4 Data Addressing

Overview

The ADSP-21160's two data address generators (DAGs) simplify the task of organizing data by maintaining pointers into memory. The DAGs allow both processing elements to address memory indirectly; that is, an instruction specifies a DAG register containing an address instead of the address value itself.

Data address generator 1 (DAG1) generates 32-bit addresses on the DM Address Bus. Data address generator 2 (DAG2) generates 32-bit addresses on the PM Address Bus. The basic architecture for both DAGs is shown in Figure 4.1 of the SHARC Users Manual.

The DAGs also support in hardware some functions commonly used in digital signal processing algorithms. Both DAGs support circular data buffers, which require advancing a pointer repetitively through a range of memory. Both DAGs can also perform a bit-reversing operation, which outputs the bits of an address in reversed order.

The DAGs also support a register load broadcast mode that provides for the distribution of constant data to both processing elements.

DAG Registers

This feature on the ADSP-21160 operates the same as this feature on the ADSP-2106x family DSPs. For more information, see the corresponding section in the ADSP-2106x SHARC User's Manual.

DAG Operation

DAG operations include:

- address output with pre-modify or post-modify,
- modulo addressing (for circular buffers),
- bit-reversed addressing, and
- dual processing element data broadcast register loads

Short word addresses (for 16-bit data) are right-shifted by two bits before being output onto the DM Address Bus. 32-bit normal word addresses (for 32-bit data) are right-shifted by one bit before being output onto the DM Address Bus. Extended precision normal word addresses (for 40-bit data) are remapped from the 48-bit address space to a 64-bit word address space. These address adjustment operations allow internal memory to use the address directly. For details on these address adjustments, see "Memory" on page 7-1.

Address Output & Modification

This feature on the ADSP-21160 operates the same as this feature on the ADSP-2106x family DSPs. For more information, see the corresponding section in the ADSP-2106x SHARC User's Manual.

Circular Buffer Addressing

The DAGs provide for addressing of locations within a circular data buffer. A circular buffer is a set of memory locations that stores data. An index pointer steps through the buffer, being post-modified and updated by the addition of a specified value (positive or negative) for each step. If the modified address pointer falls outside the buffer, the length of the buffer is subtracted from or added to the value, as required to wrap the index pointer back to the start of the buffer (see Figure 4.4 in the ADSP-2106x SHARC User's Manual). There are no restrictions on the value of the base address for a circular buffer.

Circular buffer addressing must use `M` registers for post-modify of `I` registers, not pre-modify. Pre-modify addressing for non-circular buffer memory accesses is still allowed.

Circular Buffer Operation

Circular buffering is enabled by setting the global `CBUFEN` bit in the `MODE1` system register. This bit has a corresponding mask bit in the `MMASK` system register. Setting the corresponding `MMASK` bit causes the `CBUFEN` bit to be cleared following a status PUSH instruction, or the execution of an external interrupt, timer interrupt, or vectored interrupt. This feature can be used to disable circular buffering easily while in an ISR that does not need this function. By disabling circular buffering with the global bit, the ISR does not need to save off (and restore) the `B` and `L` registers.

Clearing the `CBUFEN` bit disables circular buffering operation for all data load and store operations. The DAGs will perform normal post-modify load and store accesses instead. Note that a write to a `B` register will still modify the corresponding `I` register, independent of the state of the `CBUFEN` bit. The `MODIFY` instruction will always execute independent of the state of the `CBUFEN` bit; the `MODIFY` instruction will always perform circular buffering modification of the index registers (if the corresponding `B` and `L` registers are appropriately configured).

Set up a circular buffer in assembly language by initializing an `L` register with a positive, non-zero value and loading the corresponding (same-numbered) `B` register with the base (starting) address of the buffer. The corresponding `I` register is automatically loaded with this same starting address.

On the first post-modify access using the `I` register, the DAG outputs the `I` register value on the address bus and then modifies it by adding the specified `M` register or immediate value to it. If the modified value is within the buffer range, it is written back to the `I` register. If the value is outside the buffer range, the `L` register value is subtracted (or, if the modify value is negative, added) first.

Circular Buffer Registers

This feature on the ADSP-21160 operates the same as this feature on the ADSP-2106x family DSPs. For more information, see the corresponding section in the ADSP-2106x SHARC User's Manual.

Circular Buffer Overflow Interrupts

(STKY replaced by STKYx, the overflow status bits appear in STKYx only and are not repeated in STKYy).

Bit Reversal

This feature on the ADSP-21160 operates the same as this feature on the ADSP-2106x family DSPs. For more information, see the corresponding section in the ADSP-2106x SHARC User's Manual.

Dual PE Register Load Broadcasts

When the BDCST9 bit in the MODE1 system register is set, data register loads from the PM bus that use the I9 DAG2 index register are "broadcast" to a register or register pair in each processing element (PE). For example, if the register load to R0 is requested, then both R0 in PEx and S0 in PEy will receive the same data value.

When the BDCST1 bit is set, data register loads from the DM bus that use the I1 DAG1 index register are "broadcast" to a register or register pair in each PE.

Enabling either DAG1 or DAG2 register load broadcasting has no affect on register stores, or loads to UREG registers other than the register file data registers. Both MODE1 bits are cleared following chip reset.

Table 4-1 summarizes the affects of a register load operation on both PE's with register load broadcasting enabled.

Table 4-1. Dual PE Register Load Broadcasts

Instruction	PEx operation	PEy operation

`Rx = DM(I1,Ma);`
`Rx = PM(I9,Mb);`
`Rx = DM(I1,Ma),`
`Rx = PM(I9,Mb);`
	`Rx = DM(I1,Ma);`
	`Rx = PM(I9,Mb);`
	`Rx = DM(I1,Ma),`
	`Rx = PM(I9,Mb);`
		`Sx = DM(I1,Ma);`
		`Sx = PM(I9,Mb);`
		`Sx = DM(I1,Ma),`
		`Sx = PM(I9,Mb);`

DAG Register Transfers

DAG registers are part of the universal register set and may be loaded from memory, from another universal register, or from an immediate field in an instruction. DAG register contents may be stored to memory or to a universal register.

Instruction Type 5a, SIMD mode (bidirectional) register to register transfers between a register file data register and one of the DAG registers is allowed. When the DAG register is a source of the transfer, the destination can be a register file data register. This results in the contents of the single source register being duplicated in a data register in both PE's.

Use care in the case where the DAG register is a destination of a transfer from a register file data register source. Make sure that a conditional operation is used to select either one PE, or neither PE as the source. Having both PE's contribute a source value will result in the PEx write having precedence over the PEy write.

In the case where a DAG register is both source and destination, the data move operation will execute the same as it would if SIMD mode were disabled (PEYEN cleared).

32-bit normal word addressed transfers between memory and 32-bit DAG1or DAG2 registers using the 64-bit DM and PM data buses are aligned to the low order bits of the buses. Figure 4-1 illustrates these transfers.

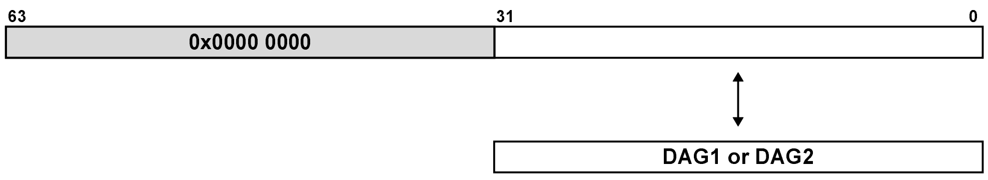

Figure 4-1. 32-bit Normal Word DAG Register Memory Transfers

40-bit normal word addressed transfers or register to register transfers between a 40-bit data register and 32-bit DAG1 or DAG2 registers using the 64-bit DM and PM data buses are aligned to bits 32-63 of the buses. Figure 4-2 illustrates these transfers.

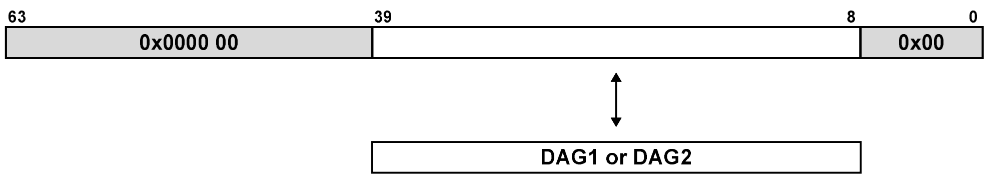

Figure 4-2. DAG Register to Data Register Transfers

Long word addressed transfers between memory and 32-bit DAG1 or DAG2 registers using the 64-bit DM and PM data buses target double DAG registers. Figure 4-3 illustrates how the bus is utilized in these transfers. If the transfer specifies an even-numbered DAG register (e.g., `I0` or `I2`), then the even numbered register value transfers on the lower half of the 64-bit bus, and the even numbered register + 1 value transfers on the upper half (bits 63-32) of the bus. If, however, the instruction specifies an odd numbered register (e.g., `I1`, or `B3`), the odd numbered register value transfers on the lower half of the 64-bit bus, and the odd numbered register - 1 value (`I0` or `B2` in this example) transfers on the upper half (bits 63-32) of the bus. In either case, the explicitly specified register (even or odd) sources or sinks bits 31-0 of the long word addressed memory.

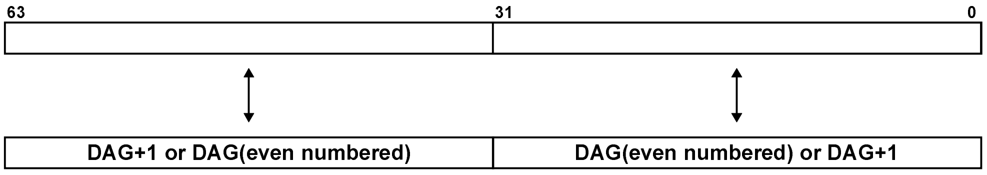

Figure 4-3. Long Word DAG Register to Data Register Transfers

DAG Register Transfer Restrictions

ADSP-21160 has similar DAG register transfer restrictions to the ADSP-2106x family DSPs (see the corresponding section in the ADSP-2106x SHARC User's Manual). The same two classes of restrictions exist: (1) automatic insertion of NOP cycle following DAG register loads and (2) potential incorrect results undetected by hardware. ADSP-21160 has changed the

DAG register loads restriction to further minimize cases in which the NOP cycle insertion occurs, per the following description. The undetected errors restriction is unchanged from ADSP-2106x.

When an instruction that loads a DAG register is followed by an instruction that uses that register (or the adjacent register of the register pair) in the same DAG for data addressing, modify instructions, or indirect jumps, the ADSP-21160 inserts an extra (NOP) cycle between the two instructions. Examples:

```
L2=8;
DM(I0,M1)=R1;
```

This is the example case in the ADSP-2106x SHARC User's Manual. This case does not stall the ADSP-21160. However, the following case will:

```
I0=8;
DM(I0,M1)=R1;
```

This is a true write/read dependency in which I0 is written in one cycle; therefore, the results of that register write are not available to a register read for one cycle. Note that if either instruction had specified I1, the stall would still occur. This is because the ADSP-21160 DAG register transfers can occur in pairs; therefore write/read dependencies are detected with a register pair granularity. For more information, see Figure 4-3.