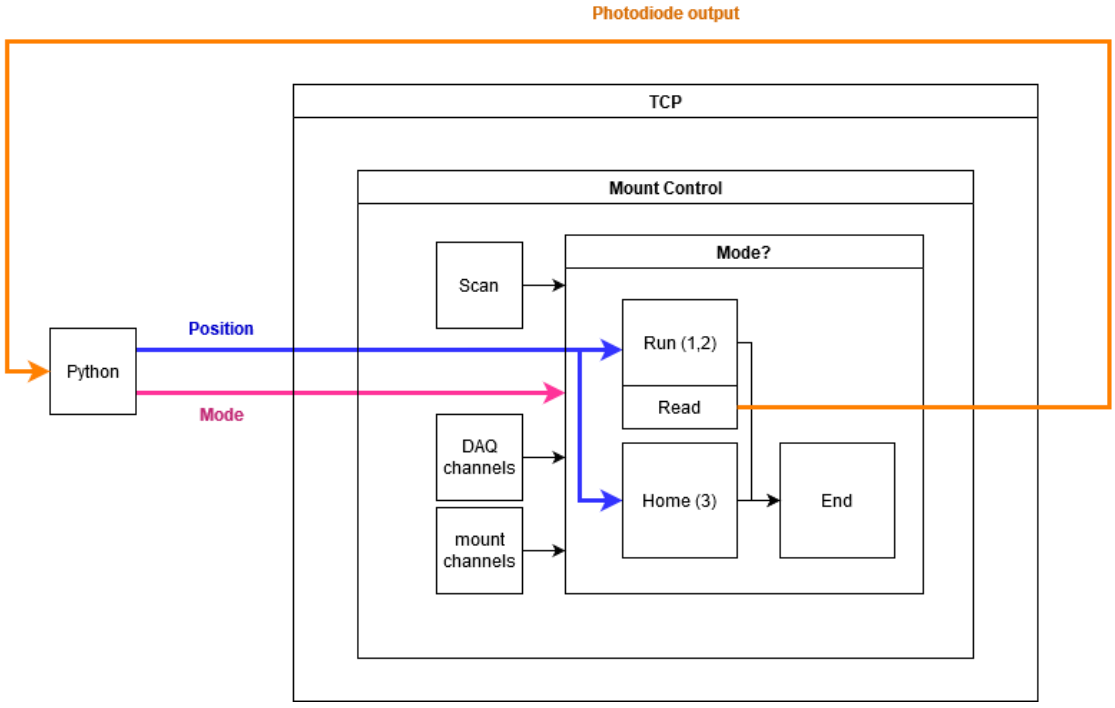
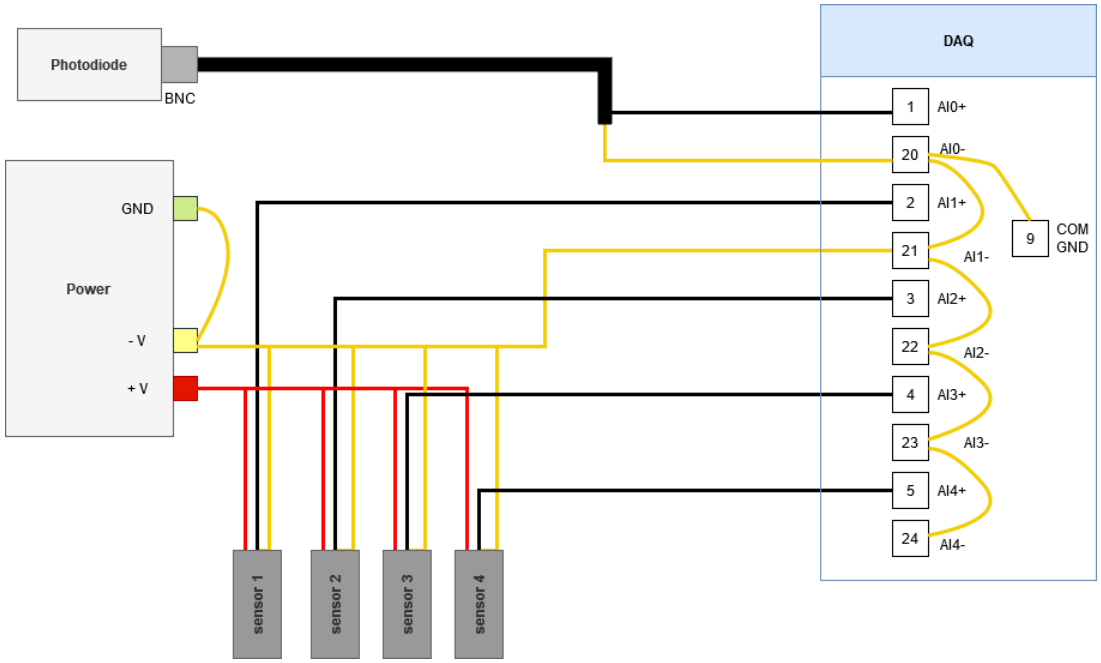
Contents



Software system diagram.

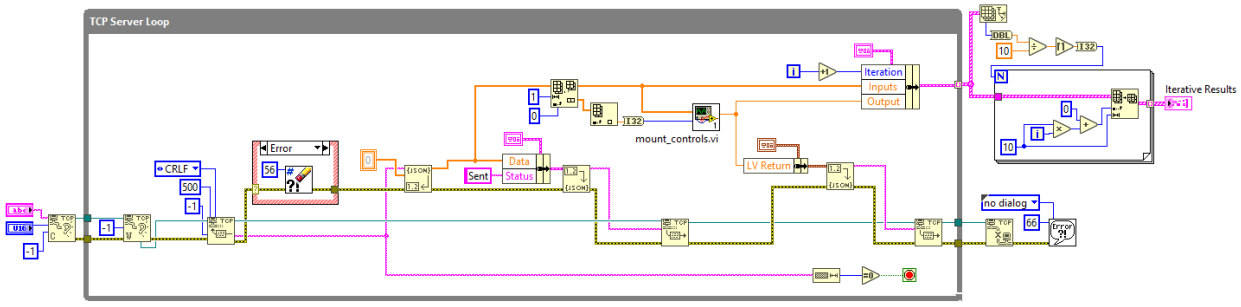


System wiring diagram.

Top Level TCP Communication

TCP Communication

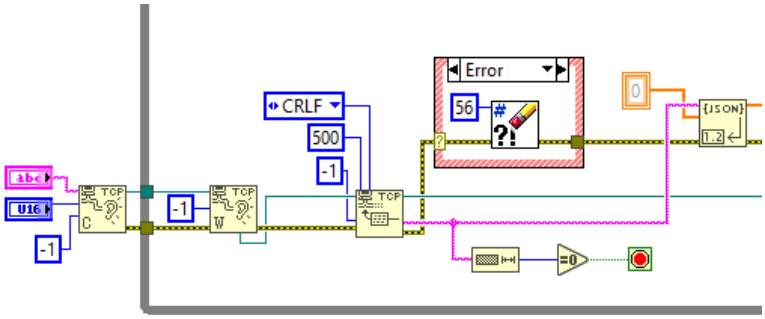
The “TCP_connection” VI is used to send position data from Python file “TCP_bare” to the mount controls subVI. The subVI outputs laser coupling percentage and the value is sent back to Python.



Full TCP Communication VI.

Read Data from Python

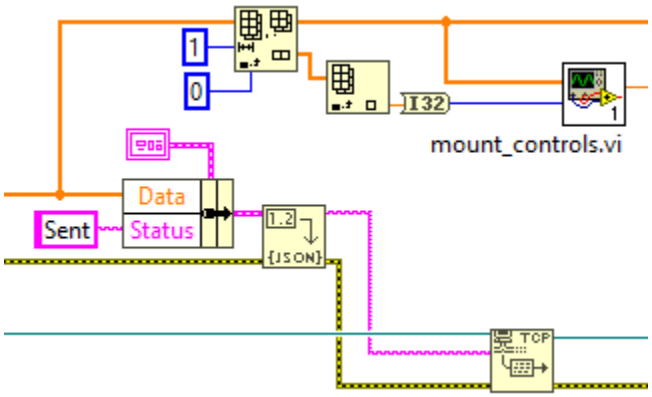
- Initialize by creating TCP listener at the designated port name and #
- Wait on data to be sent from Python
- Read array of position values from Python and unflatten from JSON
    - Ignore error 56 (read timeout)
- If sent data string length = 0, stop VI



Initialization and reading Python data.

Send Data to subVI

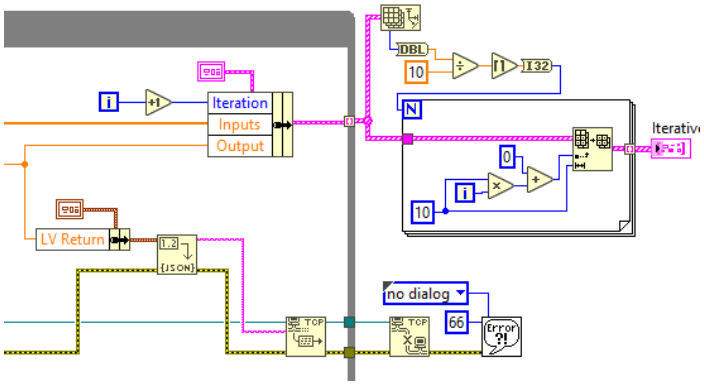
- Flatten data array and “sent” confirmation and return to Python
- Separate first value of sent array to get “Mode” and the position inputs
- Send mode and positions to “mount_controls” subVI and output percent laser coupling



Send “data received” confirmation to Python and send inputs to mount control subVI.

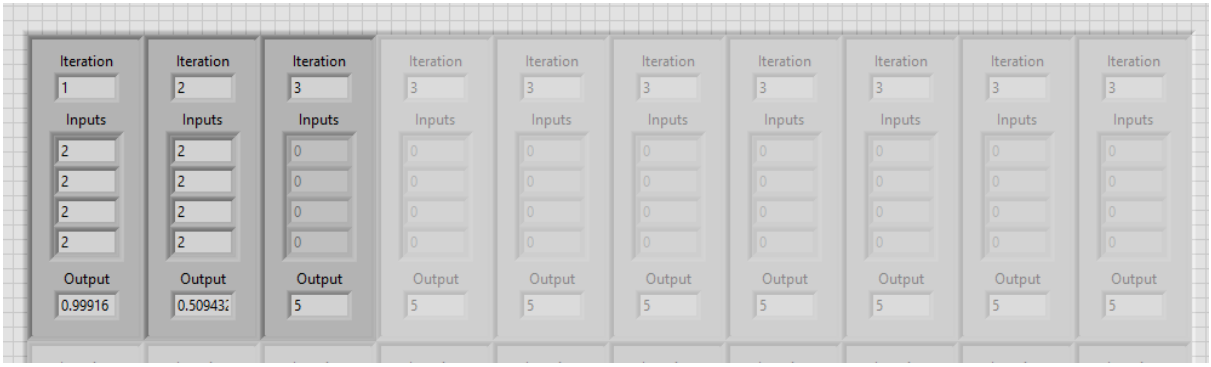
Return Data and Exit

- Send subVI output to Python
- Create cluster of loop iteration #, position inputs, and coupling % output
- Output array of clusters with 10 elements per row
    - N = # of iterations/10 = # of rows
- If error = 66, show no message
    - Connection closed by Python



Send output to Python and display iteration data when VI is closed.

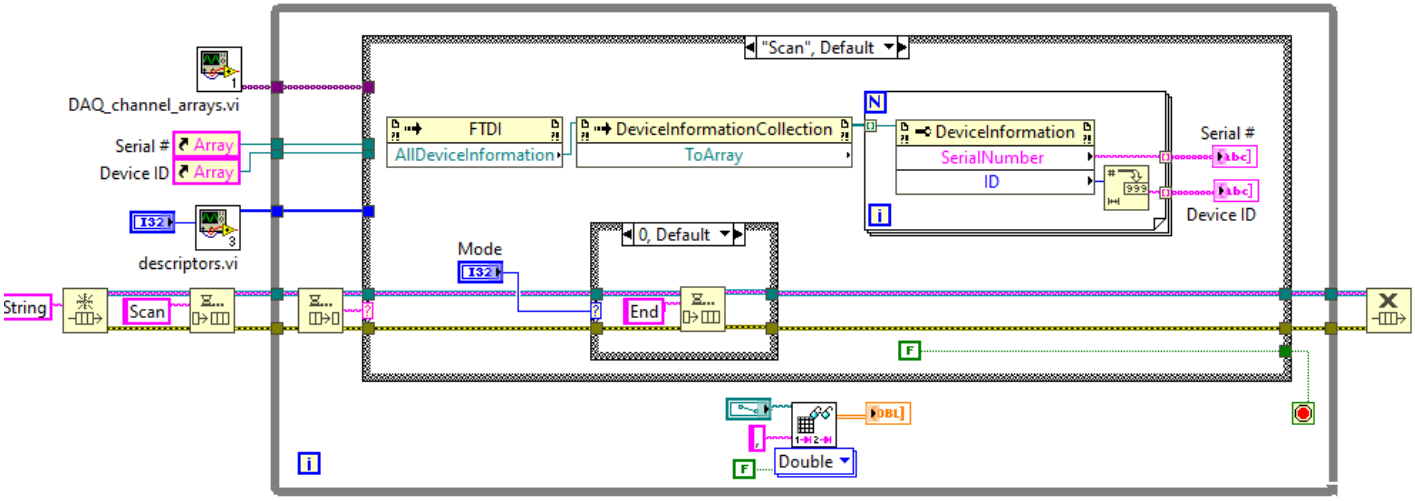
Once the VI stops and exits the loop, the front panel shows the inputs and output for each iteration.



"TCP_connection.vi" front panel with one row data displayed.

Mount Control subVI

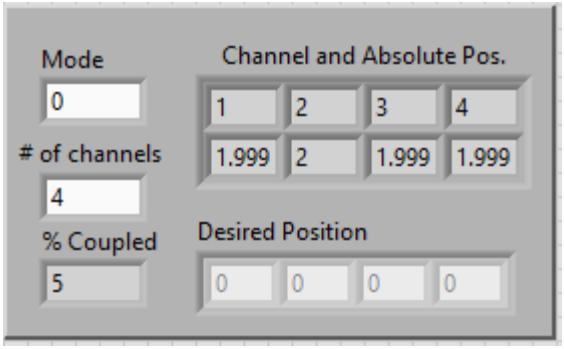
The "mount_controls.vi" subVI receives an array of positions and mode number as inputs. Outputs laser coupling percentage.



Full mount controls block diagram.

The front panel displays:

- Mode #, Python input
- Desired position array, Python input
- # of channels, user control input
- Saved position for each channel
    - Read from “saved_position.csv”
- Percent of laser coupled, LV output

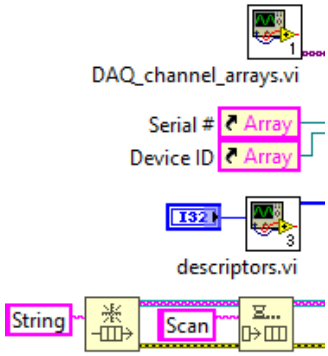


Mount controls front panel display and inputs.

Initialization

When the “controls_auto_TCP.vi” VI is run:

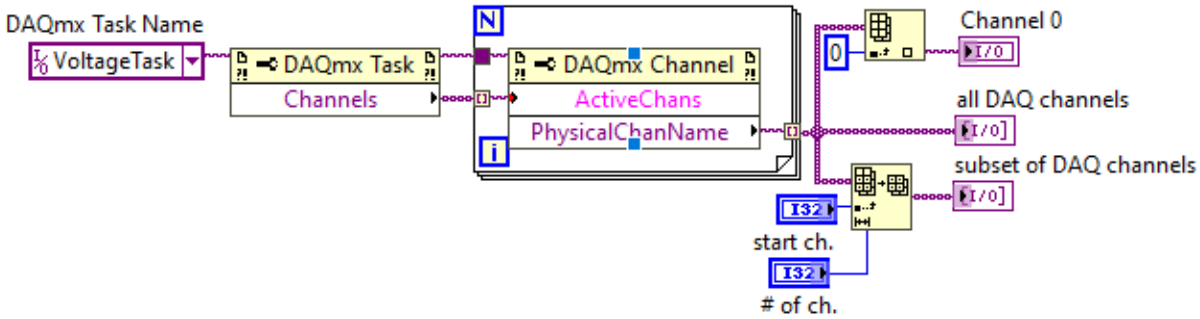
- Creates message queue and initializes with “Scan” to get array of serial #’s and device ID’s
- Runs “descriptors.vi” subVI to get array of all controller channels
- Runs “DAQ_channels.vi” subVI to get arrays of DAQ AI channels



Array and queue initialization.

DAQ Channels Configuration

The “DAQ_channels” subVI creates an arrays of DAQ analog input channels.

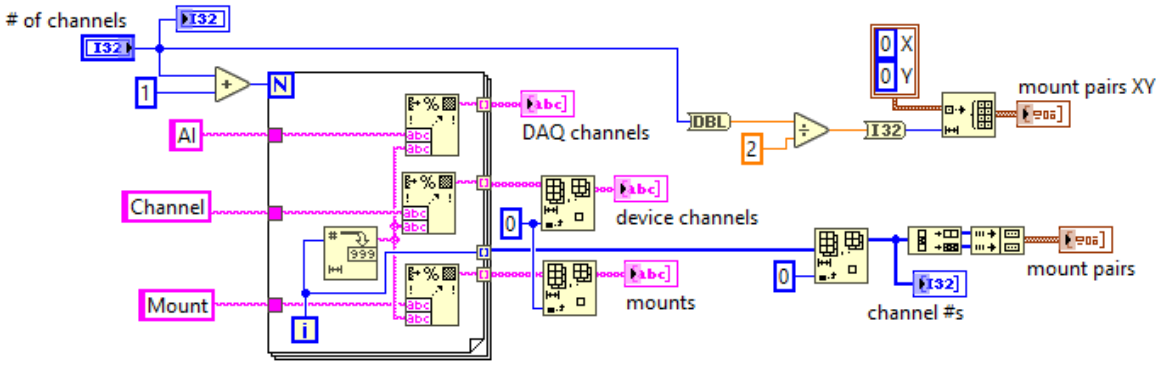


Full block diagram of DAQ channel subVI.

- Voltage sampling task created in NIMAX referenced for property node to output AI channels
- For loop creates array of the names of DAQ AI channels, N = # of active channels
- Subset of channel array created at user controlled start and length
- Channel 0 separated for reading the photodiode
    - 16 analog inputs on the DAQ (AI 0-15), 1-15 can be used to read sensors

Mount Channels Configuration

The “descriptors.vi” subVI is used to create arrays of channel numbers and labels for the front panel. User inputs “# of channels” on the “mount_controls.vi” front panel.



Descriptors full block diagram. Creates labels for device channels, DAQ channels, and mount clusters.

String Label Arrays

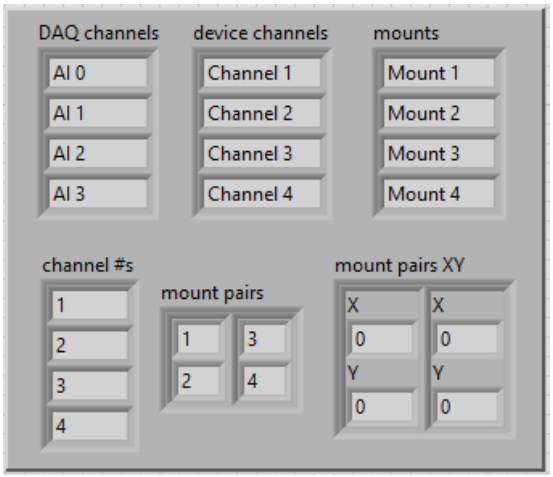
For loop creates array strings to make labels for DAQ channels, device channels, and mounts.

- N = # of channels + 1
- Formats labels and loop “i” into string
- First element (0) deleted from mount and device channel arrays

Channel Numbers and Mount Clusters

Amount of loop iterations is used to create a numeric array of channels and clusters for mount pairs.

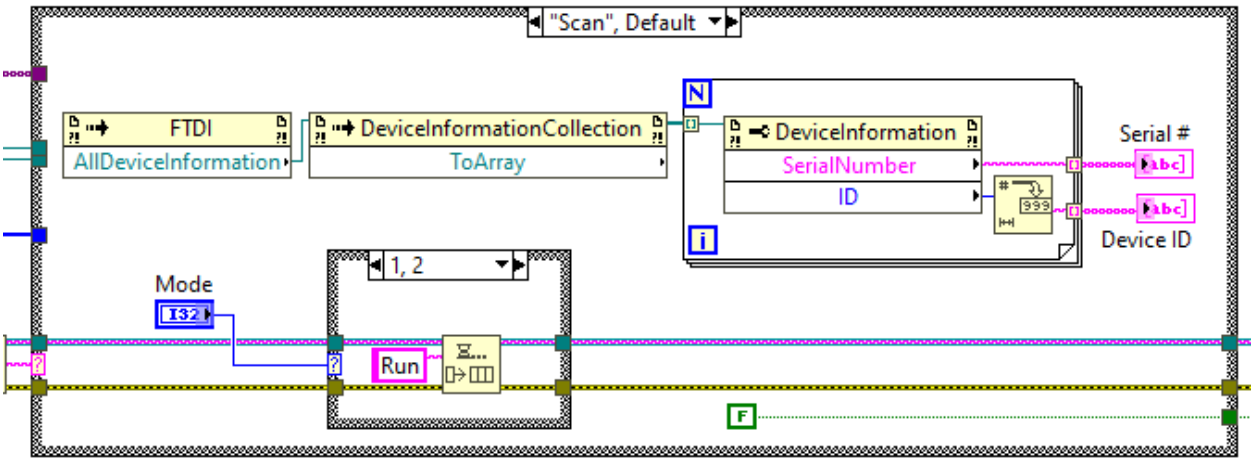
- First element (0) deleted from “i” array to get array of channel #’s
- Channel # array is decimated to split into odd and even channels
- Odd and even channels indexed and clustered to get mount channel # pairs
- Cluster of mount channel XY pair controls made size of (# of channels)/2



Descriptors array outputs. String labels (top row) and channel numbers and pair clusters (bottom row).

Scan Case

Case triggered by initial message in queue when the VI is run.

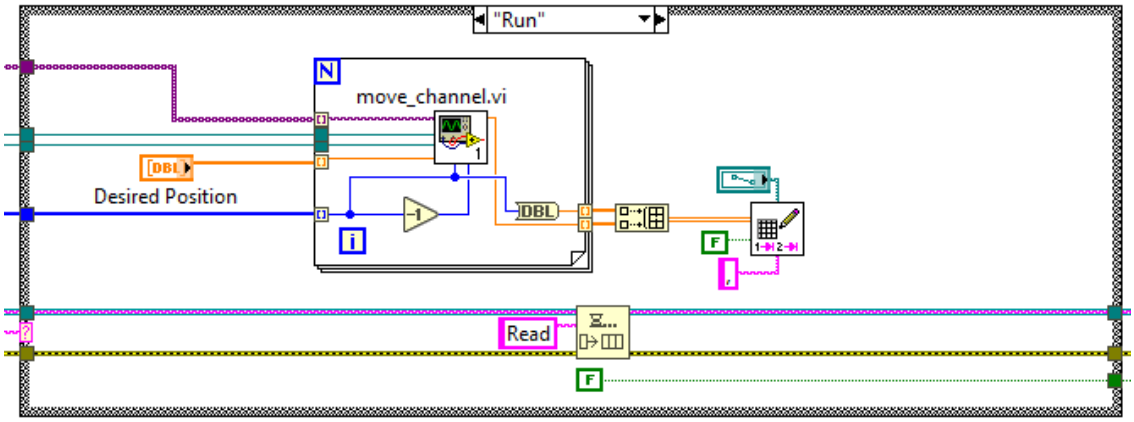


"Scan" initialization case.

- Collect all info from connected FTDI USB devices (K-cube controllers) and convert to an array
- For loop N = size of device info array; how many controllers are connected
    - Property node splits device info into arrays of motor serial numbers and controller IDs
    - String arrays are used as a local reference outside of the loop to connect to controllers
- Mode input from TCP connection VI queues message based on user input
    - 1 or 2 = "Run"
    - 3 = "Home"
    - 0/Default = "End"

Run Case

When a "Run" message is dequeued, channels are moved to the input position and "Read" is queued.

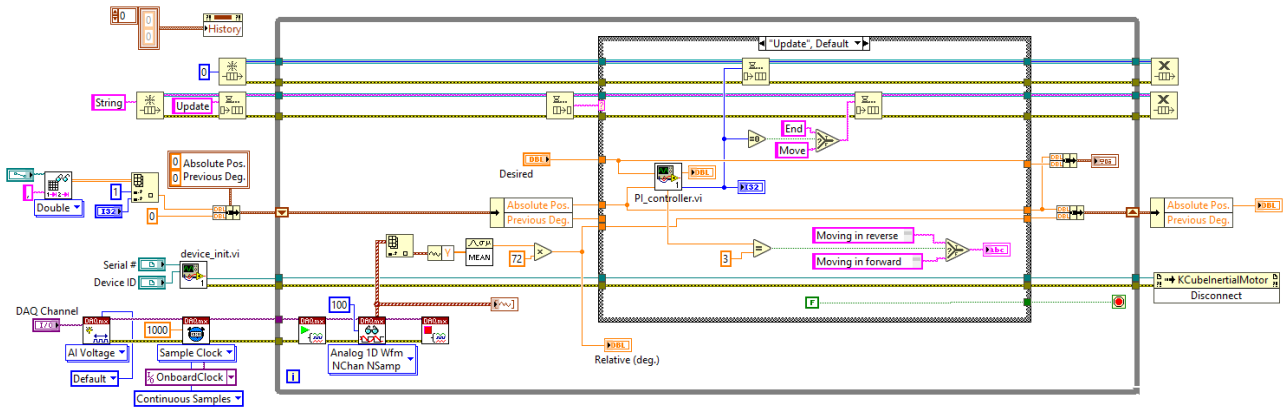


"Run" case.

- For loop takes inputs of serial #/device ID references and indexes through input arrays of the mount channel, DAQ channel, and desired position
- Runs "move_channel" for all channels and writes absolute position to file when for loop is exited

Move Channel

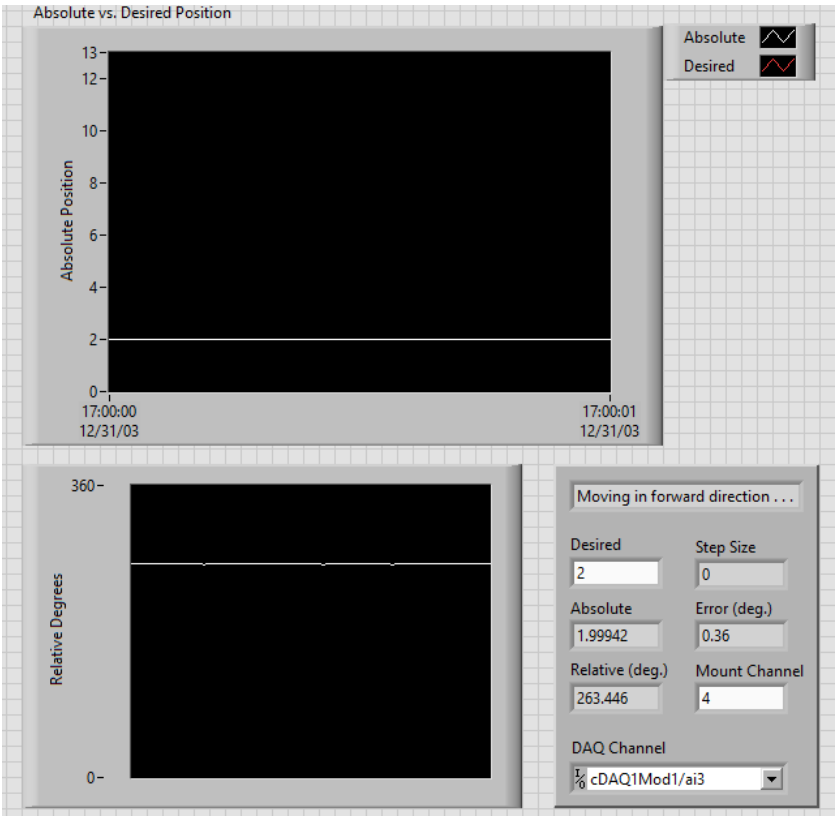
The “move_channel” subVI moves a channel to the desired position using a PI controller.



“Move_channel” subVI full block diagram.

Front Panel

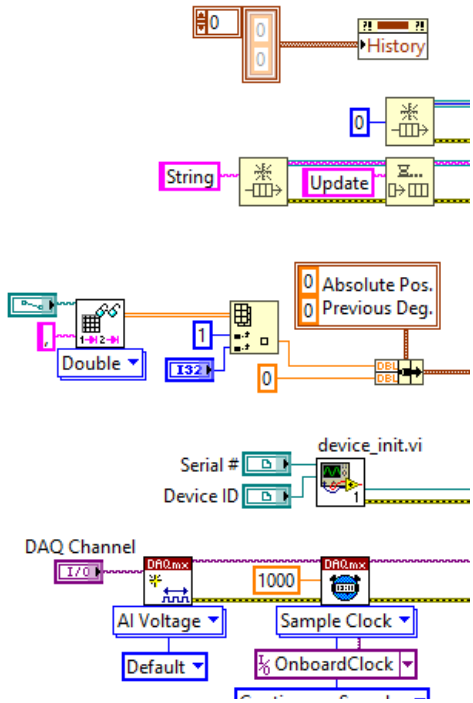
- Graph of absolute vs. desired position
- Graph of relative position
- Desired position input
- Current:
  - Step size
  - Absolute position
  - Relative position
  - Degree error
  - Mount channel
  - DAQ channel



Front panel display of “move_channel” subVI.

Initialization

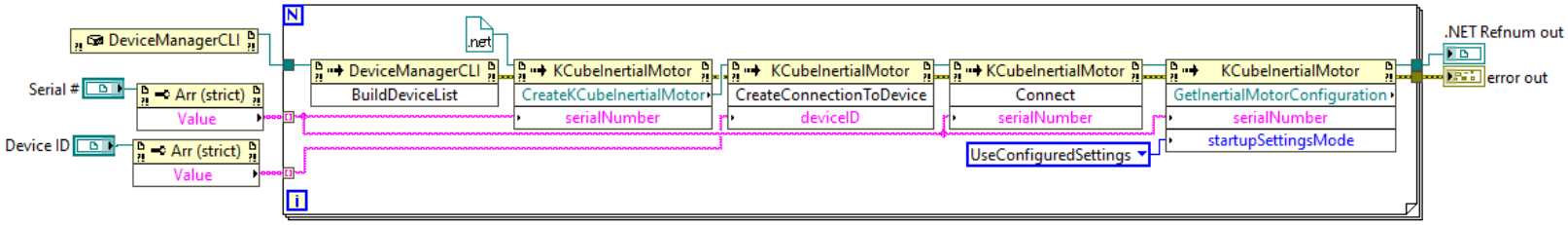
- Absolute vs. Desired Position graph reset
- Step size queue created
- Message queue created and initialized with “Update”
- Saved position file is read
- Position shift register initialized with current channel’s absolute position and 0 for previous degrees
- “device_init” subVI connects to device and controller
- DAQ AI voltage channel is created
- Continuous sampling timer is created with a 1000 Hz (samples per channel/second) sampling rate



Initialization before moving device channel.

Device Initialization

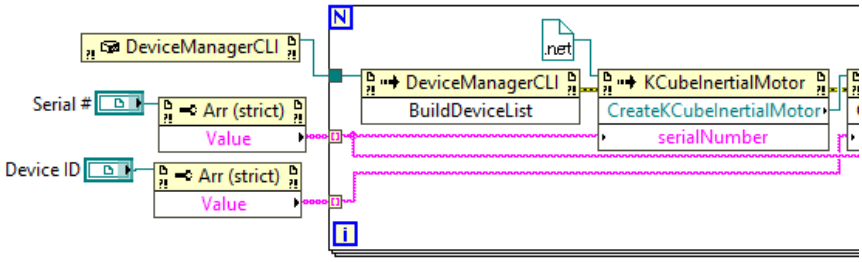
The “device_init” subVI creates the motor controller and device and connection to the mounts.



“device_init” full subVI.

To create K-cube motor controllers and motor devices:

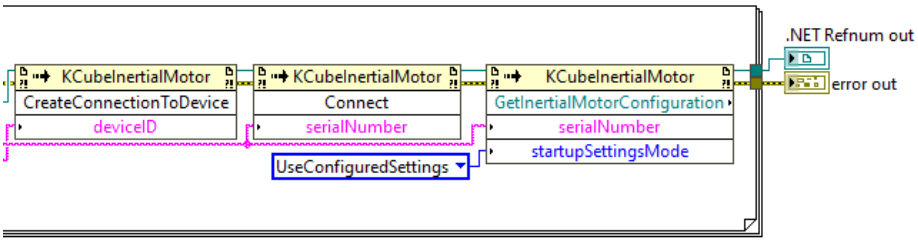
- Constructor node creates ref. of .NET device manager
- Property nodes reads serial # and device ID string arrays
- Device manager data and string arrays enter FOR loop
  - N = # of controllers
  - Device manager property builds list of devices from constructor node
  - Property node creates motor device using the serial #, refnum in = motor class constant



Create K-cube motor controllers and motor devices.

"CreateKCubeInertialMotor" used as reference to connect to:

- K-cube controller with device ID
- Inertial motor with serial #
- Get motor configuration to initialize default settings

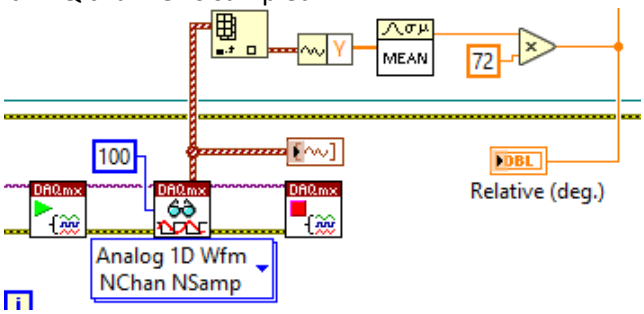


Connect to motor and initialize settings to communicate with the mirror mounts.

Sensor Sampling

Each time the loop runs, the sensor connected to the current DAQ channel is sampled.

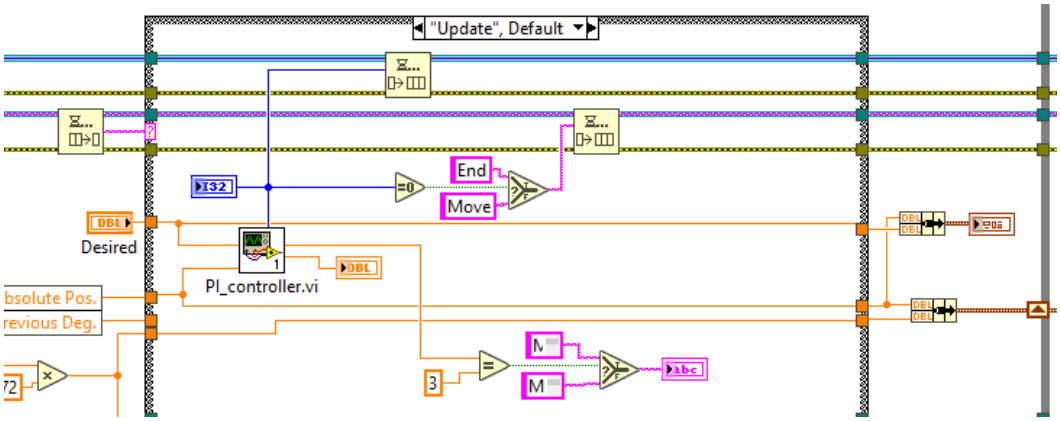
- Start DAQ task
- Read AI with 100 samp/ch
- Stop DAQ task
- Get voltage data from waveform output and average the samples
- Multiply by 72 (360/5) and output relative deg.
- Plots waveform graph of relative degrees



Sensor sampling and conversion to relative degrees.

Update Case

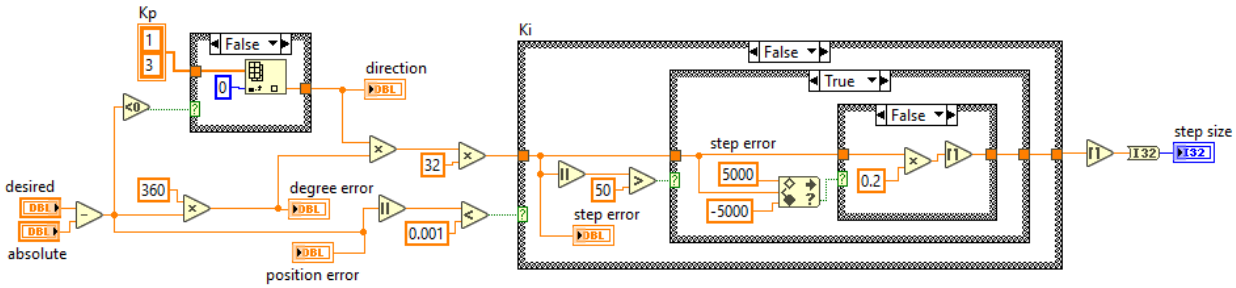
When an "Update" message is dequeued, the PI controller determines the step size needed. A "Move" or "End" messaged is queued based on whether the step size output = 0.



"Update" case while moving channels to get step size.

- Writes current relative position to "previous degrees" shift register
- Desired position and absolute position compared by the PI controller
  - Outputs and enqueues step size
- Checks if step size = 0 and if true, queue "End" message; if false, queue "Move"
- Checks direction and outputs indicator message on front panel

PI Controller

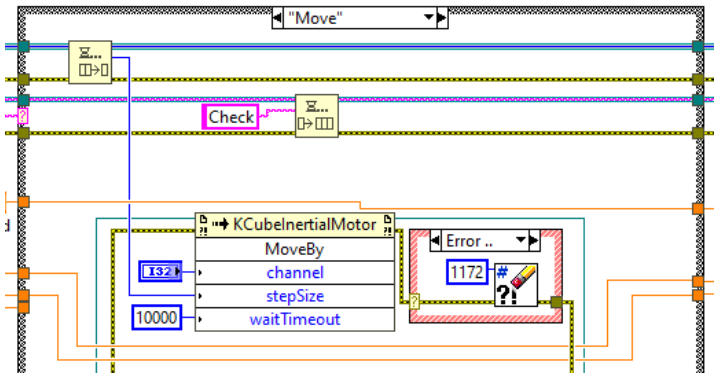


PI Controller subVI full block diagram.

- Position error = desired – absolute
- First case structure “Kp” checks if position error < 0 to get direction
    - True = reverse = 3, false = forward = 1
- Position error multiplied by 360 to get degree error
- Degree error x direction x 32 = approximate step error
    - Forward approximately 32 steps/degree forward and 3 times faster than reverse
- Second case structure “Ki” checks if absolute position error < 0.001
    - If true, step size = 0
    - If false, checks if step error is > 50
        - If false, step size = step error / 2
        - If true, check if within +/- 5000 step range
            - If within range, step size = step error
            - If out of range, step size = step error / 5

Move Case

When a “Move” message is dequeued, the channel is moved by n steps and queues “Check” message.

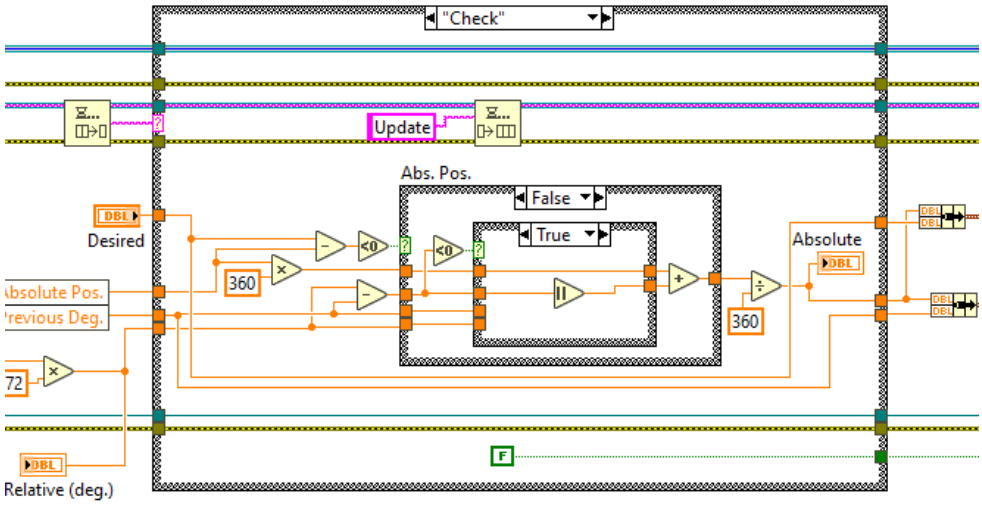


Move channel by PI controller designated step size.

- Step size dequeued and current channel moved by input step size
- If error = 1172, ignore
    - 1172 = timeout; has not received a second message
- Enqueue “Check” message to get absolute position

Check Case

When a “Check” message is dequeued, absolute position stored in the shift register and relative position before and after “Move” case is used to update absolute position. “Update” message in enqueued.



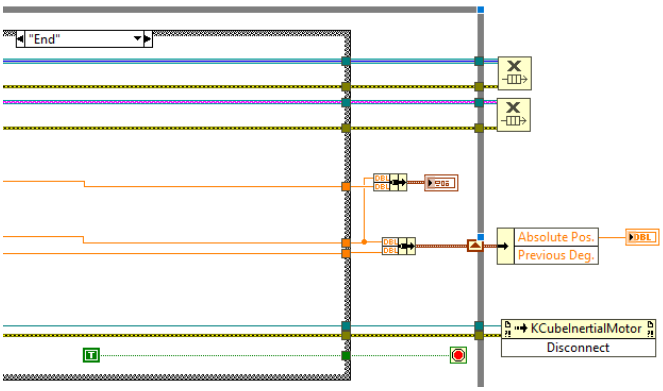
“Check” case to calculate change in absolute position and update shift register.

- Absolute position x 360 to get absolute degrees
- Change in degrees after moving = relative deg. – previous deg.
- Check if desired – absolute position < 0 to get direction
  - T: Reverse = CW = + V/degrees = - steps (towards mirror)
  - F: Forward = CCW = - V/degrees = + steps (away from mirror)
- If direction = reverse, check if change in deg. > 0:
  - T: absolute pos. = absolute deg. – change in deg.
  - F: absolute pos. = absolute deg. – (relative deg. + (360 – previous deg.))
- If direction = forward, check if change in deg. < 0:
  - T: absolute pos. = absolute deg. + |change in deg.|
  - F: absolute pos. = absolute deg. + (previous deg. + (360 – relative deg.))

End Case and Termination

When an “End” message is dequeued, the loop is stopped and references are closed.

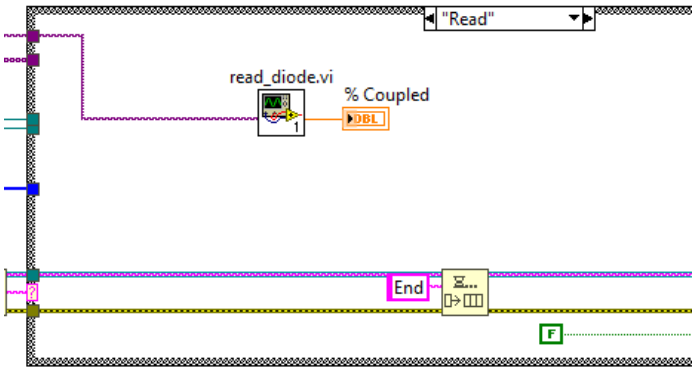
- Loop stop condition = T, loop is exited
- Message and step queues are released and motor is disconnected
- Outputs absolute position



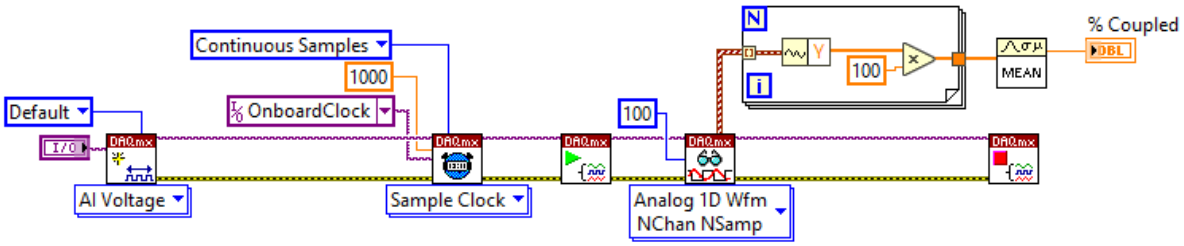
Moving VI “End” case and subVI termination.

Read Case

When a “Read” message is dequeued, the photodiode is sampled and laser coupling % is output.



“Read” case in mount control VI.



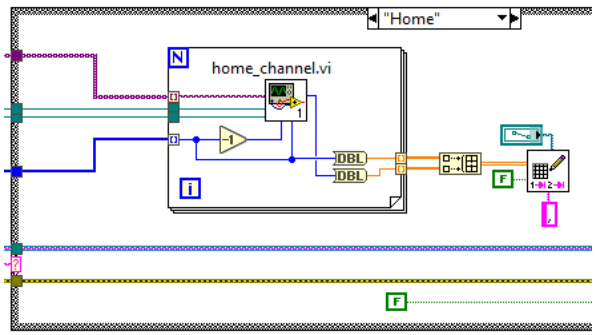
Full block diagram for “read_diode” subVI.

- Channel 0 of the DAQ is created and read continuously
- 100 samples from the power meter are multipled by 100 and averaged
    - Power meter response (photodiode stand-in) = 1 V/mV
- % coupled output

Home Case

When a “Home” message is dequeued, the channels are homed and their saved position is set to 0.

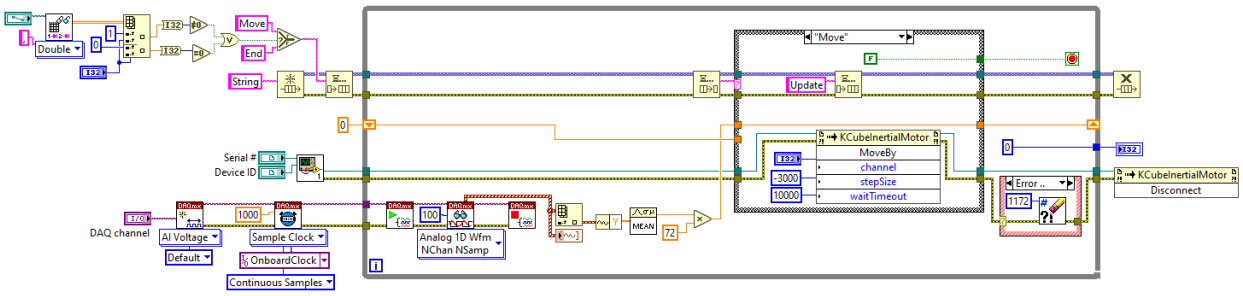
- For loop takes inputs of serial #/device ID references
- Indexes through input arrays of the mount channel and DAQ channels
- Runs “home_channel” for all channels
- Writes 0 to position file when for loop is exited



Mount Control VI “Home” case.

Home Channel

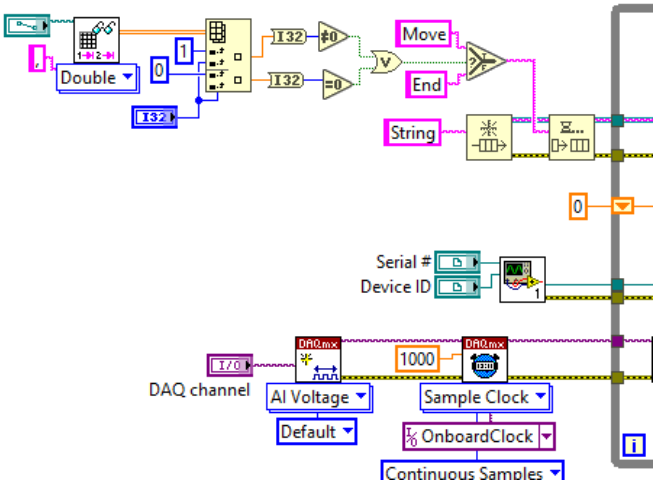
Moves channel by set step size in reverse direction until the sensor is no longer moving to find home.



"home_channel" subVI full block diagram.

Initialization

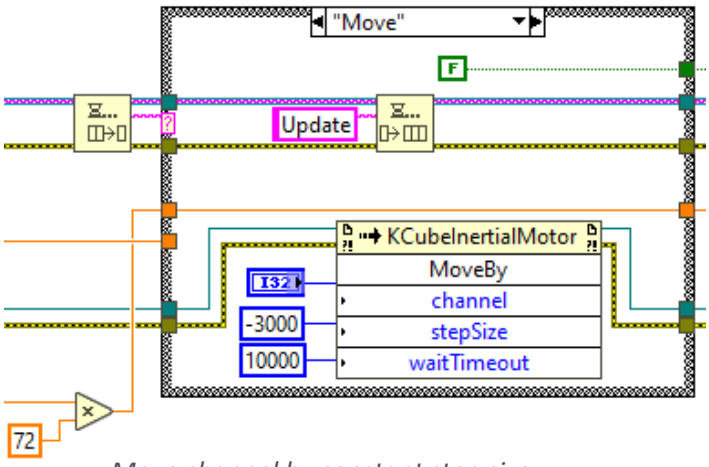
- Read saved position csv file
- If channel = 0 or position ≠ 0, initialize message queue with "Move"
    - If position = 0, enqueue "End"; channel already homed
- Initialize relative position shift register with 0
- Connect to controller and device; same as in "move_channel"
- Create DAQ channel and timer; same as in "move_channel"



Homing subVI initialization procedures.

Move

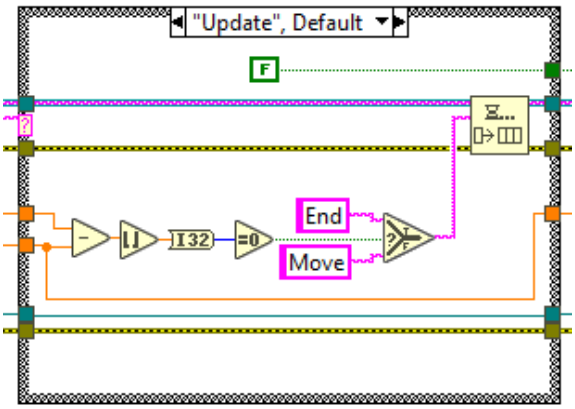
- Move current channel by -3000 steps
- Queue "update" message
- Send relative degrees to shift register



Move channel by constant step size.

Update

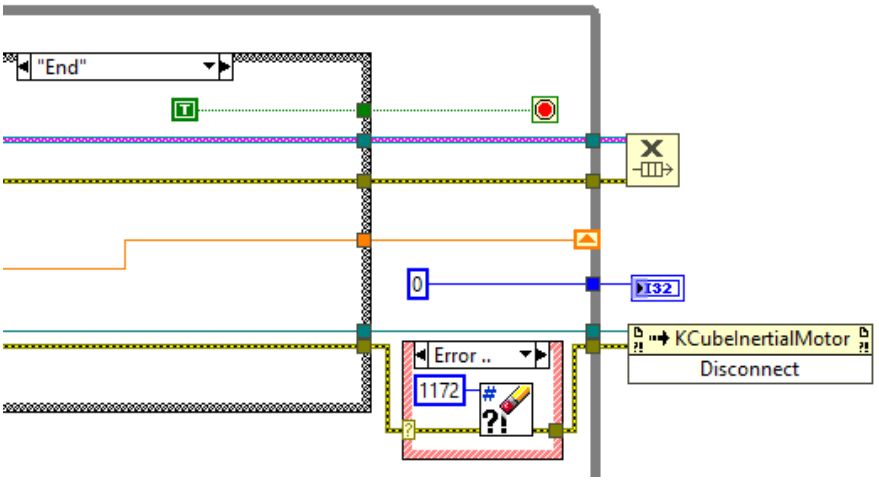
- Check if current relative deg. – previous relative deg. = 0 ; rounded towards – infinity
    - T: Sensor is not moving, enqueue “End”
    - F: Sensor can still move, enqueue “Move”



Check if sensor is still moving.

End and Termination

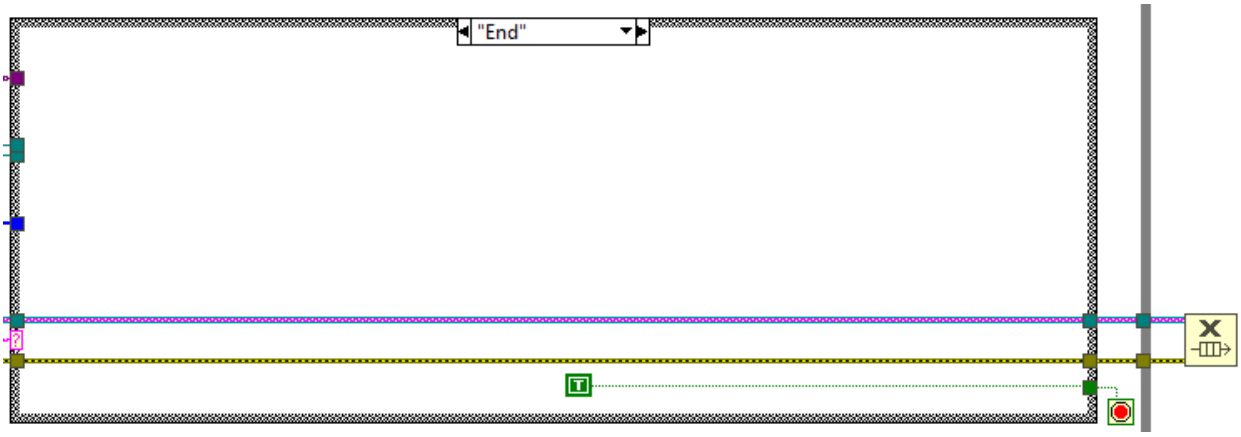
- Loop stop condition = T, exit loop
- If error = 1172 (timeout), ignore
- Output absolute position = 0
- Release message queue and disconnect from motor device



Homing VI “End” case and subVI termination.

End Case

When an “End” message is dequeued, the loop stop condition is set to “True”. The loop is exited, the message queue is released, and the “mount_controls” subVI is stopped.



Mount Controls VI “End” case and subVI termination.