# Q-Channel Functional Safety Support for Ncore 3.2/3.4

## Table of Contents


***Q-Channel Functional Safety Support for Ncore 3.2/3.4 ..... 1***

**Q-Channel protection ..... 2**

Handshake mechanism ..... 2

Q-Channel acceptance protocol ..... 3

Q-Channel denial protocol ..... 3

Check bit timing ..... 3

**Filter Block Implementation** ..... 4


## Table of Figures


Figure 1: Generic Q-Channel Interface ..... 2

Figure 2: Q-Channel Interface with Check Signals ..... 2

Figure 3: Connection Ncore to Q-Channel Controller ..... 3

Figure 4: Transient fault on QREQn - QREQn_out is not asserted, no Error reported ..... 4

Figure 5: Transient fault on QREQn_chk - QREQn_out is not asserted, no Error reported ..... 4

Figure 6: Stuck-at-fault on QREQn_chk - QREQn_out is not asserted, Error asserted ..... 4

Figure 7: Stuck-at-fault on QREQn - QREQn_out is not asserted, Error asserted ..... 4

## Q-Channel protection

Q-Channel is an evolution of the AXI low-power interface and is backward-compatible in most situations. Q-Channel simplifies clock domain crossing by making the handshake mechanism independent of the device activity indication.

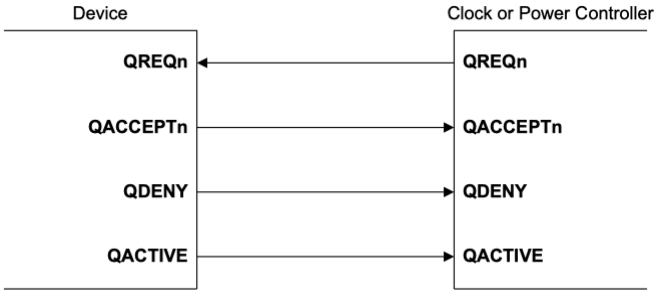



*Figure 1: Generic Q-Channel Interface*

Figure x shows a high level block diagram of a Q-channel interface  between the system controller and the local power-/clock controller within a block.

## Handshake mechanism

This group manages device quiescence and guarantees safe state transitions. The handshake interface has:

- A quiescence request signal, **QREQn**, driven by the controller.
- An acknowledgement signal pair, **QACCEPTn** and **QDENY**, which are driven back to the controller by the device to indicate acceptance or denial of a request. The acknowledgement signals are organized such that only one of them changes per handshake transition. This ensures that the interface can be implemented safely across asynchronous boundaries. **QACCEPTn** is used to accept a request. **QDENY** is used to deny a request.

The **QACCEPTn** and **QDENY** signals from a device and the **QREQn** signal from a controller must all be driven by registers.
The denial mechanism means a device can maintain an operational state while having a mechanism by which it can promptly complete the handshake of a quiescence request.

The polarities of the handshake signals have been chosen to provide a quiescent state where all interface signals are LOW. This facilitates simple default isolation rules.
The handshake signal states are independent of the state of **QACTIVE**. Therefore, transitions on **QACTIVE** are not restricted by the values on **QREQn** or on the **QACCEPTn** and **QDENY** output pair.
The controller can guarantee clock supply or power availability according to the handshake interface state. *Q-Channel handshake* describes these guarantees.

Q-Channel is asynchronous by nature and uses a 4-phase handshake to exchange information between initiator and target. The Q-Channel shall be protected by asynchronous parity. This method has been implemented in all Arm AE IP and Ncore shall use a similar approach.

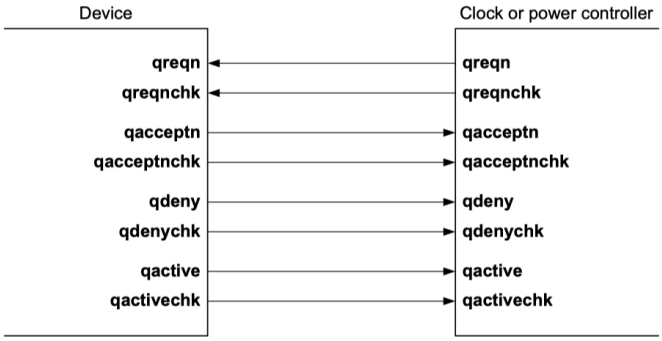



*Figure 2: Q-Channel Interface with Check Signals*

As every control signal and its complement passes the clock domain crossing through a separate synchronizer, both signals, even though launched from the same source clock, may not arrive during the same clock cycle in the destination's domain. To suppress spurious activities caused by this transient behavior, a (redundant) filter block (FBlock) shall be implemented.

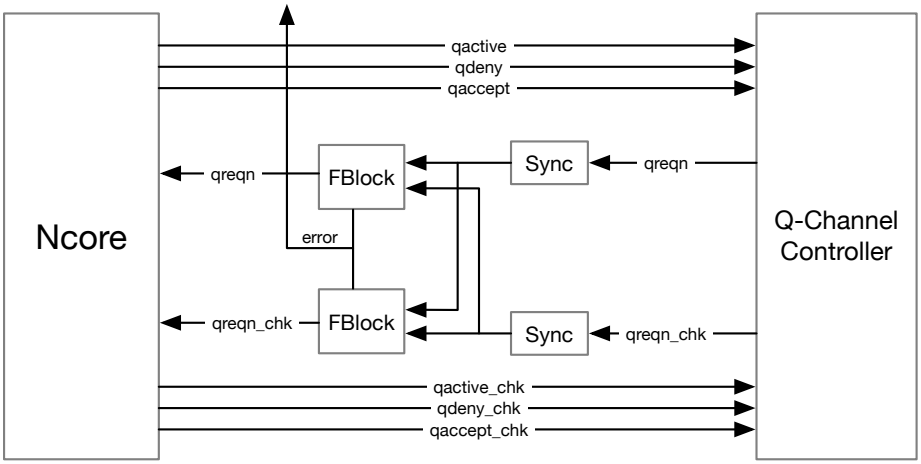
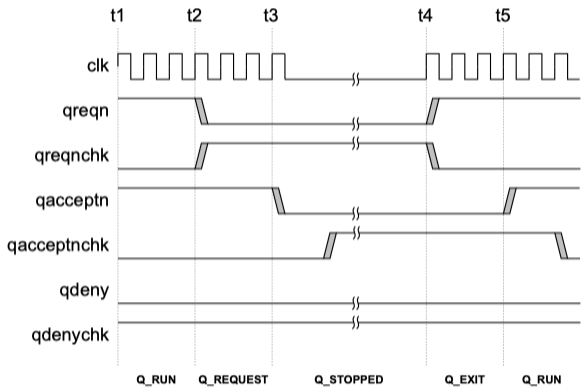


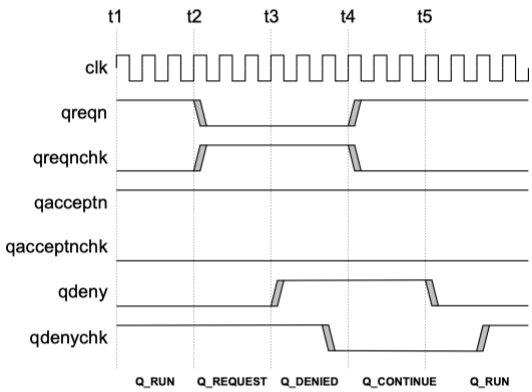

*Figure 3: Connection Ncore to Q-Channel Controller*

Having a 4-phase handshake requires to watch out for transients on both edges of QREQn and ensure that QREQn_chk properly tracks QREQn. The Q-Channel controller must implement a similar synchronizer & filter structure for each of the 3 response signals.
Ncore needs to implement a redundant set of clock controllers, each responding with a set of response signals (TBD).

## Q-Channel acceptance protocol




## Q-Channel denial protocol




## Check bit timing

The clocks on both sides of the CDC may run at different frequencies - the width of the stuck-at-counter within the filter determines the maximum allowable skew between QREQn and QREQn_chk. The following parameters define the number of bits:

The clock ratio (CR):

$F_{\text{Ncore core clock}}/F_{\text{Q-Channel controller}}$

Asynchronous skew:

This is the skew component contributed by signal propagation and the asynchronous sampling. This is usually +/- one clock cycle.

Temporal Delay Skew:

The skew between duplicated logic paths (functional vs. redundant). This is 1 or 2 clock cycles of the Ncore clock.

## Filter Block Implementation

The filter block receives the synchronized versions of QREQn and QREQn_chk - there exist 4 possible fault scenarios:

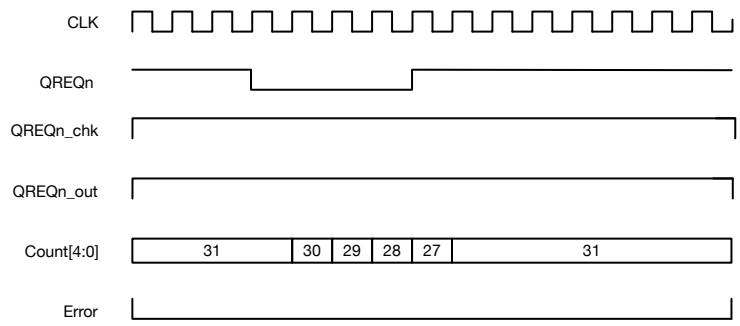



*Figure 4: Transient fault on QREQn - QREQn_out is not asserted, no Error reported*

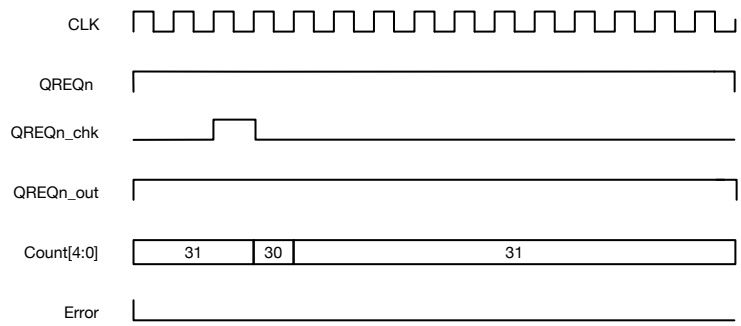



*Figure 5: Transient fault on QREQn_chk - QREQn_out is not asserted, no Error reported*

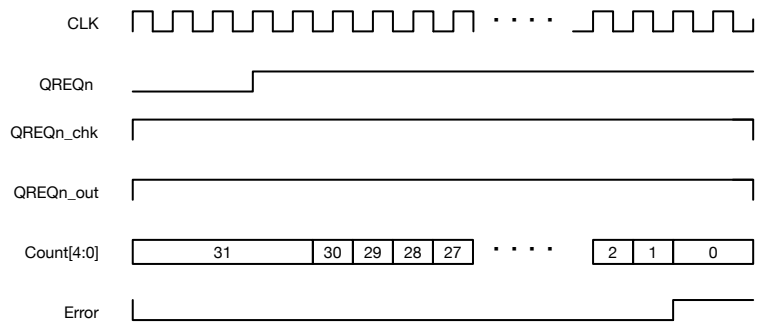



*Figure 6: Stuck-at-fault on QREQn_chk - QREQn_out is not asserted, Error asserted*

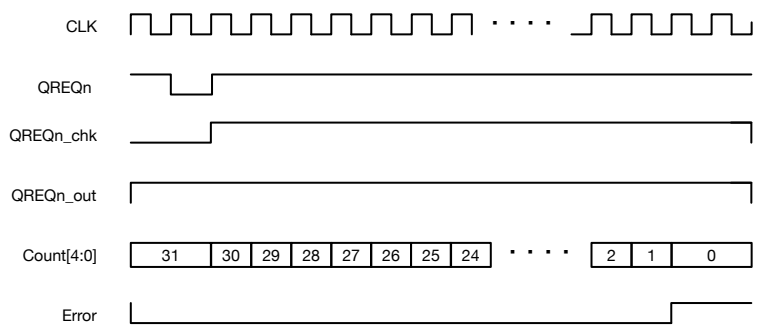



*Figure 7: Stuck-at-fault on QREQn - QREQn_out is not asserted, Error asserted*

The filter implements a stuck-at-counter with 5 bits - any disparity of *QREQn* or *QREQn_chk* that lasts longer than 31 clock cycles is considered a stuck-at-fault and activates the *Error* output.
A simple state machine shall track the state of *QREQn* and generate an output signal QREQn_out.

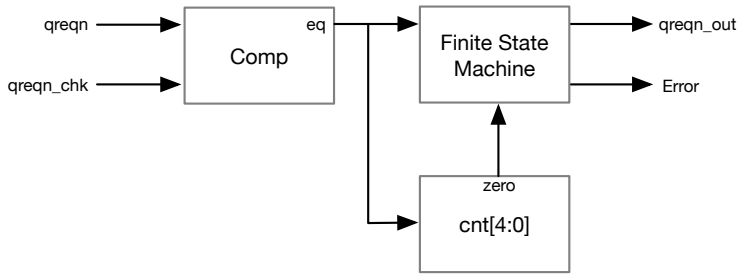



*Figure 8: Filter Block architecture*

The counter shall be preset to count = 0x1F while *QREQn != QREQn_chk*
The counter shall count backwards while *QREQn == QREQn_chk*
The counter shall stop at count == 0x00 and assert *Error*
*Error* shall be reset by *RESET* or by *QREQn_out* deassertion

In normal operation it is expected (and checked) that *QREQn* and *QREQn*_check follow each other within a limited number of clock cycles:

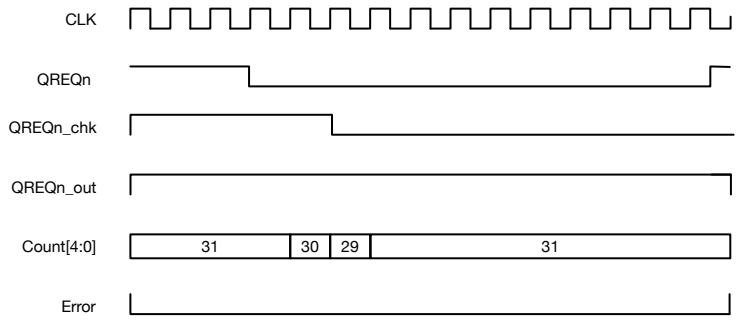



*Figure 9: Normal functionality - QREQn and QREQn_chk track each other*

The output *QREQn_out* of each filter block drives the *QREQn/QREQn_chk* for the associated clock controller. For clock activation, the clock controller must receive active levels on both signals.