Technical Notes on using Analog Devices' DSP components and development tools from the DSP division
Phone: (800) ANALOG-D, FAX: (781) 461-3010, FTP: ftp.analog.com, EMAIL: dsp.support@analog.com

# Using external bus arbitration to group more than two ADSP-21065L into a multiprocessing cluster

*contributed by hs*

## Introduction:

The new ADSP-21065L allows by its system design to face high speed real words digital signal processing applications. To acquire the data or to provide external mass storage, peripherals have to be interfaced to the external port of the SHARC processor. If the complexity of the task rises and more than two ADSP-21065L are required to meet the real time specification, an external bus arbiter is necessary to keep the external bus free of conflicts. This application note will show how easily more than two ADSP-21065L can be arbitrated using a small external programmable logic device.

## Multiprocessing capabilities:

The ADSP-21065L is already equipped with configuration pins and bus arbitration logic for up-to two processors. These dedicated pins are:

| | |
|---|---|
| $ID_{1-0}$ | : Definition of processor ID within a cluster |
| $BR_{1-0}$ | : Bus arbitration lines between two processors |

So in a cluster of two, it is totally sufficient to assign one processor to ID1 and the other to ID2. Finally, the $BR_1$ and $BR_2$ lines have to be connected together, as they are used for the handshaking between the processors. This allows to connect both external ports together and to log onto a common external address and data bus which can be mastered by a host processor, too.

## Bus arbitration protocol:

When a multiprocessing system is reset by the RESET pin, the bus arbitration logic on each processor must synchronize to insure that only one ADSP-21065L will drive the external bus. There must be one bus master, and the other processors must recognize which one it is before actively arbitrating for the bus. The bus synchronization scheme also allows the system to safely bring two ADSP-21065L into and out of reset.

As soon as more than two processors have to share the external resources, further external arbitration logic is necessary to prevent bus conflicts.

This external logic has to take care that after reset all ADSP-21065L will be synchronized and update their internal record of who the current master is (in the current bus master field, CRBM, of the SYSTAT register). The synchronization after reset performs according to the following rules:

- All ADSP-21065L except the one with ID=1 will deassert their $BR_x$ line during reset. They will keep their $BR_x$ deasserted for at least two cycles after reset and until their bus arbitration logic is synchronized.
- Afterwards an ADSP-21065L will consider itself synchronized when it sees a cycle in which only one $BR_x$ line is asserted. The ADSP-21065L will identify the bus master by recognizing which $BR_x$ is asserted, and will update its internal record of who the current master is.
- The bus master will drive all memory strobes directly after reset to prevent them from floating.

## Realization in external logic:

To emulate this protocol, every processor must be configured to ID=2 to remove multiple bus masters driving memory strobes after reset. This is done by the external logic asserting the $BR_1$ pin of each processor forming the cluster during reset. Further-more it will keep $BR_1$ pin low after reset to prevent the bus slaves from glitching into the target. An ADSP-21065L with ID=2 will drive its $BR_2$ low after reset. If it sees the $BR_1$ line asserted after reset, it will deassert its $BR_2$ and check $BR_1$ within the next two cycles. As this is held low by the programmable logic device, it will consider itself as a bus slave with ID=2 and normal processor arbitration can start.

The external logic will give bus mastership to the attached processors by monitoring their $BR_2$ pins. Every ADSP-21065L requiring external access for peripheral operations will indicate this with driving it $BR_2$ output pin low. If no other ADSP-21065L is currently requiring the external bus, the $BR_1$ line of the requesting ADSP-21065L will be deasserted (e.g. driven high) so that a bus transition cycle (BTC) can occur. All other $BR_1$ lines connected to the other processor will still be kept low. After this

BTC the ADSP-21065L has gained bus mastership and will perform the external operation. The end of this access is indicated by driving its $BR_2$ pin high again. Nevertheless the current ADSP-21065L will maintain the bus token ($BR_1$ high) until a different ADSP-21065L wants to perform an external access. This insures that the started SDRAM controller will supply proper signals to possibly available SDRAM memory bank connected to the external port.

**Priority assignments:**

Of course, it now may happen that several ADSP-21065L requiring external access at the same time. Therefore there must be some provisions that the bus in only granted to one processor. Basically this gives two possible arbitration schemes with either fixed or rotating priority.

For example the bus mastership token would be granted in a cluster of four ADSP-21065L with fixed priority according to table1

| | | *Processor Number* | | |
|---|---|---|---|---|
| *Cycle #* | *P #1* | *P #2* | *P #3* | *P #4* |
| *1* | M | - | - | - |
| *2* | M | - | BR | BR |
| *3* | - | BR | M | BR |
| *4* | - | M | - | BR |
| *5* | - | - | - | M |

*table1: fixed priority*

where (M) = bus mastership, (BR) = external bus required and (-) = no external request and for rotating priority like in table2:

| | | *Processor Number* | | |
|---|---|---|---|---|
| *Cycle #* | *P #1* | *P #2* | *P #3* | *P #4* |
| *1* | M | - | - | - |
| *2* | M | - | BR | BR |
| *3* | - | BR | M | BR |
| *4* | - | BR | - | M |
| *5* | - | M | - | - |

*table2: rotating priority*

To improve the real time character of the bus usage it is advisable to check what arbitration scheme would be best suitable and to use the BMAX register to determine the maximum amount of time for bus masterhip.

**The programmable logic device:**

The following example of an bus arbiter for four ADSP-21065L explains the requirements for the logic device and the implemented schemes.

To monitor all $BR_2$ pins of all the connected ADSP-21065L, a certain number of input pins (4) is needed. Additionally, the reset signal (1) and the processor clock signal (1) is required. Finally a RPBA pin (1) will allow to switch the arbitration schemes. For the output the chip needs the $BR_1$ pins (4), an init flip- flop for the delay after reset (1) and a number of flip-flops for the implemented state machine (3).

Over all, the design requires 7 inputs and 8 outputs in a device with register capability. Due to the complexity of the state equations, the design can not be fitted into 20V8, instead a 22V10 has to used. This device offers furthermore two unused registered output pins, so that the device could easily handle the bus arbitration for a cluster of six ADSP-21065L. Figure 1 shows one of the possible pin assignments for the device.

```
                PALL22V10
              +----++----+
CLK_IN    --| 1  ++ 24 |--  VCC
RESET     --| 2     23 |--  INIT    REG
RPBA      --| 3     22 |--  BR1_4   REG
BR2_4     --| 4     21 |--  BR1_3   REG
BR2_3     --| 5     20 |--  BR1_2   REG
BR2_2     --| 6     19 |--  BR1_1   REG
BR2_1     --| 7     18 |--  B3      REG
NC        --| 8     17 |--  B2      REG
NC        --| 9     16 |--  B1      REG
NC        --| 10    15 |--  NC
NC        --| 11    14 |--  NC
GND       --| 12    13 |--  NC
              +----------+
```

*figure1: pinout of the PLD*

The equations for the state machine can be found attached to this document.

*References:*

- *ADSP-2106x user's manual chapter 7*
- *ADSP-21065L preliminary user's manual*
- *ADSP-21065L preliminary data sheet*
- *AMD PALASM software*
- *http://www.analog.com/sharc/attack*

```
;PALASM Design Description

;------------------------------------ Declaration Segment ------------
TITLE    Bus Arbiter for Multiple ADSP-21065L PlutO
PATTERN
REVISION 1.00.00
AUTHOR   HS
COMPANY  Analog Devices/DSP Europe
DATE     04/25/98

CHIP  _bus_ar  PAL22V10

;------------------------------------ PIN Declarations ---------------
PIN  1          Clk_in                         ; system clock
PIN  2          Reset                          ; reset line
PIN  3          RPBA                           ; fixed or /rotating
PIN  4          BR2_4                          ; from P4
PIN  5          BR2_3                          ; from P3
PIN  6          BR2_2                          ; from P2
PIN  7          BR2_1                          ; from P1
PIN  23         INIT                       REG ; delay
PIN  16         B1                         REG ; state
PIN  17         B2                         REG ; state
PIN  18         B3                         REG ; state
PIN  19         BR1_1                      REG ; to P1
PIN  20         BR1_2                      REG ; to P2
PIN  21         BR1_3                      REG ; to P3
PIN  22         BR1_4                      REG ; to P4

;------------------------------------- Boolean Equation Segment ------
STATE
MOORE_MACHINE

STATE00 = /B1 * /B2 * /B3
STATE01 = /B1 * /B2 *  B3
STATE02 = /B1 *  B2 * /B3
STATE03 = /B1 *  B2 *  B3
STATE04 =  B1 * /B2 * /B3
STATE05 =  B1 * /B2 *  B3
STATE06 =  B1 *  B2 * /B3
STATE07 =  B1 *  B2 *  B3

START_UP := POWER_UP -> STATE00

;TRANSITION EQUATIONS------------------
STATE00   :=  bf21ass    -> STATE01 ; P1 req
          +   bf22ass    -> STATE02 ; P2 req
          +   bf23ass    -> STATE03 ; P3 req
          +   bf24ass    -> STATE04 ; P4 req
          +   restart    -> STATE00 ; restart state
          +->               STATE00 ; no request
```

```
STATE01   :=  restart    -> STATE00 ; reset of state
          +   bf12grd    -> STATE02 ; fixed, P1 - P2
          +   bf13grd    -> STATE03 ; fixed, P1 - P3
          +   bf14grd    -> STATE04 ; fixed, P1 - P4
          +   bt12grd    -> STATE02 ; rotating P1 - P2
          +   bt13grd    -> STATE03 ; rotating P1 - P3
          +   bt14grd    -> STATE04 ; rotating P1 - P4
          +->               STATE01 ; access running

STATE02   :=  restart    -> STATE00 ; reset of state
          +   bf21grd    -> STATE01 ; fixed, P2 - P1
          +   bf23grd    -> STATE03 ; fixed, P2 - P3
          +   bf24grd    -> STATE04 ; fixed, P2 - P4
          +   bt23grd    -> STATE03 ; rotating P2 - P3
          +   bt24grd    -> STATE04 ; rotating P2 - P4
          +   bt21grd    -> STATE01 ; rotating P2 - P1
          +->               STATE02 ; access running

STATE03   :=  restart    -> STATE00 ; reset of state
          +   bf31grd    -> STATE01 ; fixed, P3 - P1
          +   bf32grd    -> STATE02 ; fixed, P3 - P2
          +   bf34grd    -> STATE04 ; fixed, P3 - P4
          +   bt34grd    -> STATE04 ; rotating P3 - P4
          +   bt31grd    -> STATE01 ; rotating P3 - P1
          +   bt32grd    -> STATE02 ; rotating P3 - P2
          +->               STATE03 ; access running

STATE04   :=  restart    -> STATE00 ; reset of state
          +   bf41grd    -> STATE01 ; fixed, P4 - P1
          +   bf42grd    -> STATE02 ; fixed, P4 - P2
          +   bf43grd    -> STATE03 ; fixed, P4 - P3
          +   bt41grd    -> STATE01 ; rotating P4 - P1
          +   bt42grd    -> STATE02 ; rotating P4 - P2
          +   bt43grd    -> STATE03 ; rotating P4 - P3
          +->               STATE04 ; access running

STATE05   := VCC         -> STATE00     ; future expansion
STATE06   := VCC         -> STATE00     ; future expansion
STATE07   := VCC         -> STATE00     ; future expansion

;OUTPUT EQUATIONS----------------------
STATE00.OUTF   = /BR1_1 * /BR1_2 * /BR1_3 * /BR1_4
STATE01.OUTF   =  BR1_1 * /BR1_2 * /BR1_3 * /BR1_4
STATE02.OUTF   = /BR1_1 *  BR1_2 * /BR1_3 * /BR1_4
STATE03.OUTF   = /BR1_1 * /BR1_2 *  BR1_3 * /BR1_4
STATE04.OUTF   = /BR1_1 * /BR1_2 * /BR1_3 *  BR1_4

CONDITIONS
;first branch
 bf21ass =  RESET * INIT * /BR2_1                           ; highest
 bf22ass =  RESET * INIT *  BR2_1 * /BR2_2                  ;
 bf23ass =  RESET * INIT *  BR2_1 *  BR2_2 * /BR2_3         ;
 bf24ass =  RESET * INIT *  BR2_1 *  BR2_2 *  BR2_3 * /BR2_4  ; lowest

;return for RESET
 restart =  /INIT
```

```
;returns for fixed priority
 bf12grd = RESET * INIT * BR2_1 *  RPBA *          /BR2_2
 bf13grd = RESET * INIT * BR2_1 *  RPBA *           BR2_2 * /BR2_3
 bf14grd = RESET * INIT * BR2_1 *  RPBA *           BR2_2 *  BR2_3 * /BR2_4

 bf21grd = RESET * INIT * BR2_2 *  RPBA * /BR2_1
 bf23grd = RESET * INIT * BR2_2 *  RPBA *  BR2_1 *          /BR2_3
 bf24grd = RESET * INIT * BR2_2 *  RPBA *  BR2_1 *           BR2_3 * /BR2_4

 bf31grd = RESET * INIT * BR2_3 *  RPBA * /BR2_1
 bf32grd = RESET * INIT * BR2_3 *  RPBA *  BR2_1 * /BR2_2
 bf34grd = RESET * INIT * BR2_3 *  RPBA *  BR2_1 *  BR2_2 *          /BR2_4

 bf41grd = RESET * INIT * BR2_4 *  RPBA * /BR2_1
 bf42grd = RESET * INIT * BR2_4 *  RPBA *  BR2_1 * /BR2_2
 bf43grd = RESET * INIT * BR2_4 *  RPBA *  BR2_1 *  BR2_2 * /BR2_3

;handover for rotationg priority
 bt12grd = RESET * INIT * BR2_1 * /RPBA *          /BR2_2
 bt13grd = RESET * INIT * BR2_1 * /RPBA *           BR2_2 * /BR2_3
 bt14grd = RESET * INIT * BR2_1 * /RPBA *           BR2_2 *  BR2_3 * /BR2_4

 bt21grd = RESET * INIT * BR2_2 * /RPBA * /BR2_1 *           BR2_3 *  BR2_4
 bt23grd = RESET * INIT * BR2_2 * /RPBA *                   /BR2_3
 bt24grd = RESET * INIT * BR2_2 * /RPBA *                    BR2_3 * /BR2_4

 bt31grd = RESET * INIT * BR2_3 * /RPBA * /BR2_1 *                    BR2_4
 bt32grd = RESET * INIT * BR2_3 * /RPBA *  BR2_1 * /BR2_2 *           BR2_4
 bt34grd = RESET * INIT * BR2_3 * /RPBA *                            /BR2_4

 bt41grd = RESET * INIT * BR2_4 * /RPBA * /BR2_1
 bt42grd = RESET * INIT * BR2_4 * /RPBA *  BR2_1 * /BR2_2
 bt43grd = RESET * INIT * BR2_4 * /RPBA *  BR2_1 *  BR2_2 * /BR2_3

EQUATIONS
 INIT      := RESET
 INIT.CLKF  = CLK_in

;------------------------------------ Simulation Segment ------------
SIMULATION
TRACE_ON CLK_in BR2_1 BR1_1 BR2_2 BR1_2 BR2_3 BR1_3 BR2_3 BR1_3 BR2_4 BR1_4
RESET INIT RPBA

;INIT validation (when all four /BR occur after RESET)
   SETF /BR2_1 /BR2_2 /BR2_3 /BR2_4       ; RESET after start-up
   SETF /RESET RPBA
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF RESET
   CLOCKF CLK_in
   SETF BR2_1 BR2_2 BR2_3 BR2_4
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF /BR2_1 /BR2_2 /BR2_3 /BR2_4       ; arbitration starts
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF  BR2_1                      ; P#1 finished
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF  BR2_2                      ; P#2 finished
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF  BR2_3 /BR2_2               ; P#3 finished, P#2 wants
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF  BR2_2                      ; P#2 finished
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF /BR2_1
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF  BR2_4                      ; P#4 finished
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF /BR2_1 /BR2_2 /BR2_3 /BR2_4     ; arbitration starts
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF  BR2_1 BR2_2 BR2_4              ; break of lower priority
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF BR2_1 BR2_2 BR2_3 BR2_4
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
   SETF  /BR2_4
   CLOCKF CLK_in
   CLOCKF CLK_in
   CLOCKF CLK_in
TRACE_OFF
;-------------------------------------------------------------------
```