Links Performance Inc.
School of Engineering Science, Simon Fraser University
8888 University Drive, Burnaby, BC, V5A 1S6
links-440@sfu.ca

March 2, 2003

Lucky One
School of Engineering Science
Simon Fraser University
Burnaby, British Columbia
V5A 1S6

RE: ENSC 440 Design Specifications

Dear Lucky One:

Attached is the Links Performance Inc. (LPI) group's *Design Specifications for the SmartShifter™ System,* which outlines the design of our project for ENSC 440. Currently, we are in the process of designing and implementing an automatic transmission for bicycles, which will allow the bike to shift gears automatically. Also, we are designing a UI such that all the pertinent information will be displayed to the rider.

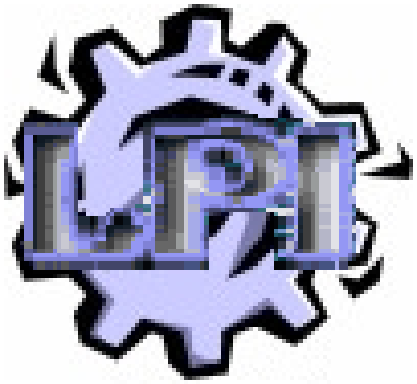
The purpose of the design specifications is to detail the design of our SmartShifter™ system. The enclosed document will provide all necessary information that is required for the completion and success of the project.

LPI consists of four fourth year engineering science students, who are all very dedicated to the SmartShifter™ project. The SmartShifter™ members are Nin Sandhu, Trevor Hawkins, Zafeer Alibhai, and Jack Choi. If there are any questions or concerns regarding the functional specifications, please don't hesitate to contact me at (604) 721-8529 or email the SmartShifter™ team at links-440@sfu.ca.

Sincerely,

Nin Sandhu
President and CEO
Link Performance Inc.

Enclosure: *Design Specifications for the SmartShifter™ System*



Links Performance Inc.

Design Specifications
for the SmartShifter™ System

Project Team:
Nin Sandhu
Trevor Hawkins
Zafeer Alibhai
Jack Choi

Contact Person:
Nin Sandhu
links-440@sfu.ca

March 2, 2003

Executive Summary

Cycling has been a growing sport since it was first introduced and over the years cycling has advanced to different terrains, riders, and has become a great pastime. The SmartShifter™ system is aimed at making riding even more enjoyable by removing the need to shift manually. The SmartShifter™ allows riders to focus more on the surroundings than worrying about what gear the bike is in. The SmartShifter™ will provide all the information the rider would need to an easy to read LCD UI located on the handle bars.

The SmartShifter™ consists of hardware and software. The hardware of the system is made up of:

- Actuator to move the derailleur mechanism
- LCD to display the UI information
- Sensors to measure speed and rpm
- Microcontroller to
  - Control derailleur movement and ensure alignment
  - Provide crucial information to the UI
  - Store system configurations

The software of the SmartShifter™ system will:

- Interpret the sensory data
- Perform simple arithmetic computations
- Update the UI
- Allow intercommunication between all of the hardware components

The final working prototype will incorporate all of features discussed above and will be completed by April 2003.

Table of Contents

1. Introduction

The SmartShifter™ is a device that adds automatic shifting to a standard mountain bike. The system shifts gears using an actuator and communicates with the rider via a UI. The product development cycle will be split in stages, starting with a prototype deliverable for April 2003 and further design and development will be conducted to bring the final product to market.

1.1 Scope

This document summarizes the design requirements of the SmartShifter™ system. This document will be the basis for the design of the working prototype. Although, this document does not completely outline the final working prototype it does however, incorporate the design of the major building blocks. The finer details will be determined as the project progresses.

1.2 Intended Audience

This document details the design requirements of our SmartShifter™ system, which implements an automatic gear shifting solution for bicycles. The design specifications are intended for our investors and the SmarterShifter™ design team to ensure proper functionality and design of the system. The design specifications will also be used as a verification/evaluation tool to ensure the SmartShifter™ System does what it is expected to upon final design.

2. System Overview

The SmartShifter™ is made up of a number of major components. These components are the UI, the sensors, the gear changing mechanism and the microcontroller as shown in Figure 2.1 below. All of these components will intercommunicate and be powered by a power supply.

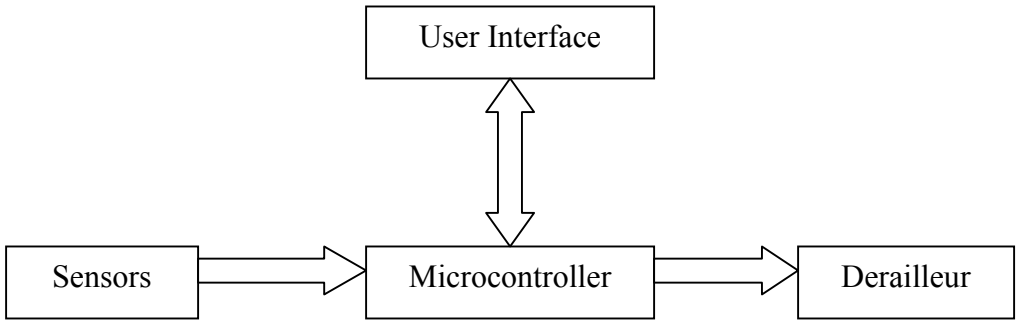


Figure 2.1: System Overview

2.1 Power Supply

There will be two options for powering the system. The first option is a standard lab power supply; while the second is a rechargeable battery pack. All of the components will have to be directly (or indirectly) connected to the power supply in order to operate.

2.2 UI

The UI will allow communication between the microcontroller and the user. The communication will be achieved with the use of a 4-line character LCD, rocker buttons, and standard push buttons.

2.3 Sensors

Two hall-effect sensors will be used as inputs to the microcontroller and therefore the control algorithm. One sensor will measure the cadence while the other will measure the speed of the bicycle.

2.4 Gear Changing Mechanism

The gear changing mechanism will be made up of an electro-mechanical device. The main part of the device will be a digital linear actuator (DLA). The control of this device will be an output of the microcontroller via a stepper drive.

2.5 Microcontroller

A M68HC11 microcontroller will be used to control the SmartShifter™ system. All of the other devices will be connected to the microcontroller via each input or output ports as required.

3. Physical Requirements

The mounting of the hardware will be such that the SmartShifter™ will not interfere with the riders operations, but will function correctly. We have chosen key locations on the bike that will achieve this goal. The following figure 3.1 outline these locations,

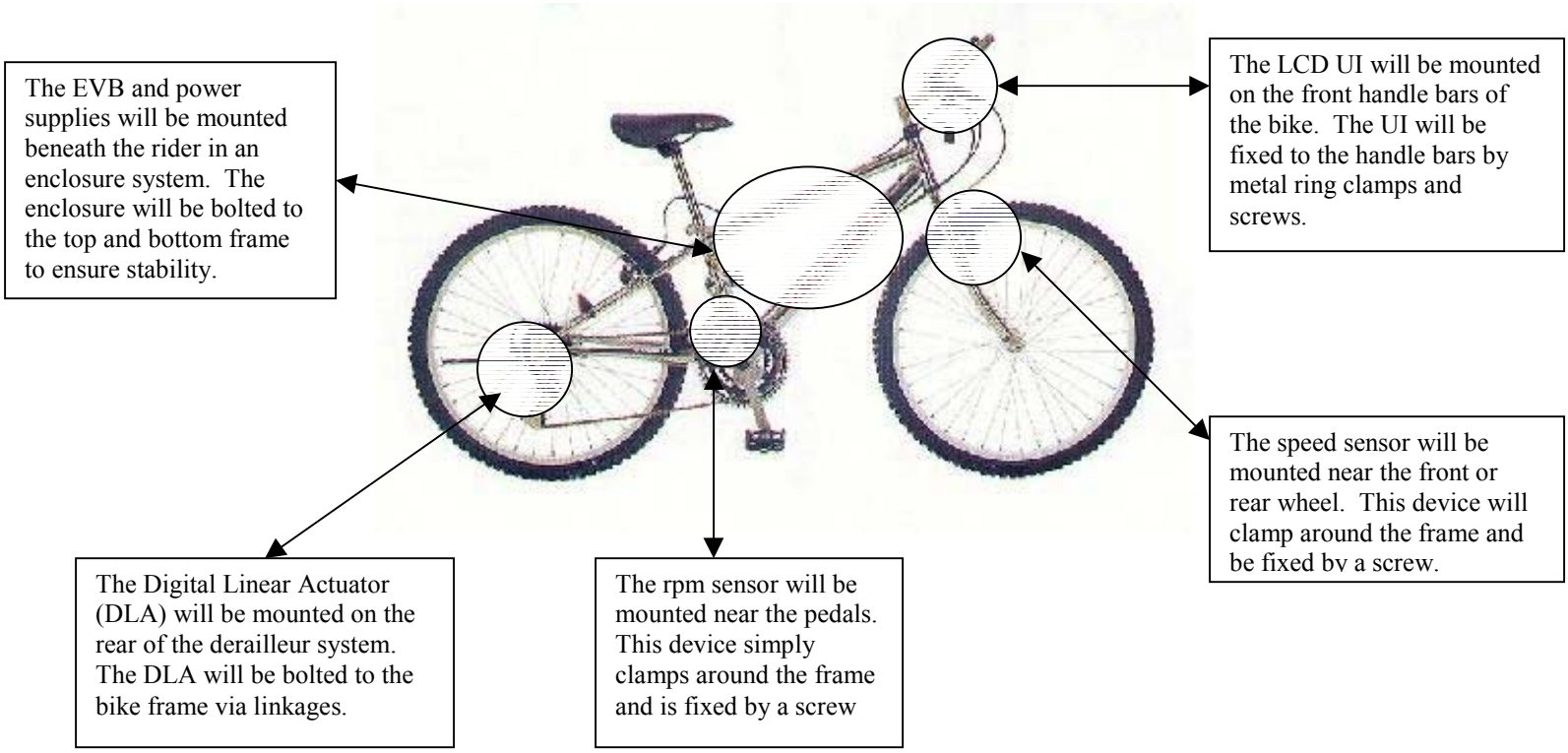


Figure 3.1: Mounting Locations

All wiring to and from input/outputs will be strapped along the frame, so that they do not interfere with the bike operations. The overall weight of the bike will increase somewhat, but it will not impede on the ride.

4. Battery Power

The SmartShifter™ System will be run off a standard power supply unit available in any lab. Our reasoning for this is that we want the SmartShifter™ system to be fully functional in a lab environment before continuing to a more outdoor use. However, if extreme complications do not occur with the project, we plan on implementing a battery pack that will run our system. The battery pack will be a 12V rechargeable cell capable of supplying an appropriate amount of current for our application. The rated max current needed was approximately 1-2 amps based from the DLA specifications. The following battery is the type that will be considered if all goes well because it meets the voltage and current specifications our project.

Figure 4.1: Rechargeable Battery Supply

The microcontroller will then be run from a simple 9V battery if we choose to implement the battery pack. The microcontroller was chosen to run from a separate power supply due to if the battery supply ever ran out the microcontroller would continue running because it was on a separate system.

5. User Interface (UI)

The UI is what the rider will use to interact with the SmartShifter™ system. It will perform various system functions and also provide the user with important data regarding the gear position, speed, and rpm.

5.1 Hardware

The SmartShifter™ user interface (UI) consists of

1. Main unit for display and setup options
2. Handlebar unit for on-the-fly control

These devices are mounted on the bicycle handlebars, as shown in Figure 5.1.

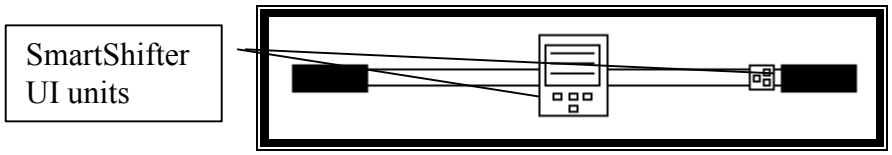


Figure 5.1: Position of SmartShifter™ UI units (Top view)

5.1.1 Display unit

The SmartShifter™'s display unit is mounted on the center of the handlebars. The display unit consists of two grey menu buttons, two buttons labeled ↑ and ↓, and a 4 x 20 character LCD. The functionality of the buttons will be dynamically labeled on the LCD directly the above each button. This unit is used to convey information to the rider and for setup options.

5.1.2 Handlebar unit

The handlebar unit consists of two buttons labeled ↑ ↓ and one button to change between manual and automatic modes. Figure 5.1, above, shows the Handlebar unit on the prototype mounted on the right such that the rider can press the buttons without moving their hands, so we place the most used and important features here. In the automatic mode, ↑ ↓ buttons are used to change the center position of the RPM window. In the manual mode, pressing ↑ ↓ buttons shifts the gear up or down respectively.

5.1.3 Hardware Interface

In total, there are seven buttons, so we have seven signals that are normally Low and go High when buttons are pressed. These signals can all be connected to their own input

pins on the microcontroller, which would of course require seven input pins. Alternatively, we can have them first go through a multiplexer, which allows us to use only three input pins and then decode the pressed button with software. We are not yet sure which option we will use, it will most likely depend on the number of free pins after all other hardware has been connected to the microcontroller.

Due to the two input capture pins used by the magnetic sensors, there are only 2 remaining available input capture pins left on port A. We therefore propose to connect all of the buttons to 7-to-1 AND gate to produce a single signal that goes High when any button is pressed. This single signal will be fed to one of the input capture pins on port A so that we can trigger an interrupt when any button is pressed.

We will connect the 3-signal multiplexed output or the seven signals to either port B (if there is room) or port D. Therefore, when an interrupt is triggered by the input capture pin, we will simply read the values of the buttons from the appropriate port and determine which button was pressed.

Finally, we must connect the LCD to the HC11. We have not made a final specification for the LCD, but we will utilize a standard 4x20 character LCD. Therefore, it should require 8 address/data pins and 3 pins for enabling/control. We will use all of port C for this and three pins of port B. If needed we can devote all of ports B and C to the LCD, the button inputs would then go on port A and D.

5.2 Menu Software

The software side of the UI refers to what will actually be seen on the display and the usability of the design.

5.2.1 Main Display

The Main Display is shown in Figure 5.2. The attributes displayed on the LCD are refreshed every 0.5 seconds.

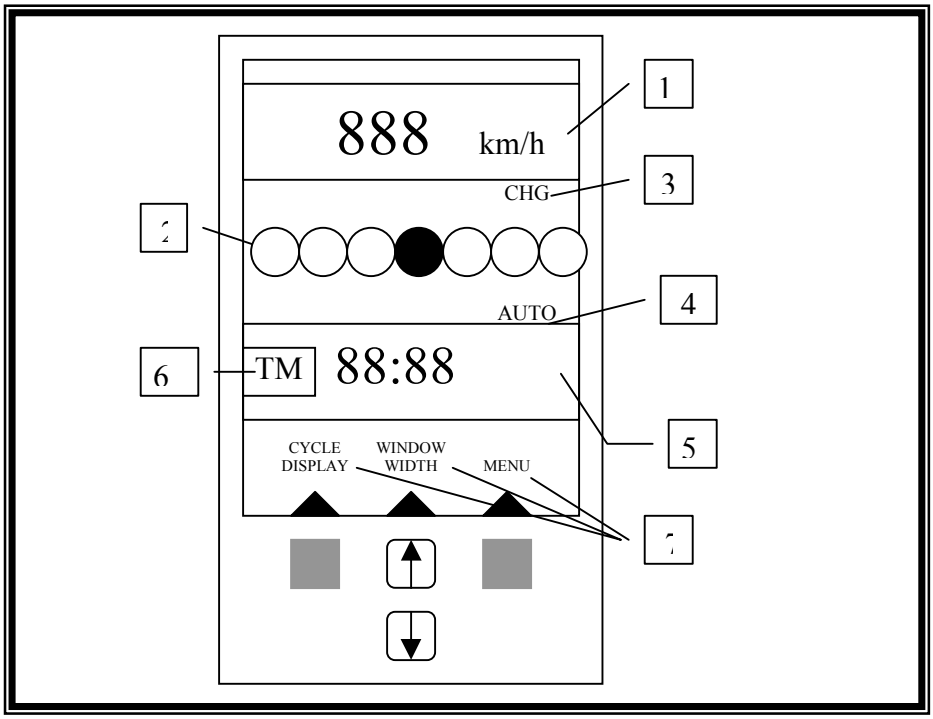


Figure 5.2: User Interface Display Unit (Main Display)

The following parameters are displayed in the main display mode.

- Speed
- Current Gear
- Clock or Timer (Optional)
- AUTO or MANUAL
- CHANGE
- Button labels

The following notes refer to Figure 5.2:

1) Speed corresponds to the current speed of the bike. It is calculated by the microcontroller
(See section 8.2.1 for more details).
2) The circles indicate the current gear. The position of the colored circle represents the same layout seen on the rear cassette (the 8 side-by-side gears of varying sizes). For example, a solid circle at the leftmost position corresponds to the leftmost gear (the easiest) being selected, and so on.
3) CHANGE is displayed when the shift is pending, to give the user some warning.
4) AUTO/MANUAL indicates the current shifting mode.
5) (optional) Displays the current time when or trip time. TM and TT in (6) indicates current time or trip time respectively. We can also add an odometer, average speed, etc.

7) Displays the functionality of the buttons directly below
- The leftmost button will cycle the display - providing the prototype offers cycle able items at position #5, such as the optional clock and timer. (CYCLE)
- The up/down buttons will be used for automatic shifting options, to set the RPM window width. When pressing these buttons the bottom line of button labels will be overwritten to shown the numerical value of the RPM window width as it is being changed.
- The rightmost button will pull up the menu described in the following section (MENU)

Note: When pressing the up/down buttons of the handlebar unit in automatic mode, the display will need to show the numerical value of the RPM window center as it is being changed. We have not decided where this will be written on the LCD, but wherever it is, it should be consistent with how the RPM window width control is handled.

5.2.2 Setup Menu Display

The Setup Menu Display is shown in Figure 5.3, and this menu is displayed when the user hits the button labeled "MENU" while in the Main Display mode.

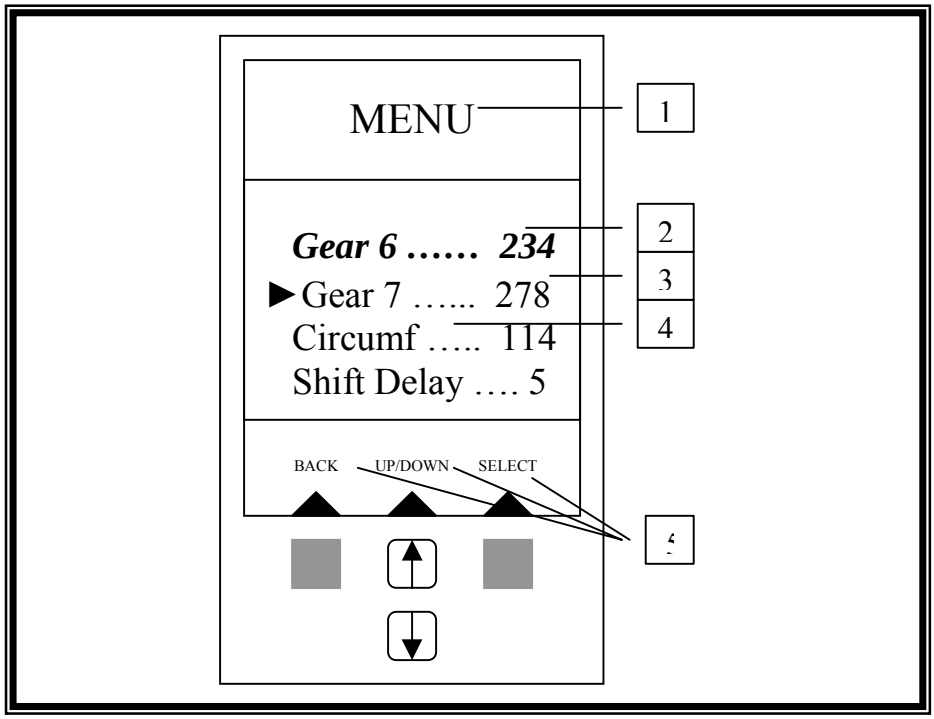


Figure 5.3: User Interface Display Unit (Setup Menu Display)

The following options are displayed in the setup menu display.

- Positions for Gear 1 to Gear 8. Each gear must have a reference position for our actuator, and we will show these as numerical values so that past configurations can be re-tried easily.
- Wheel Circumference setup. The user must enter the circumference of their wheel in centimeters
- Automatic shifting delay: number of pedal cycles outside the RPM window that must occur before a shift is executed. Other automatic shifting options will be added should time permit.

Note that all of these options will not fit in the middle region of the screen at the same time. With a 4 line LCD, only two options would fit at a time, if the menu header and button labels each require a line. Thus, the up/down buttons will be used to scroll through the list of available options and a "►" or other marker will denote the current selection. Pressing the BACK button will return the user to the Main display shown in figure 5.2.

While still in the setup menu, if the SELECT button is pressed, the display will update to indicate that the numerical value associated with the selected option is now to be set with the up/down buttons. The user will be shown that the value can be changed by placing a "↑ ↓" symbol directly beside the numerical value, but beyond this, the display will be as shown in figure 5.3. After the numerical value has been adjusted, the user will press the rightmost button to accept the changes (DONE) or the leftmost button to cancel them (CANCEL). Pressing either of these buttons will end up in them returning to the Setup Menu.

The following notes refer to Figure 5.3:

1) Menu title.
2) Gear Positions
3) Wheel Circumference
4) RPM options
5) Displays the functionality of the buttons directly below.
   - The leftmost button navigates back to the previous menu (BACK) or cancels adjustment of a numerical value (CANCEL)
   - The up/down buttons scroll up and down to select menu items, or increment/decrement a numerical value for adjustment
   - The rightmost button will select a chosen menu item (SELECT) or confirm the adjustment of a numerical value (DONE)

6. Cadence and Speed Sensors

This section details the physical design aspects of the sensors used to measure the rider's speed and cadence, such as device selection, mounting hardware, electrical characteristics and the microcontroller interface. The associated software issues, such as how the microcontroller will know when a sensor's output changes and the algorithms required too compute cadence and speed in real-time are discussed in section 8.2.1.

6.1 Hall-Effect Sensor Characteristics

In order to measure the speed of the bicycle and the rider's pedaling rate (cadence), we will use digital Hall-Effect position sensors. This was a very easy decision as these sensors are used in essentially every bike computer today due to their reliability, cost and lack of physical contact. To measure a speed or cadence, a permanent magnet is mounted on the rotating element (wheel or pedal cranks) and a Hall-Effect sensor is fixed to the bike frame. With every revolution of the rotating element, the magnet will pass the sensor once and there will be one pulse from the sensor. This occurs as Hall-Effect sensors turn on when they sense a magnetic field (South and/or North Pole) in the correct orientation. By measuring the time between pulses, the cadence (in RPM) and speed (in km/h) can be calculated, as described in section 8.2.1.

We will utilize unipolar switching Hall-Effect position sensors, which are turned on while in a South Pole flux with sufficient intensity and correct orientation relative to the sensor exists. These are the simplest type of Hall-Effect sensors, so we are applying the KISS principle here. Figure 6.1 shows the magnetic parameters for one of the sensors we will use, the Honeywell SR15C-A3, for illustrative purposes into the operation of these devices.

MAGNETIC CHARACTERISTICS

		SR15C-A3
Magnetic Type		Unipolar
25°C		
	Max. Op.	180
	Min. Rel.	75
	Min. Dif.	25
-20°C to 85°C		
	Max. Op.	215
	Min. Rel.	60
	Min. Dif.	10

Figure 6.1: Magnetic characteristics of the SR15C-A3 (courtesy Honeywell)

We can be quite confident that there will be only one pulse per wheel/crank revolution with this type of setup. The sensor would only turn on more than once if there were multiple South Pole flux spikes (with intensity above the operating point and separated by periods below the release point) for one pass of the magnet. Due to rapidly fringing

magnetic fields and the fact that the sensor only detects flux in a certain orientation, this seems like an unlikely event if the magnet and sensor and properly mounted.

These sensors have a finite switching time of a few microseconds, but this is easily fast enough for even the craziest of bike riders. Note that both the Hall-Effect sensors and microcontroller can handle switching rates on the order of megahertz. Whereas, the expected maximum rates are 2Hz (at 120RPM) for the cadence sensor and 170Hz (for 120km/h and 26" wheels) for the speed sensor.

The main concern with these sensors is if the magnet is not positioned within 1-2mm (typically) of the sensor, the magnetic flux seen by the sensor may be insufficient to turn it on. Therefore, we must ensure that the sensor and magnet are correctly positioned with sufficiently close spacing for the duration of the pass and that they remain permanently fixed in place.

6.2 Cadence Sensor

One of the sensors we will use is a Honeywell SR15C-A3. We made this decision as it is cost-effective, offers digital output and comes in an easily mountable sealed plastic case. The Honeywell sensor will be mounted to the bike frame such that it is within 1-2mm of a magnet mounted on the pedal cranks. This will allow us to measure the rider's cadence in real-time and compute their RPM. The mounting hardware for both the sensor and the magnet have yet to be designed, but these tasks should not pose significant problems. Figure 6.2 shows the internal block diagram of the Honeywell Hall-Effect sensor.

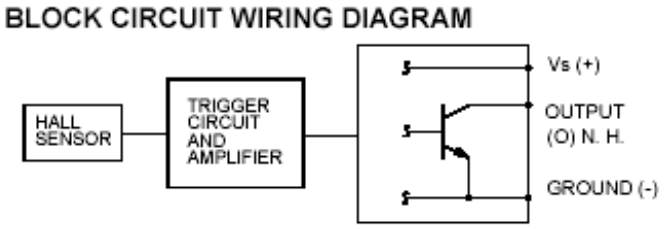


Figure 6.2: Block diagram of the SR15C-A3 Hall-Effect sensor (courtesy Honeywell)

We note that when the magnet passes by the sensor, the output transistor turns on, which pulls the normally high output pin to a low voltage (0.4V). Therefore, we will utilize the interface shown in figure 6.3 to connect the Honeywell sensor to our TTL logic (the HC11 microcontroller).

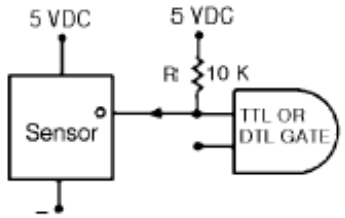


Figure 6.3: TTL interface for the SR15C-A3. (courtesy Honeywell)

Thus, the microcontroller will normally see a High logic voltage, and with each revolution of the pedal cranks there will be one Low logic-level pulse. The interface is comprised of a pull-down resistor, which is chosen so that the sink current (when turned on) is at most 20mA, as specified by Honeywell.

6.3 Speed Sensor

The second Hall-Effect sensor will sense the revolutions of the rear or front wheel so that the bike's speed can be computed. In order to simplify our design and implementation, we are using a scavenged sensor and magnet from an old bike computer. The sensor has existing mounting hardware such that we can easily mount it to the bike frame and position it within 1-2mm of the magnet on the rear or front wheel. The magnet itself mounts on the spokes of the wheel with existing hardware.

The design issues associated with this bike computer sensor are the essentially the same as the Honeywell sensor, as they are both unipolar switching Hall-Effect sensors. The main difference is that the bike computer sensor is a "two-wire" design, with no dedicated output pin. With this two-wire Hall-Effect sensor, the effective resistance of the sensor changes from tens or hundreds of mega ohms when off, to less than a kilo ohm when on. We have only done a brief characterization thus far, so we do not have the exact resistance values on hand. With this knowledge, we design an interface that will pull the microcontroller input pin to a Low logic-level when the sensor turns on, as shown in figure 6.4.

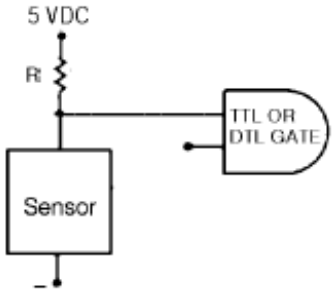


Figure 6.4: TTL interface for the scavenged bike computer Hall-Effect sensor. (Modified from Honeywell)

The pull-down resistor and Hall-Effect sensor form a voltage divider whose output is fed to the microcontroller. Therefore, based on the general voltage divider expression

$$V_{out} = V_{\sup ply}\left[\frac{R_{sensor}}{R_{sensor} + R_{pull-down}}\right], \qquad (6.1)$$

we see that the output voltage will be very near the supply voltage for $R_{sensor} >> R_{pull\text{-}down}$ and near ground for $R_{sensor} << R_{pull\text{-}down}$. This works as the sensor's resistance changes by at least 6 orders of magnitude when it switches. We played with this in the lab and it worked perfectly, with the high and low output voltages being very near the positive and negative rails, respectively. Thus, the electrical characteristics of this interface will be the same as the Honeywell sensor: both are normally High and produce one Low logic-level pulse per revolution.

In addition, the pull down resistor must be large enough so that the sinking current when the sensor is turned on is not excessive. As we do not have the specifications for this scavenged sensor, we will design for a maximum sinking current of less than 20mA, as this is acceptable for a wide range of Hall-Effect devices.

7. Gear Changing Mechanism

This section details the design of the hardware required to change gears on a standard chain-driven bicycle transmission. The software and controller issues associated with this mechanism are discussed in section 8.2.2.

7.1 Derailleur Mechanics

As detailed in LPI's Project Proposal document, we will use a complete standard bicycle transmission. Our implementation will require the rear derailleur to be modified. As a result, we require an actuator to interface with a chain-pulled rear derailleur in order to move the derailleur in a controlled manner. Figure 7.1, below, shows the two extreme positions of a standard rear-derailleur.

Figure 7.1: Standard rear derailleur. (courtesy HowStuffWorks.com)

As the derailleur moves between these extreme positions, it forces the chain to derail and change gears. Figure 7.2 illustrates in greater detail the mechanism that permits the derailleur to accomplish this task.

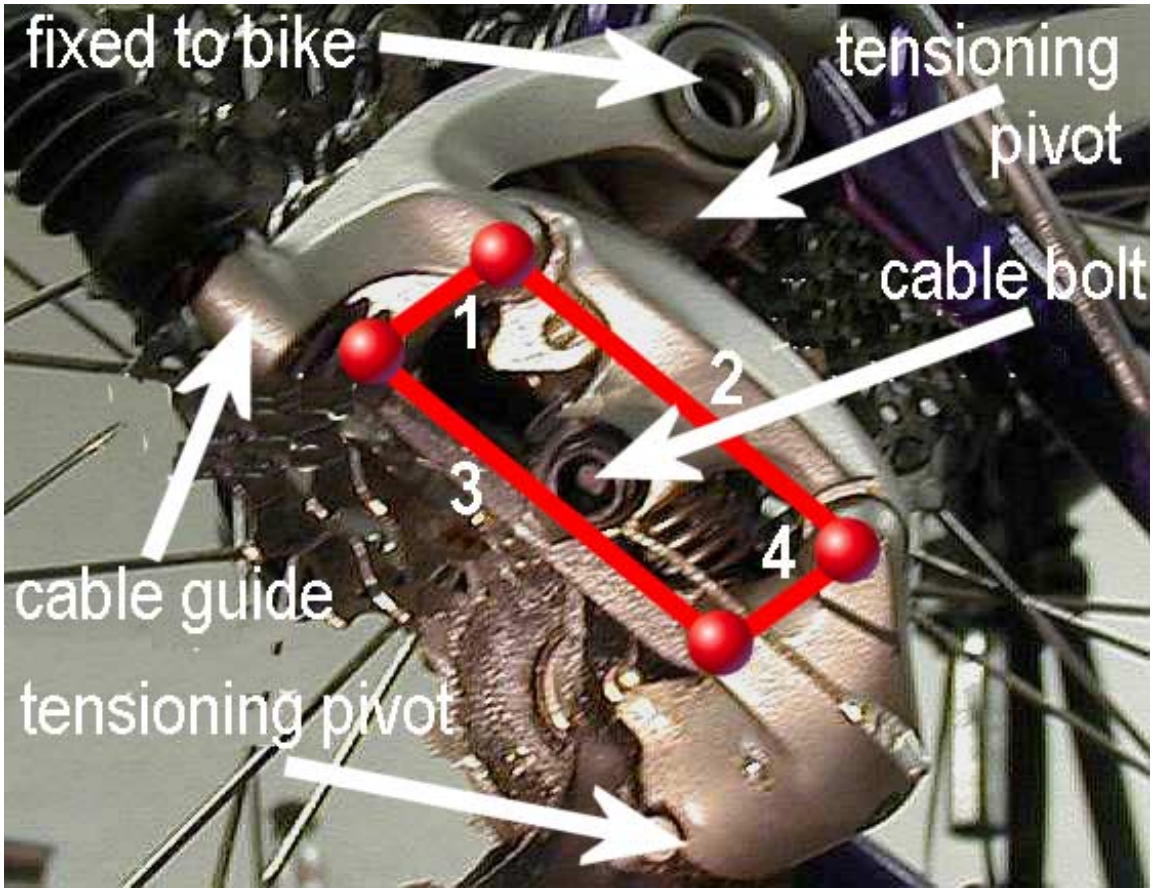


Figure 7.2: Four-bar parallelogram of a standard rear derailleur.

As shown in figure 7.2 above, the derailleur consists of a four-bar parallelogram mechanism. Link 1 is connected to the bike frame via a bolt at the very top of the link, anchoring the entire mechanism to the bike frame. The pulleys that the chain runs on are fixed to link 4, which can be viewed as the output link - it actually does the shifting.

When the user pulls the cable (via the shifter), the derailleur moves towards the easier (larger) gears. This movement occurs when the parallelogram moves, the distance between certain points on links 1 and 2 changes in an almost constant manner. This roughly linear distance change is exploited by the cable guide on link 1 and the bolt that fixes the cable to link 2, actuating the parallelogram by cable. In order to move the derailleur in the other direction, there is a spring inside the parallelogram, forcing it towards the harder (smaller) gears when no cable tension is applied.

Finally, there are two pivots with internally loaded springs (labeled in figure 7.2), which tension the chain as required. These tensioning components are completely independent of the parallelogram shifting mechanism, which is the genius of this mechanism.

7.2 Actuator Design Considerations

In order to move the derailleur parallelogram under control and effectuate gear changes, we came up with two main design ideas:

1) A remotely mounted actuator pulls the cable
2) A linear actuator is mounted directly on the derailleur.

The first idea allows greater flexibility in terms of space and packaging, but the latter removes the cable from the system. Cables are troublesome as they stretch significantly and we were unable to design a realizable feedback device that would ensure that the derailleur is correctly aligned with the correct gear.

Therefore, we will directly mount a linear actuator on the derailleur, replacing the cable. The linear actuator will relate the geometry and position of the derailleur parallelogram to the corresponding gear. This relationship should always hold, provided that the derailleur cage or bike frame is not bent or damaged. For simplicity we plan to use the cable guide on link 1 and the cable bolt on link 2 as the primary mounting points of our actuator mechanism, as shown below in figure 7.3.

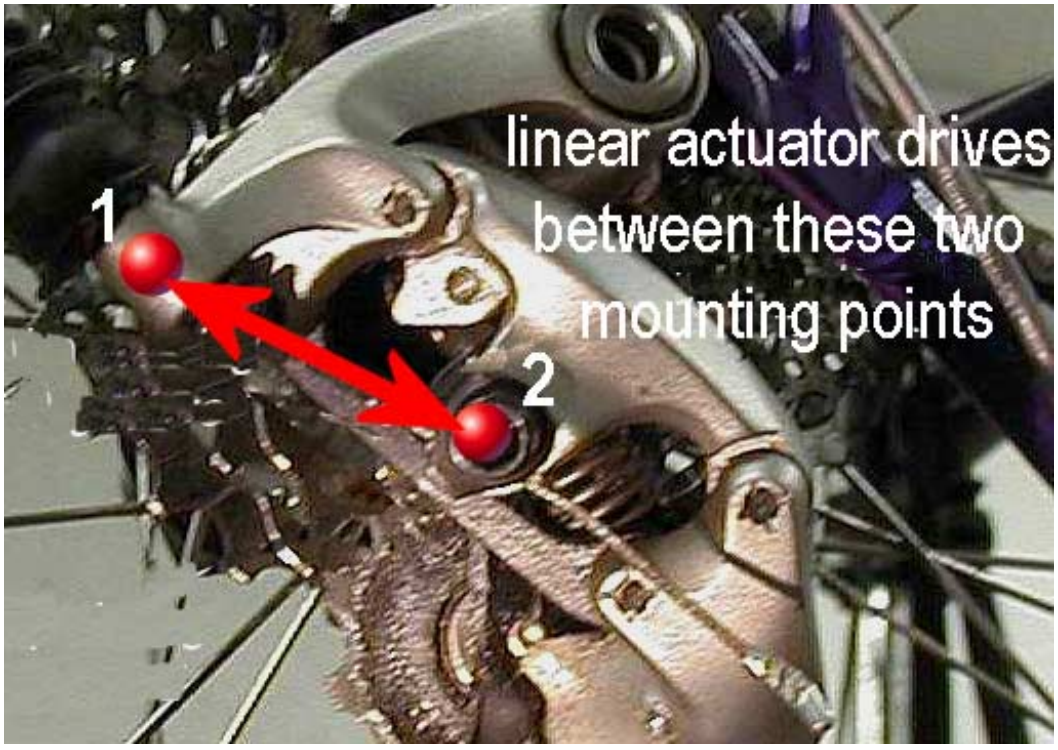


Figure 7.3: Linear actuator mounting points.

By using these points we will use the most straight-line or linear path and save considerable time when mounting the actuator. In addition, the use of a bi-directional actuator will allow us to remove the spring internal to the parallelogram and save a large amount of otherwise wasted energy.

We built a bike stand to elevate the rear wheel and removed the internal rear derailleur spring so that we could characterize the force required to actuate the parallelogram. Using a spring scale we performed tests at no load. We determined that approximately 30N is required to shift the derailleur in either direction throughout the range of gears. Unfortunately it would take a much more elaborate test setup to measure the force required under a heavy pedaling load, so we have not performed these types of tests for the prototype. We are designing the actuator for an expected force of 50N so that we have some leeway for road testing the prototype.

Next, we characterized the distance change between link 2 (with the cable bolt) and link 1 (fixed, with the cable guide). The total distance change is 13-15mm, depending on the exact derailleur used (we have a number to choose from) which represents a change of 7-8 gears. The distance change between each gear is roughly constant as determined from the indexing on shift levers. Therefore, each gear change will require a linear change of approximately 1.8mm. By examining the spacing of the sprockets on the rear cassette and moving the derailleur around, we estimated that there should be at least 20 incremental positions between each gear. Consequently, we require 20 steps in the 1.8mm of linear travel per gear, which corresponds to a minimum resolution of 0.1mm for the linear actuator.

We will need to characterize the motion of link 2 relative to the fixed link in three dimensions so that we can determine how many degrees of freedom (DOF) the linear actuator mechanism needs to have. We know that it will require at least 2 DOF; a simple straight-line actuation (1 DOF) is not possible as the path of link 2 has a curvilinear translation element. If we determine that the path of link 2 can be placed in a plane (2 DOF), we simply need pivots at each mounting point whose axes are perpendicular to this path-plane. Should it be impossible to describe this path within a plane, we will need a mechanism that allows for 3 DOF, such as a universal joint, at each mounting point. As we are still evaluating the merits of different derailleurs, we have not been able to make a final characterization of this motion, but we recognize that this is a high-priority and critical task.

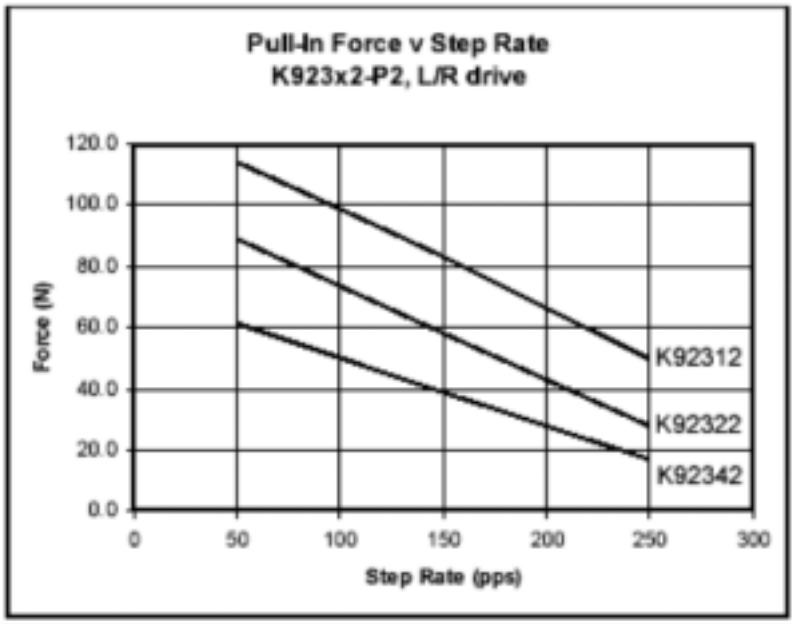
Having characterized the requirements for a linear actuator to be mounted on the derailleur, we performed some research to see what actuators are available that provide the necessary force/torque in a small lightweight package. We came up with two main design ideas for the linear actuator:

1) A rotary motor (DC or stepper) that turns a screw to accomplish a linear translation
2) A digital linear actuator (DLA)

We choose to use a digital linear actuator primarily as they are powerful and are available with de-energized holding torques greater than 50N. This means that the derailleur would stay in place between shifts without drawing any power from the batteries. With almost any other actuator we would need to build an actuated clamping or ratchet device to accomplish this task. Therefore, the linear actuator will save us from designing a screw mechanism and a clamping mechanism. In addition, these devices are reasonably cost effective, lightweight, compact and based on standard stepper motors.

7.3 Digital Linear Actuator

The DLA we choose is a Mclennan K92322-P2. We originally specified the same motor from Thomson Industries, but we found that only Mclennan, who are located in the UK, have the part in stock. We examined other types of DLA, but we could not find any that meet the prodigious de-energized holding torques that these Mclennan/Thomson motors provide. Figure 7.4, below, shows the relevant specifications of the K92322-P2.



Part Number	DC Operating Voltage	Maximum Travel	Linear Travel Per Step (inch based)	Maximum Force (typical)	Minimum Holding Force (de-energised)
Bipolar					
K92322-P2	12 V	24.1 mm	.002" (0.051 mm)	89.0 N	83.4 N

Coil Data (Unipolar & Bipolar types)	5V Uni	5V Bi	12V Uni	12V Bi
Resistance Per Phase	5 Ω		28.8 Ω	
Inductance Per Phase	3.7 mH	5.5 mH	19.7mH	39.3 mH
Power Consumption	10 W			
Insulation Resistance	20 MΩ			

Bearings	Radial Ball
Weight	156 gm (5.5 oz)
Operating Temperature Range	-20°C~70°C (-4°F~158°F)
Storage Temperature Range	-40°C~85°C (-40°F~185°F)

Figure 7.4: K92322-P2 bipolar DLA specifications. (courtesy Mclennan Servo Supplies)

Note that we should be able to maintain a drive torque of 50N up to 175 pulses per second, which corresponds to a linear speed of 9mm/s. This speed will permit the entire range of gears to be shifted in less than 1.5 seconds, which is roughly on par with cable-driven derailleurs. The resolution of this actuator is given by the 0.051mm linear travel per step, which easily meets our requirement of 0.1mm. The minimum de-energized holding force is quoted to be 83N, so we do not expect the DLA to ever slip significantly between shifts.

We chose a DLA model that uses a standard 2-phase, 4-lead, bipolar stepper motor setup. This decision was made as bipolar motors are more efficient and a 12V DLA will draw less current than one that runs on 5V. We can use standard stepper motor drives to

control our DLA, which is nice, as we are familiar with these. The DLA runs on 12V and draws 10W of power, which means we require a stepper drive that can handle roughly 1A of current continuously at 12V.

Figure 7.5, below, details the external dimensions of the DLA

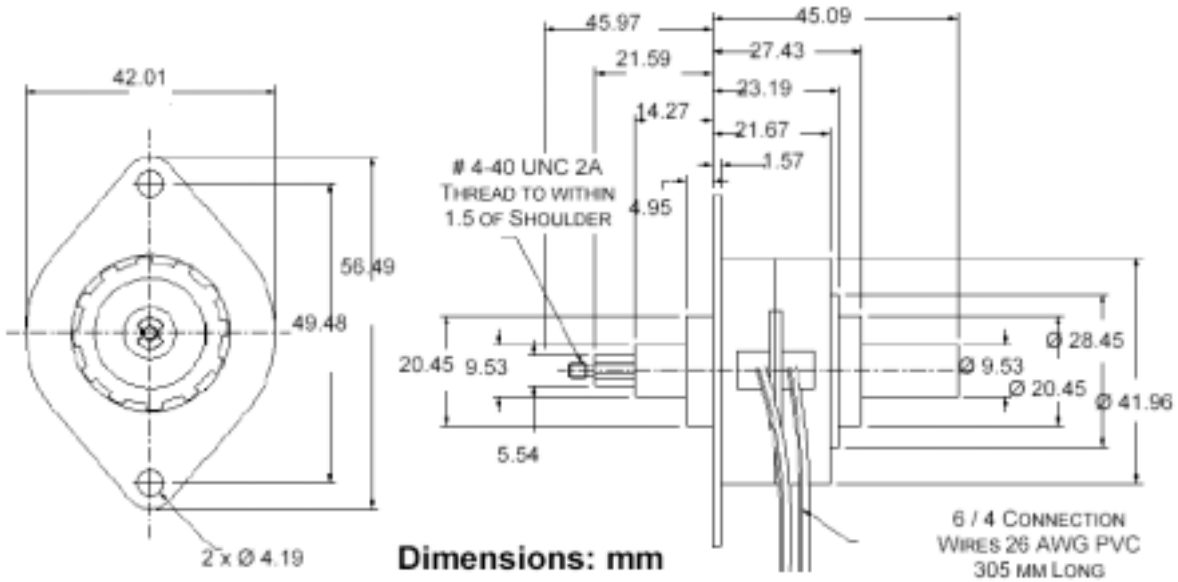


Figure 7.5: K92322-P2 dimensions. (courtesy Mclennan Servos)

The derailleur parallelogram's long links are roughly 45mm long and the short links 15mm long (the red links in figure 7.3). The link that is fixed to the bike is approximately 65mm from the bolt at the top to the cable guide at the bottom. Thus, the DLA is roughly the same size as the derailleur. However, we have ample room outside the derailleur parallelogram and we are confident that we can design suitable mounting hardware to make this possible.

The DLA weighs only 156 grams, so the main issue with the mounting hardware is that it can properly transmit the force applied by the DLA to the derailleur parallelogram without excessive flexure or derailleur slop. Note that we plan to mount the heavy end of the DLA onto the link of the derailleur 1, which is fixed to the bike frame. If bulk of the DLA mass were distributed onto link 2, this would tend to move the derailleur parallelogram down and towards the easier gears. Essentially, we wish to place as much of the mass as possible on the link that is fixed to the bike. Even if we accomplished putting 100% of the DLA mass on the fixed link, it would still probably affect the chain tensioning of the derailleur. This occurs, as the DLA will induce a torque on the fixed link, which is actually pivoted at the top with a spring for chain tensioning. However, it would be pretty easy to add external springs that provide a counteracting torque to that

produced by the DLA's mass.

The motor has just been ordered recently and we are now in the process of designing the mounting hardware that will allow us to actually use this DLA to move the derailleur. This is likely the most difficult task in the entire project that remains to be solved, so we plan to work feverishly at this. Once we have fully characterized the derailleur mechanics and designed preliminary mounting hardware ideas, we will choose a final design. The finalized design will likely be modeled using CAD so that we can get the hardware cut by the SFU machine shop's CNC machining equipment.

7.4 Linear Actuator Electrical Interface

We have yet to specify with certainty the stepper drive that we will use to control the DLA, as there are many suitable ones available. The stepper drive will form the interface between the DLA and the microcontroller. As mentioned previously, we require a drive for a two-phase bipolar stepper that supports a 12V power supply with the ability to handle roughly 1A of continuous current. Ease of use and cost are probably the most important features of the stepper drive. We will consider standard full-step drives, as micro-steppers and chopper drives offer increased performance but at extra cost and complexity.

We will likely use a Mclennan PM542 drive as it meets the above-mentioned requirements, or an HIS 39105 drive. Both of these drives are similar as are many others available. Figure 7.6 shows the schematic of the HIS stepper drive for illustrative purposes, as we will almost likely be going with this type of drive.

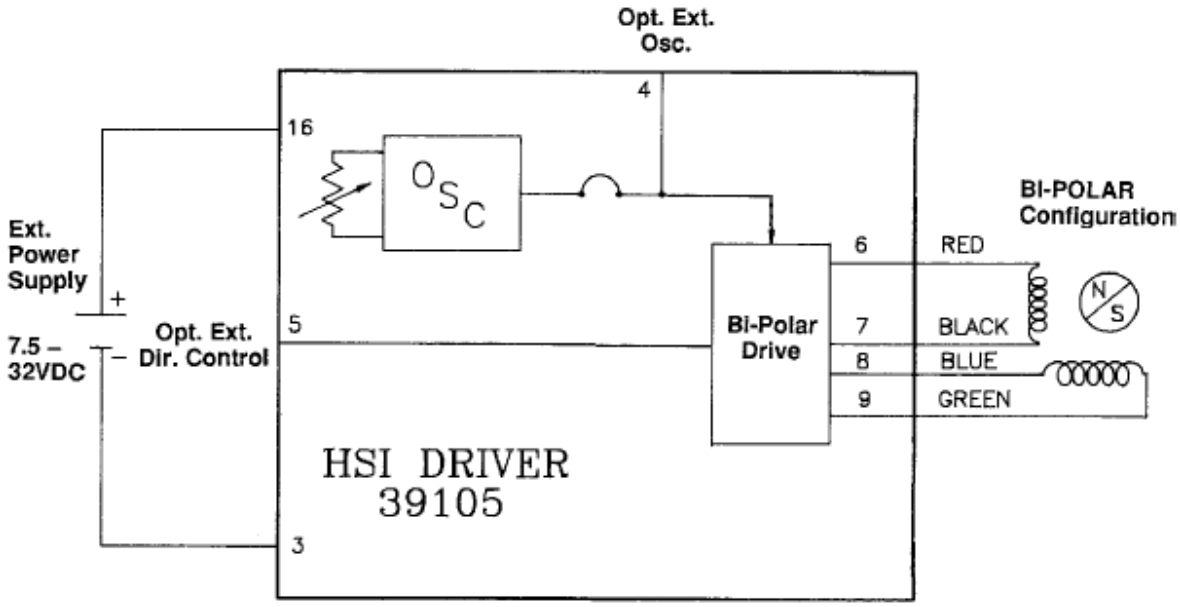


Figure 7.6: Example stepper driver. (Courtesy HSI Motors)

The drive is quite simple and we can derive the following table of inputs and outputs.

Inputs	Outputs
• Power Supply • Clock Signal • Direction Signal	• Drives DLA coils

The power supply will of course be at 12V to supply power for the stepper and motor. The clock is used to control the stepper; with every High voltage pulse, the drive will step the motor once. The drive keeps track of which coils are energized so that it is able to step properly. The direction signal is used to control which way the DLA moves, a High voltage level could indicate "in" and Low voltage "out", or vice versa. Therefore, we see that the microcontroller can easily interface to the DLA (only 2 output pins required) and control it as required.

7.5 Actuator Feedback

The DLA we will use is essentially a modified stepper motor, and it is possible to run it open-loop with no problems. For the prototype, we are very much leaning towards running the DLA in open-loop. In this setup, we would always write the current gear or position of the DLA into a non-volatile region of the MCU's memory (EEPROM) so that the system knows where the derailleur is upon every power up. This combined with

robust code should provide reliable operation as the stepper has sufficient drive and holding torque, meaning it should never slip or miss a step.

However, it might be nice to have some sort of feedback device so that we can home the derailleur to one of the extreme positions in case of unexpected errors. In addition, when the stepper drive is turned on, it may move the DLA as it will not know what sequence of coils should be energized to move exactly one step from the current position. Note that it is safe and easy to home to either extreme position as the derailleur offers mechanical set-screws that provide solid limits to the motion. We could use a Honeywell SR15C-A3 Hall-Effect sensor (see section 6.1) as a limit switch for feedback, or a physical contact limit switch. The switch would be mounted such that when the derailleur just reaches the home position (at either extreme gear), it turns on. With this setup, in order to home the derailleur the DLA is stepped in the correct direction until the switch turns on.

Note that the DLA has an automatic clutch to prevent internal damage, so we can have the user provide visual feedback and home the device themselves in the unlikely event that the system looses track of where it is in the open-loop setup. This could be a very simple option on the UI where the user holds down a button until the derailleur reaches the home position.

Realistically, the level of effort and time placed into actuator feedback depends on how high a priority we make it (as compare with other features). We do not view this task as being critical as open-loop control should work quite effectively, especially for the prototype.

8. Microcontroller

In this section, we will focus on the hardware and software applications of the microcontroller we chose for our SmartShifter™ System. The microcontroller is the heart of the system which allows all our sensors, actuators, and UI to all intercommunicate.

8.1 Microcontroller Hardware

The microcontroller that was chosen based on our project requirements was the Motorola MC68HC11. This device is responsible for obtaining external sensory data for the speed, rpm, and buttons, moving the derailleur to its appropriate gear location, displaying speed, rpm, and gear location to the UI, updating the UI, and storing wheel circumferences and other system configurations. We decided to use the MC68CH11 because:

- the device has efficient number of input/output pins to meet our needs once multiplexed
- the device has 512 bytes on-chip electrically-erasable EEPROM
- the device has 256 bytes on-chip SRAM and 8k ROM
- the device has a real-time interrupt (RTI) suitable for when manual mode gear shifting is needed or whenever the user uses the UI
- the device has three input-capture functions and five output compare functions
- the device has a A/D converter and 8-bit pulse accumulator circuit
- the device is programmable in Assembly language, which we all ready know how to program
- the device has power saving modes
- the device is capable of doing simple arithmetic calculations
- the device was obtained free through the donation of the EUSS, which made it very cost beneficial
- the device does not require a lot of power to operate (5V)
- the device is easy to interface and work with

The following figure 8.1 is the microcontroller device we plan on using for our application.

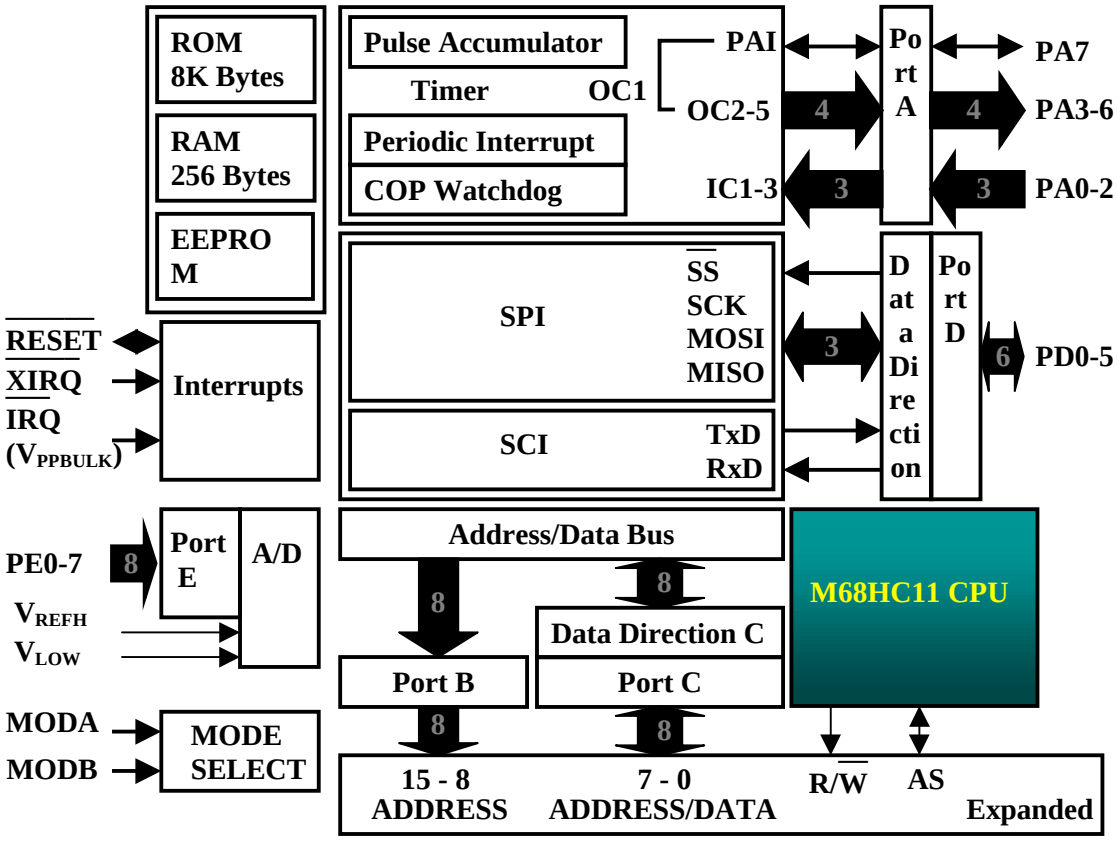


Figure 8.1: M68HC11 Diagram

Port A pins PA0-PA2 from the MC68HC11 chip will be used to read in the data from the speed and rpm sensors. The pins PA3-PA6 will be used as output for the derailleur movement and updating the UI. Pin PA7 can be programmed to do whatever extra function we wish.

The interrupt function will be used to process requests from buttons/switches and external sensors. Once the interrupt is activated it will execute code appropriate to the function that requested the interrupt, which will be discussed in more detail in the software section. The buttons will be using Port D and the bidirectional pin on Port A.

The input/output timer capture functions will be used to determine the speed and rpm of the bicycle. The speed and rpm is determined by the time between the pulses and some simple calculations discussed later in the document.

The LCD unit will interface via Port C because it has the appropriate number of pins and Port C is bidirectional allowing easy interchange of data.

The SmartShifter™ program will be programmed into the 8kb ROM, which is plenty of space for our simple application. Therefore, every time the system is activated our program will initiate immediately. The EEPROM which stores 512 bytes will be used to store user data when the SmartShifter™ system is running. Things such as wheel circumferences and derailleur positioning will be stored into the writable EEPROM because they are user specific and cannot be hard coded.

The operation mode of the microcontroller will be in single chip mode (SMOD) because we are not using expanded or any special modes of the chip.

8.2 Microcontroller Software

The programming for the microcontroller will be done using simple Assembly language. This choice was based on the fact that our system will only be doing simple arithmetic calculations and any other language such as C would use a lot of our on board available memory and possibly be less efficient than low level assemble.

The following flowchart will show the overview of how the software algorithm will interact with each piece of hardware in order to implement the control algorithm. The first flowchart, figure 8.2, details the overall program with the interrupt handlers. The second flowchart, figure 8.3, shows the main program logic that runs while the controller is not handling an interrupt.

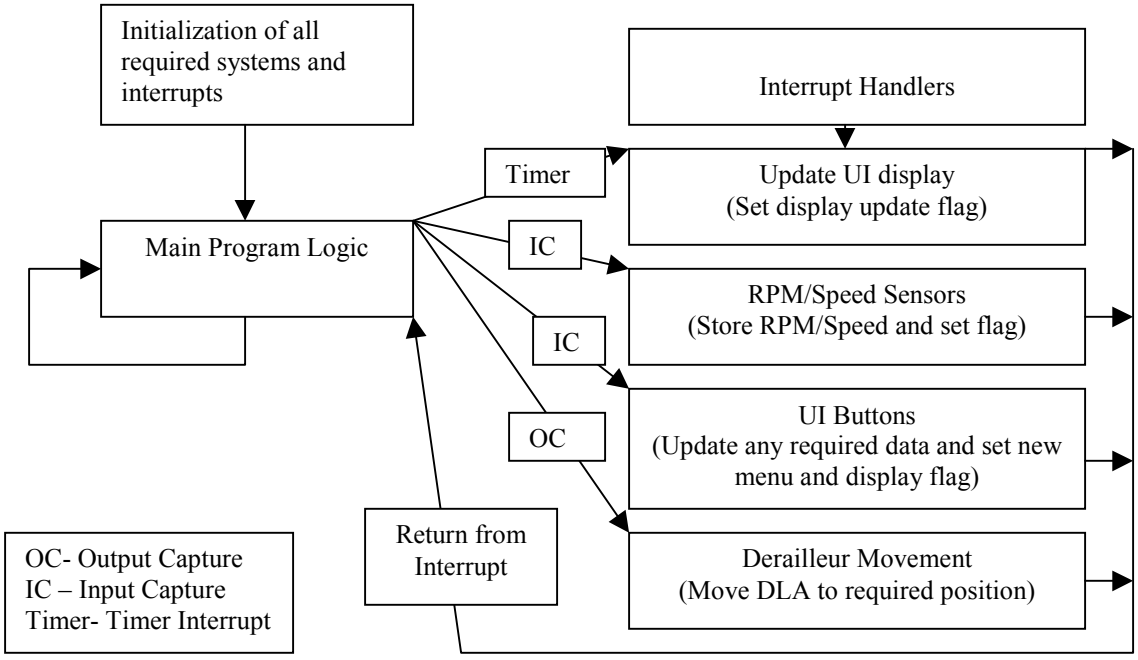


Figure 8.2: Software Overview with Interrupt Handlers

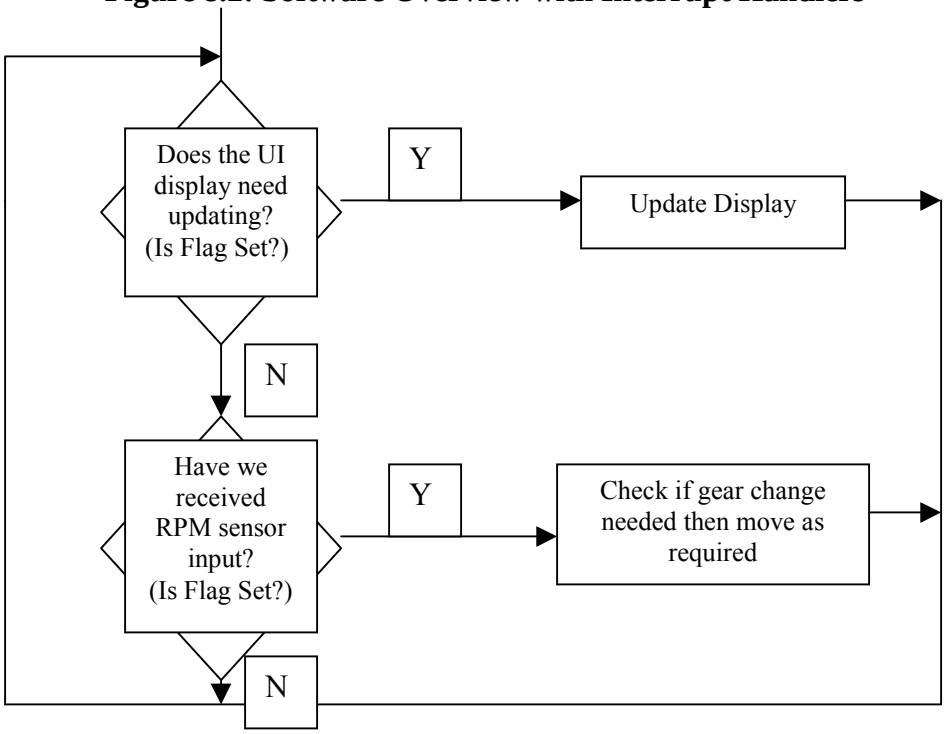


Figure 8.3: Main Program Logic

8.2.1 Sensory Algorithm

The speed and rpm sensors will connect via Port A and their operational principle is discussed in section 6.1. When the input pin is pulled low (logic 0) this will trigger an interrupt for the timer input capture, and we save the value of the timer. When the pin goes high and low again (another low pulse), we get the timer value again and take the difference from the original to find the elapsed time between low pulses. The following figure 8.4 is a flowchart of how the capture functions will work once it enters the interrupt routine. Once the program has completed it will return from the interrupt and continue where it left off.

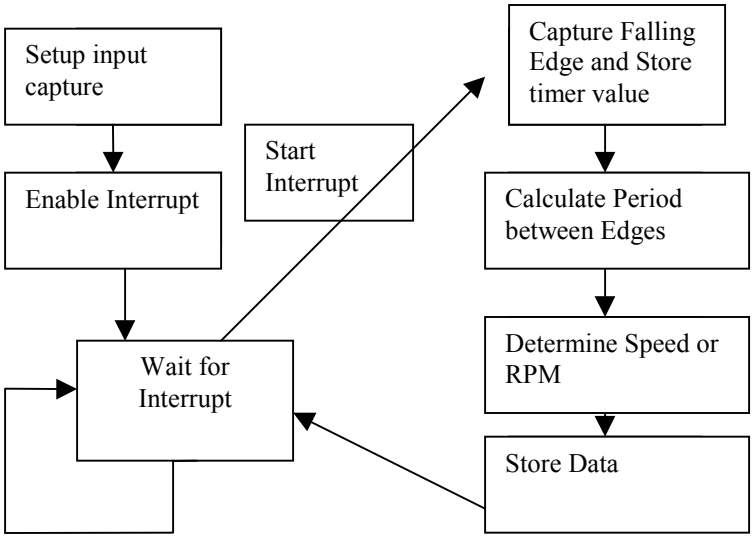


Figure 8.4: RPM/Speed Interrupt Handler

The captured values will then be stored in a specific registers. The only restraint is that we can only measure a signal period no longer than 2^{16} E clock cycles, but for our application we have fairly short pulses which vary by the circumference of the wheel and therefore this restriction will not apply here.

Taking the difference between the two timer captured values we can determine the period of the revolution. The following equations describe how the rpm and speed are calculated using the period.

$$RPM = \frac{2\pi}{(60)T} \tag{8.1}$$

To calculate the speed of the bicycle we will use the following equation,

$$Speed(km/h) = 360\frac{wheel\_circumfrence(cm)}{T(\text{sec}onds)} \tag{8.2}$$

8.2.2 Derailleur Algorithm

The derailleur algorithm is responsible for ensuring that the derailleur moves and is in the correct position. Two output pins on Port A (PA3-7) will be used to communicate with the derailleur system. One pin is used to set the direction of movement and the other pin is used to step the DLA (one pulse = one step). The following flowchart outlines the algorithm which will be used for the derailleur movement.

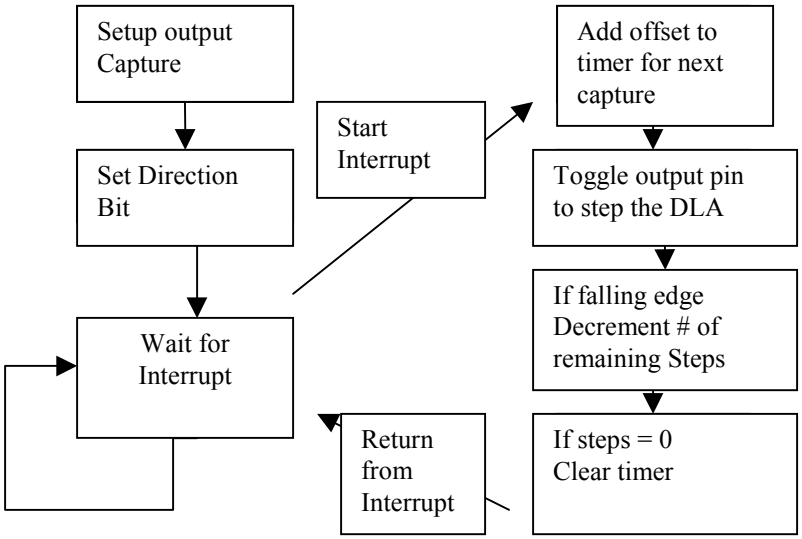


Figure 8.5: Derailleur Interrupt Handler

When the control algorithm decides a shift is needed it will determine the required number of steps to move and the correct direction. This information is passed to the derailleur movement software which then sets up an output capture to fire with twice the desired step (we have to toggle the output pin high and then low for one step) and enables the interrupt.

When the timer reaches the correct value an interrupt occurs and the derailleur interrupt handler will be called. Once in the derailleur interrupt handler the software will add an offset to the captured timer. The software will toggle the step output pin for the derailleur high or low and begin decrementing the motor steps every falling edge. Once the number of steps equals zero, the software will clear the timer, disable the interrupt and return to the main program as the derailleur is now in the correct position.

8.2.3 Automatic Gear Changing Algorithm

The following flowchart in figure 8.6 outlines the algorithm that will be used for the automatic shifting for the bicycle. Providing we are in the automatic shifting mode and that there is no shift being executed currently, we will check if a shift is required each time we receive information from the cadence sensor.

We wait for a user-defined number of pedal strokes (delay) with RPM outside of the window before executing a shift. As soon as the RPM goes outside the window the CHANGE label is shown to indicate that a shift is about to occur.

For the first implementation the controller will simply shift one gear if the user is outside the RPM window. The RPM window consists of a user-adjustable center position and window widths, which are used to calculate the upper and lower RPM window limits. Therefore, if the measured RPM is greater than the upper limit after the required number of pedal strokes we will shift one gear harder by determining the correct direction and number of steps and calling the derailleur move function. The system will shift to an easier gear if the RPM is less than the lower limit.

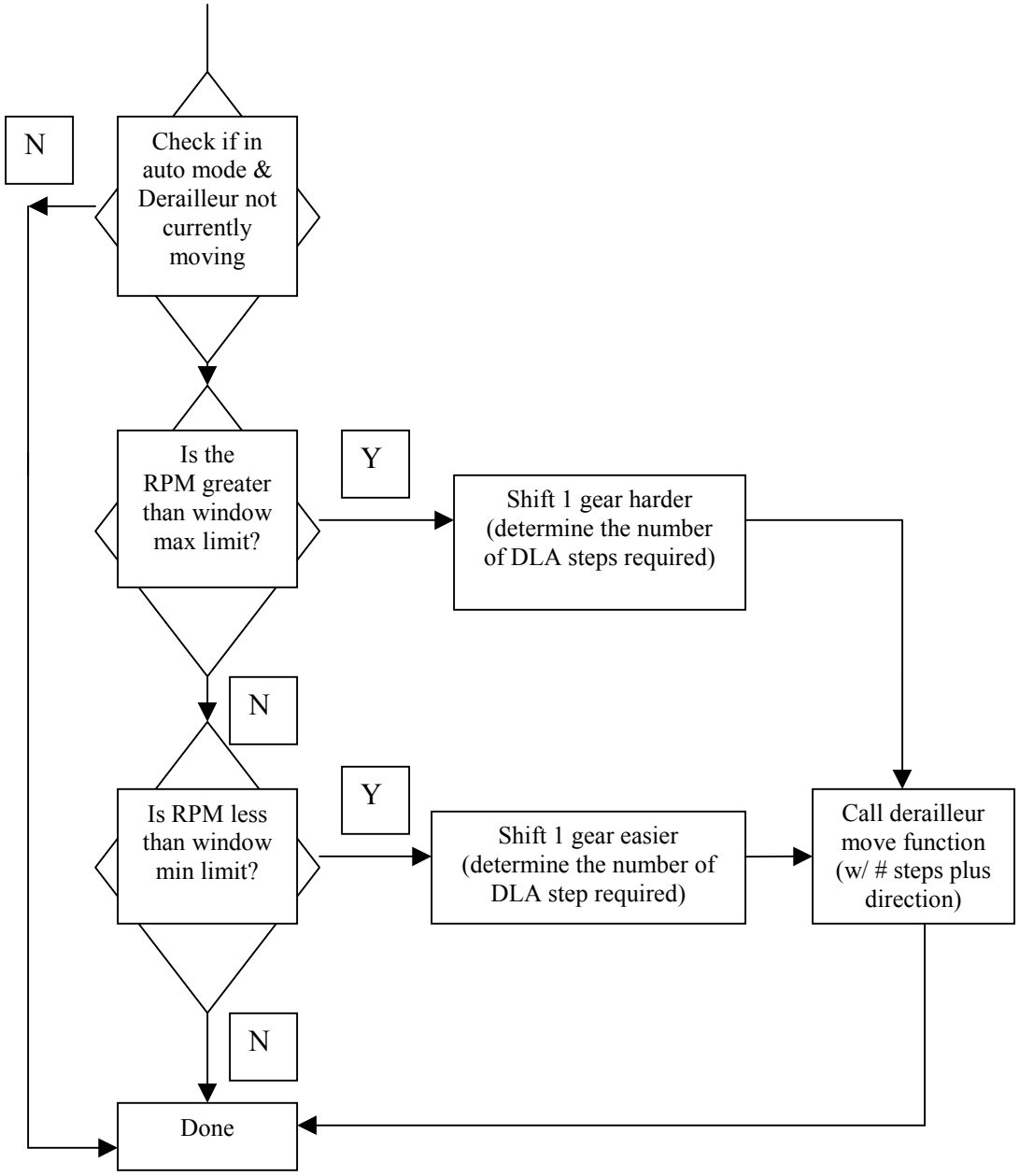


Figure 8.6: Automatic Gear Change Algorithm

8.2.4 UI Algorithm

The following flowchart in figure 8.7 outlines the algorithm that will be used for the UI. The UI code is called by an input capture interrupt which occurs whenever any button is pressed. After the handler is called, we read the correct port (not A) to get the data for all the buttons. We then determine the correct task based on the current mode/menu.

These tasks performed by the buttons all require data to be set and/or the display to be updated. Inside the interrupt handler we will not do very much, only set the required data and options and update the menu and display flags. The code that actually updates the code is called in the main program logic, when it sees that the update_display flag is set.

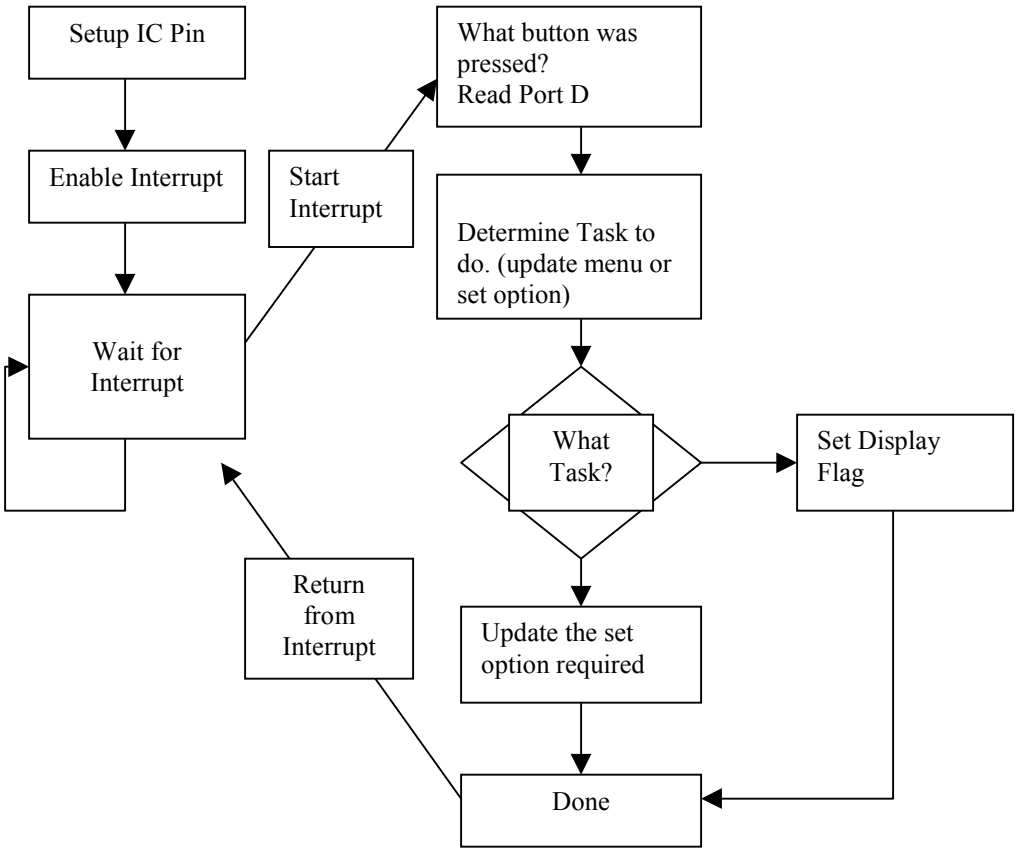


Figure 8.7: UI Flowchart

9. Conclusion

In this document, we have outlined the designs that we plan on implementing on our working prototype. The finer details will be worked out through the progress of project. We have already chosen and ordered nearly all of our required components for the project, as well as begun work on the UI. The integration process will begin on schedule and we expect only minor complications. Our extensive planning and designing has allowed us to foresee any design issues before they occur, so we can design around them. Once the hardware and software is completed we plan on using our hardware and software test plans to verify the functionality and performance of our SmartShifter™ System. The budget so far is $200 dollars less than the $700 dollars we originally expected. We don't expect anymore major expenditures, so the project might be cheaper than originally expected. Overall, we are very confident we will have a working prototype by April 2003.