# Implementing an Automated Checking Scheme for a Video-processing Device

Genesis Microchip Inc.

Sai Wu

Session # 1.13

Presented at


## Abstract

Verification remains the single biggest challenge in the design of system-on-chip (SoC) devices and reusable IP blocks. As video SOC designs continue to grow in size and complexity, new verification technology emerge that must be linked by an effective methodology for significant adoption and deployment. In addition, video processing algorithms have became so complicated that chip and system designers, HDL designers, function verification engineers need to use a same reusable C model to ensure the algorithms equivalence during the whole SoC design process. We implemented an automated checking methodology to verify a complicate Video back-end processing device. The whole verification environment is based on reusable eVC and C-Model. Verification tools are Specman, Ncsim and gcc.

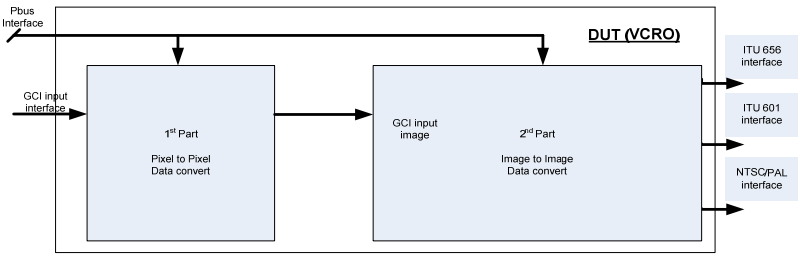
## 1. Introduction

The Genesis PrVIEW HD 300 Series IC (FLI103xx) is a single-chip TV solution for products requiring superior video quality in the analog and/or digital TV for ATSC, DVB, NTSC, OpenCable and PAL markets. This solution includes a single channel HD MPEG2 decoder, flexible analog front end with an integrated Faroudja' 3D Video Decoder, high performance industry standard 32-bit MIPS 4Kec processor (250 MIPS), multi-standard analog audio decoder, digital audio decoder and post processor, three programmable multimedia processing engines (MPE), advanced 2D graphics engine, integrated HDMI/DVI receivers with HDCP support, unified DDR memory controller and a very flexible and unique Video eXpansion Interface (VXI) providing glueless connectivity to Genesis video co-processors, or a customer's proprietary video processing chip.

The VCRO device is PrVIEW video back-end processing IP block. We can divide it into 2 parts, the first part processes color space and converts color format, this is a pixel-to-pixel convert block. The algorithm is simple and can be found in any video reference book. The second part is an image-to-image convert block, the algorithm is quit complicate. Output image dimensions frame frequency and data format are all different from those of the input image. In addition, this algorithm is proprietary and patented by Genesis, it's written in C, and used by system engineers, design engineers and verification engineers. Reuse of this C_model is most important and efficient way for VCRO device verification. Creation of multiple versions of this model for verification of the system, chip and blocks would not only require large effort, but the lack of equivalence checking at the different levels could have resulted in a final product that might not work in the intended application.

VCRO has several input / output interfaces [2]:

- GCI interface, this is a high speed video image input interface. It includes image data and associated line and frame synchronous signals.
- Peripheral bus (PBUS), the CPU configures VCRO registers over this bus.
- ITU 656, 8-bit width video interface
- ITU 601, 16-bit width video interface.
- NTSC/PAL composite S-Video output interface.



# 2. Verification environments

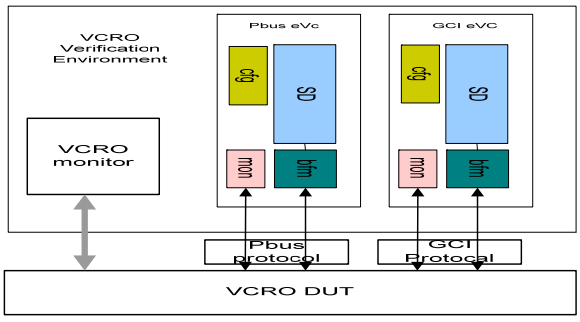
## 2.1 eVC

VCRO verification environment is based on Specman "e" language. Coverage-Driven Verification (CDV) is verification strategy. The automated stimulus approach of CDV requires an automated checker in the verification environment. To verify various client behavior connected to the data bus, e Verification Components (eVC) were reused for 2 input interfaces, they are GCI eVC and Pbus eVC. As for the 3 output interfaces, ITU 656/601 and NTSC/PAL composite S-Video interface, the DUT only drive data to output port, and doesn't get response signals from outside. VCRO monitor and scoreboard could extract data from DUT output ports directly. It's not necessary to write special eVC for these 3 output interface.
GCI and Pbus eVC are be capable of generating constrained-driven random data transfers, driving and monitoring them on the interface, checking interface signals for protocol correctness and collecting functional coverage information to ensure thoroughness of stimulus. Both eVC are all e Reuse Methodology (eRM) compliant and comprises of following blocks [1]:

- Bus Function Model (BFM) (To drive and to sample interface signals according to protocol)
- Sequence Driver (To generate desired stimuli sequences of interface transactions)
- Monitor (To collect functional Coverage Information about values, delays, transitions, interactions and assertions to detect protocol violations)

Besides GCI and Pbus eVCs, another eVC is necessary for VCRO verification environment, a register eVC, known as the vr_ad eVC. The vr_ad eVC is developed by Cadence; it is used for register verification. We can use vr_ad eVC in conjunction with the Pbus eVC, so that whenever the Pbus eVC performs register read/write operations, the associated vr_ad eVC pre-defined registers will be updated and DUT register contents will be verified by a self-checking scheme. Here are all the eVC instanced in VCRO verification environment.

- Vr_ad eVC
- Pbus eVC
- GCI eVC



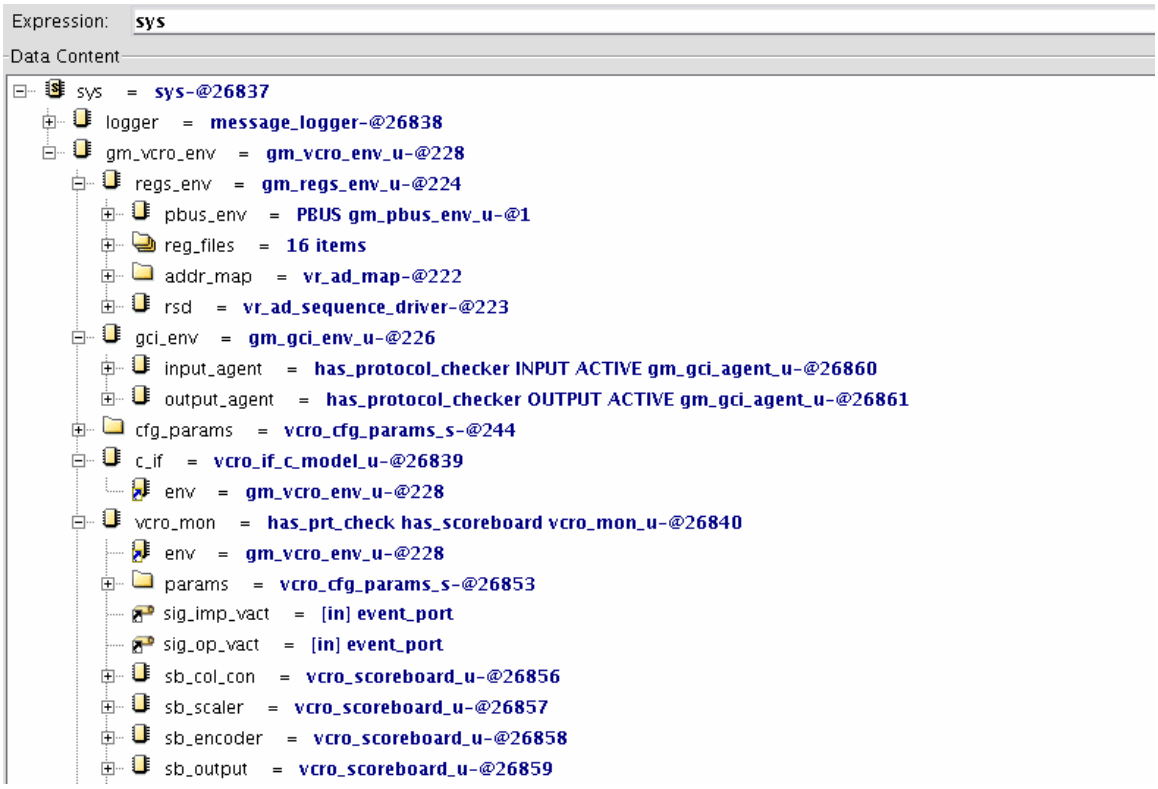

## 2.2 VCRO monitor

A monitor is required in VCRO verification environment to perform score-boarding, collecting coverage items and cross-coverage. This monitor will provide white-box verification capability to the gm_vcro_env. In addition, all scoreboards and image data checker are instanced in monitor.

## 2.3 VCRO parameter configuration

Cfg_params is a struct generated at gm_vcro_env, it include all parameters for VCRO configuration. When programming VCRO registers, Pbus eVC agent get data from this struct, and write to associated registers. When configurate GCI and Pbus eVC, configurate parameters are got form cfg_params struct too. All eVC configuration parameters come from same struct, this ensure eVC working mode matching with VCRO. For example, the image size and color space parameters, GCI agent get these parameters from cfg_params. VCRO registers are programmed on these parameters too. GCI active

agent BFM drive programmed resolution and color space image into VCRO. This can ensure input image is exactly match with VCRO image processing working mode.




# 3. Data & protocol checking scheme

VCRO self-checking scheme includes timing checking (protocol checking) and data checking. Timing checking verifies input/output signals behavior are protocol specification compliant. Data checking verifies output pixel and image data is what we expected.

## 3.1 Timing checking

VCRO has 5 interface, 2 input interfaces and 3 output interface. All input and output ports behavior must be compliant with associated interface protocol. If input interface timing is not correct, VCRO could not work correctly, and this may cause output data mismatch with expected data. But for output interface, we should pay more attention on it. Maybe output data is correct, but timing is not correct, e.g. horizontal synchronous signal is not match with data valid signal, this should be serious RTL bug.

- Input interface timing checking
  Two VCRO input interface GCI and PBUS, all connected to associate eVC. Both eVC monitor protocol checker can be reused. What we do is only configure eVC monitor has_protocol_checker item on, then the reusable protocol checker will be instanced in monitor.

```
extend gm_gci_env_u {
  ...
  Keep has_protocol_checker == TRUE;
};

extend gm_pbus_env_u {
  ...
  Keep has_protocol_checker == TRUE;
};
```

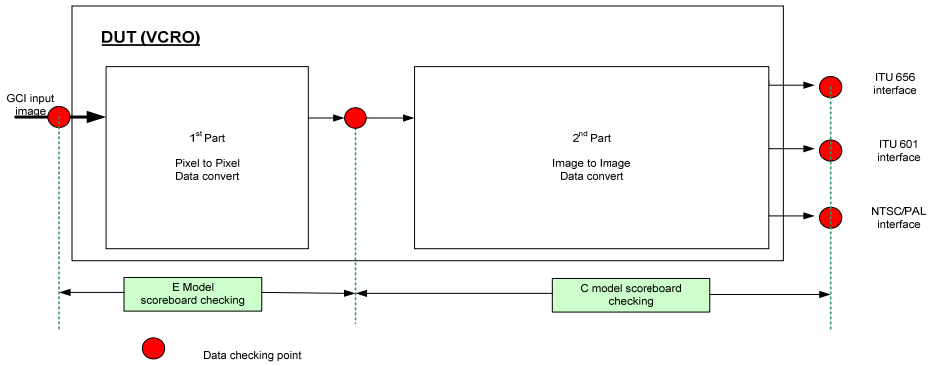
- Output interface timing checking
  Three output interfaces don't connected with eVC, no protocol checking could be reuse here. System verilog asserting (SVA) was implemented to execute timing checking. All system verification SVA modules define and instanced at DUT testbench, no any RTL file was touched in.

```
module vcr_assert(
   …
   imp_hs: assert property (@(posedge VCRO_IP_CLK) (IMP_HSYNC) |-> (!IMP_HACTIVE));
   imp_vs: assert property (@(posedge VCRO_IP_CLK) (IMP_VSYNC) |-> (!IMP_VACTIVE));
   …
Endmodule

// instance system verilog SVA module
vcr_assert vcro_prt_assert (
   …
   );
```

## 3.2 Data checking

Data checking is the most important and difficult working of VCRO verification. As the image size could be very huge, for progressive HDTV, image size is up to 1920x1080. In addition, image quality requirement is critical, and image data processing algorithm is very complicate. Data checking is the biggest bottleneck and challenge of whole VCRO verification.

We divide VCRO device into 2 parts, the 1st part is pixel to pixel data converting, and the 2nd part is image to image data converting. For the 1st part, it's e model self-checking, and 2nd part is C model self checking.




### 3.2.1 e model

For 1st part, an e model is used to implement pixel to pixel scoreboard self-checking. Whenever any valid pixel is driven to DUT GCI interface by GCI eVC active agent, this pixel is driven to e model at same time. E model convert this pixel to expected pixel, then monitor push it into scoreboard. Similarly, whenever monitor gets a 1st part RTL output valid pixel, it pop out at the bottom of scoreboard and compared it with RTL pixel. If compare results is mismatch, DUT error occur. The following are some code of this e model scoreboard checking:

**Monitor:**

```
-- execute pixel to pixel scoreboard check
extend has_scoreboard vcro_mon_u {
  ...
  -- whenever 1 pixel driven to DTU, add converted pixel to scoreboard
  on gci_drv_pixel {
     convert_pixel = env.c_if.pixel_convert(gci_drv_pixel,color_space, i);
     monitor.pix_scoreboard.add(convert_pixel);
   };
```

```
    ...
    -- whenever recieve valid pixel converted output data...
    on rcv_rtl_pixel {
      rcv_pixel = 'PIXEL_OP_PATH/oDATA';

    -- compared with scorebaord pop out pixel...
      monitor.pix_scoreboard.match(rcv_pixel);
    };
  };
```

**Scoreboard:**

```
  unit vcro_scoreboard_u {
    ...
    !scoreboard : list of uint (bits:16);

    -- add new data into scoreboard
    add(pixel : uint) is {
      scoreboard.add(pixel);
    }; -- add()

    -- compare RTL pixel and expected pixel
    compare_pixel(exp_data : uint, det_data : uint, compare_dest : bool) : list of string is {
      if ( exp_data !=  det_data) {
          result.add(append("Expected data:", exp_data, ", Actual data:", det_data));
        }; // if (exp_frame !...
    }; -- compare_frames()

    -- execute scoreboard pixel data check
    match(char : uint) is {
        …
        var tem_sb_data : uint = scoreboard.pop0();
        var diff_list : list of string = compare_pixel(tem_sb_data, char, compare_destinations);
          check that diff_list.is_empty()
          else dut_error("Scoreboard '", name,"' - detected data mismatch, "\n",
          "detected data: ", char, "\n",
          "expected data: ", tem_sb_data);      };
    }; -- match();
    ...
  };
```

### 3.2.2 Integrate C model into verification environments

Reusable C model is the most important verification component. As system designer, RTL designer and verification engineers use same reusable C models; this could ensure algorithms identical during whole SOC design and verification period. In addition, image convert algorithm is very complicate, reusing this complicate C model make VCRO verification schedule a lot more predictable and result in reduction of half-the effort.

Specman has an interface connected to C or C++ models. This interface has following features [1]:

- pass data between e environment and C models
- Call c routines from e code.
- Use e data types in C code
- Call e methods and manipulate from C code.

In VCRO verification environment, we need passing data between e environment and C models, and calling c routines from e code. The following are steps of VCRO data checking on reusable C models:

1. Write e routine

   First of all, we need create an e file which defines all the C routines. Then compile this e file in advance, and link the compiled objected file into our verification environment. Here are the e codes to define e routine to call associated c routine:

   ```
   //C routine to dump RTL image into Genesis format image file
   ```

```
routine ertl2tga(height : int, width : int, luma : list of int,...) : int is C routine e_rtl2tga;

//C Model for image convert
routine eimg_656_convert( ..., x_in_size : int , y_in_size: int , ...) : int is C routine e_img_656_convert ;

//C Model for image convert
routine eimg_601_convert( ..., x_in_size : int , y_in_size: int , ...) : int is C routine e_img_601_convert ;

//C Model for NTSC/PAL encoder
routine eimg_tv_convert( ..., x_in_size : int , y_in_size: int , ...) : int is C routine e_img_tv_convert ;
```

2. Selecting RTL data checking point and implement image data checking.

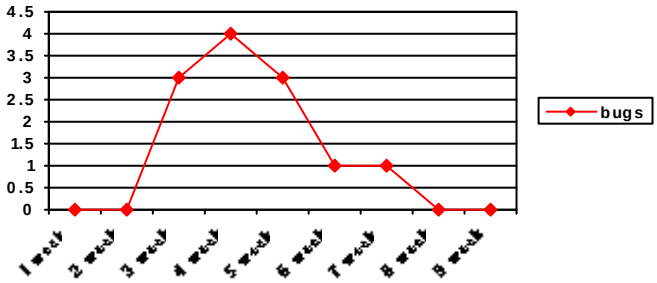
   VCRO 2nd part has 3 output interfaces, each interface associated with one C_model.

   - Whenever 1 frame driven to VCRO 2nd part input, this RTL image data was dumped to Genesis format image files and saved as vcro_2nd_ip_image_00*.
   - When monitor get an valid stable output image at any of output interface, dump RTL image to Genesis format image files and save as vcro_2nd_656_op_image_00*
   - Use vcro_2nd_ip_image_00* as C_model input image, get expected output image, then compare this expected output image with VCRO ITU 656 interface RTL output image.
   - Date checking of ITU 601 and NTSC/PAL TV output interfaces is similar.

```
extend has_scoreboard vcro_mon_u {

 -- VCRO 2nd part block input image end event
 on vcro_2nd_ip_image_end {
   ...
 -- dump VCRO 2nd part block input data into genesis frame format file
   ertl2tga(params.ip_active_hight, params.ip_active_width,..."vcro_2nd_ip_image");

 };
 ...
 -- VCRO 656 output frame end event
 on vcro_2nd_656_image_end {
   ...
   -- dump VCRO ITU 656 output image into genesis frame format
   ertl2tga(params.op_active_height, params.op_active_width, ..."vcro_2nd_656_op_image");

   -- Begin to compare VCRO ITU656 output image data with C_model output image data ....");
   cmp_f = eimg_656_convert (          "vcro_2nd_ip_image","vcro_2nd_656_op_image",
                                params.ip_active_width,params.op_active_width...)
   check that (cmp_f == 0)
     else dut_error("VCRO output data is not correct!");
};
```

## 3. Verification results

Reusable eVC, protocol checker and C_model make VCRO verification schedule a lot more predicable. In addition, self-checking scheme and random generation resulted in reduction of half-the effort. After 4 weeks regression, total 12 bugs found, and 3 bugs are internal IP block bugs never found before. In addition, no any new RTL bug found at FPGA component verification and chip validation. Compared with traditional direct verification, specman random verification and reusable self-checking methodology is much more efficient and verification quality is satisfactory.



## References

[1] "***e*** Reuse Methodology (***e***RM) Developer Manual",
Verisity Design, Inc, 2006.
[2] “Draft ITU-T Recommendation and Final Draft International Standard of
Joint Video Specification,” *Joint Video Team*, Mar. 2003.