Tsi107™
PowerPC™ Host Bridge
Design Guide

Document Number: 80C2000_AN001_01
Document Status: Formal
Release Date: October 2003

Tundra Semiconductor Corporation

Trademarks

TUNDRA is a registered trademark of Tundra Semiconductor Corporation (Canada, U.S., and U.K.). TUNDRA, the Tundra logo, Tsi107, and Silicon Behind the Network, are trademarks of Tundra Semiconductor Corporation. All other registered and unregistered marks (including trademarks, service marks and logos) are the property of their respective owners. The absence of a mark identifier is not a representation that a particular product name is not a mark.

Copyright

Published in Canada

Disclaimer

Tundra assumes no responsibility for the accuracy or completeness of the information presented, which is subject to change without notice. Tundra products may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. In no event will Tundra be liable for any direct, indirect, special, incidental or consequential damages, including lost profits, lost business or lost data, resulting from the use of or reliance upon the information, whether or not Tundra has been advised of the possibility of such damages. The information contained in this document does not affect or change Tundra's product warranties.

Mention of non-Tundra products or services is for information purposes only and constitutes neither an endorsement nor a recommendation.

This information is updated from time to time, and may be out of date. Please contact a member of the Tundra Technical Support Team, or check the Support section of the Tundra web site at www.tundra.com to ensure that you have the most recent version of this document.

This document describes how to use the features of the Tsi107 PowerPC host bridge support chipset. It also describes how to convert a system from the Tsi106 to the Tsi107 to increase performance, reduce cost, and improve board space. It includes the following topics:

To locate any published errata or updates for this document, refer to the Tundra website at www.tundra.com/tsi107.

1.1 Introduction

The Tundra Semiconductor Corporation (Tundra) Tsi107 is the next generation PowerPC Host Bridge, after the Tundra Tsi106. The Tsi107 is upwardly compatible with the Tsi106 at a software level, preserves all the essential hardware features of the Tsi106, and adds many new features, including:

- Integrated memory data bus registers
- Integrated on-the-fly ECC correction
- Two additional ROM/Flash chip selects ($\overline{\text{RCS2}}$, $\overline{\text{RCS3}}$)
- Fewer restrictions on $\overline{\text{RCS1}}$ accesses
- Integrated 5-port PCI arbiter
- Integrated 5- or 16- port interrupt controller
- Full PCI peripheral/target mode support, including IDSEL and $\overline{\text{INTA}}$
- I^2C controller

The Tsi107 also includes many other features, such as:

- DMA controller
- Programmable timers
- Watchpoint registers (debug registers)
- I_2O controller

These features assist in designing embedded systems, but since they typically do not require any special hardware support, they may be used or ignored, as determined by the system architecture. Figure 1 shows a comparison of a typical Tsi106-based system and the equivalent system based on the Tsi107.

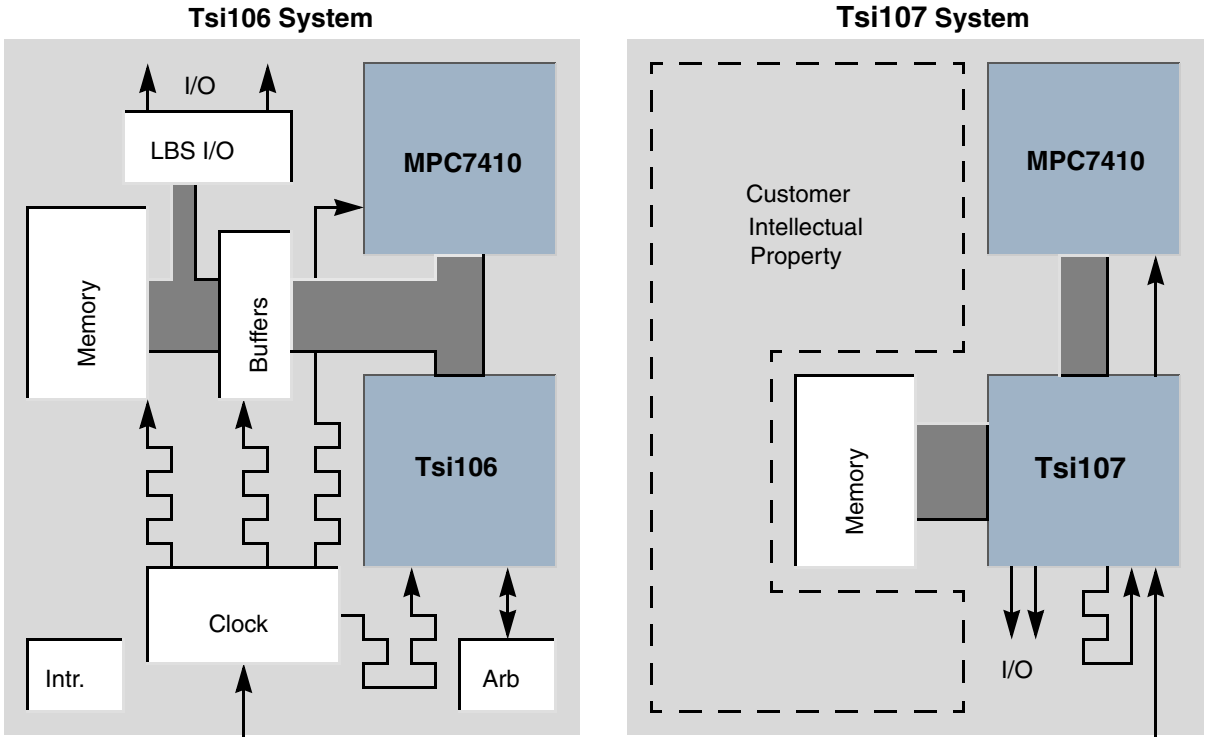

Figure 1. System Architecture Comparison

In general, designers who are familiar with the Tsi106 will find that the Tsi107 increases performance, eases design effort, and decreases overall board space. Designers new to processors implementing the PowerPC instruction set architecture will find that the Tsi107 supplies almost all of the interface circuitry needed for the host processor, with the remainder being the familiar sort of signals all embedded systems need (power, reset, etc.).

1.2 Processor Interface

The Tsi107 provides all the interface signals needed to interface between the host processor and other devices (such as SDRAM, ROM, PCI, etc.). In general, there will be a one-to-one connection between the Tsi107 and the CPU; at most there would be three loads (the Tsi107, two CPUs and a local-bus-slave device). Table 1 lists all needed connections between the Tsi107 and processor bus devices.

Table 1. Tsi107 Processor Bus Connections

Tsi107 Pin	Processor Pin	External Pull-ups?	Description
A(0:31)	same	Optional[1]	Address bus
DH(0:31), DL(0:31), DP(0:7)	same	No	Data Bus (note: not the same as the MDH/MDL/PAR memory bus).
TSIZ(0:2), $\overline{\text{TBST}}$	same	Optional	Address info (size, burst)
TT(0:4)	same	Optional	Address types (read, write, atomic, cache)
$\overline{\text{CI}}$, $\overline{\text{GBL}}$, $\overline{\text{WT}}$,	same	Optional	Address coherency
$\overline{\text{TS}}$, $\overline{\text{TA}}$, $\overline{\text{TEA}}$	same	Recommended[3]	Address/Data tenure start and completion
$\overline{\text{AACK}}$, $\overline{\text{ARTRY}}$	same	Recommended[3]	Address tenure completion signals
$\overline{\text{BR0}}$, $\overline{\text{BG0}}$, $\overline{\text{DBG0}}$	$\overline{\text{BR}}$, $\overline{\text{BG}}$, $\overline{\text{DBG}}$	Optional	Bus request/grant
$\overline{\text{BR1}}$, $\overline{\text{BG1}}$, $\overline{\text{DBG1}}$	$\overline{\text{BR}}$, $\overline{\text{BG}}$, $\overline{\text{DBG}}$	Optional	Bus request/grant (optional second CPU)
none	$\overline{\text{ABB}}$, $\overline{\text{DBB}}$ [2]	Optional	Bus busy (unused by Tsi107)

NOTES:
1 Pull-ups on the address bus are optional and only needed where power consumption is a concern during instances where the host CPU and Tsi107 are in a low-power idle state (where the address bus is rarely driven). If these circumstances do not apply, the address bus pull-ups are not required.
2 Not implemented on the MPC7400 or other G3 processors.
3 “Recommended” pullups are not required, but are very useful for bus monitoring, logic analyzer attachment, etc.

There are other signals such as $\overline{\text{MCP}}$, $\overline{\text{INT}}$, and $\overline{\text{HRESET}}$ which are provided by the Tsi107 and connected to the host processor; however these signals are not part of the 60X bus protocol so refer to Section 1.8, “Interrupt Controller” on page 27 and Section 1.10, “Reset” on page 36, respectively, for information on special pins.

1.3 Local Bus Slave

The local bus slave (LBS) is a means by which a user-created I/O device can accept 60X bus cycles with minimal intervention by the Tsi107 (which otherwise handles all 60X bus transactions). Designers who need a high-speed I/O channel or special types of memories (FIFOs, dual-port SRAMs, etc.) may use the LBS to control such devices.

Designing an LBS interface is covered in more detail in the application note, “Designing a Local Bus Slave I/O Controller,” and so will not be covered in detail here. In terms of system design, however, the LBS should be designed so as to minimize the overall capacitive loading of the data bus. This might require, for example, that large I/O system need buffering, or provide isolate address and control signal decoding in an FPGA or PAL.

1.4 Clocks

A significant improvement offered by the Tsi107 over previous solutions is the integration of a full clocking system, which given a single PCI clock input, can synthesize all the clocks needed by a typical embedded system:

- 2 processor bus clocks — CPU_CLK(0:1)
- 1 local bus slave clock — CPU_CLK(2)
- 4 memory bus clocks — SDRAM_CLK(0:3)
- 5 PCI bus clocks — PCI_CLK(0:4)

In addition, the Tsi107 includes a DLL adjustment that makes it easy to add or remove delay from memory clock signals. This allows the designer to adjust the timing window of SDRAM signals to compensate for heavily loaded systems, or to achieve PC100 compliance. The clock can be configured in many ways, but most systems will use one of two architectures based upon whether the Tsi107 is an agent or a host. It is important to realize that the descriptions below reflect only the most widely used arrangements, many others are certainly possible. The CPU and SDRAM clocks are independent of the PCI clock arrangement and will be discussed separately.

1.4.1 Host Mode Clocking

When configured as a host, a typical system will use the PCI clock buffer to drive all PCI agents as well as itself (with the PCI_SYNC_IN input). When the clocks are routed to equal lengths, the PCI requirements regarding allowable clock skew are easily met, and all devices should be synchronized. Figure 2 shows an example of a typical host-mode Tsi107 system.

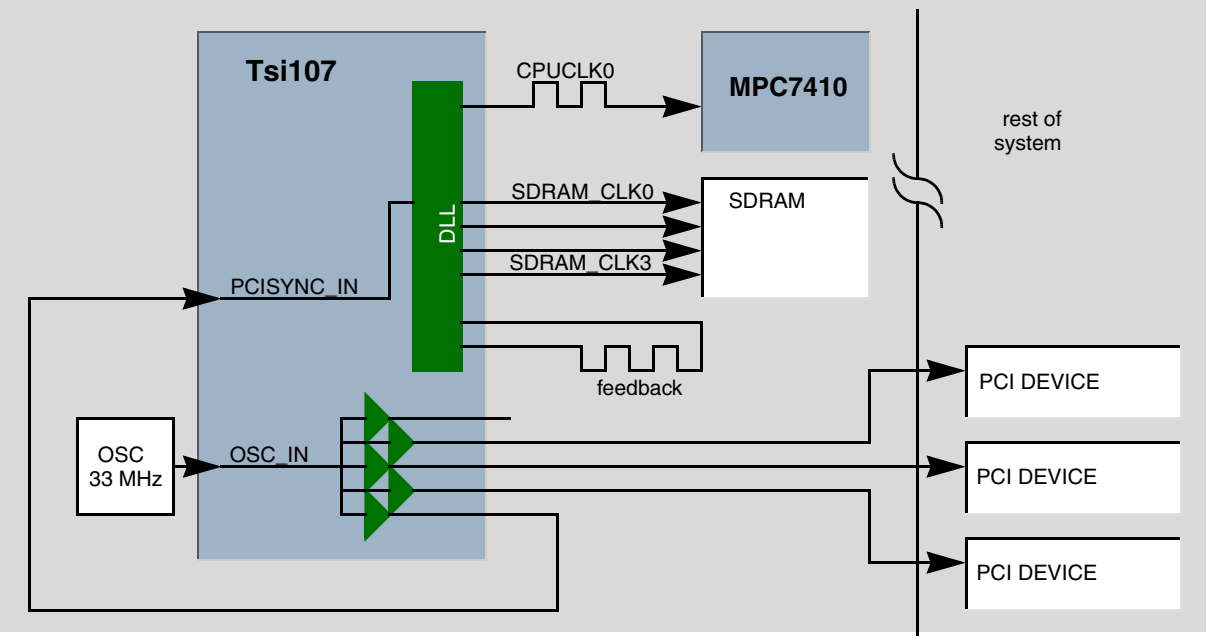

Figure 2. Host Clock Architecture

This architecture uses a 33 MHz oscillator (or frequency synthesizer) to create the baseline PCI clock. The clock is distributed equally to the Tsi107 core logic via PCI_SYNC_IN as well as the PCICLK pins of the other PCI devices.

1.4.2 Agent Mode Clocking

When configured as an agent, the Tsi107 will typically be part of a larger system, whether as a component on an embedded board, or as a PCI card plugged into a motherboard. As such, it is usually not expected to provide skew-controlled clocks to all other PCI devices on the PCI bus, but instead receives a clock from another source. For this environment, the PCI clock tree is not usually used. Figure 3 shows an example of a typical agent-mode Tsi107 system.

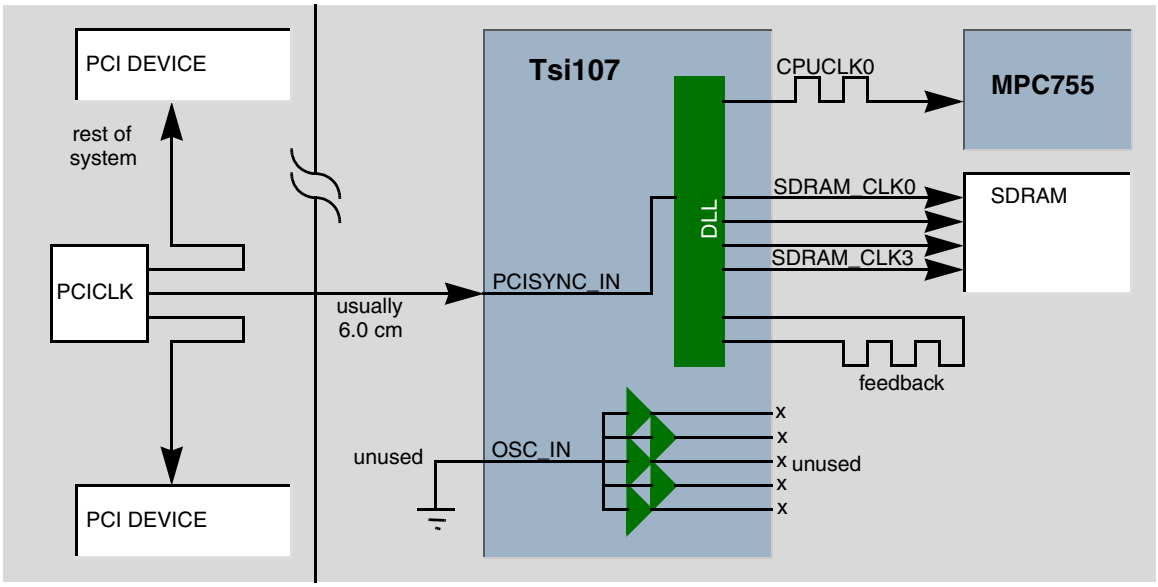

Figure 3. Agent Clock Architecture

Agents usually receive a clock signal (PCICLK) generated elsewhere in the system which drives the PCISYNC_IN pin. Because there is a small amount of delay from OSC_IN to PCICLK(0:4), it is not normally recommended that PCICLK flow through the PCI clock buffer, as this would increase the overall system skew. If absolutely required, it may be possible to do so by correspondingly shortening the PCICLK trace. On an embedded motherboard there is usually more flexibility about clock trace lengths as opposed to a PC-type plug-in PCI card.

1.4.3 Memory Clocks

The Tsi107 provides four clocks signals, SDRAM_CLK(0:3), which can be used to drive one or more SDRAM components. The Tsi107 controls these clocks with a digital locked loop (DLL) which can be used to adjust the relationship between clocks and SDRAM control signals or data. Such deliberate skewing of clocks is often required to compensate for a heavily loaded memory bus, or to communicate with SDRAM components which do not exactly match the AC timing provided by the Tsi107. The usual method of creating skew is to add PCB trace length to clocks This is especially true in the case of PC100 and PC133 SDRAM components or modules, which require additional output hold time.

The DLL in the Tsi107 is similar to a PLL except that it divides the clock period into discrete intervals, in this case into 128 intervals. The DLL drives the SDRAM_SYNC_OUT signal and measures the number of intervals until the clock is detected on the SDRAM_SYNC_IN pin. As trace length is added to the feedback path (SDRAM_SYNC_OUT to SDRAM_SYNC_IN), the DLL numerically adjusts the delay to the next clock edge so that the SDRAM_CLK signals starts sooner, relative to the internal bus clock (referred to as “sys_logic_clk”).

Why sooner? The assumption is that the clock and data traces to the SDRAM are all of equal length, but that the control signals are more heavily loaded (often true), or that additional output hold time is needed for SDRAM. The usual way of compensating for such issues is to add trace delay to the SDRAM clocks, but this can take a lot of board space with four or more clocks. By adding the trace delay to the feedback path alone, less board space is required, design is easier, and routing the board is easier. An example of the effect of lengthening SDRAM feedback path is shown in Figure 4.

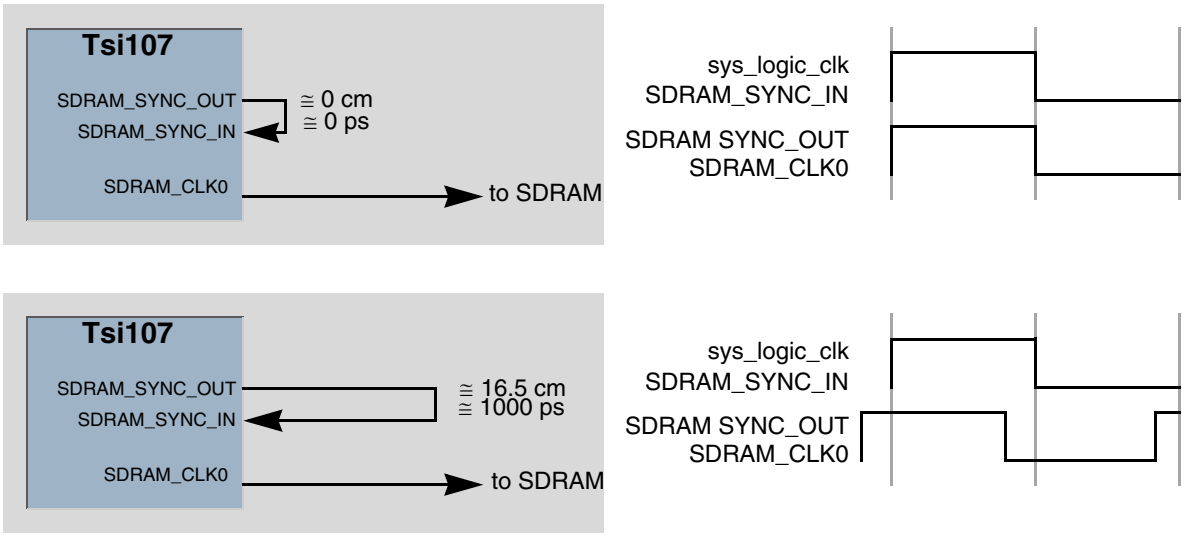

Figure 4. SDRAM Feedback Path Overview

To understand the Tsi107 DLL clock generator, it is essential to realize that the DLL has absolutely no effect whatsoever on the AC timing of any SDRAM signal. All Tsi107 signals are synchronous to the internal core bus clock, *sys_logic_clk*, regardless of what length of feedback path is used. The only signals affected by the DLL are SDRAM_CLK(0:3), CPU_CLK(0:2) and SDRAM_SYNC_OUT.

1.4.4 Clocking and SDRAM Memory Systems

Section 1.5.2 covers the connections of the memory system; these are relatively straightforward but clocks need special treatment - they are the heart of an SDRAM memory system. Assuming good layout techniques are used and the traces for the SDRAM controls, data and clocks are kept relatively equalized (traces routed to the same lengths, to within ± 10%), then the DLL of the Tsi107 can be used to set the timing for the SDRAM components.

To determine the timing adjustments, it is necessary to determine what effect PCB trace length and SDRAM component loads have on the propagation delay. This application note is not going to rehash an entire book's worth of electromagnetic wave equation derivations; instead, the appendix has several good references for a designer to refer to in checking that the assumptions used here are a good match for the design. If not, slight modifications these equations will be needed.

Table 2. Assumed PCB Electrical Parameters

Trace Height	Trace Width	Impedance Z_0	Inductance L_0	Capacitance C_0	Propagation Delay
0.005	0.005	63.83 Ω	3.93 nH/cm	0.91 pF/cm	58.06 ps/cm
			9.97 nH/in	2.31 pF/in	147.47 ps/in

Using these values, the only other data we need is:

- Bus speed 66–100 MHz
- Maximum memory trace length (clocks, controls, and data) 2–15 cm
- Capacitive loads (number and quantity) 5–100 pF

The first two can usually be determined (or predetermined). The latter is generally obtained from the data sheet, however in the case of SDRAM DIMM sockets, the user may install a module of varying loading factors. The designer using DIMMs and allowing user-upgrades should design to the worst-possible loading using.Table 3.

Table 3. Typical SDRAM DIMM Module Capacitance

DIMM Type	SDRAM Component Width	Typical Capacitance			Notes
		Controls: CKE, WE, CS, etc.	Clock	DQ, DQM	
unbuffered	4 bits	70 pF	40 pF	20 pF	1, 3
unbuffered	8 bits	50 pF	30 pF	12 pF	1
unbuffered	16 bits	30 pF	15 pF	5 pF	1
registered	any	7.5 pF	26 pF	9 pF	2, 4

Notes:
1. Loads are per-bank. For dual-bank DIMMs, multiply loads by 2.
2. Loads are per-bank for DQ and CLK0. All others remain constant.
3. Not generally recommended due to large loading.
4. Preferred for large memory arrays.
5. No guarantees that these values are worst-case.

Two equations have to be dragged in from the references, however. The first is a simple reminder that capacitive loads should be treated as a distributed load over the total trace length, as shown in the following equation (as opposed to just a lumped value):

$$C_O' = C_O + \frac{N_O \cdot C_L}{length}$$

In addition, a second equation for determining the line delay of the trace (both inherent in the copper traces and due to the capacitive loading) is needed:

$$delay = \sqrt{L_O \cdot C_O'}\ (ps/cm)$$

Using the information and equations listed above, we can see what sort of designs are possible, as shown in Table 4.

Table 4. Example SDRAM Designs

	Case “A” 2 Registered DIMMs	Case “B” Single SODIMM module	Case “C” 3 Unbuffered DIMMs
DIMM type	registered	unbuffered	unbuffered
SDRAM type	any	16-bit	4 bits
PCB Trace	8.9 cm	6.4 cm	12.7 cm
Co’	0.91 pF/cm + (2 DIMMs * 7.5 pF) / 8.9 cm	0.91 pF/cm + (1 SODIMMs * 30 pF) / 6.4 cm	0.91 pF/cm + (4 DIMMs * 70 pF) / 12.7 cm
	2.60 pF/cm	5.60 pF/cm	22.96 pF/in

Table 4. Example SDRAM Designs

	Case “A” 2 Registered DIMMs	Case “B” Single SODIMM module	Case “C” 3 Unbuffered DIMMs
Trace delay	sqrt(3930 pH/cm * 2.60 pF/cm)	sqrt(3930 pH/cm * 5.60 pF/cm)	sqrt(3930 pH/cm * 22.96 pF)
	101 ps/cm	148 ps/cm	300 ps
TOF	899 ps	947 ps	3814 ps
Prognosis	Excellent	Good	Poor

As you might expect, the key to achieving a high-speed memory design is to minimize the capacitive loading and trace length; registered DIMMs have become invaluable for just this reason. Once the data have been determined, we can examine the timing of the memory system.

Table 5. Memory Timing Analysis

Factor	Case “A” Value	Case “B” Value	Case “C” Value
Cycle Time	10000 ps	10000 ps	10000 ps
Clock-to-output	- 5500 ps	- 5500 ps	- 5500 ps
Jitter	- 150 ps	- 150 ps	- 150 ps
Time-of-flight	- 899 ps	- 947 ps	- 3814 ps
Input Setup	- 2000 ps	- 2000 ps	- 2000 ps
Input Hold	- 1000 ps	- 1000 ps	- 1000 ps
Margin	451 ps	403 ps	- 2464 ps
Max Bus Speed	100 MHz	100 MHz	78 MHz

As expected, the capacitive load on case “C” was far too high to achieve 100 MHz; even 83 MHz operation was well beyond its capabilities. For cases where 100 MHz is possible, Figure 5 shows an example of timing factors as information flows between the Tsi107 and the SDRAM.

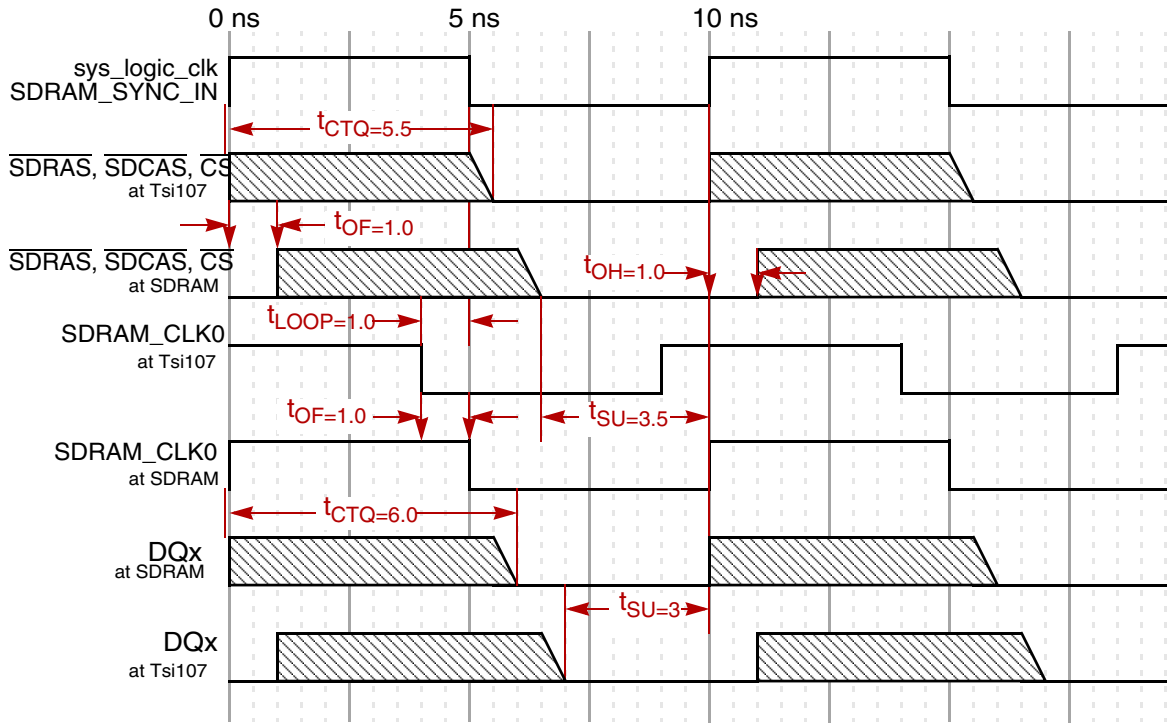

Figure 5. Memory Timing for 100 MHz

1.4.5 Expanding Memory Clocks

Four clock are sufficient to directly drive two DIMM modules (two physical banks each), four registered DIMMs (one clock per module), or four discrete SDRAM components. If this is insufficient, the memories or modules must share clock signals; in such a case, the extra capacitive loading on the clock traces will cause the time-of-flight (TOF) to be correspondingly longer. Using the feedback path, this effect can be removed by lengthening the trace as described in Table 2. This longer feedback path will cause the SDRAM clocks to “launch” earlier so that they will arrive along with the memory address and control signals.

If sharing is not sufficient, a clock regeneration device, also known as a zero-delay buffer, may be used to replicate some of the clocks so that more are available, as shown in Figure 6.

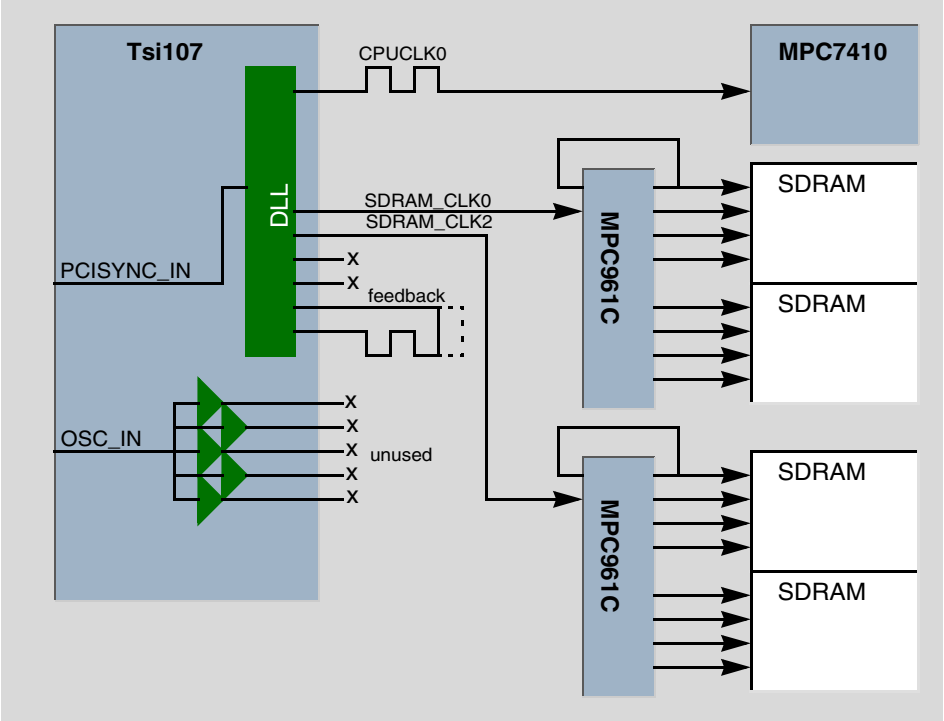

Figure 6. Clock Expansion Using Zero-Delay Buffers

Typical zero-delay buffers generate output clocks that are aligned to within 150ps of the input clock, which of course is not really zero, but can be adjusted to effectively zero by lengthening the Tsi107 DLL feedback path. This effectively treats the delay through the zero-delay buffer as extra trace length, so the feedback path is lengthened using the formula:

$$\text{ExtraTsi107delay} = \frac{350\text{ps}}{58\text{ps/cm}}$$

where 58 ps/cm (147 ps/in) is the propagation time of our example PCB trace (refer to Table 2 for assumptions).

Note that similar zero-delay buffers are included on registered SDRAM DIMMs. If a system restricts the type of memory usable to registered DIMMs, it may be possible to eliminate the extra zero-delay buffers (particularly since registered SDRAM DIMMs do not require more than one clock input, CK0).

1.4.6 CPU Clocks

The Tsi107 also supplies three clocks which can be used to clock two processors and one local-bus slave device, as needed. An unfortunate side-effect of the Tsi107 clock generator architecture is that the CPU clocks are actually generated by the SDRAM clock DLL, so CPU clocks are always synchronous to the SDRAM clocks. When the DLL feedback is used to adjust the timing relationship between the SDRAM_CLK and the SDRAM, the processor bus will be inadvertently affected.

The solution to this is to add the same amount of delay to the CPU clock that is added to the SDRAM_SYNC feedback path. Since 1 ns of feedback trace length causes the SDRAM_CLK signals to begin 1 ns early (relative to the internal bus clock), adding 1 ns of trace length to the CPU_CLK signals will re-synchronize the CPU clocks so that they arrive at the destination in sync with the internal Tsi107 bus clock.

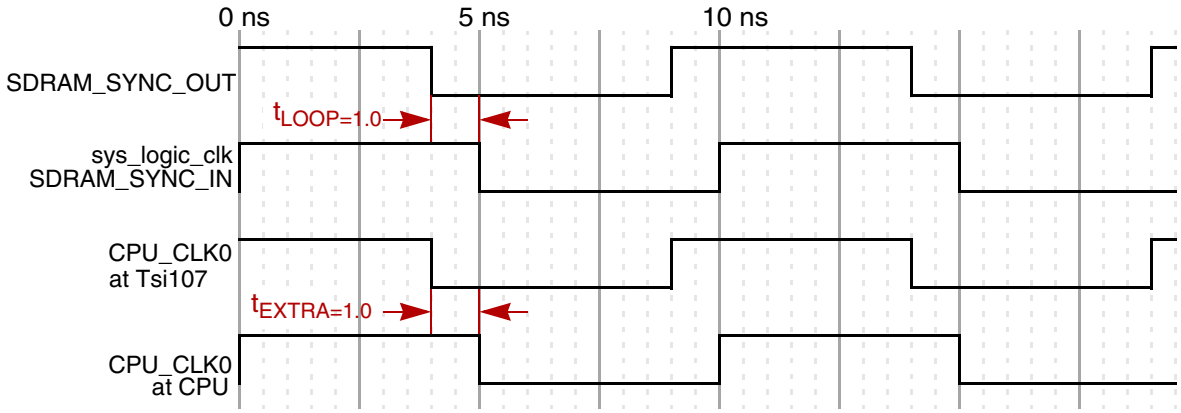

Figure 7. Resynchronizing CPU Clocks

Figure 7 shows how adding to the CPU_CLK traces (of delay t_{EXTRA}) restores the synchronization needed to communicate with the host processor.

1.5 Memory Architecture

The Tsi107 contains a high-speed SDRAM/DRAM memory controller and a ROM controller. The memory interface is completely separate from the processor bus, so that heavy loading on the SDRAM bus will not affect the processor bus. In addition, the memory interface of the Tsi107, unlike the Tsi106, has internal registered buffers which allow it to increase performance and to implement “on-the-fly” ECC. Since EDO is more-or-less a subset of SDRAM, it will not be covered further.

1.5.1 Banks

A momentary digression on the use of the term “bank.” The meaning of “bank” varies depending upon what level of a memory system you are looking at. Within the SDRAM memory silicon, the term is used to describe devices which can maintain 2-to-4 open pages simultaneously (no activation delay). This is the meaning of bank as used to set the MCCR1 register in the Tsi107: 64 Mbit/4-banks, 64 Mbit/2-banks, etc. The Tsi107 uses this information to drive the bank address pins (BA(0:1)) properly. The term “internal bank” is also often used.

At the level of a memory module (SODIMM, DIMM, etc.), the term bank is used to describe the number of independently selectable groups of SDRAM components. A standard SDRAM module contains 64-bits of memory using components of various widths (4, 8 or 16 bits). Each bank of memory is controlled by a pair of chip select pins ($\overline{CS0}$ and $\overline{CS2}$ for 64-bit modules), which are connected together to configure the module as a 64-bit bus. To increase density, some modules have a second, independent set of components controlled by a second pair of chip selects ($\overline{CS1}$ and $\overline{CS3}$). This is described in the I^2C SPD EEPROM as a second “bank”, and such devices are referred to as “dual-bank” modules, as opposed to “single-bank” modules.

Lastly, the Tsi107 refers to a bank as one of eight possible groups of memory, each of which is controlled by a separate chip select pin ($\overline{CS(0:7)}$). Tsi107 banks have different sizes and types and can be positioned at various addresses, but have common timing.

Note that a dual-bank DIMM is typically wired up to two Tsi107 banks, as shown in section on page 15. When DIMMs are connected in this fashion, what happens when single-bank DIMMs are inserted? The result is that the Tsi107 bank controlling $\overline{CS1}$ is unused; to get a contiguous range of memory, the starting and ending addresses of banks 2-7 should be adjusted to skip over unused physical banks.

1.5.2 Memory Connections

Connecting memory to the Tsi107 is fairly straightforward, with the exception of the SDRAM clock signals, which have been discussed in section 1.4.3 on page 8. Otherwise, most signals on the Tsi107 have the same name as the signals on the memory devices/modules, and can be connected one-to-one. Figure 8 shows a typical Tsi107 memory connection. The remaining signals such as SDA, SCK, SA(0:2), etc. are either standard I^2C controls for module information or are unused.

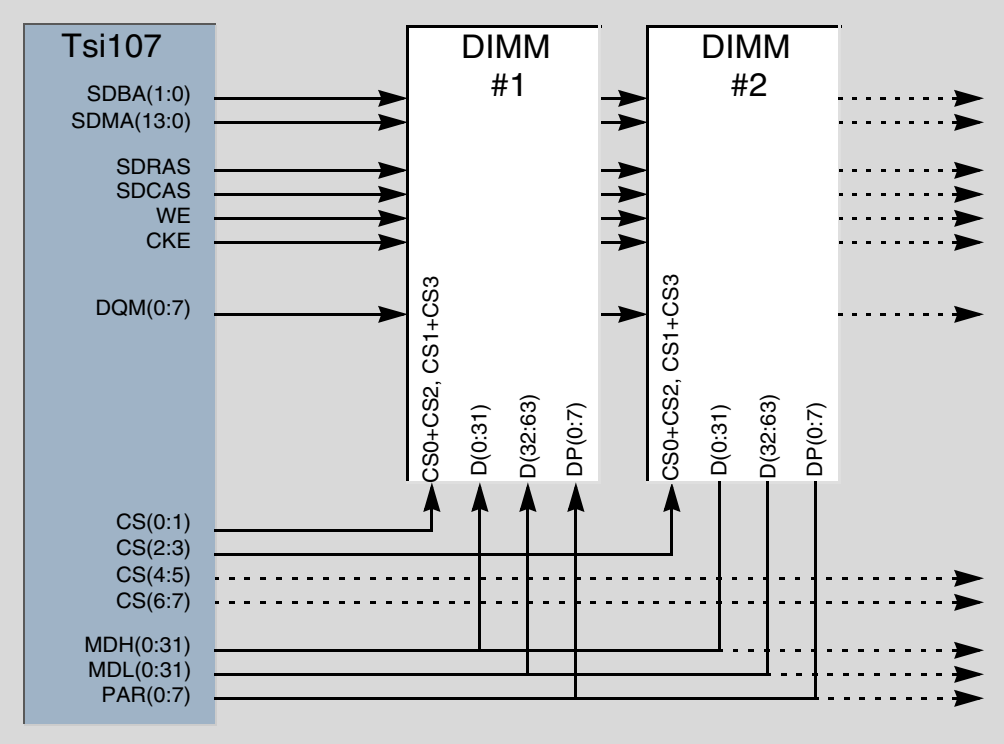

Figure 8. Tsi107 Memory Connections

There are three particular considerations to keep in mind when connecting the memory system:

- Each physical bank of memory must connect to one and only one $\overline{CS}$ line
- Memory data connects to the memory data bus (MDH/MDL/PAR), not the processor data bus (DH/DL/DP)
- The DQM signals must match with the corresponding byte lane:

DQM0	⇔	MDH(0:7)
DQM1	⇔	MDH(8:15)
DQM2	⇔	MDH(16:23)
DQM3	⇔	MDH(24:31)
DQM4	⇔	MDL(0:7)
DQM5	⇔	MDL(8:15)
DQM6	⇔	MDL(16:23)
DQM7	⇔	MDL(24:31)

The association between DQM and MDH/MDL is important when the Tsi107 is in non-ECC/non-parity memory modes, since it requires the ability to modify a single byte. In ECC or parity modes, all DQMs are

driven to the same value since only 64-bit quantities are read or written, so in that one case the DQMs can be freely associated.

The remaining memory connections can be connected from point-to-point. Table 6 shows a list of the interconnections between the Tsi107 and a typical DIMM or SODIMM module. Note that, unlike most buses on processors implementing the PowerPC instruction set architecture, the memory buses (with the exception of the data bus) use industry-standard little-endian notation, where A0 is the least-significant-bit of the addresses.

Table 6. Tsi107 Memory Address Pin Connections

Tsi107 Signal	Pin	SDRAM Name	JEDEC 168 pin SDRAM DIMM	JEDEC 144-pin SDRAM SODIMM	Notes
SDMA(13:0)	E10, F9, D9, F8, E8, D8, B8, E7, C7, B7, A7, B6, A6, A5	A(13:0)	132, 126, 123, 38, 121, 37, 120, 36, 119, 35, 118, 34, 117, 33	72, 70, 112, 111, 109, 105, 104, 103, 34, 32, 30, 33, 31, 29	5
SDBA(0:1)	A9, A8	BA(0:1)	122, 39	106, 110	
DQM(7:0)	D11, F12, C2, B3, A10, A11, B1, A2	DQM(7:0)	131, 130, 113, 112 47, 46, 29, 28	118, 117, 26, 24, 117, 115, 25, 23	
CS0	E6	CS	30+45	69	1
CS1	C4	CS	114+129	71	1
CS(2:7)	D5, E4, C10, F11, B10, B11	CS	30+45 or 114+129	69 or 71	1
SDRAS	B4	SDRAS	115	65	
SDCAS	D4	SDCAS	111	66	
WE	A3	WE	27	67	
CKE	A12	CKE	63+128	62+68	3
MDH(0:31)	M6, L4, L6, K2, K4, K5, J4, J6, H4, H5, G3, G5, G6, F5, F1, E1, B14, D15, B15, E16, D16, C16, D18, D17, B17, F18, E19, E20, B19, B20, B21, A22	D(0:31)	2, 3, 4, 5, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 19, 20, 55, 56, 57, 58, 60, 65, 66, 67, 69, 70, 71, 72, 74, 75, 76, 77	3, 5, 7, 9, 13, 15, 17, 19, 37, 39, 41, 43, 47, 49, 51, 53, 83, 85, 87, 89, 93, 95, 97, 99, 121, 123, 125, 127, 131, 133, 135, 137	4

Table 6. Tsi107 Memory Address Pin Connections

Tsi107 Signal	Pin	SDRAM Name	JEDEC 168 pin SDRAM DIMM	JEDEC 144-pin SDRAM SODIMM	Notes
MDL(0:31)	M5, L1, L2, K1, K3, J1, J2, H1, H2, H6, G2, G4, F4, G1, F2, E2, F14, F15, A16, F17, B16, A17, A18, A19, B18, E18, D19, F19, A20, C19, D20, A21	D(32:63)	86, 87, 88, 89, 91, 92, 93, 94, 95, 97, 98, 99, 100, 101, 103, 104, 139, 140, 141, 142, 144, 149, 150, 151, 153, 154, 155, 156, 158, 159, 160, 161	4, 6, 8, 10, 14, 16, 18, 20, 38, 40, 42, 44, 48, 50, 52, 54, 84, 86, 88, 90, 94, 96, 98, 100, 122, 124, 126, 128, 132, 134, 136, 138	4
PAR(0:7)	D2, C1, A15, A14, D1, D3, F13, C13	DP(0:7)	21, 22, 52, 53, 105, 106, 136, 137	57, 59, 77, 79, 58, 60, 78, 80	2
none	none	REGE	147	none	

Notes:
1. For DIMMS, $\overline{CS0}$ and $\overline{CS2}$ should be connected to one $\overline{CS}$ pin to enable 64-bit data mode. $\overline{CS1}$ and $\overline{CS3}$ should be connected to a second $\overline{CS}$ pin if a second (physical) bank will be supported.
2. No known SODIMM module actually implements parity.
3. Connect pins together.
4. The MSB of the 32-bit data buses (MDH0 and MDL0) must be connected to the MSB of the SDRAM (D0) to preserve the association between DQM and MDH/MDL. The connections shown maintain this order.
5. Note that the memory address bus is “little-endian”, so A0 is the LSB.

1.5.3 I^2C EEPROM Data

DIMMs and SODIMMs contain an I^2C EEPROM which contains a description of the SDRAM components used on the assembly. This allows the memory controller to adjust the memory timing parameters in the Tsi107 MCCR(1:4) register to get the best performance. Using the I^2C controller of the Tsi107, it is relatively easy to obtain the data from DIMMs and from one SODIMM. Since DIMMs have dedicated address pins for the EEPROMs, all the I^2C signals can be wired from point-to-point. Unfortunately, SODIMMs do not have I^2C EEPROM address pins, instead all EEPROMs have the address 0x50. If a system uses more than one SODIMM, only one of the devices can be directly connected to the I^2C bus.

There are two workarounds for this:

- Use only information from the first SODIMM.
- Use software-controlled switches to switch SCK between each SODIMM.

The latter method requires some general-purpose outputs to be available and requires a low-impedance (bidirectional) switch for each SODIMM I^2C port. The first method simply assumes that the timing information will be based on the first (and only) SODIMM. This is not unreasonable, in fact, since the Tsi107 does not support variable timing for each bank (RDLAT, CL, ACTOPRE, etc. are all common). The only variables which can differ on each bank are the size and type (2-bank/4-bank, 16Mb/64Mb) of SDRAM; software can be used to discover such information. The only caveat is that the first SODIMM should not be faster than the remaining devices, or too-aggressive timing would be used.

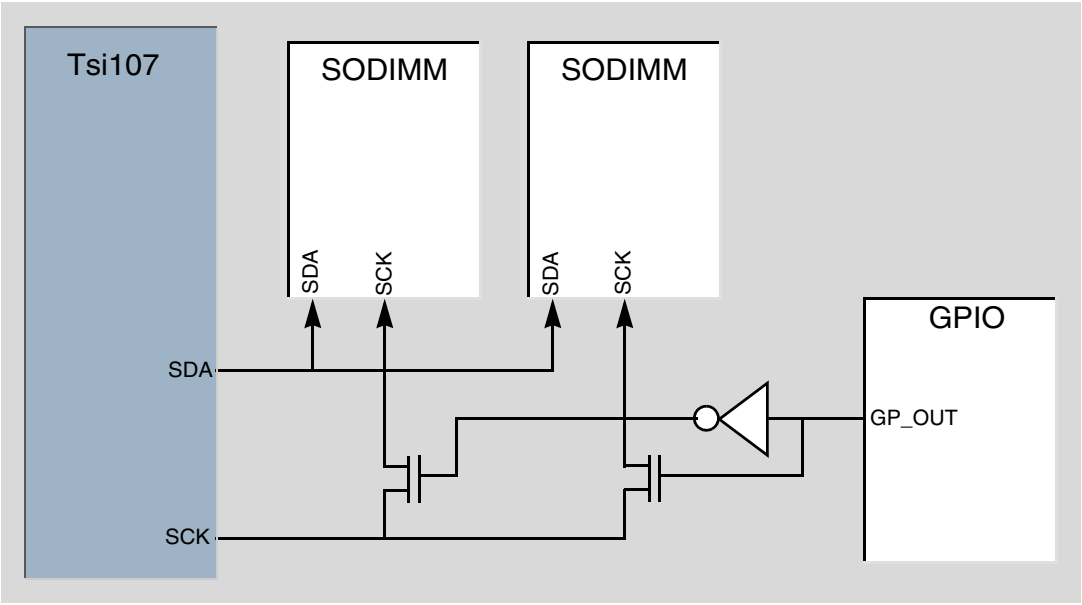

Figure 9. I^2C SODIMM Expansion

Typically, initialization code sets the SDRAM settings to “safe” values; slow but reliable operation at any frequency or SDRAM speed. Then, once memory is available for use, and software can be more easily written, the EEPROM can be read and processed.

1.5.4 Flash Memory

The Tsi107 supports up to four flash devices for code and data storage. Most embedded systems will need at least one flash device to boot from, and the remaining chip selects may be used as needed, including for general I/O purposes (see Section 1.9, “I/O Interfacing” on page 32). While the flash memory controller has some options on the width of the devices used, it is not infinitely flexible. In particular, other than the 8-bit flash modes, which are handled in a special manner, the width of a flash is always the same as the width of the SDRAM. Table 7 shows the only allowable combinations.

Table 7. Allowable Tsi107 Flash Sizes

DBUS(0:2) Setting	ROM0	ROM1	ROM2	ROM3	Notes
0 0 0	32	32	32	32	Using 32-bit SDRAM width1
0 0 1	32	8	32	32	
0 1 0	8	32	32	32	
0 1 1	8	8	32	32	

Table 7. Allowable Tsi107 Flash Sizes

DBUS(0:2) Setting	ROM0	ROM1	ROM2	ROM3	Notes
1 0 0	64	64	64	64	Using 64-bit SDRAM width
1 0 1	64	8	64	64	
1 1 0	8	64	64	64	
1 1 1	8	8	64	64	

Notes:
1 If the SDRAM is 32-bits wide, the processor is 32-bits wide as well. The MPC750 does not support 32-bit wide mode (the 60X does), so this mode should not be used with an MPC750.

No other combinations are possible; in particular, it is not possible to use 64-bit SDRAMs and 32-bit flash banks.

The flash memory controller shares the memory address lines, bank selects, write enable, and the memory data parity lines to create the flash control signals (AR, $\overline{\text{FOE}}$, $\overline{\text{WE}}$) and adds the dedicated flash control signals ($\overline{\text{RCS(0:3)}}$ and $\overline{\text{FOE}}$. Table 8 shows this remapping.

Table 8. Tsi107 Flash Address Renames

Tsi107 Signal	Flash Address Signal	Typical Flash Destination
SDMA0	AR0	A0
SDMA(1:10)	AR(1:10)	A(1:10)
SDBA0	AR11	A11
PAR7	AR12	A12
PAR(6:0)	AR(13:19)	A(13:19)
SDBA1	AR20	A20
SDMA(11:13)	AR(21:23)	A(21:23)

The flash addresses are renamed “AR” instead of “A” because the latter is the standard name already used for the host processor address bus. Note that since the flash controller shares the address lines used by the SDRAM controller, large flash arrays can increase the capacitive loading and slow down the overall memory bus speed. It may be necessary or desirable to add a buffer between the Tsi107 signals and the address pins of the flash devices if more than the minimal 8-bit boot flash is used (which has only one load and is indistinguishable from a buffer).

In a similar fashion, a large flash array may also load the memory data bus, though not as severely, and may also require buffering. Unlike address buffers, which can be permanently enabled, the data bus buffers must switch directions using the $\overline{\text{FOE}}$ and $\overline{\text{RCSx}}$ pins. Figure 10 shows an example of a heavily-loaded flash system.

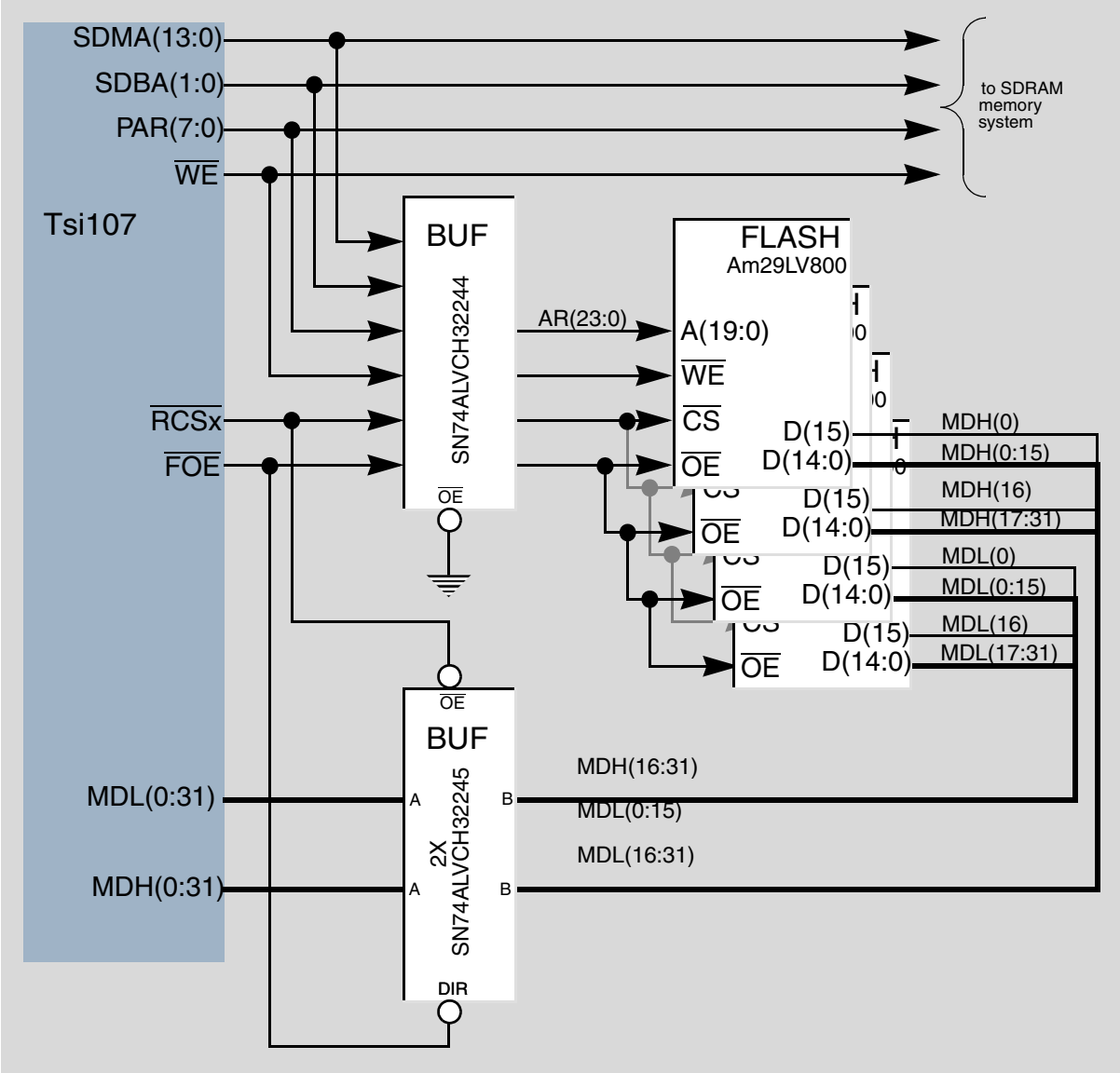

Figure 10. Buffered Flash System

The flash address and control buffers uses a high-density output-only buffer, such as the SN74ALVCH32244, which can be permanently enabled. The data bus buffers use bidirectional transceiver, such as the SN74ALVCH32245, which is enabled on $\overline{\text{RCSx}}$ (any or all AND'ed together), with the direction controlled by the $\overline{\text{FOE}}$ pin such that when $\overline{\text{FOE}}$ is low, data flows from B to A.

One special note about memory data bus bit connections: while the SDRAM data bus bits can be connected in any order, flash devices have particular (predefined) associations associated with the data bits. This is not only true with externally programmed flash devices, but programmable flash memories assign particular meanings to the data bits, so it is generally recommended to connect the MSB of the Flash/ROM (D7 or D15, depending on size) to the corresponding MSB of the processor data bus byte lane (MDH0/MDL0, MDH8/MDL8, MDH16/MDL16, and MDH24/MDL24, again depending on the device size).

1.6 PCI Interface

The PCI interface of the Tsi107 has several enhancements over the Tsi106, in particular full support for 66 MHz PCI operation and the ability to configure the part as an agent (with support for programmable inbound and outbound address ranges set by an external PCI master). There are several differences between host mode and agent mode, which are summarized in Table 9.

Table 9. Tsi107 Host and Agent Differences

Feature	Host Mode	Agent Mode	Description
IDSEL	must be grounded	must be connected to an AD(31:0) pin	The Tsi107 only allows configuration cycles when in agent mode.
Tsi107 Registers accessible from PCI	Partially	Yes	Only embedded functions can be accessed on a host (I^2O, DMA, etc.). Standard memory and processor configuration registers are inaccessible (MCCR1, PICR1, etc.).
Multiple Tsi107s supported	No	Yes	Only one Tsi107 may serve as the host (this is true of PCI hosts in general).
$\overline{\text{INTA}}$	Yes	Yes	In host mode, $\overline{\text{INTA}}$ cannot be wired to $\overline{\text{INT}}$(0:4)

The agent mode feature is possible due to the addition of the ITU (Inbound Translation Unit); when the Tsi107 is set to Agent mode, it will accept configuration cycles when the IDSEL pin is asserted. This new feature allows multiple Tsi107s to co-exist on the same PCI bus, which is extremely difficult for Tsi106-based systems to accomplish. An example multiple-Tsi107 system is shown in Figure 11.

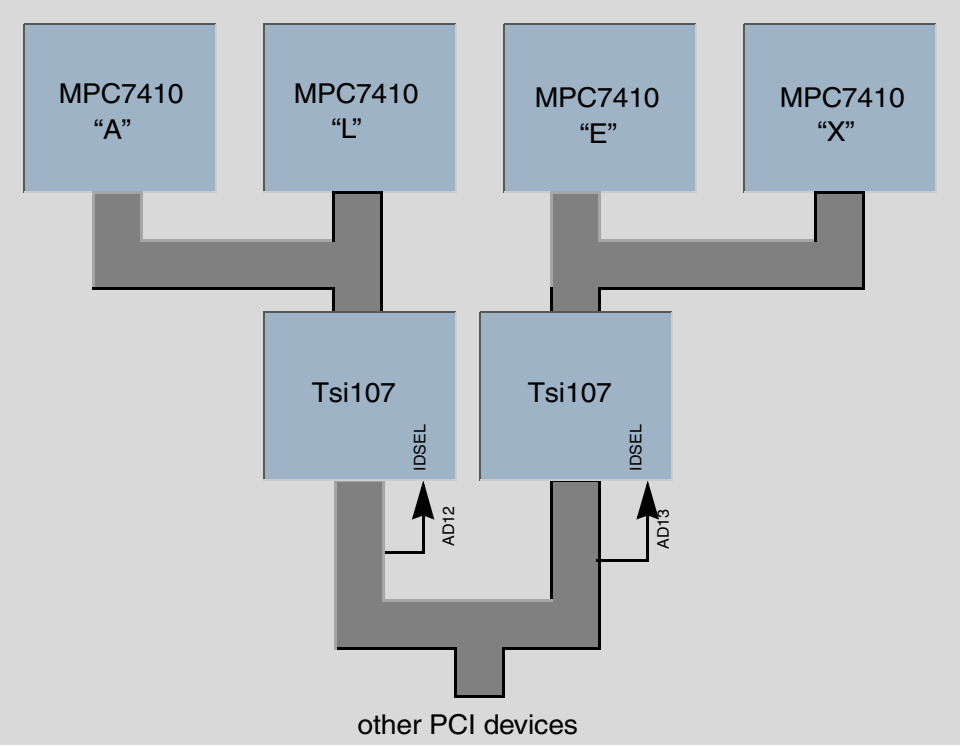

Figure 11. Multiple Tsi107-based System

In systems where such an environment is not desired, or where the Tsi107 is operating as a host, the active-high IDSEL pin must be tied low to prevent configuration cycles from being accepted.

1.6.1 66-MHz PCI PLL Adjustment

The Tsi107 supports operating the PCI bus at 66 MHz. Supporting this higher speed requires little special design effort other than the usual, good, high-speed design practices. One issue that may arise, though, is to design a card which automatically adjusts to a changing clock rate. In a PCI environment, the M66EN signal is used to globally configure with adjusting the PLL settings; if all cards are 66 MHz-capable, M66EN floats high and the PCI clock input will switch to 66 MHz. However, the Tsi107 relies upon statically-encoded PLL settings to set both the PCI frequency and its own core frequency. Furthermore, this internal core frequency must match the bus frequency of the processor in order for the two devices to be able to communicate.

Therefore, for an Tsi107-based system to operate in a flexible 33 MHz or 66 MHz PCI bus, external logic is required to dynamically change the PLL settings. If the Tsi107 is operating in a 33 MHz environment, it is sufficient to ground the M66EN pin as all non-66 MHz-capable cards do. In this environment, the PLL settings can be specified to get the maximum performance and then left alone. For a dual-speed environment, however, it may be necessary to insert logic into the PLL settings path, as shown in Figure 12.

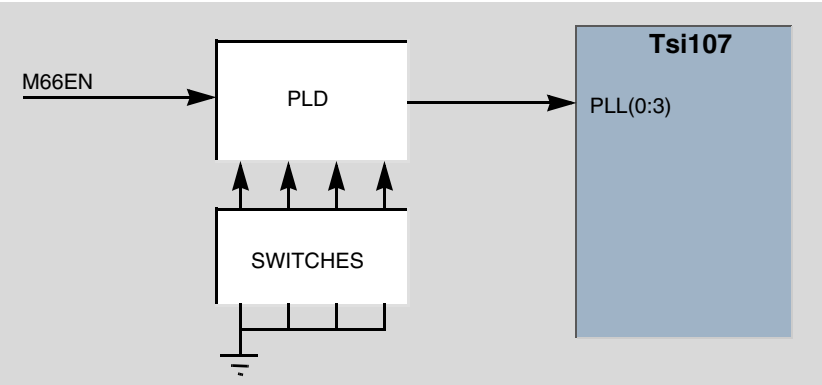

Figure 12. Tsi107 PLL Dynamic Configuration

The actual logic is highly dependant on the available and desired speeds of the host processor and the memory bus. If the PLL setting is fixed for a board (not changeable by the end user), a simple example would be:

```
PLL(0 TO 3) <= "0000"   WHEN (M66EN = '1')      -- PCI=66, MEM=66
          ELSE "0100"                           -- PCI=33, MEM=66
```

and so forth. In the preceding example, one of two different PLL codes are selected based upon the M66EN pin. This pin, and the PLL codes presented to the Tsi107, must be active and stable during the $\overline{\text{HRESET}}$ signal until it is de-asserted.

For systems which want to support multiple processor and/or memory bus speeds, the preceding logic can be generalized to alter the user settings based upon the M66EN status.

```
CASE (SPEED & M66EN) IS
WHEN "00" => PLL(0 TO 3) <= "0001";   -- PCI=33, MEM=66
WHEN "01" => PLL(0 TO 3) <= "1101";   -- PCI=66, MEM=66
WHEN "10" => PLL(0 TO 3) <= "1000";   -- PCI=33, MEM=100
WHEN "11" => PLL(0 TO 3) <= "1100";   -- PCI=66, MEM=100
END CASE;
```

In the above example, the M66EN input is controlled by the PCI bus and “SPEED” is controlled by a single user-changeable setting, for example, a four-position rotary switch. As long as the memory bus speed is held constant, the processor PLL settings do not need to be changed. Otherwise, the CPUPLL settings can be added to the programmable logic to control both the CPU and the Tsi107.

1.6.2 PCI Output Hold Time

The Tsi107 has programmable output hold time which can be adjusted to meet system requirements. The PCI specification allows 2 ns skew between any two clock signals, point-to-point. The default output hold time of the Tsi107 is 2.9ns, which works well with a typical 33 MHz PCI-based system.

However, when the PCI bus speed is increased to 66 MHz, the allowable clock skew tightens to 1ns. With the output hold at nearly 3 ns and with a 15ns bus period, it may be difficult to design a system; one way is to use the programmable output hold feature of the Tsi107. Using the M66EN signal to control the default settings for PCI hold time, as shown in Figure 13, performs this adjustment automatically.

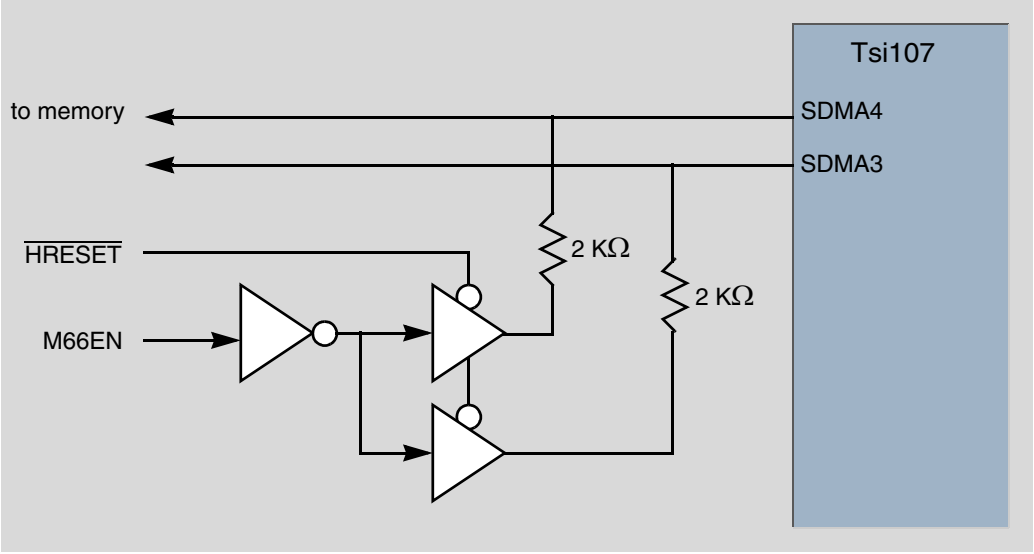

Figure 13. Automatic PCI Hold Time Adjustment

This circuit pulls SDMA4 low if M66EN is high during reset, indicating 66 MHz PCI operation. This changes the default PCI hold time from “110” (2.9ns) to “000” (0.5 ns). This function fits easily in a PAL such as the GAL22LV10, using the following equations:

```
DRV <= "00"             WHEN (M66EN = '1')      -- 66 MHz PCI
                        ELSE "11";              -- 33 MHz PCI
SDMA43_CFG <= DRV       WHEN (HRESET_B = '0')   -- Assert during reset
                        ELSE "ZZ";              -- otherwise float
```

It is also possible to provide the above adjustment in software if the code is able to determine the PCI bus speed. It is not possible to sense the M66EN signal without external hardware (you cannot determine the PCI bus speed from the HID1 PLL information, because those bits are an encoded version of the external PLL settings). The system startup software may not be able to reliably access PCI devices until this is change done, which can be a particular concern if the startup-software happens to be located on the PCI bus.

1.6.3 PCI Arbitration

While the Tsi106 requires the use of an external PCI arbiter, a function typically provided by a “south bridge” (such as the Winbond W83C553), the Tsi107 includes a five-port PCI arbiter (not including internal requesters such as the DMA engines and the core logic). Tsi107 systems can directly control or attach to the PCI bus without relying upon third-party chips.

Of course, if the PCI arbiter is not needed, it can be disabled and the Tsi107 will act like any other PCI requester.

1.7 Power

The Tsi107 follows the trend of newer CMOS processes which in using low-core voltages to attain higher operating speeds, unlike its single-voltage predecessor the Tsi106. In addition, the Tsi107 also divides the I/O power pins of various signals into logical groups, each of which may be set to different voltages. This allows the use of lower-voltage (faster) processor buses, allows clamping PCI signals to 5V if required (dictated by the PCI bus), and allows the use of 2.5V or 3.3V memory signalling.

Table 10 shows the various supported voltages.

Table 10. Tsi107 Power Supplies

Power Group	Function	No. of Pins	Nominal Voltage	Notes
VDD	Internal (core) power	15	2.5V-2.7V	
BVDD	Processor I/O power	24	2.5V or 3.3V	
OVDD	PCI/Other I/O power	16	3.3V	Interrupts, I^2C, PCI clocks, reset, NMI, JTAG
GVDD	Memory I/O power	25	2.5V or 3.3V	
LVDD	PCI Clamp Voltage	6	3.3V or 5V	
AVDD LAVDD	PLL/DLL filtered power	2	2.5V	Separate filters required
GND	Ground, common	64	ground	

The Tsi107 core logic is quite a bit smaller than a MPC755 or other embedded processor, so it requires much less power (refer to the hardware specification for exact details). Since the power is low, there are several ways to provide this power:

- Shared with other 2.5V supplies (such as L2 cache I/O) if 133 MHz is not needed.
- Auxiliary output of a multiple-output switching power supply (as with the MPC7410)
- Small linear regulator.

Figure 14 shows a simple linear regulator which will suffice in many instances for the core power needs of the Tsi107.

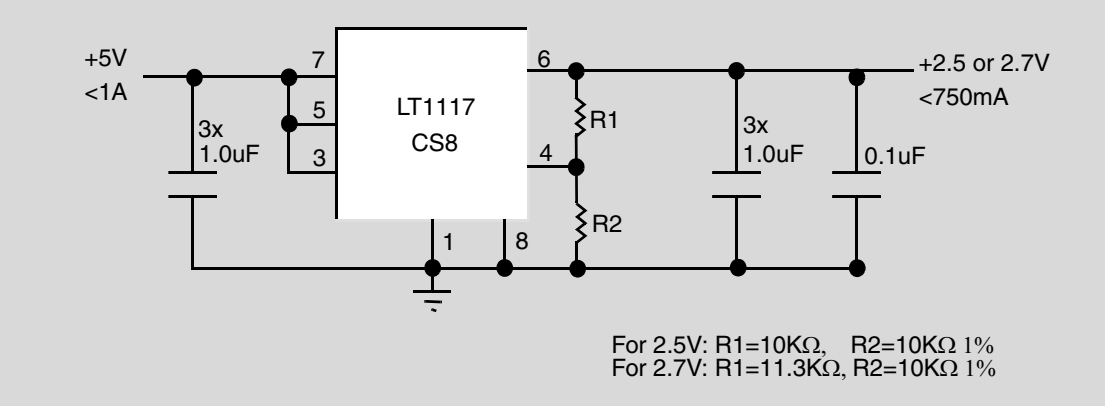

Figure 14. Simple Tsi107 Core Power Supply

If 2.5V is needed elsewhere in the system, the Tsi107 core power can be shared with other such devices. L2 cache SRAMs, L2 I/O signals, and processor bus I/O drivers are can all be operated at 2.5V or less with newer generations of embedded processors. If more power is needed, to minimize the heat dissipation produced by higher-power linear regulators, a simple switching power supply can be used.

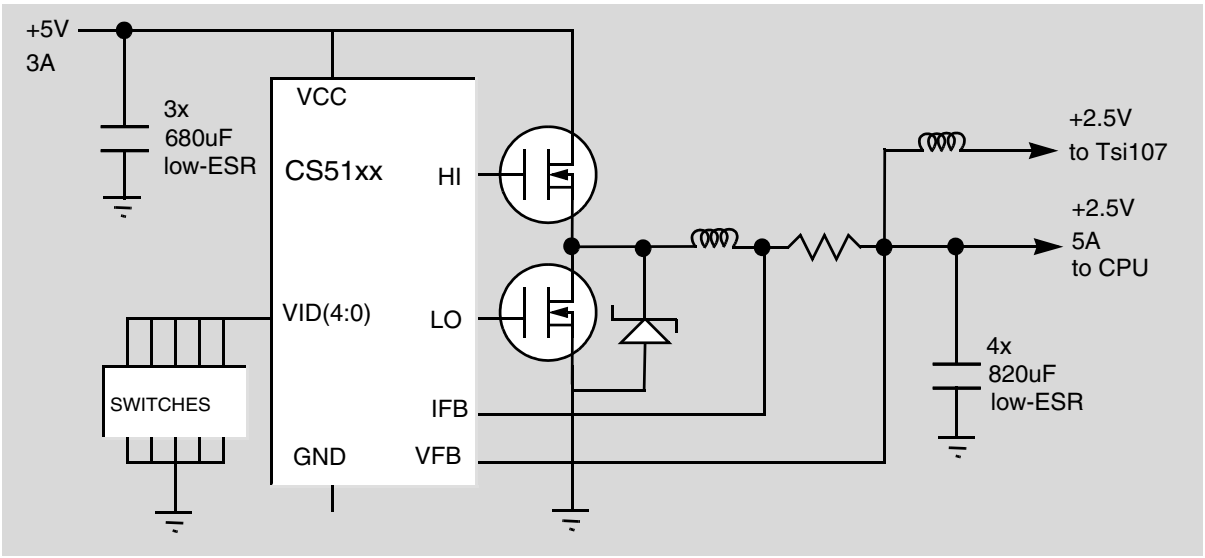

Figure 15. Shared Tsi107 Power Supply

While switching power supplies are more complicated and require more components than simple linear regulators, modern switching components are fairly easy to design with if the recommendations in the manufacturers data sheets are followed. In addition, efficiencies of 85-95% are possible, which translates into very low heat dissipation.

1.7.1 PLL/DLL Filters

The Tsi107 requires filters for the PLL power supply pins (AVDD and LAVDD) to insure that power supply noise is not coupled into the voltage-controlled oscillator (VCO), which would cause clock jitter and imprecise I/O timing (which can limit the maximum bus performance). The filter is a simple R-C network, with values chosen to minimize noise in the resonant frequency band of the VCO (approximately 500 kHz to 10 MHz). Note that these AVDD and LAVDD filters cannot be shared, there should be one filter per pin.

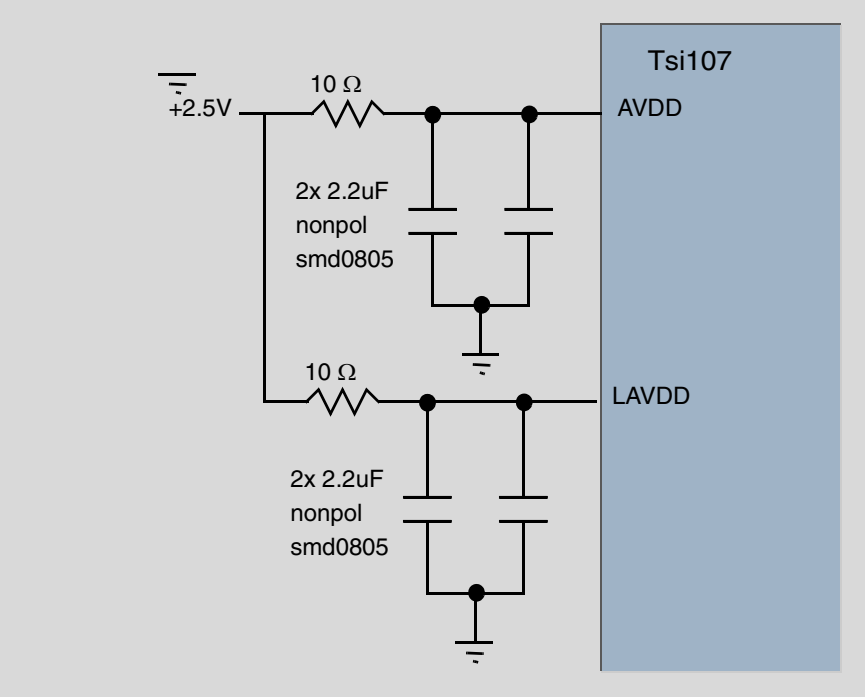

Figure 16. Tsi107 PLL/DLL Filters

The PLL power connections should be kept as short as possible between the pin and the series resistor. The AVDD pin is near the exterior of the Tsi107, and so its filter can be placed on either the top or bottom layer, as desired. The ideal placement for LAVDD is on the bottom of the board in the center (vacant) ring; this achieves the minimum trace length but requires using a dual-sided board. For PCBs not using dual-sided components, the LAVDD filter should be placed as near the exterior as possible. The power consumption is very small so normal (~6 mA), so normal trace widths can be used; preferentially the traces should not cross or otherwise couple to “noisy” traces such as clocks, address or data buses.

1.8 Interrupt Controller

The Tsi107 includes an interrupt controller, the EPIC (Embedded Programmable Interrupt Controller) which gathers interrupts from several sources and, if enabled, signals the CPU with the INT output. The EPIC function replaces logic usually provided by external logic, typically a “south bridge” component. In addition to reducing board space and cost by eliminating a (possibly unneeded) south bridge, the EPIC also includes several features not possible with a standard PC-type PIC, including:

- 5 parallel or 16 serial interrupt inputs
- PCI INTA# assertion
- Programmable timer interrupts
- I^2O interrupts
- DMA completion interrupts
- $\overline{\text{SRESET}}$ generation

The EPIC is a subset of the industry-standard OpenPIC™, lacking only the ability to route interrupts to multiple processors. An example is shown in the following Figure 17.

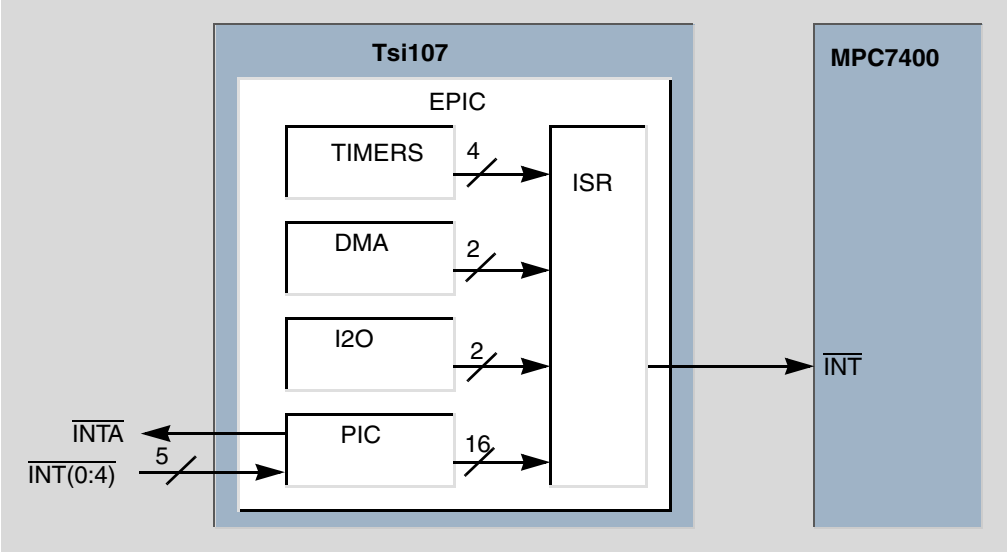

Figure 17. Tsi107 EPIC Interrupt Connections

As shown in Figure 17, the EPIC gathers a variety of internal and external interrupts sources and forwards them to the processor. Interrupts can be masked, prioritized, and set to various combinations of polarity and edge-/level-sensitivity.

The connections of the interrupt pins vary depending on the application. For agent boards, which do not typically handle external interrupts, $\overline{\text{INTA}}$ is connected to the INTA# pin of the PCI card or the appropriate INTA#-INTD# pins of the PCI host.

For host boards, the $\overline{\text{INTA}}$ pin is typically not used, and $\overline{\text{INT}}$(0:4) is connected to the interrupt pins (INTA#-INTD#) of the various PCI interrupt sources.

1.8.1 Serial Interrupt Expansion

The Tsi107 can directly accept up to 5 external interrupts with no additional hardware support. If additional interrupts are needed, up to 16 external interrupts can be supported by inserting a serially-controlled multiplexer. The EPIC demultiplexes the serial data stream and drives internal interrupt lines accordingly. The hardware requirements are minimal and can be easily supported in a complex PLD, FPGA or ASIC (but more than a 22LV10 will handle).

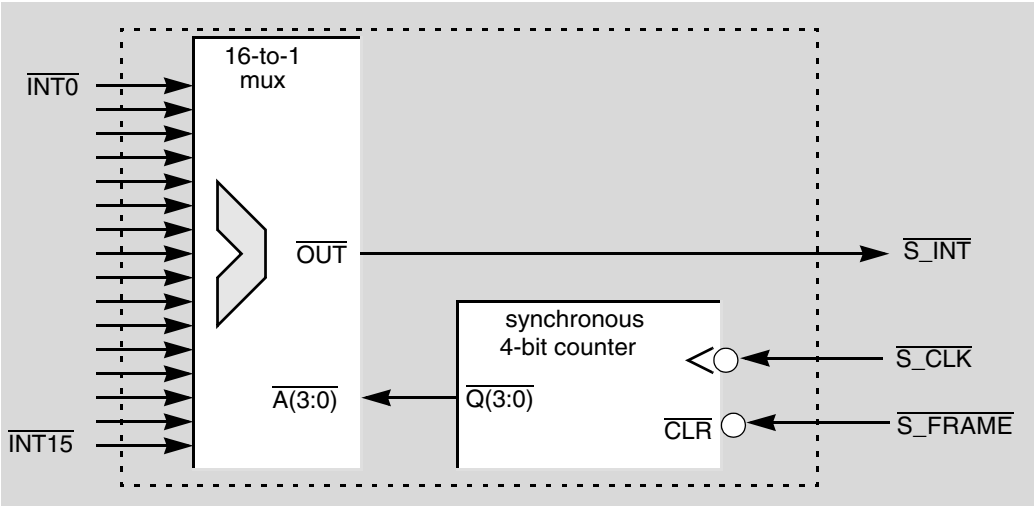

Figure 18. Tsi107 External Serial Multiplexer Block Diagram

As shown in Figure 18, this logic does not make use of the S_RST signal. This signal is asserted (for two clock cycles) only when the EPIC is converted from parallel to serial mode, and is could be used to notify external hardware to also convert from parallel to serial. Most systems will be fixed in one mode or the other, so S_RST will not be used here.

Figure 19 shows the waveforms of the simple multiplexer. By using a negative-edge triggered clock (or inverting the clock), the multiplexer can be reset to zero or advance to the next sample point in sufficient time for the Tsi107 to sample the S_INT pin on the rising edge of the clock. Since S_CLK is limited to 1/2 to 1/14th of the memory bus clock (effectively 7MHz to 50 MHz), this is well within the capabilities of most modern FPGAs or PLDs.

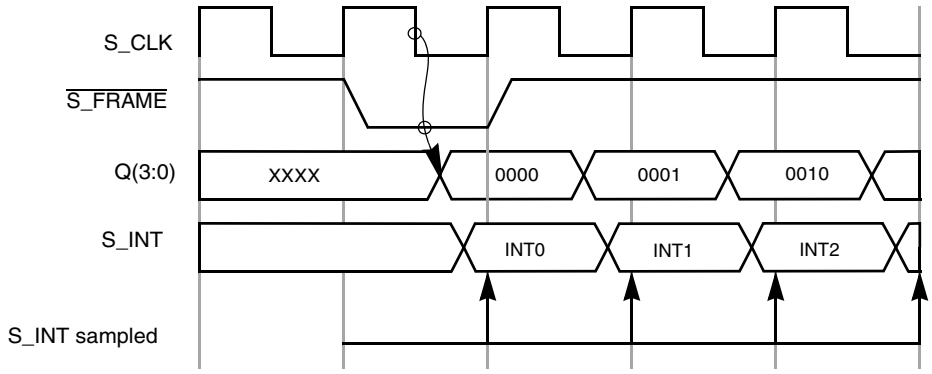

Figure 19. Tsi107 EPIC Interrupt Connections

Note that the reset signal must be synchronous; otherwise the clock-to-output timing of the Tsi107 may not be sufficient at higher speeds to guarantee that the Q counter resets to zero.

A sample of VHDL code which implements the serial multiplexer function follows:

```
LIBRARY ieee;
USE ieee.numeric_std.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_1164.all;

ENTITY SERINT IS
  PORT(
          s_clk     : IN     std_logic;
          s_frame_B : IN     std_logic;
          int_B     : IN     std_logic_vector (0 to 15);
          s_int     : OUT    std_logic;
  );
END SERINT;

--

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ARCHITECTURE BEHAVIOR OF SERINT IS

-- Architecture Declarations

  SIGNAL   q    : std_logic_vector(3 downto 0);
  CONSTANT one  : std_logic_vector(3 downto 0) := "0001";

BEGIN
  counter : PROCESS( s_clk, s_frame_B )
  BEGIN
          IF (s_frame_B = '0') THEN
                  q <= (others => '0');
          ELSIF (s_clk'EVENT AND s_clk = '0') THEN
                  q <= q + one;
          END IF;
  END PROCESS counter;

  output : PROCESS ( q, int_B )
  BEGIN
          s_int <= '1';           -- Default Assignment
          IF (q = '0000') THEN
                  s_int <= int_B(0);
          ELSIF (q = '0001') THEN
                  s_int <= int_B(1);
          ELSIF (q = '0010') THEN
                  s_int <= int_B(2);
          ELSIF (q = '0011') THEN
                  s_int <= int_B(3);
          ELSIF (q = '0100') THEN
                  s_int <= int_B(4);
```

```
            ELSIF (q = '0101') THEN
                    s_int <= int_B(5);
            ELSIF (q = '0110') THEN
                    s_int <= int_B(6);
            ELSIF (q = '0111') THEN
                    s_int <= int_B(7);
            ELSIF (q = '1000') THEN
                    s_int <= int_B(8);
            ELSIF (q = '1001') THEN
                    s_int <= int_B(9);
            ELSIF (q = '1010') THEN
                    s_int <= int_B(10);
            ELSIF (q = '1011') THEN
                    s_int <= int_B(11);
            ELSIF (q = '1100') THEN
                    s_int <= int_B(12);
            ELSIF (q = '1101') THEN
                    s_int <= int_B(13);
            ELSIF (q = '1110') THEN
                    s_int <= int_B(14);
            ELSE
                    s_int <= int_B(15);
            END IF;
  END PROCESS output;
END BEHAVIOR;
```

1.8.2 Multiprocessing Interrupts

While the Tsi107 supports two processors on the processor bus, the EPIC interrupt controller supports only one CPU (and one interrupt output, INT). This architecture is acceptable for non-SMP use as long as one CPU can be dedicated to processing all interrupts. It is basically not possible to achieve SMP (symmetric multiprocessing) if the EPIC unit is involved, since the second processor is essentially relegated to co-processor status. To get around these limitations, the designer can:

- Use an external OpenPIC to route Tsi107 or PCI interrupts to either processor.
- Use an external interrupt controller to collect non-Tsi107 interrupts for the second processor.
- Use the $\overline{\text{INTA}}$ to interrupt the second Tsi107.
- Connect the Tsi107 $\overline{\text{INT}}$ output to both CPUs.

The first two methods are straightforward engineering, and will not be covered further. The third is somewhat different: it uses the capability of the I2O message unit to allow the first processor to interrupt the second processor. This is shown in Figure 20, which also shows the flow of an interrupt from PCI to a second processor.

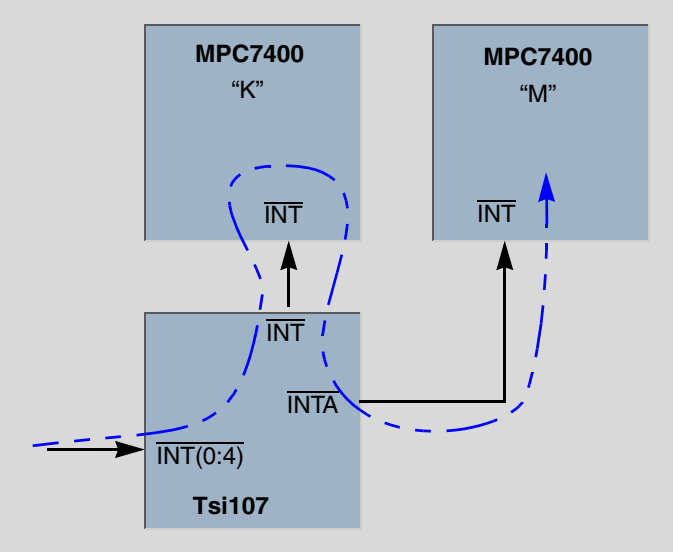

Figure 20. Tsi107 Multiprocessing Logic

The sequence of interrupt handling is as follows:

1. Tsi107 received interrupt (from PCI or internal sources)
2. Tsi107 forwards interrupt to CPU “A”
3. CPU “A” handles interrupt, clears interrupt at the source
4. If interrupt is for CPU “A”, continue in interrupt handler.
5. If interrupt is for CPU “B”, sets $\overline{\text{INTA}}$ output in I^2O message unit
6. Return from exception

Note that this method requires more effort on the part of CPU “A” than on CPU “B”. CPU “A” must actually perform part of the typical interrupt handling process, sufficient to clear the interrupt source. If it did not do this, the interrupt would likely immediately cause another interrupt exception. So CPU “A” must perform at least part of every interrupt exception. CPU “B” is relegated to handling the data processing side.

The fourth method is a variation of this one, but requires even more careful attention to software design. If the $\overline{\text{INT}}$ signal from the Tsi107 is connected to both processors, both processors must handle every interrupt. To do this, the software must determine (based upon its CPU number, via the EPIC unit) whether to handle the interrupt or to ignore it while the other CPU. The determination of CPU number and division of labor need be done only during startup. The limitation of this method is that a CPU will suffer needless interrupts and will be idled until the other processor handles the exception (and re-enables interrupt handling).

Whichever of these methods are chosen depends upon the system requirements. For maximum performance and flexibility, or for true SMP architectures, an external OpenPIC interrupt controller will be required. If some software overhead can be tolerated, a glueless non-symmetric MP system can be designed with one of the above approaches.

1.9 I/O Interfacing

The Tsi107 has four ROM chip selects, RCS(0:3), which may be used to attach ROM, flash memory or general-purpose I/O to the memory interface. RCS0 is reserved for initialization code fetched after HRESET is deasserted, but the others may be used for other code or for memory-mapped I/O devices. There are some limitations of the memory controller that govern how the RCS pins may be used:

- No I/O device can be wider than the SDRAM/ROM bus width
- If I/O devices are smaller than the SDRAM/ROM bus width, software alignment must be used
- Only RCS0 and RCS1 support a special byte-wide access mode
- All memory and I/O devices have the same (programmable) timing

Using the RCS signals to control I/O is fairly straightforward as long as the devices can fit within these limitations. If the devices requires individual timing, or asynchronous timing controls, the local-bus slave interface is more suitable and flexible.

Figure 21 shows an example system with both flash memory and I/O connected to the memory controller of the Tsi107. As long as the device has a typical memory-type interface (chip-select, output-enable and write-enable) it should be possible to connect it to an Tsi107.

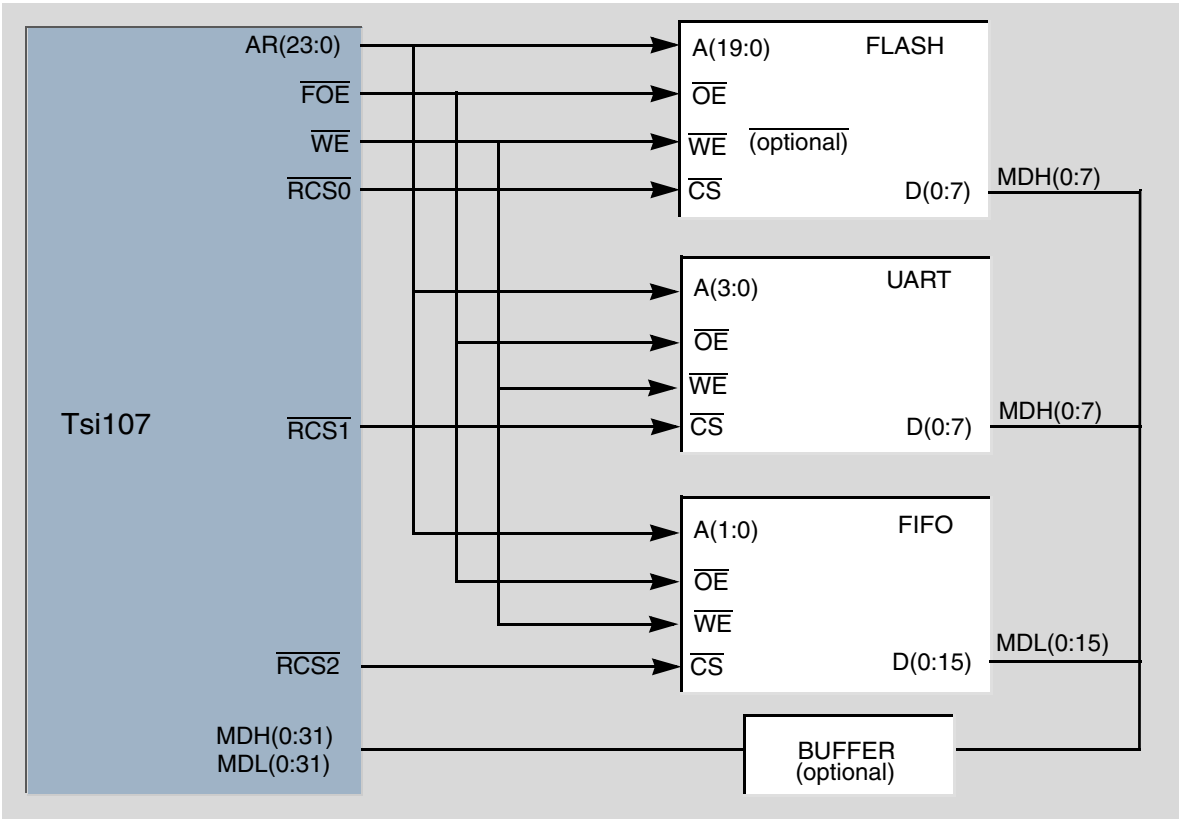

Figure 21. Using RCS pins for I/O

When I/O devices are smaller than the programmed bus width, as with an 8-bit UART on the 32- or 64-bit RCS2 space, software must align reads and writes to account for the byte lanes which are not used. Since the Tsi107 expects the device to be 64-bits, it does not adjust the addresses and does not provide byte lane

write controls (as would the SDRAM interface), so software must compensate by adjusting addresses used. Figure 22 shows an example of byte lane usage.

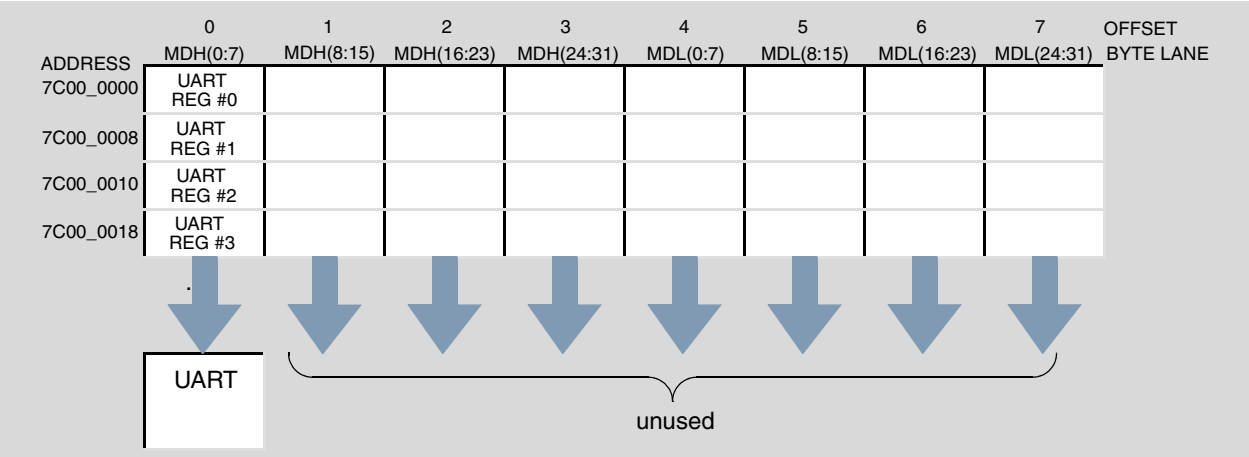

Figure 22. Reduced Bus Width for I/O

Assuming that the UART is a typical PC16550-compatible device, it has 8 sequentially-addressable registers. When this device is attached to the most-significant bits of the memory data bus, MDH(0:7), the 8-bit registers will appear every eight locations in memory, as shown in Table 11.

Table 11. Example UART Address Mapping

UART Register	UART Address	Tsi107 I/O Address
RBR / THR	0	0x7C00_0000
IER	1	0x7C00_0008
IIR / FCR	2	0x7C00_0010
LCR	3	0x7C00_0018
MCR	4	0x7C00_0020
LSR	5	0x7C00_0028
MSR	6	0x7C00_0030
SCR	7	0x7C00_0038

NOTE: Assuming 64-bit $\overline{\text{RCS2}}$ ROM width

The important thing to notice is that the Tsi107 still performs 64-bit reads and writes to those locations. When the Tsi107 reads from the UART, it will latch in 8 bits of data on MDH(0:7) and 56 bits of random data from MDH(8:31)+MDL(0:31). As long as byte read instructions are used, anything on the unused byte lanes will be ignored. For write instructions, the Tsi107 expects that the I/O device will not monitor unused byte lanes; for this reason, I/O devices cannot share chip selects by using different byte lanes.

In the above example, if the UART had been connected to the least-significant bits of the memory data bus, the addresses would have required an offset to select the proper byte lane. For MDL(24:31), the offset is 7, so the sequential UART addresses would be 0x7C00_0007, 0x7C00_000F, 0x7C00_0017, etc. For

high-speed or highly-loaded systems, this technique can be used to minimize loading effects which would occur if every I/O device (as well as memory) is attached to DH(0:7).

1.9.1 Adjusting I/O Timings

The Tsi107 controls the timing for access to ROM and I/O devices using the MCCR1.ROMFAL bits, which sets the number of bus clock cycles needed to perform read and write cycles to devices. Since there is only one register setting, all devices on the ROM controller must share the exact same timing for all accesses. If a slow flash is used, say 150 ns, then any I/O devices will also share a 150 ns access time.

Since the MCCR1.ROMFAL settings may be changed at any time, there are three methods to compensate for this problem:

- Copy program code to SDRAM and run from SDRAM; permanently speed up MCCR1.ROMFAL
- Run or copy program code from PCI; permanently speed up MCCR1.ROMFAL
- Dynamically change MCCR1.ROMFAL

The first two solutions are simplest, and basically involve cases where a system can avoid accessing code or data in the ROM after system initialization has completed. In these cases, there is no need to retain a slow MCCR1.ROMFAL setting since the slow ROM/Flash devices are no longer accessed. In such cases, the software can simply change MCCR1.ROMFAL shortly after the last access to the ROM has completed.

The last solution is fairly complex, requiring tightly-controlled software assistance; this approach would be required in situations where, due to size or cost limitations, software must run partially or entirely from Flash/ROM. Since changes to the MCCR1.ROMFAL setting can affect the ability to read instructions from the ROM, the software must ensure that code is resident in cache. Once cache-bound, the software can increase the speed of I/O accesses, perform the access (or multiple accesses), and then restore the slower MCCR1.ROMFAL settings. In essence, the software changes the MCCR1.ROMFAL settings every time it accesses the I/O device. Due to the complexity of this method, it is typically only used for critical I/O, but it is particularly well-suited to interrupt drivers, where access to I/O is encapsulated.

1.9.2 PortX

Despite the name, the PortX facility is not a separate, general-purpose I/O subsystem, it is intimately tied to the ROM memory controller. PortX is best thought of as a programmable address strobe signal ($\overline{AS}$) which can be asserted in the middle of all ROM I/O cycles. This is only enhancement to I/O interfacing which PortX brings; however, it is quite useful to strobe addresses into a multiplexed address/data device, and also to alter the I/O signals for devices which have particular restrictions on the relationships between $\overline{RCSx}$, $\overline{FOE}$, and $\overline{WE}$. Figure 23 shows an example of the $\overline{AS}$ signal in a PortX access cycle.

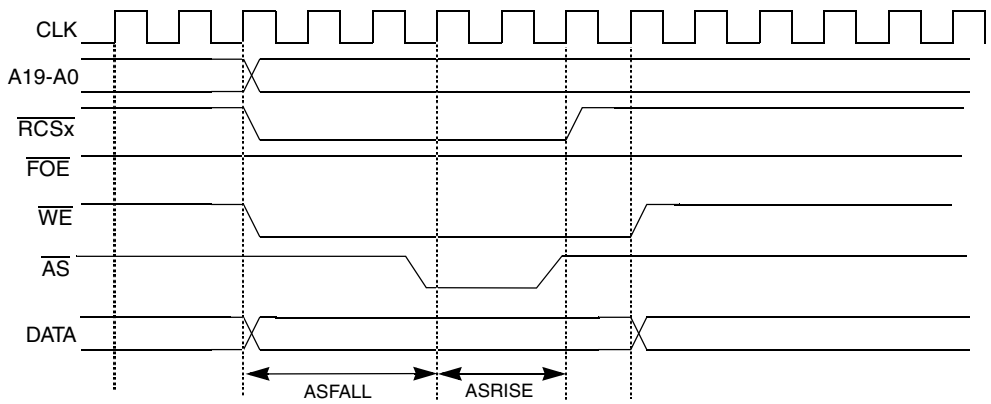

Figure 23. PortX Access Timing

As seen in Figure 24, the $\overline{WE}$ signal extends beyond the $\overline{CS}$ assertion interval. For devices which require particular relationships between $\overline{CS}$ and $\overline{WE}$, $\overline{AS}$ can be used with simple logic to adjust the relationships for such I/O or Flash/ROM devices. Examples are shown in Figure 24.

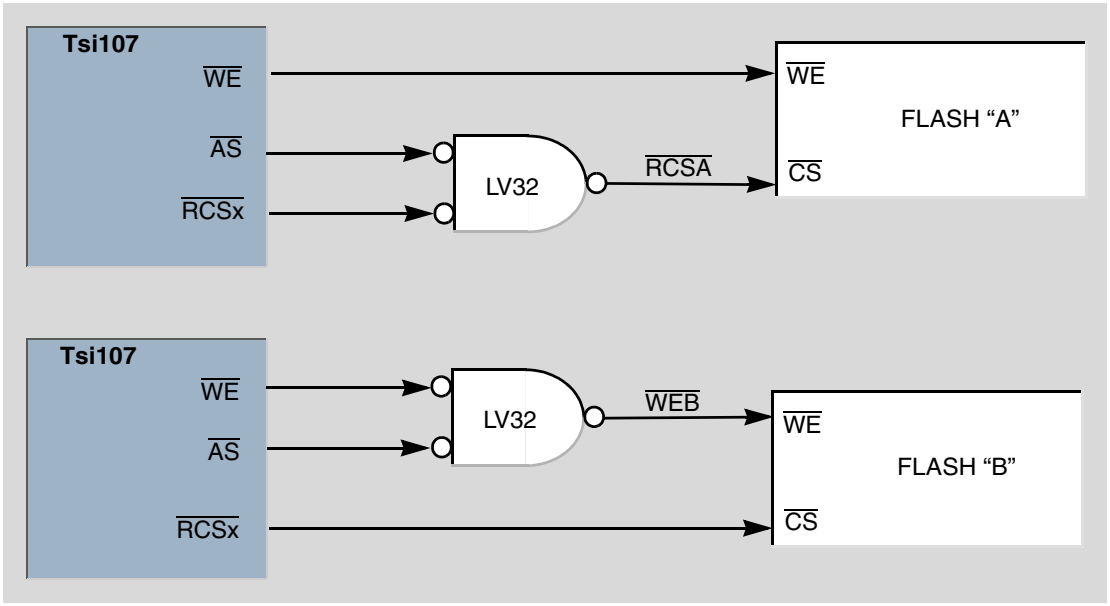

Figure 24. Tsi107 PortX Signal Adjustments

Using the logic shown above, the ROM/I/O timing can be adjusted as shown in Figure 25.

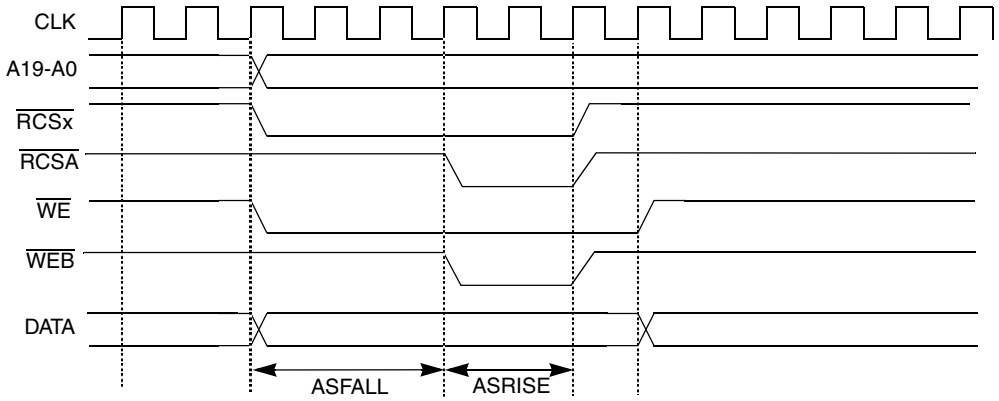

Figure 25. PortX-Modified I/O Timing

This logic is often useful with flash memory and other devices which require extra time after the chip-select for recovery, or those which will not allow $\overline{WE}$ asserted concurrently with $\overline{CS}$. Note, though, that since $\overline{AS}$ timing values must be programmed into the MCCR registers, this solution is not suitable for code that must be available immediately after reset (i.e., startup code in $\overline{RCS0}$).

1.10 Reset

A system using the Tsi107 must provide it with a proper reset signal; the Tsi107 requires a reset pulse that is at least 100 µs (for PLL stabilization) plus 255 PCI bus clocks in duration. For systems connected to the PCI bus, the PCIRST# signal easily meets this restriction since PCI guarantees a reset assertion period of 1 ms.

To simplify system design, the Tsi107 also has the capability of asserting a reset signal ($\overline{HRESET\_CPU}$) to the host processor, as shown in Figure 26. In this configuration, the Tsi107 asserts the processor $\overline{HRESET}$ pin when the PCIRST# signal is asserted. The Tsi107 will keep $\overline{HRESET\_CPU}$ asserted for 2^{17} processor bus clocks after its $\overline{HRESET}$ pin is released; this extra time allows the Tsi107's clock generator DLL and the processor's PLL to stabilize in sequence. After this additional time has elapsed, all clocks in the system should be reliable and ready for reset to be released.

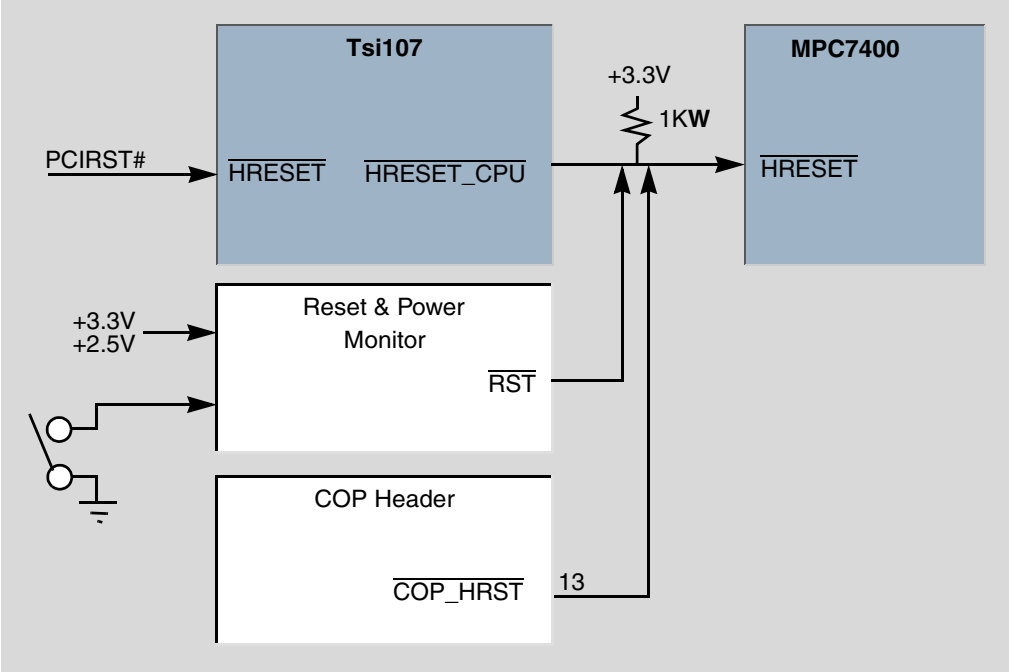

Figure 26. Tsi107 Reset Logic

Since the $\overline{\text{HRESET\_CPU}}$ pin is an open-drain output, it requires a weak pull-up (1K to 10K in value) to provide a valid signal when not asserted. However, since it is open-drain, it can be “wire-OR” connected (with no additional logic) with other optional reset signals, such as those from a COP debugger port, a power-supply monitor, a watchdog timer, and a reset switch (assuming that the latter devices are also open-drain).

1.10.1 Self-Reset

Using the EPIC (embedded programmable interrupt controller) it is possible for software to restart itself. The usual connections from the EPIC to the processor allow it to assert the $\overline{\text{SRESET}}$ signal; this allows software to restart cleanly but it does not force external hardware to initialize itself. If the software needs to force a complete system initialization, it can use the $\overline{\text{SRESET}}$ signal in a different manner as shown in Figure 27.

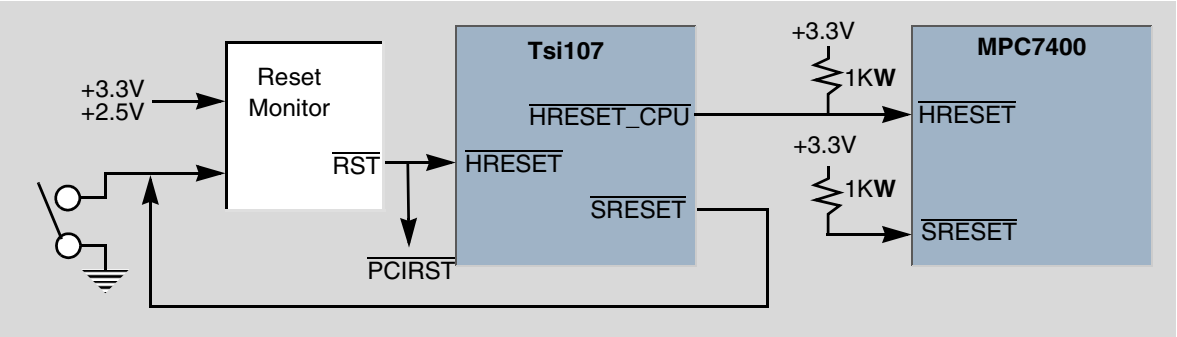

Figure 27. Tsi107 Self-Hard-Reset Connections

As shown in Figure 27, the $\overline{\text{SRESET}}$ output in the EPIC unit is used to drive system reset logic, assuring that all other devices see a general reset signal. Note that when the reset controller asserts $\overline{\text{HRESET}}$ to the Tsi107, the $\overline{\text{SRESET}}$ output will be cleared. To insure a stable reset system, the $\overline{\text{SRESET}}$ output should be connected to a switch debounced input of the reset controller, so that $\overline{\text{HRESET}}$ will continue to be asserted for the required amount of time, even when the $\overline{\text{SRESET}}$ signal is deasserted.

1.11 Packaging

The Tsi107 uses a 25x25 TBGA package, which while much larger than the Tsi106 package, reduces overall board space by incorporating the memory data bus buffers, clock drivers, and I/O decoders.

Table 12. Tsi106 vs. Tsi107 Board Space Usage

Object	Tsi106	Tsi107
Tsi10x	525 mm^2	1089 mm^2
Data Bus Buffers	1300 mm^2	0
Clock Driver	400 mm^2	0
I/O Decoder PAL	400 mm^2	0
Total	2275 mm^2	1089 mm^2

In addition, the Tsi107 uses annular ring of pads instead of a solid grid, so vias and escapes can be easily placed within the chip area to facilitate PCB routing. For boards using double-sided components, the Tsi107 core supply, PLL filters and power/ground bypass capacitors can be placed within the center ring (on the bottom of the PCB) for even greater space savings as well as superior electrical characteristics.

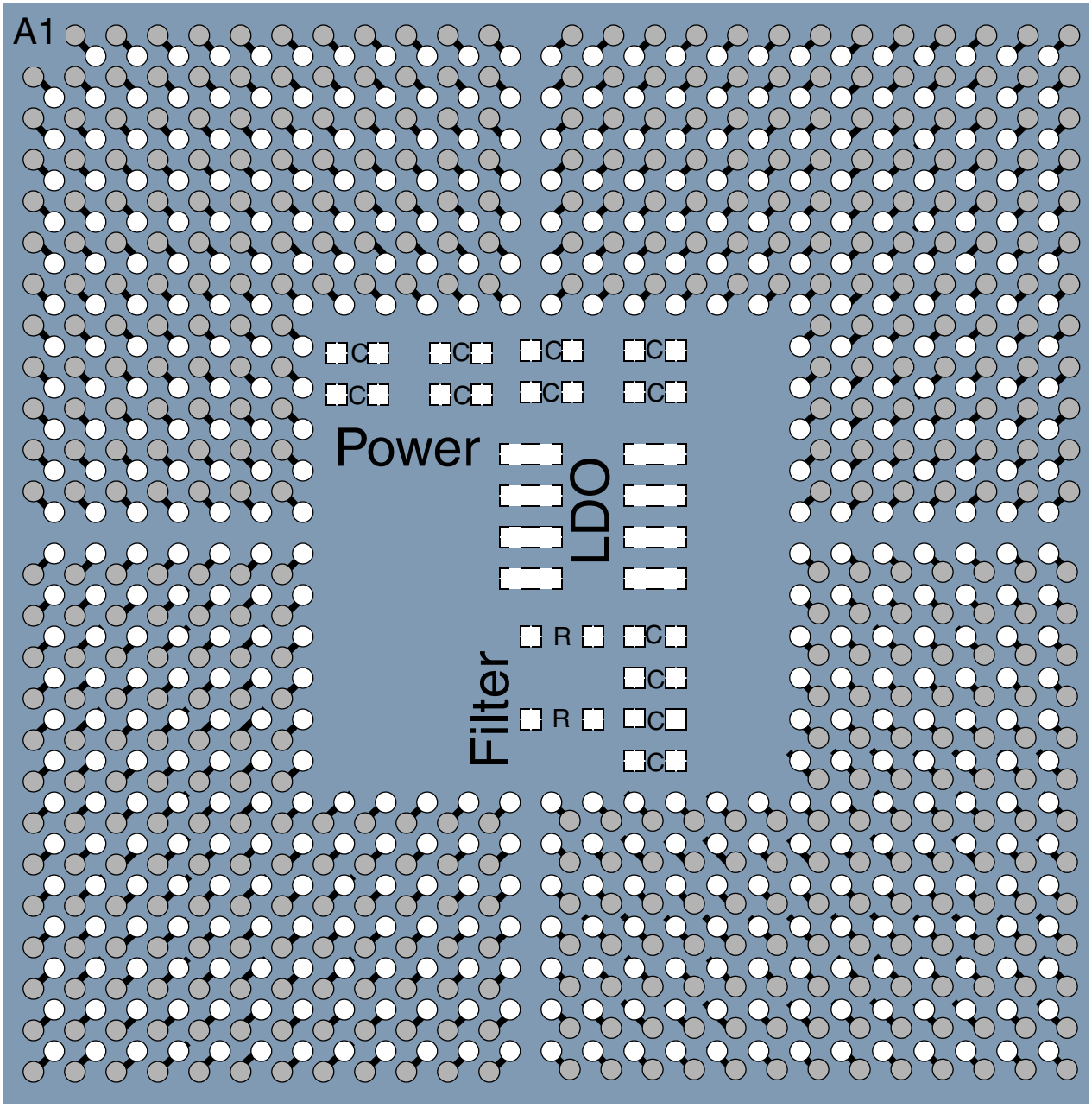

Figure 28. Tsi107 Escape Pattern With Components

Note: In Figure 28, the Tsi107 pad pattern is divided into four quadrants. Each BGA pad (typically 0.028 inches) connects diagonally to a via between the pads, for escape to lower layers. Of course, not all vias will be necessary, and since there are no pads on the bottom layer, it is fairly easy to escape using only four signal planes.

1.12 References

The reference materials shown in Table 13 may be useful to the designer.

Table 13. Reference Documentation

Description	Author	Document
High-Speed Digital Design: A Handbook of Black Magic	Howard Johnson and Martin Graham	Prentice-Hall ISBN 0-133-95724-1
Foundations of Microstrip Circuit Design	T. C. Edwards	John Wiley, NY, 1981
SDRAM System Design using the Tsi106	Gary Milliorn	www.tundra.com/tsi106
Designing a Local Bus Slave I/O Controller	Gary Milliorn	www.tundra.com/tsi107