

Generating VGA signals with the TM1xxx



By Geert van der Heijden

status: release 1 (see future work)

Abstract

The Trimedia TM1xxx in combination with the SAA7167A can generate various VGA standard video signals and other interlaced and non-interlaced video signals. Besides the standard interlaced signals (PAL and NTSC), non-interlaced signals like 640x480 at 60 Hz are generated (VGA). The Video Out module of the TMxxx is able to generate video signals up to a pixel rate of 40M pixels/sec, using the flexible sample clock Direct Digital Synthesizer and the programmable image parameters in the Video Out module.

This application note describes in detail how the Video Out module can be programmed to generate non-interlaced signals and how the interconnection is between TM1xxx and SAA7167A.

Block schematic

Figure 1VGA interface block schematics.

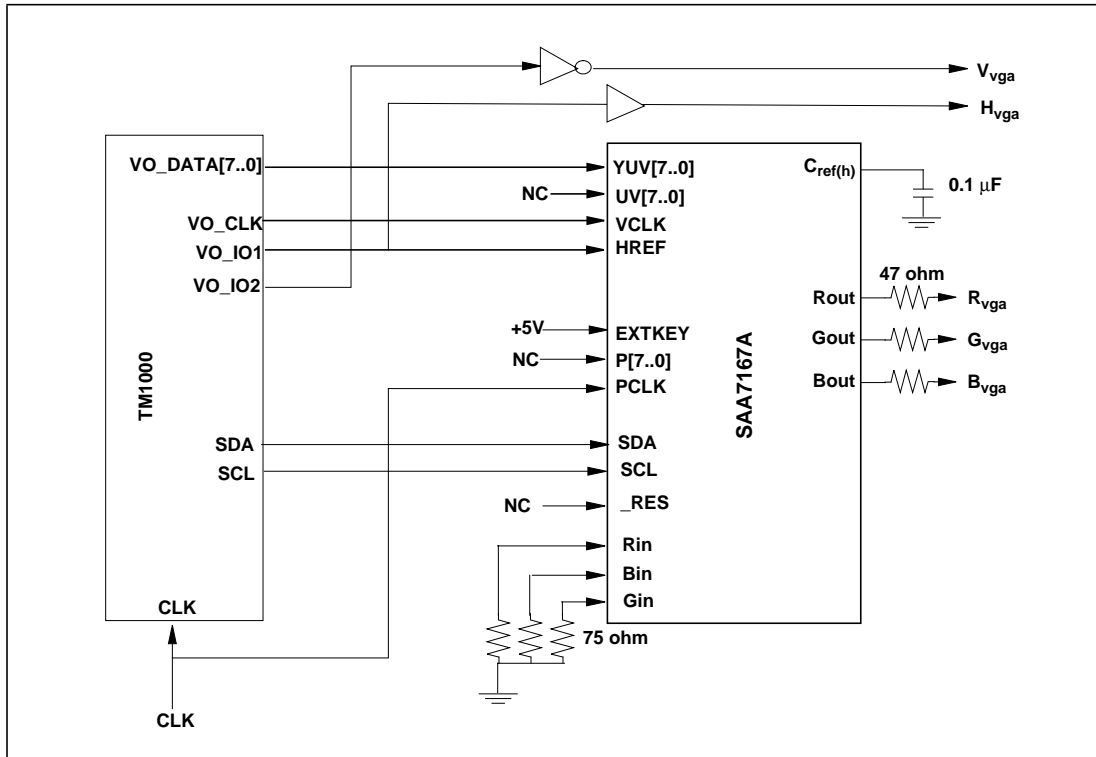


Figure 1 illustrated the connections between TM1000 and SAA7167A. The Video Out module of TM1000 generates the 8-bit video data signal (VO_DATA[7..0]), the video clock (VO_CLK) and the horizontal reference (HREF = VO_IO1) signal for the SAA7167A. The SAA7167A demultiplexes the video data, executes a YUV to RGB conversion and digital-to-analog conversion. The VO_IO1 and VO_IO2 signals are used as respectively horizontal and vertical synchronization signals. The VO_IO2 signal is converted to support the VGA (640x480) negative V synchronization signalling.

The SAA7167A is controlled via the IIC-bus by TM1000, which is illustrated in the example program in Appendix A.

It appears that there should be a clock signal available on PCLK in order to reach a correct operation of the IIC circuitry. If this is not the case the SDA line is pulled low by the SAA7167A which blocks the IIC communication. Because IIC communication has to be available from the beginning (the boot-up sequence) the best solution is to use the TM1000 clock oscillator signal also for PCLK.

The SAA7167A supports also color keying, with an external analog video signal (Rin, Gin and Bin) or under control of the PCLK, P[7..0] or EXTKEY signal. In this setup it is programmed in keying mode, with the analog signals tied to ground and key value zero. In this way the TM1000 YUV signal is passed through.

Using the SAA7167A it is also possible to mix the TM1000 video signals with a VGA signal on the analog video inputs of the SAA7167A.

The pixel frequency is limited by the maximum pixel frequency of TM1000 and the DA converters of SAA7167A, which is respectively 40M and 66M pixels/sec. The SAA7167A accepts video in different formats (YUV 4:2:2, 4:1:1, 2:1:1 and RGB 5:6:5) but only YUV 2:1:1 (is

equivalent to CCIR656) is used in this setup, otherwise additional hardware is needed to demultiplex the video data signals from TM1000.

For a detailed description of the SAA7167A see the datasheets.

VGA example program

An example program is included which generates VGA signals for 640 x 480. For programming the video out control registers the timing characteristics for the particular VGA mode have to be known. For VGA (640x480) these are:

- Resolution: 640 pixels and 480 lines
- Pixel frequency: 25.175 MHz
- Horizontal (H) frequency: 31.469 kHz
- Vertical (V) frequency: 59.940 Hz
- H/V polarity: both negative
- H-total: 31.778 μ s (800 pixels)
- H-video: 25.422 μ s (640 pixels)
- H-blank: 6.356 μ s (160 pixels)
- H-front: 0.636 μ s (16 pixels)
- H-sync: 3.813 μ s (96 pixels)
- H-back: 1.907 μ s (48 pixels)
- V-total: 16.683 ms (525 lines)
- V-video: 15.253 ms (480 lines)
- V-blank: 1.430 ms (45 lines)
- V-front: 0.318 ms (10 lines)
- V-sync: 0.064 ms (2 lines)
- V-back: 1.049 ms (33 lines)

These parameters can be fully supported by the TM1000 video out module, its enhanced programmability enables deviations from these settings for non-VGA standards related timings. A clock frequency up to 80 MHz (equal to a pixel rate of 40M pixels/sec) is supported, for both interlaced and non-interlaced video formats.

The example program is included in Appendix A, it illustrates the register setting of the video out module of TM1000 and the settings of the control register of the SAA7167A.

The example also illustrates that the Video Out module is forced in non-interlaced mode, by programming Field 2 start beyond the number of lines and not to programming FRAME 2 START of the VO_FRAME register to 0, programming it to 0 will generate no VO_IO2 signals to indicate the start of a frame.

Future Work

Currently a prototype has been built with a TM1000 running at 80 MHz and a SAA7167A. This system has been tested with a VO_CLK at 70 MHz (pixel rate is 35M pixels/sec), it should also be tested on a 100 MHz (or higher frequency) system to evaluate the maximum frequency.

The prototype system contains programmable logic for additional evaluation, like demultiplexing to achieve a higher frequency on the SAA7167A. The results of these tests will illustrate the maximum achievable VGA standard.

Appendix A

File VGA.c

```
/*  
* (c) Copyright 1997 Philips Electronics N.V.
```

Generating VGA signals with the TM1000

```
* All rights reserved. Reproduction in whole or in part is
* prohibited without the written consent of the copyright owner.
*
* Philips Semiconductors TriMedia
* Eindhoven, The Netherlands
*
* Name      : vga.c
*
* Author    : Geert van der Heijden
*
* Date      : July 28, 1997
*
* Function   : Control of video output module.
*
* Description : VGA video test program
*
* History    :
* 97-07-28 : Initial version
*
*/

/* Standard include files */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
/* TM1 specific include files. */
#include <tm1/mmio.h>
#include <ops/custom_defs.h>
/* TM1 device lib include files. */
#include <tm1/tmVO.h>
#include <tm1/tmIIC.h>
#include <tm1/tmBoard.h>
#include <tm1ib/HAL.h>

#include "vga.h"

/* Global vo control structure */
static voInfo voIf;

/***** Buffer Allocation Routines *****/
unsigned long
alignedAlloc(int bufSz)
{
    unsigned long  retval;

    retval = (unsigned long) malloc(bufSz + 63);
    if (!retval)
        return (0);

    return ((retval + 63) & 0xfffffc0);
}

void vo_ISR()
{
    #pragma TCS_handler
}

void vo_init(unsigned char *next_y_ptr,
             unsigned char *next_u_ptr,
             unsigned char *next_v_ptr,
             int width,
             int height,
```

```

        int stride_uv,
        int stride_y)
{
    unsigned int error = 0;

    /* connect video encoder */
    /* change later to use *encoderConfig */
    /* switch i/o controller on IREF+ board in software */
    if((error = iicWriteReg(0x72, -1, 0x7b)) != 0) {
        fprintf(stderr, "Failed to write to I/O expander %x\n", error);
        exit(1);
    }

    /* configure SAA7167A */
    if((error = iicWriteReg(SAA7167A_W, 0x00, 0xfe)) != 0) {
        fprintf(stderr,
            "Failed to write subaddress 0 to SAA7167A %x\n", error);
        exit(1);
    }
    if((error = iicWriteReg(SAA7167A_W, 0x01, 0x00)) != 0) {
        fprintf(stderr,
            "Failed to write subaddress 1 to SAA7167A %x\n", error);
        exit(1);
    }
    if((error = iicWriteReg(SAA7167A_W, 0x02, 0x00)) != 0) {
        fprintf(stderr,
            "Failed to write subaddress 2 to SAA7167A %x\n", error);
        exit(1);
    }
    /* initialize global voIf structure */
    memset((char *)&voIf, 0, sizeof(voInfo));

    /* set video out registers */
    voIf.RClock = (unsigned int)((FREQ/(3.0 *
        (double)HAL_get_clock_frequency())
        * pow(2.0, 32));
    voIf.RFrame = VO_NI_FIELDPRESET | VO_NI_FIELD2START |
        VO_NI_FRAMELENGTH;
    voIf.RField = VO_NI_F2OLAP | VO_NI_F1OLAP |
        VO_NI_F2VIDEOLINE |
        VO_NI_F1VIDEOLINE;
    voIf.RLine = VO_NI_VIDEOPIXELSTART | VO_NI_FRAMEWIDTH;
    voIf.RImage = (height << VO_NI_SHIFT_IMAGEHEIGHT) | width;
    voIf.RYThr = VO_NI_YTHRESHOLD | VO_NI_IMAGEVOFF |
        VO_NI_IMAGEHOFF;
    voIf.ROIStart = 0;
    voIf.ROlhw = 0;

    /* Set yuv pointers in memory and some additional VO registers */
    voIf.RYadd = (unsigned long)next_y_ptr;
    voIf.RUadd = (unsigned long)next_u_ptr;
    voIf.RVadd = (unsigned long)next_v_ptr;
    voIf.ROladd = 0;
    voIf.Rvud = (stride_uv << 16) | stride_uv;
    voIf.Ryold = VO_NI_OLOFFSET | stride_y;
    voIf.RCtl = VO_NI_CTL;
    voIf.voISR = vo_ISR;

    if((error = voOpen(&voIf)) != VO_RETURN_OK) {
        printf("vo_init: Failed to open video out error: %x\n", error);
        exit(1);
    }
}

```

Generating VGA signals with the TM1000

```
if((error = voSetRegs(voIf.userID, &voIf)) != VO_RETURN_OK) {
    fprintf(stderr,
            "vo_init: Failed to SetRegs video out error: %x\n",
            error);
    exit(1);
}
}
```

```
main(int argc, char *argv[])
{
    unsigned char *ybuffer, *ubuffer, *vbuffer;
    int x, y, old_y = 0, new_y, frame_cnt=0;
    int l = 0;
    unsigned int id;

    printf("VGA output on TriMedia Test program\n");

    boardGetID(&id);
    printf("HAL reports ");
    switch (HAL_get_processor_version() )
    {
        case HAL_PROCESSOR_UNKNOWN:
            printf("unknown processor version ");
            break;
        case HAL_PROCESSOR_CTC:
            printf("CTC processor.\n ");
            printf("IICtest not supported on CTC\n");
            printf("HAL reports CTC ");
            break;
        case HAL_PROCESSOR_TM1:
            printf("vanilla TM1 processor ");
            break;
        case HAL_PROCESSOR_TM1S:
            printf("TM1s processor ");
            break;
    }
    printf("running at %3.2f MHz on ", 0.000001 *
           (float)HAL_get_clock_frequency());

    switch (id)
    {
        case BOARD_VERSION_UNKNOWN:
            printf("unknown board type\n");
            break;
        case BOARD_VERSION_PHILIPS_CTC:
            printf("CTC board: \n");
            break;
        case BOARD_VERSION_PHILIPS_TM1_DEBUG:
            printf("TM1 Debug board\n");
            break;
        case BOARD_VERSION_PHILIPS_TM1_IREF:
            printf("TM1 IREF board\n");
            break;
        default:
            printf("Unrecognized board type.\n");
            break;
    }
    printf("\n");

    ybuffer = (unsigned char *)alignedAlloc(NI_SIZE);
```

```

ubuffer = (unsigned char *)alignedAlloc(NI_SIZE/2);
vbuffer = (unsigned char *)alignedAlloc(NI_SIZE/2);
vo_init(ybuffer, ubuffer, vbuffer,
        NI_WIDTH, NI_HEIGHT, NI_WIDTH/2, NI_WIDTH);

/* clear Y, U, V buffer */
for(y=0;y<NI_SIZE;y++) {
ybuffer[y] = 112;
if((y%40 == 0) & (y != 0)) {
ybuffer[y-1] = 0xff;
ybuffer[y] = 0xff;
}
}
for(y=0;y<NI_SIZE;y+=40*NI_WIDTH) {
for(x=0;x<NI_WIDTH;x++) {
ybuffer[y + x] = 0xff;
}
}
for(y=0;y<NI_SIZE/2;y++) {
ubuffer[y] = 58;
vbuffer[y] = 72;
}
printf("Filled the y framebuffer \n");

for(y=0;y<NI_HEIGHT;y++) {
COPYBACK((char*)(ybuffer+y*NI_WIDTH), NI_WIDTH >> 6);
COPYBACK((char*)(ubuffer+y*NI_WIDTH/2), NI_WIDTH >> 7);
COPYBACK((char*)(vbuffer+y*NI_WIDTH/2), NI_WIDTH >> 7);
}

while(1) {
}
}

```

File VGA.h

```

/*
* (c) Copyright 1997 Philips Electronics N.V.
* All rights reserved. Reproduction in whole or in part is
* prohibited without the written consent of the copyright owner.
*
* Philips Semiconductors TriMedia
* Eindhoven, The Netherlands
*
* Name      : VGA.h
*
* Author    : Geert van der Heijden
*
* Date      : July 28, 1997
*
* Function  : Header file for vidout.c
*
*/

#ifndef VGA_H_INCLUDED
#define VGA_H_INCLUDED

```

Generating VGA signals with the TM1000

```
#define NI_WIDTH          (640)
#define NI_HEIGHT        (480)
#define NI_SIZE          (NI_WIDTH * NI_HEIGHT)
#define FREQ              25.175e06L
#define VO_NI_FIELDPRESET (1 << 24)
#define VO_NI_FIELD2START (524 << 12)
#define VO_NI_FRAMELENGTH (525)
#define VO_NI_F2OLAP      (2 << 28)
#define VO_NI_F1OLAP      (0x0e << 24)
#define VO_NI_F2VIDEOLINE (530 << 12)
#define VO_NI_F1VIDEOLINE (34)
#define VO_NI_VIDEOPIXELSTART (96 << 12)
#define VO_NI_FRAMEWIDTH  (800)
#define VO_NI_SHIFT_IMAGEHEIGHT (12)
#define VO_NI_YTHRESHOLD  (0 << 20)
#define VO_NI_IMAGEVOFF   (0 << 8)
#define VO_NI_IMAGEHOFF  (32)
#define VO_NI_UOFFSET     (NI_WIDTH/2 << 16)
#define VO_NI_VOFFSET     (NI_WIDTH/2)
#define VO_NI_OLOFFSET    (0)
#define VO_NI_YOFFSET     (NI_WIDTH)
#define VO_NI_CTL         (0x42f00001)

#define SAA7167A_W        (0xbe)

#endif /* VGA_H_INCLUDED */
```