Building Multimedia Security Applications in the MPEG Reconfigurable Video Coding (RVC) Framework

Junaid Jameel Ahmad
Shujun Li
University of Konstanz
Germany

Ihab Amer
German University in Cairo
(GUC), Egypt

Marco Mattavelli
École Polytechnique Fédérale
de Lausanne (EPFL)
Switzerland

ABSTRACT

Although used by most of system developers, imperative languages are known for not being able to provide easily reconfigurable, platform independent and strictly modular applications. ISO/IEC has recently developed a new video coding standard called Reconfigurable Video Coding (RVC), with the objective of providing modular and concurrent specifications of complex video codecs that constitute a better starting point for implementation of applications using video compression. Multimedia security applications are traditionally developed in imperative languages mainly because the required multimedia codecs were only available in specification and implementations based on imperative languages. Therefore, aside from the technical challenges inherited from multimedia codecs, multimedia security applications also face a number of other challenges which are only specific to them. Since a number of multimedia codecs are already available in the RVC framework, multimedia security applications can now also be developed using this new development framework. This paper explains why the RVC framework approach can be used to efficiently overcome those technical challenges better than existing imperative languages. In addition, the paper demonstrates how the RVC framework can be used to quickly develop multimedia security applications by presenting some examples including a joint H.264/AVC video encryption-encoding system, a joint JPEG image encryption-encoding system and a image watermarking system in JPEG compressed-domain.

Categories and Subject Descriptors

I.4.2 [**Image Processing and Computer Vision**]: Compression (Coding); E.3 [**Data**]: Data Encryption

General Terms

Design, Languages, Performance, Security

MM&Sec'11, September 29–30, 2011, Buffalo, New York, USA.

Keywords

Reconfigurable Video Coding (RVC), video tool library (VTL), Crypto Tools Library (CTL), joint multimedia encryption-encoding (JMEE), digital watermarking, MPEG, H.264/AVC, JPEG

1. INTRODUCTION

In recent years, the security of multimedia applications has become an important requirement that creators, producers, distributors and consumers of multimedia products cannot ignore anymore. This is because nowadays it is much easier for attackers to make pirate copies, to crack commercial multimedia systems, to attack online multimedia services, and so forth. So as to ensure the protection of multimedia content, a number of multimedia security schemes (multimedia encryption, watermarking and information hiding etc.) have been developed and are being used in different forms.

In order to devise any multimedia security scheme (e.g., joint multimedia encryption-encoding, watermarking and information hiding in compressed-domain, etc.), traditionally researchers have been working directly on the codec implementations mostly available in the form written using imperative languages such as C/C++, Java, etc. However, most imperative languages are not strictly modular and often have dependencies on a specific platform[1], which results in a number of challenges the developers have to overcome in building reconfigurable and platform-independent multimedia security systems. Furthermore, many multimedia security techniques may be applied to multiple multimedia codecs, so benchmarking the security properties of different multimedia security techniques with multiple multimedia codecs can be a very time and resource consuming task because those multimedia codecs are often implemented in completely different software (SW) architectures and written using different imperative languages. This fact also makes a judicial comparison of those multimedia security techniques rather difficult since the underlying multimedia codecs are not working on the same base.

Using imperative programming languages as the main development tool has introduced some technical challenges in the design and implementation of more and more compli-

[1]In the context of MPEG RVC framework, the word "platform" has a more broader meaning than usual. Basically, it covers the whole environment converting source code to executables and running the executables, which include the compilers, the operating system, the virtual machine (if any), and the underlying hardware.

cated multimedia codecs [24]. The main difficulties and impediments include the difficulty of reusing modules, the reconfiguration of existing codecs, and the impossibility of porting an existing implementation to a completely different platform without major redesign tasks. So as to reduce the problem related to those technical challenges, the ISO/IEC SC29/WG11 committee, better known as MPEG, has recently standardized a framework called RVC (Reconfigurable Video Coding) [32, 33]. The salient features of the RVC framework include *modularity*, *reusability*, *reconfiguration*, *platform independence* and *code analyzability*. While the RVC framework has been standardized in the context of video coding, it is actually a general framework for all data-driven applications. As a result, it has been successfully applied to development of different kinds of multimedia (video, audio, image and graphics) codecs [11–13, 20, 21, 26, 33, 39]. In addition, it has also been used to develop a Crypto Tools Library (CTL) [19], which provides a set of both multimedia codecs and cryptosystems developed using languages and tools standardized by the RVC framework, making it possible to efficiently build reconfigurable multimedia encryption applications for different platforms.

In this paper, we discuss some technical challenges in building and benchmarking multimedia security applications and show how these challenges can be addressed by using the RVC framework. We showcase a number of multimedia security applications to highlight the ease and benefits of building multimedia security applications in the RVC framework.

The remainder of this paper is organized as follows. After giving an overview of the RVC framework in Sec. 2, Section 3 discusses the challenges for building and benchmarking multimedia security applications and shows how they can be better handled using the RVC framework. Section 4 showcases several examples of multimedia security systems developed based on the RVC framework. Conclusions and future work directions are summarized in Sec. 5.

2. RECONFIGURABLE VIDEO CODING

The RVC framework was standardized by the ISO/IEC to better respond to the technical challenges of developing specifications of complex video coding algorithms structured into multiple video coding standards [32, 33]. One main concern of MPEG is how to make the codec specification modular, so that common building blocks of different standards are identified and video codecs can be specified as different configurations (e.g., different video coding standards, different profiles and/or levels, different system requirements) of a standard library of components. So as to achieve this goal, the RVC standard defines a framework that covers different normative and non-normative steps of the whole video codec development process. The community interested in using the RVC standard has also developed non-normative supporting tools [3,8,9] that support the process of editing/simulating a standard RVC codec specification, and transforming it into proprietary implementations towards various platforms.

In essence, the RVC framework provides a specification that is a *good* starting point for SW and HW implementations by enabling developers to work on a single platform-independent design at a higher level of abstraction than traditional sequential specifications. The portability of the specification and the backends of the supporting tools enable to generate, from the same high level design, implementations that target different platforms such as general-purpose PCs, heterogeneous embedded systems, mobile phones, and FPGAs [22, 35]. In principle, the RVC framework also supports hardware/software co-design by converting part of a design into software and the other part into hardware [46]. The RVC framework is based on dataflow programming [27] that allows automatic code analysis to facilitate large-scale design-space exploitation stages such as identifying components for increasing the explicit parallelism of implementations running on multi-core and many-core systems, or other transformations to optimize efficient partitioning and scheduling of implementations [22, 25, 40].

The RVC standard is composed of two parts: MPEG-B Part 4 [32] and MPEG-C Part 4 [33]. MPEG-B Part 4 defines the languages and the formalism for specifying configurations of video codecs, and MPEG-C Part 4 defines a standard library of algorithms in the form of dataflow components (VTL[2]) that are composed of a number of functional units (FUs) as platform-independent building blocks of MPEG standard compliant video decoders [33]. To support the RVC dataflow framework, MPEG-B Part 4 standardizes three languages: a dataflow programming language called RVC-CAL (a subset of the original CAL Language specification [27]) for describing platform-independent FUs, an XML dialect called FNL (FU Network Language) for describing connections between FUs, and another XML dialect called RVC-BSDL for describing the syntax format of video bitstreams.

Figure 1 illustrates how a video decoder is represented and how proprietary or platform-specific implementations are generated from an RVC codec specification. The first step of the process is to describe the video decoder in the form of an FU network description (FND) by using FNL, where the FUs are components of the VTL. Given the FND, an instantiation process is invoked to select the required FUs from the MPEG VTL to produce an abstract decoder model (ADM). The ADM can then be automatically translated into decoder implementations that can execute on the target platform to decode video data. The transformation process from the ADM to platform dependent implementations is not specified by the standard and is performed by mapping the RVC-CAL and FNL code into compilable source code written in a target programming language such as C/C++, Java, Verilog/VHDL by an appropriate code synthesis tool. Note that the whole process is fully automated if both the VTL and the FND are available and any of the non-normative SW or HW synthesis tools are used [35, 46].

Although the RVC framework is developed in the context of video coding, it is actually a general-purpose framework that is particularly well adapted to specify and implement any data/streaming-driven application such as cryptosystems. By using the RVC framework, we have already built a library of cryptographic FUs that we call Crypto Tools Library (CTL) [19]. We have used the standard MPEG RVC VTL, the non-standard CTL and some other non-standard FUs to develop some multimedia security applications including joint multimedia encryption-encoding applications and digital watermarking systems in compressed domain.

[2]Currently, VTL covers decoding algorithms only, however encoding algorithms may be part of future editions of the RVC standard. In order to facilitate our discussion, in the rest of this section, we will focus more on decoders as what has been standardized, but all the concepts can be generalized to encoders.

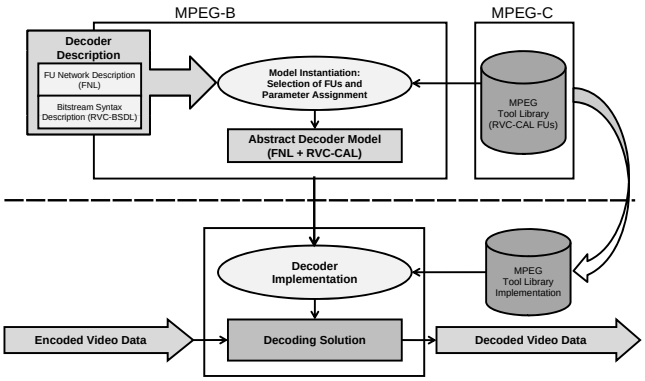

Figure 1: The process of decoder representation and implementation generation in the RVC framework.

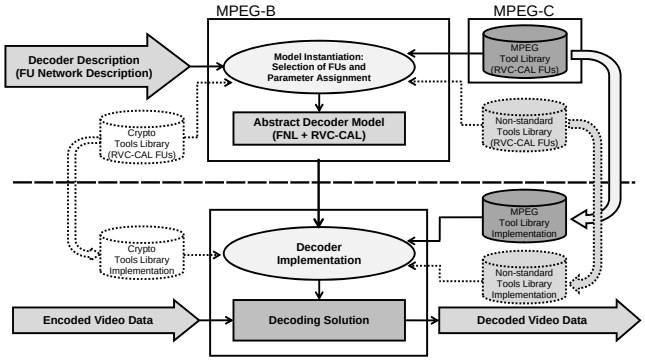

Figure 2: Graphical representation of jointly using the VTL, the CTL and other non-standard FUs in the RVC framework.

Figure 2 shows a graphical representation of using VTL with CTL and other non-standard FUs in the RVC framework.

The real conceptual innovation of the RVC framework is the usage of RVC-CAL as abstraction for the specification, the analysis and the generation of implementations. RVC-CAL as mentioned above is a subset of the CAL [27] dataflow programming language that was created as part of the Ptolemy project at the University of California, Berkeley in 2003 [10]. In CAL, FUs are implemented as actors containing a number of fireable actions and internal states. In the CAL's term, the data units that are exchanged among actors are called tokens. Each actor can contain both input and output ports that receive input tokens and produce output tokens, respectively. Each action of an actor may fire depending on four different conditions: 1) input token availability; 2) guard conditions; 3) finite-state machine based action scheduling; 4) action priorities. In CAL, actors are the basic functional entities that can run in parallel, but actions in an actor are atomic, meaning that only one action can fire at one time. This structure gives a balance between modularity and parallelism and makes automatic analysis on actor merging and splitting possible. Figure 3 shows the internal structure of a CAL actor in an FU network.

FNL is used to specify a network of actors (or FUs) by providing information about connections among FUs and FU parameters. The third language defined in the RVC framework, RVC-BSDL, is used to specify a bitstream syntax and

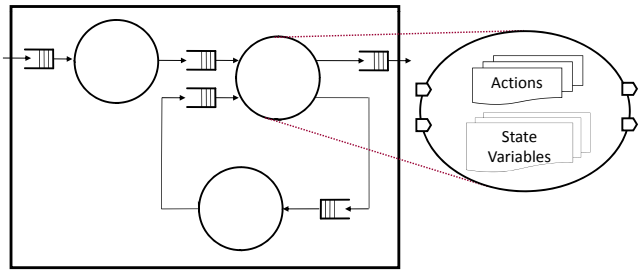

Figure 3: The internal structure of an RVC-CAL actor in an FU network.

supports the automatic synthesis of the corresponding bitstream parser.

Currently, there are three main RVC tools available: OpenDF [8], ORCC (Open RVC-CAL Compiler) [9] and Graphiti [3]. Both OpenDF and ORCC provide an RVC-CAL editor and an RVC simulator. OpenDF includes a Verilog HDL code generation backend, and ORCC includes C/C++, Java, LLVM and VHDL code generation backends. Graphiti is a graphical tool for editing FU networks.

The concept of data-flow programming is actually very general and the basic idea can be traced back to the 1960s [47]. The past half century has witnessed many data-flow programming languages and development tools that follow the basic concept and provide some features similar to what the RVC framework can offer. Many programming languages and development tools have been widely used in industry. However, the RVC framework was designed to offer a richer set of features that are needed for multimedia applications than other existing solutions as shown in Table 1. It should be noted that here we consider only features that are relevant for the goals of this paper. Table 1 is in no sense an exhaustive overview of the pros and cons of all solutions.

3. CHALLENGES FOR MULTIMEDIA SECURITY APPLICATIONS

In this section, we present the challenges of building and benchmarking multimedia security applications using imperative languages (like C/C++, Java etc.) and how these challenges can be more easily handled by using the RVC framework. To make a concrete sense of the challenges, we use multimedia encryption/decryption as an example of multimedia security applications, however in general these challenges do indeed apply to other multimedia security applications as well.

3.1 Challenges

Implementing a multimedia encryption/decryption algorithm in imperative languages poses the following challenges.

- **Locating encryption/decryption points:** Because most imperative languages do not have a very strict requirement on modularity, it is often very time consuming to locate the proper points to insert the encipher/decipher in the source code of an existing multimedia codec. Note that for some codec implementations and multimedia encryption algorithms, it is necessary to insert the encipher/decipher code at multiple places, which further complicates the problem.

Table 1: Comparison of the RVC framework and some existing solutions. The ten columns in the table represent the following features: A) high-level (abstract) modeling and simulation; B) platform independence; C) code analyzability (i.e., semi-automated design-space exploitation); D) hardware code generation; E) software code generation; F) hardware-software co-design; G) the number of supported target languages; H) open-source or free implementations; I) international standard.

	A	B	C	D	E	F	G	H	I
RVC	Yes	Yes	Yes	Yes	Yes	Yes	6	Yes	Yes
ImpulseC [23]	No	No	No	Yes	No	Yes	1	No	No
Handel-C [37]	No	No	No	Yes	No	No	1	No	No
Spark [28]	No	No	No	Yes	No	Yes	1	No	No
BlueSpec [43]	Yes	No	Yes	Yes	Yes	No	2	No	No
Koski [36]	Yes	Yes	Yes	Yes	Yes	Yes	3	No	No
Daedalus [15, 48]	Yes	Yes	Yes	Yes	Yes	Yes	3	Yes	No
PeaCE [29]	Yes	Yes	Yes	Yes	Yes	Yes	3	Yes	No
Simulink [5–7]	Yes	Yes	Yes	Yes	Yes	No	4	No	No
LabVIEW [4]	Yes	Yes	Yes	No	No	No	0	No	No
Esterel [1, 2]	No	Yes	No	Yes	Yes	No	2	Yes	Not yet
CoWare [45]	Yes	Yes	No	Yes	Yes	Yes	2	No	No
Synopsys System Studio [14]	Yes	Yes	Yes	Yes	Yes	Yes	3	No	No
Cryptol [16, 38]	Yes	Yes	Yes	Yes	Yes	No	5	No	No
CAO [17, 42]	Yes	Yes	No	No	Yes	No	3	No	No

- **Distortion of codec implementation:** Mixing the implementations of encipher/decipher with encoder/decoder introduces distortions in the core implementation of encoder/decoder and affects its reusability. Furthermore, this mixing of code may also introduce some side effects, e.g. the synchronization of different modules may be disturbed. Hence, the developers also have to study those potential side effects and find a proper solution.
- **Platform dependence:** In order to implement any multimedia encryption algorithm for different hardware/software platforms, developers have to implement them in different target implementation languages supported by those platforms. Since the implementation of same codec in different programming languages generally follow different implementation styles (depending on the features provided by programming languages and expertise of the codec developer), implementing the same multimedia encryption algorithm for different platforms is a very laborious task.
- **Non-judicial performance benchmarking:** In the light of above points, while it may not be very difficult to manually implement a specific multimedia encryption algorithm for a specific multimedia codec, we will have much more trouble to conduct a judicial performance benchmarking of different multimedia encryption algorithms for multiple multimedia codecs. In cases when some codecs are developed in different programming languages, it will become extremely difficult to incorporate them into a single benchmarking system for the purpose of a fair performance comparison. There will be even more troubles, if we have to extend the benchmark system to cover different hardware/software platforms.

3.2 Solutions

In essence, building multimedia encryption/decryption applications in the RVC framework provides natural solutions to the above-described challenges. In the following, we briefly explain how these challenges can be more easily met by using the RVC framework.

- **Locating encryption/decryption points:** FUs in the RVC framework are strictly modular. They have constant and well-defined I/O interfaces (FUs encapsulate their own states and communicate with the outside world only via tokens), so the insertion of encipher/decipher FUs does not require developers to look into the inside code of codec FUs. Rather, it is only a matter of visually reviewing the graphical network of the codec, locating the candidate FUs (e.g., zig-zag ordering and entropy coding in Fig. 5) that produce tokens of our interests and tunneling the data channels to pass through the added encipher/decipher FUs.
- **Preservation of codec implementation:** Since addition of encipher/decipher FUs does not require to change any encoding/decoding FUs, building multimedia encryption/decryption algorithm in the RVC framework preserves the generic encoder/decoder FUs and does not compromise their reusability.
- **Platform independence:** As highlighted in Sec. 2, RVC is a platform-independent framework. Building multimedia encryption algorithms in the RVC framework allows developers to code only once and generate the hardware/software implementations for different platforms via an automated code generation process. The liberty of having platform-independent representations not only saves enormous time of application

developers, but also allows easier maintenance of the source code.

- **Judicial performance benchmarking:** While working in the RVC framework, different multimedia encryption algorithms can be built by reusing exactly the same video codec. This allows the performance benchmarking to be more judicial and to be conducted for many platforms without re-programming. As highlighted in Sec. 2, video codecs built in the RVC framework can reuse many FUs from the VTL. For scenarios where we need to benchmark the same multimedia encryption algorithms over different multimedia codecs, the reusability of the same VTL FUs makes the overall performance benchmarking to remain judicial.

4. SYSTEM DESIGN EXAMPLES

In this section, we showcase three multimedia security applications developed using the RVC framework. The first one is a joint encryption-encoding H.264/AVC videos and the second is a joint decryption-decoding systems for JPEG images. The third example is an image watermarking system working in JPEG compressed-domain. We present the three examples in the following three subsections, respectively.

4.1 Joint H.264/AVC Encryption-Encoding

In this subsection, we present sign bit encryption and decryption of H.264/AVC videos [34]. A stream cipher called ARC4 (Alleged RC4) [41] is used as the underlying cryptosystem for encrypting sign bits. The sign bit encryption system is designed to maintain format compliance of the encrypted video bitstreams. We used the RVC-based baseline profile implementations of encoder [21, 22] and decoder [33]. In order to cross verify the functional correctness of the H.264/AVC sign bit encryption/decryption system, we used both the reference implementation of H.264/AVC [18] and Vega H.264 Analyzer [30] to validate the conformance of the generated encrypted bitstreams.

4.1.1 ARC4 based Sign Bits Flipper

As part of the joint multimedia encryption-encoding system, we built a sign bit flipper based on the ARC4 stream cipher and used it to flip the sign bits of all DCT coefficients. Figure 4 shows the FU network of our sign bits flipper module. The `Extract_Sign_Bits` FU extracts the sign bits of all DCT coefficients, combines each eight consecutive bits to form one byte and then sends each byte to the `ARC4` stream cipher FU for further processing. The `ARC4` FU encrypts each input byte and sends the output byte to the `Change_Sign_Bits` FU, which breaks each byte into eight encrypted sign bits and then assigns them back to the corresponding DCT coefficients. It should be noted that we used the same sign bit flipper FU as the encipher and the decipher in encoder and decoder sides since both the ARC4 stream cipher and the sign bit encryption are totally symmetric.

4.1.2 Joint Video Encryption-Encoding

In order to encrypt the sign bits of all DCT coefficients, we decided to perform the encryption just before the entropy coding stage (the Context Adaptive Variable Length Coding (CAVLC) in our case). Therefore, at the encoder side, the sign bits flipper FU network is inserted after the zig-zag

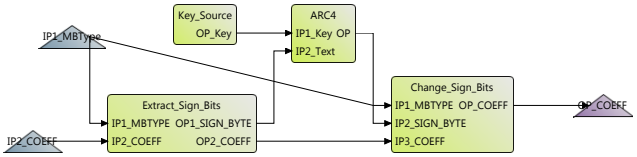

Figure 4: The ARC4 based sign bits flipper FU network.

ordering of DCT coefficients, but before the start of the core functionality of the CAVLC. Figure 5 shows a partial view of the XDF network of the sign bit encryption system. All DCT coefficients pass through the `RC4_Sign_Bits_Flipper` FU of Fig. 4 (highlighted by a red frame box; the same hereinafter) to get encrypted before they travel to other FUs of `CAVLC`. This clearly emphasizes how easy it is for the developers to insert the encipher into the encoder as a reconfigurable add-on. It should be emphasized that no other changes were done to the original encoder except inserting the encipher. This point will be repeatedly shown in other examples presented in this section.

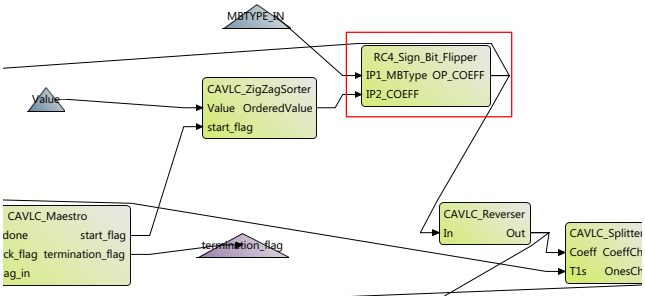

Figure 5: The ARC4 based sign bits encryption-encoding for H.264/AVC videos.

4.1.3 Joint Video Decryption-Decoding

Since the sign bits were encrypted just before the CAVLC encoder, to correctly decrypt the sign bits of all DCT coefficients at the decoder side, the sign bits flipper FU network is inserted just after entropy decoding. Figure 6 shows a partial view of the XDF network of the sign bit decryption system. The `RC4_Sign_Bits_Flipper` FU is inserted between the `Algo_BlockExpand_AVC` and `Algo_BlockSplit_AVC` FUs (which are standardized FUs in VTL [33]). All DCT coefficients pass through the `RC4_Sign_Bits_Flipper` FU to get decrypted before they travel to the `Algo_BlockSplit_AVC` FU and go through the remainder of the decoder. Once again, we show it is relatively-easy to locate the decipher's insertion point and to insert the decipher into the decoder.

4.2 Joint JPEG Encryption-Encoding

In this subsection, we present DC encryption and decryption of JPEG images [31]. We again used the ARC4 stream cipher as the underlying cryptosystem for encrypting DC coefficients. This DC encryption system is designed to maintain format compliance of the encrypted image bitstreams. We used the RVC-based implementations of JPEG codec available in the Open RVC-CAL Applications Project [12].

4.2.1 ARC4 based DC Encryption

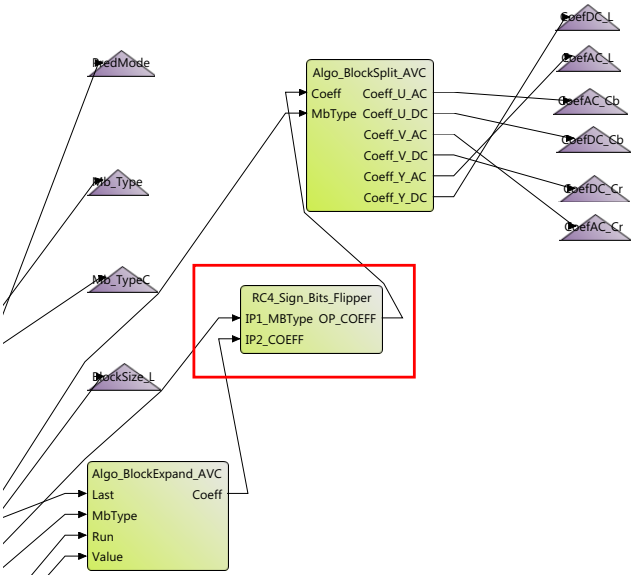

Figure 6: The ARC4 based sign bits decryption-decoding for H.264/AVC videos.

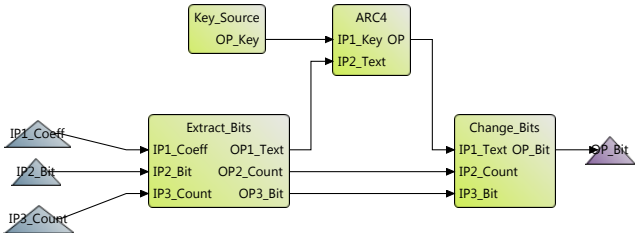

Figure 7: The ARC4 based DC encipher FU network.

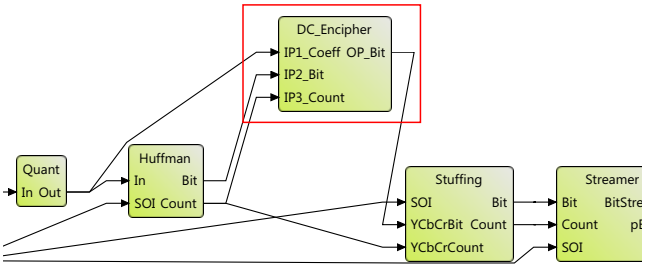

Figure 8: The ARC4 based DC encryption-encoding for JPEG images.

As part of the joint JPEG image encryption-encoding system, we built a DC encipher application based on the ARC4 stream cipher and used it to encrypt the DC coefficients. This application scans the bitstream generated by the JPEG encoder's Huffman encoding FU to locate the codeword and fixed-length additional bits corresponding to each DC coefficient.

Figure 7 shows the DC encipher FU network. In order to retain the compression efficiency, we chose to only encrypt the fixed-length coded additional bits after the Huffman category codeword of each DC coefficient. In addition, in order to keep the encipher simple, we used the quantized DC coefficients to skip the category code without decoding it. Alternatively, this can also be done by building the Huffman tables of the category coding inside the encipher. But this will make the encipher complex. Another input of the `DC_Encipher` FU, `IP3_Count`, informs the `DC_Encipher` FU about the number of bits in each block and is used to skip AC coefficients in each block without decoding them. The `Extract_Bits` FU extracts the fixed-length additional bits from the encoded bitstream, and sends them to the `ARC4` stream cipher FU for further processing. The `ARC4` FU encrypts those additional bits and sends the output bits to the `Change_Bits` FU, which replaces the original additional bits by the encrypted ones.

4.2.2 *Joint Image Encryption-Encoding*

In order to perform the DC encryption using the DC encipher module presented above, the encipher should be placed just after the Huffman coding stage, but before the start of bit stuffing. Figure 8 shows a partial view of the FU network of the DC encryption system. The bitstream generated by the `Huffman` FU passes through the `DC_Encipher` FU network of Fig. 7 to gets encrypted before it travels to the `Stuffing` FU and the rest of the encoder.

4.2.3 *ARC4 based DC Decryption*

For the joint JPEG image decryption-decoding system, we built a DC decipher application based on the ARC4 stream cipher. This application scans the encrypted bitstream and decrypts the fixed-length coded additional bits of each DC coefficient. Unlike the DC encipher application where we successfully avoided building the Huffman tables, the DC decipher cannot be built without the decoding Huffman tables because the original DC coefficients are not available before Huffman decoding happens (which differs from the encoder side, where the original DCT coefficients are all available before Huffman encoding). This makes the DC decipher application more complex than the encipher application.

Figure 9 shows the FU network of the core of the DC decipher. After building the decoding Huffman tables, the `Extract_Bits` FU scans the bitstream to extract the encrypted fixed-length coded additional bits of the DC coefficients from the input bitstream and sends them to the `ARC4` stream cipher for further processing. The `ARC4` FU decrypts those additional bits and sends the output bits to the `Change_Bits` FU, which replaces the encrypted fixed-length additional bits by the decrypted (original) ones.

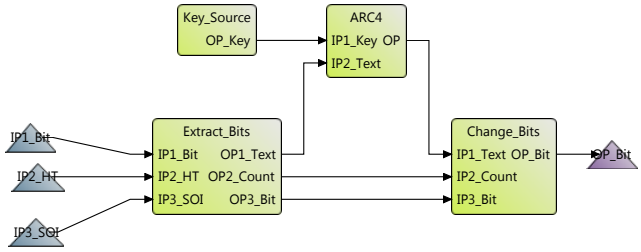

Figure 9: The ARC4 based DC decipher FU network.

4.2.4 *Joint Image Decryption-Decoding*

Since the encryption of DC coefficients at the encoder side is performed after the Huffman encoding stage, the decryption should be performed before the Huffman decoder. Figure 10 shows a partial view of the FU network of the DC decryption system. The bitstream and the Huffman table definitions generated by the `Parser` FU pass through the `DC_Decipher` FU network of Fig. 9 to get decrypted before they travel to the Huffman decoding and the rest of the decoder.

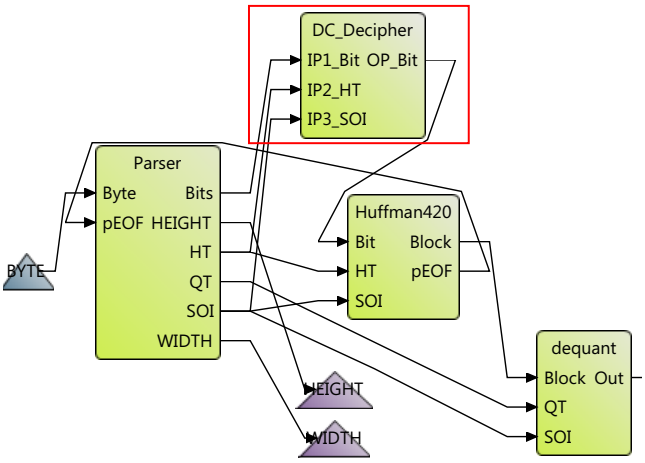

Figure 10: The ARC4 based DC decryption-decoding for JPEG images.

4.3 Compressed-Domain JPEG Image Watermarking

In this subsection, we present the third example, an image watermarking scheme working in JPEG compressed-domain, which is a ported edition of the H.264/AVC watermarking scheme proposed in [44] to JPEG compressed-domain. In this scheme, a number of macroblocks are randomly selected for watermark embedding and in each selected macroblock one watermark bit is embedded in exactly one quantized AC coefficient. The random paths at the macroblock level and the AC coefficient level are both driven by a stream cipher, so an attacker does not have knowledge about where the watermark bits are embedded. The random path at the AC coefficient level is derived from a content-dependent sequence encrypted by the stream cipher.

We implemented the watermark embedder and detector FU networks of this scheme, which work with the RVC JPEG codec available in the ORCC Applications project [12]. Figures 11 and 12 shows our implementations of the watermarking embedder and detector FU networks, where the `KPi_Extractor` FU generates a content-dependent sequence and sends it to the `ARC4` FU for encryption. The encrypted sequence is then used to select an AC coefficient from the current macroblock for watermark embedding/detection.

The random path generator at the macroblock level was implemented as an FU subnetwork as shown in Fig. 13. Based on the size of the image, the `Prepare_Indices` FU prepares the list of possible macroblock indices and sends all indices to the `ARC4` stream cipher, which then encrypts those indices to produce a number of random indices. The `Swap_Indices` FU keeps an internal list of indices for all macroblocks, and for the i-th random index $I(i)$ from the

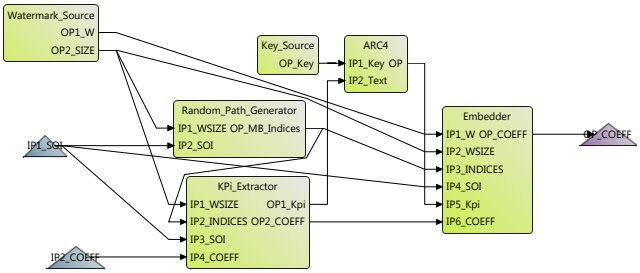

Figure 11: The watermarking embedder FU network.

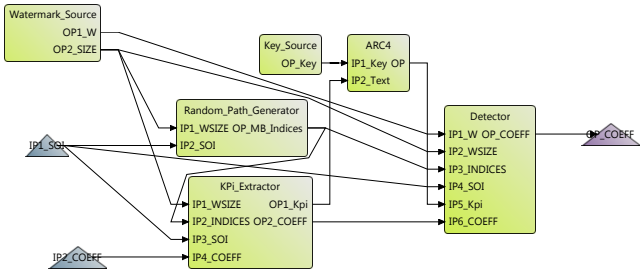

Figure 12: The watermarking detector FU network.

`ARC4` FU it swaps the i-th internal index with the $I(i)$-th one. The swapping operations can finally lead to a shuffled list of macroblock indices, and any W continuous indices can be used as the random path for watermarking purpose, where W is the number of watermark bits embedded/detected.

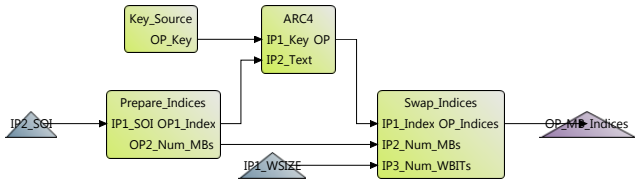

Figure 13: The random path generator FU subnetwork.

4.3.1 *Watermarking Embedder in JPEG Encoder*

Figure 14 shows the watermarking embedder FU network incorporated into the RVC JPEG encoder. Since this watermarking scheme is proposed to work on quantized AC coefficients, we inserted the watermark embedder FU network after DCT coefficients are quantized, but before being Huffman coded.

As described above, the watermarking embedder FU network selects a number of macroblocks and then embeds one bit into one AC coefficient in each selected macroblock. The embedding operation is done by replacing the least significant bit of the selected AC coefficient with the watermark bit. After the watermark bit is embedded, the whole macroblock is forwarded to the remainder of the encoder.

4.3.2 *Watermarking Detector in JPEG Decoder*

Figure 15 shows the watermarking detector FU network incorporated in the RVC JPEG decoder. Similar to the watermark embedder FU network, the detector FU network

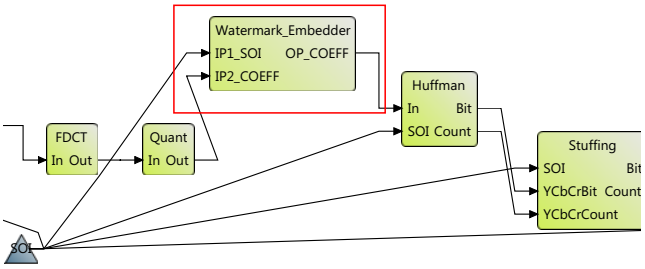

Figure 14: The watermarking embedder working with the RVC JPEG encoder.

has been placed so that it can scan the quantized DCT coefficients before they go through the de-quantization step. The detector FU network works very similarly to the embedder FU network: it reconstructs the same random path to select a number of macroblocks, then selects the same AC coefficient in each selected macroblock, and finally reads the least significant bit of the selected AC coefficient to extract one watermark bit. After extracting the watermark bit, the whole macroblock is forwarded to the rest of the decoder.

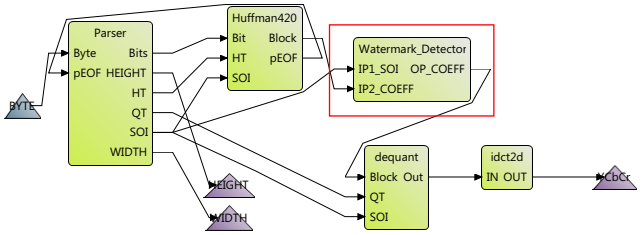

Figure 15: The watermarking detector working with the RVC JPEG decoder.

4.4 Run-Time Performance

In addition to all the benefits we can gain from using the RVC framework, the run-time performance of the automatically generated implementations from the RVC code is also of great concern since our ultimate goal is to build practical applications that can run efficiently on different target platforms. For all the three multimedia security applications, we generated C source code using ORCC [9]. Then, we used the C compiler in the Microsoft Visual Studio 2008 to generate executables that can run on Windows platforms. The performance results reported in this section are calculated by running these on a general-purpose desktop PC (HP Compaq 8000 Elite Convertible Minitower with an Intel Pentium Dual-Core E5400 2.70GHz CPU and 2.0 GB main memory) under safe-mode command prompt of Windows 7. To have a reference for comparison, we also report the results of the H.264/AVC and JPEG codecs without encryption nor watermarking.

For H.264/AVC codec, Table 2 reports the run-time performance for encoding and decoding the first 99 frames of three test videos with and without sign bits encryption. The results show that the time overheads caused by sign bits encryption and decryption are both below 11%.

For the JPEG codec and the two JPEG multimedia security applications, all the executables were run on three 512×512 test images to see the real run-time performance of the RVC applications. We used the default quantization tables listed in Sec. K.1 of [31], and the JPEG quality factor was set to 50. The run-time performance is shown in Table 3. One can see that all the executable run with a reasonably fast speed. The time overheads caused by JPEG DC encryption and decryption are less than 10% and 2%, respectively. Similarly, the time overheads for JPEG image watermark embedder and detector are less than 9% and 1%, respectively.

We also checked the influence of the additional encryption and watermarking operations on the compression efficiency of the H.264/AVC and JPEG codecs. As expected, all the three RVC multimedia security applications have no or only a negligible influence on the compression ratio: 1) the JPEG watermarking-encoding application preserves the compression ratio; 2) the H.264/AVC and JPEG encryption-encoding applications only slightly change the compression ratio.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed the challenges imposed by imperative languages in the development and judicial benchmarking of multimedia security applications. With the help of three multimedia security application case studies, we have shown how those challenges can be overcome by using the concepts, the languages and the tools provided by the MPEG RVC framework. In addition, we also reported some preliminary results for the run-time performance benchmarking of our three multimedia security applications.

In future, we plan to extend the run-time performance benchmarking of multimedia security applications developed in the RVC framework against the corresponding implementations in imperative languages on different platforms. Based on the benchmarking results, we will be able to figure out how to further improve our RVC implementations and RVC supporting tools. In addition, we will continue to develop more examples of multimedia security applications. For the long-run, we are also going to build benchmark systems for multimedia encryption and watermarking working in compressed-domain.

Acknowledgments

Junaid Jameel Ahmad and Shujun Li were supported by the Zukunftskolleg of the University of Konstanz, Germany, which is part of the "Excellence Initiative" Program of the DFG (German Research Foundation).

The authors would like to thank Matthieu Wipliez, Marwan Abd Ellah, Endri Bezati and Ghislain Roquier for their help in answering our questions on their RVC H.264/AVC and JPEG codecs. We also thank Lobna Genena for helping port the RVC H.264/AVC encoder from OpenDF to ORCC.

6. REFERENCES

[1] Esterel. `http://www.esterel-technologies.com/files/Esterel-Language-v7-Ref-Man.pdf`.

[2] Esterel Synchronous Language Web Main page. `http://www-sop.inria.fr/esterel.org/files/`.

[3] Graphiti. `http://graphiti-editor.sf.net`.

[4] LabVIEW. `http://www.ni.com/labview/whatis/`.

[5] Mathworks Simulink - Simulation and Model-Based Design. `http://www.mathworks.com/products/simulink/`.

Table 2: The run-time performance of H.264/AVC video joint encryption-encoding and decryption-decoding applications on the first 99 frames of three QCIF test videos (in milliseconds). The percentages in the brackets are the overheads of encryption/decryption added on the encoding/decoding processes.

Test Videos	Encoding	Encoding-Encryption	Decoding	Decoding-Decryption
foreman	90129.10	99601.40 (10.5%)	2424.81	2586.93 (6.69%)
highway	93300.00	94403.80 (1.18%)	2337.42	2450.15 (4.82%)
suzie	89149.90	94599.20 (6.11%)	2259.57	2473.08 (9.45%)

Table 3: The run-time performance of JPEG image encryption and watermarking on three 512×512 test images (in milliseconds). The percentages in the brackets are the overheads of encryption/decryption and watermark embedder/detector added on the encoding/decoding processes.

Test Images	Encoding	Encoding-Encryption	Encoding-Watermarking	Decoding	Decoding-Decryption	Decoding-Watermarking
airplane	232.49	253.93 (9.22%)	251.04 (7.98%)	505.31	510.61 (1.05%)	507.38 (0.41%)
Lenna	225.93	245.25 (8.55%)	243.35 (7.71%)	504.29	513.94 (1.91%)	508.14 (0.76%)
peppers	234.28	256.69 (9.57%)	253.62 (8.26%)	511.48	514.68 (0.63%)	513.87 (0.47%)

[6] Mathworks Simulink Coder. `http://www.mathworks.com/products/simulink-coder/`.

[7] Mathworks Simulink HDL Coder. `http://www.mathworks.com/products/slhdlcoder/`.

[8] Open Data Flow (OpenDF). `http://sourceforge.net/projects/opendf`.

[9] Open RVC-CAL Compiler (ORCC). `http://orcc.sourceforge.net`.

[10] Ptolemy project home page. `http://ptolemy.eecs.berkeley.edu`.

[11] Reconfigurable Image Processsing (RIP) Library. `http://orc-apps.svn.sourceforge.net/viewvc/orc-apps/trunk/RIP/`.

[12] RVC implementation of JPEG codec. `http://orc-apps.svn.sourceforge.net/viewvc/orc-apps/trunk/JPEG/`.

[13] SmartMotion Project. `http://orc-apps.svn.sourceforge.net/viewvc/orc-apps/trunk/SmartMotion/`.

[14] Synopsys Studio. `http://www.synopsys.com/SYSTEMS/BLOCKDESIGN/DIGITALSIGNALPROCESSING/Pages/SystemStudio.aspx`.

[15] Daedalus. `http://daedalus.liacs.nl`, 2007.

[16] Cryptol: The Language of Cryptography. Case Study, `http://corp.galois.com/downloads/cryptography/Cryptol_Casestudy.pdf`, 2008.

[17] CAO and `qhasm` compiler tools. EU Project deliverable D1.3, Revision 1.1, `http://www.cace-project.eu/downloads/deliverables-y3/32_CACE_D1.3_CAO_and_qhasm_compiler_tools_Jan11.pdf`, January 2011.

[18] JM: H.264/AVC reference software. Current software version: 18.0, `http://iphome.hhi.de/suehring/tml`, May 2011.

[19] J. J. Ahmad, S. Li, M. Mattavelli, M. Wipliez, and M. Raulet. Crypto Tools Library (CTL): Applying RVC-CAL to Multimedia Security Applications. ISO/IEC JTC1/SC29/WG11, MPEG2010/m18404, 94th MPEG Meeting, Guangzhou, China. `http://www.hooklee.com/default.asp?t=CTL`, October 2010.

[20] H. I. A. A. Ali and M. N. I. Patoary. Design and implementation of an audio codec (AMR-WB) using data flow programming language CAL in the OpenDF environment. Technical report IDE1009, Master's Thesis in Embedded and Intelligent Systems, School of Information Science, Computer and Electrical Engineering, Halmstad University, Sweden, 2010.

[21] H. Aman-Allah, K. Maarouf, E. Hanna, I. Amer, and M. Mattavelli. CAL dataflow components for an MPEG RVC AVC baseline encoder. *Journal of Signal Processing Systems*, 63(2):227–239, 2011.

[22] I. Amer, C. Lucarz, G. Roquier, M. Mattavelli, M. Raulet, J. Nezan, and O. Déforges. Reconfigurable Video Coding on multicore: An overview of its main objectives. *Signal Processing Magazine*, 26(6):113–123, 2009.

[23] A. Antola, M. Fracassi, P. Gotti, C. Sandionigi, and M. Santambrogio. A novel hardware/software codesign methodology based on dynamic reconfiguration with Impulse C and CoDeveloper. In *Proceedings of 3rd Southern Conference on Programmable Logic (SPL'2007)*, pages 221–224. IEEE, 2007.

[24] S. Bhattacharyya, J. Eker, J. W. Janneck, C. Lucarz, M. Mattavelli, and M. Raulet. Overview of the MPEG Reconfigurable Video Coding framework. *Journal of Signal Processing Systems*, 63(2):251–263, 2011.

[25] J. Boutellier, V. M. Gomez, O. Silvén, C. Lucarz, and M. Mattavelli. Multiprocessor scheduling of dataflow models within the Reconfigurable Video Coding framework. In *Proceedings of the 2008 Conference on Design and Architectures for Signal and Image Processing (DASIP 2009)*, 2009.

[26] D. Ding, H. Qi, L. Yu, T. Huang, and W. Gao. Reconfigurable video coding framework and decoder reconfiguration instantiation of AVS. *Signal Processing – Image Communication*, 24(4):287–299, 2009. RVC implementation of Intra decoder is available at `http://orc-apps.svn.sourceforge.net/viewvc/orc-apps/trunk/AVS/`.

[27] J. Eker and J. W. Janneck. CAL language report: Specification of the CAL actor language. Technical

Memo UCB/ERL M03/48, Electronics Research Laboratory, University of California at Berkeley, 2003.

[28] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. SPARK: A high-level synthesis framework for applying parallelizing compiler transformations. In *Proceedings of 16th International Conference on VLSI Design (VLSI'2003)*, pages 461–466. IEEE Computer Society, 2003.

[29] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y.-P. Joo. PeaCE: A hardware-software codesign environment for multimedia embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 12(3):Article 24, 2007.

[30] Interra Systems. Vega H.264 Analyzer. `http://www.interrasystems.com/dms/pdf/Vega_H264_Datasheet.pdf`.

[31] ISO/IEC. Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines. ISO/IEC 10918-1 (JPEG), 1994.

[32] ISO/IEC. Information technology – MPEG systems technologies – Part 4: Codec configuration representation. ISO/IEC 23001-4, 2009.

[33] ISO/IEC. Information technology – MPEG video technologies – Part 4: Video tool library. ISO/IEC 23002-4, 2009.

[34] ITU-T. Advanced video coding for generic audiovisual services. Recommendation H.264, 2003 (last updated in 2010). also published as ISO/IEC 14496-10 under the title "Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding" in 2004 (last updated in 2010).

[35] J. W. Janneck, I. Miller, D. Parlour, G. Roquier, M. Wipliez, and M. Raulet. Synthesizing hardware from dataflow programs: An MPEG-4 Simple Profile decoder case study. *Journal of Signal Processing Systems*, 63(2):241–249, 2011.

[36] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, J. Riihimäki, and K. Kuusilinna. UML-based multiprocessor SoC design framework. *ACM Transactions on Embedded Computing Systems (TECS)*, 5:281–320, 2006.

[37] E. Khan, M. W. El-Kharashi, F. Gebali, and M. Abd-El-Barr. Applying the Handel-C design flow in designing an HMAC-hash unit on FPGAs. *IEE Proceedings - Computers and Digital Techniques*, 153(5):323–334, 2006.

[38] J. R. Lewis and B. Martin. Cryptol: High assurance, retargetable crypto development and validation. In *Proceedings of the 2003 Military Communications Conference (MILCOM'2003)*, pages 820–825. IEEE, 2003.

[39] J. Li and E. Abdel-Raheem. Modeling DV/DVCPRO standards on Reconfigurable Video Coding framework. *Journal of Electrical and Computer Engineering*, 2010:509394, 2010.

[40] C. Lucarz, M. Mattavelli, and J. Dubois. A co-design platform for algorithm/architecture design exploration. In *Proceedings of 2008 IEEE International Conference on Multimedia and Expo (ICME 2008)*, pages 1069–1072. IEEE, 2008.

[41] A. J. Menezes, S. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.

[42] A. Moss and D. Page. Bridging the gap between symbolic and efficient AES implementations. In *Proceedings of the 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM'2010)*, pages 101–110. ACM, 2010.

[43] R. Nikhil. Tutorial – BlueSpec SystemVerilog: Efficient, correct RTL from high-level specifications. In *Proceedings of Second ACM/IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE'2004)*, pages 69–70. IEEE, 2004.

[44] M. Noorkami and R. M. Mersereau. Compressed-domain video watermarking for H.264. In *Proceedings of 2005 IEEE International Conference on Image Processing (ICIP 2005)*, volume 2, pages 890–893. IEEE, 2005.

[45] K. V. Rompaey, D. Verkest, I. Bolsens, and H. D. Man. CoWare – a design environment for heterogeneous hardware/software systems. *Design Automation for Embedded Systems*, 1(4):357–386, 1996.

[46] G. Roquier, C. Lucarz, M. Mattavelli, M. Wipliez, M. Raulet, J. W. Janneck, I. D. Miller, and D. B. Parlour. An integrated environment for HW/SW co-design based on a CAL specification and HW/SW code generators. In *Proceedings of 2009 IEEE International Symposium on Circuits and Systems (ISCAS 2009)*, page 799. IEEE, 2009.

[47] W. R. Sutherland. *The On-Line Graphical Specification of Computer Procedures*. PhD thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, 1966.

[48] M. Thompson, H. Nikolov, T. Stefanov, A. D. Pimentel, C. Erbas, S. Polstra, and E. F. Deprettere. A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs. In *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'2007)*, pages 9–14. ACM, 2007.