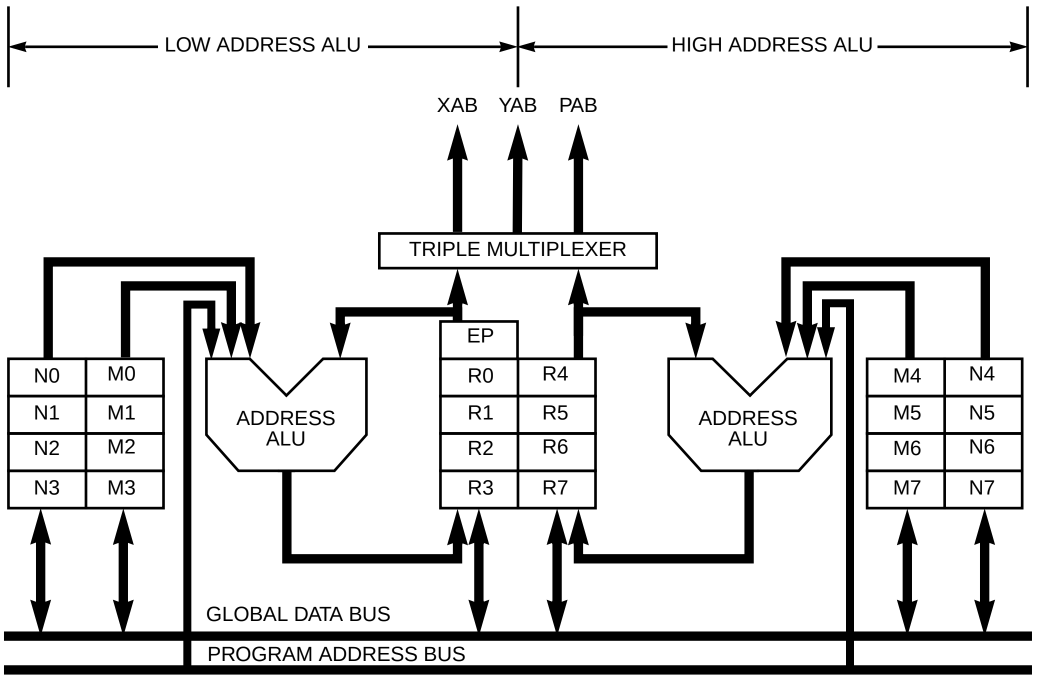
# 4 ADDRESS GENERATION UNIT

## 4.1 AGU ARCHITECTURE

The AGU is one of the three execution units on the DSP56300 Core. The AGU performs the effective address calculations (using integer arithmetic) necessary to address data operands in memory and contains the registers used to generate the addresses. It implements four types of arithmetic: linear, modulo, multiple wrap-around modulo and reverse-carry and operates in parallel with other chip resources to minimize address-generation overhead. The AGU is divided into two halves, each of which has an address arithmetic logic unit (ALU) and four sets of registers (see Figure 4-1).

**Figure 4-1. AGU Block Diagram**



These registers are the address registers (R0 - R3 and R4 - R7), offset registers (N0 - N3 and N4 - N7), and the modifier registers (M0 - M3 and M4 - M7). The eight Rn, Nn, and Mn registers are treated as register triplets — e.g., only N2 and M2 can be used to update R2. The eight triplets are R0:N0:M0, R1:N1:M1, R2:N2:M2, R3:N3:M3, R4:N4:M4, R5:N5:M5, R6:N6:M6, and R7:N7:M7. Each register may be read or written by the global data bus (GDB).

The two arithmetic units can generate two 24-bit addresses every instruction cycle — one for any two of the XAB and YAB, or one PAB address. The AGU can directly address 16,777,216 locations on the XAB, 16,777,216 locations on the YAB, and 16,777,216 locations on the PAB. The two independent address ALUs work with the two data memories to feed two operands to the data ALU in a single cycle. Each operand may be addressed by an Rn, Nn, and Mn triplet.

The two address ALUs are identical (see Figure 4-1); each one of them contains a 24-bit full adder (called offset adder), which can add 1) plus one, 2) minus one, 3) the contents of the respective offset register N, or 4) minus N to the contents of the selected address register. A second full adder (called a modulo adder) adds the summed result of the first full adder to a modulo value, M or minus M, where M is stored in the respective modifier register. A third full adder (called a reverse-carry adder) can add 1) plus one, 2) minus one, 3) the offset N (stored in the respective offset register), or 4)minus N to the selected address register with the carry propagating in the reverse direction — i.e., from the most significant bit (MSB) to the least significant bit (LSB). The offset adder and the reverse-carry adder are in parallel and share common inputs. The only difference between them is that the carry propagates in opposite directions. Test logic determines which of the three summed results of the full adders is output.

Each address ALU can update one address register, Rn, from its respective address register file during one instruction cycle. The contents of the selected modifier register specify the type of arithmetic to be used in an address register update calculation. The modifier value is decoded in the address ALU.

The address output multiplexers (see Figure 4-1) select the source for the XAB, YAB, and PAB. These multiplexers allow the XAB, YAB, or PAB outputs to originate from R0 - R3 or R4 - R7.

## 4.2 SIXTEEN-BIT COMPATIBILITY MODE

When the SIXTEEN-BIT COMPATIBILITY mode bit (see Figure 6-5 on page 6-10) is turned on, the following occur in the AGU:

- Move operations to/from any of the AGU registers (R0-R7, N0-N7 and M0-M7) clear the 8 MSBits of the destination.
- The 8 MSBits of any AGU address calculation result are cleared.
- The sign bit of the selected N register is bit15 instead of bit 23.
- The 8 MSBits of the address are ignored in the calculations of memory regions.

Note: Proper memory access operation is not guaranteed if an address register had non-zero bits in its 8 MSBits before changing to 16 bit compatibility mode. This is because the 8 MSBits will not be cleared when the register is the source for the address.

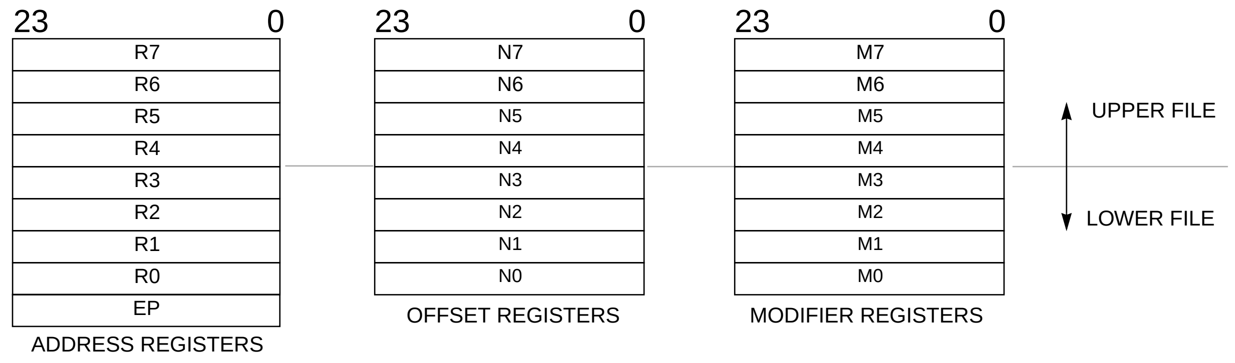
This mode of operation supports compatibility to object code written for the DSP56000 Family of Digital Signal Processors.

Due to pipelining, a change in the bit takes affect only after the following three instruction cycles. Inserting three NOP instructions after the instruction that changes the value of this bit will ensure proper operation.

## 4.3 PROGRAMMING MODEL

The programmer's view of the AGU is eight sets of three registers (see Figure 4-2). These registers can be used as temporary data registers and indirect memory pointers. Automatic updating is available when using address register indirect addressing. The Rn registers can be programmed for linear addressing, modulo addressing (regular or multiple wrap-around), and bit-reverse addressing.

**Figure 4-2. AGU Programming Model**



### 4.3.1 Address Register Files (R0 - R3, EP and R4 - R7)

The eight 24-bit address registers, R0 - R7, can contain addresses or general-purpose data. The 24-bit address in a selected address register is used in the calculation of the effective address of an operand. When supporting parallel X and Y data memory moves, the address registers must be thought of as two separate files, R0 - R3 and R4 - R7. The contents of an Rn may point directly to data or may be offset. In addition, Rn can be pre-updated or post-updated according to the addressing mode selected. If an Rn is updated, modifier registers, Mn, are always used to specify the type of update arithmetic. Offset registers, Nn, are used for the update-by-offset addressing modes. The address register modification is performed by one of the two modulo arithmetic units. Most addressing modes modify the selected address register in a read-modify-write fashion; the address register is read, its contents are modified by the associated modulo arithmetic unit, and the register is written with the appropriate output of the modulo arithmetic unit. The form of address register modification performed by the modulo arithmetic unit is controlled by the contents of the offset and modifier registers discussed in the following paragraphs.

#### 4.3.1.1 Stack Extension Pointer (EP)

The contents of the 24-bit EP register is used to point to the stack extension in data memory whenever the stack extension is enabled and move operations to/from the on-chip hardware stack are needed. The EP register is a read/write register and is referenced implicitly by some instructions (DO, JSR, RTI, etc.) or directly e.g. by using the MOVEC instruction. The stack extension pointer is not initialized during hardware reset, and must be set (using a MOVEC instruction) prior to enabling the stack extension. For a more detailed description of the stack extension mode of operation, please refer to Section 6.3.5.

### 4.3.2 Offset Register Files (N0 - N3 and N4 - N7)

The eight 24-bit offset registers, N0 - N7, can contain offset values used to increment/decrement address registers in address register update calculations or can be used for 24-bit general-purpose storage. For example, the contents of an offset register can be used to step through a table at some rate (e.g., five locations per step for waveform generation), or the contents can specify the offset into a table or the base of the table for indexed addressing. Each address register, Rn, has its own offset register, Nn, associated with it.

### 4.3.3 Modifier Register Files (M0 - M3 and M4 - M7)

The eight 24-bit modifier registers, M0 - M7, define the type of address arithmetic to be performed for addressing mode calculations, or they can be used for general-purpose storage. The address ALU supports linear, modulo, and reverse-carry arithmetic types for all address register indirect addressing modes. For modulo arithmetic, the contents of Mn also specify the modulus. Each address register, Rn, has its own modifier register, Mn, associated with it. Each modifier register is set to $FFFFFF on processor reset, which specifies linear arithmetic as the default type for address register update calculations.

## 4.4 ADDRESSING MODES

The DSP56300 Core provides four different addressing modes: register direct, address register indirect, PC relative and special (see Table 4-1).

### 4.4.1 Register Direct Mode

These effective addressing modes specify that the operand is in one (or more) of the 10 Data ALU registers, 24 address registers or 7 control registers.

#### 4.4.1.1 Data or Control Register Direct

The operand is in one, two or three Data ALU register(s) as specified in a portion of the data bus movement field in the instruction. This addressing mode is also used to specify a control register operand for special instructions. This reference is classified as a register reference.

#### 4.4.1.2 Address Register Direct

The operand is in one of the 24 address registers specified by an effective address in the instruction. This reference is classified as a register reference.

### 4.4.2 Address Register Indirect Modes

When an address register is used to point to a memory location, the addressing mode is called address register indirect (see Table 4-1). The term indirect is used because the register contents are not the operand itself, but rather the address of the operand. These addressing modes specify that an operand is in memory and specify the effective address of that operand.

#### 4.4.2.1 No Update (Rn)

The address of the operand is in the address register, Rn (see Table 4-1). The contents of the Rn register are unchanged by executing the instruction.

#### 4.4.2.2 Postincrement By 1 (Rn)+

The address of the operand is in the address register, Rn (see Table 4-1). After the operand address is used, it is incremented by 1 and stored in the same address register. The type of arithmetic used to calculate is determined by Mn. The Nn register is ignored.

#### 4.4.2.3 Postdecrement By 1 (Rn)-

The address of the operand is in the address register, Rn (see Table 4-1). After the operand address is used, it is decremented by 1 and stored in the same address register. The type of arithmetic used to calculate is determined by Mn. The Nn register is ignored.

#### 4.4.2.4 Postincrement By Offset Nn (Rn)+Nn

The address of the operand is in the address register, Rn (see Table 4-1). After the operand address is used, it is incremented by the contents of the Nn register and stored in the same address register. The type of arithmetic used to calculate is determined by Mn. The contents of the Nn register are unchanged.

#### 4.4.2.5 Postdecrement By Offset Nn (Rn)-Nn

The address of the operand is in the address register, Rn (see Table 4-1). After the operand address is used, it is decremented by the contents of the Nn register and stored in the same address register. The type of arithmetic used to calculate is determined by

Mn. The contents of the Nn register are unchanged.

#### 4.4.2.6 Indexed By Offset Nn (Rn+Nn)

The address of the operand is the sum of the contents of the address register, Rn, and the contents of the address offset register, Nn (see Table 4-1). The type of arithmetic used to calculate is determined by Mn. The contents of the Rn and Nn registers are unchanged.

#### 4.4.2.7 Predecrement By 1 -(Rn)

The address of the operand is the contents of the address register, Rn, decremented by 1 (see Table 4-1). The contents of Rn are decremented and stored in the same address register. The type of arithmetic used to calculate is determined by Mn. The Nn register is ignored.

#### 4.4.2.8 Short displacement (Rn+short displacement)

In this addressing mode the address of the operand is the sum of the contents of the address register Rn and a short displacement occupying 7 bits in the instruction word. The displacement is first sign extended to 24 bits and then added to Rn to obtain the address of the operand. The contents of the Rn register is unchanged. The type of arithmetic used to calculate is determined by Mn. The Nn register is ignored. This reference is classified as a memory reference.

#### 4.4.2.9 Long displacement (Rn+long displacement)

This addressing mode requires one word (label) of instruction extension. The address of the operand is the sum of the contents of the address register Rn and the extension word. The contents of the Rn register is unchanged. The type of arithmetic used to increment Rn is determined by Mn. The Nn register is ignored. This reference is classified as a memory reference.

### 4.4.3 PC Relative Modes

In the PC relative addressing modes, the address of the operand is obtained by adding a displacement, represented in two's complement format, to the value of the program counter (PC). The PC points to the address of the instruction's opcode word. The Nn and Mn registers are ignored, and the arithmetic used is always linear.

#### 4.4.3.1 Short Displacement PC Relative

The short displacement occupies 9 bits in the instruction operation word. The displacement is first sign extended to 24 bits and then added to the PC to obtain the address of the operand.

#### 4.4.3.2 Long Displacement PC Relative

This addressing mode requires one word of instruction extension. The address of the operand is the sum of the contents of the PC and the extension word.

#### 4.4.3.3 Address Register PC Relative

The address of the operand is the sum of the contents of the PC and the address register Rn. The Mn and Nn registers are ignored. The contents of the Rn register are unchanged.

### 4.4.4 Special Address Modes

The special address modes do not use an address register in specifying an effective address. These modes specify the operand or the address of the operand in a field of the instruction or they implicitly reference an operand.

#### 4.4.4.1 Immediate Data

This addressing mode requires one word of instruction extension. The immediate data is a word operand in the extension word of the instruction. This reference is classified as a program reference.

#### 4.4.4.2 Immediate Short Data

The 8-bit or 12-bit operand is in the instruction operation word. The 8-bit operand is used for immediate move to register, ANDI and ORI instructions and it is zero extended. The 12-bit operand is used for DO and REP instructions and it is zero extended. This reference is classified as a program reference.

#### 4.4.4.3 Absolute Address

This addressing mode requires one word of instruction extension. The address of the operand is in the extension word. This reference is classified as a memory reference and a program reference.

#### 4.4.4.4 Absolute Short Address

For the Absolute Short addressing mode the address of the operand occupies 6 bits in the instruction operation word and it is zero extended. This reference is classified as a memory reference.

#### 4.4.4.5 Short Jump Address

The operand occupies 12 bits in the instruction operation word. The address is zero extended to 24 bits. This reference is classified as a program reference.

#### 4.4.4.6 I/O Short Address

For the I/O short addressing mode the address of the operand occupies 6 bits in the instruction operation word and it is one extended. I/O short is used with the bit manipulation and move peripheral data instructions.

#### 4.4.4.7 Implicit Reference

Some instructions make implicit reference to the program counter (PC), system stack

(SSH, SSL), loop address register (LA), loop counter (LC) or status register (SR). The registers implied and their use is defined by the individual instruction descriptions (Appendix A).

## 4.5 ADDRESS MODIFIER TYPES

The DSP56300 Core address ALU supports linear, modulo, and reverse-carry arithmetic types for all address register indirect modes. These arithmetic types easily allow the creation of data structures in memory for FIFOs (queues), delay lines, circular buffers, stacks, and bit-reversed FFT buffers. Data is manipulated by updating address registers (pointers) rather than moving large blocks of data. The contents of the address modifier register, Mn, define the type of arithmetic to be performed for addressing mode calculations; for modulo arithmetic, the contents of Mn also specify the modulus. All address register indirect modes can be used with any address modifier. Each address register, Rn, has its own modifier register, Mn, associated with it.

### 4.5.1 Linear Modifier (Mn=$XXFFFF)

Address modification is performed using normal 24-bit linear (modulo 16,777,216) arithmetic. A 24-bit offset, Nn, and ±1 can be used in the address calculations. The range of values can be considered as signed (Nn from –8,388,608 to +8,388,607) or unsigned (Nn from 0 to +16,777,216) since there is no arithmetic difference between these two data representations.

### 4.5.2 Reverse-Carry Modifier (Mn=$000000)

Reverse carry is selected by setting the modifier register to zero. The address modification is performed in hardware by propagating the carry in the reverse direction — i.e., from the MSB to the LSB. Reverse carry is equivalent to bit reversing the contents of Rn (i.e., redefining the MSB as the LSB, the next MSB as bit 1, etc.) and the offset value, Nn, adding normally, and then bit reversing the result. If the + Nn addressing mode is used with this address modifier and Nn contains the value $2^{(k–1)}$ (a power of two), this addressing modifier is equivalent to bit reversing the k LSBs of Rn, incrementing Rn by 1, and bit reversing the k LSBs of Rn again. This address modification is useful for addressing the twiddle factors in 2k-point FFT addressing and to unscramble $2^k$-point FFT data. The range of values for Nn is 0 to + 8M (i.e., Nn=$2^{23}$), which allows bit-reverse addressing for FFTs up to 16,777,216 points.

### 4.5.3 Modulo Modifier (Mn=MODULUS–1)

The address modification is performed modulo M, where M ranges from 2 to + 32,768 (see Table 4-2). Modulo M arithmetic causes the address register value to remain within an address range of size M, defined by a lower and upper address boundary.

The value m=M–1 is stored in the modifier register, Mn. The lower boundary (base address) value must have zeros in the k LSBs, where $2^k \geq M$, and therefore must be a multiple of $2^k$. The upper boundary is the lower boundary plus the modulo size minus one (base address plus M–1). Since $M \leq 2^k$, once M is chosen, a sequential series of memory blocks (each of length $2^k$) is created where these circular buffers can be located. If $M<2^k$, there will be a space between sequential circular buffers of $(2^k)–M$.

The address pointer is not required to start at the lower address boundary or to end on the upper address boundary; it can initially point anywhere within the defined modulo address range. Neither the lower nor the upper boundary of the modulo region is stored; only the size of the modulo region is stored in Mn. The boundaries are determined by the contents of Rn. Assuming the (Rn)+ indirect addressing mode, if the address register pointer increments past the upper boundary of the buffer (base address plus M–1), it will wrap around through the base address (lower boundary). Alternatively, assuming the (Rn)- indirect addressing mode, if the address decrements past the lower boundary (base address), it will wrap around through the base address plus M–1 (upper boundary).

**Table 4-1. Addressing Modes Summary**

| Addressing Modes | Uses Mn Modifier | Operand Reference | | | | | | | | | Assembler Syntax |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | S | C | D | A | P | X | Y | L | XY | |
| Register Direct | | | | | | | | | | | |
| Data or Control Register | No | | ✓ | ✓ | | | | | | | |
| Address Register Rn | No | | | | ✓ | | | | | | |
| Address Modifier Register Mn | No | | | | ✓ | | | | | | |
| Address Offset Register Nn | No | | | | ✓ | | | | | | |
| Address Register Indirect | | | | | | | | | | | |
| No Update | No | | | | | ✓ | ✓ | ✓ | ✓ | ✓ | (Rn) |
| Postincrement by 1 | Yes | | | | | ✓ | ✓ | ✓ | ✓ | ✓ | (Rn)+ |
| Postdecrement by 1 | Yes | | | | | ✓ | ✓ | ✓ | ✓ | ✓ | (Rn)– |
| Postincrement by Offset Nn | Yes | | | | | ✓ | ✓ | ✓ | ✓ | ✓ | (Rn)+Nn |
| Postdecrement by Offset Nn | Yes | | | | | ✓ | ✓ | ✓ | ✓ | | (Rn)–Nn |
| Indexed by Offset Nn | Yes | | | | | ✓ | ✓ | ✓ | ✓ | | (Rn+Nn) |
| Predecrement by 1 | Yes | | | | | ✓ | ✓ | ✓ | ✓ | | –(Rn) |
| Short/Long Displacement | Yes | | | | | | ✓ | ✓ | ✓ | | (Rn+displ) |
| PC Relative | | | | | | | | | | | |
| Short/Long Displacement PC Relative | No | | | | | ✓ | | | | | (PC+displ) |
| Address Register | No | | | | | ✓ | | | | | (PC+Rn) |
| Special | | | | | | | | | | | |
| Short/Long Immediate Data | No | | | | | ✓ | | | | | |
| Absolute Address | No | | | | | ✓ | ✓ | ✓ | ✓ | | |
| Absolute Short Address | No | | | | | | ✓ | ✓ | ✓ | | |
| Short Jump Address | No | | | | | ✓ | | | | | |
| I/O Short Address | No | | | | | | ✓ | ✓ | | | |
| Implicit | No | ✓ | ✓ | | | ✓ | | | | | |
| Notes: | S= | System Stack Reference | | | | | | | | | |
| | C= | Program Control Unit Register Reference | | | | | | | | | |
| | D= | Data ALU Register Reference | | | | | | | | | |
| | A= | Address ALU Register Reference | | | | | | | | | |
| | P= | Program Memory Reference | | | | | | | | | |
| | X= | X Memory Reference | | | | | | | | | |
| | Y= | Y Memory Reference | | | | | | | | | |
| | L= | L Memory Reference | | | | | | | | | |
| | XY= | XY Memory Reference | | | | | | | | | |

If an offset, Nn, is used in the address calculations, the 24-bit absolute value, |Nn|, must be less than or equal to M for proper modulo addressing. If Nn>M, the result is data dependent and unpredictable, except for the special case where Nn=P x $2^k$, a multiple of the block size where P is a positive integer. For this special case, when using the (Rn)+Nn addressing mode, the pointer, Rn, will jump linearly to the same relative address in a new buffer, which is P blocks forward in memory. Similarly, for (Rn)–Nn, the pointer will jump P blocks backward in memory.

This technique is useful in sequentially processing multiple tables or N-dimensional arrays. The range of values for Nn is –8,388,608 to +8,388,607. The modulo arithmetic unit will automatically wrap around the address pointer by the required amount. This type address modification is useful for creating circular buffers for FIFOs (queues), delay lines, and sample buffers up to 8,388,607 words long as well as for decimation, interpolation, and waveform generation. The special case of (Rn) $\pm$ Nn modulo M with Nn=P x $2^k$ is useful for performing the same algorithm on multiple blocks of data in memory — e.g., parallel infinite impulse response (IIR) filtering.

### 4.5.4 Multiple Wrap-Around Modulo Modifier

The multiple wrap-around addressing mode is selected by setting bit 15 of modifier register to one (see Table 4-2). The address modification is performed modulo M, where M may be any power of 2 in the range from $2^1$ to $2^{14}$. Modulo M arithmetic causes the address register value to remain within an address range of size M defined by a lower and upper address boundary. The value M-1 is stored in the modifier register Mn least significant 15 bits while the 16th bit (bit 15) is set to one and the rest of the most significant 8 bits are considered don’t care. The lower boundary (base address) value must have zeroes in the k LSBs, where $2^k$ = M, and therefore must be a multiple of $2^k$. The upper boundary is the lower boundary plus the modulo size minus one (base address plus M-1).

The address pointer is not required to start at the lower address boundary and may begin anywhere within the defined modulo address range (between the lower and upper boundaries). If the address register pointer increments past the upper boundary of the buffer (base address plus M-1) it will wrap around to the base address. If the address decrements past the lower boundary (base address) it will wrap around to the base address plus M-1. If an offset Nn is used in the address calculations, it is not required to be less than or equal to M for proper modulo addressing since multiple wrap around is supported for (Rn)+Nn, (Rn)-Nn and (Rn+Nn) address updates (multiple wrap-around cannot occur with (Rn)+, (Rn)- and -(Rn) addressing modes).

### 4.5.5 Address-Modifier-Type Encoding Summary

Table 4-2 is a summary of the address modifier types discussed in the previous paragraphs. There are four modifier types:

- Linear Addressing
- Reverse-Carry Addressing
- Modulo Addressing
- Multiple Wrap-Around Modulo Addressing

Bit-reverse addressing is useful for $2^k$-point FFT addressing. Modulo addressing is useful for creating circular buffers for FIFOs (queues), delay lines and sample buffers. The linear addressing is useful for general-purpose addressing. The multiple wrap-around address modifier is useful for decimation, interpolation and waveform generation since the multiple wrap-around capability may be used for argument reduction.

**Table 4-2. Address-Modifier-Type Encoding Summary**

| Modifier Mn | Address Calculation Arithmetic |
|---|---|
| XX0000 | Reverse-Carry (Bit-Reverse) |
| XX0001 | Modulo 2 |
| XX0002 | Modulo 3 |
| : | : |
| XX7FFE | Modulo 32767 ($2^{15}$-1) |
| XX7FFF | Modulo 32768 ($2^{15}$) |
| XX8001 | Multiple Wrap-Around Modulo 2 |
| XX8003 | Multiple Wrap-Around Modulo 4 |
| XX8007 | Multiple Wrap-Around Modulo 8 |
| : | : |
| XX9FFF | Multiple Wrap-Around Modulo $2^{13}$ |
| XXBFFF | Multiple Wrap-Around Modulo $2^{14}$ |
| XXFFFF | Linear (Modulo $2^{24}$) |

Notes: XX means don’t care

All other combinations are reserved