Journal of Theoretical and Applied Information Technology
31st October 2011. Vol. 32 No.2
© 2005 - 2011 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

HARDWARE BASED BINARY ARITHMETIC ENGINE

[1]N. SARANYA, [2] DHIRENDRA KUMAR TRIPATHI, [3]DR. R. MUTHAIAH

[1]Assiciate System Engineer, IBM India Pvt., Ltd.,, Banglore, India-560066
[2]Asst. Prof., Department of Information and Technology, SASTRA University, Thanjavur, India-613401
[3]Assoc. Prof., Department of Information and Technology, SASTRA University, Thanjavur, India-613401
E-mail: saran11sastra@gmail.com , dkt@core.sastra.edu , sjamuthaiah@core.sastra.edu

ABSTRACT

Context Based Binary Arithmetic Coding (CBAC) is a part of JZ profile of Audio Video Coding Standard (AVS).The goal of this paper is to present the efficient hardware based binary arithmetic coder which is the main part of binarisation involved in CBAC of AVS. This paper explains about the efficient arithmetic coding involved in the video transcoding. The major concerns of using JZ profile of AVS is movie compression for high-density storage and relatively higher computational complexity can be tolerated at the encoder side to provide higher video quality. This arithmetic coder avoids the slow multiplication, here the traditional arithmetic calculation is transformed to software domain using log. The proposed arithmetic engine will give high compression gain and an efficient coding design for handling the high resolution video sequence. This is used for the effective pipelining process and increase the overall processing speed. The implementation of arithmetic coder in CABAC (Context Adaptive Binary Arithmetic Coding) architecture is satisfying the CBAC feature. The study of this paper is required to know about the process of efficient AC(arithmetic coding) and the hardware based arithmetic engine in CBAC. The CBAC engine is implemented on Xilinx Virtex-5 ML501 Board. The synthesis and simulation results are presented. The proposed architecture can work with the 31.384MHz rate.

Keywords: AC,CBAC,CABAC,AVS,Videotranscoding,Binarisation,Qcoder

1. INTRODUCTION

The different types of techniques are used for data compression applications and it has different degrees of complexity. This type of techniques uses some common process. This follows the common processes of Numerical Processing, Logical Processing, Source Modeling and Entropy Coding[1]. At first the original data is given as an input, then numerical processing takes place which will have the processes like predictive coding and linear transform for the images. Logical Processing will change the data to a suitable form for compression. The variation in the statistical properties of data is given by source modeling. It is responsible for the identified data contexts and the statistics gathered. This makes the source model accurate. The final process is entropy coding. This is the process of representation of image in compact form. In different entropy coding method, arithmetic coding places the best vide in effectiveness in compression and effectiveness.

CBAC in AVS belongs to Q coder family[2]. This belongs to the JZ profile of AVS[3].It offers an average of 13% bit saving compared with CABAC[4]. It adopts the simpler binarisation method and the context selection scheme. In addition to this, the traditional multiplication is transformed to the logarithmic domain. This produces the variation from CABAC.

2. CODING STAGES OF CABAC

Figure 1 shows the architecture for the CABAC architecture.
1. Before the process of arithmetic coding, a non-binary valued symbol is converted into binary valued symbol.
2. The binary valued symbol is encoded by the arithmetic coder before the transmission.
3. The probability model for the binary value of symbol is given as a context model.
4. Depending on the recently coded symbol the particular model is selected for coding process. It finally stores the values as “1” or “0”.
5. Then the arithmetic coder will encode the binary value of the selected model.
6. The selected context model is updated depending on the actual coded value.

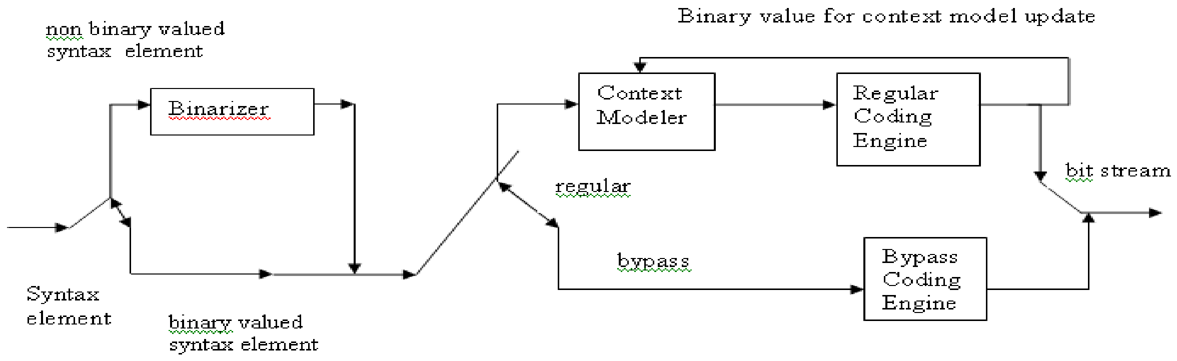


Figure 1: CABAC Architecture

3. PURPOSE OF ARITHMETIC CODING

Arithmetic coding generates the non-block codes when compared to the Variable Length Coding. In this coding process the single arithmetic code is assigned to the entire sequence of symbol. Rounding errors are produced by Huffman coding, hence it has the restriction in code length to multiples of a bit. This deviation due to error is much higher when compared to the inaccuracies of arithmetic coding.

4. PRINCIPLE OF ARITHMETIC CODING

The Principle of arithmetic coding depends on the recursive subdivision of intervals. Every binary symbol is assigned to the specific context model, which is updated during the whole coding process for the adaptive probability estimation. First the interval is subdivided in to two subintervals rLPS and rMPS. rLPS is the range value belongs to LPS and rMPS is the range belong to mps. PLPS is the probability estimation value of LPS.

By checking whether the given bin to be encoded is MPS or LPS, the specific subinterval is selected as the current new interval. This coding process continuously updates two registers (i.e) R is assign to the range of the interval and the code Register C is assign to the lower value of interval.

The arithmetic decoding process is the reverse process of encoding first the context model of bin value which is to be decoded is selected. Then the bin value is decoded by the arithmetic decoder engine. The context models are also continuously updated during the whole decoding process. This process is same as that of the encoding engine. In this decoder arithmetic engine the register range R which have the current interval and the register offset which have the position of input bit stream. If the binary value which is decoded as MPS of LPS is decided by the position of offset falls whether rLPS or rMPS range. These 2 registers are continuously updated.

5. PREVIOUS AC ENGINE

In CABAC[5], the arithmetic calculation of rLPS and probability estimation updates are estimated by using the look up table. This process requires large memory and it has less coding efficiency. The proposed method optimize the area versus speed processing in following way. In oreder to reduce the FPGA synthesized area at first a limited number of probability estimation units are implemented on hardware and they are repeatedly used. This saves area as compared to full parallel architecture for the probability estimation. The second optimization is selected is implementation of logarthimic engine using hardware multiplier. This makes logarithmic calculation much faster hence there will be overall increase in the speed CABAC engine.

6. STEPS OF AC ENGINE

1. Values of rLPS and rMPS subintervals are calculated.
2. Finite precision takes place during the whole coding process using renormalization.
3. Probability estimation is updated adaptively.

Step:1 rLPS and rMPS can be calculated by

$$rLPS = R * PLPS \quad (1)$$
$$rMPS = R - rLPS \quad (2)$$
or
$$rMPS = R * PMPS \quad (3)$$
$$rLPS = R - rMPS \quad (4)$$

rLPS is assign to upper interval and rMPS is assign to lower interval.

Step:2 At the time of encoding process, the register R is left shifted to make the MSB of the new interval R is always 1. At the time during decoding process the range and offset values of registers are left shifted to make the value always 1. Renormalization process has the leading 1 detection circuit and its responsible for the critical path delay.

Step:3 As the third step, the probability estimation of binary source takes place. Here P is the probability of symbol 1 and q is the probability of symbol 0 and for the adjusting the adaptation speed parameter N is used, then here pk & qk are the probability estimation of 1 and 0 after the certain K events, the probability estimation after K+1 event is calculated b]

$p_{k+1}=(N*p_k) / N+1$ (if 0 occurs) (5)

$q_{k+1}=(N*q_k) / N+1$ (if 1 occurs) (6)

If PLPS & PMPS are used for marking probability of LPS and MPS, then can be written as

(if LPS occurs)
PMPS _ new = f *(PMPS_old) (7)

(if MPS occurs)
PLPS_new = f *(PLPS_old) (8)

7. HARDWARE BASED BINARY ARITHMETIC ENGINE

The Logarithm is used in CBAC of AVS to transform the slow multiplication into addition and subtraction. This hardware based Arithmetic Engine is designed by using this steps.
The range value of logarithmic domain is needs to be updated using the subtraction when most probable symbol (MPS) occurs. The range of subinterval is given as range and the offset position is given as offset [6][7].

MPS

Rang new = Range * pMPS (9)

Offset new = Offset (10)

By using log the above equation is transformed to

Lg_Rangnew=Lg_Range –Lg_PMPS (11)
Offsetnew = offset (12)

Here PMPS is the probability of MPS and Lg-x is log value of x.

The value of logarithm domain is transformed to the original domain when least probable symbol occurs[1]

LPS

Range new = Range - (PMPS * Range)
= Range - rMPS (13)

Offset new = Offset + (Range*PMPS)
= Offset + rMPS (14)

rMPS gives the subinterval size associated to MPS.

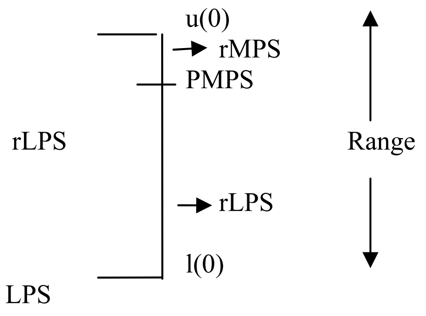


Figure 2: Range Interval

Then the Probability adaptive estimation process is given by the equation[5]
Initial Value: l(0)= 0 , u(0)= 1

l(0) is the initial value of LPS (ie) the least value of lower interval and u(0) is the initial value of MPS (ie) the greatest value of upper interval. This l(0) and u(0) values are updated adaptively using the formula given below
Interval Update:
$l(n)= l(n-1)+[u(n-1)-l(n-1)]F_x(x(n-1))$ (15)

$u(n)= l(n-1)+[u(n-1)-l(n-1)]F_x(x(n))$ (16)

During the process of renormalization, the range value and offset value are left shifted to make MSB to be 1. The difference in the number of bits shifted between range and offset values are stored. This is done for the consecutive decoding cycles.

8. IMPLEMENTATION AND RESULTS

This proposed Arithmetic Engine is simulated and synthesized using XILINX 12.1 ISE tool[8]. This is implemented using VHDL language. For implementing Xilinx Virtex-5 ML501 Board is used[9].This arithmetic engine operates at the frequency of 31.834MHz.The time latency required by the output after the clock is 7.430ns.

In order to synthesize the CABAC engine fixed point binary number system is adopted. In this eight bit binary system is used. In this case for example .45 is converted into the .45X255≈116. The most critical part of the CABAC engine is logarithmic engine. In literature there are two popular way are given to implement log engine in the hardware . The first one is CORDIC based approach another is the series approximation [10]. Though the CORDIC based approach can be implemented with the less FPGA resources however the major limitation is it takes more time to calculate a Log value which will be serious limitation in real time processing. The second approach require more multipliers however it is very fast as compared to CORDIC. Since FPGA has many hardware multipliers available in its fabric, hence second approach is chosen for hardware implementation of logarithmic engine.Using arithmetic engine the interval range of upper limit and lower limit of input is retrieved as an output .This interval updation takes place by analyzing whether the input lies in LPS or MPS range. The post layout simulation results are shown in figure 3 to figure 9. Figure 3 shows the log Value of the given 8 bit number. Figure 4 shows that given input lies in the range of mps .Figure 5 and 6 shows further updation of the both lower and upper interval of MPS. Figure 7 shows that given input lies in the range of LPS. Figure 8and 9 shows further updation of the both lower and upper interval of LPS.

Table 1. DEVICE UTILISATION SUMMARY

LOGIC UTILISATION	USED	AVAILABLE	%USED
Slice Registers	212	13312	1%
4 I/P LUTs	373	26624	1%
FF pairs	140	26624	0%
Bonded IOBs	61	487	12%
No. of Multipliers	13	32	40%
Gclk	1	8	12%

9. CONCLUSION

In this paper, we present the hardware based Arithmetic Engine which combined both original and logarithmic domain. This improves the coding efficiency and increases the overall processing speed. The device utilization summary(table 1) gives the details about the efficient use of the devices. From the whole block of image, the sub block of image which is specified can be efficiently found out by using the adaptive updation in hardware based binary arithmetic engine .This is useful to get the higher resolution image.

In future this Arithmetic Engine can be designed with the software and hardware co-design approach. Further research about AC is currently underway mainly targeting towards two directions. 1. To build more accurate statistical models for source, which is important for coding gain. 2. To explore cost-effective VLSI architectures of AC codec. This arithmetic coding will be further developed to overcome the challenge in process of multi bin per cycle for the demand of encoding and decoding of the high resolution picture. The use of this hardware based binary arithmetic engine will further increases the video quality with high resolution.

REFERENCES

[1]. Amir Said, "*Introduction to arithmetic Coding and Practice,Imaging Systess*" Laboratory HP Laboratories Palo Alto,HPL-2004-76, April 21, 2004.

[2]. Study and comparison of H.264/MPEG4 part10 AVC main profile with AVS P2 Jizhun profile p-p-t.

[3]. Arithmetic coding for data Compression Ian H.Witten, Radford M.Neal and John G.Cleary Junhao Zheng, WenGa DavidWu , DonXie "*An efficient VLSI architecture for CBAC of AVS HDTV decoder*".

[4]. W.B.Pennebaker,J.L.Mitchell, "*An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder*", IBM Journal of Research and Development 32(1988)717–726.

[5]. W.Yu,Y.He, *Arithmetic Codec on Logarithm Domain*, Picture Coding Symposium,2006

[6]. .D.Marpe, H.Schwarz, T.Wiegand, "*Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard*", IEEE Transaction Circuits and Systems for Video Technology 13 (2003) 620–636.

[7]. Xilinx:" Virtex-5 Family overview" DS100(v5.0) February 6, 2009.

[8]. Xilinx:" ISIM User Guide" UG660(v11.3) September 16, 2009.

[9]. Uwe Meyer- Baese , "*Digital Signal Processing with Field Programmable Gate Arrays*",Springer,2007

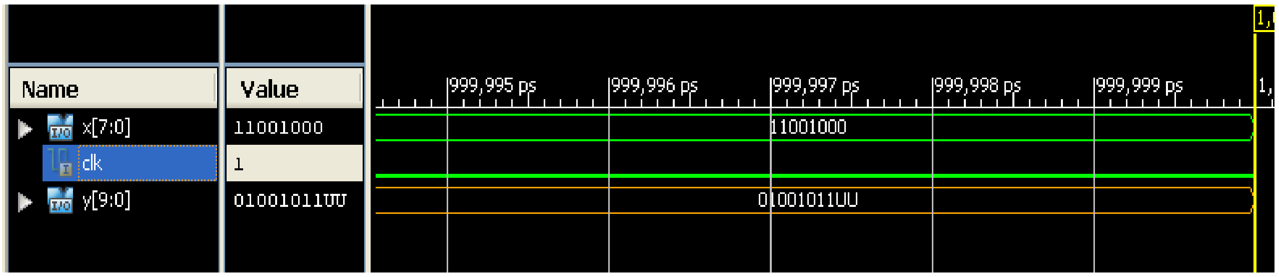
[10].

AUTHOR PROFILES:

N. SARANAYA received the B.E. degree in electronics & communication engineering. She received her M.Tech in VLSI Design form SASTRA University, Thanjavur, India. Currently she is with the Tata Consulatncy Services as embedded system designer .Her interests include VLSI based Arithmetic Coding, Image processing and embedded system.

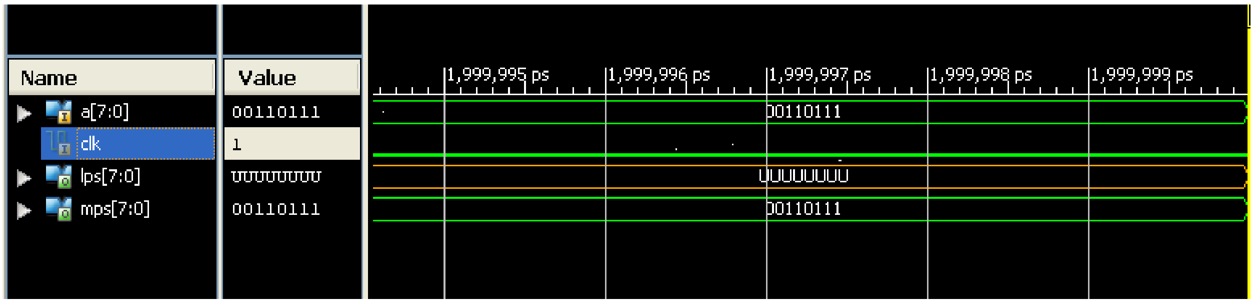
DHIRENDRA KUMAR TRIPATHI received the B.E. in electronics & communication engineering from UPTU, in 2005. He completed his M.Tech. in VLSI at NIT Trichy,INDIA. Currently, he is an Assistant Professor at SASTRA University,Thanjavur,India. His interests are in MIMO, SDR and Coginitive Radio.

Dr. R. MUTHAIAH received the B.E. degree in electronics & instrumentation engineering from Annamalai University, in 1989. He received M.E Power electronics & industrial drives. He received the Ph.D. degree in Image processing from SASTRA University. Currently, he is a Associate professor at School of computing in SASTRA University, Thanjavur, India. His research interests include VLSI based Image processing and Signal processing.



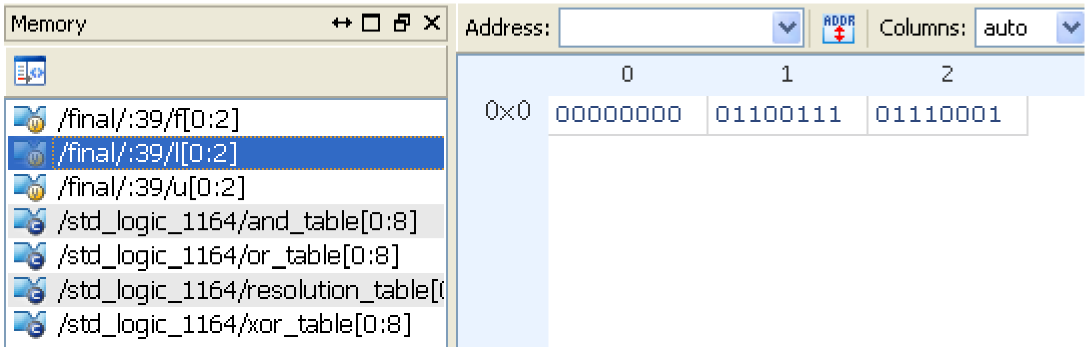
```
x[7:0]   11001000
clk      1
y[9:0]   01001011UU
```

FIGURE 3 LOG VALUE OF THE INPUT



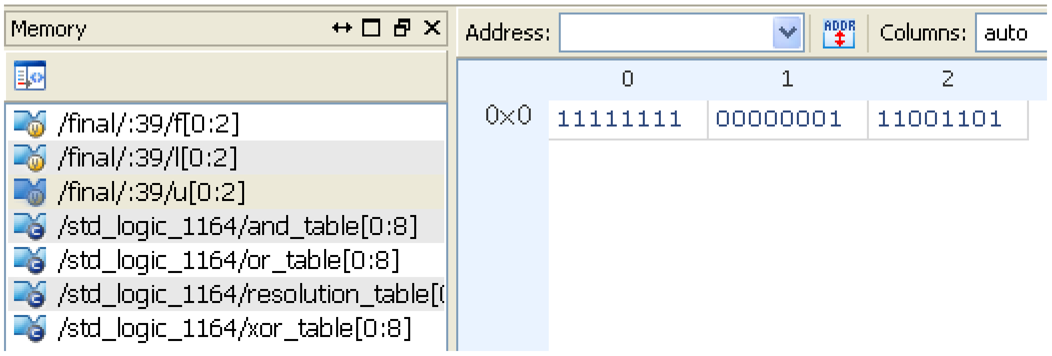
```
a[7:0]     00110111
clk        1
lps[7:0]   UUUUUUUU
mps[7:0]   00110111
```

FIGURE 4 INPUT LIES IN RANGE OF MPS



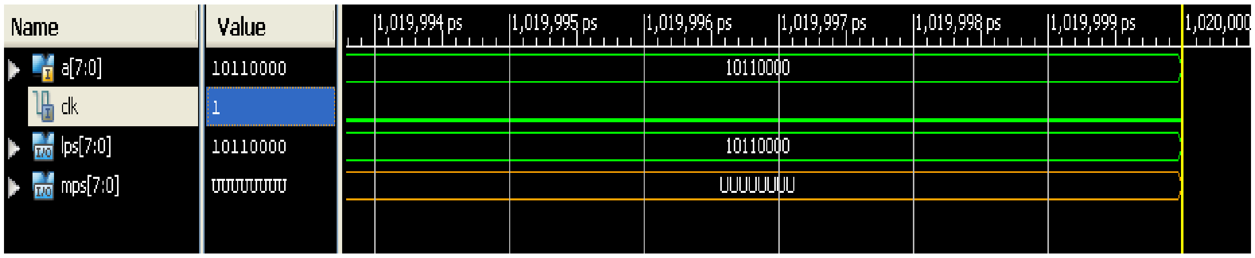
```
/final/:39/f[0:2]
/final/:39/l[0:2]
/final/:39/u[0:2]
/std_logic_1164/and_table[0:8]
/std_logic_1164/or_table[0:8]
/std_logic_1164/resolution_table[
/std_logic_1164/xor_table[0:8]

        0          1          2
0x0  00000000   01100111   01110001
```

FIGURE 5 UPDATION OF LOWER INTERVAL FOR MPS



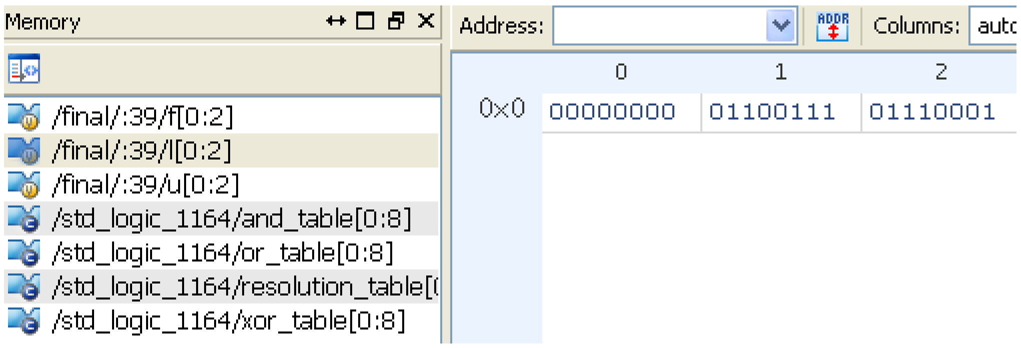
```
/final/:39/f[0:2]
/final/:39/l[0:2]
/final/:39/u[0:2]
/std_logic_1164/and_table[0:8]
/std_logic_1164/or_table[0:8]
/std_logic_1164/resolution_table[
/std_logic_1164/xor_table[0:8]

        0          1          2
0x0  11111111   00000001   11001101
```

FIGURE 6 UPDATION OF UPPER INTERVAL FOR LPS



Name	Value
a[7:0]	10110000
clk	1
lps[7:0]	10110000
mps[7:0]	UUUUUUUU

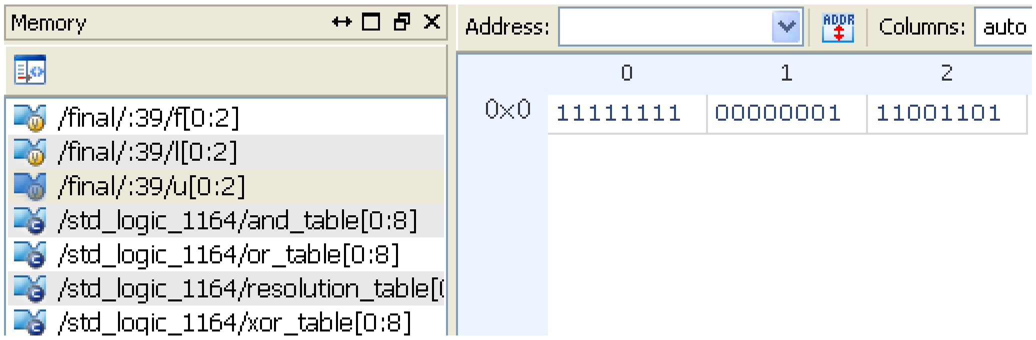
FIGURE 7 INPUT LIES IN RANGE OF LPS



```
/final/:39/f[0:2]
/final/:39/l[0:2]
/final/:39/u[0:2]
/std_logic_1164/and_table[0:8]
/std_logic_1164/or_table[0:8]
/std_logic_1164/resolution_table[(
/std_logic_1164/xor_table[0:8]
```

	0	1	2
0x0	00000000	01100111	01110001

FIGURE 8 UPDATION OF LOWER INTERVAL FOR MPS



```
/final/:39/f[0:2]
/final/:39/l[0:2]
/final/:39/u[0:2]
/std_logic_1164/and_table[0:8]
/std_logic_1164/or_table[0:8]
/std_logic_1164/resolution_table[(
/std_logic_1164/xor_table[0:8]
```

	0	1	2
0x0	11111111	00000001	11001101

FIGURE 9 UPDATION OF UPPER INTERVAL FOR MPS