利用 BSP-15 DSP 处理器上 GPDP 实现 VGA/XGA 信号采集

关键词：BSP-15 DSP GPDP iMMediaTools YUV RGB

随着视频会议系统应用的扩展，2003 年 7 月，ITU 批准了 H.239 标准。基于此项技术，视频会议厂商需要得到双流视频服务。而作为双流的新增那一路媒体流，需要的是类似 VGA 信号的高清晰信源输入。目前基于 BSP-15 的视频编码卡应用中，并不能直接支持 VGA 信号的直接输入，但我们可以借助于其通用的输入/出口 GPDP 的输入接口来实现。

Using GPDP on BSP-15 DSP to achieve VGA/XGA analog signal capture

Key word: BSP-15 DSP GPDP iMMediaTools YUV RGB

By the Video conference application to be spread, the H.239 standard be passed by ITU, on July 2003. Based on this technology, the Two-Media-Stream functions become necessary. It always use the VGA or other High-Definition signal as input. Now in the video encode device, such as BSP-15 card, does not support the VGA as input signal directly, but we can achieve it by the GPDP, a general purpose data input/output port.

随着视频会议系统应用的扩展，2003 年 7 月，ITU 批准了 H.239 标准，基于此项技术，视频用户需要得到双流视频服务。而作为双流的另外一路媒体流，需要的是类似 VGA 信号的高清晰信源输入。目前基于 BSP-15 的视频编码卡应用中，并不能直接支持 VGA 信号的直接输入，但我们可以借助于其通用的输入/出口 GPDP 的输入接口来实现。

BSP-15 是美国 EQUATOR（目前已被 Pixelworks 并购）公司研制的，一款高性能媒体信号处理器。BSP-15 是专门为视频应用而设计的高效 DSP，以高度集成的单芯片方案来满足宽带产品的各种需要。BSP-15 在 400MHz 的时钟频率下，其处理能力为 40 GOPS(每秒 400 亿次整数运算)，处理速度相当于奔腾 III 的 8.5 倍。同时作为一款通用的 DSP，BSP-15 实现了完全 C 语言编程，从而可方便地将在 PC 架构下验证过的专用算法轻松地移植到该系统上来。结合 iMMediaTools 软件开发环境及其提供的专用视音频库，BSP-15 可以为数字视/音频应用，特别是视频会议系统上的应用提供更丰富的解决方案。目前在 BSP-15 上已经实现的音视频编解码算法有：JPEG 、MPEG1/2/4、H.263+/H.264、WM9、G .72x 和 MP3 等专用算法。与同类专用 DSP，例如 TI 的 DM642-600MHZ 和 ADI 的 Blackfin BF533-600MHZ 相比，其具有更高的运算速度，并且其开发工具自带 Linux 和 eCOS 操作系统，无须用户为此再次付费。其缺点为功耗偏大。尽管如此，凭借其卓越的音视频的性能表现，在多媒体通讯应用中，BSP-15 已占据了相当的市场份额。

BSP-15 其丰富的外设接口，也是它能够得到广泛应用的原因之一。它拥有两个独立的 ITU-656 视频输入/出接口、通用输入/出接口 GPDP、最高支持到 128MB 的 SDRAM 接口、音频 IEC958/IIS、通讯控制 IIC、模拟 RGB 接口最高支持 1280×1024 的分辨率输出驱动 VGA、ROM/FLASH 控制器和 PCI V2.2 接口。

下图为 BSP-15 的结构框图：

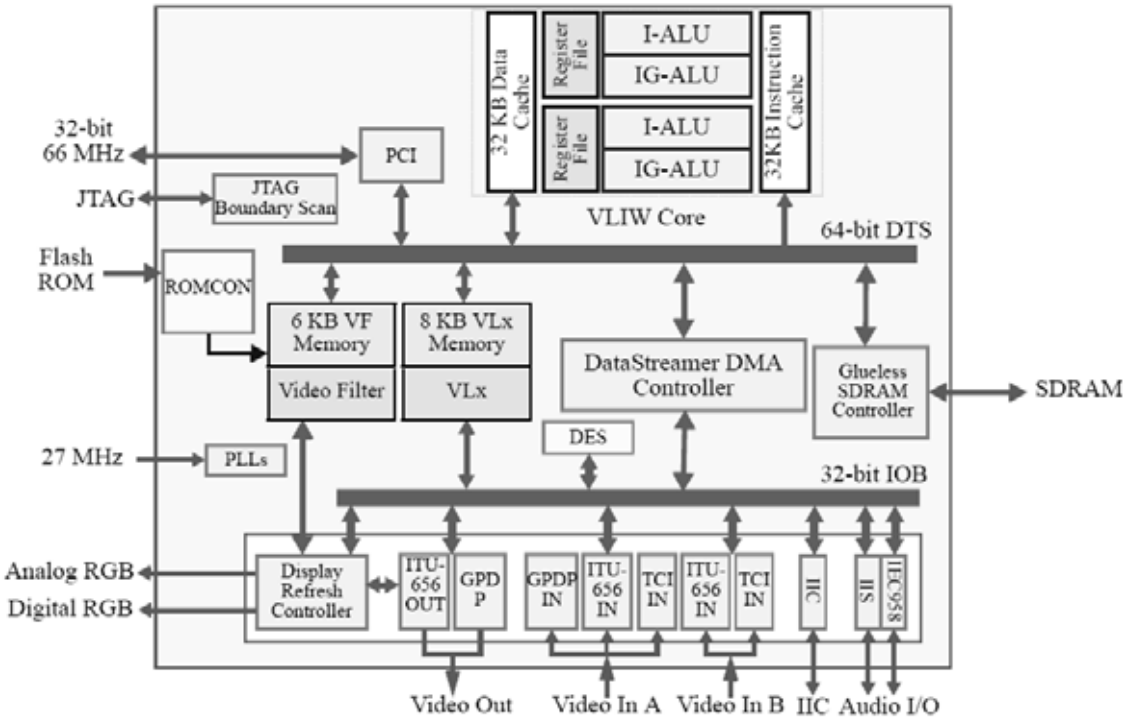

图 1.1 BSP-15 结构框图

在电路上，GPDP 的输入与第一路视频输入为引脚复用，这样我们可选用第二路视频输入为 NTSC/PAL 输入，第一路则专用于 VGA 信号输入。在 EQUATOR 提供 BSP-15 的技术资料中显示，其 GPDP 最高可以支持 60MB/S 的传输速率。我们可以来推算一下，传送一个幅面为 1024×768 像素、帧率为 15 帧每秒的 RGB24 媒体流所需要的传输带宽是多少？需要的带宽：B=1024×768×3(R、G、B) ×15（帧）= 33.75MB/S。如果传送的为 YUV422 或 YUV420 压缩过的媒体流，则需要的带宽会更小，而对于第二路流的帧率，在视频会议系统中，15 帧已经可以满足要求，所以 GPDP 60MB/S 的带宽足够用来传送高清晰的 VGA 信号了。

下面，我们来设计所需要的 VGA 采集系统。该系统为用于完成 VGA（XGA）模拟视频信号采集、数字化并通过 GPDP 接口将数字信号传送到 BSP-15 上的一套设计，VGA 数字信号交由 BSP-15 上的上层应用程序实现压缩和转换等功能。可将 VGA 视频采集实现设计划分为两大部分，即：硬件实现层（VGA 采集硬件及 FPGA 设计）、驱动层（BSP-15 媒体驱动程序）。

一）硬件实现层

硬件设计是整个系统的基础工程。如图 1.2 所示，前端 VGA 模拟信号的采集变换采用 Philips 公司的 TDA8751，这是一款高性能的 10bit 采样精度的三路 A/D 转换电路，用于 VGA 信号采集可实现 1024*768（XGA 75HZ）的采样精度。FPGA 完成图像的帧缓存功能，实现上传图像数据的完整性。DSP 部分既 BSP-15，实现 VGA 图像的接收和压缩等处理，为 PC 端高层应用提供相应的编码格式。

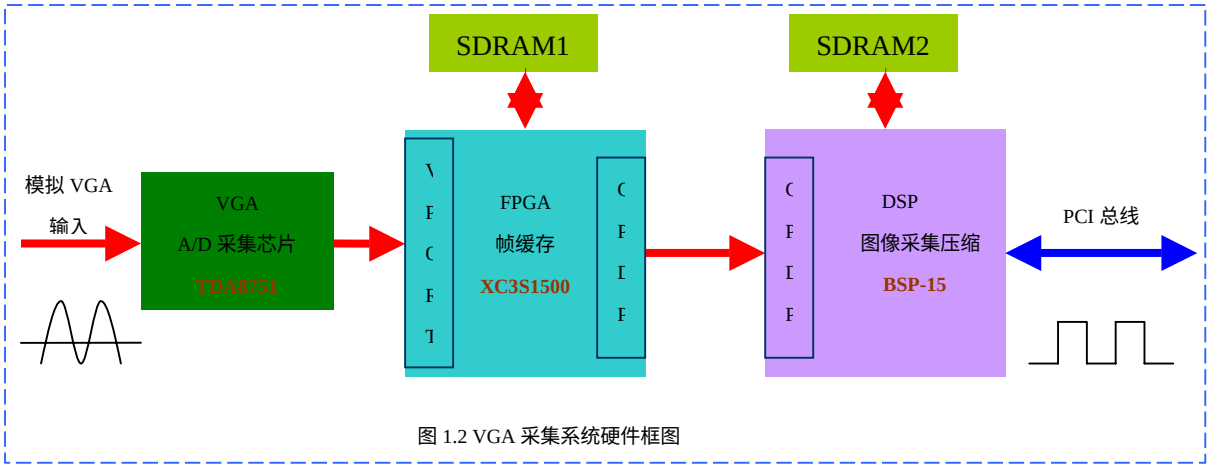

图 1.2 VGA 采集系统硬件框图

其中 FPGA 的设计较为重要和复杂。Xilinx 公司的 Spartan-3 系列是采用最先进的 Virtex-II FPGA 器件架构，并采用 90nm 技术，具有极高的性价比及单位成本内有效 I/O 管脚最多等优点。该设计中选用的 XC3S1500-FG456，内部时钟可达 326MHz，可提供 1，500，000 个系统门及 29，952 个逻辑单元，32 个专用乘法器及 32×18Kb 块存储资源。无论从速度和 RAM 的容量上，都足以满足 VGA 采集帧缓存的要求。FPGA 的设计可细分为五个模块，其分别是：GPDP 通信和管理模块、采集和采集管理(VPORT)模块、帧存和帧存管理模块、SDRAM 接口模块及逻辑和时序控制模块。

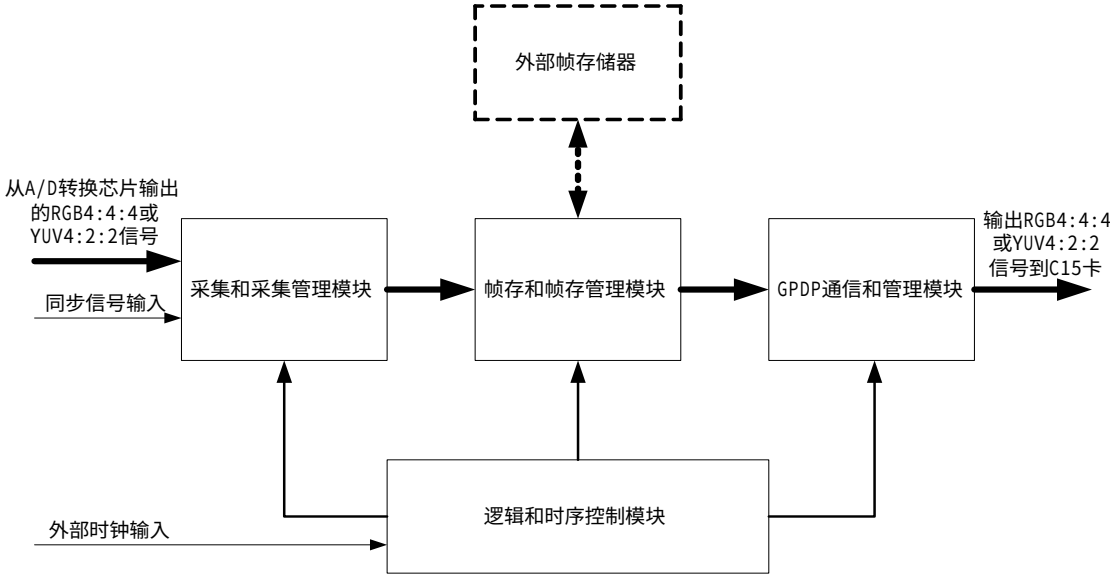

图 1.3 FPGA 逻辑实现的原理框图

其中 GPDP 通信和管理模块是实现与 BSP-15 的 GPDP 互联的功能模块，也是整个 VGA 数据传输通路中的关键接口部分，该部分实现的优与劣会直接会影响到 VGA 信号上传的实时性等性能问题。下面我们来重点介绍 GPDP 通信和管理模块的实现过程。

BSP-15 的 GPDP 接口共有 22 个信号引脚，其中输入输出分别各占 11 脚。输入引脚分别是： Data_in[7:0](OUT)、Rcv_clk(IN)、Rcv_req(IN)、Rcv_ack(OUT)；输出引脚分别是：Data_out[7:0](IN)、Xmt_clk(IN)、Xmt_req(OUT)、Xmt_ack(IN)。与之对应的 FPGA 的 GPDP 通信和管理模块引脚定义如下图所示，实现 FPGA 与 BSP-15 的硬件互联。连接示意图如下：

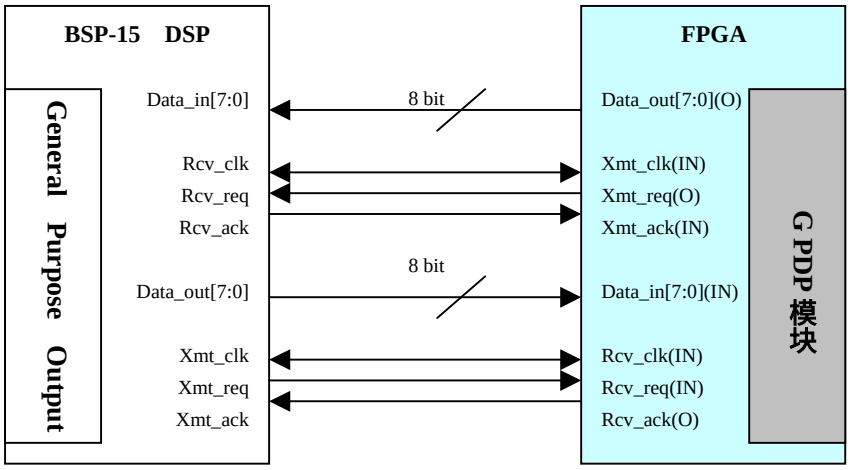

图 1.4 FPGA 于 BSP-15 的 GPDP 接口连接示意图

VGA 上的 GPDP 接口与 BSP-15 通讯为多字节顺序读和多字节写模式。FPGA 上 GPDP 部分的设计目标是实现，将帧存和帧存管理模块以帧为单位发出的 VGA 信号，向 BSP-15 的 GPDP 发送，并接收来自 BSP-15 的向状态控制 REG 的读写操作。所有 GPDP 上的通讯，均由 BSP-15 发出访问控制。

GPDP 通信和管理模块其功能描述及原理框图如下：

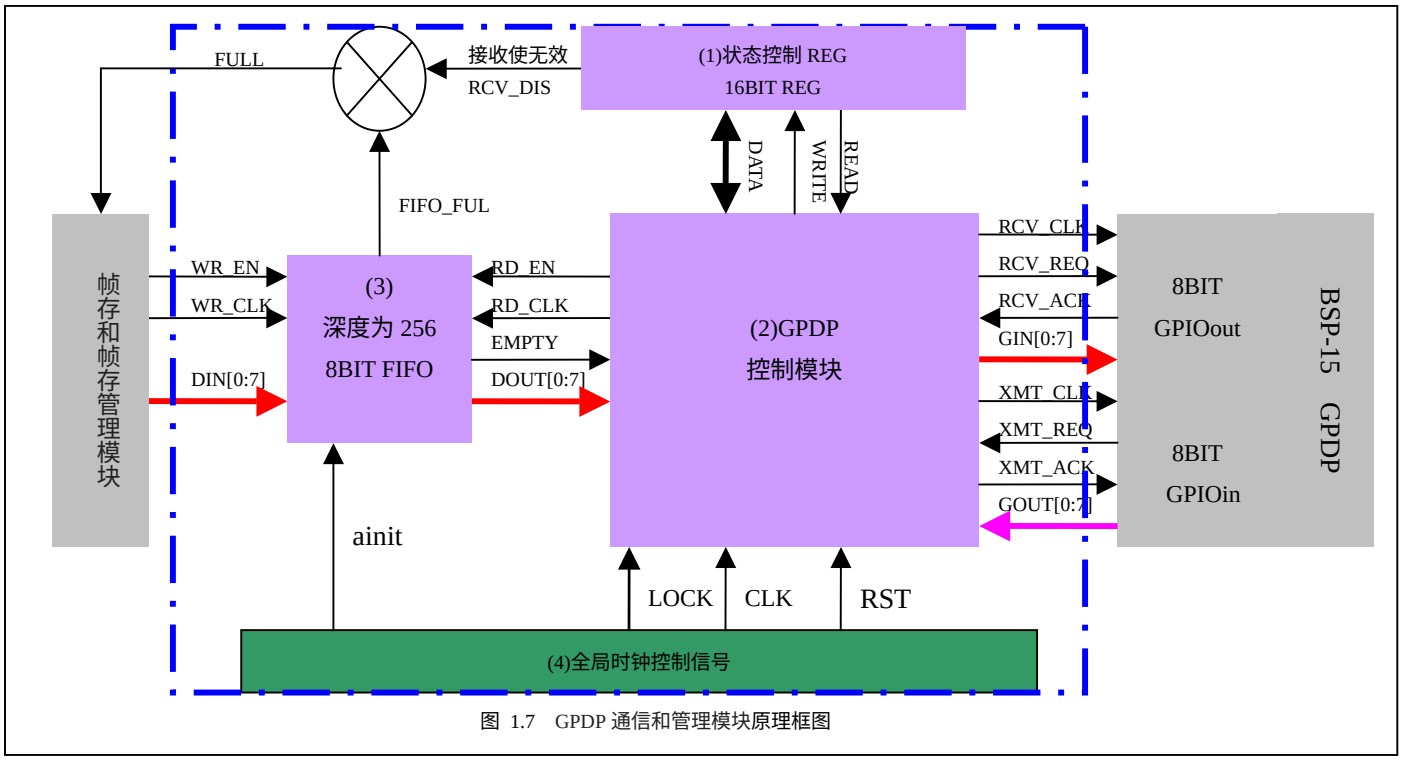

图 1.7 GPDP 通信和管理模块原理框图

在该模块中：

（1）状态控制 REG 为用于 BSP-15 在与 FPGA 通讯的读写寄存器 REG，对应的 REG 的地址和内容如下：

地址 0X00：SYS_CONTROL_REG—16bit（R/W）--系统控制寄存器

Bit12—Bit15	Bit8—Bit11	Bit4—Bit7	Bit0—Bit3
软复位		帧率	

地址 0X01：SYS_FRAME_NUM—8bit（R/W）--系统帧计数寄存器

Bit12—Bit15	Bit8—Bit11	Bit4—Bit7	Bit0—Bit3
VGA 状态		帧数	

地址 0X02：SYS_RESOLUTION_ROW—16bit（R）--系统解析度寄存器

Bit12—Bit15	Bit8—Bit11	Bit4—Bit7	Bit0—Bit3
解析度行数目			

地址 0X03：SYS_RESOLUTION_COL—16bit（R）

Bit12—Bit15	Bit8—Bit11	Bit4—Bit7	Bit0—Bit3
解析度列数目			

地址 0XAA：RCV_DIS—8bit（W）

Bit4—Bit7	Bit0—Bit3
RCV DIS 控制位	

（2）GPDP 控制模块：在全局时钟控制下，完成对状态寄存器和 FIFO 的读写操作，是整个 GPDP 模块的核心环节。

（3）256 深 8BIT 宽的数据 FIFO：用于缓存从帧存和帧存管理模块推出的 VGA 数据。

（4）全局时钟控制信号：提供整个 GPDP 模块的时钟和复位信号。

模块化的设计方法为当前比较流行的可编程语言的设计思路，特别是在 FPGA 这样的大规模可编程器件中，给设计和调试带来极大的灵活性及方便。作为整个 GPDP 模块的核心部件，GPDP 控制模块实现了 BSP-15 的 GPDP 接口与 FPGA 内部的 FIFO 及寄存器之间的数据交互，从而实现

对 VGA 信号的采集和上传等工作。一次完整的 VGA 采集过程是这样的：

1、 首先由 BSP-15 发出 VGA 采集命令，由 BSP-15 发出写状态控制 REG （0XAA）非零命令；

2、 然后使 RCV_DIS 无效（0），使 8BIG FIFO 处于接收数据状态；

3、 同时完成 DOUT[0：7]=>GIN[0：7]和 RD_EN=1 操作；

4、 当 EMPTY 无效的时候，完成 CLK->RCV_CLK；CLK->RD_CLK；操作；

5、 当 EMPTY 有效的时候，RCV_CLK RD_CLK 保持低；

6、 采集结束，由 BSP-15 发出写写状态控制 REG （0XAA）零命令，使 RCV_DIS 有效（1）停止 VGA 的数据采集工作。

GPDP 模块对 C15-2U 提供标准 GPDP 通讯协议，对 VGA 上的 FPGA 中央控制单元提供标准的 8BIT FIFO 入接口。中央控制单元通过判断 FULL 信号，来决定是否启动 VGA 数据传送。全局控制提供 54MHz 的 CLK 和全局复位信号 RST，实现全局同步。完整的 VHDL 源代码见附件《GPDP_transfer.vhd》。

```
------------------------------------------------------------------
--Copy Right (C) 2005-2008 Firstlink LTD.
--Files name:        GPDP_transfer.vhd
--Version:          V1.0
--Project:          VGA 采集卡项目
--Writer:           fengcr
--Creat:            2005-02-01
--Function:         用于 VGA 采集卡上的 FPGA-XC3S1500-456 上的 GPDP 部分
--                        该部分有对内和对外两个接口:
--                  1.对外,主要完成与 BSP-15 GPDP 接口的通讯,完成外部的命令接收和 VGA 信号的输出.
--                  2.对内部,通过 32bit 宽的 FIFO 与控制单元接口,负责接收来自 DDRAM 中的 VGA 数据.
--updata:           2005-03-08
--             根据更新后的<<VGA 采集卡 FPGA 模块接口文档.doc>>文档,实现新的内部接口定义.
------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
Library spartan3;
use spartan3.components.all;
-- pragma translate_off
library UNISIM;
use UNISIM.VComponents.all;
-- pragma translate_on
entity gpdp_transfer is
    Port (
```

```
        --/*复位输入*/
             RESET_IN :            in std_logic;

           --/*时钟输入*/
           GPDP_CLK_IN :           in std_logic; --54MHz
        FIFO_CLK_IN :              in std_logic; --13.5Mhz

           --/*DCM 时钟锁定标志*/
        VGA_DCM_LOCKED:       in std_logic;

           --/*传输 FIFO 相关接口*/
           DIN :                        in std_logic_vector(31 downto 0);
           WR_EN :                 in std_logic;
           WR_CLK :                     in std_logic;
           FULL :                  out std_logic;
           AINIT :                 in std_logic;
        DATA_COUNT_WR:        out STD_LOGIC_VECTOR(4 downto 0);
        --/*C15 的 GPDP 相关接口*/
           GPDP_RCV_CLK :          out std_logic;
           GPDP_RCV_REQ :          out std_logic;
           GPDP_RCV_ACK :          in std_logic;
           GPDP_IN_DATA :               out std_logic_vector(7 downto 0);
           GPDP_XMT_CLK :          out std_logic;
           GPDP_XMT_REQ :          in std_logic;
           GPDP_XMT_ACK :          out std_logic;
           GPDP_OUT_DATA :         in std_logic_vector(7 downto 0);

        -- /*FIFO 数据有效标志*/
        FIFO_DATA_VALID:           in STD_LOGIC;

        --/*VGA 信号输入标志*/
           VGA_INPUT_FLAG:         in std_logic;

        --/*C15 控制输出接口*/
           SYS_CONTROL_INTERFACE :       out std_logic_vector(15 downto 0);

        --/*GPDP 模块测试输出接口*/
           TEST_GPDP :                   out std_logic_vector(7 downto 0)  );
end gpdp_transfer;
architecture Behavioral of gpdp_transfer is
--CoreGen FIFO32BIT
component fifo32bit
    port (
    din:           IN std_logic_VECTOR(31 downto 0);
    wr_en:         IN std_logic;
    wr_clk:        IN std_logic;
    rd_en:         IN std_logic;
    rd_clk:        IN std_logic;
    ainit:         IN std_logic;
    dout:          OUT std_logic_VECTOR(31 downto 0);
    full:          OUT std_logic;
    empty:         OUT std_logic;
    wr_count: OUT std_logic_VECTOR(4 downto 0));
end component;
--Gloabl clock buf with Enable port
attribute syn_black_box : boolean;
attribute syn_black_box of fifo32bit : component is true;
component BUFG
```

```
    port (
       I :          in std_logic;
       O :          out std_logic
       );
end component;
component BUFGCE
    port (
       I :          in std_logic;
    CE :       in std_logic;
       O :          out std_logic
       );
end component;
--GPDP Read command State machine define
type gpdp_wr_state is (wr_st0,wr_st1,wr_st2,wr_st3,wr_st4,wr_st5);
signal wr_current_state : gpdp_wr_state;
--Gpdp command Reg
signal cmd_01:      std_logic_VECTOR(7 downto 0);
signal cmd_02:      std_logic_VECTOR(7 downto 0);
signal cmd_03:      std_logic_VECTOR(7 downto 0);
signal cmd_04:      std_logic_VECTOR(7 downto 0);
signal cmd_05:      std_logic_VECTOR(15 downto 0);
signal cmd_06:      std_logic_VECTOR(7 downto 0);

--VGA data recieve Disable signal
signal rcv_dis:     std_logic;
signal rcv_dis_reg: std_logic_VECTOR(7 downto 0);

--FIFO32BIT read signal(rd_en,rd_clk,dout,rd_count,empty,fifo_full)
signal rd_clk:      std_logic;
signal rd_count:     std_logic_VECTOR(4 downto 0);
signal rd_en:       std_logic;
signal fifo_full:   std_logic;
signal dout:        std_logic_VECTOR(31 downto 0);
signal empty:       std_logic;

signal dout_reg :   std_logic_VECTOR(31 downto 0);
signal send_count :      std_logic_VECTOR(2 downto 0);

signal rd_en_reg :  std_logic;
signal xmt_start:   std_logic;
signal read_reg_start:   std_logic;
signal read_data_start:  std_logic;
signal WR_CLK_BUF:  std_logic;
signal send_start:       std_logic;
signal send_start_reg:   std_logic;
signal send_start_reg1:  std_logic;

signal GPDP_RCV_REQ_reg1 :     std_logic;
signal GPDP_RCV_REQ_reg2 :     std_logic;
signal GPDP_RCV_REQ_reg3 :     std_logic;
signal GPDP_RCV_REQ_reg4 :     std_logic;
signal GPDP_RCV_REQ_reg5 :     std_logic;

signal GPDP_RCV_CLK_reg :      std_logic;
signal GPDP_RCV_REQ_reg :      std_logic;
signal GPDP_XMT_CLK_reg :      std_logic;
signal SYS_CONTROL_REG_reg :   std_logic_vector(15 downto 0);
begin
```

```
----------------------------------program start-------------------------------------------------
   GPDP_RCV_CLK <= (not GPDP_RCV_CLK_reg) and GPDP_RCV_REQ_reg and send_start;
   GPDP_RCV_REQ <= GPDP_RCV_REQ_reg;

   GPDP_XMT_CLK <= GPDP_XMT_CLK_reg;
   SYS_CONTROL_INTERFACE <= SYS_CONTROL_REG_reg;

   bufg1:     BUFG port map(I => GPDP_CLK_IN, O => GPDP_XMT_CLK_reg);

   --ask RCV_ACK signal to BSP-15
   GPDP_XMT_ACK <= GPDP_XMT_REQ;

----------------------------------GPDP DATA IN-------------------------------------------------
   --Gpdp write to FPGA state machine
   process(RESET_IN,GPDP_XMT_CLK_reg,GPDP_XMT_REQ)
   begin
        if(RESET_IN = '1') then
             wr_current_state <= wr_st0;
        elsif rising_edge(GPDP_XMT_CLK_reg) then
             case wr_current_state is
                  when wr_st0 =>
                       if(GPDP_XMT_REQ ='1' ) then
                            wr_current_state <= wr_st1;
                       else
                            wr_current_state <= wr_st0;
                       end if;
                  when wr_st1 =>
                       if(GPDP_XMT_REQ ='1' ) then
                            wr_current_state <= wr_st2;
                       else
                            wr_current_state <= wr_st0;
                       end if;
                  when wr_st2 =>
                       if(GPDP_XMT_REQ ='1' ) then
                            wr_current_state <= wr_st3;
                       else
                            wr_current_state <= wr_st0;
                       end if;
                  when wr_st3 =>
                       if(GPDP_XMT_REQ ='1' ) then
                            wr_current_state <= wr_st4;
                       else
                            wr_current_state <= wr_st0;
                       end if;
                  when wr_st4 =>
                       wr_current_state <= wr_st5;
                  when wr_st5 =>
                       wr_current_state <= wr_st0;
                  when others =>
                       wr_current_state <= wr_st0;
             end case;
        end if;
   end process;
-----------------------------------------------------------
   process(GPDP_XMT_CLK_reg)
   begin
        if rising_edge(GPDP_XMT_CLK_reg) then
             --rcv disable untill get rcv command
```

```
            if(rcv_dis_reg /= "00000001") then
                  rcv_dis <= '1';   --disable vga send
            else
                  rcv_dis <= '0';   --require start vga send
            end if;
      end if;
   end process;
---------------------------------GPDP DATA OUT-----------------------------------------------
   --FULL is cleared only when fifo not full and asked to send VGA data
   FULL <= rcv_dis or fifo_full ;
   --generate GPDP read clock
   bufg3:     BUFG port map(I => GPDP_CLK_IN, O => GPDP_RCV_CLK_reg);
   --generate GPDP send data req signal
   GPDP_RCV_REQ_reg1 <= (not rcv_dis) and (not empty);
   process(rd_clk)
   begin
      if rising_edge(rd_clk) then
            if RESET_IN = '1' then
                  GPDP_RCV_REQ_reg2 <= '0';
            else
                  GPDP_RCV_REQ_reg2 <= GPDP_RCV_REQ_reg1;
                  GPDP_RCV_REQ_reg3 <= GPDP_RCV_REQ_reg2;
                  GPDP_RCV_REQ_reg4 <= GPDP_RCV_REQ_reg3;
                  GPDP_RCV_REQ_reg5 <= GPDP_RCV_REQ_reg4;
            end if;
      end if;
   end process;
   GPDP_RCV_REQ_reg <= GPDP_RCV_REQ_reg1 or GPDP_RCV_REQ_reg5;
   --generate FIFO read clock
   rd_clk <= GPDP_RCV_CLK_reg;

   process(rd_clk)
   begin
      if rising_edge(rd_clk) then
            if RESET_IN = '1' then
                  send_count <= "000";
            elsif GPDP_RCV_ACK = '1' and GPDP_RCV_REQ_reg = '1' then
                  if send_count = "100" then
                        send_count <= "001";
                  else
                        send_count <= send_count + "001";
                  end if;
            end if;
      end if;
   end process;
   rd_en_reg <= '1' when send_count = "001" else '0';
   send_start_reg <= '0' when send_count = "000" else '1';

   process(rd_clk)
   begin
      if falling_edge(rd_clk) then
            if RESET_IN = '1' then
                  rd_en <= '0';
                  send_start <= '0';
            else
                  rd_en <= rd_en_reg;
                  send_start_reg1 <= send_start_reg;
                  send_start <= send_start_reg1;
```

```
            end if;
        end if;
    end process;
    process(send_count)
    begin
        case send_count is
            when "010" => GPDP_IN_DATA <= dout(7 downto 0);        --FIFO dout reg 0--7 bit
            when "011" => GPDP_IN_DATA <= dout(15 downto 8);   --FIFO dout reg 8--15 bit
            when "100" => GPDP_IN_DATA <= dout(23 downto 16);  --FIFO dout reg 8--15 bit
            when "001" => GPDP_IN_DATA <= dout(31 downto 24);  --FIFO dout reg 8--15 bit
           when others => GPDP_IN_DATA <= "00000000";
        end case;
    end process;

    --core fifo32bit
    u_fifo32bit : fifo32bit
          port map (
              din => DIN,
              wr_en => WR_EN,
              wr_clk => WR_CLK,
              rd_en => rd_en,
              rd_clk => rd_clk,
              ainit => AINIT,
              dout => dout,
              full => fifo_full,
              empty => empty,
              wr_count => DATA_COUNT_WR
                  );
  end Behavioral;
```

由于篇幅的限制，尽管 FPGA 部分的设计极为复杂，在此我们不再赘述其它模块的设计过程，仅将与 BSP-15 连接的 GPDP 接口部份的设计拿出来作一介绍。至于其他模块，其功能均已完整实现，并可提供 VHDL 源设计代码作为参考。

二）驱动层

驱动层是在 BSP-15 上的，用于实现硬件 GPDP 接口控制功能的软件部分。尽管 EQUATOR 公司的 iMMediaTools 软件集成开发工具中提供了丰富的外设驱动和媒体库代码，但到目前为止，依然没有看到 EQUATOR 对 GPDP 硬件接口模块驱动程序的提供。为此，我们只有自己动手来完成这部分的内容。幸运的是，iMMediaTools 不仅提供了丰富的外设驱动程序，它还保留了完整的媒体库生成环境，我们所要做的仅仅是把我们需要的 GPDP 部分的驱动添加到其原来的媒体库中即可。

iMMediaTools 的媒体库的源码安装在其开发工具目录$ETI-TOOLS-INSTALL-DRIVER\eti_tools\src\mediadrv 下，libmediadrv.a 为最终生成的媒体库文件，添加所需要的 GPDP 软件驱动模块到该目录下，同时需要适当修改该目录下的 Makefile 文件，运行 MAKE 既可完成 GPDP 驱动程序的添加工作。

GPDP 驱动模块的设计应遵循如下步骤：

1、 编写 GPIO 驱动代码

2、 在媒体库中注册 GPIO 驱动

3、 修改 Makefile 文件

4、 Build 新的媒体库文件 libmediadrv.a

具体改动如下：*(所有文档的路径都是针对 iMMediaTools 5.5 安装目录的)*

1、创建目录及添加驱动文件： (1) ..\eti_tools\src\mediadrv\gpio\drvgpio.c
(2) .. \eti_tools\src\mediadrv\gpio\drvgpio.h

2、注册 GPIO 驱动

A）修改文件： .. \ETI_TOOLS\src\mediadrv\util\drv.h(30)：

```
#include <eti/drvgpio.h>//include gpio header.Add by fengcr
```

B) 修改文件： .. \ETI_TOOLS\src\mediadrv\init\ drvinit.c：

1):SCODE EtiDriverInit() : *//add Gpioinit.*

```
SCODE EtiDriverInit()
{
    SCODE err = ETIDRV_OK;

    DrvHandle BoardHandle = NULL;

     /* add for Initial Driver again after Encrypt. 2004-09-15*/
     s_bInitialized = FALSE;
          if (s_bInitialized) {
        return err;
     }
     s_bInitialized = TRUE;
     ○○○○○○
     err = EtiGpioInit();                          //add by fengcr 2004-08-27
     if (CHECK_RC && ETIDRV_CHECK_ERROR(err)) {
         ETIDRV_ERROR_MSG(("EtiGpioInit failed\n"));
         return err;
      }
      ○○○○○○○○○○○○○○
```

2):SCODE EtiDriverUninit() : *//add Gpioinit.*

```
      ○○○○○○○○○○○○○○
          err = EtiGpioUninit()            ;/*add by fengcr for stop GPIO 2004-09-15*/
          if (CHECK_RC && ETIDRV_CHECK_ERROR(err)) {
              ETIDRV_ERROR_MSG(("EtiGpioUninit failed\n"));
              return err;
          }
   ○○○○○○○○○○○○○○
```

C) 修改文件：

(1) ..\eti_tools\src\mediadrv\stream\drvstream.h 共计两处：

```
typedef enum {
    ETI_STREAM_VIN,          //!< Video capture stream type.
        ETI_STREAM_TCI,          //!< TCI stream type.
        ETI_STREAM_VCXO,         //!< VCXO stream type.
        ETI_STREAM_IIS,          //!< IIS stream type.
        ETI_STREAM_IEC958,       //!< IEC958 stream type.
        ETI_STREAM_GPIO,         //!< GPDP stream type. Add by fengcr
        //ETI_DRV_STREAM_DES,     //!< DES stream type.
              ETI_STREAM_SIZEOF       //!< Total number of stream types.
             } ETI_STREAM_ENUM;
    ------------------------------------------------------------------------------
-----------------------------
//! Structure used for passing settings to the Setup function.
    typedef struct _STREAM_SETTINGS {
    int            iVersion;       //!< Driver version expected by app.
    UINT16       uiAccessMode;   //!< Specifies whether to use the data cache or not
when
               //!<   transferring data. Default value is DS_DESC_DATA_ACCESS_NC.
               //!<   Use the DS_DESC_DATA_ACCESS_* macros to set a different value.
    int            iBufSize;                //!< Size of buffers in bytes.
    BUF_SETTINGS        BufSettings;     //!< Buffer management settings.
    union {
        VIN_SETTINGS      Vin;          //!< Video capture settings.
          TCI_SETTINGS      Tci;          //!< TCI settings.
        VCXO_SETTINGS    Vcxo;        //!< VCXO settings.
        IIS_SETTINGS       Iis;          //!< IIS settings.
        IEC958_SETTINGS   Iec958;       //!< IEC958 settings.
```

```
        GPIO_SETTINGS     Gpio;       //!< GPIO settings. Add by fengcr
        } Media;                          //!< Union of settings for the media type.
    BOOL                bAutoMute;      //!< Automatically mute the output,
                                        //when no buffers are  available.
    }STREAM_SETTINGS;
```

(2) ..\eti_tools\src\mediadrv\stream\drvstream.c 共计七处：

1. 函数 EtiStreamOpen () 中添加 EtiGpioOpen ()

2. 函数 EtiStreamDefault () 中添加 EtiGpioDefault ()

3. 函数 EtiStreamSetup () 中添加 EtiGpioSetup ()

4. 函数 EtiStreamKick () 中添加 EtiGpioKick ()

5. 函数 EtiStreamHalt () 中添加 EtiGpioHalt ()

6. 函数 EtiStreamClose () 中添加 EtiGpioClose ()

7. 函数 EtiStreamCheCkFifos () 中添加 :
   1): GPIO_STATUS GpioStatus;
   2): EtiGpioCheCkFifos ()

(3) 修改..\eti_tools\src\mediadrv\stream\drvstream.c 一处：

```
if ((MediaType == ETI_STREAM_IIS || ETI_STREAM_GPIO ||  // Add by fengcr
            MediaType == ETI_STREAM_IEC958) &&
            iPort == STREAM_OUTPUT_PORT) {

   pContext->bOut = TRUE;
}
```

3、Build 媒体库（libmediadrv.a），修改 Makefile 文件：

(1) 修改..\eti_tools\src\mediadrv 目录下的 Makefile 文件

```
## Object files that compose the target(s)
OBJS :=                         \
        board/drvboard          \
        board/drvdolphin        \
        board/drvshark          \
        board/drvstingray       \
        buf/drvbuf              \
        buf/drvbufdsd           \
        chip/drvchip            \
        drc/drvdrc              \
        drc/drvdrcdl            \
        iec958/drviec958        \
        iic/drviic              \
        iis/drviis              \
        gpio/drvgpio          \  Add by fengcr
       encrypt/drvencrypt   \  Add by fengcr
        init/drvinit            \
        rcon/drvrcon            \
        stream/drvstream        \
        tci/drvtci              \
        util/drvutil            \
        vcap/drvvcap            \
        vcxo/drvvcxo            \
        vdis/drvvdis            \
        vf/drvvf                \
```

```
        vin/drvvin              \
        vout/drvvout            \
        basicvideo/drvbasicvideo
## Names of headers to install
INSTALL_INC :=                  \
        board/drvboard.h        \
        buf/drvbuf.h            \
        chip/drvchip.h          \
        drc/drvdrc.h            \
        iec958/drviec958.h      \
        iic/drviic.h            \
        iis/drviis.h            \
        gpio/drvgpio.h      \  Add by fengcr
        encrypt/drvencrypt.h  \  Add by fengcr
        init/drvinit.h          \
        rcon/drvrcon.h          \
        stream/drvstream.h      \
        tci/drvtci.h            \
        util/drv.h              \
        util/drvcommon.h        \
        util/drvutil.h          \
        vcap/drvvcap.h          \
        vcxo/drvvcxo.h          \
        vdis/drvvdis.h          \
        vf/drvvf.h              \
        vin/drvvin.h            \
        vout/drvvout.h          \
        basicvideo/drvbasicvideo.h
```

(2) 在..\eti_tools\src\mediadrv 目录下，Build 媒体库。

在系统下设置环境变量：

```
ETI_TOOLKIT_INSTALL  =  d:\equator\eti_tools
ETI_TOOLKIT_LOCAL    =  d:\equator\eti_tools
```

(DISK) :\..\eti_tools\src\mediadrv\make

libmediadrv.a 及相应的头文件会自动安装到 ETI_TOOLS 目录下。

4、在..\eti_tools\src\mediadrv 目录下执行 Make 命令，Build 媒体库 libmediadrv.a。

至此，我们已经介绍完了基于 Equator 公司的 BSP15 上 GPDP ，实现 VGA/XGA 视频信号采集的方法。

参考文献:

1、《VHDL 语言设计》
2、《BSP-15 I/O Interfaces》(BSP15Vol4RevB)
3、《Xiliinx 可编程器件设计》
4、《ISE6.1.0.3》

封春日
武汉武大方略数码科技有限公司 研发部项目经理
Email：fengcr@firstlink.com.cn fengchunri@163.com