Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

Reconfigurable video coding framework and decoder reconfiguration instantiation of AVS☆

Dandan Ding[a,*], Honggang Qi[b], Lu Yu[a], Tiejun Huang[c], Wen Gao[c]

[a] *Institute of Information and Communication Engineering, Zhejiang University, Hangzhou 310027, China*
[b] *School of Information Science and Engineering, Graduate University of Chinese Academy of Sciences, Beijing, China*
[c] *School of Electronics Engineering and Computer Science, Peiking University, Beijing, China*

A R T I C L E I N F O

Article history:
Received 31 May 2008
Received in revised form
10 November 2008
Accepted 19 December 2008

Keywords:
RVC
Decoder configuration
AVS
VTL
BSDL

A B S T R A C T

In 2004, a new standardization activity called reconfigurable video coding (RVC) was started by MPEG with the purpose of offering a framework which provides reconfiguration capabilities for standard video coding technology. The essential idea of RVC framework is a dynamic dataflow mechanism of constructing new video codecs by a collection of video coding tools from video tool libraries. With this objective, RVC framework is not restricted to specific coding standard, but defined at coding tools level with interoperability to achieve high flexibility and reusability. Three elements are normative in RVC framework: decoder description (DD), video tool library (VTL) and abstract decoder model (ADM). With these elements, a standard or new decoder is able to be reconfigured in RVC framework. This paper presents the procedure of describing a reconfigured decoder in DD, reusing and exchanging tools from VTLs and initializing ADM in the dataflow formalism of RVC framework. A decoder configuration which can be instantiated as AVS intra decoder configuration or other new decoder configurations in RVC framework is described as an example by using coding tools from China audio video coding standard (AVS) and MPEG series. It is shown that the process mechanism offered by RVC framework is versatile and flexible to achieve high reusability and exchangeability in decoder configurations.

1. Introduction

New video coding technologies have been developed rapidly over the past two decades. Many standards have been explored to provide different video coding solutions for the market. For instance, H.264/AVC was standardized in May 2003 and audio video coding standard (AVS) was published as Chinese national standard in February 2006. Variety of the coding standards pushes multimedia devices to support multiple codecs, such as MPEG-2, H.264/AVC and AVS, in order to satisfy different needs of industry. However, it is complicated for multimedia devices to support all kinds of codecs. Moreover, with the development of video coding, more coding technologies will be adopted in the future. Those new technologies can be new coding standards or specific coding methods such as interpolation, inverse transform, and so on. It is difficult for multimedia devices which conform to some certain standards to adapt themselves to the new technologies. An obvious case is from IPTV, there clearly exists the transition from one codec to another in a single channel. As the size of video resolution evolves from SD to HD (and may be HD to SHD, in the future), this transition from one codec to another will be a common phenomenon in IPTV. The current usage of multi-codec-in-a-channel is

☆ This work was supported in part by the National Science Foundation of China under Grants 90207005 and the Major State Basic Research Development Program of China (973 Program) under Grants 2009CB320900.

* Corresponding author.
E-mail address: dandan2036@163.com (D. Ding).

0923-5965/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2008.12.002

limited in that the transition of one codec to another is strictly scheduled [1]. A simple and robust solution is required to allow more flexible usage of codecs transmission in a single channel.

At the same time, multimedia applications are growing much faster than the standardization of new coding strategies. New requirements from applications challenge the traditional video coding standards. For example, in the present video market, as the video coding standards are defined at codec level and there is no negotiation between them, compressed data conforming to one video coding standard cannot be decoded by decoders of other standards. Thus, as is often the case, video resources from different video coding standards cannot be adequately shared. Potential applications in video coding market should also be cared. A typical example is that, some tools and syntax elements from some standards such as H.264/AVC are not so useful in some actual applications but the decoder has to support them in order to keep itself as a "legal" one. The encoder is also obliged to produce bitstreams conforming to a standard even some syntax elements are never used. If the encoder discards those unnecessary syntax elements to produce a compact bitstream, currently, this bitstream cannot be decoded correctly because the decoder does not know what happens to it unless a dedicated decoder for this encoder is designed. Ref. [4] shows another example which is for low-complexity or high-performance reason, the encoder is possible to employ some new coding tools and needs to communicate with the decoder to guarantee a consistent decoding process. Obviously, negotiation between the encoder and decoder is expected to serve this kind of applications flexibly and dynamically.

For solving those problems, MPEG developed reconfigurable video coding (RVC) based on coding tool level not only to support multiple video codecs but also to allow the interoperability between different video codecs to include improvements and innovations. The essential idea of RVC is to introduce an interoperable model at coding tool level and allow flexible reconfiguration of those coding tools. Current codec level definition of video coding standards restricts the implementations to some profiles of a specific standard which lacks flexibility and does not allow interoperability between different codecs. Additionally, as many of the existing standards share common or similar coding tools, it is more convenient to exploit such commonalities in coding tool level [2]. To achieve this goal, RVC provides a framework allowing a dynamic development, implementation and adoption of standardized video coding solutions with features of higher flexibility and reusability [3]. Reconfiguration of RVC is no longer kept in codec level of a standard. It could be a new configuration of coding tools from different profiles of one standard, a hybrid configuration with coding tools from different standards, and so on. Moreover, RVC supports the introduction of new coding tools in order to speed up the standardization of new technologies and achieve specific design or performance trade-offs to satisfy application constraints. It allows a completely novel configuration with newly developed coding methods or a combination of existing and new coding tools.

The coding tool level definition of RVC framework allows flexible reconfiguration of coding tools to create different codec solutions on-the-fly. Video tool library (VTL) is used in RVC to collect coding tools which is also called functional units (FUs). To satisfy the increasing requirements better and explore more potential needs of industry, RVC framework supports different VTLs, such as MPEG VTL, AVS VTL and other VTLs to improve its capability. As a new standard developed in recent years, AVS can be applied in IPTV, mobile TV and other related applications in video field. Based on RVC framework, AVS VTL is built which includes a collection of AVS coding tools and conformance test for those tools. Participation of AVS in RVC enriches the significance of RVC framework and brings affirmative impulse for video coding field. This paper takes coding tools from AVS and MPEG VTL for example to present the procedure of reconfiguring a decoder in RVC framework and how to explore the reusability and exchangeability for decoder configurations.

In this paper, the reconfiguration formalism of RVC framework which is based on the dataflow model built from coding tools of VTLs is first introduced. It is shown that the process mechanism and management strategy adopted in RVC framework provide great flexibility and reusability for designers to reconfigure new decoding solutions according to application constraints. Then the procedure of constructing decoder configurations in RVC framework is described. First, based on the syntax description for AVS intra decoder, it illustrates how to employ the method defined in RVC framework to describe a reconfigured bitstream flexibly. According to the bitstream syntax description, a dedicated syntax parser for the bitstream can be generated automatically. Secondly, granularity of coding tools from VTLs such as AVS and MPEG VTL is analyzed and a partition method is realized to explore reusability and exchangeability as much as possible for the proposed decoder configuration. Finally, the architecture for the proposed decoder configuration example is constructed by combining the syntax parser and coding tools from AVS and MPEG VTL. An instantiated implementation and simulation of the architecture which serves as an AVS intra decoder configuration are shown to explain the dataflow of this decoder model. It shows that RVC framework is capable of reconfiguring decoders theoretically and practically.

The remaining of the paper is organized as follows: RVC framework is described in Section 2. Section 3 introduces the AVS standard and its coding tools. Section 4 proposes an example to show the procedure of reconfiguring a decoder in RVC framework which includes the decoder description (DD) process, the coding tool partition with flexible reusability and exchangeability, as well as the final architecture for the decoder configuration example. Section 5 presents implementation of the architecture and dataflow model of the decoder configuration example which is instantiated as an AVS intra decoder configuration in RVC simulation environment. Section 6 concludes the paper. Finally, the future work is prospected in Section 7.

2. RVC framework

RVC offer a flexible mechanism of combing coding tools to reconfigure decoding solutions. RVC framework adopts dataflow process formalism to modular designs which is different from traditional design methods. In order to describe the modular configurations in decoder, as shown in the RVC conceptual diagram (Fig. 1), DD which contains information of bitstream syntax and FU Network is transmitted to decoder together with the encoded video data. In decoder, three elements are normative in RVC framework, DD, VTL, and abstract decoder model (ADM) (Fig. 3). The DD, using FU network language (FNL) to specify connections of coding tools from VTLs and the profiled bitstream syntax description language (BSDL) which is called "RVC-BSDL" to describe the reconfigured bitstream syntax structure, initiates ADM as a basic model to set up a decoding solution. Further introduction of those elements is presented below.

2.1. Video tool library

The VTL collects video coding tools which are named as FU in RVC. Each FU is a well self-contained modular element with the specification of its I/O interfaces. A subset of the dataflow actor-oriental language CAL is specified in RVC framework (called RVC–CAL) to describe the algorithm and behavior inside FU. Compared with the traditional C/C++ programming language, CAL is clearly portable and can be used on wide variety of different platforms [6]. In that way, the ability of FUs to perform computation depends on the availability of sufficient input tokens. Communication between FUs is implemented by sending each other input and output tokens, which leads to high modularity and compliance with the principle of RVC. Fig. 2 shows the basic structure of an FU which includes two units. The "Processing Unit" describes the internal computation of FU and the "Context-Control Unit" processes the context signal which is intermediate variable of decoding and the control signal which is directly decoded from the input bitstream.

2.2. Decoder description

2.2.1. FU network description

As described above, FUs communicate with each other through input and output tokens. However, for FU itself, it does not know whether it is involved in the reconfiguration process or not. Also, it knows neither the source of its input tokens nor the destination of its output tokens. FNL is exactly the language used for FU network description (FND) to specify FUs involved in the decoder configuration as well as connections between them. In RVC framework, FNL is a XML dialect and packaged in the DD bitstream. When it is received, a DD decoder procedure is called immediately to initialize the interconnection of assigned FUs. An example in sub-clause 4.2 explains how FNL is used to describe network of FUs.

2.2.2. Bitstream syntax description

Bitstream syntax description (BSD) is the description for syntax structure of a bitstream that is supposed to be decoded by a decoder composed from an FND. It is used to generate or instantiate a syntax parser FU from generic syntax parser mechanism [8]. The BSDL from MPEG-21 [9] is used to describe the BSD and the bitstream syntax schema described in BSDL is defined as bitstream syntax schema (BS schema).

There are two kinds of bitstreams in RVC: one is bitstreams exactly conforming to an existing standard, and the other is reconfigured bitstreams which are different from any existing standard. For the first case, the syntax parser for current bitstream can be modularized as generic bitstream syntax parser FU in VTLs and the BSD may be optional. However, for the latter case, the bitstream syntax parser FU is not fixed because bitstream syntax structure of the reconfigured bitstream is in a new format which is different from any existing standards. Thus for this kind of situations, the syntax parser should be generated according to BSD. To facilitate the synthesis of parsers from a BS schema, RVC framework profiles a

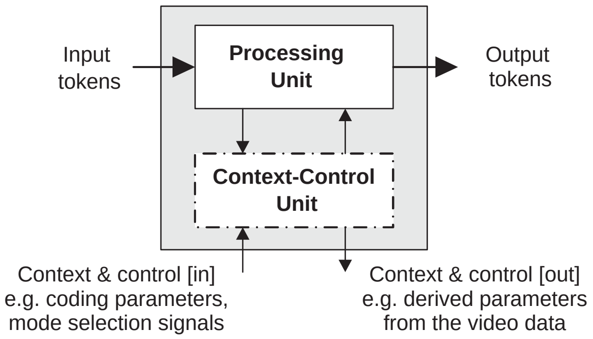

Input tokens
Processing Unit
Output tokens
Context-Control Unit
Context & control [in] e.g. coding parameters, mode selection signals
Context & control [out] e.g. derived parameters from the video data

Fig. 2. Basic structure of an FU in RVC VTL [7].

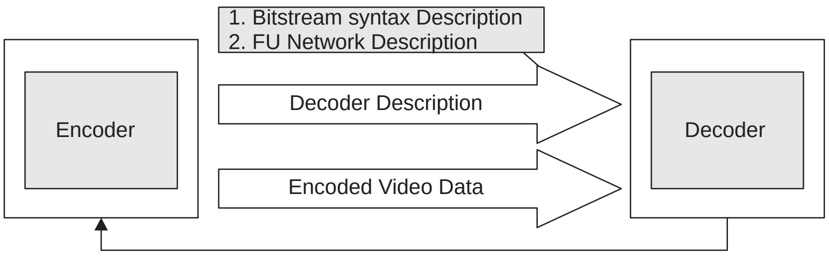


Fig. 1. Conceptual diagram of RVC [8].

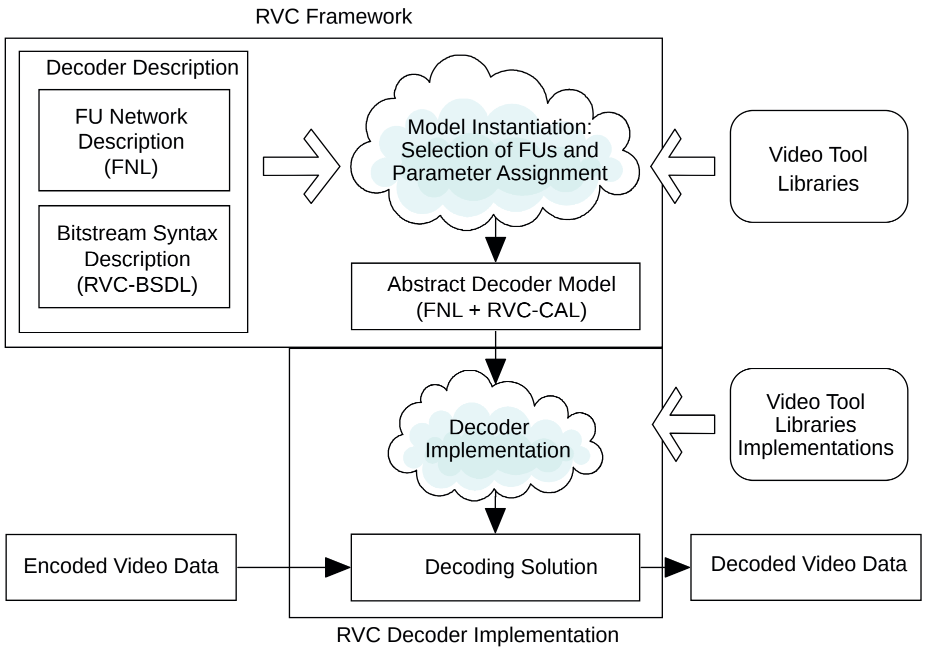

RVC Framework
Decoder Description
FU Network Description (FNL)
Bitstream Syntax Description (RVC-BSDL)
Model Instantiation: Selection of FUs and Parameter Assignment
Video Tool Libraries
Abstract Decoder Model (FNL + RVC-CAL)
Decoder Implementation
Video Tool Libraries Implementations
Encoded Video Data
Decoding Solution
Decoded Video Data
RVC Decoder Implementation

Fig. 3. Components and instantiation process for ADM and the platform-dependent decoder implementation in RVC framework.

version of BSDL called RVC–BSDL which includes extension and usage restrictions based on BSDL. RVC–BSDL provides all necessary information for the complete parsing of any bitstream. According to RVC BS schema for a reconfigured bitstream, corresponding syntax parser FU can be generated automatically so as to build a complete decoder configuration.

2.3. *Abstract decoder model*

With those elements mentioned above, RVC framework initiates the ADM which is a behavioral model of decoder configuration composed by FUs and their relative connections. The ADM will be used to implement the decoder using proprietary tools and mechanisms. As shown in Fig. 3 [8], the decoder implementation is platform-dependent and regarded as non-normative part of RVC framework. It can be generated by substituting any implementation of proprietary VTLs which own identical I/O behavior of the FUs in standard VTLs, or also obtained directly from the ADM by generating SW or HW implementations by means of appropriate synthesis tools [22,23].

3. China audio video coding standard

China audio video coding standard which is referred as AVS [13] is a digital audio and video coding standard developed by AVS Workgroup of China, whose video part (AVS part 2) has been adopted as the national standard of China in February 2006. AVS has been developed for several years and can be applied in many video coding applications.

Similar with MPEG-x and H.26x series of standards, AVS also owns a block-based hybrid coding framework. It adopts the spatial and temporal predictions to eliminate the spatial and temporal data redundancy, and the prediction residuals are transformed, quantized and entropy encoded. However, the internal technologies are different from those of previous standards.

The major coding tools of AVS are 8×8 integer transform, 8×8 spatial prediction, 8×8 motion compensation, in-loop deblocking filter and context-based adaptive 2-dimensional variable length coding (CA-2D-VLC) [14]. The pre-scaled integer transform (PIT) technique is used for providing a unique specification for the finite precision implementations and yielding significant saving in processing complexity [15]. AVS adopts 8×8 block-based intra prediction which has 5 luminance prediction modes and 4 chrominance prediction modes. The reconstructed pixels of neighboring blocks before deblocking filter are used as reference pixels for the intra prediction of current block. For the motion compensation of AVS, a 16×16 pixel macroblock can be partitioned into 16×8, 8×16 or 8×8 block. Symmetric mode is introduced to exploit motion continuity in sequential pictures and this mode can efficiently save bits in coding motion vector. For the sub-pixel interpolation, compared with H.264/AVC, the data bandwidth requirement of AVS is reduced by 11%, whereas the computation complexity remains similar [16]. In the deblocking filter, the block boundaries are 8×8 aligned and the number of block boundaries, boundary strength levels and pixels in the deblocking filter are reduced for simplicity. In the entropy coding, the code words for all syntax elements are constructed based on Exponential–Golomb codes or fixed-length codes and an efficient context-based adaptive 2D-VLC entropy coding method is designed for coding 8×8 block-size transform coefficients. These coding tools are harmonized optimally in AVS to achieve a reasonable trade-off between high performance and low complexity.

4. Procedure of decoder configuration in RVC framework

The process mechanism of RVC framework introduced in Section 2 owns high flexibility and interoperability for general decoder configurations. In this section, the decoder configuration procedure in RVC framework is described. For the bitstream syntax description, it explains how to simply define a BS schema for a reconfigured bitstream by taking bitstream syntax description of AVS for example. It also depicts how to validate BS schema by BSDL parsers to ensure that the automatically generated syntax parser is correct. For the FUs of VTLs, with the reasonable FU partition and input and output token specifications, high reusability and exchangeability can be achieved for the reconfigured decoders. That is shown by analyzing FU partition granularity of AVS VTL which collects coding tools from AVS and MPEG VTL which collects coding tools from MPEG series of standards. Finally, by combining syntax parser and FUs from VTLs, concrete architecture of the proposed decoder configuration example is constructed with flexible reusability, exchangeability and extension ability.

4.1. BS schema definition and validation

RVC–BSDL provides a way to create schema for a bitstream and instantiate new parser FU that cannot be presented in VTLs. In this section, based on the example of BS schema for AVS intra decoder configuration, it explains how to fully describe the syntax structure of a reconfigured bitstream. Then the BS schema is validated to ensure a consistent description of input bitstream.

Fig. 4 shows a segment from AVS bitstream syntax specification. It is flexible in BS schema to specify the structure of a reconfigured bitstream depending on different conditions. In Fig. 5, [a] is a dedicated description of AVS intra decoder configuration, whereas, for the reconfigured bitstream, the encoder is autonomous enough to rearrange the bitstream syntax structure as long as this is reflected in the BS schema. For example, in Fig. 5[b], if the encoder believes that syntax element "pred_mode_flag" is not useful, it can be ignored in the BS schema. Even the data types of syntax elements can be changed depending on different implementations, such as the syntax element "intra_chroma_pred_mode" in Fig. 5[b]. The simple and direct approach to reconfigure a bitstream offered by BS schema allows the encoder to devise its own specific products according to applications.

A BS schema should be validated to make sure it is consistent with the corresponding bitstream. In RVC, BSDL parsers, including BintoBSD and BSDtoBin are used to validate BS schema. The BintoBSD parser is a generic processor using a BS schema to parse a bitstream and generate the corresponding BS Description. Inversely, the BSDtoBin parser is a generic processor using a BS schema to parse BS description and generate the corresponding bitstream [9].

Fig. 6 shows an example of AVS intra BS schema to illustrate the validation procedure. A fragment of AVS intra BS schema is shown in Fig. 7 which presents part of bitstream syntax description for AVS. The BintoBSD parser inputs the AVS BS schema and the bitstream generated by AVS reference software, and produces the BS description of AVS in XML format. The BSDtoBin progressively parses the BS description with BS schema of AVS and generates the output bitstream by encoding the element values according to their data types and appending them in to bitstream.

For a correct BS schema, two possibilities are used to check the consistency [8]:

(1) The "Input Bitstream" is compared with the "Generated Bitstream" produced by the operation "BintoBSD-BSDtoBin" to make sure it is the same as the latter one.
(2) The "BS Description" created after the first "BintoBSD" operation is compared with the "Generated BS Description" created by the operation "BSDtoBin–BintoBSD" to make sure it is the same as the latter one.

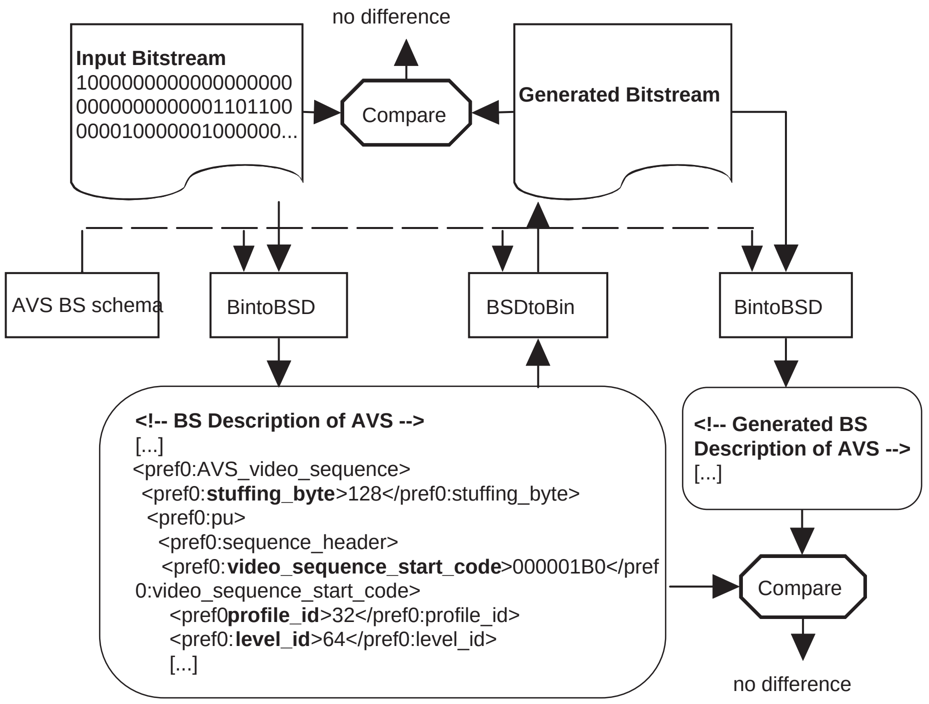
macroblock() {	Description
if (PictureType != 0 || (PictureStructure == 0 && MbIndex >= MbWidth × MbHeight / 2))	
mb_type	ue(v)
if (MbType != 'P_Skip' && MbType != 'B_Skip') {	
if (MbType == 'B_8x8') {	
for (i=0; i<4; i++)	
mb_part_type	u(2)
}	
if (MbType == 'I_8x8') {	
for (i=0; i<4; i++) {	
pred_mode_flag	u(1)
if (! pred_mode_flag)	
intra_luma_pred_mode	u(2)
}	
intra_chroma_pred_mode	ue(v)
[…]	

Fig. 4. A segment taken from AVS bitstream syntax specification [13].

a
```
 <xsd:element name="macroblock">
  <xsd:complexType>
   <xsd:sequence>
    <!-- element "mb_type" is ignored because it is not used! -->
    <!-- element "mb_part_type" is ignored because it is not used! -->
    <xsd:sequence minOccurs="0" bs2:if="$MbType = 0">
     <xsd:sequence bs2:nOccurs="4">
      <xsd:element name="pred_mode_flag" type="bs1:b1" bs0:variable="true"/>
      <xsd:sequence minOccurs="0" bs2:if="$avs:pred_mode_flag = 0">
       <xsd:element name="intra_luma_pred_mode" type="bs1:b2"/>
      </xsd:sequence>
     </xsd:sequence>
     <xsd:element name="intra_chroma_pred_mode" type="avs:expGolomb"/>
[...]
```

b
```
 <xsd:element name="macroblock">
  <xsd:complexType>
   <xsd:sequence>
    <!-- element "mb_type" is ignored because it is not used! -->
    <!-- element "mb_part_type" is ignored because it is not used! -->
    <xsd:sequence minOccurs="0" bs2:if="$MbType = 0">
     <xsd:sequence bs2:nOccurs="4">
       <!-- element "pred_mode_flag" is ignored -->
       <xsd:element name="intra_luma_pred_mode" type="bs1:b2" fixed="0"/>
     </xsd:sequence>
     <xsd:element name="intra_chroma_pred_mode" type="bs1:b2"/>
[...]
```

Fig. 5. An example of defining different BS schemas for different reconfigured bitstreams used for different application scenarios.



```
<!-- BS Description of AVS -->
[...]
<pref0:AVS_video_sequence>
 <pref0:stuffing_byte>128</pref0:stuffing_byte>
  <pref0:pu>
   <pref0:sequence_header>
    <pref0:video_sequence_start_code>000001B0</pref
0:video_sequence_start_code>
     <pref0profile_id>32</pref0:profile_id>
     <pref0:level_id>64</pref0:level_id>
     [...]
```

Fig. 6. An example of validating AVS intra BS schema to illustrate the validation procedure of BS schema by BSDL parsers: BintoBSD and BSDtoBin.

Both items above are used to check the BS schema and Fig. 6 shows the obtained results. According to the correct BS schema, syntax parser FU such as the syntax parser for AVS intra decoder configuration in this example can be generated automatically. A possible systematic procedure for the automatic synthesis of RVC parsers can be found in Refs. [17–19], which will not be specified in this paper any more.

```
<xsd:element name="bitstream">
 <xsd:complexType>
  <xsd:sequence>
    <xsd:element name="AVS_video_sequence" minOccurs="0" maxOccurs="unbounded">
        [...]
            <xsd:choice>
              <xsd:element ref="avs:sequence_header" bs2:ifNext="000001B0"/>
              [...]
            </xsd:choice>
        [...]
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute ref="bs1:bitstreamURI"/>
 </xsd:complexType>
</xsd:element>

<xsd:element name="sequence_header">
 <xsd:complexType>
   <xsd:sequence>
     <xsd:element name="video_sequence_start_code" type="avs:StartCodeType"/>
     <xsd:element name="profile_id" type="xsd:unsignedByte"/>
     <xsd:element name="level_id" type="xsd:unsignedByte"/>
     [...]
   </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Fig. 7. A fragment from AVS BS schema which describes the syntax structure of AVS intra bitstream.

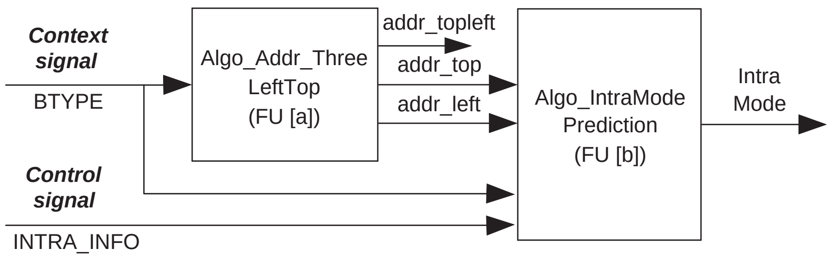


Fig. 8. An example of FU partition for intra mode prediction to reuse FU among H.264/AVC, MPEG-4 and AVS.

4.2. FU reusability and exchangeability of video tool library

In VTL, video tools are in the form of FUs with input and output token specifications. An important idea for RVC is to achieve high reusability and exchangeability of FUs. This principle spirit leads to partition FUs of VTLs in a reasonable granularity [10]. After analyzing coding tools from AVS and MPEG, FUs proposed in this decoder configuration example are partitioned in the similar granularity to the FUs already implemented to reuse them as much as possible in order to avoid redundant implementations. Intra prediction algorithms of H.264/AVC and AVS are used as an example to demonstrate the reusability and exchangeability in decoder configuration.

In the example of intra prediction configuration, for AVS, intra mode of current 8×8 luminance block is predicted according to syntax elements decoded from bitstream and information coming from neighboring left and top block. It can be efficiently implemented if just input all related information decoded from bitstream and leave all processes inside one FU. However, problem is that this way of partition makes little sense in RVC, because the FU can only be used by AVS. It has been noticed that 8×8 intra mode prediction process of H.264/AVC is also related to the neighboring left and top block. Moreover, inverse DC prediction of MPEG-4 refers to the left, top and top-left neighboring block of current block too. The similar processes make it possible to share the same FUs among AVS, H.264/AVC and MPEG-4 decoder configuration. Thus, intra mode prediction process is partitioned into two FUs, one is exactly a common FU for AVS, H.264/AVC and MPEG-4 (Fig. 8[a]), and the other is used for intra mode prediction (Fig. 8[b]). As information of neighboring blocks, which is used for mode prediction, is stored in a buffer, the common FU “Algo_Addr_ThreeLeftTop” is designed to calculate the buffer addresses of neighboring blocks. Considering address of the top-left block is not used in AVS and H.264/AVC, token “addr_topleft” is not connected in this example. When the FU “Algo_IntraModePrediction” receives the addresses, it reads out related information and predicts the intra mode of current block. Though this

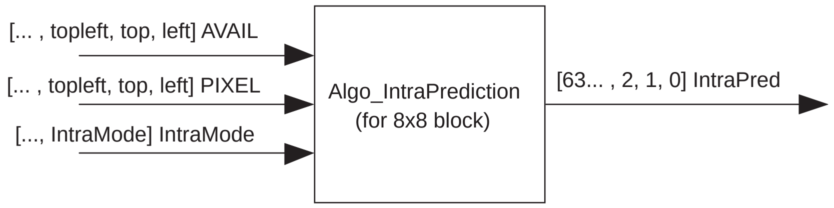

[... , topleft, top, left] AVAIL
[... , topleft, top, left] PIXEL
[..., IntraMode] IntraMode
Algo_IntraPrediction (for 8x8 block)
[63... , 2, 1, 0] IntraPred

Fig. 9. An example of exploring FU exchangeability for intra prediction of H.264/AVC and AVS with consistent token specifications in RVC framework.

partition method requires more tokens (three extra tokens: "addr_topleft", "addr_top" and "addr_left") in reconfiguration process, it is more significant to obtain the reusability and exchangeability between FUs for efficient reconfiguration.

Exchangeability is also considered in the decoder configuration for convenience of extending or replacing FUs to achieve better trade-offs between performance and complexity. Literature [4,5] shows an example of reconfiguring a new decoder by combining inverse quantization and inverse transform tools from AVS and other tools from MPEG-4 simple profile. Complexity of the new decoder is reduced and the performance is improved significantly at high bitrate. Another typical example about exchangeability of decoder configuration is shown in this paper.

In Fig. 8, the reason to design FU "Algo_IntraModePrediction" in this way is to reach high exchangeability between AVS and H.264/AVC [11] because they both have the same token requirements for intra mode prediction. It is similar in the intra prediction process shown in Fig. 9, in which the input tokens of FU "Algo_IntraPrediction" is strictly specified: if the left neighboring pixels of current block are available, token "AVAIL" outputs value of "true" for left pixels and 16 left pixels are output to the token "PIXEL" channel. Or else, token "AVAIL" outputs value of "false" to indicate that the left neighboring pixels of current block is not available, and correspondingly, there are no left pixels in the token "PIXEL" channel. After the left token, the same specification is used to the top and top-left case. Also, the output token "IntraPred" is defined in horizontal raster order. With the token specifications, FUs with the same or similar token requirements but different internal algorithms can be easily invoked to replace each other in this network. For example, though in H.264/AVC intra prediction, it only requires 8 left pixels, the exchangeability of FU "Algo_IntraPrediction" between H.264/AVC and AVS is not influenced because the second 8 left pixels can be directly ignored in prediction.

4.3. *Decoder configuration architecture*

The architecture of the proposed decoder configuration example includes three networks: "Parser", "Decoding" and "Motion compensation". Though there is no motion compensation in intra configuration, the name "Motion compensation" is still reserved here because this network will be expanded to support inter configuration in the future. Each network consists of related FUs from VTLs. Table 1 [12] lists detailed contents of the architecture and networks.

Table 1
FUs list for example of AVS intra decoder configuration.

Network name	FUs name	Reuse/exchange in MPEG-4
Parser	Algo_Synp	
	BlockExpand	*R*
	BTYPESplit	R
	BlockSplit	R
	INTRA_INFOSplit	*R*
Decoding	Algo_ISzigzagOrAlternative	E
	Algo_IQ	
	Algo_IT8 $\times$ 8_1d	
	Algo_IntraModePred_LUMA	
	Algo_IntraPred_LUMA_8 $\times$ 8	
	Algo_IntraModePred_CHROMAZ	
	Algo_IntraPred_CHROMA_8 $\times$ 8	
	Algo_transpose8 $\times$ 8	R
	Mgnt_IntraPred_LUMA_Addr	*R*
	Mgnt_IntraPred_CHROMA_Addr	*R*
	Algo_Addr_ThreeLeftTop	R
Motion compensation	Algo_ADD	R
	Mgnt_FB	R
	Mgnt_Reconstruct_LUMA_Addr	*R*
	Mgnt_Reconstruct_CHROMA_Addr	*R*
	Mngt_PixelMerger_420	R

Note:
(1) "R" indicates the proposed decoder configuration example reuses that FU from MPEG-4 simple profile.
(2) "*R*" indicates the proposed decoder configuration example reuses that FU from H.264/AVC.
(3) "E" indicates that these FUs in AVS and H.264/AVC have the same tokens requirement, and can replace each other in the proposed decoder configuration.

With the partition method mentioned above, FUs used in the proposed architecture are divided into reasonable granularity to reuse existing FUs. The FUs listed in Table 1 fall into two categories: FUs oriented to algorithm and FUs oriented to data management [20]. The proposed decoder configuration reuses 7 FUs from MPEG-4 simple profile and 6 FUs from H.264/AVC and avoids duplicate definitions and implementations efficiently.

Result shows that the proposed FUs partition method in this example not only improves reusability of existing FU, but also provides sufficient flexibility to exchange FUs with different algorithms. Still in the case of the proposed decoder configuration example, it is shown that 7 FUs of H.264/AVC and AVS characterize the same token requirements and can be conveniently and directly replaced by each other.

5. Implementation and simulation for the proposed decoder configuration

All FUs used in the proposed decoder configuration example are specified in RVC–CAL language. In implementation, the proposed decoder configuration example can be instantiated as an AVS intra decoder or other hybrid decoders in RVC framework. In the example shown below, it is configured as an AVS intra decoder by using corresponding coding tools from AVS VTL besides reusing FUs from MPEG VTL. The reconfigured AVS intra decoder is compiled with RVC simulation tool and result of simulation shows that it can decode AVS intra conformant bitstreams correctly and elegantly.

5.1. Parallel and serial dataflow model

In implementation, the decoder reconfiguration can be structured as parallel and serial dataflow model in RVC simulation environment, separately. Difference between the parallel and serial version is the content of token channels. For the parallel version, Y, U and V tokens are processed in respective channel, whereas for the serial version, Y, U and V tokens are combined in one channel sequentially. For example of the 420 colour format, in the parallel channel, tokens of block Y0, Y1, Y2, Y3 are sequentially arranged in Y channel, tokens of bock U0 are arranged in U channel, and the same with tokens of bock V0 in V channel (Fig. 10[a]), whereas in the serial channel, tokens are in the order of block Y0, Y1, Y2, Y3, following tokens of block U0 and V0 (Fig. 10[b]). The reason of constructing parallel and serial version is that it is useful for providing the SW and HW generator more efficient synthesis tools for ADM. From the implementation point of view, depending on which SW or HW the user want to generate for the decoder, either serial or parallel version of the FUs can be used for the ADM [21]. For the proposed configuration, both parallel and serial version of FUs are available in VTL in order to have the maximum flexibility in implementation.

5.2. FU network description

FNL is used in the decoder configuration to connect FUs into a network. How the decoder configuration can be described in terms of a network of FUs is shown in this section. The network in Fig. 8 composed of 2 FUs is taken as an example to illustrate the connection specification. By connecting FUs into networks and then connecting those networks, a full decoder configuration is established.

Fig. 11 describes the network specification of FUs in Fig. 8. The root element "XDF" is used to identify a FU network which includes kinds of elements to describe its structure. The element "Package" contains a structured representation of a qualified identifier (QID) which provides the name attribute of the network [8]. The element "Port" declares I/O tokens of the network. "Instance 1" and "Instance 2" indicate FUs involved in the network with internal parameters. Directions of tokens are declared at last to specify the source and destination to make sure the seamless dataflow between FUs and networks.

5.3. Decoder configuration

Figs. 12 and 13 show the implementation architecture of the proposed decoder configuration example which functions as an AVS intra decoder. The whole decoder consists of three networks: "Parser", "Decoding" and "Motion compensation". Fig. 12 describes the "Parser" network and Fig. 13 shows the Y component implementation architecture of "Decoding" and "Motion compensation". For U and V component, they are processed in the similar way as Y.

The input bitstream is serialized bit by bit through FU "Serialize" before decoding process. The "Parser" network decodes the coming bitstream and produces context-control signal and a sequence of data for each block which will be used in the following networks. The syntax parser "Algo_synp" which can be generated

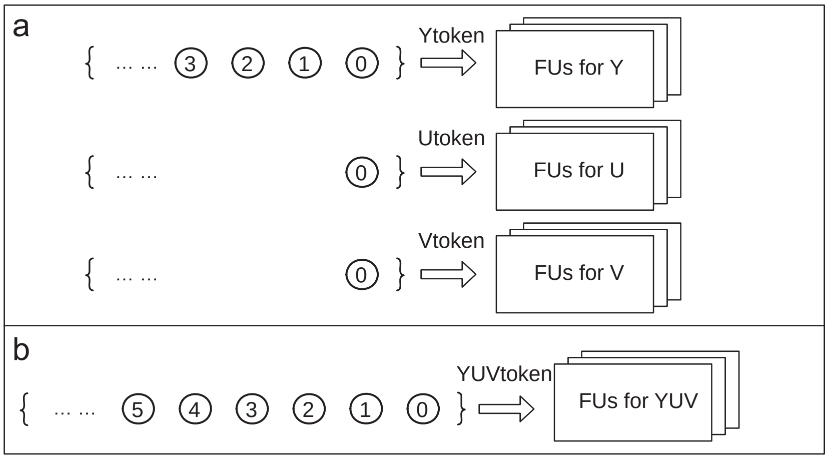


Fig. 10. Difference between parallel [a] and serial [b] dataflow model for instance of the 420 colour format.

```
<XDF name="IntraModePrediction">
 <Package>                                                                         [a]
  <QID>
   <ID id="AVSIntra"/>
  </QID>
 </Package>
 <Port kind="Input" name="BTYPE"/>                                                 [b]
 <Port kind="Input" name="INTRA_INFO"/>
 <Port kind="Output" name="IntraMode"/>
 <Instance id="1">                                                                 [c]
  <Class name="Algo_Address">
   <QID>
    <ID id="cal"/>
   </QID>
  </Class>
  <Parameter name="BTYPE">
   <Expr kind="Var" name="BTYPE_SZ"/>
  </Parameter>
  [...]
  <Note kind="label" value="Algo_Addr_ThreeLeftTop"/>
 </Instance>
                                                                                   [d]
 <Instance id="2">
  <Class name="Algo_IntraMode">
  [...]
  <Note kind="label" value="Algo_IntraModePrediction"/>
 </Instance>
 <Connection src="" src-port="BTYPE" dst="1" dst-port="BTYPE"/>                     [e]
 <Connection src="" src-port="BTYPE" dst="2" dst-port="BTYPE"/>
 <Connection src="" src-port="INTRA_INFO" dst="2" dst-port="INTRA_INFO"/>
 <Connection src="1" src-port="addr_top" dst="2" dst-port="addr_top"/>
 <Connection src="1" src-port="addr_left" dst="2" dst-port="addr_left"/>
 <Connection src="2" src-port="IntraMode" dst="" dst-port="IntraMode "/>
</XDF>
```

Fig. 11. A network description for intra prediction to show how the network is described in terms of FUs in decoder configuration procedure.

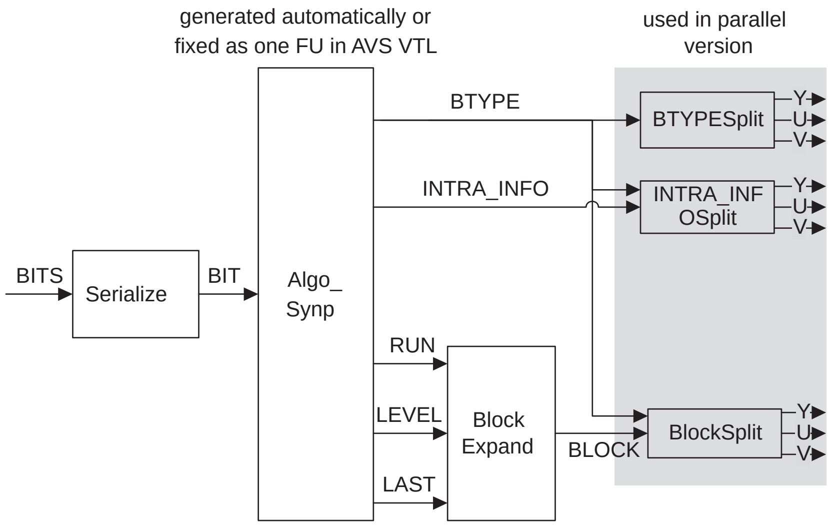


Fig. 12. Architecture of "Parser" network in the proposed AVS intra decoder configuration.

automatically or fixed as an FU in AVS VTL consumes the binary bitstream and sends out five tokens: "BTYPE", "INTRA_INFO" and "RUN, VALUE, LAST" for each 8×8 block. Token "BTYPE" combines various control signals in a fixed order and controls the decoding process. Token "INTRA_INFO" collects all necessary information for intra prediction of current block and allocates a fixed bitwidth for each kind of information. Tokens "RUN, LEVEL, LAST" are directly decoded from bitstream. FU "BlockExpand" stuffs zeroes according "RUN, LEVEL, LAST" and offers a sequence of 64 coefficients for each one 8×8 bock. Three "split" FUs are employed to separate tokens into Y, U and V channel in parallel version whereas in serial case, they are not invoked.

The "Decoding" network is to decode residuals and implement prediction. Sixty-four residuals of one 8×8

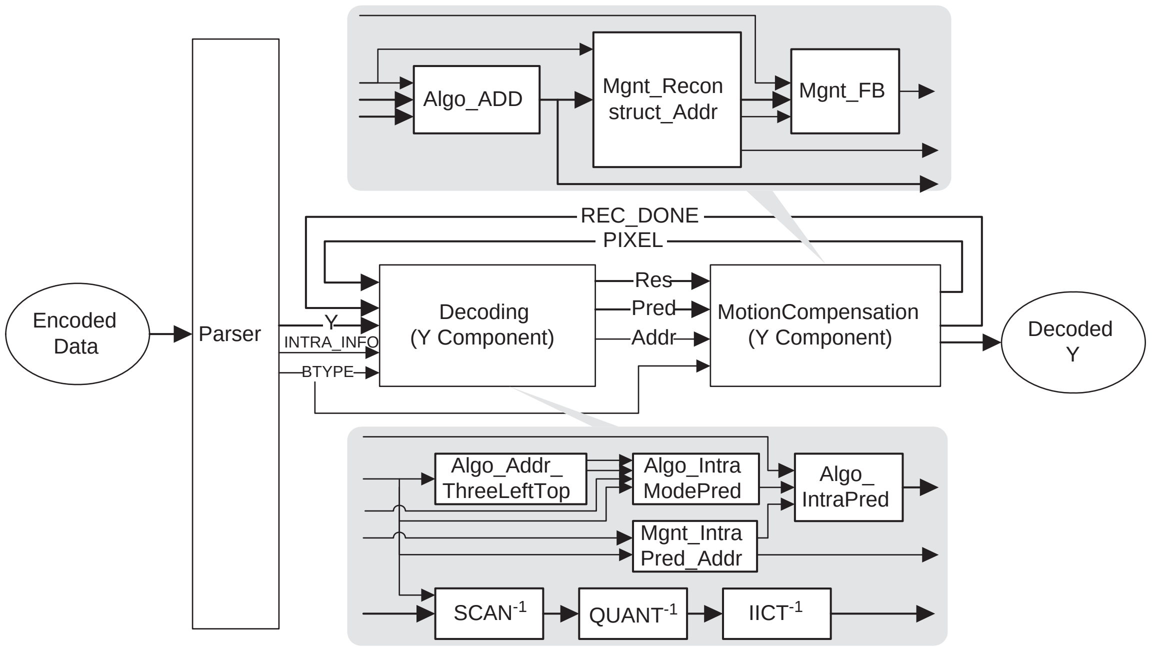
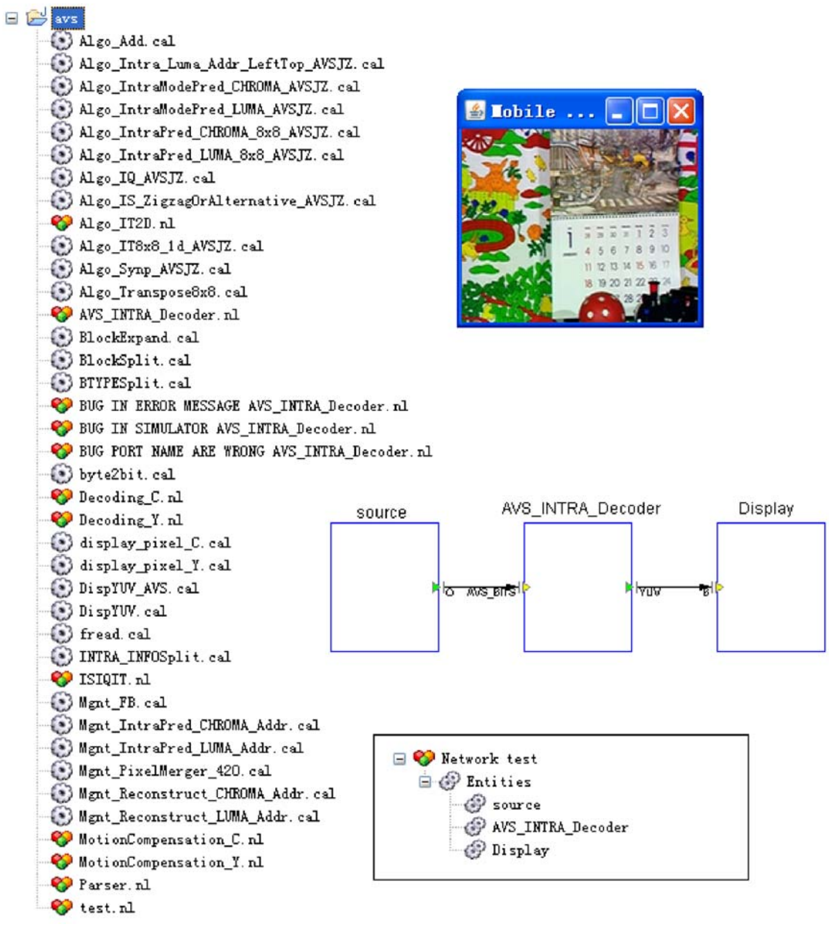


Fig. 13. Implementation architecture of the proposed decoder configuration example which can be instantiated as different decoder configurations (for instance of Y component).


```
avs
  Algo_Add.cal
  Algo_Intra_Luma_Addr_LeftTop_AVSJZ.cal
  Algo_IntraModePred_CHROMA_AVSJZ.cal
  Algo_IntraModePred_LUMA_AVSJZ.cal
  Algo_IntraPred_CHROMA_8x8_AVSJZ.cal
  Algo_IntraPred_LUMA_8x8_AVSJZ.cal
  Algo_IQ_AVSJZ.cal
  Algo_IS_ZigzagOrAlternative_AVSJZ.cal
  Algo_IT2D.nl
  Algo_IT8x8_1d_AVSJZ.cal
  Algo_Synp_AVSJZ.cal
  Algo_Transpose8x8.cal
  AVS_INTRA_Decoder.nl
  BlockExpand.cal
  BlockSplit.cal
  BTYPESplit.cal
  BUG IN ERROR MESSAGE AVS_INTRA_Decoder.nl
  BUG IN SIMULATOR AVS_INTRA_Decoder.nl
  BUG PORT NAME ARE WRONG AVS_INTRA_Decoder.nl
  byte2bit.cal
  Decoding_C.nl
  Decoding_Y.nl
  display_pixel_C.cal
  display_pixel_Y.cal
  DispYUV_AVS.cal
  DispYUV.cal
  fread.cal
  INTRA_INFOSplit.cal
  ISIQIT.nl
  Mgnt_FB.cal
  Mgnt_IntraPred_CHROMA_Addr.cal
  Mgnt_IntraPred_LUMA_Addr.cal
  Mgnt_PixelMerger_420.cal
  Mgnt_Reconstruct_CHROMA_Addr.cal
  Mgnt_Reconstruct_LUMA_Addr.cal
  MotionCompensation_C.nl
  MotionCompensation_Y.nl
  Parser.nl
  test.nl
```

```
Network test
  Entities
    source
    AVS_INTRA_Decoder
    Display
```

Fig. 14. Complete implementation of the proposed decoder configuration example which is instantiated as an AVS intra decoder and its test model in RVC simulation environment OpenDF.

block are output as a sequence of tokens "Res" after the process of inverse scan ($SCAN^{-1}$), inverse quantization ($QUANT^{-1}$) and inverse transform ($IICT^{-1}$). The intra prediction which has been explained above predicts pixels for current 8×8 block and outputs a sequence of data in horizontal raster order. The "Motion compensation" network receives the residuals and predicted pixels from decoding network to reconstruct pictures and stores reference pixels used for next prediction. A feedback context signal "REC_DONE" is introduced to negotiate between the "Decoding" and "Motion compensation" network because intra prediction for current block cannot be started until the reconstruction of pixels required by next block intra prediction is finished. In the proposed implementation, context signal "REC_DONE" is output every time one block is reconstructed to make sure that the reference pixels are available. The other feedback token is a sequence of reference data used for intra prediction which flows in "PIXEL" channel.

The decoder configuration which acts as an AVS intra decoder is successfully implemented in RVC simulation environment OpenDF. A test model is set up to read in AVS bitstream, decode the bitstream and display the reconstructed pictures. Fig. 14 gives a snapshot of the test model network as well as a reconstructed picture from the reconfigured decoder in OpenDF. Result of the simulation shows that with the dataflow mechanism and feedback tokens for communication between networks, multiple video formats, such as CCIR, CIF and QCIF are supported in the architecture. The realized configuration architecture keeps the advantage of dataflow and modular formalism in RVC framework and allows straightforward exchange and extension with new FUs.

6. Conclusion

This paper describes the decoder reconfiguration procedure in RVC framework by showing a decoder configuration architecture which successfully combines coding tools from AVS and MPEG VTL to form different decoding solutions. BS schema of AVS intra decoder configuration is detailed to explain how to define the BS schema for a reconfigured bitstream. FUs partition granularity of AVS and MPEG VTL is analyzed to reach high reusability in the proposed decoder configuration architecture. Besides the reasonable granularity, input and output tokens of each FU are specified in content and order to guarantee seamless connections between FUs. Moreover, the FU granularity and token specifications bring convenient exchangeability and easy extension ability for reconfiguration in RVC framework. Based upon the facets mentioned above, the configuration example shown in this paper can be instantiated as different decoders through exchanging and reusing FUs from VTLs. The proposed decoder configuration architecture reuses FUs from MPEG and AVS VTL efficiently, which reduces redesigns of FUs. It is also capable of exchanging some FUs between AVS and H.264/AVC which have the same token requirements but different algorithms to configure new decoders. Based on the architecture, an intra configuration of AVS is implemented by connecting AVS syntax parser and corresponding tools from AVS and MPEG VTL. It is obvious that the decoder configuration architecture shown in this example keeps the dataflow and modular mechanism of RVC and is flexible enough to be changed and extended to support other new configurations.

7. The future work

At present, there is still something to be done in RVC proceeding. In the realization of RVC framework, for the automatic generation of syntax parser, an optimized version is in development to deal with more complex cases in H.264/AVC which contains far more new technologies to improve performance. How to implement CABAC is also an emergent issue in RVC to work more efficiently. With the work of RVC proceeds, more complicated implementations of decoder configuration will be synchronically progressed. Meanwhile, more reusability and exchangeability will be explored between different VTLs to promote the development of RVC.

Acknowledgement

This work was supported in part by the National Science Foundation of China under Grants 60833013 and 60833006.

References

[1] Euee S. Jang, Chungku Lee, Study of Application Requirements Related to RVC, ISO/IEC JTC1/WG11 Document M15117, Antalya, January 2008.
[2] Euee S. Jang, Jens Ohm, Marco Mattavelli, Whitepaper on Reconfigurable Video Coding (RVC), ISO/IEC JTC1/WG11 Document W8485, Hangzhou, October 2006.
[3] Reconfigurable Video Coding Requirements, ISO/IEC JTC1/WG11 Document W8943, SanJose, April 2007.
[4] Dandan Ding, Lu Yu, Christophe Lucarz, Marco Mattavelli, Video Decoder Reconfigurations and AVS Extensions in the New MPEG Reconfigurable Video Coding Framework, 2008 IEEE Workshop on Signal Processing Systems (SiPS 2008), Washington, October 2008.
[5] Dandan Ding, Honggang Qi, Lu Yu, RVC Exploration Experiments Report on New 8×8 Inverse Transform and Quantization Tools, ISO/IEC JTC1/WG11 Document M14756, Lausanne, July 2007.
[6] Johan Eker, Jörn Janneck, CAL Language Report, ERL Technical Memo UCB/ERL M03/48, 2003.
[7] E.S. Jang, K. Asai, C.-J. Tsai, Study of Video Coding Tool Repository, ISO/IEC JTC1/WG11 Document W7329, Poznan, July 2005.
[8] Gwo Giun Lee, Euee S. Jang, Marco Mattavelli, Chun-Jen Tsai, Christophe Lucarz, Mickaël Raulet, Dandan Ding, Study Text of ISO/IEC FCD 23001-4: Codec Configuration Representation, ISO/IEC JTC1/WG11 Document N10165, Busan, October 2008.
[9] Information Technology—MPEG systems technologies—Part 5: Bitstream Syntax Description Language (BSDL), ISO/IEC FDIS 23001-5: 2007.
[10] C.-J. Tsai, Suggestions on the Direction of VCTR, ISO/IEC JTC1/WG11 Document M12074, Busan, April 2005.
[11] Dandan Ding, Honggang Qi, Lu Yu, Tiejun Huang, Wen Gao, Results of RVC Explore Experiment: Analysis of Function Units Reusability between MPEG-4/AVC and AVS for RVC Toolbox, ISO/IEC JTC1/WG11 Document M15415, Archamps, April 2008.
[12] Dandan Ding, Honggang Qi, Lu Yu, Tiejun Huang, Wen Gao, Results of RVC CE1.3: AVS Intra Configuration of RVC Framework, ISO/IEC JTC1/WG11 Document M15414, Archamps, April 2008.
[13] China Audio and Video Standard (AVS), Information technology advanced coding of audio and video part2: Video, GB/T 20090. 2/-2006.

[14] L. Yu, F. Yi, J. Dong, C. Zhang, Overview of AVS-video: tools, performance and complexity, Society of Photo-Optical Instrumentation Engineers Conference (SPIE 2005) 5960 (2005) 679–690.
[15] Ci-Xun Zhang, Jian Lou, Lu Yu, Jie Dong, Wai-Kuen Cham, The technique of pre-scaled integer transform, in: Proceedings of 2005 IEEE International Symposium on Circuits and Systems (ISCAS 2005), Vol. 1, pp. 316–319.
[16] Shuo Yao, Hai-Jun Guo, Lu Yu, Ke Zhang, A hardware implementation for full-search motion estimation of AVS with search center prediction, IEEE Transactions on Consumer Electronics 52 (4) (2006) 1356–1361.
[17] Christophe Lucarz, Marco Mattavelli, Joseph Thomas-Kerr, Jörn Janneck, Reconfigurable media coding: a new specification model for multimedia coders, IEEE Workshop on Signal Processing Systems (SiPS 2007), 2007, pp. 481–486.
[18] David Li, Dandan Ding, Christophe Lucarz, Marco Mattavelli, Efficient data flow variable length decoding implementation for the MPEG RVC framework, IEEE Workshop on Signal Processing Systems (SiPS 2008), Washington, October 2008.
[19] Christophe Lucarz, Dandan Ding, Marco Mattavelli, Samuel Keller, Automatic generation of parsers using XSLT and updated versions of the VLD FUs, ISO/IEC JTC1/WG11 Document M15384, Archamps, April 2008.
[20] Sunyoung Lee, Euee S. Jang, Yi-Shin Tung, Kohtaro Asai, Yoshihisa Yamada, Marco Mattavelli, Gwo-Giun Lee, Study Text of ISO/IEC 23002-4 FCD: Video Tool Library, ISO/IEC JTC1/WG11 Document N10169, Busan, October 2008.
[21] Christophe Lucarz, Marco Mattavelli, Dave Parlour, Serialized versions of the MPEG-4 SP FUs, ISO/IEC JTC1/WG11 Document M14873, Shenzhen, November 2007.
[22] Matthieu Wipliez, Ghislain Roquier, Mickaël Raulet, Jean-Francois Nezan, Olivier Dèforges, Code generation for the MPEG reconfigurable video coding framework: from CAL actions to C functions, 2008 IEEE International Conference on Multimedia and Expo (ICME), Hannover, June 2008.
[23] Jörn W. Janneck, Ian D. Miller, Dave B. Parlour, Marco Mattavelli, Christophe Lucarz, Matthieu Wipliez, Mickaël Raulet, Ghislain Roquier, Translating Dataflow Programs to Efficient Hardware: an MPEG-4 Simple Profile Decoder Case Study, Design, Automation and Test in Europe (DATE), Munich, 2008.