FPGA와 DSP를 이용한 실시간 차선 및 차량인식 시스템 구현

준회원 김 일 호*, 정회원 김 경 환*°

FPGA-DSP Based Implementation of Lane and Vehicle Detection

Il-Ho Kim* *Associate Member*, Gyeonghwan Kim*° *Regular Member*

요 약

본 논문에서는 FPGA(Field Programmable Gate Array)와 DSP(Digital Signal Processor)를 이용하는 실시간 차선 및 차량인식 시스템의 구현에 대하여 기술한다. 실시간 시스템의 구현을 위해서 FPGA와 DSP의 역할을 효율적으로 분할할 필요성이 있다. 시스템의 알고리즘을 특징요소 추출부분을 기준으로 분할하여 대량의 영상정보를 이용하여 소량의 특징요소를 추출하는 과정을 FPGA로 구현하고 추출된 특징요소를 사용하여 차선과 차량을 정의하고 추적하는 부분을 DSP에서 수행하게 하고, FPGA와 DSP의 효율적 연동을 위한 인터페이스 구성을 제안함으로써 실시간 처리가 가능한 시스템 구조를 제안한다. 실험 결과 제안한 실시간 차선 및 차량인식 시스템은 640×480 크기를 갖는 비디오 영상 입력에 대해 약 15 (frames/sec)로 동작하여 실시간 응용으로 충분함을 알 수 있다.

Key Words : lane detection, vechile detection, FPGA(field programmable gate arrays), DSP(Digital signal processor), real-time video processing

ABSTRACT

This paper presents an implementation scheme of real-time lane and vehicle detection system with FPGA and DSP. In this type of implementation, defining the functionality of each device in efficient manner is of crucial importance. The FPGA is in charge of extracting features from input image sequences in reduced form, and the features are provided to the DSP so that tracking lanes and vehicles are performed based on them. In addition, a way of seamless interconnection between those devices is presented. The experimental results show that the system is able to process at least 15 frames per second for video image sequences with size of 640 x 480.

Ⅰ. 서 론

최근 자동차의 자율 및 안전 주행을 위한 컴퓨터 비전(computer vision) 응용 중 주행 차선과 선행 차량을 검출하기위한 연구가 활발히 진행되고 있다[1,2]. 주행 차선과 선행 차량을 검출하기 위한 연구는 궁극적으로 운전자의 안전을 목표로 하기에 높은 정확성이 필요하고, 실제 주행하는 차량에 탑재가 가능해야 한다. 또한 실제 차량의 주행환경은 다양하기 때문에 악조건에서도 동작 가능한 FPGA나 DSP등의 전용 하드웨어(dedicated hardware)에서의 개발이 필요하다. 하지만, 고성능 시스템은 알고리즘의 복잡도가 높아져 단일 하드웨어만으로는 실시간 시스템 구현이 어렵다. 따라서 고성능 시스템의 구현을 위해 FPGA-DSP 등의 구조를 갖는 다중 하드웨어를 사용한 연구가 활발히 진행되고 있다.

* 서강대학교 전자공학과 MMI 연구실(gkim@sogang.ac.kr), (° : 교신저자)
논문번호 : KICS2011-10-496, 접수일자 : 2011년 10월 25일, 최종논문접수일자 : 2011년 12월 14일

지금까지 FPGA와 DSP를 함께 활용한 컴퓨터 비전 응용 시스템 구현에 관한 연구를 살펴보면, Z. Liu et al.[3]은 컬러 영상 퓨전 시스템(color image fusion system)의 구현을 위해 FPGA로 영상 취득 및 DSP로의 영상 전송을 담당하고 알고리즘의 수행은 DSP에서 전담하는 구조를 제시하였고, S. S. Kim와 S. Jung[4]은 뉴럴 네트워크를 DSP에서 구현하고 단순 로직 유닛(logic unit)으로써 FPGA를 사용하였다. 이운근 등[5]은 영상 입력부와 소벨 에지 검출을 FPGA로 구성하고 나머지 부분을 DSP에서 수행하는 구조를 제시하였다. 앞서 언급한 논문에서 제안된 구조는 영상 입력부로부터 간단한 전처리(pre-processing) 부분 또는 단순 로직으로써 FPGA로 구성하였기 때문에 DSP의 성능에 주로 의존하게 된다. 따라서 복잡한 알고리즘으로 구성된 시스템을 구현 시 DSP에서 실시간 처리가 힘든 구조가 된다. J. Zhang et al.[6] 와 L. Yan et al.[7] 은 FPGA-DSP 구조의 한계를 극복하기 위해 FPGA와 다수의 DSP를 병렬적으로 배치하여 실시간 시스템 구현을 이루었지만, 다수의 DSP를 사용함으로써 비용의 증가를 초래하였다.

본 연구에서는 실시간 차선 및 차량인식 시스템을 구현하기 위해 FPGA-DSP 구조를 사용하였다. 기존 FPGA-DSP 구조의 한계를 극복하기 위해 FPGA와 다수의 DSP를 사용하는 구조 대신, FPGA와 DSP의 역할을 효율적으로 분담하여 실시간 처리가 가능한 구조를 제안한다. 차선 및 차량인식 시스템은 일반적으로 차선과 차량을 검출하기 위해 입력 영상으로부터 차선과 차량을 대표할 수 있는 특징요소를 추출하고 분석을 통해 차선과 차량을 정의하게 된다. 영상 입력부와 대량의 영상정보로부터 소량의 특징 정보를 재생산하는 과정을 FPGA로 구현하고, 소량의 특징 정보를 이용해 차선과 차량의 정의 및 검증 과정과 영상 출력부를 DSP에서 처리하도록 하여 기존 FPGA-DSP 구조의 한계를 극복하였다. FPGA-DSP 구조를 효율적으로 사용하기 위해서는 개별 하드웨어에서의 처리뿐만 아니라 두 하드웨어 사이의 효율적인 인터페이스(interface) 구성이 필요하다. FPGA와 DSP를 효율적으로 연동시키기 위해서 FPGA에서 처리된 특징 정보와 입력 영상을 DSP로 전송할 수 있어야 하고, 두 하드웨어의 동작을 제어할 수 있어야 한다. DSP에는 코어(core)외에도 주변장치를 함께 제공하기 때문에 DSP의 주변장치를 이용하여 목적에 맞는 인터페이스를 구성할 수 있다. 따라서 본 논문에서는 DSP의 주변장치를 활용하여 데이터 전송 및 두 하드웨어의 동작을 제어할 수 있는 인터페이스를 구성하였다.

본 논문의 구성은 Ⅱ장에서 시스템 구현에 사용된 알고리즘 소개 및 개별 하드웨어 구현에 대해 설명하고, Ⅲ장에서는 FPGA와 DSP사이의 인터페이스 구성에 대해 설명한다. Ⅳ장에서는 실험결과를 분석하고, 마지막으로Ⅴ장에서 결론을 맺는다.

Ⅱ. 차선·차량인식 알고리즘의 FPGA 및 DSP 구현

본 절에서는 그림 1에 나타낸 전체 시스템 구조 중 특징 추출부의 FPGA 구현과 인식 및 검증 부의 DSP 구현에 관하여 소개한다. 시스템 구현에 사용된 FPGA는 Xilinx사의 Spartan-3A DSP를 사용하였고, DSP는 Texas Instruments 사의 TMS320DM6437를 사용하였다.

2.1 특징 추출부의 FPGA 구현

특징 추출부는 대체로 규칙적이고 국부적이며, 대량의 입력 영상정보에 대해 반복적으로 처리하는 특성을 가진다. 이러한 특징 추출과정은 물리적으로 다수 존재하는 연산유닛을 목적에 맞게 조합하여 병렬

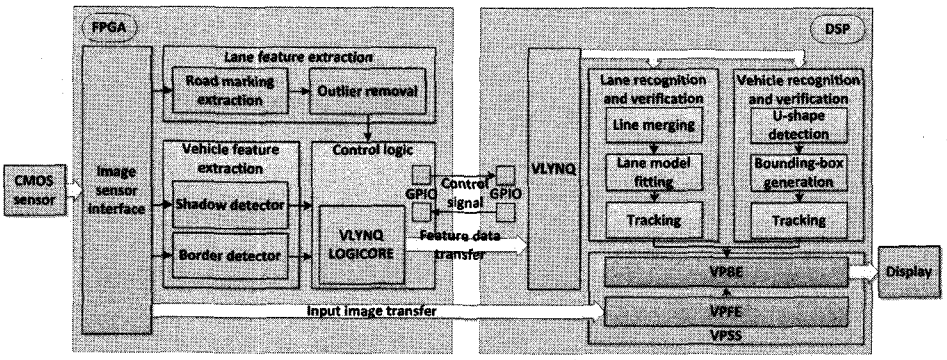


그림 1. 전체 시스템 구조

적으로 데이터를 처리하여 대량의 데이터에 대한 단순 연산에 강점을 보이는 FPGA로 구현하는 것이 적합하다. 따라서 특징 추출부를 FPGA로 구현하여 전체 시스템의 실시간 동작을 보장할 수 있도록 하였다.

2.1.1 차선 특징 추출(Lane feature extraction)

차선 특징 추출과정은 차선표식 추출 및 이상치 제거과정으로 구성된다. 각 과정에서의 알고리즘 및 구현된 FPGA 구조에 대해 설명한다.

(1) *차선표식 추출(Road marking extration)*

차선표식은 각 나라마다 폭과 색깔이 다르지만, 일반적으로 일정한 차선폭을 가지고 있으며 주변도로에 비해 밝기 값이 크고 경계가 생긴다는 특징을 가진다. 따라서 에지 검출(edge detection)을 기반으로 차선표식의 경계를 찾고, 주변도로보다 상대적으로 밝은 영역[8]을 검출한다.

FPGA로 구현된 차선 특징 추출부의 구성은 그림 2(a)와 같다. 차선표식 추출 과정은 픽셀 그룹 처리(pixel group processing)로써 윈도우 연산을 기반으로 구성된다. 윈도우 연산을 일반 MPU(micro processor unit)로 구현하게 되면 윈도우 내의 각 픽셀 값을 메모리로부터 순차적으로 읽어와 연산을 수행하고, 다시 메모리에 쓰는 작업을 수행해야 하므로 처리결과를 얻기까지 많은 클락 사이클(clock cycle)이 소요된다. 반면, FPGA의 경우 그림 2(b)와 같이 윈도우의 크기와 같은 레지스터 배열과 각 레지스터별로 연산 모듈을 배치해 최초 지연(delay)후에는 매 클락 사이클마다 여러 픽셀에 대하여 동시 참조가 가능하기 때문에 여러 픽셀에 빈번히 접근해야 하는 픽셀 그룹 처리에 높은 성능을 보여준다[9]. 우선, 에지 기반 검출(edge based detection) 과정은 윈도우 연산을 통해 얻은 주변 화소와 소벨 연산자(Sobel operator)와의 연산을 통해 에지를 검출하고, 검출된 에지사이의 폭을 계산하여 LUT(look-up table)로 구성해 놓은 차선폭과의 비교 연산을 통해 차선표식 후보를 검출하게 된다. 이때, 차선폭에 대한 최소·최댓값은 세계 좌표계(world coordinate)와 영상 좌표계(image coordinate)와의 관계가 고정적인 것으로 가정하여 미리 계산한 결과를 LUT로 구성하였다. 밝기 기반 검출(brightness based detection)은 주변화소 정보를 이용하여 어두운 영역 사이에 존재하는 밝은 영역의 너비를 계산하고, 에지 기반 검출과 마찬가지로 비교연산을 통해 차선표식 후보지의 시작 좌표와 끝 좌표를 출력한다.

(2) *이상치 제거(Outlier removal)*

FPGA로 구현된 이상치 제거 모듈의 구성은 그림 3(a)와 같다. 에지, 밝기 기반으로 구한 차선표식 후보지는 주변 배경으로 인해 차선뿐 아니라 노이즈 영역까지 추출하게 된다. 연결 요소 분석(connected component analysis, CCA)을 적용하여 종 방향의 연결성을 판단하여 이상치를 제거한다. 연결 요소 분석은 두 번의 스캔(scan)과정이 필요한데, 첫 번째 스캔에서 이웃화소(A,B,C,D)의 밝기 값과 현재 좌표의 화소(X)의 밝기 값을 비교하여 X에 라벨을 부여한 후(그림 3(b) 참고) 메모리에 저장한다. 하지만, 첫 번째 스캔만으로는 정확한 라벨을 부여할 수 없으므로 가능한 라벨을 모두 병합 테이블(merger table)에 구성한다. 저장한 라벨을 스캔하면서 병합 테이블에 배치된 라벨들을 비교하여 최종 라벨을 부여하게 되는데, FPGA로 구현 시 라벨의 수에 따라 상당한 크기의 메모리가 필요하게 된다[10]. 하지만, 이상치 제거를 위한 연결 요소 분석의 결과가 전체 영상의 연결된 모든 요소를 찾을 필요가 없기 때문에 부분적으로 연결 요소 분석을 적용하여 메모리 사용을 최소화 하였다. 그림 3(c)와 같이 윈도우 버퍼를 통해 얻은 주변화소 정보

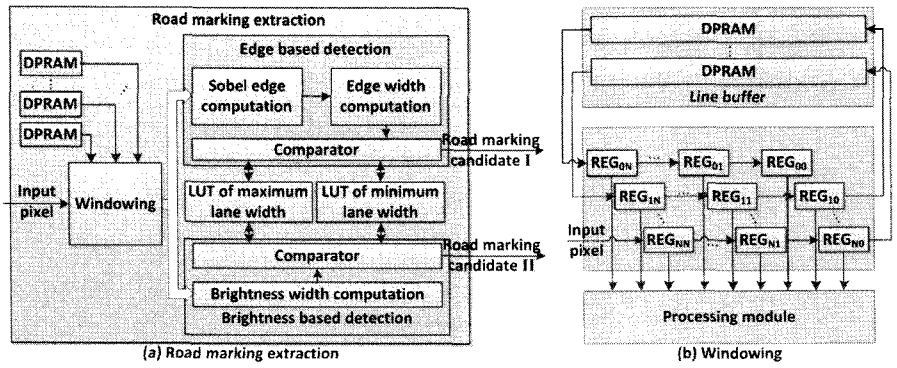


그림 2. 차선표식 추출 모듈의 구성 : (a) 차선표식 추출 (b) 윈도우 연산

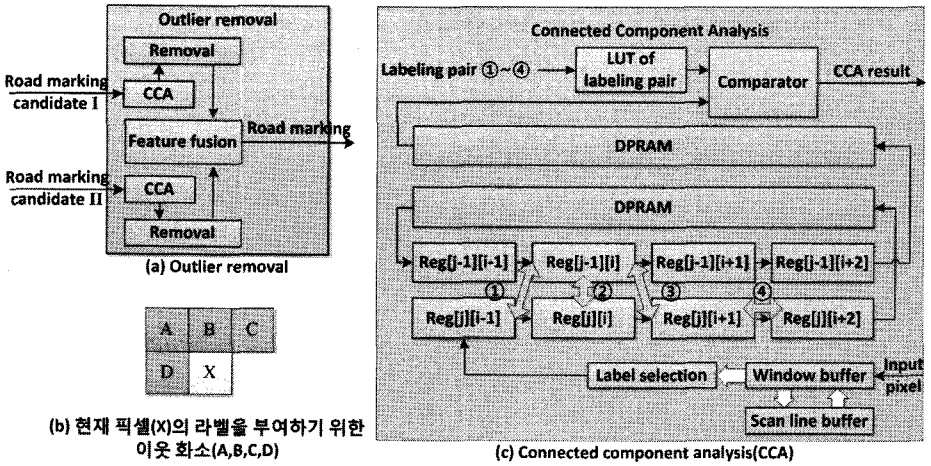


그림 3. 이상치 제거 : (a) 이상치 제거 모듈 구성, (b) 현재 픽셀(X)의 라벨을 부여하기 위한 이웃 화소(A,B,C,D), (c) 연결요소분석 모듈 구성

를 이용하여 라벨을 부여하고 레지스터를 거쳐 DPRAM에 저장된다. 그림 3과 같이 첫 번째 스캔을 통해 얻은 라벨 정보는 연결된 요소임에도 다른 라벨로 부여되는 경우가 존재하기에 현재 라인과 이전 라인에 존재하는 ①~④의 라벨을 비교하여 갱신이 필요한 라벨에 대하여 현재 라벨 정보와 변경될 라벨 정보를 함께 LUT에 저장한다. 저장된 라벨 쌍에 대한 정보는 5 라인 지연 후에 갱신하여 출력된다. 에지, 밝기 기반으로 구한 차선표식 후보지를 부분적 연결 요소 분석을 통해 노이즈 성분을 제거한 결과를 병합하여 최종 차선표식의 양 경계 좌표와 중앙 점의 영상 좌표를 출력한다.

2.1.2 차량 특징 추출(Vehicle feature extraction)

인공구조물인 차량의 특징을 추출하기 위해 경계와 그림자 영역을 추출한다. 경계추출은 수평·수직방향의 경계추출로 구성되고, 그림자 영역은 에지와 밝기를 통해 추출한다. 각 과정에서 사용된 알고리즘과 구현에 사용된 FPGA구조에 대해 설명한다.

(1) *경계 추출(Border detection)*

그림 4(a)는 FPGA로 구현된 경계 추출 모듈의 구조이다. 차량의 후방 창문, 범퍼 등에 의해 생기는 수평·수직 경계 성분을 찾기 위해 차선표식 검출 과정에서 사용되었던 소벨 에지를 사용한다. 수평방향의 경계를 추출하기 위해 수평 에지 성분들의 간격을 구하고 차량의 최소·최대 폭을 계산하여 수평간격이 차량의 예상 폭에 포함되는 에지 성분들을 제외한 나머지를 제거한다. 이때, 차선폭과 마찬가지로 차량의 예상 폭에 대한 정보를 미리 계산하여 LUT에 구성하였다. 수직방향의 경계를 찾기 위해서는 에지 정보를

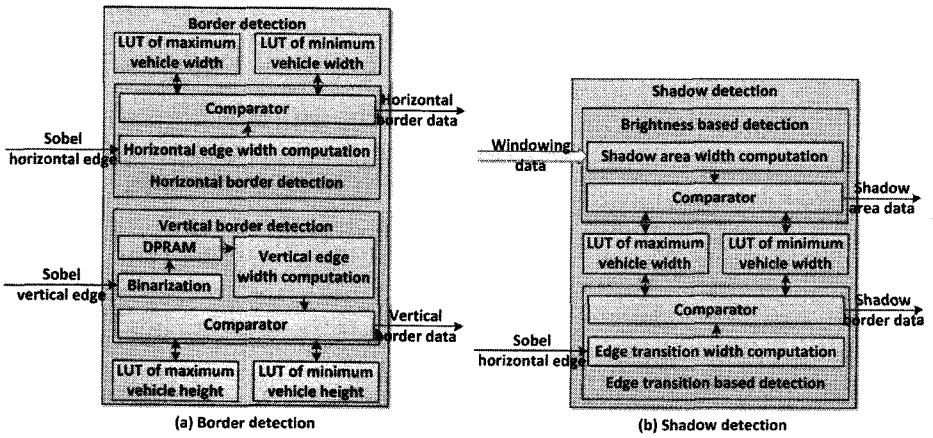


그림 4. 차량 특징추출 모듈의 구조 : (a) 경계 추출, (b) 그림자 추출

세로 방향으로 분석이 필요하기 때문에, 현재 프레임의 수직 에지 성분을 1bit로 이진화(binarization)하여 DPRAM에 저장한다. 저장된 에지 정보를 vertical sync 구간에 읽어와 차량의 예상 높이 정보를 구성해 놓은 LUT와의 비교 연산을 통해 수직방향의 경계를 추출한다. 추출한 수평 · 수직방향 경계의 시작 좌표와 끝 좌표를 출력한다.

(2) *그림자 추출(Shadow detection)*

차량의 범퍼 아래 부분에 생기는 그림자 영역은 주변지역보다 어둡기도 하지만 경계 성분이 강하다는 특징을 지니고 있다. 하지만 이러한 그림자 영역은 차량의 범퍼 아래 부분 이외에도 주변 배경으로 인해 많은 지역에서 나타나기도 한다. 이를 구분하기 위해 차량의 예상 폭에 포함되는 영역만을 추출하였다. 그림 4(b)와 같이 윈도우 연산을 통해 얻은 주변 지역의 픽셀 정보를 이용하여 어두운 지역을찾고, 에지변화 폭과 LUT에 구성해 놓은 차량의 예상 폭과의 비교를 통해 그림자 영역을 추출하게 된다. 그림자 영역의 정보는 후에 수평 · 수직방향의 경계성분과 함께 차량을 정의하기 위한 정보로써 사용되게 된다. 에지변화, 밝기 기반으로 구한 그림자 영역의 결과의 시작 좌표와 끝 좌표를 출력한다.

2.2 인식 및 검증부의 DSP 구현

FPGA에서 처리된 특징정보를 이용하여 차선과 차량을 정의하고 추적하는 과정을 DSP가 담당하게 된다. 전체 시스템의 실시간 동작을 위해 TMS320DM6437에서의 최적화 작업이 필요하다. 본 연구에서는 다양한 최적화 방법[11] 중 메모리의 효율적 사용과 코드 최적화를 통하여 실시간 동작을 보장하였다. TMS320DM6437의 경우 32KB의 Level 1 Program(L1P) 캐쉬, 80 KB의 Level 1 Data(L1D) 캐쉬, 128KB의 L2 통합 메모리를 지원한다[12]. C64x+의 L1D 캐쉬는 L2D 메모리와 외부 메모리를 모두 인출(fetch)할 수 있으며, L2D 캐쉬는 외부 메모리만을 인출할 수 있다. L1D와 L2D를 모두 메모리로 사용할 경우 가장 빠른 속도를 보이지만, 용량의 한계로 인하여 FPGA로 추출된 특징 요소들을 모두 저장할 수 없게 된다. 따라서 내부 메모리에 비해 매우 느린 속도를 가진 외부 메모리에 명령어와 데이터를 저장할 수 밖에 없게 되고, 이는 알고리즘의 처리시간을 증가시키는 요인이 된다. 이러한 문제를 해결하기 위해 L1D와 L2D를 모두 캐쉬화하여 사용하는 것이 효율적인데, C64x+ 는 MAR(memory attribute register)[13]를 이용하여 외부메모리를 L2D 캐쉬가 인출할 수 있도록 cacheable 상태로 변경할 수 있다. CPU가 외부메모리에 접근할 때 L2D 캐쉬를 이용하여 라인 크기만큼의 데이터를 인출할 수 있기 때문에 외부 메모리를 직접 접근하기 전에 캐쉬에 미리 인출한 데이터를 가져오므로 알고리즘의 처리 속도를 높일 수 있게 된다. 차선 및 차량인식의 후처리 부분은 보다 정확한 계산을 위해 많은 실수 연산을 필요로 한다. 그러나 C64x+ 는 부동소수점(floating point) 연산을 지원하지 않으므로, 실수 연산은 보통 S/W 에뮬레이션으로 구현되는데, 많은 사이클을 필요로 한다. 이러한 문제를 해결하기 위해서 IQmath 라이브러리를 적용하여 부동소수점 연산이 이루어지는 알고리즘들을 고정소수점(fixed point) 연산으로 변환하였다[14]. 반면, 높은 정확도를 요구하는 알고리즘의 경우에는 고정소수점 변환으로 인해 나타나는 정밀도의 한계로 인해 최종 결과에 영향을 미치는 경우가 발생하게 된다. 고정 소수점 연산 정밀도의 한계를 극복하기 위해, 부동소수점 연산 시 intrinsic 레벨까지 최적화된 함수를 제공하는 C fast RTS 라이브러리를 사용하였다. 본 절에서는 그림 1에서 차선과 차량 인식 및 검증 모듈을 구성하는 기능 단위에 대해서 메모리의 효율적 사용과 코드 최적화를 중심으로 설명한다.

2.2.1 차선 인식 및 검증(Lane recognition and verification)

차선 인식 및 검증 과정은 차선 특징정보를 이용하여 차선을 인식하고 추적하는 과정을 진행하게 된다. 표 1은 차선 인식 및 검증 부분의 알고리즘별 프레임당 평균 처리시간을 메모리의 효율적 사용 및 코드 최적화의 적용에 따라 미치는 영향을 보여준다.

표 1. 효율적인 메모리 사용과 코드 최적화에 따른 차선 인식 및 검증 부분의 알고리즘별 처리시간 비교

차선인식 알고리즘	최적화 전(ms)	효율적인 메모리 사용(ms)	코드 최적화(ms)
Line merging	463.5	20.0	9.7
Lane model fitting	968.7	342.2	14.3
Tracking	120.5	34.8	5.8

(1) *차선표식 병합(line merging)*

FPGA에서 처리된 차선표식 결과는 주행 차선뿐 아니라 주변 차선까지 추출되게 된다. 주행 차선을 검출하는 것이 목적이므로 차선표식을 군집화(clustering)[8]하여 주행 차선과 주변 차선을 선별하게 된다. 차선표

식의 중앙 점과 경계 점을 이용하여 방향성, 길이, 위치를 고려하여 군집화를 진행한다. 차선표식 군집간의 방향성과 영상평면에서의 거리, 위치를 고려하여 주행차선의 군집만을 선택하고 나머지를 제거한다. 차선표식 정보를 군집화하기 위해 복잡한 연산보다는 잦은 메모리 접근이 필요로 하기에 효율적인 메모리 사용을 통해 처리시간을 크게 감소시킬 수 있었다.

(2) *최소자승법(least square method)을 이용한 근사화(lane model fitting)*

직선 및 곡선으로 이루어진 차선을 정의하기 위해 2차 곡선으로 가정하였고, 앞서 구한 차선 후보지 데이터를 최소자승법을 적용하여 차선의 방정식을 구할 수 있었다. 앞서 주행차선을 찾기 위해 군집화 과정을 하였으므로 상대적으로 신뢰도 높은 결과를 얻을 수 있게 된다. 최소 자승법을 이용한 근사화 과정은 실수 연산이 주로 포함되어 있어 고정소수점 프로세서인 TMS320DM6437 상에서 가장 많은 처리시간을 필요로 하는 기능 단위이다. IQmath 라이브러리를 적용하여 고정소수점 연산으로 변환하는 작업과 함께 높은 정확도를 유지하기 위해 부분적으로 C fast RTS 라이브러리를 사용하였다. 따라서 효율적인 메모리 사용만으로는 높은 감소율을 얻을 수 없었지만 코드 최적화를 통해 처리시간이 크게 감소되었음을 표 1을 통해 확인할 수 있다.

(3) *차선 추적(Tracking)*

차선 추적과정[8]은 차선인식 결과를 누적하고, 다음 프레임에서의 차선표식 군집화에 사용될 판단조건으로써 사용된다. 주행차선은 연속적이고 영상평면에서의 이동이 크지 않다는 가정을 통해 군집화 과정에 신뢰도 높은 차선 분류를 할 수 있도록 한다. 또한 인식과정이 실패하였을 경우 누적된 정보를 이용하여 예상 차선 위치를 출력함으로써 지속적인 인식이 될 수 있게 한다. 추적과정에는 결과의 누적과 비교를 위해 메모리에 자주 접근해야 하고, 현재 프레임과 이전 프레임의 차선의 관계를 계산하기 위한 산술 연산도 포함되어 있기 때문에 효율적인 메모리 사용과 코드 최적화를 통해 처리시간이 크게 감소되었다.

2.2.2 차량 인식 및 검증(Vehicle recognition and verification)

차량 인식 및 검증은 FPGA로부터 추출된 수평·수직 방향의 경계 성분과 그림자 성분을 이용하여 차량을 정의하고 추적하는 과정을 진행하게 된다. 표 2는 차량 인식 및 검증 부분의 알고리즘별 프레임 당 평균 처리시간을 메모리의 효율적 사용 및 코드 최적화의 적용에 따라 미치는 영향을 보여준다.

표 2. 효율적인 메모리 사용과 코드 최적화에 따른 차량 인식 및 검증 부분의 알고리즘별 처리시간 비교

차량인식 알고리즘	최적화 전(ms)	효율적인 메모리 사용(ms)	코드 최적화(ms)
U-shape detection	901.5	22.8	10.9
Bounding-box generation	198.0	32.1	8.5
Tracking	2158.1	57.0	3.8

(1) *"U"형태 추출(U-shape detection)*

경계 추출과정을 수행하여 얻은 수평·수직경계를 병합하여 차량의 후보지를 찾는 과정이 진행된다. 차량의 뒷모습은 "U"형태를 지니고 있다[15]. 하나의 수평경계와 양 끝에서 만나는 두 수직경계가 존재하는 영역을 "U"형태를 지니는 차량 후보지로 선정하게 된다. "U" 형태 추출은 하나의 수평경계를 이용하여 다수의 수직경계를 비교해야하므로 FPGA에서 처리된 입력 데이터를 반복하여 접근하므로 효율적 메모리 사용을 통해 처리시간을 크게 감소시킬 수 있었다.

(2) *바운딩박스[16] 생성(Bounding-box generation)*

"U"형태를 지니는 영역이 차량이 맞는지를 판단하기 위해 영역내의 히스토그램 분석을 통하여 대칭성을 지니는지를 판단한다. "U"형태의 영역 안에서 히스토그램 분석을 해 보면, 해당 영역에 차량이 존재할 경우 대칭성을 가지며 히스토그램의 분포가 비슷하지만 차량이 존재하지 않을 경우 분포가 달라진다. "U" 형태의 영역을 바운딩 박스로 지정하고 영역내의 히스토그램 분석을 통해 차량 후보지를 선별하게 되고, 그림자 영역과의 비교를 통해 최종 차량을 정의하게 된다. 바운딩 박스 생성의 경우에는 히스토그램 분석을 위한 연산 및 그림자 영역과의 비교 연산이 포함되어 있어 효율적 메모리 사용과 코드 최적화를 통해 처리 시간이 크게 감소되었다.

(3) *차량 추적(Tracking)*

차량 추적과정은 생성된 바운딩 박스들의 결과를 누적하여 인식된 결과를 관리한다. 인식에 실패하였을 경우 누적된 결과를 분석하여 바운딩 박스의 움직임 등을 판단하여 인식에 실패함을 인지하였을 경우 차량을 지속적으로 검출할 수 있도록 해준다. 바운딩 박스의 누적관리로 인해 메모리에 잦은 접근이 필요하여 상대적으로 많은 처리시간을 필요로 하기에 효율적인 메모리 사용을 통해 처리시간을 크게 감소시킬

수 있었다.

Ⅲ. FPGA-DSP 인터페이스

FPGA와 DSP를 효율적으로 사용하기 위해 그림 1에서와 같이 데이터 전송과 두 하드웨어의 동작을 제어하기 위한 부분이 필요하다. 제안하는 시스템의 성능은 두 하드웨어간의 연결의 치밀성과 효율성에 따라 크게 좌우될 수 있다. 따라서 본 연구에서는 TMS320DM6437이 제공하는 주변장치와 기능을 적극적으로 활용하여 이러한 목적을 성취하고자 하였다. VLYNQ를 사용하여 데이터 전송을 하고 GPIO를 활용하여 두 하드웨어를 연동한다. FPGA에선 DSP의 주변장치들을 사용할 수 있도록 인터페이스에 관련된 설계가 필요하다. FPGA에서 병렬로 처리된 특징요소를 DSP에 전송할 수 있도록 전송관리가 필요하고 처리가 완료되었음을 DSP로 알려줄 수 있어야 한다. 또한 DSP가 FPGA동작을 인터럽트를 이용하여 제어함으로써 전체 시스템의 동기화를 이룰 수 있게 된다.

3.1 데이터 전송(data transfer)

구현에 사용된 DSP인 TMS320DM6437은 데이터 전송을 위해 VPSS(Video Processing Sub-System), VLYNQ, EMIF(External Memory Interface)등의 주변장치를 제공한다. VPSS는 외부로 들어오는 비디오 신호 입력을 처리하는 VPFE(Video Processing Front End)와 비디오 신호를 외부 디스플레이 장치로 출력하는 VPBE(Video Processing Back End)가 포함된 시스템이다. EMIF는 외부장치의 메모리를 사용할 수 있도록 인터페이스를 제공한다. 마지막으로 VLYNQ는 외부장치와의 직렬통신 인터페이스를 제공한다[12]. DSP로 입력 영상을 전송하기 위해 VPFE를 사용하고, 처리된 특징 정보를 전송하기 위해 VLYNQ를 사용하고, 마지막으로 VPBE를 통해 최종 결과를 출력하였다.

VLYNQ를 이용하여 FPGA에서 처리된 특징요소를 구분하고 전송의 효율을 높이기 위해 그림 5와 같이 32bit 크기의 패킷(packet)으로 데이터를 가공하여 내부 로직 FIFO에 저장하였다. 패킷의 상위 3bit에는 각 특징요소별로 ID 코드를 부여하여 구성하였고, 개별 특징 결과는 영상에서의 위치 좌표로 출력되기 때문에 표 3과 같이 나머지 bit를 구성하였다. 현재 프레임의 처리가 모두 종료되었음을 ID코드를 따로 부여하여 DSP에서 인식할 수 있도록 하였다. 최종적으로 VLYNQ로 전송받은 데이터는 DSP에서 디코딩(decoding) 작업을 수행하여 개별 특징 결과를 지정된 메모리 영역에 저장하였다.

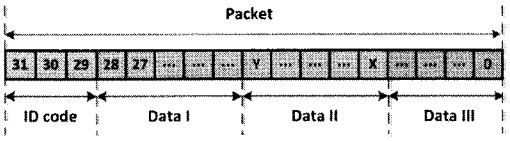


그림 5. 패킷의 구조

3.2 동기화(synchronization)

서로 다른 두 하드웨어 사이에 프레임별 동기를 맞추기 위해 인터럽트를 사용한다. DSP에서 FPGA의 처리결과를 가져갈 것과, DSP의 처리가 종료되었음을 FPGA에 알려주는 인터럽트를 사용하여 두 하드웨어의 동작을 동기화 시켰다. 인터럽트의 발생은 태스크(task)의 문맥전환을 수반하기 때문에, 인터럽트의 빈번한 발생에 의한 처리시간의 증가가 시스템 동작에 부담이 가지 않도록 인터럽트 방식을 정해야한다. 그림 6은 패킷과 인터럽트 생성 모듈의 구조를 보여준다. 처리가 완료된 전처리 결과는 각각 패킷 형태로 가공된 후 개별 DPRAM에 저장된다. 각 DPRAM에 저장된 데이터를 ID코드의 순서대로 읽어와 VLYNQ

표 3. ID code 및 data 구성

Module	ID code	Data I	DataII	Data III
Lane feature extraction	0	중앙점 x좌표(10bit)	좌우 경계 x좌표(10bit)	y좌표(9bit)
Horizontal border based shadow detection	1	경계 시작점의 x좌표(10bit)	경계 끝점의 x좌표(10bit)	y좌표(9bit)
Edge transition based shadow detection	2	그림자 경계 시작점의 x좌표(10bit)	그림자 경계 끝점의 x좌표(10bit)	y좌표(9bit)
Brightness based shadow detection	3	그림자 영역 시작점의 x좌표(10bit)	그림자 영역 끝점의 x좌표(10bit)	y좌표(9bit)
Vertical border detection	4	경계 시작점의 y좌표(9bit)	경계 끝점의 y좌표(9bit)	y좌표(9bit)
End of frame processing	5	0	0	0

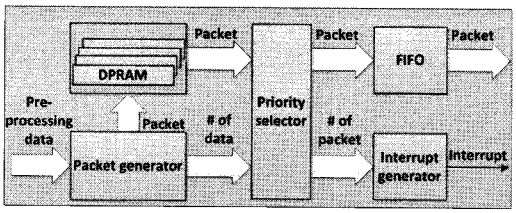


그림 6. 패킷과 인터럽트 생성 모듈의 구조

내부 FIFO에 순차적으로 저장한다. FIFO에 일정 패킷 크기만큼 저장되었을 때, 인터럽트를 발생시켜서 DSP에서 패킷크기만큼의 데이터를 가져갈 것을 요청하도록 하였다. 패킷 사이즈가 256일 때 인터럽트는 프레임 당 평균 6회가 발생하며 데이터 전송 및 디코딩에 걸리는 시간은 FPGA가 DSP에 발생시킨 인터럽트 횟수에 비례한다. DSP에서 FPGA의 VLYNQ 내부 로직 FIFO에서 256개의 패킷을 읽어오는데 소요되는 시간은 약 0.5ms가 소요된다.

3.3 스케줄링(scheduling)

차선 및 차량인식 시스템의 FPGA와 DSP상에서의 전체 동작 시나리오에 대해 정리하면 그림 7과 같다. 그림 7에 표시된 번호는 시스템의 동작 순서를 의미하며 각각을 설명하면 다음과 같다.

① DSP는 FPGA와 연결된 GPIO핀을 통하여 FPGA의 전처리 모듈을 수행시킨다.

② FPGA는 DSP와 연결된 GPIO핀을 통하여 명령을 인지하고 전처리를 처리하며, 처리한 데이터는 VLYNQ 내부 로직 FIFO에 저장한다. FIFO에 저장한 데이터의 개수가 일정 패킷 사이즈가 될 때마다 DSP와 출력으로 연결된 GPIO핀을 통해 DSP에게 일정 패킷 사이즈가 되었음을 알린다.

③ DSP에 FPGA와 입력으로 연결된 GPIO핀으로부터 edge detection logic을 통해 인터럽트를 발생시킨다.

④ 인터럽트가 발생하면, 인터럽트 서비스 루틴에서 VLYNQ를 통해 FPGA가 처리한 데이터를 읽어온다.

⑤ FPGA의 처리가 끝났을 때, 디코딩 작업을 통해 해당 메모리 영역에 각 특징요소를 저장하고 후처리를 진행한다.

⑥ 후처리 모듈의 수행 결과를 VPBE를 통해 출력한다.

Ⅳ. 실험 결과

본 연구에서 제안한 차선 및 차량인식 시스템의 구현결과에 대한 성능을 검증하기 위해 AVNET에서 제작한 AES-SP3ADSP-DVCI-G 보드[17]를 사용하였다. AES-SP3ADSP-DVCI-G 보드는 FPGA로Spartan-3A DSP가 탑재되어 있고, DSP로 TMS320DM6437이 탑재되어 있다. 설계한 하드웨어 모듈의 동작 및 성능을 검증하기 위하여 FPGA 구현 모듈은 Xilinx 사의 ISE를 사용하여 합성을 하였고, DSP로 구현 모듈은 CCS를 이용하여 구현하였다. 실험은 총 5531 장의 데이터 세트(data set)에 대해 진행하였다. 데이터 세트에 대해 차선인식은 91.4%의 인식 성공률과, 오 검출 4.4%, 미 검출 5.2%의 성능을 보였다. 그리고 차량인식은 90.4%의 인식 성공률과, 오검출 4.6%, 미검출 5%의 성능을 보였다. 그림 8에서 볼 수 있듯이 연속된 프레임에 대하여 소프트웨어 상에서 구현한 알고리즘의 결과와 하드웨어 상에서 구현한 결과가 동일함을 확인할 수 있다.

차선 및 차량인식 시스템 구현에 사용된 FPGA 자원 총 사용량 대비 주요 모듈별 사용 자원과 사용 비율을 표 4에 정리하였다. 62.5MHz의 주파수로 동작하는 FPGA를 이용하여 테스트한 결과 640×480 영상에 대해 화소 클럭이 27MHz인 CMOS센서 입력에 대해 FPGA 구현 부분이 30 (frame/sec) 의 full-frame rate로 동작함을 확인하였다.

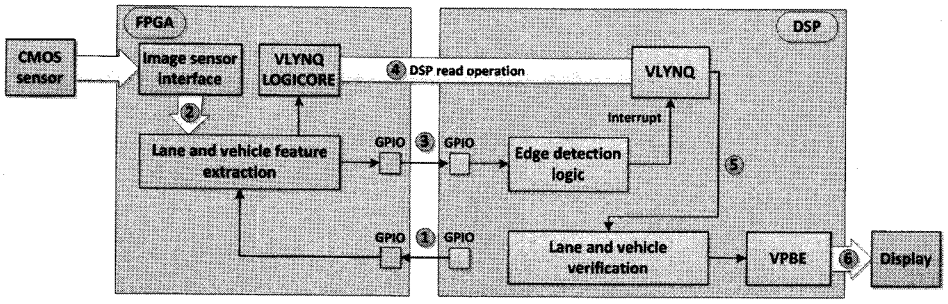


그림 7. 차선 및 차량인식 시스템 동작 스케줄링

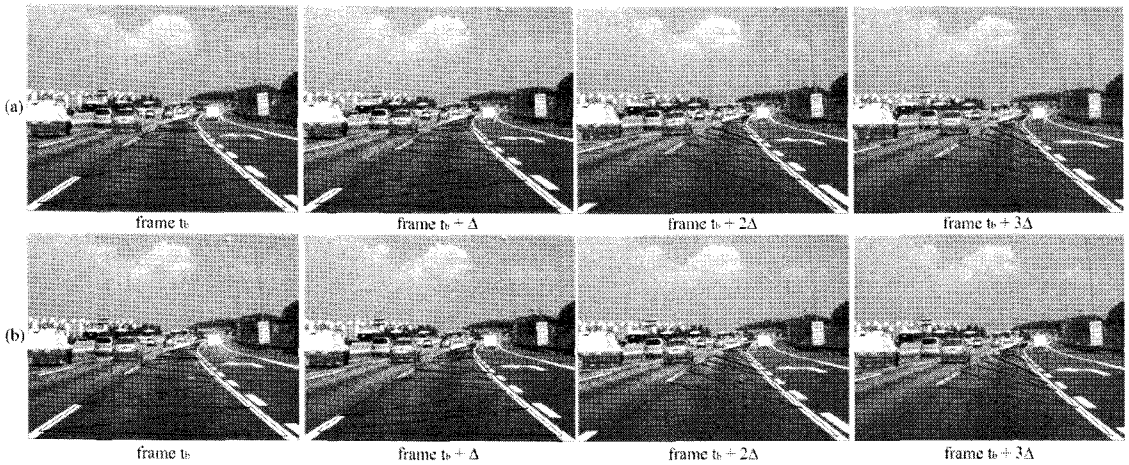


그림 8. 구현 결과 비교 : (a) 소프트웨어 구현 결과, (b) 하드웨어 구현 결과

표 4. 차선 및 차량인식 시스템 FPGA 자원 사용률

Module	Slice Registers(총 33,280)	Slice LUTs(총 33,280)	BlockRAM FIFO(총 84)
Road marking extraction	3,403(20.47%)	3,831(22.64%)	7(17.07%)
Outlier removal	8048(48.41%)	4,790(28.31%)	3(7.32%)
Sobel edge detection	426(2.56%)	323(1.91%)	2(4.88%)
Shadow detector	434(2.61%)	670(3.96%)	3(7.32%)
Border detector	4,118(24.77%)	6,936(40.99%)	21(51.22%)
Packet and interrupt generator	195(1.17%)	372(2.20%)	5(12.20%)
총 사용량	16,624	16,922	41

FPGA를 단순 입력 영상처리 및 전처리로 사용하고, DSP에서 알고리즘을 처리하는 구조로 시스템을 구현한다면, 표 5에서 확인할 수 있듯이 통합 146.2ms가 소요되어 실시간 처리에 부족한 성능을 보이게 된다. 하지만, 본 연구에서 제안한 특징요소 추출단위 분할의 경우 통합 55.9ms로 61.76%의 처리시간 감소를 얻을 수 있다. FPGA에서 처리하는 특징추출부분은 on-the-fly로 처리되기 때문에 전체 시스템의 처리시간은 FPGA가 처리한 데이터를 전송받아 디코딩하여 인식 및 검증 부분을 수행하는 처리시간만 포함되게 된다. 전체 시스템의 처리시간은 약 15 (frame/sec)의 처리속도로 동작하여 실시간 처리가 가능함을 볼 수 있다.

표 5. 단일 DSP 플랫폼과 FPGA-DSP 구조에서의 처리시간 비교

	Module	DSP only (ms)	FPGA+DSP (ms)
차선 인식	Road marking extraction	30.6	FPGA
	Outlier removal	3.2	
	Line merging	6.5	23.2
	Lane model fitting	14.3	
	Tracking	5.8	
차량 인식	Border detection	32	FPGA
	Shadow detector	30.6	
	U-shape detection	10.9	29.8
	Bounding-box generation	8.5	
	Tracking	3.8	
기타	데이터 전송 및 디코딩	해당 없음	2.9

V. 결 론

본 논문에서는 차선 및 차량인식 시스템을 FPGA와 DSP를 이용해 구현하였다. 전체 시스템을 FPGA와 DSP를 이용해 실시간 처리하기위해 특징요소 추출단위 분할 방법을 제안하였다. 대량의 영상정보로부터 소량의 특징 점을 재생산하는 부분을 FPGA로 구현하고, 나머지 부분을 DSP로 구현하여 실시간 처리가 가능해짐을 볼 수 있었다. 또한 DSP의 주변장치를

적극적으로 활용하여 FPGA와 DSP사이의 인터페이스를 구성하여 두 하드웨어를 효율적으로 연동시킬 수 있었다. 이는 다수의 DSP를 병렬적으로 배치하여 구현하였을 때 보다 더 적은 자원을 사용하면서 실시간 처리를 할 수 있음을 보여준다. M. Boumediene et al.[15] 는 차량인식만을 Intel社의 E7300 2.6GHz의 프로세서로 640×480 영상에 대해 약 90%의 인식 성공률과 11 (frame/sec)의 처리 성능을 보였지만, 본 연구에서는 차량인식만을 대상으로 할 경우 약 30 (frame/sec)로 동작 가능하며 차선 및 차량인식 시스템에 대해 약 15 (frame/sec)의 처리 성능을 보여 실시간 시스템으로써 충분한 처리성능을 보여준다. 제안된 FPGA와 DSP의 역할 분담 방법을 이용하여 다양한 영상처리 알고리즘을 실시간으로 구현 가능하다는 것을 확인할 수 있었지만 DSP로 구현된 부분에 적용된 이외의 최적화 방법을 적용하여 전체 시스템의 성능을 향상시킬 수 있다. 향후 전체 시스템의 처리 성능과 알고리즘의 성능을 높이는 방안에 대한 추가 연구가 필요하다.

참 고 문 헌

〔1〕 J. C. McCall and M. M. Trivedi, "Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation," *IEEE Transactions on Intelligent Transportation Systems*, 7(1), pp. 20-37, 2006.

〔2〕 Z. Sun, G. Bebis and R. Miller, "On-road vehicle detection: a review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5), pp. 694-711, 2006.

〔3〕 Z. Liu, Y. Wang and W. Liu, "Design of a color image fusion system based on DSP and FPGA," *International Congress on Image and Signal Processing*, pp. 797-800, 2010.

〔4〕 S. S. Kim and S. Jung, "Hardware implementation of a real-time neural network controller with a DSP and FPGA for nonlinear systems," *IEEE Transactions on Industrial Electronics*, pp. 265-271, 2007.

〔5〕 이운근, 이순웅, 조석빈, 고덕화, 백광렬, "도로교통 영상처리를 위한 고속 영상처리시스템의 하드웨어 구현", *제어·자동화 시스템공학 논문지*, 9(7), pp. 498-506, July 2003.

〔6〕 J. Zhang, H. -B. Su and Q. -Z. Wu, "Research of multi-mode tracking based on multi-DSP and FPGA," *International Conference on Intelligent Human-Machine Systems and Cybernetics*, pp. 230-234, 2010.

〔7〕 L. Yan, T. Zhang and S. Zhong, "A DSP/FPGA - based parallel architecture for real-time image processing," *in Proceedings of the Sixth World Congress on Intelligent Control and Automation*, pp. 10022-10025, 2006.

〔8〕 M. Felisa and P. Zani, "Robust monocular lane detection in urban environments," *in Proceedings of Intelligent Vehicles Symposium*, pp. 591-596, 2010.

〔9〕 C. Torress-Huitzil and M. Arias-Estrada, "FPGA-based configurable systolic architecture for window-based image processing," *EURASIP Journal on Applied Signal Processing*, 7, pp. 1024-1034, 2005.

〔10〕 M. Jablonski and M. Gorgon, "Handel-C implementation of classical component labelling algorithm," *in Proceedings of EUROMICRO Systems on Digital System Design*, pp. 387-393, 2004.

〔11〕 노시봉, 안희준, 이명진, 오혁준, "임베디드 DSP 기반 시스템을 위한 H.264 소프트웨어 부호기의 실시간 최적화", *한국통신학회논문지*, 34(10), pp. 983-991, October 2009.

〔12〕 Texas Instuments, *TMS320DM6437 digital media processor*, 2006.

〔13〕 Texas Instuments, *TMS320C64x+ DSP Megamodule*, 2010.

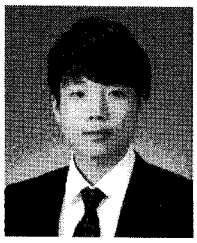
〔14〕 Texas Instuments, *TMS320C64x+ IQmath library user's guide*, 2008.

〔15〕 M. Boumediene, A. Ouamri and M. Keche, "Vehicle detection algorithm based on horizontal/vertical edges," *International Workshop on Systems, Signal Processing and their Applications*, pp. 396-399, 2011.

〔16〕 M. Bertozzi, A. Broggi and S. Castelluccio, "A real-time oriented system for vehicle detection," *Journal of System Architecture*, 43(1-5), pp. 317-325, 1997.

〔17〕 AVNET, *Spartan-3A DSP DaVinci development kit user guide*, 2009.

김 일 호 (Il-Ho Kim) 준회원

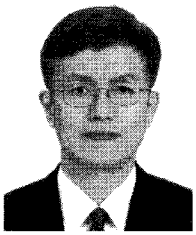


2010년 2월 서강대학교 전자공학과 졸업

2010년 3월~현재 서강대학교 전자공학과 석사

<관심분야> 컴퓨터비전, FPGA, DSP

김 경 환 (Gyeonghwan Kim) 정회원



1984년 2월 서강대학교 전자공학과 졸업

1986년 2월 서강대학교 전자공학과 석사

1996년 2월 State University of New York at Buffalo 전기 및 컴퓨터 공학과 박사

1997년 9월~현재 서강대학교 전자공학과 교수

<관심분야> 영상신호해석, 패턴인식, 임베디드 시스템 디자인