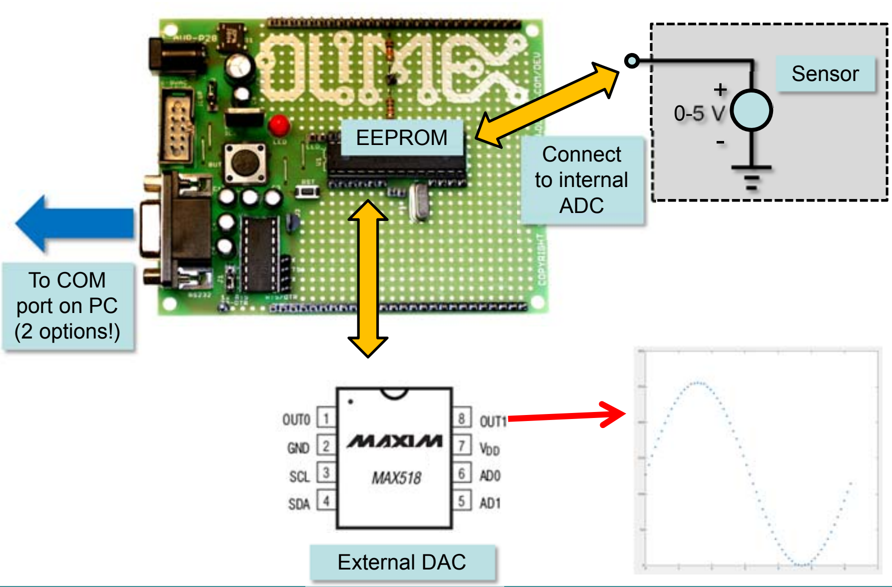
Embedded Systems

Lab 5 Considerations

Big Picture



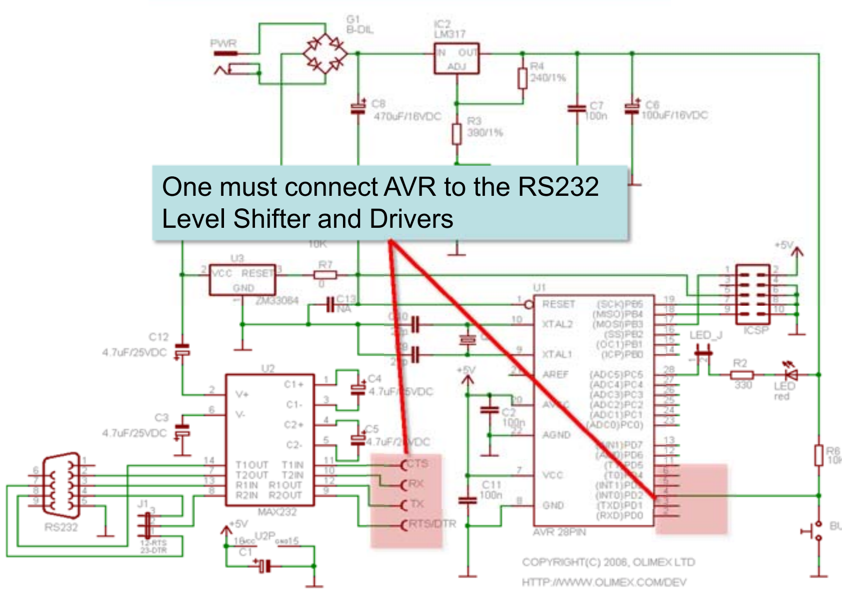
Lab5 - Where to Start?

- **Get HW for Lab5:** *MAX518, (USB2RS232 adapter), adjustable resistor, …*
- **RS232–based communication with PC**
  - **Establish connections between ATmega88 board & converter (2 options!)**
  - **Configure terminal software on PC**
  - **Write a test program (e.g., echo) to see if the USART on the μC is correctly configured**

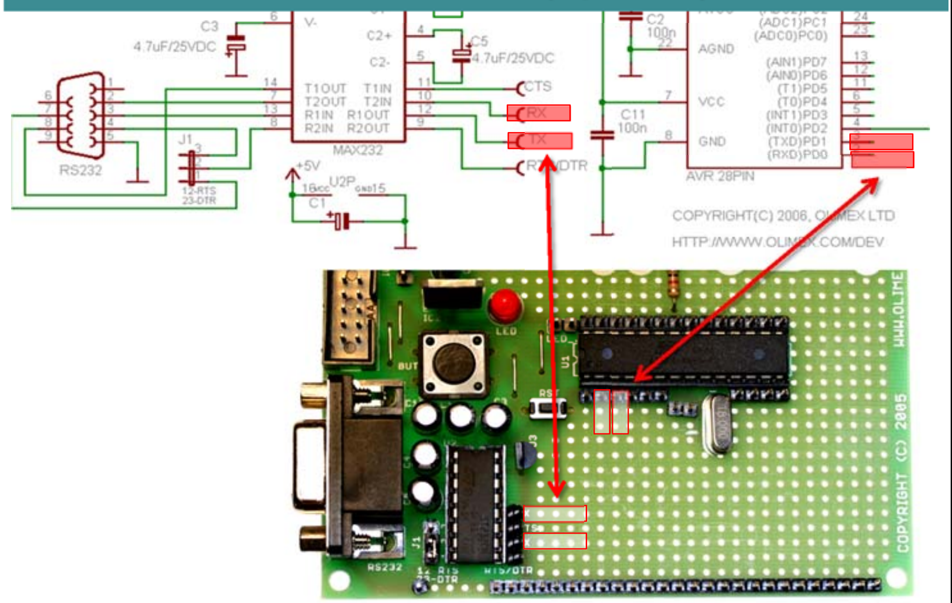
    USART initialization, print char, send char, … you can use libraries …

    Use the oscilloscope for troubleshooting (timing, …)
- **Start with the main program**
  - **Consider a finite-state machine (FSM) for parsing commands**
  - **Several options for string2number and number2string conversion**
  - **Implement ADC function (read section in ATmega88 documentation!)**
  - **Implement EEPROM storage and retrieval**
  - **Implement DAC I2C communication (upcoming lecture notes, MAX518 datasheet, …)**
  - **Can use library for I2C communication**
  - …

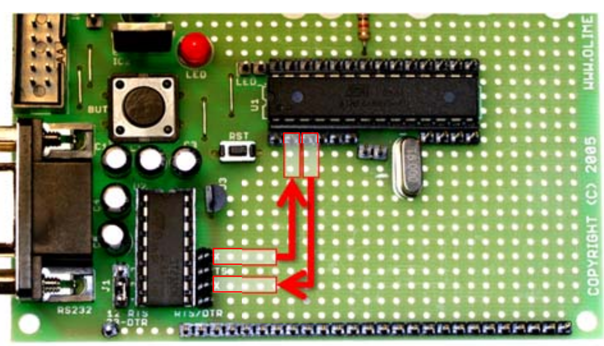
Serial Connection (Option A)



Serial Connection (Option A)



Serial Connection (Option A)

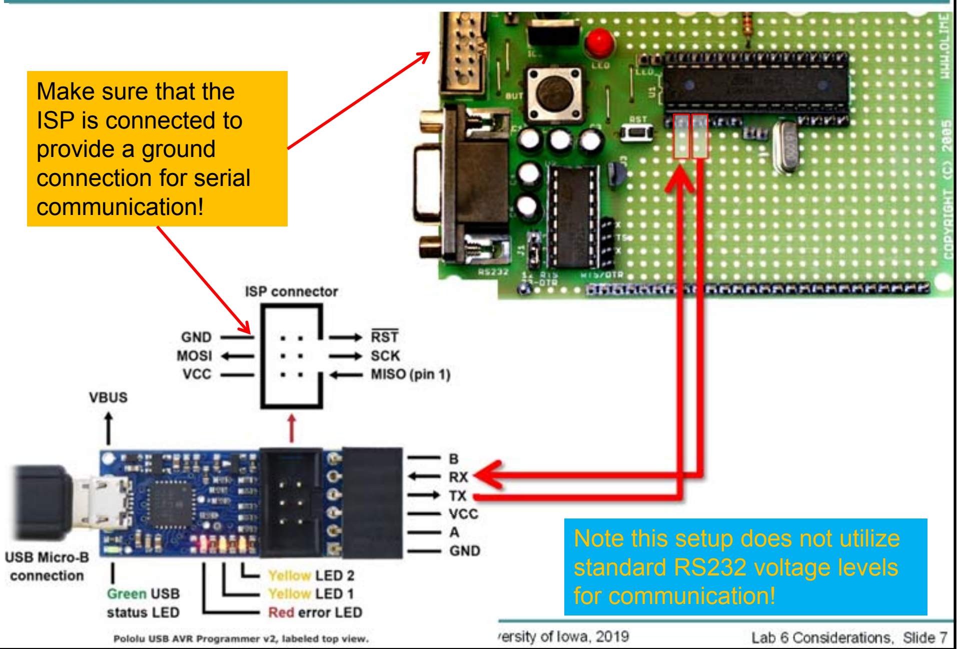


To COM port on PC

Leave other pins and jumpers alone: we are NOT using flow control

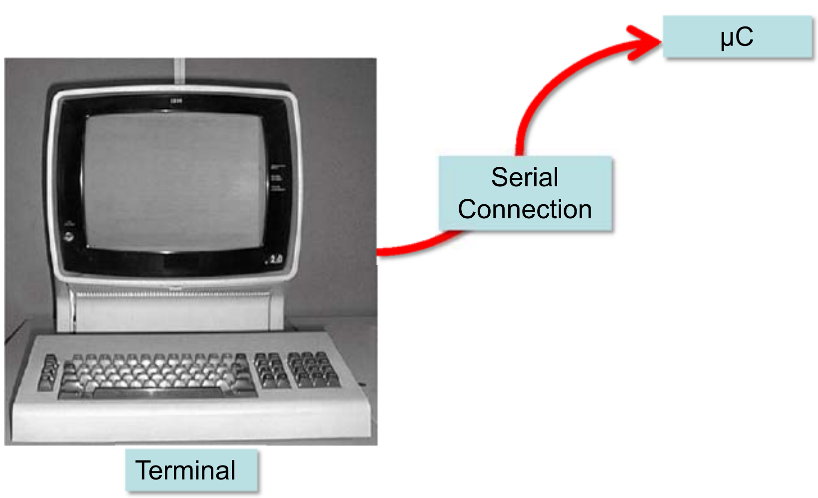
Serial Connection with Pololu AVR Programmer (Option B)

Make sure that the ISP is connected to provide a ground connection for serial communication!

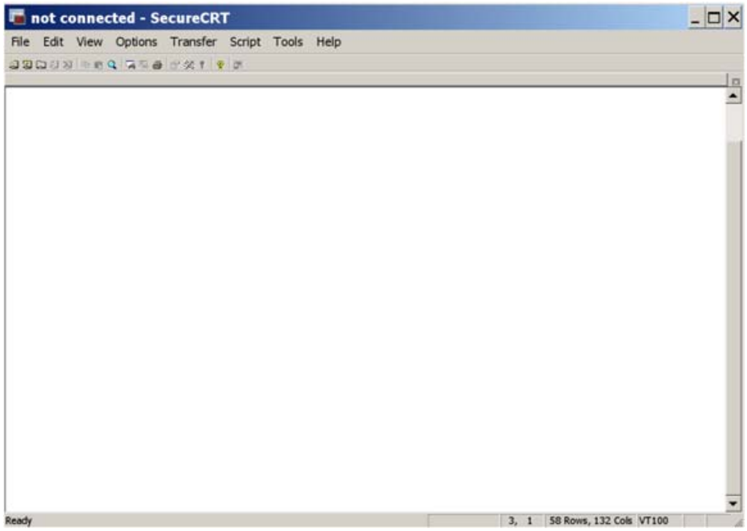


Note this setup does not utilize standard RS232 voltage levels for communication!

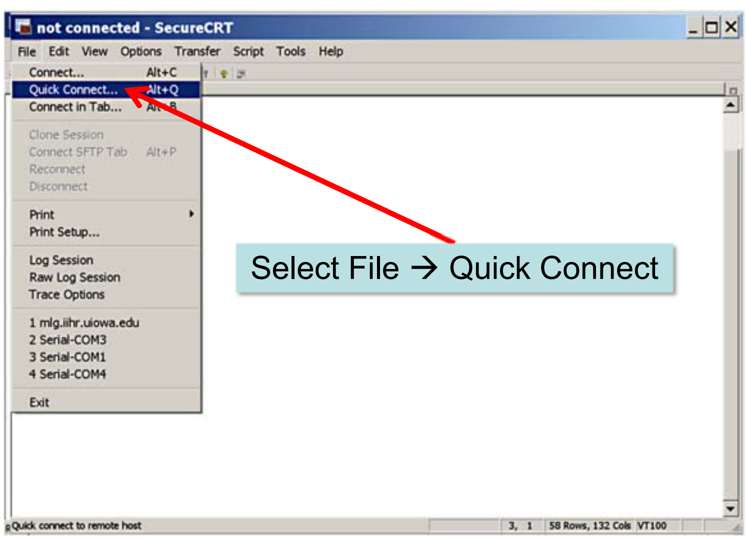
Terminal



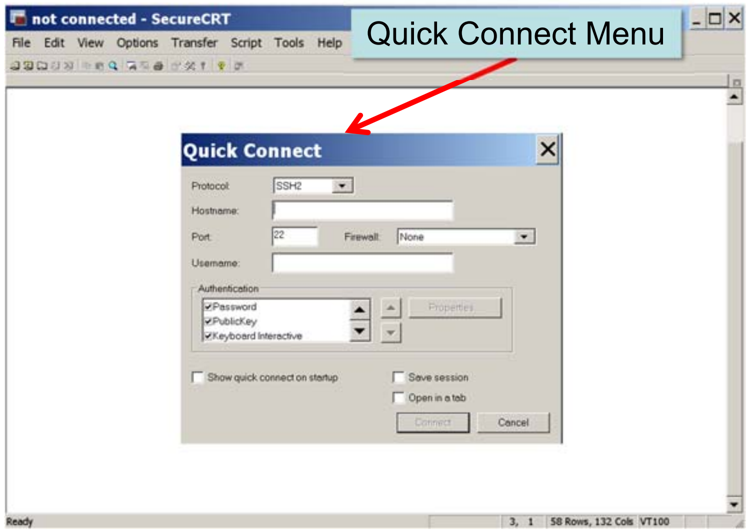
SecureCRT Terminal Emulator



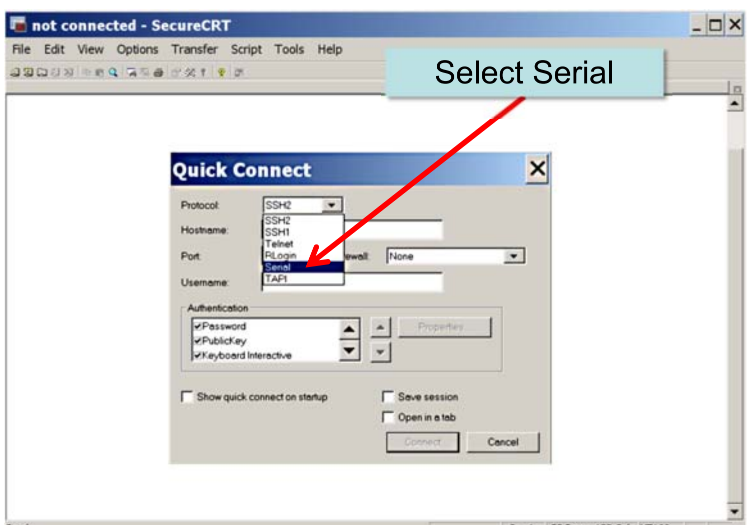
SecureCRT Terminal Emulator



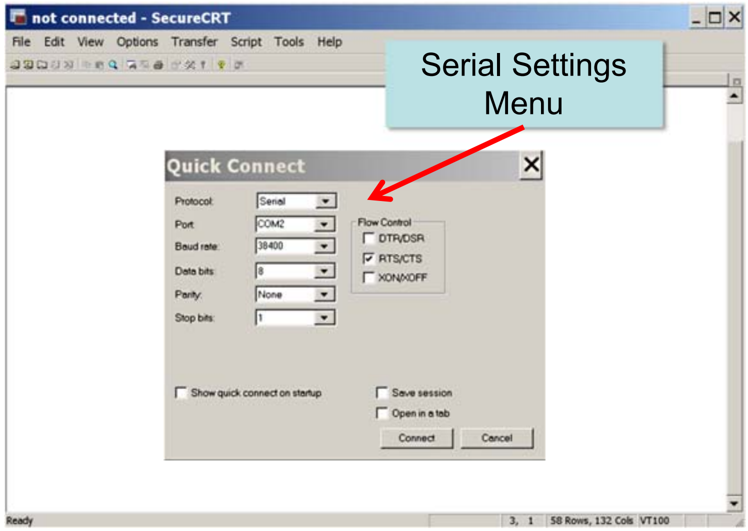
SecureCRT Terminal Emulator



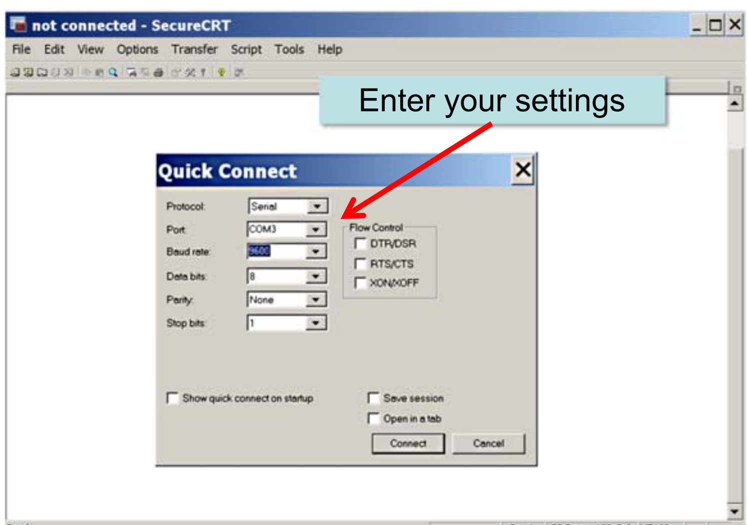
SecureCRT Terminal Emulator



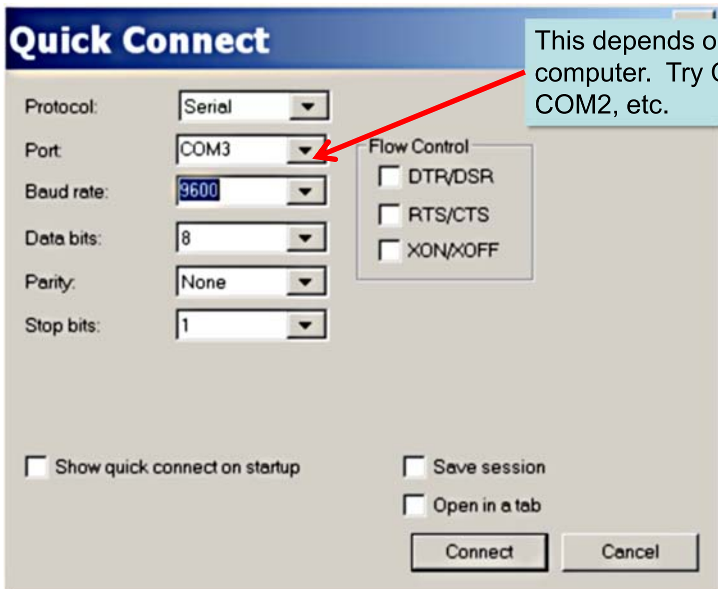
SecureCRT Terminal Emulator



SecureCRT Terminal Emulator

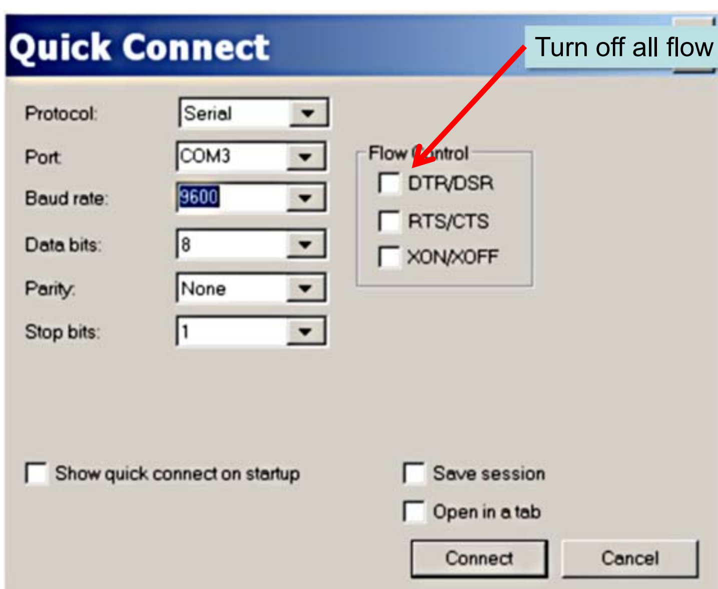


SecureCRT Terminal Emulator



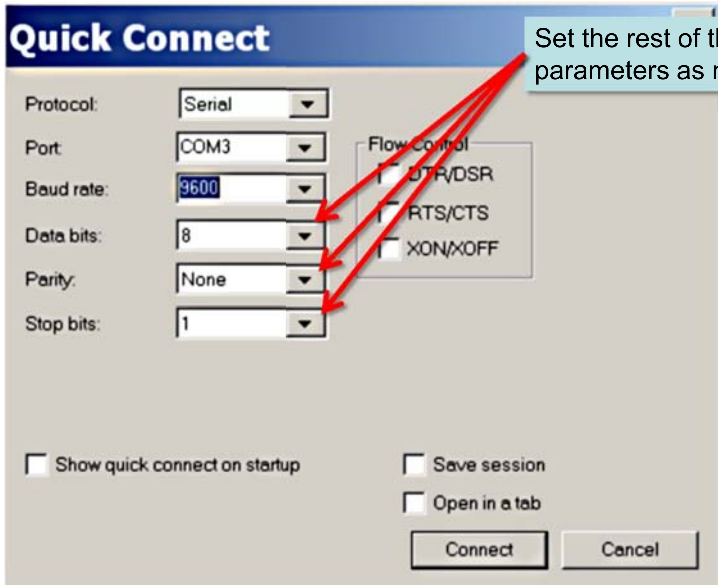
This depends on the computer. Try COM1, COM2, etc.

SecureCRT Terminal Emulator



Turn off all flow control

SecureCRT Terminal Emulator



Set the rest of the parameters as needed


```
…
const char sdata[] = "Hello World!\n";          // String in SRAM
static const char fdata[] PROGMEM = "Flash Gordon\n";  // String in
Flash

int main(void)
{
   unsigned char c;
   char str[25];
   int  adH,adL,dac;
   int i;

   sei();                       // Enable interrupts

   usart_init();                // Initialize the USART
   usart_prints(sdata);         // Print a string from SRAM
   usart_printf(fdata);         // Print a string from FLASH
```

You have to implement these routines.

Refer to previous lecture on serial communication.

Also, there are many resources available on the internet: Peter Fluery, http://winavr.scienceprog.com/avr-gcc-tutorial/programing-avr-usart-module.html, etc. You can user these, but need to understand how the software works.

```
const char sdata[] = "Hello World!\n";          // String in SRAM
static const char fdata[] PROGMEM = "Flash Gordon\n";  // Str Flash

int main(void)
{
   unsigned char c;
   char str[25];
   int adH,adL,dac;
   int i;

  // "echo" test

   usart_prints("Please type 4 characters!");
   for (i=0;i<=4-1;i++){
      c = usart_getc();     // Get character
      usart_putc(c);        // Echo it back
      str[i] = c;
   }
   str[i] = '\0';
```

This is how to get the characters the user types at the PC keyboard into a character string.

This routine also echo's the typed characters back so the user can see what he/she is typing.

Use code such as this for your user interface routine.


```
const char sdata[] = "Hello World!\n";          // String in SRAM
static const char fdata[] PROGMEM = "Flash Gordon\n";  // Str Flash

int main(void)
{
   unsigned char c;
   char str[25];
   int adH,adL,dac;
   int i;
   float v;

                              …

   // Get the voltage, make a formatted string, and then
   // send via the USART.
   readADC(&v);
   sprintf(str,"v = %.3f V\n",v);
   usart_prints(str);
```

You have to implement this routine.

Review the C standard library **sprintf** function.

Make sure you include the proper header files

This will significantly increase the size of your code.

String Conversion

How does one convert from a string to a number? For example, consider the string str:

```
const char str[] = "123";
```

We want to convert this to a number *n* = 123 so we can do arithmetic:

```
              …
n = str2num(str);   // Convert to number
n = n + 10;         // Do arithmetic
              …
```

Where do I get a “str2num” routine? An easy method is to use the C compiler’s string scan routine **sscanf** (next slide) or **atoi**/**atof**.

Potential problem with this is that it pulls in large chunks of code which can quickly fill up flash memory.


```
const char sdata[] = "Hello World!\n";          // String in SRAM
static const char fdata[] PROGMEM = "Flash Gordon\n";  // Str Flash

int main(void)
{
   unsigned char c;
   char str[25];
   int  ch;
   int i;

    …

   // Convert a string to a number.

   sscanf(str,"%d",&ch);
   if ( ch > 1 ) {
      sprintf(str,"\nInvalid channel: %d\n",ch);
      usart_prints(str);
   }

}
```

Note how we used the **sscanf** routine to do the conversion from string to a number.

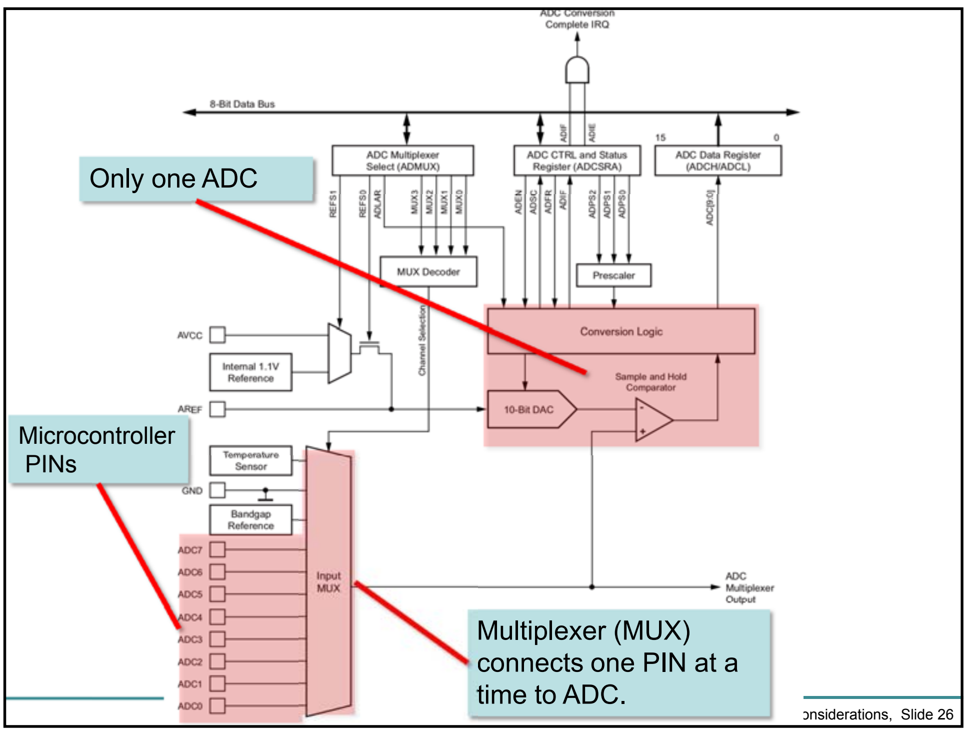
C - Resources

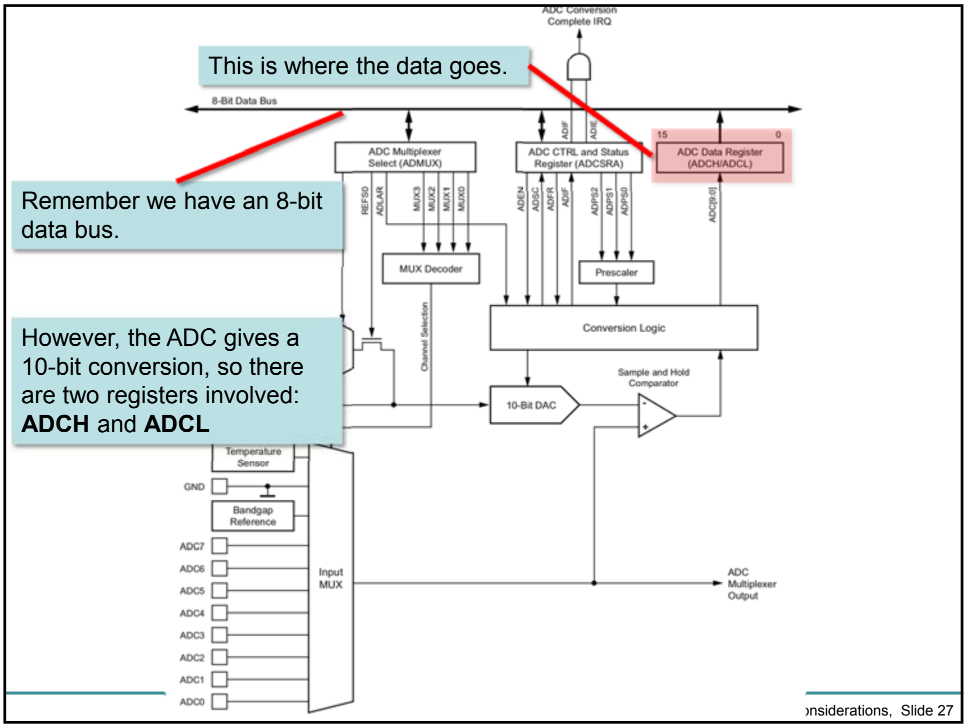
- https://www.gnu.org/software/gnu-c-manual/
- Arrays
  - https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Arrays
- Pointers
  - https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Pointers
- Useful libraries
  - <string.h>
  - <stdio.h> → sprintf, scanf, …
  - <stdlib.h> → itoa, …

ADC

ADC (Analog to Digital Converter)

- 10-bit Resolution
- 0.5 LSB Integral Non-linearity
- ± 2 LSB Absolute Accuracy
- 13 - 260 µs Conversion Time
- Up to 76.9 kSPS (Up to 15 kSPS at Maximum Resolution)
- 6 Multiplexed Single Ended Input Channels
- 2 Additional Multiplexed Single Ended Input Channels (TQFP and QFN/MLF Package only)
- Temperature Sensor Input Channel
- Optional Left Adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- Selectable 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler





Conversion Modes

Single Conversion Mode. As the name implies, application software instructs the ADC to start a conversion. Flags are used to indicate while the conversion is in progress and/or done.

Auto Triggering. Various sources (see table below) can trigger a conversion automatically. This allows for background processing: Timer overflow can trigger ADC which makes conversion automatically.

Table 23-6. ADC Auto Trigger Source Selections

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

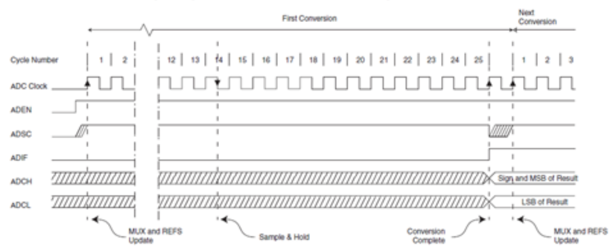
ADC

Note the first conversion takes longer than rest.

Table 23-1. ADC Conversion Time

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	13.5	25
Normal conversions, single ended	1.5	13
Auto Triggered conversions	2	13.5

Figure 23-4. ADC Timing Diagram, First Conversion (Single Conversion Mode)



ADC Reference

ADCs need a reference voltage to compare the voltage they measure with.

The ATmega88's ADC reference voltage sources selection is controlled by the **REFS1** and **REFS0** bits in the **ADMUX** register

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	

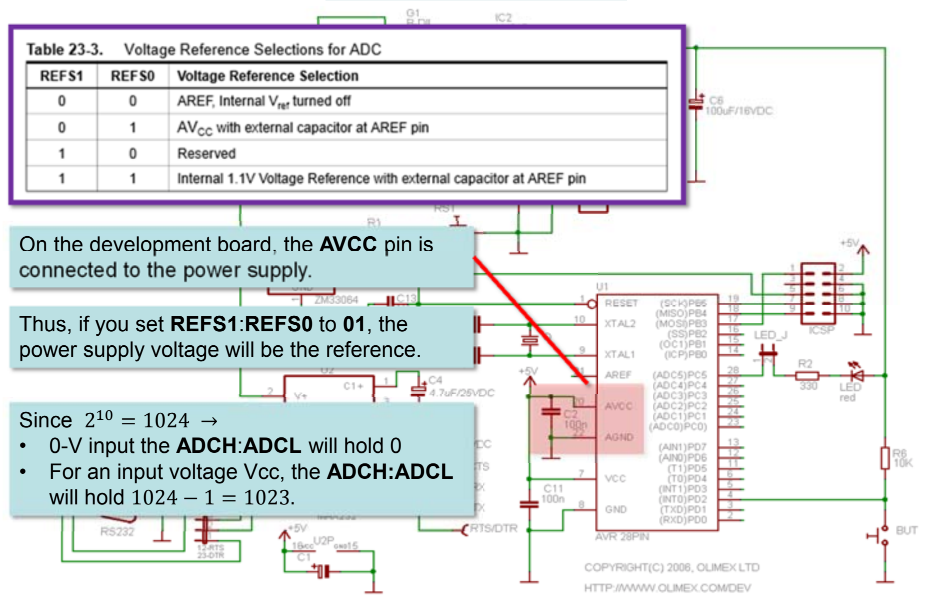
Table 23-3. Voltage Reference Selections for ADC

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

ADC Reference

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



On the development board, the **AVCC** pin is connected to the power supply.

Thus, if you set **REFS1**:**REFS0** to **01**, the power supply voltage will be the reference.

Since $2^{10} = 1024$ →
- 0-V input the **ADCH**:**ADCL** will hold 0
- For an input voltage Vcc, the **ADCH:ADCL** will hold $1024 - 1 = 1023$.

ADC Reference

While using the power supply as a reference for the ADC is convenient, it is problematic is some instances.

The power supply regulator's output voltage many not be very precise.

Electrical Characteristics (LM7805)
Refer to the test circuits. -40°C < T_J < 125°C, I_O = 500mA, V_I = 10V, C_I = 0.1µF, unless otherwise specified.

Symbol	Parameter	Conditions		Min.	Typ.	Max.	Unit
V_O	Output Voltage	T_J = +25°C		4.8	5.0	5.2	V
		5mA ≤ I_O ≤ 1A, P_O ≤ 15W, V_I = 7V to 20V		4.75	5.0	5.25	
Regline	Line Regulation[(1)]	T_J = +25°C	V_O = 7V to 25V	–	4.0	100	mV
			V_I = 8V to 12V	–	1.6	50.0	
Regload	Load Regulation[(1)]	T_J = +25°C	I_O = 5mA to 1.5A	–	9.0	100	mV
			I_O = 250mA to 750mA	–	4.0	50.0	

Loads on the main power supply cause fluctuations and this now changes the reference for the ADC.

EEPROM

ATmega88PA – EEPROM

- **Electrically Erasable Programmable Read-Only Memory**
- **Nonvolatile memory**
- **The ATmeag88PA has 512 bytes of data EEPROM**
  - **It is organized as a separate data space**
  - **Can read/write single bytes**
- **The EEPROM has an endurance of at least 100,000 write/erase cycles**
- **Programming can take several ms**
- **Access to EEPROM is accomplished by reading/writing:**
  - **EEPROM address registers (EEARH & EEARL),**
  - **EEPROM data register (EEDR), and**
  - **EEPROM control register (EECR)**
- **See Section 8.4 "EEPROM Data Memory" in datasheet for details!**

Electrically Erasable Programmable Read-Only Memory (EEPROM)

- EEPROM memory consists of independent cells each representing a single bit → cells are combined to form bytes
- Cells are based on floating-gate transistor technology:
  - An electrical charge trapped on the transistor gate determines the logic level of the cell
- Erasing a cell
  → a charge is placed on the gate and the cell is read as logic one (1)
- Programming a cell
  → discharge the gate and the cell is read as logic zero (0)
- It is only possible to program (discharge) a cell that has been erased (charged)!
  - Programming a byte that is already programmed, without erasing in between, will result in a bit-wise AND between the old value and the new value
  - Use combined "erase and program" operation! → EEPMn bits in EECR

ATmega88PA - EEPROM Registers

- EEAR: specifies which EEPROM byte to read or write
- ATmega88PA: address values between 0 and 511
  → 16-bit register

EEARH and EEARL – The EEPROM Address Register

Bit	15	14	13	12	11	10	9	8	
0x22 (0x42)	–	–	–	–	–	–	–	EEAR8	EEARH
0x21 (0x41)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R/W	
	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	X	
	X	X	X	X	X	X	X	X	

The initial value of EEAR is undefined → proper value must be written before the EEPROM can be accessed

ATmega88PA - EEPROM Registers

- **Write: the EEDR register contains the data to be written to the EEPROM in the address given by the EEAR register**
- **Read: the EEDR contains the data read out from the EEPROM at the address given by EEAR**

EEDR – The EEPROM Data Register

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	MSB							LSB	EEDR
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	

ATmega88PA - EEPROM Read

- **EEPROM read enable bit EERE → read strobe to the EEPROM**
  - **Important: the user must poll the EEPE bit before starting the read operation:**
    - → If a write operation is in progress (EEPE=1), it is neither possible to read the EEPROM, nor to change the address register.
  - **Set up correct address in EEAR register,**
  - **Set EERE bit to trigger the EEPROM read**
  - **EEPROM read access takes one instruction**
    - Requested data is available immediately
    - When the EEPROM is read, the CPU is halted for four cycles before the next instruction is executed

EECR – The EEPROM Control Register

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

ATmega88PA - EEPROM Write

- The EEPROM programming mode bit setting defines which programming action that will be triggered when writing EEPE
  - Can program data in one atomic operation (erase the old value and program the new value) or
  - split the erase and write operations in two different operations.
- While EEPE is set, any write to EEPMn will be ignored.

EECR – The EEPROM Control Register

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

Table 8-1. EEPROM Mode Bits

EEPM1	EEPM0	Programming Time	Operation
0	0	3.4ms	Erase and write in one operation (atomic operation)
0	1	1.8ms	Erase only
1	0	1.8ms	Write only
1	1	–	Reserved for future use

ATmega88PA - EEPROM Write

- EEMPE: EEPROM Master Write Enable
- The EEMPE bit determines whether setting EEPE to one causes the EEPROM to be written
  - When EEMPE is set, setting EEPE within 4 clock cycles will write data to the EEPROM at the selected address
  - If EEMPE is zero, setting EEPE will have no effect!
  - When EEMPE has been set by software, hardware clears the bit to zero after four clock cycles

EECR – The EEPROM Control Register

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

ATmega88PA - EEPROM Write

- **EEPE: EEPROM Write Enable → write strobe to the EEPROM**
- **Procedure for writing the EEPROM:**
  1. Wait until EEPE becomes zero
  2. Write new EEPROM address to EEAR
  3. Write new EEPROM data to EEDR
  4. Clear EEPM1 and EEPM0 (→ set erase & write mode)
  5. Write a logical one to the EEMPE bit while writing a zero to EEPE
  6. Within 4 clock cycles after setting EEMPE, set EEPE bit
- **It is recommended to have the global interrupts disabled during EEPROM write operations!**

EECR – The EEPROM Control Register

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

ATmega88PA - EEPROM Write

- **When the write access time has elapsed, the EEPE bit is cleared by hardware**

  →The user software can poll this bit and wait for a zero before writing the next byte
- **When EEPE has been set, the CPU is halted for two cycles before the next instruction is executed**

EECR – The EEPROM Control Register

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

- For more information see
  - datasheet → read/write C code snippets
  - application note:
  - https://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en591206

DAC

I2C DAC - MAX518

19-0393; Rev 1; 9/02

MAXIM

2-Wire Serial 8-Bit DACs with Rail-to-Rail Outputs

General Description

The MAX517/MAX518/MAX519 are 8-bit voltage output digital-to-analog converters (DACs) with a simple 2-wire serial interface that allows communication between multiple devices. They operate from a single 5V supply and their internal precision buffers allow the DAC outputs to swing rail-to-rail.

The MAX517 is a single DAC and the MAX518/MAX519 are dual DACs. The MAX518 uses the supply voltage as the reference for both DACs. The MAX517 has a reference input for its single DAC and each of the MAX519's two DACs has its own reference input.

The MAX517/MAX518/MAX519 feature a serial interface and internal software protocol, allowing communication at data rates up to 400kbps. The interface, combined with the double-buffered input configuration, allows the DAC registers of the dual devices to be updated individually or simultaneously. In addition, the devices can be put into a low-power shutdown mode that reduces supply current to 4µA. Power-on reset ensures the DAC outputs are at 0V when power is initially applied.

The MAX517/MAX518 are available in space-saving 8-pin DIP and SO packages. The MAX519 comes in 16-pin DIP and SO packages.

Features

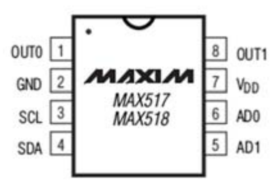
- Single +5V Supply
- Simple 2-Wire Serial Interface
- I^2C Compatible
- Output Buffer Amplifiers Swing Rail-to-Rail
- Space-Saving 8-pin DIP/SO Packages (MAX517/MAX518)
- Reference Input Range Includes Both Supply Rails (MAX517/MAX519)
- Power-On Reset Clears All Latches
- 4µA Power-Down Mode

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE	TUE (LSB)
MAX517ACPA	0°C to +70°C	8 Plastic DIP	1
MAX517BCPA	0°C to +70°C	8 Plastic DIP	1.5
MAX517ACSA	0°C to +70°C	8 SO	1
MAX517BCSA	0°C to +70°C	8 SO	1.5
MAX517BC/D	0°C to +70°C	Dice*	1.5

Ordering Information continued at end of data sheet.
**Dice are specified at T_A = +25°C, DC parameters only.*
***Contact factory for availability and processing to MIL-STD-883.*

MAX518 – Pin Description

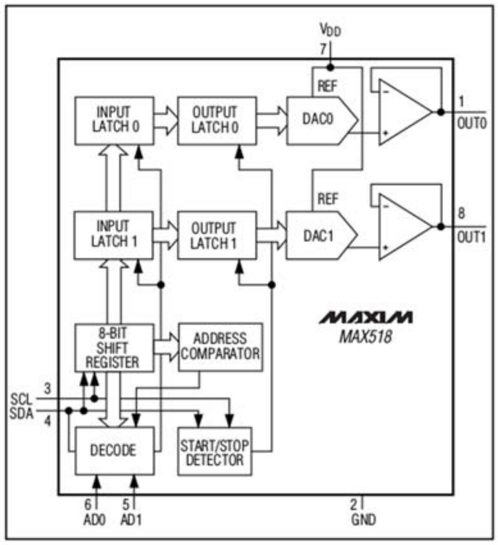


Don't forget to add decoupling capacitors!

PIN			NAME	FUNCTION
MAX517	MAX518	MAX519		
1	1	1	OUT0	DAC0 Voltage Output
2	2	4	GND	Ground
—	—	5	AD3	Address Input 3; sets IC's slave address
3	3	6	SCL	Serial Clock Input
4	4	8	SDA	Serial Data Input
—	—	9	AD2	Address Input 2; sets IC's slave address
5	5	10	AD1	Address Input 1; sets IC's slave address
6	6	11	AD0	Address Input 0; sets IC's slave address
7	7	12	VDD	Power Supply, +5V; used as reference for MAX518
—	—	13	REF1	Reference Voltage Input for DAC1
8	—	15	REF0	Reference Voltage Input for DAC0
—	8	16	OUT1	DAC1 Voltage Output
—	—	2, 3, 7, 14	N.C.	No Connect—not internally connected.

MAX518 - Overview

Functional Diagram



MAX518 – I2C Communication

DAC output changes

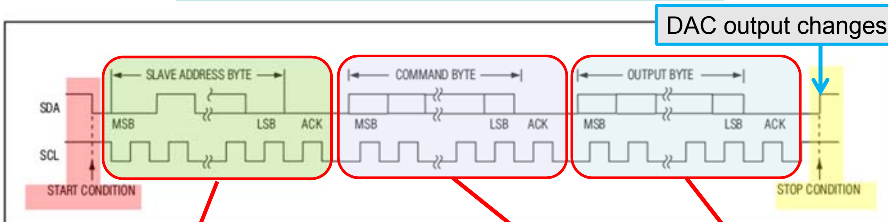


Figure 4. A Complete Serial Transmission

determines output voltage

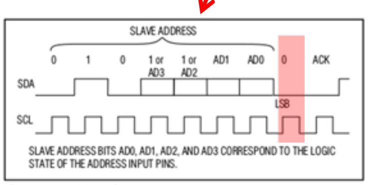
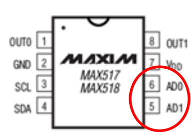


Figure 6. Address Byte



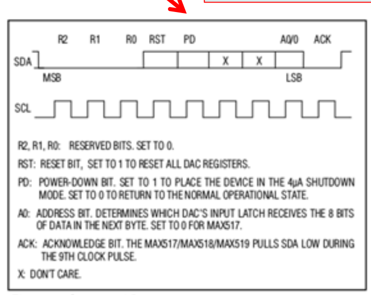


Figure 7. Command Byte

MAX518 – Output Register/Byte

Table 1. Unipolar Code Table

DAC CONTENTS	ANALOG OUTPUT
11111111	$+V_{REF}\left(\frac{255}{256}\right)$
10000001	$+V_{REF}\left(\frac{129}{256}\right)$
10000000	$+V_{REF}\left(\frac{128}{256}\right)=\frac{V_{REF}}{2}$
01111111	$+V_{REF}\left(\frac{127}{256}\right)$
00000001	$+V_{REF}\left(\frac{1}{256}\right)$
00000000	0V

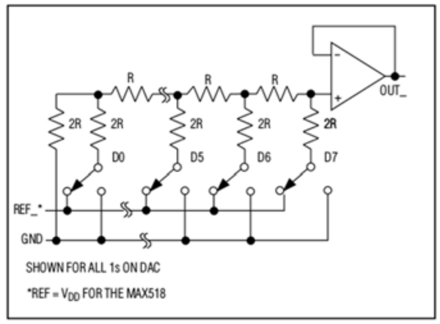


Figure 15. DAC Simplified Circuit Diagram

MAX518 – Setting One vs Both Channels

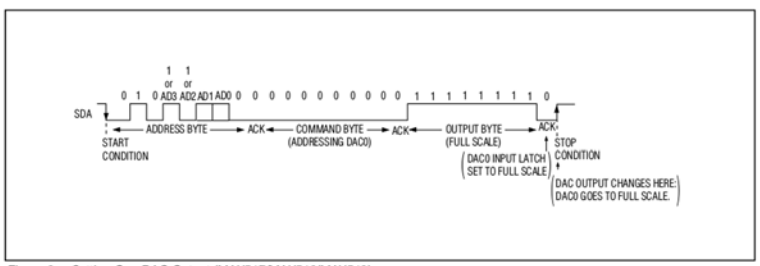


Figure 8a. Setting One DAC Output (MAX517/MAX518/MAX519)

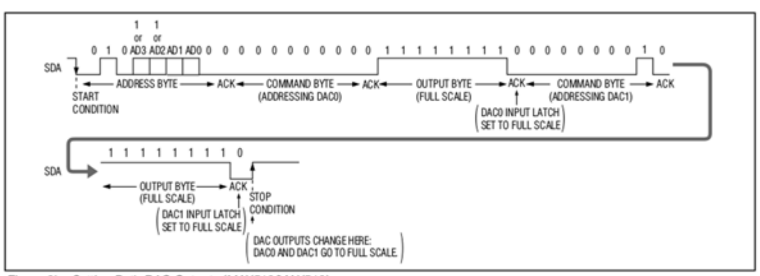


Figure 8b. Setting Both DAC Outputs (MAX518/MAX519)

… EOL