The LiquidMotion Programmable/Configurable Motion Estimation Instruction Set Processor Data sheet Version 5.0 11-09-2009

Introduction

The LiquidMotion processor is a reconfigurable ASIP (Application Specific Instruction Set Processor) designed to execute user-defined block-matching motion estimation algorithms optimized for hybrid video codecs such as MPEG-2, MPEG-4, H.264 AVC and Microsoft VC-1. A generic overview of how these codecs operate is shown in Fig. 1.

Motion estimation is used during inter-frame analysis to remove temporal redundancy and typically accounts for more than 50% of the whole entire cycle budget. This is especially true in the advanced video coding standard H.264[1], which includes advanced features for motion estimation such as variable block sizes, fractional pel support down to quarter-pel resolution, multiple reference frames and multiple motion vector candidates. The implementation of these features help H.264 to deliver high quality result, but they can also introduce a performance bottleneck into the video processing chain.

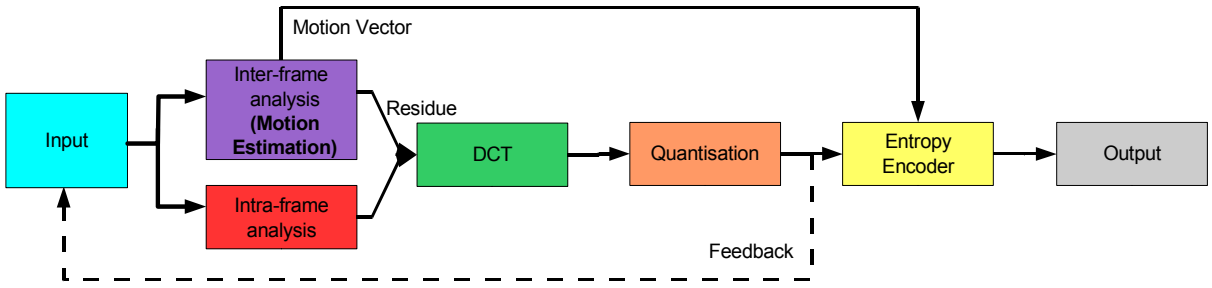

Fig. 1 - Generic Video Codec Block Diagram

Traditionally, the preferred implementation for motion estimation in hardware has been based on full-search algorithms due to their regular dataflow, which makes them well suited to systolic array principles. This is a simple approach capable of achieving a high level of hardware utilization and generally avoids global routing which results in high clock frequencies. Practical full search implementations, however, need to consider the memory interface to the frame data, and this can often lead to designs requiring large data widths and port counts, or a large number of registers to buffer the pixel data. Data broadcasting techniques can be used to reduce this need but this can also reduce the achievable clock frequency.

[1] H.264 has been selected as the preferred coding standard by the Blu-ray Disc Association and for the new high-definition television broadcasting standard (DVB-S2) due to its excellent coding performance that typically halves the bit-rates compared with previous standards.

Full search implies a large number of SAD operations, and even for reduced search areas, a number of optimizations are still required to make it more computationally tractable. One of the drawbacks of the full search approach in hardware is that its throughput is determined by the size of the search area, and this increases dramatically for high definition video formats thereby limiting its scalability and increasing its energy consumption. Considering fractional-pel in full search compounds this problem further since an exhaustive search to quarter-pel precision will increase the number of points to be searched by more than an order of magnitude, on top of the overhead in interpolating these pixels using computationally intensive filters.

LiquidMotion offers scalable performance dependent on the features of the chosen algorithm and the number and type of execution units implemented. Hardware configuration can typically be achieved at compile time by adapting the architecture to the chosen algorithm, and in a FPGA implementation, it is possible to pre-compile a range of hardware bitstreams with different configurations from which one can be chosen to match the current video processing requirements. The LiquidMotion processor microarchitecture can be easily scaled to high definition video even when using low cost FPGAs such as the Xilinx Spartan3 - and the ability to program the search algorithm to be used, and to reconfigure the underlying hardware that it will execute on, combines to give an extremely flexible video processing platform. The processor is programmed using a reduced instruction set optimized for the development of motion estimation algorithms.

A base configuration consisting of a single 64-bit integer pipeline, which is capable of processing a hexagonal motion estimation algorithm such as the one used in the x264 video encoder[2], can be implemented in as little as 2000 logic cells on a Xilinx FPGA. As an example of its performance, macroblocks from a high motion video clip such as the American football test sequence can be computed in around 700 clock cycles using a diamond search algorithm with a final square refinement as implemented in x264, and even at a modest clock frequency of 100 MHz, this translates to around 140,000 MBs/second which is enough to support the high definition 720p@30 frames/second. This performance scales linearly with the number of execution units implemented, and is unaffected by the addition of fractional-pel support.

[2] x264 is a open-source implementation of a H.264 encoder: http://www.videolan.org/developers/x264.html

Hardware description

Fig. 2 shows the architecture of a configuration with 6 execution units: four integer-pel execution units (64x4 bits data path), one fractional-pel execution unit (64-bit data path) and one interpolation unit.

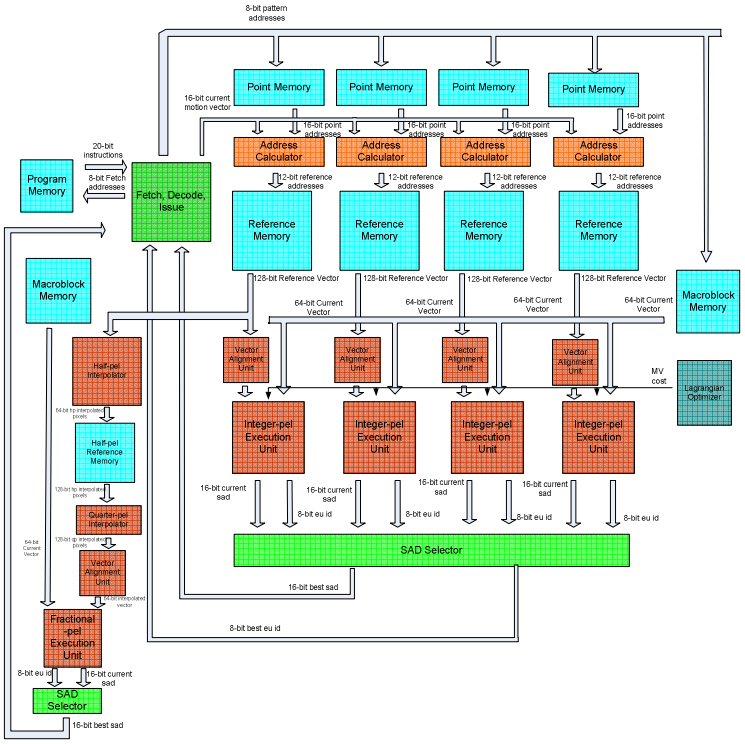

Fig. 2 - LiquidMotion sample configuration architecture

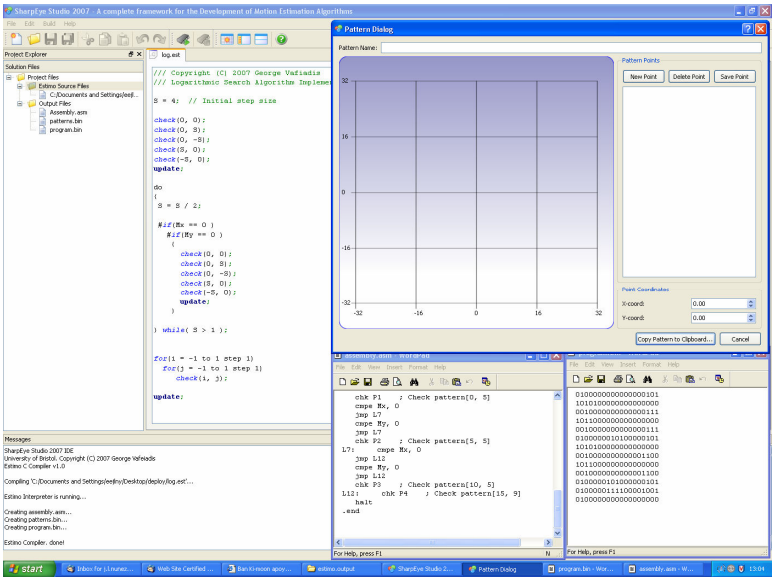

Fig. 3 - Toolset for algorithm development

Motion estimation algorithms have not been standardized and over the years many fast motion estimation algorithms have been proposed by both industry and academia. Well-known fast motion estimation algorithms include logarithmic search, three-step search, diamond search, hexagon search, etc. along with more complex methods such as PMVFAST and UMH. These algorithms work by searching only a subset of all candidate positions for the best match, using a dynamic search pattern iteratively with each local winner set as the starting point for the following iteration. The same principles of calculating motion vectors by block-matching the current frame with other reference frames in the video sequence are shared by MPEG-2, MPEG-4, VC-1 and H.264, although the more recent standards have more sophisticated ways of achieving this.

Tables 1 and 2 indicate the level of performance and complexity obtained by different configurations of the LiquidMotion processor. The LiquidMotion compiler toolset shown in Fig. 3 enables a designer to implement any of the above algorithms, or to develop his/her own using a simple C-like syntax which can then be compiled into machine code ready to run on the hardware.

Number of Integer Execution units implemented	Throughput in macroblocks per second (16x16, diamond, 200 MHz, 4 ppp)	Throughput in Macroblocks per second (16x16, 8x8, diamond, 200 MHz,4ppp)	Throughput in macroblocks per second (16x16, hexagon, 200 MHz, 6 ppp)	Throughput in Macroblocks per second (16x16, 8x8, hexagon, 200 MHz, 6ppp)	Throughput in Macroblocks per second (16x16 UMH 200 MHz, 16 ppp)	Throughput in Macroblocks per second (16x16, 8x8 UMH 200 MHz, 16 ppp)
1	372,960 (1080p@30)	233,918 (720p@50)	260,983 (1080p@30)	173,988 (720p@30)	84,813	56,542
2	692,640 (1080p@50)	461,760 (1080p@50)	495,867 (1080p@50)	330,578 (1080p@30)	166,233 (720p@30)	110,822
3			708,382 (1080p@50)	472,255 (1080p@50)		
4	1,212,121 (1080p@50)	808,080 (1080p@50)			319,680 (1080p@30)	213,120 (720p@50)
6			1,239,669 (1080p@50)	826,446 (1080p@50)		
8					593,692 (1080p@50)	395,794 (720p@30)
16					1,038,961 (1080p@50)	692,640 (1080p@50)

Table 1 - Estimated number of execution units required depending on motion estimation algorithm and video format

	Virtex - 4 SX35		
Configuration	**LUTs used /LUTs available**	**Memory blocks used/Memory blocks available/Minimum memory bits**	**Critical path (ns) Logic levels**
1 IPEU/ 0 FPEU	2259/30720 (7.4%)	21/192 (10%)/95 Kbits	4.976/8
2 IPEU/ 0 FPEU	3805/30270 (12.6%)	38/192 (19%)/179 Kbits	5.040/8
3 IPEU/ 0 FPEU	5571/30270 (18.4 %)	55/192 (28%)/263 Kbits	5.032/7
1 IPEU/ 1 FPEU	9143/30270 (30.2%)	31/192 (16%)/95+42 Kbits	4.986/6
2 IPEU/ 1 FPEU	10985/30270 (36.2%)	48/192(39%)/179+84 Kbits	4.996/9

Table 2 - Complexity of different hardware configurations when implemented on a SX35 Xilinx Virtex4 device

The application specific instructions generated by the compiler specify the parallelism available at the search point level, and the resulting level of performance depends on the number of execution units present. Similar to VLIW processors, parallelism is extracted at compile time using a simple fetch, decode and issue unit which deals with one instruction at a time, and just like superscalar processors, different implementations of the microarchitecture remain binary compatible so that hardware upgrades will not need a software recompile.

Hardware Interface

The LiquidMotion processor is fully synchronous, and its top-level interface is shown in Fig. 4. Fig 5. shows thirty general purpose registers, and these, can be accessed through the register access ports for reading and writing. If a new program/point memory needs to be loaded this can be used using the dma_address and dma_data_in ports to write addresses 0 to 255 (256 locations per memory). The dma_data_in port should be padded with zeros since the program memory instructions only have 20 bits while the point memory locations only have 16 bits. To control which memory to write the dma_pom_we (point memory) and the dma_prm_we (program memory) signals are used.

The command register (Reg0) is used to start the processing of a new macroblock and its contents are used to indicate the partition mode required and the coordinates of the macroblock to be processed. Once bit 31 in the command register is set to 1, the processor will begin executing the internally stored program using the specified partition mode on the macroblock currently loaded. A new command must be issued if a new partition mode is required, and this allows the rate-distortion optimization algorithm part of the video codec to decide whether or not more partitions should be pursued. Fig. 6 shows the strategy used by the x264 encoder, while Fig. 7 shows the hardware calls performed by the rest of the video coding algorithm.

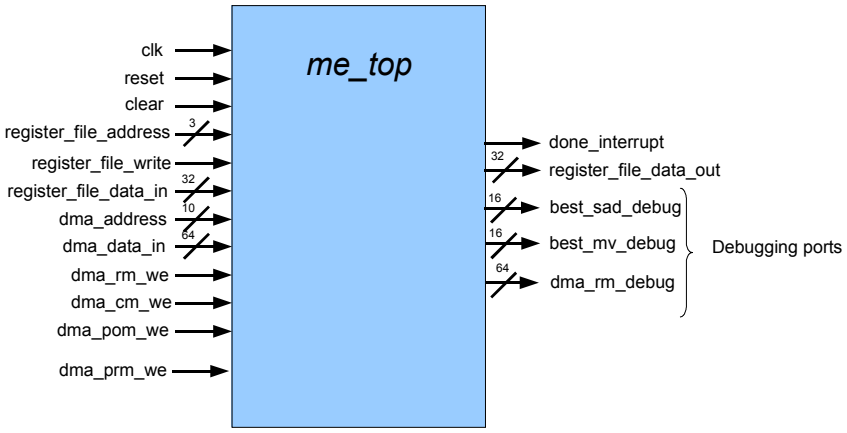

Fig.4 - LiquidMotion interface

The DMA ports are used to load new macroblocks, reference area data and program data into the processor's internal buffers. These DMA transfers must be carried out by an external controller, and can be performed in parallel with executing a search. At the start of each new frame row it is necessary to load a full reference area of 7x5 macroblocks (112x80 pixels search area) but for each subsequent macroblock from the same row only the newest column of 1x5 macroblocks

needs to be loaded. Internally, the processor operates a sliding window mechanism so that the overlapping reference area can be transparently reused.

The done_interrupt signal will be raised by the processor once the processing of the current macroblock is complete. Finally, the debugging ports can be used to monitor the SAD and winning motion vector calculations as they are performed, and also to read back the contents of the internal memories.

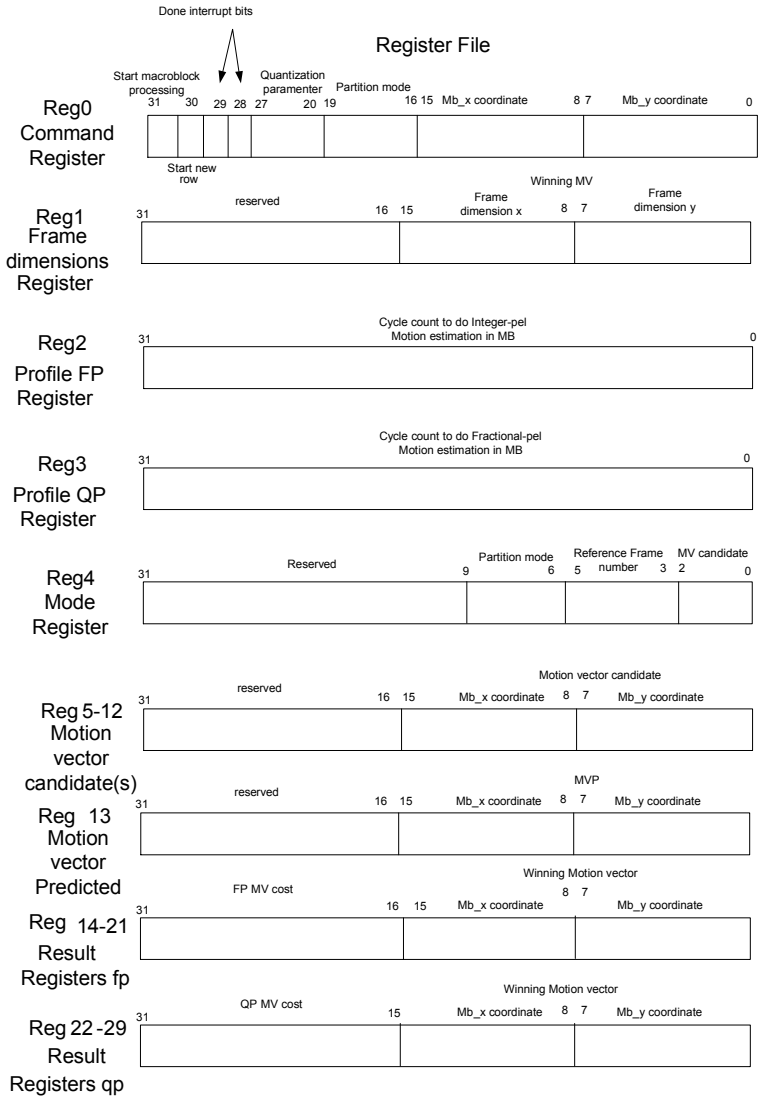

Fig. 5 - Register File description

Signal Name	I/O	Width	Description
clk	In	1	System clock signal
reset	In	1	Synchronous reset
clear	In	1	Asynchronous clear

register_file_address	In	3	Access general purpose registers
register_file_write	In	1	Write enable for the general purpose registers
register_file_data_in	In	32	Data in for general purpose registers
register_file_data_out	Out	32	Data out for general purpose registers
dma_address	In	10	Read/write addresses for reference memory, current macroblock memory, point memory and program memory
dma_data_in	In	64	Reference memory/current macroblock memory/ program memory/point memory data input
dma_rm_we	In	1	Write enable reference memory
dma_cm_we	In	1	Write enable current macroblock memory
dma_pom_we	In	1	Write enable point memory
dma_prm_we	In	1	Write enable program memory
done_interrupt	Out	1	Macroblock processing terminates
best_sad_debug	Out	16	SAD value debugging port
best_mv_debug	Out	16	Motion vector debugging port
dma_rm_debug	Out	64	Reference memory debugging port

Table 3 - Processor interface signal description

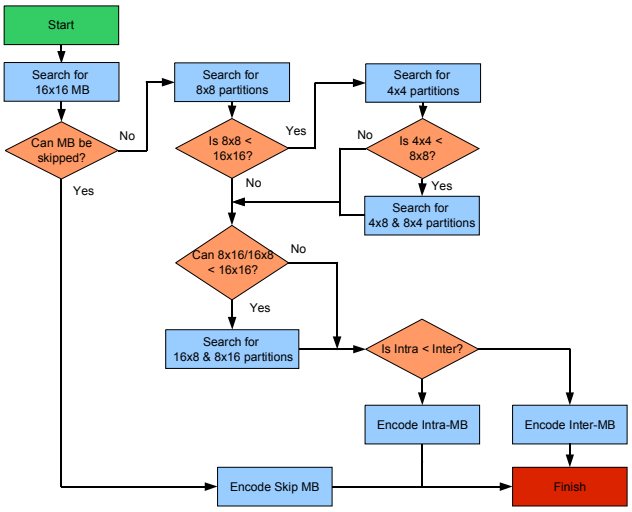

Fig.6 - Mode selection in x264

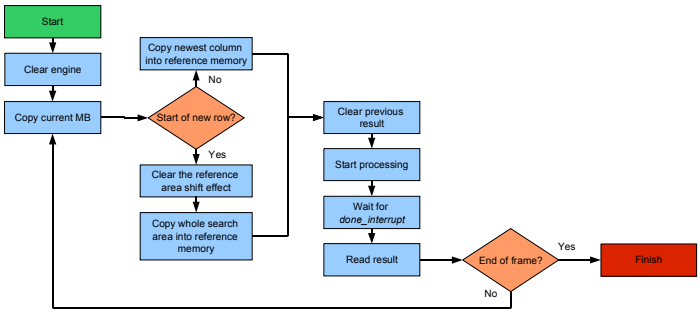

Fig.7 - Software access to the ME processor

IP Integration

A prototype implementation has already been developed which successfully integrates the base configuration of the LiquidMotion core with a LEON3 System-on-Chip design, as illustrated in Fig. 8, and is available for demonstration. The processor is instantiated inside a wrapper component with an AMBA interface, as depicted in Fig. 9, which is implemented on a PCI-based FPGA board developed by Avnet containing a single Xilinx Spartan-3 device. A supporting API is also available, and the entire design has been successfully integrated into an x264 plug-in for the VLC cross-platform multimedia player. The system can be seen working at the following URL: http://uk.youtube.com/watch?v=TkRnm8qvdDA

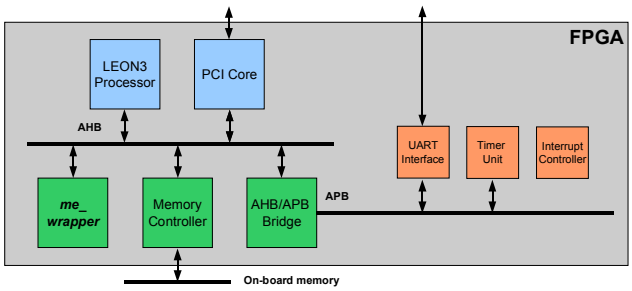

Fig. 8 - Processor SoC integration example

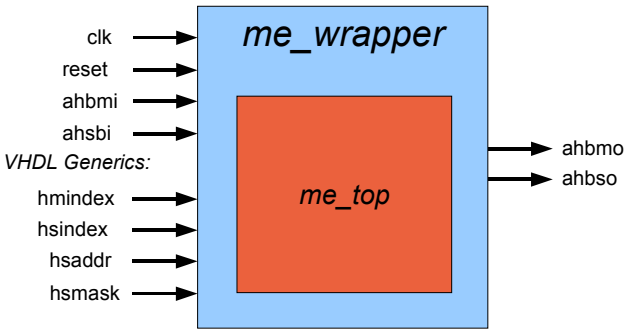

Fig. 9 - Processor AMBA wrapper

Summary of the LiquidMotion processor features

- Programmable block-matching motion estimation algorithms (hexagonal, diamond, log, PMVFAST, UMH, etc) using an optimized reduced instruction set architecture.
- Lagrangian hardware support adds the cost of the motion vector to the SAD cost using Lagrangian multipliers typically reducing bit rate by 5-10% .
- Configurable number of integer execution units (1 to 16) to give scalable performance, from low cost energy efficient single-pipeline configurations up to highly parallel implementations capable of executing the advanced motion estimation features available in H.264.
- Half-pel and quarter-pel support using a configurable 6-tap interpolation filter array implemented with 48 processing elements and a single fractional-pel execution unit. Fractional-pel instructions can be issued to these execution units in parallel with the main integer execution units, allowing each to operate on different macroblocks.
- Accompanying toolset/compiler for easy development of new motion estimation algorithms – enabling a designer to use a simple C-like syntax to implement publicly available motion estimation algorithms, or his/her own algorithms without having to resort to any error prone assembly language syntax.
- Multiple reference frame (1 to 5) and partition size support (16x16, 16x8, 8x16, 8x8) - capable of working with any rate distortion optimization algorithms present in the rest of the video encoder so that partition modes and reference frames can be used selectively for an optimal performance/energy trade off.
- Multiple motion vector candidates and a large search area of 112x128 pixels enable precise motion estimation with little impact on execution time.
- Unrestricted motion vector capability – as described in the H.264 standard, which allows a vector to describe motion that lies outside of the reference frame.
- Efficient implementation of the early termination and duplicate-point optimizations in configurations with multiple execution units – these features terminate or change the

execution flow of the motion estimation algorithm when a pattern fails to improve on the previous best match bypassing search points that have already been computed,

- VHDL RTL description deliverable optimized for ASIC or FPGA implementation – fully synchronous and synthesizable allowing it to make use of the special features available in modern FPGAs.
- Working demonstration using the x264 encoder and PCI bus available – take a look at http://uk.youtube.com/watch?v=TkRnm8qvdDA for a very brief example!.