Part IA Engineering

Digital Circuits &

Information Processing

Handout 2

Sequential Logic

Richard Prager
Tim Flack
January 2009

Contents of Handout 2

Handout 2 Section A

Binary Numbers

Sequential logic is used to construct counters, and sequence detection circuits. In order to design satisfactory counters we need to understand the binary number system fully. In this section we therefore introduce binary, octal and hexadecimal numbers.

The 2's-complement and sign-magnitude techniques for representing negative numbers are then described.

Binary Numbers

Binary is base 2. Each digit is either 0 or 1.

1	0	1	0	1	0	$= 42_{10}$
32	16	8	4	2	1	Binary
2^5	2^4	2^3	2^2	2^1	2^0	coefficients
MSB					LSB	

0	0	1	0	1	1	$= 11_{10}$
32	16	8	4	2	1	Binary
2^5	2^4	2^3	2^2	2^1	2^0	coefficients
MSB					LSB	

A binary digit is called a bit.
In computers, binary numbers are often 8 bits long.
An 8 bit binary storage location is called a byte.
A byte can store numbers from 0 to $(2^8 - 1) = 255$.

MSB Most significant bit.
LSB Least significant bit.

Decimal to Binary Conversion

Use the remainders from successive division by 2.

(a) Convert 42_{10} into binary:

$$\begin{aligned} 42/2 &= 21 \quad \text{remainder } 0 \\ 21/2 &= 10 \quad \text{remainder } 1 \\ 10/2 &= 5 \quad \text{remainder } 0 \\ 5/2 &= 2 \quad \text{remainder } 1 \\ 2/2 &= 1 \quad \text{remainder } 0 \\ 1/2 &= 0 \quad \text{remainder } 1 \end{aligned}$$

So the answer is 101010_2 (reading upwards).

(b) Convert 11_{10} into binary:

$$\begin{aligned} 11/2 &= 5 \quad \text{remainder } 1 \\ 5/2 &= 2 \quad \text{remainder } 1 \\ 2/2 &= 1 \quad \text{remainder } 0 \\ 1/2 &= 0 \quad \text{remainder } 1 \end{aligned}$$

So the answer is 1011_2 (reading upwards).

Octal: Base 8

Just as base 2 only uses two digits (0 & 1), base 8 only uses 8 digits: 0, 1, 2, 3, 4, 5, 6 and 7.

0	5	2_8	$= 42_{10}$
64	8	1	Octal
8^2	8^1	8^0	coefficients
MSB		LSB	

To convert from decimal to base 8 either use successive division by 8:

$$\begin{aligned} 42/8 &= 5 \quad \text{remainder } 2 \\ 5/8 &= 0 \quad \text{remainder } 5 \end{aligned}$$

So the answer is that $42_{10} = 52_8$ (reading upwards).

Or alternatively, convert to binary, divide the binary number into three-bit groups and work out the octal digit to represent each group. We have shown that $42_{10} = 101010_2$ so:

5			2_8			
1	0	1	0	1	0	$= 42_{10}$

Hexadecimal: Base 16

For base 16 we need 16 different digits. We therefore need new symbols for the digits to represent 10–15.

$1010_2 = 10_{10} = A_{16}$ $\quad$ $1101_2 = 13_{10} = D_{16}$
$1011_2 = 11_{10} = B_{16}$ $\quad$ $1110_2 = 14_{10} = E_{16}$
$1100_2 = 12_{10} = C_{16}$ $\quad$ $1111_2 = 15_{10} = F_{16}$

0	2	A_{16}	$= 42_{10}$
256	16	1	Hexadecimal
16^2	16^1	16^0	coefficients
MSB		LSB	

Labelling Hex Numbers

Hex numbers are indicated 59_{16} or 59_H when they occur in text. In assembler programming the convention \$59 is often used. Two hexadecimal digits are often used as a convenient way to specify the contents of a byte.

Decimal to Hex Conversion

To convert from decimal to hex either use successive division by 16:

$42/16 = 2$ remainder A
$2/16 = 0$ remainder 2

So the answer is $2A_{16}$ (reading upwards).

Or alternatively, convert to binary, divide the binary number into four-bit groups and work out the hex digit to represent each group. We have shown that $42_{10} = 101010_2$ so:

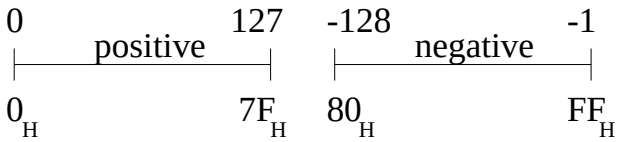
2				A_{16}				
0	0	1	0	1	0	1	0	$= 42_{10}$

You can also do the conversion automatically on your calculator.

Negative Numbers

So far we can only represent positive numbers. In an 8-bit byte we can represent the decimal numbers from 0 to 255 ($255 = 2^8 - 1 = \mathrm{FF}_\mathrm{H}$).

Whatever we do, there are still only 256 different combinations of bits in a byte so if we want to represent negative numbers we have to give up some of the range of positive numbers we had before. We choose the following strategy which is called 2's complement.



All negative numbers have the MSB set.

Changing Sign

The rule for changing a positive 2's complement number into a negative 2's complement number (or vice versa) is:
Invert all the bits and add 1.

What happens when we do this to an 8 bit binary number x.

- *Invert all the bits*: $x \rightarrow (255 - x)$
- *Add one*: $(255 - x) \rightarrow (256 - x)$

But 256 $(= 100_\mathrm{H})$ will not fit into an 8 bit byte. It equals 00_H and causes the carry flag to be set. If we choose to ignore the carry flag then $256 - x$ will behave just like $00 - x$.

We can therefore use **normal** binary arithmetic to manipulate the 2's complement of x and it will behave just like $-x$.

2's Complement Addition

	0	0	0	0	0	1	1	1	7
	0	0	0	0	0	1	0	0	+4
(0)	0	0	0	0	1	0	1	1	11

To subtract, negate the second argument, then add.

	0	0	0	0	0	1	1	1	7
	1	1	1	1	1	0	0	1	+−7
(1)	0	0	0	0	0	0	0	0	0

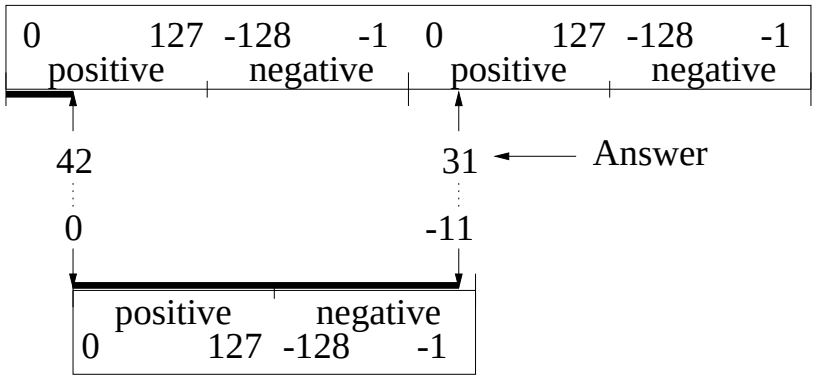
	0	0	0	0	1	0	0	1	9
	1	1	1	1	1	0	0	1	+−7
(1)	0	0	0	0	0	0	1	0	2

	0	0	0	0	0	1	0	0	4
	1	1	1	1	1	0	0	1	+−7
(0)	1	1	1	1	1	1	0	1	−3

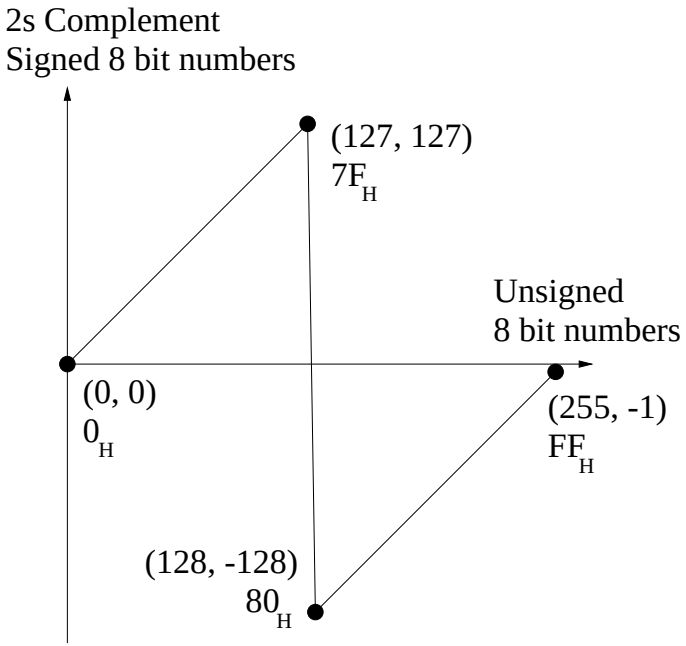
	1	1	1	1	1	0	0	1	−7
	1	1	1	1	1	0	0	1	+−7
(1)	1	1	1	1	0	0	1	0	−14

2's Complement Addition

$2A_H$	42
$F5_H$	+−11
$(1)1F_H$	31



Signed vs. Unsigned 8 Bit Numbers



2s Complement Overflow

When working with *unsigned* numbers we use the carry from the 8th bit to show when the number has got too big.

With *signed* numbers we are deliberately ignoring the carry flag so we need a new rule to detect when the answer is out of range.

The rule is: 2s complement overflow occurs when

the carry into the MSB

$\neq$

the carry out from the MSB.

		0	0	0	0	1	1	1	1	15
		0	0	0	0	1	1	1	1	+15
OK	(0)	0	0	0	1	1	1	1	0	30

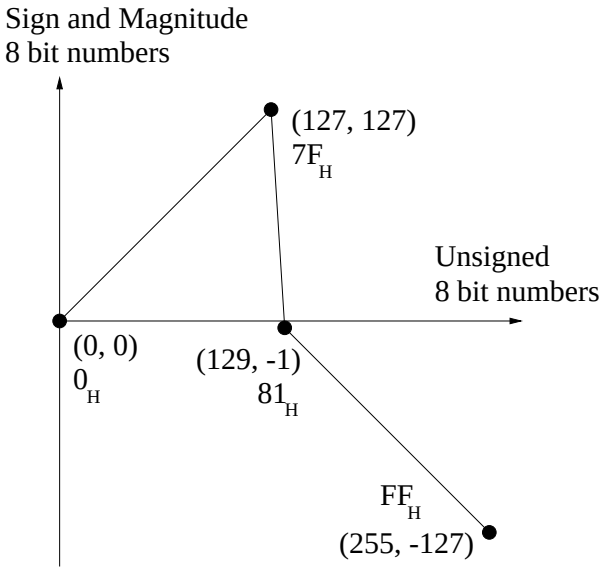
		0	1	1	1	1	1	1	1	127
		0	0	0	0	0	0	0	1	+1
overflow	(0)	1	0	0	0	0	0	0	0	−128

		1	1	1	1	0	0	0	1	−15
		1	1	1	1	0	0	0	1	+ − 15
OK	(1)	1	1	1	0	0	0	1	0	−30

		1	0	0	0	0	0	0	1	−127
		1	1	1	1	1	1	1	0	+ − 2
overflow	(1)	0	1	1	1	1	1	1	1	127

Sign and Magnitude Representation

MSB indicates the sign: ($0 \Rightarrow +$, $1 \Rightarrow -$). Remaining 7 bits contain the magnitude.



Not often used. Separate circuits are required for adding positive and negative numbers. Note that $+0$ has a different representation to -0.

Binary Coded Decimal

Each decimal digit of a number is coded as a four bit binary quantity.

BCD is used in some business computers and cash registers. It is not an efficient way of storing the numbers, but is easy to code and decode.

$1248_{10} = 0001\ 0010\ 0100\ 1000_{BCD}$

$1234_{10} = 0001\ 0010\ 0011\ 0100_{BCD}$

This is 8421 BCD because the bits have these weightings. As we don't need numbers above 9 another alternative is 2421 BCD, where the top bit only has a weight of 2. Another variant, called 'Excess-3 code' is to record each digit value plus 3 using the 8421 weights.

Alphanumeric Character Codes

ASCII: 

American Standard Code for Information Interchange.

In the standard version this is a 7 bit code with the remaining bit being set to zero or used for parity. The first 32 numbers are 'control codes' originally used for controlling modems. The rest are upper case letters, lower case letters, numbers and punctuation.

An extended version of ASCII uses all 8 bits to provide 128 additional graphics characters. These vary from computer to computer.

Other systems: EBCDIC — an IBM system, not much used. Unicode — a 16 bit system, to include Chinese characters etc.

Parity

Parity is the simplest sort of error detection coding used for transferring binary data through an imperfect data channel.

In the transmission circuitry, it involves setting a particular bit in each binary number so as to ensure that an even number of bits are always on.

In the receiver circuitry we count the number of bits that are on in each number and reject any that have an odd number.

Parity can only detect 1-bit errors. More serious errors may go undetected.

Handout 2 Section B

Bistables

In this section we describe the construction of circuits which have two internal states. They are called bistables. We gradually build up more sophisticated bistables from simple ones.

Bistables circuits are the building blocks of computer memory chips, sequencers and counters.

We start by describing the set-reset (SR) bistable.

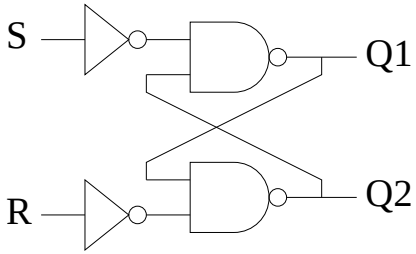
A gated-SR bistable is built using an SR bistable.

A master-slave bistable is built using two gated-SR bistables.

A D-type latch is built from a master-slave bistable.

A JK bistable is built from a master-slave bistable.

Set Reset Bistable



$$\begin{aligned} Q1 &= \overline{\overline{S}.Q2} = S + \overline{Q2} \\ Q2 &= \overline{\overline{R}.Q1} = R + \overline{Q1} \end{aligned}$$

Assume initially that $S = R = 1$ is not allowed.

- If S turns on then $Q1$ turns on.
- $Q2$ will be off as $Q1$ is on and R is off.
- Therefore, even if S turns off, $Q1$ will stay on. (As long as R does not turn on).

- If R turns on then $Q2$ turns on.
- $Q1$ will be off as $Q2$ is on and S is off.
- Therefore, even if R turns off, $Q2$ will stay on. (As long as S does not turn on).

So:

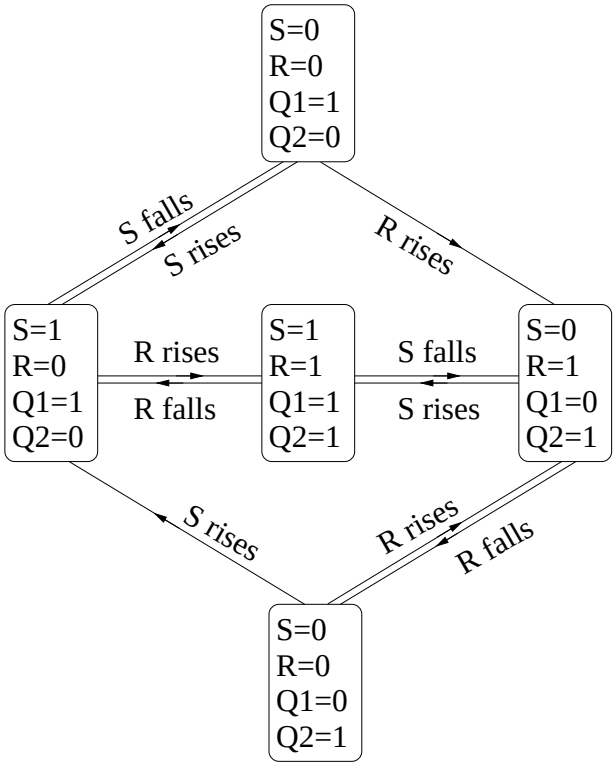
S turns $Q1$ on and $Q2$ off. They then stay in this state until R turns on.

R turns $Q2$ on and $Q1$ off. They then stay in this state until S turns on.

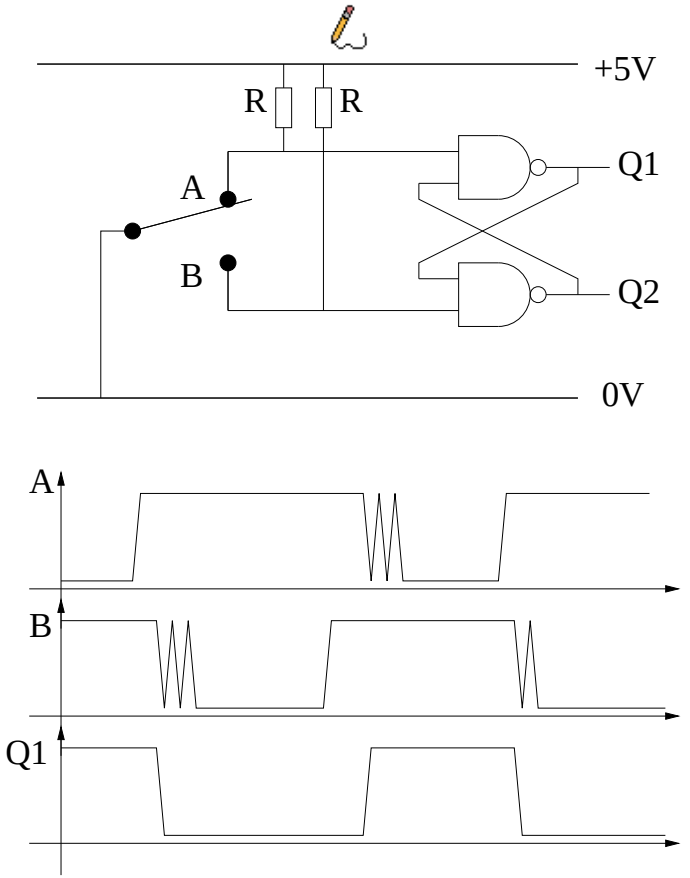
We therefore have memory. The circuit state depends not only on the current values of S and R but also on their values in the past.

State Diagram of SR Bistable

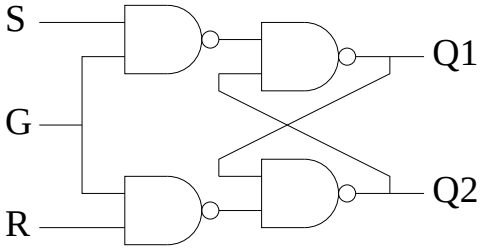
The complete operation of the bistable, including what happens when $S = R = 1$, can be described using a state diagram.

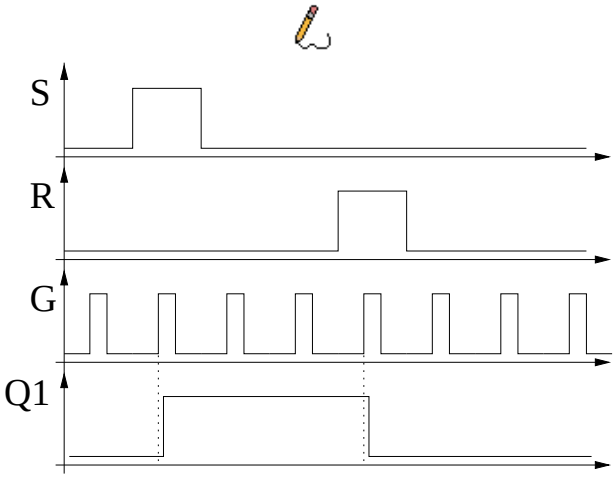


Contact Debouncing



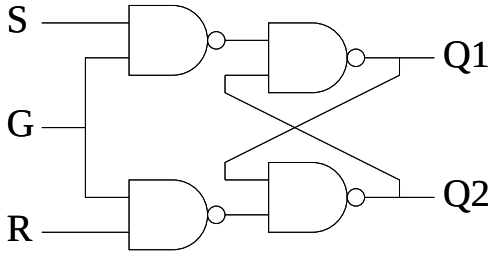
Gated SR Bistable

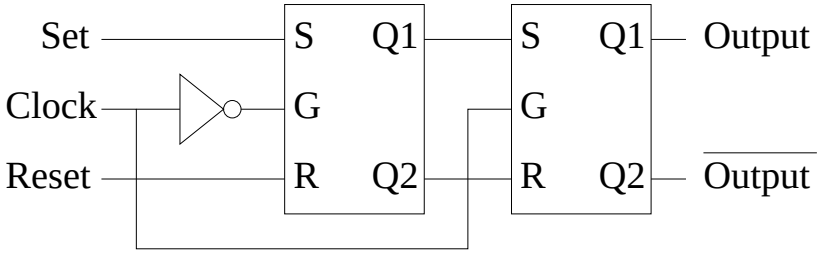




The bistable can change state whenever the gate input is high.

Master-Slave Bistable





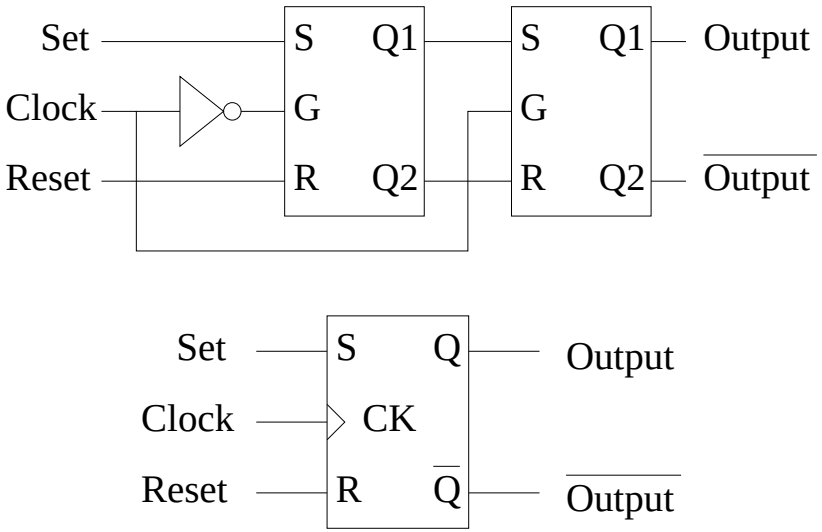
When clock is low

first bistable is sensitive to input signal.

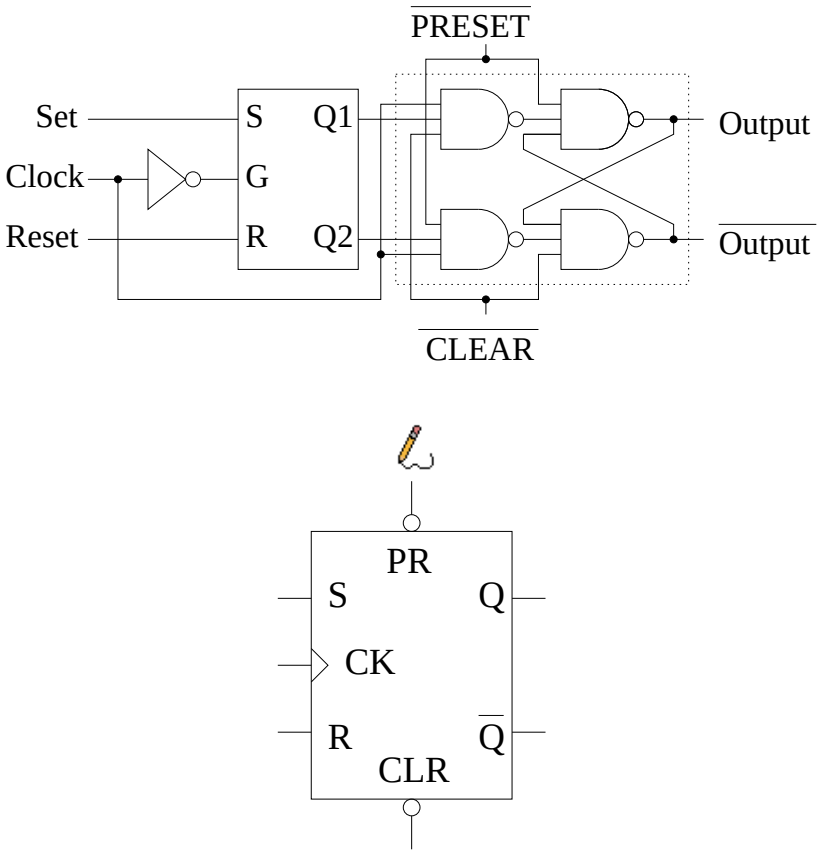
When clock rises

state of first bistable is transfered to second bistable.

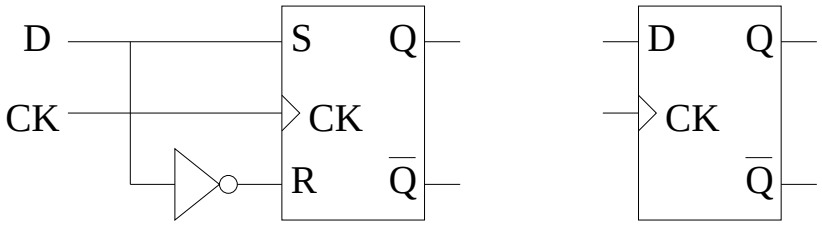
Master Slave Symbol



Asynchronous Inputs

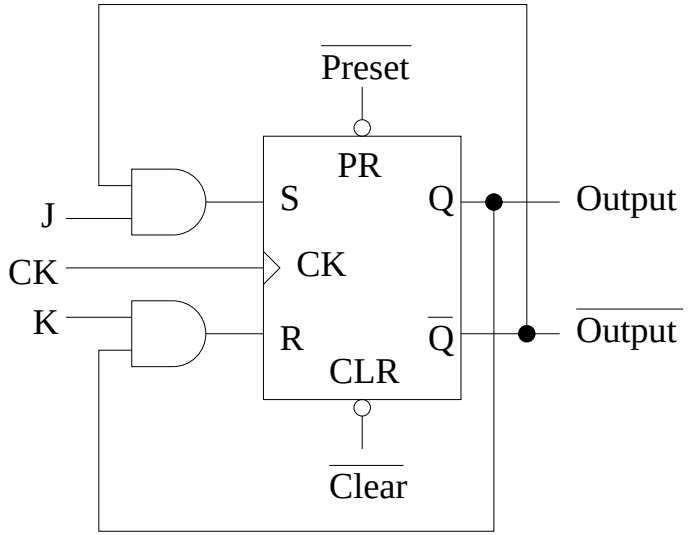


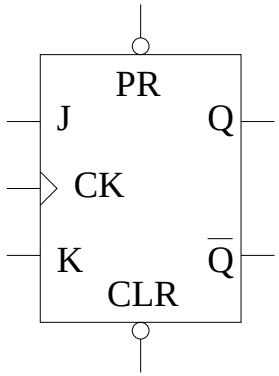
D-type Latch



A D-type (delay) latch has only one data input: D.

JK Bistable





Building a JK Bistable

- Start with simple SR bistable.
- Add gate input: bistable can only change when gate is high.
- Put two SR bistables together: create master-slave bistable. Output only changes when clock input rises.
- Add asynchronous inputs: preset and clear.
- Add AND gates to prevent $S = R = 1$ at input to first SR bistable of master-slave pair. Create JK bistable.

Operation of JK Bistable

In the Master-Slave bistable the input state $S = R = 1$ is prohibited. In the JK bistable this state is allowed and performs a new and useful function: it toggles the bistable (i.e. inverts its previous state).

Inputs when clock low		Output after clock rises
Input J	Input K	$Q(n+1)$
0	0	$Q(n)$ (unchanged)
0	1	0
1	0	1
1	1	$\overline{Q(n)}$ (toggles)

Characteristic Table

The characteristic (or excitation) table of the JK bistable defines what inputs are required to achieve a particular output change.

Output before $\rightarrow$ after		Required inputs	
$Q(n)$	$Q(n+1)$	Input J	Input K
0	0	0	$\times$
0	1	1	$\times$
1	0	$\times$	1
1	1	$\times$	0

$\times \Rightarrow$ "don't care"

If at any time when the clock is low a signal exists at the input encouraging the bistable to change state, that change will take place as the clock rises.

Handout 2 Section C

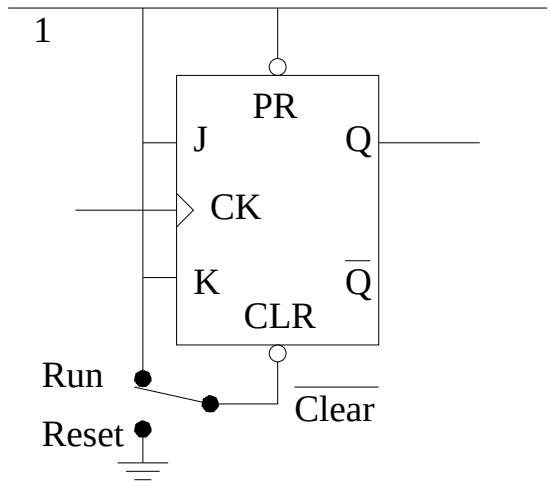
Applications of Bistables

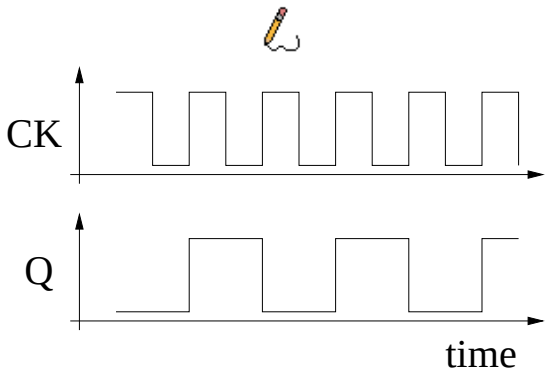
In this section we introduce a number of circuits using bistables. They fall into two groups.

- Counter circuits: divide by n counters, ripple counter, synchronous counter, Johnson counter.
- Memory circuits: shift register, parallel loading shift register.

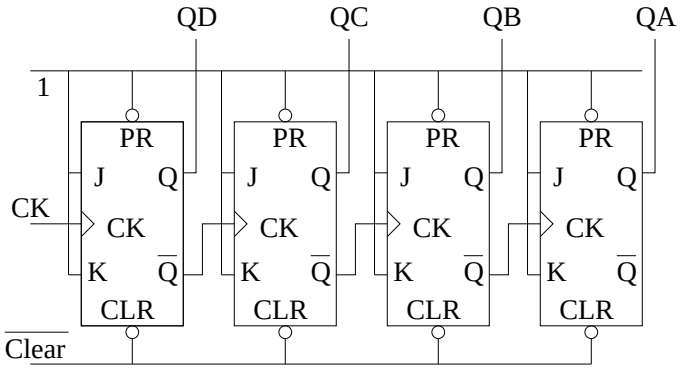
Counters and sequencers are used in the design of control logic inside microprocessors.

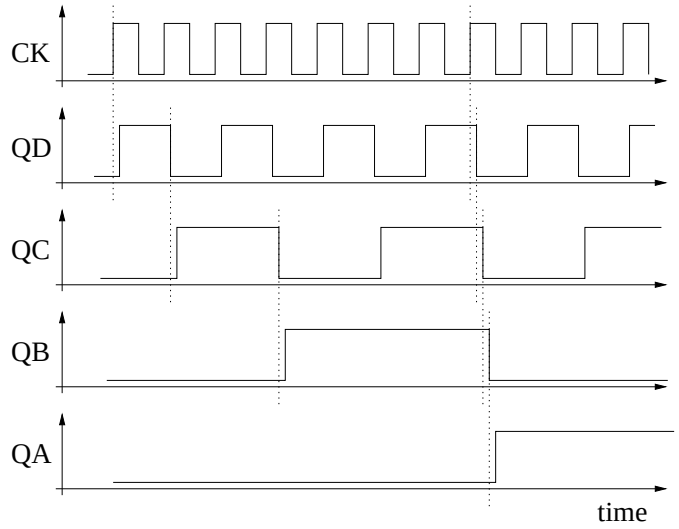
Divide by 2 Counter





Ripple Counter

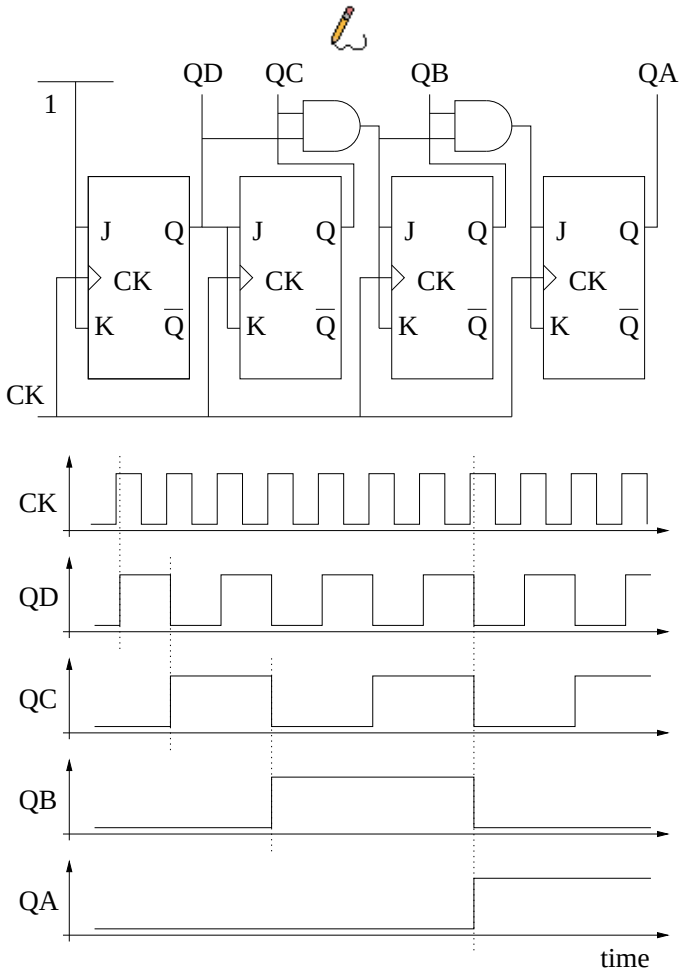




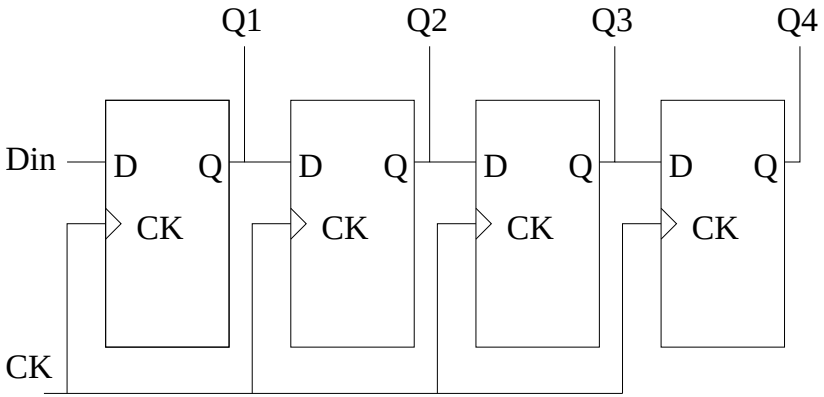
Ripple Counter States

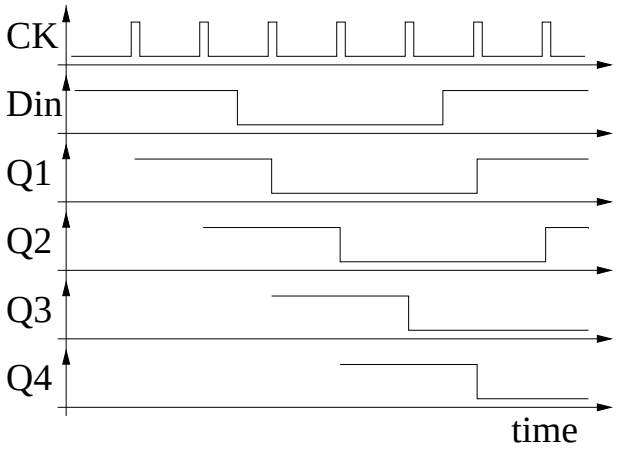
Correct States				Transitory States			
QA	QB	QC	QD	QA	QB	QC	QD
0	0	0	0				
0	0	0	1				
				0	0	0	0
0	0	1	0				
0	0	1	1				
				0	0	1	0
				0	0	0	0
0	1	0	0				
0	1	0	1				
				0	1	0	0
0	1	1	0				
0	1	1	1				
				0	1	1	0
				0	1	0	0
				0	0	0	0
1	0	0	0				

Synchronous Counter

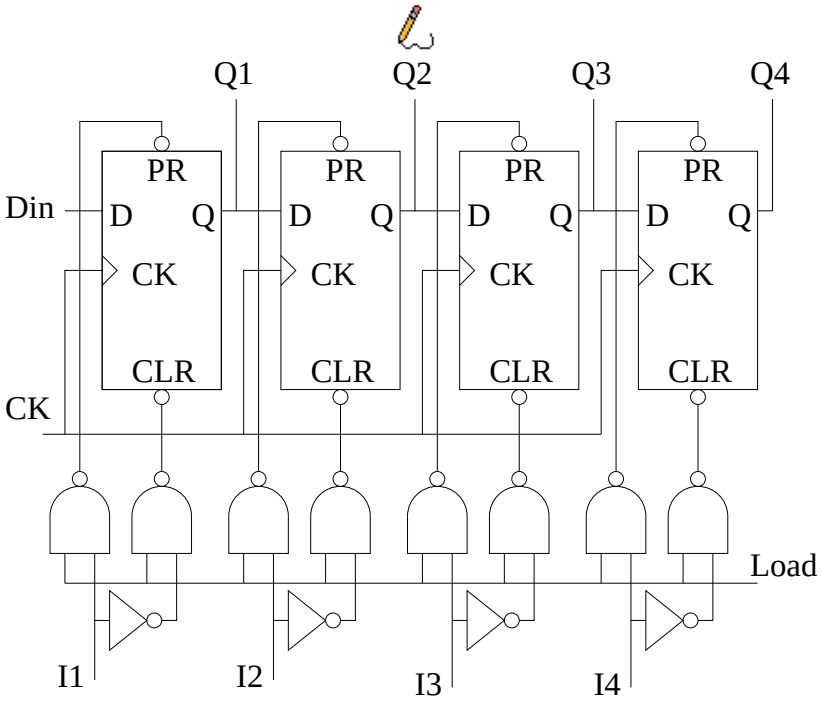


Shift Register





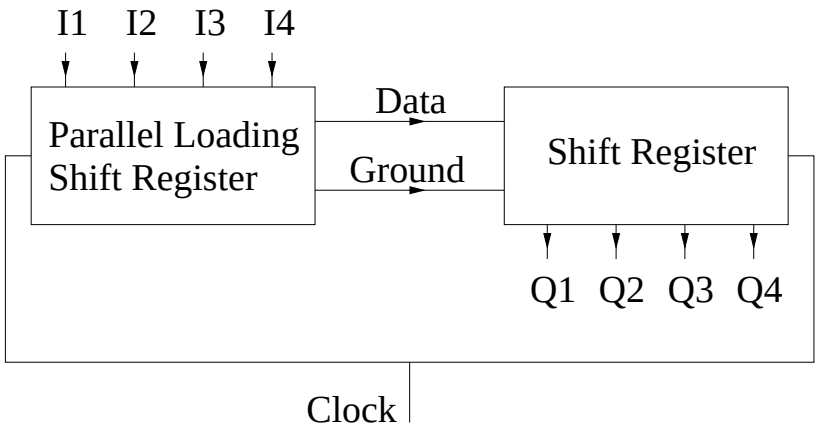
Parallel Loading



Serial Data Link

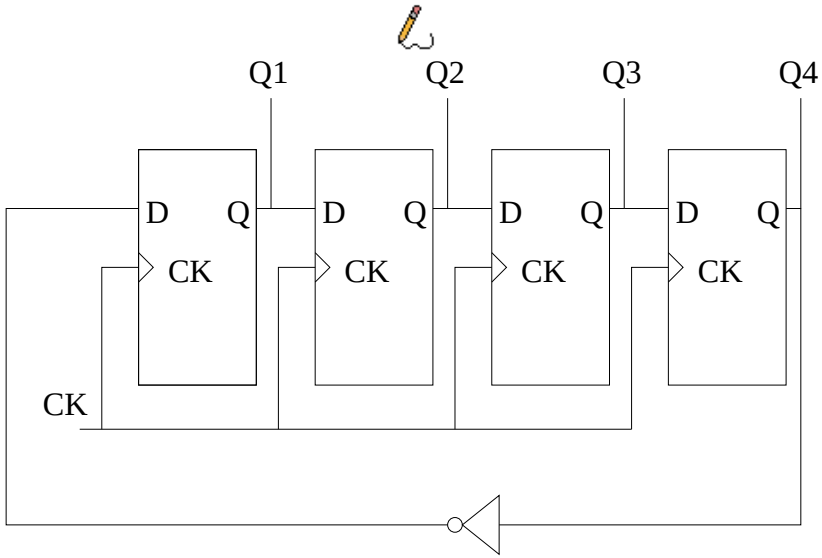
Application of shift registers.

A parallel loading shift register is used to send one data bit at a time across the serial link. Benefit: less wires are needed for the link than would be required for a fully parallel link.

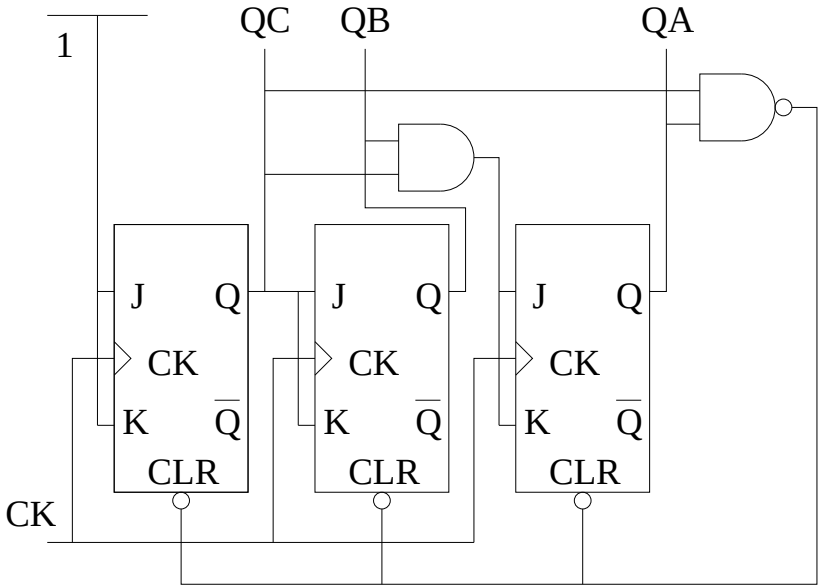


Johnson Counter

A twisted ring Johnson counter uses a shift register of length N to produce a sequence of length $2N$.



Divide by Five

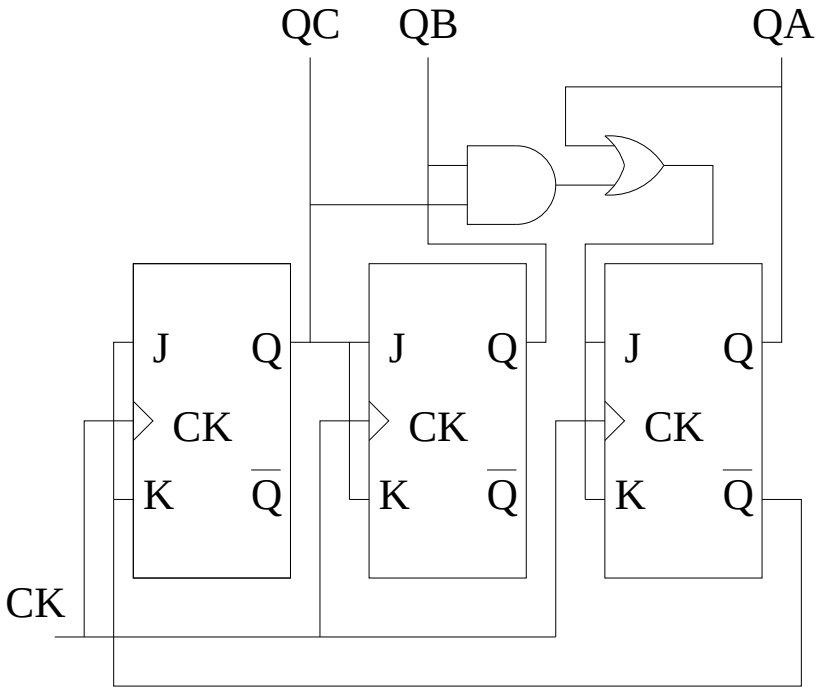


Asynchronous clear can be used to produce divide by N counters for N not equal to a power of 2.

Problem: glitch as 101 appears briefly before the counter clears to zero.

Divide by Five

Synchronous design.



Handout 2 Section D

Synchronous Logic Design

In this section we describe how to design synchronous sequential logic circuits.

There are 5 stages: state diagram, bistable allocation, state transition table, Karnaugh maps, and finally the circuit diagram.

The technique is introduced using the design of a divide-by-three counter. Further examples based on a sequence generator for a boiler ignition system and a sequence detector are then described. Sequence detectors are used in the construction of digital communication systems.

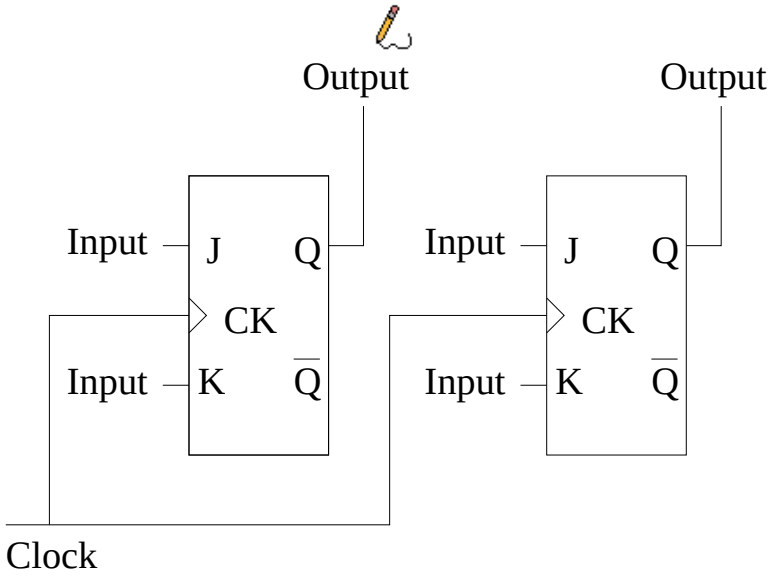
Synchronous Design

Generate sequences. For example: traffic lights.

Detect sequences. For example: electronic combination lock inside burglar alarm.

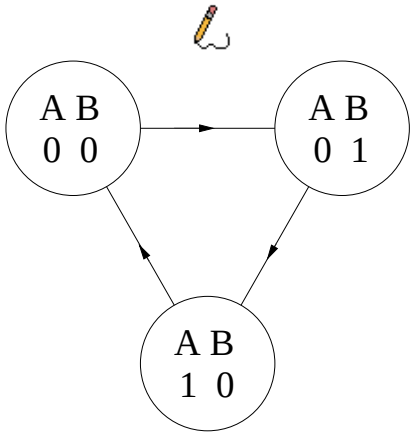
- State diagram.
- Bistable allocation.
- State transition table.
- Karnaugh maps for each input.
- Full circuit.

Outline Synchronous Circuit



Divide by 3 counter

State diagram with bistable allocation:



State transition table:

Current state		Next state		Required inputs			
Q_A	Q_B	Q_A	Q_B	J_A	K_A	J_B	K_B
0	0	0	1	0	×	1	×
0	1	1	0	1	×	×	1
1	0	0	0	×	1	0	×
1	1	0	0	×	1	×	1

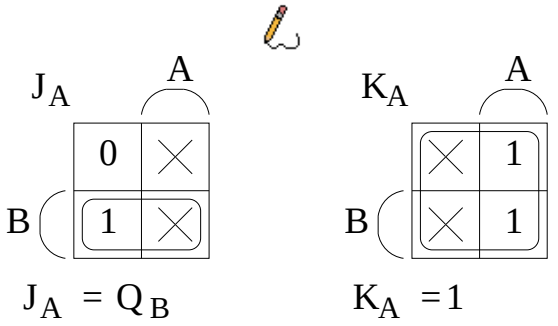
JK Reminder

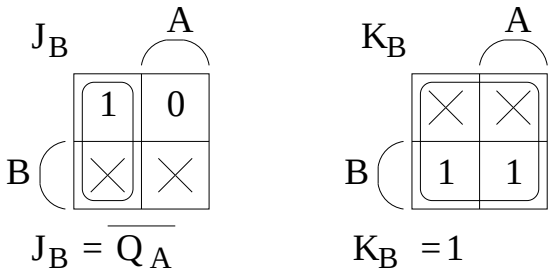
If at any time when the clock is low a signal exists at the input encouraging the bistable to change state, that change will happen as the clock rises.

J Permission to turn Q on.

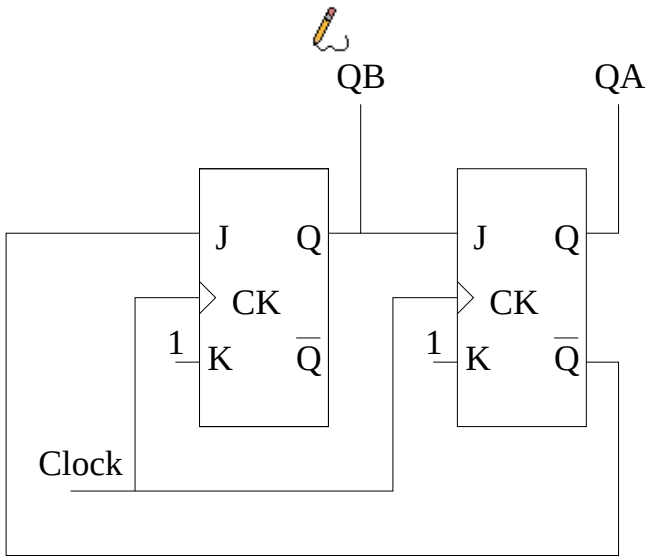
K Permission to turn Q off.

Karnaugh Maps





Circuit



Sequence Generator with Inputs

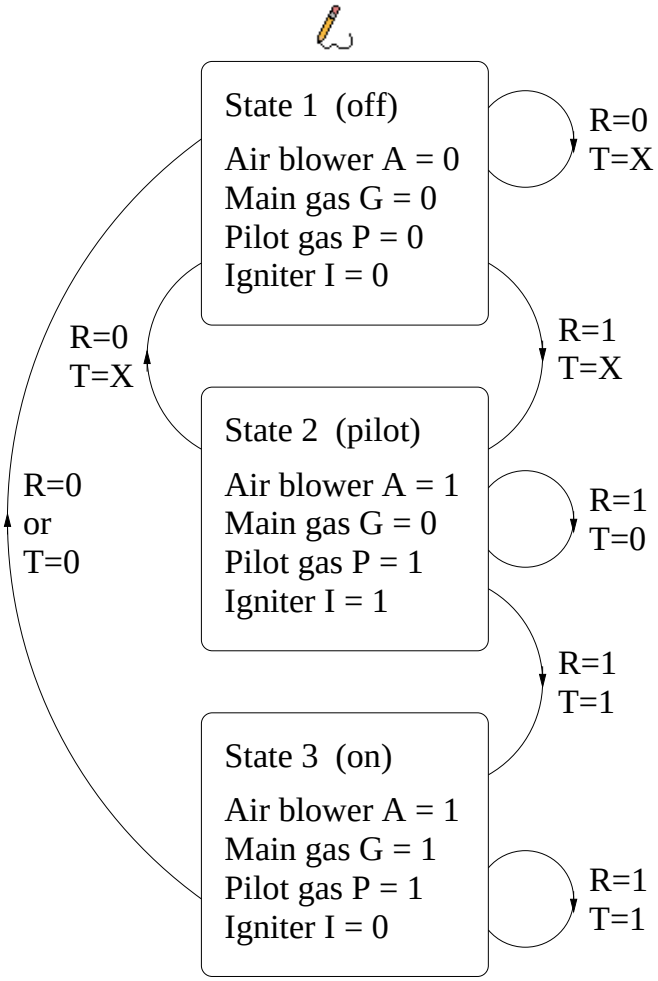
A boiler control unit has a ‘run’ input R.

When R goes to 1, turn on the air blower ($A = 1$) the pilot light gas supply ($P = 1$) and the igniter ($I = 1$).

When the signal from the pilot light thermocouple indicates the pilot light is on ($T = 1$), turn off the igniter and turn on the main gas supply ($G = 1$).

If R goes to 0 or T goes to 0, turn everything off.

State Diagram



Check the State Diagram

You must make sure that there is a path leaving each state for every combination of possible inputs.

In this case we have two inputs R and T. There are therefore 4 possible combinations. ($\overline{R}.\overline{T}$, $R.\overline{T}$, $\overline{R}.T$, and $R.T$)

We need to make sure that there are 4 paths leaving every state.

Note that "$R = 1$, $T = \times$" counts for two paths ($R.\overline{T}$ and $R.T$) also "$R = 0$ or $T = 0$" counts for three paths ($\overline{R}.\overline{T}$, $R.\overline{T}$ and $\overline{R}.T$)

Bistable Allocation

		Bistable Outputs Q_A	Q_B
State 1	off	0	0
State 2	pilot	0	1
State 3	on	1	0
	UNUSED	1	1

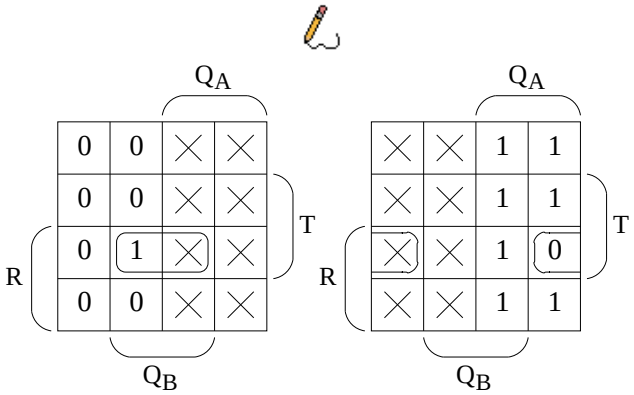
State Transition Table

Inputs		Current state		Next state		Required bistable inputs			
R	T	Q_A	Q_B	Q_A	Q_B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	0	×	0	×
0	1	0	0	0	0	0	×	0	×
1	0	0	0	0	1	0	×	1	×
1	1	0	0	0	1	0	×	1	×
0	0	0	1	0	0	0	×	×	1
0	1	0	1	0	0	0	×	×	1
1	0	0	1	0	1	0	×	×	0
1	1	0	1	1	0	1	×	×	1
0	0	1	0	0	0	×	1	0	×
0	1	1	0	0	0	×	1	0	×
1	0	1	0	0	0	×	1	0	×
1	1	1	0	1	0	×	0	0	×
0	0	1	1	0	0	×	1	×	1
0	1	1	1	0	0	×	1	×	1
1	0	1	1	0	0	×	1	×	1
1	1	1	1	0	0	×	1	×	1

Concise State Transition Table

Inputs		Current state		Next state		Required bistable inputs			
R	T	Q_A	Q_B	Q_A	Q_B	J_A	K_A	J_B	K_B
0	×	0	0	0	0	0	×	0	×
1	×	0	0	0	1	0	×	1	×
0	×	0	1	0	0	0	×	×	1
1	0	0	1	0	1	0	×	×	0
1	1	0	1	1	0	1	×	×	1
0	×	1	0	0	0	×	1	0	×
1	0	1	0	0	0	×	1	0	×
1	1	1	0	1	0	×	0	0	×
×	×	1	1	0	0	×	1	×	1

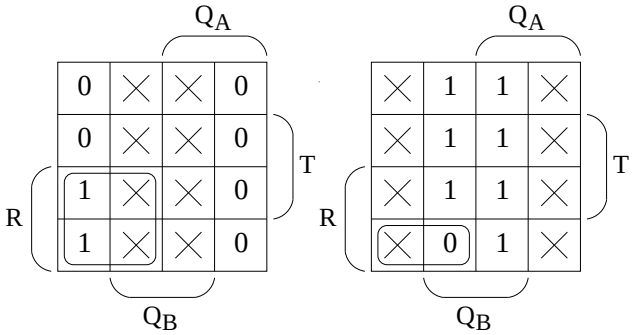
Karnaugh Maps



$J_A = R \cdot T \cdot Q_B$

$\overline{K_A} = R \cdot T \cdot \overline{Q_B}$

$K_A = \overline{R \cdot T \cdot \overline{Q_B}}$

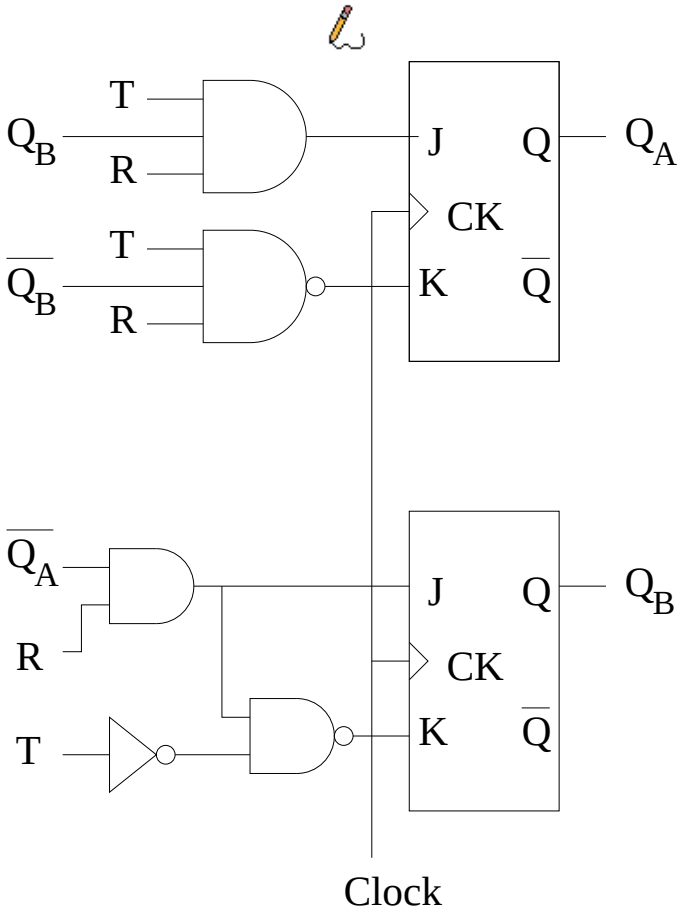


$J_B = R \cdot \overline{Q_A}$

$\overline{K_B} = R \cdot \overline{T} \cdot \overline{Q_A}$

$K_B = \overline{\overline{T} \cdot J_B}$

Partial Circuit



Output Logic

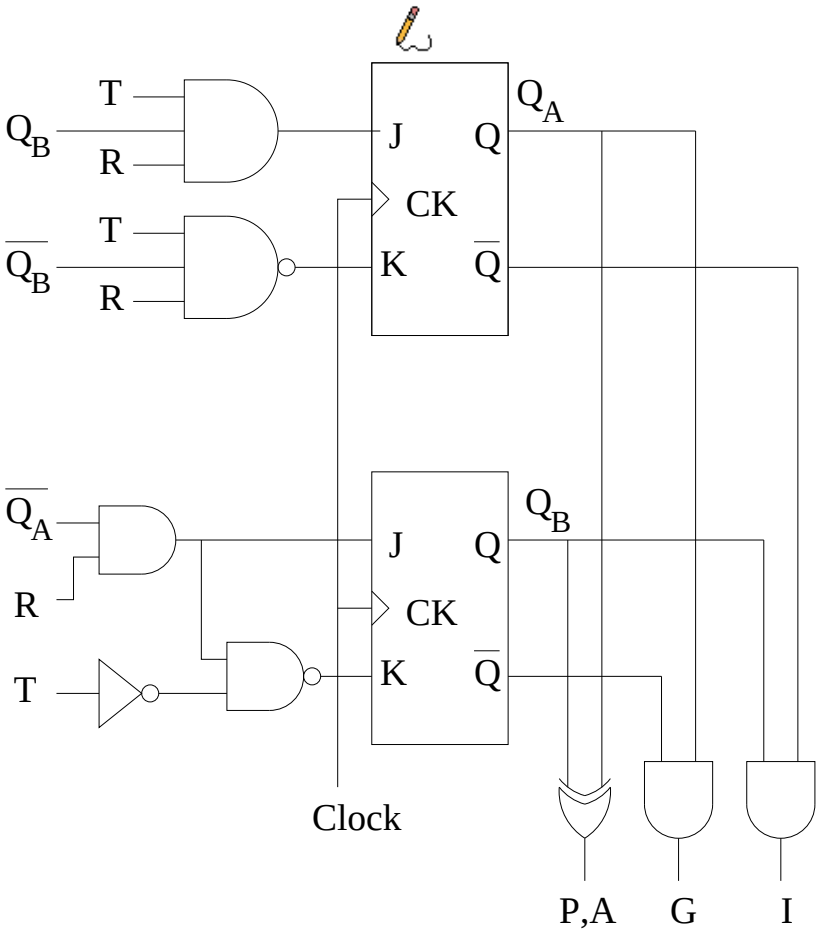
From the state diagram we have:

		Bistable Outputs		Control Signals			
		Q_A	Q_B	P	A	I	G
State 1	off	0	0	0	0	0	0
State 2	pilot	0	1	1	1	1	0
State 3	on	1	0	1	1	0	1
	UNUSED	1	1	0	0	0	0

Thus:

$$
\begin{aligned}
P &= Q_A \oplus Q_B \\
A &= Q_A \oplus Q_B \\
I &= \overline{Q_A}.Q_B \\
G &= Q_A.\overline{Q_B}
\end{aligned}
$$

Complete Circuit



Revision

State Diagram: Define output in each state. Define transitions for all combinations of inputs.

Bistable Allocation: n bistables provides 2^n states. Assign code to each state. Note the unused states.

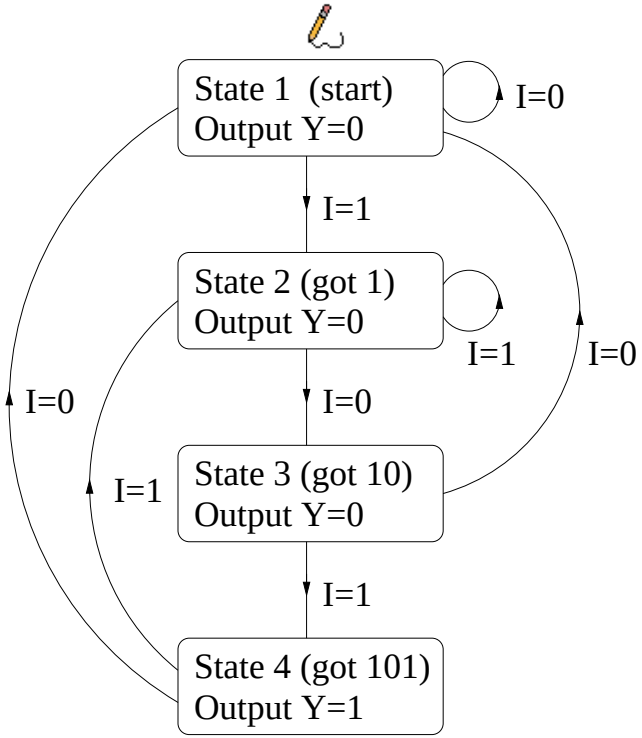
State Transition Table: Work out bistable inputs required to perform correct state transitions. Remember to include a separate line in the table for every combination of current state and input variables. Handle unused states.

Karnaugh Maps: Work out logic to implement the functions required by state transition table.

Circuit: Remember that all the bistables should be clocked synchronously (together).

Sequence Detector

Detect the received sequence 101 on an input I. When the sequence is detected $Y = 1$. Only respond to non-overlapping sequences. The sequence 10101 should give an output after bit 3, but not after bit 5.



Bistable Allocation

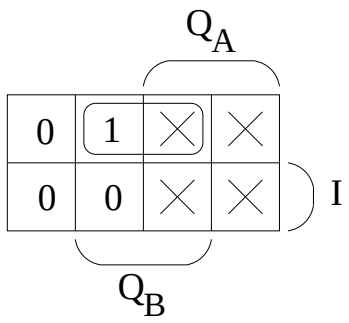
We have 4 states therefore we need 2 bistables and there are no unused states.

		Bistable Outputs	
		Q_A	Q_B
State 1	start	0	0
State 2	got 1	0	1
State 3	got 10	1	0
State4	got 101	1	1

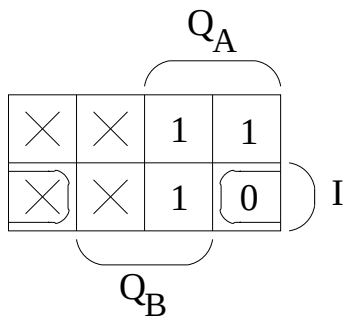
State Transition Table

Input	Current state		Next state		Required bistable inputs			
I	Q_A	Q_B	Q_A	Q_B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	×	0	×
1	0	0	0	1	0	×	1	×
0	0	1	1	0	1	×	×	1
1	0	1	0	1	0	×	×	0
0	1	0	0	0	×	1	0	×
1	1	0	1	1	×	0	1	×
0	1	1	0	0	×	1	×	1
1	1	1	0	1	×	1	×	0

Karnaugh Maps

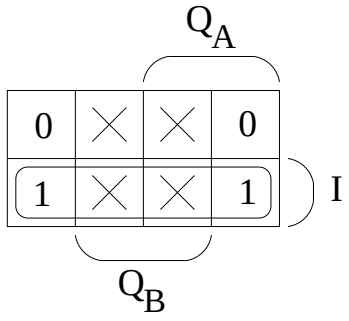


$J_A = \overline{I} \cdot Q_B$

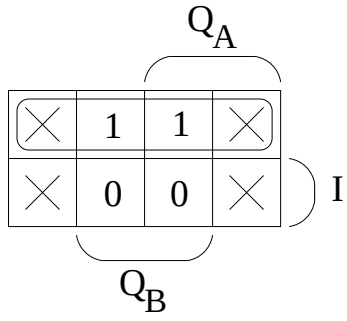


$\overline{K_A} = I \cdot \overline{Q_B}$

$K_A = \overline{I \cdot \overline{Q_B}}$



$J_B = I$



$K_B = \overline{I}$

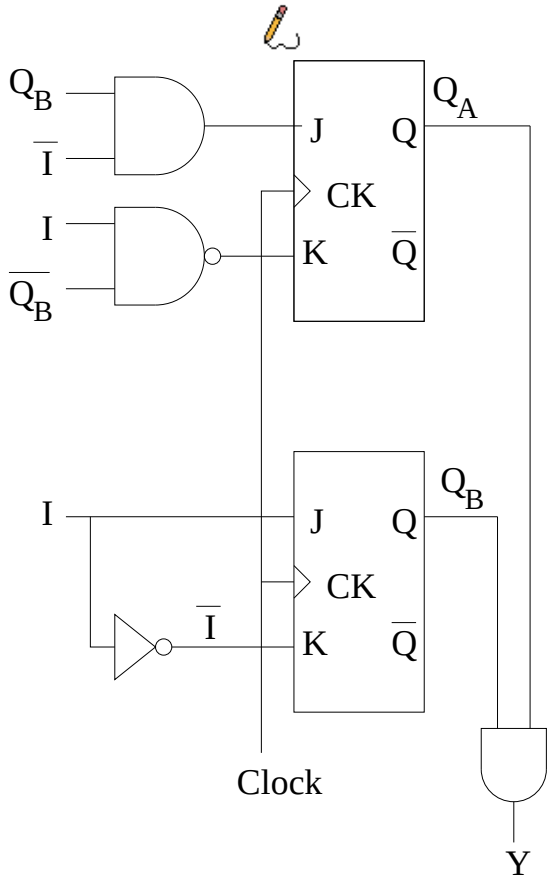
Output Logic

From the state diagram we have:

		Bistable Outputs		Output
		Q_A	Q_B	Y
State 1	start	0	0	0
State 2	got 1	0	1	0
State 3	got 10	1	0	0
State4	got 101	1	1	1

Thus $Y = Q_A.Q_B$

Circuit Diagram



Handout 2 Section E

Analogue Conversion &

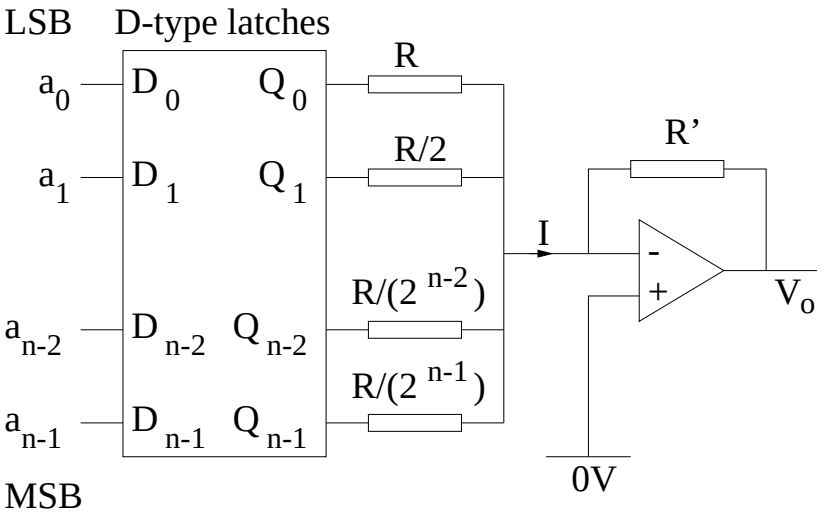
Adder Circuits

This section we introduce a number of circuits, using both combinational and sequential logic, that are used in the construction of microprocessor systems. It therefore provides background information for handout 3 on microprocessors as well as using techniques from handouts 1 and 2.

Circuits for digital to analogue and analogue to digital conversion are described. These types of circuit are often used to interface microprocessors to external instrumentation.

A binary adder circuit is then introduced. Binary adders form the heart of the arithmetic logic units inside microprocessors.

Weighted Resistor DAC



Assume the op-amp is ideal. Therefore the inverting input is a virtual earth and $V_o = -IR'$. Let the latch output voltage be A for each $Q = 1$, and 0 for each $Q = 0$.

$$I = \frac{a_0 A}{R} + \frac{2a_1 A}{R} + \ldots + \frac{2^{n-1} a_{n-1} A}{R}$$

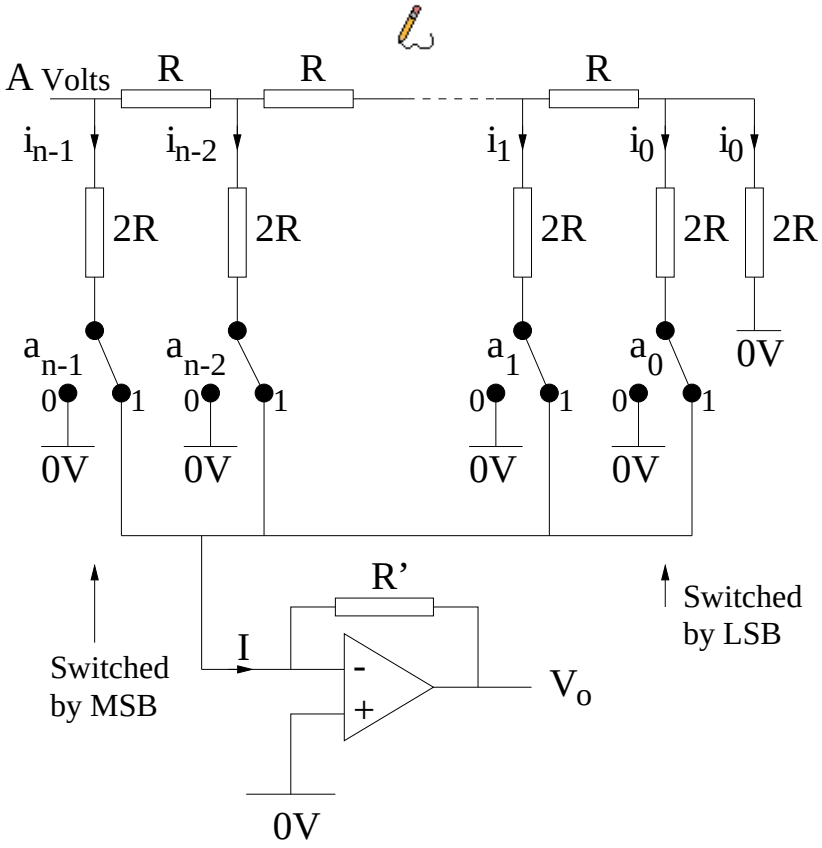
$$V_o = \left(\frac{-AR'}{R}\right) \sum_{m=0}^{n-1} a_m 2^m$$

Drawbacks

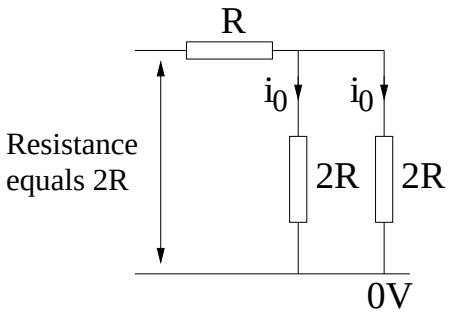
Limitations of the weighted resistor DAC.

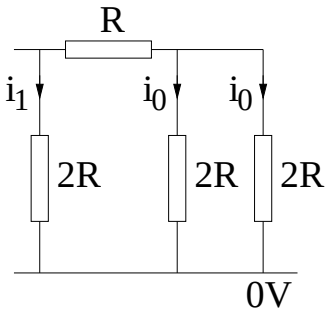
- Lots of resistance values which are difficult to implement.
- If the LSB is 5% accurate then the MSB must be $\frac{5\%}{2^{n-1}}$. If $n = 8$ this comes to 0.04% which is time consuming and expensive to achieve.
- A wide range of resistor values are required. These will need to be made from different materials so the way they change with temperature will be different. The output of the circuit will probably drift as it warms up.

R-2R Ladder DAC



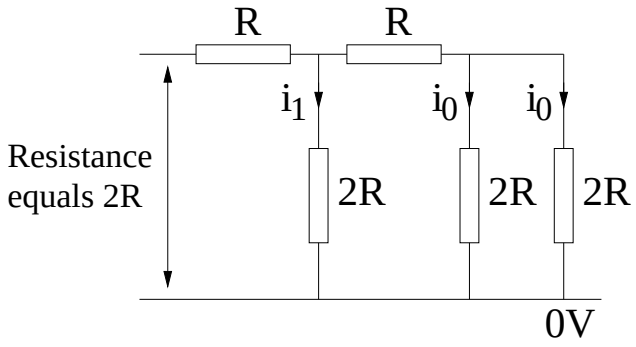
Assume the op-amp is ideal. Therefore the inverting input is a virtual earth and $V_o = -IR'$.

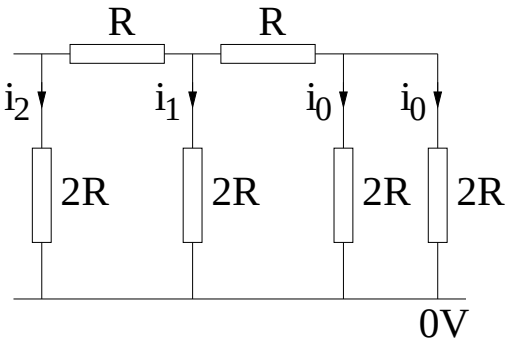




The resistance of the three resistors shown above is equal to $2R$.

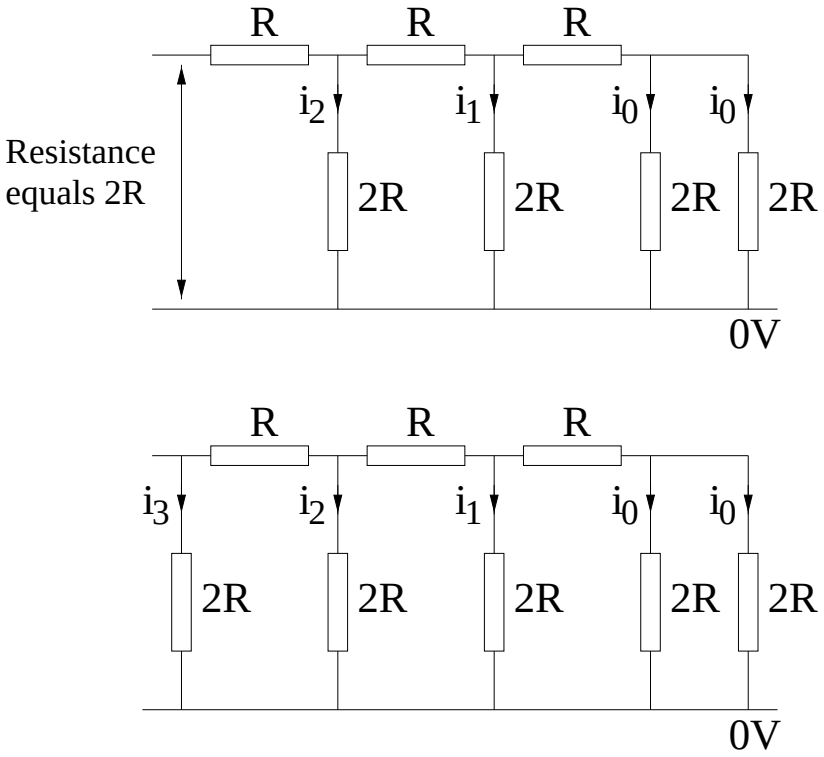
Therefore $i_1 = 2i_0$.





The resistance of the five resistors shown above is equal to $2R$.

Therefore $i_2 = 2i_1$.



The resistance of the seven resistors shown above is equal to $2R$.

Therefore $i_3 = 2i_2$.

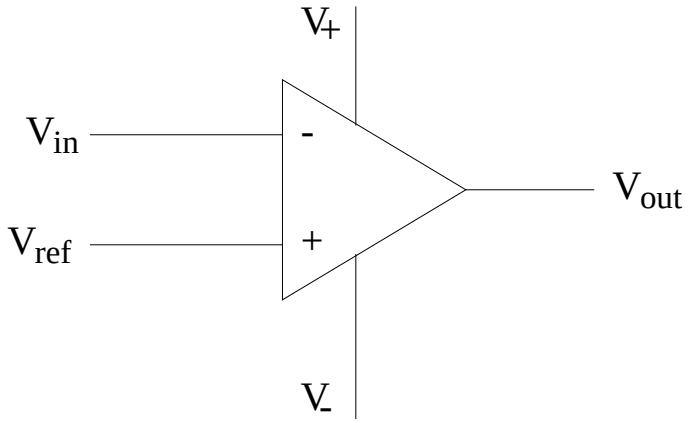
We have shown that $i_{m+1} = 2i_m$ so $i_m = 2^m i_0$

So $i_{n-1} = 2^{n-1} i_0$ and for the left-most resistor $A = 2^{n-1} i_0 \times 2R \quad \Rightarrow \quad i_0 = \frac{A}{2^n R}$

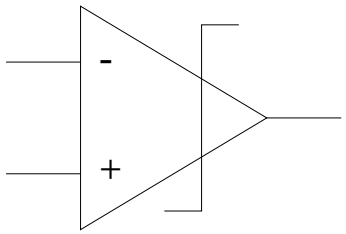
$$
\begin{aligned}
I &= \sum_{m=0}^{n-1} a_m i_m \\
&= \sum_{m=0}^{n-1} a_m 2^m \frac{A}{2^n R} \\
&= \frac{A}{2^n R} \sum_{m=0}^{n-1} a_m 2^m \\
\Rightarrow V_o &= \frac{-AR'}{2^n R} \sum_{m=0}^{n-1} a_m 2^m
\end{aligned}
$$

Analogue to Digital Conversion

1-bit ADC (comparator) without hysteresis.

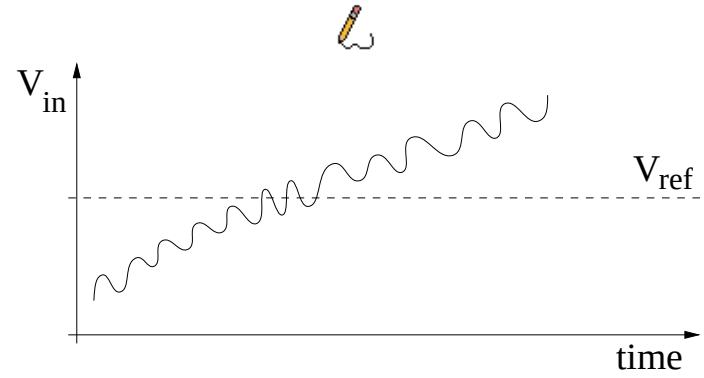


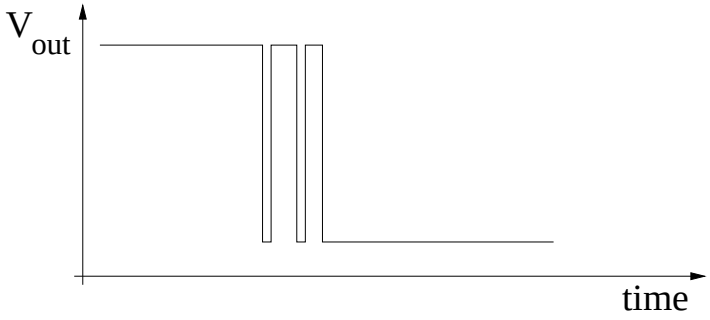
Symbol:



$V_{out} = \text{Gain} \times (V_{\text{ref}} - V_{\text{in}})$, but when the gain is very large:

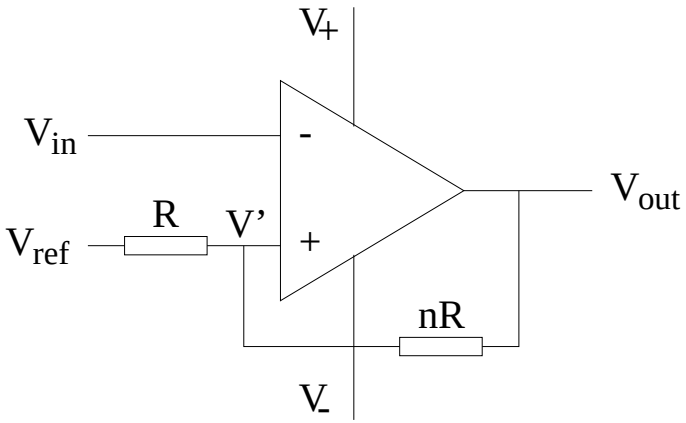
$$V_{in} > V_{ref} \Rightarrow V_{out} = V_{-}$$
$$V_{in} < V_{ref} \Rightarrow V_{out} = V_{+}$$



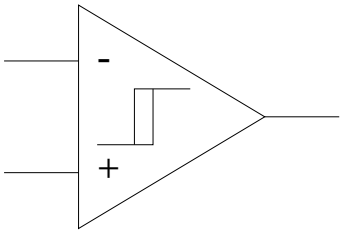


1-bit ADC (comparator) with hysteresis.

Schmidt trigger.



Symbol:

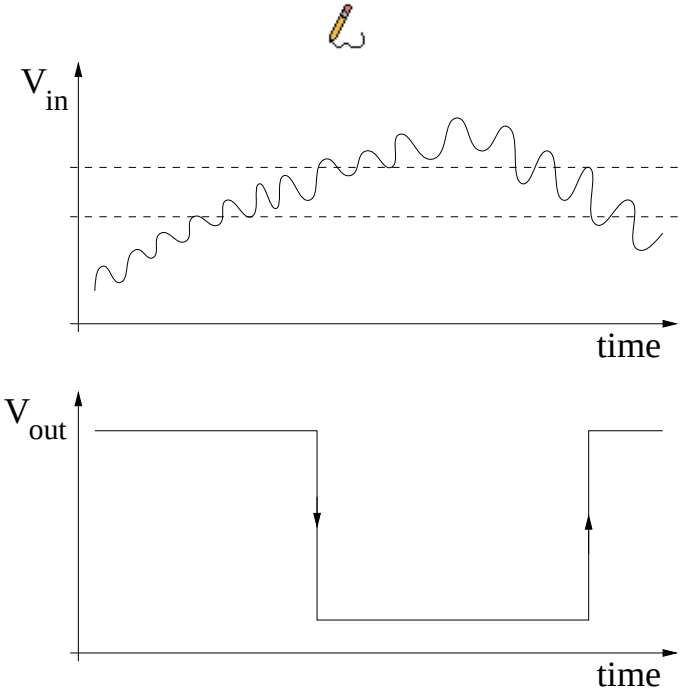


$$V_{in} > V' \;\Rightarrow\; V' = \frac{nV_{ref} + V_-}{1+n}$$

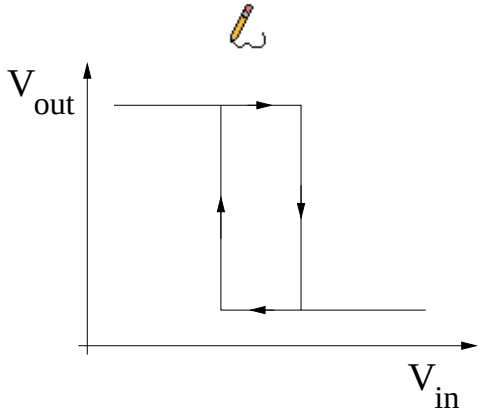
which gives the lower threshold

$$V_{in} < V' \;\Rightarrow\; V' = \frac{nV_{ref} + V_+}{1+n}$$

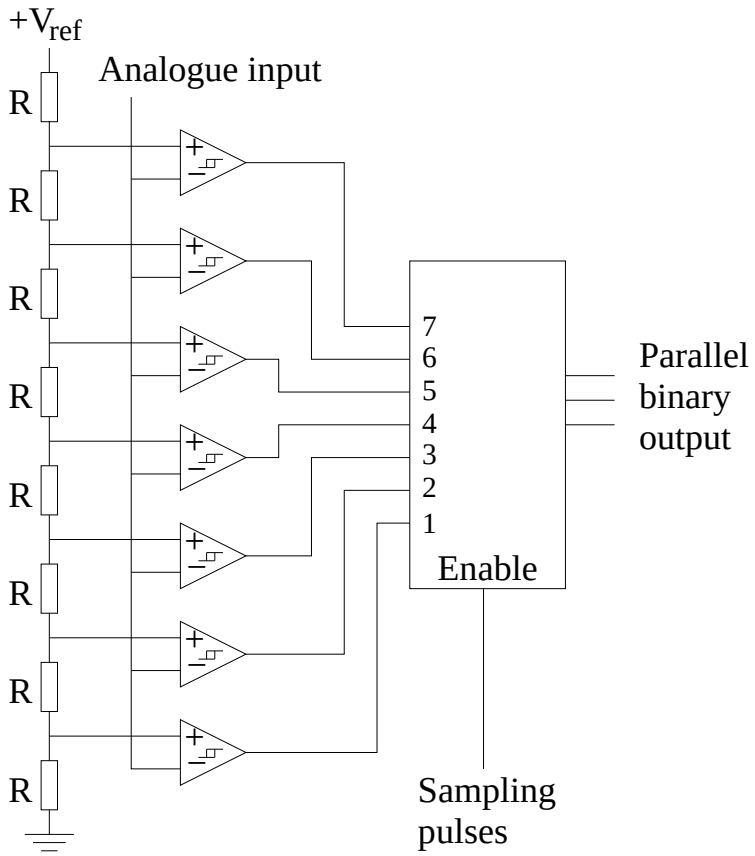
which gives the upper threshold



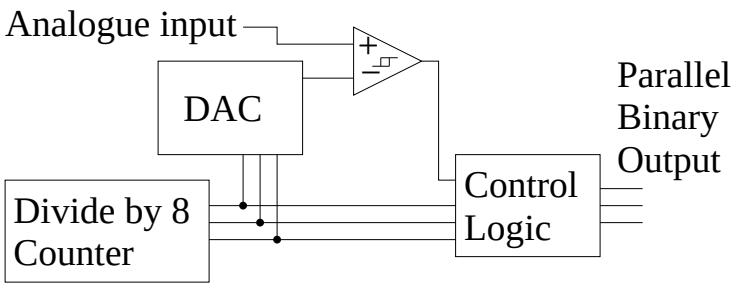
Transfer characteristics of the comparator with hysteresis.



Flash ADC



- 8 bit flash ADC requires 256 comparators.
- 12 bit flash ADC requires 4096 comparators!
- Alternative is to have a ramp that gradually rises created by a DAC and a single comparator to detect the value at which the input becomes less than the ramp value. This is called the *stair-step ramp* or *counter* method.
- The *stair-step ramp* method is much slower than the flash ADC.

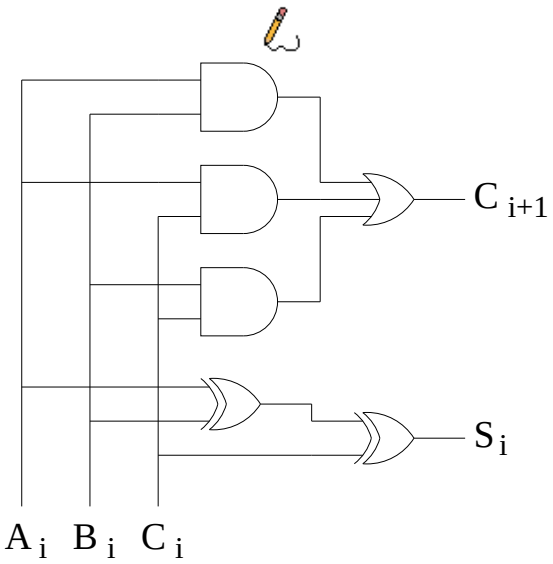


Full Adder

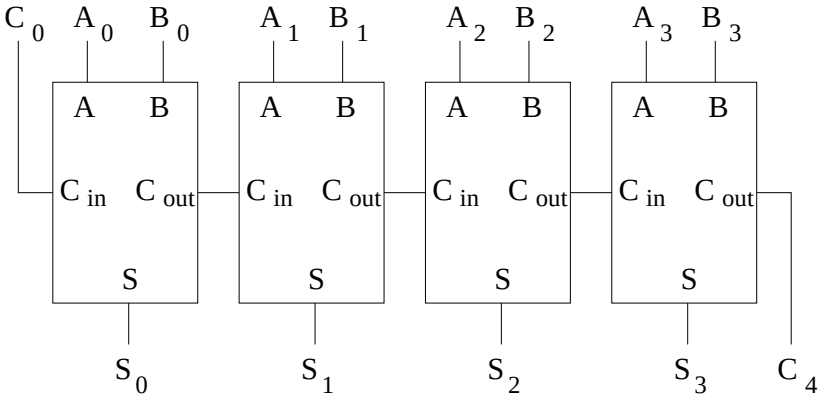
A_i	B_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$C_{i+1} = A_i.B_i + A_i.C_i + B_i.C_i$$
$$S_i = A_i \oplus B_i \oplus C_i$$

Full Adder Circuit



Ripple Carry Adder Circuit



What happens if $\overline{A_0}$, $\overline{A_1}$, $\overline{A_2}$ and $\overline{A_3}$ are fed to the A inputs and C_0 is set to 1?

In this case $S = B - A$ because we have changed the sign of A by inverting it and adding 1 (using C_0).