Freescale Semiconductor

AN1271
(Freescale Order Number)

12/95

PowerPC™

Application Note

PowerPC™ 60x Microprocessor to AD1848 CODEC Interface

This document describes how to interface the Analog Devices SoundPort® Stereo CODEC (AD1848) to the PowerPC 60x local bus. The AD1848 integrates key audio data conversion and control functions into a single integrated circuit. It is intended to provide a complete, single-chip audio solution for business audio and multimedia applications and provides a direct interface to the Industry Standard Architecture (ISA) AT bus. The AD1848, which supports Microsoft Windows Sound System™, is currently widely used throughout the personal computer industry.

In this document, the term "60x" is used to denote a 32-bit microprocessor from the PowerPC architecture family. PowerPC 60x processors implement the PowerPC architecture as it is specified for 32-bit addressing, which provides 32-bit effective (logical) addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits (single-precision or double-precision).

The PowerPC 60x family of microprocessors are high performance, RISC processors that conform to the PowerPC architectural specifications. With on-chip caches, superscalar operation, a powerful instruction set, and high-operating frequencies, this family of general-purpose microprocessors can be used to perform signal processing functions. In addition, PowerPC 60x microprocessors also have very similar system buses allowing the design of generic interface logic. The local bus to CODEC interface (LCI), can be used with the PowerPC 601™, PowerPC 603™, and PowerPC 604™ microprocessors.

Figure 1 provides a block diagram of the PowerPC 60x bus to AD1848 interface.

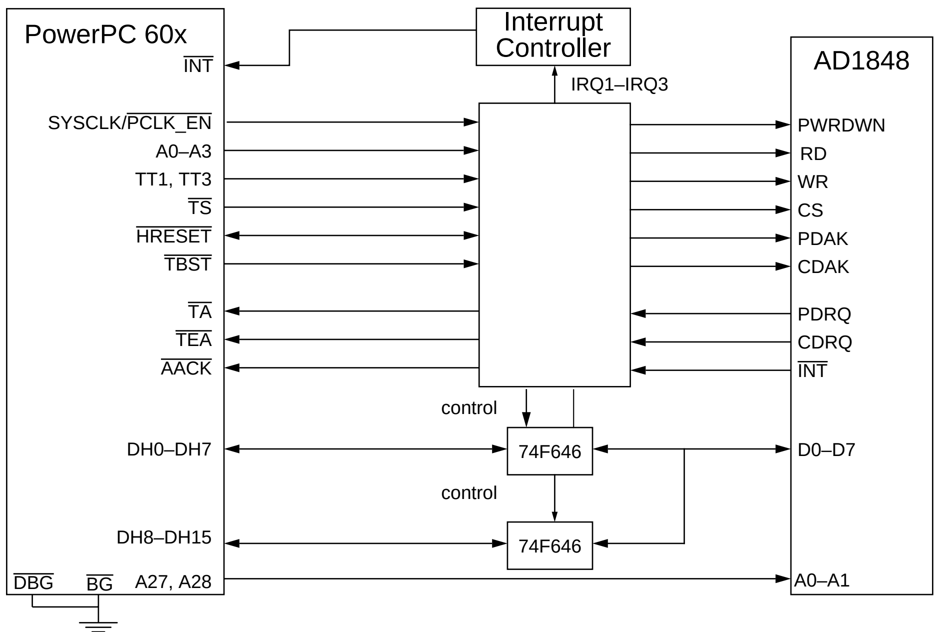


Figure 1. PowerPC 60x Bus to AD1848 Interface Block Diagram

The AD1848's system interface allows a near-glueless interface solution for ISA-bus based systems. There are however, a number of market areas (for example, games and video-on-demand) in which non-ISA based systems are required to deliver high quality audio. This document describes one possible PowerPC-based solution for these market areas.

Part 1 Interface Issues

The AD1848 CODEC features two sets of handshake lines that enable direct-memory access (DMA) data transfers with the host computer bus. In addition, the CODEC supports programmed I/O (PIO) cycles both for control register accesses and also for data transfers in systems that lack DMA capability. An advantage of the PIO-cycle-only interface is its simplicity; a disadvantage is that software polling of the CODEC is required to perform data transfers. Although the interface described here is intended for PowerPC systems with no local bus DMA controller, it is designed to work with the CODEC's dual-channel DMA mode. The LCI supports this configuration by converting each assertion of an AD1848 DMA request signal into an interrupt request and by translating PowerPC accesses to specific memory areas into AD1848 DMA cycles. When implemented efficiently, this DMA emulation reduces the bus bandwidth required to service the CODEC. This represents an important feature of a local-bus design.

In addition to the CODEC's CDRQ and PDRQ lines, the AD1848K's $\overline{\text{INT}}$ output can also be used to generate an interrupt to the PowerPC 60x processor. As the PowerPC devices supported by the LCI design only have one general-purpose interrupt input, and in this design the AD1848 provides three sources, an interrupt controller is required to combine the signals and to enable the system to determine the cause of the interrupt.

A number of different types of DMA transfers are supported by the AD1848, ranging from 8-bit mono DMA cycles to 16-bit stereo DMA cycles. Note that, to implement this interface using one MACH210 device, 8-bit mono transfers are not supported. The limiting factor here is the number of product terms generated by the master state machine. If however, a suitable logic device is used, an 8-bit mono capability may be easily incorporated. The state machine shown in Figure 2 indicates (by dashed lines) the extra transition paths required to support 8-bit mono DMA cycles. The single-channel DMA mode allows the AD1848K to be used in systems with only a single DMA channel. The dual-channel DMA mode has been supported in preference to single-channel DMA mode as the former facilitates simultaneous playback and capture and is virtually as simple to implement.

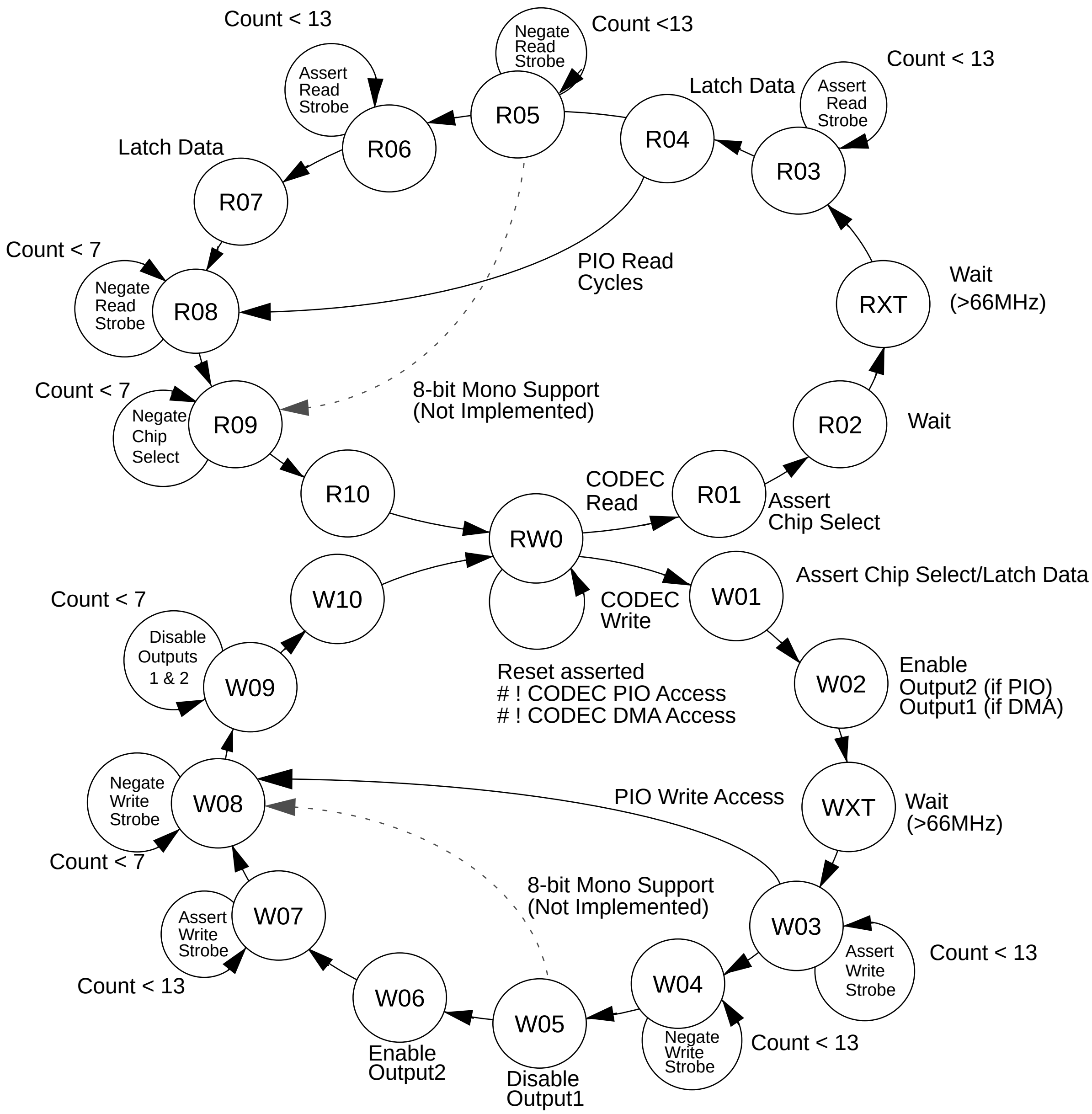


Figure 2. State Machine CODEC_sm

The AD1848 is designed for little-endian systems in which the least-significant byte of a multi-byte data item (that is, the byte occupying the lowest memory address) is transferred first. In addition to their default big-endian operating mode, PowerPC microprocessors also support little-endian mode. Since this little-endian mode changes the physical address of data items rather than their internal byte order, the LCI can actually support both PowerPC byte-ordering conventions (the address modification depends on the width of the addressed item. Only the three least-significant address lines are affected. Addresses shown are based on access data width specified in Table 1). However, since the physical address space occupied by the CODEC is fixed by hardware, different logical addresses are required for big- and little-endian modes. Table 1 shows the physical addresses of AD1848 registers, DMA space, and LCI-control areas. It also provides the logical addresses (assuming PowerPC 60x address translation is disabled) that will generate the appropriate physical addresses for both modes. Since, in little-endian mode, the physical address output by a PowerPC 60x microprocessor depends on the width of the addressed data item, all accesses must conform to the widths shown in the final column of the table.

The addresses shown in Table 1 reveal, through the number of unconditional terms, that the majority of PowerPC 60x address lines are not used in the selection of CODEC registers, DMA space or control locations. For example, any access in the region $6000 0000 to $6FFF FFFF will cause the AD1848's PWRDWN line to be asserted (driven low). PWRDWN will also be asserted if $\overline{\text{HRESET}}$ is asserted. In both of these cases, the PWRDWN signal will only be negated by an access to the region $7000 0000 to $7FFF FFFF.

Table 1 also shows that microprocessor accesses to h4xxx xxxx are converted into AD1848 DMA cycles. Half-word loads from h4xxx xx00 emulate stereo DMA read/capture cycles and half-word stores emulate stereo DMA write/playback cycles. As can be seen in Figure 4, the CODEC and processor data buses are connected via two 8-bit transceiver/registers. So, in order to emulate a 16-bit stereo DMA cycle, in which 32 bits of data are transferred, the processor must perform two accesses. From the CODEC's perspective, these two accesses must be sequential. If the processor attempts another PIO/DMA cycle in the interval between two 16-bit stereo accesses, the LCI terminates the cycle by asserting $\overline{\text{TEA}}$. Furthermore, although the state machine implementation shown in Figure 2 does not support 8-bit mono DMA cycles, the design may be easily modified to support these. In this case, Table 1 indicates the appropriate address and data width for the accesses.

Table 1. Address Map

CODEC Registers/Interface Control Locations	PowerPC 60x Physical Address	Big-Endian Logical Address	Little-Endian Logical Address	Access Data Width
Indexed address register	$5xxx xx00	$5xxx xx00	$5xxx xx07	Byte
Indexed data register	$5xxx xx08	$5xxx xx08	$5xxx xx0F	Byte
Status register	$5xxx xx10	$5xxx xx10	$5xxx xx17	Byte
PIO data register	$5xxx xx18	$5xxx xx18	$5xxx xx1F	Byte
CODEC DMA(8-/16-bit stereo)	$4xxx xxx0	$4xxx xxx0	$4xxx xxx6	Half-word
CODEC DMA(8-bit mono)	$4xxx xxx1	$4xxx xxx1	$4xxx xxx6	Byte
CODEC power down (assert)	$6xxx xxxx	$6xxx xxxx	$6xxx xxxx	Any width
CODEC power down (negate)	$7xxx xxxx	$7xxx xxxx	$7xxx xxxx	Any width

Part 2 Frequency of Operation

Since the LCI is controlled by state machines clocked at the PowerPC 60x's bus frequency, it can work over a wide range of the microprocessor's clock frequencies. In order to generate signals for the relatively slower CODEC interface, the LCI employs the state machine CODEC_time (see Figure 3) to serve as a counter/timer. CODEC timing intervals are generated by counting an appropriate number of PowerPC bus clocks. Since changing the microprocessor's bus clock frequency has a proportional effect on the timing generated by the LCI, longer counts are generally required for higher processor speeds. However, since the counter is actually a PLD-based state machine, the count values cannot be changed dynamically. The maximum frequency of operation of the LCI is therefore limited by values set at PLD-compile time. If the LCI has to run over a range of frequencies, the count values used must reflect the maximum operating speed required. (The interface will still work at lower frequencies but will lose some efficiency.) The maximum frequency of the LCI is also determined by the type of data bus transceivers used, the speed of the MACH210, and the PowerPC 60x family member. If IDT74FCT162646 transceivers are used with a 7-nanosecond (ns) MACH210, the LCI can operate up to 66 MHz bus clock frequency with a 601 and up to 50 MHz with a 603.

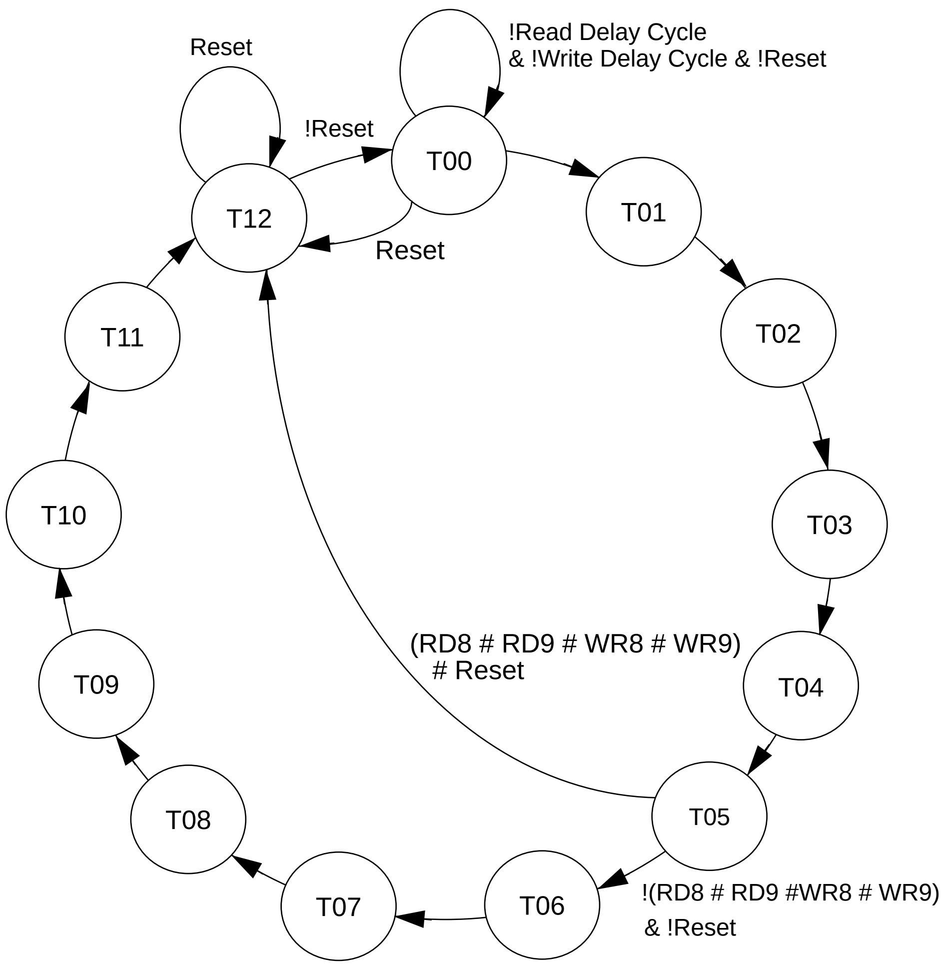


Figure 3. State Machine CODEC_Time

Table 2 shows the relationship between the two state machines, the CODEC and buffer control lines and AD1848 timing parameters.

Table 2. CODEC & Latch Control vs. Control Logic

STATE [1]	CODEC ACTION	CDAK	PDAK	RD	WR	CS	LATCH ACTION	DIR	G1	CP1	G2	CP2	DURATION	PARAMETER [2]
PIO Read														
RW0	—	H[3]	H	H	H	H	Idle	L[3]	H	L	H	L		
R01	Assert CS	H	H	H	H	L	DIR to high	H	H	L	H	L		tCSSU
R02	Wait	H	H	H	H	L	—	H	H	L	H	L		
RXT	No action	H	H	H	H	L	No action	H	H	L	H	L		
R03	Assert Read	H	H	L	H	L	—	H	H	L	H	L	13 Clocks	tSTW
R04	—	H	H	L	H	L	Store 2	H	H	L	H	H		
R08	Negate Read	H	H	H	H	L	Enable 1 & 2 [4,5]	H	L	L	L	L	7 Clocks	tCSHD/tSUDK2
R09	Negate CS	H	H	H	H	H	Disable 1 & 2[4,5]	H	H	L	H	L	7 Clocks	tCSHD
R10	—	H	H	H	H	H	—	H	H	L	H	L		
DMA Read														
RW0	—	H(L) [6]	H	H	H	H	Idle	L	H	L	H	L		
R01	Assert CDAK	L	H	H	H	H	Change DIR	H	H	L	H	L		
R02	Wait	L	H	H	H	H	—	H	H	L	H	L		tDKSU
RXT	No action	L	H	H	H	H	No action	H	H	L	H	L		tDKSU@>66 MHz
R03	Assert Read	L	H	L	H	H	—	H	H	L	H	L	13 Clocks	tSTW
R04	—	L	H	L	H	H	Store 1 [7]	H	H	H	H	L		
R05	Negate Read	L	H	H	H	H	—	H	H	L	H	L	13 Clocks	tBWND
R06	Assert Read	L	H	L	H	H	—	H	H	L	H	L	13 Clocks	tSTW
R07	—	L	H	L	H	H	Store 2 [7]	H	H	L	H	H		
R08	Negate Read	L	H	H	H	H	Enable 1 & 2	H	L	L	L	L	7 Clocks	tDKHDb
R09	Negate CDAK	H(L)	H	H	H	H	Disable 1 & 2	H	H	L	H	L	7 Clocks	tSUDK1/tBWND
R10	—	H(L)	H	H	H	H	—	H	H	L	H	L		
PIO Write														
RW0	—	H	H	H	H	H	—	L	H	L	H	L		
W01	Assert CS	H	H	H	H	L	Store 1 & 2	L	H	H	H	H		tCSSU
W02	Wait	H	H	H	H	L	Enable 2	L	H	L	L	L		
WXT	No action	H	H	H	H	L	No action	L	H	L	L	L		
W03	Assert Write	H	H	H	L	L	—	L	H	L	L	L	13 Clocks	tSTW
W08	Negate Write	H	H	H	H	L	—	L	H	L	L	L	7 Clocks	tCSHD/TSUDK2
W09	Negate CS	H	H	H	H	H	Disable 2	L	H	L	H	L	7 Clocks	tCSHD
W10	—	H	H	H	H	H	—	L	H	L	H	L		

Table 2. CODEC & Latch Control vs. Control Logic (Continued)

STATE [1]	CODEC ACTION	CDAK	PDAK	RD	WR	CS	LATCH ACTION	DIR	G1	CP1	G2	CP2	DURATION	PARAMETER [2]
DMA Write														
RW0	Idle	H	H(L)	H	H	H	Idle	L	H	L	H	L		
W01	Assert PDAK	H	L	H	H	H	Store 1 & 2	L	H	H	H	H		
W02	Wait	H	L	H	H	H	Enable 1	L	L	L	H	L		
WXT	No action	H	L	H	H	H	No action	L	L	L	H	L		
W03	Assert Write	H	L	H	L	H	—	L	L	L	H	L	13 Clocks	tSTW
W04	Negate Write	H	L	H	H	H	—	L	L	L	H	L	13 Clocks	tBWND
W05	—	H	L	H	H	H	Disable 1	L	H	L	H	L		
W06	—	H	L	H	H	H	Enable 2	L	H	L	L	L		
W07	Assert Write	H	L	H	L	H	—	L	H	L	L	L	13 Clocks	tSTW
W08	Negate Write	H	L	H	H	H	—	L	H	L	L	L	7 Clocks	tDKHDb
W09	Negate PDAK	H	H(L)	H	H	H	Disable 2	L	H	L	H	L	7 Clocks	tSUDK1/tBWND
W10	—	H	H(L)	H	H	H	—	L	H	L	H	L		

Notes:

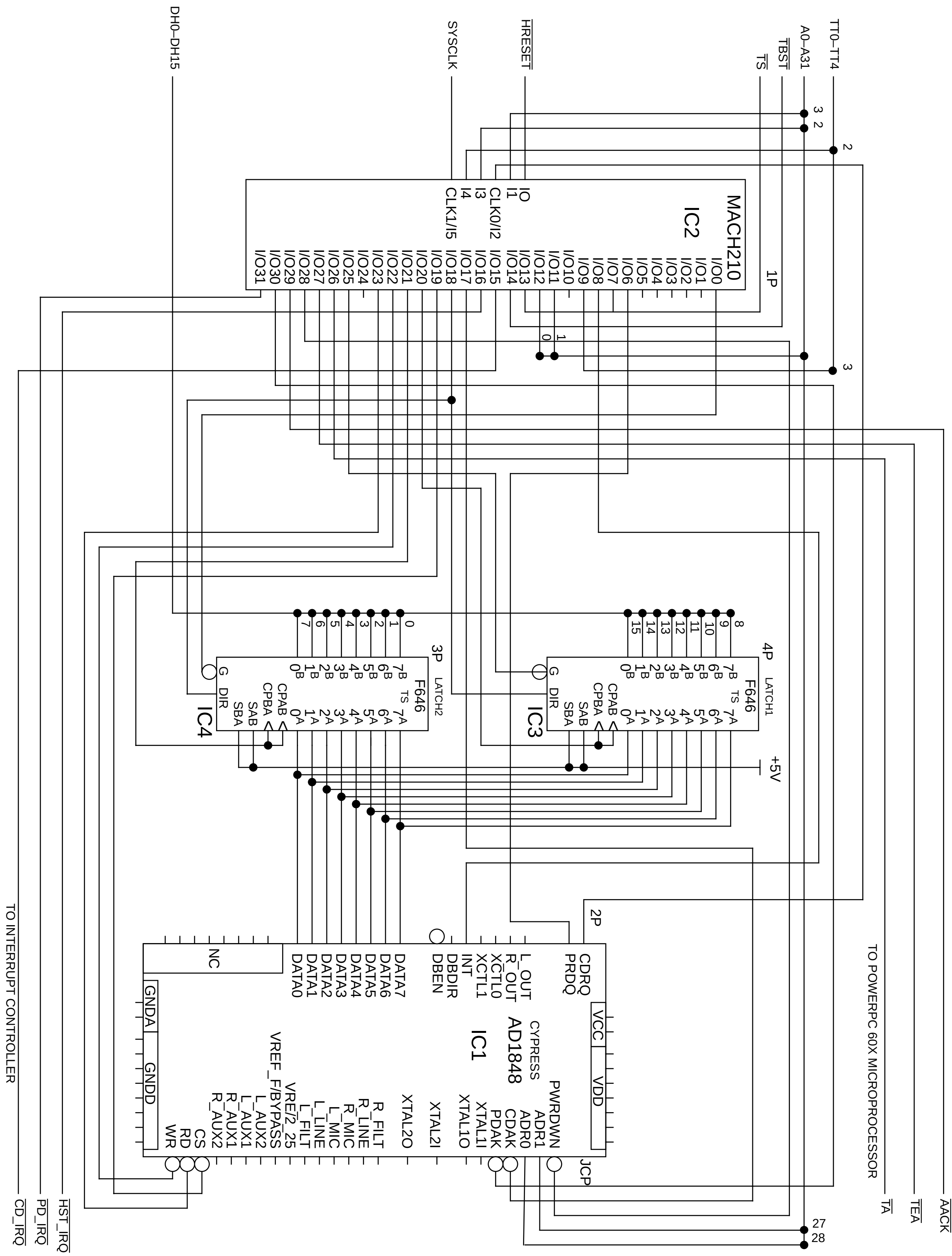
1. State refers to the state before rising edge of clock, H/L indicates that the output follows the clock rising edge
2. Parameter refers to AD1848 AC timing parameters
3. H indicates digital 1, L indicates digital 0
4. Enable/Disable 1 indicates enable/disable outputs of 646 latch 1 [IC3 in schematics]
5. Enable/Disable 2 indicates enable/disable outputs of 646 latch 2 [IC4 in schematics]
6. (L) indicates alternative state dependent on previous CODEC access
7. Store 1/2 indicates store data on 646 latch 1/2 [IC3/4]

Part 3 Design Description

The interface control logic, as shown in Figure 4, is implemented using one MACH210 device (refer to Part 4, “IC2 Equations List”). This device implements all the logic required to convert PowerPC 60x bus transactions into AD1848-compatible PIO and DMA cycles. It decodes all CODEC accesses, drives CODEC and data latch inputs and generates an appropriate response to the PowerPC 60x microprocessor on the $\overline{\text{AACK}}$, $\overline{\text{TA}}$, and $\overline{\text{TEA}}$ lines. The design features two state machines as shown in Figure 3. CODEC_time is used to generate AD1848 timing intervals from a fast PowerPC 60x SYSCLK and CODEC_sm, shown in Figure 2, provides the main control for all CODEC transactions.

Figure 4 is a schematic which shows the connectivity required to implement the interface. Based largely on the state of CODEC_sm, IC2 drives AD1848 control lines to affect either PIO or DMA transactions. IC2 also drives transceiver control lines to ensure that data is read and written correctly. Two 74F646 octal transceivers/registers (IC3, IC4) or a double density equivalent such as the IDT74FCT162646 can be used. On reset, the MACH210 is programmed to assert the CODEC’s PWRDWN line; this can be negated (and asserted) by a PowerPC 60x access to a specific area of the memory map. Finally, the MACH device is used to generate two active low interrupt signals (IRQ2, IRQ3) from the IC1’s PDRQ and CDRQ outputs. The remaining circuitry on the schematics is made up of either general-support circuitry for the AD1848 (IC1), or buffering for a microphone input, two line outputs, and a line input (all stereo).

Refer to *PowerPC 601 RISC Microprocessors User's Manual* (MPC601UM/AD), *PowerPC 603 RISC Microprocessors User's Manual* (MPC603UM/AD), and *Analogue Devices Parallel-Port 16-Bit SoundPort® Stereo CODEC AD1848K* data sheet (Rev. 0.1) for more information.



No other circuitry such as power supply connections, or analogue I/O is shown in this schematic.

Figure 4. PowerPC 60x Local Bus to CODEC Interface Schematic

Part 4 IC2 Equations List

```
Name            cmach5;
Partno          p00001;
Date            08/01/95;
Revision        1.0;
Designer        Colin MacDonald;
Company         Freescale Copyright (C);
Assembly        CoreX;
Location        IC1;
Device          Mach210;
FORMAT          f;

/*********************************************************/
/*  DESCRIPTION:                                         */
/*  State Machine for CODEC Control                      */
/*                                                       */
/*********************************************************/
/*  History:                                             */
/*  CMACH2   version for MACH devices                    */
/*  CMACH4   generates own CODEC_sel                     */
/*           Could improve solution further by latching processor */
/*           signals and giving ta earlier on writes     */
/*  CMACH5   uses e_tea to allow coincident aack         */
/*           Improved ta timing a little WR8T5->WR8T0    */
/*********************************************************/

/** Pin Assignments **/
Pin 35 = clk;          /* PowerPC 60x bus clock          */
Pin 2 = !g2;           /* 74F646 Latch1 out enable       */
Pin 18 = a0;           /* 60x address                    */
Pin 17 = a1;           /* 60x address                    */
Pin 32 = a2;           /* 60x address                    */
Pin 11 = a3;           /* 60x address                    */
Pin 33 = tt1;          /* 60x tt line                    */
Pin 15 = tt3;          /* 60x tt line                    */
Pin 10 = !reset;       /* hreset                         */
Pin 14 = int;          /* AD1848 host IRQ (IN)           */
Pin 19 = !ts;          /* transfer start                 */
Pin 20 = !tbst;        /* 60x burst signal               */
Pin 40 = !pwrdwn;      /* CODEC PWRDWN input (OUT)       */
Pin 24 = !hst_irq;     /* INT generated IRQ (OUT)        */
Pin 41 = !aack;        /* address acknowledge (OUT)      */
Pin 39 = !tea;         /* transfer error (OUT)           */
Pin 38 = !ta;          /* transfer acknowledge (OUT)     */
Pin 8  = pdrq;         /* AD1848 PDRQ output             */
Pin 13 = cdrq;         /* AD1848 PDRQ output             */
Pin 27 = !cs_;         /* AD1848 cs input                */
Pin 30 = !wr_;         /* AD1848 wr input                */
Pin 37 = !g1;          /* 74F646 Latch1 out enable       */
Pin 42 = !pdak;        /* AD1848 pdak input              */
Pin 25 = !cdak;        /* AD1848 cdak input              */
Pin 29 = cp2;          /* 74F646 Latch2 clock            */
Pin 28 = cp1;          /* 74F646 Latch1 clock            */
Pin 26 = dir;          /* 74F646 Latch1 dir              */
```

```
Pin 31 = !rd_;              /* AD1848 rd input               */
Pin 43 = !pd_irq;           /* IRQ2 from AD1848 PDRQ         */
Pin 21 = !cd_irq;           /* IRQ3 from AD1848 CDRQ         */

/** Pinnode Assignments **/
Pinnode 66 = time0;         /* time/state                    */
Pinnode 71 = time1;         /* time/state                    */
Pinnode 74 = time3;         /* time/state                    */
Pinnode 69 = time2;         /* time/state                    */
Pinnode 55  = state4;       /* time/state                    */
Pinnode 47  = state0;       /* time/state                    */
Pinnode 51  = state2;       /* time/state                    */
Pinnode 63  = state1;       /* time/state                    */
Pinnode 58  = state3;       /* time/state                    */
Pinnode 75  = !e_ta;        /* early ta                      */
Pinnode 102 = !cdc_sel;     /* CODEC chip select             */
Pinnode 104 = !e_tea;       /* Early tea for central cntl    */

GROUP BLOCK_A = g2;
GROUP BLOCK_D = g1,cdc_sel;

Field CODEC_sm = [state4..0];
$define RW0     'b'00000
$define R01     'b'00001
$define R02     'b'00010
$define R03     'b'00011
$define R04     'b'00100
$define R05     'b'00101
$define R06     'b'00110
$define R07     'b'00111
$define R08     'b'01000
$define R09     'b'01001
$define R10     'b'01010
$define RXT     'b'01011    /* extra wait state for >66MHz bus operation */

$define W01     'b'10001
$define W02     'b'10010
$define W03     'b'10011
$define W04     'b'10100
$define W05     'b'10101
$define W06     'b'10110
$define W07     'b'10111
$define W08     'b'11000
$define W09     'b'11001
$define W10     'b'11010
$define WXT     'b'11011    /* extra wait state for >66MHz bus operation */

Field CODEC_time = [time3..0];
$define T00     'b'0000
$define T01     'b'0001
$define T02     'b'0010
$define T03     'b'0011
$define T04     'b'0100
$define T05     'b'0101
$define T06     'b'0110
```

```
$define T07         'b'0111
$define T08         'b'1000
$define T09         'b'1001
$define T10         'b'1010
$define T11         'b'1011
$define T12         'b'1100

RD1     =   CODEC_sm:R01;
RD2     =   CODEC_sm:R02;
RD3     =   CODEC_sm:R03;
RD4     =   CODEC_sm:R04;
RD5     =   CODEC_sm:R05;
RD6     =   CODEC_sm:R06;
RD7     =   CODEC_sm:R07;
RD8     =   CODEC_sm:R08;
RD9     =   CODEC_sm:R09;
RD10    =   CODEC_sm:R10;
IDLE    =   CODEC_sm:RW0;
XTRD    =   CODEC_sm:RXT;

WR1     =   CODEC_sm:W01;
WR2     =   CODEC_sm:W03;
WR4     =   CODEC_sm:W04;
WR5     =   CODEC_sm:W05;
WR6     =   CODEC_sm:W06;
WR7     =   CODEC_sm:W07;
WR8     =   CODEC_sm:W08;
WR9     =   CODEC_sm:W09;
WRTN    =   CODEC_sm:W10;
XTWR    =   CODEC_sm:WXT;

TM0     =   CODEC_time:T00;
TM3     =   CODEC_time:T03;
TM5     =   CODEC_time:T05;
TM12    =   time3 & time2;
/* TM12 -> TM0 so can don't care time1 & time0 to save terms */

CPU_RD          =   tt1;
CPU_WR          =   !tt1;
MEM_CYCLE       =   tt3;
ADD_ONLY        =   !tt3;
BURST           =   tbst;

DMA             =   cdc_sel & !a2 & !a3;
PIO             =   cdc_sel & !a2 &  a3;
RD_DEL          =   (RD3 # RD5 # RD6 # RD8 # RD9);
WR_DEL          =   (WR3 # WR4 # WR7 # WR8 # WR9);

CODEC_TS        =   !a0 &  a1 & ts;
CDMA_ADD        =   !a2 & !a3 &  tt1;
PDMA_ADD        =   !a2 & !a3 & !tt1;
PIO_ADD         =   !a2 &  a3;
PDN_ADD         =   a2 & !a3;
PUP_ADD         =   a2 &  a3;
CDMA_START      =   CODEC_TS & !ADD_ONLY & !BURST & CDMA_ADD;
```

```
PDMA_START    =    CODEC_TS & !ADD_ONLY & !BURST & PDMA_ADD;
PIO_START     =    CODEC_TS & !ADD_ONLY & !BURST & PIO_ADD;
PDN_START     =    CODEC_TS & !ADD_ONLY & !BURST & PDN_ADD;
PUP_START     =    CODEC_TS & !ADD_ONLY & !BURST & PUP_ADD;

/* Active DMA - not enough prod terms to implement hold-off */
/* Need to be careful with S/W !                             */

DMA3    =   !a3 & !reset;
PIO3    =   a3 & !reset;
WRITE   =   (WR1 # WR2 # WR3 # WR4 # WR5 # WR6 # WR7 # WR8 # WR9 # WRTN # XTWR);
READ    =   !WRITE;

sequenced CODEC_time {
present T00     if !(RD_DEL # WR_DEL) & !reset  next T00;
                if  (RD_DEL # WR_DEL) & !reset  next T01;
                if  reset       next T12;
present T01         next T02;
present T02         next T03;
present T03         next T04;
present T04         next T05;
present T05     if  (RD8 # RD9 # WR8 # WR9) & !reset  next T12;
                if !(RD8 # RD9 # WR8 # WR9) & !reset  next T06;
                if  reset       next T12;
present T06         next T07;
present T07         next T08;
present T08         next T09;
present T09         next T10;
present T10         next T11;
present T11         next T12;
present T12     if  reset       next T12;
                if !reset       next T00;
}

sequence CODEC_sm {
present RW0     if !(DMA # PIO) # reset                             next RW0;
                if  (DMA & CPU_RD # PIO & CPU_RD) & !reset          next R01;
                if  (DMA & CPU_WR # PIO & CPU_WR) & !reset          next W01;
present R01                                                         next R02;
present R02                                                         next RXT;
/* Don't need this step is <66MHz bus & 7nS PLA          */
present RXT                                                         next R03;
present R03         if !TM12                                        next R03;
                    if  TM12                                        next R04;
present R04         if  DMA                                         next R05;
                    if !DMA                                         next R08;
present R05         if !TM12                                        next R05;
                    if  TM12                                        next R06;
```

```
/* --------------------------------------------------------------*/
/* If sufficient product terms can have generic state machine for       */
/* 8-bit mono DMA and 8-bit stereo/16-bit mono. This PLA has            */
/* insufficient terms to do this !! So only support 16-bit cycles       */
/* Terms to add for generic—replace existing R05 term with: -           */
/*
present R05          if!TM12                                      next R05;
                     if  TM12 &  cdrq                             next R06;
                     if  TM12 & !cdrq                             next R09;
*/
/*---------------------------------------------------------------*/

present R06          if !TM12                                     next R06;
                     if  TM12                                     next R07;
present R07                                                       next R08;
present R08          if !TM12                                     next R08;
                     if  TM12                                     next R09;
present R09          if !TM12                                     next R09;
                     if  TM12                                     next R10;
present R10                                                       next RW0;

present W01          next W02;
present W02          next WXT;
/* Don't need this step is <66MHz bus & 7nS PLA                 */
present WXT                                                       next W03;
present W03          if !TM12 next W03;
                     if  TM12 & DMA                               next W04;
                     if  TM12 & PIO                               next W08;

/* Problem with this jump is that G1 should be enabled in WR2 for
   PIO and DMA writes. If DMA cycle, G2 is enabled in WR6, so get
   contention in WR6,WR7 since G1 & G2 low. Can't solve by changing
   jump point as not enough product terms need to fix elsewhere          */

present W04          if !TM12 next W04;
                     if  TM12 next W05;
present W05                                                       next W06;

/* --------------------------------------------------------------*/
/* If sufficient product terms can have generic state machine for       */
/* 8-bit mono DMA and 8-bit stereo/16-bit mono. 22V10 PLA has           */
/* insufficient terms to do this !! So only support 16-bit cycles       */
/* Terms to add for generic—replace existing W06 term with: -           */
/*
present W05          if  pdrq                                     next W06;
                     if !pdrq                                     next W08;
*/
/*---------------------------------------------------------------*/

present W06                                                       next W07;
present W07          if !TM12                                     next W07;
                     if  TM12                                     next W08;
present W08          if !TM12                                     next W08;
                     if  TM12                                     next W09;
present W09          if !TM12                                     next W09;
```

```
                if  TM12                                        next W10;
present W10     next RW0;
}

cdc_sel.d           = PIO_START      &   !(pdak # cdak)
                    # CDMA_START     &   !pdak
                    # PDMA_START     &   !cdak
                    # cdc_sel & !e_ta &  !reset;

e_ta.d              = (WR8 & TM0) # (RD8 & TM5); /* N.B. RD8/TM5 timing critical */
                                     /* If the address were latched  */
                                     /* could give early ta on writes*/
pwrdwn.d            = reset
                    # PDN_START
                    # pwrdwn & !PUP_START  ;  /* Must assert PUP after reset */

e_tea.d             = PIO_START      &   pdak
                    # PIO_START      &   cdak
                    # CDMA_START     &   pdak
                    # PDMA_START     &   cdak
                    # CODEC_TS       &   BURST
                    # CODEC_TS       &   ADD_ONLY;

ta.d                = e_ta
                    # PDN_START
                    # PUP_START;

hst_irq             = int;  /* active hi CODEC out to active low in */

tea.d               = e_tea;

aack.d              = e_ta # e_tea;

pd_irq              = pdrq;  /*  active hi CODEC out to active low in */

cd_irq              = cdrq;  /* active hi CODEC out to active low in */

pdak.d              =  WR1     &    DMA3
                    #  WR2     &    DMA3
                    #  XTWR    &    DMA3
                    #  WR3     &    DMA3
                    #  WR4     &    DMA3
                    #  WR5     &    DMA3
                    #  WR6     &    DMA3
                    #  WR7     &    DMA3
                    #  WR8     &    DMA3
                    #  WR9     &    pdrq & !reset & pdak  /* 16-bit stereo support */
                    #  WRTN    &    pdak & !reset        /* 16-bit stereo support */
                    #  IDLE    &    pdak & !reset;       /* 16-bit stereo support */

cdak.d              =  RD1     &    DMA3
                    #  RD2     &    DMA3
                    #  XTRD    &    DMA3
                    #  RD3     &    DMA3
                    #  RD4     &    DMA3
```

```
                     # RD5     &   DMA3
                     # RD6     &   DMA3
                     # RD7     &   DMA3
                     # RD8     &   DMA3
                     # RD9     &   cdak & !reset & cdrq
/* This term supports 16-bit stereo DMA cycles. To determine if 16-bit  */
/* stereo test if cdrq asserted or negated after rd_/wr_ negated        */
/* If not asserted then either 2nd 16-bit stereo cycle or 16-bit mono.  */
/* R09 occurs 6 times (TM1-5/12) only if cdrq continues to be           */
/* asserted will cdak continue to be asserted. once negated             */
/* can't be asserted again until R01                                    */
                     # RD10    &   cdak & !reset          /* 16-bit stereo support */
                     # IDLE    &   cdak & !reset;         /* 16-bit stereo support */

rd_.d                = RD3
                     # RD4                                /* latch data here PIO/DMA */
                     # RD6
                     # RD7;                               /* latch data here DMA only*/

wr_.d                = WR3
                     # WR7;

cs_.d                = RD1     &   PIO3
                     # RD2     &   PIO3
                     # XTRD    &   PIO3
                     # RD3     &   PIO3
                     # RD4     &   PIO3
                     # RD8     &   PIO3
                     # WR1     &   PIO3
                     # WR2     &   PIO3
                     # XTWR    &   PIO3
                     # WR3     &   PIO3
                     # WR8     &   PIO3;

g2.d                 = RD8     &   !reset
                     # WR2     &   !reset  &  PIO3        /* g2 cntrls 8-bit latch on     */
                     # XTWR    &   !reset  &  PIO3        /* DH0–DH7. Using g2 for PIO    */
                     # WR3     &   !reset  &  PIO3        /* gives accesses on lwrd locns */
                     # WR6     &   !reset                 /* Don't need to qualify DMA as */
                     # WR7     &   !reset                 /* PIO cycles never get here    */
                     # WR8     &   !reset;                /* PIO wrts WR3->WR8 included   */

g1.d                 = RD8     &   !reset
                     # WR2     &   !reset  &  DMA3
                     # XTWR    &   !reset  &  DMA3
                     # WR3     &   !reset  &  DMA3
                     # WR4     &   !reset  &  DMA3;       /* Don't need to qualify DMA as */
                                                          /* PIO cycles never get here    */

dir.d                = RD1     &   !reset
                     # RD2     &   !reset
                     # XTRD    &   !reset
                     # RD3     &   !reset
                     # RD4     &   !reset
                     # RD5     &   !reset
```

```
                    # RD6   & !reset
                    # RD7   & !reset
                    # RD8   & !reset
                    # RD9   & !reset
                    # RD10  & !reset;

cp1.d               = RD4   & !reset & DMA3    /* 1st byte fm CODEC is lsb   */
                    # WR1   & !reset;

cp2.d               = RD4   & !reset & PIO3    /* puts data on DH0–DH7       */
                    # RD7   & !reset           /* PIO reads skip this state  */
                    # WR1   & !reset;          /* writes ok for both PIO & DMA */


/** Clocks **/
[state4..0].ckmux  =  clk;
[time3..0].ckmux   =  clk;
e_ta.ckmux         =  clk;
pwrdwn.ckmux       =  clk;
ta.ckmux           =  clk;
aack.ckmux         =  clk;
rd_.ckmux          =  clk;
wr_.ckmux          =  clk;
cs_.ckmux          =  clk;
g1.ckmux           =  clk;
g2.ckmux           =  clk;
cp1.ckmux          =  clk;
cp2.ckmux          =  clk;
dir.ckmux          =  clk;
cdak.ckmux         =  clk;
pdak.ckmux         =  clk;
tea.ckmux          =  clk;
e_tea.ckmux        =  clk;
cdc_sel.ckmux      =  clk;

/** Resets  **/
state0.ar          =  'b'0;
state1.ar          =  'b'0;
state2.ar          =  'b'0;
state3.ar          =  'b'0;
state4.ar          =  'b'0;
time0.ar           =  'b'0;
time1.ar           =  'b'0;
time2.ar           =  'b'0;
time3.ar           =  'b'0;
e_ta.ar            =  'b'0;
rst_pla.ar         =  'b'0;
pwrdwn.ar          =  'b'0;
aack.ar            =  'b'0;
tea.ar             =  'b'0;
e_tea.ar           =  'b'0;
ta.ar              =  'b'0;
pdak.ar            =  'b'0;
cdak.ar            =  'b'0;
rd_.ar             =  'b'0;
```

```
wr_.ar          =   'b'0;
cs_.ar          =   'b'0;
cp1.ar          =   'b'0;
cp2.ar          =   'b'0;
dir.ar          =   'b'0;
g1.ar           =   'b'0;
g2.ar           =   'b'0;
cdc_sel.ar      =   'b'0;

/** Enables **/
pwrdwn.oe       =   'b'1;
hst_irq.oe      =   'b'1;
aack.oe         =   'b'1;
tea.oe          =   'b'1;
ta.oe           =   'b'1;
pdak.oe         =   'b'1;
cdak.oe         =   'b'1;
rd_.oe          =   'b'1;
wr_.oe          =   'b'1;
cs_.oe          =   'b'1;
cp1.oe          =   'b'1;
cp2.oe          =   'b'1;
dir.oe          =   'b'1;
g1.oe           =   'b'1;
g2.oe           =   'b'1;
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™ semiconductor