

Advanced Alarm Clock System

A report submitted in partial fulfilment of the requirements, of the Computer Engineering studies at the Institute of Electrical and Electronics Engineering, Malta College of Arts, Science and Technology. Malta.

Mr. Marlon Calleja

June 2013

Abstract

The project excels in the field of home automation in which a real-time-clock activates a blinds opener module and a music player device once the alarm time is reached, an LCD is used to display the time and provide interface to the user for setting the alarm minute and hour. The blinds opener module and alarm clock base station are modular in nature, they are controlled by an 8051 microcontroller and linked wirelessly through RF modules. The objectives of this project is to replace ordinary alarm clocks with an advanced alarm clock system, giving a user-friendly interface and a better wake-up experience in the morning. From studies it shows that if a gradual wake-up routine is adopted by letting natural light fill-up the room over time, it can reduce stress and make the user feel more energetic throughout the day.

Declaration of Authenticity

I, the undersigned, declare that the report entitled: **Advanced Alarm Clock System**; is authentic, and in so far is the result of my own study and research, and that any conclusions, statements, recommendations, or assumptions are mine, unless otherwise stated.

Marlon Calleja
June 2013

Acknowledgements

- Ing. Seguna Clive
- Mr. Baldacchino Nicholas
- Mr. Attard Joseph
- Mr. Bugeja Joshua
- Mr. Mercieca Julian

Table of Contents

| | |
|---|----|
| Chapter 1 - Introduction..... | 1 |
| 1.1 - Aim of the Project..... | 2 |
| 1.2 - Project Description | 2 |
| 1.3 - The Idea and Necessity of the Project..... | 3 |
| 1.4 - Specifications | 4 |
| Chapter 2 - Literature Review..... | 5 |
| 2.1 - Previous Studies | 6 |
| 2.2 - Analysis on Existing Technology and Solutions. | 8 |
| 2.3 - Research on Hardware and Software..... | 13 |
| Chapter 3 - Methodology..... | 18 |
| 3.1 - Planning the Project. | 19 |
| Chapter 4 - Project Implementation and Results..... | 23 |
| 4.1 - Initial Configuration on Proteus Software..... | 24 |
| 4.2 - LCD Pin Configuration..... | 26 |
| 4.3 - LCD Initialisation. | 28 |
| 4.4 - RTC Chip Selection. | 29 |
| 4.5 - RTC Testing on Proteus Software. | 31 |
| 4.6 - Implementation of LCD Module. | 35 |
| 4.7 - Implementation of RTC..... | 36 |
| 4.8 - Serial Communication & RF Solutions. | 38 |
| 4.9 - Configuration of RF Modules..... | 42 |
| 4.10 - Testing of RF Modules. | 44 |
| 4.11 - Alarm Clock Configuration..... | 45 |
| 4.12 - Motor Operation. | 47 |
| 4.13 - Alarm Clock Implementation..... | 52 |
| 4.14 - Final Arrangements. | 54 |

| | |
|--|----|
| Chapter 5 - Conclusions and Limitations..... | 59 |
| 5.1 - Final Overview. | 60 |
| 5.2 - Limitations. | 61 |
| References and Bibliography..... | 62 |
| Appendices..... | 65 |
| Appendices 1 - Flowcharts | |
| Appendices 2 - Program Code | |
| Appendices 3 - Circuit Diagrams | |
| Appendices 4 - Datasheets (provided on CD) | |

List of Figures

| | |
|--|----|
| Figure 1 - Project Block Diagram. | 3 |
| Figure 2 - System overview of a related project (Jake Metz et al, 2010). | 8 |
| Figure 3 - Servo Motor Direction Control using PWM. | 11 |
| Figure 4 - Philips Wake-up Light HF3520/01 | 12 |
| Figure 5 - T89C51 Training Board. | 15 |
| Figure 6 - Display time using 7-Segment LEDs, converters & counters (Brian Marshall, 2000). | 16 |
| Figure 7 - Milestone Plan. | 19 |
| Figure 8 - Proteus LCD Connection. | 24 |
| Figure 9 - External Oscillator and Reset Circuitry. | 25 |
| Figure 10 - Proteus RTC Connection. | 31 |
| Figure 11 - LCD Time Display Flowchart. | 33 |
| Figure 12 - LCD connected on breadboard. | 35 |
| Figure 13 - Time being displayed on LCD. | 38 |
| Figure 14 - EasyBee connected with USB UART 2 Board. | 42 |
| Figure 15 - HyperTerminal interface showing typed commands for configuring RF module as router. | 43 |
| Figure 16 - UART Connection Between RF module and MCU (ZigBit ATZB-24-A2 Datasheet). | 44 |
| Figure 17 - Motor Direction Control using Relays. | 48 |
| Figure 18 - Motor Direction Control using L293D. | 49 |
| Figure 19 - Opto-Sensor Circuitry. | 50 |
| Figure 20 - USB Power Output Circuit. | 53 |
| Figure 21 - Resistor soldered in series with 4ohm speaker. | 54 |
| Figure 22 - Power Supply Circuit of T89C51 Training Board (JR51AC2 User Manual). | 55 |
| Figure 23 - Alarm Clock Base Station PCB. | 56 |
| Figure 24 - Blinds top sensor and aluminium rail | 57 |
| Figure 25 - Alarm Clock Base Station. | 58 |
| Figure 26 - Blinds Opener Module. | 58 |

List of Tables

| | |
|---|----|
| Table 1 - Pin Description of 1602 Hitachi HD44780 Based LCD. | 26 |
| Table 2 - DC Gear Motor Specifications. | 47 |
| Table 3 - Truth Table of L293D for Motor Direction Control. | 49 |

Chapter 1.

Introduction

1.1 - Aim of the Project

The intension of the project is to address the field of home automation and sleep health. The project's aim is to aid the user both mentally and physically by adapting a natural wake-up routine, enhancing the experience of morning wake-up for a better overall feeling of reduced stress. The project is intended to replace ordinary alarm clocks with an advanced system which uses music and natural light to wake up the user more effectively and provide positive energy to start the day.

1.2 - Project Description

The final project consists of a real time clock which can be set by the user to activate the alarm at a desired point in time. By having a microcontroller interfaced with an LCD module and RTC chip, the current time can be displayed and updated every second. When the alarm time is reached, a wireless RF signal is transmitted from the alarm clock base station and picked up by the receiving end. The microcontroller at the receiver circuit will then control the direction and rotation of the motor through the H-Bridge. The blinds are designed to open at a slow pace, allowing more light to enter the room over time. From studies it shows that when applying this method, it mentally helps the body to wake up and feel more energetic. Photo interrupter sensors are used to detect if the blinds are fully opened or closed, thus allowing the microcontroller to stop the motor from further rotation. A music player device was also implemented to play pre-loaded sound tracks from an SD card when the set alarm time is reached.

Both microcontrollers are programmed in C language and implemented with a T89C51AC2 library . As can be seen from Figure 1, both microcontrollers are linked together wirelessly through the transceivers, forming a peer-to-peer network. This method creates a communication environment for the modules since the microcontrollers keep each other updated from any software or hardware event which has been resulted in their respective station.

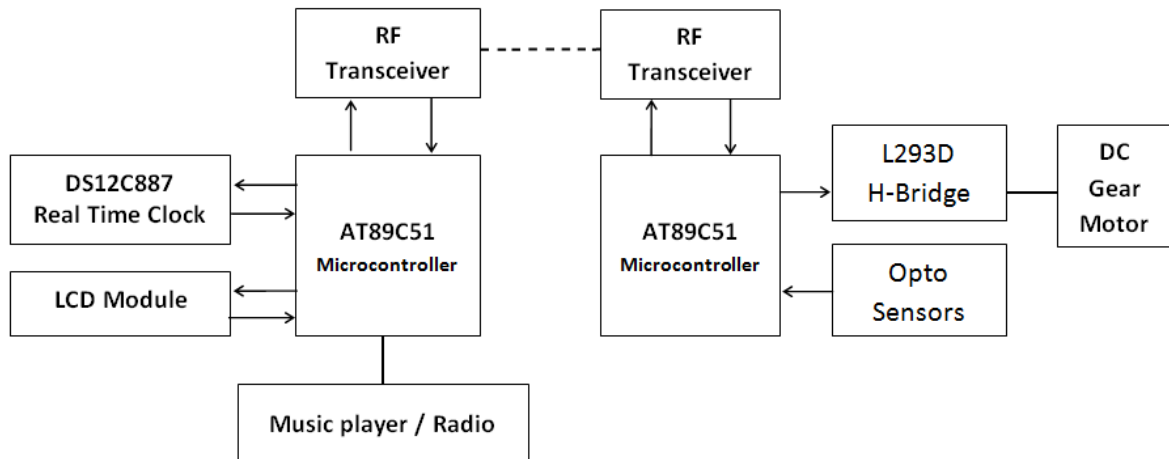


Figure 1 - Project Block Diagram.

1.3 - The Idea and Necessity of the Project

The idea of this project came about as a result of daily morning routines, having to wake up early and rushing to prepare for work or school lectures. Even after a suitable amount of sleep, the feeling of tiredness is frequently experienced. From a Discovery Channel program named The Gadget Show, the author was inspired by a particular episode. The presenters of the program were given the task to set-up an automated environment for a hotel room scenario. Gadgets were used to enhance the experience and morning wake-up of a protagonist in which he assessed the quality. The lights and TV were switched on and off through voice recognition, a coffee maker and motorised blinds were set to function within the alarm time period.

If a good wake-up routine is arranged, it can provide an extra boost to our body and improve the performance of our day. Ordinary alarm clocks tend to break our sleep almost immediately and for this reason fill us with stress. An excellent way to wake up is naturally by having the room filled up with sunlight and by the relaxing sound of music in the background.

1.4 - Specifications

The project hardware is modular in nature, being comprised of two units:

Alarm Clock Base Station

- **Supply Voltage:** 9-12V
- **Maximum Current:** 1A
- **Output:** USB panel mount for connecting music player or +5V rechargeable devices.
- **Internal Adjustment:** LCD contrast level
- **Dimensions:** 23 x 20 x 11 (cm)
- **Features:**
 - Metal enclosure with Perspex window.
 - 16x2 LCD interface for time display and setting.
 - Nickel push buttons for setting time, alarm or turning on/off USB output.

Blinds Opener Module

- **Supply Voltage:** 12V
- **Maximum Current:** 300mA
- **Dimensions:** 46 x 25 x 53 (cm)
- **Features:**
 - Clear Plastic cover.
 - Nickel push buttons for closing/opening blinds.
 - Small scale model of roller blinds.
 - 12V DC - 2RPM Gear motor.

Chapter 2.

Literature

Review

2.1 - Previous Studies

Sleep and wake is the natural cycle of life which we exploit daily. "By cutting down on sleep, we learn less, we develop less, we are less bright, we make worse decisions, we accomplish less, we are less productive, we are more prone to errors, and we undermine our true intellectual potential!" (Dr Piotr A. Wozniak, 2012). Sleep is required to reset our brain to a fresh state and prepares it for new information to be acquired the following day, a rested mind is able to absorb information more efficiently and as a result improves learning. Good quality sleep is also required to help us concentrate in handling quick and correct decisions during work so we make less mistakes and therefore be more productive. Sleep is essential to keep our body functioning and healthy, without rest we won't have the energy to be active physically nor mentally.

From news published on American Thoracic Society an article explains the conducted studies of Alan Mulgrew, MD, involving car accidents caused due to sleep deprivation. (American Thoracic Society, 2007). A test was carried out on 1600 people which were divided in two groups, 50% of which suffered from sleep apnoea while the remaining 50% followed a normal sleep pattern. Sleep apnoea is a disorder which affects the way of breathing during sleep, it interrupts the natural sleep cycle and prevents you from having the required amount of rest. As mentioned in the article, over a three year period a total of 373 accidents were reported which comprise 250 accidents from the first group and 123 accidents from the second group. It was stated that the people which suffered from sleep apnoea were twice as likely to have an accident and three to five times likely to be involved in a personal injury due to a serious car crash.

The example of patients suffering from sleep apnoea was discussed to show the fact that people which lack in sleep are less concentrated and have a lower reaction time, thus are more prone to accidents. This concludes that sleep is highly important in our daily life since it can keep us more alert from possibilities of danger which surround us. A good sleep practice is also required moreover the extent of rest, it was found that it's not the amount of sleep that matters mostly but the quality. Being interrupted during sleep is unhealthy since it is the major cause of stress and low concentration, on the other hand oversleeping can lead to medical problems such as obesity and headaches.

For a better sleep experience, to feel more energetic and refreshed in the morning, a good wake-up routine is required. Ordinary alarm clocks tend to start our day with a high level of stress since they bring us from deep sleep to wide awake almost immediately. Being exposed to sunlight is a better and natural practise since it can gently stimulate our body and mind to wake up. In a part of an online article the author discusses and proves the fact that early-morning sunlight can help us sleep at night. (Virgil D. Wooten, M.D., 2012). It was explained that sunlight regulates our biological clock and keeps it on track, when we expose ourselves to light in the morning it resets our daily clock and counters any forward drifts (e.g. staying up late). It is hard to sleep when you're under a lot of stress, hence if a good wake-up routine is followed, the level of stress could be reduced and as a result helps you to sleep at night. This shows that a quality wake up is directly related to sleep and is equally important to maintain a stable sleep-wake cycle.

Apart from sleep apnoea, SAD (Seasonal Affective Disorder) is another illness from which people commonly suffer. SAD affects people in a different way, it is a type of depression that typically occurs during the autumn and winter seasons when the day is shorter than the night, thus giving less exposure to sunlight within a 24 hours period. (Josepha Cheong, M.D., et al, 2010). Main symptoms include low mood and lack of interest in life, also it can affect physically the body by reducing the level of energy, thus feeling more tired and less active than normal. Treatment for SAD typically involves daily light therapy, medication is only required for more severe cases of depression.

The symptoms of SAD have been discussed so the author could make an emphasis on how much sunlight is important in our daily life since it is medically proven that lack of natural light could cause problems in the way of living. As it can be noticed, the previous article is related to the subject and thus methods could be adopted. If the mentioned wake-up routine is used in practice, early-morning sunlight can help with the therapy, although a long duration of light exposure is still required for the treatment to be effective.

2.2 - Analysis on Existing Technology and Solutions

Dedicated research and studies already solved the majority of problems which concern rest. Existing technology in the field of home automation have enhanced the experience of sleep and wake by adapting a wide range of needs. In a finished project several components and modules were assembled to automate the morning wake-up experience. (Jake Metz et al, 2010). A centralized base station was implemented to communicate with various module circuits either through wire or via wireless RF signals. The network consisted of a coffee maker, a thermostat, a blinds opener module and a voice synthesizer circuit, built to allow the user to input a custom vocal wake-up announcement. All the modules are controlled by an alarm clock circuit located in the base station which are activated at a specific time interval when the allotted time chosen by the user is reached. Figure 2 shows an overview of the mentioned project, as a general remark the internet section was cut due to time constraints. The original idea was initially intended to provide internet connectivity to the alarm clock base station so it will allow the user to set the time via a web-interface and select which modules would be turned on. Also, the user could login into an online calendar so the first appointment of the day can be downloaded and announced audibly.

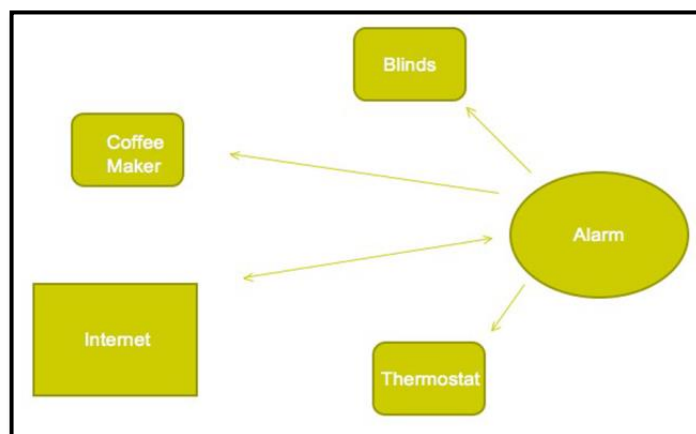


Figure 2 - System overview of a related project (Jake Metz et al, 2010).

Within the above mentioned project, the alarm clock module is controlled by a PIC microcontroller which reads the current time from a DS1307 RTC (Real-Time-Clock) chip, an I²C interface protocol is used so both devices can constantly update and communicate with each other. Each time segment recorded by the microcontroller

(hh:mm) is outputted in a respective format to be understood by a BCD to seven segment converter IC, the time is then decoded and displayed on seven segment LEDs. If the alarm time set by the user matches the current time, the base station microcontroller instructs the transmitter to output a signal which is picked-up by the designated receivers in the network. Each module is then controlled by its respective microcontroller once the relay has been activated either through wire or via wireless RF signals. Every relay is supplied with the required amount of voltage to operate and is controlled by a PIC microcontroller which is vital for the operation of the components to perform the required tasks.

After a brief analysis on the above mentioned project, different methods could be studied. If the seven segment LEDs were replaced by an LCD module to display the time, the execution of the project could have been better in terms of programming, presentation and cost since less components would have been used. A more efficient communication could also have been achieved between the devices since less I/O pins would have been taken from the microcontroller, thus saving valuable pins for other operations. Through further programming, the displaying of special/custom characters and text animation would also have been possible.

The Hitachi HD44780 based LCD is ideal to perform the required task since it is economical and widely used in the market. It uses a 16X2 display (16 characters wide, 2 lines of text) and has sixteen pins of connectivity. Three pins are used as control lines, eight pins are used for data I/O, four pins are used to supply the LCD and backlight with voltage and ground and one pin is used to change the contrast level of the LCD display. The LCD module can be operated by inputting instructions and setting the logic levels of the control pins, thus a general procedure must be followed. The control pins should be set or cleared according to the information to be interpreted by the LCD. If the data to be inputted is a command, the RS (Register Select) control pin must be set to Low, if the data to be inputted is in character form, the RS control pin must be set to High. If the data is to be written or displayed on the LCD, the R/W (Read/Write) control pin must be set to Low, if the data is to be read from the LCD module, the R/W control pin must be set to High. For loading the configuration of RS and R/W into the LCD module, the E control pin must be supplied with a High to Low pulse to finalise the process.

The blinds opener module could have been implemented with a cheaper solution since a lot of resources weren't required to handle the specific task. Only few I/O pins were used from the PIC microcontroller, therefore the major amount of utility was unnecessary. Instead of using a microcontroller to drive the motor in the relay, a 555 timer IC might have been an alternative option for generating PWM signals and operate the motor at a set speed after a signal has been transmitted to the receiver. (Rick Bickle, 2005). In the tutorial the author describes that pulse width modulation is a technique used for switching the power of a component on and off at a desired frequency. PWM is required in DC motor speed control to have an efficient and stable output, preventing heat from being generated during operation and hence wasting less energy. Since the power switching is fast for the motor, it is detected as if a constant voltage level is being applied. The voltage level can be set by adjusting the duty cycle of the PWM, therefore if a duty cycle of 50% is present, half of the maximum voltage supplied by the power source will be theoretically present on the motor. The required speed at which the blinds are opened can be adjusted by choosing the correct value for the components. If the 555 timer is configured as an astable oscillator, the components that determine the duty cycle of the PWM are the capacitor and resistor which compose the circuit's RC network (R1 and C1). The greater the value of the capacitor and resistor, the higher the duty cycle, thus a faster rotation of the DC motor is achieved.

After further analysis on the above mentioned project, it was found out that the blinds opener module was using a servo motor for the rotation of the blinds. This deviated completely the author's studies of using a 555 timer IC for driving the motor, in servos the PWM determines the angle of the horn/arm and not the rotational speed of the spline (shaft). In the project that Jake Metz and his team has built, the microcontroller was programmed to set the PWM period to 8.0us and the duty cycle to keep changing from 30% to 100%. Consequently the servo motor operated to the changing duty cycle of the PWM, set to rotate by 180° for the duration of 5 seconds.

Figure 3 shows a sketch which represents the servo's horn/arm angle rotation in relation to the pulse width (duty cycle), the drawing has been made to explain more clearly the principles of servo motor control. From the drawing it shows that every signal is 20ms wide and has a different pulse width, the periodic time of the wave can be set by the user. The width of the pulse determines the degree position to which the motor rotates, minimum and maximum pulse widths are set from the factory and can vary between

different motor types and manufacturers. The minimum pulse width of 1ms rotates the motor to its 0° position while the maximum pulse width of 2ms rotates the motor to its 180° position. The 90° position is known as the neutral position since it is the point where the servo is stationed exactly in the middle. If a pulse width in the range of 1ms - 1.5ms is sent to the servo, the arm rotates a number of degrees anti-clockwise from the neutral position. If a pulse width in the range of 1.5ms - 2ms is sent to the servo, the arm rotates a number of degrees clockwise from the neutral position.

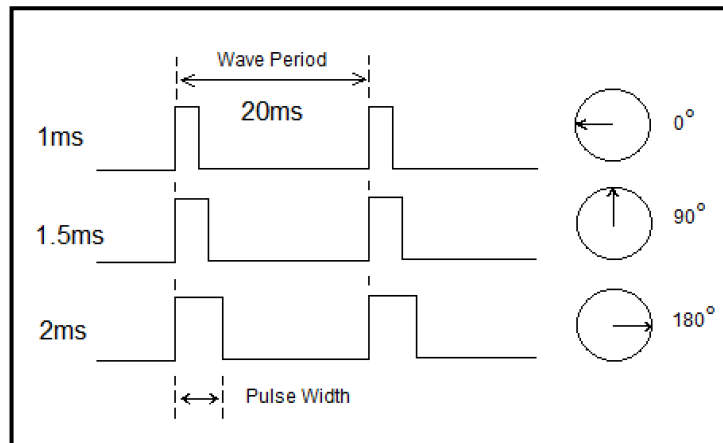


Figure 3 - Servo Motor Direction Control using PWM.

Although the author mentioned that the blinds opener module could have been implemented with a 555 timer for driving the DC motor in the relay, some operations would be difficult to be applied into practise. Without using a microcontroller in the circuit, absolute control on the hardware will be difficult to achieve. The data received by the RF module cannot be interpreted to perform a certain action. The motor wouldn't have the ability to rotate in both directions and have the indication to stop at a desired point in time. Also if an unexpected result occurs, the circuit cannot react to the problem and perform a process of steps to solve it. If for example a collision occurs while the blinds are opening or closing, the motor cannot be stopped from rotating since no instructions can be given.

For each different category of blinds, a different mechanism is used for lighting up the room. From a small range of blinds; roman blinds, roller blinds, venetian blinds and vertical blinds, the roller blinds were chosen to illustrate the author's concept since they are the least complicated to implement. Roller blinds make use of a 360° rotatable drum

to pull-up or down the blinds completely after multiple turns, thus the amount of light that enters the room is affected by the uncovered area of the window. Apart from folding on one side of the window, venetian blinds and vertical blinds are mainly designed to rotate their slats back and forth at an angle of 180° , thus the amount of light that enters the room is affected by the angle of the slats and location of the sun. As a conclusion, blinds which are designed to rotate their slats cannot simulate sunrise correctly since they depend on two variables.

For the rotation of the drum, a DC gear motor was chosen since it offers high torque at low speed, hence PWM isn't required for the operation of the motor because speed is already limited. Speed isn't crucial for the current application since the concept of the project is to wake up the user by the gradual increase of light intensity. Servo motors were not taken into consideration when choosing the ideal motor since they can only rotate within an angle of 0° to 180° and are only suitable for angle blinds. The only advantage that was considered in servo motors was that if angle blinds were chosen for the project, less energy would have been consumed per day since less rotation is required for the blinds to be fully open or close. Although efficiency is acquired by this method, an impact on the concept of the project is resulted.

During the design of the project and previous studies it was noted that the time and period at which the sun shines cannot be controlled since the weather does not permit this at all times. Also as already mentioned, during the winter and autumn seasons it generally takes longer to break dawn. Along with this statement one could also add that the majority of people wake up earlier for work, thus darkness would still be present. When considering these assumptions, in certain circumstances the project would be irrelevant to implement.

After analysing existing technology in the market it was concluded that the particular problem can be solved by implementing a light source which can simulate sunrise during wake-up. This specific scenario was introduced within a development where a lamp alarm clock was created. (Stephen E. Blackman, 2001). The appliance is fully controlled and can allow the user to set the desired time at



Figure 4 - Philips Wake-up Light HF3520/01

which light is produced. Furthermore it includes a speaker which produces pre-set sounds during wake-up and a dimmer switch for reducing the light intensity. A similar product is shown in Figure 4, it is designed by Philips and has similar features as the appliance mentioned above.

2.3 - Research on Hardware and Software

The author's preference choice of microcontroller is the Atmel AT89C51, it is part of the 8051 family and is based on a CISC (Complex Instruction Set Computer) architecture. In an automated password based car parking system the Atmel microcontroller was implemented to handle the operation of the project. (Nayab Suhail Hamirani et al, 2011). As it was explained, the AT89C51 microcontroller was preferred due to its simplicity and cost even though an AVR or PIC microcontroller could have been used to perform the same task. The intention of the project was to increase the security and parking space for the cars by ensuring a password based locking facility and a rotational multi-storey building. With the project's evidence of work from the write-up, the 8051 microcontroller can be assured for its functionality since it handled both the operation of the DC motors and the interfacing of the LCD successfully.

With respect to the above mentioned project, the author doesn't agree with the fact that preference for the 8051 microcontroller can be made because of its cost. In fact, the 8051 microcontroller is expensive when compared to the performance of PIC and AVR microcontrollers. The last two mentioned family type microcontrollers are more efficient and faster for executing commands per clock cycle since they are based on a RISC (Reduced Instruction Set Computer) architecture.

From the lecturer's notes it was explained that the primary difference between CISC and RISC architecture is that a RISC-based chip uses more basic instruction sets to achieve a greater clock frequency, hence it can process more information per clock cycle than a CISC processor. CISC-based chips, however, give developers the ability to do more with a shorter program due to the greater library of complex instructions embedded in the CISC chip. Although the CISC architecture helps speed up the

programs' execution, the number of instructions loaded on the processor negatively impacts processor performance. As a result, more transistors are built into the microprocessor to improve performance, which can result in an increase in the unit's cost.

At the present time within the engineering industry, PIC and AVR microcontrollers are more preferred over 8051 microcontroller for their recent design, they have an updated programming software which makes it compatible with the latest computer platforms. Also the software is provided with more powerful tools and features for debugging the source code. Even though the 8051 microcontroller is not ideal for performance, it is still easy to program and implement hardware, making it a good choice for projects. After taking into consideration the requirements needed for the project, it was concluded that the 8051 microcontroller is sufficient for its operation. It has enough data ports for interfacing all the hardware while keeping a stable and accurate execution of the real-time environment. Also, the author has a better knowledge base on the AT89C51 chip since the required training was provided by MCAST. For this reason confidence is achieved to use the microcontroller since it is established by the EEEI and is proven for its functionality in students' projects.

For a better throughput performance, a T89C51 board was used in the project since it offers a number of expanded features over a stand-alone microcontroller. The training board can be easily supplied with a voltage range of 9V to 12V from an onboard plug pack connection. A regulator stabilises the voltage at +5V for the microcontroller to operate efficiently with no intension of being damaged from over-voltage. An onboard 18.432Hz crystal is already included with the package so it can define the clock speed of the processor to handle the instructions responsively. Communication can also be made with a PC through the onboard RS232 socket when it is connected to the computer's serial port via a 5 pin polarized connector to a DB9 connector. A special software should then be installed on the computer so it can allow the microcontroller to be interfaced serially with the PC. A special socket (ET-Download) is also included and uses the same connection as the RS232 socket to communicate with the PC. The main difference is that the socket is specially designed for the purpose of flashing the source code into the microcontroller's memory with a programmer software. The T89C51 includes a large 32kb of program flash memory, together with a large RAM for storing temporary variables. A large 2k EEPROM is also included for retaining preset values

while the power is disconnected. All the microcontroller's I/O pins are connected to a header so soldering can be avoided and thus a standard 34 pin IDC connector can be used to connect to the socket for easy external connections.

Figure 5 illustrates a photo of the T89C51 training board which was taken to show a preview of the board's structure and installed components. The only drawbacks that has been considered for the implementation of the board was the cost and size. Extra features are included with the board which are not at interest for the project, taking unnecessary costs from the budget and space for the implementation of the project. The T89C51 board was chosen for the project since it offers an all-in-one package, making it more compact and easier for prototyping and flashing source code.

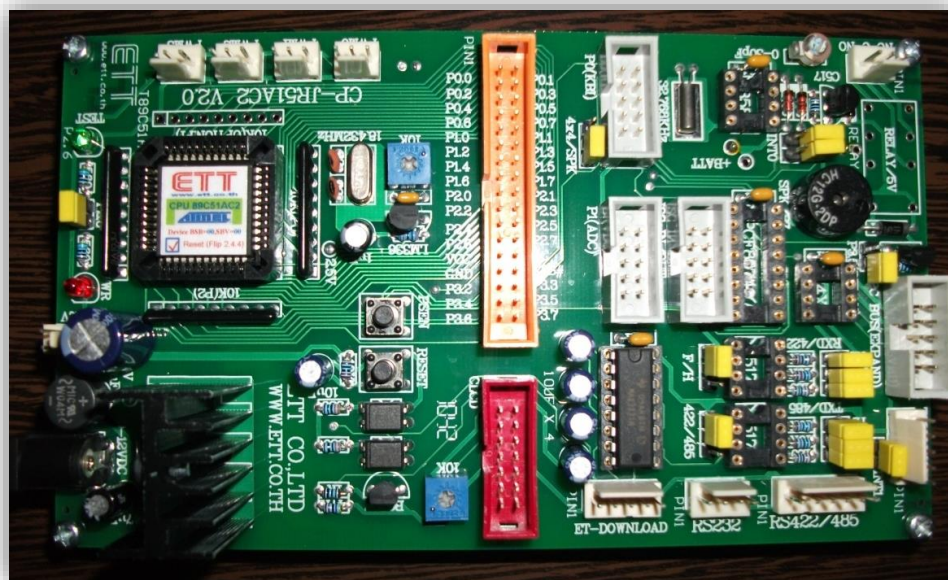


Figure 5 - T89C51 Training Board.

The source code of the microcontroller can be programmed either in Assembly language or embedded C. After consulting with the lecturer about this matter, it was advised to work with C language since it is easier to maintain and debug, also the same program function can be implemented with less coding when compared to Assembly. For its High-Level language, it is easier to read and understand, and for that reason it is much more preferred in the engineering industry. The advantage of using Assembly language is that in certain operations, greater efficiency can be resulted since the code takes less time to be converted into machine language.

The fundamental part of the project is the digital clock, a time keeping function is required for the hardware to be synchronized with real-time so it can react accordingly when the current time reaches the set alarm time. From an online article the author explains the construction and operation behind the digital clock. (Brian Marshall, 2000). For an accurate time base and perfect driving of the "second hand" portion of the display, a signal of 1-Hz is required. The signal can be generated using a crystal oscillator or by extracting the 60-Hz oscillation from a power line, however the crystal oscillator method can result in more accurate timing.

The 60-Hz signal can be divided down to a 1-Hz signal by using 7490 decade counters. One counter can be configured to divide the 60-Hz signal by 10, thus generating a signal of 6-Hz. Another counter can be configured to divide the 6-Hz signal by 6, hence outputting the required oscillation of 1-Hz. This process creates the heartbeat of the clock since it pumps a signal every actual second. To display the count of seconds into actual time, two 7-segment LEDs should be used in each segment (hh:mm:ss). For every segment, two decade counter and two binary to 7-segment display converter are also required. Figure 6 shows a block diagram of the "second hand" portion of the display so a better analysis can be made.

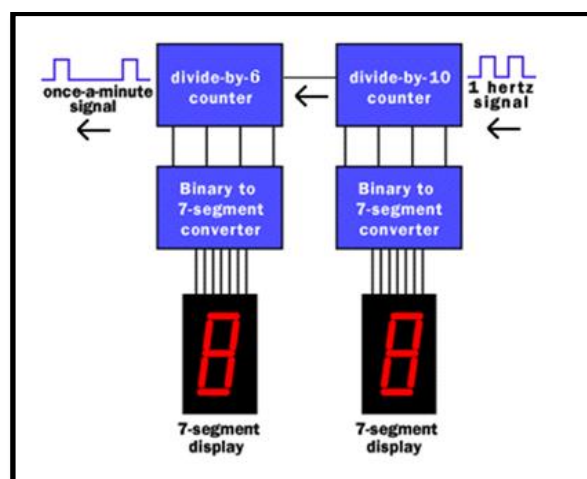


Figure 6 - Display time using 7-Segment LEDs, converters & counters (Brian Marshall, 2000).

The 1-Hz signal generated by the previous counters is used to increment the divide-by-10 counter on each pulse it receives. The divide-by-10 counter produces a 0-1-2-3-4-5-6-7-8-9 sequence on its outputs, incrementing the divide-by-6 counter every time it resets back to 0. The divide-by-6 counter produces a 0-1-2-3-4-5 sequence on its

outputs, generating a pulse for the Minutes segment every time it resets back to 0. Since the decade counters represent the numbers in binary form, binary to 7-segment converters are used to light up the respective bars on the 7-segment LEDs, displaying the equivalent decimal value for each segment. While the Minutes segment counting functions the same as the Seconds segment, the Hours segment requires a divide-by-2 counter instead of the divide-by-6 counter if the clock is to be configured in a 12-hour format. Logic gates are used to instruct the clock to cycle back to 1:00 when the time 12:59:59 is reached.

For implementing the time keeping functions in a single package, the lecturer's advice was to use an RTC chip. A 1-HZ quartz crystal is embedded in this IC so it can provide itself with an accurate oscillation of time. A CMOS battery is also included so it prevents any configuration to be lost during a power outage, also it can keep oscillating with a constant time update. The chip is ideally used in an MCU based system since a microcontroller is needed for the interfacing of the RTC. Registers are used for storing time and configuration in a Binary-Coded-Decimal or binary format. The main feature which makes an RTC chip excel over a counter based clock is the utility of using a calendar. Apart from a 24/12 hour time keeping function, the date is also recorded and can be synchronised up to centuries. Features of Daylight Savings Time, leap year compensation and setting between a 24-hour clock and a 12-hour clock with AM and PM are also included. Alarm time registers can be set so the RTC can notify the microcontroller when the current time matches the alarm time. As a conclusion, an RTC chip gives higher utility and reliability in a smaller sized package with a cheaper solution since external components, excluding the microcontroller, are not required.

Chapter 3.

Methodology

3.1 - Planning the Project

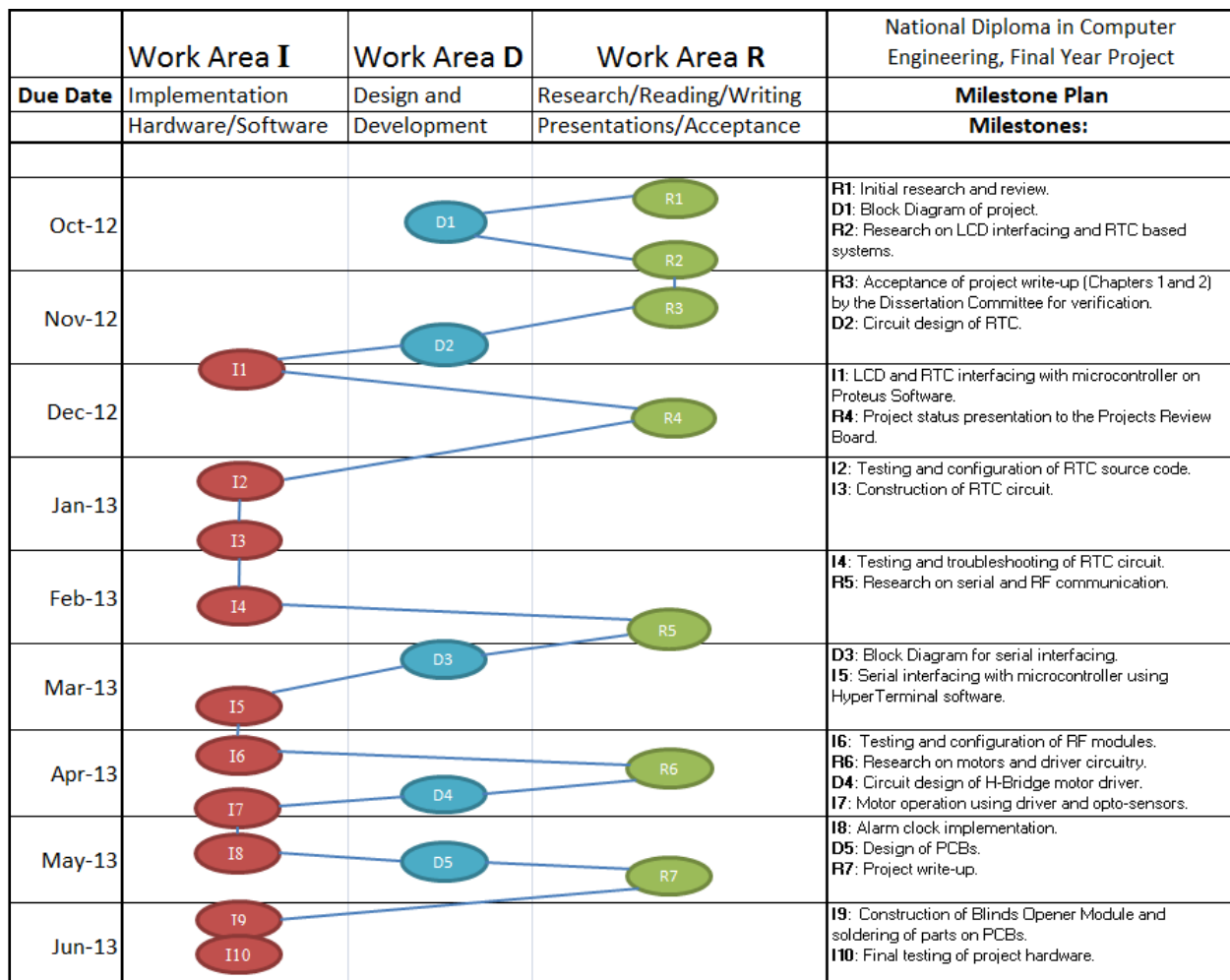


Figure 7 - Milestone Plan.

For project planning and analysis, three main areas have been chosen to represent the course work. The research area is the fundamental section since it is the key for the project's idea. Activities of dissertation, research and presentation are listed under the research area to show the theory and analysis done behind the completion of the project. The research area was important for the selection of components to distinguish which part is the most efficient, cost effective and ideal for the project. For this reason, the design area was dependent on the research area since a general background on the functionality of the project was needed before circuits could be designed. The design area held the planning behind the project's construction and served as an intermediary between the research and implementation areas. The implementation area

can be seen as the practical section since it contains the actual build-up of the project, it is based on the research and planning made beforehand. Activities of testing, debugging and troubleshooting fall under the implementation area to show the practise of steps taken to a full functional project.

Once the author's project has been approved by the MCAST Projects Review Board, initial research and review was carried out for a consecutive amount of four weeks. During the duration of this time period, evaluation on the project's implementation and construction was made by referring to existing technology in the market. A grasp on the project's background was obtained and therefore the execution of the project commenced accordingly from milestone **R1**.

With a basic knowledge on the area and topic of the project, an alteration on the proposal block diagram was made. By the end of milestone **D1**, the block diagram was revised with a detailed aspect and served as a plan for the construction of the project. For reaching milestone **R2**, the author committed to further research on LCD interfacing and RTC based systems, it was intended to aid in the construction of the project. The components used were chosen after consulting matters with lecturers and after comparing parts from datasheets, the author expresses the preference of components in the literature review. By the end of milestone **R3**, the Dissertation Committee accepted a preliminary write-up of Task 1 so verification can be made. During the same work phase, ordering of parts was made so components were at hand when circuits were put into practice.

Subsequent to further planning and consultation, it was concluded that the first problem that should be solved from the project was the Real-Time-Clock. The particular section was chosen since it holds the majority of the project's work and is required since all the components are dependent on the RTC.

By the end of milestone **D2**, a circuit was built on Proteus software so testing and simulation could be performed before putting the actual circuit into practice. With reference to technical sites and books, LCD and RTC interfacing with microcontroller was completed by the end of milestone **I1**. For the Real-Time-Clock simulation to work successfully and for the time to be displayed correctly on the LCD, a period of four weeks was taken. Testing and configuration of the source code was done with Keil software, the coding was written in C language and a library for the 8051 microcontroller

was used. Shortly after the end of milestone **I2**, confirmation by the lecturer was given to build the actual circuit since the simulation phase was finished and was ready to be tested in real-time.

The construction of the actual circuit was based on the design of Proteus, by the end of milestone **I3** the circuit was ready. The microcontroller was then flashed with the source code using Atmel Flip software. Although the simulation worked as intended, the time displayed on the actual LCD wasn't oscillating. Procedures of troubleshooting and consultation with lecturers was done for the circuit to function correctly, reaching milestone **I4**.

Since the RTC circuit was functional, research on RF communication and serial interfacing commenced immediately, the RF modules were chosen after consulting matters with the lecturer during the period of milestone **R5**. A block diagram was sketched by the end of milestone **D3** to plan connections between the microcontroller and RF modules using a serial interface.

Before the microcontroller could be interfaced with the RF modules, the lecturer's advice was to test serial communication between the microcontroller and the PC using HyperTerminal software, when the task was complete milestone **I5** was reached. To configure the RF modules with At-commands, reference with datasheets was made and a serial interface was used to communicate with the PC serial port. After testing communication between the transceivers and microcontroller the author reached milestone **I6**.

To reach milestone **R6**, in depth research was made on motors and driver circuitry so an efficient solution can be implemented. After selecting the ideal components, by the end of milestone **D4** a circuit design was made on Proteus software so a better plan of the motor operation can be viewed. At milestone **I7** the motor was fully functional, connection of H-Bridge driver chip and photo interrupter sensors was made and tested on the breadboard with the microcontroller.

By the end of milestone **I8**, the microcontrollers were programmed to activate the blinds motor and music player device upon reaching the alarm time. Once all the hardware was functional, at milestone **D5** the PCBs were already designed on Proteus and submitted to MCAST to etch the boards locally. To utilise the time while waiting for the

PCBs to be finished, finalisation of the write-up was made and completed shortly by the end of May at milestone **R7**.

When the PCBs were ready, soldering of components was made. Also a small scale model was built to demonstrate the concept of the project, enclosure for the hardware was also developed to contain all the components in one place. Shortly after milestone **I9**, final testing was made to ensure the functionality of all the hardware before reaching the project's deadline date.

Chapter 4.

Project

Implementation

and Results

4.1 - Initial Configuration on Proteus Software

With reference to an 8051 microcontroller technical book which was initially recommended by the lecturer, fundamental parts of the project were tested on Proteus. The simulation software was used to prepare the program functionality for the actual hardware so when the components are imported, testing and implementation can initiate immediately. Subsequent to further consultation with the lecturer, it was decided that the LCD module should be the first component to be operated prior to the implementation of the RTC. This was decided since a time display source is required before the RTC could be tested, without the LCD it would be difficult to evaluate if the microcontroller is reading the time from the RTC correctly or if it is functioning at all. Figure 8 shows the initial circuit design for operating the LCD module on Proteus.

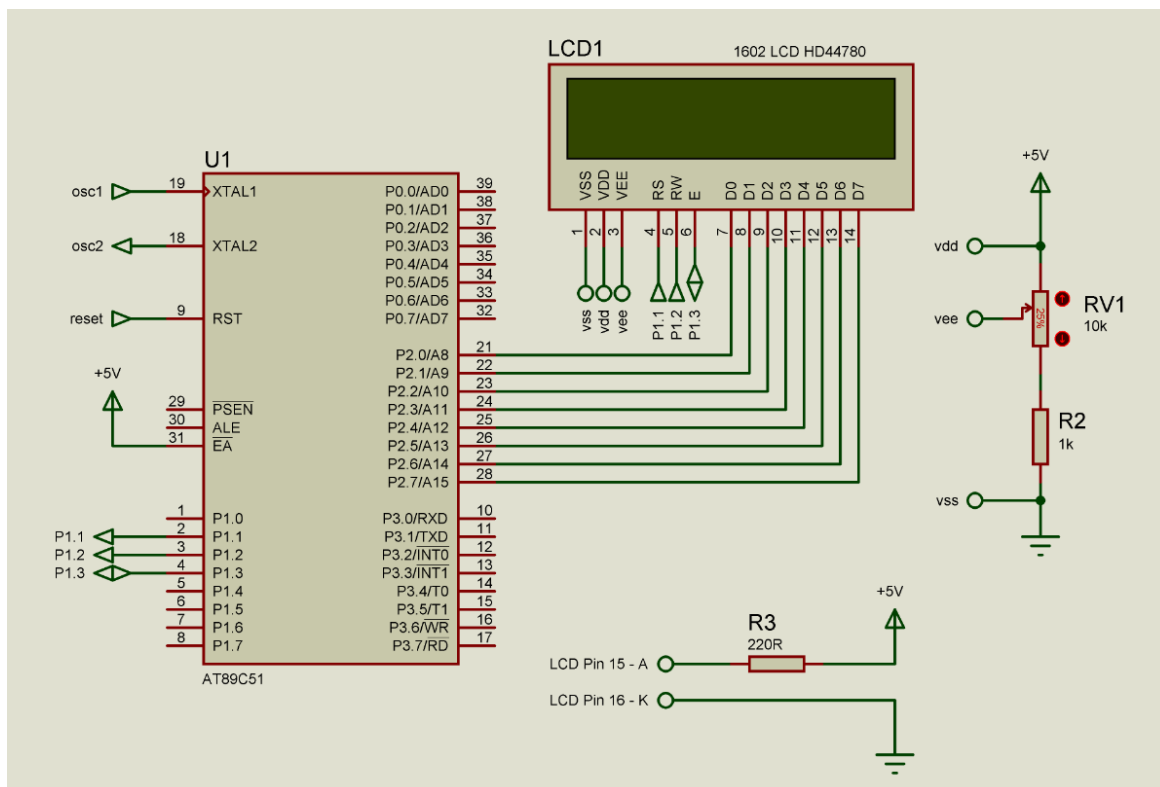


Figure 8 - Proteus LCD Connection.

As the component library of Proteus can provide a 16x2 LCD with only 14 pins, the actual LCD module has 2 extra pin-outs for supplying the backlight with voltage and ground (A and K). As shown in Figure 8, separate circuitry has been added to represent the connection made in practice. A 220 ohms resistor (R3) has been used to limit the

current of the backlight and thus reduce the brightness level. For simulation purposes the 10K pot resistor (RV1) and the 1K resistor (R2) weren't necessary but they were added to illustrate the actual circuit as VEE was connected to ground in Proteus. In practice, a trimmer pot has been used for adjusting the contrast level of the LCD, the 1K resistor was connected in series so it can limit the maximum current flow when the potentiometer is set to its minimum position. EA was connected to +5V so it can execute the source code from internal memory. Connection with P1.0 was not made since it is utilised by Timer 2 as a clock input, function of the timer is explained later in section 4.8. RST, XTAL1 and XTAL2 connections with terminal inputs/outputs osc1, osc2 and reset are discussed below.

The circuitry shown in Figure 9 is based on the actual schematics and components' value of the T89C51 board. Identical circuitry for the reset and external oscillator were constructed on Proteus so similar results from the actual hardware could be simulated.

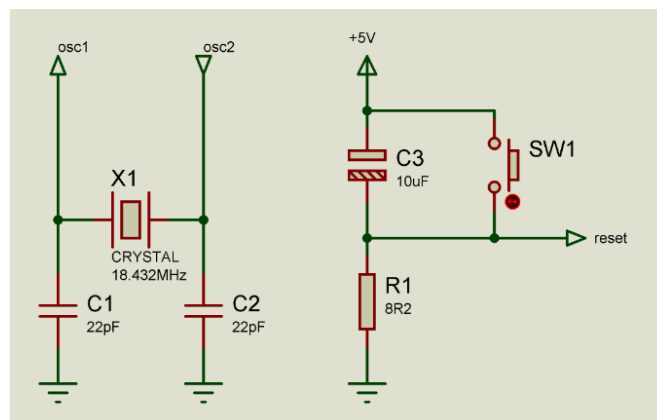


Figure 9 - External Oscillator and Reset Circuitry.

Referring to Figure 9, in the external oscillator circuit an 18.432MHz crystal (X1) is normally used with 8051 to determine the operating frequency of the microcontroller so it can synchronize the timing of machine cycles. The 22pF capacitors (C1 and C2) are used for start-up stability to ensure the crystal's oscillation.

During normal operation, the microcontroller's reset pin is normally held low by the pull down resistor (R1). A value of 8R2 is only used for simulation purposes with Proteus since the T89C51 board normally uses a 10k resistor to determine the charging time of C3. The switch (SW1) is used to short the 10uF capacitor (C3), in effect the +5V supply voltage momentarily pulls the reset pin high and resets the microcontroller to a power-on

state if the switch is kept pressed for a minimum duration of two machine cycles. When the switch is released, the capacitor is charged over time, pulling the reset pin low again for normal operation by the microcontroller. The total duration for the capacitor to be fully charged and the duration that the reset pin is kept high was calculated by the multiplication of the RC values.

$$R \times C = t \quad \therefore \quad 10k\Omega \times 10\mu F = 100ms (0.1s)$$

The main function of the reset circuit is to provide an automatic Power-On-Reset to the microcontroller. Upon power-up, the reset pin is initially high since C3 is not at its maximum charge, hence the microcontroller would be in a reset state. This provides the required time for the voltage to stabilise before execution of the source code could start from the beginning of the program. To manually reset the microcontroller during mid-operation, switch SW1 is pressed.

4.2 - LCD Pin Configuration

Table 1 shows a brief description of the LCD pin-out connections.

Table 1 - Pin Description of 1602 Hitachi HD44780 Based LCD.

| Pin | Symbol | I/O | Description |
|------|--------|-----|---|
| 1 | Vss | - | Power supply (GND) |
| 2 | Vdd | - | Power supply (+5V) |
| 3 | Vo/Vee | - | Contrast adjust voltage; through a variable resistor. |
| 4 | RS | I | Register Select control pin - set low to input a command, set high to input a character. |
| 5 | R/W | I | Read/Write control pin - set low to write data to the LCD, set high to read data from LCD. |
| 6 | E | I/O | Enable control pin - toggle with a high to low pulse to load the data and configuration into the LCD. |
| 7-14 | D0-D7 | I/O | Data lanes. D0 - Bit 0 (LSB), D7 - Bit 7 (MSB). |
| 15 | A | - | Backlight (+5V) |
| 16 | K | - | Backlight (GND) |

Vdd, Vee and Vo

The LCD and backlight is powered on with a supply voltage of +5V when connections to Vdd, Vee and their respective ground is made. A 10k Pot resistor is used to vary the voltage potential on Vo, thus adjusting the contrast level of the displayed characters. For the best contrast level, the pot is set to a 1k pre-set position resistance value.

RS (Register Select)

The RS pin was connected to P1.1 of the microcontroller so the data type received by the LCD can be controlled. RS has the role of selecting from two different registers as follows. If the RS pin is given a value of 0, the instruction command code register is selected, thus the LCD interprets the data as a command. If the RS pin is given a value of 1, the data register is selected, thus the LCD interprets the data as ASCII character data.

R/W (Read/Write)

The R/W pin was connected to P1.2 of the microcontroller so the dataflow direction between each device can be determined. The LCD's memory can be written with data when the R/W pin is given a value of 0. The microcontroller can read the LCD's memory when the R/W pin is given a value of 1.

E (Enable)

The enable pin was connected to P1.3 of the microcontroller so the LCD can latch the data present on its data pins when a pulse is generated. A high to low pulse must be applied in order for the LCD to load the data and configuration of RS and R/W into memory. The enable pin is given a value of 1, then the pulse is delayed for a period of 1ms before a value of 0 is given. The delay is required since the pulse must be a minimum of 450ns wide for the latching to be effected.

D0 - D7

The data lanes D0 - D7 are connected in parallel to port 2 of the microcontroller so communication between each device can be made. The lanes are used by the microcontroller to send bits of data to the LCD or read the contents of the LCD's internal registers. Before any data can be written in the LCD's memory, D7 is used as a busy

flag to check if the LCD is currently busy processing commands. To read the busy flag, P2.7 on the microcontroller is connected to D7 and is set to High so it can act as an input. Following every command or data inputted in the LCD, RS is given a value of 0 and R/W a value of 1. D7 is then monitored by the microcontroller, waiting for the busy flag to be equal to 0. New information can only be received by the LCD when the condition is met, otherwise it is busy processing data.

4.3 - LCD Initialisation

The LCD can be configured in two different modes, 4-bit mode or 8-bit mode. The author made the decision of connecting the LCD in 8-bit mode since an efficient output is required for displaying time. The main purpose of connecting the LCD in 4-bit mode is to save valuable pins, although an impact on the LCD performance is resulted. Before an instruction could be received by the LCD, the microcontroller needs to divide the data into two four bit segments and then send them sequentially. When using 4-bit mode it therefore increases coding and keeps the microcontroller busy for a longer duration to send data, thus it could have the potential to keep updates of the 'second' segment from not being displayed on the LCD since the busy time will be longer than a one second period .

With reference to a flowchart (Appendices - Figure A.1) found in a Hitachi LCD datasheet, initialisation in 8-bit mode was prepared on Keil software so it can be exported onto Proteus and operate the LCD. On every power-up of the circuit, a delay is given to the microcontroller so the voltage would stabilise before executing commands. Before the LCD could be ready to receive the actual Function Set instruction, a procedure is repeated for 3 times with the necessary delay intervals so the LCD is given a reset. After the reset is performed, the actual Function Set instruction is given to the LCD so it would specify the interface, the number of lines and the font used. Subsequently, an instruction for specifying whether the display or the cursor are turned off/on is given. The display is then cleared and the last initialisation process command is given. The last instruction determines the entry mode, whether the cursor and/or the display shifts automatically to the left or right when entering a string of data.

Once the initialisation process is complete data can be written to the LCD memory. Before characters can be displayed on the LCD the cursor location needs to be set, command 0x80 sets the position of the cursor to the beginning of the first line while command 0xC0 sets the position to the beginning of the second line.

While referring to the 8051 technical book the microcontroller program was coded in C language. To reduce repetitive code when sending data to the LCD, methods have been used for the interpreting of data, thus two functions have been allocated for this purpose. The 'lcdcmd' function is used to set RS and R/W respectively so the data present on Port2 can be written into the LCD's memory and interpreted as a command. The 'lcddata' function configures RS with a value of 1 so the data present on Port2 can be interpreted as an ASCII character and hence is displayed on the LCD. Initially when compiling the program errors were given, although debugging procedures were followed the source of the problem wasn't found. Consultation with the lecturer made the author realise that the program was saved as (*.asm), a format used for Assembly language. Since the program is written in C language, saving it as (*.c) made the program work normally with no errors. After the author earned confidence with sending commands and displaying data on the LCD using Proteus, implementation of the real time clock commenced.

4.4 - RTC Chip Selection

After analysing various RTC chips, the author's selection range was cut down to three different ICs: DS1307 and PCF8583. The PCF8583 RTC was included in the selection range since a socket is specially allocated on the T89C51 training board for this specific chip. A CMOS battery holder is soldered to the board so a back-up power source can be present. A 32.768KHz crystal is also included so the RTC chip can be provided with the necessary oscillation for clock timing. All these features provide facility for implementation since the RTC chip requires only the fitting in its respective socket, without the need of adding extra components. The DS1307 was considered since it was used in a similar project, the project was studied earlier in section 2.2 where it has shown successful operation with the RTC.

Both DS1307 and PCF8583 make use of an I²C protocol to communicate with the microcontroller. The I²C interface is comprised of a 2-wire serial bus, the SCL (Serial Clock Input) is used to synchronize data movement on the serial interface while the SDA (Serial Data Input/output) is used for data input/output. A device that sends data onto the bus is defined as a transmitter while the device that receives the data is defined as a receiver. The device that controls the message is called a master while the devices that are controlled by the master are referred to as slaves, in this case the microcontroller acts as the master while the RTC acts as the slave. A maximum of two pins is only required by the microcontroller when an I²C protocol interface is used, thus letting valuable inputs/outputs to be utilised by other operations.

After consulting matters with the lecturer, the DS12887 RTC was recommended since it is easier to implement than I²C protocol based chips. The only drawback that was considered for the DS12887 RTC was that it occupies 12 pins from the microcontroller. Since sufficient I/Os are available for all the hardware to be interfaced, it is not an issue for the author's project. When compared to DS1307 and PCF8583 chips, the package is larger in size since a lithium battery and quartz crystal are built-in, thus additional components are not required for purchase. After analysing the DS1307 RTC from its datasheet, it was noted that the IC does not include alarm bytes or any interrupt pin output. The alarm is crucial for the project since the majority of the hardware functionality is triggered by the alarm. Although the alarm function can be implemented with software by constantly comparing the RTC time to the set alarm variables, it is an inefficient method since the microcontroller will be constantly executing an extra command. By having an interrupt pin output connected to the microcontroller and alarm bytes set with data, an interrupt could be generated once the alarm time matches the current time. When using this method, the microcontroller performs a sub routine only when a pulse is present on the external interrupt pin of the microcontroller. This concluded that the DS1307 RTC is not ideal for the project and thus the DS12887 RTC was used.

Besides choosing the DS12887 RTC for the recommendation made by the lecturer, it was preferred over the PCF8583 RTC for the reason that programming reference from the 8051 microcontroller book was more related to the chosen IC. By having a reliable source of reference for interfacing the RTC with the microcontroller, confidence is achieved for the component's functionality since it is proven. After the selection of the

RTC chip was made, implementation of the real time clock could commence. Since the required components to implement the real time clock were still not at hand, it was decided that the RTC should be tested on Proteus.

4.5 - RTC Testing on Proteus Software

Before implementing the RTC, the lecturer explained briefly how it can be interfaced. With reference to the 8051 microcontroller technical book and DS12C887 datasheet, the circuit shown in Figure 10 illustrates the initial connections made on Proteus for testing the RTC.

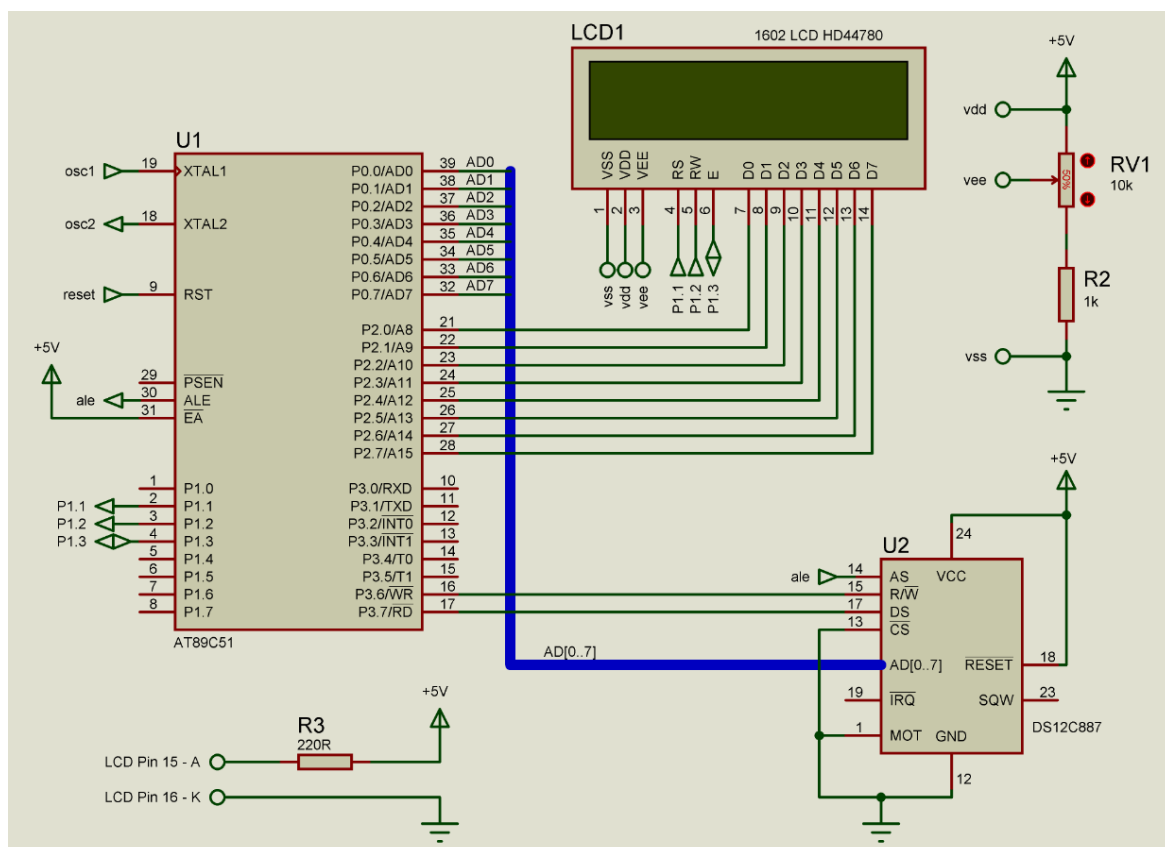


Figure 10 - Proteus RTC Connection.

Since an Atmel microcontroller is being used, MOT (Mode Select) is connected to ground to select Intel bus timing. Connection with CS (Chip Select) to ground has also been made so the chip is accessible when reading or writing data. R/W is connected to the WR pin of the microcontroller so latching of data from Port0 into RTC memory is

possible, DS is connected to the RD pin of the microcontroller to enable reading. AS (Address Strobe) is connected to the ALE pin of the microcontroller for de-multiplexing the address/data bus AD[0..7], also to latch addresses within the RTC. The RESET pin is connected to Vcc since it must be kept high for the device to be accessible. Pull-up resistors were not used with Port 0 since the RTC can operate with active-low signals.

After the author made the required connections with the microcontroller and had sufficient knowledge on how the RTC operates, further programming was made. Originally the code found on the 8051 microcontroller book for interfacing the RTC was programmed to send the time serially using RS232 and displays it on the PC screen through a special interface. Therefore, the coding had to be altered to suit the needs of the project for displaying the time on an LCD.

For accessing RTC registers and RAM, reference to the datasheet was made. The DS12C887 RTC address map was used to identify easily the address locations of registers and memory which keeps the RTC time. The time can be changed through the program on execution when the respective address location is loaded with a valid BCD value, using the command `XBYTE[address location] = value(Hex)`. Since the time of the RTC is saved in a BCD (Binary-Coded-Decimal) format, it needs to be converted into ASCII before it can be displayed on the LCD. The following code shows the function 'bcdconv' used for converting data from BCD to ASCII.

```
void bcdconv(unsigned time)
{
    //convert BCD to ASCII and send it to lcddata function for display
    unsigned char x,y;
    x=time&0x0F;
    x=x|0x30;
    y=time&0xF0;
    y=y>>4;
    y=y|0x30;
    lcddata(y);
    lcddata(x);
}
```

If for example a variable holding the 'minutes' segment in BCD format needs to be displayed on the LCD, when the function is called from the main program, the function accepts the loaded variable for execution. Since the LCD accepts one character at a time, the variable needs to be divided into upper and lower 4 bits with masking procedures. Two variables are initialised within the method, one variable is used to represent the upper part of the segment while the other represents the lower part. For

the lower part, the upper nibble is disabled by giving a value of 0FH, the resulted value is then combined to 30H so it can be interpreted as ASCII. For the upper part, the lower nibble is disabled by giving a value of F0H, the resulted value is then shifted right to the lower 4 bits and is then combined to 30H. The two variables are then displayed sequentially on the LCD when processed by the 'lcddata' function since the LCD accepts only one character at a time.

The first RTC program compiled was intended to constantly read the BCD hours, minutes and seconds from the RTC, convert the values to ASCII and display each segment sequentially on the LCD. When every segment is displayed, the LCD cursor is set to its starting position so it can overwrite any characters with an updated time. The process flow of the program can be explained more clearly from the flowchart.

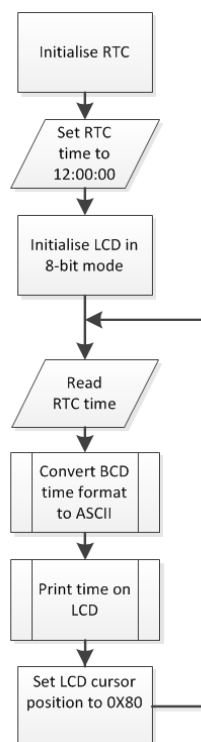


Figure 11 - LCD Time Display Flowchart.

When the program was tested on Proteus with the LCD, the time was not displayed correctly as expected. Instead of printing the hours and minutes in their respected segment, the seconds were garbled throughout the whole segments. For testing purposes, Proteus shows a small window with the RTC time so it could be compared with the LCD display. Since the time shown in the window was displayed correctly, it

was concluded that the RTC was functioning correctly and thus the source of the problem was coming from the LCD.

After setting time delay between each command given to the LCD, initially the time was displayed correctly but when the minutes are updated with a new value, the same problem persisted with displaying the seconds instead. This alteration fixed a minor problem and thus the author asserted that with the correct delay time, the time can be displayed correctly. After referring to the 8051 microcontroller book, a related solution for solving the problem was found. The LCD has a special function which allows the delay time to be set more efficiently. As the busy flag was already discussed in section 4.2, it can be used to utilise more efficiently the processing time of the microcontroller than normal preset delays. Since every command received by the LCD has a different processing time, the busy flag can be used as a dynamic delay and thus be more responsive. When using normal preset delays, if the LCD finishes earlier from processing a command, the microcontroller still has to finish from the delay subroutine to continue executing the program.

After adding a busy flag function to the program, the simulation worked as expected and no garbled data was displayed on the LCD. Since the time was shown correctly on the LCD, further defects could be detected. During the simulation it was noted that the time displayed on the LCD was not oscillating with a correct frequency, a normal time interval of 1 second was taking longer in real time. Consultation with the lecturer was made regarding the problem and it was explained that the operation is normal with the simulation, thus it should be tested on hardware.

Although the program worked fine, the author made an observation and it was noted that the process flow of the program was not efficient. For the majority of the time, the hour and minutes segment is printed with a same value, thus unnecessary processing is resulted; for this case the seconds segment is excluded since it is updated regularly.

For this reason, the process flow was given a new structure. At the start of the program all segments are displayed normally but when the seconds segment is reached the program falls into a loop and the cursor is constantly positioned to the beginning of the seconds segment, thus printing only the seconds on every update. When the seconds reach the end of the 59th second and reset back to 0, the program exits the loop to update the minutes. In the same process if the minutes reset back to 0, the hours are

also updated by positioning the cursor at the beginning of the respective segment and printing the new value, subsequently the program falls again in the loop to update the seconds. For flowcharts and program code refer to Appendices 1 and 2 respectively.

At this point in time, all required components to implement the RTC were at hand, After making final changes on the structure of the program, hardware implementation has initiated immediately on the Hitachi HD44780 based LCD so the real time clock could be actuated physically. The author decided that the ideal procedure for implementing the RTC is to divide the implementation into stages, the first task was to display a set of characters on the LCD.

4.6 - Implementation of LCD Module

A PCB header pin rail was cut down to 16 pins to fit the LCD's pin-out holes. Soldering was made to hold the rail firmly in place while providing individual connection to every output/input of the LCD. During the testing phase, A bread-board was used to provide the LCD with external connection to hardware. Further soldering wasn't required since the LCD could be connected with components by simply fitting the 16 pin header in the respective slots on the breadboard, as shown in Figure 12.



Figure 12 - LCD connected on breadboard.

Before setting up the LCD on the breadboard, it was noted that the training board provides a direct connection to the microcontroller through a 14pin IDC header, also a trim pot is included for adjusting the brightness of the display. The board's manual was used to further read about this connection, it was concluded that the socket is irrelevant for the author since it is set up for 4-bit transfer only.

The T89C51 board was connected to the breadboard through the 34pin IDC header. An IDC header adapter was inserted in the breadboard so connection with the board can be made via a cable. This facilitated the work since the T89C51 can be connected or disconnected from the board with ease and less time is wasted than using separate wiring .

When connection to the LCD was made from the respective pins on the IDC header of the breadboard, the microcontroller was flashed with the equivalent program tested on Proteus for displaying a string of text. The connections made between the LCD is equivalent to the circuit design of Proteus as shown in Figure 8. All the programming and implementation performed on software was applied physically so similar results from the simulation could be obtained. Since the simulation on Proteus worked fine, the author expected that the same results are obtained from the hardware.

When power was applied to the circuit through the plug pack connection of the training board, the LCD wasn't displaying any text. The circuit was switched off and a multi-meter was used to test for continuity between the LCD and microcontroller pins. After a considerable amount of time checking if the connections between the LCD are connected correctly, it was noted that the potentiometer was set to its maximum preset value and thus the contrast level was not ideal for displaying characters. When the potentiometer was set to its mid position, the expected set of characters were displayed correctly. Since the task to display a string of text on the LCD was finished, the RTC was implemented so the second task to display the time could commence.

4.7 - Implementation of RTC

As a general remark, the DS12C887 was bought since the DS12887 chip was not available for purchase. As read from the datasheet, the chips have minimal differences since the DS12C887 chip was designed as a direct upgrade replacement and a century byte was added to the RTC time.

For the implementation of the RTC, the chip was fitted on the breadboard and connection with the microcontroller was made. Connection with the RTC chip was made equivalent to the circuit design of Proteus as shown in Figure 10 so similar results from the simulation are achieved. Since the IDC header of the board does not provide

connection to the ALE pin of the microcontroller, a wire had to be soldered directly from the board and connected on the breadboard with the AS pin of the RTC. To find the connection on the board, the T89C51 datasheets was used for locating the pin on the microcontroller, a multi-meter was then used to locate the respective connection on the board by testing for continuity with the pin.

When everything was settled in place and the microcontroller was flashed with the RTC program, the circuit was switched on. Although the time was read and displayed correctly on the LCD, the time was not updating. The circuit was switched off and the same procedure done on the LCD for testing each connection was performed. Although a considerable amount of time was spent testing each connection and making sure that every component was in place and connected correctly, the time remained stuck with the same set value.

When reference to the DS12C887 datasheet was made, it was explained that when the chip is shipped from the factory, the internal oscillator is turned off. Although control register A was already set with the appropriate value for turning on the oscillator, the time wasn't updating. The problem daunted the author for the reason that when using the same program for both the simulation and the actual circuit, the time displayed on Proteus was oscillating while in practice it kept displaying the same time value. The author reflected that the RTC chip was defective, although after analysing the problem it was concluded that if the RTC chip was faulty, it wouldn't display the time at all. Further attempts of setting time delays between RTC commands given to the microcontroller were made, although the same problem persisted

Consultation with the lecturer was made and it was suggested to set the AUXR register. When reference to the T89C51AC2 microcontroller datasheet was done, it was noted that if the AUXR register is initialised within the program, the ALE pin is enabled hence latching of address outputs at Port 0 is authorized. When the program was executed with the new configuration, the time displayed on the LCD started updating normally with a correct oscillation frequency. For the time to be read in a 24-hour format, register B (address location 11) was initialised with the appropriate values. A result of the operation is shown in Figure 13.

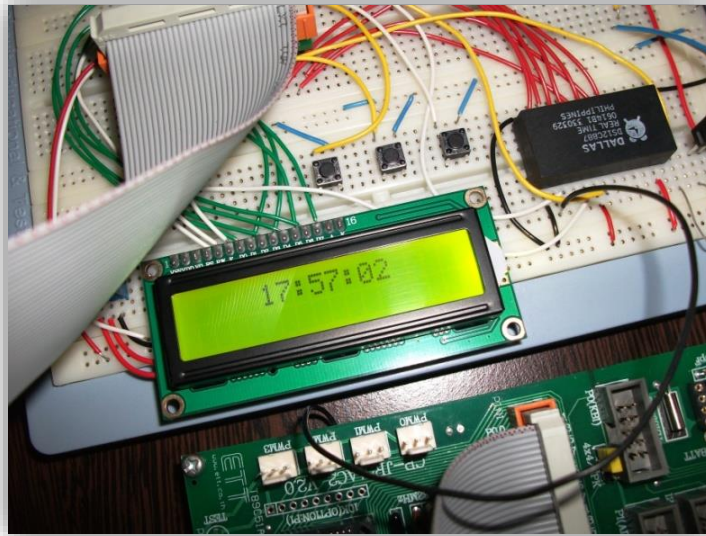


Figure 13 - Time being displayed on LCD.

Since the RTC is a crucial part for the functionality of the hardware, it was estimated that 50% of the project was finished at this stage for completing the second task. Subsequently the author decided to operate the RF modules since they require time to manage, also if wireless communication results successfully the majority of the project will be finished. At this point in time, research on RF modules and wireless communication was made to search for solutions and learn how to interface.

4.8 - Serial Communication & RF Solutions

The initial plan for wireless communication was to use an HT12E encoder and a HT12D decoder, having either a TWS-434 transmitter or an RWS-434 receiver connected to the respective IC. The idea was to transfer parallel bits of data from the base station microcontroller to the encoder, the encoder will then convert these parallel signals into serial bits so the data can be read by the transmitter. Upon receiving serial data from the encoder, the data is transmitted wirelessly and picked up by the receiver at the blinds opener module. Upon receiving, the data is fed serially to the decoder and is converted into parallel data so it can be read by the end microcontroller.

Although a solution was set, they weren't available for purchase neither locally or overseas, also a reliable source which proves their functionality couldn't be found.

Consultation with the lecturer was made and it was advised to use a pair of Easy Bee communication boards instead. The boards were borrowed from the lecturer since they were available and generously have been provided. Confidence was achieved since the boards were already used in students' projects and worked successfully. After further consultation with the lecturer on how the boards can be interfaced and configured, testing and implementation of the RF modules initiated immediately. A sample program was also provided to use as reference at a later stage.

The boards are based on Atmel ATZB-24-A2 ZigBit modules and can be programmed with SerialNet AT-command language through a serial interface (UART). Since the RF modules are designed as transceiver units, their role within the network can freely be configured either for transmitting or receiving data. A supply select pin header is also provided to choose between +3.3V and +5V input. The lecturer's advise was to use HyperTerminal software as a serial interface, also to test serial communication between the microcontroller before actually using the RF modules.

The first task was to send a string of text serially from the microcontroller to the computer's serial port and display it on HyperTerminal. Since the PC does not have a serial port a USB serial adaptor was used, connection to the microcontroller was made through the RS232 socket using a 4 pin polarized connector. A MAX232 level conversion IC is already included on the board, it is used to convert +12V RS232 signals to +5V level signals suitable for the microcontroller and vice-versa.

Since the initial RTC program found on the 8051 microcontroller book was intended to transfer the time serially and display it on the PC screen, it was used as a reference to understand how serial reception works. To enable serial reception specific microcontroller registers should be set with appropriate values, this is shown in the following code fragment.

```
SCON=0x50;           //enables serial reception in 8-bit mode
TMOD=0x20;           //Timer 1
TH1=0xFD;            //9600 baud rate
TR1=1;               //turn on timer 1
```

Reference from the AT89C51 datasheet was made to understand more clearly what each register does. The serial control register SCON is given a value of 50 Hex to enable serial reception in 8-bit mode. TMOD is given a value of 20 Hex to select Mode 2

(8-bit auto reload) for Timer 1. The baud rate determines the data transmission speed, it can be set by assigning the TH1 register with a suitable value, in this case a value of FD Hex sets the baud rate to 9600. TR1 is located in the TCON register and is set to turn on Timer 1, hence enabling the serial clock.

For transmitting data, the serial data buffer register SBUF is loaded with a single character and the transmit interrupt flag TI is monitored until it is equal to 1. With a value of 1, it indicates that the character has been transmitted. Consequently, TI should be cleared so SBUF can accept further characters for transmission and the flag can be monitored. To avoid repetitive code when sending a string of characters, the following function 'tx_data' has been implemented within the program.

```
void tx_data (unsigned char x[])
{
    //Transfer serially a string of data
    int pnt=0;           //array pointer

    while(x[pnt] != '\0')
    {
        SBUF = x[pnt];           // load SBUF with char data for transmitting
        while(TI==0);           // Wait for byte to be transmitted
        TI=0;                   // Reset the transfer interrupt flag
        pnt++;                  // increment array pointer
    }
}
```

When the function is called from the main program, the accepted data string is stored within an array. If for example the word 'Hello' is processed, H will be found in location 0 of the array, thus subsequent characters are placed in incremented locations. Variable 'pnt' is assigned a value of 0 so it can initially point to x[0]. On every cycle of the loop, the character found within the pointed location of the array is transmitted and 'pnt' is incremented so it can point to the subsequent location. The while loop is performed until a NULL value is found hence the end of the string will be reached.

Once the author was finished from writing the program, HyperTerminal was used to test its functionality. The software baud rate was set to 9600 so it matches the microcontroller configuration, also Flow control was set to 'None' and other settings were kept to their default value as instructed. The microcontroller program was intended to transfer the word 'Hello' to the PC screen but when the code was executed a string of symbols was displayed instead. After checking if the microcontroller was set correctly, it was noted that the computer's COM port was configured with an incorrect baud rate

therefore it was set to 9600 Bd. Although everything was set to the same baud rate, symbols were still displayed. The lecturer's advice was to use a baud rate of 38400, also to use the sample program provided earlier as reference. Within the sample program it was observed that Timer 2 has been used, the AT89C51 datasheet explains that its operation is similar to Timer 0 and Timer 1, however additional enhancements are included to provide an efficient output. The fundamental part of the program has been located to explain more clearly the new configuration.

```
SCON=0x50;  
T2MOD=0;  
T2CON=0x30;  
RCAP2H=0xff;  
RCAP2L=0xf1;  
TR2=1;
```

SCON is configured with the same value as before to enable serial reception in 8-bit mode. T2MOD is cleared to program P1.0/T2 as clock input and disable Timer 2 as up/down counter. T2CON is loaded with 30 Hex to use Timer 2 as receive/transmit clock for the serial port. RCAP2H and RCAP2L determine the clock-out frequency and baud rate of the operation, in this case it is configured to 38400 Bd. TR2 is located in the T2CON register and is set to turn on Timer 2, hence enabling the serial clock.

With the new configuration, the word 'Hello' was displayed correctly. Since transmission from the microcontroller was working fine, reception could be tested. The method used for receiving characters is similar to the transmitting method, instead of loading SBUF with data, the receive interrupt flag RI should be initially monitored. When a value of 1 is detected, it indicates that data has been received and thus reading SBUF retrieves the character. Consequently, RI should be cleared so SBUF can store new characters and the flag can be monitored.

To test reception, an LED was connect to a microcontroller pin output. The LED was configured to act as an indicator, hence lighting up when a character is received. HyperTerminal was set with the same baud rate value and minor settings have been changed to allow printing of characters. The execution of the program worked as expected, when a key was pressed from the keyboard, the microcontroller turned on the LED for a short period of time to indicate that data has been received. Since both tasks were completed, the RF modules could be interfaced with the microcontroller. As it can

be observed, HyperTerminal was used as a testing tool for setting the correct configuration before putting the RF modules into practice.

4.9 - Configuration of RF Modules

The ATZB-24-A2 datasheet and the Atmel AVR2051 user guide were used to get familiar with UART interfacing and Serial Net AT-command language before configuration can be made. It was decided to use HyperTerminal for configuring the RF modules rather than adding AT-commands to the microcontroller source code. Since the configuration is saved into memory, the commands require to be inputted only once. The RF modules were connected to the PC through a USB-to-UART serial converter board which was also provided by the lecturer, wires were soldered to the RF modules' UART and connection to the USB board was made in turns. As advised by the lecturer, TX pin output of the EasyBee board was connected to the RX pin on the USB board while RX was connected to TX. Figure 14 shows the connection made with the device, a USB cable was used to connect to the PC.

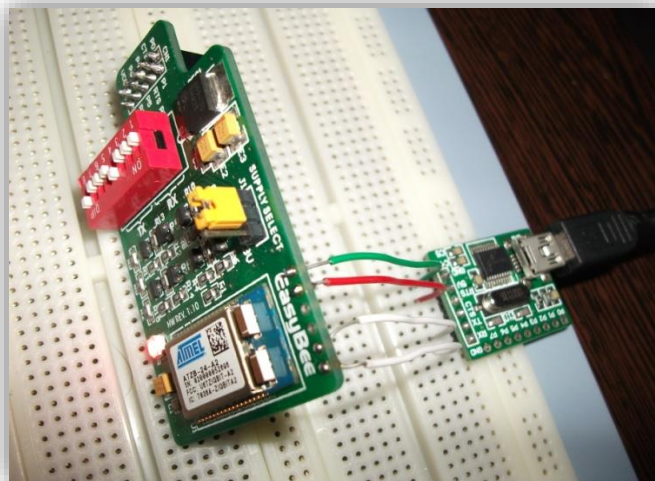
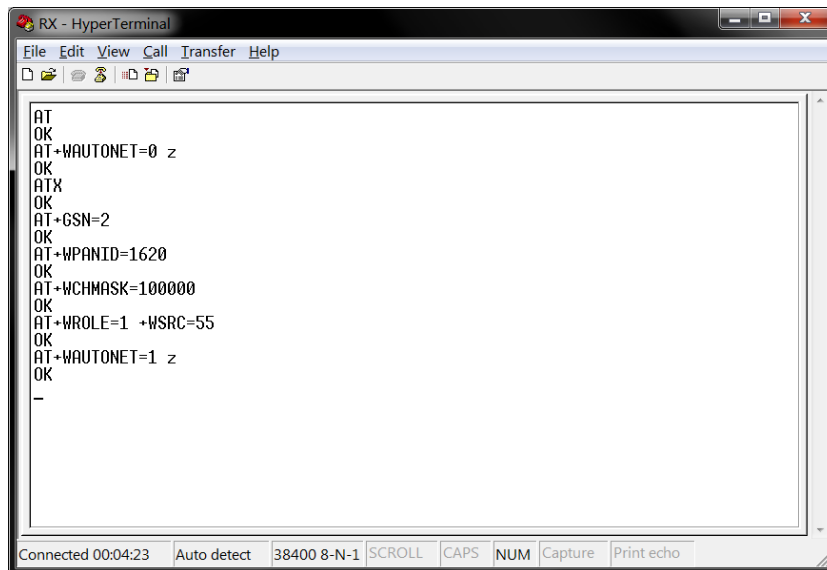


Figure 14 - EasyBee connected with USB UART 2 Board.

After configuring HyperTerminal with the appropriate settings, commands could be inputted via the keyboard. With reference to the AVR2051 user guide, AT-commands

were inputted to configure one module as a router, hence preparing the node for receiving data. The commands inputted on HyperTerminal are shown in Figure 15.



```
AT
OK
AT+WAUTONET=0 z
OK
ATX
OK
AT+GSN=2
OK
AT+WPNID=1620
OK
AT+WCHMASK=100000
OK
AT+WROLE=1 +WSRC=55
OK
AT+WAUTONET=1 z
OK
-
```

Connected 00:04:23 Auto detect 38400 8-N-1 SCROLL CAPS NUM Capture Print echo

Figure 15 - HyperTerminal interface showing typed commands for configuring RF module as router.

The command 'AT' has been entered to check if communication between the host and RF module has been established, the OK result code is returned to indicate that the command was executed successfully. 'AT+WAUTONET' is inputted with a value of zero to disable automatic network, subsequent commands cannot be accepted when the node is joined to a network. The 'ATX' command sets the node to transmit EVENT and DATA to a host, further commands set the node's MAC address, PAN ID and channel mask. 'AT+WROLE' sets the node role, in this case it is configured as a router and a unique network address of 55 is given. Automatic joining to the network has been enabled so the node can connect automatically to another node within the same network range when both devices are active. This method eliminates the need for inputting 'AT+WJOIN' every time the circuit is turned on.

The second RF module was configured as a network coordinator, hence having the role to transmit data. Using the same commands different addresses were assigned to the RF module so networking can be possible, the MAC address was set to 1 while the network address was set to 0.

4.10 - Testing of RF Modules

To test the RF modules for communication, the transmitter was left connected to the PC while HyperTerminal was still running in the background. Direct connection to the receiver's UART was made from the microcontroller TxD and RxD pin-outs so the received data can be processed. Once the receiving end circuit was turned on, an 'EVENT:JOINED' message was displayed on HyperTerminal to indicate that communication between the RF modules has been established. The program used earlier for testing reception was utilised, the LED connected to the microcontroller should light up when data is transferred through RX and stored in SBUF.

When the command 'ATD55' was entered on HyperTerminal followed by a string of data "Hello", an OK message was displayed to indicate that the data has been sent successfully to node 55. Although the data has been picked up by the receiver, the output LED did not light up. The author concluded that the source of the problem is located between the microcontroller and the receiver since reception at SBUF is not positive. The conclusion was made for the reason that if the transmitted data was not received by the RF module, an 'ERROR' message would have been displayed instead. After making changes to the microcontroller's program by implementing delays and initialising Timer 2 with different values, the problem still persisted. With reference to the ATZB-24-A2 datasheet it was noted that connections between the RF module and the microcontroller UART were swapped, hence correct wiring between TxD and RxD was made shortly. Figure 16 shows the correct UART connection between the two devices, RTS and CTS were left disconnected since hardware flow control is not required. With the new configuration, the LED indicated successful reception between the microcontroller and RF module when data was received.

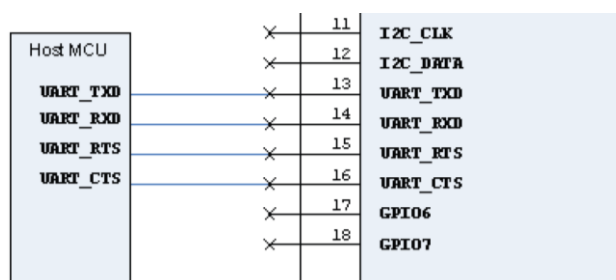


Figure 16 - UART Connection Between RF module and MCU (ZigBit ATZB-24-A2 Datasheet).

On completion of the task, the author concluded that the transceiver boards have given a better outcome for the execution of the project than it was expected. When comparing the results to the initial idea of using encoders/decoders, advantages of the recent operation can be mentioned. The transceiver boards have better communication expansions since they can freely be configured to transmit or receive data during program execution, thus having a two-way communication. Also, a more reliable data transmission is provided since they can respond with an OK or ERROR message, giving indication whether the data has been received successfully. Apart from using a High-level language to configure the boards, a range of commands is available for setting the RF modules to suite a specific scenario. Only two pins are occupied from the microcontroller, also additional components are not required for the interfacing of the boards however their price range is still relatively expensive.

4.11 - Alarm Clock Configuration

At the current stage a specific element which is important for the functionality of the project was still not configured, the alarm clock function is crucial for the project since the majority of the hardware is dependent on the event. For the implementation of the alarm, reference with the 8051 microcontroller book and RTC datasheet was made. The alarm time can be set likewise as the time by filling the respective memory locations with valid BCD values. From the datasheet it was explained that when the alarm time matches the current time, a signal is generated from the IRQ pin output of the RTC. Since the microcontroller can handle external interrupts, it was decided to connect IRQ to INT0 input on the microcontroller so it can react to alarm events.

When an alarm interrupt is generated, the microcontroller stops execution from the main program to handle the interrupt sub routine instead since it is given a higher priority. Once it finishes processing the sub routine, the microcontroller continues execution of the main program from the line where it was interrupted. For the signal to be permitted by the RTC, register B (address location 11) should be given appropriate values to set

the Alarm Interrupt Enable bit. Also, the microcontroller's IEN0 register should be set to enable external interrupt 0.

To test the program an LED was connected to a microcontroller pin output so indication for alarm events can be provided. When the circuit was turned on and the LCD time display matched with the set alarm time, the LED responded correctly to the event however it was noted that the time stopped from updating. After further reference with the datasheet, the author concluded that the program was executing the interrupt routine continuously since the INT0 pin was kept active from IRQ. It was explained that when the alarm interrupt occurs, the RTC flag bit AF will be set to 1. To exclude the IRQ signal, AF should be cleared by grounding the RESET pin. To control grounding during program execution, the pin was connected to an output on the microcontroller. During normal operation the pin was set high so the RTC is accessible, at the start of the interrupt routine the pin is set low (grounded) so the Alarm Interrupt Flag is cleared and thus interrupt signal is excluded. With the new configuration in place, when the current time matched with the alarm time, the LED lit up and the LCD time display kept updating.

For the user to set the time and alarm during program execution an interface had to be developed. The initial idea was to use a keypad, although it was concluded that the microcontroller did not offer sufficient ports for the interfacing to be possible since it requires a minimum of eight free pin-outs. An efficient method was designed which can provide the same level of usability, as shown in Figure 13 a row of three momentary push was introduced. When the left button is pressed (CYCLE), it is programmed to present the LCD with a menu for choosing between setting of time or alarm, also it is used to cycle between the minute and hour time segments. The middle and right buttons (DEC/1 and INC/2) are used to select the respective option from the menu, also they are programmed to increment and decrement the minute and hour time segments. Subsequently whenever a new time is set from the interface, it is written into RTC memory and the LCD time is updated.

During a power-outage the RTC will normally keep oscillating, hence the LCD will still display the actual time when the circuit is switched on at a later stage. Since the RTC uses a non-volatile memory, the alarm time and configuration will remain saved during the absence of power. In a typical scenario the time is only changed twice a year for

daylight savings and the alarm time is normally changed for the weekends when the user does not have early appointments. With this assumption it can be concluded that the keypad is not crucial for the operation of the project since the feature for changing the time is not intended for a daily-basis. As a whole it only provides facility for setting the alarm time faster, although the final hardware enclosure will be larger in size to hold the keypad.

4.12 - Motor Operation

Table 2 - DC Gear Motor Specifications.

| | | | | | |
|----------------------|---------------|---------------------------------|------|-----------------------|-----|
| Voltage Input | 12V DC | Weight | 280g | Shaft diameter | 6mm |
| Amperes | 200mA | Diameter | 37mm | | |
| Torque | 40Ncm - 4kgcm | Length (excluding shaft) | 85mm | | |
| Speed | 2RPM | Shaft length | 15mm | | |

The selection of the motor was already discussed in section 2.2, thus specifications of the motor are listed in Table 2. To make further justifications on the selection of the motor, a stepper motor was not chosen since precise positioning control is not required when opening/closing the blinds. Normal DC motors are used in high speed operations however for their fast rotation less torque is provided, hence the overall operation is contrary to the project's requirements. As a conclusion, a DC gear motor was chosen since it offers high torque at low speed. When taking into consideration the torque of the selected motor, it was concluded that the component is more powerful than the requirements needed for driving a small design model. The motor was still used in the project since the author felt more confident working with a higher torque value, giving assurance that the motor won't stall during operation. A motor with a slow speed of 2RPM was chosen to suite the concept of the project so the blinds will open gradually and let more light enter over time. The initial idea for controlling the rotation of the motor was to use relays, Figure 17 shows an early design of the circuit which makes part of the receiving end.

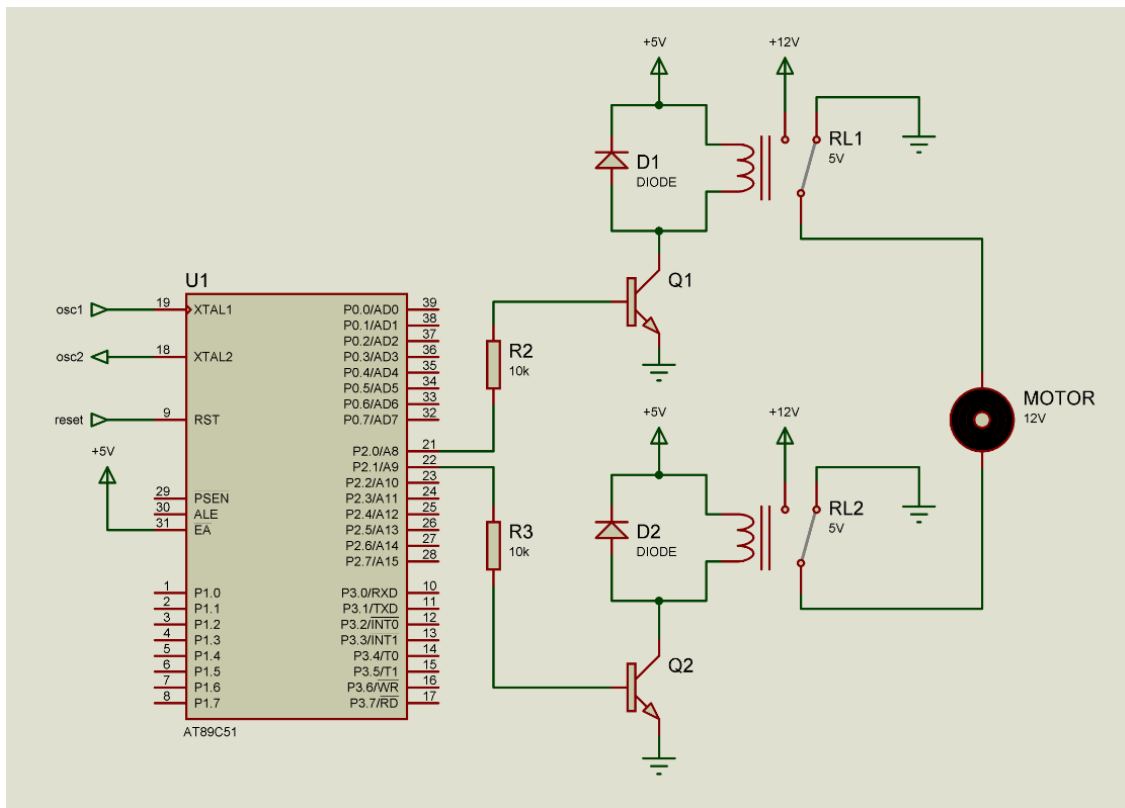


Figure 17 - Motor Direction Control using Relays.

The theory behind the circuit is to control the rotational direction of the motor by applying a high logical output on the base of a transistor so current can flow through the respective relay. When a relay is activated a voltage of +12V will be applied to the respective pole on the motor, hence rotating the motor in a clockwise or anti-clockwise motion. The diodes which are connected in parallel to the relay are used to filter unwanted EMF, resistors R2 and R3 intend to limit the base voltage of the respective transistor.

When analysing the circuit it was concluded that the same operation can be implemented with a simpler and efficient method using only four transistors to compose an H-Bridge. After further research on components the final decision was taken, the L293D chip integrates the same function in a single package. It was chosen for the implementation of the motor since it offers a compact design and easy operation, also it is relatively cheap and efficient for its level of utility. The chip has driving capabilities for two DC motors and is designed to provide bidirectional drive currents up to 600mA at voltages from 4.5V to 36V. Since the selected motor operates at a voltage of +12V and

maximum current of 200mA, the L293D H-Bridge driver is suitable for the operation. Figure 18 show the initial connection made on the breadboard for testing the motor.

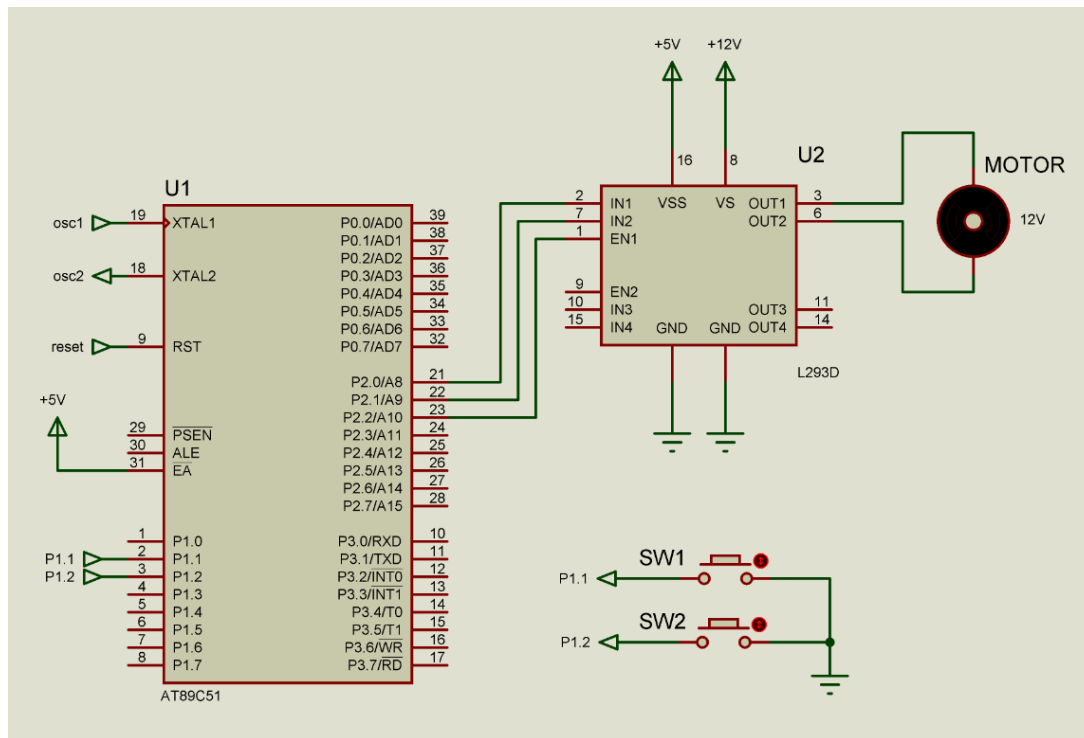


Figure 18 - Motor Direction Control using L293D.

VSS was connected to +5V for the logical operation of the driver, a voltage of +12V was applied on VS to power the motor. With reference to the L293D datasheet a number of three functions where programmed, one for rotating the motor clockwise, anti-clockwise and one for stopping the motor from rotation. A truth table has been added to show the possible logic levels which could be applied to the driver inputs for performing the corresponding function, 'X' has been assigned for irrelevant data. As it can be observed the motor can be stopped by inputting one of the three different combinations, hence the once on the right where not used for the operation since they perform the same function.

Table 3 - Truth Table of L293D for Motor Direction Control.

| EN1 | IN1 | IN2 | Motor Activity | EN1 | IN1 | IN2 | Motor Activity |
|-----|-----|-----|-----------------------|-----|-----|-----|----------------|
| 0 | X | X | Stop/Break | 1 | 0 | 0 | Stop/Break |
| 1 | 0 | 1 | Rotate Clockwise | 1 | 1 | 1 | Stop/Break |
| 1 | 1 | 0 | Rotate Anti-Clockwise | | | | |

To test the operation of the motor, two momentary push buttons were used, connection to the microcontroller's pin inputs was made as shown in Figure 18. When SW1 button was pressed, the first function was called from the main program and the necessary logic levels were applied to the driver. Since the EN1 pin was set high, the L293D outputs were enabled and thus the motor could rotate in a clockwise motion. When SW2 button was pressed the function for stopping the motor was called and the EN1 pin was set low, hence the driver outputs were disabled and as a result the motor was halted from operation. According to the set program, after a short delay the second function for anti-clockwise rotation is called. The program execution worked as expected however it was only intended for testing the H-Bridge driver functions and motor operation.

A different technique is required when applying the motor functionality to the blinds, the motor should be stopped from operation upon closing or opening the blinds completely so it won't over rotate and waste energy or cause damage to the structure. After analysing different methods for the operation, one possible solution was found. A vertical rail could be used to guide the fabric of the roller blinds horizontally, with the aid of two sensors the end and beginning of the fabric can be detected within the rail. At an earlier stage research on sensors was already made and it was concluded that photo interrupter sensors are the most suitable for the project. An infrared emitter and a shielded infrared detector compose the sensor. A gap is situated between the emitter and detector so when an object passes through the passage and breaks the light beam, hence the microcontroller can be used to detect the event. Figure 19 shows the connection made on the breadboard.

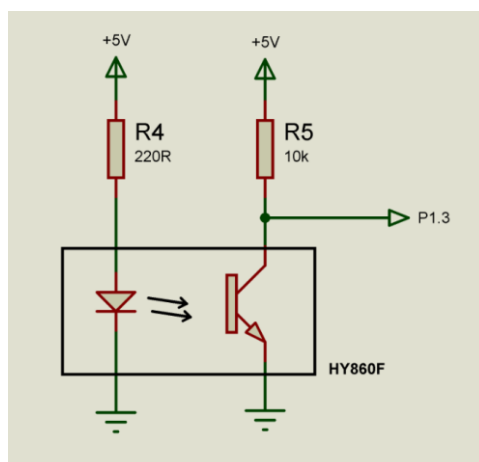


Figure 19 - Opto-Sensor Circuitry.

With reference to the HY860F datasheet connection to P1.3 of the microcontroller was made from the transistor's collector pin. Since the sensor operates with the same basics as a switch, the microcontroller pin was set high to act as an input. When an object is situated between the gap, the beam will be broken and thus no current will flow to ground through the transistor, hence the microcontroller input pin will read logic high by the pull-up resistor R5. Initially when the 10k resistor was not used, the microcontroller was not reading the input. When the object is removed from the gap, the transistor will start conducting and thus current will flow to ground, hence the microcontroller pin will read logic low. The 220 ohm resistor R4 is used to limit the voltage of the diode.

The sensor's logic was tested with the motor so when an object is located within the gap it stops rotation. The program used for testing initially sets the motor for constant rotation, subsequently the microcontroller monitors the sensor at a constant rate and once the light beam is interrupted, the function for stopping the motor is called. When the object is removed, the function for rotating the motor is executed.

When the circuit was tested for the first time, the motor did not rotate. At the current stage an unexpected-event occurred. The author measured the voltages present on the H-Bridge driver pins with a multi-meter expecting the readings of +5V on VSS and +12V on VS, however while positioning the probes, accidental short between the leads caused over voltage within the circuit. From earlier testing the EasyBee board was still connected on the breadboard, hence the chip was damaged instantly by the overvoltage. The conclusion was made since the power LED indicator of the board was lighting dim, also when HyperTerminal was used to input commands, the RF module was not replying with neither 'OK' or 'ERROR' messages. Also, the T89C51 board began to work erratically however it was fixed by resetting the microcontroller with a new program. This event delayed the process since re-ordering of a new EasyBee board was required, also minor components were replaced to avoid unnecessary troubleshooting from being damaged. Since the RF modules were already operated at an earlier stage, some components were still not functional, thus it was decided to continue with the troubleshooting process of the motor circuit since the ordered part was not crucial at the current stage.

When the circuit was re-configured with the same hardware, the program tested earlier was used to troubleshoot further the problem since the motor was still not rotating after

initialisation. After referring to the program and analysing the process flow, It was concluded that the sensor's logic was implemented with a reverse aspect, hence when an object was placed within the gap of the sensor, the motor started rotating.

4.13 - Alarm Clock Implementation

The final challenge for the operation of the project was the music player device, it was implemented to ensure wake up for the user when the alarm event occurs. The concept of filling the room gradually with light is only designed to prepare the user's mind to wake up, putting it into a more conscious state before activating the actual alarm. This method is more practical than waking the user directly from deep sleep, by which morning stress is triggered.

The lecturer's advise was to use an existing product and configure its circuitry so connection with the microcontroller can be made. After inspecting the market for an adequate product a conclusion was made, a low cost device which can demonstrate the project's concept effectively was chosen. The product includes slots for USB drives and SD cards to load music, also an FM radio is included. The use of music was preferred over the radio function since it can give the user choice for loading desirable songs rather than listening to a radio station which has the tendency to be distorted in bad weather.

When the device was purchased, it was initially configured to be powered from a 3.7V, 850mAh built-in rechargeable lithium battery. Since the device can also be powered through a USB port, it was decided to remove the battery and provide +5V from a USB cable. From the device specifications it was noted that the maximum power output of the internal speaker is 3W. Since the microcontroller cannot provide enough power to operate the device directly from its pins, it was decided to use a transistor. Also, with the same transistor a switching function can be implemented so when the alarm event occurs, the microcontroller can provide base voltage to the transistor, hence allowing current to flow through the device. Connection with the microcontroller was made as shown in Figure 20, the 10K resistor was used to limit the voltage present on the base.

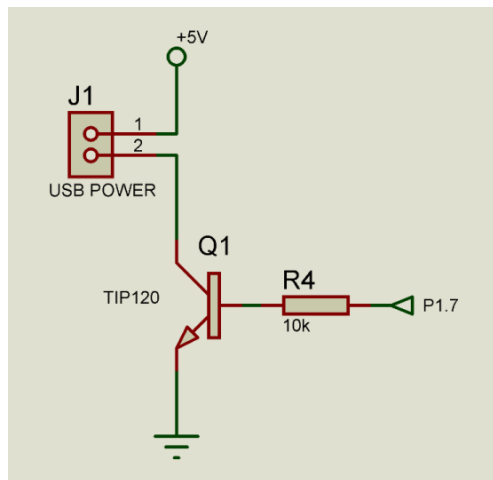


Figure 20 - USB Power Output Circuit.

Calculation for the maximum current input of the speaker was made so a transistor with a higher rating could be chosen.

$$P \div V = I \quad \therefore \quad 3W \div 5V = 0.6mA$$

Since a couple of transistors were already at hand, a TIP120 transistor was used for the switching function since it can handle up to 5A. Initially a 2N2222A transistor was chosen but at a later stage it was concluded that the component is not sufficient for the operation since it can handle an exact maximum value of 0.6mA, hence it can be damaged when operated at its maximum rating for a long period of time.

To test the circuit, the red wire of the USB cable was connected to pin1 of the jumper J1 and the black wire was connected to pin2, the other end of the cable was connected to the device. Initially a USB drive was used to load music, however when the circuit was turned on with the speaker set to its maximum pre-set volume, the device began to lose power when a high level of sound was generated. It was also noticed that the LCD brightness was flickering in the instance the device was turned on. For testing purposes the USB drive was replaced with an SD card because it draws less power, with the new configuration the device kept working however the LCD still worked erratically. Since the T89C51 board +5V regulator (L7805CV) can handle up to 1.5A, the source of the problem was taken into conclusion that is the AC-DC PSU which is limited at a maximum output current of 1A. The power is not sufficient to operate all the hardware when the device is set to its maximum volume, hence the author decided to limit the device power requirements by reducing the volume range.

Since time was a constraint, a 47ohm resistor was soldered in series to the speaker to limit the current input of the device, hence maintaining a stable operation of the circuit when the volume is set to its maximum. Different resistance values where tested until an adequate volume range was found. This method is not efficient since half the power is dissipated in heat, hence a 1W resistor was used to prevent damage.

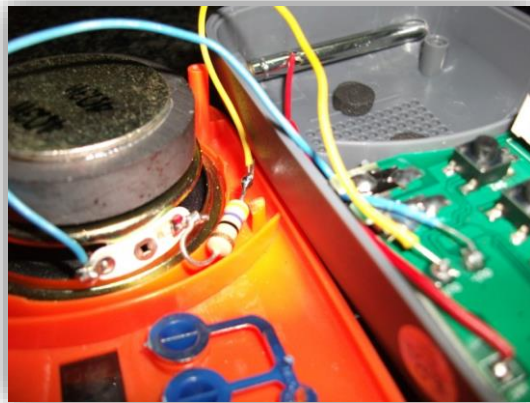


Figure 21 - Resistor soldered in series with 4ohm speaker.

4.14 - Final Arrangements

Once all the hardware was functional, all the software fragments where combined together to form a single program, also bug fixes and further configurations were made. In rare cases the microcontroller was not reacting to the alarm interrupt, hence reference to the RTC datasheet was made. Since the Alarm Flag is cleared by setting the RESET pin low, a different method was found which resulted more efficient. The flag can also be clearer by simply reading address location 11, thus the RESET pin was connected to VCC since logic output from the microcontroller wasn't required. With the new configuration the microcontroller reacted to all succeeding alarm events.

The end microcontroller was programmed to filter received data from the RF module so unexpected data can be ignored. Activation of the motor can only occur when 'DATA:ON' is transferred during the serial interrupt sub-routine, this method ensures that the motor will only rotate during the alarm event. The EasyBee manufacturer's site was used to help with the implementation of the function, PIC sample programs where downloaded so the logic could be applied to the 8051 source code.

For the user to be able to open or close the blinds automatically two buttons were introduced within the system, one for rotating the motor in reverse and the other for rotating the motor forward. Upon pressing a button the motor starts rotating in the respective direction, when the same button is re-pressed the motor stops rotating. Two LEDs were also added to indicate whether the motor is busy. When the motor is idle the green LED will be on, when the motor starts rotating the green LED will turn off and instead a red LED will light up. The extra feature was added for testing purposes.

Both the end circuits are powered via a T89C51 board which regulates the voltage at +5V, Figure 22 shows the power supply circuit. The power supply can be connected to either the polarized header or the plug pack connection. For the Alarm Clock Base Station a supply voltage range from 9V to 12V can be provided to the board since it is regulated to a steady output voltage of +5V at VCC, for the Blinds Opener Module a 12V PSU was used to power the motor. The plug pack pin can be either polarity as a full wave rectifier is built in the circuit, the capacitors' function is for smoothing and stabilising the regulator while the 1N4001 diode is used for feedback.

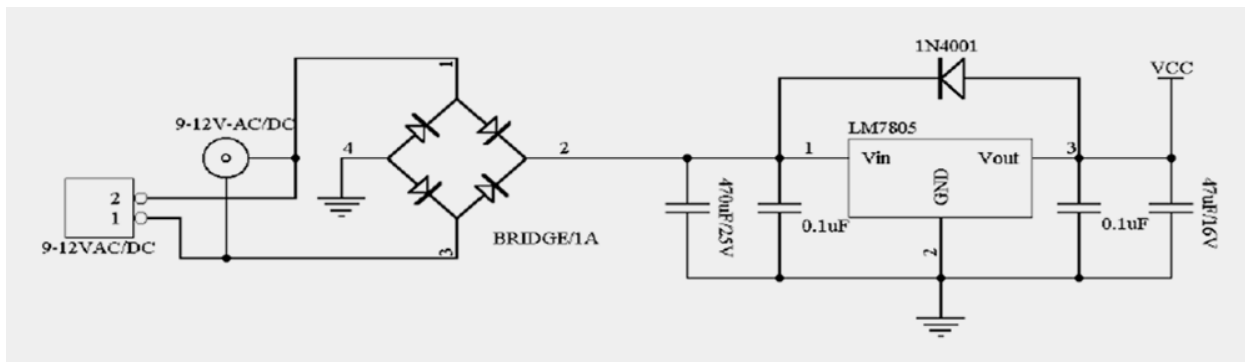


Figure 22 - Power Supply Circuit of T89C51 Training Board (JR51AC2 User Manual).

When all the hardware and software was functioning correctly, the components were soldered to their respective end PCB, initially a multi-meter was used to test each track for continuity so broken tracks or short circuits can be detected. The only problem encountered during testing was that the tracks for the contrast trimmer pot were swapped, thus LCD characters weren't being displayed since an incorrect resistance was present. Figure 23 shows a preview of the soldered components which compose the Alarm Clock Base Station, for schematics and PCB layouts refer to Appendices 3.

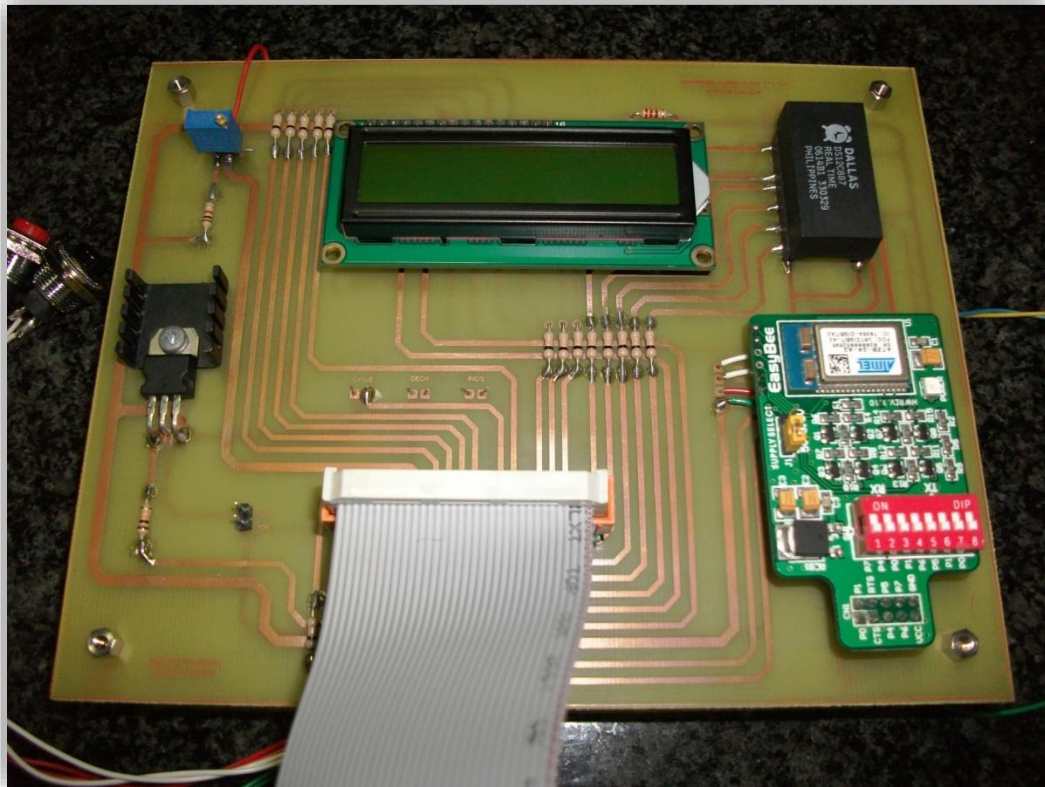


Figure 23 - Alarm Clock Base Station PCB.

To demonstrate the concept of the project, a small scale model of the blinds was built, the reason for using roller blinds was already discussed in section 2.2. Before building the structure, a sketch of the design was made so a list of materials and planning of the operation can be extracted. After calculating the dimensions of the model, wooden planks were cut down to form the outer shell of the structure. Subsequently, the circular centre of a wooden rod was drilled to fit the motor shaft and thus provide a parallel alignment with the base of the structure when placed horizontally. Fabric was attached perpendicular to the rod so it can fold around its circumference when the shaft rotates in a circular motion on its axis. An iron rod was attached to the fabric to provide downwards force and eliminate unwanted fluctuation of the blinds. The main purpose of the rod is to activate the sensors upon passing through the gap, hence stopping the motor from further rotation. An aluminium rail was fixed on each side of the structure to guide the rod to the bottom and top sensors while opening or closing the blinds. Figure 24 shows a photo of the top sensor and rod being guided through the rail.

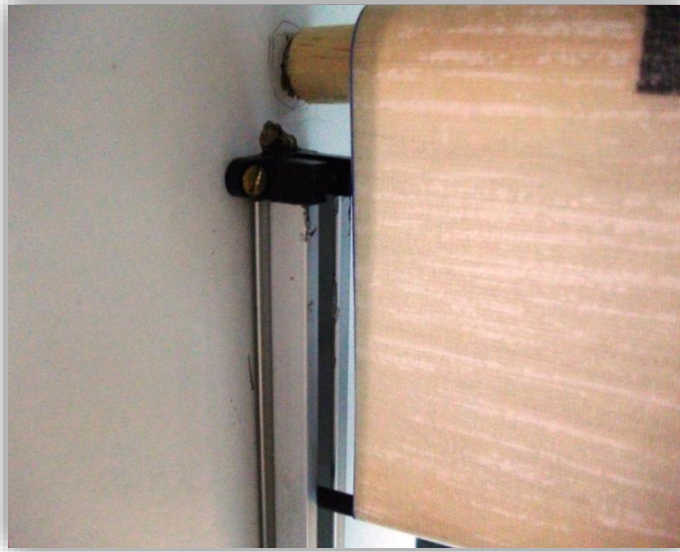


Figure 24 - Blinds top sensor and aluminium rail.

Both the end PCBs were fitted in enclosures for a more presentable result, also to protect the components from elements and static charges. A USB mount panel was attached to the enclosure of the Alarm Clock Base Station so the music player can be easily connected or disconnected from the circuit via a USB cable, the panel connection with the power source is shown in Figure 20. The main reason for applying this technique is to provide an extra feature to the user, +5V rechargeable devices such as mobiles can be connected to the USB panel mount for power or charging solution. Figures 25 and 26 demonstrate the final result of the project hardware. As it can be noted, both the Alarm Clock Base Station and the Blinds Opener Module are modular in nature, hence they can be situated far from each other since they are designed to communicate wirelessly. The main purpose for using RF modules is to eliminate physical wiring between each end, thus providing an easier installation of the system.



Figure 25 - Alarm Clock Base Station.



Figure 26 - Blinds Opener Module.

Chapter 5.

Conclusions

and

Limitations

5.1 - Final Overview

As a whole the project's functionality was successful, the challenges faced during the implementation phase and interfacing with the microcontroller were the RTC and RF modules. A long period of time was dedicated to display the time correctly on the LCD, also to communicate serially with the EasyBee boards. As a result the timekeeping function of the RTC was more accurate than expected, it was calculated that the RTC can maintain an accuracy of ± 2 minutes per month when kept in the absence of power.

By the means of milestone panning the execution of the project was finished on time, meeting all the requirements of the proposal. Consultation with the lecturer helped in solving problems and making decisions on which component to use for the project. If additional time was provided other features would have been added, also a better quality could have been achieved.

With further studies a light source could be implemented to simulate sunrise. If the alarm is set earlier than dawn or if the weather does not permit sunlight, the light source can be configured to activate instead of the blinds, filling the room gradually with artificial light. Since the hardware is modular in nature, additional units can be implemented and linked together wirelessly so they can be activated by the alarm event (e.g. TV, toaster, coffee maker). At a later stage during the course of study the Alarm Clock Base Station functions could be implemented as a mobile application so it can reduce hardware cost and thus make the system affordable for common users.

As the project was configured for presentation and exhibition, certain features weren't implemented since they are time dependent. For the system to be effective when applied into practice, the author suggests that the software should be set to open the blinds 15 minutes before the actual alarm time, this method is more effective since the room will be filled with sunlight at the moment the music player device is switched on. Also, the blinds should be set to close over night if they are still open so the system will reset for the morning wake-up routine.

After analysing the physical cost of the project, an estimate of €250 was calculated. Since the blinds were designed to a small scale, additional cost should be allocated when building the actual system. Also, the blinds motor and sensors should be enclosed

in special compartments when fixed to the window so they can be protected from rain or other elements. Furthermore the blinds motor should be more powerful when operating a full scale design, also having a silent operation.

5.2 - Limitations

During the course of the project, part-time work and school assignments were also faced which have limited the dedication for further improvements on the functionality of the hardware. Importations of components was also an issue since it delayed the process of implementation.

Because of time constraints the calendar feature wasn't implemented with the RTC since it isn't crucial for the operation of the project, hence the functionality of the hardware was given more priority. Initially it was planned to display the date and allow setting of the day and month from the LCD interface, giving the user more utility.

The music player device was originally proposed to increase its volume gradually with time when the alarm triggers. Initial study on how the solution can be solved was investigated however time was a constraint and thus research had to end. As a result the device was set to operate at a constant volume level.

References and Bibliography.

Online articles and Journals:

American Thoracic Society, 2007. Sleep Apnea Patients Have Greatly Increased Risk of Severe Car Crashes. [press release] 20th May 2007. Available at: <<http://www-archive.thoracic.org/sections/publications/press-releases/conference/articles/2007/press-releases/sleep-apnea-patients-have-greatly-increased-risk-of-severe-car-crashes.html>> [Accessed 13th October 2012].

Brian Marshall, 2000. How Digital Clocks Work. [online] Available at: <<http://electronics.howstuffworks.com/gadgets/clocks-watches/digital-clock2.htm>> [Accessed 8th December 2012].

Dr Piotr A. Wozniak, 2012. Good sleep, good learning, good life. [online] Available at: <<http://www.supermemo.com/articles/sleep.htm>> [Accessed 13th October 2012].

Josepha Cheong, M.D., Michael Herkov, Ph.D., Wayne Goodman, M.D., 2000. Seasonal Affective Disorder. [online] Available at: <http://psychcentral.com/library/depression_sad.htm> [Accessed 15th October 2012].

Nayab Suhail Hamirani, Imdad Ali Ismaili, Asad Ali Shaikh, Faheem Ahmed and Azhar Ali Shah, 2011. Password Based Rotational Multistory Car Parking System. Journal of Emerging Trends in Computing and Information Sciences. [e-journal] 2(12), 680-685. Available through: <<http://www.cisjournal.org>> [Accessed 27th October 2012].

Rick Bickle, 2005. A Simple PWM Circuit Based on the 555 Timer. [online] Available at: <<http://www.dprg.org/tutorials/2005-11a/index.html>> [Accessed 21st October 2012].

Virgil D. Wooten, M.D., 2007. How to Fall Asleep. [online] Available at: <<http://health.howstuffworks.com/mental-health/sleep/basics/how-to-fall-asleep2.htm>> [Accessed 14th October 2012].

William Bendix, Jake Metz and Durreh Tabassum, 2010. Automated Wake-Up Routine. [project] 40, Spring 2010. Available through: <<http://courses.engr.illinois.edu/ece445>> [Accessed 20th October 2012].

Books and Lecturers' notes:

Attard Joseph. Microprocessor Architecture. lecturer's notes MCAST college Paola Malta.

Muhammad Ali Mazidi, Janice G. Mazidi and Rolin D. McKinlay, 2005. The 8051 Microcontroller and Embedded Systems, 2/E. Prentice Hall.

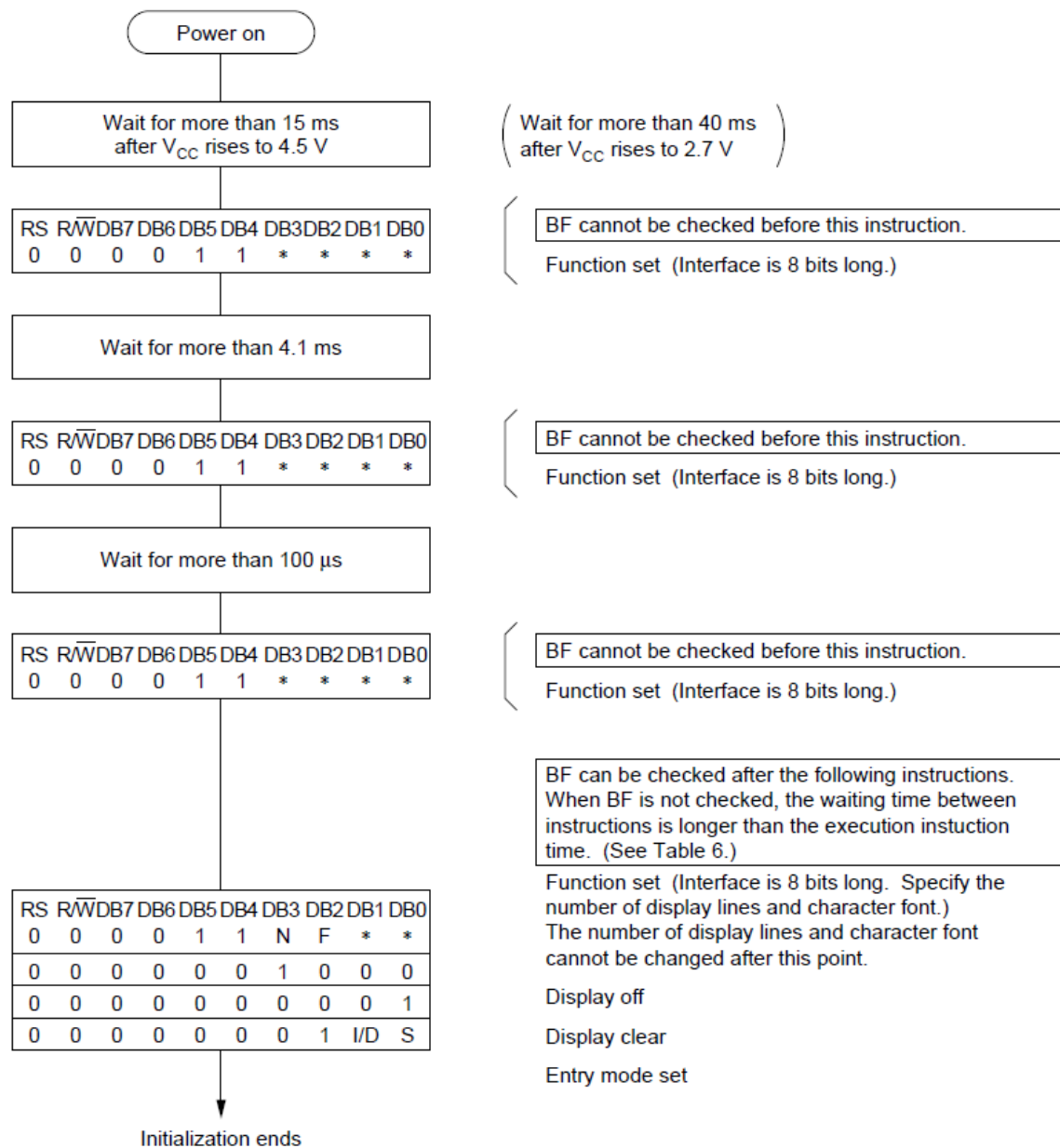
Patents:

Blackman; Stephen E. Verilux Inc, 2001. Lamp and alarm clock with gradually increasing light or sounds. U.S. Patent Number 6,236,622.

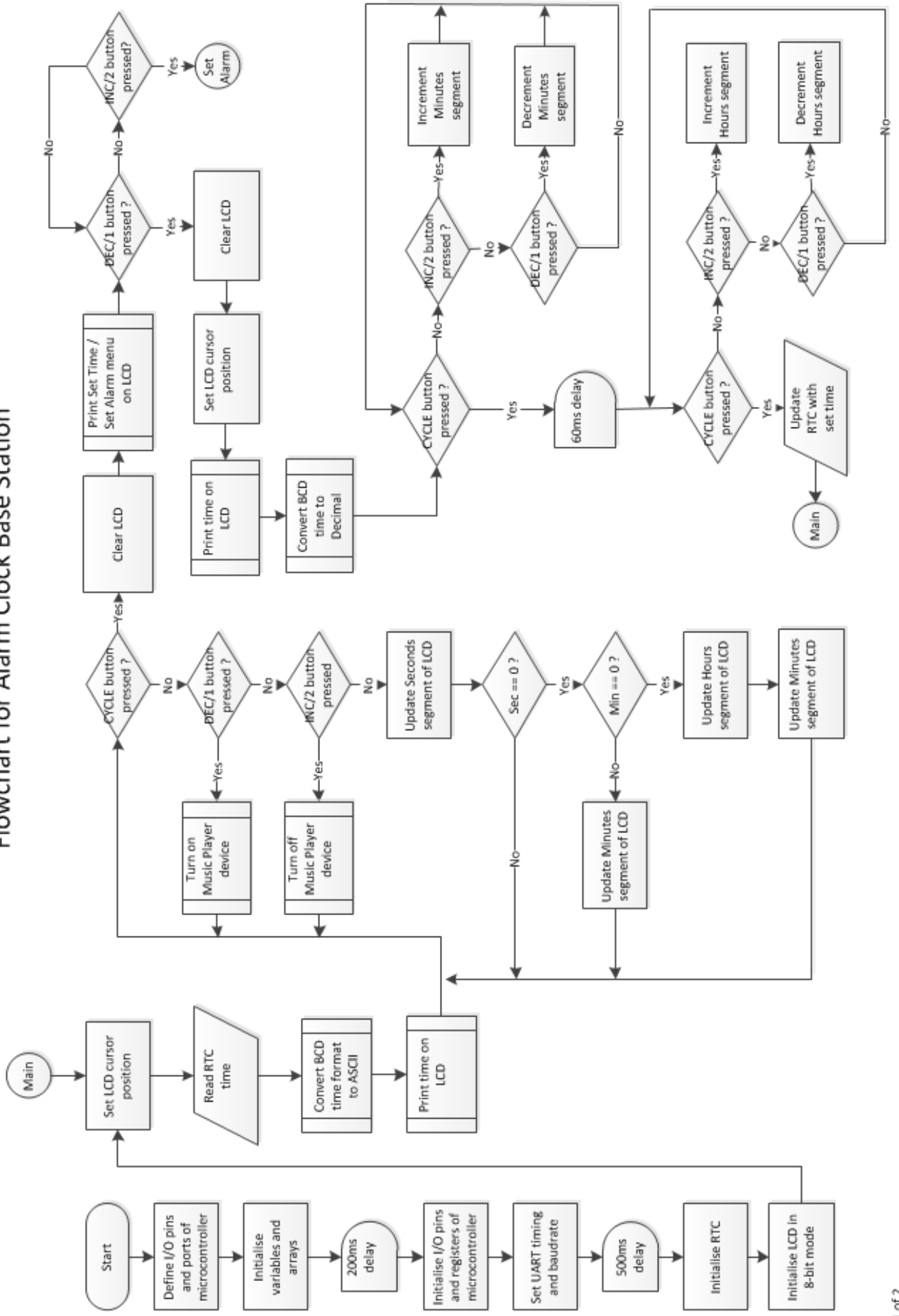
Appendices.

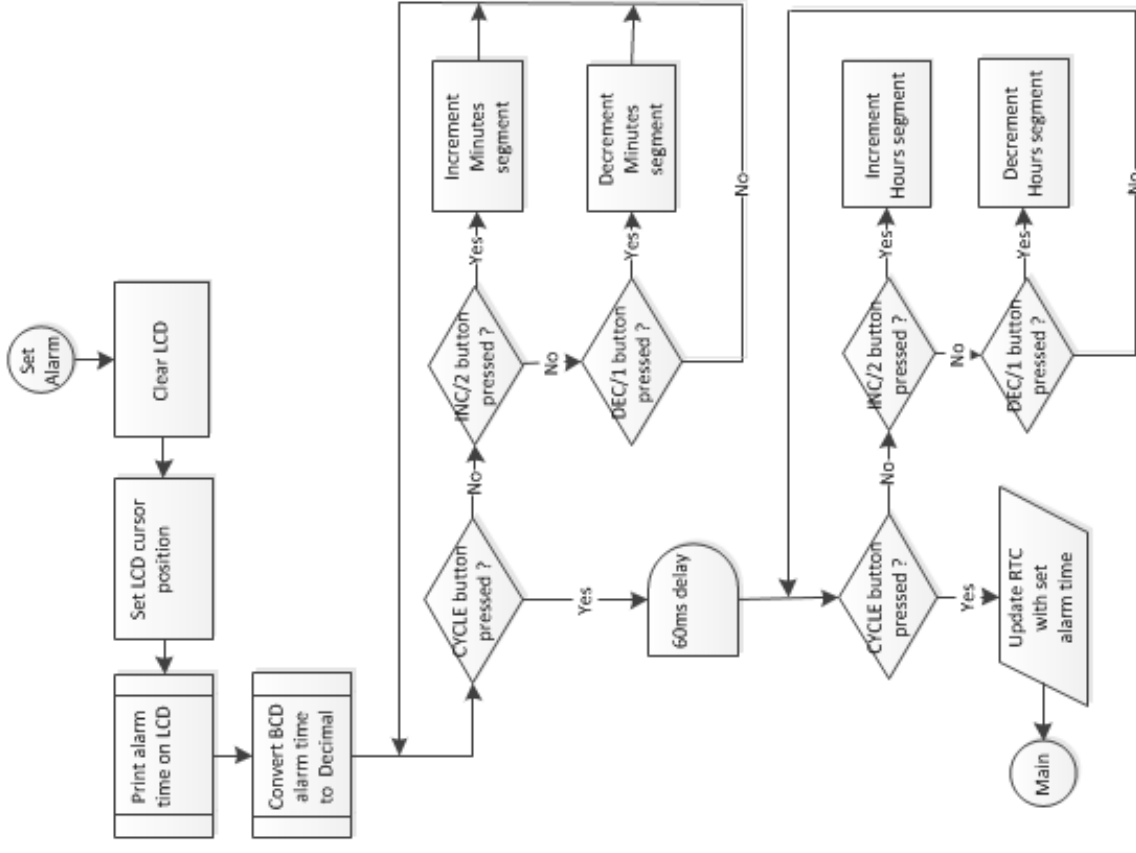
Appendices 1: Flowcharts

Figure A.1 - LCD Initialisation for 8-bit mode (Hitachi HD44780 Datasheet).

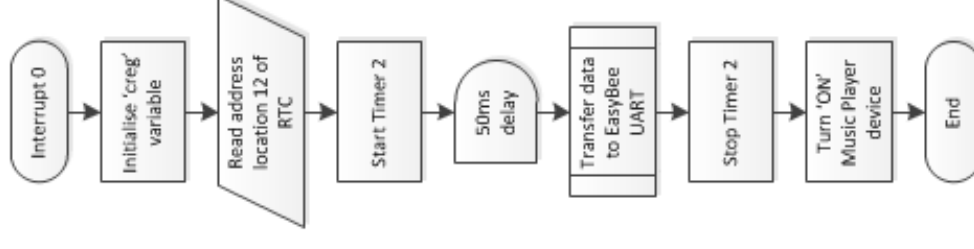


Flowchart for Alarm Clock Base Station

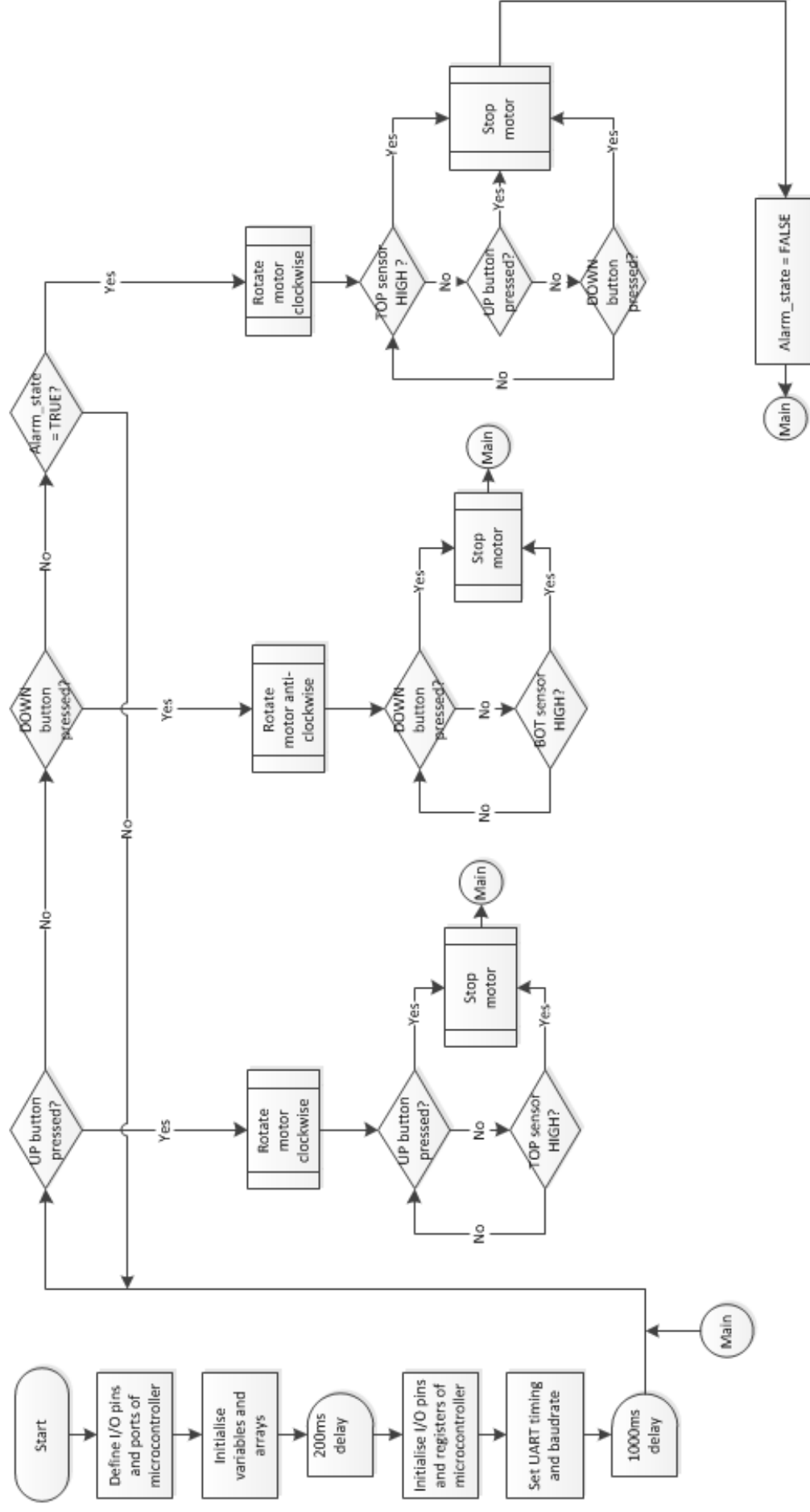




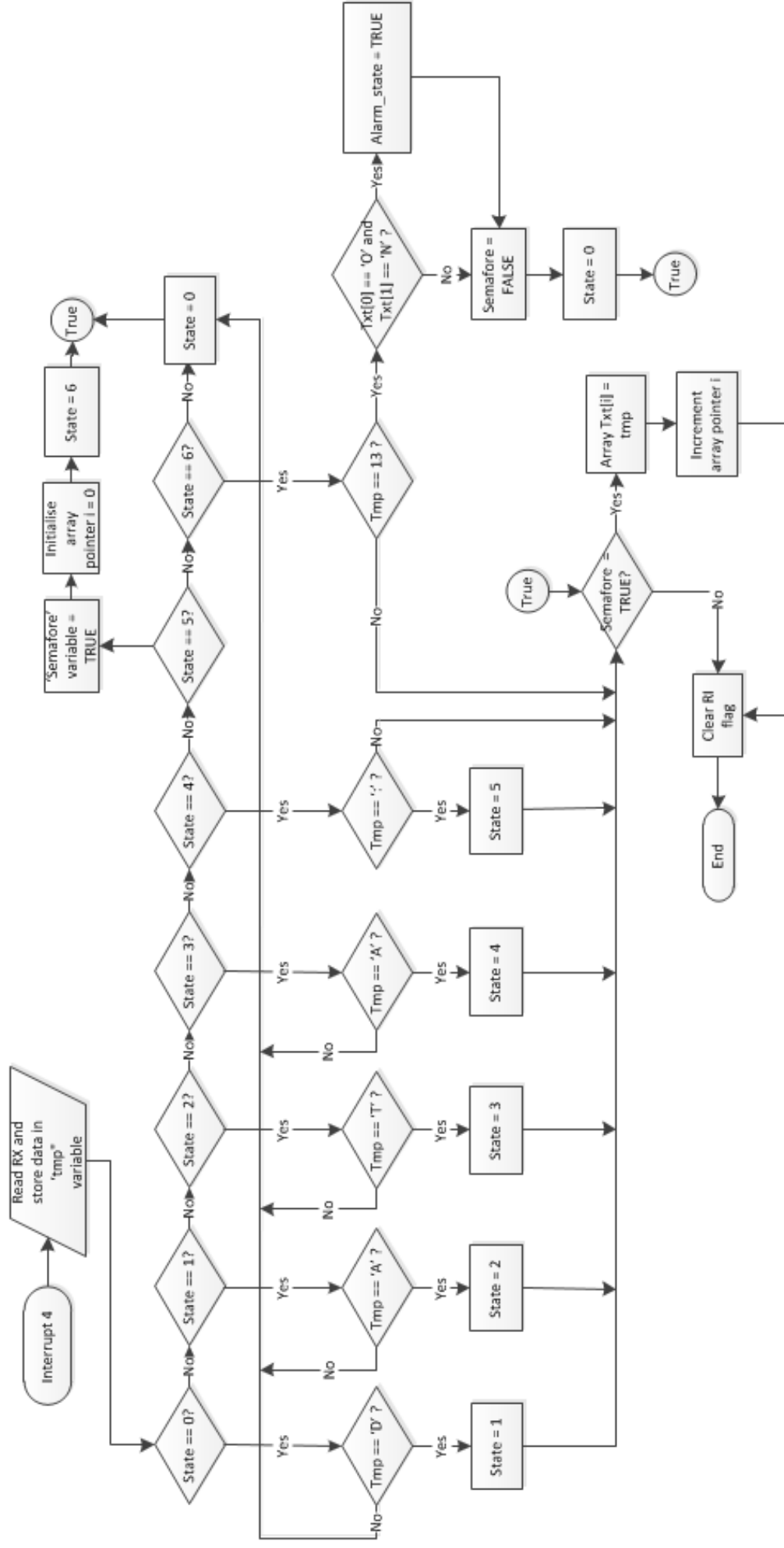
Alarm Interrupt Sub-Routine



Flowchart for Blinds Opener Module



Serial Interrupt Sub-Routine



Appendices 2 : Program Code

```
//ADVANCED ALARM CLOCK SYSTEM BY MARLON CALLEJA (June 2013)
//ALARM CLOCK BASE STATION
```

```
#include <T89C51AC2.h>    //8051 library
#include <absacc.h>
```

```
//Define ports and data pins of microcontroller
#define dataport P2
sbit rs = P1^1;
sbit rw = P1^2;
sbit en = P1^3;
sbit cycle = P1^4;
sbit dec = P1^5;
sbit inc = P1^6;
sbit busy = P2^7;
sbit radio = P1^7;
```

```
unsigned char hr,mir,sec,nun[60]={0X00,0X01,0X02,0X03,0X04,0X05,0X06,0X07,0X08,0X09,
0X10,0X11,0X12,0X13,0X14,0X15,0X16,0X17,0X18,0X19,0X20,0X21,0X22,0X23,0X24,0X25,0X26,
0X27,0X28,0X29,0X30,0X31,0X32,0X33,0X34,0X35,0X36,0X37,0X38,0X39,0X40,0X41,0X42,0X43,
0X44,0X45,0X46,0X47,0X48,0X49,0X50,0X51,0X52,0X53,0X54,0X55,0X56,0X57,0X58,0X59};
int int_hr, int_min;
```

```
void delay(int time)
{
    //Time delay function
    int i,j;
    for(i=0;i<time;i++)
        for(j=0;j<1275;j++);
}
```

```
void lcdready()
{
    //Check if LCD is busy processing data
    busy = 1;           //make the busy pin an input
    rs = 0;
    rw = 1;
    while (busy==1)     //wait here for busy flag
    {
        en = 0;         //strobe the enable pin
        delay(1);
        en = 1;
    }
    return;
}
```

```
void lcdcmd (unsigned char value)
{
    //LCD command input
    lcdready();         //check the LCD busy flag
    dataport = value;   //put the value on the pins
    rs = 0;
    rw = 0;
    en = 1;             //strobe the enable pin
    delay(1);
    en = 0;
    return;
}
```

```
void lcddata (unsigned char value)
{
    //LCD character input
    lcdready();         //check the LCD busy flag
    dataport = value;   //put the value on the pins
```

```
    rs = 1;
    rw = 0;
    en = 1;           //strobe the enable pin
    delay(1);
    en = 0;
    return;
}

void bcdconv(unsigned time)
{
    //convert BCD to ASCII and send it to lcddata function for display
    unsigned char x,y;
    x=time&0x0F;
    x=x|0x30;
    y=time&0xF0;
    y=y>>4;
    y=y|0x30;
    lcddata(y);
    lcddata(x);
}

void lcd_string (unsigned char x[])
{
    // data string array, every single char in the array is sent to lcddata function
    for display
    int pnt=0;           //array pointer

    while(x[pnt]!='\0')
    {
        lcddata(x[pnt]);
        pnt++;           // increment array pointer
    }
}

void display_time(void)
{
    //Display Time. Read the time, convert it to ASCII, and send it to the LCD
    lcdcmd(0x01);        //Clear LCD
    lcdcmd(0x81);        //position cursor
    hr=XBYTE[4];         //get hour
    bcdconv(hr);         //convert and display
    lcddata(':');
    min=XBYTE[2];        //get minute
    bcdconv(min);        //convert and display
}

void display_alarm(void)
{
    //Display Alarm Time. Read the time, convert it to ASCII, and send it to the LCD

    lcdcmd(0x01);        //Clear LCD
    lcdcmd(0x81);        //position cursor
    hr=XBYTE[5];         //get alarm hour
    bcdconv(hr);         //convert and display
    lcddata(':');
    min=XBYTE[3];        //get alarm minute
    bcdconv(min);        //convert and display
}

void convert_time(void)
{
    //Convert time from BCD format to decimal
    int x;
```

```

    for(x=0;x<60;x++)          //acquire min and save it as integer
    {
        if(min==num[x])
            int_min=x;
    }

    for(x=0;x<24;x++)          //acquire hr and save it as integer
    {
        if(hr==num[x])
            int_hr=x;
    }
}

void tx_data (unsigned char x[])
{
    //Transfer serially a string of data
    int pnt=0;                  //array pointer

    while(x[pnt]!='\0')
    {
        SBUF = x[pnt];          // load SBUF with char data for transmitting
        while(TI==0);           // Wait for byte to be transmitted
        TI=0;                    // Reset the transfer interrupt flag
        pnt++;                   // increment array pointer
    }
}

void send(void) //transmit data to RX
{
    tx_data("ATD55\r");        //AT command for sending data
    tx_data("ON\r");            //data to be transmitted
}

void alarm() interrupt 0 // ALARM INTERRUPT ---- Clear Alarm Interrupt Flag (AF) f
rom Register C of RTC, turn on external device and transmit data to RX
{
    unsigned char creg;

    creg = XBYTE[12];          // read value of address location 12 to clear AF
    TR2=1;                      // start timer 2

    delay(50);
    send();

    TR2=0;                      // stop timer 2
    radio = 1;                  // turn on radio
}

void radio_on(void)
{
    //turn on external device
    delay(50);
    radio = 1;
}

void radio_off(void)
{
    //turn off external device
    delay(50);
    radio = 0;
}

```

```
}

void main(void)
{
    delay(200);
    AUXR = 0x0A;      //initialise AUXR to enable ALE and external memory access (RTC
memory)
    IEN0=0x81;        //enable interrupts, enable external interrupt 0
    cycle = 1;        //set pin as input
    inc = 1;          //set pin as input
    dec = 1;          //set pin as input
    radio = 0;

    //Set timing and baud rate for serial communication with RF module

    SCON=0x50;         //enables serial reception in 8-bit mode
    T2MOD=0;           //Timer2
    T2CON=0x3C;        //use Timer 2 as receive/transmit clock for serial port
    RCAP2H=0xff;
    RCAP2L=0xf1;
    TR2=0;             //turn off timer 2

    //RTC initialization. Turn on oscillator and set initial time

    delay(500);        // Power-up delay - the RTC is accessible after 200ms
    XBYTE[11]=0x22;    // set as 24-hour clock. Also permits the Alarm Flag(AF) bi
t in register C to assert IRQ for interrupt 0
    XBYTE[10]=0x2C;    // turn on osc
/*
    XBYTE[0]=0x00;     SECOND
    XBYTE[2]=0x00;     MINUTE
    XBYTE[4]=0x00;     HOUR
    XBYTE[1]=0x00;     Alarm SECOND
    XBYTE[3]=0x00;     Alarm MINUTE
    XBYTE[5]=0x00;     Alarm HOUR */

    //LCD initialization in 8-bit mode
    lcdcmd(0x30);
    delay(10);
    lcdcmd(0x30);
    delay(10);
    lcdcmd(0x30);
    delay(10);
    lcdcmd(0x38);      //Set: 2 Line, 8-bit, 5x7 dots
    lcdcmd(0x0C);      //Display on, Cursor off
    lcdcmd(0x01);      //Clear LCD
    lcdcmd(0x06);      //Entry mode, auto increment with no shift

    while(1)
    {
        display_time();
        lcddata(':');

        while(cycle==1)    //display time
        {
            if (dec==0)    radio on();
            else if (inc==0) radio_off();

            lcdcmd(0x87);  //position cursor
            sec=XBYTE[0];  //get second
```

```

        bcdconv(sec);          //convert and display
        delay(20);

        if(sec!=0X00) continue;

        min=XBYTE[2];          //get minute

        if(min==0X00)
        {
            hr=XBYTE[4];       //get hour

            lcdcmd(0x81);      //position cursor
            delay(100);
            bcdconv(hr);        //convert and display
            lcdcmd(0x84);      //position cursor
            bcdconv(min);       //convert and display

        }
        else
        {
            lcdcmd(0x84);      //position cursor
            delay(200);
            bcdconv(min);       //convert and display

        }

    }
    delay(60);
    lcdcmd(0x01);
    lcdcmd(0x80);              //position cursor
    lcd_string("1.Set Time");
    lcdcmd(0xC0);              //position cursor
    lcd_string("2.Set Alarm");

    while(1)
    {
        if(dec==0)              //SET TIME-----
        {
            delay(60);
            display_time();
            lcdcmd(0x8A);
            lcd_string("TIME");
            lcdcmd(0xCA);
            lcd_string("MIN");
            convert_time();

            while(cycle==1)
            {
                if(inc==0)
                {
                    delay(2);    //wait for switch bounce time
                    int_min++;

                    if(int_min>59) int_min=0;

                    lcdcmd(0x84);
                    min=num[int_min];
                    delay(30);
                    bcdconv(min);
                }
                else if(dec==0)

```

```
        {
            delay(2); //wait for switch bounce time
            int_min--;

            if(int_min<1) int_min=59;

            lcdcmd(0x84);
            min=num[int_min];
            delay(30);
            bcdconv(min);
        }

    }

    delay(60);
    lcdcmd(0xCA);
    lcd_string("HOUR");

    while(cycle==1)
    {
        if(inc==0)
        {
            delay(2); //wait for switch bounce time
            int_hr++;

            if(int_hr>23) int_hr=0;

            lcdcmd(0x81);
            hr=num[int_hr];
            delay(30);
            bcdconv(hr);
        }
        else if(dec==0)
        {
            delay(2); //wait for switch bounce time
            int_hr--;

            if(int_hr<1) int_hr=23;

            lcdcmd(0x81);
            hr=num[int_hr];
            delay(30);
            bcdconv(hr);
        }
    }

    XBYTE[4]=hr;
    XBYTE[2]=min;
    delay(60);

    break;
}

else if(inc==0) //SET ALARM-----
{
    delay(60);
    display_alarm();
    lcdcmd(0x8A);
    lcd_string("ALARM");
    lcdcmd(0xCA);
    lcd_string("MIN");
}
```

```
convert_time();

while(cycle==1)
{
    if(inc==0)
    {
        delay(2); //wait for switch bounce time
        int_min++;

        if(int_min>59) int_min=0;

        lcdcmd(0x84);
        min=num[int_min];
        delay(30);
        bcdconv(min);
    }
    else if(dec==0)
    {
        delay(2); //wait for switch bounce time
        int_min--;

        if(int_min<1) int_min=59;

        lcdcmd(0x84);
        min=num[int_min];
        delay(30);
        bcdconv(min);
    }
}

delay(60);
lcdcmd(0xCA);
lcd_string("HOUR");

while(cycle==1)
{
    if(inc==0)
    {
        delay(2); //wait for switch bounce time
        int_hr++;

        if(int_hr>23) int_hr=0;

        lcdcmd(0x81);
        hr=num[int_hr];
        delay(30);
        bcdconv(hr);
    }
    else if(dec==0)
    {
        delay(2); //wait for switch bounce time
        int_hr--;

        if(int_hr<1) int_hr=23;

        lcdcmd(0x81);
        hr=num[int_hr];
        delay(30);
        bcdconv(hr);
    }
}
```

```
        XBYTE[5]=hr;
        XBYTE[3]=min;
        delay(60);

        break;

    }
    else if (cycle==0)          //EXIT MENU-----
    {
        delay(50);
        break;
    }
}
}
```

```
//ADVANCED ALARM CLOCK SYSTEM BY MARLON CALLEJA (June 2013)
//BLINDS OPENER MODULE
```

```
#include <T89C51AC2.h>    //8051 library
#include <absacc.h>
```

```
//set pins for H-bridge
sbit in1 = P2^0;
sbit in2 = P2^1;
sbit en1 = P2^2;
//set pins for photo-int sensors
sbit opto bot = P1^1;
sbit opto top = P1^2;
//set pins for blind's control buttons
sbit sw down = P1^3;
sbit sw up = P1^4;
//set pins for motor activity status leds
sbit led busy = P1^5;
sbit led_idle = P1^6;
```

```
unsigned char tmp, txt[3];
int semafor, i, state;
int alarm_state=0;
```

```
void delay(int time)
{
    int i,j;
    for(i=0;i<time;i++)
        for(j=0;j<1275;j++);
}
```

```
unsigned char rx_data()
{
    //Function for reading data from SBUF
    unsigned char rx;

    while(RI==0); //wait for RI flag
    rx = SBUF;    //read SBUF
    RI=0;         //clear RI flag
    return rx;
}
```

```
void serial_it(void) interrupt 4    //SERIAL INTERRUPT
{
    // Checks if data 'ON' has been recieved

    if(RI==1) tmp = rx_data();

    switch(state)
    {
        case 0:
        {
            if (tmp == 'D')
                state=1;
            else
                state=0;
            break;
        }

        case 1:
        {
            if (tmp == 'A')
                state=2;
            else
                state=0;
            break;
        }
    }
}
```

```
    }

    case 2:
    {
        if (tmp == 'T')
            state = 3;
        else
            state=0;
        break;
    }

    case 3:
    {
        if (tmp == 'A')
            state = 4;
        else
            state = 0;
        break;
    }

    case 4:
    {
        if (tmp == ':')
            state = 5;
        break;
    }

    case 5:
    {
        semafor = 1;
        i = 0;
        state = 6;
        break;
    }

    case 6:
    {
        if (tmp == 13)    //check for end of data string
        {
            if (txt[0]=='O' && txt[1]=='N')
            {
                alarm_state=1;
            }
            semafor = 0;
            state = 0;
        }
        break;
    }

    default:
    {
        state=0;
        break;
    }
}

if (semafor)    // if semafor = 1 (true) perform action
{
    txt[i] = tmp;                // move the data received into array txt[]
    i++;                        // increment array pointer
}
RI=0;    //clear RI flag
}
```

```
void rotate_f(void)
```

```
{
    in1 = 1;           //Make positive of motor 1
    in2 = 0;           //Make negative of motor 0
    en1 = 1;           //Enable L293D
    led_idle = 1;
    led_busy = 0;
}

void rotate_k(void)
{
    in1 = 0;           //Make positive of motor 0
    in2 = 1;           //Make negative of motor 1
    en1 = 1;           //Enable L293D
    led_idle = 1;
    led_busy = 0;
}

void stop(void)
{
    in1 = 0;           //Make positive of motor 0
    in2 = 0;           //Make negative of motor 0
    en1 = 0;           //Disable L293D
    led_idle = 0;
    led_busy = 1;
}

void sw_alarm(void)
{
    rotate_b();
    while(opto_top==0 && sw_up==1 && sw_down==1)    // Switch & sensor polling
    {
        delay(5);
    }
    stop();
    alarm_state=0;
    delay(50);
}

void up(void)
{
    delay(50);
    rotate_b();
    while(sw_up==1 && opto_top==0)    // Switch & sensor polling
    {
        delay(5);
    }
    stop();
    delay(50);
}

void down(void)
{
    delay(50);
    rotate_f();
    while(sw_down==1 && opto_bot==0)    // Switch & sensor polling
    {
        delay(5);
    }
    stop();
}
```

```
    delay(50);
}

void main(void)
{
    delay(200);
    sw up = 1;
    sw down = 1;
    opto top = 1;
    opto bot = 1;
    led idle = 0;
    led_busy = 1;

    IEN0=0x90;           //enable interrupts, enable serial interrupt

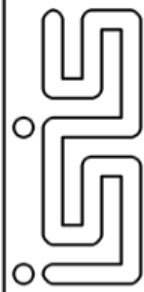
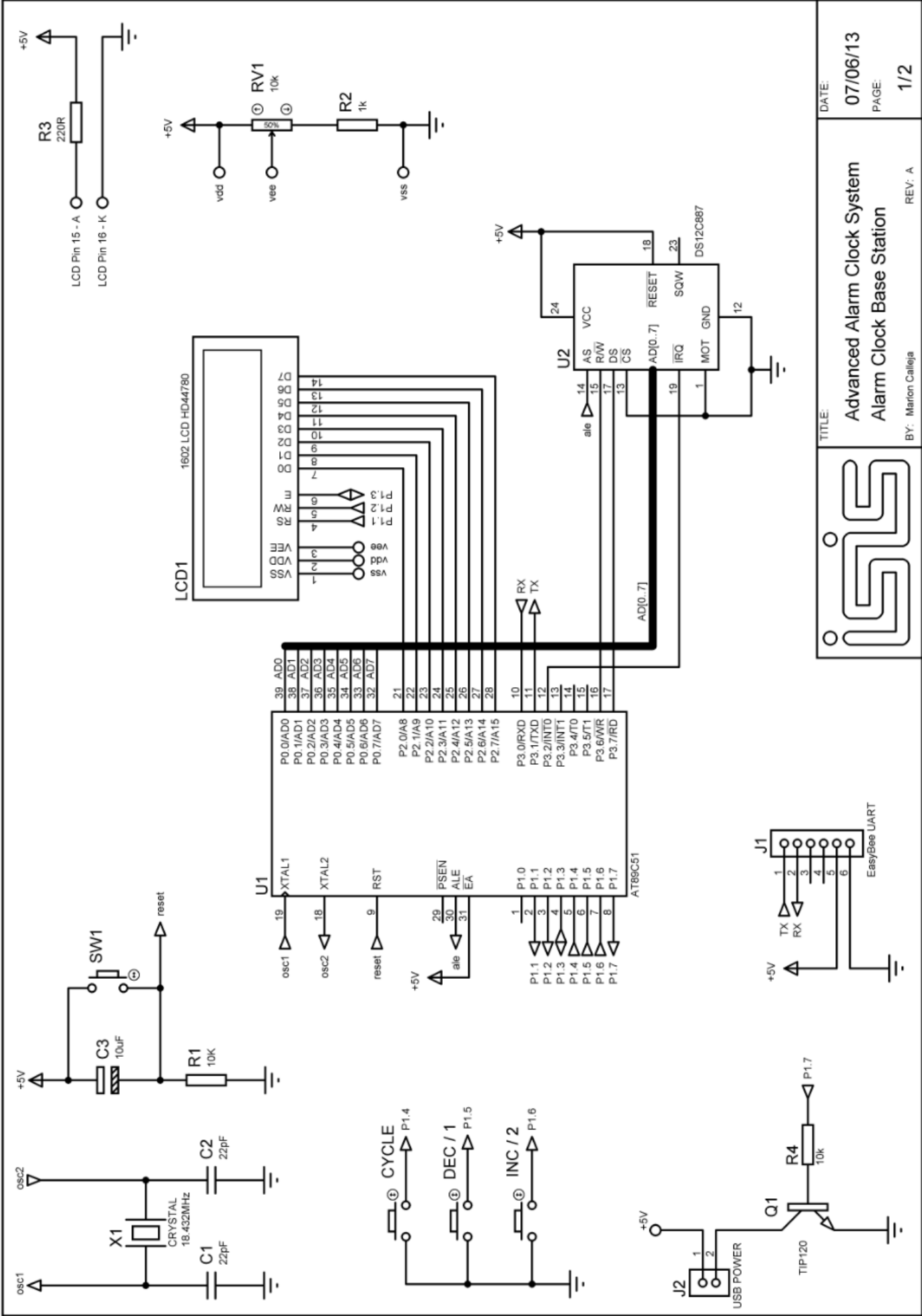
    SCON=0x50;           //enables serial reception in 8-bit mode
    T2MOD=0;             //Timer2
    T2CON=0x30;          //use Timer 2 as receive/transmit clock for serial port
    RCAP2H=0xff;
    RCAP2L=0xf1;
    TR2=1;               //turn on timer 2

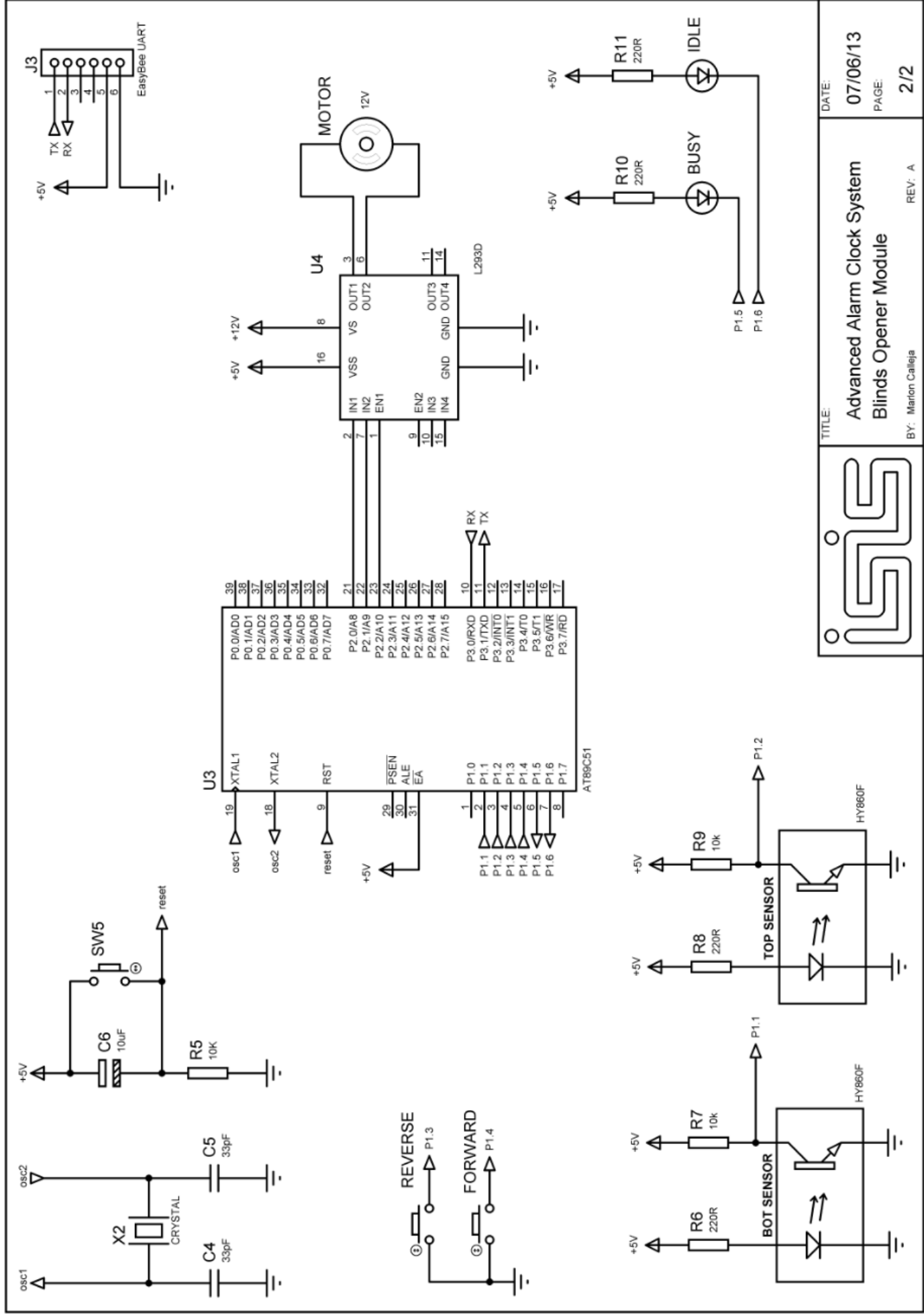
    delay(1000);         //Power-up delay

    while(1)             // Switch polling
    {
        if(sw up==0) up();
        else if(sw down==0) down();
        else if (alarm_state==1) sw_alarm();
    }

}
```

Appendices 3 : Circuit Diagrams





TITLE:

DATE:

Advanced Alarm Clock System Blinds Opener Module

07/06/13

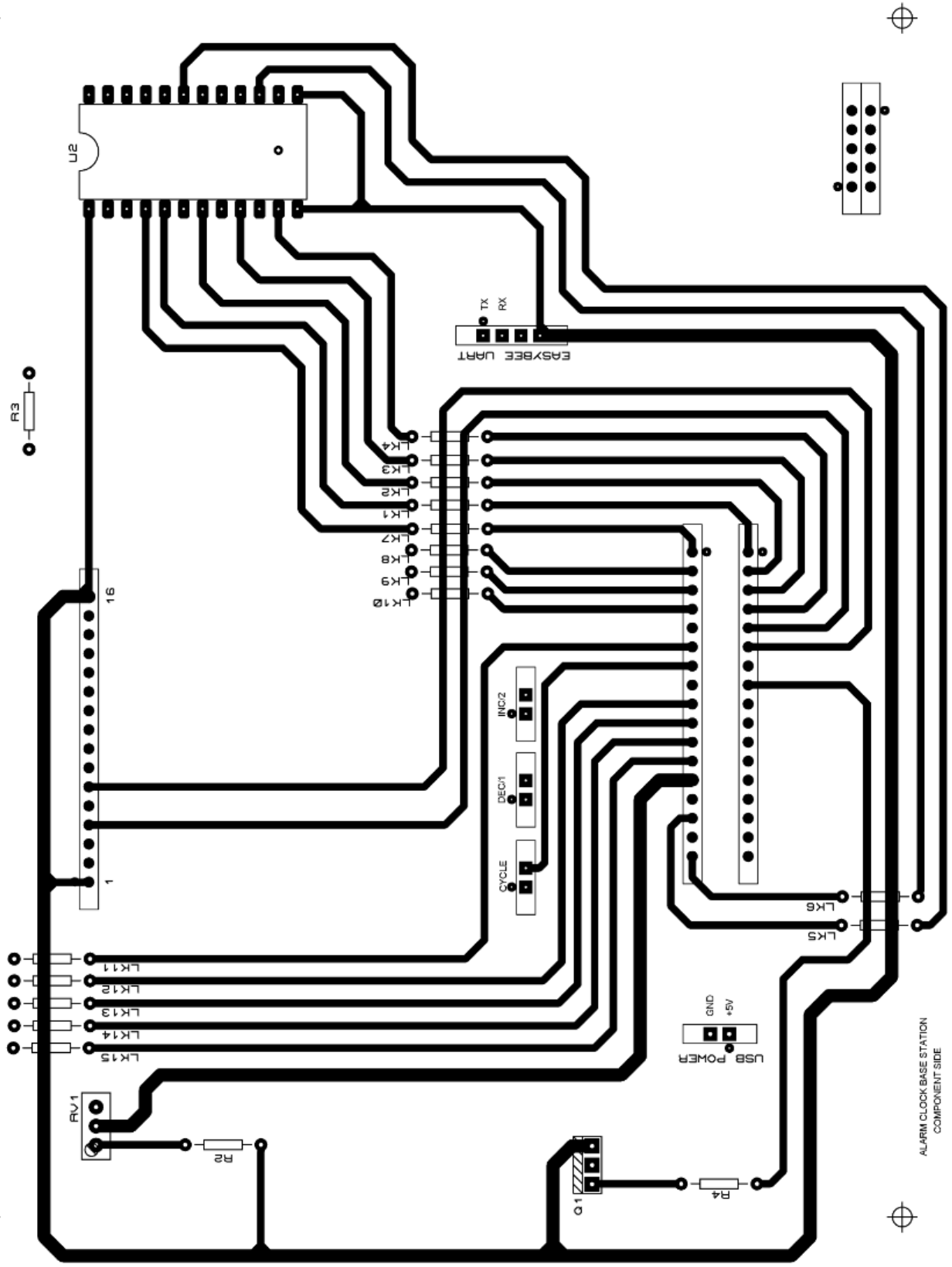
PAGE:

2/2

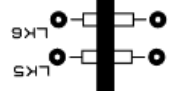
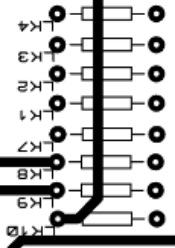
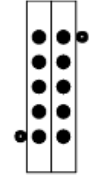
REV: A

BY: Marlon Calleja

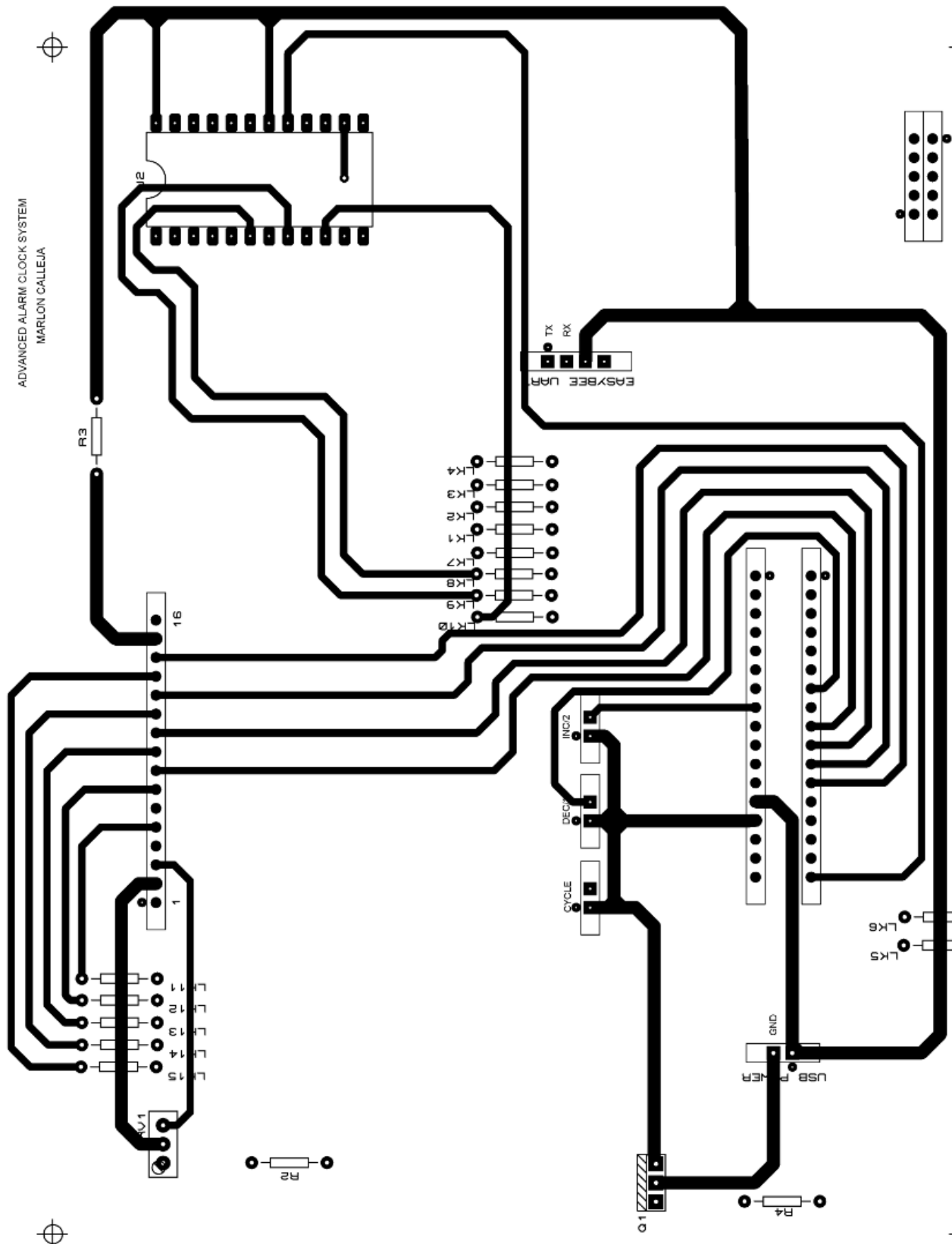




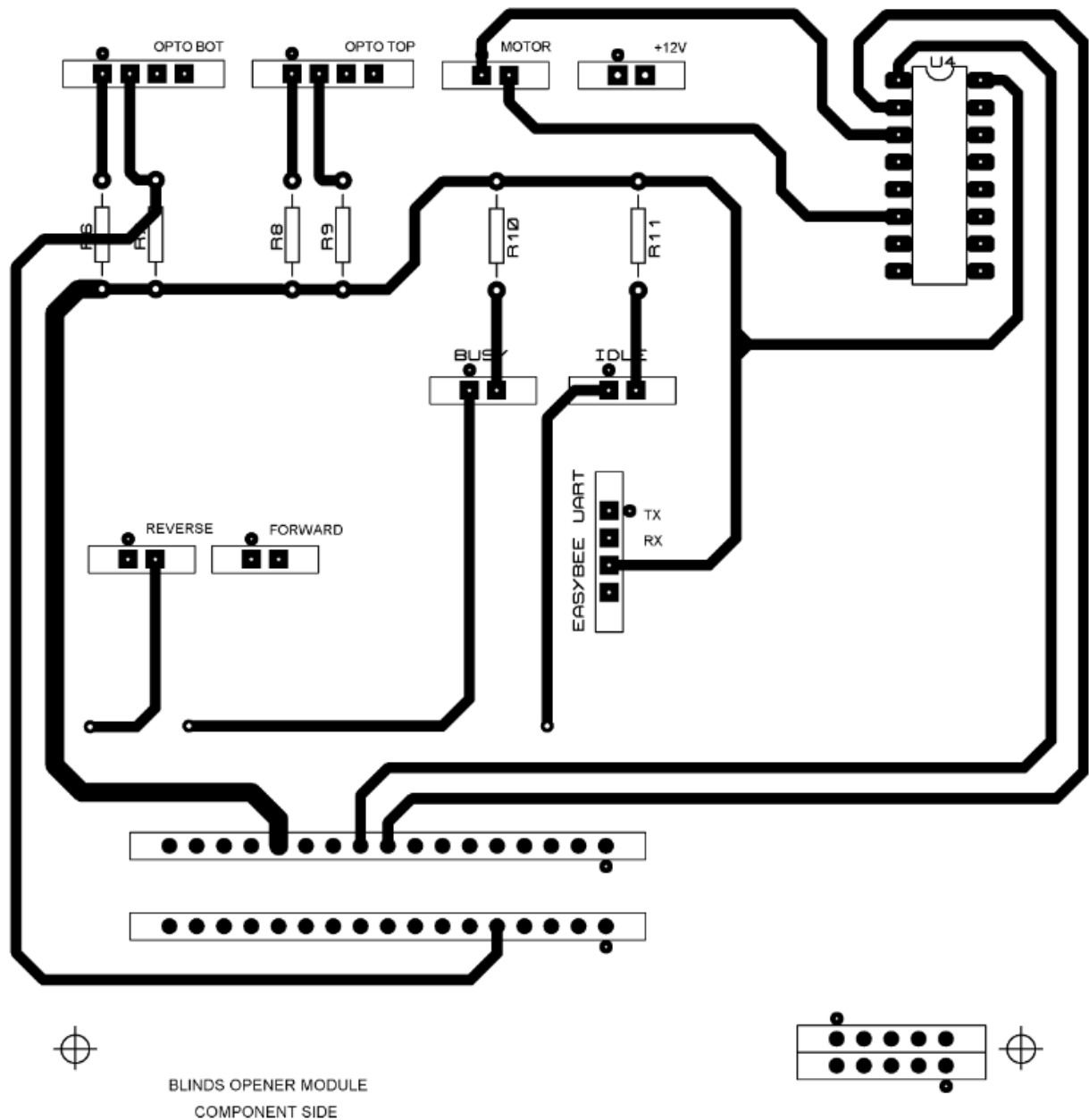
ADVANCED ALARM CLOCK SYSTEM
MARLON CALLEJA



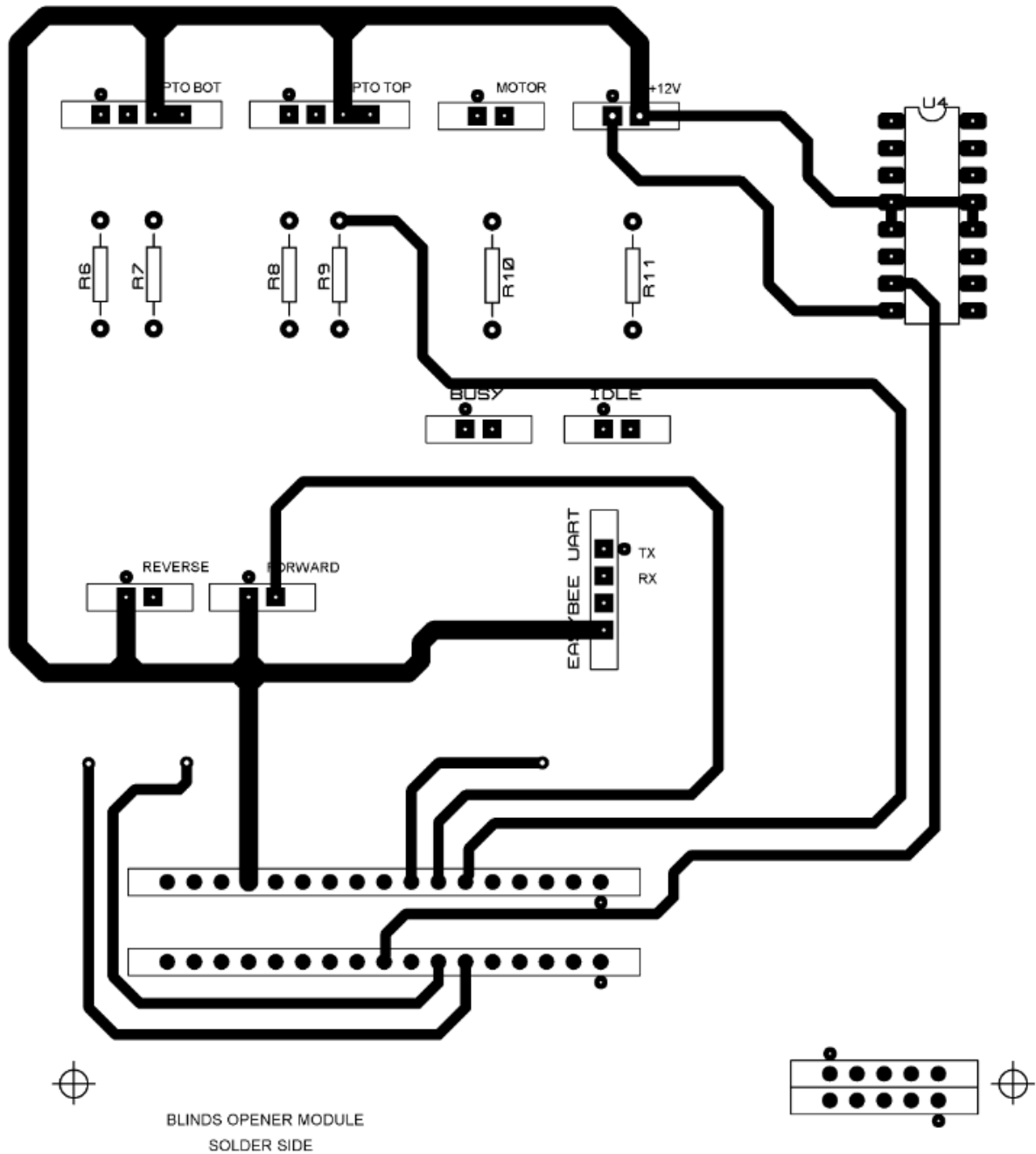
ALARM CLOCK BASE STATION
SOLDER SIDE



ADVANCED ALARM CLOCK SYSTEM
MARLON CALLEJA



ADVANCED ALARM CLOCK SYSTEM
MARLON CALLEJA



Appendices 4 : Datasheets *(provided on CD)*