Example 2: 32x32 Multiplier

(and why you want to use a DSP instead)

Digital Design Review: 32x32 Multiplier

- Longhand multiplication

```
   1110
 x 1001
 ------
   1110
  0000
 0000
1110
-------
1111110
```

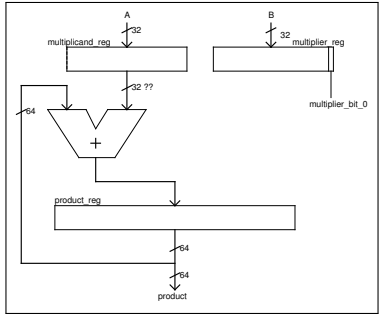
- Check first (LSB) bit of multiplier, if multiplier is a one, save a copy of multiplicand, else save all zeros.
- Check next bit of multiplier, if its a one, left-shift the multiplicand once and add it to the previously saved value. Else, add only zeros.
- Once all bits of the multiplicand are checked, and additions performed, the sum is the product.

Digital Design Review: 32x32 Multiplier

- Design constraints:
  - Synchronous (not combinatorial) using edge-triggered flip-flops
  - One clock, rising edge only, never gated
  - One reset, asynchronous, applied at power-up
  - One input to tell us to start, one output to indicate completion
- Now, ask, “what structures and paths do I need to implement the multiplication?”
  - Need storage for operands and product plus an adder
  - These will be flip-flops, not RAM. RAM makes no sense and is very expensive.
  - Then, what data paths will be needed between the blocks?

Digital Design Review: 32x32 Multiplier

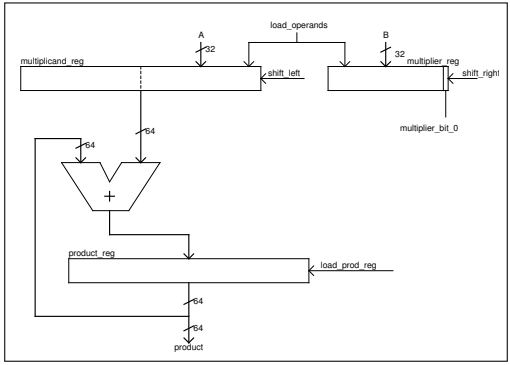
- First cut for storage and data paths
  - 32x32 multiplier must have a 64-bit output
  - Only looking at the LSB of the multiplier register
  - One clock, rising edge only, never gated
  - Clock and reset not shown. They are understood signals.



- What's missing, broken? What do we need now?

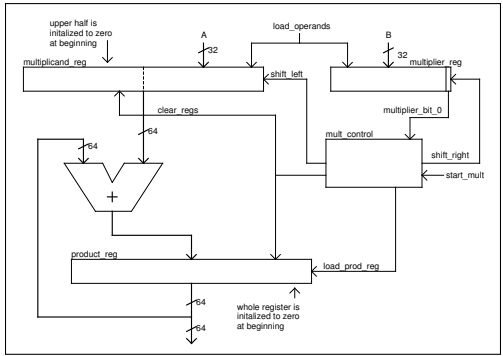
Digital Design Review: 32x32 Multiplier

- Clean up initial design - fix data paths, add some control stubs
  - Need 64-bit multiplicand register to shift left up-to 32 times
  - Need to be able to load the operand registers
  - Need to be able to load or hold value in product (free running clock)



Digital Design Review: 32x32 Multiplier

- Add remaining control signals
  - Upper 32 bits of multiplicand, product register needs to be cleared
  - Add "start" and "done" signals



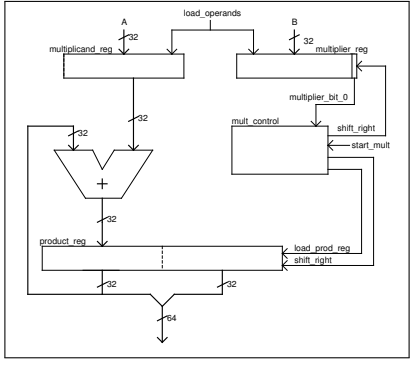
- High-level block diagram is done! Time for a *noggin simulation*.
- Control is abstracted away into its own block.

Digital Design Review: 32x32 Multiplier

- By now, you should see some optimizations that can be made.
- Approaching the design as we have exposes optimizations clearly.
- A "code first" approach will not reveal these optimizations.
- "Optimizing" your HDL code **will never create these optimizations**, as they are based in the structure of the hardware.
- "Optimizing" HDL code very often gives you exactly the same structure, only with more difficult to read code.

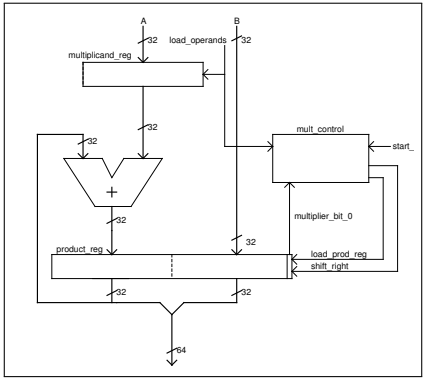
Digital Design Review: 32x32 Multiplier

- First Optimization
  - The multiplicand register is filled with zeros as its shifted left. Half of it only holds zeros that the 64-bit ALU had to process.
  - Consider: don't shift the multiplicand but shift the product right.
  - Result: 32-bit multiplicand register and ALU. No synchronous reset required for the product or multiplicand register.



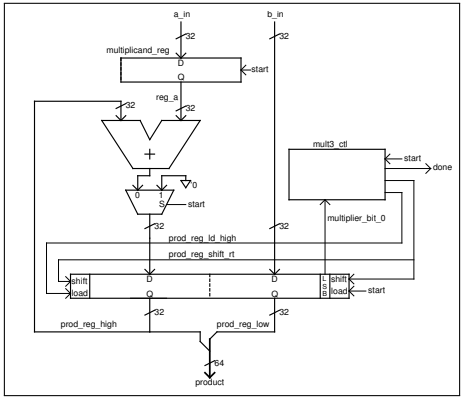
Digital Design Review: 32x32 Multiplier

- Final Optimization
  - Any values in the lower half of the product register are discarded as it is shifted right. The wasted space is exactly the size of the multiplier.
  - As wasted space in the product disappears, so does the multiplier.
  - Lower half of the product register temporarily stores multiplier.
  - Result: Removal of the 32-bit multiplier register and simpler control.



Digital Design Review: 32x32 Multiplier

- With an acceptable architecture we can create the control signals and tidy things up a bit. Fortunately, we have a simpler datapath to control.



Digital Design Review: 32x32 Multiplier

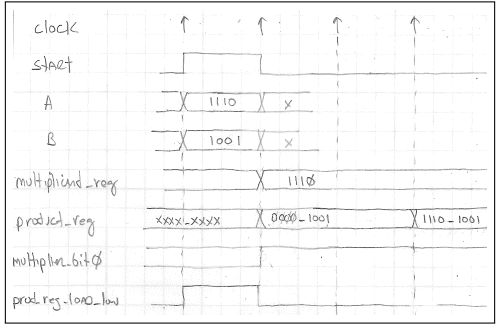
- Ground-rules for our control design
  - All control signals are synchronous to one clock (rising edge)
  - All control signals are created in synchronous state machines.
  - All control signals change state at rising clock edge.
  - All data movement happens at the rising clock edges.
  - Clock is never gated, we use enabled flip-flops.
- Use timing diagram and state machine diagrams to develop
- Start with either, go back and forth making sure they agree and that the signals are created at the correct time.

Digital Design Review: 32x32 Multiplier

- Why timing and state machine diagrams?
- Algorithms describe **what** things happen in a sequential manner. *Parallelism is hidden.*
- Timing Diagrams describe **when** things happen in a parallel manner. *Parallelism is exposed.*
- State Machine Diagrams separate parallel operations into easier to understand mini-machines.
- The visual nature of timing and state machines allow the use of visual pattern matching to observe commonality, see mistakes and to simplify our designs.

Digital Design Review: 32x32 Multiplier

- Timing and state machine diagrams are best worked out with paper and pencil or whiteboards.
- Producing these diagrams is another *eraser-intensive* exercise. Its hard work, and worth every minute.
- Time spent here is equal to 5x-10x time debugging HDL code.



- Some of this could become part of a future homework assignment

Questions?