情報保護學會論文誌
第14券 第5號, 2004. 10

융선추적을 이용한 지문 특징점 추출기의 SoC 구현

김 기 철[a)†], 박 덕 수[a)], 정 용 화[b)‡], 반 성 범[c)]
서울시립대학교[a)], 고려대학교[b)], 한국전자통신연구원[c)]

SoC Implementation of Fingerprint Feature Extraction System with Ridge Following

Kichul Kim[a)†], Deok-Soo Park[a)], Yongwha Chung[b)‡], Sung Bum Pan[c)]
University of Seoul[a)], Korea University[b)], ETRI[c)]

요 약

본 논문에서는 생체 정보를 이용한 사용자 인증분야에서 가장 널리 사용되는 지문의 특징점 추출기의 SoC(System-on-Chip) 구현을 보인다. 일반적인 지문 특징점 추출 방식은 이진화, 세선화 방법을 사용하는데, 저화질 지문 이미지의 경우 많은 오류가 발생하여 지문인식시스템 전체의 정확도를 떨어뜨리는 문제가 있다. 이러한 문제를 해결하기 위해 지문 이미지에서 융선을 직접 추적하여 특징점을 추출하는 알고리즘이 제안되었으나, 연산량이 많아 스마트카드와 같은 소규모 시스템에서 소프트웨어만으로 수행하기 어렵다는 문제점이 있다. 본 논문에서는 융선추적 알고리즘을 스마트카드용 SoC에서 구현하기 위한 방법을 제안한다. 하드웨어의 효율성을 높이기 위하여 융선추적 알고리즘을 변형하였으며, 알고리즘의 각 기능 블록은 수행할 연산량, 하드웨어 비용과 활용도, 그리고 시스템 효율성 등이 고려되어 하드웨어와 소프트웨어로 분리 구현되었다. 구현된 융선추적을 이용한 지문 추출기는 SoC용 IP로 개발이 되어, 여러가지 스마트카드용 SoC에서 사용될 수 있다.

ABSTRACT

This paper presents an System-on-Chip(SoC) implementation of fingerprint feature extraction system. Typical fingerprint feature extraction systems employ binarization and thinning processes which cause many extraction errors for low quality fingerprint images and degrade the accuracy of the entire fingerprint recognition system. To solve these problems, an algorithm directly following ridgelines without the binarization and thinning process has been proposed. However, the computational requirement of the algorithm makes it hard to implement it on SoCs by using software only. This paper presents an implementation of the ridge-following algorithm onto SoCs. The algorithm has been modified to increase the efficiency of hardwares. Each function block of the algorithm has been implemented in hardware or in software by considering its computational complexity, cost and utilization of the hardware, and efficiency of the entire system. The fingerprint feature extraction system has been developed as an IP for SoCs, hence it can be used on many kinds of SoCs for smart cards.

Keywords : *Fingerprint Recognition, Feature Extraction, SoC, Embedded System*

접수일 : 2004년 6월 1일 ; 채택일 : 2004년 8월 19일
† 주저자 : kkim@uos.ac.kr
‡ 교신저자 : ychungy@korea.ac.kr

Ⅰ. 서 론

전자산업의 급속한 발전과 고도의 네트워크화 과정은 인터넷 뱅킹과 같은 무인원격 시스템이 가능하게 하였다. 이에 따라 개인정보보호 및 사용자 인증에서의 보안문제가 중요한 이슈로 떠오르고 있다. 지문 인식[1]은 사용이 간편하고도 유지비가 적게 들면서 분실 가능성 및 오용의 위험이 적은 특성을 가지므로 개인 인증 분야의 응용에 많은 관심이 집중되고 있다.

개인 인증 분야의 또 다른 방법으로 스마트카드를 이용하는 방법이 있다. 기존의 마그네틱카드와 형태는 같지만 내부에 마이크로프로세서를 포함한 소규모 SoC(System-on-Chip)을 내장하고 있어 기존의 카드와 비교해 월등한 기능을 가지고 있다. 그러나 기존의 지문을 사용한 개인 인증은 PC를 기반으로 한 소프트웨어 중심으로 개발되고 있어, 스마트카드와 같은 소규모 SoC에서의 지문인증 기술이 매우 시급한 상황이다.

특히, 지문의 끝점(termination)과 분기점(bifurcation)으로 대표되는 특징점(minutiae)[1] 추출은 많은 양의 연산을 요구하여 SoC에서 구현하기가 쉽지 않은 부분이다. 대부분의 지문 특징점 추출 방식은 전처리(이미지개선 → 이진화 → 세선화) 과정을 통해 특징점을 추출하고, 특징점 추출 후 잘못된 의사특징점(false minutiae)을 없애는 후처리 방식을 사용하여 왔다. 그러나 이러한 방식은 지문 이미지가 좋지 않은 경우에 성능이 급격히 저하되는 단점이 있으며, 많은 연산량을 필요로 하여 스마트카드와 같은 제한된 자원을 갖는 환경에서 시스템을 구현하기에는 상당한 어려움이 따른다[2].

본 논문에서는 지문 이미지에서 융선 추적을 통한 알고리즘[3]을 이용한 특징점 추출 시스템을 SoC에 구현하였다. 융선추적 알고리즘은 기존의 알고리즘의 문제점인 이진화 및 세선화에서 발생하는 지문 이미지의 왜곡 현상이 없으므로 저화질의 지문 이미지에서도 보다 정확한 특징점 추출이 가능한 장점이 있다. 또한 기존의 알고리즘 보다 적은 계산량을 사용한다. 하지만 융선추적 알고리즘은 발표 당시 스마트카드와 같이 제한된 자원을 가진 SoC에서의 구현을 고려하지 않았기 때문에 알고리즘이어서 스마트카드에서 직접 구현하기에는 매우 어려운 알고리즘이다. 본 논문에서는 하드웨어와 소프트웨어를 동시에 사용하여 융선추적 알고리즘을 스마트카드용 SoC에 구현하는 것을 보인다. 구현된 융선추적을 이용한 지문 추출기는 SoC용 IP로 개발되어, 여러가지 스마트카드용 SoC에서 사용될 수 있다

본 논문의 구성은 다음과 같다. 2 장에서 지문특징점 추출, 특히 지문 융선추적 알고리즘에 관하여 소개한다. 3 장에서는 SoC에서의 지문 특징 추출 시스템 개발방법 및 각 기능블록의 소프트웨어와 하드웨어 설계를 설명한다. 4 장에서는 전체 시스템의 구현 및 성능평가를 기술하고, 마지막으로 5 장에서 본 논문의 결론을 맺는다.

Ⅱ. 지문특징점 추출

지문이란 인간의 손바닥에 존재하는 땀구멍이 융기한 선으로 형성된 문향을 의미하고, 지문인식이란 이러한 융선(ridge)의 흐름을 분석해 같은 흐름을 보이는 지문을 찾는 과정이라고 할 수 있다. 지문인식에 있어 필요한 생체 정보를 특징(feature)이라고 하는데, 특별히 지문에 나타나는 특징은 특징점(minutiae)이라고 한다. 특징점은 끝점과 단점의 두가지 타입으로 나뉘는데, 끝점이란 융선의 흐름이 끊어진 곳을 말하며, 분기점이란 두 융선이 하나가 되는 곳을 말한다.

지문인식은 크게 추출(extraction)과정과 정합(matching)과정으로 분리되는데, 추출과정은 지문 이미지에서 특징점의 정보를 찾아내는 과정이고, 정합과정은 추출된 특징점 정보를 미리 입력되어 있는 기존의 특징점 정보와의 비교를 통하여 동일 지문임을 판별하는 과정을 말한다. 본 장에서는 일반적인 지문인증 시스템에서 사용되는 특징점 추출 방식과 본 논문에서 구현되는 융선추적 알고리즘 방식을 간단히 소개한다.

2.1 일반적인 지문 특징점 추출 알고리즘

일반적으로 지문을 이용한 개인인증 시스템은 입력된 지문 이미지에서 추출된 각각의 특징점의 좌표(x,y), 각도(θ), 그리고 특징점의 특성(끝점, 분기점)을 추출하여 사용한다. 그러나 이러한 특징점을 정확하게 추출하는 것은 상당히 어려운데, 다음과 같은 지문 자체의 특성때문이다. 즉, 손가락은 인간이 행하는 작업에 있어 직접 접촉이 가장 많은 신체부분이기 때문에, 손가락 끝에 위치한 지문은 더러워지고, 상처받기 쉬우며, 마르거나 젖거나 하는 상태변

화가 많다. 따라서, 특징점 추출을 위하여 여러 단계의 처리과정을 거치지만, 많은 의사특징점(false minutiae)[4]을 수반하게 된다. 현재 일반적으로 사용되고 있는 특징점 추출 알고리즘을 요약하면 다음과 같다.

2.1.1 전처리(Pre-processing)

전처리는 이미지 개선, 이진화, 세선화의 단계로 구성된다. 이미지 개선 단계에서는 지문 이미지에서 발생하는 잡음을 줄여 지문의 융선과 골의 구분을 명확히 한다. 이미지 개선이 끝나면 융선을 추출하는 과정을 수행한다. 보통 지문 이미지는 256 gray scale을 가지는데, 이를 단순화 하면 융선과 골의 이진정보로 나타낼 수 있다. 전처리의 마지막 단계인 세선화는 융선의 폭을 한 화소로 줄이는 작업을 말한다. 이 과정은 찾은 융선의 연결성을 충분히 유지함과 동시에 세선화를 통해 발생할 수 있는 잘못된 특징점 정보를 최소화 하는데 그 목적이 있다[5,6].

2.1.2 특징점 추출

전처리 과정이 끝나면 특징점을 찾는 과정이 수행된다. 세선화된 이미지에서는 주변 화소값과의 비교를 통하여 끝점과 분기점을 추출하게 된다[7].

2.1.3 후처리(Post-processing)

찾아낸 특징점들 중에는 원 이미지의 손상으로 인하여 발생하는 의사특징점(false minutiae)이 있다[4]. 이는 지문 이미지의 입력시 상처 등으로 인해 융선이 끊어지거나, 압착력의 변화로 인하여 융선의 모양이 잘 나타나지 않는 부분이 잘못된 세선화 과정을 통하여 발생하는 것이 대부분이다. 후처리 과정은 이런 의사특징점을 없애는 과정에 해당된다.

2.2 지문 융선추적 알고리즘

지문 융선추적 알고리즘[3]은 앞에서 설명한 일반적인 방식을 사용하지 않는다. 이는 세선화와 이진화 과정에서의 많은 연산량을 줄이며, 전처리 과정을 통하여 발생하는 지문 이미지의 왜곡현상을 방지하기 위함이다. 즉, 이진화/세선화 과정 대신, 입력된 지문이미지에서 gray scale 값을 직접 사용하여 지문의 융선을 찾아 추적하는 방법을 사용한다.

그림 1에 융선추적 알고리즘이 설명되어 있다. 융

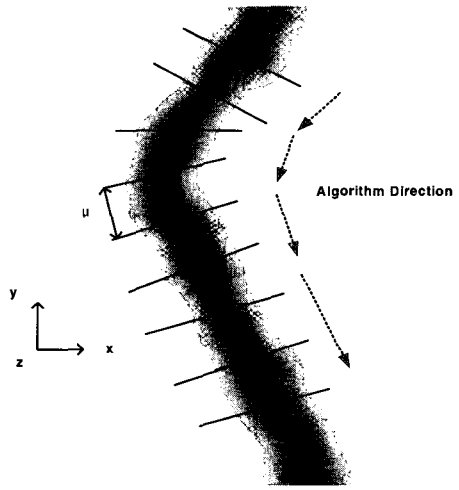


그림 1. 융선추적 알고리즘[3]

선추적 알고리즘에서는 지문 이미지에서 융선크기의 2배에 해당되는 임의의 구역에서 gray level 값이 가장 높은 화소의 위치를 융선의 중심이 되는 점으로 사용한다. Gray level 값이 가장 높은 화소의 집합이 융선을 이루는 점에 착안한 방식이다. 일정 구역에서 gray level 값이 가장 높은 화소에서의 방향성(각도)을 구하여 그 값을 φ라 한다. 그리고 φ 방향으로 일정부분(μ)을 이동하여 다시 임의의 구역을 선정한다. 임의의 구역에서의 gray level 값이 높은 점을 찾고, 구역내에서 가장 높은 gray level 의 위치들을 연결하게 되면 융선을 이루게 되므로, 최대 화소의 위치를 연결, 융선을 추적하게 된다. 이렇게 찾아낸 융선을 입력된 지문 이미지와 동일한 크기의 보조이미지에 융선만의 이미지를 재구성하며 특징점을 추출한다[3].

융선추적 알고리즘은 이진화, 세선화와 같이 많은 연산량을 요하는 과정을 사용하지 않아 상대적으로 적은 계산량을 필요로 한다. 또한 지문 이미지에서 직접 융선을 추적하므로 이진화, 세선화에서 발생하는 지문 이미지의 왜곡현상에 따른 오류를 줄일 수 있으므로, 저화질의 지문 이미지에서도 정확하게 특징점을 찾아낼 수 있다. 예를 들어, 미국표준기술연구소(NIST : National Institute of Standards and Technology)에 등록되어져 있는 지문을 사용하면, 기존 알고리즘의 경우 각각 216%[5], 119%[6], 33%[7], 207%[8]의 특징점 추출 에러를 보이지만, 융선추적 알고리즘은 26%의 에러를 나타낸다[3].

융선추적 알고리즘이 기존의 알고리즘들보다 상대

적으로 적은 연산량을 필요로 하지만 스마트카드용 SoC에서 소프트웨어로 실시간에 수행하기에는 연산량이 상대적으로 많다. 따라서 스마트카드용 SoC에서 융선추적 알고리즘을 수행하기 위해서는 하드웨어와 소프트웨어를 동시에 이용하는 방법을 사용하여야 한다.

Ⅲ. 지문특징점 추출을 위한 임베디드 시스템

3.1 기능블록의 하드웨어-소프트웨어 분리 및 알고리즘 변경

융선추적 알고리즘을 SoC에 구현하려면 알고리즘의 각 요소를 소프트웨어로 구성하는 부분과 하드웨어로 구성되는 부분으로 구분하고, 각 부분이 서로 연동하여 동작할 수 있도록 인터페이스를 구성하여야 한다. 하드웨어-소프트웨어 분리를 위하여 융선 추적 알고리즘의 주요 과정을 살펴보면, 그림 2(a)에서와 같이 크게 시작점 선정 방법과 융선추적 방법으로 분리된다[3]. 시작점 선정은 이미지안에 있는 모든 융선을 찾아내기 위하여 필요한 과정으로 이미지에 정사각형 모양의 그물을 만들고 그물의 각 교차점 근처에 아직 찾아지지 않은 융선있는지 알아보아 찾아지지 않은 융선이 있으면 새로이 융선추적를 시작하게 하는 과정이다. **시작점 선정**은 1) 정사각형 그물모양의 교차점 중 한 점을 시작점으로 선정(Starting point), 2) 교차점에서의 각도연산(Angle computation), 3) 각도와 교차점의 좌표로 구역설정(Section set), 4) 구역내에서 마스크(mask)연산을 통한 최대화소 위치결정(Max. detection), 5) 최대화소가 이전의 융선추적에서 추적된 위치인지 확인(New starting point?), 6) 알고리즘 진행경우 최대화소 위치에서의의 각도연산(Angle computation), 7) 이전에 추적된 좌표일 경우 1) 과정부터 재 시작의 순서를 갖는다.

반면, **융선 추적**은 1) 최대화소의 위치에서 φ의 방향으로 μ 만큼 진행되는 위치선정(Move), 2) 이동점을 중심으로 하는 φ + π/2 방향의 구역을 설정(Section set), 3) 마스크연산을 통한 최대화소의 위치결정(Max. detection), 4) 특징점의 발생여부를 확인(New minutiae?), 5) 특징점이 발생하지 않았을 경우 최대화소 위치에서의 각도연산 및 보조이미지 업데이트 후 1) 과정부터 재 시작, 6) 특징점 발생경우 정보저장 및 새로운 시작점선정의 순서로 진행된다.

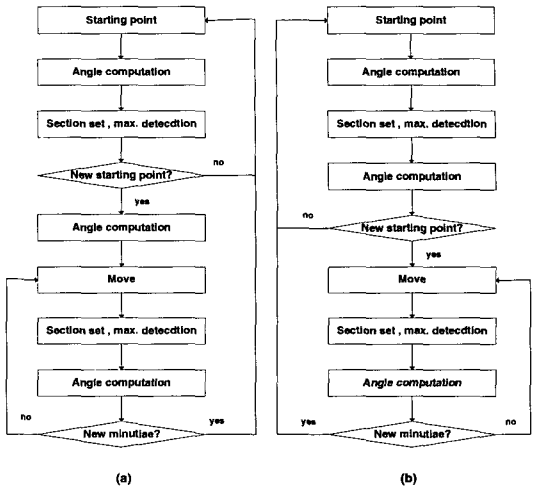


그림 2. (a) 융선추적 알고리즘의 흐름도 (b) 하드웨어 구현을 위한 흐름도

위의 순서를 자세히 관찰하면 두 가지 알고리즘 모두 각도연산 후 구역설정 및 최대화소 위치 결정의 순서가 변하지 않음을 확인 할 수 있다. 또한, 각도연산과 구역설정 및 최대화소 위치결정은 그 연산량과 활용도면에서 융선 추적 알고리즘의 대부분을 차지하고 있음을 알 수 있다. 따라서 각도연산과 구역설정 및 최대화소 위치결정 블록들은 하드웨어로 구현됨이 바람직하다.

하드웨어 블록을 효율적으로 사용하기 위하여 그림 2(b)에서와 같이 알고리즘의 순서를 변형할 수 있다. 즉, 시작점 선정은 1) 각도연산, 2) 구역설정, 최대화소 결정, 3) 각도연산, 4) 최대화소의 적합성 여부 확인(보조이미지) 순으로, 융선 추적은 1) 새로운 구역 설정 위한 융선 이동, 2) 구역설정, 최대화소 결정, 3) 각도연산, 4) 최대화소의 적합성 여부 확인(β 조건확인), 5) 보조이미지 업데이트 순으로 변형할 수 있다.

이와 같이 융선추적 알고리즘을 변형하면 연산량이 많고 하드웨어 구현시 효율적인 구조가 가능한 각도연산과 구역설정, 최대화소 결정 블록이 하드웨어로 구현되었을 때 알고리즘의 수행이 하드웨어에서 소프트웨어로 전환되는 부분은 각도연산(Angle computation)에서만 발생하므로 효율적인 시스템의 구성이 가능해진다.

알고리즘의 변형에 대하여 주목할 점은 기능 블록들의 수행순서를 바꾼 것이지 알고리즘의 본질이 바

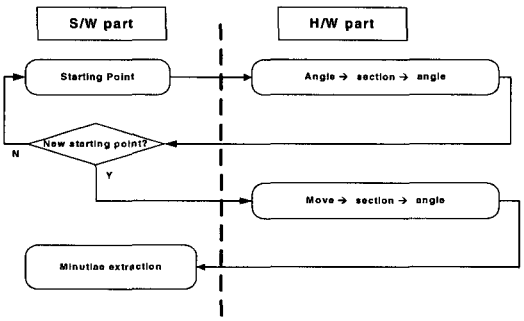


그림 3. 하드웨어와 소프트웨어 분리

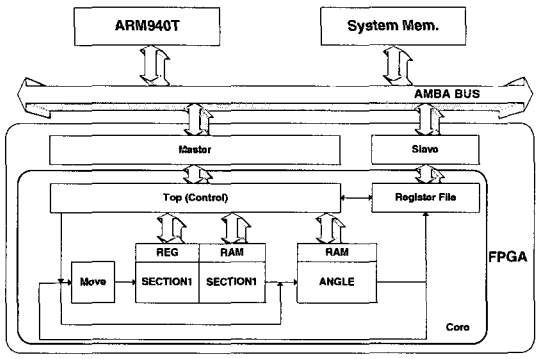


그림 4. 하드웨어 개발환경

뀐 것은 아니라는 것이다. 즉, 동일한 지문 이미지에 대하여 원래의 융선추적 알고리즘과 본 논문에서 사용된 변형된 알고리즘은 동일한 결과를 보인다. 이 변형의 장점은 하드웨어에서 소프트웨어로 전환되는 부분을 줄여 시스템을 간단하게 하는 것이며 단점은 새로운 시작점이 아닌 점에 대해서도 각도연산을 수행한다는 것이다.

반면, 특징점 추출 블록 및 보조이미지 업데이트 블록은 소프트웨어로 구현함이 타당하다고 할 수 있으며, 이렇게 분리된 결과를 요약하면 그림 3과 같다.

융선추적 알고리즘에서 주어진 한 점에서의 융선의 각도연산은 그 과정이 매우 복잡하고, arctangent, square root와 같은 함수를 사용하여 그 계산량이 많다. 따라서 각도연산을 전처리 과정과 알고리즘 수행 중 과정으로 분리하여 처리한다. 전처리 과정에서는 지문 이미지가 입력될때 지문 이미지내의 8×8 화소 블록마다 방향성을 미리 계산한다. 알고리즘 수행중 각도연산에서는 약간의 오차를 허용하지만, 연산량을 대폭 줄일 수 있는 Lagrangian interpolation 방법을 사용한다. 즉, 전처리에서 미리 계산된 8×8 블록에서의 각도를 이용하여 알고리즘이 필요한 위치에서의 각도를 Lagrangian interpolation 방법으로 구한다. 본 논문에서는 전처리 과정을 소프트웨어로 수행하고 알고리즘 수행 중의 각도연산은 하드웨어로 처리한다.

3.2 SoC의 구성과 개발 환경 및 개발 방법

본 논문에서는 그림 4에서 보는 바와 같이 시스템 버스가 구현된 보드에 CPU, 메모리, 그리고 FPGA를 탑재하여 개발이 이루어졌다. 구현된 지문 특징점 추출 시스템은 스마트카드와 같은 SoC에의 응용을 고려하였으며, 이를 위하여 CPU는 저전력의 32-bit 범용 RISC 프로세서인 ARM940T를 채택하였다. 또한, 온칩 버스로 구현될 시스템 버스는 ARM CPU를 사용하는 스마트카드에서 널리 사용될 것 예상되는 AMBA(Advanced Microcontroller Bus Architecture) AHB(Advanced High-performance Bus) 버스를 채택하였다. AMBA AHB 버스는 높은 대역폭과 복수의 bus master를 지원하는 고성능 버스이며, 스캐너 등 주변 장치와의 연결을 용이하게 한다.

FPGA는 Xilinx Virtex V2000EFG680을 사용하였고, SoC의 운영체제로는 핸드폰 등 각종 SoC에서 실시간 운영체제로 널리 채택되고 있는 Vxworks를 채택하였다. 프로그램 개발 및 시뮬레이션 환경을 포함하는 호스트 PC는 이더넷을 통하여 개발 시스템과 연결되었으며, 하드웨어 설계, 디버깅, 소프트웨어 개발을 동시에 진행하였다.

하드웨어 블록은 우선 VHDL을 사용하여 설계하였다. VHDL 프로그램은 기능 시뮬레이션을 수행한 후 합성되었다. 합성된 결과는 각 기능 블록의 기능 검증 및 하드웨어 비용, 복잡도, 그리고 성능에 대한 예측에 이용되었다. 소프트웨어 개발은 Vxworks의 통합 개발 환경인 Tornado에서 이루어졌으며, 소프트웨어로만 구현된 지문 특징점 추출 프로그램에서 하드웨어 기능 블록이 수행할 기능을 제거하고 하드웨어와의 통합운영에 필요한 연동 모듈을 추가하는 방법이 사용되었다.

하드웨어와 소프트웨어 각각의 설계 결과에 대한 검증이 완료되면, 하드웨어와 소프트웨어를 연동하여 통합 시뮬레이션을 수행하였다. 최종적으로 지문 이미지로부터 특징점을 추출하여 성능을 확인하였으며, 소프트웨어로만 구현된 특징점 추출 프로그램과 그 결과를 비교함으로써 성능을 검증하였다.

3.3 하드웨어, 소프트웨어 기능블록 설계

3.3.1 융선 이동 블록

융선이동 블록은 최대화소값의 좌표값 (x_c, y_c)와 최대화소 좌표에서의 방향 φ_c를 입력으로 받아, φ_c 방향성으로 μ 화소만큼 이동한 후의 좌표 (x_t, y_t)를 구하는 블록에 해당된다.

$$x_t = x_c + \mu\cos\varphi_c$$
$$y_t = y_c + \mu\sin\varphi_c \quad (1)$$

식 (1)의 연산은 기본적으로 x, y 축의 화소의 위치를 결정하기 위하여 한번의 곱셈연산과 한번의 덧셈연산이 필요하고, 연산을 위한 cosine, sine 값을 ROM을 이용하여 저장해 놓아야 한다.

효율적인 하드웨어 기능블록의 구현을 위하여 cosine, sine 값을 저장하는 대신 이동위치를 저장하는 방식을 사용한다. 파라미터 μ는 입력에 따라 변하는 값이 아니기 때문에, cosine, sine 값을 결정하는 방향성(φ)의 입력에 따라 μ×cosφ, μ×sinφ 값은 시뮬레이션을 통해 모든 경우를 찾아낼 수 있게 된다. 지문의 입력 이미지의 크기와 융선의 굵기에 따라 파라미터 μ 값의 최적화는 변화될 수 있으나, ROM을 이용한 하드웨어 구현을 위하여 시뮬레이션을 실시한 결과 파라미터 μ 값은 3으로 셋팅하는 것이 가장 이상적임을 확인하였다.

각도에 따른 x축과 y축의 이동좌표는 μ 값 설정을 통한 시뮬레이션을 수행하였으며, 전체 이동경로는

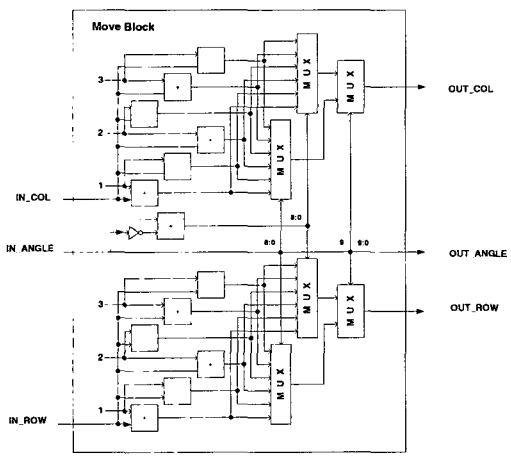


그림 5. 융선이동 블록 다이어그램

양수와 음수를 구분하여 총 26가지 임을 확인하였다. 그러므로 입력된 각도(방향성)에 따라 26가지 중 어느 방향에 속하는 지를 판단하여 진행후의 좌표값을 연산할 수 있는 구조가 가능하게 된다. 이와 같은 특성을 이용하면 빠른 연산이 가능한 효율적인 구조로 하드웨어 기능블록을 구현할 수 있다. 융선 이동 블록을 그림 5에 나타내었다.

3.3.2 구역설정 및 최대화소 결정 블록

구역설정 및 최대화소 블록은 크게 두 가지의 기능을 수행한다. 첫째, 입력좌표를 기준으로 융선의 방향에 수직 방향으로 길이가 $(2\sigma+1)$인 구역을 설정하여 구역의 시작점과 끝점을 계산한 뒤, Bresenham 알고리즘[9]을 이용하여 두 점 사이를 연결하는 구역의 좌표와 구역에 평행한 두 보조 구역의 좌표를 결정하는 일이다. 둘째로는 앞서 계산된 구역에 대해 2단계의 마스크연산을 수행한 후, 최대화소의 위치 (x_n, y_n)를 결정하고, 이전의 최대화소의 좌표 (x_c, y_c), 이동한 후의 좌표 (x_t, y_t), 그리고 새로운 최대화소의 좌표 (x_n, y_n)이 표시하는 점을 각각 A, B, C라고 할 때, 각 ∠BAC가 일정한 값 β의 범위 안에 있는지를 확인하는 기능이다. 구역설정 및 최대화소 결정 블록은 아래와 같은 세부 블록으로 설계되었다.

▸ **Sectioning Block**

Sectioning block은 Bresenham 알고리즘에 의해 구역설정 및 구역과 평행한 상하 양방향의 좌표값을 구하는 기능블록에 해당된다. 시뮬레이션을 통해 융선이동 블록의 파라미터 μ 값이 3으로 셋팅되었을 경우, σ 값은 7이 가장 적합함을 확인 할 수 있었다. 따라서 section의 좌표는 입력좌표를 기준으로 -7에서 7까지의 범위에 있으므로, 구하는 좌표는 입력좌표를 기준으로 하여 그 변위만을 계산한다. 이러한 범위에서의 Bresenham 알고리즘의 수행결과는 45°를 간격으로 8가지의 패턴을 반복하게 된다. 즉, $0° < \theta < 45°$에서 나타나는 8가지의 패턴에 대하여 시작점 변위와 Bresenham 알고리즘 수행결과를 미리 ROM에 저장해 놓은 뒤, 적절한 값을 필요에 따라 사용한다. 평행한 보조의 구역은 앞서 연산이 수행된 section을 이용한다.

▸ **Regularize Block**

2단계의 마스크연산을 수행한 후에 이전의 최대화

소의 좌표 (x_c, y_c), 이동한 후의 좌표 (x_t, y_t), 그리고 새로운 최대화소의 좌표 (x_n, y_n)이 표시하는 점 A, B, C가 만드는 각 ∠BAC가 일정한 값 β의 범위 안에 있는지를 확인하는 기능블록에 해당된다. 첫번째 마스크연산은 설정된 구역과 상하 보조 구역과의 대응되는 화소값들의 평균을 구하는 것이고, 두번째 마스크연산은 길이가 7인 마스크를 씌우는 연산에 해당된다. 매 클럭마다 메모리에서 구역과 보조구역의 gray level 값을 읽어 들여 accumulator를 사용하여 평균값을 구한 뒤, 레지스터 뱅크(bank)를 이용하여 두번째 마스크연산을 수행하게 된다. 그 값을 최대화소값과의 비교를 통하여, 새로운 최대화소값이 발생할 경우 그 값을 저장하게 된다. 후에 최대화소의 좌표를 읽어야 하므로, 현재의 최대화소가 구역의 몇번째 좌표에 해당되는지를 같이 저장해 두어야 한다. 2단계의 마스크연산이 끝나면, 최대화소값의 좌표를 읽기 위하여 sectioning_register_file을 읽어야 한다. 레지스터 파일에서 읽는 값은 기준점에서의 변위에 해당된다. 여기서 앞서 살펴본 β 조건을 적용하여야 하는데, 지문 이미지의 특성과 융선의 변화를 통하여 실험을 수행한 결과, β의 값은 30°가 가장 적당한 값임을 알 수 있었다. 최대화소의 좌표는 레지스터 파일에서 읽은 값과 기준점을 더하여 출력하게 된다.

▸ **Sectioning_Register_File**

Sectioning 블록에서 계산된 구역과 보조구역의 결과(변위) 값을 쓰는 레지스터 파일블록에 해당된다. 구역은 최대크기가 15(σ = 7) 이므로 write 주소는 0~14의 범위를 갖게 된다.

▸ **Memory**

Sectioning에서 계산된 좌표의 gray level 값을 레지스터파일에 쓰여진 순서대로 구역과 보조구역을 교대로 상위모듈이 쓰고, regularize 블록이 읽게 된다.

▸ **Point_Valid_Unit**

Sectioning의 결과를 상위 모듈에 넘겨 줄 때 좌표가 아닌 이미지의 주소값의 형태로 바꾸어 주는 역할을 하는 블록이다. 또한, 최대화소의 결과좌표가 이미지 영역에서 벗어난 경우 잘못된 동작을 일으킬 수 있으므로, 이에 대한 처리를 하는 기능도 함께 수행하게 된다. Regularize 블록에서 메모리에 접근

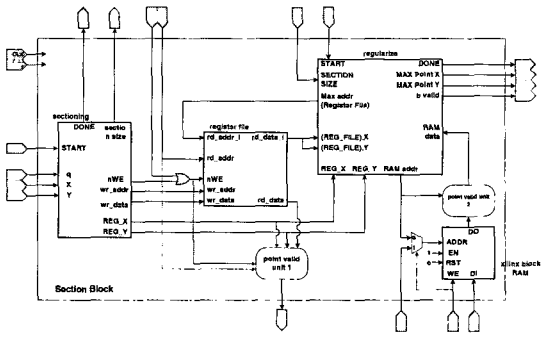


그림 6. 구역설정 및 최대화소 결정 블록다이어그램

할 때에는, 메모리의 read 주소에 해당하는 valid_register가 유용하지 않으면 메모리의 데이터 대신 '0'을 출력해 주는 기능을 수행한다.

이와 같은 5개의 보조 블록을 통합하여 구역설정 및 최대화소 결정 블록이 구현되었고, 그림 6에 구역설정 및 최대화소 결정 블록을 나타내었다.

3.3.3 각도 연산 블록

각도연산 블록은 구역설정 및 최대화소 결정블록에서 gray level이 최고인 값을 찾을 경우 그 점에서의 방향성을 구한다. 하드웨어의 효율성과 빠른 연산을 위해 Lagrangian interpolation을 사용한다. 실시간 처리를 위하여 지문 이미지가 입력으로 받아들여지는 부분에서 일정한 간격(8×8)으로 각 좌표에서의 진행방향을 전처리 과정에서 미리 연산하여 메모리에 저장하여 두고, 저장된 값을 불러와 Lagrangian interpolation을 사용하여 입력된 좌표에서의 진행 방향을 구한다. 각도 연산 블록의 구조를 그림 7에 나타내었다.

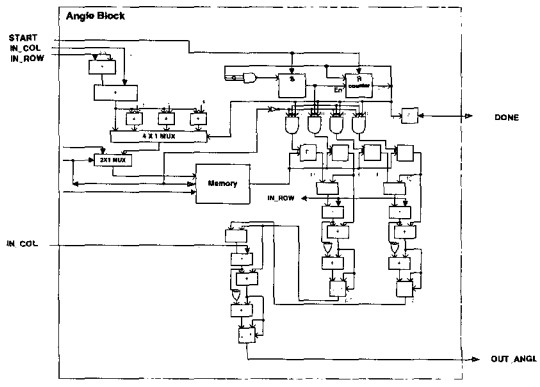


그림 7. 각도연산 블록의 구조

Ⅳ. 시스템 구현 및 성능 평가

4.1 전체시스템 구현

본 절에서는 지문 특징점 추출 시스템에서 사용되는 전체 하드웨어 모듈의 구조와 동작을 설명한다. 지문 특징 추출 하드웨어의 전체 블록은 그림 4에서 보인 바와 같이, 크게 master 블록, slave 블록, 그리고 core 블록으로 구분된다.

AMBA 버스에서 slave 모듈은 능동적으로 데이터 전송을 수행하지 못하고 데이터가 쓰이거나 읽혀지는 역할만을 수행한다. 반면에 master 모듈은 스스로 데이터 전송을 발생시킬 수 있다. 그림 4에서 slave 블록은 레지스터 파일과 외부 인터페이스를 연결해 주는 역할을 한다. 소프트웨어 모듈은 외부 인터페이스를 통하여 slave의 레지스터 파일에 연산에 필요한 정보를 입력한다. 알고리즘 진행을 위하여 지문 이미지에서 시작점 선정 또는 융선추적 알고리즘이 시작되는 화소위치의 정보가 입력된다. 최초 하드웨어-소프트웨어 모듈간 통신의 경우, 지문 이미지의 시작 주소와 8×8 화소에 대하여 미리 연산된 각도값이 저장된 메모리의 시작 주소와 함께 입력으로 들어오게 된다. 하드웨어 모듈의 레지스터 파일 입력값에 따라 지문 특징추출 하드웨어는 시작점 선정 또는 융선추적 알고리즘의 흐름에 맞는 연산을 수행하게 된다. 하드웨어 모듈의 연산 결과인 좌표와 각도가 다시 레지스터 파일에 입력이 된다. 하드웨어 모듈의 연산이 끝나면 하드웨어는 인터럽트를 발생시켜 지문추출 소프트웨어가 계속하여 지문추출에 필요한 연산을 수행하게 한다.

Master 블록은 연산에 필요한 지문 이미지의 화소값을 하드웨어 모듈에 입력하는 역할을 한다. 각도연산 블록과 구역설정 블록의 경우에는 연산수행을 위하여 미리 연산된 각도값과 지문 이미지의 gray level 값이 필요로 하게 되는데, 이 데이터들은 master 블록을 통해 메모리에 저장되어 사용된다.

Core 블록은 크게 1) 융선이동 블록, 2) 구역설정 및 최대화소 결정 블록, 3) 각도연산 블록의 세 개 블록으로 구분되고, 이 세 개의 블록을 제어하는 컨트롤 블록이 있다. 융선이동 블록의 경우는 시작점 선정에는 사용되지 않으며 융선추적 알고리즘에서만 수행된다. 그림 8에 이러한 하드웨어의 흐름도를 나타내었다.

소프트웨어 모듈은 각도연산 블록의 Lagrangian

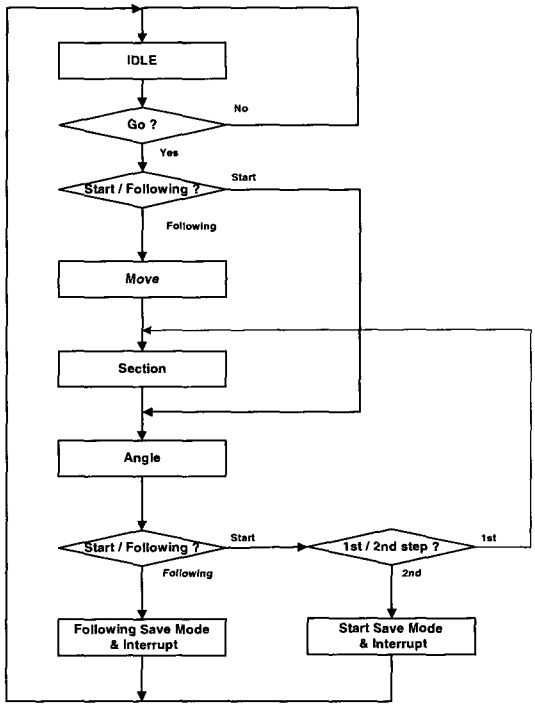


그림 8. 하드웨어 전체 흐름도

interpolation 연산을 위하여 8×8 화소 크기의 각도연산을 수행하고, 그 결과를 1° 간격의 정수값으로 근사화하여 메모리에 저장한다. 하드웨어 블록이 처음 구동되면, master 블록은 이 각도를 하드웨어의 내부 메모리에 저장하게 된다.

시작점 선정의 경우의 데이터 흐름은 각도연산 → section1(gray level의 주소값 연산) → data get → section2(maxpoint 선정) → 각도연산 순의 흐름을 가지며, 융선추적 알고리즘은 move → section1(gray level의 주소값 연산) → data get → section2(maxpoint 선정) → 각도연산의 순서로 진행된다.

하드웨어에서 한번의 수행이 끝나게 되면, 소프트웨어는 새로운 시작점 선정을 할 것 인지, 융선추적 알고리즘을 수행할 것인지를 결정하여야 한다. 이를 위해 소프트웨어는 보조이미지를 사용하게 된다. 하드웨어에서 시작점 선정의 결과가 오면, 보조이미지와의 비교를 통하여 시작점으로써의 여부를 판단한다. 반면, 하드웨어에서 융선추적 알고리즘의 결과가 오게 되면, 보조이미지와의 비교를 통하여 융선추적 알고리즘을 계속 수행할 것인지 여부를 판단하고, 보조이미지를 업데이트한다.

4.2 성능 평가

본 논문에서 개발된 지문 특징점 모듈은 스마트카드용 SoC에서 동작하도록 설계되었으므로 하드웨어의 크기, 동작속도, 그리고 알고리즘 수행시간이 성능평가의 기준이 될 수 있다. 표 1에 본 지문추출 하드웨어가 사용한 주요 자원이 나타나 있다. 표 1의 결과는 지문추출 하드웨어를 Xilinx Virtex V2000 EFG680에서 구현하였을 때의 결과이다. 또한 하드웨어는 33MHz 클럭에서 동작하였다.

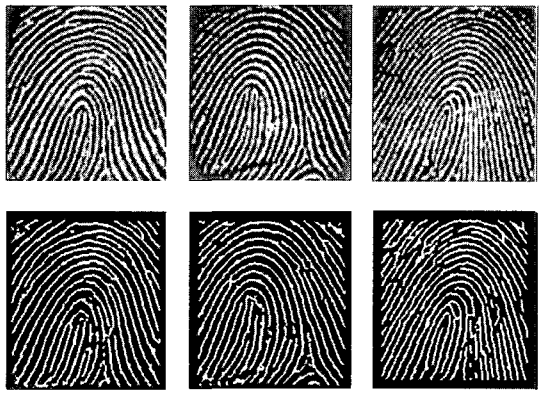
표 1. Xilinx Virtex V2000EFG680에서 구현시 사용된 주요 자원

Number of Slices	1,826 out of 19,200
Number of Slice Flip Flops	1,183 out of 38,400
Total Number 4 input LUTs	2,974 out of 38,400
Number used as LUTs	2,846
Number used as a route-thru	128
Number of Block RAMs	4 out of 160
Total equivalent gate count for design	97,054

구현된 지문추출 시스템의 수행시간을 측정하기 위하여 10개의 195×195 이미지를 사용하였다. 각 이미지에 대하여 발견된 끝점의 수, 발견된 분기점의 수, 하드웨어와 소프트웨어에서 소요된 시간이 표 2에 나타나 있다. 표 2를 보면 지문추출의 수행에 평균 6.5초가 소요되었으며 하드웨어와 소프트웨어에 평균적으로 각 각 2.0초와 4.4초가 소요되었음을 알 수 있다. 그림 9에 본 실험에 사용된 지문 이미지중 3 개의 입력 이미지와 각 입력에 대해 시스템에서 추출된 융선추적 결과를 나타내었다. 그림 9(a)은 실험 이미지중 선명한 이미지이며 그림 9(b)는 상처가 있고 주변

표 2. 지문추출 수행시간

이미지	0	1	2	3	4	5	6	7	8	9	평균
끝점	38	24	16	14	15	21	18	17	47	51	
분기점	4	6	3	10	7	13	9	13	10	12	
H/W 시간	2.0	2.0	2.3	2.1	1.9	1.9	1.9	1.9	2.1	2.2	2.0
S/W 시간	4.5	4.4	4.7	4.3	4.4	4.5	4.5	4.5	4.4	4.4	4.4
전체 시간	6.4	6.4	6.9	6.4	6.3	6.4	6.4	6.4	6.5	6.6	6.5



(a) (b) (c)

그림 9. 입력 이미지와 융선추적 결과

화질이 좋지 않은 이미지이며 그림 9(c)는 전체적 화질이 좋지 않은 이미지이다.

V. 결 론

본 논문에서는 융선추적 알고리즘을 이용한 지문 특징점 추출 시스템을 소규모 SoC에 구현하는 방법을 보였다. 융선추적 알고리즘은 기존의 특징점 추출 알고리즘에 비하여 적은 연산량을 사용하며 저화질의 지문영상에서도 우수한 성능을 보이는 알고리즘이다. 융선추적 알고리즘은 하드웨어의 효율을 높이기 위하여 수행 순서가 변형되었으며, 알고리즘의 각 기능 블록은 연산량, 하드웨어 비용, 그리고 효율성을 고려하여 하드웨어 또는 소프트웨어로 구현되었다.

구현된 융선추적을 이용한 지문 추출기는 ARM CPU와 AMBA 버스를 사용하는 SoC용 IP로 개발되어 여러가지 스마트카드용 SoC에서 사용이 가능하다. 또한, 지문 특징점 추출을 스마트카드 내부에서 수행하면 지문인식 전과정을 스마트카드내에서 처리하는 것이 가능해짐으로써, 사용자의 주요정보인 지문정보가 스마트카드밖으로 유출되는 위험을 감소시켜 프라이버시 이슈[10] 등을 해결할 수 있을 것으로 기대된다.

참 고 문 헌

[1] D. Maltoni, et al., Handbook of Fingerprint Recognition, Springer, 2003.

[2] 문대성, 길연희, 안도성, 반성범, 정용화, 정교일, "지문인증을 이용한 보안토큰시스템 구현," 한국정보보호학회 논문지, 제13권, 4호, pp.

63-70, 2003.

[3] D. Maio and D. Maltoni, "Direct Gray-Scale Minutiae Detection in Fingerprints," IEEE Trans. on Pattern Analysis Machine Intelligence, Vol. 19, No. 1, pp. 27-40, 1997.

[4] Q. Xaio and H. Raafat, "Fingerprint Image Postprocessing : a Combined Statistical and Structural Approach," Pattern Recognition, Vol. 24, pp. 985-992, 1991.

[5] B. Moayer and F. Ku, "A Tree System Approach for Fingerprint Pattern Recognition," IEEE Trans. on Pattern Analysis Machine Intelligence, Vol. 8, pp. 376-388, 1986.

[6] M. Kawagoe and A. Tojo, "Fingerprint Pattern Classification," Pattern Recognition, Vol. 17, No.3, pp. 295-303, 1984.

[7] L. O'Gorman and J. Mickerson, "An Approach to Fingerprint Filter Design," Pattern Recognition, Vol. 22, No. 1, pp. 29-38, 1989.

[8] M. Verna, A. Majumda, and B. Chatterjee, "Edge Detection in Fingerprints," Pattern Recognition, Vol. 20, pp. 513-523, 1991.

[9] J. Bresenham, "Algorithm for Computer Control of a Digital Plotter," IBM System J., Vol.4, No. 1, pp. 25-30, 1965.

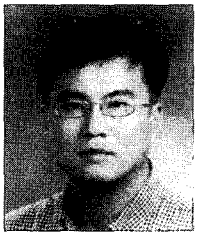
[10] S. Prabhakar, S. Pankanti, and A. Jain, "Biometric Recognition: Security and Privacy," IEEE Security and Privacy, pp. 33-42, 2003.

〈著者紹介〉

김 기 철 (Kichul Kim) 정회원
1982년 2월 : 서울대학교 전기공학과 학사
1984년 2월 : 서울대학교 전기공학과 석사
1991년 8월 : University of Southern California 전기공학과(컴퓨터공학 전공)박사
1984년 3월~1994년 2월 : 한국전자통신연구원 선임연구원
1994년 3월~현재 : 서울시립대학교 전자전기컴퓨터공학부 부교수
〈관심분야〉 SoC, 병렬처리, 멀티미디어, 생체정보

박 덕 수 (Deok-Soo Park)
2002년 2월 : 서울시립대학교 전자전기공학부 학사
2004년 2월 : 서울시립대학교 전자전기컴퓨터공학부 석사
〈관심분야〉 SoC, 생체정보, 멀티미디어

정 용 화 (Yong-Wha Chung) 종신회원
1984년 : 한양대학교 전자통신공학과 졸업
1986년 : 한양대학교 전자통신공학과 석사
1997년 : 미국 Univ. of Southern California 전기공학과(컴퓨터공학 전공) 박사
1986년~2003년 : 한국전자통신연구원 생체인식기술연구팀장
2003년~현재 : 고려대학교 컴퓨터정보학과 부교수
〈관심분야〉 생체인식, 정보보호, 생체정보 보호



반 성 범 (Sung-Bum Pan) 종신회원
1991년 : 서강대학교 전자공학과 졸업
1995년 : 서강대학교 전자공학과 석사
1999년 : 서강대학교 전자공학과 박사
1999년~현재 : 한국전자통신연구원 정보보호연구단 생체인식기술연구팀장
〈관심분야〉 생체인식, 영상처리, VLSI 신호처리