User Guide: Frame Merging Output Peripheral

Opal Densmore, Kei-Ming Kwong, Wahid Rahman,

February 5th 2014

Version for EDK 14.2

1 Goals

Develop a frame merging IP which merges two frames into a composite frame by overlaying one on top of another. The output frame is stored in memory, where an HDMI-Out IP will output the frame to a monitor.

2 Overview

Provided along with this documentation is an example project that use a system that includes two custom peripheral cores: hdmi_out and frame_merge. Due to bugs within the hardware block, refer to appendix for further documentation of the frame_merge custom core, this rest of this document will mainly focus on software IP.

The sample project takes two initialized input frames and combines them to generate the final output composite frame at 640x480px resolution and RGB888 colour space. The following is the terminology in referring to the different frames.

- Draw Frame: an input frame which is overlayed on the top
- Video Frame: an input frame which is under the draw frame
- Composite Frame: an output frame with the output of the merging

The frame merging functionality are realized in a software algorithm provided in the C API which provides the merging functionality through sequential processing by the MicroBlaze. Although the example provided only supports a specific colour space and resolution, these are settings which can be customized through the HDL or the C API.

3 Directory Structure

The following directory structure indicates important files and folders which are of key interest to a developer using the example project provided.

- project\
  - system.xmp
  - data\
    - system.ucf → Pin Connection mapping of the blocks
  - pcores\
    - hdmi_out_v1_00_a
      - user_logic.v → Interface logic between AXI BUS and custom HDL
      - hdmi_core.v → Custom HDL to output HDMI in a configuration
  - workspace\
    - sample_0\src
      - frame_merge.c
        - C API for frame merging functionality (software or hardware)
      - frame_merge.h
        - Header file for C API
      - main.c
        - Sample code to utilize API functions
      - lscript.ld
        - Linker script to identify different section to address mappings
    - sample_bsp_0\
      - The board support headers and src files

4 Setup

1. Start XPS and select Open Project. Browse to the system.xmp within the project folder.
   a. Project → Project Options → Advanced Options → Project Peripheral Search path
      Ensure that this points to the your local installation of the Atlys_AXI_BSB_support files
2. Generate Bitstream and wait for design to compile. This may take several minutes.
3. Export Design and launch Xilinx Software Development Kit.
   a. Select “Include bitstream and BMM file” option
   b. Use project\workspace as the workspace folder
4. In the workspace, sample_0\ and sample_bsp_0\ are provided as an example.

5. Create a new Xilinx C Project and select Empty Application. It is advisable to recreate your own BSP project instead of reusing the sample_bsp_0 BSP.
6. Copy the library files from sample_0\src including the linker script "lscript.ld"
    a. Build Project
7. Compile the project and Program the FPGA using the .elf file you just compiled.
8. If you have the UART connected, you should see status checks being printed.
9. On the Monitor, the composite frame (shown below) will be on the monitor. By entering 0/1/2 on the UART terminal, the draw/video/composite frame will be shown instead.

Figure 1 draw frame (left), video frame(right)

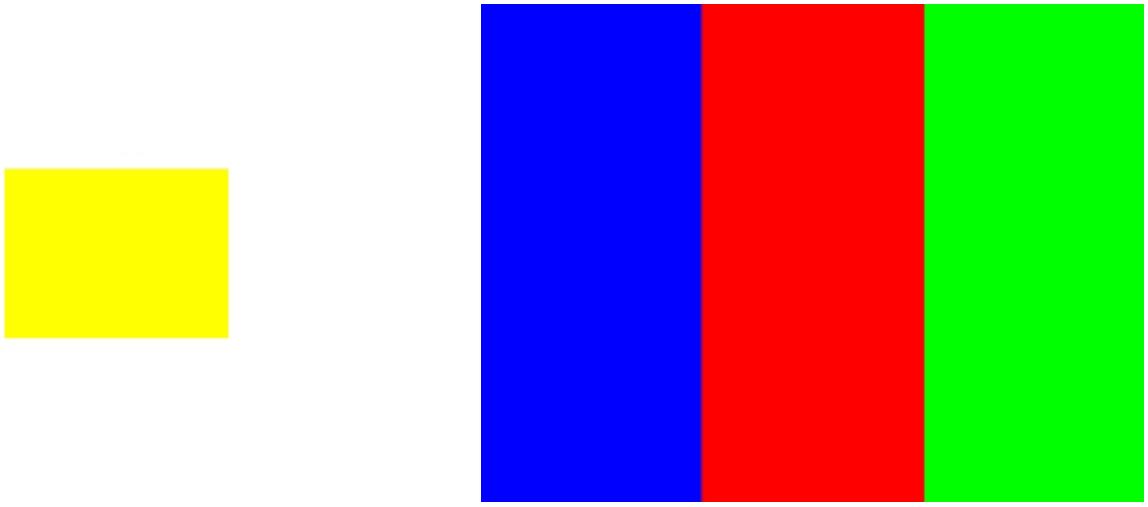
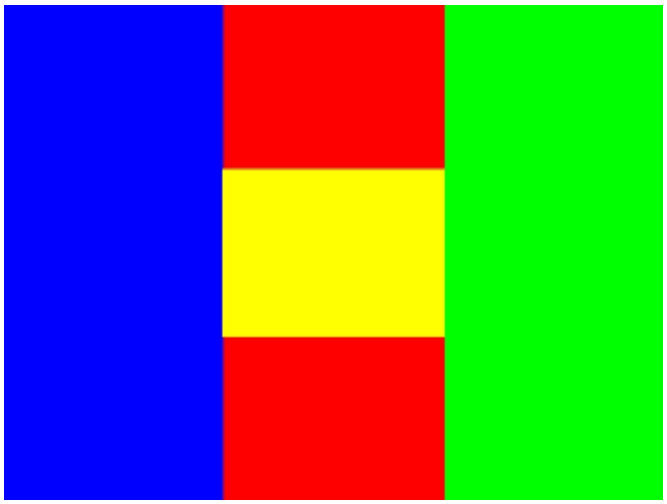


Figure 2 composite frame (Output)



5 Diagrams

The following diagrams provide a more in-depth view of the IP in terms of software algorithm. For hardware system diagrams, refer to appendix diagram sections.

Figure 3 shows the flow of the software algorithm in the frame merge IP

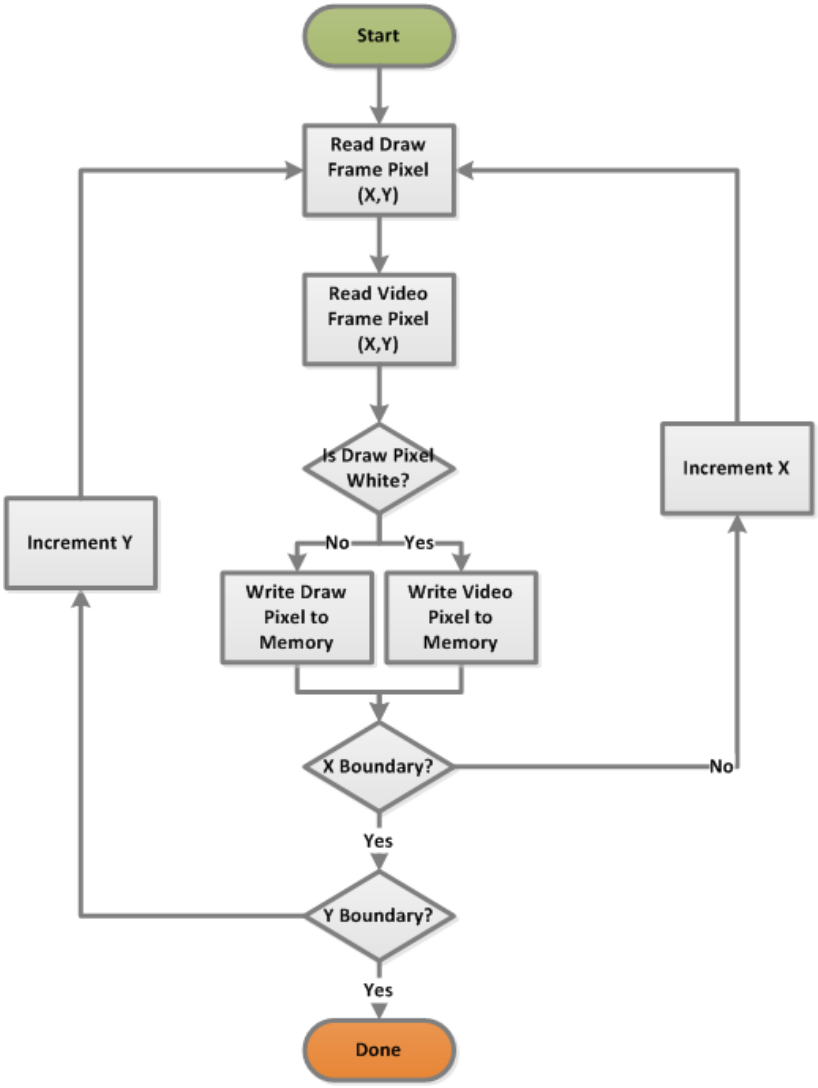
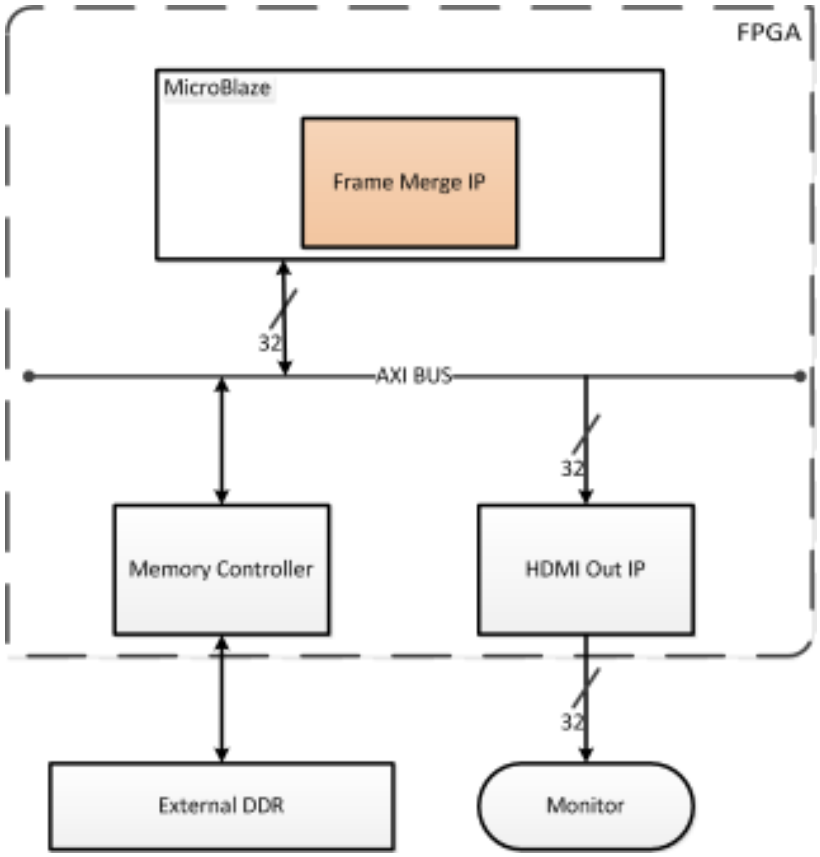


Figure 4 shows the system architecture, highlighting the relationship between the different blocks and the frame merge IP



6 Setting parameters

There are several locations to change for resolution or colour space settings: the hdmi out IP core, and the C program.

1. Hdmi out IP Core specifications (Click on the block in XPS to change)
    a. HDMI HRES refers to the horizontal resolution. For example, 1280 refers to a resolution of 1280x720.
    b. HDMI NUM BYTES PER PIXEL refers to the colour format. 2 refers to RGB565 and 4 refers to RGB888x.
    c. Note that 640x480 and 800x600 need an input clock of 25MHz, while the 1280x720 resolution needs an input clock of 75MHz. The two projects have the correct clock setup. If you change the parameters, you may also need to change the software. Note that line stride is specified in pixels, so line stride may not need to change if only the colour format is different.
2. C program (Setting the parameters passed into the library API call)
    a. HRES refers to the horizontal resolution
    b. VRES refers to the vertical resolution

7 Frame Merge API and Usage

Software API → uses the MicroBlaze for sequential processing of frame merging algorithm

void frame_merge_sw(u32 ddr_Addr, u32 Draw_Offset, u32 Video_Offset, u32 Comp_Offset, u32 hres, u32 vres);*

Software Usage Example

```
// Address Setup, offset is calculated based on resolution and pixel byte size
#define DF_OFFSET 0x0
#define VF_OFFSET 0x400000
#define CF_OFFSET 0x800000
#define HRES 640
```

#define VRES 480

Frame_Merge Merge_IP

// Define base DDR address and initialize

*volatile u32 *ddr_addr = (volatile u32 *) XPAR_S6DDR_0_S0_AXI_BASEADDR;*

// run merge

frame_merge_sw(ddr_Addr, DF_OFFSET, VF_OFFSET, CF_OFFSET, HRES, VRES)

8 Caveats/Limitations

- White is used as a reference, a white object will not be overlayed on the video frame
- Since the hdmi out IP core lacks support for reset, the reset button will not reset the project.

9 Reference Material

The following are reference links for the components used in the IP

- cvPaint - Simple drawing using camera (http://farshid.ws/projects.php?id=112)
- User Guide: HDMI output peripheral (https://github.com/molohov/gpu2/tree/master/ip)

10 Contact

Refer to https://github.com/kming/ece532/tree/master/IP_Project for the most up to date version of the source code/documentation/example project

11 Appendix: Hardware Documentation

11.0 Note

The follow section refers specifically to the hardware block “frame merge” and the usage of the API for this hardware block. However, the hardware block isn’t working properly as of Feb 7th 2014. The hardware block will corrupt the input frames, however, it will merge the two corrupted version of the input frames together.

11.1 Overview

The frame merging functionality are realized in two different methods: hardware implementation and software implementation. The frame_merge IP is the hardware implementation of the merging capabilities. Provided along with the IP is a C API called “frame_merge” which handles all the interaction with the IP core. The second method is a software algorithm provided in the same C API which provides the merging functionality through sequential processing by the MicroBlaze. Users can define the specific method to be used.

Although the example provided only supports a specific colour space and resolution, these are settings which can be customized through the HDL or the C API.

11.2 Directory Structure

The following directory structure indicates important files and folders which are of key interest to a developer using the example project provided.

- pcores\
  - frame_merge_v1_01_a
    - user_logic.v → Interface logic between AXI BUS and custom HDL
    - frame_merge_core.v → Custom HDL to merge frames
  - hdmi_out_v1_00_a
    - user_logic.v → Interface logic between AXI BUS and custom HDL
    - hdmi_core.v → Custom HDL to output HDMI in a configuration

11.3 Diagrams

The following diagrams provide a more in-depth view of the IP in terms of software and hardware components.

Figure 11.1 shows the system architecture, highlighting the relationship between the different blocks and the frame merge IP

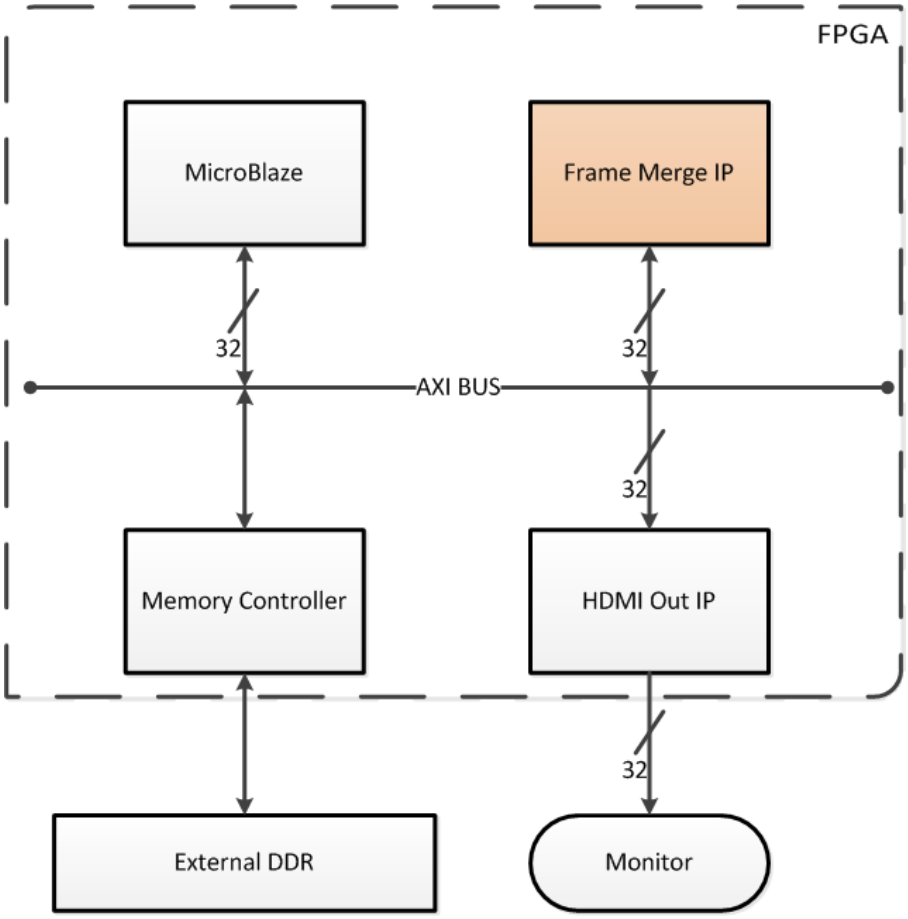


Figure 11.2 shows the main blocks within the frame merge IP

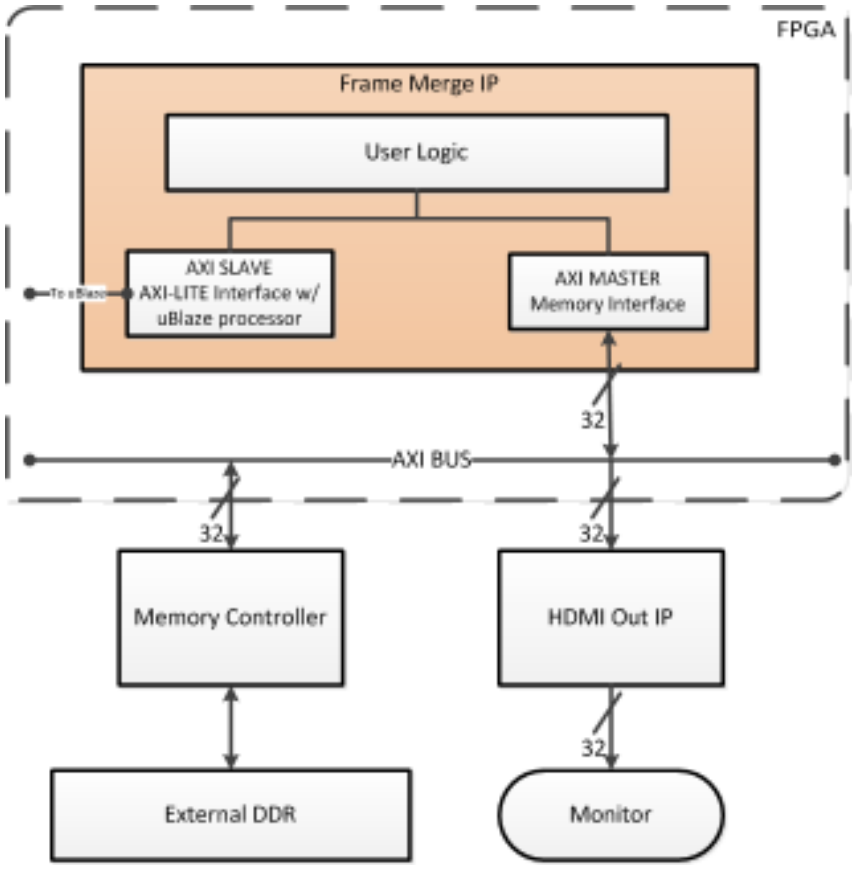
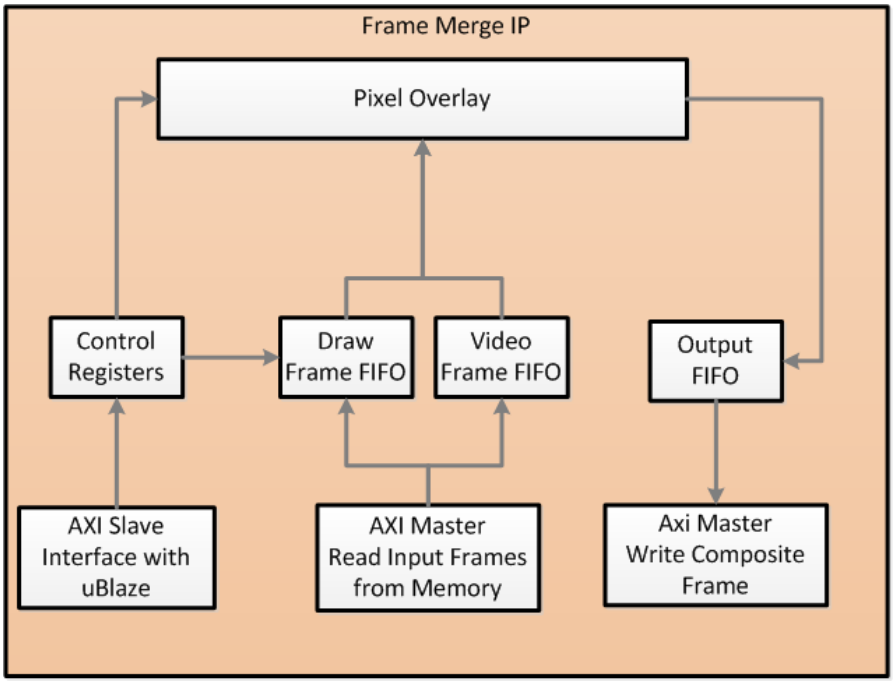


Figure 11.3 shows the detailed implementation of the user logic that forms the IP.



11.5 Memory Mapped Registers

Register 0: Go Signal → Used to tell the IP core to start processing.

Register 1: Input Draw Frame Address

Register 2: Input Video Frame Address

Register 3: Output Composite Frame Address

The registers map closely to the software algorithm with the addition of an enable signal to start the hardware core, where the software algorithm would process sequentially without the need of a "go" enable signal.

11.6 Frame Merge API and Usage

Hardware API → interfaces with the frame merge IP through AXI to handle frame merging.

*void frame_merge_Initialize(Frame_Merge *InstancePtr, u32 Base_Addr)*

*void frame_merge_Go(Frame_Merge *InstancePtr)*

*void frame_merge_Stop(Frame_Merge *InstancePtr)*

*void frame_merge_SetAddr(Frame_Merge *InstancePtr, u32 Draw_Addr, u32 Video_Addr, u32 Comp_Addr)*

Hardware Usage Example

```
// Address Setup, offset is calculated based on resolution and pixel byte size
#define DF_OFFSET 0x0
#define VF_OFFSET 0x400000
#define CF_OFFSET 0x800000

Frame_Merge Merge_IP

// Define base DDR address and initialize
volatile u32 *ddr_addr = (volatile u32 *) XPAR_S6DDR_0_S0_AXI_BASEADDR;
frame_merge_Initialize (&Merge_IP, PAR_FRAME_MERGE_0_BASEADDR);
```

```
int df_addr = (int) ddr_addr + DF_OFFSET
int vf_addr = (int) ddr_addr + VF_OFFSET
int cf_addr = (int) ddr_addr + CF_OFFSET

// At this point, the address are configured, for the IP specified
frame_merge_SetAddr (&Merge_IP, df_addr, vf_addr, cf_addr)

// Can set the hardware block to start
frame_merge_Go (&Merge_IP)
```

11.7 Caveats

There are known issues with the frame merge API provided

- Software and hardware do not have checks to ensure they don't conflict. **Do not** use the software algorithm on the same addresses while the hardware is running.