Eng. Julian S. Bruno

Real Time Digital Signal Processing

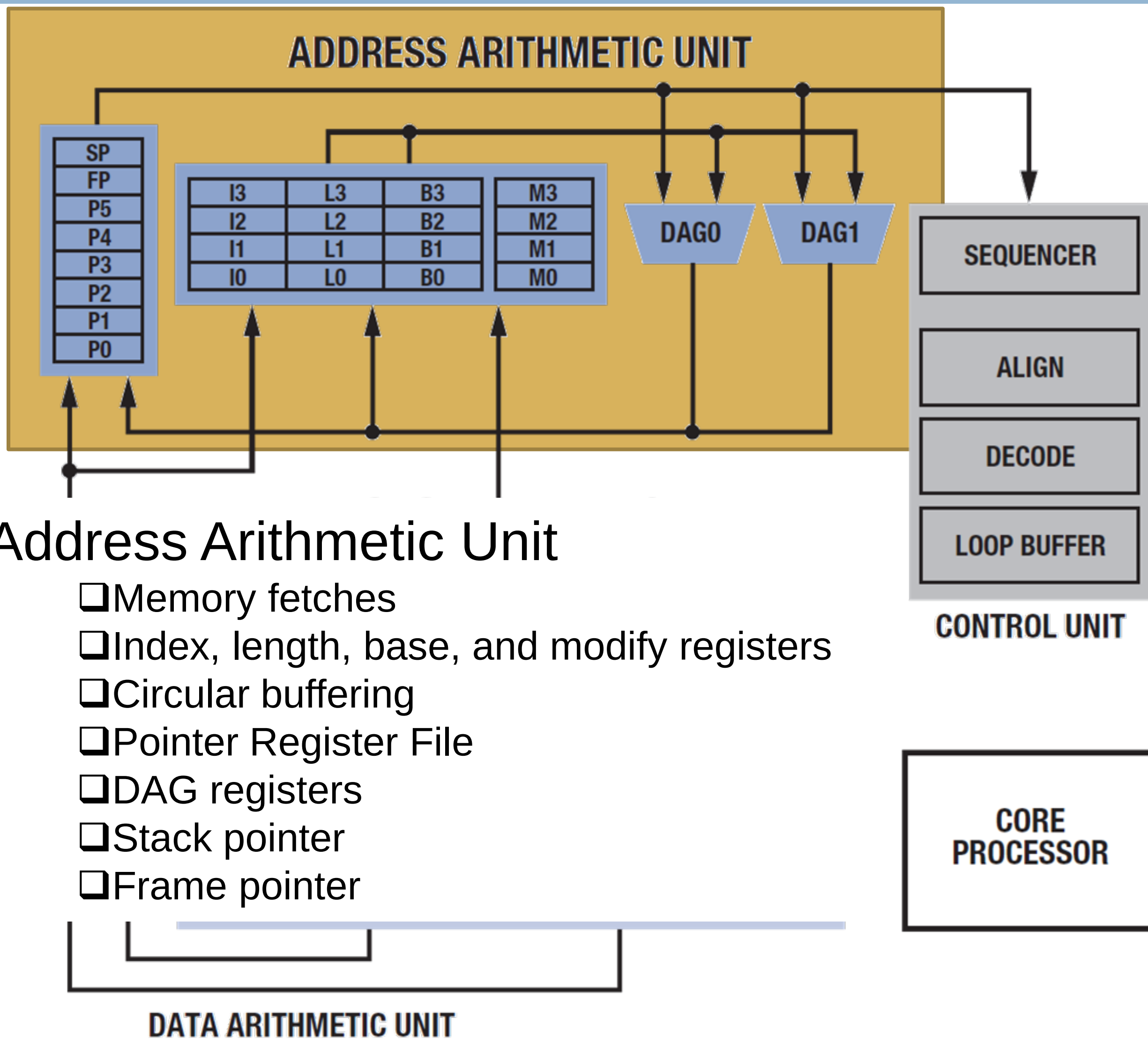
UTN-FRBA
2010

Eng. Julian Bruno

Architecture

Introduction to the Blackfin Processor

Address Arithmetic Unit – BF53X



Address Arithmetic Unit

- ❑Memory fetches
- ❑Index, length, base, and modify registers
- ❑Circular buffering
- ❑Pointer Register File
- ❑DAG registers
- ❑Stack pointer
- ❑Frame pointer

AAU - Functions

- Supply address

  Provides an address during a data access.

- Supply address and post-modify

  Provides an address during a data move and auto-increments/decrements the stored address for the next move.

- Supply address with offset

  Provides an address from a base with an offset without incrementing the original address pointer.

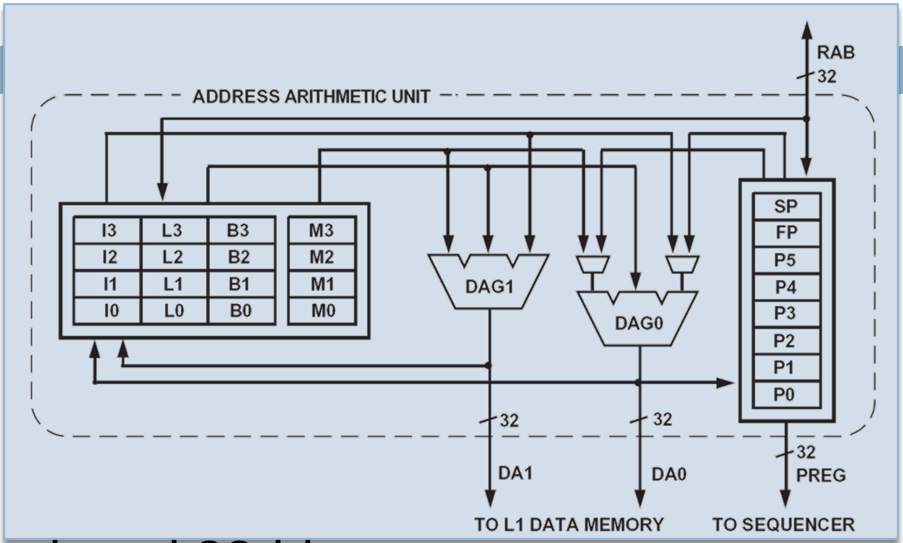
- Modify address

  Increments or decrements the stored address without performing a data move.

- Bit-reversed carry address

  Provides a bit-reversed carry address during a data move without reversing the stored address.

AAU – Description



- 2 DAGs
- 9 Pointer registers

  P[5:0], FP, USP, and SP.
- 4 Index registers

  I[3:0]: Contain index addresses. Unsigned 32-bit.
- 4 complete sets of related Modify, Base, and Length registers.

  M[3:0]: Contain modify values. Signed 32-bit

  B[3:0]: Contain base addresses. Unsigned 32-bit

  L[3:0]: Contain length values. Unsigned 32-bit

Addressing With the AAU

- The processor is byte addressed.
- All data accesses must be aligned to the data size.

```
#pragma align
#pragma alignment_region
#pragma alignment_region_end
```

- Depending on the type of data used, increments and decrements to the address registers can be by 1, 2, or 4 to match the 8-, 16-, or 32-bit accesses.

```
R0 = [P3++]; //It fetches a 32-bit word, P3+=4
R0.L = W[I3++]; //It fetches a 16-bit word, I3+=2
R0 = B[P3++](Z); //It fetches an 8-bit word, P3+=1
```


Eng. Julian S. Bruno

DAG Register Set

- I[3:0] M[3:0] B[3:0] L[3:0]
- The ***I (Index)*** registers and ***B (Base)*** registers always contain addresses of 8-bit bytes in memory.
- The ***M (Modify)*** registers contain an offset value that is added to one of the Index registers or subtracted from it.
- The B and ***L (Length)*** registers define circular buffers.
- Each L and B register pair is associated with the corresponding I register.
- Any M register may be associated with any I register.

Eng. Julian S. Bruno

Pointer Register File

- Frame Pointer (FP) used to point to the current procedure's activation record.
- Stack Pointer (SP) used to point to the last used location on the runtime stack.
- Some load/store instructions use FP and SP implicitly:
  - FP-indexed load/store, which extends the addressing range for16-bit encoded load/stores
  - Stack push/pop instructions, including those for pushing and popping multiple registers
  - Link/unlink instructions, which control stack frame space and manage the FP register for that space

Pointer Register File

- P-register file P[5:0]
  - 32 bits wide.
  - P-registers are primarily used for address calculations.
  - They may also be used for general integer arithmetic with a limited set of arithmetic operations.
  - To maintain counters.
  - However, P-register arithmetic does not affect the Arithmetic Status (ASTAT) register status flags.

Addressing Modes

- Indirect Addressing

```
R0 = [ I2 ] ;              // 32 bits
R0.H = W [ I2 ] ;          // 16 bits
[ P1 ] = R0 ;              // 32 bits
B [ P1 ] = R0 ;            // 8 bits
R0 = W[P1] (Z) ;           // 16 bits Zero Extension
R1 = W[P1] (X) ;           // 16 bits Sign Extension
```

- Indexed Addressing

```
R0 = [P1 + 0x11]
```

- Auto-increment and Auto-decrement Addressing

```
R0 = W [ P1++ ] (Z) ;      //the pointer is then incremented by 2
R0 = [ I2-- ] ;            //decrements the Index register by 4
```

- Post-modify Addressing

```
R2 = W [ P4++P5 ] (Z) ;
R2 = [ I2++M1 ] ;
```

Types of Transfers Supported and Transfer Sizes

Addressing Mode	Types of Transfers Supported	Transfer Sizes
Auto-increment Auto-decrement Indirect Indexed	To and from Data Registers	LOADS: 32-bit word 16-bit, zero extended half word 16-bit, sign extended half word 8-bit, zero extended byte 8-bit, sign extended byte STORES: 32-bit word 16-bit half word 8-bit byte
	To and from Pointer Registers	LOAD: 32-bit word STORE: 32-bit word
Post-increment	To and from Data Registers	LOADS: 32-bit word 16-bit half word to Data Register high half 16-bit half word to Data Register low half 16-bit, zero extended half word 16-bit, sign extended half word STORES: 32-bit word 16-bit half word from Data Register high half 16-bit half word from Data Register low half

Addressing Modes

	32-bit word	16-bit half-word	8-bit byte	Sign/zero extend	Data Register	Pointer register	Data Register Half
P Auto-inc [P0++]	*	*	*	*	*	*	
P Auto-dec [P0--]	*	*	*	*	*	*	
P Indirect [P0]	*	*	*	*	*	*	*
P Indexed [P0+im]	*	*	*	*	*	*	
FP indexed [FP+im]	*				*	*	
P Post-inc [P0++P1]	*	*		*	*		*
I Auto-inc [I0++]	*	*			*		*
I Auto-dec [I0--]	*	*			*		*
I Indirect [I0]	*	*			*		*
I Post-inc [I0++M0]	*				*		

Addressing Circular Buffers

- The Length (L) register sets the size of the circular buffer.
- The starting address that the DAG wraps around is called the buffer's base address (B-register)
- There are no restrictions on the value of the base address for circular buffers that contains 8-bit data. Circular buffers that contain 16- and 32-bit data must be 16-bit aligned and 32-bit aligned, respectively

Addressing Circular Buffers

P0.l = lo(buffer);

P0.h = hi(buffer);

P2 = length(buffer);

I0 = P0; B0 = P0; L0 = P2; M0 = 4;

R0 = 0;

nop;nop;nop;nop;

lsetup(Ls, Le) lc0 = p2;

Ls:

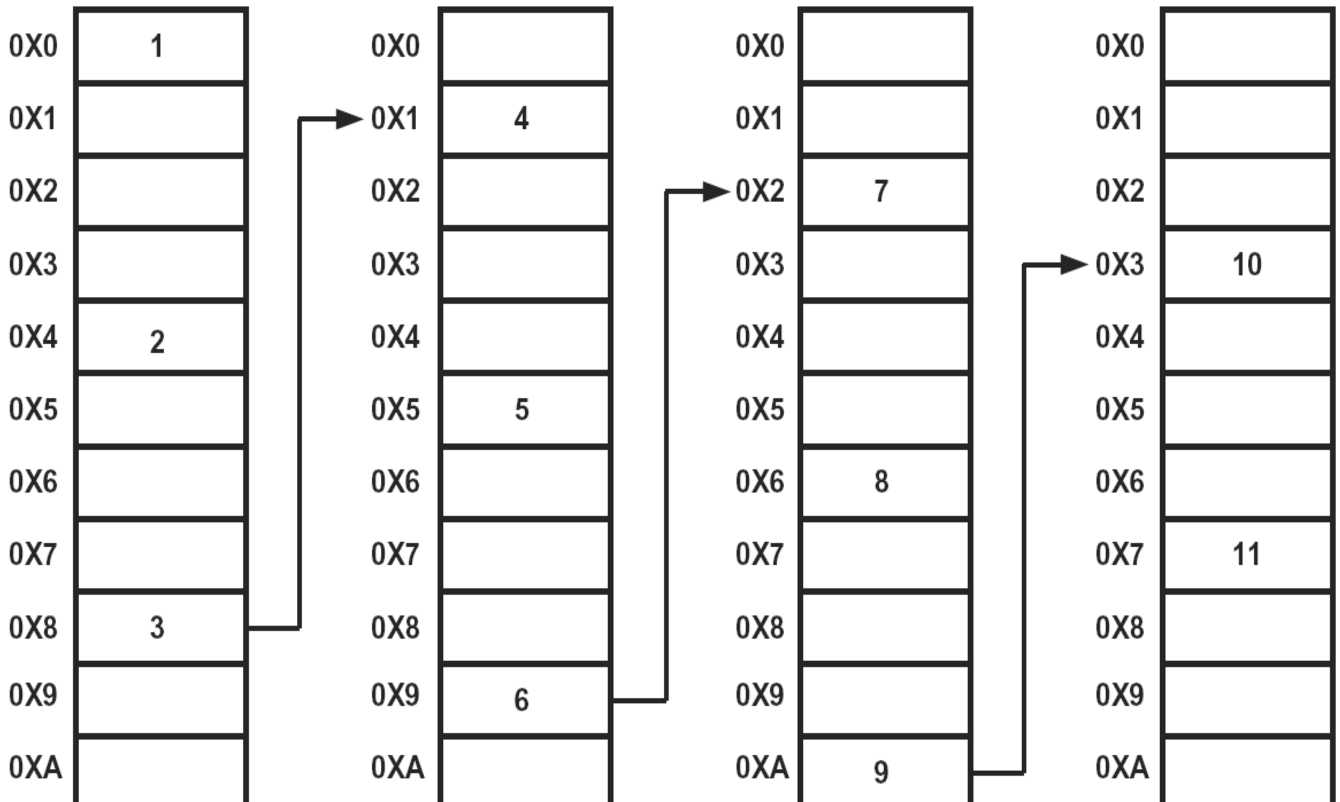
R0+=1;

P0 = I0;

B[P0] = R0;

Le:

I0+=M0;



Addressing With Bit-reversed Addresses

- To obtain results in sequential order, programs need bit-reversed carry addressing for some algorithms, particularly Fast Fourier Transform (FFT) calculations.

- To satisfy the requirements of these algorithms, the DAG's bit-reversed addressing feature permits repeatedly subdividing data sequences and storing this data in bit-reversed order.

Modifying DAG and Pointer Registers

- The DAGs support operations that modify an address value in an Index register without outputting an address.
- The operation, address-modify, is useful for maintaining pointers.
- The address-modify operation modifies addresses in any Index and Pointer register (I[3:0], P[5:0], FP, SP) without accessing memory.

```
I1 += M2 ;
```

Memory Address Alignment

- The processor requires proper memory alignment to be maintained for the data size being accessed.
- Alignment exceptions may be disabled by issuing the DISALGNEXCPT instruction in parallel with a load/store operation.
- 32-bit word load/stores are accessed on 4-byte boundaries, meaning the two least significant bits of the address are b#00.
- 16-bit word load/stores are accessed on 2-byte boundaries, meaning the least significant bit of the address must be b#0.

Bus Architecture and Memory

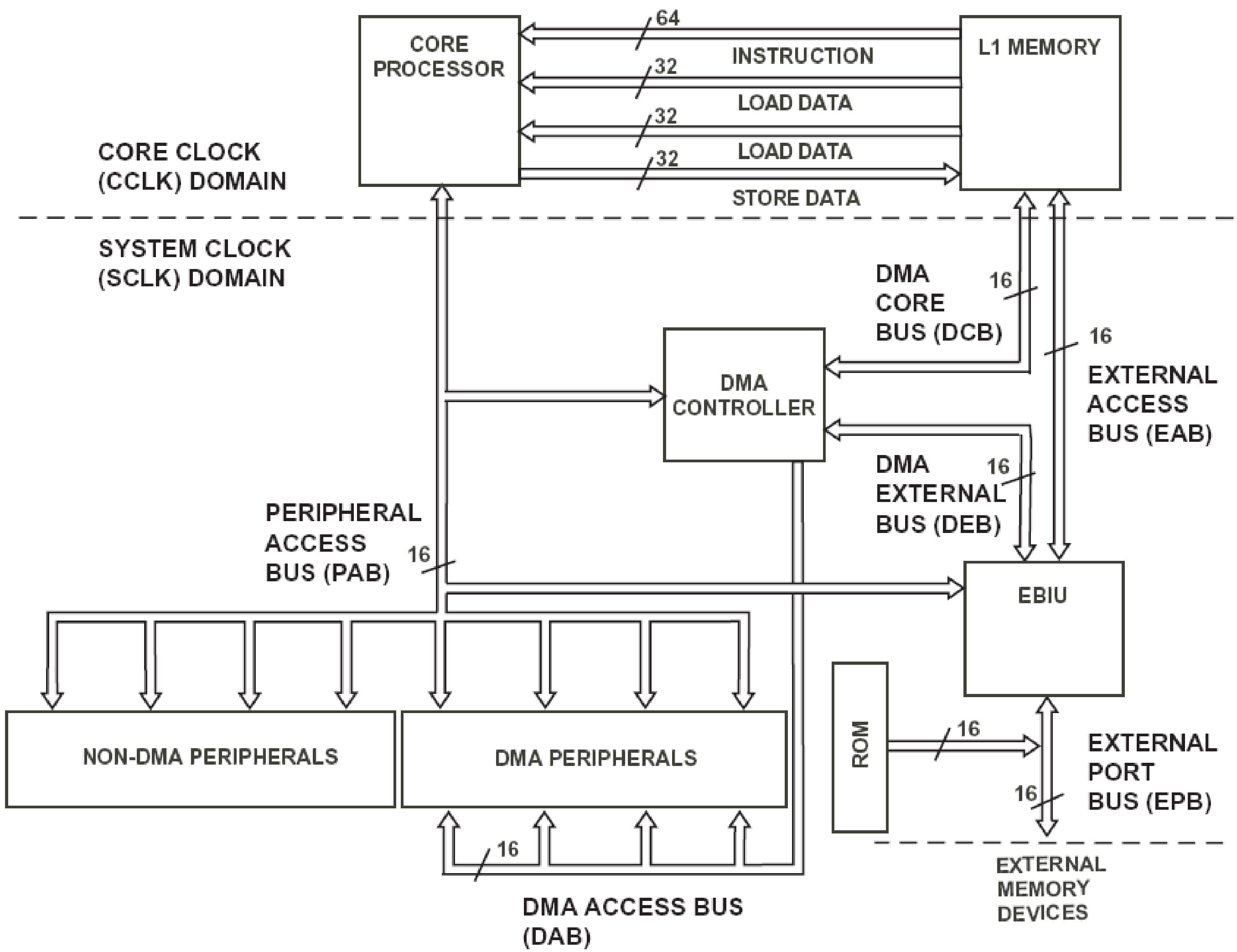
Introduction

- Blackfin processor uses a modified Harvard architecture.
- Blackfin processor has a single memory map that is shared between data and instruction memory.
- Blackfin processor supports a hierarchical memory model .
- The L1 data and instruction memory are located on the chip and are generally smaller in size but faster than the L2 external memory, which has a larger capacity.

Memory Architecture

- Blackfin processors have a unified 4G byte address range.
- The processor populates portions of this internal memory space with:
  - L1 Static Random Access Memories (SRAM)
    - Instruction / Data
    - SRAM / Cache.
      - SRAMs provide deterministic access time and very high throughput.
      - Cache provides both high performance and a simple programming model.
  - L2 Static Random Access Memories (SRAM)
  - A set of memory-mapped registers (MMRs)
  - A boot Read-Only Memory (ROM)

Processor Memory Architecture



Eng. Julian S. Bruno

On-Chip Level 1 (L1) Memory

- A modified Harvard architecture
  - one 64-bit instruction fetch
  - two 32-bit data loads
  - one pipelined 32-bit data store
- Simultaneous system DMA, cache maintenance, and core accesses.
- SRAM access at processor clock rate (CCLK) for critical DSP algorithms and fast context switching.
- Instruction and data cache options for microcontroller code, excellent High Level Language (HLL) support.
- Memory protection.

Scratchpad Data SRAM

- The processor provides a dedicated 4K byte bank of scratchpad data SRAM.
- The scratchpad is independent of the configuration of the other L1 memory banks and cannot be configured as cache or targeted by DMA.
- Typical applications use the scratchpad data memory where speed is critical.
- For example, the User and Supervisor stacks should be mapped to the scratchpad memory for the fastest context switching during interrupt handling.

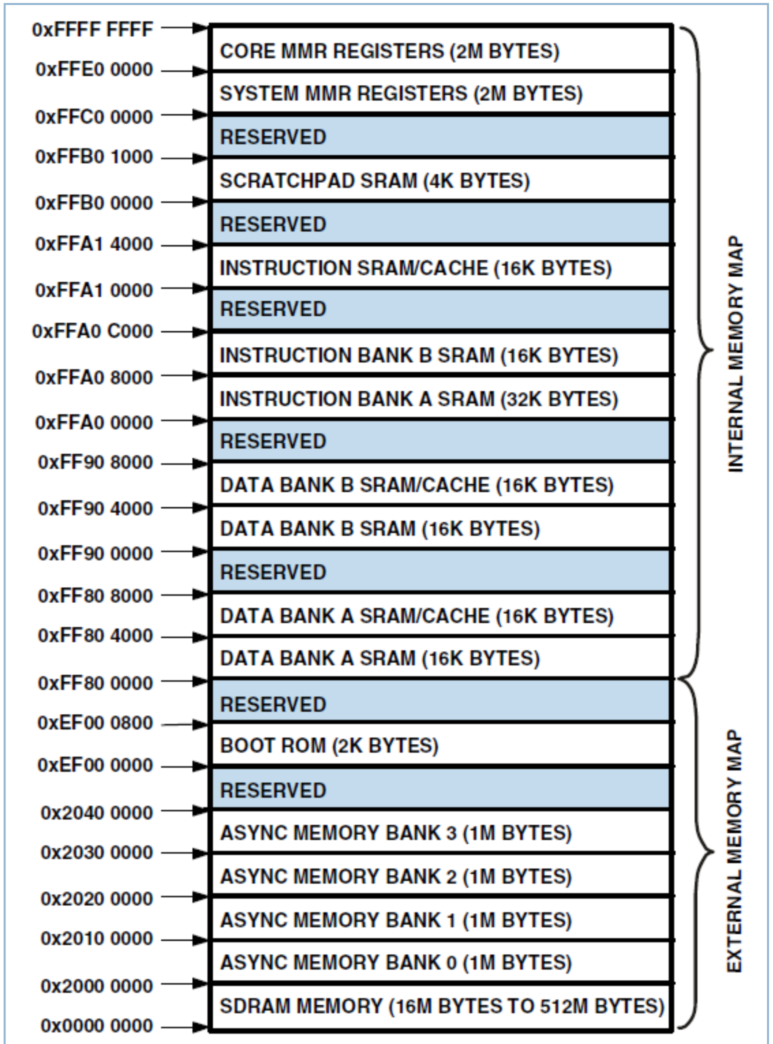
On-Chip Level 2 (L2) Memory

- Some Blackfin derivatives feature a Level 2 (L2) memory on chip.
- The L2 memory provides low latency, high-bandwidth capacity.
- On-chip L2 memory provides more capacity than L1 memory, but the latency is higher.
- The on-chip L2 memory is SRAM and can not be configured as cache.
- It is capable of storing both instructions and data. The L1 caches can be configured to cache instructions and data located in the on-chip L2 memory.

Boot ROM

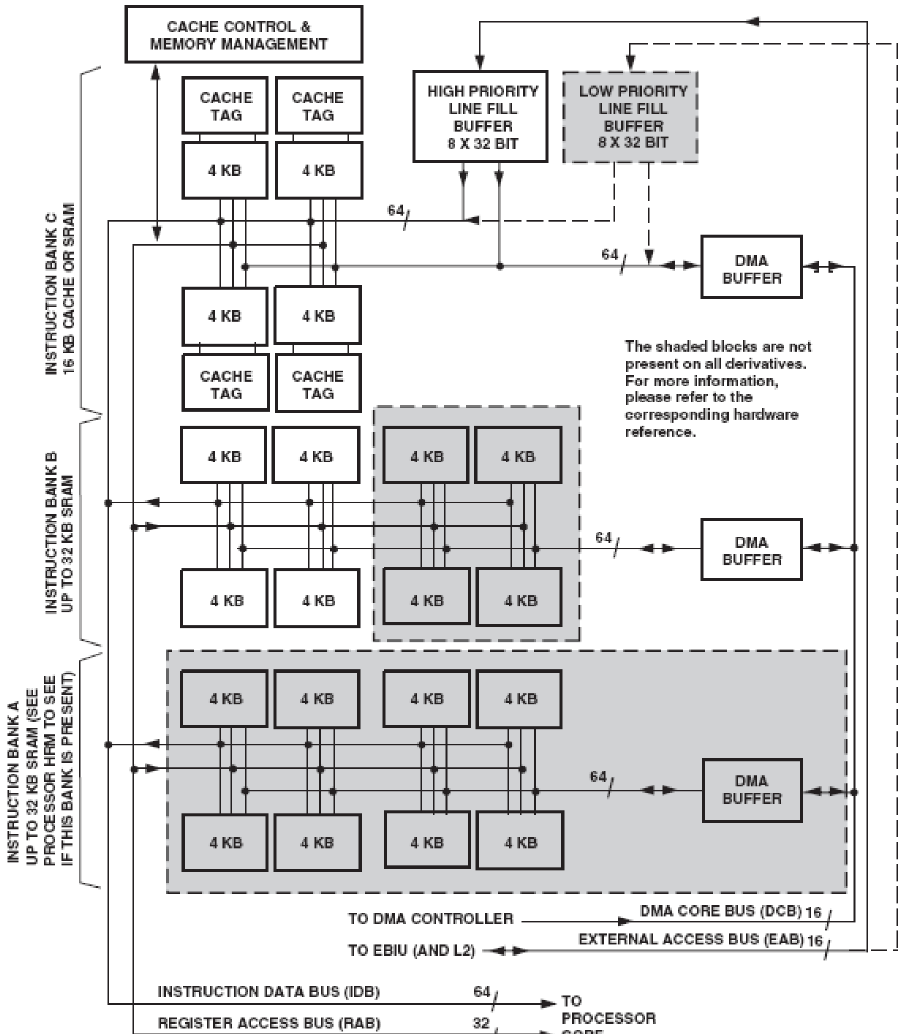
- This 16-bit boot ROM is not part of the L1 memory module.
- Read accesses take one SCLK cycle and no wait states are required.
- The read-only memory can be read by the core as well as by DMA. It can be cached and protected by CPLD blocks like external memory.
- The boot ROM not only contains boot-strap loader code, it also provides some subfunctions that are user-callable at runtime.

ADSP-BF537 Memory Map

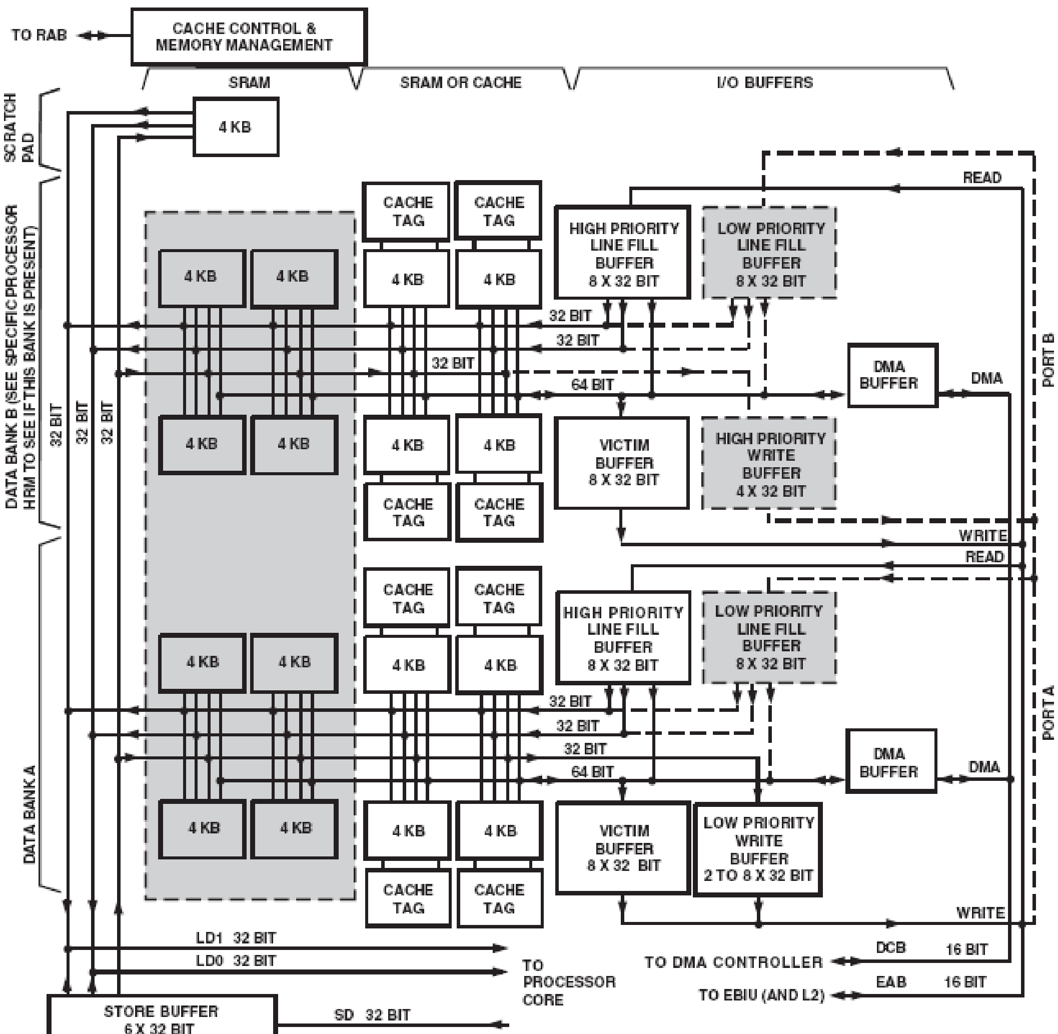


- 64K bytes of instruction memory
  - data bank A/B
  - SRAM/Cache
- 64K bytes of data memory
  - data bank A: SRAM/Cache
  - data bank B: SRAM/Cache
- 4K bytes of scratchpad memory
- 132K bytes of internal Memory are available
- 2K bytes of on-chip boot ROM
- 4M bytes of MMR registers
- 512M bytes of SDRAM

L1 Instruction Memory



L1 Data Memory

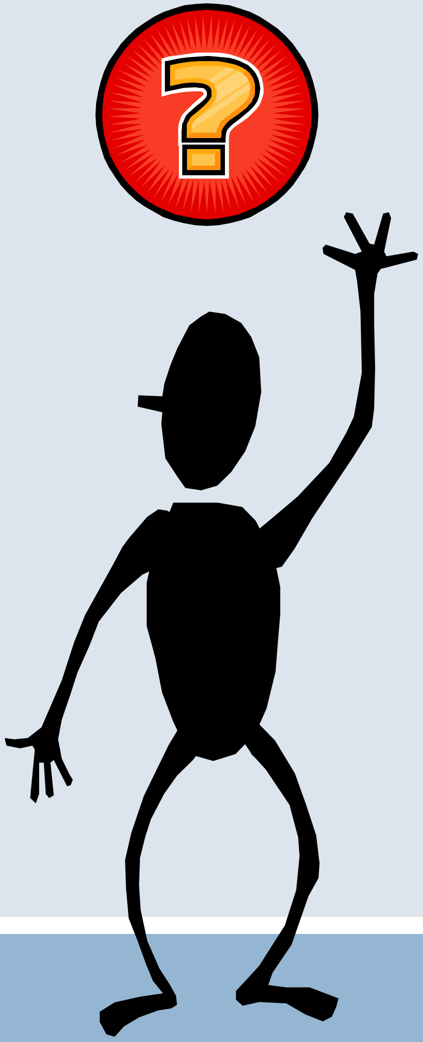


The shaded blocks are not present on all derivatives. For more information, please refer to the corresponding processor hardware reference.

Recommended bibliography

- Blackfin Processor Programming Reference, Revision 1.3, September 2008
  - Ch5: ADDRESS ARITHMETIC UNIT
  - Ch6: MEMORY

- NOTE: Many images used in this presentation were extracted from the recommended bibliography.



Questions?

Thank you!