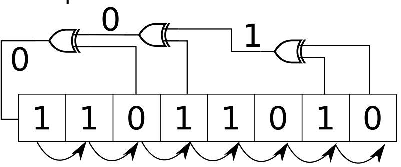
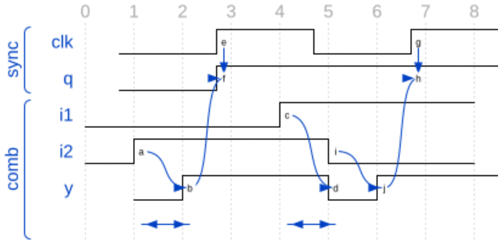
LFSR / FPGA stuff

Principle:



Combinatorial / sequencial concepts

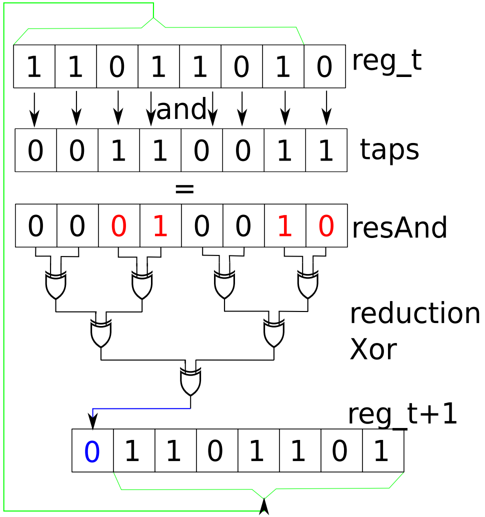


Basic Amaranth concept:

```
class MyClass(Elaboratable):

    def elaborate(self, platform):
        m = Module()

        cSig = Signal()
        sSig = Signal(2)

        m.d.comb += cSig.eq(sSig[1] ^ sSig[0])
        m.d.sync += sSig.eq(Cat(sSig[1], cSig))

        return m
```

amaranth → verilog → synthesis (yosys, Vivado, ...) → PnR (nextpnr, vtr, vivado, ...) → bitstream → programmer

LFSR: basic idea



AND truth table:

i1\i2	0	1
0	0	0
1	0	1

TAPs with AND: mask ($y_n = i1_n \& i2_n$)

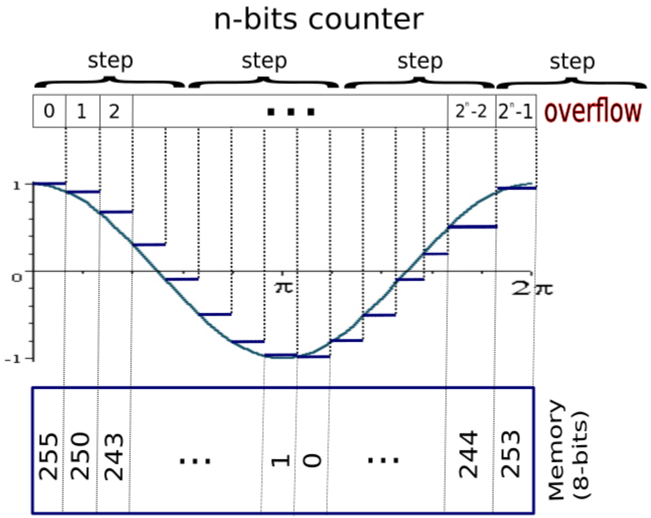
XOR truth table:

i1\i2	0	1
0	0	1
1	1	0

reduction Xor: ($y = i_0 \oplus i_1 \oplus i_2 \oplus \ldots \oplus i_{n-1}$)

With taps set at synthesis time $\rightarrow$ gateware is optimize (no more AND, picks relevant bits and improve the LUT's truth table).

NCO


n-bits counter
step
step
step
step
0
1
2
...
2ⁿ-2
2ⁿ-1
overflow
1
0
-1
π
2π
255
250
243
...
1
0
...
244
253
Memory
(8-bits)