	4,096	1
LCD	470	1	0	0
Orientation extraction (C2H)	5,314	16	512	<1
Cos, exp, and fpoint (custom instruction)	6,937	21	16,386	4
Total	15,947	49	21,018	5

Table 2. Performance Improvement

Operation	Software Operation Time (s)	Hardware Improvement	Operation Time after Improvement (s)
Normalization	16.9491	Hardware module	0.8326
Gabor filter	458.0213	Custom instruction	12.5535
Orientation extraction	9.490	C2H	1.3586
Frequency extraction	31.7303	Floating point, multiplication shift, and cos table look-up	2.8262

Design Architecture

This design considers two schemes. The first scheme is a simple fingerprint authentication service system (see Figure 4) that integrates a database server and web server into the embedded fingerprint identification system to provide a simple web-based network service. The second scheme is a complex fingerprint authentication service system (see Figure 5) that targets mass users who have strict system requirements. We began with the simple fingerprint authentication service system, and if we had enough time in the final stage, we would have expanded the system into a complex fingerprint authentication service system. However, due to time limitations and other constraints, we only implemented the simple fingerprint authentication service system.

Figure 4. Simple Fingerprint Authentication Service System

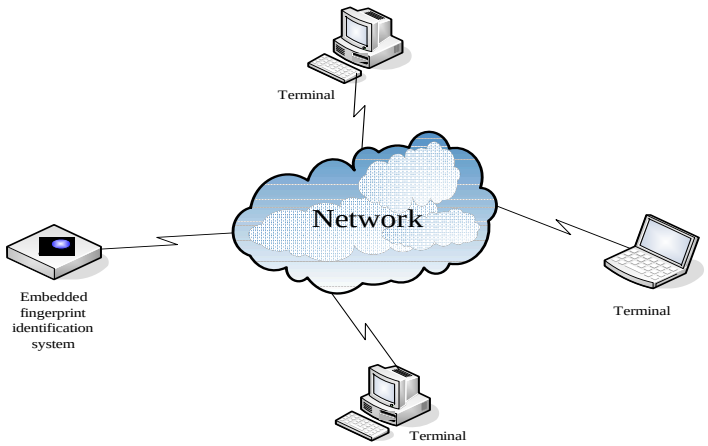
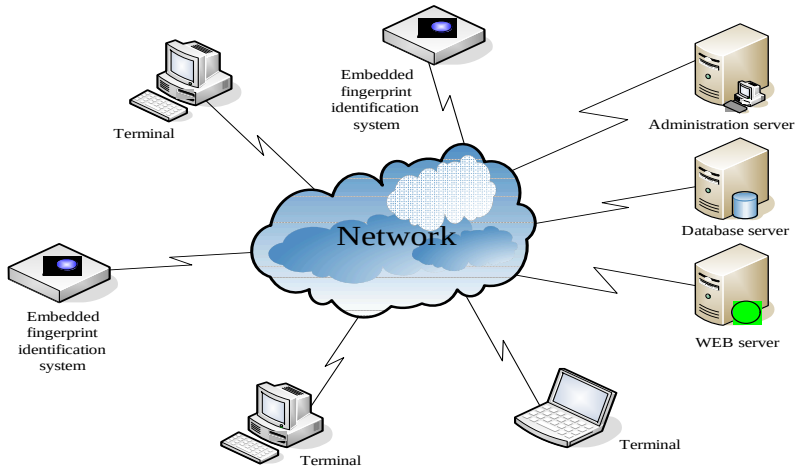


Figure 5. Complex Fingerprint Authentication Service System



Figures 6 through 8 show various block diagrams of the system.

Figure 6. Hardware System Block Diagram

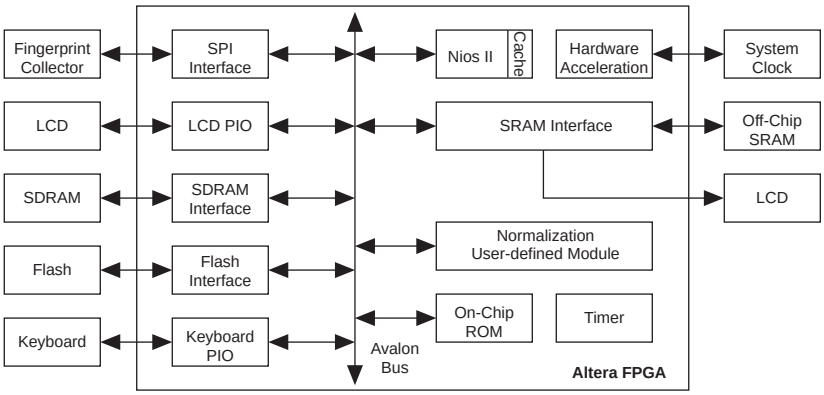


Figure 7. High-Level Block Diagram

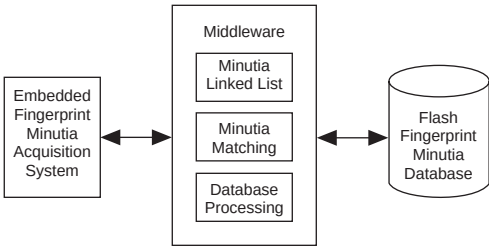
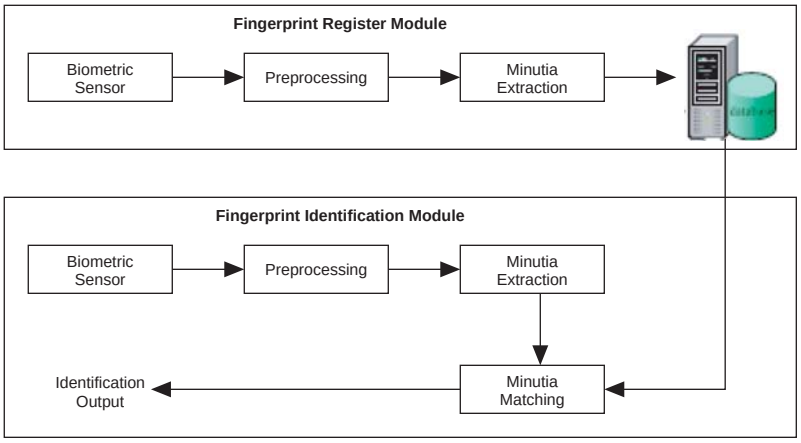


Figure 8. Structural Block Diagram



Figures 9 and 10 show the system flow charts.

Figure 9. Fingerprint Identification System Flow Chart

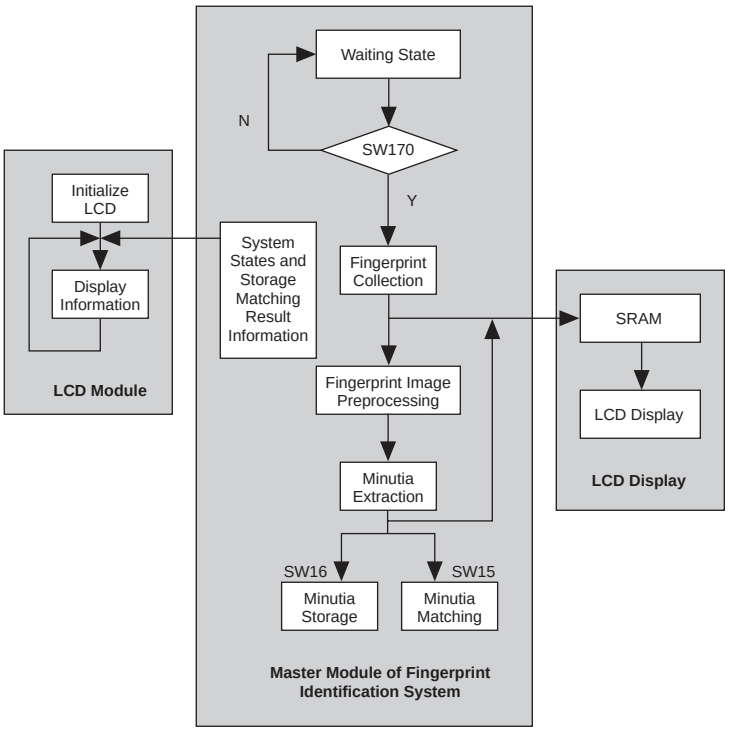
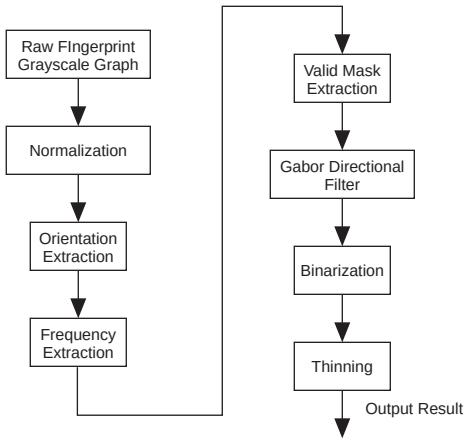


Figure 10. Fingerprint Minutia Preprocessing Algorithm Flow Chart



Figures 11 and 12 show the system in SOPC Builder.

Figure 11. SOPC Builder

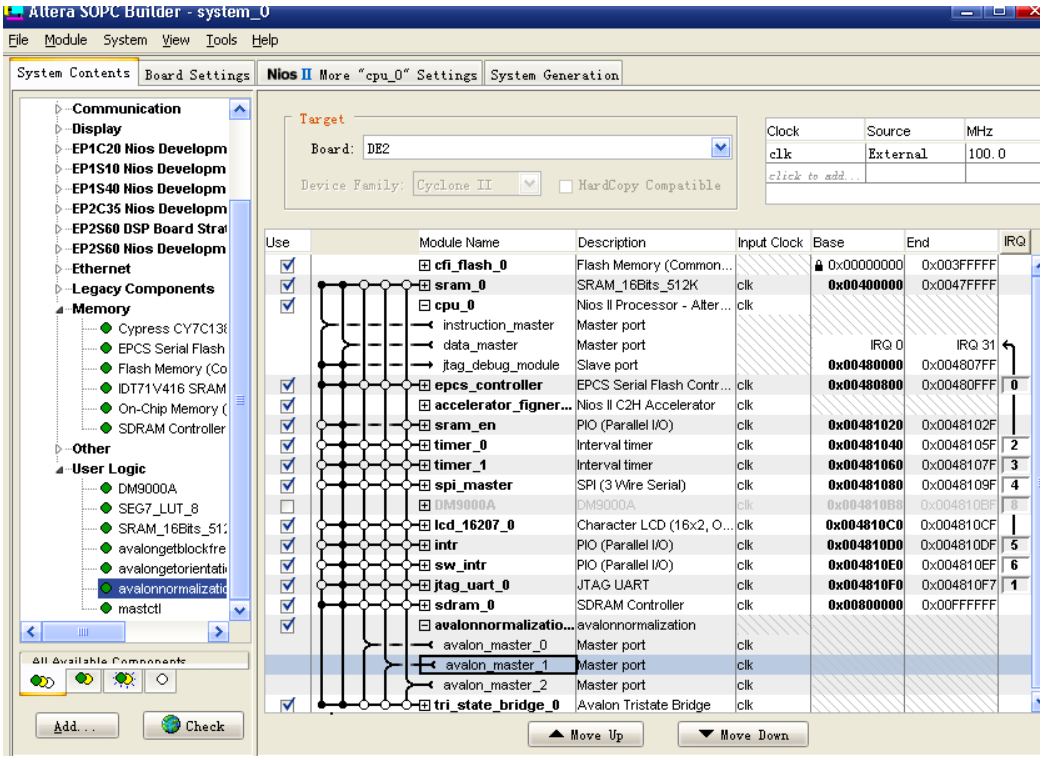


Figure 12. Custom Instructions

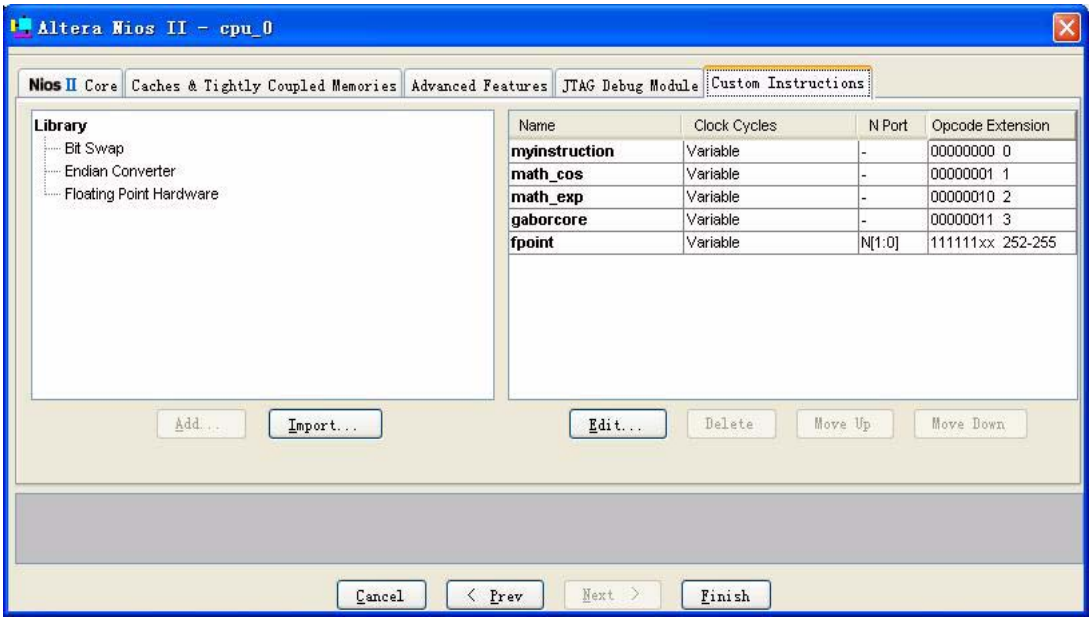
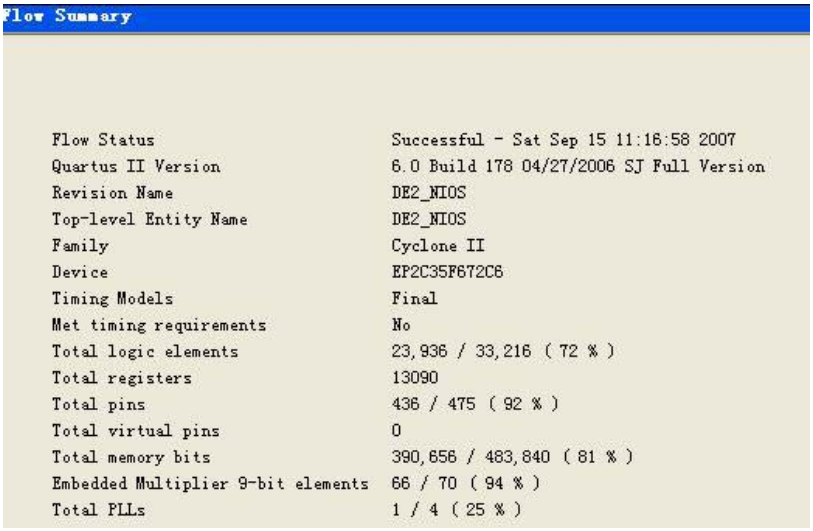


Figure 13 shows the hardware resource usage.

Figure 13. Hardware Resource Usage



Design Methodology

The system's hardware is based on the DE2 development board. The hardware design fully utilizes the board's existing interfaces, e.g., we used the LCD to facilitate the user interface. By combining the LCD and fingerprint collector, the system displays information promptly. The system software development is based on the Nios II embedded soft-core processor and leverages custom instructions, modules, and the C2H Compiler, which accelerates some software in hardware.

System Hardware Design

The following sections describe the hardware design.

Fingerprint Image Collection

Each column of the capacitor array is connected with two sampling/hold circuits to capture only one row of fingerprint images one time. The capture procedures include two stages:

- Capacitors in the chosen row are charged to U_{DD} and store the charge voltage in the first sampling/hold circuit.
- The capacitors discharge to the current source at a speed proportional to the discharged current.

After a discharge period, the second sampling/hold circuit stores the final capacitor discharge voltage. By measuring the difference (ΔU) between the charging and discharging voltages, we can obtain the capacity (C) of each capacitor. The analog signals representing the unit capacities of the row go through A/D conversion to generate the digital fingerprint image information for the row. The system places the information in a register and captures the next row's fingerprint images.

We added an SPI bus module and button interrupt module with SOPC Builder. The SPI module has a 12-MHz clock. A fingerprint image is 256 x 300, therefore, a fingerprint image can be transmitted within 0.1 second. The button interrupt has a double-edge trigger and the button is a slide switch. On power-up, the initialized fingerprint collects the module register values. When the button is interrupted, the fingerprint collection chip transforms the collected signals into digital information and stores them in the register. The CPU sends the digital image data in the fingerprint collector register to the specified memory space in the DE2 board's SDRAM via the SPI bus. After all fingerprint image data is collected, it is normalized and assigned in SRAM. The initial SRAM address is 0 and the data occupies the 0x12c00 address space. In the final stage, only the normalized fingerprint images are processed.

Custom Instructions

In this design, custom instructions target complex mathematical operations and some time-consuming algorithms during fingerprint image preprocessing. Because the embedded Nios II system consumes a lot of time performing complex mathematical operations, the system instead uses hardware to implement the operations and uses custom instructions in the CPU. Figure 12 on page 260 shows three custom instructions we added during CPU customization that accelerate the math_cos, floating operation fPoint, and exponential function math_exp trigonometric functions.

The Gabor directional filter consumes the most time in this design. Because it is very complex, there would not be enough FPGA logic resources if we only used custom modules. Our design implements some of the time-consuming algorithms through custom instructions. The main formula of Gabor directional filtration is:

$$G(x, y) = \sum_{u=-W/2}^{W/2} \sum_{v=-W/2}^{W/2} g'(u, v) N(x - u, y - v)$$

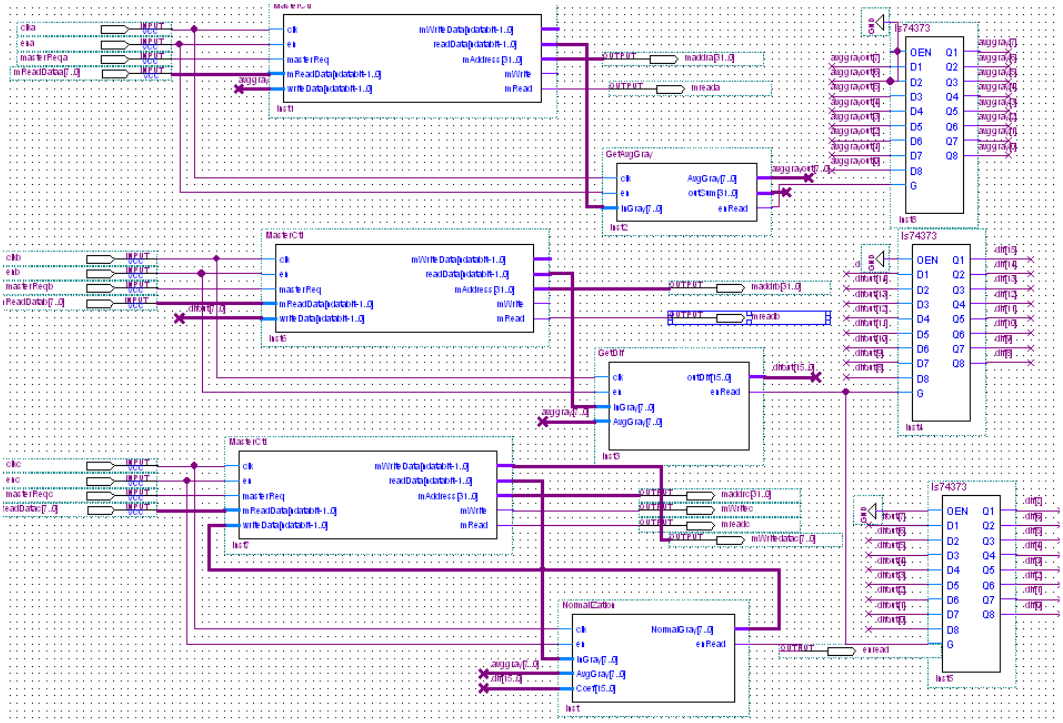
$$g'(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x'^2 + y'^2}{2\sigma^2}\right) \cos(2\pi f x')$$

N(x - u, y - v) is the grayscale value of the normalized fingerprint image. We use custom instructions to implement g'(x, y) x N(x - u, y - v), and implemented the accumulation summarization in software. After the CPU is generated, these functions generate the corresponding function interfaces. We invoke the programs in the same way as common function sub-programs.

Custom Modules

Our design contains many image processing algorithms, which require a lot of processing time. To shorten the processing time as much as possible, we convert normalization (the first part of image preprocessing) into hardware, which does not affect software processing. Figure 14 shows the normalization preprocessing hardware schematic.

Figure 14. Hardware Normalization Schematic



Because normalization compares the images and then normalizes them, we read the image using different methods, that is, we obtain the information using three main control modules and the Avalon® bus interface, and then generate the normalized image. Figure 15 shows the image processing result.

Figure 15. Image Preprocessing Before and After Normalization

C2H Compiler

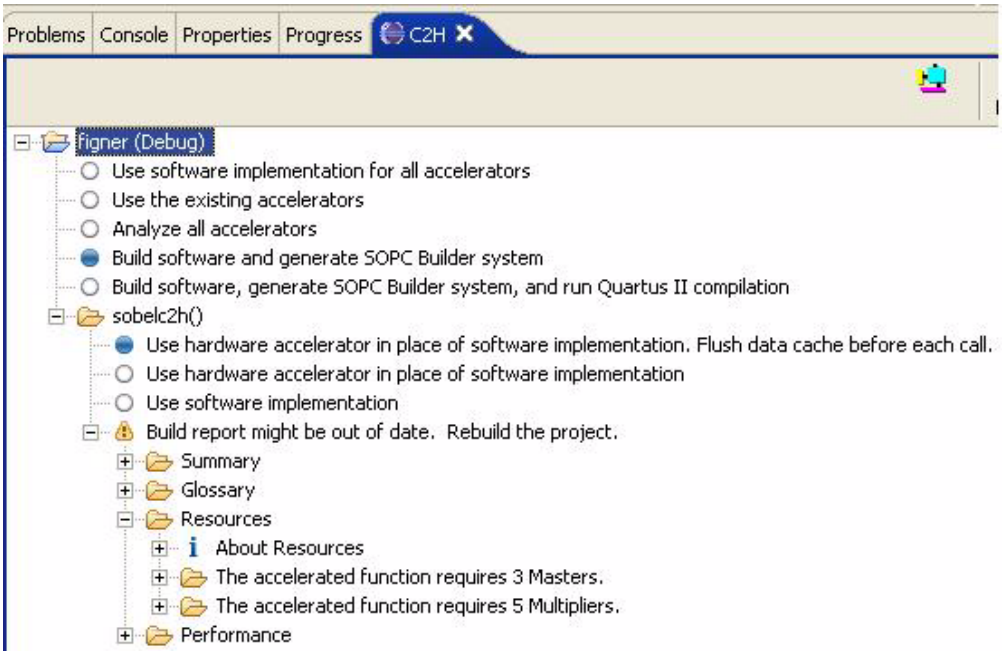
In this design, we used the C2H Compiler for integer algorithms with many loops and single operations. Because most of the algorithms use floating-point numbers, we translate them into integers before using the C2H Compiler to convert them into hardware. We used the C2H Compiler to optimize the Sobel operator for orientation extraction, which has 4 loops and integer accumulation algorithms. Figure 16 shows the Sobel operator C2H function.

Figure 16. Sobel Operator C2H Function

```
void sobelc2h(int iWindow,Img *ImgSrc,int *pDst,int *pModelCente
{
  int iRow,iCol,i,j;
  int iWidth,iHeight;//iSize;
  UINT8 *pSrcAddr,*pSrc;
  int iResult;
  UINT8 uGray;
  iWidth = ImgSrc->size.iWidth;
  iHeight = ImgSrc->size.iHeight;
  pSrc = ImgSrc->pImg;

  //iSize = iWidth*iHeight;
  //针对每隔像素均进行求取沿x或y方向上的梯度值
  for (iRow = iWindow;iRow < iHeight - iWindow;iRow++)
  {
    for (iCol = iWindow;iCol < iWidth - iWindow;iCol++)
    {
      pSrcAddr = pSrc + iRow * iWidth + iCol;
      //Sobel Core
      iResult = 0;
      for (i = -iWindow;i<=iWindow;i++)
      {
        for (j=-iWindow;j<=iWindow;j++)
        {
          uGray = *(pSrcAddr + i*iWidth + j);
          iResult += *(pModelCenter + i*iWinWidth + j) * uGray;
        }
      }

      *(pDst + iRow * iWidth + iCol) = iResult;
    }
  }
```

Figure 17 shows the C2H Compiler settings in SOPC Builder.

Figure 17. C2H Compiler Settings



After using C2H hardware acceleration, although the algorithm uses 5,314 logic resources, 512 memory bits, one M4K block, 22 DSP elements. and 11 18 x 18 DSP blocks (including 3 masters and 5 multipliers), the Sobel operator's operation speed increases 35 times and the speed of the whole orientation extraction process increases 6 times.

C2H hardware acceleration satisfies the system's speed requirements and saves the time in hardware module design by occupying some hardware resources. A system can use up to five C2H hardware accelerations, therefore if there are enough hardware resources, multiple C2H accelerations can be used.

LCD

The LCD module is a separate hardware module and is only connected to the SRAM. When the CPU is not accessing the SRAM, the LCD module displays fingerprint image data in the SRAM. During the fingerprint collection stage, the image is dumped to the SRAM by memory; then, the CPU releases SRAM control and the LCD module accesses SRAM and displays the image. When fingerprint preprocessing and minutia point extraction are complete, the CPU controls SRAM again, covering the original fingerprint image with the processed fingerprint minutia image and releasing the SRAM control to display the fingerprint minutia image.

In our system, when the CPU writes the fingerprint information data, it conflicts with the LCD reading the SRAM. To resolve this problem, we added an enabled programmable I/O (PIO) in SOPC Builder that allows the LCD to read the SRAM when the CPU is not accessing it. This design ensures that the CPU can access the SRAM at any time and the LCD can provide a stable display. Having the LCD read the SRAM relieves the CPU through the designed hardware module without CPU interference. When users implement mass and single-domain peripheral interaction functions during SOPC system design, they should use custom master peripherals because they can operate the storage area directly while interacting with off-chip devices; the CPU does not need to use a special process for storage area. These modules can be added as necessary if there are enough logic resources. This flexibility is one advantage of SOPC system design over system-on-chip (SOC) design.

System Software Design

The most important part of the software design is the control function, image preprocessing, minutia point extraction, and minutia matching.

Control Function

The control function mainly controls interrupts. It informs the system of specific operating instructions, including collecting the fingerprint, matching it, and storing it using button interrupts.

- *Collect fingerprint*—SPI communicates with the CPU to notify it when fingerprint collection ends, at which point, service function `fps_Irq_In()` is interrupted. When the collection finishes, the CPU automatically processes the fingerprints, extracts the minutia points using `FingerImgEnhance(pImgSrc,pImgSrc)`, and stores collected fingerprint information into the current finger information linked list `CurrentMinutia` (custom linked list structure) in the SDRAM.

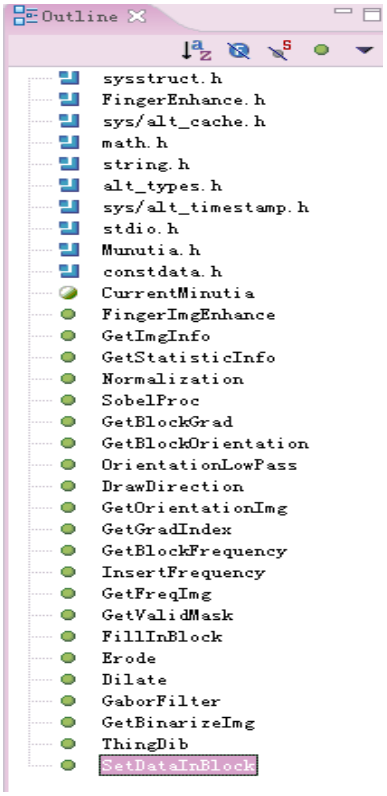
- *Match fingerprint*—If the system determines that there is a match, it performs `matchFigMinutia(&CurrentMinutia)`.

- *Store fingerprint*—To store the information, the system uses the `wOneFigMinutiaToFlash()` function. The current fingerprint information is stored in the flash device's fingerprint linked list.

Image Preprocessing

Image preprocessing enhances the fingerprint image, and includes the functions shown in Figure 18.

Figure 18. Image Preprocessing Functions



The functions are described as follows:

- *Normalization*—The system performs normalization in hardware. After the image is collected, the hardware normalizes the image. The resulting image is put into the 0 address of the SRAM. The software can read the image directly without interface problems.

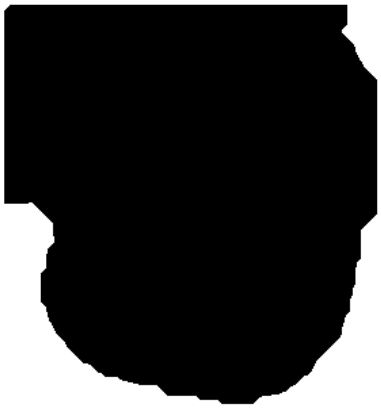
- *Orientation extraction*—The `GetBlockOrientation` function calculates the fingerprint image block orientation and invokes `Soborcore`. C2H hardware acceleration implements hardware instructions to calculate the gradients of each pixel point in the x and y orientations. Then `GetGradIndex()` is invoked, and orientation extraction is performed. Figure 19 shows the pattern.

Figure 19. Orientation Extraction Pattern

- *Frequency extraction*—The `GetBlockFrequency` function calculates the fingerprint image block frequency. The changes of fingerprint ridges and valleys constitute a 2-D sine wave. To obtain the frequency, the system invokes `GetImgInfo(ImgSrc)` as described previously. It gets the fingerprint information and frequency interpolating function `InsertFrequency(pOut,pOut,iWindow,ImgSrc.size)`, and invokes the orientation filter, `OrientationLowPass(pOut,pOut,iWindow)`, to smooth the image.

- *Valid mask extraction*—A full mask fingerprint image cannot usually be generated during collection. This function, `GetValidMask`, generates the foreground mask (the fingerprint image, which is the part of the finger touching the sensor) and background mask (hash). Valid mask extraction distinguishes the foreground from the background. It invokes `GetGradIndex(*(pOrient + iOffSetAddr))` to get the orientation. The `if(abs(xDelta - yDelta) > QNios)` function sets the threshold to compare image contrast. A valid mask dramatically changes the image as shown in Figure 20.

Figure 20. Valid Mask



- *Gabor filter*—To improve the fingerprint image quality, we use an orientation filter enhancement method. In noise removal and maintaining the fingerprint ridge structure, like a bandpass filter feature, we optimized an important operation with custom instructions and implement the formula

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_x}\exp\left(-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_x^2}\right)\right)\exp(-2\pi ifx)$$

  with software. Figure 21 shows the effect.

Figure 21. Gabor Filter Effect

- *Binarization and noise removal*—The `GetBinarizeImg()` function performs binarization and removes noise. During binarization, the Gabor-filtered grayscale image of a specific fingerprint generates another monochrome fingerprint image. The adopted adaptive local threshold scheme adjusts the threshold by the global grayscale of a block in the image. Implementing binarization in software is comparatively easy, but the loops consume a lot of processing time. Figure 22 shows the result.

Figure 22. Binarization and Noise Removal Result

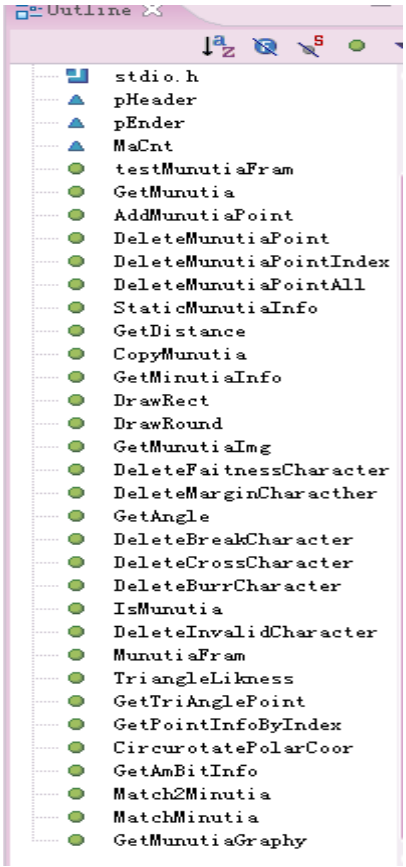
- *Thinning*—The `ThingDib(Img ImgDst, Img ImgSrc)` function extracts minutia points by thinning the binarization image to a single-pixel fringe central point and thread image. See Figure 23.

Figure 23. Thinning Result

Minutia Point Extraction

After a raw fingerprint image goes through orientation filtering, binarization, and thinning, it becomes a thinned image. We then determine the endpoints and bifurcation according to the crossing of each point on the thinned image, and extract the useful information of the two minutia points such as coordinate position, type, and orientation. Figure 24 shows the minutia point extraction functions.

Figure 24. Minutia Point Extraction Functions



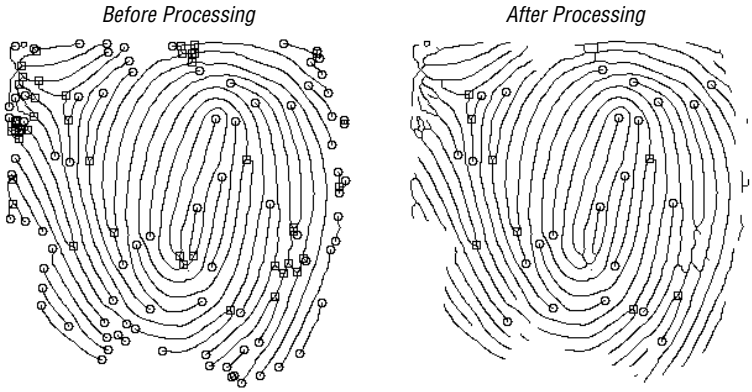
The main function, `GetMunutia(UINT8 *pDst,float *pOrient,UINT8 *pSrc)`, extracts minutia points by:

- Getting all possible minutia points: `GetMinutiaInfo(&CurrentMinutia,pOrient,ImgSrc)`
- Deleting false minutia points: `DeleteInvalidCharacter(&CurrentMinutia, &pBreakMunutia,&pBurrMunutia,&pCrossMunutia)`, which includes:
  - Deleting the false minutiae caused by image obscurity: `DeleteFaitnessCharacter(&CurrentMinutia,2,3,3,10,size)`
  - Deleting the false minutia points caused by broken lines: `DeleteBreakCharacter(&pBreakMunutia,5,20)`
  - Deleting burrs: `DeleteBurrCharacter(&pBurrMunutia,6,12)`
  - Deleting holes and bridges: `DeleteCrossCharacter(&pCrossMunutia,6,15)`
  - Deleting margin minutia points: `DeleteMarginCharacther(&CurrentMinutia,ImgSrc)`
- Output the image: `GetMunutiaImg(ImgDst,&CurrentMinutia)`

- Operations related to the linked list:
  - Add minutiae to the information linked list: `void AddMunutiaPoint(Minutia *pMinutia, CPoint point, float fOrient, enum MinutiaPointType pointType)`
  - Delete the designated minutia points: `void DeleteMunutiaPoint(Minutia *pMinutia,MinutiaInfo* pPoint)`
  - Delete the minutia points of the designated index: `void DeleteMunutiaPointIndex(Minutia *pMinutia,int index)`
  - Take statistics of Ne and Nb: `void StaticMunutiaInfo(Minutia *pMinutia,CSize size, int iWindow)`
  - Calculate the distance of two minutia points: `int GetDistance(CPoint cpPoint1,CPoint cpPoint2)`
  - Copy the template: `void CopyMunutia(Minutia *pFromMunutia, Minutia *pToMunutia)`
  - Get minutia points: `int GetMinutiaInfo(Minutia* pMunutia, float* pOrient, Img ImgSrc //thinned image)`
  - Draw the rectangle: `void DrawRect(Img ImgSrc, CPoint cpPoint, int iWindow)`, etc.

Figure 25 shows the effect after processing.

Figure 25. Before and After Minutia Point Extraction



The fingerprint minutia data structure includes the minutia point structure and fingerprint minutia set.

Minutia Point Structure

A complete description of minutia points includes the coordinate point, orientation, and type of minutia on the image. We designed the following data structure to describe the minutia point.

Minutia point information in the rectangular coordinate system:

```
typedef strut MunutiaPointInfo_st{
  int x; //horizontal coordinate value
  int y; //Vertical coordinate value
  float index; //orientation
  enum MunutiaPointType{END,CROSS} type; //minutia point type
}MunutiaPointInfo, *pMunutiaPoint;
```

Minutia point information in the polar coordinate system:

```
typedef strut MunutiaPolarInfo_st{
  int iRadius; //polar radius
  float fTheta; //polar angle
  float index; //orientation
  enum MunutiaPointType{END,CROSS} type; //minutia point type
}MunutiaPolarInfo,*pMunutiaPolar;
```

Fingerprint Minutia Set

The minutia set is the set of all minutia points of a fingerprint. Because the design uses a deletion operation to remove false minutia points, we use a two-way linked list design with the following data structure:

```
typedef struct Munutia_st{
  MunutiaPointInfo Info; //information of current minutia point under rectangular coordinate system
  MunutiaPloarInfo polarInfo;//information of current minutia point under polar coordinate system
  struct Munutia_st *pPrev; //direct to previous minutia point information
  struct Munutia_st *pNext; //direct to next minutia point information
}Munutia,*pMunutia;
```

Minutia Matching

In this design, we:

- Use typical point mode matching algorithm based on a minutia point coordinate mode.
- Use a triangle structure composed of three neighboring minutia points to locate a datum point and evaluate the conversion parameter.
- Conduct matching in the polar coordinate system after coordinate translation.
- Introduce multiple judgement conditions and a variable limit box matching algorithm to improve the recognition rate.

The main content is the matching function `matchFigMinutia(Minutia *CurrMa)`. If the current fingerprint matches, the fingerprint minutia set and the matching nodes are returned. Minutia matching functions include:

- Determine whether the triangles are similar: `float TriangleLikness(MinutiaPointInfo* PPointInfo, MinutiaPointInfo* QPointInfo, int iDelta /*= 5*/)`
- Get two coordinate points near to a datum: `mark bool GetTriAnglePoint(char* pSrc, MinutiaPointInfo* pointInfo,CSize size)`
- Rotate polar coordinate: `void CircurotatePolarCoor(float rotateAngle)`
- Get limit box size: `AmBitInfo GetAmBitInfo(int iRadius)`

- Match two minutia points: `int Match2Minutia(MinutiaPointPoolInfo PInfo, MinutiaPointPoolInfo QInfo)`
- Match current minutia set with template minutia set `pQMunutia` and roll back matching node number: `int MatchMinutia(Minutia *pQMunutia)`
- Display minutia point on graph `psr` and the displayed grayscale value is the position of the minutia point in the linked list: `void GetMunutiaGraphy(char* pSrc, CSize size)`

Design Features

Our design has the following main features:

- *Highly integrated SOPC technology and embedded custom modules*—If we implemented the image preprocessing, including image normalization, frequency extraction, orientation extraction, and filtering, directly in the embedded Nios II processor, it would take too much processing time. Therefore, the system uses an FPGA hardware algorithm to implement preprocessing. Binarization and thinning of the image consume less time and are directly implemented in the Nios II processor.
- *Custom instructions*—The fingerprint identification system involves many floating-point, multiplication, evolution, and rotation operations. If we implemented them in the embedded Nios II system, it would consume too many resources. Therefore, this design uses custom instructions to add the custom functions directly into the arithmetic logic units of the Nios II CPU, allowing these complex, time-consuming operations to operate in hardware. As a result, the data processing time increases dramatically and the design's real-time requirements are met. Our system uses five custom logic instructions.
- *C2H Compiler*—This system uses hardware acceleration with the C2H Compiler to accelerate software sub-programs that need high performance for image preprocessing. This acceleration dramatically improves the system's general performance. This system uses one C2H module.
- *Two fingerprint identification system architecture schemes*—According to practical production and actual requirements, we developed two feasible fingerprint identification system architecture schemes: complex and simple. Based on database technology, FPGA technology, and embedded technology, we propose a valid implementation. This system implements the simple fingerprint authentication service system.
- *Improved minutia matching algorithm*—In this design, we used a triangle structure composed of three neighboring minutia points to locate the datum point and evaluate conversion parameters, conducted polar coordinate system matching after coordinate translation, and introduced multiple judgement conditions and a variable limit box matching algorithm to improve the recognition rate.
- *Floating-point shift*—Because many multiplication algorithms in floating-point operations consume a lot of processing time, we shift floating-point numbers to integers, thereby improving the system's algorithm speed three to four times.
- *Convenient product design and development*—The FPGA-based Nios II soft-core processor and its SOPC solution have tailor-made, reconfigurable advantages over hard-core processors. It is easy to develop the software and hardware products with the Nios II processor. Additionally, the Nios II processor facilitates collaborative system design and permits system maintenance and upgrades. By using an FPGA with abundant resources, we can implement a multi-CPU system and simplify the hardware circuit using a design control module, satisfying the system's functional requirements and cutting the design cost. Moreover, we can upgrade the system hardware by loading an updated soft-core system without affecting the peripheral circuit.

Conclusion

With this design contest, we learned about the power and convenience of the Nios II processor, while improving our engineering ability. The advantages of the Nios II processor are as follows:

- Few resources are used. One Nios II CPU only occupies thousands of LEs.
- Constructing an SOPC system is flexible and convenient. We can create the desired system quickly and customize the peripheral modules required by the system, significantly reducing the difficulty of the hardware design and development period.
- FPGAs are flexibly designed for DSP. Users can design completely arithmetic circuits according to their needs. An FPGA can perform operation functions more efficiently at lower frequencies than CPU co-processors. For example, because this design uses a 100-MHz system clock, some algorithm operations are much faster than a 2-GHz Pentium 4 processor.
- The Nios II C2H Compiler and custom instructions help users implement algorithms and logic controls. With the C2H Compiler, users can input pointer parameters. However, floating-point numbers cannot be processed internally and users cannot modify the acceleration logic circuits generated by the compiler. Therefore, users have to first quantify the floating-point numbers and then transform them into floats after output during digital signal processing. This method is not as efficient as using the CPU's internal floating-point operation acceleration instructions. Custom instructions allow users to design hardware logic, but their input and output parameters are not as flexible as those of the C2H Compiler because common users seldom add single or double operand instructions but often process a lot of data. When we experienced this kind of problem, we had to replace the C2H Compiler and custom instruction implementation with a custom module and interrupt (or query), which destroys the system software flow to some extent. If the C2H Compiler and custom instruction advantages could be combined to create a universal function interface and allow an internal hardware implementation of a flexible design, the Nios II development platform would benefit users more.
- The FPGA-based flexible embedded design supports interfaces per the user's specific requirements. It can represent users' design ideas better than a hard-core processor design. However, FPGAs require users to plan and design the SOPC system comprehensively; in particular, the final adaptation of the hardware netlist in the Quartus® II software consumes too much time.

The following areas of the design could be improved:

- Due to the limitations of the hardware system and the small-sized volatile and non-volatile memory, the fingerprint database is small. If we had enough time to design a hardware system according to the practical requirements, abandon unnecessary hardware such as the Ethernet chip and video collection chip, and expand the memory, we could have improved the real-time performance and addressing properties of the design while lowering the development cost. Additionally, if required, we could add a network to the system so that users identify a fingerprint using a remote log-in, making the system more convenient for use in a civil access control system.
- As the scheme became clear with the progress of the design, we improved the real-time processing operations, hardware acceleration, and algorithms for the most time-consuming area of the design: fingerprint image preprocessing. As a result, fingerprint image preprocessing consumes 30 seconds instead of the original 10 minutes. Additionally, because the algorithms are too complex, we could only convert some time-consuming modules into hardware and instructions for image normalization, orientation and frequency extraction, valid mask extraction, Gabor filtering, thinning, binarization, image expansion and corrosion, etc. If we had more time for optimization, we could design the system to complete minutia extraction in 1 second.

- We evaluated the design effect for processing time and fingerprint matching accuracy. To save time, we appropriately adjusted the accuracy requirements for some areas, and then chose the proper parameters after weighing.

- We used the Cyclone® II EP2C35 FPGA in this design, which has sufficient logic units but was deficient in supporting the required hardware arithmetic logic units. If the FPGA has fewer hardware arithmetic logic units than required by the algorithms, a system signal processing bottleneck will occur. Similarly, the CPU clock is hard to improve because of the FPGA features. In the future, we plan to leverage the abundant hardware arithmetic units in the Stratix FPGAs to accelerate operations and improve operation accuracy and speed.

During this contest, we deeply studied SOPC technology, mastered its elementary design concepts, and achieved our target. We thank our instructor, colleagues in the lab, and all the other participants. The joint effort paved the way for the success of our system.

Appendix: C2H Compiler Usage

Our system is a Nios II fingerprint identification system that performs image processing. We used the C2H Compiler for integer algorithms with many loops and simple operations. Because most algorithms in the system use floating-point numbers, we translated them into integers before using the C2H Compiler to optimize them into hardware.

The part of the design we optimized with the C2H Compiler is the Sobel operator, which we used for orientation extraction. This part included 4 loops and an integer accumulation algorithm. Addresses are stored as pointers. With the optimization, the algorithm speed of the Sobel operator increased 35 times.

Algorithm Description

We used the Sobel operator to obtain the fingerprint orientation. The Sobel operator calculates the gradients $G_x(U,v)$ and $G_y(U,v)$ and of each pixel point (i,j) in the X and Y directions. Figure 26 shows the Sobel operator's template coefficient.

Figure 26. Sobel Operator Template Coefficient

X Axis Direction

1	**2**	**1**
0	**0**	**0**
-1	**-2**	**-1**

Y Axis Direction

1	**0**	**-1**
2	**0**	**-2**
1	**0**	**-1**

Figure 27 shows the Sobel operator C2H function.

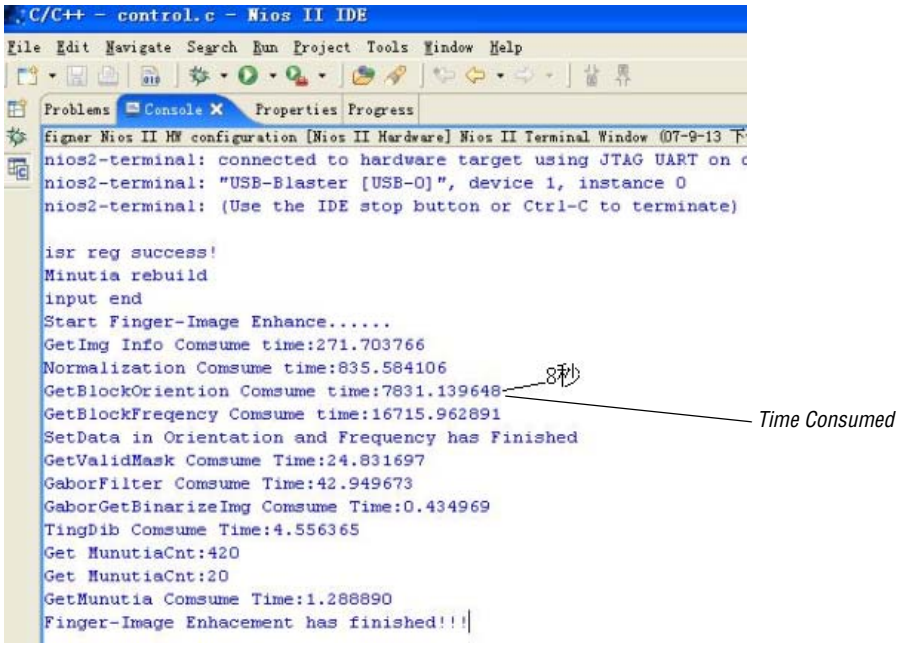
Figure 27. Sobel Operator C2H Compiler Function

```
void sobelc2h(int iWindow,Img *ImgSrc,int *pDst,int *pModelCente
{
  int iRow,iCol,i,j;
  int iWidth,iHeight;//iSize;
  UINT8 *pSrcAddr,*pSrc;
  int iResult;
  UINT8 uGray;
  iWidth = ImgSrc->size.iWidth;
  iHeight = ImgSrc->size.iHeight;
  pSrc = ImgSrc->pImg;

  //iSize = iWidth*iHeight;
  //针对每隔像素均进行求取沿X或Y方向上的梯度值
  for (iRow = iWindow;iRow < iHeight - iWindow;iRow++)
  {
    for (iCol = iWindow;iCol < iWidth - iWindow;iCol++)
    {
      pSrcAddr = pSrc + iRow * iWidth + iCol;
      //Sobel Core
      iResult = 0;
      for (i = -iWindow;i<=iWindow;i++)
      {
        for (j=-iWindow;j<=iWindow;j++)
        {
          uGray = *(pSrcAddr + i*iWidth + j);
          iResult += *(pModelCenter + i*iWinWidth + j) * uGray;
        }
      }

      *(pDst + iRow * iWidth + iCol) = iResult;
    }
  }
```

Performance Comparison

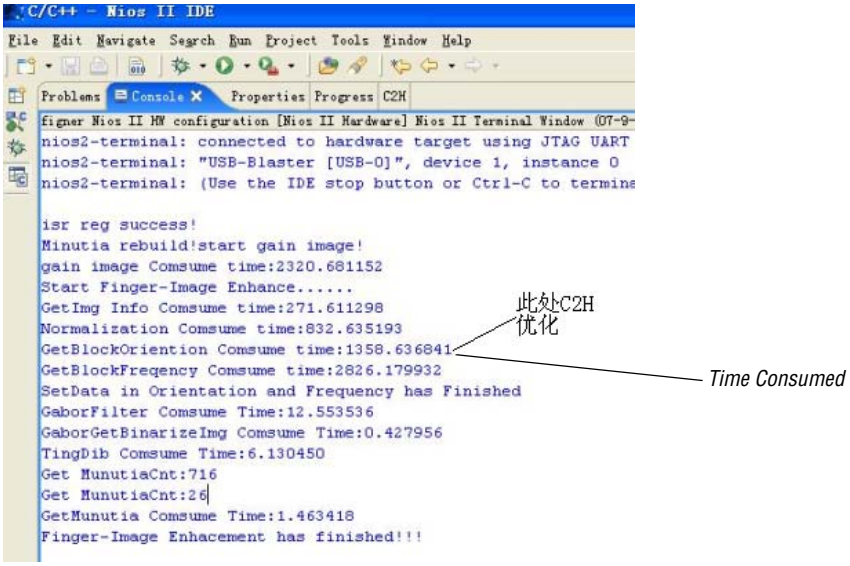
The Sobel operator's algorithm speed increased 35 times, which is only one step in orientation processing. The whole orientation extraction speed increased 6 times, and the effect was remarkable. Figure 28 shows the speed with a software-only implementation.

Figure 28. Consumed Time with Software-Only Implementation



After C2H optimization, the consumed time is only 1 second (see Figure 29).

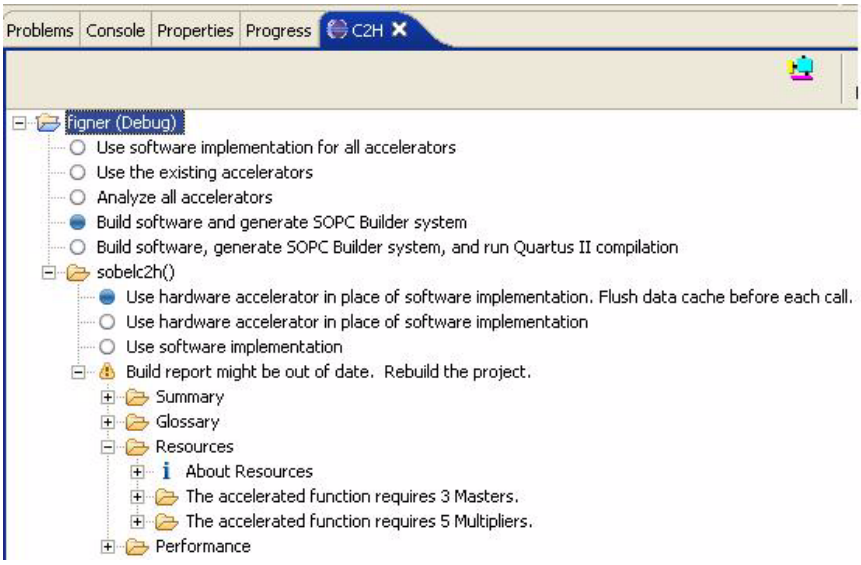
Figure 29. Consumed Time After Optimization



The processor's clock frequency is 100 MHz.

Figure 30 shows other logic that we accelerated and optimized.

Figure 30. Other Accelerated and Optimized Logic



Master Port

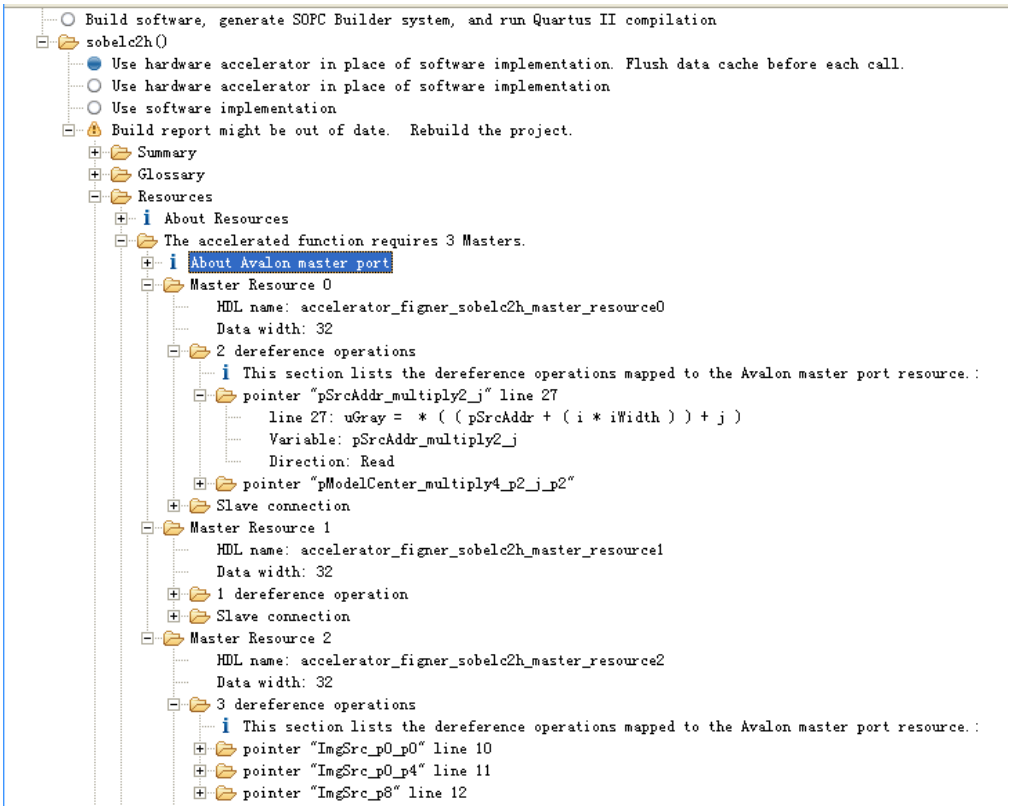
As with the algorithm, the SDRAM information is read in software through pointers. Because the C2H Compiler uses the master port to read and write SRAM and SDRAM and there are three different software pointer operations, we used three master ports for optimization. The optimization had

remarkable advantages and was very convenient to compile. The C2H Compiler freed us from developing complicated custom instructions or modules, compiling the master port operation, creating the hard-to-control working sequence of three master ports, and combining the operation with the algorithm. It also eliminated the need to implement a difficult algorithm sequence control for loading three master ports in a hardware description language. Figures 31 and 32 show the master port hardware and software, respectively.

Figure 31. Master Port Hardware

```
void sobelc2h(int iWindow,Img *ImgSrc,int *pDst,int *pModelCente
{
  int iRow,iCol,i,j;
  int iWidth,iHeight;//iSize;
  UINT8 *pSrcAddr,*pSrc;
  int iResult;
  UINT8 uGray;
  iWidth = ImgSrc->size.iWidth;
  iHeight = ImgSrc->size.iHeight;
  pSrc = ImgSrc->pImg;

  //iSize = iWidth*iHeight;
  //针对每隔像素均进行求取沿x或y方向上的梯度值
  for (iRow = iWindow;iRow < iHeight - iWindow;iRow++)
  {
    for (iCol = iWindow;iCol < iWidth - iWindow;iCol++)
    {
      pSrcAddr = pSrc + iRow * iWidth + iCol;
      //Sobel Core
      iResult = 0;
      for (i = -iWindow;i<=iWindow;i++)
      {
        for (j=-iWindow;j<=iWindow;j++)
        {
          uGray = *(pSrcAddr + i*iWidth + j);
          iResult += *(pModelCenter + i*iWinWidth + j) * uGray;
        }
      }

      *(pDst + iRow * iWidth + iCol) = iResult;
    }
  }
```

Figure 32. Master Port Software



Hardware Multiplier

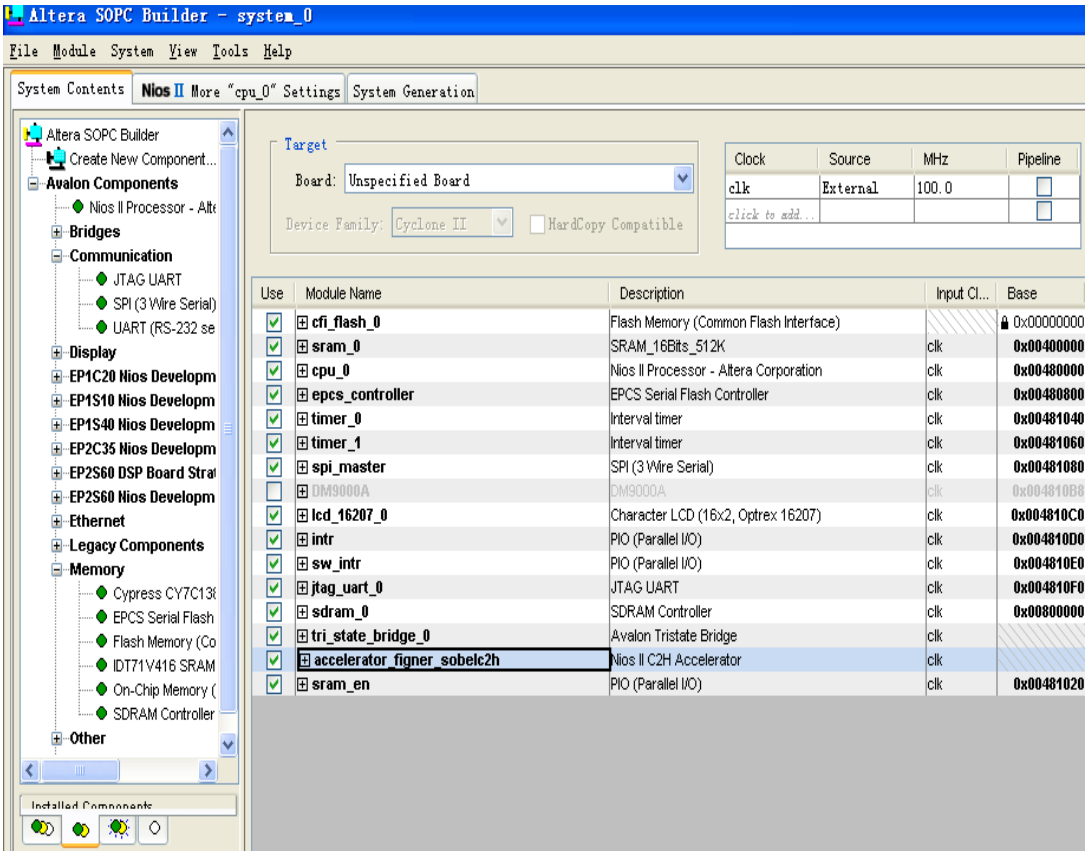
There are five multiplication operations in software that are automatically transformed into five hardware multipliers during C2H optimization. This implementation greatly shortens the processing time.

Loop

The effect of the loop optimization was remarkable. After being transformed into hardware, for, loop, while, etc. are just conditions. The loop algorithms are all transformed into hardware, increasing their speed. Therefore, using C2H optimization for image processing provides significant advantages.

Figure 33 shows the hardware in SOPC Builder.

Figure 33. Hardware In SOPC Builder



Accelerated with the C2H Compiler, the algorithm occupies 5,314 gates of logic resources, 512 memory bits, one M4K block, 22 DSP elements, and 11 18 x 18 DSP blocks (including three master ports and five multipliers). The operation speed of the Sobel operator increased 35 times and the speed of the orientation extraction process increased 6 times.

Conclusion

C2H hardware acceleration satisfied the system's time requirements and saved development time in the hardware module in exchange for using some hardware resources. A system can employ C2H hardware acceleration five times or more provided there are enough hardware resources.

During the design process, we tried to optimize the Gabor operator algorithm with the C2H Compiler because the algorithm contains so many pointer operations and complicated loop operations. Upon optimization, however, it required five master ports and 13 multipliers without giving a significant speed increase. Therefore, we gave up using the C2H Compiler for optimization and instead we optimized one operation in the loop using hardware. To optimize another area (and improve our skills), we used custom instructions instead of the Sobel operator.

To conclude, the C2H Compiler offers a sound optimization effect for algorithms that have many complicated loops, pointer operations, and master ports provided that the C2H conditions are met, that is, the operation does not have floating-point operations and the designer transfers arrays into pointers. The C2H Compiler is an excellent optimization tool. We believe it will become better and more flexible in the future.