한국산학기술학회논문지
Vol. 8, No. 3, pp. 513-521, 2007

임베디드 시스템에서 CIS 카메라 인터페이스의 구현

이완수[1], 노영섭[1*], 오삼권[2], 황희융[1]

Developing a CIS Camera Interface for Embedded Systems

Wan-Su Lee[1], Young-Sub Roh[1*], Sam-Kwan Oh[2] and Hee-Yeung Hwang[1]

요 약 최근 소형 이동단말기 시장에서 멀티미디어 기능 중에서 카메라 기능은 필수 항목으로 자리 잡았다. 그러나 많은 SoC들 중에서는 아직도 카메라 인터페이스를 지원하지 않는 경우가 많아 저 가격으로 임베디드 기기를 구현하고자 하는 경우 많은 애로사항이 따르게 된다. 따라서 본 논문에서는 임베디드 시스템에서 필수 기능으로 자리 잡은 카메라 인터페이스가 없는 경우 쉽게 카메라를 지원할 수 있는 방안을 제시 하였다. 이를 위하여 CMOS Image Sensor (CIS)를 사용하여 그 인터페이스를 구현하고 디바이스 드라이버를 작성함으로써 간단히 임베디드 시스템에서 CIS를 지원할 수 있는 방안을 제시 하였다.

Abstract Recently, camera function is one of the most primary functions out of the multimedia capabilities in the small mobile terminals. But, it has been difficult for implementing embedded devices with low cost because of not supporting camera interface in many SoCs. Thus, this paper presents a method of supporting camera function with ease for embedded devices which has not camera interface. For this purpose, the interface is implemented for a CMOS image sensor. The method is also provided that CIS(CMOS Image Sensor) is supported in the embedded system by programming the device driver.

Key Words : CMOS Image Sensor(CIS), FPGA, Embedded System, Linux, Device Driver

1. 서론

최근 소형 임베디드 기기에서의 카메라 기능은 필수 기능으로 자리 잡았다고 할 만큼 모든 임베디드 기기에서 카메라를 내장하고 있다. 하지만 일반적으로 많이 사용되는 상용 SoC들 중, 가격이 5,000원 ~ 10,000원 정도로 저렴한 S3F441FX[10], S3F4Ax[11], AT91SAM7S-(E)xxx[13], AT91SAM926x, EP93xx[12], 그리고 EP73xx등은 카메라 인터페이스를 제공하고 있지 않아, SoC 단독으로는 카메라를 사용하여 제품을 설계할 수 없도록 되어 있다. 따라서 카메라 기능을 내장한 저렴한 가격의 제품을 개발하고자 할 경우, 저가격의 SoC에 카메라를 지원하기 위한 외부 회로를 추가하고 필요한 디바이스 드라이버 소프트웨어를 개발하여야 하는데, 이는 개발 기간의 연장을 가져와 제품의 출시 시점을 늦추게 될 뿐만 아니라, 더 많은 개발 인력을 투입하여야 하게 된다. 물론 고성능이면서 카메라 인터페이스가 내장된 S3C241x, S3C244x[14], Au1200, 그리고 PXA32x[15]등을 사용하면 이와 같은 불편을 해소 할 수는 있으나, 이들의 가격이 20,000원 ~ 40,000원 정도로 높다. 이는 불필요한 기능을 내장하게 될 뿐만 아니라 높은 SoC 가격으로 인하여 제품의 가격 경쟁력을 상실하는 요인으로 작용하게 된다.

이와 같은 현실적 제약점을 해결하기 위하여 본 논문에서는 카메라를 지원하지 않는 대신 가격이 저렴한 SoC를 사용하여 임베디드 기기를 구성하고자 할 경우에 카메라를 지원하기 위하여 필요한 인터페이스 회로와 디바이스 드라이버를 구현하였다. 그리고 구현된 인터페이스와 디바이스 드라이버는 모의실험과 동작시험을 통하여 본 논문에서 제안한 방법이 타당함을 보였다.

[1]서울벤처정보대학원대학교 임베디드시스템학과
[2]호서대학교 컴퓨터공학부
*교신저자: 노영섭(ysroh@suv.ac.kr)

2. CIS와 CCD에 대한 소개와 차이점

현대의 정보통신 사회에 있어서 카메라는 여러 분야에 사용이 되고 있다. 카메라는 아날로그 사진에서 피사체를 기록하기 위한 필름을 사용하는데 이미지 센서는 바로

이 필름의 빛을 변환하는 역할을 하는 센서를 통하여 필름 대용품으로 사용되는 것이다. 이 이미지 센서는 전하결합소자(CCD)와 상보금속산화물반도체(CMOS)가 대표적이다.

특히 디지털 카메라를 이용하여 과거의 카메라 기능을 대체하고 또한 핸드폰에 카메라 기능을 부착하여 사용을 하고 있어 영상 이미지를 손쉽게 촬영하여 전송하는 기술이 활용되고 있다. 이 디지털 카메라를 핸드폰에 부착하는 등의 많은 디지털 카메라가 사용된다. 카메라에 사용하는 촬상소자[1]가 과거에는 CCD를 사용하였고 근래에 들어서는 CMOS를 이용한 이미지 센서를 사용을 하고 있다.[1]

CIS(CMOS Image Sensor)는 그림 1과 같이 하나의 단위 Pixel Cell의 내부에 하나 이상의 트랜지스터와 광 다이오드(Photo Diode)로 이루어진다. 또한, 그림 2에서 볼 수 있듯이 CIS는 평면상으로 나열된 광 다이오드의 광전변환에 의한 전하를 MOS 트랜지스터로 증폭하고 스위치 회로로 1화소씩 판독(Readout)하는 Image Capture Device로서 각 화소마다 증폭기능을 가진다.

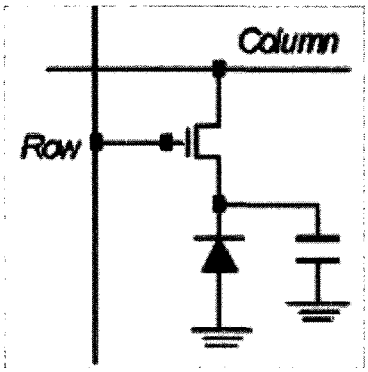

그림 1. 1-CIS의 트랜지스터 픽셀구조

때문에, 광전변환부에 축적된 전하를 아날로그 쉬프트 레지스터로 전송하는 CCD와 달리, 고감도 및 높은 신호대 잡음비(High SNR)을 얻을 수 있는 가능성이 크고, 수평, 수직의 신호선을 각각 하나씩 선택해서 임의의 화소를 판독할 수 있는 랜덤 액세스(Random Access) 기능을 갖는다.[2]

[1] 비디오카메라의 핵심부품으로 영상을 전기 신호로 변환하는 역할을 한다.

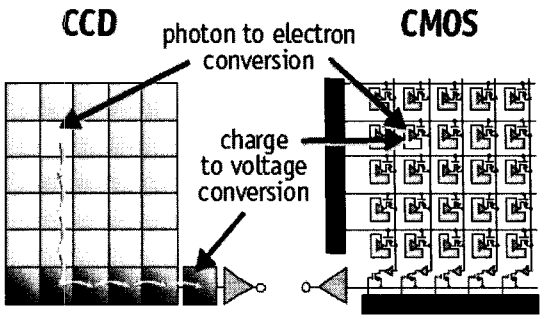

그림 2. CCD와 CIS의 구조

CCD에 의한 촬상소자는 그림 2와 같이 축적된 전하가 시프트 레지스터를 통해 증폭회로로 전송되므로, 거의 완전한 전하의 전송특성이 요구된다. 따라서 감도문제로 인해 셀 사이즈의 축소가 어렵고, 높은 스피드가 곤란하며, 낮은 조도에서 감도가 저하되는 단점을 가지고 있다. 또한, CCD는 독자의 불순물 프로파일(Profile)을 사용하기 때문에, 그림 3에서 볼 수 있듯이, 주변 로직 프로세서(Logic Processor)와의 정합성이 낮아 원칩(On-Chip)으로 만들기가 매우 어렵고, ADC등의 구동회로가 요구되기 때문에 가격 인하에 한계가 있다.[2]

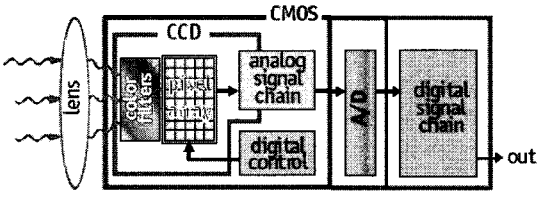

그림 3. CIS와 CCD에 필요한 회로를 설명하는 개념도

반면에, CIS는 표준 CMOS 프로세서(Processor) 기술에 기반을 두고 있기 때문에 웨이퍼(Wafer)공정을 통하여 대량생산이 가능하며, 그림 3과 같이 주변의 구동회로를 포함하여 원 칩(On-Chip)으로 만들기가 쉽고, 저전력 구현이 가능하며, 신호전하에 비례한 전류를 증폭함으로써 미약한 조도에서도 촬상이 가능한 특성을 가진다.

통상의 이미지 센서는 픽셀이 들어있는 시그널 칩들로 구성되어 있으며, SoC 칩 하나에 증폭기, A/D 컨버터, 내부 전압 발생기, 타이밍 발생기 그리고 디지털 로직 등이 결합되며, 이로 인해 공간과 전력 그리고 비용절감에 큰 장점을 갖고 있다[2]

표 1. CCD와 CIS의 비교

	CCD	CIS
전력소비	300mW	50mW
전력 공급모드	15V / 5V~9V	Single Voltage(2.8V)
시스템 통합	최소 3개의 칩	System On Chip
노이즈	CIS에 비해 적다	CCD에 비해 많다
화질	CIS에 비해 좋음	CCD에 비하여 좋지 않으며 많이 개선되고 있다.

3. CIS 인터페이스 및 컨트롤 로직 구성 개발 및 실험

본 장에서는 사용한 시스템 모델과 카메라 기능을 위해 사용한 CIS의 구동을 위한 인터페이스 및 컨트롤 로직 구성, 디바이스 드라이버 개발 과정에 대해 기술한다.

3.1 시스템 모델 및 CIS 인터페이스 설계

본 절에서 사용한 전체 시스템 모델과 사양 그리고 CIS의 구동을 위한 인터페이스 설계에 대해 설명한다.

3.1.1 시스템 모델

본 논문에서 사용한 시스템 모델은 그림 4와 같다.

임베디드 리눅스 기반의 소형 이동단말기용 PXA255 메인보드를 사용하여 PDA구성에 필요한 하드웨어를 구성하는 과정 중 한 부분으로 카메라 인터페이스를 구축하였다.

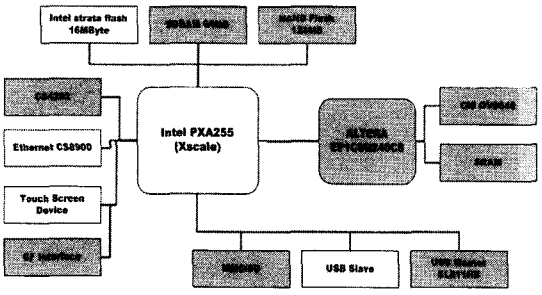

그림 4. 전체 시스템 구성도

사용한 메인보드의 하드웨어 사양은 다음과 같다.

Processor는 Intel PXA255A(400MHz)를 사용하였고 플래시 메모리는 Intel strata flash 16MB이고 SDRAM은 Samsung 64MB를 사용하였다. 그 밖에 UART, Ethernet CS8900A 10BaseT와 Touch Controller ADS7843등의 인터페이스를 가지고 있다.[3]

시스템 모델에서 사용한 시스템의 사양은 표 2와 같다.

표 2. 시스템 사양

	Item	Description
H/W	Processor	Intel PXA255 400MHz
	FPGA	ALTERA EP1C6Q240C8
	CIS	Omnivision OV9640
	SDRAM	Samsung 64MB
	NOR Flash	Intel strata flash 16MB
	NAND Flash	Samsung 128MB
	Sound	CS4202
	Ethernet	CS8900A 10BaseT
	USB	USB Slave
		USB Master SL811HS
	Etc.	MMC/SD, 6.4" TFT LCD
S/W	OS	Linux Kernel ver 2.4.18
	GUI	QT-embedded 2.3.10
		OPIE-1.2.0

3.1.2 CIS 인터페이스 설계

CIS컨트롤 로직구성을 위해 ALTERA사의 EP1C6Q240C8을 사용하였고 카메라는 Omnivision사의 1.3Mega Pixel의 CMOS Image Sensor인 OV9640을 사용하였다.

CIS 인터페이스를 추가한 시스템의 블록다이어그램은 그림 5와 같다. 메인보드와 연결되어 CIS 컨트롤로직을 구성할 FPGA와 CIS 이미지 저장을 위해서 SRAM 256K x 16Bit를 사용하였다.

CIS는 내부 레지스터를 이용하여서 외부 출력 포맷을 설정하게 되는데 본 시스템에서는 CIS의 레지스터 제어를 위한 로직을 추가 구성하지 않고 메인보드의 I^2C를 이용하여 제어하게 설계하였다.

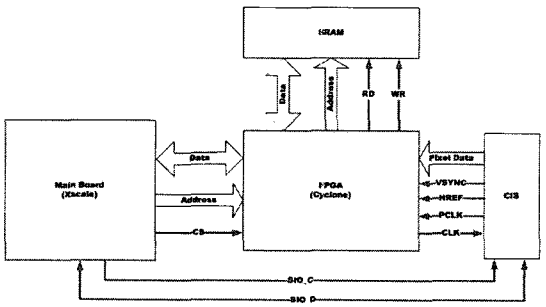

그림 5. CIS를 추가한 시스템 블록다이어그램

FPGA보드 하드웨어 구성을 하면서 EP1C6Q240C8를

위한 Configure회로구성에 많은 정성을 들였다. Cyclone에는 Configuration회로로 Active serial, Passive serial, JTAG-based Configuration 회로가 존재한다.

현재 논문에 구성된 회로는 Passive serial Configuration 회로인데 이것을 사용한 것은 EPC2 Configuration device를 사용해서 이다. EPC2는 재사용이 가능한 플래시 메모리로 1,695,680 bits의 메모리 사이즈를 갖는다.[4]

실제 구성한 Configuration 회로는 그림 6과 같다.

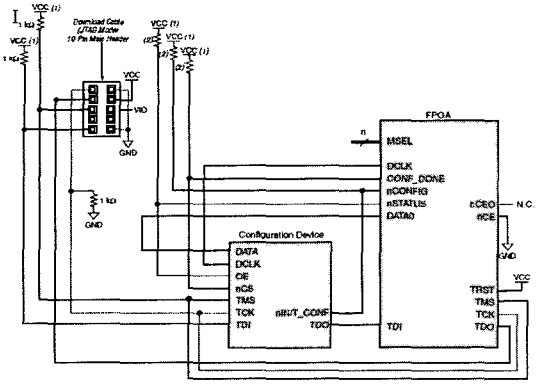

그림 6. Configuration 인터페이스 회로도
자료: Configuration Handbook(2005 ALTERA) p685 Figure 9-2

Configuration회로가 제대로 구성되지 않는다면 JTAG[2]을 통해서 FPGA를 시험할 수 있겠지만 실제 프로그램을 적재할 수 없기 때문에 Configure회로 구성은 중요하다. 구성된 Configuration회로를 정확히 알아야지만 ALTERA에서 제공되는 Quartus II를 사용해서 제대로 프로그래밍을 할 수 있다.

Bank단위로 나뉘는 EP1C6Q240C8의 I/O를 그림 7과 같이 SRAM과 CIS는 각각 EP1C6Q240C8의 Bank1과 2로 할당을 하였으며 메인보드와 연결되는 부분은 Bank3과 4에 할당하여 설계하였다.[5]

CIS내부에는 센서를 제어하기 위한 레지스터가 존재하게 되는데 이 레지스터를 제어하기 위해서 대부분 I^2C 인터페이스를 이용한다. 하지만 OV9640은 SCCB(OmniVision Serial Camera Control Bus)라는 인터페이스를 이용해서 제어하게 된다.

SCCB는 I^2C와 거의 같은 규격을 가지고 있지만 I^2C 데이터 전송에서 9번째 비트를 Acknowledge로 의무적으로 수행되어야 하는 반면 SCCB에서는 데이터 전송에서 9번째 비트를 Don't-Care Bit로 한다는 것이 차이점이다.[6][7]

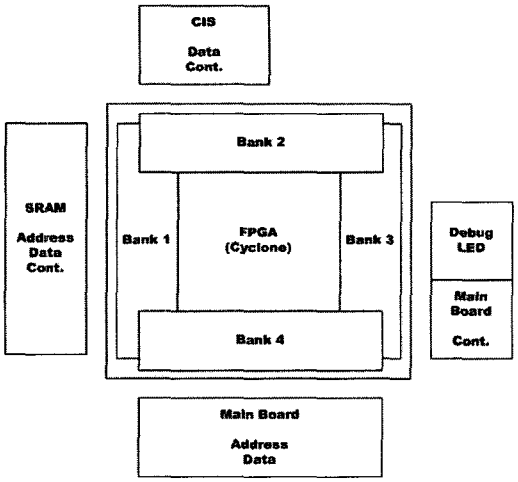

그림 7. EP1C6Q240C8의 Bank할당모습

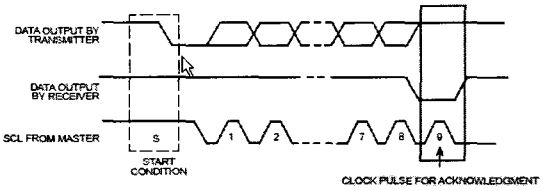

그림 8. Acknowledge on the I2C-bus

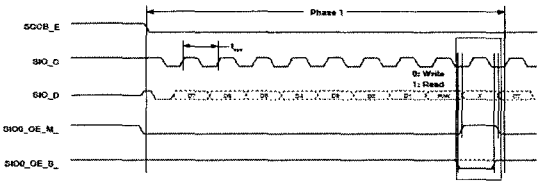

그림 9. Don't-Care Bit of SCCB

3.2 CIS 컨트롤 로직구성과 드라이버 개발

본 절에서는 ALTERA사의 EP1C6Q240C8를 이용한 CIS 컨트롤 로직개발과 CIS용 디바이스 드라이버제작에 대해 기술한다.

3.2.1 CIS 컨트롤 로직구성

CIS 컨트롤 로직구성을 위해 사용한 컴파일러는 ALTERA사의 Quartus II Ver5.0 이다.

Quartus II를 이용하여 VHDL로 CIS컨트롤 로직을 설

[2] Joined Test Action Group

계하였고 시뮬레이션은 ALDEC사의 Active-HDL Ver6.3을 이용해서 수행하였다.

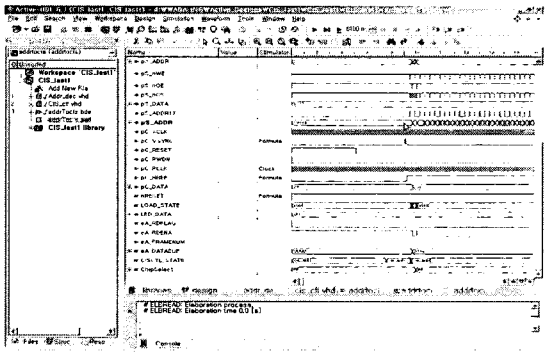

그림 10. Active-HDL을 이용한 시뮬레이션 모습

Quartus II에서도 시뮬레이션을 지원하지만 포트(port)로 정의된 신호 외에 시그널(signal)로 정의된 신호는 시뮬레이션을 돌려도 결과를 확인하기 어렵기 때문에 Active-HDL을 이용하여 시뮬레이션 하였다. Active-HDL는 그림 10과 같이 port와 signal로 정의된 모든 신호의 결과를 볼 수 있기 때문에 VHDL코딩하기에는 편하다.

CIS 컨트롤 로직은 그림 11과 같이 크게 두 부분으로 나뉜다.

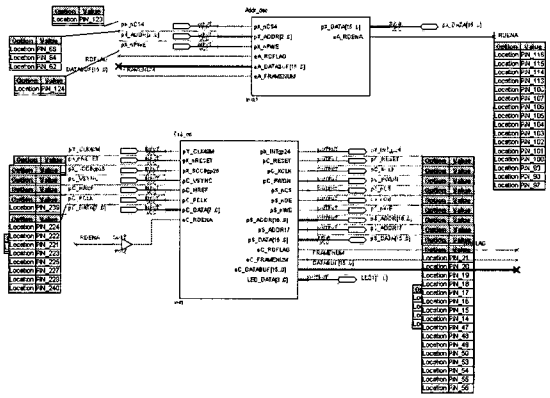

그림 11. CIS 컨트롤 로직 블록도

첫 번째 Addr_dec부는 메인보드와 FPGA의 인터페이스를 담당하는 부분과 CIS_ctl부와의 연동을 위한 부분으로 구성되어 있다. 메인보드와의 인터페이스를 담당하는 부분은 nCS4신호를 어드레스 디코딩하여서 FPGA를 접근할 수 있게 구성하였고 CIS_ctl부와의 데이터 전송을 위한 컨트롤 신호로 구성되어 있다. 그림 12는 Addr_dec부에 구성한 어드레스 디코더의 모습이다.

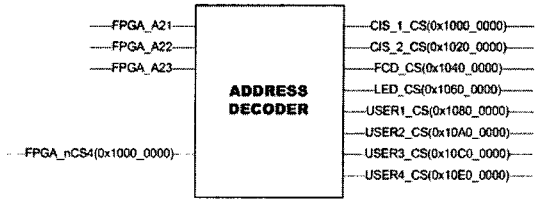

그림 12. Addr_dec의 어드레스 디코더

두 번째로 CIS_ctl부는 3개의 FSM(Finite State Machine)으로 구성되어 있다. CISCTL_STATE FSM은 CIS_ctl부의 초기화 및 전체 제어를 담당하는 부분으로 SCCB를 이용한 CIS 레지스터 제어를 하게 된다. STORE_STATE와 LOAD_STATE FSM은 CIS의 픽셀 데이터(QVGA(320*240))를 SRAM에 Store/Load하는 과정을 맡고 있다. CIS에서 나오는 한 픽셀데이터의 크기는 16비트 크기여서 SRAM에 접근하는 과정은 16비트 데이터 폭을 가지고 수행 된다.[9]

3.2.2 CIS 디바이스 드라이버 제작

본 논문의 SoC는 Intel사의 XScale을 사용하였고, 운영체제로는 Linux 2.4.18을 사용하였다. FPGA를 접근하기 위해서 nCS4를 기반으로 어드레스 디코딩 하였고 이 물리번지를 가상번지로 매핑을 하여야만 리눅스에서 접근을 할 수 있다. 이를 위한 커널내의 설정은 그림 13과 같다.[8]

```
static struct map_desc xhyper255_io_desc[] __initdata = {
 /* virtual     physical    length      domain     r  w  c  b */
 { 0xE8000000, 0x00000000, 0x01000000, DOMAIN_IO, 0, 1, 0, 0 }, // CS0 : Intel Strata Flash 16M
 { 0xF0000000, 0x04000000, 0x00100000, DOMAIN_IO, 0, 1, 0, 0 }, // CS1 : CS8900A
 { 0xF0200000, 0x04700000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS1 : 2554 ADS7843/LCD
 { 0xF0120000, 0x08000000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS3 : EXT_INS [wada]
 { 0xF0130000, 0x08100000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS2 : EXT_OUTS [wada]
 { 0xF0150000, 0x10000000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : CIS1_CS [wada]
 { 0xF0170000, 0x10200000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : CIS2_CS [wada]
 { 0xF0180000, 0x10400000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : FCD_CS [wada]
 { 0xF0190000, 0x10600000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : LED_CS [wada]
 { 0xF0200000, 0x10800000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : USER1_CS [wada]
 { 0xF0210000, 0x10A00000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : USER2_CS [wada]
 { 0xF0220000, 0x10C00000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : USER3_CS [wada]
 { 0xF0230000, 0x10E00000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : USER4_CS [wada]
 { 0xf1200000, 0x14000000, 0x00800000, DOMAIN_IO, 1, 1, 0, 0 }, // CS5 : NAND FLASH [wada]
 LAST_DESC
};
```

그림 13. FPGA 가상메모리 매핑구조체
(Linux/arch/arm/mach-pxa/xhyper255.c)

위 구조체중에서 아래의 필드로 그림 12의 nCS4를 기반으로 디코딩된 물리번지를 가상번지로 매핑을 하게 된다.

```
{ 0xF0160000, 0x10000000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : CIS1_CS [wada]
{ 0xF0170000, 0x10200000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : CIS2_CS [wada]
{ 0xF0180000, 0x10400000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : FCD_CS [wada]
{ 0xF0190000, 0x10600000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : LED_CS [wada]
{ 0xF0200000, 0x10800000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : USER1_CS [wada]
{ 0xF0210000, 0x10A00000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : USER2_CS [wada]
{ 0xF0220000, 0x10C00000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : USER3_CS [wada]
{ 0xF0230000, 0x10E00000, 0x00010000, DOMAIN_IO, 0, 1, 0, 0 }, // CS4 : USER4_CS [wada]
```

FPGA에 구성한 CIS컨트롤 로직은 한 프레임의 CIS 영상이 SRAM에 저장이 되면 XScale의 GPIO(General-Purpose I/O)24를 통해서 인터럽트를 발생하도록 구성되어 있다.[8]

디바이스 드라이버에서는 이 인터럽트 신호에 동기를 맞추어서 SRAM에 저장된 CIS 영상을 XScale로 로드하고 User영역으로 보내는 일을 한다.

그림 14는 드라이버의 open()함수이다. 내용을 살펴보면 인터럽트 핀으로 사용할 GPIO24를 입력 핀으로 설정하고 받아들일 인터럽트의 레벨을 Falling edge로 설정하였다. 또한 request_irq()함수를 통해서 그림 15의 인터럽트 핸들러 함수 CIS_interrupt()를 등록한 것을 볼 수 있다.[8]

```
#define IRQ_CIS                 IRQ_GPIO(24)

int CIS_open(struct inode *minode, struct file *mfile)   //gpioled_open()
{
    int res;
    unsigned int gafr;

    if(cis_usage != 0) return -EBUSY;

    gafr = (0x3 << 16);          //gpio24
    GAFR0_U &= ~gafr;

    GPDR(IRQ_TO_GPIO_2_80(IRQ_CIS)) &= ~GPIO_bit(IRQ_TO_GPIO_2_80(IRQ_CIS));
    set_GPIO_IRQ_edge(IRQ_TO_GPIO_2_80(IRQ_CIS),GPIO_FALLING_EDGE);

    res = request_irq(IRQ_CIS,&CIS_interrupt,SA_INTERRUPT,"CIS",NULL);
    if(res < 0){
        printk(KERN_ERR "%s: Request for IRQ %d failed\n",__FUNCTION__,IRQ_CIS);
        return -1;
    }

    MOD_INC_USE_COUNT;
    cis_usage = 1;
    return 0;
} /* end CIS_open */
```

그림 14. CIS 드라이버(CIS_open함수)

```
#define IMAGE_ADDRESS 0xF0170000

static void CIS_interrupt(int irq, void *dev_id, struct pt_regs *regs)
{
    int i;

    disable_irq(IRQ_CIS);

    pIMAGE = (unsigned int *)IMAGE_ADDRESS;
    for(i=0;i<76800;i++){        //For 320*240(76800)
    yuv[i]=(unsigned short)*pIMAGE;
    udelay(10);

    }
    kill_proc(id,SIGUSR2,1);
}
```

그림 15. CIS 드라이버(CIS_interrupt함수)

그림 15는 CIS_open()함수에서 등록한 인터럽트 핸들러 함수이다. GPIO24의 Falling edge 인터럽트가 발생하면 CIS_interrupt()함수가 수행되는데 수행하는 내용은 실제 CIS이미지가 저장된 가상주소 0xF0170000에서 QVGA(320*240)만큼의 데이터를 읽어오는 것을 알 수 있다. 여기서, 가상주소 0xF0170000는 그림 13에서 보면 CIS_CS2로 되어있는 영역인데 실제 이 주소를 접근하면 SRAM에 저장된 CIS이미지를 로드 할 수 있다. QVGA(320*240)만큼의 데이터를 로드하고서 User프로그램과의 연동을 위해 kill_proc()함수로 SIGUSR2라는 인위적인 시그널을 발생시킨다.

시그널은 받은 USER 프로그램은 커널메모리에 있는 CIS이미지를 읽어오게 되는데 그때 실행되는 함수가 그림 16의 CIS_read()함수이다. 함수의 내용을 보면 copy_to_user()함수를 이용해서 커널메모리에 존재하는 yuv[]에서 153600바이트(320*240*2Byte)의 데이터를 USER영역으로 전달하게 된다.

```
ssize_t CIS_read(struct file *flip, char *buf, size_t count, loff_t *l)
{
    copy_to_user(buf,&yuv,153600);/*153600 = 320*240*(1Pixel=2Byte)*/
    return count;
}
```

그림 16. CIS 드라이버(CIS_read함수)

CIS를 시험하는 과정은 그림 17과 같다.

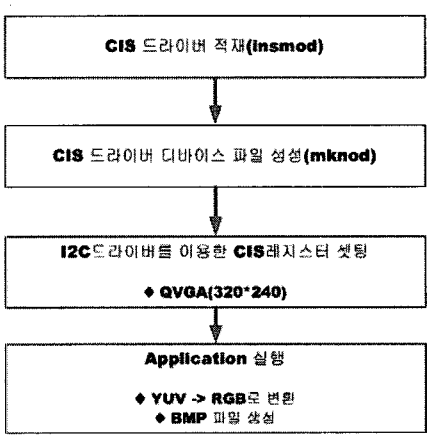

그림 17. CIS의 시험 과정

일단 CIS드라이버를 커널에 적재한 후에 드라이버를 접근하기 위해서 디바이스 파일을 만들었고 기존에 존재하는 I2C드라이버를 이용해서 OV9640의 레지스터 값을 설정한다.[9] 여기서 OV9640은 SLAVE 어드레스 0x30으로 접근하며 QVGA(320*240)해상도와 YUV 포맷의 데이터로 설정하였다. 그리고 응용프로그램을 수행하여서 동작확인을 하였다.

그림 18은 mknod로 주 번호 254, 부 번호 0을 부여하여 캐릭터드라이버용 디바이스 파일을 만드는 과정을 보여주고 있다.

```
[root@WADA255A /dev]$pwd
/dev
[root@WADA255A /dev]$
[root@WADA255A /dev]$mknod CIS c 254 0
[root@WADA255A /dev]$
```

그림 18. CIS 디바이스 파일 생성

가장 먼저 시험한 것은 CIS드라이버를 적재하고서 응용프로그램에서 데이터를 읽어왔을 때 데이터가 제대로 SRAM에 적재가 되었는지 하는 것이었다. 이것을 확인하기 위해서 일단 YUV(4:2:2) 포맷으로 나오는 영상데이터를 RGB로 변환했고, 변환된 RGB를 가지고 쉽게 만들 수 있는 BMP 형식의 영상을 저장하여 확인함으로써 CIS 영상이 제대로 나오는지를 판단했다. 아래 수식은 YUV(4:2:2)를 RGB로 변환하는데 필요한 관계식이다.

이 관계식을 기반으로 YUV를 RGB로 변환하는 함수를 작성하여서 수행하였다.

$$R=Y + 1.371x(V-128)$$
$$G=Y-0.698x(V-128)-0.336x(U-128)$$
$$B=Y+1.732x(U-128)$$

그림 19는 BMP 형식으로 사진영상을 저장하기 위하여 사용한 함수이다. 내용을 보면 BMP 헤더부분을 설정하고 RGB데이터를 써넣는 것을 볼 수 있다.

```
void make_bmp(void)
{
    unsigned int bmp_set;
    int i;

    printf("Make BMP Start! ....\n");

    fprintf( fp, "%c%c", 0x42, 0x4D ) ;

    bmp_set = 14+40+240*320*3 ;fwrite(&bmp_set,4,1,fp);
    bmp_set = 0x0000;      fwrite(&bmp_set,4,1,fp);
    bmp_set = 0x0036;      fwrite(&bmp_set,4,1,fp);
    bmp_set = 0x0028;      fwrite(&bmp_set,4,1,fp);
    bmp_set = 320;         fwrite(&bmp_set,4,1,fp);
    bmp_set = 240;         fwrite(&bmp_set,4,1,fp);
    bmp_set = 0x01;        fwrite(&bmp_set,2,1,fp);
    bmp_set = 24;          fwrite(&bmp_set,2,1,fp);
    bmp_set = 0x0000;      fwrite(&bmp_set,4,1,fp);
    bmp_set = 320*240*3;   fwrite(&bmp_set,4,1,fp);
    bmp_set = 0x0000;      fwrite(&bmp_set,4,1,fp);
    bmp_set = 0x0000;      fwrite(&bmp_set,4,1,fp);
    bmp_set = 0x0000;      fwrite(&bmp_set,4,1,fp);
    bmp_set = 0x0000;      fwrite(&bmp_set,4,1,fp);

    for( i=239; i>=0; i-- )
    {
        fwrite( &rgb[i*320*3], 320*3, 1, fp ) ;
    }

    fclose(fp);
    printf("Make BMP End! ....\n");
} ? end make_bmp ?
```

그림 19. BMP만드는 함수

그림 19에서 만든 함수를 이용하여 CIS의 사진 영상을 BMP 형식으로 저장하였으며, 저장된 사진 영상을 통하여 CIS로부터 가져온 영상데이터를 확인하였다. BMP 형식으로 저장된 사진 영상은 그림 20과 그림 21이다.

그리고 응용 프로그램은 리눅스 GUI(Graphic User Interface) 환경인 QT를 기반으로 작성하였으며, 작성된 응용프로그램과 디바이스 드라이버, 그리고 설계된 FPGA 회로를 통한 CIS의 구동은 그림 22에서 확인할 수 있듯이 원활하게 동작하였다.

본 논문에서 설계 및 구현된 회로는 그림 23의 메인 보드를 기반으로 하고 있다. 그림 23의 메인 보드는 확장 및 실험 등을 위한 커넥터를 통하여 SoC(PXA255)의 신호들 즉 SoC의 로컬 버스(local bus) 신호들을 제공하고 있으며, 설계된 FPGA 보드(그림 24)는 제공된 신호들을 사용하여 그림 25와 같이 SoC의 로컬 버스에 직접 연결되어 있다. 참고로 그림 25는 그림 23의 메인 보드와 그림 24의 제작된 FPGA 보드를 결합하여 동작시키는 모습이다.

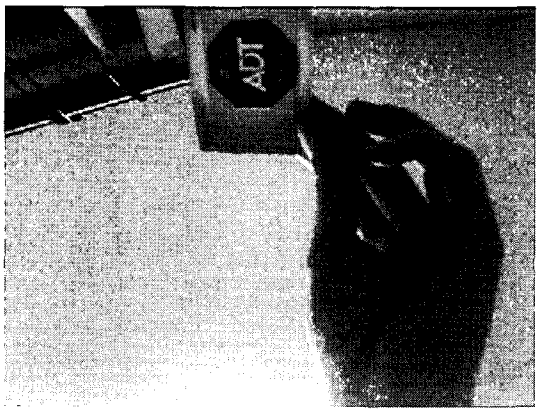

그림 20. BMP 형식으로 저장된 사진 1

그림 21. BMP 형식으로 저장된 사진 2

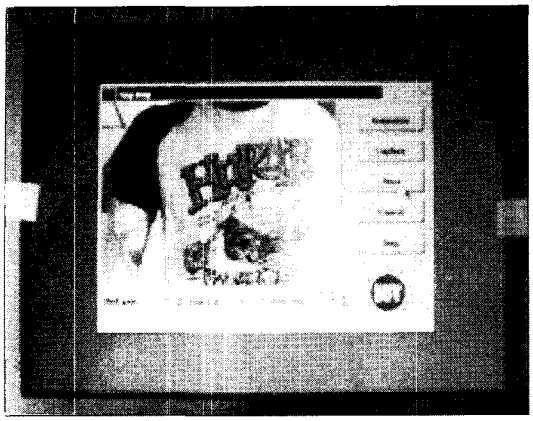

그림 22. QT기반의 CIS영상출력 모습

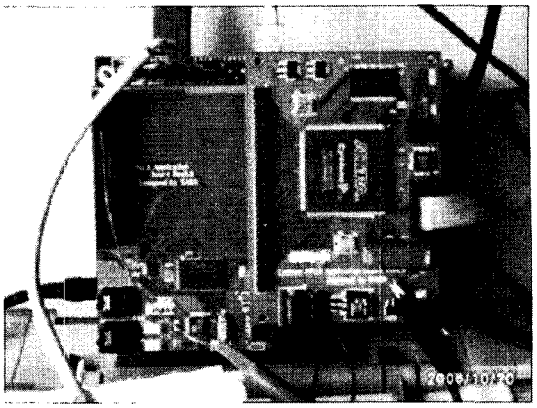

그림 25. 전체시스템 구성모습

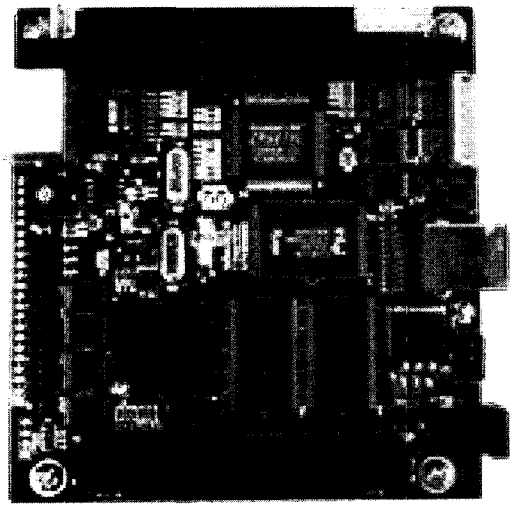

그림 23. 본 논문에서 사용한 메인 보드

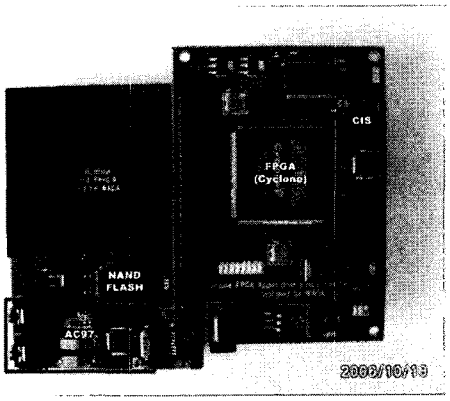

그림 24. 제작한 시험용 FPGA 회로기판

4. 결 론

최근에 거의 모든 소형 임베디드 기기에서 카메라는 필수 기능으로 자리 잡아가고 있으나 저 가격의 제품 설계에 사용할 수 있는 저렴한 상용 SoC들에서는 대부분 카메라를 지원할 수 있는 회로를 내장하지 않고 있다. 물론 고가의 SoC를 사용하면 간단하게 해결할 수도 있지만 제품 가격이 상승하므로 시장에서 가격 경쟁력을 상실하게 된다. 따라서 제품 설계자는 대부분 가격이 저렴한 SoC를 선택하고 간단한 회로를 추가함으로써 필요한 기능을 구현하게 된다.

이와 같은 현실을 바탕으로 본 논문에서는 카메라를 지원하지 않는 저가격의 SoC를 사용하여 임베디드 기기를 구성하고자 할 경우 카메라를 지원하기 위하여 필요한 인터페이스 회로와 디바이스 드라이버를 구현하였고 모의실험과 동작 시험을 통하여 본 논문에서 제안한 방법이 타당함으로 보였다. 따라서 본 논문의 결과를 이용하면 누구나 쉽게 카메라를 지원하는 임베디드 기기를 구현할 수 있는 길을 열게 되었다.

참고문헌

[1] 백남대, "CMOS 이미지 센서", 한국특허정보원, pp.1~2, 2003

[2] 한동희, "CMOS Image Sensor 특허동향", 한국특허정보원, pp.1~2, 2004

[3] 메인보드(XHYPER255A) 제작회사 - http://hybus.net/

[4] "ALTERA Configuration Handbook", pp.683~689, 2005, ALTERA

[5] "ALTERA Cyclone Device Handbook". pp.200~232, 2005, ALTERA

[6] "THE I2C-BUS SPECIFICATION VERSION 2.1", pp.7~8, 2000, philips
[7] "OmniVision Serial Camera Control Bus (SCCB) Functional Specification", pp.13~14, 2003, OmniVision
[8] "Intel® PXA255 Processor Developer's Manual", pp.105~124, pp.183~261, pp.503~540 (2003 INTEL)
[9] "OV9640 product Manual", 2004, Omnivision
[10] "S3F441FX user's manual", rev 2, 삼성전자
[11] "S3F4A0KJ user's manual", rev 1.0, 삼성전자
[12] "EP9301 data sheet", Cirrus Logic.
[13] "AT91SAM7S(E)256 data sheet", ATmel
[14] "S3C2443X user's manual", Rev 1.2, 삼성전자
[15] "Monahans P Processor developers manual", Rev A, Marvell.

이 완 수(Wan-Su Lee) **[정회원]**

- 2005년 2월 : 호서대학교 전자공학과 (공학사)
- 2007년 2월 : 서울벤처정보대학원대학교 임베디드시스템학과 (공학석사)
- 2007년 1월 ~ 현재 : (주)휴맥스 DTV 사업부

<관심분야>
임베디드시스템, 컴퓨터구조, 정보통신

노 영 섭(Young-Sub Roh) **[정회원]**

- 1988년 2월 : 인하대학교 전자공학과(공학사)
- 1996년 8월 : 한국과학기술원 정보및통신공학과(공학석사)
- 2005년 2월 : 고려대학교 전기, 전자,전파공학과(공학박사)
- 1987년 11월 ~ 1998년 2월 : LG전자 미디어통신연구소 선임연구원
- 1998년 3월 ~ 2001년 2월 : 청강문화산업대학교 이동통신과 교수
- 2001년 3월 ~ 2005년 2월 : 주식회사 싸이버뱅크 연구개발부문 상무이사
- 2005년 3월 ~ 현 재 : 서울벤처정보대학원대학교 임베디드시스템학과 교수

<관심분야>
임베디드시스템, 모바일 컴퓨팅, 이동통신, 유비쿼터스 네트워크

오 삼 권(Sam-Kweon Oh) **[종신회원]**

- 1980년 2월 : 한국항공대학교 항공전자공학과(공학사)
- 1986년 12월 :남플로리다대학교 컴퓨터과학및공학과 (컴퓨터과학석사)
- 1994년 5월 : 퀸즈대학교 컴퓨터및정보과학과(컴퓨터과학박사)
- 1995년 3월~현재 : 호서대학교 컴퓨터공학부 교수
- 1980년 2월~1984년 8월 : 삼성전자통신연구소 연구원
- 1994년 9월~1995년 1월 : 한국전자통신연구원 선임연구원

<관심분야>
임베디드시스템, 실시간및분산시스템, 컴퓨터 게임.

황 희 융(Hee-Yeung Hwang) **[종신회원]**

- 1960년 2월 : 서울대학교 공과대학 전기공학과 (공학사)
- 1964년 2월 : 서울대학교 대학원 전기공학과 (공학석사)
- 1974년 4월 : 서울대학교 대학원 전기공학과 (공학박사)
- 1968년 2월 ~ 1993년 2월 : 서울대학교 공과대학 교수
- 1983년 12월 ~ 1984년 12월 : 미국 Florida F.I.T 객원교수
- 1989년 11월 ~ 1992년 3월 : 서울대 컴퓨터 신기술 공동연구소장
- 1993년 3월 ~ 2004년 2월 : 호서대학교 공과대학 교수
- 2004년 3월 ~ 현 재 : 서울벤처정보대학원대학교 임베디드시스템학과 교수

<관심분야>
임베디드시스템, 마이크로프로세서, RTOS