TEXAS INSTRUMENTS

TVP5020 NTSC/PAL Video Decoder

Programming for the VIP Host Interface

Application Report

March 1999

Mixed-Signal Products

SLAA054

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Copyright © 1999, Texas Instruments Incorporated

Contents

1 Introduction 1

2 Hardware Platform 2
2.1 VIP Master FPGA 2
2.1.1 FPGA Reset Procedure 2
2.1.2 TVP5020 Write Procedure 3
2.1.3 TVP5020 Read Procedure 3

3 Program Overview 5
3.1 Board-Specific and Microcontroller-Specific Code 6
3.2 Header File: REG652.H 7
3.3 Source Code Modules 9

4 Program Description 10
4.1 Source Code Module: Main 10
4.1.1 Inclusion of TVP5020 Microcode Files (Lines 11–14) 10
4.1.2 Function: Main() 10
4.1.3 Function: PowerUpInitialization() 12
4.1.4 Function: Patch TVP5020 Registers() 13
4.2 Header File: Main.H 13
4.3 Source File: MAIN.C 14
4.4 Source Code Module: I2C 17
4.4.1 Function: Initia_i2c() (Lines 28–41) 17
4.4.2 Function: start_i2c() 17
4.4.3 Function: i2c_isr() (Lines 77–266) 17
4.5 Header File: I2C.H 19
4.6 Source File: I2C.C 20
4.7 Source Code Module: Timer 26
4.7.1 Function: timer0_isr() (Lines 26–59) 26
4.7.2 Function: timer0_initialize() (Lines 61–94) 26
4.7.3 Function: ResetTickCount() (Lines 96–112) 26
4.7.4 Function: current_tick() (Lines 114–131) 26
4.7.5 Function: timer0_elapsed_count() (Lines 133–150) 26
4.7.6 Function: timer0_wait() (Lines 152–167) 26
4.8 Header File: Timer.H 27
4.9 Source File: TIMER.C 28
4.10 Source Code Module: VIP5020 31
4.10.1 Function: DecoderReset() (Lines 22–35) 31
4.10.2 Function: HandleDownload() (Lines 37–77) 32
4.10.3 Function: WriteTVP5020() (Lines 79–92) 32
4.10.4 Function: GetControlByte() (Lines 94–132) 32
4.10.5 Function: WriteTVP5020VIP() (Lines 134–170) 33
4.10.6 Function: ReadSwitch() (Lines 172–189) 33
4.10.7 Function: ReadTVP5020() (Lines 192–257) 33
4.10.8 Function: ReadTVP5020VIP1() (Lines 259–294) 33
4.10.9 Function: ReadTVP5020VIP2() (Lines 295–328) 33
4.11 Header File: VIP5020.H 34
4.12 Header File: VIP5020.C 35
4.13 Source Code Module: I2C6000 42
4.13.1 Function: read_tvp6000() (Lines 23–40) 42
4.13.2 Function: write_tvp6000() (Lines 41–50) 42
4.13.3 Function: LoadTVP6000() (Lines 52–75) 42

4.13.4 Function: PatchTVP6000() (Lines 77–97 . . . 42
4.14 Header File: I2C6000.H . . . 43
4.15 Source File: I2C6000.C . . . 44
4.16 TVP6000 Initialization Data for NTSC with CCIR601 Sampling . . . 46
4.17 TVP6000 Initialization Data for NTSC with Square Pixel Sampling . . . 47
4.18 TVP6000 Initialization Data for PAL with CCIR601 Sampling . . . 48
4.19 TVP6000 Initialization Data for PAL with Square Pixel Sampling . . . 49

List of Figures

1 VIP Emulator Prototype Block Diagram . . . 2
2 Help-About Dialog Box from uVision/51 for Windows . . . 5
3 Help-About Dialog Box for Signum Systems In-Circuit Emulator . . . 6
4 TVP5020 Microcode in Hex-ASCII Format . . . 10
5 TVP5020 Microcode After Conversion to Standard C Format . . . 11
6 TVP5020 Microcode After Modification . . . 11

List of Tables

1 DIP Switch Settings for VIP Emulator Prototype . . . 5
2 Microcontroller I/O Port Utilization for VIP Emulator Prototype . . . 6
3 Source Code Module Relationships . . . 9
4 TVP5020 Register Patches . . . 10
5 I2C Controller: Master Transmitter States . . . 18
6 I2C Controller: Master Receiver States . . . 18

TVP5020 NTSC/PAL Video Decoder Programming for the VIP Host Interface

Michael A. Tadyshak

ABSTRACT

This application report provides a complete working example of a C-language program to initialize the TVP5020 NTSC/PAL video decoder using the VIP bus interface. The covered topics include the hardware platform, a detailed description of the source-code modules, and microcontroller-specific items.

1 Introduction

The TVP5020 NTSC/PAL Video Decoder enables a wide range of applications by providing support for each of the following host interfaces:

- I^2C (Inter-Integrated Circuit) bus © 1995 Philips Semiconductors
- VIP (Video Interface Port) © 1994, 1996, 1997 Video Electronic Standards Association
- VMI (Video Module Interface) © 1997 Cirrus Logic

Software development time can be reduced by utilizing this Example Initialization Program. The TVP5020-specific source code modules can be used as a reference for software development for the user’s own hardware platform.

2 Hardware Platform

This program was tested on the VIP Emulator Prototype. A block diagram of this board is shown in Figure 1. The program is executed by the Philips P80C652 microcontroller. This device is a derivative of the Intel 80C51 microcontroller and is second-sourced by Philips. The P80C652 includes an on-chip I2C bus controller. The program is stored in a 64 KB 27C512 EPROM. The board also has 32 kB of static RAM available for program use. The VIP master field programmable gate array (FPGA) communicates with the P80C652 via the I2C interface, and communicates with the TVP5020 as a VIP master device.

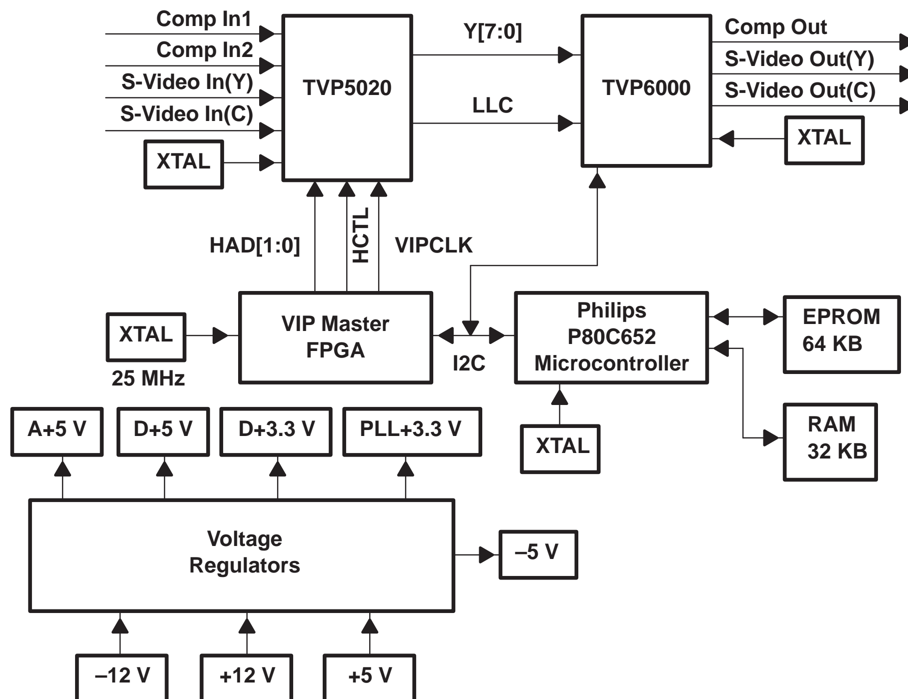

Figure 1. VIP Emulator Prototype Block Diagram

2.1 VIP Master FPGA

The VIP Master FPGA is used to exercise the VIP interface of the TVP5020. The FPGA has a 256-byte internal buffer. This program formats the data for the VIP bus and writes the data to the FPGA via I2C. The FPGA then serializes the data and generates the VIP signalling required to transmit the data to the TVP5020. This is not a practical scheme to use in a real application since the fast VIP bus is placed in series with the slow I2C bus. However, it does allow the VIP interface to run at full speed between I2C transfers.

2.1.1 FPGA Reset Procedure

The microcontroller writes the reset command to the FPGA via I2C. The I2C device ID is E0h and the data is FFh.

2.1.2 TVP5020 Write Procedure

When downloading to the TVP5020 program memory, up to 238 data bytes can be used in a VIP transfer. This program imposes the 238-byte limit solely to prevent overflow of the 256-byte buffer in the VIP Master FPGA. Multiple data blocks are downloaded using the download microcode address with each block. There must not be any accesses to TVP5020 addresses other than the download microcode address until the entire microcode download has completed.

When writing to TVP5020 registers, exactly one (1) data byte must be used. The general procedure for writing to the TVP5020 via the VIP Master FPGA is listed below. The delay loop referred to below does not explicitly appear in the source code. The relatively slow nature of the I2C bus provides the required delay.

- The microcontroller forms a data packet containing:
  - Control byte
  - Address byte
  - Data byte 1
  - Data byte 2
  - .
  - .
  - .
  - Data byte N

  Where: 1 <= N <= 238
- The microcontroller writes formatted data packet to FPGA via the I2C.
- The microcontroller starts a delay loop to allow VIP transfers to complete.
- The FPGA detects I2C stop condition.
- The FPGA writes all data in a burst transfer via VIP bus.
- The microcontroller delay loop completes.

2.1.3 TVP5020 Read Procedure

Up to 238 data bytes can be used in a VIP transfer when uploading the TVP5020 program memory. This program imposes the 238-byte limit solely to prevent overflow of the 256-byte buffer in the VIP Master FPGA. Multiple data blocks are uploaded using the *upload microcode* address with each block. There must not be any access to TVP5020 addresses, other than the upload microcode address, until the entire microcode upload has completed.

When reading from TVP5020 registers, exactly one (1) data byte must be read. The general procedure for reading from the TVP5020 via the VIP Master FPGA is listed below. The delay loop mentioned below does not explicitly appear in the source code. The relatively slow nature of the I2C bus provides the required delay.

- The microcontroller forms a data packet containing:
  - Control byte
  - Address byte
  - Dummy data byte 1 (00h)

  - Dummy data byte 2 (00h)
  - .
  - .
  - .
  - Dummy data byte N (00h)

  Where: 1 <= N <= 238

- The microcontroller writes formatted data packet to FPGA via the I2C.
- The microcontroller starts a delay loop to allow VIP transfers to complete.
- The FPGA detects an I2C stop condition.
- The FPGA performs all requested read operations via VIP bus.
- The FPGA overwrites dummy data byte locations in buffer with read data.
- The microcontroller delay loop completes.
- The microcontroller reads the data packet from the FPGA buffer via the I2C.
- The microcontroller extracts the read data from the data packet.

3 Program Overview

This program is a complete working example of a C-language program to initialize the TVP5020 NTSC/PAL Video Decoder using the VIP bus interface. TVP5020 microcode for four video modes is contained in the program to enable testing of NTSC or PAL video standards using CCIR601 or square pixel sampling rates. The video mode is selected by setting the DIP switches as shown in Table 1.

The program was compiled and linked using *uVision/51* for *Windows,* a software package for compiling code for 80C51-type microcontrollers from Keil Software, Inc. Information about Keil Software can be found on the Internet at *www.keil.com*. See the *Help-About* dialog box for this software package (see Figure 2).

The USP-51 from Signum Systems is an in-circuit emulator for debugging P80C652 code. Information about Signum Systems can be found on the Internet at *www.signum.com*. See the *Help-About* dialog box for the USP-51 emulator software (see Figure 3).

Table 1. DIP Switch Settings for VIP Emulator Prototype

VIDEO STANDARD	SAMPLING RATE	INDIVIDUAL SWITCHES	
		S0	SW1
NTSC	CCIR601	ON	ON
NTSC	Square pixel	OFF	ON
PAL	CCIR601	ON	OFF
PAL	Square pixel	OFF	OFF

Figure 2. Help-About Dialog Box from *uVision/51 for Windows*

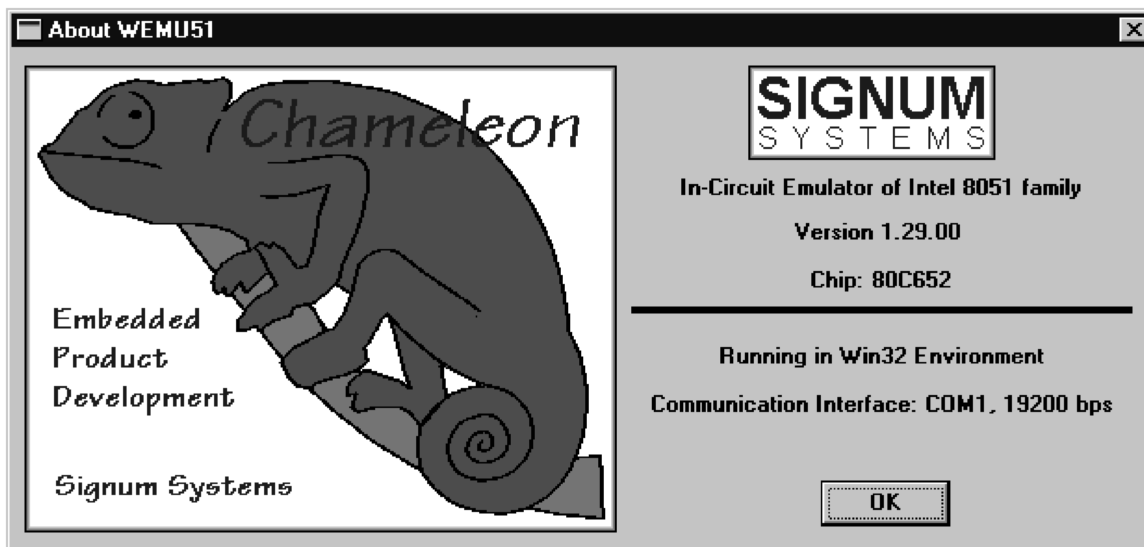

Figure 3. Help-About Dialog Box for Signum Systems In-Circuit Emulator

3.1 Board-Specific and Microcontroller-Specific Code

Most of the source code is in standard C-language. The main exception is the use of macros to access the special-function registers or special-function I/O bits of the P80C652. The complete set of macros defined for the P80C652 is contained in the file REG652.H, which is listed in pages 7 and 8. Several user-definable I/O pins are assigned for special use by the emulator board, as shown in Table 2. SW1 and S0 are used to read the corresponding DIP switches for selecting the video mode, and T6RESET provides a software means to reset the TVP6000 video encoder. SW2 and SW3 configure the onboard data path to route YUV data directly from TVP5020 to TVP6000.

Many of the P80C652-specific special-function registers are used to control the on-chip general-purpose timer and the I2C bus interface. These are localized to the TIMER and I2C source code modules.

The P80C652 provides 128 bytes of on-chip RAM, and direct support for 64 kB of external data memory (xdata space) and 64 kB of read-only memory (code space). The keywords *xdata* and *code* are sometimes needed in variable declarations to specify the type of memory storage. For example, the arrays of *unsigned char* which hold the TVP5020 microcode modules are declared with the *code* keyword so that they will be stored in EPROM instead of RAM.

Table 2. Microcontroller I/O Port Utilization for VIP Emulator Prototype

PIN NAME	SIGNAL NAME	DIRECTION	FUNCTION	DEFINITION
P1.0	S0	Input	DIP switch 2	User-defined
P1.1	SW1	Input	DIP switch 1	User-defined
P1.2	SW2	Output	Data path control 0	User-defined
P1.3	SW3	Output	Data path control 1	User-defined
P1.4		Input	Not Used	User-defined
P1.5	T6RESET	Output	TVP6000 reset	User-defined
P1.6	SCL	Output	I2C clock	Dedicated I/O
P1.7	SDA	I/O	I2C data	Dedicated I/O

3.2 Header File: REG652.H

```
// REG652.H
// Header file for Philips P80C652 Microcontroller.

/*  BYTE Registers */
sfr P0     = 0x80;
sfr P1     = 0x90;
sfr P2     = 0xA0;
sfr P3     = 0xB0;

sfr PSW    = 0xD0;
sfr ACC    = 0xE0;
sfr B      = 0xF0;
sfr SP     = 0x81;
sfr DPL    = 0x82;
sfr DPH    = 0x83;
sfr PCON   = 0x87;
sfr TCON   = 0x88;
sfr TMOD   = 0x89;
sfr TL0    = 0x8A;
sfr TL1    = 0x8B;
sfr TH0    = 0x8C;
sfr TH1    = 0x8D;
sfr IE     = 0xA8;
sfr IP     = 0xB8;
sfr S0CON  = 0x98; /* UART control */
sfr S0BUF  = 0x99; /* UART data buffer */

sfr S1CON  = 0xD8; /* I2C control register */
sfr S1STA  = 0xD9; /* I2C status register */
sfr S1DAT  = 0xDA; /* I2C data register */
sfr S1ADR  = 0xDB; /* I2C address register */

/*  BIT Register  */
/*  PSW  */
sbit CY    = 0xD7;
sbit AC    = 0xD6;
sbit F0    = 0xD5;
sbit RS1   = 0xD4;
sbit RS0   = 0xD3;
sbit OV    = 0xD2;
sbit P     = 0xD0;

/*  TCON  */
sbit TF1   = 0x8F;
sbit TR1   = 0x8E;
sbit TF0   = 0x8D;
sbit TR0   = 0x8C;
sbit IE1   = 0x8B;
sbit IT1   = 0x8A;
sbit IE0   = 0x89;
sbit IT0   = 0x88;
```

3.2 Header File: REG652.H (Continued)

```
/*  IE  */
sbit EA       = 0xAF;
sbit ES1      = 0xAD; /* I2C interrupt enable */
sbit ES0      = 0xAC; /* UART interrupt enable */
sbit ET1      = 0xAB;
sbit EX1      = 0xAA;
sbit ET0      = 0xA9;
sbit EX0      = 0xA8;

/*  IP */
sbit PS1      = 0xBD;
sbit PS0      = 0xBC;
sbit PT1      = 0xBB;
sbit PX1      = 0xBA;
sbit PT0      = 0xB9;
sbit PX0      = 0xB8;

//  P1
sbit SDA      = 0x97;
sbit SCL      = 0x96;
sbit T6RESET  = 0x95;
sbit S1       = 0x94;
sbit SW3      = 0x93;
sbit SW2      = 0x92;
sbit SW1      = 0x91;
sbit S0       = 0x90;

//  P3
sbit RD       = 0xB7;
sbit WR       = 0xB6;
sbit LED2     = 0xB5;
sbit LED1     = 0xB4;
sbit FLASHJMP = 0xB3;
sbit INTREQ   = 0xB2;
sbit TXD      = 0xB1;
sbit RXD      = 0xB0;

/*  S0CON  */
sbit SM0      = 0x9F;
sbit SM1      = 0x9E;
sbit SM2      = 0x9D;
sbit REN      = 0x9C;
sbit TB8      = 0x9B;
sbit RB8      = 0x9A;
sbit TI       = 0x99;
sbit RI       = 0x98;

/*  S1CON   */
sbit CR0      = 0xD8;
sbit CR1      = 0xD9;
sbit AA       = 0xDA;
sbit SI       = 0xDB;
sbit STO      = 0xDC;
sbit STA      = 0xDD;
sbit ENS1     = 0xDE;
sbit CR2      = 0xDF;
```

3.3 Source Code Modules

Table 3 summarizes the relationships between the various source code modules. Each source code module is contained in one *.C* source file and has an associated *.H* header file. The timer and I2C modules are described as microcontroller-specific. In order to port these functions to another hardware environment, equivalent functions, written for the specific processor, would need to be supplied or created. The main, VIP5020, and I2C6000 modules could be used virtually unchanged. In the new environment, the TVP6000 software reset (T6RESET) would need to be implemented, as well as the reading of the DIP-switch lines SW1 and S0, if required.

Table 3. Source Code Module Relationships

SOURCE CODE MODULE	DESCRIPTION	MICROCONTROLLER-SPECIFIC?	VIP EMULATOR PROTOTYPE-SPECIFIC?	CALLS FUNCTIONS IN THESE MODULES
Main	Main program	No	Yes (uses T6RESET)	I2C5020, I2C6000, timer
Timer	General-purpose timer routines and ISR	Yes	No	None
I2C	I2C bus routines and ISR	Yes	No	Timer
VIP5020	TVP5020-specific VIP routines	No	Yes (uses SW1 & S0)	I2C
I2C6000	TVP6000-specific I2C routines	No	No	I2C, timer

4 Program Description

4.1 Source Code Module: Main

4.1.1 Inclusion of TVP5020 Microcode Files (Lines 11–14)

Header files containing the TVP5020 microcode are included. These provide support for NTSC and PAL video standards with CCIR601 or square pixel sampling. Each header file declares an array of type *unsigned char*. The first byte of each array is the subaddress for writing to the TVP5020 program memory (0x7E). The TVP5020 microcode is supplied in a five-character Hex-ASCII format (see Figure 4). Conversion to a standard C-language header file can be done with a utility called *HexConv*. The output of *HexConv* is shown in Figure 5. If necessary, the *#define* constant for the code size, which includes the sub-address byte – 0x7E as well as the array name, may be given a unique name. If the target processor is an 80C51 derivative, the keyword *code* must be inserted. Also, adding a comment identifying the microcode type and version is very helpful. The resulting microcode file is shown in Figure 6.

4.1.2 Function: Main()

4.1.2.1 Declaration of TVP5020 Register Patch Data (Lines 16–24)

After the microcode is downloaded and the TVP5020 CPU is restarted, the registers are initialized with their default values (as defined by the TVP5020 data manual) by the internal CPU. Some registers must be patched with a different value in order for the TVP5020 to function properly on the VIP emulator prototype. The array *g_pTVP5020Patch* contains the address and data for three registers that must be modified. Table 4 describes these register changes.

Table 4. TVP5020 Register Patches

TVP5020 ADDRESS (Bits 7–0)	DEFAULT DATA	PATCHED DATA	COMMENT
03h	00h	19h	Enable HSYN, VSYN, AVID, SCLK, PCLK and YUV outputs
07h	00h	10h	Bypass luminance processing during vertical blank
0Dh	00h	0Fh	Select 8–bit ITU–R BT.656 interface, video port VP1

```
80000
00000
303FC
.
.
.
C3F80
```

Figure 4. TVP5020 Microcode in Hex-ASCII Format

```
#define TVP5020_CODE_SIZE 0x27b8

unsigned char TVP5020_CODE[] =
{
0x7E,
0x08,
0x00,
0x00,
0x00,
0x00,
0x00,
0x03,
0x03,
0xFC,
.
.
.
0x0C,
0x3F,
0xB0
};
```

Figure 5. TVP5020 Microcode After Conversion to Standard C Format

```
#define TVP5020_N601_CODE_SIZE 0x27b8

// TVP5020 NTSC CCIR601 Version: 63

unsigned char code T520_N601[] =
{
0x7E,
0x08,
0x00,
0x00,
0x00,
0x00,
0x00,
0x03,
0x03,
0xFC,
.
.
.
0x0C,
0x3F,
0xB0
};
```

Figure 6. TVP5020 Microcode After Modification

4.1.2.2 Initialization (Lines 37–45)

The function call *timer0_initialize()* initializes the on-chip general-purpose timer to generate a timer-tick interrupt every 2 ms. Next, timer0_wait() is called to produce a 100 ms delay to insure stabilization after reset.

The call to *DecoderReset()* performs two board-specific tasks. First, it initializes the P80C652's user-defined I/O port P1. This programs the SW2 and SW3 I/O bits and configures the board to route YUV data directly from TVP5020 to TVP6000. Second, it resets the FPGA.

4.1.2.3 Video Mode Selection (Lines 47–72)

The current state of the DIP switches is read. The two lsbs are used to select the video mode as shown in Table 1. The global variables *g_nROMCodeSize* and *g_pROMCode* are initialized with the size of the selected microcode file (in bytes and including the address byte) and with the starting address of the selected microcode data array.

4.1.2.4 Reset Timer-Tick (Line 75)

The call to *ResetTickCount()* resets the internal count of timer-tick interrupts (which occur every 2 ms). The timer-tick count can run up to about 128 seconds before rolling over. In a more complex program with multiple uses for the general-purpose timer, the timer-tick count should be reset only in the outermost loop.

4.1.2.5 Power-Up Initialization (Line 78)

The call to *PowerUpInitialization()* performs the tasks of downloading the TVP5020 microcode, restarting the TVP5020 internal CPU, patching TVP5020 registers, and initializing the TVP6000 video encoder. One parameter is passed to *PowerUpInitialization()* to indicate the selected video mode. Upon return from *PowerUpInitialization()*, the program spins in an endless loop.

4.1.3 Function: PowerUpInitialization()

4.1.3.1 Microcode Download (Line 94)

The call to *HandleDownload()* calls the specific routine which will download the microcode to the video decoder. The code size and code pointer variables are passed as parameters. *HandleDownload()* routines have been written for the I2C, VIP and VMI interfaces. The source code module VIP5020.C contains the version of *HandleDownload()* for the VIP bus

4.1.3.2 Restart Microprocessor (Lines 96–102)

After the microcode download completes, the internal microprocessor is restarted. This is done by writing a 00h byte (the data can be any value) to the restart address (7Fh). The function *WriteTVP5020()* is used whenever it is required to write to a TVP5020 register. The parameters passed to *WriteTVP5020()* are a byte count and a pointer to the storage location or the address and data. A 10ms wait is inserted after the restart command to enable the internal microprocessor to complete its initialization code.

4.1.3.3 Patch TVP5020 Registers (Lines 104–105

The call to *PatchTVP5020Registers()* implements the register modifications described in Section 4.1.2.1 and Table 4.

4.1.3.4 Reset the TVP6000 Video Encoder (Lines 107–110)

The user-definable I/O pin P1.5 is used as a software-controlled reset for the TVP6000. T6RESET is a macro which allows control of pin P1.5. The TVP6000 reset input is held active for 100ms after the TVP5020 is initialized. This is needed, since the TVP6000 is not guaranteed to have received a clock from the TVP5020 during power-up reset.

4.1.3.5 Initialize the TVP6000 Video Encoder (Lines 112–113)

The call to *LoadTVP6000()* initializes the video encoder registers. One parameter is passed to indicate the selected video mode. The register data is shown in the listing in Section 4.16 through 4.19, and is located in the header file *DATA6000.H*.

4.1.4 Function: Patch TVP5020 Registers()

4.1.4.1 Loading the Registers (Lines 120–123)

A *for* loop is used to patch the TVP5020 registers using the data in the *g_pTVP5020Patch* array. This array is a global variable and was initialized in lines 17–24. The constant *TVP5020_PATCH_SIZE holds the number of bytes in the array and must be changed if the number of register writes is changed.*

4.2 Header File: Main.H

```
// Main.H
//
// Header file for main program to initialize the TVP5020 Video Decoder
//
#define FALSE                   0
#define TRUE                    !FALSE

#define TVP5020_RESTART_SUB_ADDR        0x7F

void          main (void);
void          DecoderReset( void );
unsigned char ReadSwitch( void );
void          PowerUpInitialization( int nSwitch );
void          HandleDownload( unsigned nCount, unsigned char* pInBuf );
unsigned      WriteTVP5020(int nLength, unsigned char *pBuf );
void          PatchTVP5020Registers(void);
```

4.3 Source File: MAIN.C

```
DOS C51 COMPILER V5.10, COMPILATION OF MODULE MAIN
OBJECT MODULE PLACED IN MAIN.OBJ
COMPILER INVOKED BY: C:\C51\BIN\C51.EXE MAIN.C DB OE OR LARGE

stmt level    source

   1          // Main.C
   2          //
   3          // Main program to initialize the TVP5020 Video Decoder
   4          //
   5          #include ”Main.h”
   6          #include ”Timer.h”
   7          #include ”Reg652.h”
   8          #include ”I2C6000.H”
   9
  10          //TVP5020 microcode files
  11          #include ”5020NSQP.H”
  12          #include ”5020N601.H”
  13          #include ”5020PSQP.H”
  14          #include ”5020P601.H”
  15
  16          // Registers to modify after TVP5020 CPU startup
  17          #define      TVP5020_PATCH_SIZE         6
  18          unsigned char code g_pTVP5020Patch[] =
  19          {
  20             // sub-address, data
  21             0x03, 0x19,
  22             0x07, 0x10,
  23             0x0D, 0x0F
  24          };
  25
  26          // Size of TVP5020 microcode file (defined in 5020xxxx.H)
  27          unsigned       g_nROMCodeSize  = TVP5020_N601_CODE_SIZE;
  28
  29          // Pointer to the TVP5020 microcode
  30          unsigned char* g_pROMCode      = T520_N601;
  31
```

4.3 Source File: MAIN.C (Continued)

```
32          void main(void)
33          {
34    1         // DIP Switch value
35    1         unsigned char nSwitch = 0;
36    1
37    1         // Initialize general purpose timer
38    1         timer0_initialize();
39    1
40    1         /* Wait 100ms - for stabilization after reset */
41    1         timer0_wait(ONE_HUNDRED_MS);
42    1
43    1         // For VMI bus, this configures the TVP5020 interrupt output
                //(INTREQ)
44    1         // For VIP bus, this sends a reset code to the VIP emulation FPGA
45    1         DecoderReset();
46    1
47    1         // Two LSBs of switch select the video mode
48    1         nSwitch = ReadSwitch() & 3;
49    1
50    1         // Point to the microcode selected by the DIP switch
51    1         switch( nSwitch )
52    1         {
53    2                 case 0:
54    2                     g_pROMCode      = T520_N601;
55    2                     g_nROMCodeSize  = TVP5020_N601_CODE_SIZE;
56    2                 break;
57    2
58    2                 case 1:
59    2                     g_pROMCode      = T520_NSQP;
60    2                     g_nROMCodeSize  = TVP5020_NSQP_CODE_SIZE;
61    2                 break;
62    2
63    2                 case 2:
64    2                     g_pROMCode      = T520_P601;
65    2                     g_nROMCodeSize  = TVP5020_P601_CODE_SIZE;
66    2                 break;
67    2
68    2                 case 3:
69    2                     g_pROMCode      = T520_PSQP;
70    2                     g_nROMCodeSize  = TVP5020_PSQP_CODE_SIZE;
71    2                 break;
72    2         }
73    1
74    1         // Reset timer-tick to avoid rollover
75    1         ResetTickCount();
76    1
77    1         // Initialize the video mode
78    1         PowerUpInitialization( nSwitch );
79    1
80    1         // After video is initialized, do nothing
81    1         while(1)
82    1         {
83    2         ;
84    2         }
85    1
86    1         return;
87    1     }
```

4.3 Source File: MAIN.C (Continued)

```
  88
  89           void PowerUpInitialization( int nSwitch )
  90           {
  91    1          unsigned char buf[2];
  92    1
  93    1          /* Download video decoder microcode */
  94    1          HandleDownload( g_nROMCodeSize, g_pROMCode );
  95    1
  96    1          //Restart microprocessor
  97    1          buf[0] = TVP5020_RESTART_SUB_ADDR;
  98    1          buf[1] = 0;
  99    1          WriteTVP5020( 2, buf );
 100    1
 101    1          // Wait 10ms for TVP5020 CPU to start-up
 102    1          timer0_wait(TEN_MS);
 103    1
 104    1          // Modify registers from the default state as required
 105    1          PatchTVP5020Registers();
 106    1
 107    1          // Reset the TVP6000
 108    1          T6RESET = 0;
 109    1          timer0_wait(ONE_HUNDRED_MS);
 110    1          T6RESET = 1;
 111    1
 112    1          // Initialize TVP6000
 113    1          LoadTVP6000( nSwitch );
 114    1      }
 115
 116           void PatchTVP5020Registers (void)
 117           {
 118    1          int i = 0;
 119    1
 120    1          for( i=0; i<TVP5020_PATCH_SIZE; i+=2 )
 121    1          {
 122    2              WriteTVP5020( 2, g_pTVP5020Patch+i );
 123    2          }
 124    1
 125    1          return;
 126    1      }
 127

C51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

4.4 Source Code Module: I2C

This source code module is included for communication with the TVP6000 NTSC/PAL video encoder and the VIP master FPGA.

4.4.1 Function: Initia_i2c() (Lines 28–41)

This function initializes the I2C bus signals (SCL and SDA) to a high level. The internal P80C652 I2C interrupt is enabled and given low (normal) priority. The P80C652 on-chip I2C controller is initialized to be the I2C bus master with a baud rate of 92.16 kHz. This frequency is the P80C652 oscillator frequency (11.0592 MHz) divided by 120.

4.4.2 Function: start_i2c()

This function is called to perform a transaction on the I2C bus.

4.4.2.1 Initialize Variables for the ISR (Lines 54–64)

The current timer-tick count is saved to be used later to determine if a timeout condition has occurred. The macro EA is used to globally enable or disable all interrupts. The global variables used by the I2C interrupt service routine (ISR) are:

- **b_counter**—Byte counter. Initialized to 00h, counts up to the terminal count value.
- **num_b**—Number of bytes. Holds the terminal count value.
- **slave_rw**—Slave device ID and read/write bit.
- **i2cbuf**—Pointer to caller's data buffer.

4.4.2.2 Start the I2C Transaction (Lines 63–64)

The macro STA is used to set the start bit in the I2C control register. Then, the global interrupt control bit EA enables the hardware interrupts.

4.4.2.3 Wait for the I2C bus Transaction to Complete (Lines 66–73)

The program now remains in a loop waiting for either the transfer of all bytes, or the occurrence of an error condition. Meanwhile, I2C bus interrupts are occurring and the I2C ISR is controlling the data transfer. The timer-tick count is checked for a timeout condition by comparing the elapsed time with *g_nI2Ctimeout*. The value in *g_nI2Ctimeout* is in units of timer-ticks. The timer-tick is programmed to occur once every 2ms.

4.4.3 Function: i2c_isr() (Lines 77–266)

This is the interrupt service routine (ISR) for the I2C bus interface. The I2C controller is embedded in the P80C652. A simple register interface provides access to address, data, control, and status registers. Each time an I2C interrupt occurs, the status register (S1STA) is read to obtain the current state code from the I2C controller. The state code is used to branch to the appropriate code segment to handle the interrupt. The I2C global variables are updated and data is transferred to/from the user's data buffer. The states for the master transmitter and for the master receiver are described in Tables 5 and 6. The last step of handling the interrupt is writing one of the following four codes back to the I2C control register (S1CON) to request a specific action:

- I2C_RLS_STA – Release bus and generate a start condition
- I2C_RLS_ACK – Release bus and acknowledge the data transfer
- I2C_RLS_NACK – Release bus and do not acknowledge the data transfer
- I2C_STOP – Generate a stop condition

Table 5. I2C Controller: Master Transmitter States

I2C CONTROLLER STATE	DEFINITION	NEXT ACTION TAKEN BY I2C ISR
08h	Start condition has been transmitted	Send slave address + r/w bit
10h	Repeat start condition has been transmitted	Send slave address + r/w bit
18h	Slave address has been sent and ACK was received	Transmit first data byte
20h	Slave address has been sent and NOT ACK was received	Transmit first data byte. Flag I2C NOT ACK error.
28h	Data has been transmitted and ACK has been received	Transmit next data byte. If all data has been transmitted, issue a stop condition.
30h	Data has been transmitted and NOT ACK has been received	Transmit next data byte. If all data has been transmitted, stop the bus. Flag I2C NOT ACK error.
38h	Bus arbitration lost	Flag I2C bus arbitration lost error. Issue another start condition.

Table 6. I2C Controller: Master Receiver States

I2C CONTROLLER STATE	DEFINITION	NEXT ACTION TAKEN BY I2C ISR
40h	Slave address has been sent and ACK was received	If transaction involves only one data byte, signal the controller to NOT ACK the next data byte received. Otherwise, signal the controller to acknowledge the next data byte received.
48h	Slave address has been sent and NOT ACK was received	If transaction involves only one data byte, signal the controller to NOT ACK the next data byte received. Otherwise, signal the controller to acknowledge the next data byte received. Flag I2C NOT ACK error.
50h	Data has been received and ACK has been transmitted	If this is previous to the NEXT–TO–LAST data byte, signal the controller to acknowledge the next data byte received. If this is the next-to-last data byte, signal the controller to NOT ACK the next data byte received.
58h	Data has been received and NOT ACK has been transmitted	If this is previous to the LAST data byte, signal the controller to acknowledge the next data byte received and flag an I2C NOT ACK error. If this is the last data byte, then issue a stop condition.
F8h	No relevant state information is available	No action required
00h	I2C bus error due to detection of an illegal start or stop condition, or I2C controller detected entry into an illegal state	Flag an I2C bus error
Other	I2C controller reported a state which is not supported by the interrupt service routine	Flag an I2C unsupported state error

4.5 Header File: I2C.H

```
// I2C.H
//
// Header file for I2C bus routines
//
#define FALSE                    0
#define TRUE                     !FALSE

#define ERR_I2C_NOTACK        0x01
#define ERR_I2C_ARBILOST   0x02
#define ERR_I2C_GERROR     0x04
#define ERR_I2C_TIMEOUT    0x08
#define ERR_I2C_BUSERROR   0x10
#define ERR_I2C_DEVID      0x20

void     initia_i2c(void);
unsigned start_i2c(unsigned char i2c_addrs, int buf_length,
                   unsigned char bufaddrs);
```

4.6 Source File: I2C.C

```
DOS C51 COMPILER V5.10, COMPILATION OF MODULE I2C
OBJECT MODULE PLACED IN I2C.OBJ
COMPILER INVOKED BY: C:\C51\BIN\C51.EXE I2C.C DB OE OR LARGE

stmt level    source

   1          //
   2          // I2C.C
   3          //
   4          // Routines for I2C bus
   5          //
   6          #include ”I2C.h”
   7          #include ”Timer.h”
   8          #include ”reg652.h”
   9
  10          #define FALSE             0
  11          #define TRUE         !FALSE
  12          #define I2C_STOP       0xD5  /* generated a STOP condition on I2C,
                                            and 100kbps */
  13          #define I2C_RLS_ACK    0xC5  /* Release bus and generate a ACK */
  14          #define I2C_RLS_NACK   0xC1  /* Release bus and generate a NOT ACK */
  15          #define I2C_RLS_STA    0xE5  /* Release bus and generate START */
  16
  17          // I2C Timeout
  18          unsigned       g_nI2CTimeout   = TEN_SECONDS;
  19
  20
  21          unsigned xdata error_i2c = 0;       /* I2C Errors */
  22          unsigned xdata b_counter = 0;       /* length of i2c send buffer */
  23          unsigned xdata num_b     = 0;       /* number of bytes that will be
                                                   sent/read */
  24          unsigned xdata num_b_minus_1 = 0;   /* number of bytes that will be
  25                                               sent/read - 1 */
  25          unsigned char xdata slave_rw = 0;   /* slave address plus
                                                   read/write direction */
  26          static unsigned char *i2cbuf = (unsigned char*)0; /* pointer to I2C
  27                                                          send/receiving buffer */
  27
  28          /*
  29          ---------------------------------------------------------------------
  30          This function will initialize the I2C interface
  31          ---------------------------------------------------------------------
  32          */
  33
  34          void initia_i2c(void)
  35          {
  36   1         SDA = 1; /* set data pin as high level */
  37   1         SCL = 1; /* set clock pin as high level */
  38   1         ES1 = 1; /* enable I2C interrupt */
  39   1         PS1 = 0; /* set I2C interrupt PRIORITY level as LOW */
  40   1         S1CON = I2C_RLS_ACK; /* set 80C652 as a master only,
                                          bit rate = 92.16k */
  41   1        }
```

4.6 Source File: I2C.C (Continued)

```
42
43          /*
44          ---------------------------------------------------------------------
45          This function transfers one block of data in or out
46          ---------------------------------------------------------------------
47          */
48
            unsigned start_i2c( unsigned char i2c_addrs, int buf_length,
                                unsigned char *bufaddrs )
50          {
51   1         unsigned xdata start_point;
52   1         unsigned test = 0u;
53   1
54   1         // Set a reference time
55   1         start_point = current_tick ();
56   1
57   1         EA = 0;          // Disable i2c interrupt
58   1         b_counter = 0;
59   1         num_b = (unsigned)buf_length;
60   1         num_b_minus_1 = num_b - 1;
61   1         slave_rw = i2c_addrs;
62   1         i2cbuf = bufaddrs;  // initialized the buffer point
63   1         STA = 1;           // set STA bit of S1CON, start I2C
64   1         EA = 1;
65   1
66   1         /* wait until all data in buffer have been sent out */
67   1         while ( (b_counter < num_b) && ( error_i2c == 0u) )
68   1         {
69   2             if (timer0_elapsed_count(start_point) > g_nI2Ctimeout)
70   2             {
71   3             error_i2c |= ERR_I2C_TIMEOUT;
72   3             }
73   2         }
74   1         return( b_counter );
75   1      }
76
77          /*
78          ---------------------------------------------------------------------
79           I2C interrupt service routine
80           interrupt number=5, address=0x002Bh, using register bank2
81          ---------------------------------------------------------------------
82          */
83
84          static void i2c_isr (void) interrupt 5 using 2
85          {
86   1
87   1      unsigned char i2cst;
88   1      unsigned char nDummy = 0;
89   1      i2cst = S1STA;
90   1
```

4.6 Source File: I2C.C (Continued)

```
  91   1         switch (i2cst)
  92   1         {
  93   2            /*-------------------------------------------------*/
  94   2            /* following section will be MASTER transmit mode */
  95   2            /*-------------------------------------------------*/
  96   2
  97   2            /* a START condition has been sent */
  98   2            /* will send out slave address + r/w bit */
  99   2            case 0x08:
 100   2                S1DAT = slave_rw;
 101   2                S1CON = I2C_RLS_ACK;
 102   2            break;
 103   2
 104   2            /* a repeat START has been transmitted */
 105   2            /* will load SLA+R/W, and return ACK */
 106   2            case 0x10:
 107   2            S1DAT = slave_rw;
 108   2            S1CON = I2C_RLS_ACK;
 109   2            break;
 110   2
 111   2            /* slave address has been send and ACK received */
 112   2            /* will send out 1st byte of data */
 113   2            case 0x18:
 114   2            S1DAT = *i2cbuf; /* load a byte to data register */
 115   2            S1CON = I2C_RLS_ACK;
 116   2            break;
 117   2
 118   2                /* NOT ACK received, will send out 1st byte of data
                             anyway */
 119   2            case 0x20:
 120   2            S1DAT = *i2cbuf; /* load a byte to data register */
 121   2            S1CON = I2C_RLS_ACK;
 122   2            error_i2c |= ERR_I2C_NOTACK;
 123   2            break;
 124   2
 125   2            /* continue sending data */
 126   2            /* 1st byte of data has been sent and ACK received */
 127   2            /* If all the data were sent, then transmit a STOP */
 128   2            /* else continue to transmit next byte */
 129   2            case 0x28:
 130   2               b_counter++;
 131   2               // Last state of b_counter will be num_b
 132   2               if ( b_counter < num_b )
 133   2               {
 134   3                      S1DAT = *(i2cbuf+b_counter); /* send 1 byte data */
 135   3
 136   3
 137   3                      S1CON = I2C_RLS_ACK;
 138   3               }
 139   2               else
 140   2               {
 141   3                      S1CON = I2C_STOP; /* all data were sent,stop bus */
 142   3                }
 143   2            break;
```

4.6 Source File: I2C.C (Continued)

```
144   2
145   2                /* 1st byte of data has been sent but NOT ACK rcvd */
146   2           case 0x30:
147   2              b_counter++;
148   2              // Last state of b_counter will be num_b
149   2              if ( b_counter < num_b )
150   2              {
151   3                 S1DAT = *(i2cbuf+b_counter); /* send 1 byte data */
152   3                 S1CON = I2C_RLS_ACK;
153   3              }
154   2              else
155   2              {
156   3                 S1CON = I2C_STOP;   /* all data were sent, stop bus */
157   3              }
158   2
159   2              error_i2c |= ERR_I2C_NOTACK;
160   2           break;
161   2
162   2           /* bus arbitration lost, release bus and try to restart */
163   2           case 0x38:
164   2                 S1CON = I2C_RLS_STA;
165   2                 error_i2c |= ERR_I2C_ARBILOST;
166   2           break;
167   2
168   2           /*----------------------------------------------*/
169   2           /* following section will be MASTER receive mode */
170   2           /*----------------------------------------------*/
171   2
172   2           /*SLA+R has been sent, ACK received */
173   2           case 0x40:
174   2           if( num_b == 1 )
175   2           {
176   3              // Only one byte will be received, don't acknowledge
177   3              // This will signal the slave transmitter to stop
178   3              S1CON = I2C_RLS_NACK;
179   3           }
180   2           else
181   2           {
182   3              // More than one byte will be received, acknowledge
                    // the first one
183   3              S1CON = I2C_RLS_ACK;
184   3           }
185   2
186   2           break;
187   2
188   2
```

4.6 Source File: I2C.C (Continued)

```
189   2                 /* NOT ACK received on SLA+R, will ignore it */
190   2             case 0x48:
191   2             if( num_b == 1 )
192   2             {
193   3                // Only one byte will be received, don't acknowledge
194   3                // This will signal the slave transmitter to stop
195   3                S1CON = I2C_RLS_NACK;
196   3             }
197   2             else
198   2             {
199   3                // More than one byte will be received, acknowledge
                       // the first one
200   3                S1CON = I2C_RLS_ACK;
201   3             }
202   2             error_i2c |= ERR_I2C_NOTACK;
203   2             break;
204   2
205   2
206   2             /* a byte has been received, and ACK was returned */
207   2             case 0x50:
208   2                /* read one byte from S1DAT */
209   2                *(i2cbuf + b_counter) = S1DAT;
210   2                b_counter++;
211   2
212   2                // if this is prior to the next-to-last byte
213   2                if ( b_counter < num_b_minus_1 )
214   2                {
215   3                   // Acknowledge the next byte received
216   3                   S1CON = I2C_RLS_ACK;
217   3                }
218   2                // if this is the next-to-last byte
219   2                else
220   2                {
221   3                   // Do not acknowledge the next byte received
                          // (the last byte)
222   3                   // This will signal the slave transmitter to stop
223   3                   S1CON = I2C_RLS_NACK;
224   3                }
225   2             break;
226   2
227   2
```

4.6 Source File: I2C.C (Continued)

```
 228   2              // a byte received and NOT ACK was returned
 229   2          // This should be the last byte received
 230   2          case 0x58:
 231   2              /* read one byte from S1DAT */
 232   2              *(i2cbuf + b_counter) = S1DAT;
 233   2              b_counter++;
 234   2
 235   2              /* if this is not the last byte – error condition */
 236   2              if ( b_counter < num_b )
 237   2              {
 238   3                 S1CON = I2C_RLS_ACK;
 239   3                 error_i2c |= ERR_I2C_NOTACK;
 240   3               }
 241   2               /* if this is the last byte, then STOP bus */
 242   2               else
 243   2               {
 244   3                 S1CON = I2C_STOP;
 245   3               }
 246   2          break;
 247   2
 248   2                 // No Relevant state information is available
                           // no action required
 249   2          case   0xF8:
 250   2                 nDummy = 0;
 251   2          break;
 252   2
 253   2          // bus error due to illegal start or stop condition
                    // or SIO1 has entered an illegal state
 255   2          case   0x00:
 256   2                 S1CON = I2C_RLS_ACK;
 257   2                 error_i2c |= ERR_I2C_busERROR;
 258   2          break;
 259   2
 260   2          /* all other cases will be error in this system */
 261   2          default:
 262   2                 S1CON = I2C_RLS_ACK;
 263   2                 error_i2c |= ERR_I2C_GERROR;
 264   2          break;
 265   2       }
 266   1    }

C51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

4.7 Source Code Module: Timer

The general-purpose timer is used to insert time delays and to determine when a timeout condition has occurred. The timer is programmed so that a timer-tick interrupt occurs every 2 ms.

4.7.1 *Function: timer0_isr() (Lines 26–59)*

This is the ISR for the general-purpose timer. The timer is stopped. A constant is loaded into the timer data registers (TL0 and TH0). The timer is restarted. The timer increments and generates an interrupt when it reaches its maximum count. Each time the timer-tick interrupt occurs, the global variable *timer0_tick* is incremented by 1. The constant was calculated so that the time from the timer restart until it reaches its maximum count is 2 ms. The equation for calculating the timer reload value (TH0, TL0) from the desired timer-tick period (T) is shown below. The calculated timer reload value with the 11.0592 MHz crystal and a timer-tick period of 2 ms is F8CDh.

$$TH0, TL0 = 10000h - ((f_{osc} / 12) \times T)$$

$$TH0, TL0 = 10000h - ((11059200 / 12) \times 0.002)$$

$$TH0, TL0 = F8h, CDh$$

4.7.2 *Function: timer0_initialize() (Lines 61–94)*

This function initializes the general-purpose timer. It is called once at program startup. The timer-tick count is initialized to zero and then the timer is stopped. The timer mode is set for 16-bit counter with no prescale. The timer reload value is written. The timer interrupt is enabled and given low (normal) priority. The timer is restarted. The global interrupt control (EA) is enabled. At this point, the timer-tick interrupts start occurring.

4.7.3 *Function: ResetTickCount() (Lines 96–112)*

The call to *ResetTickCount()* resets the timer-tick count to zero. The timer-tick count can run up to about 128 seconds (2 ms × 64 k) before rolling over. In a program with multiple uses for the general-purpose timer, the timer-tick count should be reset only in the outermost loop.

4.7.4 *Function: current_tick() (Lines 114–131)*

This function returns the current timer-tick count.

4.7.5 *Function: timer0_elapsed_count() (Lines 133–150)*

This function returns the number of elapsed timer-tick counts. The parameter is the starting timer-tick count from which to measure the elapsed time.

4.7.6 *Function: timer0_wait() (Lines 152–167)*

This function generates a time delay. The parameter is the number of timer-tick counts to delay.

4.8 Header File: Timer.H

```
// Timer.H
//
// Header file for P80C652 microcontroller general purpose timer routines
//
#define TCLK           11059200       /* Clock speed in Hz */

// One TICK is 2ms
#define TEN_MS                            5u
#define ONE_HUNDRED_MS                   50u
#define ONE_SECOND                      500u
#define TEN_SECONDS                    5000u

void     timer0_initialize (void);
unsigned current_tick (void);
unsigned timer0_elapsed_count (unsigned int start_tick);
void     timer0_wait (unsigned int num_tick);
void     ResetTickCount( void );
```

4.9 Source File: TIMER.C

```
DOS C51 COMPILER V5.10, COMPILATION OF MODULE TIMER
OBJECT MODULE PLACED IN TIMER.OBJ
COMPILER INVOKED BY: C:\C51\BIN\C51.EXE TIMER.C DB OE OR LARGE

stmt level    source

   1          // Timer.C
   2          //
   3          // P80C652 microcontroller general purpose timer routines
   4          //
   5          #include ”Timer.h”
   6          #include ”reg652.h”
   7
   8          /*--------------------------------------------------------------------
   9          Constant Declarations
  10          Every 2 ms, TIMER0 OVERFLOW and an interrupt will occur once
  11          ---------------------------------------------------------------------
  12          */
  13
  14          // 2ms tick: 10000h – ((11,059,200 Hz/12) * 0.002) = 0xF8CD
  15          #define TIMER0_HI   (unsigned char) 0xF8
  16          #define TIMER0_LO   (unsigned char) 0xCD
  17
  18          /*
  19          ---------------------------------------------------------------------
  20          Local Variable Declarations
  21          ---------------------------------------------------------------------
  22          */
  23
  24          static unsigned xdata timer0_tick;
  25
  26          /*
  28          ---------------------------------------------------------------------
  29          static void timer0_isr (void);
  30
  31          This function is an interrupt service routine for TIMER 0. It should
  32          never be called by a C or assembly function. It will be executed
  33          automatically when TIMER 0 overflows.
  34          ----------------------------------------------------------------------
  35          */
  37
```

4.9. Source File: TIMER.C (Continued)

```
38          static void timer0_isr (void) interrupt 1 using 1
39          {
41    1
42    1     /*------------------------------------------------
43    1     Adjust the timer 0 counter so that we get another
44    1     interrupt in 2 ms.
45    1     ------------------------------------------------*/
46    1     TR0 = 0;          /* stop timer 0 */
47    1
48    1     TL0 = TIMER0_LO;
49    1     TH0 = TIMER0_HI;
50    1
51    1     TR0 = 1;          /* start timer 0 */
52    1
53    1     /*------------------------------------------------
54    1     Increment the timer-tick.  This interrupt should
55    1     occur approximately every 2ms.
56    1     ------------------------------------------------*/
57    1     timer0_tick++;
58    1
59    1     }
60
61          /*
62          ----------------------------------------------------------------------
64          void timer0_initialize (void);
65
66          set TIMER0 AS: mode 1; 16bit timer
67
68          This function enables TIMER 0.  TIMER 0 will generate a synchronous
69          Interrupt once every 2 ms.
70          ----------------------------------------------------------------------
71          */
73
74          void timer0_initialize (void)
75          {
76    1     EA = 0;           /* disable interrupts */
77    1
78    1     timer0_tick = 0;
79    1
80    1     TR0 = 0;          /* stop timer 0 */
81    1
82    1     TMOD &= ~0x0F;    /* clear timer 0 mode bits */
83    1     TMOD |= 0x01;     /* put timer 0 into 16-bit no prescale */
84    1
85    1     TL0 = TIMER0_LO;
86    1     TH0 = TIMER0_HI;
87    1
88    1     PT0 = 0;  /* set low priority for timer 0, PT0 is in IP register */
89    1     ET0 = 1;  /* enable timer 0 interrupt, ET0 is in IE register */
90    1
91    1     TR0 = 1;  /* start timer 0, TR0 is in TCON register */
92    1
93    1     EA = 1;   /* enable interrupts */
94    1     }
```

4.9. Source File: TIMER.C (Continued)

```
 95
 96          /*
 97          ---------------------------------------------------------------------
 98          void ResetTickCount( void );
 99
100          This function resets the timer-tick variable to zero. The function
101          should be used before each time the timer is used to time an event to
102          prevent incorrect operation due to the timer0_tick variable rolling
103          over to zero. This will work out to a maximum of 64K * 2ms, or 128
104          seconds.
106          ----------------------------------------------------------------------
107          */
108
109          void ResetTickCount( void )
110          {
111    1     timer0_tick = 0;
112    1     }
113
114          /*
115          ---------------------------------------------------------------------
116          unsigned current_tick (void);
117
118          This function returns the current timer0 tick count.
119          ---------------------------------------------------------------------
120          */
121
122          unsigned current_tick (void)
123          {
124    1     unsigned xdata t;
125    1
126    1     EA = 0;
127    1     t = timer0_tick;
128    1     EA = 1;
129    1
130    1     return (t);
131    1     }
```

4.9. Source File: TIMER.C (Continued)

```
 132
 133          /*
 134          ---------------------------------------------------------------------
 135          unsigned timer0_elapsed_count (unsigned count);
 136
 137          This function returns the number of timer-ticks that have elapsed since
 139          the specified count.
 140          ---------------------------------------------------------------------
 141          */
 141
 142          unsigned timer0_elapsed_count( unsigned start_tick )
 143          {
 144    1     return (current_tick() - start_tick);
 145
 146
 147
 148
 149    1
 150    1     }
 151
 152          /*-------------------------------------------------------------------
 153          void timer0_wait ( unsigned count );
 154
 155          This function waits for 'count' timer-ticks to pass.
 156          ---------------------------------------------------------------------
 157          */
 157          void timer0_wait( unsigned num_tick )
 158          {
 159    1        unsigned xdata test1;
 160    1        unsigned xdata start_count;
 161    1
 162    1        start_count = current_tick ();
 163    1
 164    1        while( ( test1 = timer0_elapsed_count( start_count ) ) <= num_tick)
 165    1        {
 166    2        }
 167    1     }
 168

C51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

4.10 Source Code Module: VIP5020

This module contains the TVP5020 VIP routines.

4.10.1 Function: DecoderReset() (Lines 22–35)

The function *DecoderReset()* performs two board-specific tasks. First, it initializes the P80C652's user-defined I/O port P1. This programs the SW2 and SW3 I/O bits to configure the board to use the onboard data path. Digital YUV data from TVP5020 is routed directly to TVP6000. No external cabling is required. Second, *DecoderReset()* resets the FPGA.

4.10.2 Function: HandleDownload() (Lines 37–77)

This function is called to download the microcode to the TVP5020 program memory. Two parameters are passed to this function. Parameter *nCount* is the number of data bytes to write (plus 1 for the address byte). Parameter *pInBuf* is a pointer to the data to be written. The first byte in the data must be the address for microcode downloads (7Eh).

For the VIP interface to use the VIP Master FPGA, the microcode data must be broken into smaller blocks that can be handled by the FPGA, which has a 256-byte data buffer. Local variable *nMaxBlock* is initialized with the value of global variable *g_nMaxBlockSize,* which was initialized to 240 at line 20.

The pointer *pCode* to the caller's data is incremented past the address byte in the input buffer. Local variable *nMyCount* is initialized with *nCount* and is decremented by 1 to account for skipping the address byte in the input buffer.

A *while* loop is entered to transmit all the data in the caller's buffer to the FPGA. The local variable *nBlockByteCount*, the number of bytes to write to the FPGA, is determined to be either *nMaxBlock* or the remainder. Unless all data has been transmitted, *initia_i2c()* is called to prepare for an I2C transaction. *WriteTVP5020VIP()* is called to insert the VIP control byte, copy the data to the global array *g_pBlkBuf,* and report the new number of bytes contained in *g_pBlkBuf*. Finally. The function *start_i2c()* is called to transfer the contents of *g_pBlkBuf* to the FPGA. After all data has been written, *nMyCount* becomes 0; this value is loaded into *nBlockByteCount* and the loop terminates.

4.10.3 Function: WriteTVP5020() (Lines 79–92)

This function is called to write to the TVP5020. Two parameters are passed to this function. Parameter *nData* is the number of data bytes to write (plus 1 for the address byte). Except for microcode downloads, *nData* must always be two (2), and exactly one (1) data byte must be written. Parameter *pBuf* is a pointer to the data to be written. The first byte in the data must be the address.

First, *initia_i2c()* is called to prepare for an I2C transaction. Next, *WriteTVP5020VIP()* is called to insert the VIP control byte, copy the data to the array *g_pBlkBuf,* and report the new number of bytes contained in *g_pBlkBuf*. The third parameter to *WriteTVP5020VIP()* is a 0, indicating that the address byte is present as the first byte of the caller's buffer. Finally, *start_i2c()* is called to transfer the contents of *g_pBlkBuf* to the FPGA. The number of bytes actually sent to the FPGA is returned.

4.10.4 Function: GetControlByte() (Lines 94–132)

This function looks at the flags that have been set by the caller and whether this is a write or a read operation, and returns the proper control byte as defined by the VIP specification. The flags and corresponding control bytes are defined in the file VIP5020.H.

4.10.5 Function: WriteTVP5020VIP() (Lines 134–170)

This function copies the VIP control byte, address byte, and data array to *g_pBlkBuf,* and reports the new number of bytes contained in *g_pBlkBuf*. If the FLAG_DATA_ONLY bit in the nFlags parameter is nonzero, the parameter *nAddrIn* is used as the address. Otherwise, the address is taken from the first byte of the buffer pointed to by *pBuf*.

4.10.6 Function: ReadSwitch() (Lines 172–189)

This function is VIP-emulator prototype-specific. It reads the logic levels of the DIP switches through two user-assigned I/O pins of the P80C652. The result is packed into an 8-bit return value.

4.10.7 Function: ReadTVP5020() (Lines 192–257)

This function is called to read from the TVP5020. The function is not used in this program, but is included for reference. Three parameters are passed to this function. Parameter *nLength* is the number of data bytes to read. Except for microcode uploads, parameter *nLength* must always be one (1), and exactly one (1) data byte must be read. Parameter *pBuf* is a pointer to the caller's buffer where the read data is to be stored. Parameter *nSubAddr* is the address to read from.

First, the global variable *g_bVIPNLREnbl* is tested. If true, the no-latency-read method is used. In this case, two register reads will be performed: one from the NLR phase 1 address, which initiates a register prefetch operation, and one from the NLR phase 2 address, which obtains the data.

The function *initia_i2c()* is called to prepare for an I2C transaction. The function *ReadTVP5020VIP1()* is called to prepare the data packet for the VIP read operation before this data is sent to the FPGA via the I2C bus. The function *start_i2c()* is called to write the data packet to the FPGA. The *bErrorFlag* flags an error if an incorrect number of bytes was sent. The function *start_i2c()* is called again to read back the results from the FPGA. Finally, *ReadTVP5020VIP2()* is called to extract only the read data bytes from the *g_pBlkBuf* buffer and copy this data to the caller's buffer.

4.10.8 Function: ReadTVP5020VIP1() (Lines 259–294)

This function prepares the data packet for the VIP read operation before this data is sent to the FPGA via the I2C bus. *ReadTVP5020VIP1()* inserts the VIP control byte, the optional address byte, and the requested number of dummy bytes into the *g_pBlkBuf* array, and reports the new number of bytes contained in *g_pBlkBuf*. If the flag FLAG_DATA_ONLY is nonzero, the address byte is not inserted.

4.10.9 Function: ReadTVP5020VIP2() (Lines 295–328)

This function is called after the FPGA has transferred the read data from the TVP5020, via the VIP interface, into the FPGA buffer, and the data packet has been read from the FPGA via I2C into the *g_pBlkBuf* array. *ReadTVP5020VIP2()* copies only the read data bytes from the *g_pBlkBuf* array to the caller's buffer. If the flag FLAG_DATA_ONLY is nonzero, the address byte is not present in the *g_pBlkBuf* array, and an offset of 1 is used to point to the first byte of read data. Otherwise, the address byte is present and an offset of 2 is used.

4.11 Header File: VIP5020.H

```
// VIP5020.H
//
// Header file for TVP5020 routines using VIP interface
//
#define FALSE                        0
#define TRUE                    !FALSE

// To indicate current operation to FormatDataPacket()
#define STATE_WRITE                  0
#define STATE_READ1                  1
#define STATE_READ2                  2

// Flags for indicating which VIP control byte to use
#define FLAG_RD_CFG             0x0010
#define FLAG_NLR_PHS1           0x0020
#define FLAG_NLR_PHS2           0x0040
#define FLAG_RD_FIFO_S0         0x0080
#define FLAG_RD_FIFO_S1         0x0100
#define FLAG_RD_FIFO            0x0200
#define FLAG_WR_INTR            0x0400

// VIP Control Bytes
#define VIP_REG_WR_CMD            0x41
#define VIP_CFG_RD_CMD            0x60
#define VIP_REG_RD_CMD            0x61
#define VIP_NLR_PHS1              0x62
#define VIP_NLR_PHS2              0x63
#define VIP_FFO_RD_ST0            0x70
#define VIP_FFO_RD_ST1            0x71
#define VIP_FFO_RD_DAT            0x74

// Indicates no address byte is present
#define FLAG_DATA_ONLY               1
// Identifies hardware platform
#define VIP_EMULATOR_PROTOTYPE       3
// FPGA I2C device ID's
#define FPGA_WRITE_DEVICE_ID      0xE0
#define FPGA_READ_DEVICE_ID       0xE1
// Special addresses in TVP5020
#define TVP5020_LOAD_SUB_ADDR     0x7E
#define TVP5020_RESTART_SUB_ADDR  0x7F

void          DecoderReset( void );
void          HandleDownload( unsigned nCount, unsigned char* pInBuf );
unsigned      WriteTVP5020(int nData, unsigned char *pBuf );
int           WriteTVP5020VIP( int nLen, unsigned char *pBuf, int nFlags,
                               int nAddrIn, int nCtl );
unsigned char GetControlByte( int nFlags, int nState );
unsigned char ReadSwitch( void );

#if(0)
unsigned ReadTVP5020(int nLength, unsigned char *pBuf, unsigned char nSubAddr );
int      ReadTVP5020VIP1( int nLen, unsigned char *pBuf, int nFlags, int nDummies,
                          int nCtl );
int      ReadTVP5020VIP2( int nLen, unsigned char *pBuf, int nFlags, int nParam );
#endif
```

4.12 Header File: VIP5020.C

```
DOS C51 COMPILER V5.10, COMPILATION OF MODULE VIP5020
OBJECT MODULE PLACED IN VIP5020.OBJ
COMPILER INVOKED BY: C:\C51\BIN\C51.EXE VIP5020.C DB OE OR LARGE

stmt level    source

   1          //
   2          // VIP5020.C
   3          //
   4          // Routines for TVP5020 using VIP bus
   5          //
   6          #include ”VIP5020.h”
   7          #include ”I2C.h”
   8          #include ”Reg652.h”
   9
  10          // Identify the board
  11          unsigned char      g_nBoardID = VIP_EMULATOR_PROTOTYPE;
  12
  13          // Variable for enabling No-latency read function
  14          unsigned char g_bVIPNLREnbl = FALSE;
  15
  16          // Buffer from which to send data to the VIP emulation FPGA
  17          unsigned char xdata g_pBlkBuf[256];
  18
  19          // Maximum block size for data sent to FPGA buffer
  20          unsigned      g_nMaxBlockSize = 240;
  21
  22          void DecoderReset( void )
  23          {
  24   1          unsigned char  nOutBuf = 0xFF;
  25   1          unsigned char* pOutBuf = &nOutBuf;
  26   1
  27   1          // Select internal data path on the VIP Emulator Prototype
  28   1          P1 = 0xFB;
  29   1
  30   1          // Send reset command to FPGA
  31   1          initia_i2c();
  32   1          start_i2c(FPGA_WRITE_DEVICE_ID, 1, pOutBuf);
  33   1
  34   1          return;
  35   1      }
  36
  37          void HandleDownload( unsigned nCount, unsigned char* pInBuf )
  38          {
  39   1          unsigned  nBlockByteCnt = 0;
  40   1          unsigned  nMyCount = nCount;
  41   1          unsigned  bDone = FALSE;
  42   1          unsigned  nMaxBlock = g_nMaxBlockSize;
  43   1          unsigned  newLen = 0;
  44   1          unsigned char* pCode = pInBuf;
  45   1          unsigned char nCtl = 0;
  46   1
  47   1          // Advance past sub-address byte in input stream
  48   1          pCode++;
  49   1          nMyCount--;
```

4.12 Header File: VIP5020.C (Continued)

```
50   1
51   1         while( !bDone )
52   1         {
53   2             nBlockByteCnt = ( nMyCount >= nMaxBlock ) ? nMaxBlock :
                                  nMyCount;
54   2             if( nBlockByteCnt == 0 )
55   2             {

56   3                 bDone = TRUE;
57   3             }
58   2             else
59   2             {
60   3
61   3                 initia_i2c();
62   3
63   3                 // Format data based on bus interface
64   3                 nCtl = GetControlByte( 0, STATE_WRITE );
65   3                 newLen = WriteTVP5020VIP( nBlockByteCnt, pCode,
66   3                                           FLAG_DATA_ONLY,
67   3                                           TVP5020_LOAD_SUB_ADDR, nCtl );
68   3
69   3                 start_i2c(FPGA_WRITE_DEVICE_ID, newLen, g_pBlkBuf);
70   3
71   3                pCode += nBlockByteCnt;
72   3                nMyCount -= nBlockByteCnt;
73   3             }
74   2         } //while( !bDone )
75   1
76   1         return;
77   1     }
78
79         unsigned WriteTVP5020(int nData, unsigned char *pBuf )
80         {
81   1         int newLen = 0;
82   1         unsigned char nCtl = 0;
83   1
84   1         initia_i2c();
85   1
86   1         // Format data based on bus interface
87   1         nCtl = GetControlByte( 0, STATE_WRITE );
88   1         newLen = WriteTVP5020VIP( nData, pBuf, 0, 0, nCtl );
89   1
90   1         // return actual number of bytes sent
91   1         return( start_i2c(FPGA_WRITE_DEVICE_ID, newLen, g_pBlkBuf) );
92   1     }
93
```

4.12 Header File: VIP5020.C (Continued)

```
 94          unsigned char GetControlByte( int nFlags, int nState )
 95          {
 96   1          unsigned char nRtn = 0;
 97   1
 98   1          if( nFlags & FLAG_RD_CFG )
 99   1          {
100   2                  nRtn = VIP_CFG_RD_CMD;
101   2          }
102   1          else if( nFlags & FLAG_NLR_PHS1 )
103   1          {
104   2                  nRtn = VIP_NLR_PHS1;
105   2          }
106   1          else if( nFlags & FLAG_NLR_PHS2 )
107   1          {
108   2                  nRtn = VIP_NLR_PHS2;
109   2          }
110   1          else if( nFlags & FLAG_RD_FIFO_S0 )
111   1          {
112   2                  nRtn = VIP_FFO_RD_ST0;
113   2          }
114   1          else if( nFlags & FLAG_RD_FIFO_S1 )
115   1          {
116   2                  nRtn = VIP_FFO_RD_ST1;
117   2          }
118   1          else if( nFlags & FLAG_RD_FIFO )
119   1          {
120   2                  nRtn = VIP_FFO_RD_DAT;
121   2          }
122   1          else if( nState == STATE_WRITE )
123   1          {
124   2                  nRtn = VIP_REG_WR_CMD;
125   2          }
126   1          else
127   1          {
128   2                  nRtn = VIP_REG_RD_CMD;
129   2          }
131   1          return( nRtn );
132   1      }
133
134         int WriteTVP5020VIP( int nLen, unsigned char *pBuf, int nFlags,
135                              int nAddrIn, int nCtl )
135          {
136   1          // nLen     - byte count including sub-adress byte and data bytes
                 //            OR just data bytes
137   1          // pBuf     - pointer to caller's buffer
138   1          // nAddrIn  - address to use when buffer has no address
139   1
140   1          int i = 0;
141   1          int nRtn = 0;
142   1          int nDataLen = 0;
143   1          unsigned char* pDownload = g_pBlkBuf;
144   1
```

4.12 Header File: VIP5020.C (Continued)

```
145   1         // Single Address
146   1         *pDownload++ = nCtl;
147   1         if( nFlags & FLAG_DATA_ONLY )
148   1         {
149   2             // No address in the input buffer, so pass it in
150   2             // nLen includes data bytes only
151   2             nDataLen = nLen;
152   2             *pDownload++ = nAddrIn;
153   2         }
154   1         else
155   1         {
156   2             // Addr + data in input buffer
157   2             // nLen includes # data bytes + 1
158   2             *pDownload++ = *pBuf++;
159   2             nDataLen = nLen - 1;
160   2             // pBuf now points to first data byte
161   2         }
162   1
163   1         for( i=0; i<nDataLen; i++ )
164   1         {
165   2             *pDownload++ = *pBuf++;
166   2         }
167   1
168   1         nRtn = nDataLen + 2;
169   1         return( nRtn );
170   1     }
171
172         unsigned char ReadSwitch( void )
173         {
174   1         unsigned char nVal = 0;
175   1
176   1           // See REG652.H for SW1...SW3 definitions
177   1           S0   = 1;     //MSB
178   1           SW1  = 1;     //LSB
179   1
180   1           if ( S0 == 1 )
181   1           {
182   2              nVal += 2;
183   2           }
184   1           if ( SW1  == 1 )
185   1           {
186   2              nVal += 1;
187   2           }
188   1           return( nVal );
189   1     }
190
```

4.12 Header File: VIP5020.C (Continued)

```
191         #if(0)
             unsigned ReadTVP5020(int nLength, unsigned char *pBuf,
                                  unsigned char nSubAddr )
             {
                 // nLength  = # data bytes to read
                 // pBuf     = Pointer to where first byte read should be stored
                 // nSubAddr = sub-address to read

                 unsigned char* pSubAddr = &nSubAddr;
200              int newLen = 0;
                 int nLoops = 1;
                 int nLoopCnt = 0;
                 int nFlags = 0;
                 unsigned nTest1 = 0;
                 unsigned nTest2 = 0;
                 unsigned char bErrorFlag = FALSE;
                 unsigned char nCtl = 0;

                 // determine number of loops
                 if( g_bVIPNLREnbl )
                 {
210                  nLoops = 2;
                     nFlags |= FLAG_NLR_PHS1;
                 }

                 initia_i2c();   /* initialize I2C bus */

                 while( nLoopCnt < nLoops )
                 {
                     // Check for 2nd pass
                     if( nLoopCnt > 0 )
220                  {
                         nFlags &= ~FLAG_NLR_PHS1;
                         nFlags |= FLAG_NLR_PHS2;
                     }

                     // This will  write the required instructions for VIP
                     nCtl = GetControlByte( 0, STATE_READ1 );
                     newLen = ReadTVP5020VIP1( 1, pSubAddr, 0, nLength, nCtl );

                     nTest1 = start_i2c( FPGA_WRITE_DEVICE_ID, newLen, g_pBlkBuf);
230                  bErrorFlag = ( ( nTest1 != newLen ) || bErrorFlag ) ?
                                  TRUE :  FALSE;

                     // For VIP, these remain pointing to the g_pBlkBuf buffer
                     // where VIP instructions have been stored.  The same
                     // number of bytes will be read back.
                     // Read all data into download buffer
                     nTest2 = start_i2c( FPGA_READ_DEVICE_ID, newLen, g_pBlkBuf );
                     bErrorFlag = ( ( nTest2 != newLen ) || bErrorFlag ) ?
239                               TRUE : FALSE;
```

4.12 Header File: VIP5020.C (Continued)

```
240                     // Copy the data to caller's buffer
                        // 1st pass data gets written over by second pass data for
                        // VIP no-latency reads
                        nCtl = GetControlByte( 0, STATE_READ2 );
                        newLen = ReadTVP5020VIP2( nLength, pBuf, 0, 0 );

                        nLoopCnt++;
                    }

                    if( bErrorFlag )
250                 {
                        return( 0u );
                    }
                    else
                    {
                        return( nLength );
                    }
                }

                int ReadTVP5020VIP1( int nLen, unsigned char *pBuf, int nFlags,
260                                  int nDummies, int nCtl )
                {
                    // nLen     - byte count of the sub-address = 1
                    // pBuf     - pointer to the sub-address
                    // nDummies - number of dummy data bytes to reserve

                    // This routine loads the i2c buffer with VIP commands needed
                    // for a read and returns the number of bytes to write on i2c

                    int x = nLen; // To avoid unused parameter warning
270                 int j = 0;
                    int nAddr = 0;
                    int nRtn = 0;
                    unsigned char* pDownload = g_pBlkBuf;

                    // initial sub-address to write to
                    nAddr = *pBuf;

                    // Read from a single address
                    *pDownload++ = nCtl;
280                 if( nFlags & FLAG_DATA_ONLY )
                    {
                        nRtn = nDummies + 1;
                    }
                    else
                    {
                        *pDownload++ = nAddr;
                        nRtn = nDummies + 2;
                    }
                    for( j=0; j<nDummies; j++ )
290                 {
                       *pDownload++ = 0; // Dummy data
                    }
                    return( nRtn );
294             }
```

4.12 Header File: VIP5020.C (Continued)

```
295           int ReadTVP5020VIP2( int nLen, unsigned char *pBuf, int nFlags,
                                   int nParam )
              {
                  // nLen     – number of data bytes to read
                  // pBuf     – pointer to caller's buffer
300               // nFlags   – FLAG_DATA_ONLY or 0
                  // nParam   – unused

                  // This routine copies the read data to the caller's buffer

                  // To avoid unused parameter warning
                  int x2 = nParam;

                  int i = 0;
                  int nRtn = 0;
310               unsigned char* pDownload = g_pBlkBuf;

                  if( nFlags & FLAG_DATA_ONLY )
                  {
                      pDownload += 1;
                  }
                  else
                  {
                      pDownload += 2;
                  }
320
                  for( i=0; i<nLen; i++ )
                  {
                      *(pBuf+i) = *pDownload++;
                  }
                  nRtn = ( nFlags & FLAG_DATA_ONLY ) ? nLen + 1 : nLen + 2;

                  return( nRtn );
              }

330           #endif

C51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

4.13 Source Code Module: I2C6000

This module contains the TVP6000-specific I2C routines and initialization data for the TVP6000.

4.13.1 Function: read_tvp6000() (Lines 23–40)

This function can be called to read from a TVP6000 register. The function is not used in this program, but is included here for reference. Parameter *read_length* is the number of data bytes to read. Parameter *sub_addrs* is the sub-address to read from. Parameter *output_buf* is a pointer to the caller's buffer where the read data is to be stored.

For the TVP6000, there is no restriction on the number of bytes read per I2C transaction. The I2C function *start_i2c()* is called first with the TVP6000 I2C device ID for writes to write the sub-address value. Then, *start_i2c()* is called again with the TVP6000 I2C device ID for reads to read the requested number of data bytes.

4.13.2 Function: write_tvp6000() (Lines 41–50)

This function is called to write to a TVP6000 register. Two parameters are passed to this function. Parameter *write_length* is the number of data bytes to write (plus 1 for the subaddress byte). Parameter *input_buf* is a pointer to the data to be written. The first byte in the data must be the subaddress. For the TVP6000, there is no restriction on the number of bytes written per I2C transaction. The I2C function *start_i2c()* is called to perform the I2C data transfer. The TVP6000 I2C device ID for writes is passed as a parameter.

4.13.3 Function: LoadTVP6000() (Lines 52–75)

This function initializes the TVP6000 registers. The parameter *nMode* indicates the video mode that will be set up. The register data for the selected video mode is written to the TVP6000 by passing the array name of the proper register data set to *write_tvp6000()*. Finally, the function *PatchTVP6000()* is called.

4.13.4 Function: PatchTVP6000() (Lines 77–97

This function performs an initialization step that is necessary to properly initialize the YUV interface. The YUV format is momentarily changed to 16-bit YUV 4:2:2 mode and then back to the 8-bit ITU-R BT.656 interface mode.

The global variable *g_nBoard* is tested to determine the value to write into subaddress 3Ah: 0Fh to use video port 1, or 1Fh to use video port 2. This was inserted only to make the same TVP6000 code work on both the TVP56000EVM and the VIP emulator prototype. Normally, this *if* statement should be taken out.

4.14 Header File: I2C6000.H

```
// I2C6000.H
//
// Header file for TVP6000 routines
//
#define ENCODER_WRITE    0x42 // I2C writing address of encoder
#define ENCODER_READ     0x43 // I2C reading address of encoder
#define NUM_OF_REGISTER  0x65 // No of TVP6000 registers to load
#define BOARD_TVP56000EVM   5

#if(0)
void read_tvp6000(int length_of_read, unsigned char first_subaddress,
             unsigned char* readout_buffer);
#endif

void write_tvp6000(int length_of_write, unsigned char *write_buf);
void LoadTVP6000( int nMode );
void PatchTVP6000( void );
```

4.15 Source File: I2C6000.C

```
DOS C51 COMPILER V5.10, COMPILATION OF MODULE I2C6000
OBJECT MODULE PLACED IN I2C6000.OBJ
COMPILER INVOKED BY: C:\C51\BIN\C51.EXE I2C6000.C DB OE OR LARGE

stmt level    source

   1          //
   2          // I2C6000.C
   3          //
   4          // Routines for TVP6000 using I2C bus
   5          //
   6          #include ”I2C.h”
   7          #include ”I2C6000.h”
   9          #include ”timer.h”
   8          #include ”DATA6000.h”
  10
  11          extern unsigned char      g_nBoardID;
  12
  13          static unsigned char patch1_6000[2] =
  14          {
  15             0x3A, 0x0D    //16-bit YUV 4:2:2, color bar TPG OFF
  16          };
  17
  18          static unsigned char patch2_6000[2] =
  19          {
  20             0x3A, 0x0F    // CCIR 656, VP1
  21          };
  22
  23          #if(0)
              void read_tvp6000(int read_length, unsigned char sub_addrs,
                                  unsigned char *output_buf)
              {
                  unsigned char *sub_begin;
                  sub_begin = &sub_addrs;

                  initia_i2c();   /* initialize I2C bus */

                  /* this will write the first SUB-address to TVP6000 */
                  /* after this, data will be read out start from this address */
                  start_i2c(ENCODER_WRITE, 1, sub_begin);

                  /* read all registers of TVP6000, and put them into a temporary
                     buffer */
                  start_i2c(ENCODER_READ, read_length, output_buf);
                  return;
              }
              #endif
  40
```

4.15 Source File: I2C6000.C (Continued)

```
41          void write_tvp6000(int write_length, unsigned char *input_buf)
42          {
43   1          initia_i2c();   /* initialize I2C bus */
44   1
45   1          // This will write the first sub-address to TVP6000
46   1          // after this, data will be read out starting from this address
47   1          // subaddress will be 1st element in the input_buf
48   1          start_i2c(ENCODER_WRITE, write_length, input_buf);
49   1          return;
50   1      }
51
52          void LoadTVP6000( int nMode )
53          {
54   1          switch( nMode )
55   1          {

56   2              case 0:
57   2                  write_tvp6000(NUM_OF_REGISTER, T600N601);
58   2              break;
59   2
60   2              case 1:
61   2                  write_tvp6000(NUM_OF_REGISTER, T600NSQP);
62   2              break;
63   2
64   2              case 2:
65   2                  write_tvp6000(NUM_OF_REGISTER, T600P601);
66   2              break;
67   2
68   2              case 3:
69   2                  write_tvp6000(NUM_OF_REGISTER, T600PSQP);
70   2              break;
71   2          }
73   1          PatchTVP6000();
74   1          return;
75   1      }
76
77         void PatchTVP6000( void )
78          {
79   1             unsigned nStartPoint = 0;
80   1
81   1             write_tvp6000( 2, patch1_6000 );
82   1
83   1             // Delay for 100ms
84   1             nStartPoint = current_tick ();
85   1             while( timer0_elapsed_count( nStartPoint ) < ONE_HUNDRED_MS )
86   1             { ; }

90   1             if( g_nBoardID == BOARD_TVP56000EVM )
91   1             {
92   2                    // Use video port 2 for TVP56000EVM
93   2                    patch2_6000[1] = 0x1F;
94   2             }
95   1             write_tvp6000( 2, patch2_6000 );
96   1             return;
97   1      }

C51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

4.16 TVP6000 Initialization Data for NTSC with CCIR601 Sampling

```
// DATA6000.H
// Header file containing all TVP6000 register initialization data.
//
unsigned char code T600N601[] = {
      /* Register Name                         Sub-Address */
0x3A, /* SUB-ADDRESS                                N/A    */
0x0F, /* F_CONTROL                                  3A     */
      /* RESERVED                                          */
                  0x00, 0x00, 0x00, 0x00, 0x00,  /* 3B-3F  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 40-47  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 48-4F  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 50-57  */
0x00, 0x00,                                      /* 58-59  */
0x00, /* C_PHASE                                    5A     */
0x0E, /* GAIN_U                                     5B     */
0x7D, /* GAIN_V                                     5C     */
0xCE, /* BLACK_LEVEL                                5D     */
0xB8, /* BLANK_LEVEL                                5E     */
0x31, /* GAIN_Y                                     5F     */
0x20, /* X_COLOR                                    60     */
0x0D, /* M_CONTROL                                  61     */
0x3A, /* BSTAMP                                     62     */
0x1F, /* S_CARR1                                    63     */
0x7C, /* S_CARR2                                    64     */
0xF0, /* S_CARR3                                    65     */
0x21, /* S_CARR4                                    66     */
0x00, /* LINE21_O0                                  67     */
0x00, /* LINE21_O1                                  68     */
0x00, /* LINE21_E0                                  69     */
0x00, /* LINE21_E1                                  6A     */
0x12, /* LN_SEL                                     6B     */
0x00, /* SYN_CTRL0                                  6C     */
0x40, /* RCML21                                     6D     */
0xF2, /* HTRIGGER0                                  6E     */
0x00, /* HTRIGGER1                                  6F     */
0xC0, /* VTRIGGER                                   70     */
0x89, /* BMRQ                                       71     */
0x39, /* EMRQ                                       72     */
0x61, /* BEMRQ                                      73     */
0x08, /* X2PH                                       74     */
0x90, /* X1PH                                       75     */
0x00, /* RESERVED                                   76     */
0xEA, /* BRCV                                       77     */
0x8A, /* ERCV                                       78     */
0x60, /* BERCV                                      79     */
0x0C, /* FLEN                                       7A     */
0x06, /* FAL                                        7B     */
0x06, /* LAL                                        7C     */
0x22, /* FLAL                                       7D     */
0x0E, /* SYN_CTRL1                                  7E     */
      /* RESERVED                         0x00      7F     */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 80-87  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 88-8F  */
      /* Scaling Processor Registers                       */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 90-97  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00               /* 98-9D  */  };
```

4.17 TVP6000 Initialization Data for NTSC with Square Pixel Sampling

```
unsigned char code T600NSQP[] = {
      /* Register Name                         Sub-Address */
0x3A, /* SUB-ADDRESS                               N/A    */
0x0F, /* F_CONTROL                                 3A     */
      /* RESERVED                                         */
                  0x00, 0x00, 0x00, 0x00, 0x00,  /* 3B-3F  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 40-47  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 48-4F  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 50-57  */
0x00, 0x00,                                      /* 58-59  */
0x00, /* C_PHASE                                   5A     */
0x0E, /* GAIN_U                                    5B     */
0x7D, /* GAIN_V                                    5C     */
0xCE, /* BLACK_LEVEL                               5D     */
0xB8, /* BLANK_LEVEL                               5E     */
0x31, /* GAIN_Y                                    5F     */
0x20, /* X_COLOR                                   60     */
0x0D, /* M_CONTROL                                 61     */
0xBA, /* BSTAMP                                    62     */
0x55, /* S_CARR1                                   63     */
0x55, /* S_CARR2                                   64     */
0x55, /* S_CARR3                                   65     */
0x25, /* S_CARR4                                   66     */
0xA5, /* LINE21_O0                                 67     */
0x50, /* LINE21_O1                                 68     */
0xA5, /* LINE21_E0                                 69     */
0x50, /* LINE21_E1                                 6A     */
0x14, /* LN_SEL                                    6B     */
0x24, /* SYN_CTRL0                                 6C     */
0x40, /* RCML21                                    6D     */
0xDE, /* HTRIGGER0                                 6E     */
0x00, /* HTRIGGER1                                 6F     */
0xDF, /* VTRIGGER                                  70     */
0xBF, /* BMRQ                                      71     */
0xBF, /* EMRQ                                      72     */
0x50, /* BEMRQ                                     73     */
0x08, /* X2PH                                      74     */
0x90, /* X1PH                                      75     */
0x00, /* RESERVED                                  76     */
0xD3, /* BRCV                                      77     */
0xD3, /* ERCV                                      78     */
0x50, /* BERCV                                     79     */
0x0C, /* FLEN                                      7A     */
0x06, /* FAL                                       7B     */
0x06, /* LAL                                       7C     */
0x22, /* FLAL                                      7D     */
0x0E, /* SYN_CTRL1                                 7E     */
      /* RESERVED                        0x00      7F     */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 80-87  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 88-8F  */
      /* Scaling Processor Registers                      */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 90-97  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00               /* 98-9D  */  };
```

4.18 TVP6000 Initialization Data for PAL with CCIR601 Sampling

```
unsigned char code T600P601[] = {
      /* Register Name                         Sub-Address */
0x3A, /* SUB-ADDRESS                                N/A    */
0x0F, /* F_CONTROL                                  3A     */
      /* RESERVED                                          */
                  0x00, 0x00, 0x00, 0x00, 0x00,  /* 3B–3F  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 40–47  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 48–4F  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 50–57  */
0x00, 0x00,                                      /* 58–59  */
0x00, /* C_PHASE                                    5A     */
0x15, /* GAIN_U                                     5B     */
0x8C, /* GAIN_V                                     5C     */
0xBC, /* BLACK_LEVEL                                5D     */
0xBC, /* BLANK_LEVEL                                5E     */
0x45, /* GAIN_Y                                     5F     */
0x20, /* X_COLOR                                    60     */
0x0E, /* M_CONTROL                                  61     */
0x41, /* BSTAMP                                     62     */
0xCB, /* S_CARR1                                    63     */
0x8A, /* S_CARR2                                    64     */
0x09, /* S_CARR3                                    65     */
0x2A, /* S_CARR4                                    66     */
0xA2, /* LINE21_O0                                  67     */
0x2A, /* LINE21_O1                                  68     */
0xA2, /* LINE21_E0                                  69     */
0x2A, /* LINE21_E1                                  6A     */
0x14, /* LN_SEL                                     6B     */
0x00, /* SYN_CTRL0                                  6C     */
0x00, /* RCML21                                     6D     */
0x16, /* HTRIGGER0                                  6E     */
0x01, /* HTRIGGER1                                  6F     */
0x80, /* VTRIGGER                                   70     */
0x5F, /* BMRQ                                       71     */
0x5F, /* EMRQ                                       72     */
0x61, /* BEMRQ                                      73     */
0x08, /* X2PH                                       74     */
0x90, /* X1PH                                       75     */
0x00, /* RESERVED                                   76     */
0x0E, /* BRCV                                       77     */
0xAE, /* ERCV                                       78     */
0x61, /* BERCV                                      79     */
0x70, /* FLEN                                       7A     */
0x05, /* FAL                                        7B     */
0x35, /* LAL                                        7C     */
0x22, /* FLAL                                       7D     */
0x0E, /* SYN_CTRL1                                  7E     */
      /* RESERVED                         0x00      7F     */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 80–87  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 88–8F  */
      /* Scaling Processor Registers                       */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 90–97  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00               /* 98–9D  */  };
```

4.19 TVP6000 Initialization Data for PAL with Square Pixel Sampling

```
unsigned char code T600PSQP[] = {
      /* Register Name                           Sub-Address */
0x3A, /* SUB-ADDRESS                                  N/A    */
0x0F, /* F_CONTROL                                    3A     */
      /* RESERVED                                            */
                  0x00, 0x00, 0x00, 0x00, 0x00,  /* 3B-3F  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 40-47  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 48-4F  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 50-57  */
0x00, 0x00,                                      /* 58-59  */
0x00, /* C_PHASE                                      5A     */
0x15, /* GAIN_U                                       5B     */
0x8C, /* GAIN_V                                       5C     */
0xBC, /* BLACK_LEVEL                                  5D     */
0xBC, /* BLANK_LEVEL                                  5E     */
0x45, /* GAIN_Y                                       5F     */
0x20, /* X_COLOR                                      60     */
0x0E, /* M_CONTROL                                    61     */
0xC1, /* BSTAMP                                       62     */
0x0C, /* S_CARR1                                      63     */
0x8C, /* S_CARR2                                      64     */
0x79, /* S_CARR3                                      65     */
0x26, /* S_CARR4                                      66     */
0xA2, /* LINE21_O0                                    67     */
0x2A, /* LINE21_O1                                    68     */
0xA2, /* LINE21_E0                                    69     */
0x2A, /* LINE21_E1                                    6A     */
0x14, /* LN_SEL                                       6B     */
0x00, /* SYN_CTRL0                                    6C     */
0x00, /* RCML21                                       6D     */
0x32, /* HTRIGGER0                                    6E     */
0x01, /* HTRIGGER1                                    6F     */
0x80, /* VTRIGGER                                     70     */
0x5F, /* BMRQ                                         71     */
0x5F, /* EMRQ                                         72     */
0x61, /* BEMRQ                                        73     */
0x08, /* X2PH                                         74     */
0x90, /* X1PH                                         75     */
0x00, /* RESERVED                                     76     */
0x28, /* BRCV                                         77     */
0x28, /* ERCV                                         78     */
0x71, /* BERCV                                        79     */
0x70, /* FLEN                                         7A     */
0x05, /* FAL                                          7B     */
0x35, /* LAL                                          7C     */
0x22, /* FLAL                                         7D     */
0x0E, /* SYN_CTRL1                                    7E     */
      /* RESERVED                        0x00         7F     */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 80-87  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 88-8F  */
      /* Scaling Processor Registers                         */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  /* 90-97  */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00               /* 98-9D  */   };
```