TSEA44: Computer hardware – a system on a chip

Lecture 6: Design for FPGAs

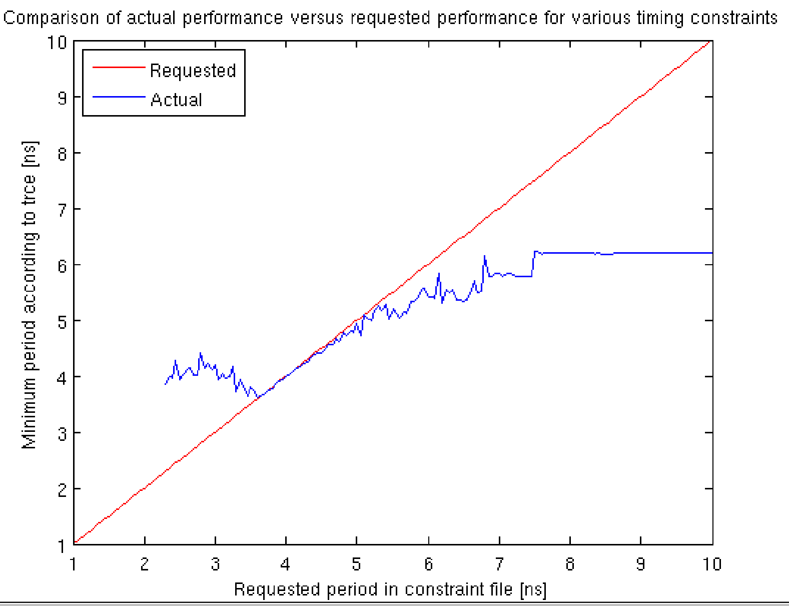
Material by Andreas Ehliar

LINKÖPING UNIVERSITY

Today

- Influence of goal hardware on architecture and code style
- Motivation
  - Clock speed
  - Area
  - Power
- Target FPGA architecture: Xilinx FPGA with 4-input LUTs
  - Same as VirtexII used in lab
  - Later generations use 6-input LUTs, but same ideas can be used

Clock cycle constraints effects on result



To get the best out of the FPGA

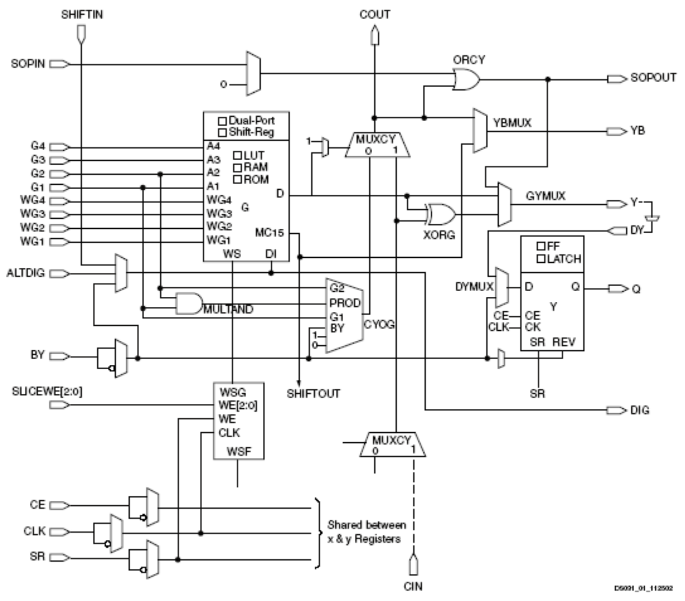
- Understand the architecture
- Use suitable descriptions
- Use available tools to extract implementation information
  - FPGA editor
  - Floorplanner
  - Planahead
  - Datasheets
  - Timing reports

FPGA components

- CLB:s
  - Slices
  - LUT
- Hard blocks
  - Block memory
  - Multipliers
  - I/O units

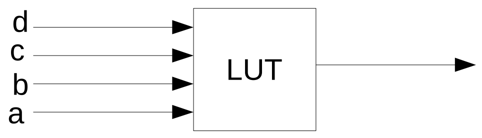
1/2 slice (total 8 of these in one CLB)

- Note
  - 4-input LUT G
  - XORG
  - CYOG
  - MUXCY
  - MULTAND



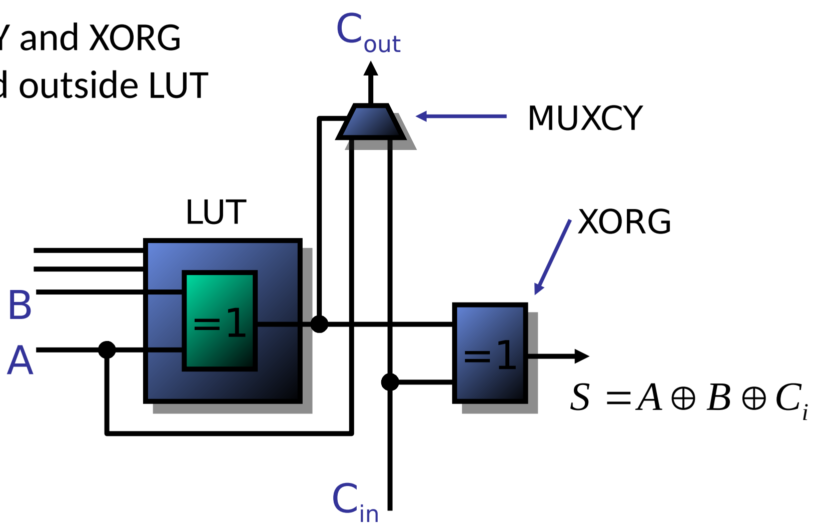
Combinatorial logic using a LUT

- 4 inputs give any logic function of at most 4 inputs



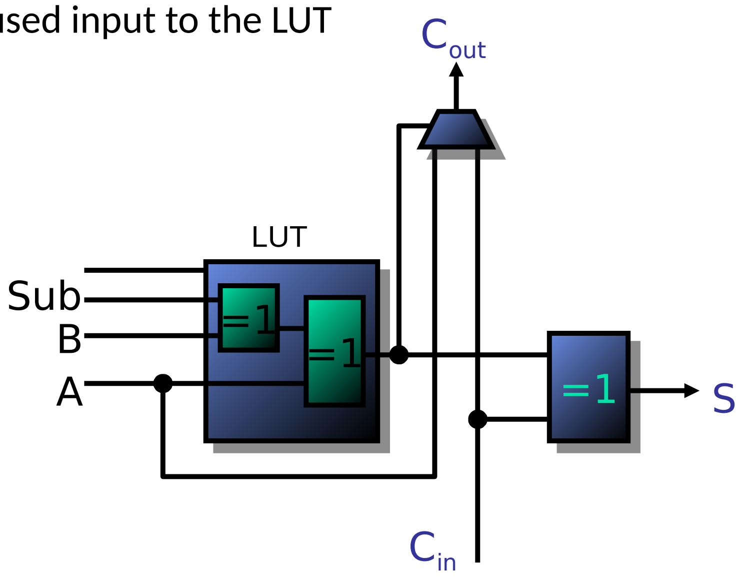
Adders and carrychains in Xilinx FPGAs

- 1 fulladder structure using carry chain acceleration
  - MUXCY and XORG located outside LUT
- 1 LUT/bit



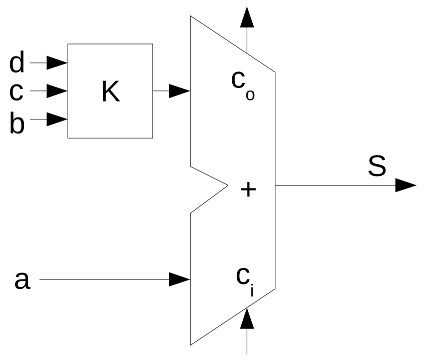
Extend to Add/subtract in Xilinx FPGAs

- Still one unused input to the LUT



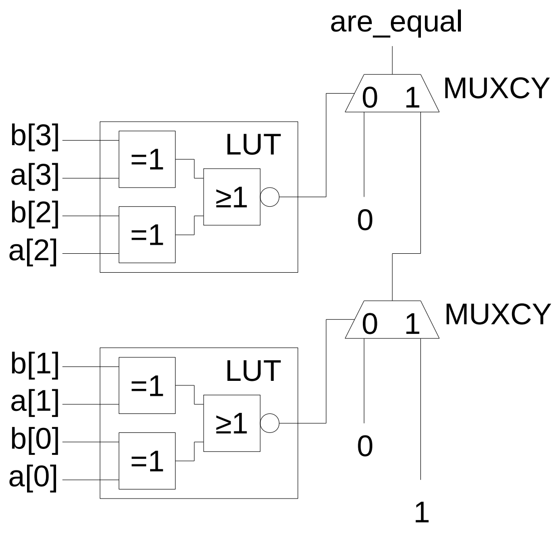
Rule of thumb for efficient adders in 4-input LUT based FPGAs

- S = a + K(b,c,d)
- Plain adder
- Adder/subtracter
- 2-to-1 mux and adder
- More strange versions
  - S = (opb | opc | opd) + opa
  - S = (opb & opc) + (opb & opa)
    (Uses MULT_AND located under LUT in slice figure)



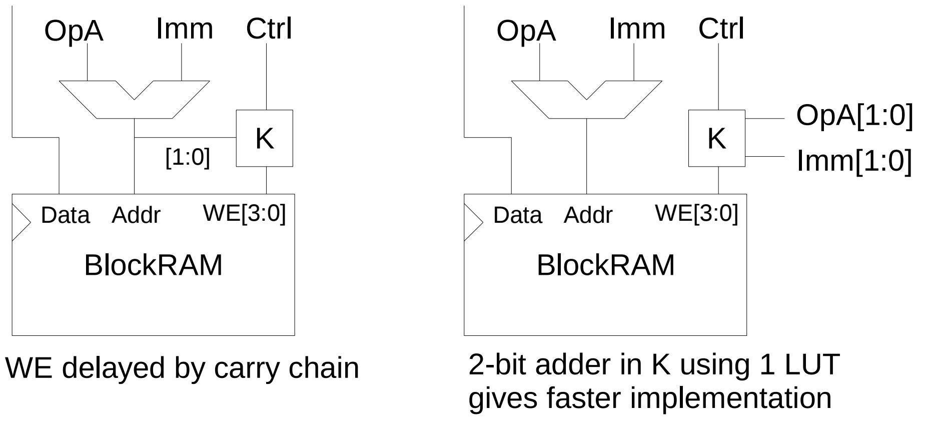
Carry chain for other purposes: Comparators

- Compare 2 bits per LUT
- Compare 4 bits per LUT if one value is constant!



Carry chain drawbacks

- Example: Address calculation selecting one byte memory



WE delayed by carry chain

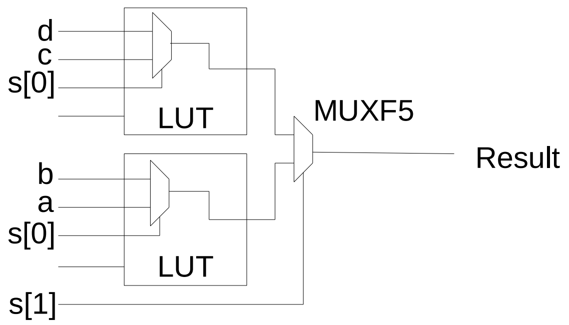
2-bit adder in K using 1 LUT gives faster implementation

- The carry chain itself is extremely fast
- Getting on the chain is not very fast

Multiplexers in FPGAs

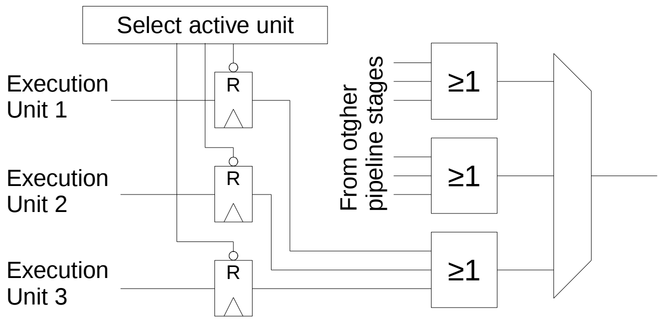
- A big difference between ASIC and FPGAs: Multiplexers are cheap in ASIC and expensive in FPGAs
- 4-input LUT: One 2-to-1 mux
- Specialized multiplexers in the slices are used to combine LUTs into larger multiplexers

Multiplexers in Xilinx FPGAs



- Possible use of spare input:
  - Invert output, set output to one or zero
  - Tricky variants based on a,b, and s[0]
- How many 4-input LUTs needed for a 4-to-1 mux (without MUXFx components)?

Avoiding multiplexers in pipelined designs

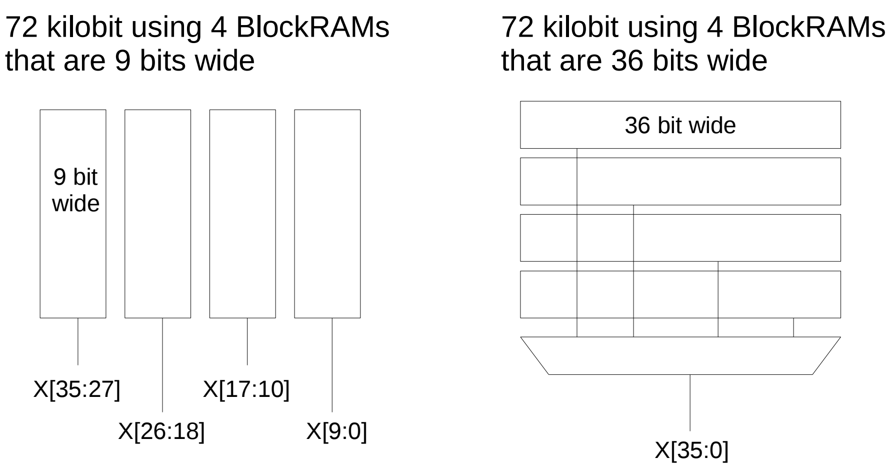


- Multiplexers are costly in FPGAs
- Alternative 1: Use or gates and make sure unused inputs are set to 0 using reset input of flip-flops
- Alternative 2: Use and gates and make sure unused inputs are set to 1. (see MULT_AND as well!)

Memory guidelines

- Standard rule: Large memories should be syncrhonous
- For high frequency design you want to register the output of the memory as well.
- For power reasons you should not enable the memory unless necessary
  - Double check that your enables work when inferring a memory!
- Smaller memories may be asynchronous if necessary
- You should not have a reset signal for your memory array
  - Easy to forget for shift registers!

Memories larger than one BlockRAM


72 kilobit using 4 BlockRAMs that are 9 bits wide
9 bit wide
X[35:27]
X[26:18]
X[17:10]
X[9:0]
72 kilobit using 4 BlockRAMs that are 36 bits wide
36 bit wide
X[35:0]

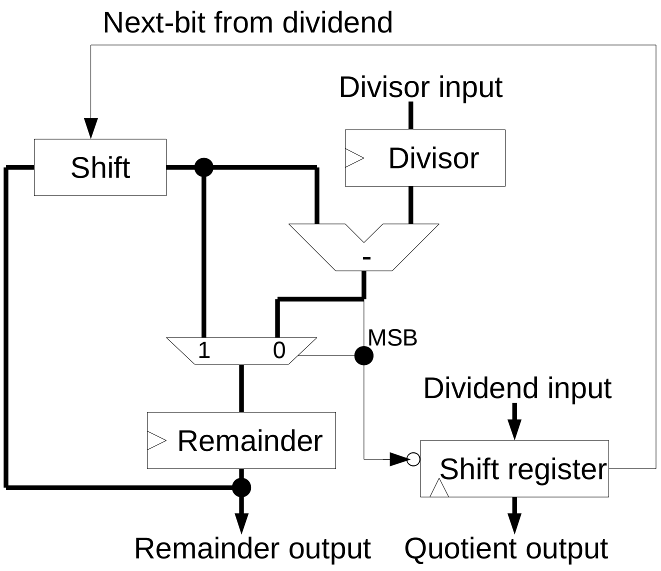
- Why use the right variant? Reduced power consumption!

A case study: A divider for a RISC processor

- Used in a 32-bit RISC processor
- Target frequency: 320 MHz in a Virtex-4 (speedgrade -12)
- Uses restoring division algorithm (basic operations are shift, subtract, and select)
  - Serial computation
  - Very similar to manual division

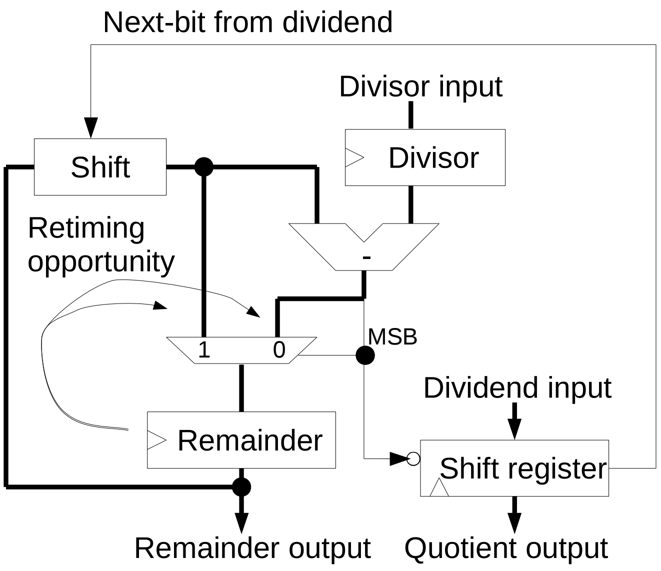
$$\frac{dividend}{divisor} = quotient \times divisor + remainder$$

Initial divider architecture



Initial divider architecture

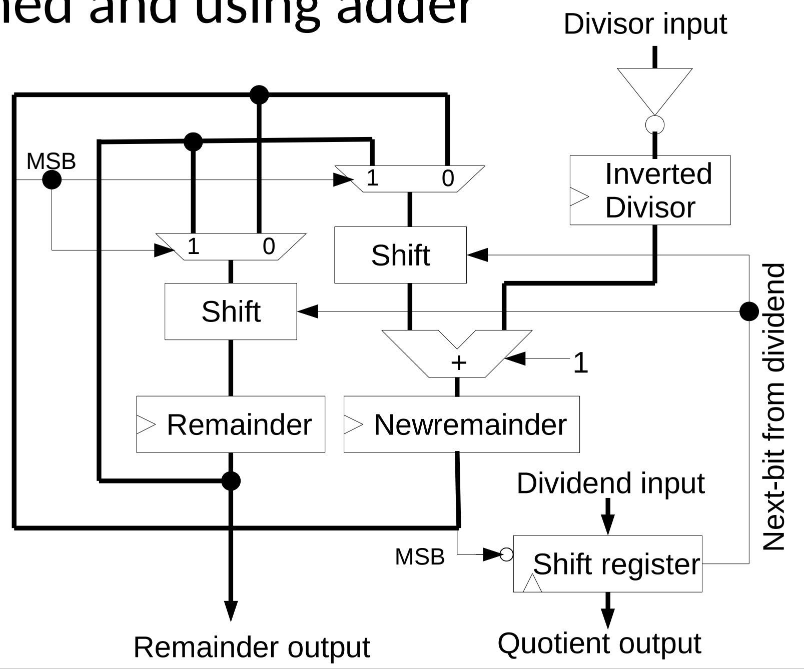
Initial speed:
300 MHz



Issues

- Cannot combine subtracter and 2-to-1 multiplexer!
- Solution: Preprocess divisor and use an addition instead

Retimed and using adder



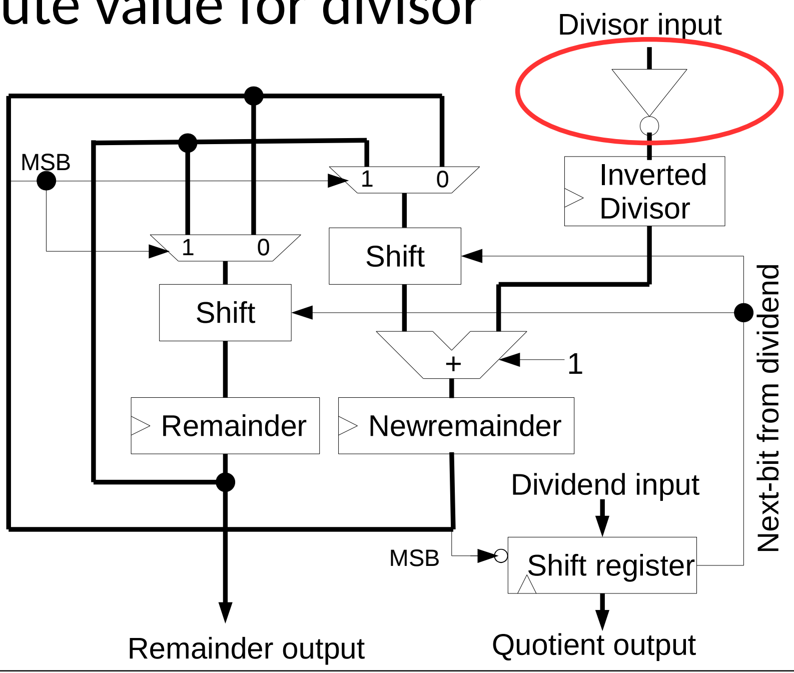
Other issues

- Synthesis tool was too clever
- Manually instantiating the components worked
- Alternatively a complete rewrite of the module worked as well
- Improves clock frequency to 377 MHz (from 300 MHz)

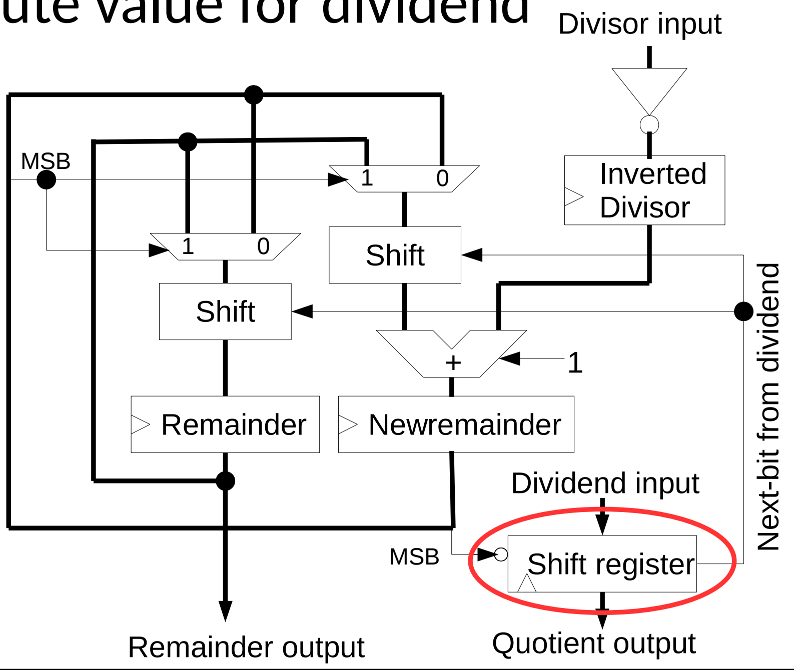
Dealing with negative numbers

- Idea: Take absolute value of dividend and divisor
- Negate quotient and remainder if necessary
- For a 32 bit divider this seems to require around 128 extra LUTs…

Absolute value for divisor

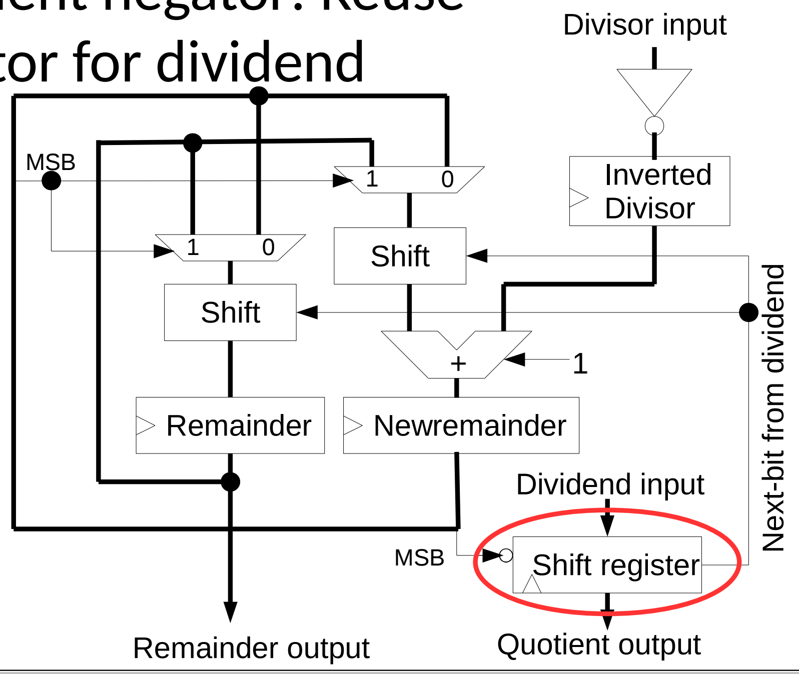


Absolute value for dividend

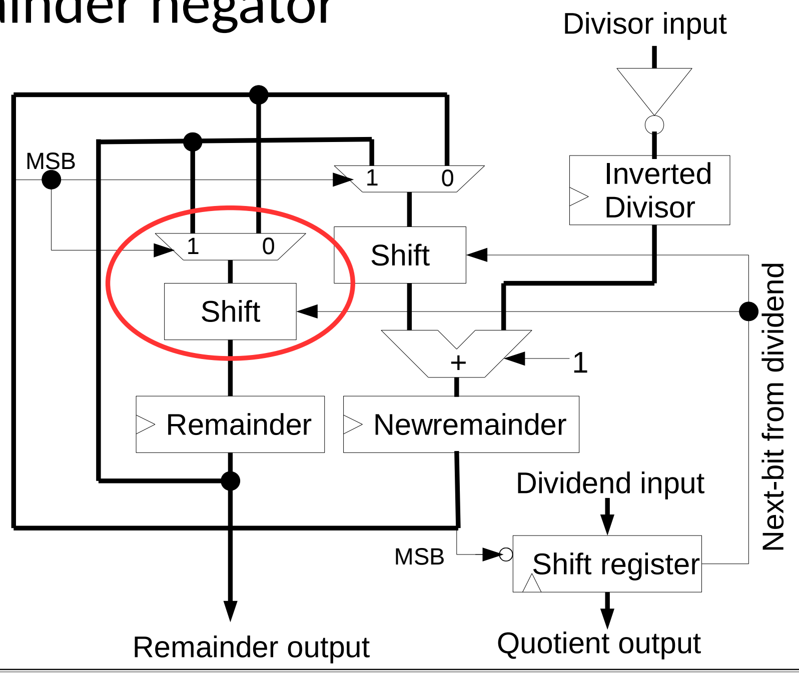


2016-11-22 00:24

Quotient negator: Reuse negator for dividend



Remainder negator



Tricky to do in practice

- Required signals for shift register:
    1. Load enable/shift enable
    2. Invert enable
    3. Input data of new dividend
    4. Input data of new dividend (MSB bit)
    5. Current value of register
- 5 inputs to a 4 input LUT?

Tricky to do in practice - Solution

- Solution: Skip MSB of dividend input for ABS operation
- Always invert the dividend, only add 1 as a carry in if appropriate
    - This can be implemented by adding a few extra LSB bits
    - If we had a positive value we can compensate for the inversion at shift out
    - We can even add a control bit to select between signed/unsigned division
- Manual instantiation was necessary to actually implement this

Results for Virtex-4, speedgrade 12

- Unoptimized, unsigned: 300 MHz, 107 LUTs
- Retimed, unsigned: 377 MHz, 140 LUTs
- Retimed, signed: 361 MHz, 151 LUTs
- Retimed, signed or unsigned: 363 MHz, 153 LUTs

Manual instantiation

- Last resort when synthesis attributes and rewriting the RTL code does not work
- Not portable between FPGA vendors
  - Suprisingly portable to ASIC however

Manual instantiation of flip-flops

- Allows you to ensure that the correct signals are corrected to the D, CE, and SR inputs
  - XST (Xilinx own synthesis tool, not used in the lab) often seem to select the wrong input for SR
  - Background: SR input is quite slow compared to D input
- Can sometimes be avoided by rewriting the code or using synthesis attributes
- Often easier to just instantiate flip-flop primitives directly

Manual instantiation of Memories and DSP Blocks

- Well documented in various application notes

Synthesis attributes

- A convenient way to force the synthesis tool to do what you mean
- In VHDL:

  attribute keep : string;
  attribute keep of mysignal: signal is "TRUE"
- In Verilog:

  (* KEEP = "TRUE" *) wire mysignal;
- Note: Synthesis attributes discussed here are for XST, not Precision!
  - (Read the Precision manual)

Synthesis attribute KEEP

- Preserves the selected signal
- Use case:
  - The synthesis tool makes a bad optimization decision.
  - By using KEEP you can ensure that a certain signal is not hidden inside a LUT and hence guide the optimization process

KEEP example from a display controller

```
wire inimagey = (yctr > 31) && (yctr < 192);
wire inimagex = (xctr > 15) && (xctr < 26);
…
always @(posedge clk) begin
  if (inimagey && (xctr == 15) ) begin
  …
  end else if(inimagey && (xctr == 26)) begin
  …
  if (inimagey && (xctr == 15) ) begin
  …
  end else if(inimagey && (yctr[2:0] == 7)) begin
  …
```

- Problem: Synthesis tool merged inimagey test with other tests in suboptimal way

Solution: Force inimagey and inimagex to be separate signals

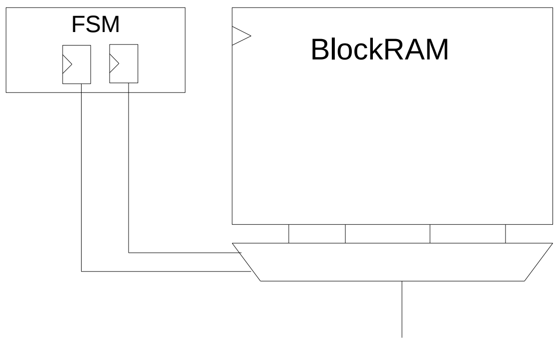
```
(* KEEP = "TRUE" *) wire inimagey;
(* KEEP = "TRUE" *) wire inimagex;

assign inimagey = (yctr > 31) && (yctr < 192);
assign inimagex = (xctr > 15) && (xctr < 26);
```

- Saved area in an area constrained situation
- Especially important when targetting both CPLD and FPGAs with a single IP core

SIGNAL ENCODING attribute

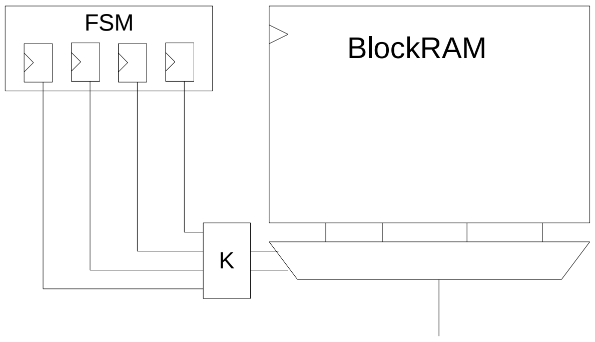
- Allows you to select encoding for state machines
- Useful when synthesis tool make suboptimal state machine encoding choices
- (Alternatively: You can disable FSM optimization if you **really** want to)

Example: Memory byte select in a processor



- Signal encoding specified 2 FF, 4 states.
- Two signals into mux control signal

Example: Memory byte select in a processor

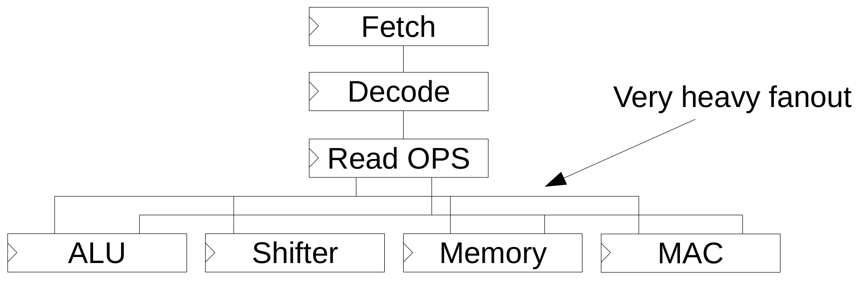


- Heuristics in the synthesis tool selected one-hot coding for the FSM…

EQUIVALENT REGISTER REMOVAL attribute

- Allows you to specify that certain registers should not be optimized away.
- Perfect when you do not want the synthesis tool to touch your carefully optimized (duplicated) flip-flops

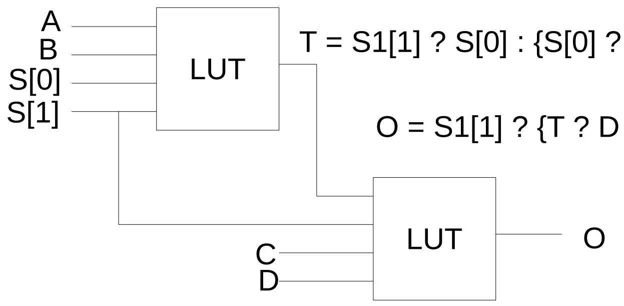
Example: Operand bus in a processor



- Problem: Manual register duplication in read operand stage is removed by synthesis tool
- Solution: Disable optimization locally by setting EQUIVALENT_REGISTER_REMOVAL to "no"

4-to-1 multiplexer using two LUT4

4-to-1 multiplexer using two LUT4



T = S1[1] ? S[0] : {S[0] ? B : A};

O = S1[1] ? {T ? D : C} : T;

Conclusions

- By mapping your design to the FPGA in an efficient manner you can significantly improve the performance of your design
- Keep this in mind early in the design phase.
- (However, don't optimize unless you really need to.)

www.liu.se

LINKÖPING
UNIVERSITY