KAERI/TR-2552/2003

디지털 영상캡쳐 카드 및 카운터를 이용한 핵물질 감시 시스템 구현 기술

Implementation of Nuclear Material Surveillance System Based on the Digital Video Capture Card and Counter

KAERI

2003. 7.

한국원자력연구소

제 출 문

한국원자력연구소장 귀하

본 보고서를 2003년도 “사용후핵연료 관리·이용 기술개발” 과제 (세부 과제 “사용후핵연료 특성계량화 기술개발”)의 기술보고서로 제출합니다.

2003. 7.

주 저 자 이 상 윤

공 저 자 송 대 용

고 원 일

하 장 호

김 호 동

목 차

표 목 차

그림 목차

1. 개 요

최근, 사회의 다변화와 무인 보안 감시의 필요성이 커지면서 특정한 장소와 목적을 위해 사용되던 감시 시스템의 적용이 일반적인 일상생활 환경으로 확대되고 있다. 디지털 기술의 급속한 발전에 따라 기존의 아날로그 영상 기록 매체를 이용한 보안 감시 시스템에서 PC를 이용한 디지털 감시 시스템으로의 전환이 급속하게 진행되고 있으며, 기존의 아날로그 CCD 카메라를 대체할 수 있는 디지털 카메라의 개발도 이미 완료되어 시장에서 활용되고 있다. 또한, 웹으로 실시간 원격 전송이 가능한 web cam의 사용도 점차 확대되고 있다.

이와 같이, 최근 몇 년 간에 진행된 반도체 기술의 급속한 발전에 따라 저가의 CMOS 카메라와 영상캡쳐 보드가 보급되면서, 2GHz 대를 넘어서는 CPU와 수십 MB을 넘는 비디오 램을 장착한 PC의 사용이 보편화된 현재에는 감시 시스템에 특별한 지식이 없는 일반 사용자들도 감시 시스템을 구현할 수 있게 되었다. 또한, 이러한 감시시스템의 적용은, 연구소의 연구실과 실험실에도 파급되어 다양한 목적을 위한 감시 시스템의 구현을 위한 욕구가 증가되고 있다.

본 기술보고서에서는, 디지털 영상캡쳐 카드와 카운터를 이용하여 핵물질 감시 시스템을 개발하기 위한 구현 기법에 대해 정리하였다. 감시 시스템의 구현을 위해 현재 국내에서 쉽게 입수할 수 있는 PCI 기반의 영상 캡쳐 카드 및 카운터의 구조적 특성과 SDK 함수의 사용법을 요약하였으며, Borland C++ Builder를 이용한 감시 프로그램 구현 기법에 대해 정리하였다. 참고로 본 기술보고서에서는 기본적으로 C++ 언어 기반의 프로그램 개발 경험이 있는 사용자를 대상으로 내용을 정리하였다.

본 기술보고서는 차세대 핵연료관리 실증 시설에서의 핵물질 안전조치를 위한 감시 시스템 개발을 위해 활용될 수 있을 것이며, 특히 다양한 실험 환경에서의 감시 시스템 구축을 필요로 하는 연구자들을 위한 참고 자료로 활용될 수 있을 것으로 기대된다.

2. 감시 시스템의 하드웨어

2.1 핵물질 감시 시스템의 구성

일반적인 감시 시스템은 다수의 아날로그 CCD 카메라와 영상 캡쳐 카드 및 디지털 센서로 구성된다. 현재 국내에서 시판되는 대부분의 영상캡쳐카드는 5V TTL 신호를 처리할 수 있는 IO 기능을 가지고 있으며 근접 센서나 열감지 센서 등에서 입력되는 신호를 이용하여 영상 캡쳐의 신호를 triggering 할 수 있도록 하고 있다. 이러한 디지털 IO 기능에는 카운터가 제외되어 있으며 단지 trigger 신호로만 사용된다.

본 연구에서 구현하고자 하는 핵물질 감시 시스템에는 방사선 계측기로부터 입력되는 신호를 처리할 수 있는 디지털 카운터 보드를 추가로 장착하여 영상 정보 및 방사선 정보를 병행하여 처리하도록 한다. 본 연구에서 구현하고자 하는 핵물질 감시 시스템의 구성은 다음과 같으며, 본 기술보고서에서는 영상 캡쳐 보드 및 카운터 보드의 운영을 위한 기본 개념과 SDK 함수의 활용 기법에 대해서만 주로 다루고자 한다.

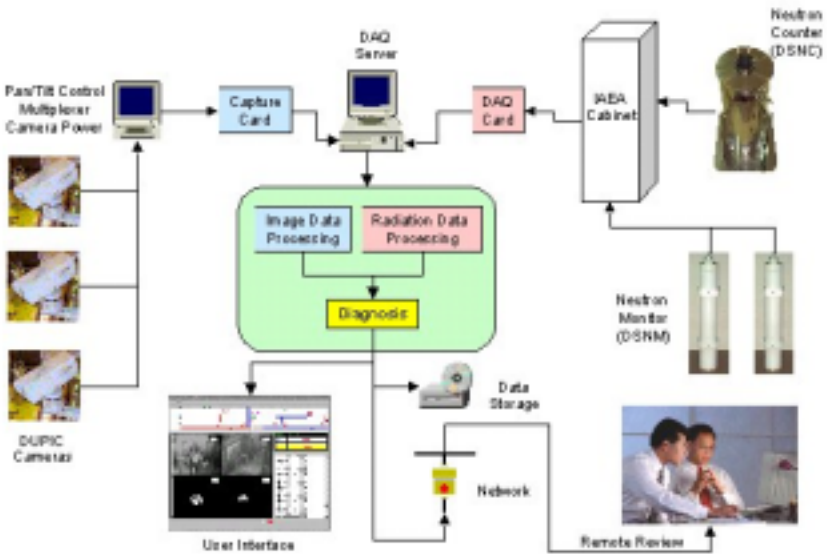

그림 2-1. 핵물질 감시 시스템의 구성도.

2.2 디지털 카운터 보드

디지털 카운터는 5V의 TTL 신호를 입력받아 일정한 reset 신호가 들어올 때까지 buffer ram에 저장하는 역할을 수행한다. 최근 국내에서도 저가의 다양한 카운터 보드가 생산되기 시작하였으며, 본 연구에서는 카운터와 엔코더 기능을 동시에 갖추고 있는 커미조아(주)의 SD501 보드를 사용하였다. 커미조아(주)에서는 다양한 종류의 PC용 카운터와 엔코더를 생산하고 있으며 일반 사용자가 활용하기 쉽도록 ActiveX component를 제공하고 있다. 본 연구에서 적용하는 SD501 카운터는 일반적인 카운터에 적용되는 8254 ADC 에 비해 정밀도가 향상된 32bit FPGA 카운터를 채용하고 있으며 주로 엔코더 센서를 계측하는데 사용된다. COMI-SD501 보드는

엔코더의 3가지 신호 (A상, B상, Z상)를 모두 입력받을 수 있으며, A/B 상과 Z상을 동시에 계측할 수 있다. 본 연구에서는 COMI-SD501의 엔코더 기능은 사용하지 않으며 일반 펄스 신호의 카운터 기능만 사용하게 된다. 일반 펄스 신호는 A상과 B상이 따로 존재하지 않으므로 신호선을 어떻게 연결해야 할지 혼동될 수 있으며, 일반 펄스 신호를 카운트하기 위해서는 펄스 신호를 터미널 보드상의 A상 입력단에 연결하면 된다. 사용되는 SDK 라이브러리의 method는 엔코더 신호를 카운트하는 것과 동일하다. 또한, 일반 펄스 신호를 카운트할 경우에도 B상 입력단에 0V 또는 5V를 인가하여 Up/Down 카운트를 할 수 있으며, B상 입력단에 0V가 인가되면 Up-count가 되고, 5V가 인가되면 Down-count가 된다. 만일, B상 단자에 아무 신호도 연결되어 있지 않으면 Up-count를 하게 되며 펄스가 입력될 때마다 카운트가 증가하게 된다. COMI SD501 엔코더 카운터에서 엔코딩 기능을 제외한 모델이 COMI SD-502 카운터이다. COMI-SD502 카운터 보드는 10 channel 24 bit 카운터로서 일반 카운터 기능 외에 펄스 신호의 주파수를 쉽게 계측할 수 있는 기능을 제공한다. 본 연구에서는 두 가지 모델에 대해 모두 성능시험을 진행하였으며 SD501 엔코더 카운터를 사용하지만 SDK 활용 기법은 거의 동일하다. 본 연구에서 적용한 COMI-SD501 카운터의 특성은 다음과 같으며 구조는 그림 2-2 에 도시된 바와 같다.

- 32bit 4channels (Encoder Counter or Direction, Pulse Input Counter)
- 16 bit 4channels Zero Pulse Counter
- 32bit 4 channels Pulse Generator
- 5 channels Digital Output for Motor Driver control
- Hardware Interrupt generator
- 37 pin D-sub connector
- PCI plug and play

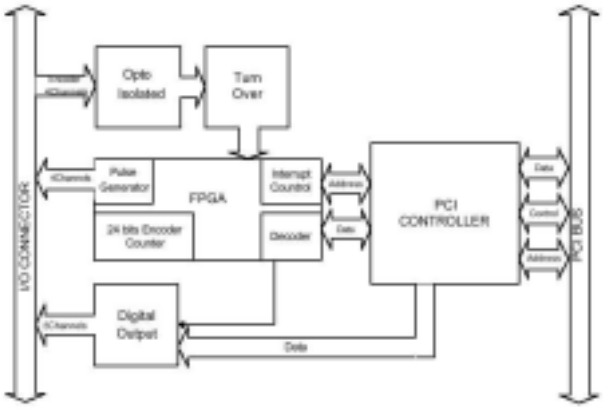

그림 2-2. COMI-SD501 엔코더 카운터의 구조.

그림 2-3. COMI-SD501.

2.3 디지털 영상 캡쳐 보드

감시 시스템에 적용되는 CCD 카메라는 영상을 외부로 출력하기 위해 NTSC 또는 PAL과 같은 신호방식을 사용하며 TV나 비디오를 통해 영상을 보여준다. 이런 영상 신호를 PC에서 사용하기

위해서는 디지털 신호로 변환시키는 과정이 필요하다. 아날로그 신호를 디지털 신호로 변환해 영상을 캡쳐하기 위해서는 통상 frame grabber라는 영상 보드를 사용해 왔다. 사진 (정지영상)과 같은 데이터는 입력에 시간적인 제약이 따르지 않으나 비디오 (동영상)의 경우에는 지속적으로 입력할 데이터가 변하기 때문에 데이터를 실시간에 입력·처리하여야 한다. 비디오 데이터는 640×480 해상도의 화면 하나만 하더라도 0.9 MB 가까운 메모리를 필요로 하며 초당 30 프레임을 입력할 경우 약 27 MB/초의 메모리를 필요로 한다. 이러한 분량의 정보량은 CPU가 처리할 수도 없으며 PC내의 버스로도 전송이 불가능하다. 따라서, 입력되는 즉시 영상을 압축하여 압축된 데이터를 디스크로 보내도록 한다. 이러한 기능을 담당하는 하드웨어를 비디오 캡쳐 보드라 한다.

과거의 ISA 버스를 사용하는 비디오 캡쳐 보드는 버스의 병목현상 때문에 고화질의 비디오를 얻을 수 없었으며 최대 320×240 size로 초당 15 프레임 정도만 가능하였다. PCI 버스가 등장한 이후로는 640×480 size로 초당 30 프레임을 입력하는 고화질의 비디오 데이터의 입력이 가능하다. 국내에서 사용되는 비디오 입력 신호는 초당 30 프레임 525 주사선을 사용하는 NTSC(National Television Standards Committee)방식이다. NTSC 신호는 다시 명암과 색상 정보를 한 선을 사용하여 보내는 혼합(composite) 신호 방식과 명암과 색상을 따로 보내는 S-Video 신호의 두 가지 방식이 사용된다. 일반적으로 캡쳐한 비디오의 화질은 최초 입력 화질에서 결정되기 때문에 S-Video 신호를 사용하는 것이 바람직나, 사용의 편리성 때문에 NTSC 신호를 이용하게 되는 경우가 많다.

현재에도, 전문적인 분야나 고도의 정밀한 화질을 필요로 하는 곳에서는 (수백 만원대의) 비싼 CCD 카메라와 frame grabber를 사용하고 있으나 최근 DVR(Digital Video Recorder) 보드의 보급과 저가의 CMOS 카메라의 등장으로 많은 분야에서 frame grabber의 역할이 크게 감소되었다. 물론 광학적인 성능만 비교하는 경우에는 상대가 안 되지만 가격 대비 성능 면에서는 최근의 DVR 보드는 충분한 경쟁력을 확보하고 있다고 판단된다.

본 연구에서는 국내에서 최근 생산되기 시작한 저가의 DVR 보드를 활용하여 핵물질 감시 시스템을 구현하고자 한다. 본 연구에서 사용한 캡쳐 카드는 ComArt 시스템의 HICAP-50 캡쳐 보드이다. ComArt 시스템(주) 는 국내에서 처음으로 자체 기술로 DVR 보드를 생산 공급하고 있으며 사용자의 필요에 따라 다양한 사양의 PC용 DVR 보드를 선택할 수 있다. HICAP 시리즈는 캡쳐 프레임 스피드에 따라 HICAP-50, 100, 200이 있으며 Techwell사의 video decoder와 Altera사의 FPGA를 processor로 사용한다. HICAP-50의 경우에는 그림 2-5에서 보는 바와 같이 16 channel의 영상 신호를 두 개의 ADC가 time sharing 방식으로 처리하며 고속의 캡쳐 스피드가 요구되지 않는 경우에 적합하다. HICAP-50의 사양은 다음에 요약하는 바와 같으며, 모델에 관계없이 SDK 함수가 통합적으로 관리·배포되고 있으므로 개발자가 사용하는 함수의 method은 거의 동일하다.

- Camera Inputs : 16ea
- Default Video Resolution : 352 x 240 (NTSC), 352 x 288 (PAL)
- Available Video Resolution : 176 x 120, 176 x 144, 160 x 120, 160 x 144, 704 x 480, 704 x 576, 640 x 480, 640 x 576
- Video Compression : S/W MJPEG Codec
- Display Frame : Max 60 Fps(3 Fps/Camera)

- Recording Frame : Max 60 Fps (3 Fps/Camera)

그림 2-4. ComArt HICAP-50.

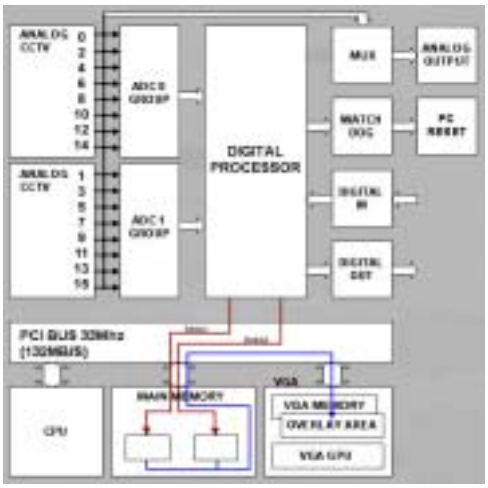

그림 2-5. ComArt HICAP-50의 구조.

ComArt사의 MIS 와 MID 시리즈 보드는 하드웨어 overlay 기능을 제공하고 있으나, 그림 2-5에서 보는 바와 같이 HICAP-50 캡쳐 보드는 하드웨어 overlay 기능이 제외되어 있으며 소프트웨어 scaling 기법을 이용하여 비디오 시그널을 화면에 나타내게 된다. 또한, HICAP-50 보드는 자체적인 아날로그 TV 출력 기능을 가지고 있으며 카메라 신호 중 하나를 선택적으로 TV로 출력할 수 있다. 그림에서 보는 바와 같이 HICAP-50 보드에는 2개의 ADC 그룹이 있으며 최대 캡쳐 프레임은 초당 60 프레임이다.

3. 감시 시스템의 소프트웨어

3.1 디바이스 드라이버

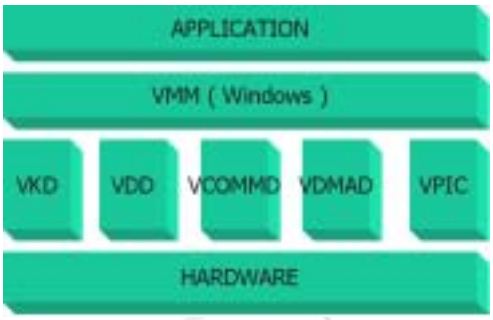

그림 3-1. PC의 계층 구조.

디바이스 드라이버는 위 그림에서 보는 것처럼 중간을 이어주는 역할이라고 생각하면 된다. 가장 기본적인 하드웨어가 있고 그 위로 디바이스 드라이버 그리고 VMM(virtual machine manager), 그 위로 application 이 있다. 우리가 보통 사용하는 프로그램은 application이라고 생각하면 되고 우리가 사용하는 OS는 VMM이며 우리는 항상 GUI를 대하고 있는 것뿐이다. OS의 가장 중요한 역할이 바로 VMM이다. 디바이스 드라이버가 나오게 된 이유는 멀티태스킹 환경 때문이다. 도스 모드에서 하나의 프로그램이 하나의 장치를 사용하면 그냥 직접 억세스해서 돌리면 되었지만 멀티태스킹 환경에서는 하나의 디바이스를 하나의 어플리케이션이 점유하면 문제가 생기게 된다. 여러 어플리케이션이 하나의 디바이스를 공유해야 되는 문제가 생겨서 virtual device라는 개념이 생긴 것이다. 즉, 여러 어플리케이션들이 하나의 디바이스를 공유하기 위해서 디바이스 가상의 디바이스를 만들어서 사용하게 해 주고 virtual machine manager가 그것들을 관리하는 것이다. 이 디바이스 드라이버는 VxD 모델과 WDM이 있는데 이것은 Virtual x driver, Windows Driver Model의 준말이다.

PC에 장착된 장치를 사용하기 위해서는 디바이스를 잡고 드라이버를 설치하는 작업이 필수적으로 요구된다. 이러한 디바이스 드라이버의 설치작업은 PNP 기능이 구현된 장치인 경우에는 OS에서 자동적으로 수행될 수 있는 경우도 있으나, 아직 국내에서 공급되는 장치들의 경우에는 수동으로 장치를 잡아 주어야 하는 경우가 많다. 따라서 사용자는 매뉴얼에 지시된 대로 장치를 장치관리자에서 잡아주도록 해야 한다.

3.2 디지털 카운터

전술한 바와 같이 본 연구에서는 볼랜드사의 C++ builder를 이용하여 감시 시스템을 구현하고자 한다. MS Visual Studio에 비해 builder가 좋은 점은 마치 visual basic과 같이 ocx 또는 ActiveX와 같은 다양한 visual component를 사용하기 쉬운 개발 환경을 제공한다는 것이다.

따라서, 본 연구에서 사용하는 개발 환경은 visual basic 환경과 유사하게 라이브러리를 설치·사용하게 될 것이다.

볼랜드 C++ 빌더 환경에서 COMI-SD 시리즈 디바이스를 제어하는 프로그램을 구현할 때는 (주)커미조아에서 제공하는 COMI-SD 시리즈용 라이브러리를 사용하게 되며, 이 라이브러리는 모든 종류의 COMI-CP/SD 시리즈 디바이스에 적용 가능한 통합 라이브러리이다. COMI-CP/SD 시리즈 카운터의 라이브러리는 ComiDasAx.ocx 라는 ActiveX component 형태로 제공된다. ComiDasAx.ocx는 Comidas.dll 과 사용자 application과의 통신을 담당해 주는 ActiveX로서, 이를 통해 드라이버를 access할 수 있다. 즉, visual basic 또는 C++ builder 개발 환경에서 사용자가 COMI-CP/SD 시리즈 디바이스를 제어하려면, 디바이스 드라이버와 Comidas.dll 파일 및 ComiDasAx.ocx 파일이 시스템에 설치되어 있어야 한다.

3.2.1 라이브러리 설치

SD-501 엔코더 카운터의 라이브러리인 ComiDasAx.ocx는 설치마법사가 자동으로 윈도우의 시스템 폴더에 복사하고 레지스트리에 등록해 주므로 사용자는 개발 환경에 component를 추가해 주기만 하면 된다. 각자의 개발 환경에서 component 등록하는 절차에 따라 toolbox에 ComiDasAx control이 등록되면, 다른 control을 사용하는 것과 같이 Form에 삽입할 수 있게 된다. ComiDasAx control은 Timer control 등과 같이 Design time에서는 Form에서 그 위치나 존재가 보이지만, Run time에서의 최종 사용자에게는 control이 보여지지 않는다. 즉, ComiDasAx control은 내부적으로 COMI-CP/SD 디바이스 시리즈를 제어하고 디바이스로부터 data를 전달받을 때만 사용되어진다.

하나의 Form에는 여러 개의 ComiDasAx control의 instance가 삽입될 수 있다. 이것은 하나의 application이 여러 디바이스를 동시에 제어·계측할 수 있음을 의미한다. 다른 control들과 마찬가지로, control의 인스턴스에 대한 접근은 인스턴스명을 이용한다. 즉, 인스턴스의 property인 경우, '인스턴스명->속성이름' 이 되며, method인 경우, '인스턴스명->메쏘드()'가 된다.

3.2.2 디지털 카운터 모듈의 구현

전술한 바와 같이, ComiDas 시리즈에서 제공하는 디바이스 라이브러리와 ActiveX control의 설치 작업이 완료되면 이를 이용하여 5V TTL 신호를 카운트하는 모듈을 구현할 수 있게 된다. application에서 하나의 Form에는 다수의 ComiDasAx control의 instance가 삽입될 수 있고, 이들은 각각 자신만의 device ID (DeviceID property)와 device instance (DeviceInstance property)로 시스템에 연결된 디바이스와 연결된다. 또한, ComiDasAx control이 제공하는 method들을 이용하여 계측하고 제어하게 된다. Visual Component Library의 다른 component와 마찬가지로, COMI-CP/SD device 시리즈를 이용하여 프로그램을 잘 구현하려면, ComiDasAx가 가지고 있는 property와 method를 잘 숙지하여야 한다.

ComiDasAx는 DeviceID와 DeviceInstance라는 두 개의 property를 가지고 있다. DeviceID는 'Comi-SD501', 'Comi-SD301' 등의 device 명을 가리킨다. 즉, 여러 종류의 device는 DeviceID 속성으로 구분되어 진다. 이러한 DeviceID 속성의 설정은 모듈의 초기화

부분에서 가장 먼저 이루어져야 하며, design time에서 설정해 줄 수도 있고 실제 소스 코드에서

ComiDasAx1.DeviceID = COMI-SD501

과 같이 설정해 줄 수도 있다.

VB의 경우에는 위와 같이 카드의 모델명 만을 설정하면 이에 해당하는 고유 번호가 자동으로 적용되지만, 볼랜드 C++ 빌더의 경우에는 device의 모델에 따른 고유 번호를 직접 기술해 주어야 한다. 즉, SD501의 고유번호는 “46337” 이므로, 볼랜드 C++ 빌더의 경우에는

ComiDasAx1->DeviceID = 46337;

과 같이 설정해 준다.

DeviceInstance 속성은, 같은 종류의 디바이스가 여러 개 설치되었을 경우에, 이를 구분하는 속성이다. 예를 들어, 각각 하나씩의 COMI-SD201과 COMI-SD301이 있다면, 이는

ComiDasAx1.DeviceID = COMI-SD201
ComiDasAx2.DeviceID = COMI-SD301

라고 설정해 주면 되지만, 같은 COMI-SD201 디바이스가 두 개 이상 설치되어 있다면,

ComiDasAx1.DeviceInstance = 0
ComiDasAx2.DeviceInstance = 1

라고 설정해 주어야 한다.

device ID의 설정이 완료되면 device를 로드해야 한다. 디바이스 로드는 LoadDevice() 함수를 이용하면 된다. 디바이스 로딩을 통해 초기화 작업이 완료된 이후에 카운팅을 하기 위한 단계에서 적용되는 함수들의 특성은 다음과 같다. 아래의 함수 설명에서 SD502 카드를 사용하는 경우에는 함수의 명칭이 SD502XXX로 변경되어 있으나, 함수의 용도와 구현 기법은 유사하므로 본 보고서에서는 SD501을 기준으로 설명하기로 한다.

◎ Function **LoadDevice** As Boolean

이 method는 하나의 COMIDAS 디바이스를 load한다. 각 device를 제어하기 위해서는 먼저 이 method를 이용하여 해당 device를 준비시켜야 한다. device loading이 성공하면 ‘1’을 return하고 실패하면 ‘0’을 return한다. 이 method는 제어하고자 하는 device 수만큼 수행되어야 한다.

◎ Sub **EncConfig** (ByVal Channel As Integer, ByVal Mode As Integer, ByVal ResetByZ As Integer)

이 method는 지정한 A/B 상 카운터 채널의 모드를 설정한다. 디바이스를 로드하고 나면 EncConfig()를 이용하여 엔코더 카운터 채널의 모드를 설정해야 한다.

- 매개변수

▶Channel : Counter 채널 번호. 채널 번호는 0부터 시작한다.

▶Mode : 이 매개변수는 차기 버전을 위해 확보된 것이며 현재에는 이 값이 반드시 0 또는 ENCODER_1X로 설정되어야 한다.

▶ResetByZ : A/B 상 카운트 값을 Z 상 입력 단자에 펄스가 입력될 때마다 reset할 것인지를 지정한다.

- 0 ⇒ 이 값으로 지정하면 A/B 상 카운트 값은 Z 상에 의해 영향을 받지 않는다.
- 1 ⇒ 이 값으로 지정하면 A/B 상 카운트 값은 Z - 펄스가 발생할 때마다 reset 된다.

◎ Function **EncRead** (ByVal Channel As Integer) As Long

이 method는 지정한 A/B 상 카운터 채널의 카운트 값을 읽어서 그 값을 반환한다. 카운트 값의 범위는 32 bit 정수 값으로서 -2147483648~2147483648 이다. 반환된 카운트 값이 음의 값이면 회전체가 역 방향으로 회전했음을 의미하며, 양의 값이면 정 방향으로 회전했음을 의미한다.

-매개변수

▶Channel : counter 채널 번호. 채널 번호는 0부터 시작한다.

-return

A/B 상 카운트 값

◎ Sub **EncReset** (ByVal Channel As Integer)

이 mothod는 지정한 A/B 상 카운터 채널의 카운트 값을 0으로 reset한다.

-매개변수

▶Channel : counter 채널 번호. 채널 번호는 0부터 시작한다.

이상과 같은 함수들을 이용하여 카운팅이 이루어지도록 하며, device의 사용이 완료되면 UnloadDevice() 함수를 이용하여 device를 언로드한다. 작성된 볼랜드 C++ 빌더용 카운팅 모듈의 프로그램 리스트를 그림 3-2 에 제시하였다.

3.2.3 Application의 배포

사용자가 SD 시리즈 엔코더 카운터를 이용하는 적절한 application을 개발한 후, 프로그램을 배포하는 경우에도 디바이스 드라이버와 Comidas.dll 및 ComiDasAx.ocx 파일을 같이 배포하여야 한다. 일반적으로 디바이스의 드라이버는 "\Program Files\COMIZOA\ Comidas-CPSD\Driver"

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "DAQCard.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "COMIDASAXLib_OCX"
#pragma resource "*.dfm"
TForm3 *Form3;
//---------------------------------------------------------------------------
__fastcall TForm3::TForm3(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------


void __fastcall TForm3::Init()
{
        ComiDasAX1->DeviceID = 46337;
        bool success = ComiDasAX1->LoadDevice();
        if (success != True)
        MessageDlg("Counter Device Error!",mtConfirmation,TMsgDlgButtons() << mbOK,0 );
        else
        for (int i=0;i<4;i++)
        ComiDasAX1->EncConfig(i,0,0);

}

void __fastcall TForm3::Reset()
{
        for (int i=0;i<4;i++)
         ComiDasAX1->EncReset(i);
}

int __fastcall TForm3::Read(int channel)
{
        int data;
        data = ComiDasAX1->EncRead(channel);
        return data;
}

void __fastcall TForm3::Stop()
{
        ComiDasAX1->UnloadDevice();
}
```

그림 3-2. SD501 카운팅 모듈 소스 리스트.

폴더에 있으며, application의 배포 시에는 이 드라이버 파일들을 같이 배포하여 최종사용자로 하여금 디바이스 설치시 드라이버를 설치하도록 해야 한다. 또한, "\Program Files\COMIZOA\Comidas-CPSD\C_CPP\Library" 폴더에는 Comidas.dll 파일이 있으며, 배포 시에는 이 파일이 대상 시스템의 시스템 폴더에 복사되도록 해야 한다. (시스템 폴더는 Windows 9x 와 Me 인 경우 C:\Windows\System 이며, NT 와 XP 인 경우 C:\Window\System32 이다.)

SD501 엔코더 카운터의 ActiveX component는 "\Program Files\COMIZOA\Comidas-CPSD\VB\Library" 폴더에 ComiDasAx.ocx 파일로 있으며, 배포 시에는 이 파일이 대상 시스템의 OS에 등록되어야 한다. 즉, 이 파일의 위치는 상관없이 OS의 registry에 등록이 되어야 한다. 레지스트리 등록 방법은 DOS command 창에서 'regsvr32 ComiDasAx.ocx' 라는 명령을 실행시켜 주는 방법과 InstallShield와 같은 setup 프로그램을 생성해 주는 utility를 사용하여, 최종 사용자가 배포된 application을 setup이 자동으로 동시에 ocx 파일이 OS에 등록되도록 하는 방법이 있다.

3.3 디지털 영상 캡쳐

3.3.1 디바이스 드라이버

ComArt DVR 캡쳐 보드를 사용하여 영상 캡쳐 모듈을 개발하는 경우, 보드와 같이 공급되는 DLL와 VXD를 포함한 SDK를 사용하게 된다. 본 연구에서 사용하게 되는 HICAP-50 영상 캡쳐 보드의 드라이버 라이브러리 구성 및 연계도는 그림 3-3에 도시된 바와 같다. 그림 3-3 에서 보는 바와 같이, 각 모듈은 서로 다른 모듈과 통신하게 되며 user application program에서는 DLL과 SDK를 이용하므로 하드웨어 계층에는 접근할 필요가 없다. 그러나, application 프로그래머는 사용되는 보드에 대한 기본적인 계층 개념과 DLL 함수들에 대해 숙지할 필요가 있다.

초기에 캡쳐 카드를 PCI 슬롯에 장착하게 되면 PNP 기능에 의해 H/W Wizard를 통해 device

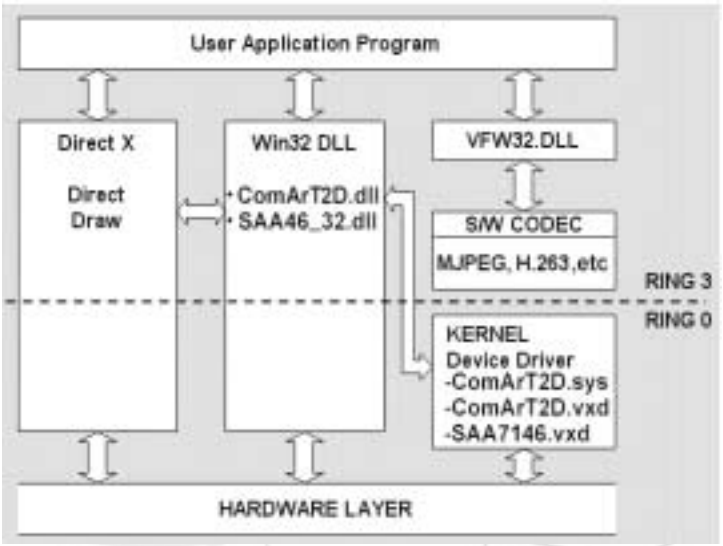

그림 3-3. ComArt 영상 캡쳐 보드의 계층 구조.

driver를 찾게 된다. 이때 드라이버가 있는 CD의 폴더에서 ComArT2M.INF를 선택해 주면 ComArT 2ND Master 드라이버가 설치된다. HICAP-50 캡쳐 카드는 그림 3-4와 같은 3개의 디바이스 드라이버가 기본적으로 'Sound, video and game controller' H/W 장치관리자에 등록되도록 해야 하며, INF 파일에 의해 자동으로 설치되는 Master 이외에는 추가적으로 수동으로 설치해야 한다.

그림 3-3에 도시된 디바이스 드라이버 라이브러리 파일의 역할은 다음과 같다.

- ComArT2D.VXD : Windows 98 series KERNEL
- ComArT2D.SYS : Windows 2K series KERNEL
- ComArT2D.DLL : Dynamic Link Library to control ComArt Boards
- SAA7146.VXD : PNP Driver component
- SAA46_32.DLL : PNP Driver component

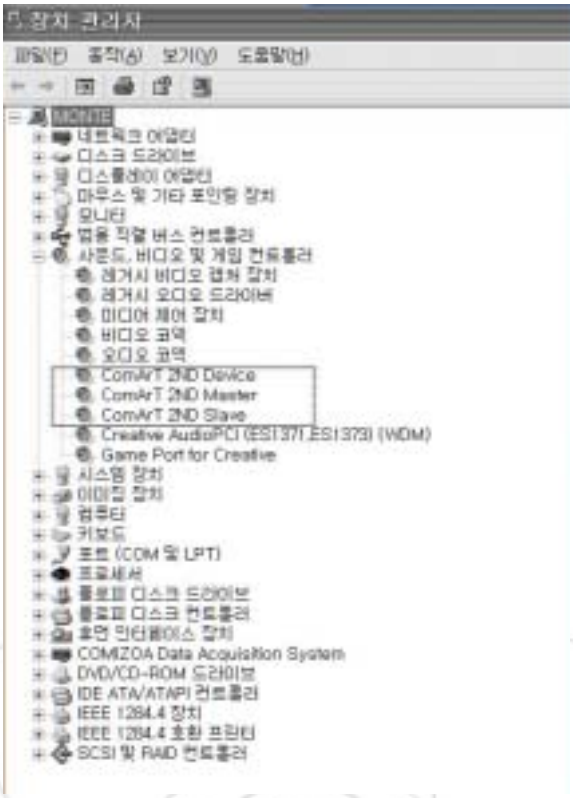

그림 3-4. HICAP-50의 디바이스 드라이버.

Device driver의 설치가 완료되면 SDK를 사용하기 위한 환경을 구현해야 한다. 실질적으로 프로그래밍에 사용되는 것은 ComArT2D.dll이며 이 dll은 ComArT2D.lib에 정의되어 있다. 또한, 이 dll에 포함된 SDK 함수의 header 파일은 ComArT2D.h 에 정의되어 있다. 따라서, 실질적으로 프로그래밍을 하는 경우에는 application에서 ComArT2D.lib를 사용할 수 있도록 링크해 주고 SDK 함수를 사용하는 class에는 ComArT2D.h를 include시켜 주면 된다. 또한, 프로그램의 배포 시에는 ComArT2D.dll이 application과 같은 폴더에 위치하도록 하면 된다.

Application에서 ComArT2D.lib를 사용하기 위한 설정에서 볼랜드 C++ 빌더의 경우에는 주의해야 할 것이 있다. SDK와 함께 제공되는 ComArT2D.lib는 MS Visual Studio 개발 환경에서 제작・배포된 것이므로 볼랜드 개발 환경과는 포맷이 맞지 않는다. 즉, 모든 Visual C++에서 만든 LIB 포맷은 Borland C++ Builder와는 맞지 않으므로 이를 “COFF2OMF” 라는 tool을 사용하여 변환시켜야만 되며, 그렇지 않으면 link error가 발생한다. COFF2OMF는 아래와 같이 도스 창에서 실행하면 된다.

coff2omf -lib:ms ComArT2D.lib ComArT2D1.lib

이렇게 변환한 후 ComArT2D1.lib 파일을 이용하여 Link 하면 된다.

일반적으로 DLL을 링크하는 방법은 Implicit Linking과 Explicit Linking의 두 가지가 있다.

Implicit Linking이란 프로그램이 시작되면서 해당 DLL을 바로 로드하는 방법으로 다음의 3가지 파일을 필요로 한다.

1) 프로그램이 실행될 때 로드되는 DLL 파일 (ComArT2D.DLL)
2) 컴파일할 때 설정해 주는 함수명이 들어 있는 헤더 파일 (ComArT2D.H)
3) 링크할 때 메인 프로그램이 빈 함수를 설정하는 LIB 파일 (ComArT2D.LIB)

Explicit Linking이란 Implicit Linking에서처럼 세 가지의 파일이 필요 없이 단순하게 DLL을 이용하여 로드할 수 있는 방법으로, 이 방법으로 DLL을 로드할 경우 다음의 세 함수를 이용하게 된다.

1) 라이브러리를 로드하는 LoadLibrary 함수.

```
HINSTANCE LoadLibrary(
LPCTSTR lpLibFileName        //DLL 파일명
);
```

2) 함수의 포인터를 찾는 GetProcAddress 함수.

```
FARPROC GetProcAddress(
HMODULE hModule,             //DLL 인스턴스
LPCSTR lpProcName            //함수 이름
);
```

3) DLL을 메모리에서 제거하는 FreeLibrary 함수.

```
FreeLibrary(hDLL);
```

예를 들어 Test_DLL.dll에 있는 Function 이라는 함수를 이용하고자 한다면 우선 다음과 같이 LoadLibrary 함수를 이용하여 DLL을 메모리에 로드시킨다.

```
HINSTANCE hDll;
hDll = LoadLibrary("Test_DLL.dll");
```

다음으로 Function 함수의 포인터를 다음과 같이 GetProcAddress 함수를 이용하여 구한다.

```
typedef int (*FunctionName)(int n);
FunctionName lpFunc;
lpFunc = (FunctionName)GetProcAddress(hDll,"Function");
```

GetProcAddress 함수의 첫 번째 인자는 로드한 DLL의 인스턴스 핸들이며, 두 번째 인자는 이 DLL 안에 있는 함수의 이름이 된다. 즉, GetProcAddress(hDll,"Function"); 이라고 하였을 경우,

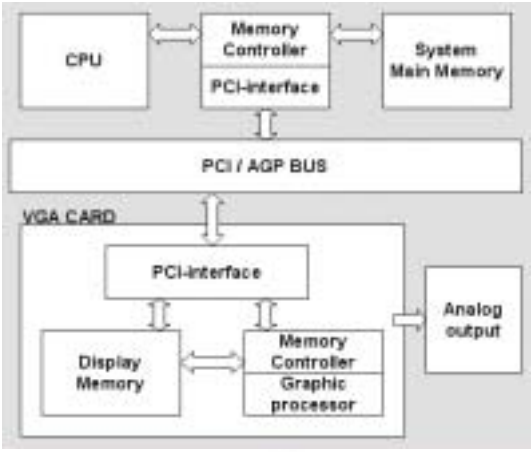

그림 3-5. VGA Interface Block Diagram.

"Test_DLL.dll" 파일에서 Function 함수를 찾아서 이 메모리의 위치를 반환하고 이 위치를 lpFunc가 받는다. 따라서, 이후의 함수의 이용은 다음과 같이 하면 된다.

lpFunc(n);

이와 같이 하면 DLL 내부에 있는 Function 함수를 수행하는 것과 동일한 결과를 나타낸다.

본 연구의 개발 환경인 볼랜드 C++ 빌더에서, LoadLibrary() 및 FreeLibrary()를 사용하는 Explicit Linking을 하는 경우에는 함수의 이름이 ComArt2D.h에 정의된 이름과 틀리게 되며, 이것은 WINAPI 의 구조적인 문제 때문이다. 따라서, 본 연구에서는 ComArT2D.LIB를 이용하여 Implicit Linking 기법을 이용하도록 하며, Explicit Linking을 이용하는 경우에는 보드의 SDK manual에 추가적으로 기술되어 있는 변경된 함수명을 사용해야 한다.

3.3.2 영상의 이해

VGA 메모리의 구조는 그림 3-5에 나타낸 바와 같다. CPU가 VGA 메모리에 어떤 data를 쓸 때, 그래픽 프로세서는 그 data를 아날로그로 변환시키며 변환된 data를 표시한다. 따라서, 우리가 모니터 스크린을 통해 보는 것은 VGA 메모리에 있는 data 이다.

비디오 stream을 화면에 보여 주는 방식은 두 가지가 있다. 하나는 preview mode이고, 다른 하나는 overlay mode이다. 만일 캡쳐 드라이버가 두 가지 방식을 모두 지원한다면 사용자는 어느 방식을 사용할 것인지 정해줄 수 있다.

preview mode에서는 캡쳐 하드웨어로부터 digitize된 화상정보 프레임들이 먼저 시스템 메모리로 전송된다. 그 다음 메모리에 있는 화상정보를 OS의 GDI(graphic device interface)를 통해, 즉 GDI 함수를 사용해, 캡쳐 윈도우에 그려준다.

overlay mode는 CPU 자원을 전혀 사용하지 않고 overlay 보드의 캡쳐 data용 buffer에 들어 있는 비디오 data를 직접 모니터 상에 뿌려주는 방식을 취한다. 비디오 오버레이(overlay) 란

외부의 아날로그 비디오 신호를 컴퓨터 화면에 보여주기 위한 기법이다. 비디오 오버레이 보드는 디지털 비디오 기술이 개발되기 이전에 비디오를 대화 기능으로 제공하는 방편으로 사용되었다. 컴퓨터로 제어되는 레이저디스크와 같은 기기의 비디오 출력을 별도의 모니터가 아니라 컴퓨터 화면의 일부로 이를 설명하는 문장과 함께 보여 줄 수 있었다. 최근의 비디오 오버레이 보드는 디지털 비디오 편집 과정의 아날로그 비디오 신호를 모니터 상에 보여 주거나 텔레비전 신호를 보여주는 데 사용된다. 대부분의 비디오 오버레이 보드는 비디오 화면 캡쳐 기능을 갖고 있으며 이를 활용하여 이미지 캡쳐용으로도 사용된다. 최근에는 그래픽 기능과 비디오 오버레이 기능을 하나의 보드에 포함시킨 보드들이 사용되고 있다. 참고로 ComArt사의 MID & MIS 시리즈는 H/W overlay 기능이 포함되어 있으나 HICAP 시리즈에는 제외되어 있으므로 본 연구에서는 S/W overlay 기법을 활용한다.

Display surface는 일종의 square 형태로 된 memory buffer라고 생각할 수 있으며 surface는 이미지 정보를 가진 메모리라고 이해하면 된다. surface는 그림 3-6과 3-7에서 보는 바와 같이 Primary surface, Back surface 및 Overlay surface로 구성된다. Primary surface는 현재 출력중인 surface 이며 실질적인 VGA 메모리이다. Back surface는 Primary surface가 출력되는 동안에 뒤에서 application에 의해 변경되는 메모리이다. Overlay surface는 Primary surface에 위치하는 가상적인 surface 개념이다. 이러한 Primary surface의 크기는 메모리 모델에 따라 상이하며, 640×480 크기의 256 colors (8 bits per pixel)인 경우 primary surface 는 307,200 (약 300K)bytes 의 video memory를 차지한다.

그래픽 coprocessor는 Primary surface에 있는 data를 scan하면서 아날로그 모니터에 출력하며, scan 중에 Overlay 되는 부분을 만나면 그 위치에서 Overlay surface의 data를 출력하는 형식을 취한다. 결과적으로 Primary surface에 있는 data는 변경되지 않으면서 Overlay surface로부터 data를 출력하게 되므로 CPU의 load를 줄일 수 있다. Overlay에서 surface의 특정한 위치를 검사하고 발견하기 위한 방법으로서 COLORKEY 값을 사용한다. 예를 들어 COLORKEY 값이 RGB(ff,0,ff)로 setup된 경우에는 그래픽 coprocessor가 Primary surface에서 RGB(ff,0,ff)를 만나면 Primary surface의 data 대신에 Overlay surface의 data를 사용하여 출력하게 된다. 즉, 사용자 application 화면은 Primary surface가 되고 수시로 변경되는 캡쳐 영상은 Overlay surface를 활용하여 출력된다는 개념으로 이해하면 된다.

일반적으로 4MB 비디오 메모리 이하에서 Overlay surface를 구현하게 되면, Primary 와 Overlay surface의 총 메모리가 4MB를 초과하게 되는 경우가 발생할 수 있으며 이 경우에는 Overlay surface가 정상적으로 출력되지 않게 된다. 따라서, 이런 경우에는 Primary surface의 해상도 설정 또는 색 품질을 낮게 하여 일정한 비디오 메모리를 확보하도록 해야 한다.

◎ DirectX의 이해

DirectX는 빠른 속도를 낼 수 있는 고성능 멀티미디어 애플리케이션을 개발하기 위해 마련된 low-level API(Application Programming Interface)이다. 즉, MS-Windows 기반의 PC에서 멀티미디어(그래픽, 비디오, 3차원 애니메이션)를 이용하기 위한 기본적인 모듈이며 Windows OS 및 Internet Explorer의 한 부분이라고 이해하면 된다. DirectX에는 Direct Draw, Direct 3D,

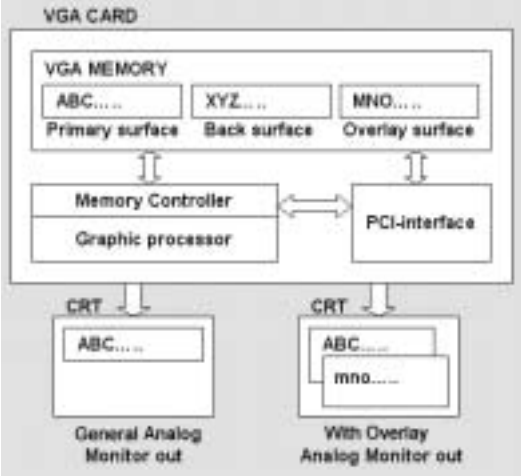

그림 3-6. VGA Display Surface Structure.

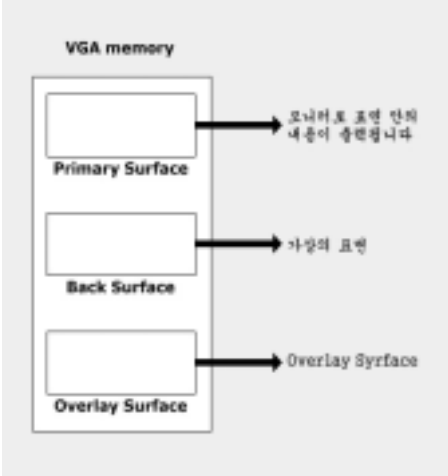

그림 3-7. Overlay Concept.

Direct Sound, Direct Music, Diret Input 등이 포함된다. DirectX는 초기에는 PC 기반의 게임이나 component 개발을 위한 개발자들을 위해 제공되었으나, 최근에는 Windows에서 멀티미디어 디바이스를 이용하기 위한 기반 component가 되어 가고 있다.

Direct 개체는 Microsoft COM(Component Object Model) 기반으로 설계되어 있으며 Direct Draw는 DirectX에서 제공되는 2D control 함수를 포함하는 개체이다. 따라서, overlay surface와 기본적인 2D 처리는 Direct Draw의 사용을 필요로 한다.

ComArt DVR 보드의 SDK는 DirectX의 일부 함수를 사용하며 사용 빈도는 지속적으로 증가될 것으로 예상된다. 따라서, 강력하고 안정된 application program 개발을 필요로 하는 경우에는 DirectX를 숙지하고 이용하도록 노력해야 한다.

3.3.3 영상 포맷

모니터에 나타나는 영상은 모든 하나의 점이 각각 디지털로 된 3바이트 이상의 데이터로 표현된다. 물론 그 이하도 그 이상도 있지만 보통 그렇게 나타낸다. 영상이라고 하는 것은 점이 모여서 선이 되고 선이 모여서 하나의 평면 영상이 되는 것이다. 즉, 영상 데이터는 이차원의 데이터를 가지고 있으며 각 점(픽셀이라고 부름)은 RGB의 데이터를 1바이트씩 3바이트의 값을 가진다. 멀티미디어 데이터는 보통 1바이트씩으로 표현되도록 하고 있으며, 컴퓨터에서의 경우 MMX 명령어들은 8비트처리를 효율적으로 처리할 수 있도록 구성되어 있다. 우리가 쓰는 PC는 일반적으로 32비트 처리를 하는데 멀티미디어 데이터는 8비트이므로 32비트 중에서 각각의 8비트씩 연산을 하도록 한다던가 32비트 중에서 8비트만 연산을 한다던가 멀티미디어 데이터의 크기에 맞추어 동작하도록 되어 있다.

일반적으로 컬러 영상은 RGB 3바이트의 데이터를 가지고 한 픽셀을 나타내게 된다. 그러한 데이터들이 2차원적으로 있으니 어떻게 보면 3차원의 데이터라고 할 수도 있다. 그러나, 일반적으로 영상처리를 공부하게 되면 밝기 성분만을 가지고 처리하는 경우가 많은데 이는 인간의 눈에 가장

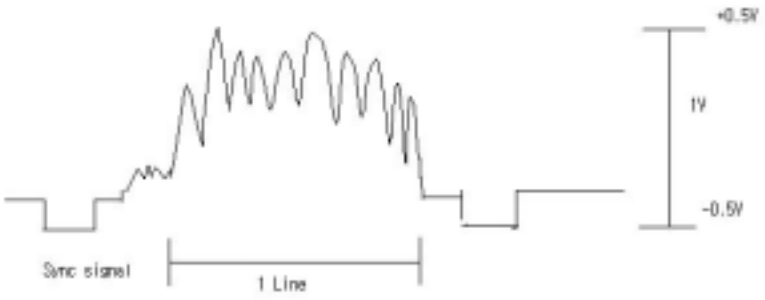

그림 3-8. 영상 시그널.

민감하게 작용하는 요소가 밝기 성분이며 컬러 성분 또한 비슷한 방식으로 처리를 하게 되므로 주로 밝기 성분만을 가지고 실험을 하게 된다. 일반적으로 영상처리 책을 보게 되면 밝기 성분만을 가지고 결과를 보여주는 경우가 많다.

영상 포맷을 이해하기 위해서는 용어에 대한 개념을 정립할 필요가 있다. 일반적으로 CCD 카메라의 영상 포맷에서 NTSC, PAL, SECAM 등의 용어가 있는데 이는 ANALOG COMPOSITE 신호이다. 여기서 ANALOG 라는 부분과 COMPOSITE이라는 단어를 이해하여야 한다. 영상신호는 1Vpp의 신호 (최고 전압과 최저 전압의 차이가 1V)인데 과거에 흑백TV 시절에는 신호는 밝기 성분만을 가진 신호였다. 그러나 컬러TV가 나오고 나서 사람들은 고민을 했다. 새로운 영상 신호가 흑백TV 사용자들에게도 보여야 하고 컬러TV 사용자에게도 보여야 했기 때문이다. 그래서 고민하여 만들어 낸 것이 COMPOSITE신호이다. 컬러와 밝기 신호를 같이 섞어 보내게 된 것이다. 즉, BACKWARD COMPATIBILITY를 가지도록 한 것이다. 신호를 흑백+컬러로 하여 흑백TV 사용자는 흑백만 가지고 TV를 보면 되고 컬러TV 사용자는 흑백 신호에 컬러 신호를 추가하여 TV를 보도록 한 것이다. 그리하여 COMPOSITE신호라 부르게 되었다.

RGB는 쉽게 이해할 수 있다. 모든 색은 RED, GREEN, BLUE 세 가지 빛의 색으로 만들어낼 수 있다. 우리의 모니터 우리의 TV 역시 이 세 가지 전자총으로 색을 만들어 낸다. RGB가 쓰이게 된 것은 우리의 TV와 모니터의 구조 때문에 나오게 된 색의 요소이다.

YCbCr 은 Y + C로 볼 수 있다. 이는 위에서 말한 흑백 + 컬러와 같다. C는 다시 Cb 와 Cr로 나뉘는데 Cb는 COLOR LIKE BLUE 개념으로 이해하면 된다. 하지만 RGB -> YCbCr로 하다보면 Cb는 B요소를 가장 많이 갖게 된다. Cr은 R... 즉 밝기 Y성분과 컬러 Cb, Cr로 구분을 해 놓은 것이라고 생각하면 되는데 이렇게 나오게 된 이유는 위의 컬러TV의 출현과 밀접하다.

CMY 색도는 RGB와 반대의 개념이다. R<->C G<->M B<->Y 각각 반대의 색이다. 빛을 더하면 백색이 물감을 더하면 검게 된다는 얘기다. 이 개념이 나오게 된 이유는 프린터의 출현과 같다. 프린터는 물감 혹은 비슷한 약품을 쓰게 되는데 이는 물감을 서로 섞는다는 개념과 같다. 프린터는 섞으면 섞을 수록 검어진다 이는 TV와 반대 개념이 되기 때문에 색도 또한 반대의

개념으로 나오게 되었다.

이 외에도 HSY, HSI 등등 YUV.. 등등.. 약간씩 값이 다른 색도계가 사용되며, 이는 개념은 거의 비슷하고 용도에 따라 약간씩 값이 틀리다.

CCIR601, CCIR656은 A/D 변환했을 시 나오는 데이터 포맷을 말한다. 딱 맞는 말이라고 할 수 없지만 YCbCr 4:2:2 포맷이라고 생각하면 거의 맞을 것이다. 이는 위에서 설명했듯이 간상세포가 원추세포보다 많기 때문에 사람은 밝기 성분에 민감하다. 따라서 색상성분의 값은 밝기 성분에 비해서 적어도 사람은 인지하지 못 하기 때문에 색상성분 데이터를 빼내어도 별 차이를 못 느끼기 때문에 사용한다.

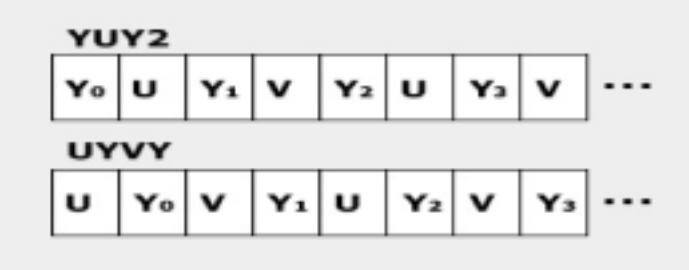

그림 3-9. YC422 Format Structure.

Primary surface가 RGB 15/16/23/32 bit로 구성되는 것과 같이, Overlay surface는 YUV family로 구성된다. Windows 체계에서는 모든 surface가 RGB를 제외하고는 4-byte의 FOUR Character Code에 의해 표현된다. FOURCC는 Microsoft에 의해 다양한 형태의 비디오 data를 구별하기 위해 VFW(Video for Window)의 한 부분으로 정의되었으며, MS에서는 VFW에서 비디오 포맷과 픽셀 layout을 고유하게 정의하기 위해 사용한다. Four CC는 윈도우의 디지털 비디오 포맷 표준을 가리키며, 예를 들어 정지영상의 포맷인 JPEG의 FOURCC는 'JPEG' 이다.

Windows 시스템에서는 YUY2 와 UYVY 같은 FOURCC가 표준 포맷으로 사용된다. 이 포맷의 구조는 YUV 의 비율이 4:2:2로 구성되며 ComArt 보드들은 YUY2와 UYVY 포맷을 캡쳐 및 live display 포맷으로 적용하고 있다. 따라서, VFW API를 이용하면 이들 FOURCC의 포맷간의 비디오 data 포맷을 변환할 수 있으며 본 연구에서도 캡쳐보드의 raw data인 UYVY 포맷을 bitmap으로 변환하여 사용하도록 한다. 일반적으로 YC422 포맷과 RGB 포맷의 관계는 다음과 같다.

$$R = Y + 1.402 \times Cr$$
$$G = Y - 0.34414 \times Cb - 0.71414 \times Cr$$
$$B = Y + 1.772 \times Cb$$

3.3.4 캡쳐 모듈의 구현

영상 캡쳐 보드인 ComArt사의 HICAP 및 MIS/MID 시리즈의 SDK는 카드 구동을 위한 다수의 함수들과 영상 처리를 위한 함수들로 구성되어 있으며 그림 3-10과 같다. 라이브러리의 설정에서 설명한 바와 ComArt2D.lib 파일을 project에 링크시키게 되면 이러한 함수들을 자유롭게 활용할 수 있게 되며, 각각의 함수와 구조체 및 매크로에 대한 prototype은 ComArt2D.h 파일에 정의되어 있으므로 프로그램 작성 시에 참조하면 된다.

그림 3-10에서 보는 바와 같이 ComArt 시리즈의 캡쳐 보드에 사용되는 함수들은 유사한 함수명을 가지고 있으며 카드 모델에 따라 모델명을 함수명 앞에 바꿔주면 되도록 되어 있다. 예를 들어 캡쳐보드 확인 함수는 32 bit unsigned character type인 dw_CArTBDSearch() 인데, HICAP-50 보드용은 HICAP50_dw_CArTBDSearch()이다.

3.3.4.1 HICAP-50의 초기화

HICAP-50 영상 캡쳐 보드의 초기화 단계에서 실행되어야 하는 필수 함수들은 다음과 같다.

그림 3-10. 영상 캡쳐링을 위한 SDK 루틴 구조.

HICAP50_dw_CArTBDSearch(...);
HICAP50_dw_CArTBDDetainInfo(...);
HICAP50_dw_BoardOpen(...);
HICAP50_dw_BoardInit();
HICAP50_bl_CaptureGroupSliceCtrl(...);
HICAP50_bl_CAPTUREFRAME_CONTROL(...);
HICAP50_dw_CaptureVideoTransferON();

HICAP-50 캡쳐 보드를 이용한 영상 캡쳐에서 보드의 초기화에 사용되는 함수와 구조체들의 특성에 대해 아래에 정리하였다.

표 3-1. CArTBDSearch.

함수명	HICAP50_dw_CArTBDSearch
용 도	PCI 슬롯에 있는 ComArt 2nd 보드를 찾는다.
원 형	DWORD WINAPI XXX_dw_CArTBDSearch(tCArTBDList *OUT_pCArTBL);
인 자	*OUT_pCArTBL memorizes the return value for detection
리턴 값	31→No Comart Board detected 32→ComArT2D Driver Open Fail 33→ComArT2D Driver Version Invalid
비 고	tCArTBDList 구조체 리턴값을 이용해 PC에 장착된 ComArt 보드의 수를 확인할 수 있다.

예 제	DWORD dwReturnV;
	tCArTBDList tCATBDINF; //구조체 (SDK manual 참조!)
	dwReturnV = HICAP50_dw_CArTBDSearch(&tCATBDINF);
	tCATBDINF.dwTNumB ← HICAP50의 경우 2 가 return되어 있으면 됨!

표 3-2. CArTBDDetailInfo.

함수명	HICAP50_dw_CArTBDDetailInfo
용 도	캡쳐보드의 하드웨어 정보를 가져온다.
원 형	DWORD WINAPI XXX_dw_CArTBDDetailInfo(DWORD IN_dwLIstOrder, TCArTBDDetailInfo *OUT_pDTINF);
인 자	IN_dwLIstOrder : XXX_dw_CArTBDSearch()를 통해 확인된 보드들의 순위. 현재는 한 PC 당 한 개의 보드가 허용되므로 항상 '0'으로 설정. *OUT_pDTINF : ComArt 보드의 세부 하드웨어 정보 저장.
리턴 값	31→dw_CArTBDSearch()가 이루어지지 않았음. 32→ComArT2D Driver Open Fail 33→ComArT2D Driver Version Invalid 34→MIS & MID 시리즈에서 H/W Micro code 쓰기 실패. 35→MIS & MID 시리즈에서 H/W Micro code 없음.
비 고	에러가 발생치 않으면 1회 만 call하면 됨. Micro code 명은 "DQ16XX.CAT" , "MID16VXX.CA"

예 제	DWORD dwReturnV;
	tCArTBDDetailInfo tBDINF; // 구조체 (SDK manual 참조!)
	dwReturnV = HICAP50_dw_CArTBDDetailInfo(0, &tBDINF);
	tBDINF.dwHWRevisionID ← H/W revision id 정보가 저장되어 return됨!

표 3-3. BoardOpen.

함수명	1. HICAP50_dw_BoardOpen 2. HICAP50_dw_BoardOpenEx
용 도	DLL에서 캡쳐 보드를 위한 메모리를 open. 1. 캡쳐링을 위한 각 channel 당 size 설정 - SCHEDULE SET 2. Image size를 640, 320, 160 scale로 설정
원 형	1. DWORD WINAPI XXX_dw_BoardOpen(DWORD IN_dwNTSC, DWORD IN_dwListOrder, DWORD *IN_dwCHSizePerSchedule); 2. DWORD WINAPI XXX_dw_BoardOpenEx(DWORD IN_dwNTSC, DWORD IN_dwListOrder, DWORD *IN_dwCHSizePerSchedule, DWORD IN_dwExtwidthFlag);

인 자	IN_dwNTSC :
	1→NTSC signal, 0→PAL signal
	IN_dwListOrder :
	XXX_dw_CArTBDSearch()를 통해 확인된 보드들의 순위.
	현재는 한 PC 당 한 개의 보드가 허용되므로 항상 '0'으로 설정.
	*IN_dwCHSizePerSchedule :
	각 채널에 캡쳐되는 이미지의 크기 설정.
	SCHEDULE_DISABLE_CH; //채널 사용안함.
	SCHEDULE_FULLSIZE_CH; // 704*480 (640*480)
	SCHEDULE_FULLQUADSIZE_CH; // 704*240 (640*240)
	SCHEDULE_QUADSIZE_CH; // 352*240 (320*240)
	SCHEDULE_QCIFSIZE_CH; // 176*120 (160*120)
	IN_dwExtwidthFlag :
	HICAP50_dw_BoardOpenEx() 사용하는 경우 '1'로 설정하며 640*480 scale의 size로 설정됨.

리턴 값	31→Never called XXX_dw_CArTBDSearch() 등의 많은 error code가 return 되며 SDK manual 참조.
비 고	캡쳐를 위한 메모리의 수는 3배가 필요하다. 즉, PAL 방식 16bit full size 인 경우, 704*480*16bit*3=2.0MB 가 되므로 메모리 할당에 주의해야 함.

예 제	dwCHSCD[0]=SCHEDULE_FULLSIZE_CH;
	dwCHSCD[1]=SCHEDULE_FULLQUADSIZE_CH;
	dwCHSCD[2]=SCHEDULE_QUADSIZE_CH;
	dwCHSCD[3]=SCHEDULE_QCIFSIZE_CH;
	
	bool pal=false;
	if (pal == true)
	dwReturnV= HICAP50_dw_BoardOpen(dwNTSC,0,&dwCHSCD[2]);
	else
	dwReturnV= HICAP50_dw_BoardOpenEx(dwNTSC,0,&dwCHSCD[0],1);

표 3-4. BoardInit.

함수명	1. HICAP50_dw_BoardInit 2. HICAP50_dw_BoardInitEx
용 도	ComArt 보드의 reset operation
원 형	1. DWORD WINAPI XXX_dw_BoardInit(void); 2. DWORD WINAPI XXX_dw_BoardInitEx(DWORD IN_dwACCamera);
인 자	IN_dwACCamera 0 : Stop filtering for AC Camera 1 : Start filtering for AC Camera
리턴 값	31→Never called XXX_dw_BoardOpen() 등의 많은 error code가 return 되며 SDK manual 참조.
비 고	Reset operation for H/W Decision of AC/DC camera filtering. AC Camera 사용시 IN_dwACCamera = 1로 설정.

예 제	dwReturnV=HICAP50_dw_BoardInit();
	if (dwReturnV != NOTBOOL_FUNCTION_SUCCESS)
	{
	MessageDlg("Initializing Error!",mtConfirmation,TMsgDlgButtons() << mbOK,0);
	finalize();

표 3-5. CaptureGroupSliceCtrl.

함수명	HICAP50_bl_CaptureGroupSliceCtrl
용 도	하나의 ADC 그룹에서 카메라별 priority 설정
원 형	BOOL WINAPI XXX_bl_CaptureGroupSlice(DWORD IN_dwGRPID, DWORD IN_dwCHA, DWORD IN_dwCHB, ...);
인 자	IN_dwGRPID ADC Group # : 0～1 (그림 3-11 참조) IN_dwCHA channel A 의 priority percent IN_dwCHB channel B 의 priority percent
리턴 값	BOOL 1 : Function success 0 : Function fail
비 고	ADC 그룹별 delay time 설정. total sum은 1～100%. disabled channel은 0으로 설정

예 제	bool blReturnV;
	blReturnV=HICAP50_bl_CaptureGroupSliceCtrl(0,50,50,0,0,0,0,0,0); //no delay
	blReturnV=HICAP50_bl_CaptureGroupSliceCtrl(1,50,50,0,0,0,0,0,0); //no delay

표 3-6. CAPTUREFRAME_CONTROL.

함수명	HICAP50_bl_CAPTUREFRAME_CONTROL
용 도	하나의 ADC 그룹별 delay capturing 설정
원 형	BOOL WINAPI XXX_bl_CAPTUREFRAME_CONTROL(DWORD IN_dwADCNumB, DWORD IN_dwDelaymSec);
인 자	IN_dwADCNumB ADC Group Number : 0～1 IN_dwDelaymSec : 지연시간 (msec)
리턴 값	BOOL 1 : Function success 0 : Function fail
비 고	ADC 그룹별 delay time 설정. Capturing priority와 같이 사용하면 캡쳐 프레임을 저하시킬 수 있음. delay time은 가급적 작게 주도록 해야 함.

예 제	bool blReturnV;
	blReturnV=HICAP50_bl_CAPTUREFRAME_CONTROL(0,0);//no delay
	blReturnV=HICAP50_bl_CAPTUREFRAME_CONTROL(1,0);//no delay

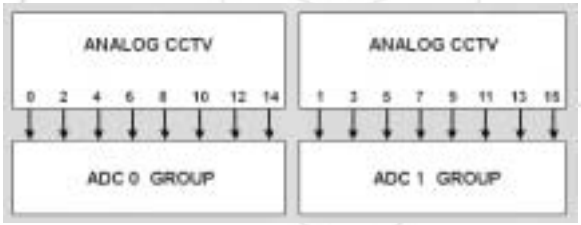

그림 3-11. HICAP-50 ADC 그룹 구성.

표 3-7. CAMERA_SIGNAL.

함수명	HICAP50_bl_CAMERA_SIGNAL
용 도	신호가 들어오는 카메라 상태 확인
원 형	BOOL WINAPI XXX_bl_CAMERA_SIGNAL(DWORD *OUT_dwCAMERA_SIGNAL);
인 자	OUT_dwCAMERA_SIGNAL 16 item을 가진 DWORD type의 array address
리턴 값	BOOL 각 채널별로 0 : No signal 1 : Camera signal
비 고	각 channel 의 signal 확인. 초당 4회까지 확인 가능.

예 제	BOOL blReturnV;
	DWORD dwCAM[MAX_INPUT_CHANNEL];
	blReturnV=HICAP50_bl_CAMERA_SIGNAL(dwCAM);
	if (blReturnV==FALSE)
	{MessageDlg("Camera Signal Error!",mtConfirmation,TMsgDlgButtons() << mbOK,0);
	finalize();}
	CheckBox1->Checked = dwCAM[0]; //camera 신호를 알려주는 check box
	CheckBox2->Checked = dwCAM[1]; //
	CheckBox3->Checked = dwCAM[2]; //
	CheckBox4->Checked = dwCAM[3]; //

표 3-8. CaptureVideoTransferON.

함수명	1. HICAP50_dw_CaptureVideoTransferON 2. HICAP50_dw_CaptureVideoTransferOFF
용 도	1. Capture Video DMA On 2. Capture Video DMA Off
원 형	DWORD WINAPI XXX_dw_CaptureVideoTransferON(void); void WINAPI XXX_dw_CaptureVideoTransferOFF(void);
인 자	None
리턴 값	31→Never called dw_BoardInit() before
비 고	캡쳐를 위해 DMA channel을 On or Off

예 제	dwReturnV=HICAP50_dw_CaptureVideoTransferON();
	if (dwReturnV!=NOTBOOL_FUNCTION_SUCCESS)
	{
	MessageDlg("Capture Video Transfer Error!",mtConfirmation,TMsgDlgButtons() << mbOK,0);
	finalize();
	return false;
	}

이상과 같은 함수들은 순차적으로 실행되어야 하며, 캡쳐 보드 초기화 모듈에서 1회만 호출되어야 한다. 이들 함수가 성공적으로 실행되면 이제 캡쳐 보드로 입력되는 영상신호가 버퍼에 저장되기 시작하며, 버퍼에 저장된 영상을 메모리로 로드할 준비가 완료되었다고 이해하면 된다.

3.3.4.2 영상 캡쳐링

영상을 캡쳐하는 기능에 관련하는 함수는 버퍼에 저장된 최근 영상의 메모리 주소를 확보하여 캡쳐된 영상 data를 가져오는 작업을 수행한다. 영상 캡쳐링 구현은 각 채널별로 멀티 태스킹으로 진행되어야 CPU의 움직임에 관계없이 채널별 동영상 캡쳐가 구현된다. 따라서, 이후에 기술하는 캡쳐링과 영상 포맷 변환 및 영상 표시 단계는 thread programming 기법으로 구현하게 된다.

영상 캡쳐를 위해 사용되는 함수의 특성은 표 3-9에 요약하는 바와 같다.

3.3.4.3 영상 포맷의 변환

전술한 바와 같이 HICAP-50의 기본적인 캡쳐 data는 YC422 포맷으로 이루어져 있다. 이것은 YCbCr 4:2:2 format이며, 4:2:2란 format은 위에 말한 바와 같이 시각의 특성을 이용하여 데이터를 줄이는 방식인데 이 방식은 통상적으로 MPEG에서 사용하는 format이다. MPEG의 경우 4:4:4, 4:2:2, 그리고 4:2:0 format을 사용한다.

캡쳐 보드의 SDK 함수 HICAP50_dw_LockWait4GetImageAddr()를 통해 불러오는 영상 data는 FOURCC 'UYVY' raw data로서 image의 header 정보가 포함되어 있지 않다. 따라서, 이 포맷을 감시 시스템에 사용하기 위해서는 bitmap으로 변환시켜야 하며 bitmap header를 포함시켜 한다. Microsoft의 platform SDK인 VFW API에는 이러한 작업을 수행하기 위한 다양한 함수와 구조체를 제공하고 있다. 본 연구에서는 VFW SDK의 'ICImageDecompress' 함수를 이용하여 캡쳐보드에서 획득된 UYVY raw data를 bitmap 으로 변환하고 아래에 기술하는 ImageEn component의 영상 source로 설정함으로써 감시시스템에 활용할 수 있게 된다.

UYVY 포맷을 Bitmap으로 변환시키는 작업은 다음과 같은 단계로 진행된다.

1. 'ICOpen' 함수를 이용해 decompressor를 open 한다.
2. 'BITMAPINFO' 구조체의 bmiHeader를 설정하며 이때 image의 크기 및 bit count 같은 정보를 정의하고, biCompresssion은 'mmioStringToFOURCC' 함수를 이용해 "UYVY"로 설정한다.
3. HICAP50_dw_LockWait4GetImageAddr()를 이용해 UYVY format의 영상 data가 저장된 주소를 불러와서 Byte array에 copy 한다.
4. 'ICImageDecompress() 함수를 이용해 UYVY 포맷을 uncompress 하며, DIB (Device Independent Bitmap) 형태의 data로 변형된다. 이후, uncompress된 DIB data의 memory pointer를 가져오기 위해 'GlobalLock' 함수를 사용한다.
5. DIB 의 image data를 다른 memory로 copy한 후 'GlobalFree' 함수를 이용해 DIB를 풀어준다.

표 3-9. LockWait4GetImageAddr.

함수명	1. HICAP50_dw_LockWait4GetImageAddr HICAP50_dw_LockWait4GetImageAddrEx 2. HICAP50_dw_UnLockWait4GetImageAddr 3. HICAP50_dw_UnLockWait4GetImageAddrEx
용 도	1. 버퍼를 lock하고 가장 최근 캡쳐된 이미지의 address를 가져옴. 최근에 캡쳐된 이미지가 없으면 이미지를 캡쳐할 때까지 대기함. 2. main thread인 경우 lock 된 버퍼를 unlock 함. 3. main thread 가 아닌 경우 캡쳐 thread 종료시 버퍼의 lock 작업을 취소함.

원 형	DWORD WINAPI XXX_dw_LockWait4GetImageAddr(DWORD IN_dwChannel, DWORD *IN_dwImageAddr);
	DWORD WINAPI XXX_dw_LockWait4GetImageAddrEx(DWORD IN_dwChannel, DWORD *IN_dwImageAddr, DWORD IN_dw4CC, DWORD IN_dwBLOCKFlag, DWORD IN_dwWaitmSec);
	void WINAPI XXX_UnLockWait4GetImageAddr(DWORD IN_dwChannel);
	void WINAPI XXX_CancelLockWait4GetImageAddr(DWORD IN_dwChannel);

인 자	IN_dwChannel : channel number *IN_deImageAddr : 받아오게 되는 UYVY type으로 저장된 이미지 버퍼의 주소. ⇒ 실질적인 영상 data 저장 주소! IN_dw4CC : MKFOURCC() 4BYTE CODE MKFOURCC('Y','U','Y','2') 또는 MKFOURCC('U','Y','V','Y') IN_dwBLOCKFlag : 캡쳐 상태에 따른 CPU Blocking 설정 0 → 캡쳐 할 때까지 cpu blocking. 1 → 캡쳐 안되어 있으며 IN_dwWaitmSec 설정 시간동안 대기. IN_dwWaitmSec : 캡쳐 영상이 없으면 대기하는 시간(msec)
리턴 값	31→Never called dw_CaptureVideoTransferON() before 등 다수의 error.
비 고	thread type 으로 프로그램이 구동시에는 Non-blocking type은 CPU를 느리게 할 수 있음. 일반적으로 blocking 하는 것이 효율적임.

예 제	//Capturing Routine!!
	dwReturnV=HICAP50_dw_LockWait4GetImageAddr(MyChannel, &dwADDR);
	//LockWait4GetImageAddrEx의 경우⇒, MKFOURCC('U','Y','V','Y'), 0, 10);
	if(dwReturnV!=NOTBOOL_FUNCTION_SUCCESS)
	MessageDlg("Capturing Error!",mtConfirmation,TMsgDlgButtons() << mbOK,0);
	
	HICAP50_no_UnLockWait4GetImageAddr(MyChannel); //main thread
	
	HICAP50_no_CancelLockWait4GetImageAddr(MyChannel); //other thread

6. BITMAPFILEHEADER 구조체를 이용하여 생성하고자 하는 bitmap 파일의 header 부분을 생성한다. 이때, bitmap file의 size는 image의 크기와 BITMAPINFO의 크기 및 BITMAPFILEHEADER의 크기가 합쳐진 크기가 되도록 하며, bfOffBits는 BITMAPINFO의 크기와 BITMAPFILEHEADER의 크기가 합쳐진 크기가 되도록 한다.
7. Memory stream type의 변수를 설정하고 BITMAPFILEHEADER 와 BITMAPINFOHEADER 및 uncompressed image data를 저장한다.
8. Graphics:TBitmap type의 Bitmap object를 생성하고 'LoadFromStream()' method를 이용하여 memory stream을 불러온다.

이상과 같은 단계를 거치면 캡쳐보드를 통해 UYVY 포맷으로 캡쳐된 raw image가 bitmap header 정보를 가진 완벽한 Bitmap 형태로 변경되며 사용자의 필요에 따라 편리하게 사용할 수 있게 된다. 위에서 설명한 영상 캡쳐 모듈을 볼랜드 C++ 빌더 개발 환경에서 구현한 프로그램 리스트는 다음과 같으며, ICImageDecompress, ICOpen, mmioStringToFOURCC 등의 Windows SDK 함수들의 특성은 MSDN (http://msdn.microsoft.com)을 참고하면 된다.

```
        //---- Place thread code here ----
                        DWORD                   dwAddr0;
        unsigned long           dwReturnV;
        HANDLE                  hDIB;
        LPBYTE                  lpPtr, bImgData  ;

              IUYVY = mmioStringToFOURCC("UYVY",0);
      hCompr = ICOpen(0, 0, ICMODE_DECOMPRESS);

        /*if (hCompr == 0)
      {MessageDlg("ICOpen Error!",mtConfirmation,TMsgDlgButtons() << mbOK,0 );
        return ;
        } */
        typImgInfo.bmiHeader.biSize             = sizeof(BITMAPINFO);
        typImgInfo.bmiHeader.biWidth            = 640;
        typImgInfo.bmiHeader.biHeight           = 480;
        typImgInfo.bmiHeader.biPlanes           = 1;   //must be 1
        typImgInfo.bmiHeader.biBitCount         = 24;
        typImgInfo.bmiHeader.biCompression      = IUYVY;
        typImgInfo.bmiHeader.biSizeImage        = typImgInfo.bmiHeader.biWidth * typImgInfo.bmiHeader.biHeight * 3;//3 +128;


             bfh.bfType = 0x4D42;
            bfh.bfSize = typUncompressedInfo.bmiHeader.biSizeImage + typUncompressedInfo.bmiHeader.biSize +
sizeof(BITMAPFILEHEADER);
            bfh.bfReserved1 = 0;
            bfh.bfReserved2 = 0;
            bfh.bfOffBits = typUncompressedInfo.bmiHeader.biSize + sizeof(BITMAPFILEHEADER);
```

그림 3-12. BitmapInfo Header 및 BimapFileInfo Header 생성 루틴.

```
TMemoryStream __thread *Stream1 = new TMemoryStream();
Bitmap1 = new Graphics::TBitmap();

//m_lpbRGB = new  BYTE[dwImageWidth * dwImageHeight * 3 + 4];   // 8.0F 3BYTE INDEXING,. VIEW HEADER or DOC


//bImgData = new BYTE[typImgInfo.bmiHeader.biSizeImage];


//Capturing Routine!!


dwReturnV=HICAPSG_dw_LockWait4GetImageAddr(MyChannel, &dwADDR);//, MAKEFOURCC('U','Y','V','Y'), 0, 10);


if(dwReturnV!=NOTBOOL_FUNCTION_SUCCESS)
MessageDlg("Capturing  Error!",mtConfirmation,TMsgDlgButtons() << mbOK,0 );

bImgData=(LPBYTE)dwADDR;
//Bi=(LPBYTE)dwADDR;


//CopyMemory(bImgData, (LPBYTE)dwADDR, typImgInfo.bmiHeader.biSizeImage);



hDIB = ICImageDecompress(hCompr, 0, &typImgInfo, bImgData, NULL);
IPtr = (LPBYTE)GlobalLock(hDIB);
if (IPtr == NULL) {
        free(bImgData);
        MessageDlg("Capture pointer NULL!",mtConfirmation,TMsgDlgButtons() << mbOK,0 );
GlobalFree(hDIB);
        long test = ICClose(hCompr);

}

CopyMemory(&typUncompressedInfo, (BITMAPINFO *)IPtr, sizeof(BITMAPINFOHEADER));

bUncompressed = new BYTE[typUncompressedInfo.bmiHeader.biSizeImage];

IPtr += typUncompressedInfo.bmiHeader.biSize ;

CopyMemory(bUncompressed, IPtr, typUncompressedInfo.bmiHeader.biSizeImage);
GlobalFree(hDIB);
//        delete bImgData;
```

그림 3-13. 캡쳐 및 Uncompress 루틴.

```
        Stream1->SetSize(hfh.bfSize);
        //stream1->Seek(0, 0);

        Stream1->Write(&hfh, sizeof(BITMAPFILEHEADER));

        Stream1->Write(&typUncompressedInfo.bmiHeader, sizeof(BITMAPINFOHEADER));

        Stream1->Write(bUncompressed,typUncompressedInfo.bmiHeader.biSizeImage );
        Stream1->Seek(0,soFromBeginning);

        delete[] bUncompressed;
            Bitmap1->LoadFromStream(Stream1);//MyStream);


/*int right = typUncompressedInfo.bmiHeader.biWidth;
int bottom = typUncompressedInfo.bmiHeader.biHeight;
Canvas->Brush->Bitmap = Bitmap1;
Canvas->FillRect(Rect(100,100,right,bottom));*/
```

그림 3-14. Bitmap object 생성 루틴.

3.3.4.4 다중 Thread의 이해

Windows Application의 개발 개념은 Win95와 WinNT에 오면서 멀티 태스킹을 필요로 하는 상황으로 바뀌었다. application은 default로 하나의 실행 thread를 가지고 있으나, 필요한 경우 많은 실행 thread를 가질 수 있다. 이 경우, 각각의 thread는 작은 프로그램과 같이 작동한다.

```
#include <vcl.h>
#pragma hdrstop

#include "CaptureThread.h"
#include "MainWin.h"
#include "CaptureCard.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall Unit2::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------

__fastcall CaptureThread::CaptureThread(bool CreateSuspended)
        : TThread(CreateSuspended)
{
        FreeOnTerminate = true;
        Priority = tpNormal;
                MyChannel = CaptureCard->channel;

}
//---------------------------------------------------------------------------
void __fastcall CaptureThread::Execute()
{
        //---- Place thread code here ----
                        DWORD                   dwADDR;
        unsigned long           dwReturnV;
        HANDLE                  hDIB;
        LPBYTE                  lPtr, bImgData  ;

                IUYUY = mmioStringToFOURCC("UYUY",0);
      hCompr = ICOpen(0, 0, ICMODE_DECOMPRESS);
```

그림 3-15. Capture Thread module 구현.

운영체제는 자동적으로 thread들을 공유해서 하나의 thread가 CPU를 혼자 점유하지 않도록 조절한다.

전술한 바와 같이, 본 연구에서 구현하는 핵물질 감시 시스템은 최대 16 채널의 동영상을 동시에 처리할 수 있도록 해야 한다. 따라서, 각 채널이 독립적으로 영상을 캡쳐하고 화면에 표시하는 작업이 동시에 멀티 태스킹으로 진행되도록 프로그램을 구현해야 한다. 이와 같은 작업을 가능하게 하는 개념이 thread programming 개념이다.

본 연구의 개발 환경인 볼랜드 C++ 빌더에서는 wizard를 이용하여 기본적인 구조를 가진 thread object를 생성시켜 project에 포함시킬 수 있다. wizard를 이용해 생성된 thread object는 크게 constructor() 부분과 execute() 부분으로 이루어지며 각 부분에서 프로그램 구현을 위해 이해해야 하는 항목들에 대한 기본적인 개념을 아래에 요약한다 (그림 3-15. 참조). 참고로 thread 관련 함수 또는 변수는 개발 환경에 따라 상의할 수 있으나 기본적인 개념은 동일하며 본 연구에서는 볼랜드 C++ 빌더 환경을 위주로 설명한다.

1) Constructor() 부분

◎ Priority

Priority는 constructor 부분에서 설정되어야 하며, application에서 thread object가 cpu를 점유하는 우선 순위를 나타낸다고 할 수 있다. 기본적으로 tpNormal을 설정하며, tpIdle부터 tpTimeCritical 까지 우선 순위를 설정할 수 있다.

◎ FreeOnTerminate

FreeOnTerminate property도 constructor 부분에서 설정되어야 하며, operation이 종료된 후 thread가 free되는 방식을 설정한다. 기본적으로 thread의 operation이 종료되면 자동적으로 free되는 독립적인 경우 true를 설정하고, 다른 thread로부터 값을 받고 free되는 종속적인 경우에는 false로 설정한다.

2) Execute() 부분

실질적인 thread object의 operation 부분이 구현되는 것이 Execute() 부분이다. 이 부분을 구현할 때에는 특히 memory 관리에 주의해야 하며 동일한 memory를 동시에 access, overwrite 하지 않도록 구현해야 한다.

◎ Synchronize()

볼랜드 환경에서 제공하는 VCL(Visual Component Library) object의 사용은 thread-safe 하지 않음에 주의하여야 한다. Synchronize는 VCL thread에 접근하는 방법을 제공하는 함수이다. VCL thread에 접근하고자 하는 경우에는 별도의 접근 함수를 생성하고 Synchronize(함수명)로 call함으로써 OS의 process 대기 열에 올려놓게 되며, OS가 적절한 시기에 Synchronize(함수명)에 의해 call된 함수를 실행하게 된다.

VCL object들이 모두 thread-unsafe한 것은 아니며 다음 component들은 thread-safe 하다.

- Data access components
- Graphic objects (Canvas 등.)
- TList 대신 TThreadList 사용하는 경우

◎ Thread variable

Global variable은 thread 사이의 communication에 유용한 도구로 사용될 수 있다. 만일 thread 내부에서는 global 이지만 같은 thread의 다른 instance에서는 보이지 않는 variable을 사용해야 하는 경우에는 __thread modifier를 사용한다.

ex) int __thread x;

pointer와 function variable은 thread variable이 될 수 없으며, AnsiStrings와 같은

copy-on-write 의미형 변수 type도 thread variable이 될 수 없다. 또한, runtime initialization 또는 finalization을 필요로 하는 program element도 __thread type으로 정의될 수 없다.

◎ Thread termination

thread의 종료 확인은 *Terminated* property를 통해 이루어지며 thread 외부에서 해당 thread를 종료시키고자 하는 경우에도 Terminate() 함수를 call 함으로써 *Terminated* property를 true로 변경한다. 일반적으로 while loop를 사용해 thread를 구동하도록 하며 *Terminated* property를 check 하도록 구현한다.

ex) while (!Terminated)

◎ Writing clean-up code

thread의 실행 종료시에 처리해야 되는 clean-up code들은 OnTerminate event handler 코드 부분에 구현하도록 한다.

◎ Avoiding simultaneous access

전술한 바와 같이 thread 모듈의 구현에는 memory 관리에 주의하여야 하며 볼랜드 C++ 빌더 환경에서는 다음과 같은 3가지의 방법을 사용할 수 있다.

(1) Locking objects

일부의 VCL object는 자체적인 locking 기능을 가지고 있으며 다른 instance에 의해 object가 사용되는 것을 방지한다. 예를 들어 canvas object (TCanvas 및 상속자)는 Lock method를 가지고 있어 Unlock method가 call 될 때까지 다른 thread의 접근을 방지한다.

(2) Critical sections

Critical section의 개념은 한번에 하나의 thread가 통과될 수 있도록 허용하는 gate의 개념으로 이해할 수 있다. 즉, 두 thread가 data variable을 공유하여 사용하고자 할 때, 두 thread가 동시에 값을 변경시키려고 하면 문제가 발생할 수 있다. 따라서, 한번에 한 thread만 global memory에 접근할 수 있도록 하는 것이 Critical section의 역할이다. Critical section을 사용하기 위해서는 *TCriticalSection*의 instance를 생성하고 공유 data에 값을 쓰기 전에 critical section을 설정하면, 다른 thread가 공유 data에 접근할 수 없다는 것이 확보되며 충돌 없이 독자적인 data 독점이 가능하게 되는 것이다.

*TCriticalSection*은 *Acquire*와 *Release* method를 사용할 수 있으며, 각각의 critical section은 보호하고자 하는 global memory와 연계되어 있다. 따라서, 모든 thread는 global memory에 접근하고자 할 때는 *Acquire* method를 사용한 후 접근하게 되고 사용이 종료되면 *Release* method로 풀어주어야 한다.

(3) Multi-read exclusive-write synchronizer

*TMultiReadExclusiveWriteSynchronizer*는 global memory의 reading은 잦은 반면 writing은 거의 필요치 않는 경우에 사용한다. 즉, writing을 사용하지 않는다면 multi-reading을 허용한다는 개념이다.

*TMultiReadExclusiveWriteSynchronizer*를 사용하기 위해서는 보호하고자 하는 global memory와 연계된 *TMultiReadExclusiveWriteSynchronizer* global instance를 생성하고 *BeginRead* method를 call함으로써 다른 thread가 그 memory를 write하지 않고 있음을 담보할 수 있다. *TMultiReadExclusiveWriteSynchronizer*의 사용을 위해 *BeginRead(), BeginWrite(), EndRead(), EndWrite()* method가 제공된다.

◎ WaitFor

WaitFor() method는 thread의 특정 task(또는, procedure)가 완료되도록 대기하는 기법을 구현하는 데 적용된다. thread의 task가 완료되면 true가 return된다. 주의할 점은 thread의 Execute() 함수를 WaitFor의 대상으로 하지 않아야 한다. Execute()는 어떠한 값도 반환하지 않기 때문이다.

3.3.4.5 Thread object의 실행

이상과 같이 thread로 처리되어야 할 모듈의 구현이 완료되면 main thread에서 이 thread의 instance를 생성시켜 구동시키게 된다.

Ex) TMyThread *SecondProcess = new TMyThread(false);

이때 thread의 parameter로 들어가는 bool 값은 thread의 constructor에 정의되는 *CreateSuspended* property를 재 설정하는 값이다. *CreateSuspended* 값이 false 이면 즉시 thread가 실행되며, true이면 *Resume()* method가 실행될 때까지 대기 상태에 들어간다.

이러한 thread는 동일한 프로그램에서 여러 개가 복제·생성되어 동시에 구동될 수 있으나 일반적으로 CPU load와 memory를 고려하여 동시에 16개 이하로 제한하는 것이 권고된다. 또한, thread는 terminate되지 않는 이상 실행 종료에 관계없이 여러 번 시작 중지 될 수 있으며, 정지 시에는 *Suspend()* method를 call 하고 회복 시에는 *Resume()* method를 call 한다.

이상에서 요약한 바와 같은 thread 모듈의 프로그램 리스트를 그림 3-16 에 제시하였다.

3.3.4.6 캡쳐 보드의 정지

이상과 같은 프로그램의 구현을 통해, 캡쳐 보드의 초기화, 영상 캡쳐, 포맷 변환 및 이러한 기능의 thread programming 기법에 대해 고찰하였다. 이와 같은 단계가 완료되면 다음에 설명되는 ImageEn component를 이용하여 감시 시스템에 적합하게 활용할 수 있게 되며, 감시 시스템의 구동이 종료되면 카운터 보드와 캡쳐 보드의 device unloading 작업이 필요하게 된다. 전술한 바와 같이 capture 보드를 통한 영상 획득은 각 channel 별로 각각의 thread가 구동되며 capture 보드를 unloading 하는 작업은 이 thread들의 구동이 완료된 것을 확인한 후 진행되어야

```
/*SlideBright->Min = -128;
SlideBright->Max = 127;
ScrollContrast->Min = 0;
ScrollContrast->Max = 255;       */


 /*ImageEnIO1->StreamHeaders = False;
 ImageEnView1->AutoFit = true; */
 blStop = False;
 //fCount=0;
 //Form2->Timer1->Enabled = True;


 //MessageDlg("Device Loaded!",mtConfirmation,TMsgDlgButtons() << mbOK,0 );
 FormMain->emioCCD3->LoadFromFileJpeg("logo.jpg");
 CaptureThreadList = new TList;

 for(int i=0;i<3;i++)
 {
 channel=i;
 if(dwCAM[i] == 0)
 {

 continue;
 }
     HICAP50_bl_BRIGHTNESSControl(channel,20);
     HICAP50_bl_CONTRASTControl(channel,140);
     trCapture = new CaptureThread(true);
 trCapture->Resume();
 CaptureThreadList->Add(trCapture);
 }
```

그림 3-16. Capture Thread 구동 구현 모듈.

한다. 만약 capture가 진행중인 상태에서 다른 thread가 device를 unloading하는 SDK 함수를 call 하게 된다든가, 이미 device가 unloading 된 상태인데 어떤 thread에서 capture를 시도하게 되면 시스템이 halt 되어 버린다. 캡쳐 보드의 finalize를 위해서는 그림 3-17에서 제시된 바와 같은 함수들이 호출되어야 한다.

```
//---------------------------------------------------------------------------

void __fastcall CaptureThread::Finalize()
{
        //TODO: Add your source code here
        HICAP50_no_CancelLockWaitNGetImageAddr(MyChannel);
        HICAP50_no_CaptureVideoTransferOFF();
        HICAP50_no_BoardClose();

}
```

그림 3-17. Capture Card Finalize 모듈.

3.3.5 ImageEn component의 이해

전술한 바와 같은 단계를 거쳐 영상캡쳐 보드로부터 획득된 raw 영상 data가 bitmap 포맷의 memory stream으로 전환되면 이 bitmap을 화면에 표시하고 처리할 수 있게 된다. 일반적으로는

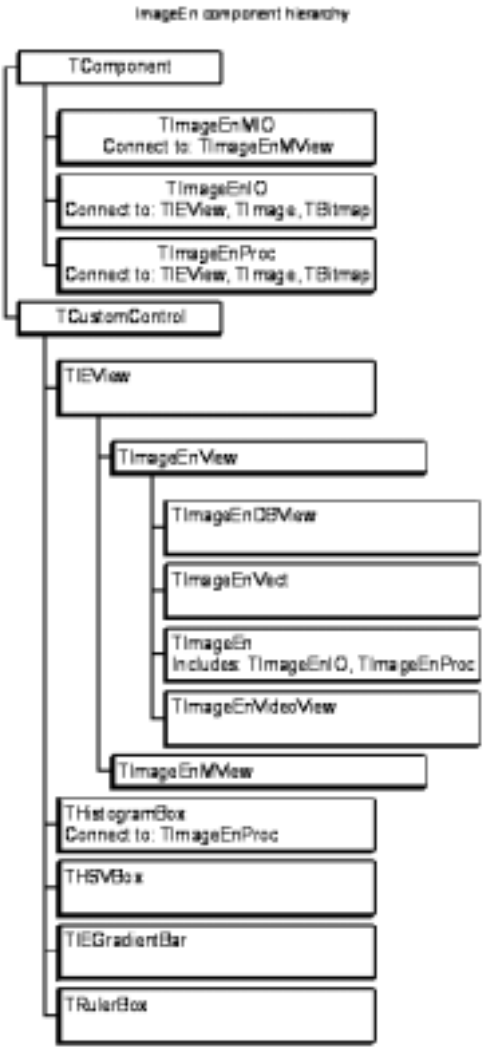

그림 3-18. ImageEn Component Hierarchy.

VFW SDK에서 제공하는 기본적인 멀티미디어 함수들을 이용하여 영상 data를 화면에 표시하고 처리하는 모듈을 개발하게 되지만, 이러한 영상 처리 작업들은 매우 복잡하고 세심한 메모리 관리를 필요로 하므로 잘못하면 전체적인 시스템의 신뢰성을 저하시킬 수 있다. 따라서, 본 연구에서는 이러한 영상 처리작업을 단순화시키고 감시 시스템의 무결성 및 신뢰성을 향상시키기 위해 영상처리 전용 component인 ImageEn을 활용하였다.

ImageEn component는 Delphi 또는 볼랜드 C++ 빌더를 위해 상용으로 제공되는 영상처리 component이다. 이 component는 다양한 포맷의 영상 data를 표시・취급・처리할 수 있는 I/O 함수 및 기본적인 영상 처리・해석 함수들을 포함한 유용한 method들로 구성되어 있으며, 그림 3-18과 같은 구조로 되어 있다.

3.3.5.1 ImageEn 설치

ImageEn component의 설치 방법은 각 개발 환경에 따라 help 파일에 상세히 기술되어 있으며 본 연구에서 사용하는 개발 환경인 볼랜드 C++ 빌더에서의 설치는 다음과 같이 하여야 한다.

- Component->Install packages->Add.
- DPKIECTRLc5.BPL과 DPKIEDBc5.BPL을 선택.
- PKIECTRLc5.BPL과 PKIEDBc5.BPL을 system directory(\windows\system32) 또는 BIN path에 copy.
- ImageEn 파일들이 있는 directory 위치를 환경설정의 "Library path"에 추가.

이상과 같이 component를 설치하면 빌더의 design tool box에 ImageEn 항목이 추가로 생성되며 관련된 control들을 사용할 수 있게 된다. 관련 control들의 사용 기법은 Example directory에 있는 예제 application을 사용해 보면 도움이 된다.

3.3.5.2 ImageEn Components의 활용

(1) Bitmap stream의 화면 표시

ImageEn을 이용하여 bitmap stream을 표시하기 위해서는 *ImageEnIO*와 *ImageEnView*를 Form에 추가하여야 한다. 표시하고자 하는 영상 channel의 수만큼 적당한 위치에 *ImageEnView*를 배치하고 각각의 view에 대한 control을 위해 *ImageEnIO*를 추가한다. *ImageEnIO*의 *AttachedImageEn* property를 해당 *ImageEnView*의 name으로 설정해 주게 되면 *ImageEnIO*를 통해 Form에 출력되는 해당 *ImageEnView*를 control할 수 있게 된다. *ImageEnIO*에 bitmap stream을 연결하여 화면에 출력되게 하는 기법은 TIEBitmap class의 *Assign* method를 이용하게 되며 bitmap stream의 assign 후에는 해당 view를 refresh 시켜 주어야 한다. 이러한 작업은 각 channel 별로 capture thread에서 Main Form을 access 하는 것이므로 전술한 바와 같이 *Synchronize()* 기법으로 진행되도록 프로그래밍 하여야 한다. 본 연구의 감시 시스템 개발을 위해 구현된 bitmap stream의 view 출력 모듈을 그림 3-19 에 제시하였다.

(2) Motion detection

본 연구에서 구현하고자 하는 핵물질 감시 시스템은, 실시간 동영상 표시와 동시에 timer에 의해 5초에 한번씩 영상 비교 및 저장 작업이 수행되도록 하여야 한다. 즉, 이전에 취득된 영상과의 비교를 통해 특정 영역에 일정 수준 이상의 영상 차이가 감지되면 이 영상을 저장하도록 하여 일종의 motion detection 기능이 구현되도록 하는 것이다. 본 연구에서는 이러한 기법 구현을 위해 *ImageEnProc* component의 method을 이용하며 이 component는 *ImageEnView*와 연결된 영상에 대한 image processing method들을 제공하고 있다. 또한, 본 연구에서는 특정 영역에 대한

```
void __fastcall CaptureThread::ImageDisplay()
{
        //TODO: Add your source code here



            switch (MyChannel) {
    case 0 :    FormMain->enioCCD0->IEBitmap->Assign(Bitmap1);
                FormMain->imgCCD0->ReFresh();
                break;
    case 1 :    FormMain->enioCCD1->IEBitmap->Assign(Bitmap1);
                FormMain->imgCCD1->ReFresh();
                break;
    case 2 :    FormMain->enioCCD2->IEBitmap->Assign(Bitmap1);
                FormMain->imgCCD2->ReFresh();
                break;
    //case 3 :    FormMain->enioCCD3->IEBitmap->Assign(Bitmap1);
                //FormMain->imgCCD3->ReFresh();  */
                }

}
```

그림 3-19. ImageEnView 구현 모듈.

```
bool __fastcall TdmDAQ::DoImgProc2(int ccd) //ImageEn->Proc->ComputeImageEquality Method
{
        //TImageEnIO *enioccd = (TImageEnIO *)FormMain->EnIOCCDList->Items[ccd];
        TImageEnView *imgccd = (TImageEnView *)FormMain->EnViewCCDList->Items[ccd];
        TImageEnIO *enioccdOld = (TImageEnIO *)Form1->EnIOCCDList->Items[ccd];
        TImageEnView *imgccdOld = (TImageEnView *)Form1->EnViewCCDList->Items[ccd];
        /*if(CaptureCard->dwCAM[ccd] == 0)
        {
        //enioccd->LoadFromFileJpeg("kaerilogo3.jpg");
        return false;
        }  */
                imgccd->SelectionBase = iesbBitmap;
                imgccd->Select(0,200,380,480,iespReplace);
        if(DataFile->dfData.run == 0)
        {
                imgccd->AssignSelTo(imgccdOld);
                imgccdOld->ReFresh();
                return true;
        }
        Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
        imgccd->CopySelectionToBitmap(Bitmap1);
        imgccd->DeSelect();
        bool motion = imgccdOld->Proc->ComputeImageEquality(Bitmap1,
        psnr_min,psnr_max,mse_min,mse_max,rmse_min,rmse_max,pae_min,pae_max,mae_min,mae_max) ;
        if(motion||(rmse_max<Parameters->RMSE_max))
        {delete Bitmap1;
        return false;
        }
        else
        {
        enioccdOld->Bitmap->Assign(Bitmap1);
        delete Bitmap1;
        imgccdOld->ReFresh();
        return true;
        }

}
```

그림 3-20. ImageEnProc component를 이용한 Motion Detection 모듈.

영상 비교를 위해 *ImageEnView*의 몇 가지 영역 선택 method와, 임시적인 bitmap 저장을 위해 background에서 운영되는 추가적인 Form을 활용하였다. *ImageEnProc* component를 활용한 Motion Detection 모듈을 그림 3-20 에 제시하였다.

ImageEnProc component를 이용한 Motion detection 기법은 다음과 같은 단계로 진행된다.

- Timer에 의해 이 모듈이 call되면 특정 영역을 선택하기 위해 view1의 SelectionBase property를 iesbBitmap으로 설정한다. 즉, bitmap 기준 좌표를 사용한다는 의미이다.
- view1의 Select method를 사용하여 특정 영역을 선택한다.
- run number가 0 인 경우, 즉, 저장된 영상이 없는 경우에는 view1의 AssignSelTo method를 사용하여 선택영역을 background에 있는 view2에 assign한다. 이 view2는 old view 역할을 한다.
- Bitmap instance를 생성하고 view1의 CopySelectionToBitmap method를 이용해 선택영역을 Bitmap에 copy 한다.
- view1을 DeSelect 하여 선택 영역을 제거한다.
- view2->Proc->ComputeImageEquality method를 이용해 Bitmap과 영상 차이를 계산한다. ComputeImageEquality method에 대해서는 아래에 다시 설명한다.
- ComputeImageEquality method에서 return된 값이 일정 수준 이상이면 영상이 변화하였으므로 motion이 감지된 것으로 판단하고, 현재의 Bitmap을 IO를 이용해 view2의 Bitmap으로 assign한 후 true를 반환한다.

◉ function **ComputeImageEquality**(SecondImage:TBitmap;var psnr_min, psnr_max:double; var mse_min,mse_max:double; var rmse_min,rmse_max:double; var pae_min,pae_max:double; var mae_min,mae_max:double):boolean;

ComputeImageEquality method는 현재의 image와 SecondImage와의 동질성을 판별하기 위한 몇 가지 계산 결과를 제공하며, 만일 두 image가 완전히 동일하다면 true 값을 반환한다. 또한, 계산되어 return되는 값은 다음과 같다.

psnr_min,psnr_max : minimum and maximum peak signal to noise ratio (PSNR)

$$PSNR = 20\log_{10}\left(\frac{2^{bits}}{RMSE}\right)dB$$

mse_min,mse_max : minimum and maximum mean squared error (MSE)

$$MSE = (RMSE)^2$$

rmse_min,rmse_max : minimum and maximum root mean squared error (RMSE)

$$RMSE = \sqrt{\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}[f_1(i,j) - f_2(i,j)]^2}$$

pae_min,pae_max : minimum and maximum peak absolute error

mae_min,mae_max : minimum and maximum mean absolute error

(3) 영상 저장

영상 비교를 통한 motion detection 단계에서 영상의 저장이 필요하게 되면, 간단히 ImageEnIO의 *SaveToFileJpeg* method를 통해 JPEG 포맷의 원하는 파일명으로 영상 저장이 이루어 질 수 있다. ImageEnIO를 이용하면 JPEG 이외에도 일반적으로 사용되는 다양한 영상 포맷으로 파일을 생성・처리할 수 있다. 또한, 본 연구에서는 그림 3-21에서 보는 바와 같이 영상 저장 단계에서 Hidden text를 영상에 삽입하도록 하여 영상 자료의 건전성 확인 작업이 가능하도록 하고 있다. 이 작업은 *ImageEnView->Proc->WriteHiddenText* method를 이용해 구현되며, 향후 저장된 영상 파일을 다시 읽을 때 임의로 변경되지 않은 원본임을 확인하는데 사용되게 된다.

```
//---------------------------------------------------------------------------
void TCaptureCard::SaveToFile(int ccd, AnsiString filename)
{
    TImageEnView *imgCCD = (TImageEnView *)FormMain->EnViewCCDList->Items[ccd];
    TImageEnIO *enioCCD = (TImageEnIO *)FormMain->EnIOCCDList->Items[ccd];
    imgCCD->Proc->WriteHiddenText(" 한국원자력연구소 ");
    enioCCD->SaveToFileJpeg(filename);
}
//---------------------------------------------------------------------------
```

그림 3-21. SaveToFileJpeg 영상 저장 모듈.

4. 요 약

본 기술보고서에서는, 디지털 영상캡쳐 카드와 카운터를 이용하여 핵물질 감시 시스템을 개발하기 위한 C++ 구현 기법에 대해 정리하였다. 본 연구에서 구현하고자 하는 핵물질 감시 시스템은 다수의 방사선 계측기와 CCD 카메라 및 방사선 계측기로부터 입력되는 신호를 처리할 수 있는 디지털 카운터 보드와 영상 캡쳐 보드로 구성되어 있다. 감시 시스템의 구현을 위해 현재 국내에서 쉽게 입수할 수 있는 PCI 기반의 영상 캡쳐 카드 및 카운터의 구조적 특성과 SDK 함수의 사용법을 분석・요약하였으며, Borland C++ Builder를 이용한 감시 프로그램 구현 기법에 대해 정리하였다.

본 기술보고서는 차세대 핵연료관리 실증 시설에서의 핵물질 안전조치를 위한 감시 시스템 개발을 위해 활용될 수 있을 것이며, 특히 다양한 실험 환경에서의 감시 시스템 구축을 필요로 하는 연구자들을 위한 참고 자료로 활용될 수 있을 것으로 기대된다.

참고 문헌

1. COMIZOA, COMIZOA SD Series Hardware Manual, 2001.
2. COMIZOA, COMIZOA DAQ System [CP/SD 시리즈] Visual Basic Manual, 2002.
3. (주)컴아트시스템, HICAP, MIG, MID, MIS 사용자 설명서, 2003.
4. (주)컴아트시스템, Software Developer's Kit Manual Ver. 4.3.1.96, 2003.
5. Inprise, Borland C++ Builder 5 Developer's Guide, 2001.
6. 정보문화사, 볼랜드 C++ 빌더 How-To, 1998.

서 지 정 보 양 식			
수행기관보고서번호	위탁기관보고서번호	표준보고서번호	INIS 주제코드
KAERI/TR-2552/2003			
제목 / 부제	디지털 영상캡쳐 카드 및 카운터를 이용한 핵물질 감시 시스템 구현 기술		
연구책임자 및 부서명 (혹은 주저자)	이상윤 (건식공정핵연료기술개발부) Lee, Sang-Yoon		
연 구 자 및 부 서 명	송대용 (건식공정핵연료기술개발부) Song, Dae-Yong 고원일 (건식공정핵연료기술개발부) Ko, Won-Il 하장호 (건식공정핵연료기술개발부) Ha, Jang-Ho 김호동 (건식공정핵연료기술개발부) Kim, Ho-Dong		

출 판 지	대전	발행기관	한국원자력연구소	발행년	2003.7
페 이 지	42p.	도 표	있음(x), 없음( )	크 기	A4
참고사항					
비밀여부	공개(X), 대외비( )		보고서종류	기술보고서	
연구위탁기관			계약 번호		

초록 (15-20줄 내외)

본 기술보고서에서는, 디지털 영상캡쳐 카드와 카운터를 이용하여 핵물질 감시 시스템을 개발하기 위한 구현 기법에 대해 정리하였다. 본 연구에서 구현하고자 하는 핵물질 감시 시스템은 다수의 방사선 계측기와 CCD 카메라 및 방사선 계측기로부터 입력되는 신호를 처리할 수 있는 디지털 카운터 보드와 영상 캡쳐 보드로 구성되어 있다. 감시 시스템의 구현을 위해 현재 국내에서 쉽게 입수할 수 있는 PCI 기반의 영상 캡쳐 카드 및 카운터의 구조적 특성과 SDK 함수의 사용법을 분석·요약하였으며, Borland C++ Builder를 이용한 감시 프로그램 구현 기법에 대해 정리하였다. 본 기술보고서는 차세대 핵연료관리 실증 시설에서의 핵물질 안전조치를 위한 감시 시스템 개발을 위해 활용될 수 있을 것이며, 특히 다양한 실험 환경에서의 감시 시스템 구축을 필요로 하는 연구자들을 위한 참고 자료로 활용될 수 있을 것으로 기대된다.

주제명키워드 (10단어내외): 핵물질 감시시스템, 디지털 영상 캡쳐, 디지털 카운터, 볼랜드 C++ 빌더

BIBLOGRAPHIC INFORMATION SHEET			
Performing Org. Report No.	Sponsoring Org. Report No.	Standard Report No.	INIS Subject Code
KAERI/TR-2552/2003			

Title / Subtitle	Implementation of Nuclear Material Surveillance System Based on the Digital Video Capture Card and Counter
Project Manager and Department	Lee, Sang-Yoon (Fuel Cycle Technology Development)
Researcher and Department	Song, Dae-Yong (Fuel Cycle Technology Development) Ko, Won-Il (Fuel Cycle Technology Development) Ha, Jang-Ho (Fuel Cycle Technology Development) Kim, Ho-Dong (Fuel Cycle Technology Development)

Publication Place	Daejon	Publisher	Korea Atomic Energy Research Institute	Publication Date	2003.7
Page	42p.	Ill. & Tab.	Yes(x), No( )	Size	A4
Notes					

Classified	Open(X), Restricted( )	Report Type	Technical Report
Sponsoring Org.		Contract No.	

Abstract (15-20 Lines)

In this paper, the implementation techniques of nuclear material surveillance system based on the digital video capture board and digital counter was described. The surveillance system that is to be developed is consist of CCD cameras, neutron monitors, and PC for data acquisition. To develop the system, the properties of the PCI based capture board and counter was investigated, and the characteristics of related SDK library was summarized. This report could be used for the developers who want to develop the surveillance system for various experimental environments based on the DVR and sensors using Borland C++ Builder.

Subject Keywords (About 10 words)	Nuclear Surveillance System, Digital Image Capture, Digital Counter, Borland C++ Builder