Simply Brighter

(In Canada)
111 Railside Road
Suite 201
Toronto, ON M3A 1B2
CANADA
Tel: 1-416-840 4991

Fax: 1-416-840 6541

(In US)
1241 Quarry Lane
Suite 105
Pleasanton, CA 94566
USA
Tel: 1-925-218 1885

Email:
sales@mightex.com

Mightex CCD Line Camera USB Protocol

Version 1.2.1

Oct. 18, 2018

Relevant Products

Part Numbers
TCN-1304-U, TCE-1304-U, TCE-1304-UW TCN-1209-U,TCE-1209-U, TCE-133A-U, TCN-1024-U, TCE-1024-U,TCN-1024-UF, TCE-1024-UF

Revision History

Revision	Date	Author	Description
1.0.0	Mar. 26, 2007	JT Zheng	Initial Revision
1.1.0	Jan. 18, 2008	JT Zheng	Add TCN-1209-U Modal
1.1.1	Oct. 16, 2009	JT Zheng	Add TCE-133A-U Modal
1.2.0	Jan. 12, 2011	JT Zheng	Add TCX-1024-U/UF Modal
1.2.1	Oct.18,2018	JT Zheng	New Logo

Mightex USB 2.0 CCD Line camera is designed for low cost spectrometer and machine vision applications, With USB 2.0 high speed interface and powerful PC camera engine, the camera delivers CCD Line image data at high frame rate. The GUI demostration application and SDK are provided for user's application developments, For users who want to embedded it to Non-Windows based system, the USB2.0 based command protocol is described.

Mightex USB 2.0 Line camera is using Cypress CY68013A as its on board USB device controller, please refer to the Cypress documents for the details of the chip's USB protocol supporting.

Descriptors

Device Descriptor:

bcdUSB:	0x0200
bDeviceClass:	0x00
bDeviceSubClass:	0x00
bDeviceProtocol:	0x00
bMaxPacketSize0:	0x40 (64)
idVendor:	0x04B4 (For evaluation samples)
idProduct:	0x0328
bcdDevice:	0x0000
iManufacturer:	0x01
	0x0409: "Mightex"
iProduct:	0x02
	0x0409: "USB-TCD1304-1"
iSerialNumber:	0x00
bNumConfigurations:	0x01

Configuration Descriptor:

wTotalLength:	0x0027
bNumInterfaces:	0x01
bConfigurationValue:	0x01
iConfiguration:	0x00
bmAttributes:	0x80 (Bus Powered)
MaxPower:	0x96 (300 MA)

Interface Descriptor:

bInterfaceNumber:	0x00
bAlternateSetting:	0x00
bNumEndpoints:	0x03
bInterfaceClass:	0xFF
bInterfaceSubClass:	0x00
bInterfaceProtocol:	0x00
iInterface:	0x00

Endpoint Descriptor:

bEndpointAddress:	0x01 **OUT**
Transfer Type:	Bulk
wMaxPacketSize:	0x0200 (512)
bInterval:	0x00
bEndpointAddress:	0x81 **IN**
Transfer Type:	Bulk
wMaxPacketSize:	0x0200 (512)
bInterval:	0x00
bEndpointAddress:	0x82 **IN**
Transfer Type:	Bulk
wMaxPacketSize:	0x0200 (512)
bInterval:	0x00

IMPORTANT:

1). End point 0 is NOT listed above, as it's used for standard USB enumeration and configuration, for details of EP0 protocols, please refer to the latest USB specification and Cypress CY68013A manual.

2). End point 1 (for IN and OUT) and 2 (for IN only) are used for camera specific commands, this documents is focused on the data communications on these End points.

The protocol is extremely simplified for quick understanding and development.

End points 1 (IN & OUT)

End Point 1(OUT) and 0x81(IN) are used for command control, it's always the Host initiates a command to Camera on EP1(OUT), and according to the command, Camera may prepare Response in its buffer and waiting for Host to fetch. The command and response has the following generic format:

EP(0x01)	→	CommandID	Length	Data
EP(0x81)	←	Result	Length	Data

For **Command**:
CommandID : This is a ONE byte field, represent the command itself, it tells device what to do.
Length: The length of data in byte.
Data: Depends on the command ID.

For **Response**:
Result: 0x01 – Means OK, 0x00 – means Error
Length: The length of response data in byte.
Data: Depends on the command ID.

****. Query Firmware Version (CommandID = 0x01)***

→ 0x01	1	0x02	
← 0x01	3	Major	Minor Revision

Host can use this command to query version of the firmware from device, device will response the current firmware version.

****. Query Device Information (CommandID = 0x21)***

→0x21	1	0x00
←0x01	sizeof(tDeviceInfo)	DeviceInfo

For the command, the data field (0x00) is not used actually. Upon receiving this command, device will prepare device information and put in the EP1(IN) buffer, for host to fetch. The tDeviceInfo has the following definition:

```
#define STRING_LENGTH   14
typedef struct
{
    BYTE    ConfigRevision;
    BYTE    ModuleNo[STRING_LENGTH];
    BYTE    SerialNo[STRING_LENGTH];
    BYTE    ManuafactureDate[STRING_LENGTH];
} tDeviceInfo;
```

it's actually a 43 bytes data structure which contains the ModuleNo[] and SerialNo[], these two fields are important if user wants to support multiple devices in the upper software, as the serialNo[] might be the only information for user to distinct the devices.

. Camera Work Mode Setting (commandID = 0x30)

→ 0x30	1	MODE (0 or 1) #define NORMAL_MODE 0x00 #define TRIGGER_MODE 0x01

This command is used for setting the current mode of the device, there's no response for this command, for the details of NORMAL and TRIGGER mode, please refer to Line Camera's user manual and SDK manual.
Note that this command has an important side-effect that it will empty the on-camera frame buffer, it will let camera firmware to neglect the grabbed frames in frame buffer. This is very useful (especially in NORMAL mode), while user wants to get a new frame. (the frames in buffer might be "old" frames).

. Set Camera Exposure Time (commandID = 0x31)

→ 0x31	2	MSB LSB (of Exposure Time)

Host can set Exposure Time of the device with this command, there's no response for it. The two bytes Data are the MSB and LSB of the new exposure time (16 bit word), note that the unit of the exposure time is 0.1ms for TCN-1304-U and TCN-1209-U and 0.01ms for TCN-133A-U/TCX-1024-U, for example, for TCN-1209-U, if setting exposure time to 10ms, the value will be 100 (100x0.1ms = 10ms), so the command is:

→0x31	2	0x00 0x64

Note:
1).For TCN1209, the minimum exposure time is 300us, so for any setting less than 3, camera will use 3.
2). For TCN133A, the minimum exposure time is 60us, so for any setting less than 6, camera will use 6.
3). For TCX-1024-U, the minimum exposure time is 40us, for any setting less than 4, camera will use 4.

. Get current buffered image frames (commandID = 0x33)

→0x33	1	0x00
←0x01	1	Frame Count （For cameras except TCX-1024-U）
OR ←0x01	2	FrameCount (MSB, LSB) (For TCX-1024-U)

Device has on board frame buffer for buffering image frames in both NORMAL and TRIGGER modes, currently, the buffer can hold up to 4 frames. This gives host the flexibility to do other operations, not worry the loss of the continuous frames (If the 4 frame buffer are FULL on camera and host still doesn't fetch them, the camera will stop grab frames from CCD sensor).

It's always recommended for host to query the buffered frames before fetching image data, as this is important information for host to arrange the proper buffer for the image data.

Note that for TCX-1024-U camera, the returned frame count is a 2 bytes value.

. Get buffered image data (commandID = 0x34)

	→0x34	1	Frame Count (For cameras except TCX-1024-U)
OR	→0x34	2	Frame Count (MSB, LSB) (For TCX-1024-U)

This is a very important command, Host send this command on EP1(OUT), and there's no response on EP1(IN), However, the frame data will be ready on EP2(IN) for host to fetch. So Host should prepare proper buffer and fetch from EP2(IN) after this command is sent successfully. Another very important point is that: The "Frame Count" here is the number of frames host wants to fetch this time, usually, user should use command (0x33) to get the count of current buffered frames, and use the returned frame count for this command(0x34), or if user have limited memory for image data, user can set the frame count less than the number returned from command(0x33), but NEVER bigger than that number.
Note: For TCX-1024-U, the Frame Count is a 2-byte value.

. Set Camera Bit Mode (commandID = 0x38, For TCE-133A-U and TCX-1024-U only)

→ 0x38 1 BitMode (of Exposure Time)

Host can set camera's bit mode with this command, there's no response for it. The "BitMode" byte is either 8 or 16 which means 8bit or 16bit mode. At 16bit mode, for TCE133A/TCX-1024-U, each pixel value is a 16bit word, but only the 12 LSB are effective (so the value is from 0x0000 to 0x0FFF), thus there're 1024 words for a single frame (TCE133A/TCX1024 has 1024 pixels). At 8 bit mode, a single frame (1024 pixels) contains 512 16bit words only, each word contain two 8bit pixel value. Please refer to the EP2(IN) frame data format for the details of frame data at different bit mode.

. Set Camera Gains (commandID = 0x39, For TCE-133A-U and TCX-1024-U Only)

→ 0x39 3 RedGain GreenGain BlueGain

Host can set Gains of a frame with this command, there's no response for it. The three bytes Data are Gains for Red, Green and Blue pixels, however, as TCE-133A-U is Mono sensor, so these three Gains should be with the same value, note that for TCE-133A-U, there're only 4 Gain Levels, from Level 1 to 4, so, the RedGain/GreenGain/BlueGain value should be from 1 to 4.
For TCX-1024-U, it's a value of 6 – 42 which means 6 -42 dB gains.

. Set Frame Time (commandID = 0x3A, For TCX-1024-U Only)

→ 0x3A 2 MSB LSB (of Frame Time)

Host can set Frame Time of the device with this command, there's no response for it. The two bytes Data are the MSB and LSB of the new frame time (16 bit word), note that the unit of the exposure time is 0.01ms for TCX-1024-U, for example, if setting frame time to 1ms, the value will be 100 (100x0.01ms = 1ms), so the command is:

→0x3A 2 0x00 0x64

Note that Frame Time is only applied to camera when it's in NORMAL mode, while it's in TRIGGER mode, the frame interval is completely controlled by an external trigger signal.
The allowed minimum frame time for 8bit mode is 40us, for 16bit mode, is 100us, that implies that the maximum frame rate in 8bit mode is 25,000fps, in 16bit mode is 10,000 fps.

. Set Soft Trigger (commandID = 0x3B, For TCX-1024-U Only)

→ 0x3B 1 1

Host can simulate a hardware trigger when the camera is set in TRIGGER mode.

. Set Trigger Burst Count (commandID = 0x3C, For TCX-1024-U Only)

→ 0x3C 2 MSB LSB (of Burst Count)

When the camera is set in TRIGGER mode, basically, the camera will grab one frame upon an external trigger assertion, however, user can set the Burst Count to be more than "1" (the default is 1), the camera will grab defined "BurstCount" frames upon each trigger assertion, the frame interval can be defined by the Frame Time (command 0x3A).

. GPIO Chip Register Write (commandID = 0x40)

→ 0x40 2 Register Value

. GPIO Chip Register Read (commandID = 0x41)

→0x41 1 Register

←0x01 1 Value

The device has a on board Philips PCA9536 chip which provide 4 GPIO pins, for user wants to use it, user can use the above two commands for writing and reading register values. For details of the register and their definitions, please refer to the specification of PCB9536.

End points 2 (IN)

End point2 is used for fetching image data only, before reading data from it, please make sure Command 0x34 is sent successfully (please refer to the command 0x34 description, and before command 0x34, command 0x33 is recommended to be issued).
After command 0x34 is sent and proper buffer is arranged, host can read data from EP2(IN), device will return the data in the following format:

****. TCN-1304-U Modal:***

Typedef struct
{
tUINT16 Dummy1[16];
tUINT16 **LightShield[13];**
tUINT16 Reserved[3];
tUINT16 **ImageData[3648];**
tUINT16 Dummy2[14];
tUINT16 Padding[138];
tUINT16 **TimeStamp;**
tUINT16 **ExposureTime;**
tUINT16 **TriggerOccurred;**
tUINT16 **TriggerEventCount;**
tUINT16 Padding2[4];
} tCCDFrames;

For each frame, only elements in **BOLD** are meaningful, others are all dummy or paddings.

Note:
1). The above structure is only for one frame, it's 3840 16bit words and 7680 bytes. For multiple frame fetching (the frame count in command 0x34 is more the ONE), the totally data fetching should be:
7680 bytes x FrameCnt
There's no gap between frames.
2). The CCD's raw data is in ImageData[3648] field, it's 16bit ADC values from the camera, user should pay attention that there's no data is more than 0xC000, otherwise the CCD sensor is "Over Exposured", user should reduce the exposure time to make it into normal range. Note that firmware won't do any automatic exposure time setting…etc., it's host's responsibility to set proper ET.
3). For the last four fields in **bold**, please refer to the SDK manual for the description of them, briefly, these are real time information the CCD sensor was used to generate THIS frame, which is extremely useful in applications.

4). The **LightShield[13]** are pixel value of Optical Black cells. Usually, user should get average value from these 13 pixel values and minus it from all effective pixel data.
5). The data stream from EP2 is in Byte, the above "tUINT16" is in little endian format.

. TCN-1209-U Modal:

Typedef struct
{
tUINT16 Dummy1[13];
tUINT16 **LightShield[16];**
tUINT16 Reserved[3];
tUINT16 **ImageData[2048];**
tUINT16 Dummy2[8];
tUINT16 Padding[200];
tUINT16 **TimeStamp;**
tUINT16 **ExposureTime;**
tUINT16 **TriggerOccurred;**
tUINT16 **TriggerEventCount;**
tUINT16 Padding2[12];
} tCCDFrames;

For each frame, only elements in **BOLD** are meaningful, others are all dummy or paddings.

Note:
1). The above structure is only for one frame, it's 2304 16bit words and 4608 bytes. For multiple frame fetching (the frame count in command 0x34 is more the ONE), the totally data fetching should be:
4608 bytes x FrameCnt
There's no gap between frames.
2). The CCD's raw data is in ImageData[2048] field, it's 12bit ADC values from the camera, user should pay attention that there's no data should be more than 0x0F00, otherwise the CCD sensor is "Over Exposured", user should reduce the exposure time to make it into normal range. Note that firmware won't do any automatic exposure time setting…etc., it's host's responsibility to set proper ET.
3). For the last four fields in **bold**, please refer to the SDK manual for the description of them, briefly, these are real time information the CCD sensor was used to generate THIS frame, which is extremely useful in applications.
4). The **LightShield[16]** are pixel value of Optical Black cells. Usually, user should get average value from these 16 pixel values and minus it from all effective pixel data.
5). The data stream from EP2 is in Byte, the above "tUINT16" is in little endian format.

. TCN-133A-U Modal:

1). For 16bit mode:

Typedef struct
{
tUINT16 **LightShield1[4];**
tUINT16 IsolatedCells1[4];
tUINT16 **ImageData[1024];**

tUINT16 IsolatedCells2[4];
tUINT16 **LightShield2[4];**
tUINT16 Padding[224];
tUINT16 **TimeStamp;**
tUINT16 **ExposureTime;**
tUINT16 **TriggerOccurred;**
tUINT16 **TriggerEventCount;**
tUINT16 **GlobalGain;** *// User set it by the GreenGain of command 0x39.*
tUINT16 Padding2[11];
} tCCDFrames_16Bit;

For each frame, only elements in **BOLD** are meaningful, others are all dummy or paddings.

Note:
1). The above structure is only for one frame, it's 1280 16bit words and 2560 bytes. For multiple frame fetching (the frame count in command 0x34 is more the ONE), the totally data fetching should be:
2560 bytes x FrameCnt
There's no gap between frames.
2). The CCD's raw data is in ImageData[1024] field, it's 12bit ADC values from the camera, user should pay attention that there's no data should be more than 0x0F80, otherwise the CCD sensor is "Over Exposured", user should reduce the exposure time to make it into normal range. Note that firmware won't do any automatic exposure time setting…etc., it's host's responsibility to set proper ET.
3). For the last five fields in **bold**, please refer to the SDK manual for the description of them, briefly, these are real time information the CCD sensor was used to generate THIS frame, which is extremely useful in applications.
4). As the CCD133A has two output channels, we have the following data format for channelA and channelB:
ChannelA: LightShield1[0], [2], IsolatedCells1[0], [2], ImageData[0], [2]…[1022], IsolatedCells2[0], [2], LightShield2[0],[2]
ChannelB: LightShield[1], [3], IsolatedCells1[1], [3], ImageData[1], [3]…[1023], IsolatedCells2[1], [3], LightShield2[1],[3]
We expect user do the following pre-processing before ImageData[] are used.
*). Optional: For ***each channel***, User should generate Average LightShield (average of the 4 LightShield pixels), and subtract this value from all ImageData pixels of this channel.
*). For each channel, If the difference between the average of the two LightShields before the Image data and the average of the two LightShields after the Image data is bigger than 0x100 (256), User should think this frame is "Over Exposured".
5). The data stream from EP2 is in Byte, the above "tUINT16" is in little endian format, e.g. the byte stream might be 0x12 0x04 for a pixel value (tUINT16), as it's little endian, it's 0x0412, but this is NOT the correct ADC value of this pixel (due to the hardware design), user might has to the following conversion:

```
RawValue = (tUINT16 *)*Streamptr;   // RawValue = 0x0412
TmpWord1 = RawValue >> 8;  // 0x04
TmpWord2 = (RawValue & 0xFF) << 4; // 0x0120
RealPixelValue = TmpWord1 + TmpWord2; // 0x124
```

2). For 8bit mode:

Typedef struct
{
tUINT16 **LightShield1[2];**
tUINT16 IsolatedCells1[2];
tUINT16 **ImageData[512];** // 1024 pixels
tUINT16 IsolatedCells2[2];
tUINT16 **LightShield2[2];**

tUINT16 Padding[232];
tUINT16 **TimeStamp;**
tUINT16 **ExposureTime;**
tUINT16 **TriggerOccurred;**
tUINT16 **TriggerEventCount;**
tUINT16 **GlobalGain;** *// User set it by the GreenGain of command 0x39.*
tUINT16 Padding2[11];
} tCCDFrames_8Bit;

For each frame, only elements in **BOLD** are meaningful, others are all dummy or paddings.

Note:
1). The above structure is only for one frame, it's 768 16bit words and 1536 bytes. For multiple frame fetching (the frame count in command 0x34 is more the ONE), the totally data fetching should be:
    1536 bytes x FrameCnt
There's no gap between frames.
2). The CCD's raw data is in ImageData[512] field, each element has two pixel values as its MSB and LSB, so 1024 pixel values are contained in 512 16bit words. Note that each pixel value should be less than 0xF8, otherwise the CCD sensor is "Over Exposured", user should reduce the exposure time to make it into normal range. Note that firmware won't do any automatic exposure time setting…etc., it's host's responsibility to set proper ET.

3). For the last five fields in **bold**, please refer to the SDK manual for the description of them, briefly, these are real time information the CCD sensor was used to generate THIS frame, which is extremely useful in applications.
4). As the CCD133A has two output channels, we have the following data format for channelA **(LSB)** and channelB **(MSB)**. Before user doing Data pre-processing below, it's recommended that user separate data of MSB and LSB, e.g. user might separate MSB and LSB to let the data has the similar format as 16bit mode above:

```
tUINT16  LightShield1[4];   // It has 4 elements now, they're actually LSB, MSB, LSB, MSB of the
                            // 2 tUINT16 LightShield1[2] of the above tCCDFrames_8Bit.
tUINT16  IsolatedCells1[4];
tUINT16  ImageData[1024];
tUINT16  IsolatedCells2[4];
tUINT16  LightShield2[4];
```

Note: Each element above only have 8 effective LSBs.
And then user should do pre-processing as following:
ChannelA: LightShield[0], [2], IsolatedCells1[0], [2], ImageData[0], [2]…[1022], IsolatedCells2[0], [2], LightShield2[0],[2]
ChannelB: LightShield[1], [3], IsolatedCells1[1], [3], ImageData[1], [3]…[1023], IsolatedCells2[1], [3], LightShield2[1],[3]

We expect user do the following pre-processing before ImageData[] are used.
*). Optional: For ***each channel***, User should generate Average LightShield (average of the 4 LightShield pixels), and subtract this value from all ImageData pixels of this channel.
*). For each channel, If the difference between the average of the two LightShields before the Image data and the average of the two LightShields after the Image data is bigger than 0x100 (256), User should think this frame is "Over Exposured".

****. TCX-1024-U Modal:***

1). For 16bit mode:

Typedef struct
{
tUINT16 **LightShield1[10];**
tUINT16 IsolatedCells1[2];
tUINT16 **ImageData[1024];**
tUINT16 IsolatedCells2[2];
tUINT16 **LightShield2[10];**
tUINT16 **ExposureTime;**
tUINT16 **TimeStamp;**
tUINT16 **TriggerOccurred;**
tUINT16 **TriggerEventCount;**
tUINT16 **GlobalGain;** *// User set it by the GreenGain of command 0x39.*
tUINT16 **FrameTime**; *// User set it by command 0x3A*
tUINT16 Padding2[2];
} tCCDFrames_16Bit;

For each frame, only elements in **BOLD** are meaningful, others are all dummy or paddings.

Note:
1). The above structure is only for one frame, it occupies 1056 16bit words and 2112 bytes. For multiple frame fetching (the frame count in command 0x34 is more the ONE), the totally data fetching should be:
2112 bytes x FrameCnt
There's no gap between frames, however, note that the whole byte count is multiple of 512bytes, thus camera will fill with bytes if (2112xFrameCnt) is not multiple of 512 bytes, for example, if it's 10 frames, the total bytes for frame data is 2112 * 10 = 21120 bytes, which is NOT multiple of 512 bytes. (21120 = 41x512 + 128), in this case, the camera will send back 42x512 = 21504 bytes, the filled 384 bytes in the end contains no-use content and should be ignored.
2). The CCD's raw data is in ImageData[1024] field, it's 12bit ADC values from the camera, user should pay attention that there's no data should be more than 0x0F80, otherwise the CCD sensor is "Over Exposured", user should reduce the exposure time to make it into normal range. Note that firmware won't do any automatic exposure time setting…etc., it's host's responsibility to set proper ET.
3). For the last six fields in **bold**, please refer to the SDK manual for the description of them, briefly, these are real time information the CCD sensor was used to generate THIS frame, which is extremely useful in applications.
4). The LightShield1[] and LightShield2[] are "Optical Black" pixels, We expect user do the following pre-processing before ImageData[] are used.
*). Optional: User should generate Average LightShield (average of the 6 LightShield pixels in the middle of the 10 LightShield Pixels), and subtract this value from all ImageData pixels of this channel.
5). The data stream from EP2 is in Byte, the above "tUINT16" is in little endian format, e.g. the byte stream might be 0x12 0x04 for a pixel value (tUINT16), as it's little endian, it's 0x0412, but this is NOT the correct ADC value of this pixel (due to the hardware design), user might has to the following conversion:
RawValue = (tUINT16 *)*Streamptr; // RawValue = 0x0412
TmpWord1 = RawValue >> 8; // 0x04
TmpWord2 = (RawValue & 0xFF) << 4; // 0x0120
RealPixelValue = TmpWord1 + TmpWord2; // 0x124

2). For 8bit mode:

Typedef struct
{
tUINT16 **LightShield1[5];**
tUINT16 IsolatedCells1[1];
tUINT16 **ImageData[512];** // 1024 pixels
tUINT16 IsolatedCells2[1];

tUINT16 **LightShield2[5];**
tUINT16 Padding[12];
tUINT16 **ExposureTime;**
tUINT16 **TimeStamp;**
tUINT16 **TriggerOccurred;**
tUINT16 **TriggerEventCount;**
tUINT16 **GlobalGain;** *// User set it by the GreenGain of command 0x39.*
tUINT16 **FrameTime**; *// User set it by command 0x3A*
tUINT16 Padding2[2];
} tCCDFrames_8Bit;

For each frame, only elements in **BOLD** are meaningful, others are all dummy or paddings.

Note:
1). The above structure is only for one frame, it's 544 16bit words and 1088 bytes. For multiple frame fetching (the frame count in command 0x34 is more the ONE), the totally data fetching should be:

1088 bytes x FrameCnt

There's no gap between frames. however, note that the whole byte count is multiple of 512bytes, thus camera will fill with bytes if (1088xFrameCnt) is not multiple of 512 bytes, for example, if it's 10 frames, the total bytes for frame data is 1088 * 10 = 10880 bytes, which is NOT multiple of 512 bytes. (10880 = 21x512 + 128), in this case, the camera will send back 22x512 = 11264 bytes, the filled 384 bytes in the end contains no-use content and should be ignored.
2). The CCD's raw data is in ImageData[512] field, each element has two pixel values as its MSB and LSB, so 1024 pixel values are contained in 512 16bit words. Note that each pixel value should be less than 0xF8, otherwise the CCD sensor is "Over Exposured", user should reduce the exposure time to make it into normal range. Note that firmware won't do any automatic exposure time setting…etc., it's host's responsibility to set proper ET.
3). For the last six fields in **bold**, please refer to the SDK manual for the description of them, briefly, these are real time information the CCD sensor was used to generate THIS frame, which is extremely useful in applications.
4). Before user doing Data pre-processing below, it's recommended that user separate data of MSB and LSB, e.g. user might separate MSB and LSB to let the data has the similar format as 16bit mode above:

tUINT16 **LightShield1[10];**
tUINT16 IsolatedCells1[2];
tUINT16 **ImageData[1024];**
tUINT16 IsolatedCells2[2];
tUINT16 **LightShield2[10];**

Note: Each element above only have 8 effective LSBs.

And we expect user do the following pre-processing before ImageData[] are used.
*). Optional: User should generate Average LightShield (average of the 6 LightShield pixels in middle of 10 LS pixels), and subtract this value from all ImageData pixels of this channel.