EECE 218
Microcontrollers

FSM-based Design
Reactive systems

- Problem: How to code reactive systems on a microcontroller that has to
  - react to changes in inputs
  - react to time elapsed

- Solution:
  - Design reactions as FSM
  - Handle all inputs via subroutines that read/write I/O data
  - Code the FSM
    - Use a big ‘switch/case’ statement executed with a fixed frequency
    - Represent the state of the machine in variables
    - Use counter variables to represent time
A nuclear reactor shutdown system

The delayed reactor trip for the nuclear reactors used to be implemented in hardware using timers, comparators and logic gates as shown in Fig. 3. The new DRT system is implemented on microprocessors. Digital control systems provide cost savings and flexibility over the hardware implementation. However, the question now is whether the new microprocessor based software controller satisfies the same specifications as the old hardware implementation.

FIGURE 3. Analog implementation of the delay relay trip system DRT (the “controller”).
A nuclear reactor shutdown system

- **[R1]** When the power and pressure of the reactor exceed acceptable safety limits, the comparators which feed in to the first AND gate cause Timer1 to start, which times out after 3 seconds and sends a message to one of the inputs of the second AND gate indicating that the time-out has occurred. If after this first time-out the power is still greater than its safety limit, then the relay is tripped (opened), and Timer2 starts. The relay must remain open until Timer2 times out which happens after 2 seconds.

Requirement [R1] ensures that the relay is opened and remains open for two seconds thus shutting down the nuclear reactor in a timely fashion. If the controller fails to shutdown the reactor properly, then catastrophic results might follow including danger to life.
A nuclear reactor shutdown system

Conversely, each time the reactor is improperly shut down, the utility operating the reactor loses money because it must bring additional fossil fuel generating stations on line to meet demand. The next informal requirement states:

- [R2] If the power reaches an acceptable level then the relay should be closed as soon as possible (thus allowing the reactor to operate once more).

A final requirement that is implicit in the hardware specification, but must be explicitly stated for the software version is:

- [R3] The controller should never deadlock.

For example, if after the power and pressure have exceeded their critical values, and the system has waited 3 seconds to check the power level again, if the power is below its critical limit, then the system should reset and go back to monitoring its inputs (failure to do so would result in a deadlock).
Finite-state machine

PRH: Pressure High
POH: Power High
TSx: Timer X start
TCx: Timer X cancel
T\text{x}TOUT: Timer X timeout
RO/C: Relay open, close
Implementation

Software Timers

```c
struct TIMER {
    int run;
    int value;
    int limit;
} T1, T2;
```

```c
void init(struct TIMER *p, int limit) {
    p->run = 0; p->value = 0; p->limit = limit;
}
```

```c
void start(struct TIMER *p) { p->run=1; p->value = 0; }
void cancel(struct TIMER *p) { p->run=0; }
bool timeout(struct TIMER *p) { return (p->value > p->limit); }
void update(struct TIMER* p) { if(p->run) p->value++; }
```
Implementation

- **Variables**
  ```
  #define IDLE 0
  #define T1R  1
  #define T1TO 2
  #define T2R  3
  #define SHUT 4
  int state = IDLE;
  ```

  /* Initialization */
  ```
  ... init(&T1,300); /* 3 sec timer */
  init(&T2,200); /* 2 sec timer */
  ```
Implementation

- On every **10 msec** execute:

```c
Switch(state) {
    case IDLE: if (PRH() && POH()) { /* Check pressure/power */
        start(&T1); state = T1R;
    }
    break;
    case T1R: if (!PRH() || !POH()) { /* If normal ... */
        cancel(&T1); state = IDLE; break; /* go back to IDLE*/
    } else if(timeout(&T1)) { /* 3 sec timeout */
        cancel(&T1); state = T1TO; /* no break, goes to next case */
    } else break;
    case T1TO: if (POH()) { /* Check power, if high */
        start(&T2); RO(); state = T2R; /* Open relay, etc.*/
    } else { state = IDLE; } /* else go to IDLE */
    break;
}
```
Implementation

/* continued */
  case T2R: if (!POH()) { /* Check power, if not high... */
    cancel(&T2); RC(); state = IDLE; /* close relay, go IDLE*/
  } else {
    if(timeout(&T2)) {
      RC(); cancel(&T2); state = SHUT; /* close relay, go SHUT */
    }
  }
  break;
}
update(&T1); update(&T2); /* Update software timers - if they are running*/
/* - end of code */
FSM-based design

- Design FSM and write code accordingly
- States: values for the ‘state’ variable
  - State change: update to ‘state’ variable
- Main code is a switch/case statement
  - case STATE → check conditions, take action/s
- Implement software timers as counters
- Execute FSM code in a loop, scheduled periodically
  - UPDATE counters!