WP127 (v1.0) March 6, 2002

# *Embedded System Design Considerations*

*By: David Naylor*

Embedded systems see a steadily increasing bandwidth mismatch between raw processor MIPS and surrounding components. System performance is not solely dependent upon processor capability. While a processor with a higher MIPS specification can provide incremental system performance improvement, eventually the lagging surrounding components become a system performance bottleneck. This white paper examines some of the factors contributing to this. The analysis of bus and component performance leads to the conclusion that matching of processor and component performance provides a good cost-performance trade-off.

## Introduction

Today's systems are composed of a hierarchy, or layers, of subsystems with varying access times. Figure 1 depicts a layered performance pyramid with slower system components in the wider, lower layers and faster components nearer the top. The upper six layers represent an embedded system. In descending order of speed, the layers in this pyramid are as follows:

1. CPU
2. Cache memory
3. Processor Local Bus (PLB)
4. Fast and small Static Random Access Memory (SRAM) subsystem
5. Slow and large Dynamic Random Access Memory (DRAM) subsystem
6. Input / Output (I / O) subsystem, which could be a Small Computer System Interface (SCSI), Fibre Channel (FC), Peripheral Component Interconnect (PCI), or another of many bus or communications protocols
7. Disk array – Integrated Drive Electronics (IDE) interface, SCSI, or FC
8. Tape subsystem

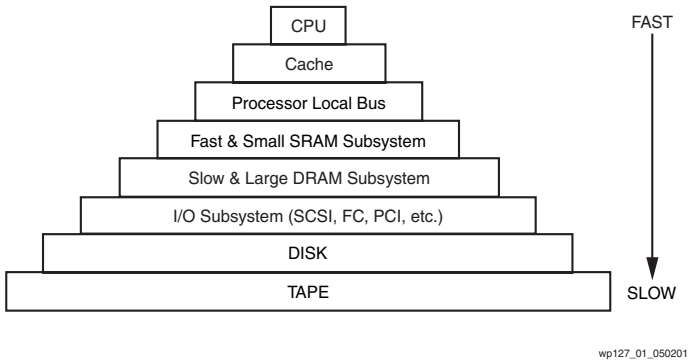


*Figure 1:* **System Component Access Times**

This hierarchy highlights the change in magnitude of the access times as the pyramid is traversed downwards. For example, assume that the CPU executes 32-bit instructions at 200 MHz (5 ns cycle time), while an Integrated Drive Electronics (IDE) interface disk drive transfers data at 33 megabytes per second (MB / s). On a 32-bit bus, 33 MB / s is equivalent to 8.25 MHz operation (121 ns cycle time). Therefore, the 32-bit CPU is running 24 times faster than the hard disk interface.

A tape drive takes many seconds to physically move a tape to the beginning of a volume and become ready to access data. Tape drives today stream data at 5 MB / s to 15 MB / s. Translating a data rate of 5 MB / s gives a 1.25 MHz (32-bit) transfer rate. In this case, a 200-MHz CPU is running 160 times faster than the tape drive interface. The bandwidth mismatch between the CPU and the magnetic peripherals is dramatic.

Disparities in speed also exist between the CPU and surrounding components within the bounds of the embedded system itself. As shown in Figure 1, the transfer times within the embedded system increase with distance from the CPU. The next sections examine some of these areas from the perspective of a small embedded computer system.

## Embedded Systems

The generally accepted definition of an embedded system is one dedicated to a particular application, for instance, a Fibre Channel controller, a disk controller, or an automobile engine controller. A non-embedded, general-purpose system typically executes many different kinds of application programs, like a desktop PC, which is used at various times to run word processing, spreadsheet, or engineering design software. The block diagram for a small embedded system is shown later in Figure 2. This white paper examines some of the component interactions and highlights the value of analyzing and matching their performance. The following sections correspond to the top six layers of the pyramid in Figure 1.

### CPU Choices

Today, CPUs are available with widely varying feature sets. Some key features include raw MIPS performance, on-chip level 1 (L1) cache configurations and sizes, number of execution units, number of on-chip registers, and local bus interface architecture (notably the choice between Von Neumann architecture with a single shared instruction and data bus versus Harvard architecture with independent instruction and data buses).

### Cache Configurations

#### Trade-Off #1: SRAM Price vs. Cache Performance

Cache is small, fast memory which is used to improve average memory response time. To maintain full speed, the CPU must have instructions and data in an internal cache, thus avoiding any need to access external memory. Level one (L1) cache is the highest speed memory in the system and is most often on board the CPU device itself. Some CPUs do not have on-chip cache, in which case the L1 cache is high-speed external SRAM tightly coupled to the CPU, with the capability of operating at near-CPU speeds. High-speed SRAM is very expensive, so typically a price vs. performance analysis is done to select the most cost-effective cache configuration for the particular system.

Unfortunately, cache is not usually large enough to contain the entire executable code base, so the CPU must periodically go off-chip for instructions and data. When the CPU is forced to make external accesses, PLB speed (i.e., the path through which the CPU communicates with other components) and main memory subsystem performance become critical system bottlenecks.

### Processor Local Bus

#### Trade-Off #2: PLB Speed vs. Width

The PLB is the CPU data path directly connected to the CPU. The PLB is the fastest parallel bus found in the system. Ideally, the PLB and all local peripherals interfaced to the PLB run at full CPU speed; however, fast buses external to the CPU are constrained by several factors. Printed circuit board (PCB) layout is complex due to the number of individual traces to be run (32 to 64 data bits, about the same number of address bits, etc.). Special techniques for high frequency signal routing add time and expense to the board creation process. Crosstalk and unexpected amplitude excursions are only two of many signal integrity issues that serve to complicate trace routing. These considerations generally lead to tradeoffs between PLB speed and width.

## Main Memory Subsystems

### Trade-Off #3: SRAM vs. SDR SDRAM vs. DDR SDRAM

System throughput is affected by how quickly memory can feed a fast CPU. In embedded systems, the next bank of memory beyond the L1 cache is often the main memory bank. Memory technology is constantly changing in an attempt to keep pace with high-speed processors. SRAMs are faster, less dense, and more expensive than DRAMs. Therefore, SRAMs are usually limited to cache applications. Main memories tend to be composed of DRAMs. Embedded system memories have evolved from Fast Page Mode (FPM) and Extended Data Out (EDO) DRAM types to Single Data Rate (SDR) Synchronous DRAM (SDRAM) to Double Data Rate (DDR) SDRAM. Looming on the design horizon are Quad Data Rate and other innovative memory technologies.

Memory selection is considered in detail in **Some SDRAM Performance Comparisons,** page 7.

## I/O Subsystem Interaction

Computer systems are not self-contained and must exchange data with other systems through I/O controllers, requiring additional off-chip accesses by the CPU. I/O peripherals are typically accessed through a PLB connected to some other standard-bus bridge. I/O controllers characteristically embed an interface to a standard bus, rather than a specific CPU bus interface. Thus, the processor's local bus must be translated to this standard using a bridge device.

I/O peripheral operations can also require access to components on the PLB (through the bridge). If the CPU needs access to the PLB at the same time an I/O controller is using the PLB, the CPU has to wait, possibly stalling its instruction execution. I/O controllers usually interface to other devices external to the embedded system, like the disk arrays and tape subsystems in the lower layers of the pyramid in Figure 1.

## Software

Software affects the performance of an embedded system by virtue of its optimization (e.g., compactness – smaller, tighter code consumes less memory) and patterns. Patterns dictate which hierarchical portion of the pyramid in Figure 1 that the system is exercising, and for what percentage of the time.

Software has a critical affect on the ability of a system to fully utilize its CPU's raw MIPS capability. If all the code can fit within on-chip CPU cache, then the CPU can execute at its full MIPS specification, and in such a scenario, the MIPS capability of the CPU would be of paramount importance. More often than not, all the code does not fit within the cache. These considerations are approximated in the memory analysis that follows; however, software details are not expanded upon in this white paper.

In the following section, a small system is discussed and the frequency characteristics of the local bus are compared to increasing CPU frequency.

# Example Embedded System

Figure 2 shows a block diagram of a small, embedded system with on-chip cache. Contemporary embedded CPU on-chip cache sizes are from 8 KB to 32 KB. It is worth noting that code kernels are also in the 8 KB range on the low end, while an RTOS kernel might consume 64 KB to megabytes. Note that application specific code is *not* included in these numbers.

A typical 8 KB kernel fills an 8 KB cache. After boot, as the kernel is executed, it is gradually pulled into the cache. Once the kernel has completed system initialization,

the application code is launched and the cache begins filling. If the application code makes calls to kernel routines, the cache is thrashed as the kernel is swapped out for the application code and vice versa.

Cache thrashing occurs when a commonly used location is displaced by another commonly used location. Every time the CPU cannot find what it needs within the cache, it must perform a main memory cycle at the slower speed of main memory. Consequently, CPU performance suffers as the CPU waits for the cache data to become available. A larger cache holds more data local to the CPU and thus reduces the need for external memory accesses. Plainly, the larger the cache, the better the CPU performance. However, larger caches cost more.

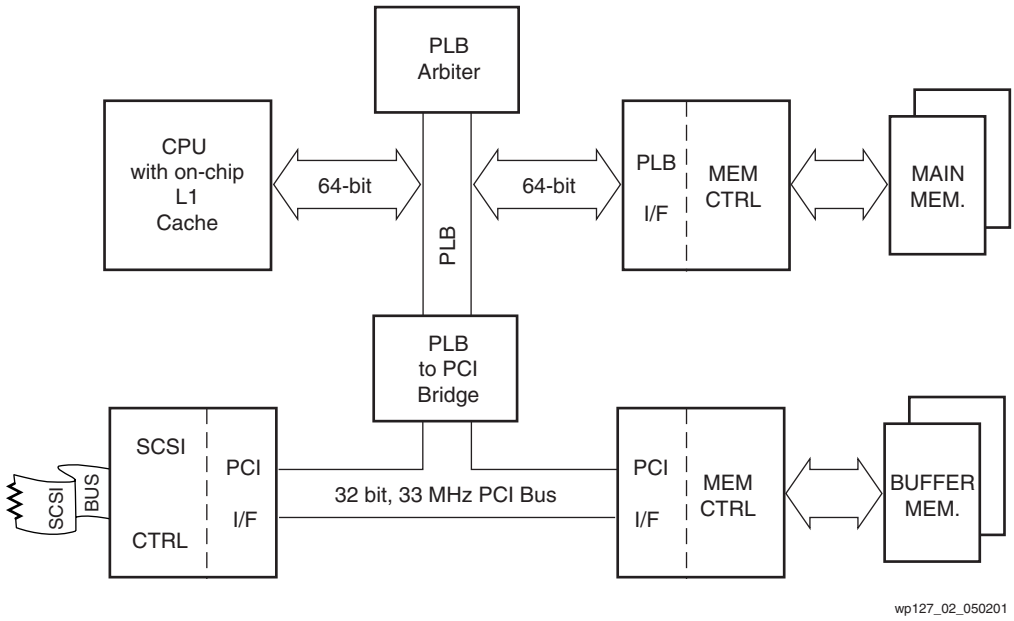


*Figure 2:* **A Small Embedded System**

## CPU Local Bus Example

### Trade-Off #4: Locating Peripheral Components on PLB or Secondary Bus

PLB performance is enhanced by increasing bus speed and width and by decreasing the number of bus agents (e.g., reducing bus loading). Speed issues have already been mentioned. Signal integrity problems tend to dictate maximum bus speed. At any particular speed, data transfer rates can be increased by widening the bus, assuming that the processor local bus interface allows it. The PLB is directly connected to the processor, so the processor interface sets the PLB width. The number of PLB agents is application dependent. On a highly loaded bus, careful analysis of PLB agents can suggest that some agents be moved to another bus. This secondary bus would be connected to the PLB using a bus-to-bus bridge. *N* loads are thus removed from the PLB and replaced by the single load of the added bridge. In this scenario, however, the new bridge would add a transfer delay in the path to the relocated agents.

As previously noted, the PLB frequency is constrained by high-speed bus routing issues. The chart in Figure 3 shows how the PLB clock rate changes in the example system as the CPU frequency is increased. As CPU clock rates are swept from 100 MHz through 350 MHz, the PLB clock rate initially matches. A downward step from the CPU clock rate is eventually required. Contemporary systems have PLB

speeds in the 100 MHz to 133 MHz range, and a 133 MHz (maximum) PLB rate is implicit here.

In Figure 3, the PLB clock rate approaches its maximum, but then steps down in frequency. Typically, the PLB is fixed at some reasonable ratio of the CPU clock. In this example, the chosen ratios step through 1.0, 0.75, 0.66, 0.50, and 0.33 as the CPU clock rate increases. This keeps the PLB clock running about as fast as possible for all CPU clock frequencies.

If a particular application frequently executes a relatively small routine entirely out of cache (e.g., number crunching or data manipulation), then the PLB is not often accessed and a high-performance processor would be appropriate. If on the other hand, the application is too large to fit into cache, then PLB performance becomes a critical component affecting system throughput. Recall that the CPU cannot transfer information faster than the PLB clock rate, regardless of its MIPS capability. Thus, a high performance processor might not be the best investment for such a system.

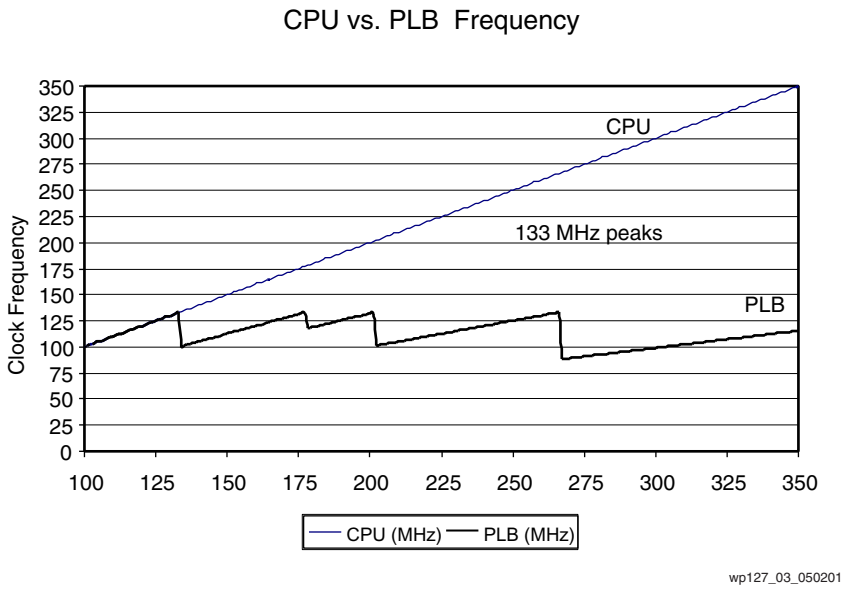


*Figure 3:* **CPU vs. PLB Clock Rates**

## Main Memory

The first main memory decision is using SRAM or DRAM. Synchronous SRAM is very fast and expensive. Typically, the memory requirements of the system are large enough that SRAM becomes a cost-prohibitive choice. Other SRAM characteristics that might contribute to its exclusion include power consumption and relatively low density (high PCB area consumption compared to DRAM). Thus, DRAM is the memory technology of choice for many embedded system designs.

## DRAM Main Memory

After the decision to use DRAM is made, the designer is faced with several DRAM options: SDR, DDR, or other more “exotic” technology. Synchronous DRAM (SDRAM) devices are available as single data rate (SDR) SDRAMs and double data rate (DDR) SDRAMs. In DDR SDRAMs, new data is available on both the rising and falling edge of the clock. SDR SDRAMs are available in the 143 MHz domain. Some DDR devices meet this clock rate and can transfer data at 286 MHz.

Both **XAPP134: *"Synthesizable High Performance SDRAM Controller"*** and **XAPP200: *"Synthesizable 1.6 Gbytes/s DDR SDRAM Controller"*** are application notes specifically addressing the latest DRAM memories.

## Memory Performance

DRAMS have a page-opening, row-address strobe (RAS) delay followed by a column-address strobe (CAS) latency, which lowers their effective data transfer rate. A memory controller adds latency as well.

### SDR SDRAMs

Contemporary SDR SDRAMs run at 143 MHz with a 3-clock latency to open a page for a read or write. Within the same opened page (e.g., the RAS delay has already occurred), there is an additional CAS latency of three. This means that burst data is available at 143 MHz, but the first data element of the burst is delayed by three clock periods (after receipt of a read or write command). For a 4-word burst within the same page, six clocks are required, lowering the average data rate from 143 megabits (Mb / s) (for a x8 device) to 95 Mb / s. If the page has to be changed, then an initial 3-clock page opening delay is added.

### DDR SDRAMs

To increase performance, the main memory bank can be changed from SDR to DDR SDRAM. These components also experience about a 3-clock latency to open a page for a read or write. A typical "8 Meg by 8" –7 grade device has a CAS latency of 2.5, at a frequency of 143 MHz. This DDR device transfers data on both edges of the 143 MHz clock, so that its burst transfer rate is 286 Mb / s. The up-front CAS latency of 2.5 lowers this. Within the same page, for a 4 word burst, nine half-clocks are required, lowering the average data rate from 286 Mb / s to 127 Mb / s. If the page has to be changed, then an initial 3-clock page opening delay is added.

The next section compares SDR and DDR SDRAM performance.

## Some SDRAM Performance Comparisons

To simplify the SDRAM performance comparison, some assumptions are made for the example system in Figure 2:

1. The memory is 64-bits wide.
2. A cache line is eight 32-bit words.
3. PLB arbitration response of one PLB clock for each CPU request.
4. The target page in memory is already open (page opening latencies are not included).
5. The memory controller interface is always running at maximum speed (e.g., a 143 MHz rated memory runs at maximum speed, regardless of the actual PLB speed).
6. The memory controller with SDR SDRAM has a latency of 63 ns to first data out on a read. This latency is computed as follows. After the read command is clocked in, it takes nine clocks at 143 MHz to read the four data elements into a FIFO in the memory controller. Two clocks are needed for command resynchronization (PLB interface to memory control interface), and one clock is needed to issue the command to the SDRAM. Three clocks are consumed by CAS latency (the page open "activate" command has already taken place), and three more 143 MHz

clocks are needed to complete the read of four data elements (9 × 6.99 ns = 62.91 ns).

7. The memory controller, with DDR SDRAM, has a latency of 53 ns to first data out on a read. This latency is computed as follows. After the read command is clocked in, it takes 7.5 clocks at 143 MHz to read the four data elements into a FIFO in the memory controller. Two clocks are needed for command resynchronization (PLB interface to memory control interface), and one clock is needed to issue the command to the SDRAM. 2.5 clocks are consumed by CAS latency (the page open "activate" command has already taken place), and two more 143 MHz clocks are needed to complete the read of four data elements (7.5 × 6.99 ns = 52.425 ns).

A cache miss causes four reads from main memory (64-bits wide). Thus, it takes one PLB clock for arbitration plus four more clocks to read a cache line.

The memory CAS latency at 143 MHz, from a typical memory data sheet, is three clocks for the SDR SDRAM and 2.5 clocks for the DDR SDRAM.

To simplify the analysis, the PLB operates in only SDR mode for both types of memory. This simplified memory controller does not indicate "ready" status until the read burst of four (into its internal FIFO) is complete. The CPU clock frequency is swept from 100 MHz to 350 MHz, and the resulting SDR and DDR cache line read times are shown in Figure 4.

To calculate the time for a cache line read of four data elements, the following three delay elements are summed:

1. The initial one PLB clock arbitration period.
2. The memory controller latency, converted into PLB clock periods by dividing the memory controller fixed latency by the PLB clock period and rounding up to the next whole clock period. This delay is from the second clock (which clocks the read command into the memory controller) to the first data element enabled out onto the PLB.
3. Three more PLB clocks needed to clock out the remaining three data elements.

Figure 4 shows the SDR and DDR read timings. As expected, the DDR read times are shorter.

The best read times for both types of memory occur when the PLB is running at its maximum speed of 133 MHz. Referring to Figure 3, recall that this happens when the CPU hits 133 MHz, 177 MHz, 201 MHz, and 266 MHz. Thus, a designer could examine these performance points to check where system performance is acceptable. A lower performance, less expensive processor might do the job.

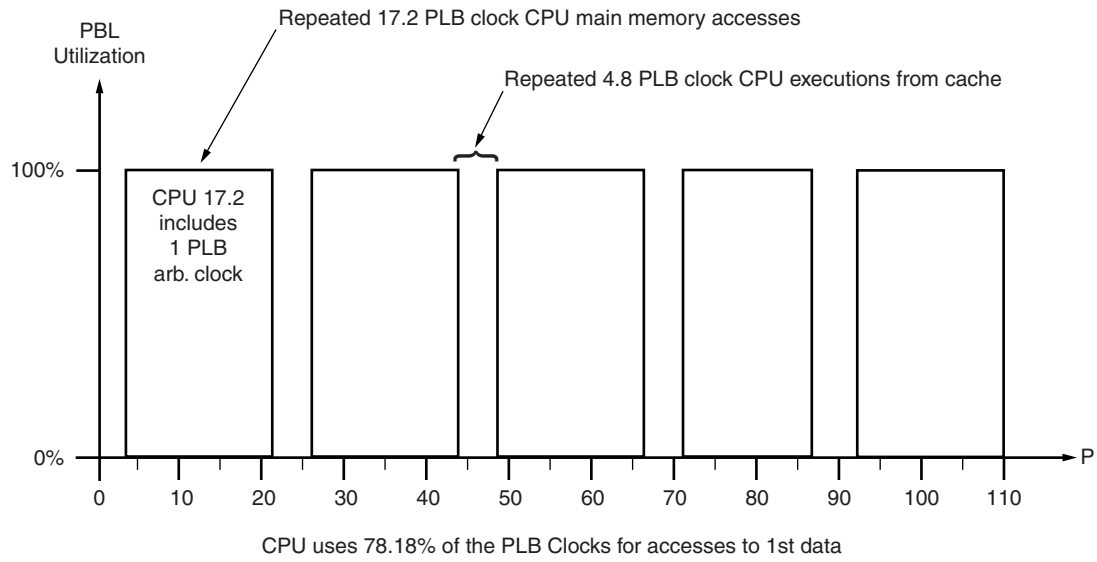


*Figure 4:* **SDR and DDR SDRAM Cache Line Read Comparison**

Cache line read time performance can be improved by designing a double data rate PLB. This would require the PLB interfaces on both the CPU and the memory controller to also work in DDR mode. In addition, a more sophisticated memory controller design would add to performance with a lower latency.

PLB performance can also be examined during larger data transfers. Figure 4 is based on a cache line read of four 64-bit double words (a cache line of eight 32-bit words). This is typical of many cache fill requirements during normal operation when a cache miss occurs. However, at cache initialization, the cache contents are usually invalid (full of power up "junk"). If the cache can demand a complete fill, the "burst" from memory is substantially longer.

In computing Figure 5, a 4-KB cache is assumed with an SDR SDRAM memory page size of 4 KB. SDR SDRAMs allow a page burst; however, current DDR memory data sheets reflect only burst lengths of 1, 2, 4, or 8 elements. Figure 5 shows an SDR SDRAM page continuous burst (512 PLB clocks, plus the memory controller latency), versus a DDR SDRAM operation (64 read bursts of eight data elements at a time). Both types of memory assume that the three-clock page-opening delay has taken place, and that all subsequent accesses are taking place in the same opened page.

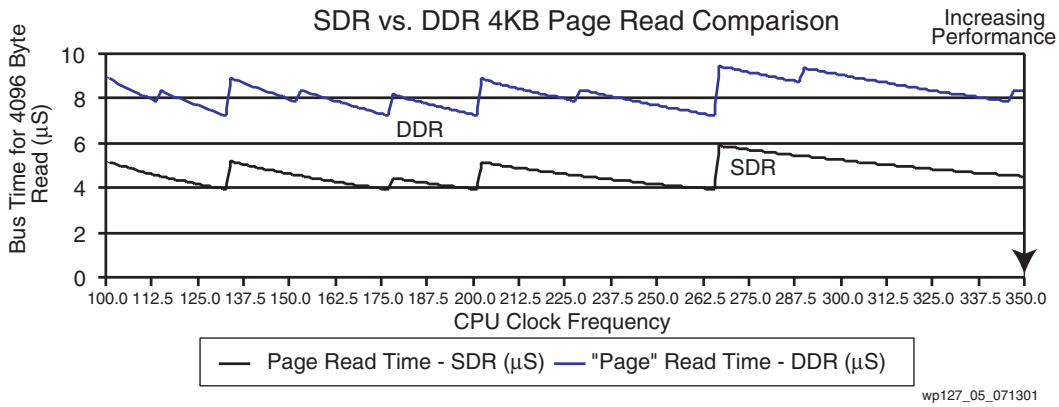


*Figure 5:* **SDR vs. DDR SDRAM 4 KB Page Read Comparison**

Figure 5 confirms that SDR SDRAMs outperform DDR SDRAMs in this long continuous burst scenario. For the DDR SDRAM, the arbitration clock cycle along with memory latency overhead has a dramatic effect when repeated many times. However, the long burst capability is only useful at power-on time, unless normal system operation requires long data transfers (for instance, to or from an I/O channel). Also, the PLB tenure protocol must allow long PLB tenures for this to be beneficial. With long-bus tenures, an SDR SDRAM could offer better performance.

## Effective Throughput

To investigate effective throughput, excluding I/O effects for now, some additional assumptions for system parameters must be added:

1. The CPU executes one instruction every 1.2 CPU clocks (i.e., 1.2 clocks per instruction).
2. CPU on-chip composite (data and instructions in same cache) cache average hit rate is 80%.
3. 30% of the data requires main memory accesses for loading/storing.
4. Cache content is 50% instructions and 50% data.
5. Main memory consists of DDR SDRAM.

Items 1 through 4 help approximate the instruction mix. In a real system design, rigorous analysis of application code flows would provide these numbers.

The chart in Figure 6 shows the relationships between the CPU clock (MHz), the PLB clock (MHz), and effective system throughput (MIPS). Notice how the MIPS curve mirrors the PLB curve. This reinforces the notion that system throughput is more strongly related to PLB performance than the raw MIPS capability of the CPU. Peak MIPS is reached at a CPU clock rate of 266 MHz.

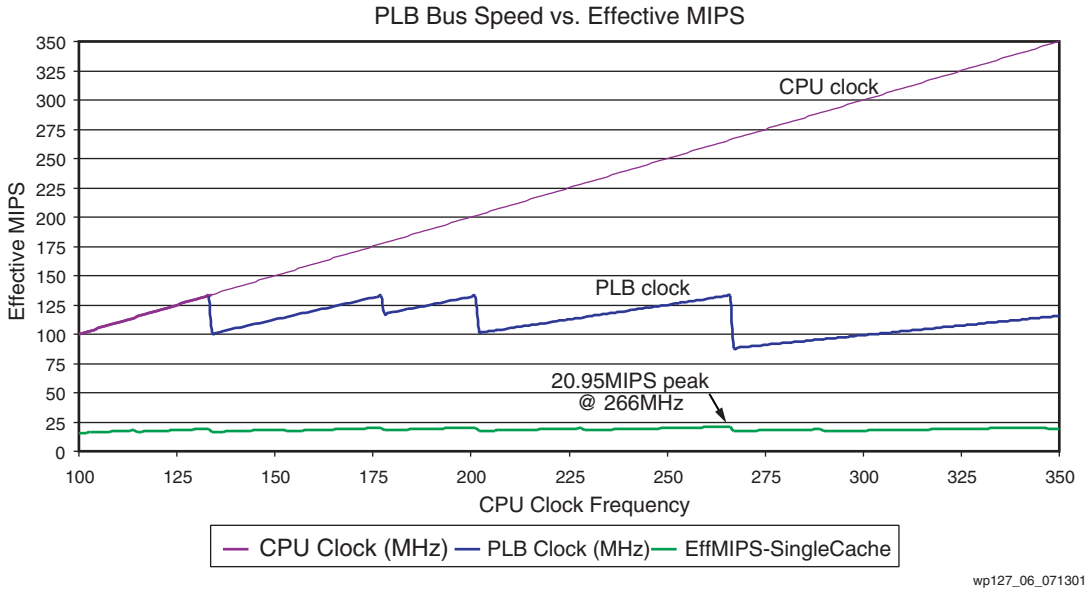


*Figure 6:* **PLB Bus Speed vs. Effective MIPS**

## CPU Bus Utilization

Some assumptions were made when developing the effective MIPS curve in Figure 6. The CPU executes one instruction every 1.2 CPU clocks. The CPU has a composite cache containing 50% instructions and 50% data. The CPU has an average hit rate of 80%, and 30% of the data must be loaded or stored. Processor local bus utilization can be estimated from the average cache hit rate (80%) and CPU clocks / instruction (1.2) numbers. From Jim Handy's *Cache Memory* book (See **References**), the formula is (paraphrased):

Throughput = (hit rate × cache latency) + (miss rate × average number of wait states on PLB)

In the example shown in Figure 2, the cache latency is zero. The formula can be expanded, according to Handy, into an equation revealing how all of the system clock cycles are consumed. For ten instructions (inst.), the example system would yield:

Clock cycles consumed = [10 inst. × (80% hit rate × 1.2 clock cycles per inst.)]
+ [10 inst. × (20% miss rate × (1.2 clocks + 16 clocks for memory access))]

The first half of the formula computes the CPU clock cycles consumed by cache hits, and the second half, the CPU clock cycles consumed by cache misses. In this discussion, a CPU clock rate of 266 MHz was chosen. From the chart in Figure 6, the PLB frequency is 133 MHz (i.e., 2:1 ratio). The 16 CPU clock memory access on a cache miss comes from Figure 4, where at 266 MHz a DDR SDRAM gives a cache line read response time, to the first 64-bit data element, of eight 133 MHz PLB clocks.

Recall that eight PLB clocks are equivalent to 16 CPU clocks in this 2:1 ratio scenario. This formula resolves to: [10(.8(1.2))] + [10(.2(1.2 + 16))] = [10(.96)] + [10(3.44)] = 9.6 + 34.4 = 44 CPU clock cycles for ten instructions.

Fractional values are valid since this is an average between wait state and zero-wait cycles.

Only 9.6 cycles execute from cache! In other words, 34.4 / 44, or 78.18% of the time, the CPU is attempting to get through 20% of the instructions. The PLB is in use by the CPU 78.18% of the time.

For this particular system, what happens to bus utilization if the cache hit rate is improved? For a 95% hit rate, the above formula yields 8.6 / 20, or 43% bus utilization, a significant improvement.

Figure 7 shows PLB utilization based upon the 80% cache hit rate / 1.2 CPU clocks per instruction scenario described in the equations.

Following the results of the equation, the CPU executes out of cache for 9.6 clocks (4.8 PLB clocks at the 2:1 ratio assumed). A cache miss is then experienced, requiring the CPU to make a main memory access to load a cache line. The CPU takes 17.2 PLB clocks to read the first data element from the memory controller. The CPU is assumed to have cache bypass capability, which allows it to continue execution while the other three cache line data elements are being read. Thus, the additional three PLB clocks are ignored in this computation. This hit-miss sequence is repeated regularly and shown in Figure 7.

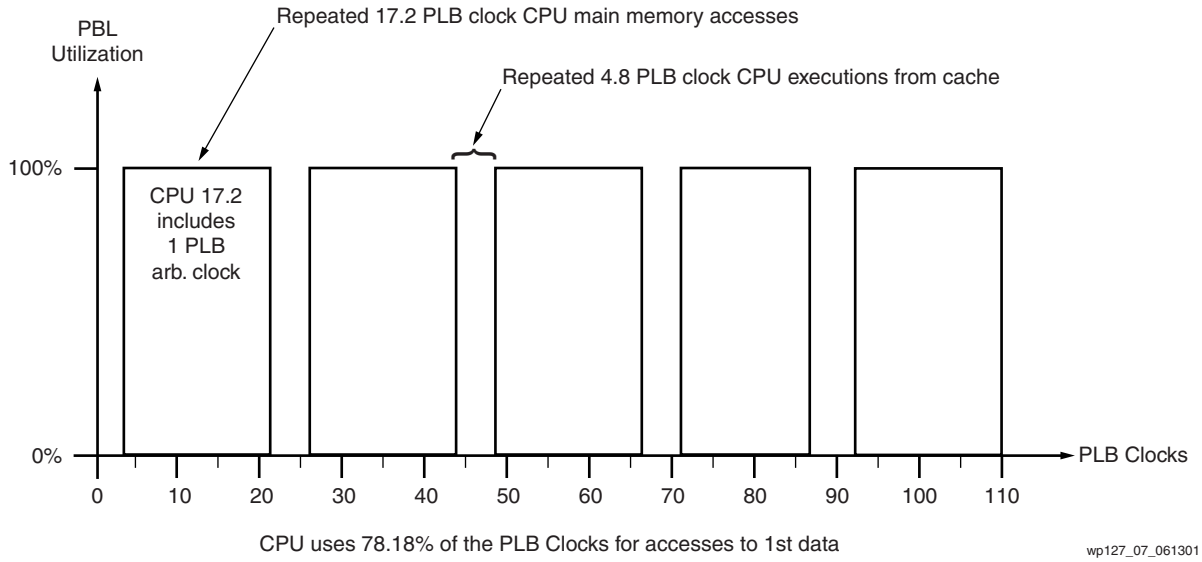


*Figure 7:* **CPU Utilization of the PLB**

**Notes:**

1. In Figure 7, the CPU repeats the cache access / PLB access sequence five times within 110 PLB clock cycles. The CPU uses (17.2 × 5) / 110 = 86 / 110 = 78.18% of the PLB clocks. As noted elsewhere, a higher cache hit rate would significantly reduce CPU utilization of the PLB.

## I/O Subsystem Performance

The example system in Figure 2, shows an expansion bridge interface between the PLB and an I/O subsystem. Up to this point, the performance analysis has ignored subsystem interaction effects.

### Effects of Buffer Memory Location

The example system shows a PCI bus slave memory connected to a SCSI bus device controller. This system is focused on data movement without CPU intervention. This arrangement allows the I/O controller to transfer data to and from the PCI slave buffer memory without requiring PLB bus cycles. The CPU retains unhampered access to the main memory.

A system focused on data manipulation might make the buffer memory part of the main memory on the PLB. All I/O accesses to buffer memory then compete with CPU main memory cache line reads. The PLB bandwidth available to the CPU decreases and the CPU performance suffers. However, a composite memory configuration would be less expensive because the PCI slave memory controller and the associated memory devices would be eliminated.

### I/O Performance Bottleneck Solutions

If the CPU is idling while waiting for I/O transfers to complete, the system is I/O bound. Some ways to speed up I/O transfers would be:

- Bridge chip with a very large FIFO
- Use wider or faster data buses
- Use faster I/O protocol (Fibre Channel vs. SCSI, etc.)

If the I/O is waiting on the CPU while it is busy executing code, then some solutions might be to:

- Increase the CPU clock rate
- Change to a different CPU
- Optimize the code by reducing the number of instructions

### Data Movement in Figure 2

The I/O subsystem shown in Figure 2 is a SCSI device controller on a 33-MHz, 32-bit PCI bus.

The I/O controller is a smart device that reads command control blocks placed into the slave buffer memory by the CPU. The I/O controller then knows how much data to read or write, and its source or destination address in the buffer memory. The I/O controller also posts operational status information into the buffer memory for the CPU to read.

The transfer of control blocks and status can be done by the CPU one word at a time, or the CPU can set the bridge chip up to execute a block DMA. The DMA operation would proceed without further CPU intervention.

Consider the case where the SCSI I/O controller is moving a frame of SCSI data across the PCI bus into the slave buffer memory. A 2048 byte frame would take 15.36 µs on this PCI bus. If the CPU is simultaneously trying to write control block data into the buffer memory, it will find that the bridge chip cannot access the PCI bus. The command would be queued by the bridge for execution after the PCI bus was again idle. The CPU itself might become stalled if it needs to have this control block processed before it can continue operation.

The PCI-to-PLB bridge has a latency associated with it. Commands posted from either direction by the CPU or SCSI controller must await translation to the protocol of the opposing bus. The bridge then requests access to the target bus and must await either bus arbiter or slave device response.

### CPU and I/O Utilization of the PLB

Consider the transfer of a 48K byte file from the PCI slave memory to the PLB main memory. The bridge is set up by the CPU with appropriate DMA parameters. The bridge will then proceed to execute the DMA as quickly as it can. The effect of this I/O transfer is reflected in Figure 8.

Figure 8 shows PLB utilization based upon the 80% cache hit rate/1.2 CPU clocks per instruction scenario previously depicted in Figure 7, combined with the additional PLB clock cycles consumed by the I/O transfers.

The following parameters are additionally assumed:

1. A bus request/grant cycle takes one PLB clock.
2. The PCI-to-PLB bridge has an internal FIFO capable of holding eight 64-bit data elements (8 x 8 bytes = 64 bytes).
3. The PLB protocol does not allow any bus master to hog the PLB for more than eight data transfers per tenure.
4. A bus tenure, once started, will not be interrupted if it is less than nine data transfers long.

The transfer of the 48K byte file takes (49152 bytes/64 bytes) = 768 bursts of eight data transfers of 8 bytes each on the PLB.

The PCI bus runs at 33 MHz and is 32-bits wide. Therefore, it takes 16 PCI clocks (16 x 30nS = 480nS) to fill up the bridge FIFO (16 X 4 bytes wide = 64 bytes). The

bridge is programmed to request access to the PLB only when its FIFO is full, allowing the bridge to burst eight 4-byte transfers onto the PLB at the PLB clock rate.

Figure 8 shows the bridge gaining access to the PLB for its data burst, forcing the CPU to wait periodically.

The bridge burst of eight 4-byte wide transfers takes 9 PLB clocks, one clock for the GRANT cycle and eight clocks for the data transfer.

Figure 8 shows how the I/O transfers impede CPU access to the PLB for cache line reads from PLB memory. The CPU loses 14.42% of the previous PLB clock cycles. This is computed as follows:

Figure 7: (17.2 x 5 repetitions) / 110 = 86 / 110 = 78.18%

Figure 8: (17.2 x 4.28 repetitions) / 110 = 73.6 / 110 = 66.91%

The CPU lost (86 - 73.6) = 12.4 PLB clocks, and 12.4 / 86 = 14.42%

Figure 8 shows that the bridge has been granted the PLB when the CPU wants to execute PLB memory reads. Thus, the performance of the CPU is affected by the I/O subsystem in this scenario.

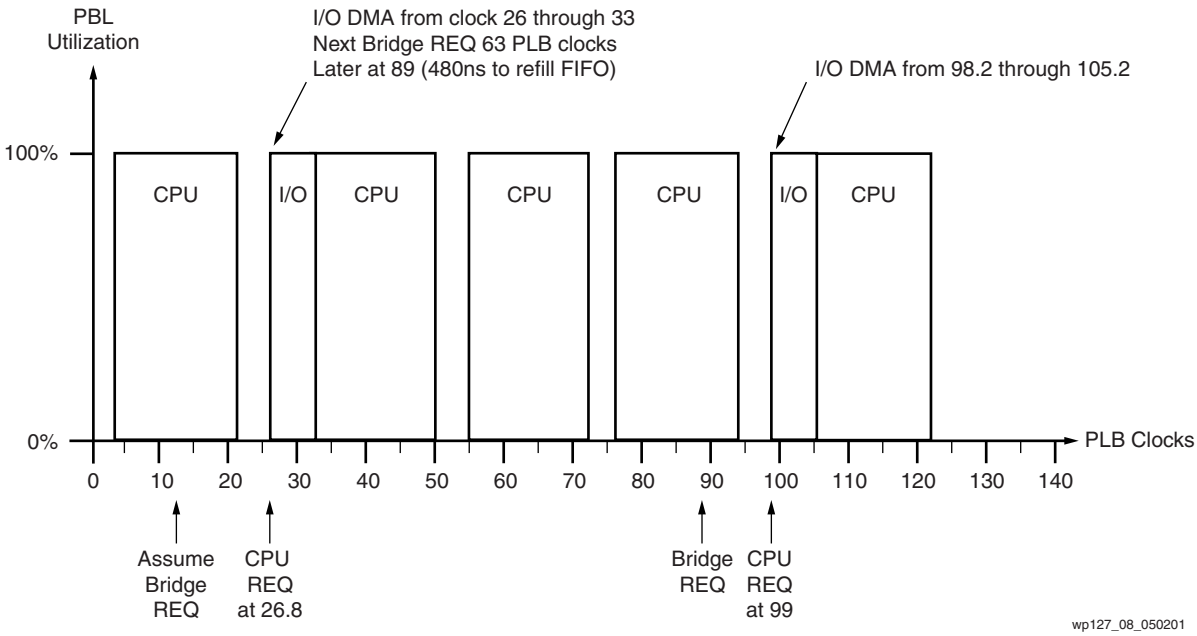


*Figure 8:* **CPU and I/O Utilization of the PLB**

## Principle of Overall Performance

This section summarizes the performance of the example system under four scenarios, see Figure 9. The CPU and PLB clock rates of 266 MHz and 133 MHz, respectively, remain constant throughout:

Case 1: The CPU is executing solely out of cache at one CPU clock per instruction.

Case 2: The CPU is executing solely out of cache at 1.2 CPU clocks per instruction.

Case 3: Figure 6 peak effective MIPS (using the five assumptions for Figure 6).

Case 4: Figure 8 peak effective MIPS (Figure 6 parameters with I/O mixed in).

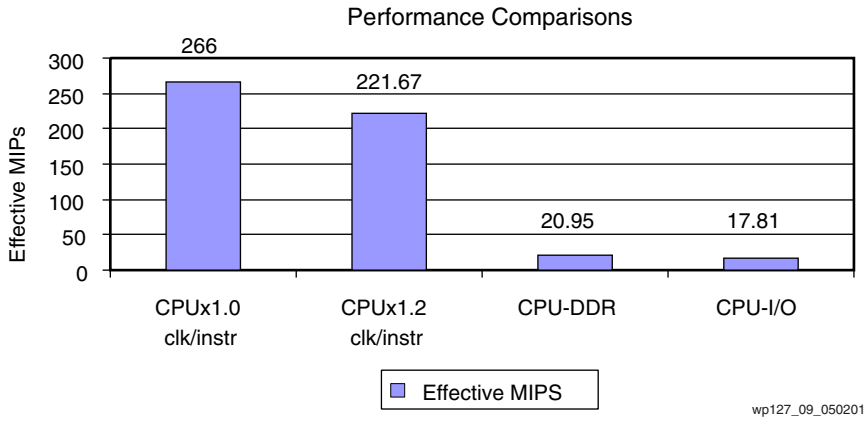


*Figure 9:* **Principle of Overall Performance**

## Conclusion

System performance is very application dependent. In this white paper, an example embedded system was analyzed, showing system performance peaks at several places as the CPU clock rate was swept through a frequency range. The characteristics of the PLB and the memory subsystem cause non-obvious performance peaks at various CPU speeds, as shown in Figure 6. These results support the premise that the full MIPS capability of a very fast and expensive processor will not always be used. Matching of processor and component performance generally provides a good cost-performance trade-off.

## References

Additional information can be found in the following documents:

- Xilinx **XAPP134**: "Synthesizable High Performance SDRAM Controller"
- Xilinx **XAPP200**: "Synthesizable 1.6 Gbytes/s DDR SDRAM Controller"
- The Cache Memory Book (Second Edition), Jim Handy, Academic Press, 1998 ISBN 0123229804

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 03/06/02 | v1.0 | Xilinx Initial Release |