



Interfacing ADSP-21365 SHARC® PDAP to ADSP-BF533 Blackfin® EBIU

Contributed by Srinivas K. and Kunal Singh

Rev 1 – November 12, 2004

Introduction

ADSP-21365 devices belong to the third generation of SHARC® processors. ADSP-21365 processors offer a very high bandwidth (up to 1 Gbps) input data port for data acquisition, called Parallel Data Acquisition Port (PDAP).

The PDAP supports the acquisition of data from an external device with different word lengths. The received data is packed automatically into 32-bit words. The various word lengths formats supported by PDAP include, 2x16-bit (mode 01), 4x8-bit (mode 00), 10-11-11-bit (mode 10), and 20-bit (mode 11). In 20-bit mode, the received data bits are mapped to the highest 20 bits of the 32-bit word. The lowest 12 bits of the 32-bit word are assumed to be zero.

The ADSP-BF533 Blackfin® family of processors (including the ADSP-BF531 and ADSP-BF532 derivatives) features an External Bus Interface Unit (EBIU) which provides interface to the external memory devices. The EBIU supports interface for the synchronous and asynchronous memory banks. The data bus on the ADSP-BF533 EBIU is 16 bits wide.

This EE-Note discusses a hardware interface scheme between the ADSP-21365 PDAP port and the EBIU interface on an ADSP-BF532 processor. Since the EBIU is 16 bits wide, the interface uses the PDAP in 2x16-bit format. However, the 4x8-bit and 10-11-11-bit formats

can also be implemented with minor modifications to the software.

To implement a handshake protocol, the proposed interface uses programmable flags and interrupts available on the two devices.

Since the PDAP is a unidirectional input-only port, the data can be transferred only from the ADSP-BF532 processor to the ADSP-21365 PDAP (that is, not from the PDAP to the Blackfin processor).

PDAP

The PDAP interface on ADSP-21365 SHARC processors consists of 20 PDAP data pins, a clock signal (PDAP_CLK), and a data valid signal (PDAP_HOLD). A rising or falling edge of the clock can latch the data (on the data pins) into the PDAP receive buffers (a 6-deep FIFO). The active edge of the clock is defined by the status of the IDP_PDAP_CLKEDGE bit in the PDAP control register.

The PDAP_HOLD signal is used to qualify a valid clock signal. The clock is considered to be valid if PDAP_HOLD is low in that particular clock cycle. Data on the PDAP port is acquired during every valid clock cycle.

Figure 1 depicts the data transfer protocol on PDAP port, when the port has been configured for the activity on the falling edge of the PDAP_CLK.

The data from the PDAP receive FIFO can be transferred to the internal memory of the processor using core-based transfers or DMA-based transfers. DMA transfers are completely transparent to the core.

The maximum CLK rate for the PDAP is 50 MHz. Hence, the maximum data bandwidth (1 Gbps) is achieved when the PDAP is configured for 20-bit transfers. However, since the EBIU on Blackfin processors is only 16 bits wide, the 20-bit mode cannot be used for the given interface.

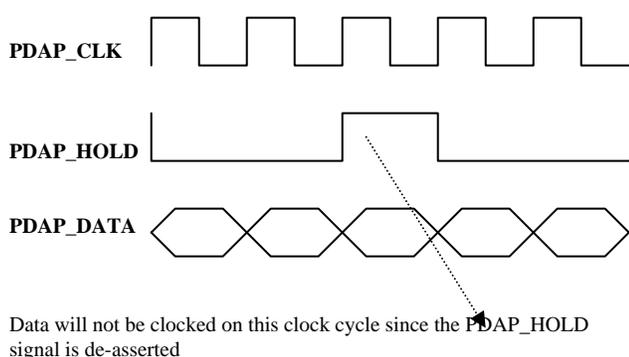


Figure 1. PDAP Port Data Transfer Protocol

As discussed before, the PDAP can be interfaced with the EBIU in mode 00, mode 01, or mode 10. The example software provided with this EE-Note covers the interface in mode 01 only.

Introduction to the Blackfin EBIU

The External Bus Interface Unit (EBIU) on Blackfin processors provides interface to external memory devices. The EBIU services requests for external memory in core mode or in DMA mode. Based on the address of the requested accesses, the accesses are controlled by the SDRAM controller or by the Asynchronous Memory Controller. The EBIU is clocked by the system clock (SCLK).

Introduction to Interrupt and Flags Signals

The given interface uses THE Interrupt signals and the GPIO flags available on the two devices as the handshake control signals.

The ADSP-21365 processor has three external asynchronous interrupts (IRQ0-2). The IRQ_x interrupts can be configured to be level-sensitive or edge-sensitive. The given interface uses the IRQ0 signal as an edge-sensitive interrupt.

ADSP-21365 SHARC processors feature 16 GPIO flags, which are available through the DAI pins. The flags can be configured as inputs to the processor or as outputs from the processor. The given interface uses the FLAG0 signal in output mode.

ADSP-BF532 processors feature 16 programmable flags (PF0-PF15.). Each PF_x signal can be configured individually as an input or an output. When the PF_x signal is configured as an input, it can be programmed to function as an asynchronous interrupt signal to the processor. The interrupt can be level-sensitive or edge-sensitive.

The given interface uses the processor's PF4 and PF6 signals. The given example also uses the PF5 signal as a user request to initiate a data transfer request. The functionality of PF5 has been added only for the purpose of testing. In the actual interface, only PF4 and PF6 are utilized.

The PF4 signal on the ADSP-BF532 processor is defined as an output FLAG. PF4 is connected to IRQ0 of the ADSP-21365 processor. PF4 is used by the ADSP-BF532 processor to request the ADSP-21365 processor for the initiation of a data transfer.

PF6 is configured as an edge-sensitive interrupt and is used as an acknowledge signal from the ADSP-21365 processor in response to a data transfer request.

In the given example code, the PF5 signal is used as an edge-sensitive interrupt to initiate a

data transfer sequence on the ADSP-BF532 processor. In a real system, the data initiation request may come from a device in the system or from the ADSP-BF532 processor itself.

PDAP to EBIU Interface

Figure 2 depicts the hardware interface scheme between the SHARC PDAP and the Blackfin EBIU.

The PDAP signals (data, clock, and hold) on ADSP-21365 processors are available through the SRU. In mode 00, `Data[15..0]` are mapped to `DAI_PIN20..5`. The clock and hold signals can be mapped to any of DAI pins 1..4. The given example uses `DAI_PIN 01` and `03`, respectively, for the clock and hold signals.

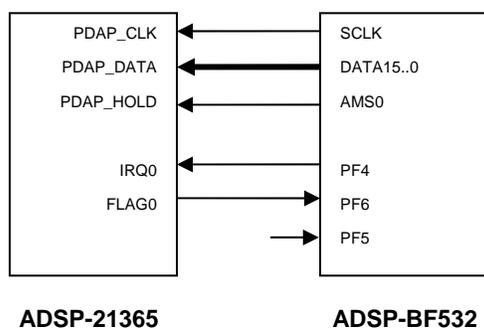


Figure 2. Hardware Interface for PDAP and Blackfin

The ADSP-BF532 processor offers different memory banks (with individual memory select signals `/AMSx`), allowing multiple memory devices to be mapped to ADSP-BF532 processors. The PDAP can be mapped to the ADSP-BF532 processor as one asynchronous memory device. The given example maps the PDAP port to the external asynchronous memory bank-0 of ADSP-BF532 processor. Therefore, `/AMS0` is connected to `PDAP_HOLD`. Thus, `/AMS0` is used to validate the `PDAP_CLK`, which in turn is the `SCLK` of the ADSP-BF532.

While implementing the above interface, ensure that the `/AMSx` signal is asserted for the duration of one `SCLK` cycle only for a single data transfer.

This requirement arises from the fact that, if the `/AMSx` signal (mapped to `PDAP_HOLD`) is asserted for more than one clock cycle, the same data would be clocked into the `PDAP FIFO`, more than once.

Data Transfer Protocol for the Interface

In the above interface all data transfers are controlled by software running on the ADSP-21365 and ADSP-BF532 processors.

A data transfer operation from the Blackfin processor to the SHARC processor consists of two phases, which are completely implemented in software:

- **Command_Phase:** The ADSP-BF532 processor passes the various control parameters (e.g., number of data words, internal memory address for the ADSP-21365 processor where the data has to be stored, and so on). In the given example code, the ADSP-21365 processor uses the above control words as DMA parameters for the PDAP DMA. The example software can be modified to include additional control variable (e.g., PDAP mode, DMA modifier, and so on).
- **Data_Phase:** After receiving the control parameters from the ADSP-BF532 processor, the ADSP-21365 processor configures its PDAP port for the data transfers.

Synchronization Between the ADSP-21365 Processors and the ADSP-BF532 Processors

ADSP-BF532 Blackfin processor can initiate data transfers by requesting a transfer from the ADSP-21365 processor. The ADSP-BF532 processor software achieves this by generating an interrupt signal (`IRQ0`) for the ADSP-21365 processor through its `PF4` pin.

The ADSP-21365 processor will jump to the `IRQ0` interrupt service routine. Inside the ISR, if a earlier data transfer operation (requested by the ADSP-BF532 processor) is not pending, the ADSP-21365 processor will configure the PDAP port to receive control words from the ADSP-BF532 processor. After configuring the PDAP port, the ADSP-21365 processor would pass an acknowledgement to the ADSP-BF532 processor by asserting `FLAG0`.

A rising edge on `FLAG0` causes the ADSP-BF532 processor to interrupt its program execution and respond to the interrupt request on `PF6` (which has been mapped to `IVG12`). Inside the ISR for `IVG12`, the ADSP-BF532 processor will transfer the control words to the ADSP-21365 processor through its EBIU.

Upon receiving the control words, the ADSP-21365 processor processes the PDAP DMA "over" interrupt. Inside the interrupt service routine, the ADSP-21365 processor re-configures the PDAP for data transfers. The DMA parameters are configured, based on the control words received. After the PDAP is configured the ADSP-21365 processor passes an acknowledgement to the ADSP-BF532 processor again (through `FLAG0`), instructing the ADSP-BF532 processor to perform the data transfer operations.

It is interesting to note that the same pins (`FLAG0` and `PF6`) are used as the acknowledgement signal to start the `command_phase` and `data_phase`. The ADSP-BF532 processor identifies the operation to be performed (whether to transfer data or control words), based on flag variables maintained by the software running on the ADSP-BF532 processor.

Driver Software for ADSP-21365 SHARC Processors

The ADSP-21365 processor would receive a new data transfer request from the ADSP-BF532 processor through the `IRQ0` signal. If a new request is received while an old request is

pending, the ADSP-21365 processor considers the new request to be invalid and does not respond to the new request. In a real-time system, a corrective action must be taken when an invalid request is identified.

When a valid data request from an ADSP-BF532 processor is encountered, the ADSP-21365 processor initiates the `command_phase`. The `command_phase` involves passing control parameters (memory index and DMA count for the data transfers) from the ADSP-BF532 processor to ADSP-21365 processor. The ADSP-21365 processor would configure the PDAP DMAs to receive the control words from the ADSP-BF532 processor. In the given code example, only memory index and DMA count values are passed as control parameters. However, the given example can be extended to pass more parameters, such as DMA modifier, PDAP mode, and so on.

After configuring the PDAP in DMA mode to receive control words, the ADSP-21365 processor passes an acknowledgement to the ADSP-BF532 processor.

The PDAP DMA "over" interrupt is generated upon completion of the `command_phase`. The ADSP-21365 processor reads these control parameters and re-configures the PDAP for the `data_phase`. Again, an acknowledgement is passed to the ADSP-BF532 processor to initiate the `data_phase`. The ADSP-BF532 processor responds to the acknowledgement by sending the actual data.

Driver Software for ADSP-BF532 Blackfin Processors

As discussed, ADSP-BF532 processors can initiate a data transfer sequence by generating `IRQ0` for ADSP-21365 processors. In the given example, you can initiate data transfers by interrupting the ADSP-BF532 processor through `PF5`, which has been configured as an edge-triggered interrupt. In the given example code,

you are asked to pass the control parameters (DMA parameters) through VisualDSP++ I/O (Output window's Console page). However, in a real-time system, the control parameters arise from the ADSP-BF532 processor. After receiving the control words from the console, the ADSP-BF532 processor requests the ADSP-21365 processor to initiate a data transfer sequence by interrupting the ADSP-21365 processor by PF4. The ADSP-BF532 processor sets the `request_flag` and `command_flag` before asserting PF0. If all of the previous data transfers have been completed, the ADSP-21365 processor responds to the request with an acknowledge signal.

When an acknowledgement is received, the ADSP-BF532 processor checks for the `command_flag` and `request_flag`.

- If both these variables are set, a command phase is initiated which involves writing the control parameters through the EBIU. The `command_flag` is cleared after completing the command phase.
- If `request_flag` is set and the `command_flag` is cleared, the ADSP-BF532 processor initiates the data transfer phase which involves writing the actual data block to the PDAP through EBIU.
- If `request_flag` is cleared, no action is taken.

Appendix

All example code is available through a ZIP file associated with this EE-Note.

Blackfin Example Code

```

/*****/
/*                                     */
/* Code on ADSP-BF532 for "PDAP-BF Interface */
/*                                     */
/*                                     */
/* - Initial Version                   */
/*       7th August, 2004              */
/*                                     */
/*           kunal singh                */
/*           DSP Apps Engineer          */
/*           Analog Devices             */
/*                                     */
/*****/

#include <stdio.h>
#include <cdefBF532.h>
#include <signal.h>
#include <sys/exception.h>

#define BUFFER_SIZE 256
/*****/
/*                                     */
/* Size of the Data Buffer               */
/*                                     */
/* This is also the maximum limit on   */
/* the DMA count for a single transfer */
/*                                     */
/*****/

volatile short *AsyncMem ;

EX_INTERRUPT_HANDLER(request);
EX_INTERRUPT_HANDLER(acknowledge);
/*****/
/*                                     */
/* Interrupt Handlers                   */
/*                                     */
/* request() : Edge sensitive interrupt on */
/*           PF5, mapped to IVG11         */
/*                                     */
/*                                     */
/* acknowledge() : Edge sensitive interrupt */
/*           on PF5, mapped to IVG12     */
/*                                     */
/*                                     */
/*****/

void InitAsyncMemory();
void Init_Data();

```

```

void initialize_PF_intrpts();
void generate_interrupt();
void drive_data ();
/*****/
/* */
/* Function Declarations */
/* */
/*****/

unsigned int dataSrc[BUFFER_SIZE] ;
/*****/
/* */
/* Data Buffer for DMA transfers */
/* */
/*****/

unsigned int command_flag;
unsigned int request_flag;
/*****/
/* */
/* Flags to keep track of command */
/* and Data phase */
/* */
/*****/

unsigned int dma_count;
unsigned int dma_index;
/*****/
/* */
/* Variables to hold DMA parameters */
/* */
/*****/

unsigned int temp;
/*****/
/* */
/* Variable for Temporary Storage */
/* */
/*****/

/*****/
/* */
/* Main Function */
/* */
/*****/
main ()
{
    InitAsyncMemory () ;
    // Initialize the EBIU

    Init_Data();
    // Intialize the Data Buffer
    // with the data values to be transfered

    initialize_PF_intrpts();
    // Configure the PF4 as o/p
    // and PF,PF6 as positive edge triggered interrupts

```

```

register_handler(ik_ivg11, request);
register_handler(ik_ivg12, acknowledge);
// Define interrupt handler for IVG11, IVG12
// IVG11: mapped to PF5
// IVG12: mapped to PF6

while (1)
{
}

/*****/
/*
/* Initialize the EBIU Registers for
/* Asynchronous Memory Bank
/*
/* The Asyn Mem Signal are connected
/* to the PDAP
/*
/*****/
void InitAsyncMemory(void)
{
    (*pEBIU_AMGCTL) = 0x3 ;
    // Enable clock out for Async accesses
    // Enable only Bank 0

    (*pEBIU_AMBCTL0) = 0x00001110 ;
    // Set up bank 0 parameters --
    // Setup time = 1 cycle
    // Write access = 1 cycle
    // Hold = 0
}

void Init_Data()
{
    int i;

    for (i=0; i<BUFFER_SIZE; ++i)
    {
        dataSrc[i] = 0xcafebead + i;
    }
    // Initialize the Data Buffer
}

void initialize_PF_intrpts()
{
    // configure PF4 as output
    // configure PF5 and PF6 as inputs
    // PF5 and PF6 are rising edge sensitive interrupts
    // PF5 is mapped to intr_A/IVG11 and PF6 is mapped to intr_B/IVG12

    temp = *pFIO_DIR;
    temp &= 0xFF9F;
    temp |= 0x0010;
    *pFIO_DIR = temp;
    // Define PF4 as output
    // Define PF5, PF6 as inputs

```

```

*pFIO_INEN = 0x0060;
// enable the i/p buffer for PF5, PF6

*pFIO_MASKA_D = 0x0020;
*pFIO_MASKB_D = 0x0060;
// enable the PF5 for the Interrupt A
// enable the PF6 for the Interrupt B

*pFIO_POLAR = 0;
// set the active-high/rising-edge polarity

*pFIO_EDGE = 0x0060;
*pFIO_BOTH = 0;
// edge sensitive interrupt for PF5, PF6
// latch interrupt only on a single edge

temp = *pSIC_IMASK;
temp |= 0x00180000;
*pSIC_IMASK = temp;
// enable the PF intr A and B

temp = *pSIC_IAR2;
temp &= 0xFFF00FFF;
temp |= 0x00054000;
*pSIC_IAR2 = temp;
// map the PF intr A to ivg11, and PF intr B to ivg12
}

EX_INTERRUPT_HANDLER(request)
{
    printf("\n enter the address location in the ADSP-21365\n");
    scanf("%u",&dma_index);

    printf("\n enter the number of transfers \n");
    scanf("%u",&dma_count);

    request_flag = 1;
    command_flag = 1;

    generate_interrupt();
    // Generate Interrupt Request for ADSP-21365
    // By toggling the PF4
}

void generate_interrupt()
{
    int wait;

    *pFIO_FLAG_C = 0x0010;
    // Clear the PF4

    for (wait = 0; wait<40; wait++)
    {
    }

    *pFIO_FLAG_S = 0x0010;
    // Set the PF4
    // Causes a low-high-low transition on PF4
}

```

```

}
EX_INTERRUPT_HANDLER(acknowledge)
{
    if (request_flag == 1)
        // Valid Acknowledge
        {
            if(command_flag == 1)
                {
                    // send command word to the ADSP-21365

                    (*AsyncMem) = dma_index & 0xffff ;
                    (*AsyncMem) = (dma_index >> 16) & 0xffff ;
                    // send dma_index

                    (*AsyncMem) = dma_count & 0xffff ;
                    (*AsyncMem) = (dma_count >> 16) & 0xffff ;
                    // send dma_count

                    command_flag = 0;
                }
            else
                {
                    drive_data ();
                    // send data to the ADSP-21365
                    request_flag = 0;
                }
        }
    else
        // Invalid Acknowledge Interrupt
        {
            printf("\n An error has ocured while transmitting\n");
            request_flag = 0;
            command_flag = 0;
        }
}

void drive_data ()
{
    int i = 0 ;

    // Point to the external async memory bank
    AsyncMem = (volatile short *)0x20000000 ;

    for (i = 0; i < dma_count; ++i)
        {
            (*AsyncMem) = dataSrc[i] & 0xffff ;
            (*AsyncMem) = (dataSrc[i] >> 16) & 0xffff ;
        }
}

```

Listing 1. Code Snippet for ADSP-BF532 Blackfin Processor

SHARC Example Code

```

/*****

```

```

/*
/* Code on ADSP-21365 for "PDAP-BF Interface
/*
/*
/* - Initial Version
/* 7th August, 2004
/*
/* kunal singh
/* DSP Apps Engineer
/* Analog Devices
/*
/*****/
#include <Cdef21365.h>
#include <def21365.h>
#include <stdio.h>
#include <signal.h>
#include <sru2136x_new.h>

/*****/
/*
/* This Example code has been developed
/* for interfacing the PDAP port of the ADSP-21365
/* with ADSP-BF532 External Port
/*
/* The PDAP would operate in MODE00 (2x16bit)
/*
/* The External Connections required between the
/* ADSP-21365 and ADSP-BF532 are
/*
/* ADSP-21365 ADSP-BF532
/*
/* DAI_PIN05..20 (PDAP_DAT) <-- D0..D15
/* DAI_PIN01 (PDAP_CLK) <-- SCLK0
/* DAI_PIN03 (PDAP_HOLD) <-- AMS0
/* IRQ0 <-- PF4
/* FLAG0 --> PF6
/*
/* PDAP would be configured to latch the
/* data at the negative edge of the PDAP_CLK
/* This is in accordance with the data/clock relationship
/* available on ADSP-BF532
/*
/* The data transfers from PDAP FIFO to the internal
/* memory would be done with DMAs
/* Simple (No Ping Pong) DMAs would be used
/*
/*****/

void dai_low_isr(int i);
void irq0_low_isr(int i);

/*****/
/*
/* dai_low_isr(): The PDAP DMA over interrupt
/*
/* irq0_low_isr(): ISR for the IRQ0
/*
/*****/

```

```

void Enable_PDAP(void);
/*****/
/*                               */
/* Enable the PDAP to receive data at*/
/* Memory index <- dma_index      */
/* Count <- dma_count            */
/*                               */
/*****/

volatile unsigned int pdap_command[2];
/*****/
/*                               */
/* Control data would be stored in the */
/* above memory buffer                */
/* The first element corresponds to the */
/* memory address for the received data */
/* The second element corresponds to the */
/* count of DMA transfers              */
/*                               */
/*****/

volatile unsigned int command_flag;
/*****/
/*                               */
/* ADSP-BF532 would interrupt (IRQ0) the ADSP-21365 */
/*                               */
/* If command_flag == 0 =>          */
/* a command word has to be received */
/*                               */
/* Else                               */
/* The Interrupt would be ingnored   */
/*                               */
/*****/

volatile unsigned int dma_index;
volatile unsigned int dma_count;
/*****/
/*                               */
/* DMA parameters for the PDAP channel */
/*                               */
/* dma_index : IDP_DMA_I0            */
/* dma_count : IDP_DMA_C0Count       */
/*                               */
/*****/

unsigned int temp;
/*****/
/*                               */
/* Variable for Temporary Storage */
/*                               */
/*****/

/*****/
/*                               */
/* Main Function                    */
/*                               */
/*****/

```

```

void main()
{
    command_flag == 0;
    // Initially command_flag == 0;

    interrupt(SIG_P12,dai_low_isr);
    // Interrupt handler for PDAP DMA over interrupt

    asm("#include <def21365.h>");
    asm("bit set mode2 IRQ0E;");
    // Define IRQ0 as edge sensitive

    interrupt(SIG_IRQ0,irq0_low_isr);
    // interrupt handler for IRQ0

    asm("bit set FLAGS 0x2;");
    // configure the FLAG0 as o/p

    asm("bit clr FLAGS 0x1;");
    // clear the FLAG0

    // Disable the o/p drivers on the DAI pins
    // To ensure that these are not driven by the DSP
    SRU (LOW, PBEN01_I);
    SRU (LOW, PBEN02_I);
    SRU (LOW, PBEN03_I);
    SRU (LOW, PBEN04_I);
    SRU (LOW, PBEN05_I);
    SRU (LOW, PBEN06_I);
    SRU (LOW, PBEN07_I);
    SRU (LOW, PBEN08_I);
    SRU (LOW, PBEN09_I);
    SRU (LOW, PBEN10_I);
    SRU (LOW, PBEN11_I);
    SRU (LOW, PBEN12_I);
    SRU (LOW, PBEN13_I);
    SRU (LOW, PBEN14_I);
    SRU (LOW, PBEN15_I);
    SRU (LOW, PBEN16_I);
    SRU (LOW, PBEN17_I);
    SRU (LOW, PBEN18_I);
    SRU (LOW, PBEN19_I);
    SRU (LOW, PBEN20_I);

    SRU (DAI_PB01_O, IDPO_CLK_I);
    SRU (DAI_PB03_O, IDPO_FS_I);
    // Configure the SRU for PDAP_CLK and PDAP_HOLD

    while(1)
    {
    }
}

void irq0_low_isr(int i)
{
    if(command_flag == 0)
    // A valid Data Request (no previous transfers pending)

```

```

{
    command_flag =1;

    dma_index = (unsigned int) pdap_command;
    dma_count = 0x2;

    Enable_PDAP();

    asm("bit clr FLAGS 0x1;");
    asm("bit set FLAGS 0x1;");
    asm("bit clr FLAGS 0x1;");
    // Toggle the FLAG0 from low to high
    // This would interrupt the ADSP-BF532
}
else
// Previous transfers are pending
{

    printf("\n an error has occured\n");
    // corrective actions can be taken
}
}

void dai_low_isr(int i)
{
    int j;

    temp = *pDAI_IRPTL_L;
    // Read the DAI interrupt

    if ( temp == IDP_DMA0_INT)
    // confirm the source of the interrupt
    // before processing the ISR
    {
        if(command_flag == 1)
        // A Command word has been received
        {
            command_flag = 0;

            dma_index = pdap_command[0];
            dma_count = pdap_command[1];

            Enable_PDAP();

            asm("bit clr FLAGS 0x1;");
            asm("bit set FLAGS 0x1;");
            asm("bit clr FLAGS 0x1;");
            // Toggle the FLAG0 from low to high
            // This would interrupt the ADSP-BF532
            // Toggle the flag1 from low to high
            // This would interrupt the ADSP-BF532
        }
    }
else
{
    // Data has been received
    // Any action can be taken here
    // For example you can process this data
    // However this example does not
}
}

```

```

        // include any data processing
    }
}

void Enable_PDAP(void)
{
    *pIDP_CTL0 = 0;
    *pIDP_CTL1 = 0;
    // disable the CIDP globally

    *pIDP_PP_CTL = 0;
    // clear the PDAP control register

    *pIDP_PP_CTL |= IDP_PDAP_RESET;
    // reset the PDAP state machine

    *pIDP_PP_CTL |= IDP_PDAP_PACKING0;
    // select the mode00

    // Enable the PDAP Mask bits for DAI_PIN05..20
    *pIDP_PP_CTL |= IDP_P20_PDAPMASK |
        IDP_P19_PDAPMASK |
        IDP_P18_PDAPMASK |
        IDP_P17_PDAPMASK |
        IDP_P16_PDAPMASK |
        IDP_P15_PDAPMASK |
        IDP_P14_PDAPMASK |
        IDP_P13_PDAPMASK |
        IDP_P12_PDAPMASK |
        IDP_P11_PDAPMASK |
        IDP_P10_PDAPMASK |
        IDP_P09_PDAPMASK |
        IDP_P08_PDAPMASK |
        IDP_P07_PDAPMASK |
        IDP_P06_PDAPMASK |
        IDP_P05_PDAPMASK;

    *pIDP_PP_CTL |= IDP_PDAP_CLKEDGE;
    // define the active clock edge for the PDAP

    *pIDP_PP_CTL |= IDP_PDAP_EN ;
    // enable the PDAP

    *pIDP_DMA_M0 = 0x1;
    *pIDP_DMA_C0 = dma_count;
    *pIDP_DMA_I0 = dma_index;
    // configure PDAP DMA parameter registers

    *pIDP_CTL1 |= IDP_EN0 | IDP_DMA_EN0;
    *pIDP_CTL0 |= IDP_DMA_EN | IDP_EN;
    // Enable the PDAP with DMAs
}

```

Listing 2. Code Snippet for ADSP-21365 SHARC Processor

References

- [1] *ADSP-21365 SHARC Processor PDAP Specifications*
- [2] *ADSP-BF533 Blackfin Processor Hardware Reference*. Rev. 3.0, September 2004. Analog Devices Inc.

Document History

Revision	Description
<i>Rev 1 – November 12, 2004 by Srinivas K. and Kunal Singh</i>	Initial Release