



# Intel<sup>®</sup> Technology Journal

Second-Generation Intel<sup>®</sup> Centrino<sup>™</sup> Mobile Technology

**High-Performance Graphics and TV Output  
Comes to the Second-Generation  
Intel<sup>®</sup> Centrino<sup>™</sup> Mobile Technology Platform**

# High-Performance Graphics and TV Output Comes to the Second-Generation Intel<sup>®</sup> Centrino<sup>™</sup> Mobile Technology Platform

Kam Leung, Mobility Group, Intel Corporation  
Steve Spangler, Mobility Group, Intel Corporation  
Todd Witter, Mobility Group, Intel Corporation  
Kevin E. Arendt, Mobility Group, Intel Corporation  
Satya Avadhanam, Mobility Group, Intel Corporation  
Satyaki Koneru, Mobility Group, Intel Corporation  
Val Cook, Mobility Group, Intel Corporation

Index words: 3D graphics, 3D graphics acceleration, memory controller hub, TV output, image filtering, color space conversion

## ABSTRACT

This paper considers two major enhancements to Intel's graphics memory controller hubs, as embodied in the Intel<sup>®</sup> 915 Express Chipset Family. First, we discuss the microarchitecture of the 3D pipeline and the steps taken to optimize it for peak performance. Secondly, we present the TV output feature, which is important for merging the personal computer and the television into a single platform, in support of Intel's digital home initiative.

In the 3D graphics section, we present an overview of the microarchitecture in the context of the design challenges inherent in a next-generation integrated graphics accelerator. This section demonstrates the key roll benchmark analysis played in optimizing the performance of various components of the graphics pipeline such as command processing, primitive processing, pixel shader floating point units (FPUs), caching algorithms, etc. Obtaining a balanced pipeline is central to achieving maximum performance.

With the merging of the personal computer and the television into a single platform for the digital home, new requirements are emerging for the standard home PC. High among these features is the inclusion of the television output interface or the digital television encoder. Television signals are quite different from the

standard computer monitor connections and therefore have very different requirements. The TV output function also needs to be highly flexible because television encoding algorithms are different depending on the country of origin.

## 3D MICROARCHITECTURE INTRODUCTION

Significant feature and performance enhancements have been incorporated into Intel's third-generation graphics processing unit. The performance of these platforms generally exceeds expectations, with the Intel 915 Express Chipset Family demonstrating approximately 2.4 times the performance of its predecessor, as measured using Future Mark's 3DMark\*2001 [1] benchmark, and approximately 10 times the performance using the 2003 version of their benchmark [2].

This paper presents the key architectural challenges associated with attaining these performance levels; we intend that our methodologies and conclusions will provide valuable insight for other design projects. A cursory understanding of the state of the personal computer graphics industry is necessary to appreciate the task presented to the graphics team.

---

<sup>®</sup> Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

---

\* Other brands and names are the property of their respective owners.

### 3D Graphics Background

The graphics and media industries can be segmented into many categories, but a few of the more defining characteristics are directly related to the device's memory subsystem and its cost, considering both power consumption and die area. Intel's integrated approach places the graphics accelerator and TV output logic on the same die as the Memory Controller Hub (MCH), where a single memory subsystem is shared with the CPU. Memory access arbitration policies are consciously tuned to favor the CPU, resulting in increased access latency for the graphics operations. Additionally, manufacturing costs and power consumption constraints severely limit the die size. However, discrete add-in cards generally have local memory subsystems and larger die size budgets, due to the premium price obtained for these devices. Working within these constraints is the overriding consideration in the development of the architecture.

Additionally, the graphics industry is at a crucial inflection point in the development of consumer-level graphics acceleration. All of the lighting and rasterization calculations were previously performed by fixed-function logic, with relatively few parameters, but are now being replaced with programmable execution units. These units are commonly called "shaders" implying the process of shading or lighting the rendered primitives. A Pixel Shader unit is programmed using an assembly language similar to a CPU. The transition from fixed-function logic to programmable shaders makes it much more difficult to select a single performance *sweet spot* and optimize the architecture for that point. The concept of the sweet spot has vanished in favor of many workload-dependant optimization points.

### 3D Microarchitecture Overview

A brief discussion of the graphics architecture aids in understanding the specific design challenges that are considered in detail in subsequent sections. From a system-level point of view a graphics application performs several tasks prior to engaging the graphics accelerator to synthesize an image. Initially, various surfaces are populated in memory, such as texture maps, constant buffers, and vertex buffers. Many state control structures are also created in memory, including compiled versions of the pixel shader programs. It is not uncommon for all of these entities to require 100's of megabytes of storage.

The application then begins the rendering process by making specific DirectX\* [3] or OpenGL\* [4] API calls to draw primitives on the render target surface. These

commands are placed in a ring buffer, and the graphics accelerator is instructed to begin processing the buffer. As shown in the simplified block diagram of Figure 1, the commands are retrieved from the memory subsystem and when applicable are presented to the graphics pipeline for processing.

#### Primitive Processing and Iteration

Primitive processing consists of computing various constant attributes of a primitive, such as its axis-aligned bounding box and coefficients required by the Iterators. The Iterators traverse the interior of a primitive, such as a triangle, and produce interpolated values for each pixel location contained within the bounds of the primitive. It is important that the throughput of the primitive processing stage be such that the Iterators, which contain a significant amount of custom floating-point logic, are not starved for input. Correspondingly, the computation capability of the Iterators must be sufficient so as not to starve the even more expensive Texture Sampler and Pixel Shaders.

#### Dispatch

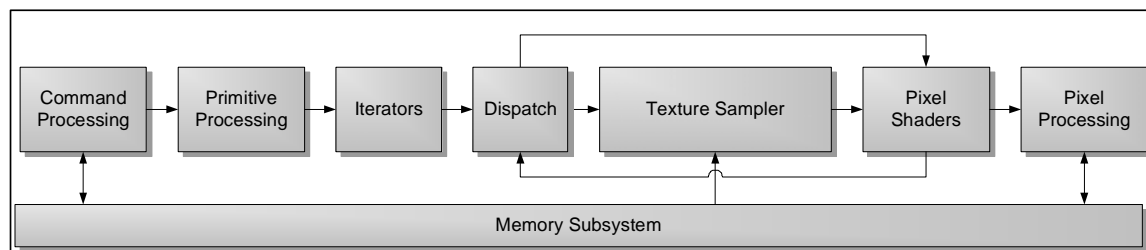
These interpolated values may then be used directly by the Pixel Shaders or presented to the Texture Sampler, as determined by the pixel shader program and managed by the Dispatch unit. The Dispatch unit also arbitrates sample requests from the Pixel Shader, where a programmatically generated sample location has been determined, rather than simply interpolated. This arbitration and scheduling task is particularly challenging.

#### Texture Sampler

When a sample request is delivered to the Texture Sampler one of texture maps that was initially loaded by the application is accessed and the resulting data are optionally filtered and presented to the Pixel Shaders. The Texture Sampler employs a multilevel caching architecture where the second-level cache is significantly larger and less capable than the L1 cache. The necessary format conversions, data swizzling, and output bandwidth of the L1 cache make it substantially more expensive than the L2 cache. Maximizing the utilization of the L1 cache and its supporting structures is essential to balancing the performance against the area and power consumption of the device.

---

\* Other brands and names are the property of their respective owners.



**Figure 1: 3D graphics pipeline**

### Pixel Shaders

The Pixel Shaders consist of many floating-point units (FPUs) that are controlled in a lock step fashion. This Single Instruction Multiple Data (SIMD) approach has the advantage of being able to amortize the cost of the control logic across multiple FPUs, but it also reduces the overall efficiency of the more expensive resources in the architecture.

### Pixel Processing

Finally, the Pixel Processing unit performs many fixed-function operations such as format conversion, color space conversion, and alpha blending. The resulting pixels are also assembled and delivered to the memory subsystem in this unit. The throughput of these operations is significant because the overall pixel fill-rate, one of the key performance metrics, is ultimately determined by this unit. However, when complex shading or large amounts of texture sampling is occurring this logic can be significantly underutilized.

### Workload Visibility and Analysis Tools

In the presence of an industry-wide inflection point in the graphics programming model, requiring a substantially more dynamic hardware architecture, the need to have visibility into key workloads is essential. This necessarily results in the enhancement and development of several new tools and analysis capabilities. We focus on the following three primary areas.

#### Enhanced Driver Capture Capability

The ability to run an actual application or benchmark against the graphics simulator is achieved by capturing the stream of commands presented to the driver. Additionally, memory resident structures, such as texture maps, vertex buffers, and shader programs are also captured. These data are written to disk in a standardized binary file format that is then used as stimulus for our functional simulator. The simulator can also be installed as a virtual device, thus eliminating the file I/O from the simulation.

### Timing Approximation from Functional Simulation

Our primary simulator provides very precise functional modeling, but contains no concurrency. This presented us with a significant challenge: how could the expected performance of a captured workload be predicted without a timing-aware simulator? Traditional instrumentation approaches are inadequate for our needs; however, by approximating the design as a fully pipelined, directed graph of logical units we could instrument the functional simulator to provide valuable timing information. Each unit is assumed to have an infinitely deep FIFO between it and the adjacent units in the graph. If a completely homogenous stream of work were flowing through the graph, and the minimum time that each quantum of work could reside in each unit were known, the throughput of the system would be limited by the slowest unit.

Unfortunately, real graphics workloads are not homogenous and each unit does not operate on the same input data. However, by selecting a quantum of work that naturally flows through the data path of the design, one can accumulate the time required to operate on each unit of work across multiple inputs. We chose to use a single primitive as our quantum of work. The functional simulator was then instrumented to determine the minimum amount of time that each primitive could reside in each unit. The maximum time required by any unit was then determined to be the pipeline throughput for that quantum of work. The sum of all of the quanta of work represents the theoretical peak processing time for the workload. In practice the time required to service each primitive varies radically and the FIFOs between the units are of finite size. However, we found this approach to correlate with compute bound workloads reasonably well. More traditional analytical techniques are used when the memory bandwidth is the dominant factor in the performance.

#### Database Driven Static Analysis

A single benchmark requires days to execute on the functional simulator. When changes to the statistics-gathering instrumentation or “what if” scenarios are desired many days of reacquiring the information from the

simulator are required. In order to minimize this impact we further instrumented the simulator to export control state, primitives, and timing information with primary keys. These data are then imported into a relational database, containing millions of rows in some tables. Relatively simple queries produced information in seconds, that would have otherwise required several days of simulation time. Additionally, stored procedures were used to perform complex computations that modeled proposed changes to the design, such as various state caching mechanisms and performance improvements in specific units. Most of these more complex queries completed in less than an hour.

These tools and capabilities assisted in providing valuable information for many of the challenges faced by the architecture team.

## GRAPHICS ARCHITECTURAL CHALLENGES

The major architectural challenges of the Intel 915 Express Chipset Family, as briefly addressed in the overview section, are divided into six categories corresponding to the basic units of the block diagram in Figure 1. Each of these challenges are addressed in the following sections.

### Choosing the Optimal Primitive Processing Rate

The time required to process a primitive, independent of its area, consists of a fixed overhead time and a per attribute computation time. The computations associated with the fixed overhead can be pipelined, but because there are varying numbers of attributes on a vertex, such as its position, diffuse color and texture coordinates, etc. the computational resources are reused. This is traditionally accomplished by re-circulating each attribute through the same logic, thus preventing efficient pipelining, but significantly reducing the cost. For example, the previous generation could process a primitive with a position, color, and 2D texture coordinate attribute at each vertex in approximately 50 clocks. This relatively large latency can be hidden if the primitive lights a sufficiently large number of pixels. However, today's graphics benchmarks and other important workloads consist of many very small triangles that do not consume much time processing pixels. These primitives are said to be "setup limited."

Our workload analysis suggests that even at our chosen processing rate, an average of 68% of the primitives are potentially setup limited. However, these same primitives represent only 7% of the total time required to render the workload, due to their small area. The challenge then

becomes deciding how much computational logic to enlist in order to improve a relatively small portion of the overall rendering time.

As we considered primitive processing designs similar to our previous generations the workload analysis indicated that approximately 17% of the rendering time would be spent on setup limited primitives. Realizing the relatively large performance penalty associated with these primitives and the trend toward smaller and smaller primitives we chose to improve the throughput of the per attribute computations to one clock/attribute, thus attaining a theoretical 10% improvement over previous generations.

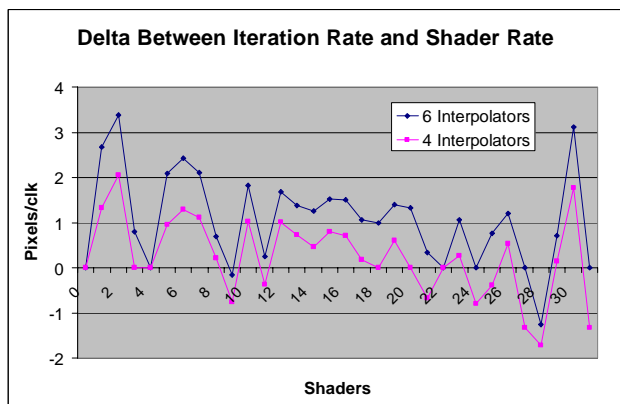
### Selecting the Iteration Rate

The Iterators determine the pixels that are lit within a primitive. It implements a walking algorithm that minimizes the pixels that are considered for testing. This testing is carried out at the rate of 16 pixels/3 clocks to sustain the required throughput of 4 pixels/clock. Once the lit pixels are determined, an early depth test, whenever possible, is performed to eliminate the covered pixels. This significantly reduces the workload on the remaining part of the pipeline. The rate for these operations provides sufficient performance that these operations are rarely the bottleneck in the pipeline. However, determining the rate at which the vertex attributes should be interpolated is a significant design decision.

Because the interpolation operations and the shading operations occur concurrently, it is desirable to have the throughput of both units matched in order to achieve the maximum performance given the available die area. However, there is not an inherent relationship between these two operations. It is very possible to require few interpolated attributes while utilizing many shader instructions to produce the output. This case is typical of procedural shaders, such as wood, marble, and water, where the output color is determined algorithmically. However, it is also possible to require many interpolated values that utilize very few instructions. These shaders tend to drive much of the computation with tables or texture maps. Filtering techniques that require large filter kernels also tend to fall into this category.

Various shaders from prominent workloads were captured to obtain the number of attribute components to be interpolated vs. the number of arithmetic instructions. We considered providing four or six interpolators per pixel, which all operate in parallel. These interpolators require floating-point precision and are relatively expensive in terms of die area and power. However, they are not as expensive as the FPUs used in the Pixel Shaders. Each line in Figure 2 identifies the difference in the interpolation and Pixel Shader rate, normalized to

pixels/clock. When the value for a given shader is positive the interpolation rate exceeds the shader rate.



**Figure 2: Difference between the iteration rate and the shader rate for two interpolator design options**

As can be seen from the graph in Figure 2, four interpolators perform reasonably well; however, the penalty for underutilizing the Pixel Shaders is also substantial. Additionally, the 4-pixels/clock throughput was desired for the very typical scenario of a 4D diffuse color and a 2D texture coordinate. Thus, the number of interpolators was determined to be six. This also meant that the capability to process two coordinate sets at a time was needed. Any combination of a (1-4)D coordinate set and a (1-2)D coordinate set is allowed to be processed in parallel.

**Determining the Dispatch Output Bandwidth**

The interpolated coordinate sets may be used as is and/or passed through the Sampler before being delivered to the Pixel Shaders. Having previously decided to generate two coordinate sets every clock during iteration, the dispatcher was designed to deliver one coordinate set to the Sampler and one coordinate set directly to the Pixel Shader (through a bypass FIFO) simultaneously. In the case of dependent read scenarios, there are additional source coordinate sets produced by the Pixel Shaders. This allows the pipeline to operate in a balanced fashion.

**Maximizing the Texture Sampler Utilization**

Texture sampling is central to the overall performance of the graphics accelerator, and the caching microarchitecture is the heart of the Sampler. These structures are discussed in detail in the following sections.

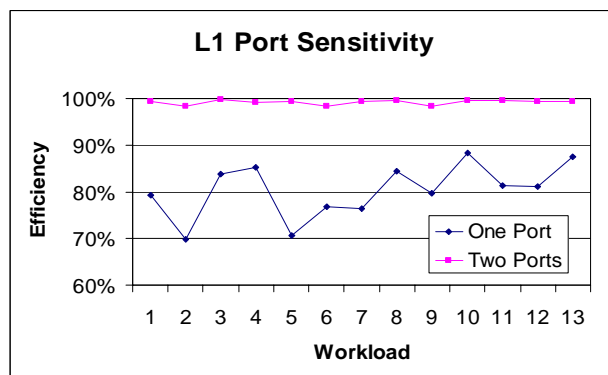
**Determining the L1 Texture Cache Read Port Count**

The Intel 915 Express Chipset Family expanded the texture sampling engine to four pixels wide, from two pixels wide in the previous-generation sampler. The four pixels are always adjacent, forming a 2x2 orientation called a pixel quad. The width in pixels of the sampling

engine determines, among other things, the amount of data that needs to be read from the L1 texture cache in parallel. Adding ports to the cache is expensive, so we needed to identify a way to get most of the performance benefit of a 4-pixel sampling engine without the cost.

Through the simulation of workloads, we determined that in most cases there is significant overlap of the texture data (texels) that the adjacent pixels fetch. Thus it is not necessary to have a dedicated independent port for each pixel on the L1 cache to efficiently fetch the required texel data.

A single-ported L1 cache was also considered, but the performance tradeoff was deemed too high. The following graph shows the relative efficiency of the 2-port and 1-port caches. The 4-port cache has 100% efficiency by definition.



**Figure 3: L1 texture cache read port sensitivity**

Specific control logic checks each incoming pixel quad to determine whether it requires more than two cache ports. This process involves comparing the addresses of the data requested for each pixel. If the data can be retrieved through two cache ports, the pixel quad is allowed to proceed unmodified. If more than two ports are required, the quad is split into two 2-pixel packets.

**Pixel to Port Mapping**

The next consideration was, in cases where the pixel quad does not need to be split due to the above considerations, to determine which pixels are mapped to each of the two cache ports. Increased flexibility generally means increased cost, and this case is no different. Information on which port each pixel is using needs to be sent downstream to enable the correct data to be routed out of the cache. Full flexibility means that many bits indicate the port/pixel mapping.

Further performance simulations, again using workloads, resulted in a decision to allow only two port/pixel mappings, termed “horizontal” and “vertical.” Horizontal indicates that horizontally adjacent pixels share a port; vertical means vertically adjacent pixels share a port.

Cases in which neither of the allowed mappings work are split into two packets of two pixels each. This is a performance/cost tradeoff. The addition of more flexibility than this does not significantly increase performance and does not justify the additional cost. The relative efficiency of this approach exceeds 90%, averaged across the workloads.

The graph in Figure 4 shows the fully flexible case (equivalent to the “two ports” case in the prior graph), a fixed mapping case where the port/pixel mapping is fixed, and the compromise option that we ended up choosing.

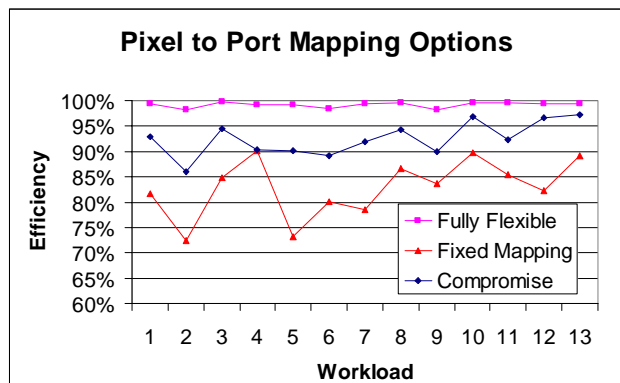


Figure 4: Various pixel-to-port mapping options

### Line Size

The size of the L1 texture cache needed to be doubled; however, there were several approaches that we considered in order to accomplish this. The number of ways could be doubled, the number of sets could be doubled, or the line size could be doubled. We used workloads under simulation to compare doubling the line size to doubling the number of ways. The number of sets is tied to the ports, so we did not consider that vector. The analysis indicated that doubling the number of ways provided a better cost/performance tradeoff.

The graph in Figure 5 compares the miss rate on a 32-byte vs. a 64-byte cache line size. The overall cache size is the same for both cases. The 32-byte case simply doubles the number of cache lines. Although the miss rate is generally higher for the 32-byte cache size, the 64-byte line size has double the bandwidth requirement per miss, which became the driving factor in our decision. We selected the smaller cache line size with twice the number of lines.

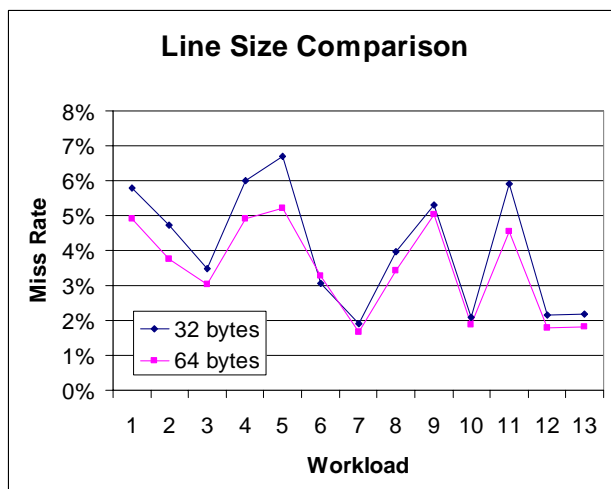


Figure 5: L1 cache line size comparison

### Determining the Pixel Shader Throughput

Based on the ratios in the pixel-shader programs between the number of interpolated coordinates and the number of arithmetic instructions, it was determined to have eight FPU's in the Pixel Shader. The 2x2 pixel quad is the basic processing entity, these eight FPU's are organized as two sets of four FPU's. Each set of 4-FPU's operate in parallel, potentially executing different instructions. The four grouped FPU's operate in a SIMD fashion. The pixel quads are delivered to the two sets of FPU's in a ping-pong/alternating fashion and are output in the same manner to maintain a strict in-order behavior through the 3D pipeline.

The execution of the program is interleaved. Execution of an instruction on a pixel quad occurs only after the completion of execution of the previous instruction, thus eliminating data-dependency checking requirements. An empty slot in the FPU's pipeline is filled with a new pixel quad, if available. This multithreaded execution mechanism boosts the utilization significantly.

The sizing of the RAMs for storing the input, output, feedback, and temporary registers is determined by the register utilization in a pixel-shader program and the latency of the execution pipeline, as shown in Figure 6. To allow for streaming, additional storage is required for two cases: first, when a pixel quad's input payload is delivered and it is waiting to start execution and second, when a pixel quad has completed execution and is delivering its output payload. Taking this into consideration and the expectation that the register usage would grow over time we selected a storage size of 64 entries.

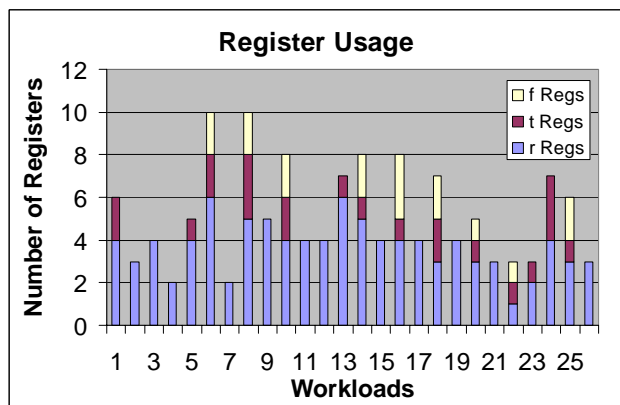


Figure 6: Register utilization

### Determining the Pixel Processing Throughput

Pixel processing includes fog blending, alpha/stencil/depth testing, alpha blending, gamma correction, dithering, and logic operations. Our analysis suggests that on average 23% of the pixels rendered require fewer than six attributes to be interpolated, consisting of one texture sample, and two shader instructions. These data suggested that there is significant performance to be captured if the back-end pixel-processing functions also operate at 4-pixels/clock. To achieve this desired fill-rate four pixel-processing blocks operate in parallel on a pixel quad.

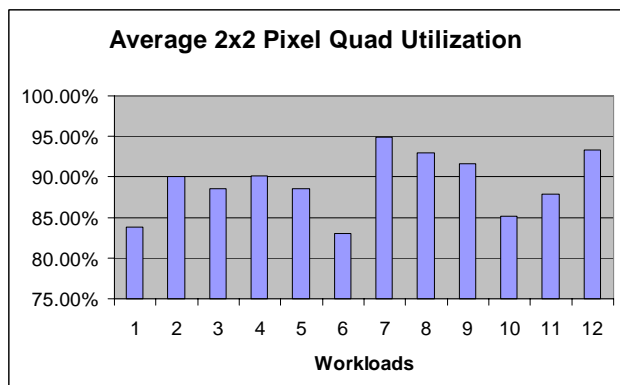


Figure 7: Average 2x2 pixel quad utilization

Grouping pixels in this fashion is somewhat inefficient, due to the fact that not all of the pixels may be lit, but will still consume the hardware resources. Our analysis indicates that this inefficiency is relatively small, as can be seen in Figure 7.

### 3D GRAPHICS RESULTS

The actual benchmark results indicate significant performance improvements over previous generations. Additionally, we outperform the two leading competitor's integrated devices.

Table 1: Benchmark results

Benchmark	865G	915G	915GM (200 MHz)	915GM (320 MHz)	Integrated Device A	Integrated Device B
3DMark01	159	1522	1054	1266	790	0
3DMark03	3100	5872	5101	5337	5224	4947

We credit much of this success to a design methodology that incorporates workload analysis as a key component of the decision-making process. Our experience suggests that investment in tools and infrastructure to provide workload-driven information during the architecture definition phase produces valuable results. The Intel 915 Express Chipset Family stands as evidence, amidst widely dispersed workload characteristics and an industry-wide programming paradigm inflection point, that advanced analysis tools and data-driven design methodologies are vitally important.

### TVOUT

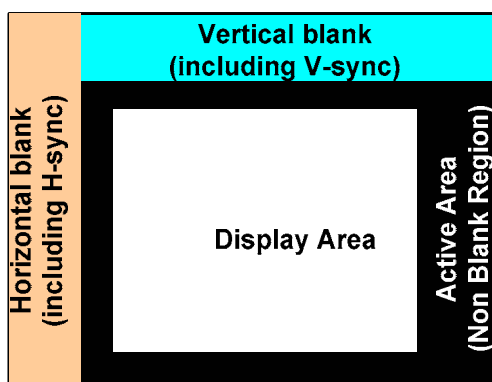
In recent years, more people are using their personal computers, or PCs, to play and store music and digital photos, and their PC's DVD drives to watch movies. Moreover, people are using their large-screen TV as a viewing device for movies, digital photos, and online gaming. Finally, the integration of digitally recording favorite TV programs onto a hard drive, combined with the television connection, has earned the PC a position in home entertainment systems. Advancements in networking have even made it possible to share the recorded programs with a remotely located PC. Such a wide variety of applications triggered the convergence of PC and consumer electronics. Initially this functionality was achieved by using an external television encoder device connected to a graphics accelerator, which is complex and expensive. The Intel 915 Express Chipset Family integrates the two functions to reduce the complexity and price.

The goal of this section of our paper is to define and explain the video encoder function and the required associated logic for the encoder to function in a PC environment. The design can be divided into three major sections, each having its own design requirements. When displaying computer graphics data on a television the data need to be converted to meet the more stringent requirements of the TV modulation.

Television video output is similar to computer video output in that it is composed of an analog signal with two parts. The first part is timing information that tells the display where to put the data on the screen, and the second

part is the image data encoded in one of the three output modes. Output modes are explained in detail later.

For the computer, the analog connection to the display device uses three connections utilizing red, green, and blue colors, commonly known as the RGB interface. In the RGB interface, color information is carried on three signals indicating the color intensities of red, green, and blue. This is convenient since that is (nearly) the same color space as the monitor and it is also (nearly) the native format of the data as they are stored in the computer's memory. The timing information is usually carried on separate signals for horizontal and vertical synchronization pulses; however, some monitors require the synchronization of information to be placed on the green channel. The vertical synchronization pulse indicates when to start a new frame of data; the horizontal synchronization pulse indicates when to start a new line. Vertical rates are usually in the 60-85 Hz range, but can go much higher. Horizontal rates depend on the total number of pixels being displayed and range from 30 kHz to over 100 kHz.



**Figure 8: Television display timing areas**

For connection to the television, originally all of the information (color and synchronization) was transmitted in just one signal (actually the signal was modulated onto the channel carrier and transmitted). This signal is called Composite Video Block Sync (CVBS) or usually just Composite Video. The data transmitted in this form use a different color space. The YUV color space represents picture information in Luminance (Y) and Chrominance (U and V). Luminance is simply the brightness portion of the image and was the only part transmitted until the advent of color television. The UV pair indicates the color with hue and saturation in vector form. The luminance information is encoded as a simple-level signal, and the chrominance information is encoded in a modulated carrier on top of the luminance. Different parts of the world use different modulation techniques.

More recently the composite signal has been separated out into two signals: this type of connection is called S-Video.

In an S-Video connection, the luminance and chrominance are transmitted each on their own pair, eliminating crosstalk between the two signals. The luminance channel additionally carries the synchronization pulses.

In the last few years, televisions have appeared with a new analog connection that uses three pairs of wires called Component Video. This connection is much like S-Video except that the U and V signals are separated out onto their own pairs of wires, further improving the signal quality. Additionally the UV signals have their amplitudes scaled to match the luminance signal; this level matched interface goes by the convention YPrPb. Because of its higher quality, this type of signal is used for analog High Definition TV (HDTV) connections.

The actual timing signal and placement of the video data are different for the TV and PC video. The computer's video is drawn in line order or progressive scan. Each line of data on the screen is followed in time directly by the line beneath it. In most TV modes, the method is to draw all of the odd numbered lines in sequence followed by all of the even numbered lines. This method is called interlaced scan. The two groups of lines are called the fields: Field 1 has the odd lines while Field 2 has the even lines (counting from one, not zero). The two fields together compose a frame. In order to differentiate the two fields, the vertical synchronization pulse is skewed by a half line between Field 1 and 2 (but not between 2 and 1).

HDTV supports both interlaced and progressive video formats. The different modes of operation are typically indicated by adding a p or i to the number of displayed lines in the format, e.g., 480i, 720p, 1080i, etc.

## PC Video is Not TV Video

There are several important differences to account for when converting PC video for use on a TV. The primary concern is the visual artifact known as "flicker" caused by the conversion from progressive scan to interlaced scan. Flicker is caused by the presence of single-pixel horizontal lines in the source material. When these lines are displayed on an interlaced monitor it appears to flicker on and off at the frame rate, because the line is only present in one of the two fields. These types of horizontal lines do not normally show up in normal video transmission because the source is generally a camera. However, since the computer video is a synthetic image, it is possible (in fact common) for a one-pixel tall horizontal black line to appear on a white background. Further, if the displayed area is in small font text (which usually has lots of little horizontal lines), the text can be difficult or impossible to read.

Another major difference is that the PC video is "underscanned" and the television video is "overscanned."

This means that the four corners of the PC's video image are visible to the viewer while the television allows the corners to fall behind the bezel of the monitor. In addition, many computer modes use pixel resolutions that are simply too big to fit on the television monitor in the standard definition mode.

One further unfortunate problem with television signals is that they are different depending on their country of origin. This is due in part to the European and Asian countries using a 50 Hz for power distribution while North America uses 60 Hz. The fallout of this is that the two systems are incompatible. Even worse is that many countries have developed their own versions of the two specifications resulting in a plethora of required formats. Fortunately, nearly all of the standards are similar enough to make it possible to generate them with one programmable logic block.

In summary, in order to correctly convert the PC's video signal for use in a television it is at the minimum necessary to perform three functions. The PC video color space must be changed from RGB to YUV; additionally it must be scaled to fit the television's display area accounting for underscan. Then that signal must be encoded into one or more of the three signal transmission standards mentioned above: Composite, S-Video, or Component. This last issue has two pieces: a digital portion and digital-to-analog converters (DACs). DACs are described in the next section.

### Color Space Conversion

The television encoder works in the YUV color space, but computers operate in RGB color space so it is necessary to convert the RGB into YUV. Color space conversion is accomplished by putting the RGB data into a 3x3 matrix multiply. Unfortunately, not all television standards use the same YUV color space (or encoded space), so the converter needs to be completely programmable.

For NTSC Composite and S-Video

$$Y = 0.299R' + 0.587G' + 0.114B'$$

$$U = -0.147R' - 0.289G' + 0.436B'$$

$$V = 0.615R' - 0.515G' - 0.100B'$$

After color space conversion, there are two luminance controls that are typically made available to the user. These are *brightness* and *contrast*. Brightness is simply a constant value added to the luminance value. The contrast value is used to adjust the gain of the luminance. Similarly, there are two controls to chrominance. They are *saturation* and *hue*. Saturation adjustment manipulates the amplitude of the modulated chrominance, which is seen as color intensity. Hue adjustment manipulates the relative phase angle of the color component, which is seen as a color shift.



Figure 9: Data flow through the TVout pipeline

### Scaling and Filtering

The conversion of progressive image data to interlaced data is essentially a scaling operation. The image is reduced to one half the original number of vertical lines. The simplest method to accomplish this would be to throw away every other line. However, this only works if the original image and the TV output image are the same size, which is not normally the case. So in this implementation a programmable scaler is placed in the pipeline to scale the incoming image to the TVout display size. This scaler can then also be used for the underscan correction (if that is desired) when displaying the PC content on a TV. There is a small one-half line offset in the scaling operation in the second field to account for the vertical offset between the fields and to preserve as much of the original data as possible.

At the same time the image is scaled for interlaced display, it should also be filtered to remove flicker. Flicker reduction is accomplished by softening the hard edges present in PC displays. Flicker reduction is a subjective feature, so several options are provided for users to select the amount of flicker or softness they desire.

The video filter is implemented as seven taps horizontally by five taps vertically, with each filter having its own set of coefficients. Five vertical taps are required in order to properly interlace the vertical data without causing flicker. The larger seven tap horizontal filter is required to properly filter the chrominance data before they are modulated. This prevents cross-color distortion.

### Encoding

The final stage of the process before conversion to the analog signal is encoding. There are four major functions within the encoder. The timing generator is responsible for generating the internal timing reference signals as well as the external composite synchronization, which will eventually be added to the luminance channel data. It also generates timing signals that place the video data on the screen. The subcarrier generator is responsible for the proper generation of the reference subcarrier signal for the modulator and for ensuring a proper phase relationship to the horizontal synchronization. The luminance processor combines the composite synchronization and the Y data and generates the luminance signal that will be sent to the DAC. The chrominance processor block takes the subcarrier signal in along with the UV data and generates the chrominance signal; it also generates the subcarrier

burst signal, which is used by the television receiver to correctly demodulate the signal.

Creating a good subcarrier is of paramount importance to the generation of good quality video. The Intel 915 Express Chipset Family uses an 8x over-sampled clock when generating video for modulated formats. For NTSC, the color subcarrier is about 3.58 MHz; with a 1x clock of 13.5 MHz there would only be three or four samples per cycle. This is just barely enough to reconstruct the signal. At 8x over sampling (108 MHz) there are 30 or 31 samples per subcarrier cycle. This provides a very accurate representation of the carrier, which results in better color recovery at the receiver.

In order to generate quality output, the data output from the encoder is in a 10-bit per channel format requiring 10-bit DACs. The peak voltage of the CVBS DAC needed to be 1.3 V; the Y channel of the S-Video and Component Video should be 1.1 V max. The CRT DAC implementation for PC video is only 8 bits with a peak voltage of 0.7 volts. There is no way to generate a quality television signal with only an 8-bit resolution on the DAC. This is partly because of the accuracy required to properly construct the subcarrier, but mostly because a large number of the values are used in creating the synchronization pulse. For Composite Video, of the 1.3 volt swing available, only about 0.7 volts are available for luminance. This means that of the 256 levels in the DAC only 138 can be used for luminance; this severely quantizes the picture data causing banding and contouring errors in the image. The 10-bits per channel used in our DAC provides the accuracy needed for good quality video.

## CHALLENGES AND SOLUTIONS

Firstly, design to support multiple TV standards:

- Each TV standard has its own specifications with respect to video bandwidth, sub-carrier frequency, color space conversion coefficients, etc. To meet all the requirements this function is designed using many programmable registers.

Secondly, the quality of the filter is crucial:

- The sharpness of the image on TV depends on the transfer function of the horizontal and vertical filters, and since this control is subjective, the filter coefficients are made programmable so that the sharpness can be adjusted by the user.
- Pixels are processed in YUV422 format in the filters. In this format luminance is sampled on every pixel, and chrominance is sampled once every two pixels. Due to this, the filter output incurs a delay between luminance and chrominance called group delay. This

delay varies based on the scale ratio and the over-sampling rate. Unwanted visual artifacts are seen on the screen if the delay is not offset. The internal logic compensates for this delay by a programmed number of samples.

Validation of the design is a major challenge.

- TVout was validated at three different levels: RTL using simulations, FPGA, and emulation. Each one of the levels has its own advantages and disadvantages. A combination of the three levels gave good coverage. For better testability of the digital logic and the DACs, 13 different test patterns are implemented in the design.

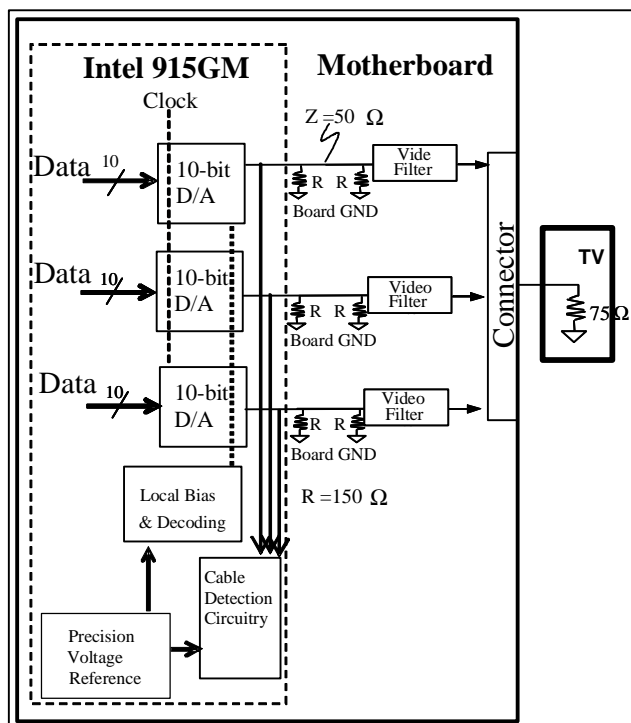
### TVout Digital-to-Analog Converter

A key component of the integrated TV encoder is the DAC embedded block. The TVout embedded block consists of three, 10-bit resolution channels that are fully integrated and custom designed for consumer quality television. Each channel can be programmed to one of three different video voltage swings to support Composite Video, S-Video, and Component Video format.

Figure 10 shows the block diagram of a DAC embedded block that interfaces to the TV.

The DAC embedded block consists of three identical, 10-bit channels, a first-order temperature-compensated voltage reference circuit, local biasing and decoding for output video voltage-level programming, clock distribution, and circuitry for video cable detection.

The DAC is designed to directly drive the TV without the need for external video buffers. There is a passive low-pass video filter at each channel output designed with a nominal bandwidth of 30 MHz to limit the video bandwidth for HDTV. The TV connection to the motherboard is made through a video connector that is compatible with standard S-Video cables and RCA-type cable connections with the use of a dongle. This interface was designed to minimize the number of video connectors and components on the motherboard as well as to minimize the complexity of routing several video signals to various RCA and S-Video-type connectors on the motherboard.



**Figure 10: Block diagram of the DAC embedded block and associated board-level components**

### DAC Circuit Architecture

The performance of the DAC is critical for achieving excellent video quality. The required accuracy of the DAC is based on the differential gain and phase distortion specifications for TV. The architecture is designed as a current-steering architecture to achieve high accuracy and low distortion of the analog video signal from the motherboard ground level (zero volts) to the maximum nominal analog video signal swing of 1.3 V.

The digital input to each DAC is latched on the rising edge of each clock and converted to an analog current. For a given digital input, the current source outputs are either summed together and directed to the output pin or directed to the motherboard ground plane by the differential current switches. An analog video voltage is created from the DAC output current flowing into the termination resistors. The video level specifications for the various video formats along with the effective load termination determine the required output current of the DAC circuit. The least significant bit (LSB) output voltage is a function of the video format supported and normally ranges between 684  $\mu\text{V}$  to 1.27 mV above the motherboard ground.

One of the key performance parameters of the DAC is the static accuracy characterized by the differential (DNL) and integral (INL) non-linearity. Both the DNL and INL

must be less than  $\pm 0.5$  LSB for all video modes over the entire analog output range and over all operating conditions in order to achieve monotonic, intrinsic 10-bit static accuracy. Therefore, the current sources controlled by each bit of the digital input word must have an error of less than  $\pm 0.5$  LSB. This specification is very difficult to achieve for a DAC implemented with a binary weighted current source array in a standard CMOS technology while simultaneously meeting dynamic specifications such as signal-to-noise and distortion. Therefore, to reduce the matching constraint for the most significant bit (MSB) current sources while improving the dynamic performance of the converter, the DAC circuit architecture is typically segmented into two current source arrays, one binary weighted and one linear weighted. A DAC implemented with only a linear weighted array will guarantee monotonic behavior at the expense of silicon area. Therefore, there is a tradeoff between silicon area and accuracy of the DAC.

In order to achieve the DAC performance requirements with high yield and without calibration in high-volume manufacturing, the allowable mismatch of the unit current source or LSB was determined using statistical analysis based on the 0.13  $\mu\text{m}$  CMOS process technology. The random and systematic mismatch among the transistors along with the voltage drop of the power routing and the current source output resistance are the dominant terms that affect the accuracy of the DAC.

An additional constraint on the DAC design arises from the dynamic accuracy that is characterized by signal-to-noise and distortion (SINAD), and the minimum effective number of bits required for analog video bandwidths. For this design, the DAC is segmented where the six MSBs of the digital input code are thermometer decoded into 63, 16 LSB weighted current sources. Bits 2, 3, and 4 of the digital input code are also thermometer decoded into 7, 2 LSB weighted current sources. The 71 total current source outputs are summed together to create the DAC current output that corresponds to the digital input. At full-scale or digital input code = 3FFh, the channel output is  $1023 \cdot I$  where  $I$  is the unit current source magnitude based on a particular video-level format.

### DAC Circuit and Layout Design

The DAC current source arrays are implemented with PMOS transistors with cascode transistors to obtain high output impedance. Design techniques were applied to the DAC circuitry to minimize noise and distortion. Special layout techniques were applied throughout the circuitry to minimize the effects of random and systematic mismatch on circuit performance.

The layout of the DAC was custom designed and drawn to fit in the I/O ring utilizing the area of a standard I/O buffer

height while minimizing the overall width of the embedded block. The placement of the current sources for the DAC was designed as a two-dimensional array and the sequence of switching the current sources according to the digital input was designed to consider the spatial arrangement of the current sources to cancel, to the first-order, linear and non-linearity gradients that may occur. A custom package power plane was designed for a low-resistance, symmetric power grid for each channel.

Referring to Figure 10, the nominal DC termination in the TV is a resistive  $75\Omega$  termination, and the effective DC resistance at each DAC output is therefore  $37.5\Omega$ . Since  $75\Omega$  routing impedances are difficult to achieve on low-cost board fabrication, each channel is double terminated with a  $150\Omega$  resistors on the motherboard, with one termination resistor placed near the chip and the other placed near the video connector for the TV. This termination scheme results in a  $50\Omega$  nominal routing impedance between the termination resistors on the motherboard. As a result, the  $75\Omega$  routing impedance is eliminated from the chip output to the input of the video filter, and optimal signal integrity is achieved, which is essential for excellent video quality.

The TVout DAC embedded circuit block accuracy performance along with optimal signal integrity of the analog video signals is essential to obtaining excellent visual quality. The selection of the DAC circuit architecture along with the circuit design and layout techniques applied resulted in a 10-bit DAC without the need for calibration in high-volume manufacturing. The performance of each channel was characterized with static and dynamic test conditions over a wide range of supply voltages (3.0 V to 3.7 V) and a wide temperature range ( $-20^\circ\text{C}$  to  $+120^\circ\text{C}$ ) for the three supported analog video formats. The excellent performance of the TVout DAC embedded circuit block along with the topology of the termination on the motherboard results in an optimal analog video channel that achieves excellent visual quality.

## CONCLUSIONS

With the introduction of the Intel 915 Express Chipset Family we have brought high-performance graphics to the digital home. Our workload-driven design methodologies have delivered best-of-class performance to our customers.

The TVOut feature provides extremely accurate and programmable color space conversion, high-quality multitap flicker filters, and high-precision DACs in order to produce outstanding image quality. When these features are combined with the encoder's support of multiple

standards, the Intel 915 Express Chipset Family provides an excellent television experience.

Coupling these two features together brings the Second-Generation Intel® Centrino™ mobile technology Platform into the digital home as never before.

## REFERENCES

- [1] 3DMark2001 SE, Futuremark Corporation, <http://futuremark.com/>.\*
- [2] 3DMark\*03 Pro, Futuremark Corporation, <http://futuremark.com/>.\*
- [3] DirectX Specification, Microsoft Corporation, <http://www.microsoft.com/windows/directx/>.\*
- [4] OpenGL Specification, <http://opengl.org/>.\*

## AUTHORS' BIOGRAPHIES

**Kam Leung** was the Intel 915 Express Chipset Family 3D graphics design manager, and currently is managing Intel's next-generation of 3D graphics development. He has B.S.E.E. and M.S.E.E. degrees from Utah State University. He joined Intel in 1990 and has spent the past 14 years working on chipset and graphics development. His e-mail is kam.leung at intel.com.

**Steve Spangler** is a 3D Graphics microarchitect working on Intel's next-generation mobile and desktop integrated chipsets. He has B.S.E.E and M.S.E.E. degrees from the University of Wisconsin at Madison. He joined Intel in 1987, working on a variety of chipsets and microprocessors with the last six years being devoted to graphics chipsets. His e-mail is steve.j.spangler at intel.com.

**Todd Witter** is a senior staff engineer in Intel's Mobility Group. Todd is responsible for the microarchitecture of the graphics display system of Intel's integrated graphics controllers. Todd joined Intel in 1997, after working as a design engineer at the Interactive Television team at Philips Semiconductors. He received his B.S.E.E. degree from California State University at Chico in 1989. His e-mail is todd.m.witter at intel.com.

**Kevin E. Arendt** is a senior staff design engineer working in Intel's Mobility Group. He is responsible for designing custom embedded circuits for chipsets for desktop and

---

<sup>™</sup> Centrino is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

\* Other brands and names are the property of their respective owners.

mobile platforms. Kevin received his B.S.E.E. and M.S.E.E. degrees from Southern Illinois University at Carbondale in 1990 and 1992, respectively. Kevin joined Intel in 1995. Prior to joining Intel, he worked as a design engineer for AT&T's Global Information Solutions Division where he worked in the Analog Cell Library Group. Kevin is a member of the IEEE and his interest is in analog integrated circuit design. His e-mail is kevin.e.arendt at intel.com.

**Satya Avadhanam** is a senior design engineer working in Intel's Mobility Group. He received his B-Tech. degree from Sri Venkateswara University, India and his M.S.E.E. degree from Arizona State University at Tempe in 1995 and 1997, respectively. Satya joined Intel in 1998 and worked on various projects in 2D and 3D graphics areas. His e-mail is satyanarayana.avadhanam at intel.com.

**Satyaki Koneru** received his B-Tech. degree in Electrical Engineering in 1993 from the Indian Institute of Technology, Bombay and his M.S. degree in Computer Engineering in 1998 from Iowa State University. He joined Intel in 1998 and has been working in the field of 3D graphics since then. He is currently a microarchitect in the Mobile Graphics Engineering group in the Mobility Group. He was involved in the architecture of the graphics core built into the Intel 915 Express Chipset Family chipsets for the Second-Generation Intel Centrino mobile technology platform. His e-mail is satyaki.koneru at intel.com.

**Val Cook** is a senior staff engineer working in Intel's Mobility Group. He received his B.S.E.E and M.S.C.S. degrees from Brigham Young University at Provo in 1989 and 1995, respectively. He joined Intel in 1995 working on their 3D graphics products, helped architect the Intel 915 Express Chipset Family 3D graphics and continues to be involved with the development of future graphics architectures. His interests include 3D rendering algorithms and architectures. His e-mail is val.g.cook at intel.com.

Copyright © Intel Corporation 2005. This publication was downloaded from <http://www.intel.com/>.

Legal notices at <http://www.intel.com/sites/corporate/tradmarx.htm>.

**THIS PAGE INTENTIONALLY LEFT BLANK**

For further information visit:

[developer.intel.com/technology/itj/index.htm](http://developer.intel.com/technology/itj/index.htm)