Proceedings,
The 2013 ASEE North Midwest Section Conference,
October 17-18, 2013, Fargo, North Dakota.

ASEE-NMWSC2013-0052

The First Lecture on Microcontrollers for Students with Limited Background in Electrical and Computer Engineering Foundations

Majura F. Selekwa
Department of Mechanical Engineering; North Dakota State University
majura.selekwa@ndsu.edu

Abstract

In the modern world, microcontrollers are found in almost every device with mechanical, chemical, and electrical applications. In general these are single chip computers integrated with various input-output interface ports. There are thousands of microcontrollers in the market, and each has its own way of manipulating its input-output interfaces. Teaching the topic of microcontrollers has never been uniform across the board since various instructors tailor their lectures on the particular type of microcontrollers used in their classes. Although many students with a good background in electrical, electronic and computer engineering can easily assimilate the material and extend the knowledge to other types of microcontrollers, it tends to be very difficult for students of non-electrical backgrounds such as mechanical and chemical engineering to respond in a similar fashion. This paper discusses one of the effective ways of approaching the the topic teaching students of non-electrical backgrounds using different types of microcontrollers in the same environment. Typically, the topic is simplified by combining the common characteristics of all microcontrollers and presenting them generically in a way that enables students to translate those characteristics to various types of microcontrollers when needed to.

1 Introduction

The title of this article is deliberately exaggerated to stress on what needs to be done in the early stages of the topic before students are exposed to any specific microcontroller classes. This information is not covered in a single first lecture only, but rather in the first two or three lectures on the subject.

Today, there are many systems that are controlled using microcontrollers; such systems, better known as embedded systems, constitute the largest share of modern consumer and industrial systems. As such knowledge of using microcontrollers in design and development of mechanical systems has become an increasingly important subject. Generally, the subject of microcontrollers is taught as Embedded Systems for electrical and computer engineering students, and mainly as part of Mechatronics for mechanical and chemical engineering students. Students in electrical

engineering, electronic engineering, computer engineering and computer science tend to have a greater leverage in understanding microcontrollers. There are many other subjects in their fields of study that help them better understand microcontrollers. On the opposite side, however, students in mechanical engineering and chemical engineering who also need to know how to use microcontrollers in their fields tend to remain disadvantaged by the way the subject is taught. While the Embedded Systems subject is dedicated to microcontrollers only, see fo example[1], on the other hand mechatronics covers microcontrollers as just one topic among many other topics, see for example[2–5]. Perhaps, because of the number of topics involved, the subject of mechatronics does not normally cover microcontrollers in as detailed manner as the embedded systems subject does. As such, many existing mechatronics textbooks also tend to cover this topic by using a single specific microcontroller, following the tradition set by early authors on the subject, for example[6]. Although some authors, see for example[2], start the topic by discussing the general microcontroller structure, most of others just jump to the particular microcontroller used by the book. And even those that discuss the general microcontroller structure, they do so very sparingly without planting the necessary seeds to make the topic understandable to students. This in effect is reflected in how the class ends up being taught in many schools, see for example[7–12]. As such students are forced to learn on type of microcontroller as if it is the only one in the market. These student tend to get stuck when required to decide for themselves which microcontroller to use in a particular design, or when required to work with a different type of a microcontroller. Their knowledge tend to remain limited to the type of microcontroller taught in class. A recent survey paper has indicated that[13] while mechanical engineering students have been increasingly becoming more interested in microcontrollers, they also have been left blind on the hardware structure common to all microcontrollers, as such students have been always looking at the final end use of particular microcontrollers only.

The objective of this paper is to provide some guidelines for instructors who teach the subject of microcontrollers to students who do not have the electrical, electronics or computer engineering background; particularly those in mechanical and chemical engineering. Although these students learn the basic principles of electrical circuit analysis in their fields of study, in general such knowledge does not prepare them well to absorb the subject of microcontrollers as presented in many textbooks without a hassle. The paper discusses how the topic should be approached step by step, starting with preparatory stages, taking students up the learning ladder from the structure of the personal computer, which is well known to students up to the general structure of microcontrollers. It is only after students have the "big picture" about microcontrollers when they can start to learn about specific microcontrollers.

2 The Microcontroller Lesson Plan

2.1 Prerequisites and Preparatory Information

All students of microcontroller systems must have some prerequisite knowledge on basic digital electronics and number systems used in digital computing. Typically students with limited background in electrical and electronics must be taught these topics prior to microcontrollers; a separate paper[14], discusses the experience of the author in preparing mechanical engineering students for microcontrollers. To prepare students who have a limited background in electronics,

first, they must be introduced to the concept of binary numbers through simple ON and OFF operations; this introduction can then be extended to basic combinational logic gates, i.e., the AND, OR, NOT, NAND and NOR gates; this preparation can run for about thirty minutes. After this basic digital electronics introduction, students can be introduced to sequential logic systems starting with the basic S-R flip flop and its evolution into the the T and D flip flops; it is from this introduction when the concept of digital clocks, memory cells and registers will discussed along with their associated key terminologies such as bytes and word. At that point, the binary number system can be discussed further extending to hexadecimal numbers system. Details of arithmetical manipulations of binary and hexadecimal numbers are not necessary, however the knowledge to use standard calculators such as the Microsoft Windows calculator in performing such operations is expected. This minimal preparatory work, which can run for up to two classes, is sufficient to put students at the right footing to understand microcontrollers.

2.2 Dissecting the Microcontroller for Students to Understand

With students who are equipped with the necessary prerequisites as discussed in the previous subsection, the first lecture on microcontrollers should start by dissecting the structure of the microcontroller into three parts: the *microprocessor*, the *system memory,* the *Input/Output Interfaces*. The system memory should be presented to show that there is a limit on the size of the information that can be handled by the microcontroller. The microprocessor part should show how data is handled within the heart of the microcontroller, while the discussion on the Input/Output Interfaces should show how the microprocessor interacts with the external world to complete the microcontroller structure. In general, this discussion can be carried out by looking at what students know best, i.e., the personal computer. The instructor needs to introduce the topic by using the structure of a normal personal computer and break it into its key six sections as follows[15]:

1. **The Input Unit** as the 'receiving' section of the computer, which receives *data* from various *input devices* such as the mouse, keyboard, scanner, and input ports for audio, video, etc., and places that data at the disposal of other parts of the computer.

2. **The Arithmetic and Logic Unit (ALU)** as the 'manufacturing' section of the computer, which processes the input data by performing arithmetic calculations $(+, -, \times, \div)$ between *two numbers* and logical operations of AND, OR and NOT as necessary.

3. **The Registry Unit** as the 'scratch pad' section of the computer where immediate data used by the ALU is stored. Students need to know that this unit contains memory cells or registers that keep a record of the ALU activities, and temporarily store data that flow between the ALU and other parts of the computer. The registry may be viewed as the "personal secretary" of the ALU, which controls how data is handled by both the ALU and other parts of the computer.

4. **The Memory Unit** as the 'warehouse' section of the computer. It is important to clarify the difference between the system memory unit and the registry unit although both are memory spaces. Unlike registers, which are accessed only by the ALU, this memory is receives information from all other parts.

5. **The Control Unit** as the 'administrative' section of the computer, which instructs all other parts on how to handle data; for example when data should be loaded into the memory unit from the input unit, when the ALU should pick up that data from the memory unit. It is worth mentioning that the control unit is basically a clock that manages the timing and the sequence of the computer operations. Its frequency determines the speed of the computer.

6. **The Output Unit** as the 'shipping' section of the computer, which takes information that has been processed by the ALU and places it on various *output devices* to make it available for use outside the computer; output devices include the computer screen, printer, our sound card.

The organization of the personal computer sections is illustrated in Figure 1. From here, students should be taught that the trio of the Control Unit, ALU and the registers together form what is known as a Microprocessor or a Central Processing unit (CPU). Each set of this trio is also known as *core,* and one microprocessor (μP) can have one or more cores, e.g., the Duo Core processor has two cores. It is common to put the input units and output units into one group known as **input/output interfaces** so that the three main parts of a computer become:

1. The Input/Output Interfaces,

2. The Memory,

3. The Microprocessor.

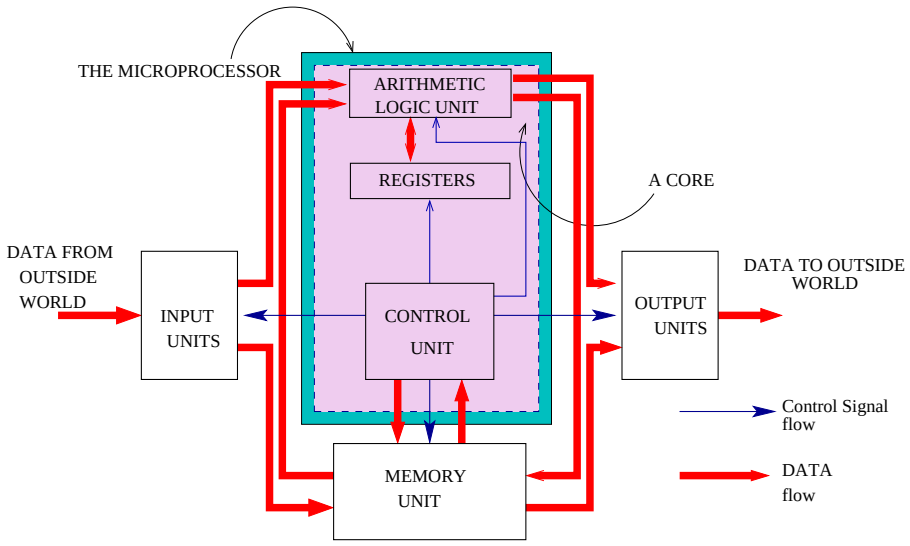


Figure 1: The Computer Organization

2.2.1 The Microprocessor

As stated earlier, many textbooks that discuss microcontrollers tend to either skip discussing microprocessors or simply do it very lightly, probably either by assuming that the reader has

some knowledge on microprocessors or by erroneously assuming that such knowledge is not necessary. Unfortunately, this is probably the critical part of the whole subject that lays out a groundwork for the student to understand how the microcontroller works and how to use it. The microprocessor implements a set of instructions known as a program, which is stored in the memory unit. Students need to know that each microprocessor is built with a **set of instructions** in binary digits, which specify how the microprocessor manipulates the data in the program. A computer program, written by the user is a series of microprocessor instructions arranged in a way that achieves a particular intended objective. In implementing a computer program, the microprocessor reads one instruction at a time sequentially; that is why it is known as a sequential machine.

After students understand the meaning of a computer program, specifics on how the processor interacts with the memory, input and the output units can be described. Importantly is the fact that these units are assigned unique integer identifiers known as as addresses; so in order for the processor to access a particular unit, it has to locate its address first and use it to access and implement the intended operation on the unit. The information flow paths between the microprocessor and other parts of the computer are known as buses, which can be data buses, control buses or address buses.

Finally, the discussion on microprocessor is closed by discussing the core registers found in the microprocessor. It is important to stress that the registry unit of each microprocessor has several registers, and different microprocessors can have different registers. Common core registers found in all microprocessors must be described as follows:

1. **The Instruction register (IR)** for storing the instruction that is being implemented by the core. Normally, each instruction to be executed is loaded into this register and stays there while being decoded and prepared prior to being executed.

2. **The Accumulator Register (A)** for storing the *data* that is manipulated by the core; its size determines the word size of the processor, i.e. the number of bits that can be handled by the processor at any given instant. Data from the memory unit **must pass through the accumulator** in order to be manipulated by the microprocessor.

3. **The Index Register (I)** for modifying the address of the operand in a way that makes the operand easily accessible by the processor; they can also be used as accumulators. Typically, there can be more than one index registers in one microprocessor.

4. **The Program Counter (PC)** for keeping track of the CPU position in the program. The program counter register contains the memory address of the next program instruction.

5. **The Condition Code (CC) Register or Status Register** for storing information about the results of the last ALU operation. Its size can be one or two bytes; each of its bits is known as flag, which has a unique function. Common CC flag functions are summarized in Table 1.

6. **Stack Pointer (SP):** Stack is a special area in memory used by the processor for temporary storage of data; is is configured as a data structure that grows downward from high memory address to low memory address. The stack data is normally accessed by the last in first out

Center	Name	Function
C	Carry	Stores a 'carry' bit of the last operation. It indicates whether the result had a carry, or the operation borrowed one bit
V	Overflow	When set HIGH, it indicates that the actual result exceeded the capacity of the processor and therefore this result is not correct
Z	Zero	When set HIGH, it indicates that the final result was zero
N	Negative	This flag is set when the arithmetic operation produced a negative result
I	Interrupt	Maskable (that can be ignored) interrupts enabled. Programmer sets this flag, if set to 1, it means the CPU can ignore certain interrupts if set to 0, then the CPU cannot ignore these interrupts
H	Half Carry	This flag is set if the operation produced a carry from the lower 4-bits (nibble) to the higher 4-bits. It is useful for arithmetic operations using BCD
X	X-Interrupt	Non-maskable (not to be ignored) external interrupts enabled. This flag is set by the programmer
S	Stop Disable	This flag is also set by the programmer to disable the STOP instruction

Table 1: Typical Contents of the Condition Code register

(LIFO) philosophy.The stack pointer contains an address to the next free location in the stack.

The structure of the core registers defines what is known as the *microprocessor's programming model.* Although the presence of many high level programming languages have eliminated the need to know the programming model of the processor, developers of those high level languages must know the programming model for the intended microprocessor. It is important for students to know this model even if they will be developing their microcontroller applications using a higher level language. Debugging microcontroller programs requires investigating the contents of its core registry, which is possible only if the students know what register to investigate.

2.2.2 The System Memory

The system memory refers to an onboard memory used by the CPU to store data and the program that is immediately used by the CPU. It does not include hard disks or optical drives, which are regarded as part of the input/output units. Since students are assumed to have knowledge of D-flip flops and memory cells, it should be clear to them that the memory stores a series of binary digits only. The concepts of memory sizes in terms of *nibbles, bytes words, dwords* and *qwords* should be known to students at this time. Important is to inform them that the system memory appears in two forms: *volatile* system memory and *non-volatile* system memory, all microcontrollers are equipped with both.

Additionally, students must know the differences between the two forms of system memory. That the volatile memory is popularly known as **RAM (Random Access Memory)** although the term *random access* is misleading, because any addressable memory can be accessed randomly. This memory does not keep its data after the power is turned off. On the other hand, non-volatile memory, which is popularly known as **ROM (Read Only Memory)**, does keep its data even after power is turned off; however, special facilities are required to transfer data to this type of memory. Several types of non-volatile system memory exist; for example:

1. **ROM (Read Only Memory)** are factory-preprogrammed memory spaces containing programs that are used by the microprocessor, and cannot be reprogrammed. Programs stored on ROM get the most direct access to the microprocessor than programs from elsewhere.

2. **PROM (Programmable ROM)** are ROM areas that are designed to be programmed by the end user. However, after programming them, the user can not erase the program. In other words, they are one time programming chips.

3. **EPROM (Erasable PROM)** are PROM areas that can be erased and reprogrammed by the end user. Normally, erasure is carried out by subjecting them to ultra violet light, or to some electrical or magnetic field.

4. **EEPROM (Electrically Erasable PROM)** are PROM areas that can be erased and reprogrammed by subjecting them to high voltages, which makes programming the EEPROM a slow process. There are special types of EEPROM known FLASH memory units, these memories use simple special circuits for fast erasing and transferring data to the unit.

Since many microcontrollers are equipped with RAM, EEPROM and FLASH altogether, this knowledge enables students to figure out what memory to use and under what conditions.

2.2.3 The I/O Ports

The usefulness of any computer lies in its ability to interact with the user. The computer should be able to read information from the user and should also display results to the user. Information from the user is passed through the input units of the computer and the information to the user is passed through the output units. Common input units include keyboards, microphone and webcams; the output units include the monitor, audio speakers, and the printers. Data storage devices such as hard disks, and optical drives can serve as both input and output units. The means by which the computer communicates with the user are grouped together as I/O ports. At this stage, students need to understand that while the personal computer is built mainly to be a personal device interacting only with the person who is using it, microcontrollers are meant to mainly interact with the physical environment only getting back to the user as a way of accepting user inputs and probably displaying user results. As such it has to have plenty of flexibility in its I/O port structure.

2.2.4 The Basic Structure of a Microcontroller

Once students have a full understanding of a computer structure and microprocessors, they can now be taught that a microcontroller is a compact computer in which the microprocessor, system memory and the I/O interface units are packaged together in one very large scale integrated

(VLSI) circuit. Microcontroller I/O interface units are known as *Ports*; typically, one microcontroller can perform a variety of functions on its input/output ports depending on how their programmed. Between the microprocessor and the I/O ports, there is an array of registers known as Special Function Registers (SFR), which control the behavior of the I/O ports. The microprocessor accesses the I/O port and can modify the behavior of the I/O port through the ports' SFR. As such, each port can be made to serve as an input port or an output port performing one of the many possible I/O function of that particular microcontroller.

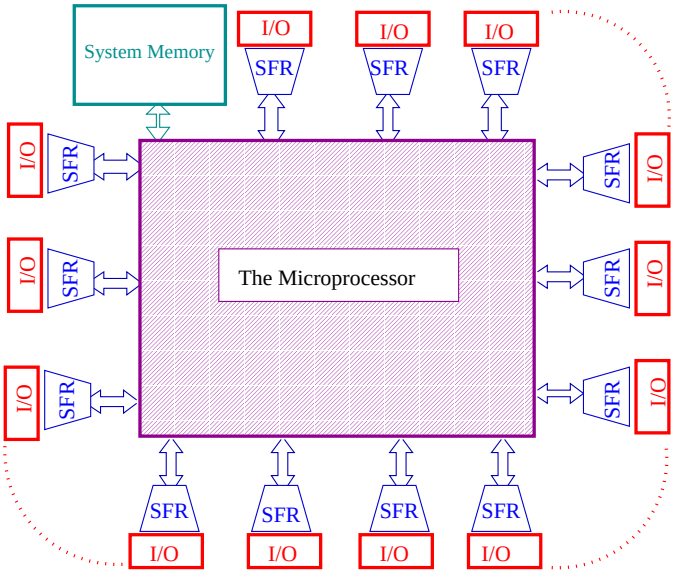


Figure 2: The general structure of any microcontroller

Figure 2 illustrates the general I/O structure of any microcontroller. Class experience with students who lack background in microprocessor systems has shown that Figure 2 sends a very clear image to students on how microcontrollers work, and also it heavily simplifies the learning process in the next topics on microcontrollers.

Since there are many types of microcontrollers in the market with different capabilities depending on the number and functions of its I/O ports and system memory, students must be taught to understand that each manufacturers provides all details on the critical information about each of their microcontrollers. To that end, students will be told that the microcontroller specific information found in textbooks is not universal, but it is rather provided by the manufacture for that specific microcontrollers. At this point students must be shown typical examples of manufacture data sheets, for example, for the MC9S12DP256 microcontroller by Freescale Semiconductor Inc[16], typical information include the I/O interface structure (Figure 3) which includes the pin assignments of the microcontroller, the system memory structure (Figure 4), and the programming model (Figure 5). Unfortunately, most textbooks on microcontrollers, tend to start at this stage only.

At this moment, students must be taught to understand that SFR bits have specific functions as determined by the manufacture; these functions are not universal for all microcontrollers, so for each microcontroller, the manufacturers data sheets must be consulted. Normally manufacturers provide such information in both graphical and textual forms. A typical SFR description for

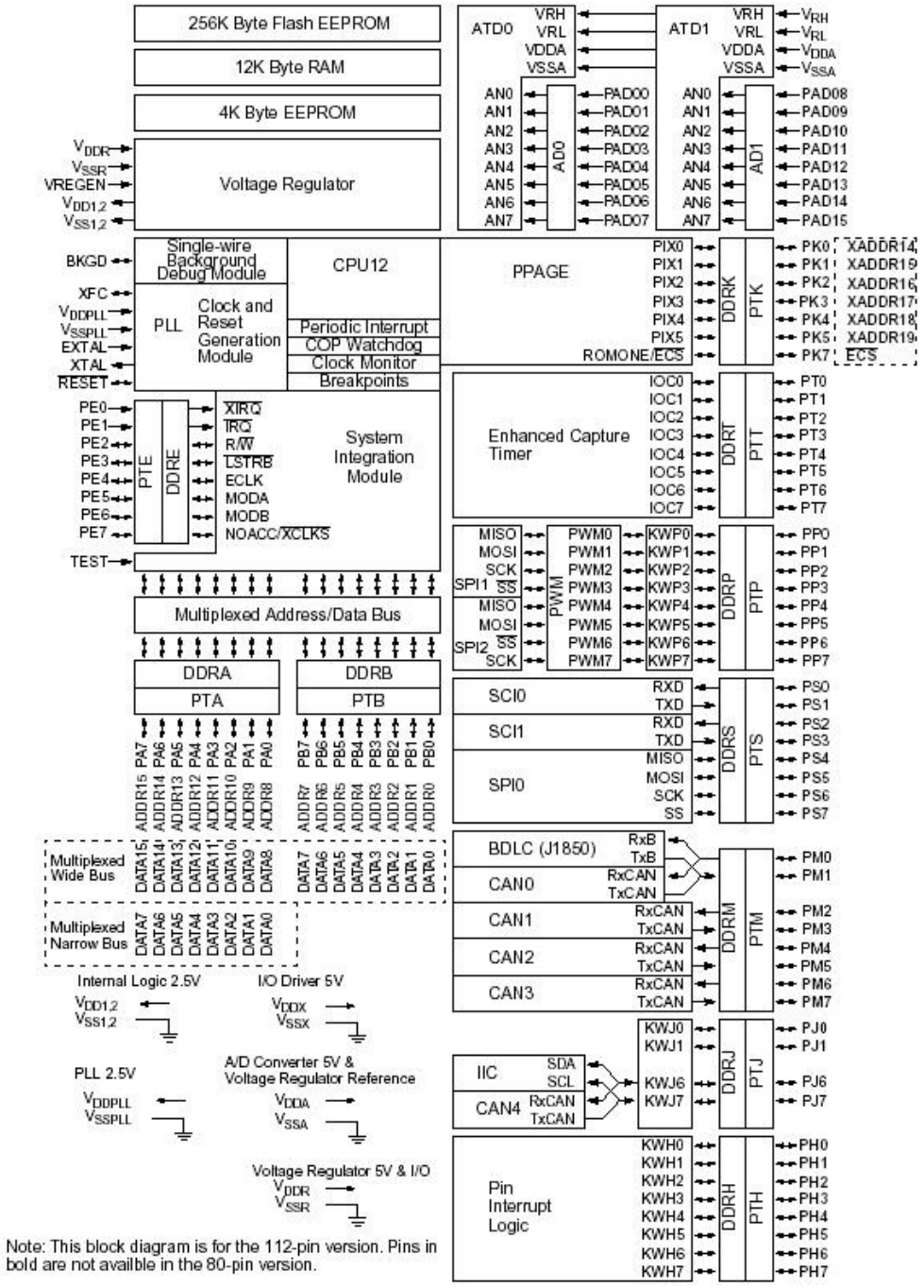


Figure 3: A Specific Structure of the MC9S12DP256 microcontroller by Freescale Semiconductor Inc [16]

the MC9S12DP256 microcontroller by Freescale Semiconductor is illustrated in Figure 6 for the general I/O port A of the MC9S12DP256 microcontroller [16].

By explaining such critical information about microcontrollers to students, they get to clearly know to figure out by themselves on how to choose and use particular microcontrollers for their applications.

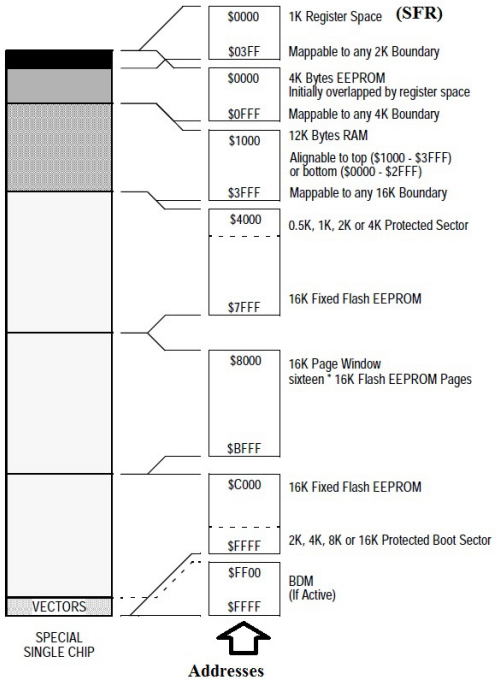


Figure 4: A specific memory structure of the MC9S12DP256 microcontroller by Freescale Semiconductor Inc [16]

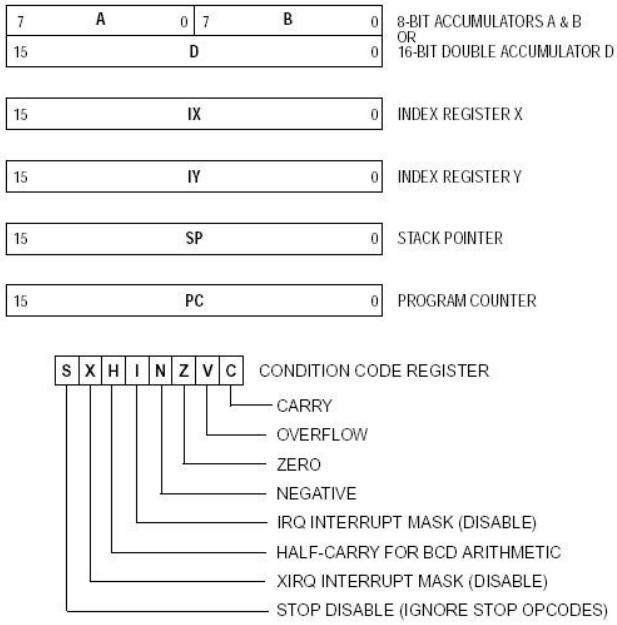


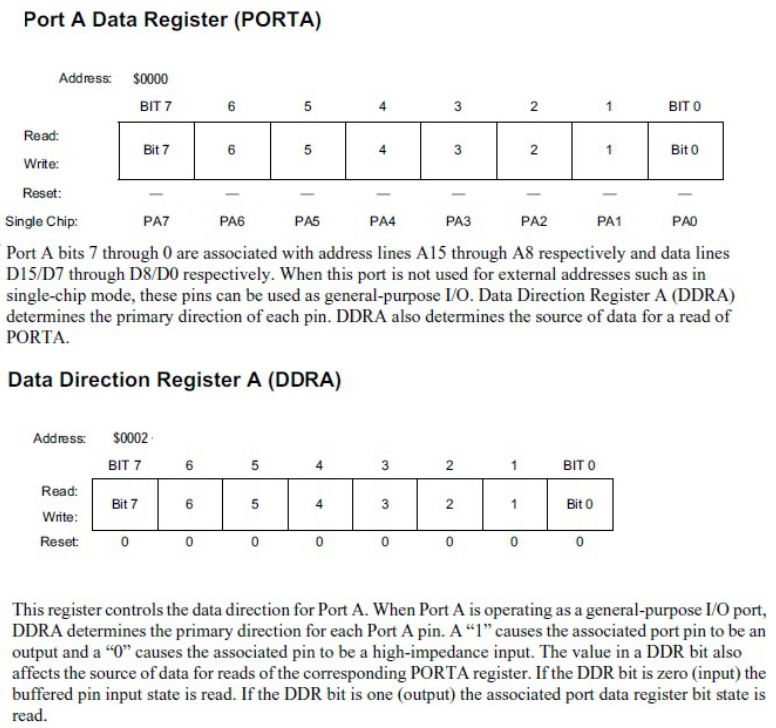
Figure 5: A specific programming model structure of the MC9S12DP256 microcontroller by Freescale Semiconductor Inc [16]

2.2.5 Microntroller Programming

Although plenty of high level programming languages exist, the training is more effective when students are first taught microcontroller programming by using assembly language. Despite its seemingly complexity, it provides a clear image to students on what is happening inside the microcontrollers. Fortunately, many instructors and text books tend to approach the subject in that way.

On teaching students how to program a microcontroller, first they must be informed that the process of programming any microcontrollers is primarily to instruct the microprocessor to either *write to* or *read from* the system memory or any of its SFRs. It involves setting ON or OFF certain bits of the microcontroller SFRs, and also reading certain bits of the microcontroller SFRs. Since, each cell of the system memory and all of the SFRs are assigned unique addresses, it is important for the programmer to know those addresses as provided by the manufacture. Later on students may learn that certain high level programming compilers have such information predefined, so the program can still be developed even if the programmer does not know all addresses. However, effective training must make it clear that such information is required.

Probably, the main reason microcontrollers are used in many real time control systems is not only their simplicity and speed but also their ability to flexibly manipulate any device connected on their I/O ports. As such, programming the microcontroller on how to interact with its I/O ports should be the primary learning objective for students. At that level, the instructor need to make it clear to students that I/O port programming for any microcontroller has two stages; the



Port A Data Register (PORTA)

Address: $0000	BIT 7	6	5	4	3	2	1	BIT 0
Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	—	—	—	—	—	—	—	—
Single Chip:	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0

Port A bits 7 through 0 are associated with address lines A15 through A8 respectively and data lines D15/D7 through D8/D0 respectively. When this port is not used for external addresses such as in single-chip mode, these pins can be used as general-purpose I/O. Data Direction Register A (DDRA) determines the primary direction of each pin. DDRA also determines the source of data for a read of PORTA.

Data Direction Register A (DDRA)

Address: $0002	BIT 7	6	5	4	3	2	1	BIT 0
Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	0	0	0	0	0	0	0	0

This register controls the data direction for Port A. When Port A is operating as a general-purpose I/O port, DDRA determines the primary direction for each Port A pin. A "1" causes the associated port pin to be an output and a "0" causes the associated pin to be a high-impedance input. The value in a DDR bit also affects the source of data for reads of the corresponding PORTA register. If the DDR bit is zero (input) the buffered pin input state is read. If the DDR bit is one (output) the associated port data register bit state is read.

Figure 6: Typical description of specific PORT SFRs of the MC9S12DP256 microcontroller as provided by manufacturer: Freescale Semiconductor Inc [16]

first stage is *port initialization* which is done once, and the second stage is *port data transfer* which can be repeated as many times as may be needed.

The initialization of an I/O port can be a simple process using only one SFR, or can be very complex process involving several SFRs; the programmer has to get all the necessary initialization information from the manufacturer's data sheet. There are up to three levels of initialization processes:

1. **Level 1:** The I/O port is initialized in one step only by **configuring** the port functionality. A typical example of this level is in using the general parallel I/O port A of the MC9S12DP256 microcontroller described in the previous subsection by defining whether the port is an *input* or an *output* by using the corresponding *data direction register*.

2. **Level 2:** The I/O functionality is activated in two steps: **Configure** the port functionality and **enable** it. A typical example of level 2 initialization is on setting up the pulse width modulation signal (PWM) ports for most microcontrollers. Although several SFRs are involved in setting up a PWM signal, the user needs to configure the desired PWM features such as frequency, polarity, and alignment; then enable the port to generate PWM signal continuously.

3. **Level 3:** The functionality is activated in three steps: **Configure** the functionality, **enable** it, and set up a **monitoring** mechanism so the functionality remains active. All interrupt driven functionalities fall in this level. For example, the RTI is configured by defining

its frequency and shape, then it is enabled; however, whenever the interrupt happens, the interrupt flags must be cleared in order for the functionality to continue running.

3 Experimenting with Specific Microcontrollers

With all the general information common to all microcontrollers provided to students as outlined in the previous section, the instructor can start to focus on a specific microcontroller used in the class. At this moment, it is possible to use two or more different microcontrollers in the same class as long as students understand the structure of their particular microcontroller and how to find the manufacturer's specific programming information. Best way to start playing with microcontrollers is to read and write on the general parallel I/O ports (GPIO); the experiment becomes more interesting and successful if the microcontroller has two or more GPIO ports. One port is configured to serve as an input while another one serves as an output, and the program becomes to read the input port and write the result to the output port. To run this experiment, dip switches need to be connected at the input pot while an array of LEDs are connected at the output port. By manipulating the dip switch, the LED should light to show the position of the dip switches.

4 Preliminary Effectiveness Evaluation

In fall of 2012, the author taught the topic to two groups of mechanical engineering students. These students needed knowledge of microcontrollers for use in their senior design projects. On the first group, the author used the standard approach of presenting the microcontroller, discuss and explain how to program it. Although in the end students were able to use the microcontroller in their project, they always had many questions to ask. The second group realized very late that they would need a microcontroller, so the author had to prep[are also a special training; that is when the author used the approach presented in this paper. The first group of students were invited again to sit in this session. After six days of discussing microcontrollers, al;l students were satisfied and with the knowledge, and continued working independently to the end of the semester. Although no statistical data was collected to evaluate the student perception, the fact that the first group struggled for most of the time until after sitting in the session of the second group is an indication that this approach may be effective. Future plans include rigorous evaluation of this approach using a big sample and better statistical methods.

5 Summary and Conclusion

This paper has outlined critical steps that must be followed by the instructor who teaches the subject of microcontrollers to students who have a limited background in electrical , electronic and computer engineering. It stresses on the need for students to be taught the general internal structure of microcontrollers and where to find specific programming information for any particular microcontroller. The paper believes that it is through careful lesson planning by the instructor that will make it easy for students who do not have any background in electrical an computer engineering to smoothly learn and understand how to design microcontroller based

systems specific for their areas of expertise, for example chemical and mechanical engineering. The proposals made by the paper are supported by preliminary effectiveness evaluation results conducted by the author on a limited group of students; further evaluation results on the proposed approach are planned.

References

[1] S. F. Barrett, D. J. Pack, Embedded Systems Design and Applications with the 68HC12 and HCS12, Pearson/Prentice Hall, Upper Saddle River, NJ, 2005.

[2] D. G. Alciatore, M. B. Histand, Introduction to mechatronics and measurement systems, McGraw-Hill Inc., Boston, 2007.

[3] W. Bolton, Mechatronics: Electronic Control Systems in Mechanical and Electrical Engineering, Pearson Education, Pearson/Prentice Hall, 2003.

[4] S. Cetinkunt, Mechatronics, Patria, 2007.

[5] A. Smaili, F. Mrad, Applied Mechatronics, Oxford University Press, 2008.

[6] A. K. Stiffler, Design with Microprocessors for Mechanical Engineers, McGraw-Hill Inc., New York, 1992.

[7] V. Giurgiutiu, J. Lyons, D. Rocheleau, W. Liu, Mechtronics/microcontroller education for mechanical engineering students at the University of South Calorina, Mechatronics 15 (2005) 1025–1036.

[8] D. G. Alciatore, Integrating Mechatronics Into a Mechanical Engineering Curriculum, IEEE Robotics & Automation Magazine (2001) 35–38.

[9] J. B. Hargrove, Curriculum, equipment and student project outcomes for mechatronics education in the core mechanical engineering program at Kettering University, Mechatronics 12 (2002) 343–356.

[10] A. B.Wright, Planting the seeds for Mechatronic curriculum at UALR, Mechatronics 12 (2002) 271–280.

[11] S. Meek, S. Field, S. Devasia, Mechatronics Education in the Department of Mechanical Engineering at the University of Utah, Mechatronics 13 (2003) 1–11.

[12] T. R. Hsu, Development of an Undergraduate Currculum in Mechatronics Systems Engineering, Journal of Engineering Education (1999) 173–179.

[13] N. Salzman, P. H. Meckl, Microcontrollers for Mechanical Engineers: From Assembly Language to Controller Implementation, Paper ID #6559, in: 120th ASEE Annual Conference and Exposition, Frankly We do Dive a D*mn, Atlanta, GA, June 23-26, 2013.

[14] M. F. Selekwa, Teaching Mechatronics Effectively in a Mechanical Engineering Program Under Limited Time, in: Proceedings of the 2013 ASEE North Midwest Section Conference, Fargo, North Dakota, 2013, Paper Number ASEE-NMWSC2013-0033.

[15] Unknown Author, The terms *receiving section, wharehouse, administrative unit,* and *shipping section* as used in this description were picked by the author from the internet several years ago. although the exact originator of the terms could not be known at this time, the author does not claim be the originator of these useful terms.

[16] Freescale Semiconductor, Inc., MC9S12DP256 Advance Information, Reveision 1.1, Technical Manual (2000).