

## X-Stream II

TECHNICAL BRIEF

*Peter J. Pupalaiakis  
Principal Technologist  
June 16, 2008*

### Summary

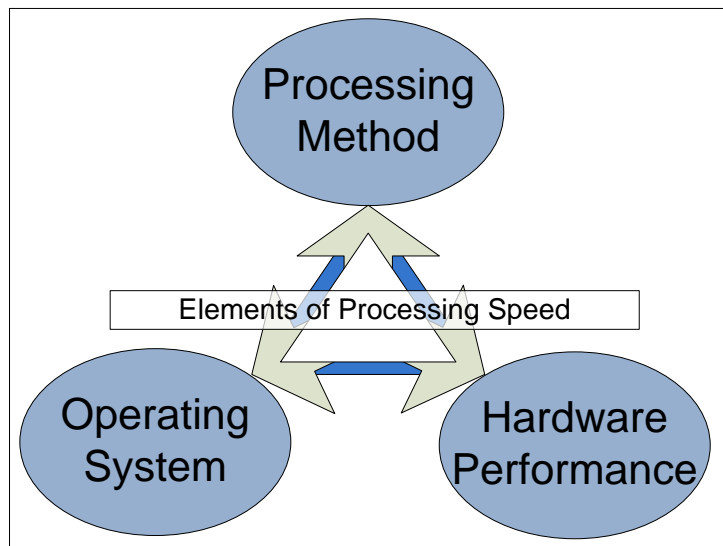
*This paper explains how X-Stream II technology improves the speed and responsiveness of LeCroy oscilloscopes.*

The digital oscilloscope of today is required to process very long waveforms in a very complex manner in order to provide measurements that provide insight into system and circuit behavior. The capability of the oscilloscope to provide insight is inextricably linked with the concept of speed and responsiveness; this because the design engineer cannot wait too long for the answers and because he must be able to drive the instrument comfortably and confidently.

The elements of processing speed can be broken into three areas:

1. The processing and waveform readout hardware characteristics.
2. The operating system
3. The method of processing data embodied by proprietary software.

The purpose of this paper is to explain how LeCroy ties the design of the oscilloscope in terms of hardware performance and operating system together with proprietary processing methods to optimize speed of complex, long waveform processing.



---

The processing hardware is dominated by the processor, including the number of bits in the processor, its instruction set and instruction set extensions, the number of processor cores the clock speed, and the cache memory. Of further importance is the front-side bus speed, the amount and the speed of main memory. Of particular importance to the oscilloscope is the readout speed; the speed that data is transferred from acquisition memory into main memory.

The operating system is important because it provides support for multiple cores and for multi-threading. Most importantly of late is to support the need for larger amounts of addressable memory. Lately, when we talk about 64 bit processors and 64 bit operating systems, we are really talking about 64 bit address buses that can address huge amounts of main memory. Finally, while not completely operating system related but more related to the processor are tools for handling multi-core and processor instruction set extensions. These include open openMP and performance primitives for signal processing and mathematical operations.

The new LeCroy WavePro 7000Zi line of oscilloscopes utilizes the most powerful hardware and operating system components available. We use an Intel® Core™ 2 Quad (four cores) each operating at 2.5 GHz. It is a 64 bit processor with 6 Mb of level 2 cache and a front-side bus operating at 1.33 GHz. It has a built in floating point unit and supports Streaming SIMD (single-instruction / multiple data) 4 or SSE4 instruction set extensions. It contains up to 8 Mb of DDR II main memory. The transfer from acquisition memory to main memory utilizes direct memory access (DMA) and employs four lanes of PCIe gen I serial links. These links move data into main memory at a rate of up to 800 Mpoints per second without processor intervention. Because LeCroy uses the Microsoft® Vista™ 64 bit operating system, the scope application can address all of the available memory and more (32 bit operating systems address up to 4 Gb max with typically only 1 Gb available to an application).

Regarding processing methods, LeCroy employs a proprietary method that makes optimal use of cache

memory. In order to properly understand this use of cache, it is useful to understand how the microprocessor and microprocessor based architectures have evolved over time.

About forty years ago when the microprocessor was born, the simple embedded computer had one memory bus. Generally, it had a program in non-volatile storage, usually an EPROM, and some storage area in volatile SRAM. The memory bus operated at a very predictable speed dictated by the system clock. In those days, if you wanted to determine the performance of your system, you could simply count up the number of instructions and the number of clock cycles per instruction for a given task and multiply the total cycles by the clock period. The advantage was that the design engineer could totally predict the performance. The disadvantage was that there was really only one lever he could apply to speed things up – reduce the number of instructions.

Over time, Moore's law led to dramatic changes in CPU power. Moore's law predicts a doubling of transistor density every 18 months, which leads to a doubling of processing speed every three years. Both the exponential increases in speed and density have yielded processors that go faster and handle more complex instructions. The handling of complex instructions has led to onboard barrel shifters, floating point units, and to powerful added instructions for multimedia and digital signal processing. A quad core processor operating at 2.5 GHz, as used in the LeCroy WavePro 700Zi oscilloscope using instruction set enhancements that work on four packed data elements simultaneously in a single instruction per core could potentially operate at 40 billion floating point operations per second.

The Moore's law progression has led to unbelievable increases in CPU performance, but this performance has not always been translatable to the periphery of the chip as indicated by the front-side bus speed or the speed of access to main memory.

The solution to this speed problem is the cache memory.

Cache memory architectures involve a cache controller with a relatively small amount of memory internal to the processor. It operates to broker transactions between the thirsty processing hardware and the relatively slow main memory. On every access to memory that the processor tries to make, the cache controller checks to see if the desired data exists in the cache. If so, the data is sped from the cache memory to the processor. If not, a so-called “cache-miss” occurs and we say that the processor stalls. The cache controller holds up processing and goes to main memory to get the missing data. Usually, to keep things moving briskly, the cache accesses more memory than is requested in an operation

called a “cache-line fill”. In this operation, multiple data elements around the requested element are accessed in a burst transfer mode. In this way the cache controller keeps the most recently accessed data in its cache memory. Most modern

cache architectures implement a “write-back” cache. This architecture suspends writes until they are required. During a cache-miss, the cache controller must flush out the oldest data elements to make room for the new data and it is during this time that data writes occur as well.

This paper does not have the room to explain all of the nuances of cache architecture designs, but it is sufficient to point out that the performance of the system can be highly unpredictable. The unpredictability is based on how data is processed. From the previous explanation, one can see that a cache architecture causes the performance to be limited by the internal CPU power and speed only when nearby data elements are operated on very frequently. The performance will be limited by the transfer speed from main memory to the CPU cache

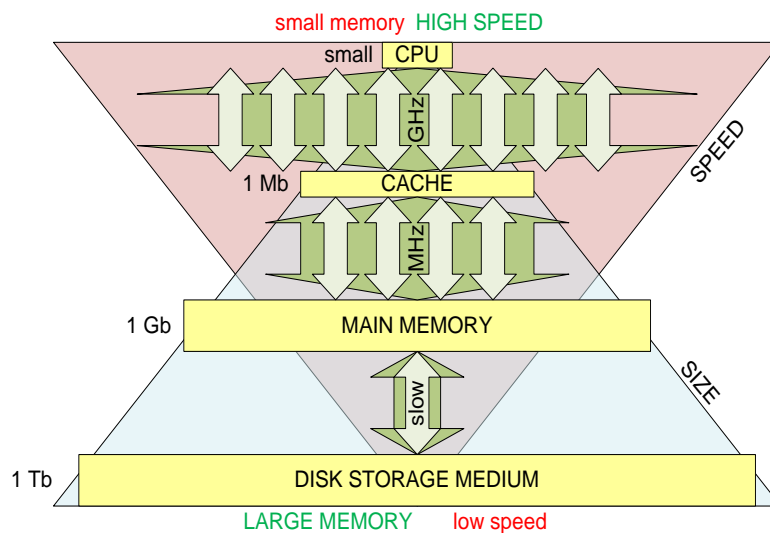
when the data elements operated on are frequently different or are located far away from each other. What governs the determination of frequency or distance of data elements can be made by simply determining whether the data being operated on fits in cache memory. Of course for a digital oscilloscope operating on long waveforms the data elements can never fit entirely in the limited cache, but the goal is to limit the number of transfers between main memory and cache memory as much as possible in order to optimize speed of processing.

Cache memory is a marvelous thing and has the possibility of dramatic speeds of processing. The

problem is that optimum use of cache is directly in conflict with the ease of programming long memory processing applications. I will try to illustrate the situation here with an example:

Consider processing that is performed on an oscilloscope waveform. Let’s consider the processing as a series

of three steps. The actual type of processing is not important to discuss so let’s say that first we will apply processing function A to a waveform, and then function B followed by function C. For the purposes of this discussion, let’s say that these functions are unary and apply to each element in the waveform. The traditional and easiest way to perform this processing is create an intermediate data buffer and then for each point in the waveform, apply processing function A to each point read from the input waveform and store the result in the buffer. Then, for each point in the buffer, processing function B is applied and stored back in the buffer. Finally, for each point in the buffer, processing function C is applied and stored back in the buffer.



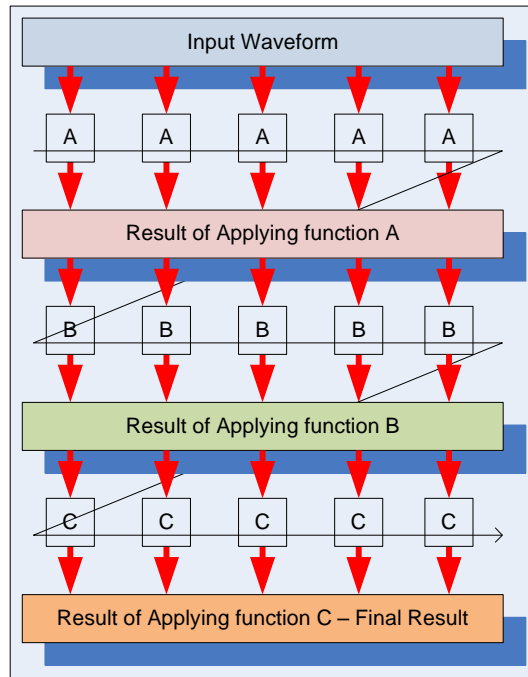
Recalling the operation of cache memory, one can see that while the software to perform this processing is relatively straightforward, it is not optimal from a cache memory standpoint, especially when applied to long waveforms. To understand this, consider the fact that during the processing of function A, each element will need to be read from main memory into cache. This is unavoidable. Assuming a modern cache architecture, If the waveform is small enough to fit into the cache then it will stay there even if the memory is written into the buffer after function A is applied. This means that even though the software instructions store the result into the buffer, the cache controller suspends this write until it needs to push something out of cache. Again, only if the waveform can fit into cache, when function B and C is applied, every data point exists in the cache and the reading, processing and storing of the waveform back in the buffer generates no bus activity. Since the bus is much slower than the CPU, the CPU operates at its maximum speed. This description shows an optimum situation. In this small waveform example, the data is read once from main memory into cache and eventually written once from cache to main memory even though many data reads and writes were coded in the software.

Now, let's consider a long memory case. Like the short memory case, as function A is applied data is moved from main memory to cache

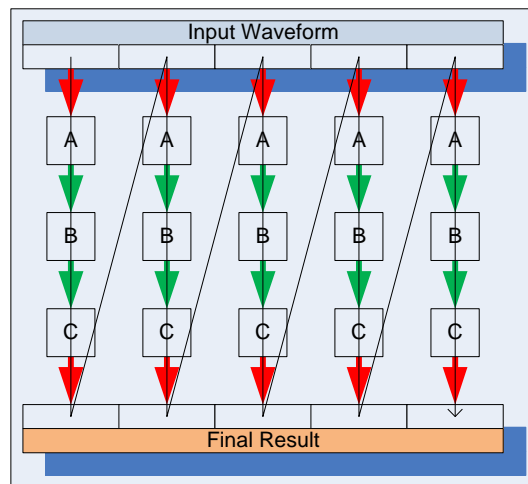
on the first access. When function A is applied and stored in the buffer, no bus activity is generated yet. But, as processing continues and data is accessed from the input waveform, eventually the cache runs out of space and must make room for new data. At that point in time, the earlier buffer writes must be resolved and the data is written to main memory (i.e. is pushed out of cache) to make room for the new data. On each subsequent read of data, old cached data must be pushed out to make room for the new data request. This means that virtually every data access to perform function A results in a read and write operation between main memory and cache – only because the waveform did not fit in the cache.

Now consider function B. At the beginning of processing function B, the cache contains the end portion of the result of function A. This means that the first elements of the data buffer are not in the cache. Therefore, each data element processed for function B also causes a read and write operation between main memory and cache.

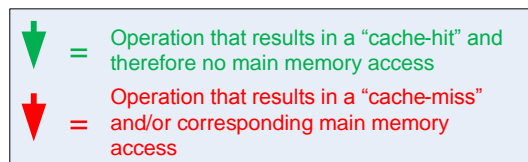
Contrasting the short waveform and long memory behavior, the short waveform (assumed to fit in cache) needs one write and one read access for each data element processed, regardless of the number of operations applied to the data element. The long waveform (assumed not to fit in cache) needs one write and one read access *per processing function applied*. It is important to consider the impact of this. Not only are there more transfers between cache and main memory, but in



**Traditional Processing Method**



**Streaming Processing Method**



most cases, the number of transfers will be the total bottleneck for system performance. In long waveform cases, the bus transfer speed will completely determine the performance. In short waveform cases, the processor speed and processor core utilization will determine the performance. The latter is desirable because it is much faster.

In oscilloscope usage, processing needs are becoming more and more demanding and the lengths of waveforms are becoming longer and longer. This means that for serious processing cases, the oscilloscope must perform complex, cascaded operations on very long waveforms. So how do you get short waveform type of performance where the processor capability dominates on long waveforms where the bus speed bottleneck tends to dominate? The answer is part of the LeCroy X-Stream architecture.

X-Stream processing behavior can be summarized most succinctly as follows:

For long waveforms, the waveform is broken into many smaller waveforms. These smaller waveforms are processed in a manner that optimizes the cache utilization. Finally, the smaller waveform results are reassembled in the end into the final long waveform result.

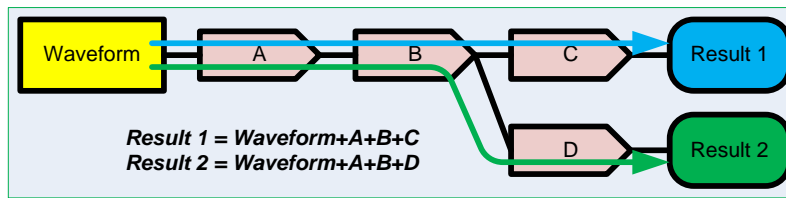
This all sounds very simple, but in practice the architecture necessary to achieve this is very complicated. LeCroy has several patents on this technology. It is the reason that LeCroy oscilloscopes have traditionally outperformed the competing instruments in processing speed by huge factors.

LeCroy introduced the X-Stream architecture in 2002 with the WaveMaster. At that introduction, LeCroy produced the fastest processing scope around. That's what X-Stream was all about – fast

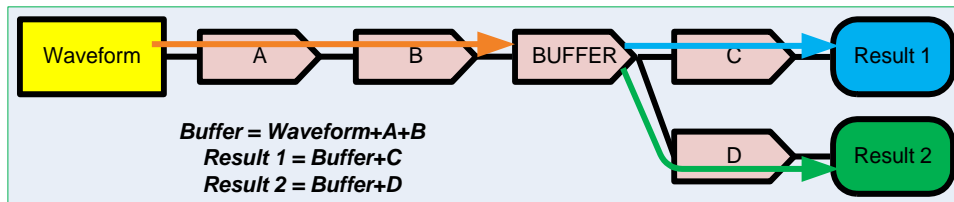
processing. Since that time, many architectural improvements were envisioned which culminated in X-Stream II introduced with the WavePro 700Zi

. While X-Stream I was all about fast processing, X-Stream II is all about responsiveness. The key elements of X-Stream II include:

1. Dynamic buffer placement to improve situations where the streaming architecture falters.
2. Preview modes that allow quick, preliminary views of waveform results during zooming and scope adjustment.
3. Processing abortability that enables stopping during scope adjustment.



**Simple Streaming Processing**



**Streaming Processing with Dynamic Buffer Placement**

Dynamic buffer placement involves the placing of buffers in the processing stream at

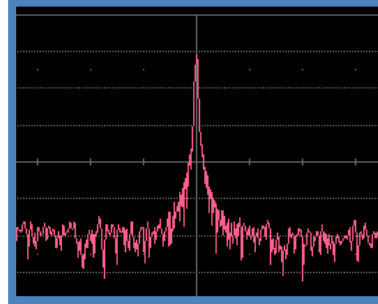
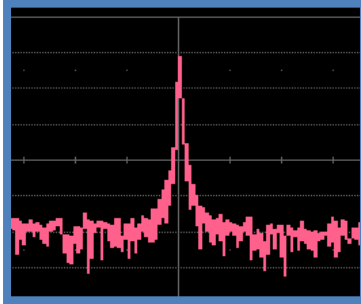
strategic locations to improve processing throughput. To understand this effect, remember that the conventional processing model involves buffers at every step of the process. It can be viewed as a buffer placed in between every single processing element. In the streaming model that LeCroy employs, small buffers that fit in cache are employed everywhere, but a buffer containing the full results of processing would only be placed at the end of the processing stream, if at all. In X-Stream II, an optimization is performed in situations where multiple processing streams pass through common processing elements, as in the figures. In many cases, processing data multiple times in the cache-friendly way still optimizes the speed of processing,

but when the processing is intensive and complex, this is not the case.

In these situations, the LeCroy X-Stream II architecture places buffers into the processing stream dynamically to avoid multiple calculations. In this manner, X-

Stream II is an intelligent blend between traditional and streaming methodologies but in all cases optimizes the throughput.

The X-Stream II optimizations for responsiveness are tied to the capability to both preview results of processing and to abort processing. The way these two work together is that whenever a change is made to a scope setting, the software simultaneously begins calculating new results and also a preview result based on rescaling of the picture on the screen. If the preview finishes before the processing, the user sees the preview followed



Preview vs. Final Calculation Result

by the updated result of the processing when it finally finishes. If the processing finishes first, the user sees the result of the processing only. If, during calculation of the processing, the user modifies a setting, the processing is aborted and restarted using the

new changes and the user only sees the preview result. The result of this behavior is important when the user is trying to set up the oscilloscope or is making rapid changes to the settings based on measurement observations. It is really helpful when the processing is complex and time consuming and the waveforms are long. Using traditional processing methods available in other oscilloscopes, the user must wait for the processing to complete before seeing the display result and making another setting change. In some cases, the processing can take more than ten seconds and if the user wants to make multiple settings changes,

he must wait for the results of each long processing interval before making another setting change. This type of operation can be very frustrating.

In conclusion, fast oscilloscope processing is a combination of the right hardware and operating system and congruent processing methods. X-Stream II is a processing method technology that extends the cache-friendly processing methods LeCroy introduced in X-Stream I and adds dynamic buffer placement for further speed optimization and adds result previews and processing abort capability for superior oscilloscope responsiveness.

	User Interface Thread	Preview Thread	Processing Thread
The user is viewing the result of a 10 Mpt FFT – He wants to zoom in to 500 Kpts. He will turn the zoom control knob four clicks to see what he wants			
The user zooms from 10 to 5 Mpts	1 Change to 5 Mpts		
The scope begins calculating a zoom preview of the result of zooming and also begins calculating the FFT		Zoom preview Update display	Start Calculating 5 Mpt FFT
The user sees the result of the zoom preview – FFT still calculating			abort
The user zooms again – the FFT calculation is aborted	2 Change to 2 Mpts		
The scope again begins calculating a zoom preview of the result of zooming and also begins calculating the FFT		Zoom preview Update display	Start Calculating 2 Mpt FFT
The user again sees the result of the zoom preview – FFT still calculating			abort
The user zooms again – the FFT calculation is aborted	3 Change to 1 Mpts		
The scope again begins calculating a zoom preview of the result of zooming and also begins calculating the FFT		Zoom preview Update display	Calculate 1 Mpt FFT
The user sees the result of the zoom preview			Update display
The FFT completes and he sees the result			
The user zooms again	4 Change to 500 Kpts		
The scope again begins calculating a zoom preview of the result of zooming and also begins calculating the FFT		Zoom preview	Calculate 500 Kpt FFT
Finally, since the FFT is smaller, it completes before the preview. The preview is aborted and the user sees the full result of the FFT calculation		abort	Update display