A/V Monitoring System Rides Virtex-5

State-of-the-art FPGA forms the basis for a multi-input audio/video remote-monitoring application.

by Manish Desai
Project Lead – ASIC - FGPA
eInfochips
manish.desai@einfochips.com

The growing demand for high-end security systems with video monitoring for a variety of applications has fueled a need for embedded systems that can quickly capture multiple audio/video channels, process and compress the information and send it to a central monitoring system via a high-speed Internet connection or host PC interface.

The new state-of-the-art Xilinx® Virtex®-5 FPGA offers an exciting opportunity for single- or few-chip solutions to such A/V monitoring applications, thanks to advanced features such as a high-end system and networking interface, built-in processor, high-speed serdes, advanced clock management and pre-bit deskew in the I/O blocks. It might seem as if all these sophisticated features could complicate the design process. But in fact, early-stage planning can streamline the job while ensuring effective usage of the FPGA's available resources.

Let's examine some of the challenges of implementing designs in Virtex-5 FPGAs and delineate techniques to get the most out of its feature set, as demonstrated by a recent project. The process involved a number of steps, starting with choosing the right Virtex-5 for the application. Other concerns included clock requirement analysis, initial floor planning, core generation and IP integration, timing considerations and constraint definition, all the way to post-place-and-route timing analysis and timing fixes. The Virtex-5 FPGA's hardwired macros proved central, as did its I/O blocks and source-synchronous design. For the sake of this article, we'll assume that we've assembled the IP blocks and that they are either ready to use or already generated with CORE Generator™.

Picking the Right Device for the Job

Most audio/video capture devices support a single channel and generate source-synchronous digital in the Y/Cr/Cb data format. Although DSPs are capable of capturing digital audio/video and can perform digital signal-processing tasks, they typically support only a few channels. Therefore, in this design we chose an FPGA, which proved to be a good alternative for both multiple-channel inputs and signal-processing tasks.

Figure 1 shows a typical security/video monitoring system with a 3G/SD/HD/SDI video interface. For this design, the camera sends information in 3G-SDI format to the board, which in turn collects the data and converts it into 10-bit (Y/Cr/Cb formatted) source-synchronous video data (10/20-bit interface) at a maximum clock frequency of 145.5 MHz. It handles source-synchronous audio data at a maximum clock frequency of 96 kHz.

We used the Virtex-5 to capture the video and audio data, and then synchronize it with the internal FPGA clock and store it in DDR memory. (The memory was 512 Mbytes and 32 bits wide, so the FPGA had to support expandability up to 2 Gbytes.) The internal logic of the FPGA performs compression (if enabled) and sends data over Ethernet; alternatively, it sends uncompressed data to the host PC for storage and further processing via the PCI/PCI Express® link.

For our design, the FPGA had to support up to 10 digital audio/video source-synchronous input channels (20-bit source-synchronous Y/Cr/Cb data format), and it had to be configurable for the SD/HD data format. It also needed a PCI Express link with support for four lanes (default, one lane), single-channel 10/100/1,000 Ethernet, a DDR memory controller with interface capacity of 128 Mbytes to 1 Gbyte and an embedded microcontroller in the form of a soft core for configuration control. Other requirements included an A/V signal-processing and optional compression algorithm, a central control unit with an advanced DMA engine and one A/V output port connected to VGA or a standard TV.

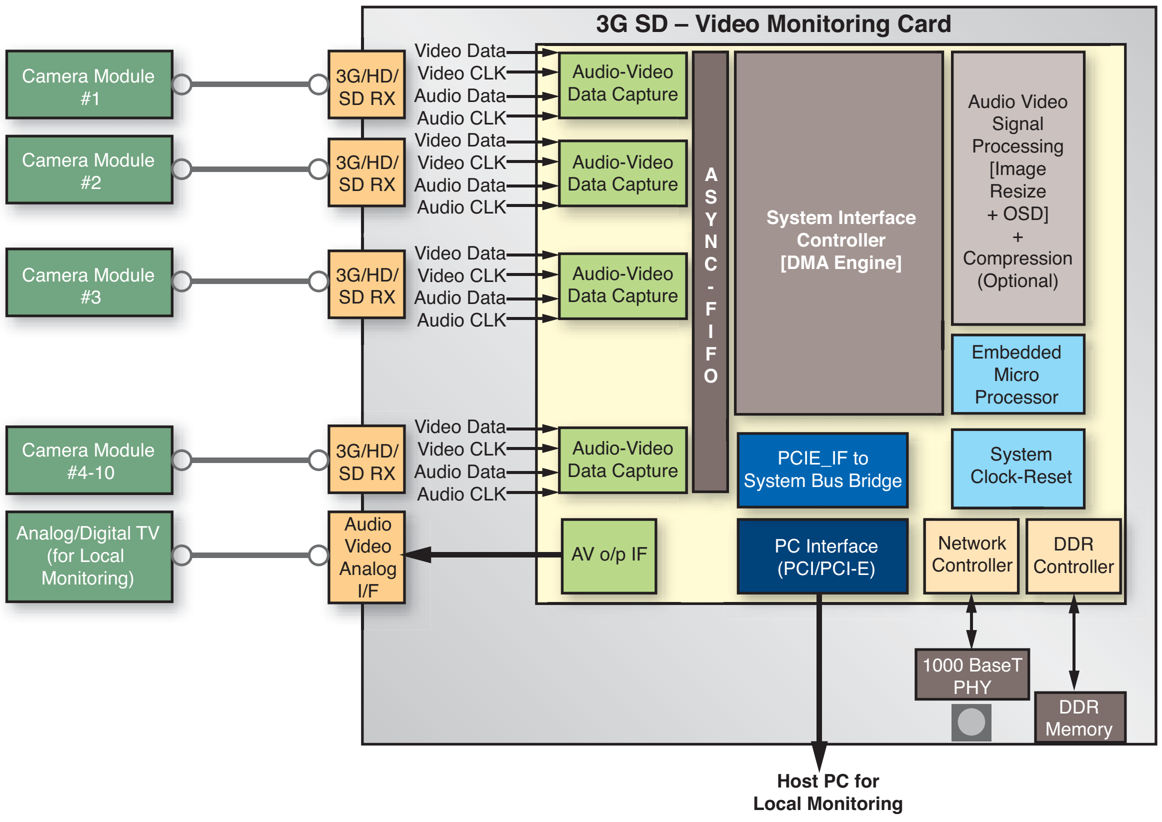

Figure 1 – Typical Video Monitoring System Block

In looking over the data sheets from various FPGA vendors, it became clear that the FPGA that best suited our requirements was the Xilinx Virtex-5 XCVSX95T-FF1136.

To meet these specifications, we had to consider several factors when implementing the design. Chief among them were clock requirement analysis, initial floor planning, core generation and IP integration, timing-constraint definition and post-place-and route timing analysis and timing fix. But the first decision was choice of the FPGA.

Selection of the FPGA

We based our selection on a number of factors. The device needed to meet our estimated I/O requirements, and it had to have an appropriate number of logic cells, a suitable block RAM size as well as a number of clock buffers and clock management devices such as phase-locked loops (PLLs), digital clock management (DCM) and multiply-accumulate blocks. In looking over the data sheets from various FPGA vendors, it became clear that the FPGA that best suited our requirements was the Xilinx Virtex-5 XCVSX95T-FF1136.

The Virtex-5 contains all the design features we needed. It is equipped with 640 I/Os and an additional multi-gigabit transceiver (MGT) for PCI Express, along with Gigabit Ethernet media-access control (MAC) and RocketIO™ or an external PHY for local- and wide-area networking. A 3G-SDI receiver handles source-synchronous data capture. The PCI Express one-, four- or eight-lane interface links with a host PC for processing and storage requirements.

The Virtex-5 also boasts a built-in soft-core processor for configuration control, high-speed multiply-accumulate units (multiple DSP48E blocks) for digital signal processing and compression-algorithm deployment. Advanced clock management is accomplished by means of a global clock buffer, regional clock buffer and I/O clock buffer with clocking module—namely, a PLL and two DCMs. Finally, it offers asynchronous FIFO and digital signal processing through more than 200 block RAMs of 18 kbits.

Clock Requirement Analysis

Once we selected the FPGA, we began the design process by analyzing clocking requirements before mapping the signals to the I/O bank or I/O pins. Over the years, we've learned to do this step early—it often proves to be the most important part of the whole system design and plays a major role in determining overall performance.

For the analysis of clock requirements, it is important to consider a few factors:

- Does the FPGA have sufficient clock-capable I/O and global clock I/O lines?
- Are there enough PLLs, DCMs and global clock buffers?
- Does the global clock I/O/buffer support the maximum required frequency?

For example, this design's clocking requirements consisted of one global system clock running at 150 to 200 MHz with PLLs used by all internal logic for processing; one global clock with a PLL/DCM PCI Express link running at 250 MHz; one global clock buffer/PLL and DCM for the Ethernet MAC running at 250 MHz; 16 regional/local clock I/Os for the source-sync video clock running at 145 MHz; one global capture clock for audio data/clock signals (48 kHz/96 kHz); one regional clock-capable pin for the DDR memory interface running at 200 MHz, and one 200-MHz clock (generated by the PLL/DCM) for pre-bit deskew in I/O blocks.

In total, we needed four to six global clock buffers and 16 local clock buffers. The FPGA XCVSX95T-FF1136 offers 20 global clock input pins and four clock-capable I/Os in each bank. The device has 14 banks with 40 pins, each supporting regional clock input buffers, and four banks containing 20 pins, each supporting global clock input buffers. You can directly connect the clock-capable pins of the I/O banks to regional or I/O buffers, and use them in specific or adjacent regions. In addition, each GTP/MGT has a reference-clock input pin.

Initial Floor Planning

After performing the clock analysis, we created an initial floor plan. This is a critical phase of the design, because the decisions made at this point will determine whether the final design is going to meet timing. The bank selection and pin assignment are important steps of floor planning. How you handle them depends on the placement of other components around the FPGA.

The Virtex-5 FPGA has a total of 18 I/O banks on which to map various input/outputs. A few I/O banks support 20 input/outputs or 10 global clocks. Most of the other banks support 40 input/outputs, on which there are four input and eight output clock-capable pins.

IOBANK #3 and IOBANK #4 each support 10 single-ended/differential global clock inputs. Each bank supports 20 pins. Any pins not used for clock/reset input can be employed for general-purpose I/O. Two other banks, IOBANK #1 and IOBANK #2, are close to the center of the FPGA and each supports 20 I/O pins. Xilinx dictates that you must map all single-ended clock inputs to positive global clock input pins.

Meanwhile, the upper and lower halves of the FPGA consist of three clocking modules (CMTs): a PLL and two DCMs. We needed to ensure that we properly mapped all global clock signals that required a PLL in the upper and lower half of the device, such that the design had a direct connection from the global clock input buffers to the PLLs. We then used the remaining 14 I/O banks, supporting 40 I/O lines, in single-ended/differential mode. Each bank consists of four single-ended and eight differential clock-capable pins. We could then map or connect the clock-capable pins to regional or I/O clock buffers.

Normally, it is good to use these clock-capable pins and regional buffers (BUFR) to map source-synchronous clock inputs. The regional buffer has a lower skew and can access three regions (one where the regional buffer is located, one above and one below). But for bank selection of source-synchronous data, we prefer to use a single I/O bank. If we need additional I/O, it is better to use I/O banks for data signals that we've previously mapped to adjacent banks. (For package information, refer to *ug195.pdf* from the Xilinx Web site.)

We followed several steps for the initial floor planning of the design. First, we placed the system clock in the upper half and then placed the audio capture (optional) clock in the lower half. We locked the CMT of each half for the I/O bank 3/4 requirements. This map ensures that each half is left with two PLL/DCMs (CMTs) that we can use for the PCI Express and Gigabit Ethernet MAC (SGMII) features.

Because we mapped synchronous data to banks that consisted of regional clocks, we mapped 10 audio/video channel inputs on the remaining I/O banks. Each video channel consisted of 20 data lines, three control signals and video clock inputs. Meanwhile, each audio channel consisted of four data signals, three control signals and one audio clock signal. This made a total requirement of 32 signals with at least two clock-capable pins (the FPGA's 14 banks can support 40 pins and four clock-capable pins).

For this design, 10 A/V channels use 10 I/O banks. We mapped the video clock and audio clock to clock-capable pins to ensure we effectively used the regional and I/O clock buffers. Based on the PCB requirements, we selected for audio/video channels banks 5, 6, 13, 17, 18, 19, 20, 22 and 25.

For DDR memory, the design supports a 32-bit data bus, 14 address lines and a few control lines. We needed 85 to 90 signals to map the DDR memory interface. As per the PCB layout, we used I/O banks 11, 23 and 15 to map all DDR I/O signals. Since DDR memory works on the system clock, we chose to map the read data strobe signal generated by the DDR to clock-enabled I/O lines.

Xilinx offers the PCI Express and Gigabit Ethernet (GbE) MAC as hard macros. The Xilinx CORE Generator tool generates the proper IP core with the combination of hard macro, block RAM and some advanced RTL logic to render the blocks usable. The tool also provides detailed constraints for pin mapping, the PLL/DCM and timing for a specific Xilinx FPGA. We advise using the recommended pin definitions as described in the release notes or UCF file that CORE Generator creates for your design. Also, you can use Xilinx's Plan-Ahead™ tool to confirm or cross-check any pin mapping you've defined manually.

Core Generation and IP Integration

The task of generating cores with CORE Generator and integrating intellectual property can be tricky. Let's examine some of the challenges involved in generating and integrating the CMT, ASYNC FIFO, block RAM, PCI Express, GbE MAC and DSP48E blocks. (For more detailed information on the PCI Express and GbE MAC blocks, visit the Xilinx Web site to make sure you have the most recent version of CORE Generator and the latest IP.)

The Virtex-5 supports various configurations of clocking modules that you can generate with the CORE Generator utility. They include filter clock jitter PLLs, a PLL-DCM pair with filter clock jitters, a PLL-DCM pair or DCM for output dual-data rate (ODDR), a standard phase-shift clock DCM and dynamic clock-switching PLLs.

To generate PLLs, you first need to see whether the input is single-ended or differential (in the example design, it is all single-ended). Then you must determine whether clock jitter is appropriate (in our case, it was 120 picoseconds) and whether you've used the global buffer to buffer all the outputs.

Each PLL can generate up to six different frequency clocks. In our case, the design needed four 200-MHz system clocks, each with 0, 90, 180 and 270 degrees of phase, and one audio capture clock of 19.048 MHz or 39.096 MHz.

To drive clocks using ODDR flip-flops in source-synchronous outputs, we implemented a DCM that drives the ODDR flip-flops for forward clocking. This DCM runs in parallel to the DCM we used for internal clocking.

We generated the ASYCN FIFO or block RAM using CORE Generator and supported ECC with interrupting logic on an embedded microprocessor core to perform data error detection.

While generating the PCI Express core, we had to ensure the reference clock had the same performance as the PC motherboard's PCI Express slot output (that is, 100 MHz). Also, we needed to define how many base address registers (BARs) the core needed and whether the BARs were memory mapped or I/O mapped. We used the BAR monitor, which helps in generating BAR hits, for address decoding.

During the design of the bridge between PCI Express and the system local bus, we used the BARs—which act as memory or I/O region chip select—to access memory-mapped or I/O-mapped registers or block RAM. We designed the bridge logic in such a way as to make sure that the core and bus properly accessed all the register or block RAM. The Xilinx PCI Express core also has a default ROM expansion capability, and to accommodate it we had to implement an address and map inside the bridge. Bit position 6 of the BAR hit points in this expansion ROM area and the internal interface must respond to these BAR hits.

If any of the above is missing, the host PC won't get any response if it tries to communicate and perform a read transaction. It will enter an unknown state or generate an unrecoverable error.

We used the CoreGen utility to generate the GbE MAC with an RGMII/SGMII external interface. We used the built-in GTP module to communicate with selected PHY devices. The GbE MAC supports the MDIO interface to configure external physical devices, a host interface and a 16-bit single-channel client interface.

The DSP48E block, for its part, is a 25x18-bit multiplier and 48-bit hard-macro accumulator. You can use it directly as an instance or, by mapping the multiply-accumulate, add and subtract functionality

The Virtex-5's pre-bit deskew capability, which is built into all I/O blocks (IODELAY primitive), helped us to meet setup-and-hold requirements at input and output stage.

implemented in RTL logic with Xilinx tools. We recommend using standard RTL logic to implement the multiply-accumulate, ADDR and multiplier. Include the design constraints during synthesis and placement and routing.

For IP integration, be sure to have a separate clock-reset module for each FPGA. The asynchronous reset must be synchronous with each and every clock, both global and regional. Internally, the reset signal is asserted asynchronously and deasserted synchronously with respect to specific clocks, and its output is applied to the specific module to which the clock belongs. Make sure you have connected all the global input clocks to the PLL/DCM core generated by CoreGen.

Also, be sure you've connected the regional clock to BUFR/BUFIO. In addition, to keep your placement-and-routing tool from using unnecessary routing resources, make sure you generate only the necessary reset signals. You need to ensure that PLL/DCM lock conditions are brought to external pins or to the configuration register. In our case, we only connected the 200-MHz system clock PLL lock to the I/O pins.

Definition of the IOB (input, output or both input and output) synthesis and mapping constraints must be part of the FPGA's top RTL module. We instantiated the topmost module of core logic inside the FPGA's top module. It must communicate with the external interface via these IOB definitions only.

IOB definitions consist of IBUF, OBUF, IBUFDS, OBUFDS and the like. Each in turn consists of supported user-defined parameters for IO_STANDARD (LVTTL, LVCMOS, etc). We used the instance definition of the above to map external I/O signals with the topmost RTL module signals.

Since we were designing with high-speed source-synchronous inputs and outputs, the Virtex-5's pre-bit deskew capability, which is built into all I/O blocks (IODELAY primitive), helped us to meet setup-and-hold requirements at input and output stage. For source-synchronous inputs, the source-synchronous clock uses BUFIO or BUFR, and it introduces additional delay. To compensate for this delay, we drove data and clock inputs via an IODELAY instance that we configured in input delay mode with known delay counts. Changing the delay count value helped us meet timing requirements at the input stage.

Similarly at the output stage, as synchronous clock signals are driven with data, we needed to make sure that data and clock signals were driven so as to meet the setup-and-hold of an FPGA or ASIC at the other end. We used IODELAY instances configured in an output delay mode with a known delay count value for both clock and data outputs. The IODELAY needs an IODELAYCTRL primitive instance at the top of the FPGA. The 200-MHz input clocking to the IODELAYCTRL instance creates a delay count precision of 70 ps on IODELAY.

Timing Consideration and Constraints Definitions

After generating and implementing the IP, the next step was to perform timing. We constrained all the input clocks for period, jitter and input offset delays, and set all output delays with respect to the source clock and input-to-output delay. We then created the timing and placement constraints in Xilinx User Constraint Files (UCFs).

We constrained all the input clocks to specific frequencies and also defined the jitter input using the following UCF code:

```
NET "i_clk_200_s"
TNM_NET = "IN_200_CLKGRP";
TIMESPEC "IN_200_CLKGRP"
= PERIOD 5 ns HIGH 50%
INPUT_JITTER 0.1 ns
```

With respect to source-synchronous data, we can set the input clock to a 0-degree phase shift or 180-degree phase shift, in the case of SDR, and 90-degree phase shift in case of DDR. Figure 2 shows the source-synchronous DDR data input timing with the clock at a 90-degree phase

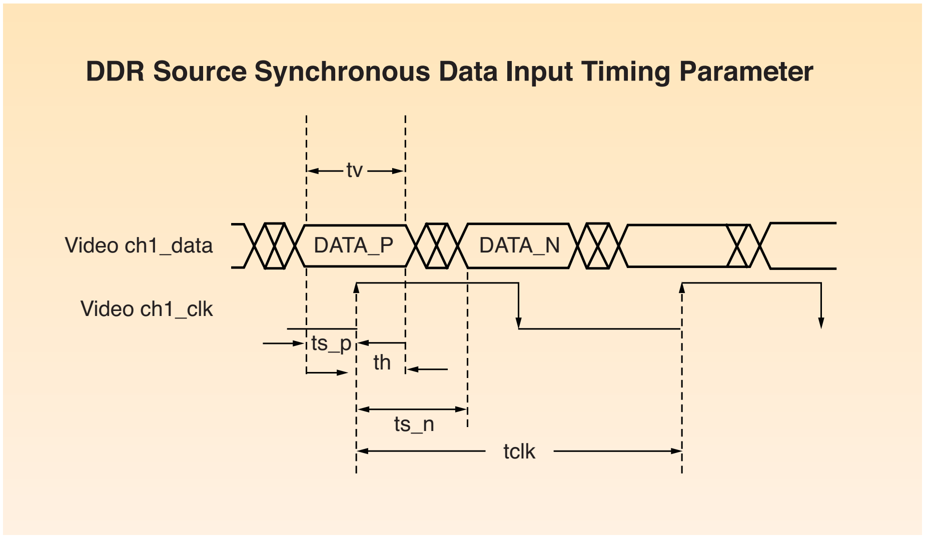

Figure 2 – Timing Diagram of DDR Inputs

Timing Parameter	Min (ns)	Max (ns)	Description
Tclk	6.6	–	Clock period
ts_p (Setup Time)	0.5	1	Input Setup time for pos-edge of clock with respect to posedge
ts_p (Setup Time)	-1.5	-2.0	Input Setup time for neg-edge of clock with respect to posedge
th (Hold Time)	0.5	–	Input Hold time
tv (Data Valid Window)	1	1.5	Data Valid window

Table 1 – External Interface Input Timing Details

shift. Table 1 shows the external interface input timing details.

The following are constraints we applied for minimum timing values in UCF:

```
// Define Clock Net
NET "i_video_ch1_clk"
TNM_NET = "VIDEO_CH1_CLK"
TIMESPEC "VIDEO_CH1_CLK"
= PERIOD 6.8 ns HIGH 50%
INPUT_JITTER 0.1 ns
// Define Time Group for Rising and
Falling (In case of DDR Inputs)
TIMEGRP "VIDEO_CH1_CLK_R"
= "VIDEO_CH1_CLK" RISING;
TIMEGRP "VIDEO_CH1_CLK_F"
= "VIDEO_CH1_CLK" FALLING;
// Define Input Constraints.
OFFSET = IN 0.5 ns VALID 1ns BEFORE
"VIDEO_CH1_CLK" TIMEGRP
"VIDEO_CH1_CLK_R"
OFFSET = IN -1.5 ns VALID 1ns BEFORE
"VIDEO_CH1_CLK" TIMEGRP
"VIDEO_CH1_CLK_F"
```

For timing constraints on the PCI Express and Gigabit Ethernet MAC cores, we applied all timing and placement constraints for block RAM and PLL/DCM as defined in the CORE Generator example.

We defined the output timings with respect to input clock or PLL-generated clocks in a UCF.

```
// Define OFFSET OUT with respect
to clock.
NET "video_data_p0" OFFSET = OUT 3
ns AFTER "i_clk_video_in";
// Define MAXDELAY from Flip Flop to
pad to be minimum (Say 0.1 ns  to
0.2 ns),
NET "video_data_p0_to_pad" MAXDELAY
= 0.1 ns
```

The OFFSET OUT doesn't confirm that all outputs on all data and clock signals are exactly at 3 ns. The tool tries to meet timings with zero or positive slack (that is, less than 3 ns).

Since many Virtex-5 designs use multiple asynchronous clocks, we then had to define the false paths in the design so those clocks would not be affected. We did this with the following constraints settings in a UCF.

```
// Define False Path.
NET "video_data_p0"
TNM_NET = "VIDEO_CH1_TIMGRP";
NET "core_clk_0"
TNM_NET = "CORE_CLK";
TIMESPEC TS_FROM_VIDEO_CH0_TO_CORE =
FROM FFS ("VIDEO_CH1_TIMGRP") TO FFS
("CORE_CLK") TIG
```

Post-P&R Timing Analysis and Timing Fix

After placing and routing our design, we ran static timing analysis (STA) and timing simulation to see if we had any further timing errors. For STA, we ensured that the timing report covered all the constrained and unconstrained paths. By using an STA report, we can validate input/output timing and internal system timings. To fix the input timing violations (setup and hold), we use IDELAY with the appropriate tap value to meet timing requirements. To fix output timing violations, we made sure the respective signal flip-flop was in IOB. To fix the internal logic timing, we used an FPGA editor to make changes to the floor plan and the design's RTL code.

We then ran timing simulation to catch errors that we didn't detect during static timing analysis. The process involved generating a netlist compatible with the simulator we used during RTL simulation, and adding it to the Xilinx library path in timing simulation script.

Timing simulation will catch errors that STA doesn't. One critical example is an address collision in dual-port RAM that occurs when two logic blocks generate two asynchronous clock domains and addresses. Timing simulation also helps identify slow-changing signal or multicycle paths and multiclock domain paths in a design, thereby prompting designers to apply better timing constraints. That also helps fix timing issues in STA.

The Virtex-5-based FPGA proved well-suited for our video monitoring system requirements. The regional clock buffer and I/O clock buffers (with pre-bit deskew at IOB level using IODELAY) allowed us to support multichannel source-synchronous audio/video inputs. Moreover, the device's PCI Express and Gigabit Ethernet MAC hard macros gave us global connectivity for remote monitoring.

The end result was a cost-effective solution for our A/V remote-monitoring application. A bit of work at the early stage of design made it easy to meet timing closure. In our future design work, we will rely on early-stage planning to ensure the effective usage of the available resources of specific FPGAs. Defining global and regional clocks in detail and performing clock-requirement analysis and initial floor planning will make our flow more efficient, enabling us to rapidly design value-added products.

For more information, contact eInfochips at *sales@einfochips.com.*