A Mechatronically Measurable Double Pendulum for Machine Interval Experiments

Piers Lawrence[†], Michael Stuart[†], Richard Brown[†],
Warwick Tucker[‡] and Raazesh Sainudiin[†*]

Department of Mathematics and Statistics[†],
University of Canterbury,
Private Bag 4800, Christchurch, New Zealand

Theoretical Statistics and Mathematics Unit[*],
Indian Statistical Institute,
8th Mile, Mysore Road, RVCE Post, Bangalore 560 059, India
Tel.: +91-80-2848 3002 ext 479, Fax: +91-80-2848 4265

Department of Mathematics[‡], Uppsala University,
Box 480, 751 06 Uppsala Sweden

isibang/ms/2010/11, October 25, 2010

[*]Corresponding Author: `r.sainudiin@math.canterbury.ac.nz`

The double pendulum is an example of dynamical system that exhibits chaotic dynamics. Whilst there have been many papers describing the theoretical aspects of chaos, there have been few that examine these theories rigorously on a physical device. It is the purpose of this report to outline the development of a physical double pendulum system from which discrete measurements of the enclosures of the angular positions of the arms can be made. We describe the electronic system that transfers the measurement data to a host PC and the host software to store the measurements for further analysis using enclosure methods. Our device is a prerequisite for rigourous parameter estimation in machine interval experiments using techniques from computer-aided proofs in analysis.

Contents

1. Motivation

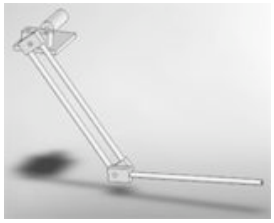
Enclosure methods that rely on machine interval arithmetic — validated computer arithmetic that encloses or bounds all numerical errors — have become an important tool in computer-aided proofs in analysis. Some examples where these methods have been applied include proofs of the Feigenbaum conjectures [1], the double bubble conjecture [2], the existence of the Lorenz attractor [3] and the Kepler conjecture [4]. However, computer-aided proofs have rarely been applied to validate heuristic solutions to challenging decision problems in statistics. The aim of this project is to be able to apply enclosure methods to address the challenging statistical decision problem of rigorous parameter estimation in a chaotic statistical experiment with the double pendulum.

Chaotic or complex non-linear systems pose some of the most challenging decision problems in engineering science. One of the simplest systems of this type is the **double pendulum (DP)**. The DP exhibits rich dynamics [5] and chaos at certain energies [6], thus making it challenging to model and measure for parameter estimation [7, 8].

Rigorous parameter estimation in such systems must account for the physical limits of the sensors' empirical resolution and the computer's numerical resolution. Past experiments with DP systems [7, 8, 9] were not rigorous; they neither enclosed the uncertainty in DP's angular positions nor employed validated numerical methods. Recent work on validated parameter estimation is largely limited to simulated data[10, 11].

Thus, our objective here is to design a measurable DP (Fig. 1A) that successfully accounts for the limit of the sensors' empirical resolution. The proposed project will produce data enclosures of the angular position of each arm through time (Fig. 1C) of our custom-built DP (Fig. 1B). Our device is a prerequisite for set-valued extensions of classical decision-theoretic properties of rigourous parameter estimators [12], such as *identifiability*, *consistency* and *efficiency*, over machine-representable filtrations using the formalism of machine interval experiments [13].

A: DP Schematic



B: Streaming DP data

C: Enclosures of two initially close trajectories

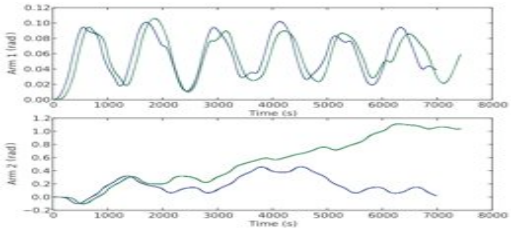


Figure 1: Double Pendulum

2. Mechanical Design Summary

2.1. The Pendulum

In designing the pendulum, the main objective was to keep the design simple so that it could be easily replicated, for example, in a high school or university workshop. The pendulum was designed to be modular, so that changes to the characteristic parameters of the system could be made with relative ease.

The characteristic parameters of the double pendulum system are:

- Centre of mass of each arm
- Moment of inertia of each arm
- Mass of each arm
- Length of the inner arm

The first three parameters depend on the shape of the pendulum and not on the scale. So we chose to fix the length of the arm, and hence the scale of the pendulum. The

remaining parameters can easily be varied by the addition of external masses to the base pendulum system.

An important consideration in the design relating to the inertia and centre of mass is the geometry of the pendulum. The geometry should be kept as homogeneous and symmetric as possible to simplify the recalculation of the parameters as masses are added to the system.

Friction was another consideration that was taken into account when designing the system. The main contributor to friction is likely to be the joints. This friction cannot be eliminated, but kept minimal by the use of low friction ball bearings. In Section 3 we describe the physical measurements.

3. Data Acquisition

The primary purpose of the pendulum is to be able to record the angular positions of each arm within some known limit. An electronic data acquisition system was designed to record the the movements of the double pendulum in real time. This section gives a broad description of the system, technical details may be found in Appendices B and C.

3.1. Position Sensors

Optical encoders are the standard means of precisely measuring angular position. Optical encoders are essentially a slotted disk with an LED illuminating one side and two photo-diodes detecting the light shining through the slots in the disk. When there is a transition from light to dark or dark to light, the transition is converted to a binary signal by the photo-diodes. Counting the transitions allows the angular position to be determined. The two photo-diodes are arranged to be 90° out of phase with respect to the binary signal that is being generated. When there is a change in direction one of the channels will fall behind the other, allowing the direction to be determined. This is known as quadrature encoding as if both the rising and falling edges of the signal for both channels are used, the resolution of the encoder wheel is increased by a factor of four.

For our device we used Avago HEDS-5505 optical encoders modules [14], these modules have 2000 slots per revolution. When combined with quadrature encoding this gives a resolution of 8000 measurable transitions which gives an angular resolution of approximately 0.045°. Two optical encoders were used, one attached at each joint of the double pendulum.

3.2. Electrical Measurement

Measurements are made using an electronic device whose main components are an FPGA (field programmable gate array) and a microcontroller. Fig. 2 shows a block diagram of the systems encompassed by the device.

The optical encoders output an incremental signal which must be converted into an integer position measurement. To do this we use an FPGA to filter and sample the incoming signal from the encoders. An FPGA is used for the following reasons:

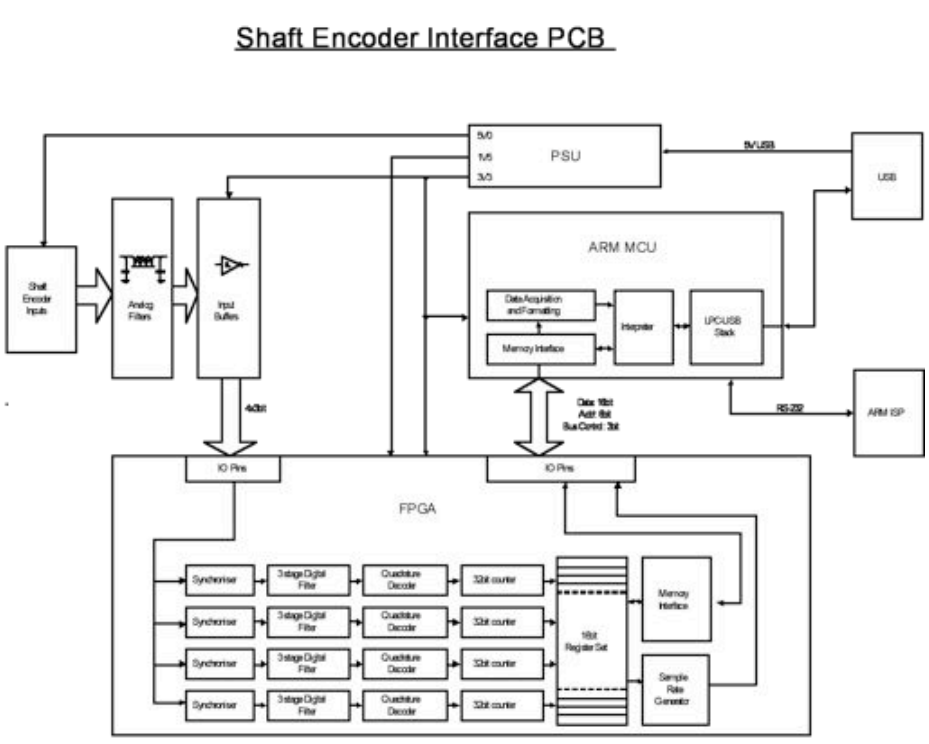


Figure 2: Block Diagram of Data Acquisition PCB

- True simultaneous sampling of multiple digital sources (in this case optical encoder outputs)
- High speed sampling of all channels possible (up to tens of Mbps)
- Very low latency digital glitch filtering is customizable on all input channels
- Ability to be reprogrammed makes the platform flexible - possibly useful for future projects or customizable for other types of sensors

The encoder signals are filtered, then sampled at a user definable rate of between 1 to 160kHz. 160kHz is the required sample rate to catch every transition if the pendulum is rotating at 20 revolutions per second, which is a considerable overestimate of the pendulum speed under normal conditions.

The FPGA updates internal counters whenever an increment or decrement event occurs, these counts occur independently of the sampling. A request is sent to the microcontroller to read the new count from a 16 bit parallel bus every sample period.

3.3. Data Transfer

To utilise the measurement data, it needs to be easily tranferred to a PC, ideally in real time. The primary consideration for the transfer of data is the speed at which it can

be sent to the host PC. For this reason the microcontroller was chosen to have *USB* (Universal Serial Bus) capabilities. In Full-Speed mode, USB supports a data transfer rate of 12Mbps, that should be well within the physical capability to produce data. USB is also desirable as it is a standard data connection supported by every personal computer, and hence makes the system portable and easily demonstrated.

The microcontroller reads acquired data from the FPGA, formats the data according to user-specified rules and then sends the data to a PC application via USB.

3.4. PC Data Logger

The PC logger application was written in python. The main features of the application are capture of data from the data acquistsion hardware, and logging of that data to a file for later use. A major limitaion was encountered during developmemnt of the logger application. The mode of data transport chosen was via a CDC class driver or virtual serial port. Although the speed at which these ports can communicate is approximately 6Mbps, it was found that the logger application could not process the incoming data fast enough, resulting in buffer overruns and consequent loss of data when sampling at speeds above 100kHz. When used with a sample rate of 50kHz it has been demonstrated that data may be stored without discontinuities, and with only minor loss of precision.

A major enhancement to the project will be the reimplementation of the logger application in C or C++ with the utilisation of raw USB bulk data transfer. It is estimated that this would comfortably extend the usable bandwidth of the data acquisition system to the estimated required sample rate of 160kHz or better.

4. Construction

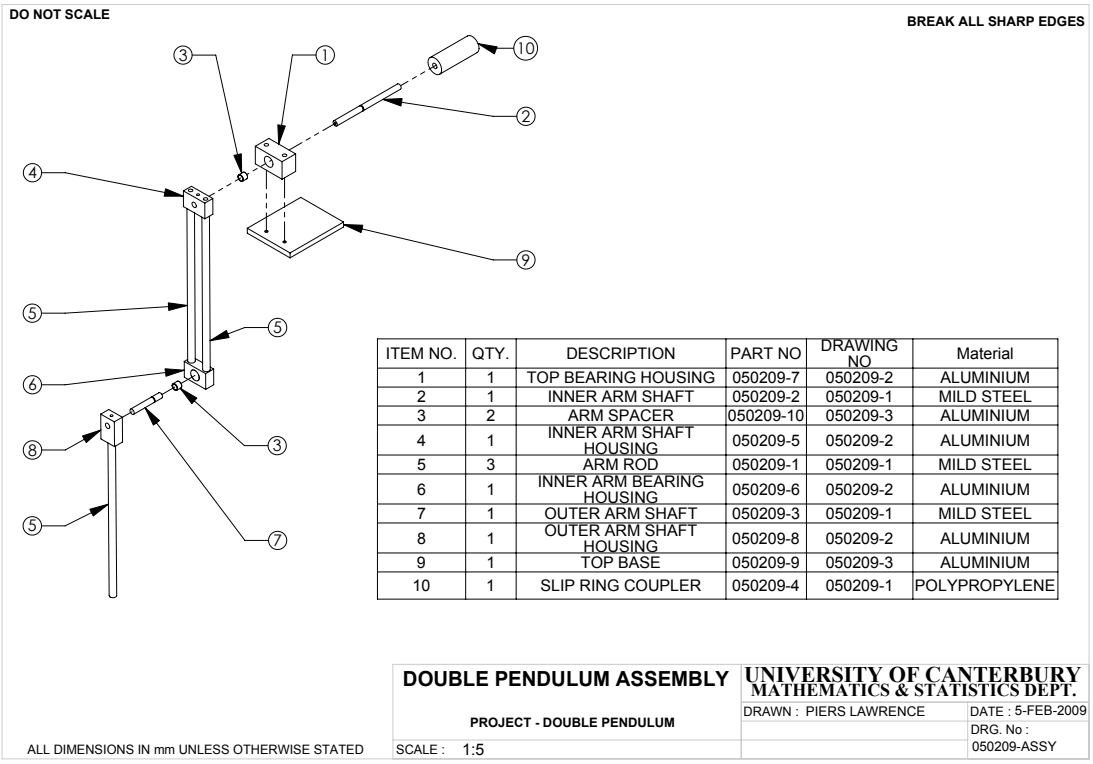
The full technical drawings of the pendulum are provided in Appendix A.

The main design considerations of the pendulum were,

- Ease of construction
- Modularity
- Adjustable mass
- Adjustable centre of mass
- Adjustable arm lengths

As discussed in Section 2, the scale of the pendulum was fixed with the distance between shaft centres of 275.4mm. The two arm lengths are approximately equal, however the inner arm has twice the mass the outer arm. The mass of each arm can be controlled by adding external brass masses to each arm. This affects both the mass and centre of mass of each arm.

The arms of the pendulum were made out of mild steel rod. This has a number of advantages, external masses can easily be added and moved on the rods to change the



ITEM NO.	QTY.	DESCRIPTION	PART NO	DRAWING NO	Material
1	1	TOP BEARING HOUSING	050209-7	050209-2	ALUMINIUM
2	1	INNER ARM SHAFT	050209-2	050209-1	MILD STEEL
3	2	ARM SPACER	050209-10	050209-3	ALUMINIUM
4	1	INNER ARM SHAFT HOUSING	050209-5	050209-2	ALUMINIUM
5	3	ARM ROD	050209-1	050209-1	MILD STEEL
6	1	INNER ARM BEARING HOUSING	050209-6	050209-2	ALUMINIUM
7	1	OUTER ARM SHAFT	050209-3	050209-1	MILD STEEL
8	1	OUTER ARM SHAFT HOUSING	050209-8	050209-2	ALUMINIUM
9	1	TOP BASE	050209-9	050209-3	ALUMINIUM
10	1	SLIP RING COUPLER	050209-4	050209-1	POLYPROPYLENE

Figure 3: Double Pendulum Exploded Isometric

mass and centre of mass of the arm, and mild steel does not dent when external masses are clamped to it. The mass, and hence initial energy, of the pendulum system was also a factor in this design decision. A preliminary design with aluminium rods was too lightweight to give lengthy runs of the pendulum.

Aluminium was used for the bearing and shaft housings. The holes for the bearings are an interference fit, so that no other clamping fittings were needed to keep the bearings in place.

Fig. 3 shows an exploded isometric view of the pendulum, without the bearings, encoders or fasteners. The encoders are fitted onto the back of the two bearing housings.

4.1. Assembly

4.1.1. Outer arm

The outer arm is assembled by screwing the $M6$ stud into one end of the arm rod then screwing this into the outer arm shaft housing. An $M6$ bolt can be screwed into the other end of the rod to constrain any additional masses added from flying off under operation, minimising any potential hazard.

4.1.2. Inner arm

Preparation of the inner arm bearing housing is essential before the arm is constructed. The bearings are aligned on an 8mm shaft with the bearings either side of the bearing housing, this is then squeezed in a machine vice to drive the bearings into the housing. The bearing on the encoder side of the bearing housing is then pushed 1mm further into the housing using an aluminium rod, taking care to keep the two bearings aligned on the 8mm shaft.

The outer arm shaft is then passed through the bearings, adding the circlip to constrain the shaft. The encoder backing plate is located on the housing using an alignment tool, securing this to the bearing housing. The encoder can now be clipped onto the backing plate and the encoder wheel fixed to the outer arm shaft with the grub screw.

The inner arm is assembled by screwing $M6$ studs into each end of both arm rods then screwing both of them into the bearing housing. The shaft housing is then dropped over the top of the studs fixing the shaft housing in place with two $M6$ nuts.

4.1.3. Top bearing housing

The bearings and encoder are installed using the same method as for the inner arm. The top bearing housing can then be affixed to the base plate using $M6$ bolts. The slip ring coupler can then be slid onto the shaft, this fit should be tight and may require some force.

4.1.4. Final assembly

The arms are fixed to their respective shafts with $M4$ cap screws, making sure to install the spacers first on each shaft to offset the arms. The slip ring coupler is then located on the top shaft and pushed onto the shaft, this should be a tight fit. The wiring and Mercotac connector can then be installed onto the encoders.

5. Data

The beginning of a data file is shown below. It is a lossless compression for our problem. More formally, the data file is a minimal sufficient statistic of the trajectory data as it only reports the measurable discrete state transitions along with the transition time. All recordings of time stamps, arm-position states are done with integers representing intervals. The plots of the trajectory for the data is shown in Fig. 4.

```
#----------Log Parameters----------
#samplerate = 10kHz
#number of samples = 200000
#Packet length = 6
#diagnostic = Off
#bitdepth = 16 bit
#rle = On
```

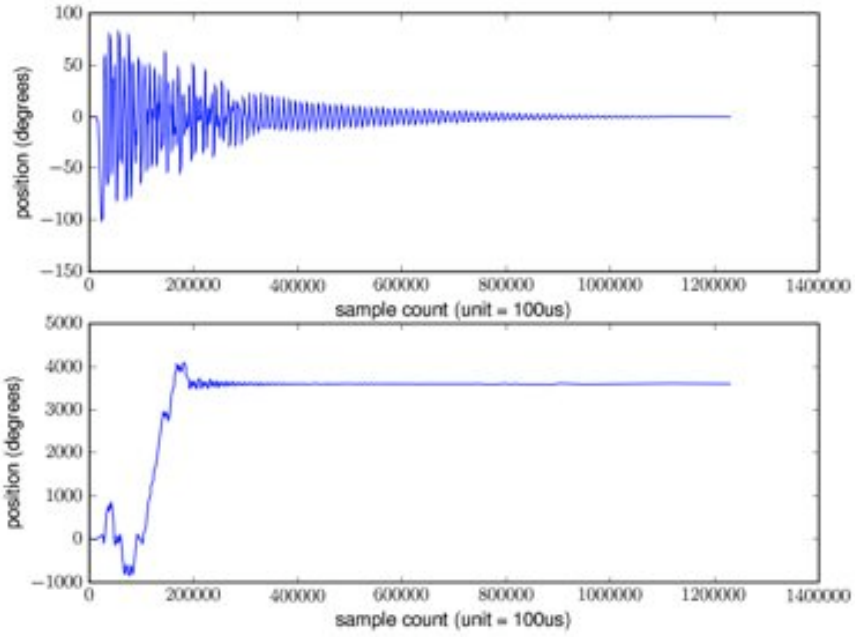


Figure 4: Trajectories of the two arms for the above data file.

```
#---------------------------------
#
##
#sample number, encoder1, encoder2
26 0 0
1042 -1 0
1578 -1 -1
6752 -2 -1
8091 -1 -1
8836 -1 0
10764 0 0
.
.
.
1218453 -1 20480
1222243 -2 20480
1229330 -1 20480
```

6. Manual

This section describes the python scripts `PyLogger` and `PyPlot`. It is intended as a user guide, to enable use of `PyLogger`, `PyPlot` and setup of the double penulum apparatus by those unfamiliar with the system.

6.1. Setup

This section describes the setup of the dp and connection to a PC.

6.1.1. Hardware

The base plate of the dp must be clamped onto a surface that is rigid and strong. The action of swinging the arms produces a large amount of force, if the clamp is fixed loosely then there is some danger that the pendulum will twist and the arms if moving at high speed, could cause damage to objects that come into contact. Be sure to stand clear of the arms when they are spinning as they can move in an unpredictable ways, getting hit by them may cause some pain.
The kit contains an electrical cable, on one end of this cable are two plugs. Both must be attached to the dp, one plugs directly into the black rotary encoder mounted on the base plate, and the other into the end of the shaft which protrudes from the rear of the base plate.
The other end of the cable is plugged into the corresponding socket on the data logger. Once this has been done, the data logger may be plugged into the PC via the USB cable included in the kit.

6.1.2. Driver Installation

If the host PC is running Linux, or MacOS then no driver is required. If the host PC is running MS Windows, then upon plugging in the dp to a USB port, a prompt will appear guiding the user to install a driver. Navigate to the point where the wizard asks if you want to install the driver from a specific location, and then browse for the file "usbser.inf".
The dp mounts as a CDC or communications device class com-port, otherwise known as a virtual com-port. The dp will appear under some unique name; unfortunatlya different name depending on what operating system the host PC runs. The name will be one of the following.

- Windows: "COMx" where x might be any number from 0 to 255.
- Linux: "/dev/ttyACMx" where x might be any number from 0 to 255, and is usually 0.
- MacOS: "/dev/tty.usbmodemDEADC0DE1".

PyLogger detects the data logger by means of these identifiers, if the device mounts under a different identifier, then the script will report that the data logger was not found and abort.

6.2. PyLogger Usage

This section describes usage of PyLogger. Before Pylogger is used, be sure to have connected the data logger to the host PC.
PyLogger is a script that employs an interactive user interface, no command line operation has been implemented yet. The user first enters setup data according to prompts, then PyLogger is ready to capture data from the data logger device.

6.2.1. PyLogger parameters

PyLogger requires setup before use. The operating system is first specified by entering one of the following.

- 'w' for Windows
- 'l' for Linux
- 'm' for MacOS

Following OS selection, the choice to setup all parameters or load defaults is offered. The default settings are sufficient for most purposes and may be loaded by pressing 'd'. The default settings are:

- samplerate = 100kHz
- number of samples = 100000
- Packet length = 3
- diagnostic = Off
- bitdepth = 8 bit
- run length encoding (rle) = On

Alternatively, to set up manually, press 's'. The following options are availible.

Bitdepth:

- '1' for 32b. Data and count are recieved as 32bit words. The packet length is 12 bytes.
- '2' for 16b. Data and count are recieved as 16bit words. The packet length is 6 bytes.
- '3' for 8b. Data and count are recieved as 8bit words. The packet length is 3 bytes.

- '4' for 16/8b.

Data is recieved as 8bit words and count is recieved as a 16bit word. The packet length is 4 bytes. Item 3, 8 bit data is usually the best choice as the packet format is very compact and this allows higher sample rates to be used without risk of buffer overflow.

Run Length Encoding:

- '1' for on.
- '0' for off.

Run length encoding is usually just left enabled as it compresses the data to a much smaller size. This also allows higher sample rates to be used without risk of buffer overflow.

Sample Count:

- enter the number of samples to be captured before quitting.

The number of samples to be taken might be set by the user after some experimentation. If the sample count is too low, the logging process will halt before the arms have stopped moving. If too long, then the data logger will time out when the arms have stoped moving for a certain time period.

Sample Rate:

- '1' for 1kHz.
- '2' for 10kHz.
- '3' for 50kHz.
- '4' for 100kHz.
- '5' for 150kHz.

The sample rate can be set low, to minimise the chance of a buffer overflow. To obtain finer data granularity, a higher setting can be used. It is usually safe to use 16bit data mode with only 10kHz sample rate, while 8bit mode allows use of 100kHz sample rate. It depends Also on the speed at which the dp arms rotate - very high speed rotation may require the sample rate to be lowered in order to avoid buffer overflow.

Log Filename:

- enter a name for the log file.

Any file name name may be specified.

Log File data delimiter:

- '0' for comma.
- '1' for tab.
- '2' for space.

The data delimiter is the separator between logged data points. If the log file is to be used with Microsoft Excel, then commas should be selected. If `PyPlot` is to be used, then spaces should be selected.

6.2.2. Making a Log

Once setup has finished, the user is prompted to press enter to start logging.
Before starting the dp should be mounted securely and the arms left untill they are still. This ensures that when the logger is started, the arms are resting in a known reference position.
Press enter to start. After this has been done, the user has 2 seconds to move the arms to the desired position, and either let them go or give them a shove. After either the number of samples specified in the setup have been captured, or the arms have been staionary for 2 continuous seconds, the logging process will stop and post processing of the data begins. Finally the data is written to the file of the name specified.

6.2.3. Log File Format

The log file will contain the following formatted data (the sample below is truncated after the second sample for brevity).

```
#
#----------Log Parameters----------
#samplerate = 100kHz
#number of samples = 100000
#Packet length = 3
#diagnostic = Off
#bitdepth = 8 bit
#rle = On
#----------------------------------
#
##
#sample number, encoder1, encoder2
0 0 0
26 1 -1
...
...
etc
```

Every line that does not begin with a # symbol is regarded as a data sample. The first number is the sample count, which indicates when the sample happened in time.

The sample count is in units of 1/Fs seconds where Fs is the sample rate specified. The second and third numbers on the line indicate the absolute position of the arms at the time given by the sample count. For example, in the sample log above:

- Unit time = 1/100kHz or 10us.
- At time 0us, arms 1 and 2 were at 0 and 0 respectively.
- At time 260us, arms 1 and 2 had moved to positions 1 and -1 respectively.

6.3. `PyPlot`

`PyPlot` is a utility that reads the content of a file created by `PyLogger` and plots the data contained in it graphically.

6.3.1. Plotting a Log

To plot a log, run the `PyPlot` script. The user is prompted for a filename or path, and upon a press of the enter key, the plot will appear.

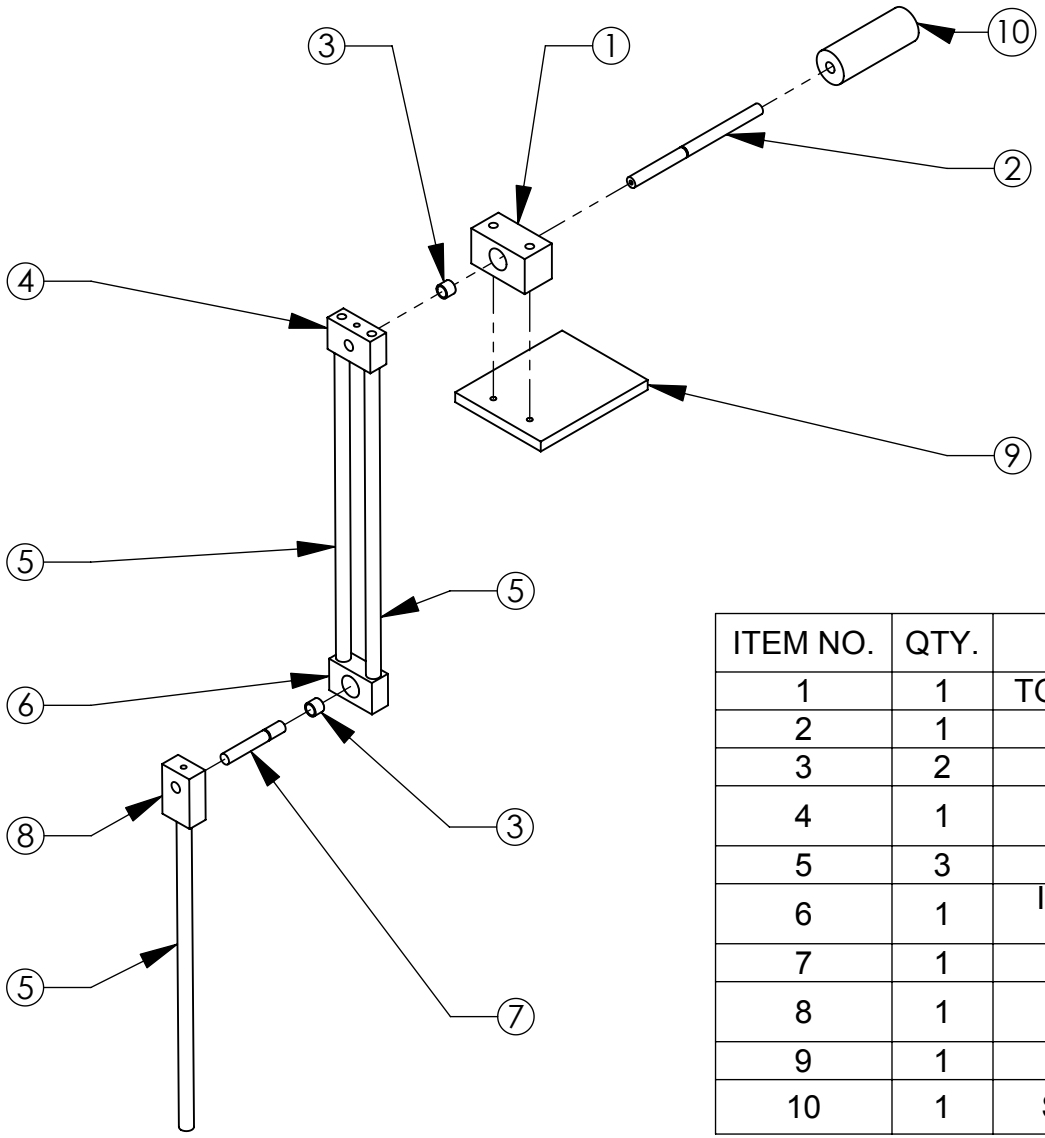
The following notes may be helpful.

- All non data entries in the file must be receeded by a # symbol.
- Sample data must be separated by spaces, tabs or commas.
- Samples must be separed by newlines.
- samples must be made of 3 columns: col1=sample number, col2= encoder 1 data, col3= encoder 2 data

A. Mechanical Drawings

DO NOT SCALE

BREAK ALL SHARP EDGES



ITEM NO.	QTY.	DESCRIPTION	PART NO	DRAWING NO	Material
1	1	TOP BEARING HOUSING	050209-7	050209-2	ALUMINIUM
2	1	INNER ARM SHAFT	050209-2	050209-1	MILD STEEL
3	2	ARM SPACER	050209-10	050209-3	ALUMINIUM
4	1	INNER ARM SHAFT HOUSING	050209-5	050209-2	ALUMINIUM
5	3	ARM ROD	050209-1	050209-1	MILD STEEL
6	1	INNER ARM BEARING HOUSING	050209-6	050209-2	ALUMINIUM
7	1	OUTER ARM SHAFT	050209-3	050209-1	MILD STEEL
8	1	OUTER ARM SHAFT HOUSING	050209-8	050209-2	ALUMINIUM
9	1	TOP BASE	050209-9	050209-3	ALUMINIUM
10	1	SLIP RING COUPLER	050209-4	050209-1	POLYPROPYLENE

DOUBLE PENDULUM ASSEMBLY

PROJECT - DOUBLE PENDULUM

SCALE : 1:5

UNIVERSITY OF CANTERBURY
MATHEMATICS & STATISTICS DEPT.

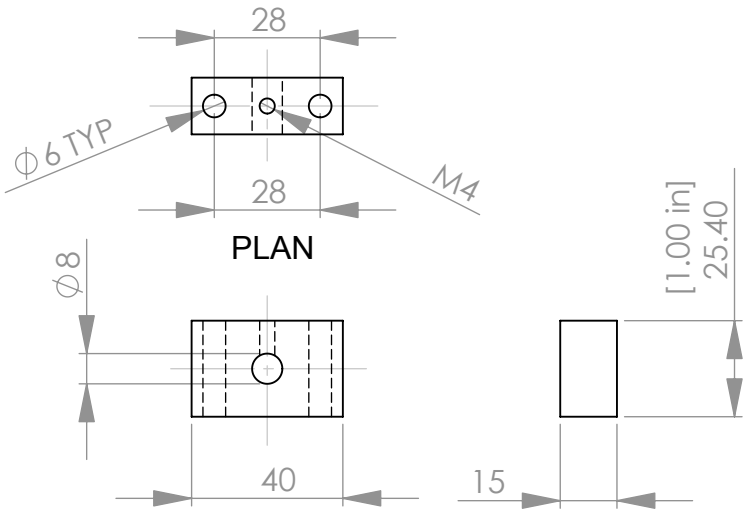
DRAWN : PIERS LAWRENCE

DATE : 5-FEB-2009

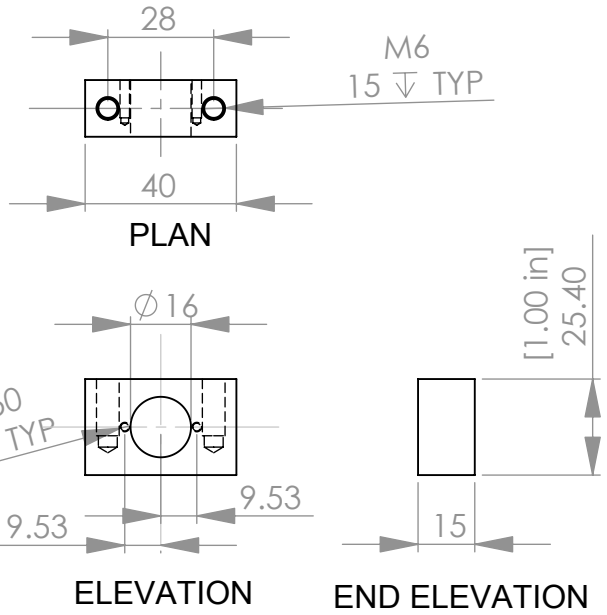
DRG. No : 050209-ASSY

ALL DIMENSIONS IN mm UNLESS OTHERWISE STATED

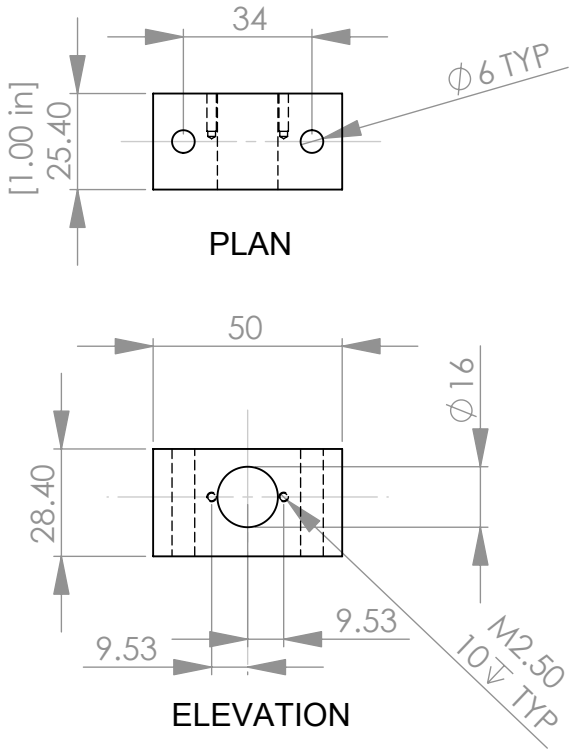
DO NOT SCALE



INNER ARM SHAFT HOUSING -
PART NO 050209-5

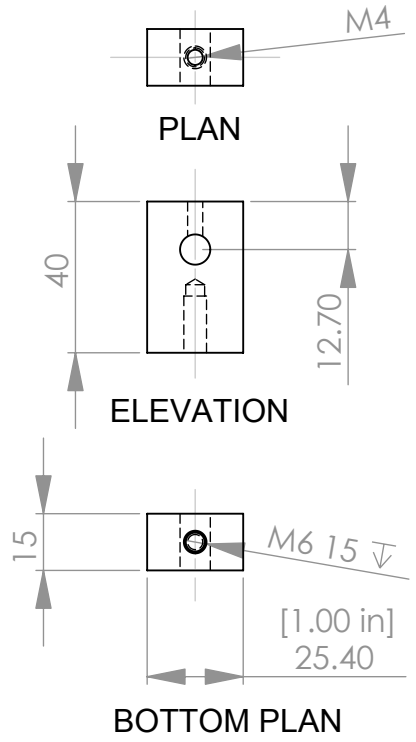


INNER ARM BEARING HOUSING -
PART NO 050209-6



TOP BEARING HOUSING -
PART NO 050209-7

BREAK ALL SHARP EDGES



OUTER ARM SHAFT HOUSING
PART NO - 050209-8

ALL DIMENSIONS IN mm UNLESS OTHERWISE STATED

HOUSINGS

PROJECT - DOUBLE PENDULUM

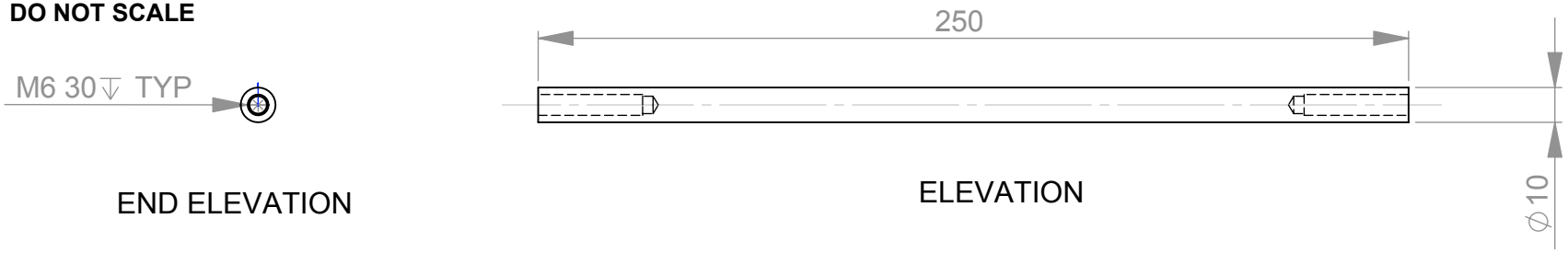
SCALE : 1:2 UNLESS OTHERWISE STATED

UNIVERSITY OF CANTERBURY
MATHEMATICS & STATISTICS DEPT.

DRAWN : PIERS LAWRENCE	DATE : 5-FEB-2009
	DRG. No : 050209-2

DO NOT SCALE

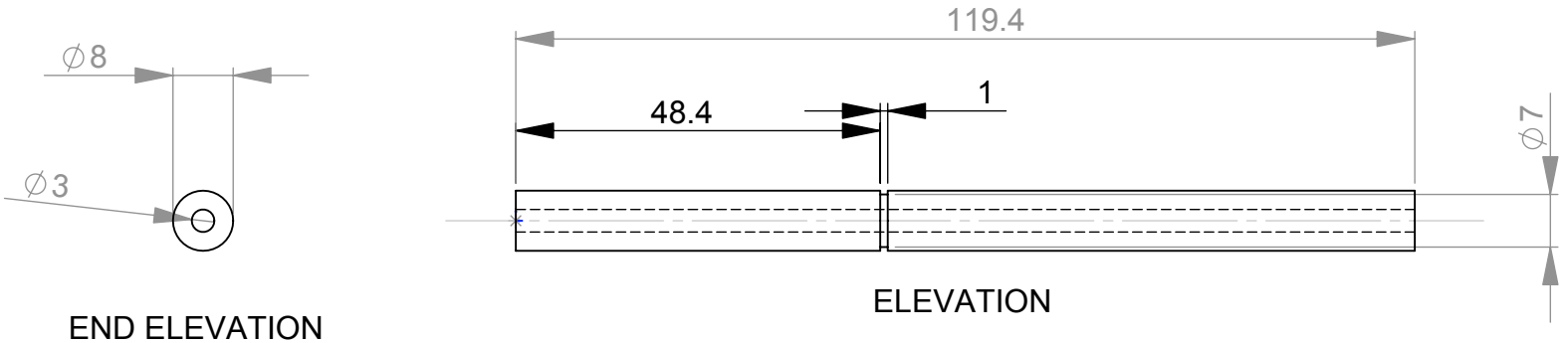
BREAK ALL SHARP EDGES



END ELEVATION

ELEVATION

ARM ROD - PART NO 050209-1

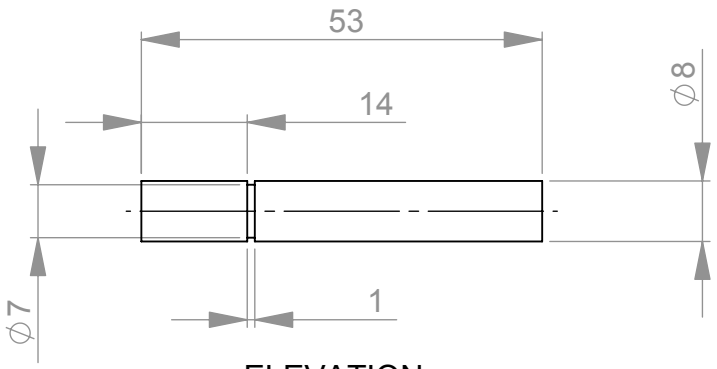


END ELEVATION

ELEVATION

INNER ARM SHAFT - PART NO 050209-2

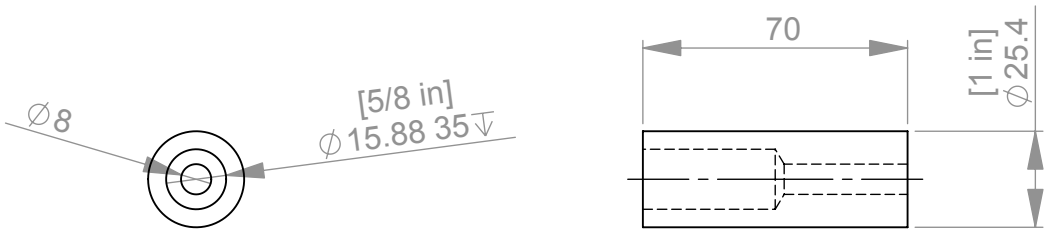
SCALE: 1:1



ELEVATION

OUTER ARM SHAFT - PART NO 050209-3

SCALE: 1:1



END ELEVATION

ELEVATION

SLIP RING COUPLER - PART NO 050209-4

ALL DIMENSIONS IN mm UNLESS OTHERWISE STATED

SHAFTS

PROJECT - DOUBLE PENDULUM

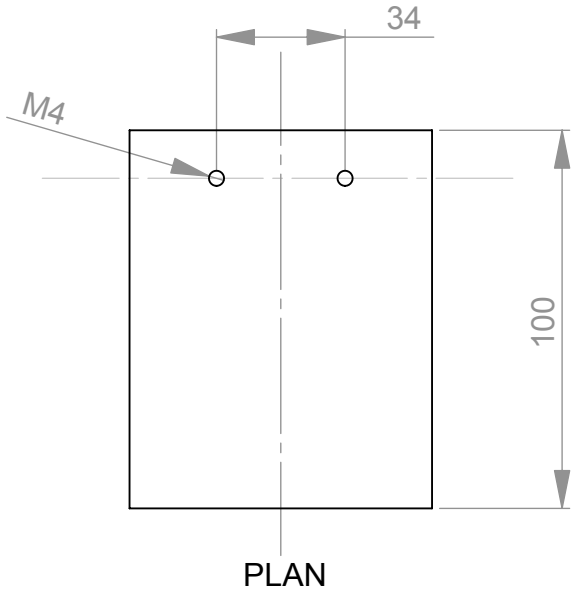
SCALE : 1:2 UNLESS OTHERWISE STATED

UNIVERSITY OF CANTERBURY
MATHEMATICS & STATISTICS DEPT.

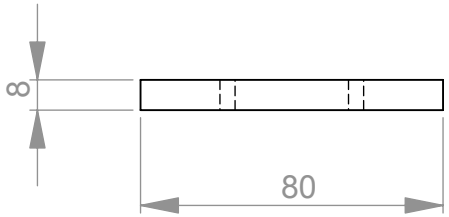
DRAWN : PIERS LAWRENCE	DATE : 5-FEB-2009
	DRG. No : 050209-1

DO NOT SCALE

BREAK ALL SHARP EDGES

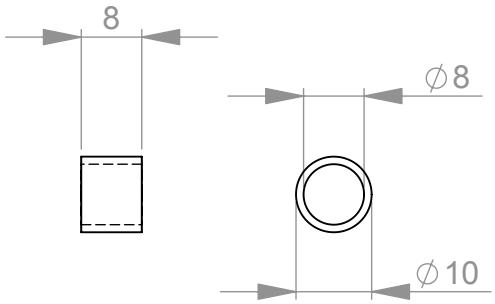


PLAN



ELEVATION

PENDULUM BASE
PART NO - 050209-9



PLAN ELEVATION

ARM SPACER
PART NO - 050209-10

SCALE: 1:1

ALL DIMENSIONS IN mm UNLESS OTHERWISE STATED

PENDULUM BASE

PROJECT - DOUBLE PENDULUM

SCALE : 1:2 UNLESS OTHERWISE STATED

UNIVERSITY OF CANTERBURY
MATHEMATICS & STATISTICS DEPT.

DRAWN : PIERS LAWRENCE	DATE : 5-FEB-2009
	DRG. No : 050209-3

B. Data Acquisition

Various technical details regarding the data acquisition hardware/software are described in this section. The data acquisition hardware and software will hereafter be referred to as the DAQ and pylogger respectively.

B.1. PCB Fabrication

The DAQ printed circuit board (PCB) was made to the following specifications:

- Two layers
- FR-4 substrate
- 0.2mmtrace and space width
- 0.5mm via hole size

The parts were obtained from various sources and the board was populated by hand. The PCB parts placement diagram is shown in Fig. 5 and the PCB dimensions are shown in Fig. 6.

B.2. Usage

The DAQ is powered by either USB or by an external souce in the range of 5-14VDC. On board regulators provide stable supplies to the various parts of the system:

- 5V - powers the optical encoders.
- 3.3V - powers the FPGA IO pins and the ARM microcontroller.
- 1.5V - powers the FPGA core.

NOTE: when USB power is to be used, fit R19, remove R17. When external power is to be used, fit R17, remove R19. If this procedure is not followed, damage could occur to the DAQ or the optical encoders.

B.3. Optical Encoder inputs

There are four optical encoder inputs. These connect to optical encoder devices that have standard 2bit Gray Code output via molex 6 way single in-line (0.1" pitch) locking connectors. The pin connections of these are shown in Table 1.

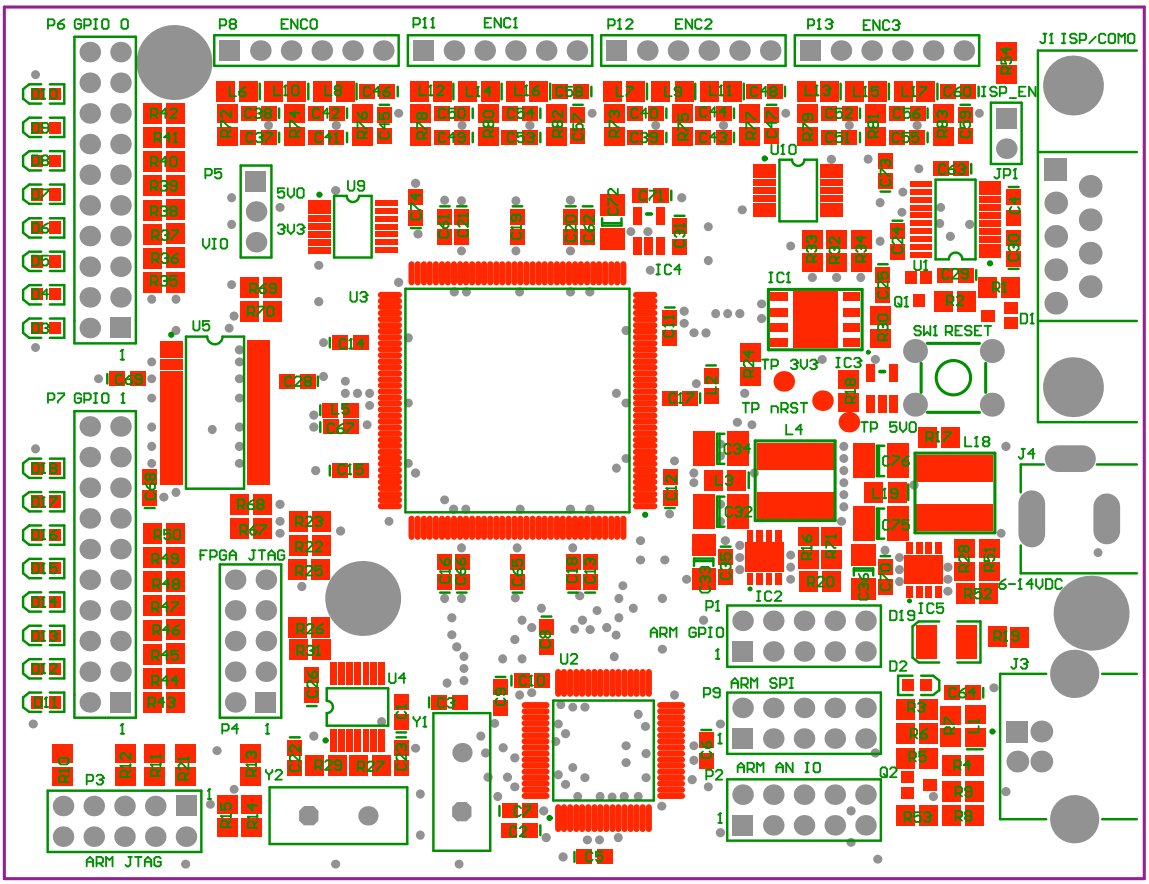


Figure 5: Parts Placement Diagram

Signal Name	Pin	Description
A	1	Gray code bit A input
B	2	Gray code bit B input
R	3	Reference input (not used)
NC	4	No connection
VCC	5	+5V
GND	6	0V

Table 1: Optical Encoder Input Connector Pinout

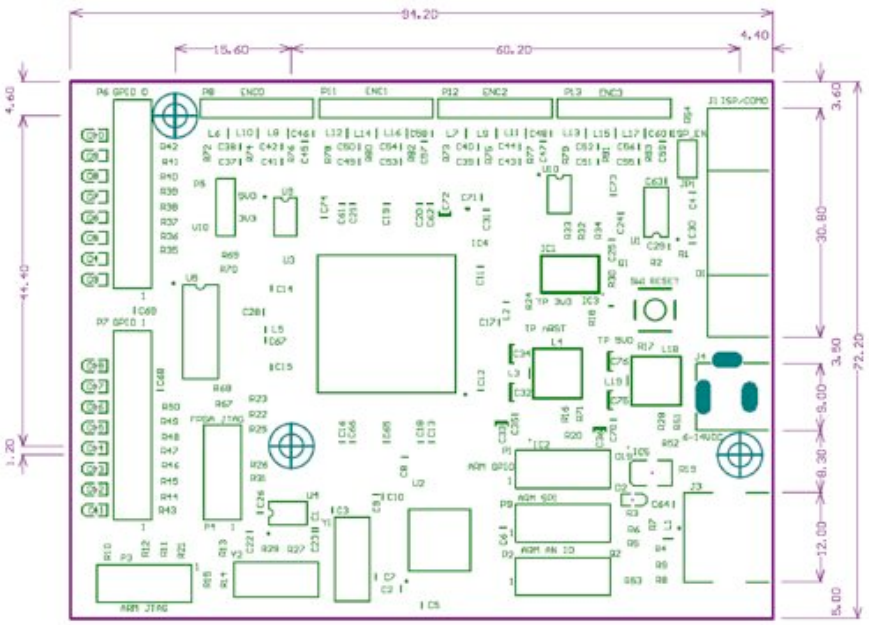


Figure 6: PCB Dimensions

B.4. FPGA System Description

The FPGA device used is an Altera EP1C6T144 cyclone series device, operating at FCLK = 12MHz. This part is capable of operating at frequencies of 50-100MHz depending on the complexity of the design. The design currently utilizes around 9% of the logic resource on the chip and runs at a rate of 12MHz (FCLK). All of the circuitry for the FPGA was written in VHDL and synthesised using Altera Quartus II web edition software.

Once the signals enter the FPGA, they pass through several stages; these are synchronization, decoders, counters, and finally data IO.

B.4.1. Synchronization

The signals are first synchronized to the FPGA clock to eliminate issues of meta-stability as the signal propagates through the system. This is normally achieved by sampling the signal on the rising or falling edge of the system clock and using the result of the sample to represent the state of the input. Every input to the FPGA must be synchronized therefore there are 12 implemented, one for each A, B and R signal for 4 shaft encoders. Since the input signals to this system are much slower than FCLK (around 200kHz absolute maximum), it is possible to improve on this by sampling the input more than once and registering a state change only when all samples are equal. This forms simple digital filter, signals with a frequency higher than 1/3 FCLK (4MHz) are rejected. This is not a perfect filter as input frequencies above FCLK can still cause aliasing. The

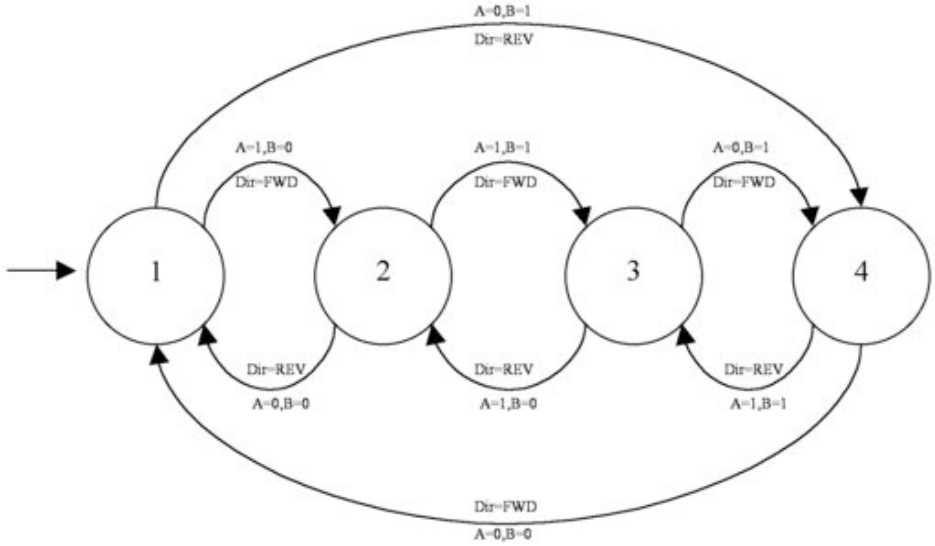


Figure 7: Quadrature Decoder State Machine

analogue filtering described previously reduces the probability of this occurring to a very low level.

B.4.2. Quadrature Decoders

The filtered and synchronized signals are then fed into the Quadrature decoders. In this design only the A and B signals are used, the R (reference) signal is ignored. A and B are square wave signals of the same frequency, but out of phase by 90°. When the shaft encoder rotates in one direction, B leads A by 90°, when the shaft encoder rotates in the other direction B lags A by 90°. The frequency of A and B is proportional to the angular velocity of the rotation, so it can be seen that the absolute position, speed and direction may be derived from the data captured from the signals. The decoder is a simple state machine that contains 4 states representing the 4 discrete states of A and B. Fig. 7 shows the state transition diagram of the decoder circuit. On every transition, an output signal called `count` is set from low to high and another called `up-down` is set high when the states are ascending or cleared low when the states are descending. These signals are used to control a position counter. Four Quadrature decoders were implemented, one for every set of shaft encoder inputs.

B.4.3. Position Counters

The position counters maintain a count value that represents the position of the shaft encoder. The counters are 32 bits wide and can count up or down, providing a count value from -2147483648 to + 2147483648. Negative counts represent rotation in one direction, while positive counts represent rotation in the other direction. The signal `up-down` from

the Quadrature decoder controls the count direction (increment or decrement), and the counter is incremented or decremented on every rising edge of the signal `count`.

The shaft encoders used in the experiment had 512 steps per encoder wheel. The Quadrature decoders decode 4 states per step therefore 2048 states existed per revolution of the shaft. This means that the position counter will count 2048 counts per revolution, this allows for up to 1048576 revolutions to be counted in either direction before the counter overflows. Because of the extreme resolution of these counters, no overflow detection capability was been implemented. Four position counters were implemented, one for every Quadrature decoder.

The position count, counter reset and counter enable signals are mapped into the register set. Because the register set is only 16 bits wide, the counter is accessible as two 16bit registers; counter-n position high & low (CNTnPRH and CNTnPRL). The reset and enable signals for all counters are accessed via the counter control (CNTCR) register.

B.4.4. Sample Rate Generator

The sample rate generator is a simple clock divider that provides a variable frequency signal on the external pin IRQ. The signal is synchronized with the sampling of the shaft encoder inputs and may be used by the host MCU as a reference or an interrupt request line. The frequency is set by the value in bits SCDIV[3..0] of the counter control register CNTCR. The sample rates available are listed in Table 2.

B.4.5. Register Set

The register set is a simple control interface for the various FPGA subsystems. The registers are described in Appendix C. The FPGA contains 64x16bit write/read only register pairs. Only 13 registers have implemented functionality at this time.

B.4.6. External Memory Interface

The EMIF provides access to the FPGA register set by an external MCU. The bus timing is shown in Fig. 9. The Data bus is 16 bits wide, while the address bus is 6 bits wide. The FPGA is configured as a slave, the host MCU must initiate and control all bus transactions.

B.5. ARM MCU

The ARM MCU is an NXP LPC2148 device. It is based on the ARM7TDMI 32bit core, running at 60MHz. The device has 512kB of on-chip flash and 40kB on-chip SRAM. The LPC2148 was chosen because of its relatively fast core, and its USB peripheral. It is capable of full speed USB, which is configured for this project in virtual com port (VCP, communications device class) mode. The firmware for this project was written using eclipse in conjunction with the Yagarto GNU tools, arm-elf-gcc, etc distribution. The

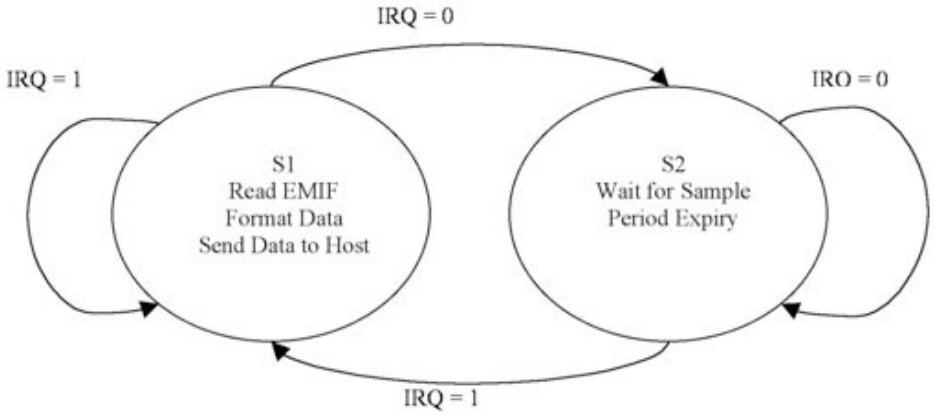


Figure 8: Data Acquisition State Machine

firmware running on the ARM MCU is based on the modules described in the following sections.

B.5.1. LPC-USB

The USB stack used in this project is based on that written by Bertrik Sikken LPC-USB. This module has been used in CDC mode and connects to a PC as a VCP. The VCP is a comparatively slow interface but has proven fast enough for the application and is a very easy interface to use at the PC end.

B.5.2. Interpreter

The interpreter receives and transmits commands/data to and from the PC via the USB module. The protocol for communication and data transfer is detailed in Appendix D. The Interpreter receives data from and transmits commands to the FPGA via the memory interface module.

B.5.3. Data Acquisition

The data acquisition module receives data from the FPGA, compacts the data using a simple run length encoding scheme, then presents the data ready to be sent out via USB. When the board is in data streaming mode (ie continuously sampling the shaft encoder position count registers) there is a special case in the operation of the EMIF. The sample rate signal generated by the FPGA and is presented on the IRQ line of the EMIF control bus. The MCU polls IRQ, and when a rising edge is detected, data is read from the FPGA, formatted and sent to the host PC via USB. Then the MCU waits for the next rising edge. This allows accurate timing generation to be implemented in the FPGA, the MCU is therefore freed from timer interrupt service routine overheads, and hence is able to read, process and transmit data at the highest rate possible. Fig. 8 shows the operation of the streaming data state machine.

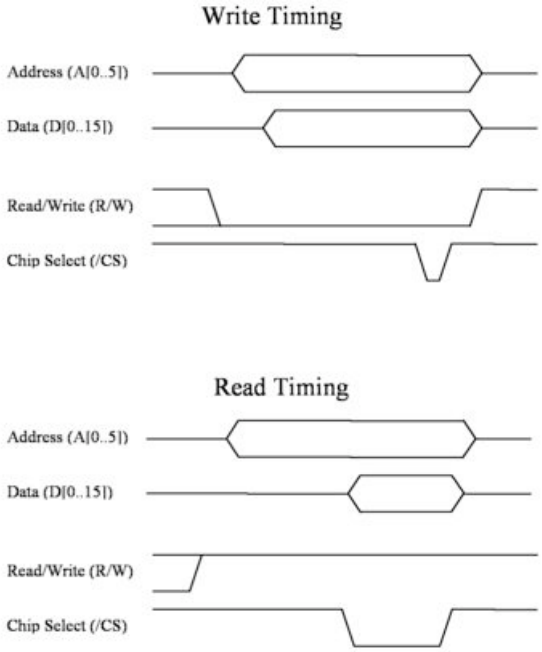


Figure 9: EMIF Timing Diagrams

B.5.4. Memory Interface

The EMIF is a parallel half duplex communications bus implemented in software. It has a 16bit wide data bus, 6bit wide address bus and 3bit bus control logic. The EMIF provides a fast data link between the ARM MCU and the FPGA, with a data rate of around 5MB/s. The ARM MCU is the bus master; all communications are initiated and controlled by it.

C. FPGA Register Set

FPGA register set description.

1. **Counter 1 Position Register High**

   Name: CNT1PRH

   R/W: Read Only

   ADDR: 0x00

   POR Value: 0x0000

   Bit Description:

   - CNT1PRH[15..0] - Counter 1 position high word.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT1PRH15	..	..	..	..	..	..	..	..	..	..	..	..	..	..	CNT1PRH0

Table 2: Register 0 (CNT1PRH) bit assignment

2. **Counter 1 Position Register Low**

   Name: CNT1PRL

   R/W: Read Only

   ADDR: 0x01

   POR Value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT1PRL15	..	..	..	..	..	..	..	..	..	..	..	..	..	..	CNT1PRL0

Table 3: Register 1 (CNT1PRL) bit assignment

   Bit Description:

   - CNT1PRL[15..0] - Counter 1 position low word.

     This word combined with CNT1PRH makes up the 32bit counter 1 position register.

     If the position count is not expected to exceed + or -32768, then it is sufficient to read only this register.

     Eg1: for a 2048 position shaft encoder, up to 16 revolutions in one direction can be tracked by CNT1PRL alone.

     Eg2: for a 2048 position device, up to 1048576 revolutions in one direction can be tracked by the combination of CNT1PRH and CNT1PRL

3. **Counter 2 Position Register High**

   Name: CNT2PRH

   R/W: Read Only

   ADDR: 0x02

   POR Value: 0x0000

   Bit Description:

   - CNT2PRH[15..0] - Counter 2 position high word.

4. **Counter 2 Position Register Low**

   Name: CNT2PRL

   R/W: Read Only

   ADDR: 0x03

   POR Value: 0x0000

   Bit Description:

   - CNT2PRL[15..0] - Counter 2 position low word.

     CNT2PRH and CNT2PRL have the same functionality as registers 0 and 1, but with regards to position counter 2.

5. **Counter 3 Position Register High**

   Name: CNT3PRH

   R/W: Read Only

   ADDR: 0x04

   POR Value: 0x0000

   Bit Description:

   - CNT3PRH[15..0] - Counter 3 position high word.

6. **Counter 3 Position Register Low**

   Name: CNT3PRL

   R/W: Read Only

   ADDR: 0x05

   POR Value: 0x0000

   Bit Description:

   - CNT3PRL[15..0] - Counter 3 position low word.

     CNT3PRH and CNT3PRL have the same functionality as registers 0 and 1, but with regards to position counter 3.

7. **Counter 4 Position Register High**

   Name: CNT4PRH

   R/W: Read Only

   ADDR: 0x06

   POR Value: 0x0000

   Bit Description:

   - CNT4PRH[15..0] - Counter 4 position high word.

8. **Counter 4 Position Register Low**

   Name: CNT4PRL

   R/W: Read Only

   ADDR: 0x07

   POR Value: 0x0000

   Bit Description:

   - CNT4PRL[15..0] - Counter 4 position low word.

     CNT4PRH and CNT4PRL have the same functionality as registers 0 and 1, but with regards to position counter 4.

9. **Counter Status Register**

   Name: CNTSR

   R/W: Read Only

   ADDR: 0x08

   POR Value: 0x0000

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CNTSR15	..	..	..	..	..	..	..	..	..	..	..	..	..	..	CNTSR0

Table 4: Register 8 (CNTSR) bit assignment

   Bit Description:

   - CNTSR [15..0]

     No Function.

10. **Counter Control Register**

    Name: CNTCR

    R/W: Write Only

    ADDR: 0x09

    POR Value: 0x0000

    Bit Description:

    - CNTCR [15..12]

      No Function.

    - CNTCR[11] - CNTR4RST

      Resets position counter 4 [CNT4PRH:CNT4PRL] to 0x00000000 when high, has no effect when low.

15	**14**	**13**	**12**
CNTCR15	..	..	CNTCR12
11	**10**	**9**	**8**
CNTR4RST	CNTR4EN	CNTR3RST	CNTR3EN
7	**6**	**5**	**4**
CNTR2RST	CNTR2EN	CNTR1RST	CNTR1EN
3	**2**	**1**	**0**
SCDIV3	SCDIV2	SCDIV1	SCDIV0

Table 5: Register 9 (CNTCR) bit assignment

- CNTCR[10] - CNTR4EN

  Enables position counter 4 when high, disables counter when low.

- CNTCR[9] - CNTR3RST

  Resets position counter 3 [CNT3PRH:CNT3PRL] to 0x00000000 when high, has no effect when low.

- CNTCR[8] - CNTR3EN

  Enables position counter3 when high, disables counter when low.

- CNTCR[7] - CNTR2RST

  Resets position counter 2 [CNT2PRH:CNT2PRL] to 0x00000000 when high, has no effect when low.

- CNTCR[6] - CNTR2EN

  Enables position counter 2 when high, disables counter when low.

- CNTCR[5] - CNTR1RST

  Resets position counter 1 [CNT1PRH:CNT1PRL] to 0x00000000 when high, has no effect when low.

- CNTCR[4] - CNTR1EN

  Enables position counter 1 when high, disables counter when low.

- CNTCR[3..0] - SCDIV3..SCDIV0

  Sample Rate clock divider. The following table indicates the sample rate as set by these bits:

11. **GPIO Control Register**

    Name: GPIOCR

    R/W: Write Only

    ADDR: 0x0A

    POR Value: 0x0000

SCDIV3	SCDIV2	SCDIV1	SCDIV0	Sample Rate (kHz)
0	0	0	0	1
0	0	0	1	10
0	0	1	0	50
0	0	1	1	100
0	1	0	0	150
0	1	0	1	200
0	1	1	0	250
0	1	1	1	300
1	0	0	0	400
1	0	0	1	500
1	0	1	0	600
1	0	1	1	1000
1	1	0	0	Reserved
..	..	..	..	Reserved
1	1	1	1	Reserved

Table 6: SCDIV[0..3] bit assignment

15	**14**	**13**	**12**
GPIOCR15	..	..	..
11	**10**	**9**	**8**
..	..	..	..
7	**6**	**5**	**4**
..	..	..	..
3	**2**	**1**	**0**
..	GPIOEN	GPIODDRL	GPIODDRH

Table 7: Register 10 (GPIOCR) bit assignment

Bit Description:

- GPIOCR [15..3]

  No Function.

- GPIOCR[2] - GPIOEN

  Tri-states all GPIO pins without altering any other GPIO setup (Not Implemented).

- GPIOCR[1] - GPIODDRL

  Sets the data direction of the GPIO port low byte pins. When set high, the pins are inputs, when low, the pins are outputs.

- GPIOCR[0] - GPIODDRH

Sets the data direction of the GPIO port high byte pins. When set high, the pins are inputs, when low, the pins are outputs.

12. **GPIO Input Register**

    Name: GPIOPIN

    R/W: Read Only

    ADDR: 0x0B

    POR Value: 0x0000

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
GPIOPIN15	..	..	..	..	..	..	..	..	..	..	..	..	..	..	GPIOPIN0

Table 8: Register 11 (GPIOPIN) bit assignment

Bit Description:

- GPIOPIN[15..0]

  GPIO Data input register. When GPIOCR[1..0] - GPIODDRH and/or GPIODDRL are set to '1', then the GPIO pins are set to inputs and their values may be read from GPIOPIN[15..8] and/or GPIOPIN[7..0].

13. **GPIO Data Output Register**

    Name: GPIOPOUT

    R/W: Write Only

    ADDR: 0x0C

    POR Value: 0x0000

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
GPIOPOUT15	..	..	..	..	..	..	..	..	..	..	..	..	..	..	GPIOPOUT0

Table 9: Register 12 (GPIOPOUT) bit assignment

Bit Description:

- GPIOPOUT[15..0]

  GPIO Data output register. When GPIOCR[1..0] - GPIODDRH and/or GPIODDRL are set to '0', then the GPIO pins are set to outputs and the values from GPIOPOUT[15..8] and/or GPIOPOUT[7..0].

  If GPIOCR[1..0] - GPIODDRH and/or GPIODDRL are set to inputs, the value in GPIOPOUT has no effect.

14. **Unused registers**

    Name: N/A R/W: N/A ADDR: 0x0D-0x3F POR Value: 0x0000

    These registers read back 0x0000 and may be written any value, but the contents have no effect on any FPGA system.

D. Data Acquisition Device Control Protocol

PC Protocol Specification.

Control over the ARM system is affected by some simple ASCII based codes. Every command consists of a control character that is optionally followed by a modifier character or data argument.

Two control modes are availible, verbose and non verbose. When verbose mode is enabled, text prompts are returned on reception of any command and data is streamed in ascii coded decimal format. Verbose mode is intended for use as a diagnostic/training tool in conjunction with a terminal emulator program such as teraterm (on MS Windows).
When verbose mode is disabled, very little human readable information is returned by the ARM firmware.

The following is a description of the various command codes that can be sent to the Data aquisition device and thier return values (if any).

The carriage return character is reffered to as 'CR' in the descriptions below.

1. **Verbose Mode**

   Control Character:'v'

   Modifier 0: '0' turns verbose mode off. In verbose mode returns "CRverbose offCR".

   Modifier 1: '1' turns verbose mode on. In verbose mode returns "CRverbose onCR".

   Power on default: Off

   Verbose mode enables messages to be sent back from the Data aquisition device to a terminal/shell. These messages allow use of the device by a user without need to write a script.

2. **Streaming Data**

   Control Character:'s'

   Modifier 0: '0' turns streaming data off. In verbose mode returns "CRstop polled streaming dataCR".

   Modifier 1: '1' turns streaming data on. In verbose mode returns "CRstart polled streaming dataCR".

Power on default: Stopped

After this command has been issued, the ARM echoes the 's' character then starts to read shaft encoder data from the FPGA. The data is sent from the USB port in a packet format which is defined by the system setup commands outlined below. The packet format is:

[DIAGNOSTIC_COUNT][SAMPLE_COUNT][ENCODER1_POS][ENCODER2_POS]

In non-verbose mode: All data is send in binary format. DIAGNOSTIC_COUNT is a 32bit (4bytes) count reporting the arm sample sate machine cycle count value, and is present only if diagnostic mode is enabled.
SAMPLE_COUNT, ENCODER1_POS and ENCODER2_POS are 8, 16 or 32bits long (1, 2 or 4 bytes) each depending on the specified Data Mode, and are always present. In this mode the ARM system aborts on buffer overflow, meaning that data streams will be truncated in preference to suffering corruption.

In verbose mode: the same rules apply but the data is reported in ASCII string format. Each element in the sequence is formatted separately, delimited by a space character, each packet is terminated by a carriage return character. The data in this mode is of variable length, run length encoding is not applicable, and the ARM system blocks on buffer overflow therefore the sample timing can become non-deterministic.

3. **Data Mode**

   Control Character:'b'

   Modifier 0: '1' sets data mode to 32 bits. In verbose mode returns "CR32bit modeCR".

   Modifier 1: '2' sets data mode to 16 bits. In verbose mode returns "CR16bit modeCR".

   Modifier 0: '3' sets data mode to 8 bits. In verbose mode returns "CR8bit modeCR".

   Modifier 1: '4' sets data mode to 16 bit count, 8 bit data. In verbose mode returns "CR16bpos, 8bit data modeCR".

   Power on default: 32 bit mode

   Data width determines the number of bits used to count the number of samples taken and the shaft encoder position.
   In 32 bit mode, the sample count and both encoder position values are 32 bits in size. The sample counter will wrap around at count 4294967296, and the position can be tracked for a maximum of 1048576 consequetive revolutions in either direction. The encoder positions are reported as absolute position values. The total packet length is 12 bytes in 32 bit mode.
   In 16 bit mode, the sample count and both encoder position values are 16 bits in size. The sample counter will wrap around at count 65536, and the position can be tracked for a maximum of 16 consequetive revolutions in one direction. The

encoder positions are reported as absolute position values. The total packet length is 6 bytes in 16 bit mode.
In 8 bit mode, the sample count and both encoder position values are 8 bits in size. The sample counter will wrap around at count 256. The encoder positions are not reported as absolute position values - the difference between the position at sample n and the position at sample n-1 is reported. The total packet length is 3 bytes in 8 bit mode.
In 16/8 bit mode, the sample count is 16 bit and both encoder position values are 8 bits in size. The sample counter will wrap around at count 65535. The encoder positions are not reported as absolute position values - the difference between the position at sample n and the position at sample n-1 is reported. The total packet length is 4 bytes in 16/8 bit mode.

4. **Run Length Encoding Mode**

   Control Character:'l'

   Modifier 0: '0' sets run length encoding on. In verbose mode returns "CRRLE enabledCR".

   Modifier 1: '1' sets run length encoding off. In verbose mode returns "CRRLE disabledCR".

   Power on default: Off

   When run length encoding is on, the ARM counts the number of sample rate cycles between changes in either of the encoder posoition values. When a change in position is detected, the sample count and new position data is sent.
   When run length encoding is off, the ARM sends sample count and position data once every sample period.

5. **Diagnostic Mode**

   Control Character:'d'

   Modifier 0: '0' sets diagnostic mode on. In verbose mode returns "CRdiagnostic mode onCR".

   Modifier 1: '1' sets diagnostic mode off. In verbose mode returns "CRdiagnostic mode offCR".

   Power on default: Off

   When Diagnostic mode is enabled, the ARM maintains a count vaule that increments every time the streaming data state machine is acivated. This count can be used to determine the how well the ARM is coping with the sample rate applied by the FPGA. When the count value reported is ¿1 then the state machine is completing at least 1 cycle per sample period. If the count value is ¡= 1 then the ARM is not able to read in the data from the FPGA, process the data and send it on to the USB stack with adequate time to spare. The timing between samples will eventually drift below the sample rate if verbose mode is on, otherwise the state machine will force an exit to the user interface.

6. **Interrogate**

   Control Character:'i'

   Modifier 0: '0' reports the name of the device. In verbose mode returns "CR-name=DBLPNDLMCR".

   Modifier 1: '1' reports the ID of thre device. In verbose mode returns "CRID=0CR".

   Modifier 2: '2' reports the device info string. In verbose mode returns "CRinfo=no info availibleCR".

   The interrogate command is a system command used for identificationn purposes. It is used by the pylogger software in the discovery method findShaftEncodeInterface() of class SerialCtrl(), module ctrl.py.

7. **Read-32**

   Control Character:'R'

   argument 0: the address to read from may be a number (ascii coded decimal) in the range of 0-32. In verbose mode a prompt is first given: "type read address:".

   returns: a number in the range of -2147483647 to 2147483648 (the register value). In verbose mode a prompt is given: "read long data from: XX = " where XX is the address specified.

   Read-32 reads two consequetive 16bit registers and reports their value as an integer. A carriage return character is expected after the address argument.

8. **Read-16**

   Control Character:'r'

   argument 0: the address to read from may be a number (ascii coded decimal) in the range of 0-32. In verbose mode a prompt is first given: "type read address:".

   returns: a number in the range of -32767 to 32768 (the register value). In verbose mode a prompt is given: "read long data from: XX = " where XX is the address specified.

   Read-16 reads one 16bit register and reports its value as an integer.
   A carriage return character is expected after the address argument.

9. **Write-32**

   Control Character:'W'

   argument 0: the address to write to may be a number (ascii coded decimal) in the range of 0-32. In verbose mode a prompt is first given: "type write address:".

   argument 1: the data to write to the register, a number in the range of -2147483647 to 2147483648. In verbose mode a prompt is first given: "type write long data".

   returns: In verbose mode a prompt is given: "writing long data: YY to: XX" where XX is the address specified, YY is the data specified.

Write-32 writes an integer value to two consequetive 16bit registers.
A carriage return character is expected after the address argument and data arguments.

10. **Write-16**

    Control Character:'w'

    argument 0: the address to write to may be a number (ascii coded decimal) in the range of 0-32. In verbose mode a prompt is first given: "type write address:".

    argument 1: the data to write to the register, a number in the range of -32767 to 32768. In verbose mode a prompt is first given: "type write long data".

    returns: In verbose mode a prompt is given: "writing long data: YY to: XX" where XX is the address specified, YY is the data specified.

    Write-16 writes an integer value to one 16bit register.
    A carriage return character is expected after the address argument and data arguments.

E. Programming the ARM Firmware

Basics of how to program the NXP ARM microcontroller.

Items Required:

1. MS Windows PC
2. RS-232 cable
3. LPC2000 Flash Utility application V2.2.3

Instructions:

1. start LPC2000 Flash Utility application

   set:
   comport to your comport number
   baudrate to 19200
   check "Use DTR/RTS for Reset and Bootloader Selection" tick box
   XTAL Freq. [kHz] text box to 12000

2. open the encoder interface device case.
3. short out JP1 labelled ISP_EN with a jumper
4. plug in serial port cable
5. plug in the USB cable

6. navigate to src/fw/main.hex
7. press Upload to Flash button
8. when complete press reset or power cycle the board.

Update:
An improved programming application now exists, Flash Magic, availible for download from `http://www.flashmagictool.com/`. This may be used in the same manner as the LPC2000 Flash Utility. Note that in order to use the tool,
under "Options/Advanced Options/Hardware Config" the following boxes must be checked.

1. "Use DTR and RTS to control RST and ISP pins"
2. "Keep RTS asserted while COM port open" in the

On the front end of the interface, the following settings can be used:

1. device: LPC2148
2. Baudrate: 38400
3. interface: none(ISP)
4. oscillator: 12MHz
5. check "verify after programming"
6. uncheck "fill unused flash"
7. check "erase all flash"
8. select file: /dp/src/fw/main.hex

To program, click start.

F. Author's Contributions

- Piers Lawrence
  - Mechanical drawings
  - Construction
  - Software development
- Michael Stuart
  - Electronic design

  - Transfer protocol design
  - Software development
  - Manual
- Richard Brown
  - Coordination
  - Report editing
- Warwick Tucker
  - Experimental Design
- Raazesh Sainudiin
  - Experimental Design
  - Report editing
  - Principal Investigator

G. Acknowledgements

Piers Lawrence was supported by a Summer Scholarship 2008-2009 jointly funded by the University of Canterbury (UC) and the Tertiary Education Commission of New Zealand. We thank Professor Alan Nicholson for access to UC's Civil Engineering Laboratories. The Department of Mathematics and Statistics at UC supported this project.

References

[1] O.E. Lanford. A computer-assisted proof of the Feigenbaum conjectures. *Bull. Amer. Math. Soc. (N.S.)*, 6(3):427–434, 1982.

[2] J. Hass and R. Schlafly. Double bubbles minimize. *Ann. of Math.*, 151(2):459–515, 2000.

[3] W. Tucker. A rigorous ODE solver and Smale's 14th problem. *Foundations of Computational Mathematics*, 2(1):53–117, 2002.

[4] T.C. Hales. A proof of the Kepler conjecture. *Ann. of Math.*, 162:1065–1185, 2005.

[5] P. Jäckel and T. Mullin. A numerical and experimental study of codimension-2 points in a parametrically excited double pendulum. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1980):3257–3274, 1998.

[6] T. Shinbrot, C. Grebogi, J. Wisdom, and J.A. Yorke. Chaos in a double pendulum. *Am. J. Phys.*, 60:491–496, 1992.

[7] R.B. Levien and S.M. Tan. Double pendulum: An experiment in chaos. *Am. J. Phys.*, 61(11):1038–1044, 1993.

[8] Y. Liang and B. Feeny. Parametric identification of a chaotic base-excited double pendulum experiment. *Nonlinear Dynamics*, 52(1):181–197, 2008.

[9] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[10] Y. Lin and M.A. Stadtherr. Validated solutions of initial value problems for parametric ODEs. *Applied Numerical Mathematics*, 57(10):1145–1162, 2007.

[11] W. Tucker, Z. Kutalik, and V. Moulton. Estimating parameters for generalized mass action models using constraint propagation. *Mathematical Biosciences*, 208:607–620, 2007.

[12] L. Le Cam. *Asymptotic Methods in Statistical Decision Theory*. Springer-Verlag, 1986.

[13] R. Sainudiin. *Machine Interval Experiments: Accounting for the Physical Limits on Empirical and Numerical Resolutions*. LAP Academic Publishers, 2009.

[14] Avago Technologies. Quick Assembly Two and Three Channel Optical Encoder. http://www.avagotech.com/docs/AV02-1046EN.