Arduino - Organizzazione della memoria

(Prof. Fischetti P.)

Ci sono tre pool di memoria nel microcontrollore utilizzato su schede Arduino basate su avr:

Flash, SRAM e EEPROM

La memoria flash (spazio del programma), è il luogo in cui vengono archiviate le istruzioni dello sketch.

SRAM (memoria ad accesso casuale statico) è dove vengono create e manipolate le variabili quando viene eseguito uno sketch.

EEPROM è lo spazio di memoria che i programmatori possono utilizzare per memorizzare informazioni a lungo termine.

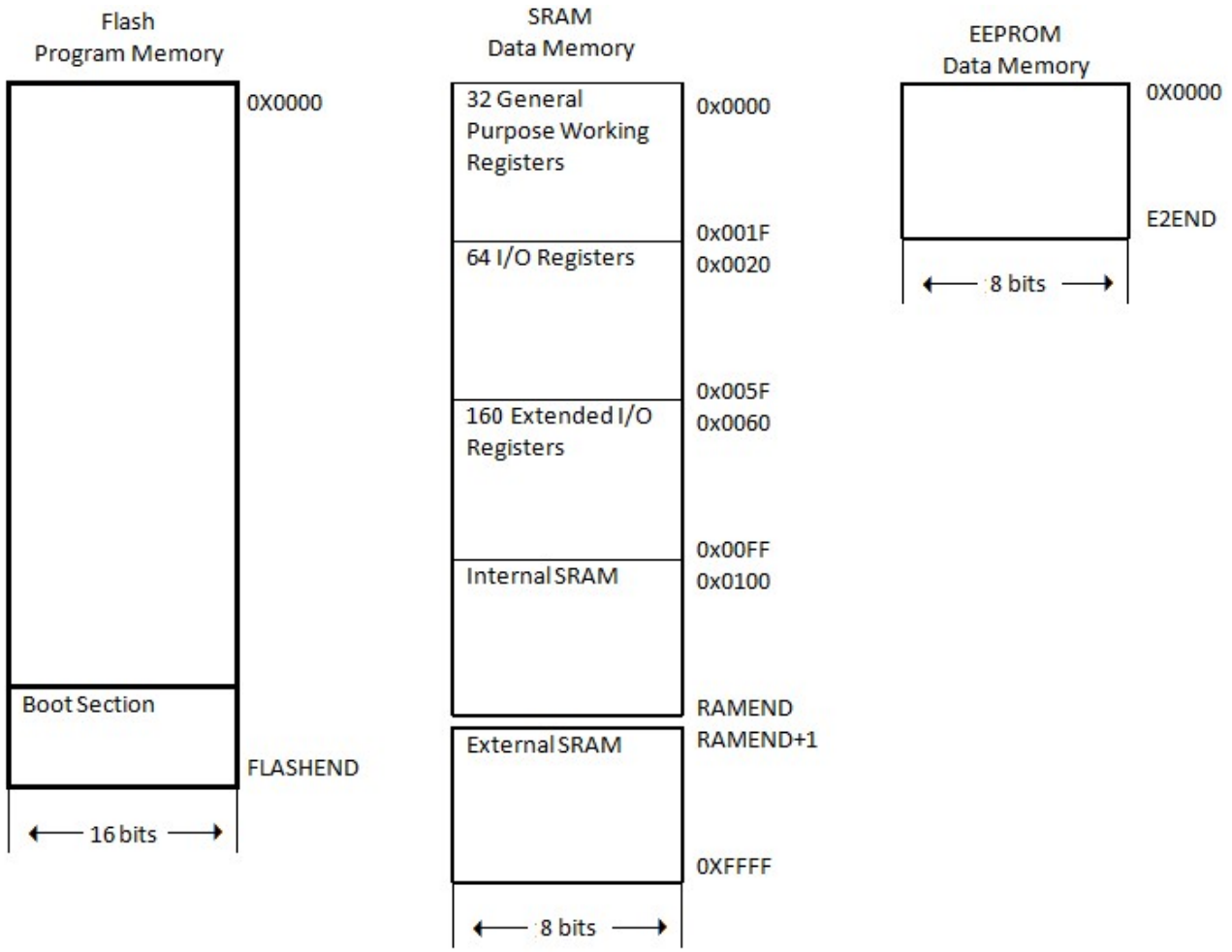
La memoria flash e la memoria EEPROM non sono volatili (le informazioni persistono dopo lo spegnimento dell'alimentazione). La SRAM è volatile e andrà persa quando si spegne e riaccende.

Il chip ATmega328 di Arduino Uno ha le seguenti quantità di memoria:

Flash	32k bytes
SRAM	2k bytes
EEPROM	1k byte

mentre ATmega2560 nel Mega2560 ha piu' memoria:

Flash	256k bytes
SRAM	8k bytes
EEPROM	4k bytE



Di seguito alcune funzioni e sketch utili per analizzare la CPU e la memoria Ram di Arduino
N.B. in grassetto l'output prodotto dal codice

Funzione: getCPU() – Restituisce in una stringa il nome della CPU (con Arduino uno Rev 3 restituisce "ATmega328")

```
String getCPU(){
  String sr="Unknown!";
  switch(SIGNATURE_0){
    case 0x1E:{
      if(SIGNATURE_1==0x094)
        if(SIGNATURE_2==0x06)
          sr="ATmega168";
      if(SIGNATURE_1==0x095)
        if(SIGNATURE_2==0x0f)
          sr="ATmega328";
      if(SIGNATURE_1==0x097)
        if(SIGNATURE_2==0x03)
          sr="ATmega1280";
      if(SIGNATURE_1==0x098)
        if(SIGNATURE_2==0x01)
          sr="ATmega328";
      }
      break;
    default:
     return sr;
```

```
  }
  return sr;
}
```

ATmega328

Funzione printEndianType()-Stampa il tipo 'endian' della cpu

```
void printEndianType()
{
int num = 1;
if(*(char *)&num == 1)
{
  Serial.println("Little-Endian");
}
else
{
   Serial.println("Big-Endian");
}
}
```

//su Arduino uno rev 3
Little-Endian

Funzione printSizeTypes(): Stampa la dimensione in byte dei tipi predefiniti di Arduino

```
void printSizeTypes(){
   Serial.println("boolean: "+ sizeof(boolean) + " byte");
   Serial.println("byte: " + sizeof(byte) + " byte");
   Serial.println("char :" + sizeof(char)+ " byte");
   Serial.println("unsigned char: " + sizeof(unsigned char)+ " byte");
   Serial.println("word: " + sizeof(word)+ " byte");
   Serial.println("unsigned int: " + sizeof(unsigned int )+ " byte");
   Serial.println("int: " + sizeof(int )+ " byte");
   Serial.println("unsigned long: " + sizeof(unsigned long )+ " byte");
   Serial.println("long: " + sizeof(long )+ " byte");
   Serial.println("float: " + sizeof(float )+ " byte");
  }
```

boolean: 1 byte
byte: 1 byte
char :1 byte
unsigned char: 1 byte
word: 2 byte
unsigned int: 2 byte
int: 2 byte
unsigned long: 4 byte
long: 4 byte
float: 4 byte

Funzione MemoryFree() – Stampa il numero di bytes liberi nella RAM

```
void setup()
{
  Serial.begin(9600);
  Serial.println(memoryFree());
```

```
}

// variables created by the build process when compiling the sketch
extern int __bss_end;
extern void *__brkval;

// function to return the amount of free RAM
int memoryFree()
{
  int freeValue;

  if((int)__brkval == 0)
     freeValue = ((int)&freeValue) - ((int)&__bss_end);
  else
    freeValue = ((int)&freeValue) - ((int)__brkval);

  return freeValue;
}
```

1846

Ma se aggiungo nel codice precedente una variabile ad esempio int che come visto occupa 2 byte:

```
 int a;
void setup()
{

  Serial.begin(9600);
  a=0;
  Serial.println(memoryFree());
}
…….
```

1844

Esattamente 2 byte in meno.
Se aggiungo una variabile float:

```
int a;
float b;
void setup()
{

  Serial.begin(9600);
  a=0;b=0;
  Serial.println(memoryFree());
}
…….
```

1840

Accesso memoria EEPROM

```
#include  <EEPROM.h>
int indirizzo=0;
byte valore;
```

```
void setup(){
Serial.begin(9600);
//Esempio Write su EEPROM
//for int(i=0;i<1024;i++)
//  EEPROM.write(i,i);
}

void loop(){
valore = EEPROM.read(indirizzo);
Serial.print("0x");
Serial.print(indirizzo,HEX);
Serial.print("\t");
Serial.print("0x");
Serial.print(valore, HEX);
Serial.println();
indirizzo +=1;
if(1024 == indirizzo)
  indirizzo=0;
delay(500);
}
```

Sketch: stampa contenuto e indirizzi di memoria (occorre conoscere i puntatori del C)

```
//dump: print memory range
void dump(char *data_buffer, const unsigned int length) {
  unsigned char byte;
  unsigned int i, j;
  char buff[5];
  sprintf(buff, "[%04x] ", data_buffer);
  Serial.print(buff);
  for (i = 0; i < length; i++) {
    byte = data_buffer[i];
    sprintf(buff, "%02X ", byte);
    Serial.print(buff);
    if (((i % 16) == 15) || (i == length - 1)) {
      for (j = 0; j < 15 - (i % 16); j++)
        Serial.print("   ");
      Serial.print("| ");
      for (j = (i - (i % 16)); j <= i; j++) {  // display printable bytes from line
        byte = data_buffer[j];
        if ((byte > 31) && (byte < 127)){ // outside printable char range
          sprintf(buff, "%c", byte);
          Serial.print(buff);
        }
        else
          Serial.print(".");
      }
      Serial.println("");// end of the dump line (each line 16 bytes)
      sprintf(buff, "[%04x] ", data_buffer+i+1);
      Serial.print(buff);
    } // end if
  } // end for
Serial.println ();
```

```
}

int a;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  a=7;
  int *pa=&a;
  Serial.println((uint16_t)&a, HEX);
  Serial.println ((int )(int *)&a,HEX);
  Serial.println ((int )pa,HEX);

  Serial.println (*pa,HEX);

  dump(( char *)&a,32); //Stampa 32 byte a partire dall'indirizzo di a
  *pa=0x4E2;
  dump(( char *)&a,32); //Stampa 32 byte a partire dall'indirizzo di a

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
12E
12E
12E
7
[012e] 07 00 00 00 00 00 03 01 00 00 00 04 01 00 00 E8 | ......0.........
[013e] 03 00 00 00 00 00 00 C5 00 C4 00 C0 00 C1 00 C2 | ................
[014e]
[012e] E2 04 00 00 00 00 03 01 00 00 00 04 01 00 00 E8 | ......0.........
[013e] 03 00 00 00 00 00 00 C5 00 C4 00 C0 00 C1 00 C2 | ................
[014e]
```