TEXAS INSTRUMENTS

# TVP5020 NTSC/PAL Video Decoder

## Programming for the I²C Host Interface

Application Report

March 1999

Mixed Signal Products

SLAA052

## 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.

# Contents


**1 Introduction . . . . . 1**

**2 Hardware Platform . . . . . 2**
2.1 Block Diagram . . . . . 2
2.2 Schematic Diagram – Microcontroller Interface . . . . . 2
2.3 Schematic Diagram – TVP5020 NTSC/PAL Video Decoder . . . . . 5

**3 Program Overview . . . . . 8**
3.1 Microcontroller-Specific Macros . . . . . 9
3.2 Header File: REG652.H . . . . . 10
3.3 Source Code Modules . . . . . 12

**4 Program Description . . . . . 13**
4.1 Source Code Module: Main . . . . . 13
4.1.1 Inclusion of TVP5020 Microcode Files (Lines 11–14) . . . . . 13
4.1.2 Function: Main() . . . . . 13
4.1.3 Function: Power-Up Initialization() . . . . . 15
4.1.4 Function: Patch TVP5020 Registers() . . . . . 16
4.2 Header File: Main.H . . . . . 16
4.3 Source File: MAIN.C . . . . . 17
4.4 Source Code Module: I2C . . . . . 20
4.4.1 Function: initia_i2c() (Lines 28–41) . . . . . 20
4.4.2 Function: start_i2c() . . . . . 20
4.4.3 Function: i2c_isr() (Lines 77–266) . . . . . 20
4.5 Header File: I2C.H . . . . . 22
4.6 Source File: I2C.C . . . . . 23
4.7 Source Code Module: Timer . . . . . 29
4.7.1 Function: timer0_isr() (Lines 26–59) . . . . . 29
4.7.2 Function: timer0_initialize() (Lines 61–94) . . . . . 29
4.7.3 Function: ResetTickCount() (Lines 96–112) . . . . . 29
4.7.4 Function: current_tick() (Lines 114–131) . . . . . 29
4.7.5 Function: timer0_elapsed_count() (Lines 133–150) . . . . . 29
4.7.6 Function: timer0_wait() (Lines 152–167) . . . . . 29
4.8 Header File: Timer.H . . . . . 30
4.9 Source File: TIMER.C . . . . . 31
4.10 Source Code Module: I2C5020 . . . . . 35
4.10.1 Function: DecodeReset() (Lines 14–18) . . . . . 35
4.10.2 Function: HandleDownload() (Lines 20–24) . . . . . 35
4.10.3 Function: WriteTVP5020() (Lines 26–38) . . . . . 35
4.10.4 Function: ReadSwitch() (Lines 40–65) . . . . . 35
4.10.5 Function: ReadTVP5020() (Lines 67–89) . . . . . 35
4.11 Header File: I2C5020.H . . . . . 36
4.12 Source File: I2C5020.C . . . . . 37
4.13 Source Code Module: I2C6000 . . . . . 39
4.13.1 Function: read_tvp6000() (Lines 23–40) . . . . . 39
4.13.2 Function: write_tvp6000() (Lines 41–50) . . . . . 39
4.13.3 Function: LoadTVP6000() (Lines 52–75) . . . . . 39
4.13.4 Function: PatchTVP6000() (Lines 77–97) . . . . . 39
4.14 Header File: I2C6000.H . . . . . 40
4.15 Source File: I2C6000.C . . . . . 41
4.16 TVP6000 Initialization Data for NTSC with CCIR601 Sampling . . . . . 43
4.17 TVP6000 Initialization Data for NTSC with Square Pixel Sampling . . . . . 44

4.18 TVP6000 Initialization Data for PAL with CCIR601 Sampling . . . . . 45
4.19 TVP6000 Initialization Data for PAL with Square Pixel Sampling . . . . . 46

## List of Figures

1 TVP56000EVM Block Diagram . . . . . 2
2 Microcontroller Interface Schematic . . . . . 4
3 TVP5020 NTSC/PAL Video Decoder Schematic . . . . . 6
4 TVP56000EVM Board Layout . . . . . 7
5 Help-About Dialog Box from uVision/51 for Windows . . . . . 8
6 Help-About Dialog Box for Signum Systems In-Circuit Emulator . . . . . 9
7 TVP5020 Microcode in Hex-ASCII Format . . . . . 13
8 TVP5020 Microcode After Conversion to Standard-C Format . . . . . 14
9 TVP5020 Microcode After Modification . . . . . 14

## List of Tables

1 Microcontroller I/O Port Utilization . . . . . 5
2 DIP Switch Settings for TVP560000EVM . . . . . 5
3 Jumper Settings for TVP56000EVM Using I2C Bus . . . . . 7
4 Sampling Rate Dependent Jumper Settings . . . . . 7
5 Source Code Module Relationships . . . . . 12
6 TVP5020 Register Patches . . . . . 13
7 I2C Controller: Master Transmitter States . . . . . 21
8 I2C Controller: Master Receiver States . . . . . 21

# *TVP5020 NTSC/PAL Video Decoder Programming for the I²C Host Interface*

*Michael A. Tadyshak*

**ABSTRACT**

This application report provides a complete working example of a C-language program to initialize the video decoder using the I²C bus interface. This Example Initialization Program executes on the TVP56000EVM evaluation module featuring the TVP5020 video decoder and the TVP6000 video encoder. The topics covered include the TVP56000EVM hardware platform, microcontroller-specific aspects, and a detailed description of the source-code modules.

## 1 Introduction

The TVP5020 enables a wide range of applications by providing support for each of the following host interfaces:

- I²C (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 user can modify and rebuild the source code for quick verification on the TVP56000EVM. Also, 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

## 2.1 Block Diagram

This program was tested on the TVP56000EVM evaluation module. A block diagram of this EVM 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 $I^2C$ bus controller. The program is stored in the Atmel 29C512 64 kB flash memory device. Since this program is a scaled-down version of the complete EVM microcode program, it does not support flash reprogramming or the serial port. To install this program on the EVM, it must be programmed into the flash memory device with a PROM programmer. The EVM also has 32 kB of static RAM available for program use.

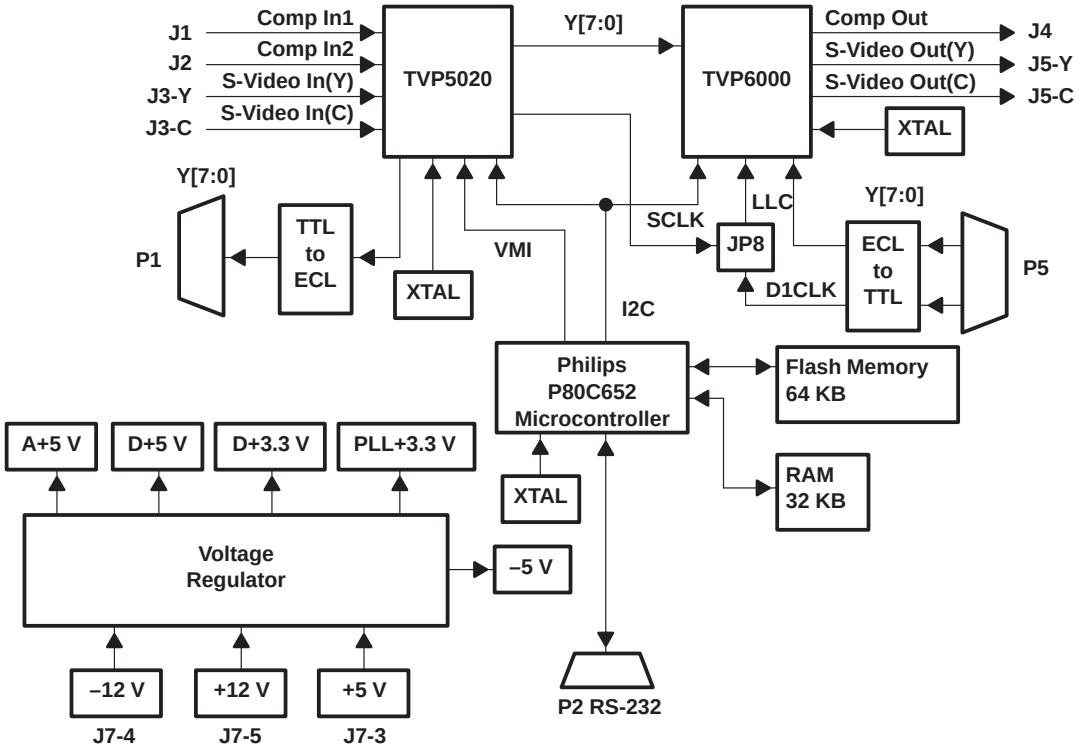



**Figure 1. TVP56000EVM Block Diagram**

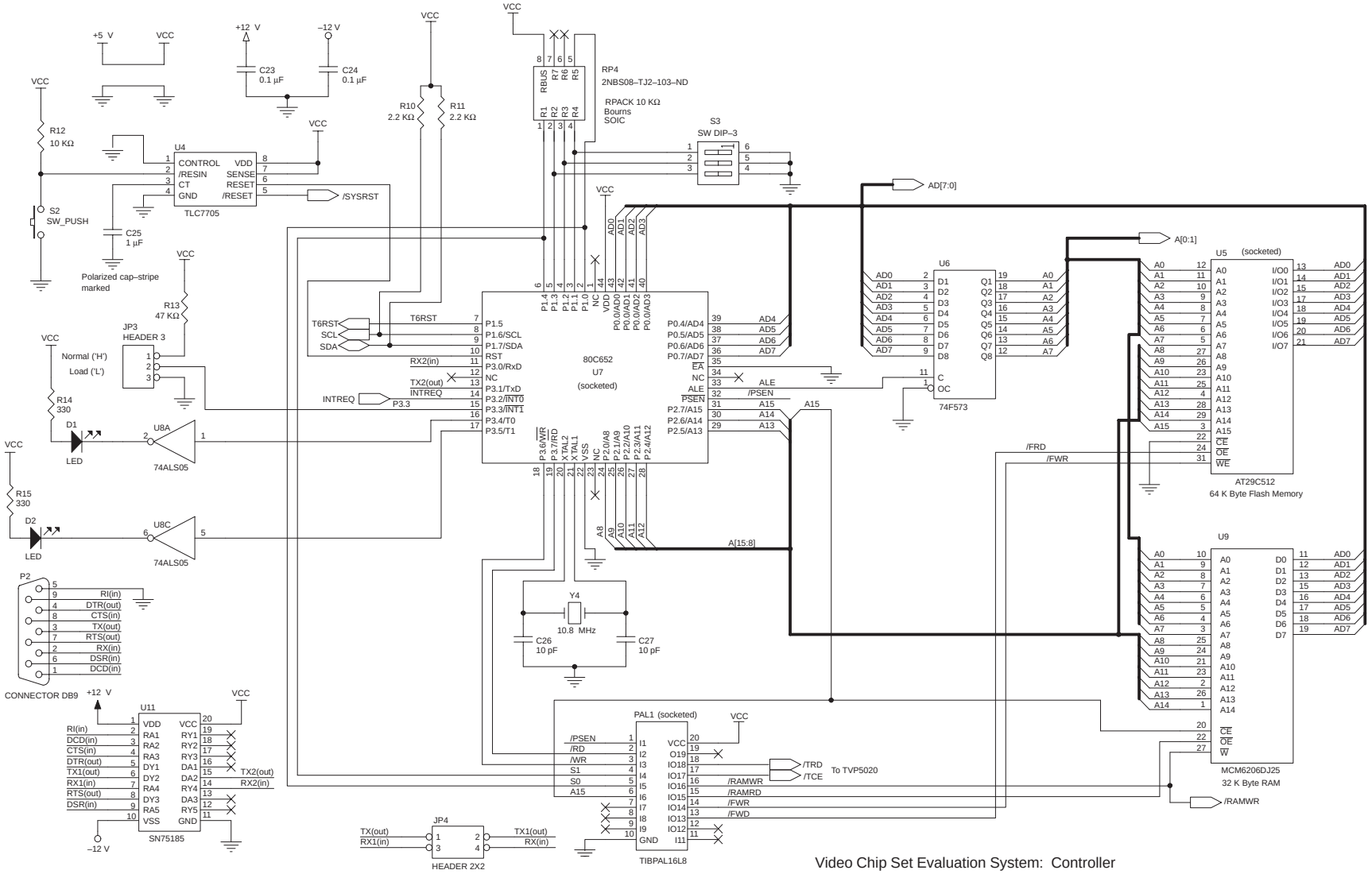
## 2.2 Schematic Diagram – Microcontroller Interface

The schematic diagram showing the microcontroller and surrounding circuitry is shown in Figure 2. Refer to this diagram to understand some of the hardware-specific aspects of the program. The P80C652 has four eight-bit I/O ports. Since this device is configured to use external memory, the I/O port P0 is used as multiplexed lower address and data (AD[7–0]), and the I/O port P2 is used to output the upper address bits A[15–8]. I/O ports P1 and P3 have some dedicated functions and are otherwise user-defined, as illustrated in Table 1.

The names in the signal name column are shown exactly as they would appear in the program. Each one is a macro representing a special-function register or special-function I/O bit as defined for the P80C652. The user-defined I/O bits INTREQ, S0, S1, FLASHJMP, LED1, and LED2 are not used by this program. SW2 and SW3 are used to read the corresponding DIP switches to select the video mode as shown in Table 2. DIP switch SW1 is not used. T6RESET provides a software means to reset the TVP6000 Video Encoder.

Notice that the I2C clock (SCL) and I2C data (SDA) signals must each be pulled up to $V_{CC}$ with a 2.2 kΩ resistor.

The programmable logic device (PAL1) controls access to the RAM and flash memory during the flash reprogram operation, and also enables the TVP5020 VMI interface to be mapped to external data memory space. For this program, PAL1's only function is to route signals straight from the microcontroller to the RAM and flash memory.




Video Chip Set Evaluation System: Controller

**Figure 2. Microcontroller Interface Schematic**

**Table 1. Microcontroller I/O Port Utilization**

| PIN NAME | SIGNAL NAME | DIRECTION | FUNCTION | DEFINITION |
|---|---|---|---|---|
| P1.0 | S0 | Output | Flash memory state lsb | User-defined |
| P1.1 | SW1 | Input | DIP switch 1 | User-defined |
| P1.2 | SW2 | Input | DIP switch 2 | User-defined |
| P1.3 | SW3 | Input | DIP switch 3 | User-defined |
| P1.4 | S1 | Output | Flash memory state msb | 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 |
| P3.0 | RXD | Input | RS–232 receive data | Dedicated I/O |
| P3.1 | TXD | Output | RS–232 transmit data | Dedicated I/O |
| P3.2 | INTREQ | Input | TVP5020 interrupt request | User-defined |
| P3.3 | FLASHJMP | Input | Flash memory jumper | User-defined |
| P3.4 | LED1 | Output | LED D1 | User-defined |
| P3.5 | LED2 | Output | LED D2 | User-defined |
| P3.6 | WR | Output | External memory write strobe | Dedicated I/O |
| P3.7 | RD | Output | External memory read strobe | Dedicated I/O |

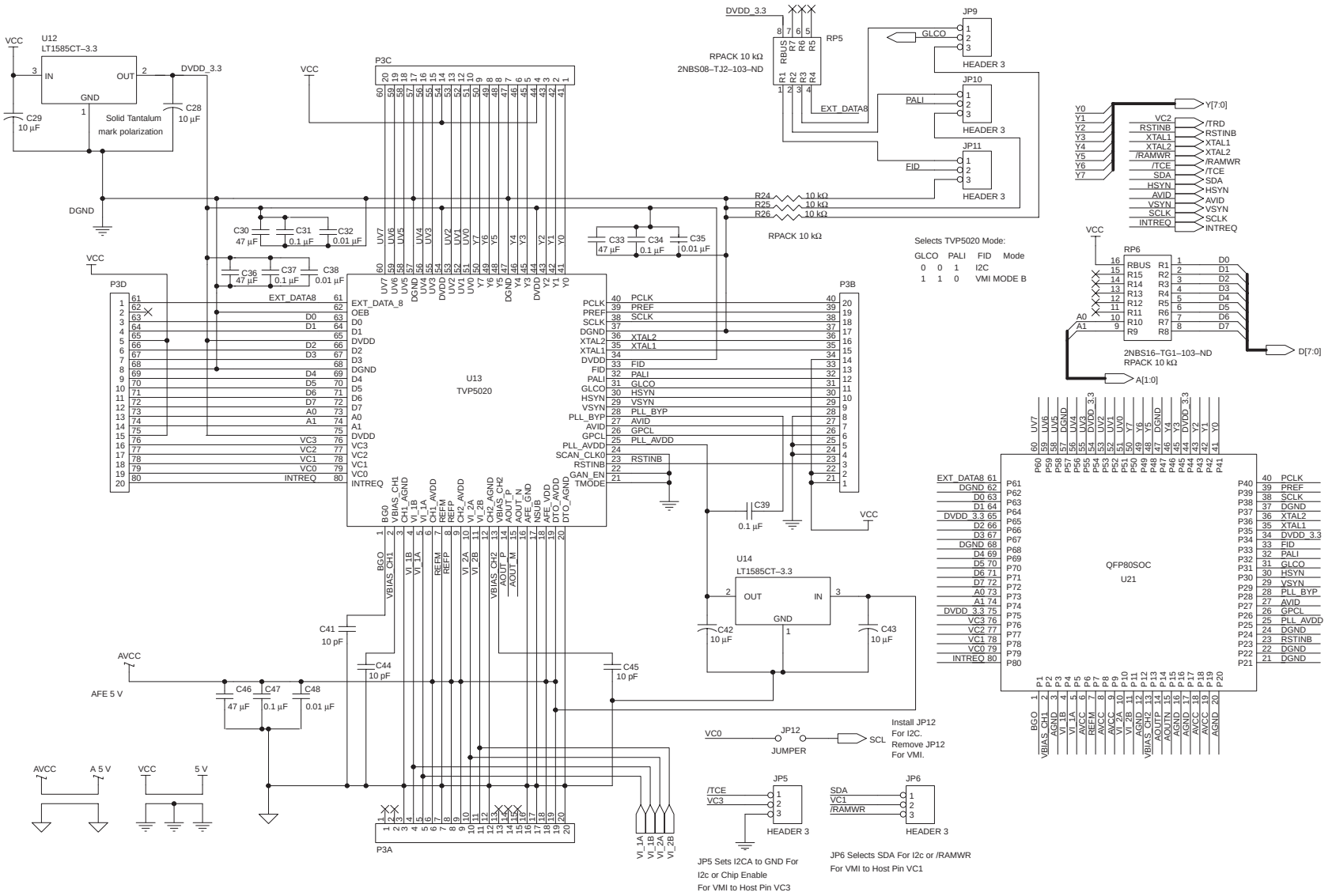
**Table 2. DIP Switch Settings for TVP560000EVM**

| VIDEO STANDARD | SAMPLING RATE | INDIVIDUAL SWITCHES | | |
|---|---|---|---|---|
| | | S3–3 | S3–2 | S3–1 |
| NTSC | CCIR601 | ON | ON | X |
| NTSC | Square pixel | OFF | ON | X |
| PAL | CCIR601 | ON | OFF | X |
| PAL | Square pixel | OFF | OFF | X |

## 2.3 Schematic Diagram – TVP5020 NTSC/PAL Video Decoder

The schematic diagram of the TVP5020 and surrounding circuitry is shown in Figure 3. The $I^2C$ bus interface is simple. TVP5020 pin VC0 (#79) is the input for the I2C bus clock (SCL). To connect the SCL from the microcontroller to the TVP5020 VC0 pin, jumper JP12 must be installed. TVP5020 pin VC1 (#78) is the I/O pin for the I2C data line (SDA). To connect the SDA from the microcontroller to the TVP5020 VC1 pin, JP6 must be jumpered across pins 1 and 2. TVP5020 pin VC3 (#76) is the input for the I2C address select (I2CA). To ground VC3, JP5 must be jumpered across pins 2 and 3. This sets the TVP5020 I2C device ID to B8h to write and B9h to read, as this program requires.

Figure 4 shows the TVP56000EVM board layout. Refer to this figure for the location and orientation of jumpers, switches, and connectors. The jumper settings are summarized in Table 3. The setting of JP4 is irrelevant because this program does not use the RS-232 serial port interface. Jumper settings to select between CCIR601 and square pixel sampling rates are shown in Table 4.




TVP5020

**Figure 3. TVP5020 NTSC/PAL Video Decoder Schematic**

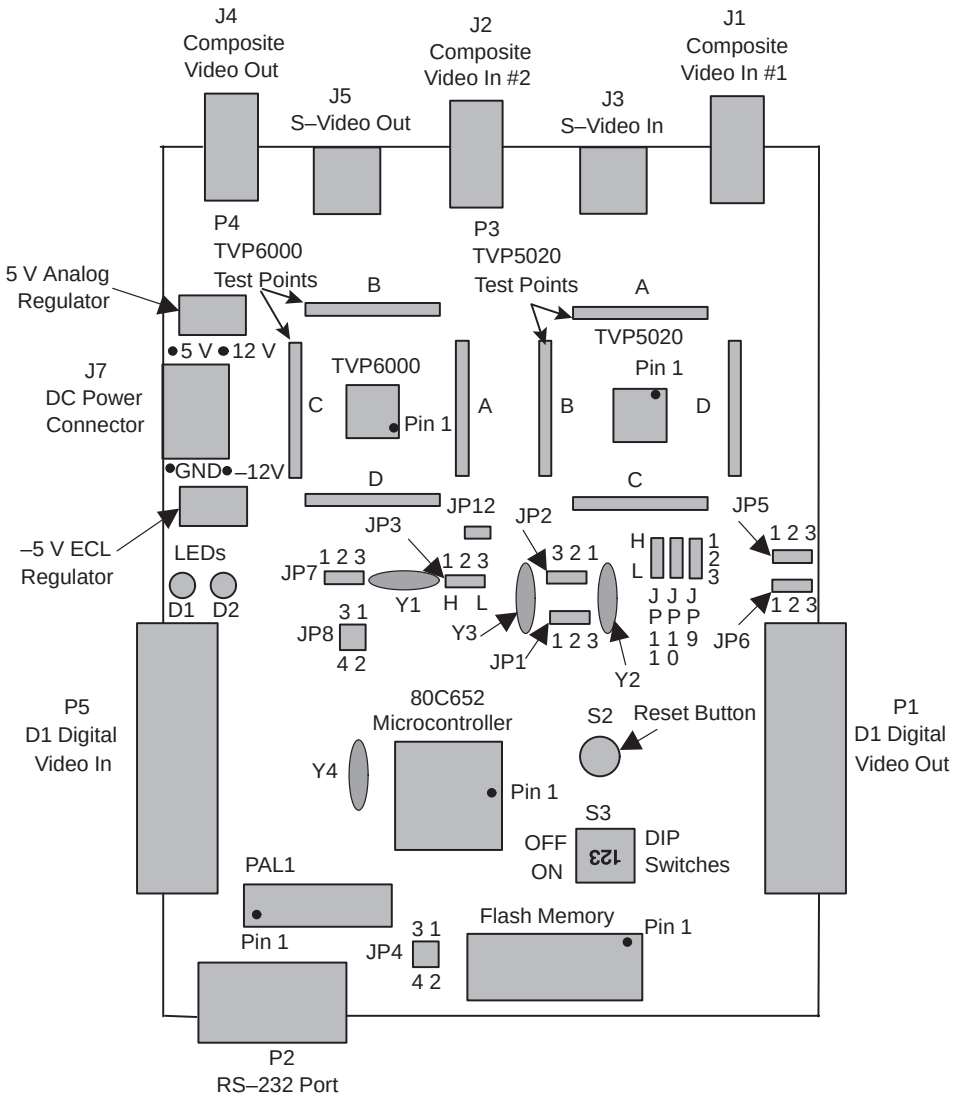



**Figure 4. TVP56000EVM Board Layout**

**Table 3. Jumper Settings for TVP56000EVM Using I2C Bus**

| JUMPER(s) | JP3 | JP4 | JP7 | JP8 | JP5 | JP6 | JP9, JP10, JP11 | JP12 |
|---|---|---|---|---|---|---|---|---|
| POSITION | H | 1–3, 2–4 | 1–2 | 1–2 | 2–3 | 1–2 | L, L, H | 1–2 |
| DESCRIPTION | NORMAL | STRAIGHT CABLE | TVP6000: CLK RCVR | SCLK | I2C | | | |

**Table 4. Sampling Rate Dependent Jumper Settings**

| SAMPLING RATE | JP1 | JP2 |
|---|---|---|
| CCIR601 | 2–3 | 1–2 |
| Square pixel | 1–2 | 2–3 |

# 3 Program Overview

This is a complete working example of a C-language program to initialize the TVP5020 NTSC/PAL video decoder using the $I^2C$ bus interface. 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 (Figure 5). 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 (Figure 6).

The linker output is in standard *Intel Intellec 8/MDS* format, a format that is widely supported by PROM programmers. The program can be installed on the TVP56000EVM by programming the flash memory device (Atmel 29C512) using a PROM programmer. TVP5020 microcode for four video modes is included in the program to enable testing of NTSC and PAL video standards using CCIR601 and square pixel sampling rates. The video mode is selected by setting the DIP-switches as was shown in Table 2.


**Figure 5. Help-About Dialog Box from *uVision/51 for Windows***

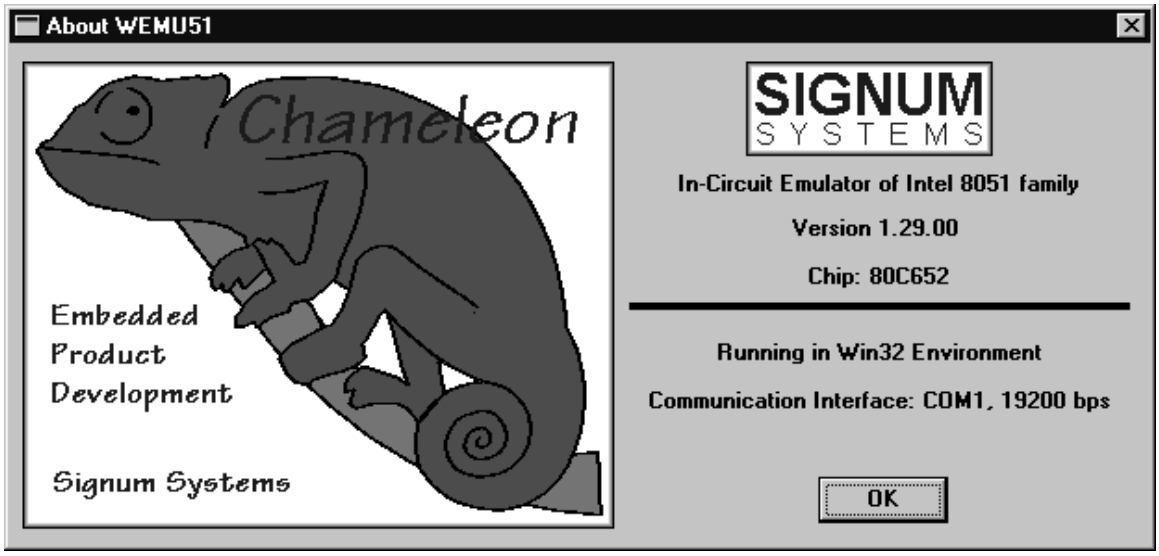



**Figure 6. Help-About Dialog Box for Signum Systems In-Circuit Emulator**

## 3.1 Microcontroller-Specific Macros

Most of the source code is standard C-language. The main exception is the use of macros to access the special-function registers, or the special-function I/O bits as defined for the P80C652. The complete set of macros defined for the P80C652 is contained in the file REG652.H (see listing beginning on page 10). As shown in Table 1, there are several user-definable I/O pins assigned for special use by the EVM. These are named in the P1 and P3 I/O ports sections of the header file. Of these user-definable I/O pins, only SW2, SW3, and T6RESET are used by this program. SW2 and SW3 are used to read the corresponding DIP-switches, and T6RESET provides a software reset for the TVP6000 video encoder.

Many of the P80C652-specific special-function registers are used to control the on-chip general-purpose timer and the $I^2C$ bus interface. These are localized to the TIMER and I2C source code.

The P80C652 provides 128 bytes of on-chip RAM, 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 required 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 are stored in flash memory.

## 3.2 Header File: REG652.H

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

/*  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 5 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,* I2C5020, and I2C6000 modules could be used virtually unchanged. In the new environment, the TVP6000 software reset (T6RESET) and the reading of the DIP-switch lines SW2 and SW3 would need to be implemented as required.

**Table 5. Source Code Module Relationships**

| SOURCE CODE MODULE | DESCRIPTION | MICROCONTROLLER-SPECIFIC? | TVP56000EVM-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 |
| I2C5020 | TVP5020-specific I2C routines | No | Yes (uses SW2 and SW3) | 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 (0×7E). The TVP5020 microcode is supplied in a five-character Hex-ASCII format (Figure 7). Conversion to a standard C-language header file can be done with a utility called *HexConv*. The output of *HexConv* is shown in Figure 8. If necessary, the *#define* constant for the code size (which includes the sub-address byte – 0×7E), 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. It is also very helpful to add a comment identifying the microcode type and version. The resulting microcode file is shown in Figure 9.

### *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 I2C 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 for the TVP5020 to function properly on the TVP56000EVM. The array *g_pTVP5020Patch[ ]* contains the I2C subaddress and data for three registers that must be modified. Table 6 describes these register changes.

**Table 6. TVP5020 Register Patches**

| I2C SUBADDRESS | 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 |

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

**Figure 7. 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,
0x80
};
```

**Figure 8. 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,
0x80
};
```

**Figure 9. TVP5020 Microcode After Modification**

#### 4.1.2.2 Initialization (Lines 37–45)

The function call to *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 allow things to stabilize after reset. The call to *DecoderReset()* does nothing and returns. This call is included to allow the same main program to be used for the $I^2C$, VIP, and VMI bus interfaces.

#### 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 was shown in Table 2. The global variables *g_nROMCodeSize* and *g_pROMCode* are initialized with the size of the selected microcode file (in bytes and including the sub-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 the 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: Power-Up Initialization()*

#### 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 $I^2C$, VIP, and VMI interfaces. The source code module *I2C5020.C* contains the version of *HandleDownload()* for the $I^2C$ 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 subaddress (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 (for I2C, this is always 2), and a pointer to the storage location of the subaddress and data byte. A 10 ms 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 6.

#### 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 100 ms 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 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              // subaddress, 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

### *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 time-out 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 time-out 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 2 ms.

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

This is the interrupt service routine (ISR) for the $I^2C$ 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 master receiver are described in Tables 7 and 8. The last step of interrupt handling 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 7. I2C Controller: Master Transmitter States**

| I2C CONTROLLER STATE | DEFINITION | NEXT ACTION TAKEN BY I2C ISR |
|---|---|---|
| **MASTER TRANSMITTER STATES** | | |
| 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 8. I2C Controller: Master Receiver States**

| I2C CONTROLLER STATE | DEFINITION | NEXT ACTION TAKEN BY I2C ISR |
|---|---|---|
| **MASTER RECEIVER STATES** | | |
| 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 a 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
                                            100 kBps */
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
49             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             /* 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 time-out 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 and a constant is loaded into the timer data registers (TL0 and TH0). Then 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 values (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) * T)

TH0, TL0 = 10000h – ((11059200 / 12) * 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 the timer is stopped. The timer mode is set for 16-bit counter with no prescaling and the timer reload value is written. The timer interrupt is enabled and given low (normal) priority. Then 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 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
   30           This function is an interrupt service routine for TIMER 0.  It should
   31           never be called by a C or assembly function.  It will be executed
   32           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
136             This function returns the number of timer ticks that have elapsed since
137             The specified count.
139             ---------------------------------------------------------------------
140             */
141
142             unsigned timer0_elapsed_count( unsigned start_tick )
143             {
144    1            return(current_tick() – start_tick);
145
146
147
148
149
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: I2C5020

This module contains the TVP5020-specific I2C routines.

### *4.10.1 Function: DecodeReset() (Lines 14–18)*

This function does nothing and returns. It is included for compatibility with code for other bus interfaces.

### *4.10.2 Function: HandleDownload() (Lines 20–24)*

This function is called to download the microcode to the TVP5020 program memory. Two parameters are passed to this function. The parameter *nCount* is the number of data bytes to write (plus 1 for the subaddress byte). The parameter *pInBuf* is a pointer to the data to be written. The first byte in the data must be the subaddress for microcode downloads (7Eh). For the I2C case, this function simply passes its parameters to the function *WriteTVP5020()*.

### *4.10.3 Function: WriteTVP5020() (Lines 26–38)*

This function is called to write to a TVP5020 register. Two parameters are passed to this function. The parameter *nData* is the number of data bytes to write (plus 1 for the subaddress byte). The parameter *pInBuf* is a pointer to the data to be written. The first byte in the data must be the subaddress. For the I2C case, except for microcode downloads, the function can only write to one register. In this case, the *nData* parameter is 2. The I2C function ‘start_i2c()’ is called to perform the I2C data transfer. The TVP5020 I2C device ID for writes is passed as a parameter. The number of bytes actually written (including the subaddress byte) is returned.

### *4.10.4 Function: ReadSwitch() (Lines 40–65)*

This function is TVP56000EVM-specific. It reads the logic levels of the DIP switches through the defined I/O pins of the P80C652. The result is packed into an 8-bit return value.

### *4.10.5 Function: ReadTVP5020() (Lines 67–89)*

This function can be called to read from a TVP5020 register. The function is not used in this program, but is included here for reference. Three parameters are passed to this function. The parameter ‘nLength’ is the number of data bytes to read. The parameter ‘pBuf’ is a pointer to the caller’s buffer where the read data is to be stored. The parameter ‘nSubAddr’ is the subaddress to read from.

For the I2C case, except for microcode uploads, only one register can be read. In this case, the *nData* parameter is 1. The I2C function ‘start_i2c()’ is called first, with the TVP5020 I2C device ID for writes, to write the subaddress value. Then, ‘start_i2c()’ is called again, with the TVP5020 I2C device ID for reads, to read the requested number of data bytes.

## 4.11 Header File: I2C5020.H

```
// I2C5020.H
//
// Header file for TVP5020 for I2C Bus
//
#define FALSE                   0
#define TRUE                    !FALSE

#define TVP5020_WRITE_DEVICE_ID         0xB8
#define TVP5020_READ_DEVICE_ID          0xB9
#define TVP5020_LOAD_SUB_ADDR           0x7E
#define BOARD_TVP56000EVM   5

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

## 4.12 Source File: I2C5020.C

```
DOS C51 COMPILER V5.10, COMPILATION OF MODULE I2C5020
OBJECT MODULE PLACED IN I2C5020.OBJ
COMPILER INVOKED BY: C:\C51\BIN\C51.EXE I2C5020.C DB OE OR LARGE

stmt level      source

   1            //
   2            // I2C5020.C
   3            //
   4            // Routines for TVP5020 using I2C Bus
   5            //
   6          #include ”I2C.h”
   7          #include ”I2C5020.h”
   8          #include ”Reg652.h”
   9
  10
  11            // Identify the board
  12            unsigned char g_nBoardID = BOARD_TVP56000EVM;
  13
  14            // Dummy function called from main() – Not needed for I2C Bus
  15            void DecoderReset( void )
  16            {
  17   1           return;
  18   1        }
  19
  20            void HandleDownload( unsigned nCount, unsigned char* pInBuf )
  21            {
  22   1           WriteTVP5020( nCount, pInBuf );
  23   1           return;
  24   1        }
  25
  26            unsigned WriteTVP5020(int nData, unsigned char *pBuf )
  27           {
  28   1           unsigned nRtn = 0;
  29   1
  30   1           // Initialize I2C bus
  31   1           initia_i2c();
  32   1
  33   1           // Perform the I2C Bus transaction
  34   1           // return actual number of bytes sent
  35   1           nRtn = start_i2c(TVP5020_WRITE_DEVICE_ID, nData, pBuf);
  36   1
  37   1           return( nRtn );
  38   1        }
  39
```

## 4.12 Source File: I2C5020.C (continued)

```
  40          unsigned char ReadSwitch( void )
  41          {
  42   1          unsigned char nVal = 0;
  43   1
  44   1         // See REG652.H for SW1...SW3 definitions
  45   1          SW1 = 1;     //MSB
  46   1          SW2 = 1;
  47   1          SW3 = 1;     //LSB
  48   1
  49   1          if ( SW1 == 1 )
  50   1          {
  51   2              nVal += 4;
  52   2          }
  53   1
  54   1          if ( SW2 == 1 )
  55   1          {
  56   2              nVal += 2;
  57   2          }
  58   1
  59   1          if ( SW3 == 1 )
  60   1          {
  61   2              nVal += 1;
  62   2          }
  63   1
  64   1          return( nVal );
  65   1      }
  66
  67          // Note: This is how to read a TVP5020 register via I2C
  68          #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 = subaddress to read

                  unsigned char* pSubAddr = &nSubAddr;

                  // Initialize I2C bus
                  initia_i2c();

                  // Write the subaddress
                  start_i2c( TVP5020_WRITE_DEVICE_ID, 1, pSubAddr);

                  // Read the data
                  start_i2c(TVP5020_READ_DEVICE_ID, nLength, pBuf);

                  return( 0u );
              }
              #endif
  89

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. The parameter *read_length* is the number of data bytes to read. The parameter *sub_addrs* is the subaddress to read from. The parameter *output_buf* is a pointer to the caller's buffer where the data read 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 subaddress 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. The parameter *write_length* is the number of data bytes to write (plus 1 for the subaddress byte). The 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 to setup. 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 code work on an EVM prototype board that uses VP1 instead of VP2. Normally, this *if* statement should be deleted.

## 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
#define BOARD_EVM3          3

#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 subaddress 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 subaddress 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                          subaddress  */
0x3A, /* subaddress                                 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                          subaddress  */
0x3A, /* subaddress                                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                          subaddress  */
0x3A, /* subaddress                                N/A     */
0x0F, /* F_CONTROL                                  3A     */
      /* RESERVED                                          */
                0x00, 0x00, 0x00, 0x00, 0x00,  /* 3B–3F  */
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                          subaddress  */
0x3A, /* subaddress                                 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  */  };
```