First Prize

Police Vehicle Support System with Wireless Auto-Tracking Camera

Institution: **Inha University, Korea Aerospace University, Hongik University**

Participants: **Sung Woong Joo, Ho Seong Suh, Young Je Moon**

Instructor: **Professor Tak Don Han**

Design Introduction

Our project, a police vehicle support system with a wireless auto-tracking camera, has three main goals:

- To improve police vehicles' ability to collect and interpret data.
- To provide real-time image transfers and an information sharing system between police vehicles and the command center.
- To create a high-performance, affordable solution using system-on-a-programmable-chip (SOPC) design concepts.

Existing police vehicle tracking systems can lose a suspected vehicle on screen because the vehicle's camera is fixed. Our project provides an auto-tracking solution that continuously shows the suspected vehicle's position on screen. To achieve this goal, we use an automated threshold calculation method that reduces the effect of light.

The pan-tilt camera uses a step motor. The camera moves right, left, up, and down. We designed the FPGA step motor controller to react quickly. In fact, the FPGA step motor controller's reaction time is faster than the software controller; therefore, the motor controller also helps the camera focus on the suspected vehicle continuously, even when the vehicle moves fast.

We designed an automatic voice alert system, which operates with the pursuit camera. We implemented hardware acceleration using Nios® II custom instructions for MPEG audio playback, eliminating the

need for an extra chip to perform MPEG audio decoding. We developed a µClinux audio driver for the WM8731DAC device on the Development and Education (DE2) development board.

We developed an FPGA-based on-board diagnostic system (OBD-II) interface to obtain vehicle information such as velocity and fault state from the engine control unit (ECU), which is a vehicle control system. The OBD-II interface measures the relative velocity and monitors the vehicle's state, replacing a high-cost laser measuring instrument. We designed the JPEG compression module using the `libjpeg` library, which is commonly used in Linux systems, and Altera's C-to-Hardware Acceleration (C2H) Compiler. The compression module provides image storage and wireless transfers. Our design improves the compression performance without requiring us to modify the code.

Another main feature of this project is a global, wireless, high-speed downlink packet access (HSDPA) function. The collected real-time data and image information is transferred wirelessly from the police vehicle to the command control center.

Suitability of In-Vehicle FPGAs

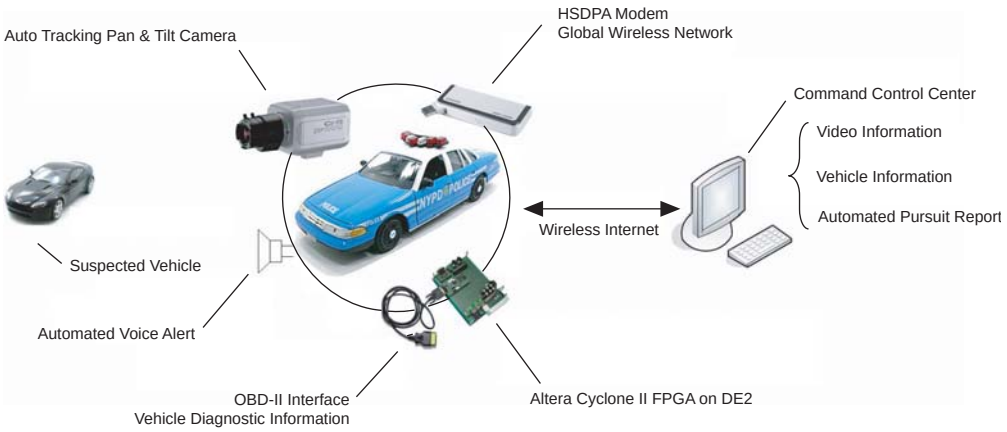
The police vehicle support system is a complex field that demands image, voice, communication, and sensor data processing. The vehicle systems increase in complexity as the amount of loaded equipment increases. For the OBD-II interface, different vehicles have different systems because their protocols are different. FPGAs are suitable for this field because they are easy to reorganize and re-synthesize.

We developed our SOPC system using the Quartus® II software and Nios II Integrated Development Environment (IDE) version 7.0. We implemented the operating system (OS) and applications using GNU tools. SOPC Builder made it easy to configure the system, and µClinux and the GNU tools offer familiar development environments.

Police Vehicle Needs

According to police pursuit policy, the officer must activate warning and recording equipment and report to the command control center upon engaging in a pursuit. It is difficult to perform all of these actions at the same time, so our project provides an automated, integrated solution. Figure 1 shows the design concept.

Figure 1. Design Concept



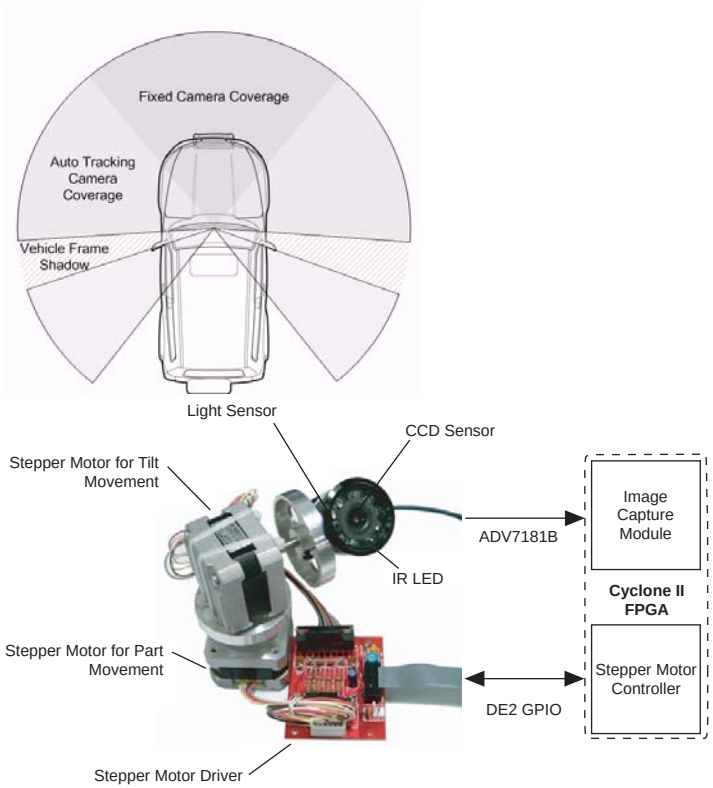
Function Description

This section describes the functionality of our design.

Auto-Tracking Camera

We made a camera module that moves horizontally and vertically so that the target vehicle's position is at the center of the screen at all times. A critical part of the pan-tilt camera module is the reaction time. A faster reaction time reduces the chance of losing the target vehicle. Figure 2 shows the pan-tilt camera module.

Figure 2. Pan-Tilt Camera Module



We chose a hardware-controlled method instead of a software-controlled method. We designed the stepper motor controller using Verilog HDL. The pan-tilt motion commands from the image processing module are transferred directly to the stepper motor controller on the FPGA. Then, the stepper motor controller receives the commands and generates operating signal pulses. Finally, the controller sends signals to each motor.

To enter tracking mode, the user aligns the camera to the target vehicle and presses a button on the DE2 board. The image processing module extracts the average color feature from the target vehicle and estimates the target vehicle's location. The pan-tilt camera tracks the vehicle as soon as the vehicle moves.

The captured 640 x 400 images are saved in USB storage and transferred to the control command center simultaneously. We tested the auto-tracking camera on the road (see Figure 3) and it works well in most cases. Our tracking mechanism does not operate at night and has a weakness with some colors because the tracking algorithm is based on color differences. Figure 4 shows the embedded systems in the vehicle.

Figure 3. Road Tracking Test

Figure 4. In-Vehicle Embedded System Configuration

Automated Voice Alert

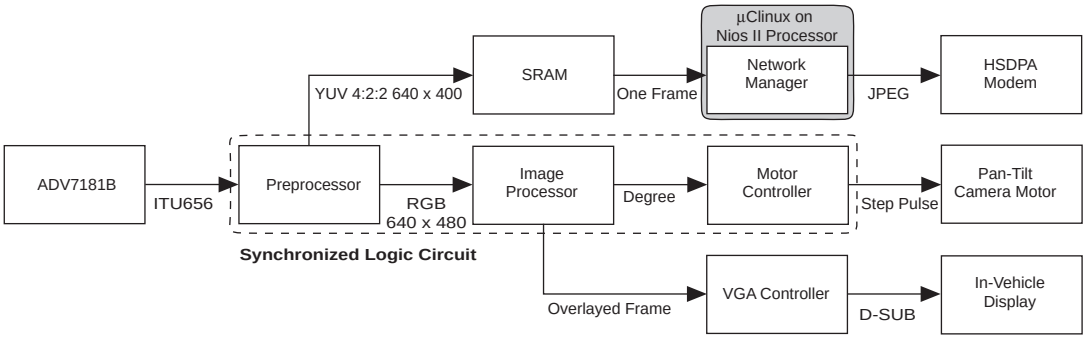
For the convenience of the officer in the vehicle, the automated voice alert system begins operating as soon as the auto-tracking camera enters tracking mode. MPEG audio data is played using a Nios II custom instruction without any extra processors. The Nios II processor operates at 100 MHz on the DE2 board.

The development board can play 128-Kbps, 44.1-KHz MPEG1 layer-3 mono-channel audio without acceleration, although the processor does need to reduce its load for multi-tasking. Therefore, we added a 64-bit multiplier, which improves the playing capacity approximately 2.5 times.

Image Capture Module

Figure 5 shows the image capture, processing, and transfer block diagram.

Figure 5. Image Capture, Processing, and Transfer Block Diagram



The camera's analog image information is converted into an ITU656 standard digital stream on the DE2 development board. This stream is used in three ways:

- It controls the auto-tracking camera's left, right, up, and down operation.
- It provides a vision sharing system and JPEG compression of the wireless transfers.
- It provides the in-vehicle display.

This image capture module preprocesses the image by modifying the image size, removing interlacing mode, and calculating the frame buffer memory address.

Image Processing Module

The camera's auto-tracking function requires a motion tracking algorithm. We used an adapted color tracking algorithm, which is easily supported on an FPGA. This algorithm calculates the average value of the changing vehicle color depending on the direction, and accordingly creates a new binary threshold value.

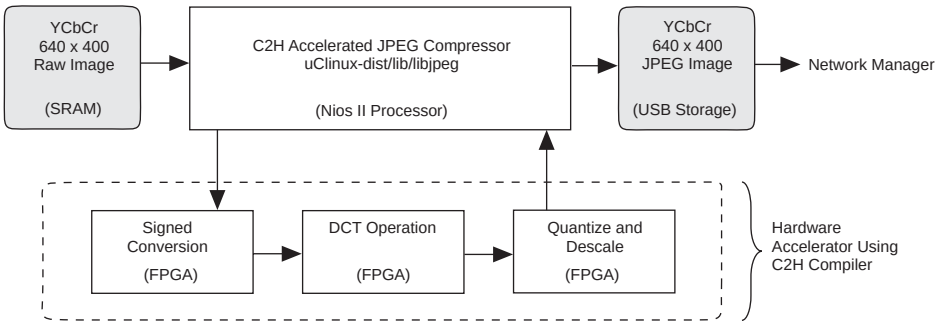
This algorithm processes by line unit, which is synchronized with the display input module in the FPGA without software processing. The design does not need outer frame buffer memory. The main benefit of this method is performance. The system transfers control commands to the motor controller every 1/30th of a second. Figure 6 shows the tracking algorithm being tested in the lab.

Figure 6. Tracking Algorithm Lab Test

C2H Accelerated JPEG Compression on μClinux

The images are 640 x 400 pixels and are compressed using JPEG. We replaced the `libjpeg` forward discrete cosine transform (DCT) function with an accelerator that we developed using the C2H Compiler. The accelerator can be accessed in the μClinux environment. Combining the C2H accelerator and μClinux is a very important feature because the acceleration operates concurrently with other tasks. By accelerating `libjpeg`, a standard library, we improved compression performance without using an extra digital signal processing (DSP) chip or other typical software. Applications using `libjpeg` have improved compression performance through recompiling without modifying any code. Figure 7 shows the JPEG compression diagram with C2H acceleration.

Figure 7. JPEG Compression Diagram



Custom OBD-II Interface

Vehicles, including police vehicles, have ECUs for system management. The ECU is a very important component in recently manufactured vehicles because it unifies the engine and various electronic controls. OBD-II is an interface that provides communication between devices and connects a computer or diagnostic tools to the ECU for vehicle maintenance.

There are many OBD-II standards depending on the vehicle manufacturer. Our project adopts the ISO9141-2 international standard. Using OBD-II, we can determine the driving speed, fuel state, and vehicle fault state. OBD-II has a 5-baud initialization procedure and a 10.4-k baud communication speed. Received information bytes have to be complemented and sent back to the ECU for communication. We used the SOPC Builder UART component because it is similar to serial communication. Figure 8 shows the DE2 board and extension equipment for the OBD-II interface. Figure 9 shows the captured image and OBD-II information display.

Figure 8. DE2 Board Extension Equipment and OBD-II Interface

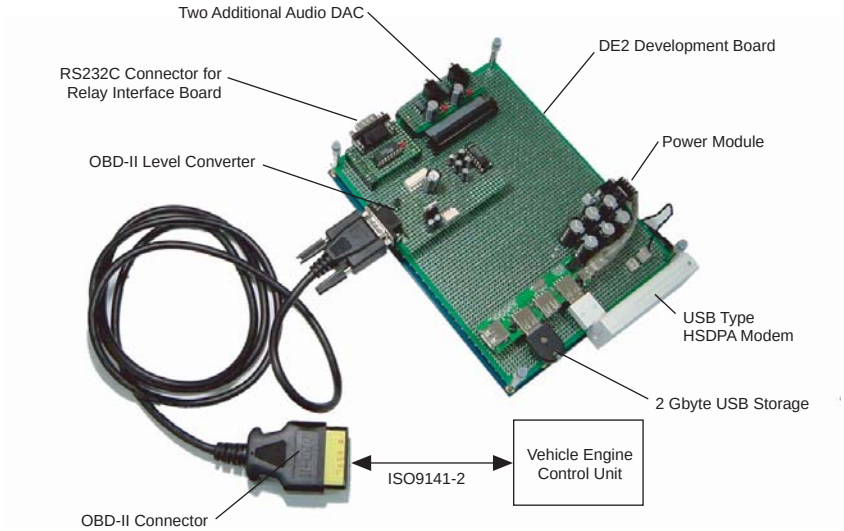
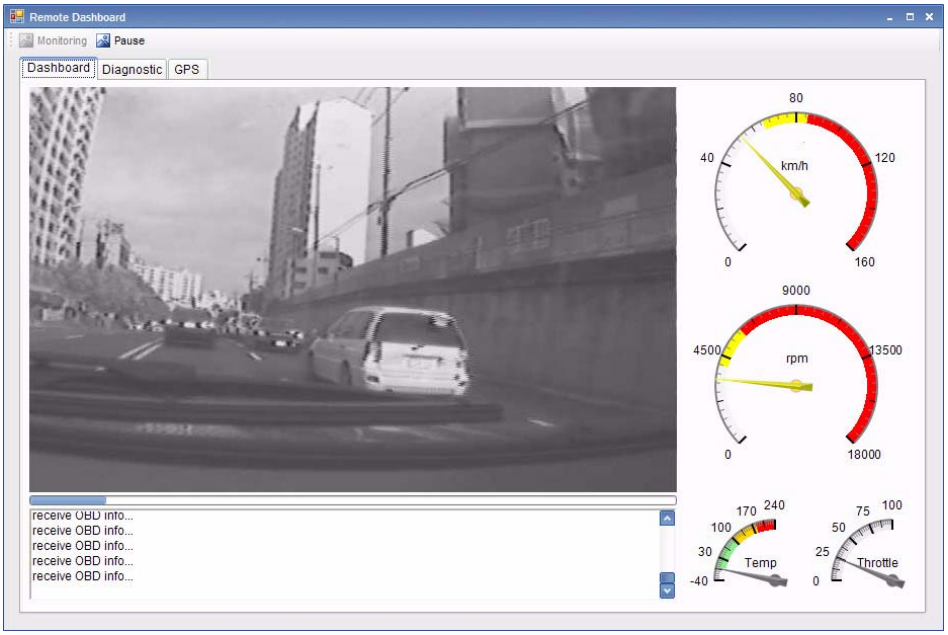


Figure 9. Captured Image and OBD-II Information Display



Performance Parameters

This section describes the performance parameters of our design. Table 1 shows the time interval between when the image processing module sends control signals and the stepmotors receive the initial operation signal. We measured the time interval using an oscilloscope. In the software program

controller, the Nios II processor receives interrupt signals and generates an operation signal, after which the stepmotor starts through the general-purpose I/O (GPIO) interface.

Table 1. Stepper Motor Pulse Generation Method Comparison

Latency	Using Interrupt Routine and Timer	Full FPGA Controlled Method
Average latency (µs)	60	35

The velocity of the auto-tracking camera is mostly determined by the image processing performance. Table 2 shows the tested frame rate of each tracking algorithm for different platforms. The DE2 board's frame rate is almost 60 frames per second (fps) because the image processing module operates in interlace mode; however, we show 29 fps, which is the number of effective frames.

Table 2. Tracking Algorithm Performance Comparison

	PC	ARM	DE2 Board
Clock	2.4 GHz	520 MHz	100 MHz
Implementation	Software	Software	Full FPGA
Frame rate (fps)	29	10	29

Table 3 shows the compression performance of the `libjpeg` DCT function accelerated using the C2H Compiler. The 640 x 400 x 24-bit bitmaps are compressed 20 times for accurate measurement. Compiling with the C2H Compiler shows worse performance than the design that has no accelerator. To solve this problem, we changed the buffer management method, resulting in a 4x performance improvement after modifying the DCT function.

Table 3. JPEG Compression Performance Comparison

	libjpeg	libjpeg	mod-libjpeg
Compiler	gcc	gcc, C2H	gcc, C2H
Accelerated function	N/A	forward_dct()	forward_dct()
Compression time (seconds)	210	281	52

Table 4 shows the modem performance test. When we designed the system, we were concerned about low performance when using a USB modem with the µClinux system. According to our test, it showed almost the same network performance as the PC environment.

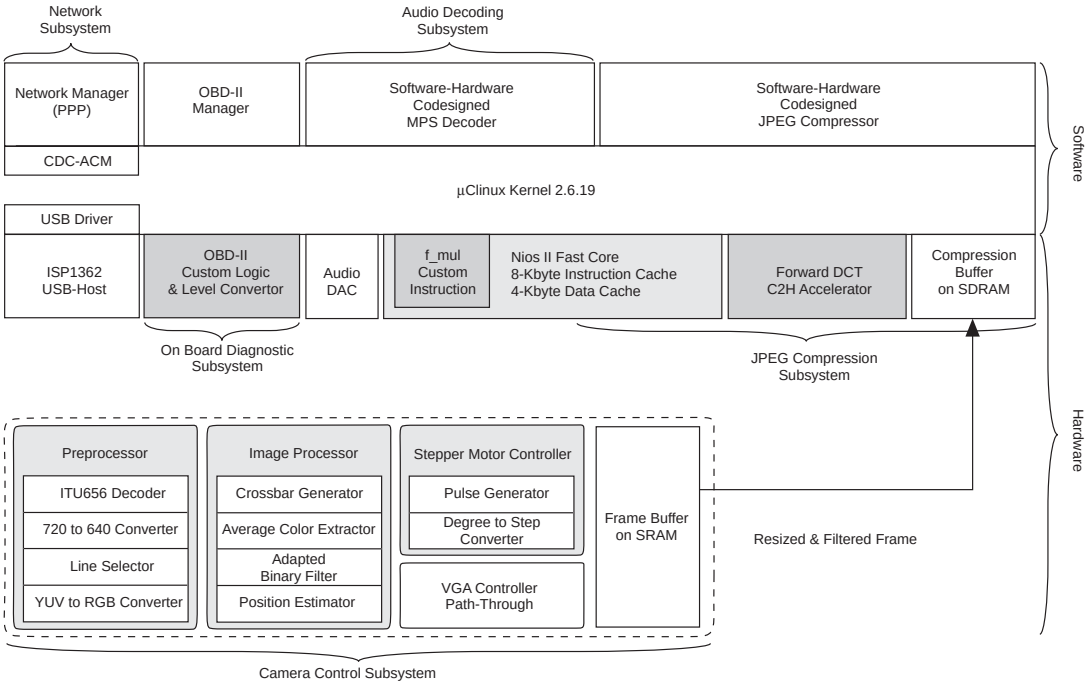
Table 4. USB HSDPA Modem Performance Test on µClinux

	Expected (PC)	DE2 Board
OS	Linux 2.6.22	µClinux 2.6.19
Download (Kbytes/second)	32	30
Upload (Kbytes/second)	28	25

Design Architecture

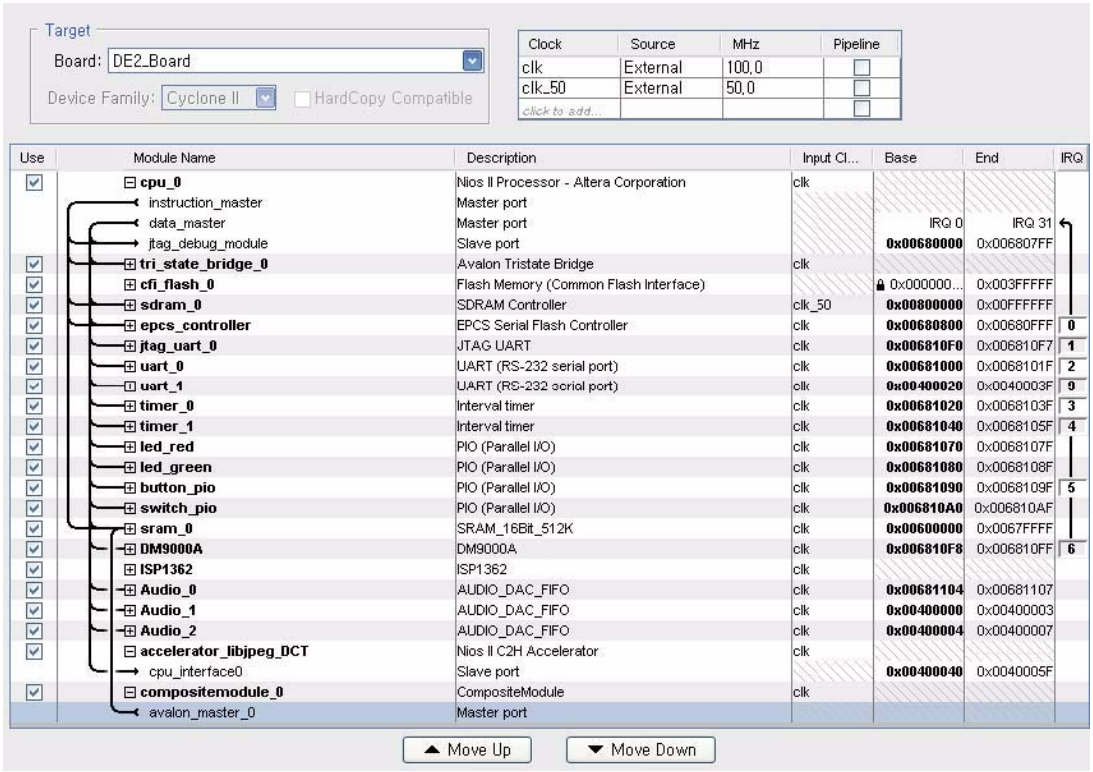
Figure 10 shows the system architecture.

Figure 10. System Architecture



µClinux controls the software systems. The FPGA contains the camera control system and sub-systems, including the image processing modules. Grey boxes in Figure 10 indicate custom-designed SOPC components. Figure 11 shows the system in SOPC Builder.

Figure 11. SOPC Builder Configuration



We moved the `libjpeg` DCT function into the C2H accelerator. We combined the image processing module, VGA controller, and stepmotor controller into a unique SOPC component. The design uses 31,000 logic elements (LEs). Figure 12 shows the Quartus II compilation report.

Figure 12. Quartus II Compilation Report

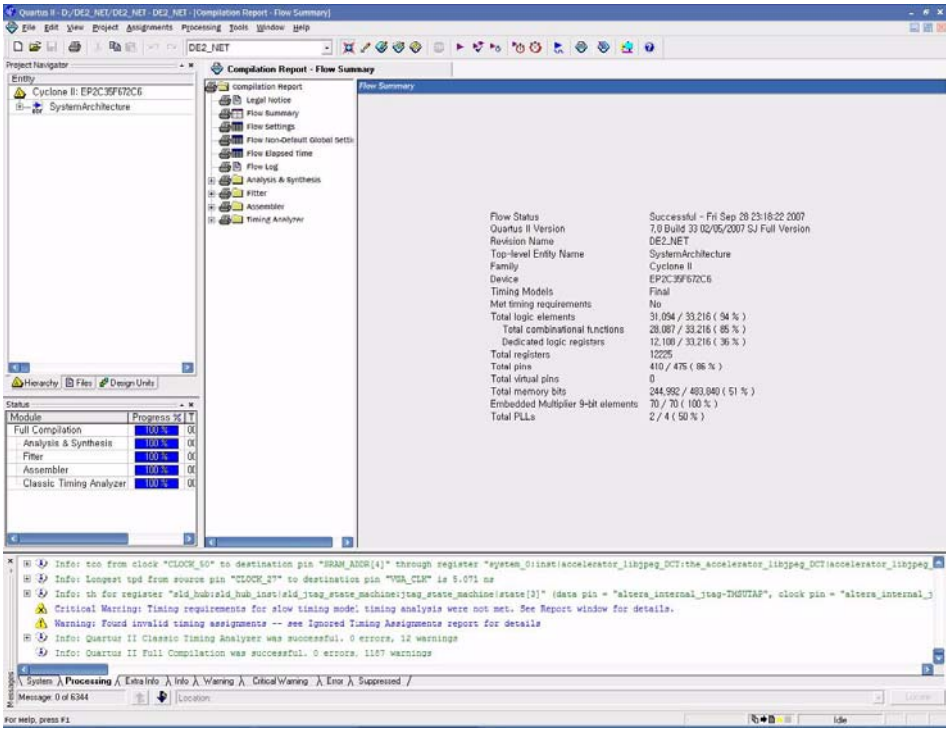


Figure 13 shows the top-level entity.

Figure 13. System Top-Level Entity

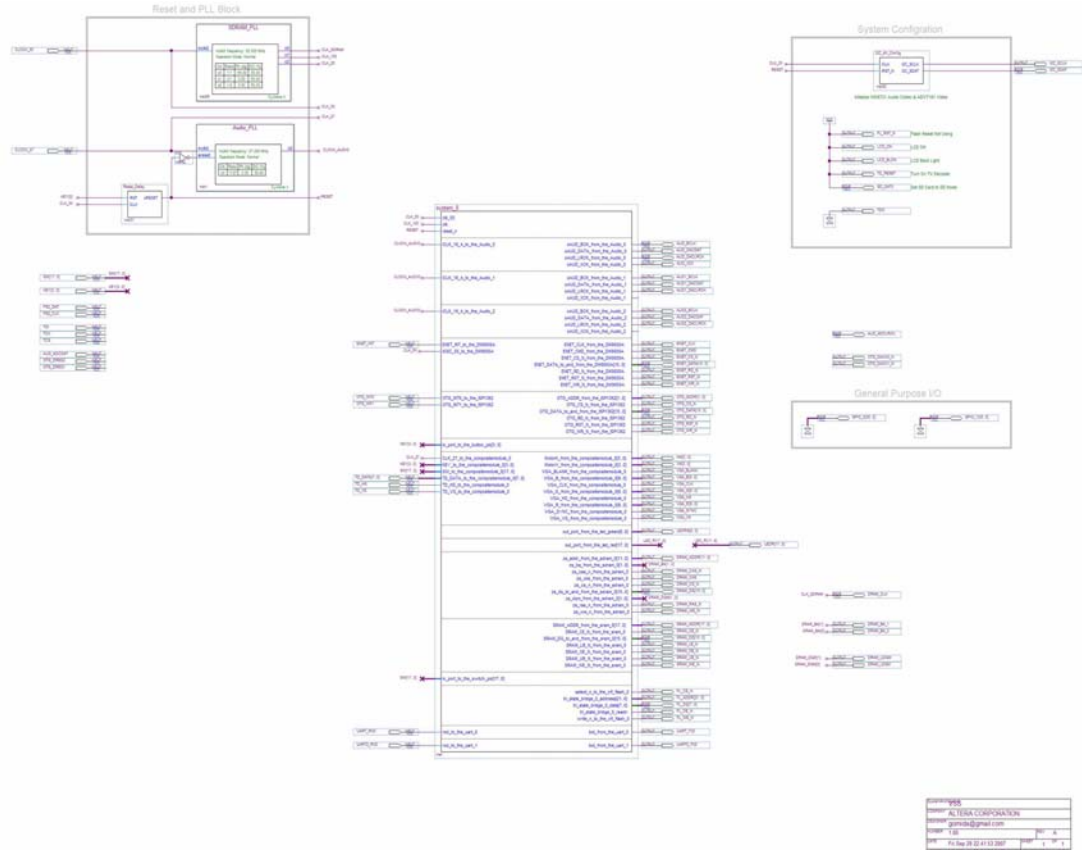
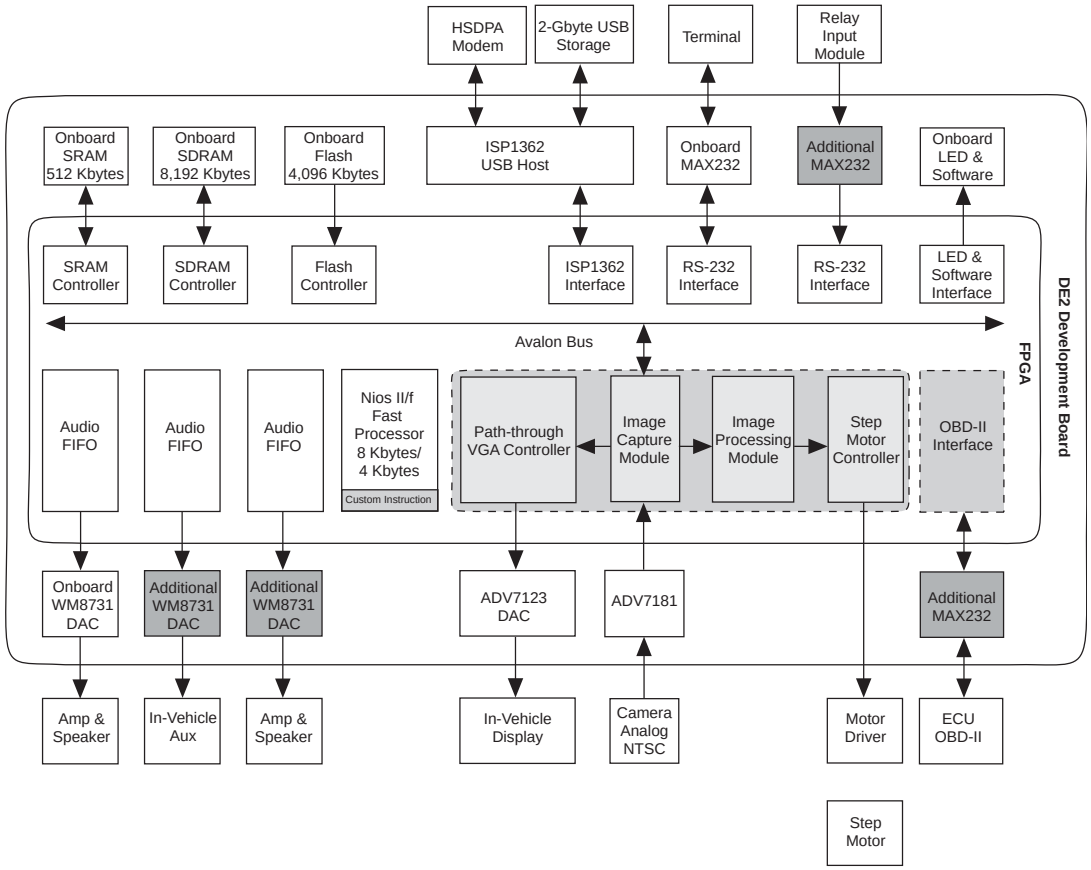
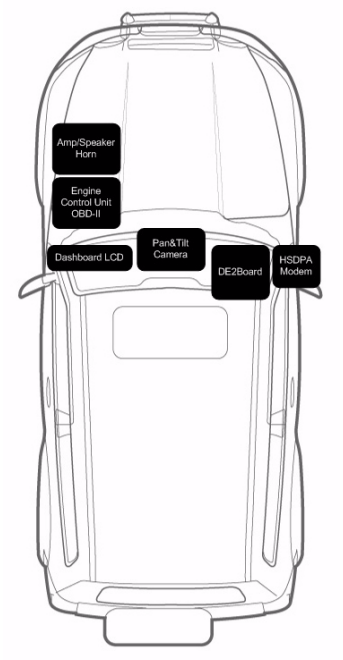


Figure 14 shows the on-chip and in-vehicle block diagram.

Figure 14. On-Chip and In-Vehicle Configuration





Design Description

This section describes our implementation method and the steps we used to build our design.

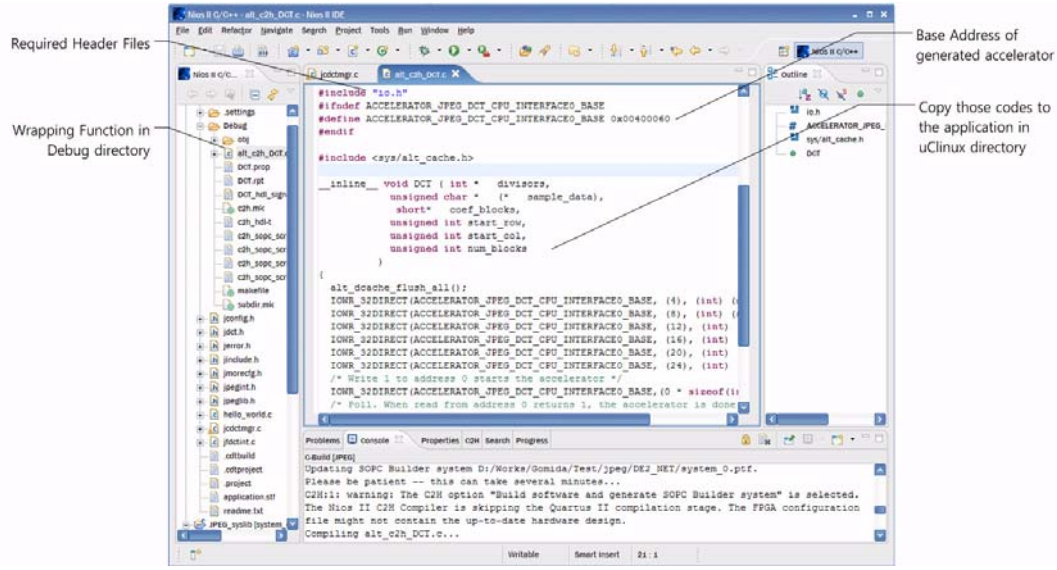
Combining µClinux and C2H

Using an operating system offers development flexibility for complex multi-device systems. The µClinux kernel is appropriate for non-memory management unit (MMU) processors. Because µClinux does not have an MMU, the Nios II processor application that accesses the custom hardware accelerator is simpler. We compiled the code we wrote in the Nios II IDE to operate in a multi-tasking environment under µClinux with few or no changes because there is no limit writing to memory-mapped addresses in µClinux. We used the C2H accelerator on µClinux with typical techniques. The following discussion describes the steps required to move the C2H accelerator from the Nios IDE into µClinux (see Figure 15).

First we made a temporary project. Then, we compiled and generated the accelerator in the Nios II IDE. After generation, the accelerator's wrapping function is saved in the **debug** directory. We copied the header files and wrapper to the µClinux development directory and programmed the FPGA.

Next, we compiled the accelerated application using Nios II gcc tools with the `elf2flt` option, we first made sure that the required header files such as **system.h** and **io.h** existed. We then copied the generated execution file to the development board. This implementation is faster than a software-only system in most cases. Unfortunately, we faced a performance problem when converting the `libjpeg` DCT function to the accelerator. "Optimizing the JPEG Library for the C2H Compiler" on page 120 describes how we solved the performance problem.

Figure 15. Moving the C2H Accelerator Wrapper into µClinux from the Nios II IDE



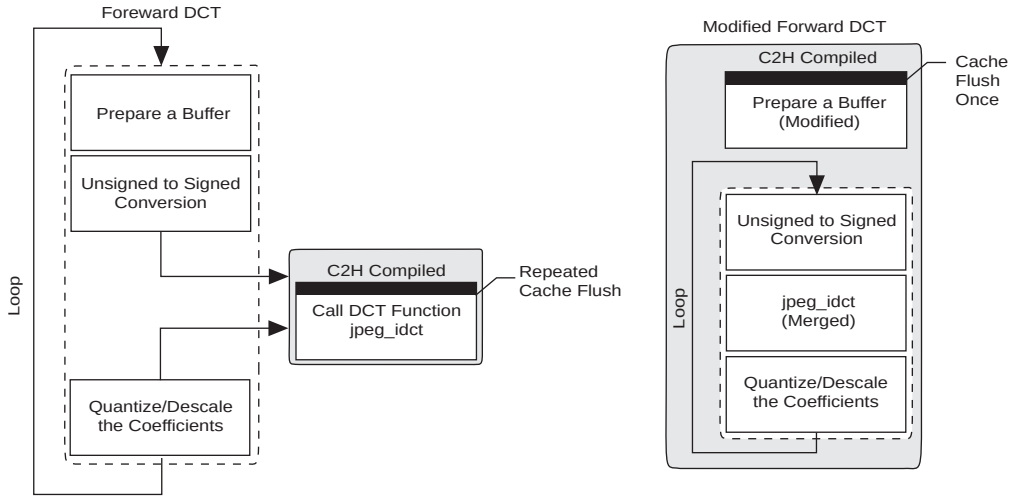
Optimizing the JPEG Library for the C2H Compiler

Generally, developers use a digital signal processor for JPEG compression, but a processor requires software to support it. Using the C2H Compiler to accelerate `libjpeg` is an interesting solution because many existing applications use `libjpeg`, which is a standard JPEG library.

When converting an original DCT function with the C2H Compiler, however, the function had lower performance than the software-only design. Flushing the data cache, which occurs every 64 bytes of data processing work, caused the performance problem. Therefore, we designed an optimized buffer

management system that was suitable for the C2H Compiler. This solution improved the performance 4 times. Figure 16 shows the optimized DCT function block diagram.

Figure 16. Optimized DCT Function Block Diagram



Creating the Custom SOPC Component

We combined the image processing module, VGA controller, and stepmotor controller as a unique component because these functions need to work together closely. We designed each block separately in Verilog HDL and added them to SOPC Builder as components. The components write image data in the SRAM as an Avalon® master. Figure 17 shows the custom component in SOPC Builder and Figure 18 shows the multiplier custom instruction.

Figure 17. Custom SOPC Builder Component

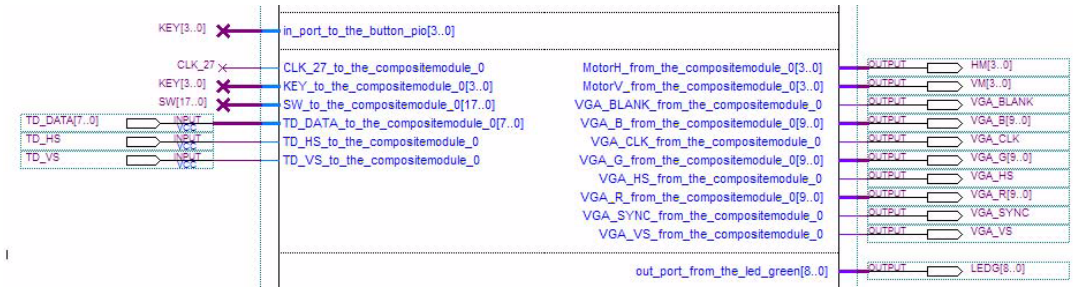
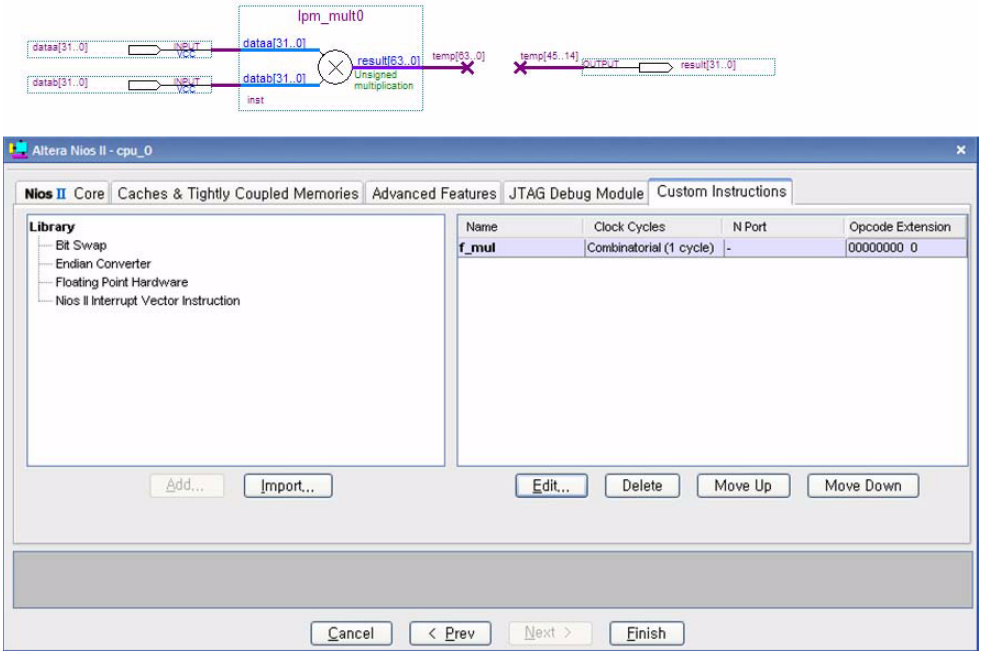


Figure 18. 64-Bit Multiplier Custom Instruction



MPEG Audio Decoding Custom Instruction

There are three main issues to consider when playing MPEG audio in a Nios II processor/µClinux environment

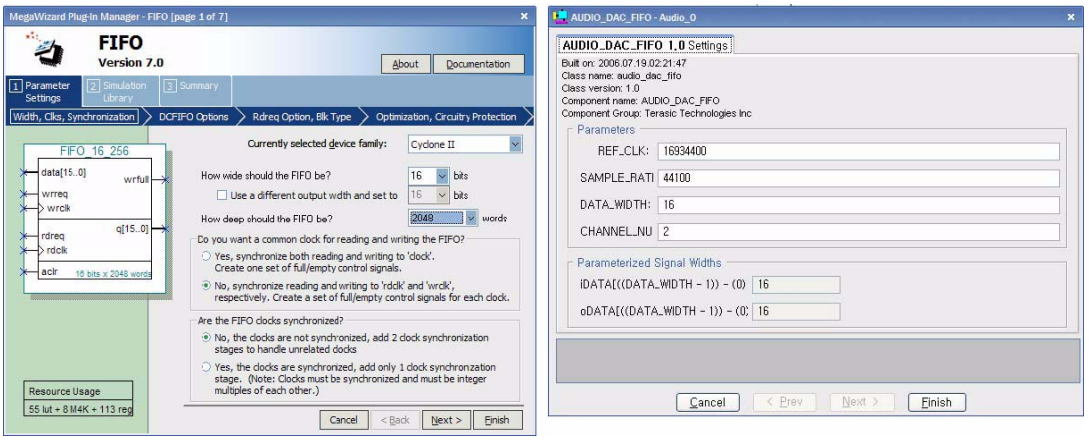
- Processor performance
- FIFO buffer size
- µClinux device driver output

The design had poor performance when the 100-MHz Nios II processor in the Cyclone® II device decoded the stereo 128-Kbps, 44.1-KHz MPEG1 layer 3 audio. With a big enough FIFO buffer, the system could play mono-channel audio but the CPU allocated all of its time to playing audio.

To solve this problem, we added a 64-bit multiplier custom instruction to the Nios II processor to implement a 64-bit multiply calculation that is frequently used by the `libmad` library. With this change, we increased the audio playback performance about 2.5 times and decreased the number of clock cycles required for the calculation.

Other issues can also cause audio playback problems, such as a bad sampling rate, a lack of buffer space, and a multi-tasking environment. Figure 19 shows a configuration that solves this problem with a 17-MHz audio reference clock.

Figure 19. DAC FIFO Setting for 44.1-KHz MPEG Audio



Design Features

The most fascinating part of this system is its integrity. One FPGA performs the whole function, including image processing, compression, transferring, MPEG audio decoding, step motor control, and OBD communication. Each block is an SOPC Builder component, so it is very easy to recycle components for different projects.

We designed the device driver so that all systems can operate in the µClinux environment. The access method connects the µClinux application, custom instruction accelerator, and C2H technology. We unified the software as well as the hardware.

The main image processing feature minimizes memory access. We only used the frame buffer memory for JPEG compression because the image processing is implemented by a line unit. Eventually, we could dramatically save Avalon bus bandwidth. Table 5 shows the design's key features.

Table 5. Key Features and Related Modules

Feature	Related Module	Implementation
Auto-Tracking Camera	Stepper Motor Controller	Custom SOPC Builder Component
	Image Capture Module	
	Image Processing Module	
Automated Voice Alert	MPEG Decoder	Nios II Custom Instruction
	WM8731 DAC Driver for µClinux	Character Device Driver
Command Control Center (Remote Dashboard)	OBD-II Interface Module	Custom SOPC Builder Component
	JPEG Compressor	C2H Compiler

Conclusion

We developed and tested a police vehicle support system representing an application in the telematics field. The auto tracking camera tracks a typical vehicle and positions it at the center of the screen at all times. The OBD interface is based on FPGA technology and captures information about the vehicle's state. A remote system distantly verifies the vehicle's image and state information and communicates using a global wireless HSDPA module.

We started the project with a fixed hardware platform and limited resources. FPGAs offer amazing technology that can easily adapt to design configuration changes. Our simple, effective design method involved modifying the hardware design when there were performance problems or problems that could not be solved in software. We noted that FPGAs are particularly valuable in the field of image processing.

We used the Altera® C2H Compiler for JPEG compression. We started out with the `libjpeg` DCT function, which is commonly used. We achieve high performance in our design by accelerating the `libjpeg` DCT function with C2H technology. We believe that the C2H Compiler, which translates C programs into HDL, is driving a changing software paradigm. The C2H Compiler has challenges, but we expect it will contribute to the development of software-to-hardware translation technology.