INF2270 — Spring 2010

Philipp Häfliger

Lecture 8: Superscalar CPUs, Course Summary/Repetition (1/2)

UNIVERSITETET I OSLO

content

From Scalar to Superscalar

Lecture Summary and Brief Repetition

- Binary numbers
- Boolean Algebra
- Combinational Logic Circuits
  - Encoder/Decoder
  - Multiplexer/Demultiplexer
  - Adders
- Sequential Logic Circuits
  - Counters
  - Shift Registers

content

From Scalar to Superscalar

Lecture Summary and Brief Repetition
- Binary numbers
- Boolean Algebra
- Combinational Logic Circuits
  - Encoder/Decoder
  - Multiplexer/Demultiplexer
  - Adders
- Sequential Logic Circuits
  - Counters
  - Shift Registers

Scalar Processors

The concept of a CPU that we have discussed so far where all scalar processors, in as far as they do not execute operations in parallel and produce only a single result data item at a time.

Vector processors

High performance computing led to vector processors, most prominently the Cray-1 in 1976 that had 8 vector registers of 64 words of 64-bit length. Vector processors perform 'single instruction multiple datastream' (SIMD) computations, i.e. they execute the same operation on a vector instead of a scalar. Some machines used parallel ALU's but the Cray-1 used a dedicated pipelining architecture that would fetch a single instruction and then execute it efficiently, e.g. 64 times, saving 63 fetches.

Multi processor

Vector computers lost popularity with the introduction of multi-processor computers such as Intels's Paragon series of *massively parallel supercomputers*: It was cheaper to combine multiple (standard) CPU's rather than designing powerfull vector processors, even considering a bigger *communication overhead*, e.g. in some architectures with a single shared memory/system bus the instructions and the data need to be fetched and written in sequence for each processor, making the von Neumann bottleneck more severe. Other designs, however, had local memory and/or parallel memory access and many clever solutions were introduced.

Clusters/Grids

But even cheaper and obtainable for the common user are Ethernet clusters of individual computers, or even computer grids connected over the internet. Both of these, obviously, suffer from massive communication overhead and espescially the latter are best used for so called 'embarassingly parallel problems', i.e. computation problems that do require no or minimal communication of the computation nodes.

Multi Core

Designing more complicated integrated circuits has become cheaper with progressing miniaturization, such that several processing units can now be accomodated on a single chip which has now become standard with AMD and Intel processors. These multi-core processors have many of the advantages of multi processor machines, but with much faster communication between the cores, thus, reducing communication overhead. (Although, it has to be said that they are most commonly used to run individual independent processes, and for the common user they do not compute parallel problems.)

Superscalar Processor Principle

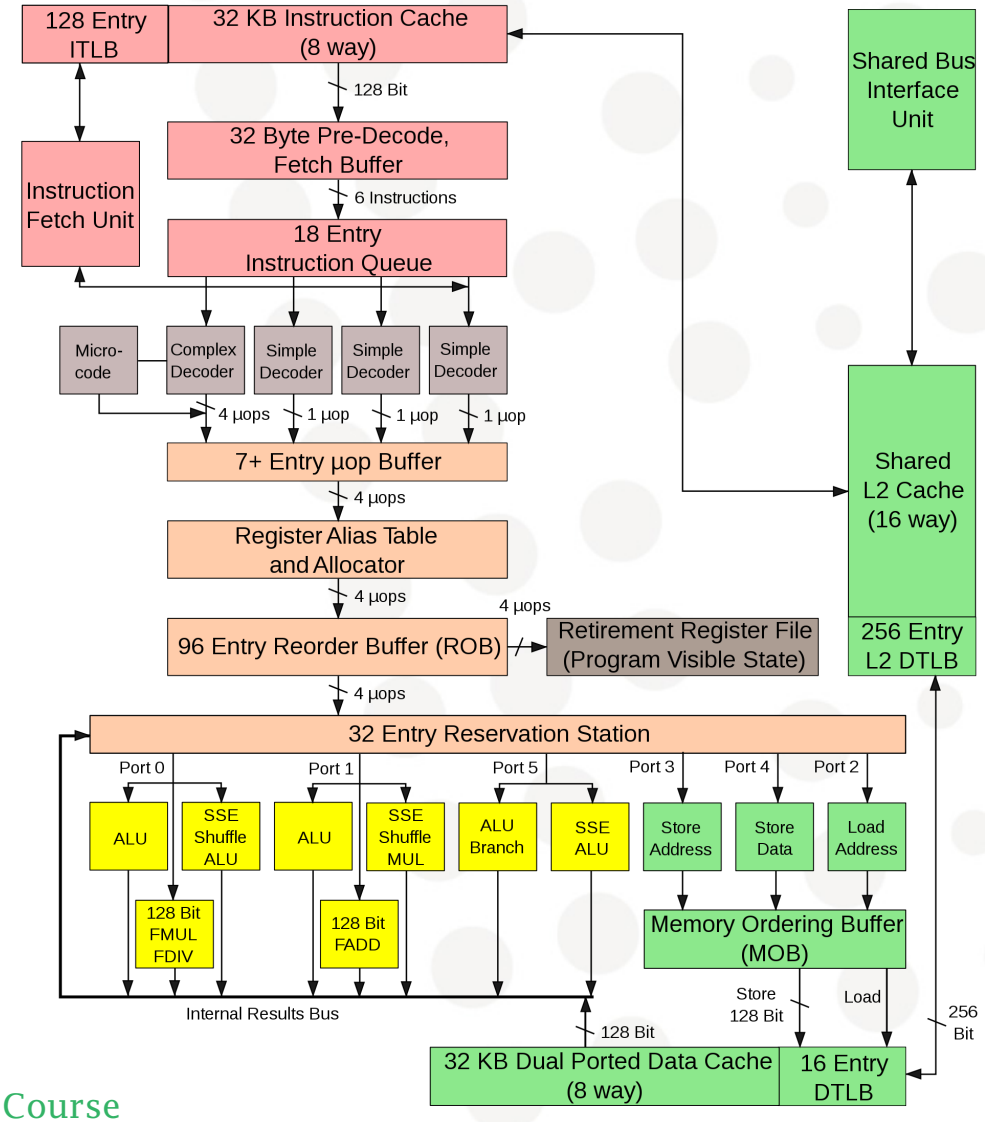
Superscalar processors were introduced even before multi-core and all modern designs belong to this class. Like vector processeors with parallel ALUs, they are actually capable of executing instructions *in parallel*, but in contrast to vector computers, they are *different* instructions. Instead of replication of the basic functional units n-times in hardware (e.g. the ALU), superscalar processors exploit the fact that there already *are* multiple functional units. For example, many processors do sport both an ALU and a FPU. Thus, they should be able to execute an integer- and a floating-point operation *simultaneously*. Data access operations do not require the ALU nor the FPU (or have a dedicated ALU for address operations) and can thus also be executed at the same time.

Superscalar Processor

For this to work, several instructions have to be fetched in parallel, and then *dispatched*, either in parallel, if possible, or in sequence, if necessary. Some additional stages are needed in the pipelining structure, and the pipeline is divided for differnt types of instructions.
Superscalar processors can ideally achive an average clock cycle per instruction (CPI) smaller than 1, and a speedup higher than the number of pipelining stages k (which is saying the same thing in two different ways).
Compiler level support can group instructions to optimize the potential for parallel execution.

Intel Core 2

As an example: the Intel Core 2 microarchitecture has 14 pipeline stages and can execute up to 4-6 instructions in parallel.


128 Entry ITLB
32 KB Instruction Cache (8 way)
128 Bit
32 Byte Pre-Decode, Fetch Buffer
6 Instructions
Instruction Fetch Unit
18 Entry Instruction Queue
Micro-code
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
4 µops
1 µop
1 µop
1 µop
7+ Entry µop Buffer
4 µops
Register Alias Table and Allocator
4 µops
4 µops
96 Entry Reorder Buffer (ROB)
Retirement Register File (Program Visible State)
4 µops
32 Entry Reservation Station
Port 0
Port 1
Port 5
Port 3
Port 4
Port 2
ALU
SSE Shuffle ALU
ALU
SSE Shuffle MUL
ALU Branch
SSE ALU
Store Address
Store Data
Load Address
128 Bit FMUL FDIV
128 Bit FADD
Memory Ordering Buffer (MOB)
Internal Results Bus
Store 128 Bit
Load
128 Bit
256 Bit
32 KB Dual Ported Data Cache (8 way)
16 Entry DTLB
Shared Bus Interface Unit
Shared L2 Cache (16 way)
256 Entry L2 DTLB

Some Elements in Superscalar Architectures (1/2)

Micro-instruction reorder buffer (ROB): Stores all instructions that await execution and dispatches them for *out-of-order execution* when appropriate. Note that, thus, the order of execution may be quite different from the order of your assembler code. Extra steps have to be taken to avoid and/or handle hazards caused by this reordering.

Retirement stage: The pipelining stage that takes care of finished instructions and makes the result appear consistent with the execution sequence that was intended by the programmer.

Some Elements in Superscalar Architectures (2/2)

Reservation station registers: A single instruction reserves a set of these registers for all the data needed for its execution on its functional unit. Each functional unit has several slots in the reservation station. Once all the data becomes available and the functional unit is free, the instruction is executed.

content

From Scalar to Superscalar

Lecture Summary and Brief Repetition

- Binary numbers
- Boolean Algebra
- Combinational Logic Circuits
  - Encoder/Decoder
  - Multiplexer/Demultiplexer
  - Adders
- Sequential Logic Circuits
  - Counters
  - Shift Registers

Lecture Content on Hardware

A rough categorization of the content:

- Digital Logic (Boolean algebra, combinational and sequential logic ...)
- Architecture (Von Neumann, cache, virtual memory, I/O ...)
- Performance Optimization (pipelining, cacheing and virtual memory strategies ...)

content

From Scalar to Superscalar

Lecture Summary and Brief Repetition

- Binary numbers
- Boolean Algebra
- Combinational Logic Circuits
  - Encoder/Decoder
  - Multiplexer/Demultiplexer
  - Adders
- Sequential Logic Circuits
  - Counters
  - Shift Registers

Binary Numbers

unsigned int: '10010' corresponds to

$$1\times2^4+0\times2^3+0\times2^2+1\times2^1+0\times2^0 = 16+2 = 18$$

int, two's complement: for n-bit integers

$$\begin{aligned} & (\text{unsigned int}) && [2^{(n-1)}, 2^n - 1] \\ = \quad & (\text{int}) && [-2^{(n-1)}, -1] \end{aligned}$$

$$\begin{aligned} & (\text{unsigned int}) && [0, 2^{(n-1)} - 1] \\ = \quad & (\text{int}) && [0, 2^{(n-1)} - 1] \end{aligned}$$

content

From Scalar to Superscalar

Lecture Summary and Brief Repetition

- Binary numbers
- Boolean Algebra
- Combinational Logic Circuits
  - Encoder/Decoder
  - Multiplexer/Demultiplexer
  - Adders
- Sequential Logic Circuits
  - Counters
  - Shift Registers

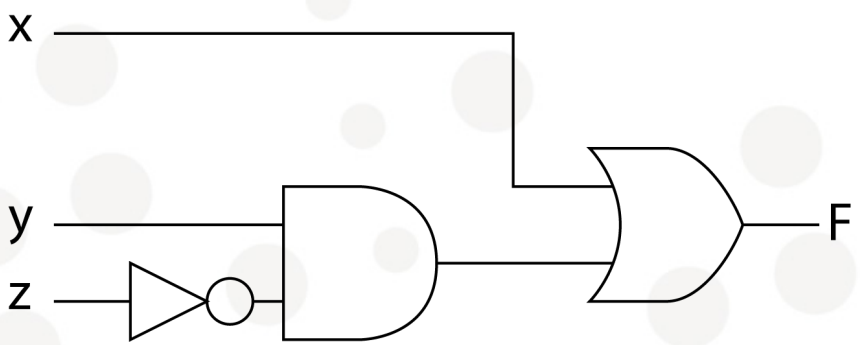
Boolean Function

- A (Boolean) function assigns exactly one output (or one output vector) to every input vector.
- Boolean expressions are composed of the three basic Boolean algebraic operators, AND, OR, and NOT
- Boolean functions can be defined by
  - Boolean expressions
  - Truth tables
  - Logic gates schematics
- Functions are identical/equivalent if they produce the same output for every input. Note: different expressions/schematics can describe the *same* function. There is only one complete truth table, however, for one function.

Boolean function Example

$$F = x \vee y \wedge \bar{z}$$

x	y	z	F
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	1



Rules governing equivalency

$$\bar{\bar{a}} = a$$

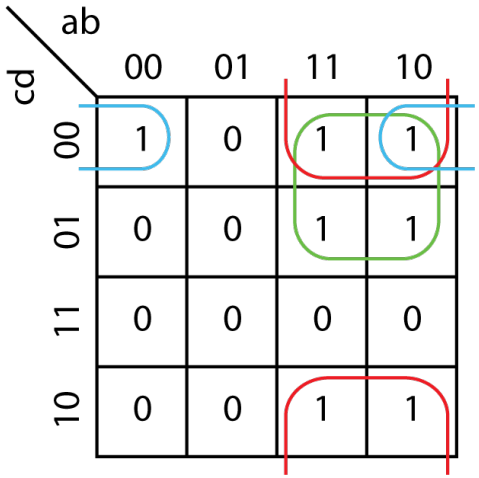
$a \wedge b \vee c = (a \wedge b) \vee c$	$a \vee b \wedge c = a \vee (b \wedge c)$	
$a \wedge \bar{a} = 0$	$a \vee \bar{a} = 1$	
$a \wedge a = a$	$a \vee a = a$	
$a \wedge 1 = a$	$a \vee 0 = a$	
$a \wedge 0 = 0$	$a \vee 1 = 1$	
$a \wedge b = b \wedge a$	$a \vee b = b \vee a$	(commutative)
$(a \wedge b) \wedge c = a \wedge (b \wedge c)$	$(a \vee b) \vee c = a \vee (b \vee c)$	(associative)
$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$	(distributive)
$\overline{a \vee b} = \bar{a} \wedge \bar{b}$	$\overline{a \wedge b} = \bar{a} \vee \bar{b}$	(deMorgan)

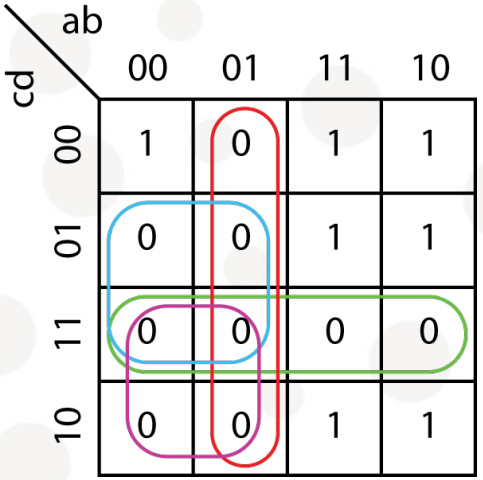
Simplification

Since there are infinitely many equivalent Boolean expressions for the same function, it is often desireable to find a simple expression for a given function. In the lecture we looked at two methods:

1. Intuitive application of the algebraic rules
2. Karnaugh maps

Example Karnaugh map





$$F = a \wedge \bar{c} \vee a \wedge \bar{d} \vee \bar{b} \wedge \bar{c} \wedge \bar{d}$$

$$\bar{F} = \bar{a} \wedge d \vee \bar{a} \wedge c \vee \bar{a} \wedge b \vee c \wedge d$$

$$F = (a \vee \bar{d}) \wedge (a \vee \bar{c}) \wedge (a \vee \bar{b}) \wedge (\bar{c} \vee \bar{d})$$

content

From Scalar to Superscalar

Lecture Summary and Brief Repetition

- Binary numbers
- Boolean Algebra
- Combinational Logic Circuits
  - Encoder/Decoder
  - Multiplexer/Demultiplexer
  - Adders
- Sequential Logic Circuits
  - Counters
  - Shift Registers

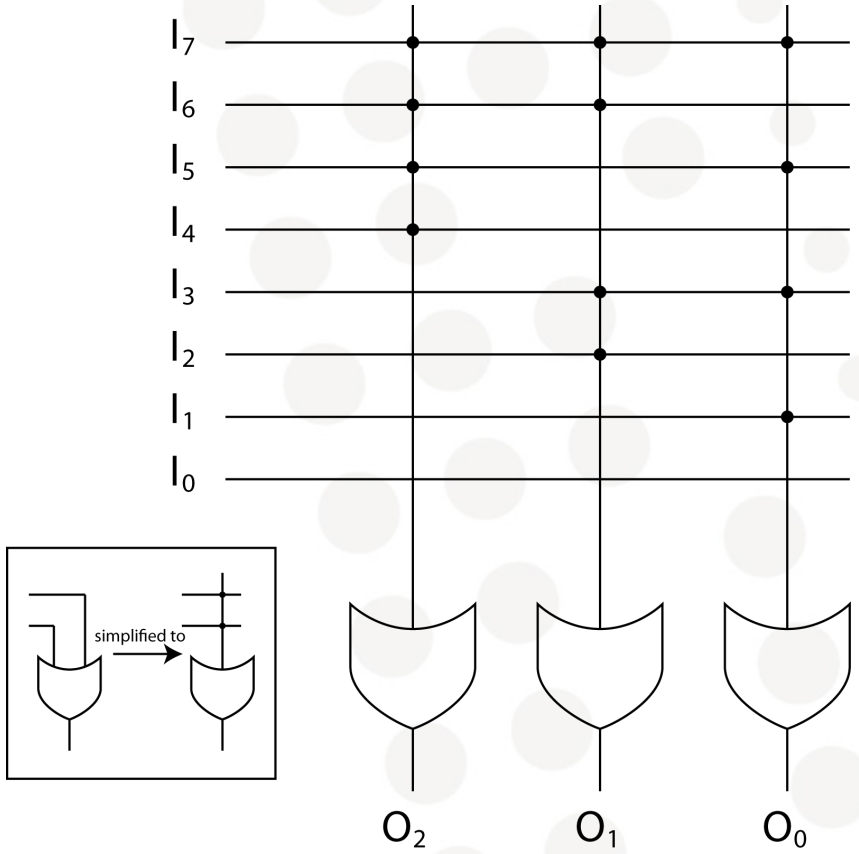
Definition

Combinational Logic circuits are circuits implementing Boolean functions

Simple 3-bit Encoder Truth Table

I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	O_2	O_1	O_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

3-bit Encoder Implementation Variant



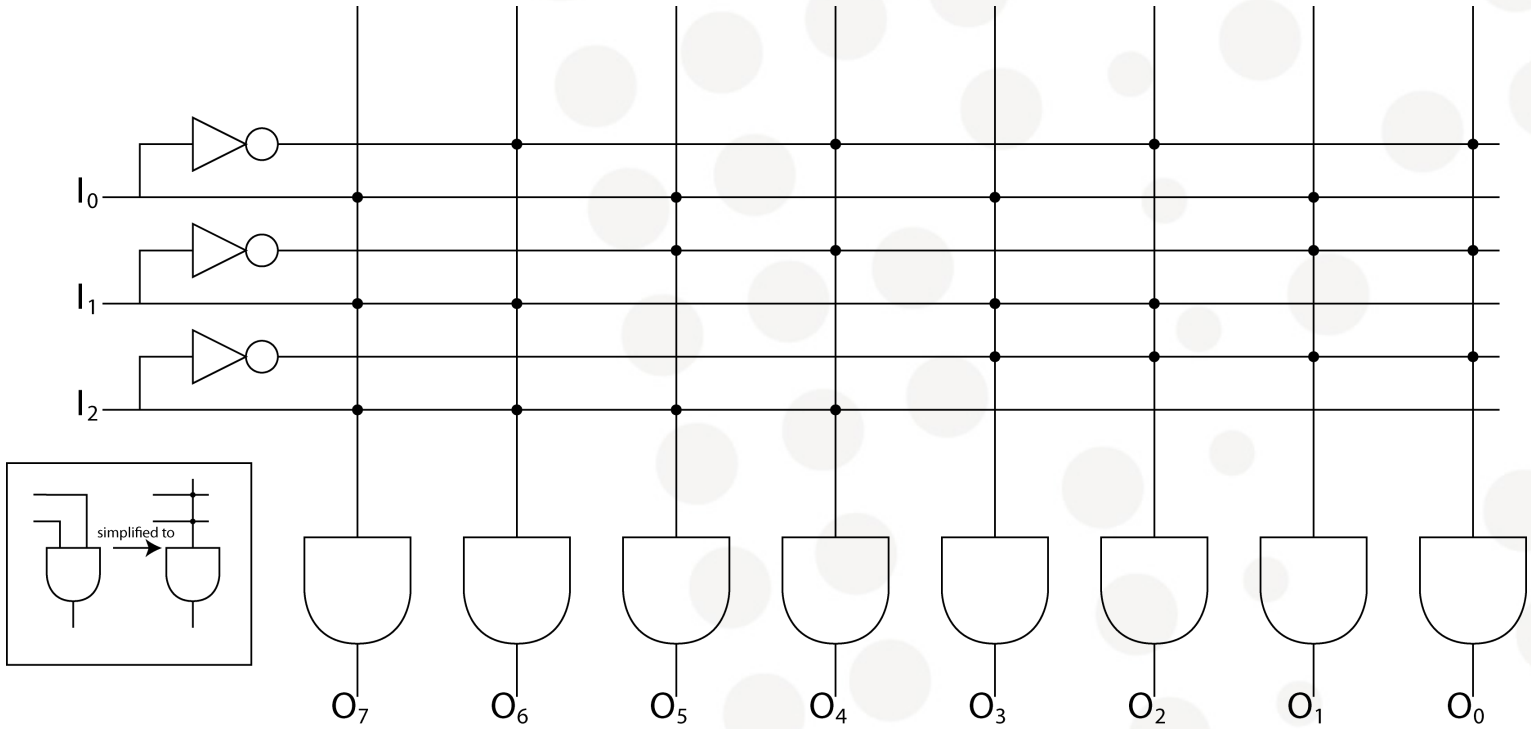
3-bit Priority Encoder Truth Table

I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	O_2	O_1	O_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

3-bit Decoder Truth Table

I_2	I_1	I_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

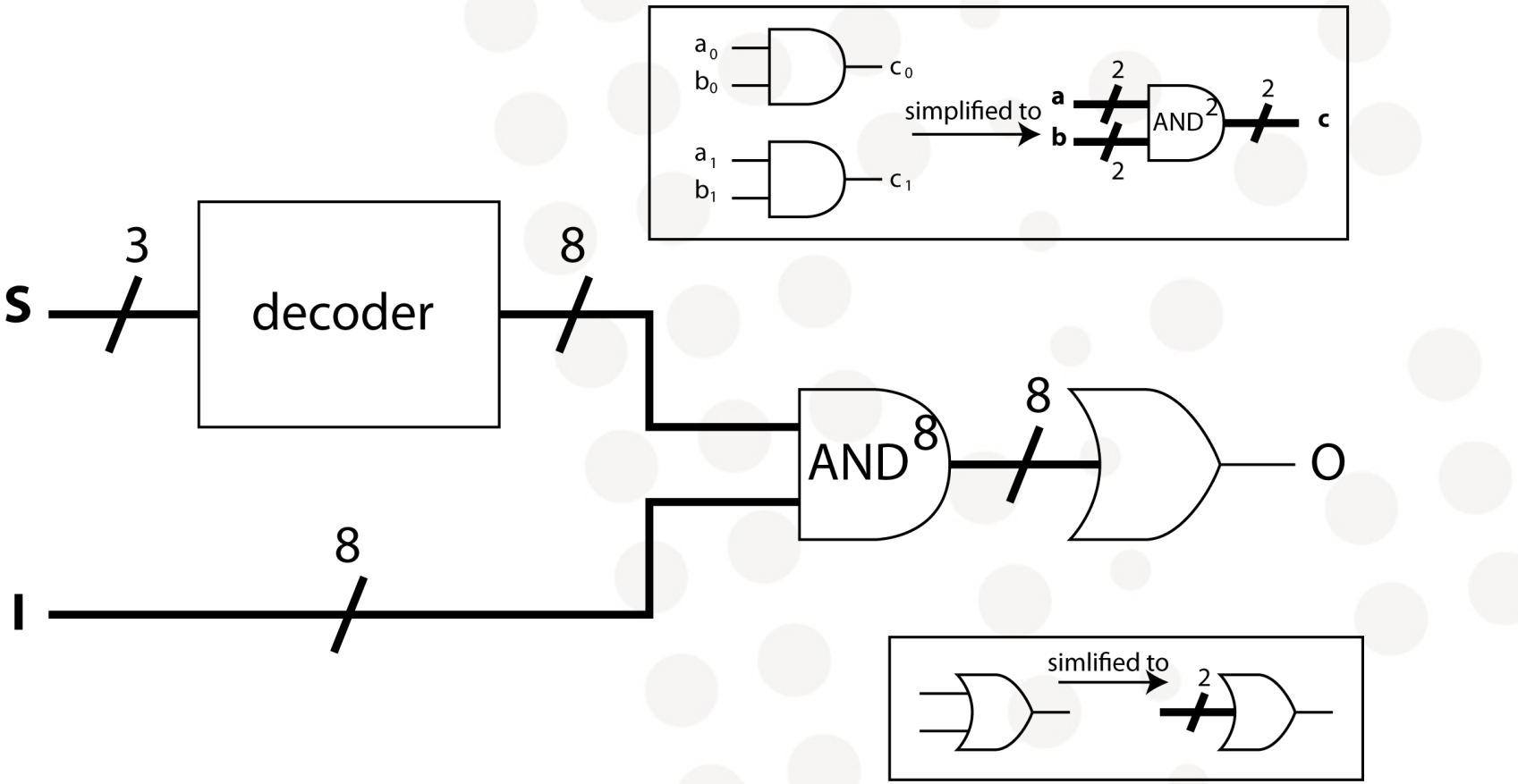
3-bit Decoder Implementation Variant



3-bit Multiplexer Truth Table

S_2	S_1	S_0	O
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

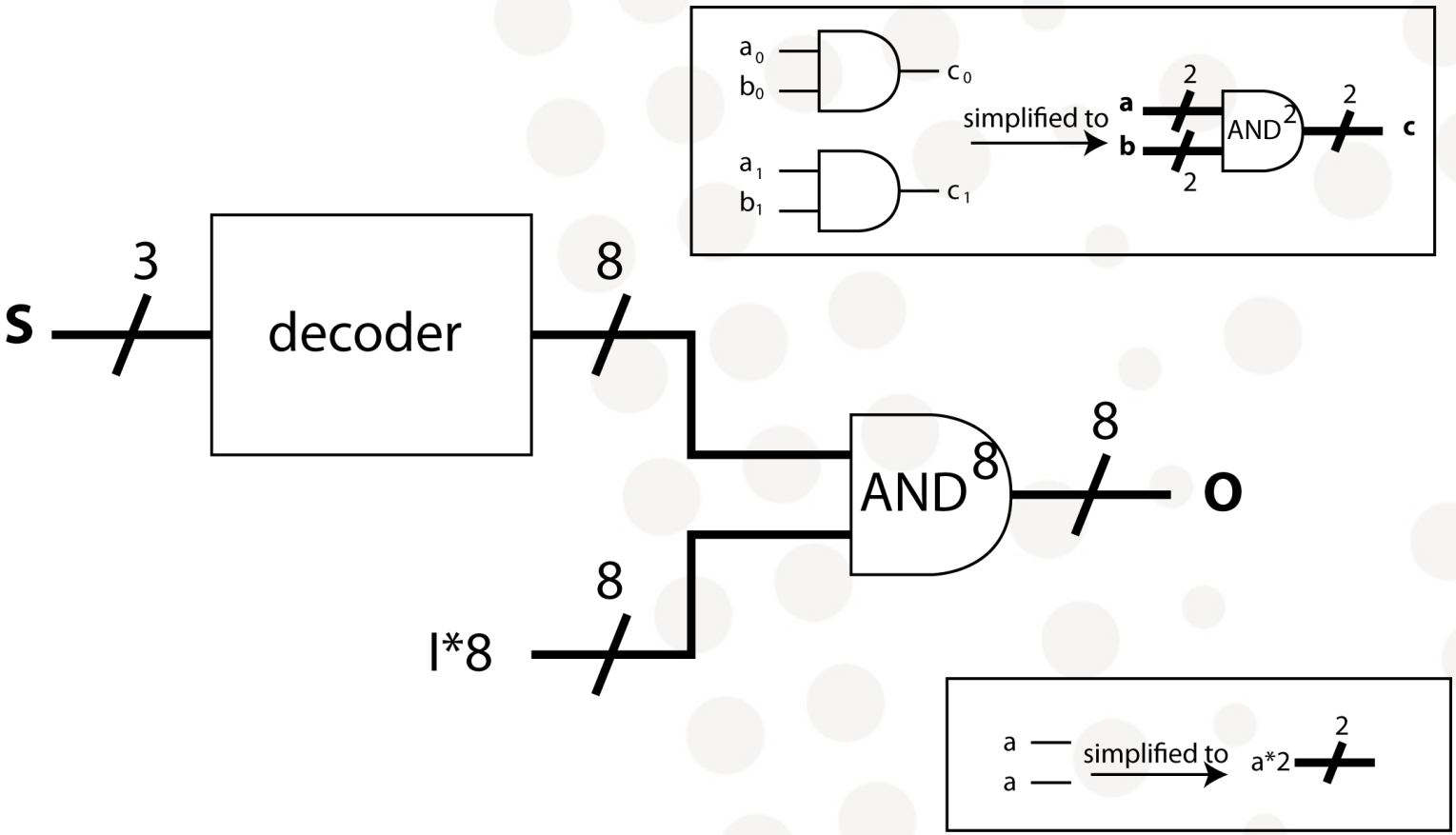
3-bit Multiplexer Implementation Variant



3-bit Demultiplexer Truth Table

S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

3-bit Demultiplexer Implementation Variant

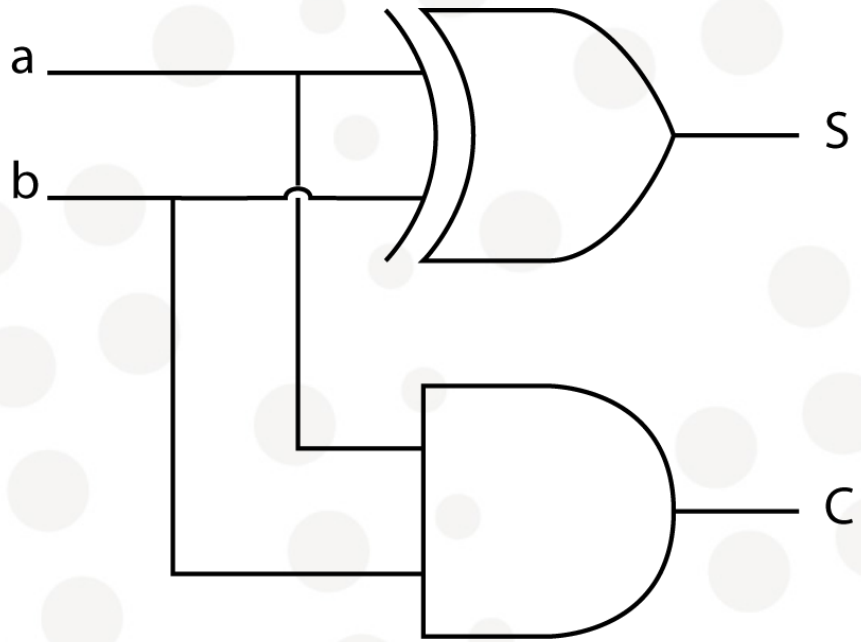


Half Adder

Truth table for a 1-bit half adder:

a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Schematics:



Full Adder (1/2)

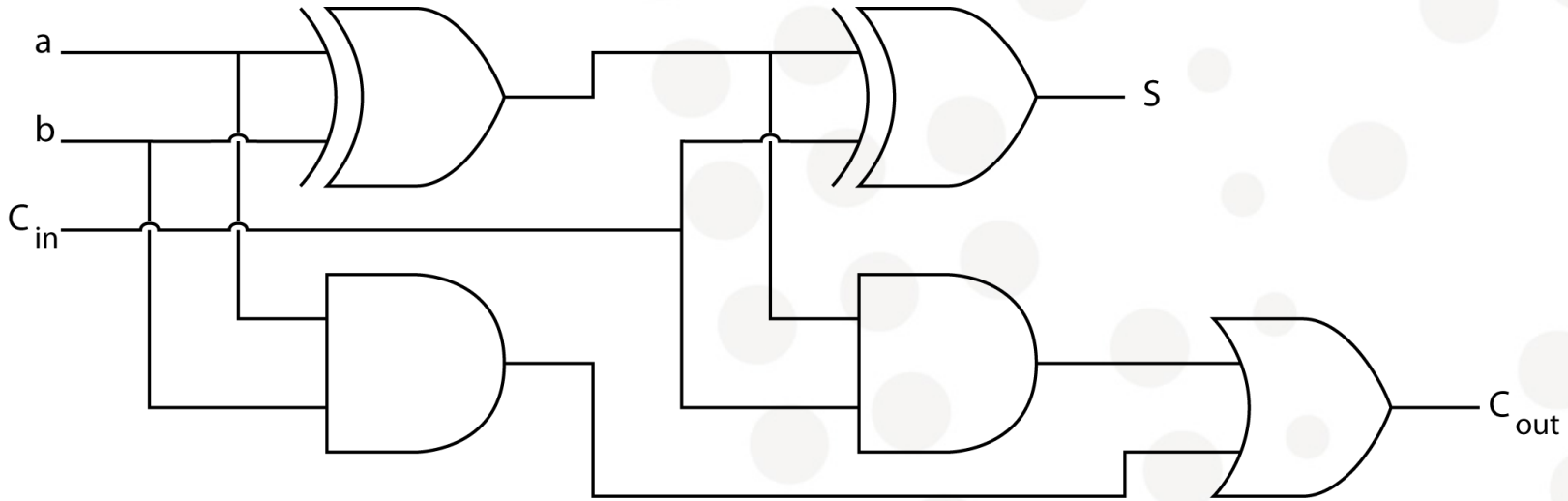
A half adder cannot be cascaded to a binary addition of an arbitrary bit-length since there is no carry input. An extension of the circuit is needed.

Full Adder truth table:

C_{in}	a	b	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adder (2/2)

Schematics:



content

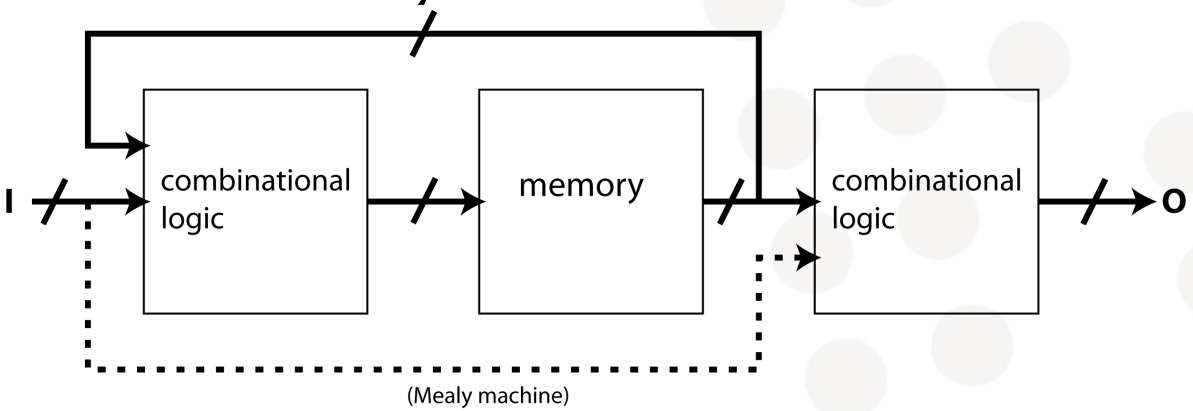
From Scalar to Superscalar

Lecture Summary and Brief Repetition

- Binary numbers
- Boolean Algebra
- Combinational Logic Circuits
  - Encoder/Decoder
  - Multiplexer/Demultiplexer
  - Adders
- Sequential Logic Circuits
  - Counters
  - Shift Registers

Definition

Sequential logic circuits are logic circuits implementing finite state machines, i.e. circuits composed of combinational logic and internal memory elements. One typical categorization of sequential logic circuits are Moore or Mealy machines.

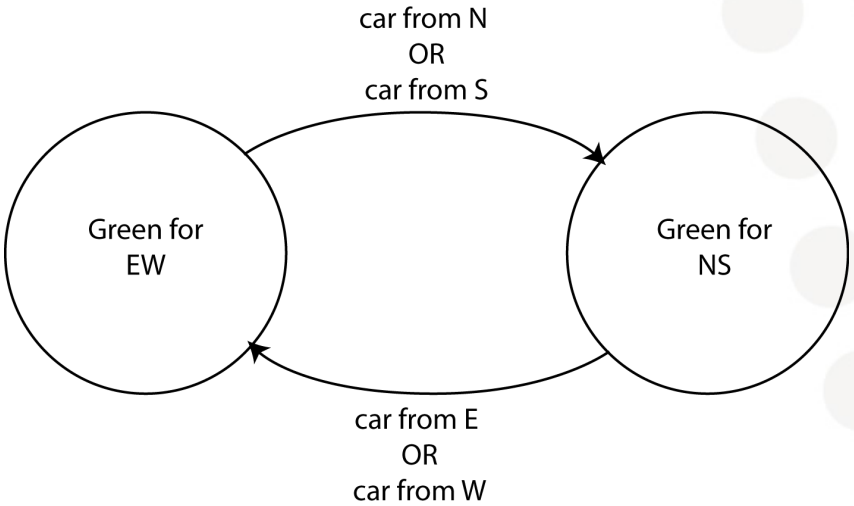


Synchronous and Asynchronous FSM

- *Synchronous* FSMs include an implicit positive transition of a global *clock* signal as transition condition for all state changes. Synchronous FSMs realized as sequential logic circuits use synchronous flip-flops as memory elements, e.g. D-flip-flops. They are generally simpler to implement and easier to verify and test. The clock frequency needs to be slow enough to allow the slowest combinational transition condition to be computed.
- *Asynchronous* FSMs change state at once if the explicit transition condition is met. They can be very fast but are much harder to design and verify.

Example: Synchronous Moore Machine

State transition graph:



Characteristic table:

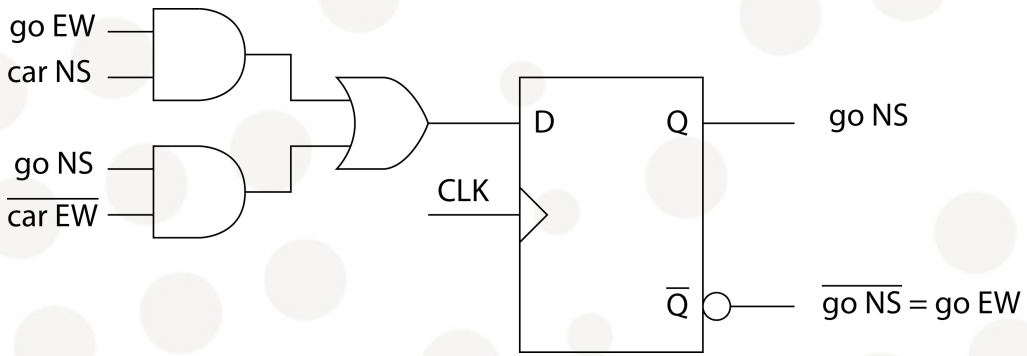
car EW	car NS	go NS	go_{next} NS
0	0	0	0
1	0	0	0
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	0
0	1	1	1
1	1	1	0

Example: Synchronous Moore Machine

Characteristic table:

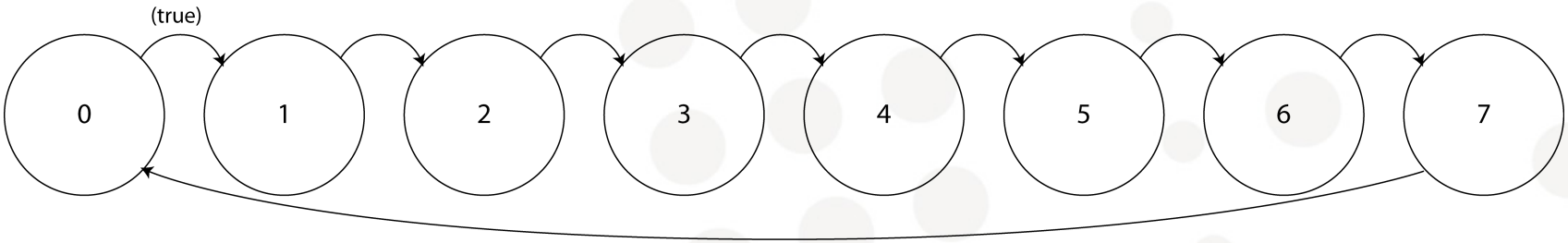
car EW	car NS	go NS	go_{next} NS
0	0	0	0
1	0	0	0
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	0
0	1	1	1
1	1	1	0

Schematics/circuit diagram:



Careful: Always also consider the conditions for a state to be *maintained*, which sometimes is not explicitly stated in the graph!

3-bit Counter State Transition Graph



3-bit Counter Characteristic Table

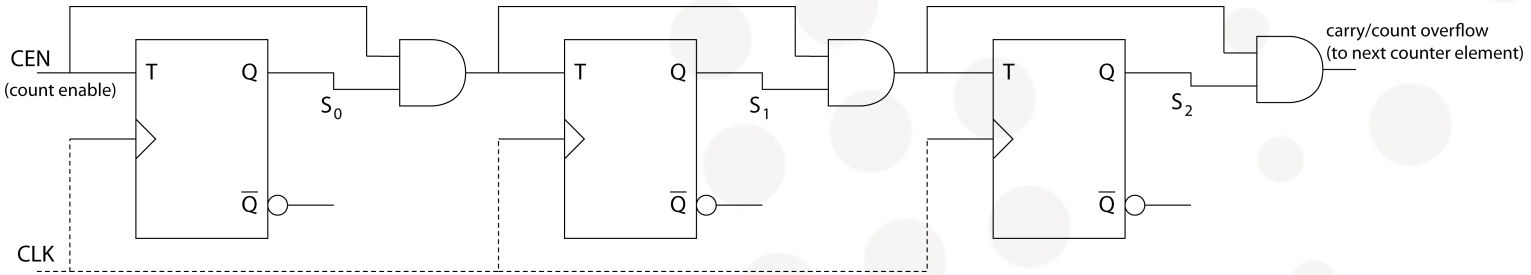
present			in	next		
S_2	S_1	S_0	NA	S_2	S_1	S_0
0	0	0		0	0	1
0	0	1		0	1	0
0	1	0		0	1	1
0	1	1		1	0	0
1	0	0		1	0	1
1	0	1		1	1	0
1	1	0		1	1	1
1	1	1		0	0	0

Counter Element Characteristic Equation

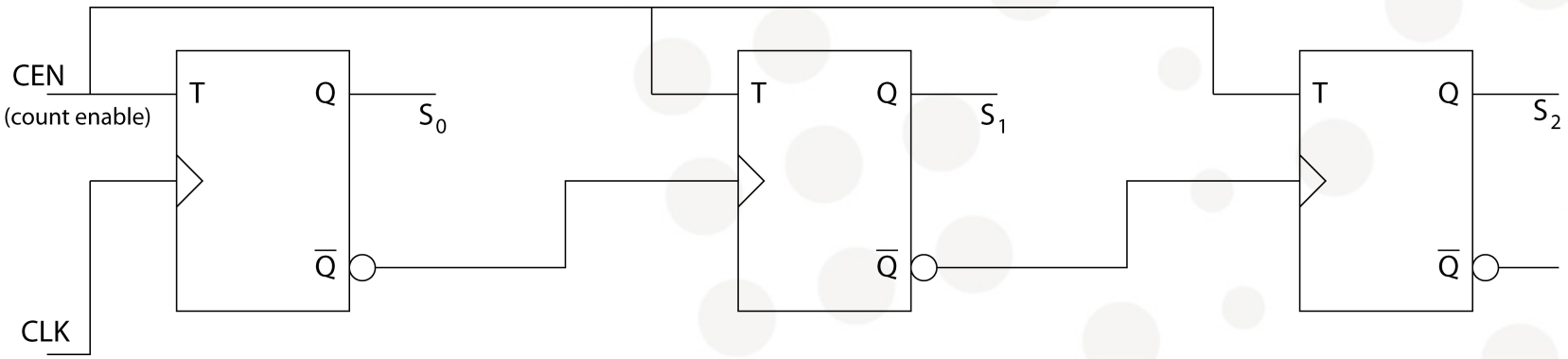
$$S_{n_{next}} = S_n \oplus \left(\bigwedge_{k=0}^{n-1} S_k \right)$$

In words: if all previous bits are 1 → flip/toggle.

3 bit Synchronous Counter



3 bit Ripple Counter



Shift Register State Transition Table

control			next		
LD	SE	LS	O_2	O_1	O_0
1	X	X	I_2	I_1	I_0
0	0	X	O_2	O_1	O_0
0	1	0	RSin	O_2	O_1
0	1	1	O_1	O_0	LSin

Shift Register Schematics