AMD to acquire Xilinx in all-stock deal valued at $35 billion

Published: Oct. 27, 2020 at 6:47 a.m. ET

By Ciara Linnane

Advanced Micro Devices Inc. AMD, -3.77% said Tuesday it has reached an agreement to acquire Xilinx Inc. in an all-stock deal valued at $35 billion. Xilinx shareholders will receive 1.7234 share of AMD for each share owned, equal to $143 in cash. AMD shareholders will own about 74% of the net entity, while Xilinx shareholders own the remaining 26%. The deal is expected to close before end 2021. "The combination will create the industry's leading high performance computing company, significantly expanding the breadth of AMD's product portfolio and customer set across diverse growth markets where Xilinx is an established leader," the companies said in a statement. "The transaction is expected to be immediately accretive to AMD margins, EPS and free cash flow generation and deliver industry-leading growth." Xilinx shares jumped 11% premarket on the news, while AMD fell 3.7%.
State Elements

- Our FPGA has “state elements” in multiple places
  - Shifter Register for configuration loading
  - configuration state (PIPs, LUTs, options, ...)
  - SRAM blocks
  - CLB flip-flops
  - MAC pipelining
- Standard Cell Library will have flip-flops and latches
  - We might want custom/optimized versions
  - Or perhaps use standard cells, but custom “tiling”
- SRAM compiler for RAM blocks (with special cells)
Latches

‣ Usually defined as “level-sensitive” (as opposed to edge-triggered)

‣ Sometime called “transparent” latch

‣ Uses in:
  ‣ Large memory blocks - SRAM
  ‣ Flip-flops
  ‣ shift registers
  ‣ standalone registers, etc.
Historical Perspective for Latches

- 80's into the 90's
- “dynamic” circuits: fast, small, low-power, unreliable!

Figure 1  Dynamic Shift Register

Fig. 8a. Stick Diagram of One Row of Shift Register Array

Fig. 8b. Hand Sketch of Layout of One Shift Register Cell
Dynamic CMOS shifter

- Now requires 2-phase clock plus complements
- Eventually, “true single phase” clocking schemes emerged.
- However, dynamic state elements largely abandoned (not counting DRAM memory)
- Why?
CMOS Static Latches

- All circuits that implement static latches in CMOS are based on “cross-coupled inverters”
- While powered on, two-stable states, one meta-stable state.
- Do we worry about meta-stability?

Explain

 roboustness

V_{out1} = V_{in2}

\begin{align*}
V_{out1} &= T(V_{in1}) \\
V_{out2} &= T(V_{in2})
\end{align*}
Making Practical Latches

- **Reading** is easy in the case of isolated latches (more complex in SRAM arrays):

  - Output capacitive load can effect the writing time.

- Decoupling with output buffer helps in timing analysis (common in standard cell designs)
Writing a Latch 1

- Overpower the state:
Writing a Latch 1

- Overpower the state (differential version):

![Electrical circuit diagram showing a latch with enable signals.](image)
Writing a Latch 2

- **Break the loop (with switches)**

- Requires en and en’

- Again, standard cell versions:
  - will add output inverter for decoupling. Makes write delay independent of load.
  - Add input inverter to make write delay independent of previous stage drive.
Addressing the write problem

Implemented with a MUX.

Implemented with a tristate buffers.
Writing a Latch 3

- Break the loop (with logic)
- cross-coupled logic gates
Cross-coupled NOR gates

- If both R=0 & S=0, then cross-coupled NORs equivalent to a stable latch:

- If either R or S becomes =1 then state may change:

- What happens if R or S or both become = 1?
Asynchronous State Transition Diagram

Transitions triggered by input changes.

SR Latch:

<table>
<thead>
<tr>
<th>SR</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>hold</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>indeterminate</td>
</tr>
</tbody>
</table>

- S is “set” input
- R is “reset” input

QQ’=00 is often called a “forbidden state”
Nand-gate based SR latch

- Same behavior as cross-coupled NORs with inverted inputs.

1. Logic diagram
2. Function table

**Fig. 5-4** SR Latch with NAND Gates
Level-sensitive SR Latch

The input “C” works as an “enable” signal, latch only changes output when C is high.

Input NANDs invert S/R

<table>
<thead>
<tr>
<th>C</th>
<th>S</th>
<th>R</th>
<th>Next state of Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$Q = 0$; Reset state</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$Q = 1$; set state</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Indeterminate</td>
</tr>
</tbody>
</table>

(a) Logic diagram

(b) Function table

Fig. 5-5 SR Latch with Control Input
D-latch

(a) Logic diagram

(b) Function table

<table>
<thead>
<tr>
<th>$C$</th>
<th>$D$</th>
<th>Next state of $Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$X$</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$Q = 0$; Reset state</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$Q = 1$; Set state</td>
</tr>
</tbody>
</table>

Compare to transistor version:
Flip-flops

Note: The terms “master” and “slave”, no longer acceptable in this new era of Diversity, Equity, and Inclusion.

Fig. 5-9 Master-Slave D Flip-Flop

Fig. 5-10 D-Type Positive-Edge-Triggered Flip-Flop
J-K FF

- Add logic to eliminate "indeterminate" action of RS FF.
- New action is "toggle"
- J = "jam"
- K = "kill"

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q(t)</th>
<th>Q(t+Δ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 0</td>
<td>0 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- J = "jam"
- K = "kill"

- New action is "toggle"
## Storage Element Taxonomy

<table>
<thead>
<tr>
<th></th>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>level-sensitive</td>
<td>edge-triggered</td>
</tr>
<tr>
<td>D-type</td>
<td>⭐</td>
<td>✓</td>
</tr>
<tr>
<td>JK-type</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RS-type</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- "latch" form: ⭐ "natural" form
- "flip-flop" form: ✓ "possible" form
- n.a.
Design Example with RS FF

- With D-type FF state elements, new state is computed based on inputs & present state bits - reloaded each cycle.
- With RS (or JK) FF state elements, inputs are used to determine conditions under which to set or reset state bits.
- Example: bit-serial adder (LSB first)
Bit-serial adder with RS FF

- RS FF stores the carry:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c_i</th>
<th>c_{i+1}</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Carry kill $a'b'$
- Carry generate $ab$
Writing a Latch 4

- Power down the inverters (to avoid the fight)
- High differential gain
  - fast (reduced voltage swings on input)
  - robust (common mode noise rejection)
- Often used in sensing circuits

Positive Level-sensitive:
transparent high, latching low

Negative Level-sensitive:
transparent low, latching high
Sky130 D-Latch
Sky130 D-Latch
Sky130 Flip-flop
Sky130 Flip-flop
Homework Assignment: What should we use for shifters and latches?

We want **small**, fast, reliable.

Minimizing the number of transistors *usually* minimizes the area.

Latches uses in:
- shifters
- PIPs, LUT function
- flip-flops (CLBs) - need reset/set

We can choose full custom or to introduce new cells into stdcell library.
Multiplexors

- The next most important circuit in FPGAs
- LUT implementation, options in CLBs, connection boxes
- Often 2-to-1 is sufficient or building block for larger multiplexors

Home Assignment: extract and draw circuit diagram for Sky130 2-to-1 multiplexor:

Discuss alternatives for FPGAs

2-to-1 multiplexor:
\[ C = s'a + s'b \]
6-Transistor SRAM (Static RAM)

- Large on-chip memories built from arrays of static RAM bitcells, where each bit cell holds a bistable (cross-coupled inverters) and two access transistors.
- Other clocking and access logic factored out into periphery
SRAM Block Example

256×32 (or 8192 bit) SRAM Generated by hard-macro module generator

- Generate highly regular structures (entire memories, multipliers, etc.) with a few lines of code
- Verilog models for memories automatically generated based on size
6T-SRAM — Layout

$V_{DD}$ and GND: in M1
Bitlines: M2
Wordline: poly-silicon
65nm SRAM

- ST/Philips/Motorola

Access Transistor

Pull down

Pull up
6T SRAM Cell Layouts

Fig. 2. Layout of Type 1a (A), Type 1b (B), Type 2 (C), Type 3 (D), Type 4 (E) and Type 5 (F) SRAM cells.

Design and evaluation of 6T SRAM layout designs at modern nanoscale CMOS processes

Dimitrios Balobas
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
dimostep@gmail.com

Nikos Konofaos
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
nikoskonofaos@eth.gr
General SRAM Structure

Address Decode and Wordline Driver

Bitline Prechargers

Differential Read Sense Amplifiers

Differential Write Drivers

Usually maximum of 128-256 bits per row or column

Address

Write Enable

Clk

Clk

Write Data

Read Data
Address Decoder Structure

One-hot 1-of-4 encoding

2:4 Predecoders

Address:
- A0
- A1
- A2
- A3

Word Lines:
- Word Line 0
- Word Line 1
- ... (15 dots)
- Word Line 15

Clocked Word Line Enable
Read Cycle

1) Precharge bitlines and senseamp

2) Pulse wordlines, develop bitline differential voltage

3) Disconnect bitlines from senseamp, activate sense pulldown, develop full-rail data signals

Pulses generated by internal self-timed signals, often using “replica” circuits representing critical paths
**Write Cycle**

1) **Precharge bitlines**
2) **Pull down one bitline full rail, open wordline**

Write-enable can be controlled on a per-bit level. If bit lines not driven during write, cell retains value (looks like a read to the cell).
SRAM Operation - Read

1. Bit lines are “pre-charged” to VDD
2. Word line is driven high (pre-charger is turned off)
3. Cell pulls-down one bit line
4. Differential sensing circuit on periphery is activated to capture value on bit lines.

During read $\overline{Q}$ will get pulled up when WL first goes high, but …

- Reading the cell should not destroy the stored value
CMOS SRAM Analysis (Read)

\[ CR = \frac{W_1/L_1}{W_5/L_5} \]
SRAM Operation - Write

1. Column driver circuit on periphery differentially drives the bit lines
2. Word line is driven high (column driver stays on)
3. One side of cell is driven low, flips the other side

For successful write the access transistor needs to overpower the cell pullup
CMOS SRAM Analysis (Write)

Size width ratio between PMOS pull-up and NMOS access

\[ \frac{W_4}{L_4} : \frac{W_6}{L_6} \]
1) Each row of the array will include more than one logical word.
2) Difficult to pitch match sense amp to tight SRAM bit cell spacing so often 2-8 columns share one sense amp. *Impacts power dissipation as multiple bitline pairs swing for each bit read.*
## Building Larger Memories

- Large arrays constructed by tiling multiple leaf arrays, sharing decoders and I/O circuitry
  - e.g., sense amp attached to arrays above and below
- Leaf array limited in size to 128-256 bits in row/column due to RC delay of wordlines and bitlines
- Also to reduce power by only activating selected sub-bank
- In larger memories, delay and energy dominated by I/O wiring

<table>
<thead>
<tr>
<th>Bit cells</th>
<th>Dec</th>
<th>Bit cells</th>
<th>Dec</th>
<th>Bit cells</th>
<th>Dec</th>
<th>Bit cells</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td>I/O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit cells</td>
<td>Dec</td>
<td>Bit cells</td>
<td>Dec</td>
<td>Bit cells</td>
<td>Dec</td>
<td>Bit cells</td>
<td>Dec</td>
</tr>
<tr>
<td></td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td>I/O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit cells</td>
<td>Dec</td>
<td>Bit cells</td>
<td>Dec</td>
<td>Bit cells</td>
<td>Dec</td>
<td>Bit cells</td>
<td>Dec</td>
</tr>
<tr>
<td></td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td>I/O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit cells</td>
<td>Dec</td>
<td>Bit cells</td>
<td>Dec</td>
<td>Bit cells</td>
<td>Dec</td>
<td>Bit cells</td>
<td>Dec</td>
</tr>
</tbody>
</table>
Adding More Ports

Differential Read or Write ports

Optional Single-ended Read port
Memory Compilers

- In ASIC flow, memory compilers used to generate layout for SRAM blocks in design
  - Often hundreds of memory instances in a modern SoC
  - Memory generators can also produce built-in self-test (BIST) logic, to speed manufacturing testing, and redundant rows/columns to improve yield

- Compiler can be parameterized by number of words, number of bits per word, desired aspect ratio, number of sub banks, degree of column muxing, etc.
  - Area, delay, and energy consumption complex function of design parameters and generation algorithm
  - Worth experimenting with design space

- Usually only single read or write port SRAM and one read and one write SRAM generators in ASIC library
Small Memories

- Compiled SRAM arrays usually have a high overhead due to peripheral circuits, BIST, redundancy.
- Small memories are usually built from latches and/or flip-flops in a stdcell flow
- Cross-over point is usually around 1K bits of storage
  - Should try design both ways
End of Lecture 12