Decimal-to-Two's Complement
1. Convert decimal to binary.
2. Pad to requested bit length (if needed).
3. Invert the bits.
4. Add 1.

NOTE: Follow "all" these steps ONLY IF given a negative decimal number. If positive, simply convert to binary.

Two's Complement-to-Decimal
1. Invert the bits.
2. Add 1.
3. Convert binary to decimal.
4. Add negative sign.

NOTE: Follow "all" these steps ONLY IF a 1 appears is the MSB position. If 0, simply convert to decimal.

• In Two's Complement Addition, always ignore the carry out bit.

Two's Complement Range (N = # of Bits)

$$\left[-2^{N-1}, 2^{N-1}-1\right]$$

Shifting
Multiplication
• 1 << *2
• 2 << *4
• 3 << *8
Division
• 1 >> /2
• 2 >> /4
• 3 >> /8

Max. clock frequency
= 1 / 10ns (critical path time)
= 100 MHz

Overflow
overflow = a3'b3's3 + a3b3s3'
sign bits
overflow overflow no overflow

Overflow
overflow = c3 xor c4
overflow overflow no overflow

Register File
driver
Boosts signal
three-state driver
like no connection
4x32 register file

Problems with Separate Registers
- Complicated wiring
  - Aka routing congestion
  - Burden on the designer
- Weak signals
- Long delays

Note: Each driver in figure actually represents 32 1-bit drivers

Write is synchronous
Read is asynchronous

Multiplier

Subtractor using Adder

Adder & Subtractor

Subtractor
1st column 2nd column 3rd column 4th column
4-bit subtractor

Barrel Shifter
xyz Shifted by 5
Net result: shift by 5:

Up/Down Counter
dir = 0 means up
dir also selects appropriate terminal count value (all 1s or all 0s)
4-bit up/down counter
4-bit register

Strength Reduction
5*C → 4*C + C (4*C same as C<<2)
C / 5 = 0.20 * C
- Approximately equals (102 / 512) * C = 0.199 * C
- (102 / 512) * C = C * (64 + 32 + 4 + 2) / 512
  = (C * 64 + C * 32 + C * 4 + C * 2) / 512
  = ((C<<6) + (C<<5) + (C<<2) + (C<<1)) >> 9

Type	Construction	Features & Drawbacks
D flip-flop	2 D-latches	Stores 1 bit, ~12 transistors per bit
Register	A number of D flip-flops in series	Store one entry of several bits
Register file	A collection of registers with drivers and buses	Store several entries of several bits, fast but large, usually more than 1 port
SRAM	Loop of inverters	Slower than register file but smaller, 6 transistors per cell (bit)
DRAM	Capacitor to store charge	Slower but much small, 1 transistor per cell, need periodic recharge
ROM	Depends	Fast reads, slow writes, retain data without power supply

SRAM cell
To sense amplifiers

DRAM cell
capacitor slowly discharging

- Mask-Programmed ROMs
- Fuse-Based Programmable ROM
- Erasable Programmable ROM (EPROM)
- Electronically-Erasable Programmable ROM (EEPROM)
- Flash Memory

Clock Divider
Example: 1 Hz pulse generator from 30 Hz clock
Divide clock by 30
– Use 5-bit up-counter
  We want tc to pulse every 30 cycles
  Detect 29
– Can also use down counter
  When count reaches 0, load 29

Inputs			Operation	Sample Output if A = 00001111 B = 00000101	AL Logic
x	y	z			
0	0	0	S = A + B	00010100	i_a = a, i_b = b, cin = 0
0	0	1	S = A - B	00001010	i_a = a, i_b = b', cin = 1
0	1	0	S = A + 1	00010000	i_a = a, i_b = 0, cin = 1
0	1	1	S = A	00001111	i_a = a, i_b = 0, cin = 0
1	0	0	S = A AND B (bitwise AND)	00000101	i_a = ab, i_b = 0, cin = 0
1	0	1	S = A OR B (bitwise OR)	00001111	i_a = a+b, i_b = 0, cin = 0
1	1	0	S = A XOR B (bitwise XOR)	00001010	i_a=a xor b, i_b= 0, cin = 0
1	1	1	S = A' (bitwise complement)	11110000	i_a = a', i_b = 0, cin = 0

DIP switches
ALU
LEDs
CALC

Multifunction Arithmetic Logic
• Drawbacks
- Wasted Area and Power
- Complicated Wiring
- Long Delay
• Observation
- Only Need to Perform One Operation At a Time.

Encoding
Encoding method where the state encoding is same as the output values
– Possible if enough outputs and all states have unique output values
Avoid glitching
– No logic between state register flip-flop and output, hence no glitches
No delay (unlike registered outputs)
– Connect output directly to its bit

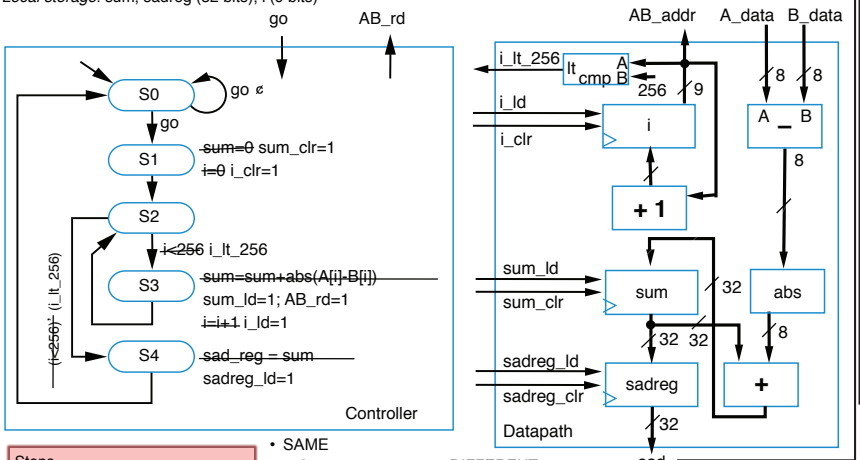
Output Encoding
Use the output values as the state encoding
Inputs: none; Outputs: x,y

One-Hot Encoding
One bit per state
– A bit being '1' corresponds to a particular state
Resultant Boolean equations
– n3 = s2; n2 = s1; n1 = s0; x = s3 + s2 + s1
Fewer gates and only one level of logic
– Less delay than two levels, so higher clock frequency
Inputs: none; Outputs: x

Inputs				Outputs				
s3	s2	s1	s0	n3	n2	n1	n0	x
A	0	0	0	1	0	0	1	0	0
B	0	0	1	0	0	1	0	0	1
C	0	1	0	0	1	0	0	0	1
D	1	0	0	0	0	0	0	1	1

Queue
Initially empty queue
1. After pushing 9, 5, 8, 5, 7, 2, 3
2. After popping
3. After pushing 6
4. After pushing 3
5. After pushing 4
ERROR! Pushing a full queue results in unknown state.

Inputs: A, B [256](8 bits); go (bit)
Outputs: sad (32 bits)
Local storage: sum, sadreg (32 bits); i (9 bits)

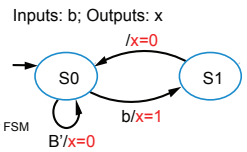


Steps
1. Create datapath
2. Connect datapath to controller
3. Derive controller FSM

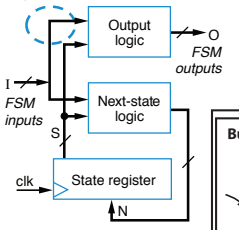
- SAME
  - States
  - Transition arcs
  - Transition conditions
    - If depends on inputs
- DIFFERENT
  - Transition conditions
    - If depends on results from datapath
  - Actions at each state

Mealy Machine

FSM implementation architecture
- State register and logic
- More detailed view
  - Next state logic – function of present state and FSM inputs
  - Output logic
    - If function of present state only – Moore FSM
    - If function of present state and FSM inputs – Mealy FSM



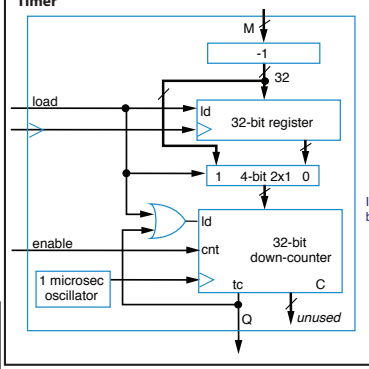
Mealy FSM adds this

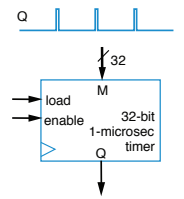


Graphically: show outputs with transitions, not with states

Mealy may have fewer states, but drawback is that its outputs change mid-cycle if input changes
- Output d not 1 for full cycle

Timer

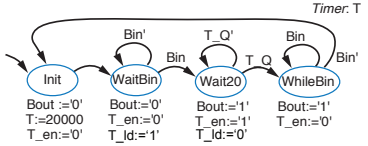




Interval specified as multiple of base time unit
- If base is 1 microsec and user wants pulse every 300 ms, loads 300,000 into timer

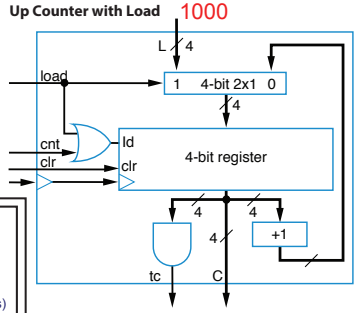
Button Debouncer

Inputs: Bin (bit) *Outputs:* Bout (bit)
Timer: T



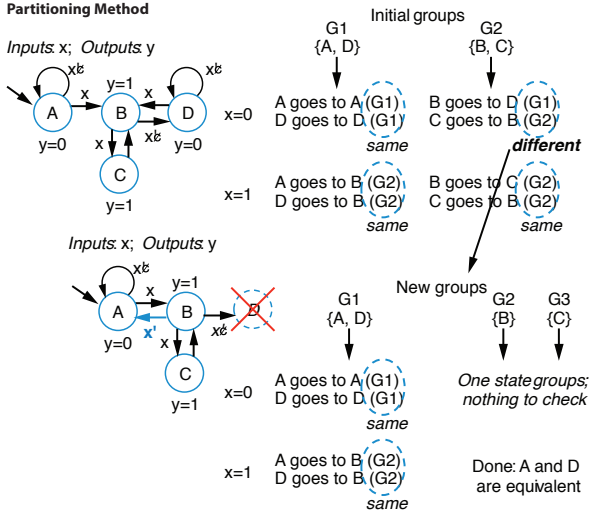
Up Counter with Load

1000

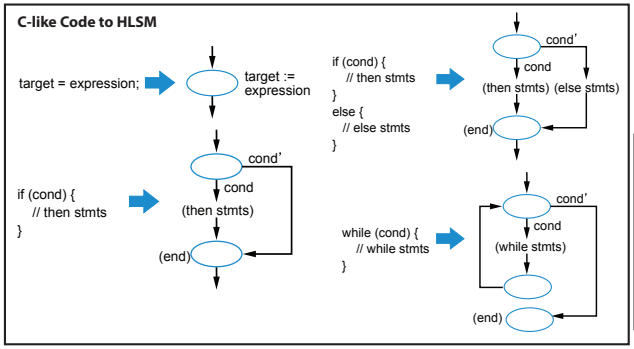


- Useful to create pulses at specific multiples of clock
  - Example: Pulse every 9 clock cycles
  - Use 4-bit down-counter with parallel load
  - Set parallel load input to 8 (1000)
    - Why load 8 and not 9? Because 0 is included in count sequence: 8, 7, 6, 5, 4, 3, 2, 1, 0 → 9 counts
  - Use terminal count to reload
  - When count reaches 0, next cycle loads 8

Partitioning Method

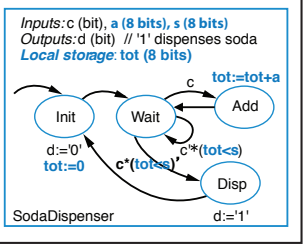


C-like Code to HLSM

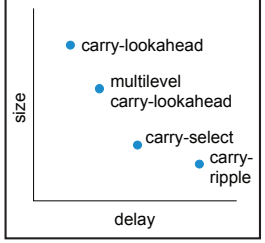


HLSM

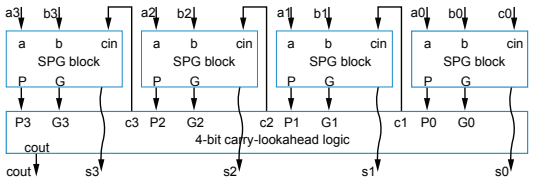
- Numbers
  - Single-bit: '0' (single quotes)
  - Integer: 0 (no quotes)
  - Multi-bit: "0000" (double quotes)
- == for equal, := for assignment
- Multi-bit outputs must be registered via local storage



Adder Comparison



Carry-Lookahead Adder



4-bit adder comparison (gate delays, gates)
- Carry-ripple: (8, 20)
- Two-level: (2, 500)
- CLA: (4, 26)
  - ***Nice compromise***

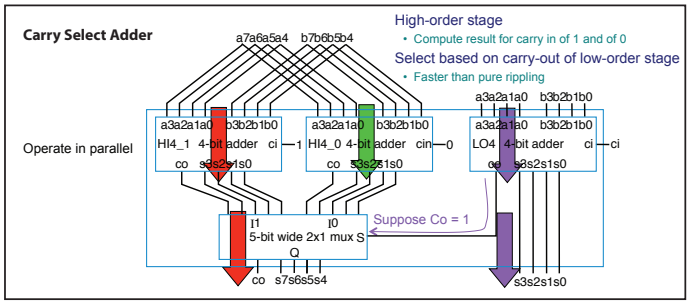
$c1 = G0 + P0c0$

$c2 = G1 + P1G0 + P1P0c0$

$c3 = G2 + P2G1 + P2P1G0 + P2P1P0c0$

$cout = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0c0$

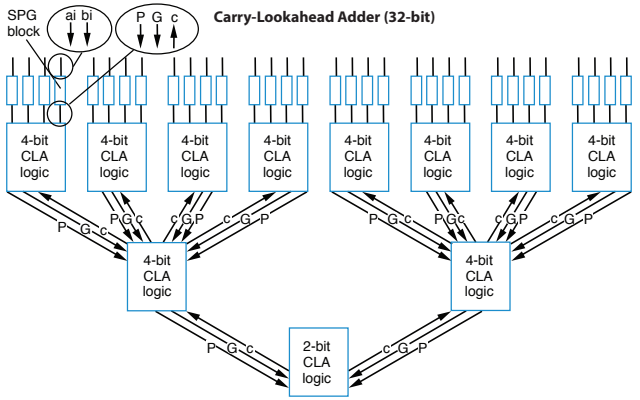
Carry Select Adder



High-order stage
- Compute result for carry in of 1 and of 0

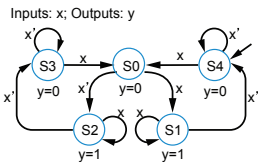
Select based on carry-out of low-order stage
- Faster than pure rippling

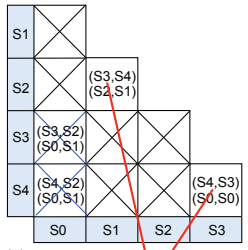
Carry-Lookahead Adder (32-bit)



Implication Table

Inputs: x; Outputs: y





- **Step 1:** Mark state pairs having different outputs as nonequivalent
- **Step 2:** For each unmarked state pair, write the next state pairs for the same input values
- **Step 3:** For each unmarked state pair, mark state pairs having nonequivalent next state pairs as nonequivalent.
  - Repeat this step until no change occurs, or until all states are marked.
- **Step 4:** Merge remaining state pairs

Inputs: x; Outputs: y