```c
/*
    Real-Time Systems
    (C) 2009 J. Friedrich
    University of Applied Sciences Esslingen

    Author:  J. Friedrich, April 2009
*/
#include "states.h"
#include "ticker.h"
#include "led.h"
#include "button.h"
#include "buttonh.h"
#include "hal.h"

static states currentState;
static states lastState;

typedef enum {NONE=0, YELLOWB2YELLOW=1, YELLOWB2OFF=2} transition;
static transition transitionActivity;

states getCurrentState() {
    return currentState;
}

void setCurrentState(states cs) {
    currentState = cs;
}

states getLastState() {
    return lastState;
}

void setLastState(states cs) {
    lastState = cs;
}


void initStateMachine() {
  setLastState(OFF); /* force initial transition */
  setCurrentState(YELLOWB); /* next after initial state */
  resetTimer();
}

/* State YELLOWB */
void YellowB() {
   /* three events are possible here:
      - eventOn
      - eventOff
      - after(1s)
    */
   if (getCurrentState() != getLastState()) {
     resetTimer();
     setLastState(getCurrentState());
     transitionActivity = NONE;
     YellowOn();  /* onEntry activity */
   }

   if (eventOn()) {   /* reaction on event "on" */
     /* here we could place a guard condition */
     /* here we could prepare for a transition activity */
     /* transitionActivity = YELLOWBTOYELLOW; */
     setCurrentState(YELLOW);         /* next state */
   }

   else if (eventOff()) {
                  /* reaction on event "off" */
     /* transitionActivity = YELLOWBTOYELLOW; */
     setCurrentState(OFF);      /* next state */
   }

   else if (getTimerValue() >= 1) {/* reaction of event "after(1s)" */
     setCurrentState(ALLOFF);  /* next state */
   }
```

```
   if (getCurrentState() != getLastState()) {
      /* Here we could place the onExit activity */
      /* Here we execute the transition activities, if any */
       switch (transitionActivity) {
           case NONE: break;
           case YELLOWB2YELLOW: break;
           default: break;
       }
   }
}

void AllOff() {
   if (getCurrentState() != getLastState()) {
     setLastState(getCurrentState());  /* eat up state transition */
     setAllOff();                 /* onEntry activity */
     resetTimer();
   }

   if(getTimerValue() >= 1) {
     setCurrentState(YELLOWB); /* set next state */
   }

   else if (eventOn()) {
     setCurrentState(YELLOW);  /* set next state */
   }

   else if (eventOff()) {
     setCurrentState(OFF);  /* set next state */
   }


}

void Red() {
   if (getCurrentState() != getLastState()) {
     setLastState(getCurrentState());  /* eat up state transition */
     RedOn();                 /* onEntry activity */
     resetTimer();
   }

   if(getTimerValue() >= 20) {
     setCurrentState(REDYELLOW); /* set next state */
   }

   if (eventOff()) {
     setCurrentState(YELLOWB);  /* set next state */
   }
}

void RedYellow() {
   if (getCurrentState() != getLastState()) {
     setLastState(getCurrentState());  /* eat up state transition */
     RedYellowOn();                 /* onEntry activity */
     resetTimer();
   }

   if(getTimerValue() >= 3) {
     setCurrentState(GREEN); /* set next state */
   }

   if (eventOff()) {
     setCurrentState(YELLOWB);  /* set next state */
   }
}


void Green() {
   if (getCurrentState() != getLastState()) {
     setLastState(getCurrentState());  /* eat up state transition */
     GreenOn();                 /* onEntry activity */
     resetTimer();
   }

   if(getTimerValue() >= 20) {
```

```
     setCurrentState(YELLOW); /* set next state */
   }

   if (eventOff()) {
     setCurrentState(YELLOWB);  /* set next state */
   }
}


void Yellow() {
   if (getCurrentState() != getLastState()) {
     setLastState(getCurrentState());  /* eat up state transition */
     YellowOn();                 /* onEntry activity */
     resetTimer();
   }

   if(getTimerValue() >= 3) {
     setCurrentState(RED); /* set next state */
   }

   if (eventOff()) {
     setCurrentState(YELLOWB);  /* set next state */
   }
}

void Off() {
    turnMachineOff();
}
```