SoundWire Error Handling

The SoundWire PHY was designed with care and errors on the bus are going to be very unlikely, and if they happen it should be limited to single bit errors. Examples of this design can be found in the synchronization mechanism (sync loss after two errors) and short CRCs used for the Bulk Register Access.

The errors can be detected with multiple mechanisms:

1. Bus clash or parity errors: This mechanism relies on low-level detectors that are independent of the payload and usages, and they cover both control and audio data. The current implementation only logs such errors. Improvements could be invalidating an entire programming sequence and restarting from a known position. In the case of such errors outside of a control/command sequence, there is no concealment or recovery for audio data enabled by the SoundWire protocol, the location of the error will also impact its audibility (most-significant bits will be more impacted in PCM), and after a number of such errors are detected the bus might be reset. Note that bus clashes due to programming errors (two streams using the same bit slots) or electrical issues during the transmit/receive transition cannot be distinguished, although a recurring bus clash when audio is enabled is a indication of a bus allocation issue. The interrupt mechanism can also help identify Slaves which detected a Bus Clash or a Parity Error, but they may not be responsible for the errors so resetting them individually is not a viable recovery strategy.
2. Command status: Each command is associated with a status, which only covers transmission of the data between devices. The ACK status indicates that the command was received and will be executed by the end of the current frame. A NAK indicates that the command was in error and will not be applied. In case of a bad programming (command sent to non-existent Slave or to a non-implemented register) or electrical issue, no response signals the command was ignored. Some Master implementations allow for a command to be retransmitted several times. If the retransmission fails, backtracking and restarting the entire programming sequence might be a solution. Alternatively some implementations might directly issue a bus reset and re-enumerate all devices.
3. Timeouts: In a number of cases such as ChannelPrepare or ClockStopPrepare, the bus driver is supposed to poll a register field until it transitions to a NotFinished value of zero. The MIPI SoundWire spec 1.1 does not define timeouts but the MIPI SoundWire DisCo document adds recommendation on timeouts. If such configurations do not complete, the driver will return a -ETIMEOUT. Such timeouts are symptoms of a faulty Slave device and are likely impossible to recover from.

Errors during global reconfiguration sequences are extremely difficult to handle:

1. BankSwitch: An error during the last command issuing a BankSwitch is difficult to backtrack from. Retransmitting the Bank Switch command may be possible in a single segment setup, but this can lead to synchronization problems when enabling multiple bus segments (a command with side effects such as frame reconfiguration would be handled at different times). A global hard-reset might be the best solution.

Note that SoundWire does not provide a mechanism to detect illegal values written in valid registers. In a number of cases the standard even mentions that the Slave might behave in implementation-defined ways. The bus implementation does not provide a recovery mechanism for such errors, Slave or Master driver implementers are responsible for writing valid values in valid registers and implement additional range checking if needed.