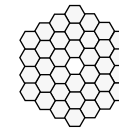


Contents

- Very brief overview of SNL (state notation language) and how it is used^{1 2}
- Description of changes that I have made
- Fuller description and example of new monitor queuing facilities
- Presentation of wish-list

-
1. Some material has been borrowed from Andy Kozubal and Bob Dalesio's SNC training slides
 2. More detail will be presented at SOSH talk on Tuesday morning



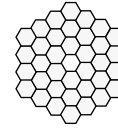
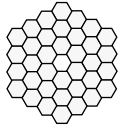
Overview

- SNL is a language designed specifically for translating state transition diagrams into C code:
 - ❑ C-like: state notation compiler (SNC) converts SNL source code to C, which is then compiled
 - ❑ multiple parallel diagrams can be implemented; event flags or variables allow communication
 - ❑ CA is directly integrated: channels appear as module-local variables
 - ❑ “just a CA client”; can *theoretically* run on IOC or under Unix
 - ❑ implemented by Andy Kozubal (LANL) to run under VxWorks (major upgrades with v1.9)
 - ❑ port to Unix (XMSEQ) by Ben-chin Cha (ANL) is no longer supported
- Typical uses:
 - ❑ coordination of subsystems (*e.g.* automated startup)
 - ❑ fault recovery (*e.g.* transition to safe state)
 - ❑ access to Unix file-system on IOC (*e.g.* save / restore)

- The standard example (light on above 5.0v and off below 4.5v):

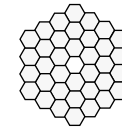
```
state light_off {  
    when ( v > 5.0 ) {  
        light = TRUE;  
        pvPut( light );  
    } state light_on  
}
```

```
state light_on {  
    when ( v < 4.5 ) {  
        light = FALSE;  
        pvPut( light );  
    } state light_off  
}
```



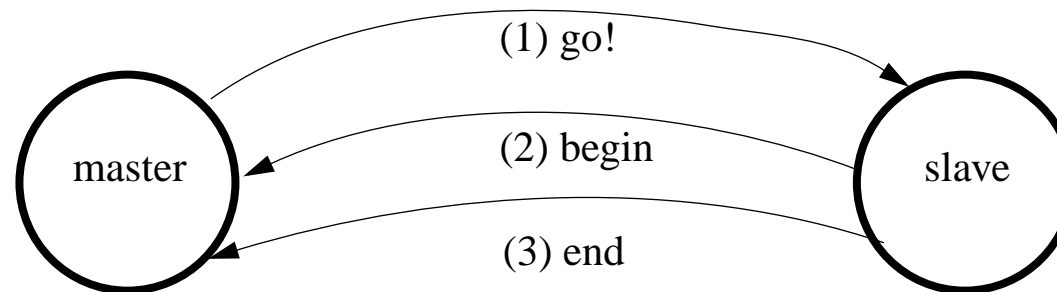
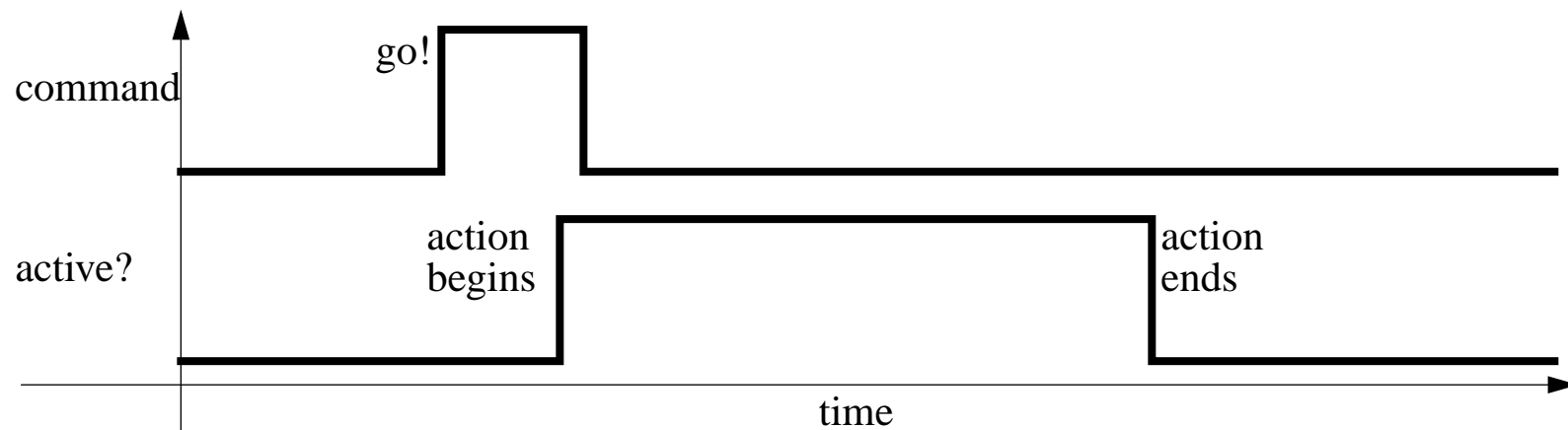
Changes that I have made

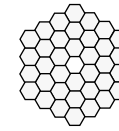
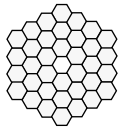
- Apology: recent changes have not been propagated back to ANL yet...
- Minor changes (not mentioning things that made it into the R3.13 version):
 - ❑ permitted ~ (bitwise negation) operator and ^L characters
 - ❑ allowed wakeup of state sets on event flag clear (as well as set)
 - ❑ removed assumptions that system clock is running at 60Hz
 - ❑ fixed variable names in some normally-disabled debug output
 - ❑ fixed race condition when monitoring run-time-assigned channel
- Major changes (all in support of queued monitors; rationale on next slide)
 - ❑ new `syncQ` statement for associating variable with event flag and monitor queue
 - ❑ new `pvGetQ()` procedure to get first entry from monitor queue
 - ❑ new `pvFreeQ()` procedure to empty a monitor queue
 - ❑ new `seqQueueShow()` procedure to list monitor queue details
 - ❑ fixed `seqChanShow()` not to claim that unassigned channels are disconnected



Queuing monitors: rationale (1)

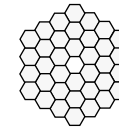
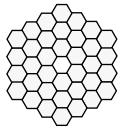
- At Keck, we use sequencers for controlling other sequencers; typically we want to send a command to another (independent) sequencer, change state, and then monitor an “active” flag for completion, expecting it to become TRUE and then FALSE





Queuing monitors: rationale (2)

- If the command completes immediately, the “begin” and “end” monitors will be delivered in rapid succession and the master sequencer may not wake up for the “begin” monitor until after the “end” monitor has been delivered. The “begin” monitor has effectively been lost.
- If this happens, the master sequencer will think that the slave never started to process the command and will typically time out.
- Accordingly, in July 1996 I proposed a means of (optionally) queueing monitors and picking them off one at a time in the sequencer code.
- I implemented the proposal and we have been using the new facilities without problems since September 1996.
- On the next slides, I give an annotated example of the use of the new facilities.
- Note that the example does not use arrays, but they are supported (we use dynamically assigned arrays to manage sets of subsystems). When an array is associated with a monitor queue, there is a single queue per array, not per element.



Queuing monitors: example (1)

```
program queueDemo
```

```
long command;  
assign command to "slave:command";
```

Define sequencer variables and associate them with channels

```
long active;  
assign active to "slave:active";  
monitor active;
```

Monitor the "active" channel

```
evflag activeFlag;  
syncQ active activeFlag 50;
```

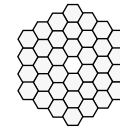
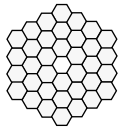
Define an event flag and create a monitor queue (length 50; defaults to 100) associating the event flag with active

```
ss main {  
    state start {  
        when () {  
            pvFreeQ( active );  
            command = TRUE;  
            pvPut( command );  
        } state sent  
    }  
}
```

Clear out any stale queue entries and send the command

```
    state sent {  
        when ( pvGetQ( active ) && active ) {  
            epicsPrintf( "-> active\n" );  
        } state active  
    }
```

pvGetQ() is quite like efTestAndClear(); it dequeues the oldest monitored value and copies it to active



Queuing monitors: example (2)

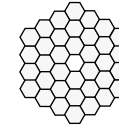
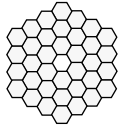
```
when( delay( 5.0 ) ) {  
    epicsPrintf( "active timeout\n" );  
} state done  
  
state active {  
    when ( pvGetQ( active ) && !active ) {  
        epicsPrintf( "-> done\n" );  
    } state done  
  
    when( delay( 5.0 ) ) {  
        epicsPrintf( "done timeout\n" );  
    } state done  
  
}  
  
state done {  
    when ( delay( 10.0 ) ) {  
        } state start  
    }  
}
```

Handle 5s timeout

“done” processing is analogous to
“active” processing

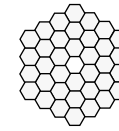
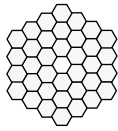
Handle 5s timeout

and again...



Wish list (1)

- This list is mostly based on one supplied to me by Andy Kozubal (several are from Ned Arnold of ANL). I have merged in some of my own wishes and some from Renaud Barillère of CERN.
- Minor additions:
 - ☐ Allow action statements to be executed on entry to and on exit from a state
 - ☐ Support channel access put with callback (`pvPutAck()`)
 - ☐ Support more of the C language (initialization, ternary operator and local variables would be nice!)
 - ☐ Allow an action not to reset timers
- Major additions:
 - ☐ Optionally permit several state sets to share a single task
 - ☐ Permit a “subroutine” state, callable as an action [*note: pre-processor macros provide a partial solution to this*]
 - ☐ Permit (parametrized) hierarchical states that can be referenced in several places [*note: also can be partially addressed by pre-processing*]
 - ☐ Provide hooks for external C code to set and test event flags (including from ISRs)
 - ☐ Provide a comprehensive test suite



Wish list (2)

- Longer-term additions:
 - ☐ Once again support the sequencer under Unix
 - ☐ Integrate with display managers to provide direct interaction with display elements
 - ☐ Allow sequencer dynamically to create internal CA database
 - ☐ Define SNL in terms of C++ objects (or Java?)
 - ☐ Documented handle on the C-code interface (????)
- It would be good to collect a fuller set and then get some community input on relative priorities.
- One problem: I don't have any time or money to work on this...