**ENGR 260 (Microcontroller Systems)**

**Homework 1**

**Problem 1:**  Microprocessor data and address manipulation

A microprocessor has a 32-bit address bus and a 16-bit data bus, also its data registers are 16-bit registers.

1. What is the size of the address space this microprocessor can address?

**This MPU can address 2^32 unique memory locations: 4294967296**

**If those memory locations hold 8 bits each which is typical, then that is 4 GiB**

**If those memory locations hold 16 bits each (which would be very unusual) then that is 8GiB**

1. Would you consider this to be a 16-bit or a 32-bit microprocessor? Please justify your answer.

**A processor is typically categorized by the size of it’s registers, although the memory address bus size is sometimes used. Look at, for example, the AVR architecture: the processors have 8 bit data registers, but a 16-bit address space. Atmel considers these to be 8 bit processors. I would consider your theoretical processor to be a 16 bit processor. Marketing likely will want to call it a 32 bit processor.**

**Problem 2:** The TM4C123GH6PM

The memory map for the TM4C123GH6PM is shown on pages 92-94 of the datasheet (Tiva\_C\_Datasheet.pdf).

1. The table has 20 memory ranges that are marked as *reserved*. We know that this microcontroller has 4 G**i**B (232) of addressable memory space, of this 4 G**i**B address space how much space is *reserved*?

**(size = final addr - initial addr + 1)**

**(The +1 is because the start address is inclusive. IE 0x0 – 0xF is 16 addresses, not 15)**

**0x0004.0000 - 0x1FFF.FFFF | 0x1FFC.0000 bytes (536608768 bytes/addresses) (524,032KiB)**

**0x2000.8000 – 0x21FF.FFFF | 0x01FF.8000 bytes (33521664 bytes/addresses) (32,736KiB)**

**0x2210.0000 – 0x3FFF.FFFF | 0x1DF0.0000 bytes (502267904 bytes/addresses) (490,496KiB)**

**0x4000.2000 – 0x4000.3FFF | 0x0000.2000 bytes (8192 bytes/addresses) (8KiB)**

**0x4001.4000 – 0x4001.FFFF | 0x0000.C000 bytes (49152 bytes/addresses) (48KiB)**

**0x4002.6000 – 0x4002.7FFF | 0x0000.2000 bytes (8192 bytes/addresses) (8KiB)**

**0x4002.A000 – 0x4002.BFFF | 0x0000.2000 bytes (8192 bytes/addresses) (8KiB)**

**0x4002.E000 – 0x4002.FFFF | 0x0000.2000 bytes (8192 bytes/addresses) (8KiB)**

**0x4003.A000 – 0x4003.BFFF | 0x0000.2000 bytes (8192 bytes/addresses) (8KiB)**

**0x4003.D000 – 0x4003.FFFF | 0x0000.3000 bytes (12288 bytes/addresses) (12KiB)**

**0x4004.2000 – 0x4004.BFFF | 0x0000.A000 bytes (40960 bytes/addresses) (40KiB)**

**0x4005.1000 – 0x4005.7FFF | 0x0000.7000 bytes (28672 bytes/addresses) (28KiB)**

**0x4005.E000 – 0x400A.EFFF | 0x0005.1000 bytes (331775 bytes/addresses) (324KiB)**

**0x400B.0000 – 0x400F.8FFF | 0x0004.9000 bytes (299008 bytes/addresses) (292KiB)**

**0x400F.A000 – 0x400F.BFFF | 0x0000.2000 bytes (8192 bytes/addresses) (8KiB)**

**0x4010.0000 – 0x41FF.FFFF | 0x01F0.0000 bytes (32505856 bytes/addresses) (31744KiB)**

**0x4400.0000 – 0xDFFF.FFFF | 0x9C00.0000 bytes (2617245696 bytes/addresses) (2555904KiB)**

**0xE000.3000 – 0xE000.DFFF | 0x0000.B000 bytes (45056 bytes/addresses) (44KiB)**

**0xE000.F000 – 0xE003.FFFF | 0x0003.1000 bytes (200704 bytes/addresses) (196KiB)**

**0xE004.2000 – 0xFFFF.FFFF | 0x1FFB.E000 bytes (536600576 bytes/addresses) (524024KiB)**

**4GiB = 4194304 KiB**

**Minus all the KiB figures above.**

**= 34,336 KiB is NOT reserved (~33.5 MiB)**

**and**

**4,159,968KiB is RESERVED**

**(This was VERY tedious)**

1. Provide two possible reasons for reserving these memory ranges.

**I brought up in class, and was told “no”, but will still insist:**

**The reserved memory areas are PRIMARILY unused address space (for example, the last reserved block, 0xE004.2000 – 0xFFFF.FFFF is clearly just spanning all the unused address space until the highest address. There is no way all 4GiB is used by actual, real devices.**

**After that, many of the reserved areas belong to peripherals on other microcontrollers in the same product line – if our controller had 5 timers but another controller has 6, then our controller will have a reserved block where this timer would have been. Look at this**

**MCU is the same line:** **https://www.ti.com/lit/ds/symlink/tm4c129xnczad.pdf**

**At 0x6000.0000 – 0xDFFF.FFFF there is the “EPI0 mapped peripheral and RAM” section, entirely inside one of the reserved regions on the MCU we use in this class. By “reserving” these areas, it is ensured that code can be transferred between controllers in the same line – if it was allowed to write into reserved areas then when moving to a different controller where something IS there, we would inadvertently access hardware we shouldn’t.**

**Then, finally, some of the address space are used by devices on the MCU that we don’t have direct use of – namely the ROM with the tiva-c firmware, which sits in one of the reserved section on our MCU.**

1. There is an internal 2Kbytes EEPROM. Please provide this memory’s start and end addresses as provided on the memory map.

**0x400A.F000 – 0x400A.FFFF | EEPROM control registers**

**NOTE: THE DATA IN THE EEPROM IS NOT DIRECTLY MAPPED ON THE MEMORY BUS! THE EEPROM IS ACCESSED WITH AN INTERFACE EXPOSED ON IT’S CONTROL REGISTERS. SEE 8.2.4.1 IN THE DATASHEET**

**The “START ADDRESS” of the EEPROM is 0x0, because accessing the EEPROM occurs by setting the EEBLOCK and EEOFFSET registers, then reading the EERDWR register to see the result.**

1. How many bits does this memory hold?

**2KiB \* 1024 \* 8 = 16384 bits**

1. How many 32-bit words does this memory have?

**512**

1. Given that 16 words make a block, determine the number of blocks the memory has.

**32**

1. What is the address of bit 31 of word0 i.e. the first word of this memory?

**Bits do not have addresses – the smallest addressable unit on ordinarily mapped memory is a byte. This is why we have to use bit-masks to access individual bits. THE EEPROM SPECIFICALLY CAN ONLY BE ACCESSED BY WORD-ALIGNED BLOCK/OFFSET ADDRESSES. Word 0 is accessed at block 0x####.0000, offset 0x####.#000 where those are the values to use in the EEBLOCK and EEOFFSET registers respectively. # indicates bits that are reserved and must have their values left unmodified.**

**Once the value has been loaded from the EERDWR register, we can access the 31st bit by masking. IE**

**\*EEBLOCK = ((\*EEBLOCK)&0xFFFF0000) | 0x00000000;**

**\*EEOFFSET = ((\*EEOFFSET)&0xFFFFFFF8) | 0x00000000;**

**read\_value = (\*EERDWR)>>31; //Get the 31st bit of the first word of the EEPROM**

**Problem 3:** Memories

1. Is ROM volatile or non-volatile?

**ROM is non-volatile. This means that it will keep it’s value when power is removed.**

1. List the devices that can drive the address bus during a CPU write cycle.

**During a CPU write cycle? At the exact cycle of the write, only the CPU drives the address bus. During the other 3/4(?) clock cycles it takes to execute a write, the program counter will drive the address bus – fetching the instruction. During the decode & execute cycles other devices may use the bus – notably the DMA controller, Memory protection unit, and MMU (if those are present)**

**Problem 4:** Major components of a microcontroller

Please explain the function of the following microcontroller components: ROM, RAM, EPROM, EEPROM, I/O Data Registers and Control and Status registers

* **ROM: Read only memory. Used for code and data that must be non-volatile. Usually mapped directly on the address bus, allowing direct access to the data or code stored on it. As the name suggests, we can only ever read this memory.**
* **RAM: Random access memory. Volatile – state is not guaranteed after loss of power. Fast working memory that can sustain a virtually unlimited number of read and write cycles. This is where we store data (variables, etc) that we are working on. The CPU may also execute code from the RAM by marking an area as executable; you would do this because the FLASH memory has wait-states when the system clock is >40mhz (on the tiva-c) and will cause your programs to run slower than expected, especially when branching.**
* **EPROM: “Electronically programmable read only memory”. EPROM can only ever be written once, and never erased. There are several registers in the Tiva-C that are EPROM – mainly around permanently disabling certain functions of the device, like debugging.**
* **EEPROM: “Electronically Erasable Programmable Read Only Memory” EEPROM and FLASH are different, but the line between the two is blurred. EEPROM can be written and erased a limited number of cycles, and must be erased in blocks or pages depending on the system. The EEPROM on the Tiva-C is accessed more like a foreign peripheral than memory-mapped ROM. The FLASH on the Tiva-C is much larger and is memory-mapped. Typical code execution on the Tiva-C occurs using code read from the FLASH memory. Writing the program to a microcontroller is called “Flashing” for this reason.**
* **I/O Data Registers: Memory mapped IO registers control peripheral devices, like USB and GPIO. The data registers specifically usually contain either the data to be read in, or the data to be sent out. Some data-out registers are write only, while some data-in registers are read-only. Some peripheral devices have a unified “data” register that is either read-only or write-only depending on the state of the device.**
* **Control Registers: Control registers place the device in different states. There are several control registers in the CORTEX-M4 CPU in the system, and also many memory mapped control registers that control the MCU, setting things like system clock sources, sleep states, turning IO on/off, etc. The CPU control register sets things like whether or not the MMU should be enabled, what endianness to use, and very importantly: the address of the NVIC table! More information on the CORTEX-M4 control register here: https://developer.arm.com/documentation/den0013/d/ARM-Processor-Modes-and-Registers/Registers/System-control-register--SCTLR-**
* **Status Registers:The CORTEX-M4 cpu’s status register stores information about the results of operations – did overflow occur, was the last comparison equal? Greater? Less? There are also device registers called status registers – particularly to do with the NVIC.**

**Problem 5:** On-Chip Flash Memory

From the memory map of page 92, on-chip Flash Memory occupies memory locations 0x0000.0000 to 0x0003.FFFF. What is the size of this on-chip Flash memory? Show your work based on the provided address range.

**0x0003.FFFF – 0x0000.0000 = 0x0003.FFFF | Add one, since 0 is also an address**

**Size = 0x0004.0000 → divide by 0x400 to get KiB: → 0x100 KiB → in base 10, 1\*16^2 = 256 KiB**