# Memorii în Sisteme Embedded

Tipuri de memorii. Arhitectura memoriilor.

# De ce avem nevoie de memorii în sisteme embedded?

- Stocarea datelor
  - Temporar
  - Pe termen lung
- Comunicare între componentele programelor
- Comunicare cu senzori și actuatoare

- Ce tipuri de memorii cunoașteți?

# Constrângeri față de sistemele PC

- **Dimensiuni reduse** – datorită costurilor și a constrângerilor de spațiu
- **Consum de putere** – preferabil cât mai redus
- **Fiabilitate și siguranță** – necesare în special în aplicații critice.
  (*Sistemele embedded nu au voie să cedeze!)*

# Arhitecturi de memorii

- Tipuri de memorii:
  - Volatile, non-volatile
- Hărți de memorie:
  - Paginare, Arhitecturi Hardvard, I/O mapat în memorie
- Organizarea memoriei:
  - Alocare statică, Stivă, Heap
- Ierarhii de memorii:
  - Cache, scratchpad, memorie virtuală
- Mecanisme de protecție:
  - ECC, securitate

# Tipuri de memorii

## Random Access Memory (RAM)

- Pot fi citite și scrise fără mecanisme de ștergere
- Memorie volatilă: își pierde conținutul după scoaterea de sub tensiune
- Imediat după alimentare conține valori aleatoare
- Folosite pentru manipularea datelor temporare folosite de programe sau pentru încărcarea temporară a programelor din memoria non-volatilă

## Read Only Memory (ROM)

- Citirea nu necesită pași suplimentari
- Pentru scriere este necesară reprogramarea care presupune operațiuni speciale (ștergere + scriere)
- Memorie non-volatilă = își păstreaza conținutul chiar și fără alimientare
- Folosite pentru stocarea programelor sau a datelor
- Număr limitat de rescrieri

# Tipuri de memorii RAM

- **SRAM** (Static Random-Access Memory):
  - Rapide, timp de acces deterministic
  - Consum mai mare de putere și densitate redusă față de DRAM
  - Folosită pentru cache, scratchpad și memorii mici embedded
- **DRAM** (Dynamic Random-Access Memory):
  - Latențe mai mari decât în cazul SRAM
  - Timpul de acces depinde de secvența adreselor
  - Densitate mai mare față de SRAM -> capacitate mai mare
  - Necesită reîmprospătare periodică (~ 64ms)
  - Folosită în general ca memorie principală

# Securitate: Remanența datelor in RAM

## Remanența datelor din RAM este afectată de temperatură

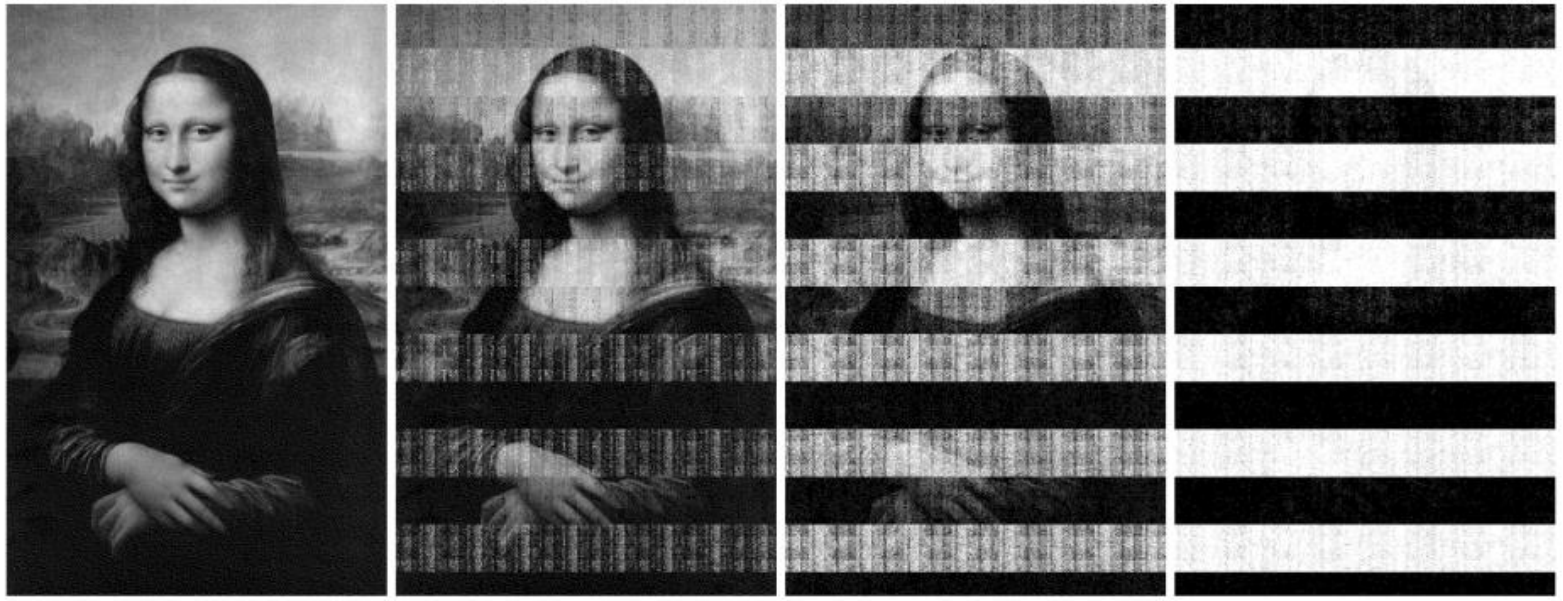


Figure 4: We loaded a bitmap image into memory on Machine A, then cut power for varying lengths of time. After 5 seconds (left), the image is indistinguishable from the original. It gradually becomes more degraded, as shown after 30 seconds, 60 seconds, and 5 minutes.

Halderman, J. Alex, et al. ”Lest we remember: cold-boot attacks on encryption keys.” Communications of the ACM 52.5 (2009): 91-98.

# Tipuri de memorii ROM

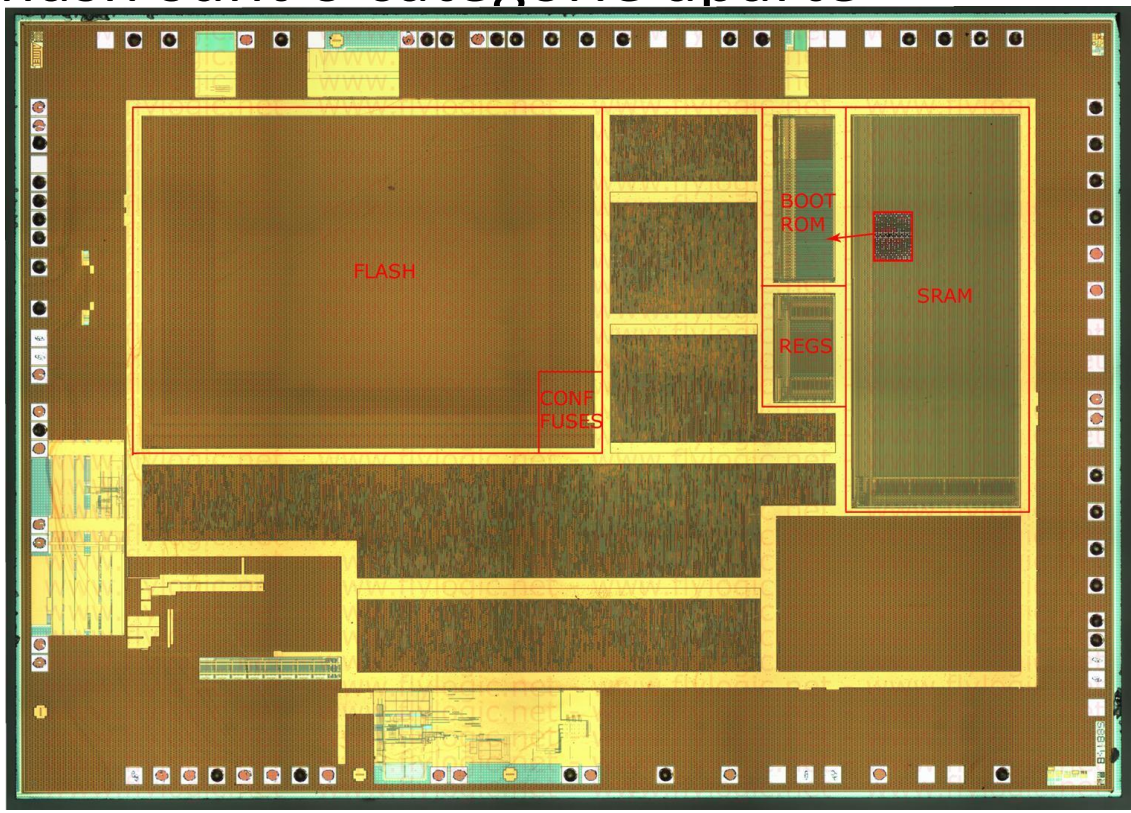
- Primele tipuri de memorii ROM nu erau reprogramabile:
  - ROM - pre-programate în cursul procesului de fabricație
  - PROM (Programmable ROM) - programabile o singură dată
- EPROM (Erasable Programmable ROM) - pot fi șterse prin expunere la lumină UV
- EEPROM (Electrically EPROM)



IDT6116SA datasheet

# Memorii Flash

- Din punct de vedere tehnic, memoriile flash sunt o categorie aparte de memorii EEPROM
- EEPROM – reprogramabil la nivel de blocuri de mici dimensiuni (byte/word)
- Flash – reprogramabil în blocuri/pagini de mari dimensiuni (KB, MB)
- Tipuri de memorii Flash
  - NAND
  - NOR

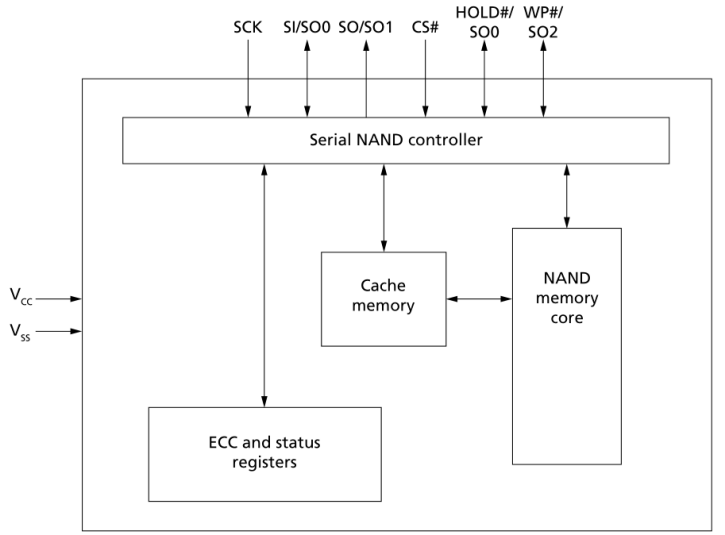


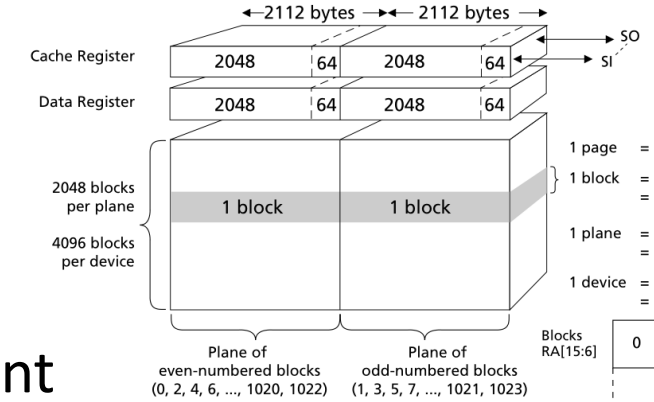

MSP430 decalpsulat
(http://43oh.com/2010/12/chip-porn-msp430-with-it-packaging-off/)

# Memorii NAND

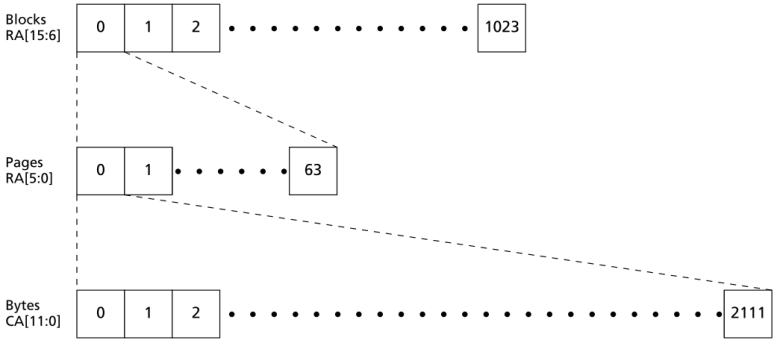
- Avantaje:
  - Rapiditate în ștergere, scriere și citire
  - Cicluri de ștergere 100.000-1.000.000
  - Durată de viață – de 10 ori mai mare decât în cazul NOR
  - Preț scăzut
  - Dimensiuni tipice mari sute de MB, GB
- Dezavantaje:
  - Siguranță scăzută, necesită management pentru Bad blocks
  - Utilizare complexă

Micron NAND Flash (SPI) - MT29F1G01AADD



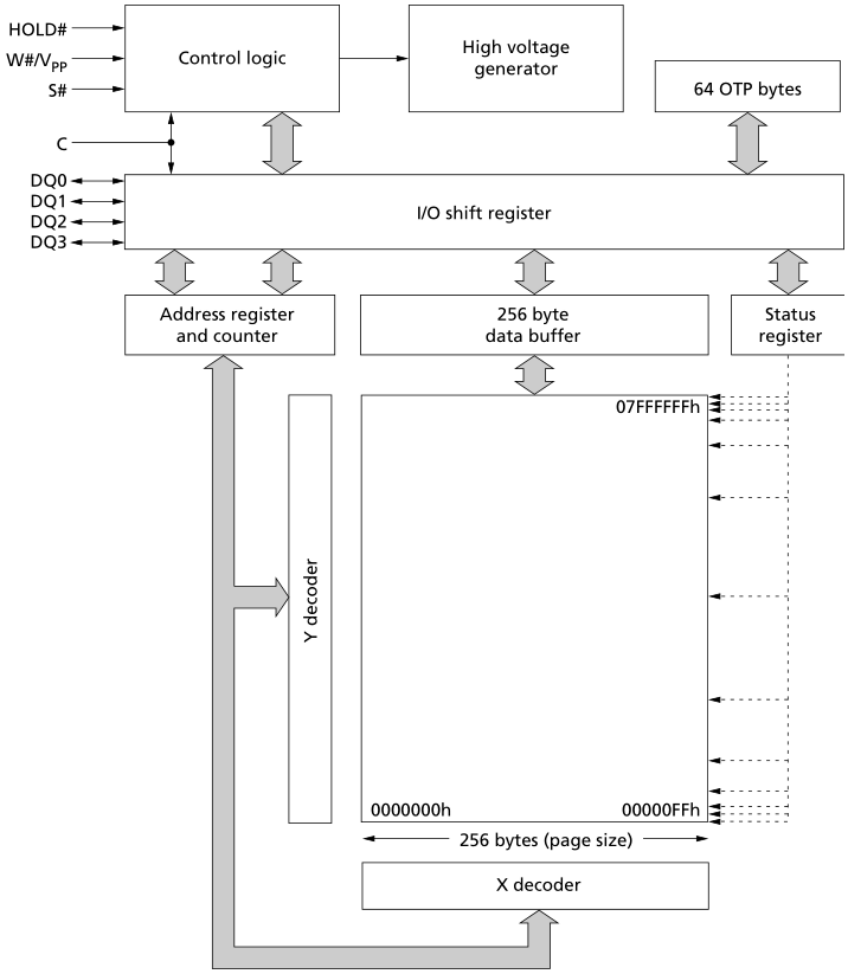

1 page = (2K + 64 bytes)
1 block = (2K + 64) bytes x 64 pages = (128K + 4K) bytes
1 plane = (128K + 4K) bytes x 512 blocks = 528Mb
1 device = 528Mb x 2 planes = 1056Mb



# Memorii NOR

- Avantaje:
  - Ceva mai rapid în citire față de NAND
  - Siguranță mai superioară față de NAND
  - Permite acces aleator
  - Ușurință în utilizare
- Dezavantaje:
  - Foarte încet în ștergere, încet în scriere
  - Cicluri de ștergere 10.000-100.000
  - Durată de viață – 10% din durata de viața a NAND
  - Preț scăzut
  - Dimensiuni tipice reduse: zeci de MB

Micron NOR Flash (SPI) - N25Q00AA



# Organizarea memoriei interne – spațiul de adresare

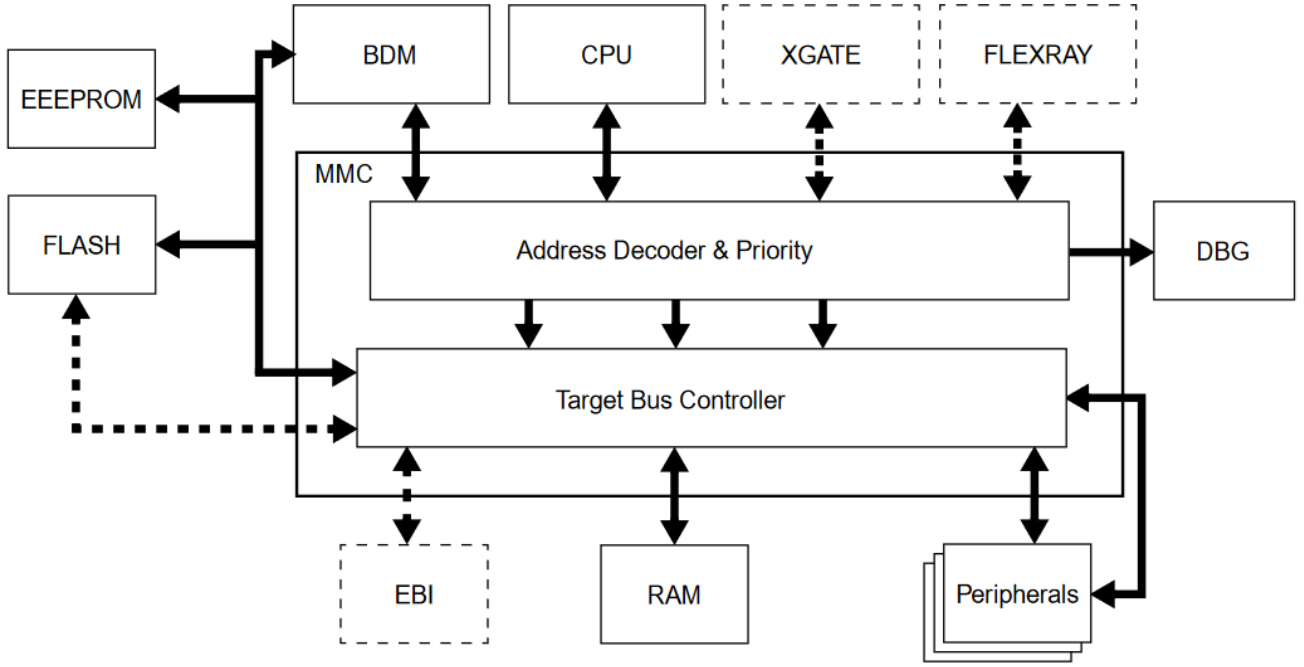
- memoriile folosite într-un sistem trebuie să fie mapate în zona adresabilă de către CPU
- capacitatea de adresare a unui CPU e dată de lățimea bus-ului de adrese
- o parte a spațiului de adresare este rezervată pentru regiștri și vectorul de întreruperi
- exemplu: un CPU cu 16 linii de adrese poate adresa zone de memorie într-un spațiu de $2^{16}$ = 64KBytes

# Organizarea memoriei interne: extinderea capacității de adresare

- adresarea directă este limitată de numărul de linii de adrese
- capacitatea de adresare a unui CPU se poate extinde prin paginarea memoriei:
  - o parte a spațiului de adresare este folosită ca o fereastră
  - prin fereastra de memorie se poate adresa direct la un moment dat o porțiune dintr-o zonă mai mare de memorie
  - schimbarea paginii controlează zona accesibilă prin fereastră

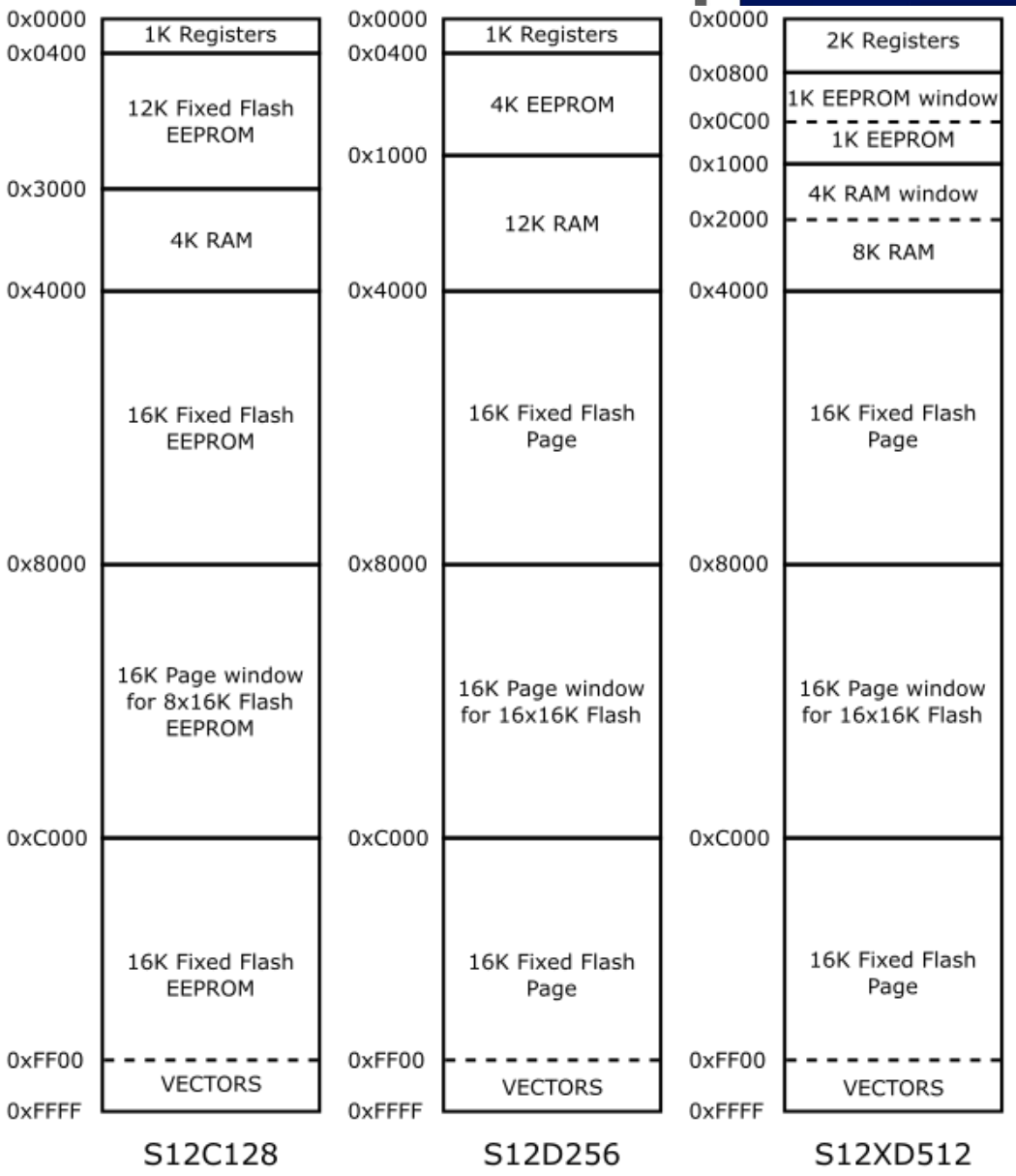
# Studiu de caz - Memoria internă a microcontrollerului S12

- Microcontroller-ul S12
  - microcontroller pentru aplicații automotive și industriale
  - microcontroller pe 16-biți
  - memorie internă: 1-64KB RAM, 16-1000KB Flash, 0-8KB EEPROM



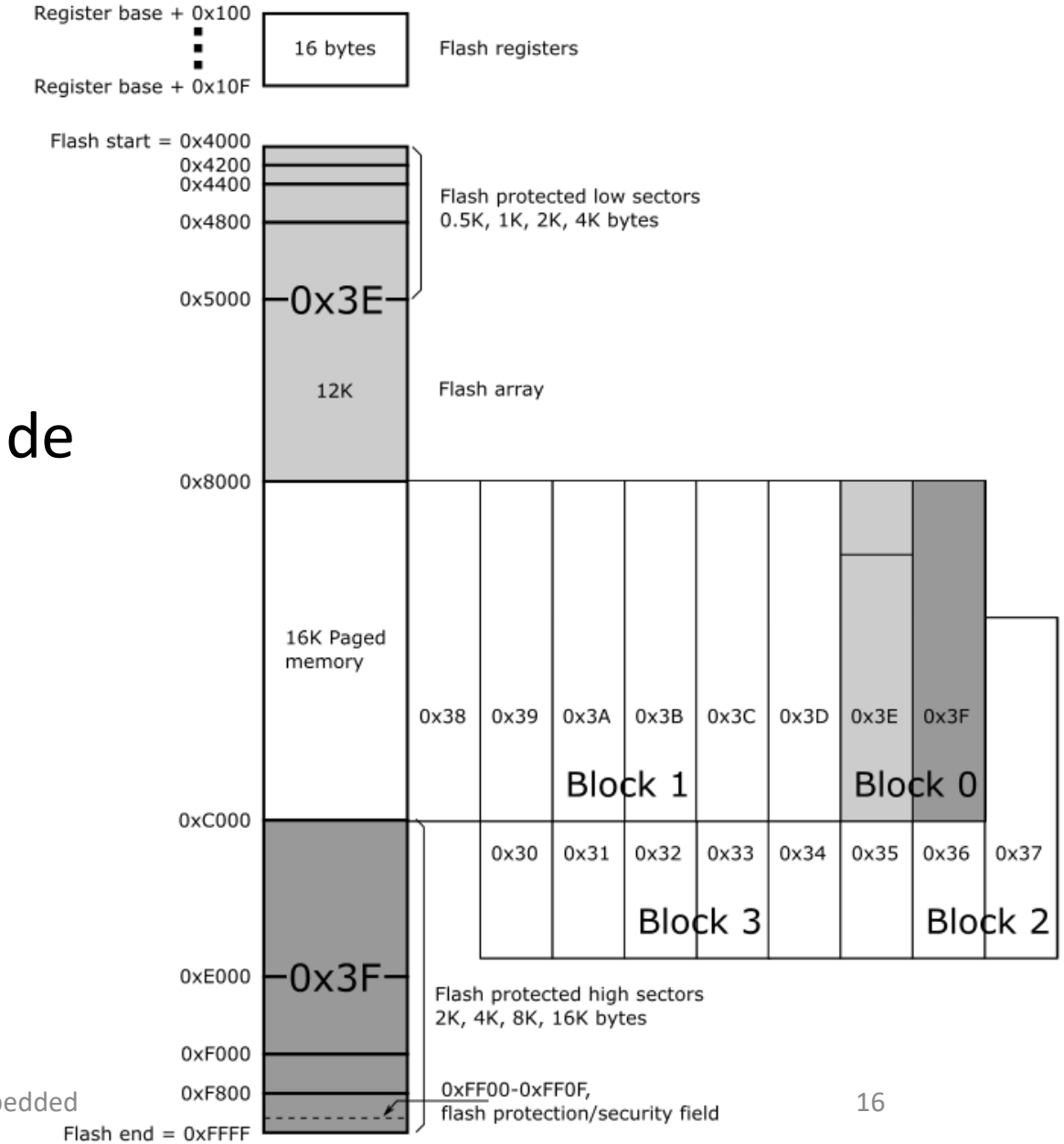
# Studiu de caz - Memoria internă a microcontrollerului S12

- Spațiul de adresare limitat la 64KB
- Folosește paginare pentru accesarea memoriei Flash și EEPROM



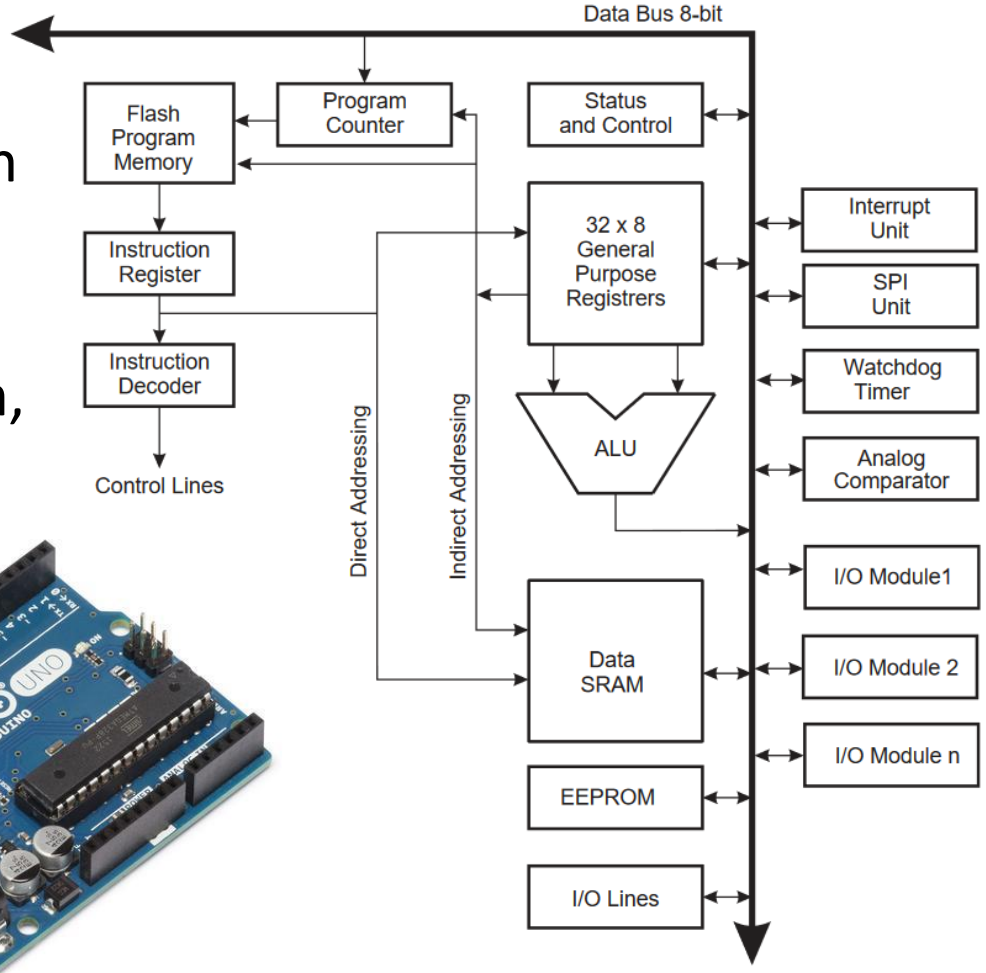
# Studiu de caz - Memoria interna a microcontrollerului S12

- Dimensiune ferestrei - 16K
- Memoria Flash este divizată în blocuri de dimensiunea ferestrei = 16K



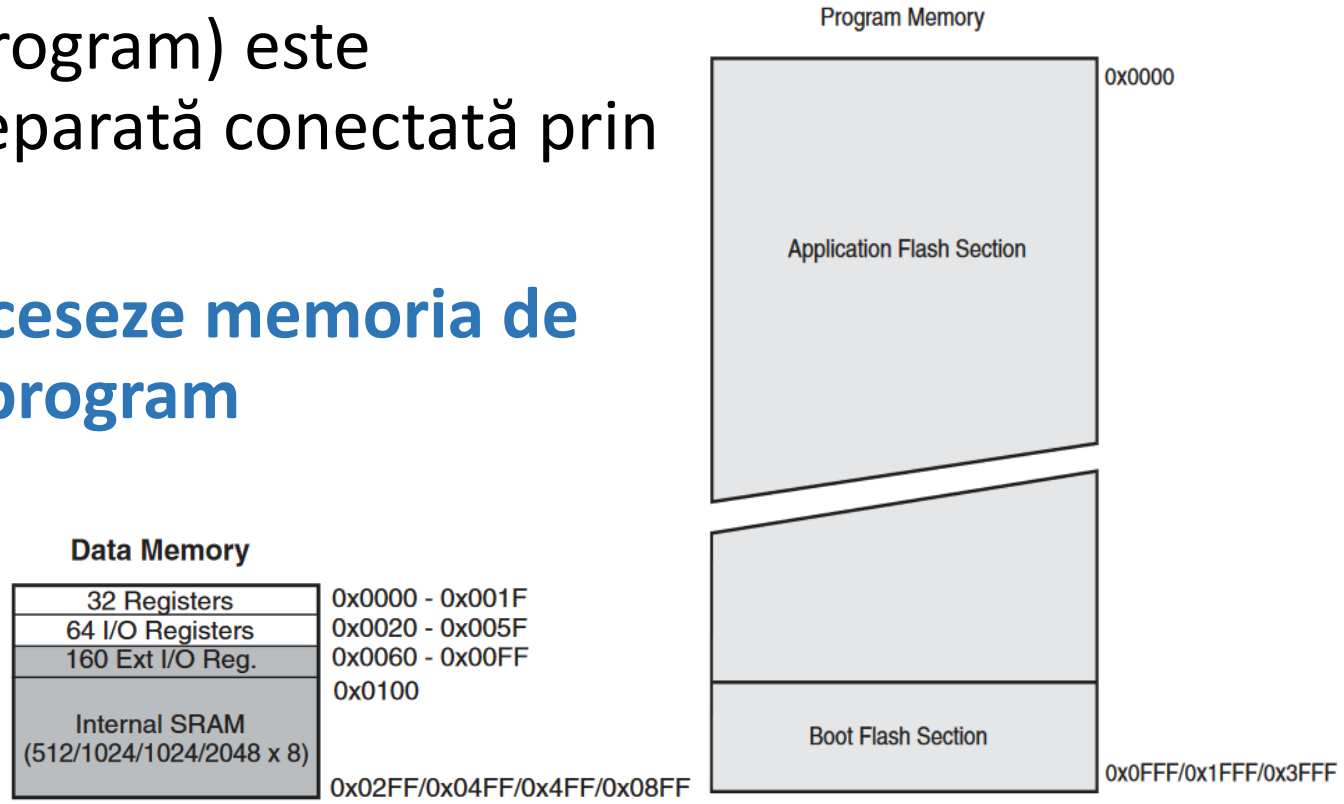
# Un alt exemplu: ATmega48/88/168/328

- Microcontrollerul Atmega:
  - Binecunoscut datorită plăcilor de dezvoltare din gama Arduino
  - Microcontroller pe 8 biți, adresare pe 16-biți
  - Memorie internă: 512B-2KB RAM, 4-32KB Flash, 256B-1KB EEPROM



# Un alt exemplu: ATmega48/88/168/328

- Arhitectură Harvard – fiecare tip de memorie (memorie de date și memorie program) este considerată ca fiind o entitate separată conectată prin linii distincte
- Avantaj: **procesorul poate să acceseze memoria de date concomitent cu memoria program**



# Întrebări

- De ce spunem că S12 este un microcontroller pe 16 biți? Dar Atmega unul pe 8?
- De ce folosim memoria volatilă? Nu am putea să folosim doar memorie non-volatilă?

# Utilizarea memoriei de date

- Memorie alocată static – Compilatorul alege adresa la care să stocheze o variabilă
- Stivă – Memorie alocată dinamic pe principiul LIFO cu management automat
- Heap – Memorie alocată dinamic fără management automat

# Memorie alocată static

```
char c;
int main(void) {
     c = 'D';
     …
}
```

Compilatorul alege adresa la care să stocheze variabila c, variabila fiind accesibilă pe toată durata execuției programului din orice context.

```
void example(void) {
     static char c = 'D';
     …
}
```

Compilatorul alege adresa la care să stocheze variabila c, variabila fiind accesibilă pe toată durata execuției programului doar în cadrul funcției example.

# Variabile în stivă (“variabile automate”)

```
void example(void) {
     char c = ‘D’;
     …
}
```

Apelul funcției example duce la atribuirea unei adrese în stivă pentru c prin decrementarea lui SP (stack pointer). Ieșirea din funcție duce la eliberarea memoriei (SP incrementat) variabila existând doarpe durata execuției funcției example().

# Alocarea dinamică a memoriei în Heap

Sistemele de operare oferă posibilitatea alocării dinamice a memoriei într-o zonă numită “heap”.

Managementul defectuos al memoriei (malloc(), free()) poate cauza multe probleme în sisteme embedded:

- Memory leak (memoria alocată nu este eliberată)
- Fragmentarea memoriei (segmentele alocabile devin tot mai mici)

Tehnici automate (“garbage collection”) necesită de multe ori oprirea execuției și reorganizarea memoriei alocate. Inacceptabil pentru aplicații în timp-real.

# Ierarhii de memorii

## Cache

- Un segment relativ mic de memorie implementat de obicei în SRAM ce reține date recent utilizate pentru a ajuta la creșterea vitezei de acces la date

## Scratchpad

- Zonă de memorie rapidă folosită la memorarea temporară a unor calcule, date sau alte componente în curs de operare
- Software-ul dictează ce se memorează în scratchpad

# Mecanisme de protecție - ECC

## ECC (Error Correction Codes)

- Unele memorii includ module ECC
- Necesită locații suplimentare de memorie și circuistică pentru identificare și corecție
- cele mai des întâlnite pot detecta și corecta erori de 1 bit/word și pot detecta erori de 2 biți
- costuri suplimentare

# Mecanisme de protecție - Securitate

## Securitate – Protecția zonelor de memorie

- anumite sectoare din memoria FLASH și EEPROM pot fi protejate la scriere
- sectoarele protejate nu pot fi șterse decât prin ștergerea întregului chip de memorie
- introducerea sau înlăturarea protecției se face prin setarea regiștrilor asociați