6.5 Car Seat Controller

This additional VHDL example demonstrates how COSPAN's various complexity reduction routines can be utilized in concert in a verification methodology. The car seat has three axes of movement: the seat bottom can be raised or lowered, the seat back can be raised or lowered, and the seat can be moved closer to or further away from the steering wheel. Preferred settings on each axis can be stored and recalled for up to three people. The controller is responsible for producing two signals for each axis: a pulse to drive the motor on that axis and a signal indicating which direction the seat should move.

The main component of the architecture is the axis module, which is instantiated once for each of the three axes; the axis module architecture appears in Figure 3. We exploited this modular design style to take advantage of the symmetry-based reduction methods in COSPAN to reduce the verification effort. The inputs to the module are *up* and *dn* buttons for moving the seat while in manual mode, a toggle (*memsw*) indicating memory mode when on and manual mode when off, a system-wide *reset* signal, a *store* button for indicating that values should be saved to memory, a button for each person (*person1*, *person2*, *person3*) to control which registers to access on loads and stores, and the system clock *clk*. The module uses an eight-bit data-path and contains the following major components:

- The *Actual Position Counter* (APC) for storing the current seat position.
- The *Desired Position Counter* (DPC), a loadable counter containing the desired position of the seat when in manual mode.
- Three registers, for storing the preferred setting for each of three persons.
- A multiplexer (I4) that determines the target position (POS) depending upon whether the controller is in memory or manual mode; if the former, POS is also determined by the person whose settings are being recalled.
- A comparator that indicates the relationship between POS and the APC.
- A motor that produces the output pulses and the input signals to the APC, depending upon the output of the comparator.

The axis design is straightforward, with the possible exception of the wire for loading the DPC from the APC. When the controller is in manual mode, the APC is designed to follow the movements of the DPC. Once the controller loads a value from memory, the APC and the DPC can lose the alignment that makes this behavior possible. As a result, the DPC must be explicitly aligned with the APC when the controller switches from memory mode to manual mode.

The car seat environment model introduces the following constraints:

- The *up* and *dn* buttons for an axis may not be asserted simultaneously.
- Storage or recall may be selected for at most one person at a time.
- *Reset* remains high for a period of time (indicating initial reset) then remains low thereafter.

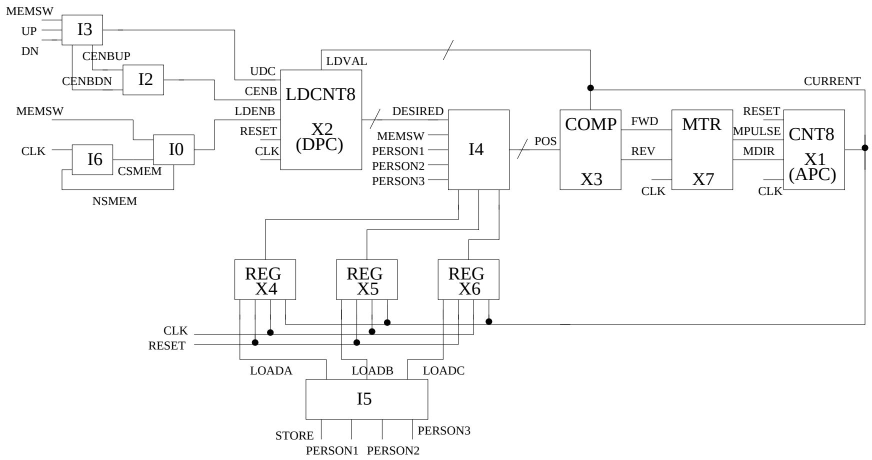

Figure 3: Architectural layout of the axis module. Slashed wires are eight bits wide.

- A period of time must be allowed for the controller to initially reset.
- The controller is in exactly one of manual mode or memory mode at all times.

Three copies of the axis module comprise the design. Since the axes do not communicate with one another, all properties of interest are relevant to the individual modules. We are interested in verifying the following properties of each axis:

1. Pushing the *up* button in manual mode causes the APC to increase.
2. Pushing the *dn* button in manual mode causes the APC to decrease.
3. Settings are properly stored to memory when store is activated.
4. Settings are properly recalled from memory when in memory mode.
5. The seat never attempts to move past the maximum and minimum positions.

Statements of the properties relative to the first axis appear below. Each of Property3 and Property4 needs to be duplicated for *person2* and *person3*. OldAPC and OldDPC store the respective values of the APC and DPC in the immediately preceding global state. Signals *fthtup* and *fthtdn* are the *up* and *dn* signals for the first axis. Signal $motorRot$ indicates that the motor should move; $motorDir$ indicates the direction in which it should move: 0 indicates backwards and 1 indicates forwards.

$$\text{Property1: After\_Eventually } (\neg reset * (\text{APC} < 255) * fthtup,\; memsw + (\text{APC} > \text{OldAPC})\ )$$