Embedded Systems: Processor Classes

- General Purpose - high performance processors
  - Pentium, SPARC,...
    - Used for general purpose software
    - General under controle of an Operating System (Windows.., UNIX..,)
    - Workstation, PC‘s
- Embededd processors and special cores
  - 486SX, ARM, Hitachi SH7000, NEC V800, Motorola PowerPc,...
    - Single program
    - Lightweight , often under realtime OS (Windows-CE, QNX, OS-9, PSOS, ...)
    - Support for -light- digital signal processing
    - Consumer electronic, cellular phones
- Microcontrollers
  - 8-, 16-, (32) Bit, application specific chipsets with various features
    - A/D, D/A, Input, Output, Timer, Interfaces, ...
    - Cost sensitive
    - Highest volume processors for automotive, washing machines, ...
- General purpose DSPs and core design in Gate-Arrays

Embedded DSP: Introduction

- Digital Signal Processing: application of mathematical operations to digital signals from the real world.
- Signals are represented digitally as a sequence of samples
- Samples obtained from physical signals via transducers (e.g. sensors) and analog-to-digital converters (ADC)
- Back-convertion to physical signals by digital-to-analog converters (DAC)
- Digital Signal Processors (DSP) is a electronic device that processes digital signals

Embedded DSP: Common Appliactions

- Applications
  - Scientific data processing
  - Communication
  - Audio and video processing
  - Graphics
  - Navigation
  - Control - robotics, guidance, machine vision

- Common signal processing algorithms
  - Filtering
  - Transformations (time <--> frequency)
  - Convolution
  - Correlation
  - Matrix calculations

Embedded DSP: Target for DSPs

- DSP task demands:
  - Fast repetition of numeric computations
  - High memory and I/O-bandwith to move data to the computation units
  - Deterministic reaction to signals from the physical world
  - Real-time processing capabilities
- DSPs must therefore perform these tasks efficiently while minimizing:
  - Power consumption
  - Memory usage for a given application
  - Devlopment time
  - Cost
- DSPs need an excellent software support in order to have the full benefit of the hardware technology

Embedded DSP: DSP vs. General Purpose MPU

- DSPs generally run one program or a small, fixed set of programs
  - No operating system is generally necessary
  - Hence small special OSes are much simpler, there is no virtual memory (disk, ..) or protection
- DSPs often work in hard real-time applications
  - Necessary to perform the signal processing in a fixed period of time
  - Must be able to react within a fixed time to anything and everything that could happen
  - Interrupt handling: the interrupts or exceptions reduce the time for computation
  - Memory acces and -latency: How to get data to the ALU/MAC and again out of the DSP ?
- DSPs often process an infinite continuous data sequence according to time slots from the environment (e.g. audio processing with 48 KHz sampling rate)

Embedded DSP: DSP vs. General Purpose MPU

- View to the market shows that in terms of dollar, the biggest markets for DSP processors today are:
  - Digital cellular telephony systems
  - Modems
  - Pagers and Bluetooth systems (wireless)
  - Disk drive servo and general drive controls
- The engineers demand
  - Good overall performance
  - Low cost
  - Energie efficiency
  - Support for hardware
  - Efficient support for software development
    - Support by GUI based tools (Mathlab, Labview, DSPWorks, ...)
    - Algorithm support by DSP-libraries

(Application specific choose of the best matched processor)

Embedded DSP: General Features of DSPs

• Single cycle CPUs	(key of fast computation,..)
• SISD and SIMD structures	(multiple computation units,..)
• Specialized instruction set (RISC)	(efficient instructions,..)
• Integer and/or floating point units	(Scaling, high signal dynamic,..)
• Specialized computation modules (ALU/MAC/SHIFTER)	(Fast, parallel with file registers)
• Data path configured for DSP applications	(FIR, IIR: coefficients and data,.)
• Multiple memory banks and buses, cache	(Parallel acces,…)
• Specialized addressing modes	(Bit-reverse for FFT,…)
• Specialized execution control	(Fast context switching,…)
• Fast interrupt response	(Real time)
• Specialized peripherals for input/output of data	(Multiport, DMA,…)
• Specialized units for multiprocessing	(Fast LINK structures,…)

Embedded DSP: Addressing

- Standard DSPs have standard addressing modes (immediate, register indirect, direct)
- Additional performance by complex addressing modes
  - Want to keep MAC datapath performing computation
  - Offload addressing to somewehere else ouside the chip
- Modes
  - Autoincrement/autodecrement (post/pre) before/after generating the address
  - Circular
    - Data producer/consumer functions with infinite I/O
    - Producer writes after tail pointer in autoincrement mode
    - Consumer reads from head pointer in autoincrement mode
    - The pointer wrap around by a special modulo arithmetic
  - Bit-reversed (e.g. usefull for FFT-Transform)
    - The pointers increment in bit-reversed order instead of normal incrmenting

Embedded DSP: Multiplication

- One cycle multiplication
  - The most instruction in real signal algorithms are multiplies
  - A fast multiplier is a big demand

- One cycle multiply-accumulate (MAC)
  - y[n] = y[n] + n * x[i]   --> common operation in filtering (see last lecture !!)

- N-bit multiply gives an 2n-bit product
  - Number of bits for the MAC ?

- Need to round the results to minimize the bias

- Integer and/or floating point DSP ?

Embedded DSP: DSP Arithemtic

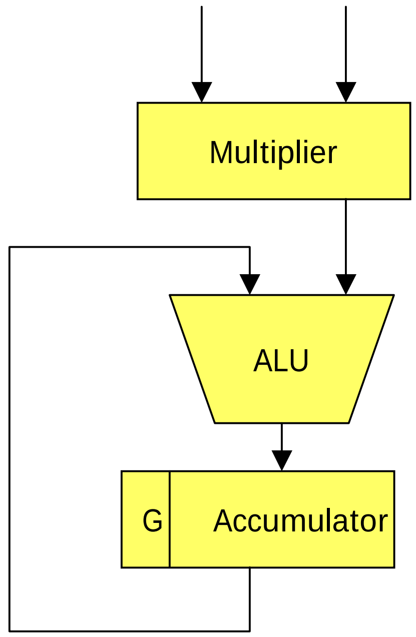
- Working with real signal - want real numbers
- DSPs support 16, 24, 32, 48, 64 (80) bit width

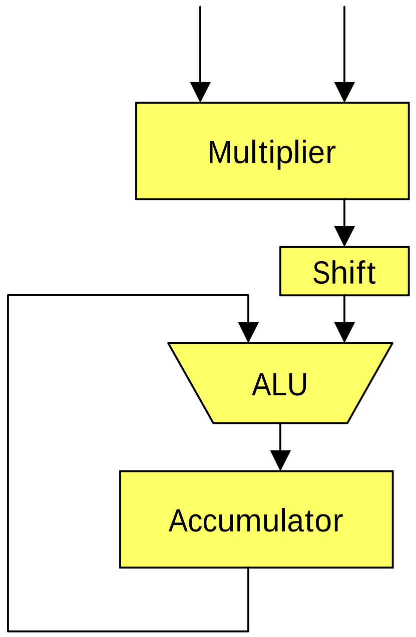
- Where is the decimal point ?
  - Programmer interprets the data in accordance to the task
  - Programmers will scale the data inside the function
  - Sometime a shift operation can prevent overflow

- DSPs work with data that was originally analog out of the real world
- Saturation arithmetic is necessary
  - Most DSP algorithms depend on special saturation arithmetic

Embedded DSP: Accumulate

- Demand: No overflow and no scale !
- Solution 1: Guard bits - make the accumulator wider than the product
- Solution 2: Scale by shifting - before adding to the accumulator





Embedded DSP: Memory

- DSP Algorithms are data-intensive
- Demand for a high data-throughput
- Minimize instruction bandwith
  - Special zero overhead loops
  - Loop buffers
  - Circular buffer
  - Bit-Reverse technology
  - One key is a cache memory for small functions
- What about data bandwith ?
  - Multiport Register File (e.g. MAC requires two new input values per cycle !)
  - Multiple data banks for program and data
  - Fast internal memory; but extern memory ?
  - What about program and/or data cache ?

Embedded DSP: Why not a CISC processor ?

- DSPs takes their power by doing a lot of things in parallel
  - MAC
  - Complicated addressing
  - Special I/O-controller hardware for data management
- Why not use a superscalar processor architecture ?

Superscalar is not predictable

- **Very Long Instruction Words (VLIW) is !**
- **Example: SHARC**

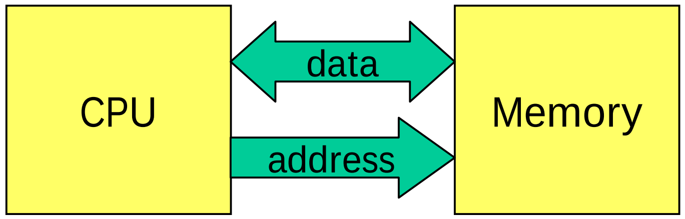
Embedded DSP: DSP versus GP Processors

- Specialized for fast computation on signals
- MIPS/MFLOPS is only MAC speed
- Performance of algorithm is the target
- Really fast only by special program techniques and a deep understanding of the hardware !

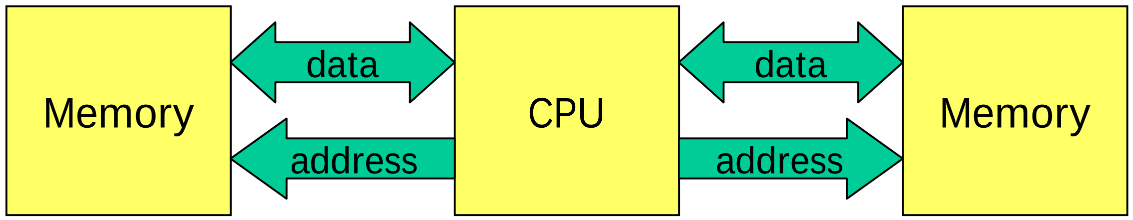
Are DSPs obsolete ?

- Additional performance improvements of general microprocessors and powerful media extensions increase the performance of GP processors to approach DSP performance.
- Can GP processors do the tasks of DSP ?

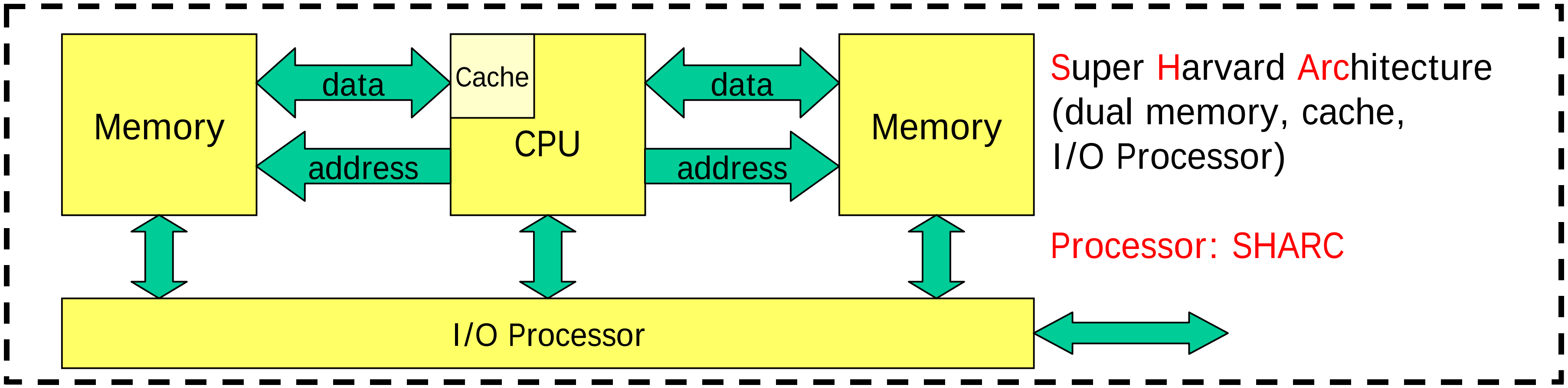
Embedded DSP: General Architectures



Von Neumann
(single memory)



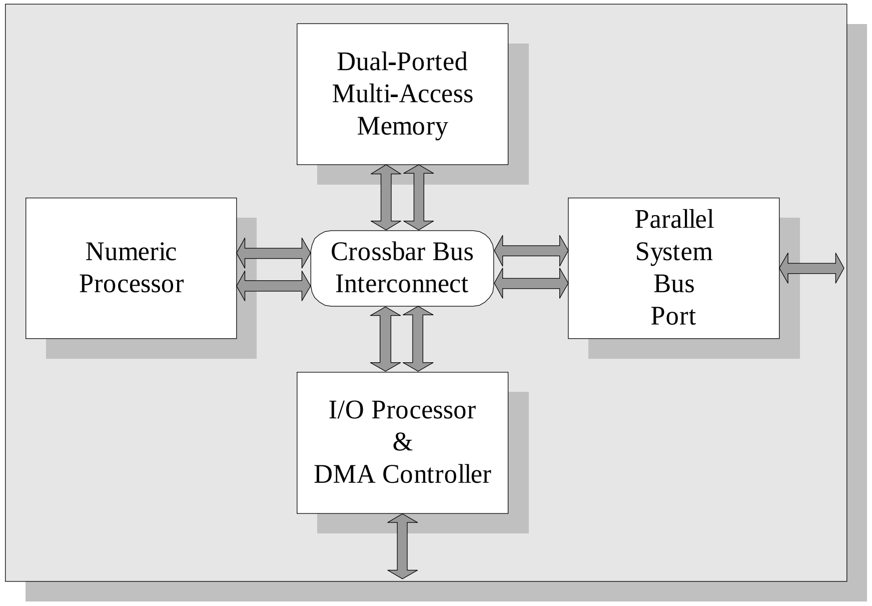
Harvard Architecture
(dual memory)



Super Harvard Architecture
(dual memory, cache,
I/O Processor)

Processor: SHARC

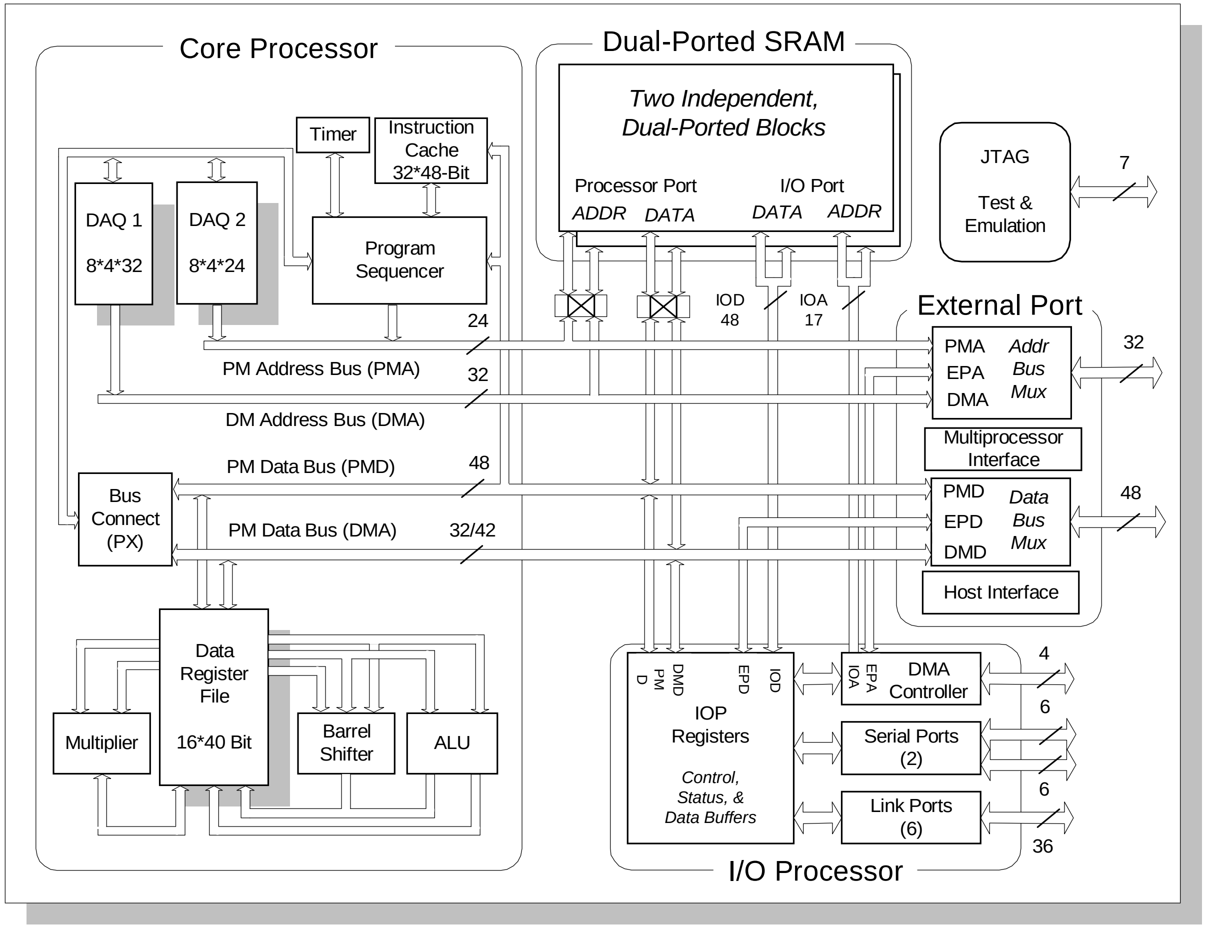
Embedded DSP: SHARC Architecture


Dual-Ported Multi-Access Memory
Numeric Processor
Crossbar Bus Interconnect
Parallel System Bus Port
I/O Processor & DMA Controller

SHARC:
Super Harvard Architecture

- A crossbar switch connecting the core numeric processor to an independant I/O processor, dual-ported memory, and parallel system bus port.

Embedded DSP: SHARC Architecture



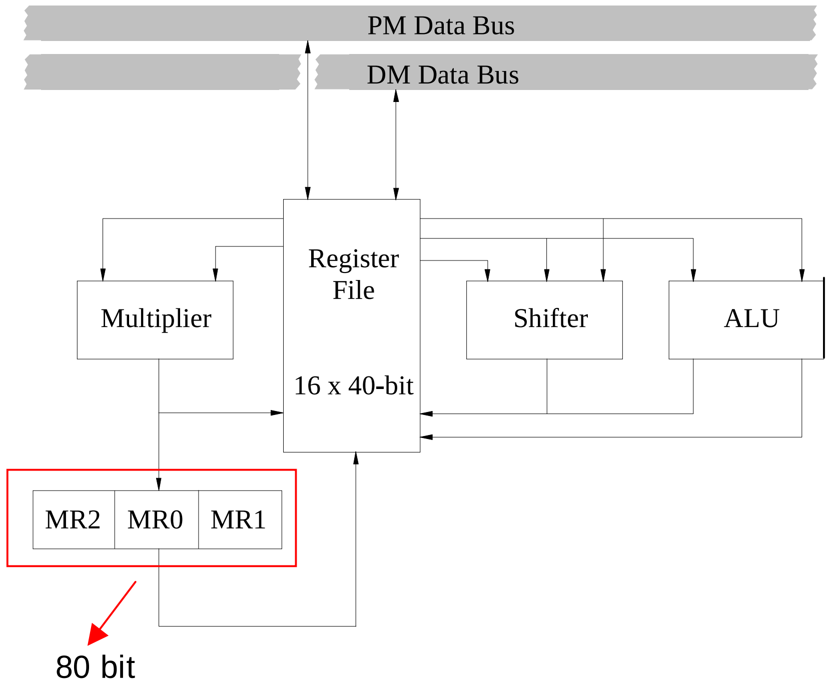
Main Technical data:

- 32-Bit IEEE FP unit (Multiplie, ALU, and Shifter)
- Data Register File
- Data Address Generator (DAG1, DAG2)
- Program Sequencer with Instruction Cache
- Interval Timer
- Dual-Ported SRAM
- External Port for Interfacing to Off-Chip Mermory & Peripherals
- Host-Port & Multiprocessor Interface
- Serial Ports
- Link Ports
- JTAG Test Access Port

Embedded DSP: Why Floating Point ?

- A digital signal processors data format detemines the ability to handle signals of various
  - precisions
  - dynamic range
  - signal-to-noise ratios
- Dynamic Range:
  - Compression and Decompression algorithms have operated on known bandwidth.
  - Adaptive filtering and imaging are two applications rquiring wide dynamic range.
- Signal-to Noise Ratio:
  - Radar and sonar or speech recognition require wide dynamic range in order to decrease signals from noisy environment.
- In general, 32-bit floating point DSPs are easier to use and allow a quicker and cheaper time-to-market (no scaling !)
- Consistency with IEEE of workstations is a real benefit.

Embedded DSP: Core Processor



- The core processor consists of
  - Three computation units ALU, MAC with a fixed point accumulator, SHIFTER
    - three formats: 32-bit fixed point, 32-bit floating-point (IEEE), 40-bit floating-point
    - ALU performs a standard set of arithmetic and logic operations in fixed and floating point formats.
    - Multiplier performs floating-point and fixed-point multiplications as well as fixed-point multiply/add and multiply/subtract operations.
    - The shifter performs logical and arithmetic shifts, bit manipulation, field deposit and extraction and exponent derivation operations on 32-bit operands.
    - The computatiion units perform single-cycle operations (no computation pipeline). The output of any unit may be the input of any unit on the next cycle. In a multifunction computation, the ALU and multiplier perform independant, simultaneous operations.
  - A register file is used to transfer data between the computation units and the data buses and for temporary storage of intermediate result. Two sets (foreground/background) with each 16 register with 32-bit.

Embedded DSP: Computation Units

- Numeric processing arranged in parallel
- General example fixed-point/floating-point:
  - F0=F1*F2 for loating-point multiply, R0=R1*R2 for fixed-point multiply

ALU Instructions (43)

Rn=Rx+Ry	Fn=Fx+Fy
Rn=Rx-Ry	Fn=Fx-Fy
Rn=(Rx+Ry)/2	Fn=(Fx+Fy)/2
COMP(Rx,Ry)	COMP(Fx,Fy)
Rn=Rx+1	Fn=ABS(Fx+Fy)
Rn=Rx-1	Fn=ABS(Fx-Fy)
Rn=-Rx	Fn=-Fx
Rn=ABS Rx	Fn=ABS Fx
Rn=PASS Rx	Fn =PASS Fx
Rn=Rx AND Ry	Fn=RND Fx
Rn=Rx OR Ry	Fn=LOGB Fx
Rn=Rx XOR Ry	Fn=ABS Fx
Rn=NOT Rx	Fn=MIN(Fx,Fy)
Rn=MIN(Rx,Ry)	Fn=MAX(Fx,Fy)
Rn=MAX(Rx,Ry)	Fn=RSQRTS Fx
……..	……

MAC Instructions (26)

Rn=Rx*Ry (...)	Fn=Fx*Fy
MRF=Rx*Ry (..)	
MRB=Rx*Ry (..)	
Rn=MRF + Rx*Ry (..)	
Rn=MRB + Rx*Ry (..)	
MRF=MRF + Rx*Ry (..)	
MRB=MRF + Rx*Ry (..)	
MRF=0	
MRB=0	
MRF=Rn	
MRB=Rn	
... =RND ...(..)	
... =SAT ...(..)	
……	

(..):
S = Signed input
U = Unsigned input
I = Integer input
F = Fractional input
FR = Fractional inputs, rounded output
SF = Default format for 1 -input operation
SSF= Default format for 2 -input operations

Embedded DSP: Computation Units

- Shifter operates on 32-bit fixed-point operands.
  - Shift and rotates from off-scale left to off-scale-right
  - Bit manipulation including bit set, clear, toggle and test
  - Bit field manipulation including extract and deposit
  - support for fixed point to floating point conversion

SHIFTER Instructions (36)

Rn=LSHIFT Rx BY Ry	Rn=BSET Rx BY Ry
Rn=LSHIFT Rx BY <data>	Rn=BSET Rx BY <data>
Rn=Rn OR LSHIFT Rx BY Ry	Rn=BTGL Rx BY Ry
Rn=Rn or LSHIFT Rx BY <data>	Rn=BTGL Rx BY <data>
Rn=ASHIFT Rx BY Ry	Rn=BCLR Rx BY Ry
Rn=ASHIFT Rx BY <data>	Rn=BCLR Rx BY <data>
Rn=Rn OR ASHIFT Rx BY Ry	BTST Rx BY Ry
Rn=Rn or ASHIFT Rx BY <data>	BTST Rx BY <data>
Rn=ROT Rx BY Ry	Rn=FDEP Rx BY Ry
Rn=ROT Rx BY <data>	Rn=FDEP Rx BY <data>
Rn=BCLR Rx BY Ry	Rn=FEXT Rx BY Ry
Rn=BCLR Rx BY <data>	Rn=FEXT Rx BY <data>
	……..

Embedded DSP: Computation Units

Field deposit instruction: R0=FDEP R1 BY R2

R1 = 0x000000FF
R2 = 0x00000210

```
R2:   00000000.00000000.00000010.00010000

                           Len6=8    bit6=16

R1:   00000000.00000000.00000000.11111111

R0:   00000000.11111111.00000000.00000000
        Bit 31                          Bit 0
```

Field extract instruction: R3=FEXT R4 BY R5

R4 = 0x87880000
R5 = 0x00000217

```
R5:   00000000.00000000.00000010.00010111

                           Len6=8    bit6=23

R4:   10000111.1 0001000.00000000.00000000

R3:   00000000.00000000.00000000. 00001111
        Bit 31                          Bit 0
```

Embedded DSP: Computation Units

Multifunction Instructions with MAC/ALU:

Ra=Rx+Ry, Rs=Rx-Ry;

Rm=R(3-0) * R(7-4) (SSFR),
MRF=MRF + R(3-0) * R(7-4) (SSFR),
Rm= MRF + R(3-0) * R(7-4) (SSFR),
MRF=MRF - R(3-0) * R(7-4) (SSFR),
Rm= MRF - R(3-0) * R(7-4) (SSFR),

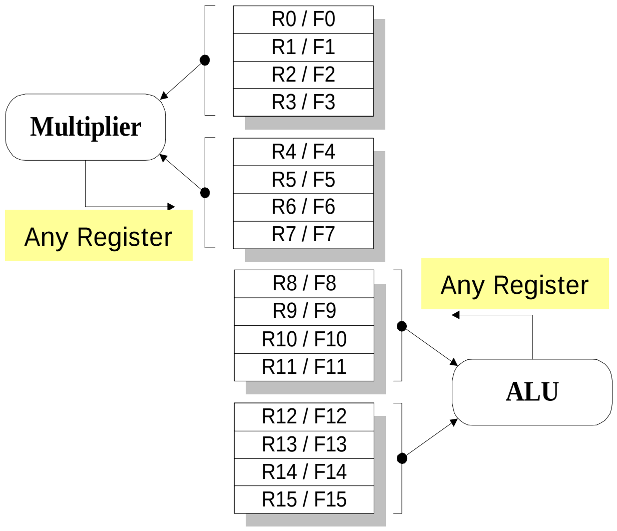
Ra=R(11-8) + R(15-12);
Ra= R(11-8) - R(15-12);
Ra= (R(11-8) - R(15-12))/2;

Floating-Point Multiplication and ALU Operation

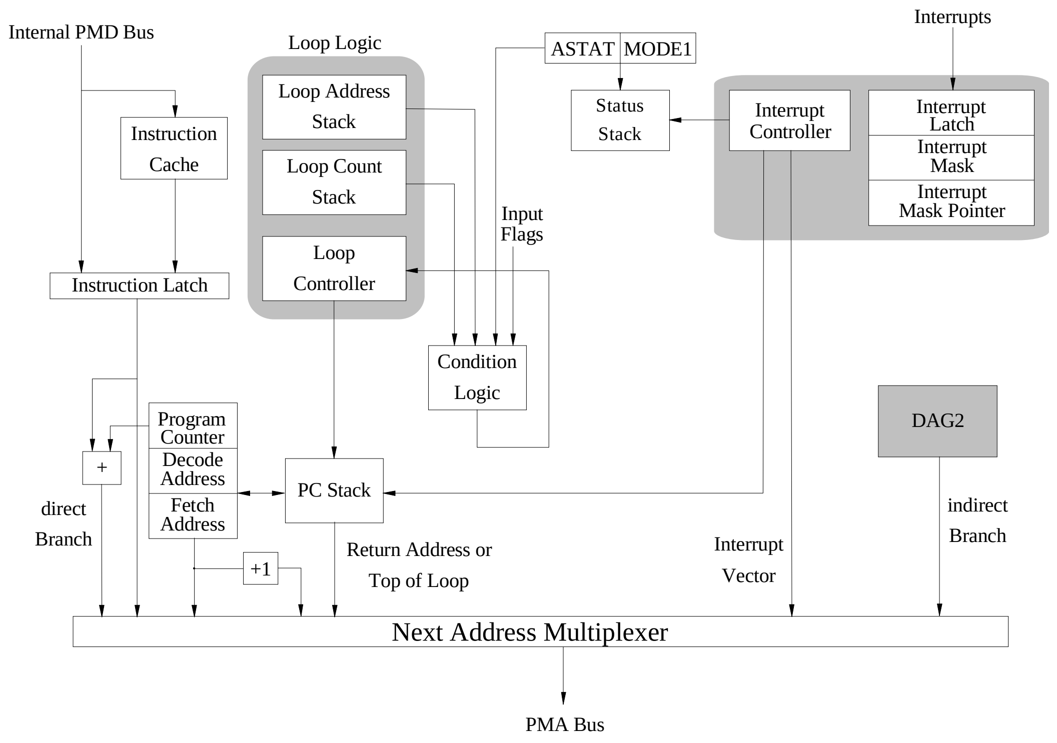
Fm=F(3-0) * F(7-4),

Fa=F(11-8) + F(15-12);
Fa=F(11-8) - F(15-12);
Fa=FLOAT R(11-8) BY R(15-12);
Fa=ABS (F(11-8),(F(15-12));
Fa=MAX F(11-8) + F(15-12);
Fa=MIN   F(11-8) + F(15-12);
....

Multiplication and Dual Add/Subtract
Rm=R(3-0) * R(7-4) (SSFR), Ra=R(11-8) + R(15-12), Rs=R(11-8) - R(15-12);
FM=F(3-0) * F(7-4), Fa= F(11-8) + F(15-12), Rs=F(11-8) - F(15-12);



Embedded DSP: Core Processor

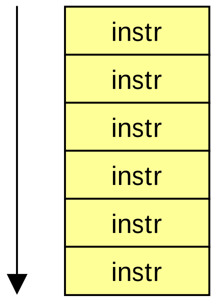


Remember to the pipelined operation:
fetch/decode/execute

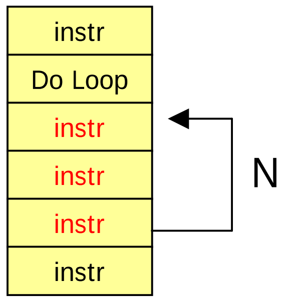
- The core processor consists of ....
  - Program sequencer
    - Address generators
    - DAGs with pre- and postmodify and length register (circular buffer)
    - Instruction cache memory (32*32 bit)
    - Loop Logic with zero overhead
    - Branch Logic
    - Control & Status register
    - Program counter
    - PC stack unit
  - Interrupt controller
  - external: 3, internal: 26 (vectorized by fixed addresses and priority)
  - Programmable timer (timer for periodically interrupts, watch-dog function)

Embedded DSP: Program Sequencer

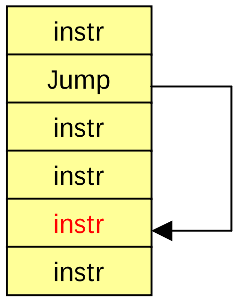
- Program Flow by:
  Loops, subroutines, jumps, interrupts, idle
  - incrementing the fetch address,
  - maintaining stacks,
  - evaluating conditions,
  - decrementing loop counter
  - calculating new addresses
  - maintaining an instruction cache
  - handling interrupts



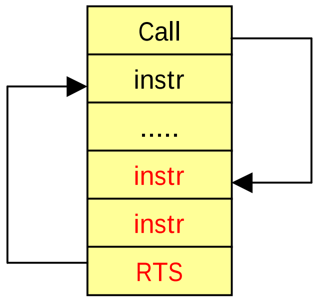
Linear Flow



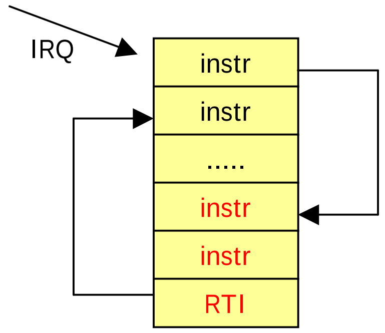
Loop



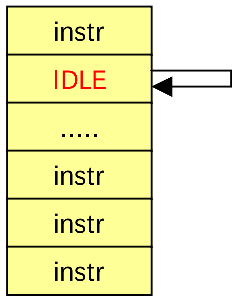
Jump



Subroutine



Interrupt



Subroutine

Embedded DSP: Overwiev SHARC Instructions

Program Flow Control Instructions

Nr.									Addressing Mode
8a	If condition	JUMP	|<addr24>| |(PC,<reladdr24>)|		|(DB)| |(LA)| |(CI)| (|DB,LA)| |(DB,CI)|	;			Direct addressing Relative addressing
8b	If condition	CALL	|<addr24>| |(PC,<reladdr24>)|		|(DB)|	;			Direct addressing Relative addressing
9a	If condition	JUMP	(Md,Ic) (PC,<reladdr6>)		|(DB)| |(LA)| |(CI)| (|DB,LA)| |(DB,CI)|	,	|compute| |ELSE compute|	;	Indirect ‚pre-modify' Relative addressing
9b	If condition	CALL	(Md,Ic) (PC,<reladdr6>)		|(DB)|	,	|compute|; |ELSE compute|;		Indirect ‚pre-modify' Relative addressing
10	If condition	JUMP	(Md,Ic), (PC,<reladdr6>),	,	ELSE		|compute, DM(Ia,Mb)=dreg| |compute, dreg=DM(Ia,Mb)|	;	Indirect ‚pre-modify' Relative addressing
11a	If condition	RTS	|(DB)|, |(LR)|, |(DB,LR)|,	,	|compute| |ELSE compute|	;			Direct addressing Relative addressing
11b	If condition	RTI	(DB),	,	|compute|; |ELSE compute|	;			Direct addressing
12	LCNTRL=	|<data16>| , |ureg|	DO |<addr24| |(PC,<reladdr24>)|		UNTIL LCE	;			Direct addressing Relative addressing
13	DO	|<addr24>| |(PC,<reladdr24>)|	UNTIL termination	;					Direct addressing Relative addressing

- Condition and Loop Termination Codes:

		True if
EQ	ALU equal zero	AZ=1
LT	ALU less than zero	..
LE	ALU less than or equal zero	..
AC	ALU carry	AC=1
AV	ALU overflow	AV=1
MV	Multiplier overflow	MV=1
MS	Multiplier sign	MS=1
SV	Shifter overflow	SV=1
SZ	Shifter zero	SZ=1
...		
Complements of the above conditions:		
NEQ		AZ=0
GE		..
GT		..
NOT AC		AC=0
....		

Examples:

```
CALL init (DB);

JUMP (M8,I12), R6=R6-1;

IF EQ CALL(PC,17) (DB), ELSE R6=R6-1;

IF NOT GT RTS(DB);

IF SZ RTS, ELSE R0=LSHIFT R1 BY R15;

LCNTR=100, DO fmax UNTIL CE;

LCNTR=R12, DO (PC,16) UNTIL CE;

DO end UNTIL FLAG1_IN

DO (PC,7) UNTIL AC
```

Embedded DSP: Program Sequencer

Branches (Loops and Calls) can be delayed (DB) or nondelayed:

- If nondelayed, the two instructions after the branch, which are in the fetch and decode stages, are not executed.
- For a call the decode address (the address of the instruction after the call) is the return address.
- During the two no-operations cyles, the first instruction at the branch address is fetched and decoded.

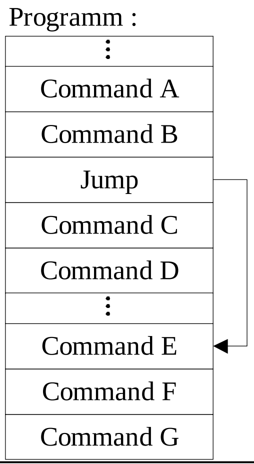
- If delayed, the processor continues to execute two more instructions while the instruction at the branch address is fetched and decoded.
- In the case of a call, the return address is the third address after the branch instruction.
- A delayed branch is more eficient, but it makes the code harder to understand because of the instructions between the branch instructions and the actual branch.

- Restrictions: Instructions in the two lines following a delayed branch may not be:
  - Other Jumps, Calls or Returns
  - Pushes or Pops of the PC stack
  - Writes to the PC stack or PC stack pointer
  - DO UNTIL instruction
  - IDLE instruction

Embedded DSP: Program Sequencer

non delayed branch:
```
JUMP label;
(NOP;)
(NOP;)
```

delayed branch:
```
JUMP label (DB)
R1=DM(I0,M0);
R2=PM(I8,M8);
```



Example: Pipeline execution

delayed or not delayed branch ?

Pipeline :

fetch	Jump	C	D	E	F	G
decode	B	Jump	C	empty	E	F
execute	A	B	Jump	empty	empty	E

```
LCNTR=30, DO label1 UNTIL CE;
            F0=DM(I0,M0), F2=PM(I8,M8);
            F1=F0*F2;
label1:     F4=F1+F4;
```

Embedded DSP: Data Addressing

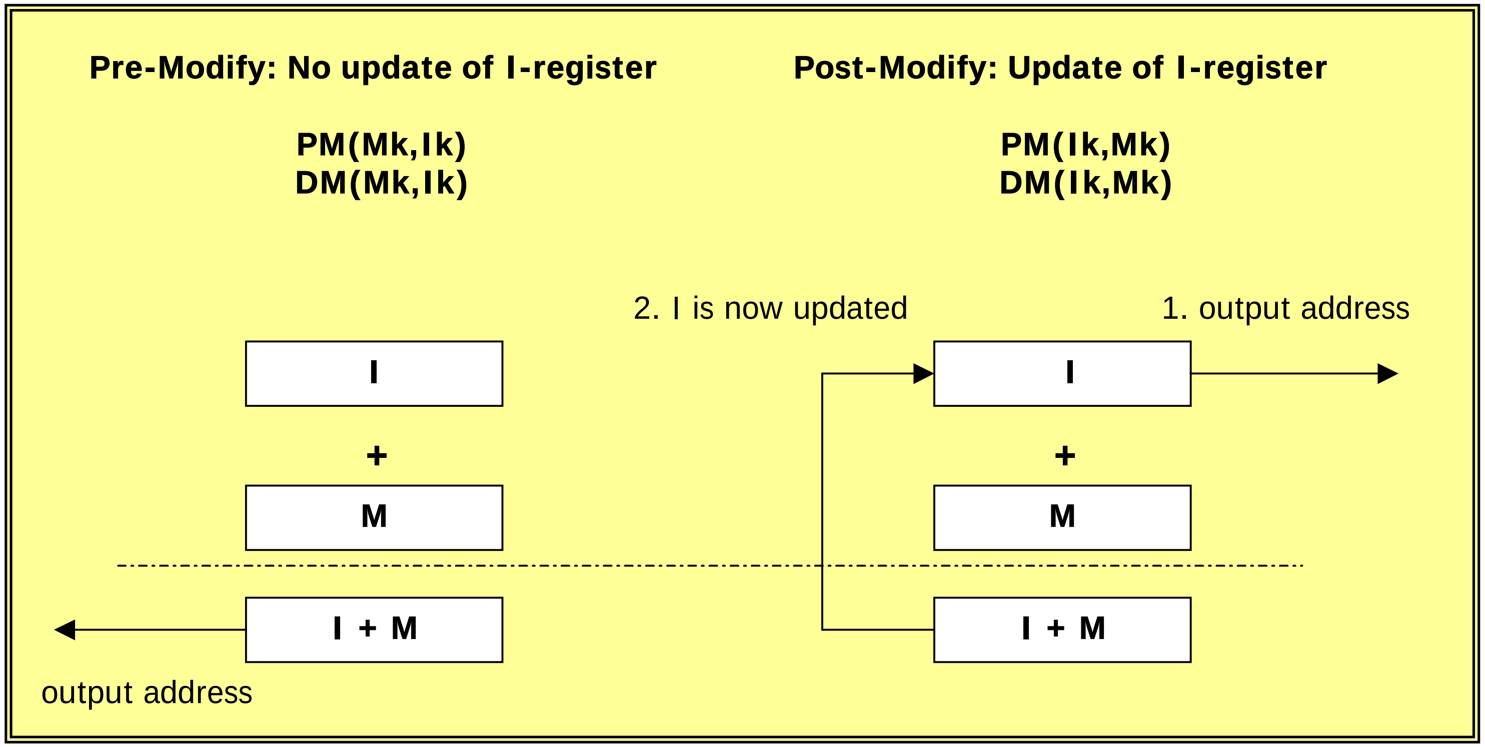
Overview

•2 data address generators (DAG1 and DAG2)
• indirect address access, address indirect by the content of a DAG
• DAG1 addresses 32-bit on data bus DM
• DAG2 addresses 24-bit on program bus PM
• Additional alternate (secondary) register for fast context switching
• Special support for signal processing applications
    • Circular data buffers
    • Bit-reversing
• Each has 4 types of special registers:

• Index (I)	= pointer to memory	[DAG1: I0-I7]
		[DAG2: I8-I15]
• Modify (M)	= increment/decrement value	[M0-M7]
		[M8-M15]
• Base (B)	= base address of a circular buffer	[B0-B7]
		[B8-B15]
• Length (L)	= length of a circular buffer	[L0-L7]
		[L8-L15]

DAG operation
• address output (pre-modufy or post-modify)
• modulo addressing (circular buffers)
• addressing in bit-reverse order

Embedded DSP: Data Addressing



Pre-Modify versus Post-Modify

Modifier Instructions

- **R6 = PM(I12,M11);** Indirect addressing PM-memory with post-modify
  - R6 = content of I12
  - I12 = I12 + M11
- **R6 = PM(M11,I12);** Indirect addressing PM-memory with pre-modify
  - R6 = content of (I12 + M11)
  - I12 is not changed
- **DM(M1,I2) = TCOUNT**; Indirect addressing DM-memory
  - Store TCOUNT in DM address (I2+M1)
- **R2 = DM(0x40000012, I1);** Immediate 32-bit modify: address = I1 + 0x40000012
- **F6 = F1 + F3, PM(I8,0x0A)=ASTAT;** Immediate 6-bit modify: address=I8, I8=I8+0x0A

Instruction Set Reference

- Compute and Move or Modify Instructions, which specify a compute operation in parallel with one or two data moves or an index register
- Program Flow Control instructions, which specify vaious types of branches, calls returns and loops. Some of these instructions may also specify a compute operation and/or a data move
- Immediate Data Move instructions, which use immediate instruction fields as operands, or use immediate instruction fields for addressing.
- Miscellaneous instructions, such as bit modify and test, no operation and idle

Embedded DSP: Overwiev SHARC Instructions

Compute & Move or Modify Instructions

Nr.	Instruction							Addressing mode
1		compute	,	|DM(Ia,Mb)=dreg1| |dreg1=DM(Ia,Mb)|	,	|PM(Ic,Md)=dreg2| |dreg2=PM(Ic,Md)|	;	Indirect addressing ‚post-modify'
2	If condition	compute	;					
3a	If condition	compute	,	|DM(Ia,Mb)=ureg| |PM(Ic,Md)=ureg|	;			Indirect addressing ‚post-modify'
3b	If condition	compute	,	|DM(Ma,Ib)=ureg| |PM(Mc,Id)=ureg|	;			Indirect addressing ‚pre-modify'
3c	If condition	compute	,	|ureg=DM(Ia,Mb)| |ureg=PM(Ic,Md)|	;			Indirect addressing ‚post-modify'
3d	If condition	compute	,	|ureg=DM(Ma,Ib)| |ureg=PM(Mc,Id)|	;			Indirect addressing ‚pre-modify'
4a	If condition	compute	,	|DM(<data6>)=dreg| |PM(<data6>)=dreg|	;			Immediate addressing mode
4b	If condition	compute	,	|DM(<data6>,Ia)=dreg| |PM(<data6>,Ic)=dreg|	;			Immediate addressing mode ‚pre-modify'
4c	If condition	compute	,	|dreg=DM(<Ia,<data6>)| |dreg=PM(Ic,<data6>)|	;			Immediate addressing mode ‚pre-modify'
4d	If condition	compute	,	|dreg=DM(<data6>,Ia)| |dreg=PM(<data6>,Ic)|	;			Addressing mode: ‚Immediate-pre-modifier'
5	If condition	compute	,	ureg1=ureg2	;			Register transfer
6a	If condition	shiftimm	,	|DM(Ia,Mb)=dreg| |PM(Ic,Md)=dreg|	;			Indirect addressing ‚post-modify'
6b	If condition	shiftimm	,	|dreg=DM(Ia,Mb)| |dreg=PM(Ic,Md)|	;			Indirect addressing ‚post-modify'
7	If condition	shiftimm	,	|Modify(Ia,Mb)| |Modify(Ic,Md)|	;			Modify address (Ia=Ia+Mb) (Ic=Ic+Md)

Embedded DSP: Overwiev SHARC Instructions

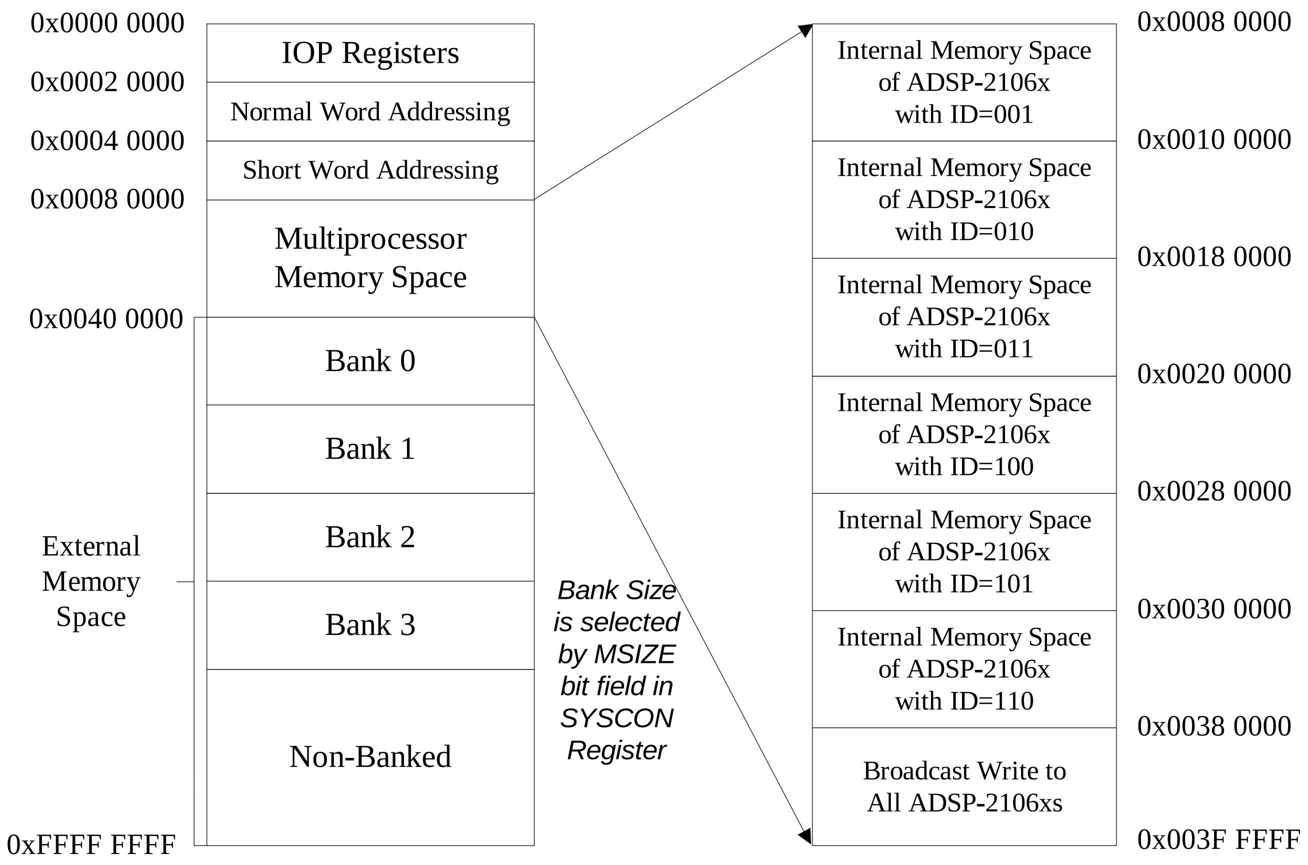
Immediate Move Instructions

Nr.			Addressing Mode
14a	|DM(<addr32>)| = ureg |PM(<addr24>)|	;	Direct addressing
14b	ureg=|DM(<addr32>)| |PM(<addr24>)|	;	Direct addressing
15a	|DM(<data32>)| = ureg |PM(<data24>)|	;	Direct addressing
15b	ureg=|DM(<data32>)| |PM(<data24>)|	;	Direct addressing
16	|DM(Ia,Mb)| = <data32> |PM(Ic,Md)|	;	Direct addressing
17	ureg = <data32>	;	Direct addressing

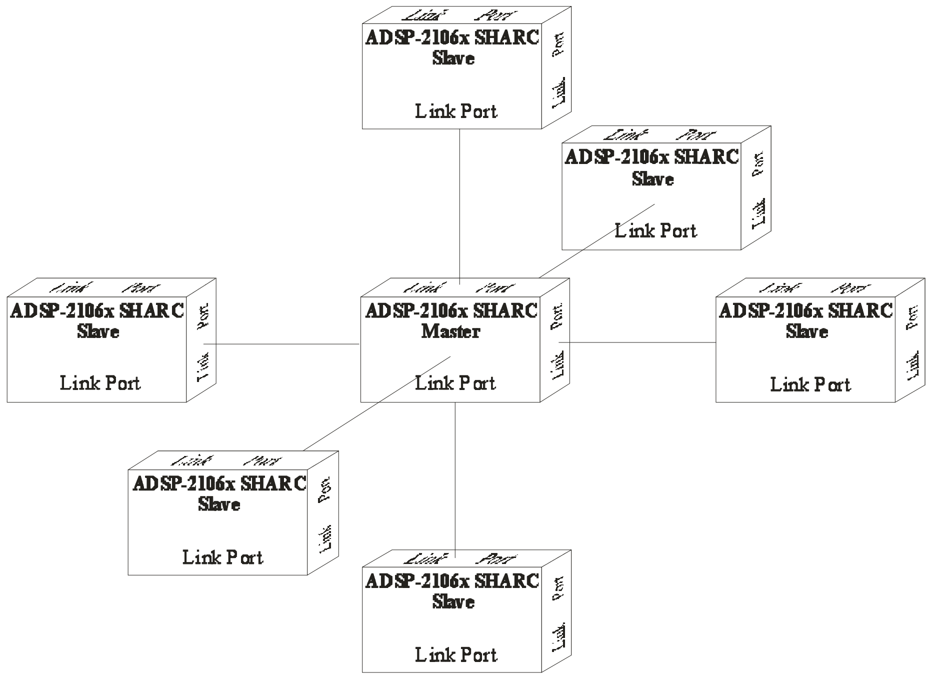
Miscellaneous Instructions

Nr.	Instruction									
18	BIT	|SET| |CLEAR| |TGL| |TST| |XOR|		sreg <data32>		;				
19a	MODIFY	|(Ia,<data32>| |(Ic,<data24>|	;							
19b	BITREV	|(Ia,<data32>| |(Ic,<data24>|	;							;
20	|PUSH| |POP|	LOOP	,	|PUSH| |POP|	STS	,	|PUSH| |POP|	PCSTK	, FLUSH CACHE	;
21	NOP ;									
22	IDLE ;									
23	CJUMP	|function| |(PC,<reladdr24>|		(DB)		;				
24	RFRAME ;									

Embedded DSP: Internal Memory


0x0000 0000
IOP Registers
0x0002 0000
Normal Word Addressing
0x0004 0000
Short Word Addressing
0x0008 0000
Multiprocessor Memory Space
0x0040 0000
Bank 0
Bank 1
Bank 2
Bank 3
Non-Banked
External Memory Space
0xFFFF FFFF
Bank Size is selected by MSIZE bit field in SYSCON Register
0x0008 0000
Internal Memory Space of ADSP-2106x with ID=001
0x0010 0000
Internal Memory Space of ADSP-2106x with ID=010
0x0018 0000
Internal Memory Space of ADSP-2106x with ID=011
0x0020 0000
Internal Memory Space of ADSP-2106x with ID=100
0x0028 0000
Internal Memory Space of ADSP-2106x with ID=101
0x0030 0000
Internal Memory Space of ADSP-2106x with ID=110
0x0038 0000
Broadcast Write to All ADSP-2106xs
0x003F FFFF

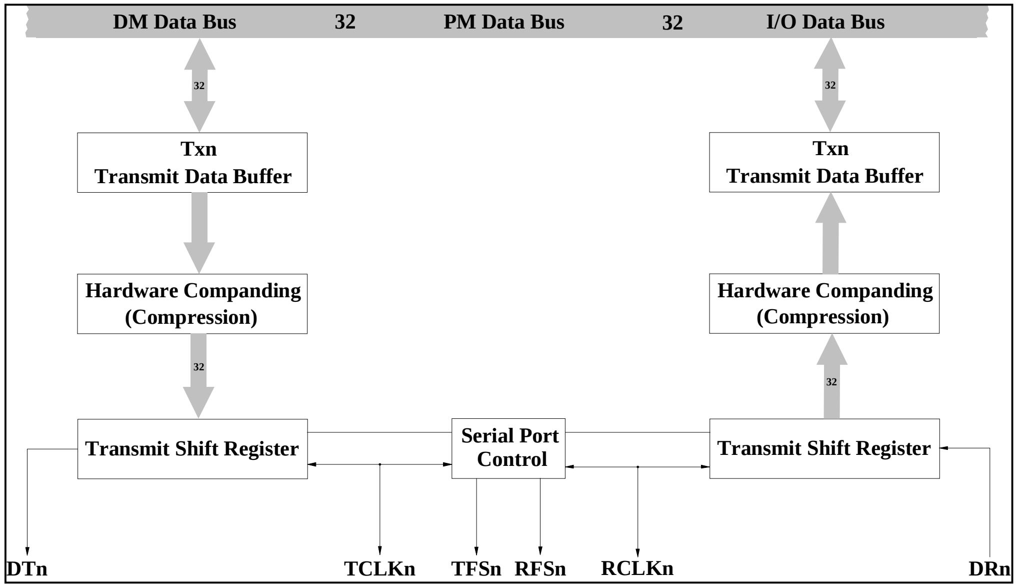
Embedded DSP: Multiprocessing by LINK ports



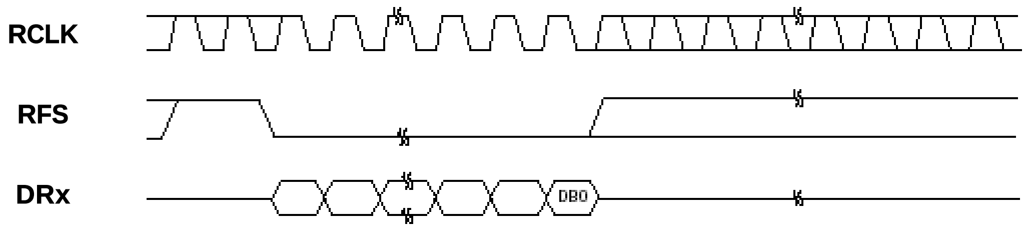
- Link ports for high speed point-to-point data transfers to other processors

- SHARC: 21062: 6 Link ports
- 4 -bit ports
- transmit and receive
- double buffer register
- Handshake signals REQ & ACK
- Single cyle or DMA
- Interrupts
- Fully asnychron to the core
- only 6 pins !

Embedded DSP: Serial Ports

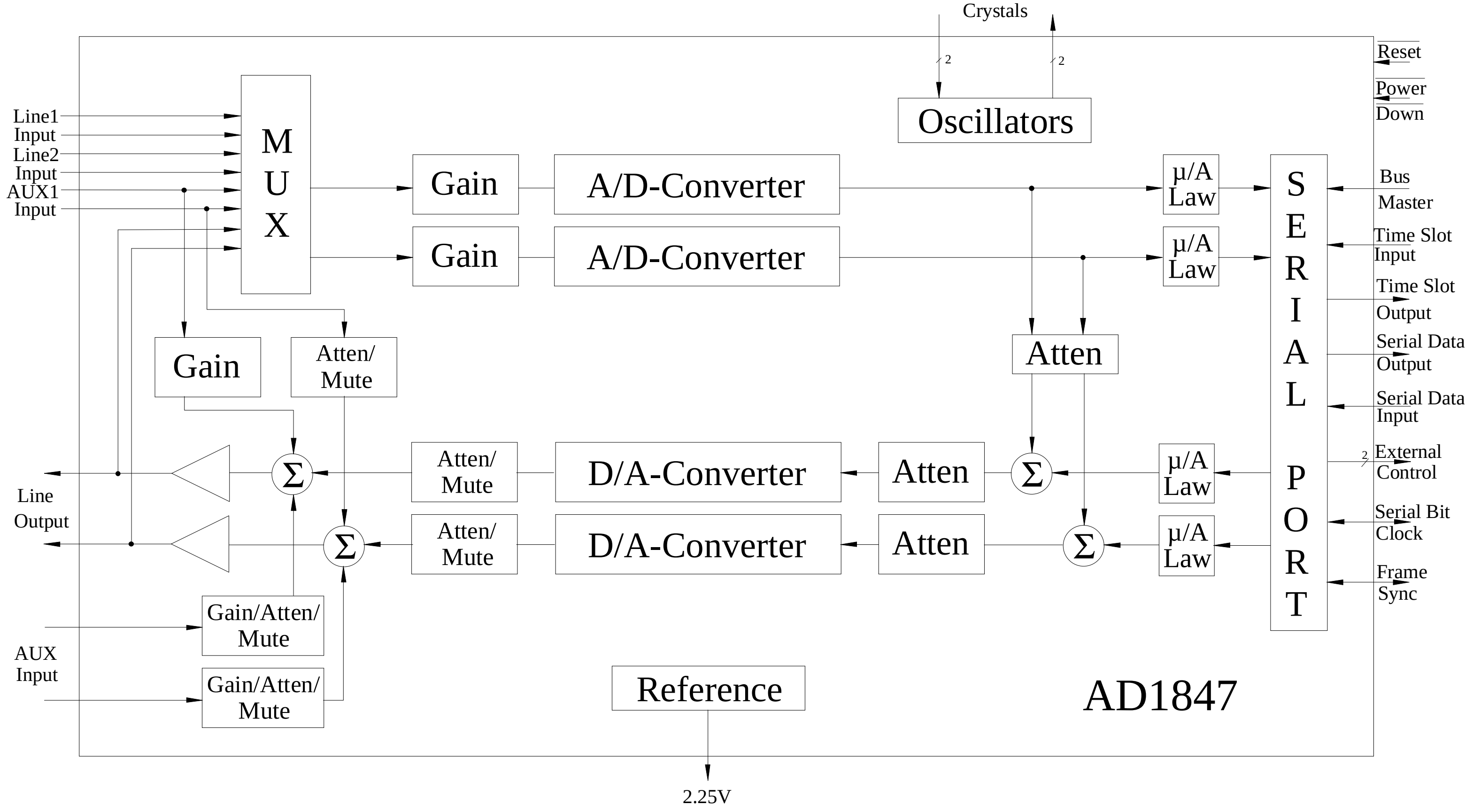


3 Wire Interface



RCLK:	Receive Clock
RFS:	Receive Frame Signal
DRx:	Receive Data
TCLK:	Transmit Clock
TFS:	TransmitFrame Signal
DTx:	Transmit Data

Embedded DSP: Serial Port to a CODEC



Time and Frequency Considerations of SSI based Converters

Serial Interface		
Cycle time [us]	Transfer rate [Mbit/sec]	Transfer time 16 Bit [us]
1	1	16
0.1	10	1,6
0.05	20	0,8
0.03	33	0,48

	Processor
Cycle time [ns]	Count of Instructions
0.025	640
0.025	64
0.025	32
0.025	19

Analog Signal
Frequency
< 32 kHz
< 312 kHz
< 625 kHz
< 1 MHz

Result:

- Use the highest possible data rate on SSI interface
- Use DMA transfer to releave the processor
- Use interrupt technology for event triggering

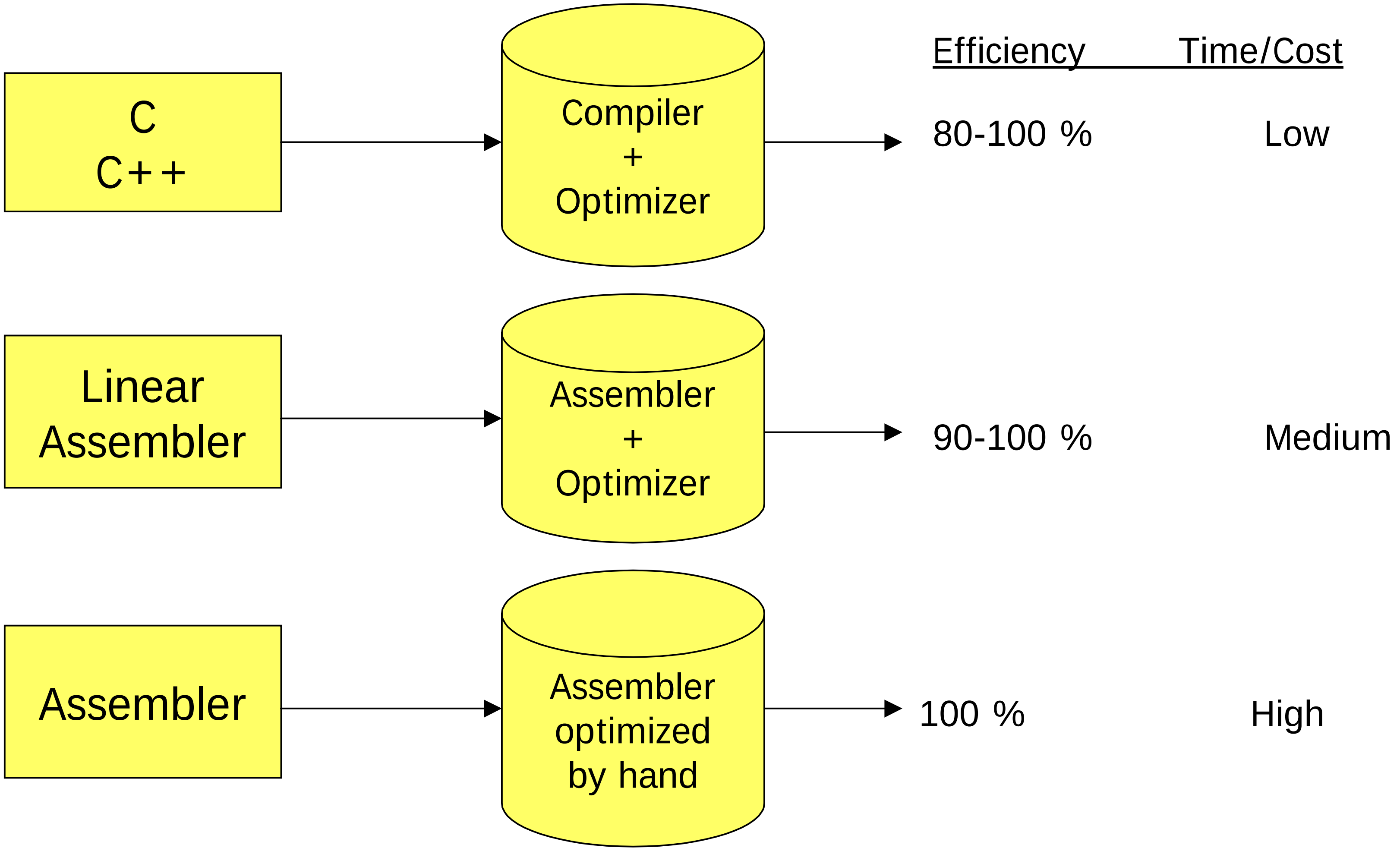
- Typicall situation for audio processing
    - 48 kHz sample rate
    - 20.8 us / sample
    - 20.8 us = interrupt
    - 40 MHz DSP <-> at least 832 single instructions

Embedded DSP: Software

Outline

- Programming Methods
  - Text editor based
  - Visual DSP Studio
- Projects
- Working with C
- Working with Assembler

Embedded DSP: Software



```
void CCpyFifoToMem()
{
   for (i = 0; i < TsCnt; i++) {
          value = FIFO(4) >> 16;
          rbuf[i] = abs(value);
   }
}
```

C-Program

ASM

18

```
•extern              _abs;
.global              _CCpyFifoToMem;
_CCpyFifoToMem:
                     modify(i7,-1); nop;
                     r2=0;
                     dm(_i)=r2;
_L$249:
                     r2=dm(_i);
                     r4=dm(_TsCnt);
                     comp(r2,r4);
                     if ge jump (pc, _L$250) ;
                     r2=dm(4456452);
                     r4=ashift r2 by -16;
                     dm(-2,i6)=r4;
                     r2=dm(_i);
                     r4=_rbuf;
                     r8=r4+r2;
                     i4=r8;
                     r2=dm(-2,i6);
                     r4=abs r2;
                     dm(i4,m5)=r4;
_L$251:
                     r2=dm(_i);
                     r4=r2+1;
                     r2=r4;
                     dm(_i)=r2;
                     jump (pc, _L$249) ;
_L$250:
_L$248:
                     i12=dm(-1,i6);
                     jump (m14,i12) (DB);
                     nop;
                     RFRAME;
```

Assembler hand coded

4

```
.SEGMENT /pm    seg_pmco;
.global
_ASMFastCpyExtAbsFifoToMem;
.extern         _rbuf;
.extern         _TsCnt;
_ASMFastCpyExtAbsFifoToMem:
   entry;
   puts=r0;
   puts=r2;
   r0=i1;  puts=r0;
   r0=i4;  puts=r0;

   i4=0x440004;
   i1=_rbuf;
   r0=dm(_TsCnt);
   lcntr=r0, DO fifo_abs UNTIL LCE;
         r2=dm(i4,m5);
         r2=abs r2;
         r2=ashift r2 by 0xfffffff0;
fifo_abs:   dm(i1,m6)=r2;

   r0=gets(1);  i4=r0;
   r0=gets(2);  i1=r0;
   r2=gets(3);
   r0=gets(4);
   alter(4);
   exit;
.endseg;
```

Assembler hand coded

```
.SEGMENT /pm    seg_pmco;
.global
_ASMFastCpyExtAbsFifoToMem;
.extern         _rbuf;
.extern         _TsCnt;
_ASMFastCpyExtAbsFifoToMem:
    entry;
    puts=r0;
    puts=r2;
    r0=i1;  puts=r0;
    r0=i4;  puts=r0;

    i4=0x440004;
    i1=_rbuf;
    r0=dm(_TsCnt);       /* TsCnt */
    lcntr=r0, DO fifo_abs UNTIL LCE;
            r2=dm(i4,m5);
            r2=abs r2;
            r2=ashift r2 by 0xfffffff0;
fifo_abs:   dm(i1,m6)=r2;

    r0=gets(1);  i4=r0;
    r0=gets(2);  i1=r0;
    r2=gets(3);
    r0=gets(4);
    alter(4);
    exit;
.endseg;
```

4

Assembler hand coded (++)

```
.SEGMENT /pm    seg_pmco;
.global         _ASMFastAbs;
.extern         _rbuf;
.extern         _TsCnt;
_ASMFastAbs:
entry;
    puts=r0; puts=r2; puts r4;
    r0=i1;  puts=r0;
    r0=i4;  puts=r0;

    i4=0x440004;
    i1=_rbuf;
    r0=dm(_TsCnt);        /* TsCnt = TsCnt/2 */
    r2=dm(i4,m5);

    lcntr=r0, DO abs2 UNTIL LCE;
            r2 = ashift r2 by 0xfffffff0; r4=dm(i4,m5);
            r2 = abs r2;
            r4 = ashift r4 by 0xfffffff0; dm(i1,m6)=r2;
            r4 = abs r4; r2=dm(i4,m5);
abs2:   dm(i1,m6)=r4;

    r0=gets(1);  i4=r0;
    r0=gets(2);  i1=r0;
    r4=gets(3);
    r2=gets(4);
    r0=gets(5);
    alter(5);
    exit;
.endseg;
```

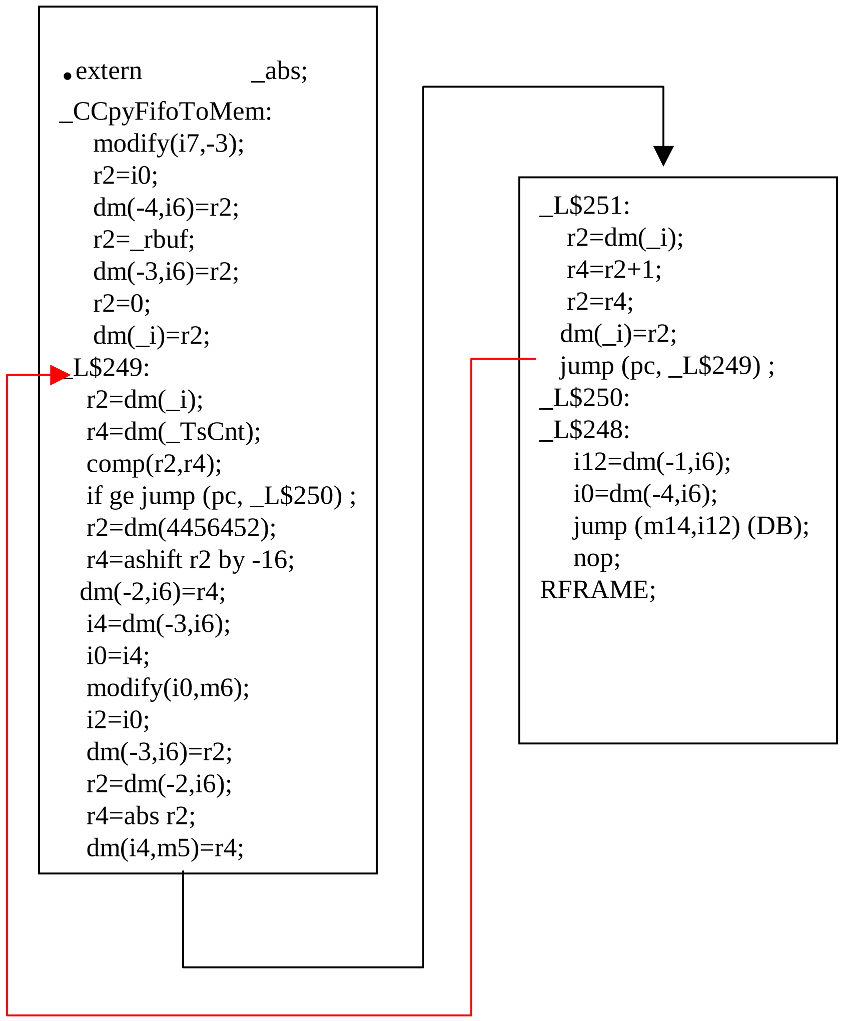
5

```
void CCpyFifoToMem()
{

    int value, *ip0;

    ip0=rbuf;

    for (i=0; i<TsCnt; i++)
    {
        value = mk_SRAM2(4) >> 16;
        *ip0++ = abs(value);
    }
}
```

C-Program

ASM

20

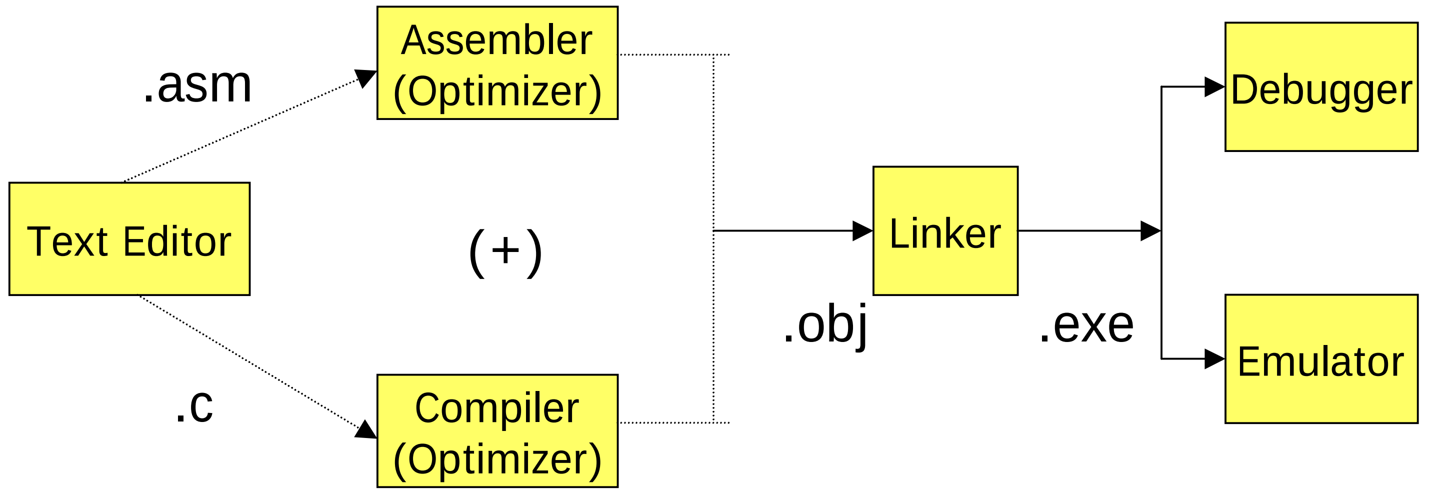


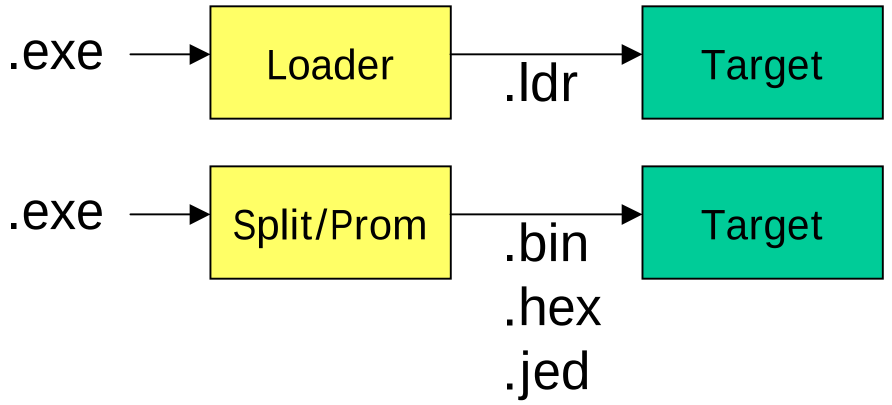
```
• extern              _abs;
_CCpyFifoToMem:
     modify(i7,-3);
     r2=i0;
     dm(-4,i6)=r2;
     r2=_rbuf;
     dm(-3,i6)=r2;
     r2=0;
     dm(_i)=r2;
_L$249:
    r2=dm(_i);
    r4=dm(_TsCnt);
    comp(r2,r4);
    if ge jump (pc, _L$250) ;
    r2=dm(4456452);
    r4=ashift r2 by -16;
   dm(-2,i6)=r4;
    i4=dm(-3,i6);
    i0=i4;
    modify(i0,m6);
    i2=i0;
    dm(-3,i6)=r2;
    r2=dm(-2,i6);
    r4=abs r2;
    dm(i4,m5)=r4;
```

```
_L$251:
    r2=dm(_i);
    r4=r2+1;
    r2=r4;
   dm(_i)=r2;
   jump (pc, _L$249) ;
_L$250:
_L$248:
     i12=dm(-1,i6);
     i0=dm(-4,i6);
     jump (m14,i12) (DB);
     nop;
RFRAME;
```

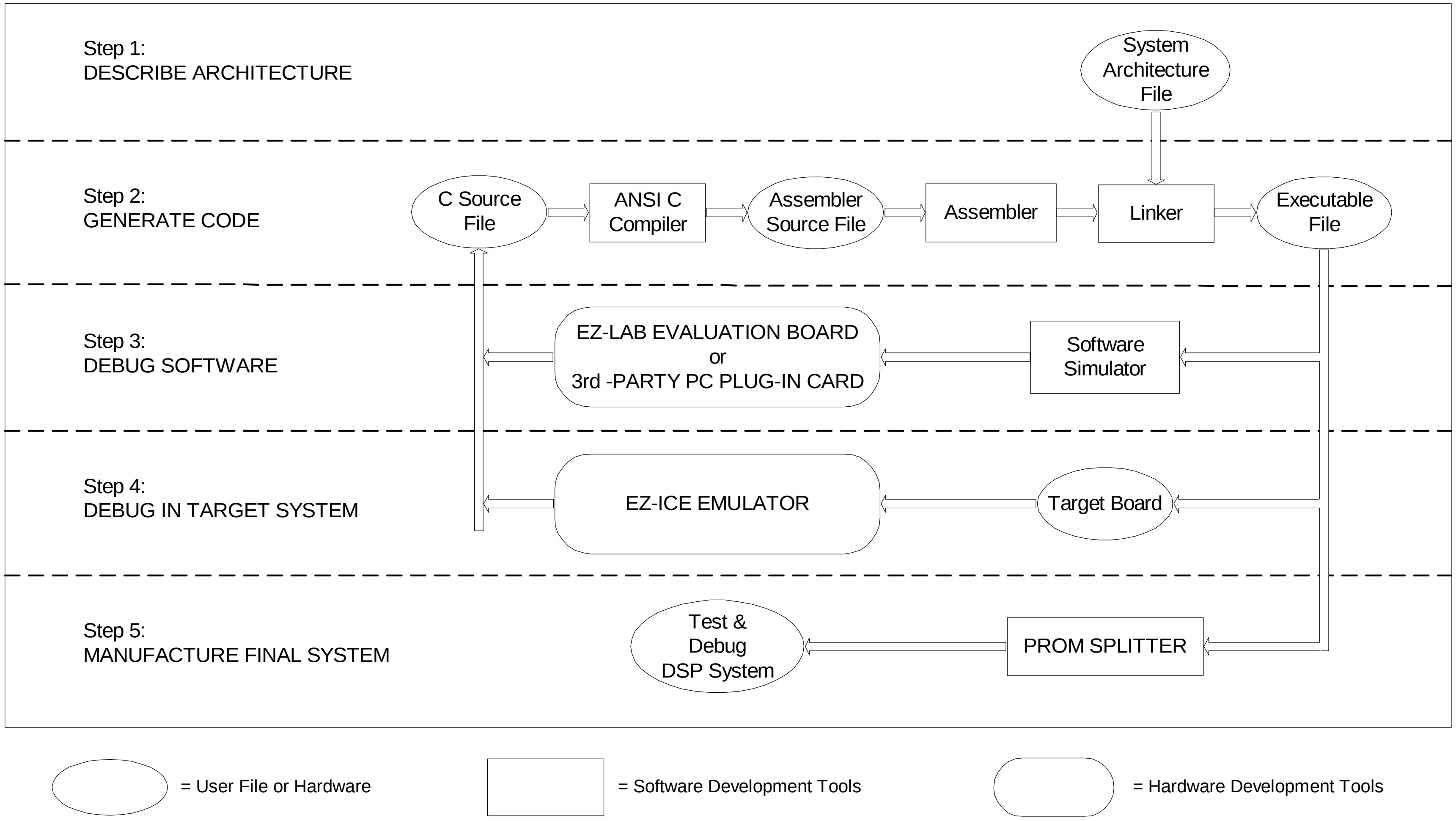
Embedded DSP: SHARC Software Tools

Method 1:

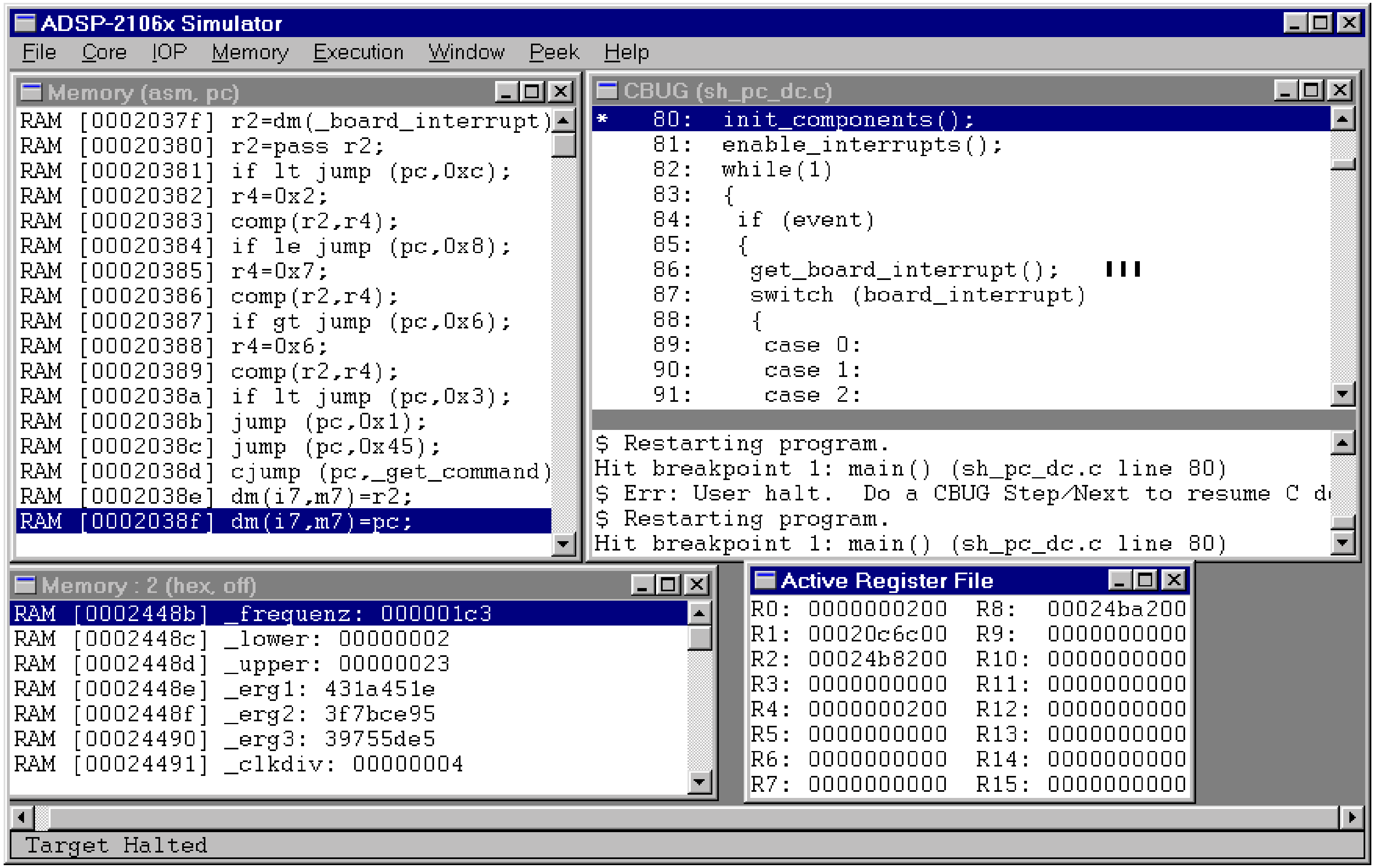




Development Environment with different modules during the 90th



SIMULATOR



```
RAM [0002037f] r2=dm(_board_interrupt)
RAM [00020380] r2=pass r2;
RAM [00020381] if lt jump (pc,0xc);
RAM [00020382] r4=0x2;
RAM [00020383] comp(r2,r4);
RAM [00020384] if le jump (pc,0x8);
RAM [00020385] r4=0x7;
RAM [00020386] comp(r2,r4);
RAM [00020387] if gt jump (pc,0x6);
RAM [00020388] r4=0x6;
RAM [00020389] comp(r2,r4);
RAM [0002038a] if lt jump (pc,0x3);
RAM [0002038b] jump (pc,0x1);
RAM [0002038c] jump (pc,0x45);
RAM [0002038d] cjump (pc,_get_command)
RAM [0002038e] dm(i7,m7)=r2;
RAM [0002038f] dm(i7,m7)=pc;
```

```
*  80:  init_components();
   81:  enable_interrupts();
   82:  while(1)
   83:  {
   84:   if (event)
   85:   {
   86:    get_board_interrupt();
   87:    switch (board_interrupt)
   88:    {
   89:     case 0:
   90:     case 1:
   91:     case 2:
```

```
$ Restarting program.
Hit breakpoint 1: main() (sh_pc_dc.c line 80)
$ Err: User halt.  Do a CBUG Step/Next to resume C d
$ Restarting program.
Hit breakpoint 1: main() (sh_pc_dc.c line 80)
```

```
RAM [0002448b] _frequenz: 000001c3
RAM [0002448c] _lower: 00000002
RAM [0002448d] _upper: 00000023
RAM [0002448e] _erg1: 431a451e
RAM [0002448f] _erg2: 3f7bce95
RAM [00024490] _erg3: 39755de5
RAM [00024491] _clkdiv: 00000004
```

```
R0: 0000000200  R8:  00024ba200
R1: 00020c6c00  R9:  0000000000
R2: 00024b8200  R10: 0000000000
R3: 0000000000  R11: 0000000000
R4: 0000000200  R12: 0000000000
R5: 0000000000  R13: 0000000000
R6: 0000000000  R14: 0000000000
R7: 0000000000  R15: 0000000000
```

Target Halted

Embedded DSP: Method2: SHARC-Visual DSP

Since 1999:

- Windows based software development environment for Analog Devices Digital Signal Processors:
  - An integrated development environment with support for editing programs, managing projects, and controlling build tools.
  - A source level, object oriented debugger with support for DSP simulation and emulation.
  - Context-sensitive help for the Windows-based development environment.
  - Online access to all documentation for the products via -pdf files.
  - Boot-Loader and generation of HEX and S files for EPROMs

Embedded DSP: Method2: SHARC-Visual DSP

- VisualDSP supports many file formats:
  - Source files

• Assembly Source Files	(*.ASM)
• Assembly Initialization Data Files	(*.DAT)
• Preprocessor Header Files	(*.H)
• Linker Description File	(*.LDF)
– ASCII text files that contain commands forthe linkers scripting language, which holds target information	
• Linker Command File	(*.TXT)

  - Build (Processed) Files

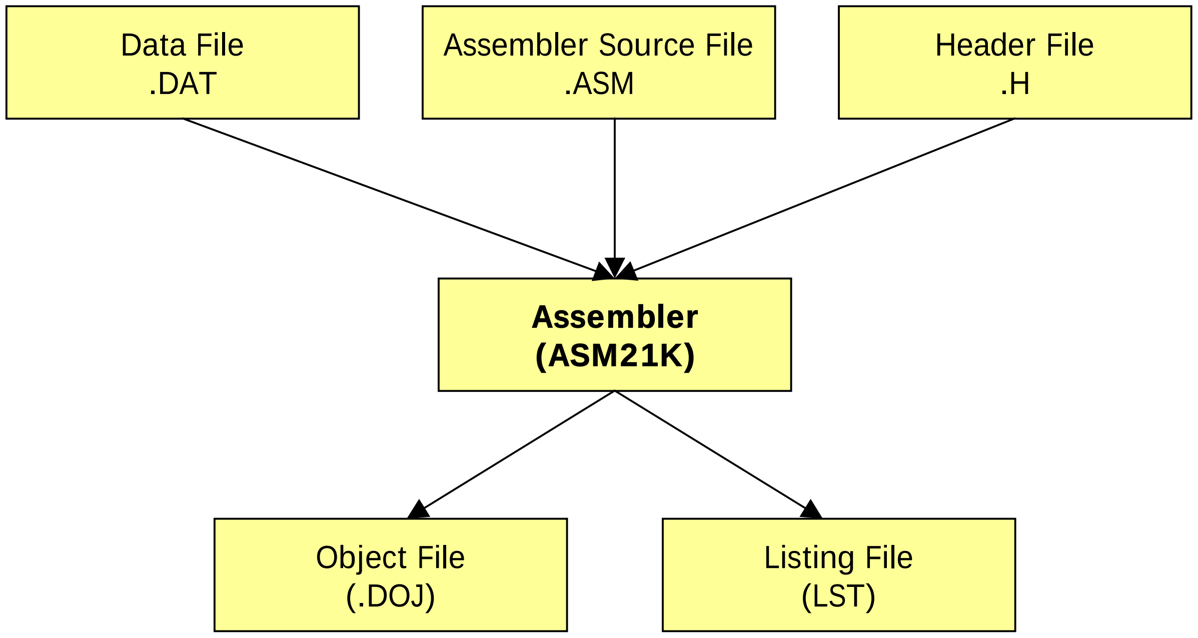
• Assembler Object Files	(*.DOJ)
• Archiver File	(*.DLB)
• Linker Executable File	(*.DXE, *.SM, *.OVL, *.DLO)
• Linker Memory Map	(*.MAP)
• Loader HEX-Files, ASCII-Files, Binary-Files	(*.LDR)
• Splitter Motorola S-Record Files	(*.S_#)
• Splitter Hex Format Files	(*.H_#)
• Splitter Byte-Stacked Format Files	(.STK)

Embedded DSP: Method2: SHARC-Visual DSP

- Debugger Files
  - Provide input to the debugger to define support for simulation or emulation of the program.
  - The debugger supports all the executable files types produced by the linker (.DXE, .SM, .OVL, .DLO).
  - To simulate I/O, the debugger supports the data file formats (.DAT) form the assembler, the loadable file formats from the loader (.LDR), and the PROM formats from the splitter (.S_#, .H_#, .STK)

Embedded DSP: SHARC Software Tools

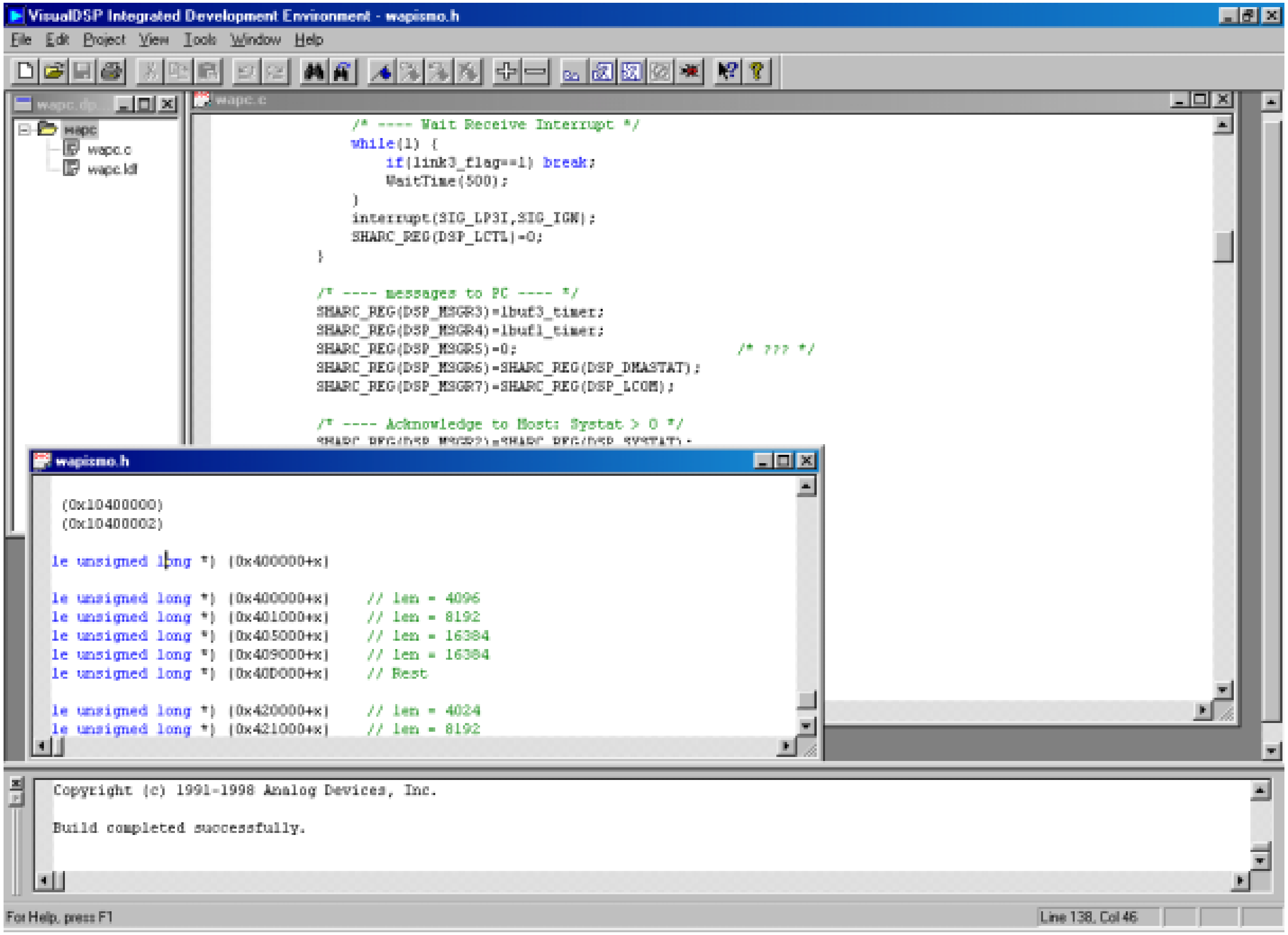
- Assembler Input & Output Files



Embedded DSP: SHARC Software Tools

- C-Compiler is realized by an additional Runtime Header Model
  - Preserves structures for register usage, runtime stack model and stack heap.
  - Need for a special memory model in the architecture file
  - Uses the following default segments:

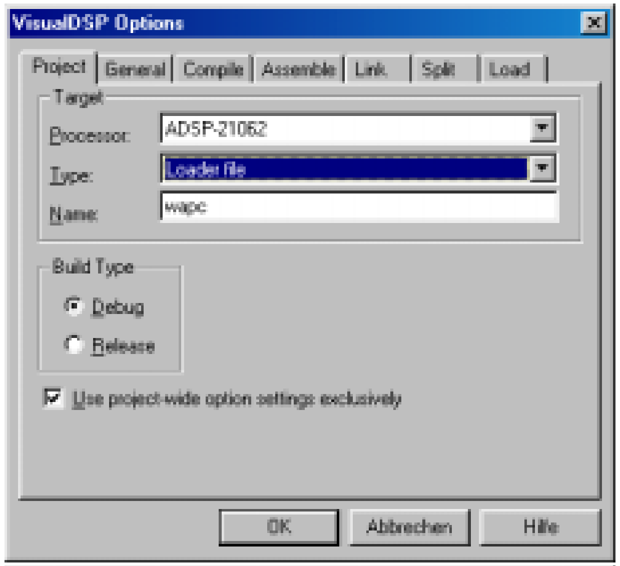
• seg_rth	(Interrupt table/runtime header)
• seg_pmco	(PM code)
• seg_pmda	(PM data)
• seg_dmda	(DM data)
• seg_heap	(heap space)
• seg_stack	(stack space)


VisualDSP Integrated Development Environment - wapismo.h
File Edit Project View Tools Window Help
wapc.dp...
wapc
wapc.c
wapc.ldf
wapc.c
wapismo.h
For Help, press F1
Line 138, Col 46

```
        /* ---- Wait Receive Interrupt */
        while(1) {
            if(link3_flag==1) break;
            WaitTime(500);
        }
        interrupt(SIG_LP3I,SIG_IGN);
        SHARC_REG(DSP_LCTL)=0;
    }

    /* ---- messages to PC ---- */
    SHARC_REG(DSP_MSGR3)=ibuf3_timer;
    SHARC_REG(DSP_MSGR4)=ibuf1_timer;
    SHARC_REG(DSP_MSGR5)=0;                              /* ??? */
    SHARC_REG(DSP_MSGR6)=SHARC_REG(DSP_DMASTAT);
    SHARC_REG(DSP_MSGR7)=SHARC_REG(DSP_LCOM);

    /* ---- Acknowledge to Host: Systat > 0 */
```

```
 (0x10400000)
 (0x10400002)

le unsigned long *) (0x400000+x)

le unsigned long *) (0x400000+x)    // len = 4096
le unsigned long *) (0x401000+x)    // len = 8192
le unsigned long *) (0x405000+x)    // len = 16384
le unsigned long *) (0x409000+x)    // len = 16384
le unsigned long *) (0x40D000+x)    // Rest

le unsigned long *) (0x420000+x)    // len = 4024
le unsigned long *) (0x421000+x)    // len = 8192
```

```
Copyright (c) 1991-1998 Analog Devices, Inc.

Build completed successfully.
```



VisualDSP Options

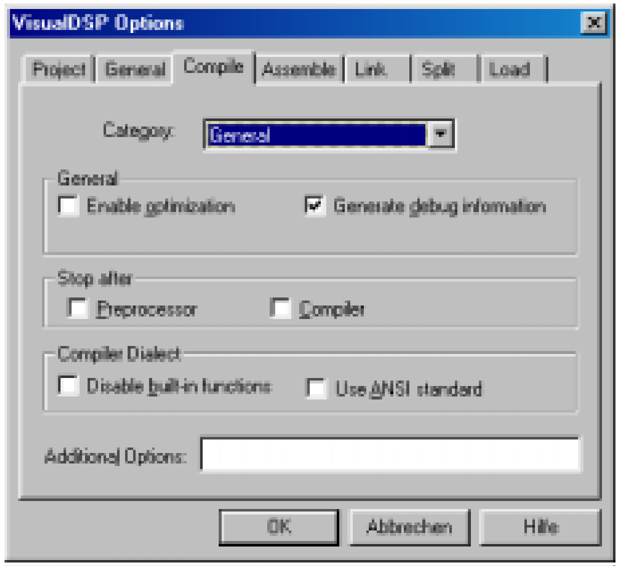
Project | General | Compile | Assemble | Link | Split | Load

Target
Processor: ADSP-21062
Type: Loader file
Name: wapc

Build Type
[✓ Debug]
Release

[✓] Use project-wide option settings exclusively

OK | Abbrechen | Hilfe



VisualDSP Options

Project | General | Compile | Assemble | Link | Split | Load

Category: General

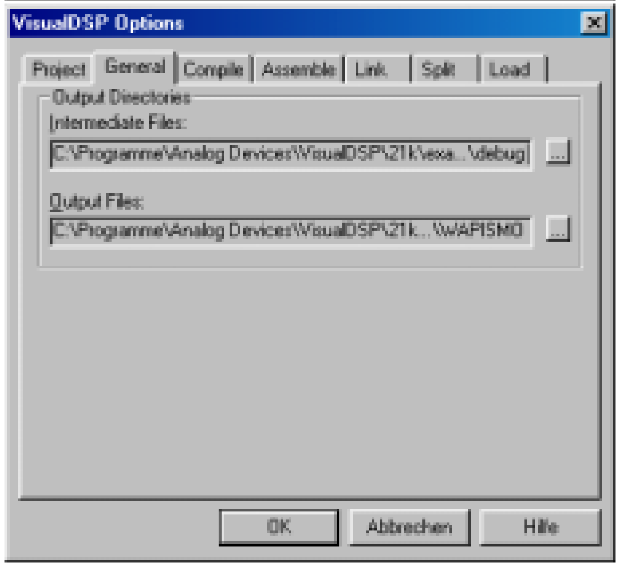
General
[ ] Enable optimization
[✓] Generate debug information

Stop after
[ ] Preprocessor
[ ] Compiler

Compiler Dialect
[ ] Disable built-in functions
[ ] Use ANSI standard

Additional Options:

OK | Abbrechen | Hilfe

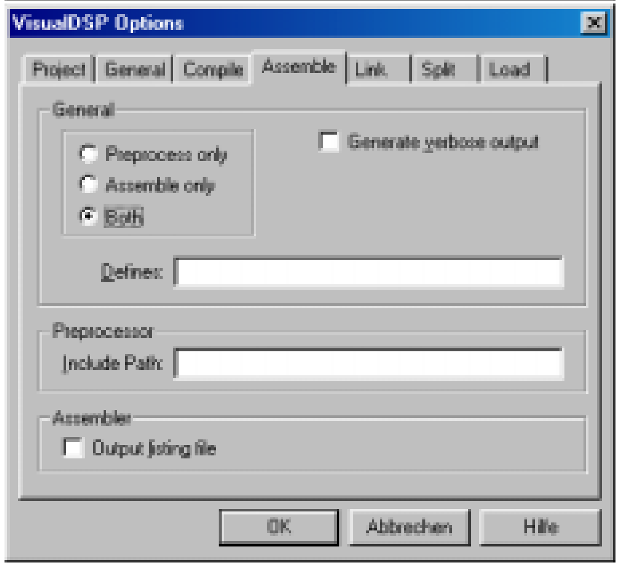


VisualDSP Options

Project | General | Compile | Assemble | Link | Split | Load

Output Directories
Intermediate Files:
C:\Programme\Analog Devices\VisualDSP\21k\exa...\debug
Output Files:
C:\Programme\Analog Devices\VisualDSP\21k...\WAP15M0

OK | Abbrechen | Hilfe



VisualDSP Options

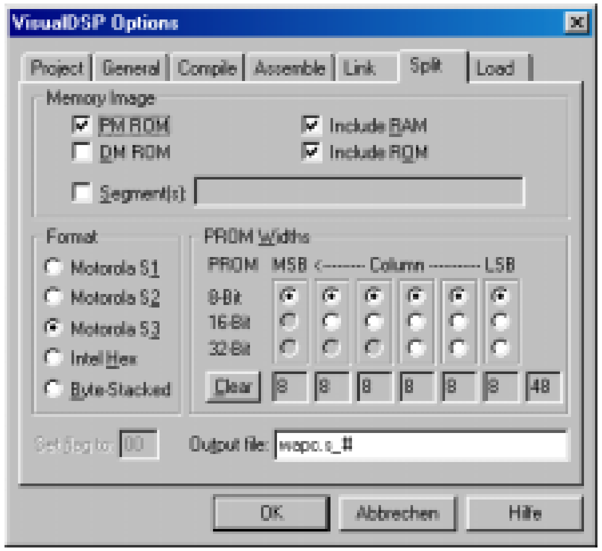
Project | General | Compile | Assemble | Link | Split | Load

General
Preprocess only
Assemble only
[✓ Both]
[ ] Generate verbose output
Defines:

Preprocessor
Include Path:

Assembler
[ ] Output listing file

OK | Abbrechen | Hilfe

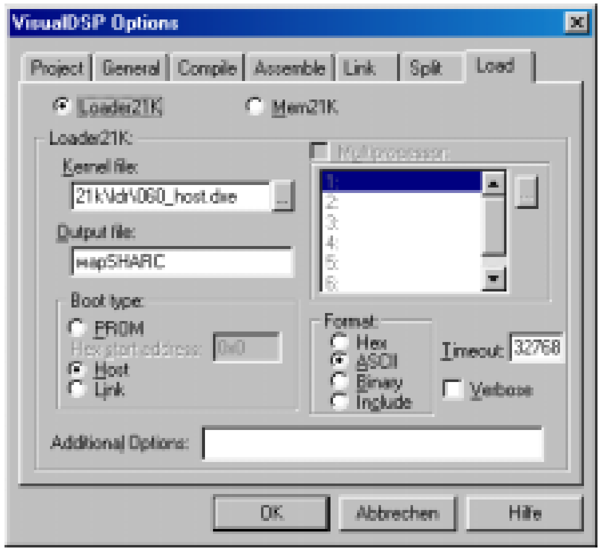

VisualDSP Options — Split

Memory Image: [✓ PM ROM] DM ROM, [✓ Include RAM] [✓ Include ROM], Segment(s):

Format: Motorola S1, Motorola S2, [✓ Motorola S3], Intel Hex, Byte-Stacked

PROM Widths: PROM MSB <------- Column --------- LSB; [✓ 8-Bit] in all columns; 16-Bit; 32-Bit; 8 8 8 8 8 8 48

Set flag to: 00 Output file: wapo.s_#


VisualDSP Options — Load

[✓ Loader21K] Mem21K

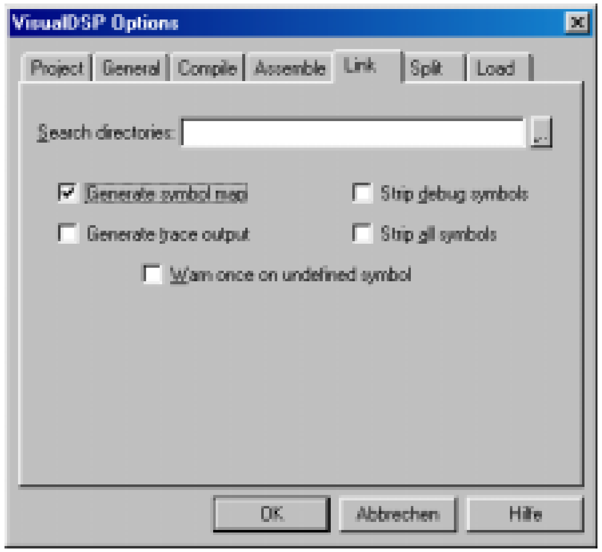
Loader21K:
Kernel file: 21k\ldr\060_host.dxe
Output file: wapSHARC
Multiprocessor: 1: 2: 3: 4: 5: 6:

Boot type: PROM, Hex start address: 0x0, [✓ Host], Link

Format: Hex, [✓ ASCII], Binary, Include

Timeout: 32768
Verbose

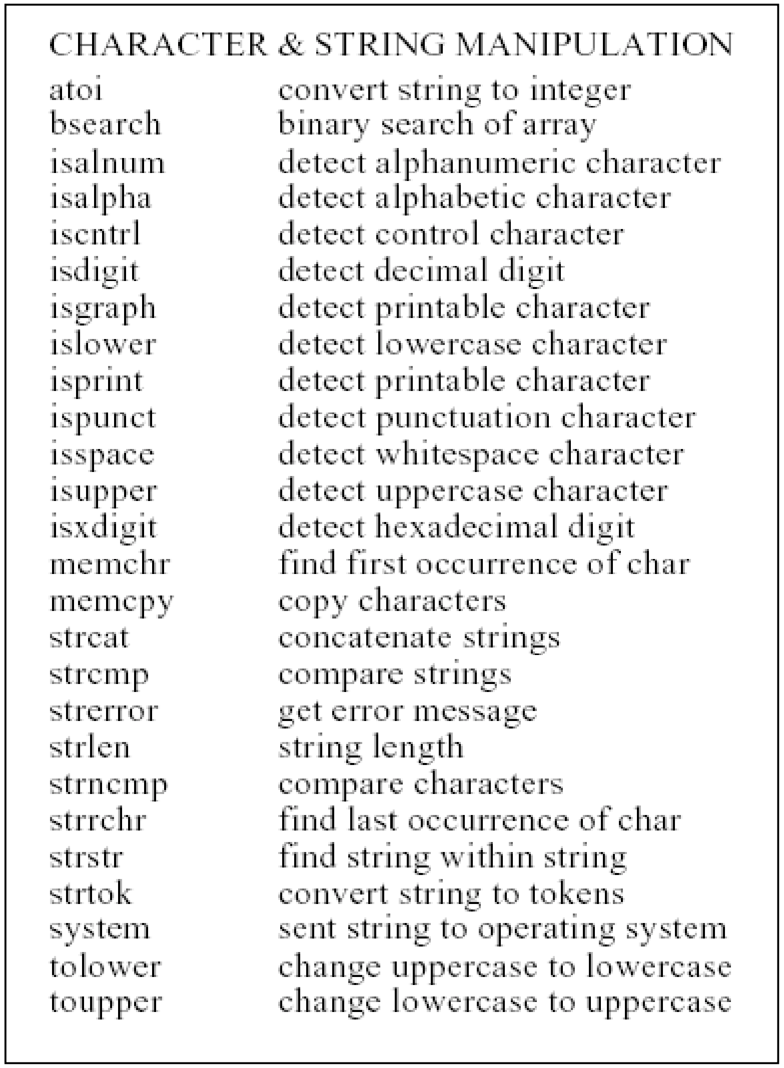
Additional Options:


VisualDSP Options — Link

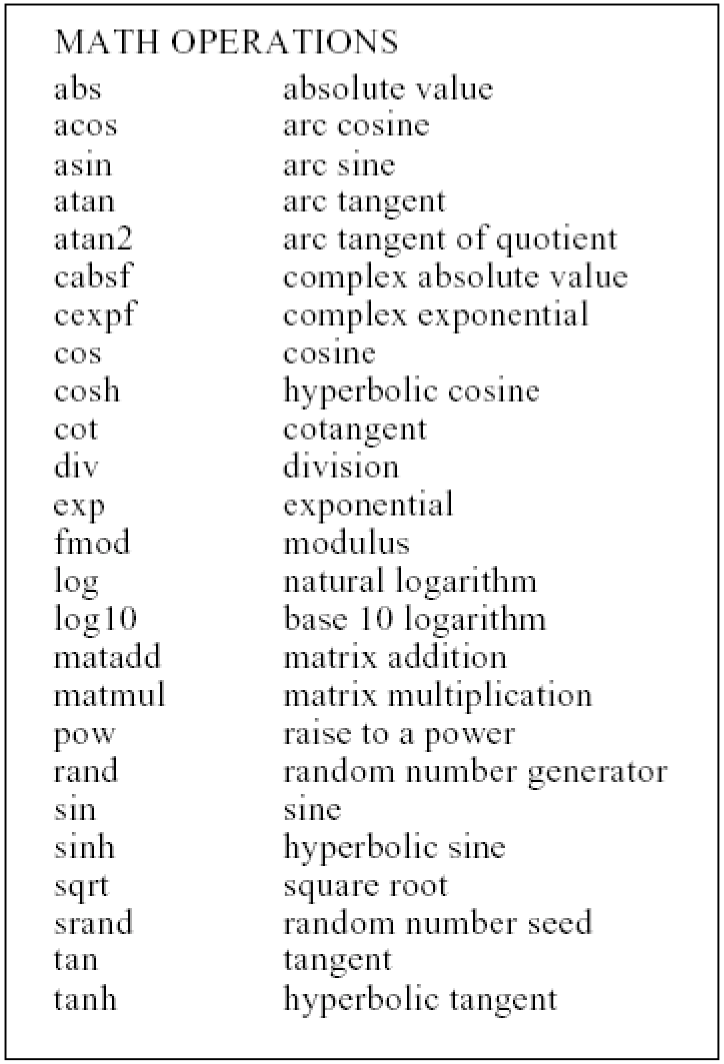
Search directories:

[✓ Generate symbol map] Strip debug symbols
Generate trace output Strip all symbols
Warn once on undefined symbol

Standard Functions

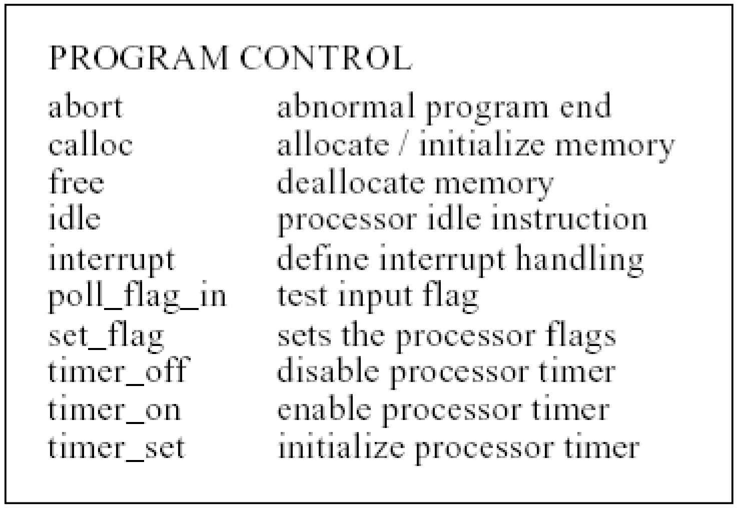


CHARACTER & STRING MANIPULATION	
atoi	convert string to integer
bsearch	binary search of array
isalnum	detect alphanumeric character
isalpha	detect alphabetic character
iscntrl	detect control character
isdigit	detect decimal digit
isgraph	detect printable character
islower	detect lowercase character
isprint	detect printable character
ispunct	detect punctuation character
isspace	detect whitespace character
isupper	detect uppercase character
isxdigit	detect hexadecimal digit
memchr	find first occurrence of char
memcpy	copy characters
strcat	concatenate strings
strcmp	compare strings
strerror	get error message
strlen	string length
strncmp	compare characters
strrchr	find last occurrence of char
strstr	find string within string
strtok	convert string to tokens
system	sent string to operating system
tolower	change uppercase to lowercase
toupper	change lowercase to uppercase

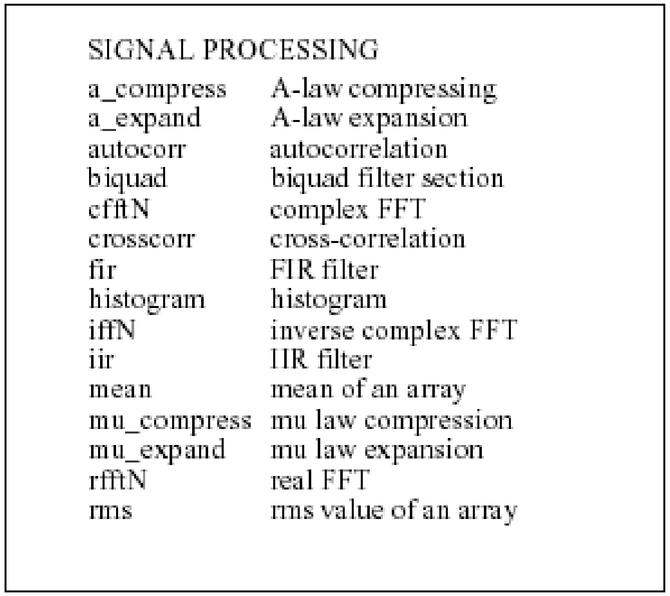


MATH OPERATIONS	
abs	absolute value
acos	arc cosine
asin	arc sine
atan	arc tangent
atan2	arc tangent of quotient
cabsf	complex absolute value
cexpf	complex exponential
cos	cosine
cosh	hyperbolic cosine
cot	cotangent
div	division
exp	exponential
fmod	modulus
log	natural logarithm
log10	base 10 logarithm
matadd	matrix addition
matmul	matrix multiplication
pow	raise to a power
rand	random number generator
sin	sine
sinh	hyperbolic sine
sqrt	square root
srand	random number seed
tan	tangent
tanh	hyperbolic tangent

Standard Functions

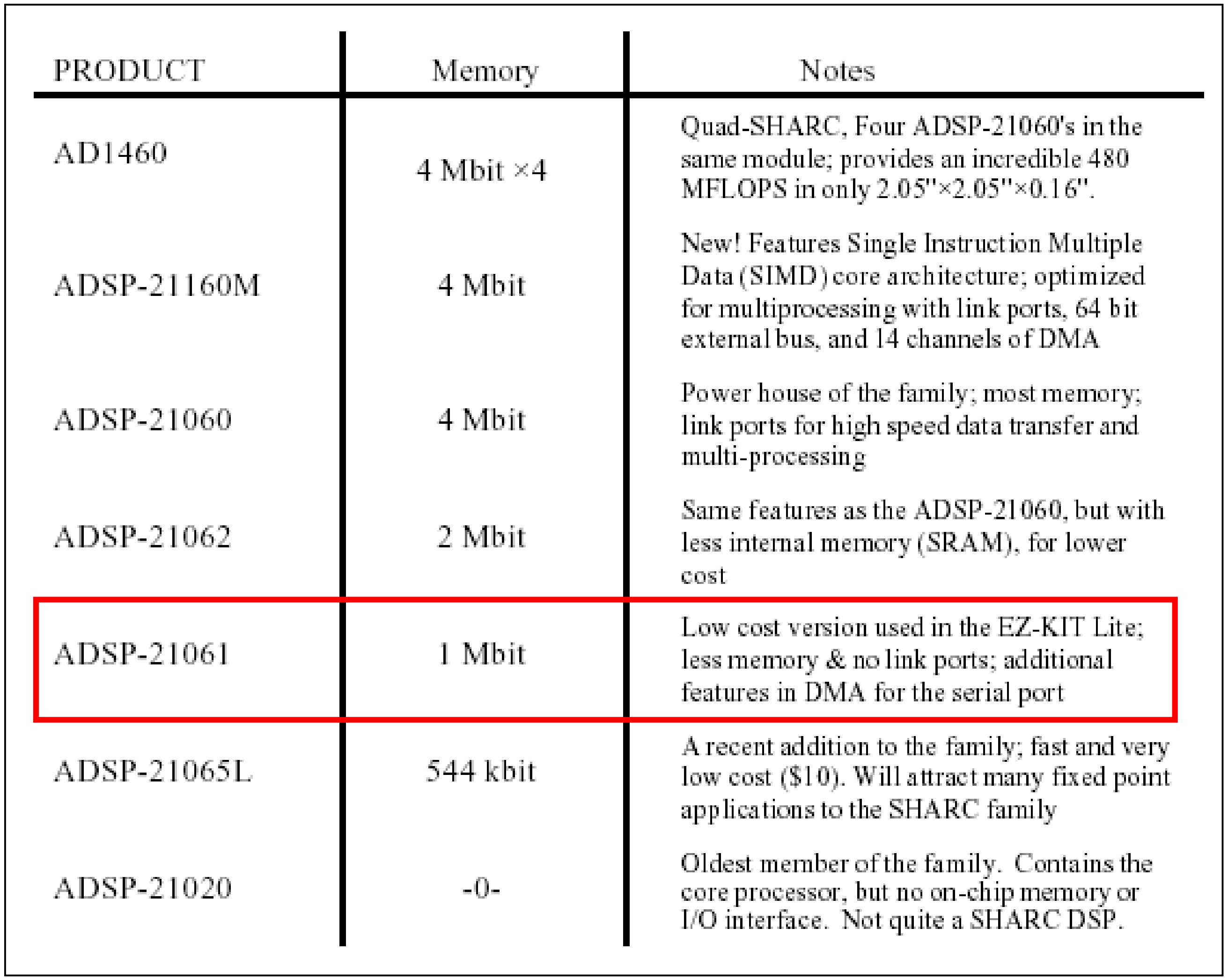


PROGRAM CONTROL	
abort	abnormal program end
calloc	allocate / initialize memory
free	deallocate memory
idle	processor idle instruction
interrupt	define interrupt handling
poll_flag_in	test input flag
set_flag	sets the processor flags
timer_off	disable processor timer
timer_on	enable processor timer
timer_set	initialize processor timer



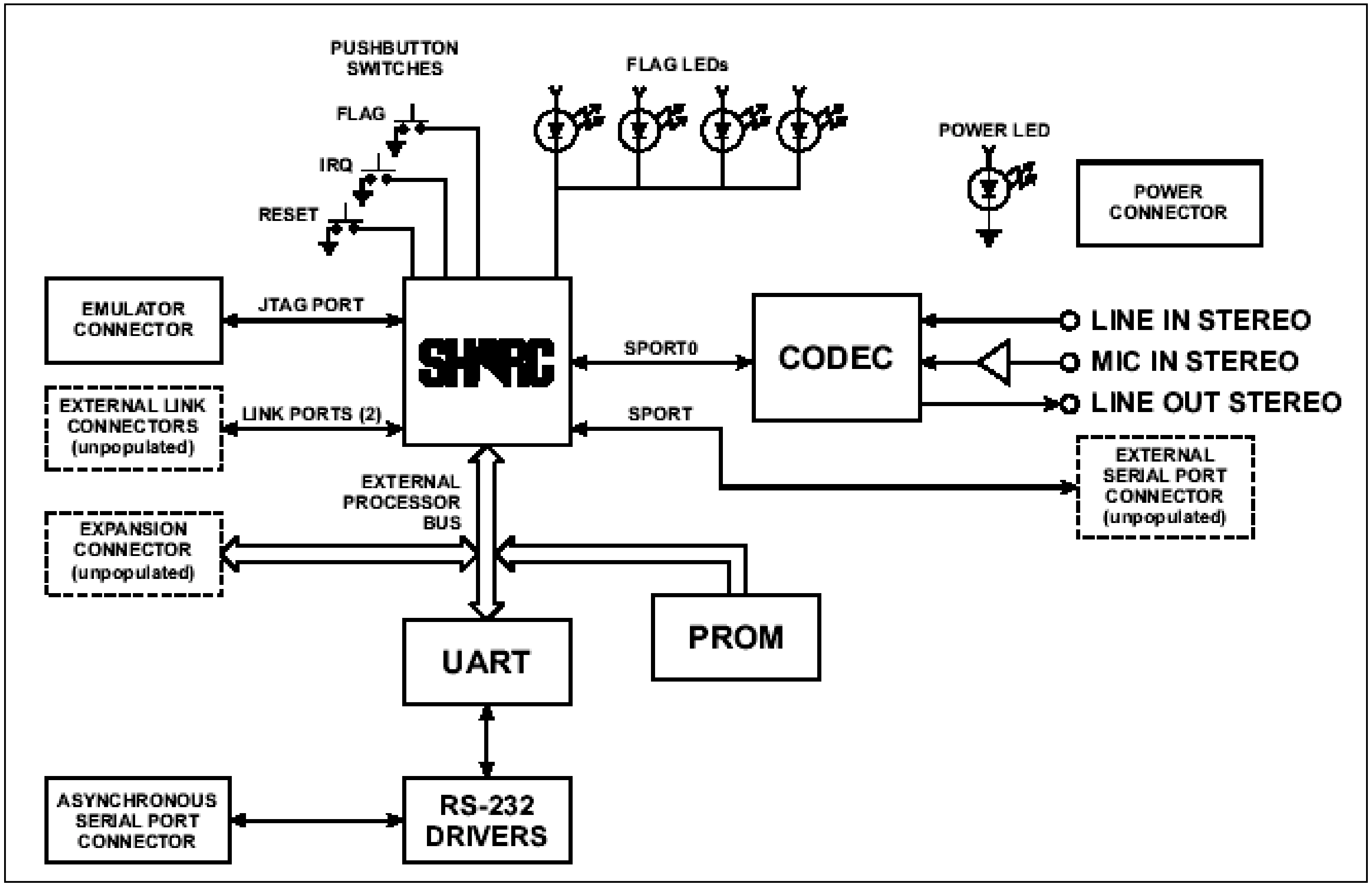
SIGNAL PROCESSING	
a_compress	A-law compressing
a_expand	A-law expansion
autocorr	autocorrelation
biquad	biquad filter section
cfftN	complex FFT
crosscorr	cross-correlation
fir	FIR filter
histogram	histogram
iffN	inverse complex FFT
iir	IIR filter
mean	mean of an array
mu_compress	mu law compression
mu_expand	mu law expansion
rfftN	real FFT
rms	rms value of an array

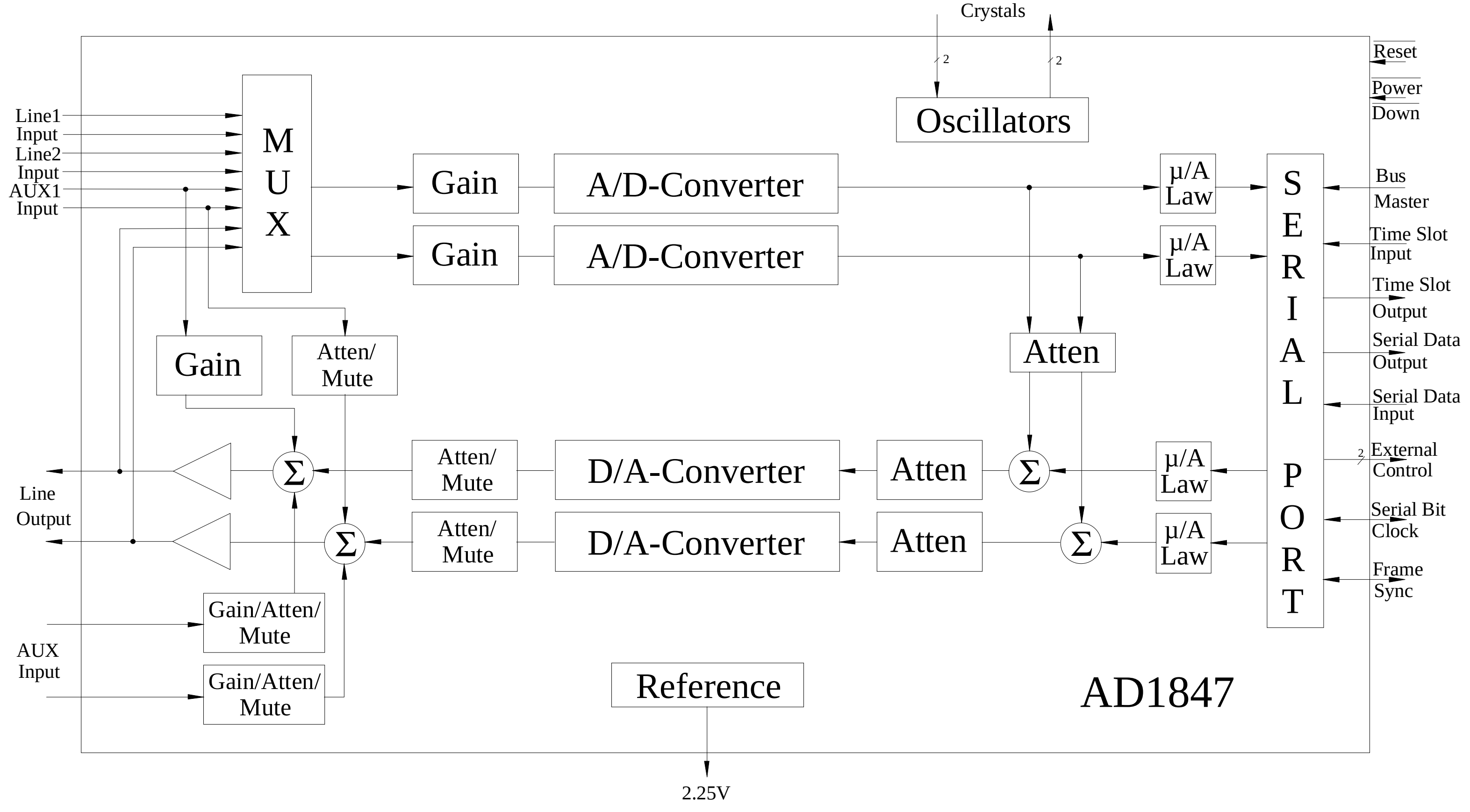
EZ-KIT



PRODUCT	Memory	Notes
AD1460	4 Mbit ×4	Quad-SHARC, Four ADSP-21060's in the same module; provides an incredible 480 MFLOPS in only 2.05"×2.05"×0.16".
ADSP-21160M	4 Mbit	New! Features Single Instruction Multiple Data (SIMD) core architecture; optimized for multiprocessing with link ports, 64 bit external bus, and 14 channels of DMA
ADSP-21060	4 Mbit	Power house of the family; most memory; link ports for high speed data transfer and multi-processing
ADSP-21062	2 Mbit	Same features as the ADSP-21060, but with less internal memory (SRAM), for lower cost
ADSP-21061	1 Mbit	Low cost version used in the EZ-KIT Lite; less memory & no link ports; additional features in DMA for the serial port
ADSP-21065L	544 kbit	A recent addition to the family; fast and very low cost ($10). Will attract many fixed point applications to the SHARC family
ADSP-21020	-0-	Oldest member of the family. Contains the core processor, but no on-chip memory or I/O interface. Not quite a SHARC DSP.

EZ-KIT


PUSHBUTTON SWITCHES
FLAG
IRQ
RESET
FLAG LEDs
POWER LED
POWER CONNECTOR
EMULATOR CONNECTOR
JTAG PORT
SHARC
SPORT0
CODEC
LINE IN STEREO
MIC IN STEREO
LINE OUT STEREO
EXTERNAL LINK CONNECTORS (unpopulated)
LINK PORTS (2)
SPORT
EXTERNAL SERIAL PORT CONNECTOR (unpopulated)
EXTERNAL PROCESSOR BUS
EXPANSION CONNECTOR (unpopulated)
UART
PROM
ASYNCHRONOUS SERIAL PORT CONNECTOR
RS-232 DRIVERS



Crystals
2
2
Oscillators
Reset
Power Down
Line1 Input
Line2 Input
AUX1 Input
MUX
Gain
A/D-Converter
Gain
A/D-Converter
µ/A Law
µ/A Law
SERIAL PORT
Bus Master
Time Slot Input
Time Slot Output
Serial Data Output
Serial Data Input
2
External Control
Serial Bit Clock
Frame Sync
Gain
Atten/ Mute
Atten
Line Output
Σ
Atten/ Mute
D/A-Converter
Atten
Σ
µ/A Law
Σ
Atten/ Mute
D/A-Converter
Atten
Σ
µ/A Law
Gain/Atten/ Mute
AUX Input
Gain/Atten/ Mute
Reference
AD1847
2.25V

Start of the Embedded DSP Mini Project
next monday at IZFP.