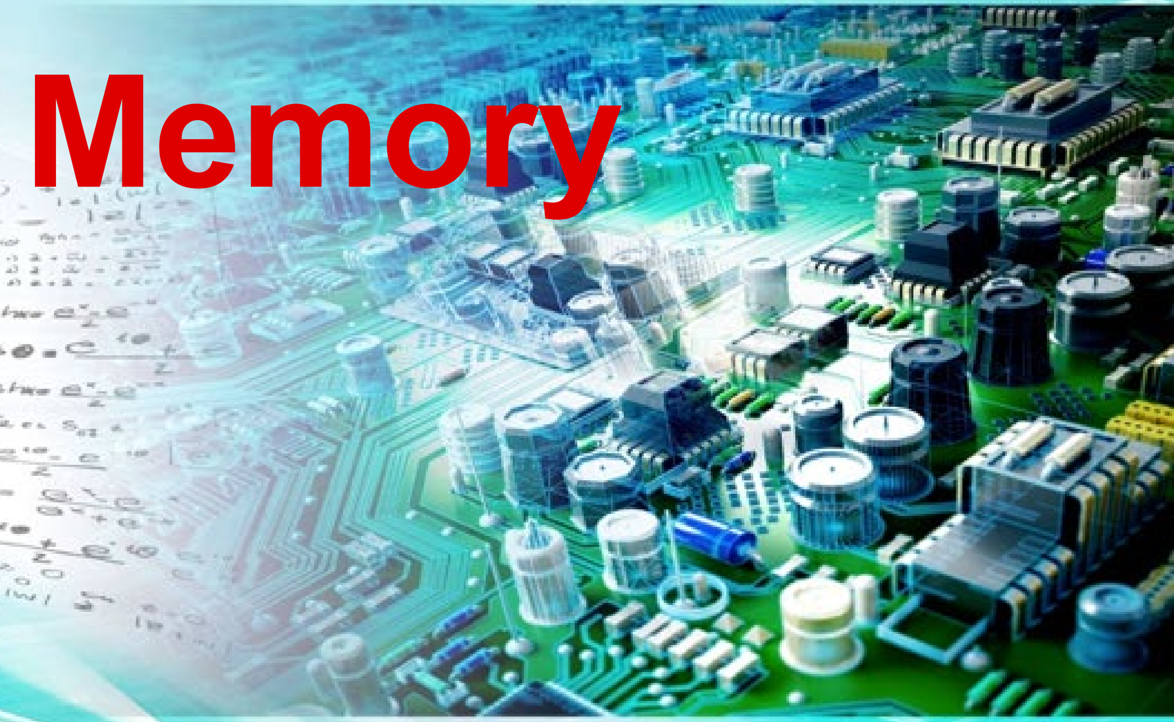
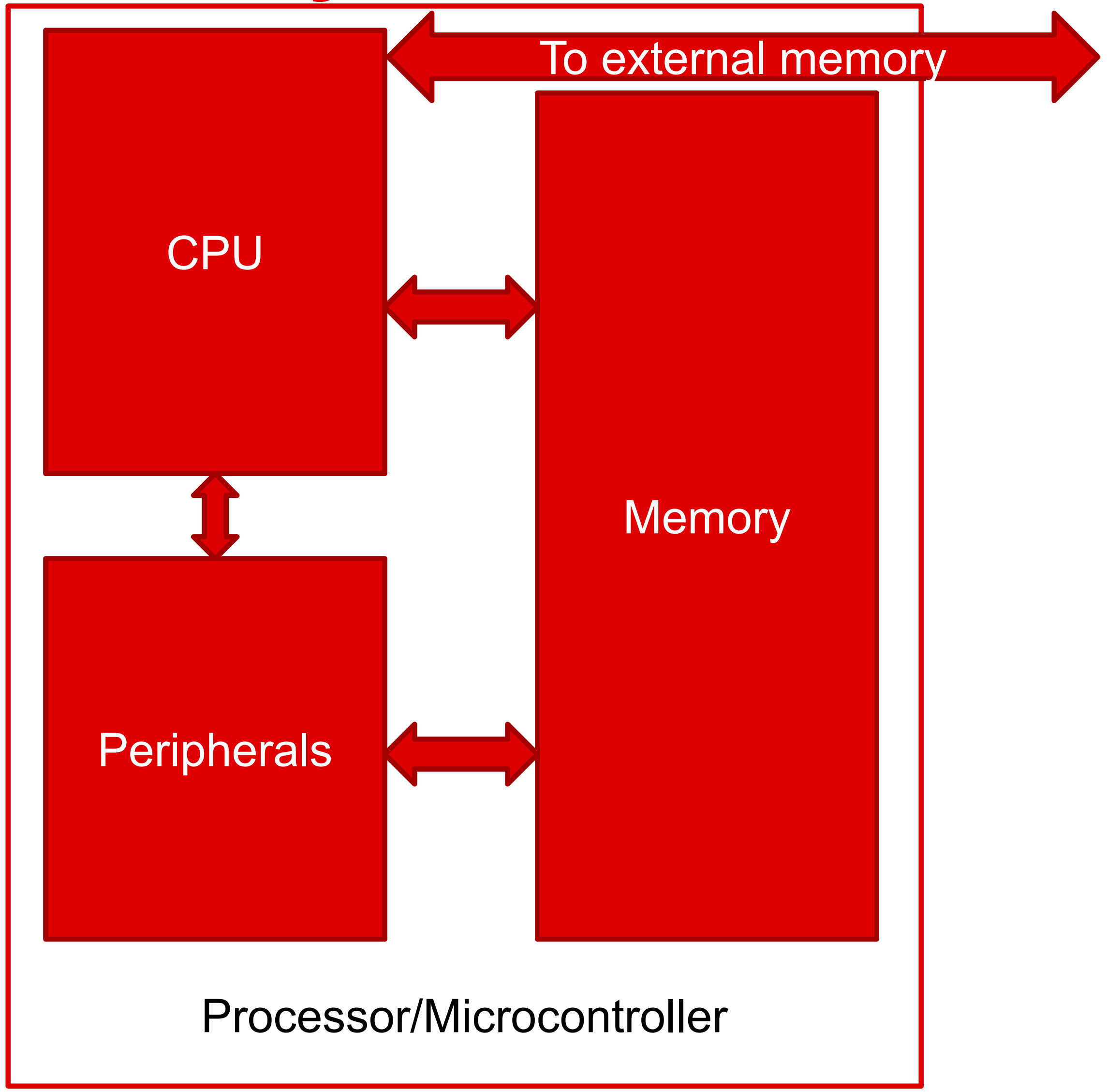
Embedded Flash Memory

TI Precision Labs – Microcontrollers

Prepared and Presented by Matthew Pate



Memory in an embedded system



- All embedded devices have some type of memory
- Role in the system:
  - Program (code) storage for the CPU
  - Data storage for both CPU and Peripherals

- Two types of memory
  - Volatile: Contents lost when device is powered down
  - Non-volatile: Contents preserved when device is powered down

- Volatile
  - SRAM
  - DRAM
  - Caches

- Non-volatile
  - $\boxed{\text{Flash}}$
  - FRAM
  - ROM

Typical tradeoffs between memory types

Topic	Volatile (SRAM)	Non-Volatile (Flash)
Retention between power cycling	No	Yes
Speed	Faster, up to CPU max operating frequency	Slower, less than CPU max operating frequency
Writes	Direct from CPU	Requires special operations to change contents
Endurance	Limitless changes to memory contents	Capped in the range of tens of thousands changes

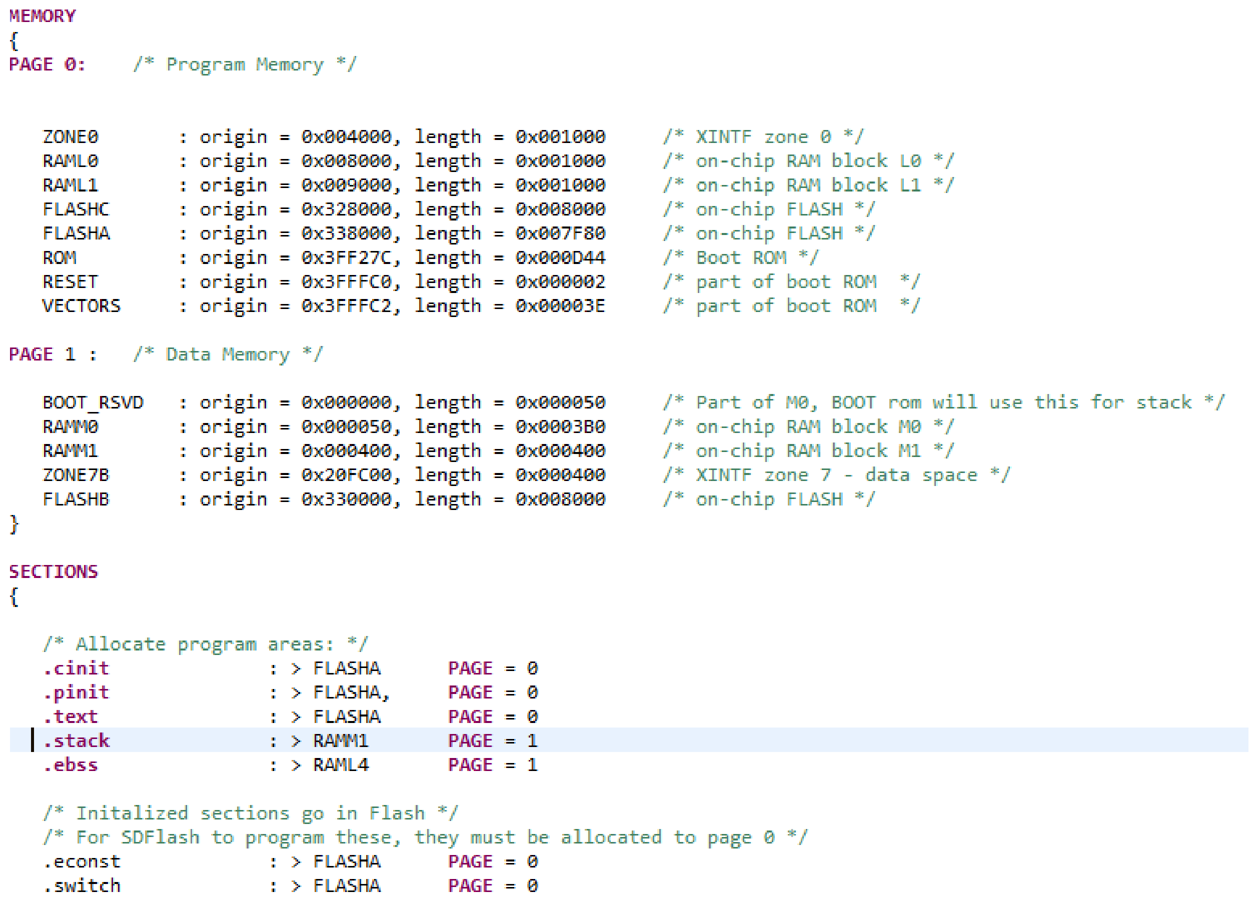
Typical memory allocations

Initialized Sections:

Name	Description	Link Location
.text	Code	FLASH
.cinit	Initialization values for global and static variables	FLASH
.econst	Constants (e.g. cnst int k=3;)	
.switch	Tables for switch statements	FLASH
.pinit	Tables for global constructors (C++)	FLASH

Uninitialized Sections:

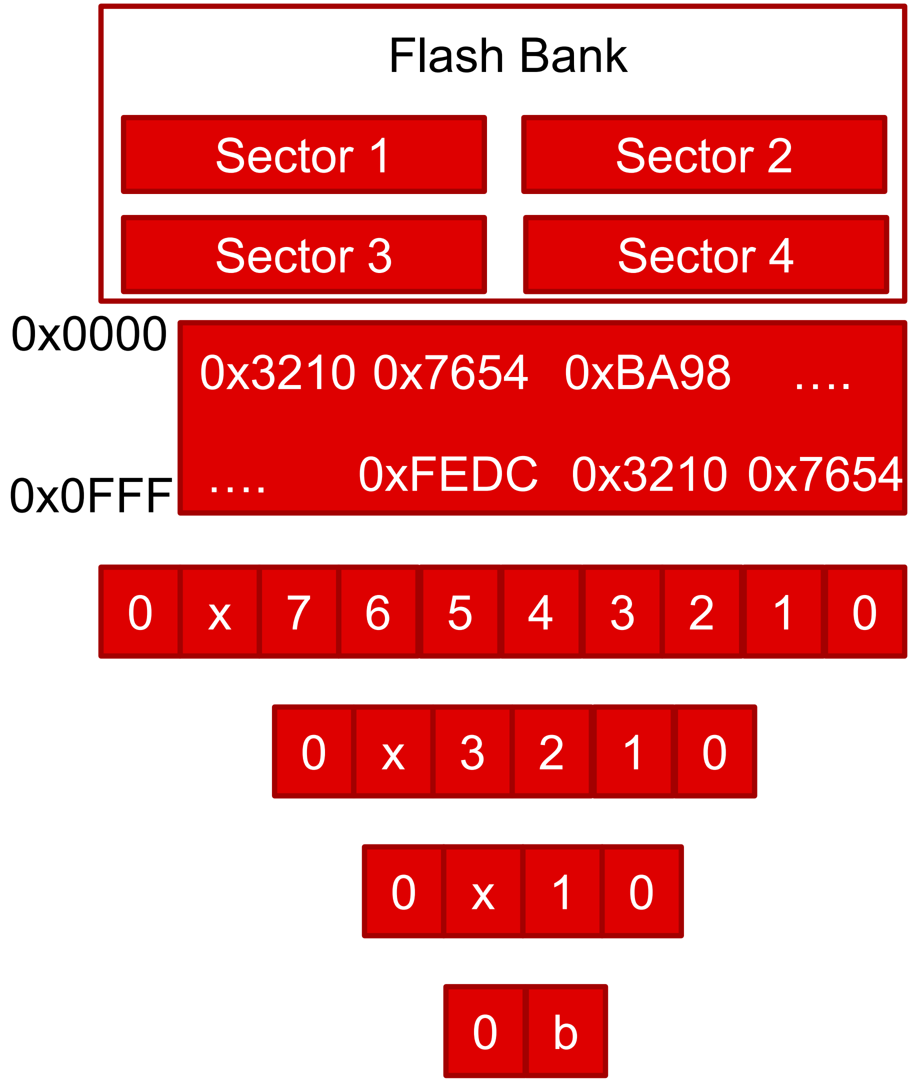
Name	Description	Link Location
.ebss	Global and static variables	RAM
.stack	Stack Space	low 64Kw RAM
.esysmem	Memory for malloc functions	RAM



```
MEMORY
{
PAGE 0:    /* Program Memory */


   ZONE0       : origin = 0x004000, length = 0x001000     /* XINTF zone 0 */
   RAML0       : origin = 0x008000, length = 0x001000     /* on-chip RAM block L0 */
   RAML1       : origin = 0x009000, length = 0x001000     /* on-chip RAM block L1 */
   FLASHC      : origin = 0x328000, length = 0x008000     /* on-chip FLASH */
   FLASHA      : origin = 0x338000, length = 0x007F80     /* on-chip FLASH */
   ROM         : origin = 0x3FF27C, length = 0x000D44     /* Boot ROM */
   RESET       : origin = 0x3FFFC0, length = 0x000002     /* part of boot ROM  */
   VECTORS     : origin = 0x3FFFC2, length = 0x00003E     /* part of boot ROM  */

PAGE 1 :   /* Data Memory */

   BOOT_RSVD   : origin = 0x000000, length = 0x000050     /* Part of M0, BOOT rom will use this for stack */
   RAMM0       : origin = 0x000050, length = 0x0003B0     /* on-chip RAM block M0 */
   RAMM1       : origin = 0x000400, length = 0x000400     /* on-chip RAM block M1 */
   ZONE7B      : origin = 0x20FC00, length = 0x000400     /* XINTF zone 7 - data space */
   FLASHB      : origin = 0x330000, length = 0x008000     /* on-chip FLASH */
}

SECTIONS
{

   /* Allocate program areas: */
   .cinit              : > FLASHA      PAGE = 0
   .pinit              : > FLASHA,     PAGE = 0
   .text               : > FLASHA      PAGE = 0
   .stack              : > RAMM1       PAGE = 1
   .ebss               : > RAML4       PAGE = 1

   /* Initalized sections go in Flash */
   /* For SDFlash to program these, they must be allocated to page 0 */
   .econst             : > FLASHA      PAGE = 0
   .switch             : > FLASHA      PAGE = 0
```

Memory terminology

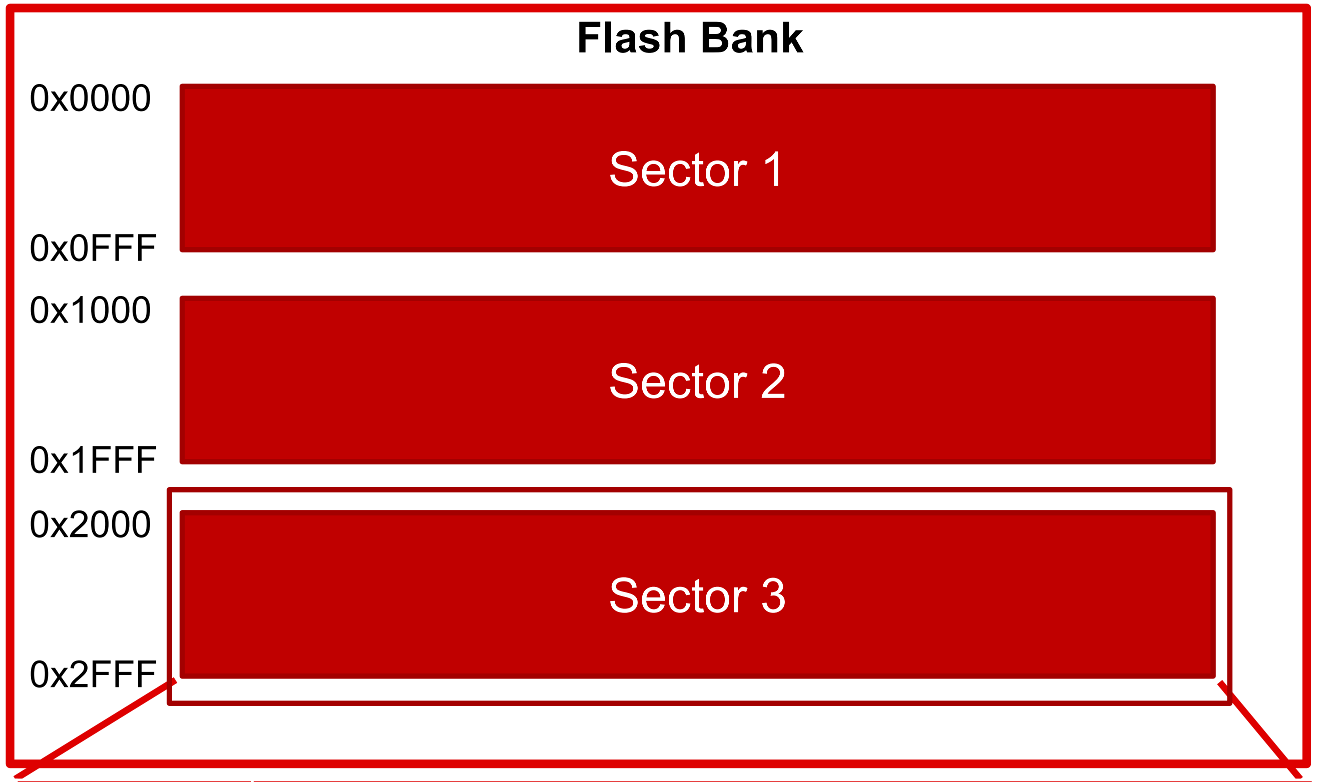
Term	Definition	Example
Bank	Flash term of grouping of multiple sectors	1 flash bank could have 4 sectors
Sector	Flash term for grouping of multiple words	1 flash sector could be 1KW long 1K = 1024
Long	Grouping of multiple words(32, 64, etc)	
Word	Grouping of multiple bits(16, 32, 64, etc)	10 kilowords = 10KW
Byte	Grouping of 8 bits	10 kilobytes = 10KB
Bit	Binary Value 0(b) or 1(b)	10 kilobits = 10Kb



Texas Instruments

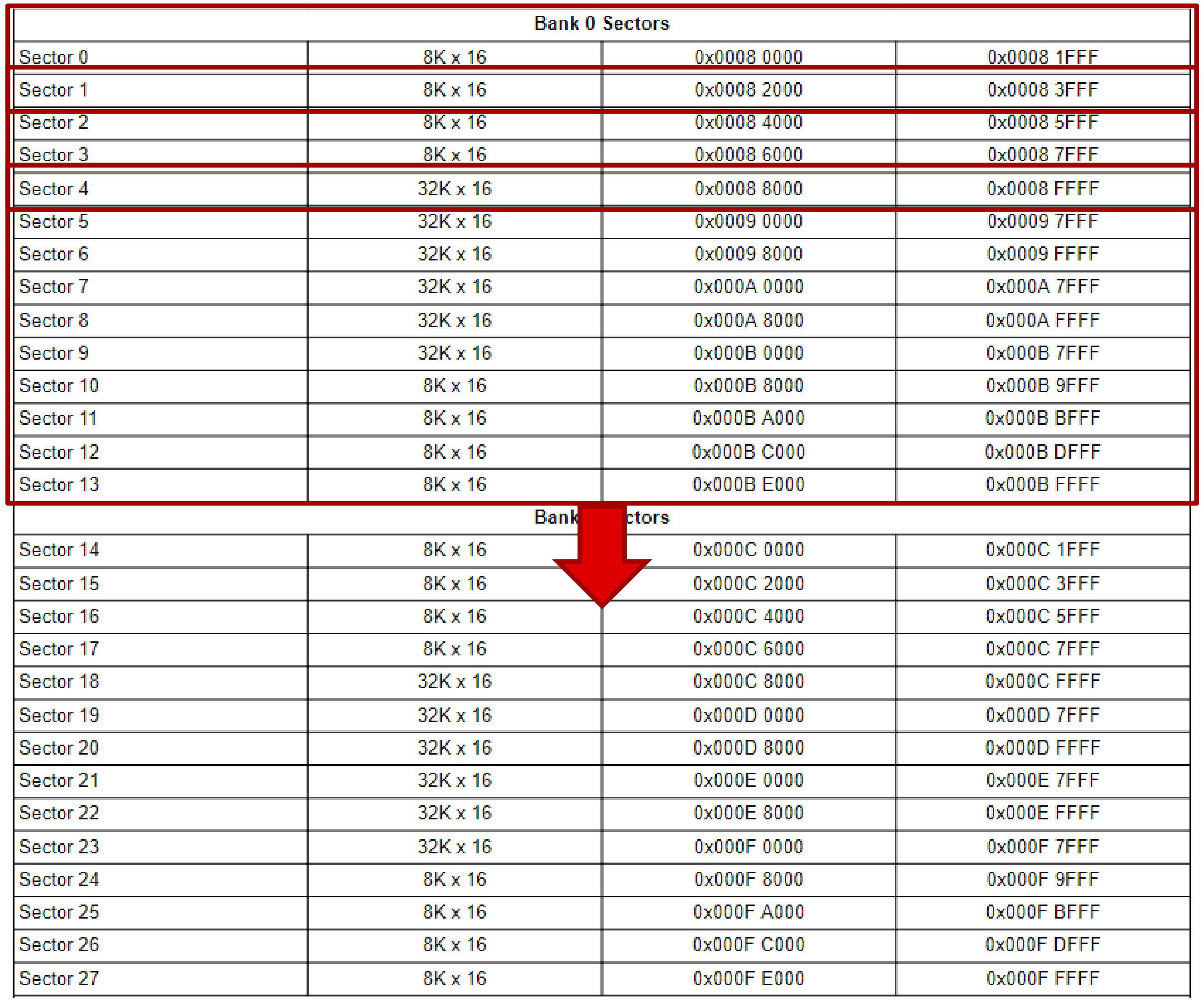
Flash write and erase

- To change the contents of a flash memory specific operations are required
- Erase/Erased State
  - Value of 1b is considered erased
  - Changing a value from a 0b to a 1b is called erasing the memory
  - Can only occur at the sector level
- Write/Programmed State
  - Value of 0b is considered programmed
  - Changing a value from a 1b to a 0b is called writing or programming the memory
  - Can occur at the bit level
  - Special case when all words in a sector are 0, considered “cleared”
- Example
  - Data is combination of 1’s(b) and 0’s(b)
  - Wish to program address 0x2000 to value of 0x8888
  - Must erase the entire sector
    - Call the Flash Erase API
  - Then program 0x8888 to address 0x2000
    - Call the Flash Program API
    - Also need to restore the pre-existing data



Address	Data			
0x2000	0xFFFF	0xFFFF	0xFFFF	0xFFFF
0x2004	0xFFFF	0xFFFF	0xFFFF	0xFFFF

Example memory map



Bank 0 Sectors			
Sector 0	8K x 16	0x0008 0000	0x0008 1FFF
Sector 1	8K x 16	0x0008 2000	0x0008 3FFF
Sector 2	8K x 16	0x0008 4000	0x0008 5FFF
Sector 3	8K x 16	0x0008 6000	0x0008 7FFF
Sector 4	32K x 16	0x0008 8000	0x0008 FFFF
Sector 5	32K x 16	0x0009 0000	0x0009 7FFF
Sector 6	32K x 16	0x0009 8000	0x0009 FFFF
Sector 7	32K x 16	0x000A 0000	0x000A 7FFF
Sector 8	32K x 16	0x000A 8000	0x000A FFFF
Sector 9	32K x 16	0x000B 0000	0x000B 7FFF
Sector 10	8K x 16	0x000B 8000	0x000B 9FFF
Sector 11	8K x 16	0x000B A000	0x000B BFFF
Sector 12	8K x 16	0x000B C000	0x000B DFFF
Sector 13	8K x 16	0x000B E000	0x000B FFFF
Bank ctors			
Sector 14	8K x 16	0x000C 0000	0x000C 1FFF
Sector 15	8K x 16	0x000C 2000	0x000C 3FFF
Sector 16	8K x 16	0x000C 4000	0x000C 5FFF
Sector 17	8K x 16	0x000C 6000	0x000C 7FFF
Sector 18	32K x 16	0x000C 8000	0x000C FFFF
Sector 19	32K x 16	0x000D 0000	0x000D 7FFF
Sector 20	32K x 16	0x000D 8000	0x000D FFFF
Sector 21	32K x 16	0x000E 0000	0x000E 7FFF
Sector 22	32K x 16	0x000E 8000	0x000E FFFF
Sector 23	32K x 16	0x000F 0000	0x000F 7FFF
Sector 24	8K x 16	0x000F 8000	0x000F 9FFF
Sector 25	8K x 16	0x000F A000	0x000F BFFF
Sector 26	8K x 16	0x000F C000	0x000F DFFF
Sector 27	8K x 16	0x000F E000	0x000F FFFF

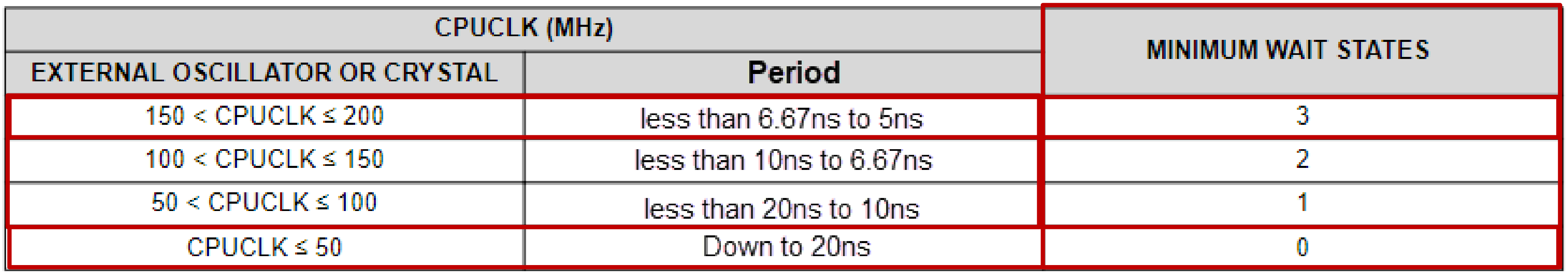
Table from SPRS880

- Code execution and flash programming cannot occur in the same bank
  - Shift code to execute from RAM
  - If device has multiple banks, can run from another bank while programming
- Sector sizes can vary device to device or on the same device
  - Check your device datasheet for implementation
  - No matter the size, erase is still required at the sector level for TI flash
- Example is from 16-bit word device, sizes shown in KW

Flash electrical specifications - reads

- Flash typically cannot operate at the full CPU frequency of a device
- Must introduce intentional delay to accommodate different CPU speeds
- Period = 1/Frequency
  - 20ns = 1/50MHz
- Each access/read to a flash word takes 20ns
- If the CPU runs faster than 50MHz still have to give 20ns for a read
- Add Wait States
- Access time = CPU period *(1+WaitStates) => WaitStates = (Access Time/CPU Period)-1
- 200MHz = 5ns period
- WS = (20ns/5ns) -1= 3
- WS are configurable per device see datasheet for exact information

Table 8-4. Flash Wait States

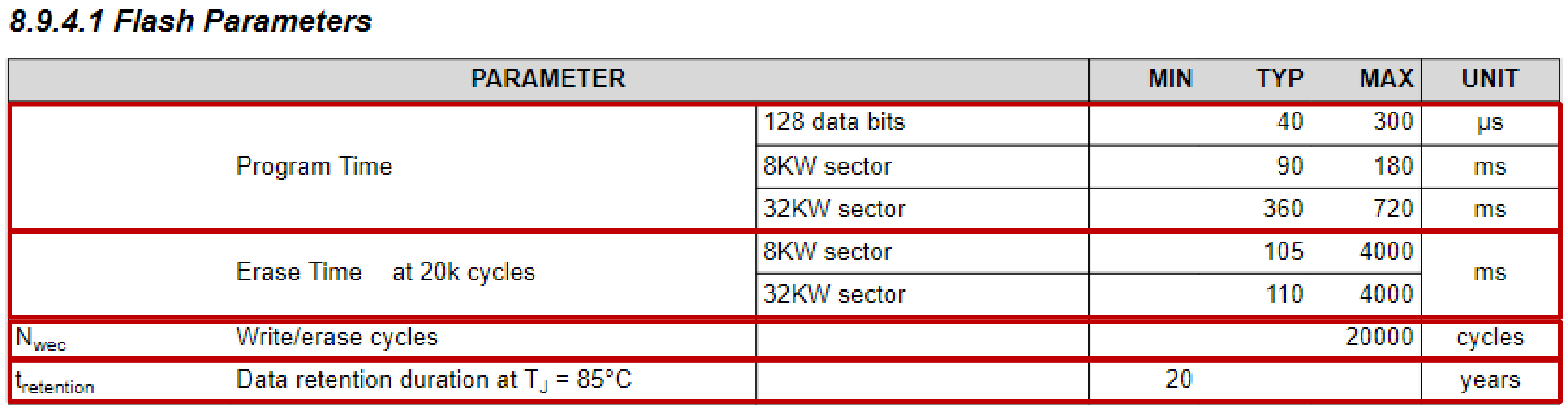


CPUCLK (MHz)		MINIMUM WAIT STATES
EXTERNAL OSCILLATOR OR CRYSTAL	Period	
150 < CPUCLK ≤ 200	less than 6.67ns to 5ns	3
100 < CPUCLK ≤ 150	less than 10ns to 6.67ns	2
50 < CPUCLK ≤ 100	less than 20ns to 10ns	1
CPUCLK ≤ 50	Down to 20ns	0

Table from SPRS880

Flash electrical specifications - writes

- Special Operations Required to write to flash memory
  - Write and Erase APIs
- Due to the physical construction of the flash memory these operations take defined times listed in the datasheet
- Program Time
  - Time to take a specific number of bits or words from erased(all 1’s) to some combination of 0’s and 1’s
- Erase Time
  - Time required to erase an entire sector making all bits 1’s
- Write/erase cycles
  - Number of times the flash can be written/erased over the lifetime of the device
- Data retention
  - How long the data is guaranteed to be correct



8.9.4.1 Flash Parameters

PARAMETER			MIN	TYP	MAX	UNIT
	Program Time	128 data bits		40	300	µs
		8KW sector		90	180	ms
		32KW sector		360	720	ms
	Erase Time at 20k cycles	8KW sector		105	4000	ms
		32KW sector		110	4000	
N_{wec}	Write/erase cycles				20000	cycles
$t_{retention}$	Data retention duration at T_J = 85°C		20			years

Table from SPRS880

Flash programming scenarios

Product Lifecycle	Programming Method
Development	JTAG connection & debugger IDE
Production	Inline or pre-assembly programming Single device PCB/Socket Multi device programmers Automated handler for large scale production
Field Updates	Embedded flash boot kernels for code Flash API calls for data updates

To find more Microcontroller and Processor technical resources and search products, visit **https://www.ti.com/microcontrollers-mcus-processors/overview.html**.