An Implementation of Hardware Accelerator using Dynamically Reconfigurable Architecture

Takashi Yoshikawa, Yutaka Yamada and Shigehiro Asano

Toshiba R&D Center

Outline

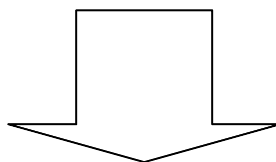
- Motivation
- Architecture overview
- Application example (H.264 decode)
- Evaluation
  - Performance
  - Area consumption
- Conclusion

Motivation

- Need better solution for multimedia applications
  - Processor solution is not efficient
    - Lots of data preparation before actual operations
      - Data alignment, Shuffle, Shift
    - SIMD operations don't always fit multimedia applications
      - Sometimes an instruction requires two or more operations (ex. ++--++--) to maximize its efficiency
    - Difficult to add newly defined instructions to an existing ISA for supporting these operations
      - 100, 200, or even more instructions needs to be added
  - Hardware engine is efficient enough but not flexible
    - Needs a new design for each application
    - Not easy to fix bugs

Motivation (Cont'd)

- Reconfigurable logic may solve these problems but….
  - Utilization of logic/network must be high enough
  - Overhead of loading configuration must be covered with processing time

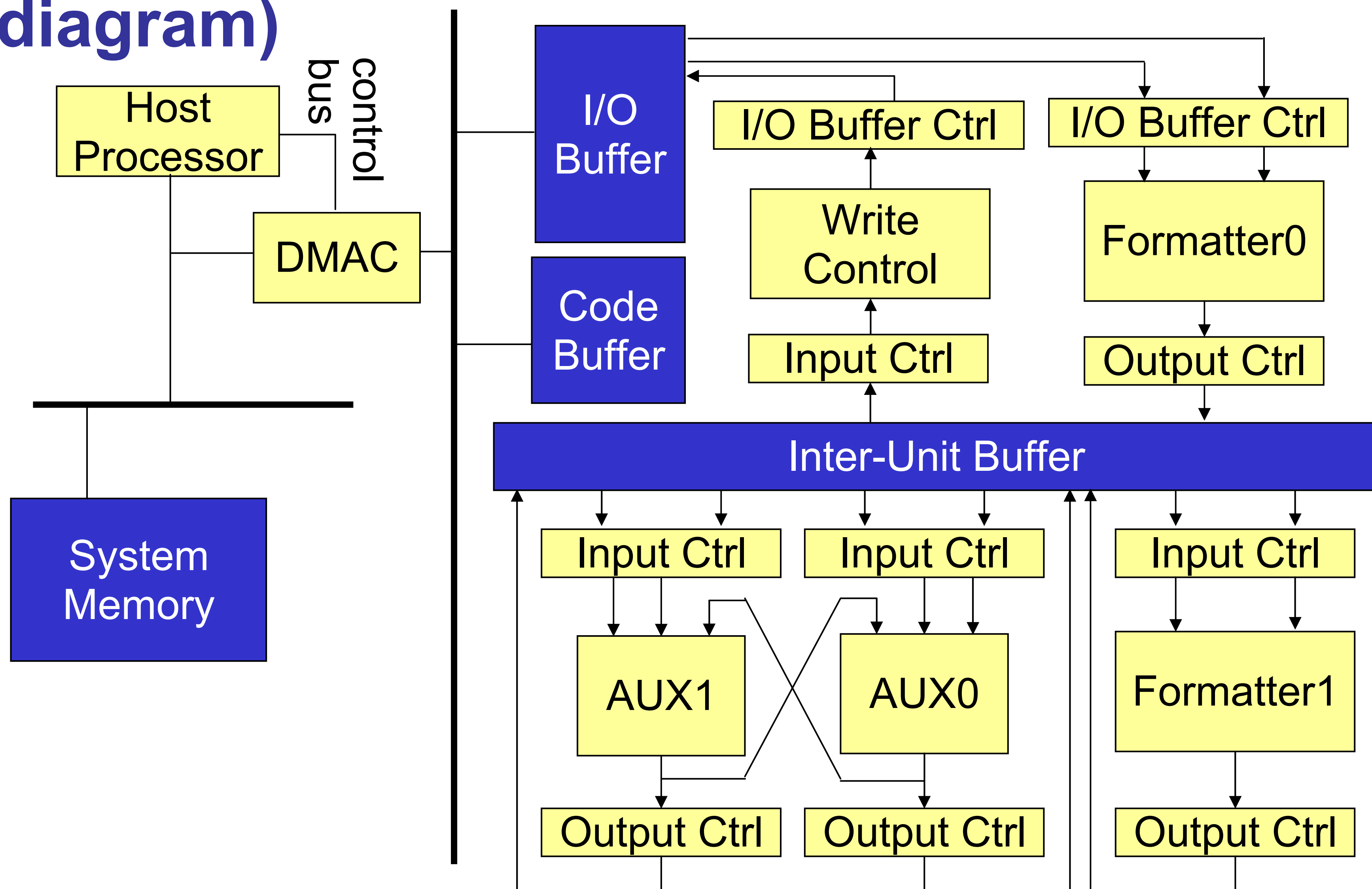


- We propose dynamically reconfigurable HW engine optimized for multimedia applications

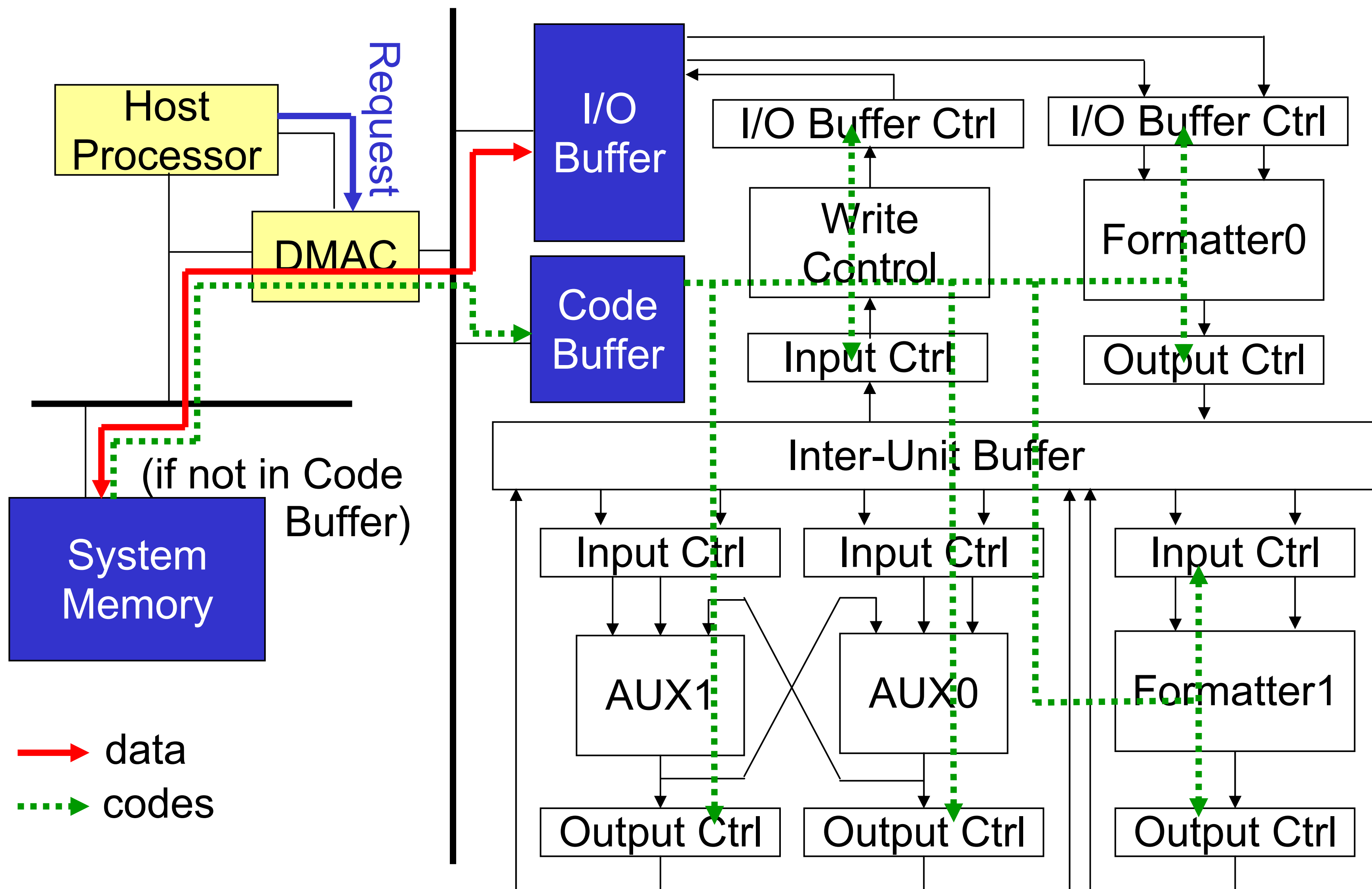
Outline

- Motivation
- Architecture overview
- Application example (H.264 decode)
- Evaluation
  - Performance
  - Area consumption
- Conclusion

Architecture overview (entire block diagram)



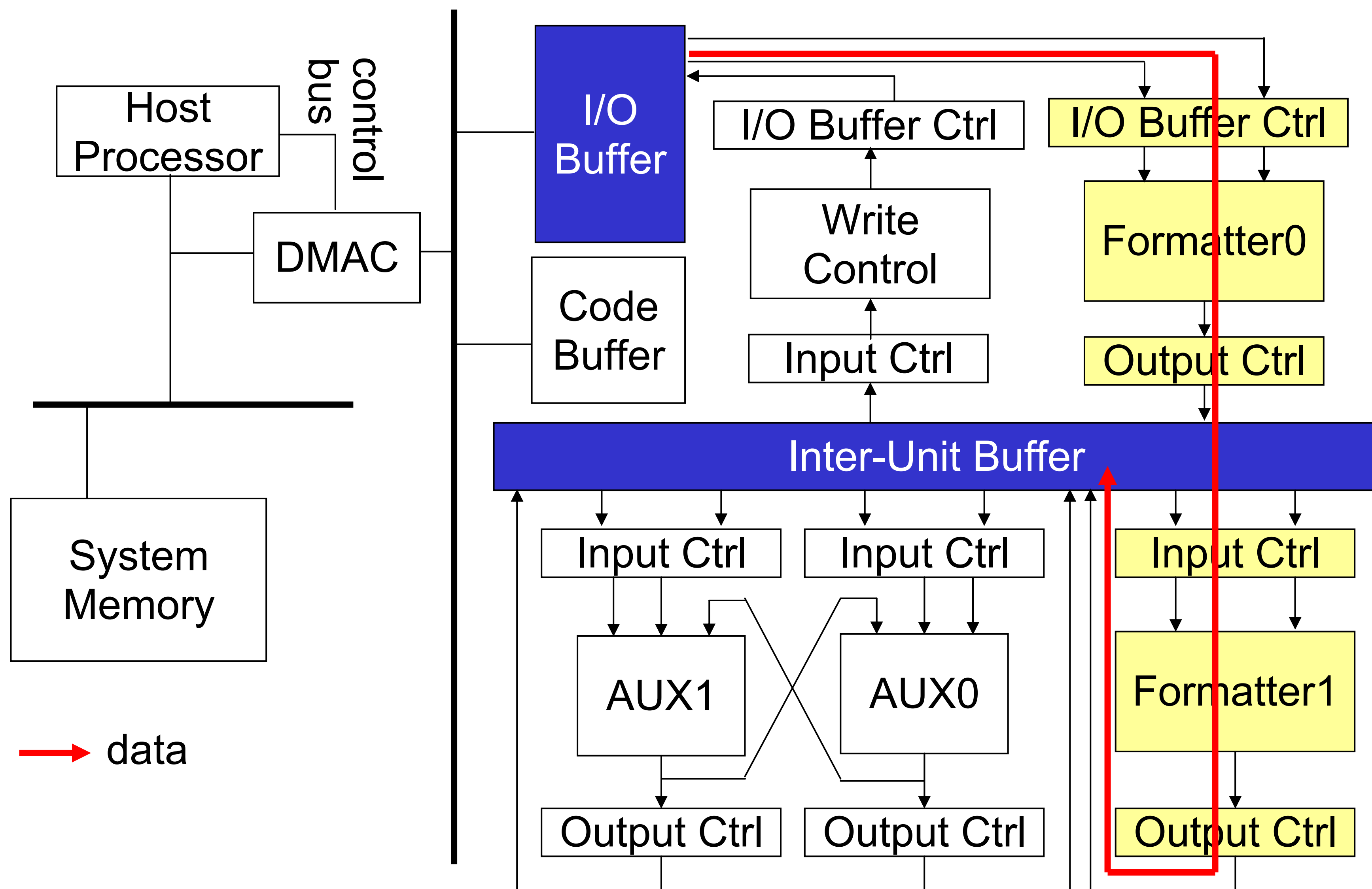
Architecture overview (data/code transfer)



Architecture overview (data/code transfer)

- DMAC controls the transfer of data/codes between a system memory and I/O Buffer/Code Buffer
  - The code consists of the configuration bits and the control code
    - The control code specifies which configuration bits should be applied to the reconfigurable units at any given cycle
- DMAC also initiates code transfer from Code Buffer to the reconfigurable units
  - Reconfigurable units: Formatters, AUXs and Write Control Unit
- A host processor issues a command with a system memory address to DMAC as a request
  - Issued through the control bus

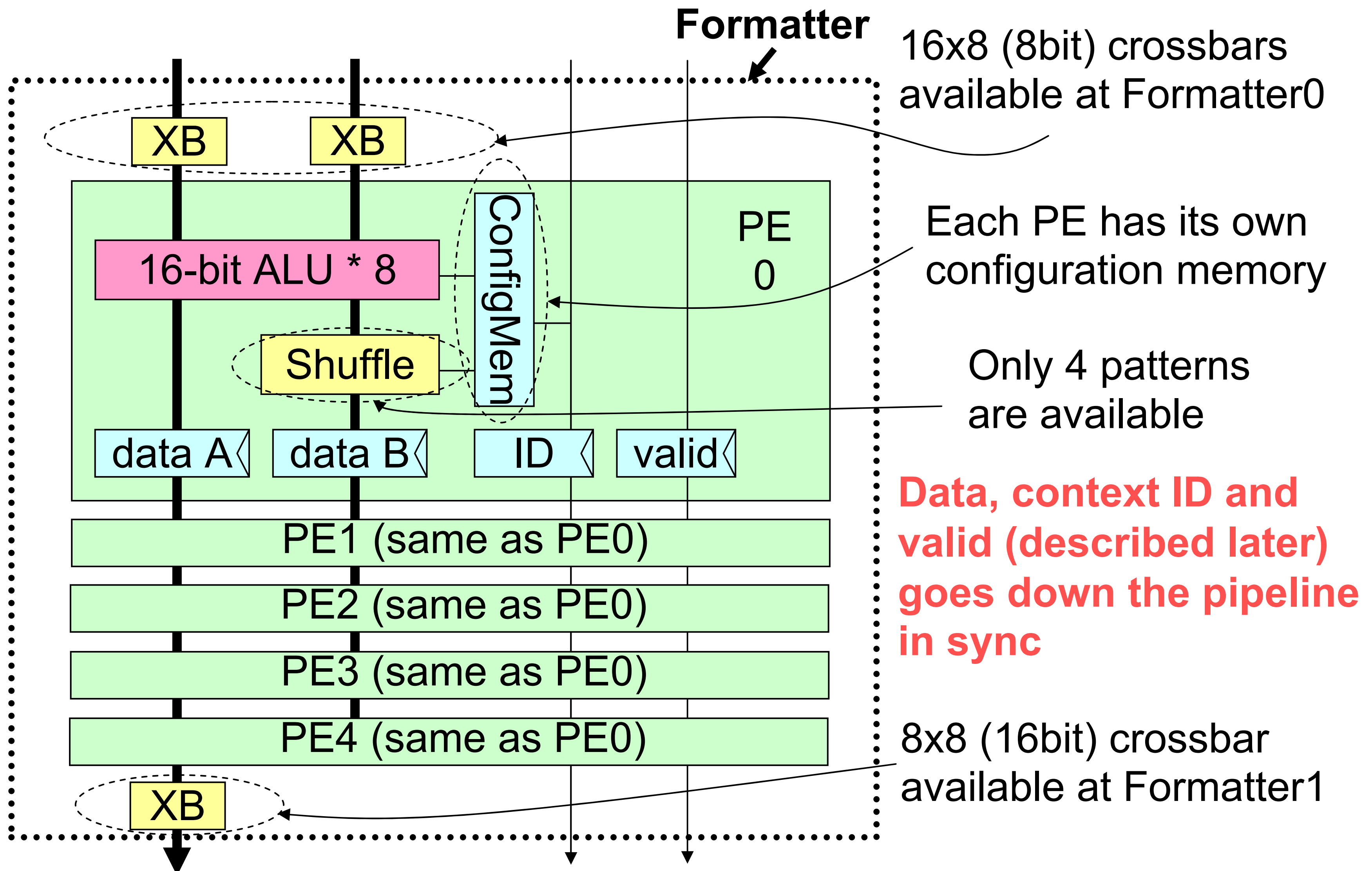
Architecture overview (Formatter)



Architecture overview (Formatter)

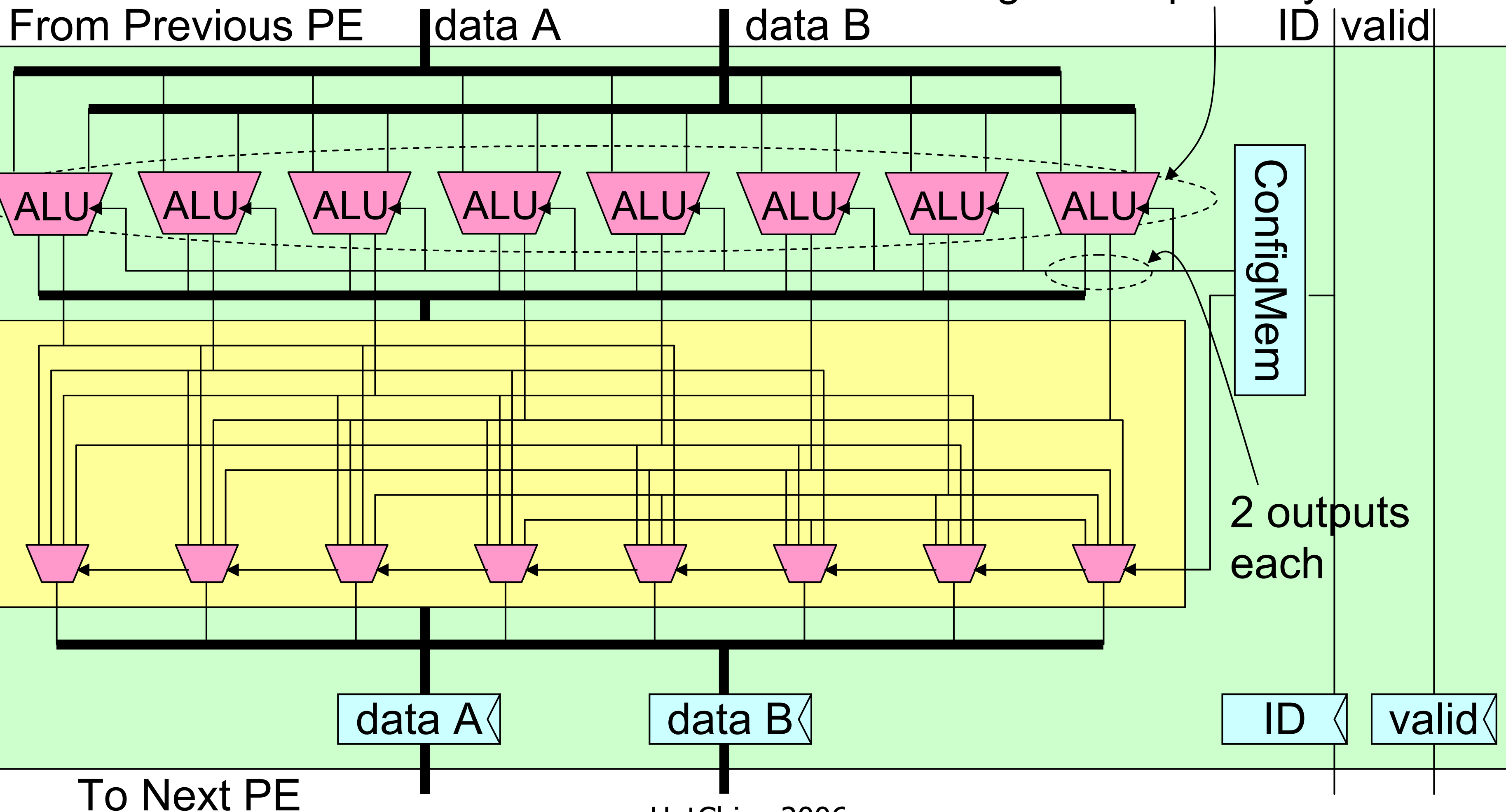
- Formatter Units
  - Each consists of Input Control, Output Control, full crossbar switches and 5 stages of processing element (PE)
    - Next two slides describe PE in detail
    - Input Control reads data from I/O buffer (or Inter Unit Buffer) and send the data along with a Context ID
      - Context ID is a pointer to one of the configurations of PE
  - Output Control receives the data from the last stage of PE and writes it into Inter Unit Buffer

Architecture overview (Formatter)

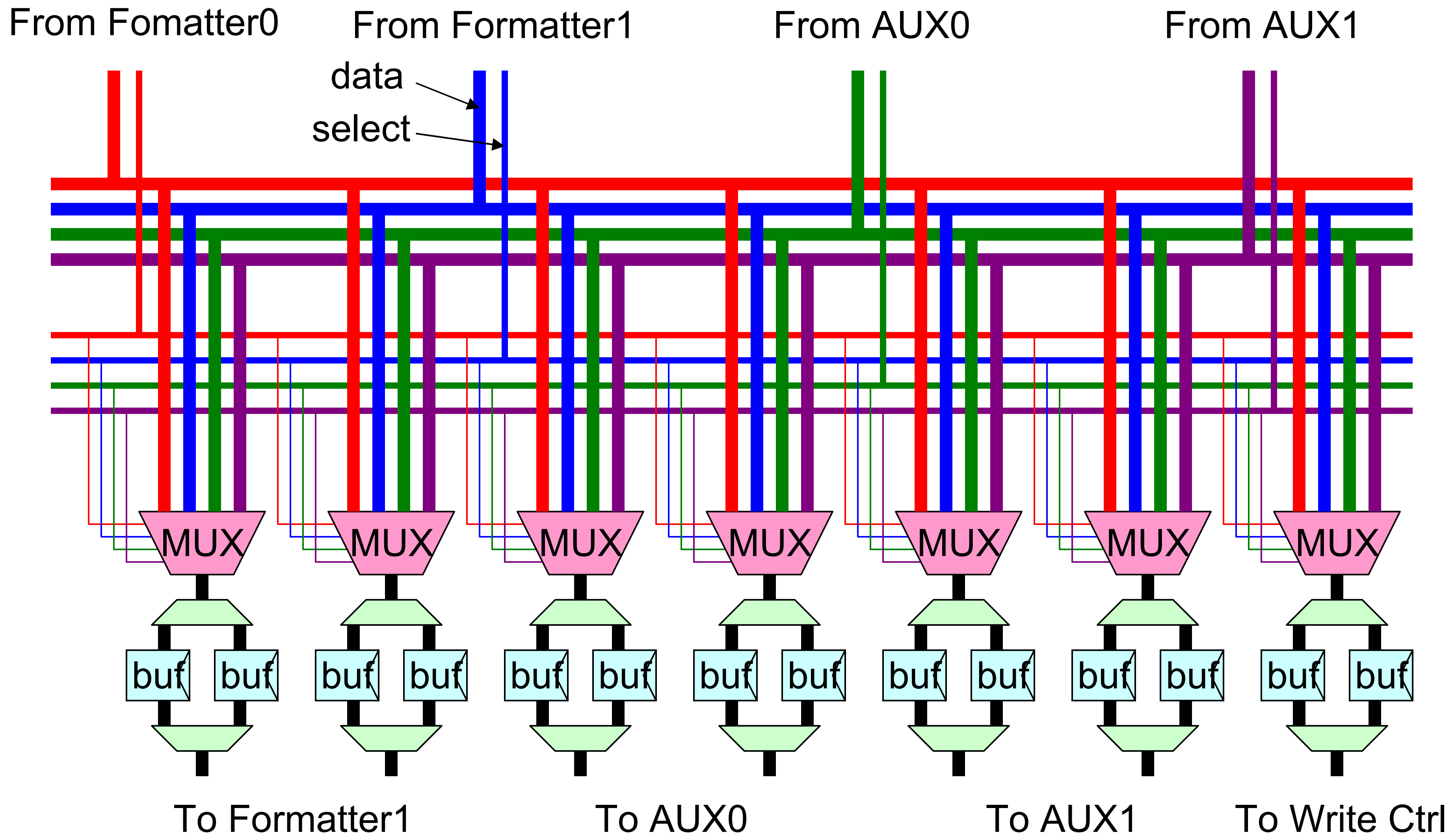


Architecture overview (Processing Element)

- Processing Element (PE)



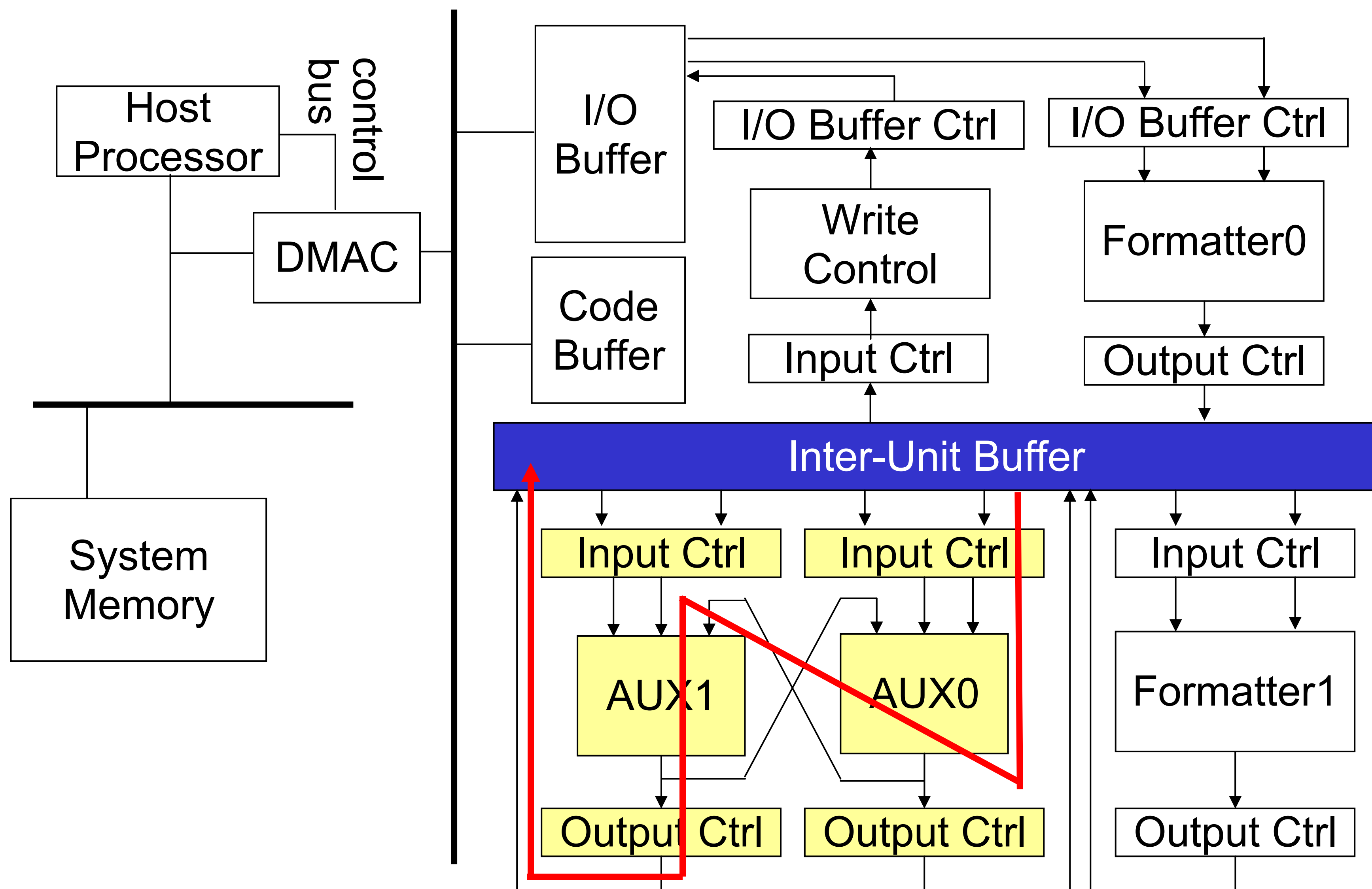
Architecture overview (Inter-Unit Buffer)



Architecture overview (Inter-Unit Buffer)

- Inter-Unit Buffer (IUB)
  - Consists of 14 data buffers (size:128bit each)
  - Formatters and AUX write output data into one of these buffers
    - The buffer ID is associated with a context ID from those units
  - Conflict of buffer writes has been statically avoided by the control code
    - All the timing of buffer writes is deterministic
  - Each buffer has a Valid Bit
    - Described in detail in the application example

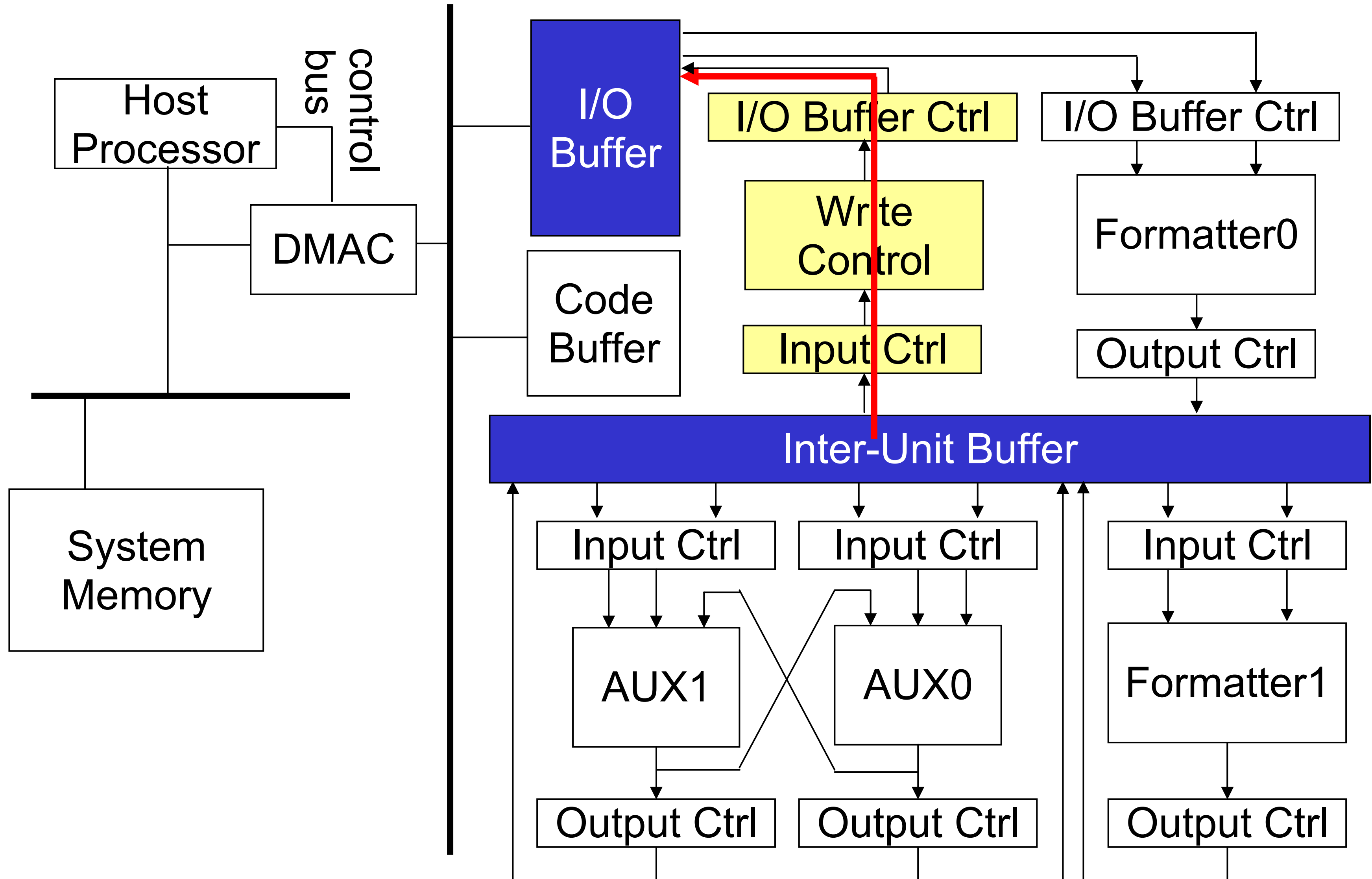
Architecture overview (AUX units)



Architecture overview (AUX units)

- AUX (Auxiliary) Units
  - Perform operations which can't be handled by Formatter
    - Multiplication, 32bit operations, etc.
  - Each AUX unit consists of Input Control, Output Control and single reconfigurable SIMD unit
    - SIMD unit consists of eight 32-bit integer units
      - Multiplier is 16-bit wide
      - Result is held in a 32x8 bit accumulator
    - All of the operation units share the same configuration
    - One AUX can transfer its output data to the other AUX directly
      - Inter-Unit Buffer may also be used if necessary

Architecture overview (Write Control Unit)

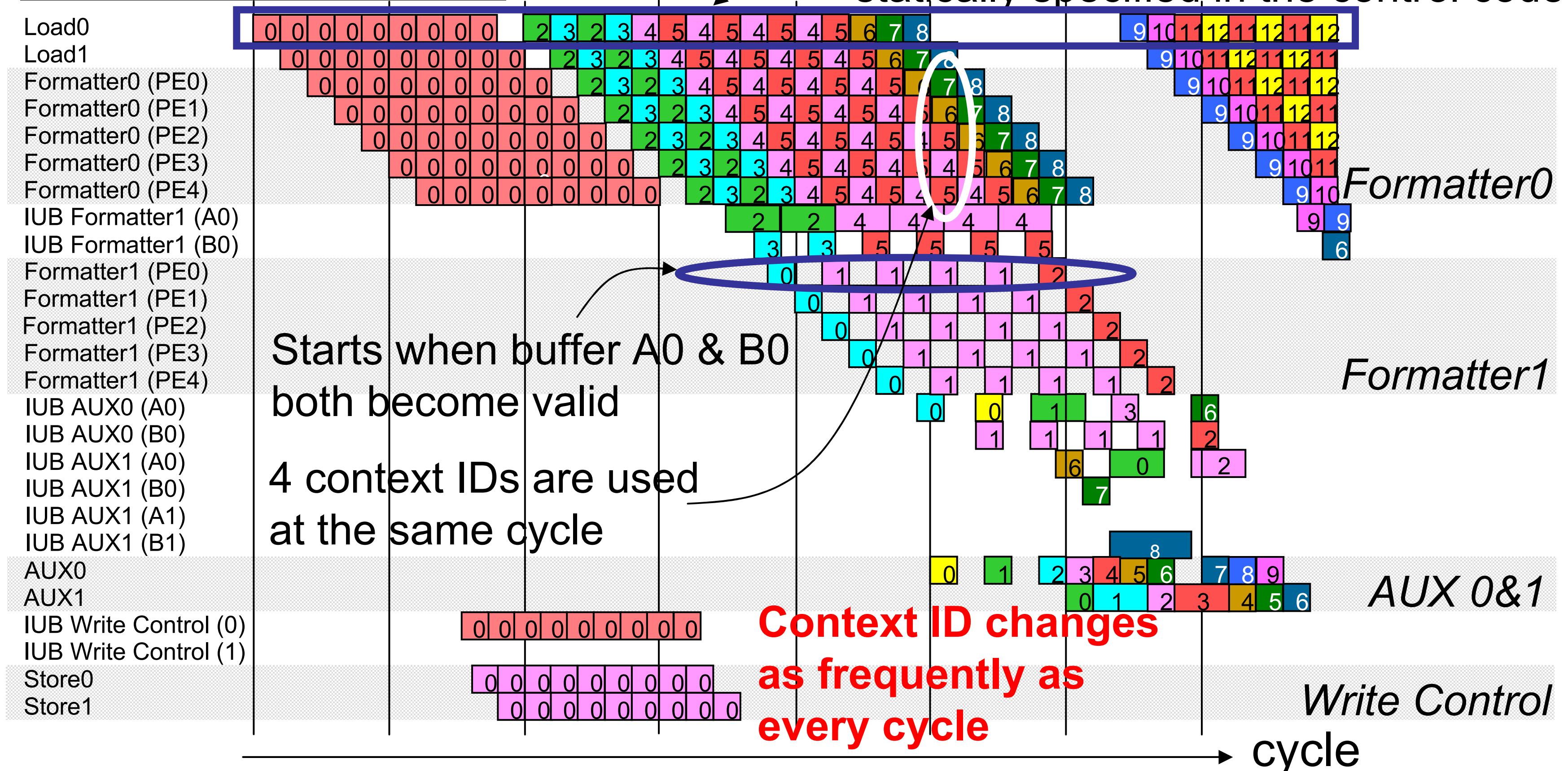


Architecture overview (Write Control Unit)

- Write Control Unit
  - Reads 128bit data from Inter-Unit Buffer and writes it into I/O buffer
    - Data shuffle available (8x8 [16bit] full crossbar)
  - Context IDs are issued in the same manner as Input Control of Formatter/AUX
  - The Context ID is associated with:
    - The buffer containing write data
    - 128bit-aligned write address
      - The address is given in conjunction with the control code
    - Byte enable (for partial write)
    - Configuration of a crossbar switch

Architecture overview (Timing Chart)

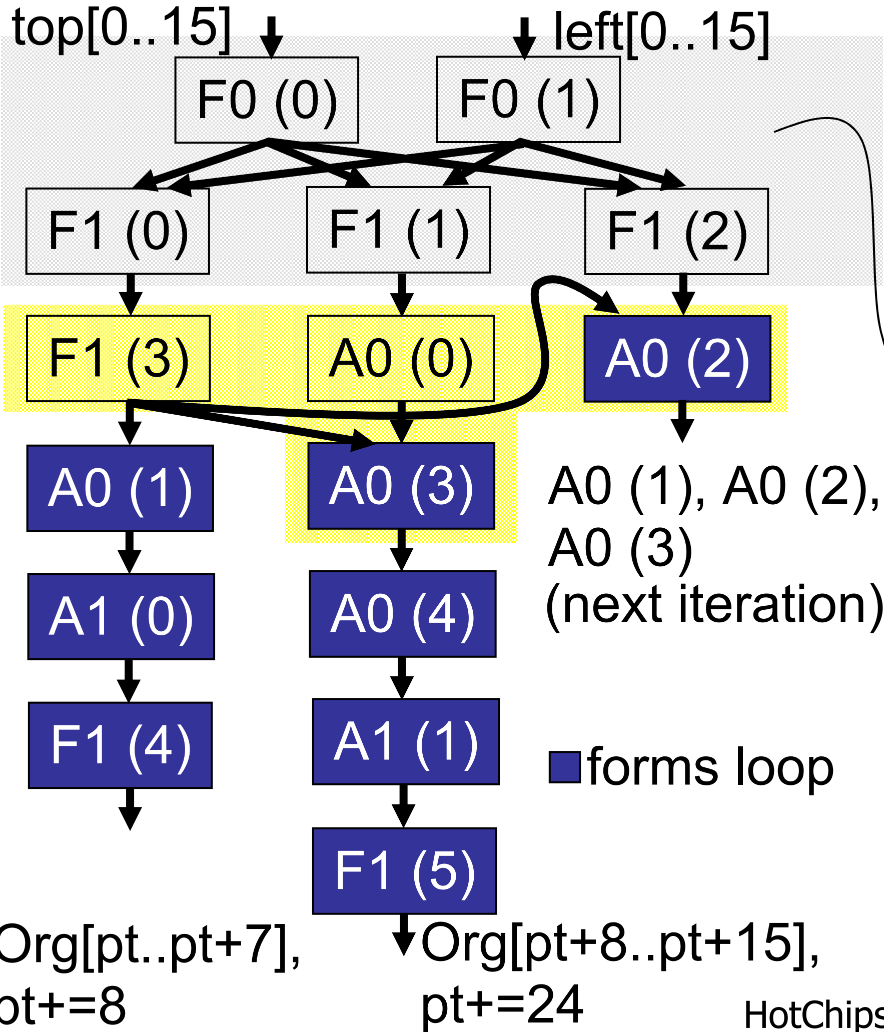
- Timing chart example



Outline

- Motivation
- Architecture overview
- Application example (H.264 decode)
- Evaluation
  - Performance
  - Area consumption
- Conclusion

Application example (H.264 decode: intra16x16 prediction [plane])



A0 (1), A0 (2), A0 (3) (next iteration)

■forms loop

Original Code

```
unsigned char Org[768]
void intra16x16_plane_prediction(
  uchar *left, uchar *top) {
  int H,V,a,b,c,t,i,pt=132,p;
  V=H=-top[7]*8;
  for (b=1;b<8;b++)
  { H+=-top[7-b]*b+top[7+b]*b;
    V+=-left[7-b]*b+left[7+b]*b; }
  H+=top[15]*8; V+=left[15]*8;
  a=(top[15]+left[15])<<4;
  b=(5*H+32)>>6; c=(5*V+32)>>6;
  t=a–7*(b+c)+16;
  for (i = 0; i < 256; i++)
  { p=t>>5; t+=b;
    Org[pt++]=(p<0)?0:(p>255)?255:p;
    if ((i & 15) == 15)
    { t+=c–b*16; pt+=16; } }
}
```

Application example (operation graph)

ALU & full crossbar/shuffle configuration for Formatter0 context 0&1

A=top[0..15] B=top[0..15]

0123456789ABCDEF

A=F89ABCDE
B=76543210

Full Crossbar (16x8)

sub sub sub sub sub sub sub sub

Shuffle pattern 1
01234567 -> 10325476

x8 fwd add fwd add fwd add fwd

Shuffle pattern 2
01234567 -> 10325476

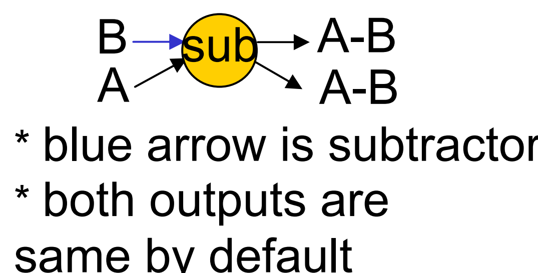
add fwd x2 fwd add fwd x2 add

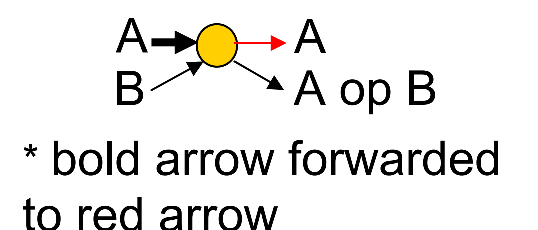
Shuffle pattern 1

add fwd fwd fwd x2 fwd add fwd

Shuffle pattern 2

add fwd add fwd add fwd add fwd



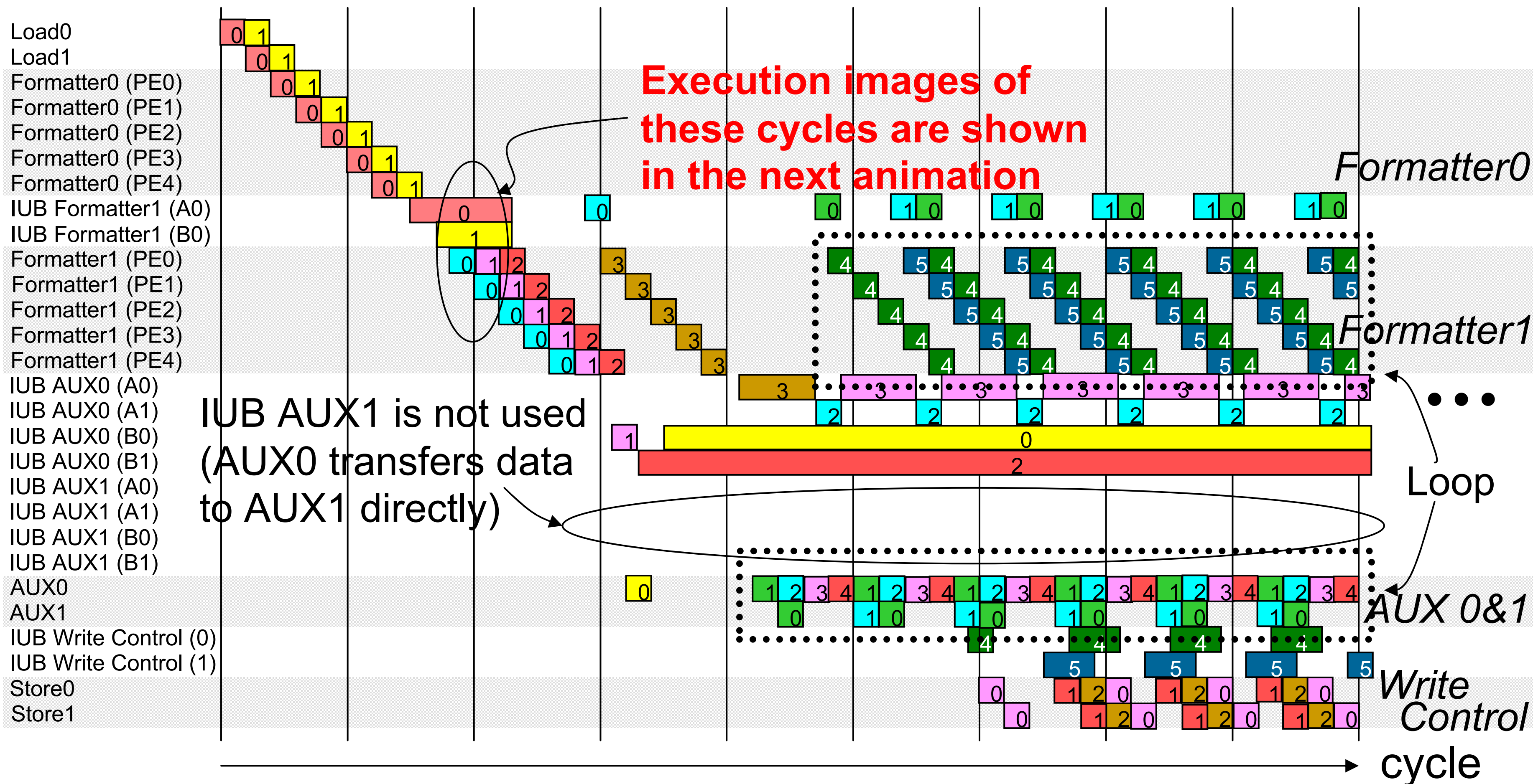


◯ unused operation (unused data paths are not shown)

TOSHIBA

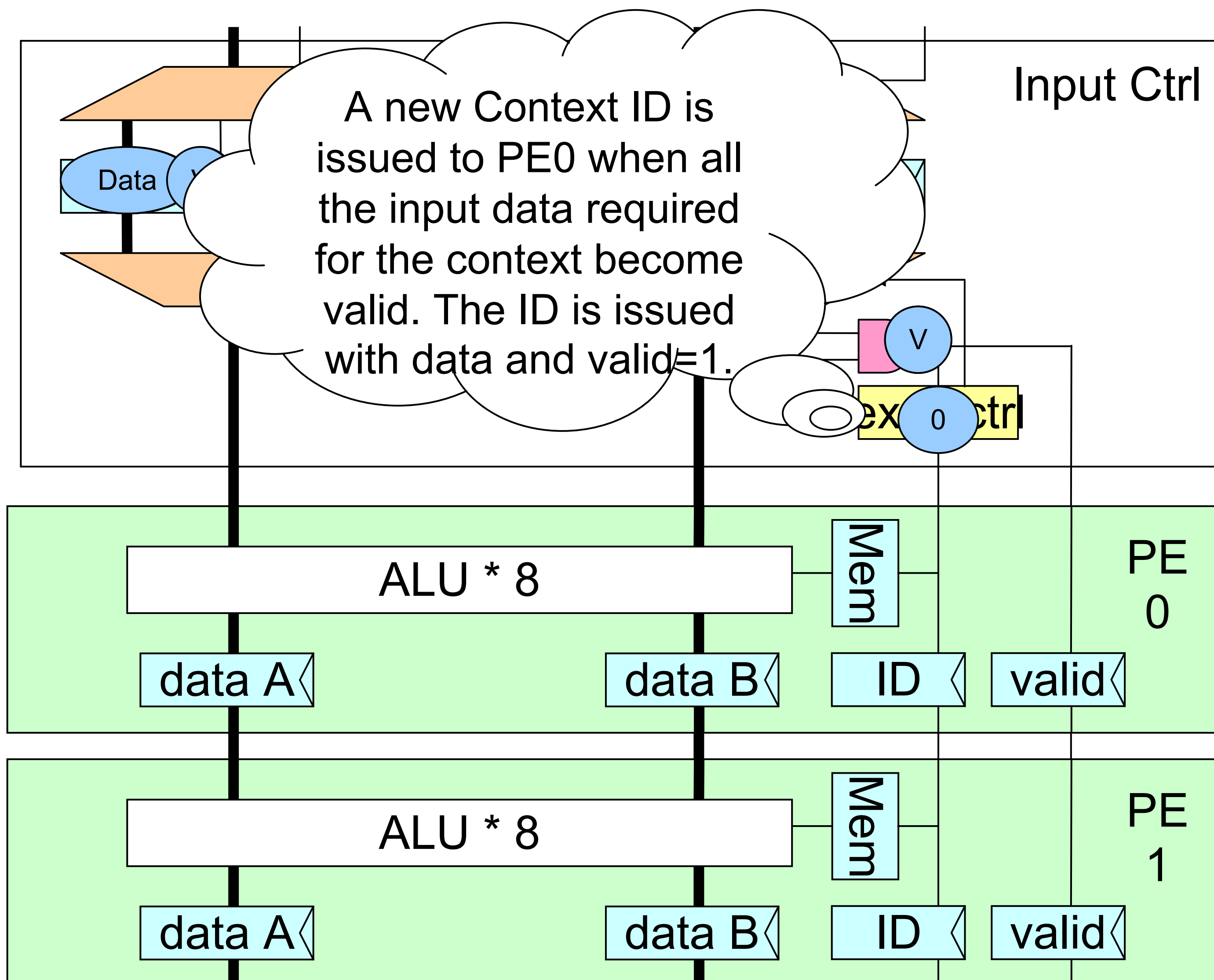
Application example (Timing Chart)

- Timing Chart (96 cycles in total)

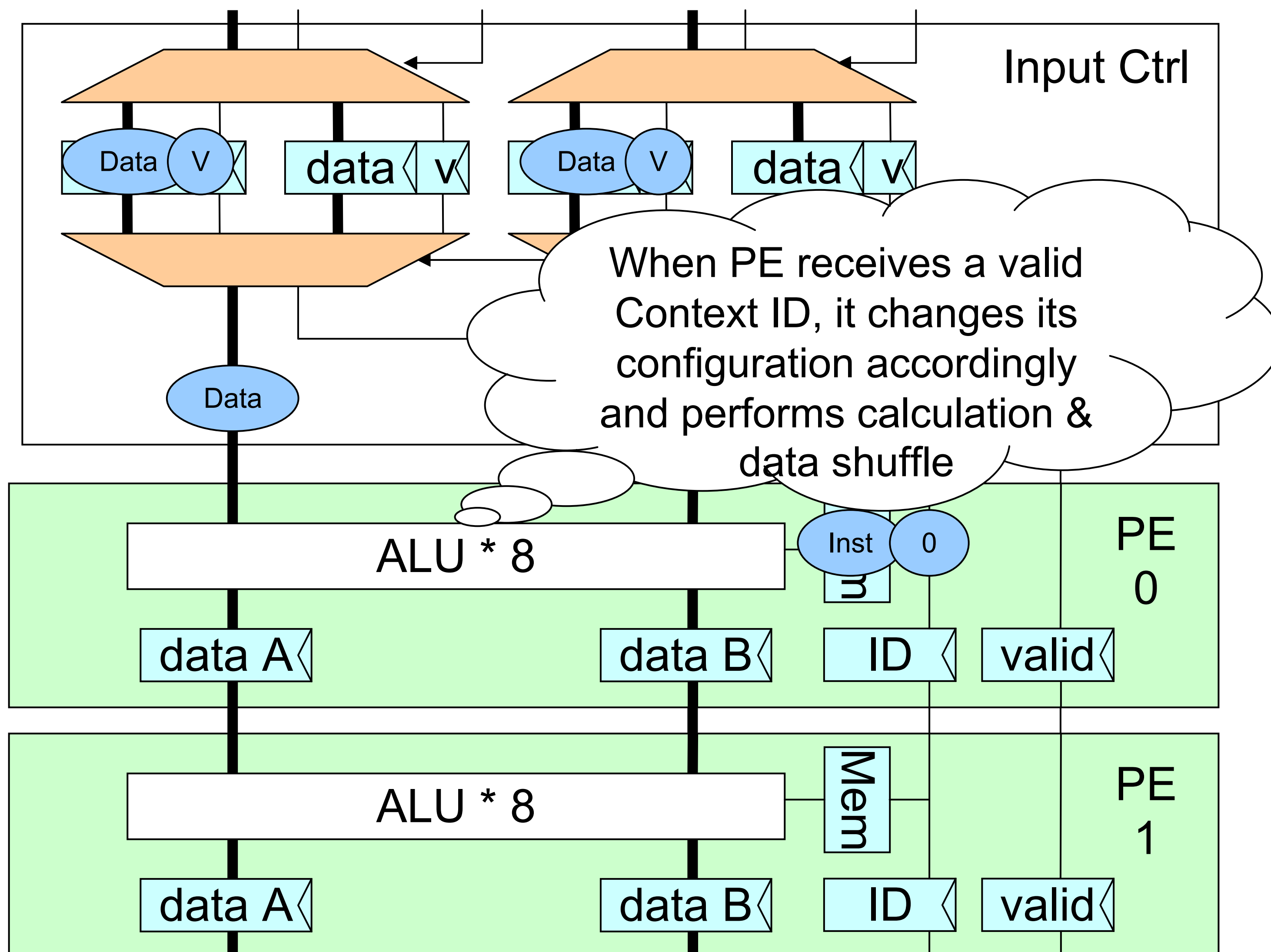


TOSHIBA

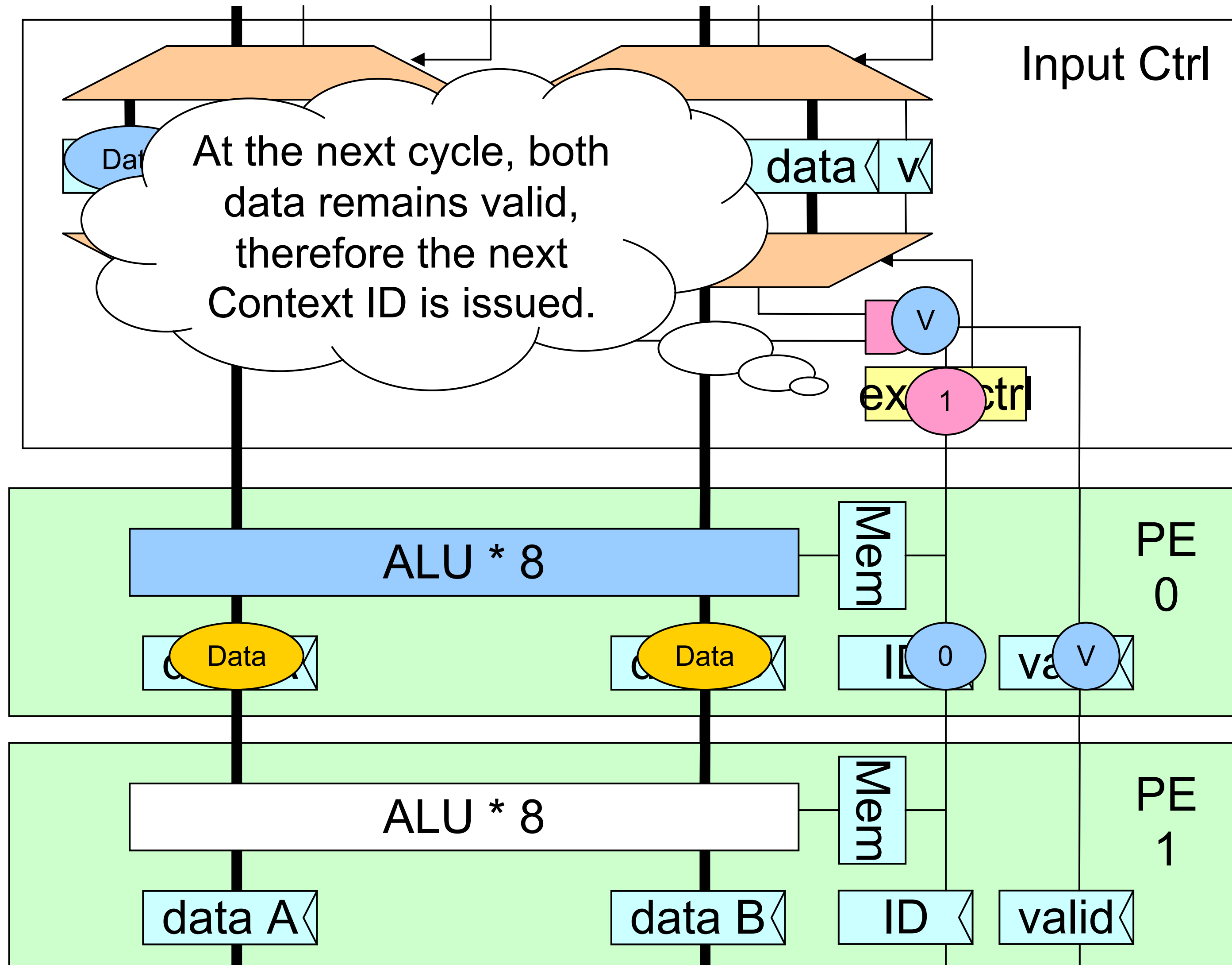
Application example (Execution Image)



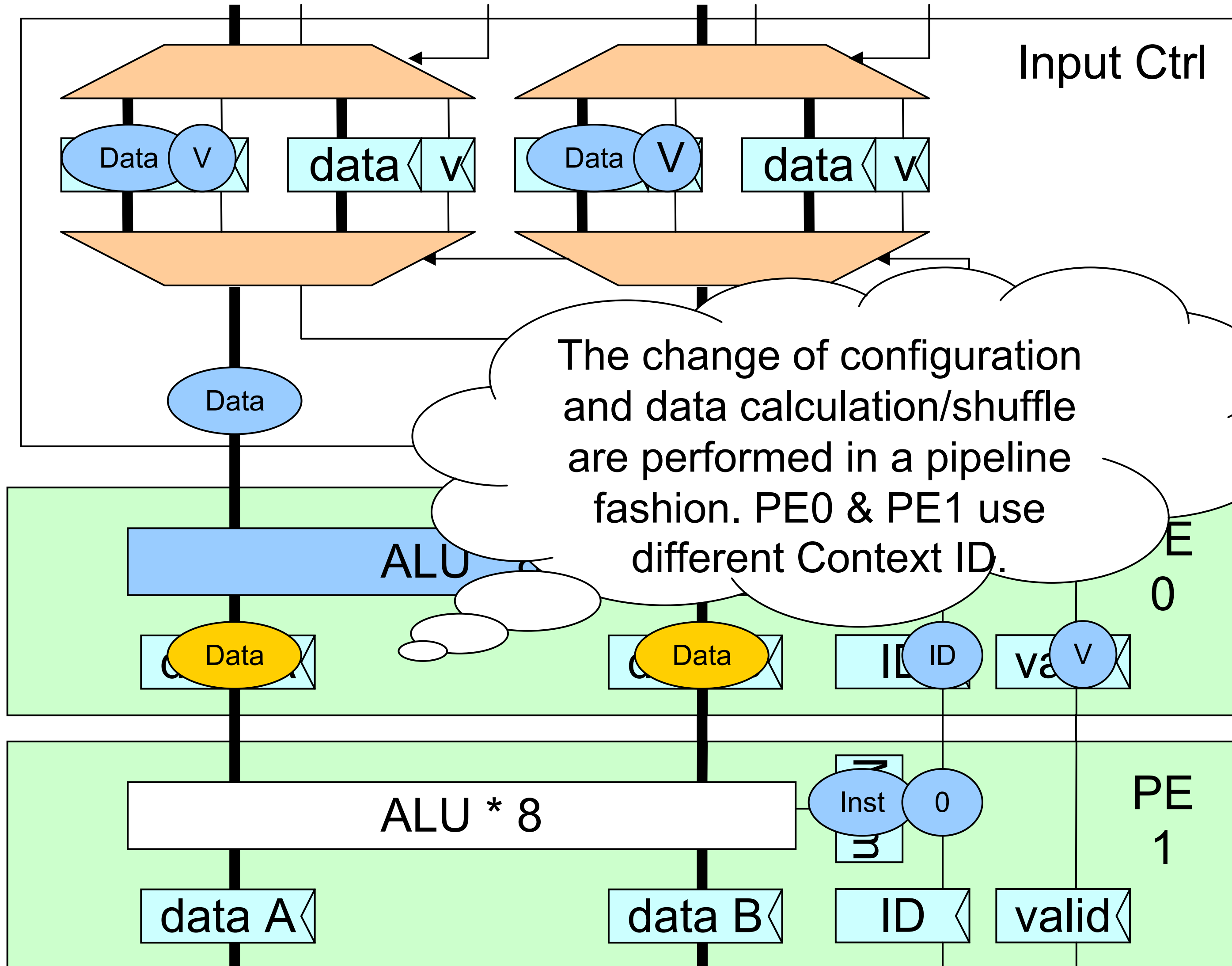
Application example (Execution Image)



Application example (Execution Image)



Application example (Execution Image)



Outline

- Motivation
- Architecture overview
- Application example (H.264 decode)
- Evaluation
  - Performance
  - Area consumption
- Conclusion

Evaluation (Performance)

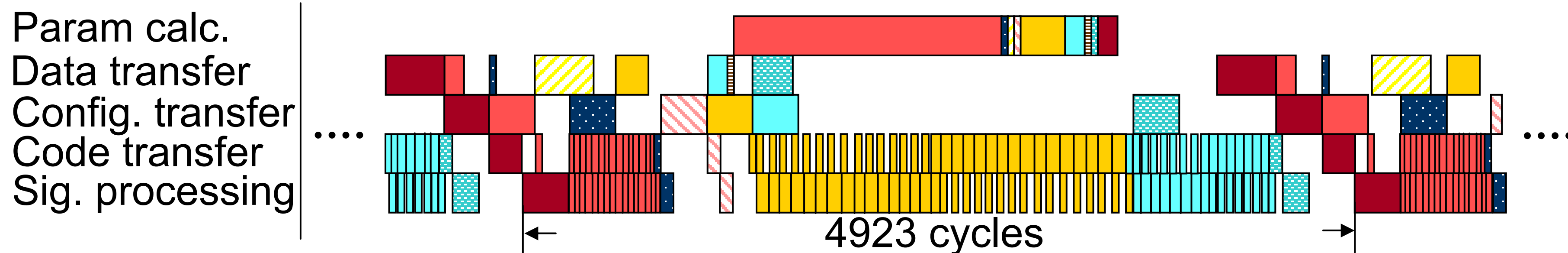
- Application: H.264 decode
  - Estimated number of cycles required for decoding one macroblock

Name	Cycles	Comment
Our Accelerator (ideal)	3792 (intra) 5705 (inter)	ideal cycles expected for decoding the most demanding macroblock
Our Accelerator (real)	4923 (intra) 7108 (inter)	Actual cycles required for decoding the most demanding macroblock
Our Accelerator (typical)	3660 (average)	When 2Mbps VGA stream is decoded
PowerPC G5 @2GHz	16576 (average)	dual G5@2Ghz recommended by Apple for 8Mbps, 24fps HD stream

More than 4 times faster than PowerPC G5 in terms of number of cycles required

Evaluation (Performance Breakdown)

- Timing chart (real, intra)



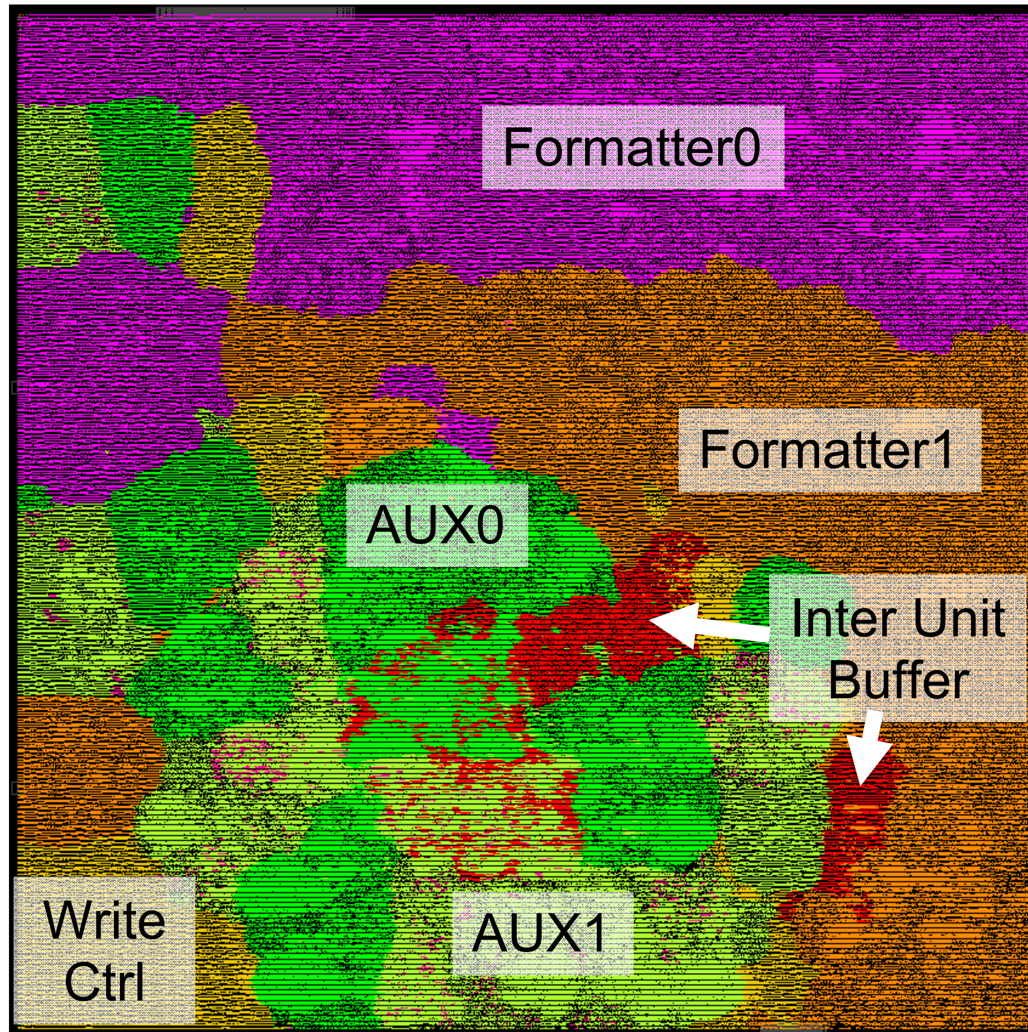
- 1100 more cycles are required than the ideal cycles
  - The reason: configuration transfer cannot be completely hidden by signal processing
    - It can be hidden if the number of configuration memories increases (currently two buffers)

Evaluation (Area consumption)

- Area (logic gate) consumption

	Logic [gate size]	F/F Memory [gate size]	Memory Size [bit]
Formatter (x2)	87,500	31,000	3,136
- Processing Element (x5 in Form.)	18,700	7,500	656
- ALU (x8 in PE)	1,700	-	-
- Shuffle (x1 in PE)	1,000	-	-
- Buffer (x1 in PE)	2,200	-	-
Micro Controller (x5)	800	12,000	1,152
Total Inter Unit Ctrl. (Input & Output)	20,000	136,000	13,696
Crossbar (Formatter0)	4,000	22,000	2,112
Crossbar (Formatter1, Write Ctrl Unit)	3,000	7,600	768
AUX (x2)	45,000	9,500	960
I/O Buffer Interface (Read & Write)	6,000	7,600	800
Inter Unit Buffer	20,000	-	-
Total	319,000	314,200	31,584

Evaluation (Chip layout)



Outline

- Motivation
- Architecture overview
- Application example (H.264 decode)
- Evaluation
  - Performance
  - Area consumption
- Conclusion

Conclusion

- Presented an implementation of hardware accelerator using dynamically reconfigurable architecture
  - Efficient signal processing can be achieved by reconfiguring ALUs and crossbars dynamically
    - More efficient than a general SIMD unit because each crossbar/16bit-ALU can be reconfigured differently
  - Not limited to the acceleration of H.264 decode
    - Can also accelerate H.264 encode and other codecs by applying different configurations

Conclusion (cont'd)

- Performance
  - Expects 30 frames/sec with 3 accelerators running at 300MHz for decoding Full HD H.264 video
    - More than 4 times faster than PowerPC G5 in terms of number of cycles required for decoding one macroblock
- Area consumption (number of logic gates)
  - 319K Gates (excluding memory)
    - Small enough considering its performance
    - Size of buffer memory will be 24KB or less

Backup Slides

Evaluation (Comparison with PowerPC G5)

- Application: H.264 decoder
  - Full HD frame (1920x1080 pixels) consists of 8100 macroblocks)
    - one macroblock = 16x16 pixels
  - To achieve frame rate = 24 Full HD frame/sec, one macroblock must be decoded in 5.14 sec
    - Requires dual 2.0GHz Power Mac G5 (Apple's recommendation)
      - decoded w/ Quicktime7.1 (AltiVec used)
      - 5.14 sec = 10288 cycles at 2.0GHz

Evaluation (Function level performance results)

Name	Cycles	Comment
IQ/IDCT	221	204 if intra16x16 prediction is not used
Intra prediction (luma/chroma)	~ 448 / 75	motion compensation (luma) included
Motion compensation	133	69 if motion compensation (luma) excluded
Inter prediction (luma/chroma)	~ 1579/~163	Worst case (6/4-tap filter applied max. times)
Deblocking filter luma	~ 2112	Worst case (filter applied maximum times)
Deblocking filter chroma	~ 784	Worst case (filter applied maximum times)
Store macroblock	100	