

Diamond Video Library

V1.0

6 August 2007

Introduction

This document describes DVL, the Diamond Video Library, developed to simplify using the SMT339 module for common video-processing applications. The DVL is designed to be used in conjunction with Diamond V3.1.9 or later.

Overview

The SMT339 provides video input and video output using the TMS320DM642 from Texas Instruments. The hardware is extremely flexible but that comes at the price of enormous complexity. It can take a long time to determine values for all the relevant registers of the numerous components in the processor; a slight error can result in bizarre effects ranging from no input or output at all to strangely corrupted images.

The DVL deals with most of that complexity behind the scenes and instead provides a simple yet powerful interface that will allow you to develop your video algorithms quickly.

Definitions

The following definitions will be used in this document.

| | |
|--------------|--|
| Field | All the odd lines or all the even lines of a picture. |
| Frame | A single picture formed by interlacing an odd field (field 1) and an even field (field 2). |

Current Limitations & Issues

1. The V1.0 library has support for PAL only. Other standard formats, in particular NTSC, will be introduced in the near future.
2. The only supported format is BT.656.
3. There is currently no complete reset mechanism in the standard firmware for the SMT339. This can lead to unexpected and persistent erroneous behaviour should faulty values be sent to video control registers, for example, if you build and run code that does not disable caching on the external memory as discussed in the [Important Note](#) below. The only known solution to this problem is to power-cycle the module. Sundance is working on new firmware to address this omission.
4. Some versions of the Sundance firmware for the SMT339 swop **Cb** and **Cr** on output. If you find that blue and red are swopped in your output images, you should contact Sundance for an update to your firmware. A temporary workaround is to interchange **Cb** and **Cr** explicitly yourself in the write tables.

Processor Types

The number and names of the SMT339 variants available from Sundance are currently in flux. You should use the appropriate Diamond processor types from the following list. Note that if you have load checking enabled and you select the wrong processor type, you may need to select a processor type starting with “**PROTO_**” rather than using exactly the name supplied in the error message (which comes from a Sundance DLL).

```
SMT339_FX60
PROTO_SMT339_FX60_64_600
PROTO_SMT339_FX60_128_600
PROTO_SMT339_FX60_64_720
```

Once the designations of these modules have been stabilised, 3L will issue updates to Diamond to support the corresponding new processor type names.

Important Note

The TMS320DM642 does not have very much internal memory, and so the data used for video will probably be held in external memory. The cache mechanism on the C6000 processors is unable to maintain coherence with external memory when DMA is used; a DMA transfer to external memory will change the memory contents but leave the cache holding old data. The effect is that the CPU will be given the old data from the cache when it expects to be reading the new data from memory.

There are two solutions to this nasty problem:

1. The software can explicitly flush the cache before each and every DMA operation involving external memory. This is *extremely* slow.
2. The external memory can be marked as not being cached. While this affects the apparent performance of the external memory, it is often much more efficient than flushing the cache.

DVL assumes that the second option will be used. Applications that use the library must explicitly disable the cache by adding an **uncached** attribute to the processor statement in the configuration file as in the following example:

```
PROCESSOR root PROTO_SMT339_FX60_64_600 uncached=0x80000000:0x83FFFFFF
```

If you are using the Diamond IDE, this can be achieved by adding the following to the Advanced/Extra Options box for the processor:

```
uncached=0x80000000:0x83FFFFFF
```

Different SMT339 variants have differing amounts of external memory; you should adjust the upper limit of the **uncached** area, **0x83FFFFFF** in the examples above, to match the amount of available memory on the particular module you are using.

Accessing the Library

The DVL is supplied as an object library, **DVL.lib**, supported by a C header file, **DVL.h**.

The header file also provides a structure, **VPREGS**, that can be used to access the video processor registers explicitly, if necessary.

Computational Model

The TMS320DM642 provides FIFOs for the input and output of video data. These FIFOs are of limited size, giving only just over three lines of buffering for a typical image. The consequences of this are different for input and output.

On input, it is possible to read images explicitly frame by frame using **VP_Read**. The standard settings throw away the even fields; consecutive reads present the odd fields.

On output, the FIFOs empty rapidly and it is impossible to keep them fed with data from explicit, separate calls. Instead, the library will start a Diamond thread that uses EDMA to keep the output FIFOs full. The EDMA takes data from a *write table* that supplies a number of pairs of fields (odd and even); each field includes pointers to **Y**, **Cb**, and **Cr** memory areas. The two fields in a pair will often reference the same memory areas, so the same data will be presented for both of the interlaced output fields. Each pair of fields in the table will be transmitted in order. Once the final pair of fields has been transmitted, the data will start to be taken from a new write table that may have been supplied by the user. The transmission of the final pair of fields from the current table will be repeated if necessary until a new write table has been presented.

Heap Requirement

Applications using the DVL must have at least 2KB of heap space available for the output thread that handles write tables.

Data Format

Data for each field are held in memory as three arrays corresponding to the **Y**, **Cb** and **Cr** components of the image. The encoding of these components is described in the Texas Instruments document “*TMS320C64x DSP Video Port/VCXO Interpolated Control (VIC) Port*”, document number *SRPU629E*. The **Y** area is twice the size of the **Cb** and **Cr** areas, and all areas must start on a double-word boundary.

Typical Operation

An application will start by setting all the necessary values in the various registers of the video components: the *encoder*, *decoder*, *capture*, and *display* modules. This is done using the function **VP_Init_Transfer**, which takes a definition of the video standard to be used, the number of *lines* in a field, and the number of *pixels* in each line of the field.

It may be possible to adjust specific register values explicitly following the call to **VP_Init_Transfer**.

Next, the output thread must be started using the function **VP_Start_Write**, which takes a pointer to the video port used for output (**VPI**), the number of lines in each field, and an initial write table that gives starting data to be sent to the display. This starts the continuous EDMA operation to display data. Applications that do no video output can omit this call.

The application can now read fields from the camera using **VP_Read**, process them, and build up the data referenced by a new write table. When complete, the function **VP_Use** can be called to make the output thread start displaying the new table.

The Sample Code

The sample code gives an example that can operate in three modes determined by the value of the macro **FUNCTION**:

- BARS** Two frames are calculated to give eight vertical bars across the output screen. One frame is a greyscale and the other is in colour. Two diagonal lines are also drawn on the frames. These frames are displayed alternately every second. See illustrations 1 & 2.
- STATIC** Four frames are captured from the camera and then displayed in pairs, alternating every second.
- LIVE** A loop reads pairs of frames from the camera and sends them to be displayed. Every five seconds the display toggles between the unaltered image and the image after a simple threshold function has been applied. See illustrations 3 & 4.

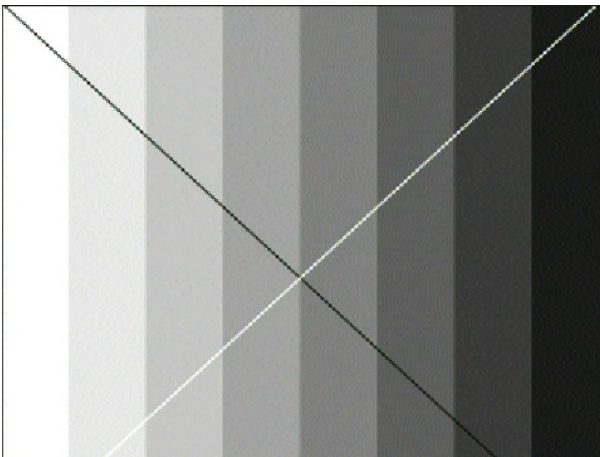


Illustration 1: Greyscale frame of BARS output

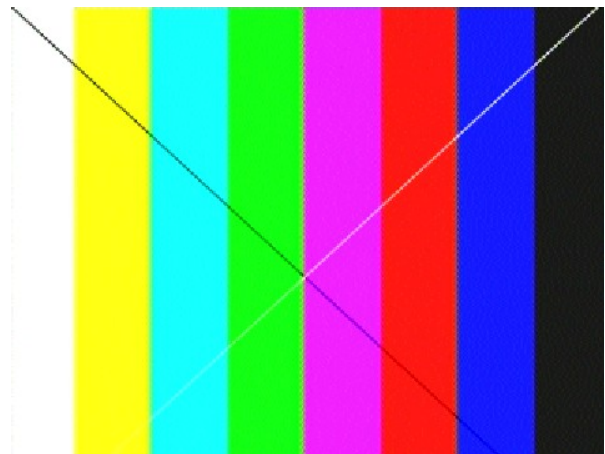


Illustration 2: Colour frame of BARS output



Illustration 3: LIVE: original data



Illustration 4: LIVE: threshold applied

Running the Sample Code

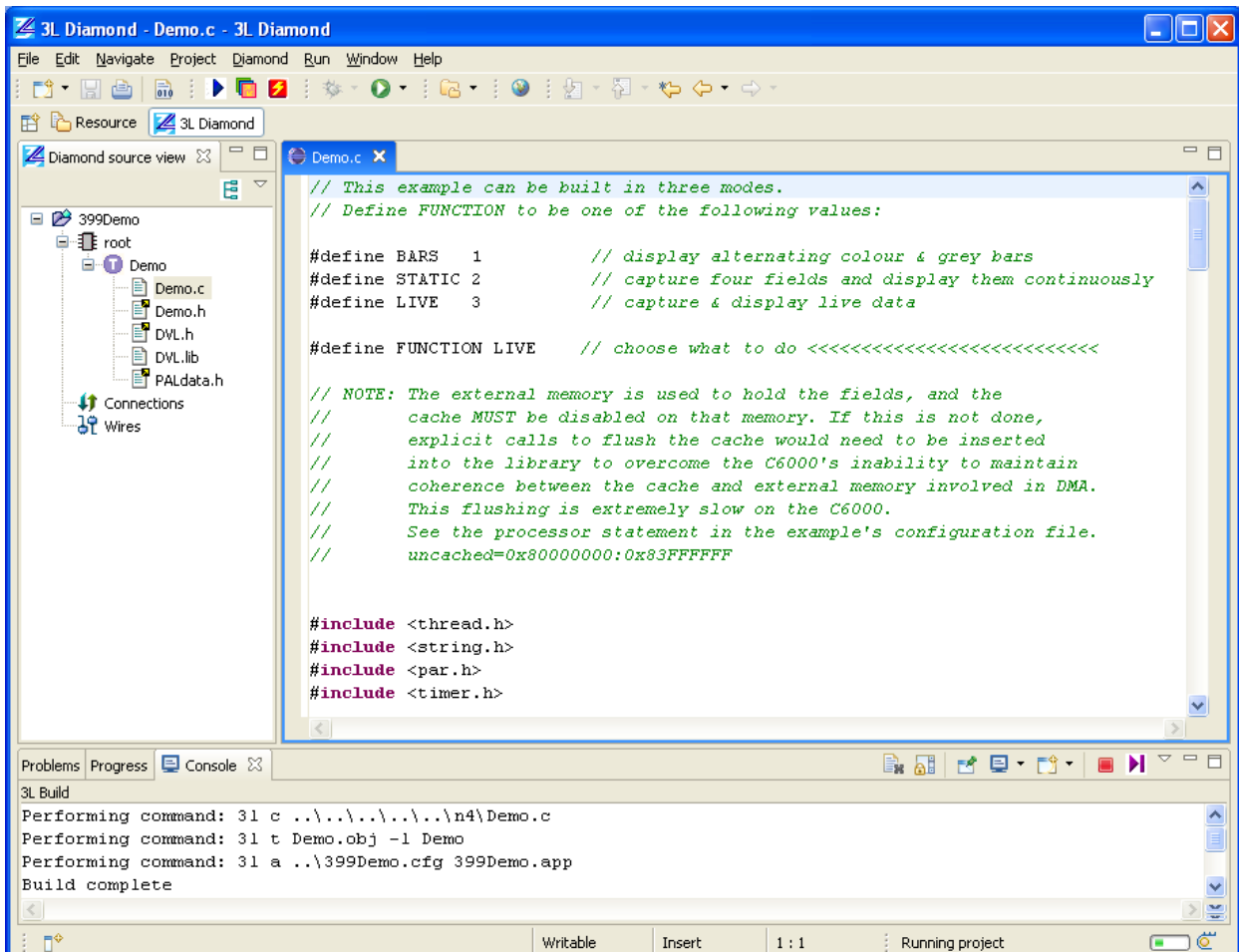
The sample code is provided as a zip file that should be unpacked to a folder of your own which will then contain two folders: **MAKE** and **IDE**.




MAKE

The **make** folder includes all the sources needed to build the sample application. You simply execute the makefile at the top level with a suitable “make” utility; **nmake** is a good example. It assumes that your default processor type is set to correspond with the variant of the SMT339 that you are using. This will create the application **339Demo.app** in the folder **output**. Make sure that your display and camera are connected and powered-up, then execute this application by double-clicking it.

IDE

Start the Diamond IDE by double-clicking the “Diamond IDE Beta” desktop icon. The IDE will start and will ask for a workspace. Browse to **IDE\SMT339_Workspace** and click OK. If the IDE does not ask for a workspace, you can select one by clicking **File** and then **Switch Workspace...**



Double-click “root” in the Diamond source view if you need to change the processor type. You can build the project by typing **Ctrl+B** or by selecting **Project** then **Build Project**. It is executed by clicking the button . The PC side can be stopped by clicking the red square, , at the bottom right. To stop the DSP from executing, reset the module using the button .

Functions

The following objects are available in the library, as defined by `DVL.h`.

`WRITE_TABLE`

The type of entries in write tables. All write tables must have pairs of entries of this type followed by a single terminator—an odd number of entries in total.

`WRITE_TABLE_TERMINATOR`

A value that must be used as the final entry in every write table.

```
int VP_Init_Transfer(VIDEO_DATA *D, int Pixels, int Lines);
```

Initialise the video processor. This must be done first.

D must be a reference to a pre-defined structure giving initialisation information for the video processor. Currently the only acceptable parameter is `&VIDEO_PAL`.

Pixels is the number of pixels on each line of the image. It is typically 720.

Lines is the number of lines in each field. It is typically 288.

The result of the function is 0.

```
int VP_Start_Write(VPDEV *D, unsigned int Lines, WRITE_TABLE *First);
```

Start the output thread. This must be done after `VP_Init_Transfer` and before any other operations.

D must be a reference to the video output port, `&VP1`.

Lines is the number of lines in each field. It is typically 288.

First is a reference to a write table giving the initial data to be displayed.

The result of the function is 0.

```
int VP_Read (VPDEV *D, unsigned int Lines, void *Y, void *Cb, void *Cr);
```

Read the given number of lines from the video input port **D** (`&VP0`).

The data are placed in the three memory areas identified by **Y**, **Cb**, and **Cr**.

The function returns 0 on successful completion of the read; it returns 1 if an overrun was detected.

```
void VP_Use(WRITE_TABLE *T);
```

Indicate that the output thread is to switch to displaying data from the given table, **T**, once it has completed displaying the final two fields of the current table.

```
void VP_ColourBars(void *Y, void *Cr, void *Cb,  
                  unsigned int Width, unsigned int Height);
```

Fill the three memory areas identified by **Y**, **Cb**, and **Cr** with a pattern of eight vertical colour bars with two diagonal lines.

```
void VP_GreyBars(void *Y, void *Cr, void *Cb,  
                unsigned int Width, unsigned int Height);
```

Fill the three memory areas identified by **Y**, **Cb**, and **Cr** with a pattern of eight vertical grey bars with two diagonal lines.