Front. Electr. Electron. Eng. China 2007, 2(1): 34–38
DOI 10.1007/s11460-007-0006-y

RESEARCH ARTICLE

WEI Benjie, LIU Mingye, ZHOU Yihua, CHENG Baodong

Parallel VLSI design for the fast 3-D DWT core algorithm

Abstract By studying the core algorithm of a three-dimensional discrete wavelet transform (3-D DWT) in depth, this paper divides it into three one-dimensional discrete wavelet transforms (1-D DWTs). Based on the implementation of a 3-D DWT software, a parallel architecture design of a very large-scale integration (VLSI) is produced. It needs three dual-port random-access memory (RAM) to store the temporary results and transpose the matrix, then builds up a pipeline model composed of the three 1-D DWTs. In the design, the finite state machine (FSM) is used well to control the flow. Compared with the serial mode, the experimental results of the post synthesized simulation show that the design method is correct and effective. It can increase the processing speed by about 66%, work at 59 MHz, and meet the real-time needs of the video encoder.

Keywords 3-D DWT, parallel design, transpose the matrix, FSM

1 Introduction

The software implementation of the discrete wavelet transform (DWT) has been used widely since the fast MALLAT convolution algorithm and the second generation of the lifting scheme of the wavelet was put forward [1,2]. On the other hand, though the very large-scale integration (VLSI) of the DWT, especially the hardware design of the 3-D DWT, has just been studied in recent years [3] and has also been applied widely into video encoding fields.

When the 3-D DWT is used in video encoding, which can reduce the redundancy of the special and time domain, we can realize video compression without motion compensation (MC) very fast. Nowadays, people usually adopt the lifting scheme of the DWT, which uses a small amount of memory, has high speed and needs no multipliers. The lifting scheme of the 3-D DWT is used for hardware design in this paper.

In the existing reference of the 3-D DWT VLSI design [4–6] they adopt the MALLAT algorithm with multiplier and at a high cost. But the 3-D DWT based on the lifting scheme occupies little chip resource. This paper presents an efficient hardware architecture using parallel technology based on the deep research of the 3-D DWT software algorithm. We use very-high-speed integrated hardware description language (VHDL) to specify the structure of the system. Then, it is synthesized and the post simulated verification for field programmable gate array (FPGA) is given by using the SPARTAN-2E series, which can be a core module embedded into the video coder.

Translated from *Transactions of Beijing Institute of Technology*, 2006, 26(3): 211–215 [译自: 北京理工大学学报]

WEI Benjie (✉), LIU Mingye, ZHOU Yihua, CHENG Baodong
School of Information Science and Technology, Beijing Institute of Technology, Beijing 100081, China
E-mail: wei9100@besti.edu.cn

WEI Benjie
The Department of Computer Science, Beijing Electronic Science and Technology Institute, Beijing 100070, China

2 3-D DWT core algorithm description

2.1 1-D DWT lifting scheme algorithm [7]

The lifting scheme algorithm is a parsimonious and effective construction method of the biorthogonal wavelet. The basic idea is that the wavelet filter can be decomposed into basic modules. The implementation procedure of the wavelet transform is composed of three key steps: split, predict, and update, which are shown in Fig. 1. Take X_n as an input signal, then the predict and update procedures may be iterated many times. As the last step, a rescaling procedure is run to gain the approximation coefficients and detail coefficients.

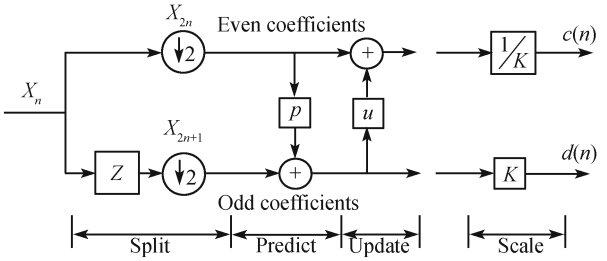

Fig. 1 Diagram of lifting scheme DWT

On account of the real-time requirement, we use a reversible lifting scheme algorithm of the integer wavelet. The computing difficulty is relatively simple and the processing is decomposed into two steps, such as
Predict

$$Y(2n+1) = X_{\text{ext}}(2n+1) - \left\lfloor \frac{X_{\text{ext}}(2n) + X_{\text{ext}}(2n+2)}{2} \right\rfloor$$

Update

$$Y(2n) = X_{\text{ext}}(2n) + \left\lfloor \frac{Y(2n-1) + Y(2n+1) + 4}{4} \right\rfloor$$

Boundary treatment: we use the two-pixel symmetry extension method to dispose the boundary.

2.2 The core algorithm of 3-D DWT

The 3-D DWT is based on the video coding requirement, aimed at processing the image sequence with the time-space and 3-D transform. We employ a 3-D separable wavelet transform. We take a 1-D wavelet transform along a row, a column and a time axis, to accomplish the 3-D wavelet transform. The transform diagrammatic sketch is shown in Fig. 2.

The 3-D wavelet core algorithm is implemented by using three 1-D wavelet transforms. A multilevel transform is the same with every level decomposed algorithm and only the size is different. Therefore, we emphasize every level core algorithm, which is uniform with the regularity. A multilevel decomposition can be realized by calling this module. The detail is expounded in Ref. [8].

First, we take a 1-D transform on the time axis. Second, we take a 2-D transform in the intra-frame.

3 Hardware design model of 3-D DWT algorithm

3.1 Hardware architecture of 1-D DWT [9]

Because the hardware architecture of a 3-D wavelet transform is based on the 1-D wavelet, it is important to establish the VLSI architecture of a 1-D wavelet. This is the foundation of the whole hardware architecture. We apply two finite state machines (FSM) to finish the architecture, in which one of the FSM controls the reading data from the RAM and implements boundary expansion of the wavelet transform, and the other FSM is used to call the data channels, predicting and updating data and putting the intermediate result into the succeeding double-port RAM to be stored.

The first FSM is shown in Fig. 3 and has fourteen states. The front eight states have the function of copying and extending. S8 finishes the continuous data input, S9–S12

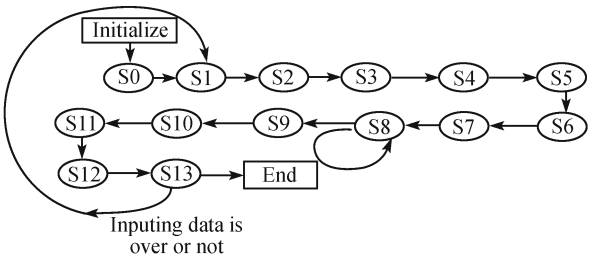

Fig. 3 FSM diagram for reading data

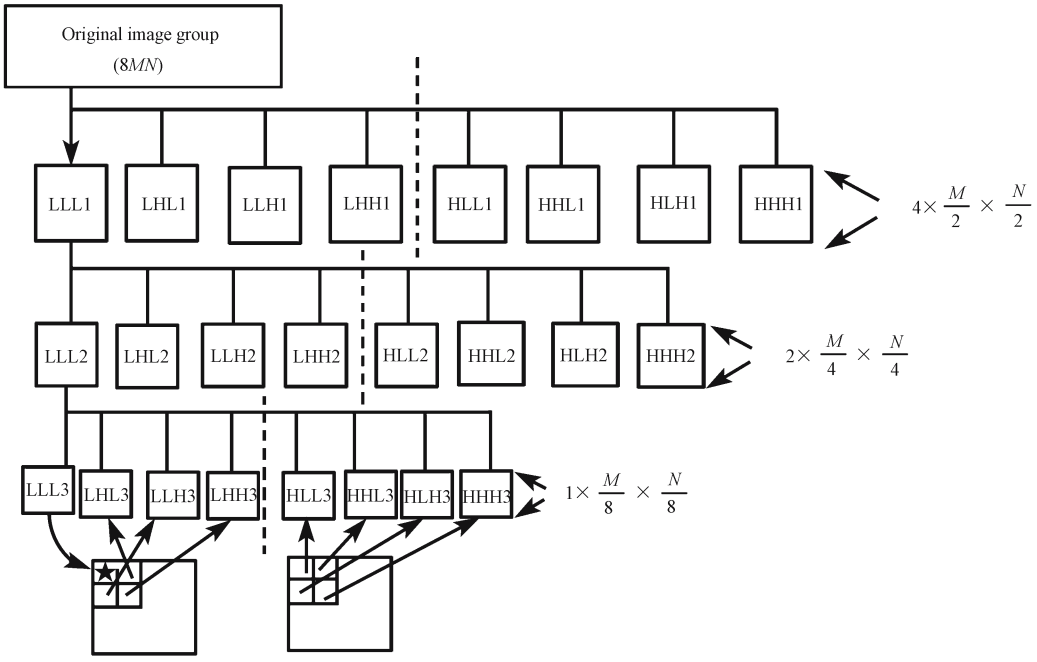

Fig. 2 3-D DWT diagram (three levels)

have the function of border copying and extending. The operation of the FSM is as follows.

S0 state: initialize RAM control and address signal, turn to S1.

S1 state: input the length of DWT and start up it, turn to S2.

S2 state: place $X(0)$ into the register and calculate the next data address, turn to S3.

S3 state: place $X(1)$ into the register and calculate the next data address, turn to S4.

S4 state: place $X(2)$ into the register and calculate the next data address, turn to S5.

S5 state: place $X(2)$ into the calculation data path of DWT, turn to S6.

S6 state: Place $X(1)$ into the calculation data path of DWT, turn to S7.

S7 state: Place $X(1)$ into the calculation data path of DWT, turn to S8.

S8 state: repeat to put one row data into the calculation data path of DWT until finish the input of the last third data and turn to S9, otherwise turn to S8.

S9 state: put $X(n-2)$ into the register and the data path simultaneously, and calculate the next data address, turn to S10.

S10 state: put $X(n-1)$ into the register and the data path simultaneously, and calculate the next data address, turn to S11.

S11 state: put $X(n)$ into the register and the data path simultaneously, and calculate the next data address, turn to S12.

S12 state: put $X(n-1)$ into the data path and calculate the next data address, turn to S13.

S13 state: put $X(n-2)$ into the data path, and it becomes the address of the second frame, turn to S1, otherwise ends to input data.

The second FSM is shown in Fig. 4. First, we initialize the data and the state, then input the three data in the three clocks continuously (S0, S1 and S2 states), and output the result at S3. At S4 we get the high frequency sub-band coefficients $Y(3)$ (odd DWT coefficients) by predicting. Finally, export the low-frequency coefficients of $Y(2)$ at S5 by updating. At S6, we assign values to $X(2)$, $X(3)$ and $Y(1)$ by using an iterative program. The program then decides whether 1-D DWT is over. When it is over, the state will be ended, otherwise turn to S4 and continue. We do the same process to finish 1-D DWT.

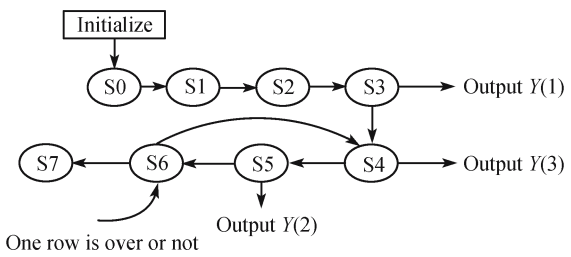

Fig. 4 FSM diagram for controlling 1-D DWT

Since the multiplier only includes –1/2 or 1/4 in the filter, the multiply operation actually executes the multiplicand shifting right and extension of the sign. Calculation delay is small and needs no pipeline. The filter for the video coding can process two sample values every two clocks. The original data can then be processed and put in its original place, which needs a dual-port RAM where it can be read and written at the same time.

3.2 General hardware architecture of the core algorithm

After realizing 1-D DWT and two FSM, we make a general hardware architecture model of the core algorithm, including three 1-D DWT, two dual-port RAMs and three multiplexers, two demultiplexers and one static RAM (SRAM), as shown in Fig. 5.

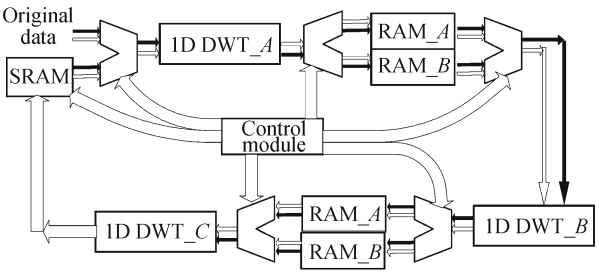

Fig. 5 Whole structure model diagram of core algorithm

When the system runs, we complete one level of the DWT by five steps in the major control module, which includes the following steps:

Step 1 Input the data from the first multiplexer. If it is a first level DWT, input the original image data, otherwise input the low-frequency data ((LLL) module) from the last.

Step 2 Control the data input according to the FSM clock, and send the data to the first 1-D DWT accurately, and do the DWT in time axis and simultaneously output the temporary results to the first double-port RAMs by the first demultiplexer.

Step 3 Similar to the operation in Step 2, after the second multiplexer, the second 1-D DWT and the second demultiplexer, we put the temporary results into the second double-port RAMs to finish the DWT in every row direction.

Step 4 Similar to the operation in Step 3, after the third multiplexer, the second 1-D DWT, to finish the DWT in every column direction, we put the data into the SRAM.

Step 5 When doing the next level DWT, we should decompose the low frequency (LLL module) to execute the subsequent operation by selecting the address and data under the major control module.

3.3 Implementation of linear address

While working on the hardware design for the existing 3-D DWT algorithm, we found that the address of the original data

and the temporary results are in three dimensions, but for the hardware implementation, the cube address must be changed into linear addresses, which is very important.

Assume that the row number is i, and j represents the column number and k is the frame number, so the original 3-D address (i, j, k) can be mapped into a linear address as $64i+8j+k$, where $i \in (0, M)$, $j \in (0, N)$, $k \in (0, L)$, M is the row number, N is the column number and L is the frame number. Their values are eight in experiments. While accessing the data, according to the sequence number of the DWT, first we do 1-D DWT for all the pixels in the time axis, where we set the addresses and the data flow (for example $8 \times 8 \times 8$ cube) as follows.

For every frame of 1–8 image, the addresses of the pixel (0, 0) in row 1 and column 1 are ordered as 0–7.

…

For every frame of 1–8 image, the addresses of pixel (0, 7) in row 1 and column 8 are ordered as 56–73.

For every frame of 1–8 image, the addresses of pixel (1, 0) in row 2 and column 1 are ordered as 64–71.

Addressed in the same way

…

For every frame of 1–8 image, the addresses of pixel (7, 7) in row 8 and column 8 are ordered as 504–511.

The second turn is the same as the first turn, it does 1-D DWT for all pixels in every row.

The third turn does 1-D DWT for all pixels in every column, the sequence for the access of addresses and data flow.

The address sequence for the first column coefficients are: 0, 64, 128,…, 448.

…

The address sequence for the eighth column coefficients in the eighth frame are: 63, 127, 191,…, 511.

3.4 Parallel design

In order to improve the efficiency of the core algorithm, we use a pipeline to carry out the parallel calculation in the hardware design. In Fig. 5, the temporary results are put into the buffer of the two dual-port RAMs. When the DWT in the first time axis is over, it changes the output direction from writing data into the A port to writing data into the B port, while initiating the next DWT in the row direction, which gets data from the B port. This way, while the last DWT is going on, we can execute the next DWT to form a parallel operation. The second and third DWT can be done in a similar manner. When the DWT in the row direction is finished, the data can be written into the RAM port, and start the column direction DWT to finish the whole 3-D DWT core algorithm.

By the pipeline design of the three phases, we let the three 1-D DWT be done in time overlay, to improve the calculation efficiency. The time consumption of 3-D DWT is equal to 1-D DWT.

4 Experimental results and analysis

On the platform of the ISE6.2 and the Modelsim5.7 tool software, we synthesized the hardware design based on the SPARTAN-2E series, about 60 000 gates (xc2s600e). The result is as follows.

The design modules occupy 1 296 slices (18%), 1 019 slices, flipflop (7%), 2 263 input LUTs (16%), 68 IOBs (13%), 64 BRAMs (34%), one GCLKs (25%), it can work at 59.03 MHz.

At the same time, we have done the simulation of a post-placing and routing for the eight frame sample image data block (8×8), the results are shown below in Table 1. Here we choose the front 40 data.

Table 1 The result data of the design

Data type	Value							
Original image data	145	144	144	146	149	151	153	151
	136	133	138	142	148	148	148	145
	127	123	124	129	140	147	144	142
	121	121	119	120	128	136	137	129
	117	118	118	119	125	130	130	121
Simulated results of post synthesized	143	−6	116	−1	127	7	131	73
	−1	0	−1	5	1	5	−1	21
	127	−1	117	11	127	−2	153	93
	−2	−5	2	0	4	−6	−1	0
	120	2	131	3	110	−50	206	1
C program results data	143	−6	116	−1	127	7	131	73
	−1	0	−1	5	1	5	−1	21
	127	−1	117	11	127	−2	153	93
	−2	−5	2	0	4	−6	−1	0
	120	2	131	3	110	−50	206	1

From the experimental results, we find that the post synthesized simulation results are the same as the C programming, which proves that the circuit can work correctly. Because of adopting the integer DWT (IDWT) algorithm, in which the integer is represented by sixteen bits, we can introduce calculation error as small as possible during implementing the IDWT with the C program and hardware.

The waveform diagram model simulation is shown in Fig. 6, which corresponds to the second row results in Table 1.

5 Conclusions

We have done the hardware design research deeply on the lifting scheme of a 3-D DWT, and put forward our own circuit structure. In other studies, because the 3-D DWT design method accomplishes DWT by mainly using convolution algorithm and needs many multipliers with high cost, the speed becomes slower than that of the lifting scheme of the DWT used in this paper.

In the process of the VLSI design for the 3-D DWT core algorithm, the FSM and the data path are used to finish the system control, data input and lifting. We present a method to translate the 3-D addresses to linear addresses, and transpose

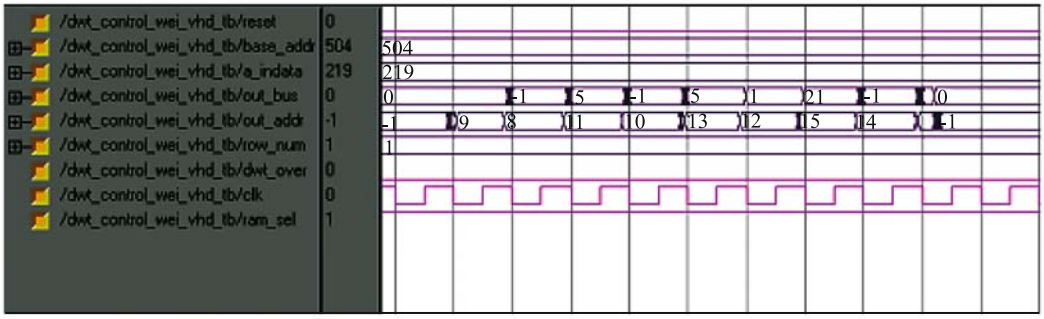

Fig. 6 The post synthesized simulation diagram of the core algorithm

the matrix. The parallel design of the hardware structure is made to make the three 1-D DWT execute a pipeline, with the throughput increased by about 2/3.

By means of comparing the simulated results of the post synthesized circuit with the C programming, it can be proven that the design is correct and efficient. In order to implement the complete VLSI design of the 3-D DWT, we need to further extend the hardware model of the core algorithm, which forms the 3-D DWT chip decomposed by many levels.

Acknowledgements This work was supported by the Defense Advanced Research Projects under Contract (No. 41308010408).

References

1. Mallat S G. A theory for multiresolution signal decomposition: The wavelet representation. IEEE Trans. On PAMI, 1989, 11(7): 674–693
2. Boliek M, Christopoulos C, Majani E. JPEG 2000 Part I Final Draft International Standard, 2000
3. Mehboob A, Denis O, Wael B. VLSI prototyping of low-complexity wavelet transform on FPGA. Canadian Conference on Electrical and Computer Engineering, 2002, 1(1): 412–415
4. Weeks M, Bayoumi M A. Three-dimensional discrete wavelet transform architectures. IEEE Trans. on Signal Processing, 2002, 50(8): 2050–2062
5. Das B, Banerjee S. VLSI architecture for a new real-time 3-D wavelet transform. IEEE Int. Conference on ASSP(ICASSP), 2002, 3(3): 3224–3227
6. Zhang Guoqing, Talley M, Badawy W. A low power prototype for a 3-D discrete wavelet transform processor. In: IEEE International Symposium on Circuits and Systems. USA: IEEE, 1999, 145–148
7. Dong Wenhui. Study on the wavelet transform, image compression, JPEG2000 and its hardware design. Dissertation for the Doctoral Degree. Beijing: Beijing Institute of Technology, 2003 (in Chinese)
8. Shen Lansun. Image Coding and Asynchronous Transmission. Beijing: Posts & Telecommunications Press, 1998 (in Chinese)
9. Yang Ke. The FPGA implementation of the JPEG2000 core algorithm. Dissertation for the Doctoral Degree. Beijing: Beijing Institute of Technology, 2005 (in Chinese)