International Journal of u- and e- Service, Science and Technology
Vol.7, No.2 (2014), pp.113-126
http://dx.doi.org/10.14257/ijunesst.2014.7.2.11

# Developing Low Cost Laboratory Apparatus for Hardware Interfacing System

Catherine S. Lacsamana[1], Tomas U. Ganiron Jr[2] and Harold S. Taylor[3]

*[1]Faculty of Science and Technology, University of Tartu, Estonia*
*[2] International Associations of Engineers (IAENG), Hongkong*
*[3]Department of Computer Engineering, Victoria University of Wellington, NZ*
*[1]catherinelacsamana11@gmail.com,[2]tomasuganironjr@gmail.com,*
*[3]haroldtaylor1113@yahoo.com*

***Abstract***

*Hardware interfacing system for visual C++ plays an important role in co-design of the embedded computer system. It links the software part and the hardware part of the system. The design process supports software interface implementation and hardware interface synthesis. This paper shows the hardware and software interfaces can be implemented by using low cost laboratory apparatus in embedded computer system, which includes the primitive interface, the synchronous interface and the data communication protocol between the hardware and the software.*

***Keywords:*** *Accelerometer, algorithm, hardware interfacing, visual C++*

## 1. Introduction

The effectiveness of the teacher, which is the very foundation of the teaching-learning process in the classroom, has been given various definitions and implied meaning. [1], saw it as the “teacher’s effect in the realization of some values” which is usually the attainment of some educational objective in terms of student behaviors, abilities, habits, or characteristics. Aside from knowing that classroom is a gathering of students with different values, practices, norms, and culture that adapted from their families, communities or region and in their classes since they started their formal schooling. The teacher must be innovative enough in adapting various techniques to become effective.

Instructional materials helps teacher to meet individual differences of the learners in the class by using aids that appeal to different senses. Instructional materials are used to supplement verbal explanation of concepts or any description so that the lesson could be real to the students. These instructional materials are categorized into audio visual, audio and visual. These are materials that when teacher used them can appeal to student both sight and hearing. These can be electronically operated materials like television, radio, film, slide motion; computer and non-electronic ones such as chalk board, charts, burners, models and many more. The absence of these materials in teaching of physics could make the class very uninteresting to student and discourage learning thereby lead to low or poor achievement. Instructional materials are very important because what students hear can easily be forgotten but what they see cannot be easily forgotten and last longer in their memory. In contribution of [2], to the importance of instructional materials to teaching and learning process, the primary purpose of instructional materials is to make learning more effective and also facilitate it and the teachers would not be able to do much where these materials are not

ISSN: 2005-4246 IJUNESST

available; therefore improvisation become necessary. [2, 3], posited that instructional media or materials can be used by lecturers to overcome noise factors, such as misconception, referent confusion and daydreaming. In this age of information and communication technology [ICT] teachers must be able to use available local resources to produce instructional and learning materials in schools. As good as improvisation might be in teaching and learning if learners are not taken part in the process of improvisation its aim may not be fully achieved. Learner participating in improvisation of instructional materials makes them exposed to creativity, innovation and curiosity, all of which are fundamental to teaching and learning of science [4]. Improvisation in science teaching is an important issue in science Education which has attracted a lot of contributions from science teachers [5].

In the University of Tartu, the Department of Physics uses different types of apparatus for data acquisition to perform experiments that demonstrate the different topics in the Physics subjects. The authors intend to fully automate these data acquisition apparatus using electronic circuits that convert measurable data through sensors to a readable bit format for the parallel port of the computer. A survey was done to determine which of the apparatus can be readily automated and can be replaced by readily available low cost materials. One of the experiments that showed possible automation was the calculation of the acceleration of the different moving bodies. It was observed that the error from this experiment was mainly due to the reaction time of the students. By substituting the apparatus with electronic circuits that are inexpensive, rugged and reliable, students can obtain data with ease and can focus more on the concept rather than its procedural and computational parts. Another advantage of automating the experiment through the computer is that a customized Graphical User Interface (GUI) can be used through the Microsoft Foundation Classes (MFC) of Visual C++. This also helps in the automatic data logging of the results.

## 2. Literature Review

Microprocessors can be defined simply as computers on a single chip. While the idea of placing all the required transistors for a computing on a single piece of silicon has been around since 1969, the first truly successful chip to do so was Intel's 8080. The 8080 was introduced in 1974 and became the heart of the world's first PC, the Altair. The chip, while state-of-the art at the time, is almost laughable by today's computing standards. The chip ran at 2 MHz, with an 8-bit architecture, and had 6,000 transistors. For comparison, an Intel Pentium 4, introduced in 2000, runs at 1.5 GHz, with a 32-bit architecture and has 42 million transistors [3, 6].

This humble chip gave birth to all microcomputers. As manufacturing technology improved, the additional silicon space was utilized for one of two purposes. The microprocessor, which sits at the heart of every personal computer, followed the path of using the extra silicon for more speed and computational power. Mathematical operations in the 8080 could be performed 8-bits at a time and usually took one clock cycle. By 1978, the Intel 8088 has moved onto a 16-bit architecture and 5 MHz clock speed, meaning not only could the additions be performed on numbers twice as large, but they could be carried out more than twice as fast.

On the other hand, in 1976 Intel introduced the 8748 which is considered the first microcontroller. This processor maintained the 8-bit architecture but added internal memory both in the form of random access memory (RAM) for temporarily storing information and erasable programmable read-only memory (EPROM) for storing code, which survived even when power to the chip, was turned off. Even today, microprocessors are dependent on external RAM and ROM (hard drives), while almost all modern microcontrollers have both types of memory internally. The designers also added additional input-output (I/O) lines so the chip could interface with other devices and an 8-bit timer, which counted the clock cycles so time-dependent events could programmed.

Chips like the 8748 became known as microcontrollers because they saw their biggest market as control systems, such as the ones you would find in washing machines and traffic lights. In 1980, Intel's 8051 became the standard for all microcontrollers. It contained over 60,000 transistors with 4 kB ROM, 128 bytes of RAM, 32 I/O lines, a serial port, and two 16-bit timers. To this day, most microcontrollers maintain the 8051 processor at their core. They have added more memory, serial ports, I/O lines, and other features such as analog-to-digital (A/D) conversion, but everything is backwards compatible to the original 8051.

### 2.1. 8051 features

A microcontroller, like any digital circuit, is simply a collection of transistors etched into silicon. These transistors, when arranged properly, allow for simple logical operations such as AND, OR, etc.; arithmetic operations such as addition, subtraction, *etc.*; and memory storage capabilities. A block diagram of these features is shown in Figure 1.

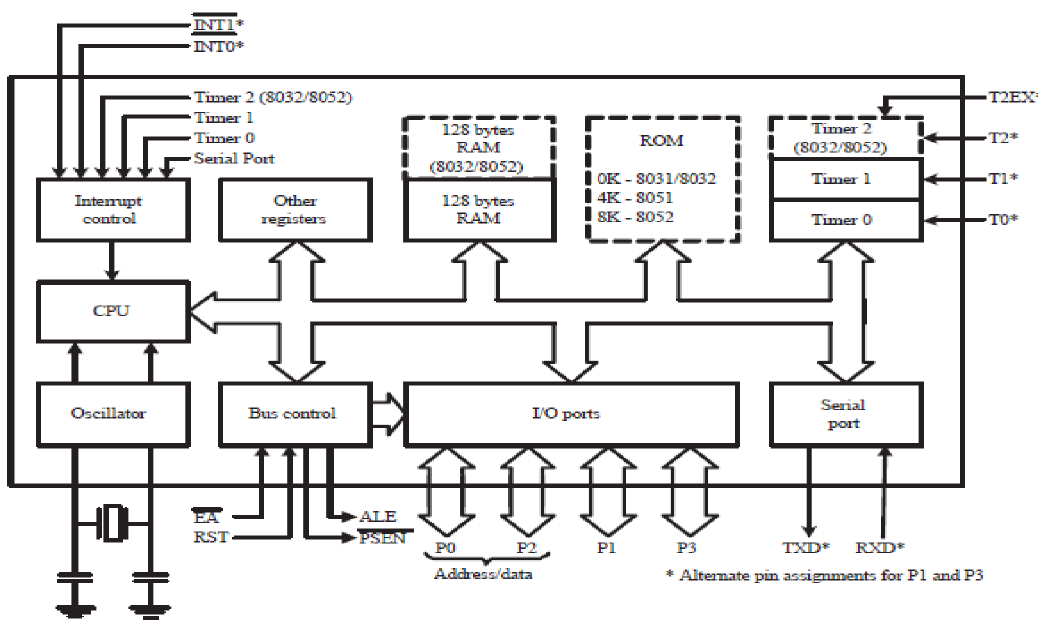



**Figure 1. Block Diagram of the Intel 8051 Microcontroller**

**2.1.1. Central Processing Unit:** The central processing unit (CPU) of a microcontroller, like the CPU of a computer, is the part of the chip that carries out the code provided by software by performing the required computations. The CPU in very simplified form, consist of five parts. The first is the program counter (PC). This is a memory array that holds the address of the next instruction in the code to be executed. On a chip reset, this array is populated with zeros so as to start the code at the first location. When the code begins, it will go to the first memory location, as indicated by the PC, and load the instruction in to the instruction register (IR). This simply holds the 8-bit instruction during its operations. The IR itself is rather mundane. The meaning of the code is determined by the instruction decode and control unit.

This unit processes the instruction and ensures that its purpose is carried out. Any arithmetic or logical operations that need to be carried out are done in the aptly named arithmetic and logic unit (ALU). This array of transistors is where the term 8-bit logic is applied. If the ALU is 8-bits wide, then the microcontroller is an 8-bit architecture since operations will be carried out 8-bits at a time. The two bytes to be operated on are loaded into the \top" of the ALU. As they descend \down" the ALU are acted upon (ANDed, ORed, carried over, *etc*.) as dictated by the instruction decode and control unit. The output from the ALU appears at the bottom. Finally, several registers, simply reserved memory locations, provide temporary space to house the bytes to be operated on and the results of the calculations. The CPU is connected to all other parts of the chip via data buses. In this way data from the memory (either internal or external), A/D converter, or serial port decoder can be used by the CPU and passed between the different chip peripherals. As an example, if an instruction in code is calling on the CPU to obtain a byte of memory from RAM, the instruction decoder will send the address for the desired byte along with a read instruction to memory and then process the returned byte sent into the CPU by memory. This byte would temporarily be stored in an unused register. The ALU would then act upon this byte as specified by the next instruction, again placing the result in a register. The result might then be passed back into memory, or used by a succeeding instruction that might perform a logical operation on the result, such as that called for by an \IF" statement in the higher-level code. The result of the logical operation could then be used to set the PC, leading to the code to jump to one of two subroutine locations, and so on. It is important to note that most programmers will never actually involve themselves directly in writing the instruction set carried out by the code. This machine language comes as the result of a compiler, a program that transforms higher level code such as assembly language or C into machine code.

**2.1.2. Memory structure:** The 8051 consist of two types of memory, ROM and RAM, both of which can be either internal or external. The ROM is used for code storage. In the original 8051, programmers would send their code to the factory to have it permanently etched into the internal ROM. Using internal ROM allowed for faster operation, but meant that code could not be changed once it was etched into the silicon. Later use of EPROM, which allowed memory to be changeable and yet retainable during power loses, meant almost all microcontrollers began using internal EPROM for code memory. The RAM is used for variable storage and functionality control. It is volatile, because its content is lost on power loss. There are four sections of RAM as shown in the memory diagram in figure 2. The first type of RAM is the general purpose RAM. This is what is used to store variables in the code. This can be accessed by the CPU as described in the example in the previous section. The second type of RAM is the bit-addressable RAM. This memory section allows the programmer to used single bit variables. This is useful because it is faster to read and operate on a single bit than operating on an entire byte, not to mention much more compact since in the other seven bits in a byte operation would go unused. The third group is the memory locations reserved to function as registers. There are four resister banks reserved at the beginning of the memory stack. While these locations can be used simply as general purpose RAM, they have the additional capability of serving as the registers for the CPU described in the previous section. Their biggest advantage is that the CPU can be programmed to see the address of these banks as a single byte rather than the usual two bytes required access other memory locations. This allows for faster operation, an important consideration when moving data into and out of the registers with nearly every operation. The final class of RAM is the special function registers (SFRs). The SFRs are used to control the operation of all the attached peripheral devices. The SFR location in memory was purposely sparsely populated so as to allow for the addition of future functionality. The exact operation of these SFRs will be discussed when their associated peripherals are discussed in more detail below.

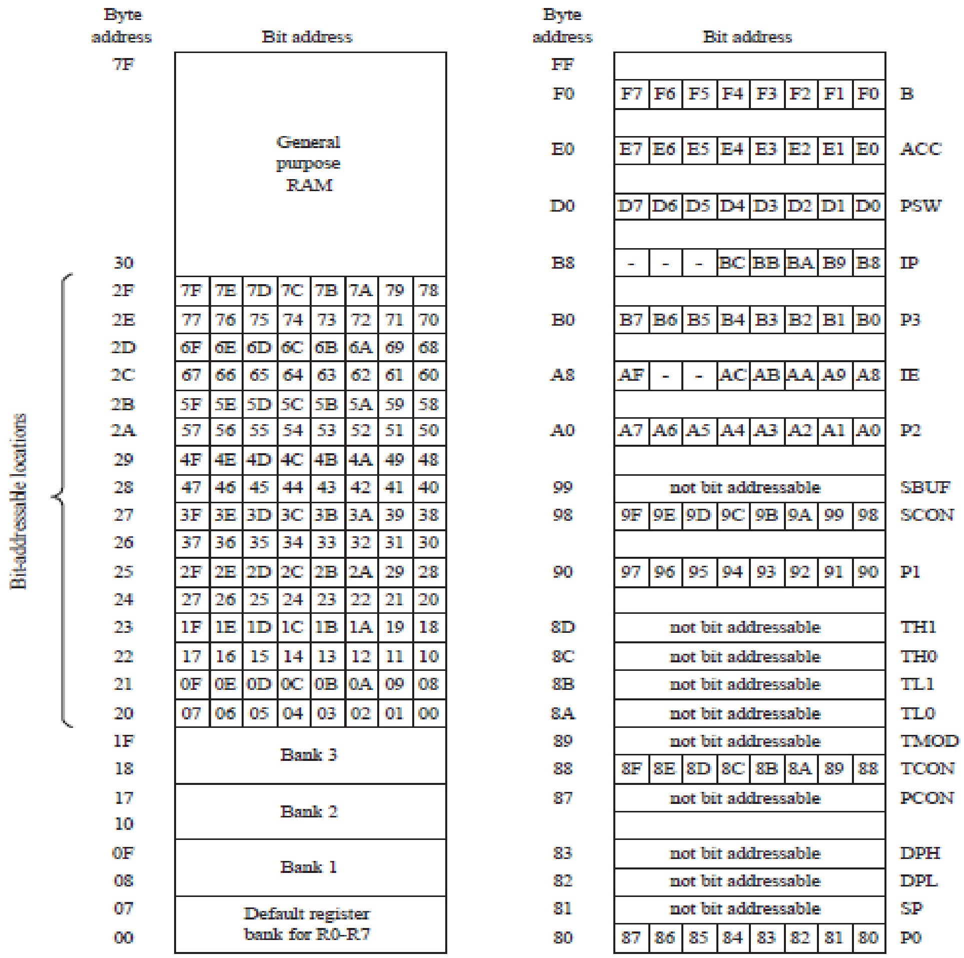



**Figure 2. 8051 RAM Memory Map**

**2.1.3. Ports:** The original 8051 had four, eight-pin, general-purpose input/output (IO) ports. These pins could be set either high or low by writing to the Port n (Pn) SRF, where \n" refers to the port number. When a one is set, the pin is pulled high internally. These ports can also be used as inputs, where an external device grounds the pin to make the read value of the Pn SFR a zero. The Pn SFR can either be written to or read from as a byte or one bit at a time. SFRs with this capability are referred to as bit-addressable and easily allow the user to change a single bit without concern for the values of the other bits in the register. The ports also served as pins for the other peripherals on the chip. Ports 0 and 2 serve as both the address and data lines for any external memory. Port 3 serves as the pins for the timers, external interrupts, and serial ports described below. Port 1 is always reserved for general I/O. If a pin is being used with a peripheral device, it cannot be used as an I/O pin. The original 8051 was not capable of sourcing or sinking much current. Later derivatives added direct sourcing and sinking capability meaning external devices, such as an LED, could be powered directly from the microcontroller pin rather than through a transistor. For this functionality, the user must specify whether the pin is to be a push-pull output or act as an open-drain input. This requires an additional SFR for each port.

**2.1.4. Interrupts:** Interrupts are event-driven interruptions of the main program. This allows the microcontroller to respond to important occurrences when they happen, rather than having to continuously check for them. Interrupts can be generated by any number of sources, primarily from the on-chip peripherals. For example, the serial port system can generate an interrupt saying a byte was just received. The A/D system can generate an interrupt saying its conversion is complete. A user can also generate an interrupt by pushing a button tied to one of the external interrupt pins. Note that these can occur at any time in the codes operation and their exact timing cannot be predicted. When an interrupt is received, a flag is set in one of several SFRs corresponding to the interrupt source. Setting this flag initializes the interrupt service routine (ISR). The ISR will store the current memory pointer for the code, load the PC with the memory location for the routine to be executed based on the interrupt source, and at the completion of that routine, point the main program back to its original place. In this manner, the CPU can give the appearance of performing several operations simultaneously, although in reality it is simply hopping from one subroutine to another. .Interrupts must be either enabled or disabled by setting the Interrupt Enable (IE) SFR. This SFR acts as the on-o_ switch for all the interrupts (later 8051-derivatives with more peripherals have added additional SFRs). Each bit in the register corresponds to a particular interrupt. Setting a one in a position will enable that interrupt. Setting a zero disables it. In silicon, the memory location acts as an input to an AND gate along with the interrupt flag bit. Only when a one is present in both, will the ISR be activated. Bit seven of the IE register is the global interrupt enable. All interrupts, regardless of the individual states are only enabled when this bit is set to one. This allows the programmer to easily suspend all interrupt occurrences without having to disable each individual interrupt source. To avoid conflicts of multiple interrupts, or receiving one interrupt while executing the routine of another, a very specific interrupt polling sequence is enforced. The programmer can modify this sequence slightly by setting each interrupt to one of two priority levels using the Interrupt Priority (IP) SFR. The higher level interrupts are first executed and the then lower level ones. The order within the level is still fixed by the polling sequence. When multiple interrupts do occur, the lower level interrupt is held and executed after the higher level one is finished. It is worth mentioning that the reset pin can be considered to function as an interrupt that is always enabled. There are no means to disable the reset pin in software. On a reset, the chip is set back to its initial state and the PC set to memory location zero.

**2.1.5. Timers:** Timers are simply counters of a specific length that increment with each incoming clocking source transition. The original 8051 had two 16-bit timers. The clocking source can come either from the main system clock (in which case it is pre-divided by 12 so as to allow for reasonable rates) or from an external source such as reserved timer pin or an external interrupt pin. A counter works in much the same way a human adds long numbers. When the sum of two numbers is greater than ten, a one is carried to the next column and the remainder placed in the original column. When adding in binary, any number greater than one result in a carry. With only a fixed number of bits with which to work, an overflow occurs when the most-significant bit (MSb) generates a carry. The counter will go from having all ones to all zeros with a carry bit being passed out. The MSb's carry bit is used to set the timer interrupt flag. In this way interrupts can be generated at regular intervals. If the length of the timer were the only parameter that could be set, the possible frequencies that could be used would be quite restrictive. To obtain the full spectrum of possible frequencies, the user can initialize the timer to some value, thus generating the interrupt sooner based on the number of clock cycles left until overflow. There are four modes of operation for the timers as illustrated in Figure 3.

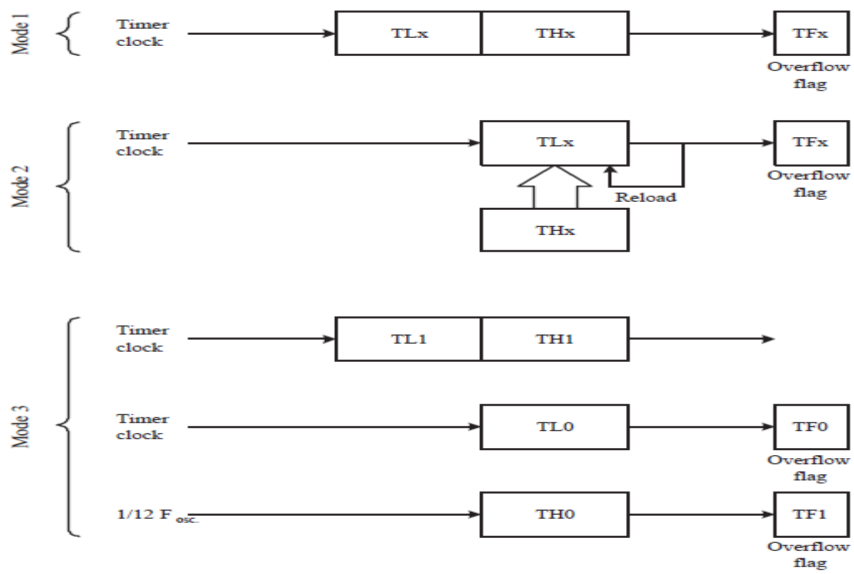



**Figure 3. Block Diagram of the Four Modes of Operation for Timer 0**

Mode 0 operates the timer in 13-bit mode, meaning the counter is 13-bits long. This mode is for backwards compatibility with the 8051 predecessor the 8048. Mode 1 operates the timer in 16-bit mode, meaning the full 16-bits are used for counting. The disadvantages of Modes 0 and 1 are that if the user specifies the time to overflow by setting the initial value of the timer, it becomes necessary to do so after every overflow to ensure regularly timed interrupts. Mode 2 makes use of a feature called auto-reload. The timer functions in an 8-bit mode with the lower byte serves as the counter and the upper byte holding the initialization value. On the 8-bit overflow, the lower byte is set to the value stored in the upper byte. Mode 3, which is only available for Timer 0, allows the timer to function as two independent 8-bit timers. All of the parameters mentioned above are set by the Timer Control (TCON)and Timer Mode (TMOD) SFRs. Modes, clocking sources, interrupt flags, and run enables for both timers are contained in these two SFRs. The values of the timers themselves can be accessed using the TH0, TL0, TH1, and TL1 SFRs where \H" and \L" refer to the high and low bytes and \0" and \1" to the timer number. Later 8051-derivatives have added additional timers and many have given those additional timers full 16-bit auto-reload capability. This is particularly useful for baud rate generation for serial port applications.

**2.1.6. Serial port:** The original 8051 came with a single serial port system. The serial port consist of two pins, TXD and RXD for transmit and receive, respectively. The serial port can be operated in one of four modes. Mode 0 allows for 8-bit synchronous communication. Mode 1 allows for 8-bit asynchronous communication. Modes 2 and 3 are the same as 0 and 1 except with 9-bits transmitted. The serial port modes, enables, and interrupt flag are set in the Serial Port Control (SCON) SFR. The two modes of interest are Modes 0 and 1. When in Mode 0, the serial port acts as a simple shift register. Data written in the Serial Port Buffer (SBUF) SFR is clocked out over the

RXD line one bit at a time at a frequency set by the system clock divided by twelve. The TXD line is used to pass out the clock pulses. The receiving party uses the rising edge of the clock to sample the data line. Incoming data also comes over the RXD line with the microcontroller continuing to generate the clock pulses over the TXD. Received bytes are also placed in the SBUF register. In Mode 1, the serial port operates as a universal asynchronous receiver transmitter (UART). This is the mode most computer users are familiar with from dealing with their computer's serial port. Here the RXD and TXD lines are used in their traditional receive and transmit roles with the timing on the lines coming from one of the timers. The original 8051 used Timer 1 to generate the baud rate, although later derivatives allowed additional timers to be used, as well as allowing different baud rates to be used for receive and transmit. The timer overflows are further divided by either 32 or 16 depending on the desired baud rate. For every byte to be transmitted, ten clocking cycles are used. First a start bit that is always a zero is sent, then the eight data bytes starting with the least significant bit (LSb) are transmitted, and finally a stop bit that is always a one is output. Interrupts are generated either after transmit or after a successful receive. For a successful receive, a proper start bit must be detected (as opposed to spurious noise on the line), and any previously received bytes have had to been read by software. Upon a successful transmit or receive, the TI or RI interrupt flags are set in the SCON register, and, if enabled, the serial port service routine activated. Since the same routine is called transmit or receive, it is necessary for the programmer is checking the status of the interrupt flags to see which occurrence caused the interrupt. For this reason, the interrupt flags are not cleared by hardware and must be cleared in software. Failing to clear these flags will result in missed receive bytes since received bytes will not be loaded into the SBUF register if the RI flag is not zero?

**2.1.7. A/D conversion:** While not an original feature of the 8051, A/D conversion, because if its importance to this project, will be briefly explanation below. The conversion specifics for the Signal C8051F020, the microcontroller used for this project will be described. The Signal C8051F020 has an eight-channel, 12-bit A/D system. The eight channels can function as either eight single-ended inputs, four differential inputs, or any combination of these two states (i.e. four single-ended and two differential inputs). The channel functionality is set by the Analog Multiplexer Configuration (AMUX0CF) SFR. The A/D converter is not capable of simultaneous sampling. Rather is uses only a single A/D converter with an analog multiplexer to select which of the eight lines to sample. The multiplexer is controlled by the Analog Multiplexer Channel Selection (AMUX0SL) SFR. An internal operational amplifier provides the capability of gaining the incoming signal. By setting the appropriate bits in the A/D Conversion Configuration (ADC0CF) SFR, gains of 0.5, 1, 2, 4, 8, or 16 can be set for the selected line. Conversions start based on the mode of operation of the A/D converter as set by the A/D Converter Control (ADC0CN) SFR. The four possible conversion start events are writing a one to AD0BUSY bit of ADC0CN, a Timer 2 overflow, a Timer 3 overflow, and a rising edge detected on the external Conversion Start (CNVSTR) pin. In all cases the AD0BUSY bit is set high. At the completion of a conversion, the AD0BUSY bit is set low, the conversion complete flag AD0INT is set high, and if enabled, the A/D Conversion Complete interrupt initialized. The result of the conversion is stored in the ADC0H and ADC0L SFRs.

## 3. Methodology

The implementation of the acceleration measurement device has gone through two revisions. The first revision consisted of four sensors that heavily relied on the algorithmic flow of the Visual C++ API [1]. The second implementation uses only two sensors and relies on the internal clock of the computer accurate up to the millisecond. Another revision is being designed that includes the use of advanced thread synchronization and embedding assembly language into the Visual C++ code to allow for more versatility and low-level accuracy.

### 3.1. Revision 1

Figure 4 is the schematic diagram of the revision 1 prototype of the accelerometer. A laser pointer was used for the light source and LDRs for light detectors. When the body interrupts the light, the potential (voltage) in the LDR will change. This signal (either high or low) will then be fed up to parallel port that will compute for the time elapsed.

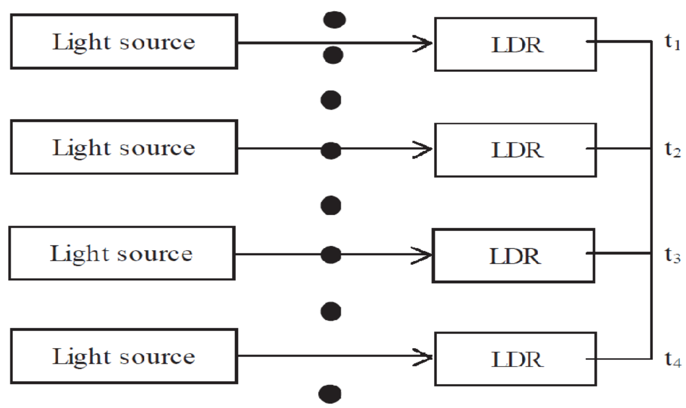



**Figure 4. Schematic Diagram of the Accelerometer**

Figure 5 is the state diagram of the application. Upon running the program, it enters the READY state. This state waits for a flag that is activated by the “Active” button. When the start flag is set, the program reads the parallel port where the four light sensors are connected.

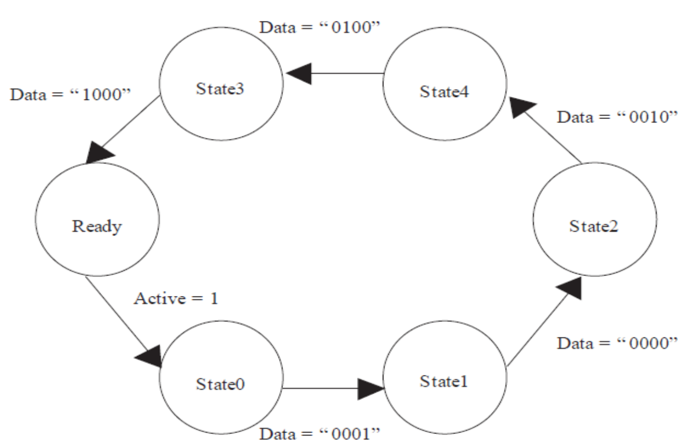



**Figure 5. State Diagram of the Accelerometer Application**

The state moves to STATE1 when the ball is placed on the first sensor and moves to STATE2 upon leaving the same sensor. It moves to the different states (STATE3 and STATE4) upon interrupting the next set of light sources and detectors. When the last light is interrupted, the program calculates the time elapsed between those points and outputs the velocity v and acceleration of the body [7, 8].

### 3.2. Revision 2

Figure 6 shows the schematic diagram of the acceleration measurement device. Similar to Revision 1, a laser pointer was used for the light source and phototransistor (WPT440F) for the light detector [3, 8]. The values of the sensors are constantly polled every millisecond and depending on the data taken through the parallel port, the time between the activation and deactivation of the two sensors are logged. [9, 10].

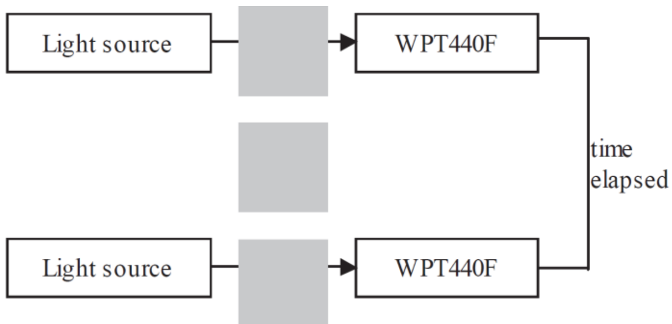



**Figure 6. Schematic Diagram of the Accelerometer**

## 4. Data Gathering

In performing the experiment for revision 1, the acceleration of a solid spherical ball was used. It was assumed that the frictional force between the ball and the inclined plane is negligible and therefore does not contribute to the torque brought about by its rotation [11, 12]. The angles 11° and 13.5° inclination for the plane were used and a 2cm diameter spherical ball. Ten trials were performed for each angle. The following formulas were used [13]:

$$a_{accepted} = \frac{5}{7} g \sin\theta$$

and

$$a_{experimental} = \frac{\sqrt{\frac{10gS\sin\theta}{7}}}{t}$$

where: g = gravitational acceleration (980 cm/s$^2$)

θ= angle of inclination (degrees)

S = displacement of the body (cm)

The above equations was obtained by considering the rotational and linear velocity of the body as it rolls down the plane [13,14]. The $a_{experimental}$ is the average acceleration of the body taken from its acceleration as it passes the three other sensors.

In performing the experiment for revision 2, the acceleration of a sliding body (collision cart) was computed. It was assumed that the frictional force between the body and the inclined plane is negligible. The angles 15° and 20° were used at a traveling distance of 62 cm. Ten trials were performed for each angle. The formulas used were:

$$a_{accepted} = gSin\theta \quad \text{and} \quad a_{experimental} = 2S/t^2$$

## 5. Result and Discussion

For revision 1, the acceleration computed by the program was compared to that of its theoretical value. An error of 13.63% was recorded when the angle is 13.5° and 16.38% when the angle is 11°. These errors were believed to be caused by the following: (1) during the release of the ball, the body is trying to overcome its moment of inertia causing its acceleration to deviate from its theoretical value; (2) just before the body reaches the last sensor, the effect of its rotational velocity becomes negligible, causing its acceleration to approach its linear acceleration only and (3) the delay (time response) of the LDR is suspected to be longer compared to that photodiode that are usually used as switch causing an error in the computed acceleration. The sources of errors in these data were used to form the second revision conditions such as the change from an LDR to a phototransistor and the change in the device being measured [15,16].

For revision 2, the acceleration computed from the experiments was also compared to the theoretical value. A 12.58 % error was found on the $15^o$ angle and an error of 7.20 % was found on the $20^o$ angle. The errors were attributed to the following: (1) data from the parallel port is not directly sent to the computing side of the processor due to the inherent delays between chipsets; (2) reliability of software on low level operations are limited to the millisecond due to the nature of the programming language used (Visual C++) and (3) the capturing of the internal clock may be limited to the delays caused by the multiple cycle commands of the different Visual C++ APIs[17, 18].

## 6. Conclusions and Recommendations

The prototypes in the conducted experiments were used to continually improve the methodology in creating a device that is low cost, easy to use and accurate for the experiments where it is going to be used. Revision 1 has shown the different sources of errors that are inherent in the first prototype. The sources of errors came from the object whose acceleration is being measured and the response time of the sensor (LDR). To eliminate the factor of the object trying to conquer its moment of inertia, a sliding cart was used, which is assumed to be frictionless with the inclined plane in Revision 2. A phototransistor was also used in the second revision so as to increase response time of the sensors.

The results of revision 2 have shown other sources of errors. The delay due to the different commands in the Visual C++ API is speculated to be one of the sources of errors. The lowest sampling time that Visual C++ can handle is 1 millisecond but due to the execution of other commands, the sampling time is not achieved consistently every 1 millisecond. This can be remedied by optimizing the Visual C++ code to perform the least number of necessary commands during operation and to include the use of using embedded assembly language code in the Visual C++ code to allow for more low-level control together with advanced thread synchronization. This is the planned design changes for the next revision.

Another recommendation would be to use a separate circuit in between the parallel port and the sensors as show in figure 7. The data logging circuit consists of a microcontroller whose sole function is to store the data sent by the sensors to a memory element that can later be retrieved by the computer through the parallel interface.

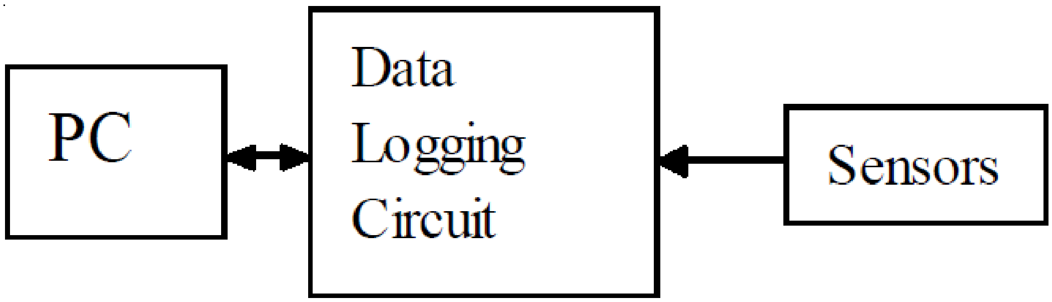



**Figure 7. Block Diagram of Suggested Configuration**

## References


[1] T. U. Ganiron Jr, D. T. Martinez and C. S. Lacsamana, “Development of Hardware Interfacing System for Visual C++”, International Journal of Advances in Applied Sciences, vol. 2, no. 4, pp. 1001-1005, **(2013)**.

[2] B. Marshall, How Microprocessors Work, How StuffWorks.com, (2003) August. Retrieved from http://computer.howstuffworks.com/microprocessor1.htm.

[3] S. Kama, “Evolution of the Trigger and Data Acquisition System in the ATLAS Experiment”, In Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 2012 IEEE, pp. 1787-1790, (**2012**).

[4] C. Hollabaugh, Embedded Linux: Hardware, Software, and Interfacing, Addison-Wesley Professional, **(2002).**

[5] F. Vahid and T. Givargis, Embedded System Design: A Unified Hardware/Software Introduction, vol. 4, New York, NY: John Wiley & Sons, (**2002**).

[6] A. Bateman and I. Paterson-Stephens, The DSP Handbook: Algorithms, Applications and Design Techniques, Vol. 1, Prentice Hall, **(2002).**

[7] M. Vuletic, L. Pozzi and P. Lenne, “Programming Transparency and Portable Hardware Interfacing: Towards General-Purpose Reconfigurable Computing, In Application-Specific Systems”, Architectures and Processors, 2004. Proceedings of 15th IEEE International Conference, pp. 339-35, (**2004**).

[8] M. Vuletic, L.Pozzi, and P. Ienne, Programming Transparency and Portable Hardware Interfacing: Towards general-purpose reconfigurable computing, In Application-Specific Systems, Architectures and Processors, “Proceedings. 15th IEEE International Conference”, pp. 339-351, **(2004).**

[9] C. Chang and C. Jifeng, “Hardware/Software Interface Design”, The United Nations University, Macau, (**2001**).

[10] P. Garcia and K. Compton, A Reconfigurable Hardware Interface for a Modern Computing System, In Field-Programmable Custom Computing Machines, 15th Annual IEEE Symposium, pp. 73-84, **(2007).**

[11] H**. W.** Huang, PIC Microcontroller: An Introduction to Software and Hardware Interfacing, Cengage Learning, **(2005).**

[12] I. Belencio, M. L. Paredes, A.Yap and R. Flores, “Trends and Developments in Evaluating a Concrete Through Short Messaging System ”, BS Thesis, Civil Engineering Department, FEATI University, Manila, **(2003).**

[13] M. Baroncini , P. Placidi , G. Cardinali and C. Scorzoni, A 2003 Sensors and Actuators, pp. A109- A13, **(2003).**

[14] J. Zhang and A. Mason , IEEE Sensors, **(2004).**

[15] R. B. Dr, Fundamentals of Microprocessors and Microcomputers, Fifth edition, p. 311, **(1991).**

[16] A. Sagahyroon , T. Al-Khudairi, FPGA Based Acquisition of Sensor Data Proceedings of International Conference on Indian Technology, “ICIT 2004”, p. 1398, **(2004).**

[17] S. Thanee, S. Somkuarnpanit and K. Saetang, FPGA-Based Multi- Protocol Data Acquisition System with High Speed USB Interface, “Proceedings of the international Multi Conference of Engineers and Computer Scientists 2010”, vol 2, **(2010).**

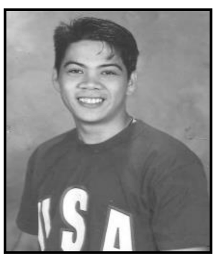

 

[18] Course Outline of Principles of Secure Software Design, University of Turtu, (**2013**). Retrieved from https://www.ut.ee/en/software-engineerin

## Authors

**Catherine S. Lacsamana** was born in San Nicolas, Ilocos Norte. She received both BS and MS Accountancy from De La Salle University, in Manila. Ms. Lacsamana has been pursuing research in financial accounting and corporate governance. She has published over 25 referred papers in relevant journals and conferences, which majority focus on the areas of social, environmental and ethical accounting and reporting, and corporate and social responsibility.

**Harold S. Taylor** received the BS Civil Engineering degree at University Canterbury in 1983 and earned his MS (1985) and PhD (1987) degrees in Civil Engineering at University of Auckland. He has published several research papers in national and international journals in the area of hydrology, water resources engineering, waste treatment plant and river engineering. He has also authored books viz, Ocean Engineering, River and Sediments, Water Resources and Engineering, and Fluid Mechanics Students' Handbook.



**Tomas U. Ganiron Jr.** This author obtained his Doctor of Philosophy in Construction Management at Adamson University (Philippines) in 2006, and subsequently earned his Master of Civil Engineering major in Highway and Transportation Engineering at Dela Salle University-Manila (Philippines) in 1997 and received Bachelor of Science in Civil Engineering major in Structural Engineering at University of the East (Philippines) in 1990. He is a registered Civil Engineer in the Philippines and Professional Engineer in New Zealand. He has received the Outstanding Civil Engineer in the field of Education given by the Philippine Media Association Inc. (1996), ASTM Award CA Hogentogler (2008) by IPENZ in New Zealand and Outstanding Researcher (2013) in Qassim University, Buraidah City.