Clock handling

The clock is a key participant in any CPU, since it controls the whole timing of the chip and ultimately the speed of the processor.

The clock signal enters the Z80 through a pad in the lower right. Like the other input pins, it has a protection diode. From there it flows through the entire chip, unbuffered, unamplified, and unmodified. This is a bit of a surprise, since many chips process the clock or amplify it on the chip.

The clock first goes around the chip clockwise, clocking the address pin latches. Then it jumps the thick ground trace and splits up. One trace goes back above the register file, where it provides the clock to the register file. This trace then goes above the ALU, where it provides clock to the ALU. Other traces continue clockwise, providing the clock signal to the timing and control logic. In the upper right, the clock trace doubles back providing another clock line through the timing logic. Finally, a trace goes down along the right, past the instruction latch (which doesn't use it), to be used in a couple places in the decode logic.

Overall, the clock path is longer than I'd expect, with several paths that double back instead of having a more tree-like structure. This may have been done for timing reasons. The doubled-back path will have more propagation delay, so if the signals coming into that path are also slightly delayed, this could help the overall timing of the chip. Alternatively, the clock could have been positioned just for convenience in layout. It's clear from the layout that the power and ground have higher priority (since they have much more current), ant the clock signal needs to work around them.

It's also interesting that a lot of the chip doesn't need the clock. The PLA, for one, is just combinatorial logic without any clock. The data pins have their own control lines, so unlike the address pins, they don't use the clock. The register storage uses unclocked inverter pairs to store each bit, so the clock is only used in a few places in the register file.

How the clock is used

Several circuits are commonly used in the Z80 with the clock. A pair of OR-NAND gates forms a latch that is enabled when the clock is low (i.e. reading the input), and holds the latched value when the clock is high. The inverse of this, a pair of AND-NOR gates forms a latch that is enabled when the clock is high. By combining the two, the Z80 creates a master-slave flip flop that is triggered on a rising clock edge. That is, the flip flop will grab the input value when the clock goes from low to high and will store this value until the next rising clock edge.

This edge-triggered master slave flip flop is used on input pins such as INT to ensure that the Z80 is using a stable value, since the input pin could change at arbitrary times since it is controlled by the outside world.

A very common circuit in the Z80 is a pass transistor controlled by the clock. This will pass the value through while the clock is high, and hold the value while the clock is low. This is similar in function to the AND-NOR latch.

A common idiom is to have a signal go through a clocked pass transistor and then into a OR-NAND latch. This operates somewhat like the master-slave flip flop. The latch will grab the signal on the falling clock edge. In more detail, while the clock is high, the signal will pass through the pass transistor. When the clock goes low, the signal will remain on the output of the pass transistor for a period of time, due to capacitance and the high impedance of MOS gates. As soon as the clock goes low, this signal will be stored in the latch. The latched signal is immune to changes on the pass transistor input while the clock is low. However, if the clock stays low too long, the signal on the pass transistor output will drain away and the latch can switch to the wrong value. This is one reason there's a maximum time the clock can stay low.

The key control signals in the Z80: the M lines, the T lines, and the PLA outputs - all change on a falling clock edge. That is, they hold their value through the low clock and high clock. This provides an important “rhythm” for the Z80. (Actually this isn't true for the PLA lines... rewrite this.)

A typical control output is generated by combining PLA, M, and T lines.

Many control lines are shifted half a cycle from the M/T/PLA signals, and change on a rising clock edge. For instance, consider the ALU shift left control line “c”. It is active during M1T4h and M1T1l. Or consider ALU to alubus control “x”, which is active during M1T2h and M1T3l.

To see how this happens for ALU c, we can look at simplified logic.
A bunch of PLA control lines and M and T signals are combined to generate w522, a signal expressed for certain instructions at certain states of execution. Because this signal is generated combinatorially, and the inputs all change only on a falling clock edge, the output will also change only on a falling clock edge and remain stable while the clock is low and then high.

Next, this signal goes through a clocked pass transistor and then generates ALU C after a bit more combinatorial logic. The pass transistor will pass its input when the clock is high and then hold it while the clock remains low. Thus, the output control signal will change on a rising clock edge, and will be delayed half a clock cycle compared to the inputs.

This is a very common pattern in the Z80, with a pass transistor causing signals to be delayed half a clock.

Following the signal a bit further, the ALU reads data off the bus through pass transistors controlled by signals such as ALU F and ALU LL which are active only when the clock is low.

The resulting timing is:

clock L	M, T, PLA change	
clock H	M, T, PLA stable	Control line changes
clock L		Control line stable ALU reads bus

Thus, the bus has a half-clock cycle to stabilize before it is read, and this is a full clock cycle after the M/T cycle that started the operation. This timing gives signals time to propagate through the chip and stabilize. However, this timing is slower than a chip such as the 6502 which does a separate operation

each phase of the clock. The 6502 uses a two-phase clock, which helps with this sort of aggressive timing.

Another interesting case is the flag control logic, and how flags get read from the register file and the data bus into the flag latches.

As in other control logic, PLA, M and T signals are combined to yield a control signal (w445) which changes state on a falling clock edge. (This signal is low for arithmetic operations in M1Tx, so they can modify the flags.)

This signal goes through a clocked pass transistor, delaying the signal half a clock, so it changes state on a rising clock edge. The signal then goes into a clocked OR-NAND latch, which is enabled by a falling clock edge, and delays the signal another half-clock. At this point, the signal is delayed a full clock cycle from the original M1Tx signal. The signal goes through another clocked pass transistor, delaying it and making it change state on rising clock. Finally, the signal is NANDed with the clock, making it active only while the clock is high, and truncating off the last half. The result is a control signal that is active while the clock is high, delayed three half-clocks from the original signal.

This is a common pattern, alternating between signals that switch on rising clock and signals that switch on falling clock, with the signal delayed half a clock by each switch. At the end, the signal can be trimmed to a single half-clock width if necessary.

Stepping back, it's worth looking at how the PLA, M and T control signals end up changing state on falling clock edges.

An instruction gets stored in the instruction register by the load_ireg control line. This line starts with M1T2, which then goes through a pass transistor, and then a clocked OR-NAND latch, which delays it until M1T3. Since this signal changes on a falling clock edge, so does the load_ireg signal. Thus, the instruction register changes on a falling clock edge. Since the PLA is purely combinatorial logic, the PLA outputs all change on falling clock edges.

(The load_ireg line is activated separately in M3 for a double-prefixed BITS + IX/IY instruction. These instructions are unusual, since the opcode is loaded in M3, after the displacement.)

But this pushes the question back one step further, to why M and T signals change on falling clock edges. The last_T signal is generated from a rather complex combination of M, T, and PLA signals. It goes through a clocked pass transistor (and is combined with some other signals that pass through clocked pass transistors) to generate the new T state control signal 1380/174 which changes on a rising clock edge. Each T state has a clocked OR-NAND latch, so each T state output changes on a falling clock edge.

For M cycles, the logic is different. Each M cycle has an AND-NOR latch, but it isn't controlled by the clock. Instead, it is controlled by w207, which goes high at the beginning of each M cycle (i.e. while the clock is low). As a result, M cycle changes happen on a falling clock edge. (Clocked AND-NOR latches switch on a rising clock edge, but w207 has opposite polarity to the clock, so the M latches have opposite behavior.)

Going back to w207, last_t goes through a clocked pass transistor to generate a signal changing on a rising clock edge. It then is NANDed with the clock. The result is a signal that is usually 0, but may be 1 when the clock is low. Thus, this NANDing with the clock is the fundamental reason for the M states to switch on a falling clock edge.

The motivation for the clock

The clock is critical in a microprocessor to keep everything from happening at once and degenerating into chaos.

Simple, combinatorial circuits do not need a clock. In a combinatorial circuit, inputs are fed into a bunch of gates and produce outputs. But if a circuit has feedback, typically a clock is needed to keep the circuit from oscillating.

For instance, consider a circuit to increment the accumulator. An incrementer is a straightforward combinatorial circuit that takes a binary input and outputs

the incremented value. The accumulator can be fed into the incrementer to generate the incremented value. But if this value is looped back to the accumulator, the circuit will become unstable, incrementing the updated accumulator value over and over.

The circuit can be stabilized by using edge-triggered flip flops to hold the accumulator value. Only on the clock edge will the accumulator be updated, so it will be incremented by one on each edge.

The problem with an edge-triggered approach such as this is that the clock may not get to all parts of a complex circuit at the same time, so some flip flops may update before others. During this time, modified signals may propagate from the updated flipflops to the ones that are not updated, causing erroneous value.

This problem can be avoided by using a master-slave model. During one clock phase, the master register is updated. During the second clock phase, the slave register is updated. In this way there is no risk of asynchronous updates, as long as the two phases leave enough time for propagation delays.

This model is very asymmetrical, since all the computation happens going into the master phase, and the second phase just copies the data. For this reason, the computation can be broken into two more-equal phases, where half the computation happens and is latched during the first phase, and the other half of the computation happens and is latched during the second phase. This can be called a “polyphase” approach.

The polyphase approach closely matches the Z80's clock structure. The first phase happens while the clock is high. The computations from this phase are latched by pass transistors when the clock goes low.

See chapter 4 “Timing architecture” in “The Architecture of Microprocessors” by François Anceau.