**College of Engineering**

**Electronics and Electrical Engineering Department**

**Mindanao State University – Iligan Institute of Technology**

**Hardware Chron:**

A Hardware Implementation

Of the Cron Scheduling System Software

As a Partial Fulfillment of the Requirements for

**EE 188 - Computer Architecture and Organization**

Required By

**JABIAN, MARVEN E. - EE PROFESSOR**

Proponents:

**ABESTANO, JOHANNAH MAE - BS ECE IV**

**ENANOR, CARYL KEEN - BS ECE IV**

**REGALADO, GIL MICHAEL - BS ECE IV**

**Table of Contents**

Abstract 3

Acknowledgement 4 4

Introduction 5

Related Literature 7

Scope and Limitation 10

Methodology 11

+-Design of Scheduling Method 11

+-Development Environment 12

+--Firmware 12

+--Software 13

+--Hardware 14

+-Hardware Components 15

+--Microcontroller 15

+--Real-time Clock 15 15

+--EEPROM 15

+-Memory Organization 17

+-Algorithm and Process Flow 18

+--Read and Write for AT24C16 EEPROM 18

+--Read and Write time from RTC Clock 19

+--The PC Interface 19

+--General Outline of the Algorithm 20

+--USB Implementation 21

+-Method of Operation 23

+--Parts of the Device 23

+--Parts of the Software 24

+--Schedule Entry Designer 25

+--Device Usage 26

Results and Discussions 27

+-Cost Summary 28

Conclusion and Recommendations 29

**Appendix A**

Simple design of the Scheduler Entry 30

**Appendix B**

Example Partial Chron Entries 31

**Appendix C**

A screen shot taken from a portion of the data sheet specification. 32

**Appendix D**

A page taken from the DS1307 Real Time Clock IC Data Sheet 33

**Appendix E**

A portion of the AT24C16’s Data sheet Specification 34

**Appendix F**

Device Addressing 35

Write Operations 35

Read Operations 36

Device Addressing Figures 37

Byte Write 37

Page Write 37

Current Address Read 38

Random Read 38

Sequential Read 38

**Appendix G**

The Byte Level Read and Write Method in MikroC 39

**Appendix H**

Real-time clock Memory Organization 39

Time Keeper Registers 40

Real-time Clock Device Addressing 41

**Appendix J**

MikroC Code for Reading and Writing Time to DS1307 42

**Abstract**

This study focuses on the implementation of the famous Cron software for scheduling automation in UNIX and Linux systems to actual hardware implementation in scheduling on and off switching of actual hardware devices. The use of the Cron specification is the most unique characteristic of this research as it is the first time it is going to be used for hardware application for purposes of automation.

**Acknowledgement**

The Proponents of this project would like to thank the Heavenly Father for giving us the opportunity to work with this project and giving us the strength, time and knowledge to fulfill our duties. Much gratitude is also due to our instructor Sir Jabian, Marven for giving us time and chances in hopes of completing this project. Finally to our parents who helped fund the early development phase of this project and for always helping us in all matters that need may arise.

Also, our gratitude goes to the open source community at GitHub who have already made significant effort in establishing a few basic technologies that have been the corner stone of some of the components of our project. Special thanks to mikeobrien of Github for opensourcing his work on Hid Library for C# on Windows. Also special thanks to hanshuebner of Github for opensourcing his work in Hid Library for NodeJS. Thank you to everyone whose initial work and effort has been very instrumental to the completion of this project.

**Introduction**

Automation has always been a very important thing in a wide array of industries. Automation reduces labor cost, increases efficiency and increases overall revenue for certain companies. Depending on the type and cost of operation, automation systems will vary in performance and reliability.

Several automation systems are also implemented for security, safety, efficient energy usage and home use. Some of these may include closing doors during midnight, and opening them during the day, and some may even include opening valves in water stations at predetermined time, or turning on home lights at exactly 6PM in the evening and turning them off during the morning to save energy.

There are so many more applications for automation today and even though they vary, some of the most common of them involve some sort of scheduling. They are able to tell time and know when to turn on and off other devices, and thus come the inspiration for this project. In this project, a simple implementation of an automation system based on a predetermined schedule is designed and developed using the most common technologies such that not only will it be feasible for industrial application but also for home and personal use.

Much research has been done in order to determine the available technologies that can be used in order to efficiently design the device. However, a large portion of them are designed for highly industrialized purposes and the costs are certainly very prohibitive for regular home or personal use. That is why this project is undertaken to create a very feasible device and complementary software for automation.

The device will have a microcontroller, a real time clock and an electrically erasable and programmable read only memory. In order to increase the ease of use and simplicity of design the device will be powered using a regular USB socket for ease of use.USB Sockets are very widespread nowadays, almost every home that will have a computer will always have USB sockets.

Schedule entries will then be designed by the user using the associated software application which will also be developed. The desktop client software will also be responsible for writing these schedule information to the device memory in order for the devices to function without the aid of the computer once all data is stored in its own memory.

After the schedule entries are input to the device, it will then continually check for schedule entries and compare them with data from the real time clock and once a certain entry is scheduled to run for the time, it will then turn off or on a certain device as specified. This will be done in one minute resolutions allowing the device to have schedules that are accurate to one minute.

Despite the fact device schedulers already exist, this research is undertaken using new methodologies for defining schedules. The unique use of the Cron syntax which is the defacto standard for scheduling in Linux is perhaps the most unique characteristic of this study. This allows us to put complex Cron statements that will allow complex schedule entries to be added with the minimal amount of coding required. Part of this project is modifying and improving the Cron syntax for hardware implementation.

In general this project is all about implementing the famous Cron Scheduling system for software applications in to hardware applications. And because this project is applied for hardware the system is named ‘Chron’ with an embedded letter ‘h’ between ‘C’ and ‘r’ indicating it is for hardware applications. The researchers believe that this has never been done before and it carries a large potential for automation systems of varying use cases.

**Related Literature**

Cron is the time-based job scheduler in Unix-like computer operating systems. Cron enables users to schedule jobs (commands or shell scripts) to run periodically at certain times or dates. It is commonly used to automate system maintenance or administration, though its general-purpose nature means that it can be used for other purposes, such as connecting to the Internet and downloading email.

The Cron standard is used as basis for scheduling data for this project. The widespread use and adaption of Cron in the software industry will increase the usage familiarity of our device for different applications. It is thus very new that the Cron standard is adapted for hardware scheduling as is the main focus of this project.

Universal Serial Bus (USB) is an industry standard developed in the mid-1990s that defines the cables, connectors and communications protocols used in a bus for connection, communication and power supply between computers and electronic devices.

USB was designed to standardize the connection of computer peripherals, such as keyboards, pointing devices, digital cameras, printers, portable media players, disk drives and network adapters to personal computers, both to communicate and to supply electric power. It has become commonplace on other devices, such as smart phones, PDAs and video game consoles. USB has effectively replaced a variety of earlier interfaces, such as serial and parallel ports, as well as separate power chargers for portable devices.

USB human interface device class (USB HID class) is a part of the USB specification for computer peripherals: it specifies a device class (a type of computer hardware) for human interface devices such as keyboards, mice, game controllers and alphanumeric display devices. This specification will be utilized by the project in order to facilitate low level interfacing with the PC and the device.

As of 2008, approximately 6 billion USB ports and interfaces were in the global marketplace, and about 2 billion were being sold each year. Given the widespread use and adaption of the USB standard, it will add improvement to our device if we use the USB as both the primary power source and interface bus with the PC.

I²C generically referred to as "two-wire interface is a multi-master serial single-ended computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, cell phone, or other electronic device. Since the mid 1990s, several competitors (e.g., Siemens AG (later Infineon Technologies AG, now Intel mobile communications), NEC, Texas Instruments, STMicroelectronics (formerly SGS-Thomson), Motorola (later Freescale), Intersil, etc.) brought I²C products on the market, which are fully compatible with the NXP (formerly Philips's semiconductor division) I²C-system. The widespread use of the I²C interface allows for the project to be easily integrated with each subcomponent within it.

A microcontroller (sometimes abbreviated µC, uC or MCU) is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other embedded systems. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes. Mixed signal microcontrollers are common, integrating analog components needed to control non-digital electronic systems.

PIC is a family of modified Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1650 originally developed by General Instrument's Microelectronics Division. The name PIC initially referred to "Peripheral Interface Controller". The PIC18F4550 is the Microcontroller to be used for the projects implementation of the device.

EEPROM is user-modifiable read-only memory (ROM) that can be erased and reprogrammed (written to) repeatedly through the application of higher than normal electrical voltage generated externally or internally in the case of modern EEPROMs. EPROM usually must be removed from the device for erasing and programming, whereas EEPROMs can be programmed and erased in-circuit. Originally, EEPROMs were limited to single byte operations which made them slower, but modern EEPROMs allow multi-byte page operations.

EEPROMs also have a limited life - that is, the number of times it could be reprogrammed was limited to tens or hundreds of thousands of times. That limitation has been extended to a million write operations in modern EEPROMs. In an EEPROM that is frequently reprogrammed while the computer is in use, the life of the EEPROM can be an important design consideration. It is for this reason that EEPROMs were used for configuration information, rather than random access memory. The Atmel AT24C16 IC is used for the implementation of the device.

A real-time clock (RTC) is a computer clock (most often in the form of an integrated circuit) that keeps track of the current time. Although the term often refers to the devices in personal computers, servers and embedded systems, RTCs are present in almost any electronic device which needs to keep accurate time. The DS1307 real time clock by Maxim is used for the implementation of the projects device.

Although keeping time can be done without an RTC, using one has benefits. Low power consumption, (important when running from alternate power); Frees the main system for time-critical tasks; and finally it is sometimes more accurate than other methods.

**Scope and Limitation**

However, due to time, and cost limitations, the project will be very limited in terms of scale and durability of design. The primary focus of the project will be on the firmware programming, software programming and hardware integration for a low cost and simple device used for scheduling and automation.

Due to time limitations only a prototype device and associated software applications will be developed, and the design for an actual device for consumer or industrial use is beyond the scope of the project. Due to the time taken towards software and firmware development, the researchers no longer have enough time to develop a suitable PCB for the circuit hence universal boards were used for quick development of the hardware.

The primary focus of this project is to demonstrate the feasibility of implementing the Cron scheduling mechanism for application of scheduling other hardware devices. And to demonstrate a prototype device that does this in the very basic level.

**Methodology**

**Design of Scheduling Method**

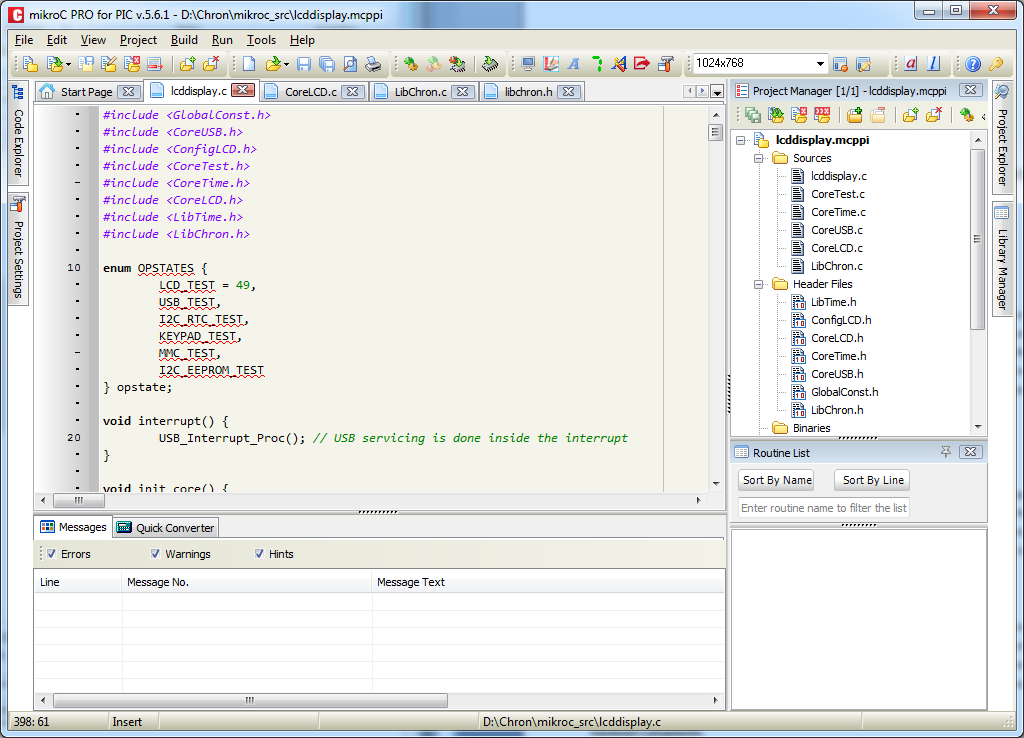
In order to automate processes, one must first establish a scheduling standard. The standard used for this project will be the existing standard used by the Cron Software. Only a fraction of the actual Cron specification will be taken in to account for this project due to inherent limitations when using only a microcontroller with limited memory and processing power. A 21 byte specification for a schedule entry is detailed on Appendix A.

The classifier can either be a NULL or a DASH. If it is a NULL the Lower Value will be considered as the defined unit for the schedule. If the lower limit is an ASTERISK, the software can evaluate it as being always valid. If the Lower Limit is a value then it can only become valid when the current time value is equal to the lower limit value. If the classifier is a DASH, then the entry will be evaluated considering the upper and lower limits. Refer to Appendix B for example Chron Entries and their explanations.

**Development Environments**

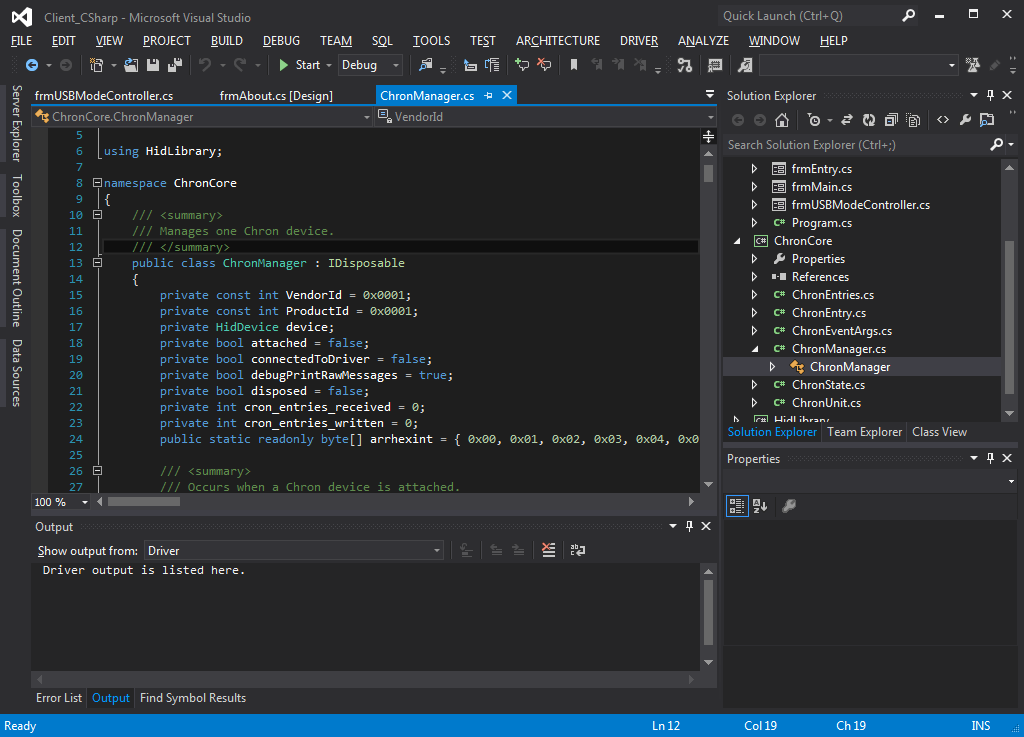
**Firmware**

Many development environments for the Firmware were considered for this project. However, for increased time to completion and simplicity of development the MikroC development environment for the PIC is utilized. Advantages for using MikroC are the well organized documentation and the wide array of Hardware and Software Libraries that can be easily utilized for use for the project. The extensive support of MikroC for PIC devices is perhaps the best strong point for choosing it as the major development platform for the Firmware.



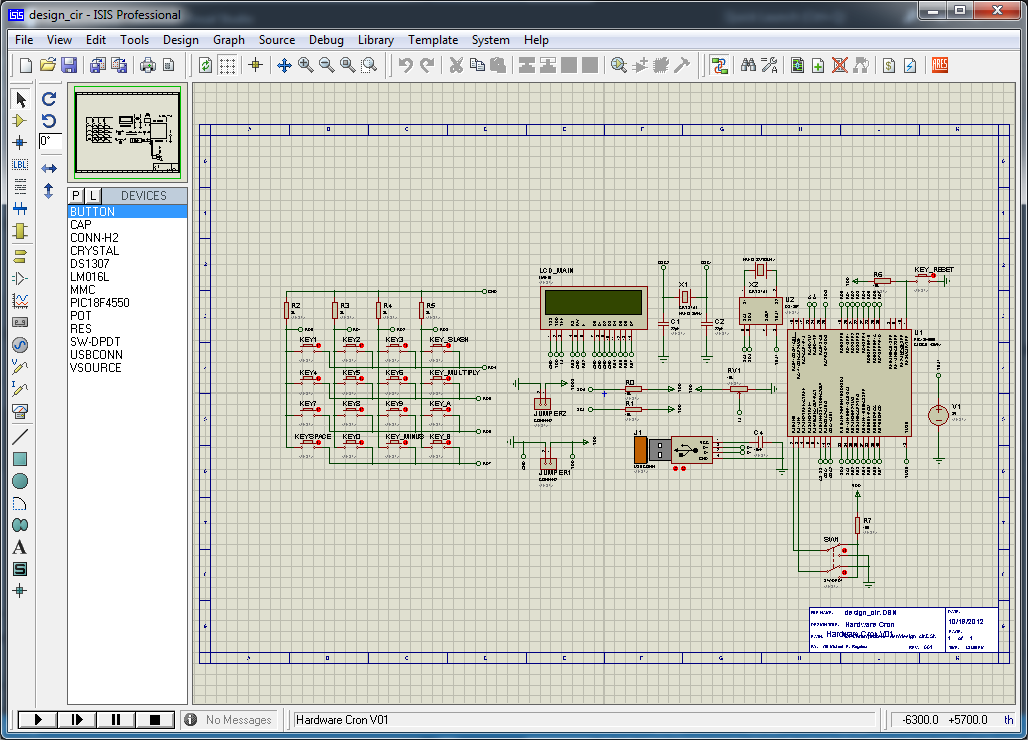
**Software**

In order to interface the device with the PC, suitable Software has to be developed to facilitate PIC and PC communications. The researchers decided to use Visual Studio C# for Windows as the Development Environment to support such purpose. Using the Visual Studio C# development environment will significantly increase development time and decrease debugging difficulty due to its fully featured debugging environment.



**Hardware**

In order to improve hardware development methods suitable simulation software for hardware components is utilized. The Proteus VSM suite is used for simulating the PIC and its associated hardware components. Also, the Proteus has support for simulating USB communications with the PC. However this only supported Proteus – Windows XP environment. So a Virtual Machine installation of Windows XP using Virtual Box was setup to facilitate this simulation.



**Hardware Components**

The ability to load schedule data from the user’s computer to the device is very important for this project. Therefore one of the most important determining factors for the selection of components is the capability to interface with the computer. For this study, the USB interface was selected as the most common place and easiest to use interface that will add the benefit of simplicity of use for the product.

**Microcontroller**

The PIC18F4550 from Microchip was selected as the core Microcontroller for this project due to the fact that it has an internal USB Module that can be easily used to interface with the Personal Computer. Some very important details of the PIC18F4550 are listed on Appendix C.

**Real-time Clock**

The device has to be able to remember time regardless of its main power source present or not, and even if the Microcontroller is busy with other operations, it will not lag and fail to keep up with remembering time. In order to do this an external Integrated Circuit is used for remembering time. The real time clock DS1307 is used for this purpose. A portion of DS1307’s data sheet is posted on Appendix D.

One of the major reasons for selecting this device was because of compatibility. It is very important that this device can be easily integrated with the main PIC18F4550 microcontroller. The interface type from IC to IC within the device must be feasible enough. This compatibility is provided by the I2C interface capability of the Real-time Clock and the PIC18F4550’s own I2C controller module and because this, the main PIC can handle I2C connections, and the DS1307 can easily be interfaced with it.0

**EEPROM**

In order for the device to keep a significant number of schedules based on the initial schedule entry design, even if the device is turned off, the design must include the use of an EEPROM. The EEPROM will allow the device to remember schedule entries even if it was turned off for a short while. The PIC18F4550 has an EEPROM size of 256 bytes as detailed on the specification on Appendix C. However a simple calculation will tell us that 256 divided by 21 will only allow us to store 12 schedule entries.

If the device is capable of scheduling 8 switches, then there will almost likely be only 1 or 2 schedule entries for each switch. This is very limited and therefore not feasible. In order to improve the operation of the device, an external EEPROM of reasonable capacity must be selected for the purpose of storing schedule entries. The block diagram below shows the general overview of the device components.

The Atmel AT24C16 was selected for this purpose. It has 2048 bytes of storage capacity distributed in 8 internal pages. The added convenience is that the AT24C16 supports I2C communications and therefore it is easier to interface it with the PIC18F4550. A portion of AT24C16’s data sheet is posted on Appendix F.

**Block Diagram of the Hardware Components**

I2C BUS

USB

RTC  
DS1307

EEPROM  
AT24C16

MCU  
PIC18F4550

Optional LCD Display

USB Power Supply

Computer Interface

8 Bit Switches

**Memory Organization**

**The following figure shows the AT24C16 memory organization.**

Byte 1

Byte 2

Byte 3

…

Byte 255

Byte 0

Page 1

Byte 1

Byte 2

Byte 3

…

Byte 255

Byte 0

Page 2

Byte 1

Byte 2

Byte 3

…

Byte 255

Byte 0

Page 0

Byte 1

Byte 2

Byte 3

…

Byte 255

Byte 0

Page …

Byte 1

Byte 2

Byte 3

…

Byte 255

Byte 0

Page 7

Now that the external memory IC was determined a simple Memory Structure is designed in order to handle the write and read of Schedule Entries to and from the AT24C16 EEPROM.

According to the data sheet the AT24C16 has a 3 byte page addressing which translates to 8 pages. Each page then contains an 8 bit address space for 256 separate addresses. Each address then finally contains 1 byte or 8 bits of data. With 8 pages, and 256 bytes per page, this translates to 2048 bytes of data. For a complete overview of AT24C16’s device addressing scheme, refer to the first Page of Appendix F.

However, due to the complicated addressing scheme caused by the pages the researcher’s decided to put only the number of 21 byte schedule entries that can fit in to a single page. Since there 256 bytes per page, then there can be 12 schedule entries per page. Finally, since there 8 pages, then the device can store 96 schedule entries in the EEPROM.

This simplifies the access and store mechanism considerable for the purpose of this research. Though it should be noted that this is not the most efficient way of doing this and dealing with storage efficiency for this matter is beyond the scope of the study.

Despite all these the device must still know how many schedule entries are written sequentially on itself. This is where the PIC18F4550’s internal EEPROM is used. The first byte of the PIC’s internal EEPROM is used for storing the number of entries stored on the External EEPROM. This way, the program can know ahead of time how many entries it has to read.

**Abstraction Memory Organization for the Device (12 Entries are distributed to 256 Bytes)**

Entry 1

Entry 2

Entry 3

…

Entry 11

Entry 0

Page 1

Entry 1

Entry 2

Entry 3

…

Entry 11

Entry 0

Page 2

Entry 1

Entry 2

Entry 3

…

Entry 11

Entry 0

Page 0

Entry 1

Entry 2

Entry 3

…

Entry 11

Entry 0

Page …

Entry 1

Entry 2

Entry 3

…

Entry 11

Entry 0

Page 7

Among others, all the associated minor components that are required to run the major components are also acquired and an LCD Display is utilized in order to streamline the process of debugging the algorithms which are detailed on the next section. The LCD may be removed in order to produce a product of smaller size foot print and decrease power consumption.

**Algorithm and Process Flow**

Now that the important components are setup the algorithms that will make them work together in harmony are then established and coded in to MikroC, compile to a Hex File then loaded for testing. The details of the algorithms will be discussed in this section.

**Read and Write for AT24C16 EEPROM**

Using the memory organization as discussed from the previous section. A suitable write operation sequence is developed in order to allow the PIC18F4550 to write to the EEPROM and be able to store data. On Appendix F, the write operations are described by pages taken from the datasheet.

For the 16k version of the chip which is being used for this project, the algorithm will divide entries in to 8 pages. Then the algorithm will put only the amount of entries which can fit per page. Since each page can fit 256 bytes, then only 12 entries of 21 bytes are stored each page. Simple loops are used for this purpose.

However, for the actual writing, each byte is written individually on predetermined locations. This method is represented on Figure 8 of Appendix F. A compatible code is developed for this as presented on Appendix G.

Also the Random Read is utilized as discussed on the first Page of Appendix F, and a corresponding equivalent code for MikroC as continued on Appendix G. This shows the basic EEPROM read and write operations to be used for the Program.

A simple pseudo code follows detailing the reading of schedule entries.

|  |
| --- |
| total = get\_total\_number\_of\_entries\_from\_internal\_eeprom;  entry[21];  entries\_read = 0;  entry\_per\_page = 12;  bytes\_per\_entry = 21;  total\_pages = 8;  for (page=0; page<total\_pages; page++) {  if (entries\_read < total) {    address\_within\_page = 0;    // 12 entries per page  for (entries=0; entries<entry\_per\_page; entries++) {    // 21 bytes per entry  for (entry\_address=0; entry\_address<bytes\_per\_entry; entry\_address++) {  entry[entry\_address] = read\_entry(page, address\_within\_page);  address\_within\_page++;  }  }  }  } |

**Reading and Writing time from the Real-time Clock**

Same with the EEPROM, the PIC must now read and write time to the DS1307 Real-time Clock. On Appendix H and I are the Memory Organization and Device addressing schemes used by the IC. A compatible code is then described in Appendix J.

**The PC Interface**

Though methods for USB Hid development with PIC18F4550 are well supported by the MikroC Development Environment, methods for establishing USB HID interfaces for the PC are widely varied depending on purpose and implementation. An open source HID Application Programming Interface created by a Github user *mikeobrien* is used for this purpose. The Hid Library he developed as an open source project will be used to interface the PC to the Device.

The decision for this move is also inspired by the use of the C# Development Platform for Window using Visual Studio 2012 as the development environment for the PC. Hence for this project the associated software components can only be run on the Windows PC. However, the use of NodeJS and the Node HID API is also recommended for Real-time Web Interfacing for NodeJS Supported platforms which include the MAC, PC and Linux.

**General Outline of the Algorithm**

Now that the Core functionalities have already been described we will then discuss the general outline of the algorithm.

The user can use a toggle switch to choose between modes. The USB mode allows the device to connect to the computer. The timer mode will read the schedule entries in a loop and check each time if the schedule entry is valid for the current time. If it is valid, then it will implement this entry and switch on or off a certain device. Below is the flowchart for the startup sequence of the device.



Once the user has selected a mode, it will proceed to looping for requests and processing them along the way. Below is the flow chart for this sequence.

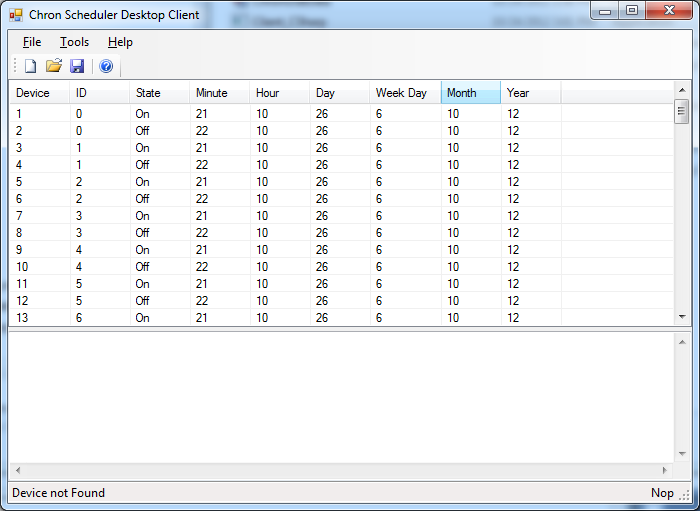


**USB Implementation**

In order to work with the PC interface the USB must be implemented in both the PC and the PIC. Though the USB module for the PIC is very well defined on the data sheet, USB interfacing with the PC is still quite a challenging task due to the fact that the PC is a much more complicated and proprietary system. In this study, the use of the USB human interface device class (USB HID class), a part of the USB specification for computer peripherals for human interface devices such as keyboards, mice, game controllers and alphanumeric display devices is used for interfacing this device to the PC.

Help and Support resources based from the MikroC was used in order to implement the USB HID specification on the firmware level. For the Host PC part, after research on the net, an open source project written by *mikeobrien*. His USB HidLibrary includes examples for use for other hardware which were then modified for use for this project.

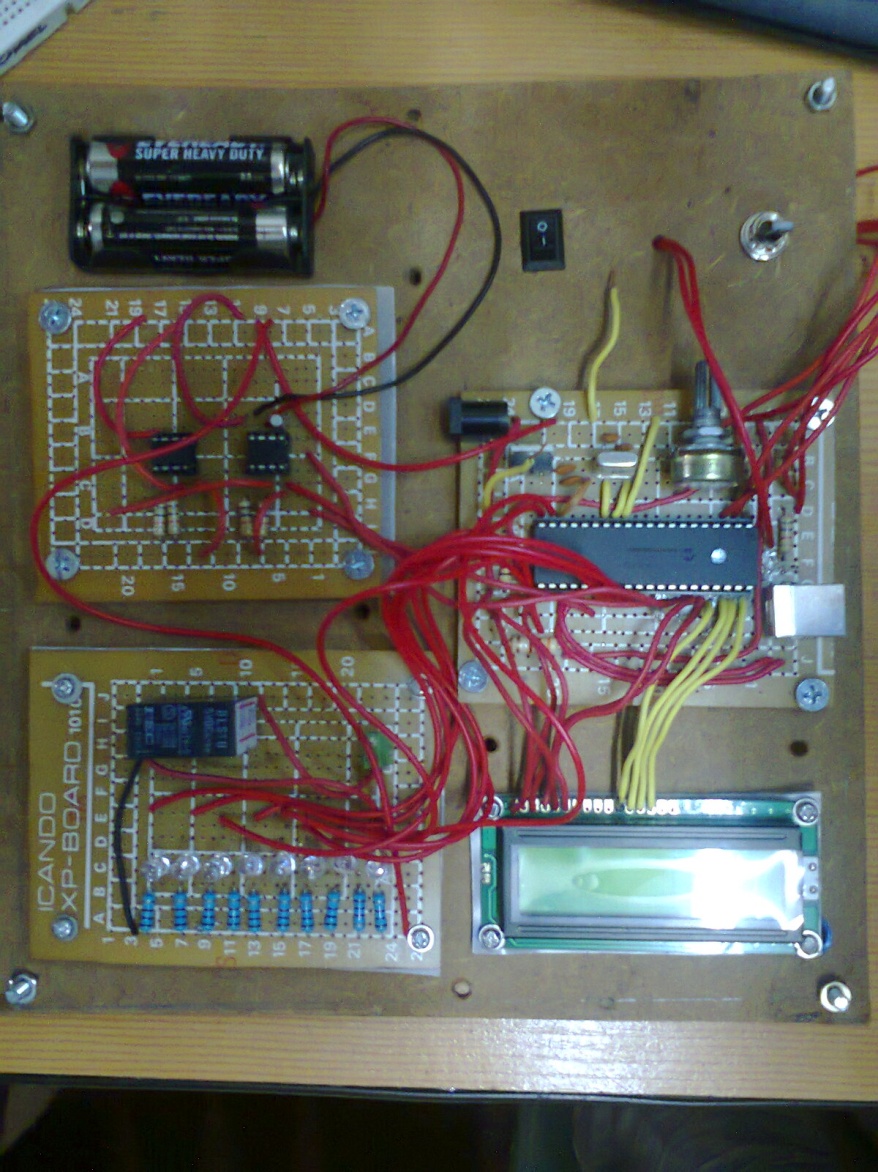
A Software interface was designed to conduct the input of schedule entries and manual overrides for the device during testing and debugging. The picture below is a screen shot of the user interface created for this purpose.



Each of the eight available ports for device switching can individually be controlled by the user interface. The schedule entries will then be sent to the device when it is connected and the Write signal is started. The device can then be switched mode to run independently from the PC and it will begin looping all schedule entries and comparing them with the current time. If a certain switch in the port is slated for implementation for such time then it will be triggered.

**Method of Operation**

**Parts of the Device**



**G**

**F**

**A**

**K**

**J**

**E**

**D**

**C**

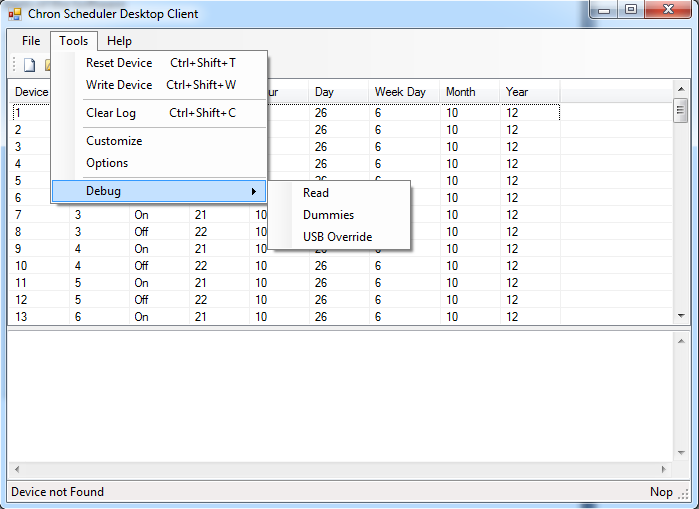
**B**

**I**

**H**

1. The 3V AA battery series which provides power to the DS1307 Real-time Clock when the primary USB power source is out.
2. The EEPROM and RTC module.
3. The switching and monitoring Module. The eight LED Devices represent one switching device. Only the first LED which is Device 0, has a 5V DC 220V AC relay.
4. LCD Display module. Can be removed on the final product. Useful for debugging the device during development phase.
5. The Main Board. Contains the PIC and the USB port which feeds power and interfaces to the PC.
6. Power switch. The O symbol is open and the ‘I’ symbol is closed.
7. The toggle switch for mode switching. Toggles between USB and time Modes.
8. The Atmel 119 24C16 EEPROM.
9. DS1307 Real-time Clock.

**Parts of the Software**



**Data Log**

**Schedule Entries**

**E**

**F**

**D**

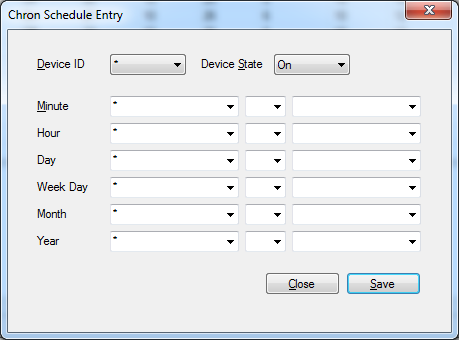
**A**

**B**

**C**

1. Allows the user to reset the device. It will sync PC time and device time. It will also set the number of entries to 0. It should be noted that since the number of entries are recorded on the PIC18F4550’s EEPROM in the first byte, this operation only 0’s the first byte of the PIC18F45550’s internal EEPROM.
2. Once you are done putting in the schedule entries, you can then choose this option to write the schedule entries to the device EEPROM.
3. Clears the Data log.
4. Reads all EEPROM entries regardless of how many entries were written before. Useful for debugging.
5. Writes 95 schedule entries which can be readily written to the device itself. This is for testing the maximum number of entries to be written.
6. Allows for controlling the individual switches in real-time.

**Schedule Entry Designer**



* The asterisk (‘\*’) will indicate an ‘all’ directive. Refer to the Appendix on the discussion of Cron scheduling for in-depth review on the matter.
* Device state allows you to turn on or off a certain device on the specified time.
* Device ID allows you to choose which device are turned on or off at the specified time. There are 8 devices available for control on this version of the hardware.

**Device Usage**

1. Prepare a copy of the Hardware Chron Desktop Client Software for Windows.
2. Open the program and start designing your preferred schedule. You may save and open predefined schedule as desired.
3. Make sure the device is in the off state.
4. Make sure to set the device toggle switch to USB mode.
5. Connect it to the USB port of your PC.
6. Turn the device on. It should begin initializing the device for USB connectivity.
7. Wait for the software’s status bar to say connected before proceeding.
8. On the software, go to Tools Menu and click Write. It will begin writing the schedule entries to your device. Make sure not to remove the device from the USB port while the schedule entries are being written.
9. After the schedule entries have been written. Turn off the device.
10. Toggle the device switch to ‘Time Mode’.
11. Turn on the device.
12. This will start the device as time mode and will be independent of the USB interface with the PC. It only requires USB connectivity to get clean 5V power from the PC.
13. Once a schedule entry is slated for activation the Device will automatically implement the required state for a certain switch.

**Results and Discussions**

The device has been successfully developed in a prototype stage. All firmware and software components were also completed. However it should be noted that the Cron specification for the device was heavily modified. Also, not all of Cron’s features are applicable to the device due to its inherent limitations. To end, the device is working properly and has limitless potential for use on different applications.

**Cost Summary**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Description | Price | Items Acquired | Sub Total | Items Required | Sub Total |
| PIC18F4550 | Flash32K 2KRAM EEPROM USB DIP40 | 470.00 | 4 | 1880.00 | 1 | 470.00 |
| DS1307 | 64 x 8, Serial, I?C Real-Time Clock DIP8 | 45.00 | 4 | 180.00 | 1 | 45.00 |
| 8.000MHz | crystal | 20.00 | 2 | 40.00 | 1 | 20.00 |
| 32.768KHZ | 32.768KHZ crystal | 20.00 | 2 | 40.00 | 1 | 20.00 |
| 22pf | cap 22pf | 0.50 | 4 | 2.00 | 2 | 1.00 |
| TX2-5V | NAIS 5V DPDT | 33.50 | 10 | 335.00 | 0 | 0.00 |
| USB type B | USB type B | 16.50 | 2 | 33.00 | 1 | 16.50 |
| 24C16 | 16K bit Serial EEPROM DIP8 | 42.00 | 2 | 84.00 | 1 | 42.00 |
| Capacitor 103 / 0.01uf | Ceramic Cap 103 / 0.01uf | 0.50 | 10 | 5.00 | 2 | 1.00 |
| 100nF (104) | 100nF/104 Ceramic Cap. | 0.50 | 10 | 5.00 | 2 | 1.00 |
| 1WT Resistor | resistor 1wt | 1.00 | 20 | 20.00 | 20 | 20.00 |
| 2x16  LCD | lcd w/backlight  2x16 (new) | 350.00 | 2 | 700.00 | 1 | 350.00 |
| Assembly | PCB, Etching and other Manufacturing Services per Device | 500.00 | 0 | 0.00 | 1 | 500.00 |
|  |  |  |  |  |  |  |
|  | Total Cost of Development |  |  | 3324.00 |  |  |
|  | Total Cost of Product |  |  |  |  | 1486.50 |
|  | Total Cost of Product w/o LCD |  |  |  |  | 1136.50 |

It should be obvious by now that significant cost reduction is made from actual device and associated software development is done and more especially when the final product is made without the LCD.

**Conclusion and Recommendations**

The researches have demonstrated that indeed it is feasible to implement the Cron specification in scheduling hardware devices. However, the cost of the project has indeed also increased with complexity. It is therefore recommended that optimization be made in order to increase capacity and lower cost of the device.

The EEPROM memory allocation can be improved by setting up better memory allocation strategies. Because the researchers only took care of completing the project within time allotted, there was no optimization made in the memory allocation of the system.

Also, the Read and Write algorithms can be improved after optimizing the memory allocations strategy. Developments to use Page Write and Read operation can be implemented in order to increase the speed of reading and writing. This was not included in this project due to the added complexity that may arise if this optimization path was taken. Relevant data for implementing this would be available on the EEPROMs data sheet as posted on the Appendix.

Improved schedule entry and parsing can be implemented in order to reduce read times and increase the number of schedules read per minute. And in fact the resolution can be increased to seconds depending on the speed of the microcontroller used.

The LCD display can also be removed to save space as it is only helpful during the debugging phase. Also, in this device, only one switch has a relay attached to it. A separate module of Relay Arrays can be used and a specialized connection system can be implemented to connect the Relay Array with the device can be developed.

On the software side, the HID driver can be implemented in to multiplatform applications using the Multi Paltform GNU Hid Library. Also, the device can be interfaced to the Internet using the NodeJS HID library.

This project, all relevant data, source code and design files has been opensourced by its researchers and is posted for public reference and use at <https://github.com/bangonkali/Chron> in hopes of improving this project for actual use and application.

**Appendix A**

Simple design of the Scheduler Entry

|  |  |  |
| --- | --- | --- |
| Byte | Name | Description |
| 0 | Enable | This will be 0 when disable and 1 when enabled. |
| 1 | DeviceID | Device that will be turned on or off |
| 2 | DeviceState | On or Off |
| 3 | MinutesLower | The lower limit for the minute |
| 4 | MinutesClassifier | Classifier for the range of the minute |
| 5 | MinutesUpper | The upper limit for the minute |
| 6 | HourLower | The lower limit for the hour |
| 7 | HourClassifier | Classifier for the range of the hour |
| 8 | HourUpper | The upper limit for the hour |
| 9 | MonthDayLower | The lower limit for the day of month |
| 10 | MonthDayClassifier | Classifier for the range of the day of month |
| 11 | MonthDayUpper | The upper limit for the day of month |
| 12 | WeekdayLower | The lower limit for the weekday |
| 13 | WeekdayClassifier | Classifier for the range of the weekday |
| 14 | WeekdayUpper | The upper limit for the weekday |
| 15 | MonthLower | The lower limit for the month |
| 16 | MonthClassifier | Classifier for the range of the month |
| 17 | MonthUpper | The upper limit for the month |
| 18 | YearLower | The lower limit for the year |
| 19 | YearClassifier | Classifier for the range of the year |
| 20 | YearUpper | The upper limit for the year |

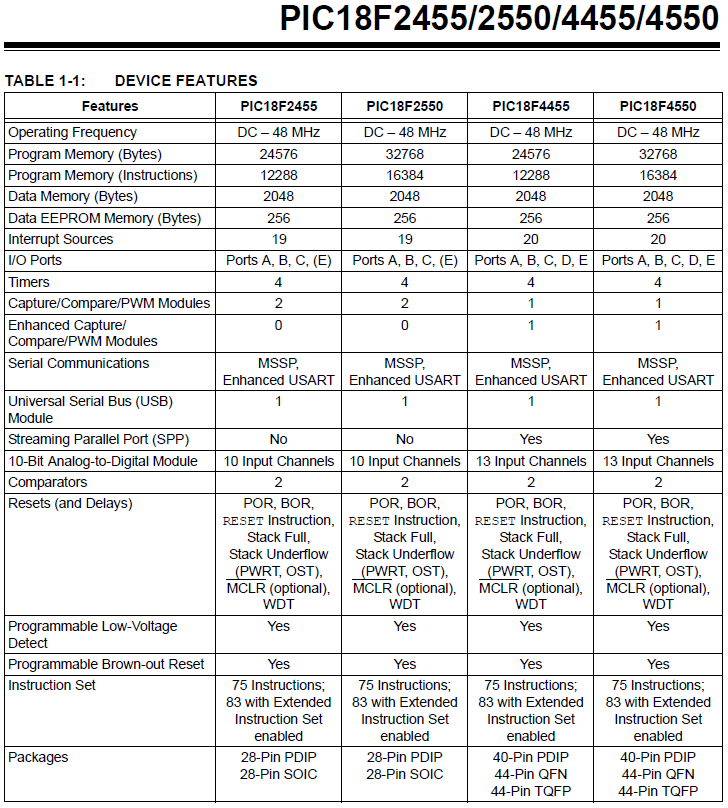
**Appendix B**

Example Partial Chron Entries

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| MM | HH | MD | WD | MO | YY | Description |
| \* | 6 | \* | \* | \* | \* | Activate every 6:00 AM every day. |
| 30 | 7 | \* | \* | \* | \* | Activate every 7:30 AM every day. |
| 15 | 12 | \* | 2-6 | \* | \* | Activate every 12:15 PM every Monday to Friday. |
| \* | 24 | 1 | \* | 1-3 | 12-13 | Activate every 12:00 AM every first day of the Month only during January, February, and March for Year 2012 to 2013. |
| \* | 20 | 1-15 | 4 | \* | 12 | Activate every 8:00 PM, only from 1st to 15th day of the month were it is Wednesday and the year is 2012. |

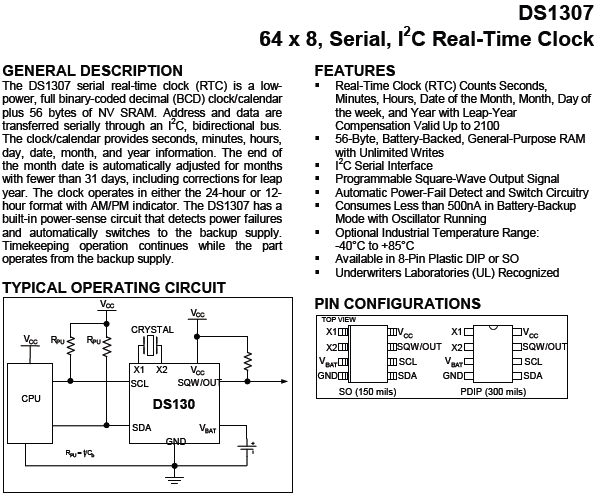
**Appendix C**

A screen shot taken from a portion of the data sheet specification.



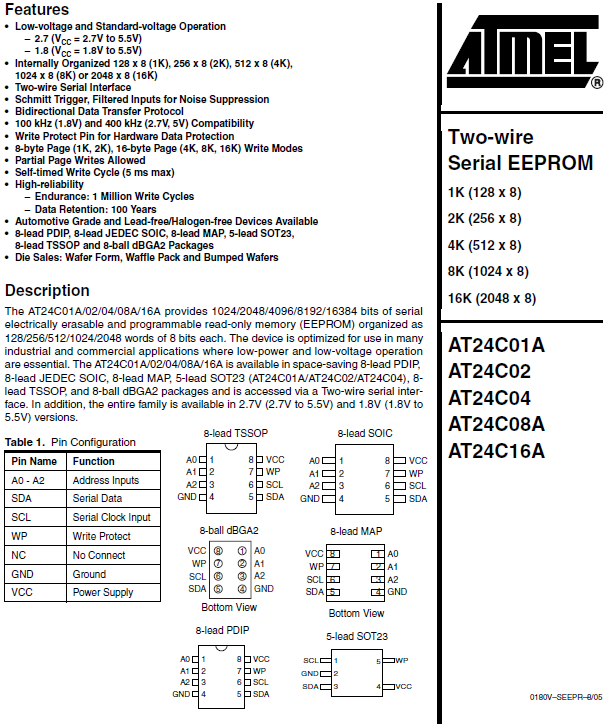
**Appendix D**

A page taken from the DS1307 Real Time Clock IC Data Sheet



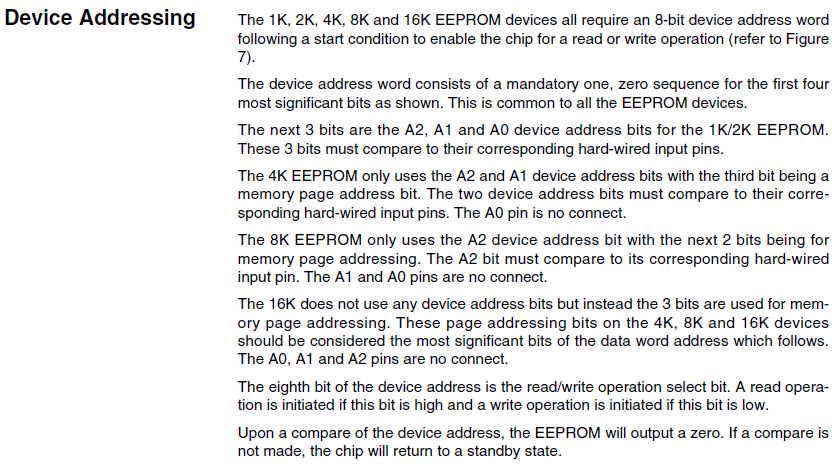
**Appendix E**

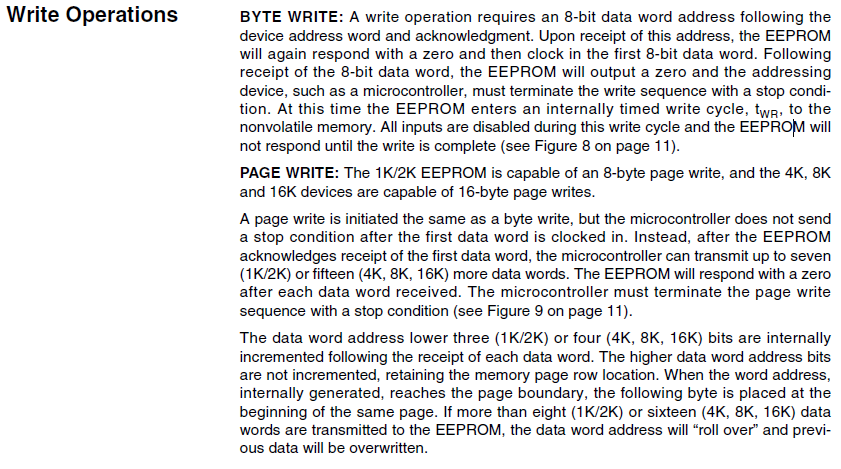
A portion of the AT24C16’s Data sheet Specification

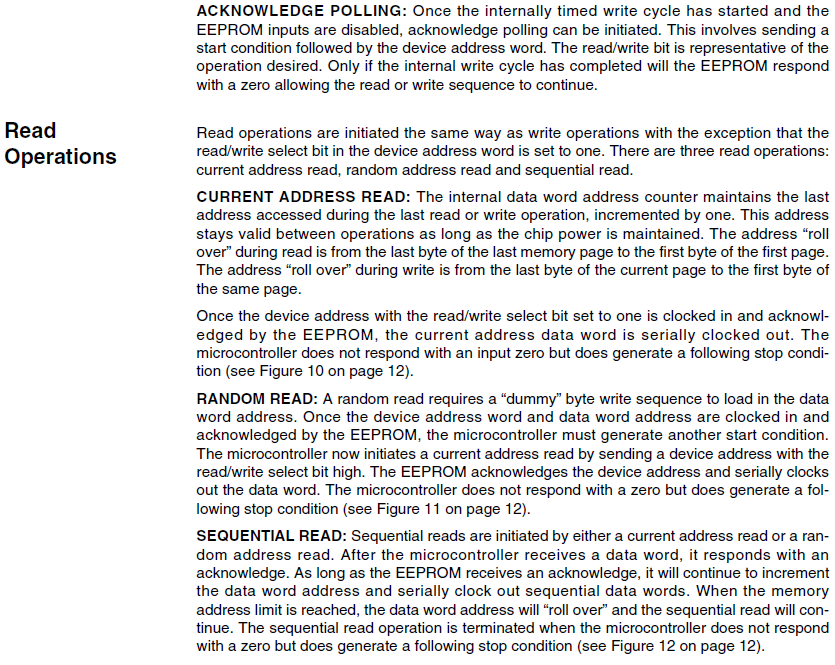


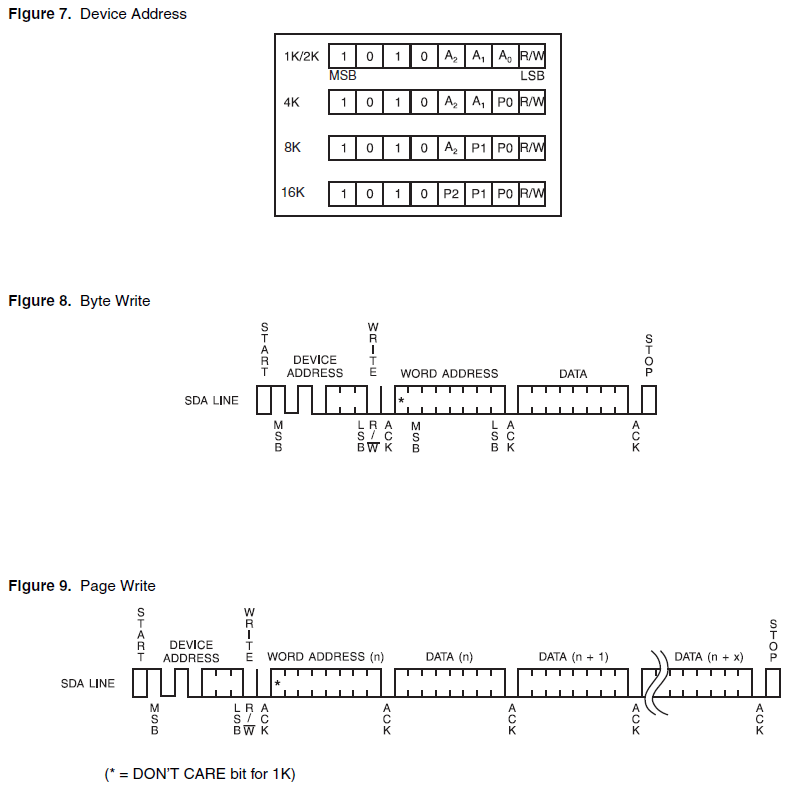
**Appendix F**

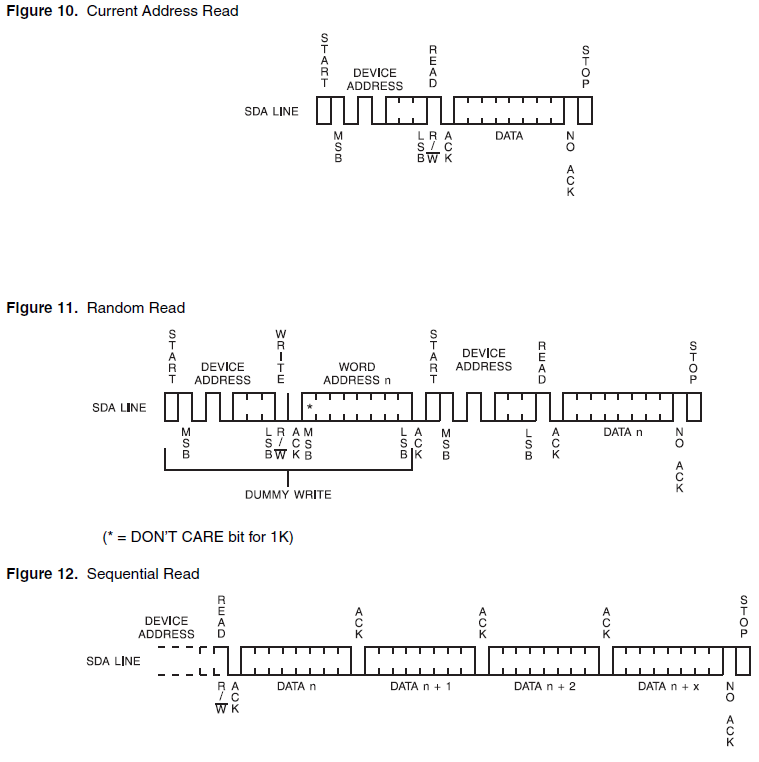
Device Addressing Scheme for AT24C16











**Appendix G**

**The Byte Level Write Method in MikroC**

unsigned char I2C\_Read\_Byte\_From\_EEPROM(unsigned char page\_write, unsigned char page\_read, unsigned char address) {

unsigned char x;

I2C1\_Init(100000);

I2C1\_Start(); // issue I2C start signal

I2C1\_Wr(page\_write); // send byte via I2C (device address + W)

I2C1\_Wr(address); // send byte (data address)

I2C1\_Repeated\_Start(); // issue I2C signal repeated start

I2C1\_Wr(page\_read); // send byte (device address + R)

x = I2C1\_Rd(0); // Read the data (NO acknowledge)

I2C1\_Stop(); // issue I2C stop signal

return x;

Delay\_ms(20);

}

**The Byte Level Read Method in MikroC**

void I2C\_Write\_Byte\_To\_EEPROM(unsigned char page\_write, unsigned char address, unsigned char byte2write) {

I2C1\_Init(100000); // initialize I2C communication

I2C1\_Start(); // issue I2C start signal

I2C1\_Wr(page\_write); // send byte via I2C (device address + W)

I2C1\_Wr(address); // send byte (address of EEPROM location)

I2C1\_Wr(byte2write); // send data (data to be written)

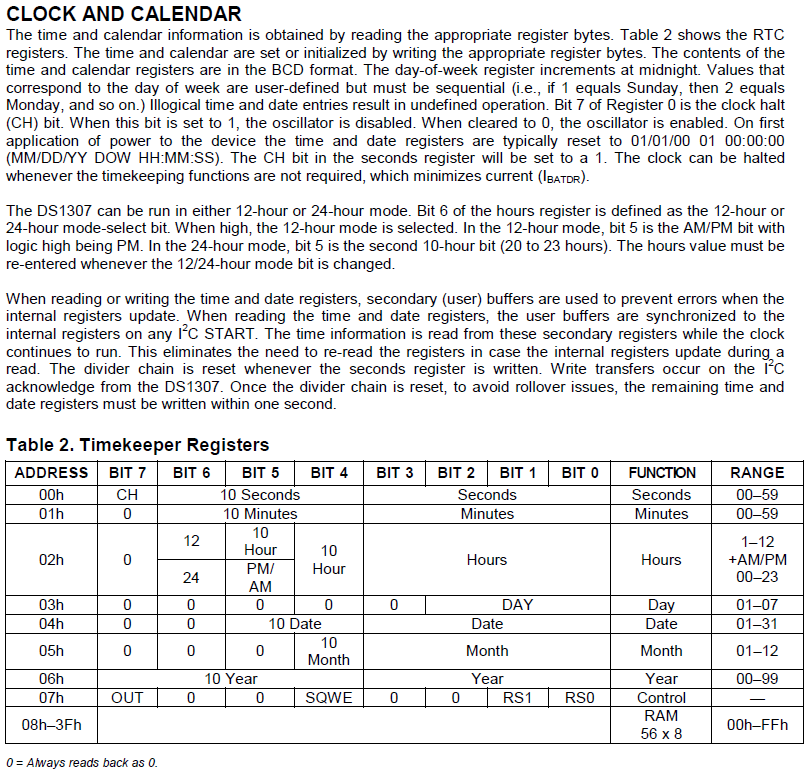
I2C1\_Stop(); // issue I2C stop signal

Delay\_ms(20);

}

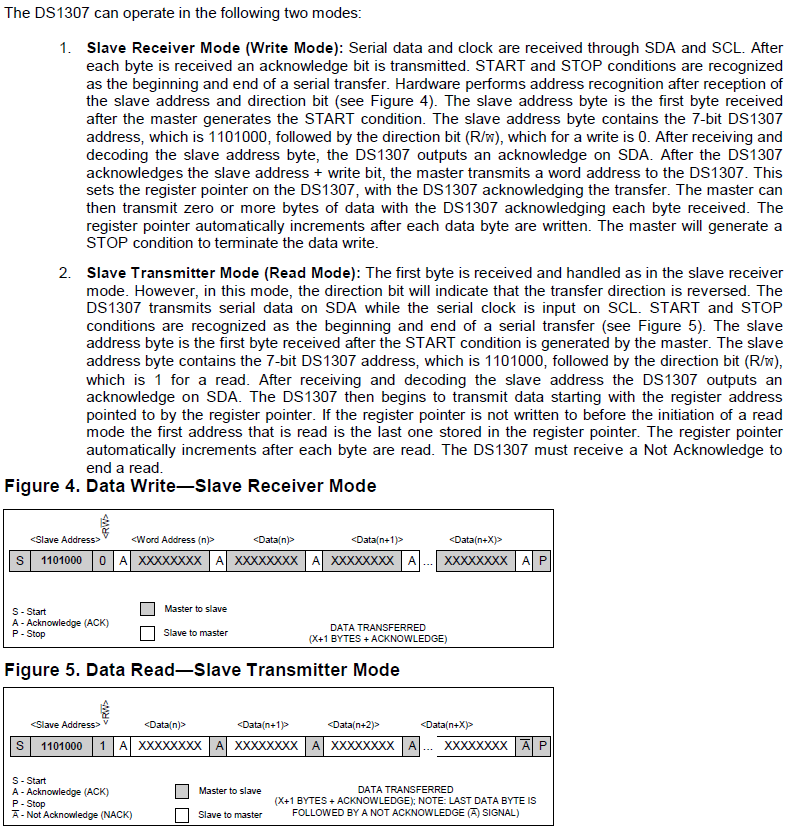
**Appendix H**

Real-time clock Memory Organization



**Appendix I**

Real-time Clock Device Addressing



**Appendix J**

MikroC Code for Reading and Writing Time to DS1307

**Reading Time**

|  |
| --- |
| void Read\_Time(unsigned char \*sec, unsigned char \*min, unsigned char \*hr, unsigned char \*week\_day, unsigned char \*day, unsigned char \*mn, unsigned char \*year) {  I2C1\_Start();  I2C1\_Wr(DEVICEID\_DS1307);  I2C1\_Wr(0);  I2C1\_Repeated\_Start();  I2C1\_Wr(0xD1);  \*sec =I2C1\_Rd(1);  \*min =I2C1\_Rd(1);  \*hr =I2C1\_Rd(1);  \*week\_day =I2C1\_Rd(1);  \*day =I2C1\_Rd(1);  \*mn =I2C1\_Rd(1);  \*year =I2C1\_Rd(0);  I2C1\_Stop();  } |

**Writing Time**

|  |
| --- |
| void Write\_Time(unsigned char sec, unsigned char min, unsigned char hours, unsigned char day, unsigned char dayofweek, unsigned char month, unsigned char year) {  I2C1\_Start(); // issue start signal  I2C1\_Wr(DEVICEID\_DS1307); // address DS1307 which is 0xD0  I2C1\_Wr(0); // start from word at address (REG0)  I2C1\_Wr(0x80 + sec); // write $80 to REG0. (pause counter + 0 sec)  I2C1\_Wr(min); // write 0 to minutes word to (REG1)  I2C1\_Wr(hours); // write 17 to hours word (24-hours mode)(REG2)  I2C1\_Wr(dayofweek); // write 5 - Tuesday (REG3)  I2C1\_Wr(day); // write 18 to date word (REG4)  I2C1\_Wr(month); // write 10 (Oct) to month word (REG5)  I2C1\_Wr(year); // write 12 to year word (REG6)  I2C1\_Stop(); // issue stop signal  I2C1\_Start(); // issue start signal  I2C1\_Wr(DEVICEID\_DS1307); // address DS1307 which is 0xD0  I2C1\_Wr(0); // start from word at address 0  I2C1\_Wr(0); // write 0 to REG0 (enable counting + 0 sec)  I2C1\_Stop(); // issue stop signal  } |