UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Universitaria de Ingeniera Técnica de Telecomunicación

EXPLORATION OF RVC APPLICATIONS USING AN ARM MULTICORE PROCESSOR

EXPLORACIÓN DE APLICACIONES RVC EMPLEANDO UN PROCESADOR ARM MULTINÚCLEO

TRABAJO FIN DE MÁSTER

Autor: Miguel Chavarrías Lapastora

Ingeniero Técnico de Telecomunicación

Tutor:

Fernando Pescador del Oso

Doctor Ingeniero de Telecomunicación

Julio de 2012

Máster en Ingeniería de Sistemas y Servicios para la Sociedad de la Información

Trabajo Fin de Máster		
Título	Exploración de aplicaciones RVC empleando un procesador ARM multinúcleo	
Autor	Miguel Chavarrías Lapastora	VºBº
Tutor	Fernando Pescador del Oso	
Ponente		
Tribunal		
Presidente	Matías Garrido González	
Secretario	David Oses del Campo	
Vocal	Pedro Lobo Perea	
Fecha de lectura		
Calificación		

El Secretario:

ACKNOWLEDGEMENTS // AGRADECIMIENTOS

En primer lugar me gustaría dedicar unas líneas para agradecer todo el esfuerzo, el cariño y el apoyo que me han dado las personas con las que he compartido mi tiempo durante el último año y medio, en el que he estudiado el máster. En este tiempo han cambiado muchas cosas y he disfrutado de muchos buenos momentos con todas ellas.

Agradecer a todo el grupo del GDEM el apoyo, la ayuda y las ganas de seguir con este proyecto, y agradecer el seguir teniendo muchas ganas de seguir trabajando juntos. En especial a Fernando, mi tutor en este proyecto, agradecerle, que como siempre, no sólo haya sido una persona y un tutor excepcional sino que ha sido todo un placer trabajar una vez más con él.

Je remercie également toutes les personnes qui travaillent dans l'IETR de Rennes, qui m'ont aidé à tout moment avec les difficultés que le travail proposait, et au niveau personnel qui m'ont aidé à m'intégrer dans un pays étranger. En particulier mon chef de projet là-bas, Mickaël, pour tout ce que vous m'avez appris et l'aide que vous m'avez apportée pendant ce projet.

A toda mi familia, y en especial a mis padres, Mercedes y Pascual, y a mi hermano Guillermo por estar siempre ahí, por ser un referente y porque sin ellos no hubiese podido hacer realidad mis proyectos e ilusiones.

A mi primo Pablo, con quien comparto toda una vida de aventuras, proyectos, vivencias y buenos momentos, por estar siempre a mi lado en todo y por ayudarme con todo lo que he necesitado.

A mi amigo y colega Óscar por otro año más juntos en la universidad, y fuera de ella, por todos los momentos que compartimos, por tantas risas y tanto apoyo todos los días.

A todos mis amigos, que me alegran cada día que comparto con ellos y que me ayudan en los momentos difíciles, Álex, María, Laura, Miguel, Inés, Elena... Et tous les nouveaux amis que j'ai eu le plaisir de rencontrer à Rennes, pour me faire profiter, apprendre et découvrir une nouvelle ville et une terre unique, où je me suis senti comme chez moi, Avae, Michou, Jérôme, Max, Khaled, Guillaume... pour tout, merci beaucoup.

Agradecer al Ministerio de Educación y al Instituto Nacional de Ciencias Aplicadas de Rennes por aportar los fondos que han hecho posible la realización de este proyecto y la estancia en Rennes.

GENERAL INDEX

FIGURES INDEX

TABLE INDEX

RESUMEN

Las posibilidades ofrecidas por el aumento de la velocidad de la frecuencia de reloj de sistemas de un solo procesador están siendo agotadas. Las nuevas arquitecturas multiprocesador proporcionan una vía de desarrollo alternativa en este sentido. El diseño y optimización de aplicaciones de descodificación de video que se ejecuten sobre las nuevas arquitecturas permiten un mejor aprovechamiento y favorecen la obtención de mayores rendimientos.

Hoy en día muchos de los dispositivos comerciales que se están lanzando al mercado están integrados por sistemas embebidos, que recientemente están basados en arquitecturas multinúcleo. El manejo de las tareas de ejecución sobre este tipo de arquitecturas no es una tarea trivial, y una buena planificación de los actores que implementan las funcionalidades puede proporcionar importantes mejoras en términos de eficiencia en el uso de la capacidad de los procesadores y, por ende, del consumo de energía.

Por otro lado, el reciente desarrollo del estándar de Codificación de Video Reconfigurable (RVC), permite la reconfiguración de los descodificadores de video. RVC es un estándar flexible y compatible con anteriores codecs desarrollados por MPEG. Esto hace de RVC el estándar ideal para ser incorporado en los nuevos terminales multimedia que se están comercializando.

Con el desarrollo de las nuevas versiones del compilador específico para el desarrollo de lenguaje RVC-CAL (Orcc), en el que se basa MPEG RVC, el mapeo estático, para entornos basados en multiprocesador, de los actores que integran un descodificador es posible.

Se ha elegido un sistema embebido con un procesador con dos núcleos ARMv7. Esta plataforma nos permitirá llevar a cabo las pruebas de verificación y contraste de los conceptos estudiados en este trabajo, en el sentido del desarrollo de descodificadores de video basados en MPEG RVC y del estudio de la planificación y mapeo estático de los mismos.

ABSTRACT

Single core capabilities have reached their maximum clock speed; new multicore architectures provide an alternative way to tackle this issue instead. The design of decoding applications running on top of these multicore platforms and their optimization to exploit all system computational power is crucial to obtain best results.

Since the development at the integration level of printed circuit boards are increasingly difficult to optimize due to physical constraints and the inherent increase in power consumption, development of multiprocessor architectures is becoming the new Holy Grail. In this sense, it is crucial to develop applications that can run on the new multi-core architectures and find out distributions to maximize the potential use of the system.

Today most of commercial electronic devices, available in the market, are composed of embedded systems. These devices incorporate recently multi-core processors. Task management onto multiple core/processors is not a trivial issue, and a good task/actor scheduling can yield to significant improvements in terms of efficiency gains and also processor power consumption.

Scheduling of data flows between the actors that implement the applications aims to harness multi-core architectures to more types of applications, with an explicit expression of parallelism into the application.

On the other hand, the recent development of the MPEG Reconfigurable Video Coding (RVC) standard allows the reconfiguration of the video decoders. RVC is a flexible standard compatible with MPEG developed codecs, making it the ideal tool to integrate into the new multimedia terminals to decode video sequences.

With the new versions of the Open RVC-CAL Compiler (Orcc), a static mapping of the actors that implement the functionality of the application can be done once the application executable has been generated. This static mapping must be done for each of the different cores available on the working platform.

It has been chosen an embedded system with a processor with two ARMv7 cores. This platform allows us to obtain the desired tests, get as much improvement results from the execution on a single core, and contrast both with a PC-based multiprocessor system.

Chapter 1

Introduction and objectives

This first chapter is dedicated to the introduction of the work and to define the objectives. First subsection gives the general introduction to the themes discussed along the work. Subsection 1.2 gives the proposed objectives that are expected to overcome. Subsection 1.3 will contextualize the work and the objectives and last subsection 1.4 describes the structure of the document according with the work developed.

1.1. Introduction

Single core capabilities have reached their maximum clock speed; new multicore architectures provide an alternative way to tackle this issue instead. The design of decoding applications running on top of these multicore platforms and their optimization to exploit all system computational power is crucial to obtain best results. Dataflow program by essence extracts the available parallelism and eases the development and the deployment on future multicore devices. Scheduling management of tasks/actors can report important synergies between the subparts of these applications.

Today most of commercial electronic devices, available in the market, are composed of embedded systems. These devices incorporate recently multi-core processors. Task management onto multiple core/processors is not a trivial issue, and a good task/actor scheduling can yield to significant improvements in terms of efficiency gains and also processor power consumption.

Since the development at the integration level of printed circuit boards are increasingly difficult to optimize due to physical constraints and the inherent increase in power consumption, development of multiprocessor architectures is becoming the new Holy Grail. In this sense, it is crucial to develop applications that can run on the new

multi-core architectures and find out distributions to maximize the potential use of the system.

At the moment, there are significant developments studying the distribution of tasks on different multicore architectures topologies [BMS+09]. Also in [BSR11] the author presents a multicore distribution over FPGA, assigning each actor of the application to each core.

Scheduling of data flows between the actors that implement the applications aims to harness multi-core architectures to more types of applications, with an explicit expression of parallelism into the application.

On the other hand, the recent development of the MPEG Reconfigurable Video Coding (RVC) standard allows the reconfiguration of the video encoders and decoders. RVC is a flexible standard compatible with MPEG developed codecs [MAR10], making it the ideal tool to integrate into the new multimedia terminals to decode video sequences [ALR+09].

With the new versions of the Open RVC-CAL Compiler (Orcc), a static mapping of the actors that implement the functionality of the application can be done once the application executable has been generated. This static mapping must be done for each of the different cores available on the working platform.

An ARM based embedded system has been chosen to work due to its high flexibility and low power consumption. These characteristics are the same, making it the ideal system for the development of numerous commercial devices that are being launched on the market.

It has been chosen an embedded system with a processor with two ARMv7 cores. This platform allows us to obtain the desired tests, get as much improvement results from the execution on a single core, and contrast both with a PC-based multiprocessor system.

1.2. Objectives

In this first part of this master thesis memory; main objectives of the work developed are described. These objectives will focus the working lines. Should be noted detailed objectives can change during the development of the work. It is due to the possible changes occurred along the development of the initial objectives. Also it

could be due to the possibility to find other interesting topics within the field of study or due to specific changes in the standards used. These are the proposed objectives:

1. The first objective is to conduct a study of the MPEG RVC standard. This study must analyse main elements of the standard, how it works and practical implementations. Also will be interesting to study the objectives pursued by the standard itself, i.e. their own goals and solutions it provides.

   Once this MPEG RVC bases approaching has been realized the work will be centred in the working environment of MPEG RVC decoding applications. Main tool of this working environment will be *Open RVC-CAL Compiler* (Orcc). Then main elements of the compiler, needed packages and functionalities will be studied.

   At this point first practice work can be developed. In this sense main objective is to export the working environment of MPEG RVC decoding applications to an embedded system platform. This migration will able us to work with RVC-based video decoding applications over two developing platforms, a PC-based and an embedded system.

2. Second objective will be to study scheduling policies jointly applicable with the RVC decoders. This will be highly related with the usage of video decoders over multicore architectures, and this fact will be crucial in the achieving of results.

   In a first approaching, theory concepts about Dataflow Process Networks (DPN), main scheduling concepts, and their relation with the RVC-CAL structure will be studied. This will able to study and to apply specific improvements over the video decoders yet performed.

   Along the study of these new scheduling policies main concepts about the methodology used will be synthesized and the conclusions about their impact over the video decoding applications extracted.

3. Other objective of this work will be the study of the very begging of the Low-Level Virtual Machine (LLVM) usage. This last point of study will be focus in the main lines of the LLVM theory concepts, main objectives and the implementation of some practice modifications over a video decoder. All this work will be centred also in the multicore platforms usage.

   The main purpose of this objective is to start the study of the LLVM concepts and objectives trying to open a new development way as follows of the work already completed.

4. Finally, other intention of this work is to provide a guide about the initiation in these topics, able for future works. Also is important to provide a tutorial about practical procedures to start working with all the elements here introduced.

1.3. Context of the work

This work has been developed as a collaboration between the Polytechnic University of Madrid (Spain) and the *Institut National des Sciences Appliquées* (INSA) of Rennes (France), and two of their respective laboratories: The *Grupo de Diseño Electrónico y Microelectrónico* (GDEM) and the *Institut d'Éectronique et de Telecommunications de Rennes* (IETR). Both groups have a long trajectory working with new video codification techniques and an international recognised experience in this sense. The student was part of this collaboration assisted by a grant from the Spanish Ministry of Education as well as a research contract with the INSA.

The student is member of the permanent researching personnel of the GDEM and he was working in the IETR as temporal trainee during the second semester of their master studies.

This opportunity has allowed the student to broaden their knowledge by collaborating directly with the researching members of the IETR and understanding the work alright developed by this group. Also this collaboration has allowed strengthen relations between both research groups.

1.4. Structure of the document

This master thesis is structured as follows: current chapter 1 gives the introduction, objectives and context of the work developed. All theory concepts studied and needed to well understand the work here exposed are commented in chapters 2 to 6. Chapter 2 discuss about MPEG RVC standard, chapter 3 introduces main lines of the LLVM techniques and development for MPEG RVC usage. Chapter 4 details the compiler structure for this environment based on the Open RVC-CAL compiler (Orcc), and chapter 5 talks about the main theory concepts about Dataflow Process Networks (DPN) and scheduling policies, key concepts to understand the improving process done over the video decoders working with different multicore platforms. Finally, chapter 6 gives a description about here used multicore platforms and their principal features.

Once theory concepts have been introduced, chapter 7 discuss about the different experiments developed along the master thesis work. This experiments and first conclusions will allow us to obtain hoped results.

General conclusions obtained at the end of the work and proposed future developing lines and works are commented in chapter 8.

Finally, the different annexes include extra concepts that could be interesting for future works and developers. Mainly, information included in the annexes refers to the practical guidelines of the MPEG RVC and LLVM working environments.

Chapter 2

MPEG Reconfigurable Video Coding

MPEG[1] Reconfigurable Video Coding (RVC) framework is a new ISO standard that aims at providing specifications of video decoders at the level of component libraries, instead of continuously working with monolithic architectures and algorithms. RVC solves the problem of the current video standards by defining two new standards: a language to describe the video codecs and a tool library for video coding algorithm used within MPEG codecs.

This section describes the main lines of MPEG RVC, the current state of the standard and the projects yet developed. Section 2.1 introduces the basic concepts of RVC, the situation motivating the development of the new standard and its objectives. Section 2.2 deals with the different parts of the standard giving their functionalities. In Section 2.3 the CAL actor language is then briefly introduced. Finally, in section 2.4, current situation of MPEG decoders yet developed and nowadays available for MPEG RVC are described.

2.1. MPEG RVC description and main concepts

MPEG working group has a long history into the development of standards for video, image and audio processing and compression techniques. This group started their activities in 1988 with MPEG-1 standard for video coding, after it, MPEG-2 was developed in 1994 and then MPEG-4 in 1998.

[1] Moving Picture Experts Group, working group of the International Organization for Standardization (ISO) and the International Electro-technical Commission (IEC)

The new developed standards were created to enhance efficient video coding techniques with the corresponding increasing of the inner complexity. These new standards also brought the increasing of the development times making more and more difficult the creation of new standards and most of time forbidding evolvement of yet developed systems. By the other side video standards were also developed by other standardisation bodies such as ITU an alternative of MPEG ones. It was motivated by the increasing of new digital devices with video representation capability and the expansion of digital TV.

At this moment MPEG started to reorder the standardization of their standards. MPEG-1, 2 and 4 standards were followed by MPEG-A (Multimedia application formats), MPEG-B (Systems technologies) or MPEG-C (Video technologies), and MPEG-H which will include the video coding standard. In this sense MPEG RVC is standardized among MPEG-B part 4 [ISO09] and MPEG-C part 4 [ISO08c], and the development of the standard was started in 2005. MPEG-RVC is based on a dataflow diagram to represent video decoders and on a dataflow language namely CAL to program all boxes of the diagram. Fig. 1 shows the evolution of the CAL language and the RVC standard.

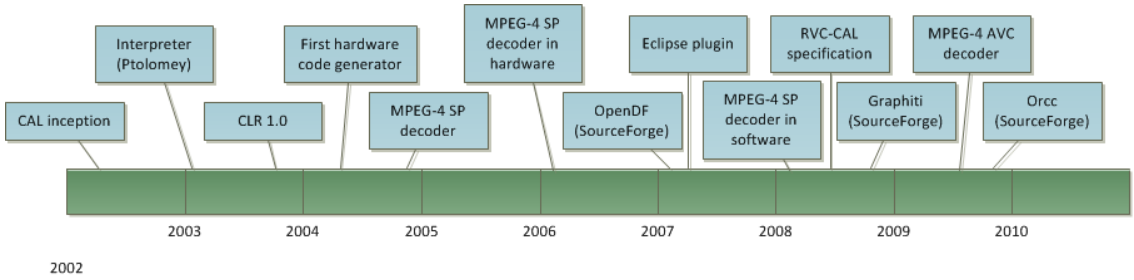

Fig. 1: CAL and RVC standard timeline.

MPEG RVC standard arises as a solution to the old monolithic way to program more and more complex video standards. Furthermore it allows the reconfiguration of the video encoders and decoders, and is a flexible standard compatible with MPEG developed codecs [GWPR11], making it the ideal tool to be integrated into the new multimedia terminals to decode video sequences whatever the bitstream it has to deal with. MPEG-RVC framework is a new ISO standard that aims at providing specifications on video decoding at the level of component libraries trying to replace the old reference software written in C or HDL languages. RVC solves the problem of the above standards by defining two new standards: a language to describe the video codecs and a tool library for video coding algorithm used within MPEG standards. It will

replace old reference software by abstract decoder representations or Abstract Decoder Models (ADMs).

Fig. 2 shows the general structure when implementing RVC decoders. Main elements of the structure are included in the figure and their respective place in the scheme.

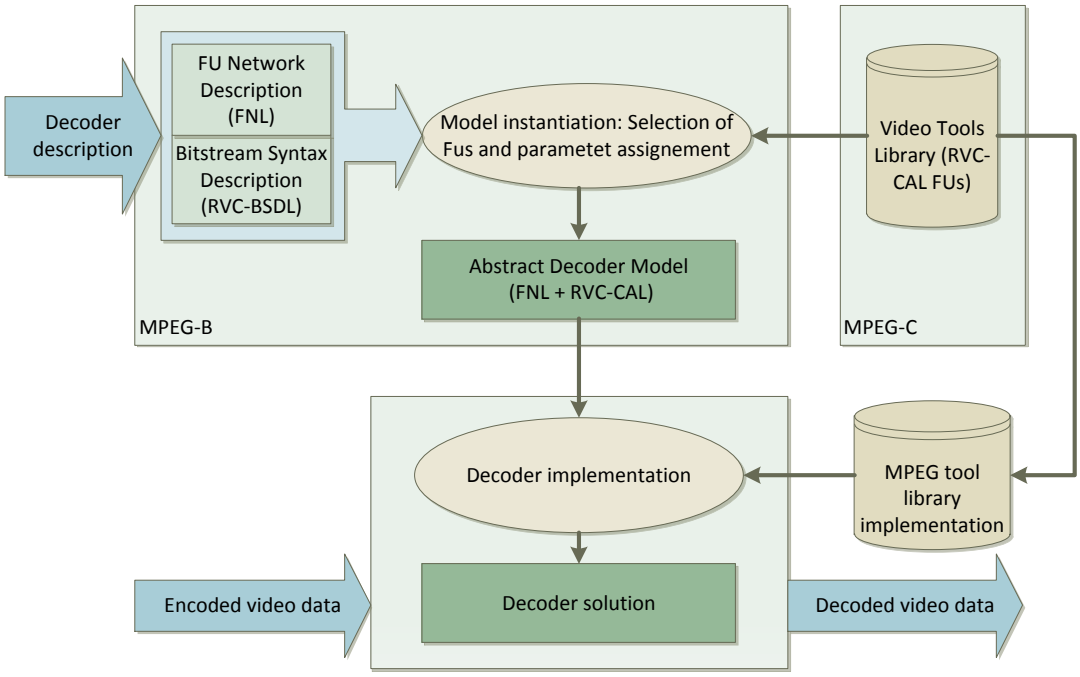

Fig. 2: RVC decoder implementation.

In this sense the objective of MPEG RVC is to perform decoders from a set of high level abstraction specifications based on dataflow model paradigms. Then, an ADM will be a decoder model described as a dataflow diagram. The advantages offered by an ADM representation are:

- Abstract programming language: A language dedicated for programming a decoding algorithm with the dataflow model without overspecifying the scheduling of the inherent decoder will help to developers to focus on the main compression algorithms than into the details of low level architecture.

- Scalable parallelism: Given that new programming methods are based on dataflow models it facilitates exploiting the parallelism of the developed programs.

- Reuse: The fact that most of coding technologies have some commonalities and that can be easily described with MPEG RVC framework, helping the development of future standards by reusing the work already done.

- Modularity: Finally, the integration of new efficient decoding tools is simplified by the high encapsulated level of the coding described tools. The dataflow description allows the modularity of decoder configuration and favours slight coding by only changing the network topology.

All these considerations provide a better starting point for developing new standards, whilst facilitating and simplifying workflows for improvement and development of efficient video decoders.

2.2. Specification of MPEG RVC decoders

The MPEG RVC framework has been standardized by the MPEG committee as parts of standards MPEG-B part 4 and MPEG-C part 4. Nowadays, these two standards are still evolving in order to include upcoming video algorithm included in future decoder descriptions. According to this, the ADM will represent the specification of a chosen profile of a decoder from the existing MPEG standards. Technically, MPEG RVC is presented with the form of a coding tools library, a decoder description language and a dataflow syntax description. Then, an ADM is set in the form of a *Video Tool Library* (VTL), a *Functional Unit Network Language* (FNL) and a *Bitstream Syntax Description Language* (BSDL) (see Fig. 3).

The algorithms of the applications will be encapsulated into independent entities called *Functional Units* (FUs); these entities are a set of coding tools provided in the VTL. Each FU is specified in two ways, a textual specification and a description performed with the reference language RVC-CAL Actor Language (RVC-CAL).

The specific configuration of the FUs needed to form a complete profile of a decoder will be expressed by the FNL, and this configuration is performed as a graph in which FUs are vertices and the communications between these FUs are edges.

The last element is the BSDL, which describes the parsing process of a coded bitstream for the specified configuration. It is presented as an XML document where all the syntax elements of the bitstream are described with the objective to link them to the corresponding interpretation rule/FU. BSDL was described in MPEG-B part 5 [ISO08a],

but their inner details are not closely related to the work done in this thesis, more information about this language description can be found in [PHH+03].

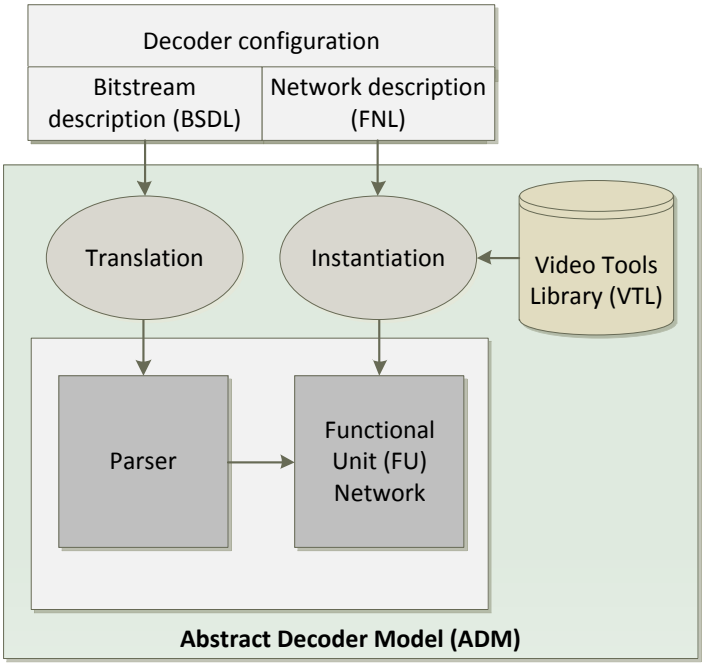

Fig. 3: Specification of an Abstract Decoder Model in MPEG RVC.

As can be seen, the VTL is one of the most relevant elements of functional architecture. It is the library of the video coding tools, the FUs, which are standardized by MPEG-C part 4. According to it, each FU has a textual specification and a corresponding code implementation written in RVC-CAL.

The textual specification gives the FU name, a short functionality description of it, the original standard and profile it comes from, and the properties of its input and output data (see Table 1). There will be two kinds of FU:

- Algorithmic (ALGO_) coding tools, these will perform tasks as the *Inverse Discrete Cosine Transform* (IDCT) or *Inverse Quantifier* (IQ).
- Data Management (MGNT_) tools, that will perform multiplexing and demultiplexing operations.

As was introduced before, one of the objectives of this framework is the reusability of the coding elements. In this sense FUs who contains algorithmic video coding tools are normally reusable by different profiles or standards. However the data management tools will adapt the FUs to the specific structure, and are specific to the decoding structure of the decoder.

To illustrate these concepts in Table 1 the textual description of an example FU is presented. In this case this FU is an algorithmic type, and it is used in MPEG-4 part 2 for the inverse scan operation. This is a simple FU and will be used in the following discussing parts to exemplify the concepts introduced.

FU Name	***Algo_IS_ZigzagOrAlternateHorizontalVertical_8x8***
Description	The functionality is to invert the one-dimensional array of the coefficients ordered in zigzag, alternate vertical or alternate horizontal, depending on AC_PRED_DIR value. It inputs a list of 64 integer coefficients (8x8 blocks) and outputs the ordered list of integer according to the value of the token AC_PRED_DIR.
Profiles@levels	MPEG-4 Simple Profile (SP)
INPUT	
Name	Token
AC_PRED_DIR	AC_PRED_DIR token
QFS_AC	AC token
OUTPUT	
Name	Token
PQF_AC	AC token
PARAMETER	
Name	Description | Range

Table 1: Textual description of the FU Algo_IS_ZigzagOrAlternateHorizontalVertical_8x8.

Looking into Algo_IS_ZigzagOrAlternateHorizontalVertical_8x8 FU one can see that it has two inputs: AC_PRED_DIR and QFS_AC whose data types are *AC_PRED_DIR* and *AC* respectively, these data are processed and transmitted to the PQF_AC output. All data types, called tokens (atomic pieces of data), refer to an identifier. Fig. 4 shows the environment of this FU and the respectively connections between it and its neighbours.

Other consideration about FUs is that it can be implemented in any software language or hardware language seeing it as “black box” components as long as their external behaviour has not been changed by the proprietary implementation.

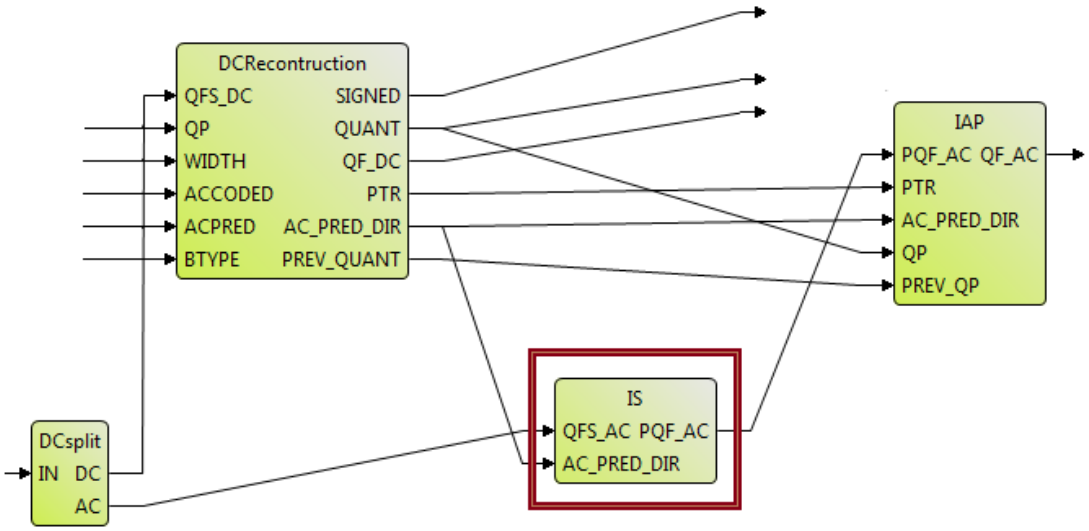

Fig. 4: Algo_IS_ZigzagOrAlternateHorizontalVertical_8x8 FU environment.

2.3. RVC-CAL Actor Language brief summary

RVC-CAL Actor Language (RVC-CAL) [BEJ+09] is a high level dataflow programming language to describe the behaviour of the FUs. The internal computation of the FUs is described with an entity called *actor*. The actor contains interfaces, input and output ports, internal states and parameters. The following lines comment their particularities and next figures show the most relevant code lines of the implementation of the actor for the Algo_IS_ZigzagOrAlternateHorizontalVertical_8x8 FU, taken it as a didactic example.

2.3.1. Basic concepts about RVC-CAL Actor Language

In a first approaching to the RVC-CAL use, in the declaration of the actor their I/O ports must be declared as *actor* <name>*ID* (<parameters>) *In ==> Out:*

```
  actor XX() In: [x] ==> Out: [x] (…)
end
```

Fig. 5: Simple declaration in RVC-CAL of an actor.

The processing of an FU is described according to a sequence of atomic steps called *actions*, which are described in the body of any actor. The execution of an action is called *firing an action* and during that, the execution of an actor will consume input tokens, produce output tokens and change its internal state. This happens for all actors and this procedure is the same for any firing cycle.

The actors can have one or more actions. Non-deterministic execution cycles can happens if the execution conditions are not specified for all possible situations. Fig. 6 exemplifies this situation in which both actions can be fired at same time, in this case the selection of the action that will be fired are indeterminate.

```
action Input1: [x] ==> [x] end
action Input2: [x] ==> [x] end
```

Fig. 6: Non-deterministic situation for two actions described in the body of an actor.

The execution of an action will be performed when the input data is available and the *guard* condition is true, if these are described. This guard condition can refer to a token input value, the state of the actor or both (see Fig. 7: Guard structure in a CAL actor.). Depending on the actor's own state this could be used to decide next actions to select. Action guarded by an actor state can be fired/triggered once the condition is true and the action has enough token on its input ports. This is part of the normative CAL programming language, the action firings regulation is managed by a *Finite States Machine* (FSM), input tokens, guards and priorities.

```
actor Select() S, A, B ==> Output:
      action S: [sel], A: [V] ==>[V]
            guard sel end
      action S: [sel], B: [V] ==>[V]
            guard not sel end
      end
```

Fig. 7: Guard structure in a CAL actor.

In the header of any actor, its name, its interfaces, and optionally its parameters are specified in order (see Fig. 8). The package identifies the profile and the standard of the respectively coding tool it comes from. Also there are collections of functions called *units* that can be reused across FUs. Therefore, the header of the actors will conform to their textual description.

```
package org.sc29.wg11.mpeg4.part2.sp.texture;
import org.sc29.wg11.mpeg4.part2.Constants.*;

actor Algo_IS_ZigzagOrAlternateHorizontalVertical_8x8 ()
      int(size=3) AC_PRED_DIR,
      int(size=SAMPLE_SZ) QFS_AC ==> int(size=SAMPLE_SZ) PQF_AC : (…)
```

Fig. 8: Header of the RVC-CAL implementation for Algo_ZigzagOrAlternateHorizontalVertical_8x8.

An action may contain a description that defines the series of statements to be applied to the involved variables. Also external *functions* declared in the actor and connected to a *label* can be called.

```
List(type: int(size=7), size=192) Scanmode = [(…)]; //Buffer for Ac coeffs
int BUF_SIZE = 128;
int(size=8) count := 1;          (…)
function wa() --> int :
       (count & 63) | if half_ then     64 else 0 end
       end
```

Fig. 9: State variables and functions of the RVC-CAL implementation for Algo_ZigzagOrAlternateHorizontalVertical_8x8.

The actor here chose as example Algo_ISZigzagOrAlternateHorizontalVertical_8x8 is integrated by several actions, Fig. 10: Example of RVC-CAL actions for implementation of Algo_ZigzagOrAlternateHorizontalVertical_8x8. shows some of variable declarations and function definition. Actions *skip* and *start* read one token from input AC_PRED_DIR. The action *skip* may fire if and if only the first data on AC_PRED_DIR has a strictly negative value. By the other side, action *start* fires for any value on AC_PRED_DIR. The action *read_write* reads one token from QFS_AC and produces one token on PQF_AC. This action fires according to the value of *count* and has some side effects on state variables.

```
skip: action AC_PRED_DIR:[ i ] ==> //Wait for a valid Inverse Scan
       guard  i < 0  end

start:  action AC_PRED_DIR:[ i ] ==>
       guard  i >= 0       do
              add_buf := i;
       end

       done: action ==> //Done reading or writing 64 ac coeffs
       guard count = 64
(…)
```

Fig. 10: Example of RVC-CAL actions for implementation of Algo_ZigzagOrAlternateHorizontalVertical_8x8.

In Fig. 11 the code for the FSM is presented. It gives legal firing sequence for all actions described for this actor. Also here the priority declarations are included, in this case the *skip* action has the highest priority.

```
schedule fsm rest :
       rest ( skip ) --> rest ;
```

```
      rest ( start ) --> read ;
      (...)
      full ( start ) --> both ;
      both ( read_write ) --> both ;
      both ( done ) --> full ;;
end

priority
      skip > start ;
      done > read_write ;
end
```

Fig. 11: RVC-CAL FSM and priorities of Algo_ZigzagOrAlternateHorizontalVertical_8x8.

2.3.2. Decoder configuration: the Functional Unit Network Language

The Functional Unit Network Language (FNL) expresses the interconnection between the coding tools available in the VTL performing a network with inputs and outputs. The final ADM will be configured with a network diagram. The standard of FNL is available in [ISO11]. In this sense, networks can be hierarchical, when some network may be a part of a more extended network, and can be used to give parameters to the actors or subnetworks.

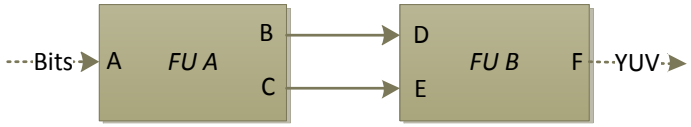

Fig. 12: MPEG RVC configuration of decoders.

In Fig. 12 a simple example of a decoder configuration diagram is presented. This diagram corresponds to an FNL representation detailed in Table 2. As already commented in Chapter 2 section *2.2 Specification of MPEG RVC decoders,* the configuration of the decoders is managed as an oriented graph where vertices are the instances of the actors and edges represent data flowing between actors. These instances are selected among a list of FUs, and are defined by their corresponding identifiers and names attributes. Optionally, instances can assign values to the parameters of an actor.

Three types of edges are defined by FNL:

1) Between an input port of a network and an instance → Input
2) Between an output of an instance and an input port of a network → Connection
3) Between an output port of an instance and the output port of a network → Output

Instances	<Instance id=" FU_A "> <Class name =" Algo_Example1 "/> </ Instance > <Instance id=" FU_B "> <Class name =" Algo_Example2 "/> </ Instance >
Connections	<Connection src=" FU_A " src -port ="B" dst=" FU_B " dst -port ="D"/> <Connection src=" FU_A " src -port ="C" dst=" FU_B " dst -port ="E"/>
Input	<Input src=" FU_A " src -port ="A"/>
Output	<Output src=" FU_B " src -port ="F"/>

Table 2: A decoder configuration using the Functional Network Language.

2.4. MPEG standards in MPEG RVC

This section verse about the evolution of the different standards of MPEG included or developed for MPEG-RVC. The study is currently centred on MPEG-4 part 2, also known as MPEG-4 *Simple Profile* (SP) and MPEG-4 part 10, as MPEG-4 *Advanced Video Coding* (AVC) or H.264. For the last one MPEG RVC, it covers the *Constrained Baseline Profile* (CBP) and *Progressive High Profile* (PHP) profiles.

Other standards and profile levels are currently developed, as most important are MPEG-2 *Main Profile* (MP), MPEG-4 part 2 *Advanced Simple Profile* (ASP), and *Scalable Video Coding* (SVC) [SMW07].

As an approaching to the new perspective given by MPEG RVC, Fig. 13 shows the general diagram of the top level view for MPEG-4 part 2 SP and part 10 CBP decoders. As can see the diagram symbolises the global functionality of the decoder. This diagram is composed of five actors, and it groups multiple FUs into one top level actor and data into one port.

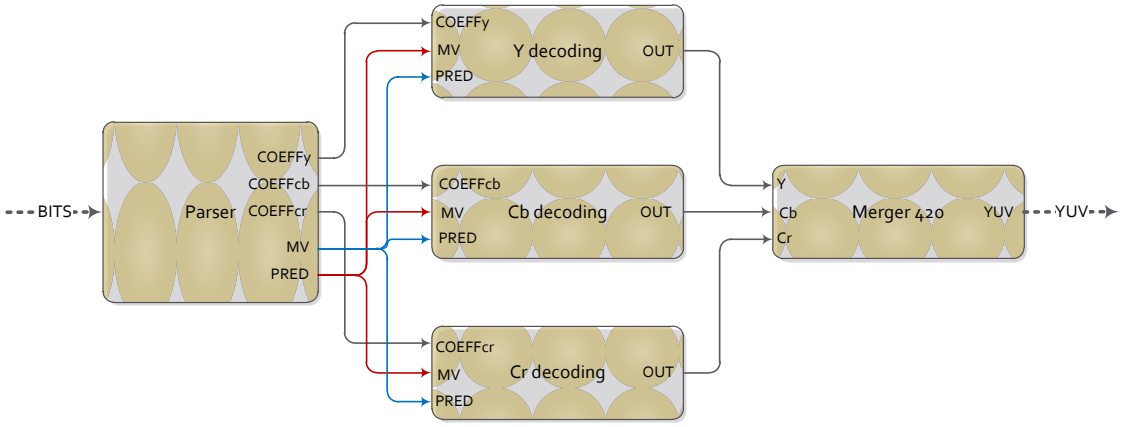

Fig. 13: MPEG RVC configuration of decoders

As also can see in Fig. 13, the three component decoding actors are performed in three separate actors to express the maximum of concurrency during the decoding process. Respectively outputs are connected to the *Merger 420* that produces the desired video stream on a single output called *YUV*.

All MPEG decoders are based on hybrid decoding schemes, including both intra and inter predictions. Each decoding component of MPEG-4 part 2 SP and MPEG-4 part 10 CBP comprise the same hybrid structure.

As example, the texture decoding of MPEG-4 part 2 SP provides a spatial (intra) prediction and residual information for the (inter) prediction. Its lower hierarchy is composed of one sub-network and 4 actors.

The MPEG-4 part 10 contains a number of new features that add to the SP hybrid structure. First, MPEG-4 part 10 supports many intra and inter predictions depending on the Profile used. In the case of the CBP, the decoder has two block-intra predictions and one inter-prediction mode on a single reference frame. Each of these prediction modes are surrounded by a demultiplexer (Demux) and a multiplexer (S elect) to easily add/remove them. This structure makes it possible to switch between several MPEG-4 part 10 Profiles without changing the overall structure of the decoder.

For instance, adding the 8x8 intra-prediction mode and the Bi-directional inter-prediction mode into this structure switches the decoder into the Progressive High Profile configuration.

Chapter 3

MPEG-RVC on a Low-Level Virtual Machine (LLVM)

As introduced in chapter 2, MPEG RVC aims to produce ADMs of MPEG decoders as programs described in RVC-CAL dataflow language. MPEG RVC facilitates development processes of decoders by building decoders at level of components of libraries [Gor11]. As next improvement step, this section will introduce the Low-Level Virtual Machine (LLVM) mainlines. The final goal of LLVM is to allow us to design decoders that can dynamically instantiate several RVC decoder descriptions. This chapter has two subsections: in 3.1 main concepts of LLVM are introduced and subsection 3.2 verses about the concept of the LLVM-based dynamic decoder.

3.1. LLVM main concepts

The main objective in the development of LLVM is to provide a system whereby be possible create dynamical decoders instead of the static ones generated by RVC tools, but saving likely features offered by the RVC description yet commented, portability, scalability and reconfigurability (see Chapter 2, section 2.1).

Before LLVM develop there was no way to use dataflow description to form RVC decoders in an automatically and dynamically way. The work done by researchers groups [GWPR11], propose to transform the RVC-CAL description to coding tools into the generic low-level description called LLVM Intermediate Representation (LLVM IR). This LLVM infrastructure will be used for dynamically and efficiently instantiating of the coding tools to create the new video decoders. Then, RVC and LLVM concepts will be combined to create a portable MPEG decoder engine that can reconfigure MPEG RVC decoder description.

Main idea is to perform a dynamic synthesis and execution of an ADM description, integrated inside the target platform. This able to keep all the information contained in the ADM description of the decoder and to use a Virtual Machine (VM) dedicated to RVC ADM for a dynamic execution of the decoder. Fig. 14 illustrates this idea; final decoder will take side information of the ADM for a dynamic generation of decoders type 1 or 2, or a hybrid version of them.

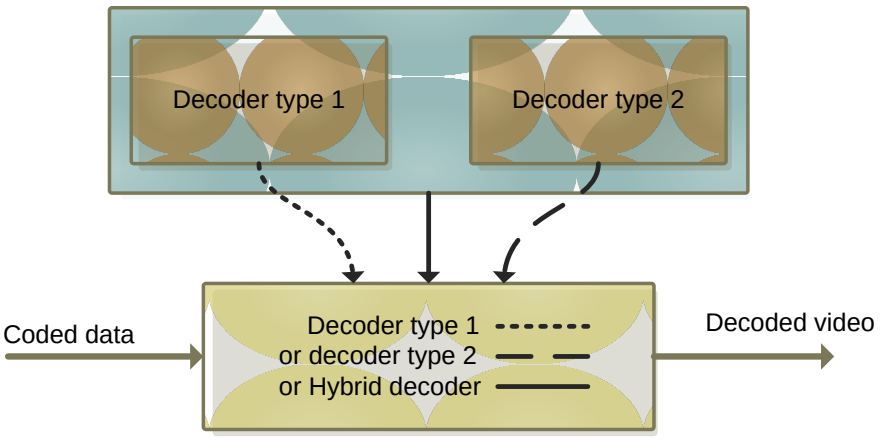

Fig. 14: Representation of the dynamic MPEG RVC decoder.

3.1.1. LLVM Intermediate Representation

The LLVM is a low-level representation of applications, close to assembly languages. It captures the key operations of ordinary processors but avoids machine specific constraints such as physical registers or pipelines. It has been designed to be a low-level representation but with high-level type information for compiler analysis and optimization. In this sense the Intermediate Representation (IR) used by LLVM is one of the key factors that differentiate it from other systems; in [Lat02] the justification of LLVM IR design choices can be read.

The LLVM IR is composed of an infinite set of typed virtual registers, coded in Three Address Code (3AC) form and in Static Single Assignment (SSA) form. Both can hold values of primitive types, as integral, floating point, or pointer values [Lat02].

Through the LLVM transformation [GWPR11], LLVM programs transfer values between virtual registers and memory exclusively via *load* and *store* operations using typed pointers. In the reference manual of LLVM [LA06], also more information about syntaxes and semantics of LLVM instructions are given.

```
%X = add i32 4, 9 ; Affect4 added to 9 in %X
%Y = call i32 @hasToken() ; Call hasToken function
%cond = eq i32 %Y, 1 ; Produces a bool value
br i1 %cond, label %True, label %False ; Cond. branch
True:
```

Fig. 15: LLVM IR coding example.

3.1.2. Just-In-Time Compiler

There is a compilation framework that allows exploiting the LLVM IR to provide a combination of features. These capabilities are the followings and they weren't on previous approaches [LA04]:

- *Persistent program information:* used compilation models save the LLVM representation along the lifetime of any application. This allows us to improve and optimize at the execution level.
- *Transparent runtime model:* no particular object model, exception semantics, or runtime environment is specified by the system. This will allow compiling any language over the system.
- *Uniform, whole-program compilation*: the fact of language independence permits to optimize and compile all code including an application in an evenly way.

The compilation framework corresponds to a collection of libraries and tools that makes easier to build offline compilers, optimizers or Just-In-Time (JIT) code generators.

3.2. LLVM and its extension for MPEG RVC

The LLVM JIT compiler represents the core of the proposed dynamic MPEG RVC decoder (see Fig 24). The LLVM is surrounded by two RVC specific components that bring LLVM compliant with RVC ADM:

1) The first component is an equivalent representation of a VTL provided by MPEG RVC, described as LLVM representation. This *portable VTL* must keep same information from RVC-CAL actors with a low-level representation of their own computational description to be manageable by LLVM.
2) The second component is the Just-In-Time Adaptive Decoder Engine (JADE). It works as a layer of the LLVM JIT Compiler, managing the description and the configuration of the ADM and the portable VTL to produce decoders in LLVM

IR. In the end, the resulting LLVM IR of an ADM will be sent to the JIT Compiler to produce the targeted machine code. JADE is also designed to manage network scheduling instructions and to control the execution of the final decoder.

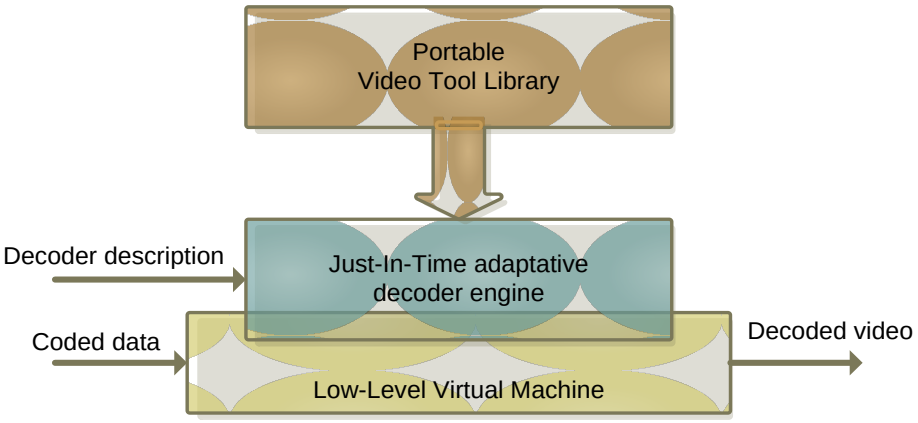

Fig. 16: Infrastructure of the dynamic RVC decoder.

Chapter 4

RVC-CAL compiler infrastructure

Open RVC-CAL Compiler (Orcc), is the development environment and it is an Integrated Development Environment (IDE). Orcc is designed for the creation, edition, transformation, analysis and debug of ADMs. This section discuss about their particularities, main concepts, specific packages and the role that plays as one of the main tools into the development of the work exposed in this document. In subsection 4.1 mainlines of Orcc are commented. Subsection 4.2 introduces base concepts about Orcc Intermediate Representation (IR), after this, section 4.3 links these ideas with the LLVM development yet introduced. Subsections 4.4, 4.5 and 4.6 are dedicated to the practice usage of Orcc and the role into the work developed and exposed in this document. Subsection 4.4 talks about the usage of this environment into the work developed; subsection 4.5 gives the list of extra packages used and subsection 4.6 introduces basic concepts of the usage of Orcc for applications that will run over multicore platforms.

4.1. Development environment with Orcc

As yet introduced, Orcc is an IDE for the design and performance of ADMs. This wide tool integrates an XDF network editor based on Graphiti[2], an RVC-CAL actor's editor based on XText [EV06] and a set of tools for the analysis and compilation of ADMs.

Orcc was developed as a result of the work done with the Cal2C compiler [RWR+08]. Orcc extends it for making a compilation environment dedicated to MPEG RVC with the RVC-CAL language, and for some software languages (Java, C++, C,

[2] Graphiti is available in this website: http://orc-apps.sourceforge.net/

LLVM...), hardware solutions (VHDL and TTA) and decoders [BSR11, SNR11]. Fig. 17 shows the general Orcc compiler infrastructure and illustrates the available *back-ends* in Orcc.

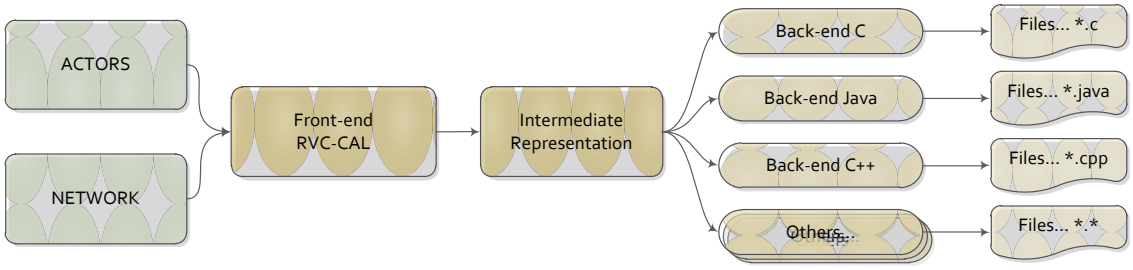

Fig. 17: Orcc compiler infrastructure.

The compiler for the development environment of Orcc is performed by an IR (Intermediate Representation) specific for RVC-CAL language. The IR used in Orcc is conservative in terms of the semantic and the structure of the original RVC-CAL actors.

This IR permits that the body of the actions is however decomposed in the form of *load and store* instructions as *Static Single Assignment* (SSA) form. This will facilitate the conversion to other languages [RWZ88]. In [Wip10] the general discussion about design choices is commented. The IR provides the simulation and debugging layers for the ADMs in RVC.

4.2. Orcc Intermediate Representation

Orcc IR is a conservative representation of a dataflow program in terms of structure and semantic while being at a lower level of representation. All actor representation is serialized in JavaScript Object Notation (JSON) description format. As example Fig. 18 shows the structure of the *Absolute Value* actor in Orcc IR. For all descriptions, the JSON will contains the name, pattern and a list of actions of the actor, and the Finite State Machine (FSM) of the actor will be translated as a list of *state-to-state* transitions. Finally, the description of the priorities, if it is given, will become a list of action tags in decreasing order.

```
"name": "Abs",
"inputs" : [ "int", [ 16 ] , "I" ],
"outputs": [ "uint", [ 15 ] , "O" ],
"actions": [
      [ (...) ]
"action _scheduler ": [
            "s0", // Initial states
            [ "s0", "s1", "s2"], // States
            [
                  [ "s0", [ [ [ "b" ], "s1" ], [ [ "a" ], "s2" ] ] ],
```

```
                [ "s1", [ [ [ "n" ], "s0" ] ] ],
                [ "s2", [ [ [ "p" ], "s0" ] ] ]
            ]
        ]
```

Fig. 18: Description of the structure of *Absolute Valor* actor in Orcc IR.

In Fig. 19 the JSON description of an action is introduced, it contains the action tag and all the produced and consumed tokens. All the sentences in the action are described as simple arithmetic expressions. The assignment statement is differentiated into assignments to local variables and *load/store* to memory operations. The high-level RVC-CAL functional expressions are translated into their equivalent in low-level IR expressions, containing functional calls, conditions or list generators.

```
"pos",false,[32 ,54 ,5],"void",[],
[
        [["O"],,["List"[1], ["uint",[15]]]],
        [["I"],["List",[1], ["uint",[16]]]],
        [["u"],["uint",[15]]]
],
[
        ["read”,[["I"],"I",1]] ,
        ["load",[["u" ,1],["I"],[0]]] ,
        ["store",[["O"] ,[0] ["var",["u",1]]]] ,
        ["write",[["O"],"O",1]]
]
```

Fig. 19: Description of the action *pos* of *Absolute Value* actor in Orcc IR.

The value and number of tokens on the input of an action and also the guard condition (if it exists) are tested by the *isSchedulable* function (see Fig. 20). It is performed with the firing conditions of the actions yet commented. In the end, if *isSchedulable* function returns a true value it means that the current action can be fired.

```
"isSchedulable_pos","bool",   [
        [["_tmp",1,1],"bool"],
        [["_tmp",0,1],"bool"],
        [["_tmp",0,2],"bool"],
        [["_tmp",0,3],"bool"]
],[
        ["hasTokens" ,[] ,[["_tmp" ,1,1],"I" ,1]] ,
        ["if",
        ["var" ,["_tmp",1 ,1]] ,
        ["assign" ,[["_tmp",0 ,1] ,[true]]] ,
        ["assign" ,[["_tmp",0,3] ,[false]]]
        ],
        ["join",[["_tmp",0,2],[["_tmp",0,1],["_tmp",0,3]]]] ,
        ["return",["var",["_tmp",0,2]]]   ]
```

Fig. 20: Description of the firing condition of the action *pos* of *Absolute Value* actor in Orcc IR.

Finally, FIFO's operations management will become *read*, *write*, *hasTokens* or *peek* statements. Over Orcc IR is necessary because the specified semantic of RVC-CAL describes that the input and output patterns may have read/write several tokens as a list, or may have to reorder tokens.

4.3. From RVC-CAL compiler IR to LLVM

At the moment of perform a translation from an RVC-CAL FU into LLVM IR the high-level information from the original model must be kept. This must save a very low-level description of the behaviour of the FU implied.

All the translation process is designed for the portable VTL, -i.e. from an RVC-CAL FU of the VTL into an LLVM-equivalent representation- on the Orcc IR as this representation is closer to the LLVM IR. In this sense, the compilation framework of Orcc for a chosen RVC-CAL dataflow program to target a specific language is performed in two steps:

1) A specific front-end parses the FUs of a chosen network and translates them to an Orcc specific IR.
2) A dedicated back-end loads both, actors and network, in Orcc IR from a generate code in the targeted language.

Also consider that to produce the portable VTL needed by JADE (see section 3.2), it is needed to develop a new LLVM back-end that starts from the low-level Orcc IR to perform the LLVM IR of an integral VTL.

4.4. Working environment with Orcc

Once theory concepts have been introduced, this section summarizes the environment used along the work developed in this end master project. This section introduces general practical procedures to obtain video decoders from a given RVC-CAL description and the corresponding dataflow. Detailed information about installation procedures for PC-based and for an embedded system platforms are given in ANNEX I.

Fig. 30 illustrates the general working scheme, and is a general guidance to take a major point of view of the whole working procedures. Also a short description about main tools here mentioned is presented in the next subsection 4.5. This general

scheme allows the designer to know the current status of the work and focus into the following tasks to perform.

In the work here developed, the decoding applications are yet developed, and the work will take them to apply improving tasks or simple modifications. In this sense the RVC-CAL code description and dataflow diagrams are yet available. These applications must be downloaded to the Eclipse's workspace, once it, the code in the target language (C, C++, LLVM ...) can be generated by Orcc.

As yet commented Orcc works over Eclipse and the compilation details will be chosen on it. At same moment, the static management of the assignment of the actors to one of the available cores in the multicore platform can be done. These concepts are commented in

Chapter 7. At first, Orcc will generate a *.xcf file with the description of the static assignment.

All the working structure provides a full compatible environment. It is supported by the fact that the decoding applications can be generated in some languages and that the rest of files and following procedures have been already tested over many different systems. Also rest of packages and needed tools for finally generate the executable decoder are fully compatible over different platforms.

Once the decoding solution files have been generated by Orcc, these must be compiled with CMake (see subsection 4.5) for the target platform. Last step corresponds to the final executable file generation. This will depend on the target platform to run the decoder, as example if the system is Windows-based, the Visual Studio (see subsection 4.5) developing platform can be used in this sense and also for debugging tasks. If the system is UNIX-based the executable can be obtained with a simple GCC compilation.

At this moment the decoding application is ready for running, and it is already for testing and improving tasks. The static assignment of the actors who implement the application can be easily modified without perform any changes into the decoder. The static assignment is described in the *.xdf file and it can be modified by a standard text editor.

During the execution time of the decoder, it will provide information about the number of decoded frames per second (FPS).

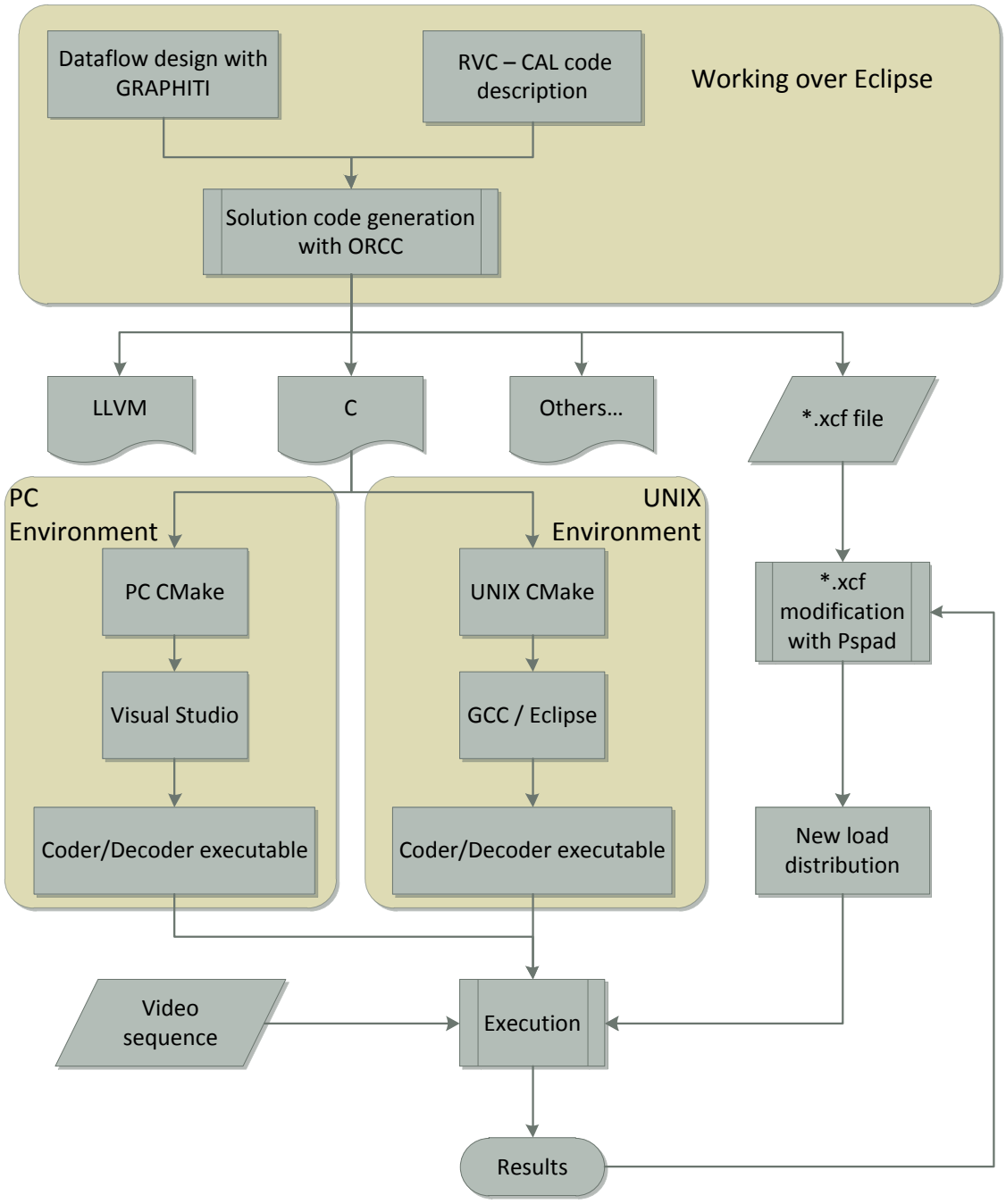

Fig. 21: General working environment scheme.

4.5. Extra tools and packages

This subsection is dedicated to the extra tools and packages needed to start up the working environment. In ANNEX I the detailed procedures to install all the system are described.

Table 3: shows the main software tools needed for the two platforms over that practice work here developed has been done. These are a PC-based system and an embedded system, in particular PandaBoard [Cor9].

PC (Windows 7, i7 processor) [Int08]	**PandaBoard (Ubuntu 11.04, ARM9 processor)**
Visual Studio 2008 or more	*Java-JRE and Java-JDK - recommended*
Java-JRE y Java-JDK	*Eclipse Indigo (v3.7 or more)* - recommended*
Eclipse Indigo (v3.7 or more)*	*Graphiti and ORCC** compiler - recommended*
Graphiti and ORCC** compiler	*SVN - recommended*
SVN-Tortoise	CMake
CMake	SDLs

**Needs Java packages to use it.*

***Needs Eclipse IDE to use it.*

Table 3: Summary of packages and tools for a PC-based system and PandaBoard

4.5.1. Eclipse

Eclipse is a free open and distribution develop group centred in the design of a software develop platform. Nowadays there are a lot of versions and specific distributions of Eclipse, in the case of this work it is Eclipse v3.7 Indigo. In addition to developing software from a classical point of view for different languages and platforms, extra tools and packages for extra features are free available. As previously commented i.e. Graphiti and Orcc packages will work over Eclipse [Ecl12].

Other important packages to complete the functionality of Eclipse are: the subversion (SVN) connectors, Xtext edition and documentation, SVN kit implementation or Eclipse for RAP and RCP developers. All these tools are easily installable through the new software manager of Eclipse (see ANNEX I).

4.5.2. Graphiti

The graphical descriptions of the flowcharts for the targeted applications are performed by Graphiti tool. This tool works on the Eclipse platform. As added value, Graphiti is very useful to take a more clear and intuitive point of view of the application as whole, and also about its subcomponents and how data is transmitted between them. The designed network or structure is saved as a *.xdf file [Gra12].

4.5.3. CMake

CMake is an extensible open source system that manages the processes for compiling and building applications over a targeted operating system in independently ways. CMake is designed to be used with native compilation environment. They used a

series of configuration files (CMakeLists.txt) located in the source files directory to generate the application and build the necessary files (i.e. project files for Visual Studio, *makefiles* for PC-based environments and UNIX *makefiles*) depending on the working environment [CMa12].

4.5.4. Visual Studio

Microsoft Visual Studio is an Integrated Development Environment (IDE). There are numerous possibilities offered by Visual Studio in terms of work environments, design of specific applications or specific objectives pursued by the software development being carried out. There is more detailed information in the official Microsoft site [VS12].

4.5.5. PSPad

PSPad is a text editor for Microsoft programmers. PSPad allows working with different development environments, it does not require specific format and can change the character encoding and the format of text files easily [PS12].

4.5.6. SVN/SVN-Tortoise

This is Windows software for monitoring and reviewing versions of the software of applications and its source code. It is not specified for a particular IDE, making it suitable for any group of development tools or workbenches [SVN12].

4.6. Static assignment of the actors over multicore architectures

As yet briefly introduced, Orcc permits to generate a text file (*.xcf type file) in where the actors of a video decoding application will be assigned to each of the available and desired cores of the target platform.

The working interface of Orcc includes the details to generate this file and the developer can perform a first assignment for all actors that implement the application. The details about how to do this assignment are available in ANNEX I subsection 1.1.12.

In the sense that applications are designed over MPEG RVC environment, the possible subsequent optimization processes are facilitated. In the sense of this study, current working methods able us to improve the obtained results by different scheduling topologies and performing a static allocation of the actors. The scheduling topologies are commented in chapter 5.

All actors that integrate a video decoding application will be mapped into a list that will assign them to each available execution core. This list is automatically generated by the compiler Orcc and can be changed later to obtain decoder executable file. The fact to use this assignment is optional and do not need to perform any change with the executable file.

Fig. 22 shows an excerpt of the allocation list, which display all available execution cores in the processor (*partition id*). For each partition block the programme will assign a number of listed actors. Each reference of the actors will correspond to an *instance* listed in each of the available partitions.

```
<Configuration>
	<Partitioning>
		<Partition id="0">
			<Instance id="source"/>
			<Instance id="decoder_texture_Y "/>
			<Instance id="decoder_motion_Y "/>
		</Partition>

		<Partition id="1">
			<Instance id="display"/>
			<Instance id="merger"/>
			<Instance id="parser"/>
			<Instance id="decoder_texture_U "/>
			<Instance id="decoder_motion_U "/>
			<Instance id="decoder_texture_V "/>
			<Instance id="decoder_motion_V "/>
		</Partition>
	</Partitioning>
</Configuration>
```

Fig. 22: Example of a list file extract.

This file list can be also migrated from one platform to another without compatibility issues, and can run the application directly with the new scheduling guidelines. In this way, applications developed can be tested on different platforms having the same assignment performance.

Chapter 5

Dataflow and scheduling, policies and methodologies

In this section, the working principles with Dataflow Process Networks (DPN) are presented. This chapter will comment most important details about dataflow management over RVC applications, also this section will take in account the usage of these technologies for improve the execution results over multicore platforms. Section 5.1 introduces the base concepts about DPN and the main concepts to profit when working since RVC-CAL based applications. In section 5.2 the different scheduling levels are commented and also main scheduling policies. Finally, section 5.3 introduces the concepts about scheduling management to apply it over multicore architectures.

5.1 From RVC specifications of decoder to Dataflow Process Networks

In the present work DPNs have an important role, they are used to produce efficient models to improve and program applications that target a concurrent execution. First benefits allowed by DPNs are that they can be implemented for a wide range of platforms without using synchronization primitives as mutex or semaphores.

Along this study DPN principles will be used to develop a *Canonical Representation* (CR) of RVC-CAL actors, optimized for the execution of RVC specifications. The CR is referred to a representation that follows DPN models in the form of firing rules and firing functions. This allows us to analyse the RVC-CAL actor execution and also an easier implementation of them in networks.

This CR of an actor requires identify the information that make up the RVC-CAL definition. In this sense will be needed to define an RVC-CAL actor with *m* inputs and *n* output as a set of data described below:

- An *identifier* of the actor
- A set of *m input ports* that contains S^m
- A set of *n output ports* that contains S^n
- A set of *parameters* Ψ contains $\psi_i \, \epsilon \, \psi$ values.
- A set of *p state variables* containing on sequence $V^p = [v_1, \dots v_p]$ where v_1 is the value of the i^{th} state variables.
- A set of *q initial value* V_0^q for state variables, possibly related to ψ_i
- A set of *functions* and *procedures*
- A set of $A_i \epsilon \, A$ *actions* where A_L contains A is the set of actions that contains a *label.*
- A partially and non-reflexive ordered relation (<) that sets the *priority* between actions
- An optional deterministic *Finite State Machine* (FSM).

An action $A_i \epsilon \, A$ is composed of:

- An input signature I_i that may refer to all or a subset of the *m input ports* of the actor
- An output signature O_i that may refer to all or a subset of the *n input ports* of the actor
- A body ϕ_i that describes the computation processed by the action
- A guard G_i that gives the required conditions to fire an action

Going back to the example used chapter 2 section 2.3, the FU Algo_ZigzagOrAlternateHorizontalVertical_8x8, presents two inputs (*AC_PRED_DIR* and *QFS_AC*) that may contain a sequence S^2, one output (*PQF_AC*) that may contain S^1, three state variables $V^3 = [buf, count, addr]$ with V_0^1 = 1, a set of actions where $[\text{skip}, \text{start}, \text{read\_write}, \text{done}, \dots] \epsilon \, A_L$, two priorities between skip/start and done/read_write, and one FSM. The action *skip* has $I_1 = [QFS\_AC]$, $O_1 = []$, an empty body ϕ_1 and a guard condition G_1 on *count.*

An RVC-CAL actor integrated in the form of actions can be considered as a particular case of a DPN. Each action is a functional process in the sense of Kahn Process Network [Kah74]. The study of it exceeds the objectives of the current study. As a brief approaching to it, Kahn introduces a denotational semantic to formalize the behaviour of concurrent functional process. It delimits a specific environment where series of deterministic processes communicate asynchronously by passing messages though unidirectional FIFO channels with unbounded capacity.

5.2 Dataflow process networks and their scheduling concepts

Management of dataflow programming allows us to benefit for more extended applications over multicore architectures. Dataflow Process Network (DPN) is a Model of Computation (MoC) [EA95], described as a series of processes called actors. Actors are connected by FIFO channels called data-FIFO (see Fig. 23). The communication between actors is a stream of data composed of a list of tokens. These tokens will control the execution of the actions described on the actors.

The action firings are related with a mapping of input tokens to output tokens applied repeatedly and sequentially in one or more data streams. To produce a firing action the input ports of the actors must have available needed data. Also the inner guard conditions of the actors must be complied. And this execution can be controlled by a list of priorities.

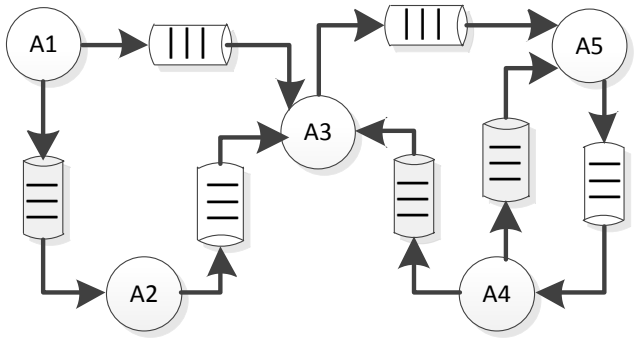

Fig. 23: An example of Dataflow Process Network with five actors

At this point, first scheduling level corresponds to the management of the different actions that will be executed when an action is fired. This management will take into account the functionality of the actions and their inner guard conditions. This scheduling level is called *action scheduler*. In addition, it manages actions at the level of the independent execution conditions of the actors.

A set of actors implements all the functionality of the application. This set will be listed and associated to an execution thread which runs in one core of the processor. At all time an actor is being executed, without non executing times on the processor. This method is the same that has been used by Von Platen in [Pla10], in which tests the benefits of using a user-level scheduler instead of use threads.

The conditions that will control the execution of any action for all actors will be the action firing conditions and the data request produced by others. The list of execution management of all of the actors is called *actor scheduler*. There are two strategies that will control this scheduling level: Round-Robin and Data-driven/Data-Demand (see Fig 24).

- *Round-Robin scheduling strategy* establishes the execution of the actors according to their firing conditions. Current actor execution will continue until firing conditions of next actor meet.

- *Data-driven/Data-Demand strategy* manages the execution of the actors based on their input/output FIFOs data demand. This strategy is based on data-driven and demand driven principles [Par95]. If some actor requires input data, predecessor actor will be called to execution. By the other side if the output FIFO of some actor is complete, successor actor also will be called to execution.

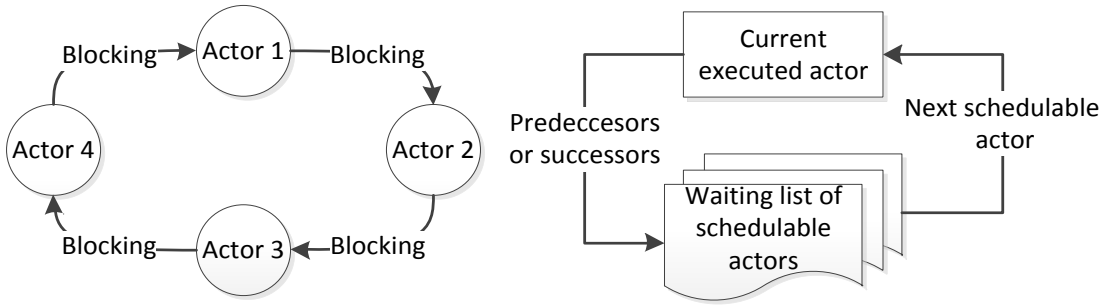

Fig 24: Round-robin (left) and Data-driven/Data-Demand (right) scheduling strategies examples

DPN models are interesting to be used over planned processors as no actors execution information needs to be saved. Only state variables will be registered, which produces the consequently computational loading reduction.

Next section will be centred on actor scheduler level. This is the most relevant scheduling level according with the objectives of this work.

5.3 Dataflow process networks expansion concepts for working over multicore platforms

If working system is a multicore platform and has more than one core, actors list will be split in order to assign some actors to each core execution thread. The different ways in which this mapping distribution will be performed can retrieve important synergies for the application efficiency. This will be the improvement way developed in following sections. Each actor can be mapped only in one core (see following formula).

$$(Actors\ list\ A\ for\ core\ 0) \cup (actors\ list\ B\ for\ core\ 1) \cup (actors\ list\ M\ for\ core\ n\ ) = \{\emptyset\}$$

Yet introduced Round-Robin and Data-driven/Data-Demand strategies will be employed over these multicore architectures. The list of each actor mapped in each core will be executed according to both strategies.

To implement Data-driven/Data-Demand strategy, scheduling information between each core scheduler is needed. This is due to the possibility that predecessor or successor actors being listed on a different core from actor which starts the request.

Scheduling information shared between cores is flowing on the scheduling-FIFOs. Our case, this information is about next actors to schedule. It is important not to confound this flow of scheduling information with the data-FIFO flow information between actors (see Fig 25).

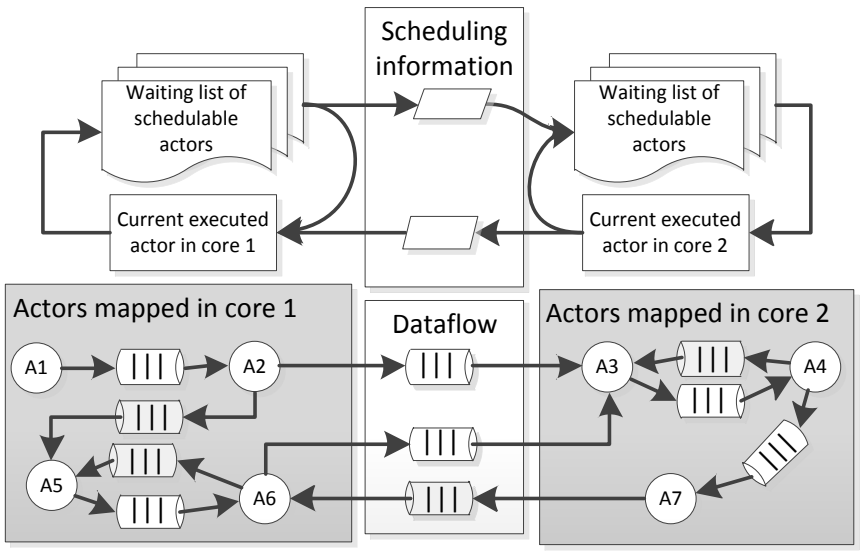

Fig 25: Example diagram of the different communication flows in a dual core architecture

This lock-free inter-core communication is based on Lamport's work [Lam83]. It is carried out without synchronization elements like semaphores. Instead, shared memory is employed on a single producer, single consumer environment, in which read/write pointer manages information flow.

In this sense, in [YCWR11] two kinds of scheduling communication flows are proposed as network topologies: mesh and ring. In mesh topology a bidirectional communication channel is used between each couple of schedulers, allowing direct communication between all of them. In this topology an exponential number of scheduling-FIFOs is needed according with the number of cores under use. In ring topology the number of scheduling-FIFOs is equal to the number of cores and direct communication between some schedulers will not be possible (see Fig 26).

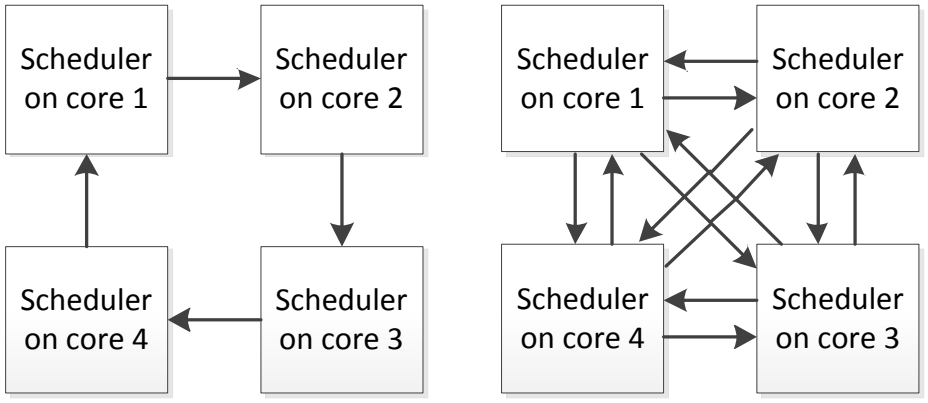

Fig 26: Possible topologies of scheduling flow communications ring (left) and mesh (right)

Chapter 6

Multicore platforms

This chapter is dedicated to the multicore platforms used in the work developed along this master thesis project. These platforms have been used for implement the theory concepts learned and for testing and performing the improving procedures over the video decoders. These platforms have been a multicore PC-based and an embedded system with a multicore processor. This chapter introduces main features about them and in annex III procedures to set up the embedded system are described. This chapter has three subsections, the first one (subsection 6.1) gives main concepts and features about the PC-based platform, the second one (subsection 6.2) discusses about the main features of the embedded system. The embedded platform chosen for this work is based in the OMAP4460 processor and it is called PandaBoard [PB12]; main concepts about it are discussed in subsection 6.3.

6.1. Multicore platform over a PC-based system

First working platform used during the development of this master thesis work, has been a PC-based system. This is a typical commercial laptop with standard features:

- Operating system (OS): Windows 7 Home Premium SP 1 [Win12]
- Processor: Intel Core i7, Q720 @ 1.60GHz [Int12]
- Multicore: 4 cores (8 tasks)
- 64 bits OS type
- Memory: 4GB

All the detailed software needed for work with the yet described environment has been installed in this computer without any compatibility problems. For this platform some decoding applications have been compiled and tested with different scheduling schemes. This situation is the same for the embedded system described in the following section. Knowing the features of both systems and checking the results obtained for them, the differences and conclusions about the improvements performed can be easily obtained.

The fact that the PC-based system has four cores and a maximum of eight tasks allows performing schedulers using more than two cores dedicated for the decoding tasks. But as is described below the PandaBoard has only two cores, then to obtain coherence results between both platforms, only two of the available cores in the PC-based system will be dedicated for decoding tasks.

As previously detailed, the primary OS installed in this platform is Windows, but could be interesting for some developers or environments to work over other OS, i.e. LINUX. Our case, this fact takes special importance, because as follow described, the OS that will be installed in the embedded system is a LINUX version. Also, last software versions for working with LLVM software are specifically provided for LINUX systems.

6.1.1. Virtual machine for LINUX Operating System

As most of the available OS, Windows permits to install a software package for the management of Virtual Machines (VM) for the execution of other operating systems or versions of them. This software will allow us to work with a different OS than the original of the computer.

The resources of the computer will be shared between both OS running at the same time. Typically all physical and virtual resources can be shared, as USB connectors, CD/DVD units, Ethernet network devices or Hard Disk space. Specifically the VM manager here used is VMware Workstation version 7.0.0 [VMw12]. And the OS running over it is a LINUX Ubuntu OS version 10.04 LTS with 32 bits, i686. The resources conceded of the platform are:

- Two processors
- 2 GB of RAM memory
- 50 GB as virtual Hard Disk

Our case, a VM has been installed over the PC-based computer. Over this VM the entire described working environment has been installed. And also the LLVM utilities described in the annex III. The decision of install the LLVM environment over the VM is due to the higher compatibility between this environment and LINUX based OS than between it and other OS.

In the sense of scheduling policies and actor's thread execution assignment over multicore platforms, if it is desired, the VM is limited because not all the available cores in the platform may be ceded to the VM. This must be taken in account before start working with a VM, looking at the maximum of transferable cores to the VM.

6.2. PandaBoard embedded system

The PandaBoard is an embedded system based in the series of processors OMAP44X0 of Texas Instruments [TI12]. Mainlines and features of the processor are described in the next subsection. This board consists in an integrated board with all the technical resources needed for set up a wide range of applications. Our case is the ideal platform to start working with video decoders and to perform an approaching to the multicore experimentation and scheduling, since the OMAP4460 is a dual core processor.

All the installation procedures are described in annex II. The OS installed over the PandaBoard for the work developed here is Ubuntu Release 11.10 Oneiric, Kernel Linux 3.0.0-1207-omap4, GNOME 3.2.1. Following

Table 4 describes the main features of PandaBoard:

GENE-RAL	Dual-core ARM Cortex-A9 MPCore with Symmetric Multiprocessing (SMP) at 1.2 GHz each.	Full HD (1080p) multi-standard video encoding/decoding SGX540 graphics core supporting all major API's including	OpenGL® ES v2.0, OpenGL ES v1.1, OpenVG v1.1 and EGL v1.3
DISP-LAY	HDMI v1.3 Connector (Type A) to drive HD displays	DVI-D Connector (can drive a 2nd display, simultaneous display;	LCD expansion header
MEMO-RY	1 GB low power DDR2 RAM	Full size SD/MMC card cage with support for High-Speed & High-Capacity SD cards	
AUDIO	3.5" Audio in/out	HDMI Audio out	Stereo audio input support

CONN-ECTI-VITY	Onboard 10/100 Ethernet	802.11 b/g/n WiFi	*Bluetooth*® v2.1 + EDR
EXPA-NSION	1x USB 2.0 High- 2x USB 2.0 High-Speed host ports	General purpose expansion header (I2C, GPMC, USB, MMC, DSS, ETM)	Camera expansion header
DEBUG	JTAG	UART/RS-232	Sysboot switch available on board

Table 4: General features of the PandaBoard.

The Fig 27 shows the aspect of the PandaBoard with all the peripherals. As can see the OMAP is located closed to the center of the board and all available connectors in the boundaries. JTAG connector is placed between the OMAP and the expansion header.

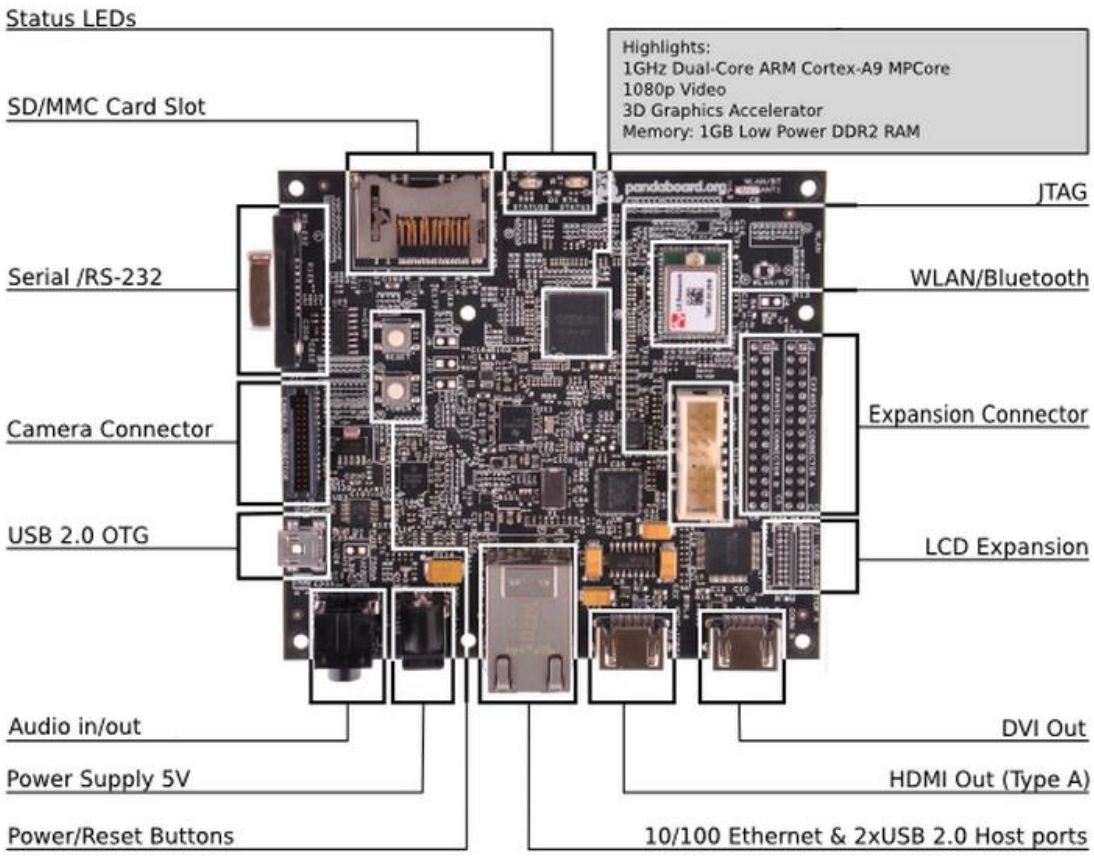

Fig 27: Image of the PandaBoard and peripherals.

6.3. The OMAP4460 Processor

In this subsection principal features of the OMAP4460 are given. It is recommended to read the manufacturer information for a detailed report about the processor.

The OMAP4460 is a SoC (System on Chip) multicore processor based on the OMAP4 architecture of the OMAP44XX processors series of Texas Instruments. It is one of the latest commercial processors offered by Texas. His main features are the multicore architecture, integrated by two ARM Cortex A9, a graphics accelerator, an Image Signal Processor (ISP), and a 1GB RAM memory.

The memory is placed just over the processor; this disposition has been already viewed in other processors of Texas as the OMAP3530 [TI09], for easy understand this concept see Fig 28. The OMAP3530 was used in the BeagleBoard, this board can be considered as the predecessor of the PandaBoard.

Fig 28: Encapsulation and POP type structure of the OMAP35XX processor.

As normally, features exposed for the PandaBoard are based on the capabilities and peripherals offered by the OMAP44XX processor. However, features of the board will be imposed by the features of the processor. In this sense Fig 29 shows the general diagram of the OMAP44XX processors, as can see the features of the board appears as blocks and expansions of the processor.

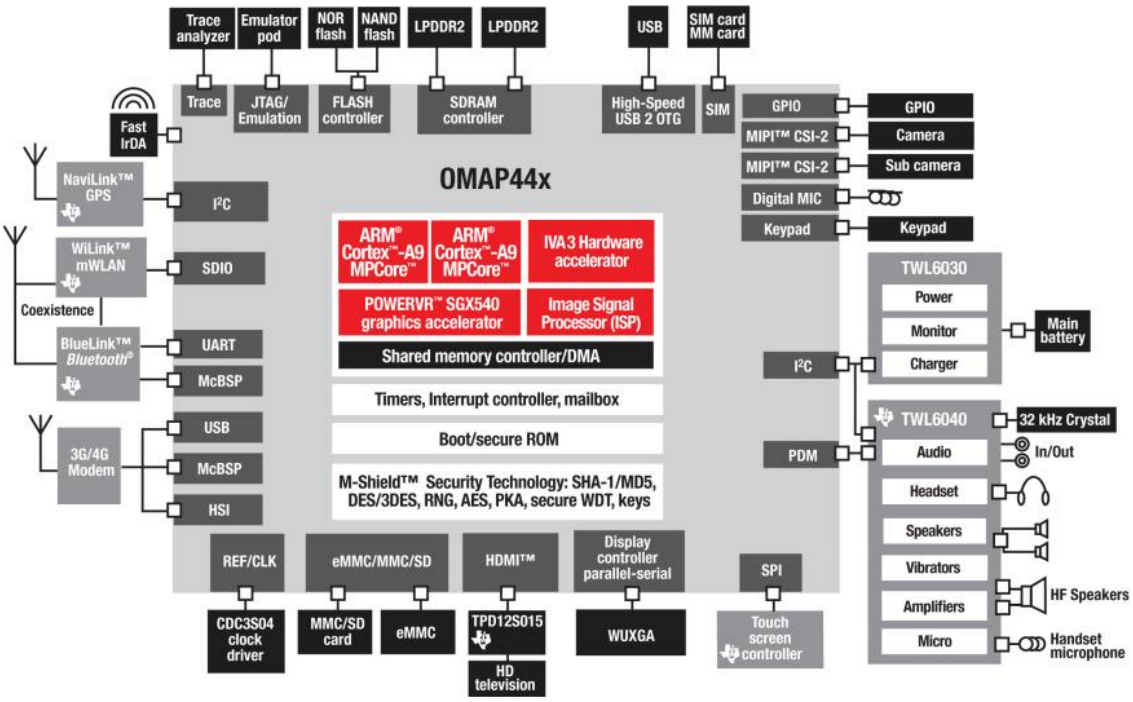

Fig 29: General diagram for the OMAP44XX processors series.

Chapter 7

Experiments and results

This chapter is dedicated to the experiments achieved with the entire working environment and following tests and results obtained for the different procedures, methodologies and video decoders developed or compiled from the already available code. The two first subsections discusses about the improving methodologies carried out over three decoding applications, in the sense of improve the execution of them by modifying the assignment of the actors to the different available cores of the platforms. Subsequently subsection 7.3 details the test and results performed and obtained for these video decoding applications based on the MPEG RVC standard.

Then last subsection 7.4, verse about the management and carrying out of the use of LLVM structures. This last point corresponds to the latest performance of the whole work of this master thesis and is a begging of the use of LLVM more than results of its use.

7.1. Scheduling methodology

The following describes the performing procedures to optimize the static scheduling of actors over decoding video applications. Fig. 30 shows the diagram that exemplifies the process carried out in this environment.

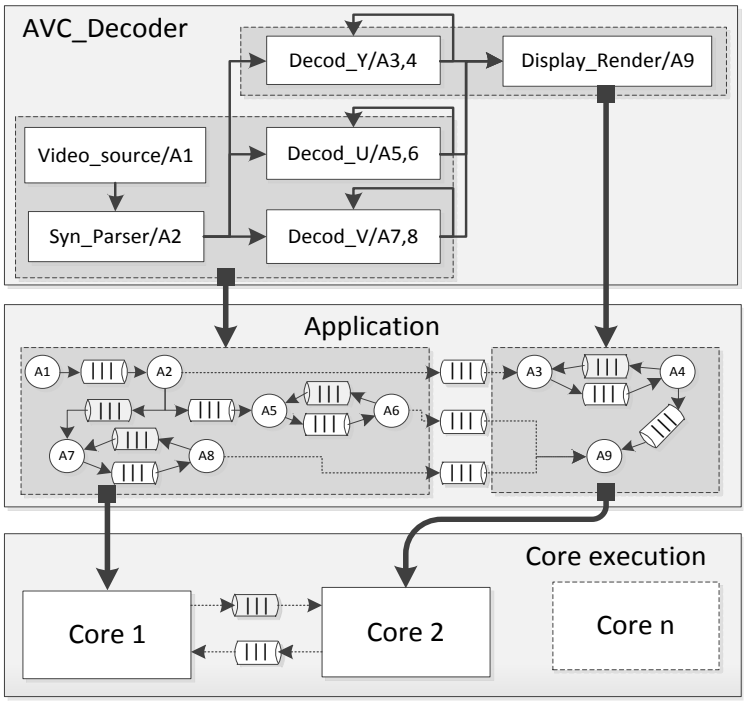

Fig. 30: General working procedure diagram.

It starts with a decoding application description. It is integrated by different sub-tasks (see top of the Fig. 30), which are implemented by certain number of actors (see middle Fig. 30). These actors are listed on the scheduler and their execution is assigned to each of the available cores (see bottom of the Fig. 30).

It is also important to consider the dataflow between actors to obtain best results. It is recommended to use as reference the dataflow diagrams of each application as described on their corresponding RVC-CAL description.

Depending on the number of cores available to run the application, it will need to split the application in one way or another, always trying to distribute the weight more evenly between the available cores. But also is important to take into account that the partition procedure will produce a dataflow communication between actors mapped in different cores. This flow must be reduced to, trying to allocate logical groups of actors into same cores.

In this case the system is working over dual-core systems (PC-based and an embedded system). Each actor of the application has been allocated to either core. In platforms with more than two cores, scheduling task will be more complex and the division functional groups.

Any case, it is recommended to start taking in account the architecture of the application and subsequently to prepare a fine adjustment with lighter duties like video input or display render actors.

All decoding applications will contain certain sub functional parts like video source, parser, luminance and chrominance (U and V decoding) decoders and video render. Parser functionality remains as sequential block, and it will not be able for use as a key element in the scheduling improving tasks. On the other side Y, U and V decoding functionalities are implemented by many actors and involve independent blocks. These are key elements in the scheduling improving works.

Normally chrominance decoding tasks could be considered as medium weight tasks, and also the parser management actor. But the parser complexity could be increased in some newer developed decoders (see subsection 7.2.3). Luminance decoding task will be the most weightily in terms of computational loading.

7.2. Improving procedures over three video decoding applications

This section presents the procedures performed over three decoding applications: MPEG-4 Part2 SP (*Simple Profile*), MPEG-4 Part 10 CBP (*Constrained Baseline Profile*) and MPEG-4 Part 10 PHP (*Progressive High Profile*). This has been done taking in account the methodology yet presented. The following describes the work done over the static assignment of the actors for available cores, in order to obtain best load distribution. For MPEG-4 Part 10 PHP decoder has also been included the use of ring and mesh topologies yet commented in subsection 5.3. These topologies are included automatically by Orcc while generating the application code.

7.2.1. MPEG-4 Part 2 SP decoder application

First, a MPEG-4 Part 2 video decoding application has been generated. Fig. 31 shows partitioning diagram of the application made in terms of groups of actors that implement the functionality of the decoder.

Best results are obtained for a distribution as shown in Fig. 31, assigning actors that implement the luminance decoding (Decod_Y) to a single core and leaving the rest in the other core. This is because the actors that perform the decoding of the luminance are the heaviest in computational load terms. Performing this operation, the

improvement obtained respect to a single core distribution rounds a gain factor between 70% and 85% in terms of increasing of decoded (see annex V) frames per second (FPS).

The implementation of display and video source functions are the ones that require less computational power. The effects of involving actors that implement these features to either core are usually insignificant and its impact on the gain factor is normally between ± 0.5% and ± 2%. This effect is similar for the other decoding applications with those has worked and described below.

In the results subsection (see subsection 7.3), Table 5 shows the results obtained from tests of decoding using this application with the previously described scheduling.

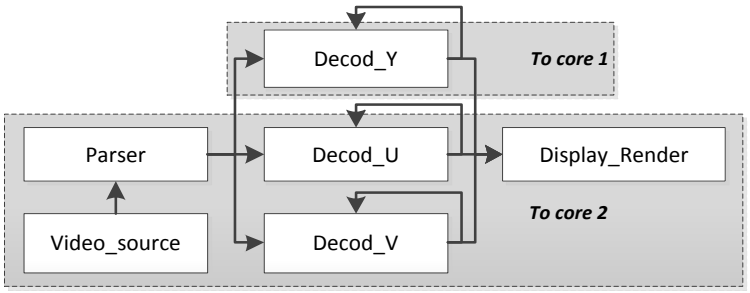

Fig. 31: Simplified MPEG-4 Part 2 decoder diagram.

7.2.2. MPEG-4 Part10 CBP decoder application

The second decoding application, which has been tested, is a MPEG-4 Part 10 CBP decoder. This decoder implements the functionality of an AVC (*Advanced Video Coding* or H.264) decoder in his simplest profile. Normally, actors that implement new decoder functionalities have in this case, a significantly higher complexity than in MPEG-4 Part 2 decoder case. However, luminance decoding task stills as the most costly in terms of computational load.

As can see in Fig. 32, a new block is implemented, it is the *CavcExpand*, and also the complexity of inner tasks and the actors that implement the other functionalities have been increased. Normally this increase is equitable between Y, U and V decoding sub-blocks.

The distribution of the actors is very similarly organized to the previous case, i.e. assigning luminance decoding to a core and the other actors and features to the other

one. This planning provides better gain factor, between 90% and 100% in terms of FPS (see the results in Table 6).

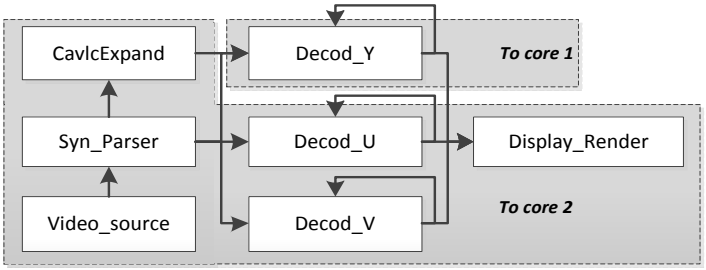

Fig. 32: Simplified MPEG-4 Part 10 CBP decoder diagram.

7.2.3. MPEG-4 Part 10 PHP decoder application

Last case of study is a MPEG-4 Part 10 PHP video decoding application. In this case should keep in mind that the actors that implement the functionality of parser decoder will have a higher complexity, which will lead to review the planning done in the previous cases.

It must make up the complexity and the increasing needed for processing computational power demanded by the parser, exporting other actors to the core used previously for the execution of luminance decoding.

Fig. 33 shows the obtained optimal scheduling. It is assigned, along with the decoding of luminance, the actors that implement the functionality of *CavlcExpand*. This compensates the increase involved in carrying out the parsing.

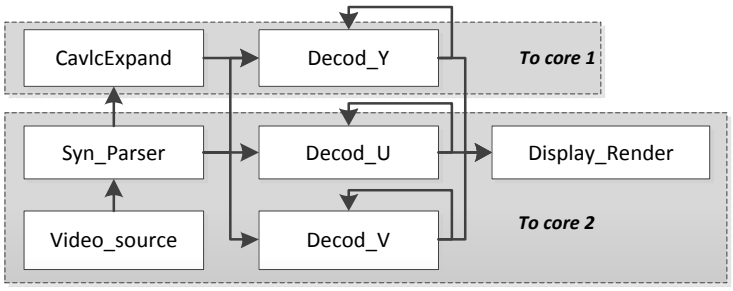

Fig. 33: Simplified MPEG-4 Part10 PHP decoder diagram

In addition, for this application smart scheduling topologies Ring and Mesh have been used, based on communication between actors schedulers. The results of decoding are included in the results of the next subsection on table 6.

7.3. Tests and results with RVC decoders over multicore systems

This section describes the results obtained working with the three decoding applications previously described. The three decoding applications have been tested over the two platforms described (PC-based and PandaBoard) for different static assignments of the actors to each available core and different video test sequences. In following tables (Table 5 and Table 6) the most interesting results are presented. The rest of detailed results and test performed are available in ANNEX V.

Following tables show the number of decoded frames by second that the application is able to decode according to the application complexity, the scheduling map performance and the number of cores, which in our case are two.

All the scheduling strategies have been tested on dataflow descriptions of MPEG-4 Simple Profile and MPEG-4 Part 10 with different sized video sequences. These decoders have been benchmarked on an Intel i7 with eight cores at 1.6GHz (only two cores was used for video decoding task), and on an ARM Cortex-A9 dual core OMAP4 at 1.2GHz (over the embedded system).

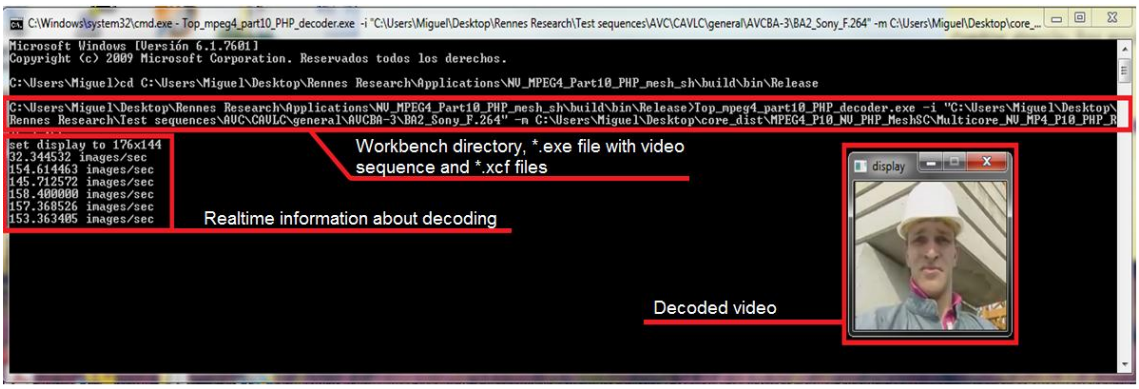

Fig. 34: Video decoder MPEG-4 Part 10 running over a PC-based platform.

Fig. 35: Video decoder MPEG-4 Part 10 running over a PandaBoard.

Main objective is to obtain a gain factor equal to the number of employed cores. Our case optimal *speedup* factor must be 2, according with the number of cores available on our system. The *Speedup* factor helps to know how the new scheduling map applied improves the functionality.

The testing video sequences are: for MPEG-4 Part 2, hit001 and jvc009 (CIF and QCIF respectively) from ISO/IEC 14496-4:2004 [ISO04]. For MPEG-4 Part 10 CBP decoder the sequences are BA2_Sony_F (QCIF) and HCBP1_HHI_A (CIF), and for MPEG-4 Part 10 PHP LS_SVA_D (QCIF) and HCBP1_HHI_A (CIF) [ISO04]. All of them are described in the standard already indicated.

One of the first conclusions obtained when carrying out the first tests of load distribution is that the decoding of the luminance (Y decoder) is far more expensive than the other tasks. Hence, it was decided to award the set of actors that implement decoding of luminance to one core, leaving the other actors of the application on the other.

MPEG-4 PART 2	**Sequence**	**Single core**	**Multicore**	**Speedup**
PC-BASED	QCIF	279.32	476.27	**1.7**
	CIF	63.77	116.55	**1.82**
EMBEDDED SYSTEM	QCIF	57.24	97.87	**1.71**
	CIF	13.14	23.48	**1.78**

Table 5: Most relevant decoding results obtained for MPEG-4 part 2.

Results show that for some cases the speedup factor obtained is higher over the embedded system than PC-based platform. This is caused by the different use of memory access and distribution between platforms and the more complex operating system scheduler of the PC-based platform.

The decoded frames rate obtained with the embedded system is lower respect to PC-based system because of the less system power skills. This situation can be solved through an optimization of the decoder application.

MPEG-4 PART 10	Codec	Sequence	Single core	Multicore	Speedup
PC-BASED	CBP	QCIF	78.76	149.86	**1.9**
		CIF	16.56	33.16	**2**
	PHP *Mesh*	QCIF	104.32	194.51	**1.86**
		CIF	23.14	43.14	**1.86**
	PHP *Ring*	QCIF	86.93	163.74	**1.88**
		CIF	15.49	28.53	**1.84**
EMBEDDED SYSTEM	CBP	QCIF	11.96	22.72	**1.89**
		CIF	2.2	4.74	**2.15**
	PHP *Mesh*	QCIF	20.32	39.41	**1.93**
		CIF	2.87	6.05	**2.1**
	PHP *Ring*	QCIF	16.71	33.35	**1.99**
		CIF	3.08	6.26	**2.03**

Table 6: Most relevant decoding results obtained for MPEG-4 part 10.

In Table 6 can see that the speedup factor is for all cases higher than 1.84 and for most of them closer to 2. And that for MPEG-4 Part 10 PHP decoder, mesh topology returns an improvement between 25% and 40% respect to use ring topology.

Also in Table 6 *speedup* factors upper than 2 can be seen. When an application is executed over single core architecture all actors are located in only one core and their execution is sequentially fixed. All time an actor is being executed but it is probably that some actors are waiting for needed information of other ones (see chapter 5 subsection 5.2). When the architecture change to a multicore structure and two execution actor loops are available this undesired effects are reduced.

7.4. Decoding application for multicore usage with LLVM

In this last part of the work, the desired objective is to view the final goals of the use of LLVM based decoders and their architecture. As this issue is an extension of the original objectives for this work, the sense of this last performance is to approach to the

opportunities offered by LLVM and to facilitate future works that will continue the developed work in this Master Thesis.

Two considerations will guide this part of the work. The first one is the goal of focus our work in the usage of video decoders for multicore platforms, and the other is to approach our work to the best usage of the Orcc compiler and his benefits.

In the last exposed sense, as was exposed, Orcc allows to generate video decoding application solutions in different languages. The choice of these solutions will depend on the future usage of the application by the developer. But it must take in account that there are structural differences between these languages. All the allowed languages are: Java, C, C++, LLVM (through AOT) and JADE (through LLVM). The main difference for us is that except Java and JADE, the rest of languages works with instances instead of the classes.

Then, the goal will be to work with languages based in classes; this is due to the most efficient use of the code and the easiest reuse of this code of the applications for different video decoders. For example, in a video decoding application will be needed an Inverse Discrete Cosine Transform (IDCT) functionality. If the chosen language is C, it will be necessary to have three IDCT different instances for each component of video (IDCT_U, IDCT_V and IDCT_Y), and this fact is extensible to the set of the internal functionality of the decoder. By the other side if the chosen code is Java, it only will be necessary to declare the class “IDCT” and this single class will be able for three components. It will be only necessary to change the variable values of the different particularly cases, but not the instance or use for a specific functionality.

7.4.1. New procedure proposed

The working goal for this new considerations will be to obtain a video decoder from an XDF structure previously done, but generating it with LLVM code extension (*.ll). As this language is different of the well known languages, the procedure will be to modify the back-end files of the C code for the application, and after that, translate the C code to LLVM code with the Clang [Cla12] tool. Besides the objective is to regenerate the C main file of the decoder, and include in the code of this file the calls to every integrating instances of the actors needed in the application. This calls must take in account the static assignation of the actors for each core performed in Orcc (in the mapping configuration and without need to manually modify the *.xcf file).

Then, the developed work in this part will consist of:

- Modify the back-ends of a video decoding application for obtaining the main file of the application as C code solution by taking in account the static assignation of the actors for each core. The procedure will be to create every needed execution threads for each core, assign them, and place the calls into their corresponding execution thread. The back-ends can be modified in Eclipse by using the network.stg files for all developed applications yet developed.
- Translate the obtained C code to *.ll files with Clang.
- Compile the LLVM application files and obtain the executable of the decoder.
- Check that the new decoder works correctly.

7.4.2. Modifying the back-ends to generate a new version of a video decoder

Here, the modified code is presented. The first images contain the extracts of the code relative to the netowork.stg file of the back-end for the decoding application. In this case the chosen decoder is a MPEG-4 part 2 called MVG, which main file is Top_MVG.c.

As reminder, this first images presents the code used for generating the solution files in C code for the decoder. Changing this code, the modification will appears once the Orcc is executed for obtaining the solution files.

The following Fig. 36 includes an extract of the code of the network.stg file. In this case, the code of the sub-functions that will be called for generating the code of any functionality, is through this functions as the back-end will generate the code for the solution files. In particular, this extract shows the code for generate the initializations of the instances needed in the decoding application (*printInitialize* and *printInitializes*), next two declarations have been newly created for the generation of the code relative to the execution threads and the calls to the instances, associating these to each thread (*print_calls* and *create_nthread*).

```
////////////////////////////////////////////////////////////////////////
// print calls to initialize()

printInitialize(instance) ::= <<
<if(instance.actor)>
<instance.name>_initialize(<instance.actor.inputs: { port | <if
(instance.incomingPortMap.(port))><instance.incomingPortMap.(port).fifoId><else>-
1<endif>}; separator=",">);<
endif>
```

```
>>

printInitializes(instances) ::= <<
void initialize_instances(){
	<instances: {instance|<printInitialize(instance)>}>
}
>>

////////////////////////////////////////////////////////////////////////////
// print scheduled calls into while loops and create the associated threads

create_nthread(id, instances) ::= <<
thread_create(threads[<id>], loop_instances_<id>, si, threads_id[<id>]);
set_thread_affinity(cpuset, <id>, threads[<id>]);
>>

print_calls(id, instances) ::= <<
static void *loop_instances_<id>(void *data){
	struct schedinfo_s si;

	while(1){
		<instances : {instance | <instance.name>_scheduler(&si);};
separator="\n">
	}
}   >>
```

Fig. 36: Excerption of the modified code of the *network.stg* file.

In Fig. 37 the usage of the already described functions are included. These calls (i.e. <printInitializes(network.instances)>) to the functions are also inserted in the *network.stg* file. It is in this place (the bottom of the file) where will appear the final code. It can be easily seen contrasting Fig. 37 and Fig. 38.

```
////////////////////////////////////////////////////////////////////////////
// Initializer and launcher

<printInitializes(network.instances)>

////////////////////////////////////////////////////////////////////////////
//Top

<options.mapping.keys : {processorId | <print_calls(id=processorId,
instances=options.mapping.(processorId))>}; separator="\n">

////////////////////////////////////////////////////////////////////////////
// Main function
int main(int argc, char *argv[]) {
	init_orcc(argc, argv);

    struct schedinfo_s *si;
	cpu_set_t cpuset;
	thread_struct threads[MAX_THREAD_NB];
	thread_id_struct threads_id[MAX_THREAD_NB];

	initialize_instances();

	<options.mapping.keys : {processorId | <create_nthread(id=processorId,
instances=options.mapping.(processorId))>}; separator="\n">


  printf("End of simulation!<if(!options.useGeneticAlgorithm)> Press a key to
continue<endif>\n");
```

```
  <if(!options.useGeneticAlgorithm)>wait_for_key();<endif>

  return 0;
}      >>
```

Fig. 37: Second excerption of the code of the *network.stg* file.

Fig. 38 shows an extract of the generated code in C language for the *Top_MVG.c file*. As can see, the *main* function has been reduced in order to simplify as much as possible the code and the complexity of the application. In this *main* function only takes place the creation of the execution threads and their corresponding association to each available core. Associated to each thread, it is included the corresponding function, this function consists on a *while loop* with the calls to the instances of each actor involved in the decoder.

```
////////////////////////////////////////////////////////////////////////////
//Top

static void *loop_instances_1(void *data){
        struct schedinfo_s si;

        while(1){
                source_scheduler(&si);
                display_scheduler(&si);
                Merger_scheduler(&si);
                decoder_serialize_scheduler(&si);
                decoder_parser_parseheaders_scheduler(&si);
                decoder_parser_mvseq_scheduler(&si);
                decoder_parser_blkexp_scheduler(&si);
                decoder_parser_mvrecon_scheduler(&si);
        }
}
static void *loop_instances_0(void *data){
        struct schedinfo_s si;

        while(1){
                decoder_texture_DCsplit_scheduler(&si);
                decoder_texture_IS_scheduler(&si);
                decoder_texture_IAP_scheduler(&si);
                decoder_texture_IQ_scheduler(&si);
                decoder_texture_idct2d_scheduler(&si);
                decoder_texture_DCReconstruction_addressing_scheduler(&si);
                decoder_texture_DCReconstruction_invpred_scheduler(&si);
                decoder_motion_address_scheduler(&si);
                decoder_motion_buffer_scheduler(&si);
                decoder_motion_interpolation_scheduler(&si);
                decoder_motion_add_scheduler(&si);
        }
}


////////////////////////////////////////////////////////////////////////////
// Main function
```

```
int main(int argc, char *argv[]) {
       init_orcc(argc, argv);

    struct schedinfo_s *si;
       cpu_set_t cpuset;
       thread_struct threads[MAX_THREAD_NB];
       thread_id_struct threads_id[MAX_THREAD_NB];

       initialize_instances();

       thread_create(threads[1], loop_instances_1, si, threads_id[1]);
       set_thread_affinity(cpuset, 1, threads[1]);
       thread_create(threads[0], loop_instances_0, si, threads_id[0]);
       set_thread_affinity(cpuset, 0, threads[0]);


  printf("End of simulation! Press a key to continue\n");
  wait_for_key();

  return 0;
}
```

Fig. 38: Excerption of the generated code for the *Top_MVG.c file.*

7.4.3. Final code translation and test of the decoder

As was commented in last subsection, the objective is to *translate* the C code to LLVM code, compile the application and test it.

Fig. 39 shows the translated LLVM code related with the calls to the instances.

```
; <label>:3                                       ; preds = %0, %3
  call void @decoder_texture_DCsplit_scheduler(%struct.schedinfo_s* %si)
  call void @decoder_texture_IS_scheduler(%struct.schedinfo_s* %si)
  call void @decoder_texture_IAP_scheduler(%struct.schedinfo_s* %si)
  call void @decoder_texture_IQ_scheduler(%struct.schedinfo_s* %si)
  call void @decoder_texture_idct2d_scheduler(%struct.schedinfo_s* %si)
  call void
@decoder_texture_DCReconstruction_addressing_scheduler(%struct.schedinfo_s* %si)
  call void @decoder_texture_DCReconstruction_invpred_scheduler(%struct.schedinfo_s*
%si)
  call void @decoder_motion_address_scheduler(%struct.schedinfo_s* %si)
  call void @decoder_motion_buffer_scheduler(%struct.schedinfo_s* %si)
  call void @decoder_motion_interpolation_scheduler(%struct.schedinfo_s* %si)
  call void @decoder_motion_add_scheduler(%struct.schedinfo_s* %si)
br label %3
```

Fig. 39: Extract of the resulting *.ll *translated* code from the C language solution.

Chapter 8

Conclusions and future works

This last section presents the conclusions obtained along the development of the different parts of this Master Thesis work. Also at the end of the conclusions, the proposal of different future works will allow continuing these working lines and improving of the methodologies here exposed.

At the begging of the work the objectives were introduced. These objectives have been achieved, and also some extra studies about the usage of the current technologies that are taking place nowadays.

The major achievements offered in this work are:

- The study of the new MPEG RVC standard, and their main theory and practice components. Also a guide about the working environment installation and usage has been done and it is available for future works.
- The scheduling policies, and the mapping procedures for the static assignment of the actors that integrate the decoders have been studied, after that, practice techniques have been done in the sense of improve the efficiency of these decoders, obtaining important results and conclusions. It has been proven that the use of these techniques can report important enhancements for the exploitation of the system capabilities.
- All new concepts and techniques already exposed have been migrated to an embedded system. For this platform a set of tests and results are available. These results have been contrasted with the obtained for a PC-based system and the improvements achieved for both.
- Main concepts about LLVM usage have been introduced and offer an ideal starting point for future developments in this sense. The LLVM usage for

video decoding applications hopes to develop dynamic decoders, and at the same time, mix this acknowledgements with the scheduling techniques for develop of a new series of video decoders more efficient and usable.

However, all the developed work has presented a number of difficulties. It should be noted that three different OS have been used for the implementation for all the practice tests. The amount of packages needed for installing the entire working environment produces that some of them can be incompatibles with the chosen OS or that the available versions of them are not yet available.

Also should take into account that most of the components here used are very new or under development by other working groups. This produces that, when the work is under development, new versions can be released and it is important to take them into account, but they will need to be reinstalled. And it is possible that some packages aren't fully tested and can appear bugs.

As future proposals to continue the work here started, it is important to review all the documentation about the topics already introduced and be aware about new packages and improvements in this sense. It is highly possible that some changes have been done over the tools of the working environment. These changes must be taken in account and new research can offer important improvements.

References

[ALR+09] I. Amer, C. Lucarz, G. Roquier, M. Mattavelli, M. Raulet, J.F. Nezan, O. Déforges, “Reconfigurable video coding on multicore”, in *Signal Processing Magazine, IEEE,* vol. 26, no 6, pp 113-123, nov 2009.

[BEJ+09] S. 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, 2009.

[BMS+09] J. Boutellier, V. Martin Gomez, O.Silven, C. Lucarz, and M.Mattaveli, “Multiprocessor scheduling of dataflow models within the Reconfigurable Video Coding framework”, in *Conference on Design and Architectures for Signal and Image Processing (DASIP),* 2009.

[BSR11] J. Boutellier, O. Silvén et M Raulet : Automatic Synthesis of TTA Processor Networks From RVC-CAL Dataow Programs. In Signal Processing Systems, oct. 2011.

[BSR11] J. Boutellier, O. Silvén, M. Raulet, “Automatic synthesis of TTA processor networks from RVC-CAL dataflow programs”. *IEEE Workshop on Signal Processing Systems* (SiPS), 2011, pp 25-30.

[Cla12] Link to Clang web site: http://clang.llvm.org/

[Cor9] Link for information about the Cortex-A9: http://www.arm.com/products/processors/cortex-a/cortex-a9.php

[EA95] E. A. Lee and T. Parks, "Dataflow Process Networks," in Proceedings of the IEEE, 1995, pp. 773-799

[Ecl12] Eclipse web site: http://www.eclipse.org/

[EV06] S. Efftinge et M. Völter: oAW xText : A framework for textual DSLs. In Workshop on Modeling Symposium at Eclipse Summit, 2006.

[Gor11] Machine Virtuelle Universelle Pour Codage Vidéo Reconfigurable, M. Jérôme GORIN, Thèse de doctorat de l'Université Pierre et Marie Curie, 2011.

[Gpa12] GParted web site for Ubuntu. http://doc.ubuntu-fr.org/gparted

[Gra12] Graphiti web site: http://www.eclipse.org/graphiti/

[GWPR11] J. Gorin, M. Wipliez, F. Prêteux, M. Raulet: LLVM-based and scalable MPEG-RVC decoder. *In Journal of Real Time Image Processing 6, 2011.*

[Int08] Intel Core i7 general information web site: http://infomicros.wordpress.com/2008/08/12/intelcorei7/

[Int12] Intel Core i7 general information web site: http://www.intel.com/content/www/us/en/processors/core/core-i7-processor.html

[ISO04] ISO/IEC International standard 14496-4:2004 video Information technology- Coding of audio-visual objects-- Part 4: Conformance testing.

[ISO08a] ISO/IEC 23001-5. *Information technology - MPEG systems technologies - Part 5: Bitstream Syntax Description Language, 2008.*

[ISO08c] ISO/IEC CD 23002-4. *Information technology - MPEG video technologies - Part 4 : Video tool library, 2008.*

[ISO09] ISO/IEC 23001-4. *MPEG systems technologies Part 4: Codec Configuration Representation, 2009.*

[ISO11] ISO/IEC 23001-4, MPEG systems technologies – Part 4: Codec Configuration Representation (2011).

[Java12] Oracle for Java web site: http://www.oracle.com/technetwork/java/javase/downloads/index.html

[Kah74] G. Kahn, The semantics of a simple language for parallel programming, in: J. L. Rosenfeld (Ed.), Information processing, North Holland, Amsterdam, Stockholm, Sweden, 1974, pp. 471–475.

[Lam83] L. Lamport, “Specifying Concurrent Progem Modelues”, ACM Trans. Program. Lang. Syst., vol. 5, pp. 190-222, April 1983

[MAR10] M. Mattavelli, I. Amer, M. Raulet, “The reconfigurable video coding standard [standards in a nutshell]”, *Signal Processing Magazine, IEEE,* vol. 27, no. 3, pp. 159-167, may 2010.

[OP12] Omappedia general information web site: http://omappedia.org/wiki/Main_Page

[Par95] T. M. Parks, Bounded Scheduling of Process Networks, Ph.D. thesis, EECS Department, University of California, Berkeley, 1995.

[PB12] PandaBoard web site: http://pandaboard.org/

[PHH+03] G. Panis, A. Hutter, J. Heuer, H. Hellwagner, H. Kosch, C. Timmerer, S. Devillers et M. Amielh : Bitstream syntax description : a tool for multimedia resource adaptation within MPEG-21. Signal processing. Image communication, 18(8):721747, 2003.

[Pla10] C. von Platen, “Project report: CAL ARM Compiler”, http://www.actors-project.eu/, Jan 2010.

[PS12] PSPad web site: http://www.pspad.com/es/

[SMW07] H. Schwarz, D. Marpe et T. Wiegand : Overview of the scalable video coding extension of the H. 264/AVC standard*. Circuits and Systems for Video Technology, IEEE Transactions on*, 17(9):1103_1120, 2007.

[SNR11] N. Siret, J.F. Nezan et A. Rhatay : Design of a processor optimize for syntax parsing in video decoders. In Design and Architectures for Signal and Image Processing (DASIP), 2011, pages 16. IEEE, 2011.

[SVN12] Tortoise-SVN web site : http://tortoisesvn.tigris.org/

[TI09] OMAP3530 general information, web site of the manufacturer: http://www.ti.com/product/omap3530

[TI12] OMAP4 of Texas Instruments, block diagrams and general information link:http://www.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=12843&contentId=53243

[Ubu11] Ubuntu 11.10 web site: http://releases.ubuntu.com/11.10/

[Ubu12] Ubuntu web site for downloadings: http://www.ubuntu.com/download/ubuntu/download

[VMw12] VMware workstation web site: http://www.vmware.com/products/workstation/overview.html

[VS12] Microsoft – Visual Studio web site: http://www.microsoft.com/visualstudio/es-es

[Win12] Microsoft Windows 7 home premium web site: http://windows.microsoft.com/es-XL/windows/shop/windows-7/home-premium

[YCWR11] E. Yviquel, E. Casseau, M. Wipliez et M. Raulet : E-cient Multicore Scheduling Of Dataflow Process Networks. *In Signal Processing Systems (SIPS), 2011 IEEE Workshop on*, pages 81-86. IEEE, 2011.

ANNEX I

i) Installation procedures

This section describes the procedures to install the working environment over a PC-based platform. Also detailed steps and important considerations about the procedure are given for manage needed tools. According to the work described along the report the objective of this annex is to provide all the tools needed to start working with MPEG RVC-CAL standard for Orcc compiler. It is highly recommended to read all annex before start installation procedure.

1. It is needed to install Visual Studio 2008 or above with all accessories of this developing platform. There is long available software related in Microsoft Corporation web site [VS12].

2. Java environment installation. All packages are available on Oracle's web site. Also user's guides about installation procedures for different OS are available [Java12].
    2.1. Install Java-JRE (Java Runtime Environment)
    2.2. Install Java-JDK (Java Development Kit)

3. Eclipse installation [Ecl12]. Must be installed and running an Eclipse's versions equal or above 3.7-Indigo. Note that Eclipse needs Java to be installed and run in the computer.
    3.1. Once Eclipse is installed first step is to go to the new software management (*Help → Install new software*) and install *Eclipse Platform* and also the following tools: *Eclipse for RCP and RAP Developers*, *Eclipse SDK, Eclipse Platform SDK* y *Xtext SDK.* If the developer considers that some other packages will be needed for his own specific project this is the moment to install it.
    3.2. Also is highly recommended to check for new available updates. This is an automatic procedure. To execute it go to *Help → Check for Updates.*
    3.3. To check the packages also installed in Eclipse and also their respective versions go to *Help → About Eclipse → Installation details*

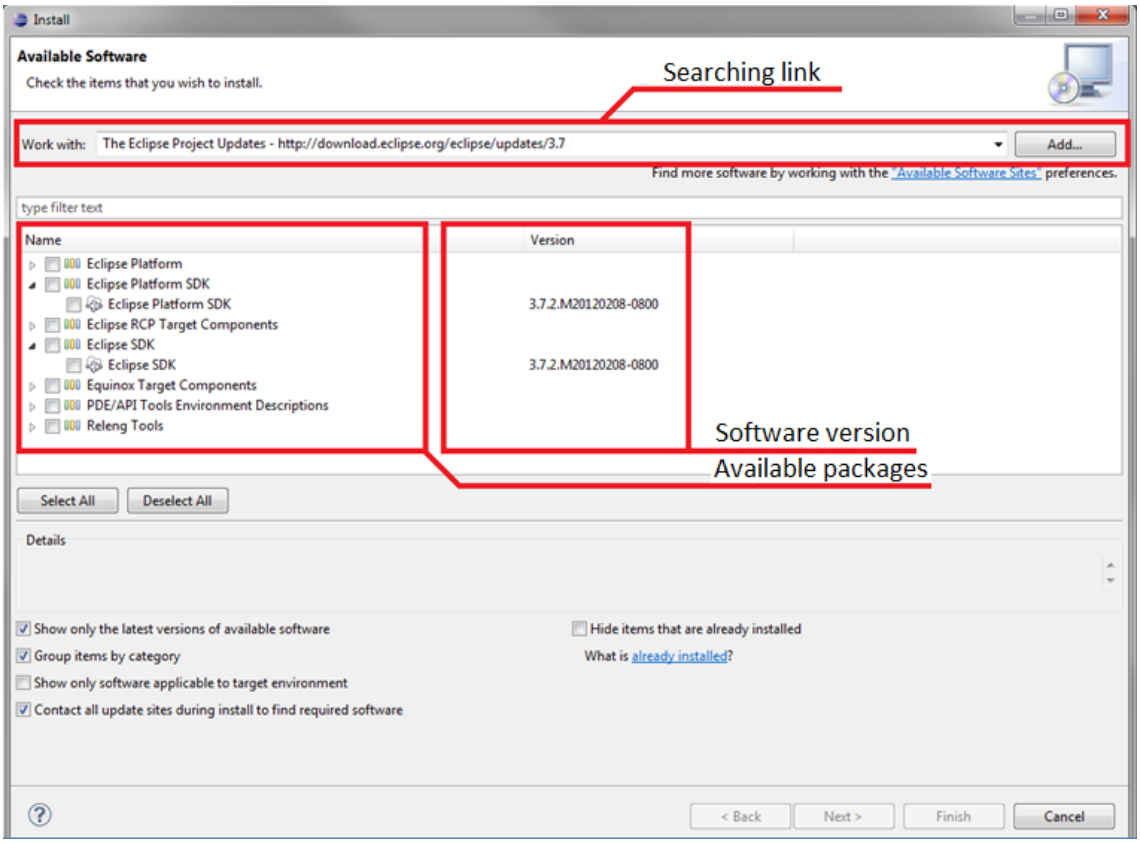

Fig. 40: New software interface window in Eclipse.

4. Graphiti and Orcc installation

   Same as section 3.1, both packages tool for Graphiti and Orcc can be downloaded and installed with the new software manager of Eclipse. Only the download link for the specific tools must be added to the list of searching sites, this can be done in the same window where found tools appears. The Orcc version used in this work can be found in the following link: http://orcc.sourceforge.net/nightly/.

Note: during the installation procedure Eclipse will notify that this software has no official certificate, this commentary must be ignored and continue with the installation.

5. SVN-Subversion (Tortoise SVN) installation. This tool helps to update and download software control versions for the applications used in the current working environment [SVN12].

6. Once Eclipse and SVN are installed, available decoding applications developed by other groups can be downloaded. To do it, go first to the following link http://sourceforge.net/projects/orcc/files/ and download it in a different Eclipse's

directory. Then, in Eclipse, go to the main tab: *Eclipse→Import→General→Workspace* and include the directory before created. At this moment a set of applications will appear, is not necessary to import all, our case only “trunk” group has been installed. Only “System” group is common for all and must be installed any case. Fig. 41 shows an image of the standard window working over Eclipse. In left side of the image there is the list of imported applications, to compile one of them with Orcc, right click and select *Run Configurations* as show in figure. In the right side the graph of the application or the RVC-CAL code of a desired actor will be showed.

6.1. To update applications the procedure is to go to the directory where applications were downloaded, right click on root folder and chose *SVN-Update* option. Then go to Eclipse and refresh desired applications showed at left side of the window, update with F5 or right click and *refresh* option.

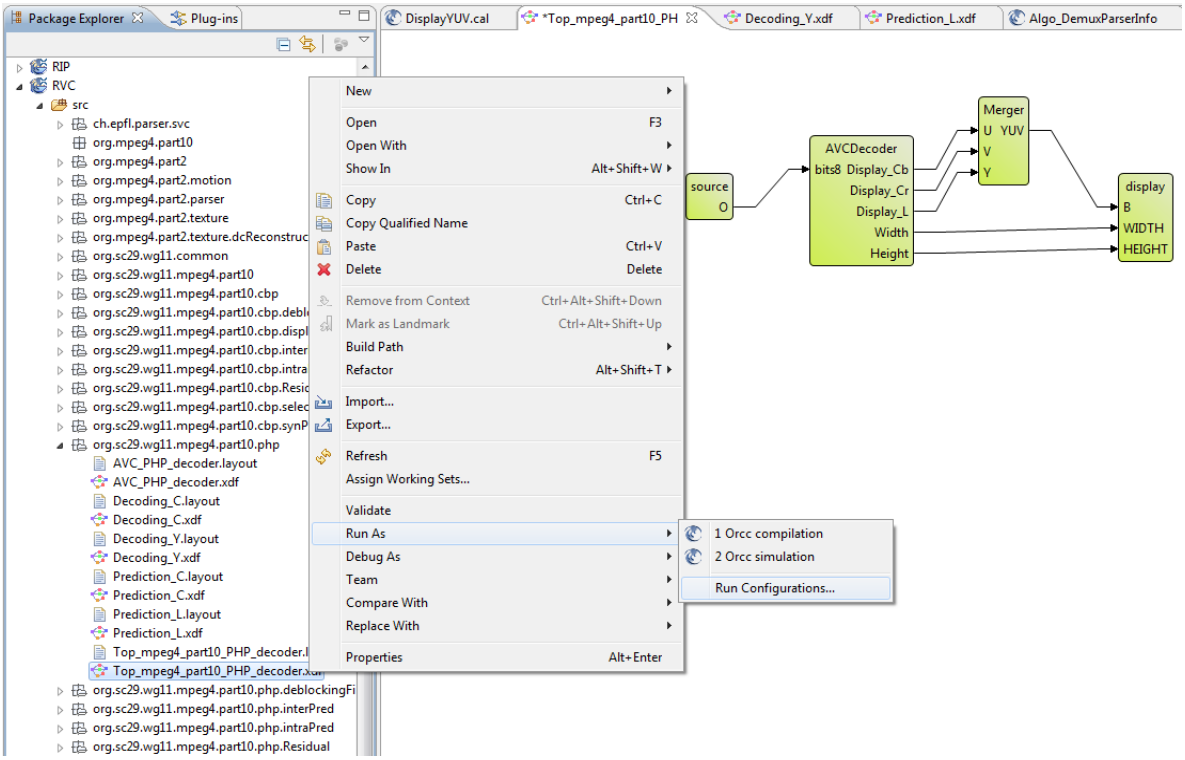

Fig. 41: General view of Eclipse working with the imported RVC applications.

7. CMake compile platform installation. Appropriate installation packages for different OS are available in CMake web site: http://www.cmake.org/. When installing CMake it will ask about the user’s paths, “all users” option must be selected.

7. Orcc libraries compilation. Depending on the version installed, the libraries for Orcc must be manually compiled. The versions used during this Master Thesis work are 1.3.9 for Graphiti and 1.1.0 for Orcc. With these versions when code for an application is generated with Orcc, *.bat file for compile solution files with CMake is automatically generated, and also Orcc will compile automatically the Orcc libraries used.

If the installed versions is previous to the used in this Master Thesis step 9 must be done. Any case we recommend to read next steps to understand all the compilation procedure.

8. *Only if working version for Orcc is previous to 1.1.0.* Go to the directory where Orcc libraries was downloaded as "/Working_directory/orcc_libs/C/Windows" and execute *.bat file with CMake. The source files directory and the target directory for built files must be indicated. This will generate needed files of the libraries; it is possible that these directories must be added as path in the following developing of Orcc-based applications.

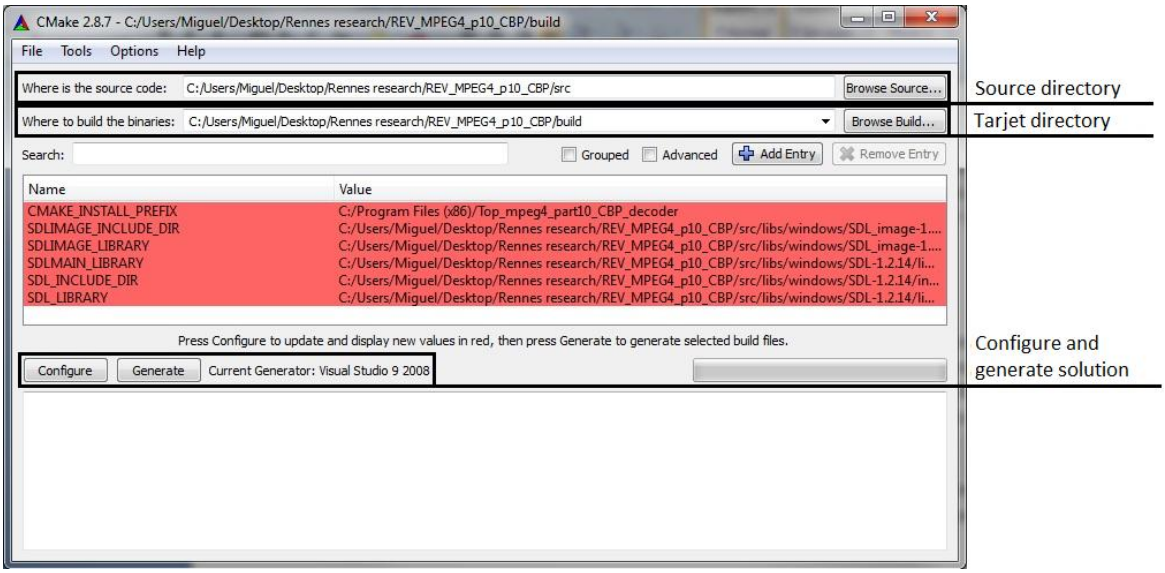

Fig. 42: CMake compilation platform over PC-based computer.

9. The solution for desired decoding application can be done with Orcc (*Run Configurations with Orcc* – see Fig. 41*).* Then the files generated must be compiled with CMake using for this the *.bat file generated by Orcc and placed in the target directory for the application. In CMake source files correspond to the .../srv/ folder

and target files must be placed in …/build/. Current generator for CMake must be the same used before, in this case Visual Studio 2008.

10. CMake will generate the specified files for working, debugging, and compile the application over the desired platform. Then going to Visual Studio 2008 and open the project for the application the final compilation can be done, this will create the corresponding *.exe file. Also at this moment, debugging task can be done over the application. To do it, at least a video file must be linked to the debugger; it is included in *Project Properties→Debug→Arguments* (Fig. 42). At the same time *.xdf file for specific scheduling tasks can be also included as argument for the decoding application.

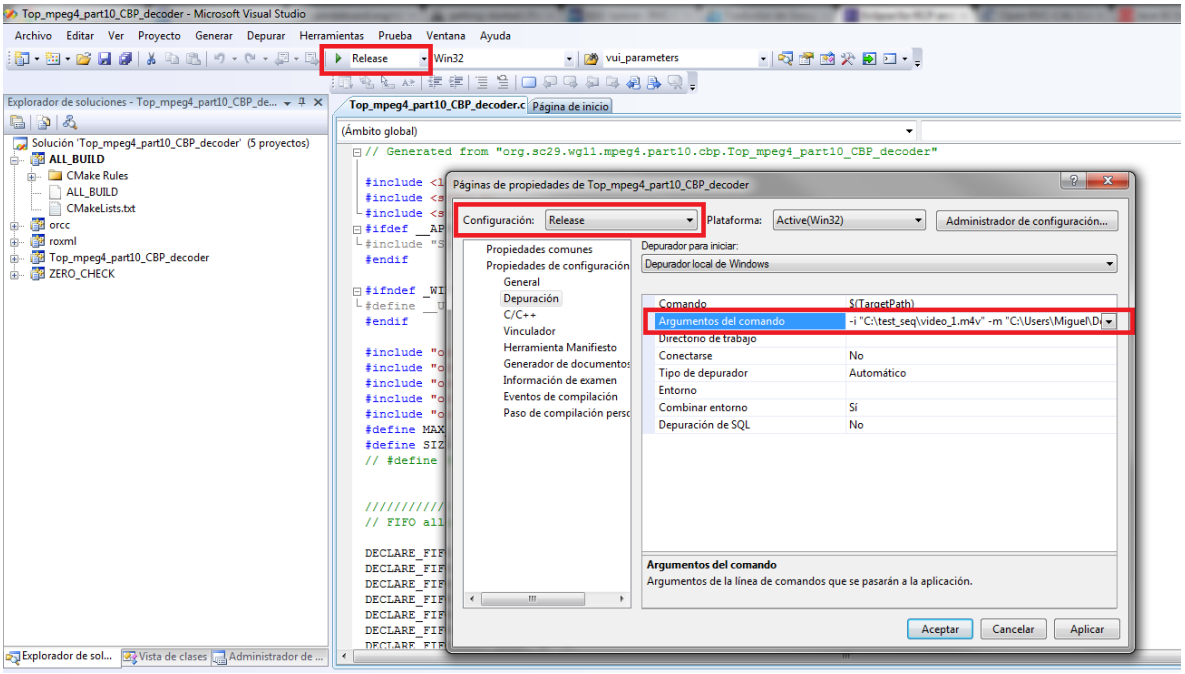

Fig. 43: Window of the Visual Studio 2008 configurations.

11.1. To work in *Release* mode instead of *Debug* change the option on the top of the window as show in Fig. 43.

11. It is needed to copy in the same folder of the *.exe file the SDL.dll file of the SDL library.

12. During the execution of the decoding application the number of FPS decoded will be displayed. This is a good source of information about the execution of the

application and the efficiency of the load distribution performed.

ii) Multicore considerations

This subsection describes main practical procedures to perform a static allocation of the actors that integrate a decoder application with Orcc. Before run Orcc compiler over Eclipse, in the options of the compilation, there is a specific tab window to specify the static distribution if it is desired.

In the *Running Configurations* window there is the *mapping* tab(see Fig. 44). In this tab will appear all the actors which integrate the targeted decoder application. At right side there is a free space for include each core identifier in which the developer wants to allocate each actor named at left side.

This configuration will be compiled for generating a *.xcf text file where all the information about the allocation will be included. This file can be modified directly by the developer after obtaining the decoder executable file. It is recommended to use a specific tool as text editor (i.e. PSPad, see chapter 4 subsection 4.5) to modify this file, and also for specify the character coding of it (UTF-8, ANSI, ISO 8859...). Developer can change the distribution of the actors moving from one partition to another each of the instances to the actors mapped.

Depending on the available cores for the targeted platform on which the decoder will run, the allocation can be done in one way or another. For example for a distribution of two cores (it is no needed to use all available cores on the platform) in the *.xcf appear a number of instances in partition 0 (<Partition id="0">) and others in the partition 1 (<Partition id="1">). The partition identifier in quotation marks should be a number, cannot take characters or other symbols, but in the mapping distribution when running Orcc, it is possible to use other identifiers, such as “c0” and “c1”.

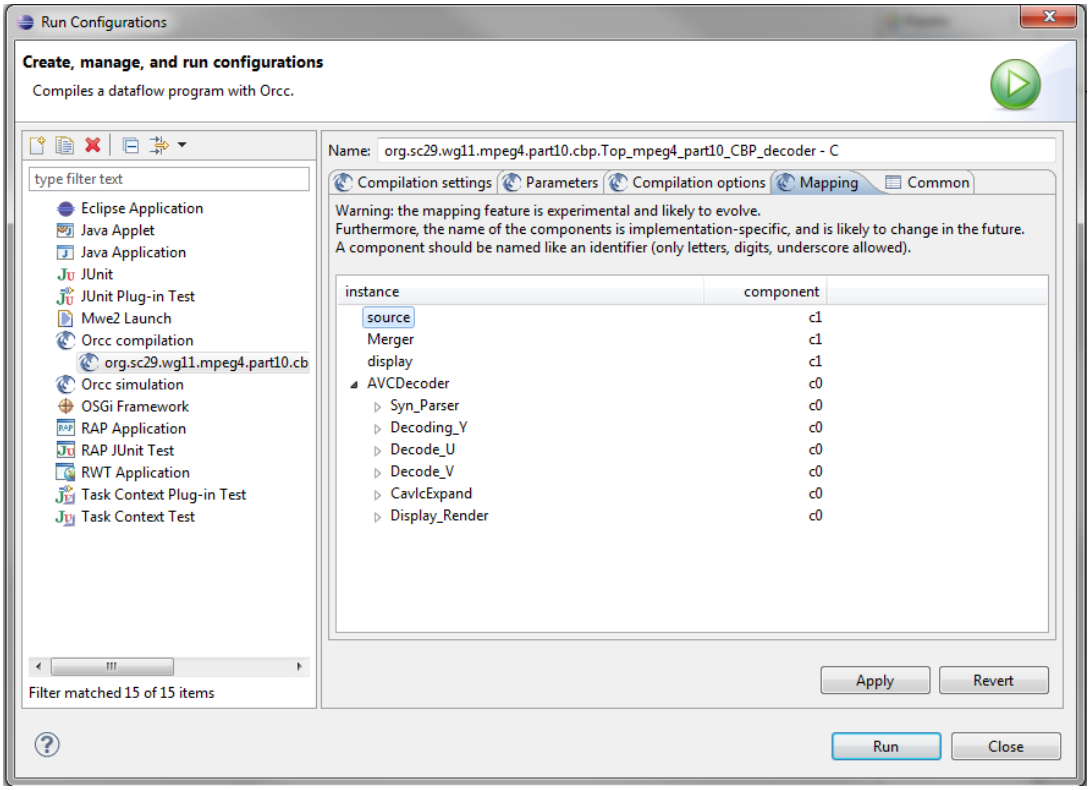

Fig. 44: Mapping properties window with Orcc over Eclipse.

It is necessary to indicate and specify the allocation of the *.xcf file when the application will be executed. For instance, if the application runs from a terminal, it will be need at least the following arguments:

C:/ Workbench/application/build/debug/Decoder_app_1.0.exe -i "C :/ video_test_sequences/video_1.container_extension" -m "/ workbench /application/srv/ file.xdf"

The “-i” identifier indicates the path of the video to be decoded, and “-m” identifier is for the file path *.xcf, also it is needed to include the extension of both files.

It is necessary to check the character encoding used in the *.xcf file, for instance opening it with a specialized text editor (PSPad, see chapter 4 subsection 4.5). For the working environment described, the encoding must be in UNIX (although it works in DOS environment) and the character set UTF-8. If this is not done, or the encoding is wrong, when running the application, it will return an error while reading the file *.xcf.

ANNEX IV shows the complete code for an example of a static load distribution. It shows in detail the distribution of the *.xcf file. This file is organized according to the references to each of the processor cores available, i.e. <Partition id="0"/> to the first core. From each reference to a core, the instances include call actors who are associated with each play list for each core. Examples of these instances would be: actor that performs the functionality of displaying: <instance id="display"/>. Actor which performs the functionality of the merger of the decoder: <instance id="Merger"/>. One of the actors that implement the parser of an AVC decoder: <instance id="AVCDecoder_Syn_Parser_Algo_Synp"/>

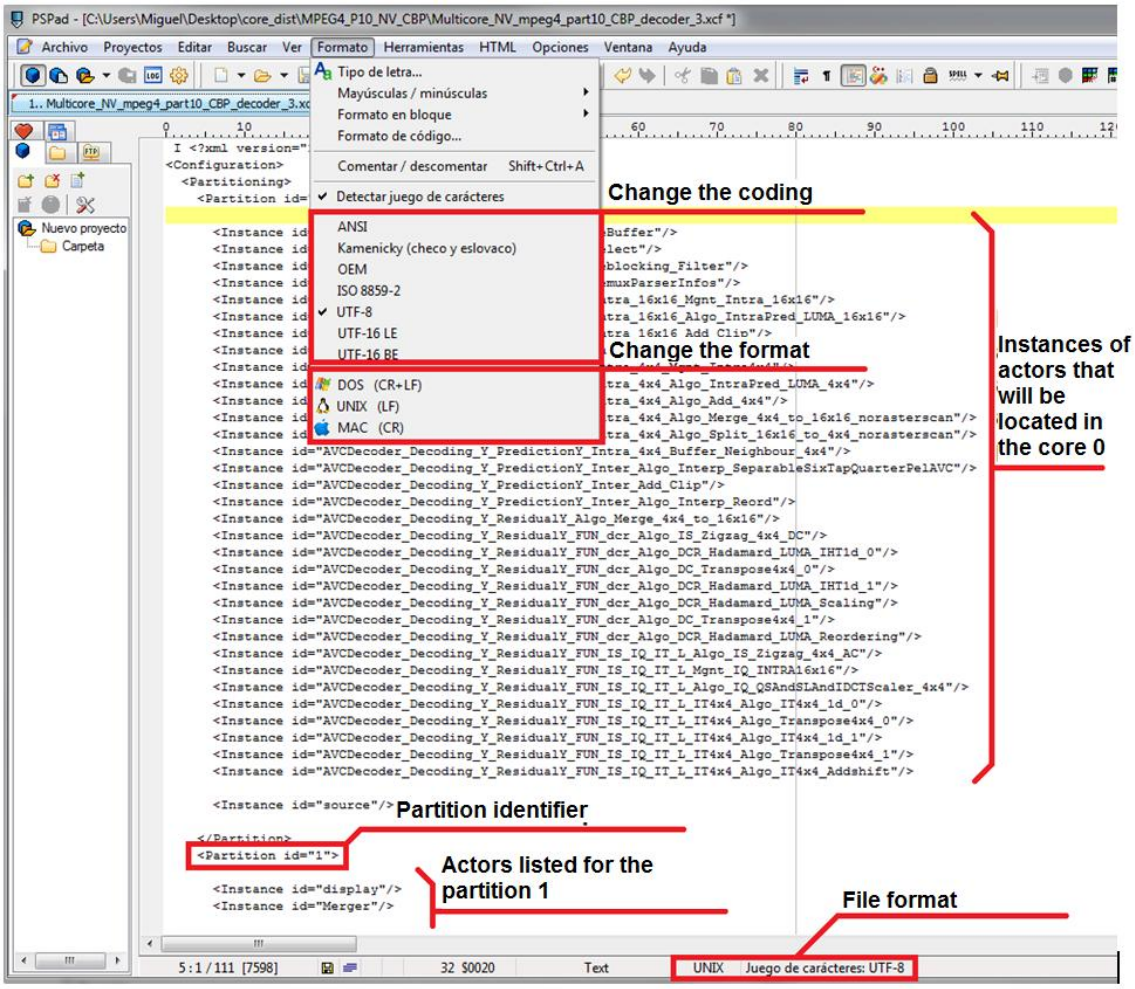

Fig. 45: Window of the PSPad text editor with an example of *.xdf file.

ANNEX II

This annex gives the procedures to set up and install all the working environment and also first steps to start working with the PandaBoard.

This annex II is the extension of the annex I, in the sense of have a complete working environment for start working and perform different experiments with all the concepts introduced along this master thesis memory. It is highly recommended to read all annex before start installation procedure.

1. Firstly must install and configure the SD card to host the operating system, used with the PandaBoard. Here the chosen version is Ubuntu Desktop 11.10 [Ubu11]. This version is available on the website of Ubuntu [Ubu12] and includes the packages needed to start working over the PandaBoard. Also includes graphical settings if wish to work with a monitor connected to the HDMI connector instead of using a serial terminal.

2. Is required to have a 4Gb SD card or higher, but it is recommended that at least it has a total space of 8GB and may also version 10, as earlier versions may be too slow. The card must be completely empty. It is not necessary to have any type of file format and no partition set. Otherwise using the GParted application [Gpa12] of Ubuntu it can be formatted.

3. Download the desired image of the compressed binary version of Ubuntu, and verify that they are appropriate for the architecture on being working, in this case OMAP4. The download can be made directly from the link of omappedia.org [OP12].

4. Once downloaded the compressed file, it is only needed to start the writing of the image of the OS files over the SD card. Then, open a terminal and execute following command:

   gunzip -c ubuntu-11.10-preinstalled-server-armel+<omap image>.img.gz | sudo dd bs=4M of=/dev/<nombre_del_dispositivo>

Note: The device (SD card) must not be mounted. This can be verified by executing the "df" command in the terminal, if it were mounted should proceed to dismantle them before executing the gunzip command, using the "umount" command. Normally the SD

card will correspond with the ID /dev/sdb. But it is important to ensure that the device identifier or name really corresponds with the desired.

5. Right now the card is ready to insert into the SD card slot and perform the PandaBoard first system load. As typical at this first execution shall be perform the initial system configuration, i.e. language, region, keyboard layout, etc... The whole installation process is semi automatic.

6. After the initial setup the system is ready to be used as normal. Typically in the first run, the system will indicate that a number of updates are available and that are recommended to install.

7. The next step is the CMake installation, using the Ubuntu software center (see Fig. 46).

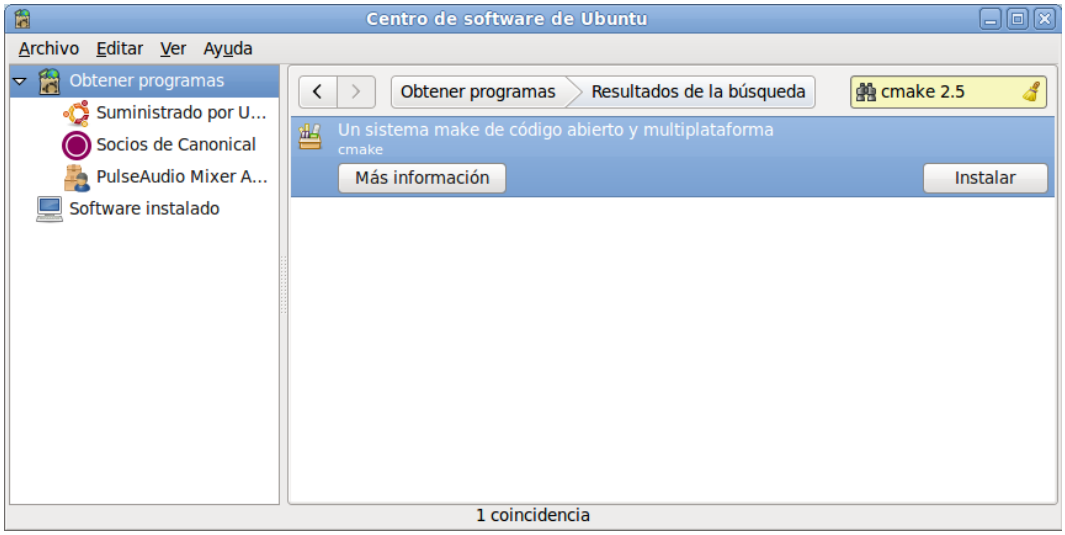

Fig. 46: Ubuntu software center window for CMake installation.

8. Downloading and installation of the packages for the development and compilation of applications:

   8.1. Downloading and installation of the *build-essential* package. Execute the following command in a terminal window: *sudo apt-get install build-essential*

   8.2. Downloading and installation of the *SDL* libraries. Execute the following command in a terminal window: *sudo apt-get install libsdl1.2-dev*

   8.3. Downloading and installation of the *SDL_IMAGES* package. Execute the following command in a terminal window: *sudo apt-get install libsdl-image1.2 libsdl-image1.2-dev*

9. At this moment the situation is the same already exposed in annex I for the PC-based environment installation. It will depend on the Orcc version, if it will be necessary to compile the libraries of Orcc. Now, the system is already prepared to start the compilation of video decoding applications performed with Eclipse.

10. Open with CMake the *.bat file generated by Eclipse/Orcc. This file must be available in the directory: /workspace/application/srv, and execution the compilation for a UNIX environment. Now, it is needed to indicate the working mode *Debug* or *Release*, by writing it in the value field of CMAKE_BUILD_TYPE. Then, lunch the configuration and generation of the solutions for the desired application (see Fig. 47).

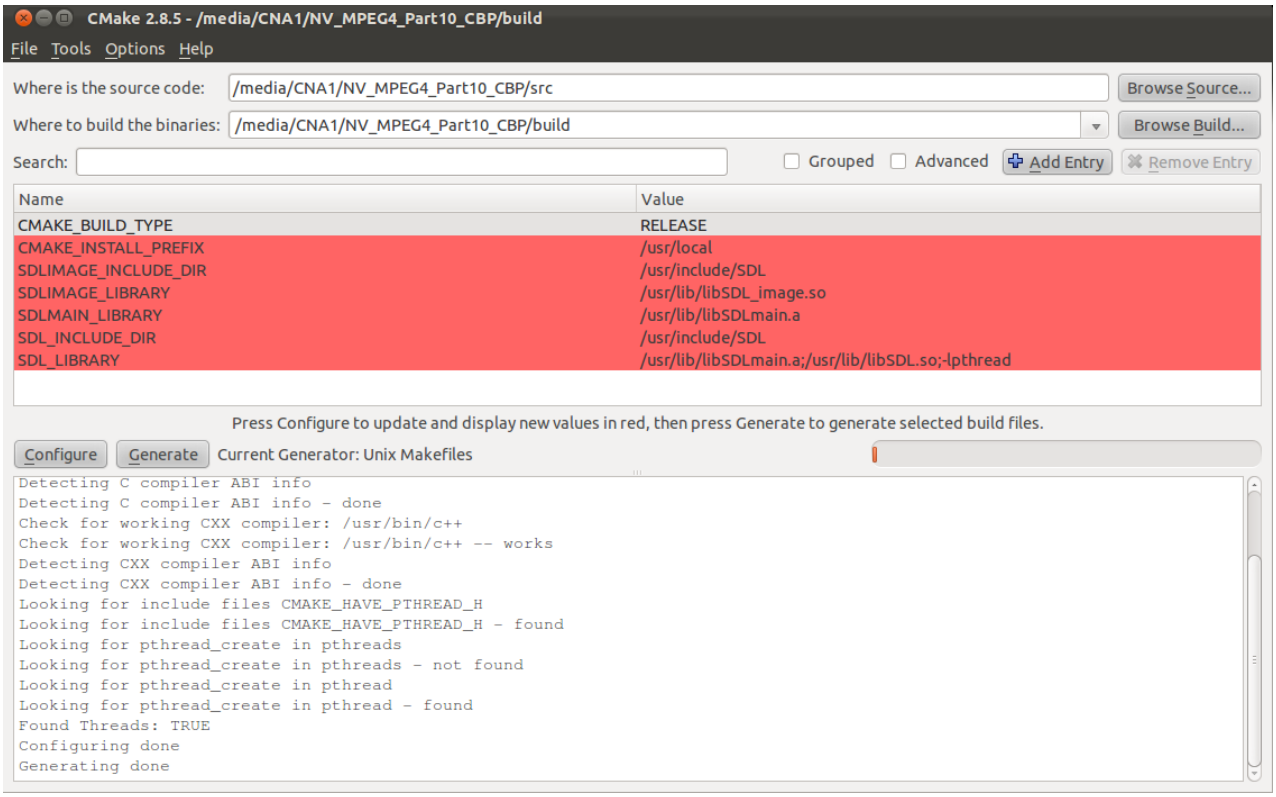

Fig. 47: CMake execution over PandaBoard.

11. Finally, open a terminal window and locate the *build* folder of the solved application. Then execute the command *make.* When *gcc-make* process have finished, the execution file of the video decoding application will be available in the */build/bin/* directory. The following lines exemplifies the execution of the *gcc-make* process:

```
root@mchavarrias-desktop:/media/CNA1/NV_MPEG4_Part10_CBP/build# ls
bin  CMakeCache.txt  CMakeFiles  cmake_install.cmake  libs  Makefile
```

```
root@mchavarrias-desktop:/media/CNA1/NV_MPEG4_Part10_CBP/build# make
Scanning dependencies of target roxml
[  1%] Building C object libs/CMakeFiles/roxml.dir/roxml/src/roxml-
internal.c.o
[  2%] Building C object libs/CMakeFiles/roxml.dir/roxml/src/roxml-parse-
engine.c.o
[  3%] Building C object libs/CMakeFiles/roxml.dir/roxml/src/roxml-parser.c.o
[  4%] Building C object libs/CMakeFiles/roxml.dir/roxml/src/roxml.c.o
Linking C static library libroxml.a
[  4%] Built target roxml
Scanning dependencies of target orcc
[  5%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/accessFile.c.o
[  6%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/compare.c.o
[  7%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/compareyuv.c.o
[  8%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/display.c.o
[  9%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/fpsPrint.c.o
[  9%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/genetic.c.o
[ 10%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/getopt.c.o
[ 11%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/orcc_util.c.o
[ 12%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/scheduler.c.o
[ 13%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/source.c.o
[ 14%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/thread.c.o
[ 15%] Building C object libs/CMakeFiles/orcc.dir/orcc/src/writer.c.o
Linking C static library liborcc.a
[ 15%] Built target orcc
Scanning dependencies of target Top_mpeg4_part10_CBP_decoder
[ 15%] Building C object
CMakeFiles/Top_mpeg4_part10_CBP_decoder.dir/Top_mpeg4_part10_CBP_decoder.c.o
[ 16%] Building C object
CMakeFiles/Top_mpeg4_part10_CBP_decoder.dir/source.c.o
[ 17%] Building C object
CMakeFiles/Top_mpeg4_part10_CBP_decoder.dir/Merger.c.o
[ 18%] Building C object
CMakeFiles/Top_mpeg4_part10_CBP_decoder.dir/display.c.o
……
……
```

12. The working procedure with the execution file is similar as already described for Windows systems. And next lines describe it:

root@user:/home/user/workspace/application/build/bin# ./application_1.0
–i "/home/user/test_sequences/video_1.container"
–m "/home/user/workspace/application/srv/file.xdf"

ANNEX III

This third annex is dedicated to explain the procedures to install and set up all the packages and the environment to start working with LLVM technologies. These procedures have been done for an Ubuntu OS which is working over a VM installed in a PC-based computer.

The needed packages are:

- SVN, which can be installed executing the following command by console: *sudo apt-get install subversion*

- Clang, which can be installed by the described procedures in their own web site [Cla12] or executing the following command by console: *sudo apt-get install clang*. It should be noted that by means the second procedure, the Clang installed version may be not the expected. Our case the Clang version used is the 3.0.

- Install Orcc and Graphiti in Eclipse. In this case the versions chosen are the offered by the Nightly, available in the following link: http://orcc.sourceforge.net/nightly/, this link must be included in the repositories of the new software manager of Eclipse.

- Must be checked that the Xtex version installed in Eclipse is above the 3.2 version.

ANNEX IV

Example of an *.xcf file for perform an assignment of actors which implement the functionality of an MPEG-4 Part 2 decoder over a dual core environment.

```
<?xml version="1.0"?>
<Configuration>
	<Partitioning>
		<Partition id="0">

			<Instance id="source"/>
			<Instance id="decoder_texture_Y_DCsplit"/>
			<Instance id="decoder_texture_Y_IS"/>
			<Instance id="decoder_texture_Y_IAP"/>
			<Instance id="decoder_texture_Y_IQ"/>
			<Instance id="decoder_texture_Y_idct2d"/>
			<Instance id="decoder_texture_Y_DCRecontruction_addressing"/>
			<Instance id="decoder_texture_Y_DCRecontruction_invpred"/>
			<Instance id="decoder_motion_Y_interpolation"/>
			<Instance id="decoder_motion_Y_add"/>
			<Instance id="decoder_motion_Y_FrameBuff"/>

		</Partition>
		<Partition id="1">

			<Instance id="decoder_serialize"/>
			<Instance id="decoder_Merger420"/>
			<Instance id="decoder_parser_parseheaders"/>
			<Instance id="decoder_parser_mvseq"/>
			<Instance id="decoder_parser_blkexp"/>
			<Instance id="decoder_parser_mvrecon"/>
			<Instance id="decoder_parser_splitter_BTYPE"/>
			<Instance id="decoder_parser_splitter_MV"/>
			<Instance id="decoder_parser_splitter_420_B"/>

			<Instance id="decoder_texture_U_DCsplit"/>
			<Instance id="decoder_texture_U_IS"/>
			<Instance id="decoder_texture_U_IAP"/>
			<Instance id="decoder_texture_U_IQ"/>
			<Instance id="decoder_texture_U_idct2d"/>
			<Instance id="decoder_texture_U_DCRecontruction_addressing"/>
			<Instance id="decoder_texture_U_DCRecontruction_invpred"/>
			<Instance id="decoder_texture_V_DCsplit"/>
			<Instance id="decoder_texture_V_IS"/>
			<Instance id="decoder_texture_V_IAP"/>
			<Instance id="decoder_texture_V_IQ"/>
			<Instance id="decoder_texture_V_idct2d"/>
			<Instance id="decoder_texture_V_DCRecontruction_addressing"/>
			<Instance id="decoder_texture_V_DCRecontruction_invpred"/>
			<Instance id="decoder_motion_U_interpolation"/>
			<Instance id="decoder_motion_U_add"/>
			<Instance id="decoder_motion_U_FrameBuff"/>
			<Instance id="decoder_motion_V_interpolation"/>
			<Instance id="decoder_motion_V_add"/>
			<Instance id="decoder_motion_V_FrameBuff"/>
			<Instance id="display"/>

		</Partition>
	</Partitioning>
</Configuration>
```

ANNEX V

In this annex, detailed results obtained along the development of this master thesis work are presented. These results are presented in two tables, the first one show the results for a MPEG-4 part 2 decoding application generated with Orcc and the second one for a MPEG-4 part 10 for CBP and PHP profiles.

Both tables are organized as follows: There is a column for each of these parameters: Type of decoder, for MPEG-4 part 10 also it will indicate the profile and the scheduling policy, if it is used. Video test sequence, the average FPS obtained that the processor is able to decode for each situation, the static distribution of the actors performed, the relation (gain) between perform a static distribution in one core or for two cores. The solution generated (*debug* or *release*) and finally the platform (PC-based or i7 and ARM or PandaBoard).

For the MPEG-4 part 10 results it is added also an extra column with a second check of results.

This wide set of results is the result of a lot of tests performed for the most of the considered situations studied along this Master Thesis work.

As can see for most of situations the gain reached is around 75% or 100%. This proves the importance of the scheduling procedures and the performance of the mapping for the static assignment of the actors that integrate the decoder.

Codec	Video test sequence	Average FPS	Core_Dist	Gain (All in one partition vs multicore)	Solution	Core
MPEG4.Part2 – RVC – Top_RVC	P_VOP_jvc001	32,22	Unicore_dis_1	--	Release	i7
MPEG4.Part2 – RVC – Top_RVC	P_VOP_jvc001	48,73	Multicore_dist_1	1,512414649	Release	i7
MPEG4.Part2 – RVC – Top_RVC	P_VOP_hit001	36,84	Unicore_dis_1	--	Release	i7
MPEG4.Part2 – RVC – Top_RVC	P_VOP_hit001	55,55	Multicore_dist_1	1,507871878	Release	i7
MPEG4.Part2 – RVC – Top_RVC	P_VOP_san002	38,21	Unicore_dis_1	--	Release	i7
MPEG4.Part2 – RVC – Top_RVC	P_VOP_san002	54,95	Multicore_dist_1	1,438105208	Release	i7
MPEG4.Part2 – RVC – Top_RVC	P_VOP_jvc010	37,15	Unicore_dis_1	--	Release	i7
MPEG4.Part2 – RVC – Top_RVC	P_VOP_jvc010	55,14	Multicore_dist_1	1,484253028	Release	i7
MPEG4.Part2 – RVC – Top_RVC	P_VOP_jvc001	6,35	Unicore_dis_1	--	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_jvc001	11,9	Multicore_dist_1	1,874015748	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_jvc001	12,24	Multicore_dist_2	1,927559055	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_hit001	6,82	Unicore_dis_1	--	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_hit001	12,9	Multicore_dist_1	1,891495601	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_hit001	13,75	Multicore_dist_2	2,016129032	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_san002	7,25	Unicore_dis_1	--	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_san002	13,36	Multicore_dist_1	1,842758621	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_san002	14,02	Multicore_dist_2	1,933793103	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_jvc010	7,04	Unicore_dis_1	--	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_jvc010	13,3	Multicore_dist_1	1,889204545	Release	ARM A9
MPEG4.Part2 – RVC – Top_RVC	P_VOP_jvc010	13,81	Multicore_dist_2	1,961647727	Release	ARM A9
Codec	**Video test sequence**	**Average FPS**	**Core_Dist**	**Gain (All in one partition vs multicore)**	**Solution**	**Core**
MPEG4.Part2 – RVC – SP	P_VOP_jvc001	32,19	Unicore_dist_1_Part10	--	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_jvc001	45,85	Multicore_dist_1_Part10	1,42435539	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_jvc001	47,45	Multicore_dist_2_Part10	1,474060267	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_jvc001	50,69	Multicore_dist_3_Part10	1,574712644	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_hit001	34,46	Unicore_dist_1_Part10	--	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_hit001	51,08	Multicore_dist_1_Part10	1,482298317	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_hit001	53,6	Multicore_dist_2_Part10	1,555426582	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_hit001	57,34	Multicore_dist_3_Part10	1,663958212	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_san002	37,11	Unicore_dist_1_Part10	--	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_san002	52,44	Multicore_dist_1_Part10	1,4130962	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_san002	54,1	Multicore_dist_2_Part10	1,457828079	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_san002	58,88	Multicore_dist_3_Part10	1,58663433	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_jvc010	36,08	Unicore_dist_1_Part10	--	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_jvc010	52,44	Multicore_dist_1_Part10	1,453436807	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_jvc010	52,94	Multicore_dist_2_Part10	1,4672949	Release	i7

MPEG4.Part2 – RVC – SP	P_VOP_jvc010	58,47	Multicore_dist_3_Part10	1,62056541	Release	i7
MPEG4.Part2 – RVC – SP	P_VOP_jvc001	6,23	Unicore_dist_1_Part10	--	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_jvc001	11,71	Multicore_dist_1_Part10	1,879614767	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_jvc001	12,42	Multicore_dist_2_Part10	1,993579454	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_jvc001	13,18	Multicore_dist_3_Part10	2,115569823	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_hit001	6,93	Unicore_dist_1_Part10	--	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_hit001	12,95	Multicore_dist_1_Part10	1,868686869	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_hit001	13,74	Multicore_dist_2_Part10	1,982683983	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_hit001	14,55	Multicore_dist_3_Part10	2,0995671	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_san002	7,28	Unicore_dist_1_Part10	--	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_san002	13,41	Multicore_dist_1_Part10	1,842032967	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_san002	14,18	Multicore_dist_2_Part10	1,947802198	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_san002	15,19	Multicore_dist_3_Part10	2,086538462	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_jvc010	7,13	Unicore_dist_1_Part10	--	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_jvc010	13,14	Multicore_dist_1_Part10	1,842917251	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_jvc010	14,04	Multicore_dist_2_Part10	1,96914446	Release	ARM A9
MPEG4.Part2 – RVC – SP	P_VOP_jvc010	15	Multicore_dist_3_Part10	2,103786816	Release	ARM A9

Codec	Video test sequence	Avera-ge FPS	Second check	Core_Dist	Gain (All in one partition vs multicore)	Solution	Core	New Gain with second check
MPEG4.Part10 – NV - CBP	LS_SVA_D.264	105,21	106,76	Unicore_NV_mpeg4_part10_CBP_ decoder	--	Release	i7	
MPEG4.Part10 – NV - CBP	LS_SVA_D.264	178,56	177,76	Multicore_NV_mpeg4_part10_CBP _decoder_1	1,70	Release	i7	1,67
MPEG4.Part10 – NV - CBP	LS_SVA_D.264	171,66	172,85	Multicore_NV_mpeg4_part10_CBP _decoder_2	1,63	Release	i7	1,62
MPEG4.Part10 – NV - CBP	LS_SVA_D.264	198,57	199,35	Multicore_NV_mpeg4_part10_CBP _decoder_3	1,89	Release	i7	1,87
MPEG4.Part10 – NV - CBP	BA2_Sony_F.264	91,10	90,16	Unicore_NV_mpeg4_part10_CBP_ decoder	--	Release	i7	--
MPEG4.Part10 – NV - CBP	BA2_Sony_F.264	161,02	158,40	Multicore_NV_mpeg4_part10_CBP _decoder_1	1,77	Release	i7	1,76
MPEG4.Part10 – NV - CBP	BA2_Sony_F.264	159,18	155,36	Multicore_NV_mpeg4_part10_CBP _decoder_2	1,75	Release	i7	1,72
MPEG4.Part10 – NV - CBP	BA2_Sony_F.264	168,93	168,18	Multicore_NV_mpeg4_part10_CBP _decoder_3	1,85	Release	i7	1,87
MPEG4.Part10 – NV - CBP	HCBP1_HHI_A.264	18,85	19,11	Unicore_NV_mpeg4_part10_CBP_ decoder	--	Release	i7	--
MPEG4.Part10 – NV - CBP	HCBP1_HHI_A.264	36,18	35,36	Multicore_NV_mpeg4_part10_CBP _decoder_1	1,92	Release	i7	1,85
MPEG4.Part10 – NV - CBP	HCBP1_HHI_A.264	34,60	35,30	Multicore_NV_mpeg4_part10_CBP _decoder_2	1,84	Release	i7	1,85
MPEG4.Part10 – NV - CBP	HCBP1_HHI_A.264	36,54	36,13	Multicore_NV_mpeg4_part10_CBP _decoder_3	1,94	Release	i7	1,89
MPEG4.Part10 – NV - CBP	LS_SVA_D.264	18,39	19,47	Unicore_NV_mpeg4_part10_CBP_ decoder	--	Release	ARM A9	
MPEG4.Part10 – NV - CBP	LS_SVA_D.264	37,43	37,54	Multicore_NV_mpeg4_part10_CBP _decoder_1	2,04	Release	ARM A9	1,93
MPEG4.Part10 – NV - CBP	LS_SVA_D.264	33,57	35,82	Multicore_NV_mpeg4_part10_CBP _decoder_2	1,83	Release	ARM A9	1,84
MPEG4.Part10 – NV - CBP	LS_SVA_D.264	36,93	37,40	Multicore_NV_mpeg4_part10_CBP _decoder_3	2,01	Release	ARM A9	1,92
MPEG4.Part10 – NV - CBP	BA2_Sony_F.264	16,79	16,85	Unicore_NV_mpeg4_part10_CBP_ decoder	--	Release	ARM A9	--
MPEG4.Part10 – NV - CBP	BA2_Sony_F.264	31,41	32,20	Multicore_NV_mpeg4_part10_CBP _decoder_1	1,87	Release	ARM A9	1,91

MPEG4.Part10 – NV - CBP	BA2_Sony_F.264	30,58	30,18	Multicore_NV_mpeg4_part10_CBP_decoder_2	1,82	Release	ARM A9	1,79
MPEG4.Part10 – NV - CBP	BA2_Sony_F.264	31,75	32,08	Multicore_NV_mpeg4_part10_CBP_decoder_3	1,89	Release	ARM A9	1,90
MPEG4.Part10 – NV - CBP	HCBP1_HHI_A.264	2,90	3,31	Unicore_NV_mpeg4_part10_CBP_decoder	--	Release	ARM A9	--
MPEG4.Part10 – NV - CBP	HCBP1_HHI_A.264	7,08	6,92	Multicore_NV_mpeg4_part10_CBP_decoder_1	2,44	Release	ARM A9	2,09
MPEG4.Part10 – NV - CBP	HCBP1_HHI_A.264	6,54	6,63	Multicore_NV_mpeg4_part10_CBP_decoder_2	2,26	Release	ARM A9	2,00
MPEG4.Part10 – NV - CBP	HCBP1_HHI_A.264	6,56	6,76	Multicore_NV_mpeg4_part10_CBP_decoder_3	2,26	Release	ARM A9	2,04
Codec	**Video test sequence**	**Average FPS**	**Second check**	**Core_Dist**	**Gain (All in one partition vs multicore)**	**Solution**	**Core**	**New Gain with second check**
MPEG4.Part10 – NV - PHP - Ring_sch	LS_SVA_D.264	105,13	103,21	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	
MPEG4.Part10 – NV - PHP - Ring_sch	LS_SVA_D.264	190,67	186,50	Multicore_NV_MP4_P10_PHP_RSC_1 *see comment*	1,81	Release	i7	1,81
MPEG4.Part10 – NV - PHP - Ring_sch	LS_SVA_D.264	181,21	182,32	Multicore_NV_MP4_P10_PHP_RSC_2	1,72	Release	i7	1,77
MPEG4.Part10 – NV - PHP - Ring_sch	LS_SVA_D.264	181,68	183,27	Multicore_NV_MP4_P10_PHP_RSC_3	1,73	Release	i7	1,78
MPEG4.Part10 – NV - PHP - Ring_sch	BA2_Sony_F.264	86,93	87,65	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Ring_sch	BA2_Sony_F.264	163,74	163,54	Multicore_NV_MP4_P10_PHP_RSC_1	1,88	Release	i7	1,87
MPEG4.Part10 – NV - PHP - Ring_sch	BA2_Sony_F.264	159,73	158,62	Multicore_NV_MP4_P10_PHP_RSC_2	1,84	Release	i7	1,81
MPEG4.Part10 – NV - PHP - Ring_sch	BA2_Sony_F.264	160,21	156,50	Multicore_NV_MP4_P10_PHP_RSC_3	1,84	Release	i7	1,79
MPEG4.Part10 – NV - PHP - Ring_sch	HCBP1_HHI_A.264	18,54	18,15	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Ring_sch	HCBP1_HHI_A.264	33,89	33,26	Multicore_NV_MP4_P10_PHP_RSC_1	1,83	Release	i7	1,83
MPEG4.Part10 – NV - PHP - Ring_sch	HCBP1_HHI_A.264	34,44	32,40	Multicore_NV_MP4_P10_PHP_RSC_2	1,86	Release	i7	1,79
MPEG4.Part10 – NV - PHP - Ring_sch	HCBP1_HHI_A.264	34,57	33,09	Multicore_NV_MP4_P10_PHP_RSC_3	1,86	Release	i7	1,82

MPEG4.Part10 – NV - PHP - Ring_sch	HCMP1_HHI_A.264 CABAC	15,49	16,35	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Ring_sch	HCMP1_HHI_A.264 CABAC	28,53	28,29	Multicore_NV_MP4_P10_PHP_RSC_1	1,84	Release	i7	1,73
MPEG4.Part10 – NV - PHP - Ring_sch	HCMP1_HHI_A.264 CABAC	28,39	27,63	Multicore_NV_MP4_P10_PHP_RSC_2	1,83	Release	i7	1,69
MPEG4.Part10 – NV - PHP - Ring_sch	HCMP1_HHI_A.264 CABAC	27,97	27,84	Multicore_NV_MP4_P10_PHP_RSC_3	1,81	Release	i7	1,70
MPEG4.Part10 – NV - PHP - Ring_sch	LS_SVA_D.264	21,21	21,12	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	
MPEG4.Part10 – NV - PHP - Ring_sch	LS_SVA_D.264	36,61	36,65	Multicore_NV_MP4_P10_PHP_RSC_1	1,73	Release	ARM A9	1,74
MPEG4.Part10 – NV - PHP - Ring_sch	LS_SVA_D.264	38,90	36,23	Multicore_NV_MP4_P10_PHP_RSC_2	1,83	Release	ARM A9	1,72
MPEG4.Part10 – NV - PHP - Ring_sch	LS_SVA_D.264	35,08	36,61	Multicore_NV_MP4_P10_PHP_RSC_3	1,65	Release	ARM A9	1,73
MPEG4.Part10 – NV - PHP - Ring_sch	BA2_Sony_F.264	16,71	17,27	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Ring_sch	BA2_Sony_F.264	31,28	32,24	Multicore_NV_MP4_P10_PHP_RSC_1	1,87	Release	ARM A9	1,87
MPEG4.Part10 – NV - PHP - Ring_sch	BA2_Sony_F.264	32,48	30,56	Multicore_NV_MP4_P10_PHP_RSC_2	1,94	Release	ARM A9	1,77
MPEG4.Part10 – NV - PHP - Ring_sch	BA2_Sony_F.264	33,35	31,21	Multicore_NV_MP4_P10_PHP_RSC_3	2,00	Release	ARM A9	1,81
MPEG4.Part10 – NV - PHP - Ring_sch	HCBP1_HHI_A.264	3,63	3,66	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Ring_sch	HCBP1_HHI_A.264	6,78	6,93	Multicore_NV_MP4_P10_PHP_RSC_1	1,87	Release	ARM A9	1,89
MPEG4.Part10 – NV - PHP - Ring_sch	HCBP1_HHI_A.264	6,73	6,70	Multicore_NV_MP4_P10_PHP_RSC_2	1,85	Release	ARM A9	1,83
MPEG4.Part10 – NV - PHP - Ring_sch	HCBP1_HHI_A.264	6,82	6,84	Multicore_NV_MP4_P10_PHP_RSC_3	1,88	Release	ARM A9	1,87
MPEG4.Part10 – NV - PHP - Ring_sch	HCMP1_HHI_A.264 CABAC	3,08	2,93	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Ring_sch	HCMP1_HHI_A.264 CABAC	5,99	6,06	Multicore_NV_MP4_P10_PHP_RSC_1	1,94	Release	ARM A9	2,07
MPEG4.Part10 – NV - PHP - Ring_sch	HCMP1_HHI_A.264 CABAC	6,18	6,02	Multicore_NV_MP4_P10_PHP_RSC_2	2,01	Release	ARM A9	2,05

MPEG4.Part10 – NV - PHP - Ring_sch	HCMP1_HHI_A.264 CABAC	6,26	6,13	Multicore_NV_MP4_P10_PHP_RSC_3	2,03	Release	ARM A9	2,09
Codec	**Video test sequence**	**Average FPS**	**Second check**	**Core_Dist**	**Gain (All in one partition vs multicore)**	**Solution**	**Core**	**New Gain with second check**
MPEG4.Part10 – NV - PHP - Mesh_sch	LS_SVA_D.264	104,32	105,05	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	
MPEG4.Part10 – NV - PHP - Mesh_sch	LS_SVA_D.264	194,51	195,67	Multicore_NV_MP4_P10_PHP_RSC_1	1,86	Release	i7	1,86
MPEG4.Part10 – NV - PHP - Mesh_sch	LS_SVA_D.264	190,24	188,47	Multicore_NV_MP4_P10_PHP_RSC_2	1,82	Release	i7	1,79
MPEG4.Part10 – NV - PHP - Mesh_sch	LS_SVA_D.264	190,16	187,26	Multicore_NV_MP4_P10_PHP_RSC_3	1,82	Release	i7	1,78
MPEG4.Part10 – NV - PHP - Mesh_sch	BA2_Sony_F.264	85,60	87,44	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Mesh_sch	BA2_Sony_F.264	168,24	168,23	Multicore_NV_MP4_P10_PHP_RSC_1	1,97	Release	i7	1,92
MPEG4.Part10 – NV - PHP - Mesh_sch	BA2_Sony_F.264	158,58	162,05	Multicore_NV_MP4_P10_PHP_RSC_2	1,85	Release	i7	1,85
MPEG4.Part10 – NV - PHP - Mesh_sch	BA2_Sony_F.264	161,62	161,35	Multicore_NV_MP4_P10_PHP_RSC_3	1,89	Release	i7	1,85
MPEG4.Part10 – NV - PHP - Mesh_sch	HCBP1_HHI_A.264	23,14	23,60	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Mesh_sch	HCBP1_HHI_A.264	43,14	44,10	Multicore_NV_MP4_P10_PHP_RSC_1	1,86	Release	i7	1,87
MPEG4.Part10 – NV - PHP - Mesh_sch	HCBP1_HHI_A.264	41,40	41,94	Multicore_NV_MP4_P10_PHP_RSC_2	1,79	Release	i7	1,78
MPEG4.Part10 – NV - PHP - Mesh_sch	HCBP1_HHI_A.264	41,34	41,58	Multicore_NV_MP4_P10_PHP_RSC_3	1,79	Release	i7	1,76
MPEG4.Part10 – NV - PHP - Mesh_sch	HCMP1_HHI_A.264 CABAC	16,54	16,50	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Mesh_sch	HCMP1_HHI_A.264 CABAC	29,58	29,54	Multicore_NV_MP4_P10_PHP_RSC_1	1,79	Release	i7	1,79
MPEG4.Part10 – NV - PHP - Mesh_sch	HCMP1_HHI_A.264 CABAC	30,02	29,93	Multicore_NV_MP4_P10_PHP_RSC_2	1,81	Release	i7	1,81
MPEG4.Part10 – NV - PHP - Mesh_sch	HCMP1_HHI_A.264 CABAC	29,52	30,21	Multicore_NV_MP4_P10_PHP_RSC_3	1,78	Release	i7	1,83
MPEG4.Part10 – NV - PHP - Mesh_sch	LS_SVA_D.264	20,02	20,32	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	

MPEG4.Part10 – NV - PHP - Mesh_sch	LS_SVA_D.264	26,85	32,29	Multicore_NV_MP4_P10_PHP_RSC_1	1,34	Release	ARM A9	1,59
MPEG4.Part10 – NV - PHP - Mesh_sch	LS_SVA_D.264	34,50	34,89	Multicore_NV_MP4_P10_PHP_RSC_2	1,72	Release	ARM A9	1,72
MPEG4.Part10 – NV - PHP - Mesh_sch	LS_SVA_D.264	37,06	39,41	Multicore_NV_MP4_P10_PHP_RSC_3	1,85	Release	ARM A9	1,94
MPEG4.Part10 – NV - PHP - Mesh_sch	BA2_Sony_F.264	16,97	16,84	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Mesh_sch	BA2_Sony_F.264	22,76	25,55	Multicore_NV_MP4_P10_PHP_RSC_1	1,34	Release	ARM A9	1,52
MPEG4.Part10 – NV - PHP - Mesh_sch	HCBP1_HHI_A.264	2,87	2,85	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Mesh_sch	HCBP1_HHI_A.264	5,42	5,62	Multicore_NV_MP4_P10_PHP_RSC_1	1,89	Release	ARM A9	1,97
MPEG4.Part10 – NV - PHP - Mesh_sch	HCBP1_HHI_A.264	5,83	5,76	Multicore_NV_MP4_P10_PHP_RSC_2	2,03	Release	ARM A9	2,02
MPEG4.Part10 – NV - PHP - Mesh_sch	HCBP1_HHI_A.264	6,05	5,99	Multicore_NV_MP4_P10_PHP_RSC_3	2,11	Release	ARM A9	2,10
MPEG4.Part10 – NV - PHP - Mesh_sch	HCMP1_HHI_A.264 CABAC	3,07	3,03	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Mesh_sch	HCMP1_HHI_A.264 CABAC	5,94	5,02	Multicore_NV_MP4_P10_PHP_RSC_1	1,93	Release	ARM A9	1,66
MPEG4.Part10 – NV - PHP - Mesh_sch	HCMP1_HHI_A.264 CABAC	6,17	5,98	Multicore_NV_MP4_P10_PHP_RSC_2	2,01	Release	ARM A9	1,97
MPEG4.Part10 – NV - PHP - Mesh_sch	HCMP1_HHI_A.264 CABAC	6,10	5,96	Multicore_NV_MP4_P10_PHP_RSC_3	1,99	Release	ARM A9	1,97
Codec	**Video test sequence**	**Average FPS**	**Second check**	**Core_Dist**	**Gain (All in one partition vs multicore)**	**Solution**	**Core**	**New gain with second check**
MPEG4.Part10 – NV - PHP - Ring_sch	CABA3_Sony_C.264 LFON	67,30	65,94	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	
MPEG4.Part10 – NV - PHP - Ring_sch	CABA3_Sony_C.264 LFON	125,49	123,29	Multicore_NV_MP4_P10_PHP_RSC_1	1,86	Release	i7	1,87
MPEG4.Part10 – NV - PHP - Ring_sch	CABA3_Sony_C.264 LFON	120,35	118,43	Multicore_NV_MP4_P10_PHP_RSC_2	1,79	Release	i7	1,80
MPEG4.Part10 – NV - PHP - Ring_sch	CABA3_Sony_C.264 LFON	117,16	118,28	Multicore_NV_MP4_P10_PHP_RSC_3	1,74	Release	i7	1,79
MPEG4.Part10 – NV - PHP - Ring_sch	CAMP_MOT_FRM0.264 LFON	4,30	4,41	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--

MPEG4.Part10 – NV - PHP - Ring_sch	CAMP_MOT_FRM0 .264 LFON	7,77	7,97	Multicore_NV_MP4_P10_PHP_RSC_1	1,81	Release	i7	1,81
MPEG4.Part10 – NV - PHP - Ring_sch	CAMP_MOT_FRM0 .264 LFON	7,73	7,79	Multicore_NV_MP4_P10_PHP_RSC_2	1,80	Release	i7	1,77
MPEG4.Part10 – NV - PHP - Ring_sch	CAMP_MOT_FRM0 .264 LFON	7,65	7,75	Multicore_NV_MP4_P10_PHP_RSC_3	1,78	Release	i7	1,76
MPEG4.Part10 – NV - PHP - Ring_sch	CANL2_Sony_E.26 4 LFOOF	89,59	90,58	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Ring_sch	CANL2_Sony_E.26 4 LFOOF	162,62	160,78	Multicore_NV_MP4_P10_PHP_RSC_1	1,82	Release	i7	1,78
MPEG4.Part10 – NV - PHP - Ring_sch	CANL2_Sony_E.26 4 LFOOF	165,85	161,22	Multicore_NV_MP4_P10_PHP_RSC_2	1,85	Release	i7	1,78
MPEG4.Part10 – NV - PHP - Ring_sch	CANL2_Sony_E.26 4 LFOOF	165,59	163,68	Multicore_NV_MP4_P10_PHP_RSC_3	1,85	Release	i7	1,81
MPEG4.Part10 – NV - PHP - Ring_sch	CANL3_SVA_B.264 LFOOF	98,18	98,34	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Ring_sch	CANL3_SVA_B.264 LFOOF	187,71	185,30	Multicore_NV_MP4_P10_PHP_RSC_1	1,91	Release	i7	1,88
MPEG4.Part10 – NV - PHP - Ring_sch	CANL3_SVA_B.264 LFOOF	167,34	169,16	Multicore_NV_MP4_P10_PHP_RSC_2	1,70	Release	i7	1,72
MPEG4.Part10 – NV - PHP - Ring_sch	CANL3_SVA_B.264 LFOOF	165,41	167,72	Multicore_NV_MP4_P10_PHP_RSC_3	1,68	Release	i7	1,71
MPEG4.Part10 – NV - PHP - Ring_sch	CABA3_Sony_C.26 4 LFON	12,93	12,86	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	
MPEG4.Part10 – NV - PHP - Ring_sch	CABA3_Sony_C.26 4 LFON	20,72	20,32	Multicore_NV_MP4_P10_PHP_RSC_1	1,60	Release	ARM A9	1,58
MPEG4.Part10 – NV - PHP - Ring_sch	CABA3_Sony_C.26 4 LFON	21,09	21,88	Multicore_NV_MP4_P10_PHP_RSC_2	1,63	Release	ARM A9	1,70
MPEG4.Part10 – NV - PHP - Ring_sch	CABA3_Sony_C.26 4 LFON	20,32	20,48	Multicore_NV_MP4_P10_PHP_RSC_3	1,57	Release	ARM A9	1,59
MPEG4.Part10 – NV - PHP - Ring_sch	CAMP_MOT_FRM0 .264 LFON	0,81	0,80	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Ring_sch	CAMP_MOT_FRM0 .264 LFON	1,28	1,17	Multicore_NV_MP4_P10_PHP_RSC_1	1,58	Release	ARM A9	1,46
MPEG4.Part10 – NV - PHP - Ring_sch	CAMP_MOT_FRM0 .264 LFON	1,55	1,56	Multicore_NV_MP4_P10_PHP_RSC_2	1,91	Release	ARM A9	1,95
MPEG4.Part10 – NV - PHP - Ring_sch	CAMP_MOT_FRM0 .264 LFON	1,54	1,56	Multicore_NV_MP4_P10_PHP_RSC_3	1,90	Release	ARM A9	1,95

MPEG4.Part10 – NV - PHP - Ring_sch	CANL2_Sony_E.264 LFOOF	17,10	17,18	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Ring_sch	CANL2_Sony_E.264 LFOOF	25,87	26,49	Multicore_NV_MP4_P10_PHP_RSC_1	1,51	Release	ARM A9	1,54
MPEG4.Part10 – NV - PHP - Ring_sch	CANL2_Sony_E.264 LFOOF	28,00	29,57	Multicore_NV_MP4_P10_PHP_RSC_2	1,64	Release	ARM A9	1,72
MPEG4.Part10 – NV - PHP - Ring_sch	CANL2_Sony_E.264 LFOOF	26,11	26,94	Multicore_NV_MP4_P10_PHP_RSC_3	1,53	Release	ARM A9	1,57
MPEG4.Part10 – NV - PHP - Ring_sch	CANL3_SVA_B.264 LFOOF	17,56	17,38	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Ring_sch	CANL3_SVA_B.264 LFOOF	24,89	24,93	Multicore_NV_MP4_P10_PHP_RSC_1	1,42	Release	ARM A9	1,43
MPEG4.Part10 – NV - PHP - Ring_sch	CANL3_SVA_B.264 LFOOF	27,32	28,44	Multicore_NV_MP4_P10_PHP_RSC_2	1,56	Release	ARM A9	1,64
MPEG4.Part10 – NV - PHP - Ring_sch	CANL3_SVA_B.264 LFOOF	27,67	28,83	Multicore_NV_MP4_P10_PHP_RSC_3	1,58	Release	ARM A9	1,66

Codec	Video test sequence	Average FPS	Second check	Core_Dist	Gain (All in one partition vs multicore)	Solution	Core	New Gain with second check
MPEG4.Part10 – NV - PHP - Mesh_sch	CABA3_Sony_C.264 LFON	73,46	70,27	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	
MPEG4.Part10 – NV - PHP - Mesh_sch	CABA3_Sony_C.264 LFON	130,46	126,67	Multicore_NV_MP4_P10_PHP_RSC_1	1,78	Release	i7	1,80
MPEG4.Part10 – NV - PHP - Mesh_sch	CABA3_Sony_C.264 LFON	124,23	124,23	Multicore_NV_MP4_P10_PHP_RSC_2	1,69	Release	i7	1,77
MPEG4.Part10 – NV - PHP - Mesh_sch	CABA3_Sony_C.264 LFON	124,34	120,24	Multicore_NV_MP4_P10_PHP_RSC_3	1,69	Release	i7	1,71
MPEG4.Part10 – NV - PHP - Mesh_sch	CAMP_MOT_FRM0.264 LFON	4,75	4,79	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Mesh_sch	CAMP_MOT_FRM0.264 LFON	7,87	7,91	Multicore_NV_MP4_P10_PHP_RSC_1	1,66	Release	i7	1,65
MPEG4.Part10 – NV - PHP - Mesh_sch	CAMP_MOT_FRM0.264 LFON	7,91	8,26	Multicore_NV_MP4_P10_PHP_RSC_2	1,67	Release	i7	1,72
MPEG4.Part10 – NV - PHP - Mesh_sch	CAMP_MOT_FRM0.264 LFON	7,79	8,33	Multicore_NV_MP4_P10_PHP_RSC_3	1,64	Release	i7	1,74
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL2_Sony_E.264 LFOOF	89,53	90,71	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL2_Sony_E.264 LFOOF	155,58	155,35	Multicore_NV_MP4_P10_PHP_RSC_1	1,74	Release	i7	1,71

MPEG4.Part10 – NV - PHP - Mesh_sch	CANL2_Sony_E.264 LFOOF	161,79	170,40	Multicore_NV_MP4_P10_PHP_RSC_2	1,81	Release	i7	1,88
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL2_Sony_E.264 LFOOF	161,92	167,02	Multicore_NV_MP4_P10_PHP_RSC_3	1,81	Release	i7	1,84
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL3_SVA_B.264 LFOOF	103,16	104,29	Unicore_NV_MP4_P10_PHP_RSC	--	Release	i7	--
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL3_SVA_B.264 LFOOF	187,51	185,07	Multicore_NV_MP4_P10_PHP_RSC_1	1,82	Release	i7	1,77
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL3_SVA_B.264 LFOOF	168,42	170,62	Multicore_NV_MP4_P10_PHP_RSC_2	1,63	Release	i7	1,64
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL3_SVA_B.264 LFOOF	168,41	166,06	Multicore_NV_MP4_P10_PHP_RSC_3	1,63	Release	i7	1,59
MPEG4.Part10 – NV - PHP - Mesh_sch	CABA3_Sony_C.264 LFON	12,70	12,98	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	
MPEG4.Part10 – NV - PHP - Mesh_sch	CABA3_Sony_C.264 LFON	18,52	20,62	Multicore_NV_MP4_P10_PHP_RSC_1	1,46	Release	ARM A9	1,59
MPEG4.Part10 – NV - PHP - Mesh_sch	CAMP_MOT_FRM0.264 LFON	0,79	0,82	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Mesh_sch	CAMP_MOT_FRM0.264 LFON	1,21	1,34	Multicore_NV_MP4_P10_PHP_RSC_1	1,53	Release	ARM A9	1,63
MPEG4.Part10 – NV - PHP - Mesh_sch	CAMP_MOT_FRM0.264 LFON	1,52	1,67	Multicore_NV_MP4_P10_PHP_RSC_2	1,92	Release	ARM A9	2,04
MPEG4.Part10 – NV - PHP - Mesh_sch	CAMP_MOT_FRM0.264 LFON	1,60	1,61	Multicore_NV_MP4_P10_PHP_RSC_3	2,03	Release	ARM A9	1,96
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL2_Sony_E.264 LFOOF	17,40	17,56	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL2_Sony_E.264 LFOOF	26,22	25,69	Multicore_NV_MP4_P10_PHP_RSC_1	1,51	Release	ARM A9	1,46
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL3_SVA_B.264 LFOOF	17,88	17,69	Unicore_NV_MP4_P10_PHP_RSC	--	Release	ARM A9	--
MPEG4.Part10 – NV - PHP - Mesh_sch	CANL3_SVA_B.264 LFOOF	24,85	27,16	Multicore_NV_MP4_P10_PHP_RSC_1	1,39	Release	ARM A9	1,54