Performance Tuning on the Blackfin Processor

ANALOG DEVICES

Outline

- Introduction
- Building a Framework
- Memory Considerations
- Benchmarks
- Managing Shared Resources
- Interrupt Management
- An Example
- Summary

Introduction

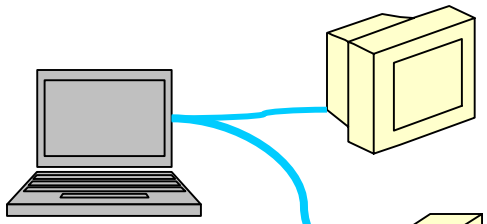
- **The first level of optimization is provided by the compiler**
- **The remaining portion of the optimization comes from techniques performed at the “system” level**
  - **Memory management**
  - **DMA management**
  - **Interrupt management**
- **The purpose of this presentation is to help you understand how some of the system aspects of your application can be managed to tune performance**

One Challenge

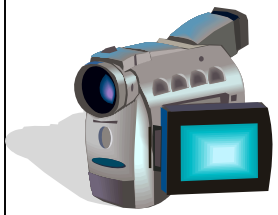
Format	Columns	Rows	Total Pixels	Type of Memory Required For Direct Implementation
SQCIF	128	96	12288	L1, L2, off-chip L2
QCIF	176	144	25344	L1, L2, off-chip L2
CIF	352	288	101376	L2, off-chip L2
VGA	640	480	307200	off-chip L2
D-1 NTSC	720	486	349920	off-chip L2
4CIF	704	576	405504	off-chip L2
D-1 PAL	720	576	414720	off-chip L2
16CIF	1408	1152	1622016	off-chip L2

All formats require management of multiple memory spaces

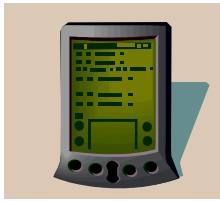
How do we get from here



Unlimited memory
Unlimited processing
File-based
Power cord



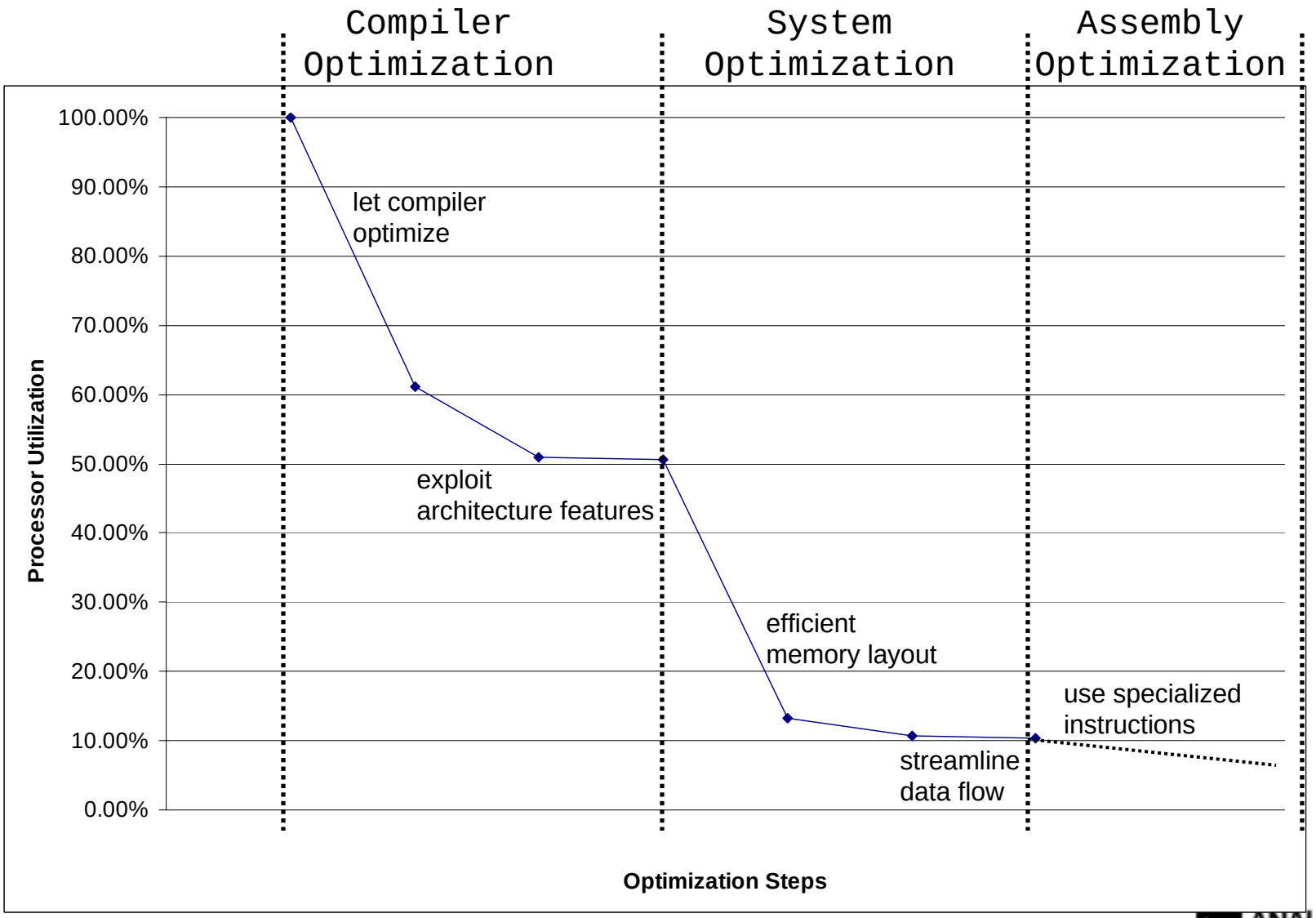
to here?

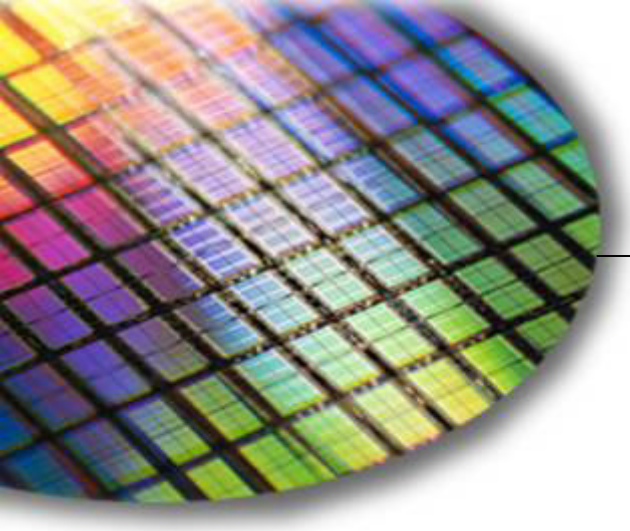


Limited memory
Finite processing
Stream-based
Battery

ANALOG DEVICES

The path to optimization





Creating a Framework

The Importance of Creating a "Framework"

- **Framework definition: The software infrastructure that moves code and data within an embedded system**
- **Building this early on in your development can pay big dividends**
- **There are three categories of frameworks that we see consistently from Blackfin developers**

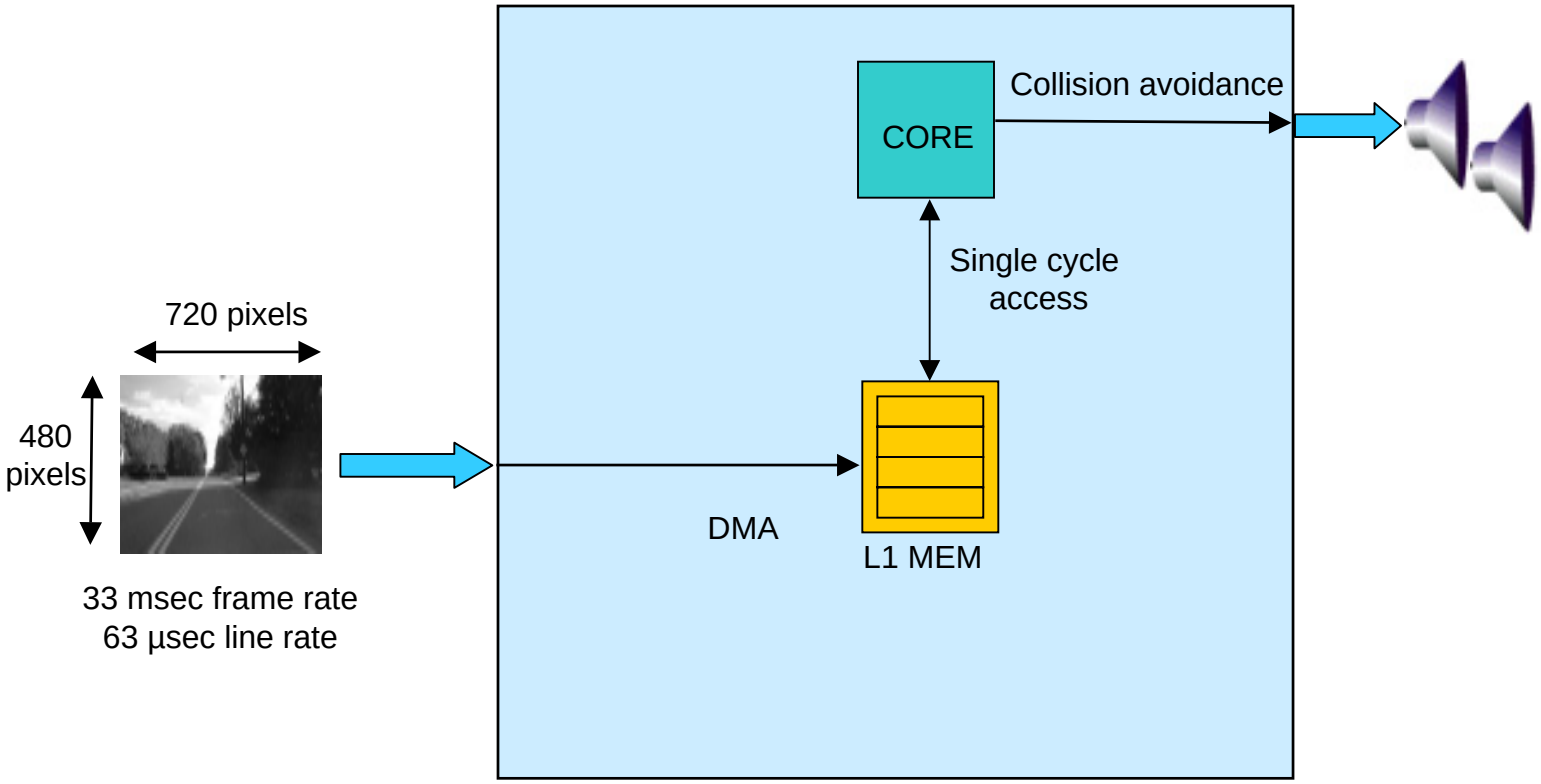
Common Frameworks

- **Processing on the fly**
  - **Safety-critical applications (e.g., automotive lane departure warning system)**
  - **Applications without external memory**
- **Programming ease overrides performance**
  - **Programmers who need to meet a deadline**
- **Performance supersedes programming ease**
  - **Algorithms that push the limits of the processor**

Processing on the fly

- **Can’t afford to wait until large video buffers filled in external memory**
- **Instead, they can be brought into on-chip memory immediately and processed line-by-line**
  - **This approach can lead to quick results in decision-based systems**
- **The processor core can directly access lines of video in on-chip memory.**
  - **Software must ensure the active video frame buffer is not overwritten until processing on the current frame is complete.**

Example of “Processing-on-the-fly” Framework

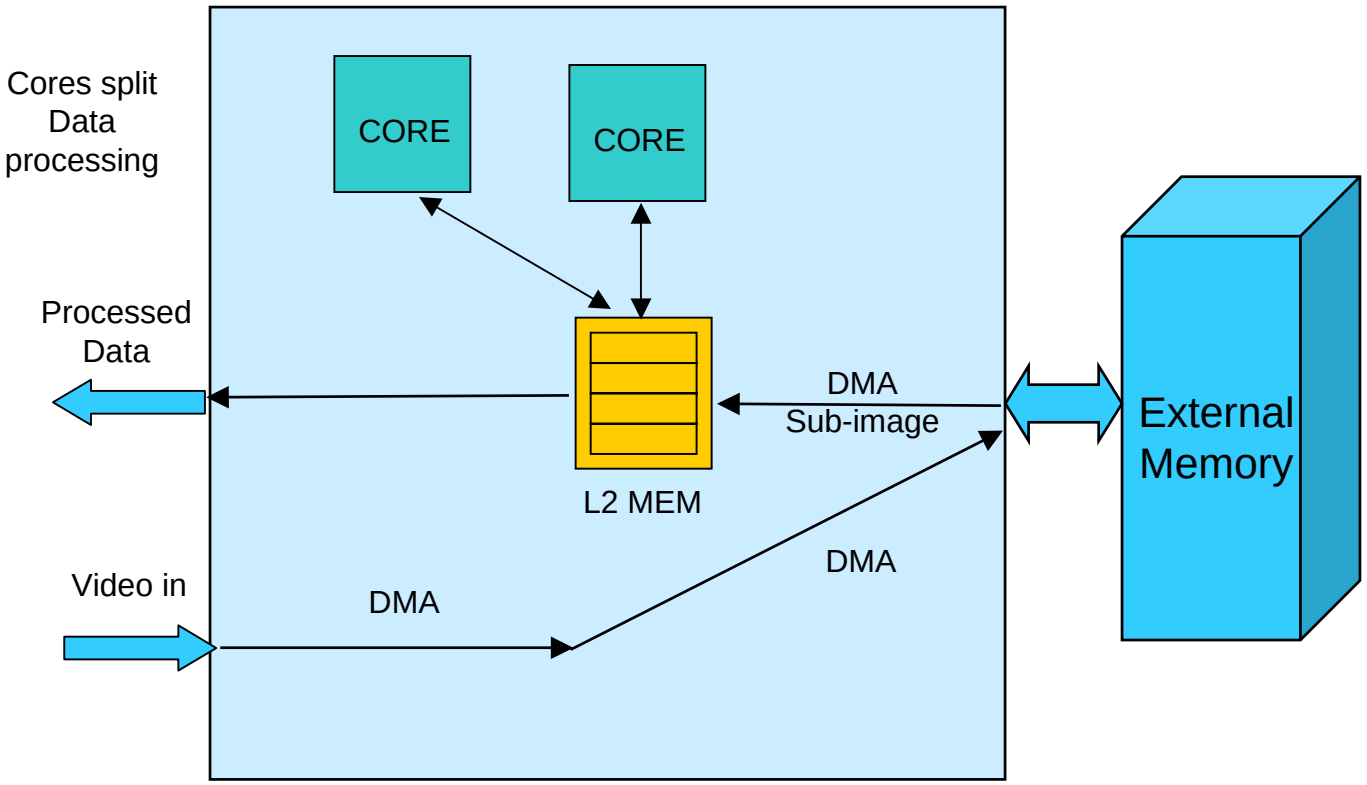


Data is processed one line at a time instead of waiting for an entire frame to be collected.

"Programming Ease Rules"

- **Strives to achieve simplest programming model at the expense of some performance**
- **Focus is on time-to-market**
  - **Optimization isn't as important as time-to-market**
    - It can always be revisited later
  - **Provides a nice path for upgrades!**
- **Easier to develop for both novices and experts**

Example of “Programming Ease Rules” framework

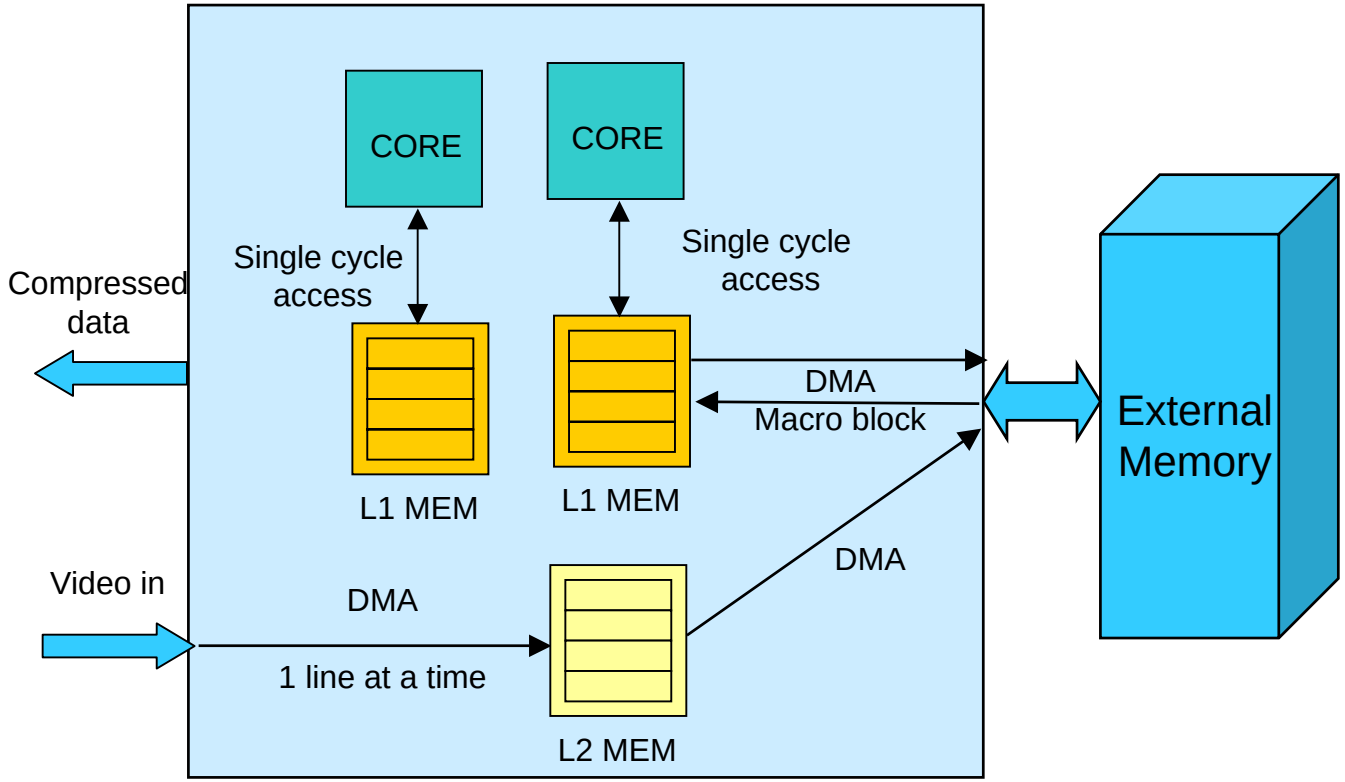


This programming model best matches time-to-market focused designs

“Performance Rules”

- **Bandwidth-efficient**
- **Strives to attain best performance, even if programming model is more complex**
  - **Might include targeted assembly routines**
- **Every aspect of data flow is carefully planned**
- **Allows use of less expensive processors because the device is “right-sized” for the application**
- **Caveats**
  - **Might not leave enough room for extra features or upgrades**
  - **Harder to reuse**

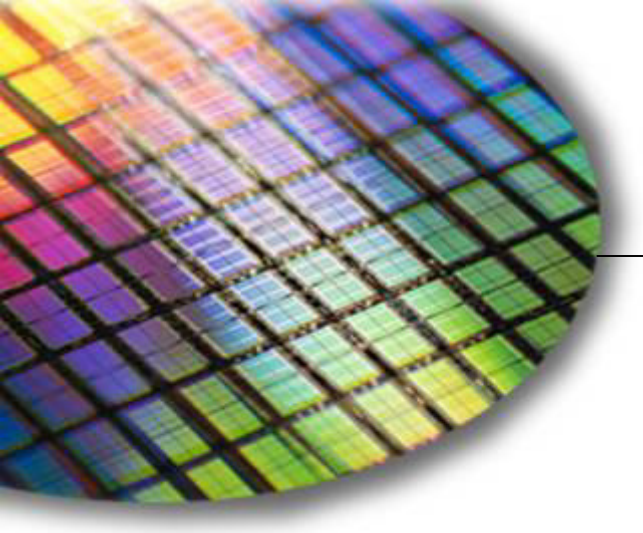
Example of “Performance Rules” Framework



Provides most efficient use of external memory bus.

Common Attributes of Each Model

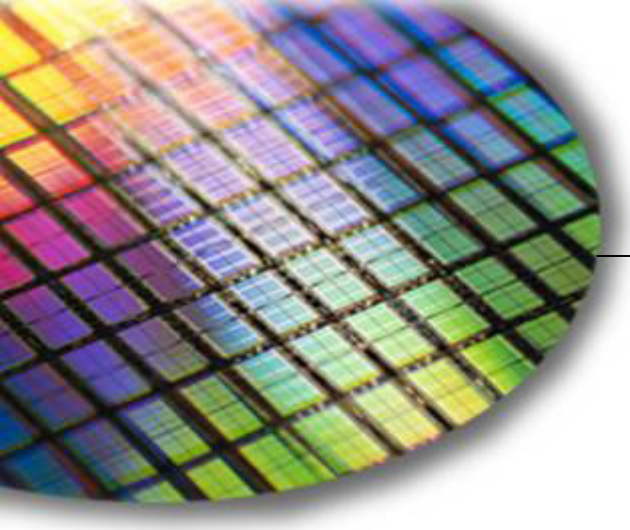
- **Instruction and data management is best done up front in the project**
- **Small amounts of planning can save headaches later on**



Features that can be used to improve performance

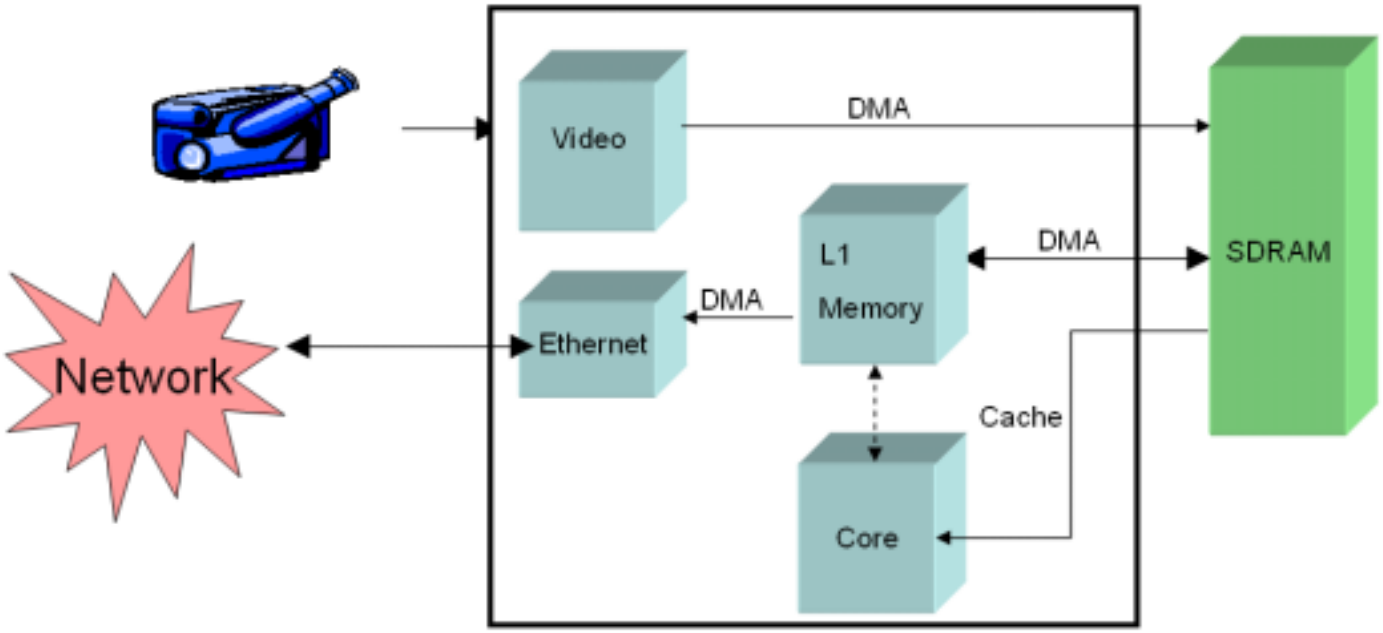
Now that we have discussed the background...

- **We will now review the concepts that will help you tune performance**
  - **Using Cache and DMA**
  - **Managing memory**
  - **Managing DMA channels**
  - **Managing interrupts**

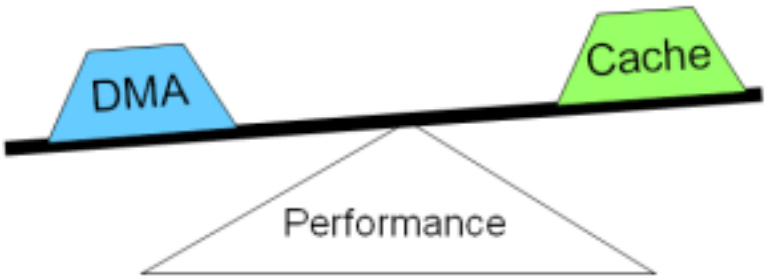


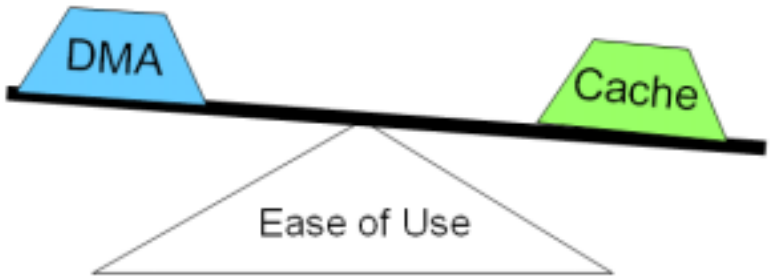
Cache vs. DMA

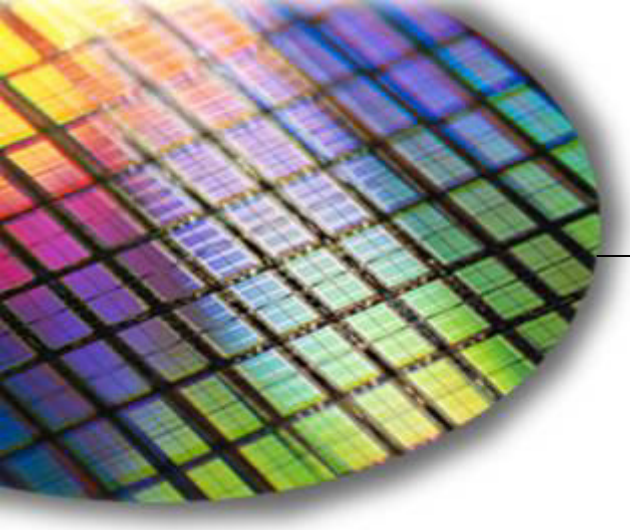
Cache versus DMA



Programmers may find a combination is best

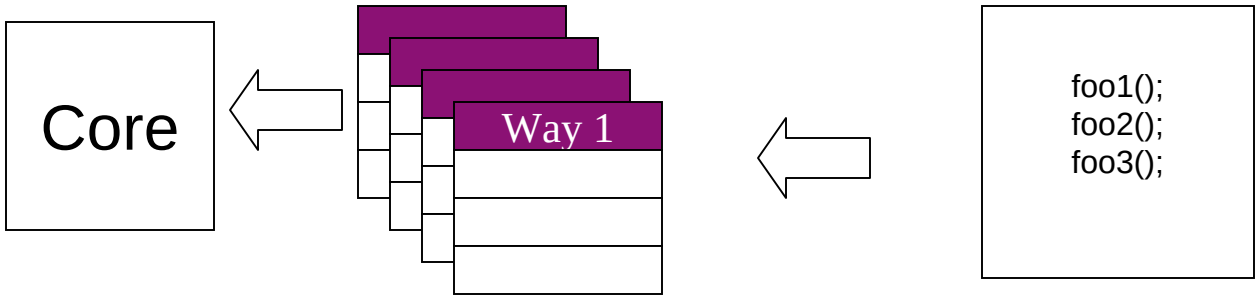






Cache

Configuring internal instruction memory as cache



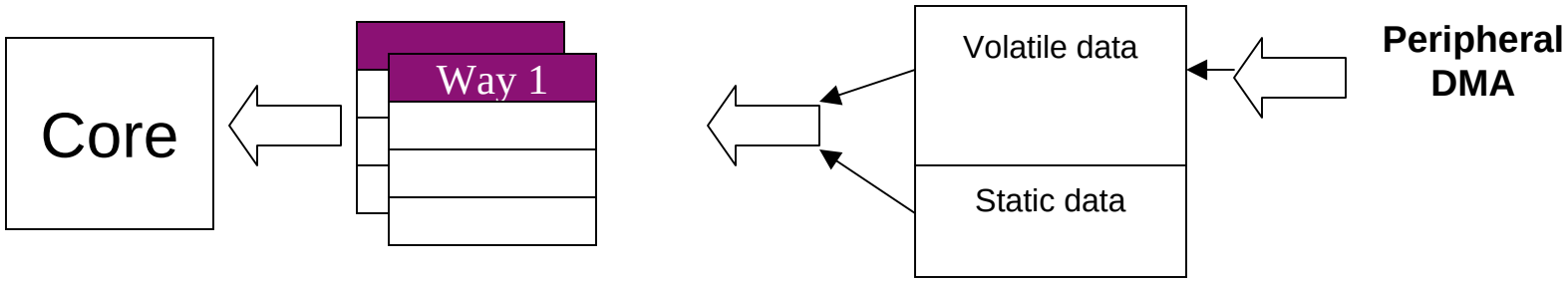
Example: L1 instruction memory configured as 4-way set-associative cache

Instructions are brought into internal memory where single cycle performance can be achieved

Instruction cache provides several key benefits to increase performance

- **Usually provides the highest bandwidth path into the core**
  - For linear code, the next instructions will be on the way into the core after each cache miss
- **The most recently used instructions are the least likely to be replaced**
- **Critical items can be locked in cache, by line**
- **Instructions in cache can execute in a single cycle**

Configuring internal data memory as cache

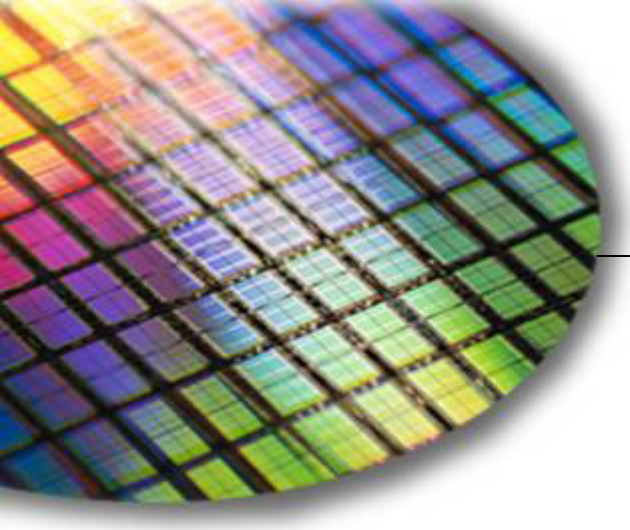


Example: Data brought in from a peripheral

- **Data cache also provides a way to increase performance**
  - Usually provides the highest bandwidth path into the core
    - For linear data, the next data elements will be on the way into the core after each cache miss
  - *Write-through* option keeps "source" memory up to date
    - Data written to source memory every time it is modified
  - *Write-back* option can further improve performance
    - Data only written to source memory when data is replaced in cache
  - "Volatile" buffers must be managed to ensure coherency between DMA and cache

Write-back vs. Write-through

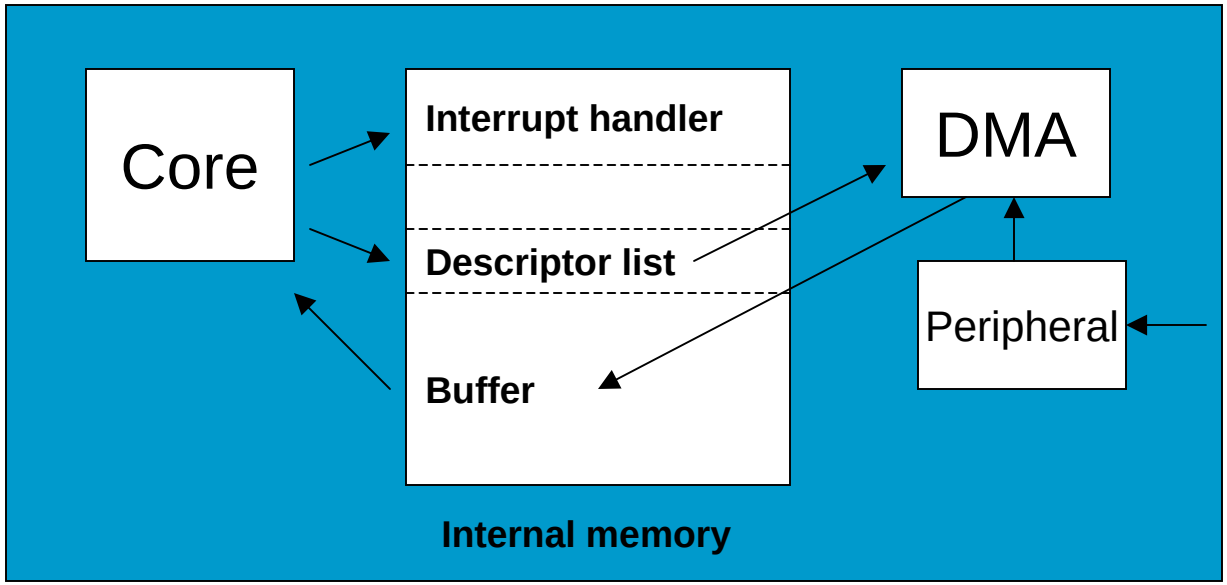
- **Write-back is usually 10-15% more efficient but …**
  - **It is algorithm dependent**
- **Write-through is better when coherency between more than one resource is required**
- **Make sure you try both options when all of your peripherals are running**



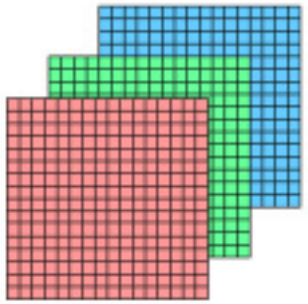
DMA

Why Use a DMA Controller?

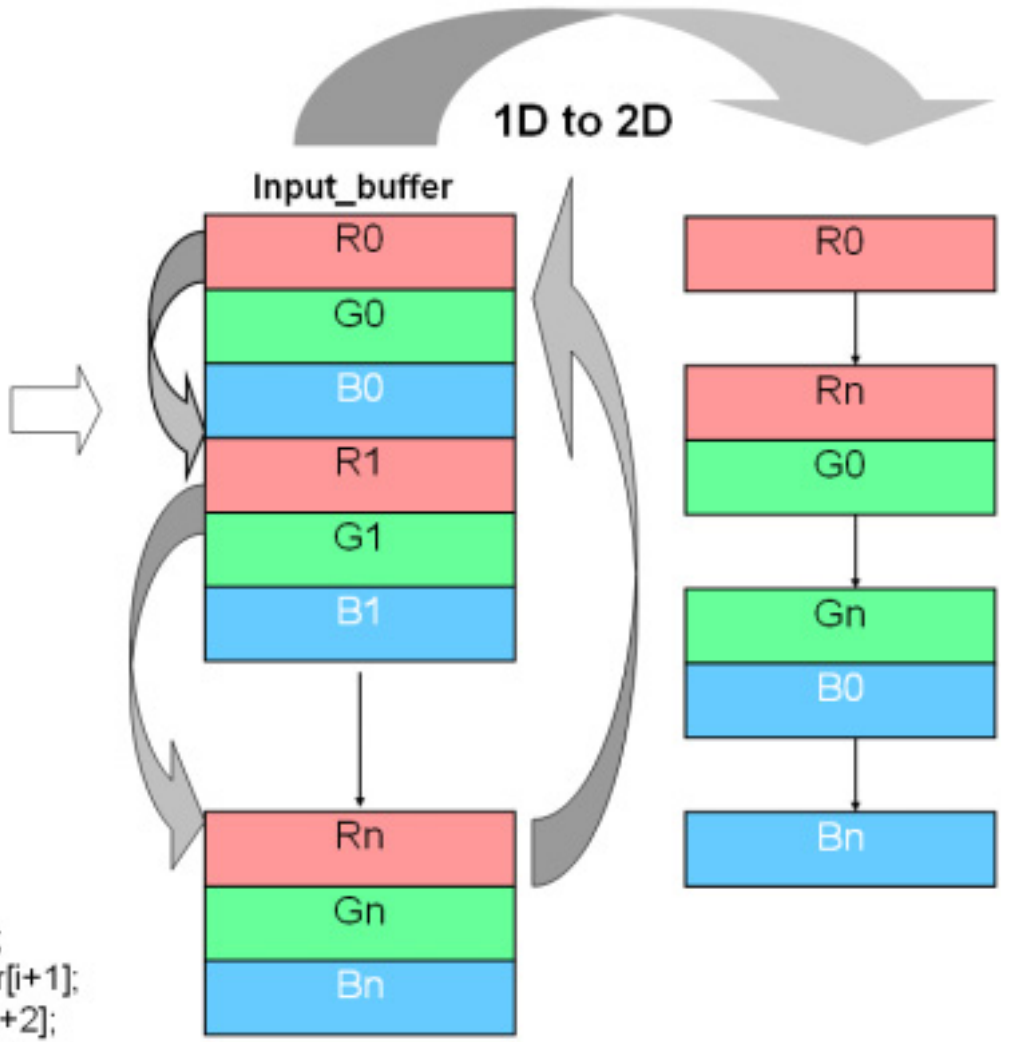
- **The DMA controller runs independently of the core**
  - **The core should only have to set up the DMA and respond to interrupts**
- **Core processor cycles are available for processing data**
- **DMA can allow creative data movement and “filtering”**
  - **Saves potential core passes to re-arrange the data**



Using 2D DMA to efficiently move image data



Red Green Blue data
from sensor

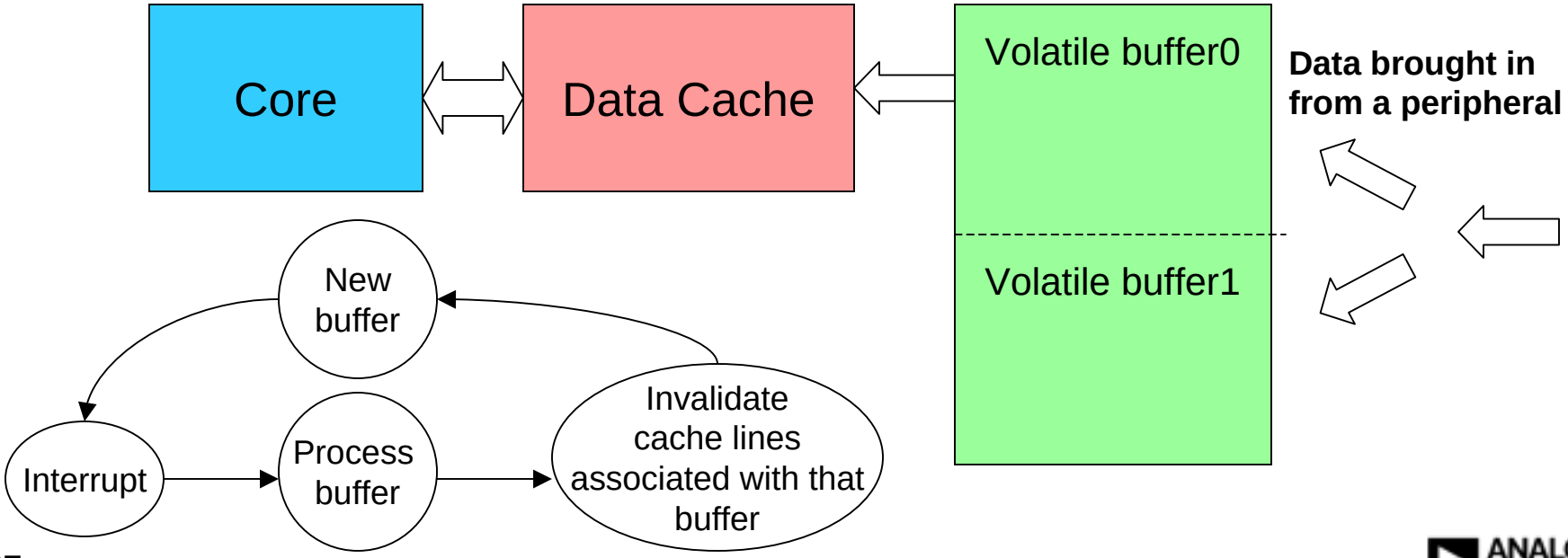


Pseudo-code to bring in this data:

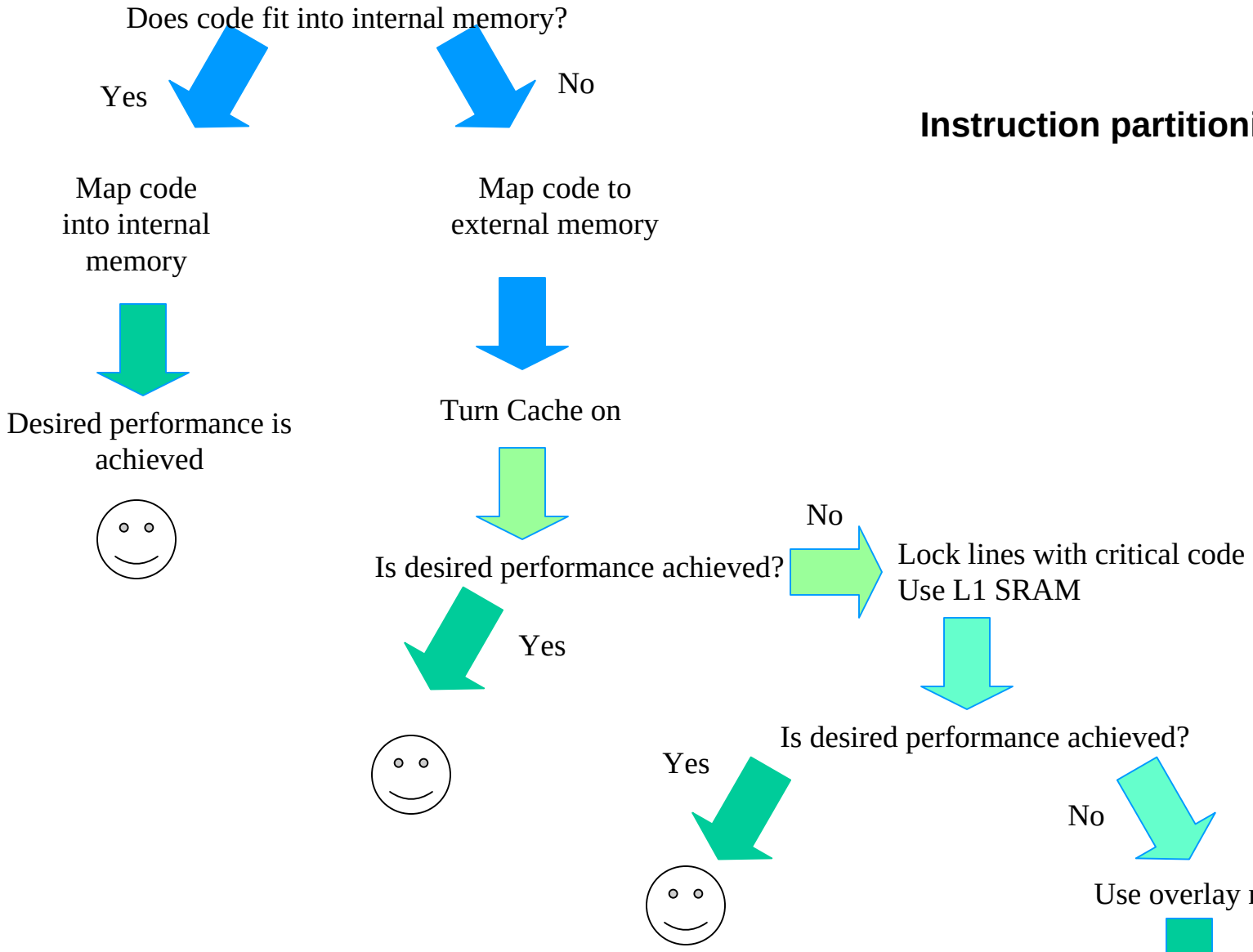
```
for(i=0; i<count; i+=3) {
        red_buffer[i]= input_buffer[i];
        green_buffer[i]= input_buffer[i+1];
        blue_buffer[i]= input_buffer[i+2];
        }
```

DMA and Data Cache Coherence

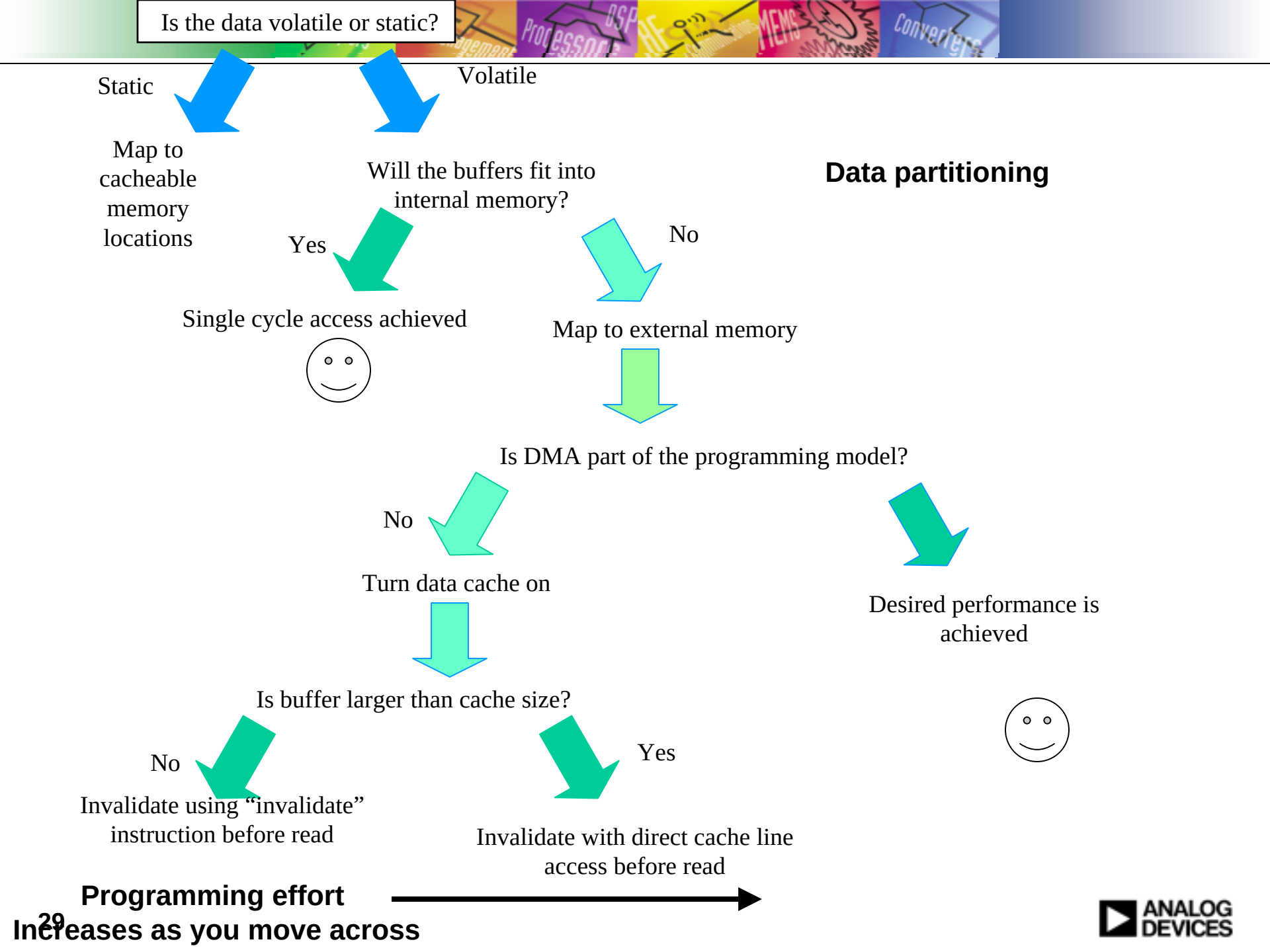
- **Coherency between data cache and buffer filled via DMA transfer must be maintained**
- **Cache must be "invalidated" to ensure "old" data is not used when processing most recent buffer**
  - **Invalidate buffer addresses or actual cache line, depending on size of the buffer**
- **Interrupts can be used to indicate when it is safe to invalidate buffer for next processing interval**
- **This often provides a simpler programming model (with less of a performance increase) than a pure DMA model**



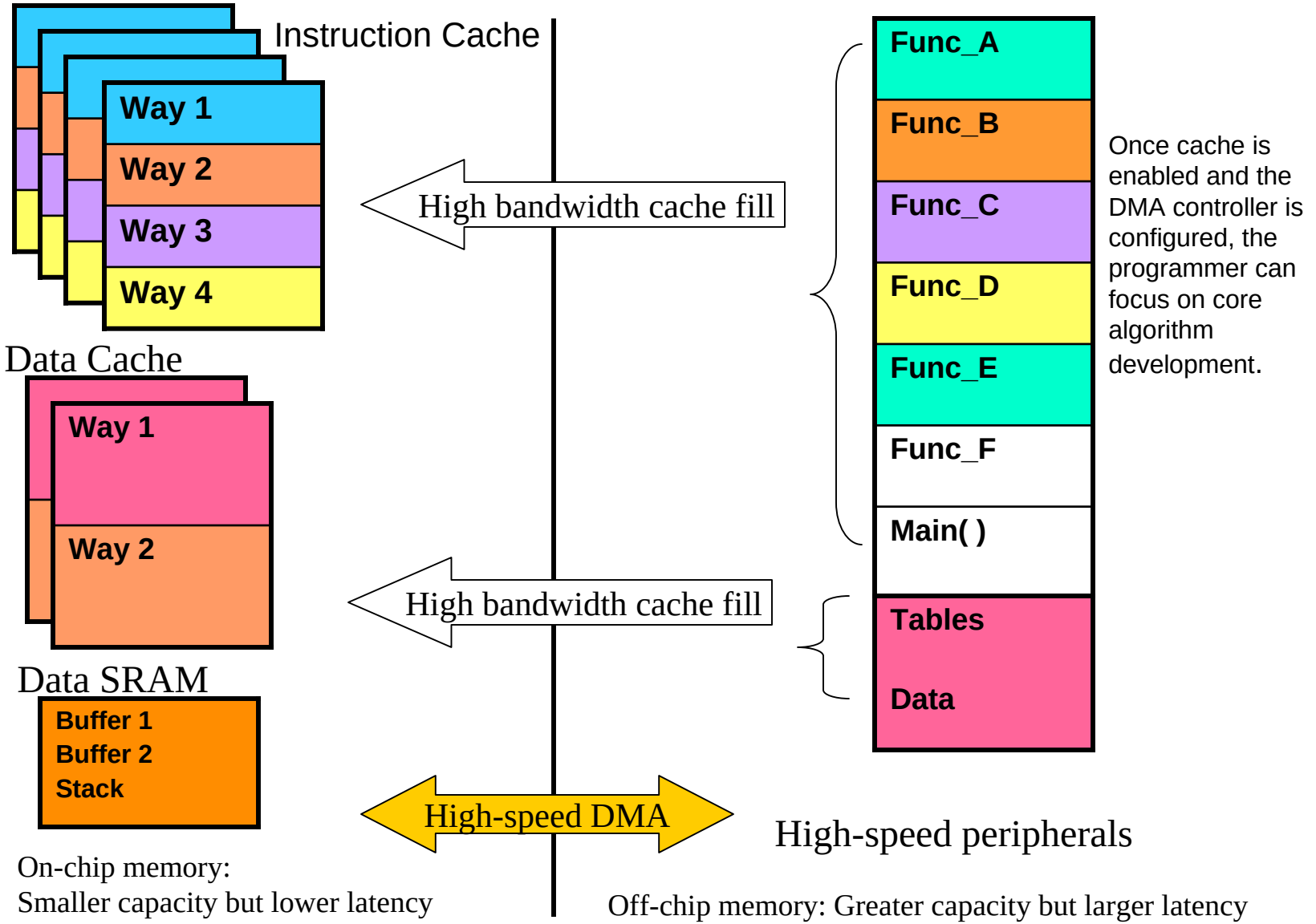
Instruction partitioning

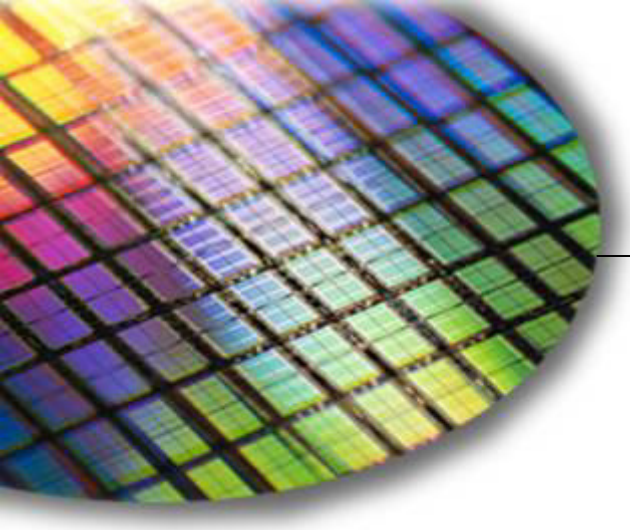


Programming effort Increases as you move across


Is the data volatile or static?
Static
Volatile
Map to cacheable memory locations
Will the buffers fit into internal memory?
Yes
No
Single cycle access achieved
Map to external memory
Is DMA part of the programming model?
No
Turn data cache on
Is buffer larger than cache size?
No
Yes
Invalidate using “invalidate” instruction before read
Invalidate with direct cache line access before read
Desired performance is achieved
Data partitioning
Programming effort
Increases as you move across

A Combination of Cache and DMA Usually Provides the Best Performance



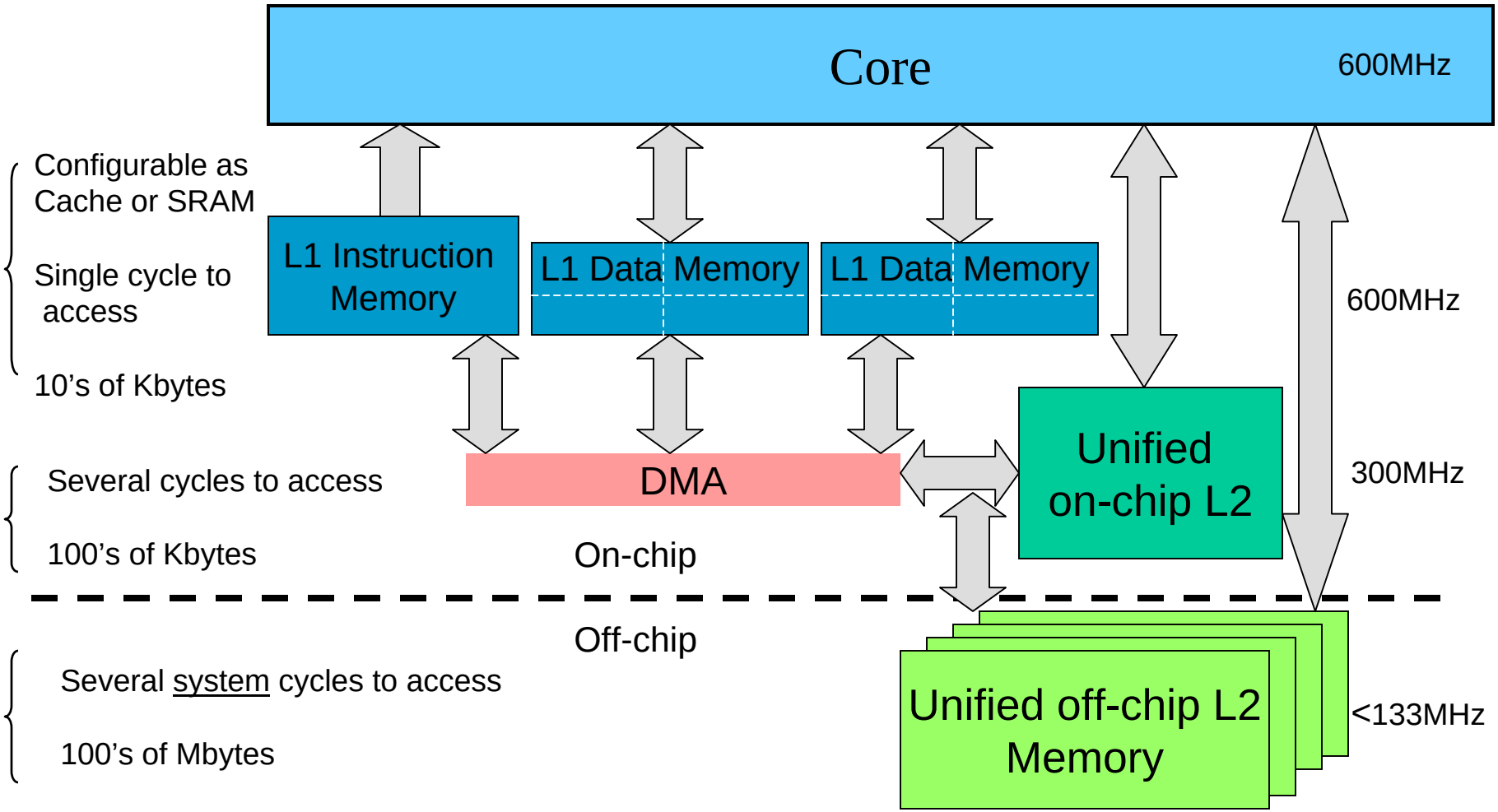


The World Leader in High Performance Signal Processing Solutions

Memory Considerations

ANALOG DEVICES

Blackfin Memory Architecture: The Basics



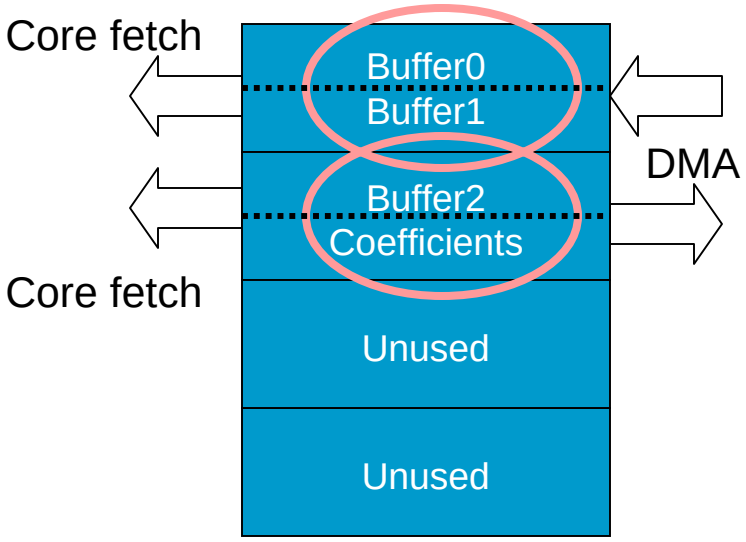
ANALOG DEVICES

Instruction and Data Fetches

- **In a single core clock cycle, the processor can perform**
  - **One instruction fetch of 64 bits and either …**
  - **Two 32-bit data fetches or**
  - **One 32-bit data fetch and one 32-bit data store**
- **The DMA controller can also be running in parallel with the core without any “cycle stealing”**

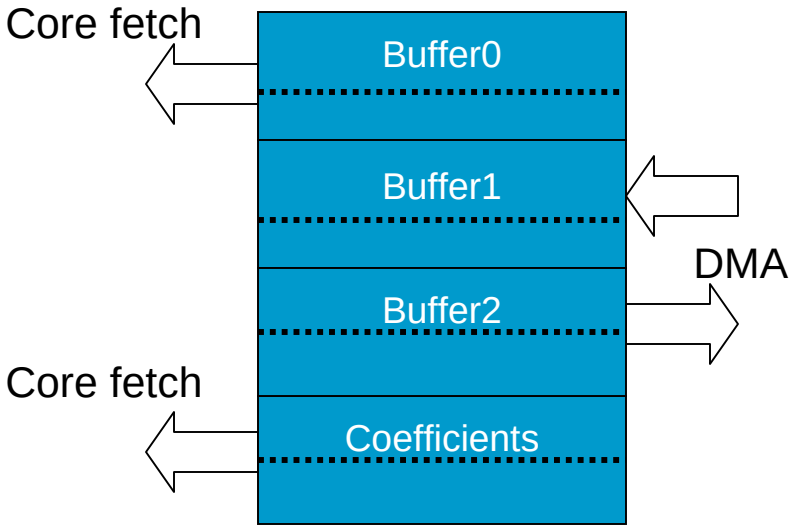
Partitioning Data – Internal Memory Sub-Banks

- **Multiple accesses can be made to different internal sub-banks in the same cycle**



Un-optimized

DMA and core conflict when accessing sub-banks

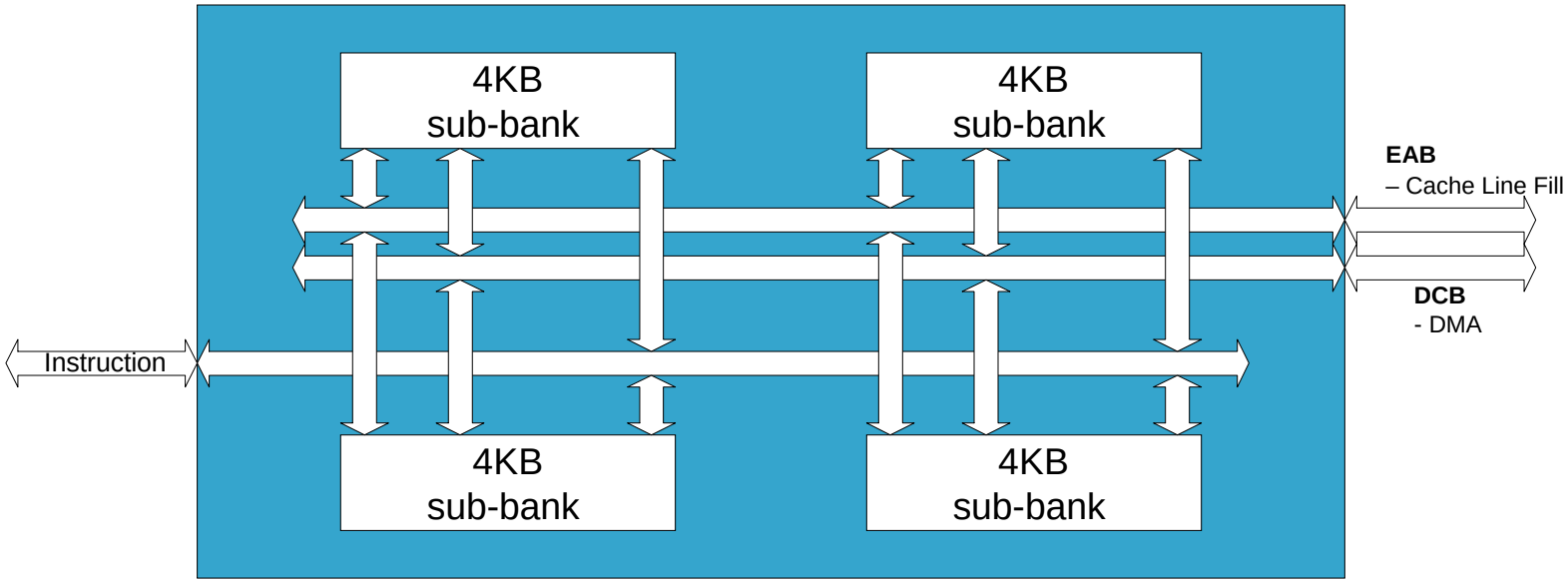


Optimized

Core and DMA operate in harmony

Take advantage of the sub-bank architecture!

L1 Instruction Memory 16KB Configurable Bank



16 KB SRAM

- Four 4KB single-ported sub-banks
- Allows simultaneous core and DMA accesses to different banks

16 KB cache

- Least Recently Used replacement keeps frequently executed code in cache
- 4-way set associative with arbitrary locking of ways and lines

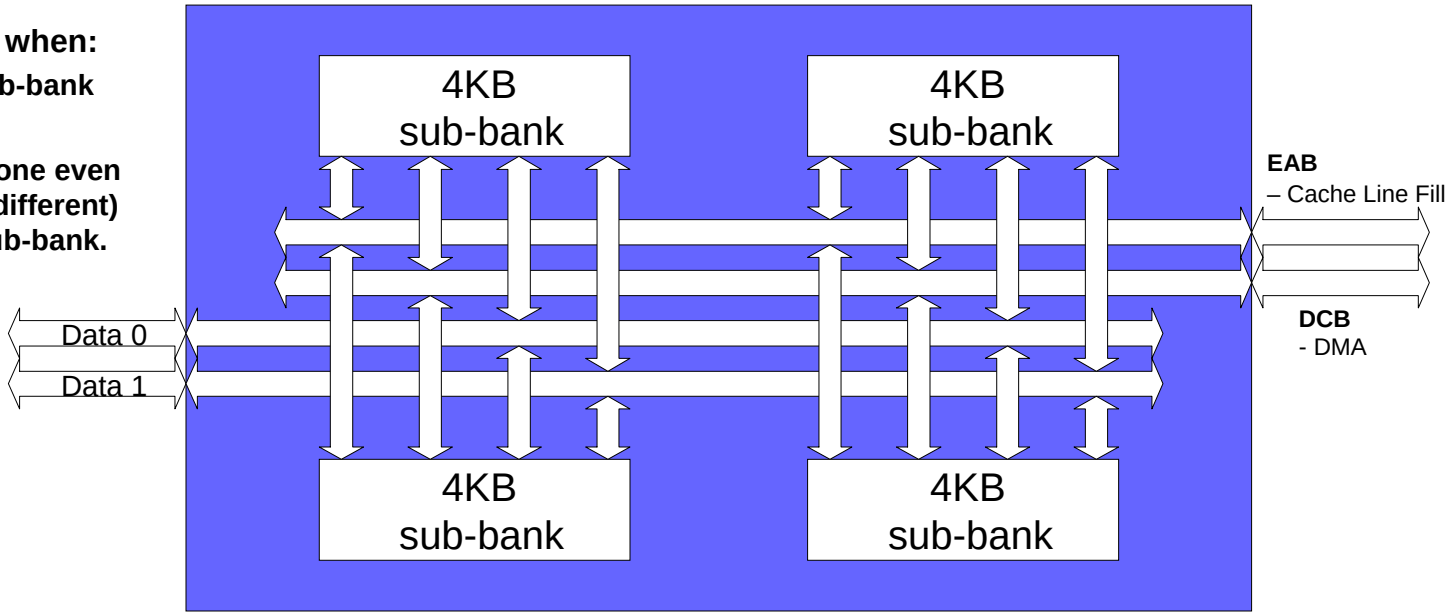
L1 Data Memory 16KB Configurable Bank

Block is Multi-ported when:

Accessing different sub-bank

OR

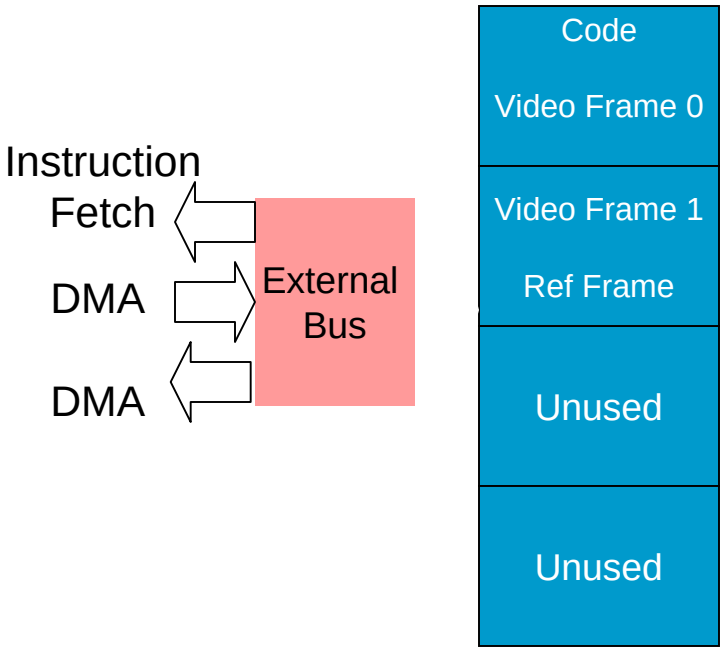
Accessing one odd and one even access (Addr bit 2 different) within the same sub-bank.

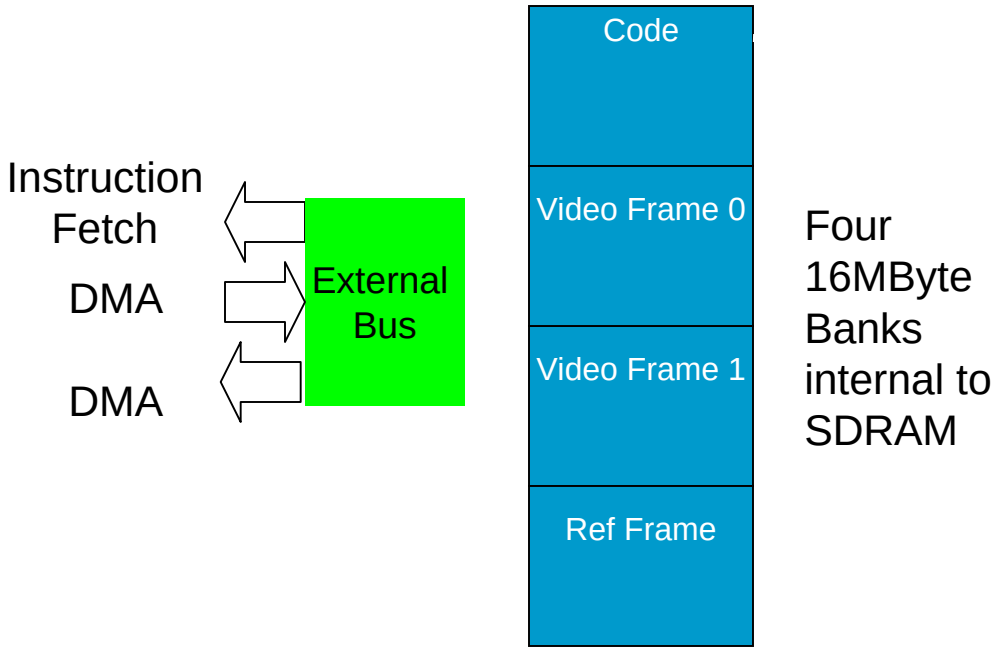


- When Used as SRAM
  - Allows simultaneous dual DAG and DMA access
- When Used as Cache
  - Each bank is 2-way set-associative
  - No DMA access
  - Allows simultaneous dual DAG access

Partitioning Code and Data -- External Memory Banks

- **Row activation within an SDRAM consumes multiple system clock cycles**
- **Multiple rows can be "active" at the same time**
  - **One row in each bank**



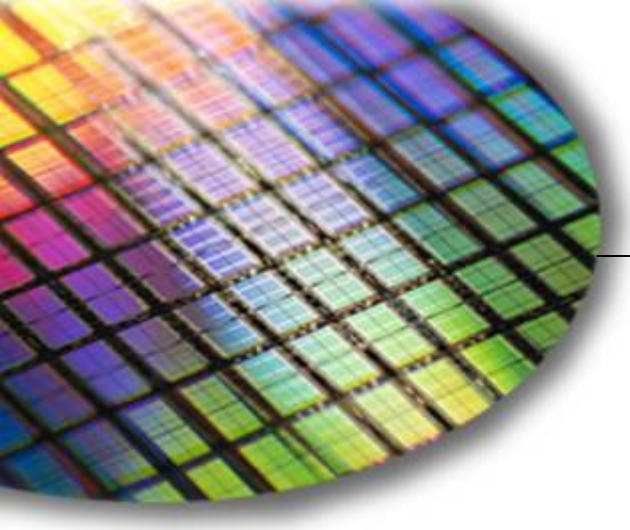


Row activation cycles are taken almost every access

Row activation cycles are spread across hundreds of accesses

Take advantage of all four open rows in external memory to save activation cycles!

ANALOG DEVICES



The World Leader in High Performance Signal Processing Solutions

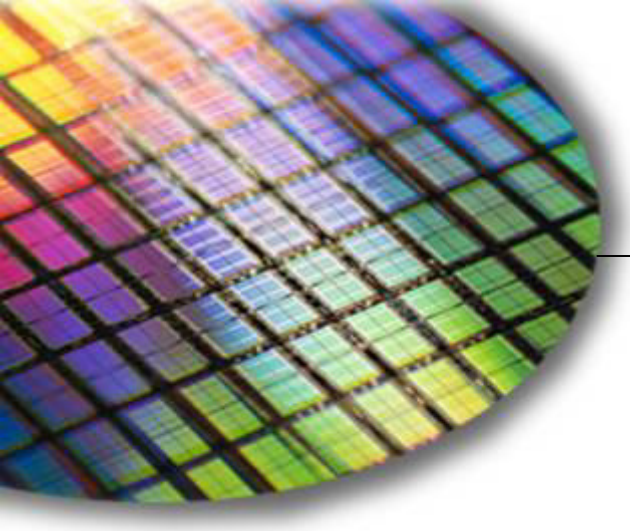
Benchmarks

ANALOG DEVICES

Important Benchmarks

- **Core accesses to SDRAM take longer than accesses made by the DMA controller**
  - For example, Blackfin Processors with a 16-bit external bus behave as follows:
    - 16-bit core reads take 8 System Clock (SCLK) cycles
    - 32-bit core reads take 9 System Clock cycles
    - 16-bit DMA reads take ~1 SCLK cycle
    - 16-bit DMA writes take ~ 1 SCLK cycle

Bottom line: Data is most efficiently moved with the DMA controllers!

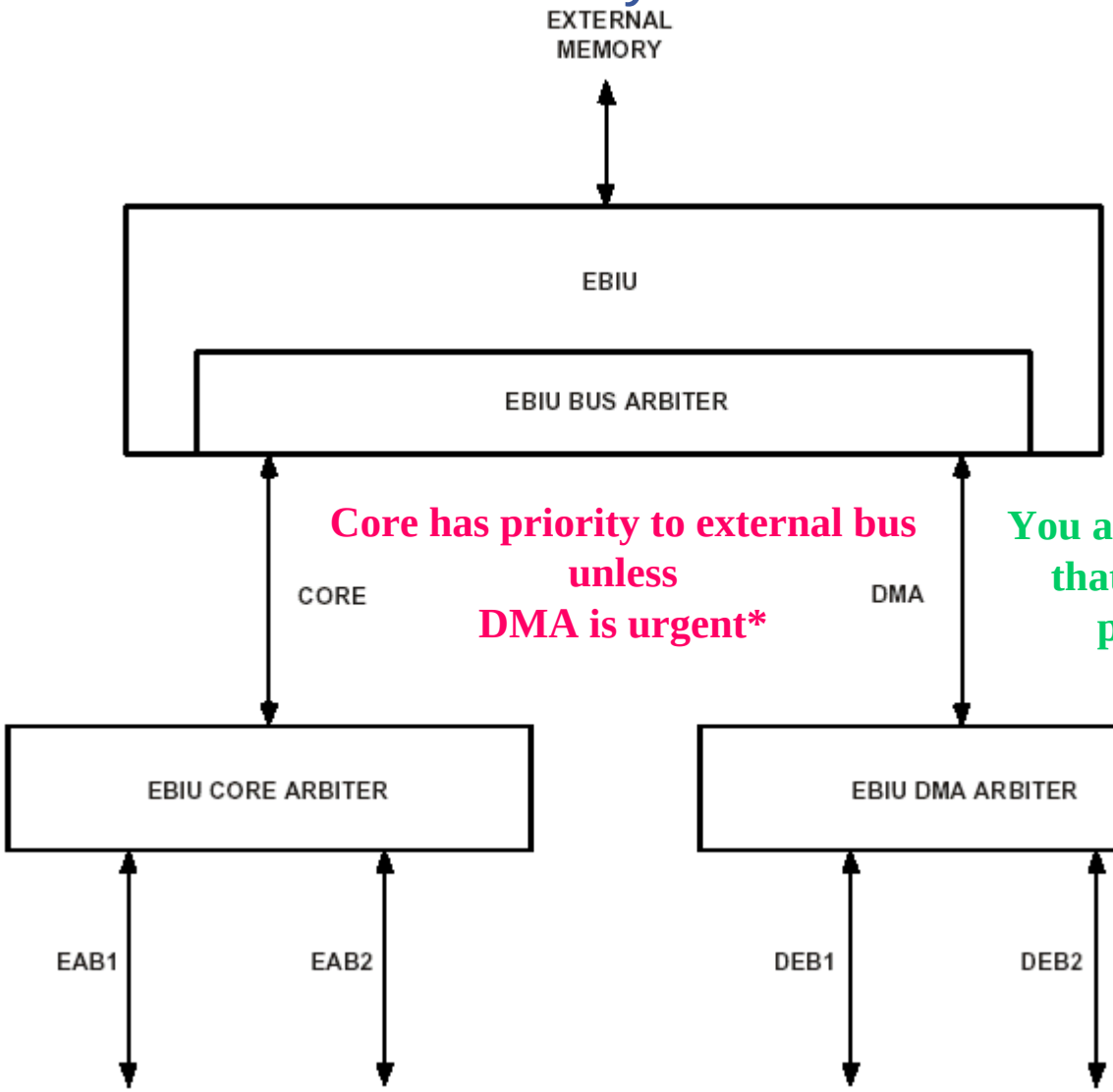


The World Leader in High Performance Signal Processing Solutions

Managing Shared Resources

ANALOG DEVICES

External Memory Interface: ADSP-BF561



Core has priority to external bus unless DMA is urgent*

You are able to program this so that the DMA has a higher priority than the core

*** "Urgent" DMA implies data will be lost if access isn't granted**

Core A is higher priority than Core B

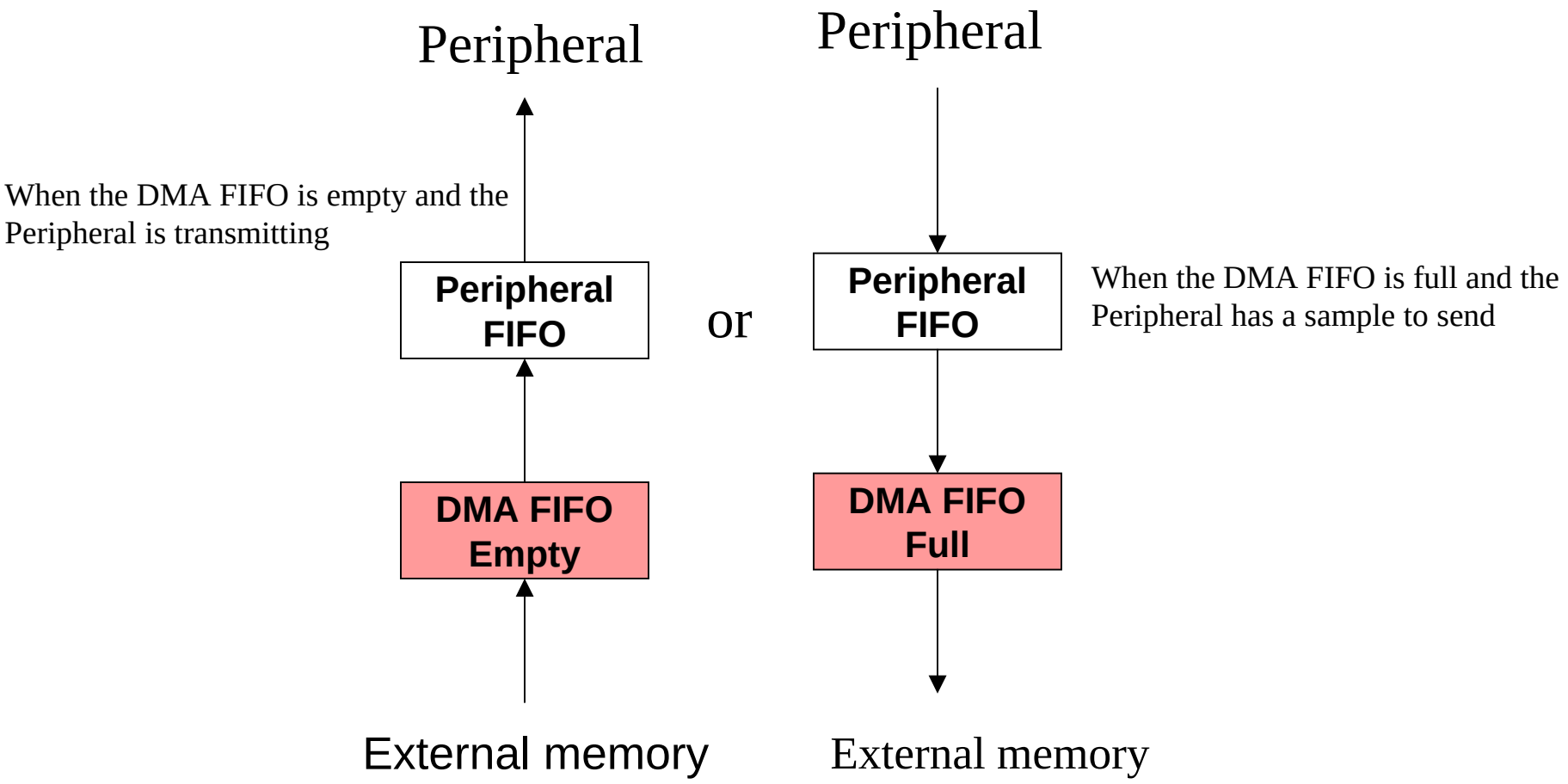
Programmable priority

ANALOG DEVICES

Priority of Core Accesses and the DMA Controller at the External Bus

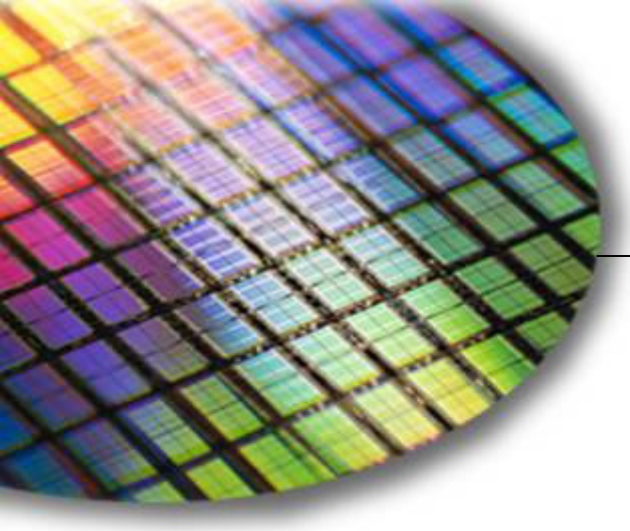
- **A bit within the EBIU_AMGCTL register can be used to change the priority between core accesses and the DMA controller**

What is an “urgent” condition?



Bus Arbitration Guideline Summary: Who wins?

- **External memory (in descending priority)**
  - **Locked Core accesses (testset instruction)**
  - **Urgent DMA**
  - **Cache-line fill**
  - **Core accesses**
  - **DMA accesses**
    - We can set all DMA's to be urgent, which will elevate the priority of all DMAs above Core accesses
- **L1 memory (in descending priority)**
  - **DMA accesses**
  - **Core accesses**

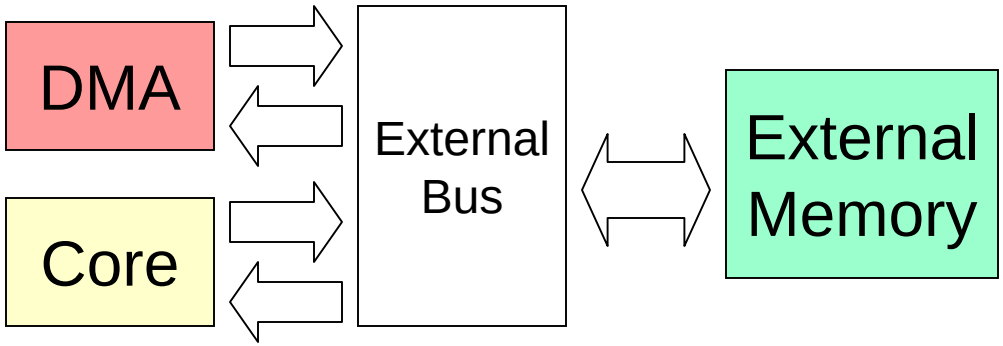


The World Leader in High Performance Signal Processing Solutions

Tuning Performance

ANALOG DEVICES

Transfers in the same direction are more efficient than intermixed accesses in different directions



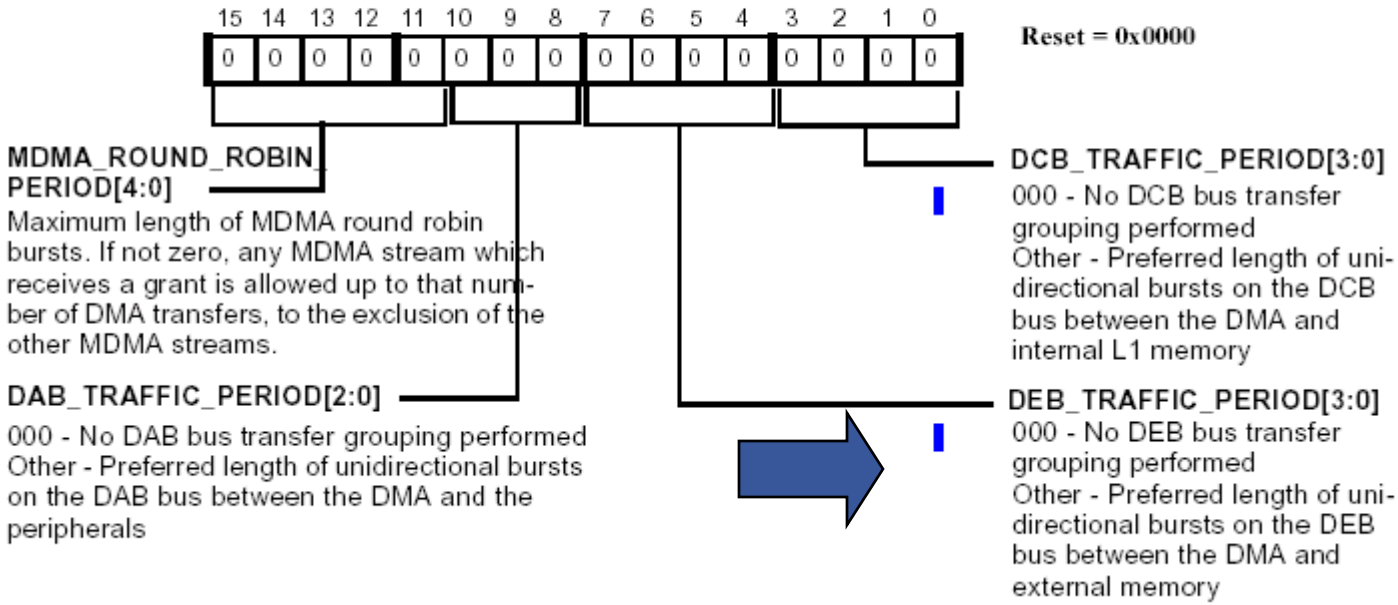
Group transfers in the same direction to reduce number of turn-arounds

Improving Performance

- **DMA Traffic Control improves system performance when multiple DMAs are ongoing (typical system)**
  - **Multiple DMA accesses can be done in the same "direction"**
    - For example, into SDRAM or out of SDRAM
  - **Makes more efficient use of SDRAM**

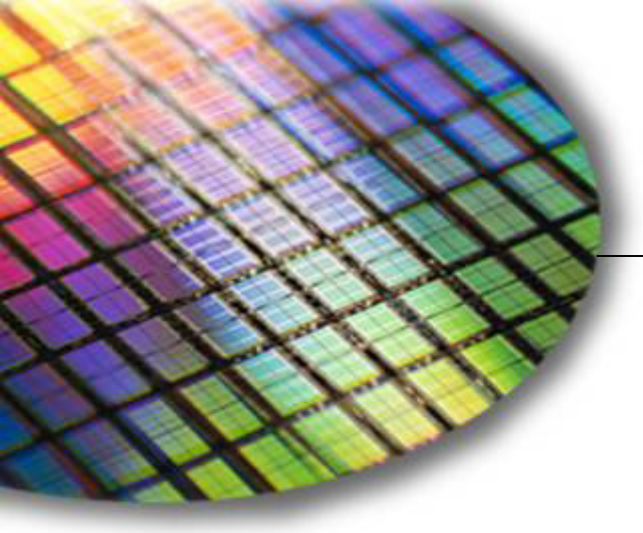
DMA Traffic Control

DMA Traffic Control Counter Period Register (TC_PER)



DEB_Traffic_Period has the biggest impact on improving performance

The correct value is application dependent but if 3 or less DMA channels are active at any one time, a larger value (15) of DEB_Traffic_Period is usually better. When more than 3 channels are active, a value closer to the mid value (4 to 7) is usually better.



Priority of DMAs

Priority of DMA Channels

- **Each DMA controller has multiple channels**
- **If more than one DMA channel tries to access the controller, the highest priority channel wins**
- **The DMA channels are programmable in priority**
- **When more than one DMA controller is present, the priority of arbitration is programmable**
- **The DMA Queue Manager provided with System Services should be used**

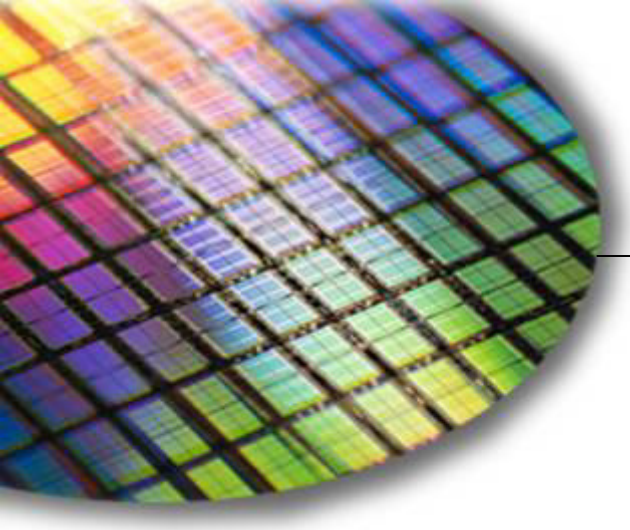
Interrupt Processing

Common Mistakes with Interrupts

- **Spending too much time in an interrupt service routine prevents other critical code from executing**
  - **From an architecture standpoint, interrupts are disabled once an interrupt is serviced**
    - Higher priority interrupts are enabled once the return address (RETI register) is saved to the stack
- **It is important to understand your application real-time budget**
  - **How long is the processor spending in each ISR?**
  - **Are interrupts nested?**

Programmer Options

- **Program the most important interrupts as the highest priority**
- **Use nesting to ensure higher priority interrupts are not locked out by lower priority events**
- **Use the Call-back manager provided with System Services**
  - **Interrupts are serviced quickly and higher priority interrupts are not locked out**



A Example

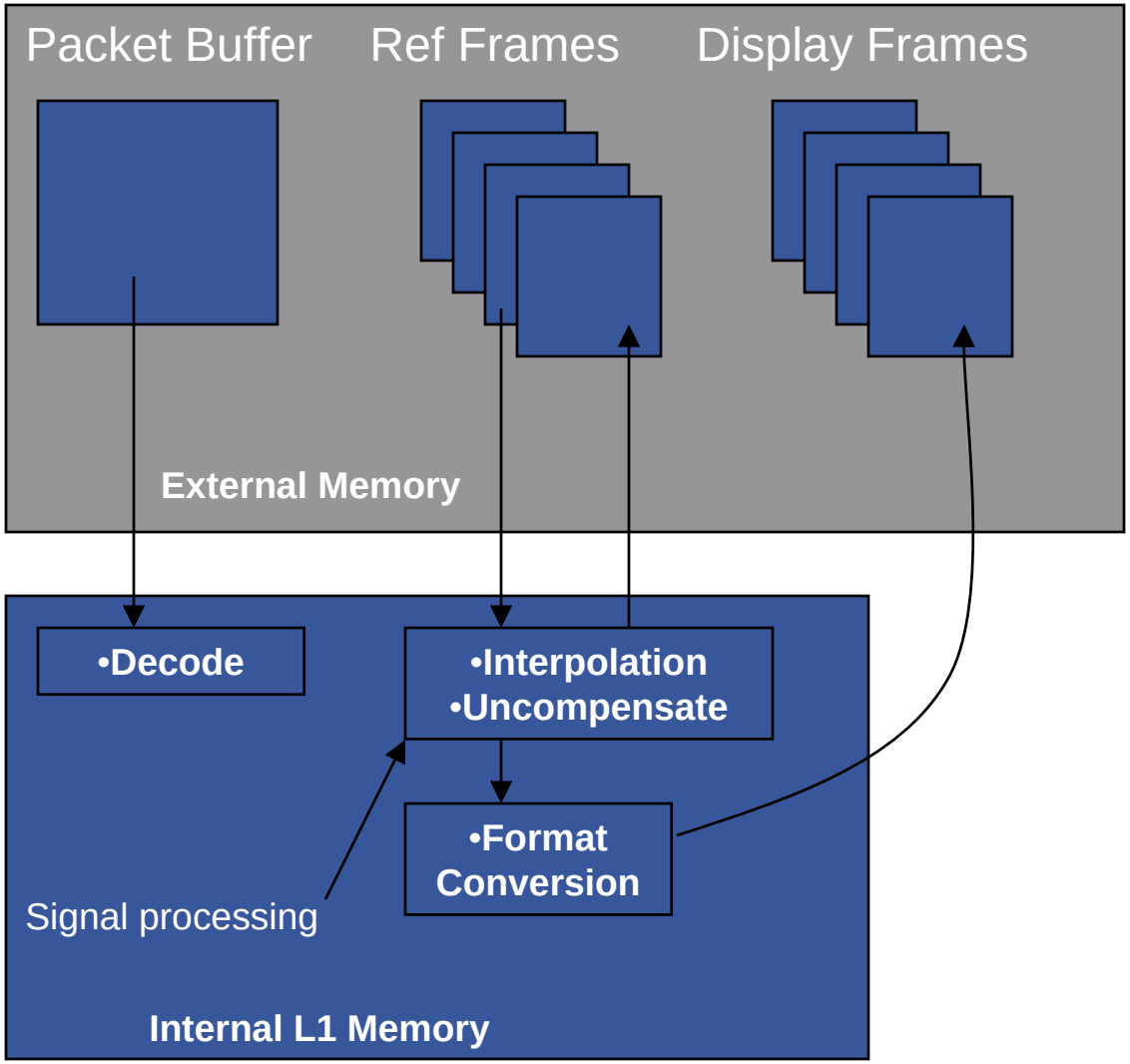
Video Decoder Budget

- **What's the cycle budget for real-time decoding?**
  - (Cycle/pel * pixel/frame * frame/sec) < core frequency
  - ⇒Cycle/pel < (core frequency / [pixel/frame * frame/sec])
  - ⇒Leave at least 10% of the MIPS for other things (audio, system, transport layers…)
  - For D-1 video: 720x480, 30 frame/sec (~10M pel/sec)
  - Video Decoder budget only ~50 cycle/pel
- **What's the bandwidth budget?**
  - ([Encoded bitstream in via DMA] + [Reference frame in via DMA] + [Reconstructed MDMA Out] + [ITU-R 656 Out] + [PPI Output]) < System Bus Throughput
  - Video Decoder Budget ~130 MB/s

Video Decoders

Input Bitstream

1011001001…



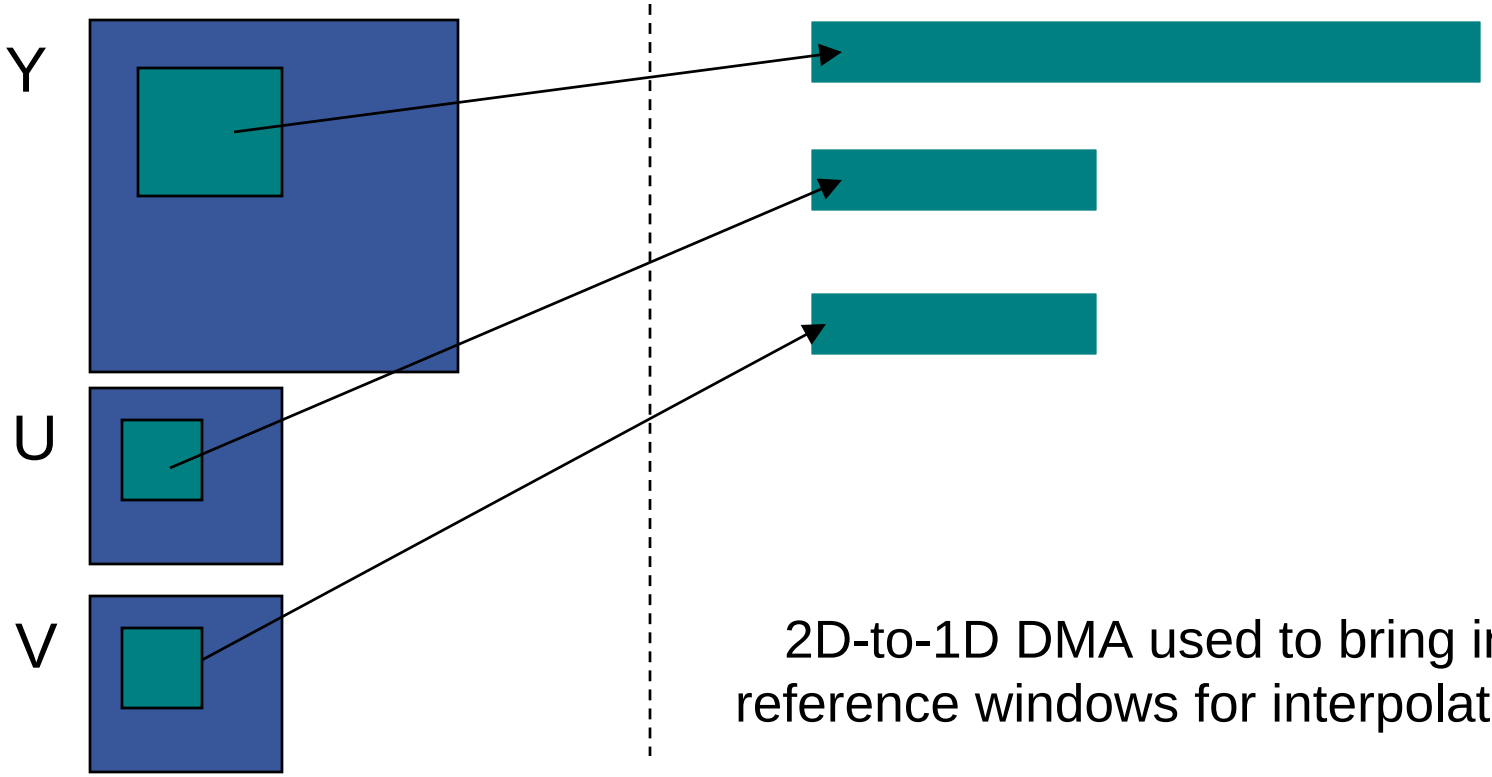
ANALOG DEVICES

Data Placement

- Large buffers go in SDRAM
  - Packet Buffer (~ 3 MB)
  - 4 Reference Frames (each 720x480x1.5 bytes)
  - 8 Display Frames (each 1716x525 bytes)
- Small Buffers go in L1 memory
  - VLD Lookup Tables
  - Inverse Quantization, DCT, zig-zag, etc.
  - Temporary Result Buffers

Data Movement (SDRAM to L1)

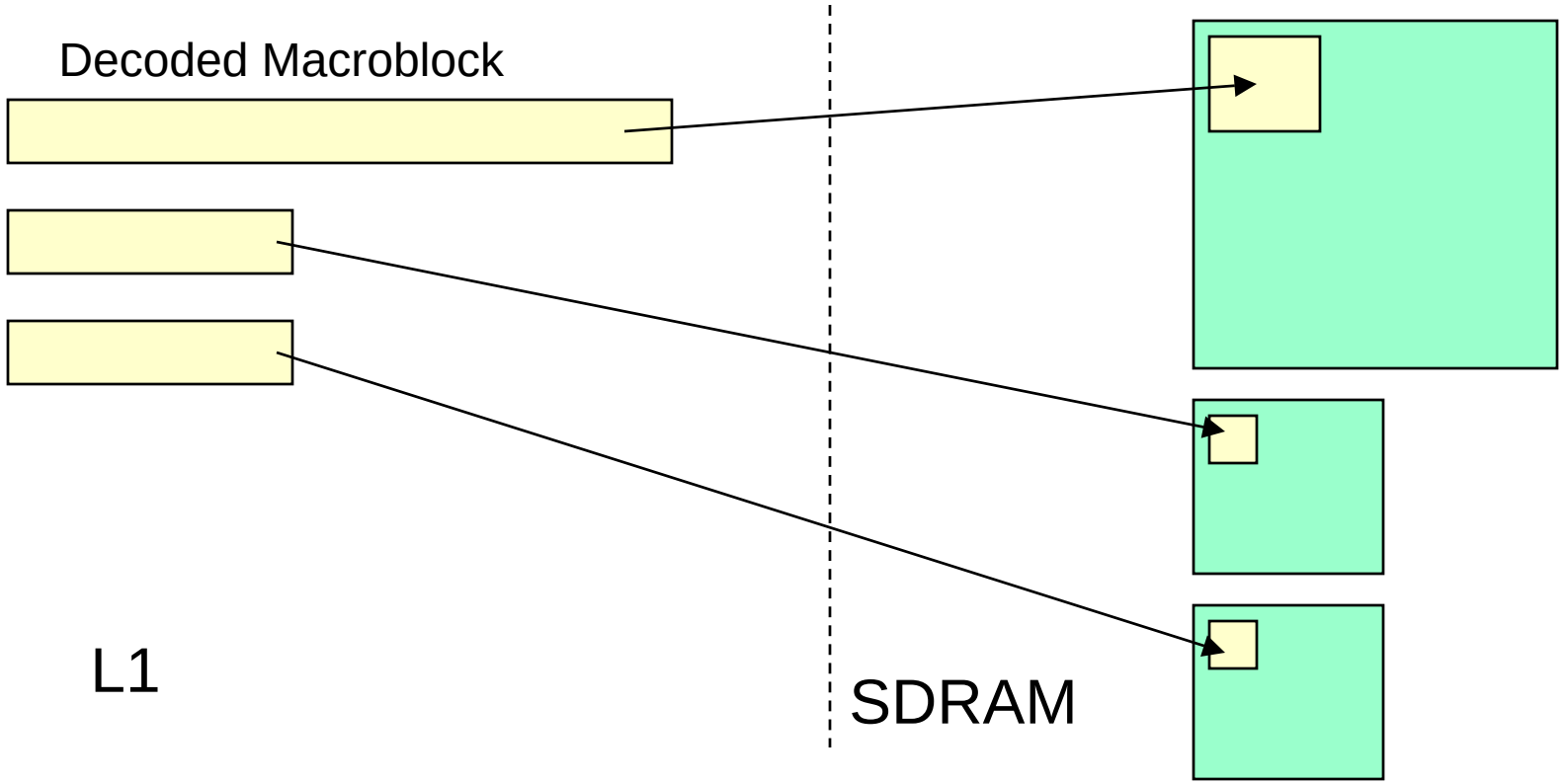
- Reference Frames are 2D in SDRAM
- Reference Windows are linear in L1



2D-to-1D DMA used to bring in reference windows for interpolation

Data Movement (L1 to SDRAM)

- All temporary results in L1 buffers are stored linearly.
- All data brought in from SDRAM are used linearly.



1D-to-2D DMA transfers decoded macroblock to SDRAM to build reference frame.

Bandwidth Used by DMA

Input Data Stream	1 MB/s
Reference Data In	30 MB/s
In Loop Filter Data In	15 MB/s
Reference Data Out	15 MB/s
In Loop Filter Data Out	15 MB/s
Video Data Out	27 MB/s

- Calculations based on 30 frames per second.

Be careful not to simply add up the bandwidths!

Shared resources, bus turnaround, and concurrent activity all need to be considered

Summary

- The compiler provides the first level of optimization
- There are some straightforward steps you can take up front in your development which will save you time
- Blackfin Processors have lots of features that help you achieve your desired performance level
- Thank you