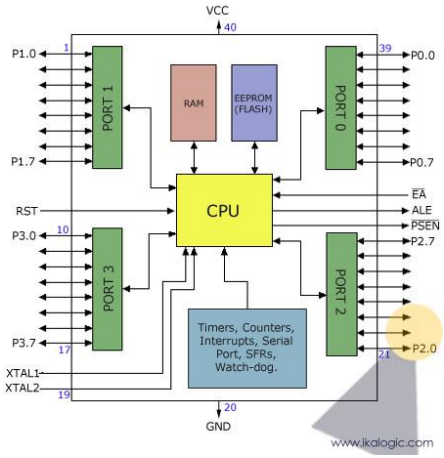
Embedded Systems Design, Spring 2025

Lecture 5



Function and architecture of microcontroller systems

Outline

- Introduction
- AVR Architecture in general
- ATMega328 Architecture in particular
- Memory Components
- Using the eeprom memory (practical example)

RISC

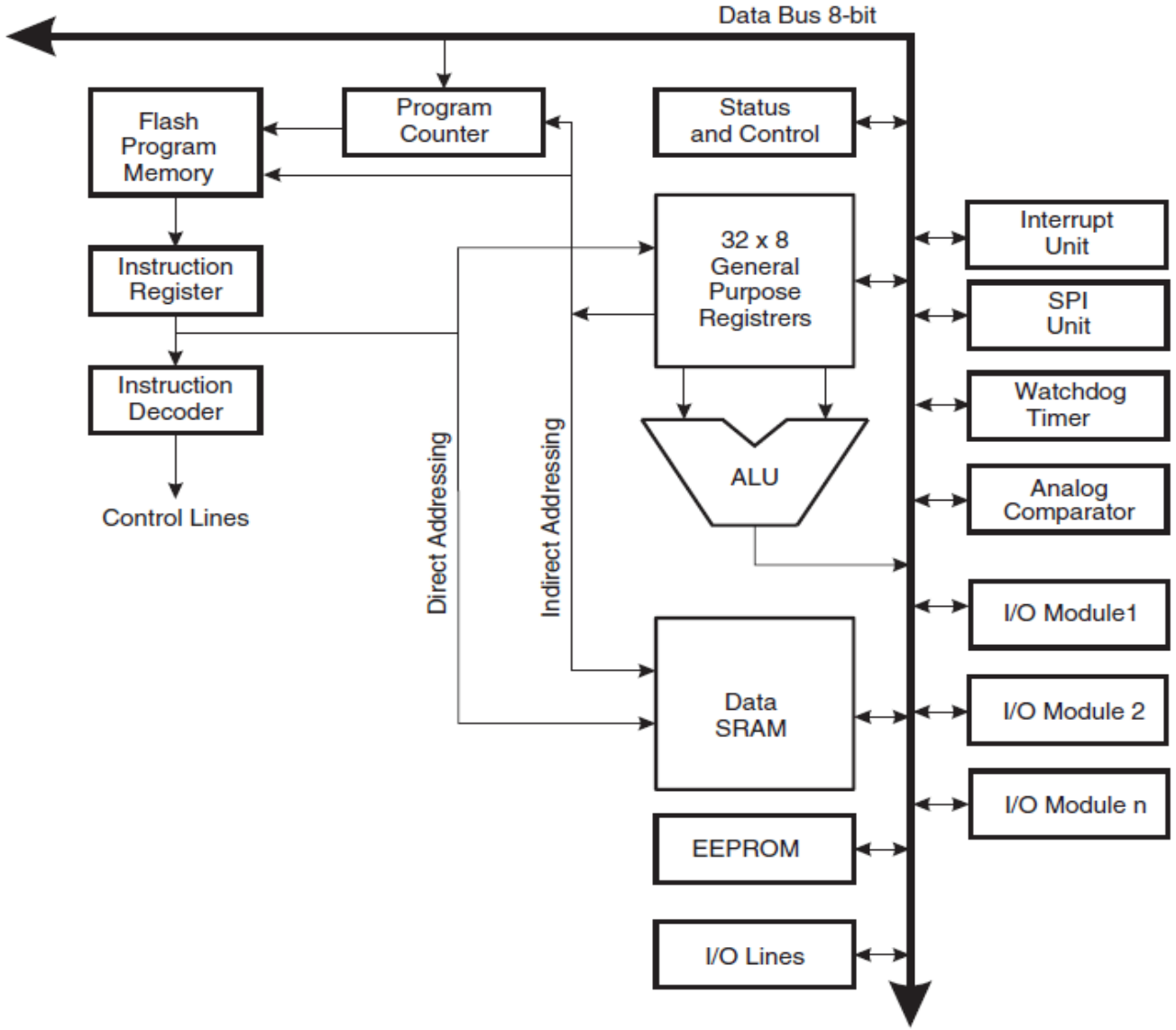
- Reduced Instruction Set Computing / (Reduced complexity)
  - Simplified instruction set (highly optimized)
  - Reduced cycles per instructions
  - For Atmel – instructions are 16 bits
  - Optimized for C code execution
- Based on Harvard architecture:
  - one instruction bus where the CPU reads executable instructions;
  - one data bus to read or write the corresponding data.

RISC vs CISC

- CISC: large & complex instruction set, one instruction performs multiple low-level operations
- CISC used in processors such as Intel, AMD, gaming consoles
- RISC used in Atmel, ARM, RISC-V etc.

Assembly	Assembly
LOAD R1,A ; Load value from memory address A into R1 LOAD R2, B ; Load value from memory address B into R2 ADD R3, R1, R2 ;Add R1 and R2, store result in R3 STORE C, R3 ; Store result in memory address C	ADD C,A, B ; Directly add values at memory addresses A and B, store in C

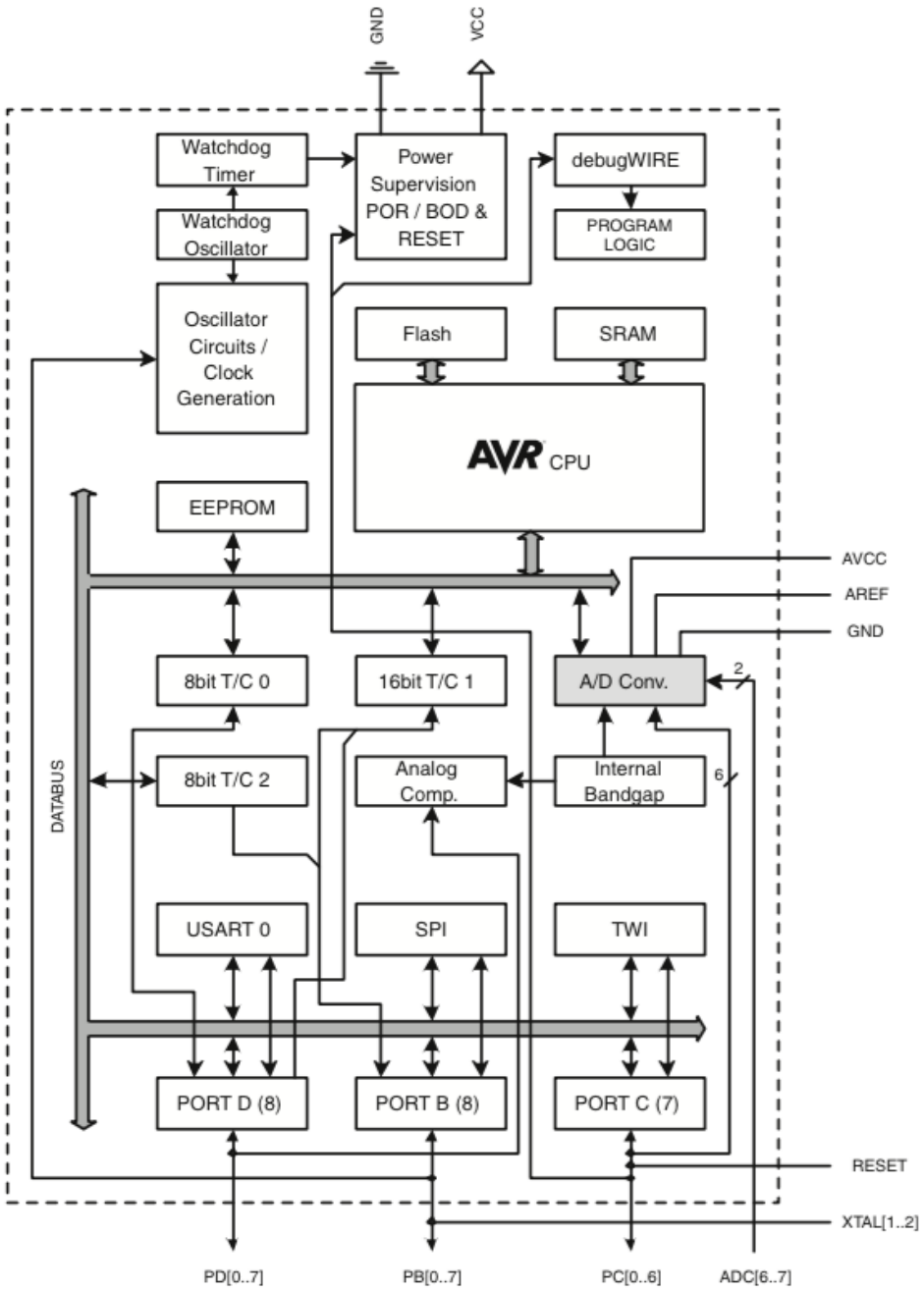
Block Diagram of the AVR Architecture



ATmega 328p

- Architecture:
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Up to 20 MIPS Throughput at 20 MHz
- Memory:
  - Flash: 32 K Bytes
  - EEPROM: 1K Bytes
  - RAM: 2 K Bytes
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
- Peripherals:
  - Two 8-bit Timer/Counters
  - One 16-bit Timer/Counter
  - Six PWM Channels
  - 8 channel 10 bit ADC

Figure 2-1. Block Diagram



General Purpose Working Registers

Figure 6-2. AVR CPU General Purpose Working Registers

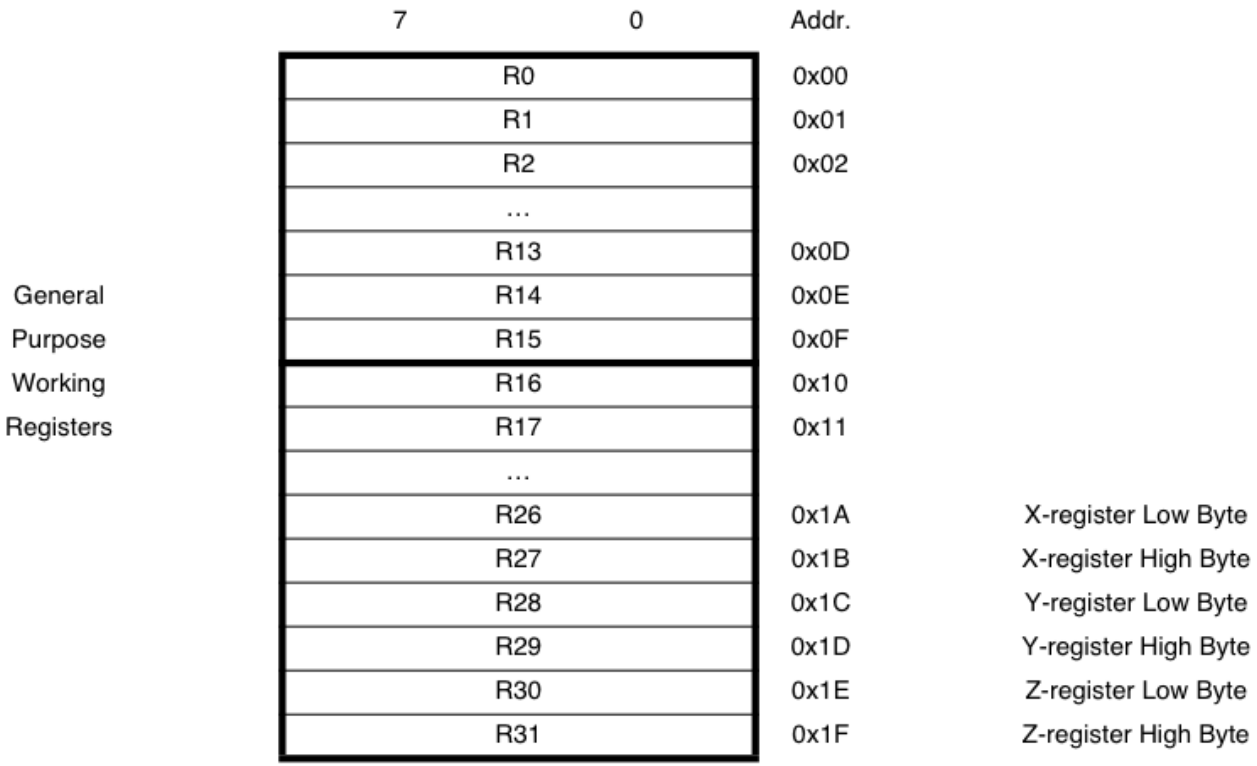


Figure 6-4. The Parallel Instruction Fetches and Instruction Executions

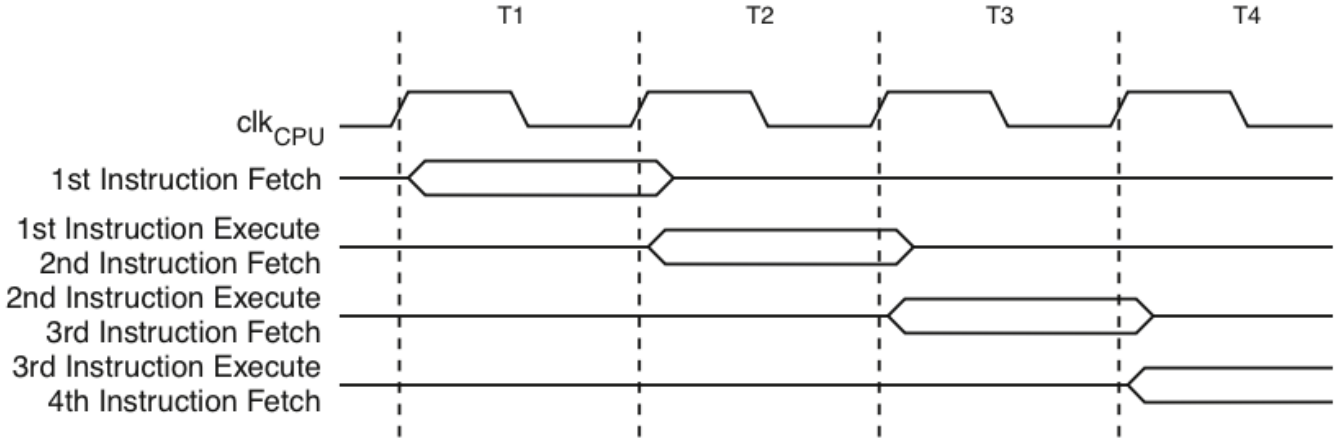
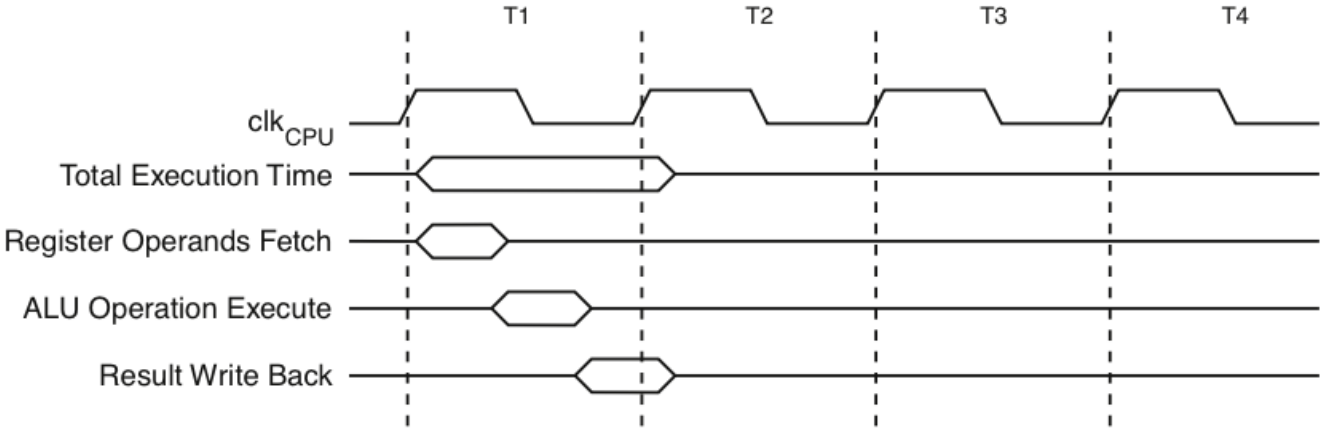


Figure 6-5. Single Cycle ALU Operation



Memories

- Program Memory: 32K bytes On-chip In-System Reprogrammable Flash memory
  - 16K x 16 to hold AVR instructions (16 bit wide)
  - Divided into:
    - Bootloader section (typically your arduino bootloader)
    - Application program section
  - Non volatile
- Data memory:
  - Volatile
  - 2303 8bit locations (for registers and SRAM)

Figure 7-2. Program Memory Map ATmega88PA, ATmega168PA and ATmega328P

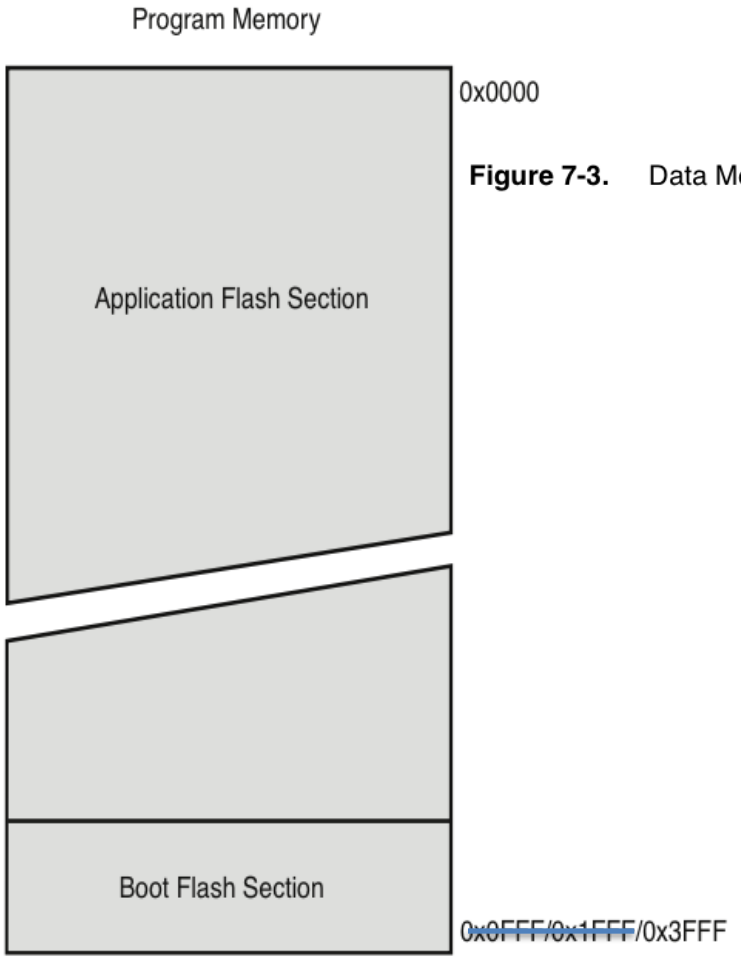
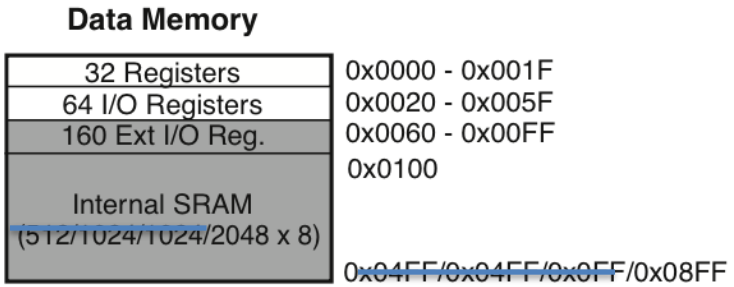
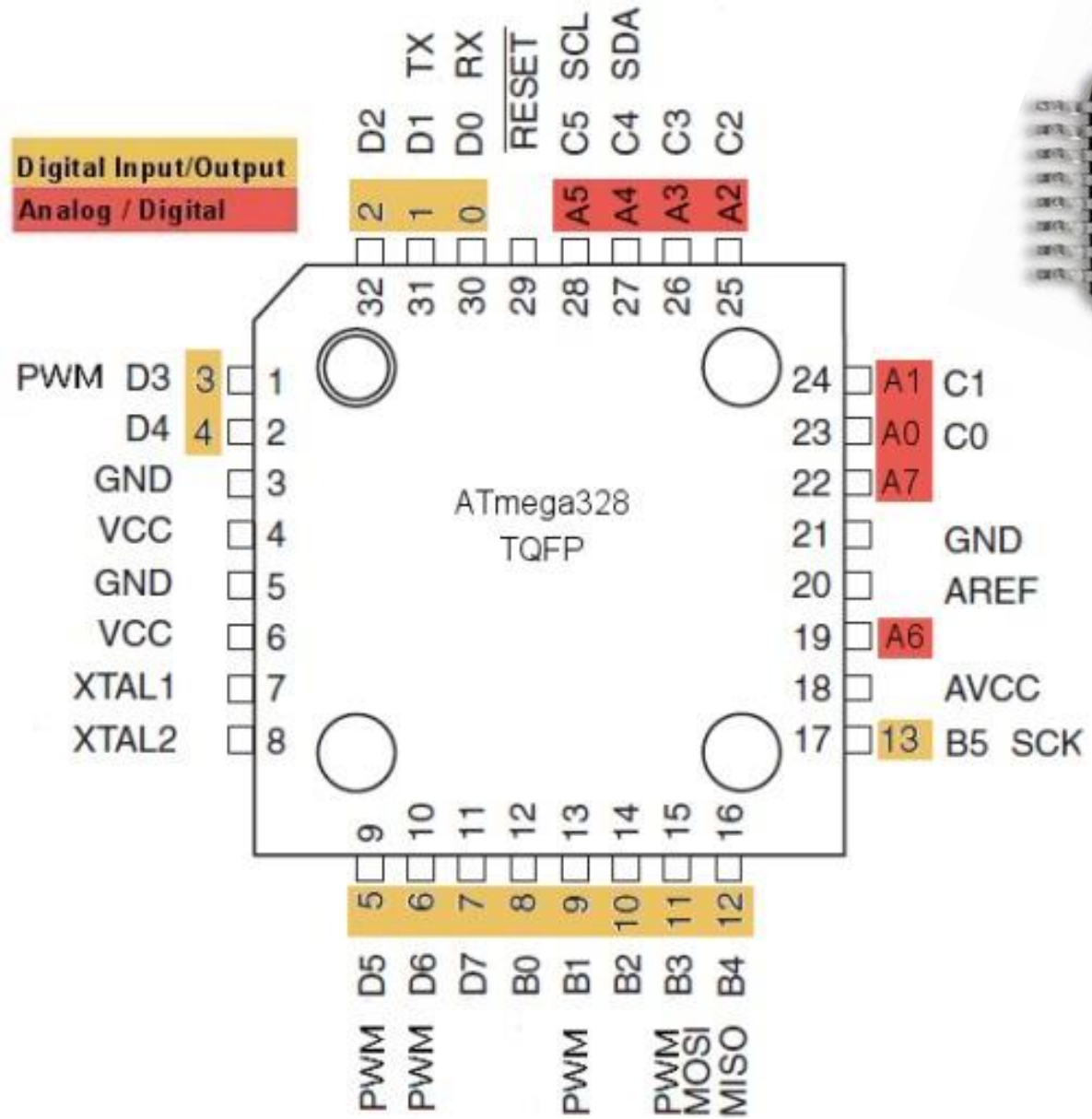
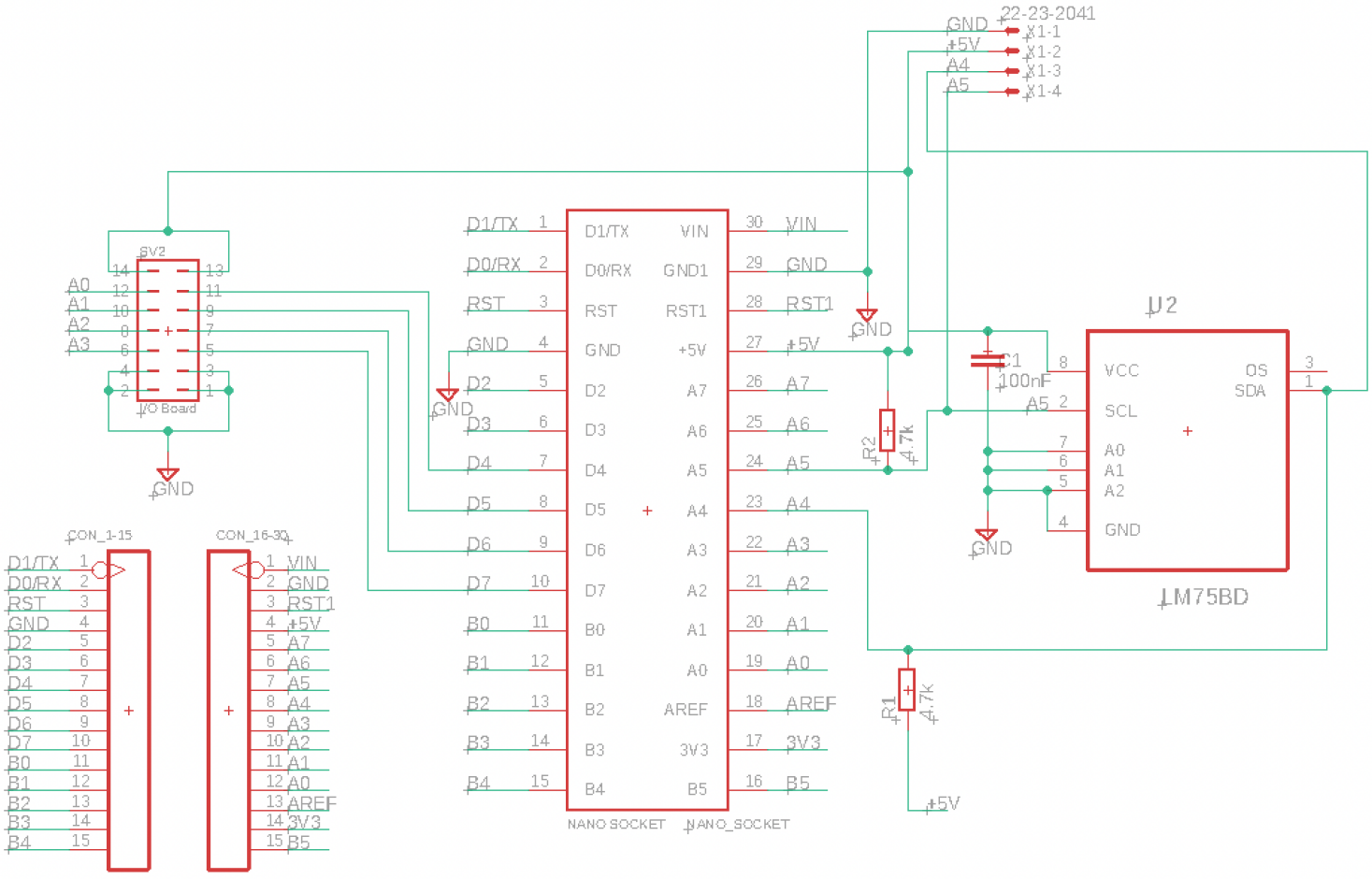


Figure 7-3. Data Memory Map

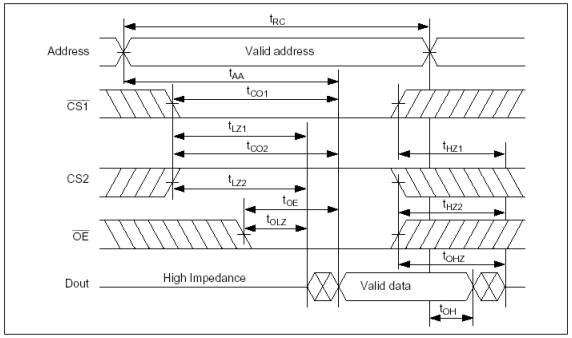


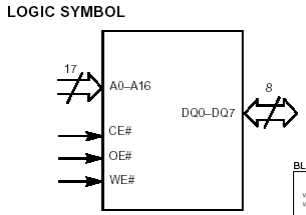

Digital Input/Output
Analog / Digital
D2 D1 TX D0 RX RESET C5 SCL C4 SDA C3 C2 C1 C0
2 1 0 A5 A4 A3 A2
32 31 30 29 28 27 26 25
PWM D3 3 1
D4 4 2
GND 3
VCC 4
GND 5
VCC 6
XTAL1 7
XTAL2 8
ATmega328
TQFP
24 A1 C1
23 A0 C0
22 A7
21 GND
20 AREF
19 A6
18 AVCC
17 13 B5 SCK
9 10 11 12 13 14 15 16
5 6 7 8 9 10 11 12
D5 D6 D7 B0 B1 B2 B3 B4
PWM PWM PWM PWM MOSI MISO

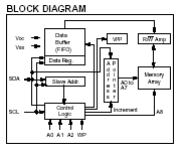
SDU Nano Shield Schematic v 2023

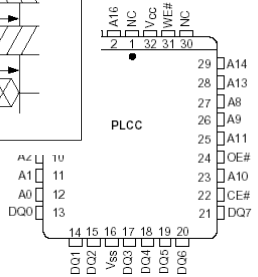


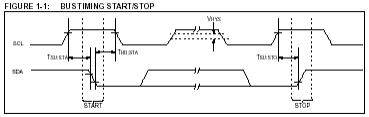
Memory Components











AMD

Am29F010A

1 Megabit (128 K x 8-bit)
CMOS 5.0 Volt-only, Uniform Sector Flash Memory

DISTINCTIVE CHARACTERISTICS

- **Single power supply operation**
  - 5.0 V ± 10% for read, erase, and program operations
  - Simplifies system-level
- **Manufactured on 0.55 µm**
  - Compatible with 0.65 µr
- **High performance**
  - 45 ns maximum access
- **Low power consumption**
  - 20 mA typical active rea
  - 30 mA typical program/
  - <1 µA typical standby c
- **Flexible sector architectu**
  - Eight uniform sectors
  - Any combination of sect
  - Supports full chip erase
- **Sector protection**
  - Hardware-based feature enables program and er combination of sectors
  - Sector protection/unprot implemented using stan programming equipmen
- **Embedded Algorithms**
  - Embedded Erase algorithm automatically

Features

- Fast Read Access Time - 120 ns
- Fast Byte Write - 200 µs or 1 ms
- Self-Timed Byte Write Cycle
  - Internal Address and Data Latches
  - Internal Control Timer
  - Automatic Clear Before Write
- Direct Microprocessor Control
  - READY/$\overline{BUSY}$ Open Drain Output
  - DATA Polling
- Low Power
  - 30 mA Active Current
  - 100 µA CMOS Standby Current
- High Reliability
  - Endurance: 10^4 or 10^5 Cycles
  - Data Retention: 10 Years
- 5V ± 10% Supply
- CMOS and TTL Compatible Inputs and Outputs
- JEDEC Approved Byte-Wide Pinout
- Commercial and Industrial Temperature Ranges

Description

The AT28C64 is a low-power, high-performance 8,192 words by 8-bit nonvolatile Electrically Erasable and Programmable Read Only Memory with popular, easy to use features. The device is manufactured with Atmel's reliable nonvolatile technology.

(continued)

Pin Configurations

Pin Name	Function
A0 - A12	Addresses
$\overline{CE}$	Chip Enable
$\overline{OE}$	Output Enable
$\overline{WE}$	Write Enable
I/O0 - I/O7	Data Inputs/Outputs
RDY/$\overline{BUSY}$	Ready/$\overline{Busy}$ Output
NC	No Connect
DC	Don't Connect

PDIP, SOIC
Top View

TSOP
Top View

LCC, PLCC
Top View

Note: PLCC package pins 1 and 17 are DON'T CONNECT.

Rev. 0001G–10/98

1

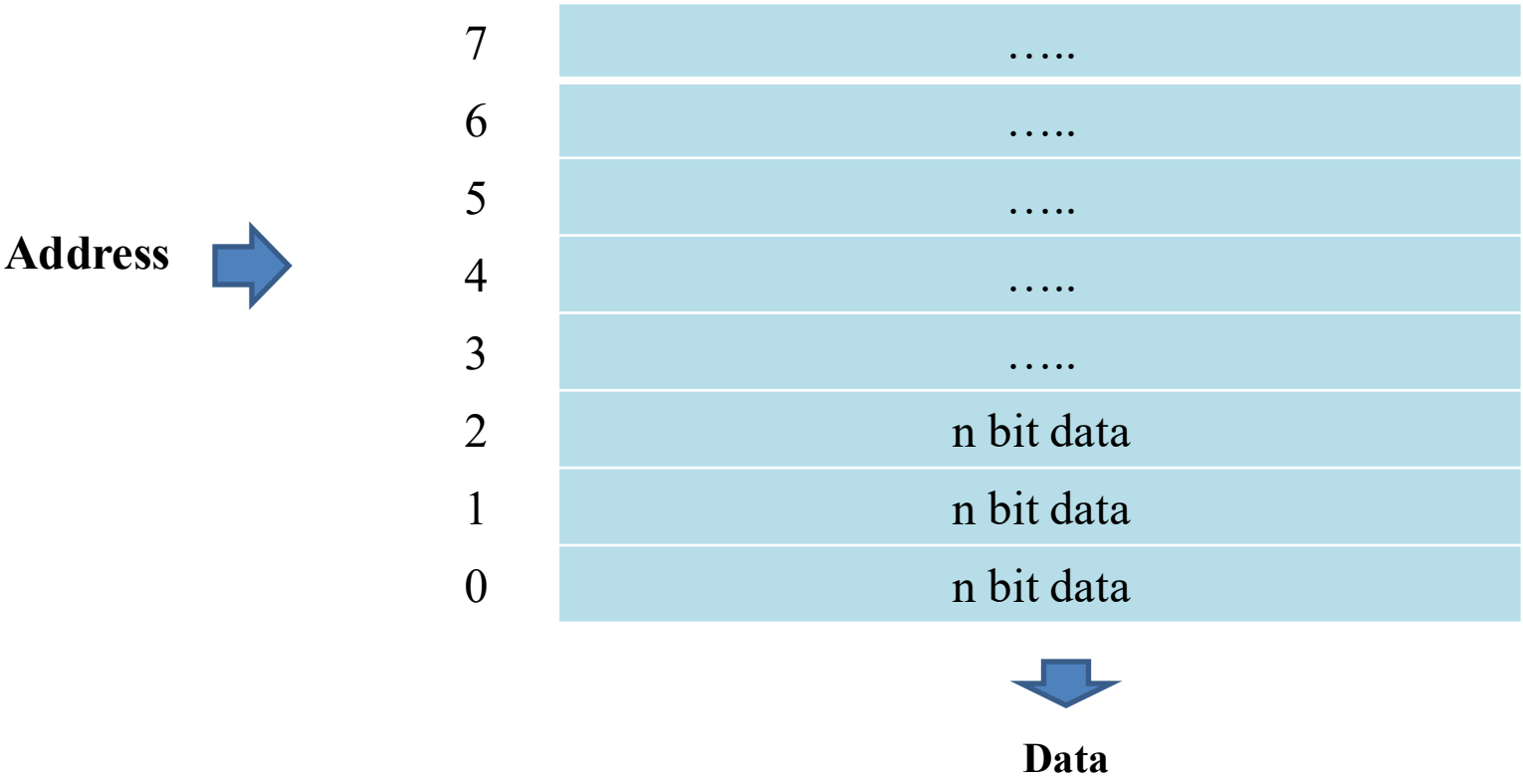
ATMEL

64K (8K x 8)
Parallel
EEPROMs

AT28C64
AT28C64X

SDU

General Memory Structure



ROM/RAM Memory

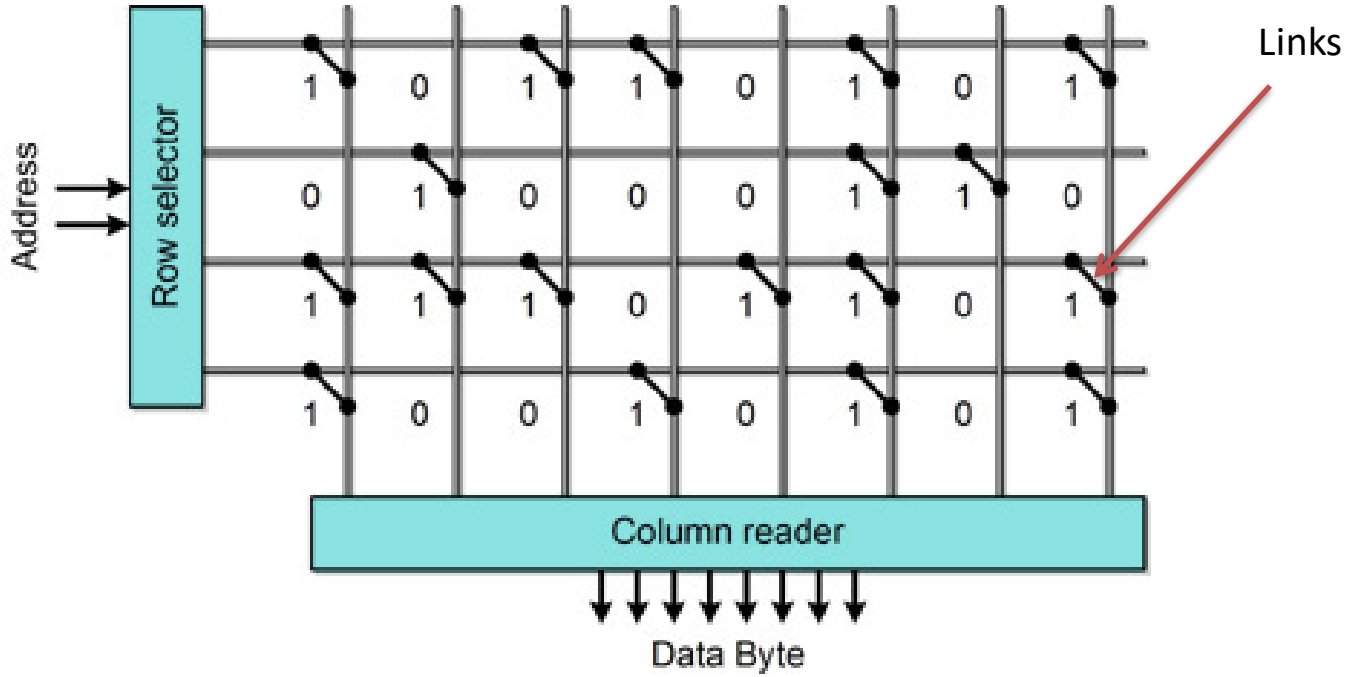
ROM – Read Only Memory

- **EPROM –** Erasable Programmable Read Only Memory
- **EEPROM –** Electrically Erasable Prom (E^2Prom)
- **FLASH**

RAM – Random Access Memory

- **SRAM –** Static RAM
- **DRAM –** Dynamic RAM

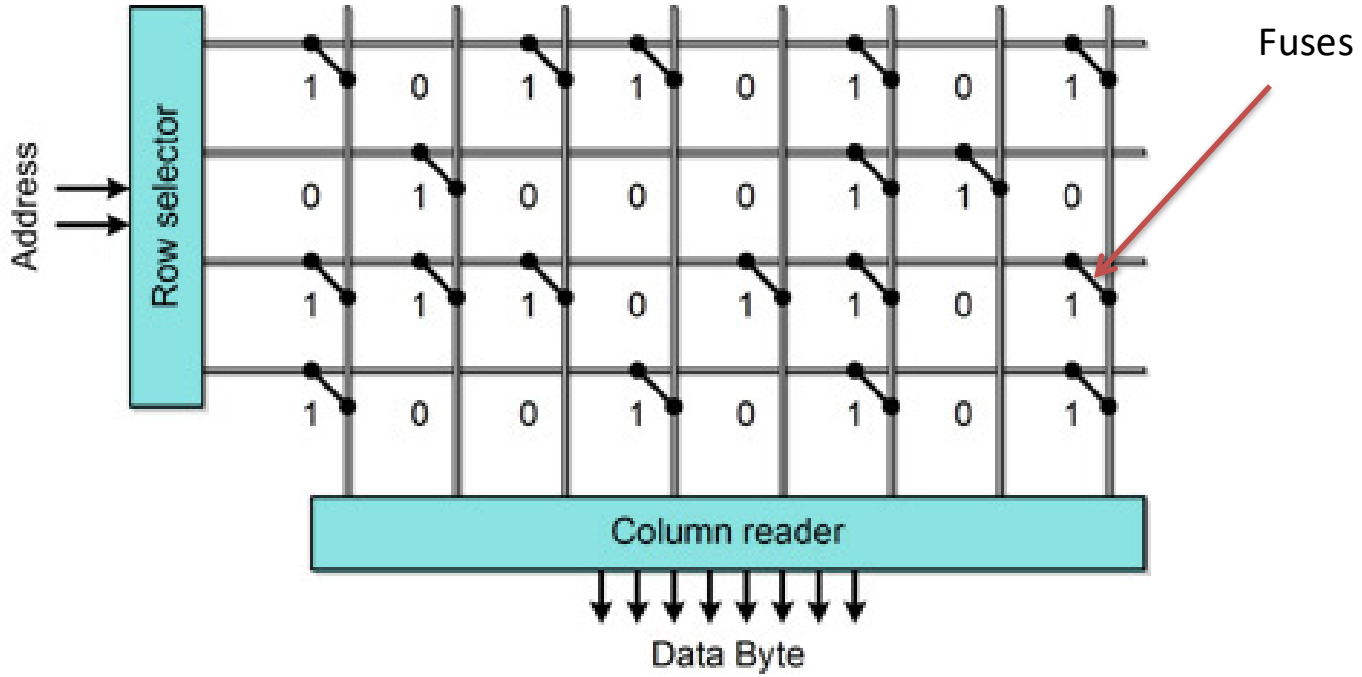
ROM (Read Only Memory)



• Links are ”pre-fabricated” using special masks, leading to no possibility for changing them later.

• Expensive to make

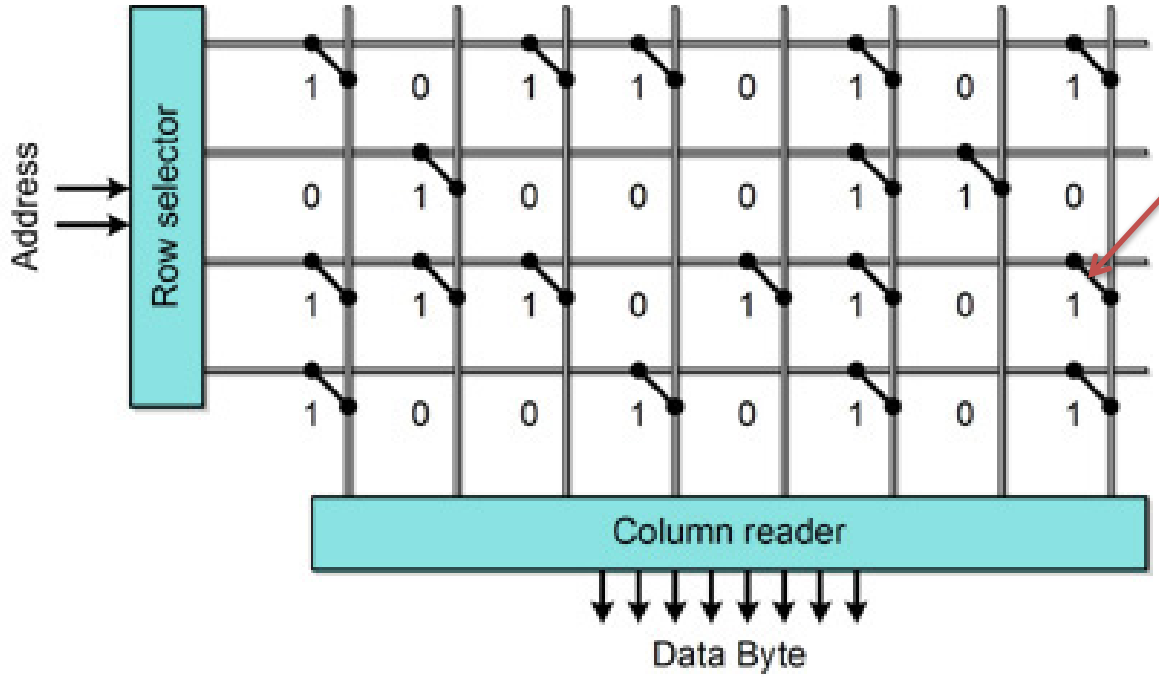
PROM (Programmable Read Only Memory)

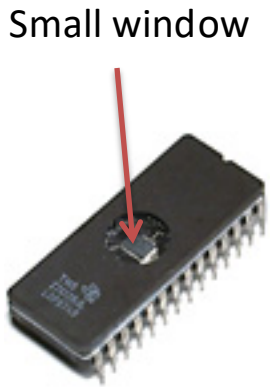


• Links are initially intact fuses. By applying a high voltage to a specific row and column a fuse can be ”blown” – leading to creation of a 0.

• Easy to program, low cost

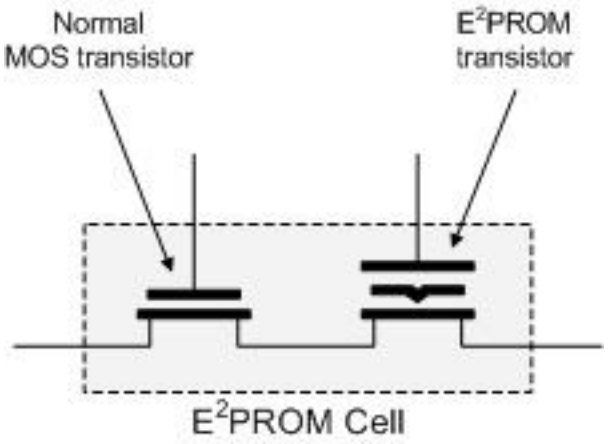
EPROM (Erasable Programmable Read Only Memory)

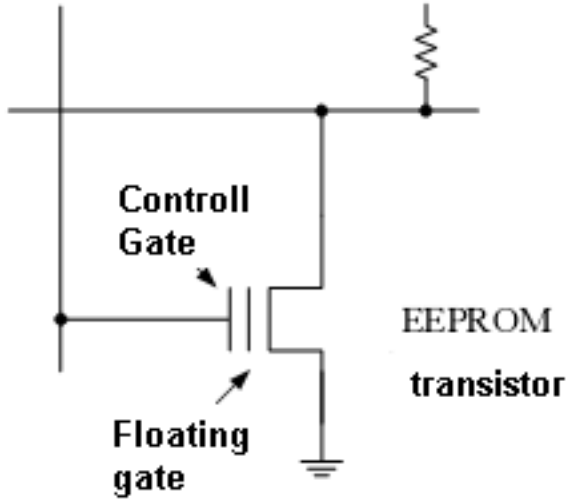




- Behavior similar to PROM with the possibility to reset the fuses to intact by use of ultra-violet light.
- Easy to program, low cost, used for many years

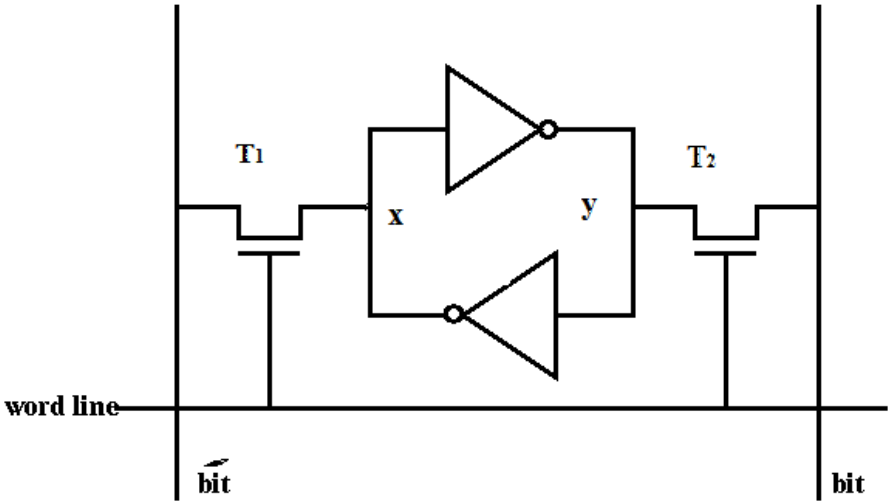
EEPROM (Electrically Erasable Programmable Read Only Memory)





- Behavior similar to EPROM with the possibility to reset the fuses to intact by use of field electron emission
- Each cell needs a write, a read and a reset transistor
- Easy to program in situ, without the need of removal
- Limited number of re-writes

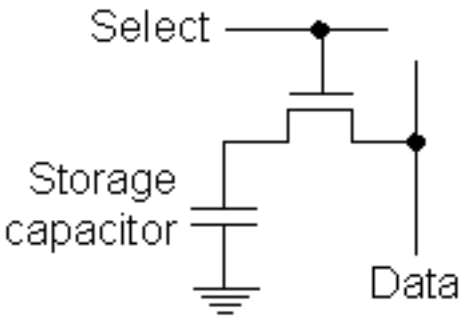
SRAM memory cell

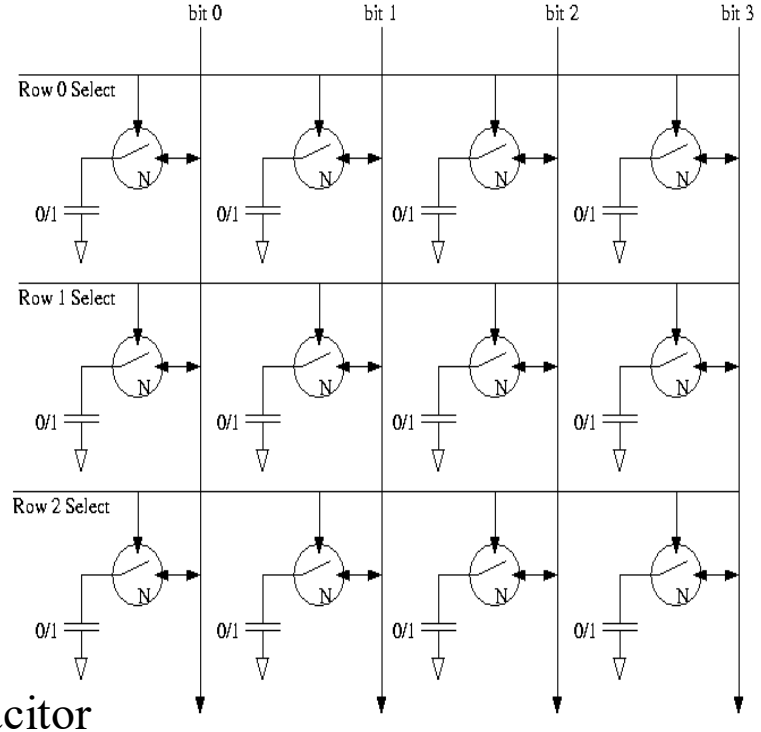


Basic six-transistor static memory cell

- Idle state: word line keeps the T1 and T2 in the OFF state
- Read state: word line puts the T1 and T2 in the ON state, bit and ~bit are read
- Write state: Bit and ~bit are changed, word line not used

DRAM memory cell

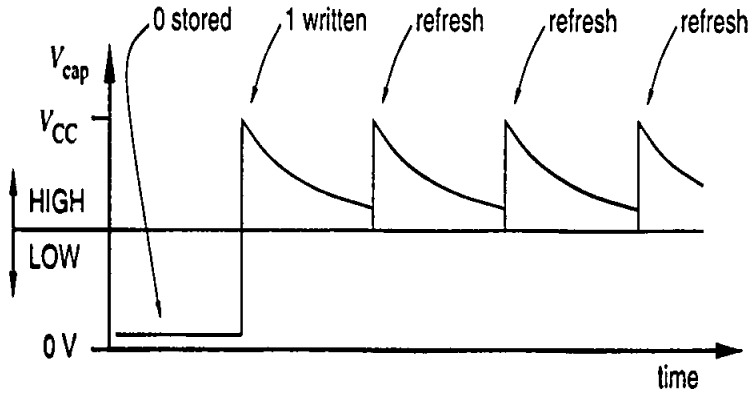




Dynamic RAM use a MOS capacitor as the basic storage cell

- The switch is used to store a charge in the capacitor
- Due to leakage of the capacitor, it needs refreshing
- Slower than SRAM

DRAM refresh



Voltage stored in a DRAM cell after writing and refresh operations

Reading/Writing to Atmega328 EEPROM

- Help from the AVR-GCC eeprom library functions

```
#include <avr/eeprom.h>
```

- Many functions for writing or reading bytes, ints, floats or even byte blocks

```
void eeprom_write_byte(uint8_t *pAddress, uint8_t value)
void eeprom_write_float(float *pAddress, float value)
…
uint8_t eeprom_read_byte(const uint8_t * pAddress)
float eeprom_read_float(const float * pAddress)
```

Example – writing a byte to eeprom address 0

```
#include … //add necessary includes for the code to work
#include <avr/eeprom.h>
void main(){
        uint8_t rb, wb = 50;
        unsigned int addr = 0; //eeprom address is 0 in this
program
        uart_init();
        io_redirect();
        while (1){
                eeprom_write_byte((uint8_t *)addr, wb); //write
                rb = eeprom_read_byte((uint8_t *)addr); //read
                printf("read from eeprom %d \n", rb); //print
                _delay_ms(1000);
                wb++; //write a different number next time
                }
}
```

What will be written

What will be read

Task

- Use your arduino, shield + digital I/O board to:
  - Read the temperature every second from the sensor
  - Read a stored temperature every second from the eeprom
  - Display both the current and the saved temperatures every second
  - If the last button is pressed on your DIO board, save the current temperature on eeprom (as int, or as float)

  - **Try restarting your arduino and see what happens!**