A Parallel/Serial Trade-Off Methodology for Look-Up Table Based Decoders

Claus Schneider

Siemens AG, Corporate Technology, ZT ME 5
D-81730 Munich
E-Mail: Claus.Schneider@mchp.siemens.de

Abstract

A methodology for architecture exploration of look-up table based decoders is presented. For the degree of parallel processing a trade-off can be made by exploring system level and register transfer level models. Executable specifications (pure functional software models, VHDL behavior models) are used to analyze the performance of different architectures. Hardware cost (area) and feasibility (timing) are determined by synthesis of RTL models. These models are generated directly out of the specification to avoid errors due to manual transformations and to reduce overall design time. Generator-based reuse modeling and hardware cost estimation is demonstrated using a decoder for MPEG variable length codes (VLC).

1 Introduction

Decoders are often specified by look-up tables, in which the codes can have fixed or variable length. For image compression, entropy coding with variable length codes (VLC) is used to reduce redundancy and thereby data transmission rates. Many video standards (e.g. MPEG-1/2, JPEG) use this coding method. Depending on the coded bit rate, ranging from some Kb/s for video phone to some Mb/s for HDTV, the decoder architecture has to be adapted.

A parallel/serial trade-off for the processing bit width of the decoder has to be made for each application. For the design space exploration, information about the required performance as well as information about the hardware costs that arise to achieve this performance is needed.

If a component is laid out for peak data rates, no data buffers are needed, because the component can process the data on demand. On the other hand, the component can be designed for average data rates only and therefore data buffers are needed to deal with peaks.

Executable specifications in form of C software models, which are often publicly available in the field of multimedia, are used to calculate data distributions and average data rates. Then, buffer sizes can be adjusted by simulation of VHDL behavioral models. Finally, for the architecture selection hardware costs have to be estimated.

The approach proposed in this paper is based on an external model generator for look-up tables and RTL synthesis to determine hardware costs for the whole parameter range using an industrial example, a decoder for VLCs.

DAC 97, Anaheim, California

1.1 Related Work

A survey of VLC architectures can be found in [Fogg94]. Assessment criteria like bit rate vs. symbolic rate and pre-/post-buffering are discussed for serial and parallel architectures. But to apply these assessment criteria, information from the system level as well as from RTL is needed.

Petri Nets [AbCo90, MüKr93] and stochastic system models [HuTo90] are often used for early system level exploration. Other approaches capture the design in specification languages [TAB95], that are not part of the design process at all. The method proposed here uses publicly available C software models and VHDL behavioral models, that are part of the regular design process to reduce the modelling effort for performance exploration.

At RT level, generator-based methods for design space exploration of regular structured logic (e.g. multipliers) are proposed in [JhDu92, GWG93]. Other approaches are based on counting literals, for example [BRSW87]. The approach presented in this paper is novel, because the RTL models for irregular structured LUTs are generated directly out of the specification to avoid errors due to manual transformations and to reduce overall design time. In addition to that, the tables are optimized in the generator to provide more accurate estimations and to reduce synthesis run time.

1.2 Paper Structure

First, an overview of the architecture exploration process is given. The system level exploration is described using data distributions, generated by pure functional software models to stimulate the behavioral simulation. At RT level the limitations of self-generating models are discussed. Then, the usage of an external model generator is presented. Finally, the results of the exploration process are summarized.

2 Overview

2.1 Decoding of Variable Length Codes

The MPEG video standard uses variable length codes, in which shorter codewords are assigned to more frequent symbols (see Figure 6). The boundaries of the codewords can not be determined explicitly by the decoder. After a codeword and its length are decoded, the bitstream has to be shifted, before the decoder can start processing the next codeword (see Figure 5).

Variable length codes can be decoded either serial (one bit per clock cycle, see Figure 7) or parallel (multiple bits per clocks cycle, see Figure 8). If the processing width of the decoder is smaller than the maximum code length a finite state machine is needed. Otherwise the decoder consists only of combinatorial logic. The hardware cost (area) of the whole decoder increases with the degree of parallel processing.

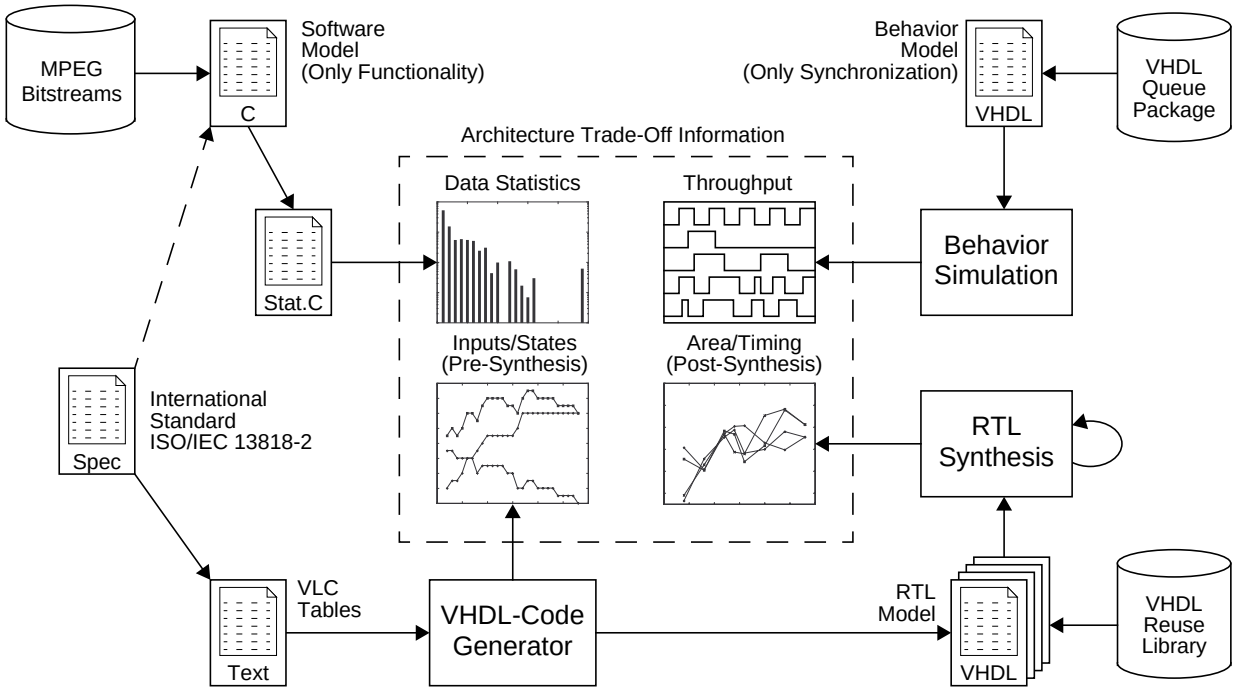

Figure 1: Parallel/Serial Trade-off: Overview

2.2 Parallel/Serial Trade-Off Methodology

For the parallel/serial trade-off of a variable length decoder, top-down information from the system level as well as bottom-up information from the RT level is needed. At system level, the design space is examined with regard to data rates and throughput for different degrees of parallelism. At RT level, hardware cost (circuit area) and feasibility (propagation delay) are calculated to support the architecture selection (see Figure 1).

Architectural explorations are performed using four different kinds of models:

- Execution of C software
- Simulation of VHDL at behavior level
- Analysis of VLC tables in text form
- Synthesis of VHDL at register transfer level

For system level exploration, executable specifications are used to get the performance information. A pure functional software model of the whole MPEG video decoder, which is publicly available [MSSG] in addition to the printed specification [MPEG], is extended to generate data statistics. Average data rates and data distributions for selected interfaces are calculated, running several MPEG video sequences. Then, the data distribution is used to stimulate the VHDL behavior model to analyze the synchronization and to perform a buffer-size/operation-schedule tradeoff.

At RT level, synthesizable VHDL code is generated to determine the circuit area and propagation delay. First, the variable length code (VLC) tables are extracted from an electronic version of the MPEG specification and written as a plain text file. After that, a generator for look-up tables reads the text file with the VLC tables and generates the VHDL code for all architectures (processing widths) in a specified range. During the code generation, informations like the number of states are collected to assist the parallel/serial trade-off process. Using these pre-synthesis analysis results, promising architectures can be selected for the time consuming synthesis task. By analyzing the pre-synthesis and post-synthesis results a close relationship between the number of states and the area of the next-state logic, which is independent of the state coding (binary or onehot), was discovered. In addition to that, the area of the output logic of the generated state machines can be estimated roughly by the number of relevant input bits [Schn97].

3 System Level Exploration

At system level, MPEG bitstreams are analyzed and statistics about data rates are generated by the C software model.

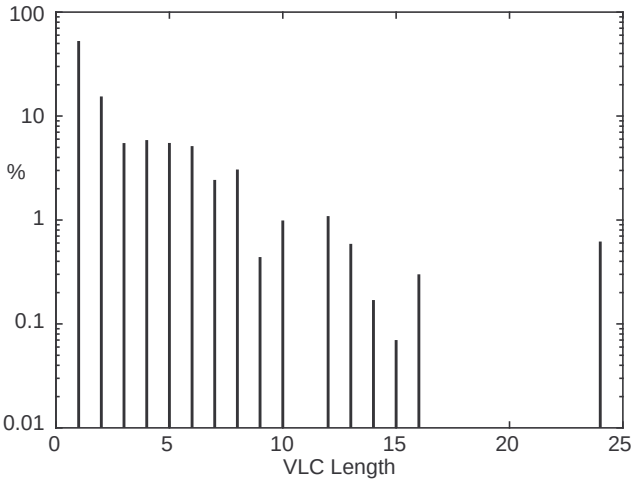

Figure 2: Distribution of VLC Lengths (MPEG-2)

Therefore the original model of the MPEG decoder was extended by statistic functions. The number of VLCs was accumulated in the statistic module for each possible length value and the average length of all VLCs was calculated as well.

The statistic in Figure 2, generated by playing several MPEG-2 video sequences, shows the distribution of the code length of variable length codes. More than 50% of all VLCs have a length of only one bit and the average VLC length is about 3 bits. Nevertheless, peak data rates of 24 bits per coded symbol are possible as well.

For a decoder, that is laid out for average data rates, a processing width of 3 bits could be chosen. But for this low performance architecture, large data buffers may be needed around the decoder to guarantee average data rates. On the other hand a fully parallel (24 bit) architecture can decode the VLCs at a constant symbol output rate and therefore only needs small or even no data buffers at all.

For the buffer-size/operation-performance trade-off the distribution of the VLC lengths can be used to stimulate the behavior level simulation model as shown in Figure 3.

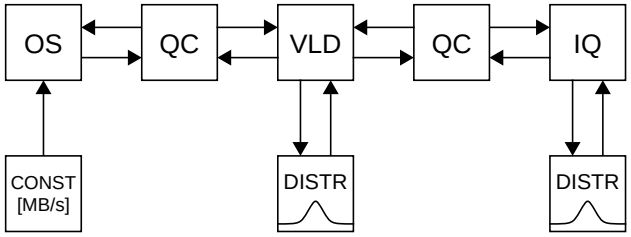

Figure 3: Performance Model

In [ScEc96] a methodology for behavioral modeling, which consists of separation of synchronization and functionality, was presented. The benefit of this behavioral modeling approach is, that controller and datapath can be modeled and verified independently. For performance modeling only the controller has to be modelled and the datapath can be omitted. In Figure 3 only a part of the performance model is shown. The variable length decoder operation (VLD) is decoupled by data buffers, which are modelled by queue controllers (QC). On one side the source data buffer is filled with a constant data rate, at the other side the output buffer is emptied by the inverse quantization operation (IQ). Data dependent processing times of the VLD and IQ operations are modelled by distribution functions, calculated by the software model.

More details about the architecture trade-off at system level can be found in [Schn97]. In the following section a generator-based approach for reuse modeling and hardware cost estimation at RT level is presented.

4 RT level Exploration

At RT level circuit area and timing information is calculated to support the architecture selection process.

On one hand for regular hardware structures, like adders and multipliers, this information can be estimated quickly using a formula or a table.

On the other hand for irregular hardware structures, rough estimations can be made by analyzing the VHDL code pre-synthesis and by counting, for example, the number of states and inputs. Better cost figures (e.g. area, timing) can be obtained by performing a more or less precise (time consuming) synthesis process.

Not only hardware cost estimation is different for regular and irregular hardware structures, but different modeling techniques are required as well. In the next sections, the limitations of self-generating models due to the synthesis subset will be discussed and an external generator-based approach will be presented to work around these limitations.

4.1 Limitations of Self-Generating Models

Self-generating VHDL models for synthesis are well suited for regular hardware structures. In most of the cases only the bit width of a component is controlled by a generic parameter. Inside the model, the hardware structure can be built using generate statements, for example [PrMR96].

For more complex and irregular hardware structures, VHDL constructs like text I/O, access types and record constants are needed for efficient modeling. Unfortunately these language features are not supported by current synthesis tools. Text I/O, for example, should not be synthesized, but it is very useful to read the whole definition of a look-up table for a decoder component from a parameter file. During the model generation of irregular hardware structures, access types are very useful as well to dynamically build intermediate data structures. Finally, text I/O is very important for writing reports and revision control information.

Despite these limitations, a self-generating VHDL model was implemented first. The look-up table was defined as an array of record inside a package. Because record constants are not supported by synthesis, the record elements of each table entry (te, see Figure 4) are passed to a function that returns a record.

```
constant vlc_max_len_c : integer := 32;
subtype vlc_vector_t is
     std_logic_vector(0 to vlc_max_len_c-1);

type vlc_rec_t is record
  vlc:          vlc_vector_t;
  length:       integer;
  symbol:       integer;
  .....
end record;

type vlc_tab_t is
     array(natural range <>) of vlc_rec_t;

function te (vlc: string; symbol: integer)
            return vlc_rec_t is
  variable vr: vlc_rec_t;
begin
  get_vlc(vlc, vr.vlc, vr.length);
  vr.symbol := symbol;
  return vr;
end te;

constant vlc_tab_10_c: vlc_lut_t:= (
.....
te("0000_0011_001",   -16),
te("0000_0011_011",   -15),
te("0000_0011_101",   -14),
.....
te("0000_0011_000",    16)
);
```

Figure 4: Self-Generating Model

For the transformation of the pure combinational look-up table to a finite state machine with a given processing width several nested loops are needed to calculate a constant for the output logic and for the next-state logic. During the elaboration the synthesis tools interpret these calculations as hardware and perform loop unrolling. This leads to long elaboration run times and high memory requirements.

4.2 External Model Generator

Because of the long elaboration run times, which are not acceptable for architecture explorations, a mixture of self-generating and externally generated models was chosen for the final implementation. Regular structures like barrel shifters and FIFOs are used from a VHDL reuse library. Irregular structures like look-up tables are generated externally using the scripting language Perl [WaSc92, PERL, HuHe96]. Perl supports regular expression matching, dynamic lists and associative memories, which are useful to read parameters, generate the model and to write reports. Another advantage is, that code can be generated using a small VHDL subset without synthesis tool dependent compiler directives or non-portable modeling tricks.

4.2.1 Basic Decoder Architecture

The decoder consists of a barrel shifter (reuse component), that shifts the VLC by the length, detected by the length look-up table (LUT). If the processing width is smaller than the maximum code width, the table look-up can't be performed in one step and therefore a state register and next-state logic is needed as well (see Figure 5). The critical timing path (thick arrows) goes through the length LUT. Therefore the length LUT is separated from the symbol LUT and the next-state logic.

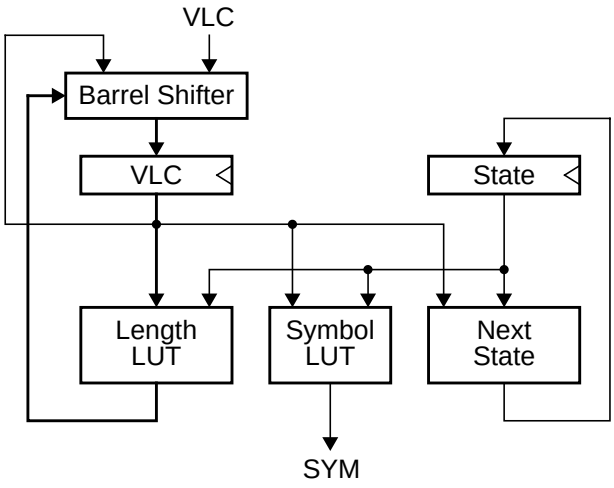

Figure 5: Basic Decoder Architecture

4.2.2 Generator Structure

The model generation process is demonstrated for the length LUT and next-state logic of the decoder. The MPEG standard consists of several tables of variable length codes. One of these tables (see Figure 6) was chosen to show the transformations, optimizations and the code generation done by the generator.

The generator performs the following tasks:

- Read table and initialize internal data structure.
- Split VLC table according to given processing width.
- Extract length-, symbol LUT and next-state logic.
- Generate code (VHDL package) of non-optimized look-up tables for different state encodings.
- Logic minimization and mapping of don't care values of length LUT.
- Code generation of optimized look-up tables.

VLC	SYM
0000 0011 001	-16
0000 0011 011	-15
0000 0011 101	-14
0000 0011 111	-13
0000 0100 001	-12
0000 0100 011	-11
0000 0100 11	-10
0000 0101 01	-9
0000 0101 11	-8
0000 0111	-7
0000 1001	-6
0000 1011	-5
0000 111	-4
0001 1	-3
0011	-2
011	-1
1	0

continued	
010	1
0010	2
0001 0	3
0000 110	4
0000 1010	5
0000 1000	6
0000 0110	7
0000 0101 10	8
0000 0101 00	9
0000 0100 10	10
0000 0100 010	11
0000 0100 000	12
0000 0011 110	13
0000 0011 100	14
0000 0011 010	15
0000 0011 000	16

Figure 6: Original Table from MPEG Specification

The generator is parameterized by the name of the VLC table definition file and the processing width. After the table is read, it is transformed to the given processing width.

One extreme is serial processing, where only one bit is decoded in each step (clock cycle). In Figure 7 the decoding process is shown for the VLC table of Figure 6. The number of decoding steps depends strongly on the length of the VLC. The symbol '0' can be decoded in one step and the symbol '16' in 11 steps, for example.

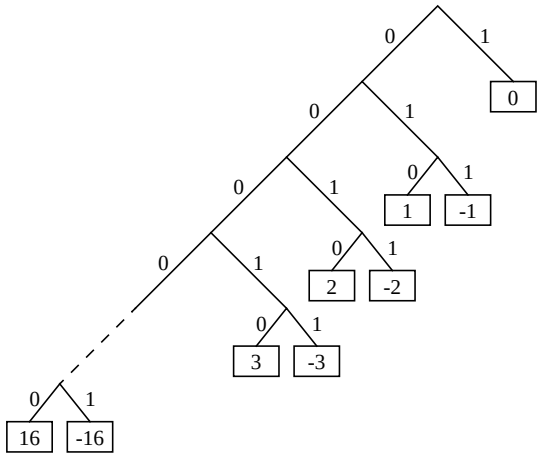

Figure 7: Serial Processing (1 Bit)

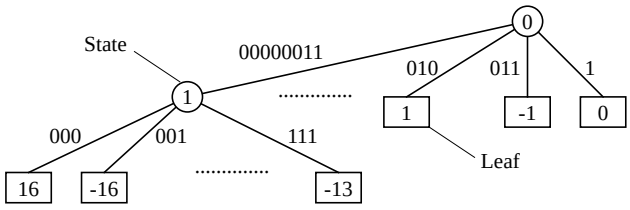

Figure 8: Parallel Processing (8 Bit)

The one bit serial architecture is characterized by a constant input data rate. A fully parallel architecture with a processing with of the maximum code length, on the other hand is characterized by a constant output data rate. An architecture variant between these extremes, with a processing width of 8 bits is shown in Figure 8.

The most important task of the generator for look-up tables is to split the table according to the processing width given by a parameter. A state is assigned to each branch (e.g. "00000011") in the code tree. For different VLC tables, equal states are assigned to equal input vectors to reduce the overall number of states. If a leaf of the code tree is reached, a symbol is decoded and the state can be reset to zero. The split table for an 8 bit processing width is shown in Figure 9.

S	VLC	N	L	Y
0	00000011	1	8	
1	001	0	3	-16
1	011	0	3	-15
1	101	0	3	-14
1	111	0	3	-13
0	00000100	2	8	
2	001	0	3	-12
2	011	0	3	-11
2	11	0	2	-10
0	00000101	3	8	
3	01	0	2	-9
3	11	0	2	-8
0	00000111	0	8	-7
0	00001001	0	8	-6
0	00001011	0	8	-5
0	0000111	0	7	-4
0	00011	0	5	-3
0	0011	0	4	-2
0	011	0	3	-1
0	1	0	1	0

continued				
0	010	0	3	1
0	0010	0	4	2
0	00010	0	5	3
0	0000110	0	7	4
0	00001010	0	8	5
0	00001000	0	8	6
0	00000110	0	8	7
3	10	0	2	8
3	00	0	2	9
2	10	0	2	10
2	010	0	3	11
2	000	0	3	12
1	110	0	3	13
1	100	0	3	14
1	010	0	3	15
1	000	0	3	16

S = State
N = Next State
L = Length
Y = Symbol

Figure 9: Table after Splitting (Width = 8 Bit)

The symbol table, which includes the error handling for invalid codes, is generated separately. Due to this separation, logic optimization and don't care mapping the length and next-state tables can be minimized (see Figure 10).

State	VLC	Length
0	00000	8
0	000010	8
0	000011	7
0	0001	5
0	001	4
0	01	3
0	1	1
1	-	3
2	0	3
2	1	2
3	-	2

State	VLC	Next
0	00000011	1
0	00000100	2
0	00000101	3

Figure 10: Length and Next-State Table after Optimization

Finally, the VHDL code for the non-optimized and optimized look-up tables is generated for different state encodings. For each architecture a separate package is written. The VHDL model for the whole variable length decoder, consisting of components like barrel shifters, registers and FIFOs with variable bit widths, is parameterized by the generated packages. The VHDL code, generated for the length table from Figure 10 is shown in Figure 11.

```
function vlc_len_lut_f(
                table : table_int_t;
              mpeg_2 : std_ulogic;
                  dc : std_ulogic;
               state : state_vec_t;
                 vlc : vlc_vec_t )
         return length_vec_t is
  variable len : length_vec_t;
begin
len:=(others => '-');
-- NOTE: vlc_width is a power of 2
-- -> len is binary coded as vlc_length - 1
case state is
 when s_0 =>
    if vlc(0 to 4)="00000" then len:="111";
 elsif vlc(0 to 5)="000010" then len:="111";
 elsif vlc(0 to 5)="000011" then len:="110";
 elsif vlc(0 to 3)="0001" then len:="100";
 elsif vlc(0 to 2)="001" then len:="011";
 elsif vlc(0 to 1)="01" then len:="010";
 elsif vlc(0)='1' then len:="000";
 end if;
 when s_1 =>
  len:="010";
 when s_2 =>
    if vlc(0)='0' then len:="010";
 elsif vlc(0)='1' then len:="001";
 end if;
 when s_3 =>
  len:="001";
 when others => null;
end case;
return len;
end vlc_len_lut_f;
```

Figure 11: VHDL Code of Length LUT

4.3 Results

The whole decoder for variable length codes was synthesized for several processing widths, different state encodings (binary, onehot) and for the optimized (opt) and non-optimized (nop) look-up tables.

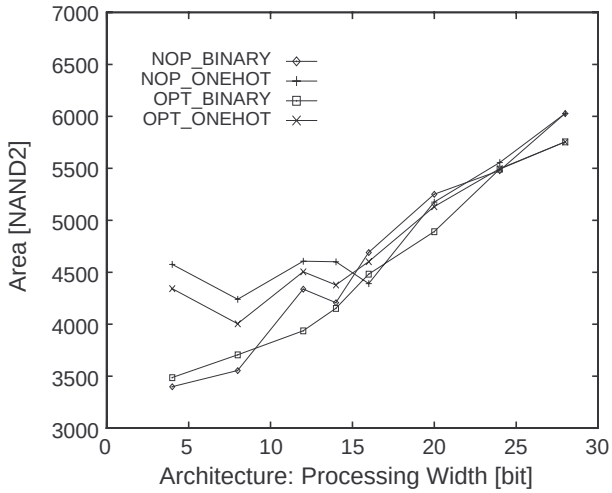

Figure 12: Area of VLC Decoder

In Figure 12 the area (2 input NAND equivalent gate count), which is dominated by the increasing bit width of the regular structures like registers, barrel shifters and FIFOs, is shown.

The timing (propagation delay) of the critical path is influenced by the delay of the length LUT, as shown in Figure 13.

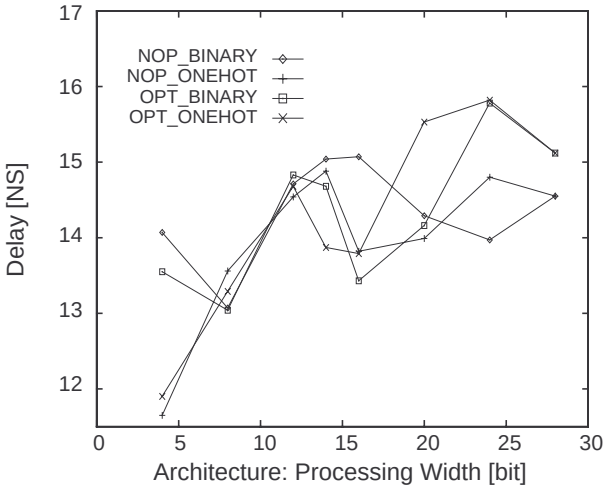

Figure 13: Timing of VLC Decoder

After collecting performance and hardware cost information, the parallel/serial trade-off can be performed. Depending on the required performance (bit/symbol rate) an architecture with minimum area (decoder and buffers) can be selected, that meets the timing constraints given by the clock frequency.

5 Conclusion and Outlook

A methodology for the parallel/serial trade-off of look-up table based decoders was presented. The architecture exploration at system level is performed with executable specifications. Based on data distribution functions, generated by pure functional software models, the synchronization is analyzed using behavior models. Especially in multimedia, where executable specifications are often publicly available as C software models, the proposed approach allows for early system level architecture exploration with low additional modeling effort.

At RT level, area and timing information is obtained by synthesis of parameterizable models. For regular hardware structures like barrel shifters, self-generating models are reused from a VHDL library. Because of the restricted synthesis subset, an external model generator is used for architecture trade-off of the irregular structured look-up tables.

During the elaboration, all VHDL constructs should be supported by the synthesis tools to improve modeling for reuse and architecture trade-off. Another problem is loop unrolling during the elaboration, which causes long synthesis run times and needs a huge amount of memory.

The self-generating model of a look-up table with less than 1K gates needed more than 4 hours (Sparc20, 1GB swap) for synthesis. But the externally generated and optimized model was synthesized in about 30 minutes only.

In addition to the reduced synthesis run time, due to the pre-optimized VHDL code, another advantage of the generator-based approach is, that the synthesis models can be generated directly out of the specification to avoid errors due to manual transformations.

References

[AbCo90] Aboulhamid, M.; Cordeau, M.: *System Level Modeling in VHDL using Timed Petri Nets*. Proceedings of the EURO-VHDL'90.

[BRSW87] Brayton, R.K.; Rudell, R.; Sangiovanni-Vincentelli A.; Wang, A.R.: *MIS: A Multiple-Level Logic Optimization System*. IEEE Transactions on CAD, Nov. 1987.

[Fogg94] Fogg, C.: Survey of software and hardware VLC architectures. Proceedings of the SPIE - The International Society for Optical Engineering, vol.2186, p.29-37, 1994.

[GWG93] Gasteier, M.; Wehn, N.; Glesner, M.: *Synthesis of Complex VHDL operators*. Proceedings of the EURO-VHDL'93.

[HuHe96] Hutchins, R.C.; Hemmady S.: *How to Write Awk and Perl Scripts to Enable your EDA Tools to Work Together*. Proceedings of the 33rd DAC, 1996.

[HuTo90] Hubbard, P.; Torres, J.: *Using VHDL for High-Level and stochastic System Modeling*. Proceedings of the EURO-VHDL'90.

[JhDu92] Jha, P.K.; Dutt, N.D.: *Rapid Estimation for Parameterized Components in High-Level Synthesis*. Sixth International Workshop on High Level Synthesis, 1992.

[MPEG] *Information Technology - Generic Coding of Moving Pictures and Associated Audio Information: Video*. Recommendation ITU-T H.262 (MPEG 2), International Standard ISO/IEC 13818-2: 1995.

[MSSG] MPEG Software Simulation Group: *MPEG-2 Encoder/Decoder.* http://www.mpeg.org/MSSG/.

[MüKr93] Müller, J.; Krämer, H.: *Analysis of Multi-Process VHDL Specifications with a Petri net Model*. Proceedings of the EURO-VHDL'93.

[PERL] *The Perl Language Home Page*. http://www.perl.com/perl/index.html

[PrMR96] Preis,V.; März-Rössel,S.: *Modeling highly flexible and self-generating parameterizable components in VHDL*. Current Issues in Electronic Modeling #5, Kluwer Academic Publishers, 1996.

[ScEc96] Schneider, C.; Ecker, W.: *Stepwise Refinement of Behavioral VHDL Specifications by Separation of Synchronization and Functionality.* Proceedings of the EURO-VHDL'96.

[Schn97] Schneider, C.: *A Methodology for Hardware Architecture Trade-off at Different Levels of Abstraction*. Proceedings of the ED&TC, 1997.

[TAB95] Tanir, O; Agarwal, V.K.; Bhatt, P.C.P.: *A Specification-Driven Architectural Design Environment*. Computer Vol. 28, No. 6, 1995.

[WaSc92] Wall, R., Schwartz R.L.: *Programming PERL*. O'Reilly & Associates, 1992.