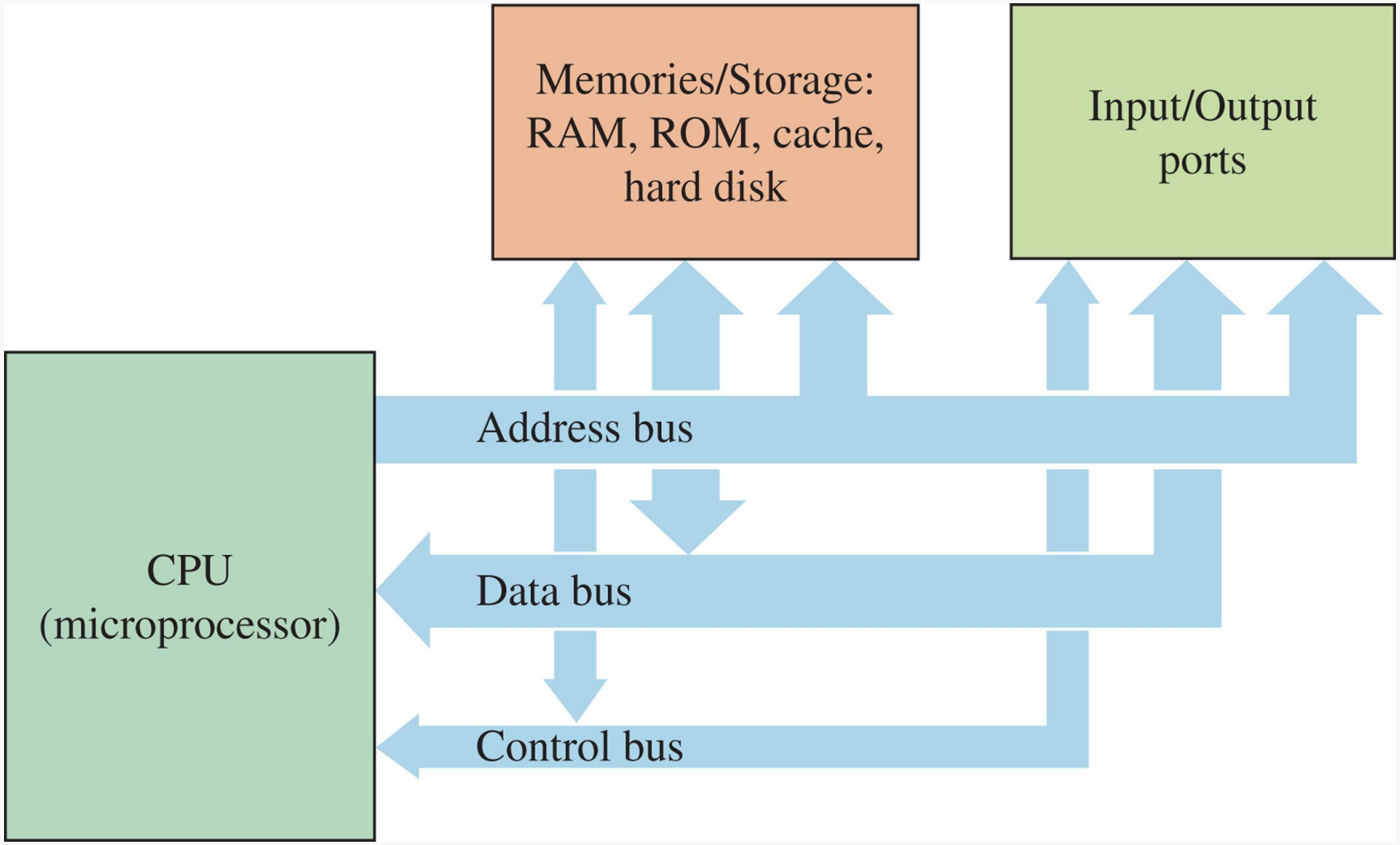
Høgskulen
på Vestlandet

ELE102 Programmering og mikrokontrollerer
Mikrokontroller

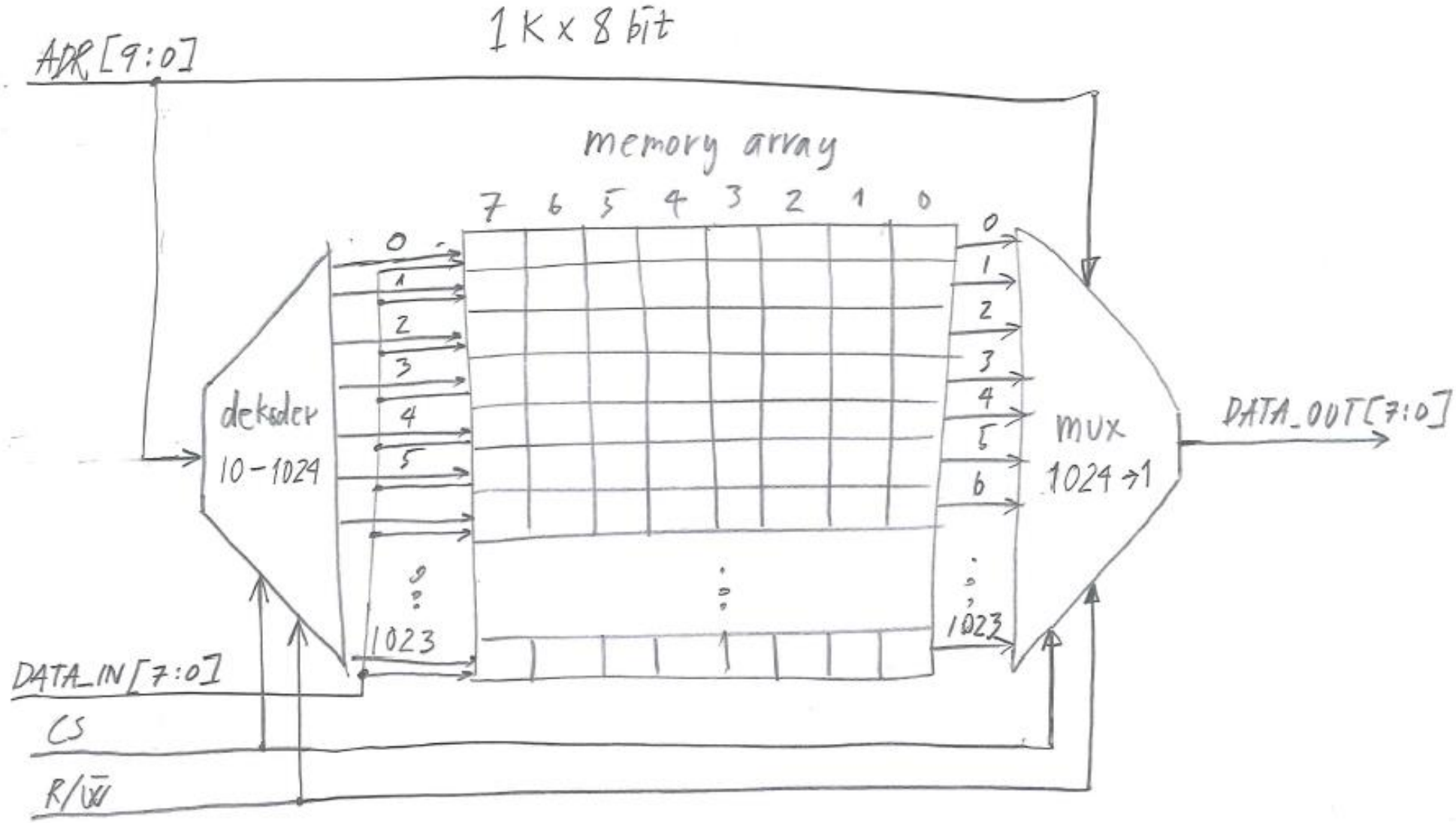
F9_000 Minne og minne-teknologi

Eivind Skjæveland
esk@hvl.no

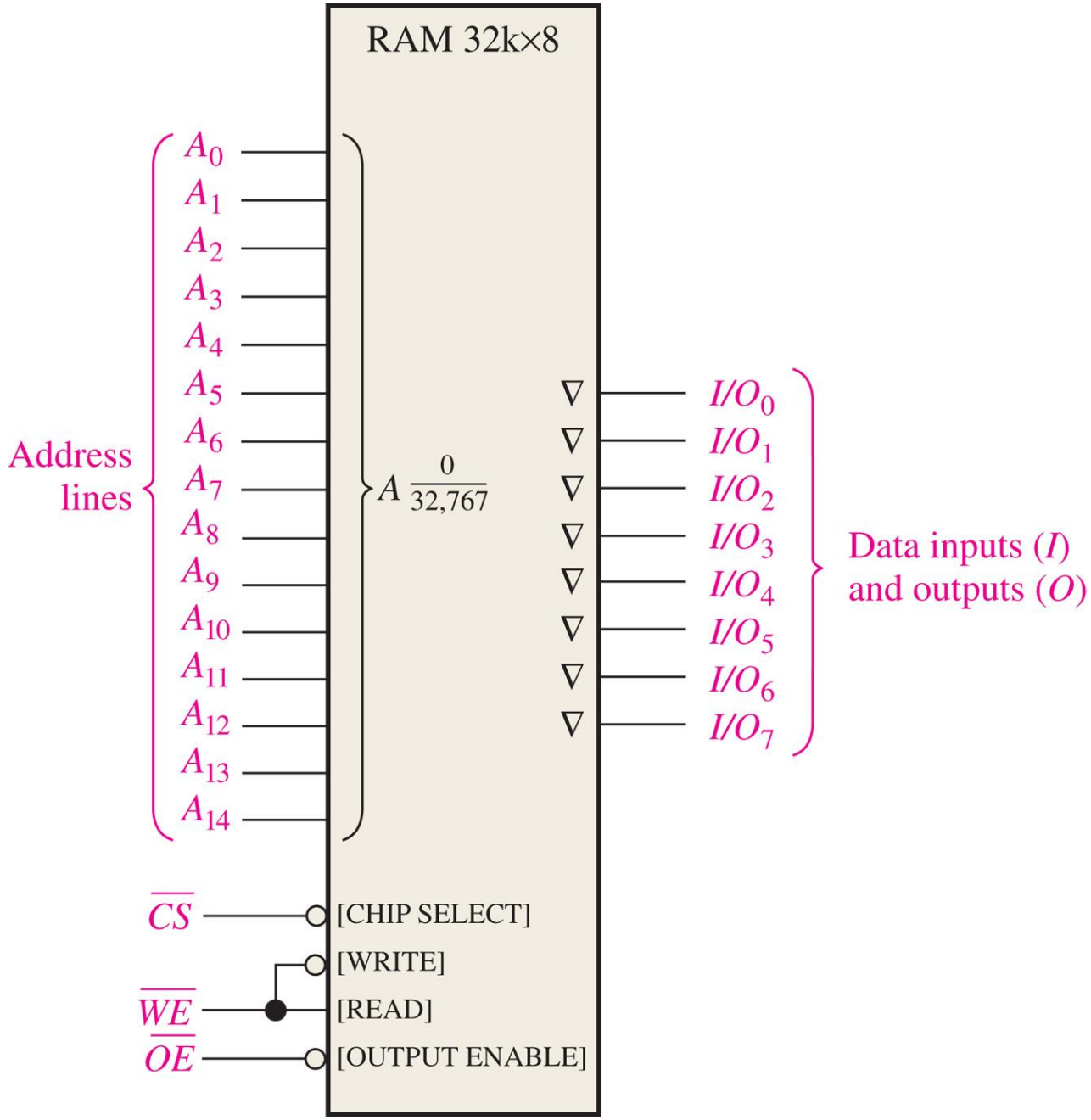
Enkel datamaskin



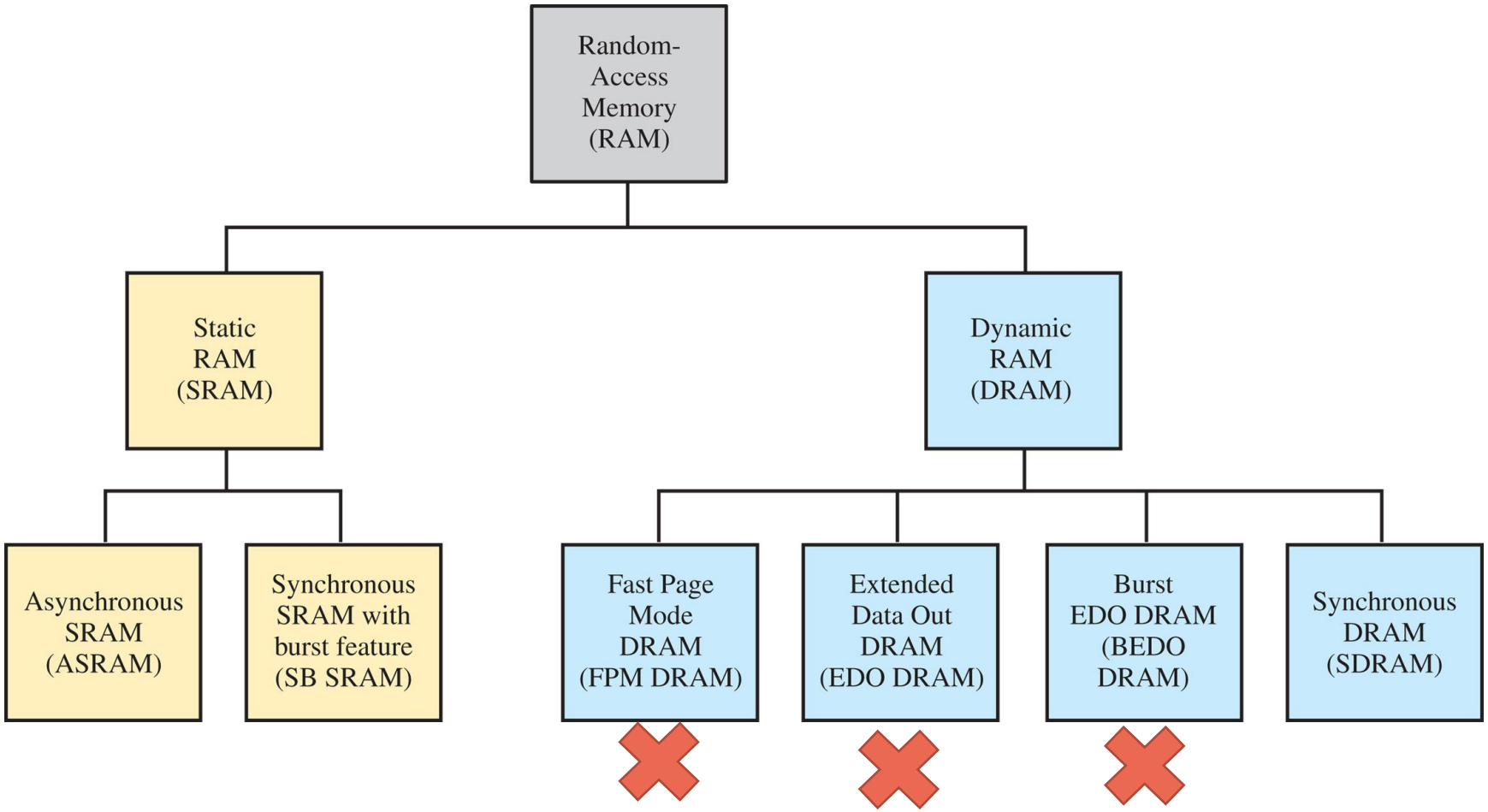
Grunnleggende minne arkitektur



32K x 8 asynkron SRAM – typisk I/O

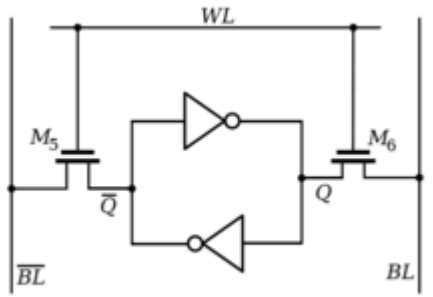


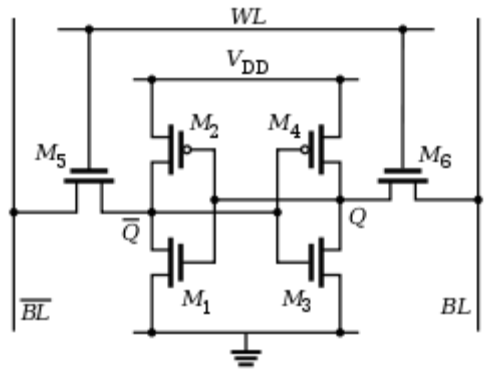
RAM familen


Random-Access Memory (RAM)
Static RAM (SRAM)
Dynamic RAM (DRAM)
Asynchronous SRAM (ASRAM)
Synchronous SRAM with burst feature (SB SRAM)
Fast Page Mode DRAM (FPM DRAM)
Extended Data Out DRAM (EDO DRAM)
Burst EDO DRAM (BEDO DRAM)
Synchronous DRAM (SDRAM)

SRAM

- ~ 6 CMOS transistorer/bit
- Lagres i bistabil latch/lås (2 stabile tilstander)
- Transitorene i invertere er små og tvinges/overstyres ved skriving
- Krever ikke refresh
- Voilatile (mister verdi uten spenning)
- Brukes til spesielle raske minne nær CPU
- Enklere å bruke enn DRAM (enklere å lage logikk for kontrol)

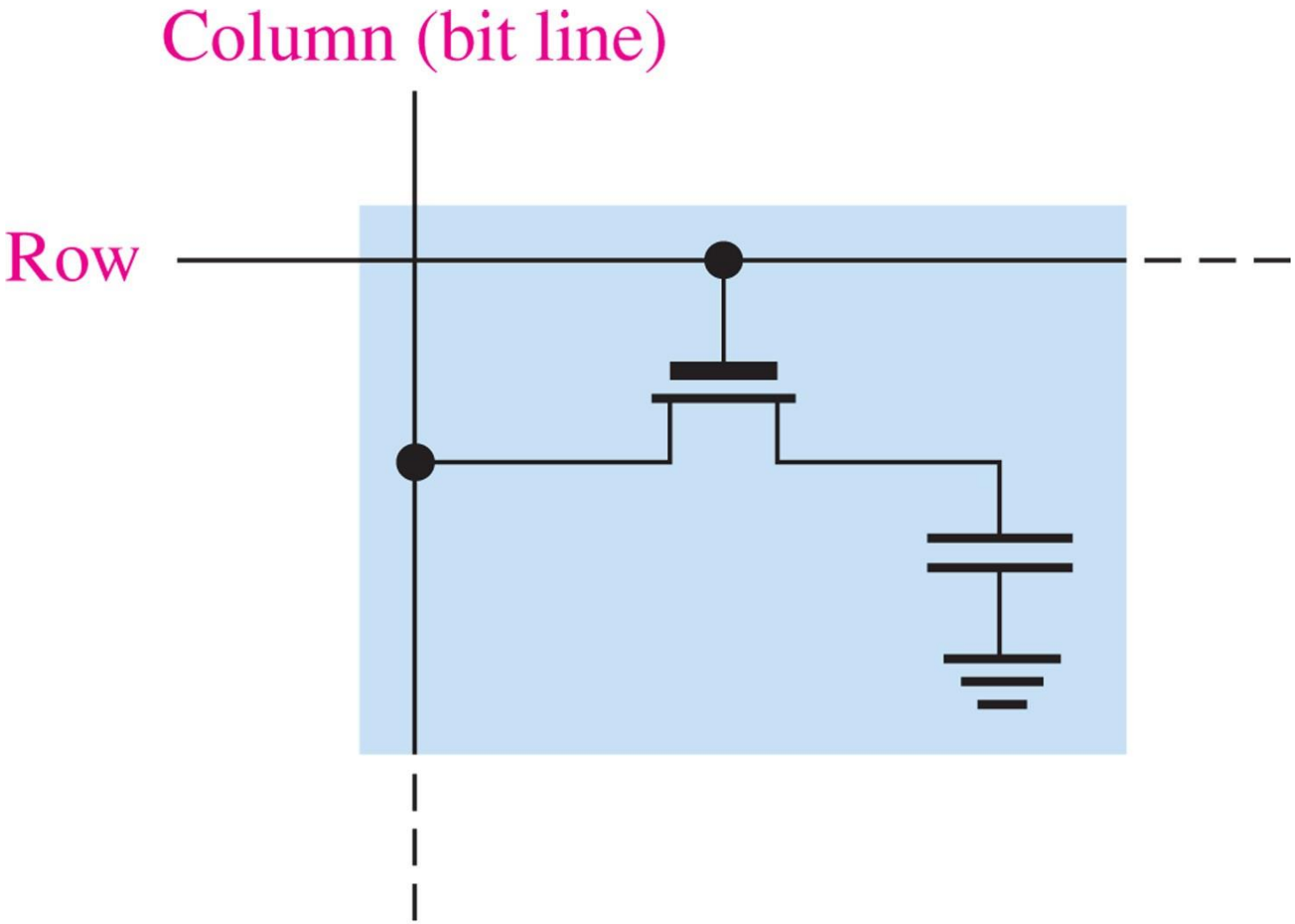




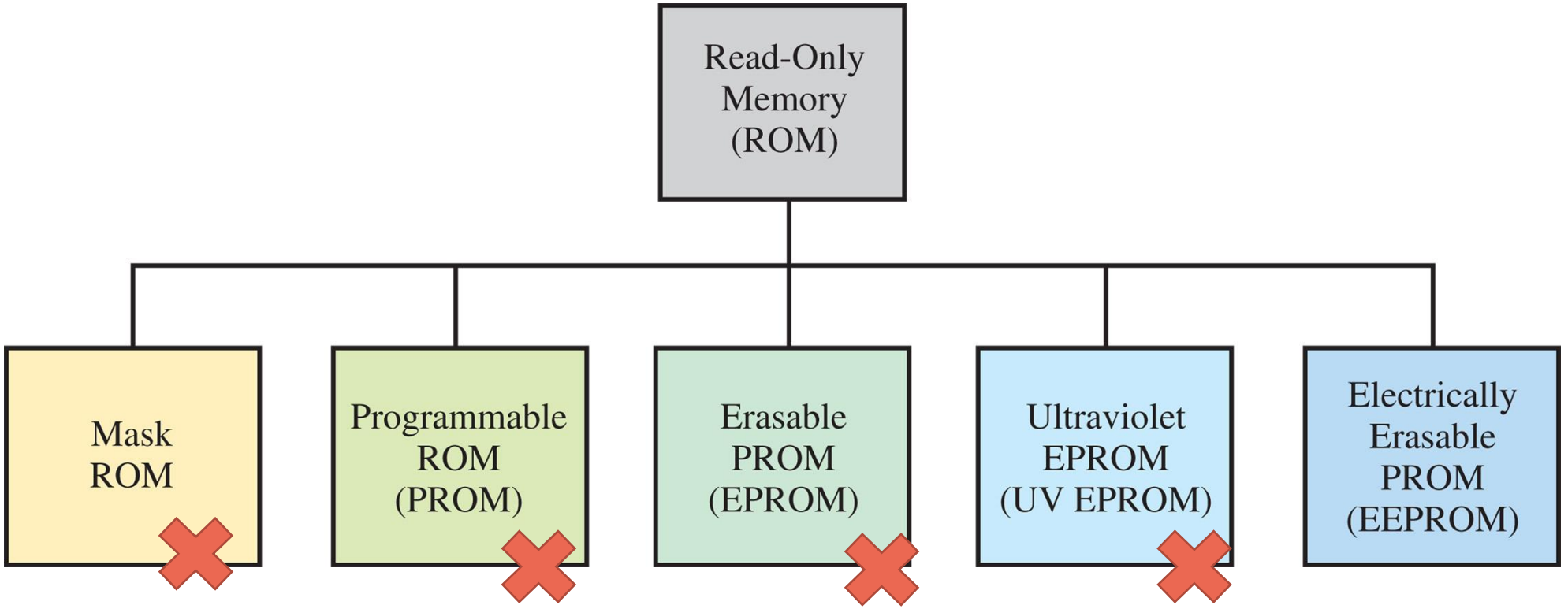
Kilde: Wikipedia

DRAM & SDRAM

- ~1 CMOS transitor/bit
- kapasitiv lagring
  - Oppladet =1
  - Inngen ladning = 0
- Må oppfriskes med jevne mellomrom
- Voilatile (mister lagret Verdi uten spenning på component)
- Krever litt annen teknologi enn den som brukes i mikrokontrollere og integreres vanligvis ikke med annen logikk

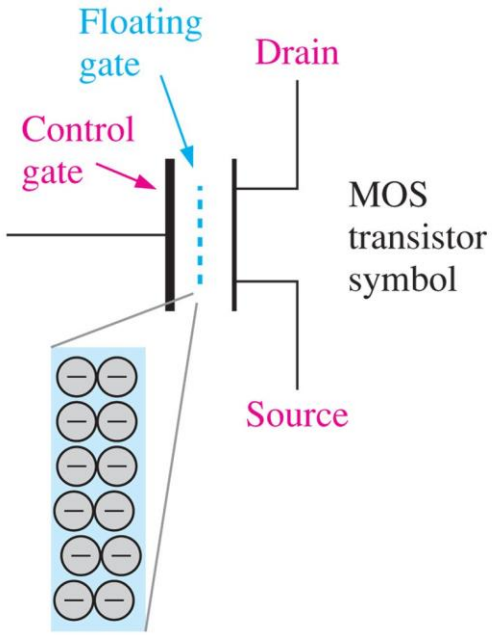


ROM familien – stort sett utkonkurert av Flash

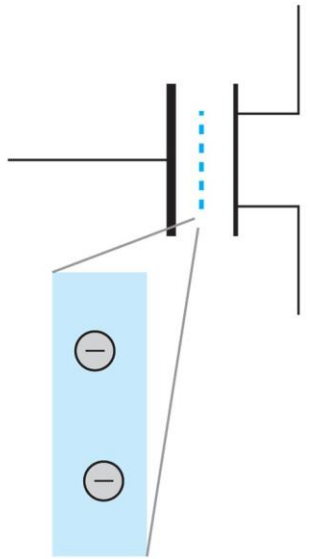


Flash

- ~1 CMOS transistor /bit
- Bit lagres som ladning på en “flytende” gate som ligger inne i et isolerende materale.
- Komplisert skrive mekanisme (bade å lage og forklare)
- Små lekkasje strømmer gir lagring I 20 år +
- Non voilatile (husker verdi selv om spenning slås av)
- Kompakt
- Brukes “overalt”
- Tåler ca 10000 – 100 000 lese-skrive-sykluser



Many electrons = more charge = stored 0.



Few electrons = less charge = stored 1.

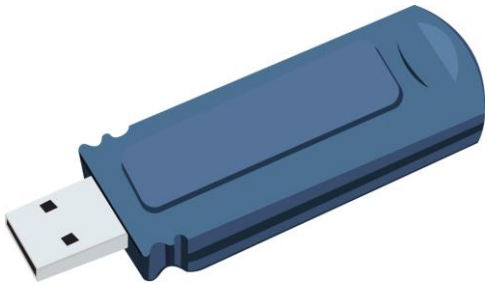
Sammenligningstabell

TABLE 11–2

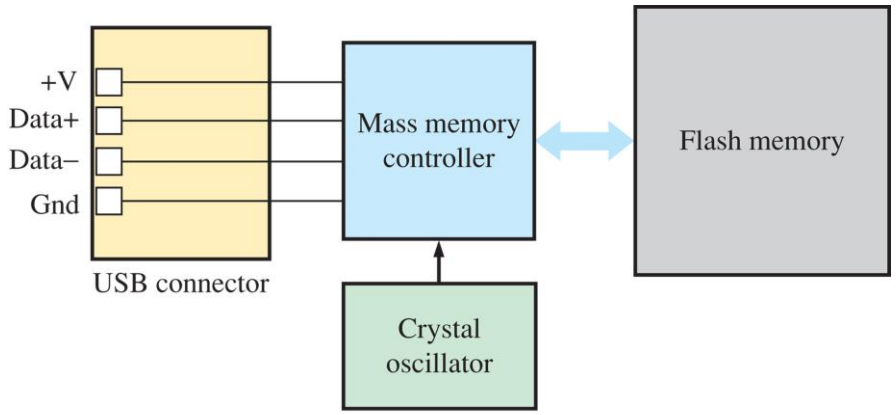
Comparison of types of memories.

Memory Type	Nonvolatile	High-Density	One-Transistor Cell	In-System Writability
Flash	Yes	Yes	Yes	Yes
SRAM	No	No	No	Yes
DRAM	No	Yes	Yes	Yes
ROM	Yes	Yes	Yes	No
EEPROM	Yes	No	No	Yes
UV EPROM	Yes	Yes	Yes	No

Flash Memory



(a) Typical USB flash drive

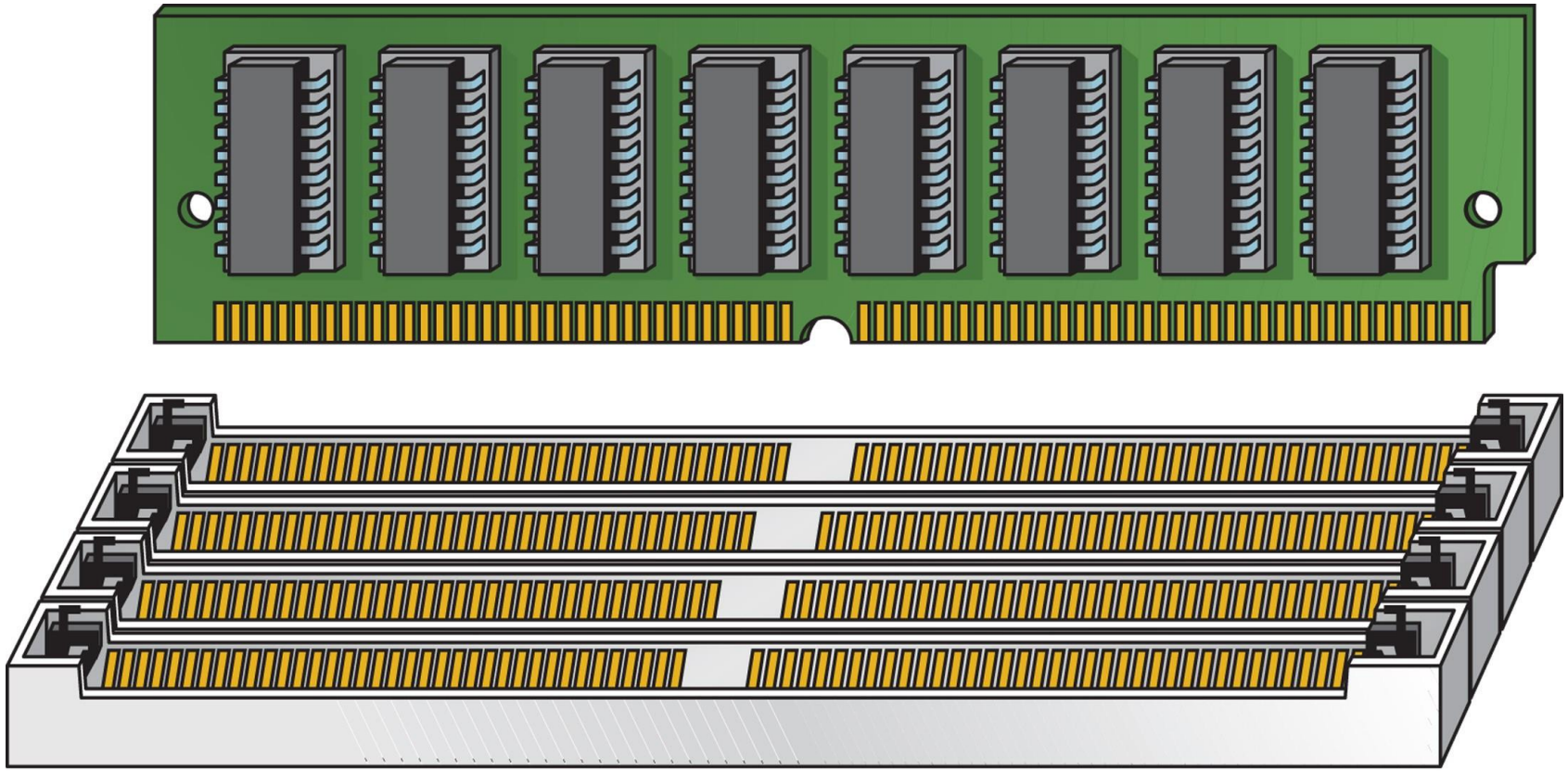


(b) Basic block diagram

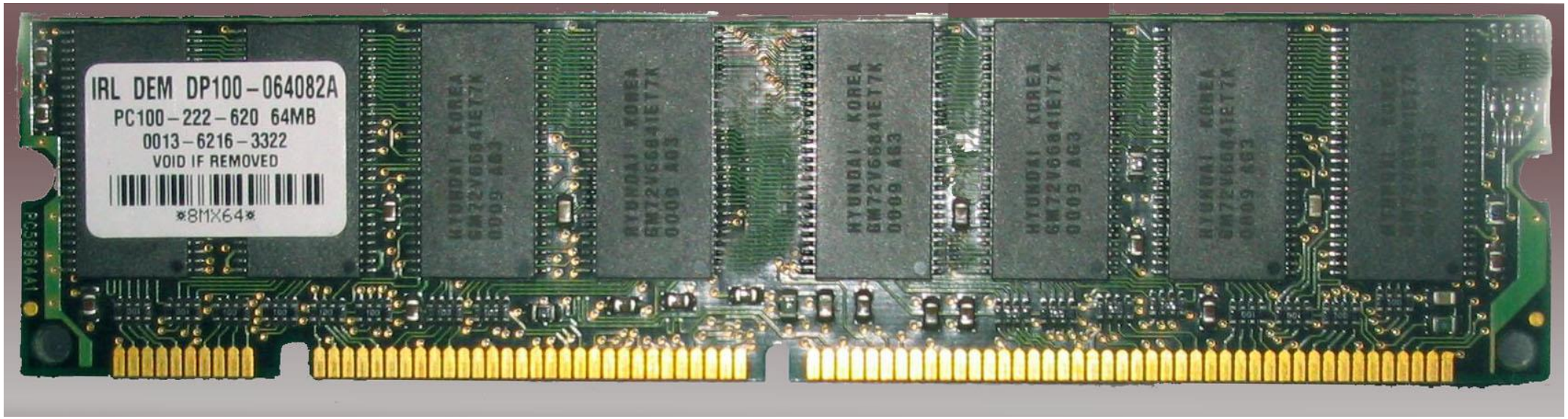
Minnekort


SanDisk
Ultra II
CompactFlash
SanDisk
4GB
MADE IN CHINA
SanDisk
Ultra II
LOCK
SD
2GB
SanDisk
SanDisk
2.0GB
microSD

Memory Moduler

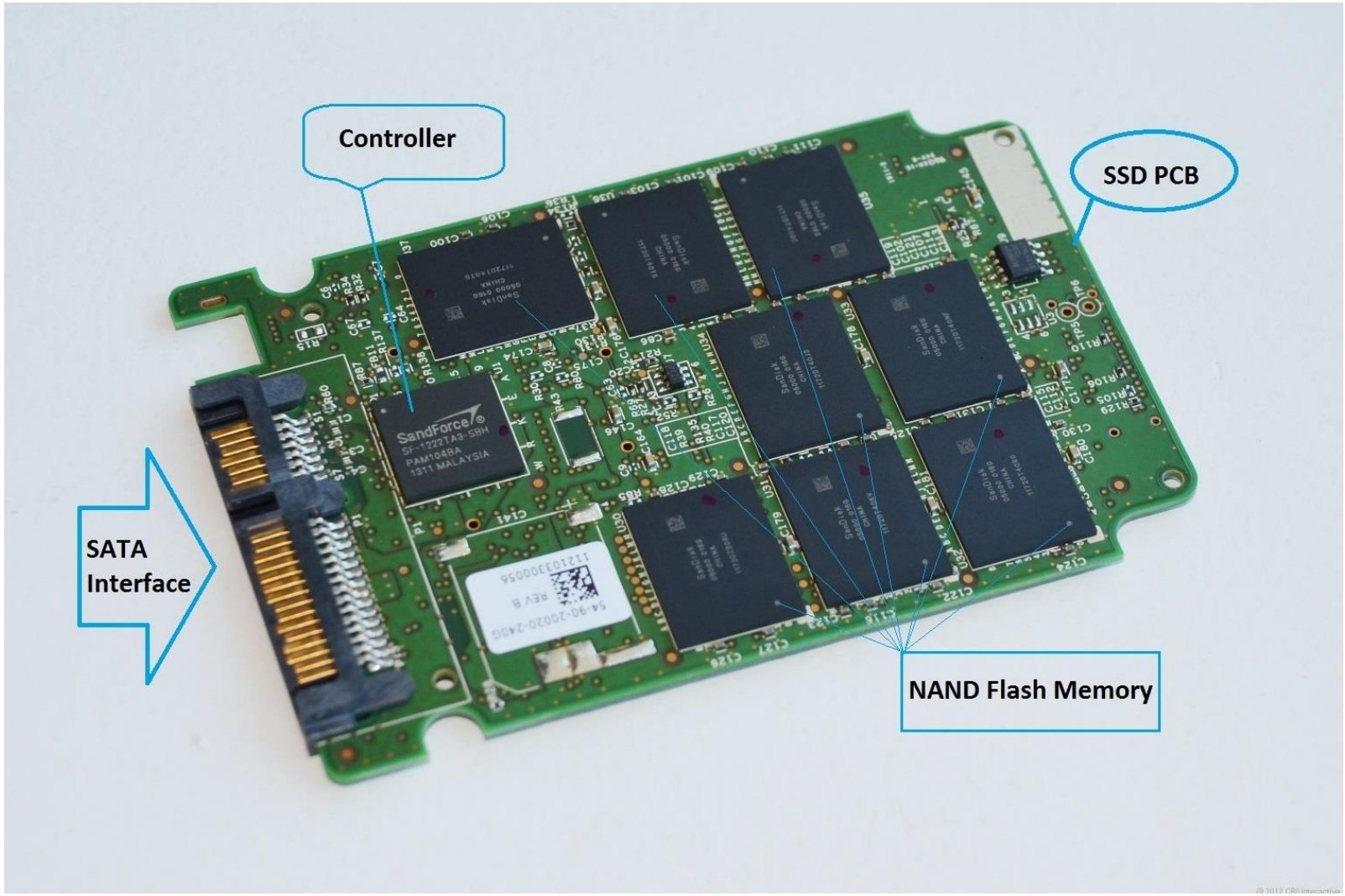


SDRAM modul


IRL DEM DP100-064082A
PC100-222-620 64MB
0013-6216-3322
VOID IF REMOVED
8M1X64
HYUNDAI KOREA
HYUNDAI KOREA
HYUNDAI KOREA
HYUNDAI KOREA

Magnetisk Harddisk

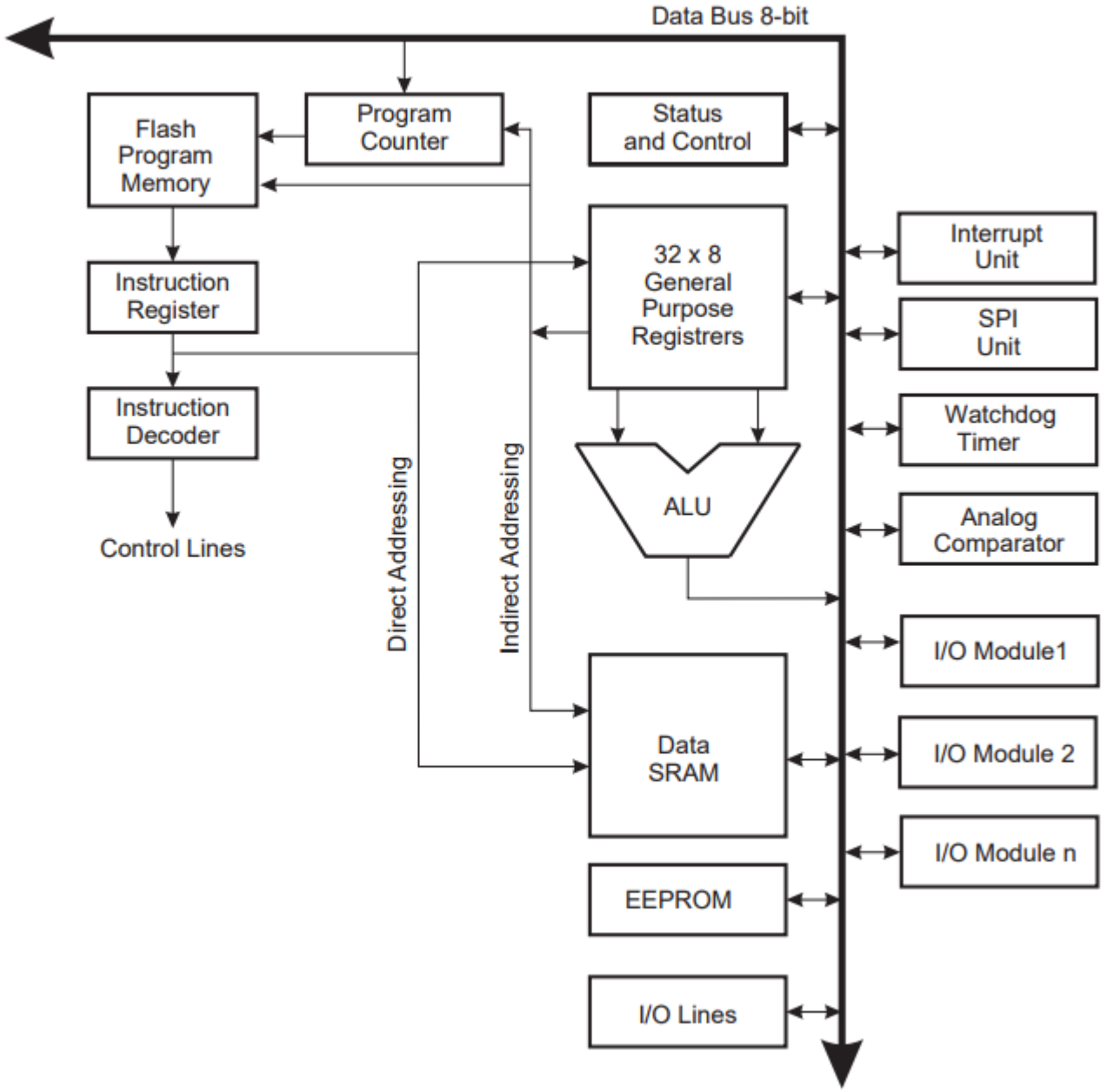
SSD Harddisk (Solid State Disk) med SATA grensesnitt



SSD Harddisk (Solid State Disk) med M2 grensesnitt

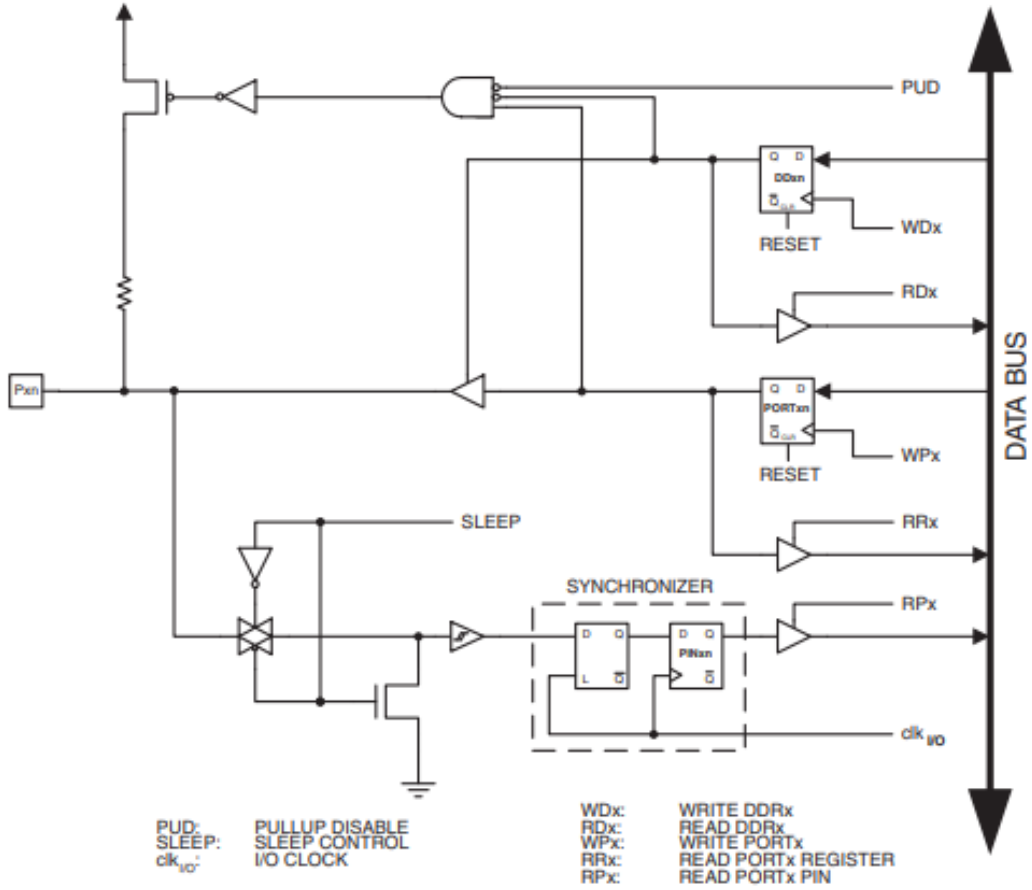
ATmega328P Blokk Diagram

Figure 7-1. Block Diagram of the AVR Architecture



ATmega328P – digital IO.port

Figure 23. General Digital I/O[1]



Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

ATmega328P Microcontroller

Features

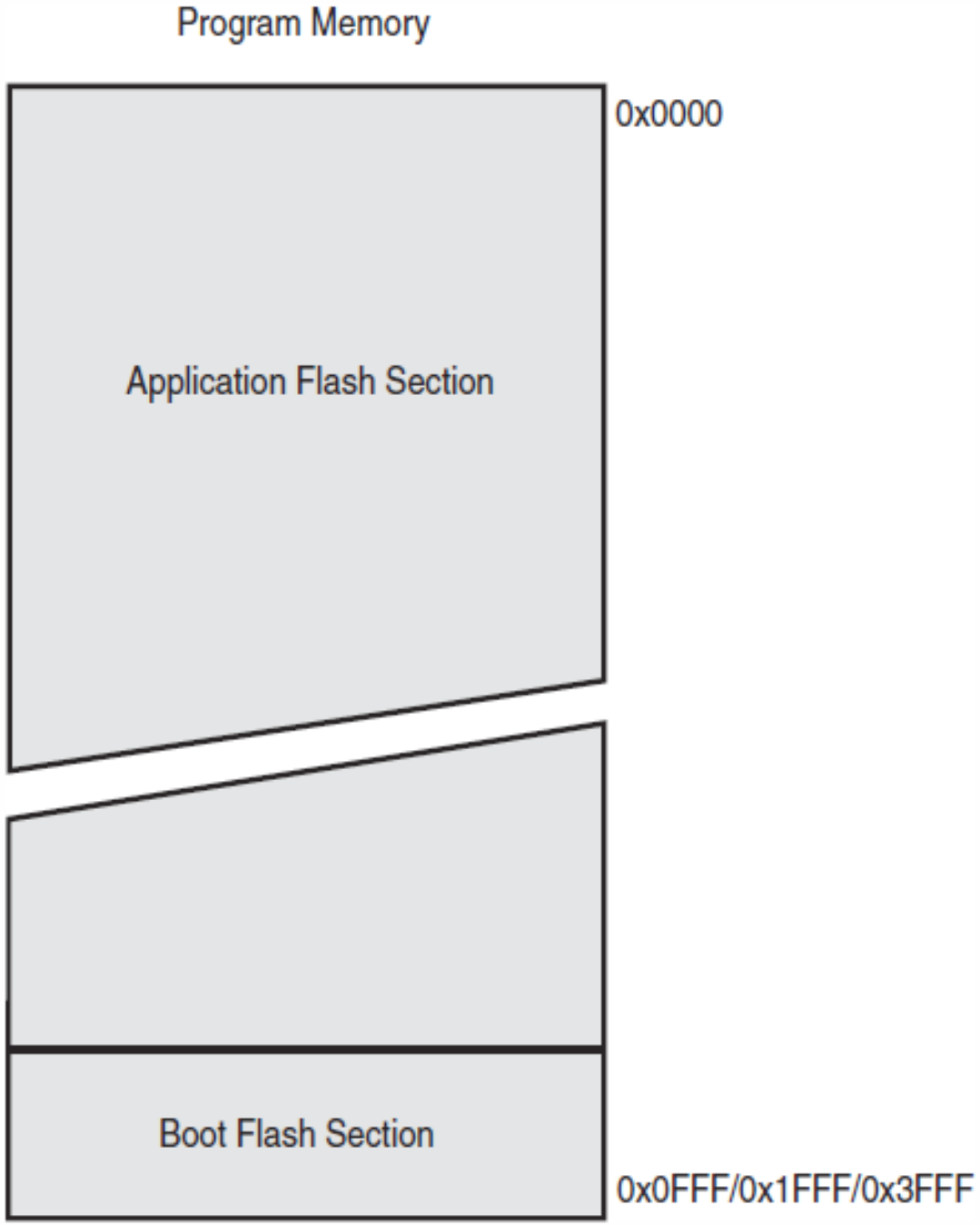
- High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller Family
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1KBytes EEPROM
  - 512/1K/1K/2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security

Program

Variable

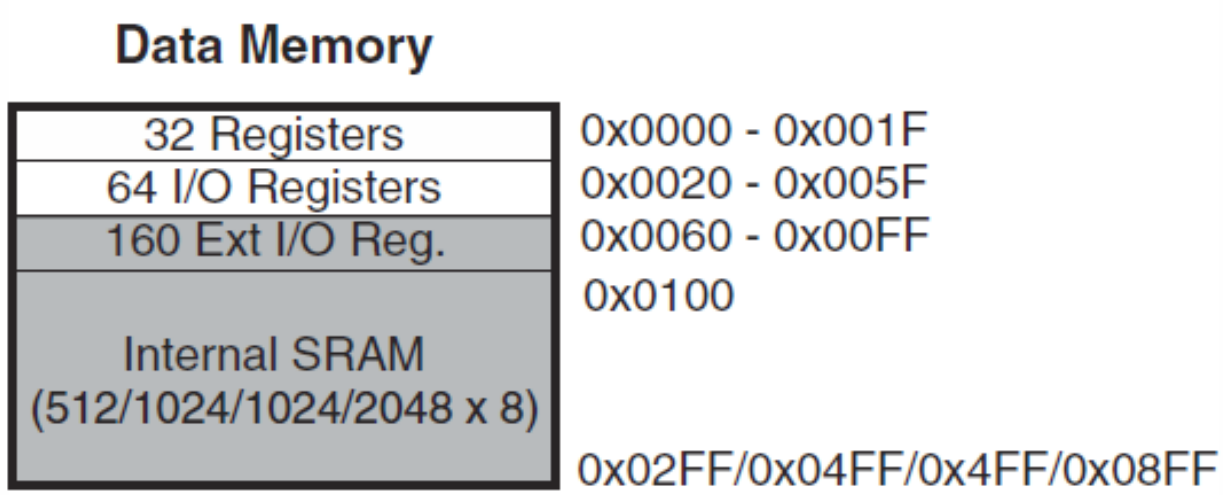
Flash Memory Map

- Eget adressemap for program
- Organisert som 16K x 16 bit



Data Memory Map

- Samme adresseområde for
  - Registere
  - Periferienheter
  - Intern SRAM
- EEPROM er tilgjengelig gjennom spesielle control registere (ikke direkte minne mappet)
  - Kan brukes til å lagre data som må huskes selv om Arduino mister spenning/blir restartet
  - #include <EEPROM.h>



Minnebruk rapport for blink eksempel

```
Archiving built core (caching) in: C:\Users\SVHA\AppData\Local\Temp\arduino_cache_846189\core\core_arduino_avr_uno_0c8
Sketch uses 928 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.
```

- Det blir så mange byte med maskinkode fordi vi bruker Arduino funksjoner.

RAM-bruk på Arduino

- Arduinoen har 2K byte med RAM
  - Brukt som arbeidsminne for programmene
  - Ikkje veldig mykje, pass på minnebruken.
- Vi bruker av RAM-en når:
  - Deklarerer globale variable
    - `char minMelding = "Hallo Studenter";`
  - Deklarerer lokale variable
    - blir frigjord når den lokale kodeblokka blir avslutta
  - Utfører funksjonskall

```
int minFunksjon(int innVerdi)
{
        int resultat;
        resultat = innVerdi*2;
        return resultat;
}
```

  - Når vi utvidar ein string (dynamisk tildeling av minne)
    - `String tekst = "Arduino-streng";`
    - `tekst += " Her legger vi til meir tekst i strengen. ";`

Minnebruk på Arduino

- Program som viser bruk av RAM:
- Prøv ut programmet
- Kva skjer når minnet er fullt?

RAMbruk §

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

String tekst = "kort tekst";
void loop() {
  // put your main code here, to run repeatedly:
Serial.println(tekst);
Serial.println("");

Serial.print("Ledig RAM: ");
Serial.println(memoryFree());

delay(1000);
tekst += " Denne strengen blir lengre og lengre";
}

extern int __bss_end;
extern void *__brkval;

//funksjon som returnerer kor mykje RAM som er ubrukt
int memoryFree()
{
  int freeValue;
  if ((int) __brkval == 0)
    freeValue = ((int)&freeValue) - ((int)&__bss_end);
  else
    freeValue = ((int)&freeValue) - ((int)__brkval);
  return freeValue;
}

```

Tiltak for å redusera RAM-bruk

- Ver forsiktig med å auka lengda på String i løkker
- Bruk konstantar i staden for variable, der det er praktisk.
  - int PWM-pinne = 6;
    - bruker RAM
  - const PWM_pinne = 6;
    - Bruker ikkje RAM, PWM-pinne blir bytta ut med 6 under kompilering.
      - Bruker av Flash-minnet, men det har vi mykje meir av.
- Bruk lokale variable så langt som råd.

EEPROM på Arduino

- Minne som ikkje mister innhald når Arduino mister spenning eller blir reprogramert
- 1024 byte EEPROM på Arduino Uno
- Bruk biblioteket <EEPROM.h>
  - EEPROM.write(adresse, val)
    - EEPROM.write(5,23); // Skriv verdi 23 i EEPROM-celle 5 // 1 byte
  - byte val = EEPROM.read(adresse);
    - byte val = EEPROM.read(adresse); val får verdien som er lagra i EEPROM-celle 5

EEPROM.h

- EEPROM.put(adresse, verdi);
  - float pi = 3.1415926;
  - EEPROM.put(16,pi) //Lagrar verdien pi (float, 4, byte) i celle #16-19

- EEPROM.get(adresse, verdi);
  - float *verdi*;
  - EEPROM.get(16, *verdi*); // hentar verdien som er lagra i celle #16 og vidare
    // antall byte bestemt av dataypen til *verdi*
- EEPROM.update(adresse, verdi);
  - int inn_data = analogRead(A0);
  - EEPROM.update(23, inn_data); // oppdaterer det som ligg i celle 23 om inn_data
    // er forskjellig frå det som ligg der frå før.
    // Grense for kor manga gonger vi kan skriva til EEPROM
    // EEPROM blir "utslitt"

EEPROM.h

- EEPROM.length(); // returnerer ## byte i EEPROM

- Kan bruka EEPROM som tabell:
  - EEPROM[4] = 42;
  - byte val = EEPROM[23];

```
#include <EEPROM.h>

void setup() {
  // put your setup code here, to run once:
 Serial.begin(9600);
}

void loop() {
  // put your main code here, to run
repeatedly:
EEPROM.write(5,23); // EEPROM(adresse,verdi);

byte val = EEPROM.read(5); //EEPROM(adresse);

Serial.println(val);
delay(2000);

float pi =3.1415926;
EEPROM.put(16,pi);

float verdi;
EEPROM.get(16,verdi);

Serial.println(verdi);

int data = analogRead(A0);
EEPROM.update(42,data);

Serial.println(EEPROM[16]);

Serial.println(EEPROM.length());
}
```

Bruk FLASH-minnet:

- Har vi for lite RAM, kan vi bruka flash-minnet til å lagra konstantar
  - Modifier PROGMEM
  - må vera globale konstantar,
  - les med pgm_read_byte_near();

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}
const PROGMEM char flashTekst[]= "Dette er ein lang tekst vi ikkje har plass til i RAM";

uint16_t k = 0;

void loop() {
  // put your main code here, to run repeatedly:
char bokstav = pgm_read_byte_near(flashTekst + k);
Serial.print(bokstav);
k++;
if (k>=strlen_P(flashTekst))
{
  k = 0;
  Serial.println();
}
delay(100);
}
```

Oppgåve EEPROM

1. Lag eit program som nullstiller innhaldet i EEPROM.
   - Last opp på Arduino, kjøyr.
2. Lag program som
   a) Les verdien på analogport A0 kvar 5 sekund
      - Kople til f. eks potmeter, temperatursensor eller fototransistor
   b) Skriv resultatet til seriell port
   c) Lagrar verdien til EEPROM
      - Husk å tildela ny verdi till ny adresse
      - Husk: int bruker 2 byte med data.
   d) Kjør programmet, samle data i ca 1 minutt.
3. Lag nytt program som
   a) Les data frå EEPROM
   b) Skriv data til serieport.

- Nyttige funksjonar
  - digitalRead()
  - analogRead()
  - EEPROM.length();
  - EEPROM.write();
  - EEPROM.put();
  - EEPROM.get();
  - Serial.println();

F9_000 Minne