University of Puerto Rico
Mayagüez Campus
Department of Electrical and Computer Engineering

µTracker

Project Final Report

Prepared for

Manuel A. Jimenez, Ph.D. – Associate Professor

INEL 4217
Section 070
Professor: Manuel A. Jimenez

By
Group: Position Aware

José A. Figueroa
Omar A. Candelaria
Pedro J. Nieves
Yadiel Lamb

November 21, 2005

Abstract:

The GPS µTracker is an electronic device that uses the Global Positioning System technology to acquire position coordinates data and analyze it, providing the user with a range of data feedback including distance between two points and direction. The device has multiple applications, some as simple as finding a car in a parking lot, others as needful as an expeditions guide and some as important as surveying. The system is developed to be portable, low power and relatively low cost, while providing a high efficiency and precision feedback to the user through a simple interface. The GPS satellite data is collected through a single chip GPS receiver, the Sony GXB5210, and processed using the Atmel AT89C51RC2, a microcontroller which is based in the Intel 8051 architecture.

Table Of Contents

List of Illustrations

1. Introduction

The work was based on a simple idea, to help people find their cars in the concrete jungle that are today parking lots. It has become a common problem of our present society, and the purpose of the project was to create a device such that would help to solve it. Even though the problem was becoming more and more common, no one had thought of doing an electronic solution to this problem. However, we saw that it was time for such a device to be developed.

The device would have to be portable, low power, and capable of location detection through wireless methods. The system couldn't also be too complicated since it is supposed to attract a general market of users of all types and social classes. In that way, the system can't be too technologically oriented. In the same manner, it couldn't be sensitive to signal interference, since it has to work on different types of environments and not in a single, specified and controlled location. The project was later proposed to Dr. Manuel Jimenez which understood the uses which the device will have in present day society, and approved it. He also suggested the implementation of the Global Positioning System (GPS) technology as the method to provide location awareness to the device.

The main idea for the development and functionality of the device was then conceived. The system will work by obtaining the current position coordinates by means of the standard latitude and longitude measurements through a GPS signal receiver chip. Then the device will provide the user a mean to store this coordinates into memory and later be able to compare the current position with the one in memory, providing the user with analysis feedback, including distance between the two points and direction. By implementing the system in this way, the device would have a wide range of different uses, some including expeditions and surveying applications.

The GPS chip selected was the Sony GXB5210, distributed and provided by Synergy, which was interfaced using an Atmel AT89C51RC2 which is based in the Intel 8051 microcontroller architecture and technology.

2. Theory

2.1 GPS Introduction

The Global Positioning System (GPS) is a satellite based navigation system offering precision navigation capability. Originally designed for military use, civilian access has been permitted to specific parts of the GPS.

Users can expect a position accuracy of 25 m or better in three dimensions. The GPS signal is available 24 hours per day throughout the world and in all weather conditions. The equipment necessary to receive and process GPS signals is affordable and reliable and does not require atomic clocks or antenna arrays.

In its most basic terms, GPS determines the position of the user by triangulation. By knowing the position of the satellite and the distance from the satellite; combinations of satellites can be used to determine the exact position of the receiver. The fundamental means for GPS to determine distance is the use of time. By using accurate time standards and by measuring changes in time, distance is computed.

GPS ranging signals are broadcast on two frequencies: L1 (1575.42 MHz) and L2 (1227.6 MHz) of which the L1 frequency is available for civilian use. The GPS signal consists of a repetitive binary signal that receivers use to determine the time at which the code was sent from the satellite, as shown. The waveform from the satellite is matched with an internally generated waveform within the receiver. The time difference between matching waveforms is used to compute the distance from a satellite.

The GPS constellation is composed of 24 satellites (21 active satellites and three orbital spares). To calculate a two dimensional position at least three satellites are necessary. For a three dimensional position at least 4 satellites are necessary. As the number of satellites in view is increased, the location error decreases.

2.2 Direction Calculation

Most of the work involved with GPS is handled internally by the Sony GPS chip. What remains to be done is to ask the chip for the destination coordinates and the current coordinates and use that information to arrive at the target location.

Essentially what we must do to find the target, is to find the angle between the destination vector and the compass orientation. For example, if the destination direction is west and the compass orientation (we are looking at that direction) is east, the device

should specify that we must move backwards to find the location (or turn 180 degrees and move forward).

Our biggest concern is to do this efficiently. The formula to calculate the angle between two vectors is given below:

$$angle = \cos^{-1}\left(\frac{V1 \bullet V2}{|V1||V2|}\right)$$

The above formula requires too many computing resources for our application. The operation requires a dot product, two magnitudes and a division, not to consider the inverse cosine that are very hard to perform with the microcontroller that was chosen for our application. The situation can be improved by taking into account that the compass direction is a unit vector. As such, the magnitude does not need to be computed.

Instead of using the above formula, we simplified the calculation by taking into account that the compass only provides eight directions. To calculate the direction, we must map the destination direction to one of the cardinal directions (North, South, East, West, North-East, North-West, South-East and South-West).

Then we compare the compass and destination directions to identify in which direction to move. The following formula is used when we have the destination mapped to a cardinal direction:

$$dir\_to\_move = ((dest\_dir - compass\_dir) + 8) \bmod 8$$

The above calculation should maintain the destination and compass directions between 0 and 7, as to maintain the direction to move within this range. Below is a table and diagram that illustrate the code assignment for each cardinal direction.

Direction	Code	Direction	Code
North	0	North-West	1
West	2	South-West	3
South	4	South-East	5
East	6	North-East	7

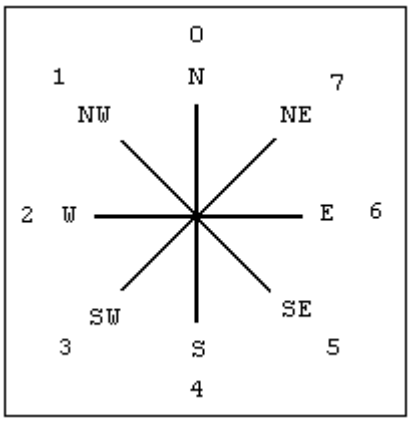


Figure 1: Cardinal Coordinate Mapping

We now have to map the destination direction to a cardinal direction. The procedure is simple. Below are two diagrams used to better illustrate the procedure.

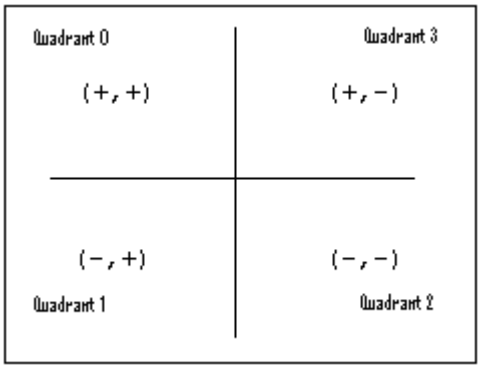


Figure 2: Quadrant Calculation

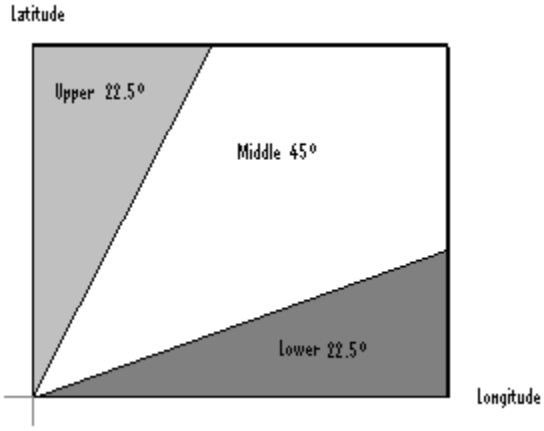


Figure 3: Destination-Vector Range

We can map the destination vector by finding in which quadrant it lies and what is the ratio between latitude and longitude. Let's go over an example to clarify the previous statement.

Suppose the destination vector is (1.0, -0.1). If we look in the quadrant diagram, the quadrant that has positive latitude and a negative longitude is the third quadrant. Then we take the ratio between latitude and longitude (absolute values):

Ratio = 1.0 / 0.1 = 10

This ratio is greater than 2.41 which is the tangent value at the dividing line between the upper 22.5 degrees and the middle 45 degrees. We automatically know it must be in the upper 22.5 degrees. Taking the quadrant as 3, the direction must be north. Now suppose the user is looking to the south, the calculation would go like this:

$$dir\_to\_move = \left((dest\_dir - compass\_dir) + 8\right) \bmod 8$$
$$dir\_to\_move = ((north - south) + 8) \bmod 8$$
$$dir\_to\_move = ((0 - 4) + 8) \bmod 8$$
$$dir\_to\_move = 4 = south$$

This means that the user must turn 180 degrees to find the destination location. A better example would be the vector (-1.0, 0.1), which is located in quadrant 1. The ratio is again 10, which means that it is in the upper 22.5 degrees in the destination-vector range. The direction is south for this quadrant. Now suppose the user is facing the south-west direction.

$$dir\_to\_move = \left((dest\_dir - compass\_dir) + 8\right) \bmod 8$$
$$dir\_to\_move = ((south - south\_west) + 8) \bmod 8$$
$$dir\_to\_move = ((4 - 3) + 8) \bmod 8$$
$$dir\_to\_move = 1 = north\_west$$

If the user is looking south_west he/she must then move straight and to the right to find the south.

3. Discussion

3.1 Block Diagram

The µtracker hand-held device can be viewed as a combination of 5 sub-system blocks. The hearth of the µtracker system is the 8052 microcontroller. The 8052 is in constant communication with the GPS and digital compass blocks. The above sub-systems are used to provide the tracking information that is used for our application. The GPS chip is used for the constant transmission of global coordinates whereas the digital compass provides the current cardinal coordinate that the user is directed at.

The interface block provides for the human interface with the device. Its main components are the LCD display and four push buttons. The 8052 controls the interface by providing the data transmission for the LCD and the interrupt handling of the pushbuttons. The battery-pack sub-system provides for the 5V and 3.3V voltage levels that are required by the devices.

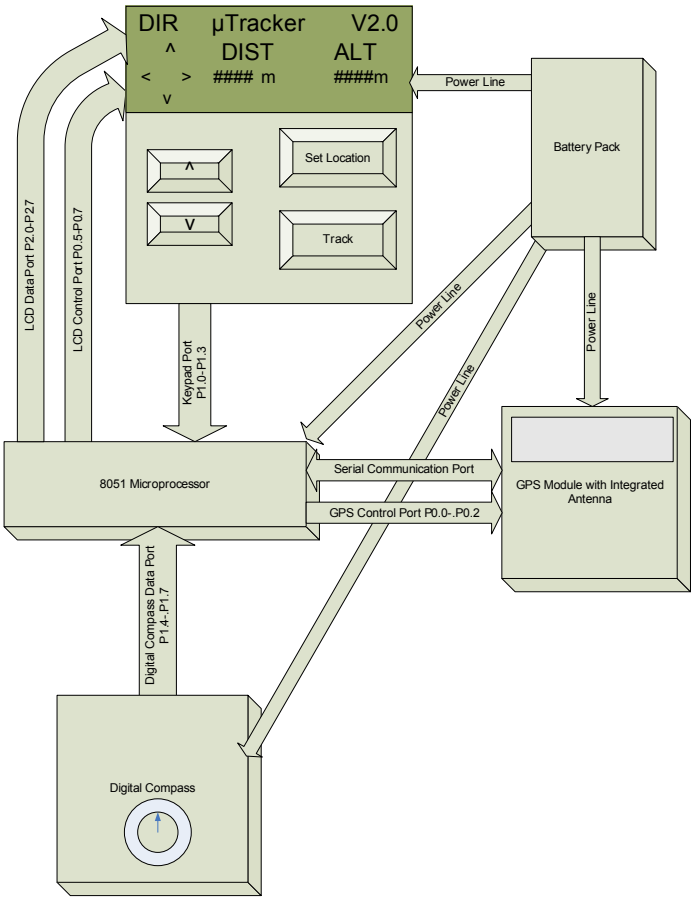


Figure 4: µTracker Block Diagram

3.2 Power Consumption

The power consumption of the tracking device is computed with the following devices. The analysis takes into account the minimum and maximum power requirements for each of the components.

Sony GXB5210
Operating voltage – 3.1 V – 3.7 V (Selection of 3.3 V)
Operating current – 50 – 70 mA (50 mA during tracking, 70mA during acquisition)
Backup Power – 3.3V @ 10 uA
Max Power – 259mW
Min Power during tracking – 165 mW
Min Power during acquisition – 231 mW
Max Power during standby mode – 0.033 mW

Atmel 80C51
Operating voltage – 2.7 – 5.5 V (Selection of 3.3 V)
Operating Current – 9.42 mA @ 11.059 MHz
Power Consumption normal mode – 31.09 mW
Power Consumption idle mode – 27.45 mW

LCD 4 x 20
Operating Voltage – 5 V
Operating Current – 10mA
Power Consumption – 50 mW

Digital Compass
Operating Voltage – 5 – 20 V (Selection of 5 V)
Operating Current – 30 mA
Power Consumption – 150 mW

The power consumption is calculated for each operating mode of the device. The system provides for four different operating modes: device off, standby, device on with track mode off, and device on with track mode on. The power consumption for each operating mode is calculated next.

- **Device OFF**
  All components are turned off.

  $$\text{Total Power} = P_{\text{Sony GXB5210}} + P_{\text{Intel 8052}} + P_{\text{LCD 4x20}} + P_{\text{Digital Compass}}$$
  $$= 0 \text{ mW} + 0 \text{ mW} + 0 \text{ mW} + 0 \text{ mW}$$

  Total Power = 0 mW

- **Standby**
  After 15 seconds of inactivity the device is switched to standby mode. This mode is only reachable if the device is On with track mode Off. During this mode the GPS chip is in standby and the LCD Screen and Digital Compass are switched off.

  $$\text{Total Power} = P_{\text{Sony GXB5210}} + P_{\text{Intel 8052}} + P_{\text{LCD 4x20}} + P_{\text{Digital Compass}}$$
  $$= 0.033 \text{ mW} + 31.09\text{mW} + 0 \text{ mW} + 0 \text{ mW}$$

  Total Power = 31.123 mW

- **Device On with track mode Off**
  The system is on but the position information is updated every minute. The GPS chip is in standby until the information is requested by the timer. The Digital Compass chip is turned off.

  $$\text{Total Power} = P_{\text{Sony GXB5210 (signal acquisition)}} + P_{\text{Intel 8052}} + P_{\text{LCD 4x20}} + P_{\text{Digital Compass}}$$
  $$= 231 \text{ mW} + 31.09 \text{ mW} + 50 \text{ mW} + 0 \text{ mW}$$

  Total Power = 211.09 mW

- **Device On with track mode On**
  The system is on and updating the position information every 2 seconds. The GPS and Digital Compass chips are on during the operation.

  $$\text{Total Power} = P_{\text{Sony GXB5210 (Tracking Mode)}} + P_{\text{Intel 8052}} + P_{\text{LCD 4x20}} + P_{\text{Digital Compass}}$$
  $$= 165 \text{ mW} + 31.09 \text{ mW} + 50 \text{ mW} + 150 \text{ mW}$$

  Total Power = 396.09 mW

The maximum power consumption of the device occurs when the device is set to Tack Mode, in which case, every component is on for the tracking operation. Given the previous analysis, the maximum power consumption of the device is calculated as 396.09mW.

The calculation of the approximate running time with batteries is based on the maximum power consumption of the components and the battery's mAh capacity.

Battery - 5 AA Duracell MN1500 (2850mAh)

Runtime in Track Mode On .
Power Comsumption =396.09 mW
Aproximate Amperage = 50 + 9.42 + 10 + 30mA = 99.42 mA
Time = 2850 mAh / 99.42 mA = 28.66 hours

Runtime in Track Mode Off.
Power Comsumption =211.09 mW
Aproximate Amperage = 50 + 9.42 + 10 mA = 69.42 mA
Time = 2850 mAh / 69.42 mA = 41.05 hours

Runtime in Standby Mode.
Power Comsumption =31.123 mW
Aproximate Amperage = 50 +9.42 mA = 59.42 mA
Time = 2850 mAh / 59.42mA = 47.96 hours

3.3 Timing Analysis

The minimum working frequency of the system will be determined by analyzing the reaction time requirements for each peripheral. The system consist of: 8052 microcontroller, Sony GXB5210 single chip GPS receiver, Dinsmore 1490 Digital Compass, LCD Display and four push buttons for general input. The LCD Display reacts to inputs only and doesn't send any acknowledge of data received. It is assumed to react whiting the specified reaction times for each internal operation.

The digital compass has the timing limitations of a regular water-based compass. It provides continuous data feedback, but doesn't show any significant changes in the output in less than 2 seconds. We can conclude that even if the compass needs a continuous data request for direction information, it won't be a limiting factor in the timing of the whole system.

The GXB5210 chip is the only chip that can be considered as a limiting factor in the timing analysis of the whole system. The GPS chip can be configured to have a baud rate of 4800, 9600, 19200 or 38400. It provides an output of 8 different NMEA standard messages with the fastest frequency working at 1Hz each. However, our system will only use one of these messages to get all the necessary data. With an output rate of 4800 baud, the chip can keep up to 6 of the 8 different output messages with its fastest frequency of 1Hz. That being know, we're using the 4800 baud rate as the transmission rate of the serial communication of our system, since it is the minimum necessary for it to work and it's the less power consuming option.

An 11 MHz crystal can be used to generate the appropriate UART frequency of 8052 microcontroller. Using mode 1 of the serial communication port, the timer TH1 must be set to 244 for us to be able to set the baud rate of our system to 4800.

The following are the timing diagrams for the serial communication interface, LCD data, track and set button operations.

3.3.1 Serial Communication

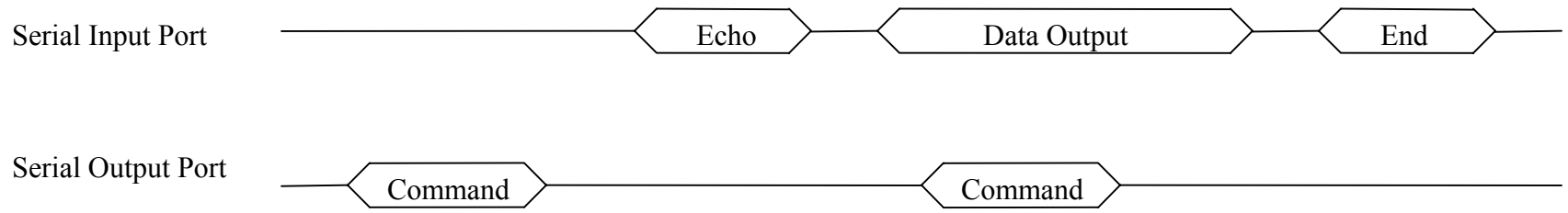


Figure 5: Timing interaction of multiple serial port commands in succession

3.3.2 LCD Data

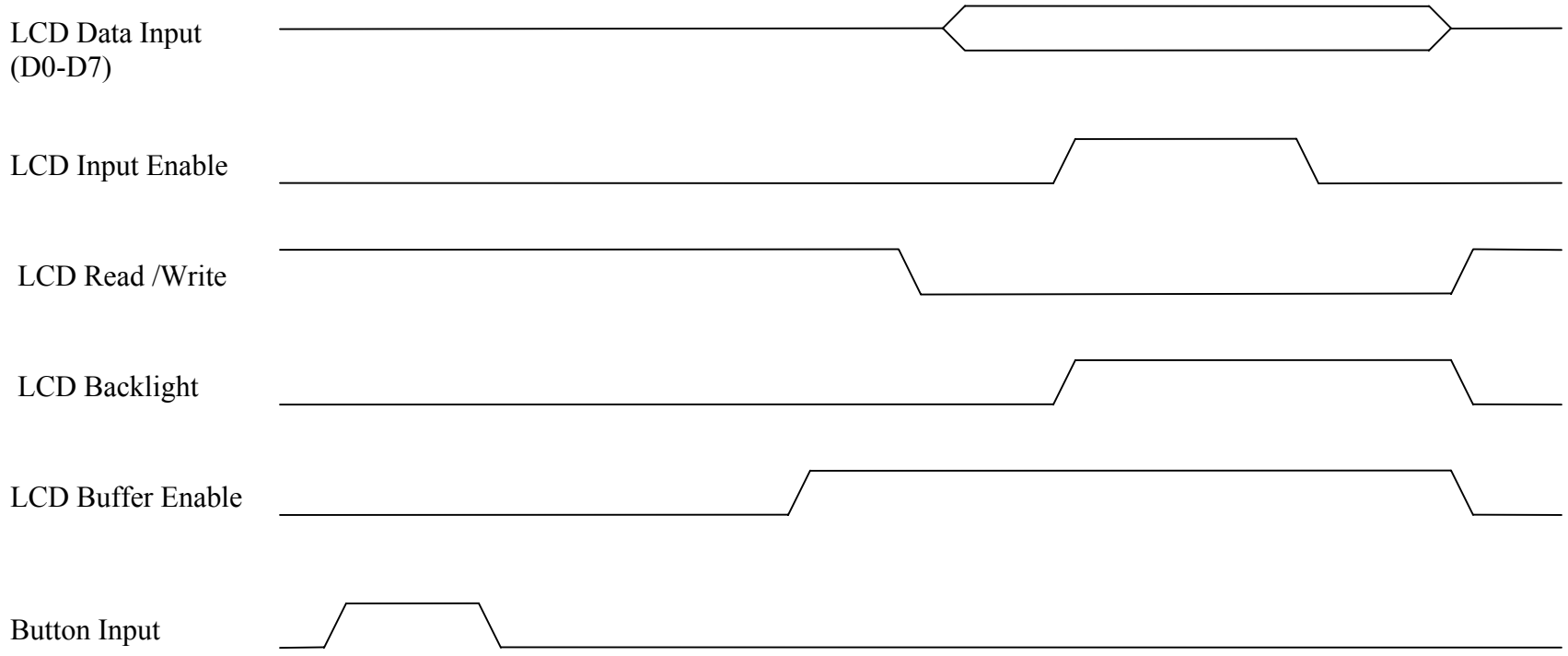


Figure 6: Data output and interaction with LCD display when a button is pressed

3.3.3 Track Button Pressed

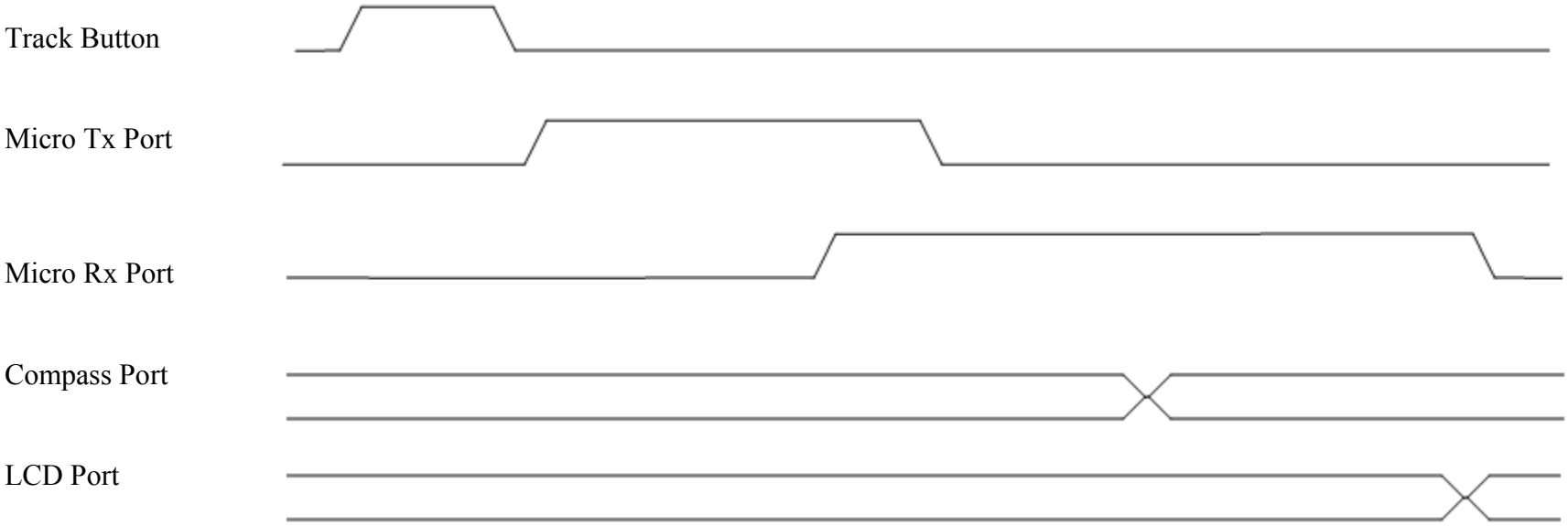


Figure 7: Timing interaction of the tracking command operation

3.3.4 Set Button Pressed

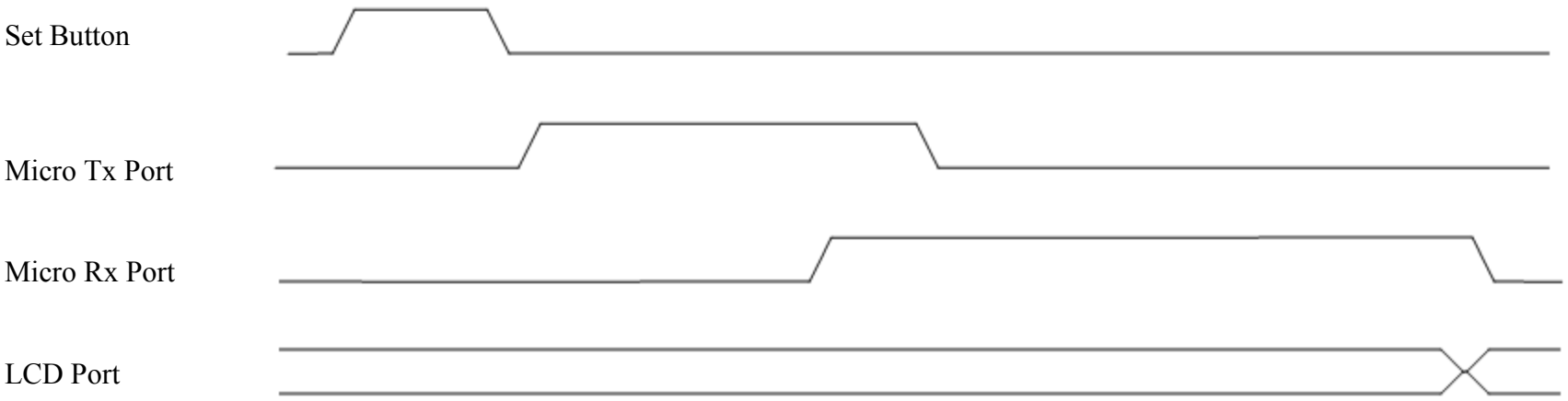


Figure 8: Timing interaction of the set command operation

3.4 Memory Map

The internal memory of the AT89C51RC2 microcontroller is organized in the following manner.

The AT89C51RC2 has the following memory areas:

- User memory area 16 - 32 KB size
- ROM bootloader memory 2 KB size
- Hardware security byte for configuration information and security levels.
- XAF area for ISP:
  - Boot Status Byte (BSB)
  - Software Boot Vector (SBV)
  - Software Security Byte (SSB)

Notes:
1. Refer to (A)T89C51RC2/RB2 or (A)T89C51IC2 datasheets and boot-loader datasheets for HSB and XAF description
2. As boot-loader is in ROM memory, no erase or write action is possible on this area.

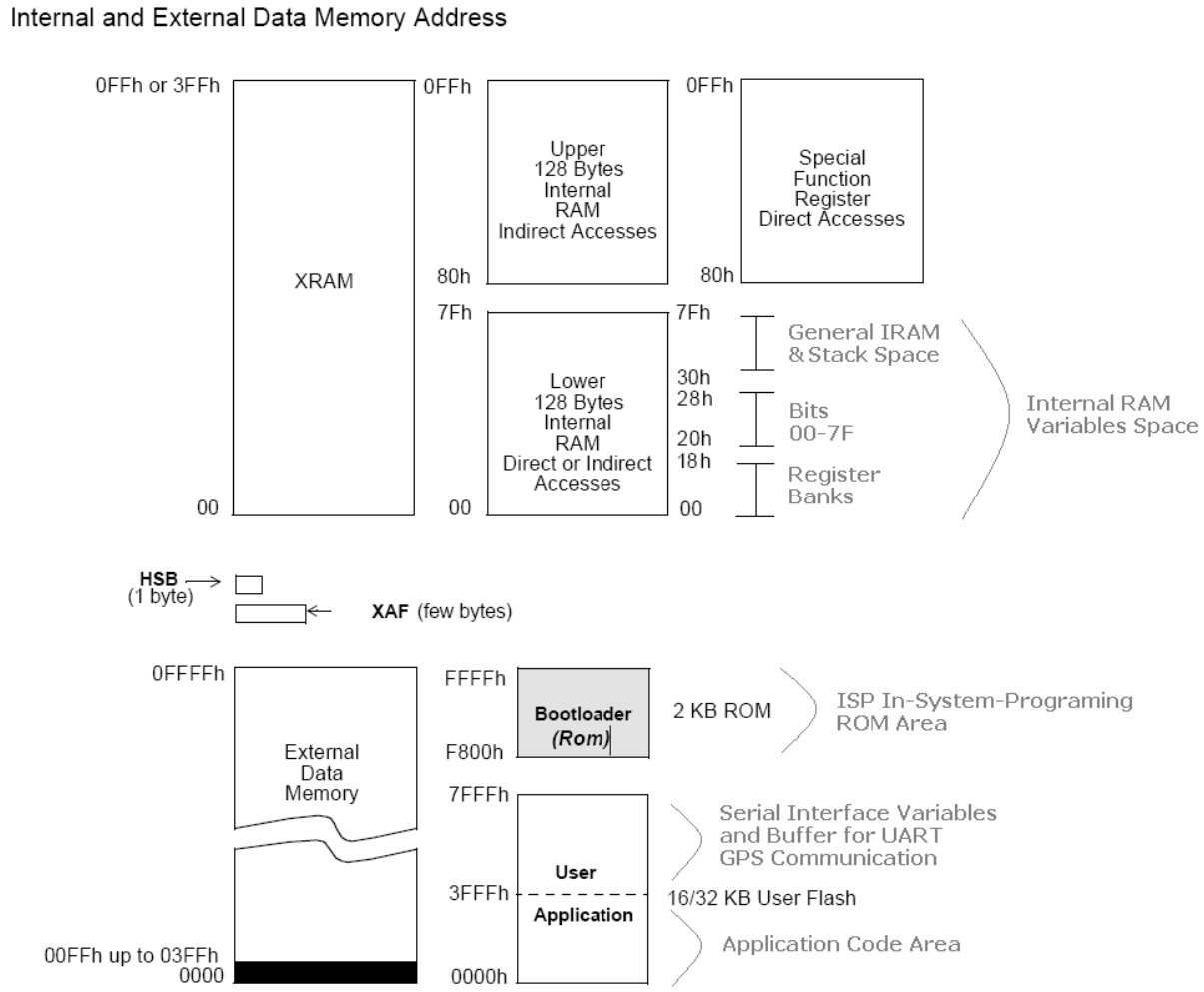


Figure 9: AT89C51RC2 Memory Mapping

The code for our application covers both the internal (00h-0FFh) and external memory areas (00FFh-0FFFFh).

The internal address space was divided as follows. The 8052 uses address space 00h to 28h as register bank and bits space. The area from 30h to 7Fh is used to store all the programming variables for our application, except for the serial buffering and parsing variables that require more address space than available. Address space from 80h to 0FFh is reserved on the 8052 for the special function registers.

The external address space is not completely accessible for user programming. The boot-loader ROM area (0F800h-0FFFFh) is used by the FLIP programming tool as an In System Programming tool for the UART port programming of the AT89C51RC2 that was used for the programming of the microcontroller. The serial interface variables for the UART data parsing and buffering cover the address space from 03FFFh to 7FFFh.

3.5 Hardware Schematic

The general schematic of the system is structured in the following manner.

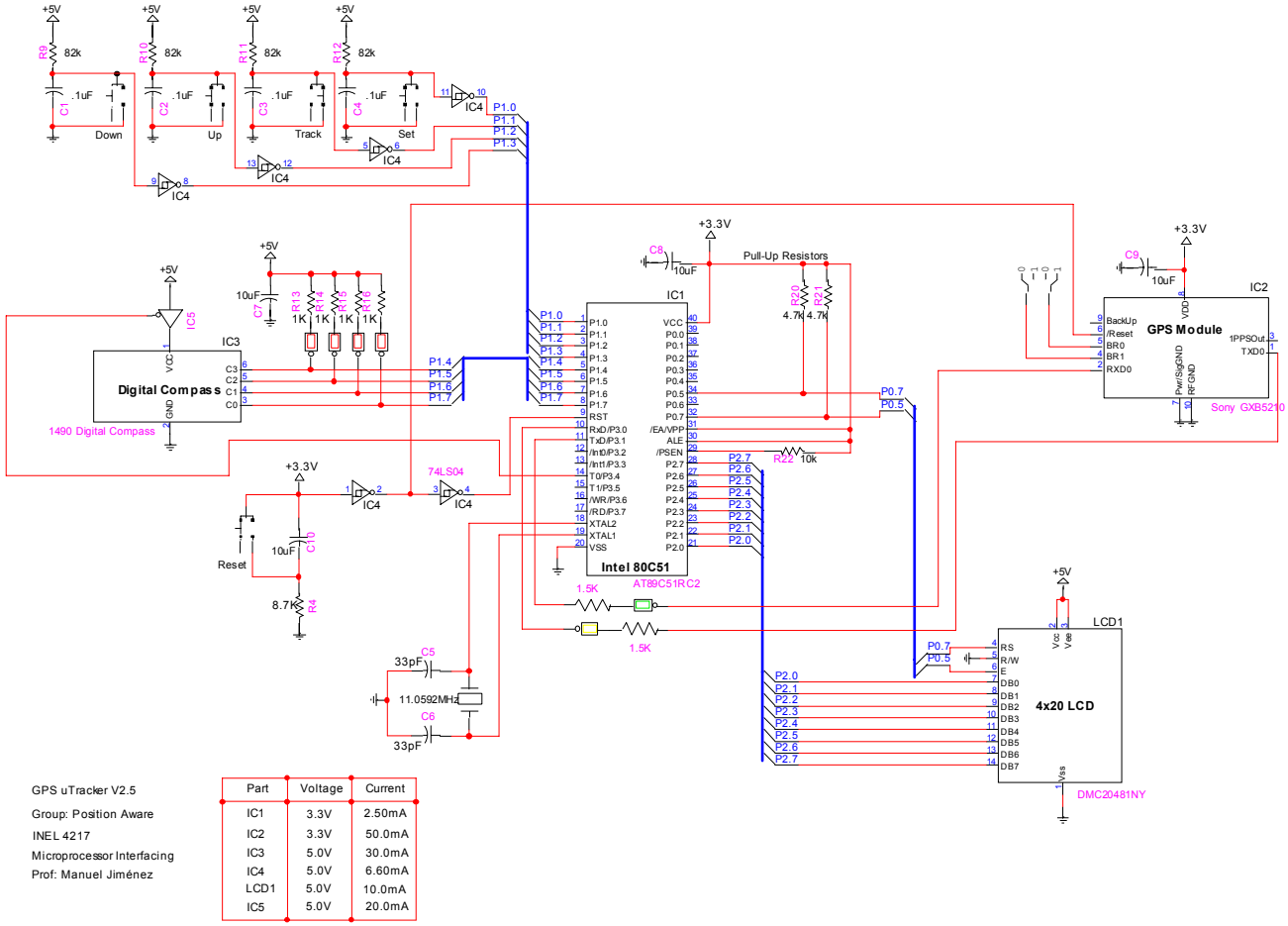


Figure 10: µTracker Hardware Schematic

The µtracker schematic shows the final hardware implementation of the tracking system. The diagram shows the interface for each of the components that where used for our application. A voltage source of 5 volts is required for the LCD and Digital compass. Similarly a source of 3.3 volts is required for the GPS and 8052 chips. Port 0 of the AT89C51RC2 require external pull up resistors in order to output TTL level signals. All other interfacing design-parameters where selected in order to comply with the manufacturers datasheets.

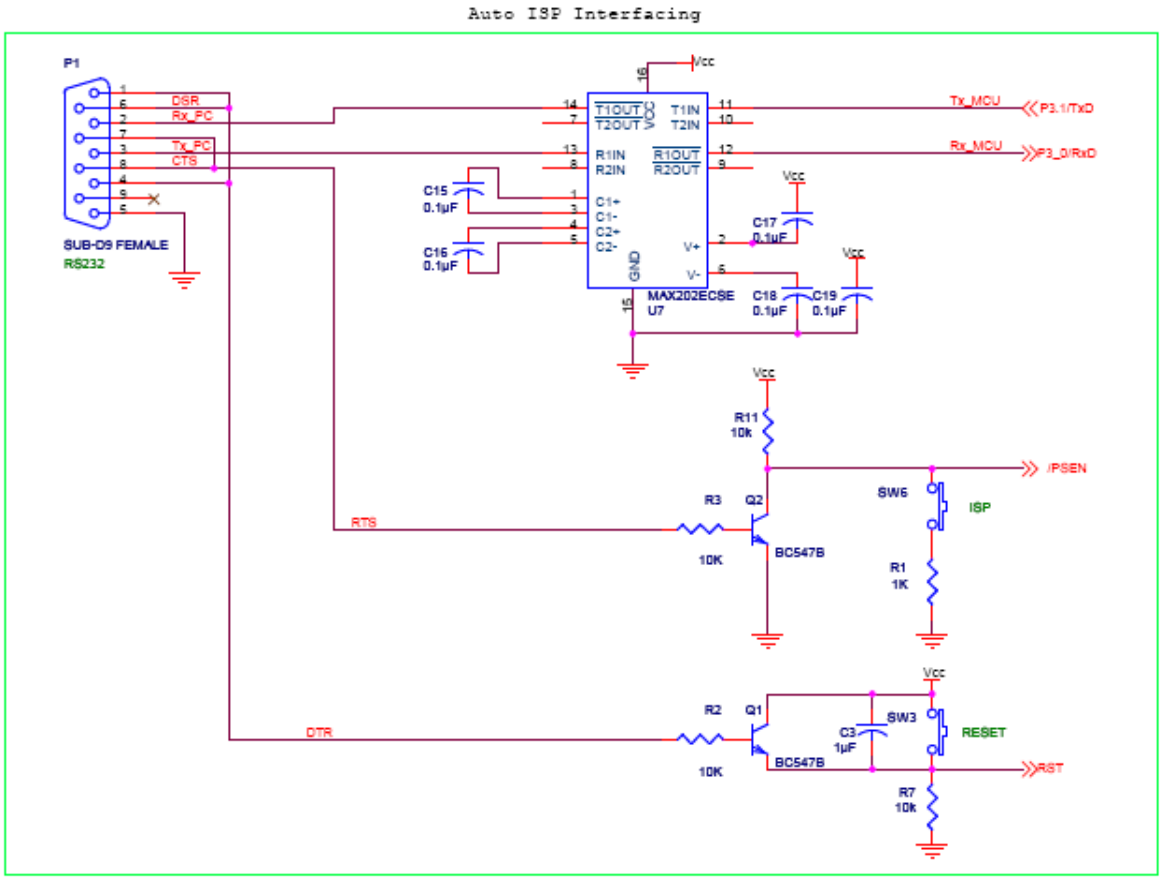


Figure 11: Atmel ISP Circuit for AT89c51RC2 UART Programming

The programming of the AT89C51RC2 required the Atmel's recommended ISP programming circuit. The programming circuit uses a straight serial cable to communicate with the microcontroller via Atmel's FLIP programming tool. Required parts for the circuit were: one MAX232 driver, five 0.1µF capacitor, one 10 µF capacitor instead of the 1 µF capacitor that is recommended, four 10 kΩ resistor, one 1 kΩ resistor for the discharge of the10 µF capacitor, two 2N3904 NPN transistors for the automatic RESET and PSEN signal handling. The final implementation of the programmer added three Schmitt trigger inverters for the correction of the programming RESET and PSEN signal.

Switch Button and Reset

The purpose of the switch button is to toggle the equipment between on and off. To realize its function, it only has to disconnect the circuit from the power supply. The circuit parameters where taken from Atmel's ISP programming schematic with a capacitor parameter modification from 1µF to 10µF. The Parameter modification was made in order to better the reset signal of the microprocessor. The reset signal drives the reset of both the 8052 and GPS chips via Schmitt trigger inverters to improve signal quality.

Push Buttons

The device has 4 push buttons among its components: SET, TRACK, UP and DOWN. The UP and DOWN buttons are used to move through the stored coordinates. The SET button is used to add new locations and the TRACK button is used to find a stored location. The push buttons where interfaced with a de-bouncing circuit.

$$V_{th} = V_{final}\left(1 - e^{-t/RC}\right)$$

Our calculations assume, $t = 5.68ms, V_{th} = 2.5V, V_{final} = 5V$. Using an 82kΩ resistor we can use a capacitor of 0.1µF. The signal is interfaced via Schmitt trigger inverters to improve signal quality.

Compass

To calculate the direction the user must follow to find its target location, we need two types of information: the direction of the target and the direction the user is facing. To obtain the direction of the target location, the GSP chip is used. To obtain the direction the user is facing a digital compass is used.

The digital compass interface follows the parameters and test schematic that was supplied with the datasheet.

LCD Display

To interact with the user the device will use the push buttons (for input) and the LCD display (for output). The device contains a 4 x 20 LCD display which is appropriate for the amount of information we are going to display.

The given LCD is based the HD44780U standard, which specifies a set of rules and commands that most of the character based LCD support. We are only going to use some of the capabilities of the LCD component, the necessary to display the information to the user.

The hardware interface to the LCD display is straight forward. The interface for the RS and E signals requited the addition of two 4.7kΩ pull up resistors in order to drive TTL level signals to the LCD.

GPS Chip

The most complex of the device components is the GPS chip. The chip selected for this project is the Sony GXB5210 single GPS chip. This chip is a complete implementation of a GPS receiver; no external components are needed to make it perform its intended function. The final hardware implementation added two LEDs and 1.5kΩ resistors two limit the current threw the LEDS. The LEDs provide the appropriate feedback in order to debug UART signal errors.

11.0592 MHz Crystal

The crystal interface follows the standard parameters for the 8052 interfacing that are specified in the AT89C51RC2 datasheet.

3.6 Hardware Termination Level

The prototype developed in this project has all of the necessary components for it to be a functional device. We still wanted more features to be added to the prototype. The unit needs to have a backup battery of 3V added to the GPS receiver in order to reduce the time for the receiver to triangulate the initial position. The receiver would then be able to have in memory all of the almanac data from the last position it received before it was turned off.

Another important feature we wanted to add to this prototype is to have the ISP programming hardware integrated. By adding this we don't have to take out the microprocessor to have it programmed, the programming is done with out any unnecessary handling of the microprocessor. This will reduce the opportunity of damaging any of the static sensitive components.

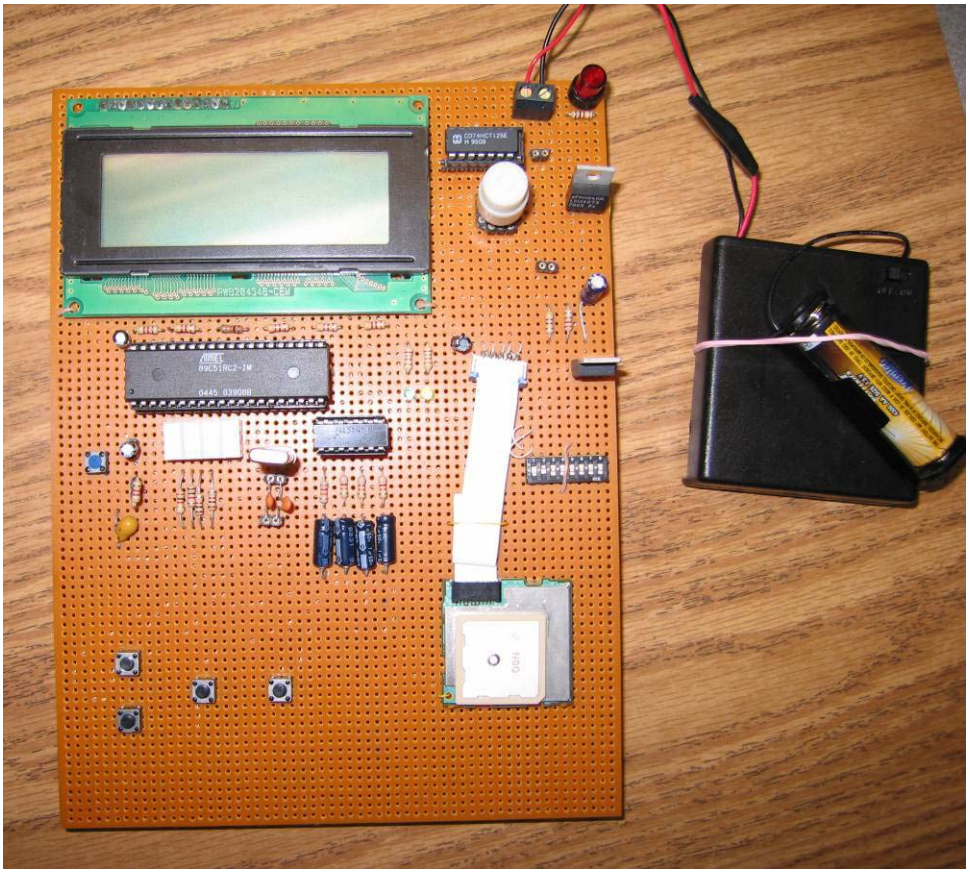


Figure 12. Prototype

3.7 Software Plan

3.7.1 General Usage

We proceed to describe a typical scenario of the tracking device's usage. Images of the device's display are provided at the end of this section. The images provide a visual description for each state of the process. Note that the format of the display is free to change in the near future, but the general functionality of the device will stay the same. The following scenario provides a general description of the device operation from the user's point of view. A complete description of the device's usage is presented in the next section.

We begin our scenario with the device in the off state. The first time the device is turned on, it will initialize and the first GPS coordinates would be received. At this stage, the device will be displaying a busy message to the user, indicating the device is preparing to provide its required functionality. This stage will take a considerable amount of time (around 15 seconds), but later queries will be considerably faster (2 seconds). The initialization time would be used to setup the UART communication parameters and to stabilize the internal signals

The first coordinates will be stored in memory so that the user does not need to press the set button the first time the device is turned on. If the user wants to set a new location, he/she will need to press the set button to obtain the new coordinates. Note that the device does not need to be turned off; this operation is- only done to increase battery life. In the case that device was active prior to the moment of adding a new location; the user will have to press the set button to add the current coordinates to the list of tracking locations.

At the time the user wants to find the stored location, she will press the track button. The device will present a list of the most recent stored locations (3 at this time) and the user will chose the appropriate coordinate using the scroll buttons (up and down) and press the track button again to begin the tracking process. The device will begin to give directions to the user until the user finds the target location. At this time the user will press the track button to deactivate the tracking mechanism.

The following figure shows the LCD interface of the device as it switches from each stage.

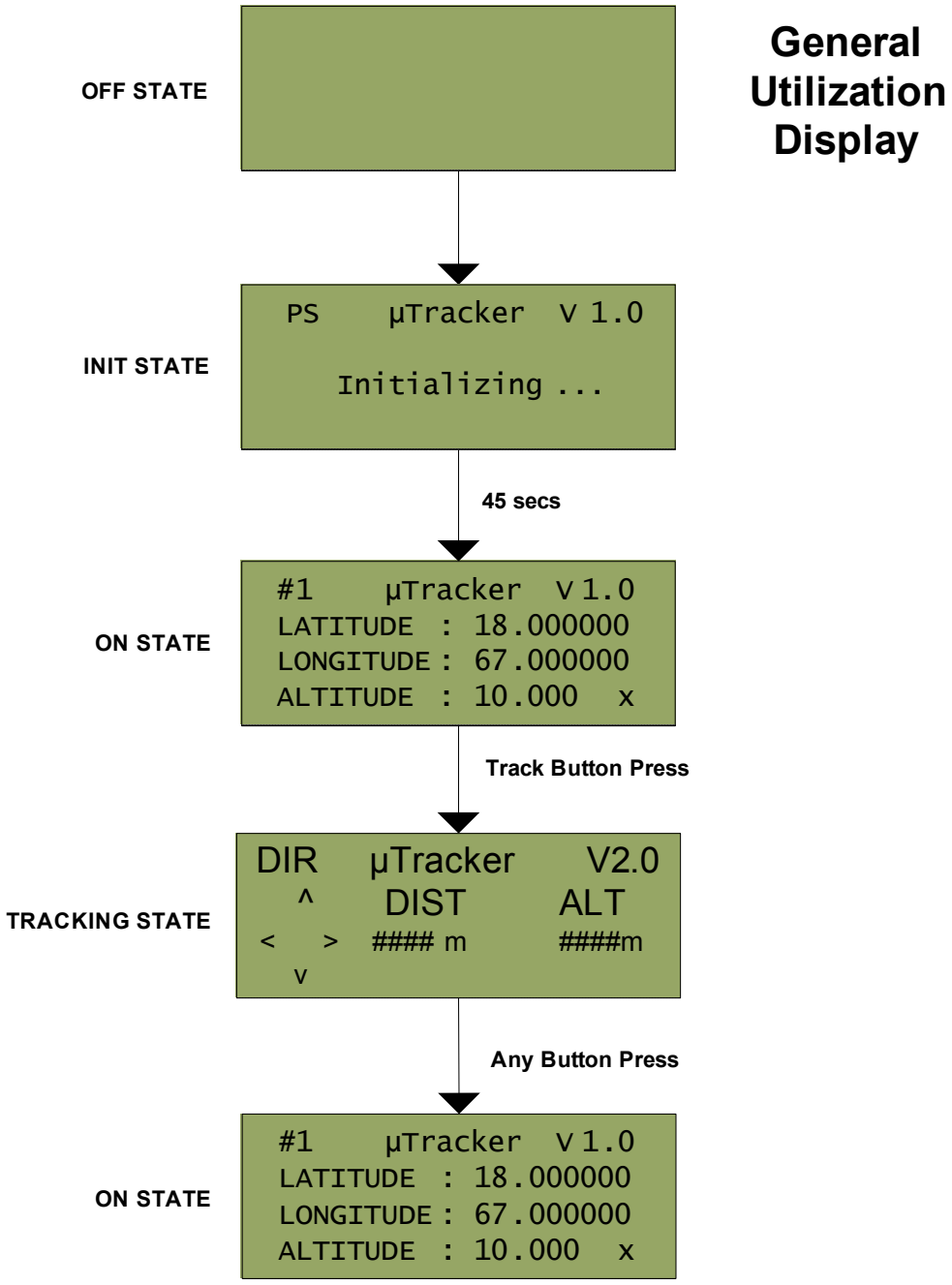


Figure 13: LCD Interface General Utilization

3.7.2 Complete Usage Description

In the previous section, we provided a general description of the device functionality. The purpose of this section is to provide a more detailed description, now from the point of view of the device itself. Many aspects of the device functionality were omitted from the previous section, because they are not really needed in a typical scenario. They are important features however and they will be explained here in more detail.

We now proceed to provide the complete description of the device's functionality, with all its features and modes of operation. Once again images of the device's display are provided at the end of this section. The images and written information complement to provide a complete description of the device.

We begin by giving a description of the device itself. The tracking device is composed of 6 items: a LCD display, 4 push buttons, 1 switch, an Atmel at89c51 microcontroller, a SONY GPS chip and a digital compass. The purpose of each of these components will be explained when they come up in the device's operation description. The tracking device can be in four states of operation: off, standby, on and track.

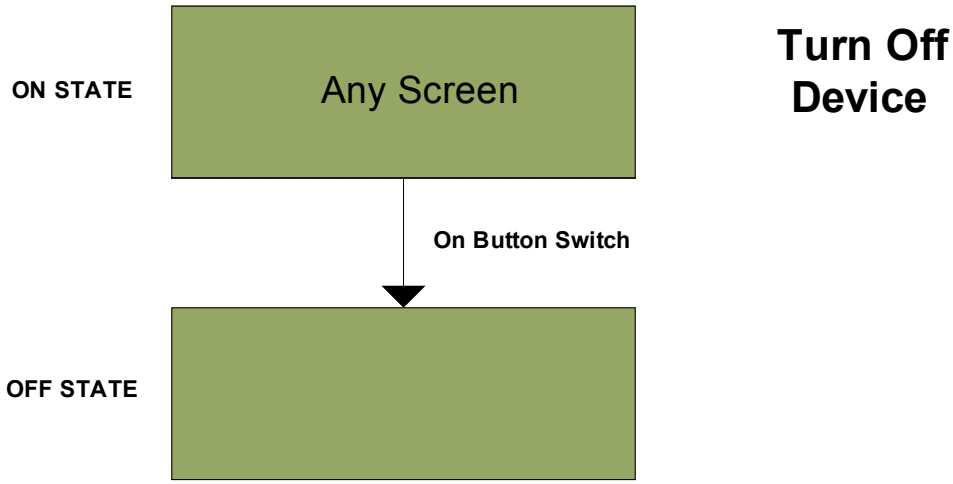


Figure 14: Off Mode Display

When the device is in the off state, all the components are turned off (they are not in standby). When the device enters this state, all the stored information is lost. The stored coordinates are deleted, so later attempts to find the stored locations will fail. In this state the device consumes no power (everything is off), which is the reason of providing it, the battery life will be extended if the device is in this state when the user is not using it. The system is taken into and out of this state using the switch component.

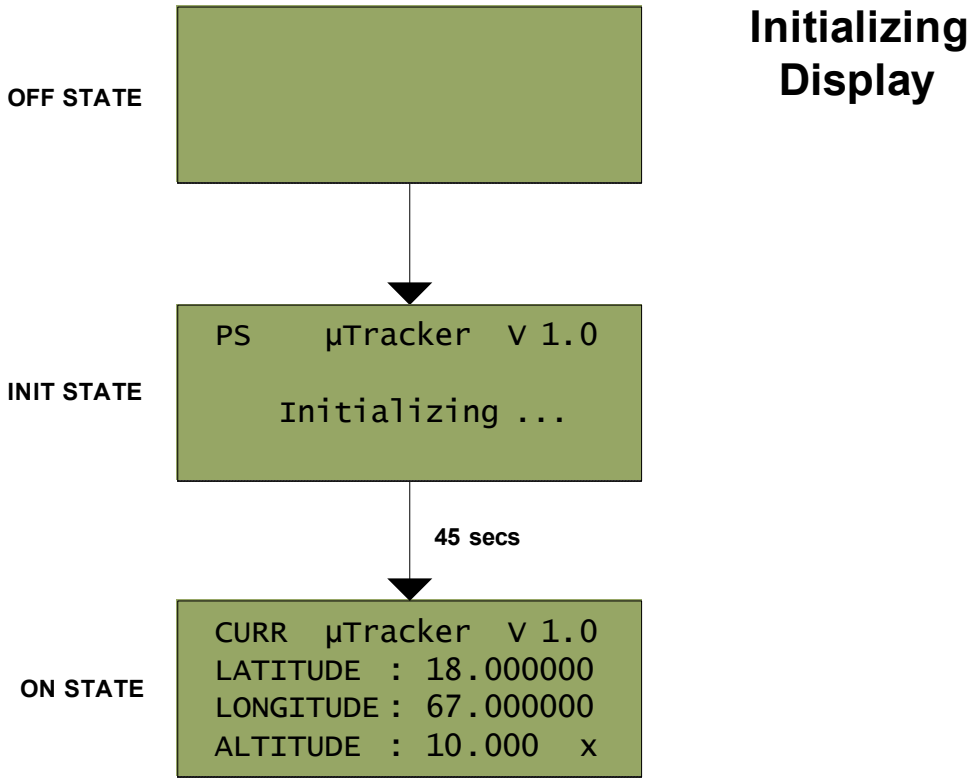


Figure 15: Initializing Display

The second mode of operation is standby. In this mode of operation, the microcontroller (Atmel at89c51) is in standby, the GPS chip is in low power mode and all the other components are off. This mode is entered when the device is on but it has been unutilized for 30 seconds or more. The stored coordinates are preserved, the GPS updates the coordinates every 10 seconds but the microcontroller does not process them. The coordinates are updated regularly to prevent the GPS chip to go into the warm state. When the GPS chip enters the warm state, later coordinates queries will last around 30 seconds, instead of the 2 seconds when the device is in the hot state.

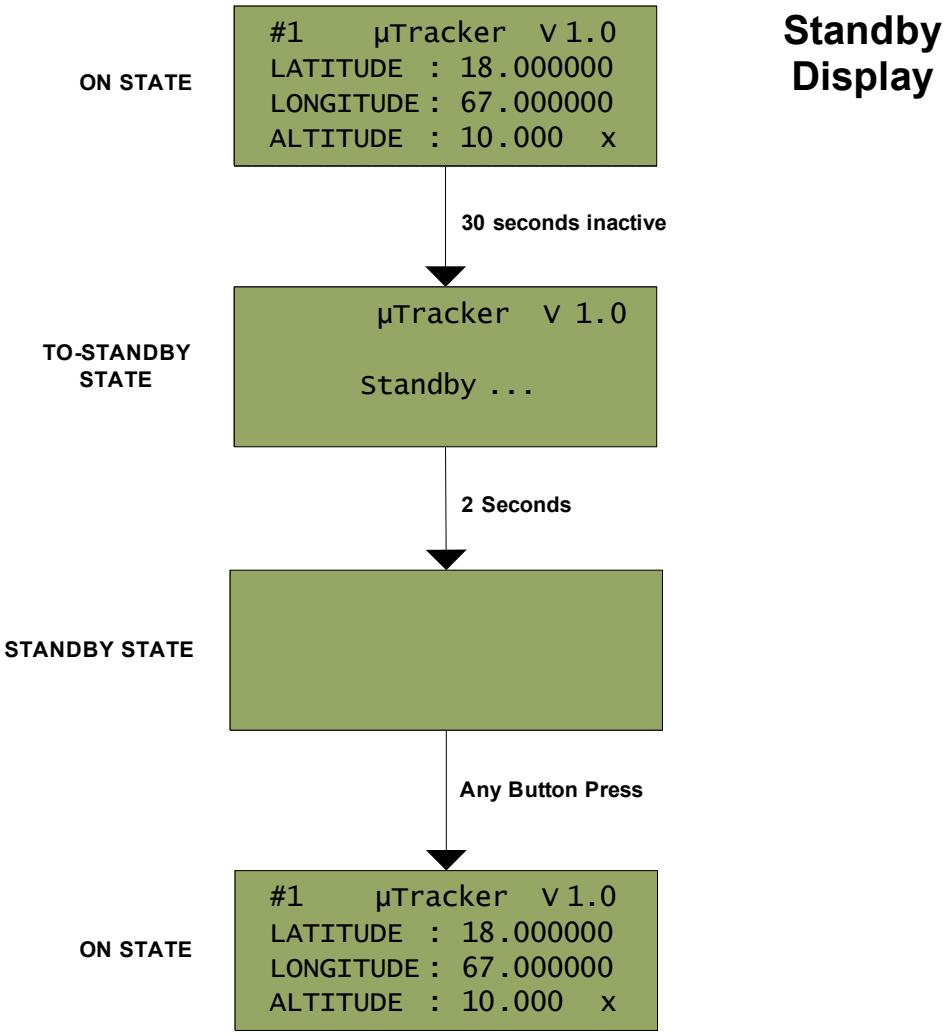


Figure 16: Standby Display

If any button is pressed the device will go into the On state. In this mode of operation, the microcontroller is on, the LCD display is on, the compass is off and the GPS chip is in active state, updating the coordinates every 2 seconds.

The user will be able to interact with the device in the on state, which makes the LCD necessary. Note the compass is off in this state because it is not needed. When the user presses the track button, the device enters in the last state: the tracking state. In this mode of operation all the components are on, including the digital compass. The GPS chip will be receiving new coordinates every 2 seconds and the direction will be calculated according to direction of the user, which is why the compass is turned on.

Now that we understand every mode of operation, we can go through all the functionality of the device and describe each feature in detail. The description provided here is general; we state what can be done and how the user will do it. We say which device will be used in every action but we don't say how we are going to interface with it. That's the purpose of a later section: Component Interfacing.

When any of the push buttons is pressed, the user interface is shown. At the press of the button, the chip will go into on state and the list of coordinates will be displayed. At this screen the scroll buttons will be used to navigate through the stored and the current coordinates. When the correct coordinate is selected, the user has the option of setting a new coordinate or tracking one of the stored ones.

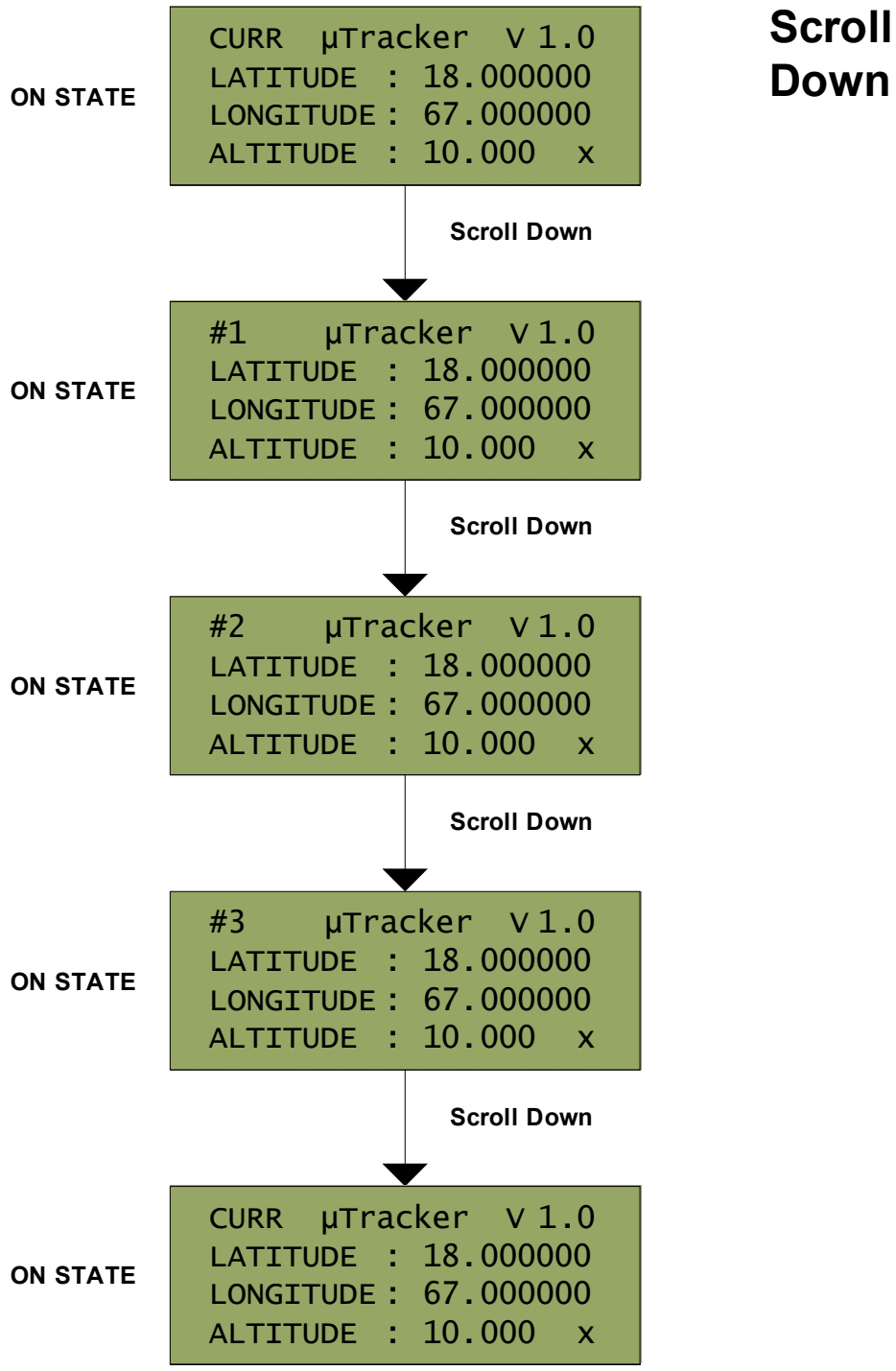


Figure 17: Scroll Down Display

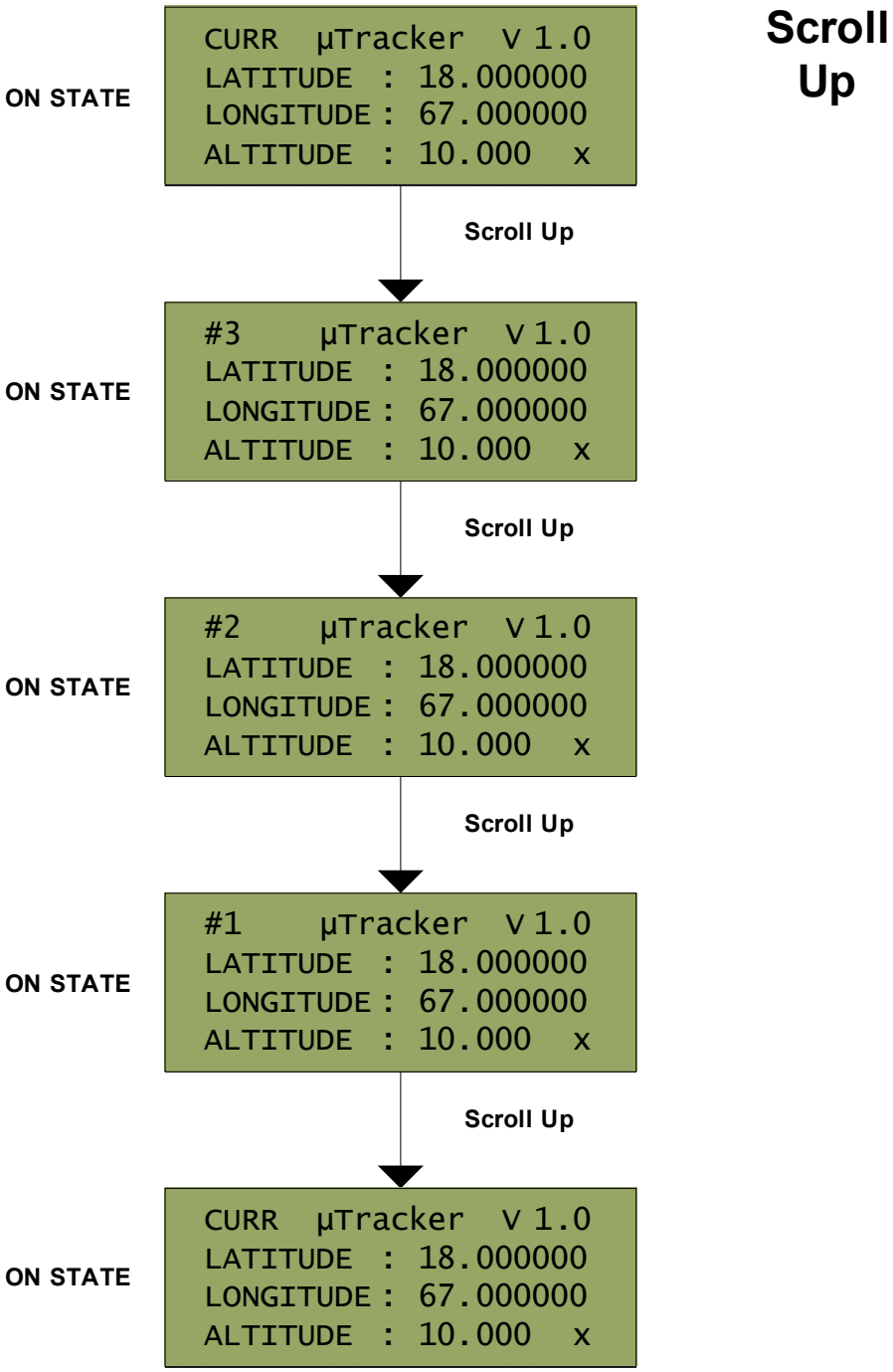


Figure 18: Scroll Up Display

When the user presses the set button, the selected coordinate will be replaced by the current coordinates. If the user presses the track button, the system will enter tracking state and it will track the selected coordinate. Note that if the user is looking at the current coordinates or the GPS signal is not present, the Set and Track buttons are disabled. An animation is show when the user tries to set or track coordinates when the GPS signal is not present.

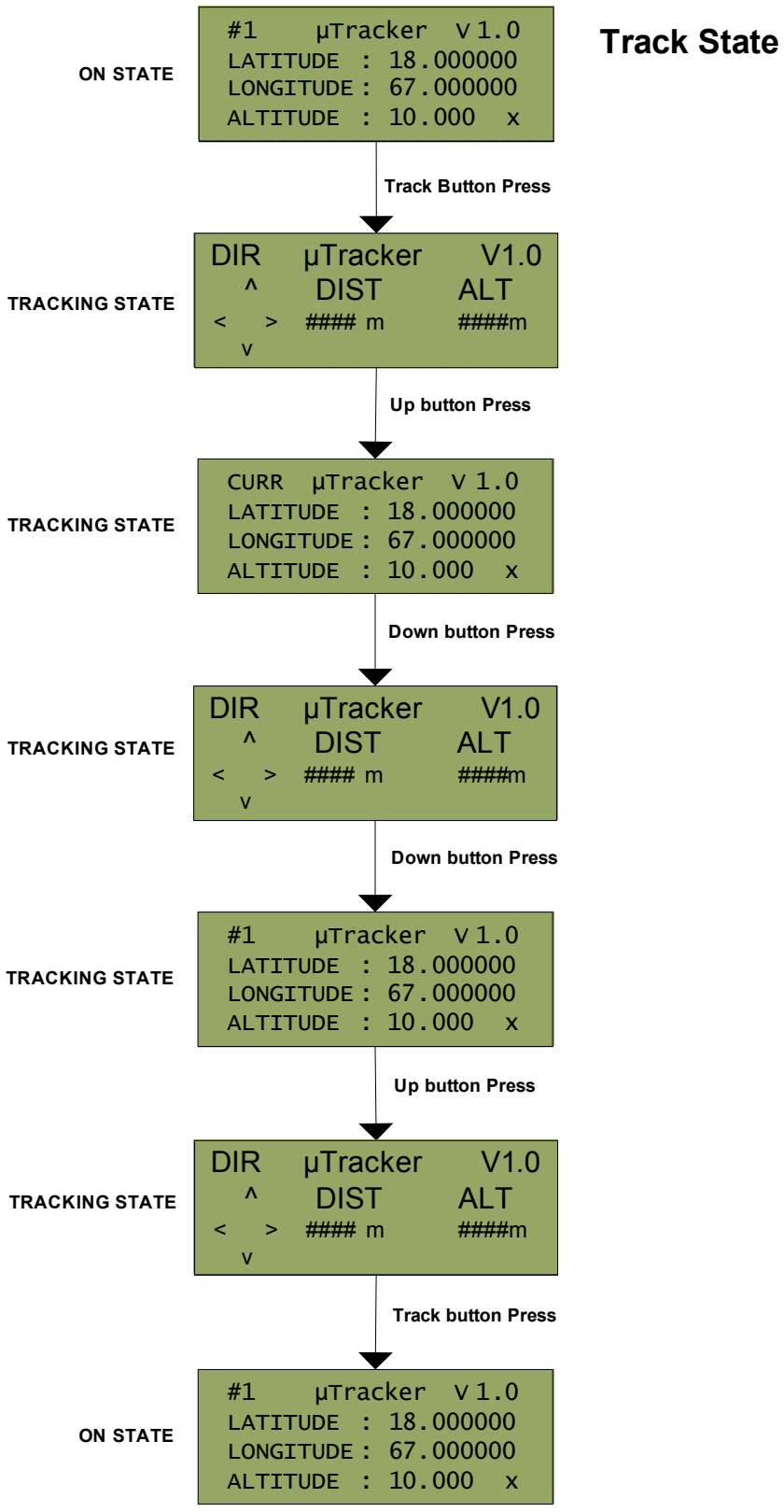


Figure 19: Track Mode Display

The device provides useful information while in track mode, which means that a different user interface is used. This interface shows the direction of the destination location, the distance and difference of altitude between the user and the target location.

If the user presses the up button while in track mode, the current coordinates are presented. If the user presses the down button the destination coordinates are presented. To stop tracking mode, press the Track button and device will go into On state again. When the device remains inactive for more than 30 seconds, it will enter standby mode.

3.7.3 System Flow Charts

The following section explains the operating flow charts for the operation of the µTracker device.

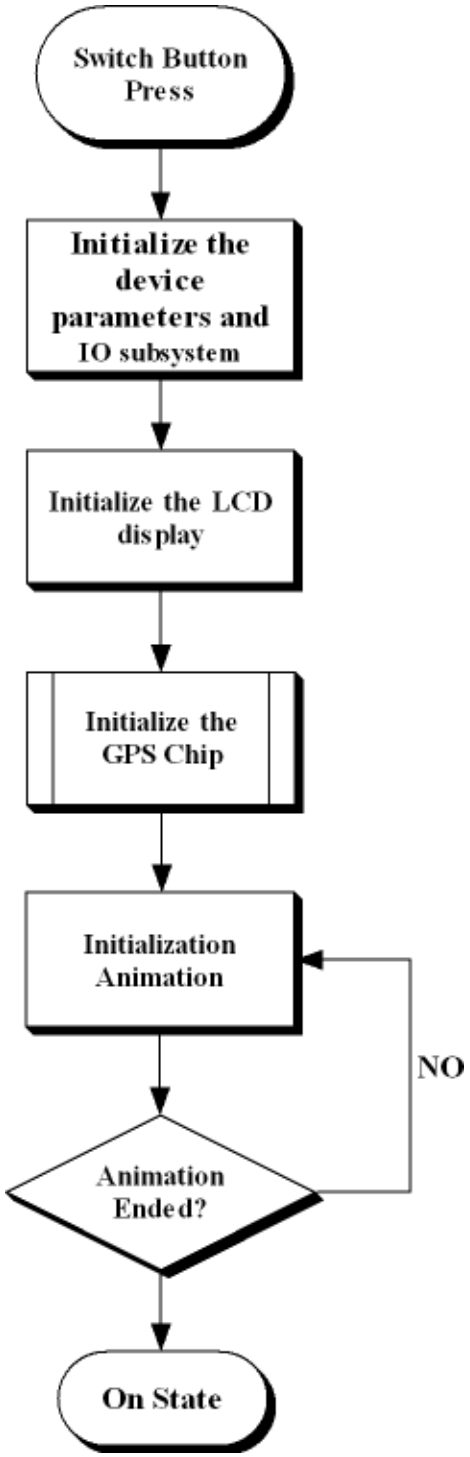


Figure 20: System Initialization Procedure

When the on/off switch is moved to the on position, power is applied to all the components. The microcontroller and GPS chip are reset by means of hardware. All the system variables are set to default values and the coordinates are set to an approximation of the current coordinate.

The IO system is then initialized. The serial port is programmed to give the appropriate baud rate and to generate interrupts for transmission and reception of characters. The transmission and reception buffers are initialized; and the GPS message parser is started.

Timer 0 is initialized to control all the timed events of the uTracker software, like animations and the standby count down timer. The ports are programmed for input or output (for input a one is written to the pin). The pins that receive the push button inputs are set to generate an interrupt when the button is pressed.

The next step is to initialize the LCD display, which requires a predefined sequence of command. After the LCD is initialized, the screen is cleared and the initialization screen is presented. This screen will be animated until the end of the initialization phase.

Finally the GPS chip is initialized. Below is a flowchart of the GPS chip initialization sequence, which requires the sending of two commands. The first command indicates to the GPS chip that we only need one of its messages, the GPGGA message, and that we want it every 2 seconds. Receiving the message every 2 seconds will allow us to parse the data before the next message is received.

The second command is used to provide an approximate location. This will allow the GPS chip to acquire the GPS signal faster. This is currently set to the Puerto Rico coordinates to allow fast reception of the location information. Finally the device waits for the GPS chip to receive the commands and goes to the On state.

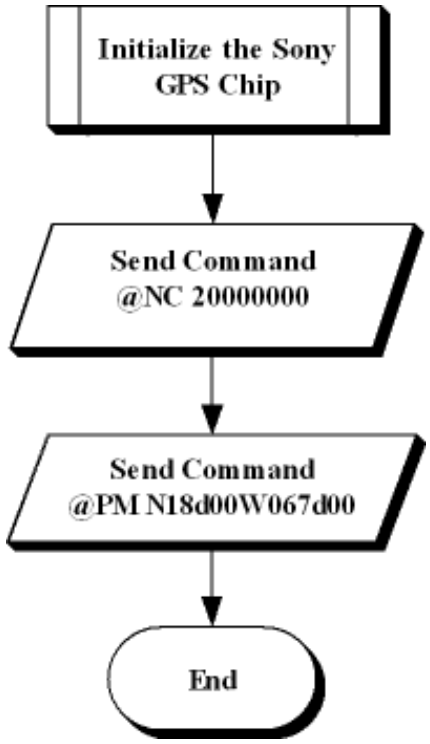


Figure 21: Sony GXB5210 Initialization Sequence

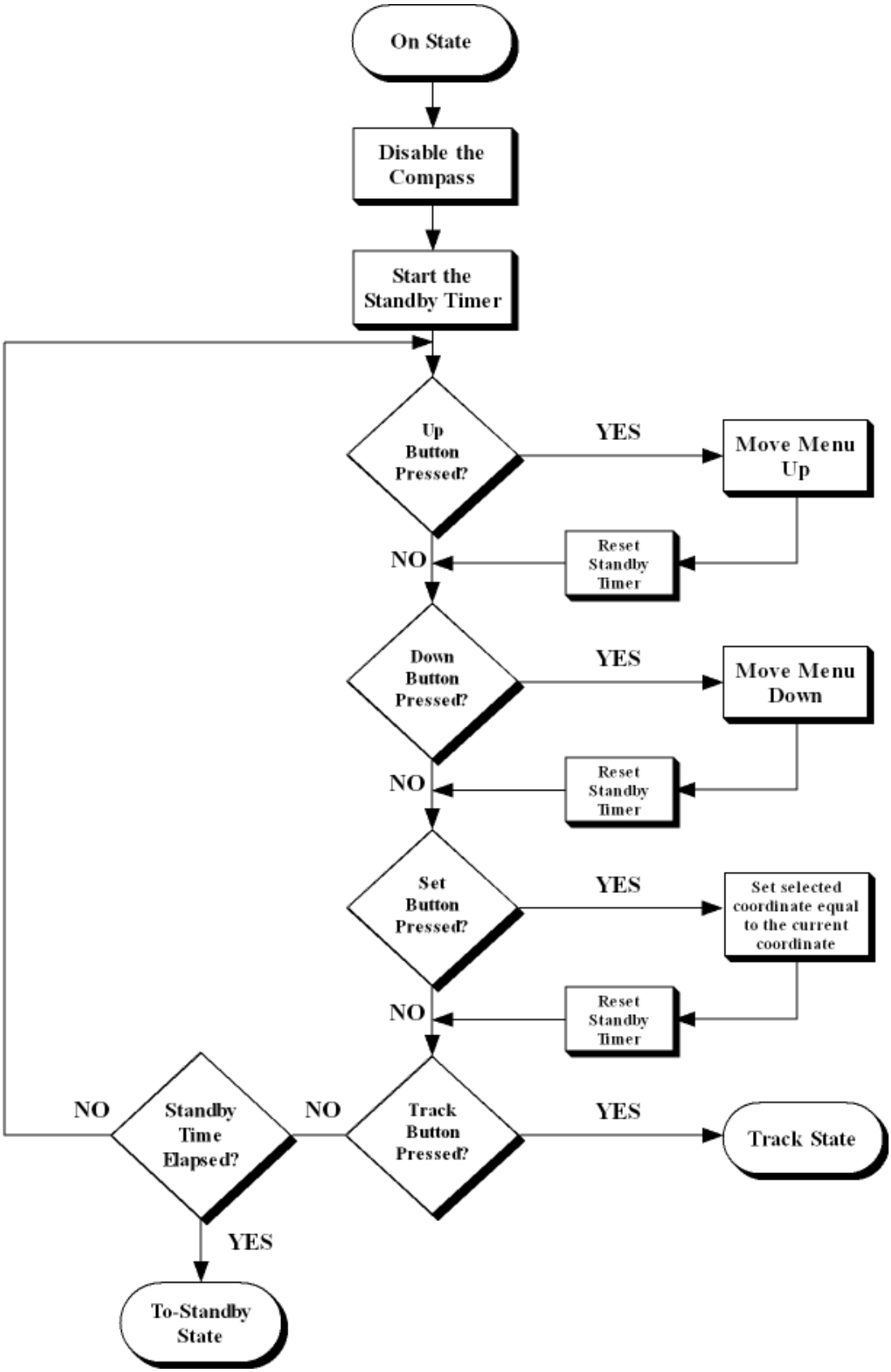


Figure 22: On State Operating Sequence

In the On State, the device provides the user a scrollable menu. In this menu he/she is able to check the current coordinates and three stored coordinates. The first time the On state is entered, the current coordinates will be displayed. The user can use the Up and Down buttons to move the menu up or down respectively.

When the user is looking at the current coordinates, he/she can press the Down button to see the first stored coordinate. If he/she presses the Down button a second time, the second stored coordinate will be displayed and the same applies for the third coordinate, pressing the Down button when looking at the third coordinate, will switch the display to the current coordinates menu again.

The functionality of the Up button is similar to the Down button, only that the menu scrolls up instead of down. If the Set button is pressed while selecting one of the stored coordinates, one of two things can happen.

If the GPS signal is present, the selected coordinate will be set to the current coordinates. If the GPS signal is not present, an animation of the GPS signal indicator will be displayed. Note that when the user is looking at the current coordinates, the Set button does not have any effect.

The Track button is used to go to the Track state, where the current selected coordinate will be tracked using the GPS chip and the compass. The functionality of the Track button is similar to the Set button. If the GPS signal is not present, the non signal animation will be shown and if the user is looking at the current coordinates, the Track button has no effect.

We can note at the beginning of the state, the compass is disabled and the standby count down timer is started. In this state the compass is not used, it is disabled to reduce power consumption. The count down timer will expire in 30 seconds. When any of the buttons is pressed the counter is reset. If the interval ends, the GPS chip is put in the To-Standby state.

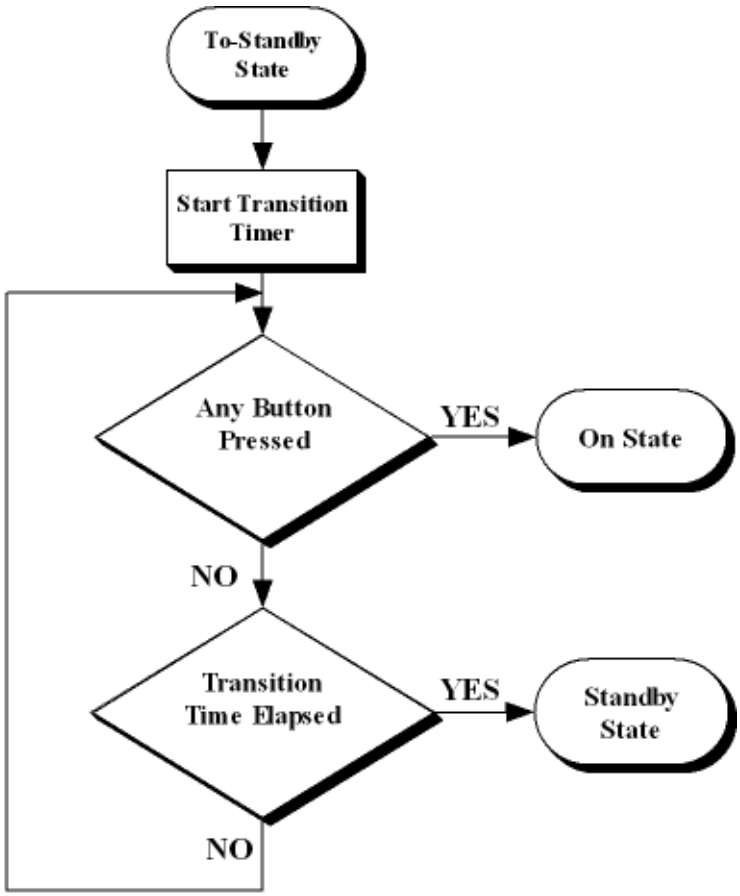


Figure 23: To Stand-By Operating Sequence

The To-Standby state is a very simple state. The device enters this state temporarily before going into the Standby state. If the user presses a button while this state is active, the device is switched to the On state. If at the end of a small interval of time, the user has not pressed any of the buttons, the system goes to the Standby state.

The main idea behind this state is to allow the user to cancel the Standby state. This will prevent the microcontroller and the GPS from going into standby. The state translates to the system switching to a less power consuming mode that will take a considerably amount of time to start again. A screen is presented to the user, when the device is in this state, so she knows the device is going into standby.

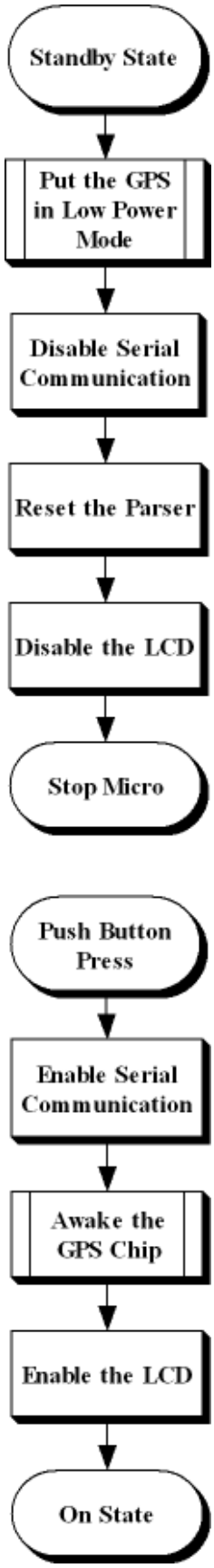


Figure 24: Stand-By Operating Sequence

The Standby state is different from the others in that it is not executed continuously while the device is in it. The Standby state is interrupted when the microcontroller is put in standby and it continues when the user presses any of the push buttons.

When going into Standby mode, a command is sent to the GPS chip to put it into low power mode. Below is a flowchart of the process of sending the command to the GPS chip. After the command is sent, serial communications are disabled. The parser is reset to eliminate any remaining parse data. The parser will start in a clean state when the microcontroller is awakened.

The LCD is disabled, to reduce power consumption. Finally, the microcontroller is put into low power mode. From this state it can only exit when a keyboard or external interrupt occurs. The device has nothing connected to the external interrupts, which means the microcontroller could only be awakened by a push button press.

When a push button is pressed, the microcontroller receives the keyboard interrupt (the keyboard interrupt will be explained in the microcontroller section). At this time the microcontroller is awakened and the interrupt is processed. Serial communications are again enabled and the GPS chip is awakened. The process to awaken the GPS chip is shown below.

It essentially consists of sending a command to the GPS chip, which indicates to the GPS chip to calculate the location at every moment. When the GPS chip was put in low power mode, the chip calculated the location every 10 seconds, now it calculates it every second.

The LCD is enabled and cleared to eliminate previous data. Then the chip is put into the On state again.

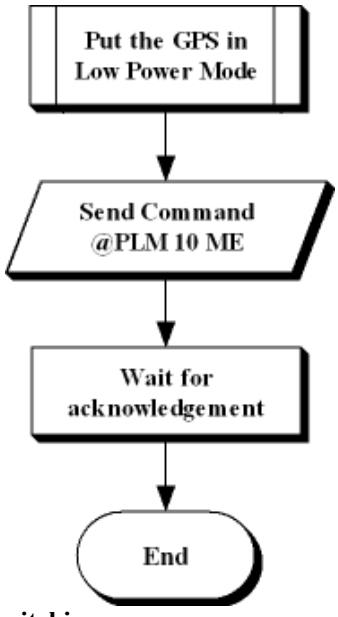


Figure 25: GPS Low Power Mode Switching

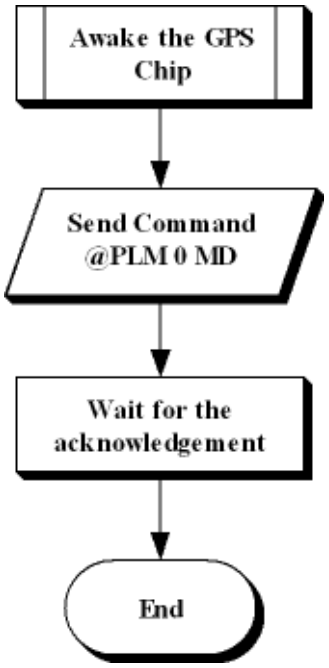


Figure 26: GPS Power Up Mode Switching

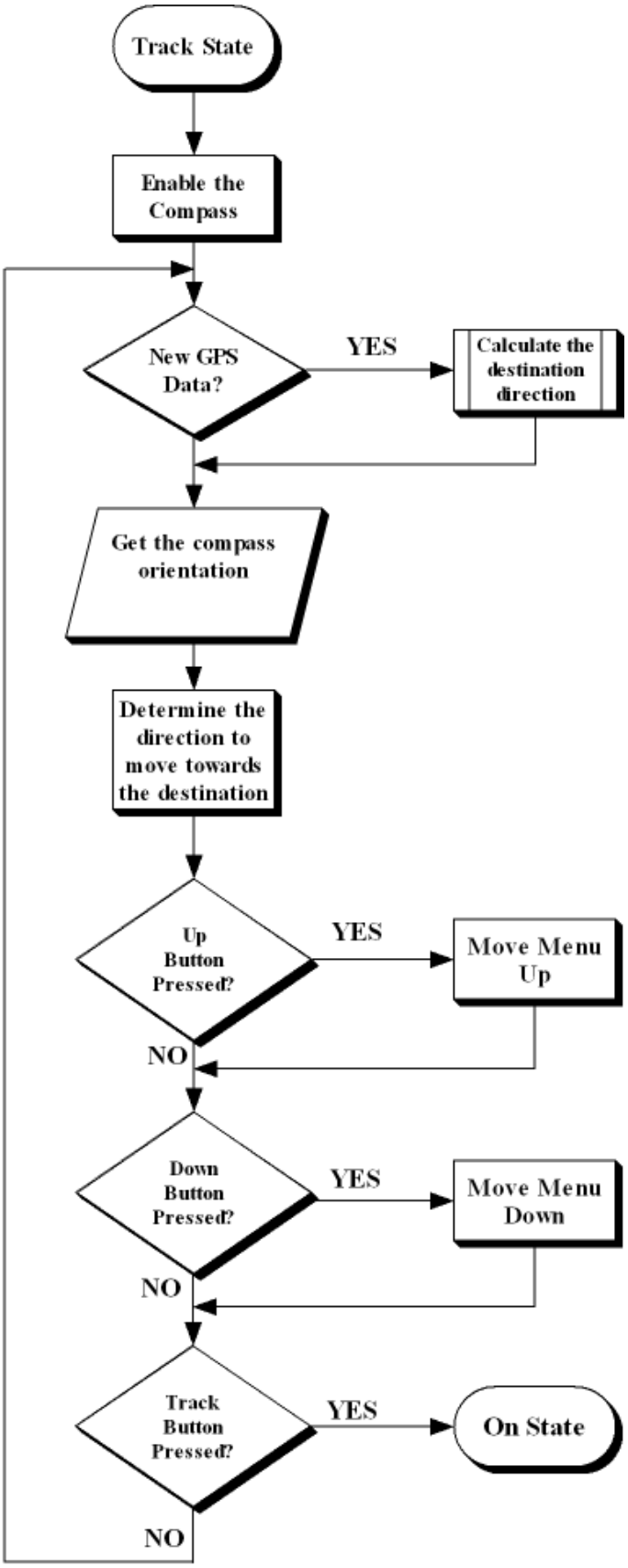


Figure 27: Track Mode Operating Sequence

At the beginning of this state, the compass is enabled. This is the only state where the compass is enabled, because it is necessary to calculate the direction to move towards the destination.

The direction vector is calculated every time that a new coordinate is received. A later section of this document explains this process in greater detail. Essentially we only map the destination direction when a new data arrives, but once every cycle we need to calculate the direction. This improves performance, because the calculations are only done when they are needed.

If the up button is pressed in this state, the device presents the current coordinates. Additional up button presses have no effect when the device is already presenting the current coordinates. To return to the track screen the down button must be pressed. In this screen if the down button is pressed the destination coordinates are presented. Additional down button presses have no effect when the device is already presenting the destination coordinates. To return to the track screen the up button must be pressed.

The set button has no effect in this state; Set button presses are simply ignored. When the track button is pressed, the device goes to the On state again. Below is the flowchart for the calculate direction subroutine.

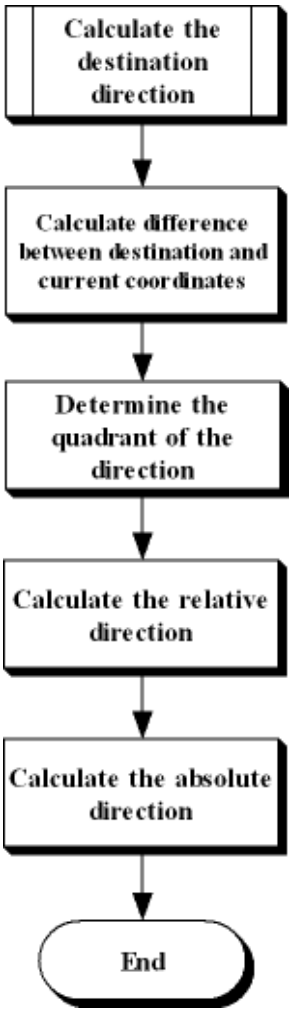


Figure 28: Destination Direction Calculation Sequence

The first step is to calculate the different between the destination and current locations. This will give us the destination vector. The next step is to calculate in which quadrant the vector resides and which of the three possible directions (inside a given quadrant) it is pointing at. Finally the absolute direction is calculated using the relative direction and the quadrant. See the theory section for more details about the functionality of each of these subroutines.

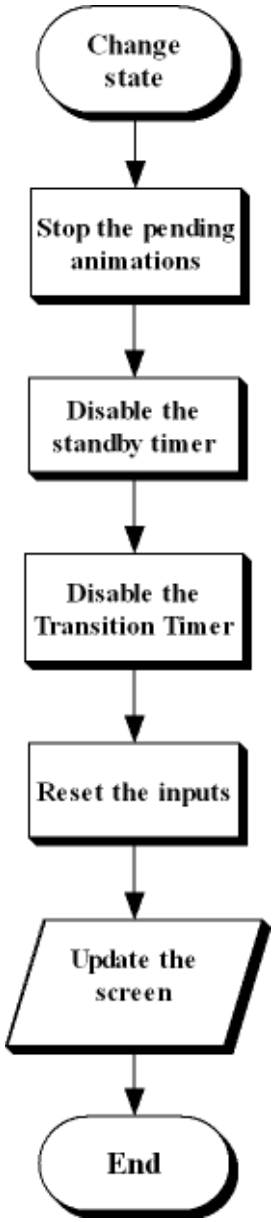


Figure 29: General Stage Changing Sequence

The preceding procedure is executed each time a change of state occurs. It is necessary because it resets many parameters to a default state. Animations and timers are reset so they do not conflict with the next state procedure.

The first step is to disable any pending animations, so they do not continue when going to the next state. The standby and transition timers are disabled, and are only enabled if the next state is the On or To-standby state respectively. The inputs are reset so no unhandled input is passed to the next state and the screen is updated to reflect the new state GUI.

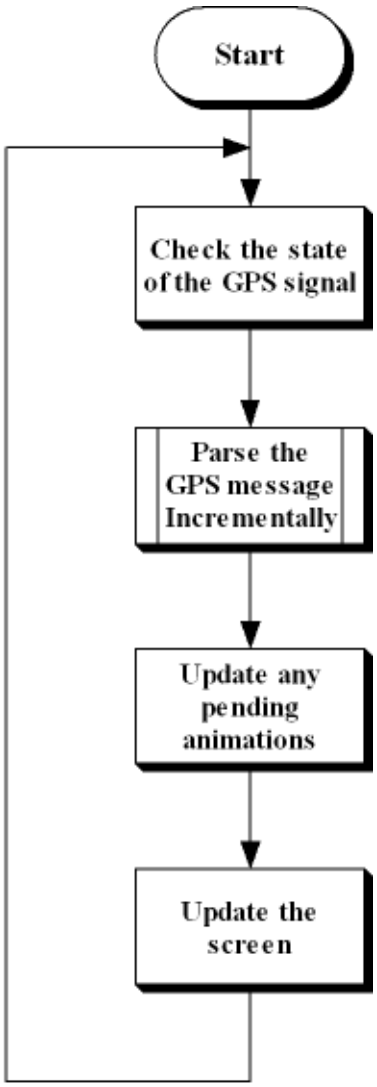


Figure 30: GPS Data-Parsing Sequence

For the On, Track, To-Standby and Initialization states, the above procedures are done in addition to the specific procedure of each of the states. They are presented here instead of presenting them in each of the states flow charts, to simplify the exposition of the information in those flow charts.

Basically the system is always checking the GPS signal and updating the screen. The screen is updated regularly but not too fast to prevent flashing of the screen. Any enabled animation is updated and presented to the user. Finally the system parses the part of the GPS message that has been received.

The GPS parsing is done incrementally. As each character is received, the parser is already parsing and checking the data for errors. This allows the parser to use the time between character receptions to process the information. Another approach would be to received the complete message and then parse it. But parsing incrementally is much more efficient, as it use the processor computing power to the maximum. Below is the flowchart of the parsing procedure:

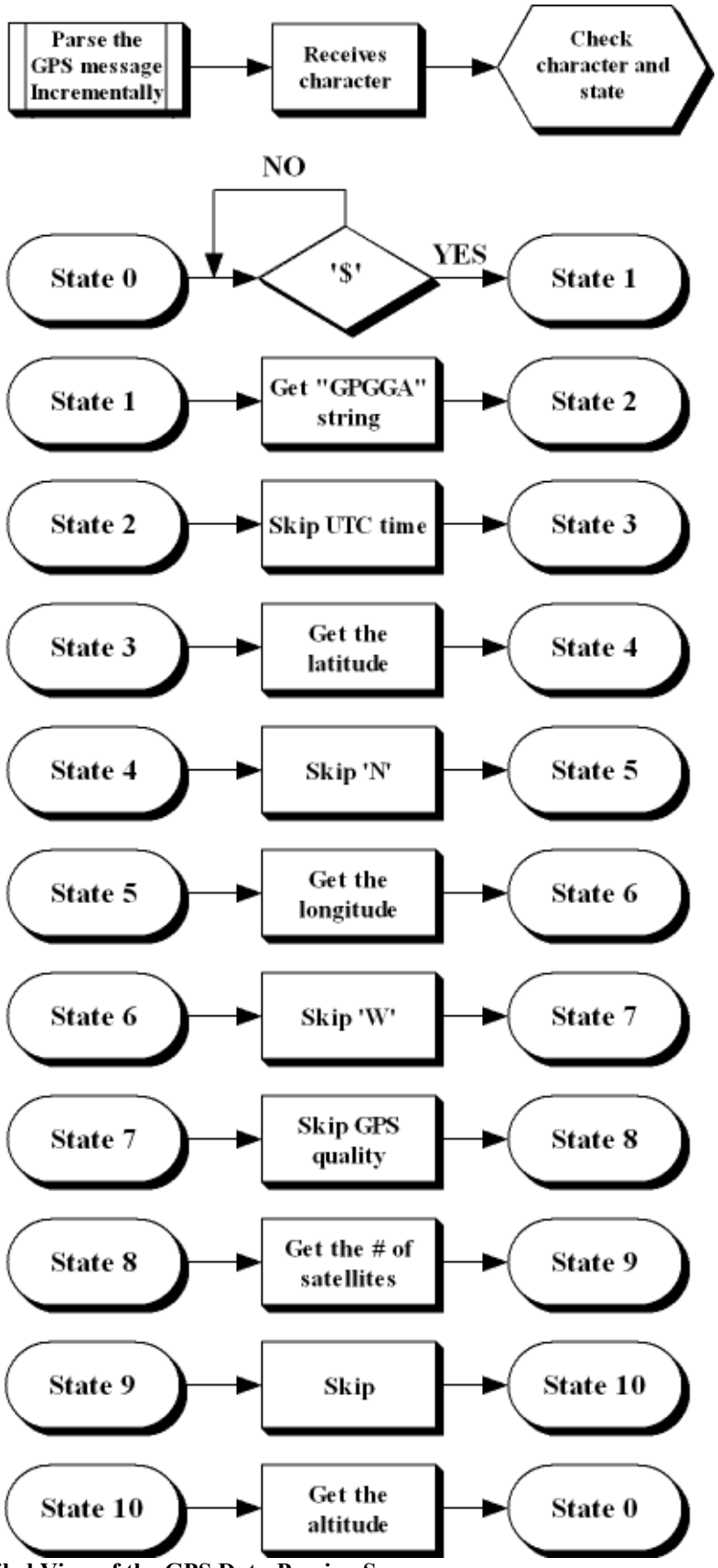


Figure 31: Detailed View of the GPS Data-Parsing Sequence

The flowchart of the procedure for getting the number values follows:

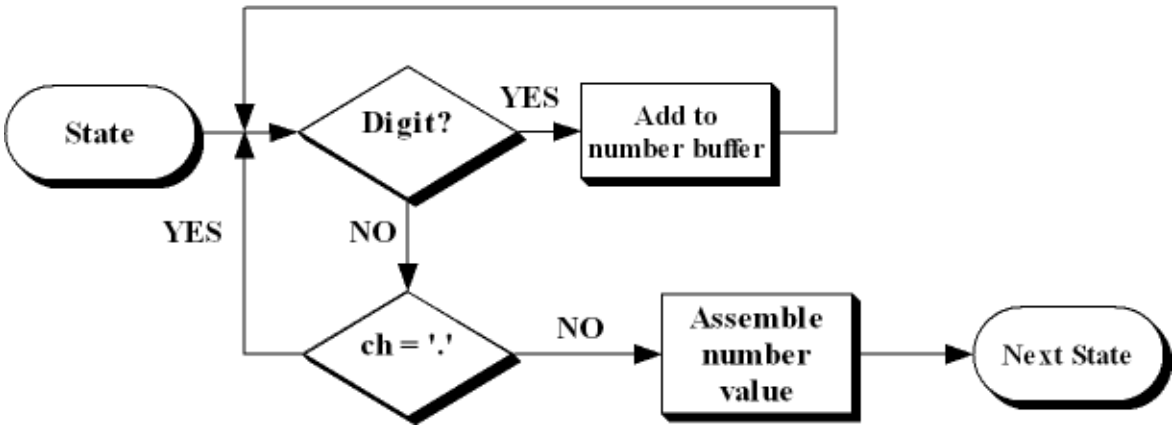


Figure 32: GPS Data-Number Parsing

Finally this is the GPGGA message used for reference:

```
$GPGGA,012041,1800.0000,N,06500.0000,W,2,07,01.2,00101.2,M,039.2,M,04,
 0000*42
```

The parser starts looking for the '$' character. When it is found, the parser knows the GPGGA message follows because it is the only enabled message. The GPGGA string is extracted and checked to see if this is the GPGGA message. The UTC time is then skipped.

Then the latitude is extracted by means of the procedure outlined in the second flow chart for obtaining number values. Note the procedure adds all the characters to the buffer and then changes them to a number.

The north indicator is skipped. Now the latitude must be extracted by using the same method used for the latitude. The West indicator is skipped. Following is the GPS signal indicator. It is skipped because we are using the number of satellites to know if the signal is present or not. The number of satellites is then extracted and converted to a number.

Finally the altitude is obtained and the rest of the message is discarded. Note from the second flow chart, that the decimal point is read but not added to the buffer. This is done to build an integer number, not a floating point number. This means that 18.000000 will be converted to:

18.000000 = 18000000

Using integer math will allow us to obtain better precision in the calculations and the calculations will be done faster.

3.7.4 General Pseudo Code

The device operation is controlled by a driving subroutine. This subroutine will call the component handling subroutines to do its work. The next section describes the code for handling each component; in this section we only explain the general handling subroutines.

```
Program uTracker


   // Loop forever
   While TRUE

      // Check the GPS signal
      isGPSSignalPresent?();


      // State machine
      State Machine

          State Init

             ResetDeviceVariables
             InitIOSubsystem
             InitLCD
             InitGPSChip

             Go to the On State


          State On

              // Reset the time
              if( any button_pressed )
                 ResetStandbyTimer();
              end if


              // Move the menu up
              if( button_up is pressed)
                 curr_coord = wrapValue(curr_coord - 1);
              end if


              // Move the menu down
              if( button_down is pressed)
                 curr_coord = wrapValue(curr_coord + 1);
              end if


              // Set the current coordinates
              if( button_set is pressed )

                 if( GPSSignalPresent )
                    selected_coord = curr_coord;
                 else
```

```
            ShowNonSignalAnimation();
         end if

      end if


      // Track the coordinates
      if( button_track is pressed )

         if( GPSSignalPresent )
            Enable the compass
            Go To the Track State
         else
            ShowNonSignalAnimation();
         end if

      end if


      // If the standby time elapsed
      if( inactive for 30 seconds )
         Go to To-Standby
      end if


   State Track

      // Check if the destination direction should be calculated
      if( new_data )
         calculate_dest_direction();
      end if


      // Identify the direction the user must move to
      find_direction_to_move();


      // Move the menu up
      if( button_up is pressed and the screen != current coords)
         move screen up
      end if


      // Move the menu down
      if( button_down is pressed and screen != dest coords)
         move screen down
      end if


      // Stop tracking
      if( button_track is pressed )
         Disable the Compass
         Go to the On state
      end if


   State To-Standby
```

```
            // Check if a button is pressed
            if( any button is pressed )
               Go to the On state
            end if


            // Transition time elapsed?
            if( transition time is greather or equal to 2 seconds )
               Go to the Standby state
            end if


         State Standby


            SetGPSLowPowerMode();
            DisableSerialCommunications();
            ResetParser();
            DisableLCD();
            StopMicro();

            ... Activated by interrupt

            EnableSerialCommunication();
            SetGPSActiveMove();
            EnableLCD();
            Go to the On state


         State All

            Parse incomming GPS data
            Update the screen

      End State Machine

   End While

End Program



// Maintains the index between (-1) - 3
Subroutine WrapValue(index, numValues)
   return ( (index + 1) + numValues) % numValues - 1;
End Subroutine



// Calcuate the destination direction
Subroutine calculate_dest_direction()

   find_quadrant();
   find_relative_direction();
   calculate_direction(quadrant, direction);
End subroutine
```

```
// Find the quadrant where the destination vector is located
Subroutine find_quadrant()

   if (latitude is positive and longitude is positive)
      return quadrant 0;
   else if(latitude is negative and longitude is positive)
      return quadrant 1;
   else if(latitude is negative and longitude is negative)
      return quadrant 2;
   else if(latitude is positive and longitude is negative)
      return quadrant 3;
   end if

EndSubroutine



// Find the relative direction
Subroutine find_relative_direction()

   if( ratio of latitude to longitude > 2.41 )
      return 0;
   else if( ratio of latitude to longitude < 0.41 )
      return 2;
   else
      return 1;
   end if

End Subroutine



Subroutine calculate_direction(quadrant, direction)

   // Longitude > Latitude
   if( direction = 2 )

      // West or East
      if( quadrant = 0 or quadrant = 1)
         return west;
      else
         return east;
      end if


   // Latitude > Longitude
   else if( direction = 0 )

      // North or South
      if( quadrant = 0 or quadrant = 3)
         return north;
      else
         return south;
      end if
```

```
   // Middle 45 degrees
   else
      return 1 + quadrant*2;
   end if

End Subroutine



// Identify the direction the user must move to
Subroutine find_direction_to_move()

   return ( (dest_direction - compass_direction) + 8 ) mod 8;

end Subroutine
```

3.7.5 Component Interfacing

Switch Button

The purpose of the switch button is to toggle the equipment between on and off. To realize its function, it only has to disconnect the circuit from the power supply. No code is necessary to handle this component.

Push Buttons

The device has 4 push buttons among its components: SET, TRACK, UP and DOWN. The UP and DOWN buttons are used to move through the stored coordinates. The SET button is used to add new locations and the TRACK button is used to find a stored location.

Each push button is connected to a given pin in the microcontroller. When one of this buttons is pressed, the microcontroller will raise a keyboard interrupt. The interrupt based approach provides many advantages. The first advantage is that all the button presses will be reported; polling the inputs does not guarantee that. The second is that our code will have better structure; we will handle the state of the device using an event driven architecture. Below is the pseudo code for the management of the push buttons.

```
Subroutine handle_keyboard_interrupt ()


   Read SET button pin and store it in internal memory
   Read TRACK button pin and store it in internal memory
   Read UP button pin and store it in internal memory
   Read DOWN button pin and store it in internal memory

End Subroutine
```

Compass

To calculate the direction the user must follow to find its target location, we need two types of information: the direction of the target and the direction the user is facing. To obtain the direction of the target location, the GSP chip is used. To obtain the direction the user is facing a digital compass is used.

The compass used in this device is Dinsmore 1490 Digital Compass. This compass does not provide us with measurements at the single degree level. Instead it indicate us the direction we are facing: north, south, west, east, north-east, north-west, south-east and south-west. This is enough for our purposes, because we do not give directions at degree-level either.

The compass provides us the information in digital format, using 4 output pins. We connect it directly into the microcontroller and poll the pins for input each time we need them. As stated in the description section, the compass will be on only when the device is in tracking mode. To control the compass we will use a transistor, which will connect the power supply to the compass and will be controlled by an output pin of the microprocessor. Below is the pseudo code to handle the compass.

```
Subroutine obtain_compass_direction ()

   Read least significant compass pin
   Read second compass pin
   Read third compass pin
   Read most significant compass pin

   Use look table to identify the direction
   Store it in internal memory

End Subroutine

Subroutine turn_compass_on ()

   Write zero to compass control pin

End Subroutine


Subroutine turn_compass_off ()

   Write one to compass control pin

End Subroutine
```

LCD Display

To interact with the user the device will use the push buttons (for input) and the LCD display (for output). The device contains a 4 x 20 LCD display which is appropriate for the amount of information we are going to display.

The given LCD is based the HD44780U standard, which specifies a set of rules and commands that most of the character based LCD support. We are only going to use some of the capabilities of the LCD component, the necessary to display the information to the user.

To display the information to the user we need the following capabilities:

- We must be able to initialize the LCD.
- Clear the screen
- Set the cursor position at a specific location
- Write a character to the screen

Below is the code necessary to provide the previous functionality. Explanations of each code snippet are provided as well.

```
Subroutine init_LCD()

   send_reset_command(0x30);
   send_reset_command(0x30);
   send_reset_command(0x30);

   send_init_command(8 bit bus)
   send_init_command(4 character display)
   send_init_command(turn LCD on)
   send_init_command(turn cursor on)
   send_init_command(automatic cursor positioning)

End Subroutine


Subroutine send_init_command(command code)

   Prepare to send command
   Set command code
   Send the command
   wait_until_command_finishes()

End Subroutine


Subroutine wait_until_command_finishes()

   while(not maximum number of iterations and not ready)
      poll ready flag

End Subroutine
```

```
Subroutine clear_screen()

   Prepare to send command
   Set clear command code
   Send the command
   wait_until_command_finishes()

End Subroutine


Subroutine write_character(character)

   Prepare to send command
   Enable character writing
   Set the character data
   Send the command
   wait_until_command_finishes()

End Subroutine


Subroutine set_cursor_position(row, col)

   Prepare to send command
   Set cursor position code
   Prepare option flag based on row and column
   Send the command
   wait_until_command_finishes()

End Subroutine
```

The previous code snippets refer many times to prepare send command, set command code and send command. An example assembly program is shown below.

```
CLR  EN
CLR  RS
MOV  DATA, #38h
SETB EN
CLR  EN
LCALL WAIT_LCD
```

The above code clears the EN input, which is necessary to set the command code (clear, set cursor position, etc). Then the RS input is clear to indicate this is a command, if it were one, then the command code will be a character to print. The command code is moved into the input ports of the LCD display, which in this case is an initialization command. Then the EN bit is set to load the command into the LCD and finally it is again cleared to execute the command. The wait subroutine waits until the command has executed.

GPS Chip

The most complex of the device components is the GPS chip. The chip selected for this project is the Sony GXB5210 single GPS chip. This chip is a complete implementation of a GPS receiver; no external components are needed to make it perform its intended function.

One of the main aspects of the GXB5210 that influenced in this decision was the simplicity of the system, having everything, including antenna, integrated into one single low power unit. Our device is going to be hand-held, portable and battery powered. These characteristics made the GXB5210 the ideal choice, since it saved space in our system, while also having low power consumption.

The Sony GXB5210 is constructed upon the Sony CXD2951 chip which works using a simple ASCII protocol that is ideal for simple systems. This data is transmitted using serial communication which works well with the 8052 microprocessor UART serial data communication support.

This device communicates using the serial port. It is based on an ASCII command interface. The device receives ASCII messages using the serial port and then it responds with the requested information. To interface with this equipment we must be able to send characters by the serial port. If we can send individual characters, then the remaining functionality can be built based on the basic character sending functionality.

To send the data we must first configure the serial port. We'll be configuring the serial port to be used in Mode 1, working at a baud rate of 4800 and with 8 data bits. The serial mode 1 uses the Timer 1 to set the baud rate, so it must also be configured to produce the necessary baud rate. The baud rate of the GPS Module is also configured using the second and third bit of Port 0, which must be set to 10b to set the 4800 baud rate on the GPS Module. Below is the pseudo code to provide this configuration.

```
Subroutine configure_serial()

   Set SCON to work in Mode 1 with input and output enable.
   Set TH1 to generate 4800 baud rate.
   Set gps_baud to 10.
   Request port 0 update.

End Subroutine
```

Commands sent to the GPS module through the serial port are ASCII based with parameters. All the commands require the system to wait for an echo of the command as a confirmation that the command has indeed been received before allowing another command to be received. Main command input will be as follows:

```
Subroutine serial_command(com, param)

   Send command through serial port, one character at a time.
```

```
   Wait for command echo before proceeding.
   If echo is not received in some time, resend command.

End Subroutine
```

After configuring the serial port, the next step is to configure the GPS Module itself to begin the message output procedure that occurs every second. The commands used for the module configuration will be: '@NC' which is going to be used to configure the GGA sentence from the National Marine Electronics Association (NMEA) standard as the single message outputted from the 8 available, '@WLK' to set the walk mode as on and '@CD' which will be used to perform a cold start of the chip allowing the beginning of the message outputs.

```
Subroutine configure_GPS_module()

   Send command @NC with parameters to set GGA output at 1Hz.
   Send command @WLK to turn walking mode on.
   Send command @CD to perform a cold start of the system.

End Subroutine
```

Other commands that will be used during the system operation will be '@PLM', to put the GPS module in a low power mode in which it will generate less output messages, and thus making it less power consuming; '@SR' to perform a hot start or a warm start after returning from the low power mode and resuming normal tracking, and '@TM' to get the current time.

3.8 Software Termination Level

The device software includes all the main system functionalities. They provide all the necessary functions for data acquisition from the GPS receiver chip and the analysis algorithms to provide the system with the distance between the two points. Also the system provides 3 memory slots to save the selected coordinates and a stand-by mode to help lower power consumption.

A part of the software that is still being implemented is to create a better algorithm to calculate the distance and direction between the two points. Because of the low accuracy of they GPS receiver chip, software have to be done to generate a better accuracy using average calculations and other methods.

Other software functions may be added to change the user interface and provide different functions depending on the application intended for the device.

3.9 Efficiency and Trustworthiness

In recent years, the aspects of efficiency and reliability in software have declined in importance. Many mayor software companies are accustomed to releasing bug fixes for their programs. The efficiency factor has declined in importance by the advent of very fast micro processors (3GHz >). However, for the embedded market, these aspects are as or more important than they were decades ago.

Embedded applications are used in hospitals, planes and military, for which reliability is not an option is a requirement. The failure of any of these systems can cost millions of dollars, and more important, they can cost people's life.

For efficiency, we have to take into account, that these systems contain slow microcontrollers. This is done to decrease the cost of the units and to reduce power consumption. The more efficient the software is, the slower the microcontroller that can be used. In embedded applications, faster software means less cost.

Our project, not being the exception to the rule, was designed with reliability and efficiency in mind. Many factors in our application environment can cause the failure of the system, which range from simple graphical interface problems to serial communication failures. Incorrect handling of errors can cause the system to malfunction and incorrect data will be displayed to the user.

As our application has to perform complex calculations (find the direction, calculate the distance), it has to be implemented efficiently. The system has to cope with many processes at the same time. The GPS data has to be received and parsed, the directions and distances have to be calculated, the GUI has to be updated; all of this at the same time.

A list is presented below with all the optimizations and measurements taken to develop a reliable and efficient system. Each of these measurements is better explained in the following paragraphs. Note they are grouped for better comprehension of their usability and contribution to the system reliability and efficiency.

Measurements

- **Interrupt Handling**
  - Give more priority to the serial interrupt
  - Implement serial data buffering
  - Limit quantity of operations to execute inside interrupts
  - Use only one timer for all the timed events
  - Use the keyboard interrupt instead of the external interrupt
- **GPS Data Manipulation**
  - Incremental parsing
  - Only process one of the 8 GPS messages

    - Set initial coordinates
    - Don't turn off the GPS chip (put it into low power mode instead)
- **Direction & Distance Calculation**
    - Calculate the direction only when new data arrives
    - Calculate the direction efficiently
    - Linear approximation of the distance
- **Data Manipulation**
    - Use internal memory whenever possible
    - Use the smallest data types for a given situation
    - Use integers whenever possible
    - Limit the quantity of floating point operations
- **General Optimizations**
    - Use macros instead of subroutines
    - Enable compiler optimization
    - Update screen only when necessary
    - Update only the parts of the screen that changed

The microcontroller communicates with the GPS chip by means of a serial communication channel. Each time a new character is sent or received an interrupt is generated. Is of great importance that all this characters be received. The loss of one of these characters will cause the parser to stop processing the current GPS message to prevent the presentation of incorrect data to the user.

The first step to prevent this problem is to give the serial interrupt the highest priority of all the interrupts. In case a serial interrupt is generated at the same time that a keyboard or timer interrupt is generated, the serial interrupt will be handled first. This will allow us to receive all the characters of the GPS message.

Additionally, limitation of the amount of operations inside of the interrupt was necessary. The problem was the serial interrupt had precedence before any other interrupt type, but that does not include itself. In case a serial interrupt was generated before the previous was handled, the received character was lost.

The solution was to include only the necessary code to add the character to the receive buffer. The processing of the character was delegated to the main process. At the beginning of the GPS message reception, the characters were added faster that they could be handled, but after the complete message was received, the main process was able to catch up with processing.

For all the timed animations, a single hardware timer was used. We could have used another timer because the microcontroller had 3 timers. One of them is used for the serial communications and one is used for the timed events. Having only one timer improved the efficiency of the system, because the number of interrupts is kept to a minimum, giving more processor time to handling the general procedures.

The keyboard interrupt extension of the microcontroller was used to handle the push buttons. This interrupt is not present in the basic 8051 microcontroller; it is an

extension of the Atmel at89c51. This interrupt is generated when one of the push buttons is pressed, which allows us to handle all the button presses. Previously, the external interrupt was going to be used for this purpose, but that approach had the problem of not reporting a button press when one of the other buttons was already pressed.

When handling the buffered data, we have many options to make the process simpler and faster. The first was the GPS chip provided us with very detailed information, much for which we does not need to concern about. The approach was to only interpret what we needed and discard all the unnecessary information.

The GPS have the capability to output 8 messages, each of which gave the information in different formats. Our implementation only uses the GPGGA message, which provides the latitude, longitude, altitude and number of satellites. At the beginning of the program, the chip is told to output only that message, to simplify the parsing process.

When new data arrived, it was put in the buffer, to be later processed. We have two approaches here: one was to receive the complete message and then parse it; the second was to be parsing the message as it was received. We chose the second; we parse the data as it is coming. This allows us to use the time between characters receptions to process the data.

To obtain the GPS signal faster, approximate coordinates are given to the chip. This allows the GPS system to obtain the signal faster. Additionally, the GPS is never turned off when the device is on to prevent the long wait time needed for the first location output. The chip is put into standby state, in which it computes the location slowly, but does not loose the necessary information for performing fast reception later.

When new data arrives, the destination direction is calculated. This calculation is not performed every time because it is a time consuming process (floating point operations, etc). The issue of calculating the direction efficiently is better explained in the theory section, here we only give a brief summary.

Using traditional methods, to find the angle between the destination direction and the user orientation, a considerably amount of floating operations would be needed. The microcontroller we are using is based on an 8-bit architecture. The handling of data types with greater sizes is emulated by software.

The emulation of a floating point number required first the emulation of a 4 byte integer data type and above that, the emulation of a floating point data type. Using floating point math incurs a performance penalty, but is necessary for our application. The method that we used reduced the number of these operations.

The previous discussion only talks about the direction calculation, but the distance also has to be calculated. Again, the traditional methods are too time consuming. The distance calculation between a pair of latitude and longitude required the use of the

spherical coordinate system. The distance is calculated by using the cosine and sine functions which are too expensive to make in the microcontroller.

A solution could be to use a lookup table that contains the values of these functions, but that would need 360 floating point entries for each of the tables, giving a total of 2880 bytes, too much for a microcontroller. What we did was to perform a linear approximation of the distance. When using this linear approximation for distances of around 10,000(the maximum for our system), the error ranged between 0.1% - 5%.

As the microcontroller is based on an 8-bit architecture, we tried to maintain the size of the device variables to the minimum possible. Most variables are in the 1 byte - 2 byte range, which are efficiently handled. 4 bytes data types were only necessary for storing coordinates.

With the exception of the buffers, all the variables are in the 128 bytes of internal memory, which the microcontroller handles faster than the additional 1024 of external memory. The last data type optimization was to change most of the floating point operations to integer operations, which gave us performance gains and better precision.

Finally a number of general optimizations were applied to the code. We used macros for single statements functions. The macros provided us with a way to better organize the code without incurring in a performance penalty. Compiler general optimizations were turned, so the overhead of expression calculation was keep to a minimum.

The screen was only updated when necessary. This prevented the flashing that occurs when the screen is changing to fast and provided a performance gain. Additionally, only parts of the screen that changed were updated. This is done to provide a smoother display and again to obtain better performance.

4. Part List

Part List	Quantity
AT89C51RC2-3CSIM Intel 8052 Microprocessor	1
GPS Single Chip Module With Integrated Antenna (Sony GXB5210)	1
Push Button	5
4 x 20 LCD Display DMC20434	1
Dinsmore 1490 Digital Compass	1
3.3 V Voltage Regulator	1
5.0 V Voltage Regulator	1
Quad Inverting Three State Gate	1
74LS14 Inverting Schmitt Trigger	1
82 kΩ Resistor	4
4.7 kΩ Resistor	2
1 kΩ Resistor	4
8.7 kΩ Resistor	1
1.5 kΩ Resistor	2
82 kΩ Resistor	4
10k Ω Resistor	4
10 µF Capacitor	3
0.1 µF Capacitor	4
33 pF Capacitor	2
11.05982 Mhz Clock	1
LED	7
Connector FFC/FPC 1MM 10POS VERT ZIF	1
Flat Flex Cable 4”	4”
Battery Holder	1
AA Duracell MN1500 (2850 mAh)	5

5. Cost Analysis

Part	Unit price	Bulk price	Minimum quantity
AT89C51RC2-3CSIM 8052 microprocessor	$8.46	$3.67	1000
Dinsmore 1490 Digital Compass	$15.95	$9.85	1000
Sony GXB5210 Single Chip GPS	$49.72	$47.60	100
DMC20434 LCD Display	$35.52	$20.21	500
Push Buttons (100g Operating Force)	$0.21	$0.19	100
Resistors	$1.00	$0.25	1000
Capacitors	$0.15	$0.05	100
Battery Holder	$0.93	$0.42	100
LM317T Adjustable Voltage Regulator (x2)	$0.83	$0.52	500
LM7805 5.0V Voltage Regulator	$0.25	$0.17	500
CD74HC125E Three State Gate	$0.72	$0.2394	1000
74LS14DR Inverting Schmitt Trigger	$0.55	$0.154	1000
Connector FFC/FPC 1MM 10POS VERT ZIF	$1.84	$0.64421	5000
Flat Flex Cable 4”	$3.35	$3.35	
Totals:	**$119.48**	**$87.32**	

Note: Bulk prices are expected to be lower at final version production because of custom developed ICs and industry special prices.

The analysis shows that the product has a marketable production price. The most expensive parts of the system are the GPS single chip module and the LCD display. We believe that these parts can get lower prices when bought for mass production and that they can be easy acquired from distributors, maybe with the possibility of supply contracts.

6. Conclusion

After all the analysis was done and the system requirements were all considered, the final result is a design that has succeeded in satisfying all the expected needs that the device was required to reach our proposed goals. The system uses the Global Positioning System effectively to analyze current position, and the software programming stores these values in memory which are later compared and present valuable information to the user, using simple but direct algorithms that manage to provide the constant feedback that the user will expect in using a system like this, while using a low power microcontroller, like the AT89C51RC2.

Specific hardware design choices were done to allow the system have a configurable power management options that are switched through the working cycle of the device. Also, because of the GPS technology properties, the system is free of interference while working outdoors, and since the position information is calculated using the GPS satellites, the system can be considered to be independent of location. Finally, the user interface of the system was done to be as simple as possible, and thus, making the device easy to understand and use, which allows it to have diverse market options for a wide range of different customers.

The design issues that still need some consideration are only a few. The most important issue is the accuracy of the GPS chip while getting the current location. The chip does a good job getting the current location with a margin of 20 meters. However, for some applications this is not enough, but can be corrected using mean value algorithms and approximations. Also a more precise calibration of the digital compass can be done to allow the system to have a higher and more exact direction calculation.

We understand that this device has a wide range of uses in different areas of work and that it also can be integrated to be part of a larger system to perform a more specific task. This may be only the first prototype of a device that will ultimately have a wide range of applications in society.

7. Future Work

The GPS system allows us to implement a great variety of applications. For this project, the original view was a little constrained. The idea was to use the device to help people find their car in the parking lot. After some thought, we realized that our system had many more applications. These applications ranged from helping people lost in the forest, to helping you to find your friends location.

At the end of the semester we finished the development of a device that performs its original functionality; it will help you to find your car, not at the precision we wanted, but enough to be usable. In the process we have many complications that prevented us to add features to the system that were not in the original specification. But the ideas are exposed here, so later groups can implement them for their projects.

Ideas

- **Hardware**
  - Use an analog compass.
  - Use an inclination sensor.
  - Provide the microcontroller programming circuit in the device.
- **Power Management**
  - Control the speed of the microcontroller
  - Control the contrast of the LCD
  - Provide a backup battery for the GPS chip
- **Additional Functionality**
  - Provide a better user interface
  - Provide more detail about the GPS signal
  - Provide an atomic clock
  - Use the counter extension for timed events management
- **Flash Memory**
  - Store more coordinates
  - Store the name of the coordinate's location
  - Caching the last coordinate to sent to the chip at startup
- **Bluetooth & USB**
  - Interchange locations with other devices
  - Upload new locations taken from other devices and lookup them in the map
  - Download new locations

When we found the digital compass, the idea of having the direction given in digital format was simple. When we were making the device we found this was not completely true. The problem was the compass is not very precise and assigns more range to some cardinal directions (North occupies 65% of the top two quadrants).

A better approach will be to use an analog compass, which will give use better precision and will allow us to select the direction appropriately. Additionally, as the compass becomes imprecise when the inclination is greater than 25 degrees, we could use an inclination sensor to detect when we need to modify the input data to decrease the error in the direction information.

We also wanted to include the programming circuit inside the device, so the programming and testing of the system became easier. What is currently done is to program the chip then take it out of the programming circuit and put it in the device's circuit, which delays the testing procedure. Later generations could make the circuit in place.

Another area that could be improved is power management. The system is portable, which means it should run on batteries and has to consume the minimum amount of power. We have many ideas in this respect, of which the first one is to make a backup circuit for the GPS chip.

The chip has the ability of preserving memory when a little amount of voltage is applied to one of its pins. If it preserves the memory when power is not applied to it, we could turn it off in the standby mode. Remember, we don't do this right now because the chip is slow acquiring the signal the first time. But if it has the data already in memory, it will be faster acquiring the signal after coming out of standby.

The second idea is to provide the user a mechanism that she could use to control the contrast of the LCD, which is one of the devices that consume more power. The third would be to control the speed of the microprocessor. The Atmel at89c51 has the capability of reducing its frequency of operation without interfering with the serial communication speed. We could use the 11.059MHZ crystal to provide the necessary baud rate and internally put the micro to run at a slower frequency.

For production of the device we think a better user interface is needed. Not something at the level of cell phones but something better than we have now. We see a user interface with more animations, a better menu and more options. We also want to display more information about the GPS signal. This information could include the quality of the signal, the speed at which the user is moving and the current time.

The topic of time is very important, because with GPS we have an atomic clock at our disposition. The idea is to update the clock based on the GPS data. When the signal is not present we could use the internal timer to update the clock. When the GPs signal is acquired again, the system will then update the clock to conform to the new GPS data.

The microcontroller we use, have for extension an array of counters that can be used to count the number of overflows of the timer 0. This allows us to use these counters instead of the software timers that are currently used to control timed events. With this measure we can provide the system with a greater degree of precision than it has right now.

The microcontroller also allows us to write to flash memory. This as being a persistent are of storage provides us with a way to store coordinates for later use. The idea is to store hundreds of coordinates and then select the ones we want and put them in internal memory. In addition the names of the locations could be stored with the coordinates. For example, we could give the UPRM name to the UPRM coordinates.

The uTracker does not provide a mechanism to add a keyboard to it. So a PC will be used to download the information to the device by means of a USB port. The user could go to a database of coordinates in the internet and download the ones she wants, and then use the device to track them. The process can also be reversed. If the system becomes Bluetooth enabled, coordinates can be shared among the devices. Later, the user could upload the new coordinates and use the PC to look them in a map.

Finally the chip will regularly write the current coordinates to flash memory. When the device is turned on it could check the stored coordinates and use them for the first approximation instead of using the predefined ones.

8. References

1490 Digital Compass. Images SI, Inc. 2005. <http://www.imagesco.com/articles/1490/01.html>.

Atmel Corporation. "AT89C51RC2 Datasheet." Atmel Corporation AT89C51RC2 Product Card. Atmel Corporation. 2005. <http://www.atmel.com/dyn/products/product_card.asp?family_id=604&family_name=8051+Architecture&part_id=2854>.

Atmel Corporation. "C51 API Program Examples." Atmel Corporation AT89C51RC2 Product Card. Atmel Corporation. 2005. <http://www.atmel.com/dyn/products/product_card.asp?family_id=604&family_name=8051+Architecture&part_id=2854>.

Atmel Corporation. **"**FLIP 2.4.4 for Windows**.**" Flip. 2005. <http://www.atmel.com/dyn/products/product_card.asp?family_id=604&family_name=8051+Architecture&part_id=2854>.

Atmel Corporation. "Hardware Interface Connection Examples for C51 MCU." Atmel Corporation AT89C51RC2 Product Card. Atmel Corporation. 2005. <http://www.atmel.com/dyn/products/product_card.asp?family_id=604&family_name=8051+Architecture&part_id=2854>.

EdSim51, The 8051 Simulator for Lecturers and Students. NyCelt LLC. 2005. <http://www.edsim51.com/>.

How to control a HD44780-based Character-LCD. Peter Ouwenhand. 22 Jan. 2005. < http://home.iae.nl/users/pouweha/lcd/lcd.shtml>.

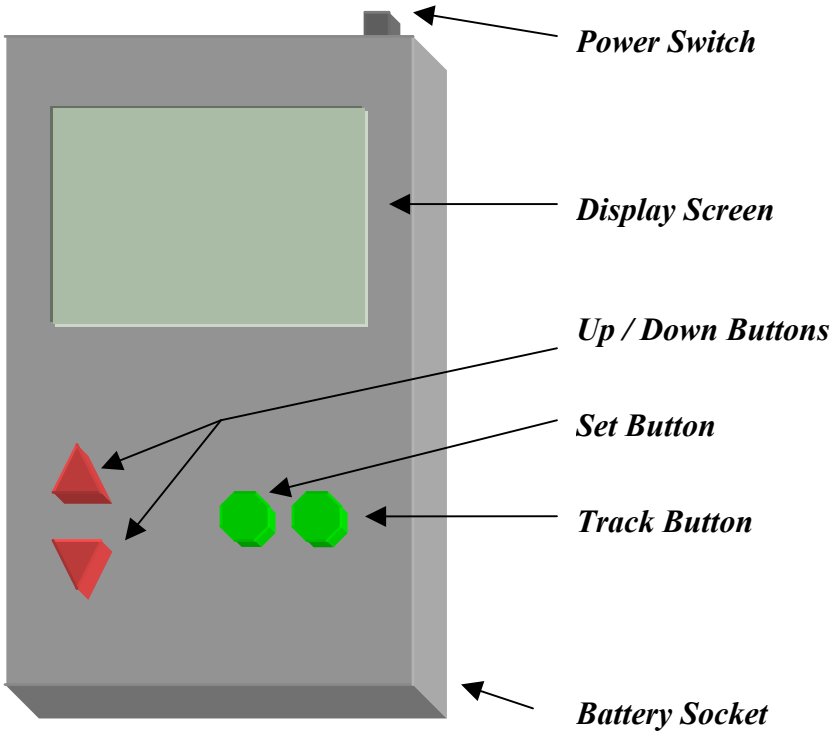
Synergy Systems LLC. "SONY GXB5210 GPS RECEIVER DATA." Board Level GPS Products. 2005. < http://www.synergy-ps.com/SONY%20GXB5210%20GPS%20Receiver%20Data.pdf>.

The 8052 Online Reference. Vault Information Services LLC. 14 Sept. 2005. <http://www.8052.com/>.

Vahid, Givargis. Embedded Systems Design: A Unified Hardware/Software Introduction. John Wiley & Sons, 2002.

9. User Manual

A. Device Components:



These are the main components of the µTracker with their functions:

1. coordinate screens. The device supports a total of 3 different coordinates that **Display screen:** Is the only output display of the system. Presents the On State and Track State screens as well as other information.

2. **Push Buttons:**
   a) **Up / Down Buttons**: Scroll through the different screens in the On State and Track State.
   b) **Set Button**: Used only in On State. Stores current position coordinates in selected memory location.
   c) **Track Button**: Toggles between On State and Track State. Selects stored position coordinates to track.

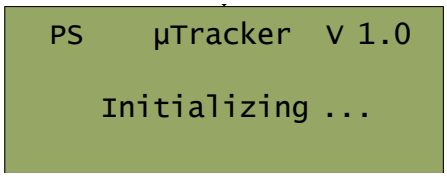
3. **Power Switch**: Turns device on or off.

4. **Battery Socket**: Stores batteries. The device requires 5 AA batteries to work. Batteries not included.

B. Getting Started:

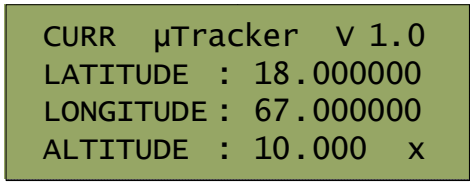
To turn on the GPS µTracker, place 5 AA batteries in the battery socket and use the power switch. After the device is turned on, the start up screen will be shown on the display. Please allow at least 5 seconds while the device initializes.

INIT STATE



After the initializing is done, the device will enter to the On State and will display the initial approximation coordinates. These coordinates are latitude 18ºN, longitude 67ºW and altitude 10 meters. These values are an initial approximation of the current coordinates and are set according to the coordinates of Puerto Rico and its surrounding areas.

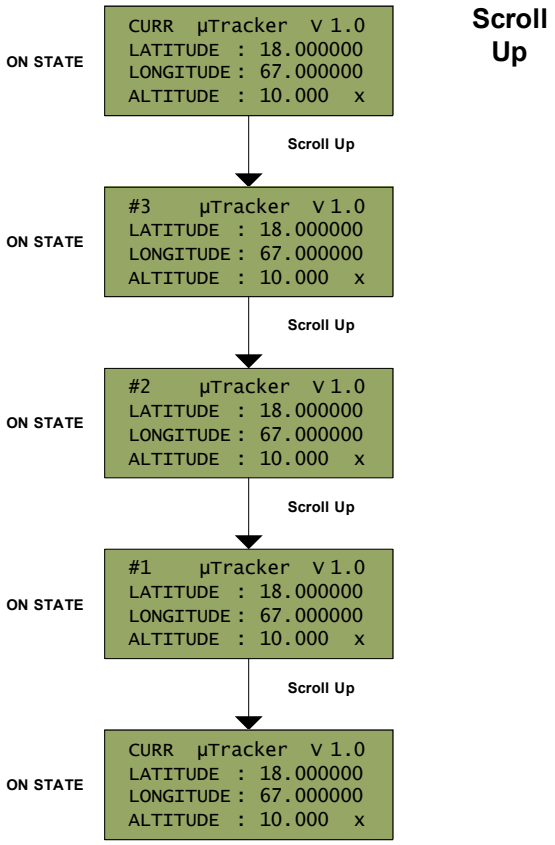
ON STATE



The device will now automatically start to connect with the Global Positioning System satellites and try to obtain its current location coordinates. We suggest that the device is used outside to allow a better communication with the GPS satellites. While the device is still looking for connection an ‘X’ character will appear at the bottom right location of the screen. After the device has acquired connection, the character will change from an ‘X’ into a number and the current coordinates will be displayed. This number informs the number of satellites to which the system is currently connected. The device takes an average time of one to three minutes to connect.

C. On State:

After the device is connected, it’s ready to save and track coordinates. To save coordinates, the device must be in On State. From the current coordinates screen, the Up and Down buttons are used to scroll through memory can be stored into different locations to later be tracked.



To save the current coordinates into a memory location, press the Up or Down buttons to scroll screens to the screen of the desired memory location. The number of the memory location is shown in the upper left corner of the screen. The number ranges from 1 to 3, and a special case where the letters “CURR” are displayed, showing that it’s the current coordinates screen.

After the desired location is selected, press the Set button to save the current coordinates into that location. The screen will be automatically refresh and show the coordinates stored. The Up and Down buttons can now be used to select another screen and store the coordinates into another location or to return to the current coordinates screen.

To track a stored coordinate, scroll through the memory locations screen until the display screen shows the location and values of the desired coordinates to be tracked. Press the Track button to select the stored coordinate and set the system to Track State.

D. Track State:

When the device enters into Track State, the track screen is shown in the display screen. The track screen displays the information obtained from the analysis and comparison of the current coordinate of the device, obtained from the GPS satellites, and the coordinate stored in memory. In the bottom right part of the screen, below the letters “ALT”, the difference in altitude is shown in meters. In the center of the screen, below the letters “DIST”, the linear distance between the two coordinates is displayed in meters. In the bottom left part of the screen, below letters “DIR”, one of eight arrows is displayed. This arrow points to the direction where the tracking coordinate is located with

respect of top of the device. The arrow displayed will change as the device is rotated horizontally.

From the tracking screen, the Up and Down button can be used to scroll between screens accessible in the Track State. These screens are the track screen, the current coordinate screen and the tracked coordinate screen. The tracked coordinate screen corresponds to the screen of the selected memory location to be tracked in On State.

If the Up button is pressed while in the track screen, the display screen will change to show the current coordinate screen. To return to the track screen while in the current coordinate screen, the Down button must be pressed.

If the Down button is pressed while in the track screen, the display screen will change to show the tracked coordinate screen. To return to the track screen while in the tracked coordinate screen, the Up button must be pressed.

If the Track button is pressed, the system will toggle back to On State, where it will be possible to store new coordinates in memory locations or track another coordinate from a different memory location.

While in Track State, the Set button is disabled.

E. Stand-by Mode:

The µTracker features a power saving stand-by mode to prevent unnecessary battery power consume while the system is left with the power on and unattended. This mode is only accessible when the system is in On State and it is toggled automatically by the system.

If the device is left in the On State for more than 30 seconds without receiving any input from the user, the system will display the stand-by screen to announce that it is about to enter to the stand-by mode. This screen is displayed for 2 seconds and then the display screen is turned off, and internal system activity is reduced. The device returns to the On State when it detects that a button has been pressed.

Appendix A: AT89C51RC2 Datasheet

Note: Files found inside project CD.

Appendix B: Sony GXB5210 Datasheet

Note: Files found inside project CD.

Appendix C: DMC20434 LCD Display Datasheet

Note: Files found inside project CD.

Appendix D: Dinsmore 1490 Digital Compass Datasheet

Note: Files found inside project CD.

Appendix E: Program Listing

Note: Files found inside project CD.