SPECIFICATIONS GENERATED BY LLM ON **Cdplayermodemanager** AND VERIFICATION PROCESS USING CBMC

PROMPT

Generate logical specifications

in the fomat "(// Initial state conditions(<state_variable> == <INITIAL_STATE>)&&(<input/output conditions 1>)&&(<input/output conditions 2>)&&... // add as many as needed)->(// Final (expected)state<state_variable> == <FINAL_STATE>)"

INPUTS:

rtU.DiscEject: A boolean (represented as 0.0 or non-0.0 double in C) indicating if a disc eject request is made.

rtU.RadioReq: An enumeration (RadioRequestMode) representing the desired radio mode (e.g., OFF, CD, AM, FM).

rtU.DiscPresent: A boolean indicating if a disc is physically present in the player.

rtU.CdReq: An enumeration (CdPlayerRequestType) representing CD playback requests (e.g., STOP, FF, REW, PLAY).

OUTPUTS:

rtY.CurrentRadioMode: An enumeration (RadioRequestMode) indicating the currently active radio mode.

rtY.MechCmd: An enumeration (MechanicalCommandType) representing the mechanical command sent to the CD player (e.g., STOP, PLAY, REW, FF, EJECT).

STATE VARIABLES:

rtDW.is_c1_model: Represents the current active state of the top-level CdPlayerModeManager chart (e.g., IN_Eject, IN_ModeManager).

rtDW.is_ModeManager: Represents the current active state within the ModeManager part of the chart (e.g., IN_ON, IN_Standby).

rtDW.was_ModeManager: Stores the previous active state of is_ModeManager, used for history junction logic (e.g., IN_ON, IN_Standby).

rtDW.is_ON: Represents the current active state when ModeManager is IN_ON (e.g., IN_CDMode, IN_AMMode, IN_FMMode).

rtDW.was_ON: Stores the previous active state of is_ON, used for history junction logic (e.g., IN_CDMode, IN_AMMode, IN_FMMode).

rtDW.is_CDMode: Represents the current active state when ON is IN_CDMode (e.g., IN_Play, IN_Stop).

rtDW.is_Play: Represents the current active state when CDMode is IN_Play (e.g., IN_Normal, IN_FastForward, IN_Rew).

rtDW.RadioReq_start: Stores the value of rtU.RadioReq from the previous step, used to detect changes in RadioReq.

rtDW.is_active_c1_model: A flag indicating if the CdPlayerModeManager chart is active (1U for active, 0U for inactive).

OUTPUT :

1. (// Initial state conditions
   (rtDW.is_c1_model == IN_Eject)
)
   ->
   (// Final (expected)state
   (rtDW.is_c1_model == IN_ModeManager)
)

   EXPLANATION :

   This rule dictates that the **IN_Eject** state is momentary. In the execution cycle immediately after an eject is triggered, the system transitions back to the main **IN_ModeManager** to resume normal operation.

2. (// Initial state conditions
   (rtDW.is_c1_model != IN_Eject)
   &&
   (rtDW.is_active_c1_model != 0U)
   &&
   (rtU.DiscEject != 0.0)
   &&
   (rtDW.is_ModeManager == IN_ON)
)
   ->
   (// Final (expected)state
   (rtDW.is_ModeManager == IN_NO_ACTIVE_CHILD)

&&
(rtDW.is_c1_model == IN_Eject)
&&
(rtY.MechCmd == EJECT)
)

EXPLANATION :

If the system is active, not in the Eject state, and the DiscEject input is active while the ModeManager is ON, then the ModeManager will become inactive, the system will transition to the Eject state, and the MechCmd output will be EJECT.

3. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtU.RadioReq == OFF)
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_Standby)
&&
(rtDW.was_ModeManager == IN_Standby)
&&
(rtY.CurrentRadioMode == OFF)
&&
(rtY.MechCmd == STOP)
)

EXPLANATION:

If the system is active and not in the Eject state, DiscEject is inactive, and the ModeManager is ON with a RadioReq of OFF, then the ModeManager will go to Standby, its 'was_ModeManager' state will be Standby, the CurrentRadioMode will be OFF, and the MechCmd will be STOP.

4. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&

(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtU.RadioReq != OFF)
&&
(rtDW.RadioReq_start != rtU.RadioReq)
&&
(rtU.RadioReq == CD)
&&
(rtU.DiscPresent == true)
)
->
(// Final (expected)state
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.was_ON == IN_CDMode)
&&
(rtY.CurrentRadioMode == CD)
&&
(rtY.MechCmd == PLAY)
&&
(rtDW.is_CDMode == IN_Play)
&&
(rtDW.is_Play == IN_Normal)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, RadioReq is CD (and different from the starting RadioReq), and a disc is present, then the system will transition to CDMode (both current and previous states), CurrentRadioMode will be CD, MechCmd will be PLAY, and the CDMode will be in Play (Normal sub-state).

5. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&

(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtU.RadioReq != OFF)
&&
(rtDW.RadioReq_start != rtU.RadioReq)
&&
(rtU.RadioReq == CD)
&&
(rtU.DiscPresent == false)
)
->
(// Final (expected)state
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.was_ON == IN_CDMode)
&&
(rtY.CurrentRadioMode == CD)
&&
(rtY.MechCmd == STOP)
&&
(rtDW.is_CDMode == IN_Stop)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, RadioReq is CD (and different from the starting RadioReq), and no disc is present, then the system will transition to CDMode (both current and previous states), CurrentRadioMode will be CD, MechCmd will be STOP, and the CDMode will be in Stop.

6. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtU.RadioReq != OFF)
&&

(rtDW.RadioReq_start != rtU.RadioReq)
&&
(rtU.RadioReq == AM)
)
->
(// Final (expected)state
(rtDW.is_ON == IN_AMMode)
&&
(rtDW.was_ON == IN_AMMode)
&&
(rtY.CurrentRadioMode == AM)
&&
(rtY.MechCmd == STOP)
)

EXPLANATION :

If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, and RadioReq is AM (and different from the starting RadioReq), then the system will transition to AMMode (both current and previous states), CurrentRadioMode will be AM, and MechCmd will be STOP.

7. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtU.RadioReq != OFF)
&&
(rtDW.RadioReq_start != rtU.RadioReq)
&&
(rtU.RadioReq == FM)
)
->
(// Final (expected)state
(rtDW.is_ON == IN_FMMode)
&&
(rtDW.was_ON == IN_FMMode)
&&
(rtY.CurrentRadioMode == FM)

&&
(rtY.MechCmd == STOP)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, and RadioReq is FM (and different from the starting RadioReq), then the system will transition to FMMode (both current and previous states), CurrentRadioMode will be FM, and MechCmd will be STOP.

8. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.RadioReq_start == rtU.RadioReq)
&&
(rtDW.is_ON == IN_AMMode)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == AM)
)

EXPLANATION :

If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, the starting RadioReq is the same as the current RadioReq, and the ON state is AMMode, then the CurrentRadioMode will be AM.

9. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&

(rtDW.RadioReq_start == rtU.RadioReq)
&&
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.is_CDMode == IN_Play)
&&
(rtU.CdReq == STOP)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == CD)
&&
(rtDW.is_Play == IN_NO_ACTIVE_CHILD)
&&
(rtDW.is_CDMode == IN_Stop)
&&
(rtY.MechCmd == STOP)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, the starting RadioReq is the same as the current RadioReq, the ON state is CDMode, the CDMode is in Play, and the CdReq is STOP, then the CurrentRadioMode will be CD, the is_Play state will be inactive, the CDMode will be in Stop, and the MechCmd will be STOP.

10. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.RadioReq_start == rtU.RadioReq)
&&
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.is_CDMode == IN_Play)
&&
(rtU.CdReq == FF)
&&

(rtDW.is_Play != IN_FastForward)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == CD)
&&
(rtDW.is_Play == IN_FastForward)
&&
(rtY.MechCmd == FF)
)

EXPLANATION :

If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, the starting RadioReq is the same as the current RadioReq, the ON state is CDMode, the CDMode is in Play, the CdReq is FF, and the is_Play state is not already FastForward, then the CurrentRadioMode will be CD, the is_Play state will transition to FastForward, and the MechCmd will be FF.

11. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.RadioReq_start == rtU.RadioReq)
&&
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.is_CDMode == IN_Play)
&&
(rtU.CdReq == REW)
&&
(rtDW.is_Play != IN_Rew)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == CD)
&&

(rtDW.is_Play == IN_Rew)
&&
(rtY.MechCmd == REW)
)

EXPLANATION :

If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, the starting RadioReq is the same as the current RadioReq, the ON state is CDMode, the CDMode is in Play, the CdReq is REW, and the is_Play state is not already Rewind, then the CurrentRadioMode will be CD, the is_Play state will transition to Rewind, and the MechCmd will be REW.

12. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.RadioReq_start == rtU.RadioReq)
&&
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.is_CDMode == IN_Play)
&&
(rtU.CdReq == PLAY)
&&
(rtDW.is_Play != IN_Normal)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == CD)
&&
(rtDW.is_Play == IN_Normal)
&&
(rtY.MechCmd == PLAY)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, the starting RadioReq is the same as the current RadioReq, the ON state is CDMode, the CDMode is in Play, the CdReq is PLAY, and the is_Play state is not already Normal, then the CurrentRadioMode will be CD, the is_Play state will transition to Normal, and the MechCmd will be PLAY.

13. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.RadioReq_start == rtU.RadioReq)
&&
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.is_CDMode == IN_Stop)
&&
(rtU.DiscPresent == true)
&&
(rtU.CdReq == PLAY)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == CD)
&&
(rtDW.is_CDMode == IN_Play)
&&
(rtDW.is_Play == IN_Normal)
&&
(rtY.MechCmd == PLAY)
)

EXPLANATION :

If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, the starting RadioReq is the same as the current RadioReq, the ON state is CDMode, the CDMode is in Stop, a disc is present, and the CdReq is PLAY, then the CurrentRadioMode will be CD, the CDMode will transition to Play (Normal sub-state), and the MechCmd will be PLAY.

14. (// Initial state conditions
    (rtDW.is_c1_model != IN_Eject)
    &&
    (rtDW.is_active_c1_model != 0U)
    &&
    (rtU.DiscEject == 0.0)
    &&
    (rtDW.is_ModeManager == IN_ON)
    &&
    (rtDW.RadioReq_start == rtU.RadioReq)
    &&
    (rtDW.is_ON == IN_FMMode)
)
    ->
    (// Final (expected)state
    (rtY.CurrentRadioMode == FM)
)

    EXPLANATION:

    If the system is active and not in Eject, DiscEject is inactive, ModeManager is ON, the starting RadioReq is the same as the current RadioReq, and the ON state is FMMode, then the CurrentRadioMode will be FM.

15. (// Initial state conditions
    (rtDW.is_c1_model != IN_Eject)
    &&
    (rtDW.is_active_c1_model != 0U)
    &&
    (rtU.DiscEject == 0.0)
    &&
    (rtDW.is_ModeManager == IN_Standby)
    &&
    (rtU.RadioReq == CD)
    &&
    (rtU.DiscPresent == true)
)
    ->
    (// Final (expected)state
    (rtY.CurrentRadioMode == CD)
    &&
    (rtY.MechCmd == PLAY)
    &&

(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.was_ON == IN_CDMode)
&&
(rtDW.is_CDMode == IN_Play)
&&
(rtDW.is_Play == IN_Normal)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, ModeManager is in Standby, RadioReq is CD, and a disc is present, then the CurrentRadioMode will be CD, MechCmd will be PLAY, ModeManager will be ON (current and previous states), ON will be CDMode (current and previous states), and CDMode will be in Play (Normal sub-state).

16. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_Standby)
&&
(rtU.RadioReq == CD)
&&
(rtU.DiscPresent == false)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == CD)
&&
(rtY.MechCmd == STOP)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&

(rtDW.is_ON == IN_CDMode)
&&
(rtDW.was_ON == IN_CDMode)
&&
(rtDW.is_CDMode == IN_Stop)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, ModeManager is in Standby, RadioReq is CD, and no disc is present, then the CurrentRadioMode will be CD, MechCmd will be STOP, ModeManager will be ON (current and previous states), ON will be CDMode (current and previous states), and CDMode will be in Stop.

17. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_Standby)
&&
(rtU.RadioReq == AM)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == AM)
&&
(rtY.MechCmd == STOP)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_AMMode)
&&
(rtDW.was_ON == IN_AMMode)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, ModeManager is in Standby, and RadioReq is AM, then the CurrentRadioMode will be AM, MechCmd will be STOP, ModeManager will be ON (current and previous states), and ON will be AMMode (current and previous states).

18. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_Standby)
&&
(rtU.RadioReq == FM)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == FM)
&&
(rtY.MechCmd == STOP)
&&
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_FMMode)
&&
(rtDW.was_ON == IN_FMMode)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, ModeManager is in Standby, and RadioReq is FM, then the CurrentRadioMode will be FM, MechCmd will be STOP, ModeManager will be ON (current and previous states), and ON will be FMMode (current and previous states).

19. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&

(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager == IN_Standby)
&&
(rtU.RadioReq == OFF)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == OFF)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, ModeManager is in Standby, and RadioReq is OFF, then the CurrentRadioMode will be OFF.

20. (// Initial state conditions
(rtDW.is_c1_model != IN_Eject)
&&
(rtDW.is_active_c1_model != 0U)
&&
(rtU.DiscEject == 0.0)
&&
(rtDW.is_ModeManager != IN_ON)
&&
(rtDW.is_ModeManager != IN_Standby)
)
->
(// Final (expected)state
(rtY.CurrentRadioMode == OFF)
&&
(rtDW.is_ModeManager == IN_Standby)
&&
(rtDW.was_ModeManager == IN_Standby)
&&
(rtY.MechCmd == STOP)
)

EXPLANATION:

If the system is active and not in Eject, DiscEject is inactive, and ModeManager is neither ON nor Standby, then the CurrentRadioMode will be OFF, ModeManager will transition to Standby (current and previous states), and MechCmd will be STOP.

21. (// Initial state conditions
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_CDMode)
)
->
(// Final (expected)state
(rtY.MechCmd == STOP)
&&
(rtDW.is_Play == IN_NO_ACTIVE_CHILD)
&&
(rtDW.is_CDMode == IN_NO_ACTIVE_CHILD)
&&
(rtDW.is_ON == IN_NO_ACTIVE_CHILD)
)

EXPLANATION:

If the ModeManager is ON and the ON state is CDMode, then the MechCmd will be STOP, the is_Play state will be inactive, the is_CDMode state will be inactive, and the is_ON state will be inactive.

22. (// Initial state conditions
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.is_ON != IN_CDMode)
)
->
(// Final (expected)state
(rtDW.is_ON == IN_NO_ACTIVE_CHILD)
)

EXPLANATION:

If the ModeManager is ON and the ON state is not CDMode, then the is_ON state will be inactive.

23. (// Initial state conditions
(rtDW.was_ModeManager == IN_ON)

&&
(rtDW.was_ON == IN_AMMode)
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_AMMode)
&&
(rtDW.was_ON == IN_AMMode)
&&
(rtY.CurrentRadioMode == AM)
&&
(rtY.MechCmd == STOP)
)

EXPLANATION:

If the was_ModeManager was ON and was_ON was in AMMode, then the ModeManager will be ON (current and previous states), ON will be AMMode (current and previous states), CurrentRadioMode will be AM, and MechCmd will be STOP.

24. (// Initial state conditions
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.was_ON == IN_CDMode)
&&
(rtU.DiscPresent == true)
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.was_ON == IN_CDMode)
&&
(rtY.CurrentRadioMode == CD)
&&

(rtY.MechCmd == PLAY)
&&
(rtDW.is_CDMode == IN_Play)
&&
(rtDW.is_Play == IN_Normal)
)

EXPLANATION:

If the was_ModeManager was ON, the was_ON was in CDMode, and a disc is present, then the ModeManager will be ON (current and previous states), ON will be CDMode (current and previous states), CurrentRadioMode will be CD, MechCmd will be PLAY, and CDMode will be in Play (Normal sub-state).

25. (// Initial state conditions
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.was_ON == IN_CDMode)
&&
(rtU.DiscPresent == false)
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.was_ON == IN_CDMode)
&&
(rtY.CurrentRadioMode == CD)
&&
(rtY.MechCmd == STOP)
&&
(rtDW.is_CDMode == IN_Stop)
)

EXPLANATION:

If the was_ModeManager was ON, the was_ON was in CDMode, and no disc is present, then the ModeManager will be ON (current and previous states), ON will be CDMode (current and

previous states), CurrentRadioMode will be CD, MechCmd will be STOP, and CDMode will be in Stop.

26. (// Initial state conditions
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.was_ON == IN_FMMode)
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_FMMode)
&&
(rtDW.was_ON == IN_FMMode)
&&
(rtY.CurrentRadioMode == FM)
&&
(rtY.MechCmd == STOP)
)

EXPLANATION:

If the was_ModeManager was ON and was_ON was in FMMode, then the ModeManager will be ON (current and previous states), ON will be FMMode (current and previous states), CurrentRadioMode will be FM, and MechCmd will be STOP.

27. (// Initial state conditions
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.was_ON == IN_NO_ACTIVE_CHILD) // Default case for was_ON
&&
(rtU.RadioReq == CD)
&&
(rtU.DiscPresent == true)
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_ON)

&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.was_ON == IN_CDMode)
&&
(rtY.CurrentRadioMode == CD)
&&
(rtY.MechCmd == PLAY)
&&
(rtDW.is_CDMode == IN_Play)
&&
(rtDW.is_Play == IN_Normal)
)

EXPLANATION :

If the was_ModeManager was ON, the was_ON was inactive (default case), RadioReq is CD, and a disc is present, then the ModeManager will be ON (current and previous states), ON will be CDMode (current and previous states), CurrentRadioMode will be CD, MechCmd will be PLAY, and CDMode will be in Play (Normal sub-state).

28. (// Initial state conditions
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.was_ON == IN_NO_ACTIVE_CHILD) // Default case for was_ON
&&
(rtU.RadioReq == CD)
&&
(rtU.DiscPresent == false)
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_CDMode)
&&
(rtDW.was_ON == IN_CDMode)

&&
(rtY.CurrentRadioMode == CD)
&&
(rtY.MechCmd == STOP)
&&
(rtDW.is_CDMode == IN_Stop)
)

EXPLANATION:

If the was_ModeManager was ON, the was_ON was inactive (default case), RadioReq is CD, and no disc is present, then the ModeManager will be ON (current and previous states), ON will be CDMode (current and previous states), CurrentRadioMode will be CD, MechCmd will be STOP, and CDMode will be in Stop.

29. (// Initial state conditions
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.was_ON == IN_NO_ACTIVE_CHILD) // Default case for was_ON
&&
(rtU.RadioReq == AM)
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_AMMode)
&&
(rtDW.was_ON == IN_AMMode)
&&
(rtY.CurrentRadioMode == AM)
&&
(rtY.MechCmd == STOP)
)

EXPLANATION:

If the was_ModeManager was ON, the was_ON was inactive (default case), and RadioReq is AM, then the ModeManager will be ON (current and previous states), ON will be AMMode (current and previous states), CurrentRadioMode will be AM, and MechCmd will be STOP.

30. (// Initial state conditions
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.was_ON == IN_NO_ACTIVE_CHILD) // Default case for was_ON
&&
(rtU.RadioReq == FM)
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_ON)
&&
(rtDW.was_ModeManager == IN_ON)
&&
(rtDW.is_ON == IN_FMMode)
&&
(rtDW.was_ON == IN_FMMode)
&&
(rtY.CurrentRadioMode == FM)
&&
(rtY.MechCmd == STOP)
)

EXPLANATION :

If the was_ModeManager was ON, the was_ON was inactive (default case), and RadioReq is FM, then the ModeManager will be ON (current and previous states), ON will be FMMode (current and previous states), CurrentRadioMode will be FM, and MechCmd will be STOP.

31. (// Initial state conditions
(rtDW.was_ModeManager == IN_Standby)
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_Standby)
&&
(rtDW.was_ModeManager == IN_Standby)
&&

(rtY.CurrentRadioMode == OFF)
&&
(rtY.MechCmd == STOP)
)

EXPLANATION:

If the was_ModeManager was in Standby, then the ModeManager will be in Standby (current and previous states), CurrentRadioMode will be OFF, and MechCmd will be STOP.

32. (// Initial state conditions
(rtDW.was_ModeManager == IN_NO_ACTIVE_CHILD) // Default case for was_ModeManager
)
->
(// Final (expected)state
(rtDW.is_ModeManager == IN_Standby)
&&
(rtDW.was_ModeManager == IN_Standby)
&&
(rtY.CurrentRadioMode == OFF)
&&
(rtY.MechCmd == STOP)
)

EXPLANATION:

If the was_ModeManager was inactive (default case), then the ModeManager will be in Standby (current and previous states), CurrentRadioMode will be OFF, and MechCmd will be STOP.

33. (// Initial state conditions
(rtDW.is_active_c1_model == 0U)
)
->
(// Final (expected)state
(rtDW.is_active_c1_model == 1U)
&&
(rtDW.is_c1_model == IN_ModeManager)
)
EXPLANATION:

If the is_active_c1_model is inactive, then the is_active_c1_model will become active (1U), and the is_c1_model will transition to ModeManager.

VERIFICATION PROCESS:

I used a code that is a CBMC verification harness for a simulated CD player and radio system.It's designed to use the CBMC tool to formally verify a specific property of the system's behavior.

break down of the different sections of the verification_harness2.c

Overall Structure (Monolithic File)

The file you have now is a "monolithic" C file. This means that instead of having separate .h (header) and .c (source) files for different parts of the code (like rtwtypes.h, model.h, and model.c), all the necessary code has been combined into a single verification_harness2.c file. This helps avoid potential issues with include paths, hidden characters, or file encoding when compiling and verifying.

CBMC Harness Logic (main function)

This is the verification part, where CBMC comes into play:

- **extern int nondet_int(); and extern _Bool nondet_bool();:** These are special CBMC built-in functions. When CBMC encounters them, it understands that the values returned by these functions are **non-deterministic**. This means CBMC will explore *all possible integer or boolean values* for these inputs within the specified constraints.
- **Initializing the Model:** model_initialize() is called to put the system in a known starting state.

  Setting Initial State for Verification:

  ```
  rtDW.is_active_c1_model = 1U;
  rtDW.is_c1_model = IN_Eject;
  ```

- This is the **precondition** for your property. You are telling CBMC: "Assume that the model's internal state rtDW.is_c1_model is initially IN_Eject." This sets up the specific scenario you want to verify.

  Setting up Non-deterministic Inputs:

  ```
  rtU.DiscInsert = nondet_bool();
  rtU.DiscEject = nondet_bool();
  rtU.DiscPresent = nondet_bool();
  ```

```
temp_radio_req = nondet_int();
rtU.RadioReq = (RadioRequestMode)temp_radio_req;
temp_cd_req = nondet_int();
rtU.CdReq = (CdCommand)temp_cd_req;
```

- Here, we declare that all external inputs (DiscInsert, DiscEject, DiscPresent, RadioReq, CdReq) can take *any* valid value. CBMC will explore all combinations of these inputs.

  Constraining Non-deterministic Inputs (__CPROVER_assume()):

```
__CPROVER_assume(rtU.RadioReq >= OFF && rtU.RadioReq <= AM);
__CPROVER_assume(rtU.CdReq >= EMPTY && rtU.CdReq <= EJECT);
__CPROVER_assume(rtU.DiscInsert == 0 || rtU.DiscInsert == 1);
__CPROVER_assume(rtU.DiscEject == 0 || rtU.DiscEject == 1);
__CPROVER_assume(rtU.DiscPresent == 0 || rtU.DiscPresent == 1);
```

- These are crucial. __CPROVER_assume() tells CBMC to *only consider input values that satisfy these conditions*. Without them, nondet_int() could produce any integer value, which would lead to invalid enum values and potential verification issues or very long verification times. You are telling CBMC: "Assume these inputs are within their valid ranges."
- **Executing One Step of the Model:** model_step(); This executes the main logic of your system for one time step, taking into account the initial state set and all possible valid non-deterministic inputs.

  EXAMPLE : Specification 1: (// Initial state conditions
  (rtDW.is_c1_model == IN_Eject)
)
  ->
  (// Final (expected)state
  (rtDW.is_c1_model == IN_ModeManager)
)

  Verification Property (assert()):

  assert(rtDW.is_c1_model == IN_ModeManager);

- This is the **assertion** or **property** you want to verify. It states: "After one step of the model, if we started in IN_Eject state, the rtDW.is_c1_model *must* be IN_ModeManager." CBMC will try to find *any* combination of valid inputs and any path through the model_step function that would make this assertion false.
- **Verification Outcome:**
  - **VERIFICATION SUCCESSFUL**: This means CBMC could **not** find any scenario (any combination of valid inputs) where the assertion rtDW.is_c1_model ==

IN_ModeManager would be false, given that the initial state was IN_Eject. In other words, the property you defined holds true for all possible valid single-step executions from that initial IN_Eject state.

- If it had found a scenario where the assertion failed, it would provide a "counterexample trace" showing the exact input values and steps that led to the failure.

CONCLUSION :

used CBMC to formally prove that if your CD player/radio model is in an `Eject` state, it will always transition back to the main `ModeManager` state after one execution step, regardless of what other valid inputs (radio request, disc insertion, etc.) are applied.