CEC Pin Framework Error Injection

The CEC Pin Framework is a core CEC framework for CEC hardware that only has low-level support for the CEC bus. Most hardware today will have high-level CEC support where the hardware deals with driving the CEC bus, but some older devices aren't that fancy. However, this framework also allows you to connect the CEC pin to a GPIO on e.g. a Raspberry Pi and you have now made a CEC adapter.

What makes doing this so interesting is that since we have full control over the bus it is easy to support error injection. This is ideal to test how well CEC adapters can handle error conditions.

Currently only the cec-gpio driver (when the CEC line is directly connected to a pull-up GPIO line) and the AllWinner A10/A20 drm driver support this framework.

If `CONFIG_CEC_PIN_ERROR_INJ` is enabled, then error injection is available through debugfs. Specifically, in `/sys/kernel/debug/cec/cecX/` there is now an `error-inj` file.

> **Note**
>
> The error injection commands are not a stable ABI and may change in the future.

With `cat error-inj` you can see both the possible commands and the current error injection status:

```
$ cat /sys/kernel/debug/cec/cec0/error-inj
# Clear error injections:
#   clear          clear all rx and tx error injections
#   rx-clear       clear all rx error injections
#   tx-clear       clear all tx error injections
#   <op> clear     clear all rx and tx error injections for <op>
#   <op> rx-clear  clear all rx error injections for <op>
#   <op> tx-clear  clear all tx error injections for <op>
#
# RX error injection:
#   <op>[,<mode>] rx-nack              NACK the message instead of sending an ACK
#   <op>[,<mode>] rx-low-drive <bit>   force a low-drive condition at this bit position
#   <op>[,<mode>] rx-add-byte          add a spurious byte to the received CEC message
#   <op>[,<mode>] rx-remove-byte       remove the last byte from the received CEC message
#    any[,<mode>] rx-arb-lost [<poll>] generate a POLL message to trigger an arbitration lost
#
# TX error injection settings:
#   tx-ignore-nack-until-eom           ignore early NACKs until EOM
#   tx-custom-low-usecs <usecs>        define the 'low' time for the custom pulse
#   tx-custom-high-usecs <usecs>       define the 'high' time for the custom pulse
#   tx-custom-pulse                    transmit the custom pulse once the bus is idle
#
# TX error injection:
#   <op>[,<mode>] tx-no-eom            don't set the EOM bit
#   <op>[,<mode>] tx-early-eom         set the EOM bit one byte too soon
#   <op>[,<mode>] tx-add-bytes <num>   append <num> (1-255) spurious bytes to the message
#   <op>[,<mode>] tx-remove-byte       drop the last byte from the message
#   <op>[,<mode>] tx-short-bit <bit>   make this bit shorter than allowed
#   <op>[,<mode>] tx-long-bit <bit>    make this bit longer than allowed
#   <op>[,<mode>] tx-custom-bit <bit>  send the custom pulse instead of this bit
#   <op>[,<mode>] tx-short-start       send a start pulse that's too short
#   <op>[,<mode>] tx-long-start        send a start pulse that's too long
#   <op>[,<mode>] tx-custom-start      send the custom pulse instead of the start pulse
#   <op>[,<mode>] tx-last-bit <bit>    stop sending after this bit
#   <op>[,<mode>] tx-low-drive <bit>   force a low-drive condition at this bit position
#
# <op>       CEC message opcode (0-255) or 'any'
# <mode>     'once' (default), 'always', 'toggle' or 'off'
# <bit>      CEC message bit (0-159)
#            10 bits per 'byte': bits 0-7: data, bit 8: EOM, bit 9: ACK
# <poll>     CEC poll message used to test arbitration lost (0x00-0xff, default 0x0f)
# <usecs>    microseconds (0-10000000, default 1000)

clear
```

You can write error injection commands to `error-inj` using `echo 'cmd' >error-inj` or `cat cmd.txt >error-inj`. The `cat error-inj` output contains the current error commands. You can save the output to a file and use it as an input to `error-inj` later.

Basic Syntax

Leading spaces/tabs are ignored. If the next character is a `#` or the end of the line was reached, then the whole line is ignored. Otherwise a command is expected.

The error injection commands fall in two main groups: those relating to receiving CEC messages and those relating to transmitting CEC messages. In addition, there are commands to clear existing error injection commands and to create custom pulses on the CEC bus.

Most error injection commands can be executed for specific CEC opcodes or for all opcodes (`any`). Each command also has a 'mode' which can be `off` (can be used to turn off an existing error injection command), `once` (the default) which will trigger the error injection only once for the next received or transmitted message, `always` to always trigger the error injection and `toggle` to toggle the error injection on or off for every transmit or receive.

So '`any rx-nack`' will NACK the next received CEC message, '`any,always rx-nack`' will NACK all received CEC messages and '`0x82,toggle rx-nack`' will only NACK if an Active Source message was received and do that only for every other received message.

After an error was injected with mode `once` the error injection command is cleared automatically, so `once` is a one-time deal.

All combinations of `<op>` and error injection commands can co-exist. So this is fine:

```
0x9e tx-add-bytes 1
0x9e tx-early-eom
0x9f tx-add-bytes 2
any rx-nack
```

All four error injection commands will be active simultaneously.

However, if the same `<op>` and command combination is specified, but with different arguments:

```
0x9e tx-add-bytes 1
0x9e tx-add-bytes 2
```

Then the second will overwrite the first.

Clear Error Injections

`clear`
: Clear all error injections.

`rx-clear`
: Clear all receive error injections

`tx-clear`
: Clear all transmit error injections

`<op> clear`
: Clear all error injections for the given opcode.

`<op> rx-clear`
: Clear all receive error injections for the given opcode.

`<op> tx-clear`
: Clear all transmit error injections for the given opcode.

Receive Messages

`<op>[,<mode>] rx-nack`

: NACK broadcast messages and messages directed to this CEC adapter. Every byte of the message will be NACKed in case the transmitter keeps transmitting after the first byte was NACKed.

`<op>[,<mode>] rx-low-drive <bit>`

: Force a Low Drive condition at this bit position. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. This tests if the transmitter can handle the Low Drive condition correctly and reports the error correctly. Note that a Low Drive in the first 4 bits can also be interpreted as an Arbitration Lost condition by the transmitter. This is implementation dependent.

`<op>[,<mode>] rx-add-byte`

: Add a spurious 0x55 byte to the received CEC message, provided the message was 15 bytes long or less. This is useful to test the high-level protocol since spurious bytes should be ignored.

`<op>[,<mode>] rx-remove-byte`

: Remove the last byte from the received CEC message, provided it was at least 2 bytes long. This is useful to test the high-level protocol since messages that are too short should be ignored.

`<op>[,<mode>] rx-arb-lost <poll>`

: Generate a POLL message to trigger an Arbitration Lost condition. This command is only allowed for `<op>` values of `next` or `all`. As soon as a start bit has been received the CEC adapter will switch to transmit mode and it will transmit a POLL message. By default this is 0x0f, but it can also be specified explicitly via the `<poll>` argument.

This command can be used to test the Arbitration Lost condition in the remote CEC transmitter. Arbitration happens when two CEC adapters start sending a message at the same time. In that case the initiator with the most leading zeroes wins and the other transmitter has to stop transmitting ('Arbitration Lost'). This is very hard to test, except by using this error injection command.

This does not work if the remote CEC transmitter has logical address 0 ('TV') since that will always win.

Transmit Messages

`tx-ignore-nack-until-eom`

This setting changes the behavior of transmitting CEC messages. Normally as soon as the receiver NACKs a byte the transmit will stop, but the specification also allows that the full message is transmitted and only at the end will the transmitter look at the ACK bit. This is not recommended behavior since there is no point in keeping the CEC bus busy for longer than is strictly needed. Especially given how slow the bus is.

This setting can be used to test how well a receiver deals with transmitters that ignore NACKs until the very end of the message.

`<op>[,<mode>] tx-no-eom`

Don't set the EOM bit. Normally the last byte of the message has the EOM (End-Of-Message) bit set. With this command the transmit will just stop without ever sending an EOM. This can be used to test how a receiver handles this case. Normally receivers have a time-out after which they will go back to the Idle state.

`<op>[,<mode>] tx-early-eom`

Set the EOM bit one byte too soon. This obviously only works for messages of two bytes or more. The EOM bit will be set for the second-to-last byte and not for the final byte. The receiver should ignore the last byte in this case. Since the resulting message is likely to be too short for this same reason the whole message is typically ignored. The receiver should be in Idle state after the last byte was transmitted.

`<op>[,<mode>] tx-add-bytes <num>`

Append `<num>` (1-255) spurious bytes to the message. The extra bytes have the value of the byte position in the message. So if you transmit a two byte message (e.g. a Get CEC Version message) and add 2 bytes, then the full message received by the remote CEC adapter is `0x40 0x9f 0x02 0x03`.

This command can be used to test buffer overflows in the receiver. E.g. what does it do when it receives more than the maximum message size of 16 bytes.

`<op>[,<mode>] tx-remove-byte`

Drop the last byte from the message, provided the message is at least two bytes long. The receiver should ignore messages that are too short.

`<op>[,<mode>] tx-short-bit <bit>`

Make this bit period shorter than allowed. The bit position cannot be an Ack bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. Normally the period of a data bit is between 2.05 and 2.75 milliseconds. With this command the period of this bit is 1.8 milliseconds, this is done by reducing the time the CEC bus is high. This bit period is less than is allowed and the receiver should respond with a Low Drive condition.

This command is ignored for 0 bits in bit positions 0 to 3. This is because the receiver also looks for an Arbitration Lost condition in those first four bits and it is undefined what will happen if it sees a too-short 0 bit.

`<op>[,<mode>] tx-long-bit <bit>`

Make this bit period longer than is valid. The bit position cannot be an Ack bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. Normally the period of a data bit is between 2.05 and 2.75 milliseconds. With this command the period of this bit is 2.9 milliseconds, this is done by increasing the time the CEC bus is high.

Even though this bit period is longer than is valid it is undefined what a receiver will do. It might just accept it, or it might time out and return to Idle state. Unfortunately the CEC specification is silent about this.

This command is ignored for 0 bits in bit positions 0 to 3. This is because the receiver also looks for an Arbitration Lost condition in those first four bits and it is undefined what will happen if it sees a too-long 0 bit.

`<op>[,<mode>] tx-short-start`

Make this start bit period shorter than allowed. Normally the period of a start bit is between 4.3 and 4.7 milliseconds. With this command the period of the start bit is 4.1 milliseconds, this is done by reducing the time the CEC bus is high. This start bit period is less than is allowed and the receiver should return to Idle state when this is detected.

`<op>[,<mode>] tx-long-start`

Make this start bit period longer than is valid. Normally the period of a start bit is between 4.3 and 4.7 milliseconds. With

this command the period of the start bit is 5 milliseconds, this is done by increasing the time the CEC bus is high. This start bit period is more than is valid and the receiver should return to Idle state when this is detected.

Even though this start bit period is longer than is valid it is undefined what a receiver will do. It might just accept it, or it might time out and return to Idle state. Unfortunately the CEC specification is silent about this.

`<op>[,<mode>] tx-last-bit <bit>`

Just stop transmitting after this bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. This command can be used to test how the receiver reacts when a message just suddenly stops. It should time out and go back to Idle state.

`<op>[,<mode>] tx-low-drive <bit>`

Force a Low Drive condition at this bit position. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. This can be used to test how the receiver handles Low Drive conditions. Note that if this happens at bit positions 0-3 the receiver can interpret this as an Arbitration Lost condition. This is implementation dependent.

Custom Pulses

`tx-custom-low-usecs <usecs>`

This defines the duration in microseconds that the custom pulse pulls the CEC line low. The default is 1000 microseconds.

`tx-custom-high-usecs <usecs>`

This defines the duration in microseconds that the custom pulse keeps the CEC line high (unless another CEC adapter pulls it low in that time). The default is 1000 microseconds. The total period of the custom pulse is `tx-custom-low-usecs + tx-custom-high-usecs`.

`<op>[,<mode>] tx-custom-bit <bit>`

Send the custom bit instead of a regular data bit. The bit position cannot be an Ack bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet.

`<op>[,<mode>] tx-custom-start`

Send the custom bit instead of a regular start bit.

`tx-custom-pulse`

Transmit a single custom pulse as soon as the CEC bus is idle.