Hands-on Lab

LabVIEW – NI-DAQ Counters and Timers

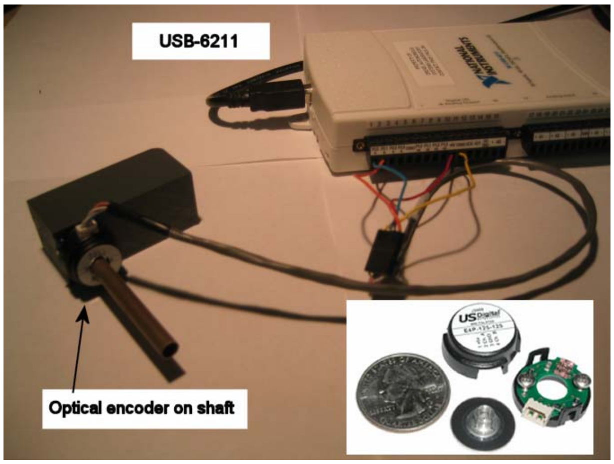


Photo: US Digital's E4P encoder with USB-6211

The USB-6211 provides two 32-bit 80 MHz counters/timers. First, as a counter, one can acquire signals. Sensors like encoders on wheels, Hall Effect devices on metal gears and mechanical switches on turnstiles generate pulse trains. Counters connected to such sensors enumerate these pulses to measure phenomena like wheel rotation, engine speed and people attendance. Second, as a timer, one can generate custom pulse trains. Some devices like motors and linear actuators require pulse-width modulated signals with specifically timed signals.

In this lab a US Digital E4P incremental optical encoder is interfaced to the USB-6211 (see Photo). The lab will configure a counter to read encoder pulses to measure shaft rotation. Additionally, this lab will configure a timer to generate a signal that drives a hobby servo motor

Concept 1: Use a counter VI to monitor shaft angle.

Step 1: Create the front panel and block diagram

Drag a numeric indicator and a `Stop` button into an empty VI front panel (see **Figure 1-1A**). The block diagram, shown in **Figure 1-1B**, uses a `While` loop.

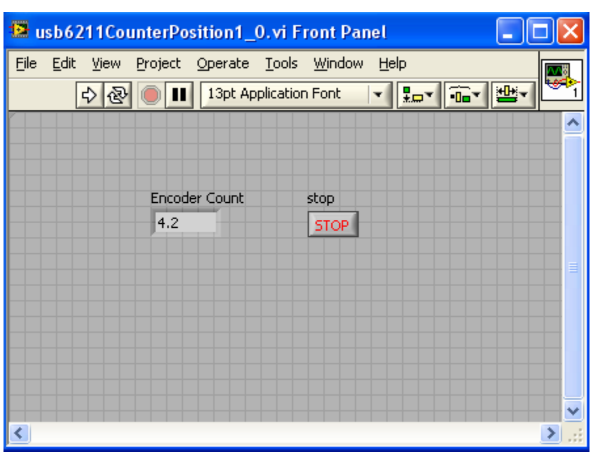


Figure 1-1A: Encoder Display Front Panel

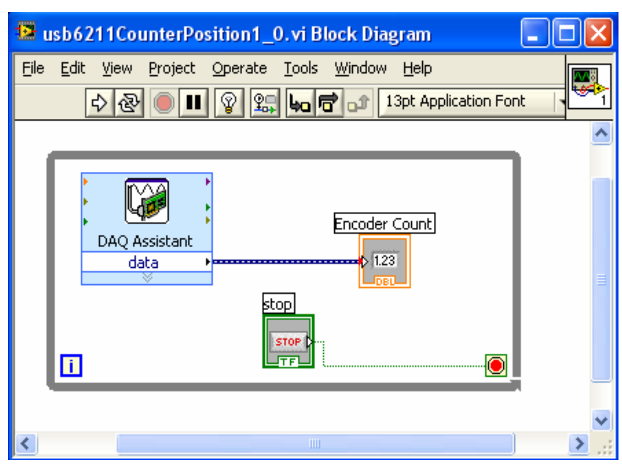


Figure 1-1B: Corresponding Block Diagram

Step 2: Configure DAQ Assistant for one Counter

Launch `DAQ Assist`. Choose `Acquire Signal, Counter Input` and `Edge Count` as shown in **Figures 1-2A** and **1-2B**. When prompted, select `ctr0` (counter 0).

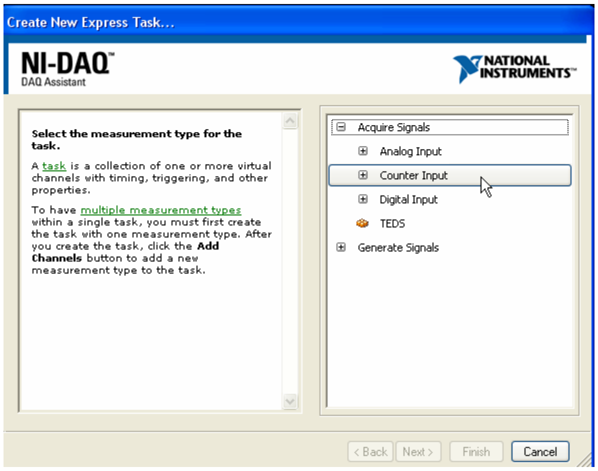


Figure 1-2A: Select Analog Output - Voltage

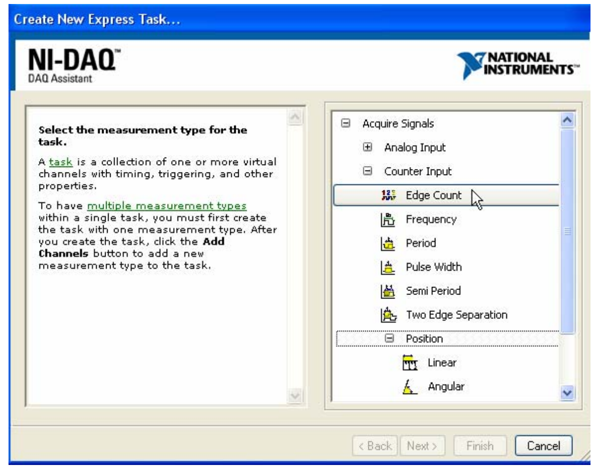


Figure 1-2B: Select analog output channel 0

Configure the next pop up box to look like **Figure 1-2C**. Set the `Acquisition Mode` to "`1 Sample (On Demand)`". Save file as **`usb6211CounterPosition1_0.vi`**

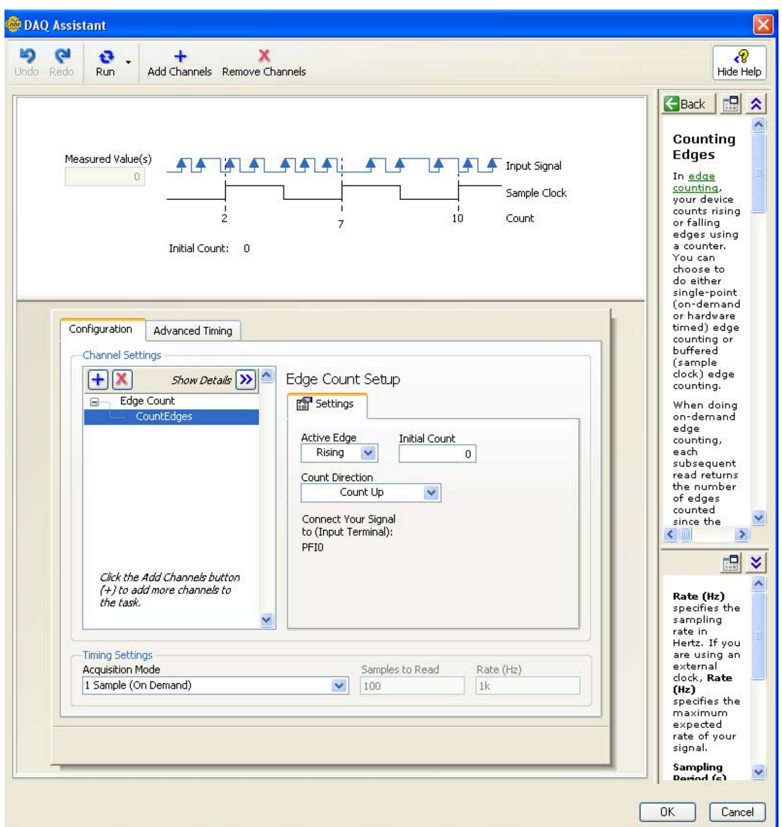


Figure 1-2C: `Generation Mode` set for `1 Sample (On Demand)`

Step 3: Wire the encoder (US Digital model #E4P) to the USB-6211

Figure 1-3A shows the E4P wired to the USB-6211 with connections given in **Figure 1-3B**. From the front panel, click the Play button (white arrow). Rotate the shaft clockwise and counter-clockwise. Observe what happens.

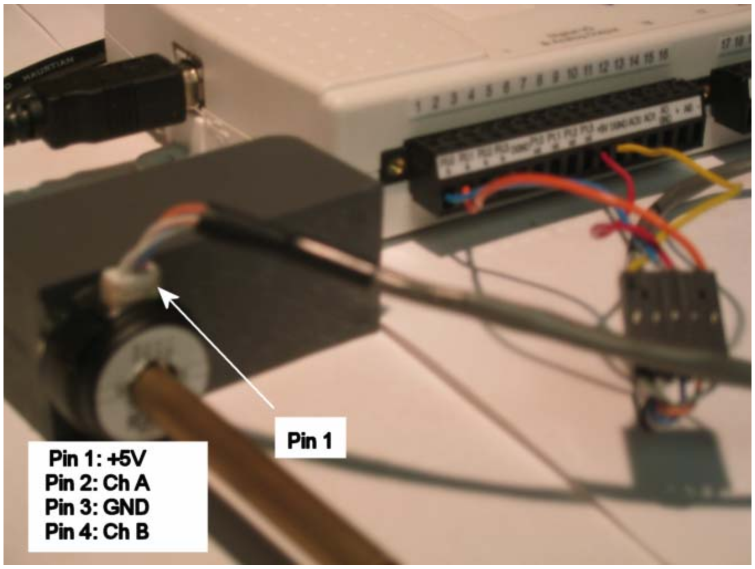


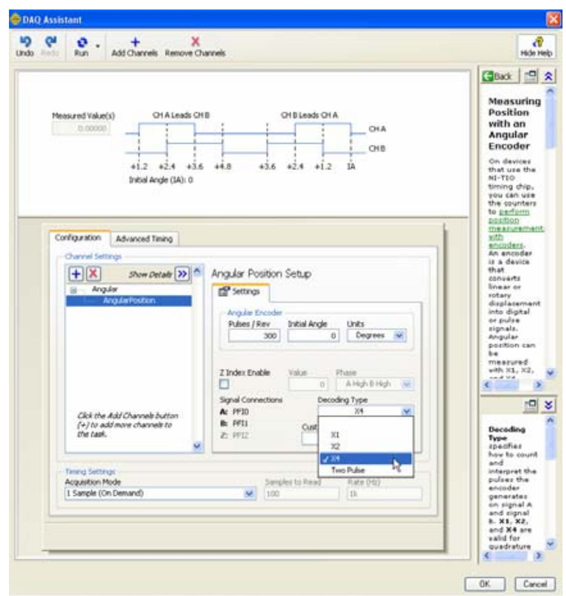
Figure 1-3A: The E4P cable brings signals to the USB-6211

E4P	USB-6211
Pin 1 (+5V)	Pin 10 (+5V)
Pin 2 (Ch A)	Pin 1 (P0.0)
Pin 3 (GND)	Pin 11 (GND)
Pin 4 (Ch B)	Pin 2 (P0.1)

Figure 1-3B: Connections

Exercise 1: In LabVIEW create programs for the following:

1.1. Write a VI that reports the shaft angle in degrees (CW and CCW rotations respectively increase and decrease angle reading). Hint: From DAQ Assistant chose `Acquire Signals – Counter Input – Position – Angular`. Use the X4 (times 4) `Decoding Type` with 300 `pulses/Rev`. Save program as **`usb6211CounterPosition1_0.vi`**.



Concept 2: Employ Shift Registers to report Angular Velocity

Step 1: Create a front panel and block diagram

Figure 2-1A shows that two `Numeric Indicators` are used to display the angle in degrees and angular velocity in degrees per second. The block diagram in **Figure 2-1B** is wired up accordingly.

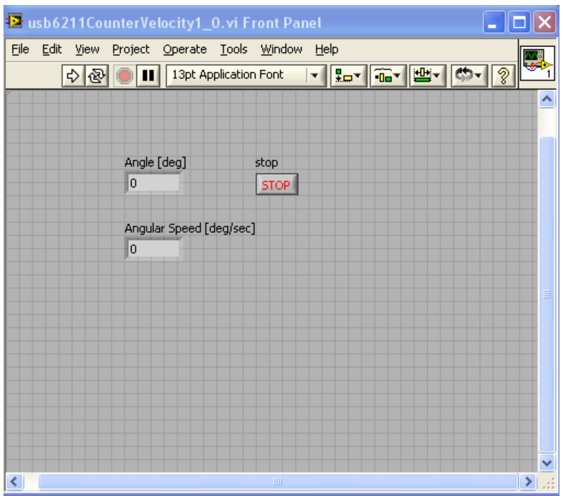


Figure 2-1A: Two numeric indicators

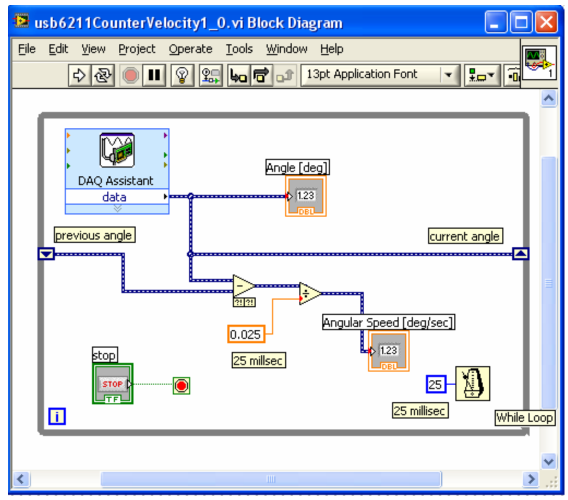


Figure 2-1B: Block diagram displays voltage

To create a shift register, click the `while-loop`. When highlighted, right click. From the menu, choose `shift register`. The `DAQ Assistant` provides the current angle. The `shift register` keeps the previous angle value. Their difference is divided by a sample time. The sample time is chosen by presenting a "`Wait Until Next Millisec Interval`" element that is set for 25 milliseconds.

Step 2: Set up DAQ Assistant

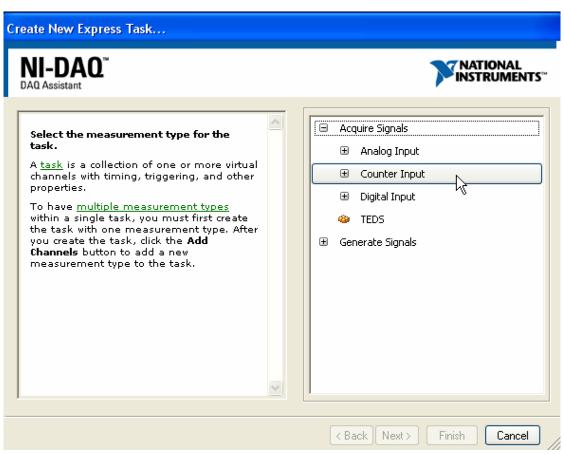


Figure 2-2A: Select `Counter Input`

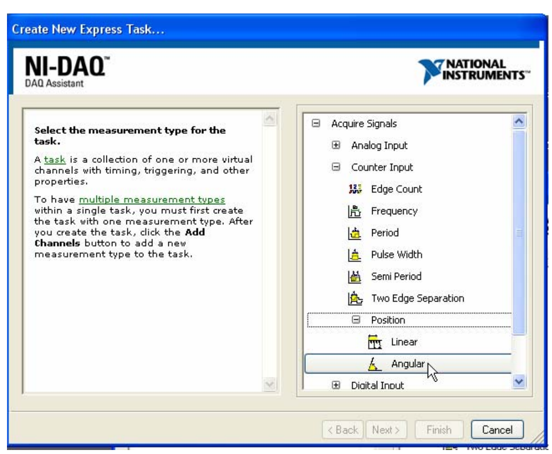


Figure 2-2B: Select `Position - Angular`

Make sure the `Acquisition Mode` is set to "`1 Sample (On Demand)`" as shown in **Figure 2-2C**. Also set the Pulses/Rev to 300 (the E4P encoder has 300 counter/revolution). For Decoding type, use X4 quadrature (this increases the encoder to 1200 counts/revolution through software). Save file as **`usb6211CounterVelocity1_0.vi.`** Execute the program by clicking the Play button on the front panel. Verify that velocity reports zero when the shaft does not rotate. Observe the sign of the angle and velocity as the shaft rotates clockwise and counter-clockwise.

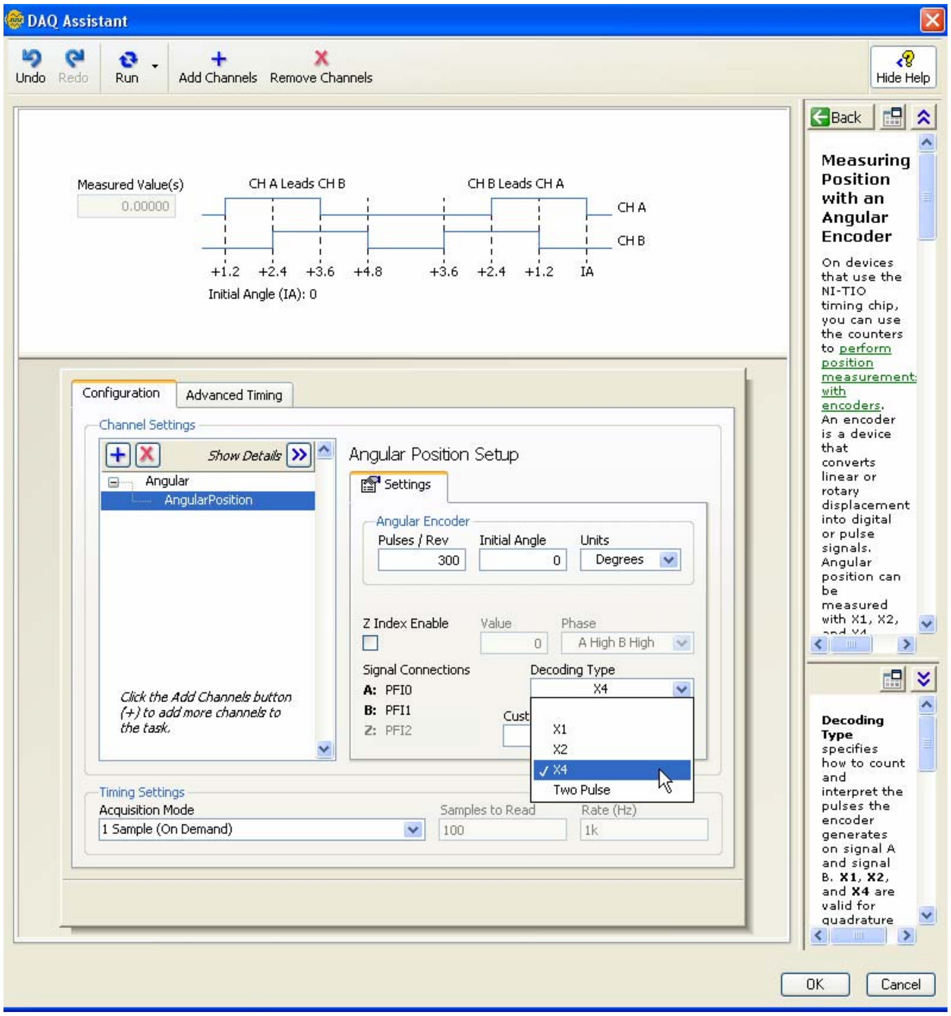


Figure 2-2C: Select "`1 Sample (On Demand)`"

Exercise 2:

2-1. Mount the cart and tether to pivot. Generate a constant voltage to the cart's motor. Use your watch to time how long it takes for the cart to rotate once. Compare this to encoder reading on velocity.

Concept 3: Generate a pulse-width modulated (PWM) signal with a timer and drive a servo

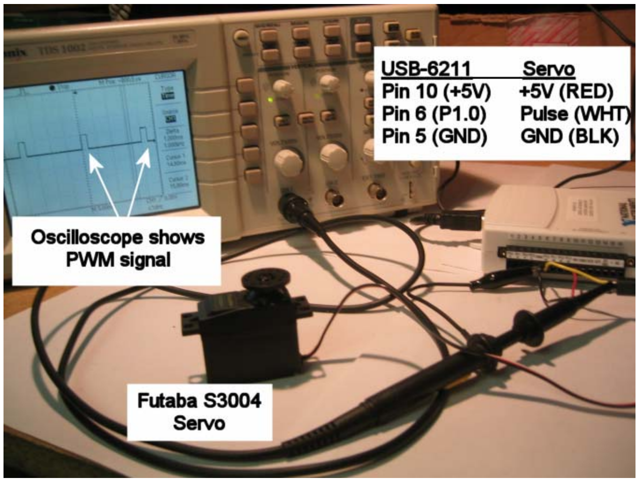


Photo: A timer is configured for PWM to move Servo

Hobby servos, like the Futaba S3004, are commonly found in radio-controlled vehicles. Such servos are gear-reduced DC motors. Shaft angle rotation is proportion to an input voltage's pulse-width. A pulse-width signal is defined by the ratio the signal is in a HI state versus being in a LO state. Called duty cycle, this ratio produces a pulse train. The photo shows this pulse width on the oscilloscope.

A typical servo has a pulse width ranging from 1 to 2 milliseconds and has a 50 Hz (20 milliseconds) frequency. At 1.5 milliseconds the shaft goes to its neutral (home) position. The USB-6211 can be programmed to create a suitable pulse train to command servo position.

Step 1: Create a front panel

Drag a `STOP` button and a `knob` into the front panel as shown in **Figure 3-1A.** Right click on the `knob` and configure the knob such that its `Data Range Minimum` and `Maximum` are `0.0010` and `0.0020` respectively. The knob's `Default Value` should be `0.0015` (i.e. the servo's neutral position). Also, for `Scale`, set the `Minimum` to `0.001` and `Maximum` to `0.002`.

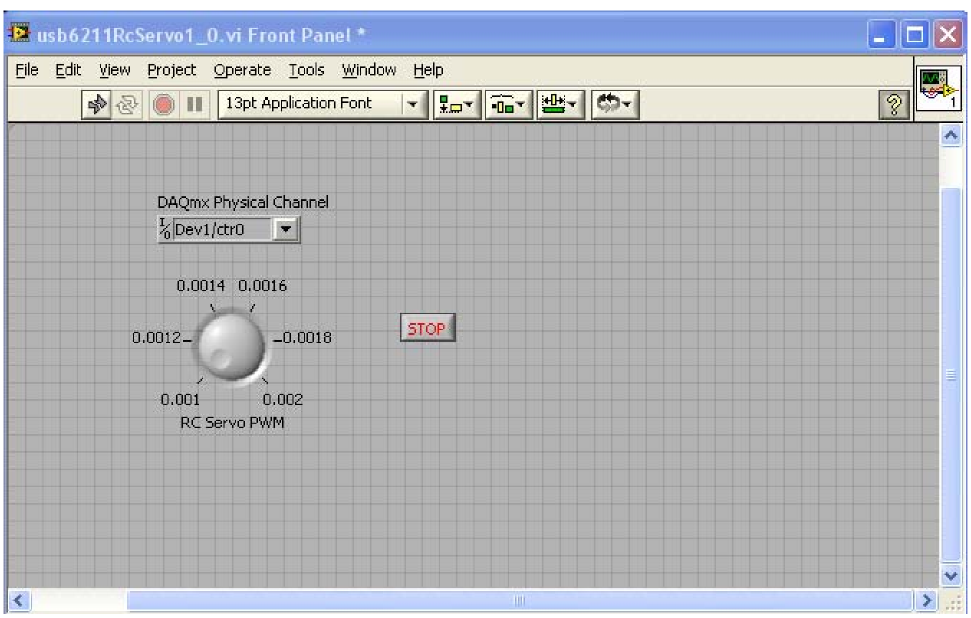


Figure 3-1A: Front panel shows knob and DAQmx Physical Channel

Refer to **Figure 3-1B** to place the `DAQmx Physical Channel` control in the front panel. Click `Classic – Classic I/O – Classic DAQmx Name Controls – DAQmx Physical Channel`. This is necessary because DAQ Assistant does not have a simple PWM option. As a result, one must create their own channels and customize the timer and signals.

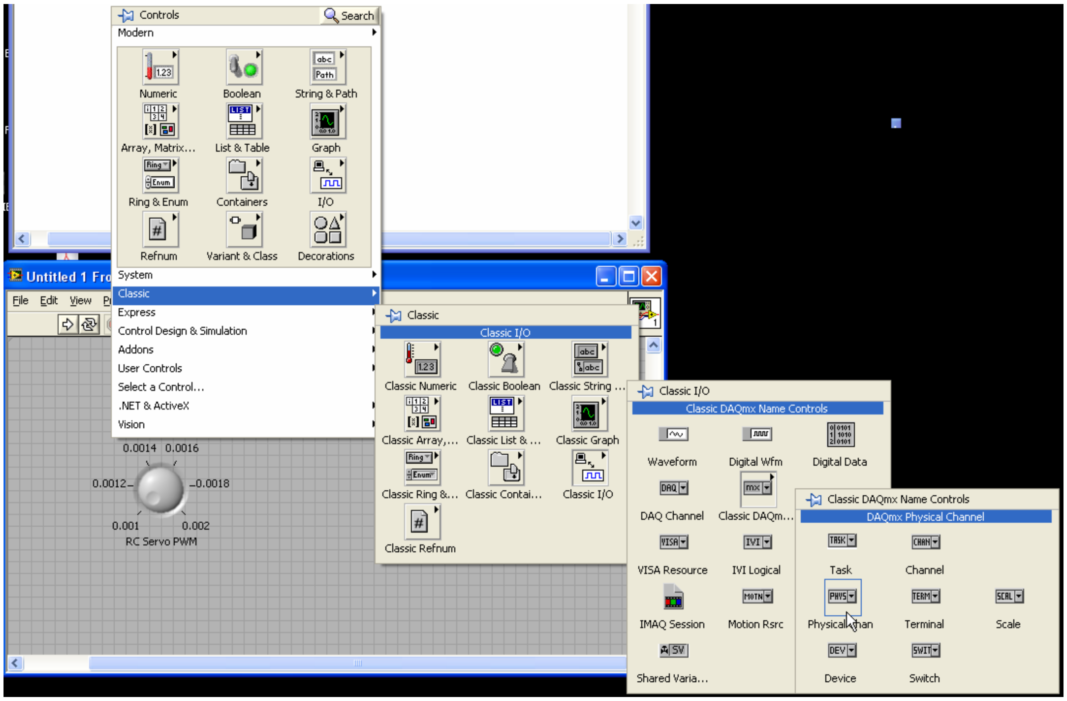


Figure 3-1B: Drag a DAQmx Physical Channel to the front panel

Right click on the `DAQmx Physical Channel – I/O Name Filtering – Counter Output` (see **Figures 3-1C** and **3-1D**. Also from the `Limit to Device` pull-down menu, select `Dev1`.

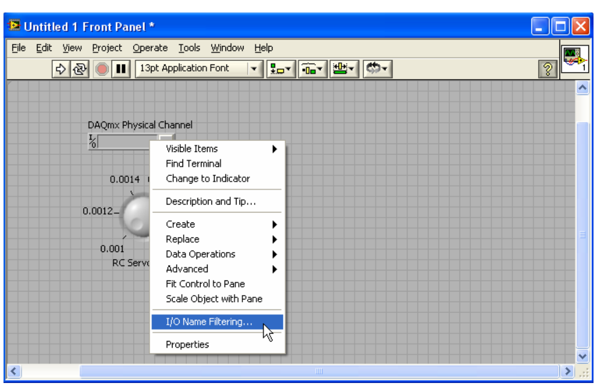


Figure 3-1C: Select `I/O Name Filtering`

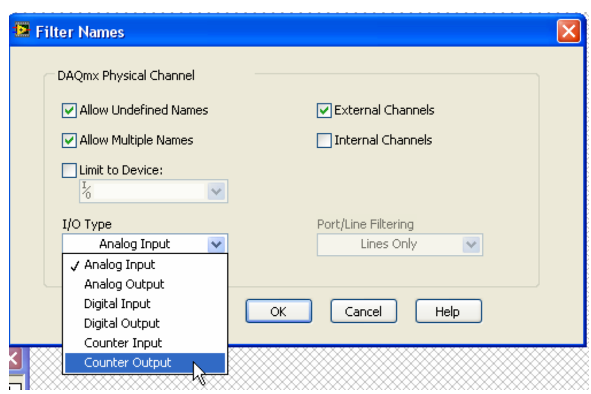
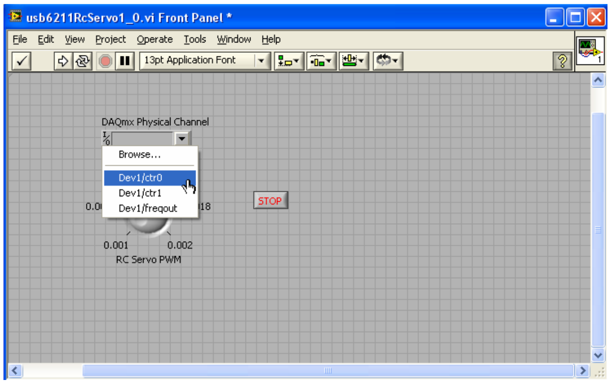


Figure 3-1D: Select `Counter Output`

Right clicking on the `DAQmx Physical Channel` allows one to set the default: `Dev1/ctr0` (see **Figure 3-1E**). The final front panel should look like **Figure 3-1F**.



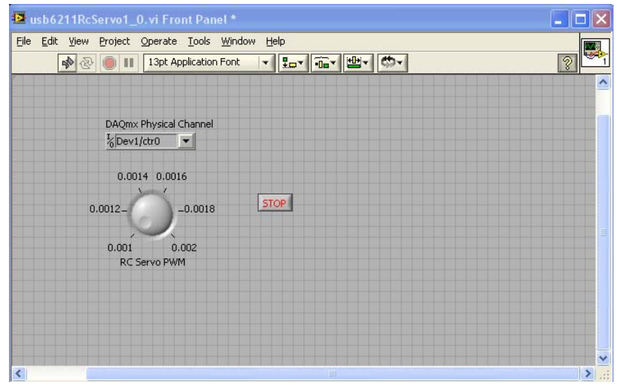


Figure 3-1E: Set the channel to `Dev1/ctr0`

Figure 3-1F: Finished front panel

Step 2: Block Diagram – Channel setup

As previously mentioned, LabVIEW does not have a PWM element that can be simply dropped into the block diagram. The building blocks to create one's own pulse train do exist. These will be accomplished and customized to generate the desired PWM signal.

First, one must have an element that initiates hardware as a timer. Referring to **Figure 3-2A**, right click and choose `Measurement I/O – NI-DAQmx – Start`. Drag this `DAQmx Start` element to the block diagram. Use this same process and add `DAQmx Timing` and `DAQmx Create Channel` elements to the block diagram (refer to **Figure 3-2B**).

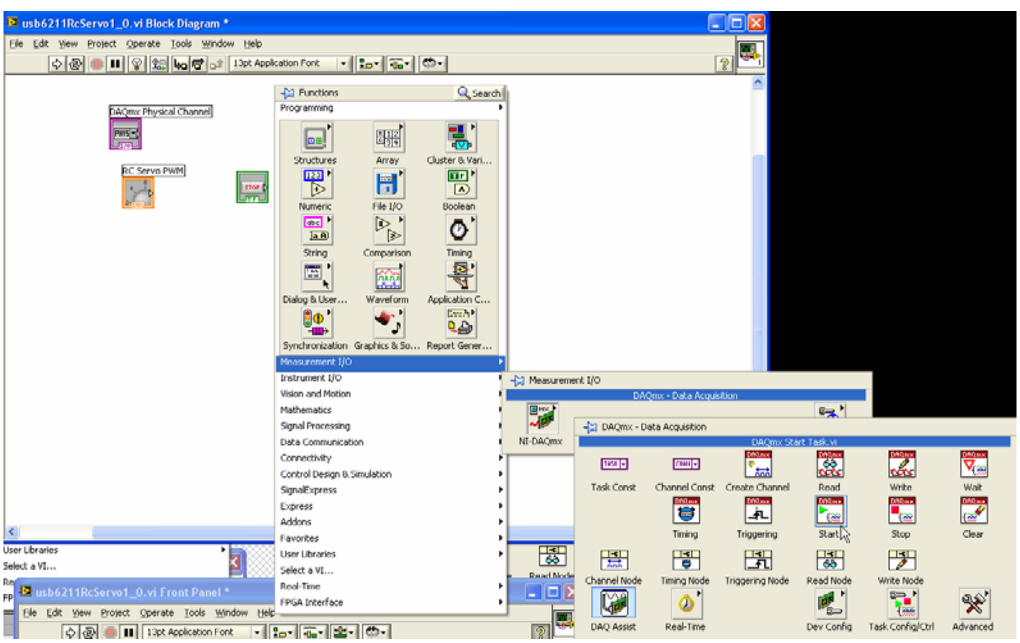


Figure 3-2A: Initiating hardware requires one have a `DAQmx Start` element.

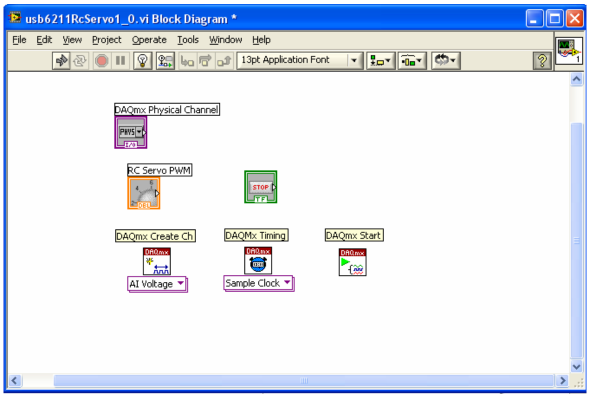


Figure 3-2B: Three DAQmx elements

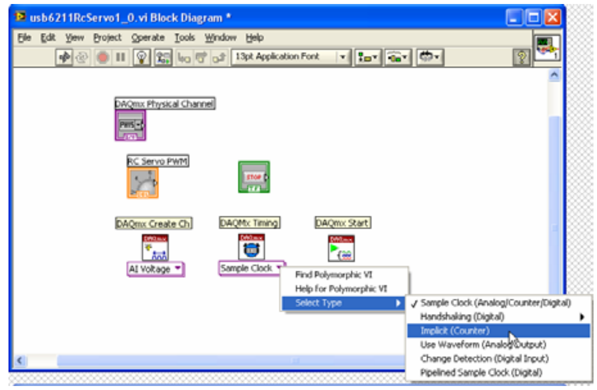


Figure 3-2C: Configure `DAQmx Start` element

To set the timer, refer to **Figure 3-2C** and right click on the `DAQmx Timing` element and choose `Select Type – Implicit (Counter)`. Next, to set the channel, refer to **Figure 3-2D** and right click the `DAQmx Create Channel` element and choose `Select Type – Counter Output – Pulse Generation – Time`. The `DAQmx Create Channel` is now configured as a `CO Pulse Time` element.

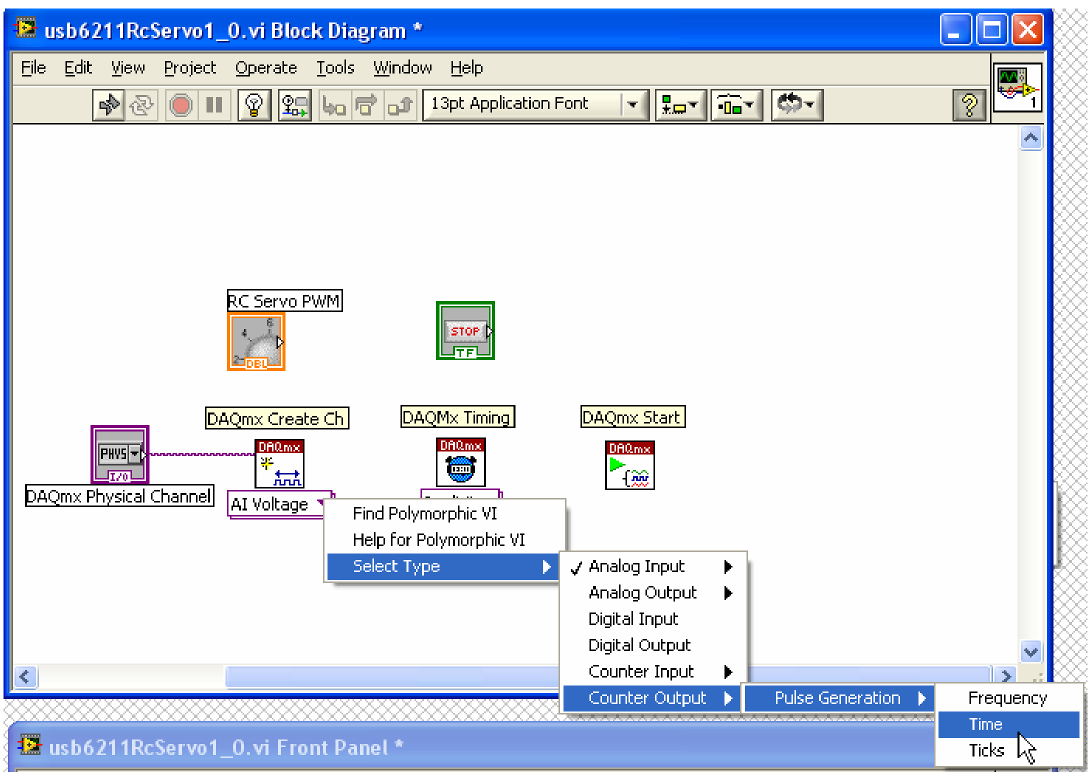


Figure 3-2D: Set the channel to output a pulse train based on time.

Bringing up context help (Choose from the block diagrams top menu: `Help – Show Context Help`) brings up a pop up box that shows pin locations for element wiring (see **Figure 3-2E**). Wire up the `DAQmx Physical Channel` (which is the knob control on the front panel) to the `CO Pulse Time element`.

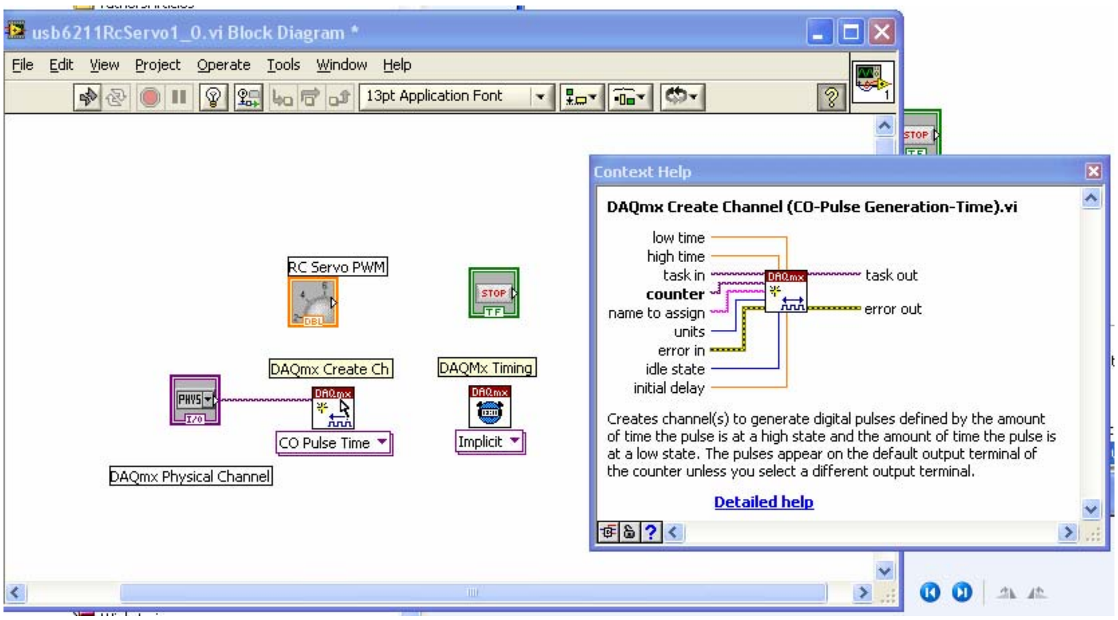


Figure 3-2E: Context help shows where on the element to wire the counter

Finish this step by wiring connections between each DAQmx element. Each element has a `Task Out` pin which wires into an adjacent element's `Task/Channels In` pin. See **Figure 3-2F** to wire connections between the `DAQmx Create Channel, DAQmx Timing` and `DAQmx Start` elements. Lastly, connect the knob element to the `CO Pulse Time "high time"` pin.

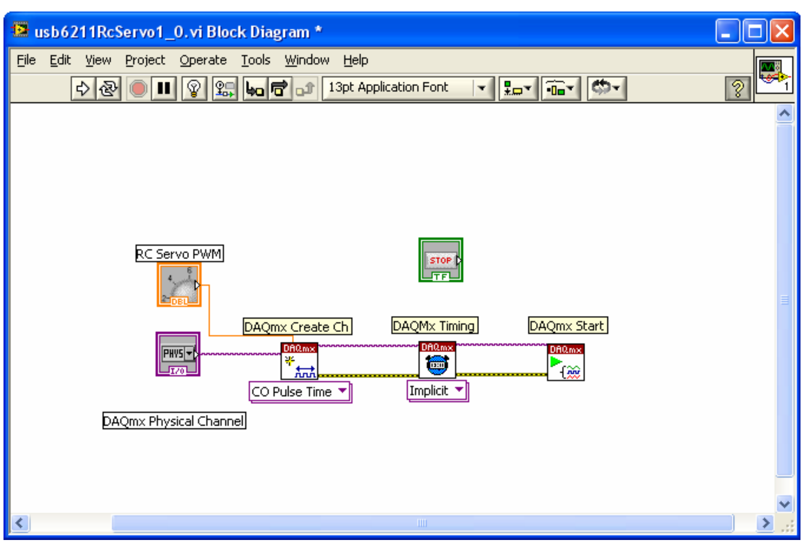


Figure 3-2E: Wiring between the 3 DAQmx elements

Step 3: Block Diagram – Generate Pulse Train

Build the block diagram to look like **Figure 3-3A**. First begin by dragging a `While Loop` structure and adding a `Wait Until Next ms Multiple` timer element. Set this timer to iterate every 20 milliseconds. Additionally, drag the `Stop` button and an `OR` logical element and wire them to terminate the `While Loop`.

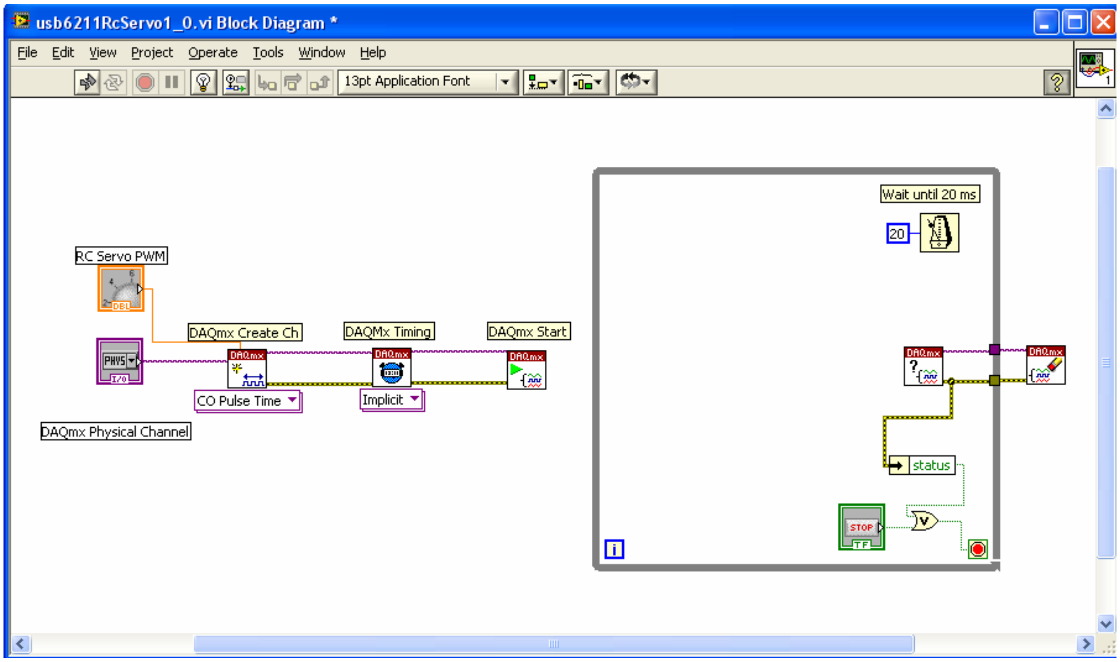


Figure 3-3A: Add the While Loop and its elements to complete the block diagram

Next, add the `DAQmx Task Is Done` element into the `While Loop`. Do this by clicking `Measurement I/O – NI-DAQmx – Task Config/Ctrl – Is Task Done` (**Figure 3-3B**).

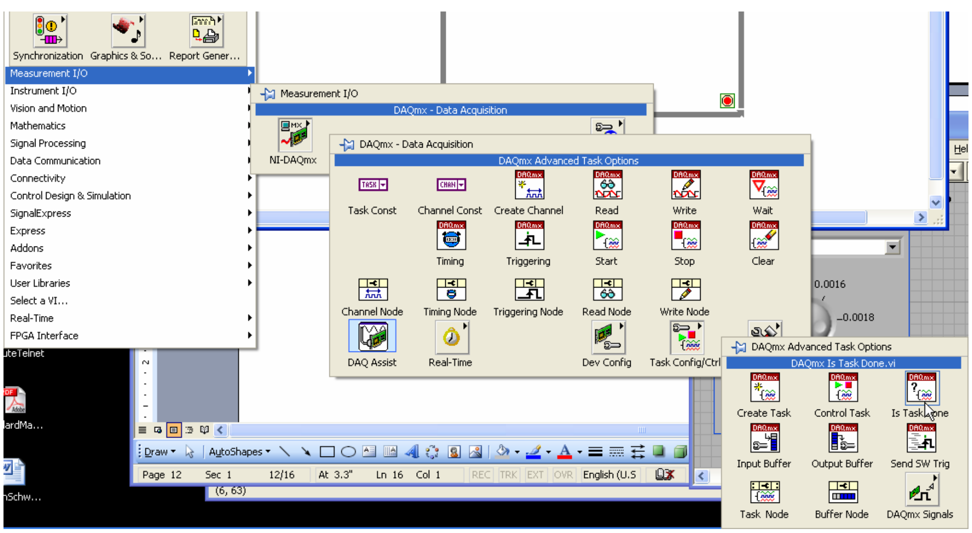


Figure 3-3B: Drag a DAQmx Is Task Done element into the While Loop

Next, this DAQmx element's error output will be monitored. If triggered, or if `Stop` is clicked, the program will terminate. As such, monitor status by clicking `Cluster & Variant` and choosing "`Unbundle by Name`" element (see **Figure 3-3C**). Refer to **Figure 3-3A** for wiring.

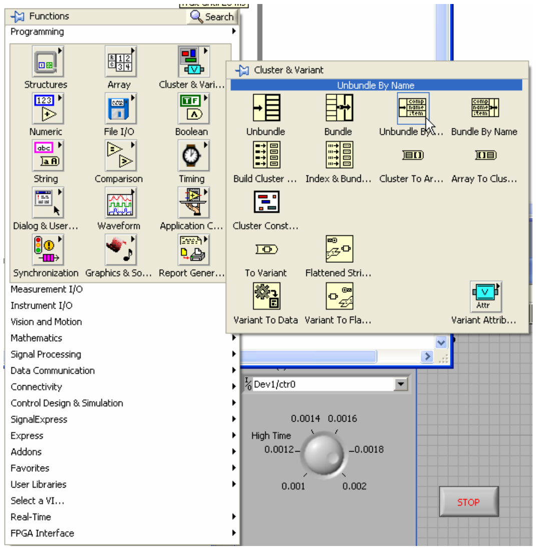


Figure 3-3C: "`Unbundle By Name`" element

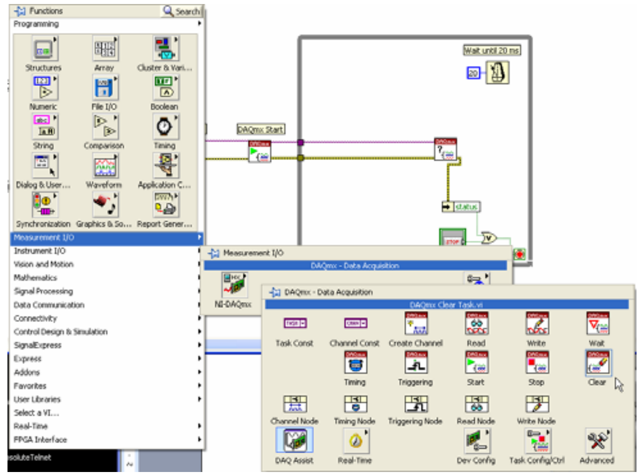


Figure 3-3D: `DAQmx Clear Task` element

Finally, add a `DAQmx Clear Task` element. Referring to **Figure 3-3D**, choose `Measurement I/O – NI-DAQmx – Clear`. Drag and place this element outside the `While Loop`. The finished block diagram should look like **Figure 3-3A**.

Step 4: Block Diagram – updating the pulse train dynamically

The completed block diagram is given in Figure 3-4A. A Case structure has been added to ensure one can update the pulse width HI time dynamically (i.e. during run-time). Steps follow.

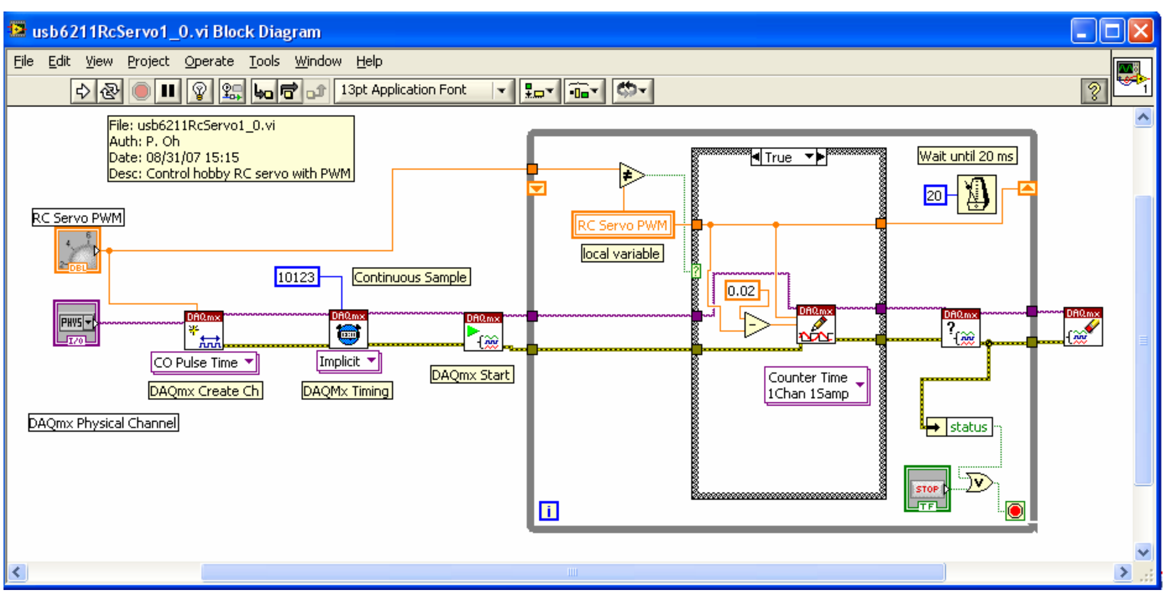


Figure 3-4A: Completed Block Diagram. Note: Case structure is inside the While Loop

Referring to **Figure 3-4B**, create a `Case` structure inside the `While` loop. Drag and place a `DAQmx Write` element inside the `Case` structure. Right click this `DAQmx Write` element and choose `Select Type – Counter – Single Channel – Single Sample – Time`.

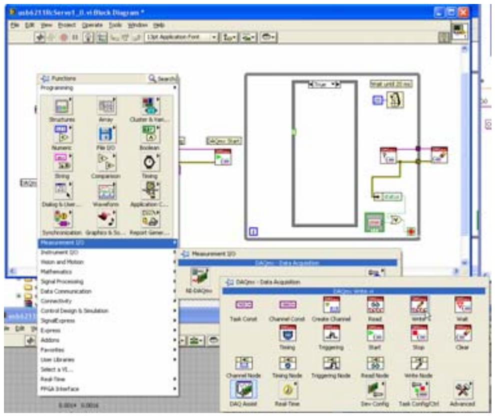


Figure 3-4B: `DAQmx Write` element

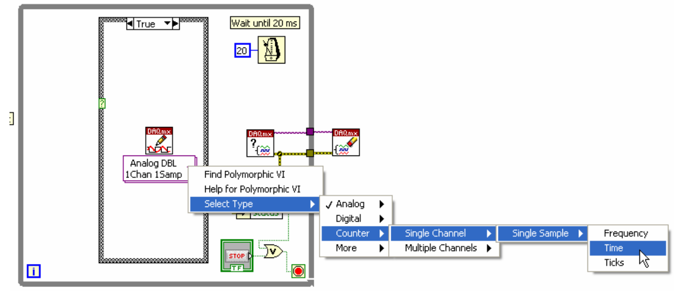


Figure 3-4C: Configure DAQmx for Time

Add a `Shift Register` to the `While` loop. Connect the `Knob` element (named RC Servo PWM) to the `Shift Register`. Drag a `Not Equal` logic element in the `While` loop (leave unwired for now). Next, drag a `Subtraction` element. Add a `Constant` (0.02) to subtract from. See **Figure 3-4D**.

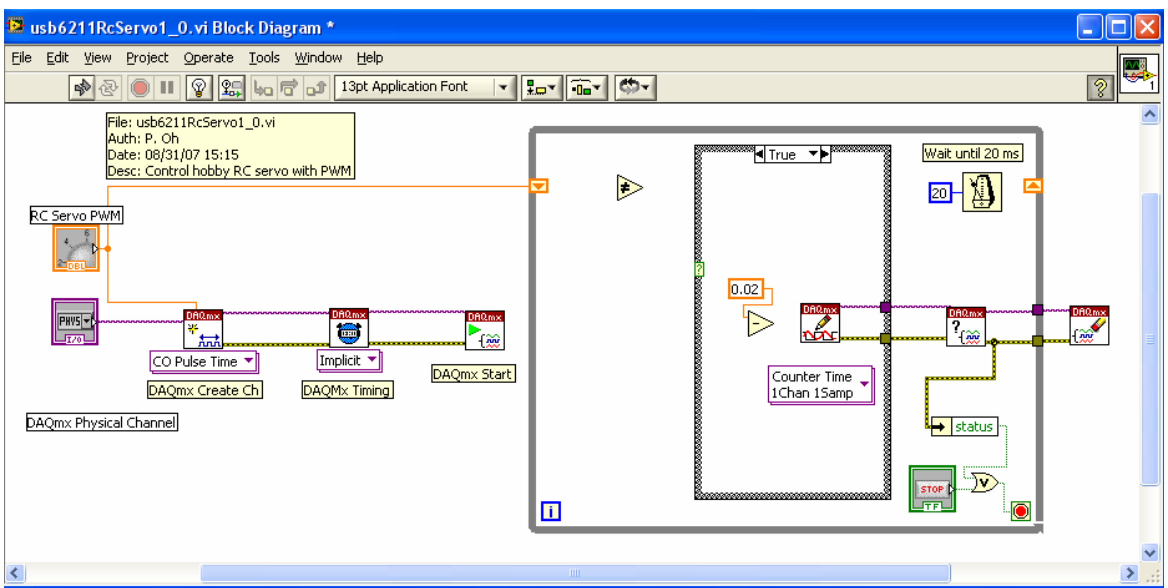


Figure 3-4D: `Shift register`, `Not Equal` and `Subtraction` elements

Next, right click on the Knob element and choose `Create – Local Variable` (see **Figure 3-4E**). Select "`RC Servo PWM`" and place inside the `While` Loop. Right click on this `Local Variable` and select `Change To Read` (see **Figure 3-4F**)

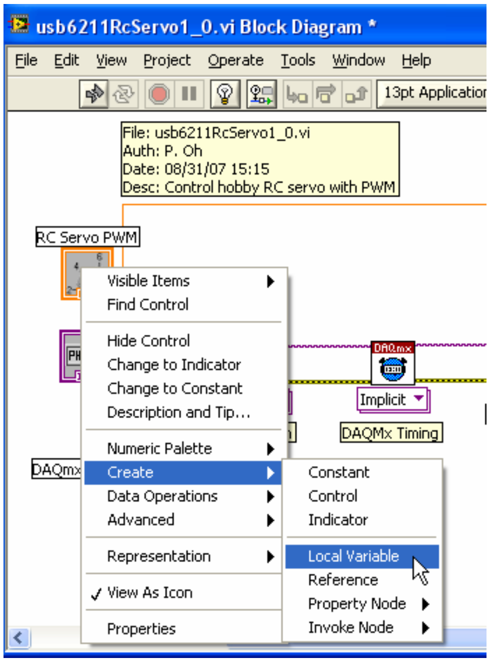


Figure 3-4E: Make a Local Variable

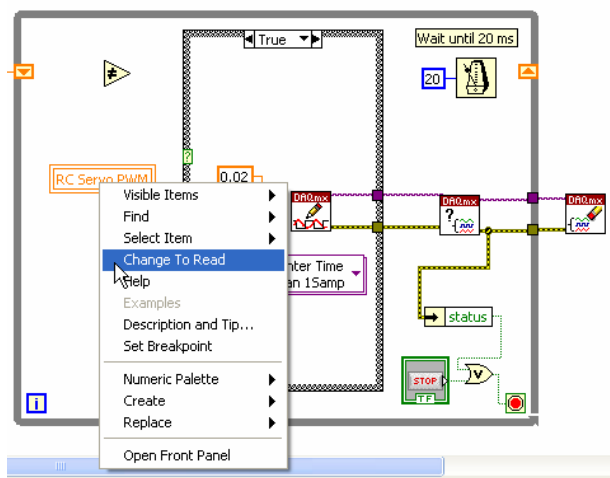


Figure 3-4F: Configure for `Change To Read`

As described earlier, the Case structure is employed to ensure the current knob value is used to update the pulse width's HI value. As such, the Shift Register value and Local Variable are compared. Wire up these elements into the Not Equal element. Wire the Not Equal element's output to the Case structure. Refer to **Figure 3-4G**.

When the `Case` structure is `True`, one must feed the current `Knob`'s value into the `DAQmx Write` element's `high time` pin. This value is subtracted from 0.020 and fed into the `DAQmx Write` element's `low time` pin. Refer to **Figure 3-4H**

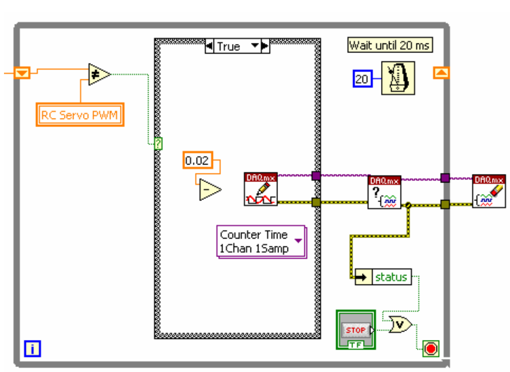


Figure 3-4G: Comparing values

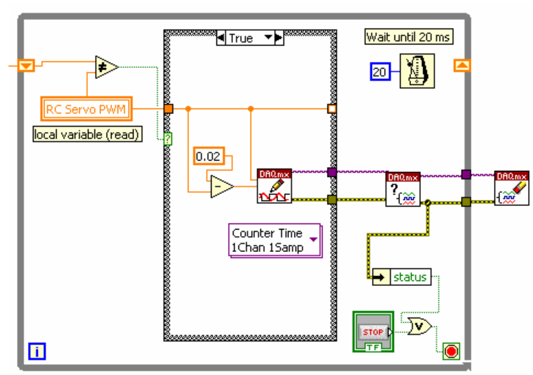


Figure 3-4H: Connect low and high times

Next, make the `Task In/Out` and `Error` connections between the `DAQmx Start` and `DAQmx Write` (see **Figure 3-4I**). In the `Case` structure's `False` condition, the `Local Variable` should not be changed (see **Figure 3-4J**). As such, simply feed the connections through.

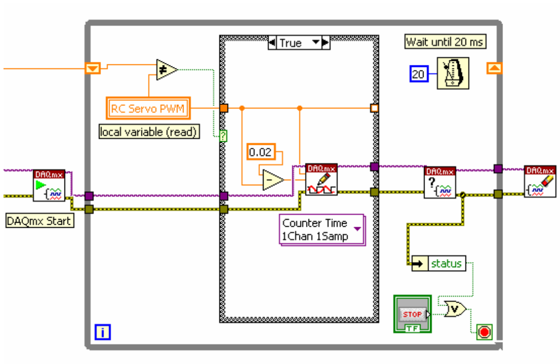


Figure 3-4I: Wire the Task and Error pins

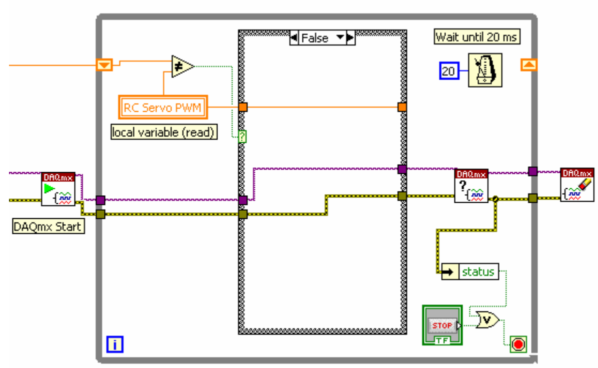


Figure 3-4J: False case – values pass unchanged

The last thing is to wire up the `Local Variable` "`RC Servo PWM`" to the `Shift Register` (far right side). Also, one needs to wire a constant (value is `10123`) into the "`sample mode`" pin of the `DAQmx Timing` element. The value `10123` represents "`Continuous Sample`". See **Figure 3-4K** for the final block diagram. Save your program as **`usb6211RcServo1_0.vi`**

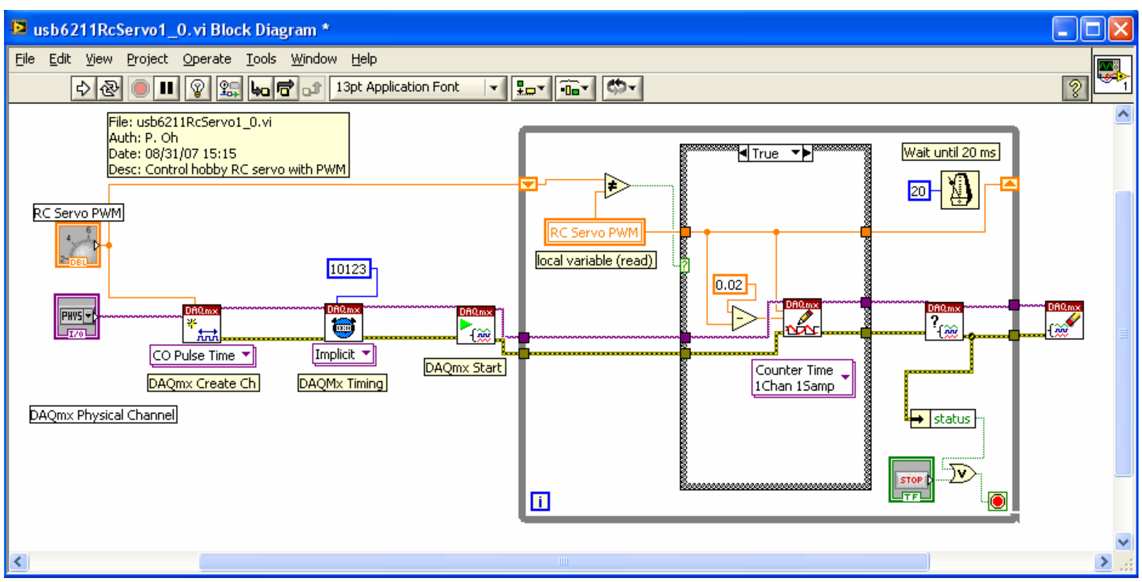


Figure 3-4K: Complete block diagram with value 10123 for sample mode

Step 4: Wire up USB-6211 and RC Servo

A typical RC hobby servo has 3 wires: +5V, GND and PWM. These connect to the USB-6211 as follows (refer to intro Photo is necessary).

RC Servo	USB-6211
Red (+5V)	Pin 10 (+5V)
Black (GND)	Pin 5 (DGND)
White (PWM)	Pin 6 (P1.0)

After screwing these wires together, click Play on the front panel. Rotate the virtual knob to see the corresponding shaft rotation.

Exercise 3:

3-1 Observe the servo rotate. Why does it rotate slowly? Hint: What is the USB-6211 output current? What is the typical running current of a DC motor?

3-2 If an oscilloscope is available, connect a probe to the PWM signal. Record the frequency and HI and LO pulse width times.

3-3 The USB-6211 features two 32-bit timers/counters. Create a VI that controls two RC hobby servos. Your front panel would have two knobs (one for each servo).