TÉCNICO
LISBOA

Computer Electronics

Lecture 9: Dedicated Processors

Why design a dedicated processor

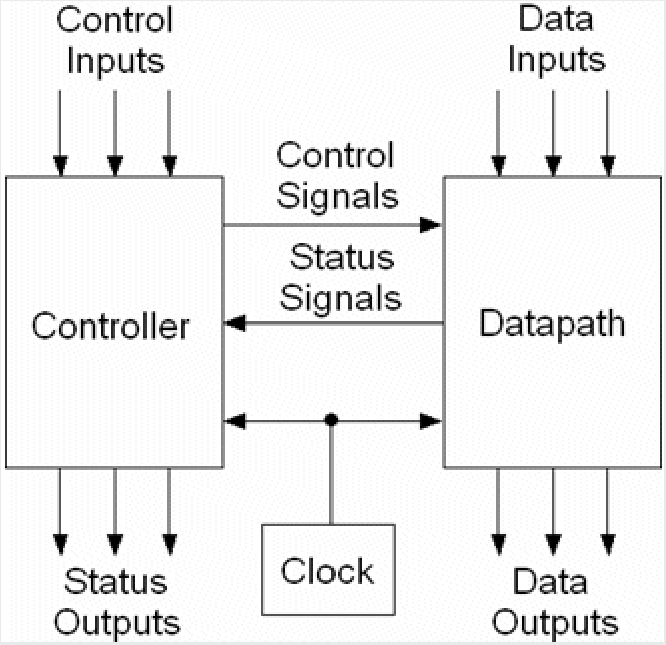
- Not needed for anything else
- No or limited programmability needed
- Can reduce cost in volume production
- Can reduce power/energy consumption
- Better *form factor* (size matters)
- Has a simpler user interface

Dedicated processors combined with GPPs

- PCIe cards
- Graphics cards
- Surveillance equipment
- Audio equipment
- Etc.

Design of dedicated processors

- Data and control paths



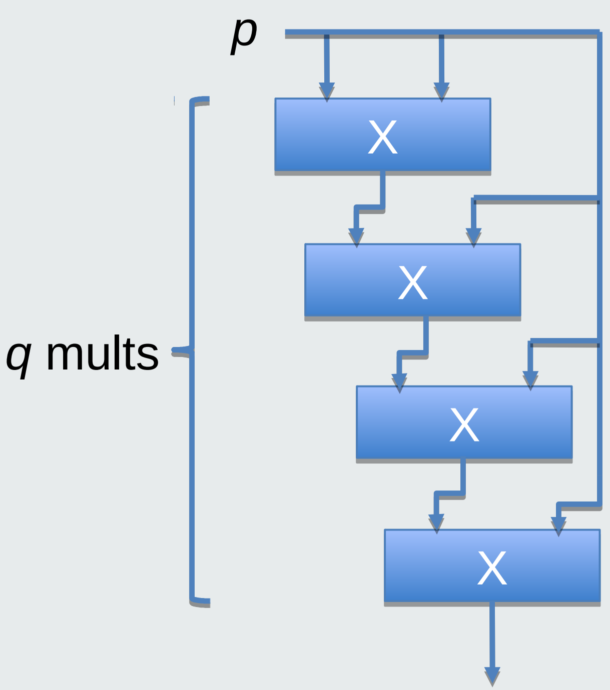
The Art of Design

- Describe the algorithm in a high-level text description
- Draw the datapath
- Write the pseudo-code for the control path (normally a FSM)
- Make sure you list:
  - IO signals: name, width, direction (I,O,IO), description
  - Registers: name, address, direction (Read, Write, RW), description
  - Memories: name, function (instructions, data, ..), volatile/non volatile, type (ROM,EPROM,RAM,reg file), internal/external, ports (single, dual, two-port, dual-port
    - Holy Grail of Computing: N-port memory!!
  - Data streams: name, description, width, continuous/interrupted
  - Buses: name, purpose, type (Native, AXI, Wishbone, ISA, …)
- Draw timing diagrams (use wavedrom, simulation of small circuits)

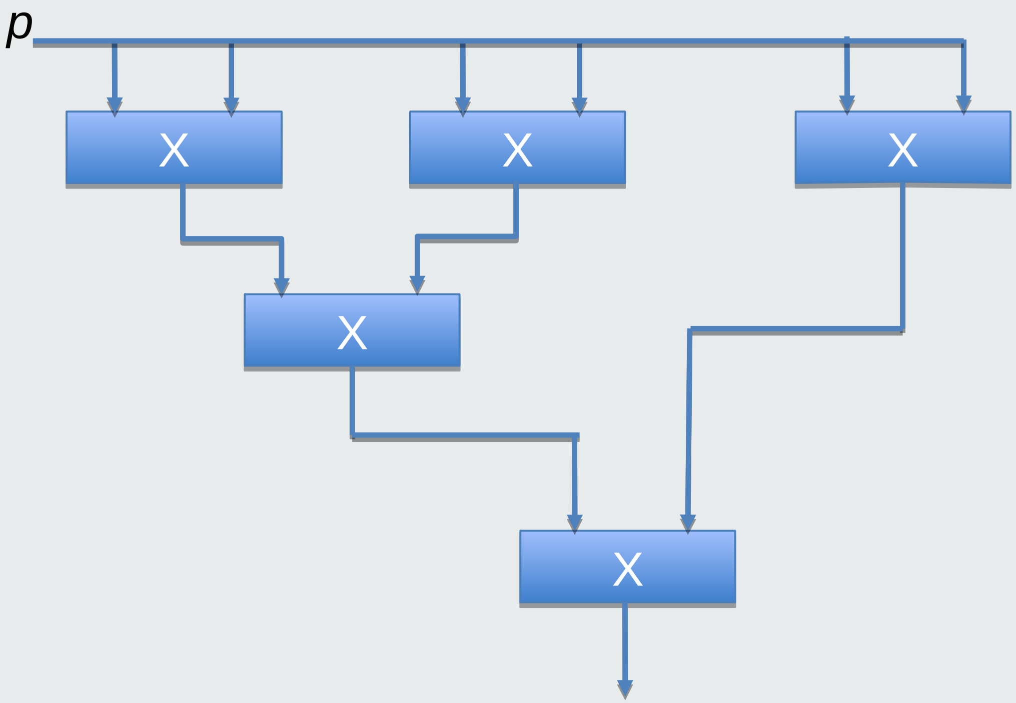
Design example: dedicated processor that computes p^q

- *p* is a streaming input (*volatile in C language*)
- Textual description already given
- Many possible architectures
- Many possible trade-offs…
- Let's get to the block diagram

Parallel architectures



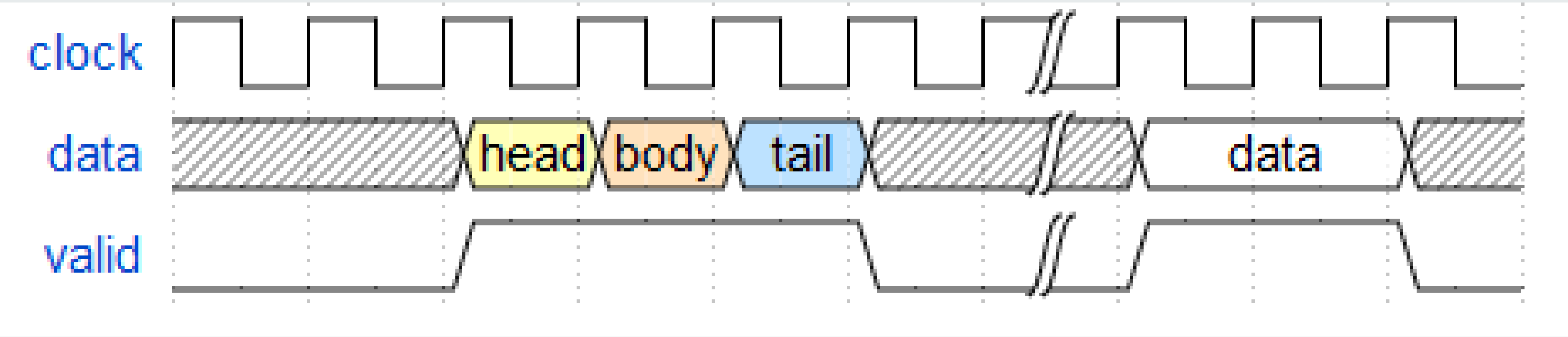
Linear array



Tree structure

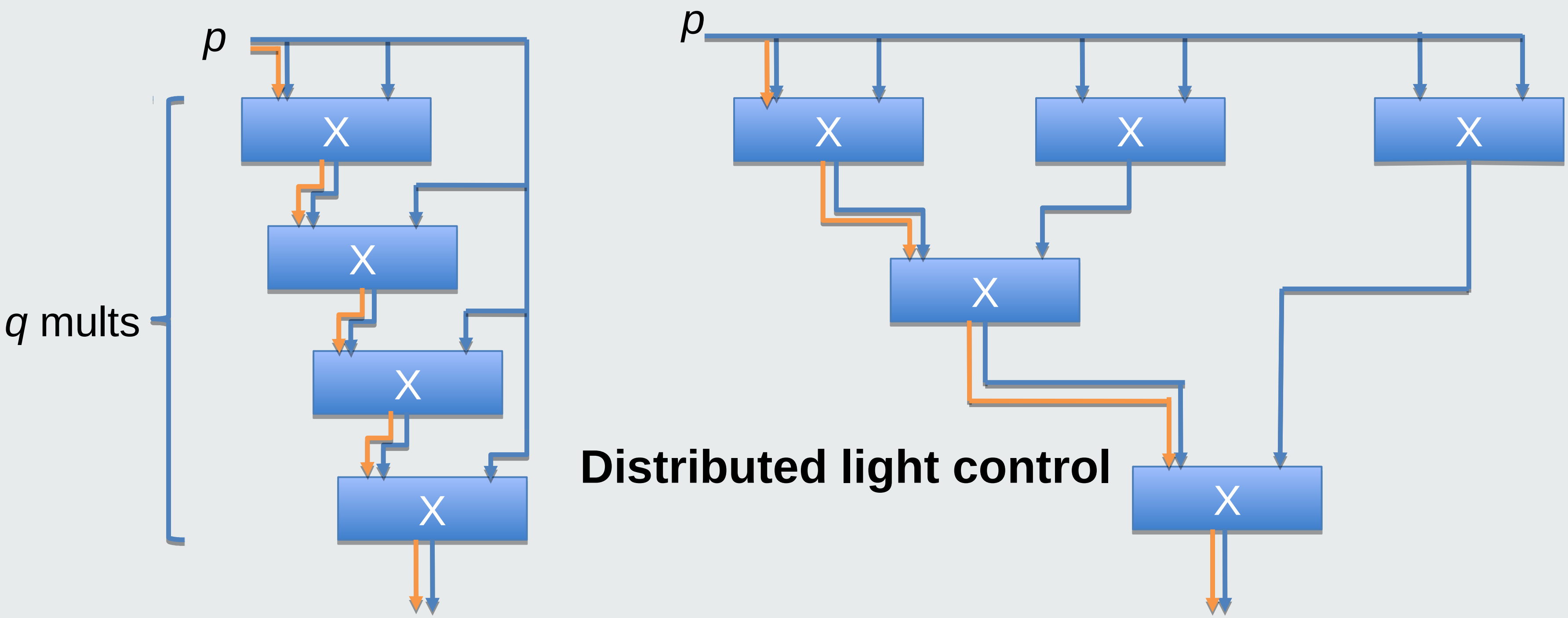
Same throughput lower latency

Data valid signals



Interesting online tool: http://wavedrom.com/editor.html

Propagation of the valid signal



Distributed light control

Linear array

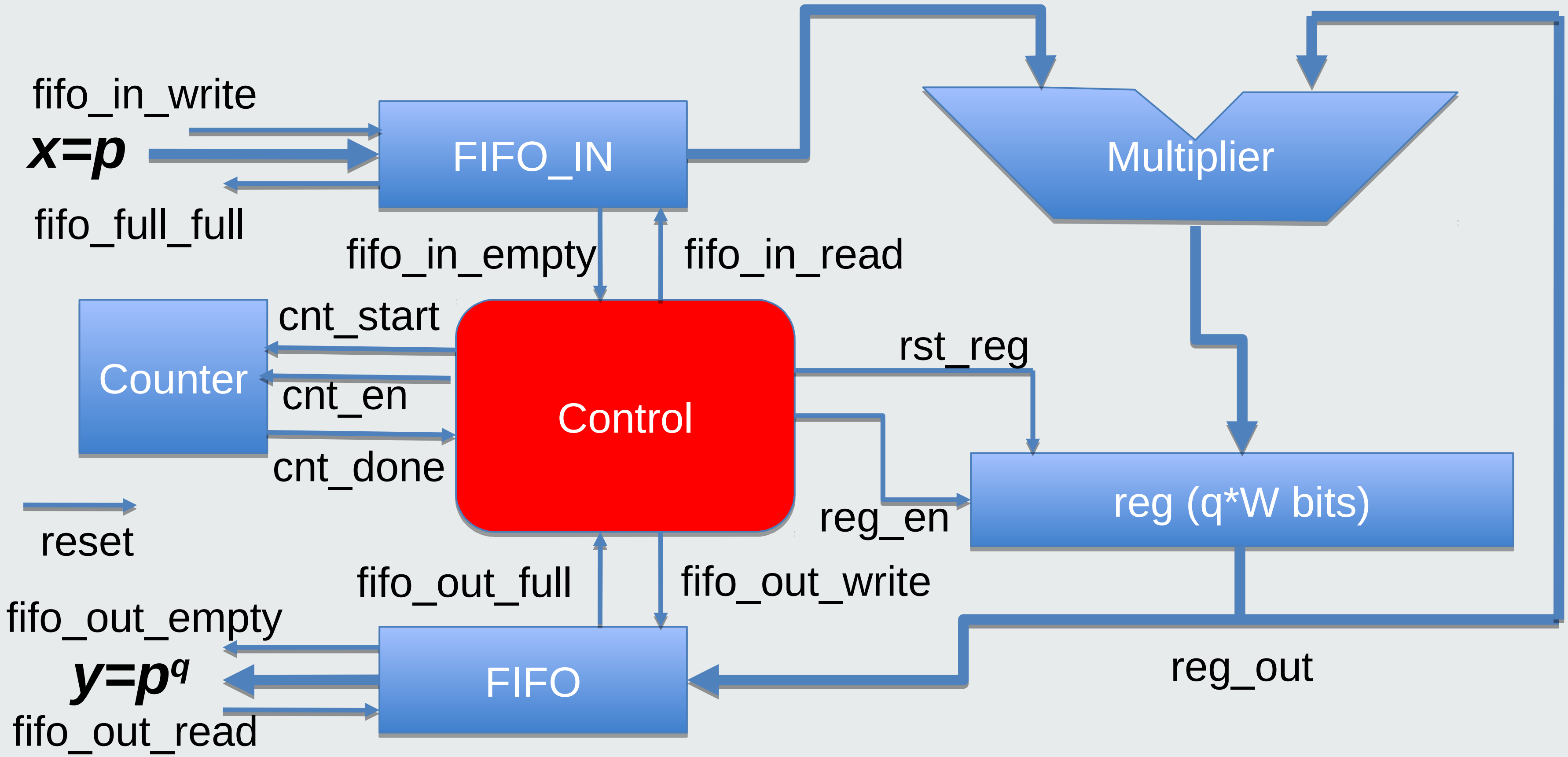
Tree structure

Same throughput lower latency

Parallel architecture: software accessible registers

- Register *p:* write-only, 16 bits for example
- Register *q:* write-only, 16 bits for example
- Register *r* (result): read-only, 32 bits for example
- Result comes fast: start and done registers may be dispensed with

Serial architecture: block diagram



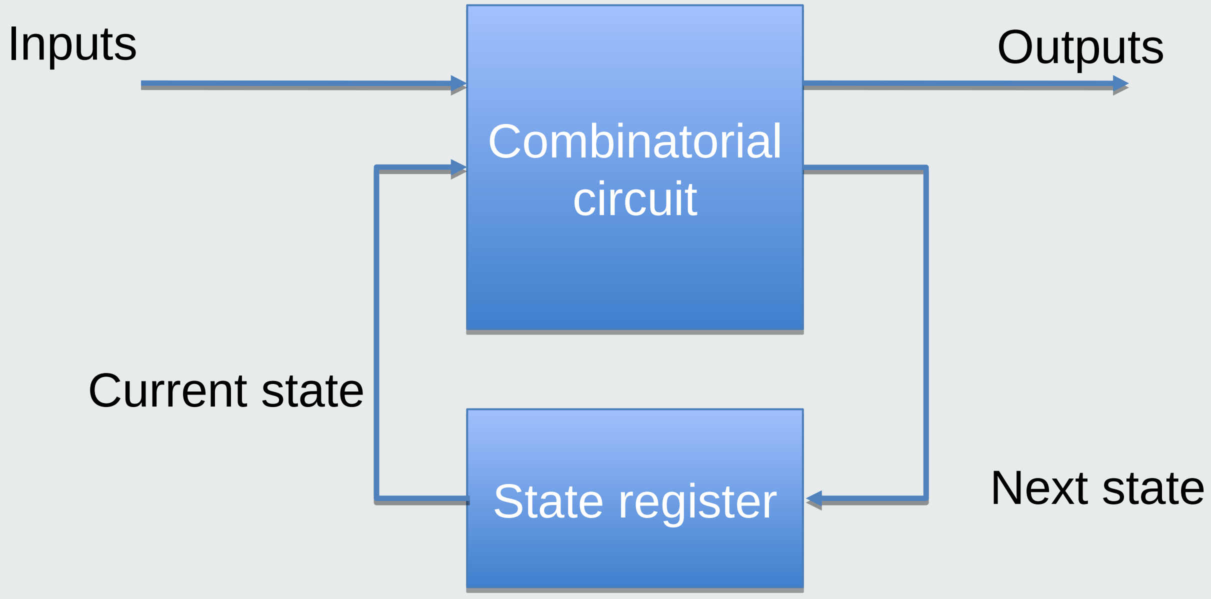
Serial architecture: software accessible registers

- Register *p:* write-only, 16 bits for example
- Register *q:* write-only, 16 bits for example
- Register *r* (result): read-only, 32 bits for example
- Result comes several clock cycles later, especially if *q* is large:
  - Start (1 bit, write-only) and Done (1 bit, write-only) registers are advisable
  - Alternatively, the timer can be used to wait for exactly the time the calculation will take

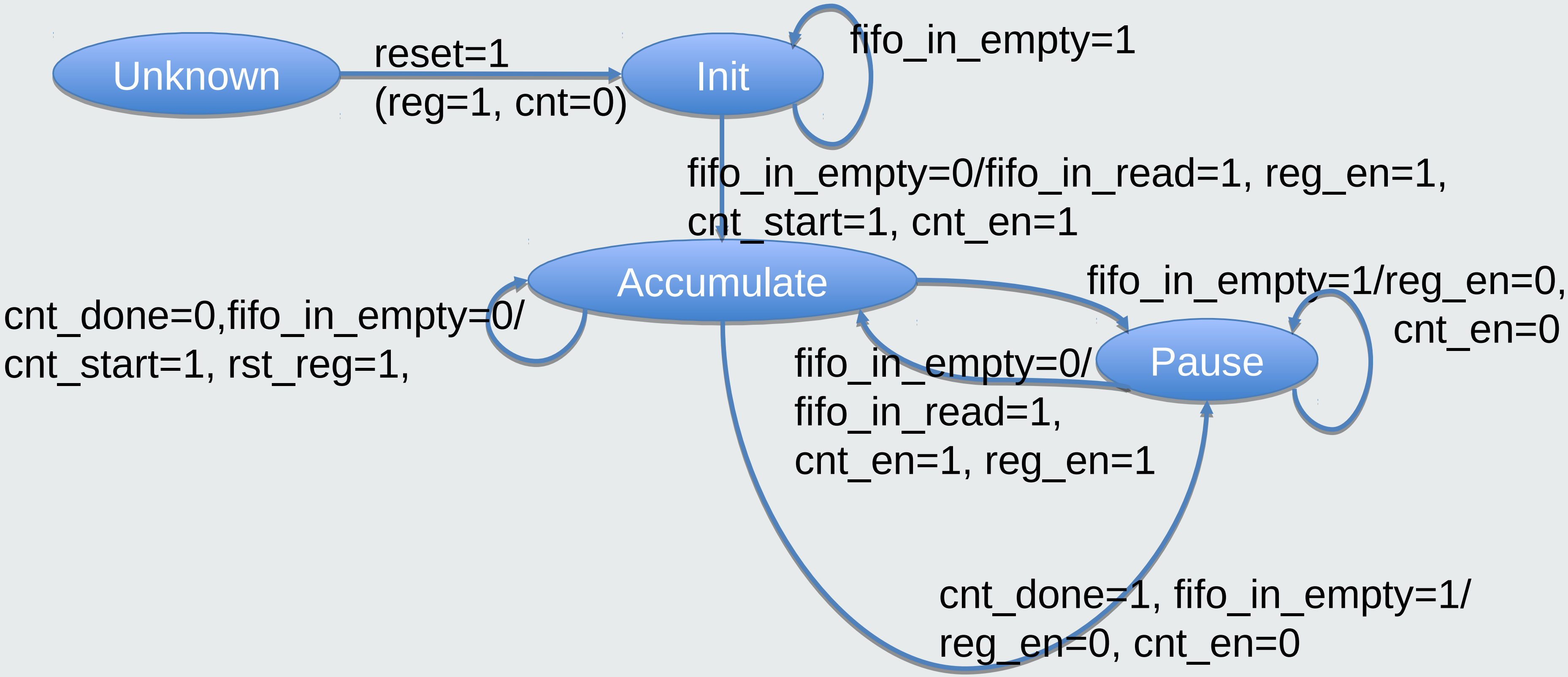
Design control circuit

- Flowchart … better for software
- State transition diagram or table
- Pseudo-code
- Experienced designer: HDL (Verilog, VHDL, etc)

Controller canonical form



State transition diagram



reset=1
(reg=1, cnt=0)

fifo_in_empty=1

fifo_in_empty=0/fifo_in_read=1, reg_en=1,
cnt_start=1, cnt_en=1

cnt_done=0,fifo_in_empty=0/
cnt_start=1, rst_reg=1,

fifo_in_empty=1/reg_en=0,
cnt_en=0

fifo_in_empty=0/
fifo_in_read=1,
cnt_en=1, reg_en=1

cnt_done=1, fifo_in_empty=1/
reg_en=0, cnt_en=0

State transition table (very incomplete… no space!)

state	Inputs				Outputs			Next state
st	reset	cnt_done	fifo_in_empty	fifo_out_full	fifo_in_read	fifo_out_write	Reg_en	nxt_st
?	1	-	-	-	-	-	-	init
Init	0	0	1	0	0	0	0	init
Init	0	0	0	0	1	0	0	Accumulate
Accumulate								

Write finite state machines that look like assembly programs!

This program or FSM does not work, only good for teaching ==>

```
`SIGNAL(pc, 3)
`SIGNAL(pc_next, 3)
 `REG_AR(bclk, rst_bclk, 3'b0, pc, pc_next)

 `COMB begin

   //default assignments
   pc_next = pc+1'b1;

   rst_reg = 1'b0;
   start_counter = 1'b0;
   en_counter = 1'b0;
   …

  case(pc)

    0: if (!START) begin //wait for sw start
          pc_next = pc;
    1: if (fifo_in_empty)
          pc_next = pc;
        else begin
           rst_reg = 1;
           start_counter = 1;
        end
    2: begin
          en_reg = 1;
          en_counter = 1;
    3: if (counter != q)
           pc_next <= 1;
   default: begin
           pc_next <= pc;
           done <= 1;

   endcase;

end // `COMB
```

Next steps

- Write HDL code (say Verilog) and simulate
- Run synthesis
  - Compiles Verilog and produces a generic netlist (gates and flip-flops not related to any technology)
- Run mapping
  - Converts generic netlist into technology dependent netlist using a component library
- Run place and route
  - Places cells and wire connections into layout
- If user time constraint violations tweak design or tool options and rerun
- Complex tools: full configuration requires scripting (in tcl normally)
- A competent designer or design manager can earn a few hundred K/year