Adding Watchpoints and Actions

In class, we talked about autonomy in space systems being represented as an IF…..THEN condition. The way this is represented in the CFS flight software is with an application called the “Limit Checker” (LC). We will not be going into the full feature set of the LC app because we don’t need to in order to accomplish this practicum. So what follows is a useful but simplified version of how this all works.

In the LC app, the IF….THEN statement is formulated as:

IF Conditions THEN Actions

Where **Conditions** is a series of logical statements such as :

(Condition1 AND Condition2) OR (Condition 3)

And **Actions** is a command to run when the conditions evaluate to TRUE, such as:

SET_CAP_B_ACTIVE

So, to use the limit check, we need to:

1. Define the individual conditions (Condition1, Condition2, etc…)
    a. The LC calls these conditions “watchpoints”
    b. A watchpoint is a data value, a threshold, and how long the threshold has to be violated in order for the condition to be considered active.
    c. In watchpoint terminology, if the threshold is violated the watchpoint is considered to have “failed”
2. Define the IF…THEN statement that matches conditions and actions.
    a. The LC calls these “action points”
    b. There is a logic expression that incorporates watchpoints
    c. There is a command to run when the logical expression is active.

Why separate watchpoints and action points? Flexibility and avoiding redundancy. For example, we can easily define 3 action points using combinations of just 2 watch points:

W1: The condition where “capacitor A charge > 97%”

W2: The condition where “capacitor B charge > 97%”

We can then define 3 action points:

IF (W1) THEN (SET_CAP_B_ACTIVE)

IF (W2) THEN (SET_CAP_A_ACTIVE)

IF ((W1) AND (W2)) THEN (PANIC)

Adding Watchpoints

The LC App keeps watchpoints in a “table” file. The table is fully populated with entries (all but 1 of which are listed as “LC_WATCH_NOT_USED” and labeled #0, #1, #2, and so on). You will edit unused entries to make your own watchpoints.

I have added a simple watchpoint which says “When Capacitor A Charge >= 3” and added it as watchpoint #0.

To edit this watchpoint, and add your own, follow these instructions:

- Change directory to where LC keeps the watchpoint table
    - **cd ~/cfs/apps/lc/fsw/tables**
- Edit the watchpoint table
    - **gedit lc_def_wdt.c**
- Update items #0, #1, #2, etc… with the watchpoints that you want. With the following cavats:
    - .DataType will ALWAYS be **LC_DATA_BYTE**.
    - .OperatorID will be the mathematical operator to use (See Table 1)
        - In watchpoint #0 we use LC_OPER_GE which is the comparison “greater than or equal to” (>=).
    - .MessageID will ALWAYS be **WHE_HK_TLM_MID**
    - .WatchpointOffset will be the offset of the telemetry point (See Table 2)
        - In watchpoint #0, offset 14 is “Capacitor A charge”
    - .BitMask is ALWAYS **LC_NO_BITMASK**
    - .CustomFuncArgument is ALWAYS **0**
    - .ResultAgeWhenStale is ALWAYS **0**

  - .ComparisonValue.Unsigned32 is the value you want to compare to.
    - In watchpoint #0 this is 3 (meaning 3% charge)
- When you are finished editing, save your file and exit.

You now have defined some watchpoints in the software! Now, to go make action points!

Adding Action Points

The LC App keeps watchpoints in a “table” file. The table is fully populated with entries (all but 1 of which are listed as “LC_ACTION_NOT_USED” and labeled #0, #1, #2, and so on). You will edit unused entries to make your own action points.

I have added a simple actionpoint which says “discharge capacitor A if watchpoint 0 triggers”. Recall from above that we had set watchpoint 0 to trigger when capacitor A charge was > 3%. I used action point #1 (AP1) for this to show that there is no hard-coding of action points IDs and watch point IDs. AP1 can use WP0 and so on…

To edit my action point, and add your own, follow these instructions:

- Make sure you are still in the tables directory:
  - **cd ~/cfs/apps/lc/fsw/tables**
- Edit the actionpoint table
  - **gedit lc_def_adt.c**
- Update items #0, #1, #2, etc… with the actionpoints that you want. With the following cavats:
  - .DefaultState will ALWAYS be **LC_APSTATE_ACTIVE**.
  - .MaxPassiveEvents will ALWAYS be 0.
  - .MaxPassFailEvents will ALWAYS be 0.
  - .MaxFailPassEvents will ALWAYS be 0.
  - .RTSId will be the action to take (See Table 3)
    - For my example, the action is to discharge capacitor A so we use WHE_CAP_A_DISCHARGE_CC.
  - .MaxFailsBeforeRTS will be the number of seconds the conditions should persist before running the action. LC uses the term “fail” to mean “the RPN equation says to run the action”, which can be a bit confusing at first.
    - This MUST be a value > 0
    - For my action, I wanted the action to run the second it was detected, so I used “1”.

        - If you wanted to wait until the next second, you could use the value “2” and so on.
    - .EventType will ALWAYS be **CFE_EVS_INFORMATION**
    - .EventID should be 1000 + your action point ID
        - In my example, my action point is #1, so I used 1001.
    - .EventText is the text you want displayed to the user on the CFS screen.
        - In my example, I used “Discharge Capacitor A”
    - .RPNEquation is the RPN equation that describes the conditions to determine whether to run the action.
        - The last item MUST be LC_RPN_EQUAL.
        - If you simply want to run the action if a watchpoint triggers, you just put the number of the watchpoint.
            - For my example, I want to run the action if watch point 0 triggers, so I just put 0 there.
        - You can use combinations of watchpoints using logical operators (See Table 4).
        - It is possible that your rules NEVER need a complex expression and ALWAYS just take the action whenever the watchpoint triggers. If you feel you need something more complex, read the next section on RPN notation.
- When you are finished editing, save your file and exit.

Now you have action points!

Re-build the flight software!

Just like before, go to the correct directory and build the software.

1. Build the CFS code
    a. Change to the CFS directory: **cd ~/cfs**
    b. Set environment variables: **. ./setvar.sh**
    c. Change to the build directory: **cd build/cpu1**
    d. Configure the build: **make config**
    e. Build the OSAL, cFE, and CFS apps: **make**
    f. Wait while it all compiles. Look for “>>> DONE! <<<” at the end.
2. Make sure that CFS runs successfully on your platform.
    a. From the cpu1 directory (that you just built in) go to the exe folder: **cd exe**
    b. Run CFS: **sudo ./core-linux.bin –reset PO**
    c. Wait until you see the “Stop FLYWHEEL” message.
    d. Type CTRL-C to stop CFS and return to the command prompt.

Postfix (Reverse Polish) Notation

Everyone has probably seen Facebook quizzes relating to order-of-operations such as:

3+5*6 = ?

Now, we know that multiplication comes before addition, so the answer here is 33 (not 48). One way to clarify order-of-operations is to use parenthesis so as to not make mistakes:

3+(5*6) = ?

This notation is called INFIX notation because the operators (+, *) are "in order" with their operands. Doing this is great for people to read, but actually takes up a lot of memory and processing space on embedded computers. So, instead, we use POSTFIX notation (operators come AFTER their operands). So, we could write our original equation in postfix as:

5 6 * 3 + = ?

Aside from making us sound like Yoda when we say that out loud, POSTFIX gets rid of the need for parenthesis – order of operation is encoded in the expression now. The process of evaluating goes something like this:

Step 1: Grab 5, then grab 6, then grab the operator *

Step 2: Multiple 5 * 6 to get 30

Step 3: Replace 5 6 * with its result 30, so instead of "<u>5 6 *</u> 3 +" we have "30 3 +"

Step 4: Grab 30, then grab 3, then grab the operator +

Step 5: Add 30 and 3 to get 33

Step 6: Replace 30 3 + with its result 33, so instead of "30 3 +" we have 33

Step 7: If all we have is a single number left, that must be the answer. 33!

POSTFIX is usually called “reverse polish notation” or RPN. The Wikipedia page for “Reverse Polish Notation” is excellent if you want to learn more.

CFS uses POSTFIX (RPN) notation when evaluating watch points. In this case a watchpoint is considered “TRUE” if it has triggered and “FALSE” if it has not triggered. As simple TRUE/FALSE statement, action points are not adding and multiplying, they use logical operators such as AND, OR, XOR, NOT, and EQUALS.

So, let’s say we want an action to be run if watchpoint0 (WP0) triggers AND watchpoint 1(WP1) has not triggered OR whenever watchpoint 3 (WP3) triggers.

Using INFIX we would write:

(WP0 AND !(WP1)) OR WP3

In POSTFIX we would write:

WP1 ! WP0 AND WP3 OR

And in the ADT table watchpoints are represented by a simple number, and the expression must ALWAYS end with LC_RPN_EQUAL so, we would represent the above as:

1, LC_RPN_NOT, 0, LC_RPN_AND, 3, LC_RPN_OR, LC_RPN_EQUAL

(There are plenty of infix to postfix online converters to use if this gives you a headache).

Appendix A – Watch and Action Point Table Constants

Table 1 - LC Watchpoint Operators.

Operator	Value in Watch Table
<	LC_OPER_LT
<=	LC_OPER_LE
!=	LC_OPER_NE
==	LC_OPER_EQ
>=	LC_OPER_GE
>	LC_OPER_GT

Table 2 - WHE Telemetry points, offsets, and values.

Telemetry Data Point	OFFSET	Values
Capacitor A charge	14	0-255. Represents % of safe charge.
Capacitor A State	15	0: Charging. 1: Discharging. 2:Leaking. 3:Broken
Capacitor B Charge	16	0-255. Represents % of safe charge.
Capacitor B State	17	0: Charging. 1: Discharging. 2:Leaking. 3:Broken
SBC State	18	0: Off. 1: Powered. 2: Observing. 3:Error
TEMP as integer	19	0-255. Temperature as an integer (truncated) e.g., 15.2 will be reported at 15. 15.9 will also be reported at 15.
Louver State	20	0: Closed. 1: Open
Heater State	21	0: Off. 1: On
Active Capacitor	22	0: A. 1: B.
Damage State	23	0: None. 1: Minor. 2: Major

Table 3 - Actions for WHE.

WHE_NOOP_CC	Perform a no-op (useless, really)
WHE_CAP_A_ACTIVE_CC	Make capacitor A the active capacitor
WHE_CAP_A_DISCHARGE_CC	Discharge capacitor A
WHE_CAP_B_ACTIVE_CC	Make capacitor B the active capacitor
WHE_CAP_B_DISCHARGE_CC	Discharge capacitor B
WHE_OBS_START_CC	Start an observations
WHE_OBS_STOP_CC	Stop an observation

WHE_POWER_SBC_CC	Turn on power to the SBC.
WHE_THERM_HTR_OFF_CC	Turn off the heater
WHE_THERM_HTR_ON_CC	Turn on the heater
WHE_THERM_LOUVER_CLOSE_CC	Close the Louver
WHE_THERM_LOUVER_OPEN_CC	Open the Louver
WHE_TLM_RESET_CNTS_CC	Reset telemetry counts

Table 4 - Action Point RPN Equation Operators

LC_RPN_AND	Logical AND
LC_RPN_OR	Logical OR
LC_RPN_XOR	Logical Exclusive OR
LC_RPN_NOT	Negation
LC_RPN_EQUAL	Equality