

# Formal Verification of an MPEG Decoder Chip

*A case study in the industrial use of formal methods*

Avi Parash  
Zoran Corporation  
Haifa, Israel  
parash@zoran.co.il

## Abstract

*This paper describes an industrial experience of Zoran corporation in utilizing (starting with evaluation of) formal specification and verification in a company active in the consumer market (high sensitivity for low-priced products). The evaluation process described here, provided ground basis for a strategic decision to incorporate formal verification as a mandatory part of Zoran's verification methodology. The paper specifically describes formal specification and verification for part of the DVD decoder IC, the control logic block of the MPEG1/2 video decoder. It describes an implementation of a formal MPEG encoder which is capable of generating non-deterministic set of control sequences associated with the encoded bitstream.*

**Keywords:** Formal Methods, Formal Specification, Formal Verification, MPEG1, MPEG2, Hardware Verification.

## 1 Introduction

This paper describes applying formal verification using model checking of the control unit in a DVD decoder chip. The work was initiated to evaluate and explore how formal techniques and tools could be introduced into a small-mid size industrial company, for which cost/performance is very sensitive.

Zoran Corporation has successfully developed a single chip solution, for playback of DVD, VCD, SVCD, CD-I and CD-DA [4] types of discs. The chip decodes and displays MPEG1 and MPEG2 system bitstreams including video and audio. In addition it decodes sub-picture and On Screen Display (OSD). The company's integrated circuits are used in a variety of video and audio products addressing growing consumer multimedia markets.

The verification methods that were previously used to verify the chip were based on a software simulator approach (a behavioral implementation of the chip written in software) Vs. the hardware implementation written in verilog by a set of tests. The tests tried to cover as many possible combinations as were initially defined in the test plan. Although the chip was successful in first version, over the time several problems were identified in the control mechanism. The problems which were uncovered by the simulation environment were later analyzed and observed by the formal methods.

RuleBase is a model-checking based formal verification tool, developed by the IBM Haifa Research Laboratory [1]. It uses an enhanced version of SMV[3] as its verification engine. SMV is an efficient and robust symbolic model checker that uses binary decision diagrams (BDD) to represent the unit under test (UUT).

The work described here provides brief technical overview of the MPEG elements which were used in building the formal model of the block. The emphasis is on the evaluation process and the conclusions from this experience as of using formal verification tool as a part of the chip verification methodology.

The remainder of this paper is organized as follows: Section 2 gives a short overview on MPEG high level video syntax and semantics specification. Section 3 describes a MPEG decoder chip structure. Section 4 provides a summary of formal verification experience. Section 5 describes the implementation of a formal models. Summary and conclusions are given in section 6.

## 2 MPEG video syntax and semantics background

MPEG Video is a generic method for compressed representation of video sequences. The MPEG-2 concept is similar to MPEG-1, but includes extensions to cover a wider range of applications. The representation supports constant bit rate transmission, variable bit rate transmission, random access, channel hopping, scalable decoding, bitstream editing, as well as special functions such as fast forward playback, fast reverse playback, slow motion, pause and still pictures.

### 2.1 Video sequence

The highest syntactic structure of the coded video bitstream is the video sequence. A video sequence commences with a sequence header which may optionally be followed by a group of pictures header and then by one or more coded frames. The order of the coded frames in the coded bitstream is the order in which the decoder processes them, but not necessarily in the correct order for display. The video sequence is terminated by a `sequence_end_code`.

### 2.2 Progressive and interlaced sequences

The MPEG2 specification deals with coding of both progressive and interlaced sequences. The output of the decoding process, for interlaced sequences, consists of a series of reconstructed fields that are separated in time by a field period. The two fields of a frame may be coded separately (field-pictures). Alternatively the two fields may be coded together as a frame (frame-pictures). Both frame pictures and field pictures may be used in a single video sequence. In progressive sequences each picture in the sequence shall be a frame picture. The sequence, at the output of the decoding process, consists of a series of reconstructed frames that are separated in time by a frame period.

### 2.2 Frame and Field

A frame is the union of a top field and a bottom field. The top field is the field that contains the top-most line of each of the three matrices (a luminance matrix (Y), and two chrominance matrices (Cb and Cr)). The bottom field is the other one.

### 2.3 Picture

A reconstructed picture is obtained by decoding a coded picture, i.e. a picture header, the optional extensions immediately following it, and the picture data. A coded picture may be a frame picture or a field picture. A reconstructed picture is either a reconstructed frame (when decoding a frame picture), or one field of a reconstructed frame (when decoding a field picture).

### 2.4 Picture types

There are three types of pictures that use different coding methods.

An **Intra-coded (I) picture** is coded using information only from itself.

A **Predictive-coded (P) picture** is a picture which is coded using motion compensated prediction from a reference frame.

A **Bidirectionally predictive-coded (B) picture** is a picture which is coded using motion compensated prediction from a past and/or future reference frame(s). Figure 1 illustrates an example of the relationship among the three different picture types.

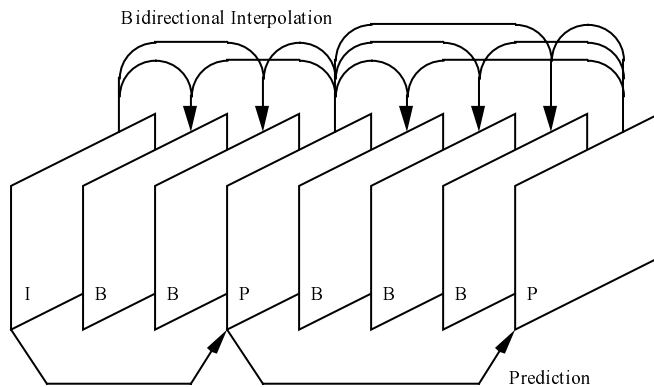


Figure 1

## 2.5 Field pictures

If field pictures are used then they shall occur in pairs (one top field followed by one bottom field, or one bottom field followed by one top field) and together constitute a coded frame. The two field pictures that comprise a coded frame shall be encoded in the bitstream in the order in which they shall occur at the output of the decoding process.

When the first picture of the coded frame is a P-field picture, then the second picture of the coded frame shall also be a P-field picture. Similarly when the first picture of the coded frame is a B-field picture the second picture of the coded frame shall also be a B-field picture.

When the first picture of the coded frame is a I-field picture, then the second picture of the frame shall be either an I-field picture or a P-field picture.

When coding interlaced sequences using frame pictures, the two fields of the frame shall be interleaved with one another and then the entire frame is coded as a single frame-picture.

## 2.6 Frame reordering

When the sequence contains coded B-frames the number of consecutive coded B-frames is variable and unbounded. The first coded frame after a sequence header shall not be a B-frame. A sequence may contain no coded P-frames. A sequence may also contain no coded I-frames in which case some care is required at the start of the sequence and within the sequence to affect both random access and error recovery.

The order of the coded frames in the bitstream, also called *decoding order*, is the order in which a decoder reconstructs them. The order of the reconstructed frames at the output of the decoding process, also called the *display order*, is not always the same as the coded order and this section defines the rules of frame reordering that shall happen within the decoding process.

When the sequence contains no coded B-frames the coded order is the same as the display order.

When B-frames are present in the sequence, re-ordering is performed according to the following rules:

- If the current frame in coded order is a B-frame the output frame is the frame reconstructed from that B-frame.
- If the current frame in coded order is a I-frame or P-frame the output frame is the frame reconstructed from the previous I-frame or P-frame if one exists.
- If the final frame has been decoded at the end of the sequence the frame reconstructed from the previous I-frame or P-frame is output.

## 2.7 TFF and RFF

TFF- (Top Field First): In a field picture `top_field_first` shall have the value "0", and the only field output by the decoding process is the decoded field picture.

In a frame picture `top_field_first` being set to “1” indicates that the top field of the reconstructed frame is the first field output by the decoding process. `top_field_first` being set to “0” indicates that the bottom field of the reconstructed frame is the first field output by decoding process.

RFF (Repeat First Field): This flag is applicable only in a frame picture, in a field picture it shall be set to zero and does not affect the decoding process.

If this flag is set to 0, the output of the decoding process corresponding to this reconstructed frame consists of two fields. The first field (top or bottom field as identified by `top_field_first`) is followed by the other field.

If it is set to 1, the output of the decoding process corresponding to this reconstructed frame consists of three fields. The first field (top or bottom field as identified by `top_field_first`) is followed by the other field, then the first field is repeated.

Figure 2 summarizes the optional combinations of a MPEG2 bitstream video syntax.

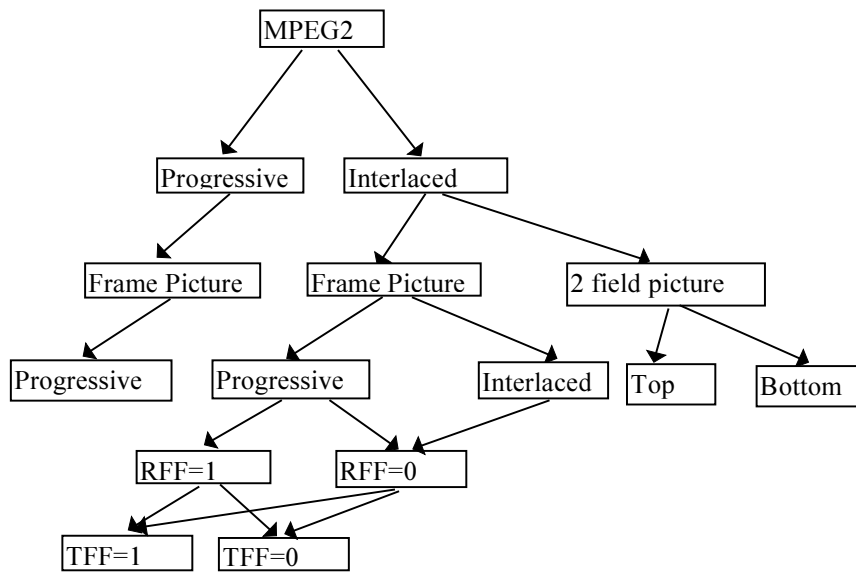


Figure 2

### 3. MPEG decoder chip and simulation environment

The MPEG decoder chip consists of a demultiplex processor, video decoder, reconstruction unit, memory management unit, control block, and video out logic and audio decoder. The on-chip processor is responsible to demultiplex the system stream into video, audio and sub-picture layers to separate streams. The video layer data is processed by the video out logic and controlled by the CONTROL block. The CONTROL block gets the video stream information from the on-chip processor, and several synchronization signals from the video logic. It is responsible to control the flow of data block (picture content) from the input (decoding order) into the memory and from the memory into the video out interface (display order).

The CONTROL block simulation environment had to contain several models: processor for initializing the block with the various commands, video interface to provide the relevant syncs.

The implementation of such models which have to be robust enough, was considered very hard to accomplish. As a result, a large set of full-chip tests was defined and implemented to try and test as many as possible logic combinations. The tests run-time was very long (each test ran more than 4 hours) but although the number of tests was quiet impressive (40 full-chip tests only for this function), the post-silicon validation found some major bugs which one of them even eliminated one of the main functions of the chip (one of the trick command). Those problems were later fixed in the final step of the validation phase.

The following scheme describes the CONTROL block environment:

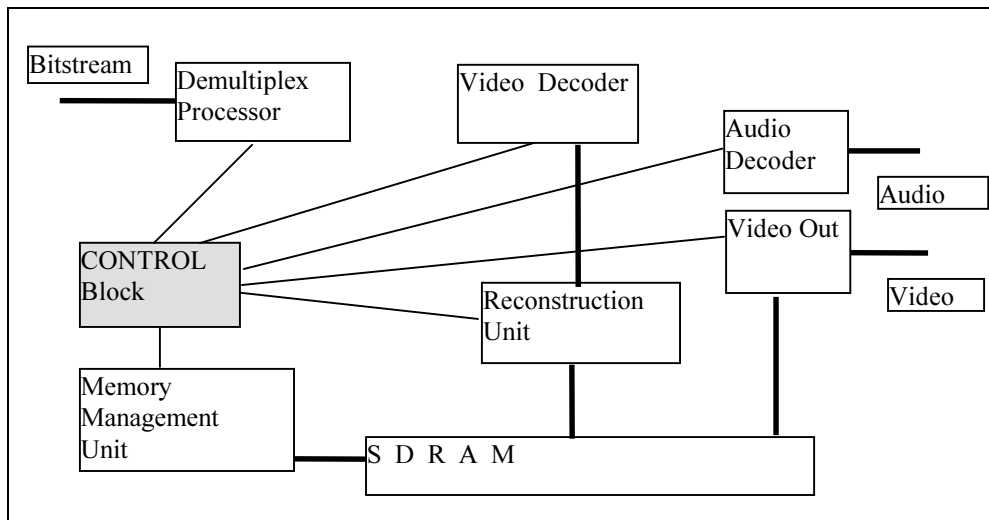


Figure 3

## 4. Formal Verification

### 4.1 Formal verification tool

Formal verification (model checker) is a technology which allows designers to check some properties of their design and then to prove that it is true under all circumstances. Today, two formal verification approaches are available, namely model checking, and theorem proving. In this paper we exclude the description of equivalence checking that is used to compare between two levels of logic representations in formal methods.

Rule-Based (1) is a Model checker. As such, the user provides the design model in VHDL or Verilog RTL code and also provides the design specification in the form of queries. A query describes a property of the design to be verified. The query may also specify a set of constraints on primary inputs of the design. The constraints limit the set (or sequence) of possible values on those inputs consistent with the legal operation of the design. The tool either verifies that the property is true under all possible, allowable conditions, or provides an error trace showing a case where the property fails.

### 4.2 Formal Verification Goals

The main goal was to provide a fast and exhaustive unit verification environment of the CONTROL block. The classic alternative (i.e. unit simulation environment) was considered hard to accomplish due to the need of developing behavioral models which simulate the rest of the chip. Thus only chip level simulation environment was available for testing the block behavior. The second goal was to conduct an evaluation of using the formal verification tool as part of the chip verification methodology.

### 4.3 Tool evaluation process

Table 1 describes the amount of efforts spent during the evaluation of the formal verification tool:

- **Training:** The course was given by IBM to 3 of Zoran's engineers. (Two of them were the designers of the CONTROL block and the third was the verification Eng.). Although it was relatively short (3 days), it covered all the typical and extensive usage of the tool's language structures. No additional training was needed except of a few questions once in a while.
- **Development:** The rest of the evaluation process was devoted to the development of the CONTROL environment and properties. It is important to emphasize that this relatively long period of formal environment development is due to the complicated definition of the block interface. The information used to formally define the environment was initially based on long interaction with the designers who described the function of the block in the full chip and extraction of the various signals from previous simulations timing diagrams. In addition, the

environment was incrementally enhanced and corrected according to the examination of the waveforms associated with the property results.

- **Analysis:** After the environment was stable (i.e. properties passed or real bugs found), incremental enhancement was done to add more robust definition to previously defined input signals. The amount of time required from the designers from this stage was minor. Except for a couple of false alarms (due to incomplete definition of the property or insufficient representation of the input environment), the designer was called to review the results when real bugs were found.

The CONTROL block is a relatively large design (as it was estimated originally) and it required several types of reduction techniques, such as multiple properties versions where each covers a different aspect or range of environment. As the environment became more and more mature, complicated properties which involved larger parts of the design and a greater number of registers, the run-time of each iteration became longer (even reached 24 hours for a property check).

Task	Duration (days)	Designer involvement	Overall (days)
Rule-Base course	3	full	3
Definition of verification plan	5	partial	8
First version environment and first rule running	3	partial	11
First bug found	2	full	13
Environment stable and 20 rules running	5	partial	18
Enhanced environment and 70 rules (5 bugs found)	10	minor	28
Regression	5	non	33

**Table 1 : Level of efforts and verification-design corporation**

## 5 Formal implementation

The first step was to implement the basic environment of the CONTROL block (i.e. processor interface featuring the basic commands, deterministic sync generation along with a simple rule to check the integration of the CONTROL block with the environment. This task was accomplished within 3 days (the relative simulation environment bring-up was estimated in two weeks).

One of the most important advantage of using formal environment was the ability to use an MPEG video encoder (partial ordering information only) which is capable of generating ALL the possible combinations of bitstreams. The various combinations include: Sequence length varies from one to endless, sequence picture type (I, P or B picture in all the various combinations), RFF on any interlaced frame, and TFF to maintain correct display order.

The approach used in simulation environment cannot provide this rich set of options, first due to the limited number of tests it was possible to run, and second due to the need to generate those bitstreams sequences manually. The difficulty in encoding MPEG with RFF and TFF is the necessity to "look into the future". Since RFF impacts the polarity of the displayed field order and the chosen displayed frame is set by the sequence of pictures which is preceding the current picture, it is impossible to set the value of the RFF without knowing what the properties of the following pictures are.

In formal verification we solved the "look into the future" problem by using the "invar" statement as described in the next paragraph.

### 5.1 MPEG Formal Encoder

The formal MPEG encoder is the basic element of the formal model. It is capable of generating non-deterministic set of control sequences associated with the encoded bitstream. This implementation enables a full coverage of all the possible bitstreams information which could be

issued by a MPEG encoder. The number of different bitstreams and their lengths is not limited. Without this formal implementation and the reduction features of the RuleBase tool, there was no way to cover all the testing combinations in a simulation environment. Also, the development of such simulation model is considered as a very complicated task. The simulation based testing model requires usage of predefined set of bitstreams (or video sequences). Each bitstream is not limited in length or in its content (e.g. any legal sequence of picture types). Thus, an enormous number of tests are required.

The formal model definition approach enabled fast development of such encoder (sample of the code related to the MPEG encoder is given below). The formal verification tool based on an efficient search procedure is used to automatically determine the correctness of the properties for all the possible input combinations.

The following code describing the MPEG formal encoder is written in EDL (Rule-Base Environment Description Language). It starts with the video sequence structure generation which includes the picture type, picture structure and other parameters related to the picture header. In order to generate only legal bitstreams in respect to "top field always followed bottom field", we used the **invar** statement which removed all the illegal bitstreams from the reachable states.

```

var
  picture_type_id: {I,P,B};
assign
  init (picture_type_id)= I;
  next (picture_type_id):=
    case
      (processor_state=first_field)&processor_write&first_picture : I;
      (processor_state=first_field)&processor_write: {I,P,B};
      (processor_state=second_field)&processor_write:
        case picture_type_id=I : {I,P};
          picture_type_id=P : P;
          picture_type_id=B : B;
        else : picture_type_id;
      esac;
    else : picture_type_id;
  esac;

```

This machine starts with the first picture which is always I. It saves the value of its TFF and RFF in IDEAL\_SAVED\_TFF, IDEAL\_SAVED\_RFF. It then goes to the second picture which can be I,P or B. If it is I or P, then it display the previous TFF (i.e. 'T' if SAVED\_TFF=1). If it had SAVED\_RFF=1 then it will increment the TOPorBOTTOM from 'T' to 'B' and t`T` again. It will also save the values of TFF and RFF of the current reference picture. If the second picture was B, it will start the display according to the TFF value of the B picture. When ever there is a contradiction between the previous decision and the current (i.e. RFF caused next phase to be Top but the next decoded picture has TFF=1 which means Top again), the ERROR\_IN\_BITSTREAM will be set. This rule is based on the assumption that the display order should be always Top after Bottom or vice-versa.

```

var Ideal_First_Field : {T,B};
assign (Ideal_First_Field):=
  case
    processor_write&picture_type_id=B&TFF : T;
    processor_write&picture_type_id=B&!TFF : B;
    processor_write&(picture_type_id=I||picture_type_id=P)&Saved_TFF : T;
    processor_write&(picture_type_id=I||picture_type_id=P)&!Saved_TFF : B;
  else : T ;
  esac;

var Ideal_Last_Field : {T,B};
assign NEXT (Ideal_Last_Field):=
  case
    processor_write&picture_type_id=B&TFF & !RFF: B;
    processor_write&picture_type_id=B&TFF & RFF: T;
    processor_write&picture_type_id=B&!TFF & !RFF: T;
    processor_write&picture_type_id=B&!TFF & RFF: B;
  esac;

```

```

processor_write&(picture_type_id=I|picture_type_id=P)&Saved_TFF&!Saved_RFF : B;
processor_write&(picture_type_id=I|picture_type_id=P)&Saved_TFF&Saved_RFF : T;
processor_write&(picture_type_id=I|picture_type_id=P)&!Saved_TFF & !Saved_RFF: T;
processor_write&(picture_type_id=I|picture_type_id=P)&!Saved_TFF & Saved_RFF: B;
else : Ideal_Last_Field;
esac;

```

This part will decode the expected field type (Top or Bottom) according to the set of rules described above. The `ideal_first_field` and `ideal_last` field represent the outputs of a decoder model which follow the specification rules.

```

var Saved_TFF: boolean;
    Saved_RFF: boolean;

assign
  INIT (Saved_TFF):=0;
  NEXT (Saved_TFF):=
  case
  picture_type1=FRAME:
    case
    processor_write&(picture_type_id=I|
    processor_write&(picture_type_id=P)&TFF : 1;
    processor_write&(picture_type_id=I|
    processor_write&(picture_type_id=P)&!TFF : 0;
    else : Saved_TFF;
    esac;
  picture_type1=FIELD_PAIR:
    case
    processor_write&(picture_type_id=I|picture_type_id=P)&TOPorBOTTOM=TOP : 1;
    processor_write&(picture_type_id=I|picture_type_id=P)&TOPorBOTTOM=BOTTOM : 0;
    else : Saved_TFF;
    esac;
  esac;

  INIT (Saved_RFF):=0;
  NEXT (Saved_RFF):=
  case
    processor_write&(picture_type_id=I|
    processor_write&(picture_type_id=P)&RFF : 1;
    processor_write&(picture_type_id=I|
    processor_write&(picture_type_id=P)&!RFF : 0;
    else : Saved_RFF;
  esac;

```

The solution for the "look into the future" problem is provided here:  
 The following lines are responsible to take out all the bitstreams that are not legal. The `error_in_bitstream` will be '1' when the Ideal field order will show Top after Top or Bottom after Bottom fields.

```

var ERROR_In_Bitstream: boolean;
assign (ERROR_In_Bitstream) := (Ideal_First_Field=Ideal_Last_Field)&
!first_picture&processor_write;

invar (ERROR_In_Bitstream=0);

```

INVAR stands for: analyze all the paths and extract only those that follow the condition within the brackets.

## 5.2 Formal properties

The first set of properties covers the basic behavior of the CONTROL block (i.e. decoding and display order). These properties helped to bring up a stable environment in means of synchronization with the various external inputs coming from the various blocks in the full chip environment. Although these properties are basically the most important feature of the block and was exhaustively checked in the simulation based environment, it turns out to have the first bug which was found after the second day of running the tool (first rule). The guidelines in writing formal property are to leave as many environment variables free (i.e. no behavioral limitation). This rule was the key element of finding the first bug as the function of one of the basic assumptions (processor command has no timing relevance) was proven to be wrong. All the other properties (70) covered other aspects of the CONTROL functions such as On-Line-Commands, Status bits etc.

The design before RuleBase optimizations was 407 flip-flops, 6700 gates and 88 inputs. Table 2 provides information on selected rules (the numbers are after Rule-Base reduction):

RULE NAME	# FLIP-FLOPS	# ENV VAR	# GATES	# ITERATIONS	# BDD SIZE	# STATES	# USER TIME (SEC)	# MEM (MB)
Rule 1 : MPEG1 decoding order	136	77	4420	544	16569	2.1e15	1200	57
Rule 2 : MPEG2 decoding order	181	53	3720	1536	22131	7.3e12	5200	92
Rule 3 : MPEG1 display order	141	74	4368	897	266555	4.6e19	16405	100
Rule 4 : MPEG2 display order1	142	74	4427	689	36486	6.8e15	8600	94
Rule 5 : MPEG2 display order2	142	74	4407	929	51987	2.1e19	1800	78
Rule 6 : Trick mode command1	199	57	4012	421	261809	1.3e11	62500	165
Rule 7 : Trick mode command2	150	80	3957	1713	184070	5.4e14	2800	60
Rule 8 : Trick mode command3	149	54	3215	769	17875	2.0e10	1000	52

**Table2: Formal properties examples**

- The first five rules were written during the first week of the environment bring-up. Notice that the number of flip-flops each property uses is high. It shows that although the basic environment was able to provide only subset of the block features, it needed many of its flip-flops.
- Rules 4 and 5 are checking MPEG2 display order. Due to a large number of environment variables, it required a partitioning of the original rule into two parts where each part covers different modeling of the inputs.
- Rule 6 is an example of a property that required large environment definition. Most of the rules used environment that was capable to run only one bitstream (i.e. no end/start commands). This was due to the large number of environment variables associated with this feature. The Rule 6 property had to use multiple bitstreams environment (an addition of around 15 environment variables).
- Notice that the user time values are depended on the particular machine power.

## 6 Summary and Conclusions

### 6.1 Design for Formal verifiability

The formal verification process can be made more effective if the design architecture and its implementation follow set of guidelines which can be divided into 3 categories:

**Reduction:** Symbolic model checking addressed the state explosion problem by using BDDs. While several techniques are incorporated in the Rule-Based tool, the size problem still requires to follow some guidelines. Most of the reduction guidelines focus on design partitioning and modularity, with the goal of isolating the control logic from the data path which can be replaced by abstract models. It was found that these guidelines are very effective from design point of view and design for verifiability.

**Documentation:** Keeping well defined and documented interfaces between partitions to ease the development and maintenance of environment models. This is also a good design practice which often leads to better design qualities as well.

**Formal Specification:** Translating the informal specification, usually written in natural language, into a set of formal properties. This task often assumes background knowledge and ability to formulate sentences into formal structures.

## 6.2 Implications to other projects and conclusions

In addition to the COTROL block we verified during the evaluation period, we also verified few other blocks which are different from the first one. The VIP [5] interface is a protocol based logic which had to be compliant with the VIP specification. The Closed-Caption [6] block is part of the video out logic (data path oriented) is applying the closed caption data into the video stream. The work on this block covered few architectural problems which could be hardly found in any other method.

BLOCK NAME	MAIN VERIFICATION METHOD	TIME TO BUILD FORMAL MODELS AND BASIC PROPERTIES	OVERALL VERIFICATION DURATION	# BUGS FOUND (AFTER MAIN VERIFICATION METHOD COMPLETED)
CONTROL	Full-Chip simulation	2 weeks	8 weeks	5
VIP interface [5]	Unit level simulation	3 days	2 weeks	3
Closed Caption [6] logic	Manual tests	2 days	1 week	7

**Table 3: Formal verification results in other projects**

We have experienced that the amount of time required for building a formal environment and rules is getting shorter as we worked on the various projects. It was also proven that there are blocks which can be verified only by formal verification, thus saving additional effort to build a simulation environment. The good results we observed, led us to assimilate formal verification in all our projects as an integral and mandatory part of the verification methodology. It was also affected the way our specification is written with greater consideration to formality and simplification of rules and logic descriptions. Furthermore, our design methodology includes now design for verifiability section which follow the guidelines provided in this section.

## 6.3 Implications on our Industry

Zoran is a small-mid size company which concentrates on delivering high quality products to the consumer market along (and more importantly) with delivering them in a very short time to market. A typical design cycle from an idea, through PRD (Project Required Document), Architectural specification, micro-Architecture specification, logic design, logic verification, circuit and layout to engineering samples takes about 12-18 months (for chip size of 6M and above transistors without memories). The verification methodology shares both design verification (pre-silicon verification activities) and validation (post-silicon verification activities). An exhaustive design verification (based on simulations) can provide a cleaner chip but it might caused us to

lose the short window of opportunity to launch the product to the market. Formal verification methods offer not only larger coverage of the verified properties but also better design methodology (design-for-formal verifiability). It was also found that our experienced designers and architects leverage their specification and design capabilities by utilizing formal methods.

## **7. Acknowledgments**

I would like to thank Uzi Zangi from Zoran and Tali Yatzkar-Haham from IBM Haifa, whose cooperation contributed to the work presented here. I also thank Yaron Wolfsthal from IBM Haifa, Aharon Aharon and Ram Ofir from Zoran for supporting this work.

## **References**

- [1] I. Beer, S. Ben-David, C. Eisner, A. Landver, "RuleBase: An Industry-Oriented Formal Verification Tool", Proceedings of the design Automation Conference, DAC'96.
- [2] GENERIC CODING OF MOVING PICTURES AND ASSOCIATED AUDIO INFORMATION: VIDEO Recommendation ITU-T H.262 ISO/IEC 13818-2 (MPEG specification)
- [3] K. McMillan, " SMV - Symbolic Model Checking", Kluwer Academic Publishers, 1993.
- [4] DVD specification, version 1.0 August 1996
- [5] Video Interface Port specification - VIP version 1.1, march 1997
- [6] Closed Caption specification - EIA-608, October 1989