HADMÉRNÖK

IV. Évfolyam 4. szám - 2009. december

Fürjes János
furjes.janos@chello.hu

GRAPHICS CARDS IN RADIO RECONNAISSANCE: THE GPGPU TECHNOLOGY

Absztrakt/Abstract

Jelen írás egy modern technológiát elemez, amely költséghatékony módon képes a keskenysávú vevők jelfeldolgozását elvégezni. Az állandóan fejlődő felhasználó piac, az igények és az új technikák összehangolása, mind-mind felhasználható, így azok szakmai területen csökkentik a fejlesztési és a működési költségeket.

This paper examines an up to date technology, which is a cost effective way of processing narrowband receiver channels can be carried out. Techniques aligning to the demands of the permanently developing user's market can also be used in the professional field reducing the development and the operation costs.

Kulcsszavak/Keywords: *GPGPU, rádiófelderítés ~ GPGPU radio reconnaissance*

INTRODUCTION

Who has ever thought over how pixels are calculated by a three dimension featured graphics program? Probably only a few of us, as it is out of our interest. However, when purchasing computer parts, we usually consider carefully what to spend on our scraped and saved money. But those who want to keep abreast of the times are compelled to innovate their machine quite often. The most dynamically developing component in the IBM PC market is the field of the graphics cards. When a new card generation is born, their performance doubles. In the beginning, an alternative usage was already considered to utilize the card's capacity not only for specialized graphics routines but also for general computing tasks. Regarding the fast developing rate we can only make proper decision if we follow the technological progress.

HISTORICAL OVERVIEW- THE EVOLUTION OF GRAPHICS

The story started with the Hercules cards. Cards from the first generation could only display fix characters in monochrome mode. In the next step the four and the sixteen color CGA/EGA

graphics cards appeared, which were followed by the (S)VGA 256 color cards with 1024*768 pixel resolution. Finally the features of the top cards reached the 1280*1024 pixel resolution with 16 millions of colors. At that time the computer game industry emerged to transfer some of the hardware elements facilitating the CPU's job to the graphics card.

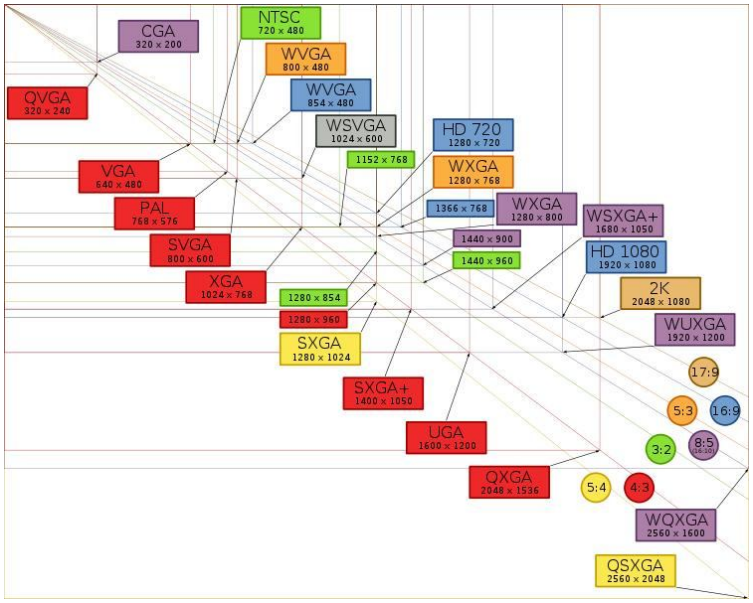

Figure1. The most commonly used resolutions [1]

Evolution of graphics chips

The 3dfx (in the beginning: 3Dfx), which already closed down, was a company that used to design 3D graphics processors. It is regarded as the originated producer of the 3D PC graphics processor. Although 3D technology had already existed, 3dfx made it wide known when introducing Voodoo 1 card. Its main trade partner was Quantum3D having share in military industry. At this moment a hardware device developed for this purpose of handling parallel graphics tasks appeared. In 2000 the company went down, and its technology was transferred by nVidia. In 2003 an engineering team separated from Cray Computing joined nVidia. That time onward events speeded up. They invented a cell structured processor which was able to cooperate with additional executing elements and to reach high computing accuracy.
The other relevant company called ATI (Array Technologies Incorporated) from Canada was established by three Chinese immigrants in 1985. At first ATI designed GPUs for the major manufacturers (IBM) being interested in computer science, but by 1987 it became an independent one producing EGA/VGA Wonder cards. In 2000 ATI launched its popular Radeon series. Besides, it went on cherishing its relations with other companies (Microsoft uses ATI chips in Xbox 360). In 2006 AMD purchased ATI for 5.4 billion dollars.
Recently ATI is the main rival of nVidia. In 1991 ATI created the Mach8 chip which was able to display graphics information without CPU. In 1992 it was followed by Mach32 and in 1994 by Mach64 as the harbinger of 3D acceleration (in 1999 ATI Rage 128). In 2000 the next step was the Radeon series with DirectX support whose developing process has been in progress from that time. Nowadays ATI's top quality graphics card is HD5970 containing 2GB DDR5 memory module and 2*RV870 chips consisting of 2.15 billion transistors with 40 nm technology. This configuration includes 2*1600 computing units at a clock rate of 725 MHz

executing 2320 billion floating point operations per second. The applied technology allows the clock rate to be set up to 1000 MHz meaning that the operations per second achieve 3200 billions. In order to utilize the huge computing capacity implemented in the card AMD issued ATI CAL developer environment which provides an effective way to exploit the embedded calculation capabilities. To promote the standardization Chronos group specified the general purpose completion of OpenGL[1] shader language, called OpenCL[2]. Forming a standardized interface of the integration of general purpose calculations, OpenCL[2] makes programmers job easier, hiding the complicated methods of hardware handling. Naturally it leads to decrease in operation speed, but usually the accuracy of developing and testing processes have more importance.

Figure 2. ATI's top quality graphics card: HD5970 [2]

GRAPHICS CARD FUNCTIONS

First let's see what tasks are carried out in such a card in normal operation mode.

Scene calculations

The scene processing can be divided in many coherent steps as described below:

- Application tasks
- Scene level calculations
- Transform and Lighting
- Triangle Setup and Clipping
- Rendering

[1] OpenGL: Open Graphic Library,
[2] OpenCL: Open Computing Language,

Application tasks

The running program controls the movement of the stand point and the objects, and their interactions. This element is usually called application engine. The basis of each pixel is an extracted detail from the modeled state of the whole virtual domain. Having all these information the actual drawing could start but still there would not be any sense in lavishing resources for forming such objects which cannot be seen from the actual viewpoint. As the virtual field is regarded as to be modeled, only the stand point information is provided for the user.

Scene level calculations

In 3D computer graphics, Visible Surface Determination (VSD) is the process used to determine which surfaces and parts of surfaces are not visible from a certain viewpoint. A related area to VSD is *culling*, which usually happens before VSD in a rendering pipeline. Primitives or batches of primitives can be rejected in their entirety, which *usually* reduces the load on a well-designed system.
The advantage of culling early on the pipeline is that entire objects that are invisible do not have to be fetched, transformed rasterized or shaded.
Often, objects are so far away that they do not contribute significantly to the final image. These objects are thrown away if their screen projection is too small.
In computer graphics, accounting for level of detail involves decreasing the complexity of a 3D object representation as it moves away from the viewer or according other metrics such as object importance, eye-space speed or position. Level of detail techniques increases the efficiency of rendering by decreasing the workload on graphics pipeline stages, usually vertex transformations. The reduced visual quality of the model is often unnoticed because of the small effect on object appearance when distant or moving fast.
Although most of the time LOD is applied to geometry detail only, the basic concept can be generalized. Recently, LOD techniques included also shader management to keep control of pixel complexity. A form of level of detail management has been applied to textures for years, also providing higher rendering quality.

Transform and Lighting

On this level the desired geometry can be drawn. To do so, the vertexes positions must be specified by their spatial coordinates (x, y, z). Connecting all vertexes belonging to the same object leads to the frame of the actual model. Then models are transformed and lighting according to the selected viewpoint.

Triangle Setup and Clipping

On this level the necessary calculations for the efficient rendering process are done. Also, invisible vertexes are removed from and behind the models.

Rendering

Rendering is the process of generating an image from a model, by means of computer programs. The model is a description of three-dimensional objects in a strictly defined language or data structure. It would contain geometry, viewpoint, texture, lighting, and shading information. The image is a digital image or raster graphics image. The term may be by analogy with an "artist's rendering" of a scene. 'Rendering' is also used to describe the process of calculating effects in a video editing file to produce final video output

Scanline rendering and rasterization: A high-level representation of an image necessarily contains elements in a different domain from pixels. These elements are referred to as primitives. In a schematic drawing, for instance, line segments and curves might be primitives. In a graphical user interface, windows and buttons might be the primitives. In 3D rendering, triangles and polygons in space might be primitives. If a pixel-by-pixel (image order) approach to rendering is impractical or too slow for some task, then a primitive-by-primitive (object order) approach to rendering may prove useful. Here, one loops through each of the primitives, determines which pixels in the image it affects, and modifies those pixels accordingly. This is called rasterization, and is the rendering method used by all current graphics cards.
Rasterization is frequently faster than pixel-by-pixel rendering. First, large areas of the image may be empty of primitives; rasterization will ignore these areas, but pixel-by-pixel rendering must pass through them. Second, rasterization can improve cache coherency and reduce redundant work by taking advantage of the fact that the pixels occupied by a single primitive tend to be contiguous in the image. For these reasons, rasterization is usually the approach of choice when interactive rendering is required; however, the pixel-by-pixel approach can often produce higher-quality images and is more versatile because it does not depend on as many assumptions about the image as rasterization.
The older form of rasterization is characterized by rendering an entire face (primitive) as a single color. Alternatively, rasterization can be done in a more complicated manner by first rendering the vertices of a face and then rendering the pixels of that face as a blending of the vertex colors. This version of rasterization has overtaken the old method as it allows the graphics to flow without complicated textures (a rasterized image when used face by face tends to have a very block-like effect if not covered in complex textures; the faces aren't smooth because there is no gradual color change from one primitive to the next). This newer method of rasterization utilizes the graphics card's more taxing shading functions and still achieves better performance because the simpler textures stored in memory use less space. Sometimes designers will use one rasterization method on some faces and the other method on others based on the angle at which that face meets other joined faces, thus increasing speed and not hurting the overall effect.

As mentioned above scene calculation is made up of several steps and done by the CPU and the GPU. In the early times all tasks but rendering were carried out by the CPU. As graphics processors have been getting more and more sophisticated GPUs took this role over.

Application tasks	CPU	CPU	CPU	CPU	CPU
Scene level calculations	CPU	CPU	CPU	CPU	CPU
Transform and Lighting	CPU	CPU	GPU	GPU	GPU
Triangle Setup and Clipping	CPU	Graphics chip	GPU	GPU	GPU
Rendering	Graphics chip	Graphics chip	GPU	GPU	GPU
DirectX	5.0 (1997)	6.0 (1998)	7.0 (1999)	8.0 (2000)	9.0 (2002)

It is noticeable that from DirectX7 the hardware supported T&L function appeared in graphics chips so scene calculation steps are done by the GPU. The main disadvantage of this approach was the fact that GPU used a fix algorithm for transformation and lighting levels. However

developers were not able to utilize the GPU's full resources, they were happy with the innovations.

Multipurpose application

If graphics cards are intended to be utilized, a different approach - listed below - should be considered.

- Program preparing task
- Environment preparing task
- Program execution
- Waiting for execution
- Data storage

Program preparing task

During this phase the card capabilities are checked and also modules necessary for the handle of the graphics card are loaded (Access routines are initialized, memory and processors are allocated). Then the card's physical addresses are linked with the application's virtual addresses running on the PC. When accomplishing the program preparing tasks, the applied framework depending on the hardware structure is regarded to be initialized.

Environment preparing task

According to the received addresses an applicable code is generated from the source program. The translation can be carried out in real time, which increases the execution time, or a pre-translated one. Buffers used in the card's program are linked with the PC's address domain then are loaded with data. The data transfer is accomplished with hardware support. When moving data individually between the PC and the chip memory, the GPUs are able to reach considerably high operating speed (~GB/s). It is called direct memory access (DMA[1]). The CPU is notified about the end of data transfer through a so called EVENT[2] object. A CPU's thread is listening to this object whether it is activated by the GPU or not. One of the main advantages of the multi tasking operating systems using preemptive timing is the fact that the listening thread is inactive during the waiting period so that the processor and other recourses are free for other tasks.

Program execution

The stream processors are started to execute the instruction threads. Parallel with this operation the control returns to the CPU so the main program execution is independent from the GPU. Providing synchronization between CPU and GPU is the programmer's task.

Waiting for execution

The synchronization of the stream and the application threads are accomplished during this phase. To make active these EVENT[2] objects is the task of the GPU (with direct hardware

[1] DMA: (Direct Memory Access) direct memory access through dedicated hardware unit.
[2] EVENT object: this object is used for waiting for an external event.

support), while checking these threads is carried out by one of the threads operating within the main program.

Data storage

Now data at the output of the stream processors is available so additional processing can be done. The transfer of the data is accomplished with DMA[1] operations through watching an EVENT[2] object.

STRUCTURE OF GRAPHICS CARDS

The next chapter will study the inner structure of the graphics cards so that it is easier to understand their capabilities.

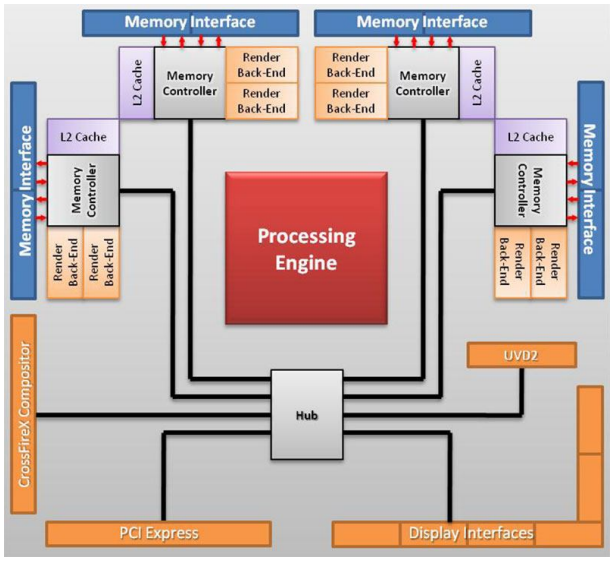

Figure 3. The structure of the AMD made RV700 GPU. [3]

Figure 3. clearly shows that the central HUB provides physical connection between the units inside the processor. Also it is responsible for communication and traffic control. The monitor is connected to the card through the display interface unit. The card communicates with the PC through PCIExpress interface. In multi-card applications, cards are connected to each other through CrossFireX interface. The memory blocks embedded on the card are connected to the GPU's processing engine through wideband (128-256 Bits) memory interface. Generally the memory is segmented by the processor so the processes competing with each other have parallel access to the memory. Also the processor contains the UVD2[1] unit which is responsible for coding-decoding high resolution video streams.

[1] UVD2: Unified Video Decoder version 2, general purpose video decoder unit

The structure of the processor unit (GPU)

The GPU is specially designed and developed for executing multithread processes running parallel. Its structure contributes the principle of SIMD (single command-multidata). The GPU is controlled by the command processor which is responsible for dispatching the threads, for accomplishing memory operations (DMA), and for signing the events. The processor consists of several SIMD engines (the number of the engines depends on the version of the processor). SIMD engines are the master unit of program execution. Each of them executes different programs. Data share between SIMDs is carried out through the global and the local data share memory blocks. Accessing local memory is considerably faster than the global one as it is divided between the units, but the global memory is available for all SIMD units.
The next picture shows the GPU containing the most SIMD engines so far.

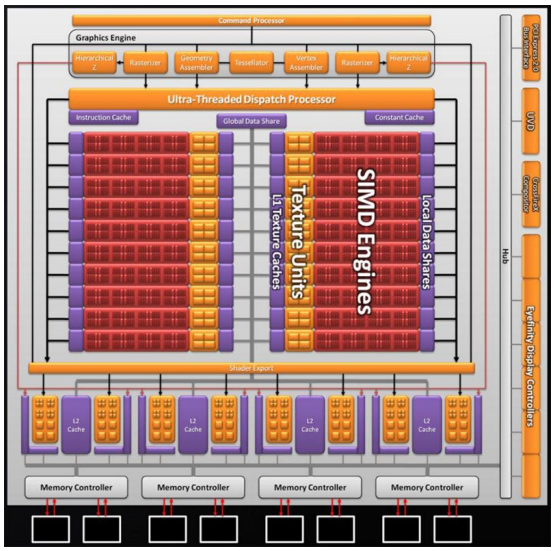

Figure 4. GPU RV870 produced by AMD. [4]

The structure of the SIMD engines

Thread processors inside the SIMD unit executes the programs. Sixteen of them are embedded in a SIMD unit. All of the thread processors execute the same code with different data. The main advantage of this is the fact that an operation on a parallel data source (matrix multiplication) is accomplished in one sixteenth times less. On the other hand, its disadvantage is the request of the 16 threads multi memory access at a time. A thread processor is made up of 4 simple and one double point stream core. It means that in one clock cycle it can execute 4 floating point operations or one double point operation. Every one of the stream cores consists of 1024 pcs of general purpose registers which can be accessed according to the clock cycle. All of them are a 128 bit wide one that can store four of 32 bit floating point or **integer type** data. Coupling the sub-registers (marked with x, y, z, w) double floating point or **integer type** operations are achievable. But during these operations there is no possibility for 4 times rate execution.

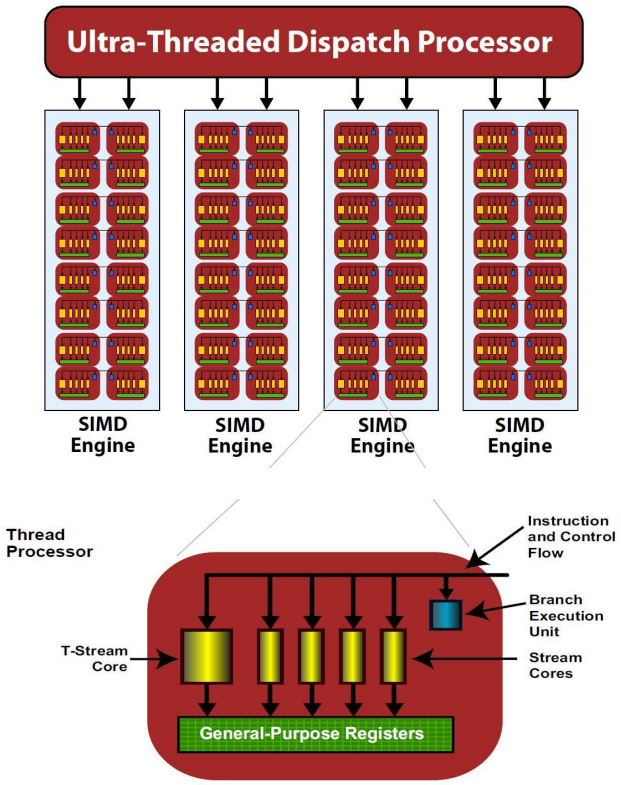

Figure 5. The structure of the SIMD engines and the thread processors [5]

The picture above shows the structure of the SIMD engines and the thread processors.
In fact, these units execute the calculating operations and store their results. To produce very effective codes it is essential to take into consideration the number of the registers and the operating units in the executing cores.

The operation of the video cards, the shaders [6]

The programs created directly for running on graphics cards are called shaders.
The Direct3D and OpenGL graphic libraries use three types of shaders.
Vertex shaders are run once for each vertex given to the graphics processor. The purpose is to transform each vertex's 3D position in virtual space to the 2D coordinate at which it appears on the screen (as well as a depth value for the Z-buffer). Vertex shaders can manipulate properties such as position, color, and texture coordinate, but cannot create new vertices. The output of the vertex shader goes to the next stage in the pipeline, which is either a geometry shader if present or the rasterizer otherwise.
Geometry shaders can add and remove vertices from a mesh. Geometry shaders can be used to generate geometry procedurally or to add volumetric detail to existing meshes that would be too costly to process on the CPU. If geometry shaders are being used, the output is then sent to the rasterizer.
Pixel shaders, also known as fragment shaders, calculate the color of individual pixels. The input to this stage comes from the rasterizer, which fills in the polygons being sent through the graphics pipeline. Pixel shaders are typically used for scene lighting and related effects such as bump mapping and color toning. (Direct3D uses the term "pixel shader," while OpenGL uses the term "fragment shader." The latter is arguably more correct, as there is not a one-to-one relationship between calls to the pixel shader and pixels on the screen. The most common

reason for this is that pixel shaders are often called many times per pixel for every object that is in the corresponding space, even if it is occluded; the Z-buffer sorts this out later.)
The unified shader model unifies the three aforementioned shaders in OpenGL and Direct3D 10.
As these shader types are processed within the GPU pipeline, the following gives an example how they are embedded in the pipeline:

Simplified graphic processing unit pipeline

The CPU sends instructions (compiled shading language programs) and geometry data to the graphics processing unit, located on the graphics card.
Within the vertex shader, the geometry is transformed and lighting calculations are performed.
If a geometry shader is in the graphic processing unit, some changes of the geometries in the scene are performed.
The calculated geometry is triangulated (subdivided into triangles).
Triangles are transformed into pixel quads (one pixel quad is a 2×2 pixel primitive).

General computing task

It is concluded from the Vertex structure that freely programmable graphics processors can not only be used for special rendering tasks but also for general purpose parallel operations.
It is only a step ahead to get contribution for the efforts to produce appropriate and effective developer environment as it can be realized in OpenCL and DirectX11.

Multichannel receiver

Regarding graphics chips during their general computing processes the most optimized operations are those which contains the same operations for the input stream with different data. The multichannel receiver accomplishes the same theory. It receives different carriers with the same bandwidth. On account of the relatively small bandwidth (max.: 9 kHz) and the high number of the channels it can be used the in the most effective ways in HF applications.

The structure of the receiver

The following figure shows the structure of the multichannel receiver.

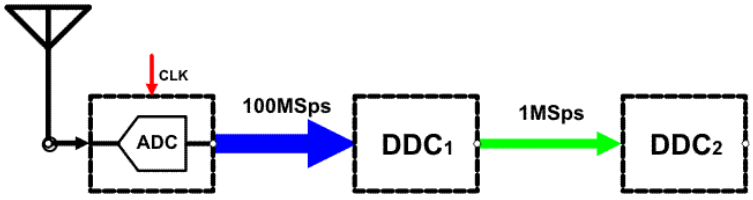

Figure 6. The block diagram of a multi channel receiver.

The incoming RF signal is digitized with a high speed ADC. From the full HF band (0-30 MHz) 500 kHz sub bands are extracted with a DDC unit. The DDC allows us to select the interested band, and to shift its frequency down to base band reducing the sampling rate. In the above case, the initial sampling rate is reduced from 100 MSps to 1 MSps. These amounts of data can be handled by video cards[1].

[1] Based on real tests, the number of the processable samples is a maximum of 4 million/second.

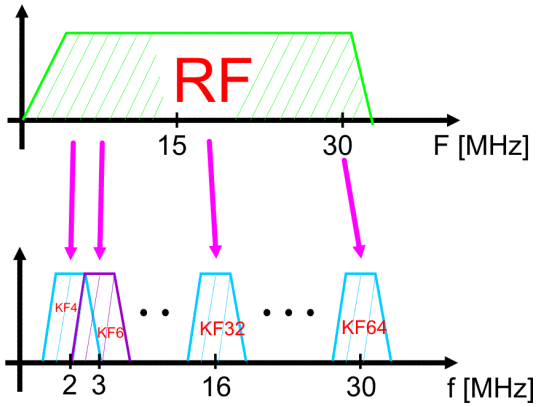

Figure 3. Sub-band forming

The first DDC unit processes the high input data rate. Its output data is put in a 2 dimensional array where the columns represent the time domain and the rows represent the frequency domain. So it is easy to implement the shift between the frequency bands. The task of the second DDC unit is to simultaneously receive narrow band channels from the 500 kHz bands. This is carried out by the software applied in the video card so its output represents only the selected channel. With this method the single sideband super heterodyne receiver can be implemented. Of course the solution of demodulation and decoding process is accomplished by a different unit which works in an out of real time environment.

Mathematical description

Figure 7. shows the structure of a DDC unit.

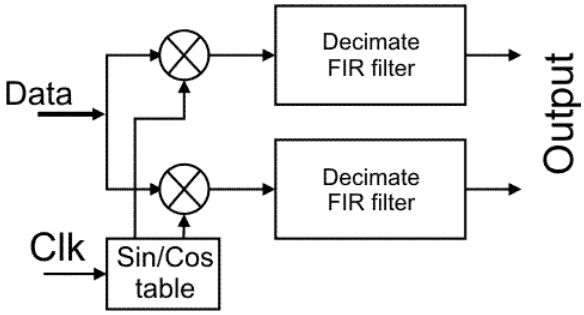

Figure 7. The structur of the DDC [7]

The picture above shows that the input data is multiplied at a time by sine and cosine having increasing arguments so that the received frequency is converted down to 0 Hz.

$$I_{out}[t] = D_{in}[t] * \sin(\omega_c * t)$$

Where,
I: the output data,
D: the input data
ω_c: the interested carrier.

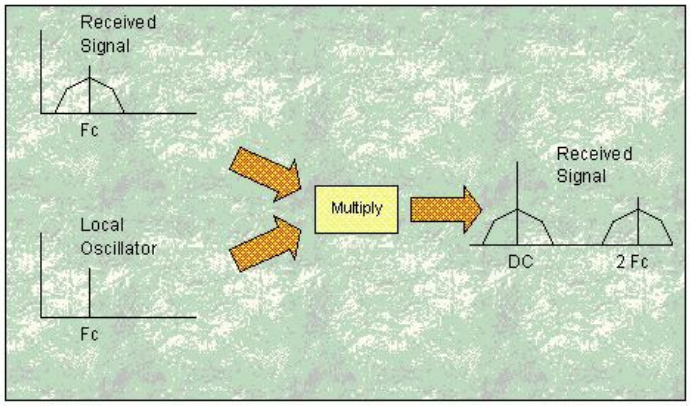

Figure 4. The result of multiplication [8]

According to the formula it is obvious that the high accurate sine and cosine values required for the multiplication of the incoming data cannot be derived from a table due to the high number of the values. During the multiplication process along with the base band signal a double frequency component also appears. This component must be eliminated with the FIR filter right after the multiplication unit. The FIR output data has to be calculated in the case when the sample of the pre set bandwidth is requested. Saving considerable resources the FIR routines have to be applied at only every n^{th} sample.

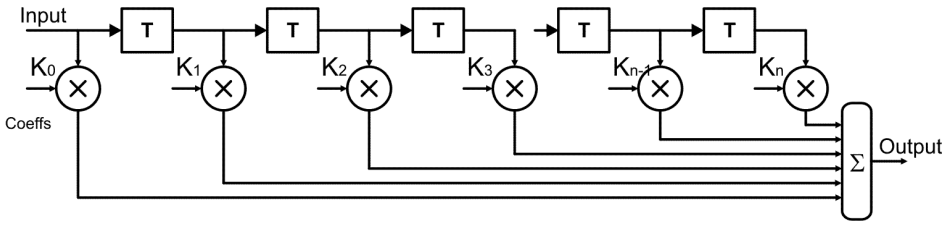

Figure 5. The structure of a FIR unit

It is concluded that the number of multiplying and adding operations done on every single multiplied data depends on the goodness of filtering. [9]

$$D_{out}[t] = \sum_{k=0}^{K} I[t-k] * K_k$$

Where:
D: the output data,
I: the input data,
K: the coefficient values for filtering

Capabilities

According to the above mentioned structure, a video card can process approximately 320 narrowband channels in the full HF (0-30 MHz) band. Implementing more cards in the system, the number of the channels also increases up to 512 as the limit of channels processed due to the computer capacity.

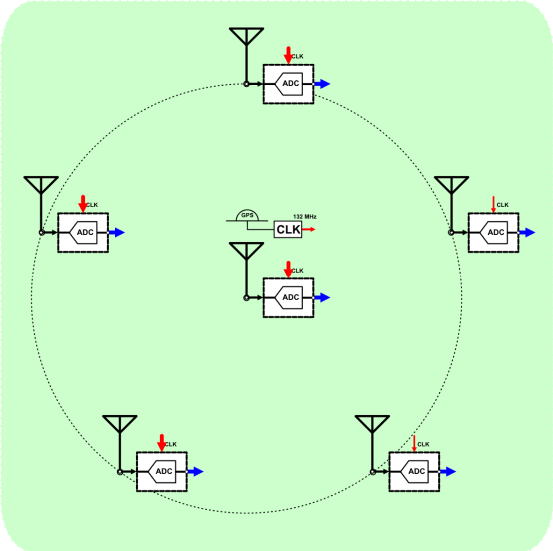

Figure 6. Direction finding antenna system

Figure 10. shows the structure of a real time direction finding system in which many parallel receiver channels are processed. The decrease of the number of the processed channels is directly proportional to the number of the antennas working parallel. In case of a 6 antenna system, 50 channels can be processed simultaneously. Using phase accurate adding up, digital beamforming can also be accomplished at a higher signal to noise ratio.

References

[1] Display resolution
http://en.wikipedia.org/wiki/Display_resolution, Downloaded: 2009.12.01.
[2] Radeon HD5970: Egy kártya mind felett,
http://www.ipon.hu/_userfiles/Image/Svindler/5970/Sapphire_box_big.jpg,
Dowloaded: 2009.12.03.
[3] Pintér Gábor: ATI Radeon HD 5870 - beköszönt a DirectX 11-es korszak,
http://www.hoc.hu/teszt/548-ati-radeon-hd-5870--bekoszont-a-directx-11-es-korszak/2-technologia-felepites.html, Downloaded: 2009.12.04.
[4] Pintér Gábor: ATI Radeon HD 5870 - beköszönt a DirectX 11-es korszak,
http://www.hoc.hu/teszt/548-ati-radeon-hd-5870--bekoszont-a-directx-11-es-korszak/2-technologia-felepites.html, Downloaded: 2009.12.04.
[5] ATI Stream Computing - AMD Corp.
http://developer.amd.com/gpu_assets/Stream_Computing_User_Guide.pdf, p 15.
Downloaded: 2009.12.01.
[6] Shader, http://en.wikipedia.org/wiki/Shaders, Downloaded: 2009.12.01.
[7] Tony J. Rouphael: RF and Digital Sigal Processing for Software-Defined Radio, Elsevier 2009, p.65.
[8] HUNT ENGINEERING: Digital Down Conversion theory,
http://www.hunt-dsp.com/pdfs/tech/ddctheory.pdf, Downloaded: 2009.12.01.
[9] Tony J. Rouphael: RF and Digital Sigal Processing for Software-Defined Radio, Elsevier 2009, p.73.