AMD Alchemy™ Solutions Au1100™ Processor LCD Performance

Application Note

Revision: **30274A**
Issue Date: **April 2003**

Contacts

www.amd.com   pcs.support@amd.com

1.0 Introduction

This document describes the performance characteristics of an Au1100™ processor based design using the integrated LCD controller.

This document assumes the reader is familiar with LCD technology and the *AMD Alchemy™ Solutions Au1100™ Processor Data Book* (see 7.0 “References”).

This document also assumes the reader is familiar with the applications note “Au1x00 SDRAM Performance” which outlines SDRAM performance for a typical system (see 7.0 “References”). The remainder of this document uses numbers for a 396MHz system with a 99MHz SDRAM interface, as outlined in the SDRAM applications note.

2.0 LCD Controller Overview

The Au1100 processor features an integrated LCD controller for connecting to liquid crystal displays and cathode ray tubes. The LCD controller supports the common industry standard TFT and STN panel technologies and is able to drive cathode ray tubes via an external digital-to-analog converter (DAC). In the discussion to follow, the term display is a reference to either a TFT or a cathode ray tube with an appropriate DAC. The majority of the information presented in this document is applicable for an STN panel, but the calculations differ.

The Au1100 processor databook contains details and additional information on the operation of the LCD controller. The general arrangement of the Au1100 processor LCD controller is depicted below.

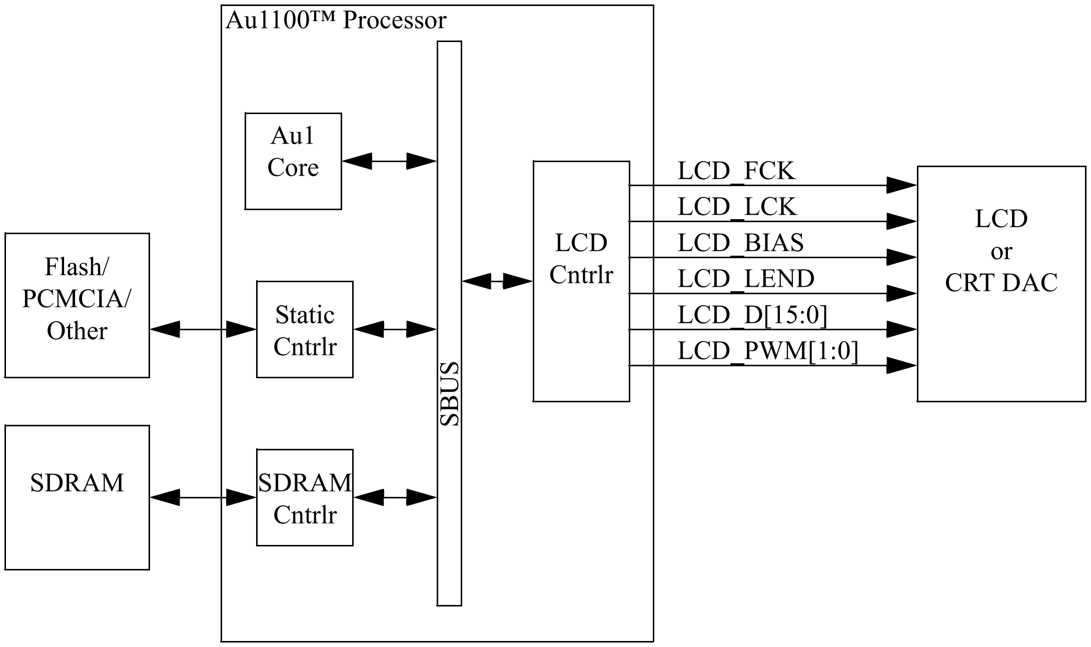


Figure 1: Au1100™ Processor LCD controller

The performance of any input/output peripheral is usually described in terms of the maximum amount of data that can be moved through the interface in a given time period. For example, a 100Mbps Ethernet controller can move a maximum of 12.5MB/s. If the actual performance is less than the maximum, data movement occurs at a slower pace.

In the case of the output-only LCD controller, the performance is essentially a constant. Unlike many peripheral I/Os, if the LCD controller fails to satisfy the constant performance requirement, the display refresh fails, resulting in visual artifacts (and not just slower data movement).

The performance constant for the LCD controller is easily calculated for a given display type. However, the LCD controller is only one aspect of performance in an Au1100 processor based design. The remainder of this document identifies influences on the system performance of a design using the Au1100 processor LCD.

3.0 Unified Memory Architecture Fundamentals

Figure 1: "Au1100™ Processor LCD controller" depicts a unified memory architecture (UMA) arrangement where the memory used by the LCD controller for the framebuffer is shared with the rest of the system. In this arrangement, the Au1 core performs all drawing in the framebuffer, which resides in SDRAM, and the LCD controller continuously refreshes the display by fetching the framebuffer contents and sending the pixel data to the display.

In a non-unified memory architecture, the LCD (or graphics) controller has a dedicated memory pool that contains the framebuffer. Furthermore, the LCD (or graphics) controller has priority over processor-initiated accesses to the framebuffer memory in order to maintain the refresh of the display.

By eliminating the need for a dedicated framebuffer memory pool, a UMA is a more cost-effective graphics solution than a non-unified memory architecture environment. However, since the Au1 core, LCD controller and other peripherals share the SDRAM, memory latency and bandwidth can affect system performance.

3.1 System Bus (SBUS)

The system bus (SBUS) is the main bus within the Au1100 processor. As such, access to the system bus is necessary in order to access the SDRAM, the Static Bus, or the integrated peripherals.

The SBUS typically operates at one-half the Au1 core frequency, and the SDRAM controller operates at one-half the frequency of the SBUS.

The Au1100 processor SBUS has four bus master slots for handling six system bus masters:

- Au1 core
- Ethernet MAC controller and DMA controller
- USB Host controller and IrDA controller
- LCD controller

The arbitration scheme for the system bus is round-robin; each bus master slot has equal opportunity to obtain access to the system bus. For a particular system bus master X, if no other system bus masters request the bus, then bus master X immediately wins the system bus. By contrast, if all other system bus masters request the bus, then bus master X must wait for three other system bus master slots' transfers before it wins the system bus, as depicted in the following figure.

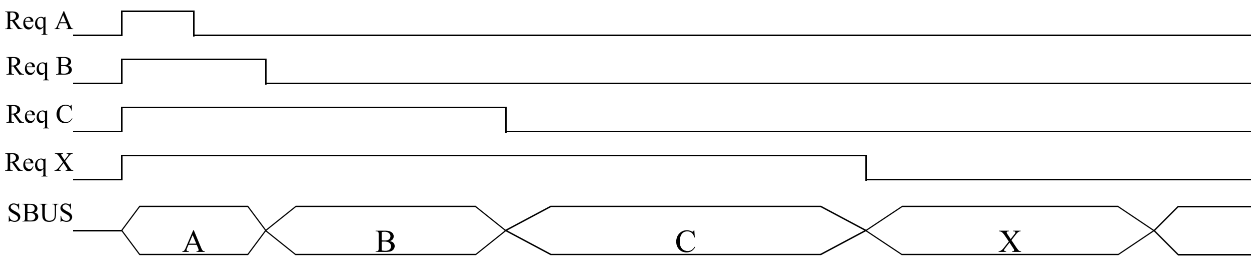


Figure 2: System Bus Arbitration

When a system bus master wins arbitration of the system bus, it performs transfers to/from the integrated peripherals, SDRAM, or the Static bus.

3.2 Latency and Bandwidth

Latency is defined as the amount of time between when a request for a resource is initiated and when the request for that resource is granted. In the scope of this discussion, latency is the time between when a system bus master (e.g. LCD controller) requests access to the system bus (e.g. in order to access framebuffer memory) and when the system bus is granted to that master. Bandwidth is the amount of data that can be moved across the system bus in a time interval. In the Au1100, latency and bandwidth are inversely related such that an increase in latency results in a decrease in bandwidth (since less time is available to move data), and vice versa.

Two factors influence latency and bandwidth: system bus arbitration, and transfer time.

As stated previously, access to SDRAM requires access to the system bus. For all practical purposes, the latency onto the system bus is the latency to the SDRAM. Figure 3: "System Bus Latency for a Bus Master" illustrates the round-robin arbitration scheme with all system bus masters requesting the bus simultaneously, and the corresponding effect on system bus latency for bus master X.

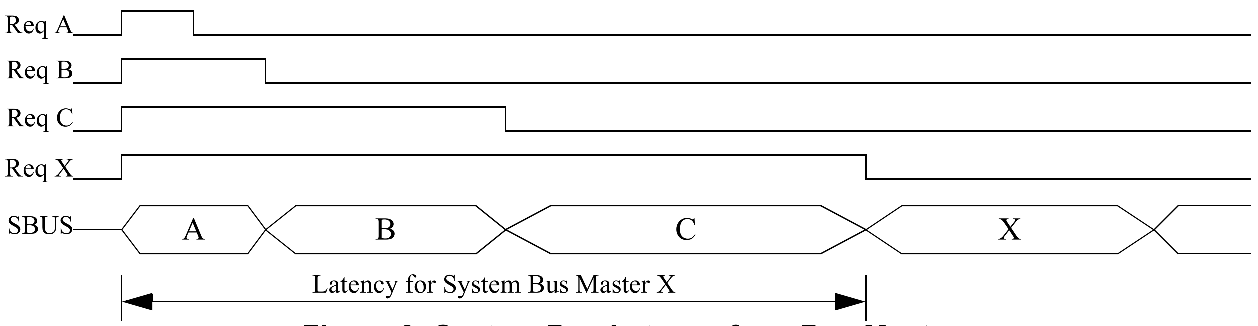


Figure 3: System Bus Latency for a Bus Master

The illustration also demonstrates the impact of transfer time on latency, and in turn bandwidth. While the transfer time to integrated peripheral registers is negligible, the SDRAM, and Static Bus transfer times can add appreciable delay to system bus latency.

Note that from the perspective of system bus master X, increases in system bus latency result in fewer opportunities for system bus master X to perform transfers to/from SDRAM in a given time interval. Thus an increase in system bus latency results in a decrease in effective SDRAM bandwidth for system bus master X (the actual SDRAM bandwidth potential is unchanged, as outlined in the SDRAM applications note).

3.2.1 SDRAM Interface

For a 396MHz Au1100 processor operating the SDRAM controller at 99MHz, an SDRAM single-beat access is 60ns (6 cycles at 10.1ns), and an SDRAM burst access is 121ns (12 cycles at 10.1ns). Accesses to SDRAM can add upwards to 121ns to the system bus latency for other system bus masters.

A typical SDRAM configuration is capable of approximately 248.9MB/s throughput. The SDRAM bandwidth is important since it is the main storage for applications, data and the LCD framebuffer. There must be enough SDRAM bandwidth to satisfy the LCD controller refresh demand as well as run the applications.

The SDRAM bandwidth needed by the LCD controller is a product of the display resolution size, pixel depth and refresh rate. The following table lists some common resolutions and the resulting SDRAM bandwidth requirement.

Table 1: LCD Controller SDRAM Bandwidth

	Horizontal (Pixels)	Vertical (Pixels)	Depth (Bits Per Pixel)	Refresh Rate (Hz)	Bandwidth (MB/s)
QVGA	320	240	8	60	4.6MB/s
QVGA	320	240	16	60	9.2MB/s
VGA	640	480	16	60	36.8MB/s
XGA	800	600	16	60	57.6MB/s
XGA	800	600	16	72	69.1MB/s

The above values represent the SDRAM bandwidth demand of the LCD controller as it continuously refreshes the display. With a total SDRAM bandwidth of 248.9MB/s, the LCD controller consumes a relatively small percentage, leaving ample bandwidth for the Au1 core to run applications and perform graphics operations.

The LCD controller timing values should be configured so as to minimize the SDRAM bandwidth demand. In particular, the refresh rate should be set to the lowest rate permitted by the display.

3.2.2 Static Bus Interface

The Static Bus permits a wide variety of external devices to connect to the Au1100 processor. The transfer time for these peripherals is in the tens and hundreds of nanoseconds. Flash memories typically range from 90ns to 120ns, and PCMCIA cards typically are 150ns, 200ns or 250ns.

Furthermore, the Static Bus features the EWAIT# signal, and PWAIT# signal for PCMCIA, which can be asserted by external devices to insert an arbitrary number of wait states into a transfer. The assertion of these signals further increases latency for other system bus masters.

The full impact of static bus peripherals that assert EWAIT# or PWAIT# is discussed after outlining the latency requirements of the LCD controller.

4.0 Latency and Bandwidth with Respect to the LCD Controller

To refresh the display, the LCD controller must fetch all the pixels of a frame, and do so at the refresh rate of the display. To fetch a frame, the LCD controller generates a series of burst accesses to SDRAM. Since a single SDRAM burst fetches only 32 bytes, multiple SDRAM accesses are needed to fetch an entire frame.

The LCD controller implements two 320-word buffers for moving data from SDRAM to the pixel engine. The two buffers are ping-pong buffers: the pixel engine pulls data from one buffer while the other buffer is filled from SDRAM. If the pixel engine consumes a buffer, and the next buffer is not yet filled, the pixel engine incurs an under-flow and repeats the last pixel, resulting in display artifacts. The time to empty a 320-word buffer determines the maximum time allowed to fill a 320-word buffer in order to avoid the buffer under-flow condition.

The pixel engine pulls one pixel from the buffer every pixel clock while rasterizing (for the sake of simplicity, the horizontal non-display times are ignored). Thus, the pixel clock period multiplied by the size of the buffer and divided by the number of pixels in the buffer yields the buffer empty/fill time for a given display configuration.

The LCD pixel clock is derived from values programmed into sys_clksrc, sys_freqctrl and lcd_clkcontrol. Table 2 provides example pixel clock settings for common display types.

Table 2: LCD Pixel Clock Timing

	Horizontal (Pixels)	Vertical (Pixels)	FREQn	lcd_clkcontrol[PCD]	Pixel Clock
QVGA	320	240	48MHz	1	12MHz (83.3ns)
VGA	640	480	96MHz	1	24MHz (41.6ns)
XGA	800	600	96MHz	0	48MHz (20.8ns)

The number of pixels the buffer contains is determined by the lcd_control[BPP] field. Table 3 summarizes the possible combinations:

Table 3: LCD Buffer Pixels

lcd_control[BPP]	Bits Per Pixel	Number of Pixels Per Buffer
000	1	10240
001	2	5120
010	4	2560
011	8	1280
100	12	640
101	16	640

The time needed to empty a 320-word buffer is simply the product of the pixel clock period and the number of pixels contained in the buffer. Table 4 summarizes the buffer empty time for the example pixel clocks.

Table 4: LCD 320-Word Buffer Empty Time

	Horizontal (Pixels)	Vertical (Pixels)	Bits Per Pixel	Pixel Clock (ns)	Pixels Per Buffer	Buffer Time (ns)
QVGA	320	240	1	83.3	10240	852,992
	320	240	2	83.3	5120	426,496
	320	240	4	83.3	2560	213,248
	320	240	8	83.3	1280	106,624
	320	240	12	83.3	640	53,312
	320	240	16	83.3	640	53,312
VGA	640	480	1	41.6	10240	425,984
	640	480	2	41.6	5120	212,992
	640	480	4	41.6	2560	106,496
	640	480	8	41.6	1280	53,248
	640	480	12	41.6	640	26,624
	640	480	16	41.6	640	26,624

Table 4: LCD 320-Word Buffer Empty Time

	Horizontal (Pixels)	Vertical (Pixels)	Bits Per Pixel	Pixel Clock (ns)	Pixels Per Buffer	Buffer Time (ns)
XGA	800	600	1	20.8	10240	212,992
	800	600	2	20.8	5120	106,496
	800	600	4	20.8	2560	53,248
	800	600	8	20.8	1280	26,624
	800	600	12	20.8	640	13,312
	800	600	16	20.8	640	13,312

To avoid the buffer under-flow condition, the time needed to fill the other 320-word buffer must not exceed the time to empty a 320-word buffer. A 320-word buffer permits tens, hundreds, or even thousands of microseconds of time in which to fill the next buffer.

To fill a 320-word buffer requires 40 SDRAM 8-word bursts, or approximately 4,840ns (121ns * 40 bursts); significantly less than the 320-word buffer empty time. The design and capability of the Au1100 processor LCD controller permits ample time to fetch LCD buffers as well as perform other useful work in the system.

4.1 How Latency and Bandwidth Affect the LCD Controller

The two main points of the preceding discussion are that the 320-word ping-pong buffers permit adequate time to retrieve framebuffer contents from SDRAM as well as establish an upper-bound for avoiding display artifacts.

This section examines the conditions that can cause the 320-word buffer fill time to exceed the empty time. The 320-word buffer fill time in effect creates a hard real-time SDRAM bandwidth demand of 40 bursts in one buffer empty/fill time. Failure to complete 40 SDRAM burst in this time interval causes the LCD pixel engine to under-flow and repeat pixels. It is during this time period that efficient accesses to SDRAM is extremely important.

Consider the situation where the Au1 core is transferring a block of data to/from a PCMCIA card (e.g. network or storage card). Only the Au1 core and the LCD controller are actively requesting the system bus. The system bus arbitration scheme results in the Au1 core and LCD controller alternating transfers on the system bus. Thus for each LCD controller access, there is an Au1 core access to PCMCIA. Table 5 summarizes the time required to fill a 320-word buffer when both the Au1 core and LCD controller are using the system bus.

Table 5: PCMCIA and LCD Transfer Times

PCMCIA Transfer Time	40 PCMCIA Accesses	40 SDRAM Accesses	PCMCIA +LCD Time
150ns	6,000ns	4,840ns	10,840ns
200ns	8,000ns	4,840ns	12,840ns
250ns	10,000ns	4,840ns	14,840ns
300ns (PWAIT# asserted)	12,000ns	4,840ns	16,840ns
350ns (PWAIT# asserted)	14,000ns	4,840ns	18,840ns
400ns (PWAIT# asserted)	16,000ns	4,840ns	20,840ns
500ns (PWAIT# asserted)	20,000ns	4,840ns	24,840ns
600ns (PWAIT# asserted)	24,000ns	4,840ns	28,840ns

By comparing the time to fill a buffer from this table with that of the time to empty a buffer in Table 4: “LCD 320-Word Buffer Empty Time”, it is apparent that a number of display configurations, especially the 12bpp and 16bpp configurations, are susceptible to display artifacts when accessing slow PCMCIA cards. For example, the 640x480x16bpp display refresh fails if PCMCIA card accesses consistently need 600ns (buffer fill time of 28,840ns exceeds buffer empty time of 26,624ns).

Also note that this example does not take into consideration peripherals other than the Au1 core and LCD controller which may request the system bus. System bus requests by other peripherals simply add more time to the actual time needed to fill a 320-word buffer.

In addition, AMD has observed some PCMCIA cards assert PWAIT# to extend the transfer time to 1,000ns (1 microsecond), and even longer. If the Au1100 processor based product permits using PCMCIA cards with this type of transfer time, the ability to fill the 320-word buffer in the allotted time is extremely difficult, and will result in display artifacts.

The number of 320-word buffer fills needed per refresh for common configurations is provided in Table 6.

Table 6: LCD 320-Word Buffer Fills Per Refresh

	Horizontal (Pixels)	Vertical (Pixels)	Framebuffer Size (Pixels)	Bits Per Pixel	Pixels Per Buffer	Buffer Fills Per Refresh
QVGA	320	240	76,800	1	10240	7.5
	320	240	76,800	2	5120	15
	320	240	76,800	4	2560	30
	320	240	76,800	8	1280	60
	320	240	76,800	12	640	120
	320	240	76,800	16	640	120
VGA	640	480	307,200	1	10240	30
	640	480	307,200	2	5120	60
	640	480	307,200	4	2560	120
	640	480	307,200	8	1280	240
	640	480	307,200	12	640	480
	640	480	307,200	16	640	480
XGA	800	600	480,000	1	10240	46.8
	800	600	480,000	2	5120	93.7
	800	600	480,000	4	2560	197.5
	800	600	480,000	8	1280	375
	800	600	480,000	12	640	750
	800	600	480,000	16	640	750

The number of 320-word buffer fills per refresh multiplied by the display refresh rate determines the number of opportunities per second for buffer under-flows to occur. If a 320-word buffer under-flow does occur, the display artifacts last only until the start of the next refresh.

The LCD controller under-flow problem is the direct result of long latency, and not a bandwidth short-coming. The SDRAM has adequate bandwidth to supply the LCD controller; however, the ability of the LCD controller to access the SDRAM in an efficient manner is impacted by the system bus latency introduced by competing accesses to the static bus.

4.1.1 LCD Controller lcd_control[22:21] Setting

As previously noted, an increase in system bus latency results in a decrease of effective SDRAM bandwidth for the LCD controller. To combat the effects of long latency, the Au1100 processor LCD controller implements a feature that determines how many SDRAM burst accesses it should perform per system bus arbitration. By increasing the number of SDRAM bursts per LCD controller access, the LCD controller effectively increases its bandwidth to the SDRAM and consequently increases the likelihood of the LCD controller filling its 320-word buffers in time, even with the occurrence of long latency static bus accesses.

The number of SDRAM bursts per system bus arbitration is selected by lcd_control[22:21].

Table 7: lcd_control[22:21] Settings

lcd_control[22:21]	Number of SDRAM Bursts
00	1
01	2
10	3
11	4

By setting lcd_control[22:21]=11, the LCD controller performs 40 SDRAM bursts in 10 system bus arbitrations. Table 8 expands upon the previous example of the LCD controller alternating system bus transfers with the Au1 core, presenting the change in actual transfer time for a 320-word buffer fill.

Table 8: PCMCIA and LCD Transfer Times with lcd_control[22:21]=11b

PCMCIA Transfer Time	10 PCMCIA Accesses	40 SDRAM Accesses	PCMCIA +LCD Time
150ns	1,500ns	4,840ns	5,340ns
200ns	2,000ns	4,840ns	6,840ns
250ns	2,500ns	4,840ns	7,340ns
300ns (PWAIT# asserted)	3,000ns	4,840ns	7,840ns
350ns (PWAIT# asserted)	3,500ns	4,840ns	8,340ns
400ns (PWAIT# asserted)	4,000ns	4,840ns	8,840ns
500ns (PWAIT# asserted)	5,000ns	4,840ns	9,840ns

Table 8: PCMCIA and LCD Transfer Times with lcd_control[22:21]=11b

PCMCIA Transfer Time	10 PCMCIA Accesses	40 SDRAM Accesses	PCMCIA +LCD Time
600ns (PWAIT# asserted)	6,000ns	4,840ns	10,840ns

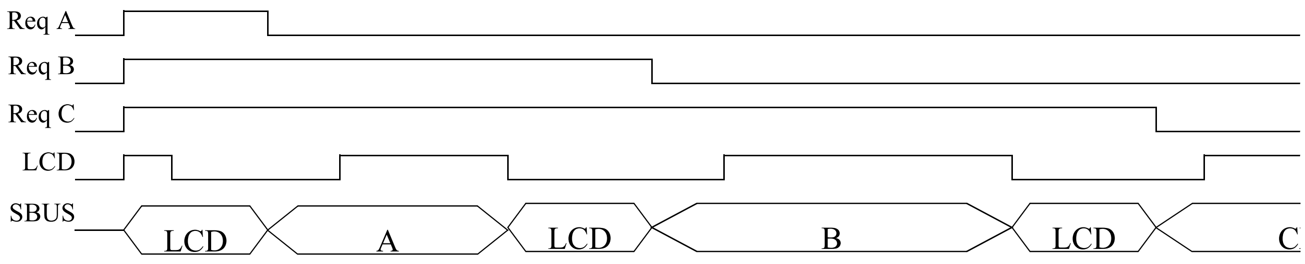
The lcd_control[22:21]=11b (4 SDRAM burst per arbitration) significantly increases the chances of the LCD controller filling a 320-word buffer in the allotted time.

While this table indicates that it is possible to avoid under-flow in all situations, keep in mind that this does not include system bus accesses by other masters, or PCMCIA (or static bus) transfers with transfer times greater than 600ns. The presence of more system bus requestors or longer PCMCIA transfer times increases the likelihood of a buffer under-flow, and the undesirable display artifacts.

4.1.2 LCD Controller sys_powerctrl[17] Setting

To further combat the effects of system bus latency, the Au1100 processor (stepping BE and newer) features a setting in sys_powerctrl[17] to change the system bus arbitration scheme in favor of the LCD controller. Setting sys_powerctrl[17] to 1 gives the LCD controller priority over other system bus requestors.

Figure 4: System Bus Arbitration with sys_powerctrl[17]=1



The change in the arbitration scheme permits shorter system bus latency for the LCD controller, and therefore more opportunities onto the system bus which in turn increases the likelihood of filling the 320-word buffer on time.

Note that this setting does not allow the LCD controller unconditional access to the system bus. The LCD controller must still wait if another system bus master is using the system bus. It does, however, reduce the number of arbitration cycles needed for the LCD controller to win the system bus. The end result is that the system bus latency for the LCD controller decreases, while the latency for the other bus masters slightly increases.

This setting is likely to help LCD display refresh in a system where many peripherals are requesting the system bus, but may not help when the Au1 core is accessing slow PCMCIA cards during the fill of the 320-word buffer.

5.0 LCD Performance Tuning

An Au1100 processor design has adequate SDRAM bandwidth and latency requirements to successfully drive a display using the integrated LCD controller. The following sections detail optimizations that can be made to improve overall system performance.

5.1 Hardware Design Considerations

Since the function of the LCD controller is fixed and predictable, there are only a few hardware design decisions to be made. These decisions are:

- LCD display size
- LCD refresh rate/timing
- Selection of Au1100 processor operating frequency
- Selection of the SDRAM
- Appropriate setting of lcd_control[22:21]
- Static bus peripheral timings

The LCD display is the single largest factor affecting overall system performance. The display size, depth and refresh rate determine the SDRAM bandwidth and the Au1 core graphics performance. The larger the display, the more SDRAM bandwidth that is needed, and the more performance that is needed from the Au1 core to do graphics. The choice of LCD display size must balance market/ customer requirements and application functionality.

The LCD refresh rate and timing must be optimized to demand the least possible bandwidth from the Au1100 processor SDRAM. Aggressive refresh rates or timing merely consumes SDRAM bandwidth and increases the chance for the under-flow condition and display artifacts.

The operating frequency of the Au1100 processor ultimately determines the overall system performance and the SDRAM clock frequency. The design should use an Au1100 processor running at an appropriate frequency to yield the desired application and graphics performance, as well as an appropriate SDRAM bandwidth.

The SDRAMs selected for the design should provide the necessary SDRAM bandwidth; prototyping and profiling the intended application is recommended. The “SDRAM Performance” application note provides insight into the selection criteria and expected bandwidth for the SDRAM in an Au1100 processor design.

The lcd_control[22:21] bits should be set according to the needs of the system. For systems with long latency static bus accesses, it may be necessary to use a setting of 4 SDRAM bursts per system bus arbitration to improve the ability of the LCD controller to fill the 320-word buffer. This feature might also prove useful for larger display panels that require aggressive refresh timings.

Accesses to static bus peripherals can have an unusually large transfer time, which directly translates into a dramatic increase in system bus latency. System designers must carefully consider the timing of all peripherals on the static bus and optimize the timings to consume the least amount of time possible. The prime example is the PCMCIA interface, where card transfer times can vary from 150ns to 250ns depending upon the card inserted. In addition, the card can also assert PWAIT# to extend the cycle time indefinitely.

5.2 Software Design Considerations

The LCD controller merely fetches pixel data from the framebuffer residing in SDRAM; it is the responsibility of software executing on the Au1 core to perform all graphics operations. The graphics driver for the Au1100 processor LCD controller can optimize framebuffer caching and mapping to improve overall system performance.

5.2.1 Framebuffer Caching

Generally speaking caching data improves overall performance. However, a framebuffer presents a unique challenge in that it is a large, infrequently referenced data structure. For even a small display panel with resolution 320x240 at 16bpp, the resulting framebuffer of 153,600 bytes easily exceeds the 16KB data cache of the Au1 core. As a direct result, caching the framebuffer displaces other useful, non-framebuffer data (such as working variables, data-sets, stack, etc.) from the cache. Furthermore, the cache is best utilized when the memory is referenced frequently; framebuffers pixels are typically only written once by graphics operations and remain unchanged until a subsequent graphics operation changes the pixel.

The net result is that it is undesirable to have the framebuffer occupy the entire cache since it reduces overall cache hit rate and in turn reduces overall system performance. However, for performance reasons, it is always desirable to do the most efficient access possible to the framebuffer. The Au1100 processor offers several options for improving framebuffer accesses.

If using the translation look-aside buffers (TLB) to access the framebuffer (that is, KSEG0 or KSEG1spaces are not used exclusively to access the framebuffer), then the framebuffer cache setting in the TLB should be one of the following, in order of preference:

1. CCA=6 (cached into way 0), with the data cache way 0 locked
2. CCA=6 (cached into way 0), without the data cache way 0 locked
3. CCA=7 (non-cached, write buffer merging and gathering)
4. CCA=2 (non-cached, no write buffer merging and gathering)
5. CCA=3 (cached, uses entire data cache)

CCA, cache coherency attributes, is a field in the MIPS® TLB. See the *Alchemy™ Au1100™ Processor from AMD Data Book* “2.4 Virtual Memory” for more information. CCA values are provided in “Table 2. CCA Values” of the data book.

5.2.2 Framebuffer and CCA=6

Case 1 is CCA=6, which is cached and streaming. Furthermore software locks way 0 of the data cache. This configuration has two mutually beneficial effects:

1. it permits the framebuffer to be cached by confining framebuffer data to way 0 of the cache, and
2. non-framebuffer data is kept out of way 0 which prevents it from being purged by framebuffer contents.

This configuration has a 4KB cache for the framebuffer, and a 12KB cache for non-framebuffer items. The following example code configures the 4KB framebuffer cache by locking way 0 of the cache. The parameter to this routine is the framebuffer address.

```
      .global dcacheStreamInit
      .set noreorder
dcacheStreamInit:
      li t0,128 # number of dcache sets
dcsiloop:
      cache 0x15,0(a0) # wb inv address if in cache
      pref 0x4,0(a0) # streaming prefetch into way 0
      cache 0x1D,0(a0) # dcache fetch and lock
      addiu t0,t0,-1 # decrement sets
      bne zero,t0,dcsiloop
      addiu a0,a0,32 # increment address by cacheline size
      j ra
      nop
      .set reorder
```

When this setting is used in conjunction with lcd_control[C]=1, there is no need to flush the data cache to SDRAM; the data cache snoop mechanism returns current data for cache lines that contain framebuffer data.

This is the preferred configuration as it permits framebuffer caching, coherent updates, and prevents non-framebuffer items from being purged from the data cache.

Case 2 is CCA=6, and way 0 of the data cache is not locked. In this configuration, most of the benefits just described are realized, but non-framebuffer data can land in way 0. In doing so, framebuffer and non-framebuffer data can displace each other from way 0, degrading the full benefits of locking way 0.

5.2.3 Framebuffer and CCA=7

Case 3 is CCA=7, which is non-cached, with write buffer merging and gathering. In this configuration, the framebuffer is not cached, but writes (e.g. blits) to the framebuffer can be merged and gathered for more efficient burst accesses to the SDRAM. Burst accesses to SDRAM result in improved throughput and increase overall system performance.

The lcd_control[C] setting should be 0 (non-coherent) as the framebuffer contents are never in the data cache.

5.2.4 Framebuffer and CCA=2

Case 4 is CCA=2 which is non-cached and non write buffer merging or gathering. In this configuration, all framebuffer accesses travel through the writebuffer individually, thus consuming more SDRAM bandwidth than burst accesses with CCA=7.

The lcd_control[C] setting should be 0 (non-coherent) as the framebuffer contents are never in the data cache.

5.2.5 Framebuffer and CCA=3

Case 5 is CCA=3, which is cached. Furthermore, CCA=3 permits framebuffer contents to use the entire data cache. As previously noted, if the framebuffer occupies the entire data cache, the overall system performance degrades. Therefore using CCA=3 is not recommended.

If this setting is used, the lcd_control[C] setting must be 1 (coherent); the data cache snoop mechanism returns current data for cache lines that contain framebuffer data.

5.2.6 Framebuffer Mapping

When using the translation look-aside buffers (TLB) to access the framebuffer (that is, KSEG0 or KSEG1spaces are not used exclusively to access the framebuffer), the framebuffer mapping should attempt to use a single TLB.

Most software environments/operating systems use a 4KB page size. The number of pages required to cover an entire framebuffer of various sizes is provided in Table 9: “Number of Framebuffer Pages”.

Table 9: Number of Framebuffer Pages

	Width (pixels)	Height (pixels)	Depth (bits per pixel)	Size (Bytes)	4KB Pages
QVGA	320	240	8	76,800	19
QVGA	320	240	16	153,600	38
VGA	640	480	16	614,400	150
XGA	800	600	16	960,000	235
SVGA	1024	768	16	1,572,864	384

The Au1 core has a 32 dual-entry TLB that can map a maximum of 64 pages. If the framebuffer is mapped using 4KB pages, then as drawing takes place across the display, two performance limiting effects come into play:

1. TLB misses occur more frequently which degrades the performance of the drawing routines, and
2. the updates to the TLB to map framebuffer pages displace other valid code and data mappings from the TLB and degrade overall system performance.

The larger the display size, the higher the frequency of the TLB misses and the longer it takes for graphics operations to complete.

As graphics load/store instructions miss in the TLB, an exception is taken. Software optionally stores context, performs a table walk, updates the TLB, optionally restores context and re-initiates the load/store operation that caused the TLB miss. Furthermore, the MIPS TLB contains mapping for both code and data, so TLB misses due to framebuffer accesses can result in displacing valid TLB entries for program instruction/code pages. In effect, program code, data and framebuffer all compete for the limited number of entries in the TLB. Avoiding TLB misses is therefore desirable, and mapping the entire framebuffer using a single TLB eliminates such performance limiting effects.

The Au1 TLB can handle page sizes up to 16MB (and in reality up to 32MB due to the dual-entry TLB). For display sizes that the Au1100 processor LCD controller can handle, a 1MB page size covers the entire framebuffer, and 2MB covers surface flipping/ping-pong buffers. Thus, the entire framebuffer can be mapped with a single TLB entry.

In order to map the entire framebuffer with a single TLB, the following must occur:

- The framebuffer memory must be a valid TLB PageSize bytes in size, or PageSize*2 in size. The framebuffer memory must be mapped by exactly one TLB, with either one or both entries valid.
- The framebuffer memory must be aligned on a PageSize, or PageSize*2, boundary, e.g. for a 1MB PageSize, the alignment of the physical address must be on 1 MB boundary.
- The process virtual address into which the framebuffer is mapped must also be aligned on the same boundary, e.g. for a 1MB PageSize, the alignment of the virtual address must be on a 1MB boundary.

With a single TLB entry, TLB misses and the associated performance degradation are minimized.

Depending upon the software environment, one additional performance improvement can be realized by mapping the framebuffer with a static, or wired, TLB entry. The MIPS32™ TLB permits certain TLB entries to not participate in the random TLB replacement algorithm (dynamic) and thus remain in the TLB indefinitely (static) until removed by software. By using a static TLB entry, TLB misses caused by framebuffer accesses are completely eliminated.

Combining the optimizations for framebuffer caching and mapping, the ideal framebuffer configuration uses a single, static TLB entry covering the entire framebuffer with CCA=6 and data cache way 0 locked.

6.0 Conclusion

The Au1100 processor LCD controller provides a cost-effective, flexible solution for connecting to a variety of displays. While the performance of the LCD controller is constant, system design issues, in particular the long-latency static bus accesses, can impact the ability of the LCD controller to maintain display refresh. Several optimizations including choice of LCD panel, Au1100 processor operating frequency, and software optimizations of framebuffer caching and mapping are presented for fine-tuning the Au1100 processor based design.

7.0 References

1. *Alchemy™ Au1100™ Processor from AMD Data Book*, AMD, 2002.
2. *AMD Alchemy Solutions Au1000, Au1100 and Au1500 Processors SDRAM Performance - Application Note*, AMD, 2003.