MPEG-4 AVC - H.264 CABAC / CAVLC IP Datasheet



SUMMARY

1 Introduction..... 3

1.1 H.264 Overview.....3

1.2 IP overview.....4

1.3 Commercial References..... 5

2 Features..... 5

3 Performances..... 6

4 Resources..... 6

5 Architecture..... 7

5.1 Principle..... 7

5.2 Hardware interface.....8

5.3 Software interface..... 9

5.4 System integration..... 10

5.4.1 Integration with CPU bus.....10

5.4.2 Integration with AHB bus.....10

5.5 Operating mode..... 11

6 Deliverables..... 12

1 Introduction

1.1 H.264 Overview

The H.264 is also known as MPEG-4 ISO/IEC14496-10 or MPEG-4 / AVC. This standard has been co-developed by JVT group composed by MPEG-ISO/IEC members and VCEG-ITU-T members.

Three profiles have first been defined, each with several levels. A High Profile has been added, and standardisation work is going on.

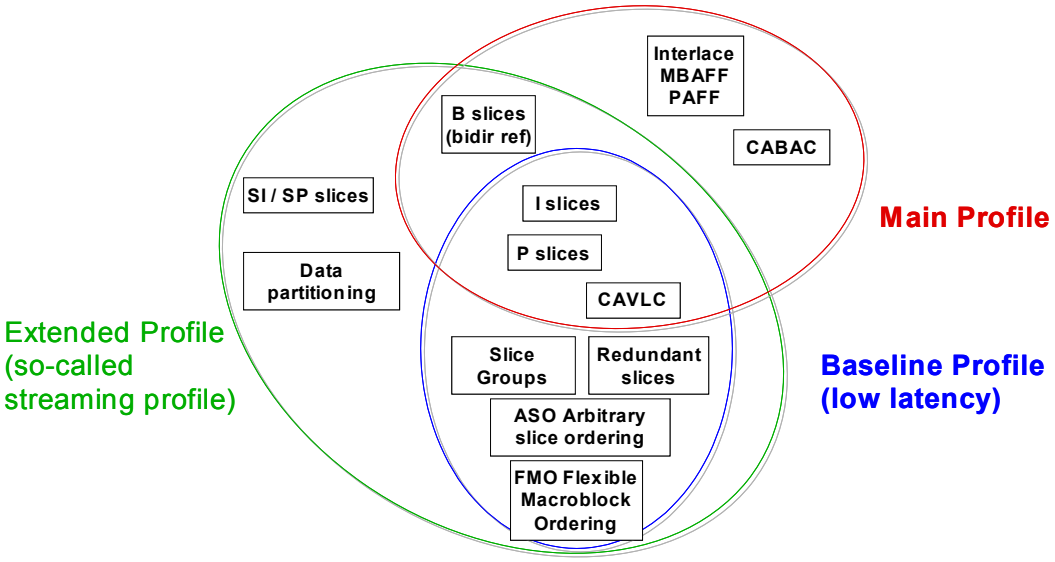

Figure 1- Profiles of MPEG-4 part 10 / AVC H.264

Ateme has developed a full Main Profile Level 1-4 implementation of H.264 encoder and decoder, ensuring Baseline, Extended and High Profiles partial compatibility.

The encoder is available as:

- Software library for x86 (PC),
- VHDL firmware, for embedded equipements.

The decoder is available as software library for x86 and for DSP.

The synoptic for a full encoder is shown below, with the Bitstream encoding highlighted.

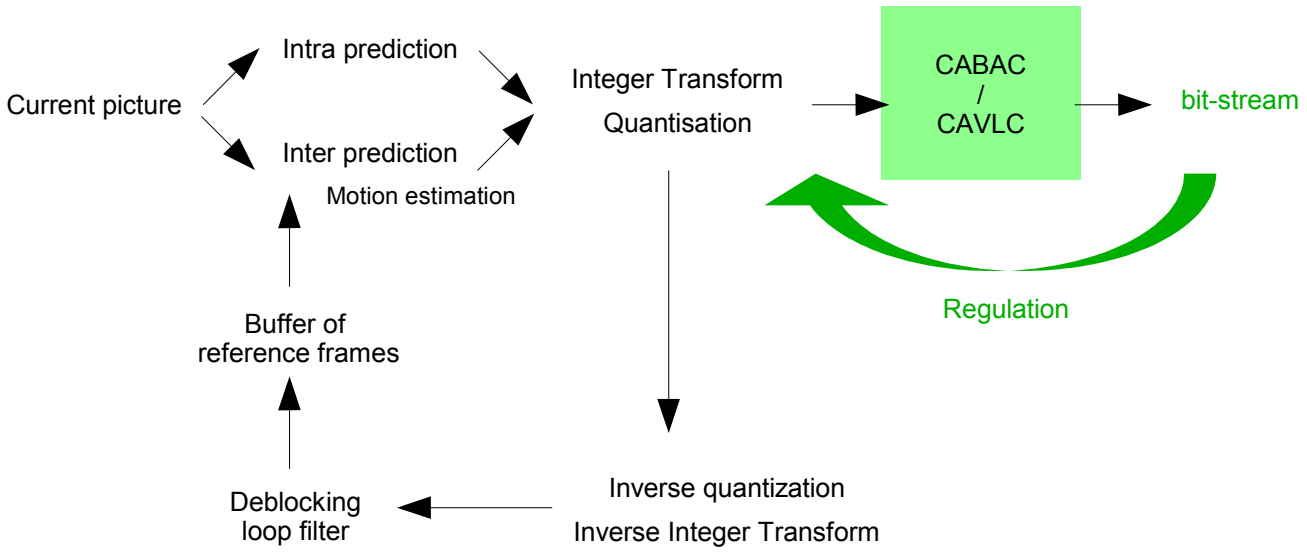

Figure 2- synoptic of MPEG-4 part 10 / AVC H.264 encoder

A video sequence is composed of images. In the H.264 process, an image is split in macroblocks (MB). A macroblock is a 16x16 pixels luminance block and two 8x8 pixels chrominance blocks.

A slice is a collection of macroblocks belonging to a single image.

The H.264 bitstream is structured in NAL units. Each NAL unit contains the information of one slice.

1.2 IP overview

Ateme's H.264 bitstream IP is a hardware module able to encode the macroblocks information using CABAC, CAVLC and/or PCM coding mode, and to generate H.264 bitstream compliant with ISO/IEC14496-10 specification.

NAL and RBSP (a subsection of the NAL unit) headers are not generated, but included as raw data in the bitstream.

The neighborhood information necessary for encoding is autonomously managed.

The module is designed to compute full size video sequences. It can be adapted to any format.

1.3 Commercial References

The Bitstream IP is available under the following references:

	CAVLC	***CABAC***	***PCM***	***SD – levels 1-3***	***HD – levels 1-4***
H264_SD-CAVLC	X		X	X	
H264_SD-CABAC		X	X	X	
H264_SD-Bitstream	X	X	X	X	
H264_HD-CAVLC	X		X	X	X
H264_HD-CABAC		X	X	X	X
H264_HD-Bitstream	X	X	X	X	X

SD stands for Standard Definition, or Full D1, or 720x480 30 / 29.97 fps in NTSC and 720x576 25 fps in PAL.
This corresponds to a throughput of max 60750 Macroblocks per second.
Level 3 is up to 10 Mbps.

HD stands for High Definition, with 720p 50 / 60, 1080 i 50 / 60, 1080 p 25 / 30.
This corresponds to a throughput of max 367200 Macroblocks per second.

- HD 720p/25-30fps is level 3.1 - up to 14 Mbps
- HD 720p/50-60fps is level 3.2 - up to 20 Mbps
- HD 1080 i&p/25/30fps is level 4 up to 25 Mbps

Other products are available from Ateme:

- Motion Estimation IP
- Deblocking filter IP
- Full SD main profile Level 3 encoder IP,
  including video acquisition, decision, regulation, memory management, PCI, etc...

2 Features

- Platform independent design (written in VHDL)
- Full support of Main and Baseline profiles
- CABAC, CAVLC and PCM coding modes support
- MBAFF support (Adaptative Frame and Field)
- Non adjacent macroblocks support
- Neighborhood context autonomously managed
- PAL and NTSC support
- HD ready

3 Performances

High throughput performances are obtained with low clock frequency.

Clock frequency	***100 Mhz***	***150 Mhz***
CAVLC	33 Mbps	50 Mbps
CABAC	25 Mbps	37.5 Mbps

4 Resources

The IP has been optimized, thus for information on footprint and size, Logic Element number, and memory usage, please consult us for your specific target.

5 Architecture

5.1 Principle

The IP uses a generic bus interface for registers and streams.
It is delivered encapsulated with two AHB agents:

1. AHB Slave for registers access
2. AHB Master to access directly data in external memory

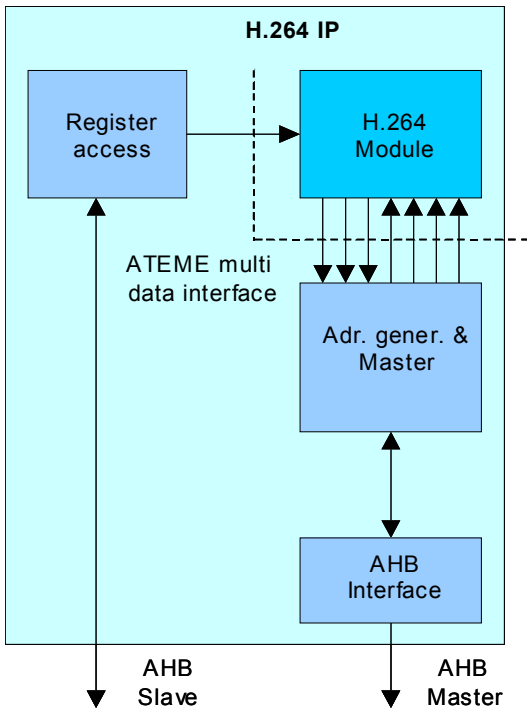

Figure 3- CPU or AHB bus interfaces

The three associated modules:

- Register access,
- Address generation and management of data accesses,
- Master AHB interface,

are delivered in source code with the H.264 module.
Thus the IP can be used either directly, for example by a CPU,
or with all associated modules in a SoC design.

5.2 Hardware interface

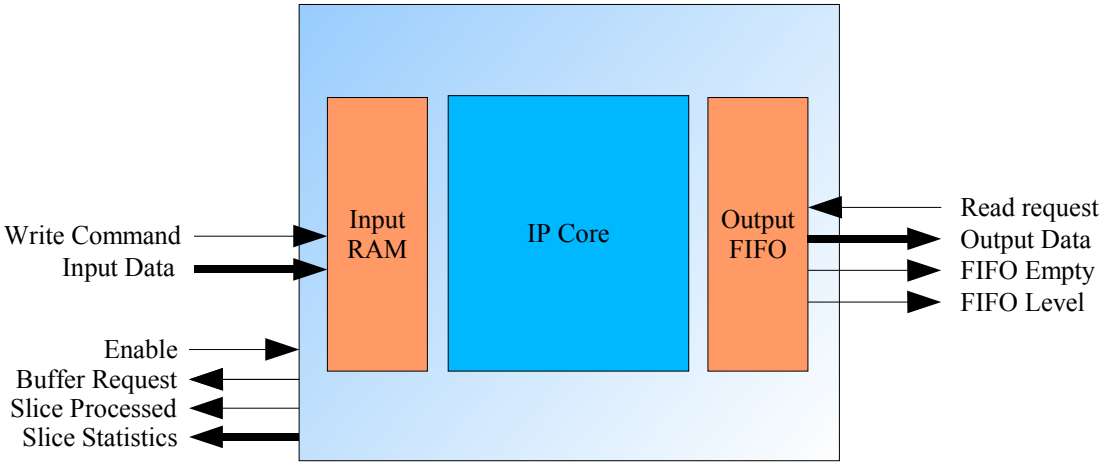

Figure 4- Hardware Interfaces

The IP uses a synchronous RAM at the input and a synchronous FIFO at the output.

The input RAM is managed as two buffers of 928 bytes. From a system point of view, it can be considered as a FIFO memory, since write address are managed by the core.

When a buffer request is emitted, a buffer must be filled (*see software interface*). When a slice is processed, an interrupt is asserted and statistics on the bitstream size becomes available.

The above table lists the IP ports.

Signal name	DIR.	Active	Description
Rstn	I	0	Reset signal
Enable	I	1	Enables processing and buffer requests to be emitted
Clk	I	↑	IP internal clock
RamInClk	I	↑	Input RAM clock
RamInWrite	I	1	Input RAM write command
RamInData	I	-	Input RAM data bus
FifoOutClk	I	↑	Output FIFO clock
FifoOutRead	I	1	Output FIFO read request
FifoOutEmpty	O	1	Output FIFO empty flag
FifoOutData	O	-	Output FIFO data bus
FifoOutLevel	O	-	Output FIFO level
BufferIT	O	0	Buffer request
SliceIT	O	0	Slice interrupt
CoeffBitsCount	O	-	Count of the bitstream bits which are the encoding result of coefficients
HeaderBitsCount	O	-	Count of the bitstream bits which are not the encoding result of coefficients, nor NAL or RBSP header bits
TotalBitsCount	O	-	Count of the total bitstream bits, except the NAL and RBSP header bits

BufferIT and SliceIT are active low pulses of 1 clock cycle.
The IP core and the memories have independent clocks, in order to ease integration.

5.3 Software interface

Little endian byte arrangement must be used.
Two structures of data are necessary:

- SliceInfo to describe the slice to work on,
- MbInfo to describe the macroblock to compute.

SliceInfo structure description

```
typedef struct {
Uint8    bQp;                          // pic_init_qp
Uint8    bFlags;                       // 0000 xxxx : slice_type
                                       // 00xx 0000 : cabac_init_idc
                                       // 0x00 0000 : isMbaff (1=Mbaff is used)
                                       // x000 0000 : entropy_coding_mode_flag
Uint16 wNumMb;                         // Macroblocks number in slice (might be 0)
Uint8    bNumRefIdxL0ActiveMinus1;     // num_ref_idx_l0_active_minus1
Uint8    bNumRefIdxL1ActiveMinus1;     // num_ref_idx_l1_active_minus1
Uint8    bNalUnitType;                 // 00x0 0000 : constrained_intra_pred_flag
                                       // 000x xxxx : Nal_unit_type
Uint8    bReserved;                    // 0, for 32b alignment
Uint16 wHeaderSize;                    // Header size in bits
Uint16 bSkipPrevention;                // Length in bytes where emulation prevention is not needed
Uint8    bHeader[wHeaderSize];         // Header data
} SliceInfo;
```

MbInfo structure description

```
typedef struct {
Uint8    bFlags;                       // 0000 xxxx coded_block_pattern (luma)
                                       // 00xx 0000 coded_block_pattern (chroma)
                                       // 0x00 0000 mb_field_decoding_flag
                                       // x000 0000 mb_skip_flag
Uint8    bType;                        // mb_type
Int8     sbDeltaQp;                    // slice_qp_delta
Uint8    bPredModeC;                   // intra_chroma_pred_mode
Uint4* bPredModeL[16];                 // prediction mode of the 16 luma cells.
Uint16 wPosX;                          // macroblock abscissa (in macroblocks)
Uint16 wPosY;                          // macroblock ordinate (in macroblocks)
Uint16 wCodedBlockFlagL;               // coded_block_flag of the 16 AC luma cells
Uint4* wCodedBlockFlagCr;              // coded_block_flag of the 4Cr AC chroma cells
Uint4* wCodedBlockFlagCb;              // coded_block_flag of the 4Cb AC chroma cells
Uint8    bCodedBlockFlagDC;            // 0000 000x coded_block_flag of the DC luma cell
                                       // 0000 00x0 coded_block_flag of the DC Cr chroma cell
                                       // 0000 0x00 coded_block_flag of the DC Cb chroma cell
Uint8    bSubType[4];                  // sub_mb_type[ ]
Uint8    bRefIdx_l0[4];                // ref_idx_l0[ ]
Uint8    bRefIdx_l1[4];                // ref_idx_l1[ ]
Uint32 mvd_l0[16];                     // mvd_l0[ ][ ][ 1 ] << 16 + mvd_l0[ ][ ][ 0]
Uint32 mvd_l1[16];                     // mvd_l1[ ][ ][ 1 ] << 16 + mvd_l1[ ][ ][ 0]
Uint16 wCoeffL[256];                   // Luma coefficients
Uint16 wCoeffCr[64];                   // Chroma Cr coefficients
Uint16 wCoeffCb[64];                   // Chroma Cb coefficients
} MbInfo;
```

** : Uint4 means that one byte is the concatenation of two 4bits data in little endian order.*

The IP is designed to work in conjunction with a DMA controller. As a consequence, SliceInfo must be padded to 928 bytes. Headers greater than 916 bytes are managed by introducing SliceInfo structures with wNumMb set to 0.

MbInfo.bPredModeL[ ] must be set to 2 when MbType is different from I_4x4.
When the partition of a B or P macroblock is not 4x4, MbInfo.mvd_lx[ ] must be valid on the first cell of the subpartition (e.g.: for a 16x8 partition, only cells 0 & 8 require a valid mvd value).

For I_PCM macroblocks, the pixels values are transmitted extended to 16 bits through MbInfo.wCoeffL, Mbinfo.wCoeffCr & Mbinfo.wCoeffCb in the image order.

5.4 System integration

5.4.1 Integration with CPU bus

A typical multi-master system integration of the IP is represented on the figure below.

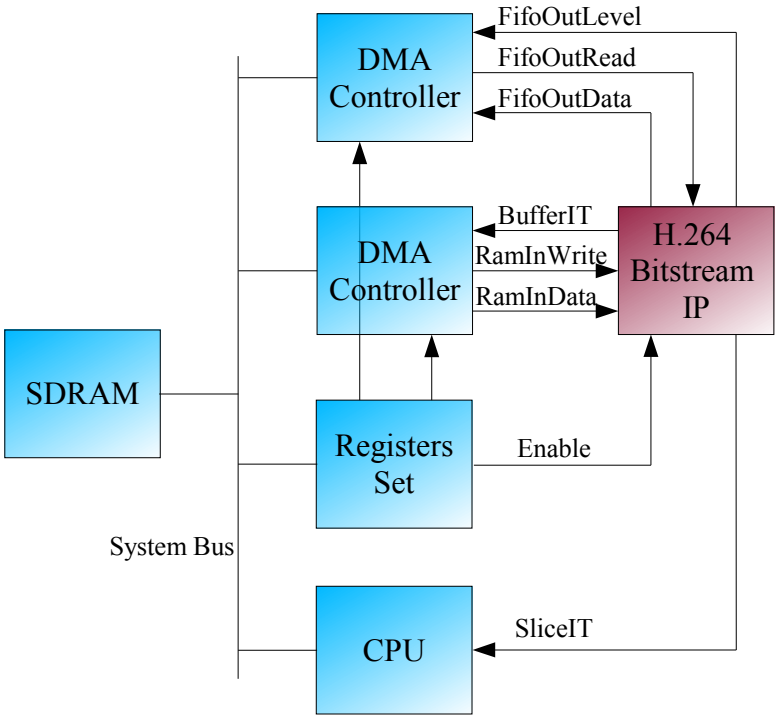

Figure 5- CPU bus system integration

A CPU manages the system. It is in charge of memory allocation, DMA controllers programming, and computation enabling.
Note: the output DMA controller must be able to transfer the very last FIFO content which size is variable, once the encoding job is finished (at the end of the video sequence).

5.4.2 Integration with AHB bus

The IP is simply configured by registers, then it works in an autonomous way, with Master data access in external memory, for input and output streams.

5.5 Operating mode

A SliceInfo structure must be followed by wNumMb MbInfo structures (*see software interface*) .
To process a new slice, it is not required to stop the IP: the module waits for a SliceInfo structure once it has received the correct number of MbInfo structures.

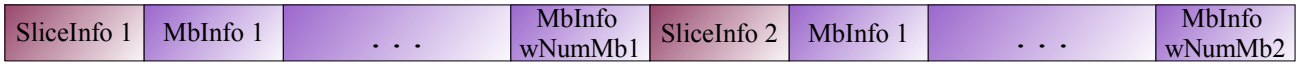

Figure 6- data flow

The macroblocks order must be respected, according to the MBAFF mode.

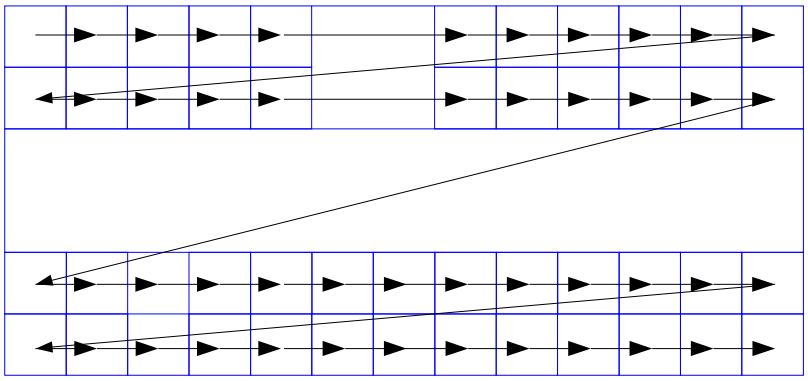

Figure 7- SliceInfo.Mbaff = 0

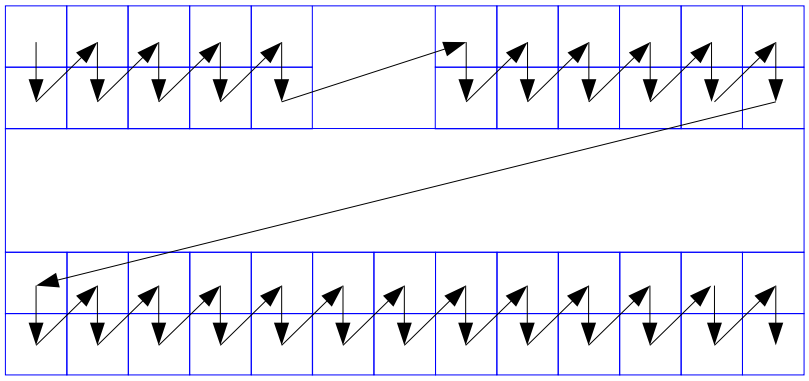

Figure 8- SliceInfo.Mbaff = 1

6 Deliverables

The deliverables includes:

- Bitstream IP VHDL source code
- User's manual
- Simulation files
- Reference software on PC (windows)

The reference software encodes a video file, and delivers output files, corresponding to the input and output of the IP. This enables to insert known data into the IP, and to compare the IP output with the software output.

An evaluation version of the IP is available.

As option, a PCI board with Altera Stratix II FPGA will soon be available, to ease real-time benches.