APPLICATION NOTE

How to Implement AMI Modules From SOM-A200 series CSB AMI-120 Interface

Released Version:V1.14
Released Date: May,09, 2004

Advantech Co., Ltd
Risc Embedded Computing Division
http://www.advantech.com/risc

RISC Express

ABSTRACT

This application note explains how to design AMI (ARM Module Interface) modules from SOM-A200 series Customer Solution Board(CSB) AMI-120 interface. The content includes AMI-120 connector information, pin configuration and description, memory map and how to add on user driver with Advantech SOM-A255x series BSP.

For more information on this and other Advantech products please visit our website at:

http://www.advantech.com

http://www.advantech.com/risc

For technical support and service for please visit our support website at:

http://eservice.advantech.com.tw/eservice/

Or directly mail to Advantech RISC platform application engineer

AE.RISC@advantech.com.tw

Revision History

Version	Date	Reason
V1.13	2004.04.26	AMI C4 pin should be SYS_VCC3P3 , not GND.
V1.14	2004.05.09	AMI C10 pin is changed from N.C. to nPXA_CS2.

I. Introduction

AMI-120 is designed for expanding features for demanding embedded applications. It is a Advantech self-defined expansion bus to customize or configure AMI modules for any specific applications. Customers can design their self-defined modules or extra functions except using Advantech AMI modules on Advantech RISC platform. This application note explains how to implement AMI modules on SOM-A200 series CSB (Customer Solution Board) through AMI-120 interface. The mechanical of AMI connector is the same as PC/104 Plus shown as Figure 1 and is a 4*30-pin connector. This type of connector provides a strong, dust free (virtually gas tight) connection for higher reliability.

Figure 1. AMI-120 Connector

II. Mechanical

Advantech SOM-A200 series CSB provides an AMI-120 interface for expansion function. The AMI-120 connector P/N of SOM-A200 series CSB is EPT 264-40303-02 as Figure 2. The matting connector of AMI-120 connector is EPT 264-60303-12 as Figure 2 and its shroud is EPT 264-17302 as Figure 4. The other solution is using PC/104 plus straight headers as EPT 272-30000-31 as Figure 5.

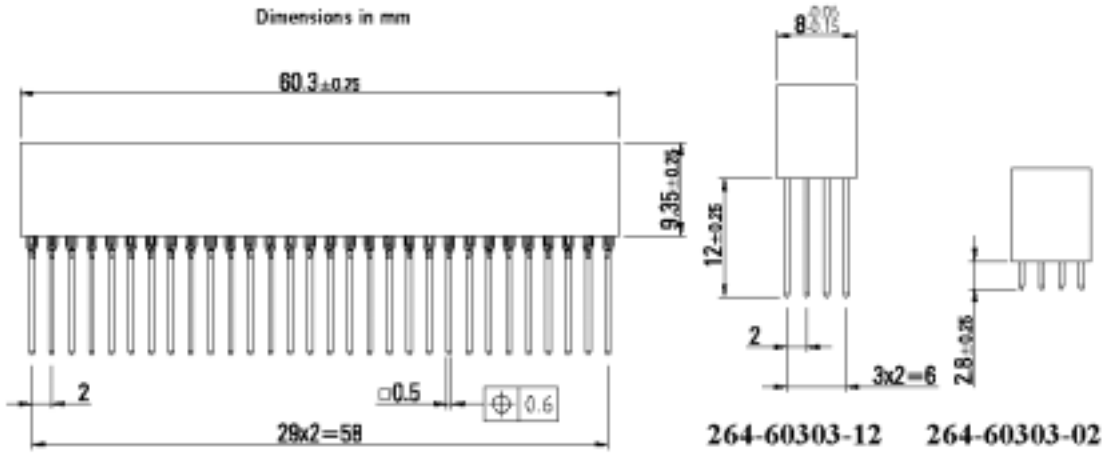

Figure 2. The dimension of AMI-120 connector and drawing

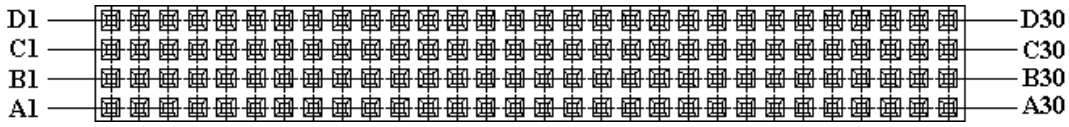

Figure 3. The pin drawing of AMI-120 connector

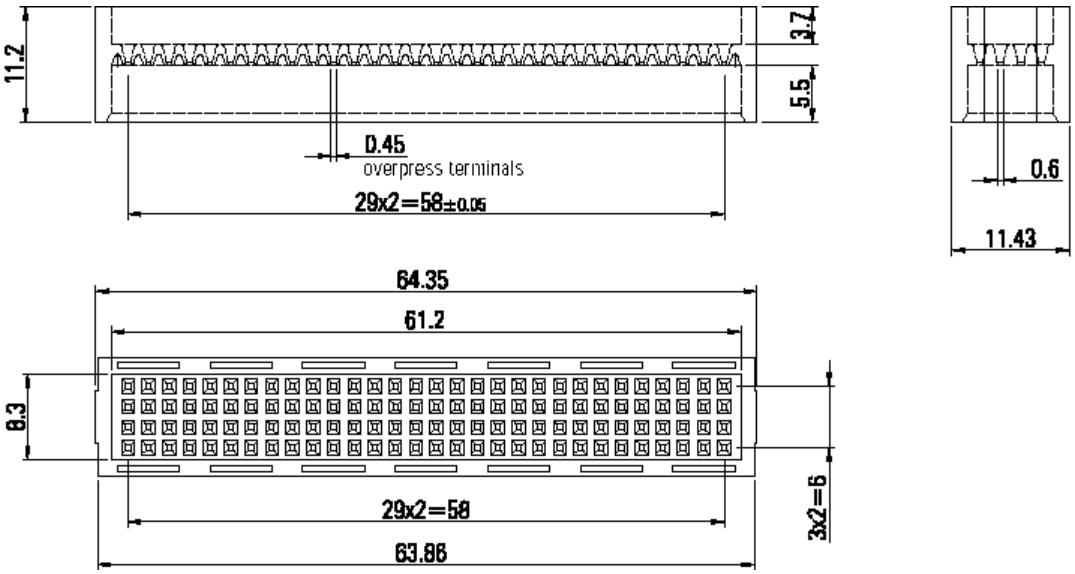

Figure 4. The shroud of AMI-120 matting connector drawing (EPT 264-17302)

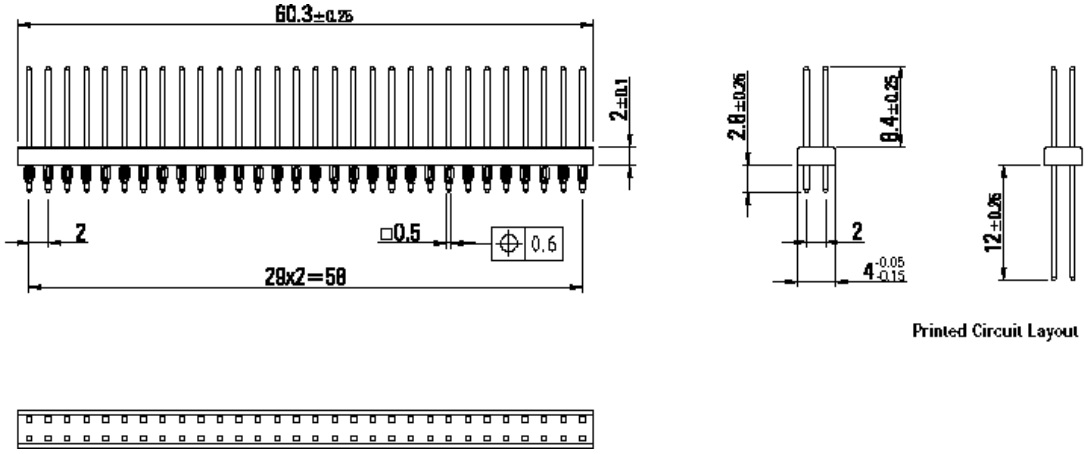

Figure 5. The PC/104 plus straight header drawing (EPT 272-30000-31)

III. Pin Configuration and Description

a. Pin Configuration

The following table describes all pins of the AMI-120 connector listed in numeric order.

Table 1. Pin Configurator of AMI-120 connector

Pin	Signals	Pin	Signals	Pin	Signals	Pin	Signals
A1	SDRAM_CLK1	B1	GND	C1	SDRAM_CKE1	D1	PWR_EN
A2	GND	B2	3.6864MHz	C2	SYS_VCC3P3*	D2	SYS_VCC3P3*
A3	PXA_GPIO27	B3	Reserved	C3	PXA_GPIO3	D3	PXA_GPIO9
A4	Reserved	B4	PXA_GPIO7	C4	SYS_VCC3P3*	D4	SYS_VCC3P3*
A5	PXA_A0	B5	PXA_A1	C5	PXA_A15	D5	PXA_A14
A6	PXA_A2	B6	PXA_A3	C6	PXA_A13	D6	PXA_A12
A7	PXA_A4	B7	PXA_A5	C7	PXA_A11	D7	PXA_A10
A8	PXA_A6	B8	PXA_A7	C8	PXA_A9	D8	PXA_A8
A9	PXA_A16	B9	PXA_A17	C9	PXA_A24	D9	PXA_A25
A10	PXA_A18	B10	PXA_A19	C10	nPXA_CS2***	D10	nPXA_OE***
A11	PXA_A20	B11	PXA_A21	C11	nPXA_WE***	D11	GND
A12	PXA_A22	B12	PXA_A23	C12	PXA_RD_nWR***	D12	RDY
A13	SYS_VCC3P3*	B13	nPXA_CS3***	C13	nPXA_CS4***	D13	nPXA_CS5***
A14	PXA_D0	B14	PXA_D1	C14	PXA_D15	D14	PXA_D14
A15	PXA_D2	B15	PXA_D3	C15	PXA_D13	D15	PXA_D12
A16	PXA_D4	B16	PXA_D5	C16	PXA_D11	D16	PXA_D10
A17	PXA_D6	B17	PXA_D7	C17	PXA_D9	D17	PXA_D8
A18	PXA_D16	B18	PXA_D17	C18	PXA_D31	D18	PXA_D30
A19	PXA_D18	B19	PXA_D19	C19	PXA_D29	D19	PXA_D28
A20	PXA_D20	B20	PXA_D21	C20	PXA_D27	D20	PXA_D26
A21	PXA_D22	B21	PXA_D23	C21	PXA_D25	D21	PXA_D24
A22	nSDRAM_RAS***	B22	nSDRAM_CAS***	C22	nPXA_RESET***	D22	GND

A23	nSDRAM_CS0***	B23	nSDRAM_CS1***	C23	nRESET_OUT***	D23	GND
A24	DQM0	B24	DQM1	C24	nBATT_FALT***	D24	SYS_VCC3P3*
A25	DQM2	B25	DQM3	C25	nVDD_FALT***	D25	SYS_VCC3P3*
A26	Reserved	B26	Reserved	C26	Reserved	D26	Reserved
A27	Reserved	B27	Reserved	C27	SYS_VCC**	D27	SYS_VCC**
A28	Reserved	B28	Reserved	C28	Reserved	D28	PXA_GPIO20
A29	Reserved	B29	Reserved	C29	PXA_GPIO22	D29	PXA_GPIO19
A30	MBREQ	B30	MBGNT	C30	GND	D30	SDRAM_CLK2

*SYS_VCC3P3 is +3.3V, no matter system is in run mode or sleep mode.

**SYS_VCC is +5V and will be removed while system is in sleep mode.

***n-indicated active low signal

b. Pin Description

The following table describes the signals of AMI-120 connector.

Signal Attribute: n — Active low signal
IC — Input, CMOS threshold
ICOCZ — Input, CMOS threshold, output CMOS level, tristatable
OCZ — Output, CMOS levels, tristatable

Table 2. Pin Description of AMI-120 connector

Name	Pin(s) No.	Attribute	Description
PXA_A 0:25	A5-A12, B5-B12, C5-C9, D5-D9	ICOCZ	26-bit system address bus. Bits PXA_A 10:24, as inputs, are used for DMA access to SDRAM.
PXA_D 0:31	A14-A21, B14-B21, C14-C21, D14-D21	ICOCZ	32-bit system data bus.
DQM 0:3	A24, B24, A25, B25	ICOCZ	Byte Data Output Mask Enable for SDRAM and SRAM-type write transfer. Bits DQM 0:3, as inputs, are used for DMA access to SDRAM.
nPXA_OE	D10	ICOCZ	Memory Output Enable.
nPXA_WE	C11	ICOCZ	Memory Write Enable.
PXA_RD_nWR	C12	ICOCZ	Read/Write direction control for memory.
RDY	D12	IC	Static data ready signal for nPXA_CS 3:5. This signal should be connected to the data ready output pins of variable latency I/O devices that require variable data latencies.
nSDRAM_CAS	B22	ICOCZ	SDRAM Column Address Strobe. This bit is as input is used for DMA access to SDRAM
nSDRAM_RAS	A22	ICOCZ	SDRAM Row Address Strobe. This bit is as input is used for DMA access to SDRAM

SDRAM_CLK1	A1	ICOCZ	SDRAM Clock 1. PCM-7220 onboard SDRAM is using this pin for memory clock. This bit is as input is used for DMA access to SDRAM. The frequency of SDRAM clock is 99.5MHz.
SDRAM_CLK2	D30	ICOCZ	SDRAM Clock 2. PCM-7220 expansion memory module SDRAM is using this pin for memory clock. This bit is as input is used for DMA access to SDRAM. The frequency of SDRAM clock is 50MHz
SDRAM_CKE1	C1	OCZ	SDRAM Clock Enable 1. PCM-7220 onboard SDRAM is using this pin for memory clock enable.
nSDRAM_CS0	A23	ICOCZ	SDRAM Chip Select 0. PCM-7220 onboard SDRAM is using this pin for memory chip select. This bit is as input is used for DMA access to SDRAM.
nSDRAM_CS1	B23	ICOCZ	SDRAM Chip Select 1. PCM-7220 expansion memory module SDRAM is using this pin for memory chip select. This bit is as input is used for DMA access to SDRAM.
nPXA_RESET	C22	IC	Hardware Reset. This pin is an active low signal and a level-sensitive input used to start the processor from address 0. This pin is connected directly from Intel PXA255 processor J13 pin.
nRESET_OUT	C23	OCZ	Reset Out. This signal is asserted when nRESET is asserted and deasserts when the processor has completed resetting. nRESET_OUT is also asserted for "soft" reset events (sleep and watchdog). This pin is connected directly from Intel PXA255 process K11 pin.
PWR_EN	D1	OCZ	Power Enable. PWR_EN enables the external VDD power supply and is an active high signal. While the system is going into sleep mode, this signal will be deasserted. At the same time, the VDD power supply should be removed.
nBATT_FALT	C24	IC	Battery Fault. System will enter sleep mode or force an imprecise data exception while this pin is asserted. Intel PXA255 process will not recognize a walk-up event while this signal is asserted. Minimum assertion time for nBATT_FALT is 1 ms.

nVDD_FALT	C25	IC	VDD Fault. System will enter sleep mode or force an imprecise data exception while this pin is asserted. nVDD_FALT is ignored after a walk-up event until the power supply timer completes (approximately 10 ms). Minimum assertion time for nVDD_FALT is 1 ms.
3.6864MHz	B2	OCZ	3.6864MHz clock output from Intel PXA255 processor.
MBREQ	A30	IC	Memory Controller Alternate Bus Master Request. Allows an external device to request the system bus from the Memory Controller.
MBGNT	B30	OCZ	Memory Controller Grant. Notifies an external device that it has been granted the system bus.
PXA_GPIO3	C3	ICOCZ	General Purpose I/O. This pin is connected directly from Intel PXA255 processor K14 pin. Reserved for external display controller interrupt.
PXA_GPIO7	B4	ICOCZ	General Purpose I/O. This pin is connected directly from Intel PXA255 processor G15 pin.
PXA_GPIO9	D3	ICOCZ	General Purpose I/O. This pin is connected directly from Intel PXA255 processor F12 pin. Reserved for Advantech or Intel companion chip interrupt.
PXA_GPIO19	D29	ICOCZ	General Purpose I/O. This pin is connected directly from Intel PXA255 processor N14 pin. This pin is pull-down by 100k ohm resistor.
PXA_GPIO20	D28	ICOCZ	General Purpose I/O. This pin is connected directly from Intel PXA255 processor N12 pin. This pin is pull-down by 100K ohm resistor.
PXA_GPIO22	C29	ICOCZ	General Purpose I/O. This pin is connected directly from Intel PXA255 processor M12 pin.
PXA_GPIO27	A3	ICOCZ	General Purpose I/O. This pin is connected directly from Intel PXA255 processor B9 pin. Reserved for external USB host controller interrupt.
nPXA_CS2	C10	OCZ	Static Chip Select 2. Reserved for Advantech or customer future IC chip select.
nPXA_CS3	B13	OCZ	Static Chip Select 3. Advantech programs a memory block for users self-defined modules I/O control. Refer to section V. AMI bus memory map.
nPXA_CS4	C13	OCZ	Static Chip Select 4. Reserved for Advantech or Intel companion IC chip select.

nPXA_CS5	D13	OCZ	Static Chip Select 5. Reserved for display controller chip select.
GND	B1, A2, D11, D22, D23, C30	Grounded	Must be connected to PCB ground
SYS_VCC3P3	C2, C4,D2, D4, A13, D24, D25	Power	Supply +3.3V power. No matter the system is in normal or sleep mode, these power sources still provide +3.3V power.
SYS_VCC	C27, D27	Power	Supply +5V power. These power sources will provide +5V power but will be removed while the system is in sleep mode.
NC	C10	Disconnected	This pin is disconnected.
Reserved	B3, A4, A26-A29, B26-B29, C26, C28, D26	Reserved	These pins are reserved or have been used for the PCM-7220 SBC already. **Note:** Don't connect these pins and keep these pins disconnect.

IV. AMI-120 bus Timing

Refer to Table 6-21 on Intel PXA255 processor manual page 6-45 Asynchronous Static Memory Control Registers (MSCx) shown as Table 3. We can configure these registers to set the chip select access timing. Bit 0-15 of MCS1 register is about chip select 2 setting and AMI-120 doesn't provide chip select 2 for users expanding. Here we only discuss the values of MCS1 about the chip select 3. Advantech sets "**3ff1**" as the values of MCS1 register high word (bit 16:31).

Table 3. MSC1 Bit Definitions

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	RBUFF3	RRR3			RDN3				RDF3				RBW3	RT3		
	0	011			1111				1111				0	001		

Bits	Access	Name	Description
31	R/W	RBUFF3	Return Data Buffer vs. Streaming behavior. When slower memory or I/O devices are used in the system (e.g. VLIO, slow SRAM/ROM), this bit must be reset to allow the system to not have to remain idle while all data is read from the device. By resetting this bit, the system is allowed to process other information. When set, the internal bus may halt while all data is returned from the device. The value of the RBUFF bit does not affect the behavior of the external memory bus. Once a transaction begins on the memory bus, it must be completed before another transaction starts. When Synchronous Static

			memory devices have been enabled for a given bank, this value will default to Streaming behavior (assuming a faster device). The register bit will still read as 0 (Return Data Buffer) unless it has specifically been programmed to a 1. This cannot be overridden. 0 – Slower device (Return Data Buffer) 1 – Faster device (Streaming behavior)
30:28	R/W	RRR3<2:0>	ROM/SRAM recovery time. Chip select deasserted after a read/write to next chip select (including the same static memory bank) or nSDCS asserted is equal to (RRRx * 2) memclks. This field must be programmed with the maximum of tOFF (divided by 2), write pulse high time (Flash/SRAM), and write recovery before read (Flash).
27:24	R/W	RDN3<3:0>	ROM delay next access Address to data valid for subsequent access to burst ROM or Flash is equal to (RDNx + 1) memclks. nWE assertion for write accesses to SRAM is equal to (RDFx + 1) memclks. The nOE (nPWE) deassert time between each beat of read/write for Variable Latency I/O is equal to (RDNx + 2) memclks. For variable latency I/O, this number must be greater than or equal to 2.
23:20	R/W	RDF3<3:0>	ROM delay first access. RDF programmed RDF value interpreted 0-11 0-11 12 13 13 15 14 18 15 23 Address to data valid for the first read access from all devices except VLIO is equal to (RDFx + 2) memclks. Address to data valid for subsequent read accesses to non-burst devices is equal to (RDFx + 1) memclks. nWE assertion for write accesses (which are non-burst) to all Flash is equal to (RDFx + 1) memclks. nOE (nPWE) assert time for each beat of read (write) is equal to (RDFx + 1) memclks for Variable Latency I/O (nCS[5:0]). For variable latency I/O, RDFx must be greater than or equal to 3.
19	R/W	RBW3	ROM bus width 0 – 32 bits 1 – 16 bits This value must be programmed with all memory types including synchronous

			static memory. This value must not change during normal operation.
18:16	R/W	RT3<2:0>	ROM type 000 - Nonburst ROM or Flash Memory 001 - SRAM 010 - Burst-of-four ROM or Flash (with non-burst writes) 011 - Burst-of-eight ROM or Flash (with non-burst writes) 100 - Variable Latency I/O (VLIO) 101 - reserved 110 - reserved 111 - reserved Burst refers to the device's timing. When the subsequent reads from the device take less time than the first read from a device, it is referred to as burst timing. The address bits must also be taken into account for burst timing devices. For example, in a burst-of-four device, only the lower two non-byte address bits can change for burst timing. For 32-bit devices, this is PXA_A [3:2]. The address order can go 00, 01, 10, 11 where the reads from 01, 10, and 11, take less time to come out of the device. For burst-of-eight devices, the lower three non-byte address bits can change. Writes to these devices are non-burst.

For the detail timing, refer to Figure6-20 on Intel PXA255 manual page 6-52 as Figure 7 and Figure 8. Here the frequency of memory clock is 99.5MHz. The relative parameters are defined as follows:

* tAS = Address setup to nCS = 1 MEMCLK
* tCES = nCS setup to nWE = 2 MEMCLKs
* tASW = Address setup to nWE low (asserted) = 1 MEMCLK
* tDSWH = Write data setup, DQM to nWE high (deasserted) = (RDN+2) = 4 MEMCLKs
* tDH = Data, DQM hold after nWE high (deasserted) = 1 MEMCLK
* tCEH = nCS held asserted after nWE deasserted = 1 MEMCLK
* tAH = Address hold after nWE deasserted = 1 MEMCLK
* nWE high time between burst beats = 2 MEMCLKs

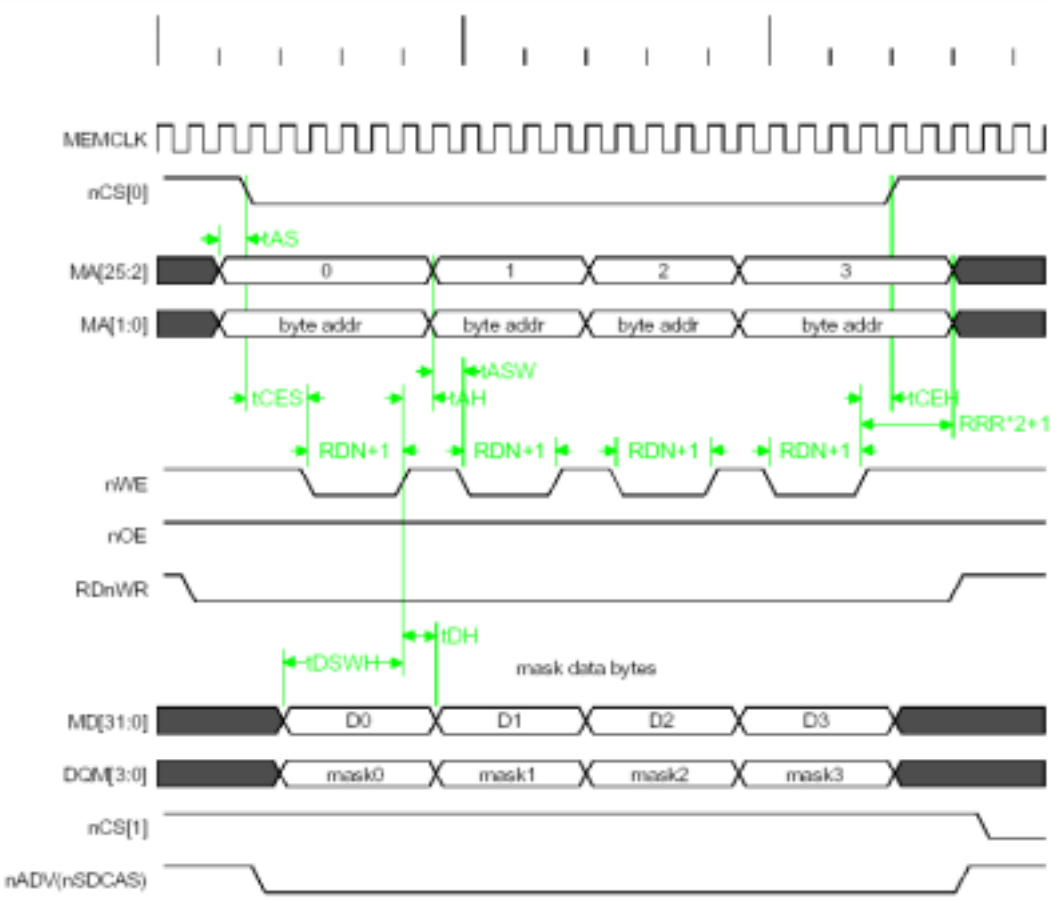

Figure 7. 32-bit SRAM Write Timing Diagram

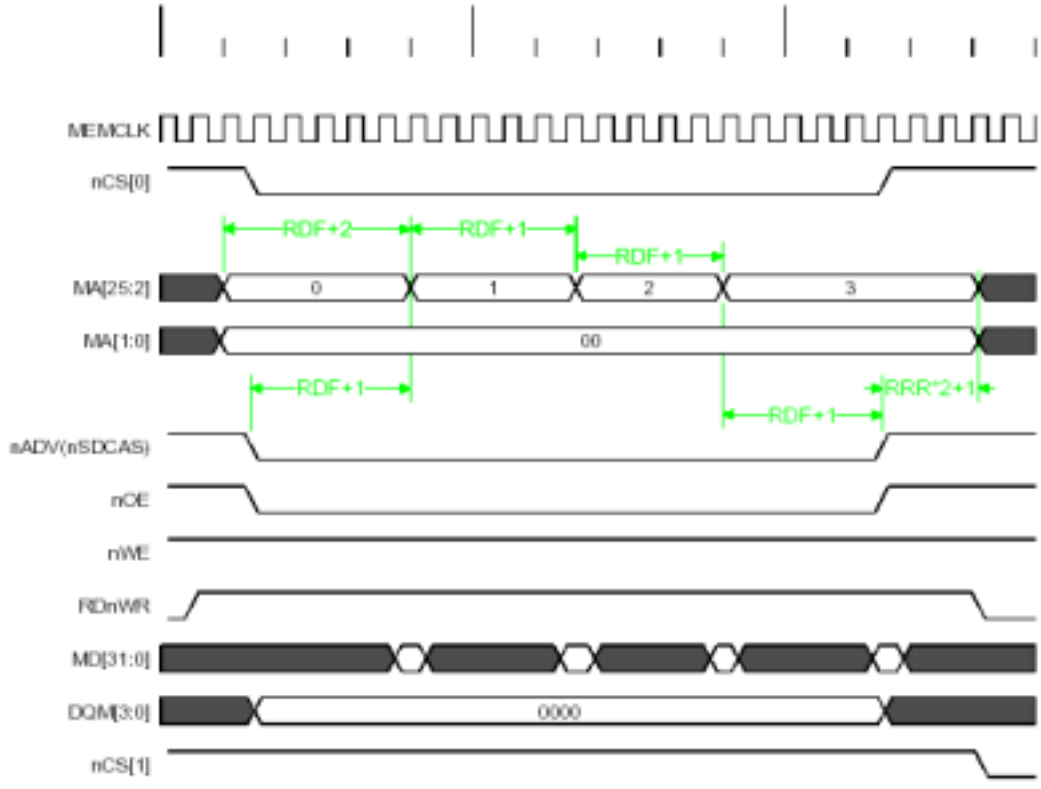

Figure 8. 32-bit SRAM Read Timing Diagram

V. AMI-120 bus memory map

Advantech programs partial intel PXA255 processor static chip select 3 memory block for customized AMI-120 modules memory or I/O control. We also reserved static chip select 4 for Advantech companion chip to expand more I/O interface and make the system more powerful. Besides, chip select 5 is also reserved for access external display controller to enhance the display performance. The partial memoy block of static chip select 3 is from physical address **0x0E80 0000h** to **0x0EFF FFFFh(total 8 MB)**. User can design glue logic circuit or use CPLD to decode this memory block. That is, when nPXA_CS3 is asserted and address PXA 23:25 are "**101**", these block will be accessed and the corresponse chip select signal will be asserted. Don't access other memory block of the rest static chip select3. Because Advantech has used the rest block for controlling PCM-7220 I/O devices like buzzer and so on. We recommend that customers use the 8MB(0x0E80 0000h~0x0EFF FFFFh) memory block to design the AMI-120 modules.

VI. AMI-120 module SW development

This section describes how user develops his designed Windows CE drivers or applications to merge into PCM-7220 system. We here focus on the explanation for the kernel interface of PCM-7220 Board Support Package that user-specific drivers or applications would link with. And we expect the user is familiar with Microsoft Windows CE Platform Builder.

The PCM-7220 Board Support Package contains the OS components, device drivers, and applications in binary form that are ready-to-wear by user. Additionally, after installing this BSP with Microsoft Windows CE Platform Builder, user can add and remove the components, modify the registry settings, as well as develop specific drivers for his distinctive hardware features on AMI module to make a adaptable image.

A. Memory Map of AMI-120

Definitions:

```
#define AMI_RESERVED1_PHYSICAL          0x0E800000
#define AMI_RESERVED2_PHYSICAL          0x0EC00000
#define AMI_RESERVED1_U_VIRTUAL         0xB4800000
#define AMI_RESERVED2_U_VIRTUAL         0xB4C00000
```

Sample Program:

```
volatile unsigned *pUserRegister = NULL;
if(!pUserRegister) {
   if(!(pUserRegister=VirtualAlloc(0,0x1000,MEM_RESERVE,PAGE_NOACCESS))) {
      RETAILMSG(1, (TEXT("VirtualAlloc() failed!\r\n")) );
   }
```

```
    else {
      if(!VirtualCopy((PVOID)pUserRegister,(PVOID)AMI_RESERVED1_U_VIRTUAL,0x1000,PAGE_READWRITE|PAGE_
      NOCACHE)) {
        VirtualFree((PVOID)pUserRegister, 0, MEM_RELEASE);
        pUserRegister = NULL;
        RETAILMSG(1, (TEXT("VirtualCopy() failed!\r\n")) );
      }
      else {
        RETAILMSG(1, (TEXT("VirtualCopy() succeed!\r\n")));
        //To do user's register controlling here…
        pUserRegister = 0xff;
      }
    }
}
return TRUE;
```

B. Available GPIOs on AMI-120:

PXA_GPIO3 (Pin C3)

PXA_GPIO7 (Pin B4)

PXA_GPIO9 (Pin D3)

PXA_GPIO19 (Pin D29)

PXA_GPIO20 (Pin D28)

PXA_GPIO22 (Pin C29)

PXA_GPIO27 (Pin A3)

Sample Program:

```
#include "xsc1.h"
volatile GPIO_REGS *pGPIOReg = NULL;
if(!pGPIOReg) {
   if(!(pGPIOReg=(volatile GPIO_REGS *)VirtualAlloc(0,0x1000,MEM_RESERVE,PAGE_NOACCESS)))
   {
      RETAILMSG(1, (TEXT("VirtualAlloc() failed!\r\n")) );
   }
   else
      if(!VirtualCopy((PVOID)pGPIOReg,(PVOID)GPIO_BASE_U_VIRTUAL,0x1000,
                                 PAGE_READWRITE|PAGE_NOCACHE)) {
         VirtualFree((PVOID)pGPIOReg, 0, MEM_RELEASE);
         pGPIOReg = NULL;
         RETAILMSG(1, (TEXT("VirtualCopy() failed!\r\n")) );
```

```
        }
        else {
            RETAILMSG(1, (TEXT("VirtualCopy() succeed!\r\n")));
        }
}

//Disable GPIO Alternative Function
pGPIOReg->GAFR0_x &= ~GPIO_3_AF3;

//Set GPIO Direction as Output
pGPIOReg->GPDR_x |= GPIO_3;

//Set GPIO Level High
pGPIOReg->GPSR_x |= GPIO_3;

//Set GPIO Level Low
v_pGPIOReg->GPCR_x |= GPIO_3;

return TRUE;
```

C. PCM-7220 BSP Installation:

After installing PCM_7220_BSP.msi, the BSP folder, $(_WINCEROOT)\Platform\PCM_7220, will be created. This folder includes the sub-folders:

.\Advantech\ Retail : Contains the Advantech target platform ready-made components.

.\Files : Contains modifiable platform REG and BIB configuration files

.\Gwe\Buildexe: Contains the Gwes.exe building environment.

.\Kernel\Buildexe: Contains the Kern.exe building environment.

.\Kernel\AdvHal: Contains user hardware adaptation layer interface.

Firstly, user could develop his driver with standard Windows CE device drivers APIs.

Then, if the driver uses a system interrupt vector (defined in AdvOalintr.h), user should modify the relative gpio definitions in the kernel interrupt service routine (AdvHal.c). Finally, rebuild the platform to make a new image and test.

Sample Interrupt-Service-Thread of user developed driver:

```
#include <nkintr.h>
#include <Pkfuncs.h>
#include <AdvOalIntr.h>
// ...
BOOL     bStopThread                    = TRUE;
HANDLE hUserIntrEvent                   = NULL;
HANDLE hUserThreadTerminatedEvent = NULL;
DWORD    dwTriggerCount                 = 0;

void UserIntrThread(DWORD dwParam)
{
      // ...
      SetThreadPriority(GetCurrentThread(),THREAD_PRIORITY_ABOVE_NORMAL);
      hUserThreadTerminatedEvent=CreateEvent(NULL,FALSE,FALSE,NULL);
      if(hUserThreadTerminatedEvent==NULL)
         return;

      hUserIntrEvent = CreateEvent(NULL,FALSE,FALSE,NULL);
      if(hUserIntrEvent==NULL)
         return;
      else {
         InterruptDisable(SYSINTR_USERGPIO1);
         if(InterruptInitialize(SYSINTR_USERGPIO1,hUserIntrEvent,NULL,0)) {
            while(!bStopThread) {
               if(WaitForSingleObject(hUserIntrEvent,INFINITE)==WAIT_OBJECT_0) {
                  dwTriggerCount++;
                  RETAILMSG(1, (TEXT("dwTriggerCount=0x%x \r\n"), dwTriggerCount));
                  InterruptDone(SYSINTR_USERGPIO1);
               }
            }
            InterruptDisable(SYSINTR_USERGPIO1);
            SetEvent( hUserThreadTerminatedEvent );
         }
         else
               return;
    }
}
```

User adaptable kernel interface:

```
// ***************************************************************************
//      @func   BOOL | AdvOEMInterruptEnable | Enable a hardware interrupt
//      @rdesc  Returns TRUE if valid interrupt ID or FALSE if invalid ID.
//      @comm           This function is called by the PCM-7220 Wince Kernel
//                      when a device driver calls InterruptInitialize().
// ***************************************************************************

BOOL AdvOEMInterruptEnable(DWORD idInt, LPVOID pvData, DWORD cbData)
{
   switch(idInt) {
      case SYSINTR_USERGPIO1:
         //lpWriteDebugStringFunc(TEXT("OEMInterruptEnable: SYSINTR_USERGPIO1.\r\n"));
         GPIO_DisableAlternativeFunction( USERGPIO1 );
         GPIO_SetDirectionAsInput( USERGPIO1 );
         GPIO_EnableRisingEdgeTrigger( USERGPIO1 );
         GPIO_DisableFallingEdgeTrigger( USERGPIO1 );
         return TRUE;

/*
      case SYSINTR_USERGPIO2:
         //lpWriteDebugStringFunc(TEXT("OEMInterruptEnable: SYSINTR_USERGPIO2.\r\n"));
         GPIO_DisableAlternativeFunction( USERGPIO2 );
         GPIO_SetDirectionAsInput( USERGPIO2 );
         GPIO_EnableRisingEdgeTrigger( USERGPIO2 );
         GPIO_DisableFallingEdgeTrigger( USERGPIO2 );
         return TRUE;
         ...
*/
      }
      return FALSE;
}


// ***************************************************************************
//      @func   BOOL | AdvOEMInterruptDisable | Disable a hardware interrupt
//      @rdesc  Returns TRUE if valid interrupt ID or FALSE if invalid ID.
//      @comm           This function is called by the PCM-7220 WinCE Kernel
```

```
//                      when a device driver calls InterruptDisable().
// *************************************************************************

BOOL AdvOEMInterruptDisable(DWORD idInt)
{
   switch(idInt) {
      case SYSINTR_USERGPIO1:
      //lpWriteDebugStringFunc(TEXT("OEMInterruptDisable: SYSINTR_USERGPIO1.\r\n"));
         GPIO_DisableRisingEdgeTrigger( USERGPIO1 );
         GPIO_DisableFallingEdgeTrigger( USERGPIO1 );
         GPIO_ClearDetectedInterrupt( USERGPIO1 );
         return TRUE;

/*
      case SYSINTR_USERGPIO2:
         //lpWriteDebugStringFunc(TEXT("OEMInterruptDisable: SYSINTR_USERGPIO2.\r\n"));
         GPIO_DisableRisingEdgeTrigger( USERGPIO2 );
         GPIO_DisableFallingEdgeTrigger( USERGPIO2 );
         GPIO_ClearDetectedInterrupt( USERGPIO2 );
         return TRUE;
         ...
*/
   }
   return FALSE;
}



// *************************************************************************
//      @func   BOOL | AdvOEMInterruptDone | Signal completion of interrupt processing
//      @rdesc  Returns TRUE if valid interrupt ID or FALSE if invalid ID.
//      @comm           AdvOEMInterruptDone is called by the PCM-7220 WinCE Kernel
//                      when a device driver calls InterruptDone().
// *************************************************************************

BOOL AdvOEMInterruptDone( DWORD idInt )
{
   switch(idInt) {
      case SYSINTR_USERGPIO1:
         //lpWriteDebugStringFunc(TEXT("OEMInterruptDone: SYSINTR_USERGPIO1.\r\n"));
```

```
            GPIO_EnableRisingEdgeTrigger( USERGPIO1 );
            return TRUE;

/*
        case SYSINTR_USERGPIO2:
            //lpWriteDebugStringFunc(TEXT("OEMInterruptDone: SYSINTR_USERGPIO2.\r\n"));
            GPIO_EnableRisingEdgeTrigger( USERGPIO2 );
            return TRUE;
            ...
*/
    }
    return FALSE;
}

// ***************************************************************************
// @func: AdvOEMInterruptHandler
// @comm: Figure out what caused the interrupt, then return the interrupt ID for that driver
// ***************************************************************************

int AdvOEMInterruptHandler()
{
        if( GPIO_InterruptDetected(USERGPIO1) )
        {
            //lpWriteDebugStringFunc(TEXT("Got SYSINTR_USERGPIO1 Interrupt.\r\n"));
            GPIO_DisableRisingEdgeTrigger( USERGPIO1 );
            GPIO_ClearDetectedInterrupt( USERGPIO1 );
            return(SYSINTR_USERGPIO1);
        }
        // ...
        return SYSINTR_NOP;
}
```