Third Prize

3-D Accelerator on Chip

Institution: **Donga & Pusan University**

Participants: **Young-Hee Won, Jin-Sung Park, Woo-Sung Moon**

Instructor: **Sam-Hak Jin**

Design Introduction

Recently, consumers are becoming interested in cellular phones and portable game devices that play 3-dimensional (3-D) games. It is difficult for mobile device processors to compute 3-D graphic operations because they require a lot of arithmetic operations and most mobile device processors cannot process them. To solve this problem the processor chip must incorporate 3-D accelerator units to reduce the computing time. In this project, we developed a 3-D graphic display system that quickly computes 3-D graphic operations by including a hardware 3-D accelerator in the chip, and creating applications in it.

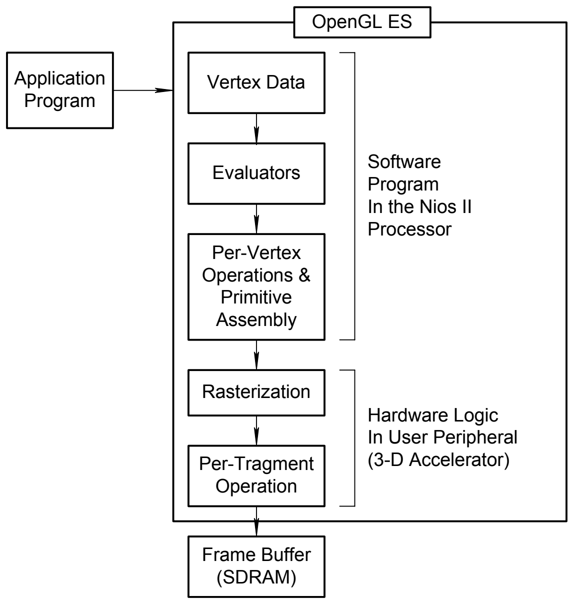
Using the Nios® II processor to develop 3-D graphic displaying system makes it possible to integrate the main processor and 3-D accelerator in one chip. The system is smaller, faster, and more stable than when using a hard-core processor chip and a separate 3-D accelerator chip.

Our 3-D graphic display system is based on OpenGL ES version 1.1, which is a royalty-free, cross-platform application programming interface (API) for full-function 2-D and 3-D graphics on embedded systems, including handheld devices, appliances, and vehicles. It is a well-defined subset of desktop OpenGL, creating a flexible and powerful low-level interface between software and graphic acceleration. OpenGL ES Pipeline is based on OpenGL 1.3 Pipeline, which includes geometry processing, rasterization, fragment processing, and frame buffer operations. Programmers who have used the desktop OpenGL can easily develop application programs for OpenGL ES. Therefore, our system can also be used to develop 3-D graphics applications, such as 3-D games.

Function Description

Our system is based on OpenGL ES version 1.1; therefore, the system offers same interface. If the application program makes vertex data by using offered functions, it computes the 3D operations, such as rotation or transfer 3-D vertexes, lighting, clipping, and so on. Additionally, the software makes final vertex data that is viewed by the camera set to see the 3-D world. See Figure 1.

Figure 1. Pipeline Flow of OpenGL ES in the System

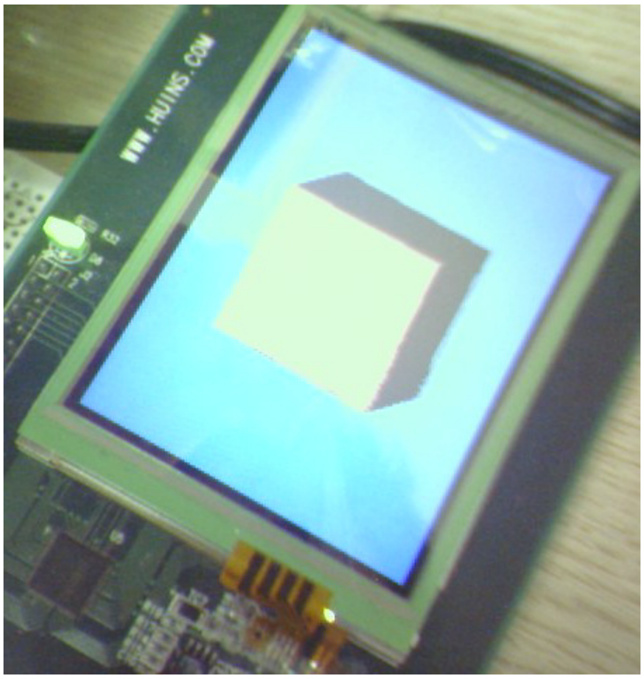

OpenGL ES
Application Program
Vertex Data
Evaluators
Per-Vertex Operations & Primitive Assembly
Software Program In the Nios II Processor
Rasterization
Per-Tragment Operation
Hardware Logic In User Peripheral (3-D Accelerator)
Frame Buffer (SDRAM)

The software part of the 3-D graphic processing function delivers the positions of the three points of the triangles and the color of polygons to the 3-D accelerator, which is the hardware part of the 3-D graphic processing function. The 3-D accelerator makes the *x*, *y*, and *z* positions of the inside of the polygons to fill up it. It also writes the color data of points to the exact address of the frame buffer, which is placed in SDRAM. The thin-film transistor (TFT)LCD controller module independently reads the frame buffer and displays the 3-D graphics on the TFT LCD repeatedly. The application programmer programs the applications using the OpenGL ES library and the system displays the 3-D graphic output of application.

Performance Parameters

The most important performance parameter of a 3-D graphic display system is the speed of computing and displaying a frame of 3-D graphics. The general unit of 3-D graphic display speed is frames per second (fps). We tested the speed of our system displaying a cube spinning on an *x*, *y* diagonal axis, as shown in Figure 2.

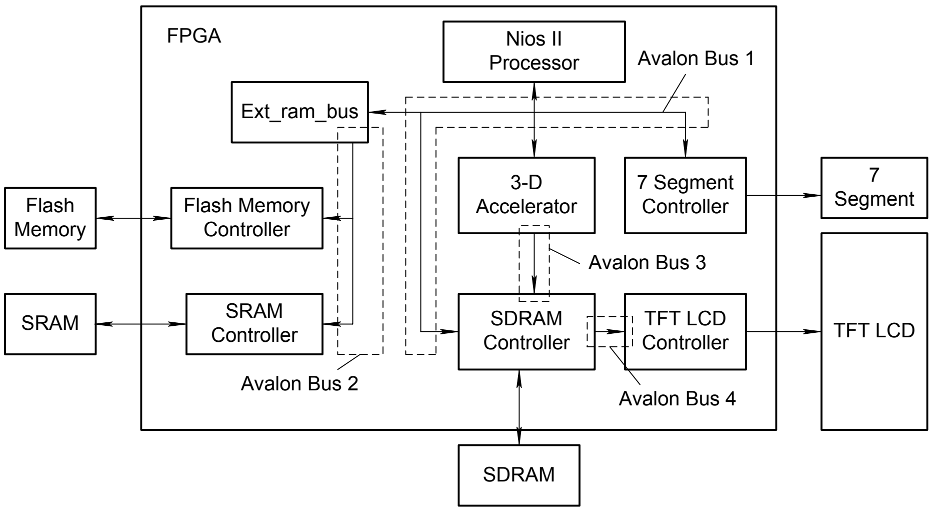
Figure 2. Test Display



Design Architecture

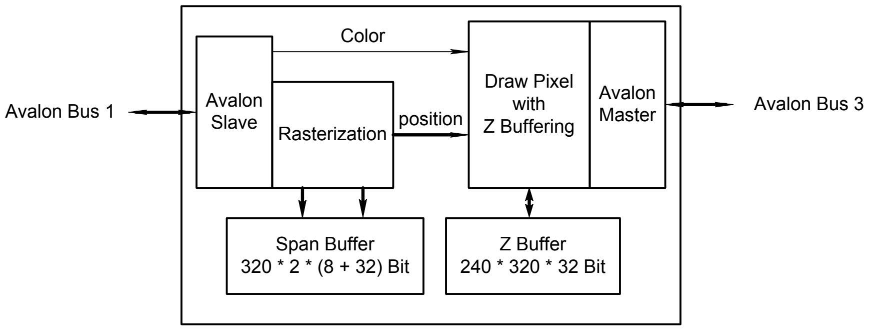
Figure 3 shows the top-level block diagram. The Nios II processor connects to ext_ram_bus, 3-D accelerator, 7-segment controller, SDRAM controller, TFT-LCD controller, and so on. The ext_ram_bus module is a tristate Avalon® bus bridge that connects the Nios II processor to flash memory and SRAM, which are the instruction memory and data memory, respectively, used to run the Nios II processor.

Figure 3. Top-Level Block Diagram



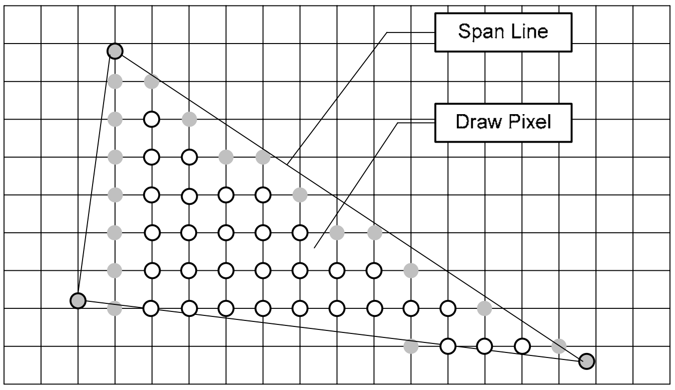
The 3-D accelerator (Renderer) is a slave of Avalon bus connected to the Nios II master (see Figure 4). The Renderer receives the polygon data and starts rasterization. After rasterization, the outline pixel position data of the triangle of polygon is restored in the span buffer, which uses the Altera® FPGA's internal memory. The structure of the span pixel data is *x* (8-bit integer) and *z* (32-bit fixed-point real number), and the address is *y* (9-bit integer).

Figure 4. Block Diagram of Rendering Module


Color
Avalon Bus 1
Avalon Slave
Rasterization
position
Draw Pixel with Z Buffering
Avalon Master
Avalon Bus 3
Span Buffer 320 * 2 * (8 + 32) Bit
Z Buffer 240 * 320 * 32 Bit

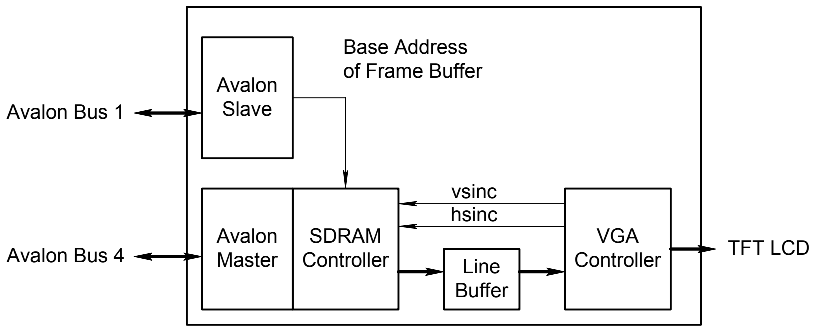
After rasterization, the position data of *y*, the right side of *x*, the left side of *x*, and the *z* position of each side is delivered to the draw pixel module. This module makes the *x* and *z* position data of the pixels between the received pixel of the right side and left side and loads the *z* position data of that (*x*, *y*) position to compare the depth that is drawn and to draw. If the pixel to draw is closer to the camera than the pixel drawn previously, the module writes color data of the pixel to the frame buffer in SDRAM through the Avalon bus connection and the *z* position data to the *z* buffer. If it is further than the currently drawn pixel, it is ignored. See Figure 5.

Figure 5. The Rasterization Process



The LCD Control module displays the data of the frame buffer to the screen of the TFT LCD. It receives the base address of the frame buffer from the Nios II processor. The SDRAM Control module, a master of Avalon bus synchronized to the VGA Controller's sync signals, repeatedly reads data from the frame buffer in SDRAM. See Figure 6.

Figure 6. LCD Controller Block Diagram



While the software operates, it uses a custom instruction to multiply or divide fixed-point real number type data.

The software is very complex and can be changed by the application program. We developed the OpenGL ES library by modifying the details of open-source OpenGL ES code to fit into our system. The functions have same interface with OpenGL ES that the well-defined subset of desktop OpenGL has. Therefore, an application program using our functions operating in the system processes the 3-D graphic operations as shown in Figure 1, in software and hardware.

Design Methodology

We developed the LCD Controller first and displayed a sample image to the TFT LCD with a simple software program. Next, we developed the OpenGL ES software library by modifying the open-source code. We reprogrammed the process of writing pixel data to the frame buffer because our system uses memory differently. We changed computations using floating-point numbers to fixed-point numbers, and some functions, including those concerning lighting, were changed to fit our system.

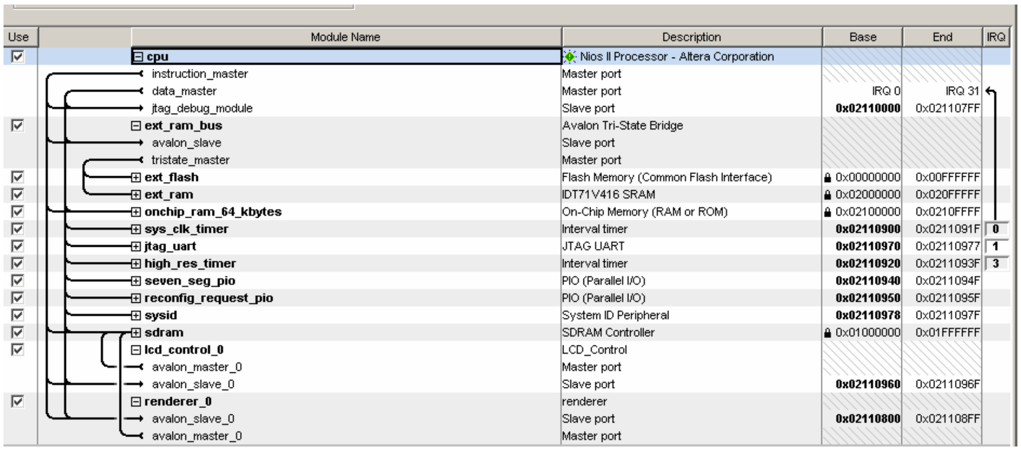
After we checked the 3-D graphic frame displaying of our test application without the 3-D accelerator, we developed the 3-D accelerator module. We changed the software functions to deliver data to hardware instead of computing it with the main processor.

We used a 50-MHz system clock to reduce compilation time. After we checked the system's operation, we set a phase locked loop (PLL) to generate a 100-MHz clock to run the final system.

The main CPU is the Nios II processor. The design uses the Nios II /f processor, flash memory, and RAM controllers, which are connected to the Nios II processor by a tristate Avalon bridge, ext_ram_bus. Additional modules, such as a timer, JTAG UART, etc., run and debug the Nios II processor. The 7-segment PIO Controller displays the system speed.

We wrote the 3-D accelerator module (Renderer) in VHDL with Avalon bus slave and master signals. We used the **New Component** menu option in SOPC Builder to add the VHDL code. The slave side of the module connects to the Nios II processor, and the master side connects to the SDRAM controller, as shown in Figure 7.

Figure 7. SOPC Builder System



Use	Module Name	Description	Base	End	IRQ
✓	cpu	Nios II Processor - Altera Corporation			
	instruction_master	Master port			
	data_master	Master port	IRQ 0	IRQ 31	
	jtag_debug_module	Slave port	**0x02110000**	0x021107FF	
✓	ext_ram_bus	Avalon Tri-State Bridge			
	avalon_slave	Slave port			
	tristate_master	Master port			
✓	ext_flash	Flash Memory (Common Flash Interface)	0x00000000	0x00FFFFFF	
✓	ext_ram	IDT71V416 SRAM	0x02000000	0x020FFFFF	
✓	onchip_ram_64_kbytes	On-Chip Memory (RAM or ROM)	0x02100000	0x0210FFFF	
✓	sys_clk_timer	Interval timer	**0x02110900**	0x0211091F	0
✓	jtag_uart	JTAG UART	**0x02110970**	0x02110977	1
✓	high_res_timer	Interval timer	**0x02110920**	0x0211093F	3
✓	seven_seg_pio	PIO (Parallel I/O)	**0x02110940**	0x0211094F	
✓	reconfig_request_pio	PIO (Parallel I/O)	**0x02110950**	0x0211095F	
✓	sysid	System ID Peripheral	**0x02110978**	0x0211097F	
✓	sdram	SDRAM Controller	0x01000000	0x01FFFFFF	
✓	lcd_control_0	LCD_Control			
	avalon_master_0	Master port			
	avalon_slave_0	Slave port	**0x02110960**	0x0211096F	
✓	renderer_0	renderer			
	avalon_slave_0	Slave port	**0x02110800**	0x021108FF	
	avalon_master_0	Master port			

We used the **New Component** menu option in SOPC Builder to include the LCD Controller module and specify its signals. The signals between the LCD Controller and Avalon bus became the pins of the Nios II module. The slave side of the module is connected to the Nios II processor and the master side is connected to the SDRAM controller. We developed the LCD Controller module in VHDL, created a Symbol File of it, and connected it to the Nios II module. See Figure 8.

Figure 8. Top-Level Circuit

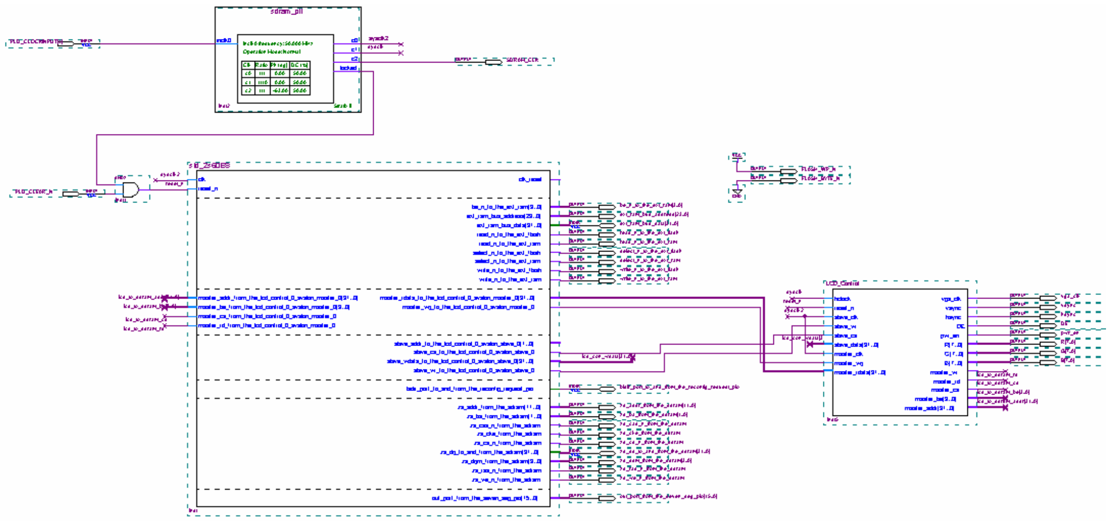
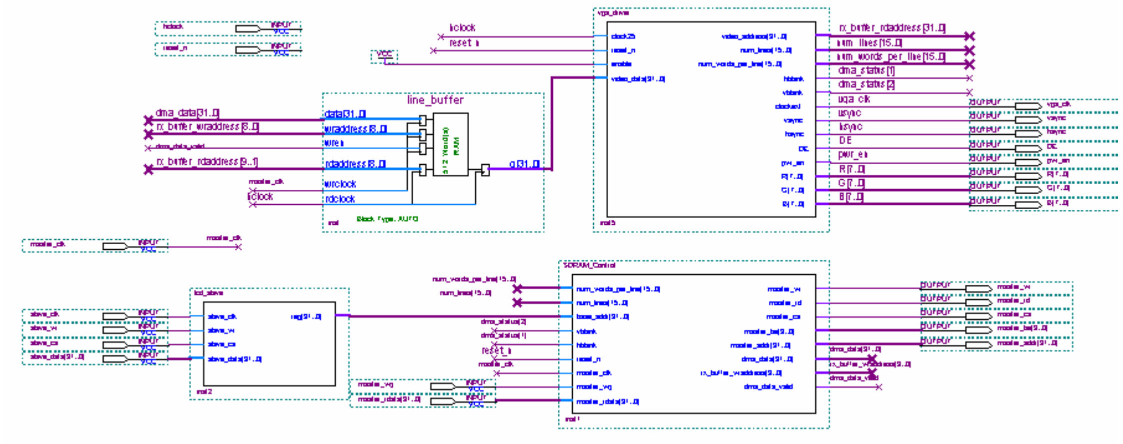


Figure 9 shows the LCD Controller block association in the Quartus® II software. We created each block in the LCD Controller module using VHDL, and used the Quartus II MegaWizard® Plug-In Manager to develop the line buffer, which uses the FPGA's internal memory.

Figure 9. LCD Controller Block Diagram



We also used the Quartus II MegaWizard Plug-In Manager to create the 3-D accelerator line span buffer, and the multiplication and division operations.

While the software operates, it performs many multiplication and division operations of fixed-point real numbers. These numbers are defined as:

```
#define __GL_X_MUL(a,b) ((__gl_x)((((__gl_ll)(a))*(b))>>__GL_X_FRAC_BITS))
#define __GL_X_DIV(a,b) ((__gl_x)((((__gl_ll)(a))<<__GL_X_FRAC_BITS)/(b)))
```

Where `__gl_x` is a 32-bit, fixed-point read number, `__gl_ll` stores 64-bit data, and `GL_X_FRAC_BITS` means the bits under point, defined as 16. We created custom instructions for these numbers as shown in Figures 10 and 11.

Figure 10. Fixed-Point Division Custom Instruction

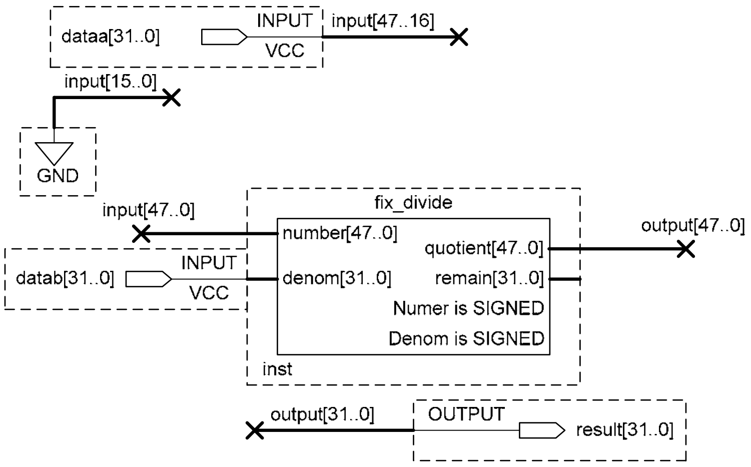
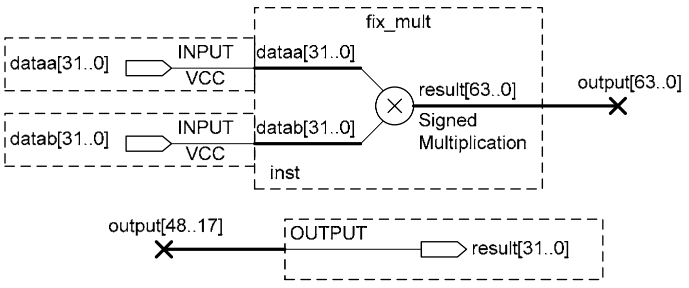


Figure 11. Fixed-Point Multiplication Custom Instruction



We used the Quartus II MegaWizard Plug-In Manager to create the multiplication and division processors and added these modules to the design as custom instruction with the definition:

```
#define __GL_X_MUL(a,b)   (__gl_x)__builtin_custom_inii( 0, a,b)
#define __GL_X_DIV(a,b)   (__gl_x)__builtin_custom_inii( 1, a,b)
```

Using custom instructions makes the system faster because the custom instruction only uses 1 clock cycle instead of more than 2 required without the custom instruction.

Design Features

We developed the 3-D graphic display system with the Nios II processor and our 3-D accelerator in one chip. This design makes it possible for small devices, such as cellular phones or portable game devices, to display 3-D graphics faster in a smaller device.

Because the 3-D accelerator is in the same chip as the main processor, the system size is smaller than the same functional system using a hard-core processor chip, separate 3-D accelerator chip, the signal connections between the two chips, power sources, memories, and so on.

The design is also faster than using variable chips, because the access to SDRAM is controlled by the Avalon bus without complex control signal protocols. Additionally, custom instructions make it more capable. The system has reduced wires and power source regulators by using a synchronized clock from the PLL blocks. Therefore, the system can run without noise or clock sync errors, making it much more stable than using variable chips.

Conclusion

We developed the 3-D graphic display system with the Nios II soft-core processor and our designed 3-D accelerator in one chip. This system allows small devices, such as cellular phones or portable game devices, to display 3-D graphics faster in a small size. Although the system is not currently fast enough for a consumer product, we could develop additional hardware modules for various operations and change the software processes to hardware processes to divide the processing loads in each section of pipeline, thereby increasing the computing speed.

The Nios II processor is very useful for embedded system engineers. With it, we were able to integrate a processor and our designed hardware in one chip, making the system smaller, faster, and more stable. In particular, using Nios II custom instructions makes the system much more efficient than using hard-core processors or only FPGAs.

The Avalon bus was very easy to create and use to connect blocks in the FPGA. For example, with just a few mouse clicks in SOPC Builder made it possible to connect blocks, even several master blocks accessing several slave blocks. In our system, the Nios II processor, 3-D accelerator, and LCD controller are all masters of the SDRAM controller, and using the Avalon bus makes the operation smooth, without error or crossing data.

The Quartus II software, which includes SOPC Builder and MegaWizard Plug-Ins, made it very easy to include and connect several current designs. It reduced our development period and made the design process less complex.