WiSHABI

Wireless, Single Handed, Accelerometer-Based, USB-HID Compliant, PC Interface

Jadon Clews

B. Eng. Mechatronic Engineering Project Report

Department of Mechanical Engineering

Curtin University of Technology

2008

Jadon Clews
13 Bangalay Court
Halls Head WA 6210

October 31st, 2008

A/Prof Tilak T. Chandratilleke
Department of Mechanical Engineering
Curtin University of Technology
Kent Street
Bentley WA 6210

Dear Sir

I submit this report entitled "Wireless, Single-Handed, Accelerometer-Based, USB-HID Compliant, PC Interface", based on Mechatronic Project 491/493, undertaken by me as part-requirement for the degree of B.Eng. in Mechatronic Engineering.

Yours faithfully

Jadon Clews

Acknowledgements

Dr Muhammad Ilyas Mazhar – For his supervisory style which allowed me the flexibility to complete this project in a manner which best suited me whilst still providing me with an easily approachable source of advice.

Dr Euan Lindsay – For his enthusiasm and teaching style throughout my university education, which I found inspiring and which also made attending classes a much more enjoyable experience.

Marcus Holliday – For proof-reading and providing advice in regards to this report and for being genuinely interested in this project.

My Family – For their unwavering support and for feigning interest whenever I wanted to discuss this project.

Abstract

The recent decline in manufacturing costs of acceleration sensing devices known as accelerometers, has lead to a proliferation of said devices within consumer electronics (such as cameras, notebook computers and mobile phones). This project was undertaken to demonstrate a practical application of such a device by utilising its ability to sense static acceleration attributed to gravity. The following report describes the design and construction of a system which enables comprehensive control of a PC, via a wireless, hand-held device without the need for additional driver software. The system consists of a receiver-unit which connects directly to a PC (utilising the USB protocol) and a transmitter-unit which fits comfortably in the user's hand. Two modes of operation exist; mouse-mode and keyboard-mode. When operating in mouse-mode, the device acts as a wireless mouse and the motion of the on-screen cursor corresponds to the tilt-angles of the transmitter-unit. Keyboard-mode allows text-entry of the alphanumerical character set by means of combined transmitter tilt-orientation and button presses. The tilt-orientation of the transmitter-unit is detected by means of a tri-axis accelerometer. The complete system is a working example of another use for this versatile and increasingly affordable technology.

Nomenclature

- **0x** – Prefix used when representing hexadecimal (base sixteen) numerical values (e.g. 0xFF is equal to decimal 255).
- **AA** – A standardised size use for describing battery cells.
- **ADC** – Acronym for Analogue to Digital Converter. Used to convert a voltage level within a specific range to a digital value which can then be processed by a digital system such as a microcontroller.
- **ATmega** – A subset of Atmel's range of AVR microcontrollers.
- **Atmel** – A manufacturer which specialises in the production of microcontrollers.
- **AVR** – A range of microcontrollers manufactured by Atmel.
- **Byte** – An abbreviation for Binary Term. A standard storage measurement of computer data consisting of 8 bits.
- **Bit** – An abbreviation of Binary Digit. A basic unit of information storage typically represented by a value of either 1 or 0.
- **g** – Unit of measurement for acceleration where one 'g' is equal to the static acceleration attributed to gravity (approximately $9.81 m/s^2$).
- **GUI** – Acronym for Graphical User Interface. A computer operating system that is based upon icons and visual relationships rather than text (e.g. Windows or Linux).
- **GUI key** – A key included on a typical keyboard which performs functionality specific to the utilised operating system.
- **HID** – Acronym for Human Interface Device. A class of USB device (with specific protocols) which includes peripherals such as mice and keyboards.
- **I/O** – Abbreviation for Input/Output. On a microcontroller, an I/O port is a set of pins which can function as either digital inputs or outputs.
- **IC** – Acronym for Integrated Circuit. A tiny complex of electronic components and their connections that is produced in or on a small slice of material, usually silicon.
- **IDC** – Acronym for Insulation-Displacement Connector. A connector that pierces the insulation on a wire to make the connection, removing the need to strip the wire before connecting. Also used to refer to the sockets which are compatible with these connectors.

- **ISP** – Acronym for In-System Programming. The ability for some microcontrollers, and other programmable electronic ICs to be programmed whilst still installed within a complete system.
- **LED** – Acronym for Light Emitting Diode. A semiconductor diode that emits light when conducting current.
- **MAX232** – A specialised IC that converts signals from an RS-232 serial port to signals suitable for use in TTL compatible digital logic circuits.
- **NiCd** – Abbreviation for Nickel-Cadmium. A type of rechargeable battery.
- **NiMH** – Abbreviation for Nickel Metal-Hydride. A type of rechargeable battery.
- **PC** – Acronym for Personal Computer.
- **PCB** – Acronym for Printed Circuit Board. A flat board whose front contains slots for ICs and other electronic components, and whose back is printed with electrically conductive pathways to provide connections between the components.
- **RF** – Acronym for Radio Frequency. The area (or band) of the electromagnetic spectrum where most radio communication takes place.
- **RGB** – Acronym for Red, Green, Blue. Refers to a type of LED which is capable of emitting visible light in three distinct colours, or combinations thereof.
- **RS-232** – A Widely recognized protocol standard for serial data interchange. Once commonly utilised by PCs it is currently being superseded by alternative serial communication protocols (such as USB).
- **RX** – Abbreviation for Receive or Receiver.
- **TTL** – Acronym for Transistor-Transistor Logic. Commonly used to describe a class of integrated circuits which operate at discrete levels of 0 and +5 volts.
- **TX** – Abbreviation for Transmit or Transmitter.
- **UART/USART** – Acronyms for Universal Asynchronous Receiver/Transmitter and Universal Synchronous/Asynchronous Receiver/Transmitter. Serial data processing protocols commonly utilised by microcontrollers.
- **WiSHABI** – A partial acronym for the title of this project. Derived from Wireless, Single-Handed, Accelerometer-Based Interface.

Table of Contents

List of Figures

List of Tables

1 Introduction

The project detailed by this report involved the design and development of a prototype PC interface device. The device is intended to provide the user a means of both mouse and keyboard-like control of a PC from a single, wireless, hand-held unit.

This report will provide information pertaining to the design and construction of the device as well as how it actually operates. Additionally, the report will explain how the device is intended to be used, including its capabilities and limitations.

The report is broken down into eight main sections, each of which is described in the following list:

1. Introduction – An overview of the content of this report and the project it describes including objectives and the progression of the project development.
2. Background Information – Underlying information required to develop an understanding of how the device operates. Also includes a description of the major sources of data utilised for project development.
3. System Overview – A description of how the device is intended to be used as well as some key concepts involving how it operates.
4. Transmitter-Unit – Details pertaining specifically to the design and construction of the transmitter-unit's hardware and firmware.
5. Receiver-Unit – Details pertaining specifically to the design and construction of the receiver-unit's hardware and firmware.
6. Conclusions – An overview of the results of the project including its limitations and some proposed improvements for the system.
7. References – A list of all the sources used in development of both the project and this report.
8. Appendices – Supplementary data and information.

1.1 Project Objectives

The objectives initially desired for the completed system can be broken into two parts. The primary objectives are those which had to be met in order for the project to be considered a success. Secondary objectives are those that were desired, but not required.

1.1.1 Primary Objectives

It was deemed necessary that the completed, functional system should be:

P1 Wireless – The controller should not be tethered by any data or power cables. This means a wireless transmitter/receiver combination had to be implemented and the transmitter-unit had to be capable of operating from batteries.

P2 Operable with a single hand – The transmitter-unit had to be small enough to fit in a person's hand and any buttons mounted on the transmitter-unit had to be easily accessible by the user's fingers.

P3 Simple to install – The receiver should be connected to the PC via a commonly used interface.

P4 Simple to use – The system should require minimal instructions for setup and operation.

P5 Capable of providing an indication (visual or audible) of the current operating mode (mouse or keyboard).

P6 Compatible with all major PC operating systems.

Note: The primary objectives are numbered for reference purposes only. They all share equal, mandatory priority.

1.1.2 Secondary Objectives

It was deemed ideal that the completed, functional *transmitter*-unit should be:

S1 Capable of transmitting large distances to maximise operational range.

S2 Rechargeable (without replacing batteries) and remain operational whilst being recharged.

S3 Operate for a long time on a single charge to minimise required recharging frequency.

S4 Comfortable to use (no sharp edges).

S5 Easily firmware upgradable.

S6 Aesthetically pleasing.

It was also deemed ideal that the completed, functional *receiver*-unit should be:

S7 Usable without requiring any additional software to be installed on the connected PC.

S8 Capable of receiving data from large distances to maximise operational range.

S9 Capable of drawing the required power from the connected PC.

S10 Capable of providing a visual indication of the tilt-orientation of the transmitter-unit.

S11 Easily firmware upgradable.

S12 Aesthetically pleasing.

Note: Like the primary objectives, numbering of the secondary objectives is for reference purposes only.

1.2 Project Progression

Once the objectives had been defined, design and construction of the system was undertaken. The following sections describe the steps taken to achieve the desired objectives. Although denoted in a linear format, some overlap existed from step to step.

1.2.1 Detailed Functional Design

The first step was to determine the specific details pertaining to how the completed system should operate. This included the following details:

- Wireless data transmission and reception method – 433.92Mz Frequency.
- PC interface method – USB.
- Accelerometer interface method – Serial.
- Number of buttons required on the transmitter-unit – 5.
- Number of outputs required on the receiver for LED indicators – 12.

The selection of specific hardware components (in particular the microcontrollers) depended predominantly on the design decisions defined here.

1.2.2 Hardware Selection and Initial Circuit Designs

The next step was the selection of the primary electronic devices with which to build the system. The selected components follow:

- Microcontroller (x2) – Atmel ATmega8 AVR.
- Wireless TX/RX Modules – 433.92MHz, 4800baud, Transmitter and Receiver-Modules.
- Accelerometer – Analog Devices ADXL330 3-Axis ±3g Accelerometer.
- Power-Module – 3-9Volt DC-DC Converter.

With the key components selected, both the transmitter and receiver circuits were designed.

1.2.3 Initial Circuit Construction

Once the required components were obtained, the circuits were built. Initially the components were assembled on two separate breadboards (one each for the transmitter and receiver circuits) as this would allow for any refinements or modifications to be undertaken easily.

1.2.4 Basic Firmware Coding and Circuit Testing

The next step involved writing the basic code functions to be used by the final system. These functions also provided a means of testing certain components within the circuits. For example the functions involving basic USB communications were implemented and tested to prove that the USB related components of the receiver system were connected appropriately. The key underlying functions written at this stage included those required for:

- Serial communications.
- Data packet encoding/decoding.
- USB communications.
- Analogue to Digital (ADC) conversion (for Accelerometer)
- I/O port manipulation for LED outputs
- I/O port manipulation for button inputs

1.2.5 Transmitter-Unit Final Construction

With the ability to decode readings from the accelerometer, the next step required the selection of a suitable enclosure for the transmitter and installation of the fully constructed transmitter circuit. Selecting and testing the suitability of a transmitter enclosure was necessary at this stage so that the transmitter-unit's tilt-orientation detection subroutine (see section 5.3.4 Determining Tilt-Sector) could be implemented and tested. This in turn would allow mouse and keyboard-modes to be properly tested with calibrated settings.

1.2.6 Advanced Firmware Coding

Once the transmitter-unit was fully constructed, calibration of the accelerometer was confirmed and the main functionality of the system (mouse and keyboard-modes) was implemented. These subroutines formed part of the receiver-unit's firmware.

1.2.7 Receiver-Unit Final Construction

At this stage, the system was operating in such a way that it met all of the primary objectives for the project. Next, the receiver-unit's circuit was transferred from the breadboard to a more permanent circuit board which was then installed into a suitable enclosure. This point saw the completion of all the hardware required for the project.

1.2.8 Firmware Refinement

The final stage involved refinement of the existing firmware. This included testing, bug removal and the implementation of additional functionality to meet the remaining secondary objectives. After this stage the project was considered fully operational and focus was directed to completing the accompanying documentation.

1.3 Project Applications

The prototype developed as a result of this project provides a means of interfacing with a PC wirelessly using only one hand and without requiring a flat surface such as a desk or podium. Such functionality could be useful in the following example scenarios:

- When giving a presentation such as a lecture.
- When PC control is necessary in the field with no immediately accessible flat surfaces for a regular mouse or keyboard.
- When operating a home entertainment PC from an armchair or a bed.

Additionally, the receiver-unit built as a part of this project provides a sound hardware platform for developing USB devices with wireless data reception capability and easily accessible I/O ports.

The completed project is also useful for demonstrating a less intuitive application for an accelerometer.

2 Background Information

To better understand how the complete system works, a few key points must be explained:

1. How an accelerometer can be used to determine tilt-orientation.
2. The formatting necessary to wirelessly transmit a data packet.
3. Basics details of the USB Human Interface Device (HID) protocol

The following sections will provide an understanding of these concepts as well as information regarding how the majority of the data utilised for this project was obtained.

2.1 Measuring Tilt with an Accelerometer

Most modern accelerometers are simple "micro electro-mechanical systems" (MEMS) consisting of cantilever beam and proof mass structures for each measured axis. Forces acting on the structure cause a displacement along the axis which leads to variations in a measured capacitance. There is also a newer technology that is being increasingly implemented. In this case, accelerometers use a heated gas bubble combined with thermal sensors. When the accelerometer is tilted or accelerated, the sensors pick up the location of the gas bubble (similar to the air bubble in a spirit level often seen on construction sites). In either case, the accelerometer outputs a signal which represents a value proportional to the acceleration experienced along the relative axis.

If an accelerometer is stationary and has an axis parallel to the direction of gravity, the output from the device for that axis would represent acceleration with a magnitude of 1g. If the same axis was aligned perpendicular to the direction of gravity, the output would be indicative of acceleration with a magnitude of 0.

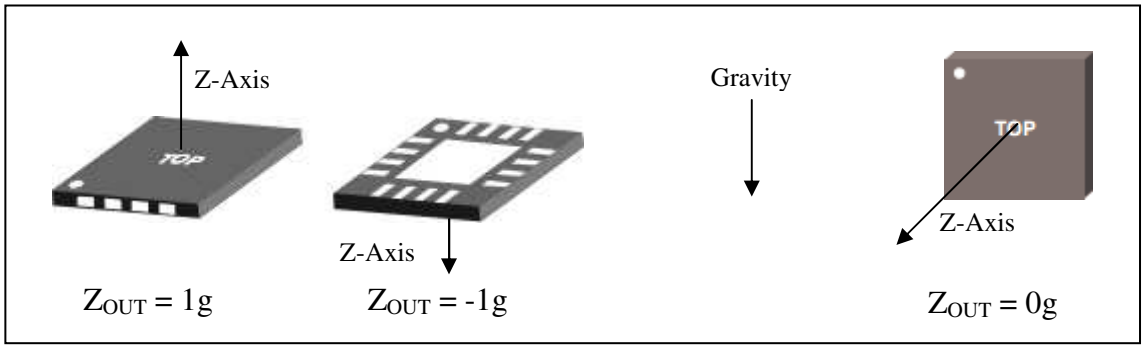


Figure 2-1: Accelerometer Output with respect to Gravitational Orientation

If the accelerometer begins in a resting position oriented such that its Z-axis output equates to 1g, and it is then rotated on an axis perpendicular to Z, the new Z-axis output can be calculated as a function of the angle by which the device is rotated.

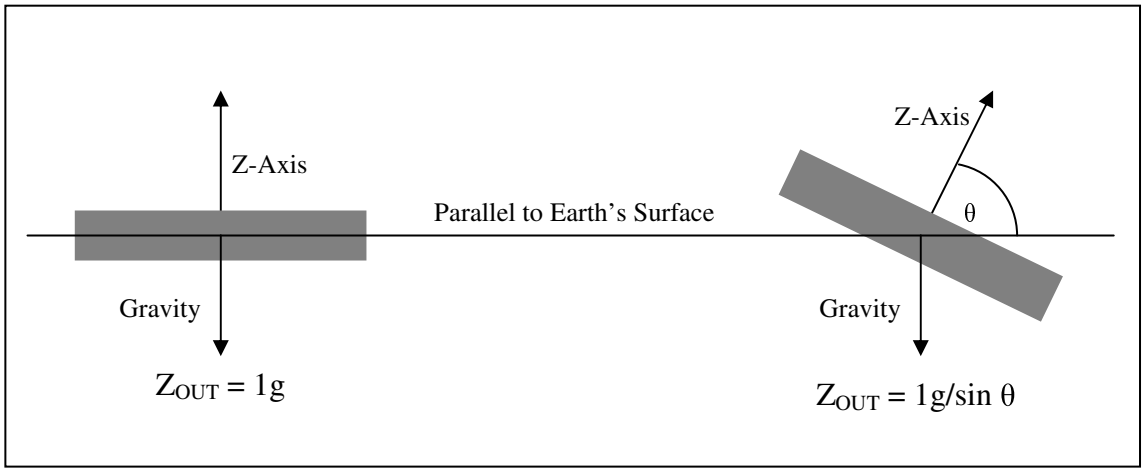


Figure 2-2: Derivation of Tilt Angle from Accelerometer Output

Thus the tilt angle θ can be calculated from the formula:

$$\theta = \sin^{-1}\left(\frac{1g}{Z_{OUT}}\right) \qquad \text{Equation 2-1}$$

An accelerometer will provide the output signal in one of two forms (some devices provide the option of both):

1. Digital – The signal is a stream of digital bits intended to be read and decoded by means of a serial receiver (a common microcontroller peripheral). The decoded data will consist of a data byte representing the acceleration vector.
2. Analogue – The signal is a voltage level that will vary between a minimum (Usually 0 volts) and a maximum (usually the device source voltage – 3.3 volts). To be useful, this analogue signal must be converted to a digital signal via an "Analogue to Digital Converter" (ADC). Many microcontrollers have ADC capabilities which are used to convert the signal to a useable data byte representing the acceleration vector.

In both cases, the acceleration vector is eventually represented in the form of a data byte or bytes. Typically this is an 8-bit value but other resolutions are possible. In the case of an 8-bit representation of an acceleration vector, a byte value of 0 equates to the minimum measurable acceleration value, while a byte value of 255 equates to the maximum measurable acceleration. Thus for a ±3g accelerometer, the 8-bit representation of the axis outputs equates to an acceleration value as defined in the following table:

Byte Value (Decimal)	Byte Value (Hexadecimal)	Acceleration
0	0x00	-3g
42	0x2A	-2g
85	0x55	-1g
128	0x80	0g
169	0xA9	1g
212	0xD4	2g
255	0xFF	3g

Table 2-1: Byte Representation of Acceleration Vectors (8-bit resolution, ±3g)

In this example, an increment in the byte value (x) equates to an approximate increase of 23.4×10^{-3}g. As such, the acceleration value (A) can be calculated with the following formula:

$$A = 0.023x - 3 \qquad \text{Equation 2-2}$$

This equation would vary depending on the resolution of the measured signal and the maximum magnitude of acceleration that the accelerometer is capable of sensing.

When using an accelerometer to determine tilt angle, only the values representing acceleration vectors of ±1g are useful because a magnitude of 1 is the maximum acceleration possible under the sole influence of gravity. So for a ±3g accelerometer, a data range of about 0x55 to 0xA9 is used to derive the angle of tilt for the corresponding axis. The value measured here combined with equations 2-1 and 2-2 provide the means by which a microcontroller can determine the tilt-orientation of an accelerometer and whatever it is attached to.

2.2 Serial Communications

Data transfer between two directly connected electronic devices is often facilitated by means of some variety of serial communications. A few commonly used protocols exist for serial communications implemented by microcontrollers. Some examples include:

- UART/USART – Universal Asynchronous Receiver/Transmitter or Universal Asynchronous/Synchronous Receiver/Transmitter.
- SPI – Serial Peripheral Interface.
- I^2C – Inter-Integrated Circuit (TWI – Two Wire Interface is a variation of I^2C).
- CAN-bus – Controller-area Network bus.

The different protocols have various advantages and disadvantages, but they all operate by means of transmitting and receiving a series of bits which are received and arranged into data bytes.

2.2.1 Serial Data Level Conversion

Sometimes it is required that devices which employ different serial communication protocols be capable of communicating with each other. For example an application might require a microcontroller's USART communicating with the serial port of a PC. In this case the two protocols have the same data structure but the USART employs TTL-compatible logic levels while the PC's serial port utilises logic levels as specified by the RS-232 standard. The two differing logic levels are incompatible, so for the devices to successfully communicate, a third device must be employed to act as a bridge and convert the levels in both directions.

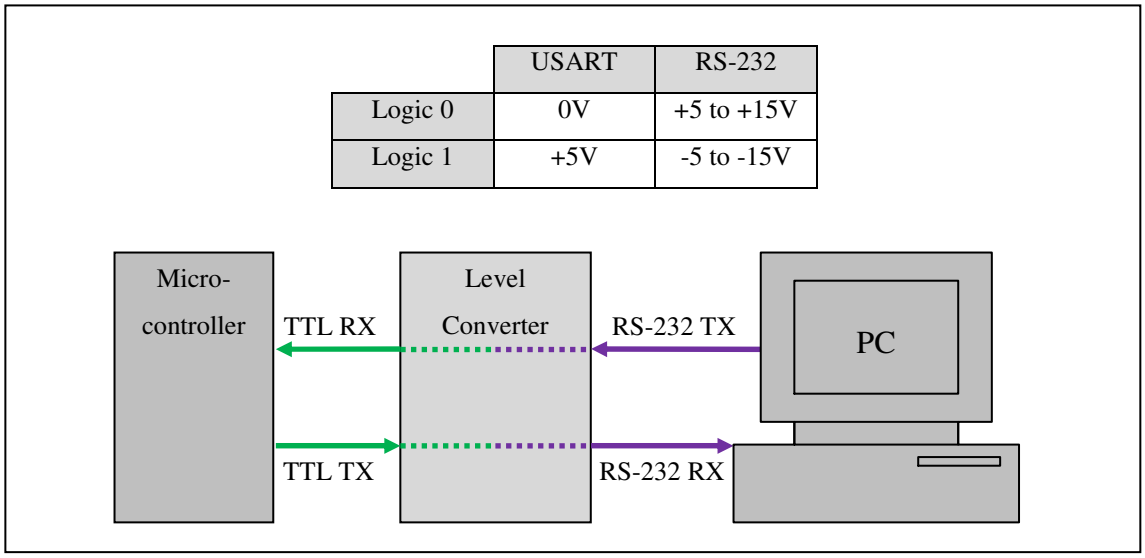


Figure 2-3: Converting Serial Line Levels

The level converter often takes the form of a specifically designed IC. A suitable device for this example would be the MAX232 IC which, when combined with a few additional components, converts TTL-compatible levels to RS-232 and vice-versa.

2.2.2 Wireless Serial Communications

Serial communication between two directly connected devices is relatively straightforward. The devices can send and receive the bare minimum data bytes with little or no chance of said data being corrupted or irrelevant data being unintentionally received.

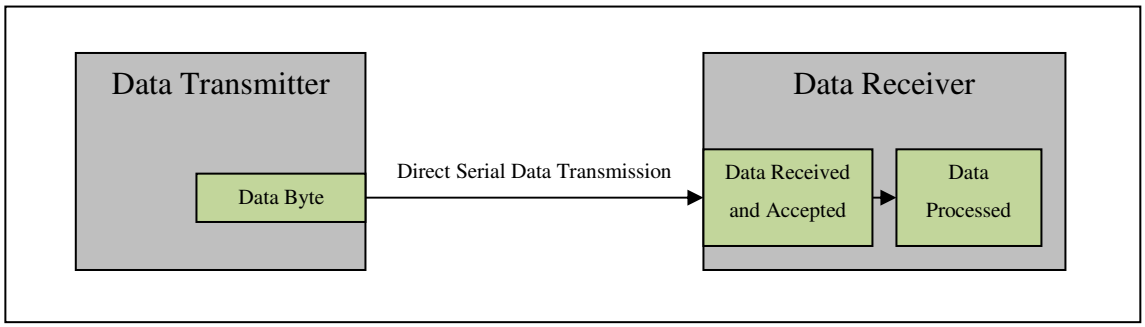


Figure 2-4: Directly Connected Serial Communication

If however, a device was set up to receive serial data "over-the-air", there becomes a dramatic increase in the likelihood of data unintentionally being received from an unrelated, third-party device with wireless transmission capabilities.

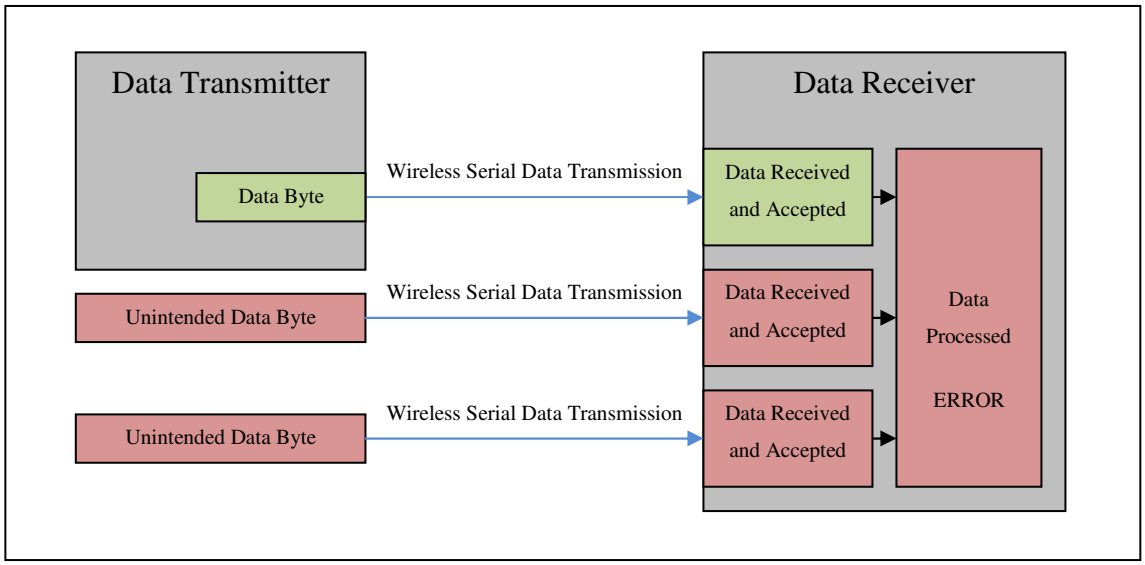


Figure 2-5: Wireless, Unauthenticated Serial Communication

Thus some kind of authentication protocol must be incorporated into the transmitted data so that the receiver can reject unauthenticated transmissions from other wireless sources (and also protect against random noise in the relevant frequency range). This is referred to as encapsulating the data within a "data packet". The packet contains the actual data byte/s as well as some additional bytes included for authentication and identification purposes.

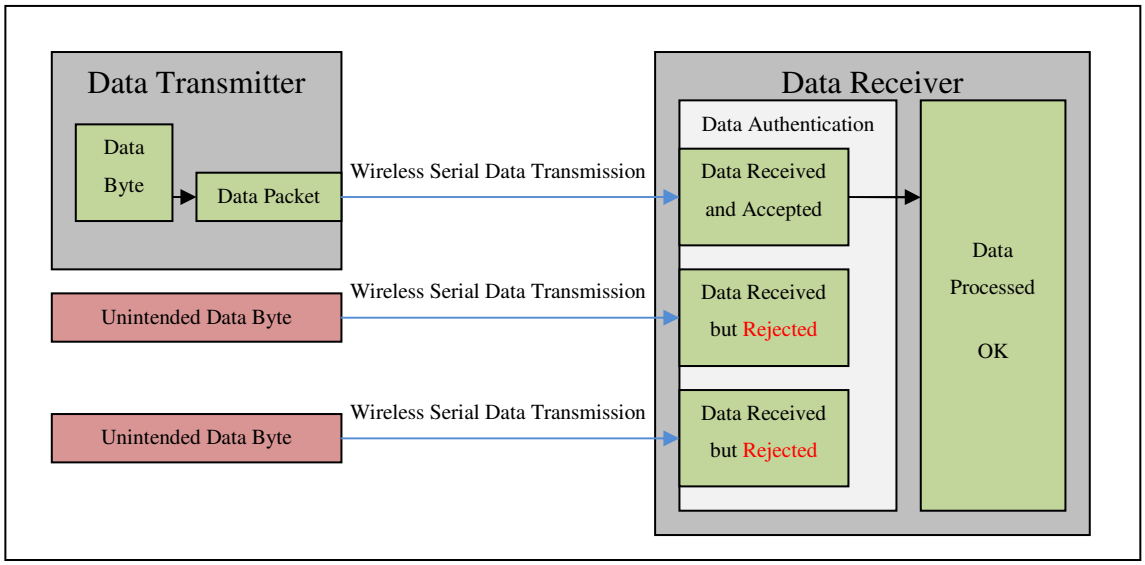


Figure 2-6: Wireless, Authenticated Serial Communication

With a defined packet structure in place, the receiver has a means of authenticating data such that only the bytes actually intended for the receiver will be accepted and processed. The rules applicable to a particular packet structure can vary greatly but this is inconsequential provided both the transmitter and receiver follow the same protocol.

Typical data packets often contain data bytes to indicate a variety of information depending on the application. Such bytes could represent various data. Examples of data contained within a packet include (but are not restricted to):

- Beginning of the transmission.
- Address of sender (if multiple transmitters exist).
- Address of intended recipient (if multiple receivers exist).
- Number of data bytes within the packet.
- Beginning of actual data bytes.
- End of actual data bytes.
- Checksum (calculated value that is used to check for errors in the received data).
- End of transmission.

An example system may require basic, one-way, wireless communication between two microcontrollers. No addressing is required and no form of error-checking is deemed necessary. The number of data bytes is always the same. The following data packet format could be used:

Byte	Value	Meaning
Byte 1	0x01	SOH (Start of Header)
Byte 2	0x02	STX (Start of Text)
Byte 3	0xXX	1st Data Byte
Byte 4	0xXX	2nd Data Byte
...	...	...
Byte n-3	0xXX	2nd Last Data Byte
Byte n-2	0xXX	Last Data Byte
Byte n-1	0x03	ETX (End of Text)
Byte n	0x04	EOT (End of Transmission)

Table 2-2: Example Data Packet Format

In this case, to receive and authenticate the data, the receiver would look first for the SOH and STX bytes in succession. Until these bytes are received, all other bytes are rejected. In the event that SOH and STX are received, the receiver then expects the actual data. After the final data byte the receiver expects an ETX byte followed by EOT. These final two bytes confirm and authenticate the data transmission before the receiver returns to waiting for an SOH byte.

This is just one of many possible data packet structures and processing procedures. Additional data can be incorporated into the packet to provide stronger authentication requirements, additional information or more efficient error-checking capabilities.

If there is only a small number of actual data bytes within the packet, incorporating additional bytes into the packet structure can have an adverse effect on the system. For example, a packet with a single SOH byte followed by a single data byte has half the size of a packet containing SOH, STX, data and EOT. The smaller packet can be processed in half the time required for the larger packet.

The structure of the data packet depends on the requirements of the application and the environment it is intended to be used in (i.e. an area with a lot of RF noise may require additional authentication rules). The specific details of wireless data transmission including the implemented packet format relevant to this project are detailed later in section 3.4 Wireless Data Packets.

2.3 USB Protocol – HID Compliance

The Universal Serial Bus (USB) is a serial communications protocol commonly used to interface external devices with a host (usually a PC). Human Interface Device (HID) is a class of USB device with its own specific protocols. The HID class exists as a standard for USB keyboards, mice, joysticks or other interface devices that transfer blocks of information to and/or from the host at moderate rates.

USB specifications (including class protocols) are defined by the *USB Implementers Forum* (USB-IF), an industry standards body incorporating a number of companies from the computer and electronics industries (www.usb.org/).

The following sections contain relevant information pertaining to the general implementation of the USB protocol and the more specific HID-class.

2.3.1 USB Protocol

For the user, a USB device is simple to install and set-up. The price of being user-friendly is greater complexity for the developers tasked with designing and programming the USB device.

There are a few types of physical connectors commonly used with USB devices. Typically (and as defined by the USB specification) the "server" device (usually a PC) makes use of the Standard-A type connection while the "client" device utilises one of the other standard connection types.

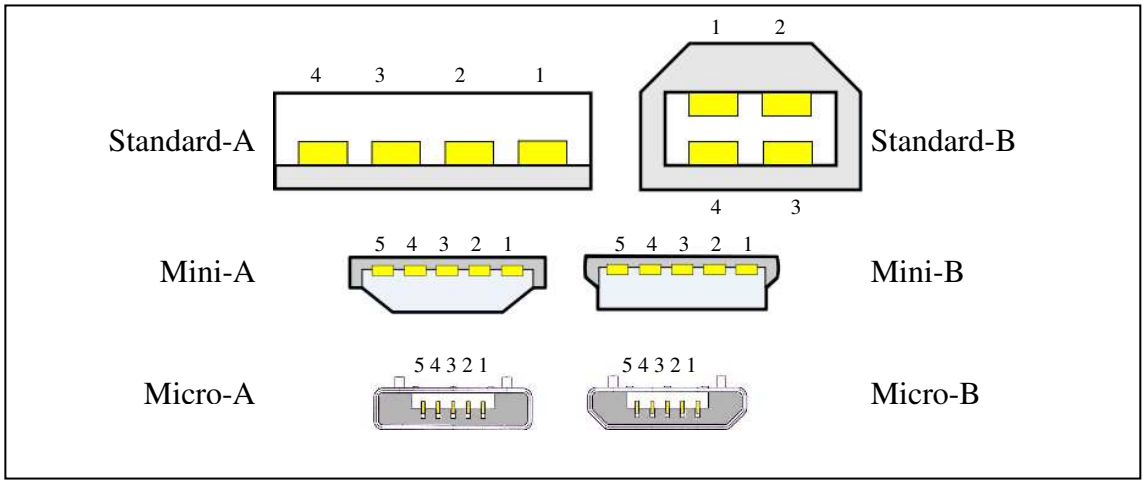


Figure 2-7: Standard USB Connection Receptacles

Important features of the USB protocol include:

- Many peripherals (up to 127) can be connected simultaneously on a single "bus".
- Bi-directional communication is possible (although, in many cases, only uni-directional communication from the device to the PC is required).
- Devices utilising the protocol are "hot-swappable" which means they can be connected and disconnected without the need to restart the PC.
- Power can be provided for low-consumption devices via the USB connection, so no external power supply is required.
- Having a low number of required electrical contacts (4 or 5 depending on the connector) means that the USB cable takes up very little space.
- Standardised physical connectors are keyed which ensures that connections are made easily with no chance of contacts being incorrectly shorted.

Although the USB protocol is considered to be complex when compared with other protocols, it can still be implemented with relative ease into device designs by one of three methods:

1. Serial to USB converter – Standard TTL level serial communications from a microcontroller are often converted to the RS-232 serial standard (implemented on many PCs) by means of a specially designed IC (see section 2.2.1 Serial Data Level Conversion). In the event that a PC does not have an RS-232 serial port, users have the option of purchasing a "Serial to USB converter". This is a device which connects to one of the PCs USB ports and then behaves as a regular RS-232 serial port. This option has the advantage of simplifying the device design as the USB protocols are handled by an external IC. However the device will be restricted in the same way as a regular serial device.
2. Microcontroller with internal USB peripheral – Some of the more recently available microcontrollers have a hardware implementation of the low level USB protocol operations as one of their inherent features. Implementing such a microcontroller in the design of a USB compliant device would minimise the additional hardware and firmware required and still have the full performance capability of USB. Currently

however, such microcontrollers are more expensive and less available than similar devices without the USB feature.

3. Microcontroller with firmware implementation of USB – The USB protocol can be implemented using regular I/O pins of a microcontroller via the use of complex firmware. Using a firmware USB "driver" has the advantage of being applicable to a wider range of available microcontrollers and thus providing more hardware options. The disadvantages include a reduction in available memory space (required by the USB driver), plus creating the driver is likely to be arduous and possibly not worth the benefits. In some cases it may be prudent to utilise a pre-written driver supplied by a third party and modified to suit the requirements of the new design.

Information pertaining to the implementation of USB from hardware and firmware perspectives as applicable to this project can be found in sections 5.1.4 USB Interface Hardware and 5.3.2 USB Interface Firmware Driver respectively.

2.3.2 HID-Class

Specifications exist for a number of types of USB peripherals or "classes". Some of the more common classes include:

- Hub (used to connect multiple devices to the same bus).
- Communications Device (Telephones, modems etc).
- Mass Storage (CD-ROM drives, hard disk drives, USB flash drives etc).
- Power Device (control for power conservation or uninterruptable power supplies).
- Printer.
- HID – Human Interface Device (keyboard, mouse, joystick etc.).

The final class in the list (HID-class) is commonly used for USB keyboards, mice and joysticks (although it can be used for other devices as well). The advantage of having a standardised class is the need for only a single driver in order for any operating system to work with any device compliant with the protocols of the relevant class. In the case of HID devices, all recent versions of the major operating systems are pre-loaded with the required driver. This means the user can simply plug in their HID-compliant device and begin using it with no additional software installation required (as per objective S7).

The data exchanged between the device and PC resides within structures called *reports*. The firmware of the device must be configured to support the HID report format which is flexible and can handle most types of data. A single report can contain up to 255 bytes.

The format of the reports is defined by an array of values known as the *report descriptor*. The report descriptor of a device is sent to the host when the device is first connected so that the host can make sense of the data being received (inputs) and be aware of the format of any data which must be regularly sent to the device (outputs). For example, if the device is a mouse, the *reports* will contain data pertaining to the status of the mouse buttons and any changes in position of the mouse cursor. This means the *report descriptor* will contain values indicating which data bytes in the report represent a left-click, which data represents horizontal cursor motion etc.

The following table is an example of a report format described by a report descriptor which could be used for a USB keyboard.

Byte #	Byte Data	Data Description
Byte 1	Modifier Status	Each bit within this byte represents the status of a keyboard modifier (Ctrl, Alt, Shift etc.)
Byte 2	Keystroke 1	These three bytes represent regular keyboard keystrokes. In this example up to three simultaneous keystrokes (plus modifiers) can be sent to the PC simultaneously.
Byte 3	Keystroke 2	
Byte 4	Keystroke 3	
Byte 5	LED Status	Each bit in this byte represents the status of a keyboard LED (Caps Lock, Num Lock etc). This byte would actually be described in the report descriptor as an output which means it would be sent from host to device, unlike the previous bytes.

Table 2-3: Example HID Report Descriptor (Keyboard)

The flexibility of a HID device lies in the ability to define a report descriptor as one sees fit. This means a device could utilise a report format which indicates any combination of standard keyboard, mouse or joystick controls, or a subset thereof. The following table represents the HID report format defined by the HID report descriptor for a simple keyboard/mouse combination device.

Byte #	Byte Data	Data Description
Byte 1	Modifier Status	Each bit within this byte represents the status of a keyboard modifier (Ctrl, Alt, Shift etc.)
Byte 2	Keystroke	This byte represents a regular keyboard keystroke. In this example only one keyboard keystroke (plus modifiers) can be sent simultaneously.
Byte 3	Mouse Button Status	The three least significant bits of this byte each represent the status of a mouse button (left, right and centre click). The remainder of the byte is padded with zeros.
Byte 4	Cursor Delta X	This byte will be a value (-127 to 127) representing the change in position (in pixels) of the on-screen mouse cursor in the horizontal direction.
Byte 5	Cursor Delta Y	This byte will be a value (-127 to 127) representing the change in position (in pixels) of the on-screen mouse cursor in the vertical direction.

Table 2-4: Example HID Report Descriptor (Keyboard/Mouse Combination)

There are two technical documents essential for working with HID-class devices. They are *Device Class Definition for Human Interface Devices* and *HID Usage Tables*. Both documents were created by members of the *USB Device Working Group* which is affiliated with the previously mentioned USB-IF. The documents are available at http://www.usb.org/developers/hidpage/ and are also included on the CD accompanying this document (see Appendix G).

2.4 Information Sources

Two websites proved to be invaluable sources of data pertaining to this project. Each contained many projects (usually submitted by hobbyists) which revolved around different implementations of AVR microcontrollers. Some of these projects utilised hardware and firmware implementations that were similar to the requirements of this project, thus providing already tested and documented data which could be modified and combined to suit the requirements of this application.

Additionally, the two websites included forums by which hobbyists could communicate to share or request information or help regarding projects. These forums also proved invaluable sources of useful information.

The two websites were:

1. AVR Freaks – Includes a large number of user-submitted projects and a well established forum all which revolve around implementations of AVR microcontrollers. www.avrfreaks.net/
2. Objective Development AVR-USB – Includes projects and a forum which revolve around the implementation of their firmware USB driver in AVR microcontrollers. www.obdev.at/products/avrusb/index.html

Specific projects utilised as data sources are listed in section 7 References, along with the other sources of information utilised for this project. Additionally, the referenced projects are included on the attached CD (see Appendix G).

3 System Overview

The complete interface consists of two primary sub-systems; the transmitter-unit and the receiver-unit. The user manipulates the transmitter by hand and relevant data is sent wirelessly. This data is received by the receiver-unit which is connected to a PC's USB port. Both systems utilise a microcontroller for data input, processing and output. Data is transferred from the transmitter-unit to the receiver-unit in the form of data packets via the use of wireless transmitter and receiver-modules.

3.1 Physical Layouts

The following two sections provide details pertaining to how both the transmitter and receiver-units were physically laid out. Diagrams also indicate the dimensions of the units.

3.1.1 Transmitter-Unit Layout

The transmitter-unit is small enough to be operated within a user's hand. The shape of the unit was intended to be comfortable to hold whilst providing thumb access to three buttons on the top (A, B and C) and index-finger access to another two buttons on the bottom (D and E). At the end of the unit (opposite to the buttons) is a jack for a DC plug-pack which provides power to re-charge the internal batteries.

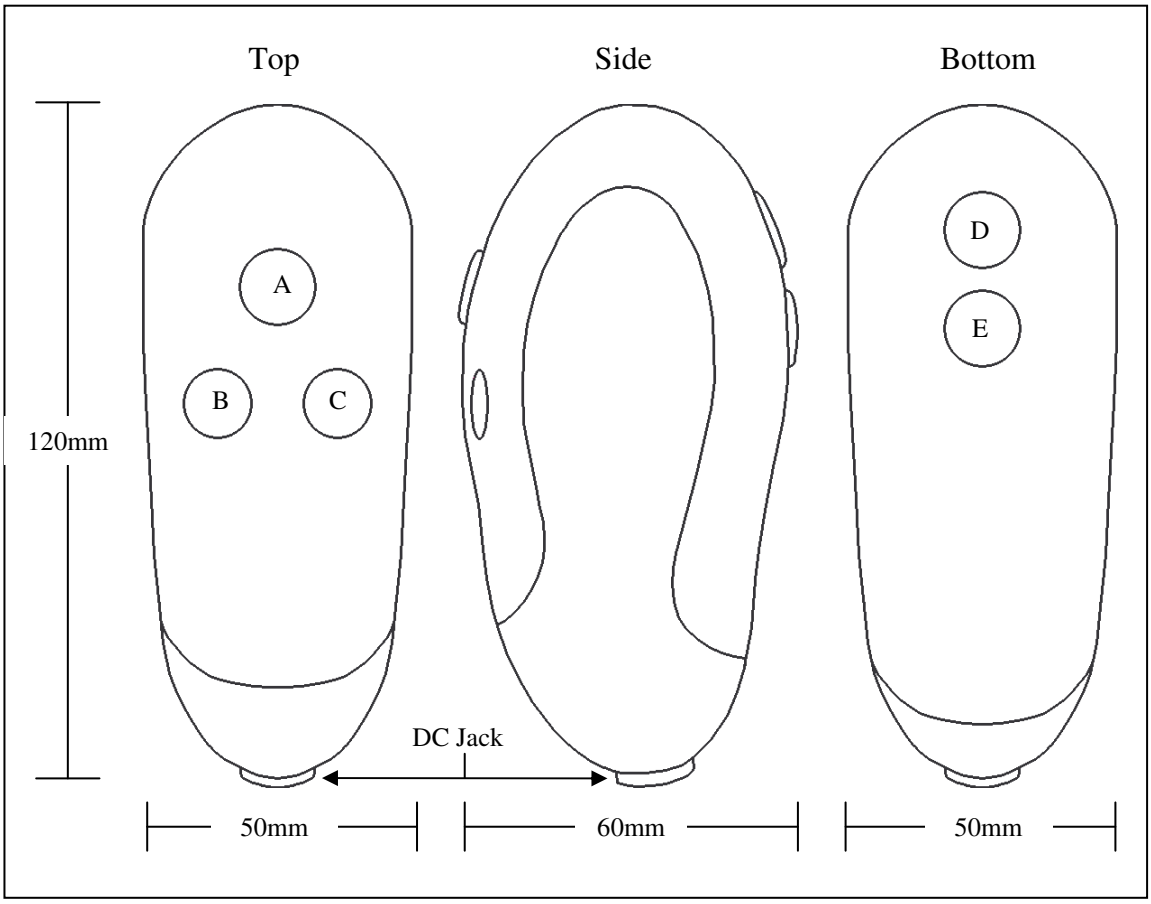


Figure 3-1: Transmitter-Unit Layout and Dimensions

3.1.2 Receiver-Unit Layout

The receiver-unit consists of an aluminium enclosure with a USB standard-A, female receptacle at one end. The other end provides access to three 10-pin, male IDC connectors. Mounted on the top of the unit are a single tri-colour (RGB) LED and a quarter-wave whip antenna.

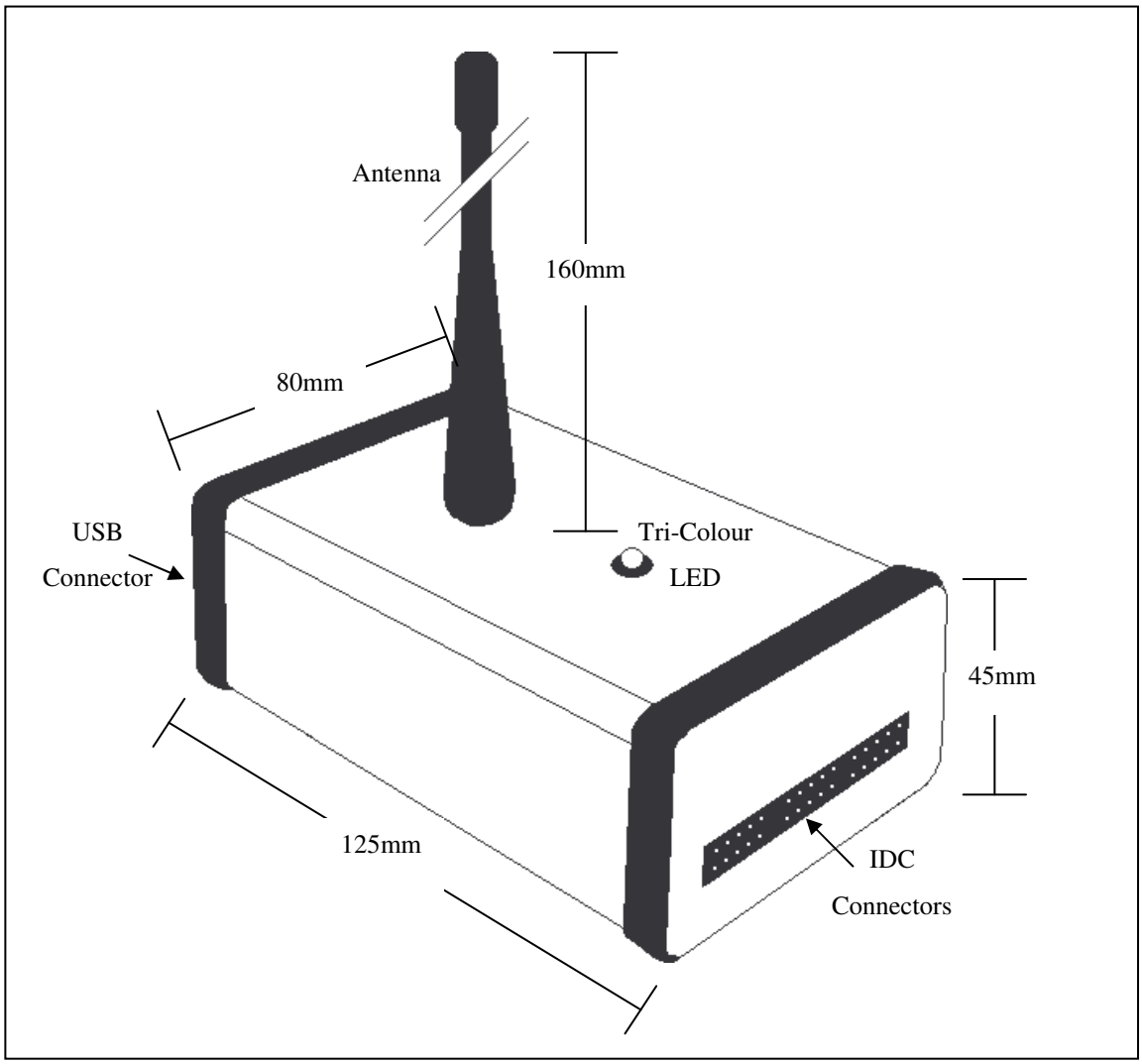


Figure 3-2: Receiver-Unit Layout and Dimensions

In addition to the main receiver-unit is a small external peripheral containing a 3x3 grid of LEDs. These LEDs connect to the microcontroller's data I/O pins via the centre 10-pin IDC connector on the main receiver-unit. The purpose of this 3x3 LED-grid module is explained in the section 3.2.2 Tilt-Orientation.

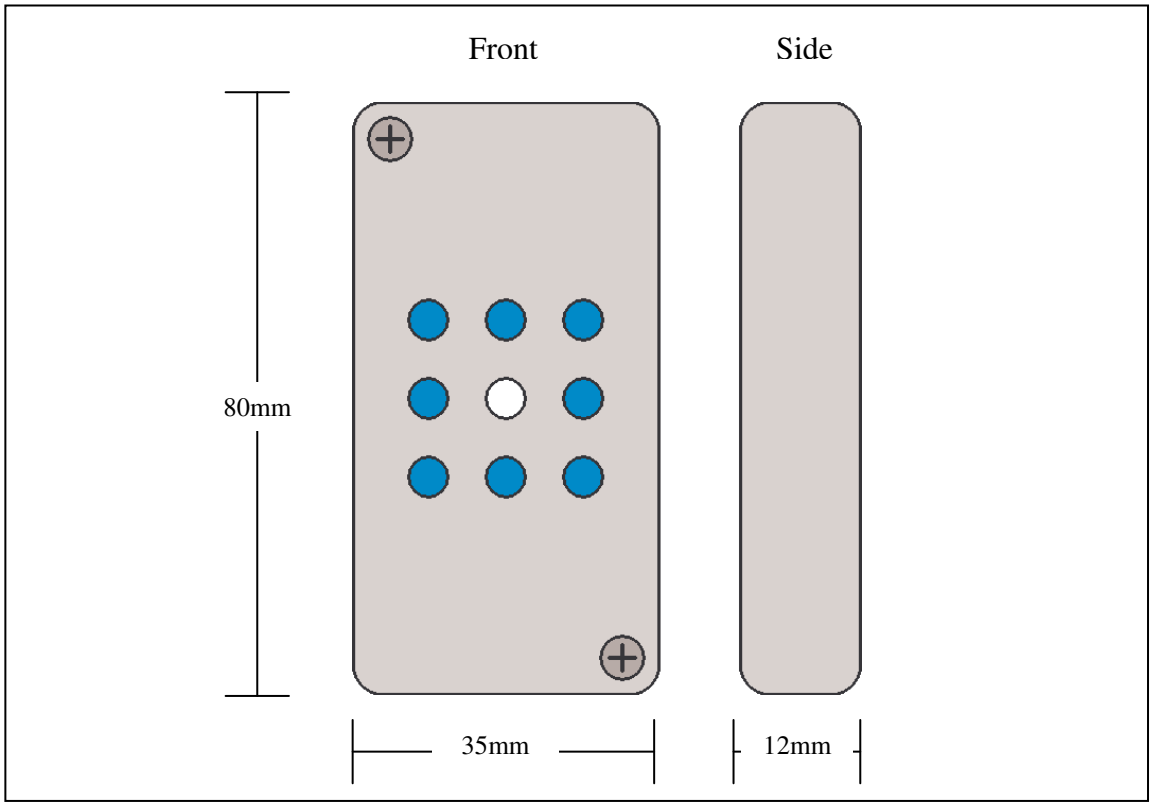


Figure 3-3: 3x3 LED-Grid Layout and Dimensions

3.2 Intended Usage

The transmitter-unit will operate continuously provided the batteries have charge and/or the unit is plugged into an external DC power supply. The transmitter will run for approximately 2.5 hours from a full charge before it needs to be plugged in again.

The USB enabled receiver-unit is plug-and-play ready which means installation simply involves connecting the unit to a PC via an available USB port. The unit will take a few moments to initialise, during which time the indicator LED will glow red. Once ready, the receiver-unit will enter "mouse-mode" which is indicated by the LED glowing blue.

3.2.1 Modes of Operation

The different colours of the LED and what it indicates are listed in the following table:

LED Colour	Function
Red	Receiver initialising/No signal
Blue	Operating in mouse-mode
Green	Operating in keyboard-mode

Table 3-1: LED Colour and Corresponding Modes of Operation

3.2.2 Tilt-Orientation

Both mouse and keyboard-modes are designed to send USB commands to the connected PC according to the tilt-orientation of the transmitter-unit. The orientation is determined by the accelerometer and is divided into nine possible orientation states. Each state is represented by a sector within a 3x3 grid:

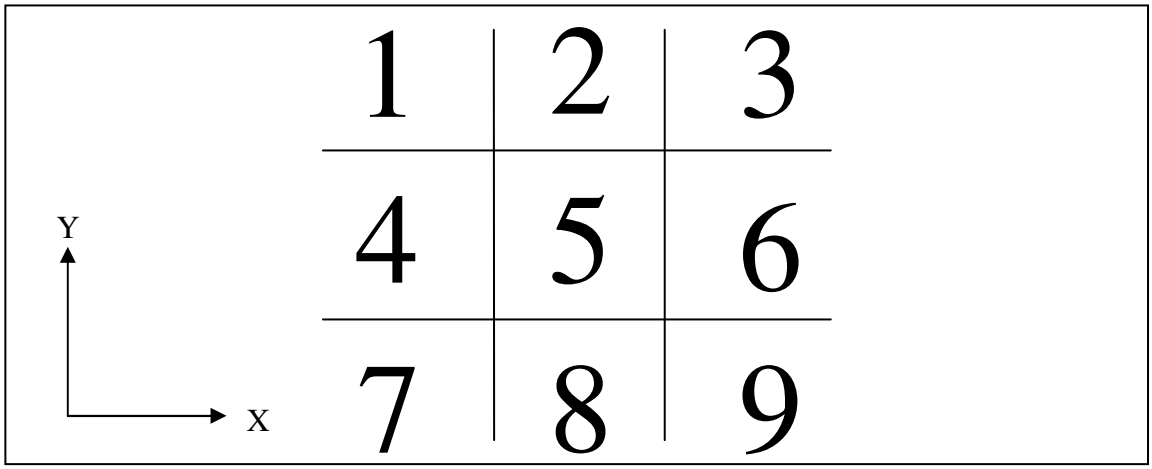


Figure 3-4: Tilt-Orientation Sector Designations

Adjusting the tilt-orientation along the Y-axis is done by tilting the transmitter-unit as shown in the following diagram:

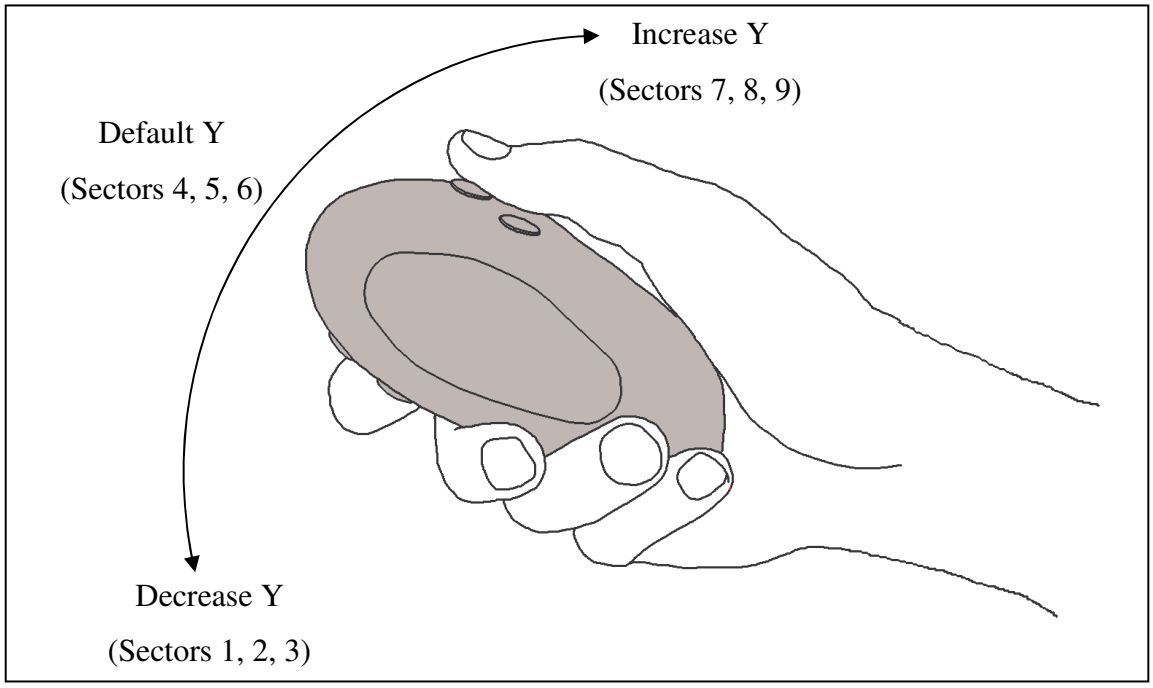


Figure 3-5: Adjusting Tilt-Orientation along the Y-Axis

Similarly, adjusting the tilt-orientation along the Y-axis is done by tilting the transmitter-unit as shown in the following diagram:

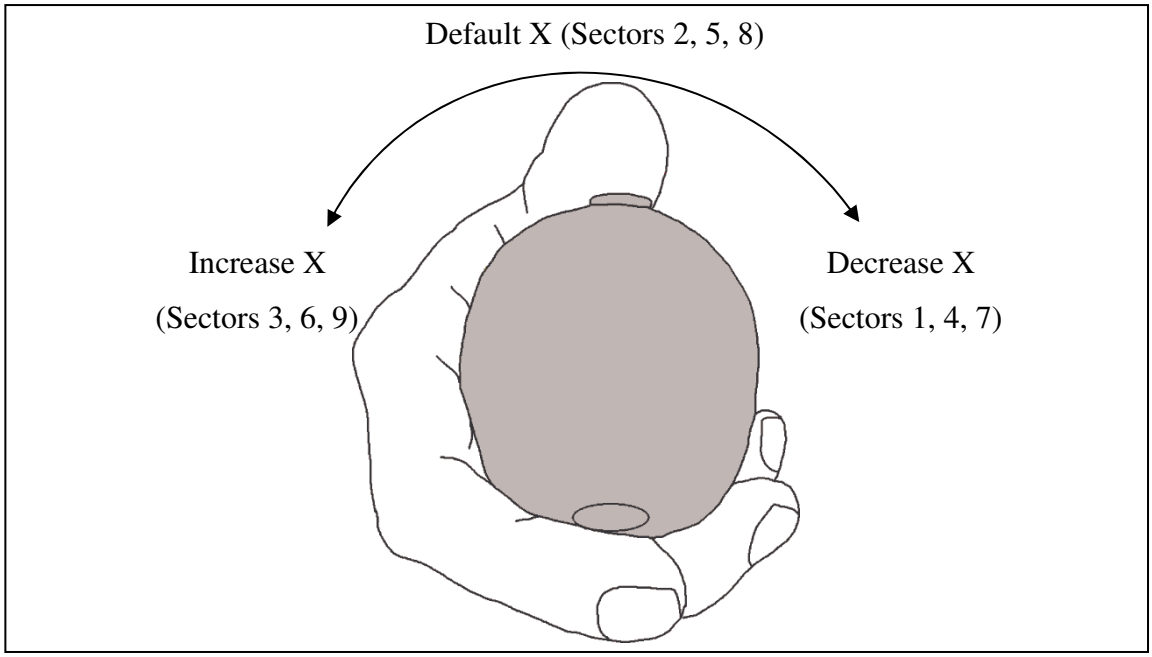


Figure 3-6: Adjusting Tilt-Orientation along the X-Axis

The external, 3x3 LED-grid can be connected to the receiver-unit at any time without having an effect on the behaviour of the system. The receiver-module controls the LEDs in such a way that the user can visually confirm the current tilt-orientation sector. The centre LED (representing sector-5) is white and remains permanently illuminated. The remaining LEDs are blue and are only ever alight if the transmitter-unit is tilted into their corresponding sector. The always-on LED in the centre provides a reference point so that the 3x3 grid is useful even under low or zero light conditions. Example sector indications are shown in the following figure:

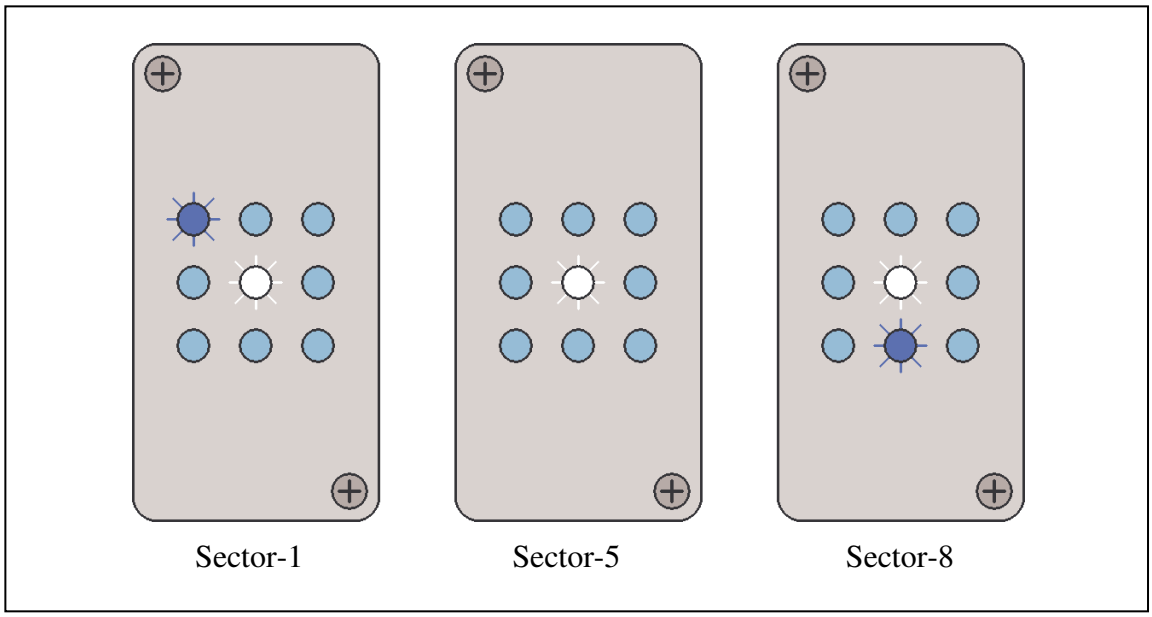
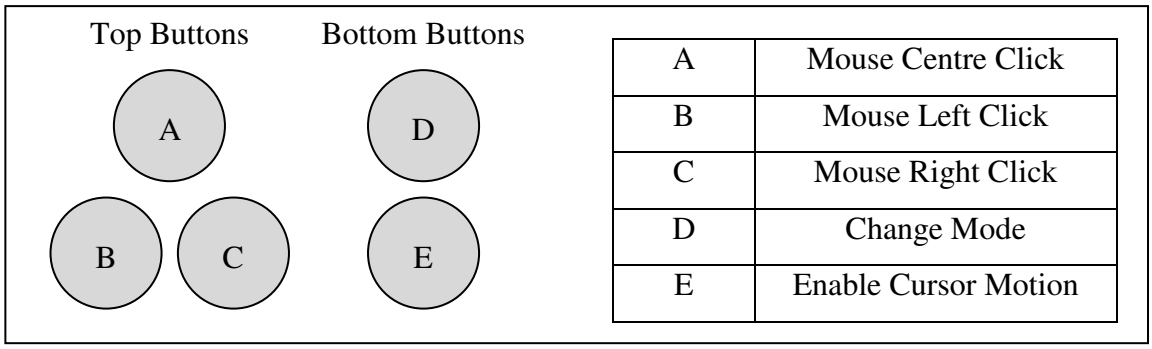


Figure 3-7: 3x3 LED-Grid Display with corresponding Tilt-Orientation Sector

3.2.3 Operating in Mouse-Mode

The following figure shows the functionality of each of the transmitter's buttons when the system is running in mouse-mode.



A	Mouse Centre Click
B	Mouse Left Click
C	Mouse Right Click
D	Change Mode
E	Enable Cursor Motion

Figure 3-8: Mouse-Mode Button Assignments

When using the system in mouse-mode, buttons A, B and C operate identically to the buttons on a regular mouse. The motion of the cursor is altered by means of tilting the transmitter-unit only when button-E is held. This prevents the cursor from moving unintentionally when the transmitter-unit is not being used. The cursor will move in the direction dependant on the current tilt-sector according to the following figure:

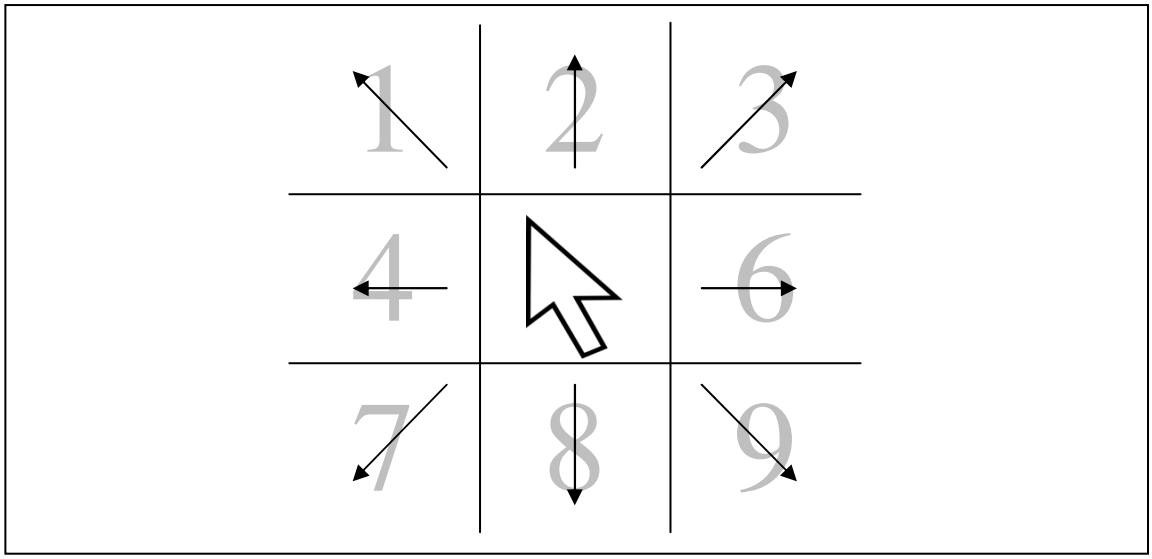
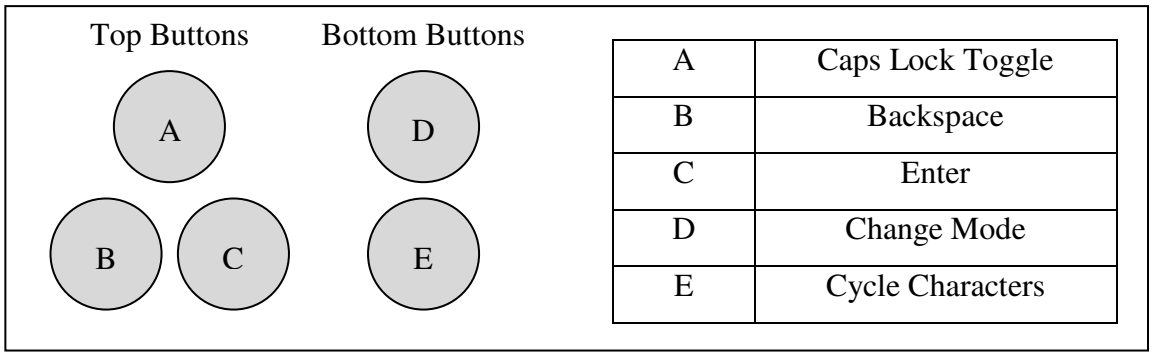


Figure 3-9: Cursor Motion according to Tilt-Orientation Sector

Button-D is used to switch from mouse-mode into keyboard-mode. This button must be pressed and held for one second to take effect. A successful change of operating mode is indicated by a change in colour of the tri-colour LED.

3.2.4 Operating in Keyboard-Mode

The following figure shows the functionality of each of the transmitter's buttons when the system is running in keyboard-mode.



A	Caps Lock Toggle
B	Backspace
C	Enter
D	Change Mode
E	Cycle Characters

Figure 3-10: Keyboard-Mode Button Assignments

When using the system in keyboard-mode, pressing button-E repeatedly will cycle through a set of alpha-numerical characters according to the tilt-orientation of the transmitter-unit when the button is first pressed. The characters are typed to screen then selected and replaced by the next character if the button is pressed again within a one-second duration. If the button is not pressed for a second, the last typed character is assumed to be the desired character and is de-selected. The following diagram shows the characters that are cycled through, according to the initial tilt-orientation:

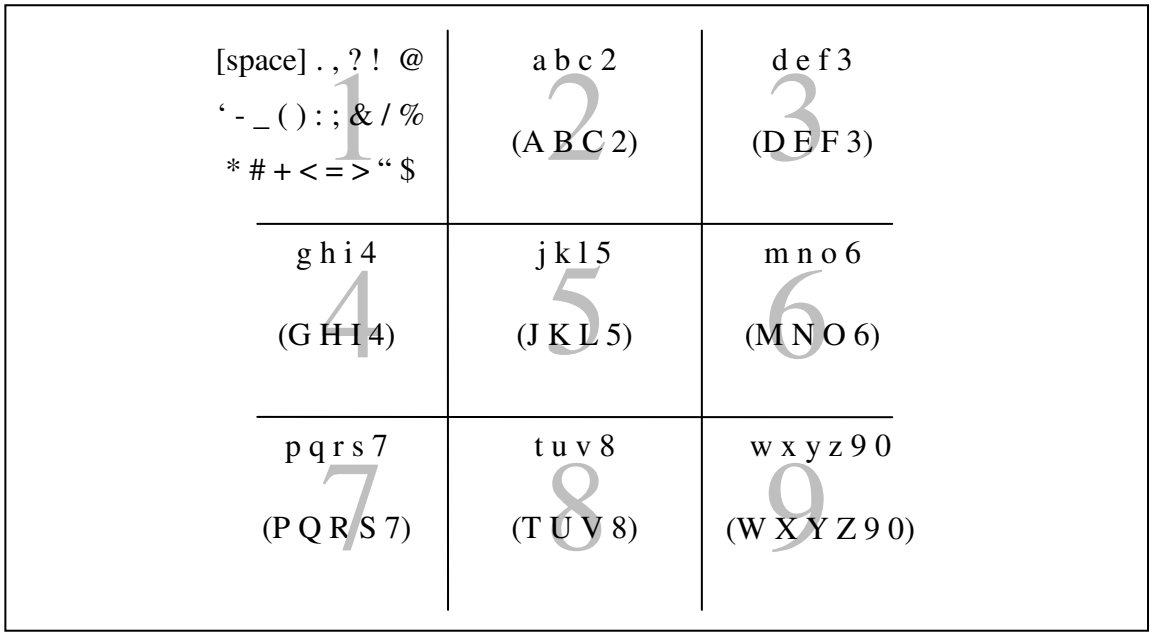


Figure 3-11: Alphanumeric Character Selection according to Tilt-Orientation Sector

For example, if the transmitter-unit had a sector-2 tilt-orientation, and button-E is pressed, the letter ‘a’ (or ‘A’ if Caps Lock is on) would be typed on-screen and then selected. If one second elapses and button-E is not pressed again then the system assumes ‘a’ is the desired character, so it is de-selected and the system becomes ready for the next keystroke. If however, button-E is pressed again within the one-second duration, the character ‘a’ is replaced with ‘b’ and the one-second timer is reset. This process is repeated until the user accepts an entered character.

The tilt-sector designation of character sets was designed to be similar to the layout of a mobile-phone keypad. This was intended to minimise the time required to learn how to type with the device (objective P4) as the keypad format is already familiar to most people.

Pressing button-A will toggle Caps Lock. If a character has just been typed with the unit but not yet confirmed (one-second duration has not elapsed) pressing button-A will toggle the case of the currently selected character (e.g. ‘a’ changed to ‘A’ or ‘Z’ changed to ‘z’).

Pressing button-B emulates a Backspace keystroke. If a character has just been typed with the unit but not yet confirmed (one-second duration has not elapsed) pressing button-B will clear the selected character and cancel the current character input operation.

Button-C operates as an Enter keystroke. If a character has just been typed with the unit but not yet confirmed (one-second duration has not elapsed) pressing button-C will confirm the selected character early instead of awaiting the full second.

Similarly to when in mouse-mode, button-D is used to switch modes. This button must be pressed and held for one second to take effect. A successful change of operating mode is indicated by a change in colour of the LED.

3.3 Data Flow

The overall flow of data is uni-directional. It begins with the user directly manipulating the transmitter-unit and ends with mouse and keyboard commands being sent to the connected PC. The data-flow can be broken down and a basic representation can take the form of a number of relatively linear steps:

1. Status of the buttons and output from the accelerometer are input to the transmitter-unit's microcontroller.
2. The transmitter-unit's microcontroller takes the analogue signals from the accelerometer, and the digital signals from the buttons. This data is processed and encapsulated by additional data to form a data packet.
3. The data packet is sent to the wireless transmitter-module which broadcasts the packet over-the-air.
4. The receiver-module receives the data packet and forwards it to the receiver-unit's microcontroller.
5. The microcontroller decodes the packet into useful data bytes which are sent to the appropriate output control functions (LED and USB drivers).
6. LED control functions update the status of the receiver-unit's LEDs and the USB driver functions send the appropriate USB commands to the PC.

These steps are repeated continuously whilst the two units are operational.

Following is a diagram detailing the basic data flow within the overall system:

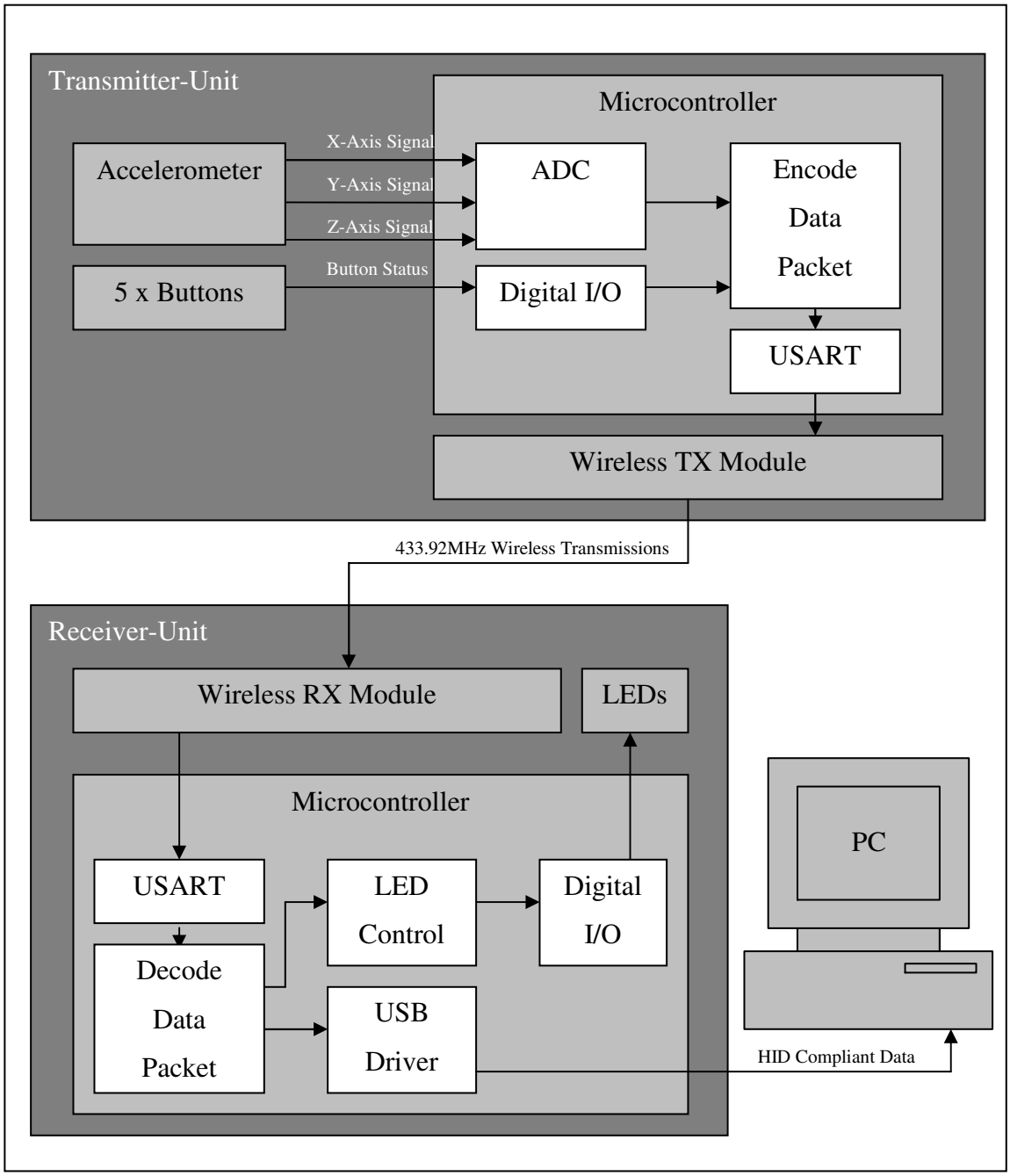


Figure 3-12: Data-Flow throughout the System

The transmitter-unit basically acts as a "dumb" device by simply forwarding the user input data (button status and accelerometer outputs) to the receiver via the wireless link. The actual processing of the input data is performed by the receiver-unit's microcontroller.

3.4 Wireless Data Packets

The need to encapsulate wirelessly transmitted data into data packets is explained in section 2.2.2 Wireless Serial Communications. Packet formation is something that must be consistent between both the transmitter and the receiver-units. The protocol utilised in the final system design underwent a number of revisions to balance processing speed with robustness (a more robust packet format is less likely to incur errors). It was deemed prudent to incorporate only a simple two-byte header into the packet followed by the four data bytes. Testing proved that this simple format was just as robust within the tested environments as a packet with additional authentication protocols. The following table shows the format of the data packet utilised in the final design:

Byte Name	Byte Value (Hexadecimal)	Byte Description
SOH (Start of Header)	0x01	These two bytes form the packet header.
SOT (Start of Text)	0x02	
Data1	0xXX	These four bytes are the actual data sent from the transmitter to be processed by the receiver.
Data2	0xXX	
Data3	0xXX	
Data4	0xXX	

Table 3-2: Utilised Data Packet Format

The actual data bytes are stored as elements of a four-byte data array. A similar array exists for both the transmitter and the receiver firmware and are named Global_TX_Bytes[] and Global_RX_Bytes[] respectively. The value stored in each of the elements at any time represents specific data as detailed in the following table:

Array Element	Data
0	Push-Button States.
1	Accelerometer Output Channel-X
2	Accelerometer Output Channel-Y
3	Accelerometer Output Channel-Z

Table 3-3: Data Represented within Packets

3.5 USB Implementation

A few different protocols exist by which PC peripherals such as keyboards and mice can be interfaced with a PC. The protocols which were considered for this project include:

- Parallel Port
- Serial Port (RS-232)
- PS/2 Port
- USB Port

Ultimately the decision was made to make the device HID compliant using the USB protocol. Following is a list of reasons which factored into this decision:

- Plug-and-play capable (as per objectives P3 and P4).
- Power provided via the USB port (as per objective S9).
- Available on all modern PCs including laptops/notebooks.
- Compatible with recent versions of the main, available operating systems including Windows, Mac OS X and Linux (as per objective P6).

Possibly the primary deciding factor however was the discovery of the existence of a freely-available, firmware-based, USB driver developed by a third party. A web-search for USB drivers implemented in firmware turned up a product named "AVR-USB" from a company called "Objective Development". The web-site also included a number of projects available as example implementations of the driver which proved invaluable to the project development. Further information about the driver (including the web address and product licensing information) can be found in Appendix A. Details of the integration of the USB driver into this project will be discussed later in section 5.3.2 USB Interface Firmware Driver.

3.6 HID Report Descriptor

General USB information (including the required use of a HID Report Descriptor) can be found in section 2.3 USB Protocol – HID Compliance. Basically, the USB driver implemented in the firmware needs to be provided with a report (series of bytes) which matches the format specified by the HID Report Descriptor. The USB driver creates the reports in the defined format, and then transmits the reports to the connected PC.

For the purpose of this particular application, the device had to be capable of emulating a mouse as well as a simple keyboard. Thus the HID Report Descriptor defined two possible report structures identified by the value of the first byte within the array. The value of this "Report ID" byte is either 1 or 2, and indicates either a keyboard or mouse report respectively. The structures of these reports are outlined in the following tables:

Byte	Data
B0	Report ID. Equals '1' in this case, indicating a keyboard report.
B1	Modifier Status. Each bit represents the state of one of the 8 possible modifiers (0=off, 1=on). The modifiers are (from least to most significant bit): b0:Left Ctrl b1: Left Shift b2: Left Alt b3: Left GUI key b4: Right Ctrl b5: Right Shift b6: Right Alt b7: Right GUI key
B2	Bytes 2 to 7 can each be a different keyboard keystroke (excluding modifier keys). Up to 6 keystrokes can be sent simultaneously using this report structure. For this particular application though, only one keystroke is ever sent at a time. The possible values range from 0x00 to 0x65 and the corresponding keystrokes are defined in the document *HID Usage Tables* mentioned in section 2.3.2 HID-Class.
B3	
B4	
B5	
B6	
B7	

Table 3-4: Keyboard Input Report Format (for HID)

Byte	Data
B0	Report ID. Equals '1' in this case, indicating a keyboard report.
B1	Keyboard LED status. The first five bits in this byte represent the status of the five possible keyboard LEDs (0=off, 1=on). The LEDs are (from least to most significant bit): b0: Num Lock LED b1: Caps Lock LED b2: Scroll Lock LED b3: Compose LED b4: Kana LED b5 to b7: N/A (padded with zeros) This output functionality isn't utilised in the final design but its existence allows for future expansion (e.g. a hardware Caps Lock indicator).

Table 3-5: Keyboard Output Report Format (for HID)

Byte	Data
B0	Report ID. Equals '2' in this case, indicating a mouse report.
B1	Mouse Buttons. The first three bits represent the state of three mouse buttons (0=off, 1=on). The possible buttons are (from least to most significant bit): b0: Button 1 (Left Click) b1: Button 2 (Right Click) b2: Button 3 (Centre Click) b3 to b7: N/A (padded with zeros)
B2	This byte is signed and thus possible values range from -127 to 127. This value determines the number of pixels by which the mouse cursor shall be displaced along the X-Axis (-ve values to the left, +ve values to the right.
B3	This byte is signed and thus possible values range from -127 to 127. This value determines the number of pixels by which the mouse cursor shall be displaced along the Y-Axis (-ve values upwards, +ve values downwards.

Table 3-6: Mouse Input Report Format (for HID)

The actual HID Report Descriptor utilised for this project forms part of the file usb.c which can be found on the attached CD (see Appendix G).

The receiver-unit's firmware operates by first processing the data from the received wireless packets, then creating appropriate input reports depending on the transmitter-unit's status and the current operating mode (mouse or keyboard). These reports take the form of simple arrays of data bytes. The arrays are sent to the USB driver which then transmits the data to the PC according to the USB and HID protocols.

3.7 Using a Serial Level Converter for Debugging

System debugging often required having the microcontroller send specific data to a PC which could then be shown on-screen using a serial terminal emulator. For example, the transmitter-unit's microcontroller could be programmed to continuously transmit the value read from one of the accelerometer outputs to enable manual calibration. The microcontroller's USART utilises TTL line levels for the serial signal (0V for 0 and +5V for 1). This is incompatible with the levels specified by the RS-232 protocol implemented by a PC's serial port (+5 to +15V for 0 and -5 to -15V for 1). Thus line level conversion is required for the PC to be able to recognise the serial data. The concept of converting serial protocol line levels from one standard to another is explained in section 2.2.1 Serial Data Level Conversion. A TTL to RS-232 converter unit was created specifically for this project and was used for debugging both the transmitter and receiver-units. Because it was only required for debugging, the unit remained external and was not incorporated into the final design. The schematic and component details for the converter are located in Appendix E.

4 Transmitter-Unit

This section contains specific details pertaining to the design and construction of the transmitter-unit and its subsystems. This is broken into three parts; the electronics used in the transmitter, the physical assembly of the unit and the firmware programmed into the microcontroller.

4.1 Transmitter-Unit Electronics

The design of the transmitter-unit produced the following requirements:

- Five push-button inputs.
- An accelerometer to sense tilt-orientation.
- A transmitter to wirelessly send data.
- The unit must be capable of operating from batteries or a mains-connected DC plug-pack.
- All the electronics must fit inside a relatively small enclosure.

The following circuit diagram details the components selected to meet these requirements and the way in which they were connected.

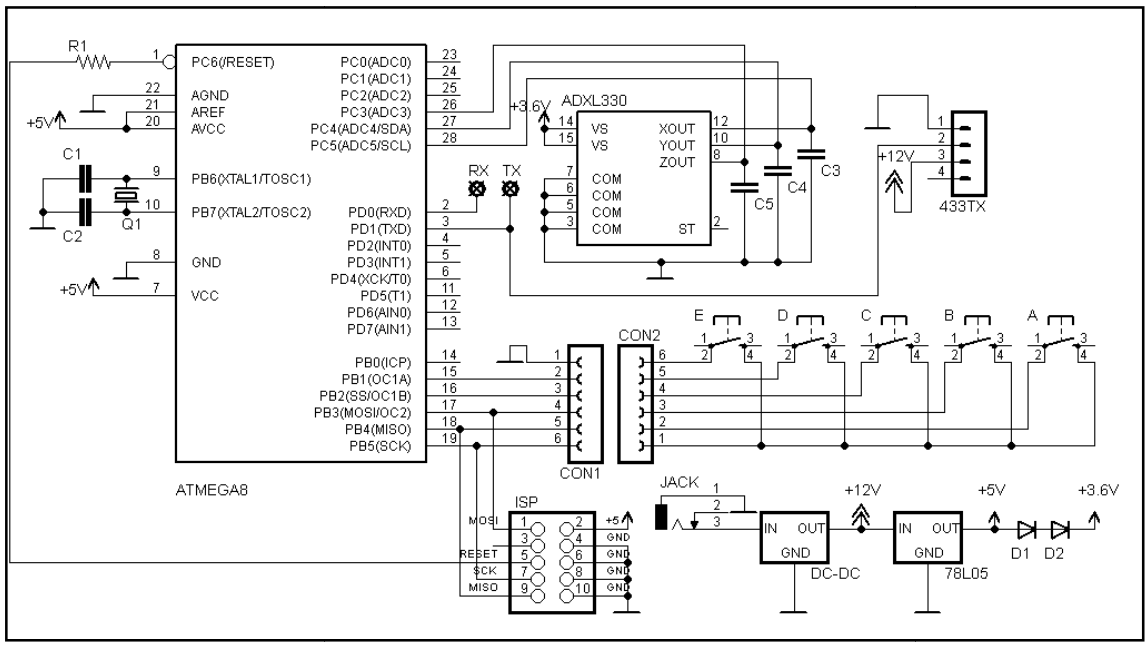


Figure 4-1: Transmitter-Unit Circuit Schematic

Note: A full-size, colour version of the schematic and the list of component details can be found in Appendix B.

Specific information regarding each of the main components of the transmitter system will be provided in the following sections. This information will include important device specifications and an explanation of why the relevant component was selected.

A list of all the components used in the transmitter-unit including the corresponding manufacturer, supplier and cost details can be found in Appendix D.

4.1.1 Transmitter Microcontroller

The basic design of the transmitter circuit defined the requirements of the utilised microcontroller. The required capabilities include:

- Serial communications – for sending data to the wireless transmitter-module and also used for debugging.
- ADC (Analogue to Digital Converter) with at least three channels – to convert the analogue signals from the accelerometer into useable data bytes.
- At least five digital input channels – for the five push-buttons.
- In-System Programmable – so that firmware alterations can be made without removing the microcontroller from the circuit.

The decision as to which microcontroller would be used for the *transmitter* was made *after* a microcontroller was selected for the *receiver*-unit. Since this already selected device also met the requirements of the transmitter microcontroller, it was decided that the two devices would be identical. This would reduce the amount of time required to create code functions that would be common to both microcontrollers (specifically hardware initialisation and USART control). Some microcontrollers utilise different names for similar hardware registers which would require slightly modified code to perform identical operations. Using identical microcontrollers prevented this complication.

The microcontroller selected for use in the transmitter was the AVR ATmega8 manufactured by Atmel. Additional data pertaining to this device is located in Appendix D and the datasheet is included on the accompanying CD (see Appendix G).

The clock signal for the microcontroller is supplied by means of a 12.0MHz crystal, again identical to the microcontroller used for the receiver-unit. The decision behind the clock speed for the *receiver* relates to the USB driver and is explained in section 5.1.1 Receiver Microcontroller. It was deemed prudent to clock the transmitter's microcontroller at an identical speed to ease the synchronisation of serial communications with the USART.

4.1.2 Serial Interface

Serial communications were required from the microcontroller for two reasons:

1. Data must be forwarded to the wireless transmitter-module in serial format.
2. Serial transmission and reception is required for system debugging.

The microcontroller's onboard USART was utilised for serial communications and connected directly to the wireless transmitter-module.

The USART was also employed for system debugging by means of a TTL to RS-232 level converter unit (see section 3.7 Using a Serial Level Converter for Debugging).

4.1.3 TX Module

A 433.92MHz wireless serial transmitter (TX) module was selected due to its low cost and ease of implementation. The unit has a data pin which simply connects to the serial data output from the microcontroller (USART). Provided the module is powered, this is all that is required for the unit to wirelessly transmit serial data.

The transmission range of the module is proportional to the voltage level of its supply. This fact had a significant influence on the selection of a suitable power-module (see section 4.1.6 Power-Module).

Additional data pertaining to the wireless transmitter-module can be found in Appendix D and the datasheet can be found on the accompanying CD (see Appendix G).

4.1.4 Accelerometer

There are two important features that differ between accelerometers available on the market today. They are:

1. The number of measurable axis – Single-axis accelerometers are available which sense acceleration in a single dimension. Two-axis accelerometers are also available which measure acceleration along two axis perpendicular to each other. Finally, three-axis accelerometers contain sensors which measure acceleration in all three dimensions.
2. The output data type – Accelerometers output the measured acceleration in one of two formats (see section 2.1 Measuring Tilt with an Accelerometer). The first is an analogue voltage which represents the acceleration by a voltage level proportional to the supply voltage range. The second output format is a digital signal whereby the data is output in the form of serial data bytes.

It was deemed preferable to utilise an accelerometer with analogue outputs as this would require less complex code to interface with the microcontroller.

Additionally, a three-axis microcontroller would be proffered as it would allow for various mounting orientations without hindering the tilt-orientation measuring functionality. Also, three measured axis would allow for a wider range of tilt-orientation variations.

The accelerometer selected to meet these specifications was the ADXL330. Additional data pertaining to this unit is located in Appendix D and the datasheet can be found on the accompanying CD (see Appendix G). The accelerometer was purchased pre-mounted on a break-out board to ease incorporation into the transmitter circuit.

4.1.5 Buttons

The transmitter design specified the incorporation of five push-buttons which were to be mounted on the enclosure in such a way that they would be easily accessible and operable whilst the unit was held by a single hand. The push-buttons therefore had to be large enough that they could be easily located by fingers, without requiring the user to visually examine the unit. It was also deemed preferable that the buttons provide a solid tactile response when depressed so as to provide the user assurance of when a button has been properly depressed, again without having to visually examine the unit.

In keeping with objective S6, it was desirable for the buttons to "suit" the transmitter enclosure to maintain a desirable aesthetic. Thus the buttons were not selected until after a suitable enclosure had been chosen.

Large, momentary-contact, PCB-mount, tactile push-buttons were selected along with large, round, black button caps to maintain the aesthetic. Because the buttons were designed to be mounted on a PCB, they were each soldered to a small section of stripboard. Plastic stand-offs were then attached to the boards which provided a means of attaching the button "modules" to the transmitter enclosure. Hook-up wire was soldered to the relevant copper strips on the boards which provided a means of interfacing the buttons with the microcontroller's I/O pins.

4.1.6 Power-Module

A number of the defined objectives (section 1.1 Project Objectives) formed the basis on which the power-module selection was made:

- Objective P1 specified that the transmitter-unit should operate wirelessly. In addition to wireless data communications, this objective also required that the transmitter-unit be self-powered. Thus the unit had to be battery-operated.
- Objective P2 required the power-module to be small enough to fit within a hand-held enclosure.
- Objective S1 combined with the wireless transmitter-module characteristics (higher supply voltage increases transmission range) defined an ideal voltage supply range.
- Objective S2 stated that the transmitter should preferably be rechargeable and capable of operating whilst being re-charged.
- Objective S3 defined the preference for a long operating time from a single charge.

The first four objectives listed here (P1, P2, S1 and S2) led to the selection of an Altronics DC-DC Converter module. At the design stage, it was deemed likely that objective S3 would also be met because of the low power requirements of the other selected components. This objective however, was not confirmed until after the circuit was built and testing was conducted.

The power-module was designed around Texas Instruments TL499A Switching regulator. It takes a DC input voltage and outputs a (modifiable) output voltage. The module also incorporates trickle-charge capability via an external DC source for use with NiCd of NiMH batteries. For this application, the input voltage was provided by two 1.2V, AA sized, series connected NiMH batteries (2.4V). The module was configured to output 12V which was deemed ideal for the transmitter-module supply.

This output level was ideal for the transmitter-module, however other components within the transmitter circuit required different supply voltage levels as detailed in the following table:

Transmitter-Module Component	Specified Supply Voltage
433.92 MHz Serial Transmitter-Module	+12V (For Large Transmission Range)
AVR ATmega8 Microcontroller	+5V (Ideal)
ADXL330 Accelerometer	+3.3V (Ideal)

Table 4-1: Transmitter-Unit Component Supply Voltage Requirements

Thus, three, distinct voltage levels were required for various components within the unit. The power-module directly supplied the transmitter-module with a 12V source. A 78L05, +5V voltage regulator was incorporated into the power-module to provide a regulated supply voltage for the microcontroller. Two 1N4148 diodes connected in series to the +5V output created a 1.4 volt drop. Thus a +3.6V supply level existed for the accelerometer. Although this differed slightly from the accelerometer's ideal supply voltage (+3.3V) it was still well within the allowed supply range and was implemented with readily available components.

Testing of the completed transmitter-unit with the final firmware version, found that the selected batteries and power-module provide the unit with 2.5 hours of power from a single charge.

4.2 Transmitter-Unit Physical Assembly

The following sections detail the enclosure used for the transmitter-unit as well as how the internal electronics were arranged to fit. It was important that a suitable arrangement of the internal components be found which would enable everything to fit within the enclosure whilst still allowing for circuit modification and firmware alterations should the need arise.

4.2.1 Transmitter Enclosure

The most important factor in selecting a suitable enclosure for the transmitter-unit was objective P2 which specified that the unit should be small enough to fit in a user's hand. Secondary objective S4 required the enclosure to be rounded with no sharp edges. Objective S6 also factored into the selection by specifying that the enclosure should be aesthetically pleasing.

These three key factors led to the final decision. The selected enclosure was previously utilised as the housing for a cordless screwdriver. The existing batteries, motor and other electronics were removed from the unit to make room for the transmitter electronics. Following is a photo of the original screwdriver unit prior to any modification.

Figure 4-2: Cordless Screwdriver which became the Transmitter-Unit Enclosure

The following photograph shows the modified enclosure with all of the transmitter-unit electronics installed. At this stage of construction, an existing hole in the enclosure (originally used for a button which switched the rotational direction of the cordless screwdriver) was left exposed to provide access to the ISP header, thus enabling firmware modifications without the need to regularly disassemble the unit. The transmitter microcontroller is also visible through the hole.

Figure 4-3: Assembled Transmitter-Unit with Externally Accessible ISP Header

The next photograph shows the final version of the modified enclosure. The three thumb-buttons are visible near the top of the unit and the external DC-supply jack can be seen at the bottom. The exposed holes have been filled with putty typically used for auto body-repair. Once hardened, the area was sanded smooth then spray-painted black.

Figure 4-4: Fully Enclosed Transmitter-Unit

Machining of the transmitter enclosure required the drilling of six holes (five buttons plus the DC jack) plus grinding down of some of the internal plastic supports to make room for the electronics. The drilling, cutting and shaping was all performed using a Dremel rotary tool and a set of needle files.

4.2.2 Layout of Transmitter Components and Circuit Boards

Many configurations were tested before a suitable arrangement was found in which all the components would fit within the enclosure. Of the main components, only the buttons and the DC jack were attached permanently to the enclosure. Everything else was installed in such a way that it could be easily removed should the need arise. The major components are listed below in the same order that they were installed:

1. Button-Modules - The button modules had to be permanently glued to the inside walls of the unit as this was deemed the most effective method of providing a sound mechanical attachment without using externally visible bolts or screws. Thus the buttons were installed whilst maintaining the aesthetic (objective S6).
2. DC Jack - The externally accessible DC jack was the other permanently attached component. The jack was also glued to the enclosure for the same reasons as the button-modules (sound mechanical attachment with negligible aesthetic impact).
3. Power-Module/Battery-Pack - By far the largest components of the unit, the power-module and battery-pack were attached together before being installed. It was deemed prudent that they be fitted first (apart from the permanently installed components listed previously) as it was the least likely candidate for re-arrangement if required later. Simply put, the power-module/battery-pack was installed first and the remaining components were fit around them.
4. Microcontroller/Peripheral Components - The microcontroller was attached to a section of stripboard which was in turn was attached to another section of stripboard via a DIP (Dual In-Line Package) socket and pin arrangement. The second layer contained the clock-source crystal and capacitors as well as connection points for the other components within the unit. This arrangement was deemed necessary in order to fit the microcontroller within what little space was available.

5. Transmitter-Module – Initially mounted towards the bottom of the unit, the transmitter-module was relocated as far from the power-module as possible to minimise the risk of any RF-noise that might affect the wireless signal. Thus the transmitter was relocated to the top of the unit. This new location was close to the peripheral components circuit board so the transmitter-module was attached directly to it instead of via hook-up wire like the other components.
6. Accelerometer – The accelerometer was the final component to be installed. The orientation of the accelerometer's axis' could be compensated for within the firmware, so this was not a factor in deciding its mounting location. Ultimately, the accelerometer was attached in its final position because this was the most easily accessible space that remained within the unit.

The following photo shows how the internal components were arranged within the transmitter-unit.

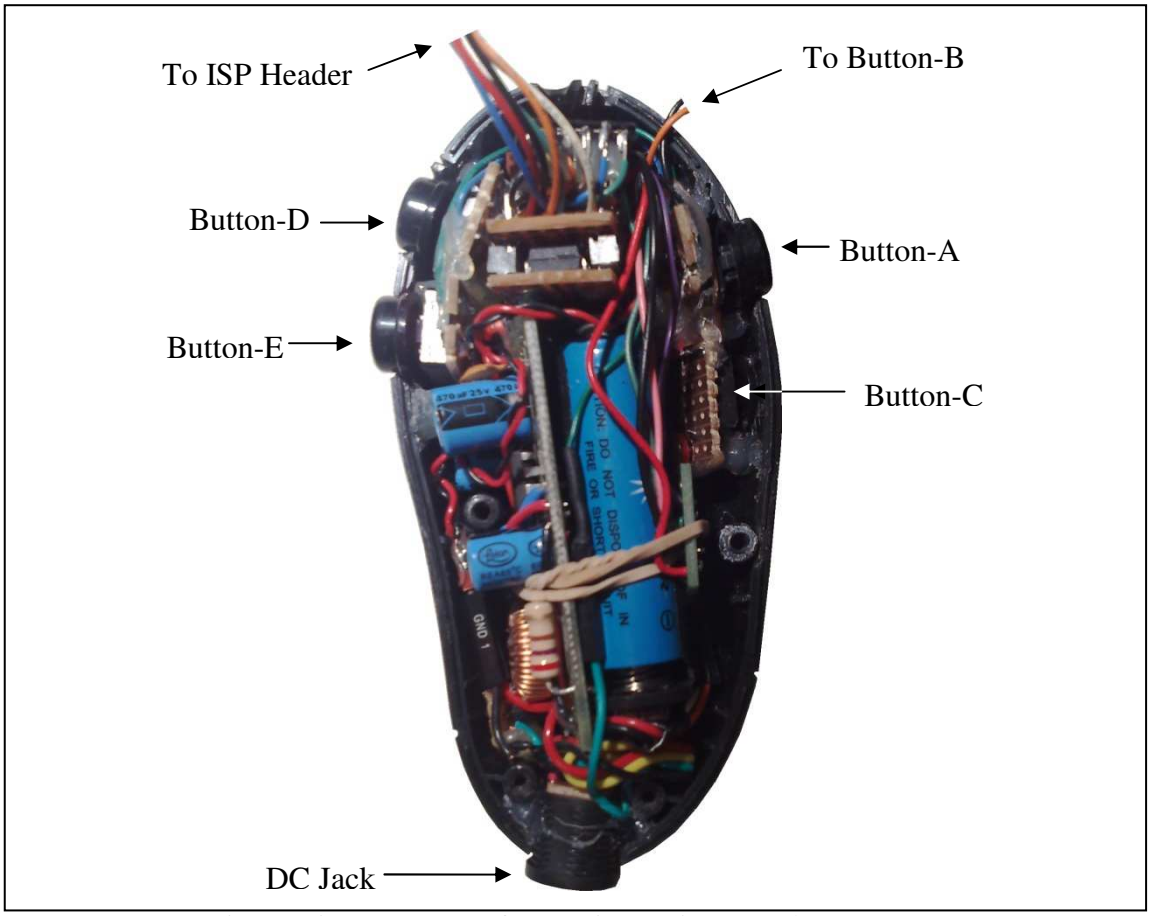


Figure 4-5: Arrangement of Transmitter-Unit's Internal Components

The next figure details the final mounting positions of the major components within the transmitter-unit enclosure.

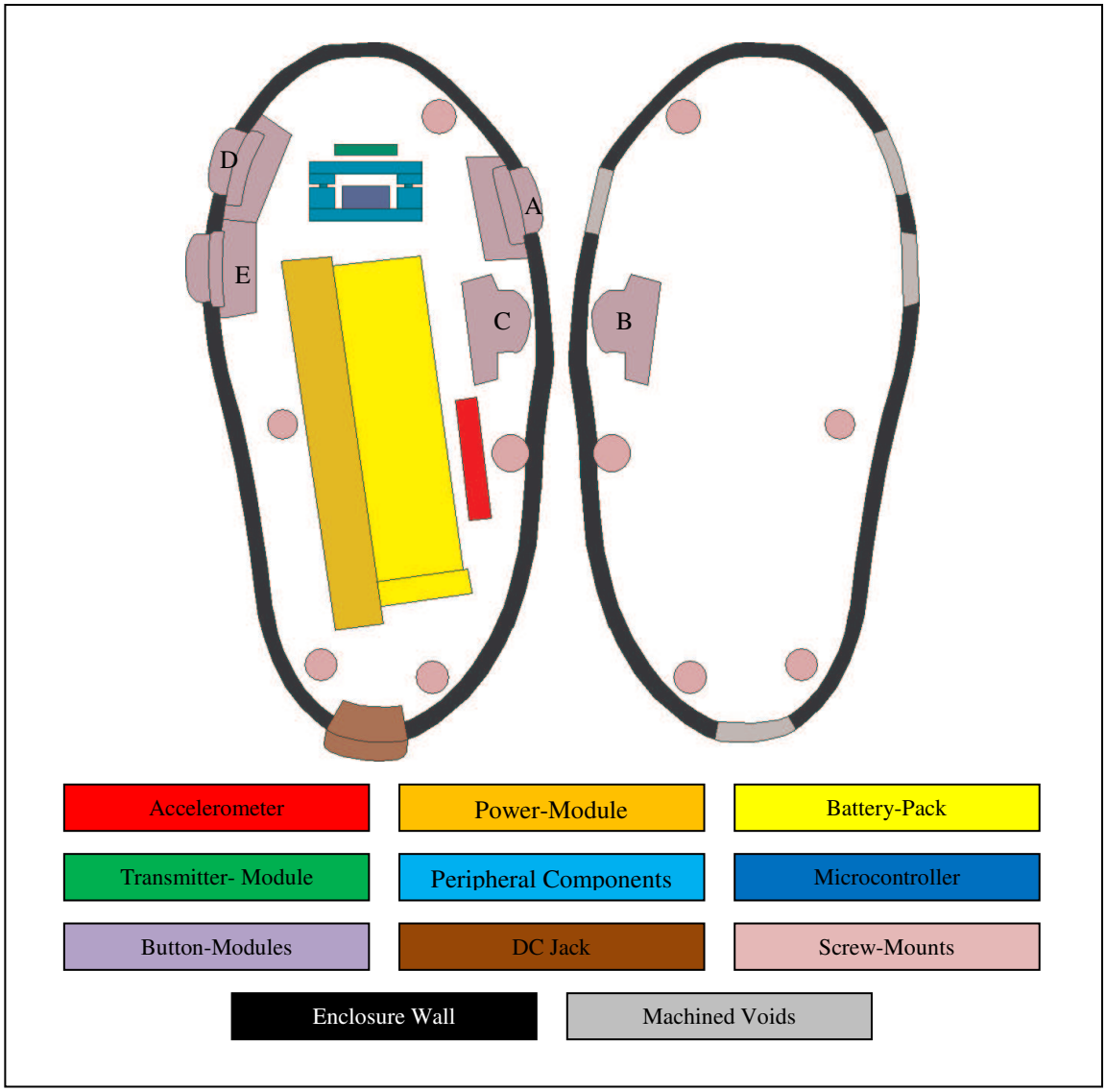


Figure 4-6: Transmitter-Unit Internal Component Layout

4.3 Transmitter-Unit Firmware

The firmware developed for the transmitter-unit was relatively straightforward. As mentioned previously (section 3.3 Data Flow), the transmitter-unit performs minimal processing of data and simply serves to capture button states and accelerometer outputs, then forward this data to the receiver-unit. This data is encapsulated in a wireless packet as per the specifications outlined in section 3.4 Wireless Data Packets.

The code for the transmitter-unit is separated into two c-code files, each of which can be found on the included CD (see Appendix G). The files include:

- WiSHABI_TX.c – the main code for the transmitter.
- serial.c – the required USART related functions.

The following flowchart provides a general representation of the complete firmware execution.

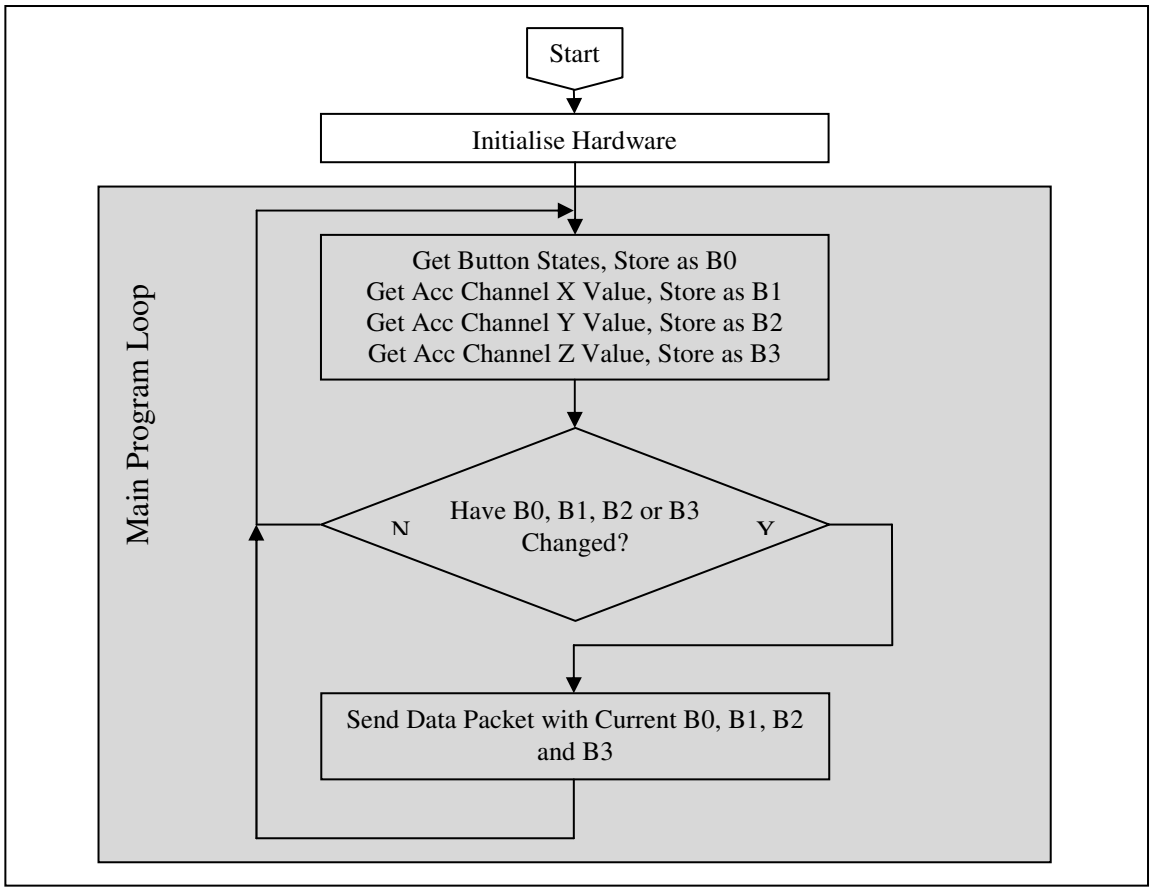


Figure 4-7: Transmitter Firmware Flowchart Representation

The execution of the transmitter firmware can be broken down into three major collections of functions:

1. Serial interface (including packet formation and transmission).
2. ADC interface required for the accelerometer.
3. Digital I/O interface required for the push-buttons.

These three areas are covered in more detail in the following sections.

4.3.1 Serial Interface

The transmitter microcontroller's USART functionality was utilised primarily for the purpose of serial transmission. A serial "driver" was written, containing the minimum functions required to transmit a byte of data. The driver included a function named USART_Transmit() which, when called with a data byte as an input parameter, simply "transmits" said byte via the TX pin of the microcontroller. This pin is connected directly to the wireless transmitter-module, thus enabling wireless, serial, data transmission. The code for the serial driver (serial.c) is located on the accompanying CD (see Appendix G).

Another function was written specifically to transmit the unit's input data within the defined structure of a data packet. Send_Packet() calls USART_Transmit() multiple times to sequentially transmit the individual bytes which make up the packet. The data packet format is defined in section 3.4 Wireless Data Packets. The table in this section details the data values for the bytes which make up the packet. Each of these values is sequentially sent as a parameter to USART_Transmit in order to transmit the packet. The code function Send_Packet() is part of the main code file for the transmitter-unit (WiSHABI_TX.c).

The values of the actual input data bytes which form part of the data packet are stored as elements of a globally accessible array of bytes (see section 3.4 Wireless Data Packets). Send_Packet() accesses these elements when they need to be transmitted. The values of the elements are updated every iteration of the main program loop so the input data is constantly up-to-date.

4.3.2 ADC (Analogue to Digital Converter)

As previously mentioned (section 4.1.4 Accelerometer) the accelerometer device utilises analogue signals as the output from its three channels (X, Y and Z). Thus the analogue to digital converter (ADC) capability of the microcontroller had to be incorporated into the firmware.

The ATmega8 microcontroller supports 12 bit analogue to digital conversion on up to eight channels. There is however only one actual ADC system, so reading a voltage level on a specific channel first requires specifying which channel the ADC should connect to.

A function was written for this purpose. This function (called Get_ADC()) takes a byte value as an input representing the channel to read and then returns the converted value in the form of a data byte.

Although the onboard ADC subsystem is capable of retuning digital values with a resolution of up to 12 bits, the firmware only utilises the 8 most significant bits. An 8 bit value was deemed sufficient for the purpose of this application. This has the additional advantages of reducing both ADC code complexity as well as the required memory space. Thus the returned value from Get_ADC() is a single, 8-bit, data byte.

For every iteration of the main firmware loop, Get_ADC() is called for each output from the accelerometer. The returned values are stored in the appropriate elements of the main data array (Global_TX_Bytes[]) ready to be transmitted wirelessly as part of a data packet.

The Get_ADC() function is part of the main code listing for the transmitter-unit (WiSHABI_TX.c) which is included on the CD.

4.3.3 Button Detection

The data packet format listed in section 3.4 Wireless Data Packets specifies that the first byte of actual data represents the states of the five push-buttons mounted on the transmitter-unit. As can be seen from the transmitter circuit diagram (Appendix B) the push-buttons are connected to the first five pins of port B. Thus the state of each button can be determined simply by reading the value on port B.

When the transmitter microcontroller first initialises its data ports, the internal pull-up resistors on port B are activated. This means the default state for each button is high (logical 1) and to bring them low (logical 0), the button must short the corresponding pin with ground. The result is a logical 1 for each button by default and a logical 0 when the button is pressed "on". This is counter-intuitive so measures were taken to prevent confusion. When the byte on port B is read in, the value is inverted. Thus the resultant byte sent from the transmitter-unit and processed by the receiver-unit uses the following format:

Bit	Connected Pin	Description (1 = ON, 0 = OFF)
0	PORTB.0	Button A Status
1	PORTB.1	Button B Status
2	PORTB.2	Button C Status
3	PORTB.3	Button D Status
4	PORTB.4	Button E Status
5	N/C	Not Connected (Always = 0)
6	N/C	
7	N/C	

Table 4-2: Button Status Data Byte Format

The functionality described in this section is implemented in the firmware as part of the main transmitter-unit program listing (WiSHABI_TX.c).

5 Receiver-Unit

This section contains specific details pertaining to the design and construction of the receiver-unit and its subsystems. This is broken into three parts; the electronics used in the receiver, the physical assembly of the unit and the firmware programmed into the microcontroller.

5.1 Receiver-Unit Electronics

The design of the receiver-unit produced the following requirements:

- A receiver to capture wirelessly broadcasted data packets.
- Nine regular LEDs arranged in a 3x3 grid to display tilt-orientation.
- A tri-coloured (RGB) LED to indicate operating mode or loss of wireless signal.
- USB connectivity via a standard-B type USB connector.
- Capable of operating from the voltage supplied via the USB port.

The following circuit diagram details the components selected to meet these requirements and the way in which they were connected.

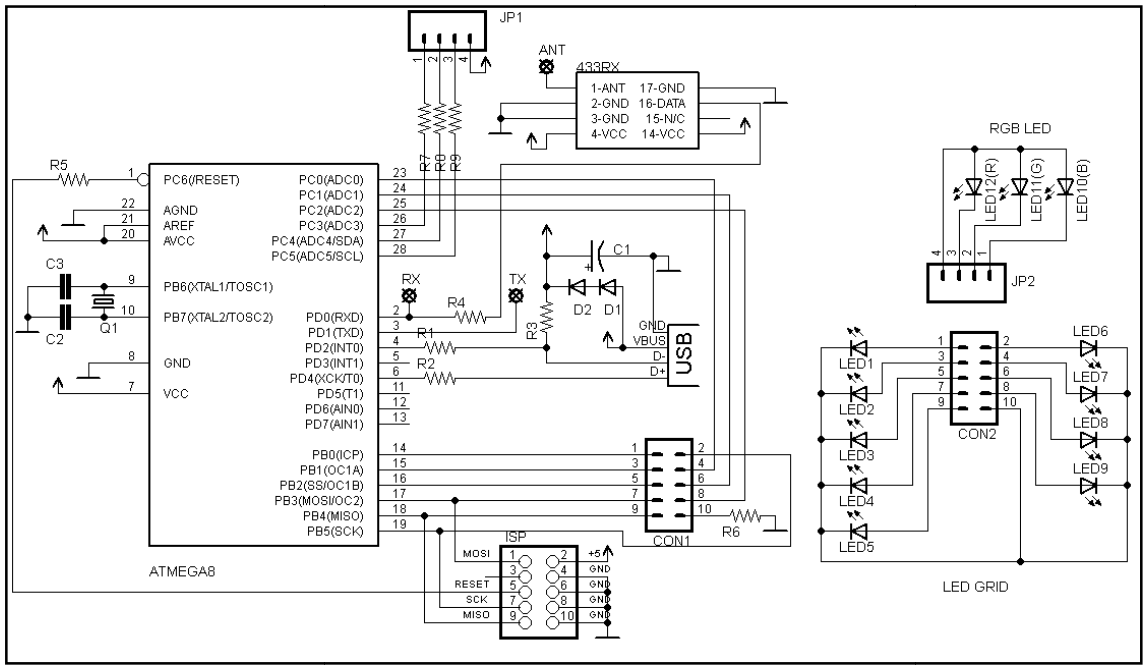


Figure 5-1: Receiver-Unit Circuit Schematic

Note: A full-size, colour version of the schematic and the list of component details can be found in Appendix C.

Specific information regarding each of the main components of the receiver system will be provided in the following sections. The information will include important device specifications and an explanation of why the relevant component was selected.

A list of the components used in the receiver-unit including the corresponding manufacturer, supplier and cost details can be found in Appendix D.

5.1.1 Receiver Microcontroller

The basic design of the receiver circuit defined the requirements of the utilised microcontroller. The required capabilities include:

- Serial communications – for receiving data from the wireless receiver-module and also for debugging.
- Operable at a Clock-Speed of 12.0MHz – to meet the requirements of the firmware USB driver.
- Hardware interrupt enabled I/O pin – for use by the firmware USB driver.
- At least twelve digital output channels – nine to control the 3x3 LED-grid and three to control the RGB LED.
- In-System Programmable – so that firmware alterations can be made without removing the microcontroller from the circuit.

The most important factor that influenced the choice of microcontroller was the requirements defined by the USB driver. More information on this is located in a later section (5.1.4 USB Interface Hardware) but ultimately, the AVR ATmega8 microcontroller was selected as this is the device for which the USB driver was written (so no modifications were required). Additionally, the ATmega8 met all the other requirements listed here.

5.1.2 Serial Interface

Serial communications were required from the microcontroller for two reasons:

1. Data must be received from the wireless transmitter-module in serial format.
2. Serial transmission and reception is required for system debugging.

The microcontroller's onboard USART was utilised for serial communications and connected directly to the wireless receiver-module.

The USART was also employed for system debugging by means of a TTL to RS-232 level converter unit (see section 3.6 Using a Serial Level Converter for Debugging).

5.1.3 RX Module

A 433.92MHz wireless serial receiver (RX) module was selected due to its low cost and ease of implementation. The unit has a data pin which simply connects to the serial data input at the microcontroller (USART). Provided the module is powered, this is all that is required for the microcontroller to receive wirelessly transmitted serial data.

To help increase the range of the wireless system, an antenna was incorporated into the design and mounted on the receiver-unit's enclosure. A ¼ wave whip antenna was selected as it was designed specifically for the 433.92MHz frequency. Testing proved that the inclusion of the antenna increased the unobstructed reception range from less than one metre to approximately 45 metres before any signal loss occurred.

The datasheet for the receiver-module is included on the CD (see Appendix G).

5.1.4 USB Interface Hardware

A suitable, firmware-only USB driver was selected to apply to this project. The firmware itself is discussed later in section 5.3.2 USB Interface Firmware. The hardware requirements for implementing USB are specified with the firmware. Only two I/O pins are required from the microcontroller, provided that one of them can be used as a hardware-interrupt. In addition, three resistors, two 1N4148 diodes, a single 10µF electrolytic capacitor and the actual USB connector are also required. All of these components and their required arrangement can be seen in the circuit schematic for the receiver-unit (Appendix C).

Additionally, the microcontroller has to be clocked with an external crystal at 12, 15, 16 or 20MHz to be compatible with the firmware driver. A clock speed of 12.0MHz was selected as this was the default configuration for the driver meaning no additional firmware modifications were required.

5.1.5 LEDs

The reasons for incorporating the tri-colour LED and the 3x3 LED-grid are explained in sections 3.2.1 Modes of Operation and 3.2.2 Tilt-Orientation respectively.

A total of twelve digital output pins were used to control the LEDs. The I/O pin and the corresponding connected LED are listed in the following table.

Microcontroller I/O Pin	Connected LED
PORTB.0	Grid LED for Sector 1
PORTB.1	Grid LED for Sector 2
PORTB.2	Grid LED for Sector 3
PORTB.3	Grid LED for Sector 4
PORTB.4	Grid LED for Sector 5
PORTB.5	Grid LED for Sector 6
PORTC.0	Grid LED for Sector 7
PORTC.1	Grid LED for Sector 8
PORTC.2	Grid LED for Sector 9
PORTC.3	RGB LED Blue
PORTC.4	RGB LED Green
PORTC.5	RGB LED Red

Table 5-1: Microcontroller I/O Pin and the Corresponding Connected LED

5.2 Receiver-Unit Physical Assembly

The following sections detail the enclosure used for the receiver-unit as well as how the internal electronics were arranged.

5.2.1 Receiver Enclosure

The choice of enclosure had little effect on the outcome of the project, provided it did not hinder performance in any way. Thus the only significant factor which determined the selected enclosure was objective S12 which specified that it should be aesthetically pleasing. There were two additional (although minor) factors which influenced the decision:

1. Enclosure Material – It was deemed preferable that the enclosure be made from a material that could be easily machined, as holes had to be cut to provide access to various connectors as well as mounting positions for the antenna and tri-colour LED. Plastic was the most likely candidate to meet this requirement.
2. Size of Enclosure – Unlike the transmitter, the receiver-unit did not have to be constructed to fit within a small enclosure. Thus an enclosure size large enough to work with such that all components remained easily accessible would ease the construction of the prototype.

An aluminium enclosure was found which was decidedly aesthetically pleasing and of an appropriate size. The aluminium construction gave the unit a sturdy, well-constructed appearance, which in turn gave the impression of a professionally constructed unit. The only disadvantage was the added difficulty of machining holes and slots, as the aluminium is harder to work with (using basic tools) than other available materials. This was an acceptable cost for an otherwise ideal enclosure.

The following figure shows the finished receiver-unit from two different angles.

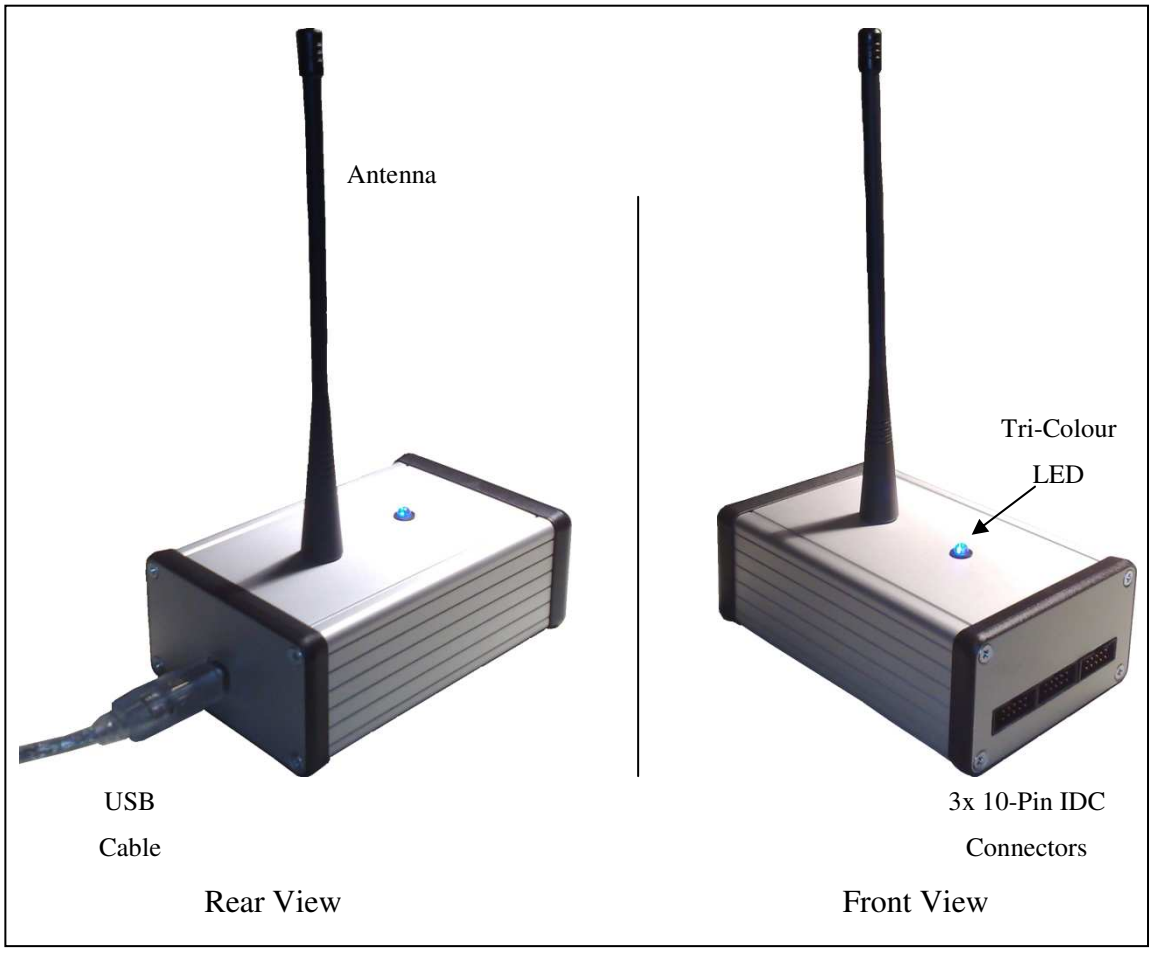


Figure 5-2: Fully Enclosed Receiver-Unit

Dimensions of the unit can be seen in Figure 3-2 in section 3.1.2 Receiver-Unit Layout.

Two holes were drilled in the lid of the unit to serve as mounting points for the tri-colour LED and the external antenna. A rounded square hole was cut from the rear panel to provide access to the USB standard-B connector. Finally, a rectangular slot was cut from the front end to provide access to the three 10-pin IDC connectors. All of this machining was performed using a Dremel rotary tool and a set of needle files.

Initially the use of an aluminium enclosure incurred the possibility of hindering wireless data reception. This fact (plus the desire for increased wireless range – objective S8) led to the decision to incorporate the externally mounted antenna.

5.2.2 3x3 LED-Grid Enclosure

Although technically part of the receiver-unit, it was decided that the 3x3 LED-grid should be incorporated as an externally connectable module. This way the user would have the ability to mount it in a location separate to the main receiver-unit and the option to completely remove the grid.

The grid was mounted within a small, plastic enclosure. The connection to the receiver-unit is made via a 10-core ribbon cable and 10-pin IDC connectors. The dimensions of the unit are provided in Figure 3-3 under section 3.1.2 Receiver-Unit Layout. The following figure shows front and rear views of the 3x3 LED-grid enclosure with a connected ribbon cable.

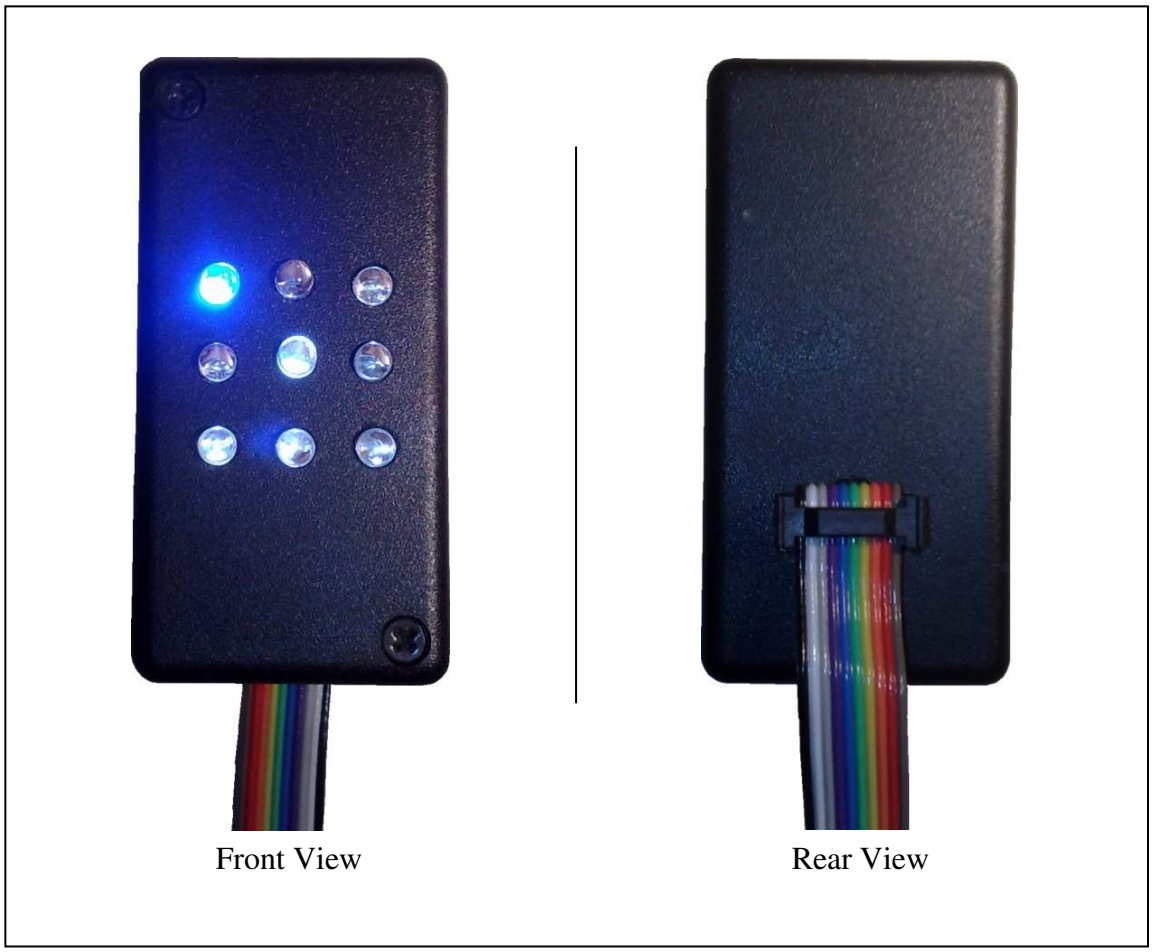


Figure 5-3: Fully Enclosed 3x3 LED-Grid Unit

5.2.3 Layout of Receiver Components and Circuit Board

A section of stripboard was cut to size for the enclosure and used to mount all the electronics except for the tri-colour LED and the antenna. Arranging the components on the receiver-unit's circuit board was straightforward thanks to the relatively large size of the board.

The following diagram details the final mounting positions of the major components on the receiver-unit's circuit board.

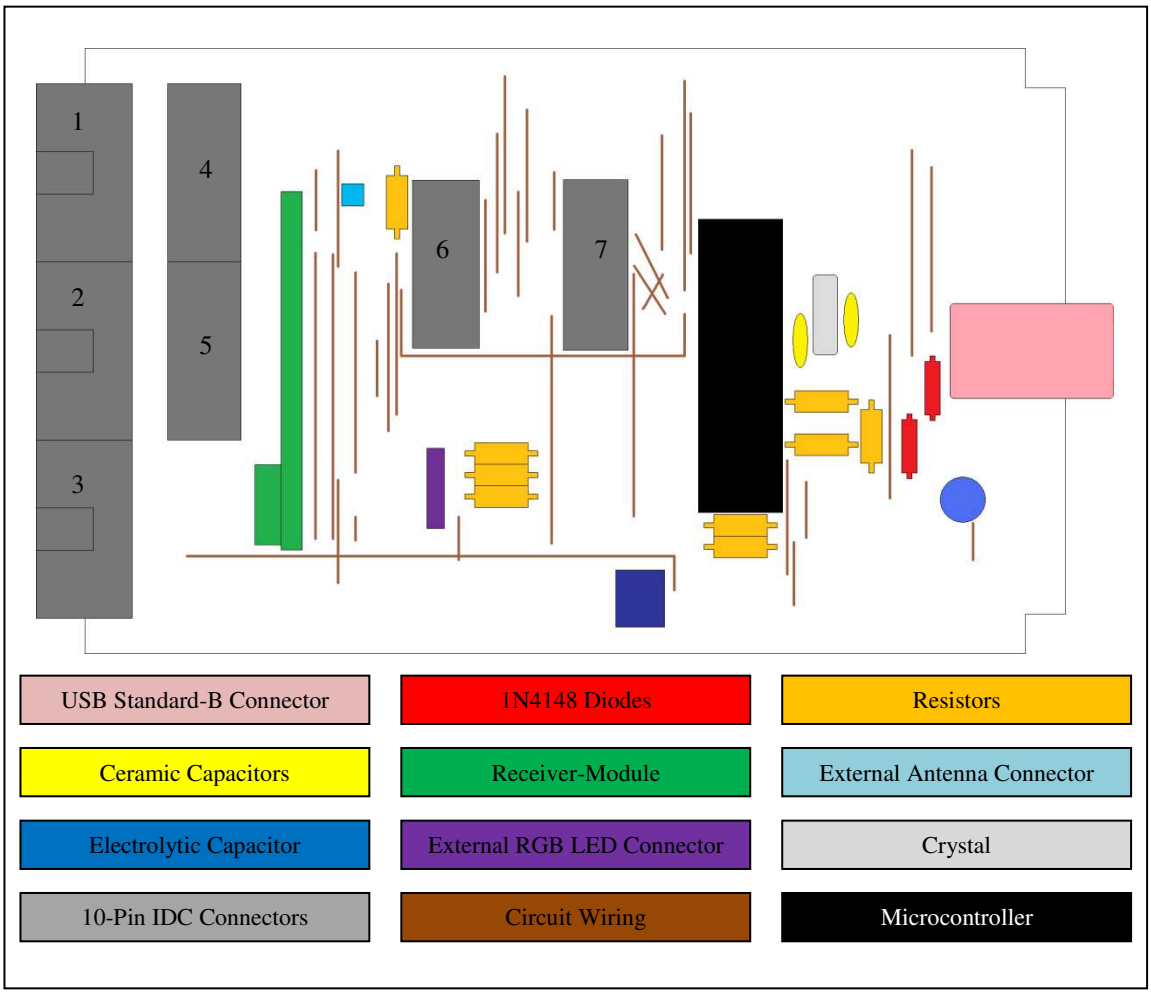


Figure 5-4: Receiver-Unit Circuit Board Component Layout

The next figure is a photograph of the fully populated circuit board.

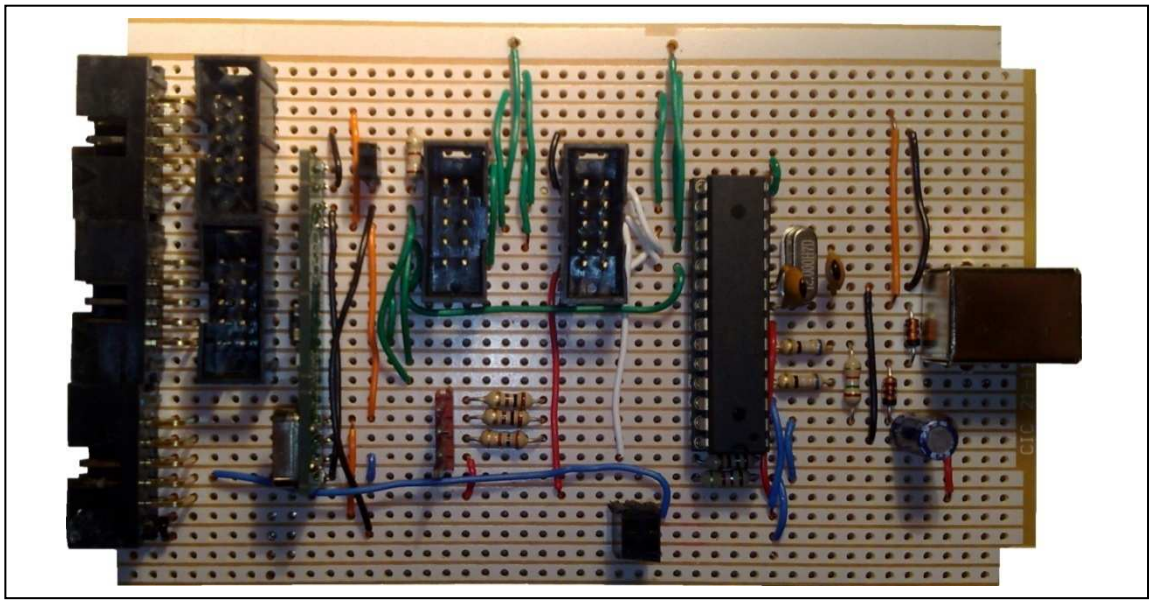


Figure 5-5: Receiver-Unit Circuit Board

Note the multiple, 10-pin, IDC connectors, each of which is numbered 1 through 7 in Figure 5-4. Early development stages of the circuit board construction saw some connectors mounted with the intention of remaining internal (specifically, the ISP port and serial access). Additionally, the LED-grid unit was going to be incorporated into the main receiver-unit. When the decision was made to have the grid unit connected externally, the need arose for an externally accessible connector. At this stage it was decided that the ISP and serial connections would also be made externally accessible. Thus additional connectors were mounted at the edge of the board making them externally accessible after machining a slot into the enclosure. The purpose of each of the numbered 10-pin IDC connectors is detailed in the following list:

1. Externally accessible ISP connection.
2. Externally accessible I/O pin connection for 3x3 LED-grid.
3. Externally accessible serial (USART) TX and RX connections.
4. Intermediate ISP connector.
5. Intermediate I/O pin connection
6. Original I/O pin connection.
7. Original ISP connection.

The external ports are connected to the original ports via short, internal 10-core ribbon cables as visible in the following figure.

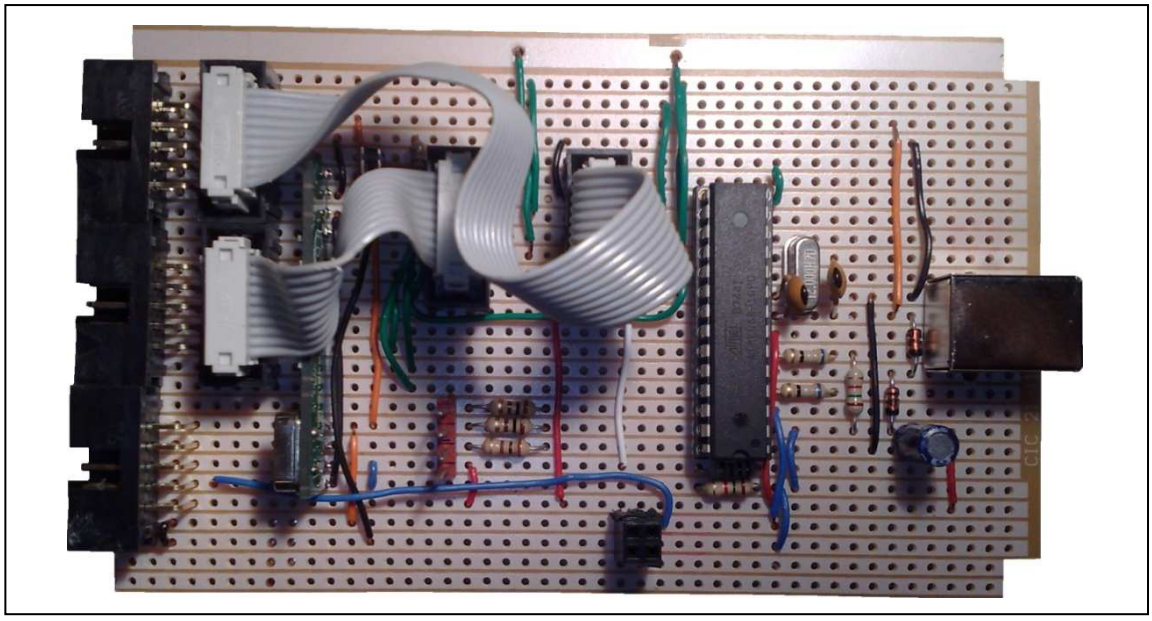


Figure 5-6: Receiver-Unit Circuit Board with Internal Ribbon Cable Connections

Pin identifications for the three externally accessible, 10-pin, IDC connectors can be found in Appendix F.

5.3 Receiver-Unit Firmware

Because the transmitter-unit was designed to perform negligible data processing, the decoding of the input data (button states and accelerometer outputs) has to be performed by the receiver-unit. The input data is received wirelessly in the form of data packets as described in section 3.4 Wireless Data Packets.

In addition to the processing of input data, the receiver-unit also had to contain the firmware USB driver which accounts for a large portion of the overall code.

The code for the receiver-unit is separated into a number of c-code files each of which are located on the accompanying CD (see Appendix G). The files include:

- WiSHABI_RX.c – the main code for the receiver.
- usbdrv.c – the third-party created USB driver code.
- usb.c – the application specific USB related functions.

- serial.c – the required USART related functions.
- mouse_mode.c – the functionality specific to when the receiver is operating in mouse-mode.
- keyboard_mode.c – the functionality specific to when the receiver is operating in keyboard-mode.

The following flowchart provides a general representation of the complete firmware implemented in the receiver-unit's microcontroller.

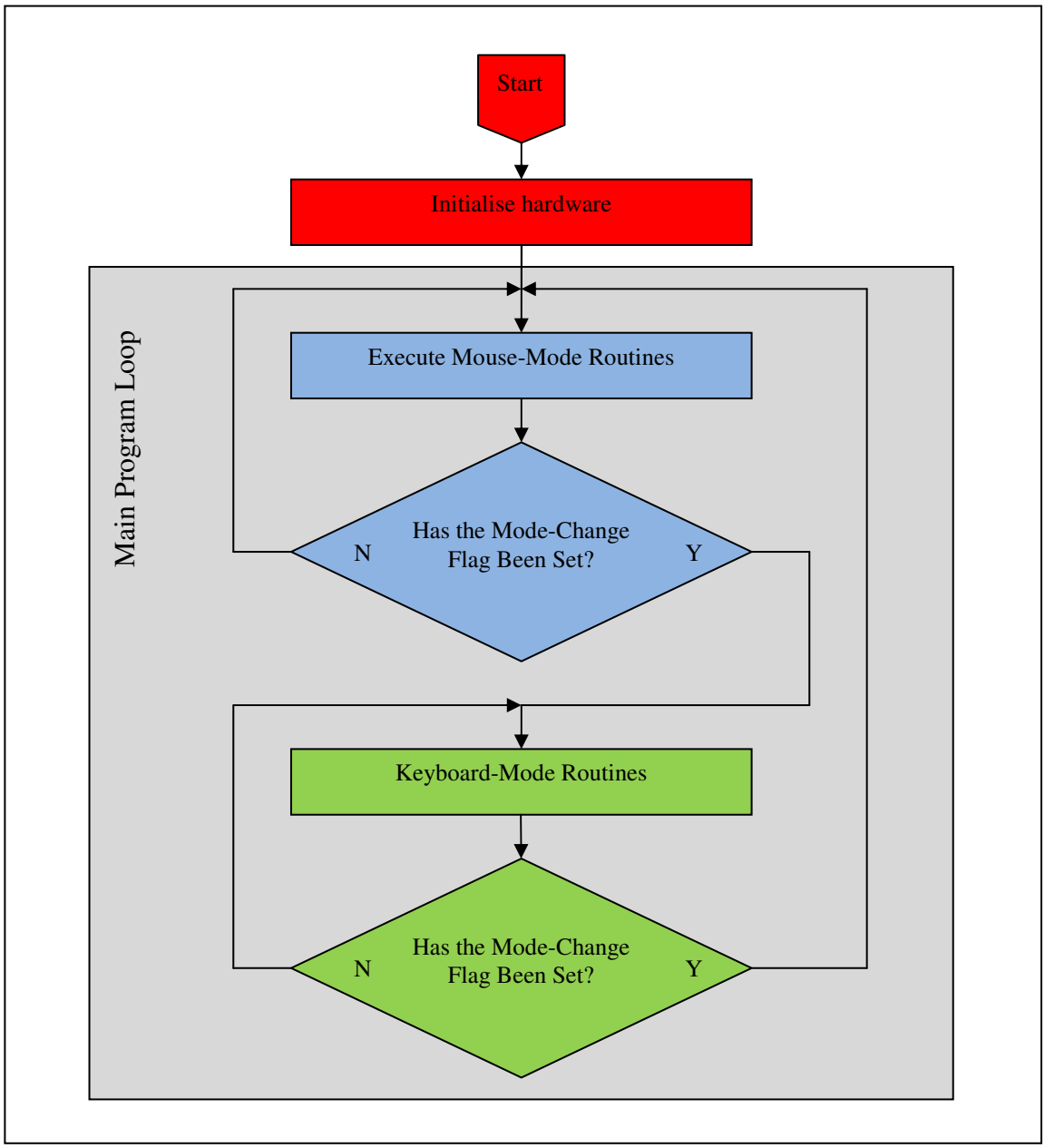


Figure 5-7: Receiver-Firmware Flowchart Representation

Note: The colours in the above diagram (red, green and blue) represent the colour of the RGB LED during the execution of the corresponding code.

Detailed information regarding both mouse and keyboard-modes will be provided in subsequent sections (5.3.6 and 5.3.7 respectively).

5.3.1 Serial Interface

The receiver microcontroller's USART functionality was utilised primarily for the purpose of serial data reception. A serial "driver" was written containing the minimum functions required to receive a byte of data. The driver included a function named USART_Receive() which waits for a serial byte to be received on the RX pin of the microcontroller and then returns the value of said byte. The RX pin is connected directly to the wireless receiver-module, thus enabling wireless, serial data reception.

Whenever the microcontroller receives a data byte, the USART-receive interrupt is flagged, causing the firmware code execution to be diverted to the USART receive interrupt subroutine. The received byte is checked to ensure that it matches the requirements of the data packet format. If so, subsequent data bytes are checked and provided they all match the data packet format, the globally accessible byte array containing the input data from the transmitter-unit is updated.

The following flowchart details the input data authentication and collection procedure performed by the USART-receive interrupt subroutine:

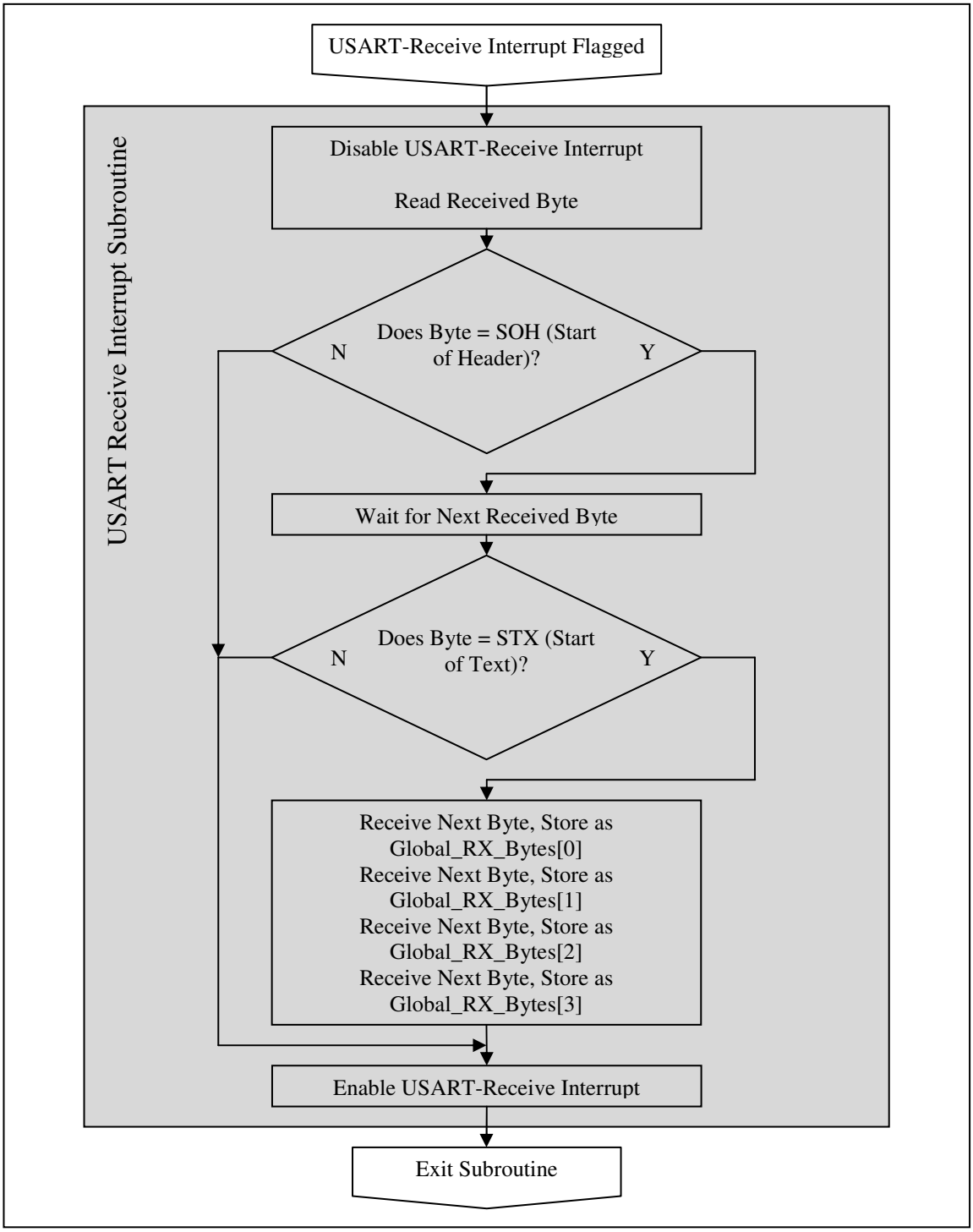


Figure 5-8: USART Receive Interrupt Subroutine Flowchart Representation

This form of data-packet authentication proved to effectively prevent unintended data (background RF noise) from being received and processed by the receiver-unit. By making data reception interrupt-driven (as opposed to periodically polling for received data bytes) the receiver-unit will always have the most up-to-date input data, and the code execution will continue un-interrupted in the event that the transmitter-unit is out of range or not powered. The data packet format is defined in section 3.4 Wireless Data Packets. The USART receive subroutine is called ISR(USART_RXC_vect) and is part of the main code file for the receiver-unit (WiSHABI_RX.c).

5.3.2 USB Interface Firmware Driver

The USB protocol is relatively complex to implement in firmware. Some background information on USB devices (specifically HIDs) can be read in section 2.3 USB Protocol – HID Compliance and further details pertaining to the USB implementation specific to this project is located in section 3.5 USB Implementation. Additionally, further details involving the utilised USB driver can be found in Appendix A.

The USB driver required the inclusion of a number of custom, application-specific functions and declarations. The most important of which include:

- Defining the HID Report Descriptor Variable (section 3.6 HID Report Descriptor).
- Defining the HID Report Variables (one for keyboard reports and one for mouse reports).
- usbFunctionSetup() Function which includes code for handling the reception of data from the PC.
- usbFunctionWrite() Function which includes code for processing data from the PC.
- usbReset() Function which handles a microcontroller initiated USB hardware reset.
- hid_clear() Function which sends all zeroes as keyboard and mouse control parameters.
- send_packets() which sends specified values as keyboard and mouse control parameters.

These functions and variables are accessed from the main loop as well as the mouse-mode and keyboard-mode loops (detailed later in sections 5.3.6 and 5.3.7 respectively).

The c-code listing for the application-specific USB functions exists as a separate file called usb.c.

Provided with the USB driver is a configuration file usbconfig.h. This header file is simply a list of customizable definitions which are to be altered as per the specific requirements of the application. For example, if a different AVR microcontroller, clock speed or I/O pins were used or if the setup was for a class other than HID. For this project, most of the definitions were left as the default values with the exception of the following two:

1. USB Device-Name was changed to "WiSHABI" which is a partial acronym for the title of this project (Wireless Single Handed Accelerometer Based Interface) and made a suitable device name.
2. USB Vendor-Name was changed to the email address of the device creator (jadonclews@gmail.com) as per the recommendation of the creators of the USB driver (see Appendix A).

Both the USB Device-Name and Vendor-Name are required by the PC to provide details to the user about the connected peripheral.

5.3.3 LED Control

A function was created to specifically set all of the LEDs according to the value of a 16-bit input parameter. The twelve least significant bits of the input value represent the desired state of the twelve LEDs (1=on, 0=off). The following table details the purpose of each bit within the input parameter and the microcontroller pin it controls.

Input Byte	Controlled LED	Connected Microcontroller Pin
b15	Unused – The values here are always equal to zero and have no consequence.	Not Connected
b14		
b13		
b12		
b11	RGB LED 3 (Red)	Port C, Pin 5
b10	RGB LED 2 (Green)	Port C, Pin 4
b09	RGB LED 1 (Blue)	Port C, Pin 3
b08	Grid LED 9 (Blue)	Port C, Pin 2
b07	Grid LED 8 (Blue)	Port C, Pin 1
b06	Grid LED 7 (Blue)	Port C, Pin 0
b05	Grid LED 6 (Blue)	Port B, Pin 5
b04	Grid LED 5 (White)	Port B, Pin 4
b03	Grid LED 4 (Blue)	Port B, Pin 3
b02	Grid LED 3 (Blue)	Port B, Pin 2
b01	Grid LED 2 (Blue)	Port B, Pin 1
b00	Grid LED 1 (Blue)	Port B, Pin 0

Table 5-2: LED Control Byte and Connected Hardware Pins

The LEDs in the 3x3 grid all share a common cathode (ground) which means a logic “1” must be output at the I/O pin to switch on the corresponding LED. Conversely, the three colours of the RGB LED share a common anode which means each colour is activated with a logical “0”. Thus the LED control function has to invert the input bits intended to control RGB LED 1, 2 and 3.

The final implementation of the LED control function did not require the assignment of values for the RGB LED so these bits (b09 to b11) were ignored and the colour of the RGB LED was determined by a second input variable which represented the current operating mode. Thus the first input variable controlled the state of the grid LEDs whilst the second determined the state of the RGB LED.

The LED control function was named Set_LEDs(). It is called by the function Poll_And_Update_LEDs() which determines the current tilt-orientation (using function Poll_Sector() to be described in section 5.3.4 Determining Tilt-Sector) and operating mode, then uses Set_LEDs() to control the LED states accordingly. Both Set_LEDs() and Poll_And_Update_LEDs() form part of the main receiver code (WiSHABI_RX.c).

5.3.4 Determining Tilt-Sector

As explained previously (in section 3.2.2 Tilt-Orientation) the transmitter's tilt-orientation is divided into nine sectors. The calculation of the current sector is determined by the values within the transmitted data.

The current tilt-sector (a value from 1 to 9) is returned by the function Poll_Sector() which reads and processes the values within the Global_RX_Bytes[] array which represent the outputs from the accelerometer. The function first determines the X-position of the tilt-orientation which narrows the current tilt-sector possibilities down to one of three vertically-aligned sectors (i.e. sectors 1,4,7, sectors 2,5,8 or sectors 3,6,9). Next the function determines from the accelerometer values the current Y-position of the tilt-orientation. Thus the tilt-sector is narrowed down to a single possibility which is returned by the function.

Poll_Sector() is called by functions within the main receiver code (WiSHABI_RX.c) as well as the mouse-mode (mouse_mode.c) and keyboard-mode (keyboard_mode.c) files. The function itself is part of the code listed in the file WiSHABI_RX.c.

5.3.5 No-Signal Timeout Detection

As explained in previous section 3.2.1 Modes of Operation, the tri-colour LED uses the colour red to indicate that the receiver-unit is not receiving a signal. This functionality is implemented in both mouse and keyboard-modes.

Every iteration of the relevant mode loop increments a counter named Global_Timeout. If this value reaches a defined limit, another loop is initiated whereby the tri-colour LED is changed to glow red until the Global_Timeout counter is reset. The loop also continuously polls the USB driver so as to avoid losing USB connectivity.

Global_Timeout is reset to zero whenever the USART receive interrupt vector (see section 5.3.1 Serial Interface) receives a valid data packet. Thus the counter will only increment beyond one if the mode-loop completes multiple iterations without the unit receiving a wireless data packet.

This functionality is implemented in both the mouse_mode.c and keyboard_mode.c routines which are both included on the attached CD (see Appendix G).

5.3.6 Mouse-Mode

The purpose of mouse-mode and the device's intended usage is explained in section 3.2.3 Operating in Mouse-Mode. From a firmware perspective, the microcontroller is required to continuously check the input data value representing the transmitter button states. If any of the buttons are pressed, the relevant functions are initiated.

The flowchart on the following page provides a visual representation of the functional execution whilst operating in mouse-mode:

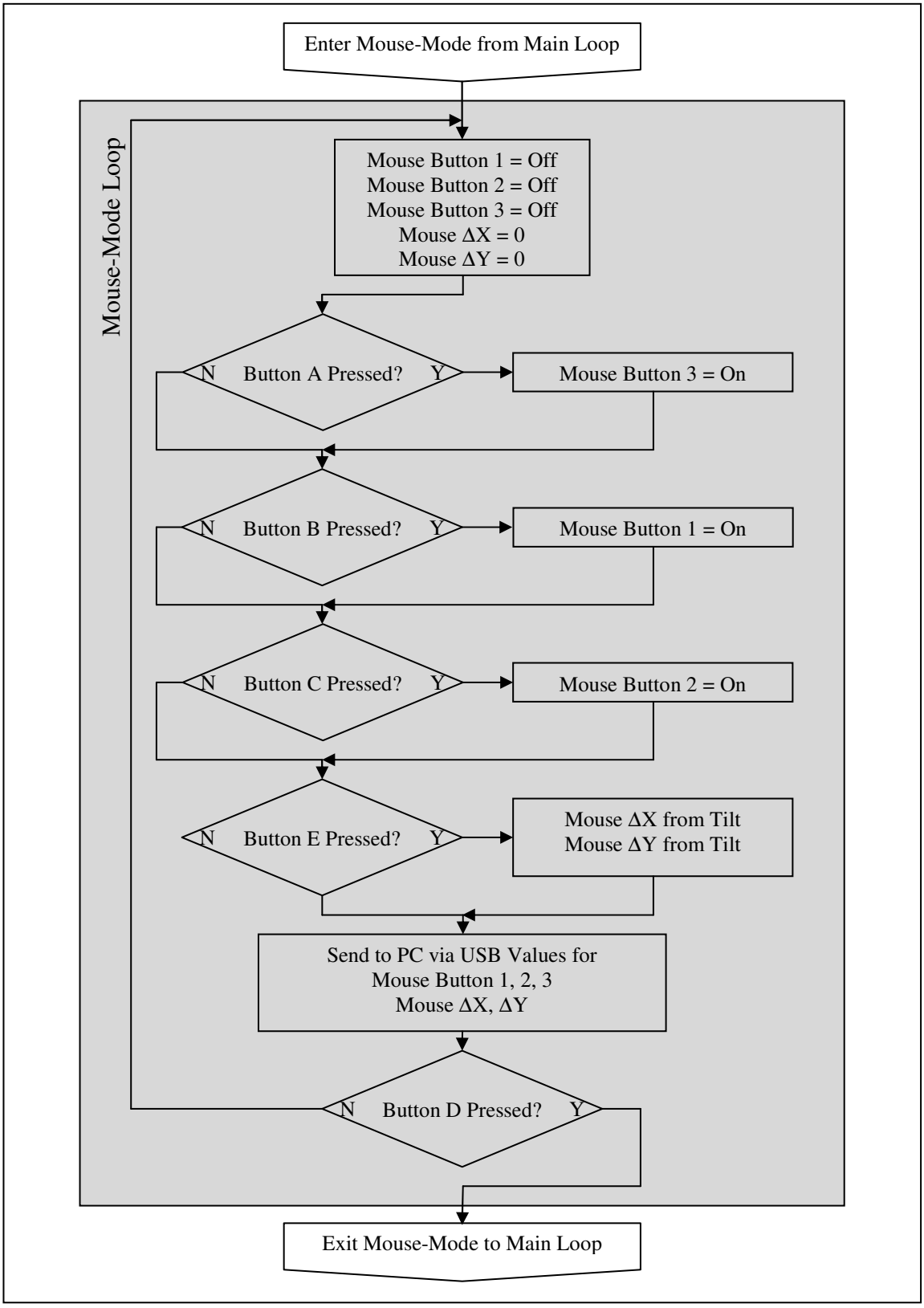


Figure 5-9: Main Mouse-Mode Loop Flowchart Representation

The ΔX and ΔY values represent the number of pixels by which the mouse-cursor should move on-screen along the X and Y axis. The direction of this movement is determined by the current tilt-orientation sector of the transmitter-unit. The sector value (1 to 9) is determined by the function Poll_Sector() as explained previously in section 5.3.4 Determining Tilt-Sector. The magnitude by which the cursor will move in the relevant direction varies. When button-E is first pressed, the magnitude is very small (a single pixel) allowing the user to "fine-tune" the precise cursor position. After a number of iterations and provided button-E is still depressed, the magnitude will increase to a moderate amount (5 pixels) thus increasing the speed by which the cursor is relocated. The magnitude will increase again to a maximum displacement value (10 pixels) provided button-E is held for enough iterations of the mouse-mode loop. The variances in displacement magnitude of the cursor position allow for quick relocation across large on-screen distances without the user losing the ability to control the precise position of the cursor.

The code relevant to mouse-mode is located in the file mouse_mode.c which is included on the CD (see Appendix G).

5.3.7 Keyboard-Mode

The purpose of keyboard-mode and the device's intended usage is explained in section 3.2.4 Operating in Keyboard-Mode. From a firmware perspective, the microcontroller is required to continuously check the input data value representing the transmitter button states. If any of the buttons are pressed, the relevant functions are initiated.

The flowchart on the following page provides a visual representation of the functional execution whilst operating in keyboard-mode:

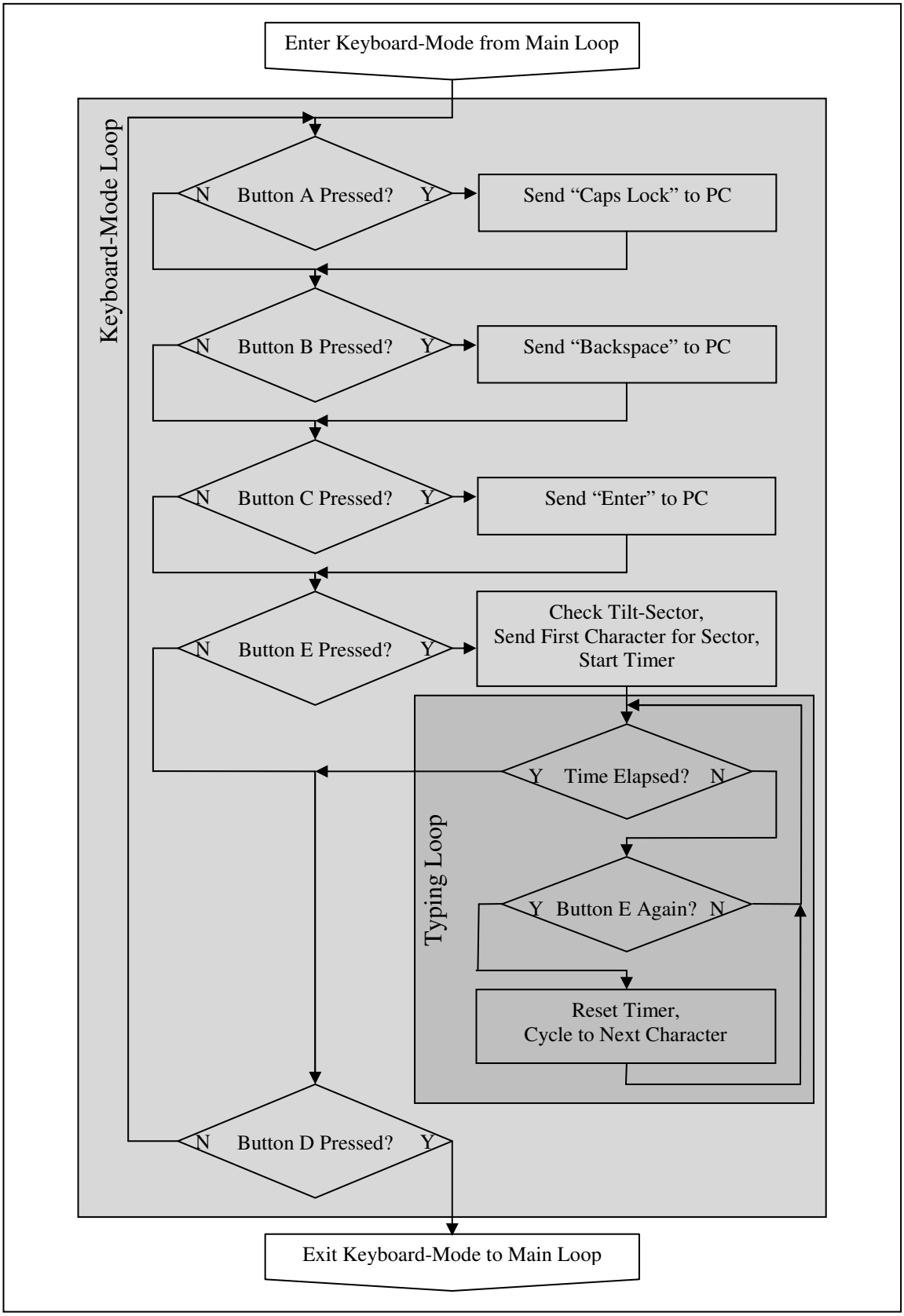


Figure 5-10: Main Keyboard-Mode Loop Flowchart Representation

Pressing button-E redirects the code execution into the typing loop which cycles the character typed on-screen through a sub-set of the total alphanumerical character-set. The sub-set depends on the tilt-orientation of the device when button-E is first pressed. Figure 3-11 in section 3.2.4 Operating in Keyboard-Mode defines the character sub-set that corresponds to each tilt-sector.

The code relevant to keyboard-mode is located in the file keyboard_mode.c which can be found on the accompanying CD (see Appendix G).

6 Conclusions

Overall the project was deemed a success. All of the primary and secondary objectives (defined in section 1.1 Project Objectives) were satisfactorily achieved.

The following sections cover the main problems which arose during the progression of the project, as well as the limitations inherent in the design and some improvements that could be made to the system.

6.1 Unanticipated Problems

There were two problems which arose that caused significant set-backs in the project progress. Although overcome, had these problems been anticipated, a different system design may have been utilised. The problems are described in the following two sections.

6.1.1 Assembling Transmitter Components within a Small Enclosure

A small enclosure was necessary for the transmitter-unit to be hand-held (as per objective P2). A relatively large amount of time was required to determine an ideal location and orientation of each component so as to ensure they all fit within the chosen enclosure. Two design changes should have been implemented to prevent this problem and likely reduce the required construction time:

1. Smaller Power/Battery Module – Together, the power-module and batteries required by far the most amount of room within the transmitter-unit enclosure. A different power-module with a smaller footprint should have been utilised. Additionally, a more suitable battery (such as the kind used in modern mobile telephones) should replace the pair of AA cells utilised in the final design.
2. Custom Printed Circuit Board/s – Designing a custom circuit board or boards on which the microcontroller, transmitter-module, accelerometer and miscellaneous components could be mounted would have enabled the components to be fitted closer together. The stripboard used, although easy to work with, is not capable of minimising the surface area required by inter-connecting conductive tracks like a custom PCB design could.

6.1.2 Combining Keyboard and Mouse Functionality into a Single Device

The company behind the USB driver utilised for this project also provide access to a number of example projects. Some of the available projects utilise the driver to emulate keyboard keystrokes and others emulate mouse control. There was however very little information or examples regarding the implementation of a device which could combine both mouse and keyboard control. Doing so proved to be a time consuming process which required a deeper understanding of how the USB driver operated. Before successfully combining mouse and keyboard functionality into a single device, two alternative solutions were considered but rejected:

1. Implement a Second Microcontroller – Programming one microcontroller with the USB driver to act as a mouse-like device and programming the second to act as a keyboard-like device would have been simpler. The two sub-systems could communicate via one of the microcontroller's inherent serial protocols. Both microcontrollers would then be connected to separate USB ports on the PC and the system would actually be recognised as two separate devices. This configuration is obviously less than ideal and was thus rejected.
2. Switch to a Microcontroller with Hardware USB – The possibility of utilising a microcontroller that incorporated a hardware implementation of the USB protocol (as described in section 2.3.1 USB Protocol) was revisited as this would reduce the firmware requirements and complexity. This option was again rejected due to the difficulty in obtaining such a microcontroller at reasonable price.

Ultimately the problem was overcome and although it could have been avoided, perseverance paid off as the deeper understanding of the USB driver proved to be advantageous in subsequent stages of the project. However, the effect on the project progress schedule could have been reduced by developing a deeper understanding of the USB driver prior to attempting to implement it.

6.2 Project Limitations

There are a few notable limitations inherent in the complete system. Although not detrimental to the performance, overcoming these limitations would be advantageous:

6.2.1 Typing Speed

The rate at which text can be entered increases with practise much like a regular keyboard. However, compared to a regular keyboard, the maximum possible typing speed is very low, making the system suitable only for typing relatively small stings of text.

As stated previously (section 3.2.4 Operating in Keyboard-Mode), the tilt-sector designation of character sets was designed to be similar to the layout of a mobile-phone keypad. This was intended to minimise the time required to learn how to type with the device (objective P4) as the keypad format is already familiar to most people.

It is likely that an altered typing method could allow faster text entry. A different combination of tilt angles and button presses could allow faster character selection, however it is likely that such a design would require a significantly larger learning period for the user.

6.2.2 Battery Life

Although a battery life of 2.5 hours was deemed adequate to meet the relevant objective (S3) many wireless mouse and keyboard devices available today are capable of running for a significantly longer duration (typically days or even months). A number of possible alterations to the design would increase the battery life of the transmitter-unit:

1. On/Off Switch – By simply implementing a power switch in-line with the supply source, a user could completely deactivate the unit when it is not being used. This option was actually considered in early designs but was rejected as it would slightly decrease the ease of use of the system by the user and a switch would also negatively affect the transmitter-unit's aesthetic.
2. Incorporate Microcontroller Power-Saving Functionality – The ATmega8 microcontroller used in the transmitter-unit has five different sleep-modes which can

be incorporated into the firmware programming. None of these modes were included in the final design. Extra programming could detect when the transmitter is not in use and activate a sleep-mode to further reduce power consumption.

3. Modified Circuit Design – The receiver system only sends USB data when a button is pressed on the transmitter. A modified circuit design could cause the transmitter-unit to be powered only when one of the buttons is pressed. In addition to setting the state of the connected microcontroller, a button-press could be designed to actually provide power to the device, thus power is only drawn when one or more buttons are actually being manipulated. This design is similar to many car-alarm remote controls where the unit is only powered when one of the buttons (arm/disarm, boot-open etc.) is pressed. Implementing this option would be the most ideal for reducing power-drain, however testing would be required to ensure the response time is not significantly reduced due to the required initialisation time of the different hardware devices (USART, accelerometer etc) whenever a button is pressed.

6.2.3 RF Noise

The transmitter-unit constantly outputs a wireless data stream. This RF signal could possibly cause interference with other devices utilising a similar RF frequency. The options listed in the previous section (6.2.2 Battery Life) which are intended to improve the battery life of the unit would also have the added advantage of preventing the unit from transmitting except when it is actually being used, thus reducing the RF output.

A further reduction in RF "noise" could be achieved by modifying the transmitter-unit's firmware. As stated, the unit currently outputs a continuous stream of data packets. The firmware could be modified to simply output a data packet "pulse" whenever the state of a button changes. The exception to this would be when button D is held during mouse-mode, as this requires a constant stream of accelerometer output data in order to continuously control the motion of the mouse cursor.

6.2.4 Cursor Positioning

As explained in section 3.2.3 Operating in Mouse-Mode, the direction of movement of the on-screen cursor corresponds to the transmitter-unit's tilt-sector. With only nine possible sectors, cursor motion is limited to eight possible directions, each 45 degrees apart.

Additional data-processing code implemented in the receiver-unit's function Poll_Sector() (see section 5.3.4 Determining Tilt-Sector) could provide a larger range of detected tilt-sectors. For example, tilt-sectors represented in a 5x5 grid could provide cursor speed and angle control as indicated by the following diagram:

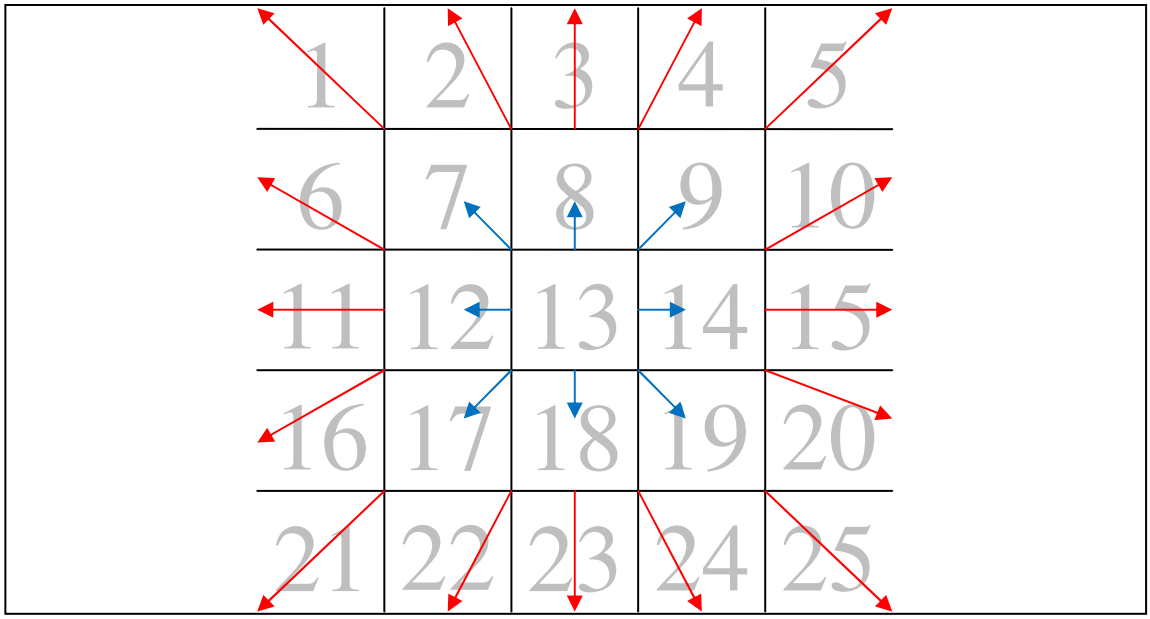


Figure 6-1: Improved Tilt-Sector Designations

The blue arrows represent fine cursor control (low speed) while more extreme tilting of the transmitter-unit would result in faster motion at angles indicated by the red arrows.

As mentioned, implementing this extra functionality would require additional code for the receiver-unit's firmware. Additionally, it may also require alterations to the transmitter-unit's firmware in order to obtain higher resolution output values from the accelerometer (see section 4.3.2 ADC (Analogue to Digital Converter)). This may be necessary to accurately define output accelerometer ranges for each sector.

6.3 Possible Improvements

A number of improvements could be made to the system in order to provide added functionality. Some proposed system upgrades follow.

6.3.1 Software Support

Although one of the objectives (S7) specified that no additional software should be required for the system to function, creating optional software to work with the device could provide the user with additional functionality. Some examples of software capability include:

1. On-screen display of the tilt-orientation sector, thus eliminating the need for the external 3x3 LED-grid unit.
2. Customisable button assignments (e.g. switch left and right mouse click buttons).
3. The ability to re-calibrate the device to match an individual's preferences (i.e. the motion range which defines tilt-sectors).
4. The ability to store different calibration settings and button assignments for multiple users or environments (e.g. sitting at a desk or standing in a lecture theatre).

6.3.2 Receiver-Unit Miniaturisation

The receiver-unit is relatively bulky which is ideal for the prototyping phase. However a smaller footprint is always preferable with regards to wireless receiver-units as a smaller device requires less space on the user's desk. By designing and implementing a custom PCB layout combined with smaller (possibly surface-mount) components, the size of the receiver-unit could be greatly reduced.

6.3.3 Caps Lock Indicator

The device gives the user the ability to toggle the PC's caps lock status. An additional LED configured to indicate the current status of the caps lock would be ideal to prevent accidentally typing in the wrong case.

6.3.4 Docking Station

The final design includes a transmitter-unit which can be recharged by attaching it to a mains-connected DC power supply. A more ideal solution would involve a docking station which could rest permanently on a desk. Whenever not in use, the device could be simply placed onto the docking station which will facilitate charging of the transmitter's batteries.

Additionally the receiver-unit's internal electronics (including connectors and indicator LEDs) could be installed within the docking station to reduce the number of system units. This would provide the designer two power options whereby:

1. The docking station is connected to the PC via a USB cable for data communications only. Power for both the receiver electronics and recharging the transmitter is provided by a mains-connected power supply. This option would be simpler to implement and provide the system with more than enough power.
2. The docking station is connected to the PC via a USB cable for both data communications as well as power. The 5 volts available from a USB port could power the receiver-unit when the system is in use and then be redirected to recharge the transmitter when it is in the docking station. This option would be more ideal but significantly more complex to implement.

7 References

1. Dean Camera, 2007, 'Newbie's Guide to AVR Timers', in *AVR Freaks*, accessed 25 March 2008, from <http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=50106>
2. Dean Camera, 2007, 'Using the USART – Serial Communications', in *AVR Freaks*, accessed 25 March 2008, from <http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=45341>
3. Dean Camera, 2007, 'Interrupt driven USARTs' in *AVR Freaks*, accessed 25 March 2008, from <http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=48188>
4. Ken Worster, 2007, 'Newbie's Guide to the AVR ADC', in *AVR Freaks*, accessed 25 March 2008, from <http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=56429>
5. Anonymous, 2007, 'Accelerometers', in *RotoView*, accessed 29 July 2008, from <http://www.rotoview.com/accelerometer.htm>
6. Anonymous, 2008, 'Accelerometers', in *Hitachi Metals America, Ltd.*, accessed 29 July 2008, from <http://www.hitachimetals.com/product/sensors/accelerometer/h30CD_accelerometer.cfm>
7. Anonymous, 2001, 'HID Usage Tables1.12', in *USB Implementers' Forum*, accessed 14 March 2008, from <http://www.usb.org/developers/hidpage#HID_Usage>
8. Anonymous, 2001, 'Device Class Definition for HID 1.11', in *USB Implementers' Forum*, accessed 14 March 2008, from <http://www.usb.org/developers/hidpage#HID_Usage>
9. Anonymous, 2007, 'HIDKeys – An Example USB HID', in *Objective Development*, accessed 29 March 2008, from <http://www.obdev.at/products/avrusb/hidkeys.html>
10. Anonymous, 2007, 'EasyLogger', in *Objective Development*, accessed 29 March 2008, from < http://www.obdev.at/products/avrusb/easylogger.html>

11. Mikkel Holm Olsen, 2007, 'C64 USB Keyboard', in *Symlink*, accessed 29 March 2008, from <http://symlink.dk/projects/c64key/>
12. Mikkel Holm Olsen, 2007, 'SpiffChorder', in *Symlink*, accessed 29 March 2008, from < http://symlink.dk/projects/spiffchorder/>
13. Alex Badea, 2007, 'InfraHID' in *Vamposdecampos*, accessed 29 March 2008, from < http://vamposdecampos.googlepages.com/infrahid.html>
14. Jan Axelson 1999, *USB Complete*, Lakeview Research, Madison, WI 53704.
15. Deitel, Paul & Deitel, Harvey 2007, *C How to Program Fifth Edition*, Pearson Prentice Hall, Upper Saddle River, NJ.
16. Jeri R. Hanly & Elliot B. Koffman, *C Program Design for Engineers Second Edition*, Addison Wesley, USA.

Appendix A – Objective Development USB Driver Information

The USB driver utilised for this project is called AVR-USB and was developed by a company named Objective Development. Their website can be found at http://www.obdev.at/index.html.

Extracted from the product description page at the Objective Development website:

"AVR-USB is a software-only implementation of a low-speed USB device for Atmel's AVR microcontrollers"

The driver is freely available for use under the terms of the *GNU General Public License Version 2 (GPL)*. In addition to the terms and conditions of the GPL, Objective Development also strongly recommends that users of the driver do the following (also extracted from their website):

1. Publish your entire project on a website and drop us a note with the URL. Use the Feedback Form for your submission. If you don't have a web site to publish your project, we can host it in our Wiki.
2. Adhere to minimum publication standards. Please include at least:
    - a circuit diagram in PDF, PNG or GIF format
    - full source code for the host software
    - a Readme.txt file in ASCII format which describes the purpose of the project and what can be found in which directories and which files
    - a reference to our website at http://www.obdev.at/avrusb/
3. If you improve the driver firmware itself, please give us a free license to your modifications for our commercial license offerings.

Additionally, the developers require that the device and vendor names identified by the connected USB device adhere to the following rules:

1. The USB device MUST provide a textual representation of the manufacturer and product identification. The manufacturer identification MUST be available at least in USB language 0x0409 (English/US).

2. The textual manufacturer identification MUST contain either an Internet domain name (e.g. "mycompany.com") registered and owned by you, or an e-mail address under your control (e.g. "myname@gmx.net"). You can embed the domain name or e-mail address in any string you like, e.g. "Objective Development http://www.obdev.at/avrusb/".

Text documents (License.txt and USBID-License.txt) containing the detailed licensing information are stored with the driver firmware and are included on the CD accompanying this document (see Appendix G).

Appendix B – Transmitter-Unit Circuit Diagram and Component List

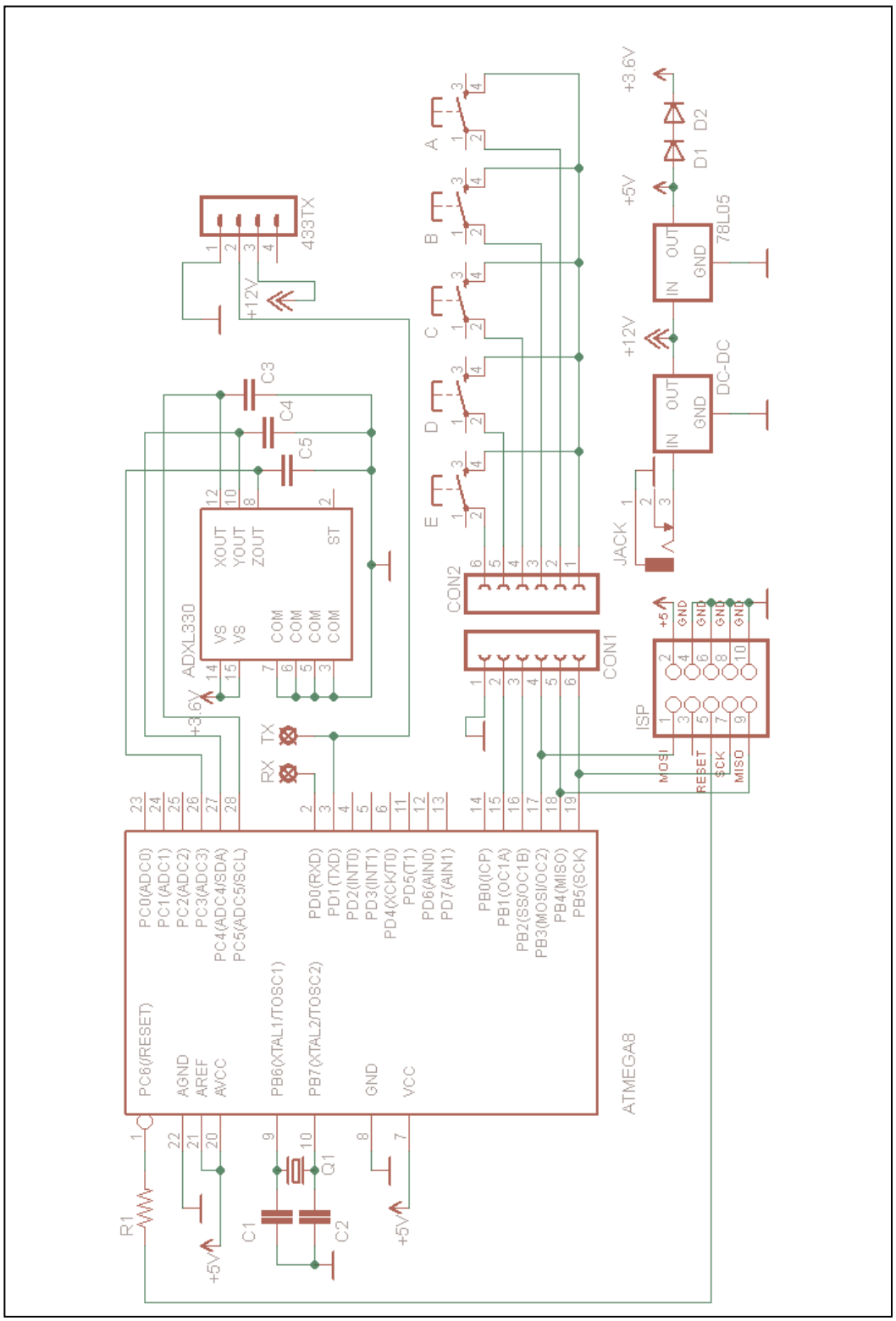


Figure B-1: Transmitter-Unit Circuit Diagram

Label	Component
433TX	433.92 MHz Transmitter-Module
78L05	78l05, Low Power, +5V Voltage Regulator
ADXL330	ADXL330 3-Axis, ±3g, Accelerometer
ATMEGA8	AVR ATmega8 Microcontroller
C1	27pF Ceramic Capacitor
C2	27pF Ceramic Capacitor
C3	0.1µF SMD Capacitor
C4	0.1µF SMD Capacitor
C5	0.1µF SMD Capacitor
CON1	6-Pin, In-Line Female Connector
CON2	6-Pin, In-Line Male Connector
D1	1N4148 Diode
D2	1N4148 Diode
DC-DC	3-9V DC-DC Converter Module
ISP	10-Pin Male IDC Socket
JACK	2.1mm Male DC Power Socket
Q1	12.0MHz Crystal
R1	100Ω Resistor
A	12mm, PCB-Mount, Tactile Push-Button
B	12mm, PCB-Mount, Tactile Push-Button
C	12mm, PCB-Mount, Tactile Push-Button
D	12mm, PCB-Mount, Tactile Push-Button
E	12mm, PCB-Mount, Tactile Push-Button

Table B-1: Transmitter-Unit Component List

Appendix C – Receiver-Unit Circuit Diagram and Component List

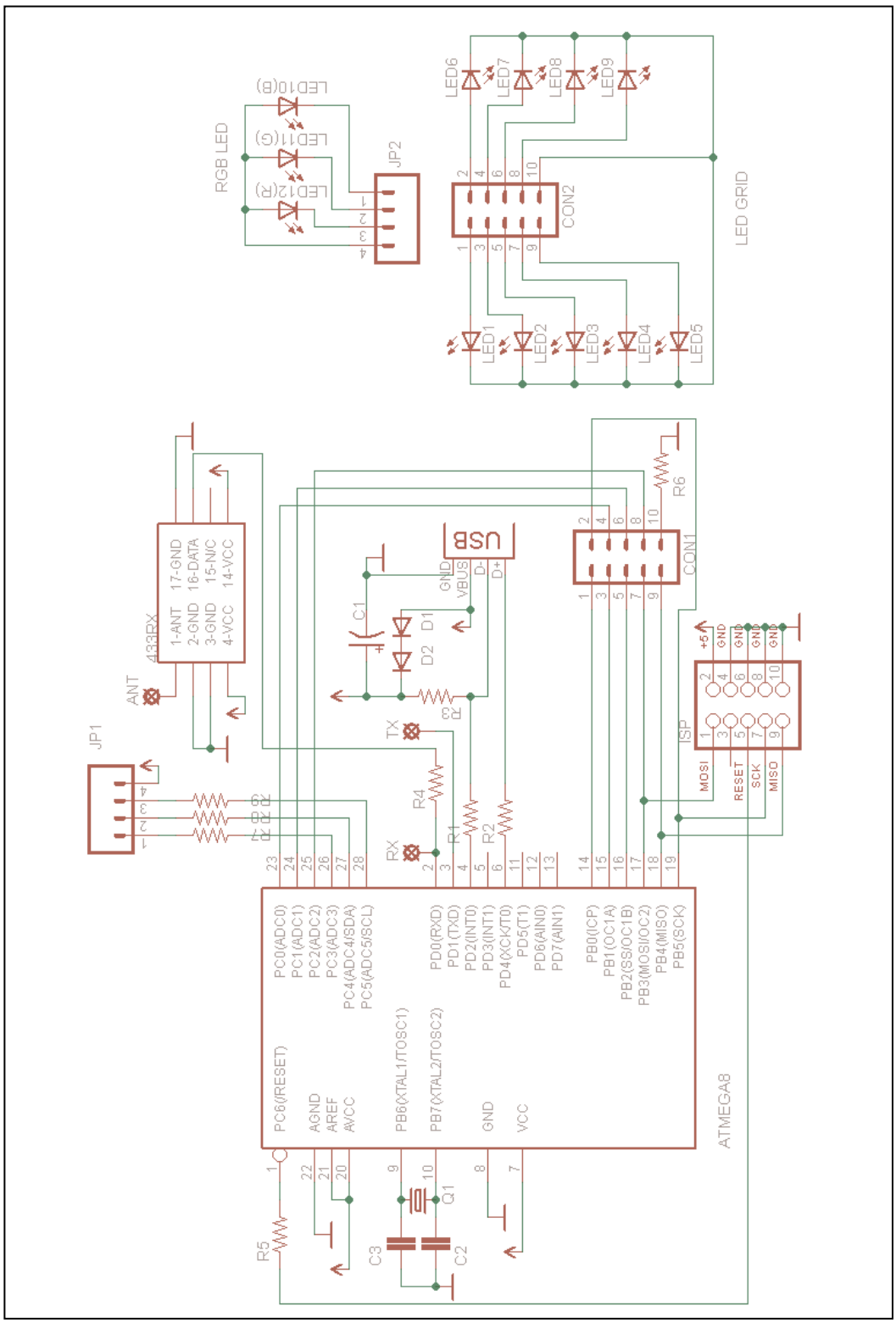


Figure C-1: Receiver-Unit Circuit Diagram

Label	Component
433RX	433.92 MHz Receiver-Module
ATMEGA8	AVR ATmega8 Microcontroller
C1	10μF Electrolytic Capacitor
C2	27pF Ceramic Capacitor
C3	27pF Ceramic Capacitor
CON1	10-Pin Male IDC Socket
CON2	10-Pin Male IDC Socket
D1	1N4148 Diode
D2	1N4148 Diode
ISP	10-Pin Male IDC Socket
JP1	4-Pin, In-Line Male Connector
JP2	4-Pin, In-Line Female Connector
LED10(B)	5mm RGB LED Blue Connection
LED11(G)	5mm RGB LED Green Connection
LED12(R)	5mm RGB LED Red Connection
LED1-9	5mm Blue LEDs
Q1	12.0MHz Crystal
R1	68Ω Resistor
R2	68Ω Resistor
R3	1.5kΩ Resistor
R4	1kΩ Resistor
R5	100Ω Resistor
R6	180Ω Resistor
R7	100Ω Resistor
R8	100Ω Resistor
R9	100Ω Resistor
USB	Standard-B Female USB Socket

Table C-1: Receiver-Unit Component List

Appendix D – Main Component Details

Component	Manufacturer	Supplier	Unit Cost	#	Extended Cost
ATmega8 Microcontroller	Atmel	Jaycar	$19.95	2	$39.90
ADXL330 Accelerometer	Analog Devices	Ocean Controls	$45.00	1	$45.00
433.92MHz Transmitter	-	Jaycar	$9.95	1	$9.95
433.92MHz Receiver	-	Jaycar	$9.95	1	$9.95
12mm Tactile Push Button	-	Altronics	$0.90	5	$4.50
5mm LED	Bright LED Electronics Corp	Altronics	$3.80	9	$34.20
5mm RGB LED	Bright LED Electronics Corp	Altronics	$5.20	1	$5.20
160mm Whip Antenna	Centurion	RS-Online	$12.40	1	$12.40
Aluminium Enclosure	Hammond Manufacturing	Altronics	$33.50	1	$33.50
Miscellaneous (Passives, wiring, connectors, etc)	-	Altronics/ Jaycar	~$40.00	1	$40.00
		Total Cost:			$234.60

Table D-1: Main Component Manufacturers, Supplies and Cost

Supplier	Website
Altronics	http://www.altronics.com.au/
Jaycar	http://www.jaycar.com.au/index.asp
Ocean Controls	http://www.oceancontrols.com.au/
RS-Online	http://australia.rs-online.com/web/home.html

Table D-2: Component Supplier Websites

Appendix E – Serial Line Level Conversion Module

Following is data pertaining to the TTL-RS-232 serial line level converter which was created for this project to provide a simple means of debugging firmware.

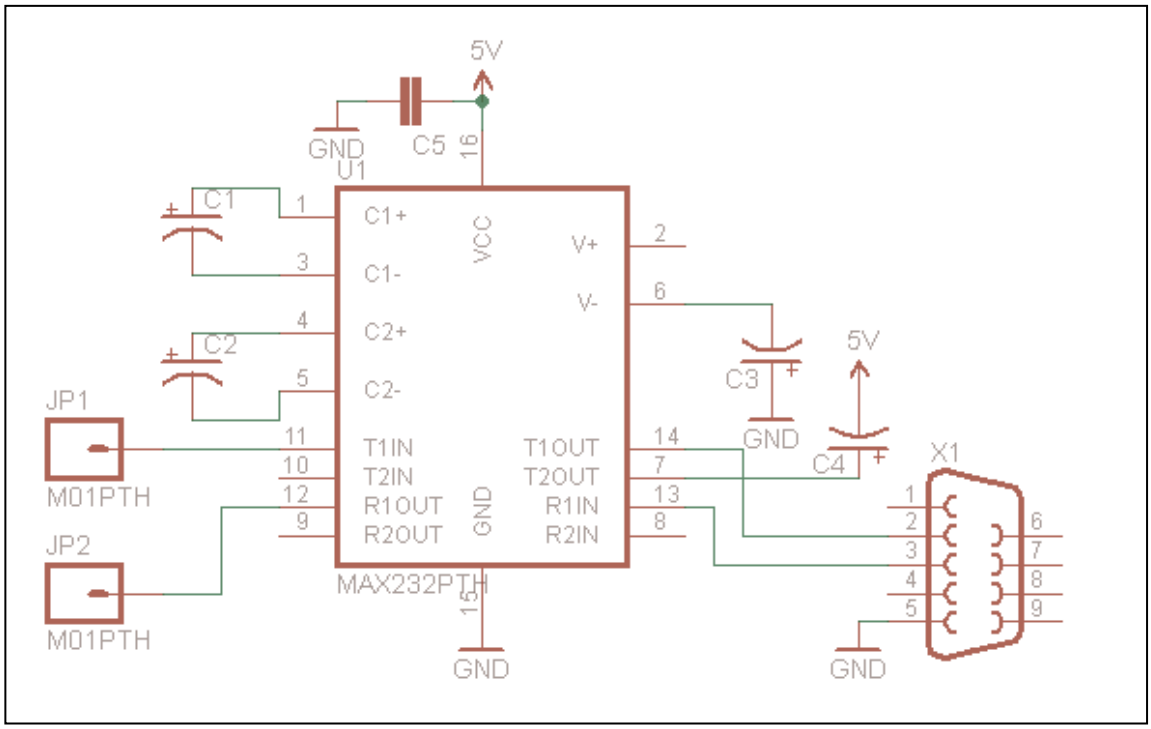


Figure E-1: Serial Line Level Converter Circuit Diagram

Label	Component
C1	10µF, 16V, Electrolytic Capacitor
C2	10µF, 16V, Electrolytic Capacitor
C3	10µF, 16V, Electrolytic Capacitor
C4	10µF, 16V, Electrolytic Capacitor
C5	0.1µF Ceramic Capacitor
JP1	Microcontroller TX Connector
JP2	Microcontroller RX Connector
MAX232	MAX232 TTL-RS-232 Converter IC
X1	DB9, 9-Pin PC Serial Port Connector

Table E-1: Serial Line Level Conversion Unit Component List

Appendix F – 10-Pin IDC Connector Pin Identification

Mounted on the main receiver-unit are three externally-accessible, 10-pin, male, IDC sockets. The following data identifies the purpose of the pins in each connector.

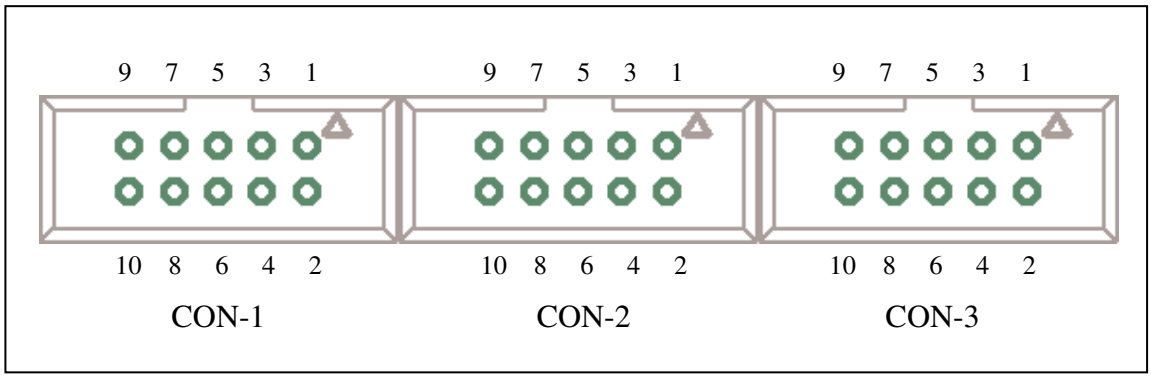


Figure F-1: 10-Pin IDC Connector Pin Numbers

Pin	CON-1 (ISP)	CON-2 (Data I/O for LED-Grid)	CON-3 (Serial RX/TX)
1	MOSI	PB0	VCC
2	VCC	PB5	GND
3	N/C	PB1	TX
4	GND	PC0	RX
5	Inverted Reset	PB2	N/C
6	GND	PC1	N/C
7	SCK	PB3	N/C
8	GND	PC2	N/C
9	MISO	PB4	N/C
10	GND	GND	N/C

Table F-1: 10-Pin IDC Connection Pin Assignments

Appendix G – Attached CD Contents

Included on the CD attached to this document are a number of directories. The names of each directory and its contents are listed in the following table:

Directory	Contents
Datasheets	The datasheets for the main electronic components used in the complete system can be found here in PDF format. Datasheets exist for the microcontrollers, the TX module, the RX module, the DC-DC converter module and the accelerometer.
Documentation	This report in Microsoft Word document format is located under this directory. Additionally, the subdirectory "Images" contains the modified image files used within the document.
Firmware	This directory contains all of the c-code utilised in the final system. Subdirectory "RX" contains the code for the receiver-unit's microcontroller and subdirectory "TX" contains the code for the transmitter-unit's microcontroller. Note the USB driver and its licensing information (see Appendix A) is located within the directory "usbdrv" which is under the RX subdirectory.
Photos	Photos of the completed transmitter and receiver hardware units can be viewed under this directory.
Reference Projects	A number of existing projects provided example firmware and hardware configurations that assisted in the progression of this project. These projects (including firmware and schematics) are located within this directory.
Schematics	The circuit diagrams used in this document were exported from schematics created using the free CadSoft Eagle software package (www.cadsoft.de/). The schematic files are located within this directory.
HID Data	HID Documents as per section 2.3.2 HID-Class.

Table G-1: CD Directory Content