EMBEDDED SYSTEMS CMPE-453

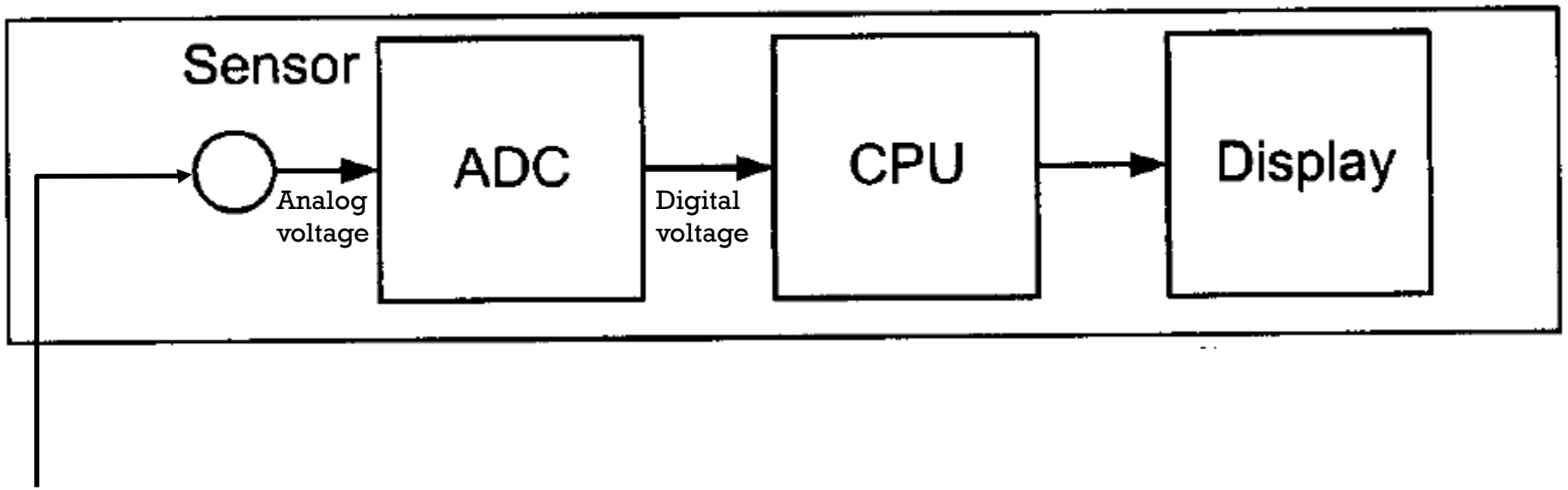
Department of Computer Engineering

ADC (Analog to Digital Converter)

WHY ADC?

- Microcontrollers need to interact with the environment.
- Physical quantities are analog in real life.
  - Temperature, pressure, distance, acceleration, force etc.
- Microcontroller operates in digital domain i.e. Logic 1 and 0.
  - Step 1: Convert physical quantities to a voltage (Sensors).
  - Step 2: Scale the voltage so that it is within the acceptable range for the microcontroller (Signal conditioning).
  - Step 3: Convert the analog voltage to digital value (ADC).

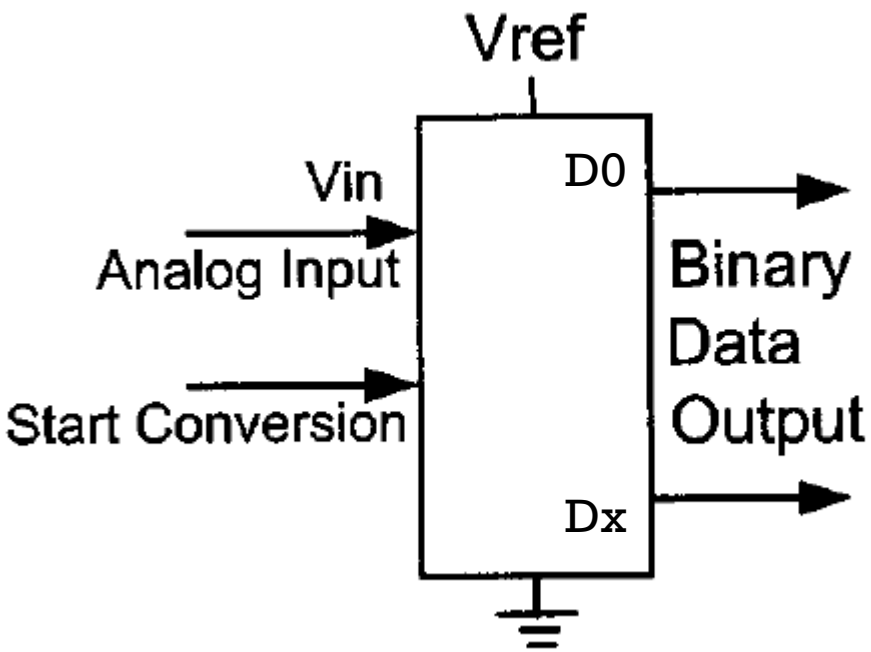
ADC DEVICES



Physical quantities:

Temperature, pressure, distance, acceleration, force etc.

ADC Block Diagram



Resolution & Step Size

Table 13-1: Resolution versus Step Size for ADC (V_{ref} = 5 V)

n-bit	Number of steps	Step size (mV)
8	256	5/256 = 19.53
10	1024	5/1024 = 4.88
12	4096	5/4096 = 1.2
16	65,536	5/65,536 = 0.076

Step size is the smallest change that can be discerned by an ADC.

Conversion Time

Time spent for convertion of analog input to digital number

- Clock source connected to ADC
- Data conversion method
- Technology used :
  - MOS (Metal Oxide Semiconductor)

  or
  - TTL (Transistor-Transistor Logic)

$$Vref = Vref(+) - Vref(-)$$

Vref

Table 13-2: V_{ref} Relation to V_{in} Range for an 8-bit ADC

V_{ref} (V)	V_{in} Range (V)	Step Size (mV)
5.00	0 to 5	5/256 = 19.53
4.0	0 to 4	4/256 = 15.62
3.0	0 to 3	3/256 = 11.71
2.56	0 to 2.56	2.56/256 = 10
2.0	0 to 2	2/256 = 7.81
1.28	0 to 1.28	1.28/256 = 5
1	0 to 1	1/256 = 3.90

Step size is V_{ref}/ 256

Table 13-3: V_{ref} Relation to V_{in} Range for an 10-bit ADC

V_{ref} (V)	V_{in} (V)	Step Size (mV)
5.00	0 to 5	5/1024 = 4.88
4.096	0 to 4.096	4.096/1024 = 4
3.0	0 to 3	3/1024 = 2.93
2.56	0 to 2.56	2.56/1024 = 2.5
2.048	0 to 2.048	2.048/1024 = 2
1.28	0 to 1.28	1/1024 = 1.25
1.024	0 to 1.024	1.024/1024 = 1

Digital Data Output

$$D_{out} = \frac{V_{in}}{step\ size} = \frac{V_{in}}{V_{ref}/2^n}$$

Example

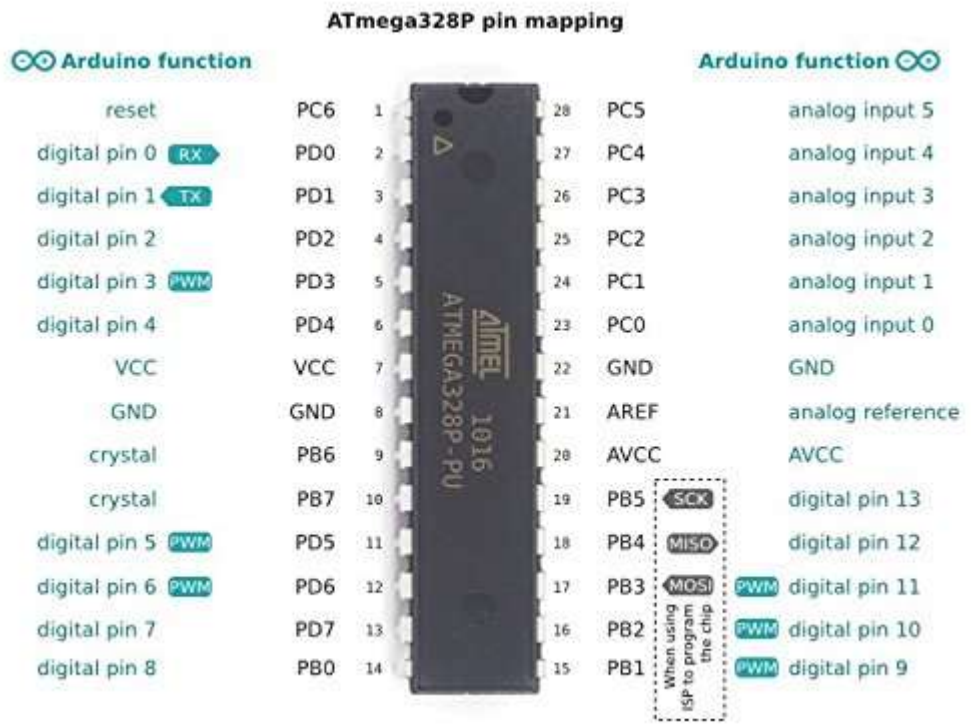
For an 8-bit ADC, we have V_{ref} = 2.56 V. Calculate the D0–D7 output if the analog input is: (a) 1.7 V, and (b) 2.1 V.

Solution

step size is 2.56/256 = 10 mV,

(a) D_{out} = 1.7 V/10 mV=170 in decimal, which gives us 10101010 in binary for D7–D0.
(b) D_{out} = 2.1 V/10 mV = 210 in decimal, which gives us 11010010 in binary for D7–D0.

ANALOG INPUT PINS



arduinomblog.wordpress.com

- Multiple input channels
- Single digital output register
- Need SC (Start conversion) and EOC (End of conversion)
- Analog Vin to n-bit digital number

ADC IN AVR-MICROCONTROLLER

- An on chip ADC
- Uses successive approximation for conversion
- How successive approximation Works?

SUCCESSIVE APPROXIMATION

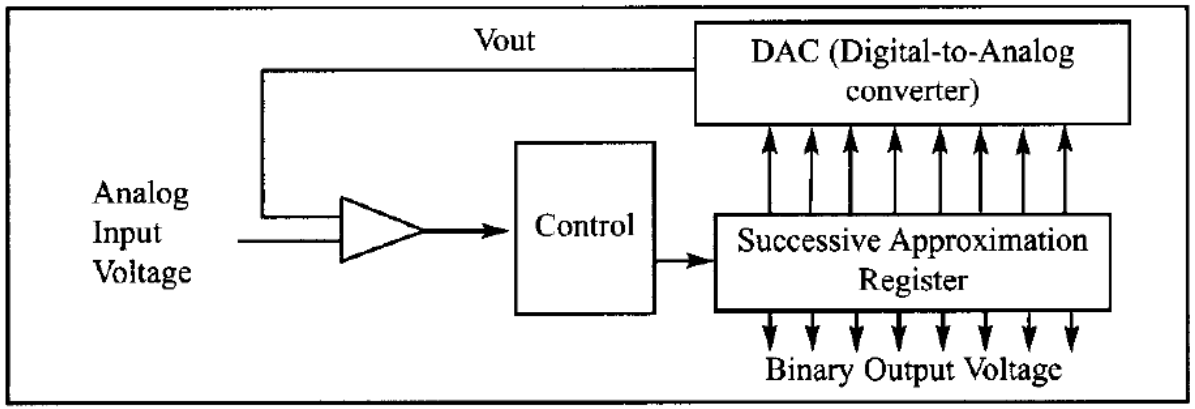
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



Example: Calculating V_{DAC} from SAR

SAR = 1010

V_{DAC} = (SAR) x Stepsize = (1010) x 1 V = 10 V.

SUCCESSIVE APPROXIMATION

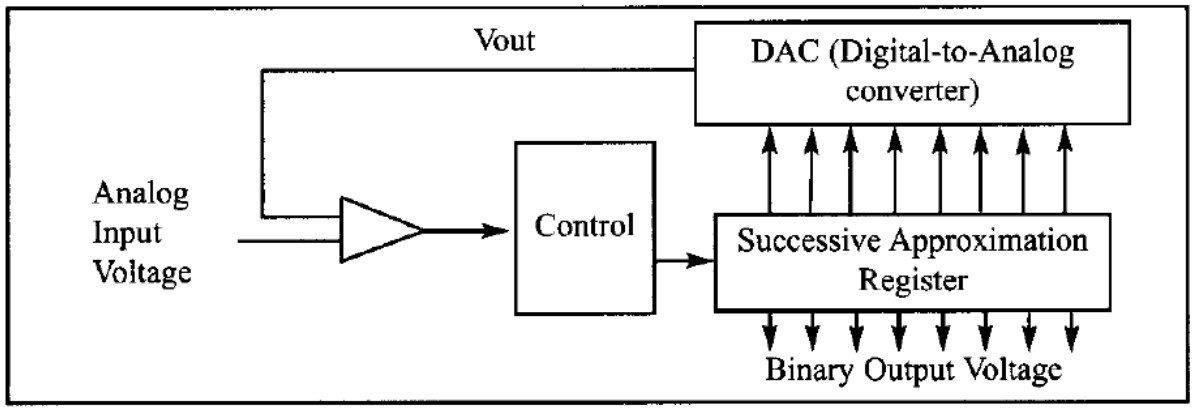
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
				3	

SUCCESSIVE APPROXIMATION

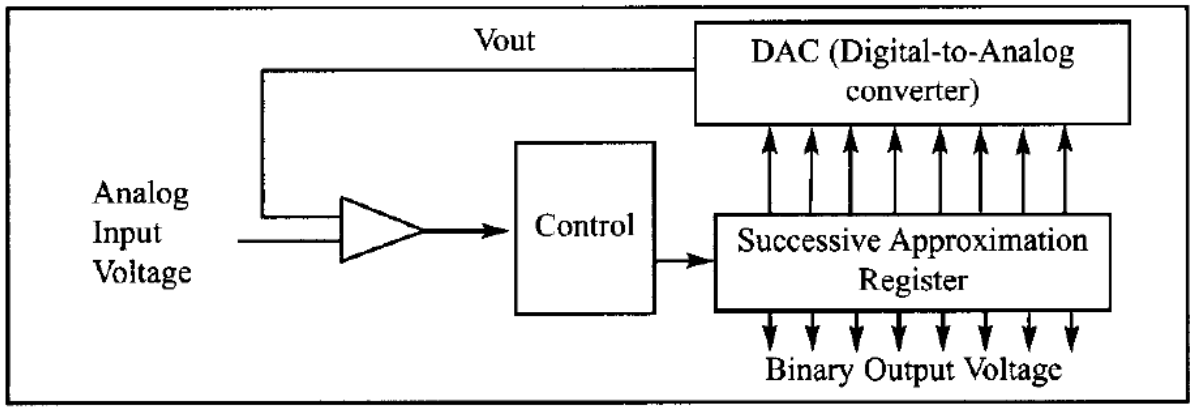
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
	1000			3	

SUCCESSIVE APPROXIMATION

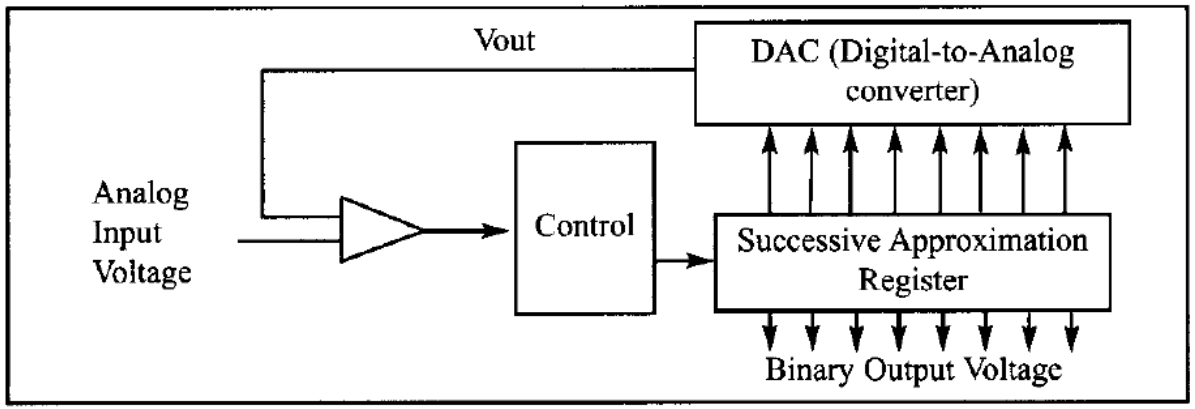
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000			3	

SUCCESSIVE APPROXIMATION

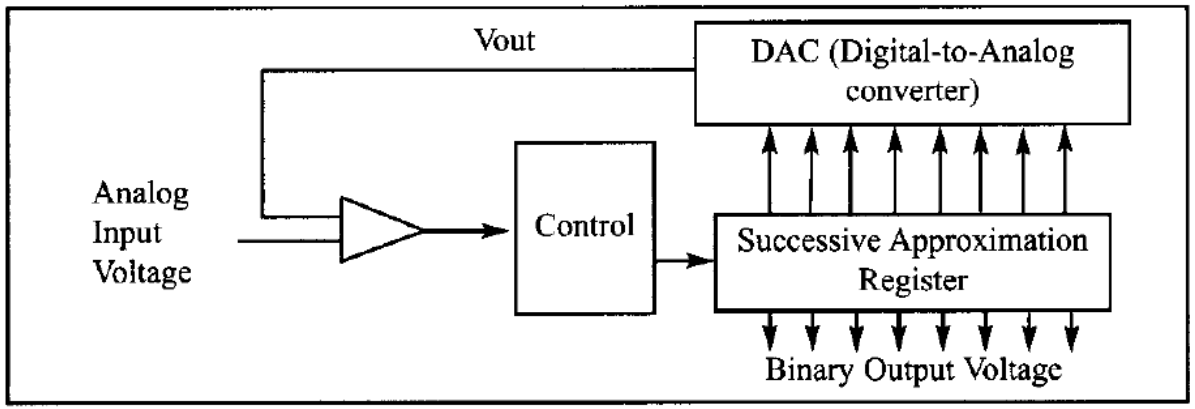
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000			3	2

SUCCESSIVE APPROXIMATION

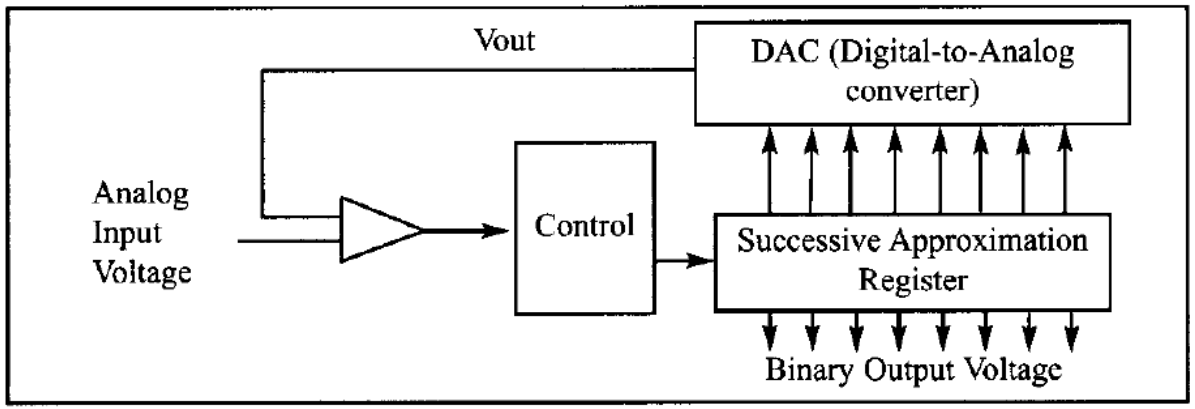
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000			3	2
	x100				

SUCCESSIVE APPROXIMATION

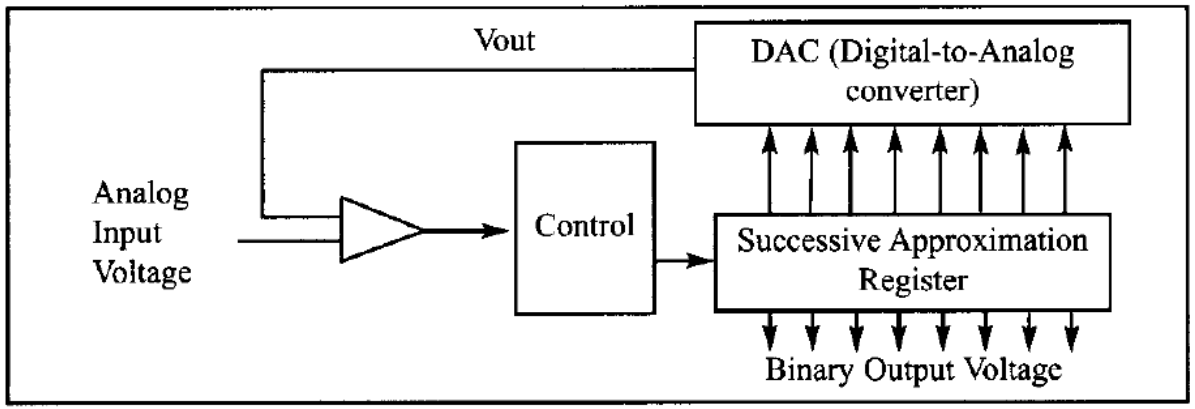
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	x100				

SUCCESSIVE APPROXIMATION

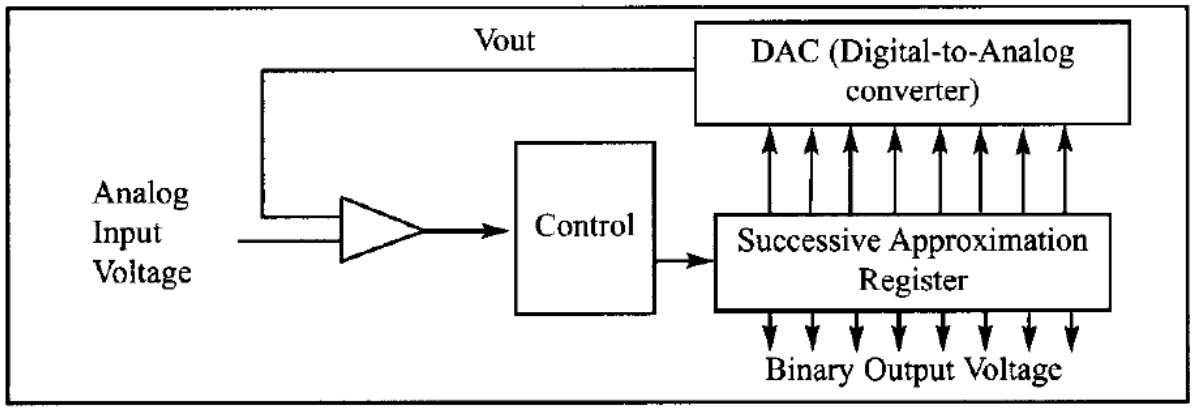
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100				

SUCCESSIVE APPROXIMATION

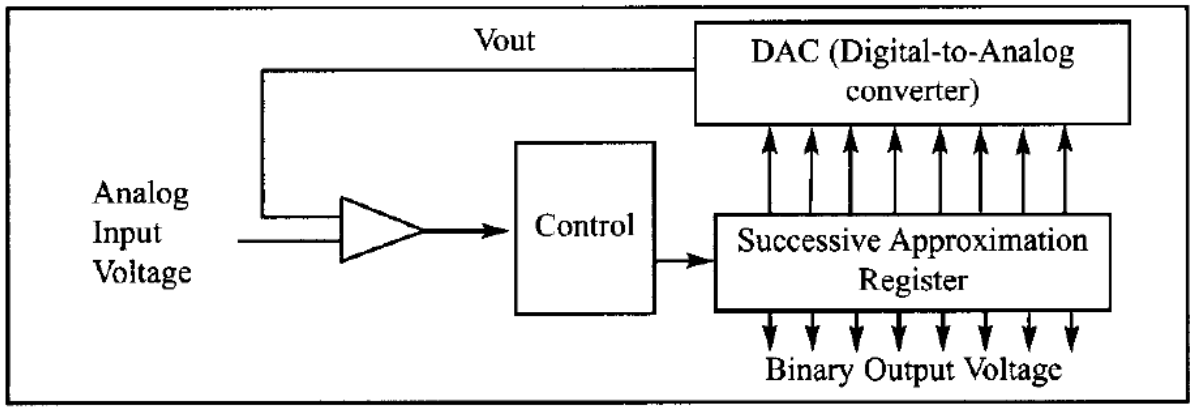
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100			2	

SUCCESSIVE APPROXIMATION

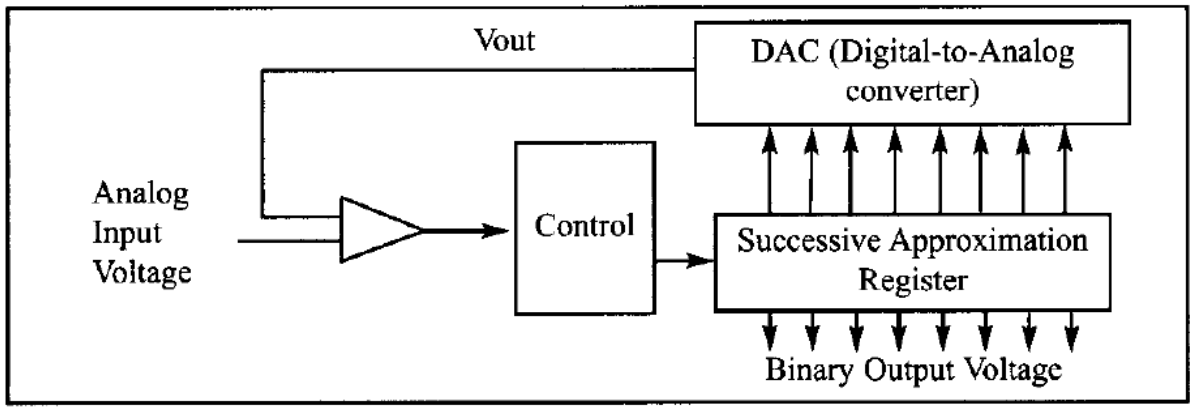
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100			2	1

SUCCESSIVE APPROXIMATION

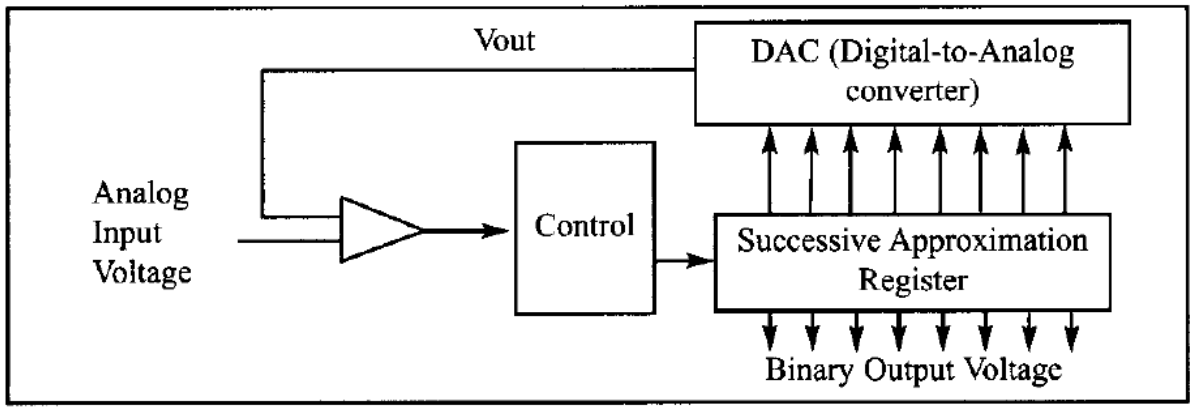
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100			2	1
	1x10				

SUCCESSIVE APPROXIMATION

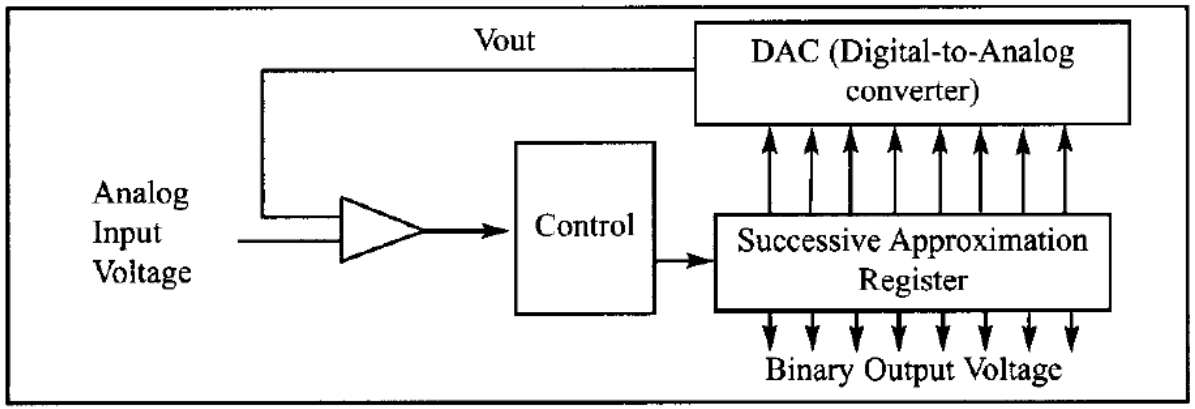
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1x10				

SUCCESSIVE APPROXIMATION

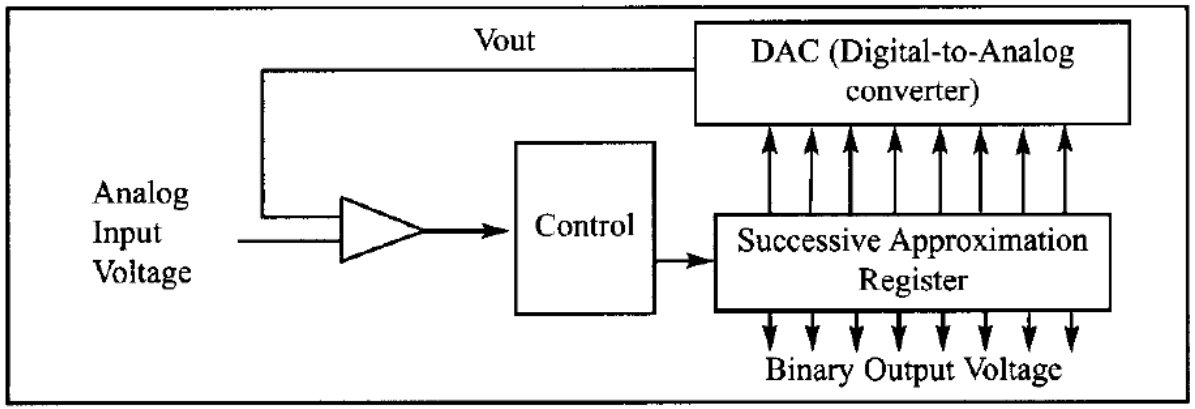
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010				

SUCCESSIVE APPROXIMATION

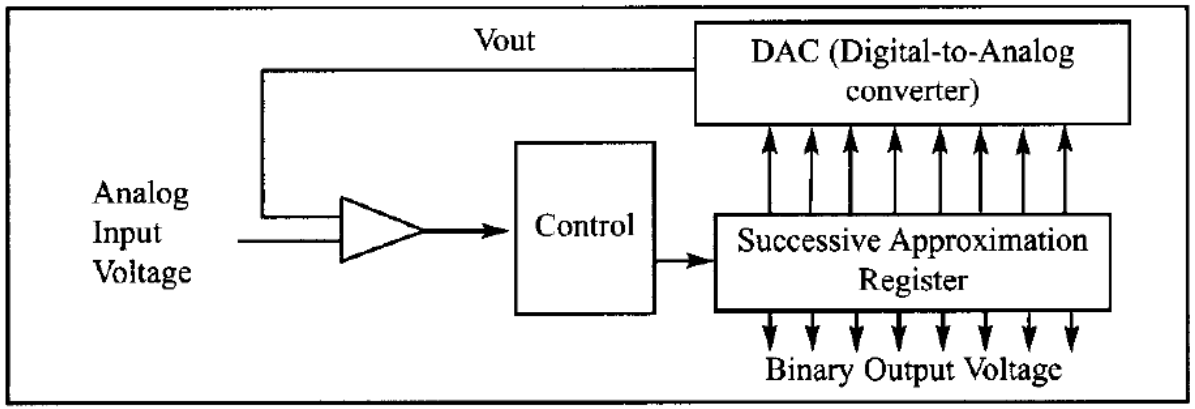
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010			1	

SUCCESSIVE APPROXIMATION

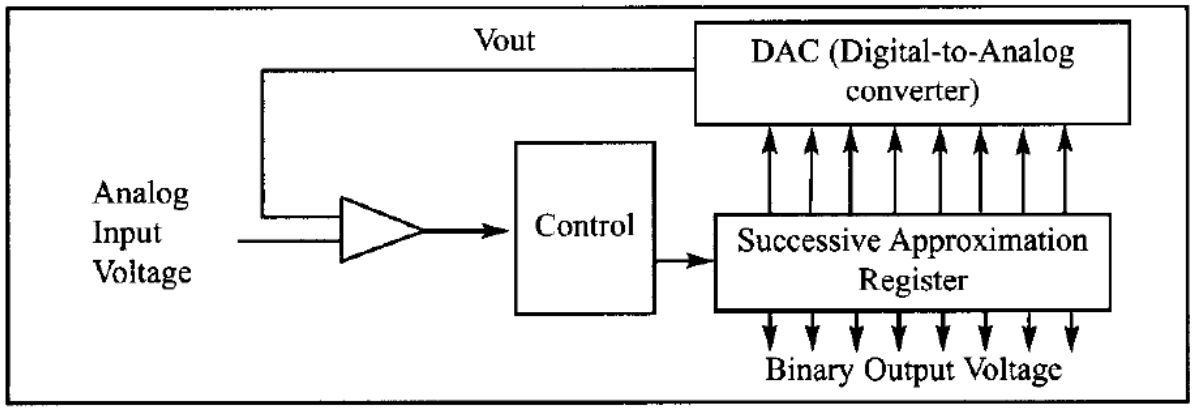
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010			1	0

SUCCESSIVE APPROXIMATION

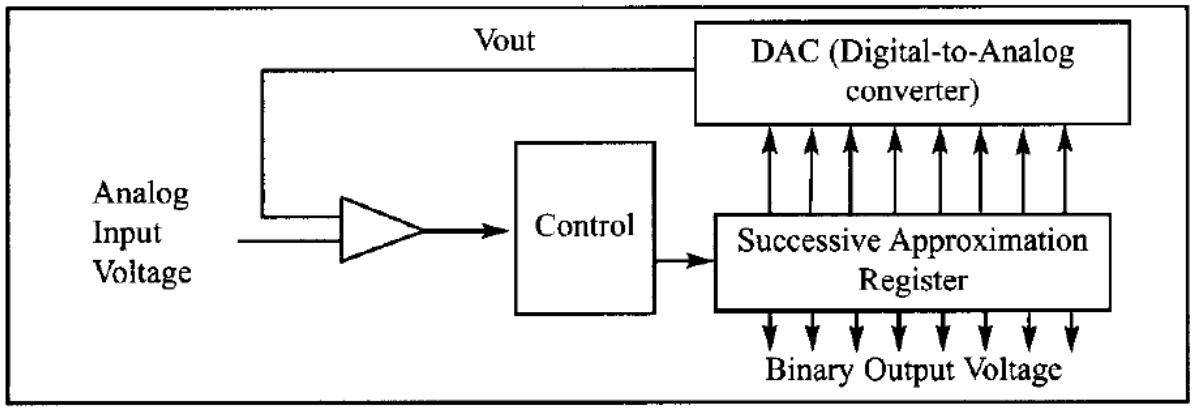
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010			1	0
	10x1				

SUCCESSIVE APPROXIMATION

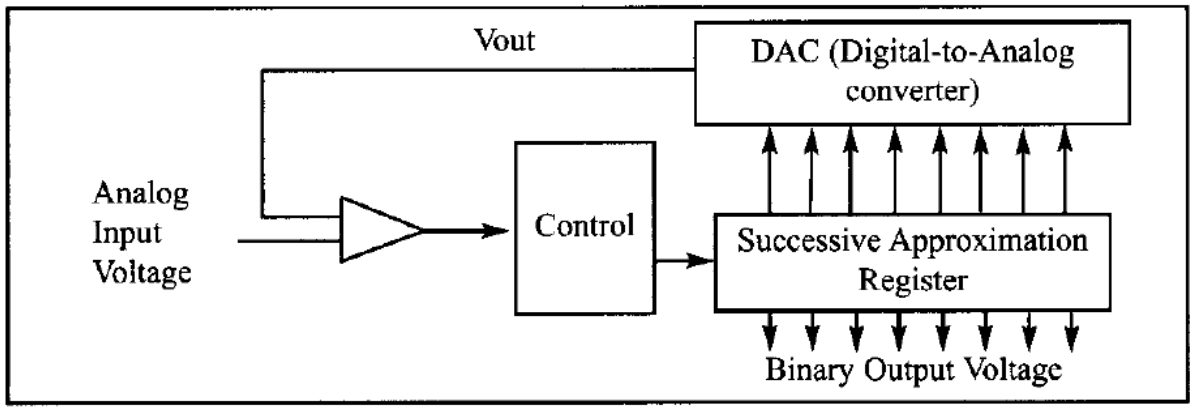
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010	10	$V_{in} > V_{DAC}$	1	0
	10x1				

SUCCESSIVE APPROXIMATION

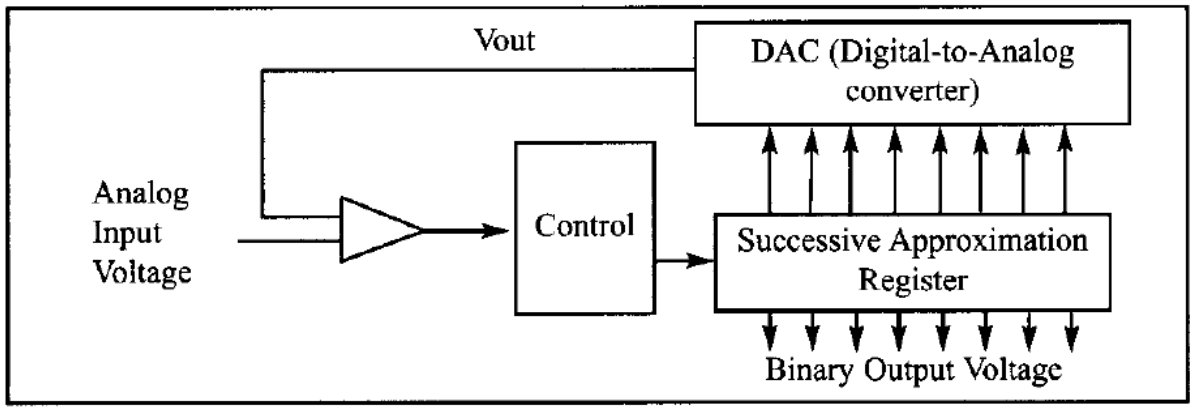
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010	10	$V_{in} > V_{DAC}$	1	0
	1011				

SUCCESSIVE APPROXIMATION

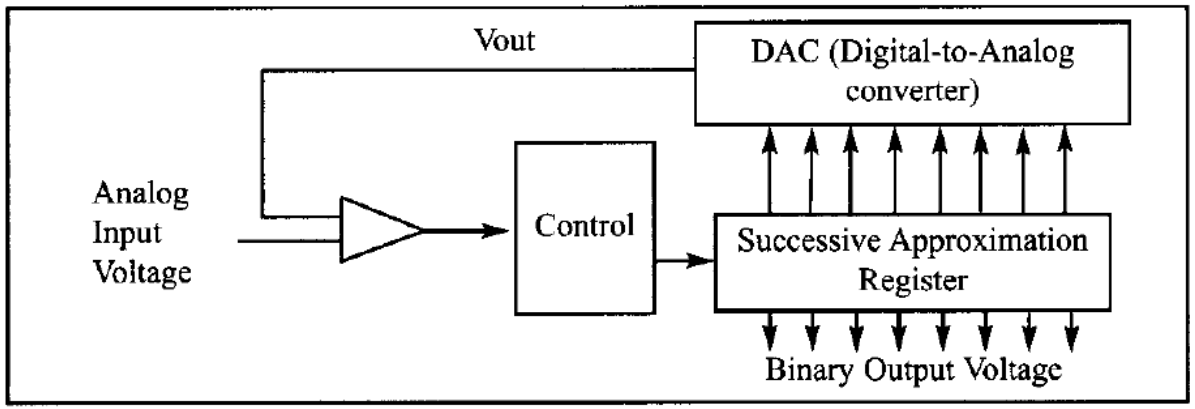
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010	10	$V_{in} > V_{DAC}$	1	0
	1011			0	

SUCCESSIVE APPROXIMATION

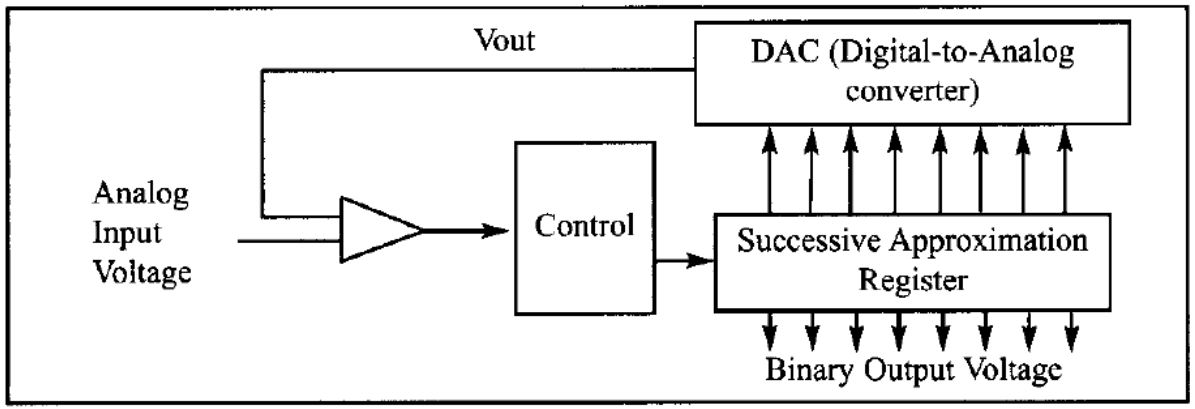
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010	10	$V_{in} > V_{DAC}$	1	0
	1011			0	-1

SUCCESSIVE APPROXIMATION

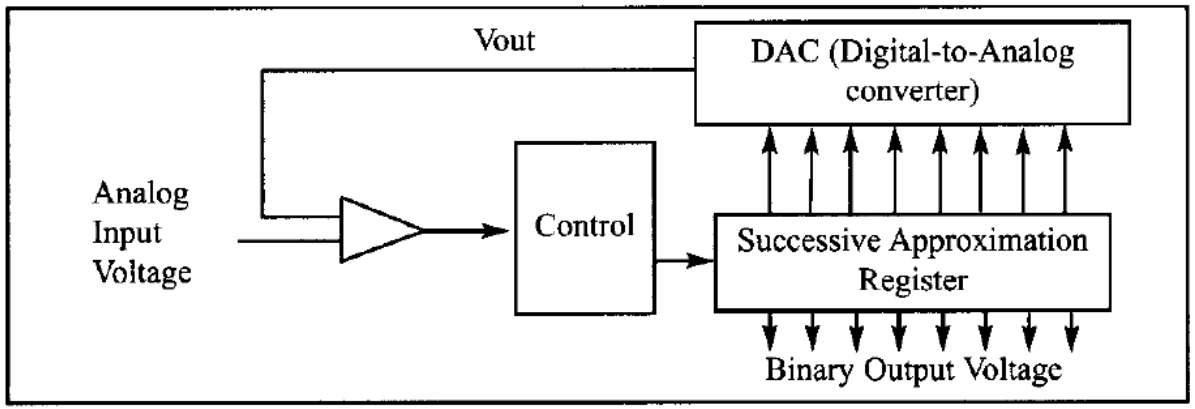
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010	10	$V_{in} > V_{DAC}$	1	0
	1011			0	-1
	101x				

SUCCESSIVE APPROXIMATION

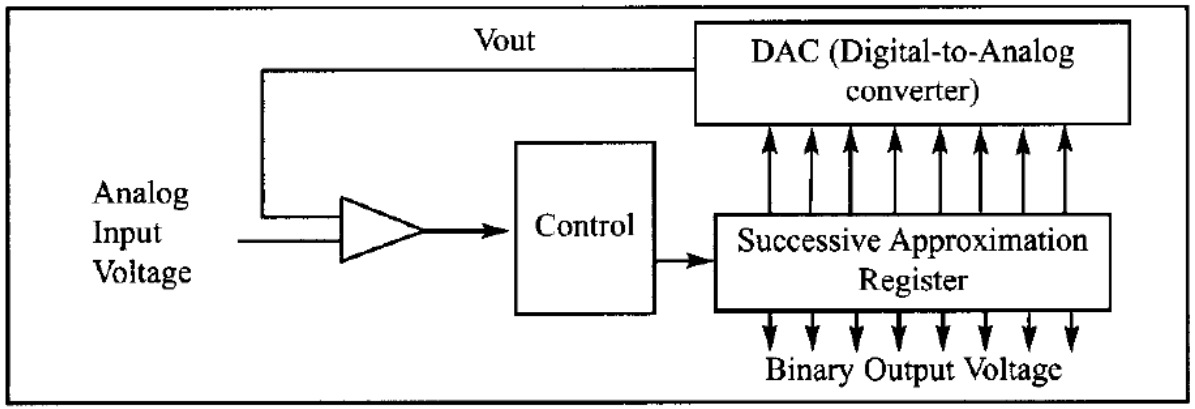
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010	10	$V_{in} > V_{DAC}$	1	0
	1011	11	$V_{in} > V_{DAC}$	0	-1
	101x				

SUCCESSIVE APPROXIMATION

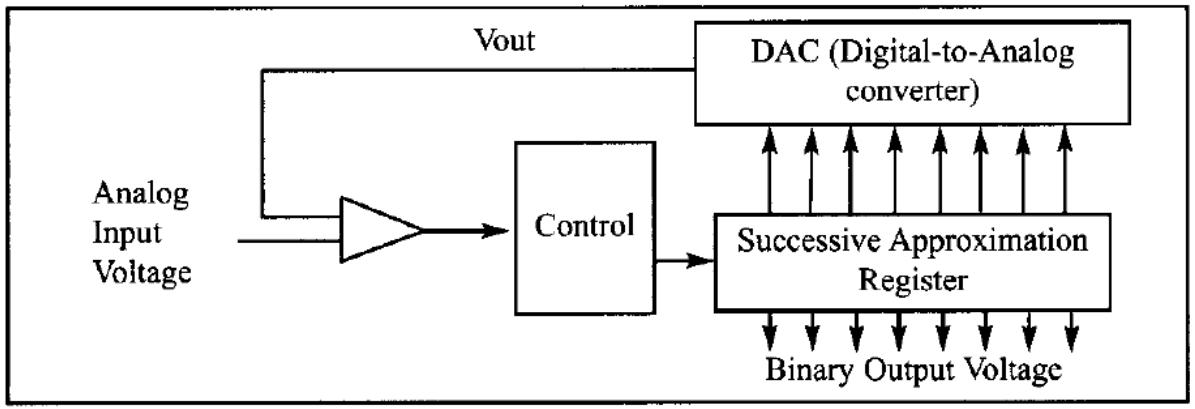
- Step 1: set index =n-1 (i.e. 3)
- Step 2: set SAR = 1000
- Step 3: read an input-voltage i.e. V_{in}
- Step 4: if (Index > 0) set SAR[Index -1] =1
- Step 5:

  if $V_{in} > V_{DAC}$

  SAR[Index]=1

  else

  SAR[Index]=0
- Step 6: Index--
- Step 7: Go to Step 4



V_{in} (V)	4-bit SAR	V_{DAC} (V)	Comment	Index	Index-1
11.2	1000	8	$V_{in} > V_{DAC}$	3	2
	1100	12	$V_{in} < V_{DAC}$	2	1
	1010	10	$V_{in} > V_{DAC}$	1	0
	1011	11	$V_{in} > V_{DAC}$	0	-1
	1011	11			

IN CASE OF ATMEGA328P

- 10-bit ADC
- 6 analog inputs
- Converted digital output data : ADCL (LSB Byte) and ADCH (HSB Byte) : Total 16 bits
- Output : 10 bits, 6bits of of 16bits are unused, the option to either the upper or lower bits unused.
- Vref can be selected as:
  - AVCC (Analog Vcc)
  - Internal 1.1 V reference
  - External AREF pin.
- Conversion time based on crystal frequency connected to XTAL pins (Fosc) and ADPS0:2 bits

Example

For an 10-bit ADC, we have $V_{ref} = 2.56$ V. Calculate the D0–D9 output if the analog input is: (a) 0.2 V, and (b) 0 V. How much is the variation between (a) and (b)?

Solution

Because the step size is 2.56/1024 = 2.5 mV, we have the following:
(a) D_{out} = 0.2 V/2.5 mV = 80 in decimal, which gives us 1010000 in binary.

(b) D_{out} = 0 V/2.5 mV = 0 in decimal, which gives us 0 in binary.

The difference is 1010000, which is 7 bits!

HOW TO SELECT V_{REF}?

ADMUX: ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

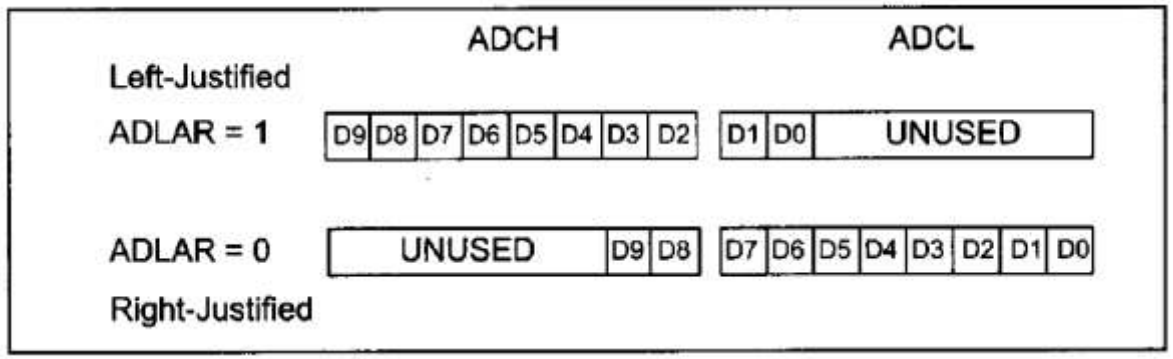
Table 23-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, internal V_{REF} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V voltage reference with external capacitor at AREF pin

HOW TO SELECT ADC INPUT CHANNEL

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6



ADC DATA REGISTER

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

When ADCL is read, the ADC data register is not updated until ADCH is read

HOW TO SELECT CLOCK PRESCALAR

ADC operates reliably between 50KHz to 200 KHz clock frequency

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CPU clock	Prescale	ADC frequency	ADCSRA bits set
1 MHz	4	250 kHz	ADPS1
	8	125 kHz	ADPS1 and ADPS0
	16	62.5 kHz	ADPS2
	32	31.25 kHz	ADPS2 and ADPS0
8 MHz	16	250 kHz	ADPS2
	32	125 kHz	ADPS2 and ADPS0
	64	62.5 kHz	ADPS2 and ADPS1
	128	31.25 kHz	ADPS2 and ADPS1 and ADPS0
12 MHz	64	187.5 kHz	ADPS2 and ADPS1
	128	93.75 kHz	ADPS2 and ADPS1 and ADPS0
16 MHz	64	250 kHz	ADPS2 and ADPS1
	128	125 kHz	ADPS2 and ADPS1 and ADPS0

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADCSRA: OTHER IMPORTANT BIT FIELDS

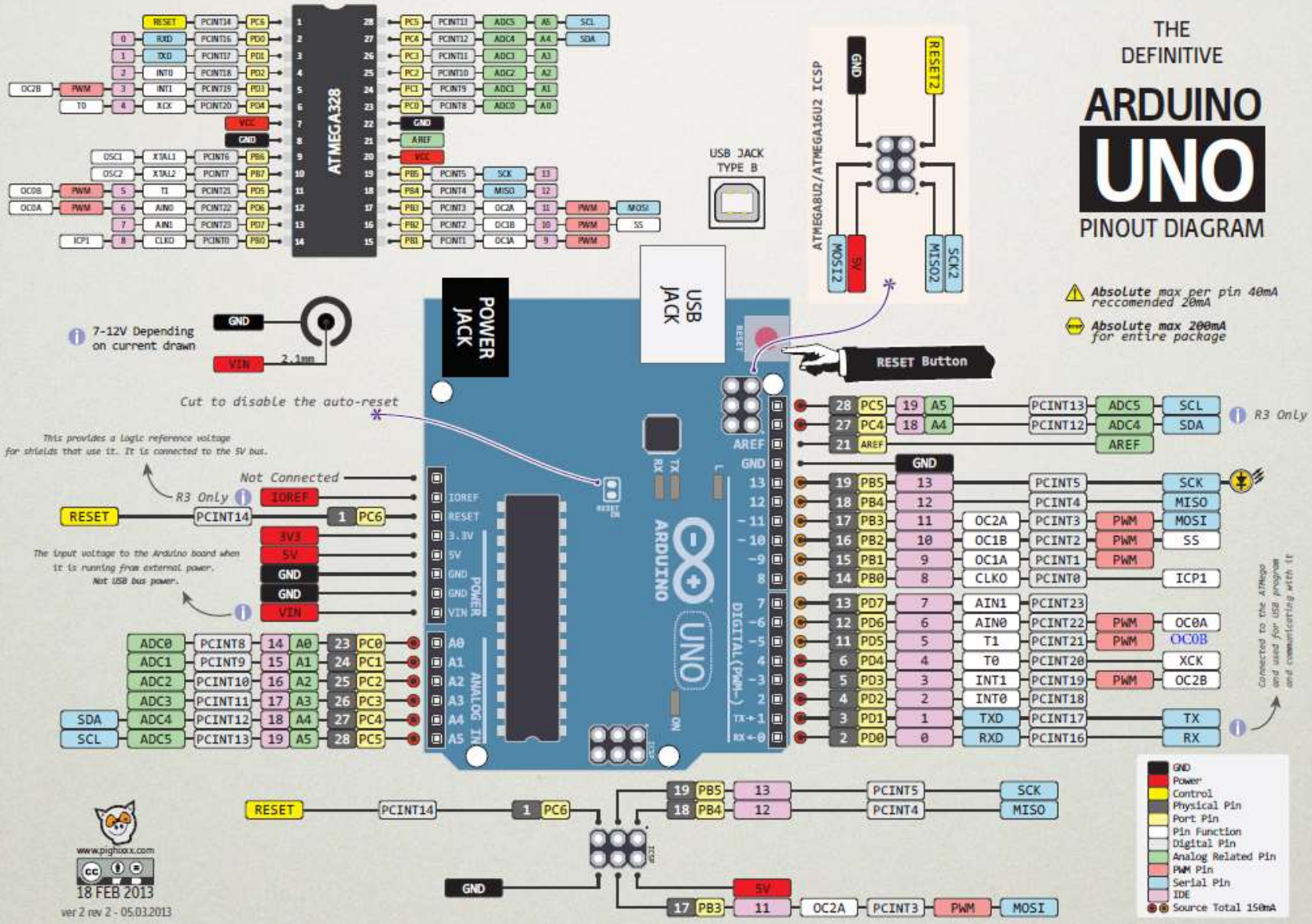
- ADEN:
  - 1: Enable ADC
  - 0: Disable ADC
- ADSC:
  - In single conversion mode:
    - Writing 1: start a new conversion
    - Writing 0: No effect
    - Reading 1: conversion in progress
    - Reading 0: conversion completed.
  - In free running mode:
    - Write 1 only to start first conversion.
- ADATE
  - 1: Enable Auto-trigger

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

STEPS TO USE ADC

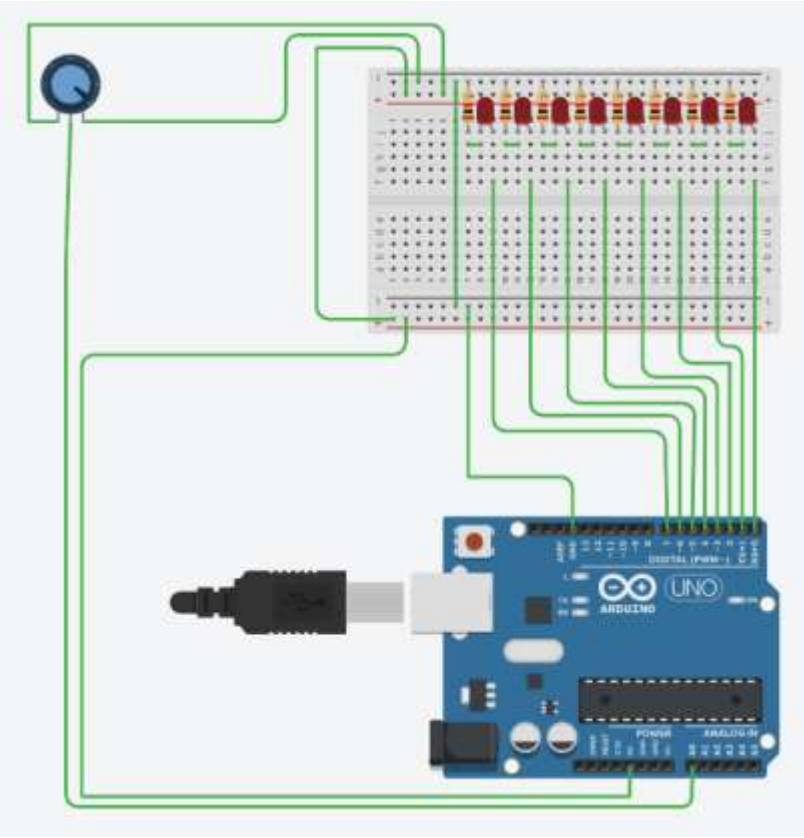
I. Set ADC clock pre-scalar (deafult: disabled i.e. No conversion).
II. Define VREF. (Default: Externally supplied ref. Voltage)
III. Select Analog channel (Default: PC0).
IV. ADC mode (single conversion vs auto-triggered)
V. Interrupt to call when ADC completes (optional).
VI. Other options: 8-bit mode, power-safe etc.

Arduino Connections



EXAMPLE 1 ADC: WITH POTENTIOMETER

```
Initialize ADC
Initialize LED port
while(1)
    {
    start a conversion
    wait for the conversion to complete
    get three MSBs of ADC-data register
    turn on the LEDs one by one according to the value
        represented by 3-MSBs
    display it on LEDs
    }
```



```
#include <avr/io.h>
#include <util/delay.h>
#include "pinDefines.h"
#define LED_DDR DDRD
#define LED_PORT PORTD
// -------- Functions --------- //
static inline void initADC0(void)
        {
        ADMUX |= (1 << REFS0); /* reference voltage on AVCC */
        ADCSRA |= (1 << ADPS2); /* ADC clock prescaler /16 */
        ADCSRA |= (1 << ADEN); /* enable ADC */
        }
int main(void)
        {
        // -------- Inits --------- //
        uint8_t ledValue;
        uint16_t adcValue;
        uint8_t i;
        initADC0();
        LED_DDR = 0xff;
        // ------ Event loop ------ //
        while (1)
                {
                ADCSRA |= (1 << ADSC); /* start ADC conversion */
                loop_until_bit_is_clear(ADCSRA, ADSC); /* wait until done */
                adcValue = ADC; /* read ADC in */
                /* Have 10 bits, want 3 (eight LEDs after all) */
                ledValue = (adcValue >> 7);
                /* Light up all LEDs up to ledValue */
                LED_PORT = 0;
                for (i = 0; i <= ledValue; i++)
                        {
                        LED_PORT |= (1 << i);
                        }
                _delay_ms(50);
                } /* End event loop */
        return 0; /* This line is never reached */
        }
```

0110100110 >> 7

= 011

HOW TO SELECT V_{REF}?

ADMUX: ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 23-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, internal V_{REF} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V voltage reference with external capacitor at AREF pin

HOW TO SELECT CLOCK PRESCALAR

ADC operates reliably between 50KHz to 200 KHz clock frequency

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CPU clock	Prescale	ADC frequency	ADCSRA bits set
1 MHz	4	250 kHz	ADPS1
	8	125 kHz	ADPS1 and ADPS0
	16	62.5 kHz	ADPS2
	32	31.25 kHz	ADPS2 and ADPS0
8 MHz	16	250 kHz	ADPS2
	32	125 kHz	ADPS2 and ADPS0
	64	62.5 kHz	ADPS2 and ADPS1
	128	31.25 kHz	ADPS2 and ADPS1 and ADPS0
12 MHz	64	187.5 kHz	ADPS2 and ADPS1
	128	93.75 kHz	ADPS2 and ADPS1 and ADPS0
16 MHz	64	250 kHz	ADPS2 and ADPS1
	128	125 kHz	ADPS2 and ADPS1 and ADPS0

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

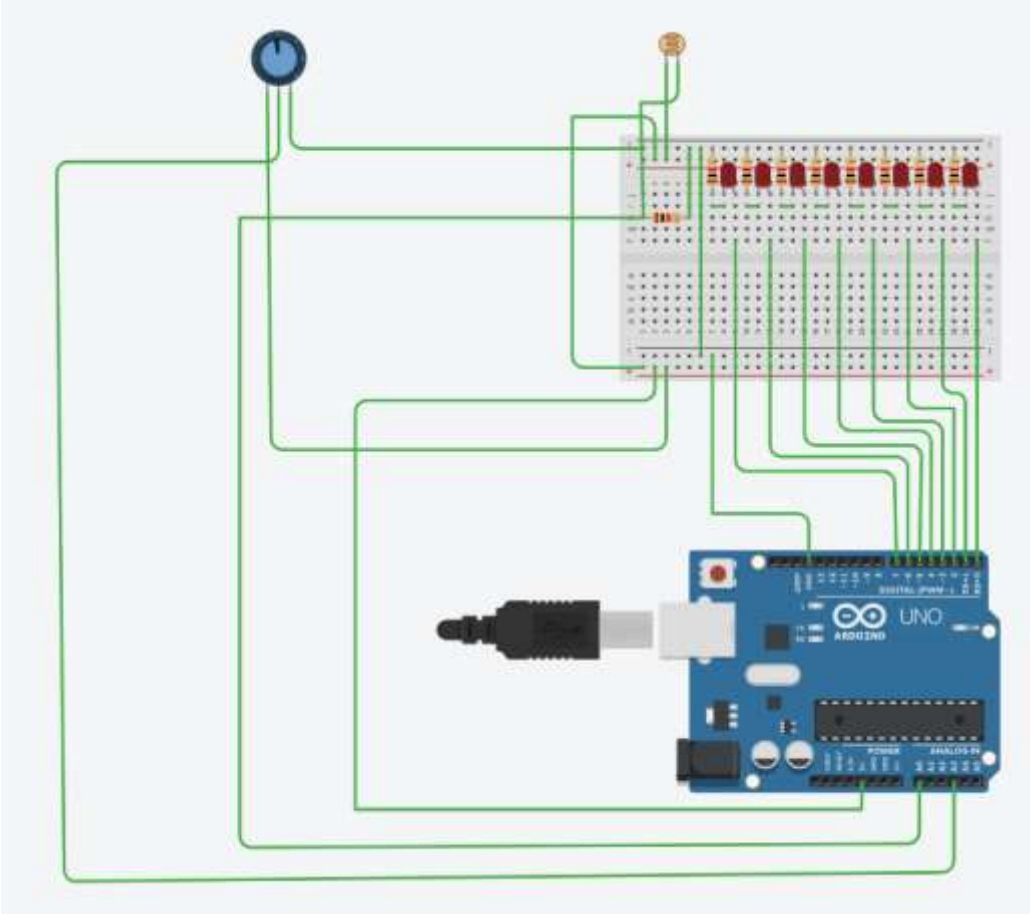
ADCSRA: OTHER IMPORTANT BIT FIELDS

- ADEN:
  - 1: Enable ADC
  - 0: Disable ADC
- ADSC:
  - In single conversion mode:
    - Writing 1: start a new conversion
    - Writing 0: No effect
    - Reading 1: conversion in progress
    - Reading 0: conversion completed.
  - In free running mode:
    - Write 1 only to start first conversion.
- ADATE
  - 1: Enable Auto-trigger

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

EXAMPLE 2

- Purpose: Using MUX of the on-chip ADC
- Specification: Build an AVR-based application with circuit as shown the figure.
- The microcontroller reads a threshold voltage from potentiometer.
- When the voltage signal from LDR crosses the threshold (i.e. Intensity of day light crosses a limit), 8 LEDS connected to PORTD are turned off.
- Otherwise, LEDs remain on.



PSEUDOCODE

```
Initialize ADC
Configure Port D
while(1)
	{
	Threshold =read data from pot channel
	sensor_value=read data from LDR channel
	if (sensor_value<Threshold)
		Turn on LEDs
	else
		Turn off LEDs
	}
```

```
// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
#include "pinDefines.h"

#define LED_DDR DDRD
#define LED_PORT PORTD
#define POT PC3
#define LIGHT_SENSOR PC0
uint16_t readADC(uint8_t channel)
{
ADMUX = (0xf0 & ADMUX) | channel;
ADCSRA |= (1 << ADSC);
loop_until_bit_is_clear(ADCSRA, ADSC);
return (ADC);
}

int main(void)
{
// -------- Inits --------- //
uint16_t lightThreshold;
uint16_t sensorValue;
// Set up ADC
ADMUX |= (1 << REFS0); /* reference voltage on AVCC */
ADCSRA |= (1 << ADPS1) | (1 << ADPS0); /* ADC clock prescaler /8 */
ADCSRA |= (1 << ADEN); /* enable ADC */

LED_DDR = 0xff;
// ------ Event loop ------ //
while (1)
        {
        lightThreshold = readADC(POT);
        sensorValue = readADC(LIGHT_SENSOR);
        if (sensorValue < lightThreshold)
                LED_PORT = 0xff;
        else
                LED_PORT = 0x00;
        } /* End event loop */
return 0; /* This line is never reached */
}
```

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FREE RUNNING MODE

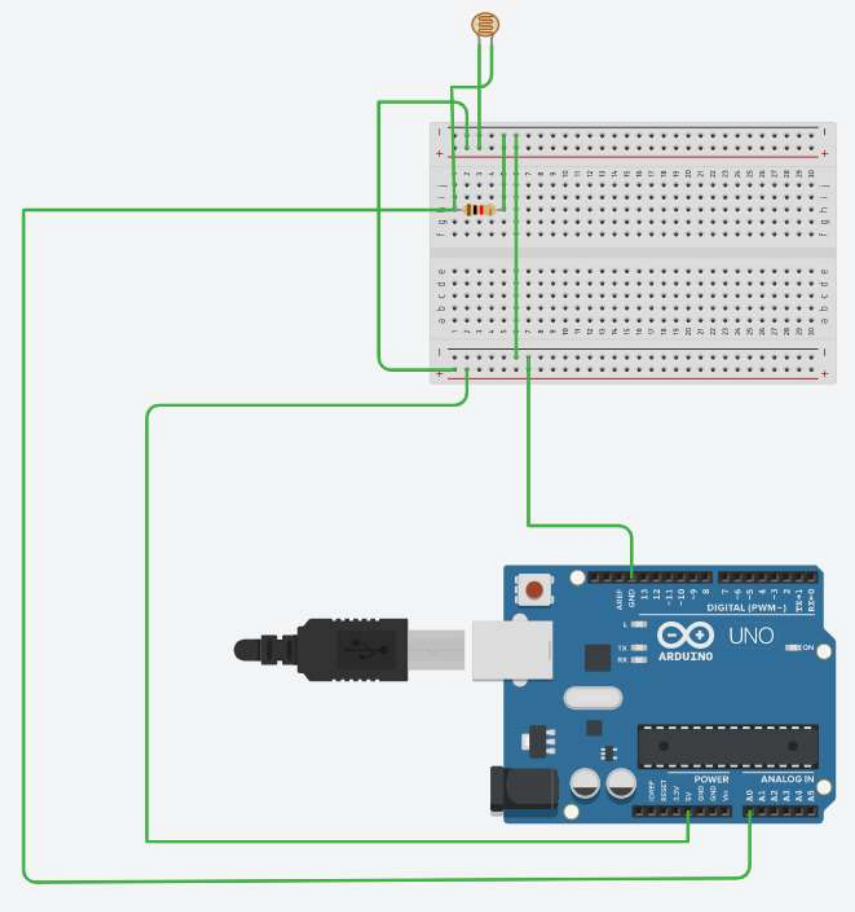
Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free running mode
0	0	1	Analog comparator
0	1	0	External interrupt request 0
0	1	1	Timer/Counter0 compare match A
1	0	0	Timer/Counter0 overflow
1	0	1	Timer/Counter1 compare match B
1	1	0	Timer/Counter1 overflow
1	1	1	Timer/Counter1 capture event

EXAMPLE 3

- Build an AVR based application where micrcontroller gets reading from an LDR and send the value to laptop through USART.
- Laptop runs a python script to draw the signal in real-time.



PSEDUOCODE

```
initialize USART
initialize ADC in free running mode
while (1)
      {
      read ADC value
      transmit it through USART
      }
```

```
// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
#include "pinDefines.h"
#include "USART.c"

#define SAMPLE_DELAY 200 /* ms, controls how closely the plot drawn by python script follows the changes in sensor signal. 200 looks ideal for light sensor */
// -------- Functions --------- //
static inline void initFreerunningADC(void)
{
ADMUX |= (1 << REFS0); /* reference voltage on AVCC */
ADCSRA |= (1 << ADPS1) | (1 << ADPS0); /* ADC clock prescaler /8 */
ADMUX |= (1 << ADLAR); /* left-adjust result*/
ADCSRA |= (1 << ADEN); /* enable ADC */
ADCSRA |= (1 << ADATE); /* auto-trigger enable */
ADCSRA |= (1 << ADSC); /* start first conversion */
}

int main(void)
{
// -------- Inits --------- //
initUSART();
initFreerunningADC();
// ------ Event loop ------ //
while (1)
        {
        transmitByte(ADCH); /* transmit the high byte, left-adjusted */
        _delay_ms(SAMPLE_DELAY);
        } /* End event loop */
return 0; /* This line is never reached */
}
```