DSP Lab
Lecture Notes for Lab #1

Spring 2012

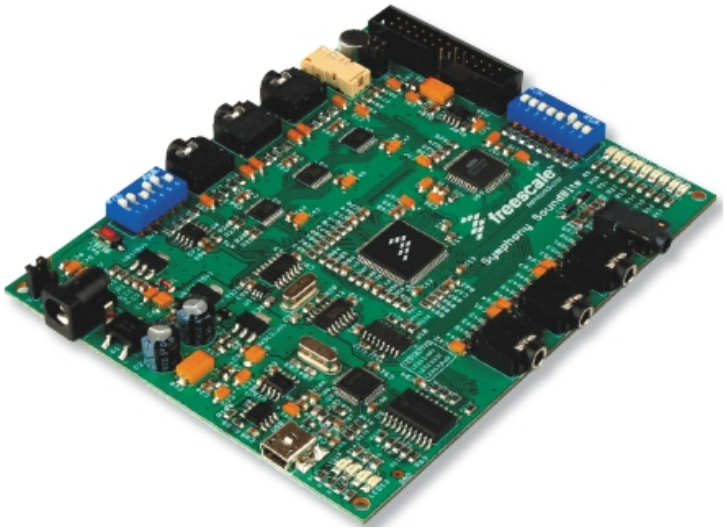
Some materials referenced from class materials by:
Dr. Bruce Wiggins of the University of Derby
Prof. Clifford T. Mullis of the University of Colorado

Other References:
The Scientist and Engineer's Guide to Digital Signal Processing, By Steven W. Smith, Ph.D
Freescale Manuals: DSP56000 Family Manual, Assembler Reference Manual

Introduction to DSP Lab

What is a DSP?

- A specialized CPU used for specific digital signal processing tasks.
- In many applications it is a coprocessor that aids the main microprocessor in the performance of a specific function.
- DSPs are very useful for real time signal processing operations (such as used for audio and image processing) since they can be optimized for real time operation and fast reliable performance.

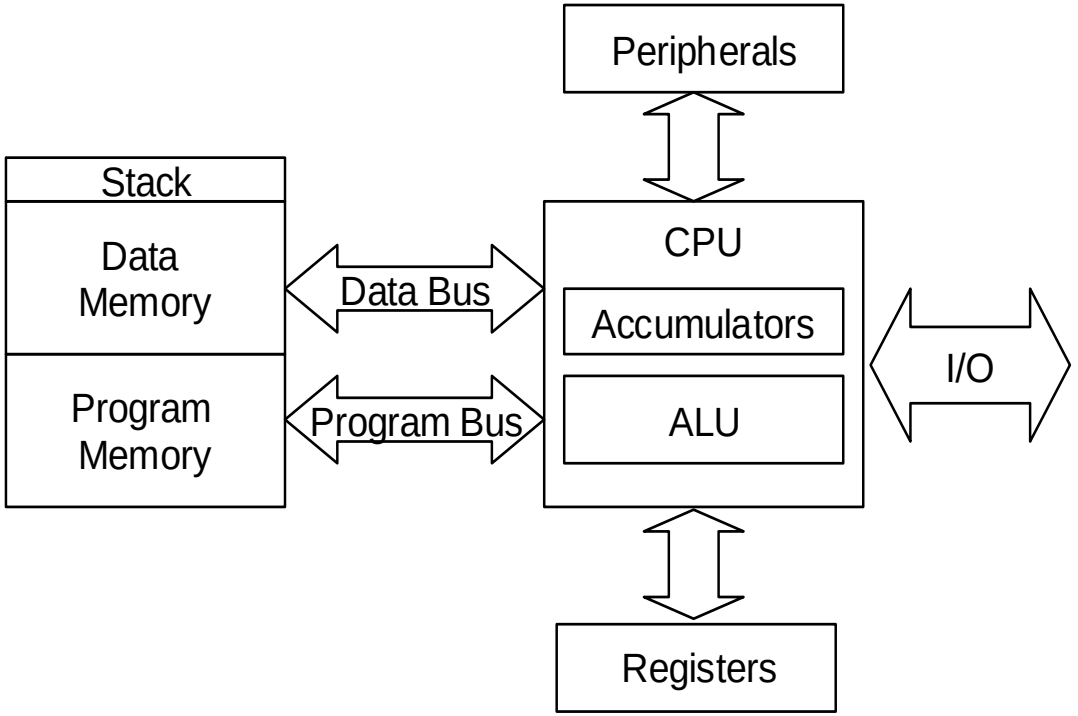


DSP vs Microprocessors

- Microprocessors are typically built for a range of general purpose functions while DSPs handle specific tasks
- Microprocessors aren't often used for real time operations while this is the main function of DSPs
- DSPs have much stronger numeric capabilities
- All DSPs share three main characteristics:
- Ability to perform high speed arithmetic operations
- Ability to transfer data to and from the real world
- Multiple access memory architectures for fast processing capabilities

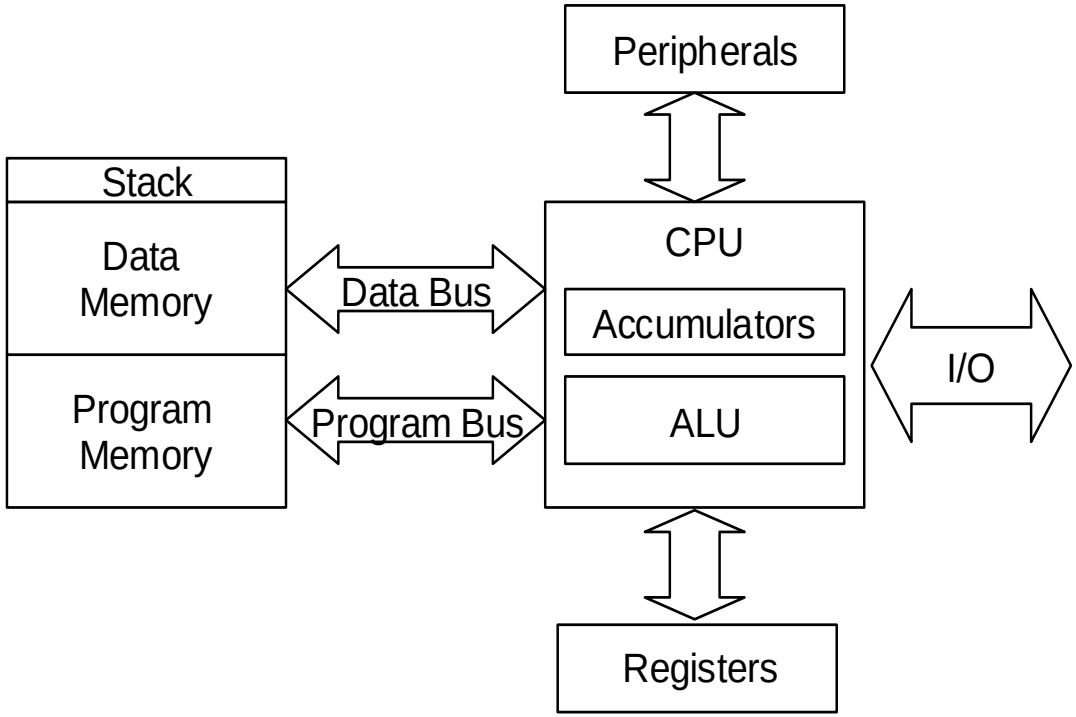
Main DSP Components

Arithmetic Logic Unit (ALU): Performs most of the arithmetic operations required by the DSP. Two's complement is usually used but the ALU may work using fixed point or floating point math. For some DSP's, additional units are implemented that take over some common arithmetic tasks such as multiply-and-add operations or compares.



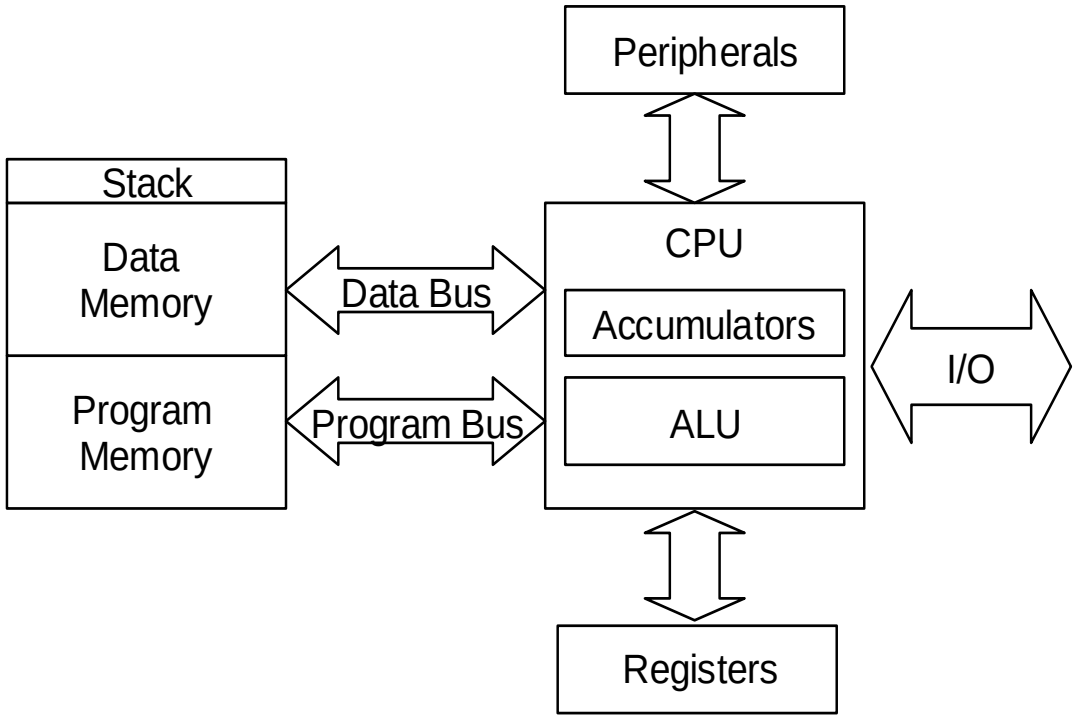
Main DSP Components

- Accumulators: Data registers used as the input and output to the ALU for most arithmetic functions. The accumulator bits are grouped as follows:
  - Guard bits
  - High-order word
  - Low-order word



Main DSP Components

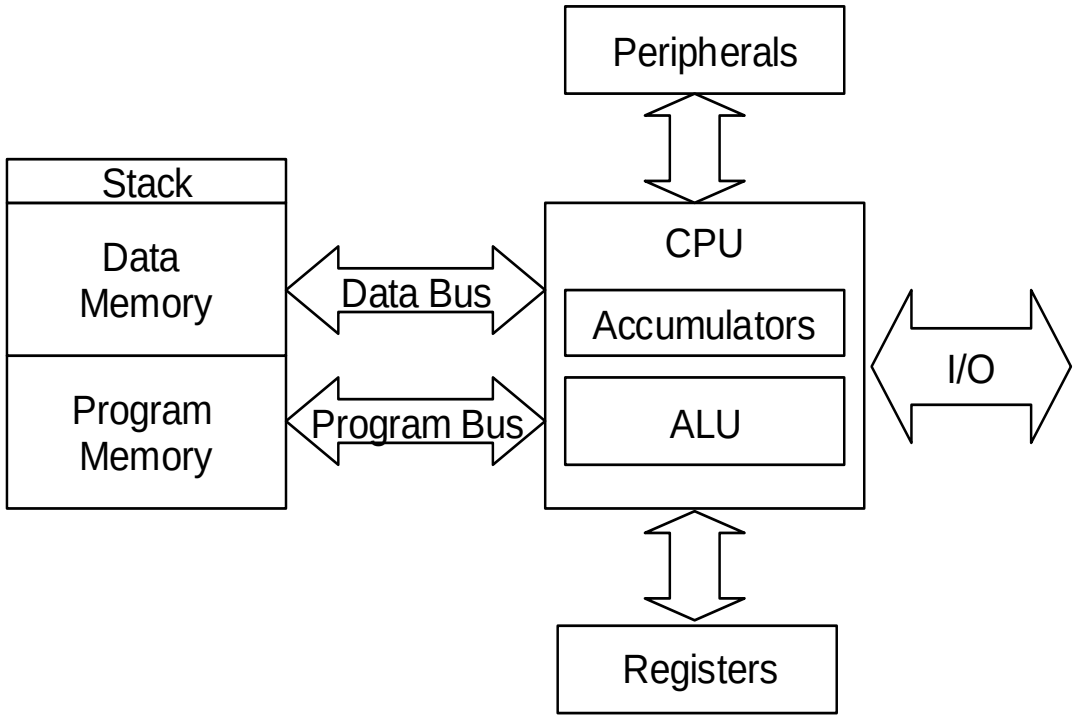
- Registers: There are two type of registers for the DSP:
  - Program registers: Locations that store system information, DSP modes and status bits.
  - Data or Auxiliary Registers: Also referred to as memory mapped registers, they are usually used to generate addresses that point to a location in data memory (pointers). They can also be used to store temporary data values.



Main DSP Components

- Stack: Section in data memory used to store temporary data of two types:
  - Local variables for functions and interrupt routines
  - Reference pointers to the program address of the return function

- Stack Pointer: Register that contains the address of the top of the system stack. This address always points to the last item pushed onto the stack.



DSP I/O

- DSPs are mostly useful due to their ability to interface with the real world in a fast real-time manner.
- This is typically achieved through high speed synchronous serial ports that are well suited for receiving audio, telecommunications and similar signals.
- Serial ports are inexpensive and easy to interface to peripherals such as basic analog to digital converters.
- Most serial ports operate using direct memory access (DMA) and therefore do not generate any overhead to the DSP.
- By interfacing through serial ports vs dedicated I/O ports, the DSP allows data to be received from any number of peripherals and hardware with very little customization required.

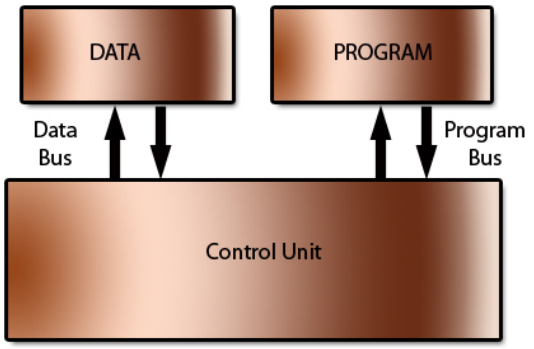
DSP Architectures

There are two main types of DSP processor architectures that are meant to provide the DSP with the ability to access multiple memory locations simultaneously (within one instruction cycle).

- Harvard Architecture: Most common is the Harvard architecture which provides two separate memory busses allowing two simultaneous memory locations to be accessed within a single cycle. Some DSPs limit one of these busses to fetching instructions while the other is used for fetching operand. This can be problematic since many instructions require more than one operand and therefore require two cycles to be executed. A variation of the Harvard architecture are SHARC processors which provide an instruction cache to store instructions prior to their use allowing therefore both memory busses to be used for operands.

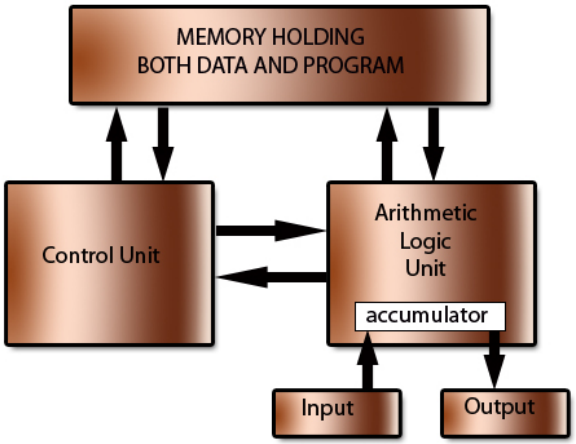
- The modified Von Neuman architecture provides a single memory buss but runs the memory access clock at a faster speed than the instruction clock therefore allowing multiple memory accesses per instruction cycle. The advantage of this architecture vs Harvard is that it allows smaller chips (less physical busses).

The Harvard architecture



(c) www.teach-ict.com

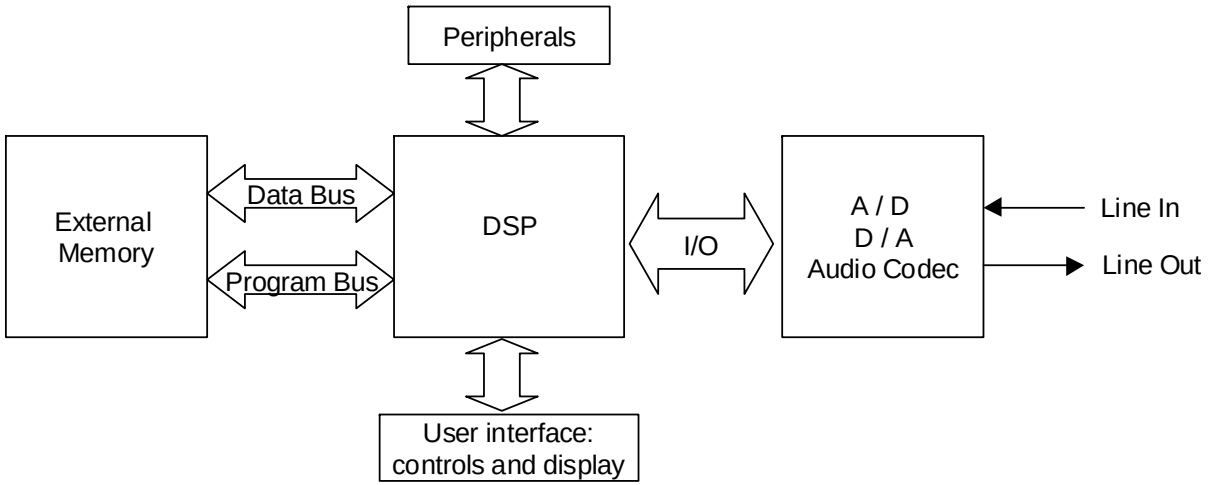
The Von Neumann or Stored Program architecture



(c) www.teach-ict.com

Development Kits

We will be working with the Symphony Soundbite development kit from Freescale. Development kits of EVMs (evaluation modules) are boards built for the specific purpose of code and applications development for a DSP. They are usually integrated by DSP manufacturers to provide to their customers in order to demonstrate DSP functionality and provide a development platform. The Symphony Soundbite board is a low-cost board designed specifically for college laboratories.



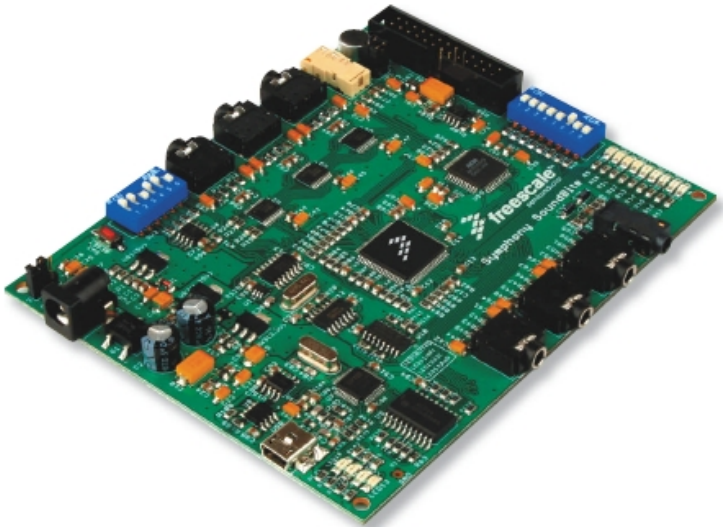
Symphony Soundbite Board

The Symphony Soundbite is an audio processing board that is capable of simultaneously processing 8 channels of audio in and out via 4 stereo channels. Some of its main features are:

- Can be powered by USB bus voltage or external power adaptor
- On-board USB interface that provides JTAG debug, I2C and SPI serial communication with the DSP
- 1 of the 4 input and output channels share digital and analog audio capabilities through a SPDIF transceiver.
- On-board microphone and pre-amplifier
- GPIO interface
- 9 LEDs can be used as universal indicators

Symphony Soundbite Board

- The Symphony Soundbite board is based on a Symphony DSPB56371 DSP
- 24 bit fixed-point processor, with 180 MIPS at 180 MHz
- On-chip memories:
  - 4-44k x 24-bit words of PRAM
  - 28-36k x 24-bit words of XRAM
  - 16-48k x 24-bit words of YRAM.



Software

- Software development for the Symphony Soundbite board is recommended using the Symphony Studio IDE which is based on the Eclipse platform. The Symphony Studio QuickStart provides the basic reference for starting a project and getting connected to the board.

- Though the IDE platform allows DSP programming using C/C++, this lab requires that code development for most labs be developed using assembly language; though features beyond the requirements of each exercise can be developed on C if so desired.

Assembler Project Template

Summary of Files

- **Main.asm**: startup code and main test code
- **Process_samples.asm**: audio processing routine; includes a processing enable/disable flag.
- **Sb_codecs.asm**: enables communication between DSP and audio codec
- **Sb_isr_esais.asm**: Interrrupt service routines, actual audio processing between codec and memory.
- **Sb_leds.asm**: Enable LED r/w, reads and sets LED states
- **Sb_switches**: Enable switch GPIO and reads states
- **Sb_eeprogram.asm**: EEPROM programming routines to allow Sounbite to self boot.
- **Soundbite_macros.equ**: Contains macros for reading and writing to/from switches and LEDs
- **soundbite.equ**: Contains a number of useful EQU statements

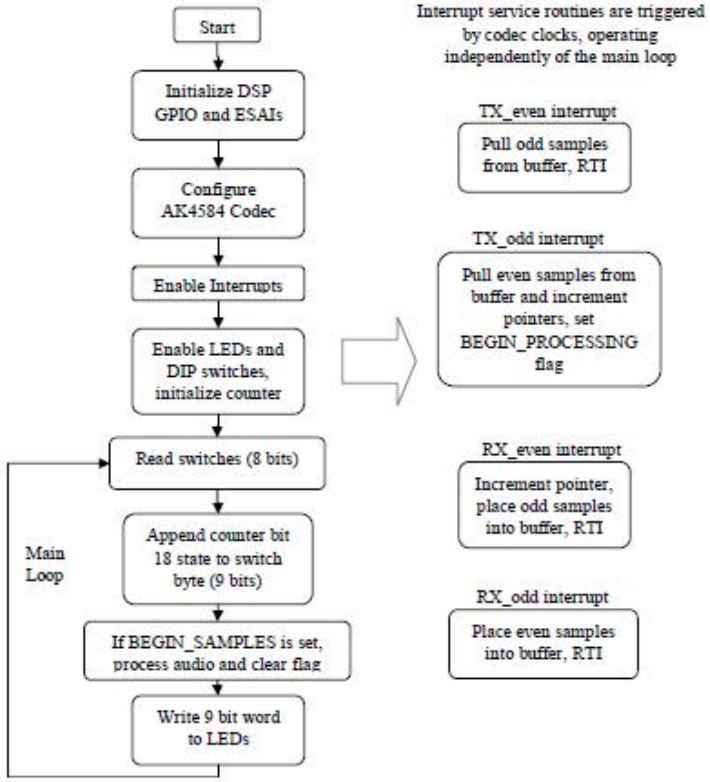


Figure 3-1. Assembly Project Template Flowchart

DSP Memory and Standard Registers

- There are 3 memory spaces in the DSP
  - Program
  - X Data Memory
  - Y Data Memory
- When creating a variable, one must review the memory map to make sure it is stored in an adequate location
- The X and Y memory data spaces hold channel data in circular buffers for each codec.
- X memory holds the software stack intended for subroutines to use to save and restore registers or any other values as desired.
- X memory also holds setup registers required for AK4584 initialization
- Pay particular attention to the circular buffers used to store the audio data.

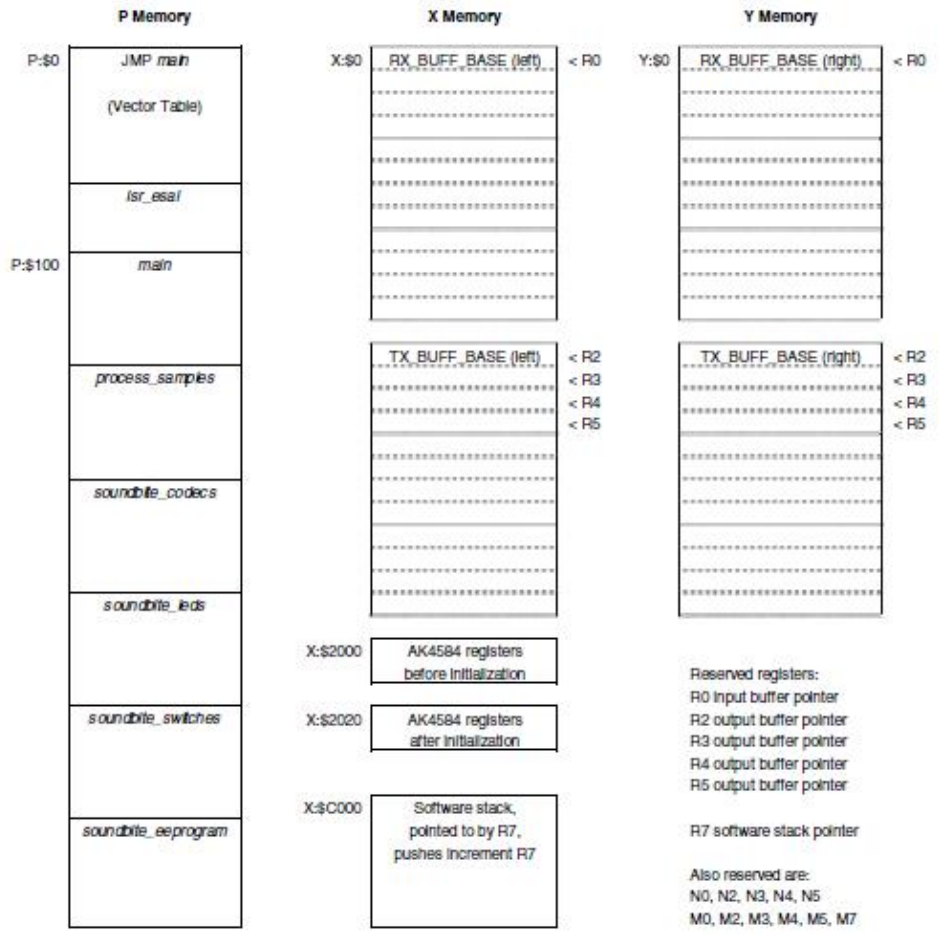
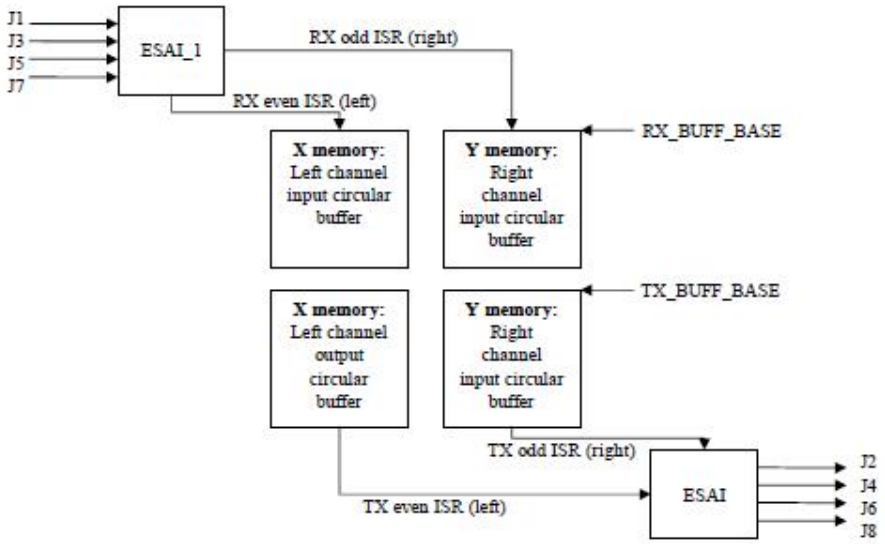


Figure 4-1. Assembly Project Template application memory map in the P, X and Y memory spaces.

Input and Output Buffer Structure

- Four total interrupts: 2 for receiving input data and 2 for transmitting outout data.
- Right channel ISR also sets BEGIN_PROCESSING flag
- Call to “process_samples” happens when BEGIN_PROCESSING flag is set
- Input circular buffers contain 3 blocks of 4 samples each so include previous data which makes it easy to directly implement 2nd order filters



Input and Output Buffer Structure

- Register R0 is a pointer to the input circular buffer
  - R0 points to first input channel
  - R0+1 to 2nd input channel and so on…
- Registers R2 through R5 point to each output channel independently

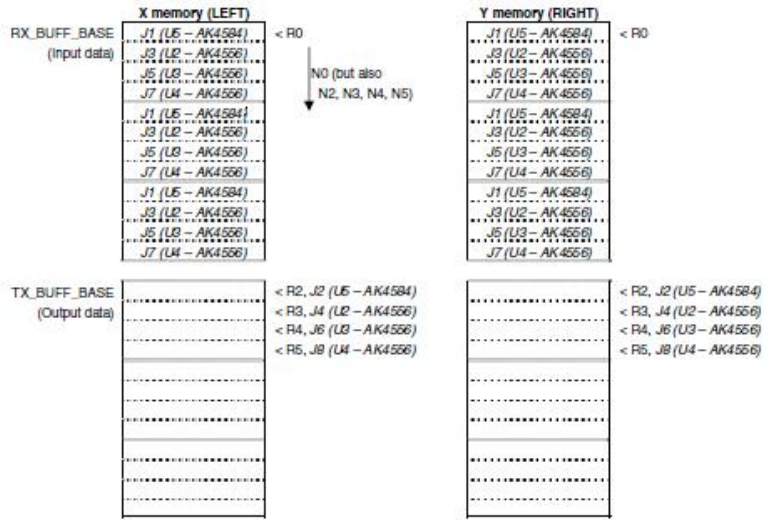


Figure 4-3: Diagram of the default left and right input and output buffer structure with the data sources and destinations.

main.asm | soundbite.equ | process_samples.asm | sb_codecs.asm | sb_isr_esais.asm

```
55;*************************************************************************
56; reserve 'buffsize*keep' words for storage of input sound data
57; even data in X (left) and odd in Y (right)
58;
59; 'keep' must be at least 1 and may be changed to keep more or less previous
60; that the 3 set up below (current plus previous 2 samples)
61;*************************************************************************
62
63buffsize    equ     4   ; 4 I2S (stereo) channels (X = left, Y = right)
64keep        equ     3   ; 3 = current sample plus past 2 samples for each channel
65
66    org x:0     ; reserve space for buffer in X
67RX_BUFF_BASE    dsm keep*buffsize   ; RX buffer: make large enough for 'keep' samples
68TX_BUFF_BASE    dsm keep*buffsize   ; TX buffer: make large enough for 'keep' samples
69
70    org Y:0     ; reserve space for shadow buffer in Y, a copy of X, (these symbols are not used)
71RX_BUFF_BASEy   dsm keep*buffsize   ; RX buffer: make large enough for 'keep' samples
72TX_BUFF_BASEy   dsm keep*buffsize   ; TX buffer: make large enough for 'keep' samples
73
```

Reserved Registers (For Assembly Template)

- Notice that the number of previous samples to keep can be modified in the assembly project template ("keep" variable).
- This can be useful for higher order filters with the correct use of the offset registers

Table 5-1: Reserved Registers and their usage.

Register	Description of Usage
R0	Input buffer pointer used for all ESAI_1 input channels
R2	Output pointer used for ESAI SDO0 [AK4584 codec (U5), J2]
R3	Output pointer used for ESAI SDO1 [AK4556 codec (U2), J4]
R4	Output pointer used for ESAI SDO2 [AK4556 codec (U3), J6]
R5	Output pointer used for ESAI SDO3 [AK4556 codec (U4), J8]
R7	Software stack pointer used by interrupt service routines
N0, N2, N3, N4, N5	Offset registers used with the input and output pointers used for incrementing the Rx registers by *buffsize* (see *sb_isr_esais.asm*)
M0, M2, M3, M4, M5	Modifier registers set with the value *keep* (see *sb_isr_esais.asm*) to make the Rx registers behave as circular buffer pointers
M7	Modifier register for R7, keeps default linear arithmetic value of $FFFFFF

Simple Gain Project (Courtesy of Dr. Wiggins)

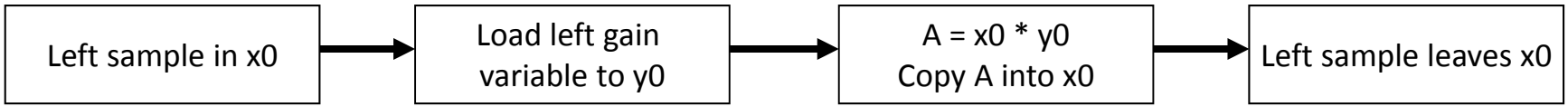
- First step is to create run-time modifiable variables and assign them to memory
- Variables are assigned in process_samples prior to defining the program (line 44: *org p:*)
- Simple variables for left and righ channel gain

```
                org     x:$20
LGAIN           dc      0.8 ;
RGAIN           dc      0.1 ; placed BEFORE org p:
```

Quick Look at the audio processing function

```
90
91 PROCESS_AUDIO:   ; Replace the code in this routine with your own custom audio processing
92                  ; routine.  This is just a copy of the pass-through code above...
93                  ; remember to preserve registers not in the reserved register list
94                  ; so-as not to stomp on any main loop usage of them.
95
96                                  ; copy input to output directly (duplicate of above)
97     move    x0,x:(r7)+           ; with no processing...
98     move    x1,x:(r7)+           ; preserve x0,x1,y0,y1 before processing
99     move    y0,x:(r7)+
100    move    y1,x:(r7)+
101
102    move    x:(r0),x0            ; left data copy
103    move    x:(r0+1),x1
104    move    x:(r0+2),y0          ; the offsets below copy the current sample to the
105    move    x:(r0+3),y1          ; same position in the ouptut buffer for each channel
106    move    x0,x:(r0+TX_BUFF_BASE-RX_BUFF_BASE)
107    move    x1,x:(r0+TX_BUFF_BASE-RX_BUFF_BASE+1)
108    move    y0,x:(r0+TX_BUFF_BASE-RX_BUFF_BASE+2)
109    move    y1,x:(r0+TX_BUFF_BASE-RX_BUFF_BASE+3)
110
111    move    y:(r0),x0            ; right data copy
112    move    y:(r0+1),x1
113    move    y:(r0+2),y0          ; the offsets below copy the current sample to the
114    move    y:(r0+3),y1          ; same position in the ouptut buffer for each channel
115    move    x0,y:(r0+TX_BUFF_BASE-RX_BUFF_BASE)
116    move    x1,y:(r0+TX_BUFF_BASE-RX_BUFF_BASE+1)
117    move    y0,y:(r0+TX_BUFF_BASE-RX_BUFF_BASE+2)
118    move    y1,y:(r0+TX_BUFF_BASE-RX_BUFF_BASE+3)
119
120    move    x:-(r7),y1           ; restore x0,x1,y0,y1 after processing
121    move    x:-(r7),y0
122    move    x:-(r7),x1
123    move    x:-(r7),x0
124
125    rts
```

Simple Gain Project – Basic Process



Notice that arithmetic operation's result is stored in accumulator A and therefore needs to be copied back out to x0 so it can be passed through to outputs.

```
;Process Left Samples Here.........
move    x:LGAIN,y0      ;move LGain into y0
mpy     x0,y0,A         ;Left Sample  A = x0 * y0
move    A,x0            ;Put Sample back!
;--------------------------------


;Process Right Samples Here.........
move    x:RGAIN,y0      ;move LGain into y0
mpy     x0,y0,A         ;Left Sample  A = x0 * y0
move    A,x0            ;Put Sample back!
;--------------------------------
```

Remember to preserve all registers prior to use, including the accumulator register a0 and b0!

DSP56371 Functional Blocks

- Data arithmetic logic unit (Data ALU)
- Address generation unit (AGU)
- Program control unit (PCU)

Core Execution Units

- DMA controller (with six channels)
- Instruction patch controller
- PLL-based clock oscillator
- OnCE module (on chip emulator)
- Memory

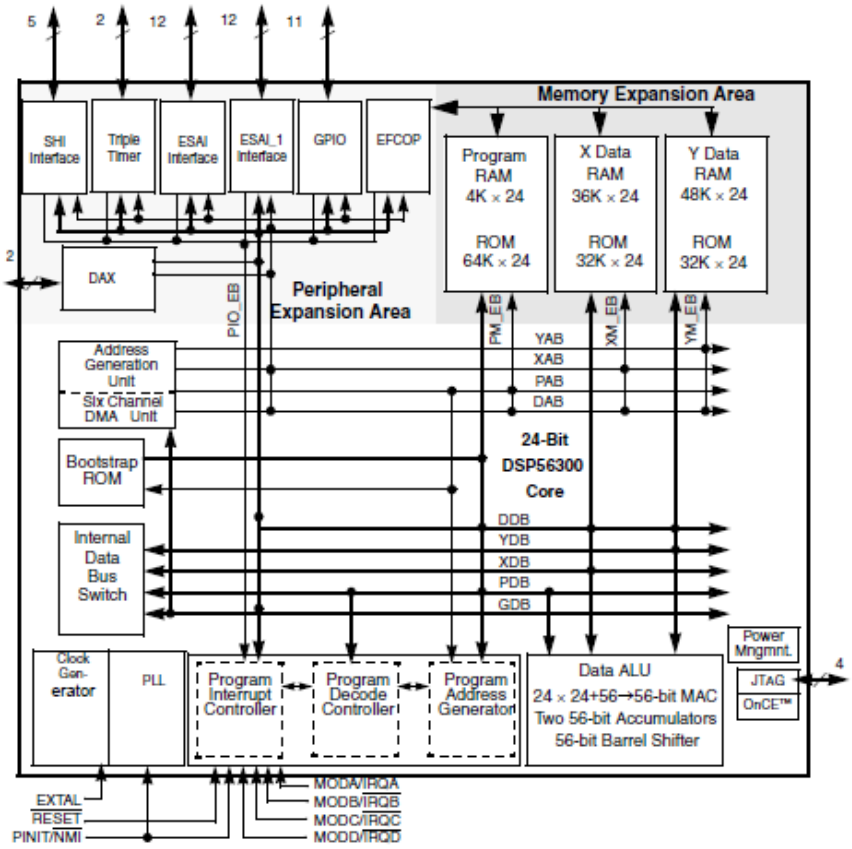


Figure 1-1 DSP56371 Block Diagram

Data ALU

Arithmetic Logic Unit: Performs all arithmetic and logical operations

Registers:

- Data Registers:
  - 4 24-bit input registers x0,x1,y0,y1 that hold the data for processing by the ALU (do not confuse with Xmem or Ymem)
  - Accessed using the XDB or YDB as 24 or 48 bit operands
- Accumulator Registers:
  - 2 48-bit accumulator registers plus an 8 bit accumulator extension for each (total 56 bits)
  - All arithmetic operation results are stored in the A or B accumulators

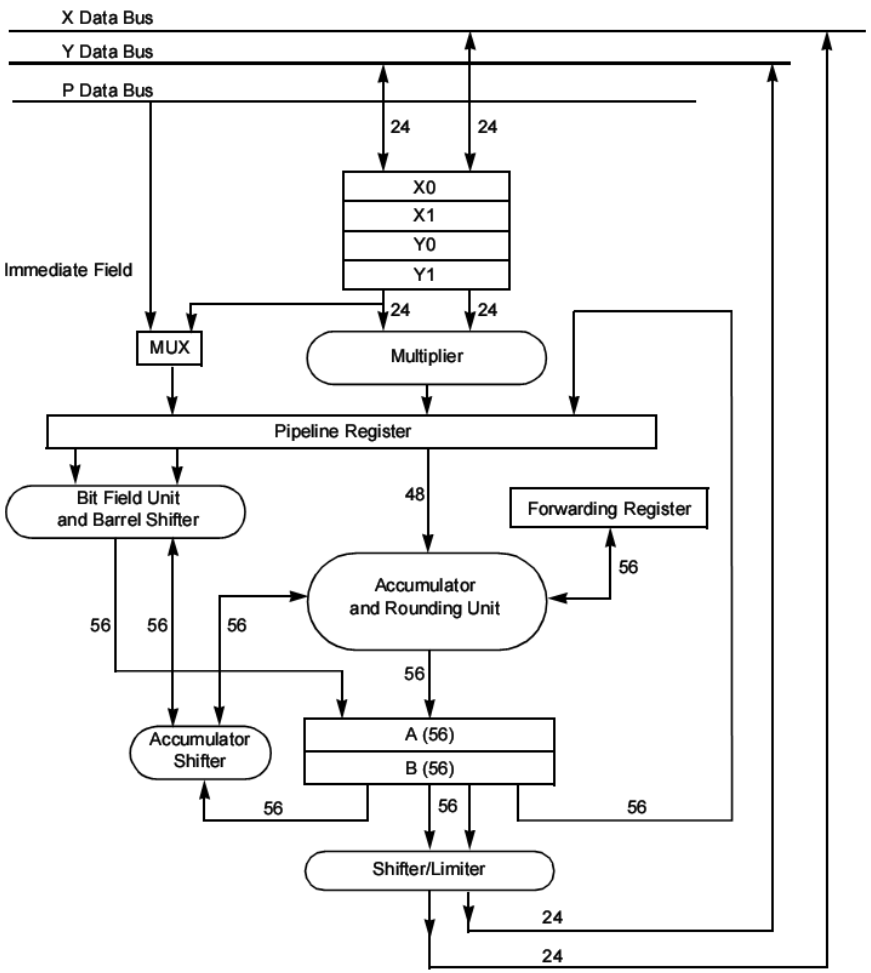
Side Note: Bit depth and dynamic range

- Note: Dynamic range is typically estimated at 6dB per bit, so 16 bit signal provides about 96dB, a 24-bit up to 144db and a 56 bit, up to 336dB.
- This is all theoretical, there are real-world limitations that reduce and limit the range. For example, a 24-bit A/D would have to recognize signals down to a billionth of a volt to maintain true 144dB of range.
- There are however real differences due to headroom and resolution.
- For DSP purposes, it greatly affects the resolution of our calculations which could significantly enhance or degrade the processing chain.

Back to Data ALU

- The DSPs ALU can perform any of its core instructions in one instruction cycle. However, the DSP56371 has a fully pipelined accumulator which it can perform operations in a single clock cycle (2 clock cycles per instruction cycle)

- Data can be loaded from the XDB and YDB while the ALU units are performing an operation



ALU Main Units

MAC Unit

- The multiplier executes 24-bit × 24-bit, parallel, fractional multiplies, between two's-complement signed,unsigned, or mixed operands.
- The 48-bit product is right-justified and added to the 56-bit contents of either the A or B accumulator.
- A 56-bit result can be stored as a 24-bit operand. The LSP can either be truncated or rounded.
- A basic multiply operation is performed by clearing out the accumulator first

• Bit field Unit

- Contains a 56-bit parallel bi-directional shifter with 56-bit input and 56-bit output.
- Multi-bit left or right shift (arithmetic or logical) for ASL, LSL, ASR and LSR.
- 1-bit rotate left or right for ROR and ROL
- Bit field merge, insert and extract for MERGE, INSERT, EXTRACT and EXTRACTU
- Count Leading Bits for CLB
- Logical operations for AND, OR EOR and NOT.

Accumulator and Arithmetic

- Accumulators are made up of 3 concatenated registers (8/24/24) for a total of 56 bits
- The 8-bit sign extension is stored in A2 or B2, and can be used when more than 48-bit accuracy is needed.
- The 24-bit most significant product (MSP) is stored in A1 or B1
- The 24-bit least significant product (LSP) is stored in A0 or B0.
- It is a fractional accumulator so values are limited from almost 1 to -1. Why not 1?
- Care must be taken when copying data for integer operations using the extension

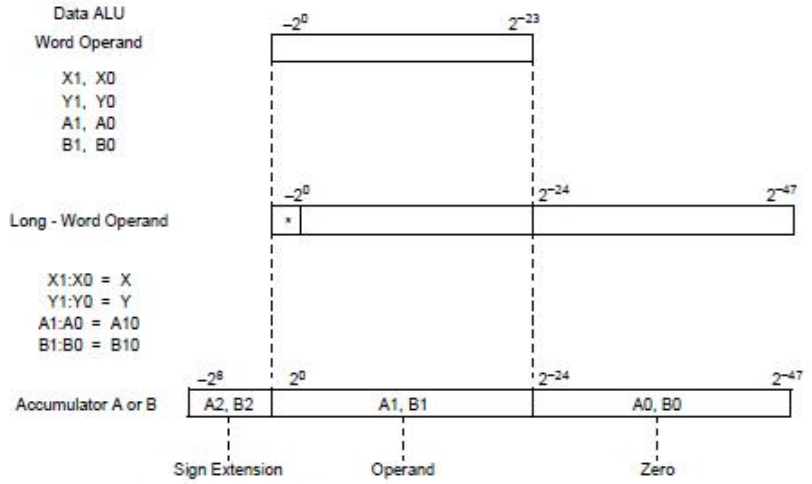


Figure 3-2. Bit Weighting and Alignment of Operands

- If a 48-bit number is written to A or B, and EXT portion will be sign extended.
- If a 24-bit number is written to A or B, the LSP (A0 or B0) will be zero filled.
- If a 24-bit number is written DIRECTLY to A1,A0,B1 or B0) no sign extensions take place.

Limiting, Scaling and Rounding

- Limiting is the process by which the DSP limits the value to its fractional range prior to copying a 56-bit number into memory
  - Occurs automatically when the full accumulator is copied:

    Move A,X (56 bit accumulator to 48 bit memory)
  - Does not occur if copying the MSP or LSP parts of the accumulator directly (careful!)
- Scaling is performed every arithmetic operation by shifting by one bit in either direction (or leaving bits unchanged)
- Rounding can also be performed during an arithmetic operation (MAC will round LSP to 0 and either add 1 or not to the MSP)

Address Generation Unit (AGU)

- 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.
- Four types of arithmetic:
  - Linear
  - Modulo
  - Multiple wrap-around modulo
  - Reverse-carry
- The AGU operates in parallel with other chip resources to minimize address-generation overhead.

Address Generation Unit (AGU)

- The AGU is divided into identical halves, each with its own address arithmetic logic unit
- Each address ALU has four sets of register triplets:
  - Address register (R0-R3,R4-R7), an offset register (N0-N3,N4-N7), and a modifier register (M0-M3,M4-M7). Notice that the triplets are tied so only N2 and M2 can modify R2 for example.
- Each contains a 24-bit full adder—an offset adder—which can perform additions/subtractions on an address register (Plus one, minus one, Plus the contents of the respective offset register N, Minus the contents of the respective offset register N)
- A modulo 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 reverse-carry adder can perform offset additions, with the carry propagating in the reverse direction (that is, from the Most Significant Bit (MSB) to the Least Significant Bit (LSB)
- The two ALUs can generate two 16-bit addresses every instruction cycle.

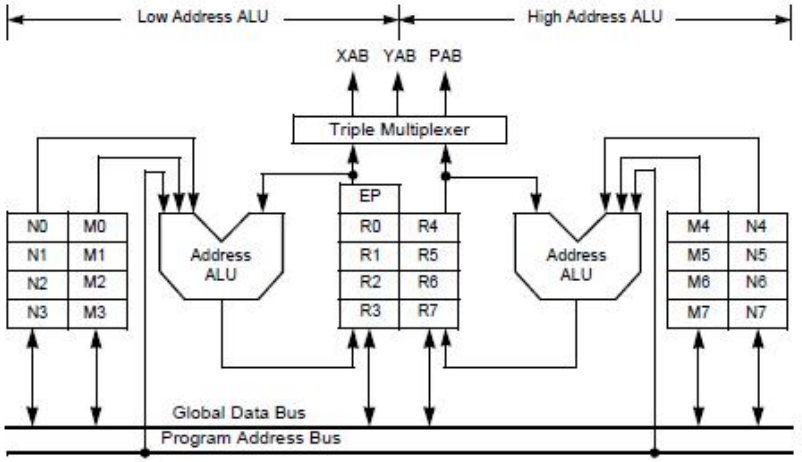
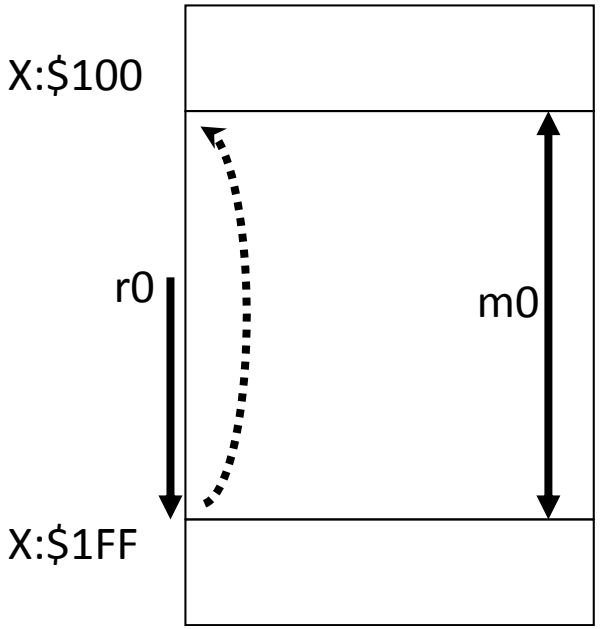


Figure 4-1. AGU Block Diagram

Modulo Arithmetic for Circular Buffers

- Circular buffers mean that if incrementing or decrementing the pointer past the beginning or end of the buffer, it automatically wraps round to the other end.
  - R – Current Address
  - M – Modulo length – 1
  - N – Table update rate (doesn't have to used)
- Circular buffers must be aligned to the next power of 2 (so for a 20 sample buffer, a 32 memory buffer must be allocated)



AGU Arithmetic

– No update (!)	(r0)
– Postincrement by 1	(r0)+
– Postdecrement by 1	(r0)-
– Postincrement by Offset N	(r0)+n0
– Postdecrement by Offset N	(r0)-n0
– Indexed by Offset N	(r0+n0)
– Predecrement by 1	-(r0)
– Short Displacement	(r0+63)
– Long Displacement	(r0+64)

- No other operation is supported. For example, don't try +(ro)…won't work!

- DSP Numbers!

DSP Numbering

- Consider an example where a computer has 8 bits available, in the simplest case the range of numbers is:

  0 to 2^8 …. So 0 to 256.

- To make it useful for specific applications, a more useful representation might be -127 to 128 or in a more advanced case say a representation of exponential numbers such as 1,10,100,1000….10^254, 10^255.

- What is important is that everyone accessing the data understand what the values mean. In our case, it is the DSP algorithms that must understand the value a certain number represents.

- There are two encoding schemes in use by digital processors, fixed and floating point.

Basically floating point allows the direct representation of decimals, while in fixed point decimals are represented by integers and the limits of the precision are based on the fixed value of decimals. So that if 1 bit is available for decimals the precision will be 0.5 (0 or 1), two bits available for decimals the precision will be .25, and so on.

Fixed Point Arithmetic

- Unsigned integer:
  - Bit patterns are assigned to numbers 0 and up. So for 16 bits (2^n) one can represent the range 0 to 65535
  - Your basic binary number conversions….
- Offset binary:
  - A first attempt to represent negative values
  - 0 crossing point is offset so that a range of $-2^{n-1}-1$ to 2^{n-1} is possible. 32767 to -32768 for 16 bit for example
  - A typical example of offset binary is the quantization process in ADC and DAC converters so a specific voltage range (i.e. -5 to +5) is mapped to a range of values.
- Signed integer:
  - Negative numbers are represented by the MSB
  - Numbers “grow” in both directions depending on the MSB sign bit.
  - In this scheme “0” has two identical representations. For example, 0000 and 1000 would both indicate a zero.
- And…..Two’s complement

Fixed-Point: Two's Complement

- Standard math for signed fixed-point numbers is computationally expensive to implement.
- Two's complement is a circular computing method which is easy for processors to perform
- The number wraps around as arithmetic operations are executed.
- Sign bit is used in the MSB where a 1 indicates a negative number and a 0 a positive number.
- For positive numbers, conversion from binary to decimal is direct (0111 = 7) yet for negative numbers requires a conversion formula where:
  - Convert the absolute value of the negative number: -5 = 0101
  - Complement all the bits directly so 0s become 1s and 1s become 0s: 1010
  - Add 1 to the LSB: 1011

Fixed-point numbering comparison

UNSIGNED INTEGER	
Decimal	Bit Pattern
15	1111
14	1110
13	1101
12	1100
11	1011
10	1010
9	1001
8	1000
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000

16 bit range:
0 to 65,535

OFFSET BINARY	
Decimal	Bit Pattern
8	1111
7	1110
6	1101
5	1100
4	1011
3	1010
2	1001
1	1000
0	0111
-1	0110
-2	0101
-3	0100
-4	0011
-5	0010
-6	0001
-7	0000

16 bit range
-32,767 to 32,768

SIGN AND MAGNITUDE	
Decimal	Bit Pattern
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

16 bit range
-32,767 to 32,767

TWO'S COMPLEMENT	
Decimal	Bit Pattern
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

16 bit range
-32,768 to 32,767

Finite Precision Concepts

- Precision:
  - Number of non-zero bits. Ex. Signed A(13,2) has a precision of 16 bits
- Resolution:
  - Smallest non-zero magnitude that can be represented.
  - Ex. Signed A(10,5) has a resolution of $1/2^5 = 0.03125$
- Range:
  - Difference between maximum positive and negative values
  - Ex. Signed A(13,2) has a range of 8191.75 to -8192 = 16,383.75
- Accuracy:
  - Maximum difference between represented and real value
  - Accuracy = Resolution/2. Ex. Signed A(13,2) has accuracy of 1/8
- Wordlength:
  - Minimum number of bits required to represent a number.
  - Unsigned U(a,b) = a + b
  - Signed U(a,b) = a + b + 1

Which is why fractional arithmetic is typically used (Q.15)

Floating Point Basics

- Similar to scientific calculations except it uses base 2 instead of base 10.
- Good at displaying a range of very large to very small numbers using few digits and always a single nonzero digit to the left of the decimal point. For example:
  - The number of atoms on the earth 1.2x10^50
  - or the distance a turtle crawls in a second 2.6x10^-23.

- There are several floating point standards but the main in use is ANSI/IEEE Std. 754-1985 which defines a 32 bit representation for single precision and 64 bit for double precision.

- The available bits are divided into three separate groups:
  - Bits 0 through 22 form the mantissa
  - Bits 23 through 30 form the exponent
  - Bit 31 is the sign bit.

Floating Point Basics…cont'd

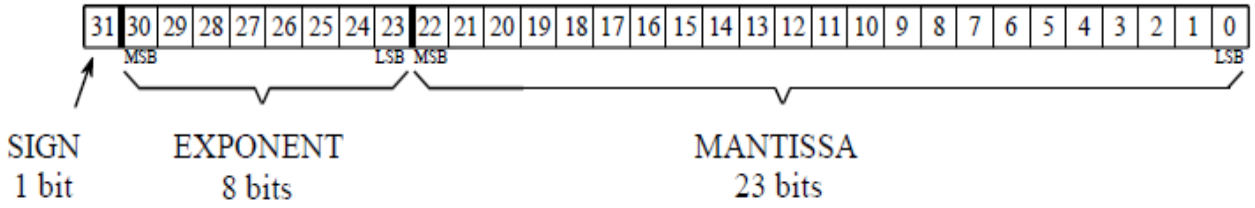
- The floating number is defined by: $v = (-1)^S \times M \times 2^{E-127}$
  - The first term S is the sign bit (0 for a positive number, 1 for negative)
  - The E is the exponent (in this case an 8 bit number with a range of -127 to 128)
  - The M forms the mantissa using 23 bits which store a binary fractional value in the format:

    n + (n^-1)/2 + (n^-2)/4 + (n^-3)/8

    So for example 1.0101 would represent 1 + 0/2 + 1/4 + 1/8

- A bit is saved since the leading number in base 2 will always be 1 (and only one non zero integer can be represented this number is implied).

Floating Point Math…cont'd



Example 1

0 00000111 11000000000000000000000

\+ 7 0.75

$$+1.75 \times 2^{(7-127)} = +1.316554 \times 10^{-36}$$

Example 2

1 10000001 01100000000000000000000

\- 129 0.375

$$-1.375 \times 2^{(129-127)} = -5.500000$$

Number Precision

- Number precision errors in computational math are similar to quantization errors in ADC and DAC converters. There are a limited number of bits/values to store a value and the spaces between are errors in representation.

- So if there are 2 bits to store a range of numbers 0-6 and we need to add 2+3 = 5

  00 = 0
  01 = 2
  10 = 4
  11 = 6
  The returned result has to be represented as 4 or 6 and therefore the precision will be limited to 1

- Precision in fixed vs floating-point math:
  - In fixed point math as above, the precision is constant throughout the range.
  - In floating point, the gaps between represented numbers vary with a greater gap between larger numbers and therefore increased precision in smaller numbers.

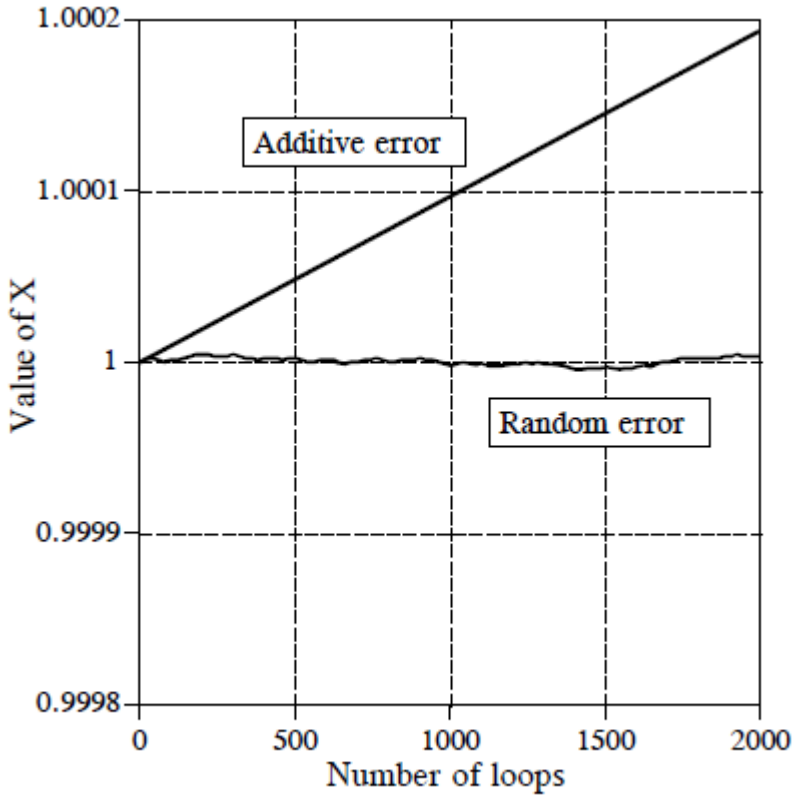
- In 32 bit precision floating point, the gap between a number and the next is approximately ten million times smaller (or 2^{-24} to 2^{-23}) which illustrates why it is larger for larger numbers and really small for the smallest numbers.

Word Length Errors

Simple program can demostrate word length errors in computational math by adding two random numbers to a signal and then subtracting them back. After several iterations the effects of the errors are clear.

Pseudocode:

```
X = 1 'initialize X
FOR I = 0 TO 2000
   A = RND 'load random numbers
   B = RND 'into A and B
   X = X + A 'add A and B to X
   X = X + B
   X = X ! A 'undo the additions
   X = X ! B
```



Reference: DSP Guide by Steven W. Smith, Ph.D.

Off-topic! Audio connections

- Stereo connector can be plugged directly to signal generator and analyzer
- TRS connector has left, right and common connection on single connector