**Motor Controller on dsPIC33FJ Notes**

1. **Only Altitude and Azimuth are implemented by the Motor Controllers**
   1. Right Ascension and Declination are calculated in the ASCOM .dll on the controlling PC, the telescope (HUB and Motor Controllers use ONLY Altitude and Azimuth, or RA and DEC converted into Altitude and Azimuth. Azimuth is synonymous to Right Ascension, Altitude is synonymous with Declination, but are converted to Earth coordinates. Azimuth and Altitude are coordinates local to the telescope.
2. **I/O Connections**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Altitude Controller** | | | **Azimuth Controller** |
|  | **Pin** | **Colour** | **Signal** | **Signal** |
| a | RD0 | Yellow | Pulse Pin | Pulse Pin |
| b | RD1 | Blue | Direction Pin | Direction Pin |
| c | RG6 | Green | QEI Channel A | QEI Channel A |
| d | RG7 | White | QEI Channel B | QEI Channel B |
| e | RD8 | Yellow | Button Left Green |  |
| f | RD9 | Blue | Button Right Green |  |
| g | RD10 | Yellow | Button Left Red |  |
| h | RD11 | Blue | Button Right Red |  |
| i | RF0 | Yellow | Sector Sensor | Sector Sensor |
| j | RF1 | Orange/Red | Home Sensor | Home Sensor |
| k | RF2 | Blue | 7 Segment U1RX |  |
| l | RF3 | Yellow | 7 Segment U1TX |  |

1. **Home Switches**
   1. Left Front (Green) Altitude Motor find Home
   2. Right Front (Green) Azimuth Motor find Home
   3. Left Rear (Red) Altitude Motor immediate Halt
   4. Right Rear (Red) Azimuth Motor immediate Halt
2. **Calculation of Output Waveform**
   1. Clock Speed: 40,000,000
   2. Prescaler (T2): 1:1
   3. PR2: Motordrive.period
   4. OC1R: Motordrive.pulse\_msw – Pulse goes low
   5. OC1RS: Motordrive.pulse\_lsw – Pulse goes high (1/2 of PR2)
   6. For Slew, PR3=0x00, PR2=0x0100, OC1R=0x00, OC1RS=0x0080
      1. 1/40,000,000 = 0.000000025 seconds
      2. Period = 0x0100 x 0.000000025 = 0.0000064 = 6.4us
      3. Pulse = 0x0080 x 0.000000025 = 0.0000032 = 3.2us
3. **Raising Errors**
   1. Illegal instruction errors, such as sending a Slew command when already slewing are trapped by the .dll. It follows that no errors, except for status updates, will be returned by the motor drivers.
4. **Motor Backlash**
   1. In order to compensate for the delay between the optical sensor sensing movement and the start of the output shaft turning a short amount of motor drive is applied, and therefore the motor turns, prior to the optical sensor input being used to measure the motor rotation. This backlash results from the flexibility of the final belt drive between the final output shaft and the shaft on which the optimal sensor is mounted.
   2. There are four types of backlash applied dependent on the current direction of rotation and the to be applied direction:

|  |  |  |  |
| --- | --- | --- | --- |
| Current Direction | New Direction | Backlash Type | Count\* |
| CW | CW | 1 | 0 |
| CW | ACW | 2 | 0 |
| ACW | CW | 3 | 0 |
| ACW | ACW | 4 | 0 |

* 1. Count will be updated after experimentation once the telescope is complete.

1. **Significance of LEDs on Explorer 16 Board**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Altitude Controller** | | **Azimuth Controller** |
|  | **LED** | **Significance if Lit** | **Significance if lit** |
| a | LED3 | Programme running, in main loop | Programme running, in main loop |
| b | LED4 |  |  |
| c | LED5 |  |  |
| d | LED6 | Update sent to HUB | Update sent to HUB |
| e | LED7 | Command received from HUB | Command received from HUB |
| f | LED8 | QED signal change | QED signal change |
| g | LED9 | Motor Direction Anticlockwise | Motor Direction Anticlockwise |
| h | LED10 | Motor Direction Clockwise | Motor Direction Clockwise |

1. **Visible Rotation of Motors**
   1. Altitude Motor - Output shaft of 50:1 gearbox rotates clockwise, final output shaft rotates clockwise.
   2. Azimuth Motor – Output shaft of 50:1 gearbox rotates clockwise, final output shaft rotates clockwise.
2. **Communications from ASCOM**
   1. Communications from the ASCOM DLL to the Arduino HUB are via REST, passed as a string command to the respective command. The string contains 4 comma delimited fields, each containing a text version of a double value. Conventionally, in commands containing bearings, the first field is bearing.
3. **Command Codes**
   1. Within the communications packet from the HUB

|  |  |  |
| --- | --- | --- |
| **Command** | **Code** | **Sync Number** |
| AZ\_Home\_command | 0x10 | i1 |
| Az\_Halt\_command | 0x12 | i2 |
| Halt\_command | 0x14 | i3 |
| Azimuth\_Display\_command | 0x16 | i4 |
| Motor\_update\_command | 0x05 | C1 |
| TargetAltitude\_command | 0xA0 | C2 |
| TargetAzimuth\_command | 0xA2 | C3 |
| Tracking\_command | 0xA4 | C4 |
| TrackingRates\_command | 0xA6 | C5 |
| AbortSlew\_command | 0xB0 | C6 |
| FindHome\_command | 0xB2 | C7 |
| Park\_command | 0xB4 | C8 |
| SlewToAltAz\_command | 0xC0 | C9 |
| SlewToTarget\_command | 0xC8 | C10 |
| SyncToAltAz\_command | 0xD0 | C11 |
| PulseGuide\_command | 0xE0 | C12 |
| GuideRateRightAscension\_command | 0xE2 | C13 |
| GuideRateDeclination\_command | 0xE4 | C14 |
| UnPark\_command | 0xB6 | C15 |

1. **Motor Update, C1**
   1. An update message is sent from each motor controller, via RS232, every 500 milliseconds, to the HUB in order that the HUB can give instantaneous replies to ASCOM queries, without the necessity to poll the respective motor driver.
   2. Structure of Motor Update Packet, from HUB to Motor Controller and Motor Controller to HUB

typedef struct {

unsigned char header; [0] STX

unsigned char command\_number; [1]

unsigned char status\_byte; [2]

union sg\_double azimuth\_bearing; [3 - 6] - contains Altitude or Azimuth

union sg\_double altitude\_bearing; [7 - 10]

union sg\_double field3; [11 - 14]

union sg\_double field4; [15 - 18]

unsigned char footer; [19] ETX

} drivermessage;

* 1. Structure of status byte, sent by the Motor Controller to the HUB within the Motor Update Packet.

typedef struct {

unsigned char running: A0

unsigned char slewing: A1

unsigned char tracking: A2

unsigned char homeing: A3

unsigned char pulseguiding: A4

unsigned char athome: A5

unsigned char atpark: A6

unsigned char change: A7

} motorstatus;

* 1. The source of the Packet is determined within the HUB, by the hardware port from which it is received.

1. **Target Altitude, C2**
   1. The Target Altitude command is forwarded by the HUB only to the Altitude motor and contains a bearing to which, at some future point, will be used by a “Slew\_to\_Target” command, C10, which will be applied to motors.
2. **Target Azimuth, C3**
   1. The Target Azimuth command is forwarded by the HUB only to the Azimuth motor and contains a bearing to which, at some future point, will be used by a “Slew\_to\_Target” command, C10, which will be applied to both motors.
3. **Tracking and Tracking Rate, C4 and C5**
   1. Tracking (true) instructs the mount to commence tracking at the motor speeds defined in TrackingRates\_command(TrackingRateAzimuth\_value, TrackingRateAltitude\_value. During tracking the dll will update the tracking rates to reflect changes in the calculated RA and DEC World values due to the time changing.
   2. The TrackingRates sent reflects the calculated microstep pulses per second to be achieved by the motor, to convert the calculated TrackingRate values to microstep pulses per second the TrackingRate will be multiplied within the dsPIC software, by RA\_SPEED\_ADJUST, or DEC\_SPEED\_ADJUST values as appropriate.
   3. Once received the TrackingRates are applied during a T2 interrupt on the dsPIC.
4. **Slewing Methodology, C9 and C10**
   1. Slewing instructs the respective motor driver to start the motor and run until a specific QED reading is achieved.
   2. TargetAltitude\_value and TargetAzimuth\_value contain the QED values that will have been recorded when the respective motor achieves the desired location. The QED value is calculated by subtracting the current location from the desired location and then multiplying by a factor, below is the C code which achieved the calculation:
   3. long Calculate\_Target\_QED(double Current\_Bearing, double Target\_Bearing) {

double bearingdifference = Current\_Bearing - Target\_Bearing;

if (bearingdifference < 0) {

Motordrive.dir = COUNTERCLOCKWISE;

} else Motordrive.dir = CLOCKWISE; return Target\_Bearing \* (double) DEGREE2QED;

}

1. **Seven Segment Display**
   1. Arduino Nano Pin Out:

|  |  |  |
| --- | --- | --- |
| Pin | Signal | Colour |
| 2 | RX from Altitude Driver Board | Blue |
| 3 | TX to Altitude Driver Board | Yellow |
| 7 | Altitude CS | White |
| 8 | Altitude DIN | Green |
| 9 | Altitude CLK | Yellow |
| 10 | Azimuth CS | White |
| 11 | Azimuth DIN | Green |
| 12 | Azimuth CLK | Yellow |

1. **Pulse Guiding**

Pulse Guiding is a method of keeping the telescope pointing towards a defined star in the field of vision of a Spotter scope. A CDD camera image of the sky pointed to by the Spotter scope keeps track of the position of a defined star in its image. As the star appears to move (due to the Earth’s rotation) the movement will be detected by PHD, PHD commands will be sent to the mount to move the telescope to counter this apparent movement, the commands from ASCOM (PHD) (PulseGuide) will contain a direction and a duration, the speed already having been defined using the GuideRateAscension and GuideRateDeclination properties, a double containing the speed, which will be a user (via the ASCOM .dll) defined fraction of SIDEREAL rate to be added to the current tracking speed for the duration (in milliseconds) of the PulseGuide command.

The ASCOM command is of the format :

void PulseGuide(

GuideDirections *Direction*,

int *Duration*

)

where Direction is given by:

0 = North (+ declination/altitude).

1 = South (- declination/altitude).

2 = East (+ right ascension/azimuth).

3 = West (- right ascension/azimuth)

and Duration is in milliseconds.

The rate of motion for movements about the right ascension axis is specified by the [GuideRateRightAscension](http://www.ascom-standards.org/Help/Platform/html/P_ASCOM_DeviceInterface_ITelescopeV3_GuideRateRightAscension.htm) property. The rate of motion for movements about the declination axis is specified by the [GuideRateDeclination](http://www.ascom-standards.org/Help/Platform/html/P_ASCOM_DeviceInterface_ITelescopeV3_GuideRateDeclination.htm) property.

Upon receipt of a PulseGuide command at the HUB, the HUB will:

* 1. Determine the motor (field[0]) to which the Pulse Guide needs to be applied:
     1. 0 and 1 = altitude motor
     2. 2 and 3 = azimuth motor
  2. Send the command to the appropriate motor:

int PulseGuide\_f(String command) { // f12}

Parse\_Command\_to\_Fieldf(command);

switch ((int) command\_fieldf[0]) {

case North:

Send\_Message\_to\_Driver(Altitude\_motor, PulseGuide\_command, command\_fieldf[0], command\_fieldf[1], 0, 0); // send message to driver

break;

case South:

Send\_Message\_to\_Driver(Altitude\_motor, PulseGuide\_command, command\_fieldf[0], command\_fieldf[1], 0, 0); // send message to driver

break;

case East:

Send\_Message\_to\_Driver(Azimuth\_motor, PulseGuide\_command, command\_fieldf[0], command\_fieldf[1], 0, 0); // send message to driver

break;

case West:

Send\_Message\_to\_Driver(Azimuth\_motor, PulseGuide\_command, command\_fieldf[0], command\_fieldf[1], 0, 0); // send message to driver

break;

}

Upon receipt of the modified PulseGuide command the addressed motor will:

1. Save the appropriate variables:
   1. PulseGuide\_Direction = Drivermessage.bearing
   2. PulseGuide\_Duration = Drivermessage.field2
   3. Previous\_Motor\_Speed = Motordrive.period
2. Calculate the new speed:
   1. If PulseGuide\_Direction = 0 speed = Motordrive.period + GuideRateDeclination
   2. If PulseGuide\_Direction = 1 speed = Motordrive.period - GuideRateDeclination
   3. If PulseGuide\_Direction = 2 speed = Motordrive.period + GuideRateRightAscension
   4. If PulseGuide\_Direction = 3 speed = Motordrive.period - GuideRateRightAscension
3. Calculate the new direction:
   1. If PulseGuide\_Direction = 0 (North) direction = CW (Altitude motor)
   2. If PulseGuide\_Direction = 1 (South) direction = ACW (Altitude motor)
   3. If PulseGuide\_Direction = 2 (West) direction = CW (Azimuth motor)
   4. If PulseGuide\_Direction = 3 (East) direction = ACW (Azimuth motor)
4. Turn the Motor On:
   1. Turn\_Motor\_On\_Guiding(speed, direction) {

Set\_Motor\_Direction(direction);

Motordrive.period = speed;

Motordrive.pulse\_rising = Motordrive.period >> 1; // OCR - signal goes high

Motordrive.pulse\_falling = Motordrive.period; // OCRS - signal goes low

InitTimer2();

OC1CONbits.OCM = 0b100; // pin starts low, driven high on OC1P/OC1PS

T2CONbits.TON = 0x01; // Start Timer2

millispulseguiding = 0; // zero the pulseguiding timer

MotorStatus.motorstatus.change = true;

}

* 1. MotorStatus.Motorstatus.pulseguiding = true;

case PulseGuide\_command:

PulseGuide\_Direction = (int) CommandMessage.Drivermessage.bearing; // the required duration, in milliseconds for the Pulse Guide

PulseGuide\_Duration = (long) CommandMessage.Drivermessage.field2; // the direction required for the Pulse Guide, 1 = CW, 0 = ACW

Previous\_Motor\_Speed = Motordrive.period; // save the current direction

MotorStatus.Motorstatus.pulseguiding = true;

millispulseguiding = (long) 0; // zero the pulse guiding timer

switch (PulseGuide\_Direction) {

case NORTH:

Turn\_Motor\_On\_Guiding(Previous\_Motor\_Speed + GuideRateDeclination);

break;

case SOUTH:

Turn\_Motor\_On\_Guiding(Previous\_Motor\_Speed - GuideRateDeclination);

break;

case WEST:

Turn\_Motor\_On\_Guiding(Previous\_Motor\_Speed + GuideRateRightAscension);

break;

case EAST:

Turn\_Motor\_On\_Guiding(Previous\_Motor\_Speed - GuideRateRightAscension);

break;

}

Send\_Reply\_to\_HUB(PulseGuide\_command, good, PulseGuide\_Direction);

break;

1. In the main loop:
   1. If pulseguiding is true
      1. Check if millispulseguiding >= PulseGuideDuration
         1. Motordrive.period = Previous\_Motor\_Speed
         2. Motordrive.dir = Previous\_Motor\_Direction
         3. Turn\_Motor\_On\_Guiding(Motordrive.period, Motordrive.dir)
         4. MotorStatus.motorstatus.pulseguiding = false

if (MotorStatus.Motorstatus.pulseguiding == true) {

if (millispulseguiding >= PulseGuide\_Duration) { // has the specified pulse guiding period expired?

Turn\_Motor\_On\_Guiding(Previous\_Motor\_Speed); // by reloading the saved speed

}

MotorStatus.Motorstatus.pulseguiding = false;

}

1. Focuser

**TRACKING RATES:**

**ARDUINO HUB:**

int TrackingRates\_f(String command) { // f5

Parse\_Command\_to\_Fieldf(command);

#ifdef DEBUG

Serial.print("Setting TrackingRates Azimuth ");

Serial.print(command\_fieldf[0], DEC);

Serial.print(" Altitude ");

Serial.println(command\_fieldf[1], DEC);

#endif

TrackingRateAzimuth\_value = command\_fieldf[0];

TrackingRateAltitude\_value = command\_fieldf[1];

Send\_Message\_to\_Driver(Azimuth\_motor, TrackingRates\_command, TrackingRateAzimuth\_value, 0, 0, 0); // send message to driver

Send\_Message\_to\_Driver(Altitude\_motor, TrackingRates\_command, TrackingRateAltitude\_value, 0, 0, 0);

return (int)true; }

# DSPIC:

case TrackingRates\_command: // variables will contain the micropulse count per second to implement when tracking engaged

Motor\_Speeds.tracking = CommandMessage.Drivermessage.field1;

Command\_Packet\_Reply\_value = Motor\_Speeds.tracking;

Command\_Packet\_Reply\_answer = good;

Command\_Packet\_Reply\_command = TrackingRates\_command;

Command\_Packet\_reply = true;

break;

void \_\_attribute\_\_((\_\_interrupt\_\_, \_\_no\_auto\_psv\_\_)) \_T2Interrupt(void) {

if (MotorStatus.Motorstatus.tracking == true) { // go on forever until cancelled

if (Motordrive.period != Motor\_Speeds.tracking) {

Motordrive.period = Motor\_Speeds.tracking;

PR2 = Motordrive.period; // change the motor speed (pulses per second) to the new value

}