­­­

|  |
| --- |
| DomeBuddy |



29 October 2022

by Keith Rickard  
keithrickard@hotmail.com



# Contents

[Introduction 4](#_Toc59909964)

[Schematic 5](#_Toc59909965)

[Installation 6](#_Toc59909966)

[Power Source 6](#_Toc59909967)

[Power Supply Unit (PSU) 6](#_Toc59909968)

[PSU Resistor 7](#_Toc59909969)

[Connector Box 8](#_Toc59909970)

[DomeBuddy Controller Box 10](#_Toc59909971)

[Dome Motors 12](#_Toc59909972)

[DomeBuddy Dongle 13](#_Toc59909973)

[Setting Up DomeBuddy 14](#_Toc59909974)

[Testing DomeBuddy 14](#_Toc59909975)

[Motors Connection Test 14](#_Toc59909976)

[Turning the Dome 15](#_Toc59909977)

[Bluetooth Connection 15](#_Toc59909978)

[Calibrating the Dome 17](#_Toc59909979)

[Calibration Process 17](#_Toc59909980)

[Operating Dome Buddy 18](#_Toc59909981)

[Important Notes 18](#_Toc59909982)

[Starting Up 18](#_Toc59909983)

[Session Operation 18](#_Toc59909984)

[Parking the Dome at its Home Position 20](#_Toc59909985)

[Defining the Home Position 20](#_Toc59909986)

[Updating Arduino Firmware 22](#_Toc59909987)

[Arduino Nano Microcontrollers 22](#_Toc59909988)

[Using the Arduino IDE 22](#_Toc59909989)

[Changing DomeBuddy Controller Firmware Parameters 24](#_Toc59909990)

[RAMP\_SPD 25](#_Toc59909991)

[MAX\_SPD 25](#_Toc59909992)

[ENC\_DIR 25](#_Toc59909993)

[MTRA\_DIR 25](#_Toc59909994)

[MTRB\_DIR 26](#_Toc59909995)

[Changing DomeBuddy Dongle Firmware Parameters 26](#_Toc59909996)

[ENQ\_DELAY 26](#_Toc59909997)

[Programming the Bluetooth Module 27](#_Toc59909998)

[Programming DomeBuddy Controller’s HC-05 28](#_Toc59909999)

[Programming DomeBuddy Buddy’s HC-05 29](#_Toc59910000)

[DomeBuddy Components 30](#_Toc59910001)

[DomeBuddy Controller 30](#_Toc59910002)

[Arduino Nano 30](#_Toc59910003)

[HC-05 Bluetooth Module 30](#_Toc59910004)

[Cytron Motor Driver 30](#_Toc59910005)

[DC-DC Converter 30](#_Toc59910006)

[Connectors 31](#_Toc59910007)

[DomeBuddy Dongle 31](#_Toc59910008)

[Arduino Nano 31](#_Toc59910009)

[HC-05 Bluetooth Module 31](#_Toc59910010)

[RJ12 6P6C plug 32](#_Toc59910011)

[Motors and Rotary Encoder 32](#_Toc59910012)

[Windscreen Wiper Motor 32](#_Toc59910013)

[Rotary Encoder 32](#_Toc59910014)

[Rotary Encoder Plug and Socket 33](#_Toc59910015)

[Sketches (Source Code) 34](#_Toc59910016)

[DomeBuddy Controller 34](#_Toc59910017)

[DomeBuddy Dongle 40](#_Toc59910018)

# Introduction

**DomeBuddy** is a controller whose primary function is to rotate an observatory dome so that its opening (slit) is always aligned to the direction of where the housed telescope is pointing. It ensures that the dome will keep up with telescope’s tracking and rotate it to the telescope’s new position after slewing.  
  
The controller is connected to the following:

* 12V power source (PC power supply unit)
* Two car windscreen wiper motors which directly turn the dome
* A rotary encoder with a wheel in contact with the dome rim to accurately measure the dome’s direction and distance of travel.

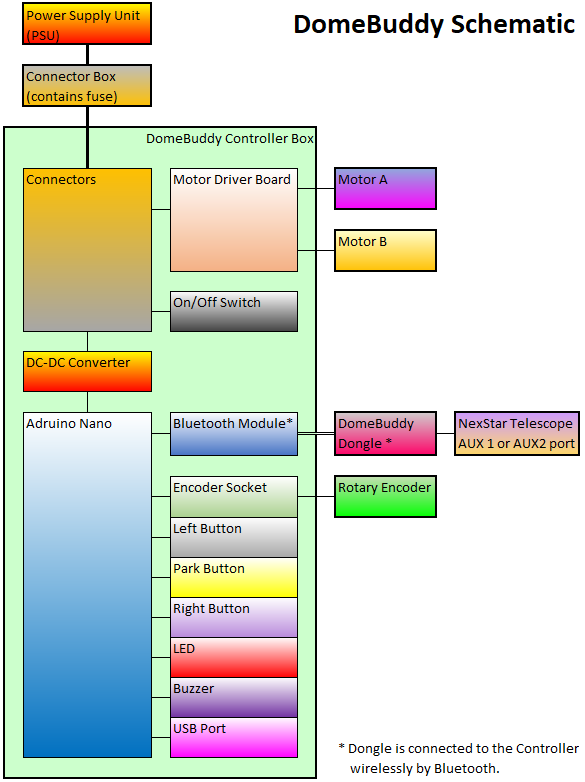
Other features include:

* Parking / unparking the dome, i.e. stop and starting the dome
* Rotate and park the dome at its home position
* Synchronise the dome’s opening with the direction the telescope is pointing

In order for the **DomeBuddy Controller** to know where the telescope is pointing, the **DomeBuddy Dongle** is connected to the telescope’s mount with its plug. In the case of the Celestron Nexstar this is either of the AUX1 or AUX2 ports. The **Dongle** communicates with the telescope’s on-board computer and periodically asks it for the telescope’s azimuth and its slew/track status. It then sends this information to the **Controller** via Bluetooth so that **Controller** can act on this information accordingly.  
  
**DomeBuddy** is self-sufficient and will slave itself with the telescope without any intervention of driver software such as ASCOM.

# Schematic

Here is a block diagram of the how the components of the **DomeBuddy** system are connected.

­­

# Installation

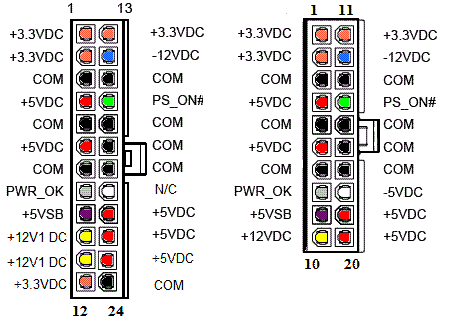
The following information assumes the dome drive motors and rotary encoder have already been installed inside the dome.



## Power Source

DomeBuddy requires a 12V power source to power the 12V car windscreen wipers which turn the dome. These motors require about 3.5 amps each under no load. The DomeBuddy control box can provide 30 amps momentarily to the motors, but only 8-10 amps maximum are required to get the dome turning at full speed.

### Power Supply Unit (PSU)

An excellent power source is a PC ATX power supply unit (PSU) as many can supply the amperage required to drive the motors. A PSU which can supply 30 amps to its +12 volt lines should suffice. A PSU with an on/off switch on its case is a good choice. PSUs from disused PCs are a useful source.   
  
The PSU will have large number of wires coming out of it which are used to connect to a computer motherboard, hard drives, etc. The large rectangular mother board plug comes in two versions - the following shows these pinouts:  
  


Before modification, the PSU should be tested on a bench. With mains power to the PSU turned off, the **Green** wire (**PS\_ON**) should be grounded by connecting it to a **Black** wire (**COM**). An unfolded paper clip can be inserted into the appropriate wire sockets of the plug. Turning on the mains should power up the PSU and start the fan inside turning. A multimeter can be used to test the voltages on each wire. Connect the negative probe to a **Black** wire (**COM**) and then connect the positive probe to each of the following wires in turn to get the corresponding give value:

* Yellow wire +12V
* Red wire +5V
* Orange wire +3.3V
* Blue wire -12V
* White wire (if it exists) -5V
* Purple +5V

If all the above lines provide close to these voltages then the PSU should be good to use.

### PSU Resistor

If the PSU fails to fire up then it may need to have a load resistor attached to activate it. Look at the label on the side of the PSU to see which gives details about the amperage of the +5V and +12V lines. The lines which give the highest amperage are the ones to use to connect a resistor to – most commonly, this will be the **Red 5V** wires. Use a 22 ohm ceramic resistor to connect one of these wires to one end and a **Black** wire (**COM**) to the other. The resistor will get hot so fixing it to the PSU case will help to sink and dissipate its heat.

|  |  |
| --- | --- |
| 0R22 (0.22 Ohm) 10W Ceramic WW - Click Image to Close | *22 Ohm Resistor* |

## Connector Box

|  |  |
| --- | --- |
|  |  |
| *Junction Box* | *M20 Cable Glands* |

The use of a junction box with a cable gland at each end is an excellent way to connect the PSU wires to a 13 amp mains flex cable. The flex goes to the DomeBuddy controller box.  
  
All the connectors on the end of the PSU wires need to be cut off, leaving cables cut to length. If necessary, connect a **Red** (**+5V**) or **Yellow** (**+12V**) wire to a 22 ohm resistor and a **Black** (**COM**) wire to the other end (please see the **PSU Resistor** section above for more information).

All the remaining wires are fed through the one of the cable glands. The cables that are not used are insulated at their ends. The **Green** (**PS\_ON**), seven **Yellow** (**+12V**) and seven **Black** (**COM**) wires will be used. All of these wires need to have their ends stripped leaving about 5mm bare wire.  
  
Connect seven **Black** (**COM**) wires together using two 5 way Wago connectors, and do the same with the seven **Yellow** (**+12V**) wires. Lifting an orange lever on the Wago connector allows the wire to be inserted.

|  |  |
| --- | --- |
|  |  |
| *5-way Wago Connector* | *Automotive Fuse Holder* |

The flex cable is inserted through the opposite cable gland. The **Blue** (**Neutral**)wire of the flex cable is connected to the **Black** wires, and the **Brown** (**Live**)wire to the **Yellow** wires. A piece of the **Blue** and **Brown** wires are used to join the Wago connectors together. The following illustrates how these wires should be connected.

|  |  |
| --- | --- |
|  |  |
| *Connecting the Black & Blue wires* | *Connecting the Yellow & Brown Wires* |

The **Brown** wire leaving the connectors is connected to an inline automotive fuse using **Blue** female spade connectors. The other end of the fuse holder is connected to the flex cable.  
  
A 15 amp fuse should be used. If this fuse blows while the motors start turning the dome, then try replacing it with a 20 amp fuse, and so on up to 30 amps.  
  
Finally, the **Green** (**PS\_ON**) wire is connected to theflex cable’s **Yellow/Green** (**Earth**) wire. This can simply be connected to a 2-way Wago connector or one piece of a terminal connector strip.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| *Terminal connector strip* | *Female spade connector* | *Automotive blade fuse* |

Here is a picture of the Connection Box before the PSU cables are connected:



The junction box maybe connected to a wall, or similar. When done, the cable glands should be tightened up and the box’s cover fitted on.

## DomeBuddy Controller Box



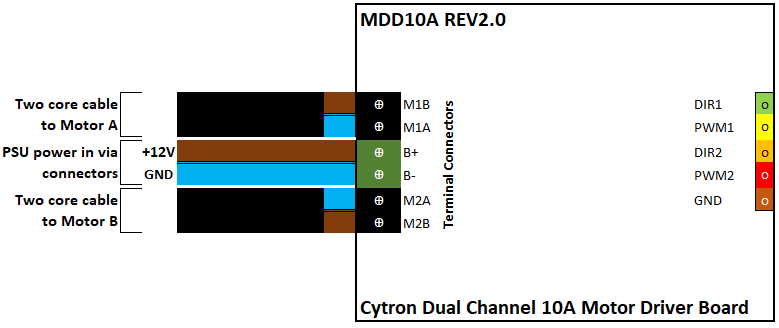
The **DomeBuddy Controller** box needs to be sited so that its buttons, switch and USB port are easily accessible, ideally inside the dome room.

The inside of the box has a lot of wires, cables, connectors, etc. Please take careful note of what is connected to what - better still take several pictures - in case something becomes disconnected.

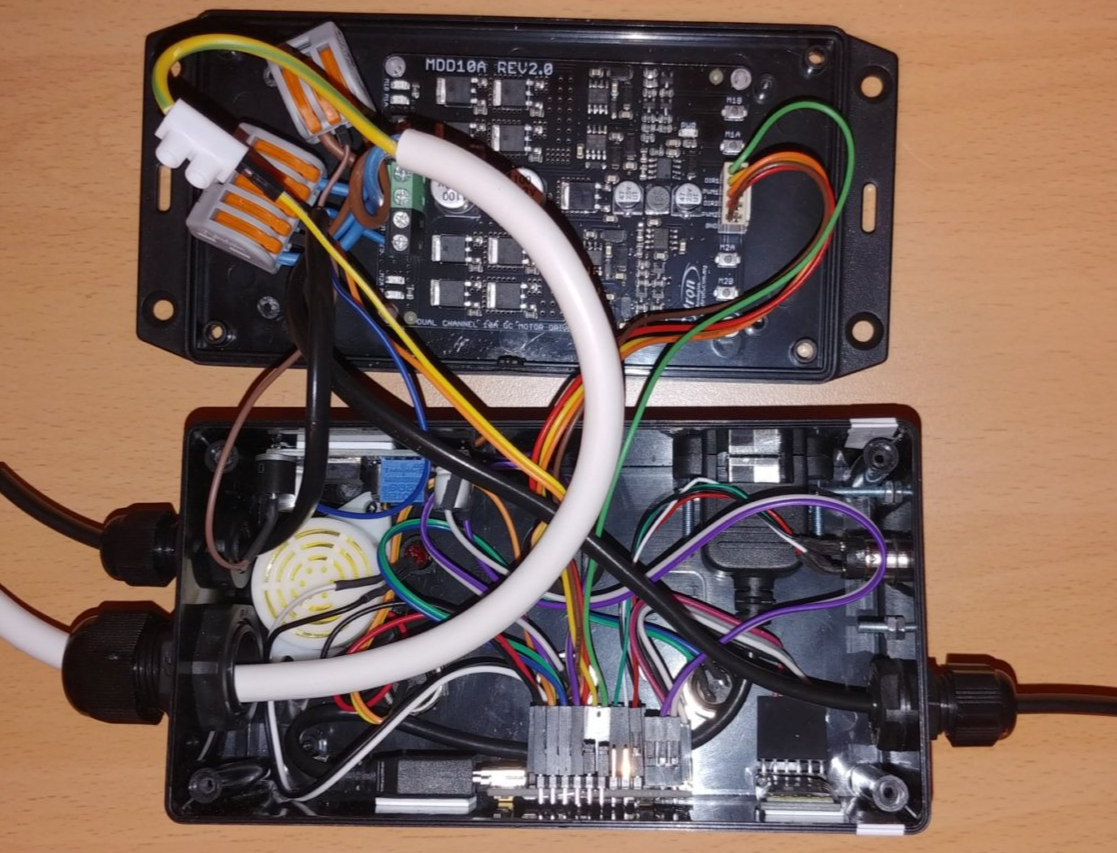
The flange base should be screwed to a wall, or similar. The flex cable from the power supply unit is inserted through the POWER IN gland of the controller box and connected inside after it has been cut to length. The cable needs to be stripped to leave 50mm length wire. All the wires need to be stripped to leave 5mm bared ends.  
  
The **Brown** and **Blue** wires of the flex cable are inserted into an empty slot of the Wago connectors with the same colour wire coming out of them. (The **Brown** wire is connected to the 3-way connector and the **Blue** to the 5-way one.) This is done by lifting the orange lever of the connector, inserting the wire then pressing and snapping the lever down again.

The **Yellow/Green** wire is connected to the single terminal connector. This wire is connected to the on/off switch.

Two core mains cable (1.25mm2) is run between the **Controller** box and each of the wiper motors. The bared ends of the **Blue** and **Brown** cable wires are connected to the motor drive board inside the **Controller** box. The wires are connected as follows:



Here is an example of how the wires are connected to the **DomeBuddy Controller**:



When everything has been connected correctly, screw the **Controller** box’s lid on, take out the slack of the cables and then tighten up the cable glands.

Finally, insert the rotary encoder plug into the encoder socket of the **Controller Box**.

## Dome Motors

The two drive motors which turn the dome are car windscreen wiper motors with a rubber tyred wheel on the end of a special fabricated spindle.

|  |  |
| --- | --- |
|  |  |

Wiper motors have high torque and can easily be driven by the motor driver board inside the **DomeBuddy Controller** box.  
  
The two core wires from the controller need to have **Red** 4.8mm female spade connector attached to the 5mm bared ends of the cable’s **Brown** and **Blue** wires.

|  |  |
| --- | --- |
|  |  |
| *Connecting the motor* | *4.8 mm female connector* |

With the motor’s spindle pointing up, the **Blue** wire is connected to the left pin of the motor’s connector and the **Brown** wire to the right pin.

## DomeBuddy Dongle



The **DomeBuddy Dongle’s** 6P6C RJ11 plug simply plugs into either **AUX1** or **AUX2** port on control panel of the Celestron NexStar mount.

|  |  |
| --- | --- |
|  | *A = PC Interface Port B =* ***AUX1*** *Port C =* ***AUX2*** *Port D = +12V Output Jack*  *E = Auto Guider Port* |

The ports have a +12V line which powers the **DomeBuddy Dongle**. The **Dongle** can be fixed to the mount with self-adhesive pads or Velcro strips.

# Setting Up DomeBuddy

## Testing DomeBuddy

Below are tests which should be carried out before using **DomeBuddy** for the first time. Make sure the dome is unlocked and free to rotate!

### Motors Connection Test

This test checks that the motors

1. Ensure the power to **DomeBuddy** is switched off at the mains outlet.
2. If the PSU has an on/off switch, turn this off too.
3. Ensure the **DomeBuddy Controller** on/off switch is turned off by pressing the top of the switch down.
4. Ensure the **DomeBuddy Dongle** is not powered up. Either unplug it from the telescope’s control panel or switch off the telescope.
5. Disengage the motors from the dome.
6. Turn on the mains outlet.
7. Turn on the PSU, if applicable.
8. Turn on the **DomeBuddy Controller**.
9. The **Controller** will give two half second beeps.
10. The **Controller’s** LED will remain constantly lit indicating that is parked i.e. inactive.
11. Now press the **RIGHT** button to make both motors turn clockwise. They will slowly accelerate to full speed and then slowly decelerate when the button is released.

If neither motor turns, then turn off the **Controller** and then check all connections between the **Controller** and the motors, and the button connections inside the **Controller** box. Repeat steps 8 to 11 again.  
  
If one or both motors turn anti-clockwise then the motor wires are connected the wrong way round. Turn off the **Controller** and then switch the wires round. (If this is not possible then the **Controller’s** firmware can be modified – see the section **Updating Arduinoo Firmware**.)

With the **Controller** powered up, press the **LEFT** button to make both motors turn anticlockwise. If the motors do not turn correctly then check the connections as described above.

### Turning the Dome

This test ensures that the power to the motors is sufficient to turn the dome and that the fuse in the **Connector** **Box** is of the correct value. The starting value for the fuse should be 15 amps.

1. Ensure the **DomeBuddy** **Controller** is switched off.
2. Engage the motors so that their tyres are in contact with the drive surface of the dome.
3. Ensure the **DomeBuddy Dongle** is not powered up. Either unplug it from the telescope’s control panel or switch off the telescope.
4. Turn on the **Controller**.
5. The **Controller** will give two half second beeps.
6. The **Controller’s** LED will remain lit constantly indicating that is parked i.e. inactive.
7. Now press the **RIGHT** button to turn the dome clockwise. Allow the motors to turn at full speed after they slowly accelerate, and do a full rotation before releasing the button to allow them to slowly decelerate.
8. Do the same by pressing the **LEFT** button to make the dome turn anticlockwise.
9. If the fuse blows while this test is performed, replace the amperage of the fuse by 5 amps repeat the above steps. Do not exceed a 30 amp fuse.

If the system does not run with a 30 amp fuse then this would suggest a fault with motors or the dome is stuck.

### Bluetooth Connection

This test ensures the **DomeBuddy Controller** can connect to the **DomeBuddy Dongle** via Bluetooth. The Bluetooth module in each have been paired previously so if one is replaced then they need to be paired again. See the section **Programming The Bluetooth Module**.  
  
The **Controller’s** module is set to Master mode and the **Dongle’s** is in Slave mode.

1. Turn off the **DomeBuddy Controller** if it is already turned on.
2. Disengage the motors from the dome
3. Ensure the **DomeBuddy Dongle** is plugged into either of the **AUX1** and **AUX2** ports on the telescope’s control panel.
4. Turn on the telescope to power up the **Dongle**.
5. After 5 seconds the **Dongle’s** LED will flash rapidly to confirm that it is waiting to make a Bluetooth connection to the **Controller**.
6. Turn off the telescope and **Dongle**.
7. Now turn on the **Controller**. Two half second beeps will be given and the LED will remain lit constantly.
8. Momentarily push the **PARK** button and release immediately once it beeps. The LED will start flashing to indicate that is waiting to make a Bluetooth connection with the **Dongle**.
9. Now turn on the telescope again so that the **Dongle** is powered up.
10. After a few seconds, a connection will be made. This is indicated by the **Dongle’s** LED will flash on then off approximately once every 5 seconds, and the **Controller’s** LED will do the same.

If this test is unsuccessful then one of the following reasons could be the case:

* The Bluetooth Module inside either the **DomeBuddy Controller** or **Dongle** may be disconnected from their corresponding Arduino Nanos – check all connections.
* The Bluetooth Module in either is not working
* The Arduino Nano in either is not working.

See the appendices for an explanation of how to how to reconnect the Bluetooth Modules or Arduino Nanos it they are to be replaced.

## Calibrating the Dome

This is a very important procedure. If **DomeBuddy** does not know how many encoder reference points there are in one rotation then it cannot move the dome’s slit into the correct position for the telescope.

This procedure must be done before the first use of **DomeBuddy** and can be periodically updated in the future as required.

The number of encoder reference points are stored into the **DomeBuddy** **Controller’s** EEPROM.

### Calibration Process

1. Turn off the **DomeBuddy Controller**.
2. Ensure the DomeBuddy Dongle is not powered up. Either unplug it from the telescope’s control panel or switch off the telescope.
3. Engage the motors so that their tyres are in contact with the drive surface of the dome.
4. Ensure the rotary encoder is engaged with the drive surface.
5. Make a mark of where the dome is positioned so that it can returned to that same place after is has been moved.
6. While pressing the **PARK** button on the **Controller** box, switch on the **Controller** and keep the **PARK** button pressed for 5 seconds.
7. The **Controller** will give 10 quick beeps as it enters its calibration mode.
8. Now press either the **LEFT** or **RIGHT** buttons to make the dome do one complete rotation.
9. Bring the dome back to its starting position. If you overshoot it, just simply use the **LEFT** and **RIGHT** buttons to bring the dome to it.
10. With the dome back at its start position, press and hold the **PARK** button for 5 seconds.
11. The **Controller** will play a little tune to confirm that it recorded the number of encoder reference points into its EEPROM.
12. The **Controller** will then continue into its ordinary operation mode.
13. To calibrate again, go back to Step 1.

After successful calibration, the **DomeBuddy Controller** will use the stored encoder reference points number when is it switched on in the future.

# Operating Dome Buddy

## Important Notes

* If the **DomeBuddy Controller** is taken out of Park mode then the dome may immediately rotate to the telescope’s azimuth.
* After the telescope has stopped slewing, the dome will always take the shortest route to bring its slit in front of the telescope. This may be in the opposite direction to which the telescope slewed. This is normal behaviour.
* When the dome has completed its rotation, it may momentarily start moving again. This is due to one of three reasons:  
    
   1. The dome overshot its finishing point  
   2. The dome stopped before it finishing point.  
   3. The telescope has tracked to a new azimuth reference point
* If the **DomeBuddy Controller’s** beeper and LED are both active then the **Controller** is unable to detect any movement of the dome. This might mean the dome is stuck or the rotary encoder is not engaged.
* “Parked” means the dome is inactive and not responsive to the position of the telescope.

## Starting Up

1. Make sure the dome is unlocked and free to rotate
2. Ensure the mains outlet and the power supply unit (PSU) are switched on and the **DomeBuddy Controller** is turned off.
3. Ensure the dome is at its home position and the motors and rotary encoder are engaged.
4. Switch on the **Controller** – it will give two half second beeps to indicate that is has successfully powered up.
5. The **DomeBuddy** controller does not have to be powered up before the controller is switched on.

## Session Operation

1. If the **DomeBuddy Controller’s** LED is constantly lit then this indicates that the dome is Park mode.  
     
   Pressing the **PARK** button to unpark the dome and make it active.
2. **IMPORTANT:** If the **DomeBuddy Controller’s** LED is constantly lit and its buzzer is constantly sounding then this indicates that the rotary encoder is not turning. The encoder could either be disengaged or the dome is stuck.

It may be best to switch off the **Controller** to investigate the error, besides the rotary encoder may have lost its position.  
Pressing either the **LEFT** or **RIGHT** buttons will stop the buzzer and reactivate the dome, but it may be best to rotate the dome to its home position and then restart the **Controller**.

1. If the dome is unparked the **Controller’s** LED will start flashing rapidly indicating that it is waiting to connect to the **Dongle** via Bluetooth.
2. If the **Dongle** is already powered up, the **Controller** will connect to it via Bluetooth within a few seconds.  
   Provided the **Dongle** is communicating with the telescope, its LED will flash every 5 seconds to indicate that it has sent the telescope’s azimuth and slew/tracking status to the **Controller**.
3. Once the **Controller** is connected to the **Dongle** and it is unparked, and the telescope is not slewing, the **Controller** will turn the dome so that it’s slit is in front of the telescope.  
     
   The **Controller’s** LED will flash about every 5 seconds to indicate it has received the telescope’s azimuth and slew status information from the **Dongle**.
4. While connected and unparked, the **Controller** will respond to the telescope’s movement whether it is tracking or slewing.
5. If the dome’s slit is not in the correct place, use the **LEFT** or **RIGHT** buttons to bring the slit to the correct position. The **Controller** will automatically synchronise the dome’s position to the telescope’s current azimuth.
6. To park the dome, simply press the **PARK** button. The **Controller’s** LED will become lit constantly to indicate that the dome is parked.
7. Pressing any of the three buttons (**LEFT**, **PARK** and **RIGHT**) while the dome is automatically rotating will stop it immediately. The dome will be deactivated and the **Controller’s** LED will become lit constantly.

## Parking the Dome at its Home Position

1. If the **Controller’s** LED is not lit constantly, press the **PARK** button so that it is.
2. Press and hold the **PARK** button for 5 seconds. The **Controller** will give a long 3 second beep and then rotate the dome to its home.
3. When the dome arrives at its home position the **Controller** will play a little tune to indicate this.
4. It is now safe to switch off the **Controller**. When it is switched on next time, provided the dome has not been moved, the **Controller** will know the azimuth of the dome so saving time.

## Defining the Home Position

1. Ensure the dome is unparked. If the **Controller’s** LED is lit constantly then is parked. Press the **PARK** button to unpark it.
2. Ensure the telescope is switched on and properly synched to the sky so that its azimuth is known.
3. Ensure the dome is properly synched with telescope by using the **LEFT** and **RIGHT** buttons.
4. Press the **PARK** button to park the dome. The **Controller’s** LED will be constantly lit.
5. Rotate the dome to its desired home position using the **LEFT** and **RIGHT** buttons. Making a physical mark on the dome and one of the motor mounts might be useful for future reference.
6. Press and hold the **PARK** button for 5 seconds. The **Controller** will give two one second beeps to confirm that it has stored the home position in its EEPROM memory.
7. The **Controller** will remain in Park mode until the **PARK** button is pressed.

# Updating Arduino Firmware

## Arduino Nano Microcontrollers

The **DomeBuddy Controller** and the **DomeBuddy** dongle each have an Arduino Nano microcontroller inside which controls the behaviour of each. Their behaviour is dependent on the firmware code which has been loaded into their ROMs.  
  
To update the firmware, the appropriate C++ source code, otherwise known as a sketch, has to be compiled and uploaded to the Arduino. This is done by downloading the **Arduino IDE** software to a PC and connecting the Arduino to the PC via a USB cable. The **Controller** needs a male USB A to male USB B cable (e.g. printer USB cable) and the **Dongle** needs a male USB A to male Mini USB cable.

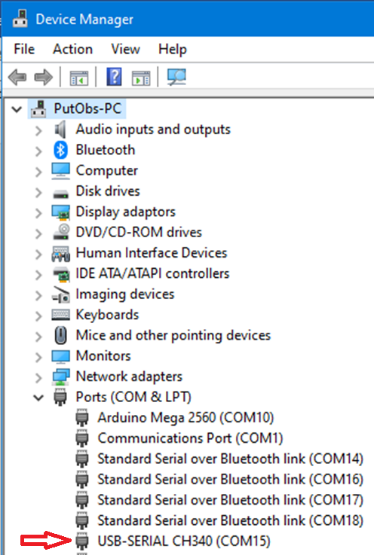
## Using the Arduino IDE

**IMPORTANT:** Make sure the **DomeBuddy** device being worked on is not powered up when making the connection.  
  
If not done so already, download and install the **Arduino IDE** software from here:

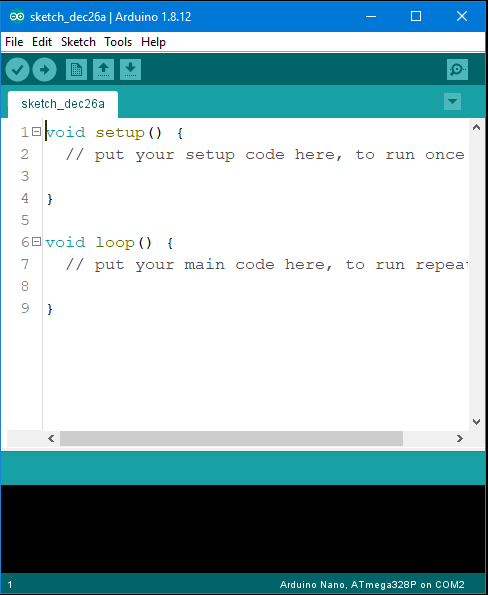
<https://www.arduino.cc/en/software>

**IMPORTANT:** Do not connect both the **Controller** and **Dongle**, and any other Arduinos,at the same time as this can cause confusion and the firmware code could mistakenly be uploaded to the wrong Arduino!

The Arduino Nano to be worked on can be connected to the PC. The USB socket for each **DomeBuddy** device can be found on outside of their enclosures. When connected, the Arduino will be presented on a COM port. On a Windows 10 PC, right click on the Start  button and then click on *Device Manager* and then expand “Ports” to see the Arduino device list there. Take note of which Port number it is on. It may appear as “USB-SERIAL CH340”.

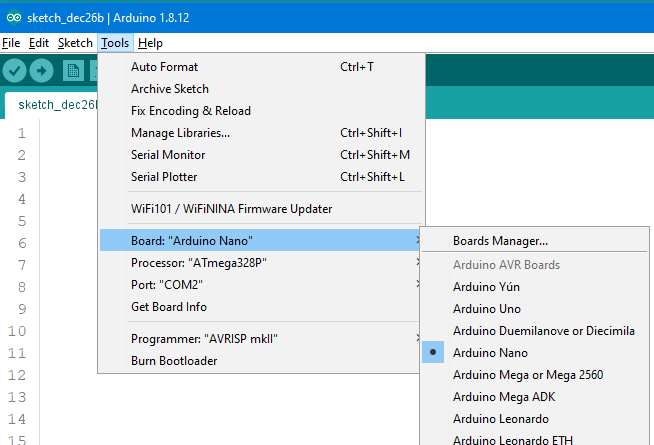


The **DomeBuddy** device will start-up as normal as it is using the USB cable power as a power source – this is normal behaviour.  
  
Store the Arduino sketch in a folder on the PC’s hardrive. Then open the **Arduino IDE**.



Click on **File** then **Open** to find and the firmware file from its folder. Its filename has a **.ino** extension.

Now click on **Tools** then **Board** and select “Nano” from the list.



From the same menu, ensure the Processor is “ATmega328P” and that the Port is set to the COM port the Arduino is connected to.

The **IDE** is now ready to compile and upload to the Arduino. Do this by clicking on **Sketch** then **Upload**, or clicking on the  icon on the toolbar or by pressing **CRTL+U** on the keyboard. The output window at the bottom of the screen will show the status of the uploading process. The **DomeBuddy** device will restart just as it would if it was powered up.  
  
If the **Sketch** did not upload then checking the settings in the **Tools** menu again, an maybe change the Processor to “ATmega328P (Old Bootloader)”.

When all is done, the **IDE** can be closed and the **DomeBuddy** device can be disconnected.

## Changing DomeBuddy Controller Firmware Parameters

There are some parameters in the **DomeBuddy** **Controller** sketch which can be changed to change its operating behaviour. After changing any value will require the sketch to be upload to the **Controller’s** Arduino.

The parameters can be found in the text of the sketch as follows:

// PARAMETERS

#define RAMP\_SPD 10

#define MAX\_SPD 0

#define ENC\_DIR 1

#define MTRA\_DIR 1

#define MTRB\_DIR 1

The values shown above are the defaults values.

### RAMP\_SPD

This is affects the acceleration rate when ramping up the motors. There are 256 stepped speeds from zero speed to full speed. The *RAMP\_SPD* value is the time in milliseconds between steps. A default value of 10ms means that the motors will accelerate to full speed in 2.56 seconds (10ms x 256 = 2.560ms = 2.56 seconds). Changing this value can reduce any slippage of the motor tyres while ramping-up the rotation of the dome.

The *RAMP\_SPD* is also used to determine the rate of deceleration to bring the motors to a stop.

### MAX\_SPD

As mentioned above, there are 256 steps from zero to full speed. 0 means 256 which is full speed. 128 is half maximum speed, 64 is quarter of maximum speed.  
  
Changing this value will therefore change the maximum speed of the dome rotation. This may be necessary of the motor tyres slip too excessively at a high speed.

### ENC\_DIR

If when the dome rotates to the telescope’s position after slewing it ramps-up to fully speed and then it immediately ramps-down to zero speed and then keeps repeating this, then the most probable reason is that the direction of the dome is not being measured correctly by the encoder.  
  
The default value of 1 assumes the dome rotates to the right (clockwise) when it needs to slew to the right. If it does not do this in one complete motion means this *ENC\_DIR* needs to be set to 0.

### MTRA\_DIR

If Motor A turns in the wrong direction to what is intended then the polarity of the current to that motor in wrong way round. This can be simply fixed by switching the spade connectors round.  
  
If this is not possible then changing *MTRA\_DIR* to 0 will.

### MTRB\_DIR

If Motor B turns in the wrong direction to what is intended then the polarity of the current to that motor in wrong way round. This can be simply fixed by switching the spade connectors round.  
  
If this is not possible then changing *MTRB\_DIR* to 0 will.

## Changing DomeBuddy Dongle Firmware Parameters

There are some parameters in the **DomeBuddy** **Dongle** sketch which can be changed to change its operating behaviour. After changing any value will require the sketch to be upload to the **Dongle’s** Arduino.

The parameters can be found in the text of the sketch as follows:

// PARAMETERS

#define ENQ\_DELAY 5000

The values shown above are the defaults values.

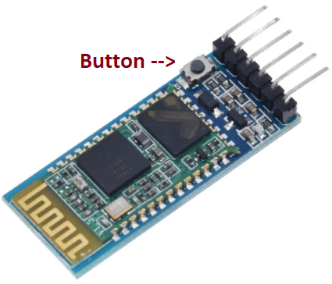
### ENQ\_DELAY

This parameter determines how often the **Dongle** asks the Celestron NexStar telescope for the its azimuth and whether it is slewing or not.

The value is in milliseconds and cannot be less than 600.

# Programming the Bluetooth Module

The **DomeBuddy Controller’s** Bluetooth module needs to pair with **DomeBuddy Dongle’s** Bluetooth module so that a wireless connection can be made and so that it can receive from the **Dongle** information about the status of the telescope. Both devices use an **HC-05** module. The **Controller’s** module runs in Master modes and the **Dongle’s** in Slave mode.  
  
This section only needs to be read if one of the modules has failed and is being replaced with a new one. The replacement one should have a button as shown in the picture so that the module can go into AT mode and allow AT commands to be accepted.



Using the **Arduino IDE**, the following sketch needs to be uploaded to the Arduino which will be doing the programming.  
  
#include <SoftwareSerial.h>SoftwareSerial btSerial(12, 11); // 12 = HC-05's RXD, 11 = HC-05's TXDvoid setup() { btSerial.begin(38400); Serial.begin(9600); Serial.println("\nBluetooth Module Programmer");}void loop() { if(Serial.available()) btSerial.write(Serial.read()); if(btSerial.available()) Serial.write(btSerial.read());}

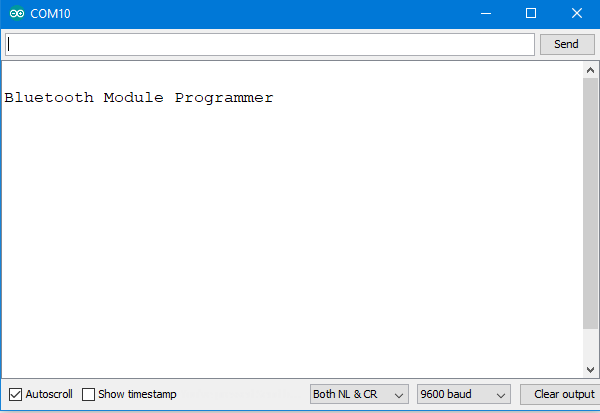
The pins of the **HC-05** module need to be connected to the Arduino as follows:  
  
 RXD 🡨 D12

TXD 🡪 D11

GND to GND

VCC to +5V

To make the **HC-05** enter its AT mode, disconnect the Arduino from it’s USB cable, press and hold the module’s button while reconnecting the USB cable.  
  
Next go into the **Arduino’s IDE** Serial Monitor screen by clicking on **Tools** then **Serial Monitor**, or pressing Ctrl+Shift+M on the keyboard.



Make sure the “Both NL & CR” and “9600 baud” are selected at the bottom of the window.

Type “AT” and the click **Send** or press the Enter key. If all is good the **HC-05** should respond in the window “OK”. If not check the wiring connections, the code that has been entered or disconnect the USB cable and reconnect it while pressing the **HC-05**’s button.

## Programming DomeBuddy Controller’s HC-05

Enter the following commands. The **HC-05** button must be pressed while the ENTER key is being pressed:

1. AT+ORGL *Restores default values*
2. AT+ROLE= 1 *Put into MASTER mode*
3. AT+CMODE=0 *Connect to Bluetooth device with BIND address*
4. AT+BIND=98D3,71,F5F856 ***DomeBuddy Dongle’s*** *Bluetooth address*
5. AT+PSWD=1234 ***DomeBuddy Dongle’s*** *pin code*
6. AT+UART=38400,0,0 *UART setting (38400 baud)*
7. AT+NAME=DomeBuddy Controller

*Gives the module this name*

The **DomeBuddy Dongle’s** address given in Step 4. is the address of the original **HC-05** supplied. Otherwise, the current **HC-05** has to be used instead.

## Programming DomeBuddy Buddy’s HC-05

Enter the following commands. The **HC-05** button must be pressed while the ENTER key is being pressed:

1. AT+ORGL *Restore default values*
2. AT+NAME=DomeBuddy Dongle *Give the module this name*
3. AT+ROLE=0 *Put into SLAVE mode*
4. AT+PSWD=1234 *This module’s pin code*
5. AT+UART=38400,0,0 *UART setting (38400 baud)*
6. AT+ADDR? *Get module’s address*

The address retrieved in Step 5. must be used as the AT+BIND address in Step 4. of programming the **DomeBuddy Controller’s** **HC-05** module.

# Diagnostics

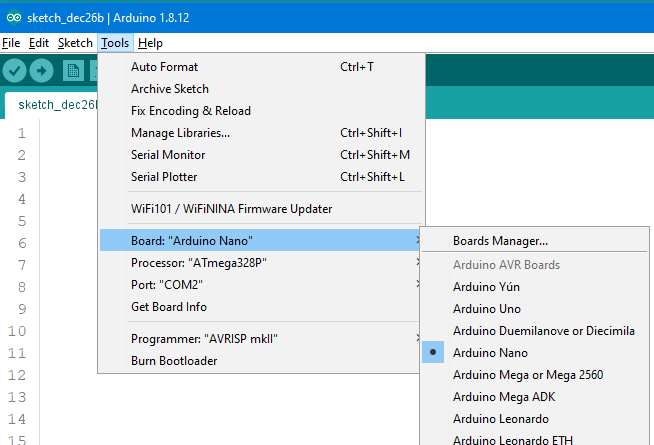
Both of the **DomeBuddy Controller** and **DomeBuddy Dongle** have diagnostic and testing features which are useful troubleshooting the **DomeBuddy** system.

## DomeBuddy Controller

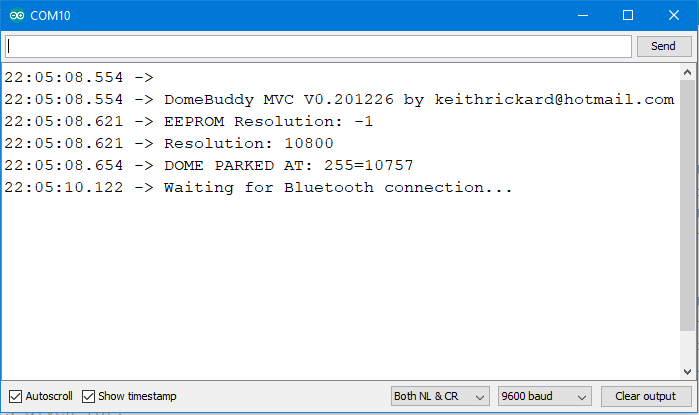
### Diagnostics

To see the diagnostic output of the **DomeBuddy Controller**, connect the **Controller** to the PC with a USB cable and then open the **Arduino IDE** on the PC.

Open **Tools** and ensure the Port is set to the COM port that the **Controller** is connect to on the PC.



Open the **Serial Monitor** from the **Tools** menu or by pressing Ctrl+Shift+M. Make sure “9600 baud” is selected. Also, ticking “Show timestamp” may help further.



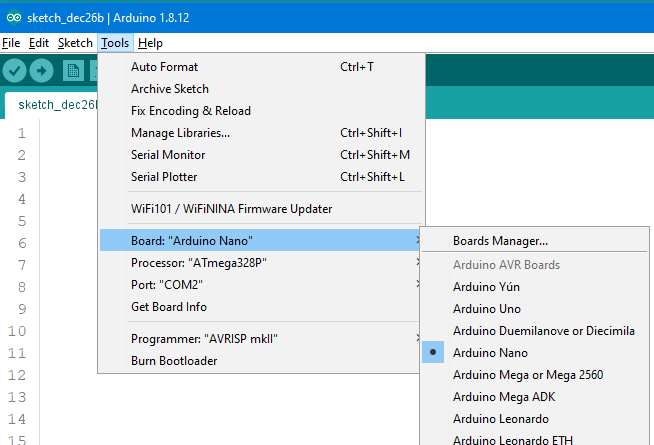
From this point onwards, the **Controller** can be used as normal. Any action the **Controller** makes will be shown up on the **Serial Monitor** screen.  
  
**Hint:** Where a string of digits is given in the form “255=10757”, the 255 is the azimuth reference point (0 – 255) and the 10757 is the encoder count for the current position of the dome.

## DomeBuddy Dongle

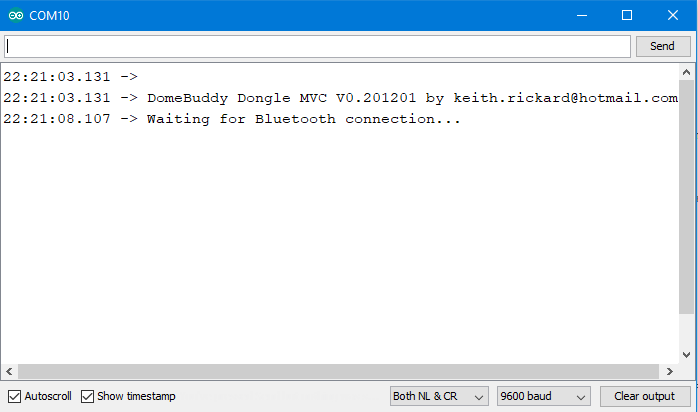
### Diagnostics

To see the diagnostic output of the **DomeBuddy Dongle**, connect the **Dongle** to the PC with a USB cable and then open the **Arduino IDE** on the PC.

Open **Tools** and ensure the Port is set to the COM port that **Dongle** is connected to on the PC.



Open the **Serial Monitor** from the **Tools** menu or by pressing Ctrl+Shift+M. Make sure “9600 baud” is selected. Also, ticking “Show timestamp” may help further.



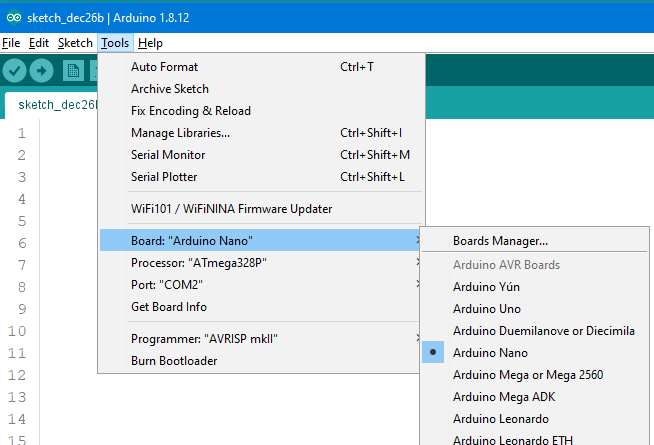
From this point onwards, the **Dongle** can be used as normal. Any action the **Dongle** makes will be shown up on the **Serial Monitor** screen.

### Test Mode

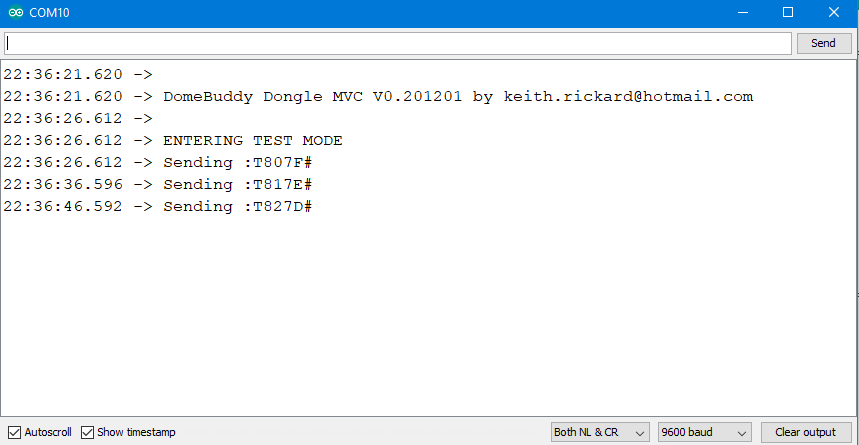
To enter the **DomeBuddy Dongle’s** test mode first connect the **Dongle** to the PC with a USB cable and then open the **Arduino IDE** on the PC.

If the **DomeBuddy Controller** is connected to the **Dongle** then it will respond to it and carry out any action with moving the dome accordingly.

Open **Tools** and ensure the Port is set to the COM port that the **Dongle** is connected to on the PC.



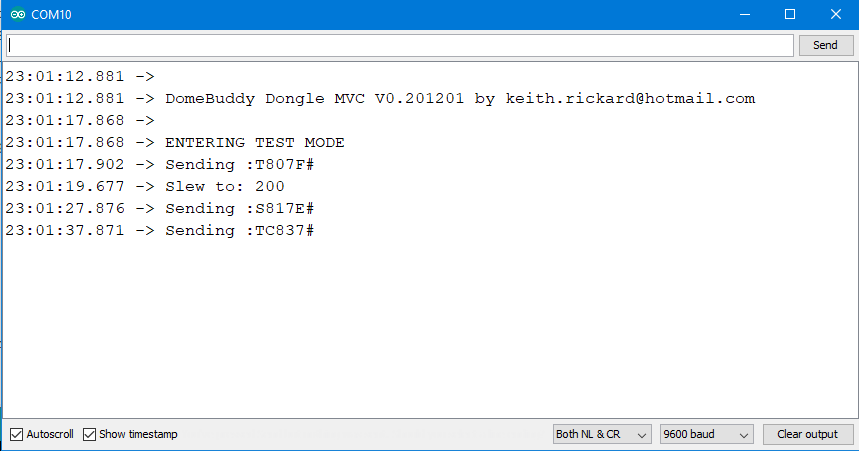
Open the **Serial Monitor** from the **Tools** menu or by pressing Ctrl+Shift+M. Make sure “Both NL & CR” and “9600 baud” are selected. Also, ticking “Show timestamp” may help further.  
  
Now enter “T” in the command bar and click **Send** within 5 seconds of opening the window. If you fail to do this, close the **Serial Monitor** window, re-open it and try again.



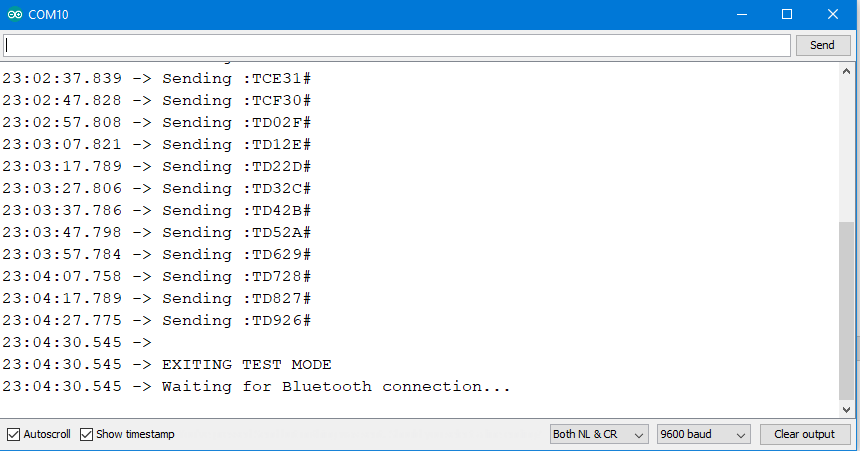
The **Dongle** will now emulate the telescope. The “telescope” will be tracking at a faster rate. It will move from one azimuth reference point (0 – 255) every 10 seconds. The telescope will start a due south, azimuth reference point 128.  
  
You will see lines like “Sending :T807F#” being shown in the **Serial Monitor** window. An explanation of the this data is as follows:

|  |  |
| --- | --- |
| : | Start marker of the data being sent. |
| T | **T** means the “telescope” is tracking. **S** means it is slewing. |
| 80 | Hexadecimal for 128, the current azimuth reference point. |
| 7F | Hexadecimal for -128. This ensures data has been received correctly by the **DomeBuddy Controller**. |
| # | End marker of the data being sent. |

The user can enter a number from 0 to 255 in the *Command* bar at the top of the **Serial Monitor** window to make the “telescope” slew to this azimuth reference point.



It will take 20 seconds to conduct a test telescope slew. To exit the Test Mode, send an “X” in the command bar.



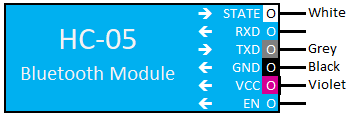
# DomeBuddy Components

## DomeBuddy Controller

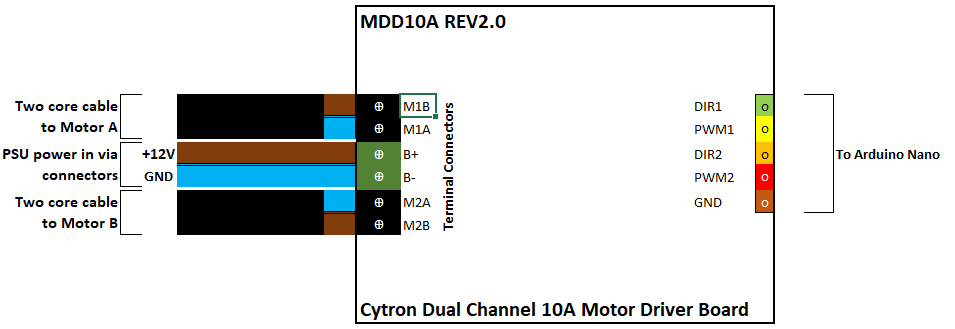
### Arduino Nano



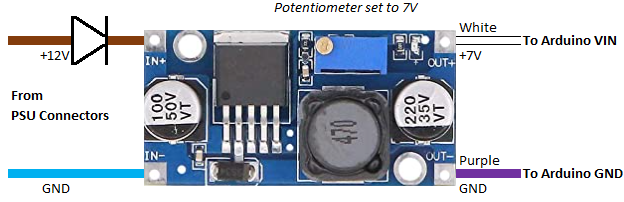
### HC-05 Bluetooth Module



### Cytron Motor Driver



### DC-DC Converter



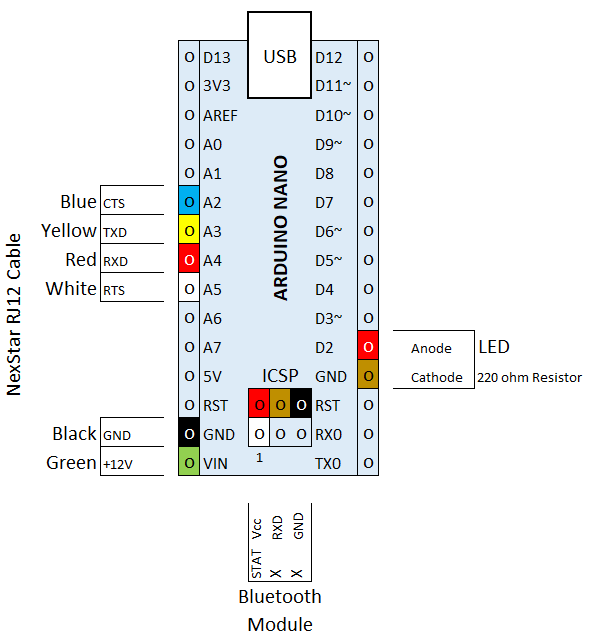
### Connectors

3-way and 5-way Wago connectors.

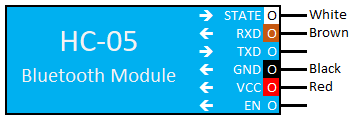
|  |
| --- |
|  |
|  |
|  |

## DomeBuddy Dongle

### Arduino Nano



### HC-05 Bluetooth Module



### RJ12 6P6C plug

|  |  |
| --- | --- |
|  | **NexStar AUX1 port pinout**   1. Blue CTS 2. Yellow TXD 3. Green +12V 4. Red RXD 5. Black GND 6. White RTS |

## Motors and Rotary Encoder

### Windscreen Wiper Motor

|  |  |  |
| --- | --- | --- |
|  |  | Connector |
| Windscreen wiper motor for Nissan Qashqai 2007 onwards (28800-JD000) | Wiring Diagram | 1. Slow speed 2. Fast speed 3. No connection 4. Home switch 5. Common |

### Rotary Encoder

|  |  |
| --- | --- |
|  | Wiring  Black Ground Red Vcc (+5V to +24V) Green A phase (ChA) White B phase (ChB) |
| Wisamic 600P Incremental Rotary Encoder  with 2 phase quadrature output |  |

### Rotary Encoder Plug and Socket

|  |  |  |
| --- | --- | --- |
|  |  | 1 ChA White  2 Vcc Red  3 ChB Green  4 GND Black |
| Looking into the plug | Looking to the socket |  |

# Sketches (Source Code)

## DomeBuddy Controller

#define VERSION "\nDomeBuddy Controller MVC V0.201226by keithrickard@hotmail.com"

// Work started 20 October 2020. For the Arduino Nano.

// PARAMETERS

#define RAMP\_SPD 10

#define MAX\_SPD 0

#define ENC\_DIR 1

#define MTRA\_DIR 1

#define MTRB\_DIR 1

#include <SoftwareSerial.h>

#include <EEPROM.h>

#define BTBAUD 38400

#define ENC\_ChA 2

#define ENC\_ChB 3

#define DIR1 4

#define PWM1 5

#define PWM2 6

#define DIR2 7

#define MTR\_GND 8

#define LED\_GND 9

#define LED 10

#define btTXD 11

#define btSTATUS 12

#define btRXD A7

#define BUZZER A0

#define BTN\_L A1

#define BTN\_R A2

#define BTN\_P A3

#define GND1 A4

#define GND2 A5

#define PRESSED 1

#define RELEASED 0

long enc360 = 10800; // Estimated number of encoder counts per complete dome rotation 1350/75 = 18 \* 600 = 10800

long enc180;

volatile long currEnc;

long lastEnc;

byte homeAZM;

int scopeAZM; // 0 = N, 64 = E, 128 = S, 192 = W.

byte domeAZM;

byte stuck ;

byte parked;

byte ledFlash = LOW;

byte btCxnMsg;

byte calibrating = 0;

SoftwareSerial btSerial(btTXD, btRXD); // Bluetooth Module is in Master mode & bound with DomeBuddy Dongle's address

//==============================================================================================================

void setup() {

pinMode(ENC\_ChA, INPUT\_PULLUP);

pinMode(ENC\_ChB, INPUT\_PULLUP);

pinMode(DIR1, OUTPUT);

pinMode(PWM1, OUTPUT);

pinMode(DIR2, OUTPUT);

pinMode(PWM2, OUTPUT);

pinMode(MTR\_GND, OUTPUT);

pinMode(LED, OUTPUT);

pinMode(LED\_GND, OUTPUT);

pinMode(BUZZER, OUTPUT);

pinMode(BTN\_P, INPUT\_PULLUP);

pinMode(BTN\_R, INPUT\_PULLUP);

pinMode(BTN\_L, INPUT\_PULLUP);

pinMode(GND1, OUTPUT);

pinMode(GND2, OUTPUT);

pinMode(btSTATUS, INPUT);

digitalWrite(GND1, LOW);

digitalWrite(GND2, LOW);

digitalWrite(MTR\_GND, LOW);

digitalWrite(DIR1, LOW);

digitalWrite(PWM1, LOW);

digitalWrite(DIR2, LOW);

digitalWrite(PWM2, LOW);

digitalWrite(BUZZER, HIGH);

digitalWrite(LED, HIGH);

digitalWrite(LED\_GND, LOW);

btSerial.begin(38400); // Open connection to Bluetooth module

btSerial.setTimeout(500);

Serial.begin(9600);

Serial.println(VERSION);

attachInterrupt(digitalPinToInterrupt(ENC\_ChA), encISR, RISING); // encISR monitors encoder movement

if (checkButton(BTN\_P, PRESSED, 0)) calibrateDome(); // If PARK button is being pressed, calibrate dome resolution

scopeAZM = domeAZM = homeAZM = EEPROM[0]; // Get azimuth of the parked Dome

EEPROM.get(1, enc360);

Serial.print("EEPROM Resolution: ");

Serial.println(enc360);

if (enc360 < 0) enc360 = 10800; // 600 \* 1350 / 75

// enc360 = 73728; // 4096 \* 1350 / 75

enc180 = enc360 /2;

currEnc = azm2enc(domeAZM); // Synchronise encoder reading

Serial.print("Resolution: ");

Serial.println(enc360);

Serial.print("DOME PARKED AT: ");

Serial.print(homeAZM);

Serial.print("=");

Serial.println(currEnc);

stuck = 0;

parked = 1;

btCxnMsg = 1;

buzz(500);

delay(500);

buzz(500);

}

//==============================================================================================================

void loop() {

if (digitalRead(btSTATUS)) { // Check to see if a Bluetooth connection has been made

digitalWrite(LED, stuck); // Light the LED steadily if dome stuck

if (btCxnMsg) Serial.println("Bluetooth connected");

btCxnMsg = 0;

}

else {

digitalWrite(LED, ledFlash = 1 - ledFlash); // No Bluetooth connection so flash the LED rapidly

if (btCxnMsg) Serial.println("Waiting for Bluetooth connection...");

btCxnMsg = 0;

}

if (parked) digitalWrite(LED, HIGH); // Light LED if parked

digitalWrite(BUZZER, stuck); // Sound buzzer if dome is stuck

getScopeAZM(); // Get any update of the telescope's azimuth

domeAZM = enc2azm(normEnc()); // Normalise currEnc then map domeAZM from value 0 - 255

if (!parked) moveDome(scopeAZM); // Move dome if necessary and if not parked

manualMove(BTN\_R); // See if RIGHT (CW) button is pressed to turn dome clockwise

manualMove(BTN\_L); // See if LEFT (CCW) button is pressed to turn dome counter clockwise

parkButton(); // See if PARK button is being pressed and act on it is so

}

//==============================================================================================================

void getScopeAZM() { // Get and interpret scope's azimuth from DomeBuddy Dongle via Bluetooth

if (!digitalRead(btSTATUS)) return; // Return immediately if not connected to DomeBuddy Dongle via Bluetooth

if (!btSerial.available()) return; // Return if no received data present

digitalWrite(LED, HIGH);

//Arduino should receive via Bluetooth {':', S' or 'T', Xh, Xl, -Xh, -Xl, '#'} i.e. 7 characters

char buf[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

int value, chksum;

do {

while (btSerial.peek() >= 0){ // Find the sentence start marker ':'

if (btSerial.peek() == ':') break; // Found the marker so break out

if (btSerial.peek() < 0) return; // Return if timed out

btSerial.read(); // Dump the invalid character in the receive buffer

} // Loop back and try again

btSerial.readBytesUntil("#", buf, 7); // Get azmiuth info (7 bytes)

} while (btSerial.peek() >= 0);

Serial.print("\nDome Dongle: "); // Report what has been received from Dome Dongle

Serial.println(buf);

if (buf[1] != 'T') { // 'T' = tracking, return if not.

Serial.println("TELESCOPE SLEWING");

return;

}

value = buf[2] - ((buf[2] >= 'A') ? 55 : 48); // Get telescope azimuth into value (from hex)

value = value \* 16 + buf[3] - ((buf[3] >= 'A') ? 55 : 48);

chksum = buf[4] - ((buf[4] >= 'A') ? 55 : 48); // Calculate the checksum (from hex)

chksum = chksum \* 16 + buf[5] - ((buf[5] >= 'A') ? 55 : 48);

chksum = ~chksum & 255;

if (value == chksum){

scopeAZM = value; // Azimuth has been received correctly

Serial.print("TELESCOPE AZIMUTH= ");

Serial.println(scopeAZM);

}

}

//==============================================================================================================

void calibrateDome(){

Serial.println("CALIBRATING DOME");

buzz(500); delay(500); buzz(500); delay(500); buzz(500); // Give 3 beeps

calibrating = 1;

enc360 = 0x7FFFFFF; // Set dome resolution to a really high number!

Serial.println("Updating dome encoder resoultion...");

do {

currEnc = 0; // Set the encoder value to 0 before waiting for buttons to be pressed

while(checkButton(BTN\_P, RELEASED, 0)){ // Keep looping while the PARK button is not being pressed

digitalWrite(LED, HIGH); // Keep the LED lit

manualMove(BTN\_R); // See RIGHT (CW) button is being pressed and turn dome accordingly

manualMove(BTN\_L); // See LEFT (CCW) button is being pressed and turn dome accordingly

}

buzz(10); // The PARK button has been pressed so beep

} while(checkButton(BTN\_P, RELEASED, 5000)); // Loop back if the PARK button was pressed for less than 5 secs

Serial.println("DOME CALIBRATED!");

Serial.print("Resolution: ");

Serial.println(abs(currEnc));

buzz(50); delay(200); buzz(50); delay(700);

buzz(50); delay(200); buzz(50); delay(200); buzz(50); delay(450);

buzz(50); delay(200); buzz(50); delay(200); buzz(50); delay(200); buzz(50); delay(200);

delay(500); buzz(50); delay(200); buzz(50);

EEPROM.put(1, abs(currEnc)); // Success! Save the new value of the dome resolution

delay(1000);

}

//==============================================================================================================

void parkButton() {

if (checkButton(BTN\_P, RELEASED, 0)) return; // Return immediately if PARK button not pressed

buzz(10);

if (checkButton(BTN\_P, PRESSED, 5000)){ // If button is pressed > 5 secs then try to home sync the dome

if(domeAZM == homeAZM) {

parked = 0;

buzz(1000);

Serial.println("DOME ACTIVE");

return;

}

parkDome(); // Otherwise park/unpark the dome

return;

}

// SYNC HOME POSITIION -----------------------------------------------------------------------------------------

buzz(10);

EEPROM.update(0, homeAZM = domeAZM); // Set home azimuth with the dome azimuth

Serial.print("HOME POSITION SET ");

Serial.print(homeAZM);

Serial.print("=");

Serial.println(currEnc);

buzz(1000); // Sound buzzer to confirm sync

delay(500);

buzz(1000);

}

//==============================================================================================================

void parkDome() {

if (!parked) { // Is dome parked?

Serial.print("DOME PARKED ");

Serial.print(domeAZM);

Serial.print("=");

Serial.println(currEnc);

buzz(1000);

parked = 1; // Dome now parked

return;

}

Serial.println("Parking dome to home position...");

buzz(3000);

do {

moveDome(homeAZM); // Move dome to home position

if (stuck) break; // Parking aborted if dome is stuck

} while (domeAZM = enc2azm(currEnc) != homeAZM); // If dome not quite at the home azimuth - try again

parked = 1; // Park the dome

if(stuck) Serial.println("ABORTED - dome appears to be stuck");

else {

Serial.print("PARKED AT HOME POSITION ");

Serial.print(homeAZM);

Serial.print("=");

Serial.println(currEnc);

delay(500);

buzz(50); delay(200); buzz(50); delay(75); buzz(50); delay(75);

buzz(50); delay(200); buzz(50); delay(450); buzz(50); delay(200); buzz(50);

}

}

//==============================================================================================================

void manualMove(byte button) {

if (checkButton(button, RELEASED, 0)) return; // Return if button not pressed

if (parked) {

Serial.println("DOME ACTIVE");

btCxnMsg = 1;

parked = 0;

buzz(1000);

return;

}

byte dir = (button == BTN\_L); // dir is 0 if button is R (CW) or 1 if L (CCW)

buzz(10);

Serial.print("MANUAL MOVE ");

Serial.print(dir ? "L (CCW) from " : "R (CW) from ");

Serial.print(domeAZM);

Serial.print("=");

Serial.print(currEnc);

stuck = 0;

// START-UP ------------------------------------------------------------------------------------------------

byte pwm = 0;

digitalWrite(DIR1, MTRA\_DIR ? dir: 1 - dir); // Select direction on the motor control board

digitalWrite(DIR2, MTRB\_DIR ? dir: 1 - dir); // dir = 0 is R (CW), dir = 1 is L (CCW)

do { // Ramp-up the motors

analogWrite(LED, pwm);

analogWrite(PWM1, pwm);

analogWrite(PWM2, pwm);

pwm++;

if (checkButton(button, RELEASED, 0)) break; // Break out if a button is no longer pressed

delay(RAMP\_SPD);

} while (pwm != MAX\_SPD); // Exit if maximum speed reached (pwm = 0 is 256)

// CONTINUE --------------------------------------------------------------------------------------------------

if (!pwm) analogWrite(pwm, 255);

while (checkButton(button, PRESSED, 0)); // Keep the dome turning while button is being pressed

// SLOW DOWN -------------------------------------------------------------------------------------------------

do{ // Slow down the motors

pwm--;

analogWrite(LED, pwm);

analogWrite(PWM1, pwm);

analogWrite(PWM2, pwm);

delay(RAMP\_SPD);

} while (pwm); // When pwm is 0 then the motors are turned off

// FINISH-UP --------------------------------------------------------------------------------------------------

do { // Wait for the dome to finally stop moving

lastEnc = currEnc;

delay(500);

} while (lastEnc != currEnc);

if(!calibrating) domeAZM = enc2azm(normEnc()); // Normalise currEnc between 0 and enc360 - 1 and update domeAZM

Serial.print(" to ");

Serial.print(domeAZM);

Serial.print("=");

Serial.println(currEnc);

scopeAZM = domeAZM;

buzz(10);

}

//==============================================================================================================

bool moveDome(byte targAZM) {

if (stuck || (domeAZM == targAZM)) return 0; // Return if stuck or target is same as current dome azimuth

// INITIALISE ------------------------------------------------------------------------------------------------

long targEnc = azm2enc(targAZM); // Convert target (0 - 255) into encoder reference point

byte pwm = 0;

long distEnc = targEnc - currEnc; // Get distance to travel

Serial.print("\nAUTO MOVING from ");

Serial.print(domeAZM);

Serial.print("=");

Serial.print(currEnc);

Serial.print(" to ");

Serial.print(targAZM);

Serial.print("=");

Serial.println(targEnc);

Serial.print("distEnc#1 ");

Serial.println(distEnc);

if (abs(distEnc) > enc180) { // Is the distance more than 180 degs?

if (currEnc < enc180) currEnc += enc360; // If currEnc is less than 180 degs then 360 degs needs to be added

else targEnc += enc360; // Otherwise 360 degs needs to be added to the targEnc

distEnc = targEnc - currEnc; // Recalculate distance (now <= 180 degs)

}

long midPoint = targEnc - distEnc / 2; // Calc mid point reference point between start and end postions

byte dir = (distEnc < 0); // Get direction (0 = right (CW), 1 = left (CCW) )

lastEnc = currEnc; // Remember starting encoder position

// START-UP --------------------------------------------------------------------------------------------------

digitalWrite(DIR1, MTRA\_DIR ? dir : 1 - dir); // Select direction on the motor control board

digitalWrite(DIR2, MTRB\_DIR ? dir : 1 - dir);

unsigned long timer = 0;

lastEnc = currEnc; // Remember starting encoder position

Serial.print("enc360 ");

Serial.println(enc360);

Serial.print("currEnc#1 ");

Serial.println(currEnc);

Serial.print("targEnc#1 ");

Serial.println(targEnc);

Serial.print("distEnc#2 ");

Serial.println(distEnc);

Serial.print("midPoint ");

Serial.println(midPoint);

Serial.print("dir ");

Serial.print(dir);

Serial.println(dir ? "=L (CCW)" : "=R (CW)");

Serial.println("RAMPING-UP");

do { // Ramp-up the motors

if (checkBtns()) break; // A button has been pressed so stop immediately

if (!dir) if (currEnc > midPoint) break; // Mid point check for RIGHT (CW)

if (dir) if (currEnc < midPoint) break; // Mid point check for LEFT (CCW)

analogWrite(LED, pwm);

analogWrite(PWM1, pwm); // Update motors' speed

analogWrite(PWM2, pwm);

pwm++;

delay(RAMP\_SPD); // Delay creates a rate of ramping-up

} while (pwm != MAX\_SPD); // Exit ramping-up if halfway there or maximum speed reached

// SLEW ------------------------------------------------------------------------------------------------------

if (dir) targEnc += (currEnc - lastEnc); // Amend targEnc to account for time to slow down (assumed to same as ramp-up)

else targEnc -= (currEnc - lastEnc);

if(!pwm) analogWrite(LED, pwm);

Serial.print("currEnc#2 ");

Serial.println(currEnc);

Serial.print("targEnc#2 ");

Serial.println(targEnc);

Serial.println("SLEWING");

Serial.print("pwm ");

Serial.println(pwm);

while (!(parked = checkBtns())) { // Monitor the moving dome until time to slow down & no buttons have been pressed

if (!dir) if (currEnc > targEnc) break; // See if it is time to start slowing down the dome RIGHT (CW)

if (dir) if (currEnc < targEnc) break; // LEFT (CCW)

if (millis() > timer) { // Check periodically to see if the dome is moving

timer = millis() + 1000;

if (!isMoving()) break; // Is dome moving? Stop the dome immediately if not

}

}

// SLOW DOWN -------------------------------------------------------------------------------------------------

Serial.println("SLOWING DOWN");

do { // Slow down the motors

pwm--;

analogWrite(LED, pwm);

analogWrite(PWM1, pwm);

analogWrite(PWM2, pwm);

delay(RAMP\_SPD);

} while (pwm); // When pwm is 0 then the motors are turned off

// FINISH-UP -------------------------------------------------------------------------------------------------

Serial.print("STOPPED");

do { // Wait for the dome to finally stop moving

lastEnc = currEnc;

delay(500);

} while (lastEnc != currEnc);

domeAZM = enc2azm(normEnc()); // Normalise currEnc and update domeAZM

Serial.print(" at ");

Serial.print(domeAZM);

Serial.print("=");

Serial.println(currEnc);

return domeAZM != targAZM; // Return 0 if domeAZM is same as target azimuth, 1 if not

}

//==============================================================================================================

byte isMoving() {

long chkEnc = lastEnc; // Get last known position

lastEnc = currEnc;

stuck = 0;

// return ; // REMOVE THIS LINE FOR FINAL VERSION!!!

if (abs(chkEnc - currEnc) > 10) return 1; // Good news! Dome is rotating.

parked = stuck = 1; // Dome is stuck

Serial.println("DOME NOT MOVING! Please resolve issue.");

return 0;

}

//==============================================================================================================

bool checkButton(byte button, bool state, unsigned long delayTime) {

//state = 0 test releasing of button, = 1 test pressing of button

delayTime += millis(); // Wait upto delayTime ms for button being released/pressed before quitting

do {

if (!digitalRead(button) == state) { // Has button been released/pressed? (button pin: 1 = released, 0 = pressed)

delay(50); // Wait 50ms for debounce

if (!digitalRead(button) == state) return 1; // Test button again - return if it is still released/pressed

}

} while (millis() < delayTime);

return 0; // Button not released/pressed within delay time, return 0.

}

//==============================================================================================================

bool checkBtns() { // Look at all buttons. Returns 1 if a button is being pressed.

return !digitalRead(BTN\_P) || !digitalRead(BTN\_R) || !digitalRead(BTN\_L);

}

//==============================================================================================================

void buzz(int duration) {

digitalWrite(LED, HIGH); // Sound buzzer and light LED for 'duration' milliseconds

digitalWrite(BUZZER, HIGH);

delay(duration);

digitalWrite(BUZZER, LOW);

digitalWrite(LED, LOW);

}

//==============================================================================================================

long normEnc() {

long value = currEnc; // Noramlises currEnc between 0 and enc360 - 1

return currEnc = value % enc360 + ((value < 0) ? enc360 : 0);

}

//==============================================================================================================

byte enc2azm (long enc){

return (enc << 8) / enc360;

}

//==============================================================================================================

long azm2enc (byte azm){

return (azm \* enc360) >> 8;

}

//==============================================================================================================

void encISR() {

#ifdef ENC\_DIR

(PIND & B1000) ? currEnc++ : currEnc--; // PIND & B1000 is pin D3 (ENC\_ChB)

#else

(PIND & B1000) ? currEnc-- : currEnc++; // PIND & B1000 is pin D3 (ENC\_ChB)

#endif

// When ENC\_ChA rises from 0 to 1, ENC\_ChB = 1 means RIGHT (CW), ENC\_ChB = 0 means LEFT (CCW)

}

//==============================================================================================================

## DomeBuddy Dongle

#define VERSION "\nDomeBuddy Dongle MVC V0.201201 by Keith Rickard"

// Work started 19 October 2020. For Arduino Nano and Celestron NexStar GPS telescope.

// PARAMETERS

#define ENQ\_DELAY 5000

#include <SoftwareSerial.h>

#define nsCTS A2

#define nsTXD A3

#define nsRXD A4

#define nsRTS A5

#define btRXD 11

#define btTXD A7

#define btSTAT 12

#define LED 2

#define BTBAUD 38400

#define NSBAUD 19200

#define CMD\_AZM "\x3B\x03\x04\x10\x01\xE8"

#define CMD\_AZM\_HEX "3B 03 04 10 01 E8"

#define CMD\_AZM\_END "\xEB"

#define CMD\_SLEW "\x3B\x03\x04\x10\x12\xD7"

#define CMD\_SLEW\_HEX "3B 03 04 10 12 D7"

#define CMD\_SLEW\_END "\xD7"

#define CMD\_PREAMBLE "\x3B"

// 3Bh = preamble

// 03h = packet length

// 04h = hand controller device (source)

// 10h = azimuth motor controller AZM (destination)

// 01h = MC\_GET\_POSITION (get azimuth) - returns 24 bits

// E8h = checksum - LSB( NOT(03h + 04h + 10h + 01h) + 1)

long scopeAZM;

char scopeSLEW = 'T';

char buf[10];

byte btCxnMsg;

SoftwareSerial btSerial(btTXD, btRXD); // Serial object for Bluetooth module (btTXD is not used so it is assigned to a "useless pin")

SoftwareSerial nsSerial(nsTXD, nsRXD); // Serial object for the Celestron NexStar GPS AUX1 or AUX2 port

//===============================================================================================================================

void setup() {

btSerial.begin(BTBAUD);

pinMode(nsCTS, INPUT\_PULLUP); // Arduino's "RTS"

pinMode(nsRTS, OUTPUT); // Arduino's "CTS

pinMode(LED, OUTPUT); // LED indicator

digitalWrite(nsRTS, HIGH); // RTS is inactive so set HIGH (Arduino has no data to send yet)

Serial.begin(9600);

Serial.println(VERSION);

btSerial.begin(BTBAUD);

nsSerial.begin(NSBAUD);

nsSerial.setTimeout(5000);

btCxnMsg = 1;

delay(5000);

if(Serial.available()) if(Serial.read() == 'T') testMode(); // If 'T' is received, go into TEST mode

}

//===============================================================================================================================

void loop() {

if(!digitalRead(btSTAT)){

Serial.println("Waiting for Bluetooth connection...");

while(!digitalRead(btSTAT)) { // See if a Bluetooth connection to DomeBuddy has been made

digitalWrite(LED, HIGH); // If not, rapidly flash the LED

delay(100);

digitalWrite(LED, LOW);

delay(100);

}

btCxnMsg = 1;

}

if (btCxnMsg) {

Serial.println("Bluetooth connected");

btCxnMsg = 0;

}

digitalWrite(LED, HIGH); // Turn on LED to show comms are in progress

getAZM(); // Ask NexStar for the telesope's slew status and azimuth

sendAZM(); // If got, send the slew status and azimuth to DomeBuddy Controller

delay(500);

digitalWrite(LED, LOW); // Turn off LED - azimuth sent! LED should only remain on for ~500ms

delay(ENQ\_DELAY - 600); // Wait about 5 seconds before looping back

}

//===============================================================================================================================

void testMode(){

digitalWrite(LED, HIGH);

Serial.println("\nENTERING TEST MODE");

char c = '\0';

byte testAZM = 128; // Assume initially test scope is pointing due south

byte testTarg = testAZM;

byte testNeg = -testAZM;

char testSLEW = 'T';

unsigned long testTimer = millis();

do{

if(millis() > testTimer){

testTimer = millis() + 10000;

testNeg = ~testAZM;

Serial.print("Sending ");

Serial.print(':'); // Azimuth is being sent, 0 degrees is due north

Serial.print(testSLEW); // Send slewing info ('S' = slewing, 'T' = tracking)

if(testAZM < 16 ) Serial.print('0'); // Send Azimuth as two hexdigit number

Serial.print(testAZM, HEX); // Send only the HSB (0 - 255) i.e. 256 point resolution

if(testNeg < 16 ) btSerial.print('0'); // Send negative Azimuth as two hexdigit number

Serial.print(testNeg, HEX); // Sending this ensures azimuth is sent correctly

Serial.println('#'); // Send terminator

btSerial.listen(); // Select btSerial port

btSerial.print(':'); // Azimuth is being sent, 0 degrees is due north

btSerial.print(testSLEW); // Send slewing info ('S' = slewing, 'T' = tracking)

if(testAZM < 16 ) btSerial.print('0'); // Send Azimuth as two hexdigit number

btSerial.print(testAZM, HEX); // Send only the HSB (0 - 255) i.e. 256 point resolution

if(testNeg < 16 ) btSerial.print('0'); // Send negative Azimuth as two hexdigit number

btSerial.print(testNeg, HEX); // Sending this ensures azimuth is sent correctly

btSerial.print('#'); // Send terminator

if(testSLEW == 'T') testAZM++; // Simulate tracking (accelerated!) by moving "clockwise" N->E->S->W->N

if(testAZM == testTarg) testSLEW = 'T';

}

if(Serial.available()){

if(isDigit(Serial.peek())){ // See if an azimuth value (0-255) has been sent by the user.

testTarg = Serial.parseInt();

Serial.print("Slew to: ");

Serial.println(testTarg);

testSLEW = 'S';

}

else c = Serial.read();

}

} while (c != 'X');

Serial.println("\nEXITING TEST MODE");

}

//===============================================================================================================================

void getAZM(){

//Returns scopeSLEW as 'S' or 'T' and scopeAZM as the scope's azimuth 0-255 (0 is due north)

nsSerial.listen(); // Select nsSerial port

int chksum;

byte nsAZM;

// GET AZIMUTH ------------------------------------------------------------------------------------------------------------------

do {

Serial.print("Getting Azimuth - sending ");

Serial.print(CMD\_AZM\_HEX);

Serial.print("...");

digitalWrite(nsRTS, LOW); // Activate NexStar's RTS line to say "Ready To Send data"

while (digitalRead(nsCTS) && digitalRead(btSTAT)); // Wait for NexStar to say "Clear for you To Send data" (CTS = LOW when ready)

if (!digitalRead(btSTAT)) { // Bluetooth still connected?

Serial.println("\nBluetooth connection lost");

return;

}

Serial.println(" sent\nWaiting for response from NexStar");

nsSerial.write(CMD\_AZM); // Send command to ask for Azimuth (6 bytes)

findPreamble();

nsSerial.readBytesUntil(CMD\_AZM\_END, buf, 10); // Get echo of the command (6 bytes) - terminator is discarded when receiving

buf[5] = CMD\_AZM\_END; // Put terminator back at end received data

SerialPrintBufHex('E',6); // Report contents of buf

findPreamble();

nsSerial.readBytes(buf, 8); // Get AZM's response (8 bytes)

// buf = 0x3B, 0x05, 0x10, 0x04, HSB, MSB, LSB, chksum

SerialPrintBufHex('R', 8); // Report contents of buf

digitalWrite(nsRTS, HIGH); // Deactivate NexStar's RTS line

} while (!checkSumCalc(6)); // Loop back if checksum has failed

nsAZM = buf[4]; // Pick out only the top 8 bits of the 24 bit azimuth value

// GET SLEW STATUS --------------------------------------------------------------------------------------------------------------

do{

Serial.print("Getting Slew status - sending ");

Serial.print(CMD\_SLEW\_HEX);

Serial.print("...");

digitalWrite(nsRTS, LOW); // Activate NexStar's RTS line to say "Ready To Send data"

while (digitalRead(nsCTS) && digitalRead(btSTAT)); // Wait for NexStar to say "Clear for you To Send data" (CTS = LOW when ready)

if (!digitalRead(btSTAT)) { // Bluetooth still connected?

Serial.println("\nBluetooth connection lost");

digitalWrite(nsRTS, HIGH); // Deactivate NexStar's RTS line - Arduino not ready to send

return;

}

nsSerial.write(CMD\_SLEW); // Send command to ask if scope is slewing (6 bytes)

Serial.println(" sent\nWaiting for response from NexStar");

findPreamble();

nsSerial.readBytesUntil(CMD\_SLEW\_END, buf, 10); // Get echo of the command (6 bytes) - terminator is discarded when receiving

buf[5] = CMD\_SLEW\_END; // Put terminator back at end of received data

SerialPrintBufHex('E', 6); // Report contents of buf

findPreamble();

nsSerial.readBytes(buf, 10); // Get AZM's response (6 bytes expected)

// buf = 0x3B, 0x03, 0x10, 0x04, 0x00 or 0xFF, chksum

SerialPrintBufHex('R', 6); // Report contents of buf

digitalWrite(nsRTS, HIGH); // Deactivate NexStar's RTS line

} while (!checkSumCalc(4)); // Loop back if checksum has failed

scopeSLEW = buf[4] ? 'T' : 'S'; // 'S' = slewing (0x00), 'T' = tracking (0xFF)

scopeAZM = nsAZM;

Serial.print("Telescope ");

Serial.println((scopeSLEW = 'T') ? "tracking" : "slewing");

Serial.print("Azimuth ");

Serial.println(scopeAZM);

}

//===============================================================================================================================

void findPreamble() {

for (byte i = 0 ; i <= 9; i++) buf[i] = 0; // Clear buffer

do { // Find the sentence start marker CMD\_PREAMBELE

if (Serial.peek() == CMD\_PREAMBLE) return; // Found the marker so break out

if (Serial.peek() < 0) return; // Nothing left in receive buffer

} while(Serial.read()); // Dump the invalid character in the receive buffer & oop back and try again

}

//===============================================================================================================================

void SerialPrintBufHex(byte m, byte n){

Serial.print((m == 'R') ? "Received" : "Echo ");

for(byte i = 0; i < n; i++){

Serial.print(" ");

if(buf[i] < 16) Serial.print("0");

Serial.print(buf[i], HEX);

}

Serial.println();

}

//===============================================================================================================================

bool checkSumCalc(byte n){

if(buf[0] != 0x3B){

Serial.println("Invalid preamble - should be 3B");

return 0;

}

int chkSum = 0;

for(byte i = 1; i <= n; i++){ // Bytes in buf[1] through to buf[n] are added to determine checksum.

chkSum += buf[i];

}

chkSum = -chkSum & 0xFF; // Calculate two's complement of the total and keep the lower 8 bits only.

Serial.print("Checksum: ");

if(chkSum < 16) Serial.print("0");

Serial.print(chkSum, HEX);

Serial.println((chkSum == buf[n+1]) ? " - success" : " - failed");

return chkSum == buf[n+1];

}

//===============================================================================================================================

void sendAZM(){

//scopeSLEW is 'S' or 'T' and scopeAZM is the scope's azimuth 0-255 (0 is due north)

if(digitalRead(btSTAT)) return; // Return if no active Bluetooth connection

long negAZM = (~scopeAZM) & 0xFF; // Calculate NOT of scopeAZM

// Send ':', 'S' or 'T', Xh, Xl, -Xh, -Xl, '#'

Serial.print("Transmitting azimuth data ");

// SEND AZIMUTH TO SERIAL PORT

Serial.print(':'); // Azimuth is being sent, 0 degrees is due north

Serial.print(scopeSLEW); // Send slewing info ('S' = slewing, 'T' = tracking)

if(scopeAZM < 16 ) Serial.print('0'); // Send Azimuth as two hexdigit number

Serial.print(scopeAZM, HEX); // Send only the Highest Significant Byte (0 - 255) i.e. 256 point resolution

if(negAZM < 16 ) Serial.print('0'); // Send negative Azimuth as two hexdigit number

Serial.print(negAZM, HEX); // Sending this ensures azimuth is sent correctly

Serial.println('#'); // Send terminator

// SENT AZIMUTH TO BLUETOOTH

btSerial.listen(); // Select btSerial port

btSerial.print(':'); // Azimuth is being sent, 0 degrees is due north

btSerial.print(scopeSLEW); // Send slewing info ('S' = slewing, 'T' = tracking)

if(scopeAZM < 16 ) btSerial.print('0'); // Send Azimuth as two hexdigit number

btSerial.print(scopeAZM, HEX); // Send only the HSB (0 - 255) i.e. 256 point resolution

if(negAZM < 16 ) btSerial.print('0'); // Send negative Azimuth as two hexdigit number

btSerial.print(negAZM, HEX); // Sending this ensures azimuth is sent correctly

btSerial.print('#'); // Send terminator

}

/\*===============================================================================================================================