Digital VLSI System Design

Prof Dr. S. Ramachandran
Department of Electrical Engineering
Indian Institute of Technology, Madras

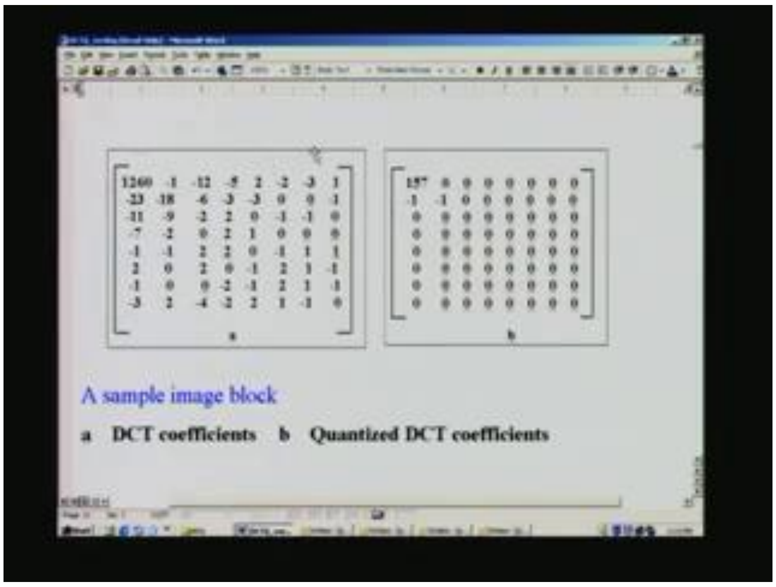
Lecture – 45

System Design Examples

(Continued)

We were looking at the novel algorithm we have developed for DCTQ quantization etcetera.

(Refer Slide Time: 02:32)



$$\begin{bmatrix} 1260 & -1 & -12 & -5 & 2 & -2 & -3 & 1 \\ -23 & -18 & -6 & -3 & -3 & 0 & 0 & -1 \\ -11 & -9 & -2 & 2 & 0 & -1 & -1 & 0 \\ -7 & -2 & 0 & 2 & 1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 2 & 0 & -1 & 1 & 1 \\ 2 & 0 & 2 & 0 & -1 & 2 & 1 & -1 \\ -1 & 0 & 0 & -2 & -1 & 2 & 1 & -1 \\ -3 & 2 & -4 & -2 & 2 & 1 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 157 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

a b

A sample image block

a DCT coefficients b Quantized DCT coefficients

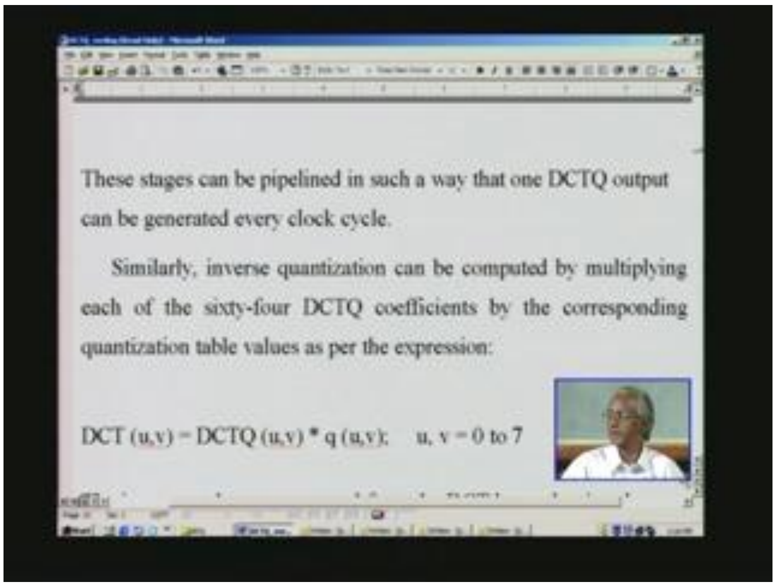
We saw this matrix which is the outcome of the application of DCT on an image block; we have not shown the image block deliberately. For each of these corresponding elements it may be anything from 0 to 255, being eight bit for the actual pixel values. You will normally see some 156, 160 like that all around that may be an average as far as the input pixels are concerned. The DCT is nothing other than the evaluation of three matrices C X and Ct and that gives final result which will be the 8 by 8 matrix shown here.

The very first coefficient, as mentioned, is the DC coefficient and the value is quite high when compared to all other coefficients. That is the reason I said it is 8 times the average of pixels intensities put together. If you divide this by the quantization matrix, for every 8 by 8 matrix you will have quantization matrix; first coefficient will be 8 corresponding to this term and 16 for this, then 16 for this and another 16 for this, then 19, 19, like that it goes 22 and so on until 84 or some such thing. You have to take one DCT and divide it by corresponding coefficient. In this case, it will be 8; so 1260 divided by 8 and after rounding it off, that is, dropping all the fractions what you will get is 157. It is not a rounding off but it is actually truncation.

If you want to recover this what you need to do is just multiply that by 8, this value. What do you get when you get multiply? Do you get it back? No. If you multiply this 8 into 66, I think it is 1256. You started with 1260 but when you reconstruct, this is what is called reconstruction, you do not really recover what you have. Quantization is responsible for this because you are quantizing by 8. That means to say we take only every eighth value. That is why we have in this fashion. This is a lossy process in work and therefore, you cannot reconstruct the actual image but it will be an approximation of the image. On most occasions, it will be quite good. Sometimes you may be surprised to see a very good image constructed.

The purpose of quantization is to create as many 0s as possible as we have mentioned before which need not be encoded in VLC- variable length coding and only these non-zero coefficients are actually coded and converted into serial bit and sent over as a bit stream on the transmission channel. This is just a sample block we have seen here. This is a DCT coefficient as mentioned and this is after quantization. Quantization is nearly division by another constant, a group of constants.

(Refer Slide Time: 05:55)



These stages can be pipelined in such a way that one DCTQ output can be generated every clock cycle.

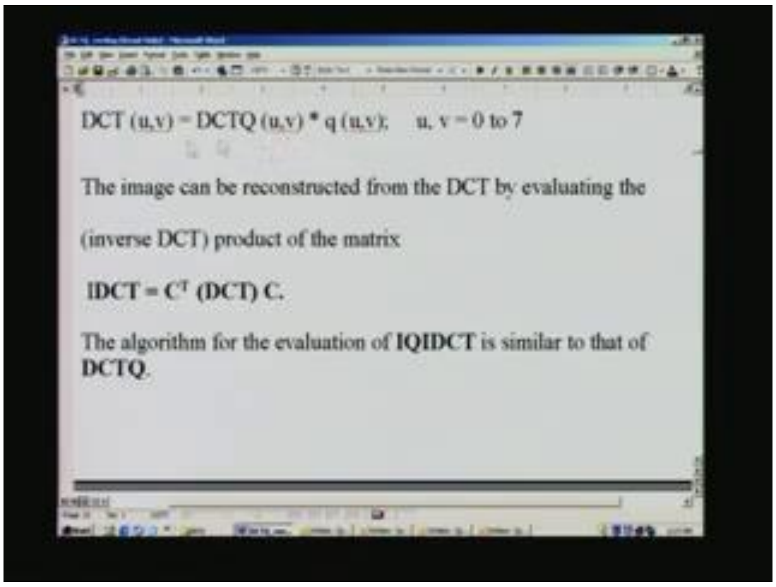
Similarly, inverse quantization can be computed by multiplying each of the sixty-four DCTQ coefficients by the corresponding quantization table values as per the expression:

DCT (u,v) = DCTQ (u,v) * q (u,v); u, v = 0 to 7

These stages can be pipelined in such a way that one DCTQ output can be generated every clock cycle. I think we have seen this as well and we will see a little more later on.

Similarly, inverse quantization can be computed by multiplying each of the sixty four DCTQ coefficients by the corresponding quantization table values as per the expression. This is what we have already explained. Actually this is a quantization matrix, 8 by 8 matrixes. Once again you can see u, v correspond to DCT coefficients earlier. Here we recover the DCT portion. We start with DCTQ and multiply. Earlier we divided by 8 in order to get the DCTQ from DCT; now we have to multiply by 8 in order to get the very first coefficient; similarly, for the other coefficients 16, 19 and so on. We will create the corresponding DCT value here. This is the process of reconstructing the image at the receiving end at the decoder.

(Refer Slide Time: 07:16)


DCT (u,v) = DCTQ (u,v) * q (u,v);   u, v = 0 to 7

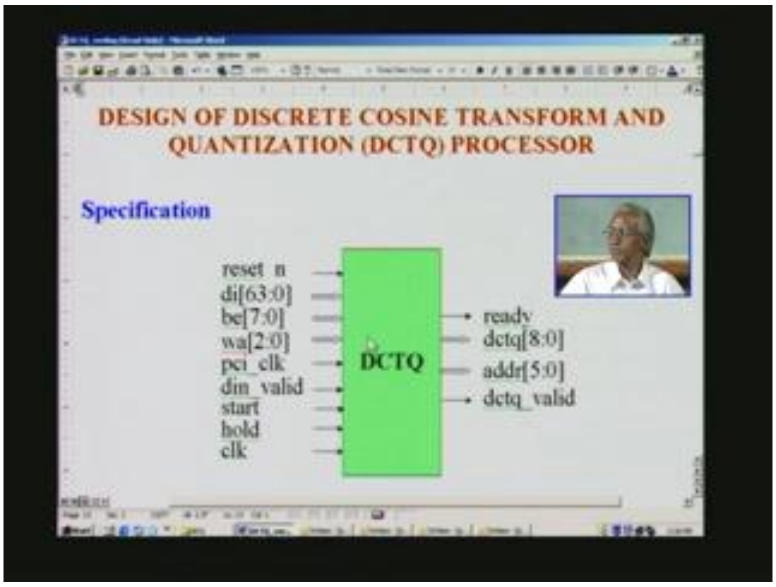
The image can be reconstructed from the DCT by evaluating the (inverse DCT) product of the matrix

IDCT = C^T (DCT) C.

The algorithm for the evaluation of **IQIDCT** is similar to that of **DCTQ**.

The encoder decoder that we have seen before is also abbreviated as codec. The next step is we still have only the DCT coefficients available; we have not yet recovered the actual image. We can recover it only by doing an inverse operation of this which is similar to DCT, only order has to be C T into the DCT and C. These are all, once again, 8 by 8 matrices, three matrices. The parallel algorithm we have done before is equally applicable for this IDCT as well; out comes 1 pixel per clock. That is the throughput that you can get here. It has to be the same rate as the DCT creation and the same algorithm works too.

(Refer Slide Time: 08:07)


DESIGN OF DISCRETE COSINE TRANSFORM AND QUANTIZATION (DCTQ) PROCESSOR

Specification

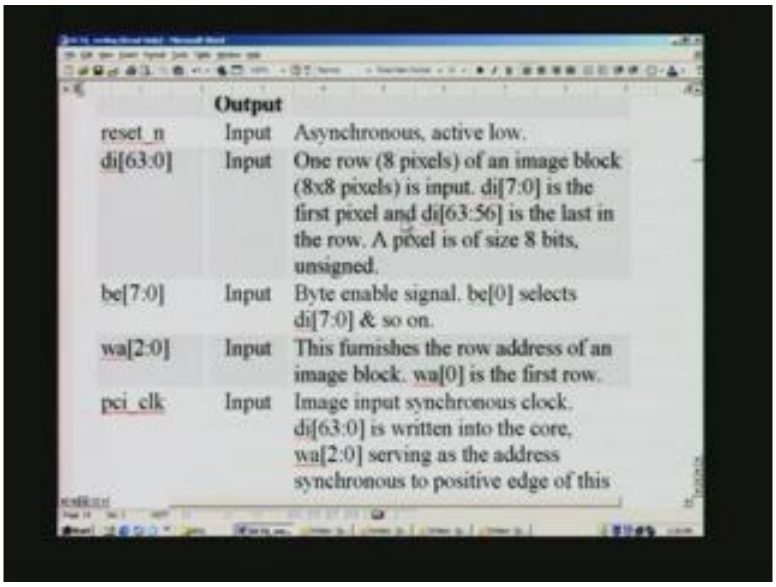
Now we will really start going into the design. Before we design any system that you want to design, you must first have an overall bird's eye view. For example, you ultimately want to do a chip so a good starting point will be to put a block box and then label this. This is the design for DCT quantization as I mentioned here. What we will formulate is this specification. Let us examine what signals you will require in order to communicate the image. For example, I can use a host processor in order to communicate the signal, I mean image. We need a very high throughput so we can think of using a parallel bus such as pci bus which we have already had a glimpse of when we did pci orbiter earlier. We had used any block of image. We can apply one row of a block at a time if you have 64 bits and through pci bus a 64 bit bus will be handy and you can use pci signals. For example, b stands for the byte enable and there are 8 bytes here. Which byte is referred? If you enable that particular byte you need signal. These are all basically pci signals. If you have a core, you need to reset at any point of time. What we have here is reset n and this is standard in any of our designs seen earlier. Once you have this you can communicate the image information from the host processor or from any other processor such as video grabber which we have seen before. Into this DCTQ which is the hardware implementation after going through verilog coding, which we will do shortly.

In order to write different rows of a block; there are eight such rows. So you need a right address for that; three bits are needed for that. You also need a clock such as pci clock. This is the host computer interconnection right from all this. When this data is valid should also be indicated by the host and that is by this. Once you have put in eight rows, you can write it in eight clock pulses using this pci clock. After you have written one block of information you can start the entire DCTQ process. When the DCTQ process is going on you can also input next block of image; this DCTQ group processing there are sixty four coefficients and as I mentioned before you can have one pixel processed every clock equivalent to one pixel process. Thereby, you have a sixty four clock cycle requirement for processing one block; whereas, you need just eight clock cycles as far as pci clock is concerned in order to input here. That relieves the burden of the host processor which is dumping in the image. So it can attend to any other duty other than nearly inputting the information. This is a time consuming operation. As we have seen before, realization of three matrices is actually a very time consuming operation. I think it is n cube times, n stands for 8 and so many computations are involved, multiplication as well as addition. You can cross check from the matrix we have seen before.

Once you have loaded the image you can as this DCTQ to start the DCTQ process by asserting this signal. Once it has started, you can hold the processing at any point of time. Thereby, you do not have to bring in what is called latency which I will explain later on if you use hold signal. There is also a system clock required for a DCTQ operation to turn out the DCTQ and that clock is fed here. When this DCTQ is ready to accept input will be indicated by an output signal called ready. The actual DCTQ output after evaluating will be output here and it will be a 9 bit output. This 9 bit output is signed; it will be in two’s complement. This is as per the requirements of the JPEG MPEG standards. That is what we said before, right at the introduction, that whatever we design will have to conform to the standards. Otherwise, there is no communication from one product and another product. Which coefficients we are referring to? Is it DC coefficient or the AC coefficient? If so, what is the address? That question will also have to be answered.

The answer is output this address for the coefficient DCTQ coefficient. When this DCTQ is valid? That also has to be indicated by a high going signal. With this you can very well design a DCTQ processor. We have seen all this from a bird’s eye point of view. We have also used some of our knowledge about the actual host interface such as pci interface. Once we have formulated what we have briefly explained each of the signals is listed. This is a block diagram of DCTQ.

(Refer Slide Time: 13:55)

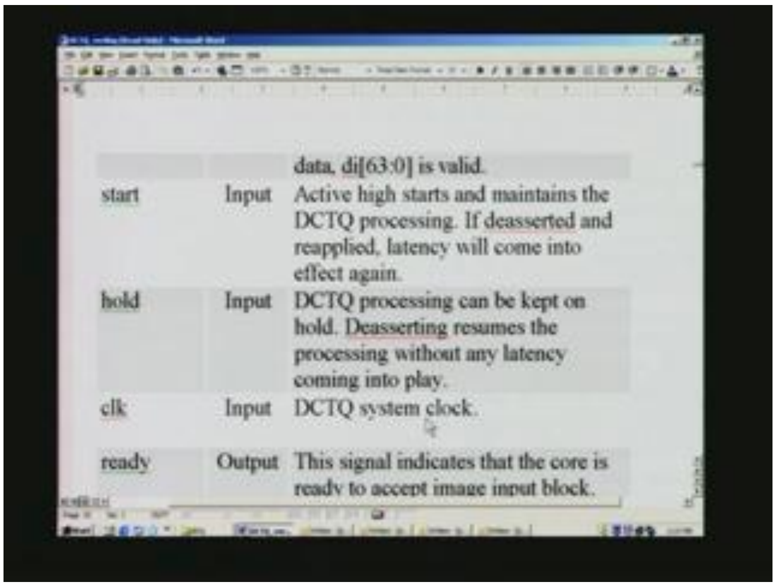


	Output	
reset_n	Input	Asynchronous, active low.
di[63:0]	Input	One row (8 pixels) of an image block (8x8 pixels) is input. di[7:0] is the first pixel and di[63:56] is the last in the row. A pixel is of size 8 bits, unsigned.
be[7:0]	Input	Byte enable signal. be[0] selects di[7:0] & so on.
wa[2:0]	Input	This furnishes the row address of an image block. wa[0] is the first row.
pci_clk	Input	Image input synchronous clock. di[63:0] is written into the core, wa[2:0] serving as the address synchronous to positive edge of this

Each of these signals will have to be explained. I will quickly explain because I have already explained. This reset is active low asynchronous which is standard in our design and input is fed through this di 1 row, that is, 8 pixels of an image block; 8 by 8 pixels is input and into this di 7 through 0 is the first pixel and the last byte is the last pixel in the particular row of 8 pixels In a block you have eight such rows. We have to write one after another. A pixel is of size eight bits, unsigned. Note that it is unsigned. This is nothing other than x matrix in that Cx Ct evaluation and so the x is complete only with 8 rights, as such, because every right you write only one row of the block.

You have this byte enable; byte enable be 0 selects di 7 0 and so; on the higher order it is selected by this 7. We have seen this as well. The 0 is the first row address and 7 is the last row address of the block. Image input synchronous clock, that is, the pci clock and all the bits are written using this address serving as the address synchronous to positive edge of this clock. We have to define all these signals. When it really happens, whether it is positive edge or negative edge, whatever the case is we will have to specifically spell it out.

(Refer Slide Time: 15:28)

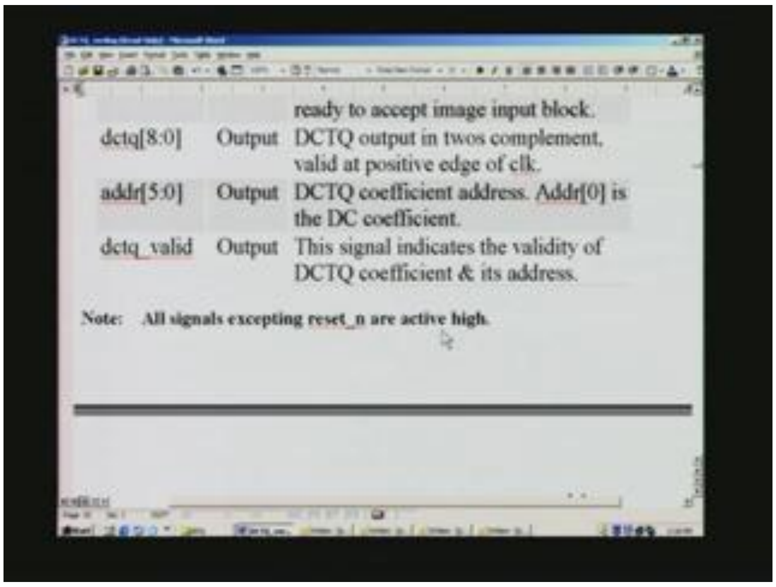


		data, di[63:0] is valid.
start	Input	Active high starts and maintains the DCTQ processing. If deasserted and reapplied, latency will come into effect again.
hold	Input	DCTQ processing can be kept on hold. Deasserting resumes the processing without any latency coming into play.
clk	Input	DCTQ system clock.
ready	Output	This signal indicates that the core is ready to accept image input block.

Similarly, data in valid: when the input data is valid. Then we have start signal and whether they are input or output are designated straightaway here. Active high starts and maintains the DCTQ processing. Once you have started, it should not make it low; should keep it high. If you make it low, the whole thing will be suspended and again have to start fresh. Every time you start of fresh, you have to pay the penalty of latency. In this particular design, you will see, later on, that the latency is around 45, which is quite hefty. We will explain why later on. If deasserted and reapplied latency will come into effect again. That is what we are saying. This latency is applicable only for the very first block that we are processing, which is to say, the output starts coming only after the latency time has expired. It will start only after delay. Once it starts, 1 pixel per one clock rate is maintained as long as you do not make it zero. The movement at any point of time you make it 0 again that 45 latency will come into the picture.

If you want to avoid that a better alternative will be to use this hold so that it freezes at the point where you left and you can resume at a later time as the need arises by deasserting the hold. That is what is mentioned here. DCTQ processing can be kept on hold; deasserting resumes the processing without any latency coming into play. Clock is the system clock for DCTQ operation. Ready is the output signal. This indicates that the core is ready to, core, in the sense the DCTQ core to accept image input block.

(Refer Slide Time: 17:08)

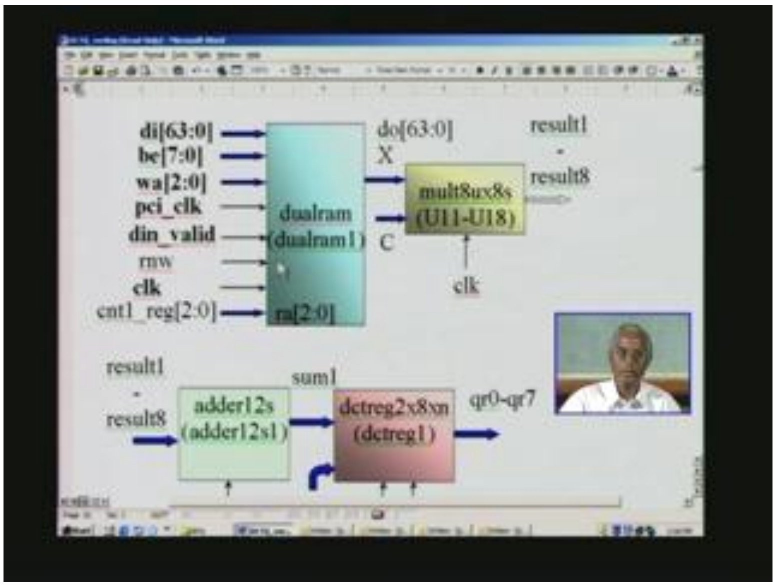


		ready to accept image input block.
dctq[8:0]	Output	DCTQ output in twos complement, valid at positive edge of clk.
addr[5:0]	Output	DCTQ coefficient address. Addr[0] is the DC coefficient.
dctq_valid	Output	This signal indicates the validity of DCTQ coefficient & its address.

Note: All signals excepting reset_n are active high.

DCTQ is the output in two’s complement valid at the positive edge of clock. Note that this clock is different from pci clock, that is for inputting the image; whereas, this is outputting the actually churned DCTQ. The corresponding DCTQ address is furnished here; address 0 is the DC coefficient; all others up to 63 are AC coefficients. When it is valid is indicated by this and also the validity of address. All the signals excepting reset n are all active high. This is the standard design.

(Refer Slide Time: 17:48)

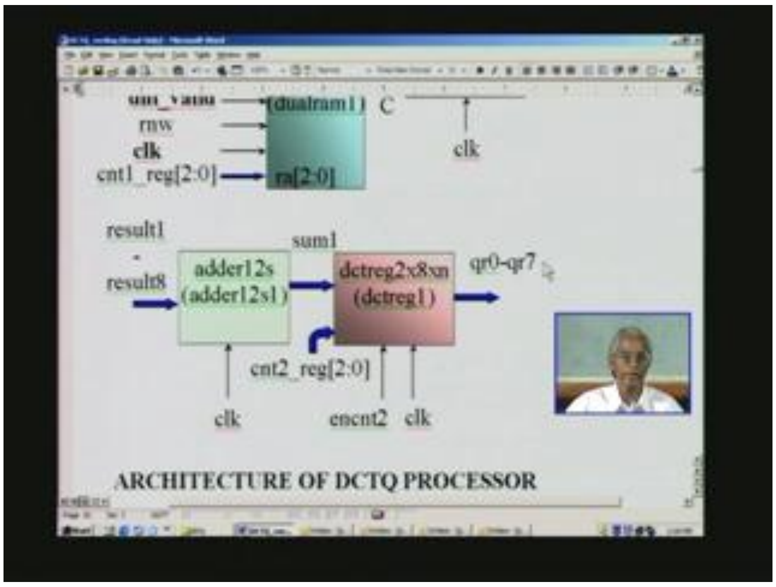


We will have a look into the architecture. The first step is we have to specify what we want. That is what we have done by describing for putting a block and then describing these signals. They are the basic definitions. Once with this and they are undercurrent for all this is the implementation of the algorithm that we have seen before. That is the crumbs actually. What we are trying do is merely finally turn that algorithm into a chip. Just prior to making the verilog design, we need to put a hardware structure here and this is what is called architecture. The architecture will have to reflect the actual hardware that we need. Let us see.

We have already seen dual RAM designed earlier and that is precisely what we will use here so I will not describe dual RAM except for the signals merely to recollect what we have done before. All these bold notes are the pin configurations that we have seen before. We have seen a block diagram with all the I/O pins there. That is what will be the final IC when you do it as an ASIC or FPGA. Those things are mentioned bold within the architecture because there are so many signals in the architecture. Other than bold, they are all internally created. So it will come from some other module. You have already seen these things before. Input is applied through di and when it is valid byte enable, which row address we are talking about is here, pci clock at the positive edge of the clock we input here. It is a dual RAM

In the sense we have two RAMs. The first block that you have to put in will be put into one of the RAMs, say RAM 1 and RAM 2 is empty to start with. Therefore, DCTQ quantization will not start. After completing the first RAM 1 field, we will give a start signal. That start signal is down below which we will see later on. We have seen up to this. Din valid is the data invalid. You need eight pci clock cycles in order to write one image block. Once that is done, we switch control from one RAM to another. That is done by toggling this rnw; this toggling of rnw must be by the DCTQ processor. This will be the signal internally created. We will see later on that it is coming from the controller of which we will see in the design. This is the system clock.

(Refer Slide Time: 25:25)



In a dual RAM, you need not only write into the RAM but also read in order to process the DCTQ. Unless you read, you cannot process DCTQ which will happen right from here. For that, you need the row address. Once again, three bit address is required and so the nomenclature read address. So you will read what you have already filled in. For example, to start with, first block is written in RAM 1 and then second block is written in RAM 2. When this happens concurrently you would have started this DCTQ. It is being read from RAM 1 and this is addressed by a counter created in the controller. The controller is not an fsm realization as we have seen before in previous designs; it is a counter based design. You will see that it is very handy for where you need to address memories, registers in large number. So counter based design will be very handy. This is a deviation from routine designs.

Once you have filled and given a slot for DCTQ processing what happens is this read address is applied to this RAM 1, let us say, and gets the very first row and into this X here. Note that the row has only 64 bits. That is what is applied here and corresponding to this what we will do is we will evaluate Cx matrix. So all this structure, the architecture, will have to reflect this parallel algorithm we have developed before. A same nomenclature is put at strategic points for you to graphs what they are. Remember that only one row of the image is being applied at one point of time. So at every positive edge of this clock we will keep on changing to a new row. That means to say it is a bump let at a very high rate of the system clock. That is the beauty of the pipelining. As I mentioned before, we have forty five stages of pipelining which should be examining as we proceed further.

The first thing is we want to realize Cx. As you may recollect it is the first row of the C matrix multiplied by the first column of the X matrix. That is what we have seen before. That is what we are doing at this stage. This C row remains constant for next 8 clock cycles including the first one and only the X will change from the first column, then second column and so on in order to create one complete partial product row of the matrix. That is what you had seen earlier which means that you had to change this one eight times. So for every clock this will keep on changing and at every clock, you will do eight multiplications at a time, which means you need eight multipliers. That is what it is here. We have also seen the multiplier design earlier, precisely the same multiplier; therefore, this will also not be covered here. You remember first input is unsigned; we mentioned before that this X, which is image, is unsigned whereas C is the cosine. So, naturally it calls for a signed number and that is the reason why we are using 8u into 8s multiplier. There are eight multipliers and nomenclatures as different ICs you can identifiers U11 through U18. You need eight pipeline stages; as you recollect all our multipliers earlier were eight pipelines. There are eight pipeline registers. So any data going here will manifest only after a delay of eight clock cycles. That is what is responsible for the latency.

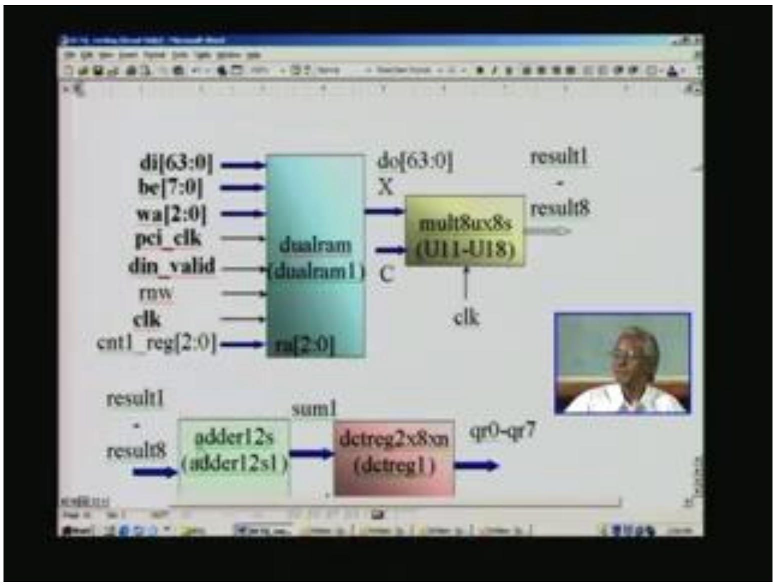
We have also seen in dual RAM, in a RAM which is being called by dual RAM, there is one clock delay there because one pipeline register was used there and another pipeline in the top level dual RAM. So this contributes to two clock cycles delay so there is a two pipeline here and eight here. We need to keep track of all this and I hope you will keep the count, two here, eight here and next was, say, adder. Out comes the result which is nothing other than CX; eight products are available and they are designated as result 1 through result 8. All of which need to be added which we can use our previously designed adder, which is adder 12s. You recollect that it is not register. I mentioned there that it is not a register for a particular reason. The reason is will be clear now. That is because we have following register and what is the point in having a register here and duplicating one more register? It would be a waste of register there. In order to avoid that we dispense with the register, whereas, you will see that there will be a register in the next stage because there is no register in the following stage. We have designed this also. In fact, we have designed all these three blocks. What we have not designed is this register. Just keep track of this. If you recollect it will be five pipeline stages, 2 8 5 and here it is actually a group of eight registers. In fact, there are 15 registers. We will understand why it is so when you go into verilog code.

To register all the partial products, first C X created one row of a partial product P00, P01 and so on up to P07; all those will have to be registered. Each of these will be arriving at different clock points. You will get P00 at the first clock, P01 at the second clock and so on. You have to keep track of all that and then you have to register at the appropriate time. There are registers designated as qr0 to qr7. The very first register is qr0 and that will be registered when the address for this one is given here. This is derived from the controller. As I mentioned before, it is all based on counters and second counter is invoked here so three bit counter, whereas, the first is six bit counter of which we have used only the lsb portion. Let us see where we use the msb.

Here, this is nothing other than address for the register we want to have. So if it is very first P00, this address will have to be 0, for P01 it will have to be 1. Automatically, this counter etc, will have to run very synchronously. If you make even a single mistake everything will be topsy turvy. These registers will actually be registered inside only with an enable control. This enable control is the same as the enable control for the counter which we will be seeing in the controller design. Again, there are eight registers so the data will go around all these and emerge only after eighth clock pulses. So, the entire qr0 to qr3 is 7 which is none other than the partial product Ps and will be available only after 8 clock cycles. With the arrival of the next clock, we have already seen that there is no relaxation as far as the input is concerned. Once the first block is processed it does not wait; it immediately starts with the second process. If you inspect this data bus here you will see that there is incessant activity there. Every clock pulse some activity will keep going on. All of them are meaningful; therefore, there is no respect for the DCTQ processor all through. Only here there is a little in the sense that this will be available to you, this partial product, for eight clock pulses because unless this is stationary you cannot compute the second partial product which will take 8 clock pulses.

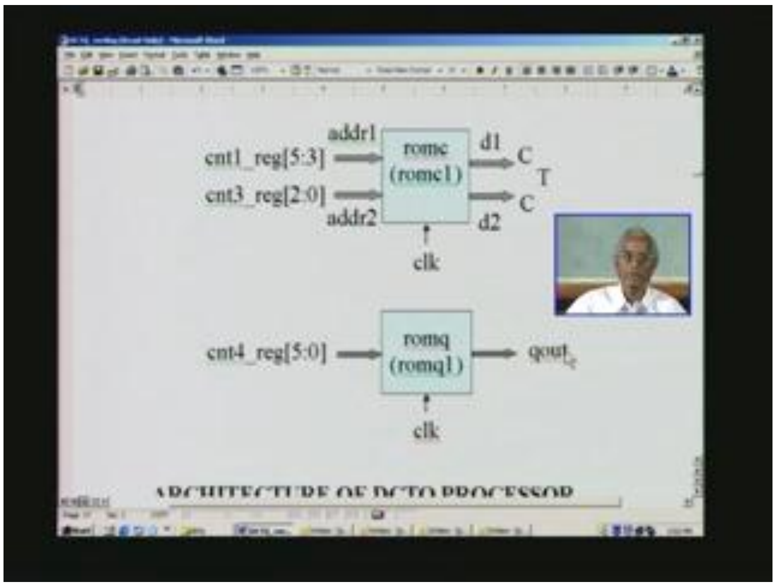
So if you do not have duplicate registers, I said 15 registers, you cannot accomplish this because the first register will be overwritten and you will lose what you have stored which we need for further computation. It will be overwritten and we do not want that to happen. That is why we need to be very careful about this design. Although the design is quite simple, we need to take extreme care.

(Refer Slide Time: 29:42)



What we had seen before is C X and we have only one row of a partial product. The total delay, you remember is 2 pipelines, 8 here, 5 here, then 8 here. This architecture continues.

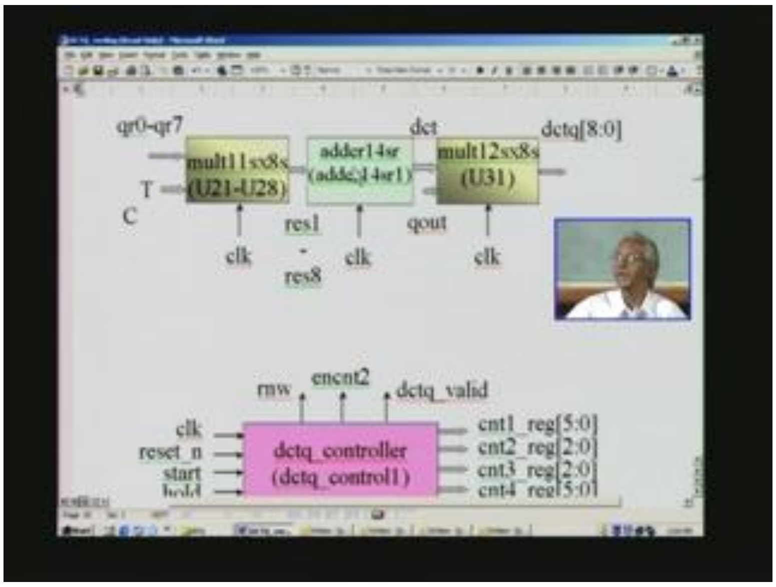
(Refer Slide Time: 29:57)



Next we need to have C and CT and they are available in the ROM store. Similarly, we want to have a quantization table; that is also ROM. In fact, we have seen the design of both the ROMs earlier. We pointed out that it is an inverse quantization that we are storing there when we dealt with the design. That is precisely what we see here. For this, once again this address you remember there are two addresses. One address is derived from counter 1, note that this address is derived from msb of the same counter 1 there. As we have to not only address X here but also see simultaneously. Therefore, this same counter is addressing both. The only difference is that this is in lsb, which means that this happens more frequently then the msb and that is once in 8 times only because you need to have all the rows of the matrix for 1 value of 1 set of C and repeat the same thing again in order to get the partial product that we have seen. This is quite simple. Counter 4 gives the address for the romq; outcomes the quantization matrix value here. For example, if you give 0, out comes 8 here; if it is 1 then it will be 16 and so on as per the quantization matrix we have stored here.

Here also we do not really store the actual quantization matrix but its inverse; that also a number of times more in order to improve the accuracy. Note that this one also has latency here as well as here one clock pulse latency or something. But anyway this is a parallel process when compared to the other here.

(Refer Slide Time: 32:08)



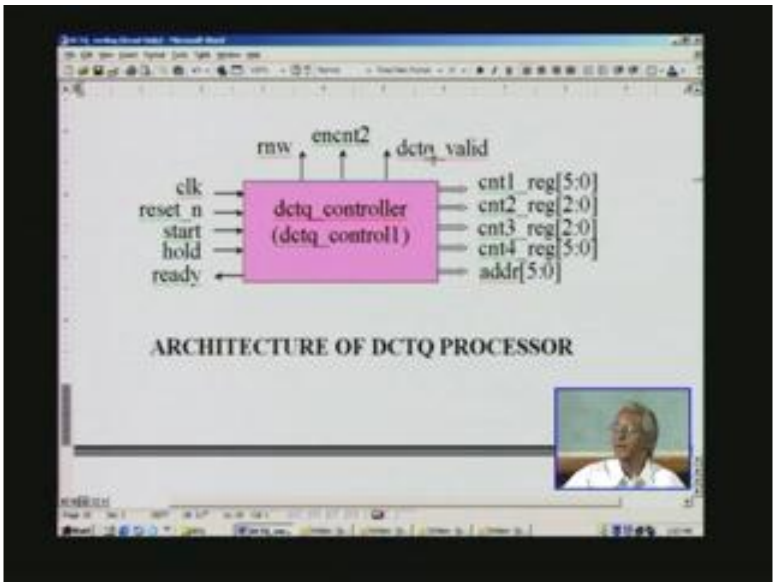
What we have done earlier is evaluated partial product and that we feed in the second stage of multiplication. For second stage of multiplication note that we need a higher precision, 11 for this one, because qr0 to qr7 I have taken as 11 bit precision. I have experimented with several bit precision and arrived at this one to give an optimum quality of the image. In fact, all through, all the precisions have been arrived at only after experimenting. So you also need another input which is nothing other than CT. Now, we have to multiply what we have earlier as partial product P which is none other than qr and the register that we have seen before which is nothing other than CX matrix, a CX 1 row of the matrix. We need to multiply that row into the CT matrix in order to get the DCT. So we have to apply this CT which is none other this same C ROM table that we have seen. You need, once again, eight multipliers as explained in the algorithm earlier and designated as U21 to U28. You can see precisely the same thing in the verilog coding. I have put here the very same nomenclature I have adapted in the verilog coding too. All the signals and everything will be 1 to 1 correspondents in the coding. After this, the eight results of what we have multiplied are there, result 1 through 8, and we have to add all of this. We need one more adder which is 14 bit with registered output

This is what I have given as an assignment earlier and hope that you have done it. I will not show it anyway. You have been given an assignment for this multiplier as well as this multiplier. This multiplier is to create DCTQ out of DCT and this multiplier has nothing other than adding all these rows into columns multiplied here in order to get the DCT here. If you add, what you get is this DCT here. DCT precision is twelve bits here. This is as per the standards and it is in signed two's complement form; qout is that romq table we have seen here; this is qout romq It gives the quantization value which we need to input to the multiplier in order to get DCTQ. DCTQ is actually a division of DCT by the quantization value but division can be interpreted as a multiplication if you inverse the quantization value. That is precisely what has been done here. The ROM we have inverted value so that we can use a multiplier there instead of divider. That is all here.

Once you multiply DCT is again twelve bits. That is the reason why it is 12s. This is just eight bit precision. Once again, a signed value is put, in fact, unsigned is enough I think. It might be unsigned. You can correct that one; out comes DCTQ nine bit here.

What we have before we wind up this architecture is we have the whole thing. You will see that it appears very complicated. So, you need to regulate the whole activity.

(Refer Slide Time: 35:48)



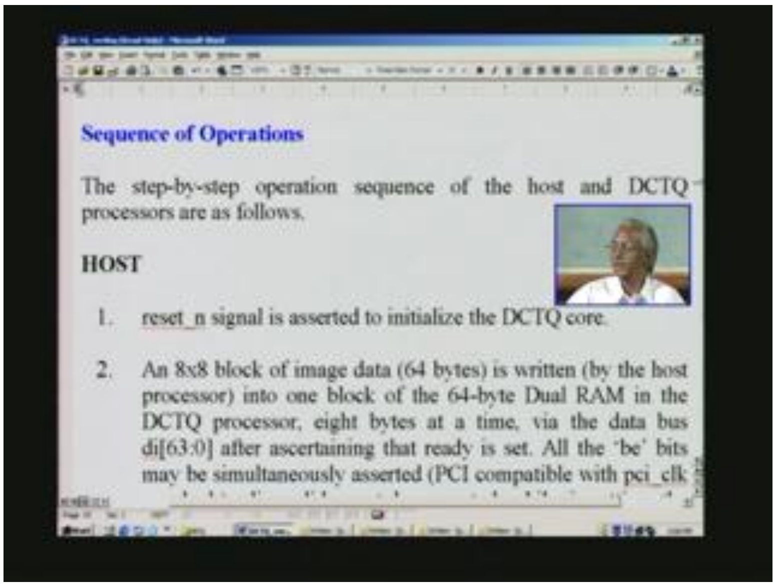
You can do so only if you have a controller specifically designed for that. You can see so many control signals coming out of the controller. Plenty of other signals are inside this. When we look into the DCTQ controller we will see more details. It calls for several counters of various bit precisions as listed. In fact, counter 5 is nothing other than address; this is nothing but the DCTQ address. Address 0 corresponds to the DC coefficient and all others are AC coefficients. We will also have to create DCTQ valid signal which will keep in step with DCTQ output. So, we will have to keep track, as I have mentioned, of all the pipelines; only then will we know when to activate this.

We will see a few more details on that later on. Enable counter 2 was required for that dct register for storing the partial product. That is what is here. We also need rnw in order to switch from RAM 1 to RAM 2 in dual RAM. That is why this signal is here. System clock for DCTQ processor is here. You need a reset to reset the whole system. There may be internal registers that we need to reset. In fact, all the variables, signals that we have inside will be reset by standard always block. You need a start in order to start the DCTQ processing. This is what I mentioned before right at the beginning of explaining the signals. You also need a whole signal. Ready is issued by the DCTQ processor; it is primarily generated at the controller. This completes the architecture. Before this, we will once again examine how many delays are involved. (Refer Slide

Time: 37:45) We saw that two pipeline here, eight here, then just add all that, 10 here, then 5 here, 15, then 8 here, 23. This delay need not been reckoned because it is a parallel affair. How much did you count? 23

We are looking at only the data flow. Here, once again, 23 plus 8, 31 then 6 here because register is involved. So how much does it comes to? 37. Once again, there is one more multiplier which is 8, so 45, out comes DCTQ only after 45 delays. Every clock, the whole thing is pushed one after another. You have just seen the entire data flow and this is what we meant by pipelining. You remember we have covered it. While designing multiplier we saw how pipelining is? What the concept is? Now you should appreciate better when we deal with the actual design such as DCTQ. So we have seen 45. We also need intermediate points later on; we will come back to this later on. This completes the architecture which has a 1 to 1 correspondence with the algorithm.

(Refer Slide Time: 39:22)



Sequence of Operations

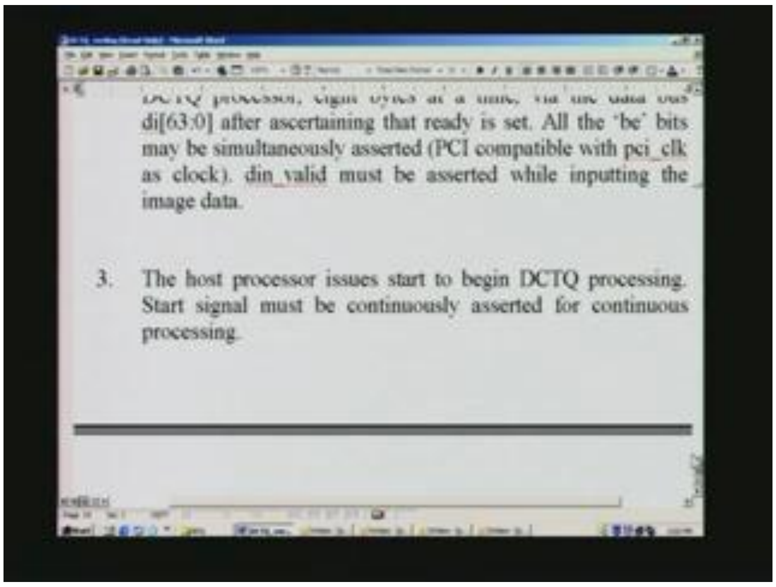
The step-by-step operation sequence of the host and DCTQ processors are as follows.

HOST

1. reset_n signal is asserted to initialize the DCTQ core.
2. An 8x8 block of image data (64 bytes) is written (by the host processor) into one block of the 64-byte Dual RAM in the DCTQ processor, eight bytes at a time, via the data bus di[63:0] after ascertaining that ready is set. All the 'be' bits may be simultaneously asserted (PCI compatible with pci_clk

The next point is whatever processor we have designed for DCTQ cannot be stand alone; we will have to build further hardware. You have to put it in a PCB and you may require other hardware or even the host processor. That is why we assumed pci bus interconnection. It calls for a host processor as well as the DCTQ processor itself so we have to intercommunicate with them. What is the way of communication? You need a sequence of operation for that. It says the step by step operation sequence of the host and DCTQ processors are as follows: What we need to do for the host is first, the host will have to send the reset signal to initialize the DCTQ core. As there are so many registers inside they will have to be initialized to different values not necessarily 0. That is what it is being done here. At any point of time you can always apply the reset. Once you apply a reset and start again latency will come into the picture so take care of that. If you do not want latency do not assert reset only right at the beginning when the system is switched on or deliberately when the whole JPEG or MPEG is initiated. Reset at that point only and thereafter do not reset. Probably after processing one frame if you feel like an initialize once again and if the need arises. Otherwise, you can use the whole signal and you can dispense with the latency. Thereby you will speed up the processing considerably. So, 8 by 8 block of image data, 64 bytes is written by the host processor into one block of the 64 byte dual RAM in the DCTQ processor; 8 bytes at a time via the data bus di. After ascertaining that ready is set; all be bits may be simultaneously asserted; PCI compatible with pci clock as input clock; din valid must be asserted while inputting the image data. It is quite plain; it does not need any explanation.

(Refer Slide Time: 41:32)

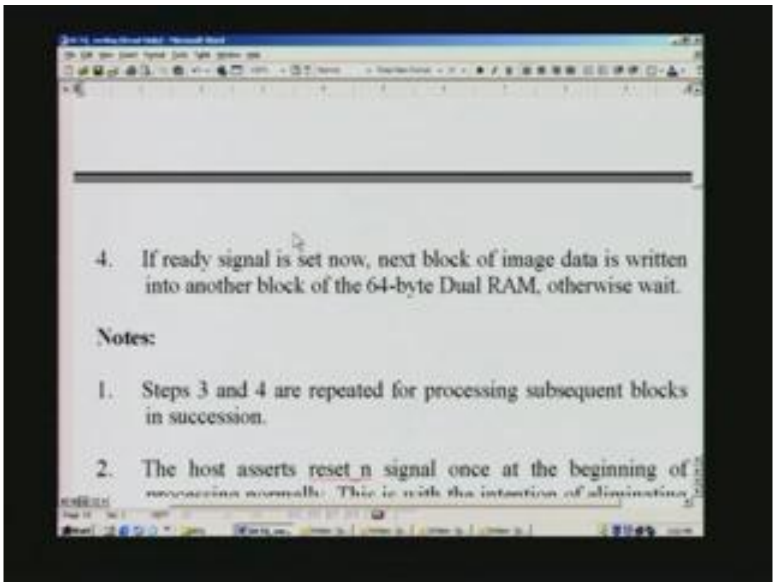


DCTQ processor, eight bytes at a time, via the data bus di[63:0] after ascertaining that ready is set. All the 'be' bits may be simultaneously asserted (PCI compatible with pci_clk as clock). din_valid must be asserted while inputting the image data.

3. The host processor issues start to begin DCTQ processing. Start signal must be continuously asserted for continuous processing.

The next point is the host processor issues start to begin DCTQ processing. Start signal must be continuously asserted for continuous processing. If you withdraw that one and apply again latency will come into the picture again. That is the reason.

(Refer Slide Time: 41:45)



4. If ready signal is set now, next block of image data is written into another block of the 64-byte Dual RAM, otherwise wait.

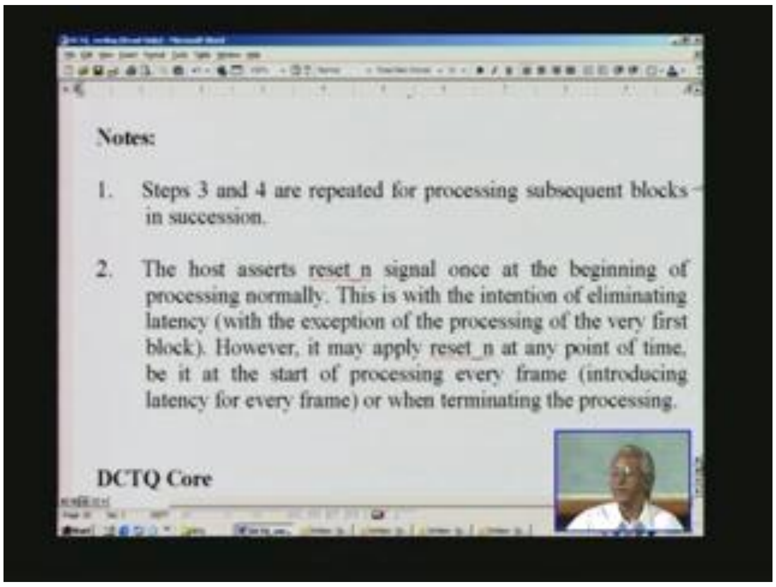
Notes:

1. Steps 3 and 4 are repeated for processing subsequent blocks in succession.

2. The host asserts reset_n signal once at the beginning of processing normally. This is with the intention of eliminating

The fourth step is the last step for the first one host. If ready signal is set now, it will always be high which means that it is always ready. DCTQ processor is always ready to receive the input image except for few clock cycles. If ready signal is set now, next block of image data is written into another block of the 64 byte dual RAM. Otherwise, you wait for that. This is the role of the host. This wait can be either bold mode or interrupt mode as far as the host processor is concerned, preferably the latter.

(Refer Slide Time: 42:28)



Notes:

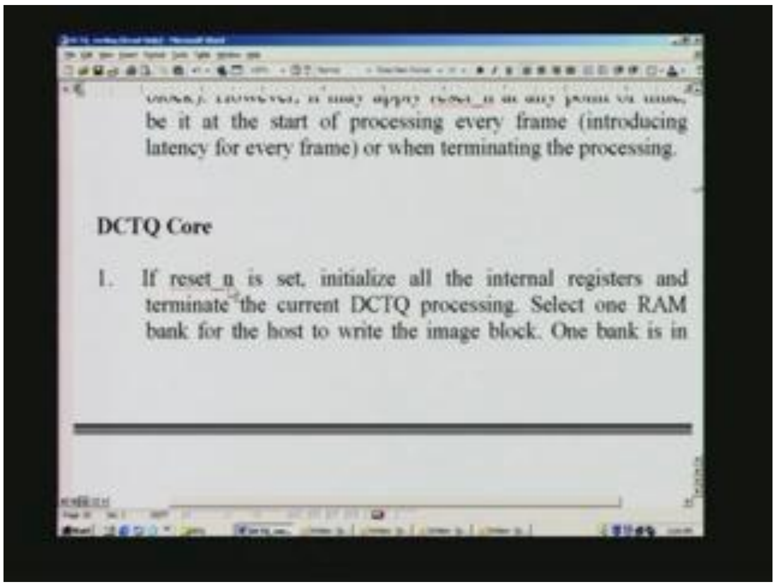
1. Steps 3 and 4 are repeated for processing subsequent blocks in succession.

2. The host asserts reset_n signal once at the beginning of processing normally. This is with the intention of eliminating latency (with the exception of the processing of the very first block). However, it may apply reset_n at any point of time, be it at the start of processing every frame (introducing latency for every frame) or when terminating the processing.

DCTQ Core

There are two notes here. Steps 3 and 4 are the last steps we have seen are repeated for processing subsequent blocks in succession. So you have to keep on churning the same thing and block after block you have to keep on writing into and DCTQ will be processed simultaneously. The host asserts reset n signal once at the beginning of processing normally. This is with the intention of eliminating latency with the exception of the processing of the very first block. The very first block has latency of 45 clock cycles which we have just seen in the architecture. It will not be there in subsequent blocks because every clock pulse actual data is being processed. If you take a big picture such as what we have seen earlier we have some 1024 by 768 which will work out to be thousands of blocks, each block being 64 bytes. Naturally, when compared to one frame wherein thousands of blocks are there latency of 45 clock cycles is a negligible thing. Therefore, I said you can even apply a reset at every frame; even then it will not make any significant delay of processing. However, it may apply reset n at any point of time, be it at the start of processing every frame, introducing latency for every frame or when terminating the processing. Finally, if you wish to terminate the entire processing, image processing or video processing, then you need to do the assertion.

(Refer Slide Time: 44:07)



block). However, it may apply reset_n at any point of time, be it at the start of processing every frame (introducing latency for every frame) or when terminating the processing.

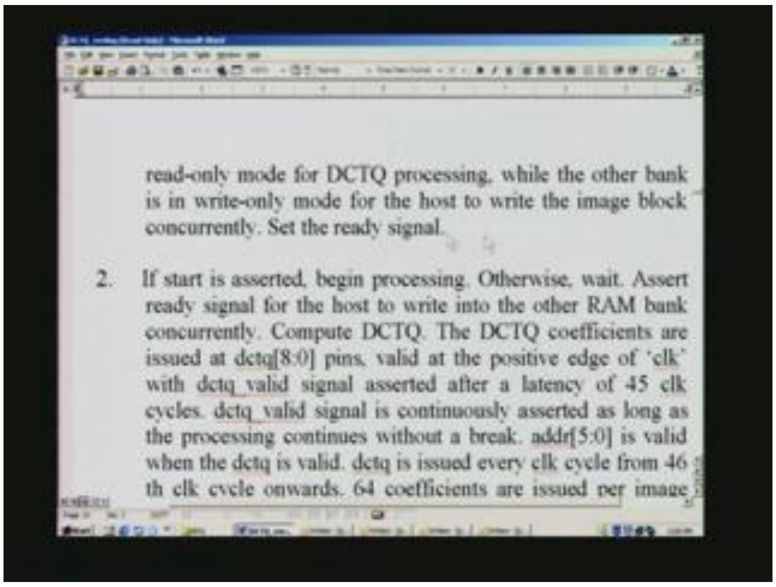
DCTQ Core

1. If reset_n is set, initialize all the internal registers and terminate the current DCTQ processing. Select one RAM bank for the host to write the image block. One bank is in

This is the role for DCTQ processor, which we will look into the verilog code later on. If reset is set, initialize all the internal registers and terminate the current DCTQ processing. For example, it was already processing something. The moment reset is applied it will abandon the DCTQ processing and initialize itself. Select one RAM bank for the host to write the image block. This is by the control rnw and it does that.

One bank is in read only mode for DCTQ processing while other bank is in write only mode for the host to write the image block concurrently. Set the ready signal. We immediately set the ready signal so that next block can be written fast. Note that you need just eight clock cycles for writing by the host. So it can attend to other course the rest of the time.

(Refer Slide Time: 44:54)

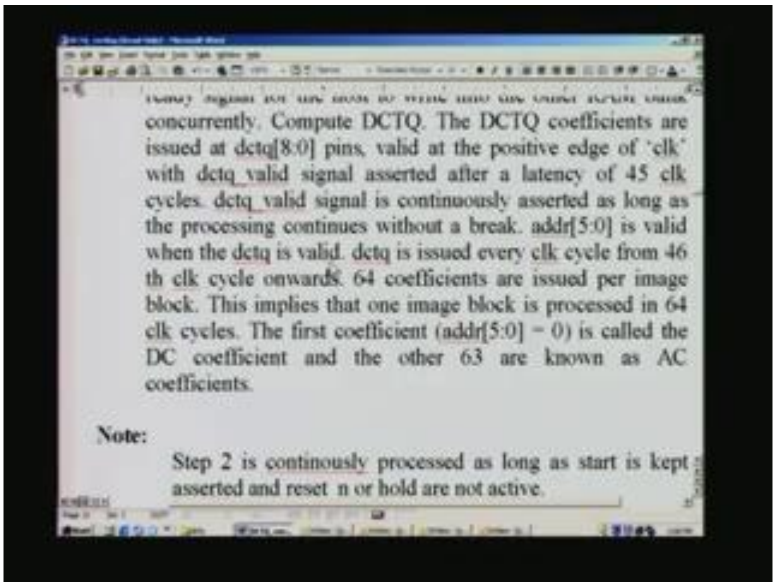


read-only mode for DCTQ processing, while the other bank is in write-only mode for the host to write the image block concurrently. Set the ready signal.

2. If start is asserted, begin processing. Otherwise, wait. Assert ready signal for the host to write into the other RAM bank concurrently. Compute DCTQ. The DCTQ coefficients are issued at dctq[8:0] pins, valid at the positive edge of 'clk' with dctq_valid signal asserted after a latency of 45 clk cycles. dctq_valid signal is continuously asserted as long as the processing continues without a break. addr[5:0] is valid when the dctq is valid. dctq is issued every clk cycle from 46 th clk cycle onwards. 64 coefficients are issued per image

The second point is that if start is asserted, begin processing, that is the DCTQ processing you start here, otherwise wait. Assert ready signal for the host to write into the other RAM bank concurrently. Otherwise, if you do not assert the ready signal host cannot write. That is why you have to do it as quickly as possible. Then start computing DCTQ. The DCTQ coefficients are issued at DCTQ 8 0; this 9 bit pins valid at the positive edge of clock. This clock is the system clock for DCTQ, with DCTQ valid signal asserted. When DCTQ becomes valid, simultaneously DCTQ valid also goes high, both simultaneously. This will be valid at the positive edge of the clock. After one clock delay only you can register in any other subsequent processing which still holds well because it changes value only at the arrival of clock. Right at the fag end of the clock also you can register into the external world, external to DCTQ processor. DCTQ coefficients are issued at these pins valid at the positive edge of the clock with DCTQ valid signal asserted after latency of 45 clock cycles. This is what we have explained earlier. DCTQ valid signal is continuously asserted as long as the processing continues without a break. Address is valid when the DCTQ is valid.

(Refer Slide Time: 45:05)



[illegible]
concurrently. Compute DCTQ. The DCTQ coefficients are issued at dctq[8:0] pins, valid at the positive edge of 'clk' with dctq_valid signal asserted after a latency of 45 clk cycles. dctq_valid signal is continuously asserted as long as the processing continues without a break. addr[5:0] is valid when the dctq is valid. dctq is issued every clk cycle from 46 th clk cycle onwards. 64 coefficients are issued per image block. This implies that one image block is processed in 64 clk cycles. The first coefficient (addr[5:0] = 0) is called the DC coefficient and the other 63 are known as AC coefficients.

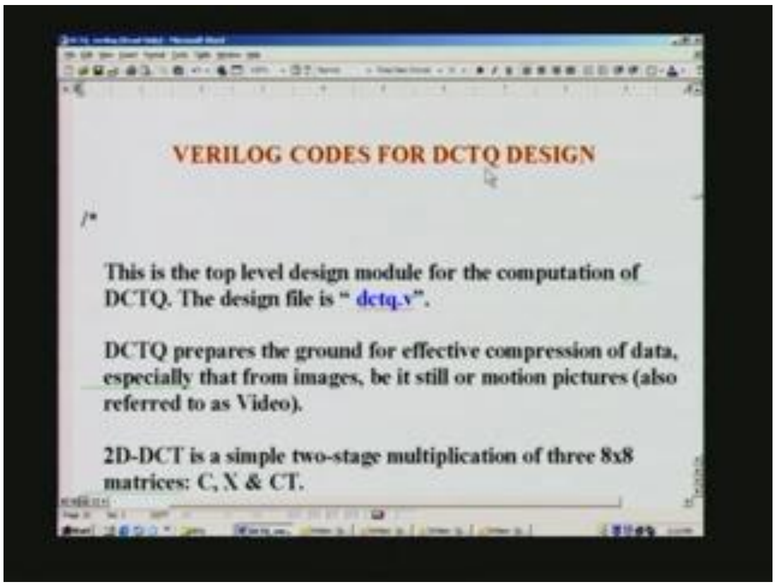
Note:

Step 2 is continously processed as long as start is kept asserted and reset_n or hold are not active.

DCTQ is issued every clock cycle from forty sixth clock cycle onwards which we have seen the reason, 64 coefficients are issued per image block. One block of image contains 64 pixels and 64 coefficients are coming. That means it is equivalent to saying that it is not 1 to 1 correspondence between a pixel and DCTQ coefficient. What we say 64 coefficients is 64 pixels. So we can say we have 1 to 1 correspondence. Every clock one coefficient is issued. Therefore it is equivalent to saying that one pixel is processed every clock. This is how we justify how it is processing at 1 clock rate. We have seen the entire architecture as well as the algorithm and explained how it was outputting at every clock. It is doing so for the simple reason that for every clock we apply the input data. Fresh data is applied every clock which we have already seen at the CX matrix input of the first multiplier. Therefore, whatever we apply will have to come out at the same rate otherwise everything will go completely in an array. 64 coefficients are issued per image block. This implies that one image block is processed in 64 clock cycles that is what we have explained just now; 64 coefficients in 64 clock cycles. Once again, there are 64 pixels so you can say that one pixel is processed at the rate of 1 clock cycle. The first coefficient, that is, the address 0 is called the DC coefficient and the others as AC coefficients. Others are 63 in number. We have a note here for step 2. We have seen that this is a repeated DCTQ computation. We have to keep repeating this block after block. Step 2 is continuously processed as long as start is kept asserted and reset or hold is not active. If there is a hold, this will be suspended; if it is reset, the whole system will be reset. And one more thing is not present, I mean, of course, it is covered in this item 2, start must be asserted there. This completes the intercommunication between the host and the DCTQ processor.

If you have any questions we can answer them if your questions pertain to what we have covered so far because we are going into the verilog codes shortly.

(Refer Slide Time: 49:27)



VERILOG CODES FOR DCTQ DESIGN

/*

This is the top level design module for the computation of DCTQ. The design file is " dctq.v".

DCTQ prepares the ground for effective compression of data, especially that from images, be it still or motion pictures (also referred to as Video).

2D-DCT is a simple two-stage multiplication of three 8x8 matrices: C, X & CT.

Thank you.