A Mobile System to Monitor Neonatal Nursing Characteristics

Gedeon K. Nyengele (Georgia Perimeter College, Electrical Engineering), *SUNFEST Fellow*

Professor Jay N. Zemel, Electrical and Systems Engineering.

Abstract*— Neonatal development is considered a complex process to monitor because, due to the inability of neonates to effectively communicate, the majority of the information about neonatal physiology needs to be extracted by electronic means. Studies have shown that information about an infant's behavioral and physiological states can be acquired by analyzing parameters related to the sucking pressure and its frequency. Multiple attempts have been made in the development of devices capable of monitoring neonatal behaviors such as breathing and feeding. However, the use of those devices is usually limited because they are costly, bulky, and hard to use. This paper proposes a design of a convenient, mobile, and energy efficient monitoring system (Neonur) that could be easily assembled and attached to a baby nutrient bottle. The monitoring system is equipped with a pyroelectric breathing sensor constructed out of polyvinylidene fluoride films and a standard disposable micro-electro-mechanical pressure sensor widely used in medical applications. The pyroelectric breathing sensor provides valuable information about the infant's respiratory state by generating electric currents that are proportional to the magnitudes of the small changes in temperature on the films produced by the infant's exhalation. Data gathered from the breathing and sucking pressure sensors is saved on the on-chip memory and later transferred to a computer via USB. The results indicate that this device is well suited for monitoring neonatal breathing and feeding characteristics, is easy to operate, and is cheap to produce.

INTRODUCTION

Feeding, as carried on infants, is a sequentially coordinated process that includes sucking, swallowing, and breathing. The sequential flow of the feeding mechanism makes feeding one of the most complex processes carried on infants

At-risk and, quite often, premature infants do not only have problems with feeding but they also constitute a very considerable percentage of the total number of neonates annually. In 2009 only, almost 12% of all neonates born in the U.S. fall in the category of at-risk or premature infants [1].

To optimize health conditions for at-risk and premature infants, issues that can have long-term effects on health must be recognized and taken care of at an early stage of neonatal development. The complexities associated with the behavioral or physiological studies of neonates result from the incapability of neonates to effectively communicate with the care givers. As a result, many of the attempted solutions to neonatal development issues rely on technological means to gather meaningful information on physiological states of infants. Studies have shown that certain feeding characteristics such as the sucking pressure and the frequency at which it occurs encapsulates important information that can help determine an infant's behavioral and physiological states. By analyzing successive sucking feeds or bursts in an infant's feeding session, it becomes possible to identify specific feeding patterns that could generally be classified as normal or abnormal tendencies.

In the U.S., a variety of technological solutions or devices for monitoring feeding characteristics of neonates have been developed from as early as 1963 [2]. Most of those devices were suitable only for research laboratories, were quite expensive, were not easy to assemble, and/or were uncomfortable to use. *Figure 1* illustrates a device developed in 1963 by the Hospital of the University of Pennsylvania with the collaboration of the Children's Hospital of Philadelphia. The device consists of a nutrient reservoir connected to a capillary which regulates the nutrient flow to a nipple. A pressure transducer measures the negative pressure due to the flow of the nutrient that results from a sucking action.

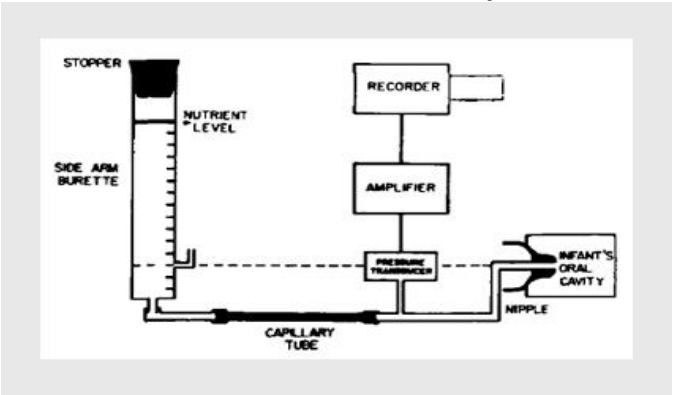


Figure 1 : System to measure neonatal sucking by Kron, Stein, & Goddard

Figure 2 shows a device that has been used by the Children's Hospital of Philadelphia since the 1980's. This multi-part device consists of an expensive processing unit wire-connected to a feeding apparatus. The device has a complex assembly structure and is time-consuming to clean after each feeding session. Also, the device has in total 13 different parts and still is uncomfortable to use because it restricts motion when in use.

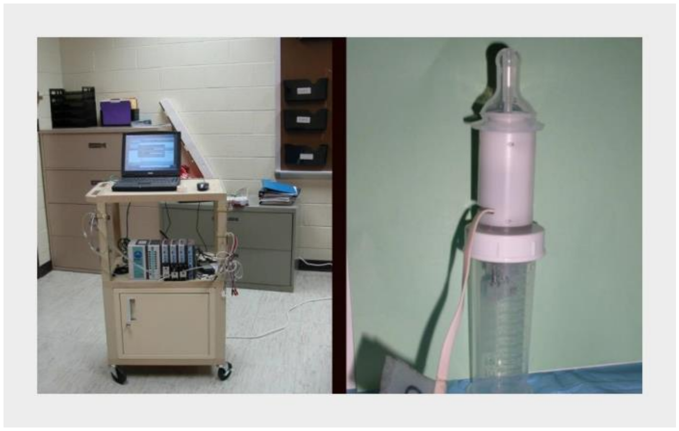


Figure 2: System developed by Litt and Kron in the late 1980's

Although technologically limited, this device is capable of providing informative data related to feeding such as the number of sucks in a burst (a fixed period of time constituted of several feeds) and the number of bursts in a feeding session.

It later became obvious that technological advances could be reasonably used in the design of a more advanced monitoring system and yet considerably easy to use. In 2008, Professor Jay N. Zemel, in collaboration with Medoff Cooper, Chen, and Rajendran, launched a design project geared toward the development of a monitoring system, later named *NEONUR*, with the end goal to have a simpler configuration, to be portable, to be easily maintainable, to be easy to operate, and to have an effective computer interface. *Figure 3* below shows the initial design of the Neonur.

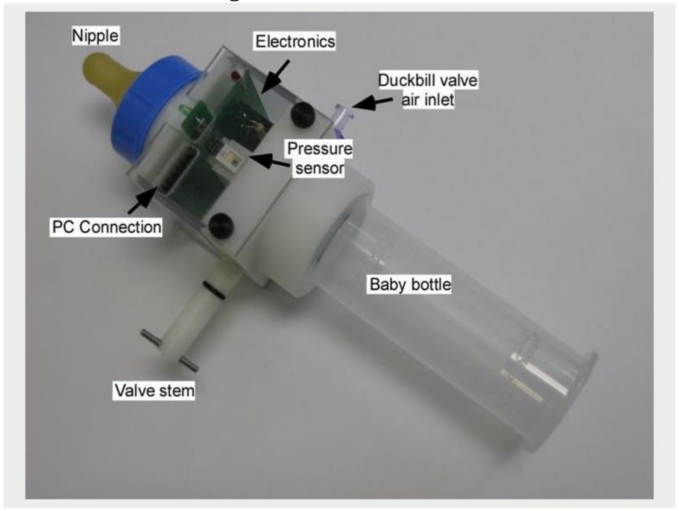


Figure 3: Neonur

The Neonur system consists of an adapted regular baby feeding bottle designed to house the feeding and breathing monitoring sub-system referred to as the actual Neonur. The overall monitoring system consists of three components: the feeding nipple, the nutrient bottle, and the measurement module. The feeding nipple and the nutrient bottle are standard components widely used in hospital nurseries. The measurement module consists of the sucking pressure sensor, the analog and digital circuitry, a fluid control valve, and a duckbill air inlet. The negative pressure applied to the feeding nipple due to suction is measured by the pressure transducer. The electric signal generated by the transducer is passed on to the microcontroller for acquisition, digital conversion, and storage. Although the current version of the Neonur device is capable of monitoring neonatal feeding characteristics, the device still has not successfully used a working and suitable breathing sensor. In addition, careful manipulations are needed to use the device as the system can be used only by someone with a strong technical background.. This paper proposes design solutions to commonly known communication issues in the Neonur and also proposes a better interfacing of an efficient breathing sensor constructed out of pyroelectric polyvinylidene fluoride films.

I. Background

A. Pyroelectricity

Pyro electricity is usually regarded as the ability of certain materials to generate electric signals when exposed to environmental temperature changes. Pyro electricity is exhibited only in crystallized non-conducting substances having at least one axis polar axis of symmetry [3]. A common usage of pyro electricity is in the design of pyroelectric thermometers, where temperature changes are determined by measuring the voltage induced by the separation of the charges in the pyro material. Another common use of pyro electricity is in the design of pyroelectric infrared sensors, the concept of which is very similar to that of pyroelectric thermometers. In the last decade, researchers have proposed the use of pyroelectric materials for battery charging.

This paper proposes the use of polyvinylidene fluorine films (PVDF) for the design of the breathing sensor. The studies of electric properties of polyvinylidene films started a large focus on its piezoelectric properties. *Figure 4* shows the PVDF used in the design of the breathing sensor next to a penny (U.S. 1-cent coin).

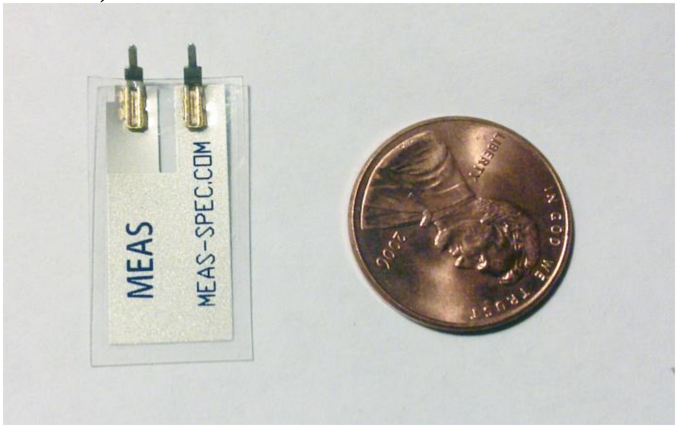


Figure 4 : Polyvinylidene film next to a penny

When poled - placed under a strong magnetic field to induce a net dipole moment - the piezoelectric coefficient of polyvinylidene films reaches 7 pC/N, which is approximately 10 times larger than that observed in any other polymer [4]. Also, polyvinylidene films show very efficient pyroelectric properties when poled, making them suitable for sensor designs.

B. *Piezoresistivity*

The piezoresistive effect is the change in the electrical resistivity of a material due to an applied mechanical stress. It is important to note that Piezoresistivity is a linear coupling between mechanical stress or strain and electrical resistivity. This property is commonly seen in semiconductors (Si and Ge), heterogeneous solids, superconductors, thin-metal films, Schottky barrier diodes, and Metal-Insulator-Metal (MIM) structures.

When mechanical strain is applied on a semiconductor such as silicon, a change in the energy band is created thus resulting in a change of the material's conductivity. Silicon and other semiconductors are frequently used for pressure measurement because of their sensitivity to mechanical strain. Generally, semiconductor sensors are encapsulated together with accompanying electrical circuits into small devices called Micro-Electro-Mechanical Systems. *Figure 5* shows the Freescale MPX2300DT1 pressure sensor module used in the design of the Neonur.

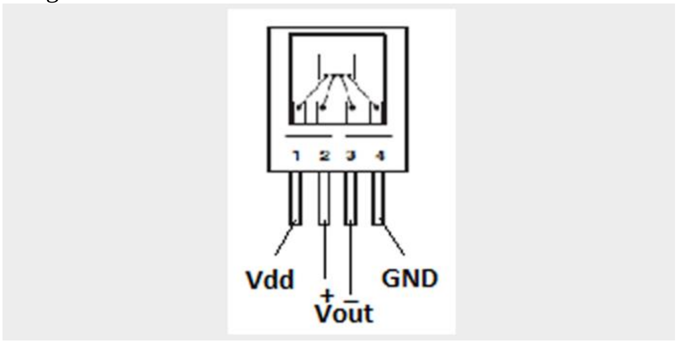


Figure 5 : Freescale MPX2300DT1 pressure sensor

C. *Microcontrollers and the PIC18F14K50*

Microcontrollers are essentially single-chip computers containing a processor core, memory, and I/O. A microcontroller usually incorporates other specialized components that are useful in embedded systems. For the most part, those components are serial ports (RS-232, USB, SPI, CAN, I2C, etc.), on-board memory (Flash, DRAM, SRAM, EEPROM, etc.), and analog I/O (ADC and DAC). The difference between a microcontroller and a microprocessor is that a microcontroller houses the processing unit and the peripheral units on a single chip whereas a microprocessor does not. The usage of microcontrollers can vary from one project to another. In general, microcontrollers are used in personal information products (cell phones, pagers, watches, calculators, etc.), in laptop components (modem, sound card, mouse, keyboard, etc.), in home appliances (alarm clock, air conditioner, remote controls, refrigerators, microwaves, etc.), in smart cards, in implantable medical devices, in toys, in automobile control systems, and in most devices with keypads. Microcontrollers are generally grouped into families based on the number of bits that are used as a unit – a word - by the processor. Modern processors usually have word sizes of 8, 16, 24, 32, and 64 bits. Although microcontrollers usually ship with a variety of built-in peripherals, all microcontrollers do not usually have the same peripherals built-in them. However, almost every microcontroller in the market has an internal memory, a clock, a CPU, Input/Output (I/O) capabilities, timers, interrupt controllers, and Analog-to-Digital Converters (ADC).

C.1. Clock

The microcontroller clock is the component that synchronizes the rate of execution of the program instructions. The pulses generated by the clock enable harmonic and synchronous operation of all the microcontroller's components. Some program instructions take exactly one clock cycle to execute while others require a couple clock cycles to execute.

Most clock sources for built-in microcontroller clock modules are based on the RC oscillator design. However, for time-critical operations, most designers use external clock sources such as standalone crystal oscillators, ceramic resonators, or a combination of crystals and microcontroller's built-in oscillator circuitry.

C.2. CPU

The CPU is commonly regarded as the "brain" of the microcontroller. It is the unit that executes the arithmetic, logic, and control instructions. Before executing any instruction, the CPU first fetches the instruction and the data to use in the operation on the data bus.

CPUs usually have a maximum clock rate at which they can reliably operate. As an example, the PIC18F14K50 microcontroller used in the design of the Neonur has a maximum clock speed of 12 MIPS (Million Instructions Per Second). *Figure 6* shows the PIC18F14K50 used in the Neonur.

Figure 6: PIC18F14K50 from Microchip, picture by author

C.3. Input / Output (I/O)

A significant difference between a microcontroller and a microprocessor is that a microcontroller has built-in hardware to deal with the external world. The microcontroller communicates with the world outside of it by its I/O lines. Most microcontrollers have more than a single I/O line and those I/O lines can be configured as input lines, for reading states from the outside world, or output lines.

C.4. Timers

Timers are internal clocks in the microcontroller. They provide a sense of time and duration during program execution. Usually, timer functionalities are provided at a clock rate that is a fraction of the system or main clock. In most microcontroller designs, timers are used as 8-bit timers or 16-bit timers. 8-bit timers can count from 0 to 255 whereas 16-bit timers can count from 0 to 65535. Using a reliable clock source, a good delay mechanism can be designed using timers. Most of the delay mechanisms implemented in the Neonur design use timers.

C.5 Interrupt Controllers

Interrupts are a mechanism which enables the microcontroller to respond to specific events, regardless of what the microcontroller is executing at that time. When an interrupt occurs, the microcontroller stops executing the current program flow and branches out to the interrupt handling routine. After the interrupt is handled, the microcontroller resumes program execution from the point where the interrupt occurred. Many of the microcontroller's functionalities are accomplished with the use of interrupts.

C.6. Analog-to-Digital Converters

In general, the signals generated by different objects in nature are analog. Microcontrollers, instead, are capable of detecting or reading binary signals. Binary signals provide information only about two defined states: ON or OFF (1 or 0). For TLL-based microcontrollers powered from 5 volts, an ON state could be any voltage below 2.5 volts whereas an OFF state could be any voltage above 2.5 volts. Luckily, microcontrollers have a built-in device into them that allows conversions of analog signals to a range of values that can be used in the microcontroller program. An Analog-to-Digital Converter (ADC) is very useful tool that maps analog voltages to numbers that can be used in electronics to interface to the world around us. With an ADC module, one can add sensors to their design and control the behavior of a system based physical quantities such as temperature, strain, light, sound, distance, etc. One of the most important characteristic of an ADC is its resolution. Resolution is a measure of how sensitive an ADC is to changes in the input signal. A 10-bit ADC module is more sensitive than an 8-bit ADC module. An 8-bit ADC module is capable of detecting 256 discrete analog levels whereas a 10-bit ADC module is able to detect 1024 discrete analog levels. Assuming that the microcontroller is powered from a 5-volt source, an 8-bit ADC module would be sensitive to voltage changes of the order of 0.02 volts or 20 mV (5 volts/256). This suggests that the ADC module does not differentiate a 0 mV signal from a 19 mV signal. However, a 10-bit ADC module would sense voltage changes of as much as 0.005 volts or 5 mV. But still, the ADC module would not differentiate a 0 mV signal from a 4 mV signal.

In order for programs to be stored on the microcontroller and for the microcontroller to execute its tasks properly, most microcontrollers are equipped with internal memory as either Flash, EEPROM, PROM, EPROM, ROM, and RAM memories. The microcontroller memory is usually divided into two separate memory blocks: data memory and program memory. The program memory is the location where the firmware (the program that runs the microcontroller) is stored. All temporary storage locations including calculation results and variables are located in the data memory.

For a successful use of the microcontroller, special memory locations are reserved for the microcontroller operations. These are called Special Function Registers (SFRs). SFRs are extensively used with peripherals built-in the microcontroller. For example, the ADCON0 register of the PIC18F14K50 microcontroller stores information about when an Analog-to-Digital conversion is completed. The ADRESH and ADRESL registers store the actual ADC conversion result.

D. *Serial Peripheral Interface (SPI)*

The SPI is a synchronous serial interface in which data, in 8-bit streams, is shifted in or out one bit at a time. In electronic designs based on microcontrollers, SPI can be used to communicate with a serial peripheral device or another microcontroller with an SPI interface. The SPI runs in a full-duplex mode meaning that the device implementing the SPI protocol can both receive and transmit data. An SPI device usually has 4 wire connections: Clock line, Serial Data Output line, Serial Data Input line, and the Chip Select line. The Serial Data Output line is used by the device to send packets of data out to another SPI device one bit at a time. The Serial Data Input line receives data packets sent from another SPI device one bit at a time. Usually, devices communicate in Master/Slave mode where the master device initiates communications. In that case, the master device is responsible for providing a Clock source to the slave device. The master device initiates communications by selecting an SPI device to communicate with. This is done my manipulating the Chip Select pin line. When driven low, the Chip Select pin state allows the slave device to respond to the master device's requests or commands. In master/slave mode, multiple devices can be configured as slaves working with a single master device. In that case, all the slave devices can share the Serial Data Output line, the Serial Data Input line, and the Clock line. The master device needs to provide separate Chip Select lines for each individual slave device. The M25P16 external flash memory used in the design of the Neonur implements the SPI protocol. *Figure 7* below shows the M25P16 memory chip used in the Neonur.

Figure 7: M25P16 from Numonyx, picture by author

Care must be given to the design of system that implements the SPI protocol because communications between devices are initiated by changes in the states of devices' pins. Most of the issues with the SPI protocol arise when a pin is not defaulted to a known state. For example, if the Chip Select pin is not pulled high initially, communication between devices would be compromised as the slave device might be selected while the master device did not intend to initiate a communication.

E. Universal Serial Bus (USB)

The Universal Serial Bus is a set of connectivity industry standards that define communication protocols used in communications between computers and electronic devices. The standards also define the connectors, the cables, and the electrical requirements that USB designers have to comply with for better device functionality. The USB specifications support four different bus speeds at which devices can operate. At Low Speed, introduced in USB 1.0, devices can communicate and transfer data at the maximum rate of 1.5 Mbps. At Full Speed, also introduced in USB 1.0, data transfers can occur at a maximum speed of 12 Mbps. The High Speed and SuperSpeed rates allow devices to communicate and transfer data at maximum rates of respectively 480 Mbps and 5 Gbps. The different speeds allow developers to select a speed range that is appropriate to the functionalities of a specific device. Also, devices running at a higher speeds can usually work on platforms supporting only lower speeds. Backward compatibility is strictly enforced by the USB specifications.

E.1 Advantages of USB

USB exhibits advantages and benefits for both device end users and developers. For end users, USB offers an easy-to-use interface. For the most part, users do not intervene in the computer setup of a USB device. When a user connects a USB device to a computer, the operating system on the computer recognizes the attachment of the device and automatically loads the appropriate driver for the device. Another major benefit of USB is its protection against data corruption. The USB protocols enable identifying data errors and notifying the sender to retransmit. Additionally, USB devices are usually built with power saving considerations. Reduced power consumption increases battery life for battery-powered devices, thus helping user to save money.

On the developer side, USB offers four different speeds and transfer types that can accommodate a variety of peripherals. Arguably the most important advantage of USB for developers is the support provided by operating systems. The different tasks performed by operating systems include detecting when devices are attached and removed from a computer, exchanging with newly connected devices to find out how data should be transferred, providing a link that enables software drivers manage communications between the USB device and computer programs that want to communicate with the device.

E.2 Disadvantages of USB

Although equipped with a variety of useful capabilities, USB, like any other interface, has several limitations. Perhaps the most fundamental one is its restriction on the distance between the device and a host. USB was designed as an expansion bus where devices are close to the computer or host. To allow longer distances, self-powered bridge devices are often used. Another limitation to the USB interface is the inability to broadcast. USB does not support sending data simultaneously to multiple devices. Usually, the host must send data packets to each device individually. Additionally, USB restricts all communications between a host and a device. The host might be a personal computer or any other device with host-controller capabilities. This suggests that Peer-to-Peer communications are not supported under USB. Hosts cannot talk to each other directly and devices cannot talk to each other directly.
The USB 3.0 specification however provides solutions to some of the USB limitations. For instance, The USB 3.0 allows developers to design systems using timestamp packets for the host to simultaneously communicate with multiple devices. The USB On-The-Go (OTG) option offers a partial solution to the Peer-to-Peer communication limitation. With USB On-The-Go, a device can function both as an end device and a host. This allows devices to communicate directly with other devices.
Arguably the most intrinsic challenge with USB is the protocol complexity. Extensive firmware is needed on the device to properly exchange on the bus. Also, although major operating systems do provide support and generic drivers for application development and communication with USB devices, vendor-specific drivers are usually required for augmented device capabilities not supported by the USB generic drivers offered by the operating system.
In addition to all the complexities and limitations, USB requires the use of a Vendor ID and a Product ID for a device to properly exchange with a host. Unfortunately Vendor IDs are not granted for free. The USB Implementers Forum (USB-IF), the people in charge of the USB specifications, charge a fee of about $2000 for a Vendor ID. The owner of a Vendor ID further assigns Product IDs to different devices. Chip manufacturers such as Microchip and Future Technology Devices International Limited (FTDI) will assign a range of Product IDs to a customer for use with the company Vendor ID in their devices at no charge. However, there is usually a limitation in the number of customer devices that can use the company's Vendor ID.

E.3 Common Uses of USB

Today, USB can be used for any device that has computer interfacing design. However, many of the USB devices on the market usually implement at least one of the defined USB classes because most USB devices have much in common with other devices that perform similar functions. The standard USB defined classes are Audio class, Communication class (CDC), Content Security class, Device Firmware Upgrade class (DFU), Human Interface Device class (HID), IrDA Bridge class, Mass Storage class, Personal Healthcare class, Printer class, Smart Card class, Still Image Capture class, Test and Measurement class, and the Video class. Many devices are built on top of these classes including keyboards, pointing devices, digital cameras, printers, portable

media players, disk drives, network adapters, video game controllers, medical devices, and more.
A class specification usually serves as a guide for firmware developers, application programmers, and driver developers.

E.4 Getting a USB device to properly run

Although a couple electronic components are needed for the design of a USB device, implementation of the USB protocol heavily relies on the firmware running on the device. Once connected to a host, the device firmware needs to be capable of identifying the device to the host (enumeration process). After device successful device enumeration, the device firmware needs to be able to handle data exchanges between the device and host. The exchanges include accusing reception of data packets, signaling errors in data transfers, properly responding to host requests, and managing power requirements set by the host.

E.4.1 Device Enumeration

Once the device is connected to the host, a couple transactions occur before the device can properly communicate with applications and drivers running on the host computer. Those transactions are meant to help the host learn about the device. The host needs to determine whether a device is a keyboard, a mouse, a speaker, a digital camera, a mass storage device, a printer, a network adapter, or any other defined type. After the host has learned about the device, the host assigns a proper driver to the device for further communications. Device enumeration is the process that accomplishes all the tasks mentioned above. The overall enumeration process consists of reading descriptors from the device, assigning an address to the device, assigning a driver to the device, and setting proper power requirements for the device. During enumeration, the device goes through a series of device states that shows progress in the process. Those states are the Powered state, the Default state, the Address state, the Attached state, the Suspend state, and the Configured state. During enumeration, the device must detect and respond to any enumeration request at any time. Also, the device should not assume any particular order in which enumeration occurs. However, in a typical enumeration, the process starts with the detection of the device attachment by the hub. The hub actually monitors the signal levels on D+ and D- lines. Initially, the D+ and D- lines are pulled down in the hub. A full-speed device would therefore pull up the D+ line, permitting the hub to identify the device as a full-speed device. Similarly, a low-speed device would pull up the D-line to notify the hub of the attachment of the low-speed device. After the hub has determined the nature of the attached device, the hub notifies the host of the attachment of the device and establishes a communication path between the device and the bus after resetting the device. Usually after this step, the host requests the device to provide different descriptors allowing the host to learn about the functionalities of the device, the number of interfaces the device implements, the meaning of the data sent by the device, the device power requirements, the device's name and type, and the device's manufacturer. Vendor IDs and Product IDs are used by the device to inform the host about the device's manufacturer. After successfully learning about the attached device, the host usually assigns and loads an appropriate driver for the device. The majority of these tasks, if not all, are carried by the device firmware. Therefore, writing device firmware for USB communication is considered a difficult task by most developers.

E.4.2 Device descriptors and endpoints

Device descriptors, as the name implies, are device description structures that are sent to the host during the enumeration process. They contain the information that allows the host to learn about the device as a whole or about specific device capabilities. The most important descriptors are the device descriptor, the configuration descriptor, the interface descriptor, and the endpoint descriptor. Standard descriptors contain a bLength field (one byte) that provides the size of the descriptor in bytes. In addition to the bLength field, standard descriptors also contain a bDescriptorType field that identifies the descriptor's type.
As for most serial interfaces, data transfers are conducted between devices' buffers. Usually, on one end of the communication bus, a device would put the data to transmit in a buffer. Data would then be sent on the serial bus bit after bit. On the other end of the bus, the receiving device would store the received bits in a buffer up until a byte or any other data size is received. Data buffers on the USB device side of the communication are called endpoints. Each endpoint on the USB device has an address consisting of an address number and direction. The USB specifications require at least one endpoint for successful communications with the host. This endpoint must have the address number 0. Endpoint numbers range from 0 to 15. Endpoint directions are either IN for storing data to be sent to the host or OUT for storing data received from the host.

E.4.3 USB transfer types

USB supports 4 different data transfer types which are control transfers, bulk transfers, interrupt transfers, and isochronous transfers. Control transfers are usually used by the host for enumeration and configurations. The host uses this transfer type to send requests to the device. In this type of transfer, data exchanges occur in both directions. Although control transfers are usually used for enumeration of devices, it can also support transfers of small amounts of data. In interrupt transfers, the host frequently polls the device to determine if the device is ready to send data to the host. Bulk and Isochronous transfer types are in a sense opposite types of transfers. In a bulk transfer, the data transfer rate is not guaranteed but data accuracy is preserved. In an isochronous transfer type, data transfer rate is guaranteed while data loss is permitted. A good example of devices that use bulk transfers is an audio speaker.

E.5 Human Interface Devices (HID)

The Human Interface Devices (HID) is one the most used USB class for devices because the HID class supports a variety of devices such as keyboards, mice, and game controllers and also because the class allows development of devices that perform vendor-specific or custom functions. In Windows operating system, the HID class was one the first classes implemented [citation – Jan Axelson]. To use the HID class, a device does not have to have a human interface. However, the device has to implement the requirements of the HID class. In a typical HID class implementation, data transfers occur with the exchange of reports. Reports are fixed-length data structures. Also, a HID device can have only one interrupt IN endpoint for storing data to be sent to the host.

For more information on the HID class and USB in general, readers are encouraged to take a look at Jan Axelson's *USB Complete book*. This best-selling developer's guide to the Universal Serial Bus covers all aspects of the USB interface including hardware design, firmware programming, and host application software development.

II. IMPLEMENTATION

II.1 Hardware

The Neonur device consists of three main components, which are the feeding nipple, the nutrient bottle, and the core measurement module. The feeding nipple and the nutrient bottle are parts of the regular baby bottles found in most nurseries. *Figure 8* below shows the different main parts of the Neonur device.

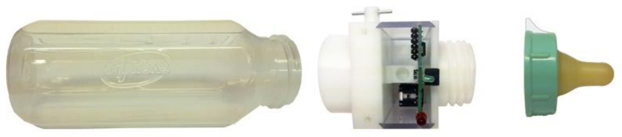


Figure 8: Neonur device main parts, picture by author

The core measurement module houses all the electronic components of the device. The main electronic components are the external flash memory chip, the voltage regulator, the sensors and instrumentation amplifier, the microcontroller, and the USB port.

II.1.1 External flash memory

The Neonur uses a 16-mega-bit external flash memory for data storage. The chip used is the regular M25P16 16 Mb serial flash memory unit from Numonyx.
The Neonur is capable of carrying measurements every 5 milliseconds. Each of the measurements consists of a pressure sensor reading and a breathing sensor reading. Each of the readings is stored as an 8-bit value. With these specifications, the memory chip is capable of storing 1,048,576 measurements or 87-minute-long feeding cycle.

Figure 9 below shows the M25P16 memory chip used in the Neonur and its pin configurations.

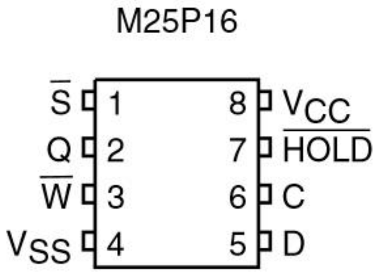


Figure 9: M25P16 schematic, picture from Numonyx

Pin 1, referred to as Chip Select pin, is pulled up to avoid the chip being selected at random. Pins 3 and 7, respectively referred to as WRITE_PROTECT and HOLD pins, are pulled up as well because neither the WRITE_PROTECT or HOLD functionalities are implemented by the Neonur firmware. The reason for this is that the memory chip has a built-in WRITE protection mechanism that requires specific wave forms to perform any write operation. Pins 4 and 8 connect to Ground and the positive voltage Vcc respectively. Pins 2 and 5 connect to the microcontroller's pins 13 and 9, which are the Serial Data Input and Serial Data Output pins respectively. The memory chip's pin 6, referred to as the Serial Clock source, is connected to the microcontroller's pin 11 (SCK). This pin receives the clock signal from the microcontroller to allow the memory chip to synchronize and exchange with the microcontroller. The Serial Clock pin on the memory chip is pulled down to avoid random fluctuations in the signal.
This mode of operation between the microcontroller and the memory flash is also referred to as Master/Slave mode, where the microcontroller performs as the master and the memory chip performs as the slave. The microcontroller takes the Master responsibilities because it provides the clock signal to the memory chip and it also initiates all the exchanges with the memory chip.
All pull-up and pull-down resistors used with the memory chip are valued 100 Kilo Ohms resistors. However, any value as low as 4.7 Kilo Ohms could still be used. The reason for the use of the 100K resistors is to limit the amount of current wasted by pull-down mechanisms to a very minimum.

II.1.2 Voltage Regulator

The Neonur device is powered by a 3-volt battery (CR-1/3N) during data acquisition. Once connected to the USB bus for data transfer, the power source is switched to USB. In this mode, the Neonur is powered from the USB bus. Two design concerns arise from this mechanism. First, careful attention needs to be given to the coupling of the two power sources as the USB specifications require that no current must be flow back to a host computer. To solve this problem, a switching mechanism is implemented. A switch is placed between the two power sources so that only one source can power the device at any given time. The second concern is that the USB bus provides a 5-volt signal whereas components such as the external flash memory can support only up to 3.6 volts. To resolve this concern, a voltage regulator is placed at the output of the switch explained above. The voltage regulator ensures that the signal fed to the circuit will not exceed 3.3 volts, which is a voltage that all components on the board can safely work with.

II.1.3 Sensors and Instrumentation Amplifier

1. Sucking pressure sensor

A pressure sensor module (Freescale MPX2300DT1) is used to record infant’s sucking activity. The Freescale sensor is a standard disposable micro-electro-mechanical module used in a variety of medical applications.
The sensor pins were bent at 90 degrees to allow the sensor module to be easily interfaced with the circuit board. The sensor module is further glued with silicon glue to a frame that allows the sensor to be easily pluggable to the Neonur core frame.

2. Breathing Sensor

The pyroelectric breathing sensor is constructed out of pyroelectric films. In the design of the Neonur, a polyvinylidene film was used. The film is first mounted on a frame that is further attached to the nipple of the nutrient bottle. *Figure 10* below shows the design of the breathing sensor.

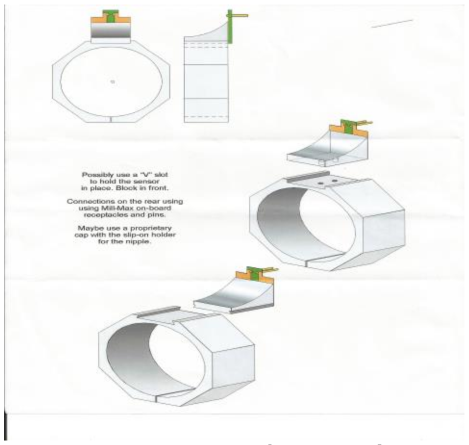

Possibly use a "V" slot to hold the sensor in place. Block in front.
Connections on the rear using using Mill-Max on-board receptacles and pins.
Maybe use a proprietary cap with the slip-on holder for the nipple.

Figure 10: Breathing sensor design, picture by author

3. Instrumentation Amplifier

Signal conditioning for both the breathing sensor and the sucking pressure sensor was of high importance in the design of the Neonur.
The Freescale pressure sensor outputs signal voltages in the range of 0 – 450mV. The voltages in this range are not well-suited for use with the microcontroller’s analog-to-digital converter which has a broader range of 0 – 3 volts when powered from a 3.3v battery. The INA 2126 instrumentation amplifier is used to amplify the signal from the sensor to the microcontroller. However, the signal from the sucking pressure sensor is inverted so that measurements on large negative pressures could be recorded.

As for the pyroelectric breathing sensor, the fluctuations in signal strength can be very large. Hundreds of volts can be easily obtained from the pyro films. However, the microcontroller ADC module does not tolerate voltages of more than 3 volts. To solve this problem, a couple techniques were used. First, the signal from the pyro sensor is rectified with the use of a Schottky-based bridge rectifier. This process makes the whole signal positive. Once the signal is rectified, a Zener diode is used to limit the output voltage of the pyro sensor to a suitable maximum voltage to be fed to the instrumentation amplifier. A resistor-capacitor pair is also used to filter the signal so that unwanted ripples can be eliminated.
Figure 11 below shows the overall design of the Neonur as accomplished in Eagle CAD software.

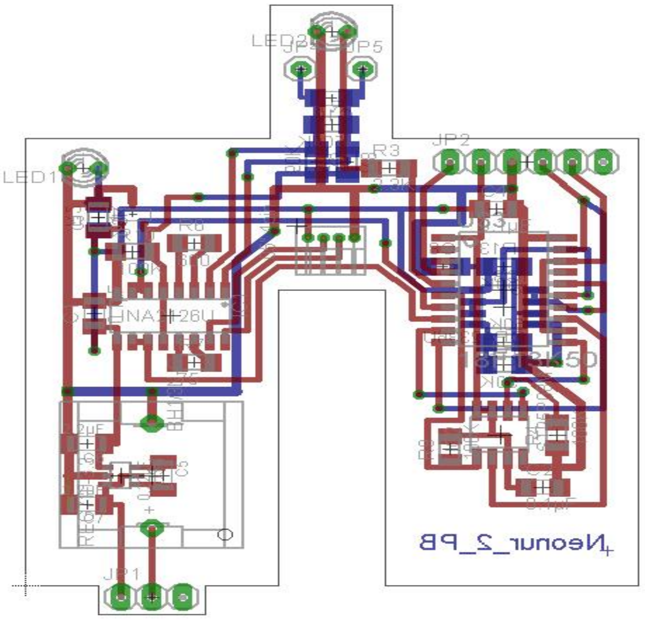


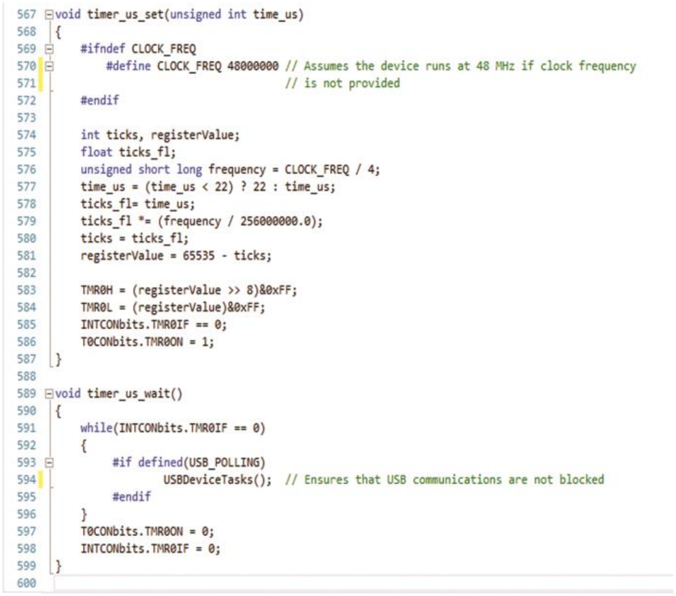
Figure 11: Neonur CAD design, picture by author

II.2 Software

For proper functioning of the Neonur, a firmware was written to assist the device with its tasks. In general, the firmware written for the microcontroller PIC18F14K50 has three major roles to play: Acquire data through the ADC module, allow data transfers between the microcontroller and the external flash memory, and finally communicate with the computer application for data upload through USB. All the code snippets provided in this article assume the following: use of PIC18F14K50 as the microcontroller and use of C18 as the compiler.

1. Data Acquisition

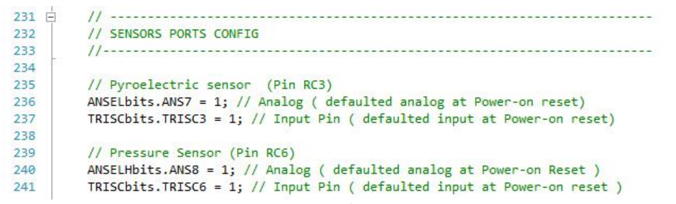
The Neonur device has a capability to perform a measurement every 5ms. To accomplish this task, it was necessary to create a software routine that would help in keeping track of this time interval. *Figure 12* below shows a routine written in C programming language to help determine when the 5ms are reached.



```c
void timer_us_set(unsigned int time_us)
{
    #ifndef CLOCK_FREQ
        #define CLOCK_FREQ 48000000 // Assumes the device runs at 48 MHz if clock frequency
                                    // is not provided
    #endif

    int ticks, registerValue;
    float ticks_fl;
    unsigned short long frequency = CLOCK_FREQ / 4;
    time_us = (time_us < 22) ? 22 : time_us;
    ticks_fl= time_us;
    ticks_fl *= (frequency / 256000000.0);
    ticks = ticks_fl;
    registerValue = 65535 - ticks;

    TMR0H = (registerValue >> 8)&0xFF;
    TMR0L = (registerValue)&0xFF;
    INTCONbits.TMR0IF == 0;
    T0CONbits.TMR0ON = 1;
}

void timer_us_wait()
{
    while(INTCONbits.TMR0IF == 0)
    {
        #if defined(USB_POLLING)
            USBDeviceTasks();  // Ensures that USB communications are not blocked
        #endif
    }
    T0CONbits.TMR0ON = 0;
    INTCONbits.TMR0IF = 0;
}
```

Figure 12: timing routines in C, picture by author

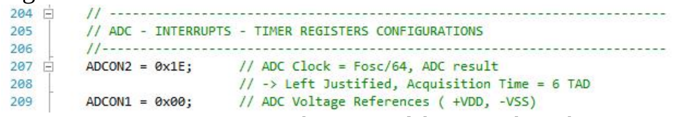
The routine consists of two functions. The first function *timer_us_set()* initializes a point in time when to start measuring time. The second function *timer_us_wait()* allows the device to wait for the 5ms to be consumed before performing the next measurement.
--CODE
The code snippet provided above assumes that the device is running at 48 MHz.
Along with timed measurements, the ADC module needs to be properly configured before any use.
To use the ADC module, the sensor pins need proper configurations. In the case of the Neonur device, the breathing and pressure sensors are respectively pins 7 and 8 on the PIC microcontroller. Two things need to be done to properly configure these two pins. First, they need to be configured as analog pins by setting their respective ANSEL bits. Second, these two pins have to be set as input pins by setting their TRIS bits. *Figure 13* shows the configuration of pins 7 and 8 as analog input pins.



```c
// -----------------------------------------------------------------------
// SENSORS PORTS CONFIG
//------------------------------------------------------------------------

// Pyroelectric sensor  (Pin RC3)
ANSELbits.ANS7 = 1; // Analog ( defaulted analog at Power-on reset)
TRISCbits.TRISC3 = 1; // Input Pin ( defaulted input at Power-on reset)

// Pressure Sensor (Pin RC6)
ANSELHbits.ANS8 = 1; // Analog ( defaulted analog at Power-on Reset )
TRISCbits.TRISC6 = 1; // Input Pin ( defaulted input at Power-on reset )
```

Figure 13: Code snippet: configuring pins, picture by author

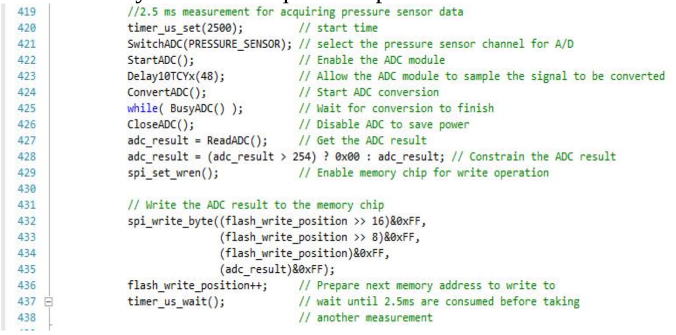
After appropriate configuration of the sensor pins, it is necessary to configure the ADC module. The PIC microcontroller provides 3 registers that control the operations of the ADC module. Those registers are ADCON0, ADCON1 and ADCON2. All the registers on the PIC18F14K50 are 8-bit registers. The ADCON0 register only implements 6 bits starting from bit 0, the rightmost bit, to bit 5 (sixth bit in the register). The first bit in ADCON0 register is the ADON bit which enables the ADC module. Usually this bit is the last bit set before starting data acquisition. The second bit, the GO/DONE bit, has two uses. It is used to determine the status of a conversion and also, when set, it tells the ADC module to start the A/D conversion cycle. Bits 2 to 5 are used to select the analog channel to use with the ADC. In the case of the Neonur, channel 7 (AN7) and channel 8 (AN8) are dynamically selected. ADCON1 and ADCON2 registers control configurations of the voltage references, A/D data format, clock source, and acquisition time. More on the ADC registers can be found in the PIC18F14K50 data sheet. *Figure* 14 below shows a code snippet of the configurations for registers ADCON1 and ADCON2 made for the Neonur.



```c
// -----------------------------------------------------------------------
// ADC - INTERRUPTS - TIMER REGISTERS CONFIGURATIONS
//------------------------------------------------------------------------
ADCON2 = 0x1E;      // ADC Clock = Fosc/64, ADC result
                    // -> Left Justified, Acquisition Time = 6 TAD
ADCON1 = 0x00;      // ADC Voltage References ( +VDD, -VSS)
```

Figure 14: Setting up the ADC module, picture by author

Once the ADC module is properly configured, it can be used to convert analog voltages generated by the pressure and breathing sensors to digital values that are stored in the memory chip. *Figure 15* below shows a sample code snippet that illustrates how the ADC module is used with the external flash memory in a data acquisition process.



```c
//2.5 ms measurement for acquiring pressure sensor data
timer_us_set(2500);         // start time
SwitchADC(PRESSURE_SENSOR); // select the pressure sensor channel for A/D
StartADC();                 // Enable the ADC module
Delay10TCYx(48);            // Allow the ADC module to sample the signal to be converted
ConvertADC();               // Start ADC conversion
while( BusyADC() );         // Wait for conversion to finish
CloseADC();                 // Disable ADC to save power
adc_result = ReadADC();     // Get the ADC result
adc_result = (adc_result > 254) ? 0x00 : adc_result; // Constrain the ADC result
spi_set_wren();             // Enable memory chip for write operation

// Write the ADC result to the memory chip
spi_write_byte((flash_write_position >> 16)&0xFF,
               (flash_write_position >> 8)&0xFF,
               (flash_write_position)&0xFF,
               (adc_result)&0xFF);
flash_write_position++;     // Prepare next memory address to write to
timer_us_wait();            // wait until 2.5ms are consumed before taking
                            // another measurement
```

Figure 15: ADC and timing routines, picture by author

2. Data Storage

After data has been successfully converted by the ADC module, it is stored in the on-board memory chip. To write to the memory chip, three things have to be done in this order: first the memory chip is selected, then the chip is made writable by sending the Write-Enable command, and finally data to store is transferred out. The line dedicated for selecting the memory chip is the pin 6 on the microcontroller. To select the memory chip for exchanges, pin 6 on the PIC microcontroller is pulled low. After pulling the pin low, the write enable command (0x06) is send to the device. Right after sending the write enable command to the memory chip, the chip select line has to be pulled high immediately. This feature is a security feature implemented in the M25P16 and many other serial flash memory units. To start the data transfer process, the memory chip needs to be selected again by pulling the select chip line low, then sending the write command (0x02) followed by the 24-bit address location where data should be written. Following the address is the data byte to write. The select line is pulled high again at the end of the transaction.
Once data transfer complete, the memory chip starts the write process. During this time the chip cannot be accessed for any

write, erase, or read operations. However, the status register can be read to determine the status of the write process.

3. Data Transfer

Usually, after data acquisition, the Neonur module is plugged to a computer for data upload. In previous designs of the Neonur device, data transfers occurred on USART. This work changed the communication interface to use the USB bus. When the Neonur is attached to a PC computer, the PC application recognizes the device and allows the user to communicate with the device by sending instructions. Before USB can work on the device, there are a couple things the device firmware needs to accomplish. First of all, the device firmware needs to successfully go through the enumeration process with the computer. The firmware written for the Neonur is built upon the core USB stack provided by Microchip. As one would soon realize, writing a USB stack from scratch is a very complex and time-consuming task. Most microcontroller manufacturers provide core USB stacks to help developers build firmware faster. There is no change made to the USB core stack. The microchip USB core is usually used with the Microchip Application Libraries as is the case with the Neonur. Details on implementing the USB protocol are provided in the appendix.
Once the PC application requests data from the device, the microcontroller reads the memory chip and transmits the read data in 64-byte packets.

III. RESULTS

The Neonur firmware is now capable of communicating with the PC application without any issues. The PC application is programmed so that it helps the users by guiding them step by step through the data transfer routine. With the implementation of the USB interface, the device can be used with any computer platform supporting USB. Also, no actions are required from the user regarding device setup on the host computer. Using USB, data transfer speed can go up to 480 Mbps. The device firmware implements the USB HID class as specified by the USB specifications, thus allowing the device to function properly without any device firmware changes on major operating systems such as Linux, Mac OS, and Windows.
The device has a minimum power consumption of about 75 mW at idle state and 111 mW at full operation. During USB transfers, power is provided by the host computer, thus saving battery energy. It is also noticed that putting the device to sleep between consecutive measurements improves the overall device power consumption.
The pyroelectric breathing sensor made out polyvinylidene fluoride films generates its own current, thus extending the battery life.

IV. CONCLUSION

The final Neonur product has not been manufactured yet, but a prototyping board in sunder construction. It will be necessary to test the device in a number of different conditions and ensure that the pyroelectric breathing sensor works as expected.
The data uploaded from the device is parsed and saved in a text file in a way that is easy for the end user to copy and paste in plot-generating software packages.
The device is capable of performing a measurement every 5ms. During this time, the device collects and samples the signal on one channel, then collects the signal on the other channel, and is finally put to sleep until a new measurement cycle begins. This technique reduces the overall device's power consumption.
The firmware running on the device allows any computer platform to communicate with the device through a defined set of instructions. 4 basic instructions are provided as an API (Application Programming Interface) for end users to develop their own programs that can exchange with the Neonur device. For Windows programmers, an API in the form of a DLL is provided for both .NET and Win32 programmers.
The USB interface allows end users to use the device without any COM port configurations or initial device setup on their part.

V. ACKNOWLEDGEMENTS

The author of this work, Gedeon Nyengele, would like to thank the following people and organizations for their help and valuable contributions to the success of this work. I acknowledge Dr. Jay Zemel for being a very supportive advisor and for explaining the concepts essential to the success of this work. I also thank all individuals that have worked on the Neonur before for laying down the framework of what today is a better device. I acknowledge Dr. Jan Van der Spiegel and the SUNFEST program for creating this wonderful opportunity and remarkable REU program. I sincerely thank Linda Kalb and Josh Taton for their reminders, follow-ups, and social guidance that made the University of Pennsylvania and the city of Philadelphia a home during this research. I finally thank the National Science Foundation (NSF) for giving undergraduates an opportunity to conduct research.

REFERENCES

[1] http://www.cdc.gov/nchs/data/nvsr/nvsr60/nvsr60_01.pdf

[2] Kron R.E., Stein, M., Goddard, K., (1963), A Method of Measuring Sucking Behaviourof Newborn." *Psychosomatic Medicine, Vol. 29,* pp. 181-191.

[3] "pyroelectricity." *Encyclopædia Britannica. Encyclopædia Britannica Online Academic Edition*. Encyclopædia Britannica Inc., 2013. Web. 01 Aug. 2013.

[4] Kawai, Heiji (1969). "The Piezoelectricity of Poly (vinylidene Fluoride)". *Japanese Journal of Applied Physics* **8** (7): 975. doi:10.1143/JJAP.8.975.