Foucault Pendulum Electronics Kit.

D05_Description_Electronic_Circuits.

www.foucaultpendulum.nl

Document version	2025-11-02
Relates to PCB version	PCB BobControl v1. PCB Receivers v1. PCB PMS_Base v0. PCB Preamps v0.
Related Documents	Schema BobControl PCB. Schema Receivers PCB. Schema PMS_Base PCB. Schema PreAmps PCB. D10_Assembly Instructions.

Power: (PCB BobControl)
The board (and the whole system) is powered from a stabilized 24 VDC, 2$^+$A power supply (not in the kit, you should buy one yourself)
The 24 V enters the board at the wire terminals marked GND and +24V in the upper right.
The DC/DC converter U2 produces + 5V for the Arduino Mega, the Ethernet shield, led's and so on, and a -5V for the negative supply of the operational amplifiers.
Note: A switched mode Power Supply Unit generally has a safety ground connection. Use it and connect the PSU with a 3-wire cord to a wall outlet with safety ground. Also connect the safety ground connection on the PSU to the 24V GND side. This will prevent unpleasant leakage currents to flow through your body when you touch any part of the pendulum system.

Note: You might have the idea to use 2 12 Volt acid-lead car batteries for the 24 Volt supply, and continuously keep them under charge. Not a bad idea in countries with a not so reliable mains supply. But there can be a problem with the stability of such a supply. The voltage on the batteries can go up to 27 Volts during charging.
The +5 and -5 Volt converter is not stabilized and has a limited input voltage range (See datasheet). Also the functioning of the receivers may vary a little with the supply voltage.
I advise to use a stabilized 24V power supply, and if everything works fine try if it stil works with slightly different voltages.
Warning: If you do use (car) batteries, use a fuse of maximal 2A in the battery connection lead !!

Generation of the 465 kHz Wire signal:
This signal is generated by the Direct Digital Synthesizer AD9833 U1 and amplified by the circuit around Q4 and Q5. These transistors are functioning as current sources because of the not decoupled emittor resistors R38 and R39. The output amplitude can be adjusted with R19. Set it to ca. 5 Vpp with the pendulum wire connected. R22 and R31 basically determine the voltage gain, but for higher frequencies the gain is limited by C17. This is done because we start with a squarewave signal which contains a lot of harmonics.
C21 gives the DC-separation. R44, and R45 keep the wire on ca. 12VDC so a contact with a (electrically conductive and grounded) Charron ring is possible. R45 and R41 bring the signal at a level that the Arduino can detect, and C22 prevents the 465 kHz signal to reach the Arduino. If you do not use the "Touch Charron ring" facility you can leave away R41, 44, 45 and C22.
Note: the "Touch Charron ring" facility is currently not implemented in the software. You will have to desing your own code for it.
Note: When the wire touches a Charron Ring the 465 kHz signal on the wire will be altered or shorted. This may corrupt the PMS system.

Why use a DDS and not a simple oscillator to generate the 465 kHz?
Well that is because I've planned an experiment where the drive timing for the bob is not triggered by a Center Pass or so, but at a very precise frequency, namely the frequency (1 / period time) of the major axis of the elliptical path. As is explained in the <u>chapter about the Period Time</u> the swing time of a pendulum becomes longer with larger amplitude. So the minor

axis of the inevitable ellipse has a sligtly higher frequency than the major axis. The aim is to only excite the frequency of the major axis and not that of the minor, and so limit ellipse growth. As this frequency difference is in the order of 10^{-5} or less the experiment requires a very stable frequency source which can be adjusted in steps of parts per million. A DDS can do that job, in combination with software frequency dividers.

This is also the reason that the 465 kHz signal is routed to the external Timer5 clock input on the Arduino pin 47 and that we configure the DDS for a squarewave output signal.

The bare signal from the DDS can be adjusted in frequency steps of around 0.1 Hz. When we divide the frequency by $\sim10^6$ to get to a frequency of ~0.5 Hz, suitable for a 4 meter pendulum, we can adjust that frequency in steps of 10^{-7} Hz.

The DDS parameters are as follows:

Xtal Frequency is supposed to be 25,000,000 (25 MHz)

FrequencyOut = FXtal * FrequencyWord / 2^{28} [Hz]

FrequencyWord = FrequencyOut * 2^{28} / FXtal So for 465 kHz we need: DDS_FrequencyWord = 465000 * 2^{28} / 25000000 = 4992899.

Remark:

The wire with the 465 kHz signal forms the antenna of a radio transmitter. However, as the wavelength of this frequency is some 650 meter, a wire of up to 6 meter forms an antenna with a length of less than λ / 100. That is an extremely inefficient antenna, and there is no reason to expect the authorities on your doorstep telling that you are operating an illegal radio transmitter.

Generation of the Drive Pulse:

The Drive Pulse is current controlled. This means that for the amplitude control we change the current through the drive coil and not the timing or duration of the impulse. (In the past I changed the width of the drive pulse to control the amplitude. This however influences the effective timing, which might interfere with the "Schumacher criterion")

The power transistor Q3 with IC1B forms the current source. The current is determined by the voltage on IC1B pin 5 and the combined values of R32 and R35 as I = V(pin 5) / (R32//R35). The default setting is 2A per Volt on IC1 pin 5, so nearly 10A full scale. You may take other values for R32 and 35, or just mount 1 or mount them in series. If you take other resistors, use a wire wound resistor. film resisitors might be damaged by large peak currents in the short pulses.

In the Arduino Timer 4 is configured to produce a squarewave of ca. 16 kHz and a variable dutycycle on pin 6. After the lowpass filter with R16 and C10 it is almost DC. The voltage follower IC1A gives it a low impedance. With R21 and Q2 the drive pulse is made, with the timing and duration given by the PG0_/DRV signal from Arduino pin 41. Q2 shorts the signal to GND, except when the active low drive pulse is given.

R20 keeps Q2 conductive during and before the start-up of the Arduino, so we won't get a spurious or long lasting drive pulse.

The PTC resistor R18 limits the current to a safe value for the case where Q3 might short or Q2 may fail of the firmware goes wild. If a large current flows for a longer time through R18 it will heat up and reduce the current to a safe level. In normal operation the large capacitors C11 and 12 deliver the short lasting drive energy. This also has the advantage that the 24VDC power supply does not need to be dimensioned for the peak drive current.

I introduced this safety feature after a drive coil almost got fire at such an event......

The drive coil is connected via the terminals "+DRV, -DRV" which can adapt rather thick wires. Mind the polarity of the pulsed drive current. The "Schumacher crirterion" requires the drive pulse to push the bob away from the center.

Depending on the drive pulse current and duration Q3 might get quite hot. In that case mount a piece of aluminium cooling plate to the transistor.

Deriving the Center Pass signal from the Drive Coil:

Note that this approach is stil experimental and not yet tested.

This may be an attractive option, because then there cannot be a misalignment between the Drive Coil and the Center Detection coil.

There are however some problems of electronic nature to overcome. The Drive Coil generally has not very much windings and so produces a weak signal when the bob flies over. So we need quite some amplification for the CenterPass detection. And then comes the 24 Volt BANG of the Drive Pulse, which will overdrive the amplifier. I've tried to tame this by the following:

Because both sides of the Drive Coil carry varying voltage levels we use a differential stage with R42, 50, 53 on one side and R43, 51, 52 on the other side. This makes the output signal on IC1 pin 8 insensitive to the common voltage of the coil, while the induced voltage across the coil is amplified by 1.5. Just before the drive pulse arrives and during the drive pulse we

engage OC1 which shorts the differential signal.

With the same timing Q6 and Q7 disconnect the signal to the second stage with IC2D.

If you dont use this feature you can leave away OC1, Q6, Q7, R40, 42,43, 50, 51 and C27. Do mount R52 and R53 to keep the the C- section of IC1 at a welll defined voltage level, although not used.

A word about the polarity of the signals: If you send a current through the drive coil such that the magnet in the bob is pushed outward, then the signal on the + wire of the coil goes negative when the bob approaches the coil from far. For this reason the amplifier stage IC1C is chosen inverting.

The selection which signal to use for Center Pass detection is done on the board with jumpers on pins 13-14-15-16 and in the firmware by using either the ADC4 or ADC5 input.

Deriving the Center Pass signal from a Center Pass Coil:

A Center Pass coil should be connected to pins 5 and 7 of the COILS flatcable connector. With jumpers on 9-10, 11-12 or 9-11,10-12 the polarity can be set such that the signal on the Arduino input A6 goes positive when the bob approaches the coil.

Place a jumper on 13-14.

IC2D forms a non inverting amplifier. The gain can be adjusted with R36. The signal is AC-coupled with C23 to an analog input of the Arduino and kept on an average DC-level of 2.5 Volt (half-scale) by R46 and 47. A low pass filter with R59 and C29 suppresses noise. R59 also protects the Arduino from to much current when the signal on C23 should exceed the 0 .. 5V range.

The selection which signal to use for Center Pass detection is done in the firmware by using the ADC5 input.

Deriving the Center Pass signal with the Capacitive method:

The selection which signal to use for Center Pass detection is done in the firmware by using the the ADC4 input.

You can leave away all components around IC2D, except R26. Place a short wire in stead of C18 to keep section D of IC2 at a well defined voltage level. If you also don't use a Rim Coil you can leave away IC2A and all its surrounding components.

Deriving the Rim Pass signal from a Rim Coil:

This works much the same as for the Center Pass Coil, now with the circuit around IC2A.

The coil should be connected to pins 1 and 3 of the COILS connector. Jumper 5-6, 7-8 or 5-7, 6-8 to select the polarity. On the Arduino ADC7 input the signal should go positive when the bob flies over te rim coil in outward direction, and negative when the bob returns inward.

The selection which signal to use for RimPass detection is done in the firmware by using the the ADC7 input.

Deriving the Rim Pass signal with the Capacitive method:

Is currntly not implemented in the frrmware. The ADC6 Receiver channel is implemented for this. The signal is A/D converted and numerically displayed on the GUI, but no further implementation has been made.

Note: The shape of this signal is very much determined by the shape of the bob in combination with the dimensions of the Rim Electrode. (not in the Kit)

I have tried it in my new sub-meter pendulum where the bob has a diameter close to that of the rim electrode I exeperimented with and so produced a signal where the rim passage was hard to detect. I abandoned this method for the time being. If your bob is shaped more like a long thin cylinder the signal may be better usefull..

The Reset Circuit.

It may be desirable that the Arduino is able to reset itself, when the frimware detects an unusual condition (currently not supported), or by a command from the PC program.

A self-reset could be realized in the firmware just by a jump to the reset-vector, but then the hardware (internal peripheral circuits and the Ethernet Shield) are not reset. Programming the RESET pin as an output and pulling it low will not work either, because that action immediatly stops itself, which results in an extremely short reset pulse.

I designed a delay circuit with R1, 2, 3 and C1, 2, which engages Q1 long enough for a proper reset of all hardware.

Led's:

The presence of the 3 supply voltages is indicated by leds marked 24V, +5V and -5V.

The board has place for 8 other led's. They are controlled by the bits 1, 3, 5, 7 of PORTC and

PORTL of the Arduino. You may decide about the colors yourself, there are some extra 3 mm leds in the kit.
You may change the resistor values if the leds appear to bright or to dim. The values given are for approximately 1 mA current.
Currently the firmware controls the leds as follows:
HFSW: shows the HalfSwing signal, so if the led is ON we are in the first HalfSwing.
DRV: flashes on for a short time at each Drive Pulse. (It does not represent the duration of the pulse).
RIM1: Is on for a short time at each detected outward going Rim Pass1.
RIM2: Is on for a short time at each detected inward going Rim Pass2.
SYNC: is on when the software timing is in sync with the bob.
COMM: Toggles after reception of each 5th message from the PC. (ca. 1 Hz for 10 messages per second). Flashes rapidly if there is a message timeout.
SP1 and SP2: spare. You can assign a function yourself.
During startup of the Arduino a led test is normally done, but can be skipped with OptionJumper-1.

Test Facilities:
The long M3 screw in the hole marked GND is meant to connect the GND leads of test equipment (multimeter, oscilloscope) to.
The supply voltages on the board can be measured with a multimeter connected to the right side of R12, 15 and 17 for the +24, +5 and -5V. Check these voltages before mounting the Arduino, Ethernet Shield, IC1 and IC2.
All signals on the Arduino Mega are easily accesible when you hook a pin-pin patch wire to your oscliloscope probe and plug the other end in the receptor of the signal to observe.
The signals A8 .. A15 on the Arduino are configured as digital outputs. In the firmware they are assigned to several signals as diagnostic tools DIAGPIN_Ax. See the software description.
Several other signals can be observed with an oscilloscope. The probe-pin can be put in one of the connections marked TP_xxxxxxx.
The Drive Pulse can be tested by hooking a probe to the cooling tab of Q3. It will show the voltage on the large capacitors, a lower voltage during the Drive Pulse.
The Drive current can be viewed with a probe hooked to the not-grounded-side of R32, or TP_I_DRV.
Signals on a jumper block: in most cases one can hook a probe to the pin when the jumper is lifted a few mm.
The through-hole resistors used generally allow a probe to be hooked to one of the wires.

Receivers and Position Measurement System (PMS)
The task of these circuits is to translate the HF signals picked up by the sensing electrodes to a DC voltage that can be processed by the Arduino.
The receiver PCB contains 6 identical receiver circuits which work in close cooperation with the pre-Amplifiers on the Base_PCB and (if used) the Pre_Amp PCB for Center- and Rim Pass Detection.
The use of the 465 kHz carrier frequency makes the system very insensitive to mains hum and samelike disturbances.
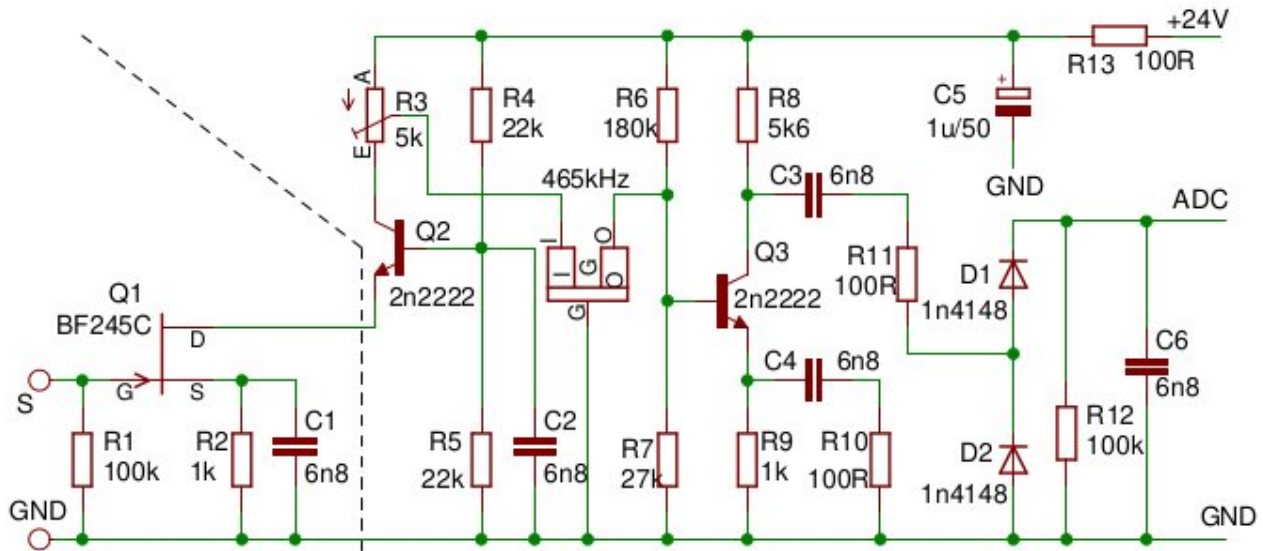
Fig 1. Receiver example.

The operation is explained with fig 1, a complete Receiver Circuit Example.
Note: The part left of the dashed line is on the Base PCB or the Preamp PCB, and connected by a possibly quite long flat cable.
The pickup electrode S is connected to the gate of Q1. This is a Junction Field Effect Transistor because the signal from the electrodes has a high-impedance. The drain current of Q1 is determined by R2 and the characteristics of the transistor, which vary a bit from piece to piece.
Q1 is connected to Q2 in the so called Cascode mode, which has a low impedance on the flatcable connection between the boards. This way the cable carries little signal voltage and makes the connection rather insensitive to crosstalk and other disturbing signals. Note that the collector current of Q2 is almost the same as the drain current of Q1.
The 465 kHz signal is taken from R3, which allows the overall gain to be adjusted, to the bandpass filter with 465 kHz center frequency. These cheap filters, designed for the Intermediate Frequency stage of AM receivers, have a bandwidth of a few kHz, which eliminates a lot of noise.
The second stage with Q3 has a rather high gain because of the partially decoupled emittor resistor R9/C4/R10. The AC signal is now large enough to be full-wave rectified by the diodes D1 and D2, in combination with R11, R12, C3 and C5, and routed to an A/D converter input on the Arduino.
Note: Fully decoupling the emittor of Q3 results in forming an oscillator circuit, based on the complex impedance of the ceramic filter and the parasitic capacitive feedback from the collector of Q3, in combination with it's high gain. Do not make R10 smaller than 100 Ohm.