```c
/****************************************************************************
 Module
   ShootSM.c

 Description
   This is based on template file for implementing state machines.

****************************************************************************/
/*----------------------------- Include Files -----------------------------*/
// Basic includes for a program using the Events and Services Framework
#include "ES_Configure.h"
#include "ES_Framework.h"

/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine */
#include "ShootSM.h"
#include "Location.h"
#include "LOCMaster.h"
#include "MasterVehicle.h"
#include "PWMTiva.h"
#include "Servo_Control.h"

/*----------------------------- Module Defines ----------------------------*/
// define constants for the states for this machine
// and any other local defines
#define NORMAL_OPERATION // this is normal mode which include location control loop
//#define TESTING_SHOOTER // this is testing mode, it removes the location control loop
#define ENTRY_STATE SHOOT_WAITING
#define RED 0
#define GREEN 1
#define ONE_SEC 976
#define HALF_SEC (ONE_SEC /2)
#define TWO_SEC (ONE_SEC * 2)
#define FOUR_HALF_SEC 4500
#define THREE_SEC 3000
#define FIVE_SEC (ONE_SEC *5)
#define LATCH_DISPENSE_TIME HALF_SEC
#define TENSION_SPRING_TIME ONE_SEC
#define RESET_TIME HALF_SEC
#define START_SHOOTER_TIME 500
#define LATCH_SERVO_CHANNEL         2
#define SPRING_SERVO_CHANNEL        3
#define DISPENSE_BALL_SERVO_CHANNEL 4
#define UNLATCH_ANGLE 110
#define LATCH_ANGLE 10
#define UNTENSION_ANGLE 0
#define TENSION_ANGLE 130
#define DISPENSE_ANGLE 20
#define RESET_DISPENSE_ANGLE 90
#define COMPETITION_FULL_DUTY 75
#define CORRECTION_FULL_DUTY 55

/*---------------------------- Module Functions ---------------------------*/
/* prototypes for private functions for this machine, things like during
   functions, entry & exit functions.They should be functions relevant to the
   behavior of this state machine
*/
static ES_Event DuringWaiting( ES_Event Event);
static ES_Event DuringMoveY( ES_Event Event);
static ES_Event DuringMoveX( ES_Event Event);
static ES_Event DuringShooting( ES_Event Event);
```

```
static ES_Event DuringResetShooter( ES_Event Event);
static void unlatch_LatchServo(void);
static void latch_LatchServo(void);
static void dispense_Ball(void);
static void reset_DispenseServo(void);
static void tension_SpringServo(void);
static void relax_SpringServo(void);

/*---------------------------- Module Variables ----------------------------*/
// everybody needs a state variable, you may need others as well
static ShootingState_t CurrentState;
static uint32_t CurrentXDestination=0;
static uint32_t CurrentYDestination=0;
static bool shooter_reset_complete = true;
static int before_shooting_goal;
static int after_shooting_goal;
static bool last_eighteen_second = false;
static bool score_changed_flag = false;
static uint32_t SHOT_DELAY_TIME = FOUR_HALF_SEC; // set shot delay time, we changed it
according to current situation

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
    RunShootSM

 Parameters
   ES_Event: the event to process

 Returns
   ES_Event: an event to return

****************************************************************************/
ES_Event RunShootSM( ES_Event CurrentEvent )
{
   bool MakeTransition = false;/* are we making a state transition? */
   ShootingState_t NextState = CurrentState;
   ES_Event EntryEventKind = { ES_ENTRY, 0 };// default to normal entry to new state
   ES_Event ReturnEvent = CurrentEvent; // assume we are not consuming event

   switch ( CurrentState )
   {
      case SHOOT_WAITING : // If current state is waiting state
         // Execute During function for state one. ES_ENTRY & ES_EXIT are
         // processed here allow the lower level state machines to re-map
         // or consume the event
         CurrentEvent = DuringWaiting(CurrentEvent);
         //process any events
         if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
         {
            switch (CurrentEvent.EventType)
            {
                case SHOOT_ACTIVE : //If event is SHOOT_ACTIVE
                    reset_shot_delay_time(); // we reset shot delay timer to 4.5 seconds
                    uint8_t dest_index = 0;
                    if(get_Team() == GREEN){
                        dest_index = queryActiveShootingGreen(); //get destination
                        if ((dest_index!=0)&&(dest_index!=5)){
                            if(dest_index == 4){
                            // when destination is 4, this means it's last 18 sec
                                dest_index = 1; // set destination to G1
```

```
                            printf("Set last eighteen second flag in shooting SM\r\n");
                            last_eighteen_second = true;
                            //for last 18 sec, we reduce shot delay time to half a sec
                            set_shot_delay_for_last_eighteen();
                        }
                        CurrentXDestination = get_Shoot_Green_X(dest_index);
                        CurrentYDestination = get_Shoot_Green_Y(dest_index);
                    }
                }
                else{
                    dest_index = queryActiveShootingRed(); //get destination
                    if ((dest_index!=0)&&(dest_index!=5)){
                        if(dest_index == 4){
                        // when destination is 4, this means it's last 18 sec
                            dest_index = 1; // set destination to R1
                            printf("Set last eighteen second flag in shooting SM\r\n");
                            last_eighteen_second = true;
                            //for last 18 sec, we reduce shot delay time to half a sec
                            set_shot_delay_for_last_eighteen();
                        }
                        CurrentXDestination = get_Shoot_Red_X(dest_index);
                        CurrentYDestination = get_Shoot_Red_Y(dest_index);
                    }
                }

                // Execute action function for state one : event one
                NextState = SHOOT_MOVE_Y;//Decide what the next state will be
                // for internal transitions, skip changing MakeTransition
                MakeTransition = true; //mark that we are taking a transition
                // if transitioning to a state with history change kind of entry
                // EntryEventKind.EventType = ES_ENTRY_HISTORY;
                // optionally, consume or re-map this event for the upper
                // level state machine
                ReturnEvent.EventType = ES_NO_EVENT;
                break;

            default:
                break;
        }
    }
    break;

case SHOOT_MOVE_Y :       // If current state is SHOOT_MOVE_Y
    // Execute During function for state one. ES_ENTRY & ES_EXIT are
    // processed here allow the lower level state machines to re-map
    // or consume the event
    CurrentEvent = DuringMoveY(CurrentEvent);
    //process any events
    if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
    {
        switch (CurrentEvent.EventType)
        {
            case ES_TIMEOUT: // if we get time out event, consume it
                if(CurrentEvent.EventParam == STAGE_TIMER){
                    ReturnEvent.EventType = ES_NO_EVENT;
                }
                break;

            case Y_REACHED : //If event is reaching desination y
                // Execute action function for state one : event one
                NextState = SHOOT_MOVE_X; // set next state to moving in x
```

```
                // for internal transitions, skip changing MakeTransition
                MakeTransition = true; //mark that we are taking a transition
                // if transitioning to a state with history change kind of entry
                // EntryEventKind.EventType = ES_ENTRY_HISTORY;
                // optionally, consume or re-map this event for the upper
                // level state machine
                ReturnEvent.EventType = ES_NO_EVENT;
                break;

            case CONSTRUCTION_END: //If event is construction end
                NextState = SHOOT_WAITING; // set next state to waiting
                MakeTransition = true; //mark that we are taking a transition
                ReturnEvent.EventType = CONSTRUCTION_END; // post this event to upper SM
                break;

            default:
                break;
        }
    }
    break;

case SHOOT_MOVE_X :       // If current state is SHOOT_MOVE_X
    // Execute During function for state one. ES_ENTRY & ES_EXIT are
    // processed here allow the lower level state machines to re-map
    // or consume the event
    CurrentEvent = DuringMoveX(CurrentEvent);
    //process any events
    if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
    {
        switch (CurrentEvent.EventType)
        {
            case ES_TIMEOUT: // if we get time out event, consume it
                if(CurrentEvent.EventParam == STAGE_TIMER){
                    ReturnEvent.EventType = ES_NO_EVENT;
                }
                break;

            case X_REACHED: //If event is reaching x destination
                #ifdef NORMAL_OPERATION // for normal operation
                if(verify_y_location()){ //reverify y location
                    // if y location is correct, we set PWM to normal speed
                    set_PWM_Full_Duty(COMPETITION_FULL_DUTY);
                    NextState = RESET_SHOOTER;
                }
                else{
                    // else, we set PWM to slower speed for location correction
                    set_PWM_Full_Duty(CORRECTION_FULL_DUTY);
                    // return to shooting move y state
                    NextState = SHOOT_MOVE_Y;
                }
                #endif

                #ifdef TESTING_SHOOTER // for testing mode, we remove verifying loop
                NextState = RESET_SHOOTER;
                #endif
                MakeTransition = true;
                ReturnEvent.EventType = ES_NO_EVENT;
                break;

             case CONSTRUCTION_END: //If event is construction end
                NextState = SHOOT_WAITING; // set next state to waiting
```

```
                MakeTransition = true; //mark that we are taking a transition
                ReturnEvent.EventType = CONSTRUCTION_END; //return this event to upper SM
                break;

             default:
                break;
          }
       }
       break;

    case SHOOTING :       // If current state is shooting
       CurrentEvent = DuringShooting(CurrentEvent);
       //process any events
       if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
       {
          switch (CurrentEvent.EventType)
          {
          case SCORE_CHANGED: // when score change, we post this event to MasterVehicle
              PostMasterVehicleSM(CurrentEvent);
              // we consume this event, because we have to make sure we go to reset
              // finished shooting first, if we already started the process
              ReturnEvent.EventType = ES_NO_EVENT;
              break;

          case ES_TIMEOUT: //If event is time out
              // if we get Stage timer timeout, ignore it: consume this event
              if(CurrentEvent.EventParam == STAGE_TIMER){
                  ReturnEvent.EventType = ES_NO_EVENT;
              }

              // if we get start shooting timer timeout event
              if(CurrentEvent.EventParam == START_SHOOTER_TIMER){
                  // first we latch the latch servo
                  latch_LatchServo();
                  // Move dispenser servo to down position to dispense ball
                  dispense_Ball();
                  // set flag for reset
                  shooter_reset_complete = false;
                  // start delay timer to make sure this process is done before moving on
                  ES_Timer_InitTimer(LATCH_DISPENSE_TIMER, LATCH_DISPENSE_TIME);
                  ReturnEvent.EventType = ES_NO_EVENT;
              }
              // if we get latch dispense timer timeout
              else if(CurrentEvent.EventParam == LATCH_DISPENSE_TIMER){
                  // first reset dispense servo arm
                  reset_DispenseServo();
                  // at the same time, tension the spring servo
                  tension_SpringServo();
                  // make sure that this reset flag is still false
                  shooter_reset_complete = false;
                  // start timer to make sure the this process is completed
                  ES_Timer_InitTimer(READY_TO_SHOOT_TIMER, TENSION_SPRING_TIME);
                  ReturnEvent.EventType = ES_NO_EVENT;
              }

              // When dispensing the ball is completed, we shoot
              else if(CurrentEvent.EventParam == READY_TO_SHOOT_TIMER){
                  // Unlatch the latch servo to shoot
                  unlatch_LatchServo();
                  // make sure that this reset flag is still false
                  shooter_reset_complete = false;
```

```
            // decrement number of ball we have
            decrement_num_ball();
            // start shot delay timer
            ES_Timer_InitTimer(SHOT_DELAY_TIMER, SHOT_DELAY_TIME);
            // here we still not changing state,have to wait for the timer
            // to timeoout to make sure it finishes shooting
            // however, if number of ball is less than 0, we post
            // no_ball event to the main state machine
            if (get_num_ball() <= 0){
                ReturnEvent.EventType = ES_NO_EVENT;
                ES_Event post_no_ball;
                post_no_ball.EventType = NO_BALL;
                PostMasterVehicleSM(post_no_ball);
            }
            else{
                ReturnEvent.EventType = ES_NO_EVENT;
            }
        }
                            // This timeout signifies shooting is complete
        else if (CurrentEvent.EventParam == SHOT_DELAY_TIMER){
            NextState = RESET_SHOOTER; // make transition to reset_shooter state
            MakeTransition = true;
            ReturnEvent.EventType = ES_NO_EVENT;
            shooter_reset_complete = false;
        }
        // if 20 second timeout and we haven't shot,
        //then we don't shoot and go back to waiting
        else if((CurrentEvent.EventParam == SHOOT_20S_TIMER) &&
        (shooter_reset_complete == true)){
            NextState = SHOOT_WAITING; // set next state to waiting
            MakeTransition = true;
            ReturnEvent.EventType = ES_TIMEOUT;
        }
        // if 20 second timeout, and we shot already, we go to reset the shooter first
        else if ((CurrentEvent.EventParam == SHOOT_20S_TIMER) &&
        (shooter_reset_complete == false)){
            NextState = RESET_SHOOTER; // set next state to reset_shooter
            MakeTransition = true;
            ReturnEvent.EventType = ES_NO_EVENT;
            PostMasterVehicleSM(CurrentEvent); //Repost this event to master SM
            // so we can handle this in the reset_shooter state
        }
        else{
            // print out for debugging timer we do not handle
            printf("In shooting stage, this timer should not be here: %d\r\n",
            CurrentEvent.EventParam);
        }
        break;

    // if we get a no_ball event, go to reset and repost this event to main SM
    case NO_BALL:
        // if we need to reset, wait for shooting to finish and go to reset state
        if(shooter_reset_complete == false){
            ReturnEvent.EventType = ES_NO_EVENT;
            PostMasterVehicleSM(CurrentEvent);
        }
        // else, go back to waiting and return no_ball event
        else{
            NextState = SHOOT_WAITING;
            MakeTransition = true;
            ReturnEvent.EventType = NO_BALL;
```

```
            }
            break;

        case CONSTRUCTION_END: //If event is construction end
            NextState = SHOOT_WAITING; // set next state to waiting state
            MakeTransition = true; //mark that we are taking a transition
            ReturnEvent.EventType = CONSTRUCTION_END;
            break;

        default:
            break;
        }
    }
    break;

case RESET_SHOOTER :       // If current state is state one
    // Execute During function for state one. ES_ENTRY & ES_EXIT are
    // processed here allow the lower level state machines to re-map
    // or consume the event
    CurrentEvent = DuringResetShooter(CurrentEvent);
    //process any events
    if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
    {
        switch (CurrentEvent.EventType)
        {
            // Here, we can exit shooting SM only when we finished reset shooter
            // When reset_delay_timer is done, we will set the boolean reset_complete to
            // true if score_changed, construction_end or ES_timeout that is not reset
            // timer comes, we can post these event back to mastervehicle until the
            // boolean reset_complete is set to true when the shooter is reset, we also
            // query the system to see if score change or not when reset_complete is true
            // and score is not changed that we can go back to shooting state

            case SCORE_CHANGED:
                printf("Shooting received score changed event - return to waiting\r\n");
                NextState = SHOOT_WAITING;
                MakeTransition = true;
                ReturnEvent.EventType = SCORE_CHANGED; // return this event to upper SM
                break;

            case ES_TIMEOUT:
               if(CurrentEvent.EventParam == SHOOT_20S_TIMER){
                  // if reset is completed, go back to waiting state
                  if (shooter_reset_complete == true){
                     NextState = SHOOT_WAITING;
                     MakeTransition = true;
                     ReturnEvent.EventType = ES_TIMEOUT; // return this event to upper SM
                  }
                  else{
                     // if we get shoot 20s timeout, but we haven't finished reset shooter
                     // repost the timeout event until we finished resetting
                     PostMasterVehicleSM(CurrentEvent);
                     ReturnEvent.EventType = ES_NO_EVENT;
                  }
               }
               // When this timeout, we know that we have complete resetting the shooter
               else if(CurrentEvent.EventParam == SHOOTER_RESET_TIMER){
                  shooter_reset_complete = true;
                  //query the score
                  if(get_Team() == GREEN){
                      after_shooting_goal = get_green_score();
```

```
                }
                else{
                    after_shooting_goal = get_red_score();
                }
                //if it is last 18 sec, go straight back to shooting
                if(last_eighteen_second){
                    NextState = SHOOTING;
                    MakeTransition = true;
                    ReturnEvent.EventType = ES_NO_EVENT;
                }
                else{
                   // if score changed, go to waiting
                   if (after_shooting_goal > before_shooting_goal){
                       printf("We scored!!!\r\n");
                       NextState = SHOOT_WAITING;
                       MakeTransition = true;
                       ReturnEvent.EventType = SCORE_CHANGED;
                   }
                   // else go back to shooting state
                   else if (after_shooting_goal == before_shooting_goal){
                      NextState = SHOOTING;
                      MakeTransition = true;
                      ReturnEvent.EventType = ES_NO_EVENT;
                   }
                   else{
                      // print out for debugging
                      printf("Number of Goal is incorrect (goal should not
                      decrease).\r\n");
                   }
                }
            }
            else{
                printf("SOME timeout we don't know\r\n");
                printf("Timer %d\r\n", CurrentEvent.EventParam);
            }
            break;

        case NO_BALL:
           // if the reset is completed, go to waiting and
           // return this event to upper SM
           if (shooter_reset_complete == true){
              NextState = SHOOT_WAITING; // set next state to waiting
              MakeTransition = true; //mark that we are taking a transition
              ReturnEvent.EventType = NO_BALL;
           }
           else{
              // if we get no_ball event, but we haven't finished reset shooter
              // repost this event until we finished resetting
              PostMasterVehicleSM(CurrentEvent);
              ReturnEvent.EventType = ES_NO_EVENT;
           }
           break;

        case CONSTRUCTION_END: //If event is construction end
           NextState = SHOOT_WAITING; // set next state to waiting
           MakeTransition = true; //mark that we are taking a transition
           ReturnEvent.EventType = CONSTRUCTION_END;
           break;

        default:
           break;
```

```
            }
         }
         break;

      default:
         break;
    }
    //   If we are making a state transition
    if (MakeTransition == true)
    {
       //   Execute exit function for current state
       CurrentEvent.EventType = ES_EXIT;
       RunShootSM(CurrentEvent);

       CurrentState = NextState; //Modify state variable

       //   Execute entry function for new state
       // this defaults to ES_ENTRY
       RunShootSM(EntryEventKind);
     }
     return(ReturnEvent);
}
/****************************************************************************
 Function
     StartShootSM

 Parameters
     None

 Returns
     None

****************************************************************************/
void StartShootSM ( ES_Event CurrentEvent )
{
   // to implement entry to a history state or directly to a substate
   // you can modify the initialization of the CurrentState variable
   // otherwise just start in the entry state every time the state machine
   // is started
   if ( ES_ENTRY_HISTORY != CurrentEvent.EventType )
   {
        CurrentState = ENTRY_STATE;
   }
   // call the entry function (if any) for the ENTRY_STATE
   RunShootSM(CurrentEvent);
}

/****************************************************************************
 Function
     QueryShootSM

 Parameters
     None

 Returns
     ShootingState_t The current state of the Shoot state machine

 Description
     returns the current state of the Shoot state machine

****************************************************************************/
```

```
ShootingState_t QueryShootSM ( void )
{
   return(CurrentState);
}

/****************************************************************************
 private functions
 ****************************************************************************/

static ES_Event DuringWaiting( ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
    {
        // implement any entry actions required for this state machine

        // after that start any lower level machines that run in this state
        //StartLowerLevelSM( Event );
        // repeat the StartxxxSM() functions for concurrent state machines
        // on the lower level
    }
    else if ( Event.EventType == ES_EXIT )
    {
        // on exit, give the lower levels a chance to clean up first
        // check the initial score according to the current team
        if(get_Team() == GREEN){
            before_shooting_goal = get_green_score();
        }
        else{
            before_shooting_goal =  get_red_score();
        }
        // repeat for any concurrently running state machines
        // now do any local exit functionality

    }else
    // do the 'during' function for this state
    {
    }
    // return either Event, if you don't want to allow the lower level machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}

static ES_Event DuringMoveY( ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
    {
        // implement any entry actions required for this state machine
        // move in y direction
        move_Y(CurrentYDestination);
    }
    else if ( Event.EventType == ES_EXIT )
    {
        // make sure to stop motor when exiting this moving state
```

```
        stopMotor();

    }else
    {
    }
    return(ReturnEvent);
}

static ES_Event DuringMoveX( ES_Event Event)
    {
    ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
    {
        // implement any entry actions required for this state machine
        move_X(CurrentXDestination); // move in x
    }
    else if ( Event.EventType == ES_EXIT )
    {
        // stop the motor when exiting
        stopMotor();

    }else
    {
    }
    return(ReturnEvent);
}

static ES_Event DuringShooting( ES_Event Event)
    {
    ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
    {
        // implement any entry actions required for this state machine
        // start start shooter timer to trigger the shooting process
        ES_Timer_InitTimer(START_SHOOTER_TIMER, START_SHOOTER_TIME);
    }
    else if ( Event.EventType == ES_EXIT )
    {

    }else
    {
    }
    return(ReturnEvent);
}

static ES_Event DuringResetShooter( ES_Event Event)
    {
    ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
    {
        // implement any entry actions required for this state machine
        // start reset timer
```

```
        ES_Timer_InitTimer(SHOOTER_RESET_TIMER, RESET_TIME);
        // reset dispense servo arm
        reset_DispenseServo();
        // relax the spring servo to prepare for the next shot
        relax_SpringServo();
    }
    else if ( Event.EventType == ES_EXIT )
    {
    }else
    {
    }
    // return either Event, if you don't want to allow the lower level machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}

/****************************************************************************
 Helper Functions
 ****************************************************************************/
uint32_t queryShootingCurrentXDestination(void){
    return CurrentXDestination;
}

uint32_t queryShootingCurrentYDestination(void){
    return CurrentYDestination;
}

/****************************************************************************
 Private Functions
 ****************************************************************************/

static void unlatch_LatchServo(void){
    printf("Unlatch\r\n");
    moveToAngle(UNLATCH_ANGLE, LATCH_SERVO_CHANNEL);
}

static void latch_LatchServo(void){
    printf("Latch\r\n");
    moveToAngle(LATCH_ANGLE, LATCH_SERVO_CHANNEL);
}

static void reset_DispenseServo(void){
    printf("Reset Dispenser\r\n");
    moveToAngle(RESET_DISPENSE_ANGLE, DISPENSE_BALL_SERVO_CHANNEL);
}

static void dispense_Ball(void){
    printf("Dispense Ball\r\n");
    moveToAngle(DISPENSE_ANGLE, DISPENSE_BALL_SERVO_CHANNEL);
}

static void relax_SpringServo(void){
    printf("Relax Spring\r\n");
    moveToAngle(UNTENSION_ANGLE, SPRING_SERVO_CHANNEL);
}

static void tension_SpringServo(void){
    printf("Tension Spring\r\n");
    moveToAngle(TENSION_ANGLE, SPRING_SERVO_CHANNEL);
}
```

```c
static void set_last_eighteen_second(void){
    last_eighteen_second = true;
}

void set_shot_delay_for_last_eighteen(void){
    SHOT_DELAY_TIME = HALF_SEC;
}

void reset_shot_delay_time(void){
    SHOT_DELAY_TIME = FOUR_HALF_SEC;
}
```