



Technical notes on using Analog Devices DSPs, processors and development tools
Contact our technical support at dsp.support@analog.com and at dsptools.support@analog.com
Or visit our on-line resources <http://www.analog.com/ee-notes> and <http://www.analog.com/processors>

Interfacing Micron® MT9V022 Image Sensors to Blackfin® Processors

Contributed by Thorsten Lorenzen

Rev 1 – April 22, 2005

Introduction

This EE-Note describes how to connect video devices such as the Micron® CMOS Image Sensor to the parallel peripheral interface (PPI) of Blackfin® processors. The PPI serves as a video interface to a host of CMOS sensors and video encoders and decoders. The system was tested on the ADSP-BF561 processor. Nevertheless, as the PPI interface of all Blackfin derivatives is fully compatible, this EE-Note applies to other Blackfin processors with PPI, as well. Because of their high performance architecture and video processing capabilities, Blackfin processors interface with a variety of video devices. This EE-Note describes the ADSP-BF561 Blackfin processor and Micron MT9V022 image sensor and includes schematics for the connection and example code for the transmission.

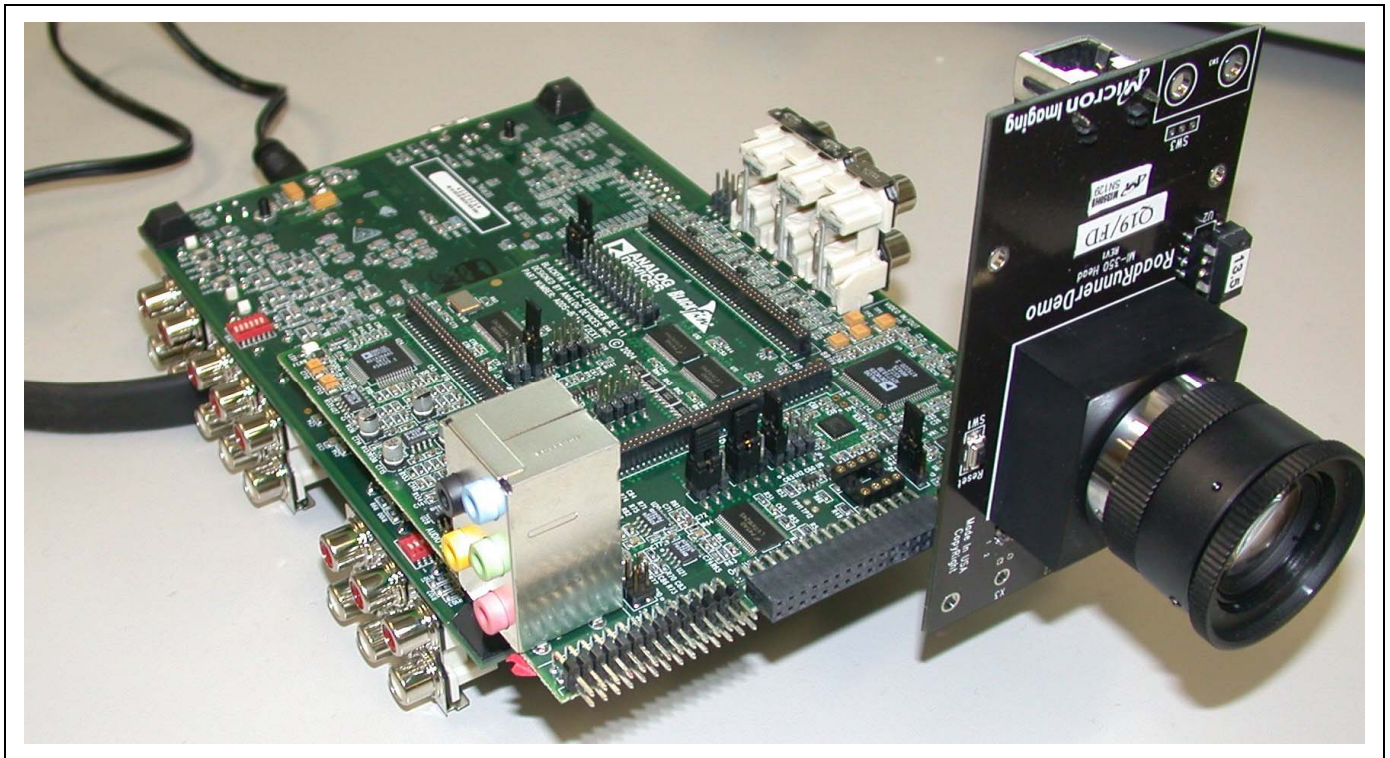


Figure 1. Video System

Micron MT9V022 Digital Image Sensor

The Micron MT9V022 is a wide-VGA, 1/3-inch optical format, CMOS digital image sensor with global shutter and high dynamic range operation up to 110 dB. It has been designed specifically to support the demanding interior and exterior automotive imaging needs, which makes this part ideal for a wide variety of imaging applications in real-world environments. The MT9V022 features DigitalClarity™, Micron's breakthrough, low-noise CMOS imaging technology that achieves CCD image quality (based on signal-to-noise ratio and low-light sensitivity) while maintaining the inherent size, cost, and integration advantages of CMOS. The active imaging array format is 752H x 480V (pixels). It incorporates sophisticated camera functions on-chip such as binning 2x2 and 4x4 to improve sensitivity. For more information, refer to the Micron Web site ^[4] (see section [References](#) at the end of this EE-Note).



This EE-Note uses the MT9V022 as an example to explain the detailed interface between a digital image sensor and a Blackfin processor. Derivatives of this sensor, including Micron Megapixel products that meet the video port specification of Blackfin processors, also connect directly to the Blackfin video port.

MT9V022 Output Format

The MT9V022 has a pixel array of 782 columns by 492 rows. The leftmost 26 columns and the top eight rows of pixels are optically black and can be used to monitor the black level.

The MT9V022 image data can be read in progressive scan mode or interlaced scan mode. Valid image data is surrounded by horizontal and vertical blanking, as shown in [Figure 3](#). The amount of horizontal and vertical blanking is programmable through a two-wire interface. By default, `LINE_VALID` is HIGH during the shaded region of [Figure 3](#). The data output of the MT9V022 is synchronized with the `PIXCLK` output ([Figure 2](#)). When `LINE_VALID` is HIGH, one 8/10-bit pixel datum is output every `PIXCLK` period as shown in [Figure 2](#). In this document, the default settings have been specified as follows:

- 8/10-bit data output
- Data is valid on the rising edge of `PIXCLK`
- `LINE_VALID` (`HSYNC`) active high
- `FRAME_VALID` (`VSYNC`) active high
- No `FIELD` signal is required

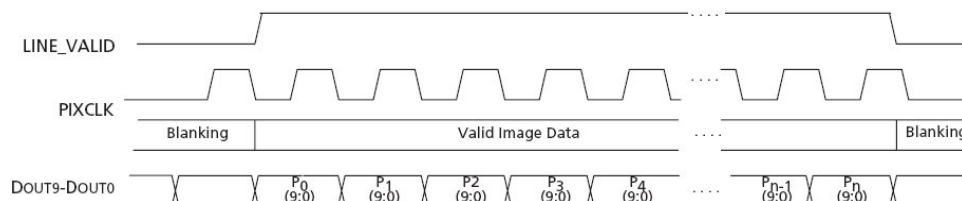


Figure 2. Output Data Timing

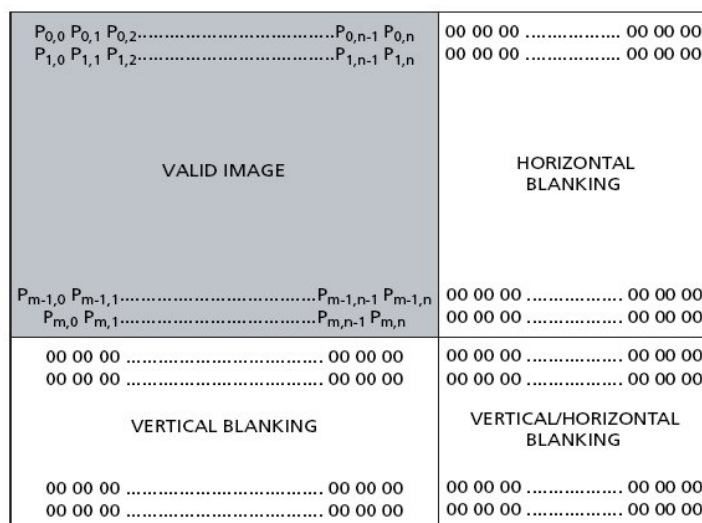


Figure 3. Image Read Out

In addition to the `LINE_VALID` signal, the `FRAME_VALID` signal indicates the start of a new frame. By default, the `FRAME_VALID` signal is HIGH (Figure 4) during the phase of valid image data as well as horizontal blanking as shown outside the vertical blanking interval VBI in Figure 3.

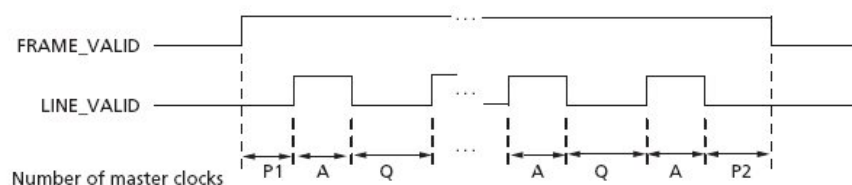


Figure 4. Frame Valid Timing

Connecting to the MT9V022

Micron offers head boards (e.g., Head350/360) for MT9V022 evaluation. Interface the Micron head board to the ADSP-BF561 EZ-KIT Lite® evaluation system board via a Blackfin A-V EZ-Extender board (see page 5). The Micron head board links all the required signals for video transmission and sensor programming to its connector (Figure 5). Refer to the *ADSP-BF561 EZ-KIT Lite Evaluation System Manual* [6] and the *Blackfin A-V EZ-Extender Manual* [7] for settings and connections. Figure 14 shows the schematic of the Blackfin A-V EZ-Extender board.

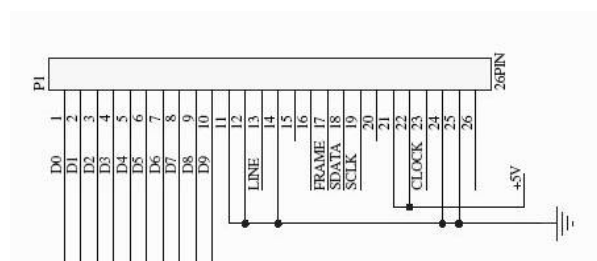


Figure 5. Micron Head Board Main Connector

Blackfin Processor Video Port Interface

Due to their multimedia processing capabilities, Blackfin processors are equipped with a minimum of one Parallel Peripheral Interface (PPI). The ADSP-BF561 Blackfin processor provides two of these interfaces.

The PPI is a half-duplex, bidirectional port that accommodates up to 16 bits of data. It has a dedicated clock pin, three frame sync pins, and eight dedicated data pins. Up to eight additional data pins are available by reconfiguring other pins (multiplexed pins). The highest system throughput is achieved with 8-bit data, since two 8-bit data samples can be packed as a single 16-bit word. In this case, the earlier sample is placed in the eight least significant bits (LSBs). The `PPIx_CLK` pin can accept an external clock input. It *cannot* source a clock internally.

The PPI supports two major categories of modes. In ITU-BT656 mode, only the clock input (`PPI_CLK`) and the parallel data bus are available. In this mode, the PPI recognizes the start of transmission by scanning the data bus for the embedded preamble. When the preamble occurs with the Start of Frame bits set, the PPI starts its transmission. In general-purpose (GP) mode, up to three additional pins can be used to support devices that are not compatible with the ITU-BT656 standard. In this mode, the horizontal synchronization signal (`HSYNC`), vertical synchronization signal (`VSYNC`), and the field indicator (`FIELD`) are used to detect the start of a transmission. For the example in this EE-Note, the PPI is configured to run in GP input mode with two external frame syncs.



The `FIELD` signal is not used in this example system.

Ensure that this pin is pulled to ground.

The PPI has a very flexible interface and can be configured to support many other devices. For example, in GP input mode the polarity of the clock can be inverted (via the `POLC` bit in the `PPI_CONTROL` register) to latch data at the rising edge (`POLC` = 0) or at the falling edge (`POLC` = 1). In addition, the polarity of the frame sync signals (`HSYNC`, `VSYNC`) can be inverted (via the `POLS` bit in the `PPI_CONTROL` register). When `POLS` is cleared (`POLS` = 0), the `HSYNC` and `VSYNC` signals are treated as active high. When `POLS` is set to one (`POLS` = 1), they are active low. As discussed earlier, configuring the PPI data bus to be eight bits wide results in the best efficiency because any DMA (programmed to be 16 bits wide) data fetch to the PPI can take two samples. If required for better quality, the PPI bus can be extended to up to 16-bits successively. In this case, a single sample will be transferred at a single DMA fetch. Still a maximum of sixty-six million samples per second (66 MS/s) can be achieved while the DMA bus bandwidth is even higher. Derivative processors equipped with an internal 32-bit bus structure (such as the ADSP-BF561) can transfer two 16-bit samples at a time instead. Another important feature is the PPI delay count register (`PPI_DELAY`) (see [Figure 6](#)), which contains the number of `PPI_CLK` cycles of delay after assertion of `PPI_FS1` (`HSYNC`) before start of when read data is read in.

In general, the PPI can be considered to be a static state machine driven by the PPI clock (`PPI_CLK`). Whenever the PPI clock is provided, the state machine is able to latch data and/or frame syncs. After being enabled when programmed to 2 or 3 frame sync input mode, it remains in an idle state until assertion of vertical frame sync (`VSYNC` `PPI_FS2`) is followed by the assertion of horizontal frame sync (`HSYNC` `PPI_FS1`) while the signal `FIELD` is low. The PPI starts transmission and stops when it has fetched the number of samples programmed in the PPI count register (`PPI_COUNT`). If it is limited to a number of samples, the transmission stops after the counter expires ([Figure 6](#)). The transmission will be reactivated after assertion of the horizontal frame sync due to the next active video line. This way it can be ensured that just the active video image is loaded without blanking. [Figure 6](#) shows the characteristic discussed

here. It also shows that the frame sync edges. When the state machine detects the assertion of either frame sync, it latches the state. Therefore, `PPI_FS1` and `PPI_FS2` do not need to remain active during transmission (dotted lines). As explained earlier, when the `PPI_COUNT` expires the transmission stops.

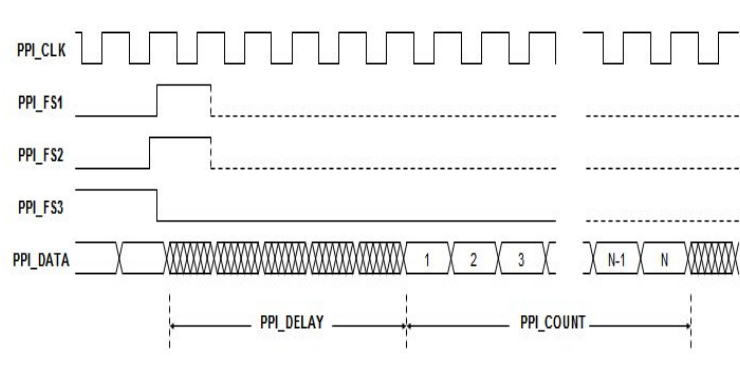


Figure 6. PPI Input Start Condition

ADSP-BF561 EZ-KIT Lite Evaluation System and Daughter Boards

The ADSP-BF561 EZ-KIT Lite evaluation system includes an ADSP-BF561 processor desktop evaluation board and an evaluation suite of the VisualDSP++® development and debugging environment with a C/C++ compiler, assembler, and linker. It also includes sample processor application programs, a CE-approved power supply, and a USB cable. Analog Devices offers daughter boards that extend the capabilities of the EZ-LIT Lite board.

The ADSP-BF561 EZ-KIT Lite board provides developers with a cost-effective method for initial evaluation of the ADSP-BF561 Blackfin processor for audio and video applications via a USB-based PC-hosted tool set. Evaluation of analog audio applications is achieved through the board's AD1836 multi-channel 96 kHz audio codec. Utilize the board's ADV7183A advanced 10-bit video decoder and the ADV7179 NTSC/PAL video encoder to evaluate video applications such as simultaneous input and output video processing enabled by the dual-core architecture of the ADSP-BF561 processor. With this EZ-KIT Lite evaluation system, you can learn about Analog Devices ADSP-BF561 hardware and software development and prototype applications quickly.

Part Number: ADDS-BF561-EZLITE

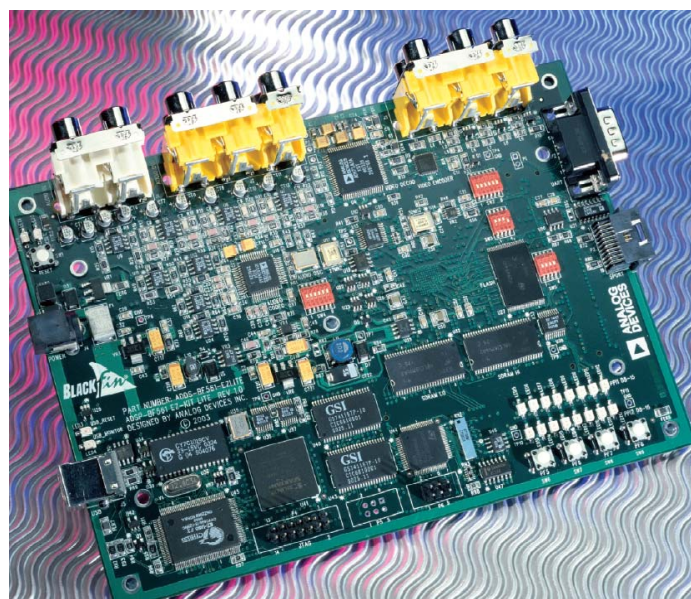


Figure 7. ADSP-BF561 EZ-KIT Lite Board

Part Number: ADDS-BF-AV-EZEXT

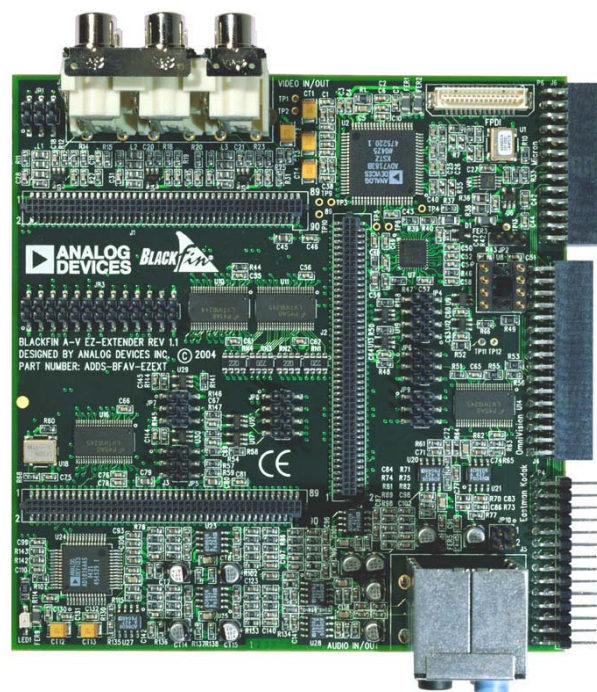


Figure 8. Blackfin A-V EZ-Extender Board

The Blackfin A-V EZ-Extender (daughter board) allows developers to connect the Parallel Peripheral Interface (PPI) on Blackfin processors to the Analog Devices standard PAL/NTSC video decoder and encoder (ADV7183 and ADV7179). In addition, you can connect digital image sensor boards like the Micron head board (MI350_head_V01) discussed in this document and an external LCD display via four different connectors.



For more information on Analog Devices tools, refer to the following web page.

<http://www.analog.com/processors/productsDatasheets/evaluationKits.html>

Blackfin Processor-to-Image-Sensor Interface

Figure 9 shows the general connections from the sensor board to the PPI. In cases where video boards do not offer a clock, use an external oscillator or a Blackfin PWM timer. The Blackfin PPI interface offers all the signal inputs to typical image sensors. It's very easy to interconnect those devices, and no extra glue logic is required.



Because the Field signal is not used in this system, the field pin (PPIx_FS3) must be connected to ground. Please check your hardware as this may need to be done manually.

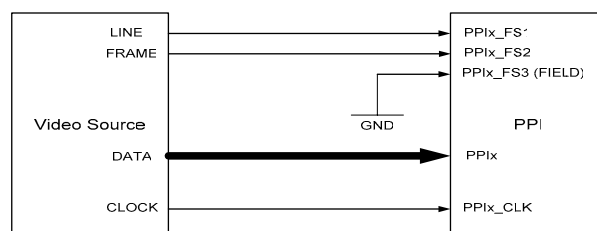


Figure 9. High-Level Interface Scheme

As mentioned earlier, Micron offers a head board unit that can be plugged into the Blackfin A-V EZ-Extender board via connector J6. The MI350_head_Vxx consists of an image sensor, oscillator, power regulator, and a standard connector (P1). Figure 14 in this document's Appendix shows the schematic of the head unit. In turn, the Blackfin A-V EZ-Extender board must be plugged into the ADSP-BF561 EZ-KIT Lite board. Figure 1 is a photograph of the system, showing all the boards connected together. To ensure proper operation, set switches and install jumpers according to Table 1 and Table 2.

| | |
|-----|--|
| SW2 | 6 > off. Keeps the video decoder from driving PPI0. |
| SW3 | 1 > off, 2 = off. Non flash boot ensures that PPI0 does not drive against the image sensor output. |
| SW5 | 2 > off. Links the external sourced clock to PPI0. |

Table 1. ADSP-BF561 EZ-KIT Lite Settings (All other switches should remain in their default state.)

| | |
|-----|--|
| JP1 | Not applicable. This is not relevant to this application. |
| JP2 | 2-3. Puts 5V to the image board. Note that boards other than Micron may require a 3V supply (1-2). |
| JP3 | 3-5, 4-6. I2C connection. Connections to the ADSP-BF537 EZ-KIT Lite would require different settings (5-7, 6-8). |
| JP4 | 1-2. Links the image sensor output clock to PPI0. An external clock source may be required for image sensors other than Micron (such as Kodak image sensors). Therefore, a jumper from 7-8 supports those devices. |
| JP5 | 1-2. Enables the PPI 16-bit bus. In case to run an image sensor in resolutions higher than eight bit. |
| JP6 | 3-5, 4-6. Controls the direction of FS1 (HSYNC) and FS2 (VSYNC) so that the image sensor signals drive PPI0 |
| JP7 | Not applicable. This is not relevant to this application. |
| JP8 | Not applicable. This is not relevant to this application. |
| JP9 | 3-5. Forces the reset per reset button. 4-6 to drive the sensor data bus into PPI0 (direction control). |

Table 2. Blackfin A-V EZ-Extender Jumper Settings (Other jumpers need not be populated.)

Debugging Video Applications with VisualDSP++ Tools

Suitable for many multimedia applications, Blackfin processors are supported by many debugging utilities available in the VisualDSP++ tools. Today, VisualDSP++ provides an Image Viewer tool that allows you to display video images stored in internal or external memory. By selecting the image width, height, and data format in which it is stored, the Image Viewer will download the data and display it as a regular image. Without any additional tools, images can be rendered using VisualDSP++. Figure 11 shows how an image is configured via the Image Configuration dialog box. The PPI port is capable of allowing the entire video stream to pass (active video plus blanking) or selectively streaming either active video or blanking data. Therefore, the `width` and `height` settings can be modified to display the image in its proper form. These settings also need to be modified to display different image sizes. The `Start` address field points to the starting location of where the image is located in memory, whereas `Pixel` format provides a collection of supported data formats. Additionally, images can be modified (gamma adjust, rotation, and flipping) and stored on hard disk in JPG and BMP formats.

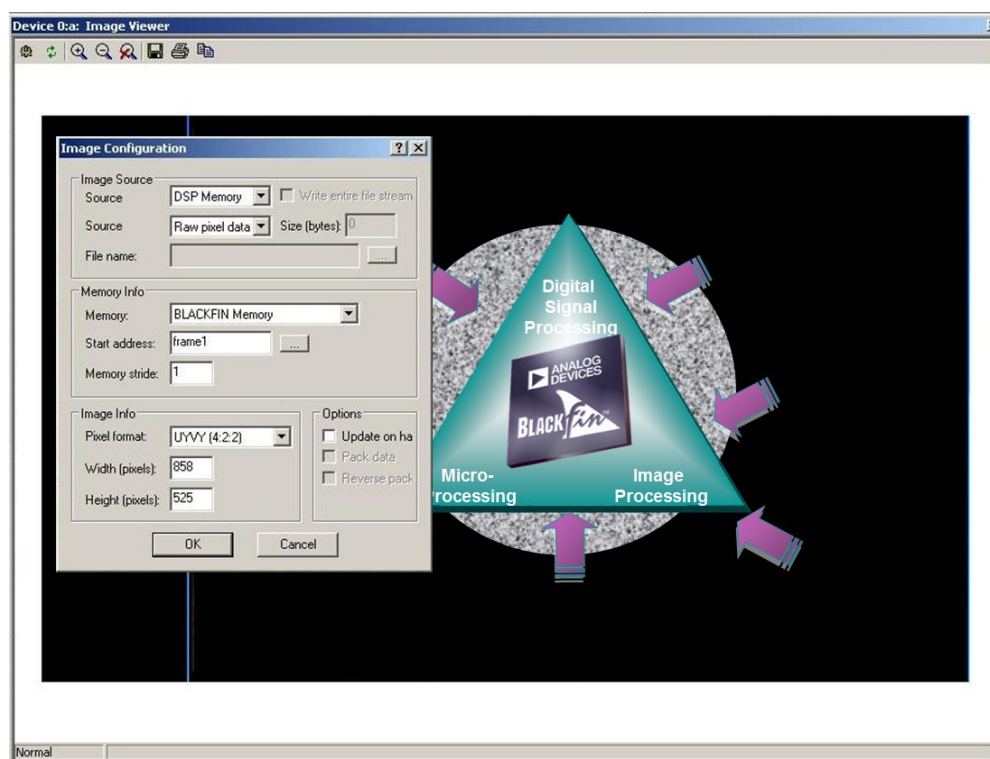


Figure 11. VisualDSP++ 4.0 Image Viewer

Another powerful tool that helps you build and debug image processing applications is the plotting tool. Use the plot capabilities in VisualDSP++ to display the results of an image being processed by certain algorithms in different views. For example, running the Hough transform over an image will result in many vectors stored in the Hough space storage. To investigate which vector points to a possible line in the image, view a spectrogram plot and a waterfall plot. Visually, the spectrogram plot shows the vectors and its importance in terms of the number of pixels that refer to this vector. For object recognition algorithms, spectrogram plots can be used to find the location of objects with the highest correlation to an object of interest. The spectrogram plot shows memory locations in two-dimensions (2-D); waterfall plots show the same locations in 3-D. Therefore, waterfall plots display the same results in the spatial domain.

For details, refer to the *VisualDSP++ User's Guide* ^[5] or VisualDSP++ online Help.

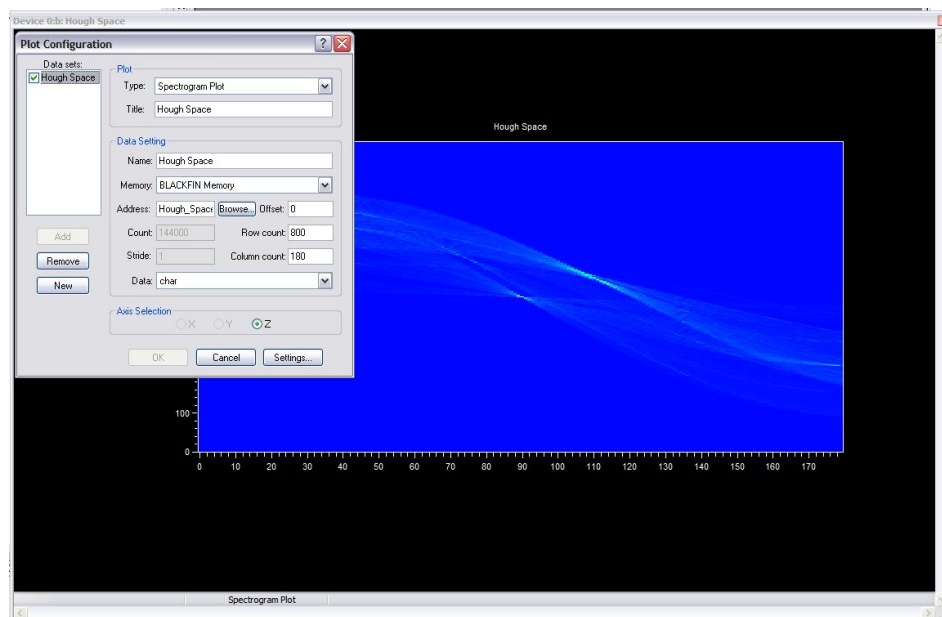


Figure 12. VisualDSP++ Spectrogram Plot

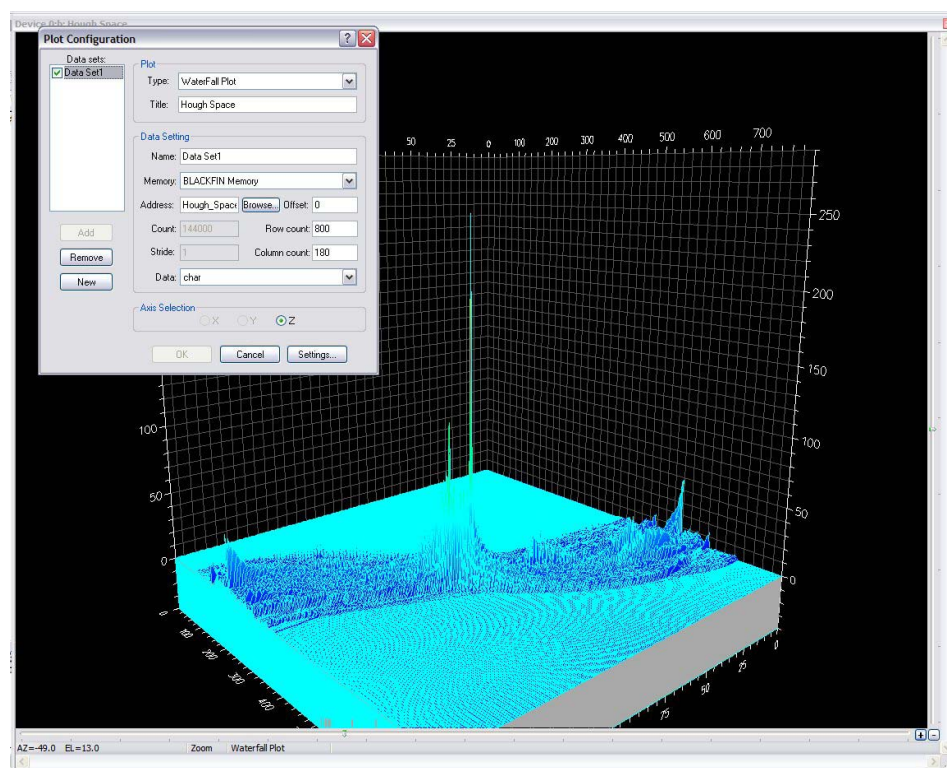


Figure 13. VisualDSP++ Waterfall Plot

Coding Example

The following section discusses the file `main.c` as shown in Listing 4. The file `main.c` shows how to program the Blackfin processor to receive proper images.

First, an array (`VideoInputFrame`) must be defined to store the received image. Since the active image contains 345,600 bytes (720x480 8-bit gray scale pixels) the array must be stored in external memory (i.e., SDRAM). This can be done with the `section` directive. In this case, the array is located in the `sdr_data` input section, which is defined in the file `adsp-bf561_C.ldf`. Refer to:

C:\Program Files\Analog Devices\VisualDSP 3.5 16-Bit\Blackfin\ldf\adsp-bf561_C.ldf

```
section ("sdr_data")
volatile unsigned short VideoInputFrame[LINES_PER_FRAME][PIXEL_PER_LINE/2];
```

Listing 1. Array Definition



The image array is defined as unsigned short (16-bit values). Therefore, each line in the array must be half the size of the number of pixels provided by the sensor (8 bits wide).

The `MI350_init()` function within `main.c` programs the sensor via the I2C protocol. Because the ADSP-BF561 Blackfin processor does not have an on-chip I2C interface, two of the programmable flags are used to emulate this function (GPIOs). Due to license restrictions, the I2C routine is not available here. Contact processor.support@analog.com or processor.europe@analog.com for a license agreement, or search the Internet with dedicated keywords.

```
void main() {

    usDestAddr = &VideoInputFrame[0][0];           // init the pointer with the start
                                                    // addr of the array

    // initialise SDRAM
    InitSDRAM();

    // initialise Micron Sensor
    MI350_init();                                   // Calls an I2C function to program the sensor

    VideoSensorFrameCapture ( (void*) usDestAddr ); // function call to config

    while(1);

};
```

Listing 2. `main()` Function

Finally, the `VideoSensorFrameCapture()` function is called to set up the data path (Listing 3).

In this example, channel 0 of DMA Controller 1 has been chosen to transmit the video stream received by the PPI interface. Since the DMA channel is set up for 2-D mode, two DMA counters (X and Y) must be initialized. The X count specifies the number of transmissions per video line, and the Y count specifies the number of lines to be transmitted per video frame. This can be considered a loop within a loop. The inner loop (X count) controls the iterations per video line, and the outer loop (Y count) controls the iteration of

the inner loop. However, as long as the inner loop and outer loop have the same modifier value (`X_MODIFY` and `Y_MODIFY`), the DMA behaves like a linear DMA. Because this DMA channel is configured to transmit 16-bit values, the modifiers must be set to "2". The DMA address pointer (`*pDMA1_0_START_ADDR`) points to the array defined earlier. Again, as the DMA is set up to a word size of 16 bits, the address must be a multiple of two. The DMA Configuration register (`*pDMA1_0_CONFIG`) holds the major settings. In this example, the DMA is configured for autobuffer mode, which makes the DMA controller reload the configurations automatically after completion of a transfer (all counters expired). The DMA controller continuously transfers images to the SDRAM memory and overwrites the old images. The `DMA2D` field in the DMA Configuration register enables the 2-D DMA mode by using the Y count and Y modifier. The `WDSIZE_16` field enables the 16-bit data bus mode of the DMA. `WNR` directs the DMA controller to write to (versus read from) memory.

The PPI frame register (`*pPPIO_FRAME`) specifies the expected number of lines per frame to be received. If the number in this register does not match the actual received number, the status bit `FT_ERR` (in the `PPIO_STATUS` register) will be set. Monitor the `FT_ERR` bit to find possible transmission problems. This error bit can also be set to generate an interrupt upon occurrence. The PPI Count register (`*pPPIO_COUNT`) specifies the number of samples to fetch per line. After expiration, the PPI interface stops receiving data until the next assertion of `HSYNC` (`PPIO_FS1`). Therefore, the PPI Count register needs to account for the number of pixels in either active video only or blanking only modes. The PPI Control register (`*pPPIO_CONTROL`) holds the major configurations to run different PPI modes. If the `POLS` bit is cleared, `FS1` and `FS2` are treated as rising edge asserted. Before the PPI starts transmission, it waits for the asserted edge of `FS1` and `FS2`. If `POLS` is set, the falling edge of `FS1` and `FS2` initiate transmission. For this application, `POLS` is cleared. Therefore, the PPI starts transmission at the rising edges of `FS1` and `FS2`. The `POLC` bit controls whether to sample data at the rising or falling edge of the PPI input clock (`PPIO_CLK`). For this example, `POLC` is cleared. The PPI samples data at the rising edge of any PPI clock cycle. The `DLEN` field is programmed to specify the width of the PPI port. Here it is programmed to be eight bits wide. The upper eight bits to the maximum bus width of 16 bits are mux'ed with the programmable flags, as well. However, image sensor resolutions higher than eight bits are supported also by programming the `DLEN` field to a bus width greater than eight. Since the PPI port is programmed to a width of 8 bits, packing can be enabled. With packing enabled, every second sample (eight bits each) stored in the PPI FIFO will trigger a request to the dedicated DMA channel to fetch two samples (a total of 16 bits). The `PORT_CFG` field enables the PPI to operate in 2 or 3 external frame sync mode. The PPI starts transmission after assertion of `FS1` and `FS2` while `FS3` is low. Finally, the `XFR_TYPE` field turns the PPI interface into non-ITU-BT656 input mode. However, the PPI is still disabled. The DMA channel must be enabled first; otherwise, it will not be able to service any request from the PPI interface. The DMA channel will not start accessing the PPI FIFO after it is enabled because the FIFO is still empty. After the PPI interface is enabled, it will store the first sample into its FIFO, which will trigger the first request to the DMA channel.


```

void VideoSensorFrameCapture ( unsigned short *usPTR ){

    // Configure DMA for PPI0
    *pDMA1_0_X_COUNT = PIXEL_PER_LINE/2;           // pixels per line to store in mem
    *pDMA1_0_Y_COUNT = LINES_PER_FRAME;           // lines per frame to store in mem
    *pDMA1_0_X_MODIFY = 2;                         // Modifier 2 because of 16-bit DMA
                                                    // mode

    *pDMA1_0_Y_MODIFY = 2;                         // Modifier 2 because of 16-bit DMA
                                                    // mode
    *pDMA1_0_START_ADDR = usPTR;                   // Destination address of the image

    // Autobuffer mode | 2-D DMA | Bus width 16 bit | write to memory
    *pDMA1_0_CONFIG = AUTOBUFFERMODE | DMA2D | WDSIZE_16 | WNR;

    // PPI0 setup
    *pPPI0_FRAME = LINES_PER_FRAME; // All lines received? (Frame track error)
    *pPPI0_COUNT = PPICOUNT;         // The PPI is set to stop receiving after X
                                    // number of samples for each line

    // Polarity | 8-bit bus | packing | two/three frame syncs | Input mode
    *pPPI0_CONTROL = POL_S | POL_C | DATALEN | DATAPACKING | CFG_GP_INPUT_3SYNCS |
        GP_INPUT_MODE;

    *pDMA1_0_CONFIG |= DMAEN;           // | Start the DMA channel
    ssync();

    *pPPI0_CONTROL |= PORT_EN;          // | Start PPI interface
    ssync();

}

```

Listing 3. VideoSensorFrameCapture() Function

Appendix

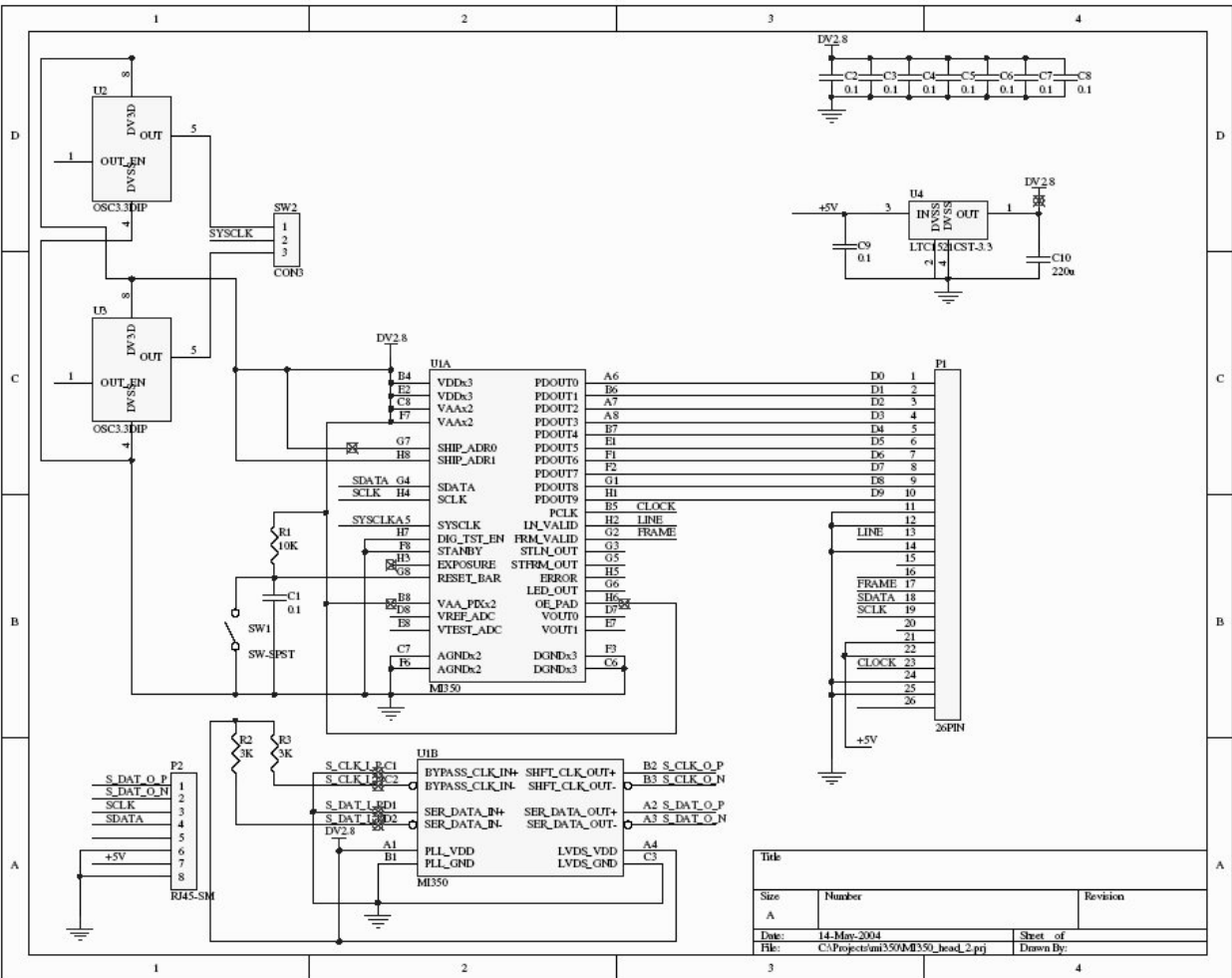


Figure 14. MI350 Micron Image Sensor Head Board

Main.c

```

/*****
/*****
/**
/** (C) Copyright 2005 - Analog Devices, Inc. All rights reserved.
/**
/*****
/*****

/*****
/*          ADI DSP Division, EMP, Munich          */
/*****
/*
*
* FILE NAME : main.c
*
* DATE      : $Date: 2005/04/12 06:59:32 $
*
* AUTHOR(S) : $Author: ANALOG\tlorenz $
*
* REVISION NO. : $Revision: 1.1.1.1 $
*
* HISTORY    : $Log: main.c,v $
* HISTORY    : Revision 1.1.1.1  2005/04/12 06:59:32  ANALOG\tlorenz
* HISTORY    : start point
* HISTORY    :
* HISTORY    :
*
* DESCRIPTION : Configuration via I2C and download images from the
*               Micron image sensor "MT9V022"
*
* SOFTWARE    : VisualDSP++4.0
*
* HARDWARE    : ADSP-BF561 EZ-KIT REV 1.3
*
*   date      author      changes
*   04-JAN-2005 T. Lorenzen    final release
*
*
*****/

#include <cdefBF561.h>
#include <ccblkfn.h>
#include <stdio.h>
#include <sys\exception.h>
#include "main.h"

#define CFG_GP_INPUT_3SYNCS 0x0020
#define GP_INPUT_MODE      0x000C
#define AUTOBUFFERMODE     0x1000

```

```

/*****
/***** Settings for the Micron sensor *****/
/*****
/*
/* Switches on the ADSP-BF561 EZ-KIT Rev 1.3:
/*   sw2: all off
/*   sw3: 1=off 2=off 3=on 4=off
/*   sw5: 1=off 2=off 3=on 4=on
/*
/* Image viewer settings:
/*   Column:720, Rows:480, grayscale image 8bit
/*
/* Blackfin AV EZ-Extender Rev 1.1 settings:
/*   JP1: do not populate      (DNP)
/*   JP2: Pin 2-3
/*   JP3: Pin 3-5, 4-6
/*   JP4: Pin 1-2
/*   JP5: Pin 1-2
/*   JP6: Pin 3-5, 4-6
/*   JP7: DNP
/*   JP8: DNP
/*   JP9: 3-5, 4-6
/*   JP10:DNP
/*
*****/

#define POL_C          0x0000    // Clock polarity setting
#define POL_S          0x0000    // Frame sync polarity setting
#define PIXEL_PER_LINE 720      // # of samples expected from the sensor
#define LINES_PER_FRAME 480     // # of lines expected from the sensor
#define DATALEN       0x0000    // 8-bit bus 0x0000, 16-bit bus 0x3800
#define DATAPACKING     0x0080    // packing enabled=0x0080
#define PPICOUNT        719

/*****
/***** Prototypes *****/
/*****

void VideoSensorFrameCapture ( unsigned short *usPTR );
void InitSDRAM(void);

/*****
/***** variable definition *****/
/*****

section ("sdr_data")
volatile unsigned short VideoInputFrame[LINES_PER_FRAME][PIXEL_PER_LINE/2];
volatile unsigned short *usDestAddr;

extern int result;

/*****
void main()
{
    usDestAddr = &VideoInputFrame[0][0];    // init the pointer with the start
                                              // addr of the array

    // initialise SDRAM

```



```

InitSDRAM();

// initialise Micron Sensor
MI350_init(); // Calls an I2C function to program the sensor

VideoSensorFrameCapture ( (void*) usDestAddr ); // function call to config

while(1);

};

void VideoSensorFrameCapture ( unsigned short *usPTR )
{
    // configure DMA for PPI0
    *pDMA1_0_X_COUNT = PIXEL_PER_LINE/2; // pixels per line to transfer
    *pDMA1_0_Y_COUNT = LINES_PER_FRAME; // lines per frame to transfer
    *pDMA1_0_X_MODIFY = 2; // Modifier 2 because of 16-bit
    *pDMA1_0_Y_MODIFY = 2; // DMA mode

    *pDMA1_0_START_ADDR = usPTR; // Destination address of the image

    // Autobuffer mode | Restart FIFO | 2-D DMA | Bus width 16 bit | write to memory
    *pDMA1_0_CONFIG = AUTOBUFFERMODE | RESTART | DMA2D | WDSIZE_16 | WNR;

    // PPI0 setup
    *pPPI0_FRAME = LINES_PER_FRAME; //The PPI is set to receive X lines per frame
    *pPPI0_COUNT = PPICOUNT; //The PPI is set to stop receiving after X
    // number of samples for each line

    // Polarity | 8-bit bus | packing | two/three frame syncs | Input mode
    *pPPI0_CONTROL = POL_S | POL_C | DATALEN | DATAPACKING | CFG_GP_INPUT_3SYNCS
    | GP_INPUT_MODE;

    *pDMA1_0_CONFIG |= DMAEN; // | DMA enable
    ssync();

    *pPPI0_CONTROL |= PORT_EN; // | Start PPI
    ssync();
}

void InitSDRAM(void)
{
    // Check if already enabled
    if (*pEBIU_SDSTAT & SDRS) {
        *pEBIU_SDBCTL = 0x00000015; //SDRAM Memory Bank Control Register
        ssync();
        *pEBIU_SDRRC = 0x000003a9; //SDRAM Refresh Rate Control Register
        ssync();
        *pEBIU_SDGCTL = 0x0091998f; //SDRAM Memory Global Control Register
        ssync();
    }
}

```

Listing 4. main.c

References

- [1] *ADSP-BF561 Blackfin Processor Hardware Reference*. Preliminary Revision 0.3, January 2005. Analog Devices, Inc.
- [2] *ADSP-BF561 Blackfin Embedded Symmetric Multi-Processor Data Sheet*. Rev. PrD, Analog Devices, Inc.
- [3] <http://www.analog.com/processors/productsDatasheets/evaluationKits.html>
- [4] <http://www.micron.com/products/imaging/>
- [5] *VisualDSP++ 4.0 User's Guide*. Rev 1, January 2005. Analog Devices, Inc.
- [6] *ADSP-BF561 EZ-KIT Lite Evaluation System Manual*, Revision 2.0, January 2005. Analog Devices, Inc.
- [7] *Blackfin A-V EZ-Extender Manual*, Revision 1.0, January 2005. Analog Devices, Inc.

Document History

| Revision | Description |
|--|-----------------|
| <i>Rev 1 – April 22, 2005 by T. Lorenzen</i> | Initial Release |