# PCI Express Digital Capture board

# (EVDC1EX V2.0)

## 개발자 매뉴얼


경기도 안양시 동안구 관양동 1307-37
관양두산벤처다임 605 호
TEL : 031-596-6770
FAX : 031-596-6772
http://www.envylogic.com
기술지원 : support@envylogic.com

# 목 차

# 1.서언

본 EVDCDXV2.0 제품은 PCI Express 4-lane BUS slot 에 설치 되어서 Differential Cable 방식인 LVDS 로 인터페이스 되는 디지털 카메라의 영상 데이터를 호스트의 프레임 버퍼 영역에 효율적으로 전송하기 위하여 개발된 제품입니다.

카메라 라인의 토글횟수를 측정하는 기능을 구현하여 카메라의 작동여부나 케이블 이상 여부를 쉽게 파악할 수 있도록 하였습니다.

또한 별도로 제품인 MIPI 4-lane interface board 와 같이 사용하면 최대 500MB/s 의 MIPI 혹은 병렬 카메라 데이타를 실시간 전송할 수 있으며 전송 케이블의 에러 여부를 파악할 수 있는 기능이 있어서 신뢰성 있는 이미지 데이타 전송이 가능합니다.

보다 직관적이고 편리한 사용자 인터페이스를 위해서 API 함수를 재구성 하였고 보드 작동 중에 상태 파악을 위한 에러 메시지를 보강하였습니다.

사용시에 불편 사항이나 문제 사항이 발생하면 확실한 해결책이 나올 수 있도록 최선의 노력을 다할 것임을 고객 여러분 들에게 약속하며, 고객 여러분의 많은 의견과 성원을 기대합니다.

칩, 보드, 디바이스 드라이버,API 등 본 제품에 들어가는 기술 요소들을 자체 개발하여 확보하였기 때문에 세밀한 기술 지원이 가능합니다. 또한 기능의 추가나 사양의 변경 등을 통하여 고객의 요구에 딱 맞는 제품의 제공이 가능합니다.

이미 여러 고객 회사들은 이러한 차별적인 서비스를 통하여 현장에서 만족스러운 결과를 얻고 있습니다.
당사는 현장에 쓸만한 제품을 개발 공급하기 위해서 노력을 아끼지 않을 것입니다.

# 2.특징

다양한 디지털 2D 카메라 및 라인 스캔 카메라 지원.

LVDS(RS-644) 인터페이스.

MIPI packet capture 기능

최대 250MHz 도트 클럭.

TTL(18bit I/O) DIO 포트 내장.

Lookup table 을 적용하여, 1, 2, 4,8,16bit 데이타 변환 가능.

DMA controller 내장하여 PCI express BUS 상에서 CPU 로드 없이 전송가능.

Camera 신호의 노이즈에 강한 capture controller 구현.

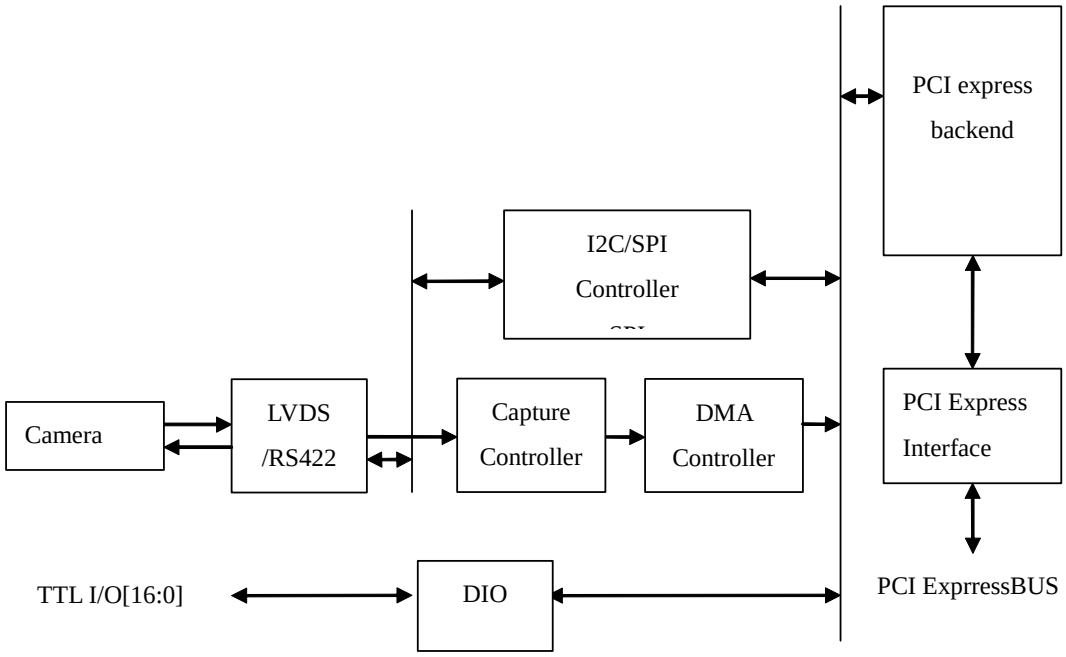
카메라 신호라인 각각에 대한 토글 횟수 측정 기능 구현하여 동작이상을 빠르게 진단 가능.

MIPI packet 을 이용하는 카메라 케이블 에러 검출 기능.

# 3.보드의 구조

## 3.1. 보드의 외형

### 3.2. 블락도




<EVDCEXV1.4 보드의 블락도>

### 3.3. Camera :

LVDS, RS-422 type 의 각종 Line Scan camera 를 사용할 수 있으며 또한 Area camera 도 지원한다. 최근에 Color CMOS/CCD sensor module 이 mobile 기기에 장착 되면서 가격이 싸고 기능이 우수한 제품들이 많이 나오고 있다. 그러한 제품도 간단한 인터페이스 회로를 추가하여 신호를 LVDS 로 변환하면 쉽게 인터페이스 하여 응용이 가능하다.

### 3.4. LVDS :

카메라에 연결하는 라인 드라이버로서 보드의 종류에 따라서 LVDS type 과 RS-422 type 을 사용한다. LVDS 를 사용할 경우는 250MHz 까지 전송이 가능하다. 별도의 MIPI interface board 를 사용하면 250MHz 에서 DDR 로 전송하여 2 배의 전송 속도로 전송 가능하다.

### 3.5. Capture Controller :

카메라의 신호를 주어진 동기 신호에 따라서 전송하는 기능을 한다. 내부적으로 Bit 변환 -> Lookup Table->Data Selector 등의 단계를 거쳐서 동작을 한다.

### 3.6. DMA Controller :

영상 데이터를 CPU 의 관여 없이 호스트의 메모리 영역에 전송하는 기능을 한다. 이 것을 통해서 실질적으로 고속 영상 데이터가 전송 가능하게 된다.

### 3.7. I2C/SPI controller :

하드웨어적으로 I2C master. SPI master 기능을 수행할 수 있도록 한다.

### 3.8. DIO :

TTL 규격의 입출력 포트를 인터페이스 하는 기능을 한다.

### 3.9. PCI express backend :

보드내의 Local BUS 와 PCI express BUS protocal 변환을 하는 기능을 한다.

### PCI Express interface :

물리적인 PCI express BUS 신호와 인터페이스를 하고 패킷 데이타를 주고 받도록 하는 기능을 한다. 여기에서는 PCI Express gen-1 4-lane 을 사용한다.

# 4.I/O connector 사양

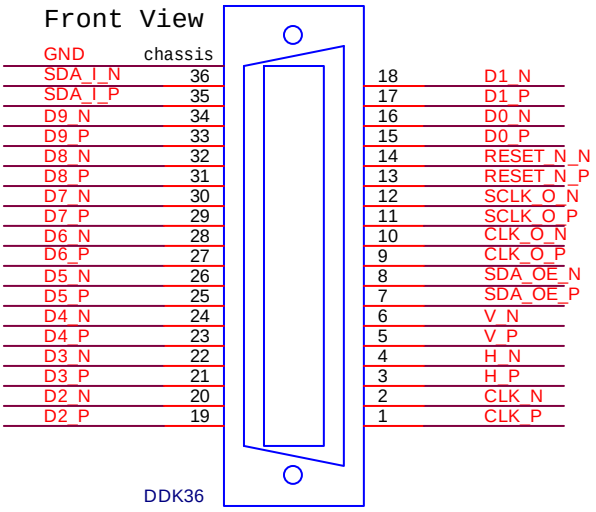
### 4.1. Camera Connector

DDK36 형식으로 브라켓이 장착이 되어 있다.

**주의 할 점 :**

그라운드 신호가 라인으로 나와 있지 않고 **connector** 의 케이스에 연결되어 있다. 따라서 케이블은 반드시 실드가 되어 있는 케이블을 사용하고 또한 실드를 통해서 **GND** 신호가 카메라에 연결이 되어야 한다.

## 4.1.1. Bracket



## 신호의 종류 및 기능

| Pin Name | Pin Number | In/Out (Board side) | I2C_MODE (I2C Cable Mode) |
|---|---|---|---|
| CLK_P | 1 | In | Clock input Normal |
| CLK_N | 2 | In | Clock input Inverse |
| H_P | 3 | In | Horizontal sync input Normal |
| H_N | 4 | In | Horizontal sync input Inverse |
| V_P | 5 | In | Vertical Sync input Normal |
| V_N | 6 | In | Vertical sync input Inverse |
| D0_P | 15 | In | Data bit 0 Normal |
| D0_N | 16 | In | Data bit 0 Inverse |
| D1_P | 17 | In | Data bit 1 Normal |
| D1_N | 18 | In | Data bit 1 Inverse |
| D2_P | 19 | In | Data bit 2 Normal |
| D2_N | 20 | In | Data bit 2 Inverse |
| D3_P | 21 | In | Data bit 3 Normal |
| D3_N | 22 | In | Data bit 3 Inverse |
| D4_P | 23 | In | Data bit 4 Normal |
| D4_N | 24 | In | Data bit 4 Inverse |
| D5_P | 25 | In | Data bit 5 Normal |
| D5_N | 26 | In | Data bit 5 Inverse |
| D6_P | 27 | In | Data bit 6 Normal |
| D6_N | 28 | In | Data bit 6 Inverse |
| D7_P | 29 | In | Data bit 7 Normal |
| D7_N | 30 | In | Data bit 7 Inverse |
| D8_P | 31 | In | Data bit 8 Normal |
| D8_N | 32 | In | Data bit 8 Inverse |
| D9_P | 33 | In | Data bit 9 Normal |
| D9_N | 34 | In | Data bit 9 Inverse |
| CLK_O_P | 9 | Out | Clock output Normal |
| CLK_O_N | 10 | Out | Clock output Inverse |
| SCLK_O_P | 11 | Out | I2C Serial Clock Output Normal |
| SCLK_O_N | 12 | Out | I2C Serial Clock Output Inverse |
| RESET_N_P | 13 | Out | Camera Reset low active Normal |
| RESET_N_N | 14 | Out | Camera Reset low active Inverse |
| SDA_OE_N_P | 7 | Out | I2C data output enable low active normal |
| SDA_OE_N_N | 8 | Out | I2C data output enable low active inverse |
| SDA_I_P | 35 | In | I2C data input Normal |
| SDA_I_N | 36 | In | I2C data input Inverse |
| GND | chassis | | GND |

### 4.1.1.1. 신호 라인

CLK_P/CLK_N :

Clock input, 카메라가 발생하는 클락 신호를 받는 라인.

H_P,H_N :

Horizontal sync input, 카메라가 발생하는 수평 싱크 신호를 받는 라인.

V_P,V_N :

Vertical sync input , 카메라가 발생하는 수직 싱크 신호를 받는 라인. Area 카메라의 경우에 사용된다.

D[9:0]_P,D[9:0]_N :

데이터 입력 라인. 카메라의 데이터를 받는 라인들이다.

CLK_O_P,CLK_O_N :

보드에서 카메라 측에 주는 클락 신호이다.

GND :

보드의 그라운드 신호이다. Connector 의 금속 케이스에 연결이 되어서 GND 가 상호 연결이 되어야 노이즈 발생이 줄어든다. 반드시 연결하도록 하여 Board 와 카메라 간에 GND 를 공유할 수 있도록 해야 한다.

SCLK_O_P,SCLK_O_N:

I2C 의 클락 출력 신호로 동작한다.

RESET_N_P,RESET_N_N:

Camera 를 초기화 할 수 있는 신호로서 Low active 이다.

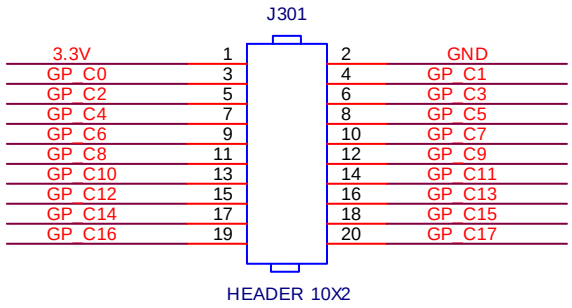
SDA_OE_N_P,SDA_OE_N_N:

I2C 의 데이터를 출력하며 LOW active 로 동작한다.

SDA_I_P,SDA_I_N:

I2C 의 데이터를 받는다.

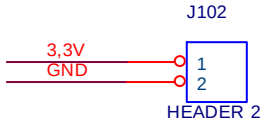
### 4.2. General TTL I/O connector




TTL 신호 규격으로 여러 가지 신호 확장 용으로 사용할 수 있는 connector 이다.
각각의 포트에 OUT,OE 레지스터가 있어서 원하는 대로 입출력을 정하여 사용할 수 있다.
***주의: D2V3 와 다른 점은 1 번핀의 전원이 5V 에서 3.3V 로 변경 되었다.**

GND : GND 단자.
3.3V : 3.3V 전원 출력 단자.
GP_C[17:0] : 각각의 라인은 입출력을 독립적으로 지정하여 사용할 수 있다.

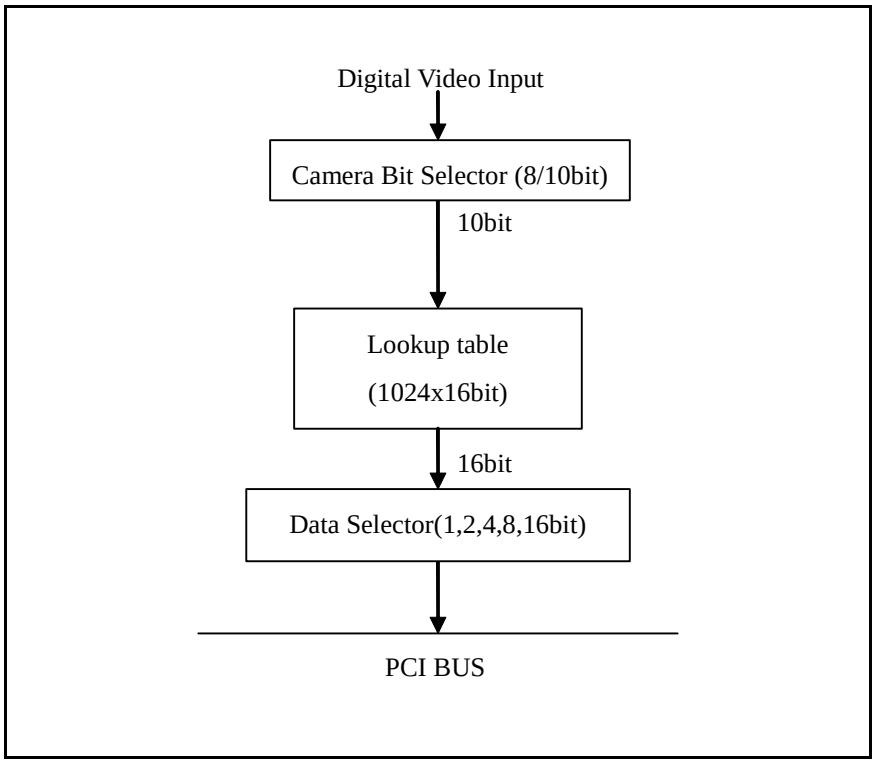
### 4.3. Power Output




최대 전류 500mA 까지 공급이 가능한 파워 단자이다.
***주의 : D2V3 제품이 5V 이나 본 제품은 3.3V 임.**

## 5.데이타 흐름

카메라에서 들오는 데이터는 여러 단계를 거쳐서 가공 되어 전송이 가능하다.
적절하게 이러한 기능을 활용하면 CPU 의 연산 시간 및 데이터 전송 량을 줄이일 수 있다.




<Data 흐름도>

### 5.1. Camera Bit Selector

카메라 입력 라인 중에서 10bit 전체를 사용하는지 아니면 하위 8bit 를 사용하는지를 선택한다. 입력이 8bit 인 경우에도 내부적으로는 10bit 로 처리한다.

### 5.2. Lookup Table

Line Filter 를 통해서 나온 데이터는 1024x16bit 형식의 테이블을 거쳐서 나오게 된다.
원래의 값을 얻기 위해서는 테이블에 어드레스와 동일한 데이터를 저장하면 된다.

사용하는 용도에 따라서 테이블 값을 지정하면 데이터 변환을 하드웨어적으로 처리하는 결과가 되어 CPU 의 처리시간을 아낄 수 있다.

### 5.3. Data Selector

Data selector 의 선택에 따라서 1,2,4,8,16bit 형식으로 데이터를 받을 수 있다.

입력 데이터를 Filter 및 Lookup table 에 의해서 처리하게 되면 실질적으로 필요한 데이터의 정보를 상위 비트로 이동시킬 수 있다. 이 때에 전송하는 데이터의 양을 줄이기 위해서 1,2,4 비트 데이터만을 취해서 전송할 수도 있으며 원하는 데이터의 형태로 변환을 하고자 할 경우에 8,16bit 를 선택하여 사용할 수 있다.

이것도 데이터를 줄이거나 혹은 CPU 의 시간을 줄이는 기능을 할 수 있다.

# 6.응용 프로그램 인터페이스 함수

### 6.1. 개요

API 서비스를 구성하는 DLL 을 초기화 하고 종료하는 함수와 Digital 입출력 지원하기 위한 함수 Capture 를 수행하는 파라미터, 싱크 신호를 발생하는 파라미터 capture 의 실행 및 멈춤 동작을 제어하는 등 관련 함수를 제공하여 쉽고 빠르게 프로그램을 할 수 있도록 하는 함수들이 있다.

또한 Display 를 효율좋게 하기 위한 Overlay 관련 함수들도 제공이 되고 있다.

### 6.2. API Function List

## 6.2.1. 초기화 및 종료

#### 6.2.1.1. DC_Init

시스템에 설치된 디바이스 드라이버들에 연결하고 초기화 한다.

```
int DC_Init();
```

return values

정상이면 0, 아니면 에러코드를 돌려준다.

DC_ERR_NO_DEVICE : 장치가 없음.

DC_ERR_NO_HANDLE : 자원을 할당하고 초기화 실패 함.

#### 6.2.1.2. DC_End

디바이스 드라이버들과의 연결을 끊고 할당한 자원을 시스템에 돌려준다.

```
int DC_End();
```

Return Values

정상이면 0, 아니면 에러 코드를 돌려준다.

Error Code : DC_ERR_NOT_INITIALIZED

#### 6.2.1.3. DC_GetMaxCards

현재 설치된 카드의 갯수를 돌려준다. 카드의 번호는 0 부터 시작한다. 카드의 갯수가 1이면 return 값은 1이 나오고 해당하는 카드의 번호는 0이 된다.

```
int DC_GetMaxCards();
```

Return Values

설치된 카드의 개수.

#### 6.2.1.4. DC_GetVersion

DLL version, Device Driver Version, Board 의 Chip version 을 돌려준다.

prototype

```
int DC_GetVersion (int card_num,DWORD *DllVer, DWORD *DrvVer, DWORD *ChipVer);
```

Paramaters

card_num : 카드번호.

DllVer: DLL version 을 돌려받을 번지. 원하지 않으면 NULL 지정.

DrvVer: Driver version 을 돌려받을 번지. 원하지 않으면 NULL 지정.

ChipVer: Chip Version 을 돌려받을 번지. 원하지 않으면 NULL 지정.

Return Values

error code

### 6.2.2. 기본적인 I/O 함수

#### 6.2.2.1. DC_WriteReg

카드의 레지스터에 데이터를 쓴다.

```
int DC_WriteReg(
int card_num,
DWORD offset,
DWORD data
);
```

Parameters

card_num : 카드번호.

Offset : 번지

Data : 레지스터에 기록할 데이타

Return Values

정상적이면 0 를 돌려주고 아니면 에러코드를 돌려준다.

### 6.2.2.2. DC_ReadReg

카드의 레지스터 값을 읽어온다.

```
int DC_ReadReg (
int card_num,
DWORD offset,
DWORD *data
);
```

Parameters

card_num : 카드번호.

Offset : offset address from the base

data : 레지스터에서 읽어올 데이터 포인터

Return Values

error code

### 6.2.2.3. DC_GpioReadIn

해당 카드의 GPIO 입력 포트의 상태를 읽어온다.

```
int DC_GpioReadIn(int card_num,DWORD *data);
```

Parameters

card_num : 카드번호.

data : [17:0]각 비트는 GPIO 입력 포트의 상태를 돌려받을 변수이다.

각 포트가 High 상태이면 해당하는 비트는 1 이 되고 Low 면 0 이 된다.

return values

error code.

#### 6.2.2.4. DC_GpioWriteOut

해당 카드의 GPIO output register 에 값을 쓴다.

```
int DC_GpioWriteOut(int card_num,DWORD data);
```

Parameters

card_num : 카드번호.

data : [17:0]각 비트는 GPIO Out 레지스터 각각에 해당한다.

return values

error code.

#### 6.2.2.5. DC_GpioWriteOE

해당 카드의 GPIO output enable 레지스터에 값을 쓴다.

```
Int DC_GpioWriteOE (int card_num,DWORD data);
```

Parameters

card_num : 카드번호.

data : [17:0]각 비트는 GPIO 포트의 OE 레지스터에 지정될 값을 의미한다. 각 비트가 '1'이면 GPIO Out 레지스터 값이 출력되고 아니면 Open 상태로 된다.

return values

error code.

### 6.2.3. Capture 동작 제어 함수

DC_Open 함수와 DC_Close 함수를 호출하면 프레임 버퍼의 할당과 제거를 내부적으로 실행한다. 따라서 번번하게 이 함수를 사용하면 메모리 조각이 많이 발생하게 되어서 전송 효율이 떨어질 수 있다.

따라서 프로그램 시작과 종료시에 이 함수를 한번씩 사용하면 메모리의 조각이 적게 발생하고 또한 메모리 할당 에러를 줄일 수 있다.

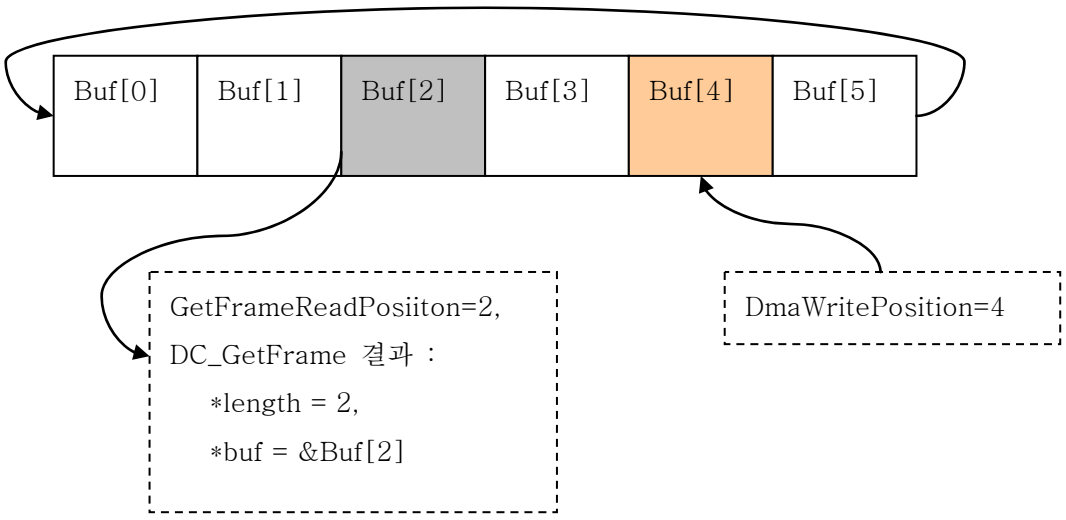
### 6.2.3.1. Capture 시에 프레임 버퍼의 구성과 동작

DC_Open 호출 시에 Frames 파라미터를 6 이라 하면 아래의 그림과 같이 6 개의 Frame buffer 인 Buf[5:0] 프레임 어레이가 생성된다.

DC_Start 를 하면 칩의 하드웨어는 들어오는 영상 데이타를 차례대로 버퍼에 채우고 Buf[5]까지 채우면 다시 Buf[0]로 돌아가서 채우게 된다. 내부적으로 GetFrameReadPosiiton 과 DmaWritePosition 두개의 인덱스가 있다. GetFrameReadPosion 과 DmaWritePosition 은 DC_Start 를 하면 초기화 되어서 0 로 시작한다. 이후 DMA 전송이 되어서 한 프레임의 데이타가 전송이 되면 DmaWritePosition 이 증가를 하게 된다.
DC_GetFrame 함수를 실행해서 전송이 마무리 된 프레임이 있을 경우에 GetFrameReadPosion 은 증가를 하게 된다. 이후 Frames-1 위치가 되면 다시 0 으로 돌아가서 계속 증가를 하게 된다.

DC_GetFrame 함수를 호출하면 전송이 완료된 Frame 이 있을 경우에, 파라미터 length 에는 DMA 전송이 마무리된 프레임의 수를 돌려주고 buf 에는 GetFrameReadposition 에 있는 Frame buffer 의 포인터를 돌려준다. 이후 GetFrame Read Posiiton 이 하나씩 증가를 하게 된다.

capture: Frames=6인 경우




<그림: 프레임 버퍼를 6 으로 지정하여 사용할 경우의 동작 상황>
위의 그림은 6 개의 Frame 이 구성 된 경우에 2 개의 프레임 버퍼가 이미 전송 마무리 된 상황을 보여주고 있다.

### 6.2.3.2. DC_Open

영상 데이터 전송에 사용되는 채널을 여는 기능을 한다. 채널에 관련된 함수의 기능은 채널이 열린 후에 사용 가능하다. 주어진 전송 정보에 맞는 프레임 버퍼를 할당하고 전송 크기와 데이터 형식을 지정한다.

```
BYTE *DC_Open(
int card_num,
int channel,
int Height,
int Width,
int Frames,
int BitsPerClk
);
```

Parameters

card_num : 카드번호.
channel : Capture channel  0 - 1.
Height  : Frame의 수직라인 수.
Width : Frame의 수평 도트 수.
Frames : Capture동작 시에 주 메모리에 할당 되는 최대 프레임 수.

BitsPerClk :
데이터 전송시 클락당 몇 비트를 취하여 전송할 것인가를 정한다. 실제 유효한 값은 1,2,4,8,16 중에 한 개다. 이 값은 결국 lookup table을 거치는 16bit data중에서 상위 몇 비트를 전송할 것인가를 정한다. 이것을 적절히 사용하면 데이터 전송 량을 현격히 줄이거나 데이터 변환을 CPU의 부담 없이 할 수 있다.
각각의 값에 따른 데이터는 취하는 방식은 다음과 같다. LUT_DO[15:0]를 lookup table의 출력 값이라 한다면
1 : LUT_DO [15]
2 : LUT_DO[15:14]
3 : LUT_DO[15:13]
4 : LUT_DO[15:12]
8 : LUT_DO[15:8]
16 : LUT_DO[15:0]
와 같이 해당 되는 비트만을 전송한다. Lookup table의 메모리를 lut_data라고 한다면,테이블의 초기 상태는

```
  for (int i=0;i<1024;i++) {
     lut_data[i] = i<<6;
```

}

와 같이 저장 되어 있다. DC_SetLookupTable 함수를 사용하여 이 값을 사용자가 지정할 수 있다.

return values

Error Codes :

DC_ERR_CHANNEL_INVALID

DC_ERR_OPEN_BITS_PER_CLK_INVALID

DC_ERR_FRAMES_INVALID

DC_ERR_FRAME_SIZE_INVALID

DC_ERR_FRAME_BUFFER_ALLOC_FAILS

### 6.2.3.3. DC_Close

DC_Open 으로 열린 채널을 닫는다. 전송 중이면 그 동작을 멈추고 DC_Open 에 의해 할당된 자원을 시스템에 돌려준다.

```
DC_Close (
int card_num,
Int channel
);
```

Parameters

card_num : 카드번호.

channel : 해당 채널 번호를 말한다. 0 에서 1.

return values

Error Codes :

DC_ERR_CHANNEL_NOT_OPENED

DC_ERR_CHANNEL_INVALID

### 6.2.3.4. DC_Start

이미 열린 채널의 카메라에서 나오는 데이터를 호스트 메모리의 프레임 버퍼로 전송을 시작한다.

```
Int DC_Start(
```

int card_num,

int channel

);

Parameters

card_num : 카드번호.

channel : Capture channel 0 - 1.

return values

DC_ERR_ALREADY_STARTED,

DC_ERR_CHANNEL_NOT_OPENED.

DC_ERR_START_FAILS : 이 경우는 치명적인 에러이므로 더 이상 진행하지 않고 하드웨어 점검을 한다.

### 6.2.3.5. DC_Stop

DC_Start 명령에 의해 진행 되고 있는 전송을 멈춘다.

Int DC_Stop(

int card_num,

int channel

);

Parameters

card_num : 카드번호.

channel : Capture channel 0 - 1.

return values

Error Codes :

DC_ERR_CHANNEL_INVALID,

DC_ERR_NOT_STARTED.

DC_ERR_CHANNEL_NOT_OPENED.

### 6.2.3.6. DC_GetFrame

DC_Start 에 의하여 영상 데이터가 전송되고 있는 중에 가장 먼저 전송이 마무리 된 프레임의 포인터를 buf 에 돌려주고 그 위치에서 버퍼링이 되어 있는 프레임의 수를 length 에 돌려준다. *length 값이 1 이면 한 개 즉 가장 최근에 마무리 된 버퍼임을 알려주고 초과 하면 1

을 뺀 갯수가 버퍼링 되어 있다는 것이다. 1을 초과 하여 버퍼링을 하고 있는 경우에 DC_GetFrame 함수를 실행하면 즉시 다음 프레임의 버퍼 정보를 돌려주고 1인 경우에 blocked 를 1로 하여 실행하면 새로운 프레임이 전송 될 때까지 대기 상태로 있는다.

버퍼에 아직 영상 데이터의 전송이 되지 않은 경우 blocked를 1로 지정해서 이 함수를 호출할 경우 프로세스가 대기 상태로 가기 때문에 별도의 thread나 process 상에서 호출하는 것이 좋다.

```
Int DC_GetFrame(
int card_num,
Int channel,
int blocked,
BYTE **buf,
WORD *length,
DWORD *eof_length)
);
```

Parameters

card_num : 카드번호.

channel : channel number.

blocked :

이 값이 0이면 프레임 데이터의 전송이 끝나지 않은 경우에 대기 하지 않고 즉시 돌아온다. 이 때에 버퍼에 데이터가 없는 경우에는 length에 0을 돌려주고 buf에는 가장 최근에 전송이 된 버퍼의 포인터를 돌려준다.

이 값이 1이면 프레임 데이터가 없을 경우에 대기 상태로 있게 된다. 전송이 마무리 되고 아직 처리되지 않은 프레임이 있는 경우에 리턴하게 된다.

buf : 프레임 데이타가 전송이 끝나는 시점에서 그 프레임의 시작 번지가 저장 될 포인터 변수의 번지이다. 사용하지 않는 경우는 NULL을 지정한다.

length : 함수가 리턴 되는 시점에서 전송이 마무리 된 프레임의 수를 돌려 받는다.

0 : blocked의 값이 0인 경우에 발생할 수 있다. 최근에 전송이 된 프레임 데이터가 없음을 말한다.

1-(Frames-1) : 해당하는 프레임 버퍼의 위치로부터 숫자 만큼의 이미 전송된 프레임이 있다는 것을 의미 한다.

사용하지 않는 경우는 NULL로 지정한다.

eof_length : 현재 프레임의 BYTE 단위의 길이를 돌려 받는다. JPEG 등의 이미지 사이즈가 일

정하지 않는 경우에 NULL 이 아닌 포인터를 주면 실제 전송 받은 데이타 사이를 돌려 받을 수 있다.
사용하지 않는 경우는 NULL 로 지정한다.

return values

DC_ERR_CHANNEL_INVALID
DC_ERR_FRAME_OVER_RUN : PCI 전송이 늦는 경우에 CHIP 상의 하드웨어 버퍼가 넘치는 경우에 발생한다. 돌려받은 프레임의 영상 내용은 깨져 있게 된다. 이후 프레임의 정상적인 전송을 위해서 전송 동작을 재구성한다.
DC_ERR_BUFFER_OVERFLOW : CPU 의 처리가 늦어져서 전송이 된 버퍼를 처리하지 못하는 경우에 발생한다. Frames-1 까지 버퍼를 하고 있다가 이것이 넘치는 경우에 발생한다.

DC_ERR_CHANNEL_NOT_OPENED.
DC_ERR_DMA_TRANSFER_FAILS : 치명적인 에러이므로 더 이상 진행이 되지 않는다. 하드웨어 점검을 해야 한다.

### 6.2.3.7. DC_CancelGetFrame

DC_GetFrame 함수를 blocked 조건으로 부를 경우 프레임 데이터의 전송이 마무리 되지 않으면 프로세스나 thread 가 대기 상태로 있게 된다. 이 때에 이 명령을 내리면 대기 상태를 즉시 빠져나오게 한다.

이 함수는 DC_GetFrame 을 호출한 thread 나 프로세스는 이미 대기상태에 있기 때문에 다른 process 에서 호출 해야 한다.

```
int DC_CancelGetFrame(
int card_num,
int channel
);
```

Parameters

card_num : 카드번호.
channel : channel number

return values

Error Codes :
DC_ERR_CHANNEL_INVALID
DC_ERR_NO_GET_FRAME_PENDING.

## 6.2.4. Capture 파라미터 제어함수들

각 채널 별로 Capture controller가 있어서 독립적인 파라미터를 적용하여 영상 데이터를 받을 수 있다. 이러한 파라미터들을 지정하는 함수들이다.

### 6.2.4.1. DC_SetCapInfo

Capture controller의 동작 방식을 지정하는 함수이다.

```
int DC_SetCapInfo(
int card_num,
int channel,
int HBP,
int VBP,
int ClkEdgeHigh,
int HSyncEdgeHigh,
int VSyncEdgeHigh,
BYTE LineArea,
BYTE Input10Bits,
int SyncValidEn
);
```

Parameters

card_num : 카드번호.

channel : channel number.

HBP : 수평 동기 신호를 검출한 시점의 클락 인덱스를 0으로 하고, 이후에 클락이 진행함에 따라서 증가하게 된다. 이때에 수평의 라인을 시작하는 인덱스 값이다.

VBP : 수직 동기 신호가 검출된 최초의 라인 인덱스를 0이라 하면 이후에 매 수평 동기 신호가 검출 될 때마다 라인 인덱스가 증가 하게 된다. 이 때에 화면의 수평라인 시작 인덱스를 정하는 파라미터이다.

ClkEdgeHigh: Camera에서 나오는 클락의 어떠한 에지에서 픽셀 데이터를 잡을 것인가를 정한다. 1 : 상승 에지 , 0 : 하강 에지.

HSyncEdgeHigh : 수평동기신호에서 어떠한 에지에 기준을 두는가를 정한다.
1 : 상승 에지, 0 : 하강 에지.

VSyncEdgeHigh : 수직동기신호에서 어떠한 에지에 기준을 두는가를 정한다.
1 : 상승 에지, 0 : 하강 에지.

LineArea : 데이터를 잡을 때에 수직 동기를 무시하고 수평 동기만을 사용할 것인지 아니면 수직 동기를 받을 것인지 정하는 것이다.
1 : Line Camera Mode, 수평 동기만 사용. 라인카메라를 사용할 경우에 지정한다.
0 : Area Camera Mode, 수평 수직 동기 모두 사용.

Input10Bits : 카메라와 연결 된 데이터를 취할 때에 10bit 를 모두 취할 것인지 하위 8bit 를 취할 것인지 정하는 함수이다. 내부적으로 영상을 전송하는 데이터 폭은 10bit 이다.

내부 데이터 비트를 cap_data[9:0], camera 에서 오는 데이터를 cam_data[9:0]라 한다면
1 : camera connector 의 데이타 비트 [9:0]을 모두 취한다.
즉 cap_data[9:0] = cam_data[9:0].

0 : camera connector 의 데이타 비트 [7:0]을 취하고 상위 [9:8]을 무시한다. 즉
cap_data[9:2] = cam_data[7:0];
cap_data[1:0] = 0;
SyncValidEn : 수평동기 신호를 동기 신호로 사용하는 것이 아니라 Active 구간에서 전송을 해야 할 Valid 신호로 사용 여부를 결정하는 입력이다. 1 이면 Valid 신호로 사용하고 0 이면 동기 신호로 사용한다.

return values

error code

### 6.2.4.2. DC_SetMipiInfo, DC_SetMipiInfoMulti

MIPI CSI-2 input 을 받을 때에 받고자 하는 데이타 타입과 Virtual Channel 을 지정하는 기능을 한다.
DC_SetMipiInfo 는 한개의 데이타 타입을 지정하는 경우에 사용하고 DC_SetMipiInfoMulti 는 총 4 가지의 타입을 지정할 수 있고 지정을 원하지 않으면 0 을 지정하면 된다.

```
int DC_SetMipiInfo(
int card_num,
int channel,
int VirtualChannel,
int DataType
);

int DC_SetMipiInfoMulti(
int card_num,
```

```
int channel,
int VirtualChannel,
int DataType,int
DataType1,
int DataType2,
int DataType3
);
```

parameter

card_num : 카드번호.

channel : channel number

VirtualChannel : MIPI packet 에서 원하는 가상 채널을 지정한다. 보통 0 을 사용한다.

DataType,DataType1,DataType2,DataType3 :

MIPI packet 에서 받기를 원하는 데이타 타입을 지정한다. 총 4 개 까지 지정할 수 있으며 원하지 않으면 0 으로 지정한다.

### 6.2.4.3. DC_SetLookupTable

capture controller 에서 나오는 데이터는 Lookup table 의 어드레스라인에 전달이 되어서 Table 에 저장된 값으로 대치 되어서 전송이 되게 된다. 테이블은 WORD data[1024] 배열 형식으로 되어 있다.

테이블에 적절한 값을 지정하여 사용하면 들어오는 데이터에 대해서 용도에 맞는 형식으로 변환할 수 있다.

```
int DC_SetLookupTable(
int card_num,
int channel,
int index,
DWORD value
);
```

Parameters

card_num : 카드번호.

channel : channel number

index : 0-1023 까지의 영역을 가지는 테이블의 인덱스 번호

value : 16bit 값이 의미가 있는 값이다. 영상 데이터가 테이블의 번지에 지정이 되어 해당하는 데이터 값이 나오게 되어 있다.

return values

Error Codes :

DC_ERR_LUT_INDEX_RANGE_INVALID

### 6.2.4.4. DC_SetClkOutFreq

클락 출력 신호의 주파수를 정하는 함수이다. 내부에 PLL 이 내장 되어 있어서 원하는 주파수를 생성할 수 있다.

DC_SetClkOutFreq,DC_SetClkOutFreq1 함수가 있다. DC_SetClkOutFreq1 함수는 mode 라는 파라미터가 추가 되어서 PLL 로 생성이 된 주파수에 1,2,1/8 배를 할 수 있는 기능을 제어할 수 있다. DC_setClkOutFreq 함수는 DC_SetClkOutFreq1 함수에서 mode=0 즉 1x 로 지정한 것과 동일한 기능을 한다.

실제 발생 주파수 F 는

F = 40Mhz * clk_o_number / ref_number

의 관계가 있다.

Fin = F/clk_o_number , Ffb = 40Mhz/ref_number 라하고

35kHz < Fin <= 1000kHz

35kHz < Ffb <= 1000kHz

의 조건을 만족한다면 , F 는 10Mhz - 75Mhz 사이의 발생이 가능하다.

이러한 조건을 만족하도록 적절하게 clk_o_number, ref_number 를 지정하면 원하는 주파수를 만들어 낼 수 있다.

```
int DC_SetClkOutFreq(
int card_num,
int clk_o_number,
int ref_number
);

int DC_SetClkOutFreq1(
int card_num,
int clk_o_number,
int ref_number,
int mode
);
```

parameters

card_num : 카드번호.

clk_o_number : 발생된 clock output 에 대해서 카운트 하는 값을 지정한다.

ref_number : 내부에 내장된 40Mhz clock 에 대해서 카운트 하는 값을 지정한다.

mode : 클락의 출력 배수를 정하는 파라미터이다. PLL 을 통해서 발생된 주파수를 x 라고하면 mode 의 값에 따라서 다음의 값이 나온다.

0 : 1x,

1 : 2x,

2 : x/8

이외의 값 : 1x

return values

error code

#### 6.2.4.5. DC_SetClkFreq

Express 4-lane board 에서 지원하는 함수로서 카메라에 전달하는 클락 주파수를 직접 지정할 수 있다.

```
float DC_SetClkFreq(int card_num, float freq);
```

parameters :

card_num : 카드 번호

freq : float 형식으로 지정할 수 있는 주파수 최소 10-250MHz 지정 가능하나 모든 주파수가 나오지 않고 최적으로 근접한 주파수가 생성 된다.

return value

실제 발생 가능한 주파수 값을 돌려준다.

### 6.2.5. 카메라 라인 토글 횟수 검사 기능 제어 함수

보드에는 카메라 각 라인에 대해서 주어진 시간 동안에 토글한 횟수를 측정하는 기능이 있다. 주어진 시간 동안만 토글 횟수를 측정하고 시간이 지나면 토글한 수를 내부 레지스터에 보관하고 있다. 이후 DC_GetCamLineToggleTimes 함수를 통해서 각 라인별 토글 횟수를 알 수 있도록 하고 있다.

#### 6.2.5.1. DC_CheckCamLines

이 함수는 주어진 시간 동안 모든 라인의 토글 횟수를 동시에 측정한다.

기존의 측정 결과는 초기화 되고 이후 주어진 시간이 지나면 자동으로 측정을 끝내고 결과를

레지스터에 저장하게 되어 있다.

```
int DC_CheckCamLines(
int card_num,
int channel,
DWORD width_ms
);
```

parameters

card_num : 카드 번호
channel : 채널 번호. 여기에서는 0.
width_ms : ms 단위로 시간 값이다.

return values

error code

### 6.2.5.2. DC_GetCamLineToggleTimes

측정이 된 카메라의 토글 횟수를 돌려주는 기능을 한다.

```
int DC_GetCamLineToggleTimes(
int card_num,
int channel,
int index,
DWORD *val
);
```

parameters

card_num : 카드 번호.
channel : 채널 번호.
index : 카메라 라인의 인덱스 번호
  [11] : Vertical Input line.
  [10] : Horizontal input line.
  [9:0] : camera Data[9:0] lines

val : 토글 값을 돌려받을 번지 값.

return values :

Error code.

## 6.2.6. I2C command

I2C 마스터 기기로서 동작하도록 하는 기본 기능들을 수행하는 함수들이다. 이 기능들은 소프트웨어적으로 작동하는 것이다.

### 6.2.6.1. DC_I2cInit

I2C 모드를 초기화 하는 기능을 한다.

```
void DC_I2cInit(int card_num,int channel,int DelayCnt);
```

**parameters**

card_num : 카드번호.

channel : channel number.

DelayCnt : I2C 동작 시에 클락 주파수를 변화시킬 수 있도록 한다. 값은 1-10 까지 가능하며 값이 클수록 클락의 주파수가 낮아진다. 시스템에 따라서 실제 주파수가 달라질 수 있으므로 적절히 조절해서 사용하도록 한다. 이 값의 단위는 약 1us 정도이며 시스템에 따라서 달라질 수 있다. 초기 상태에는 2 로 되어 있다.

**return values**

error code

### 6.2.6.2. DC_I2cRst

카메라에 Reset 신호를 발생한다.

```
void DC_I2cRst(
int card_num,
int channel
);
```

**parameters**

card_num : 카드번호.

channel : channel number.

**return values**

error code

### 6.2.6.3. DC_I2cWrite

I2C 형식으로 8bit 의 데이터를 쓰는 동작을 한다.

```
void DC_I2cWrite(
int card_num,
int channel,
BYTE data
);
```

parameters

card_num : 카드번호.

channel : channel number.

data : 8bit data

return values

error code

### 6.2.6.4. DC_I2cRead

I2C 형식으로 8bit data 를 읽는 동작을 한다.

```
int DC_I2cRead (int card_num,int channel,int hilo);
```

parameters

card_num : 카드번호.

channel : channel number.

data : 8bit data

hilo : 8it 데이터를 받은 후에 ack 신호를 상대 슬레이브 디바이스에 주어야 한다. 이 때에 ack 신호의 HIGH,LOW 를 정하는 기능을 한다. 슬레이브 디바이스의 조건에 따라서 이 파라미터를 지정하여 사용할 수 있다.

0 : LOW 신호로 응답한다.

1 : HIGH 신호로 응답한다.

return values

읽어온 데이터.

### 6.2.6.5. DC_I2cStart

I2C Start 신호를 발생한다.

```
void DC_I2cStart(int card_num, int channel);
```

parameters

card_num : 카드번호.

channel : channel number.

data : 8bit data

return values

none

### 6.2.6.6. DC_I2cStop

I2C Stop 신호를 발생한다.

```
void DC_I2cStop(
int card_num,
int channel
);
```

parameters

card_num : 카드번호.

channel : channel number.

return values

none

## 6.2.7.하드웨어 I2C, SPI, 새로운 소프트웨어 I2c 함수들

I2c 와 SPI 악세스를 위해서 새롭게 추가 된 함수들이다.
하드웨어 방식으로 악세스를 할 수 있는 함수가 추가 되어서 보다 효율적으로 전송할 수 있다.

### 6.2.7.1 DC_SetSpiFreq

SPI master 의 클락을 지정한다.

```
float DC_SetSpiFreq(
int card_num,
float freq_mhz
);
```

parameter

card_num : 카드 번호

freq_mhz : float 형식으로 값을 지정할 수 있다. 0.488-62.5 까지 가능.

return value

실제로 세팅이 된 주파수 값을 돌려준다.

### 6.2.7.2. DC_SetSpiLatency

SPI 인터페이스가 케이블을 통해서 전달이 되고 있다.
따라서 고속으로 전송할 경우에 케이블의 지연으로 latency 가 발생할 수 있다. 이 값을 지정한다.

```
int DC_SetSpiLatency(
int card_num,
int lat);
```

parameter

card_num : 카드 번호

lat : 0-7 까지 값을 지정할 수 있고 단위는 SPI clock 의 반클락 단위이다.
SPI 방식으로 읽을 경우에 이 값을 조절하여서 맞는 값이 들어오는지를 확인 후에 read 기능을 사용해야 한다.

return value

에러 코드

### 6.2.7.3. DC_SpiWrite

하드웨어적으로 SPI 쓰기 동작을 하는 함수이다.

```
int DC_SpiWrite(
int card_num,
int length,
BYTE last,
BYTE *buf);
```

parameter

card_num : 카드 번호

length : 길이 정보이다. 1-255 까지 가능하다.

last : 현재의 쓰기 동작이 SPI 쓰기의 마지막 여부를 알려준다. 1 이면 마지막, 0 이면 마지막이 아니라는 것을 의미한다. 마지막의 경우에는 SPI 의 하드웨어 동작에서 마지막 바이트를 쓴 후에 MOCS 신호가 비활성화 된다.

buf : 쓸 데이타를 저장하고 있는 버퍼의 포인터이다.

return value

에러 코드

### 6.2.7.4. DC_SpiRead

하드웨어 적으로 SPI 읽기 동작을 하는 함수이다.

```
intDC_SpiRead(
int card_num,
int length,
BYTE last,
BYTE *buf);
```

parameter

card_num : 카드 번호

length : 길이 정보이다. 1-255 까지 가능하다.

last : 현재의 쓰기 동작이 SPI 쓰기의 마지막 여부를 알려준다. 1 이면 마지막, 0 이면 마지막이 아니라는 것을 의미한다. 마지막의 경우에는 SPI 의 하드웨어 동작에서 마지막 바이트를 읽은 후에 MOCS 신호가 비활성화 된다.

buf : 데이를 읽어올 버퍼의 번지 값이다.

return value

에러코드

### 6.2.7.5. DC_SetHwI2cFreq

I2C master 의 클락을 지정한다.

```
float DC_SetHwI2cFreq(
int card_num,
float freq_khz
);
```

parameter

card_num : 카드 번호

freq_mhz : float 형식으로 값을 지정할 수 있다.37-4800 까지 가능하다.

return value

실제로 세팅이 된 주파수 값을 돌려준다.

### 6.2.7.6. DC_HwI2cWrite

하드웨어 I2C 쓰기 기능을 한다.

```
int DC_HwI2cWrite(
int card_num,
int length,
BYTE start,
BYTE last,
BYTE *buf);
```

parameter

card_num : 카드 번호

length : 길이 정보이다. 1-255 까지 가능하다.

start : 쓰기 동작이 I2C 의 시작을 포함 여부를 알려준다. 1 이면 시작이 들어가고 0 이면 아니다.

last : 현재의 쓰기 동작이 I2C 쓰기의 마지막 여부를 알려준다. 1 이면 마지막, 0 이면 마지막이 아니라는 것을 의미한다. 마지막의 경우에는 I2C Stop 신호를 발생하고 전송을 마친다.

buf : 쓸 데이타를 저장하고 있는 버퍼의 포인터이다.

return value

에러코드

### 6.2.7.7. DC_HwI2cRead

하드웨어 적으로 I2C 읽기 동작을 하는 함수이다.

```
int DC_HwI2cRead(
```

```
int card_num,
int length,
BYTE last,
BYTE *buf);
```

parameter

card_num : 카드 번호

length : 길이 정보이다. 1-255 까지 가능하다.

last : 현재의 동작이 I2C 읽기의 마지막 여부를 알려준다. 1 이면 마지막, 0 이면 마지막이 아니라는 것을 의미한다. 마지막의 경우에는 I2C Stop 신호를 발생하고 전송을 마친다.

buf : 데이를 읽어올 버퍼의 번지 값이다.

return value

에러코드

### 6.2.7.8. DC_ DC_I2cClkStretchEn

소프트웨어적으로 I2C 동작을 수행할 때에 슬레이브 디바이스그 클락을 활성화하여서 지연을 시킬 때에 그것을 가능하게 하는 기능을 한다.

```
int DC_I2cClkStretchEn(
int card_num,
int channel,
int en);
```

parameter

card_num : 카드 번호

channel : 이 값을 0 으로 지정한다.

en : 1: enable, 0 : disable

return value

### 6.2.7.9. DC_SoftI2cWrite

소프트웨어 I2C 쓰기 기능을 한다.

```
int DC_SoftI2cWrite(
int card_num,
int length,
BYTE start,
BYTE last,
BYTE *buf);
```

parameter

card_num : 카드 번호

length : 길이 정보이다. 1-255 까지 가능하다.

start : 쓰기 동작이 I2C 의 시작을 포함 여부를 알려준다. 1 이면 시작이 들어가고 0 이면 아니다.

last : 현재의 쓰기 동작이 I2C 쓰기의 마지막 여부를 알려준다. 1 이면 마지막, 0 이면 마지막이 아니라는 것을 의미한다. 마지막의 경우에는 I2C Stop 신호를 발생하고 전송을 마친다.

buf : 쓸 데이타를 저장하고 있는 버퍼의 포인터이다.

return value

에러코드

### 6.2.7.10. DC_SoftI2cRead

소프트웨어 적으로 I2C 읽기 동작을 하는 함수이다.

```
int DC_SoftI2cRead(
int card_num,
int length,
BYTE last,
BYTE *buf);
```

parameter

card_num : 카드 번호

length : 길이 정보이다. 1-255 까지 가능하다.

last : 현재의 동작이 I2C 읽기의 마지막 여부를 알려준다. 1 이면 마지막, 0 이면 마지막이 아니라는 것을 의미한다. 마지막의 경우에는 I2C Stop 신호를 발생하고 전송을 마친다.

buf : 데이를 읽어올 버퍼의 번지 값이다.

return value

에러코드

### 6.2.7.11. DC_SetSpiI2cMode

하드웨어 방식으로 SPI 나 I2C 를 사용할 것인지 아니면 Software 방식으로 I2C 를 사용할 지 사용 모드를 지정하는 기능을 한다.

```
int DC_SetSpiI2cMode(
```

```
int card_num,
int si);
```

parameter

card_num : 카드 번호

si : 1 이면 SPI or I2c hardware mode , 0 이면 소프트웨어 I2C mode 로 지정한다.

하드웨어 모드인 경우에 이름이 DC_Spi, DC_HwI2c 로 시작하는 기능을 사용할 수 있고 소프트웨어 모드인 경우는 DC_SoftI2c 로 시작하는 기능과 기존 DC_I2c 로 시작하는 기능을 사용할 수 있다.

return value

에러코드

### 6.2.7.12. DC_GetSpiI2cMode

현재의 I2C 동작 모드를 돌려준다.

```
int DC_GetSpiI2cMode(int card_num);
```

parameter

card_num : 카드 번호

return value

I2c 동작 모드를 알려준다.

1 이면 하드웨 모드로 동작 중이라는 것을 의미하고 0 이면 소프트모드이다.

### 6.2.8. 오버레이 함수

이 함수들은 DISPLAY 속도의 향상을 위해서 제공되는 함수이다. 이 함수를 사용하면 GDI 를 사용하는 것에 비해 효율좋게 화면에 보여줄 수 있다. 계산 작업이 많이 필요할 때에 사용하면 전체적인 작업 성능을 향상시킬 수 있다.

**시스템에 direct-X 7.0 이상이 설치 되어야 한다.**

#### 6.2.8.1. DC_DispOpen

Display 화면을 여는 기능을 한다.
오버레이에 관련된 자원을 할당하고 DirectDraw 를 사용하여 Surface 를 생성한다.

```
int    DC_DispOpen(
HWND hWnd,
int width,
int height,
int pixelBytes
);
```

parameters

hWnd : 오버레이 창 핸들.
width : 창의 넓이.
height : 창의 높이.
pixelBytes : 픽셀당 바이트수. 현재는 YUV 포맷을 사용하므로 2 를 지정하여 사용한다.

#### 6.2.8.2. DC_DispClose

오버레이 자원을 돌려준다.

```
int    DC_DispClose();
```

DC_DispOpen 함수로 할당한 자원을 시스템에 돌려준다.

#### 6.2.8.3. DC_SetKeyColor

오버레이를 위한 키칼러를 지정하는 함수이다.
기본값은 RGB(0xff,0xff,0)이다.

prototype

void DC_SetKeyColor(DWORD keycolor);

parameters

keycolor : RGB 값을 지정하면 배경화면에 일치하는 영역에 overlay 화면 데이터가 보인다.

return value

에러코드.

### 6.2.8.4. DC_GetKeyColor

현재 지정된 키 칼러 값을 돌려준다.

DWORD DC_GetKeyColor();

parameters

return values

### 6.2.8.5. DC_SetFourCC

Overlay 에 사용되는 화면의 pixel format 을 지정한다.
기본설정 값은 UYVY 이다.

void DC_SetFourCC(DWORD dwFourCC);

parameters

dwFourCC : Color space 포맷을 지정하는 기능을 한다.
대부분의 Graphic adaptor 는 UYUV,YUV2 포맷을 지원하며 실질적으로 많이 사용하는 방식이다.

return values

### 6.2.8.6. DC_GetFourCC

```
DWORD DC_GetFourCC();
```

parameters

return value

FourCC 값.

### 6.2.8.7. DC_SetShowRect

보이는 창을 지정한다.

```
void DC_SetShowRect(RECT *sr,RECT *dr);
```

parameters

sr :
source 윈도우의 창의 크기를 정한다.

dr :
대상이 되는 창의 크기를 의미하는 포인터이다.

return value

### 6.2.8.8. DC_Update

오버레이 창의 소스 데이터를 업데이트 한다.

```
int DC_Update(BYTE *buf,int width,int height,int pixelbytes,int flip);
```

parameters

buf
업데이트 할 버퍼의 소스를 지시하는 파라미터이다.

width
창의 넓이.

height
창의 높이.

pixelbytes
픽셀당 바이트 수

flip
FLIP 기능을 사용할 것인가 아닌가를 지정한다. Update rate 와 Display rate 가 서로 동기 되지 않으면 화면의 소스가 변화할 때에 화면이 찢어지는(Tearing) 현상이 발생하게 된다. Flip 을 사용하면 이것을 동기화 시키는 기능을 하여 찢어지는 현상이 발생하지 않게 된다. 그러나 레이트가 떨어질 수 있다.

0 : 플립기능을 사용하지 않는다. buf 의 데이터를 direct window 창에 직접 업데이트 한다.
1 : 플립기능을 사용한다. 이것을 사용하면 최대 레이트가 Graphic card DISPLAY rate 의 1/2 이 된다.

return value

## 6.3. 에러 코드

에러코드를 리턴 하는 함수 중에서 0 이 아닌 값을 돌려주게 되면 어떠한 에러가 발생했음을 말한다. 이 경우에 값에 따라서 다음의 의미를 가지게 된다.

CAP_BASE_DEVICE_OPEN_ERROR :
디바이스 드라이버를 열 수 없을 때에 일어나는 에러이다.

CAP_BASE_NO_HANDLE :
디바이스 핸들을 얻지 못했을 때에 발생한다.

CAP_CARD_CHANNEL_OVERFLOW :
채널이 범위를 벗어 났을 때에 발생한다.

DC_ERR_ALREADY_INITIALIZED
이미 초기화 됨.
발생 함수 : DC_Init.

DC_ERR_NOT_INITIALIZED :
Library 가 초기화 되지 않았음.DC_Init 를 하지 않고 다른 함수들을 호출할 때에 나온다.
발생 함수 : DC_Init 제외한 함수.

DC_ERR_NO_HANDLE:
응용 프로그램이 API 프로그램으로부터 handle 을 얻지 못함. 시스템을 재 시작해야 한다.

DC_ERR_NO_DEVICE :
장치가 설치되어 있지 않음.
발생 함수 : DC_Init

DC_ERR_CHANNEL_INVALID :
채널 번호 벗어남.
발생 함수 : DC_Open,DC_Close,DC_GetFrame,DC_GetFrameAtPos,DC_CancelGetFrame,…등
channel 을 파라미터로 가지는 함수들.

DC_ERR_ALREADY_OPENED :
이미 DC_Open 을 실행했음을 말한다.
발생 함수 : DC_Opem

DC_ERR_OPEN_BITS_PER_CLK_INVALID :
1,2,4,8,16 중에 해당하지 않음.
발생 함수 : DC_Open

DC_ERR_FRAMES_INVALID :
Open 파라미터의 최대 프레임 개수가 적절하지 않음. 최대 256 까지.
발생 함수 : DC_Open

DC_ERR_FRAME_SIZE_INVALID :
DC_Open 파라미터 중에서 Width*Height*BitsPerClk/8 값이 4 의 배수가 아니거나 값이 8Gbyte 를 초과함.
발생 함수 : DC_Open

DC_ERR_FRAME_BUFFER_ALLOC_FAILS :
프레임 버퍼 할당 실패. 주 메모리에 여유가 없을 경우에 발생한다. Frames 를 줄이거나 화면의 사이즈를 줄인다.
발생 함수 : DC_Open

DC_ERR_START_FAILS :
채널의 DMA 동작 시작에 에러 발생. 더 이상 진행이 어렵고 하드웨어 점검을 해야 한다.
발생 함수 : DC_Start

DC_ERR_ALREADY_PENDING :
이미 주어진 채널에서 DC_GetFrame 을 진행중이라는 것을 의미한다.
발생 함수 : DC_GetFrame

DC_ERR_NOT_PENDING:
DC_GetFrame 을 실행하여 대기 상태에 있는 상황이 아니다.
발생 함수 : DC_CancelGetFrame

DC_ERR_CARD_NUMBER_INVALID :
해당 카드가 없음.
발생 함수 : card_num 을 파라미터로 가지고 있는 함수의 호출 시에 발생.

DC_ERR_CHANNEL_NOT_OPENED :
채널이 열리지 않았음.
발생 함수 : DC_Start

DC_ERR_ALREADY_STARTED :

이미 채널의 데이터가 전송 중.
발생 함수 : DC_Start

DC_ERR_NOT_STARTED :
채널의 캡쳐 동작이 시작되지 않았음.
발생 함수 : DC_Stop

DC_ERR_BUFFER_OVERFLOW :
DC_GetFrame 동작이 늦어지거나 실행되지 않는 상황에서 Capture 가 진행할 때에 Capture 를 한 프레임 수가 넘치는 경우가 발생한다. 이 경우에 가장 오래된 프레임 영역에 덮어서 전송을 하게 되어 프레임을 잃어버린다. 이 경우 내부적으로는 GetFrameReadPosition 을 하나 증가시킨다. (6.2.3.1 항 참조)
발생 함수 : DC_GetFrame

DC_ERR_FRAME_OVER_RUN :
데이터의 전송 중에 PCI BUS 를 통하여 데이터 전송이 늦어져서 칩 내부의 하드웨어 버퍼가 넘치는 경우가 발생했음을 말한다. 현재의 프레임 버퍼의 내용은 사용할 수 없으므로 즉시 DMA 동작을 멈추고 새로운 프레임을 받을 준비를 하도록 한다. 보통 10us 이내에 재 싱크 동작을 수행하여 이후의 프레임을 정상적으로 잡을 수 있도록 한다. 에러가 발생한 프레임은 과거의 영상과 겹쳐 있어서 사용이 어렵다.
발생 함수 : DC_GetFrame

DC_ERR_DMA_TRANSFER_FAILS :
내부 DMA 동작에 문제가 있다. 이러한 에러 발생하면 더 이상 동작이 어렵다. 하드웨어의 설치 상황을 재점검해야 한다. 주로 PCI Slot 의 접촉 불량이 문제가 될 수 있다.
발생 함수 : DC_GetFrame

DC_ERR_LUT_INDEX_RANGE_INVALID :
Lookup table 의 Index 범위를 벗어남.
발생 함수 : DC_SetLookupTable

- 끝 -