Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2013, Article ID 783501, 10 pages
http://dx.doi.org/10.1155/2013/783501

Research Article

Development of a SoC for Digital Television Set-Top Box: Architecture and System Integration Issues

André Borin Soares, Alexsandro Cristóvão Bonatto, and Altamiro Amadeu Susin

PGMICRO, UFRGS, Avenida Bento Gonçalves 9500, 91501-970 Porto Alegre, RS, Brazil

Correspondence should be addressed to André Borin Soares; borin@inf.ufrgs.br

Received 21 January 2012; Revised 14 November 2012; Accepted 16 December 2012

Academic Editor: Oliver Sander

This work presents the integration of several IPs to generate a system-on-chip (SoC) for digital television set-top box compliant to the SBTVD standard. Embedded consumer electronics for multimedia applications like video processing systems require large storage capacity and high bandwidth memory. Also, those systems are built from heterogeneous processing units, designed to perform specific tasks in order to maximize the overall system efficiency. A single off-chip memory is generally shared between the processing units to reduce power and save costs. The external memory access is one bottleneck when decoding high-definition video sequences in real time. In this work, a four-level memory hierarchy was designed to manage the decoded video in macroblock granularity with low latency. The use of the memory hierarchy in the system design is challenging because it impacts the system integration process and IP reuse in a collaborative design team. Practical strategies used to solve integration problems are discussed in this text. The SoC architecture was validated and is being progressively prototyped using a Xilinx Virtex-5 FPGA board.

1. Introduction

Video processing systems require high performance processing elements and an efficient memory hierarchy design to reach real-time performance in the decoding of high-definition video sequences. Dedicated high performance modules are integrated into a single system which decodes the incoming bitstream and produces the output video images. In this process, reference frames are stored to be reused in the decoding process. A large size memory as an off-chip DRAM (Dynamic Random Access Memory) is mainly used and the memory accesses are directed to a single off-chip memory interface. The memory hierarchy design and computation complexity are bottlenecks to reach real-time high-definition video decoding [1].

This work presents an architectural design and FPGA (Field Programmable Gate Array) implementation of a SoC for H.264/AVC video decoding [2]. A SoC is a complex system, and the integration and validation of the design are the challenges of the development process. The architecture considers a four-level memory hierarchy [3] composed of local SRAM memories and by off-chip DRAM memories. Off-chip DRAM memories can guarantee the necessary storage capacity at low cost if compared to embedded SRAM. The memory controller is designed with a multichannel data interface because different processing modules need to share the same data port. The multichannel controller manages data access requests and optimizes the reference memory utilization, enabling data processing modules to interact efficiently in order to satisfy the performance requirements.

This paper is organized as follows. Section 2 discusses related works; Section 3 presents the set-top box hardware architecture, the H.264/AVC video decoder, the audio decoder, and the implemented memory hierarchy; integration issues and strategies employed are discussed in Section 4; FPGA implementation results are discussed in Section 5 and finally we conclude in Section 6.

2. Related Works

Related works can be grouped in three main categories: software decoding, hardware/software decoding, and hardware decoding. Pescador et al. [4] presented an H.264

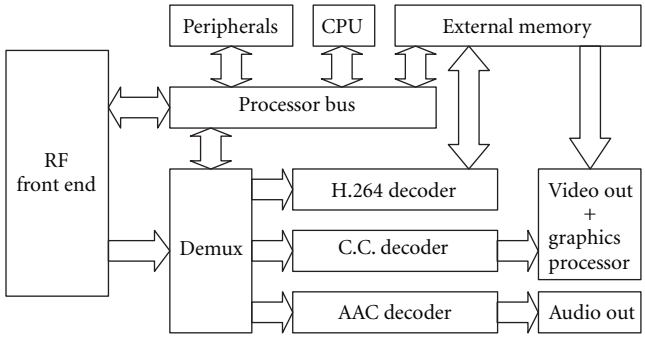

Figure 1: SBTVD compatible set-top box.

video decoder running on a VLIW DSP at 600 MHz with software highly optimized to use DSP native features. Results presented the minimum of 57% of CPU utilization for 480p sequences in the baseline profile. Yang et al. [1] developed a single chip decoder SoC for H.264 decoding using an RISC processor and hardware accelerators. The basic decoding flow and synchronization are controlled by software. The system operates at 100 MHz at the baseline profile, decoding CIF videos at 20 MHz. Lin et al. [5] presented an ASIC implementation of the H.264 video decoder with dedicated hardware modules and two memory controllers, decoding HD1080p videos at 120 MHz. It uses only 4.5 KB of local memory. Modern SoCs like [6] have all the algorithms developed in hardware to reach high performance with low power.

The video decoder presented on this work is completely implemented in hardware and presents itself as an intermediary step for the future production of a chip. Currently it is capable of decoding 720p videos at 50 MHz in real-time on an FPGA. Based on previous experiments [7], a silicon version is expected to get the benefit of low power consumption by operating at a lower frequency than software-based approaches and by the utilization of a design flow with low power techniques. Also, the inclusion of a dedicated CPU on the system will enable an effective implementation of a user interface and the execution of applications. Beyond set-top boxes, this solution could also be reused in other systems such as tablets or mobile phones. Nevertheless, a more detailed analysis concerning the overall power consumption must be performed for these applications. One of the main contributions of this work is to present the techniques used to reduce the complexity of the integration and test of the SoC design while keeping the RTL description used in the initial creation of the system components.

3. Set-Top Box Architecture

The SBTVD (Brazilian Digital Television System, in portuguese) follows the ISDB-t standard [8], which is based on the Japanese television system (ISDB). This standard comprises H.264 for video encoding and AAC for audio encoding. Video and audio decoding must be performed in hardware by specialized architectures in order to reach decoding efficiency. The Brazilian government funded the development of an H.264 decoder prototype and an SBTVD compatible SoC [9, 10]. An overview of the architecture of the SBTVD SoC developed in this work is shown on Figure 1. The design is being developed following Brazilian standards [8, 11] to enable compatibility with broadcast transmissions. The SoC is composed of an H.264 video decoder, an AAC audio decoder, a closed caption decoder, a Leon3 CPU [12], and a demultiplexer, receiving a compressed bitstream from an external RF front-end and demodulator. Additionally a graphics processor exhibits decoded images, superposes decoded subtitles and locally generated messages or images and a CPU is intended to run user interface and interactivity applications. Video decoder, main CPU, and graphics processor are connected to the external memory controller. The system bus is an AHB AMBA bus standard, implemented in VHDL language.

The LEON3 is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture developed by Gaisler Research [12] and available for use in research projects under free license. SPARC is a CPU instruction set architecture (ISA), derived from a reduced instruction set computer (RISC) lineage.

3.1. H.264/AVC Video Decoder Architecture. H.264/AVC is the latest video coding standard of the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). This state-of-the-art video coding standard outperforms previous standards by employing bipredictive motion estimation, spatial prediction, and adaptive entropy coding techniques. However, the cost of increased performance is an augmented computational complexity and the enlarged memory traffic when compared to previous standards. Dedicated modules must supply the high performance demand of the decoding processing. Memory access optimizations are also necessary to handle data of the reference frames while video is decoded and exhibited in real-time.

The H.264/AVC video coding standard defines a set of levels and profiles. Complexity and output bandwidth are related to the level, while profiles refer to different set of coding functions. There are three different profiles defined by the standard: baseline, main, and high. The main profile includes television broadcasting and video storage, supporting interlaced videos, intercoding with bipredictive slices, intercoding with weighted prediction, and entropy coding using context-adaptive arithmetic coding (CABAC) [2]. The hardware decoder in development in this work targets decoding video streams in the main profile at level 4.0. Figure 2 shows the video reconstruction path in an H.264 decoder, starting with the compressed video bitstream and ending with the YCbCr video that is stored in the RPB (reference picture buffer).

Parsing with entropy decoding extracts from the compressed video bitstream the syntax elements to interframe and intraframe video reconstruction and compressed residual data. The residual data is decoded using fixed or variable length binary codes in the entropy parsing module and is processed in the inverse transform and inverse quantization

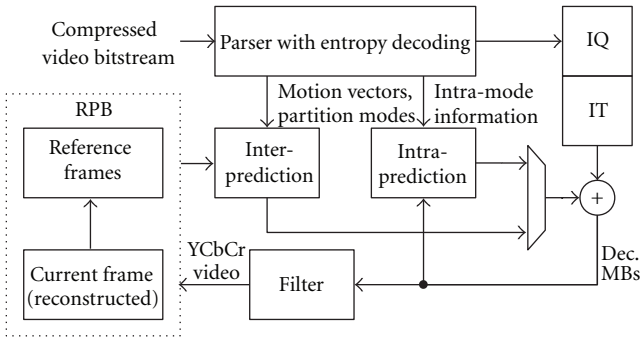

Figure 2: Block diagram of the H.264/AVC video decoder.

(IT and IQ) steps. Using information decoded from the bitstream, the decoder creates a prediction block using intra or inter prediction modes. Motion compensation (MC) is the hardware module which reconstructs an actual frame from reference frames. Intraprediction reconstructs each image block from its neighborhood.

The H.264 decoder consists of five main processing units, designed as IP (Intellectual Property) cores: the inter-prediction module, or MC (Motion Compensation), the intraprediction or Intra, the deblocking filter, the inverse transform (IT) and inverse quantization (IQ), and the entropy decoding module. The filter is the output of the decoder and generates the predicted macroblocks to the RPB by reducing block artifacts.

The RPB is the module designed to manage the reference frames used in the video decoding process. The main functions of the RPB are the frame data storage and control. Decoded frames are stored on the external memory, in which they can be located by address pointers. They are also indexed on reference lists to be used as references in the decoding process.

3.1.1. Storage Requirements for Video Applications. Video processing applications have high bandwidth requirements that increase with the video resolution, bit depths, and frame rates. For example, when decoding a SD (720 × 480) video stream of 30 frames per second, the video decoder output generates 10.4×10^6 pixels per second, while for HD (1280 × 720) video stream of 30 frames per second, the pixel output is 27.65×10^6 pixels per second.

The colorspace, pixel's depth, and picture subsampling are also determinant of the output bandwidth. There is an information compression if the pixel is represented in the YCbCr (luminance, chrominance blue and chrominance red) with 4 : 2 : 0 subsampling instead of use RGB (red, green, and blue) format. For example, a video picture in Full-HD resolution (1920 × 1080) represented in YCbCr 4 : 2 : 0 8-bit per pixel requires 3.1×10^6 bytes and in RGB 12-bit per pixel 9.4×10^6 bytes to be stored in memory.

The video decoding process in H.264 standard Main profile requires about 12.5×10^6 bytes for reference frame memory in YCbCr 4 : 2 : 0 8-bit format. This memory size is needed to store four video frames in Full-HD resolution.

3.1.2. Reference Picture Buffer Accesses Behavior in H.264/AVC Video Decoding. Image is processed in the form of regions of 16 × 16 pixels size called macroblocks of pixel (MBs) samples. Video processing is done by manipulating pixels in coarse-grain tasks that are mapped onto processing units. This is the video decoder granularity and each pixel sample in the macroblock is represented by its luminance (Y) and chrominance (Cb and Cr) components. The video decoder output is a 32-bit four pixels width line-of-pixels and a sequence of four line-of-pixels generates a 4 × 4 block of pixel samples. The macroblock thus consists of a sequence of 256 luminance samples followed by 64 Cb chrominance samples and 64 Cr chrominance samples, ordered in double-z.

There are three main modules accessing the RPB: the filter output (reference pictures), video display output, and motion compensation process. They have different access behaviors and bandwidth requirements.

The MC process is the most computationally demanding in the video decoder [13], and the one that generates more external data requests. As the video decoding process has an unpredictable behavior, the MC module can access the main memory at different data rates. Also, the region of pixels used to reconstruct the image can be different in each decoded macroblock. The MC cache uses an addressing scheme based on the (x, y) pixel block coordinates and the reference picture number.

The H.264 video decoder output generates decoded pixels in a sequence of macroblocks. It does not generate any addressing information, which must be controlled by the RPB.

Video display output module needs to fetch decoded frames in the right display order, which can be different from the decoding order. It scans an entire line of pixels in the exhibited image. As in the case of the decoder output, the video display does not generate addressing information. Pixels are requested by the video output and the RPB must perform address control.

We can estimate the memory interface access bandwidth for each processing unit by the decoded video resolution and macroblocks generation rates. The interpolation blocks in motion compensation process are all 4 × 4 size. It can be seen that MC process is the most bandwidth demanding in the H.264 decoder. The total memory bandwidth necessary to decode 1920 × 1080@30fps is 1.002 GB/s. More details can be found in [14, 15].

3.2. AAC Audio Decoder. The audio decoder was developed on a secondary prototyping board. A data link was designed to connect both boards. The MPEG-4 AAC decoder is capable of decoding a stereo bitstream with 48 kHz sampling rate in real-time operating with a clock frequency of 4 MHz. It includes a parser, a filter bank, and a spectrum decoder composed of noiseless decoding, an inverse quantizer and a rescaling module and decodes mono, stereo, and 5.1 audio

streams. The audio decoder architecture is described with more details in [14].

3.3. Multichannel Memory Controller. Currently the implementation of the quantity of memory necessary for the RPB as on-chip memory is not cost effective. On-chip SRAM is about 7 mm^2 per mega-byte in 65 nm technology [15]. Embedded DRAM has a higher density but requires additional mask layers for manufacturing, increasing system cost. Architectures of video processing systems require a single interface to off-chip DRAM memory in order to achieve the necessary storage capacity at low cost. The multichannel memory controller (MMC) designed in this work implements the RPB to the video decoder described in Section 4. An off-chip 64-bit wide DDR2 SDRAM memory running at 200 MHz is used as main system memory. Also, the multichannel memory controller implements a data/instruction interface for the Leon3 CPU through the AMBA bus standard interface.

3.3.1. DDR2 SDRAM. DDR2 SDRAM is a double-data rate synchronous DRAM interface designed to transfer data on the rising and falling edges of the bus clock signal. This memory allows higher bus speed than its previous memory standard, DDR SDRAM, and requires lower power by running the internal clock at one quarter the speed of the data bus. For example, considering that the internal memory clock is 100 MHz, the bus clock will be 200 MHz and the bus interface is capable to perform 400 megatransfers per second. Also, with data being transferred 64-bit at each bus clock edge, the maximum transfer rate is 3,200 mega-bytes per second.

Data accesses are linear and words are stored organized in banks and pages. Data can be transferred in bursts with 4 or 8 data words in each memory access. Data memory contents are accessed by page activation (ACT), using the row-address strobe command and a column-address strobe command. In the case of a read operation, data is available after the CAS Latency (CL) which can be 3 to 7 clock cycles plus an optional additive latency (AL).

DDR2 SDRAM latency can reduce significantly the system performance if data are not transferred in bursts. Also, frequent row changes or bank conflicts reduce the system performance because it is necessary to deactivate the current row or bank, and activate the next to be used. This is done by using a sequence of commands called precharge (PRE) and activate (ACT), taking about 10 cycles after the last data access operation.

By storing the images in the off-chip memory in the form of macroblocks of pixels organized as YCbCr 4 : 2 : 0 8-bit image format, it is possible to design the granularity of data transfers to the reference memory as being a macroblock. As presented before in [16], macroblock granularity provides better SDRAM efficiency and less power consumption. This granularity for data transfers is used in the H.264 decoder, video output, and graphics processor.

3.3.2. Multichannel Memory Controller (MMC) Architecture. Figure 3 shows the memory controller architecture. The off-chip memory interface consists of a physical IP (DDR2 PHY) from Xilinx [17]. The hardware modules are connected to the memory controller through channels which share the same command and address bus, being necessary to have an arbiter controlling the data requests. Each channel interface is defined as IF (interface) with an associated number, in the presented case IF0 to IF4. Each IF consists of data buffers and address generators. The address generators are necessary to index stored data in the external memory. In this system, only the CPU generates addressing information to manage data. Each IF contains data buffers in different sizes and data organization schemes, to optimize external memory data transfers.

An arbiter was designed to control read and write requests do the DDR2 PHY. Also, the MMC contains a data multiplexer and demultiplexer and a RPB control, necessary to manage the video frame buffer in the external memory. This design implements five interfaces to the external memory with corresponding command, address, data and control signals. Nevertheless, it is possible to extend the architecture to use more data channels. The multichannel interface uses a simple protocol where an acknowledge signal gives permission to the module after the received access request. The main advantages of this controller are the scalability to add more channels, the possibility to control the priorities of transfers, and the data conversion on the channel buffering. A more detailed analysis of the memory hierarchy was presented in [18].

Optimized Memory Hierarchy. When interfacing with DRAMs, it is necessary a time to setup a data transfer and this time cannot always be overlapped with another data transfer. Therefore, DRAM accesses need to have large burst sizes in order to use the DRAM interface efficiently. Whether all this data is actually needed depends on the spatial locality of the data accesses. The sizes of the DRAM bursts need to be aligned with the sizes of the cache blocks and the local buffers used to interface each processing unit. Figure 3 shows local and off-chip memory resources used by the multichannel memory controller, grouped within the levels of the memory hierarchy. It contains the following four levels.

Level 3: Off-chip DRAM memory: it is the biggest memory module with reduced cost if compared with the internal SRAM memory. This level has the larger latency.

Level 2: Memory controller buffer: it is the memory level used to store data for an off-chip memory transfers. Data is saved along with addresses and commands to be written in external memory. This memory level allows the controller (PHY) to manage off-chip memory information with autorefresh operations and different banks or row changing without disrupting the interfaces that are accessing data. The Read FIFO is only used with additive read latency (AL) option is enabled [17].

Level 1: Macroblock-level memory: in this level, data is stored in buffers until reach the minimum size of a macroblock before being transferred to external memory. This level is necessary to maximize the size of data transfer

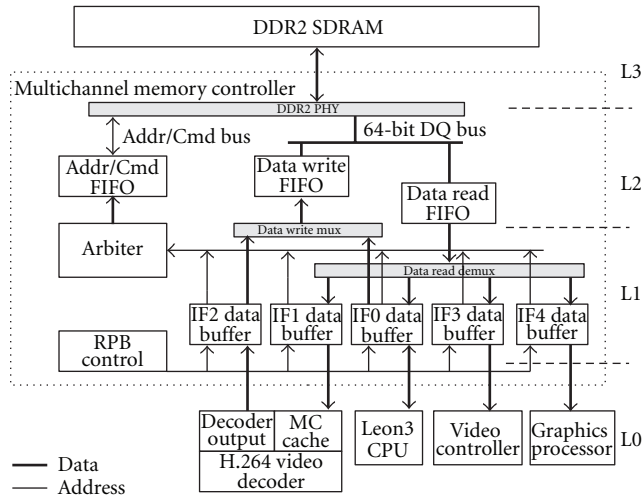

Figure 3: Multichannel memory controller block diagram. The memory resources are also grouped within the levels of the memory hierarchy (L0 to L3).

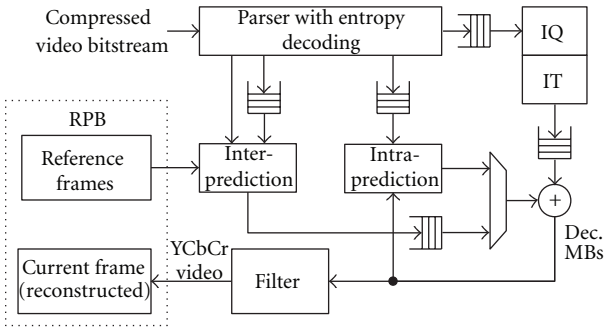

Figure 4: Buffering in H.264 video decoder.

to external memory, because the transfer takes the form of bursts of data in line memory address. It is also necessary to generate the addresses.

Level 0: Local SRAM processing buffers: it is the lowest level of memory, characterized by local memory in the processing units used to process information locally; it enables the execution of more local processing without external memory access. Figure 4 shows local buffers on the video decoder.

IF Channel Interfaces. Each channel interface consists of an address generator and a data buffer. The address generators are used to reorganize the macroblocks of pixels in reference pictures. The buffers can be parameterized for each access behavior requirement.

In the case of video display, the buffer has a size of 240 macroblocks. This is necessary to store two entire lines of macroblocks, considering full HD video resolution, while the other line is being exhibited. The decoder output contains a buffer with two macroblocks capacity. The MC requires a data buffer with storage capacity of three macroblocks. The MC architecture used in the video decoder is presented in [19]. It contains a local cache to store a tridimensional data structure which contains the requested reference pixels while the reconstruction process is executed. Cache size is $40 \times 16 \times 16$ 8-bit samples for Y and $20 \times 8 \times 16$ 8-bit samples for Cb and Cr.

The buffers are also used to align multiple 32-bit words (or a line-of-pixels LoP) to 64 or 128-bit data wide and to synchronize multiple clock domains. Larger buffer size increase memory bandwidth utilization due to the reduction of the utilization of memory commands.

The address generator receives information from the RPB control to store the reference frames, generating the addressing scheme to access data in the external memory. Each decoded frame is labeled as reference frame or not. This information is generated by the decoder during the bitstream parsing process.

Multichannel SDRAM Arbiter. The arbiter controls the access of each hardware module to the time-division multiplexed memory channel. It is associated a priority to each IF channel in the arbitration scheme, considering one classification associated to high or low latency and high or low bandwidth. Each processing unit accessing the off-chip memory is classified as latency sensitive (LS) or bandwidth sensitive (BS). LS processing units require immediate access to memory channel and use the channel for few memory transactions. In the other side, BS processing units uses the off-chip memory channel to for long memory transactions, transferring high data volumes, but without immediate access. In this implementation, the best approach to multiplex the memory channel between process units is the use of a priority-based arbitration scheme with preemption. More details are show in [18].

Reference Picture Buffer Control. Frame allocation in the RPB is performed by a dedicated buffer management module, which calculates the buffer size and the pointers to the buffers stored in the reference memory. This calculation is performed every time that the video resolution changes. Image

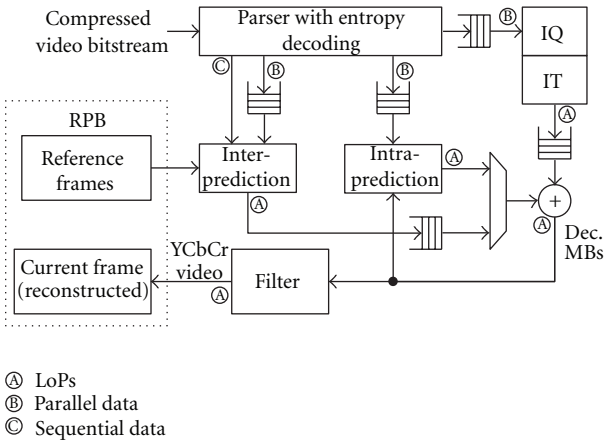

Figure 5: Data format on the H.264 decoder.

dimensions are stored in the bitstream, being recovered by the parser module.

Motion Compensation requires random memory access based on a Reference Index that points to specific buffers in the reference memory. Base pointers associated to the reference index are provided by the buffer control and fed the address generator of IF1 data buffer.

Reference list management must support a dynamic list with pointers to buffers. Two memories are used to implement this behavior. The first memory contains the pointers to the reference memory contents. The second memory contains the state of the list, including buffer state and an index to the first memory.

Operations performed by the RPB control module include buffer allocation for the video decoder output, identification of the buffer to be exhibited, and reallocation of an exhibited buffer. The graphics processor and Leon3 CPU also interface with the buffer management control when accessing data in the graphics buffer memory region. The CPU generates an on-screen display (OSD) and stores it in the graphics processor memory area located at the external memory. The OSD image is superimposed on a screen picture with the decoded video by the graphics processor. This commonly used feature was implemented in full image resolution, using the YCbCr 4 : 4 : 4 pixel format to achieve high quality graphics pictures.

Three submodules composes the buffer manager module. The first submodule receives requests from the decoder output and translate them into buffer insertion commands. It controls the start and end of a frame decoding operation. Buffer is allocated in the start of decoding operation and must be inserted in the reference list in the end of decoding procedure. The second submodule receives the requests from video output module and translate them into buffer removal commands. It also controls frame exhibition repetitions to adjust the video frame rate of the decoder to the video output module frame rate. This enables a 15 fps video to be exhibited into a 60 fps monitor, for instance. The third submodule centralizes the requests for buffer insertion and removal from the reference lists and the real state of the buffers in the DDR memory (allocated or not allocated).

This submodule provides different priorities for the reference list and buffer control operations when different clients request for buffer pointers. A request of a memory pointer to video exhibition has the highest priority, as the output timing is critical. Also, when the buffers are all occupied, a buffer must be released before it can be used in a request from the video decoder.

Buffer insertion and removal are not time-critical operations as they are executed one or two times in the time of a frame (33 ms). They are implemented as state-machines which scan the memories when a buffer must be inserted or removed from the reference list. Each operation can take up to 16 cycles due to the number of reference frames. The recovery of a buffer pointer based on a reference index is an operation that occurs one time to each 3 macroblocks. In this implementation, it takes one clock cycle.

4. Integration Issues

4.1. Integration Issues on the H.264 Video Decoder. The H.264 video decoder of this work is being developed as a collaborative design. The initial project specification established an architecture based on the algorithm structure described in [2]. The data transfers between IPs inside the video decoder were specified to follow the order in which data is produced by the decoding process, in three main formats: LoPs, parallel data, and sequential, as shown in Figure 5.

The decoder modules were developed simultaneously, following a reference software model [20]. They were validated with individual testbenches using bitstreams with the inputs and expected outputs, generated from a reference software. During integration phase, some issues required the redesign of some parts of the IPs of the video decoder. They can be grouped in two main categories: issues related to intermediate buffer insertion and issues related to data dependency. Although these issues may seem obvious at this stage of the project, they emerged during the system development due to the complexity of the design.

4.1.1. Issues Related to Intermediate Buffers. The decoder operates with a data flow regulated by the consumed frames in the output. The average rate in which the frames are consumed is determined by the exhibition rate. Frames are produced and consumed from the DDR in an irregular instantaneous rate, due to the channel multiplexing. This irregular rate propagates from the output to the input of the decoder by the full signal of the FIFOs. At the input side, the parser operation is irregular as it decodes an irregularly compressed input bitstream. If the parser is not fast enough, this produces an irregular operation in all the modules, as the FIFOs connected to the output of a given module pass by the full condition, or the FIFOs connected to the input of a module pass by the empty condition. This irregular rate can propagate from the input of the decoder to its output by the empty signal of the FIFOs. These two effects require that all modules of the decoder must be enabled or disabled in

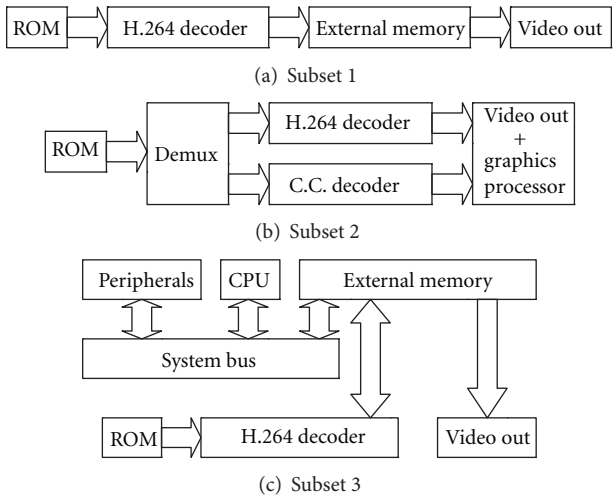

Figure 6: Subsets of SoC IPs used for incremental development.

some way by preceding or succeeding modules in the data flow chain.

IP integration required some interface adjusts to deal with these start/stop conditions, which can be grouped in two basic strategies. The first strategy consists of the insertion of an enabler signal in the IP. This strategy requires distribution of an enable signal within the IP and despite its conceptual simplicity, it requires considerable effort in debugging exception cases that can emerge due to incorrect synchronization of control and datapath operation with the enable signal. The second strategy consists of the inclusion of a buffer in the output of the module (on the level 0 of the hierarchy), disabling the module inputs when there is not enough space in the output buffer. This strategy assumes that the module process data in packets. When the output has enough space for one packet, the module processes one packet of data.

The first strategy enables smaller output buffers as it controls module operation with a finer temporal/data granularity than the second strategy. Nevertheless, it requires larger effort and development time. Also, it requires the access to the IP source code and detailed internal documentation. Access to an IP internals is not a severe issue when the IP is developed for a specific project, in a collaborative design, but it is a problem when applying this strategy in the integration of third party IPs. The second strategy requires a large buffer to store at least one data packet in the output. If this condition is not met, data loss can occur when the module maintains its operation and next module in the chain does not consume data from the buffer. This strategy does not require modifications in the interface and it is more suited to complex IPs. Both strategies were applied on the video decoder integration [21].

4.1.2. Issues Related to Data Dependency. The IPs composing the video decoder were designed with an initial functional specification of each module based on the operation of the corresponding part in the reference software. All local data dependencies were considered at design time. Nevertheless, due to the complexity of the algorithms, additional mechanisms were necessary to solve global unpredicted data dependencies and interactions between the internal decoder modules and I/O blocks on lower levels of the memory hierarchy. These data dependencies produced in the initial tests an incorrect behavior of the decoder due to the pipeline operation of some modules and to the macropipeline operation of the decoder.

Other issues are related to limitations of the platform used for hardware prototyping. As one example, Motion Compensation was designed with a cache considering an external memory interface with 320 bits wide data channel. This corresponds to 2.5 macroblock wide, and the cache must be fed by a specific arrangement of memory banks. The board used to prototype the decoder has a 64-bit wide data channel. The data conversion between MC and external memory is performed by the memory channel, designed to receive data in macroblocks and deliver it in parallel to MC cache, on the level 1 of the memory hierarchy. This strategy can be also applied to use standard off-the-shelf memory modules with a final SoC implementation in silicon. More information can be found in [21].

4.2. Integration Issues on the SoC

4.2.1. Issues Related to System Complexity. An SoC is generally a large and complex design. In the case of the SoC for the SBTVD, the presented H.264 video decoder is considered just one IP. Larger complexity implies in larger synthesis and validation time, increasing the development time.

Integration was planned to be performed as partial subsets of the final SoC. This strategy enabled to reduce developing time by working with smaller designs, thus reducing complexity, synthesis time and subsequently the verification effort. It also enabled distinct subteams to work in parallel, reducing development time.

The subsets of the SoC (shown on Figure 6) in the integration process can be enumerated as Subset 1, 2 and 3, and described as follows.

Subset 1. With the video decoder and a basic video output module with the multichannel memory controller, with test data fed by internal ROM; this subset enabled video decoder and decoded picture buffer testing and validation.

Subset 2. With the video decoder, graphics processor and demultiplexer, with test data fed by ROM; this subset enabled to test correct operation of the demultiplexer and graphics processor.

Subset 3. With the Leon3 CPU, the memory controller, and basic video controller; this subset enabled to develop and test applications on the CPU, the video frame buffer, and the development of a basic graphics library.

With these subsets, it was possible to start the software development before the end of the hardware integration process. All the subsets used the same interfaces between the corresponding modules, enabling to compose the final

(a) (b)

Figure 7: (a) Floorplanning and placed design of the H.264 video decoder of Subset 1; (b) on board demonstration of the prototype of Subset 3.

system as a composition of the subsets. The exception are the interfaces of the graphics processor and video controller used on the tests. The main difference is an input data channel for the closed caption data on the graphics processor, which displays only video if this channel is unused.

Large systems as this sometimes must to be prototyped in more than one development board. The interconnection between IPs on different boards must be designed to transfer data reliably avoiding problems related to clock differences between transmitter and receiver and skew between interface signals. The difference between clocks was treated with the use of a FIFO with different read and write clocks. As the data flows only from transmitter to receiver, the FIFO was located on the receiver and its potential consuming rate is larger than the producing rate of the transmitter. The skew between interface signals was treated by dimensioning the interconnections with the same length.

5. Results

5.1. Simulation Tests. The H.264 video decoder, using the described memory hierarchy and topology shown in Figure 3, was validated by simulation using video test sequences using Modelsim simulator. Simulation of the system shown in Figure 3 has limited application as the time necessary to simulate complex software (an operating system for instance) is very long.

5.2. On Board Tests. Board tests were initially performed on a XUPV2P board with a XC2VP30 Virtex II Pro device for individual tests with the video decoder and Leon3 CPU. As the size of the system increased it was migrated to a Xilinx ML509 board, with a XC5VLX110T Virtex-5 FPGA and 512 MB of external DDR2 SDRAM. The proposed architecture was synthesized using FPGA development tool ISE, from Xilinx. Synthesis results for the Virtex-5 are presented in Table 1.

On the ML509 board the proposed subsets were validated as follows.

(1) An Intra-only version of the decoder, with Multichannel Memory Controller configuration (Subset 1), decoding videos at 720p and 1080p.

(2) A baseline version of the decoder, with Multichannel Memory Controller configuration (Subset 1), decoding videos at 720p and 1080p. Currently the filter is also integrated, but it does not fit on the device with the decoder. This subset imposed a higher degree of effort to reach on board validation requiring the execution of manual floorplanning (shown on Figure 7(a)) before placement and routing.

(3) An Intra-only version of the decoder with demux and Closed Caption decoding (Subset 2).

(4) The SoC composed of the CPU Leon3, H.264 video decoder fed by a ROM and multichannel memory controller (Subset 3). The system runs with 50 MHz on the video decoder and CPU (shown on Figure 7(b)). The execution of an application

(5) Subset 1 using external (offboard) loopback between the ROM and the video decoder to test electrical interface between boards.

The interface with other boards, to transfer data for audio board and RF front-end, was developed and tested initially using a data source on the same FPGA which holds the video decoder and video output, passing data by a loopback connector, operating at 27 MHz. In a second step, the board with the video decoder was connected to a board with an RF demodulator and demux, with the link operating successfully. The audio decoder was not completely prepared for integration at the time of the realization of these experiments.

The SoC prototype with CPU (Subset 3) was tested with a small video being decoded in parallel with a small graphics

Table 1: Synthesis results for Xilinx XC5VLX110T FPGA.

	Slice Regs	Slice LUTs	BlockRAMs
PHY DDR2	2277	1749	3 (108 kb) L2
MMC	2714	2739	42 (1512 kb) L1
Parser	1346	3849	37 (6 kb) L0
MC	41411	23174	33 (381 kb) L0
Intra	2071	4164	3 (41 kb) L0
IqIt	5827	4792	3 (76 kb) L0
Filter	2254	2275	94 (5 kb) L0
Leon-3 CPU	4868	6618	29 (1044 kb) L0
Graphics processor	789	1331	13 (468 kb) L0

application created to test the video memory. This subset used a version of the H.264 decoder capable of decoding only frames with intraframe prediction. This version was chosen due to the limitation on the capacity of the XC5VLX110T device. When the migration to a platform with a larger FPGA is complete, the H.264 decoder will be replaced in the SoC prototype. The final goal is to develop an ASIC version of the SoC, after complete validation of the architecture on board.

The area overhead introduced in the output of MC module on level 0 is a buffer of two macroblocks, consuming 6.1 kb. The overhead in the channel of IF1 on level 1 to adapt the 128-bit port of PHY to 320-bit port of MC Cache is a buffer of 3 macroblocks, consuming 9.2 kb.

All individual modules were designed with an individual minimum performance which enables full HD decoding at maximum frequency of 100 MHz. The pipeline operation enables the system to decode one macroblock in 415 clock cycles for I frames and one macroblock in 384 clock cycles for P frames on the average. Latency is not a major concern in the decoder output as the number of cycles to decode one frame is up to 8160 times greater than one macroblock. Thus the system is capable of decoding 720p 30fps videos at 50 MHz and 1080p 30fps videos at a frequency near 100 MHz, depending on video content.

The measured performance showed that the memory elements on level 0 were correctly dimensioned. Attempts to reduce buffer sizing in certain elements caused the stop of the video decoding process due to data dependency.

In these tests, the DDR2 SDRAM memory controller runs at 200 MHz, the H.264 video decoder at 50 MHz and the output video controller runs at 110 MHz to generate 720p resolution or 148 MHz for 1080p. Tests in hardware with 1080p currently use the RPB capability to store a decoded picture and repeat it until the decoding process of the next one is finished.

6. Conclusions

In this work, we presented the development of a set-top box for Digital Television. It is a complex digital system integrating audio and video decoders and a CPU to run user interface and applications. A multiport memory interface with a memory hierarchy optimizes the off-chip memory usage.

The inclusion of this memory hierarchy has to be considered in the system design. It is known that buffering on lower levels of the hierarchy reduces potential bottlenecks in higher levels, where channel multiplexing is needed. Also, the sequential access characteristics of a video recovering task benefits from burst readings in DDR memory. Strategies were presented and analyzed to enable IP reuse and integration by buffering data and controlling processing on modules. The strategies proved to be effective, as the expected performance of the system was not reduced, and they introduced a small overhead in the architecture of the final system.

Complexity grows with the size of a system, and so does the time for design synthesis and validation. It has a strong impact on development time. In this work, the strategy of incremental integrations was crucial for reducing SoC complexity and verification effort, enabling to specify clear checkpoints in the development. With the execution of development and validation of independent parts of the system in parallel through independent on-board test of critical parts of the system, the strategy also enabled the development of the SoC in the available time for the project.

The shared memory in the higher levels enabled the integration of the video decoder with other elements, like an image processor to overlay an image from an access terminal over the decoded video or other systems through the insertion of more channels on the multichannel memory controller. The next step is the migration to a larger prototyping platform, with a device that can support the integration of the complete system.

Acknowledgments

This work is supported by CAPES and FINEP.

References

[1] K. Yang, C. Zhang, G. Du, J. Xie, and Z. Wang, "A hardware-software co-design for H.264/AVC decoder," in *Proceedings of the IEEE Asian Solid-State Circuits Conference (ASSCC '06)*, pp. 119–122, Beijing, China, November 2006.

[2] "ITU-T recommendation H. 264: advanced video coding for generic audiovisual services," Video Coding Experts Group, 2005.

[3] JEDEC, *JESD79-2F: DDR2 SDRAM Specification*, JEDEC Solid State Technology Association, Arlington, Va, USA, 2009.

[4] F. Pescador, G. Maturana, M. J. Garrido, E. Juarez, and C. Sanz, "An H.264 video decoder based on a DM6437 DSP," in *Proceedings of the International Conference on Consumer Electronics (ICCE '09)*, Las Vegas, Nev, USA, January 2009.

[5] C. C. Lin, J. W. Chen, H. C. Chang et al., "A 160K gates/4.5 KB SRAM H.264 video decoder for HDTV applications," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 170–181, 2007.

[6] ST SPEAr 1340 dual-core cortex A9 HMI embedded MPU, http://www.st.com/.

[7] A. C. Bonatto, A. B. Soares, A. Renner, A. A. Susin, L. M. Silva, and S. Bampi, "A 720p H.264/AVC decoder ASIC implementation for digital television set-top boxes," in *Proceedings of the 23rd Symposium on Integrated Circuits and Systems Design (SBCCI '10)*, pp. 168–173, September 2010.

[8] ABNT, *NBR15602 Digital Terrestrial Television—Video Coding, Audio Coding and Multiplexing*, ABNT, 2007.

[9] "Project Rede H. 264," http://www.lapsi.eletro.ufrgs.br/h264/wiki/tiki-index.php.

[10] "Project SoC-SBTVD," http://www.lapsi.eletro.ufrgs.br/soc-sbtvd/wiki/tiki-index.php/.

[11] ABNT, "NBR15604 digital terrestrial television—receivers," ABNT, 2007.

[12] "Gaisler research Leon3 processor," http://www.gaisler.com .

[13] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 704–716, 2003.

[14] A. Renner and A. Susin, "An MPEG-4 AAC decoder FPGA implementation for the Brazilian Digital Television," in *Proceedings of the 8th Southern Conference on Programmable Logic (SPL '12)*, pp. 1–6, Bento Gonçalves, Brasil, March 2012.

[15] P. Van Der Wolf and T. Henriksson, "Video processing requirements on SoC infrastructures," in *Proceedings of the Design, Automation and Test in Europe (DATE '08)*, pp. 1124–1125, ACM, Munich, Germany, March 2008.

[16] C. H. Li, C. H. Chang, W. H. Peng, W. Hwang, and T. Chiang, "Design of memory sub-system in H.264/AVC decoder," in *Proceedings of the Digest of Technical Papers International Conference on Consumer Electronics (ICCE '07)*, pp. 1–2, January 2007.

[17] Xilinx, http://www.xilinx.com/.

[18] A. C. Bonatto, A. B. Soares, and A. A. Susin, "Multichannel SDRAM controller design for H.264/AVC video decoder," in *Proceedings of the 7th Southern Conference on Programmable Logic (SPL '11)*, pp. 137–142, Córdoba, Argentina, April 2011.

[19] B. Zatt, A. Azevedo, L. Agostini, A. Susin, and S. Bampi, "Memory hierarchy targeting bi-predictive motion compensation for H.264/AVC decoder," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07)*, pp. 445–446, Porto Alegre, Brazil, March 2007.

[20] "H.S. Coordination," JM Software, http://iphome.hhi.de/suehring/tml.

[21] A. B. Soares, A. Bonatto, and A. Susin, "Integration issues on the development of an H.264/AVC video decoder SoC for SBTVD set top box," in *Proceedings of the 24th Symposium on Integrated Circuits and Systems Design (SBCCI '11)*, August-September 2011.