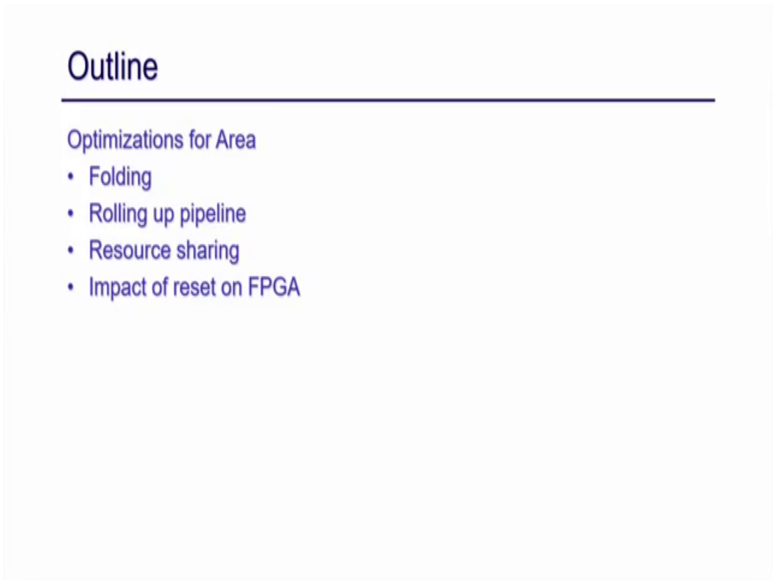
Optimization Techniques for Digital VLSI Design

Dr. Chandan Karfa
Dr. Santosh Biswas
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 08
RTL Optimizations for Area

Welcome everyone. So, today we are going to discuss on RTL optimizations for area. So, in our last discussion we have discussed about several techniques which improves the timing of it of a design or say throughput of a design or the latency input the latency of a design. So, we discussed about certain strategies. Today discussions topic is something on area. What are the kinds of techniques that we can commonly used on RTL which actually improve the area?

(Refer Slide Time: 00:56)

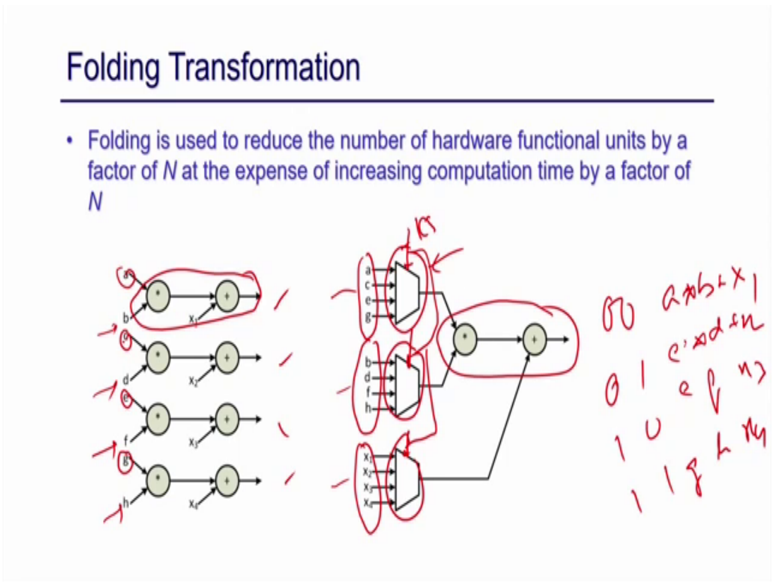


Outline

Optimizations for Area
- Folding
- Rolling up pipeline
- Resource sharing
- Impact of reset on FPGA

So, specifically we are going to discuss about strategies like folding rolling up pipelining resource sharing and some reset impact of reset, but specifically on FGPA targets ok.

(Refer Slide Time: 01:10)



Folding Transformation

- Folding is used to reduce the number of hardware functional units by a factor of *N* at the expense of increasing computation time by a factor of *N*

So, let us start with folding. So, what is folding transformation? So, whenever we design certain things we do not keep in mind that similar kind of structure is generated many places right and they are all consume areas. It may be possible in certain scenarios that those kind of patterns can be clubbed together and can be used only one component 1 module of the same structure which reduce the area by enlarge ok.

Let us take this example suppose I am doing here this multiply and add multiply and in parallel. So, there are 4 such components in your design you just copy paste from it is a different part of the design this has and this also in if you just think about the hardware 4 multiplier and the 4 hardware and these are all in running a parallel.

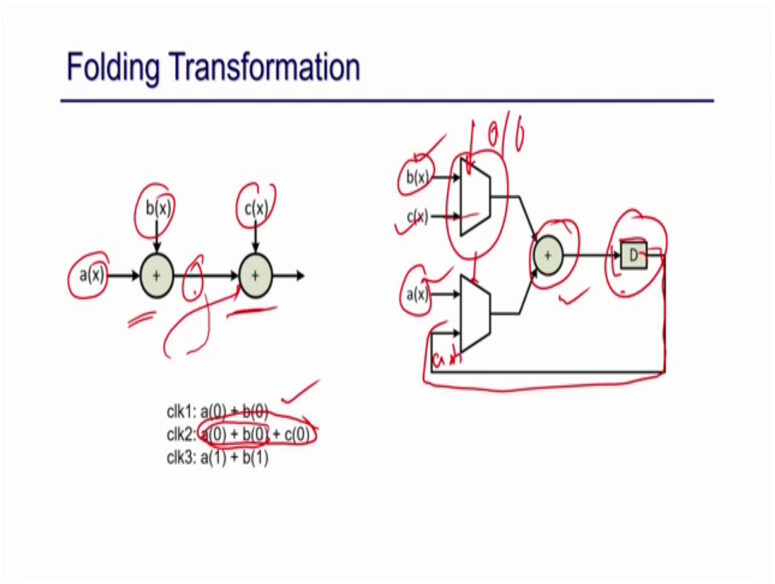
So, what we can do I identify this MAC pattern I multiply and accumulate pattern and I just use only one of the pattern in your design right and now this input should be time multiplexed. So, this a input the left input a b a c e and g will be clubbed together as a left input to a multiplexer and the right input b this b d f and g will be clubbed together as the right input of this multi player and then this x 1, x 2, x 3, and x 4 will be time invariant multiplexed again to this atom right.

And this will be now controlled by this control signal same control signal say c s and this is 2 bit right. So, this is c s and if it is 0 0 I am going to choose a b a b and x 1 right. So, that a plus b into x 1 is carried out then if it is 0 1 I am going to choose this c and d here c here here d and here x 2. So, that c plus c a into b plus 1 here c into d plus x 2 is carried out. So, similarly 1 0 and 1 1 I am going to choose e and f x 3 and I am going to choose g h and x 4 right. So, this is something in 4 clocks I am going to do this right.

So, we can understand that here your area will be reduce by multiple of 4 right. So, factor of 4 because I try to plug 4 such patterns into 1 right and now if you can find out some k such pattern your area will be it is proportional to k times. Of course, there are certain things has to be taken care because this multiplier is adding.

But usually this kind of pattern that we find out this is a big pattern and consists of the complex operations which is clubbed together so that extra area overhead of this multiplier is kind of a negligible compared to the things that we are going to merge ok. So, this is the order floating folding structure.

(Refer Slide Time: 04:12)


Folding Transformation

clk1: a(0) + b(0)
clk2: a(0) + b(0) + c(0)
clk3: a(1) + b(1)

So, there are certain things so this is where I give an example at this 4 upper structure running in parallel I just clubbed together, but it is also possible to club operator that are in series right. For example, I have this 2 adder in series I can club them into 1 adder in the circuit then what I have to do? I have to use this a this a and b as the input here.

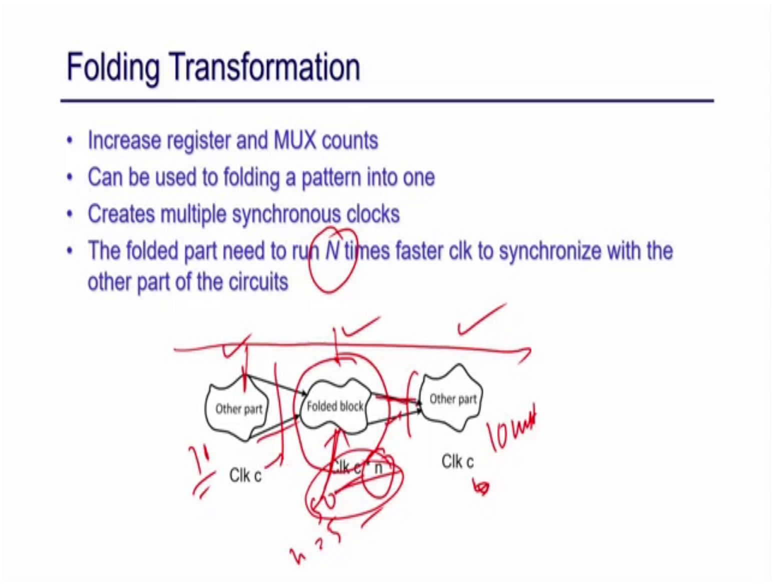
So, this is a is the left input of this multiplier and b is the 1 input and that the c will be another input for this so I will use a multiplier and whatever the results here that has to feed back to this adder right. So, this is what I am doing. So, I just put a register here and this output is feeding them right.

So, what happened? So, in first cycle this a 0 and b 0 will be calculated in the second cycle this a 0 is gradually is already stored in these registers and now this c 0 will be. So, c 0 will be selected here and then a 0 plus c 0 will be calculated. Again in the so this is 1 weight signal 0 and 1. So, 0 is just you select b and a in the second cycle it will select the c x and this a 0 to b 0 right. So, this is how the whole thing will work.

So, every 2 cycle this whole this is a 0, b 0, c 0 the output will come right in every 2 cycle ok. So, this is the both are possible if they are in parallel you can merge if you they if they are in the series then also you can merge them, but you need a feedback register so that you can reduce that intermediate results as well ok.

So, this is the overall folding transformations and. So, what are the other issues has to be considered here?

(Refer Slide Time: 05:49)



Folding Transformation

- Increase register and MUX counts
- Can be used to folding a pattern into one
- Creates multiple synchronous clocks
- The folded part need to run N times faster clk to synchronize with the other part of the circuits

So, the important factor is that here. So, suppose this is the your folded structure right, but this is not the only component of your design. So, this is just a part of your design and it is synchronous with the other part of the design ok. So, now, earlier what is happening? Since this all running in parallel or in the series there is no delay 1 clock you going to get the results right.

So, now since this is taking multiple clock then your this results whatever the input is coming here it you have to now at first say 3 cycle, 4 cycle to get the output right. So, either you have to if you want to do if you want to run all this component in the same clock then you have to add some synchronous synchronization mechanism. So, that whatever the data is expecting after 1 clock now it will wait for 4 clocks.

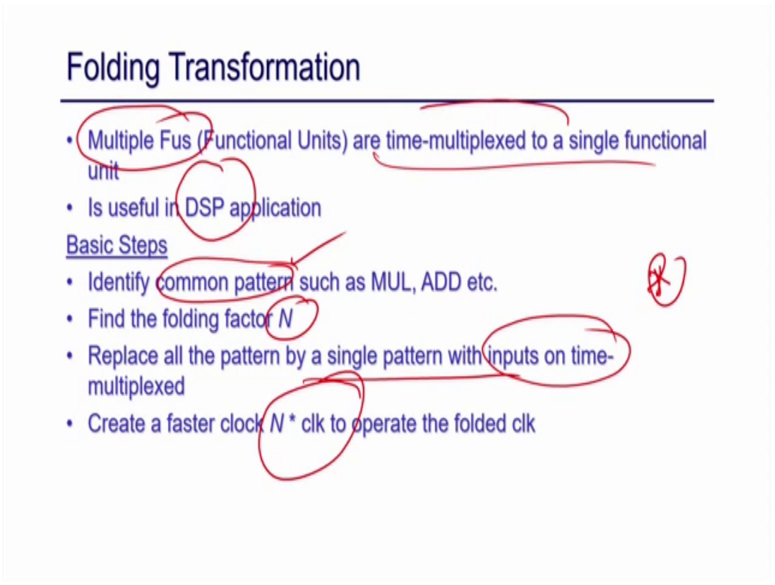
Because now you are things are folding factor is 4. So, after 4 cycle only the data will be available. Similarly this may be feeding in data every clock now it has to wait for hold the data for all 4 clock cycle so that the operation can be executed here right. So, there are 2 choices either you add all the synchronization factor you want to design or the alternative choice is that you run this particular block the folded block N time faster where my folding factor is N.

So that means, N component is multi if margin to a single block. So, now, I can actually run these particular things in clock into N. So, if the clock is say here 10 and say folding factor N equal to say 5. So, this is be 50 megahertz and this is also 10 megahertz right. So, what will happen? So even though 1 clock of this is equivalent to the 5 clock so effectively I am running in 5 blocks but since this clock is very fast it will be automatically synchronized with the other blocks right.

This is the common way of doing this. So, whenever you fold something you increase that clock of that particular block by a factor of folding factor which is N. So, that your synchronizing will be automatically taken care, but is that data will be seamlessly go through this components right because it is just for him this is 1 cycle, but effectively this is 5 cycle internals. So, all these things will be properly synchronized.

So, during folding these are the two things either you have to synchronize manually or you just increase the clock freeze of the folded structure. So, that it can be automatically synchronized with the other component of your design. So, these are this is the two factor you have to take care of ok.

(Refer Slide Time: 08:10)



Folding Transformation

- Multiple Fus (Functional Units) are time-multiplexed to a single functional unit
- Is useful in DSP application

Basic Steps

- Identify common pattern such as MUL, ADD etc.
- Find the folding factor N
- Replace all the pattern by a single pattern with inputs on time-multiplexed
- Create a faster clock N * clk to operate the folded clk

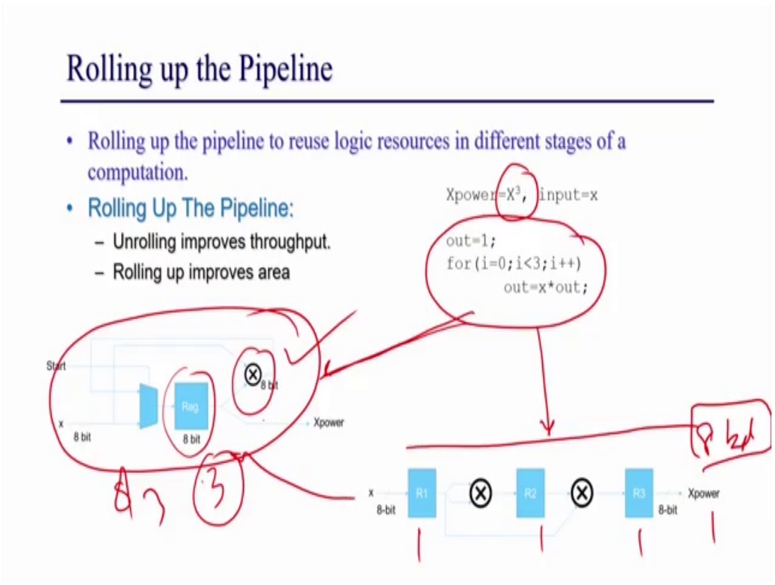
And this fold folding transformation usually apply for this complex operations like multiplier and is basically time multiplex. So, the inputs are time multiplex and it is very useful for DSP application where actually you try to input maximum DSP, DSP in the sense DSP is a specific component in FGPA how which actually can do multiplication very first and it also have some adder and accumulation option pre adder as well as accumulation option.

So, you can actually find out a multiplier accumuler multiplier and accumulate or add and multiply then accumulate this kind of structure you can figure out in your design and you can actually use the same DSP to calculate that same thing. So, this is something is useful in DSP applications and if you just think about the automating this step you have to find out the first pattern right the common pattern as big pattern you can find out that is beneficial.

Because then the accumulate resource this area improvement will be by that factor you try to even if you find say on multiplier. So, there is 1 multiplier for this design you can just find out this multiplier 4 multiplier and you can multiplex this 4 multiplier right, but if you just find and multiply and then actually you can replace all this 8 by 2 right earlier only 4 multiplier by 1 multiplier. So, you are actually saving is more. So, as big pattern you can figure out that will be beneficial for your design right.

So, you have to first figure out the common patterns then this folding factor defined by the number of patterns you have finally, identified then replace all this pattern by a single pattern with inputs are time multiplex; that means, you just add multiplexers and the control signal properly and you apply a faster clock of N time faster clock to that folded circuit so that the whole thing. So, work smoothly ok. So, this is all about this folding transformations.

(Refer Slide Time: 09:57)



Rolling up the Pipeline

- Rolling up the pipeline to reuse logic resources in different stages of a computation.
- Rolling Up The Pipeline:
  - Unrolling improves throughput.
  - Rolling up improves area

```
Xpower=X³, input=x
out=1;
for(i=0;i<3;i++)
    out=x*out;
```

Now, we will move in to the next topic which is that pipelining also rolling up the pipelining is something you improve the area right.

So, this is the same example I discuss in one of the previous discuss and now we are talking about this improvement of the power is this improving the throughput so that x x cube design right. So, you can do it iteratively or you can actually you can actually this apply you can do it in pipeline manner that the way I just discuss in the in previous class that you can do this all operation pipeline so that your throughput is 8 bit per cycle, but you can roll this pipeline into 1 which is that iterative percent of this.

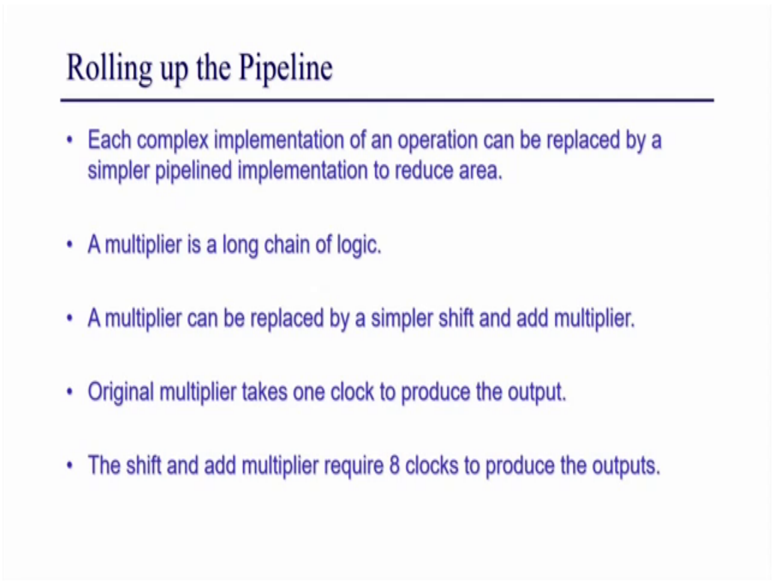
So, I have only 1 multiplier now instead of 2 multiplier, and 3 register I have now 1 multiplier into 2 1 register right. So, if we just roll this pipeline this is just the opposite operation. So, this is this is the iterative implementation and this is the pipeline implementations.

and you can just if your area is concern then you should probably take this one, but this will take 3 cycle right because your latency will increase because your throughput will be less because now we are actually producing 8 bit in 3 cycles. So, it will be 8 by 3 and your latency also 3 cycle because in every 3 cycle you are producing one output in this pipeline design and this all this component running in parallel we have shown that is our latency is also 8 bit and later sorry this throughput is 8 bit per clock and your latency is 1 right.

So, sorry latency is 3 bit, but effectively your throughput is 8 bit per clock rate because there are 3 latency is there, but your area is less here because if we just do this iterative percents ok. So, this is what is called pipelining. So, if you have a pipeline design really care out about your area, your design is very big and even saving this two multiplier is useful you can probably do this implementation instead of this right.

But again if I do this you have some others synchronization like finish because now it will take 3 cycles. So, you there are some finish signal just to cause synchronize with the other components ok. So, this is something it is your choice it based on your if your area is I mean crucial factor then you are going to choose this and if you want a throughput is important then you are going choose this 1 ok.

(Refer Slide Time: 12:20)

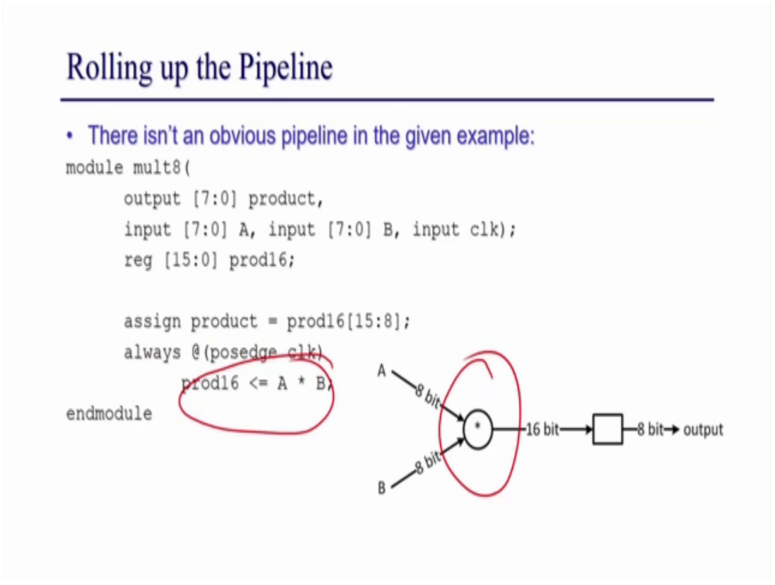


Rolling up the Pipeline

- Each complex implementation of an operation can be replaced by a simpler pipelined implementation to reduce area.
- A multiplier is a long chain of logic.
- A multiplier can be replaced by a simpler shift and add multiplier.
- Original multiplier takes one clock to produce the output.
- The shift and add multiplier require 8 clocks to produce the outputs.

So, there is another aspect is also here is sometime your design may be there is no pipeline nothing, but there are some complex operation like multiplier, but you can think about as a alternating operation of a multiplier which will actually give you better benefit right. So, that is something also it is possible for example, if you just think about this your implementation has only these multiplications A into B.

(Refer Slide Time: 12:38)



Rolling up the Pipeline

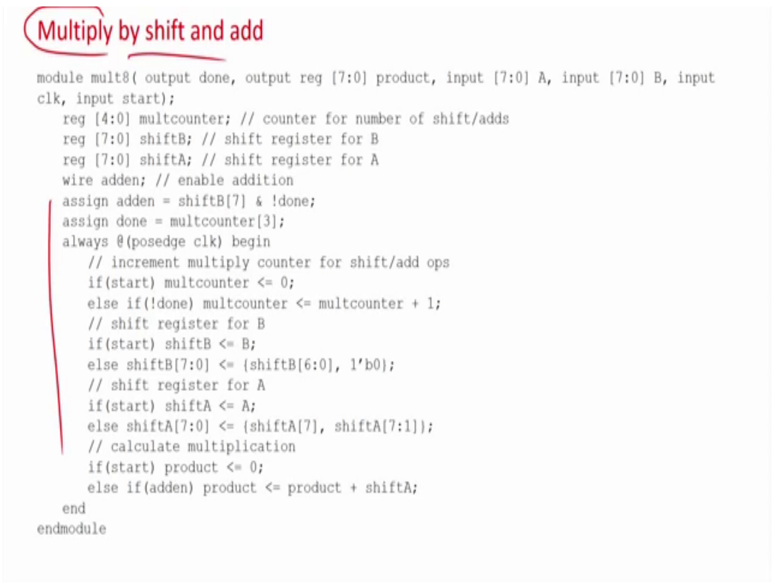
- There isn't an obvious pipeline in the given example:

```
module mult8(
      output [7:0] product,
      input [7:0] A, input [7:0] B, input clk);
      reg [15:0] prod16;

      assign product = prod16[15:8];
      always @(posedge clk)
            prod16 <= A * B;
endmodule
```

So, you are effectively have a multiplication which we assume there is a big complex implementation. So, because it will be finally, mapped to logic right and it is take lot of gates.

(Refer Slide Time: 12:54)

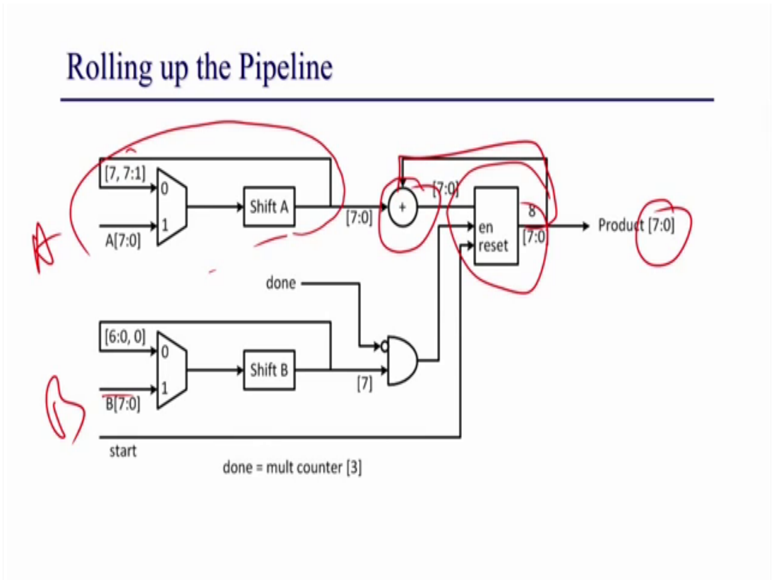


Multiply by shift and add

```
module mult8( output done, output reg [7:0] product, input [7:0] A, input [7:0] B, input
clk, input start);
   reg [4:0] multcounter; // counter for number of shift/adds
   reg [7:0] shiftB; // shift register for B
   reg [7:0] shiftA; // shift register for A
   wire adden; // enable addition
   assign adden = shiftB[7] & !done;
   assign done = multcounter[3];
   always @(posedge clk) begin
      // increment multiply counter for shift/add ops
      if(start) multcounter <= 0;
      else if(!done) multcounter <= multcounter + 1;
      // shift register for B
      if(start) shiftB <= B;
      else shiftB[7:0] <= {shiftB[6:0], 1'b0};
      // shift register for A
      if(start) shiftA <= A;
      else shiftA[7:0] <= {shiftA[7], shiftA[7:1]};
      // calculate multiplication
      if(start) product <= 0;
      else if(adden) product <= product + shiftA;
   end
endmodule
```

But if you just think about I am going to implement multiply by shift add right. So, then what is happening you just shifting your multiplicand by every time and you just add it based on the multiplier value that is 0 or 1 right.

(Refer Slide Time: 13:09)



Rolling up the Pipeline

So, if you just abstract diagram like this. So, your every cycle you are going to shift this your A’s. So, you are doing A into B right this is A into B and if you know that particular bit is 1, then I am going to shift and I am going to add to the results that that this is my partial results multiplication results. So, as the otherwise I just discard this shifting value right. This is very well way well known way of calculating multiplication using as a shift register and a adder instead of a multiplier.

But it will take 8 clock right because every clock you are just shifting 1 bit and you calculate on these things. So, if you just assume [FL] my data path which is 8 bit then it will take 8 cycle at every cycle I am shifting by 1 bit and I am going to add and then again I am shift by another bit and then am going to add and this will go for 8 cycle and finally, the result will be generated right. But you can see that I can replace that big multiplier by a simple adder right. So, it will you give you a big area benefit.

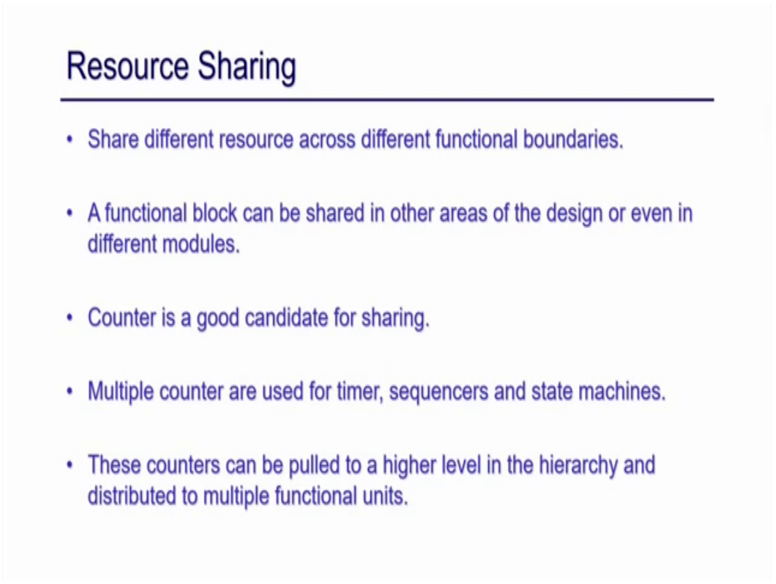
But again it is basically become a pipeline for a iterative percent. So, it will take 8 cycle, but here the multiplier is going to take only 1 cycle ok. So, it is basically again I am just repeating the same statement that it is all design choice if you want to just achieve as a low area as possible probably and your latency does not matter, because then probably you can replace this multiplier by a simpler version of multiply and by add multiplayer which will take 8 cycle right your latency will become 8 where as your latency here is 1 ok.

So, this is something also possible and sometime you have to do this kind of optimization just to achieve your target area because may be in your FGPA target your design is not get fitted right where you need this kind of little bit changes in your design. So, that the whole things get placed in your FGPA, because in FGPA the number of DSP blocks are fixed and the logic that LUT's and the logic units are fixed.

So, if you have very large number of multiplier you probably not able to fit all of them in the DSP blocks of the logic units where as you can just replace a multiplier by add and which will take less number of LUT's and probably you can probably map it to the design.

So, sometime this kind of corner case might arise and you have to do this kind of tricks, so that you can actually map your design into a target FGPA devices ok.

(Refer Slide Time: 15:37)

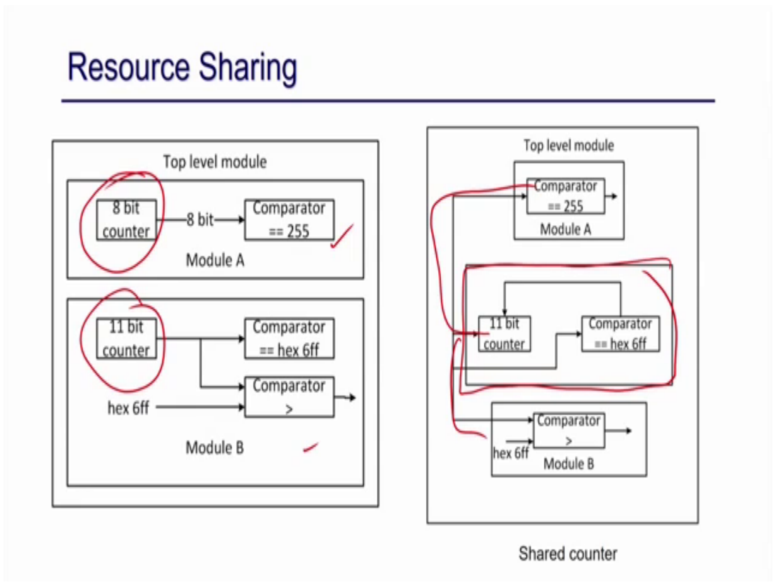


Resource Sharing

- Share different resource across different functional boundaries.
- A functional block can be shared in other areas of the design or even in different modules.
- Counter is a good candidate for sharing.
- Multiple counter are used for timer, sequencers and state machines.
- These counters can be pulled to a higher level in the hierarchy and distributed to multiple functional units.

So, this is about the rolling up pipelining I am going to talk about the next technique is called resource sharing. Resource sharing is something is again it is kind of folding, but is in different manner is like if you have some resources is use many places right it is not folding into one, but some of the resources I can actually share it is kind of folding.

(Refer Slide Time: 16:00)



Resource Sharing

Shared counter

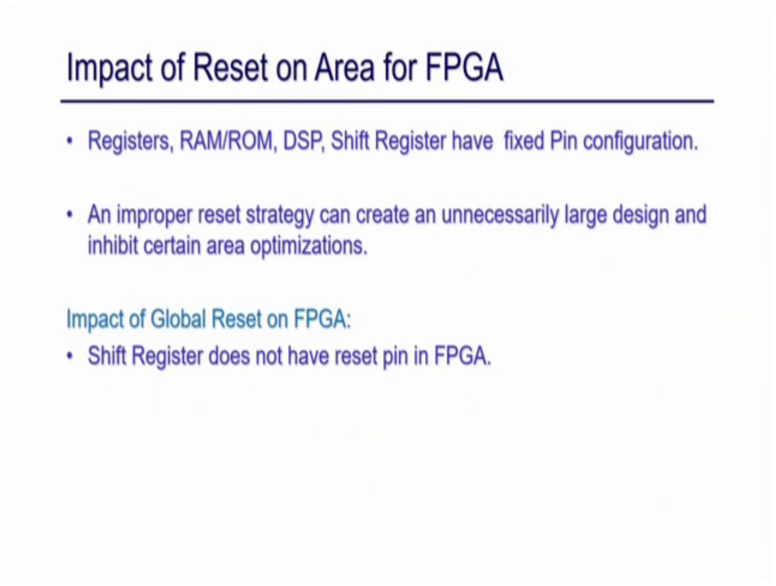
But it is little bit different in the sense that that the structure may not be exactly same. But I can reuse that component it is specifically applicable for counter if you just take this example suppose I have 2 module, where I have 2 counter right 11 bit counter and 8 bit counter.

So, what I can do? I can just bring out this counter from this 2 module I can you have a dedicated counter module which is 11 bit counter I can use the 8 lsb as the counter of this and the all 11 bit as a counter of this module right.

So, now I have a one counter module which is reused in the both the model. So, I bring out this 2 counter out of this and we create a dedicated counter module and I can use this right. So, this is what is called resource sharing. So, it is kind of finding out some most similar type of module means similar type of operations is happening being multiple hierarchical modules. You figure out bring it out them and do them once and you just use that particular output in all the modules right. So, that is what is called resource sharing.

So, that is something sometime we usually do because that is also give you good idea because here I just give a one example where for counter is very common to your design maybe there are say 100 counters and that is there in different different module and just doing this in 100 times is lot of area right, but you can just do only once and I can use it for all the module. So, this is the kind of optimization that will give you very good idea benefits ok. So, that is also an technique you should always remember when you are going to do this area optimization.

(Refer Slide Time: 17:37)



Impact of Reset on Area for FPGA

- Registers, RAM/ROM, DSP, Shift Register have fixed Pin configuration.
- An improper reset strategy can create an unnecessarily large design and inhibit certain area optimizations.

Impact of Global Reset on FPGA:

- Shift Register does not have reset pin in FPGA.

Now, we are going to talk about several strategies specifically for FGPA target ok. So, this impact of reset on area for FGPA so in as you remember or know that FGPA has very fix kind of structure right it has set of CLV's CLV is nothing but this configurable logic unit which logically it is consider the LUT's which is the you can you can do map any kind of combinational circuit into that lookup table in a base logic units it has also some your CLV which consist of memories which is basically register flip flops and it also have some dedicated ram, ROM this RAM memory unit like RAM ROM it should also have some shift registers which is very fast having efficient way of implementing the shift registers and also it is some DSP blocks right.

And the pin configurations the problem with these the pin configuration for this DSP's or say FGPA units are fixed right for example, a RAM may have may not have any say asynchronous reset or say DSP block do not have any say synchronous set or say shift register do not have any says reset pin right. So, if you try to map your design to FGPA it is very important to understand the exact pin configuration of your design of the FGPA block.

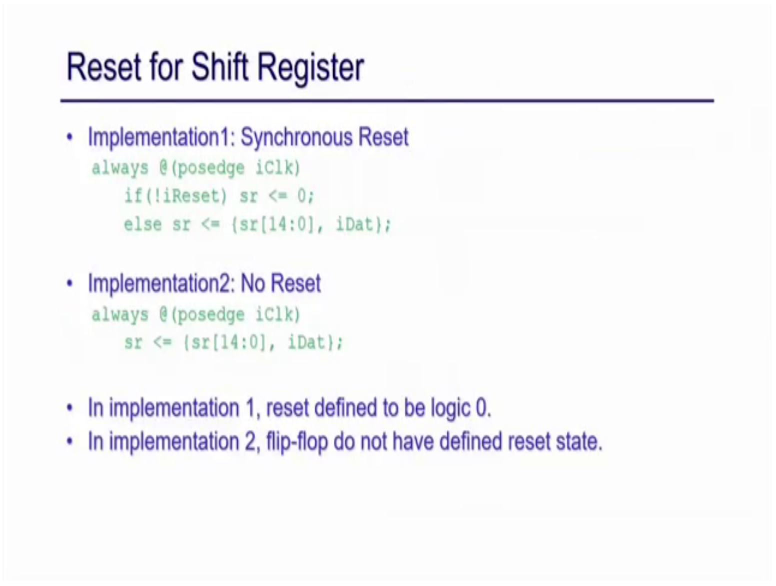
Because what happens sometime is you design something without understanding the exact structure and because only a reset pin or say synchronous reset pin or asynchronous reset pin that particular whole big RAM or say DSP cannot be mapped to the target architecture right.

For example, you try to map a big array into array into some memory or say RAM and because of that asynchronous reset it cannot be mapped to that RAM. Then there is a big problem because in the whole RAM it will map to registers and it will consume lot of registers and then your all the register when over used you cannot do other part of the register which can should be mapped to register cannot be done right.

So, this is something a very small and very partial thing we are not always bother about, but that actually create a catastrophic effect in your area and it is just because of that that particular unit the DSP unit or the shift registers or the RAM does not support that particular kind of pin and because of that it is not able to map that to the RAM or ROM it is map into the logic units logic flip flops or say CLV's or the LUT's and it has a very bad impact on your area.

So, we are going to discuss about this specific things that means, configuration and what kind of impact might happen in rest of this discussion ok.

(Refer Slide Time: 20:27)



Reset for Shift Register

- Implementation1: Synchronous Reset

```
always @(posedge iClk)
    if(!iReset) sr <= 0;
    else sr <= {sr[14:0], iDat};
```

- Implementation2: No Reset

```
always @(posedge iClk)
    sr <= {sr[14:0], iDat};
```

- In implementation 1, reset defined to be logic 0.
- In implementation 2, flip-flop do not have defined reset state.

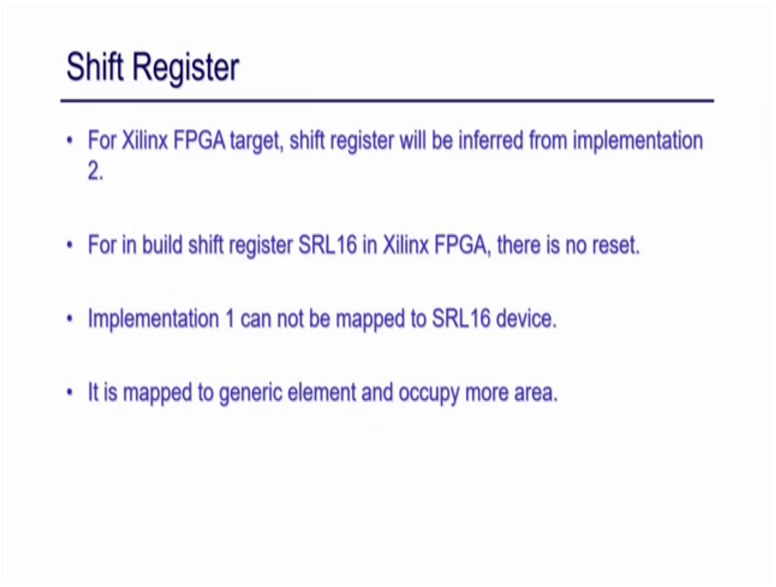
So, first we are going to discuss about the shift register and in the shift register usually do not have any reset pin shift register usually do not have any reset pin and then what happens if you just design a shift register here I just talked about a reset. So, this is the shift register right. So, I am just shifting and I am just taking 1 bit again.

Similarly, I am here this is a shifting and I am just taking a new bit. So, this is I am just shifting and just taking around bit at a time. But in this particular implementation one I have a reset and just reset I just reset the whole shift register and there is no reset here. So, this reset is defined in this design, but in this particular I do not have any reset.

So, this is a good design practice right we always make a reset, but it may be that in your design it does not matter the initial state. So, because you just plus out all your shift register component and you just store your data and then you compute something right. So, this is very common practice it does not care about the initial data component that will be it will not it has no impact on your design.

So, in that case I mean you may not define the reset right. So, even if you if you just ignore the reset you do not have any impact on your design.

(Refer Slide Time: 21:43)

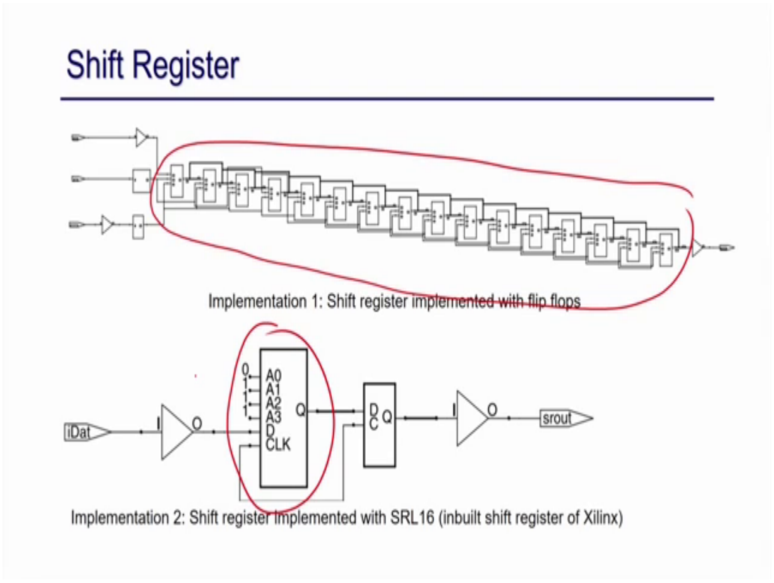


Shift Register

- For Xilinx FPGA target, shift register will be inferred from implementation 2.
- For in build shift register SRL16 in Xilinx FPGA, there is no reset.
- Implementation 1 can not be mapped to SRL16 device.
- It is mapped to generic element and occupy more area.

But the problem here is that it seems that resets the shift register do not have any reset pin. There is no reset pin in the shift register. This particular implementation cannot be mapped to the shift register dedicated register this shift register of your FGPA whereas, since this does not have any reset this will be this have the this can be mapped to the dedicated shift register of the FGPA block right.

(Refer Slide Time: 22:06)



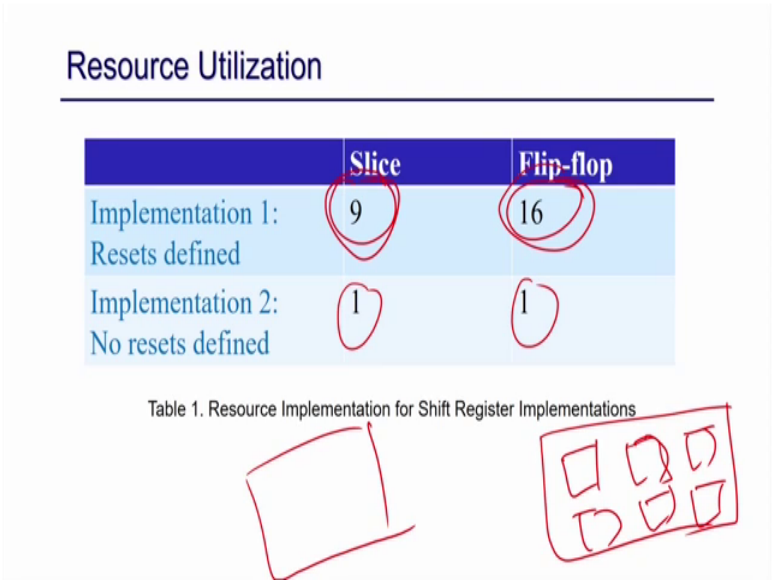
Shift Register

Implementation 1: Shift register implemented with flip flops

Implementation 2: Shift register implemented with SRL16 (inbuilt shift register of Xilinx)

So, if you just do this so what is happening the implementation one is going to be mapped into the normal flip flops which is in the CLV's and that will be connected right it is a big chain of registers where as the implementation two because it has does not have any reset pin it can be mapped to the dedicated shift register right.

(Refer Slide Time: 22:23)



Resource Utilization

	Slice	Flip-flop
Implementation 1: Resets defined	9	16
Implementation 2: No resets defined	1	1

Table 1. Resource Implementation for Shift Register Implementations

So, if you just think about the resource implementation I have only 1 flip flop here and 1 slice because this is just for the implementation we just go into that dedicated shift register Whereas this is actually map into multiple flip flops and slice.

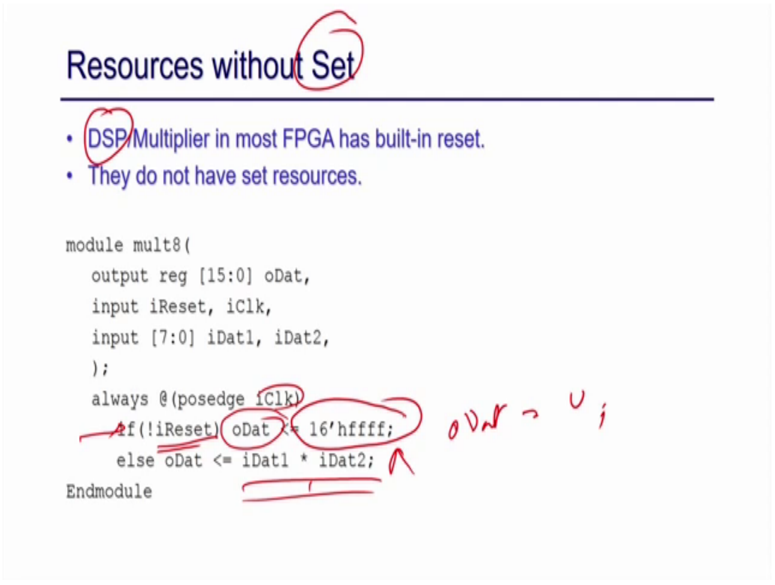
So, slice is basically in a CLV we as what I just talked about in a FGPA I have set of CLV's right. So, these are all CLV's and in CLV's configural logic blocks either I have LUT's or memories right. So, so there are two types of units inside either it has logic unit which is basically memory slice. There are two type of slice either memory slice or I have logic slice.

So, memory slice nothing, but the flip flops and this logic slices has LUT's ok. So now since I have to map it this whole 16 bit register into multiple unit 9 slices and you need 16 flip flops to implement this shift registers inside the CLV's right and in 1 CLV can a maximum 2 units either 1 2 logic units or 1 memory unit, 1 logic unit or 2 memory unit right. So, this is something inside the CLV.

So, you need 9 slices and 16 flip flops because this is not map it to shift register where it is just store into 1. So, this is something you can understand that your area is getting I mean area will be over spot this if you have a as a reset right. So, this is something we sometime we do not care about this sometime you need this, but most of the time I do not have the defined reset state because it does not matter to your design.

So, probably this is a better choice because that will result it in finding a shift register instead of a mapping it to the normal registers of your design. I mean of your FGPA ok.

(Refer Slide Time: 24:08)



Resources without Set

- DSP/Multiplier in most FPGA has built-in reset.
- They do not have set resources.

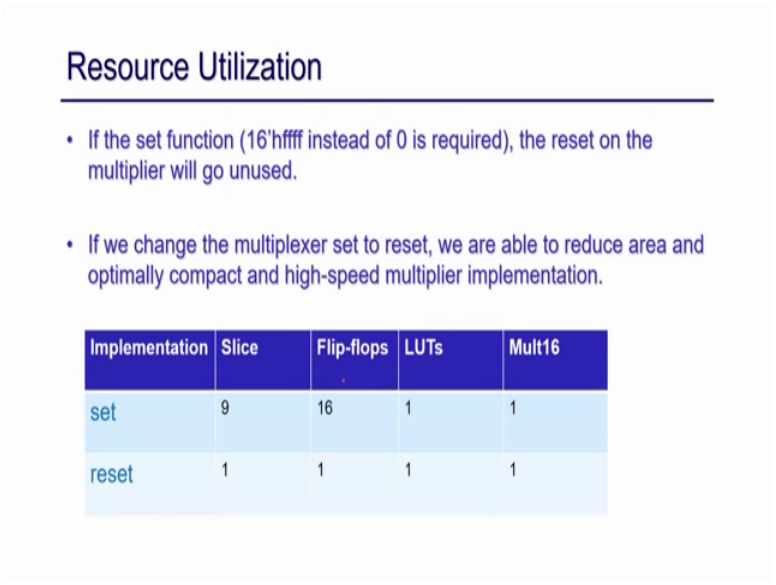
```
module mult8(
   output reg [15:0] oDat,
   input iReset, iClk,
   input [7:0] iDat1, iDat2,
   );
   always @(posedge iClk)
     if(!iReset) oDat <= 16'hffff;
     else oDat <= iDat1 * iDat2;
Endmodule
```

So, I will move into the next issue like in DSP resource without the set. So, DSP unit of your FGPA does not have any set it has reset, but it does not have any set. So, now, I am going to set some registers right. For example, in this design what I just do the multiplication so my primary interest is to map this multiplier to DSP because DSP is very high speed multiplier.

So but it has a set that I am going to set some value to this register right under certain condition. So, this is synchronous set right because it is not this I reset is synchronous to this clock. Because this is because this is not coming another which is not asynchronous to this clock right even it is a synchronous set, but it is not because I am setting this. So, this is not possible to map into this exact the whole these things into single DSP.

but if it is reset if I just write this oDat equal to equal to 0 say, then this whole thing is going to map it into a single DSP whereas, because of this has to be carried out outside right ok.

(Refer Slide Time: 25:11)



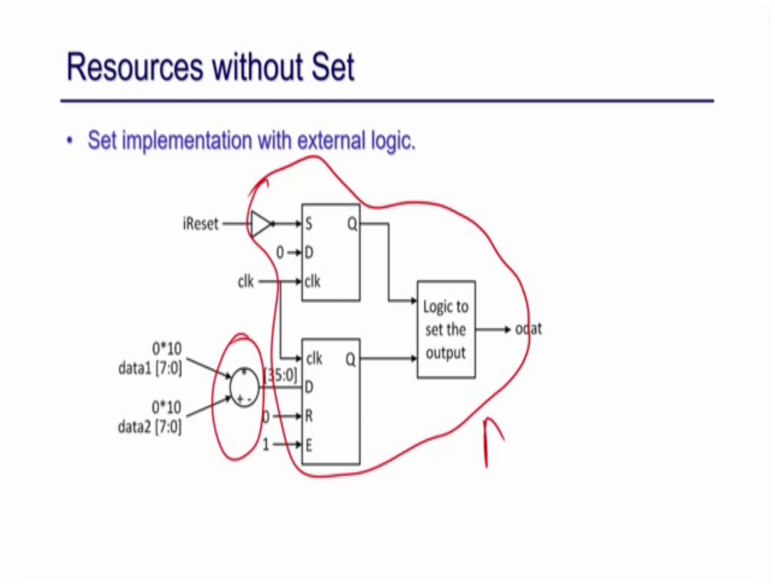
Resource Utilization

- If the set function (16'hffff instead of 0 is required), the reset on the multiplier will go unused.
- If we change the multiplexer set to reset, we are able to reduce area and optimally compact and high-speed multiplier implementation.

Implementation	Slice	Flip-flops	LUTs	Mult16
set	9	16	1	1
reset	1	1	1	1

So, you just give an example here. So, if you just do it here. So, this is your DSP unit.

(Refer Slide Time: 15:12)



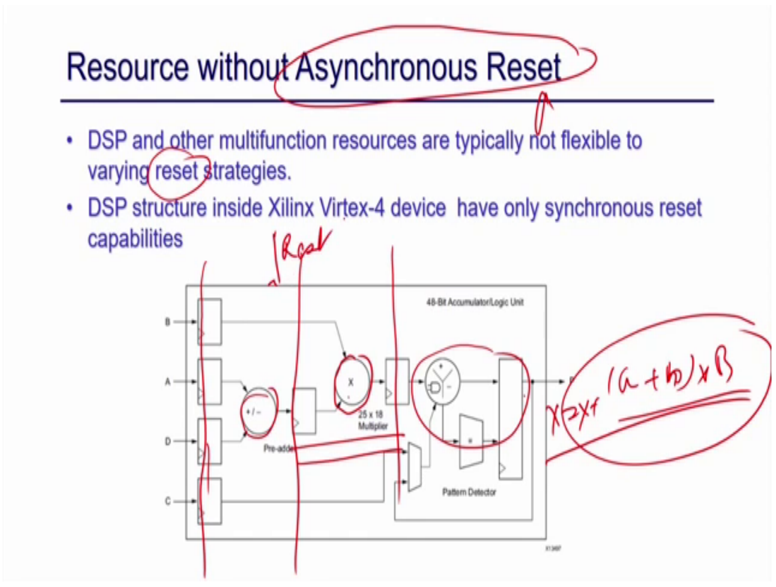
Resources without Set

- Set implementation with external logic.

Now, these setting up this 6 this setting of this value you need some additional logic here which is this. So, this logic is just setting up this multiplier results with this one right all 16 1. So, this is something is the additional resource that you require just to have the set value right. So, for example, here you just see that for the set example you need 9 slices, because this will going to map this 16 flip flop and this 1 DSP and 1 LUT and for the reset 1. Because this all this thing whole thing I just reset means you just replace this by this identical to 0 everything can be mapped into single multiplier right.

So, this is kind of you can understand that just do not having that set pin and if I set it you need a extra resource to just execute that particular set of percents in your design ok. So, this is one one issue the DSP in the set.

(Refer Slide Time: 26:11)



Resource without Asynchronous Reset

- DSP and other multifunction resources are typically not flexible to varying reset strategies.
- DSP structure inside Xilinx Virtex-4 device have only synchronous reset capabilities

Reset
48-Bit Accumulator/Logic Unit
B
A
D
C
+/-
Pre-adder
X
25 x 18 Multiplier
Pattern Detector
X=X+(a+b)×B

And the second was like the asynchronous reset right. So, again this DSP has synchronous reset, but it does not have any asynchronous reset ok. So, asynchronous reset means is that the set does not sync with your clock it can come anytime. So, so this is then it will have a problem right. So, this is something as generic DSP structure of an FGPA ok. So, we are discussing like when DSP has reset signal and it does not have any set. So, it is create a problem if you want to set it in the previous example.

But again it does not have any asynchronous reset right even if have resort reset in your design when it is not asynchronous then that it will create a problem right. So, here I give an example standard DSP unit. So, you can have a CLV's there is a multiplier, there is a pre adder, and there is a accumulator right.

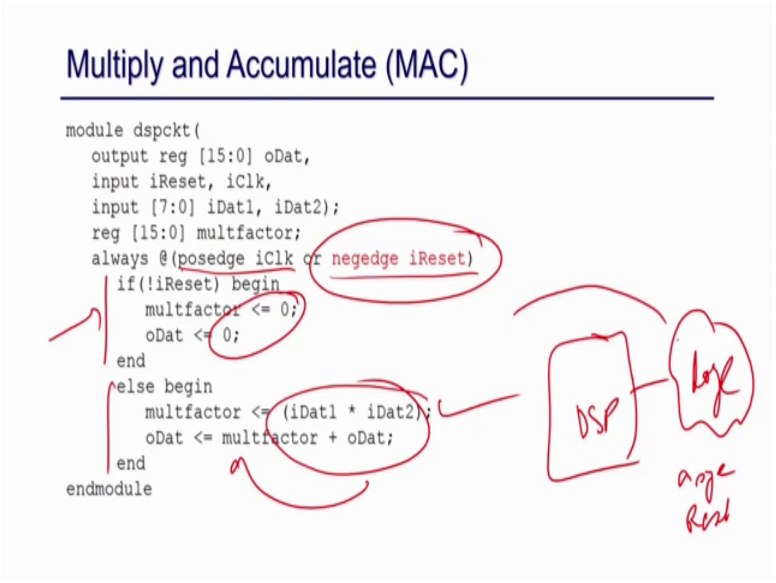
So, there is accumulator and there is a pre set of registers here, there is another set of registers here, and this is set of register here. You can see how many things you can clump just put into a single (Refer Time: 27:17) you can put a single multiplier into the d sp or multiply an accumulate into DSP you can multiple put a pre adder as well as.

So, you can just do a plus b. So, here this is really a plus d into b right you can do this also and then you can just accumulate this is basically x equal to x plus this. You can do the whole thing in the same multiplier and also not only that in this design you can have 3 set of register as well. So, you can put this so that this called delay become less right.

So, you can actually put set of register also. So, you can understand how many what is the function mean? How what are the different variations of multiplication the operations can be performed by a single DSP's it is your choice you have all these configuration.

You can choose either only multiplier multiply and add pre adders. This one set of register two set of register 3 set of register. So, all these things can be done right. But the and it is very efficient and have very fast dedicated unit right, but if you have a asynchronous reset in your design then it will create a problem right.

(Refer Slide Time: 28:20)



Multiply and Accumulate (MAC)

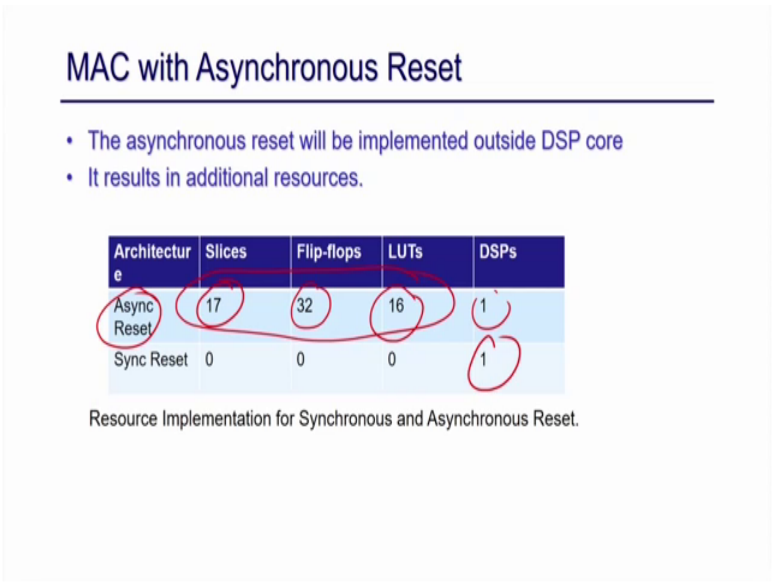
```
module dspckt(
   output reg [15:0] oDat,
   input iReset, iClk,
   input [7:0] iDat1, iDat2);
   reg [15:0] multfactor;
   always @(posedge iClk or negedge iReset)
     if(!iReset) begin
        multfactor <= 0;
        oDat <= 0;
     end
     else begin
        multfactor <= (iDat1 * iDat2);
        oDat <= multfactor + oDat;
     end
endmodule
```

For example in this multiplier I have to do this multiply and add this is MAC, multiply and then you just add it right that is what I am doing here, but I have a asynchronous reset right. So, this is posedge of clock or negative edge of ireset. So, this is asynchronous right.

So, then I am just reset because I am just resetting the multiplication factor and the output, but the problem is that since this is asynchronous reset this cannot be the whole structure because this cannot be just this reset pin cannot be just connect to the reset pin of this.

So, there may be a reset pin here you cannot be connect to the reset pin of that particular MAC because this is asynchronous. So, what will happen this particular extra thing has to be implemented outside of the DSP block. So, this can be mapped to DSP, but this has to be go outside of the DSP block.

(Refer Slide Time: 29:05)



MAC with Asynchronous Reset

- The asynchronous reset will be implemented outside DSP core
- It results in additional resources.

Architecture	Slices	Flip-flops	LUTs	DSPs
Async Reset	17	32	16	1
Sync Reset	0	0	0	1

Resource Implementation for Synchronous and Asynchronous Reset.

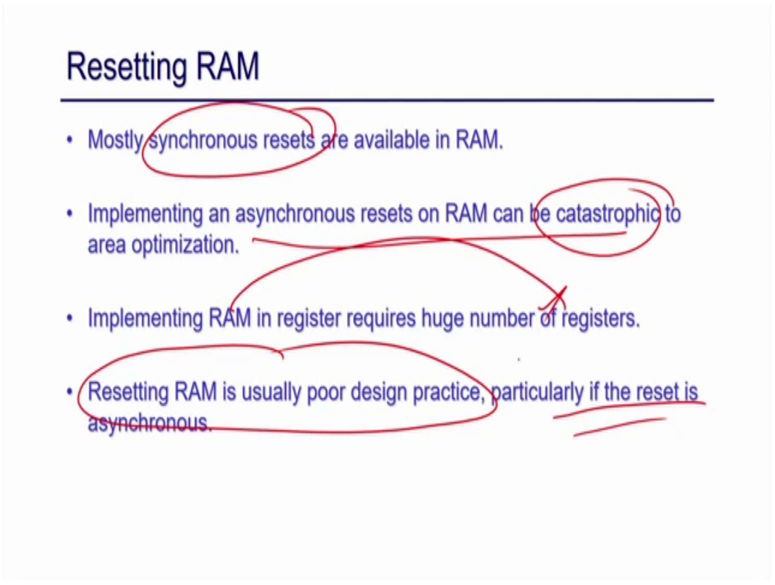
And if you just see here so and that will actually create problem right because this is your DSP. So, this is the multiplier will go to the DSP. So, DSP and then there are some logic right just to logic to async reset. So, this has to be there. So, that you can do this asynchronous and this will create extra area right that will need extra area you just I just the results are here.

So, if you just do the synchronous reset; that means, this is not there then I need only 1 DSP.

So, all these reset everything will be mapped to the same DSP, but if you have this asynchronous reset then I have to this extra logic has to be map into this LUT's I need 16 flip flops. Because now these are some set of register required I need 17 slice and 16 LUT's and I need the DSP just to do the multiplication this is the extra resource I need just to do this asynchronous reset. So, these are the kind of things that actually have an impact on any area.

And you should actually I mean I want to further if this may be not be absolutely necessary to make it asynchronous right. So, then probably you can remove this. So, you should aware of that particular kind of issues when you are actually mapping these things to FGPA ok.

(Refer Slide Time: 30:22)



Resetting RAM

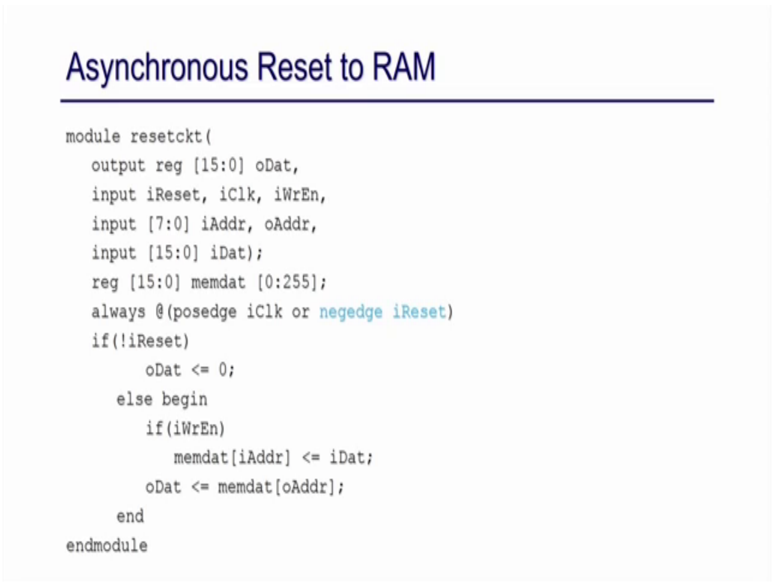
- Mostly synchronous resets are available in RAM.
- Implementing an asynchronous resets on RAM can be catastrophic to area optimization.
- Implementing RAM in register requires huge number of registers.
- Resetting RAM is usually poor design practice, particularly if the reset is asynchronous.

So, now we are going to talk about the RAM. So, we have discussed about shift register we have discuss about DSP's. Now we are going to discuss the pin configuration of RAM. So, RAM as synchronous reset does not have any asynchronous reset or asynchronous set ok.

So, if you have use any asynchronous reset for your design and it has a bad impact catastrophic impact on your optimization because then the whole because add a RAM is a b you unit right and you cannot map the whole thing. If you do not map that particular thing into RAM those will be map to registers and it is a disaster right because that will consume lot of a register of your design.

So, in general resetting a RAM is a poor design practices right you should not usually do not reset a RAM. So, that is actually have a very bad impact on in FGPA design specifically if your reset is asynchronous, because in that case that RAM will map to the huge number of registers ok.

(Refer Slide Time: 31:14)



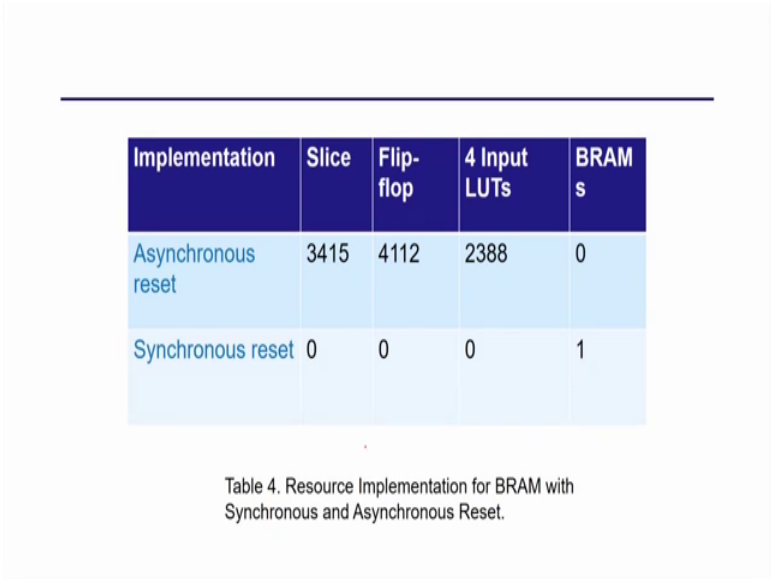
Asynchronous Reset to RAM

```
module resetckt(
   output reg [15:0] oDat,
   input iReset, iClk, iWrEn,
   input [7:0] iAddr, oAddr,
   input [15:0] iDat);
   reg [15:0] memdat [0:255];
   always @(posedge iClk or negedge iReset)
   if(!iReset)
         oDat <= 0;
      else begin
         if(iWrEn)
            memdat[iAddr] <= iDat;
         oDat <= memdat[oAddr];
      end
endmodule
```

So, you just take an example here. So, in this particular RAM I have asynchronous reset right. So, this is not and I am just be resetting there resetting that register in this RAM right.

Suppose in a synchronous reset then the problem is that the whole thing cannot be map this particular memory the being memory cannot be map to the RAM. And what will happen here this will be map to the normal registers and you can see the difference if is I have a synchronous.

(Refer Slide Time: 31:32)

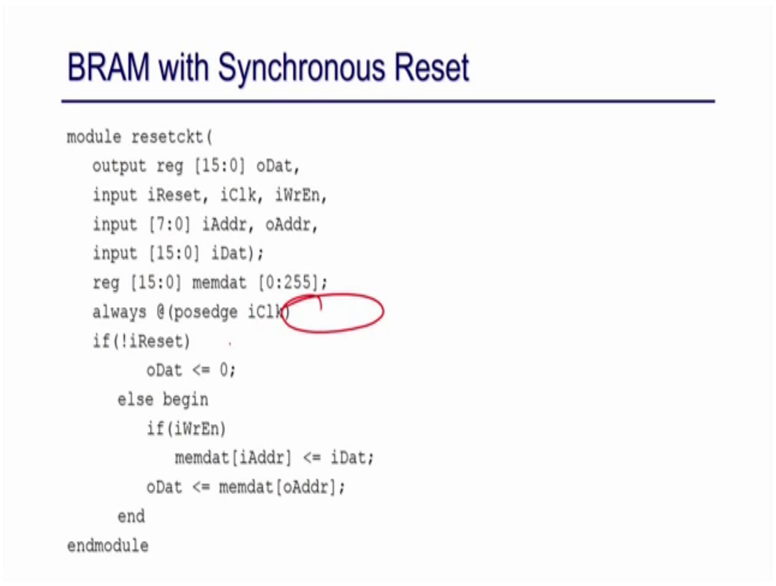


Implementation	Slice	Flip-flop	4 Input LUTs	BRAMs
Asynchronous reset	3415	4112	2388	0
Synchronous reset	0	0	0	1

Table 4. Resource Implementation for BRAM with Synchronous and Asynchronous Reset.

I have a synchronous I do not have this asynchronous reset then I can probably infer a single RAM for the whole design.

(Refer Slide Time: 31:38)

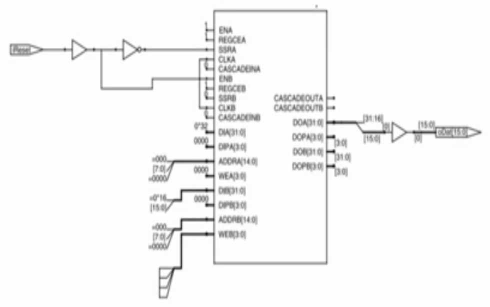


BRAM with Synchronous Reset

```
module resetckt(
   output reg [15:0] oDat,
   input iReset, iClk, iWrEn,
   input [7:0] iAddr, oAddr,
   input [15:0] iDat);
   reg [15:0] memdat [0:255];
   always @(posedge iClk)
   if(!iReset)
         oDat <= 0;
     else begin
         if(iWrEn)
            memdat[iAddr] <= iDat;
         oDat <= memdat[oAddr];
     end
endmodule
```

(Refer Slide Time: 31:42)

BRAM with Synchronous Reset

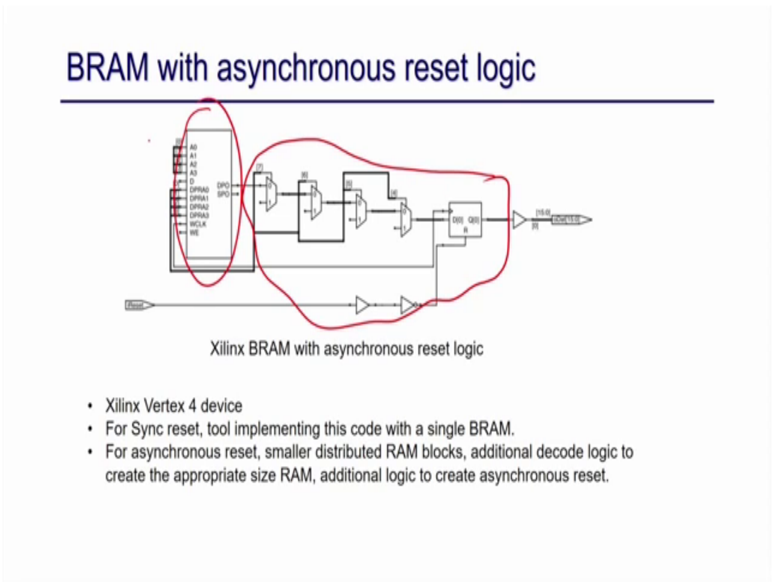


Xilinx BRAM with Synchronous Reset

Because this is nothing, but a memory just I am adding memory operation I just storing some memory reading memory and writing memory right. So, so I have just inferred a b RAM Whereas if it is asynchronous reset I cannot infer a b RAM and everything map into to LUT's right the memory and this slices and CLV's and you can see the kind of impact of this right I have just a RAM which is mapped there and it is taking thousands of slices ok.

So, this is something is very have a bad impact. So, when you are designing something for FPGA, you should avoid resetting a RAM ok. So, that will have a impact specifically if it is if it is specifically asynchronous reset ok.

(Refer Slide Time: 32:26)



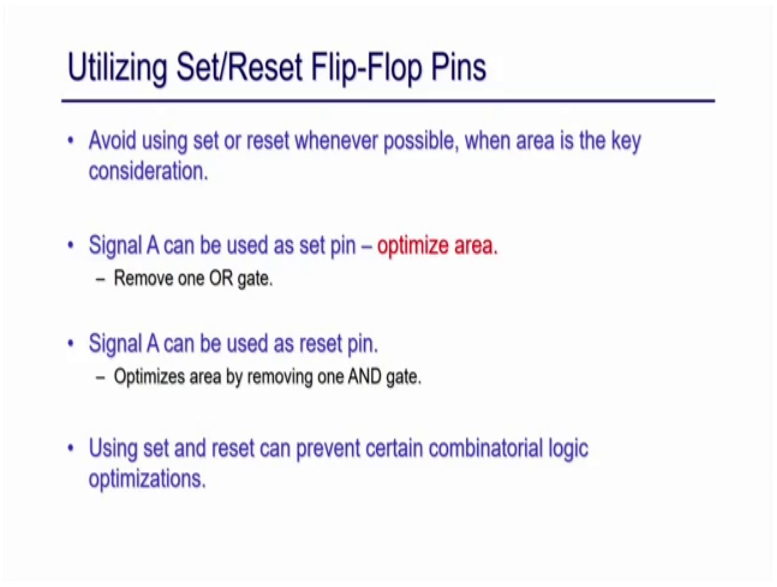
BRAM with asynchronous reset logic

Xilinx BRAM with asynchronous reset logic

- Xilinx Vertex 4 device
- For Sync reset, tool implementing this code with a single BRAM.
- For asynchronous reset, smaller distributed RAM blocks, additional decode logic to create the appropriate size RAM, additional logic to create asynchronous reset.

So, here is that extra logic is showing just us to do the asynchronous reset part. So, this is that small RAM and this is the asynchronous reset part which actually resetting that particular output ok. So, this is something give you a very fare idea how that asynchronous in reset impact in BRAM inference ok.

(Refer Slide Time: 32:50)



Utilizing Set/Reset Flip-Flop Pins

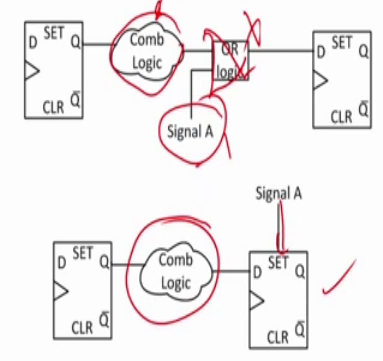
- Avoid using set or reset whenever possible, when area is the key consideration.
- Signal A can be used as set pin – optimize area.
  - Remove one OR gate.
- Signal A can be used as reset pin.
  - Optimizes area by removing one AND gate.
- Using set and reset can prevent certain combinatorial logic optimizations.

So, now I am going to talk about this the last component of this. So, even if the flip flops the registers have the set and reset pin and utilizing them actually have a area benefit right. So, we try to utilize you should you try to utilize those particular set and reset whenever possible because that actually give you give area benefits and if it give any improvements right.

(Refer Slide Time: 33:16)

Utilizing Set/Reset Flip-Flop Pins

Set pin can be used to reduce the combinational delay

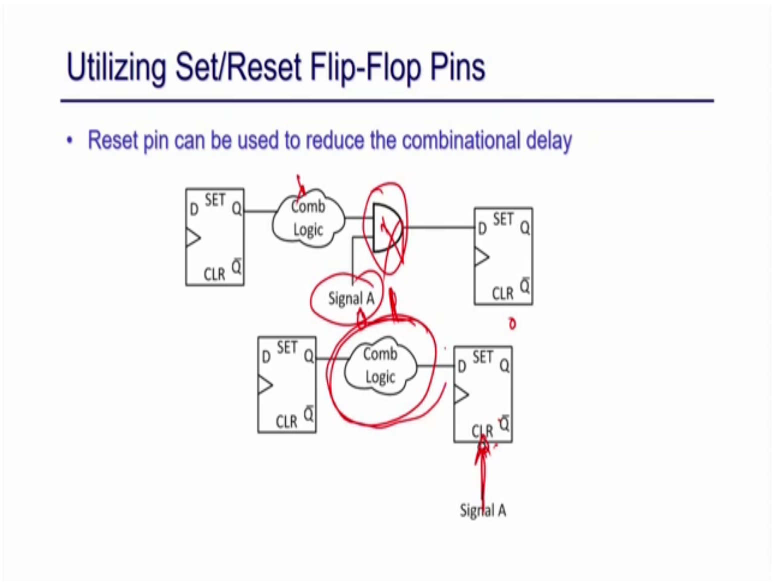


So for example in this particular design so I have this some combinational logic and then there is or gate here and then I have a signal so or with this right. So, if this signal is 1 and this signal is 1 right and it does not matter what is computing here.

So, what I can do? I can just make this as the set signal right. So, I just remove this or gate from my design and I just put this as a set because once this is this the register will definitely become 1 right does not matter what is coming here from other part. So, I can just set the register based on this signal A, so which actually reduce the 1 or gate what we just talked about here right. So, you just remove 1 or gate. So, you area will be refilled by 1 or gate and your combination delay is also improve by 1.

Similarly, if you are and get I can actually infer a reset pin out of it right.

(Refer Slide Time: 34:02)



Utilizing Set/Reset Flip-Flop Pins

- Reset pin can be used to reduce the combinational delay

So for example, here again the same example of what I have and get here. So, if this is 0 this will become 0, right the output will become 0 because this is a resetting that thing that does not matter what is this?

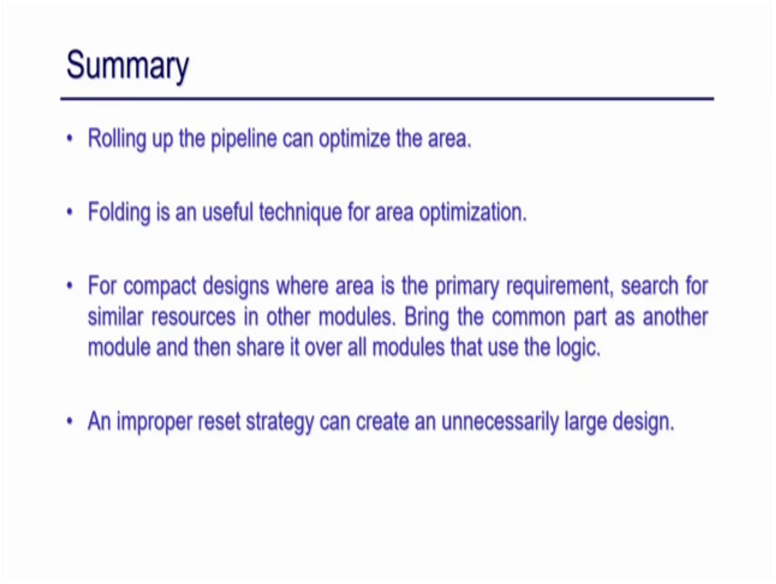
So, what I can do is as I remove this and gate I can put this signal as the reset pin to the reset pin negation of reset pin right. So, if it is if the input is 1 then it is definitely 0, otherwise it will just consider so if it is if it is 0, then it is become 1 and this is reset and if it is 0 sorry if this is 1 then this will become 0 it will not reset the pin. So, now it will the input depend on the combinational logic.

Again if we just try to utilize this you actually have you have a area benefit you, can actually improve your design you can remove 1 and gate from your design and again it will give you some timing benefit as well. So, overall the idea is that using that set and reset pin can prevent certain combination like optimization. So, we should always try to keep this option open so that I can actually and this actually primarily done by the synthesis tool right.

So, if you do not use that set and reset pin unnecessary from your design and if you have this kind of things the tool will automatically map those and or gate to the at the set and reset pin. So, you do not have to do anything, but you have to keep in mind that if we just unnecessary use your set and reset when this is this will not be done by the synthesis tool and then it will just unnecessary and 1 gate or gate in your design.

So, the idea is that when you are designing something you should always try to keep your set and reset pin un use so that this kind of combinational optimization can be done by the synthesis tool specifically for FGPA target ok. So, so we will just conclude with this.

(Refer Slide Time: 35:42)



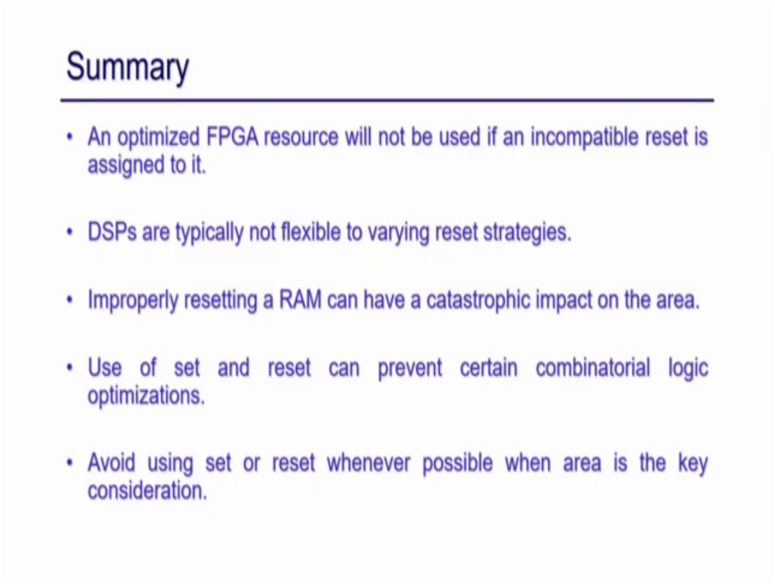
Summary

- Rolling up the pipeline can optimize the area.
- Folding is an useful technique for area optimization.
- For compact designs where area is the primary requirement, search for similar resources in other modules. Bring the common part as another module and then share it over all modules that use the logic.
- An improper reset strategy can create an unnecessarily large design.

So, what we are discuss today we have discussed several strategies that actually have a impact on area of your in your design we have discussed about this folding how folding can improve your area we have designed this pipelining rolling up pipelining. So, that you just make it alternative so that you can actually have a good area benefits.

We also design discuss about certain resource sharing of specifically for counter. How can I share a same counter for multiple designs even if they are not the same exactly same counter and also we have discuss several strategies that how to use that reset and set pin of this FGPA devices from a different component of FGPA devices so that your you can actually map better way in a FGPA right.

(Refer Slide Time: 36:35)



Summary

- An optimized FPGA resource will not be used if an incompatible reset is assigned to it.
- DSPs are typically not flexible to varying reset strategies.
- Improperly resetting a RAM can have a catastrophic impact on the area.
- Use of set and reset can prevent certain combinatorial logic optimizations.
- Avoid using set or reset whenever possible when area is the key consideration.

Specifically we have discussed about this shift register you should not use reset for shift register you should not use asynchronous reset or synchronous set for DSP's or as a whole you should not have use reset for RAM specifically you should not use asynchronous reset for a RAM. Because those are have a very catastrophic effect in area performance also we have discuss about that usually for flip flops you should keep the reset and set been opened.

So, that synthesis tool can actually map certain logic as a reset or set of this which actually have a better area benefits. So, in the next class we are going to discuss about this optimization technique for power how to improve power using some RTL optimization technique.

Thank you.