```
*****************************************************************
```
Pseudo-code for Shooting SM (Lower Level)

```
Module variables:
States: ALIGNING, RAMP_SPEED, SHOOT_BALL, QUERYINGSHOTSTATUS
Events Posted:
```

RunShootingSM

```
Takes ES_Event CurrentEvent, returns CurrentEvent

Set MakeTransition variable to false, because we are not making a
transition currently
Set state type variable NextState to CurrentState
Set event type EntryEventKind to ES_ENTRY (default to normal entry to
new state)
Set event type ReturnEvent to CurrentEvent

Switch (CurrentState)
      Case ALIGNING
            Execute During function for ALIGNING
            If there is still an event to process (not ES_NO_EVENT)
                  Switch (Event)
                        Case: ES_ALIGNED
                              Set NextState to RAMP_SPEED
                              Set MakeTransition to true
                              Set ReturnEvent to ES_NO_EVENT
                        End Case
                  End Switch
            End if
      End Case

      Case RAMP_SPEED
            Call the DuringRAMP_SPEED function
            Set CurrentEvent to returned event from during function
            If there is still an event to process (not ES_NO_EVENT)
                  Switch (CurrentEvent)
                        Case ES_RAMPED
                              Set next state to SHOOT_BALL
                              Set MakeTransition to true
                              Set ReturnEvent to ES_NO_EVENT (consumed)
                        End Case
                  End Switch
            End if
      End Case
```

```
      Case SHOOT_BALL
            Call the DuringSHOOT_BALL function
            Set CurrentEvent to returned event from during function
            If there is still an event to process (not ES_NO_EVENT)
                  Switch (CurrentEvent)
                        Case ES_TIMEOUT
                              Set next state to QUERYINGSHOTSTATUS
                              Set MakeTransition to true
                              Set ReturnEvent to ES_NO_EVENT (consumed)
                        End Case
                  End Switch
            End if
      End Case

/* dealing with QueryingShotStatus state changes in during function*/

      If MakeTransition is true (we are transitioning to a different
      state)
            Set the CurrentEvent to ES_EXIT
            Call RunShootingSM with CurrentEvent
            Set CurrentState to NextState
            Call RunMasterSM with ES_ENTRY event (start the entry
            function for the new state)
      Endif
Return ReturnEvent
End RunMasterSM
```

StartShootingSM
Takes ES_EVENT Current Event, returns nothing

```
Initialize CurrentState to ENTRY_STATE, which is ALIGNING
Call RunShootingSM with Current Event (ES_ENTRY event)
```

DuringAligning (handles aligning to the beacon)
Takes event, returns event

```
If event is ES_ENTRY
      Enable/unmask Beacon Detection interrupt on input capture
      Call to set the turning direction as CW
      Set turning speed by calling motor command with desire duty

      Store the shooting active status
      Store current shooting location
```

```
            If team is green
                  Store green goal active status
            Else if team is red
                  Store red goal active status
            End if
Else if event is ES_EXIT
      If exiting constructing state, give the lower levels a chance
      to clean up first
Else
      No lower level state machine to run
      Do the activity that is repeated as long as we are in this
state

Endif
Return Event (this event is either an event that ConstructingSM needs
to handle, or ES_NO_EVENT if a lower level SM handled it)
```

DuringRAMPSPEED

```
Takes event, returns event

If event is ES_ENTRY
      Call GoGoFlyWheel to start motor and related interrupts
      Start a 5s framework timer to allow for the motor to ramp

Else if event is ES_EXIT
      Nothing to do
Else
      No lower level state machine to run
      Do any activity that is repeated as long as we are in this
state
            If EventType is ES_TIMEOUT AND EventParam is FlyRampTimer
                  Post ES_RAMPED to MasterSM
            Endif
Return Event (this event is either an event that ConstructingSM needs
to handle, or ES_NO_EVENT if a lower level SM handled it)
```

DuringShootingBall

```
Takes event, returns event

If event is ES_ENTRY
      Change PWM duty/period to make the crank-slider push out a ball
      Start a framework called ShotsFired to fire in 2s
```

```
Else if event is ES_EXIT
      If exiting constructing state, give the lower levels a chance
      To clean up first

Else
      No lower level state machine to run
      Do any activity that is repeated as long as we are in this
state

Endif
Return Event (this event is either an event that ConstructingSM needs
to handle, or ES_NO_EVENT if a lower level SM handled it)
```

DuringQueryingShotStatus

```
Takes event, returns event

If event is ES_ENTRY
      Post ES_QUERY to Comm

Else if event is ES_EXIT
      If exiting constructing state, give the lower levels a chance
      To clean up first
            Call KillFlyWheel to turn off motor and speed control
Else
      No lower level state machine to run

      If number of cows = 0
            Post ES_DRIVE_LOAD
      Else If event is ES_TIMEOUT and param is CowResultWait
            Post ES_QUERY to CommSM
      Else if event is ES_RESPONSE_READY
            If shooting location is still open and we have cows
                  Post ES_SHOOT to ShootSM to shoot again
            Else drive to the next shooting location
                  ES_DRIVE_CHECKIN

             to end ShootingSM
            End if
      End if
Endif
Return Event (this event is either an event that ConstructingSM needs
to handle, or ES_NO_EVENT if a lower level SM handled it)
```

InitFlyInputCapture (initializes input capture to measure period between encoder edges)
Takes nothing, returns nothing (3/02/17 afb)

Start by enabling the clock to the timer [regular timer 1A]
Enable the clock to Port **F**
Make sure timer (Timer **1A**) is disabled before c onfiguring it
Set Timer **1A** up in 32-bit wide counter (individ ual, not concatenated) mode
Use the full 32 bit count, so initialize the Interval load to Oxffffffff Set up timer A in capture mode, for edge time, and upcounting
To set event to both edges, write proper bits to T **A**EVENT bits in GPTMCTL. Trigger on rising edges
Now set up the port to do capture (clock was enabled earlier)
- Start by setting the alternate function for Port **F** bit **2**
- Then, map bit **?**'s alternate function to T1CCP0 and select mux value
- Enable pin **2** on Port **F** to be an input
- Enable timer for local capture interrupt Enable Timer **1A** in
- Regular Timer **1A** in NVIC (Interrupt **21** so in EN**0** at bit **21**)

Make sure interrupts are enabled globally
Wait to kick off timer to enable it and enable timer to stall while stopped at debugger

End of InitFlyInputCapture

FlyInputCaptureResponse(an ISR to capture the times of rising and falling edges of the encoder)
Takes nothing, returns nothing [as all ISRs should!] (1/18/17 afb)
Declare 32 bit local variable called ThisCapture
As always, start by clearing the source of the interrupt, the input capture event
Grab the captured time value
Calculate the period
Add to array of 50 edges for averaging/smoothing
Update last capture to prepare for the next edge
Reset count in FlyWheelStopCheck one-shot
Create new event ThisEvent Set EventType of ThisEvent to ES_NewEdge
Post ThisEvent Event to SpeedCounter service
Return void

End of FlyInputCaptureResponse

InitBeaconDetect (initializes input capture for beacon finding)
Takes nothing, returns nothing

Enable Digital IN for BEACON on PD6
Set direction of pins to INPUT
Enable the interrupt event for PD6 rising edge
Enable interrupt in NVIC by writing to bit 8 in EN3
Change priority of event interrupt to 0. It is interrupt 104, so PRI26

End of InitBeaconDetect

BeaconDetectResponse (ISR to stop turning when beacon has been detected)
Takes nothing, returns nothing

Clear source of the interrupt
Increment BeacondDetected to keep track of how many pulses we got
If the Counter has reached the desired amount (to confirm that we are aligned with the beacon)
    Post ES_ALIGNED to MasterSM
    Stop Motor
    Mask the interrupt
Endif

End of BeaconDetectResponse

InitFlyControlPeriod (inits periodic interrupt timer for doing motor control)

Takes nothing, returns nothing (3/2/17 afb)
Start by enabling the clock to the timer (Wide Timer **4A**)
Kill a few cycles to let the clock get going
Make sure that timer (Timer **4A**) is disabled bef ore configuring
Set it up in 32bit wide (individual, not concatenated) mode
Set up timer **4A** in periodic mode so that it rep eats the time-outs
Set timeout to 2mS
Enable a local timeout interrupt

Enable the Timer **A** in Wide Timer **4** interrupt in the NVIC
it is interrupt number **102** so appears in EN **3** at bit **6**
Set control priority lower than encoder priority by writing **1** to NVIC priority register **PRI25** interrupt **3**
Make sure interrupts are enabled globally
Now wait to kick the timer off until we need it, but enable the timer to stall while stopped by the debugger

End of InitFlyControlPeriod

FlyControlResponse (an ISR to execute the control law for flywheel motor)
Takes nothing, returns nothing [as all ISRs should!] (3/2/17 afb)

Vars: STATIC float IntegralTerm, STATIC float RPMError, STATIC float LastError, STATIC unsigned 32bit int ThisPeriod [TargetRPM set as module level float]

Start by clearing the source of the interrupt

Implement control law
- ThisPeriod equals Period
- Calculate RPM by taking PER2RPM conversion divided by ThisPeriod
- Find RPMError by taking TargetRPM minus RPM
- IntegralTerm equals Integral Term plus IntegralGain times RPMError
- IntegralTerm equals value clamped between 0 and 100 as antiwindup
- To be compatible with Zeigler Nichols Tuning, RequestedDuty equals Kp*(RPMError + (IntegralTerm)) + Kd*(RPMError – LastError)
- Requested duty is clamped between zero and 100
- LastError equals RPMError to update last error for next round

SetDuty of Channel **??** to RequestedDuty
Lower signal line P **??** to show end of execution
End of ControlResponse

TO DO:
Set min/max rpm in constants
Determine per2rpm conversion in response
Set requested duty with motor code

InitFlyWheelStopCheck (initializes a one-shot to check if the motor has stopped)

ISR Init - Takes nothing and returns nothing
start by enabling the clock to the timer (Wide Timer 4)
Loop until timer hardware is ready
Disable timer B before configuring
Configure timer for 32bit (individual instead of concatenated)
Macro define 16bit refers to individual timer rather than actual 16bit
Set Timer a into one-shot mode (mask bits 0:1 and write value for 1-shot mode = 0x01)
set timeout to 500 ms
Enable local timeout interrupt. (TBTOIM = bit 0 maybe not?)
Enable interrupt in NVIC register; we have interrupt 103 so nvic_en3 bit 7
Change priority of one-shot to 2; we are using pri 25
Turn on interrupts globally
set timer to stall in debugging. We will wait until the start function to start the timer

FlyWheelStopCheck_ISR (ISR to fire if the flywheel has stopped s

ISR - Takes nothing, returns nothing
clear source of interrupt
Set RPM to zero
Clear values in the RPM calculation stream

**********Private Functions**********

clamp (a simple utility function to clamp value between to limits)

Takes an input val, a lower bound, and an upper bound, returns clamped value (3/02/17 afb)

if val is greater than ClampHigh, val is too high, so:
return ClampHigh
if val is less than ClamLow, val is too low, so:
return ClampLow
otherwise return val because val is just right

PID ISR

ShootingState is the return value of QueryShootingSM
If ShootingState is RAMP_SPEED AND speed is > +/- 5% of desired speed
Post ES_RAMPED to ShootSM

GoGoFlyWheel (utility function to activate motor and interrupt function for the flywheel)
Takes nothing, returns nothing (afb 3/3/17)

Turn motor on
Enable interrupt for FlyInputCaptureResponse to detect encoder edges and thereby
Enable interrupt for FlyWheelStopCheck to detect when wheel is stopped

End of GoGoFlyWheel

KillFlyWheel (utility function to deactivate motor and interrupt function for the flywheel)
Takes nothing, returns nothing (afb 3/3/17)

Turn motor off
Disable interrupt for FlyInputCaptureResponse to detect encoder edges and thereby
Disable interrupt for FlyWheelStopCheck to detect when wheel is stopped

End of KillFlyWheel

Right now, for simplicity we set a 5s timer for the flywheel to get up to speed. If we want a more responsive system, we can implement an event checker and flag that will be raised when the speed is within a certain range of the desired.

```
/*****************************************************************
*
 Function
    GetFlySpeedState

 Parameters
   Nothing

 Returns
   boolean

 Description
  Public function to return a boolean if the motor is
 Notes

 Author
   Drew Bell, 03/03/17, 15:23
*****************************************************************
/

bool GetFlySpeedState( void ) {
     // add code
}


/*****************************************************************
********
 Function
   CheckFlyUp2Speed
 Parameters
   None
 Returns
   bool: true if a new event was detected
 Description
   Checks to see if the flywheel is up to speed
 Notes

 Author
```

```
   Drew Bell, 03/03/17, 13:48
*************************************************************************
*******/
bool CheckFlyUp2Speed(void)
{
  static uint8_t LastFlySpeedState = 0;
  uint8_t CurrentFlySpeedState;
  bool ReturnVal = false;

  CurrentFlySpeedState =  GetFlySpeedState();
  // check for pin high AND different from last time
  // do the check for difference first so that you don't bother with
a test
  // of a port/variable that is not going to matter, since it hasn't
changed
  if ( (CurrentFlySpeedState != LastFlySpeedState) &&
       (CurrentFlySpeedState == SPEED_CORRECT) )
  {                     // event detected, so post detected event
    printf("speed correct, shooter ready /n/r");
    ReturnVal = true;
  }
  LastFlySpeedState = CurrentFlySpeedState; // update the state for
next time
*/
  return ReturnVal;
```