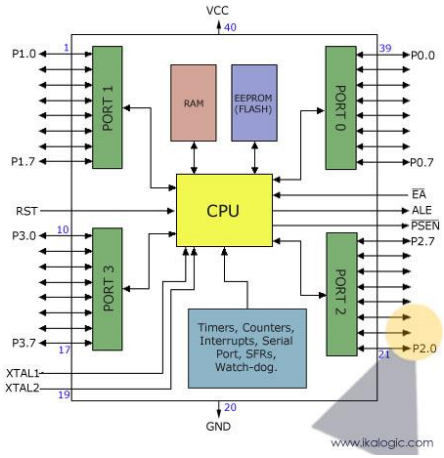
Embedded Systems Design, Spring 2025

Lecture 9



AD and DA conversion

Outline

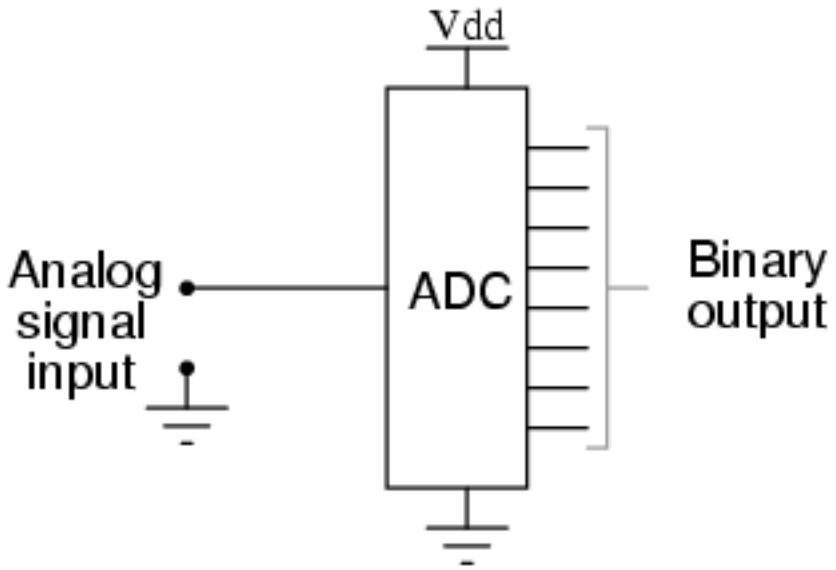
- Introduction to ADC and DAC
- ADC
- DAC
- Examples:
  - ADC with atmega328
  - DAC with PCF8591 (over I2C)

Introduction

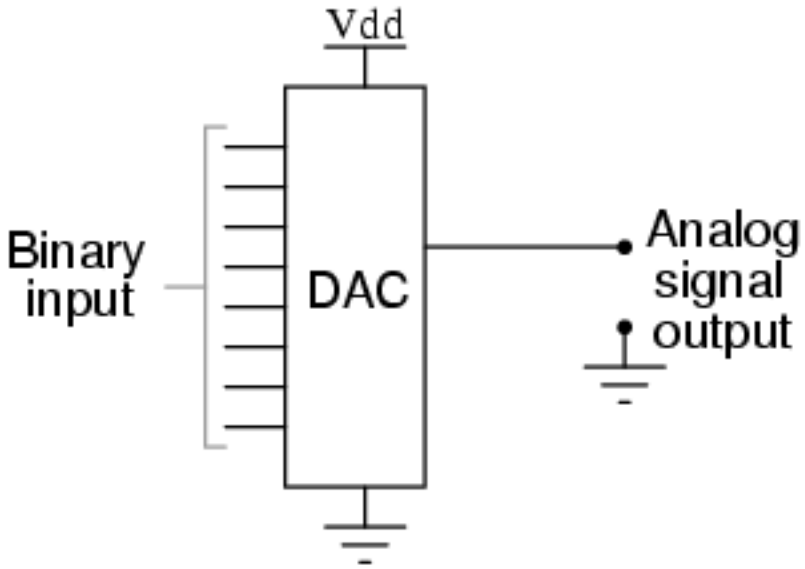
- Microcontrollers need to connect to peripherals that work in the analog domain, therefore a way of interfacing is required:
  - analog (voltage level) – digital (binary values)

- An *analog-to-digital converter*, or ADC, performs the conversion from analog domain to binary while
- a *digital-to-analog converter*, or DAC, performs the reverse operation.

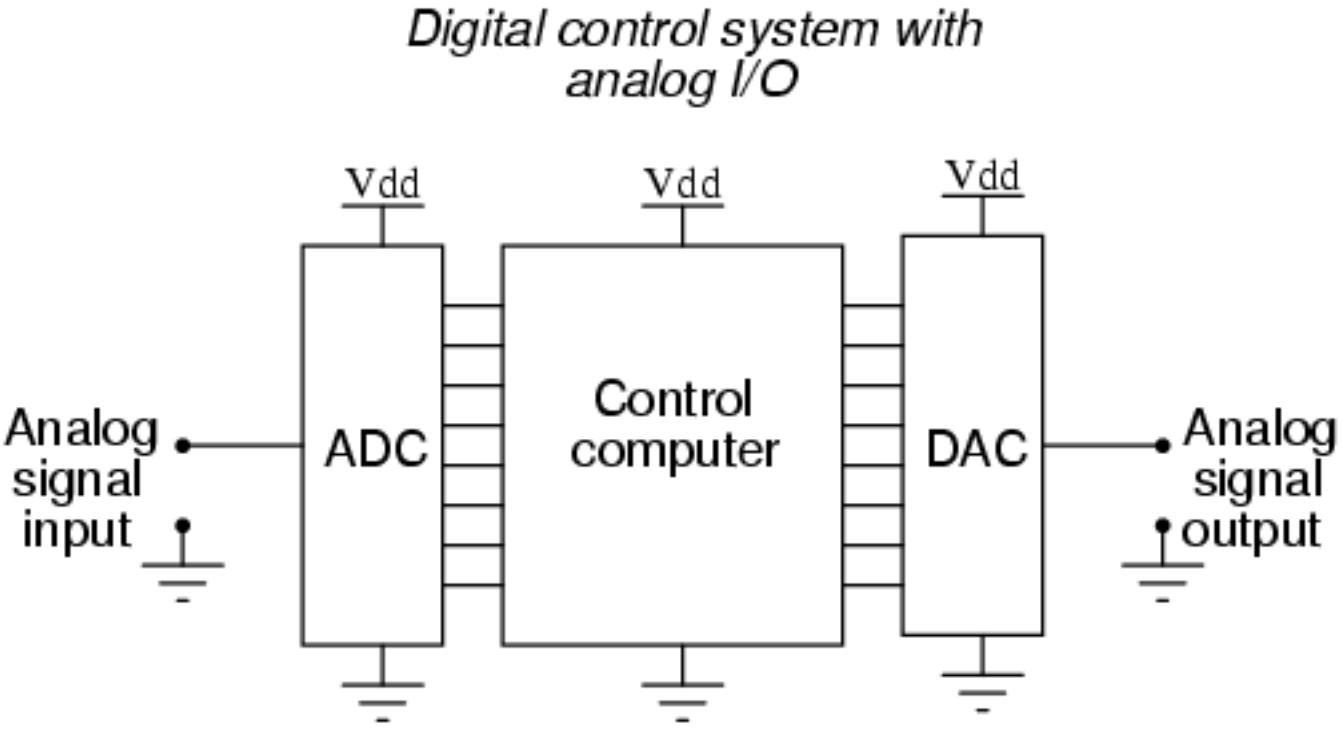
ADC



DAC

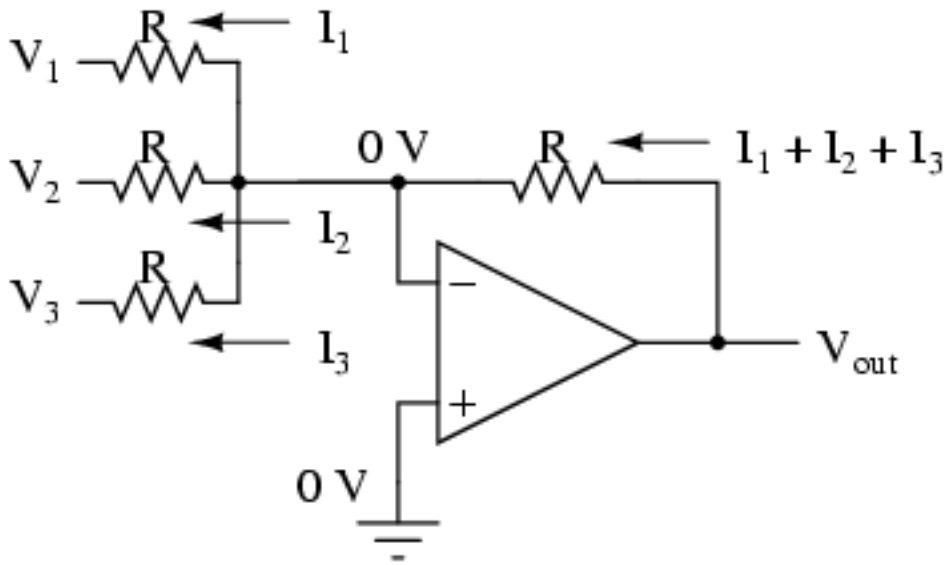


System with AD/DA



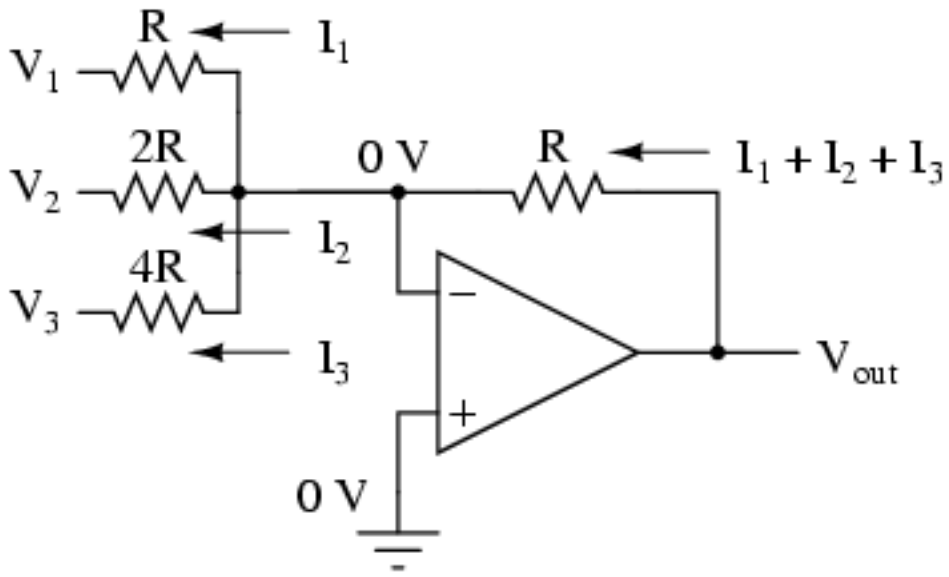
Binary weighted DAC

Inverting summer circuit



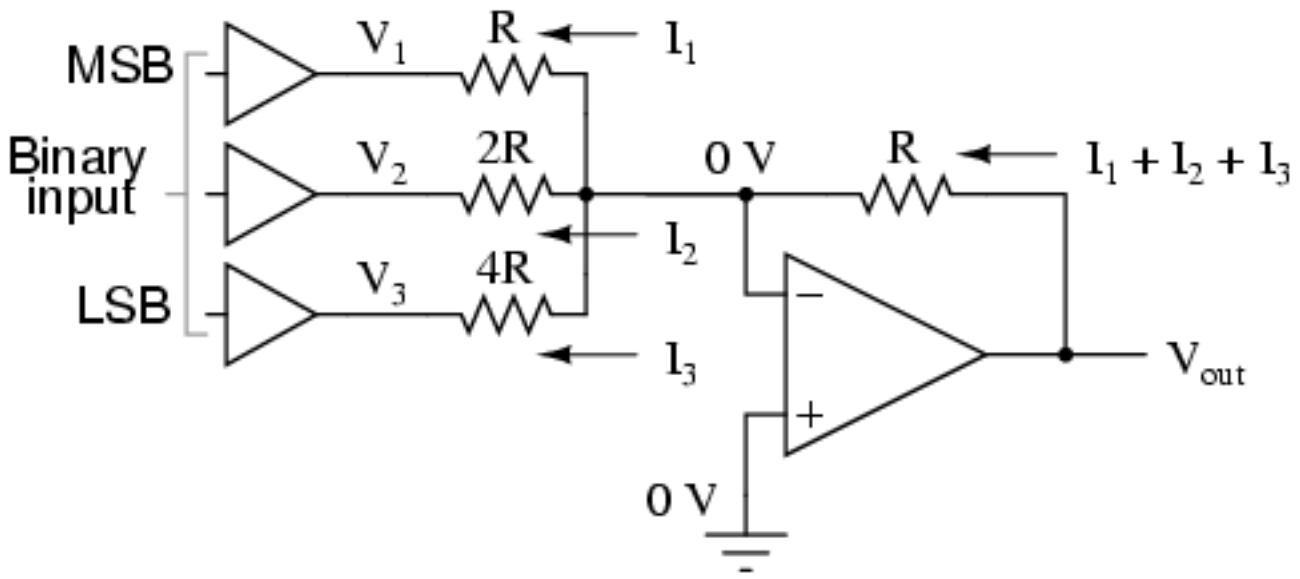
$$V_{out} = - (V_1 + V_2 + V_3)$$

Using different R



$$V_{out} = -\left(V_1 + \frac{V_2}{2} + \frac{V_3}{4}\right)$$

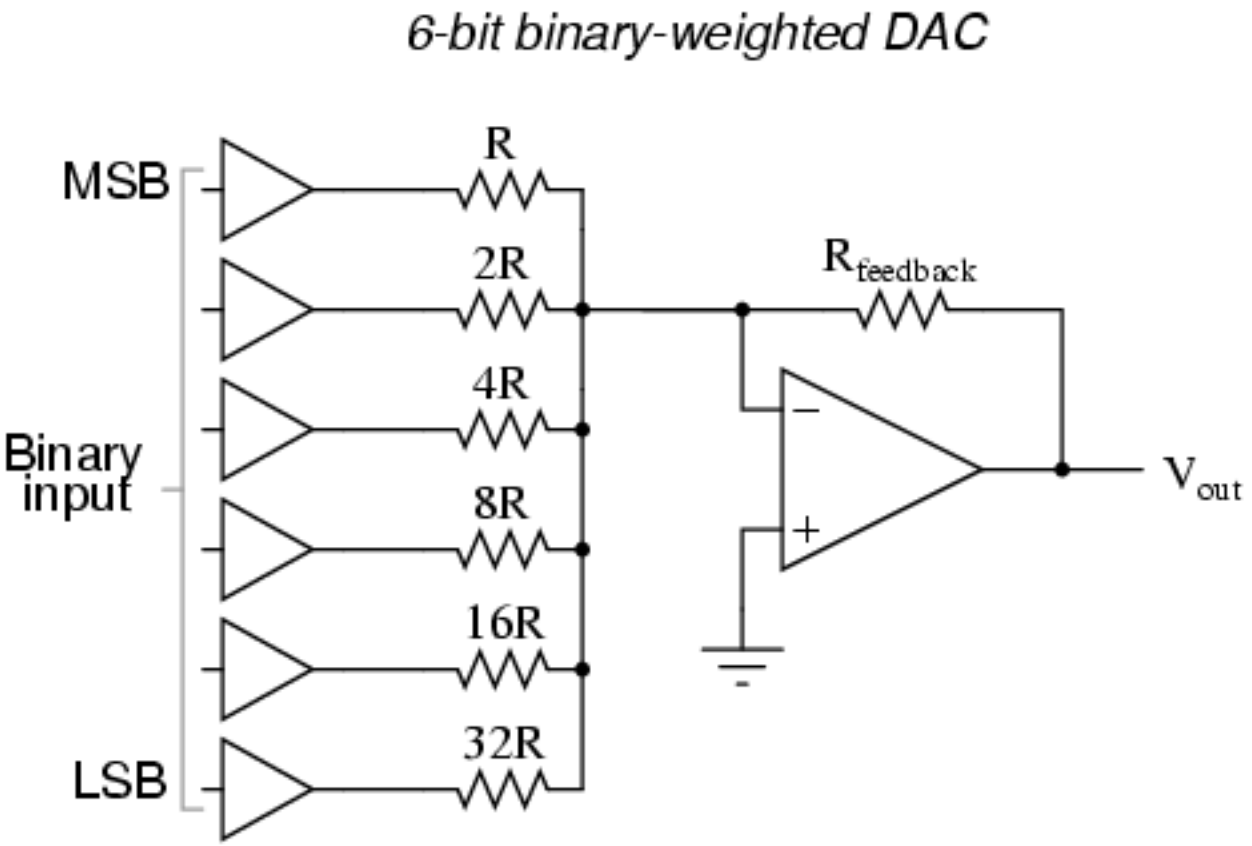
Using a binary input



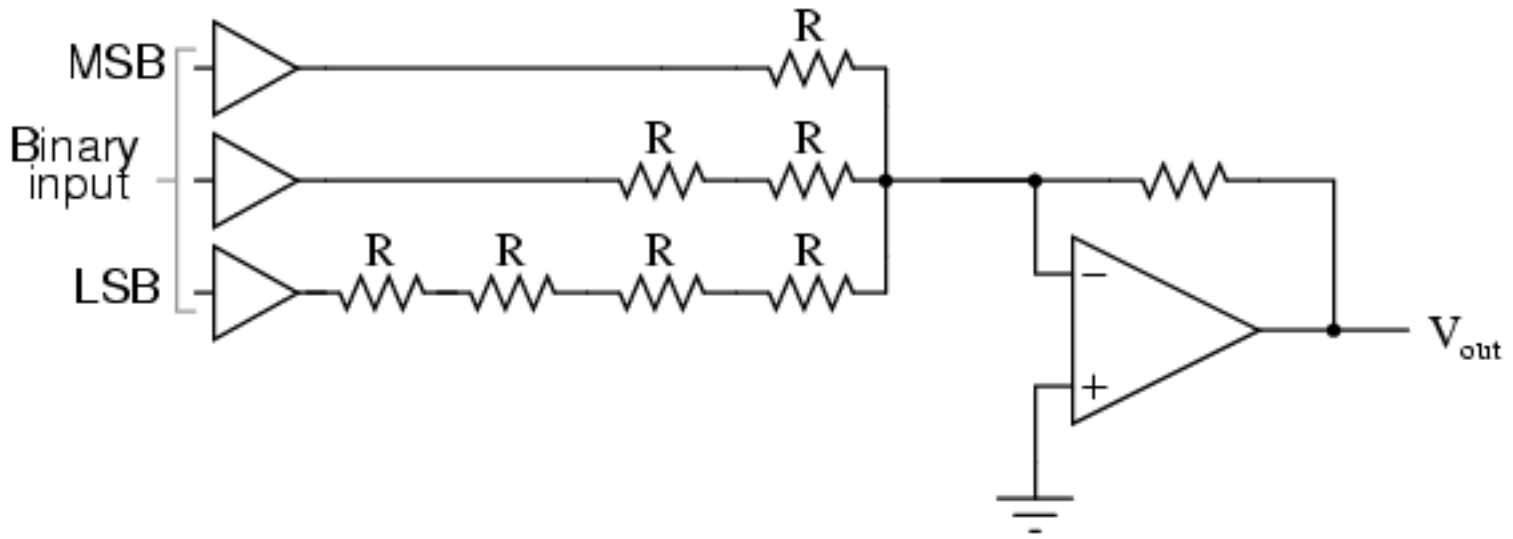
V_{output} for the previous case

- | Binary | Output voltage |
- | 000 | 0.00 V |
- | 001 | -1.25 V |
- | 010 | -2.50 V |
- | 011 | -3.75 V |
- | 100 | -5.00 V |
- | 101 | -6.25 V |
- | 110 | -7.50 V |
- | 111 | -8.75 V |

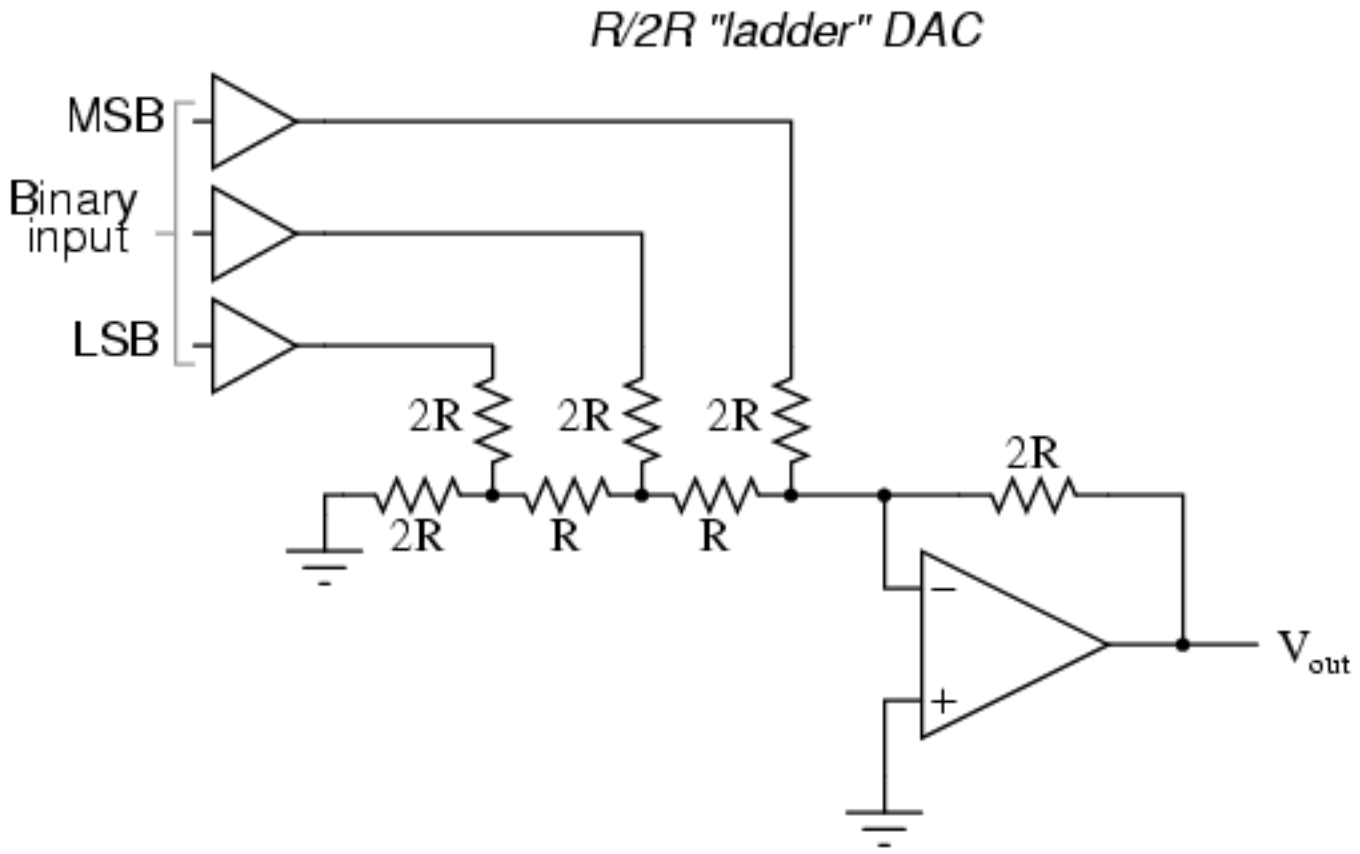
Increasing the resolution



Another approach?



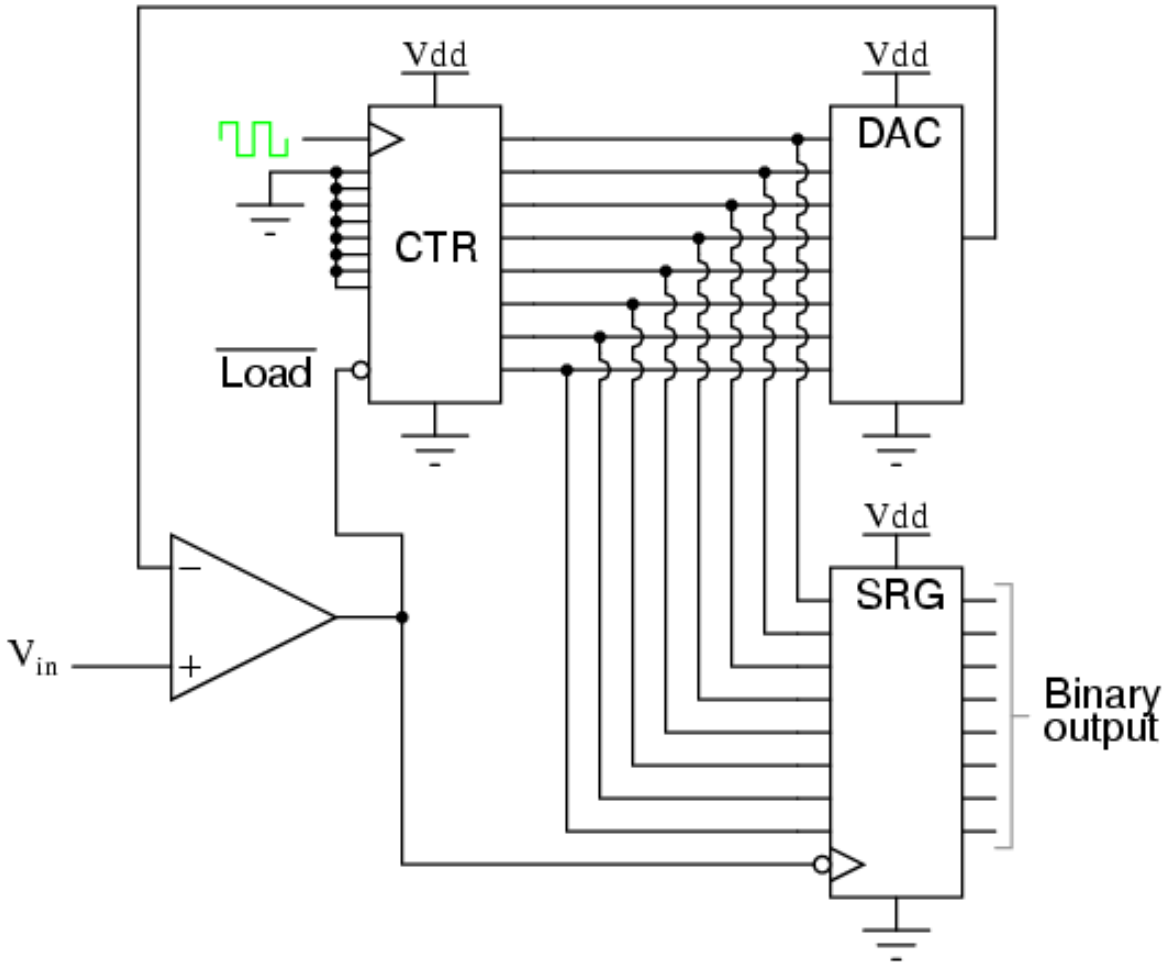
Reducing the number of components



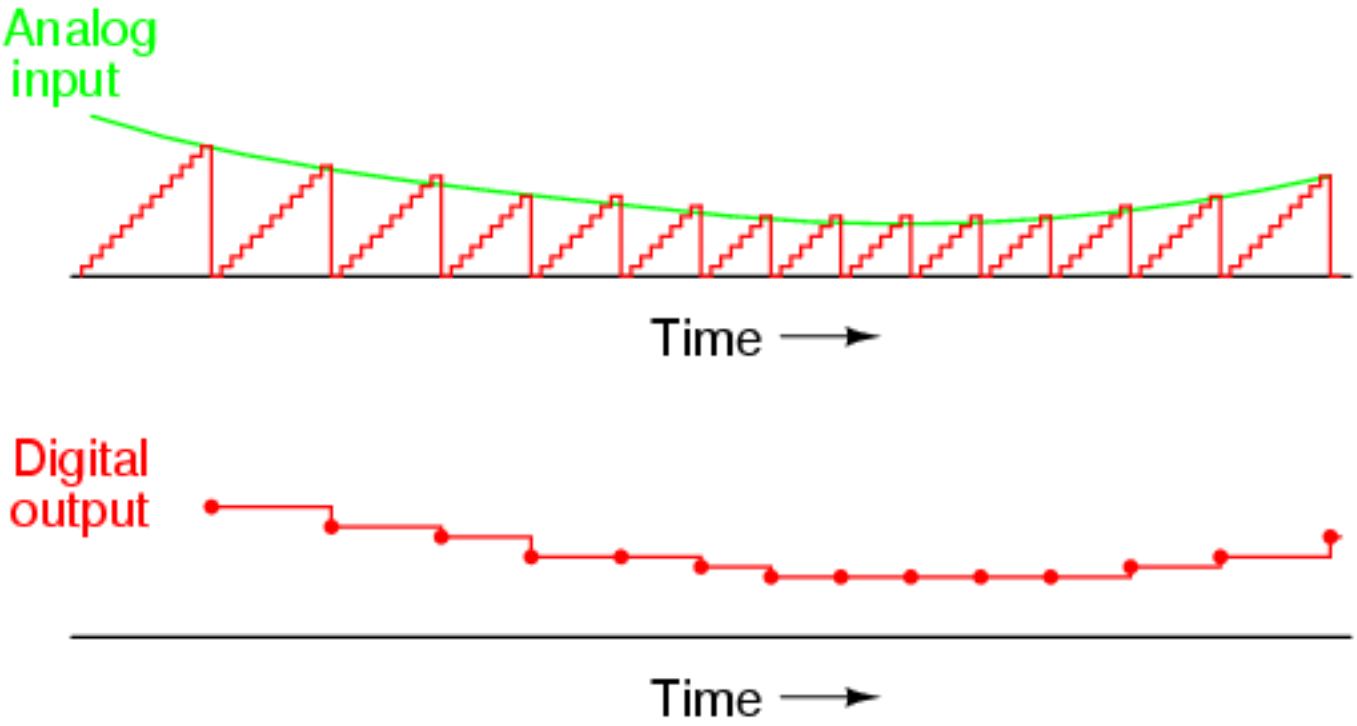
V_{output} for the previous case

- | Binary | Output voltage |
- | 000 | 0.00 V |
- | 001 | -1.25 V |
- | 010 | -2.50 V |
- | 011 | -3.75 V |
- | 100 | -5.00 V |
- | 101 | -6.25 V |
- | 110 | -7.50 V |
- | 111 | -8.75 V |

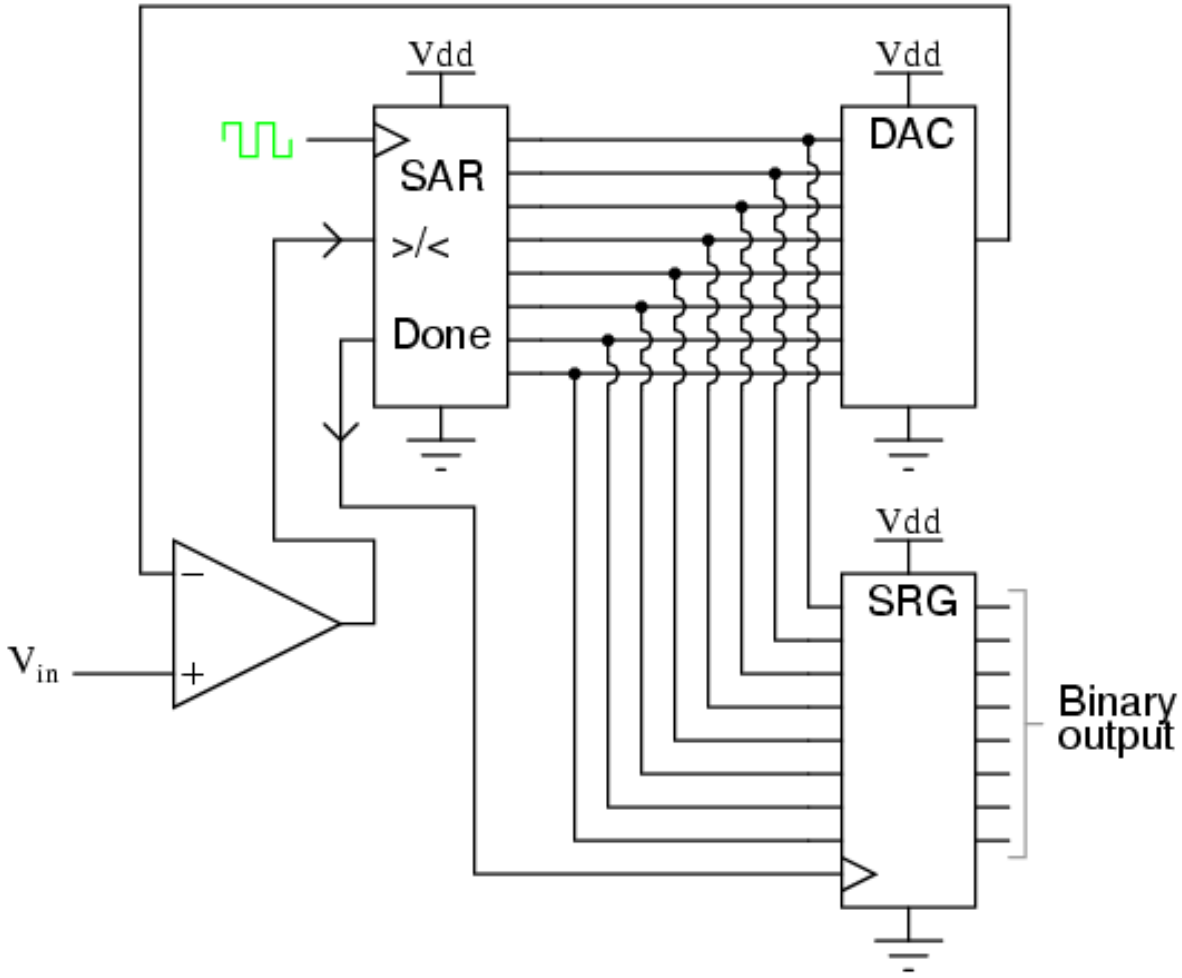
Analog to digital: ramp (counter)



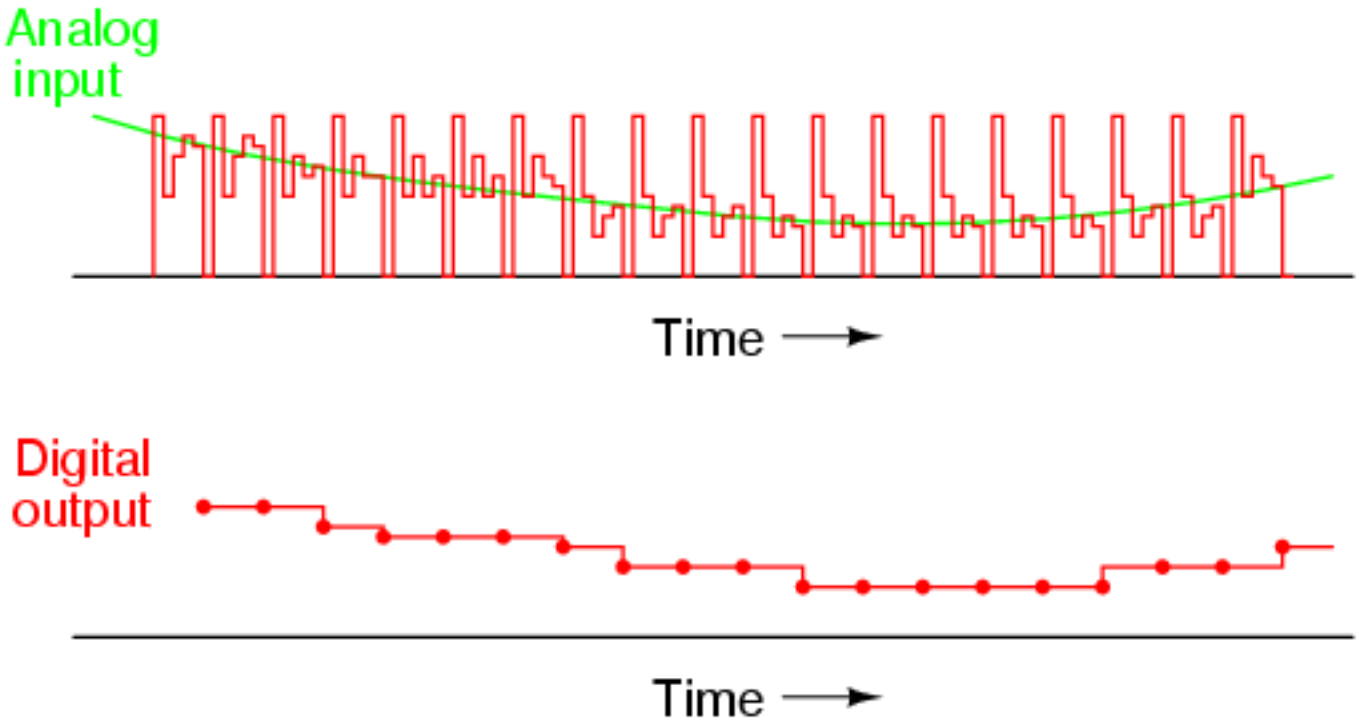
Analog Input vs Digital Output



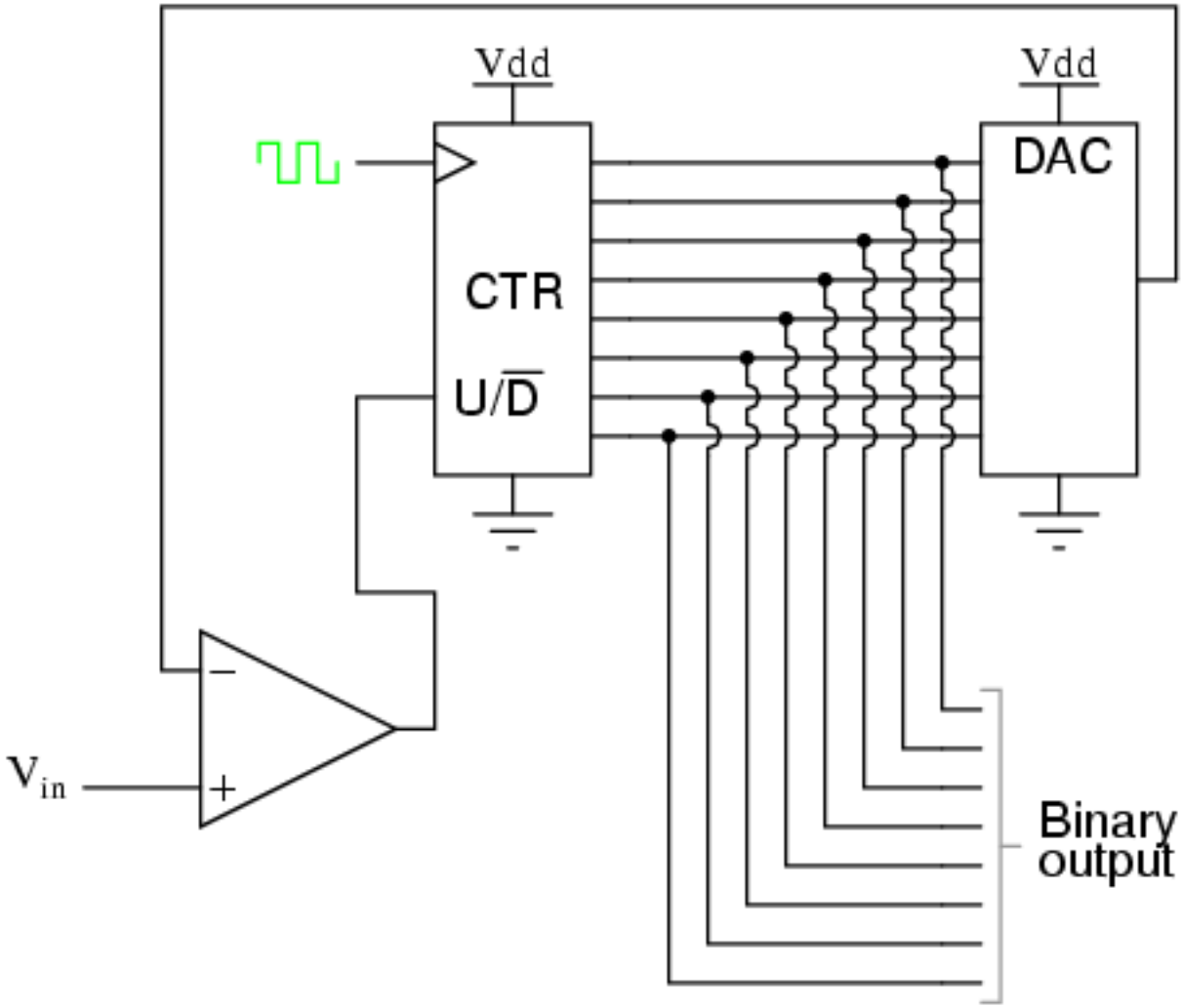
Improving the counter time



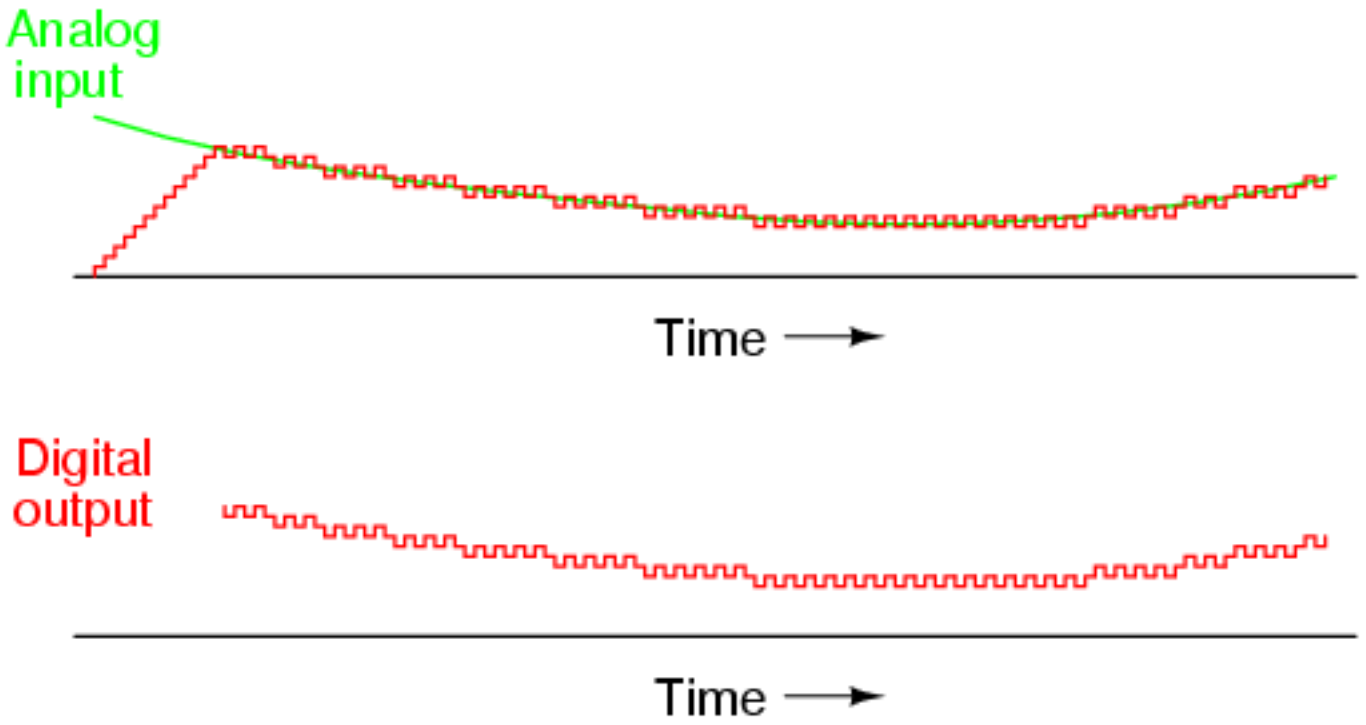
Analog input vs Digital output



Tracking ADC

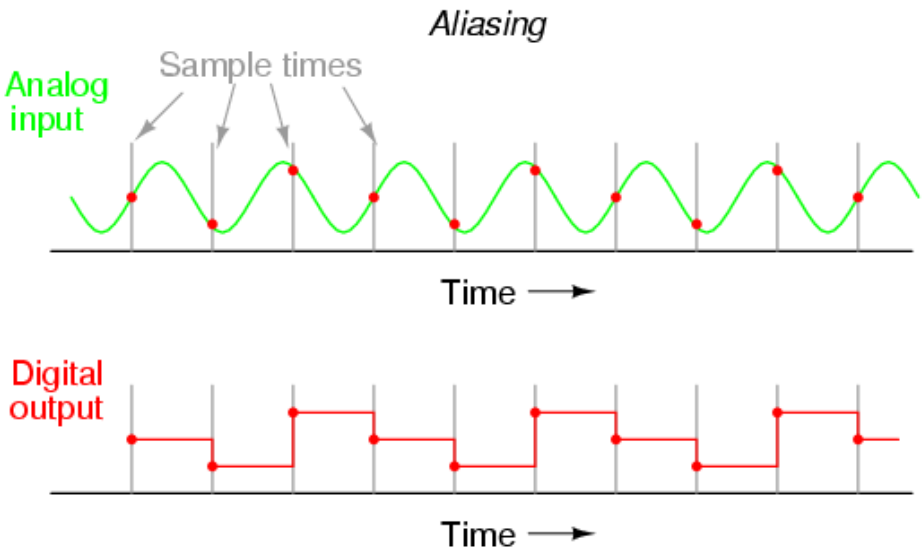


Analog Input vs Digital Output



Other considerations

- Resolution:
  - number of binary bits from the output
  - Total number of possible levels
- Sample period
  - Short: good for rapid changing signals
  - Long: good for slow changing signals
- Aliasing



ADC for atmega328

- 10 bit resolution
- 13 – 260 us conversion time (up to 15ksps max resolution)
- 6 single ended input channels
- Free running or single conversion
- Interrupt on ADC complete
- Multiple trigger sources

Modes

- The ADC can perform a single conversion (single conversion mode)

- In free running mode, the ADC uses the ADC Interrupt Flag from a previous conversion to start a new one as soon as the old one has finished.
- First conversion is started by:
  - Write a 1 to ADSC bit from ADCSRA register

- Reading ADSC bit can tell you if a conversion is in progress

Other settings

- Input channels:
  - The user has to select the corresponding input channel before a new conversion
- VREF can be selected as either AVCC, internal 1.1V reference, or external AREF pin
- The result will be a 10 bit result with max value VREF.
- It will be stored in ADCL and ADCH (each 8 bit registers)

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

Registers

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	

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

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8[1]

Registers

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 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

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	

- **Bits 2:0 – ADPS[2:0]: ADC Prescaler Select Bits**

These bits determine the division factor between the system clock frequency and the input clock t

Table 24-5. ADC Prescaler Selections

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

Example code

```
#include <avr/io.h>
#include <stdio.h>
#include <stdint.h>
#define ADC_PIN 0 //what is the ADC channel we'll use
//function prototypes
uint16_t adc_read(uint8_t adc_channel);

int main(){
        uint16_t adc_result;
        // Select Vref = AVcc
        ADMUX = (1<<REFS0);
        //set prescaler to 128 and turn on the ADC module
        ADCSRA = (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADEN);
        while(1){
           adc_result = adc_read(ADC_PIN);
           //do something with this result…
        }
}
// continues on next page
```

```
uint16_t adc_read(uint8_t adc_channel){
        ADMUX &= 0xf0; // clear any previously used channel, but
keep internal reference
        ADMUX |= adc_channel; // set the desired channel
        //start a conversion
        ADCSRA |= (1<<ADSC);

        // now wait for the conversion to complete
        while ( (ADCSRA & (1<<ADSC)) );

        // now we have the result, so we return it to the calling
function as a 16 bit unsigned int
        return ADC;

}
```

I2C Bus

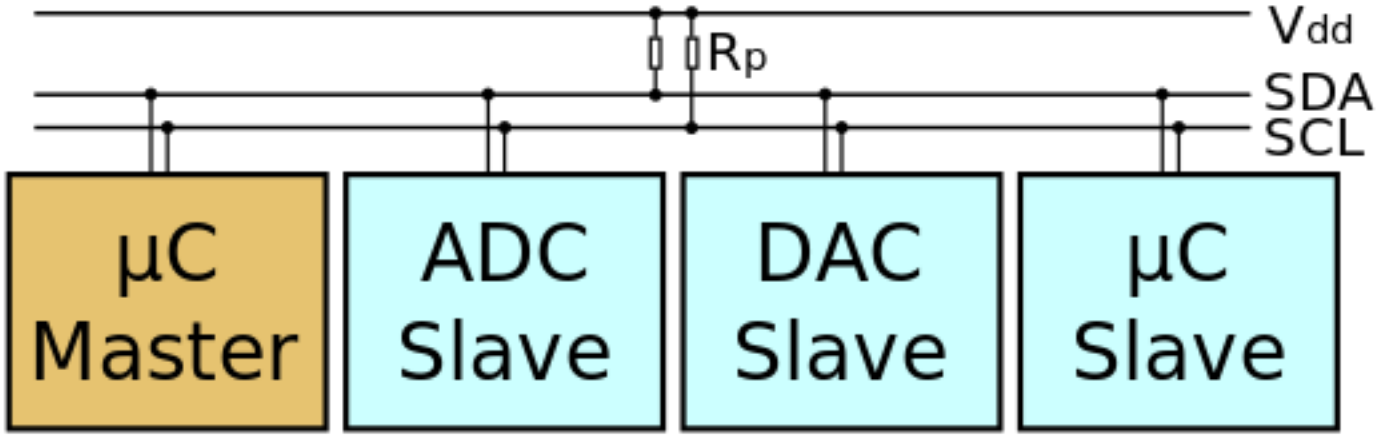
- The I2C bus was designed by Philips in the early '80s to allow easy communication between components which reside on the same circuit board.
- The name I2C translates into "Inter IC". Sometimes the bus is called IIC or I²C bus.
- The original communication speed was defined with a maximum of 100 kbits per second and many applications don't require faster transmissions.
- For those that do, there is a 400 kbit fastmode and - since 1998 - a high speed 3.4 Mbit option available.

Characteristics

- Most significant features include:
  - Only two bus lines are required
  - No strict baud rate requirements like for instance with RS232, the master generates a bus clock
  - Simple master/slave relationships exist between all components
    Each device connected to the bus is software-addressable by a unique address
  - I2C is a true multi-master bus providing arbitration and collision detection

Sample schematic

- I²C uses only two bidirectional open-drain lines,
  - Serial Data Line (SDA) and
  - Serial Clock (SCL), pulled up with resistors.



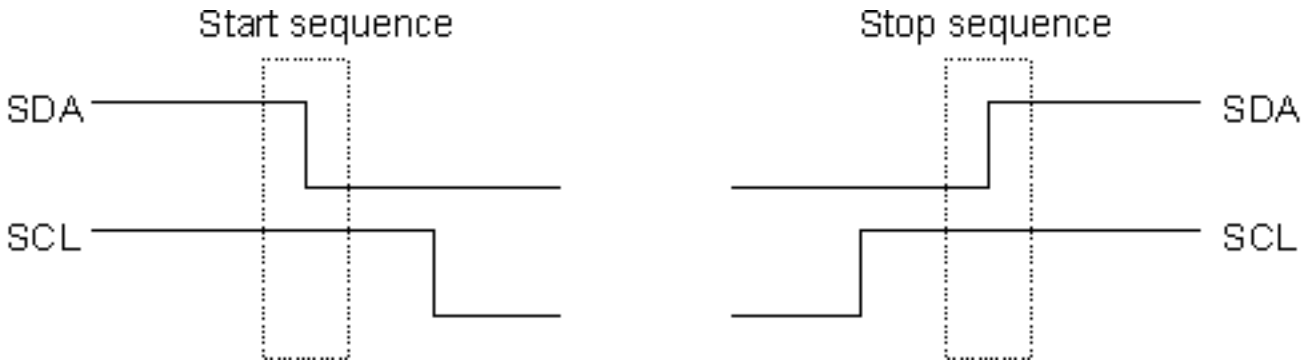
Roles

- The bus has two roles for nodes: master and slave:
  - Master node — node that generates the clock and initiates communication with slaves
  - Slave node — node that receives the clock and responds when addressed by the master

- Both master and slave can transfer data over the I2C bus, but that transfer is always controlled by the master.

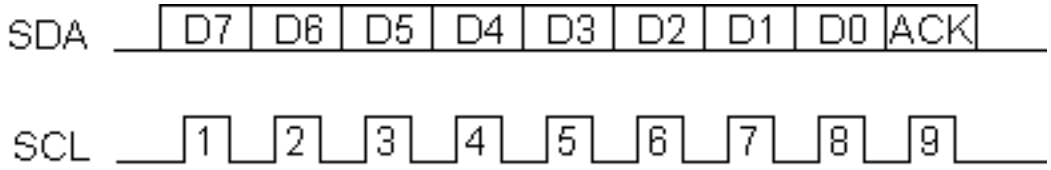
Start-Stop sequences

- The start and stop sequences mark the beginning and end of a transaction with the slave device.
- They are special, as SDA is changed while SCL is high



Data transfer

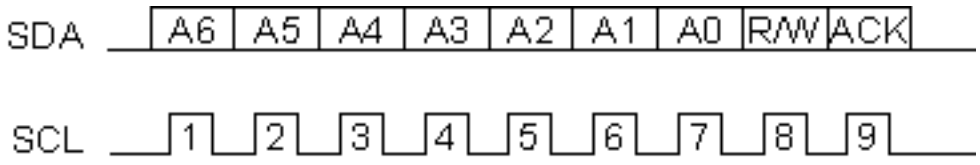
- Data is transferred in 8 bit sequences
- Bits are placed on the SDA line, with MSB first



- For every 8 bits transferred, the device receiving the data sends back an acknowledge bit
  - If the receiving device sends back a low ACK bit, then it has received the data and is ready to accept another byte.
  - If it sends back a high then it is indicating it cannot accept any further data and the master should terminate the transfer by sending a stop sequence.

Device Addressing

- Most chips you will use will have 7 bit addresses.
- This means that you can have up to 128 devices on the I2C bus
- When sending out the 7 bit address, we still send 8 bits.
- The extra bit is used to inform the slave if the master is writing to it or reading from it.
  - If the bit is zero the master is writing to the slave.
  - If the bit is 1 the master is reading from the slave.



Writing to a slave

- So to write to a slave device:

1. Send a start sequence
2. Send the I2C address of the slave with the R/W bit low (even address: write to slave)
3. Send the internal register number you want to write to
4. Send the data byte
5. [Optionally, send any further data bytes]
6. Send the stop sequence.

Reading from a slave

- 1. Send a start sequence
  2. Send the slave address with the R/W bit low (even address)
  3. Send 0x01 (Internal address of the register you want to read)
  4. Send a start sequence again (repeated start)
  5. Send the slave address with the R/W bit high (odd address)
  6. Read data byte from slave
  7. Send the stop sequence.

Complete data transfer

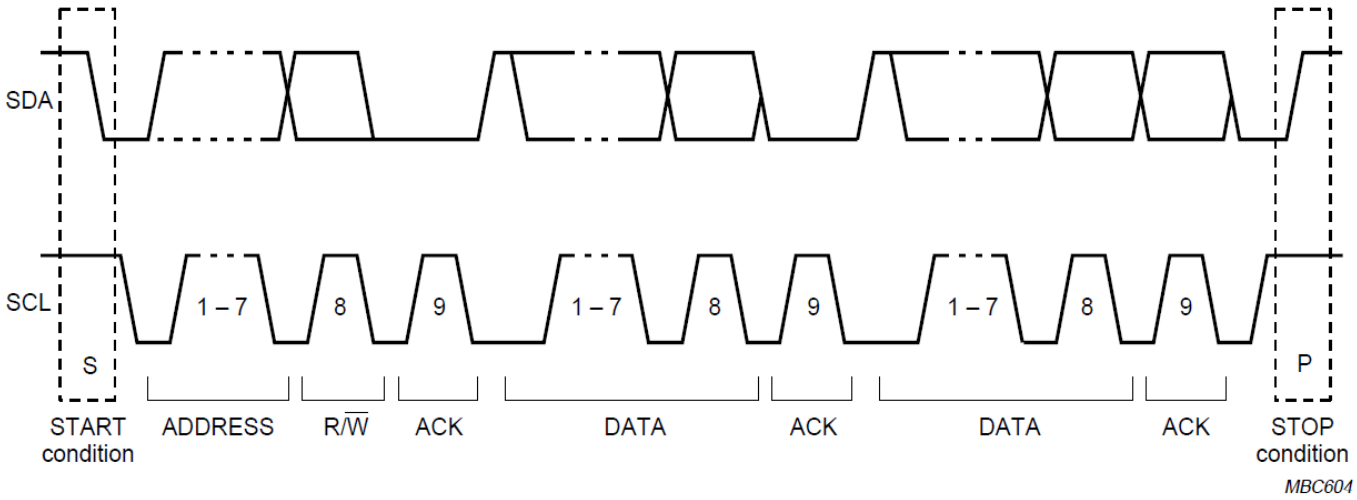


Fig.10 A complete data transfer.

PCF8591 AD-DA chip

5 BLOCK DIAGRAM

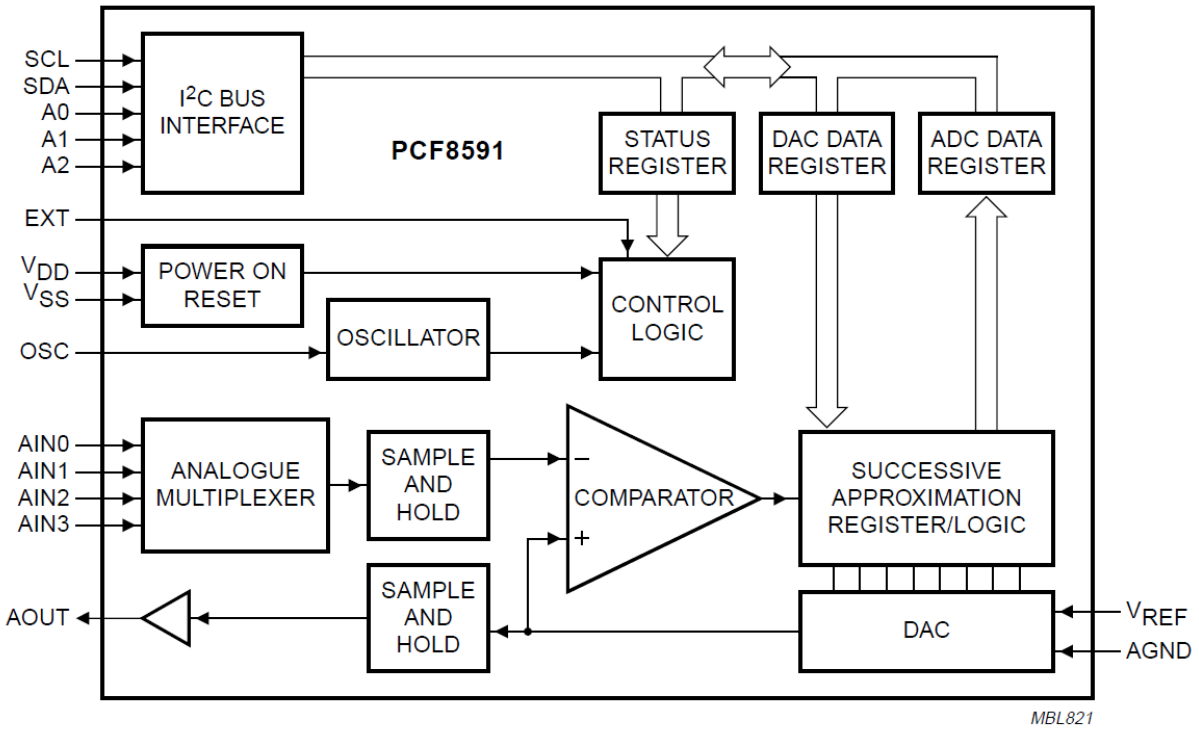


Fig.1 Block diagram.

Write mode, D/A conversion

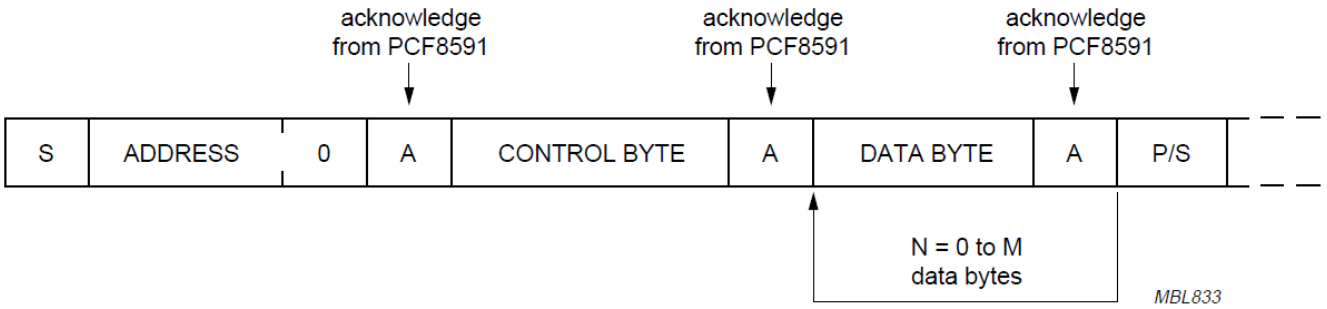
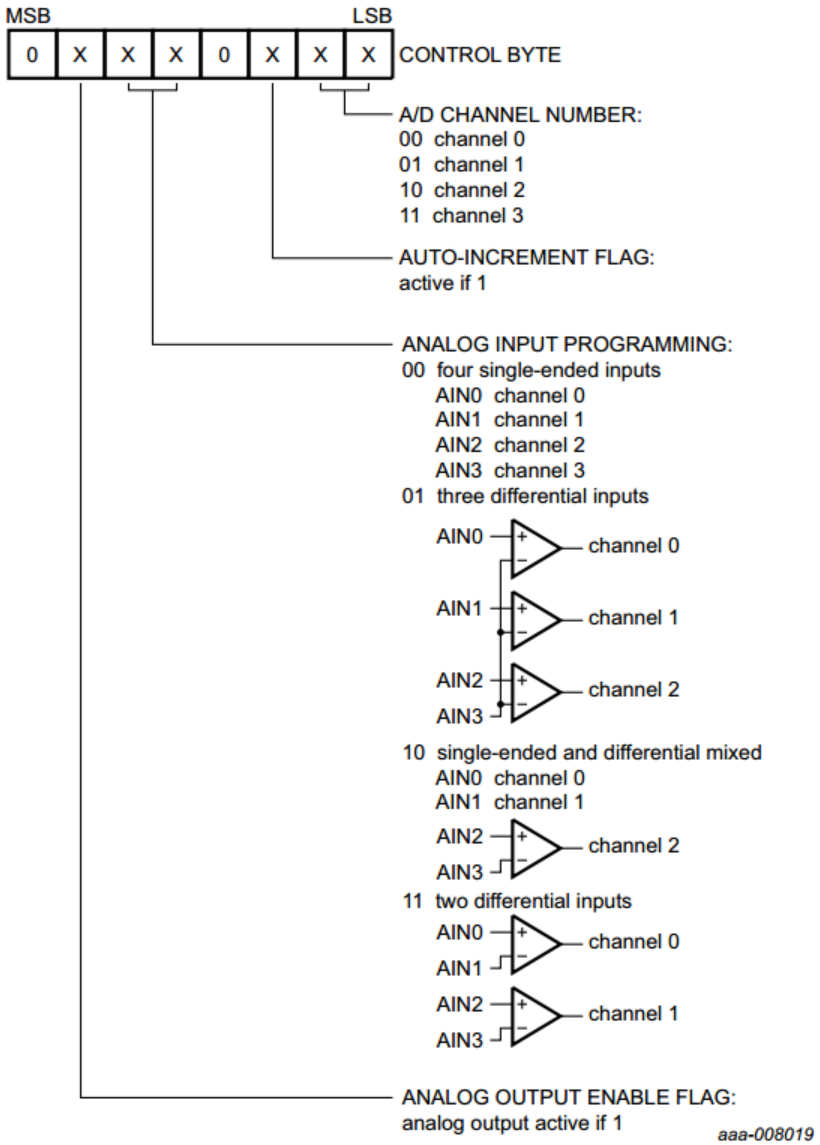


Fig.16 Bus protocol for write mode, D/A conversion.

Contents of Control Byte



Read mode, A/D conversion

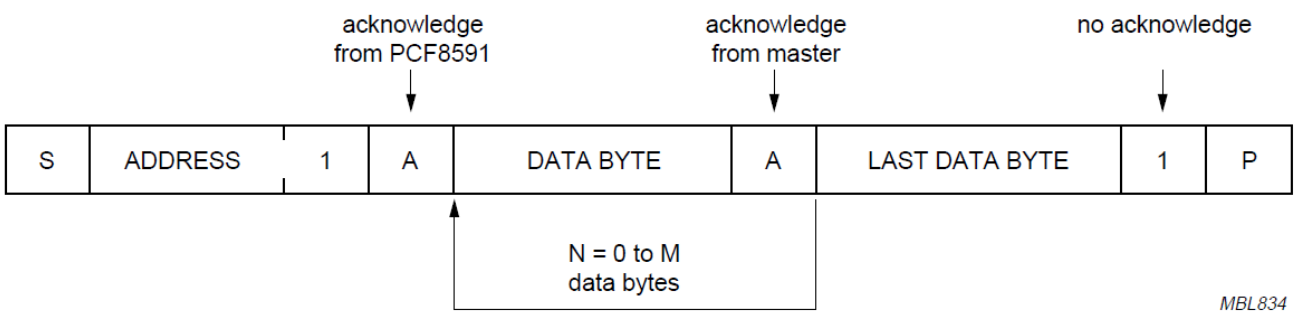
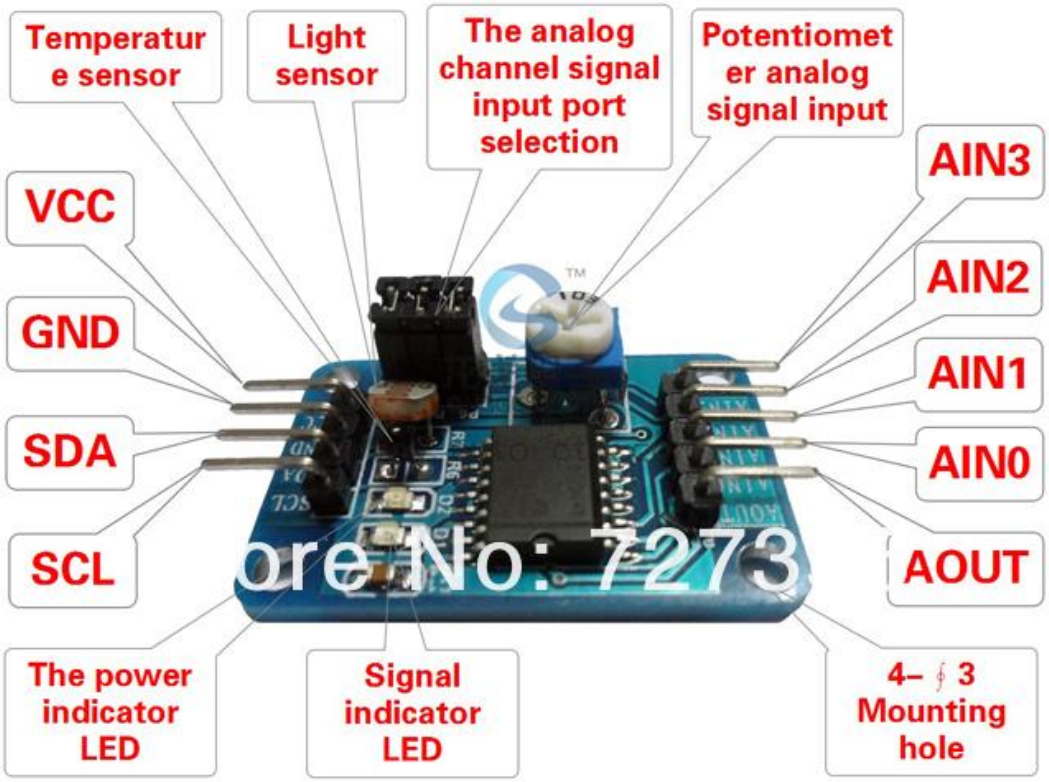


Fig.17 Bus protocol for read mode, A/D conversion.

Setting up I2C with the PCF8591 AD/DA YL-40 module



Create a new header file called pcf8591.h and add it to your project

```
#ifndef PCF8591_H_INCLUDED
#define PCF8591_H_INCLUDED
//PCF8591 address (a2 = a1 = a0 = 0)
#define PCF8591_adr 0x??
//function headers
void set_value(unsigned char target_value);
void pcf8591_init(void);
#endif
```

Create a new source file called pcf8591.c and add it to your project

```
#include "i2cmaster.h"
#include "pcf8591.h"

void pcf8591_init(){
        //initialize the i2c communication
        i2c_init();
}
void set_value(unsigned char target_value) {
        // address the DAC module and let it know that you will write
to it   i2c_start_wait(PCF8591_adr + I2C_WRITE);
        // write the control byte to enable Digital to Analog
Conversion      i2c_write(0x??);
        // write the desired value(8bit) received as parameter
        i2c_write(target_value);
        // stop the communication on the bus
        i2c_stop();
}
```

Find this value! (slide 42)

In your main.c file make sure you include the header file and call the init functions and the set_value

```
#define F_CPU 16000000UL
#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>
#include "i2cmaster.h"
#include "pcf8591.h"
int main(void) {
        DDRB = (1<<PORTB5); // use the LED on the board
        pcf8591_init();// initialize communication to DA converter
        int a;
        while(1) {
                for (a=0; a<250; a=a+10){
                        _delay_ms(100);
                        set_value(a);
                }
        }
}
```