

```

1  /*****
2  Module
3  SupplySM.c
4
5  Description
6  This is based on a template file for implementing state machines.
7
8  *****/
9  /*----- Include Files -----*/
10 // Basic includes for a program using the Events and Services Framework
11 #include "ES_Configure.h"
12 #include "ES_Framework.h"
13 #include "inc/hw_memmap.h"
14 #include "inc/hw_types.h"
15 #include "inc/hw_gpio.h"
16 #include "driverlib/gpio.h"
17 #include "inc/hw_sysctl.h"
18 #include "driverlib/sysctl.h"
19 #include "driverlib/pin_map.h" // Define PART_TM4C123GH6PM in project
20 #include "driverlib/gpio.h"
21 #include "inc/hw_timer.h"
22 #include "inc/hw_nvic.h"
23 /* include header files for this state machine as well as any machines at the
24 next lower level in the hierarchy that are sub-machines to this machine
25 */
26 #include "SupplySM.h"
27 #include "Location.h"
28 #include "MasterVehicle.h"
29 #include "LOCMaster.h"
30 #include "Location.h"
31 /*----- Module Defines -----*/
32 // define constants for the states for this machine
33 // and any other local defines
34
35 #define NORMAL_OPERATION
36 // #define TESTING_SUPPLY // for debugging
37 #define ENTRY_STATE SUPPLY_WAITING
38 #define TWO_SEC 2000
39 #define TEN_MS 10
40 #define THIRTY_MS 30
41 #define THREE_SEC 3000
42 #define HALF_SEC 500
43 #define MAX_BALL_RECEIVED 4
44 #define COMPETITION_FULL_DUTY 75
45 #define CORRECTION_FULL_DUTY 55
46 #define NF 0x08
47 /*----- Module Functions -----*/
48 /* prototypes for private functions for this machine, things like during
49 functions, entry & exit functions. They should be functions relevant to the
50 behavior of this state machine
51 */
52 static ES_Event DuringWaiting( ES_Event Event);
53 static ES_Event DuringMoveX( ES_Event Event);
54 static ES_Event DuringMoveY( ES_Event Event);
55 static ES_Event DuringPulseSupply( ES_Event Event);
56
57 /*----- Module Variables -----*/
58 // everybody needs a state variable, you may need others as well
59 static SupplyingState_t CurrentState;
60 static bool flag_10ms_timer = false;
61 static bool flag_30ms_timer = false;
62 static bool flag_3000ms_timer = false;
63 static int pulse_count = 0;
64 static bool supply_led_on = false;
65 static bool loaded_complete = false;
66 static uint32_t OneShotTimeout_10ms = 40000000*10/1000;
67 static uint32_t OneShotTimeout_30ms = 40000000*30/1000;
68 static int counter = 0;
69 static bool count_valid = true;
70 /*----- Module Code -----*/
71 /*****
72 Function

```

```

73 RunSupplySM
74
75 Parameters
76 ES_Event: the event to process
77
78 Returns
79 ES_Event: an event to return
80
81 *****/
82 ES_Event RunSupplySM( ES_Event CurrentEvent )
83 {
84     bool MakeTransition = false; /* are we making a state transition? */
85     SupplyingState_t NextState = CurrentState;
86     ES_Event EntryEventKind = { ES_ENTRY, 0 }; // default to normal entry to new state
87     ES_Event ReturnEvent = CurrentEvent; // assume we are not consuming event
88
89     switch ( CurrentState )
90     {
91     case SUPPLY_WAITING :           // If current state is waiting
92         // Execute During function for state one. ES_ENTRY & ES_EXIT are
93         // processed here allow the lower level state machines to re-map
94         // or consume the event
95         CurrentEvent = DuringWaiting(CurrentEvent);
96         //process any events
97         if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
98         {
99             switch (CurrentEvent.EventType)
100             {
101             case ES_TIMEOUT: // if we get stage timeout, consume this event
102                 if(CurrentEvent.EventParam == STAGE_TIMER){
103                     ReturnEvent.EventType = ES_NO_EVENT;
104                 }
105                 break;
106             case NO_BALL: //If event is no_ball, go to move x state
107                 // Execute action function for state one : event one
108                 NextState = SUPPLY_MOVE_X;
109                 MakeTransition = true;
110                 ReturnEvent.EventType = ES_NO_EVENT;
111                 break;
112             default:
113                 break;
114             }
115         }
116         break;
117
118     case SUPPLY_MOVE_X :           // If current state is move x state
119         // Execute During function for state one. ES_ENTRY & ES_EXIT are
120         // processed here allow the lower level state machines to re-map
121         // or consume the event
122         CurrentEvent = DuringMoveX(CurrentEvent);
123         //process any events
124         if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
125         {
126             switch (CurrentEvent.EventType)
127             {
128             case ES_TIMEOUT:
129                 if(CurrentEvent.EventParam == STAGE_TIMER){
130                     ReturnEvent.EventType = ES_NO_EVENT;
131                 }
132                 break;
133
134             case X_REACHED : //If event is reaching destination x
135                 // Execute action function for state one : event one
136                 NextState = SUPPLY_MOVE_Y; // set next state to moving in y
137                 // for internal transitions, skip changing MakeTransition
138                 MakeTransition = true; //mark that we are taking a transition
139                 ReturnEvent.EventType = ES_NO_EVENT;
140                 break;
141             case CONSTRUCTION_END: //If event is construction end
142                 NextState = SUPPLY_WAITING; //set next state to waiting
143                 MakeTransition = true; //mark that we are taking a transition
144                 ReturnEvent.EventType = CONSTRUCTION_END; // return this event to upper SM

```

```

145         break;
146     default:
147         break;
148     }
149 }
150 break;
151
152 case SUPPLY_MOVE_Y: // if current state is moving in y
153 // Execute During function for state one. ES_ENTRY & ES_EXIT are
154 // processed here allow the lower level state machines to re-map
155 // or consume the event
156 CurrentEvent = DuringMoveY(CurrentEvent);
157 //process any events
158 if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
159 {
160     switch (CurrentEvent.EventType)
161     {
162     case ES_TIMEOUT:
163         //consume stage timer time out event
164         if(CurrentEvent.EventParam == STAGE_TIMER){
165             ReturnEvent.EventType = ES_NO_EVENT;
166         }
167         //if we finished ramming the supply, stop the motor
168         if(CurrentEvent.EventParam == SUPPLY_RAMMING_TIMER){
169             stopMotor();
170             set_location_checker_flag(true); //reenable location checker
171             NextState = PULSE_SUPPLY; //set next state to pulse supply
172             MakeTransition = true;
173             ReturnEvent.EventType = ES_NO_EVENT;
174         }
175         break;
176
177     case Y_REACHED: //If event is reaching y destination
178     #ifdef NORMAL_OPERATION
179     //during normal operation, we reverify if x location is correct
180     if(verify_x_location()){
181         //if x location is correct, set the speed to high speed
182         set_PWM_Full_Duty(COMPETITION_FULL_DUTY);
183         set_location_checker_flag(false);
184         //this function run the motor in the northward direction
185         runMotor(NF);
186         //start the ramming timer
187         ES_Timer_InitTimer(SUPPLY_RAMMING_TIMER, TWO_SEC);
188     }
189     else{
190         //if x location is incorrect, we set speed to correction speed (low speed)
191         set_PWM_Full_Duty(CORRECTION_FULL_DUTY);
192         //set next state to moving in x
193         NextState = SUPPLY_MOVE_X;
194         MakeTransition = true;
195     }
196     #endif
197
198     //during testing mode, we do not have location verification loop
199     #ifdef TESTING_SUPPLY
200     //set next state to pulse supply
201     NextState = PULSE_SUPPLY;
202     MakeTransition = true;
203     #endif
204     ReturnEvent.EventType = ES_NO_EVENT;
205     break;
206
207     case CONSTRUCTION_END: //If event is construction end
208     NextState = SUPPLY_WAITING; //set next state to waiting
209     MakeTransition = true; //mark that we are taking a transition
210     ReturnEvent.EventType = CONSTRUCTION_END; //return this event
211     break;
212     default:
213         break;
214     }
215 }
216 break;

```

```

217
218     case PULSE_SUPPLY :           // If current state is pulse supply
219     // Execute During function for state one. ES_ENTRY & ES_EXIT are
220     // processed here allow the lower level state machines to re-map
221     // or consume the event
222     CurrentEvent = DuringPulseSupply(CurrentEvent);
223     //process any events
224     if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
225     {
226         switch (CurrentEvent.EventType)
227         {
228             // consume score changed event
229             case SCORE_CHANGED:
230             ReturnEvent.EventType = ES_NO_EVENT;
231             break;
232
233             case ES_TIMEOUT: //If event is event one
234             //ignore stage timer timeout event
235             if(CurrentEvent.EventParam == STAGE_TIMER){
236                 ReturnEvent.EventType = ES_NO_EVENT;
237             }
238
239             // This timeout is for indicator LED
240             else if(CurrentEvent.EventParam == SUPPLY_LED_TIMER){
241                 if (!loaded_complete){
242                     if(supply_led_on){
243                         printf("LED ON\r\n");
244                         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, BIT3LO); //Light Supplying LED
245                         supply_led_on = false;
246                     }
247                     else{
248                         printf("LED OFF\r\n");
249                         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, BIT3HI); //Light Supplying LED
250                         supply_led_on = true;
251                     }
252                     //restart led timer
253                     ES_Timer_InitTimer(SUPPLY_LED_TIMER, HALF_SEC);
254                     ReturnEvent.EventType = ES_NO_EVENT; //Consume event
255                 }
256             }
257             else if(CurrentEvent.EventParam == SUPPLY_TIMER){
258                 // if we have maximum number of balls or more
259                 if(get_num_ball() >= MAX_BALL_RECEIVED){
260                     loaded_complete = true; //set loaded complete flag to true
261                     printf("now have max_num_ball balls, post LOADED_COMPLETE\r\n");
262                     //post event to master state machine to return to idle
263                     NextState = SUPPLY_WAITING;
264                     MakeTransition = true;
265                     ES_Event topost;
266                     topost.EventType = LOADED_COMPLETE;
267                     PostMasterVehicleSM(topost);
268                     ReturnEvent.EventType = ES_NO_EVENT;
269                 }
270                 //if we don't have 5 balls, continue requesting for ball
271                 else if(get_num_ball() < MAX_BALL_RECEIVED){
272                     pulse_count = 0;
273                     counter = 0;
274                     //write the pulsing line high
275                     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, BIT0HI);
276                     //increment pulse count counter
277                     //start a 10ms timer
278                     StartOneShot_Supply_10ms();
279                     ReturnEvent.EventType = ES_NO_EVENT;
280                 }
281                 else{
282                     // print for debugging purpose
283                     printf("We should not have more than 5 balls\r\n");
284                 }
285             }
286             else{
287                 printf("Some timer in Supply we don't deal with\r\n");
288                 CurrentEvent.EventType = ES_EXIT;

```

```

289         RunSupplySM(CurrentEvent);
290         ReturnEvent.EventType = ES_NO_EVENT;
291     }
292     break;
293
294     // if we get loaded complete, return this event to upper SM
295     case LOADED_COMPLETE:
296         NextState = SUPPLY_WAITING; //Decide what the next state will be
297         MakeTransition = true; //mark that we are taking a transition
298         ReturnEvent.EventType = LOADED_COMPLETE;
299         break;
300
301     case CONSTRUCTION_END: //If event is construction end
302         NextState = SUPPLY_WAITING; //set next state to waiting
303         MakeTransition = true; //mark that we are taking a transition
304         ReturnEvent.EventType = CONSTRUCTION_END;
305         break;
306     default:
307         break;
308 }
309 }
310 break;
311
312 default:
313     break;
314 }
315 // If we are making a state transition
316 if (MakeTransition == true)
317 {
318     // Execute exit function for current state
319     CurrentEvent.EventType = ES_EXIT;
320     RunSupplySM(CurrentEvent);
321
322     CurrentState = NextState; //Modify state variable
323
324     // Execute entry function for new state
325     // this defaults to ES_ENTRY
326     RunSupplySM(EntryEventKind);
327 }
328 return(ReturnEvent);
329 }
330 /*****
331 Function
332 StartSupplySM
333
334 Parameters
335 None
336
337 Returns
338 None
339
340 Description
341 Does any required initialization for this state machine
342
343 *****/
344 void StartSupplySM ( ES_Event CurrentEvent )
345 {
346     // to implement entry to a history state or directly to a substate
347     // you can modify the initialization of the CurrentState variable
348     // otherwise just start in the entry state every time the state machine
349     // is started
350     if ( ES_ENTRY_HISTORY != CurrentEvent.EventType )
351     {
352         CurrentState = ENTRY_STATE;
353     }
354     // call the entry function (if any) for the ENTRY_STATE
355     RunSupplySM(CurrentEvent);
356 }
357
358 /*****
359 Function
360 QuerySupplySM

```

```

361
362 Parameters
363 None
364
365 Returns
366 SupplhyingState_t The current state of the Supply state machine
367
368 Description
369 returns the current state of the Supply state machine
370
371 *****/
372 SupplyingState_t QuerySupplySM ( void )
373 {
374     return(CurrentState);
375 }
376
377 /*****
378 private functions
379 *****/
380 static ES_Event DuringWaiting( ES_Event Event)
381 {
382     ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption
383     return(ReturnEvent);
384 }
385
386 static ES_Event DuringMoveX( ES_Event Event)
387 {
388     ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption
389
390     // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
391     if ( (Event.EventType == ES_ENTRY) ||
392         (Event.EventType == ES_ENTRY_HISTORY) )
393     {
394         // implement any entry actions required for this state machine
395         // call move in x funtion
396         move_X(get_Supply_location_x());
397     }
398     else if ( Event.EventType == ES_EXIT )
399     {
400         // on exit, give the lower levels a chance to clean up first
401         // make sure the motor is not running when exiting this state
402         stopMotor();
403     }
404     else
405         // do the 'during' function for this state
406     {
407     }
408     // return either Event, if you don't want to allow the lower level machine
409     // to remap the current event, or ReturnEvent if you do want to allow it.
410     return(ReturnEvent);
411 }
412
413 static ES_Event DuringMoveY( ES_Event Event)
414 {
415     ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption
416
417     // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
418     if ( (Event.EventType == ES_ENTRY) ||
419         (Event.EventType == ES_ENTRY_HISTORY) )
420     {
421         // implement any entry actions required for this state machine
422         move_Y(get_Supply_location_y());
423         // set this flag to trigger speed control in location.c
424         // that is, when the robot get close to the supply destination,
425         // it will automatically reduces it speed so we don't ram into the wall
426         set_going_to_supply_flag();
427     }
428     else if ( Event.EventType == ES_EXIT )
429     {
430         // on exit, give the lower levels a chance to clean up first
431         // make sure the motor is not running when exiting this state
432         stopMotor();

```

```

433     clear_going_to_supply_flag(); //reset this flag
434
435 }else
436     // do the 'during' function for this state
437     {
438     }
439     // return either Event, if you don't want to allow the lower level machine
440     // to remap the current event, or ReturnEvent if you do want to allow it.
441     return(ReturnEvent);
442 }
443
444 static ES_Event DuringPulseSupply( ES_Event Event)
445 {
446     ES_Event ReturnEvent = Event; // assume no re-mapping or consumption
447
448     // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
449     if ( (Event.EventType == ES_ENTRY) ||
450         (Event.EventType == ES_ENTRY_HISTORY) )
451     {
452         // implement any entry actions required for this state machine
453         //start one shot timer for ir led pulsing
454         InitOneShotInt_Supply();
455         if(get_num_ball() < MAX_BALL_RECEIVED){
456             loaded_complete = false;
457         }
458         counter = 0;
459         //start a 10ms timer
460         StartOneShot_Supply_10ms();
461         //start supply LED timer
462         ES_Timer_InitTimer(SUPPLY_LED_TIMER, HALF_SEC);
463     }
464     else if ( Event.EventType == ES_EXIT )
465     {
466         // on exit, give the lower levels a chance to clean up first
467         //lower IR and raise indicator
468         printf("Set led low\r\n");
469         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, BIT0LO); //IR led low
470         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, BIT3HI); //Construction led high
471     }else
472         // do the 'during' function for this state
473         {
474         }
475         // return either Event, if you don't want to allow the lower level machine
476         // to remap the current event, or ReturnEvent if you do want to allow it.
477         return(ReturnEvent);
478     }
479
480
481 /* Private Function */
482
483 void InitOneShotInt_Supply( void ){
484     // start by enabling the clock to the timer (Wide Timer 4)
485     HWREG(SYSCTL_RCGCTIMER) |= SYSCTL_RCGCTIMER_R4;
486     // kill a few cycles to let the clock get going
487     while((HWREG(SYSCTL_PRWTIMER) & SYSCTL_PRWTIMER_R4) != SYSCTL_PRWTIMER_R4)
488     {
489     }
490     // make sure that timer (Timer B) is disabled before configuring
491     HWREG(WTIMER4_BASE+TIMER_O_CTL) &= ~TIMER_CTL_TBMEN; //TBMEN = Bit8
492     // set it up in 32bit wide (individual, not concatenated) mode
493     // the constant name derives from the 16/32 bit timer, but this is a 32/64
494     // bit timer so we are setting the 32bit mode
495     HWREG(WTIMER4_BASE+TIMER_O_CFG) = TIMER_CFG_16_BIT; //bits 0-2 = 0x04
496     // set up timer B in 1-shot mode so that it disables timer on timeouts
497     // first mask off the TAMR field (bits 0:1) then set the value for
498     // 1-shot mode = 0x01
499     HWREG(WTIMER4_BASE+TIMER_O_TBMR) =
500     (HWREG(WTIMER4_BASE+TIMER_O_TBMR) & ~TIMER_TBMR_TBMR_M) |
501     TIMER_TBMR_TBMR_1_SHOT;
502     // set timeout
503     HWREG(WTIMER4_BASE+TIMER_O_TBILR) = OneShotTimeout_10ms;
504     // enable a local timeout interrupt. TBTOIM = bit 8

```

```
505     HWREG(WTIMER4_BASE+TIMER_O_IMR) |= TIMER_IMR_TBTOIM; // 8
506     // enable the Timer B in Wide Timer 0 interrupt in the NVIC
507     // it is interrupt number 103 so appears in EN3 at bit 1
508     HWREG(NVIC_EN3) |= BIT7HI;
509     // make sure interrupts are enabled globally
510     __enable_irq();
511     //StartTime = ES_Timer_GetTime();
512     // now kick the timer off by enabling it and enabling the timer to
513     // stall while stopped by the debugger. TAEN = Bit0, TASTALL = bit1
514     HWREG(WTIMER4_BASE+TIMER_O_CTL) |= (TIMER_CTL_TBEN | TIMER_CTL_TBSTALL);
515 }
516
517 void StartOneShot_Supply_10ms( void ){
518     // start by grabbing the start time
519     //StartTime = ES_Timer_GetTime();
520     // now kick the timer off by enabling it and enabling the timer to
521     // stall while stopped by the debugger
522     HWREG(WTIMER4_BASE+TIMER_O_CTL) &= ~TIMER_CTL_TBEN;
523     HWREG(WTIMER4_BASE+TIMER_O_TBILR) = OneShotTimeout_10ms;
524     __enable_irq();
525     HWREG(WTIMER4_BASE+TIMER_O_CTL) |= (TIMER_CTL_TBEN | TIMER_CTL_TBSTALL);
526 }
527
528 void StartOneShot_Supply_30ms( void ){
529     // start by grabbing the start time
530     //StartTime = ES_Timer_GetTime();
531     // now kick the timer off by enabling it and enabling the timer to
532     // stall while stopped by the debugger
533     HWREG(WTIMER4_BASE+TIMER_O_CTL) &= ~TIMER_CTL_TBEN;
534     HWREG(WTIMER4_BASE+TIMER_O_TBILR) = OneShotTimeout_30ms;
535     __enable_irq();
536     HWREG(WTIMER4_BASE+TIMER_O_CTL) |= (TIMER_CTL_TBEN | TIMER_CTL_TBSTALL);
537 }
538
539 void OneShotIntResponse_Supply( void ){
540     // start by clearing the source of the interrupt
541     HWREG(WTIMER4_BASE+TIMER_O_ICR) = TIMER_ICR_TBTOCINT;
542     if(counter < 20){
543         if(counter % 2 == 0){
544             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, BIT0HI); //set IR LED High
545             counter++; // increment supply ir led counter
546             StartOneShot_Supply_10ms();
547         }
548         else{
549             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, BIT0LO); //set IR LED Low
550             StartOneShot_Supply_30ms();
551             counter++; // increment supply ir led counter
552         }
553     }
554     // if counter is 20, we increment ball( here we include fix for increasing ball twice)
555     if(counter == 20){
556         if(count_valid){
557             ES_Timer_InitTimer(SUPPLY_TIMER, THREE_SEC);
558             increment_num_ball();
559             count_valid = false;
560         }
561         else{
562             count_valid = true;
563         }
564     }
565 }
```