TCP3e Driver Software Design Specification

100_PDK_C6497-C6498
January 04, 2010
QRSA00013338 Rev A
Version 01.00.00.02

20450 Century Boulevard
Germantown, Maryland 20874 USA
Voice: 301.515.8580
Fax: 301.515.7687
Web: http://www.ti.com/voip

Copyright and Contact Information

Document Copyright

© 1998-2008 Texas Instruments Incorporated
All Rights Reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Software Copyright

Product Name: TCP3e Driver
Product Release: 12 +
© 1998-2008 Texas Instruments Incorporated
All Rights Reserved.

Contact Information

20450 Century Boulevard
Germantown, MD 20874
Voice: 301.515.8580
Fax: 301.515.7687
Web: www.ti.com

NOTICE OF PROPRIETARY INFORMATION

Information contained herein is of a highly sensitive nature and is confidential and proprietary to Texas Instruments Inc. It shall not be distributed, reproduced or disclosed orally or in written form, in whole or in part, to any party other than the direct recipients without the express written consent of Texas Instruments Inc.

Document Revision History

Version	Date	Editor(s)	Notes
1.0.0.0 (A)	October 13, 2009	Slobodan Jovanovic	Initial version of document created.
1.0.0.1	November 30, 2009	Ravi Sankar Korada	Updated document version number

Confidential and Proprietary

References

No	Referenced Document	Control Number	Description
1	Nyquist SAS		Software Architecture Specification
2	TCP3 Encoder (TCP3e) Module Specification	Version 0.1.10	TCP3e Micro Architecture Specification

Confidential and Proprietary

Table of Contents

1 Scope

This document describes the design of the TCP3 Encoder driver.

2 Driver Design

2.1 Encoding Process

The data transfer between DSP and TCP3e is performed via EDMA transfers. The TCP3e has two synchronization events, WEVT and REVT, connected to EDMA channel controller as shown in Figure 1. DSP either sends one set of configuration parameters per one code block, or one set of configuration parameters per multiple code blocks of the same size and the same offset in DSP memory. The configuration parameters are loaded to three TCP3e input configuration registers. The encoding process requires three EDMA PaRAM entries for transfers of the configuration parameters, information bits and encoded bits. The DSP initiates process by setting the bit corresponding to EDMA channel Wr in the EDMA controller event set register. The EDMA transfers the input configuration registers to TCP3e and links in the PaRAM entry that describes the input data (info bits). The TCP3e issues WEVT to initiate the input data transfer. After the transfer is completed, the TCP3e encodes data and issues REVT on channel Rd, to initiate encoded data transfer back to DSP. After the transfer is completed, the EDMA controller issues an interrupt to DSP to signal the end of the encoding process. This process is repeated for every code block.

Multiple code blocks of the same size and the same offset in the memory can be encoded by sending one single set of input configuration parameters, where one filed, MCOUNT, corresponds to the number of input code blocks. This group of code blocks can also be encoded using three EDMA PaRAM entries, where two entries for information bits and encoded bits transfers are used far all MCOUNT blocks. The encoding process for MCOUNT=4 is illustrated in Figure 2. In this case TCP3e will send 4 times WEVT to the EDMA controller to initiate transfer of input code blocks from DSP to TCP3e, and 4 times REVT to the EDMA controller to initiate transfer of encoded code blocks from TCP3e to DSP. Note that TCP3e loads alternatingly ping and pong encoding engine, but for the EDMA controller its input and output memory addresses are same for both ping and pong engine. This is useful for encoding transport blocks that are segmented into several code blocks.

When a group of transport blocks of different sizes has to be encoded, the EDMA can be programmed to process the whole group using EDMA chaining. The EDMA PaRAM entries, three per transport block are programmed and the whole process is initiated by DSP, and when it is completed the EDMA generates interrupt to DSP to notify it that process is completed. If the encoded blocks are processed by multiple DSP cores the destination cores may need to be notified as soon as the block is encoded before the whole group is completed. One approach to solve this requirement is to use a hardware queue for the notification mechanism, and add one more EDMA PaRAM entry per code block which will be chained to the encoded bits transfer. It will push the transport block descriptor to the destination queue.

The EDMA chaining can be performed in different ways and the three approaches implemented in the current driver design are described in the following section.

Confidential and Proprietary

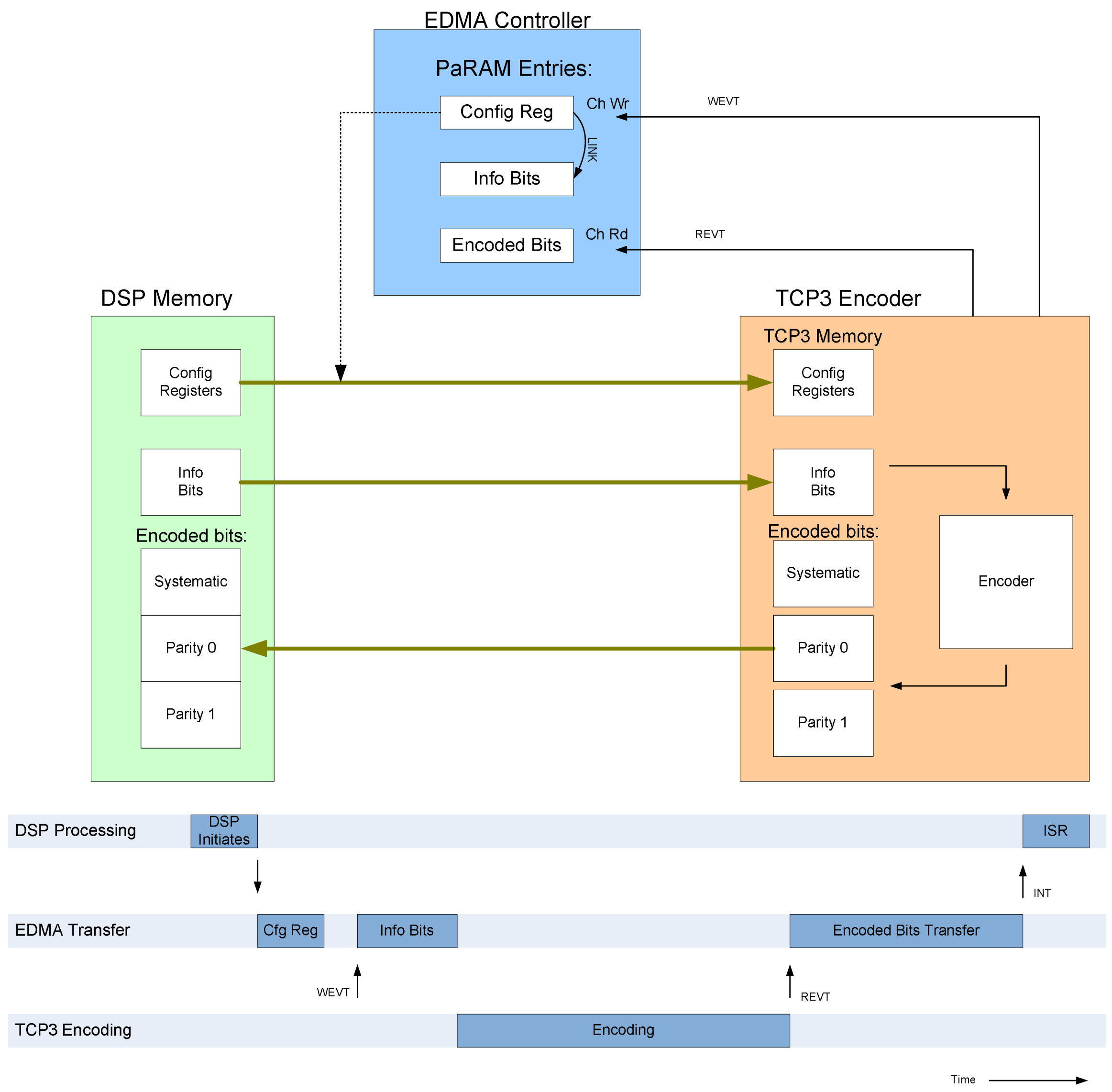

Figure 1 - Encoding process for single code block

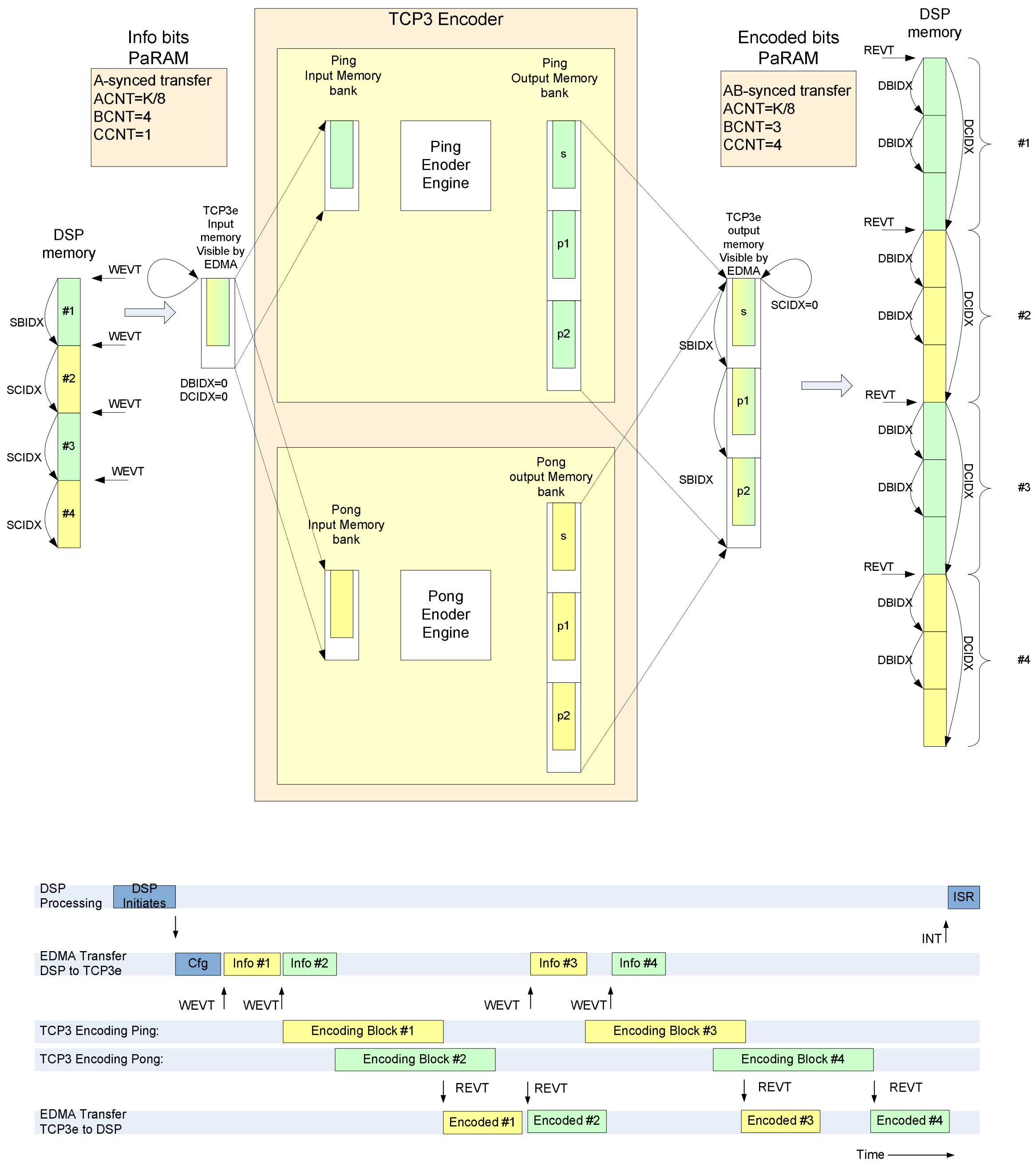

Figure 2 - Encoding process of multiple blocks of the same size

2.2 Code block chaining

Three approaches are currently supported in the driver, and can be selected at the initialization time:

a) Approach using sync buffer - The encoder engines are maximally loaded, i.e. CB's are maximally pipelined.

b) Pair-wise approach - It is simpler, does not require sync buffer, requires less physical EDMA channels. TB's with MCOUNT=1 are pipelined within pairs. CB's within TB are pipelined.

c) Approach without sync buffer - Does not use sync buffer. It pipelines TB's with MCOUNT=1. TB's with MCOUNT>1 are not pipelined. CB's within TB are pipelined.

2.2.1 Chaining with sync buffer

In this approach the two TCP3e encoding engines are maximally loaded. Normally, the loading of the new transport block to one of the encoding engines can start after the completion of the encoded bits transfer of the previous block from the same engine, but not before the completion of the loading of the previous block to the other engine. In other words it is triggered by the later of two events: a) the completion of the loading of info bits of the previous block and b) the completion of the encoded bits transfer of the block before previous.

The timing diagram of this approach is illustrated in Figure 3. For example the transfer of the configuration parameters of the transport block number 3, denoted as C3, is triggered by the end of the transfer of encoded bits of the code block number 1, denoted as O1. On the other hand, the transfer of C5 is triggered by the end of the transfer of the information bits of code block 4, denoted as D4. Also, if one configuration set describes M code blocks, the configuration of the next code block is triggered by the later of two events: a) the end of the load of Mth code block or b) the end of the coded bit transfer of (M-1)th code block. For example the transfer of C4 is triggered by the end of O3a.

The chaining of transport blocks is performed using a sync EDMA channel. Every time when triggered it copies one sync word from the sync buffer to the ESR register of the EDMA controller. If the sync word is zero the input trigger will not pass through, but if the word has some bit set to one, it will trigger the corresponding EDMA channel. In this approach each transfer of both the information bits and the encoded bits triggers the sync EDMA channel which further controls the pass through of the trigger signal to load the next transport block.

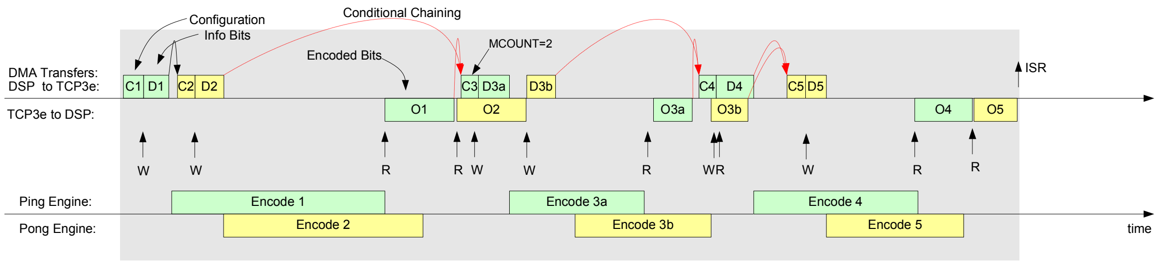

Figure 3 - Chaining using sync buffer

2.2.2 Pair-wise Chaining

In this approach the transport blocks are sent in pairs with exception of the transport blocks described with one configuration and M code blocks when they are sent individually. When both blocks are encoded and transferred from TCP3e the next pair is triggered in. The timing diagram of this approach is illustrated in Figure 4. This approach is simpler since it does not require conditional triggering using sync EDMA channel and sync buffer, but it does not fully employ the encoder engines.

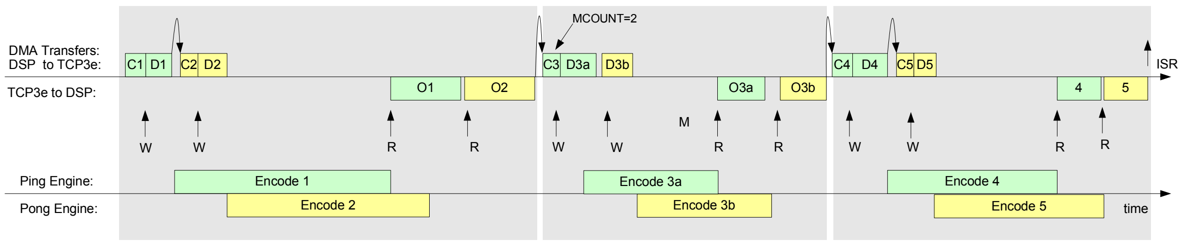
DMA Transfers:
DSP to TCP3e:
TCP3e to DSP:
MCOUNT=2
C1 D1 C2 D2 C3 D3a D3b C4 D4 C5 D5
O1 O2 O3a O3b 4 5
W W R R W W M R R W W R R
ISR
Ping Engine:
Pong Engine:
Encode 1 Encode 2 Encode 3a Encode 3b Encode 4 Encode 5
time

Figure 4 - Pair-wise chaining

2.2.3 Chaining without sync buffer

In this approach the chaining of transport blocks is performed as in the first approach with the sync channel, but without sync buffer. It is simpler since it does not require programming of the sync buffer, and triggering does not involve additional EDMA traffic of the sync words from L2 memory to EDMA controller ESR register. Here the EDMA sync channel passes through every other trigger signal by reloading sequentially two copies of dummy PaRAM sets, where only one of them has early completion chaining enabled.
The only functional difference of this approach from the first one is that the transport blocks described with one configuration and M code blocks chain the subsequent transport blocks after the Mth code block, and not (M-1)th.

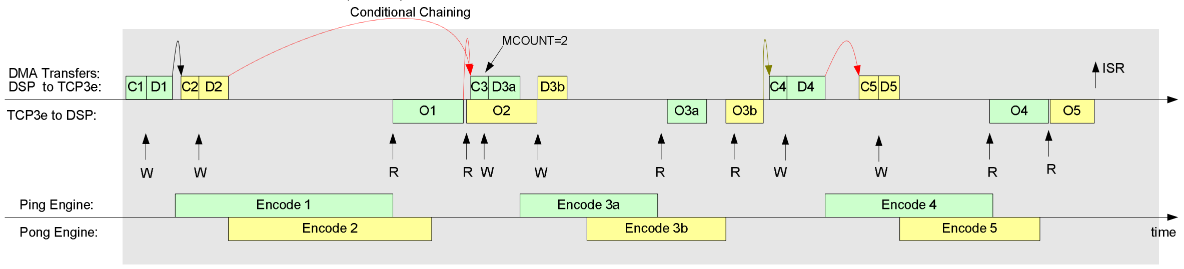

Figure 5 - Chaining approach without sync buffer

2.3 TCP3e Driver

This section describes one TCP3e driver design approach that is selected as a candidate that satisfies the best the required design goals. The approach is called double magazine approach.

2.3.1 Encoding process in the downlink bit processing chain

The position of the TCP3e driver in the downlink bit processing chain is illustrated in Figure 6 on the LTE example. The preprocessing mainly consists of the transport block segmentation. In LTE mode this only requires the calculation of the code block pointers since they are byte aligned and the TCP3e can internally calculate CRC for these blocks. Note that the TCP3e can only calculate CRC type B, (code block CRC), and not type A, (transport block CRC), so for the transport blocks consisting of only one code block the CRC has to be calculated by DSP. The post-processing involves more intensive calculations than the pre-processing. It includes rate matching, scrambling, constellation mapping and code block concatenation.

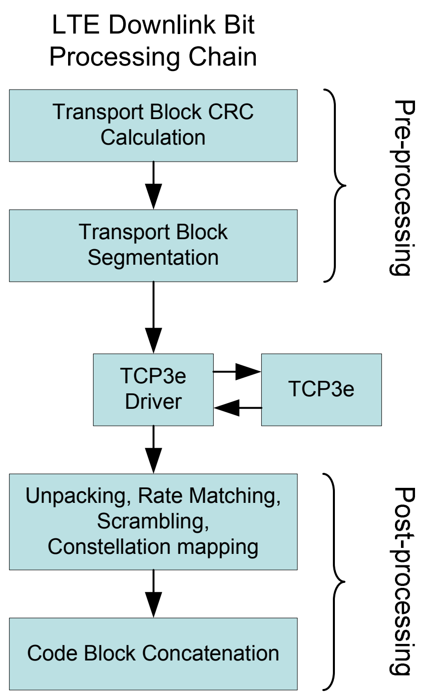
LTE Downlink Bit Processing Chain
Transport Block CRC Calculation
Transport Block Segmentation
Pre-processing
TCP3e Driver
TCP3e
Unpacking, Rate Matching, Scrambling, Constellation mapping
Code Block Concatenation
Post-processing

Figure 6 - TCP3e driver in the LTE downlink bit processing chain

2.3.2 Driver Design Goals

TCP3e driver design goals are listed below:

- Driver runs in continuous mode, unaware of the frame boundary.
- Driver involves minimum CPU intervention, minimum context switching.
- Driver receives block encoding requests at random times and in bursts.
- Driver supports multi core input and multi core output destinations.
- Minimum latency completion notification.

2.3.3 Double Magazine Approach

The main features of the double magazine approach are listed below:

- TCP3e driver consists of two magazines of PaRAM sets.
- Each magazine contains N transport block (TB) descriptors.
- Driver loads TB descriptors directly into PaRAM space of the free magazine while the other magazine is active.
- Switching between magazines is performed without CPU intervention.
- Driver runs on a single core. Requests from other cores can be received via hardware queue and passed to the driver.
- Completion notification for each TB is performed by pushing descriptors to the hardware queues. The outputs can be processed on multiple cores by specifying different destination queues.
- The enqueuing function temporarily disables the chain path between magazines to prevent the EDMA overrun during the update of the free magazine.

- If the free magazine becomes full, “sold out” case, the application can check when the magazine becomes available either by polling or by interrupt.

The high level diagram of the TCP3e driver in the downlink bit-processing chain is shown in Figure 7. The pre-processing tasks receive transport blocks from the MAC layer and perform block segmentation. The pre-processing task on the master core loads to the free magazine the self prepared TB descriptors or the descriptors prepared by the other cores and enqueued to TB queue. As soon as the blocks are completed, the driver pushes their descriptors to the destination queues. The post-processing tasks read TB descriptors from the destination queues and perform further downlink processing. The completed descriptors are recycled back to the free queue. If both magazines are full, the pre-processing task can save TB descriptors to the temporary queue and load them later once one of the magazines become free.

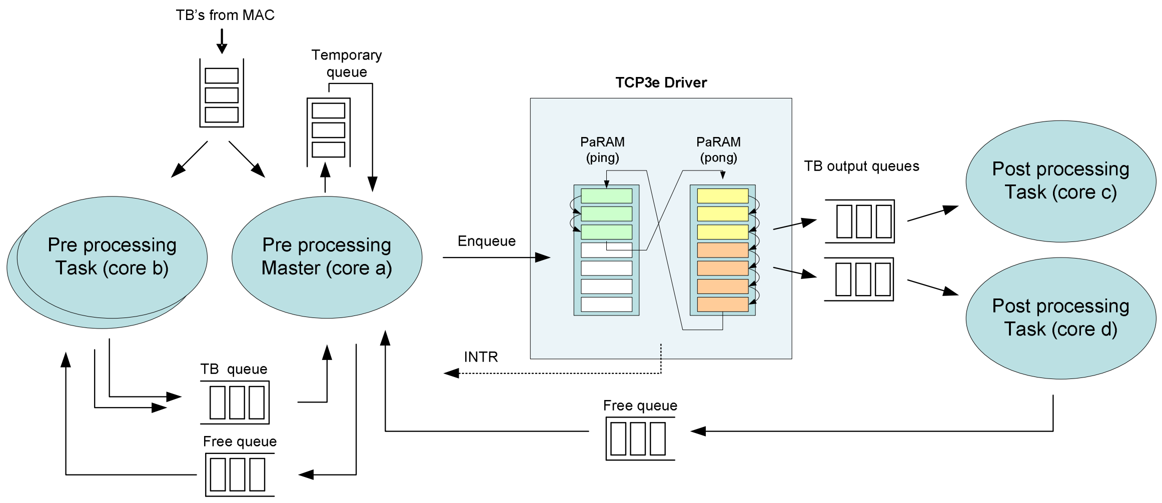

Figure 7 - TCP3e driver - double magazine approach

2.3.4 Magazine Control PaRAM entries

Several EDMA PaRAM entries and two physical EDMA channels are dedicated for magazine execution control and switching as shown Figure 8. The control PaRAM entries per magazine are listed below:

1) “GIR” – The magazine execution starts with this entry. It clears the bit in the CP_INTC1 raw status register. If the magazine is not loaded, it links in and triggers the “Stop” entry. If the magazine is loaded with one or more TB blocks it links in and triggers the “Run” entry.
2) “Run” – It copies the “running” flag to the driver state variable, links in the “Switch” entry, and triggers the encoding of the first transport block in the magazine (see Figure 10).
3) “Stop” – It copies the “stopped” flag to the driver state variable and links in the “GIR” PaRAM entry (see Figure 10).
4) “Switch” – It is triggered at the end of the last transport block in the magazine. It updates three words to the driver state variables: a) copies the “switching” flag to the driver state, b) toggles "free magazine" flag, and c) resets the number of elements in the free magazine. It also links in and triggers the “GIS” entry (see Figure 10).

5) "GIS" – This entry triggers the execution of the other magazine. It sets the bit in the CP_INTC1 raw status register thus triggering the EDMA control channel of the other magazine. It also generates the interrupt (if enabled) to DSP. It links in the "GIR" entry (with the default link to "Stop") so that the magazine is ready for the next execution.

The trigger signal path between magazines is shown in Figure 9. In the current design it passes through the CP_INTC1. The main reason for using CP_INTC1 is that the masking of the trigger signal is controllable by software. The enqueuing function temporary blocks this path during the loading of the free magazine. It disables the event in the event enable register corresponding to the magazine control channel. As soon as the loading is done it enables the event again. This protects against possible EDMA overrun during the loading process. If the running magazine is completed during the update of the free magazine, the event will be latched in the ER register and as soon as the update is completed, it will be processed by EDMA, and the free magazine will start running.

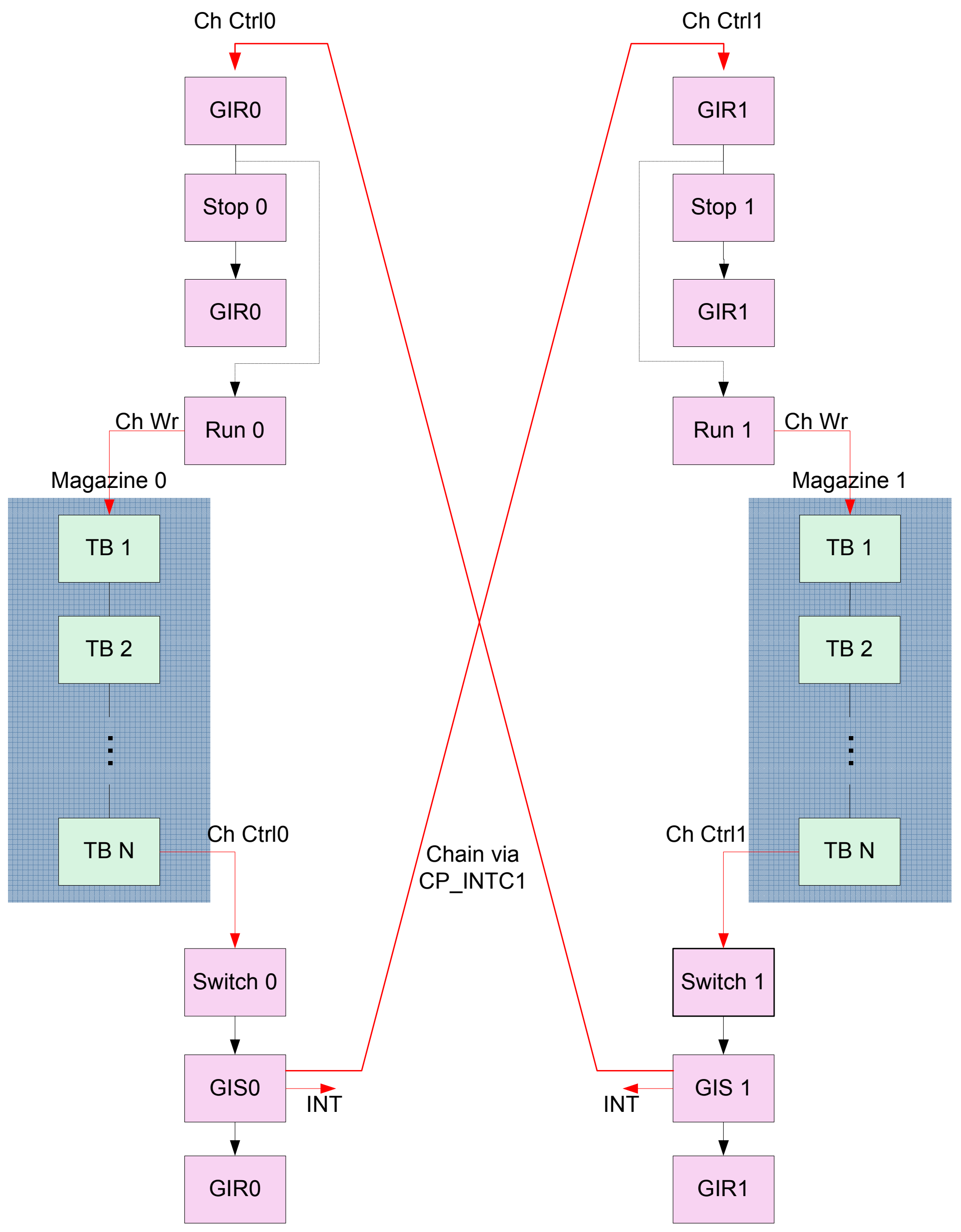

Figure 8 - Magazine control channels

The two EDMA control channels, Ctrl0 and Ctrl1, must be one of the EDMA channels that are associated to CP_INTC1. In Nyquist these channels are numbered from 34 to 63, and they are wired to the CP_INTC1 outputs numbered from 21 to 50, as out[21]→in[34], put[22]→in[35], etc. The "GIS" and "GIR" entries set and reset the input event to the CP_INTC1 system interrupt raw status register by writing the interrupt index value to the Status Index Set and Clear registers, respectively. The input event numbers are selected among reserved input numbers which are not wired to any physical block. Note that these input output event paths in the CP_INTC1 have to be programmed and enabled before starting the driver.

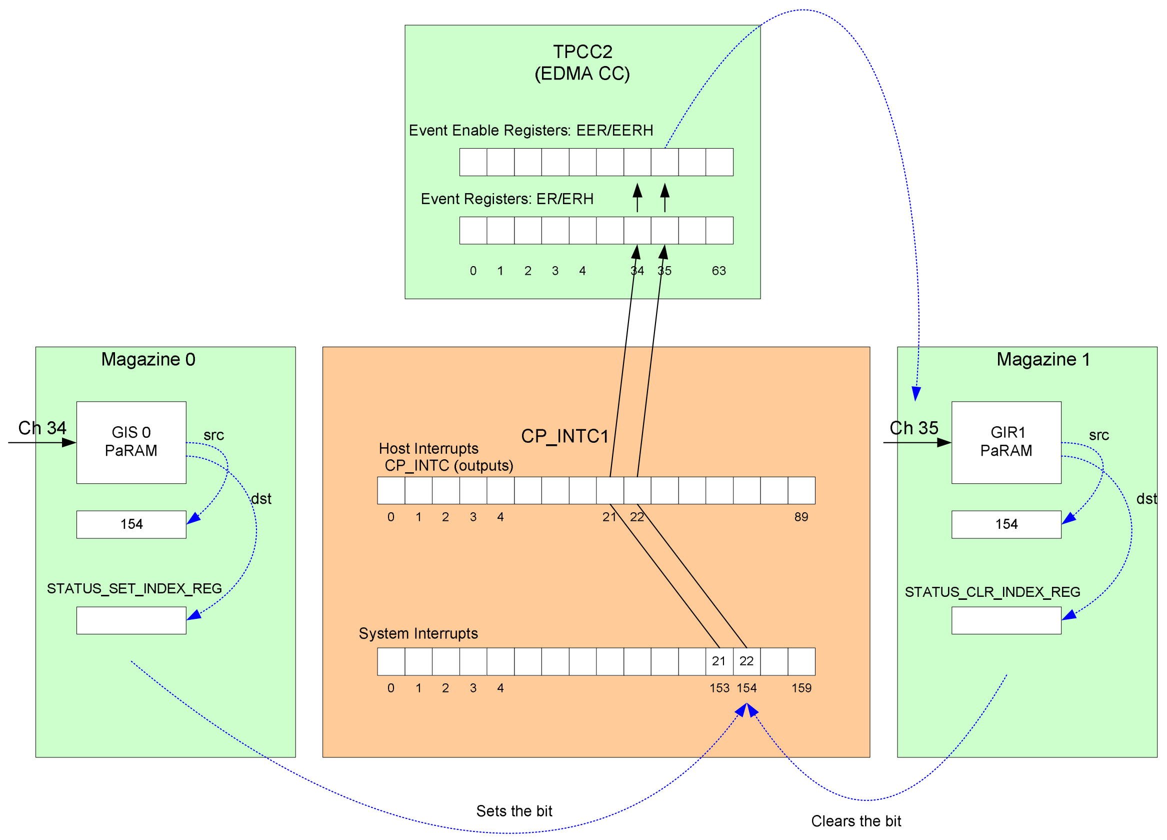

Figure 9 - Trigger signal path between magazines

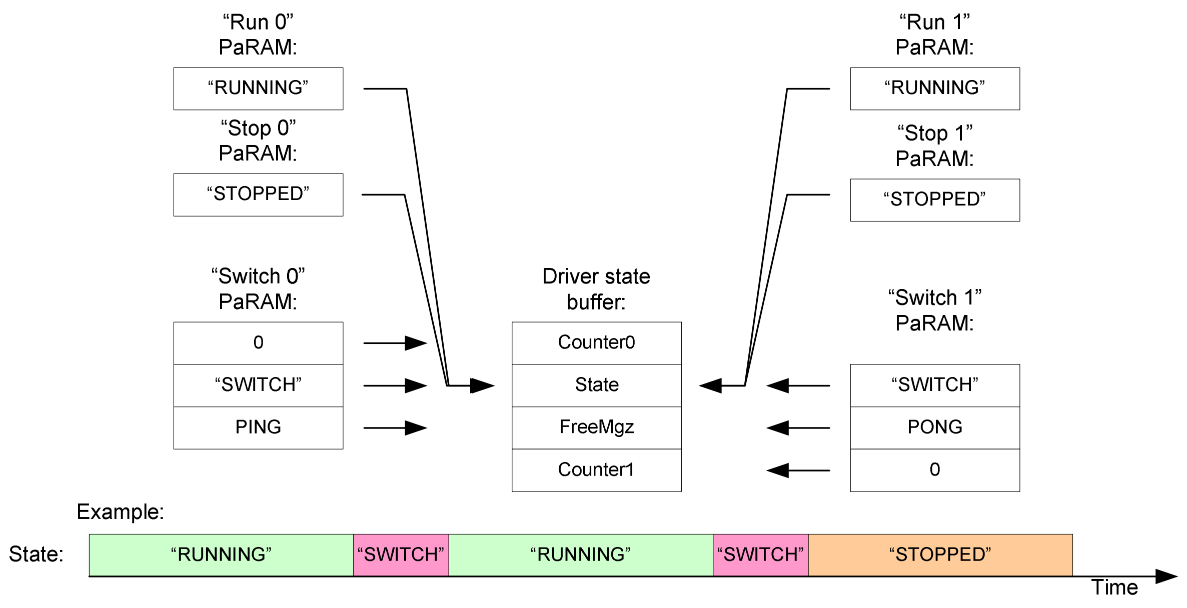

Figure 10 - Driver state variables

The control channel execution diagram is shown in Figure 11. The red arrows represent the trigger path, while dashed blue arrows represent the link path.

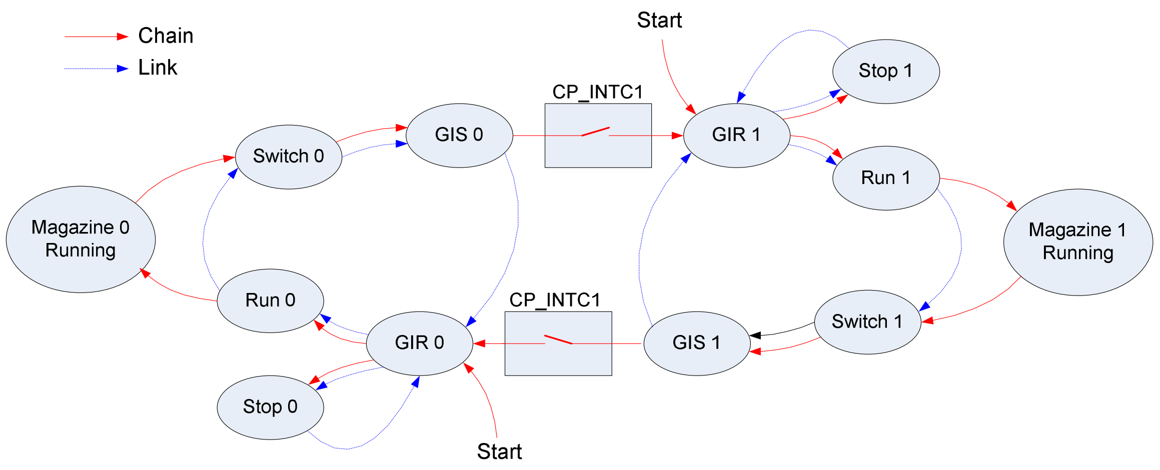

Figure 11 - Control channel execution diagram

The timing diagram of the driver state variables during the switching time is shown in Figure 12.

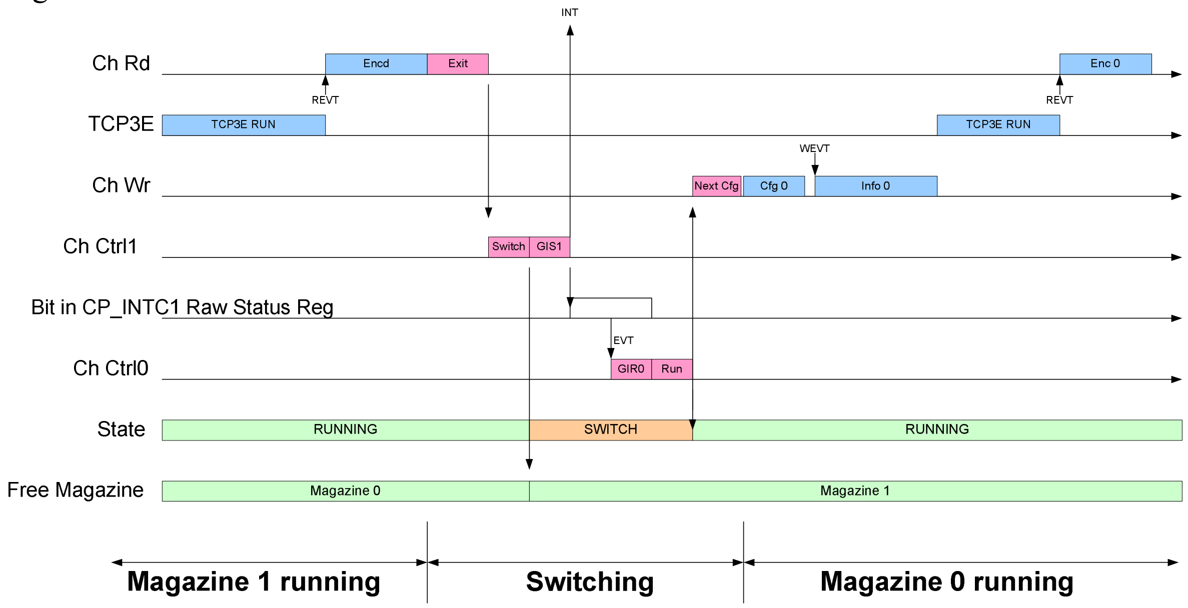

Figure 12 - Magazine switching - timing diagram

2.3.5 Magazine Data PaRAM Entries

Each transport block is handled with four PaRAM entries:

1) “Cfg_m_n” – This entry transfers the TCP3e input configuration registers (m: magazine index, n: TB index). It is executed on the WEVT channel, i.e. Ch Wr.

2) “Info_m_n” – This entry transfer information bits from DSP memory to TCP3e memory. It is executed on the Ch Wr.

3) “Enc_m_n” – This entry transfers the output encoded bits from TCP3e memory to DSP memory. It is executed on the REVT channel, i.e. Ch Rd.

4) “Que_m_n” – This entry pushes the TB descriptor to the notification hardware queue. It copies the TB descriptor address from the stored location in the TCP3e driver instance to the Register D of destination queue. If the destination queue address is set to NULL, the entry will perform dummy copy to the local address in the driver instance memory. It is triggered by the “Enc_m_n” entry. In the chaining approach with sync buffer this entry is executed on the separate physical channel, (Ch Que), while in other two approaches it is executed on the Ch Rd.

Additional PaRAM entries per magazine used for the link between them are:

5) “CLink_ m” – This entry is a dummy, and it links and chains to the “Cfg_m_n” entry of the first TB in the next magazine.
6) “ELink_m” – This entry is a dummy, and it links and chains to the “Enc_m_n” entry of the first TB in the next magazine.
7) “QLink_m” – This entry is a dummy, and it links and chains to the “Que _m_n” entry of the first TB in the next magazine. This entry is only used only with the chaining approach with sync buffer.
8) “Exit_m” – This entry is a dummy, and it links to the “ELink_m”, and triggers the magazine control channel to execute the “Switch” entry.

2.3.6 Double Magazine Driver with Chaining Using Sync Buffer

The block diagram of the TCP3e driver approach with sync buffer is shown in Figure 13 and Figure 14. The driver uses 7 physical channels:

1) Ch Rd - REVT is connected to this channel. The channel number is fixed to 2. The driver receives encoded bits over this channel.
2) Ch Wr - WEVT is connected to this channel. The channel number is fixed to 3. The driver sends the configuration registers and the info bits to TCP3e.
3) Ch Que - The Queue channel is used for the completion notification.
4) Ch Ctrl0 - The Control channel for Magazine 0.
5) Ch Ctrl1 - The Control channel for Magazine 1.
6) Ch S - The Sync channel used for triggering the next transport block in the magazine. It is triggered by “Info_m_n” via chaining and by “Enc_m_n” through the Ts channel.
7) Ch Ts - The Trigger sync channel is used for indirect triggering of Ch S by “Enc_m_n”, and also for indirect triggering of the Queue channel.

The following PaRAM entries are used for the code block synchronization:

- “Sync” – This entry is executed on the channel S and it copies one sync word from the sync word array to the ESR register. The possible sync word values are: 0x0 - no trigger, 0x4 - trigger bit for read channel, or 0x8 - trigger bit for write channel. The driver enqueue function writes these values according to the transport block MCOUNT value, with the last word set to trigger the read channel. It places two words, (0x4 and 0x0) per each transport block with MCOUNT=1, and 2×MCOUNT words, (0x4, 0x0, …, 0x0) for transport blocks with MCOUNT greater than one. It is triggered 2 times per code block. The reading from the array by EDMA and writing to it by driver enqueue function is performed in a circular manner. The length of the array should be long enough to accommodate for maximum number of CBs in two magazines to avoid overwriting by the enqueue function.

- "SyncReload" – This is the reload entry for the "Sync" entry.
- "TrigSync" – This entry is executed on the channel Ts and it copies one sync word from the trigger sync word array to the ESR register. The possible sync word values are: $2^Q + 2^S$ or 2^S where Q and S are queue and sync channel numbers respectively. The driver enqueue function writes one word per code block. This entry is executed at the end of each encoded bits transfer from TCP3e. It triggers the notification to the destination queue and it passes the trigger signal to the sync channel. The reading from the array and writing to it is performed in a circular manner.
- "TrigSyncReload" – This is the reload entry for the "TrigSync" entry.

The figures show EDMA PaRAM entry links for two magazines of size 4. As an example the EDMA entries in the magazine on the left show connections of the fully loaded magazine, while the ones in the magazine on the right show the empty magazine.

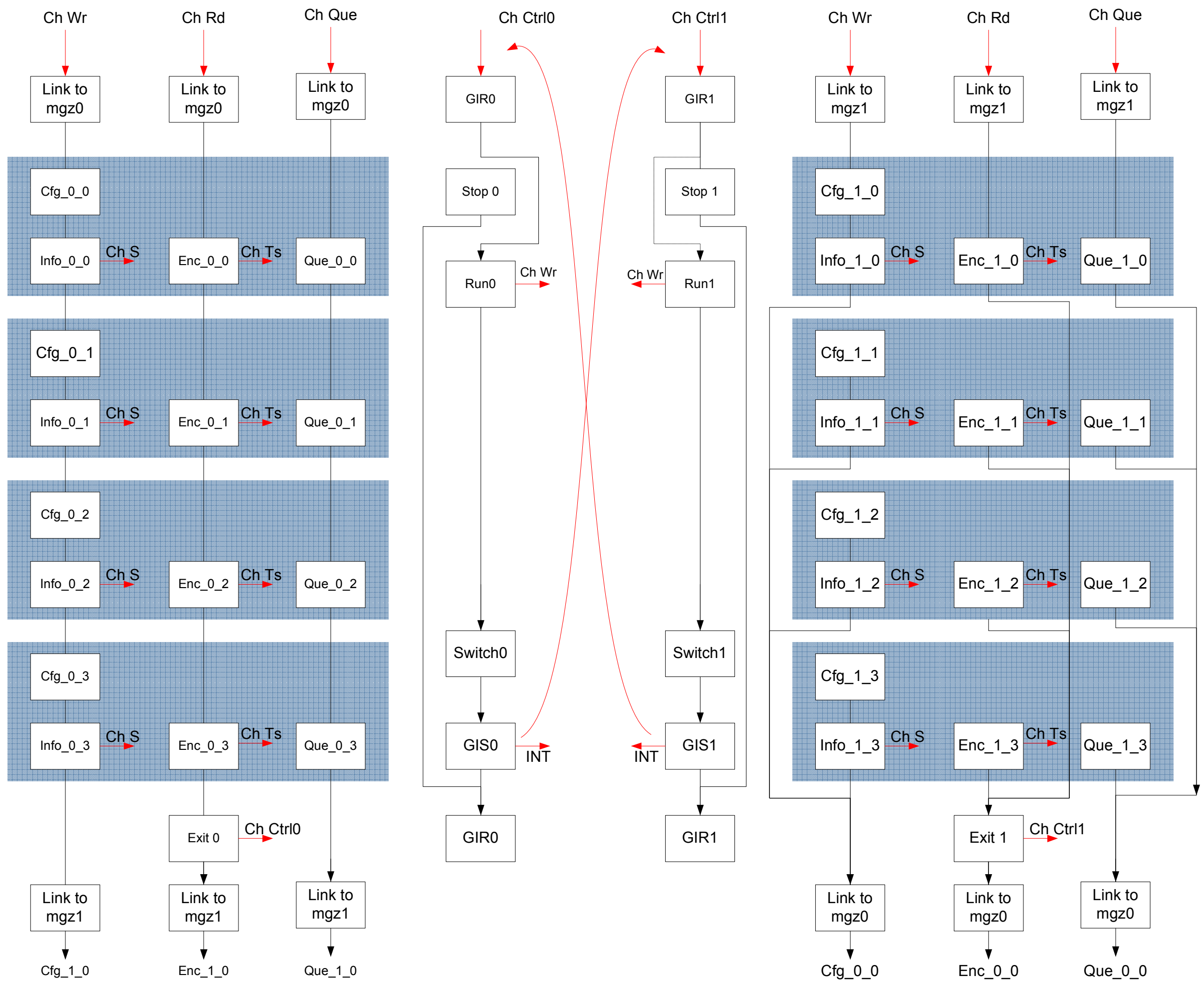

Figure 13 - Double Magazine - chaining with Sync Buffer

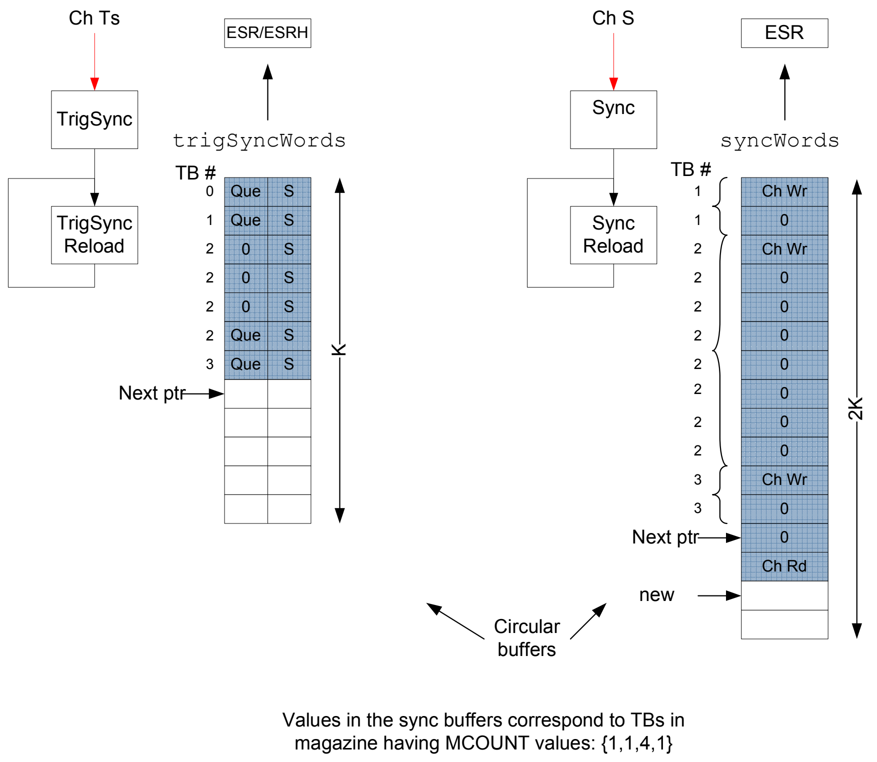

Figure 14 - Double Magazine - chaining with Sync Buffer (continued)

2.3.7 Double Magazine Driver with pair-wise Chaining

In this approach, the chaining between transport blocks is done without additional sync channels. The notification to the destination queue is performed on the same channel as the encoded data transfer. This approach requires only 4 physical channels. The chaining to the next transport block is alternatingly done by the info bits transfer and the encoded bits transfer, except when the transport block MCOUNT >1, the chaining is done by the last encoded bit transfer. The block diagram of PaRAM entry links is shown in Figure 15.

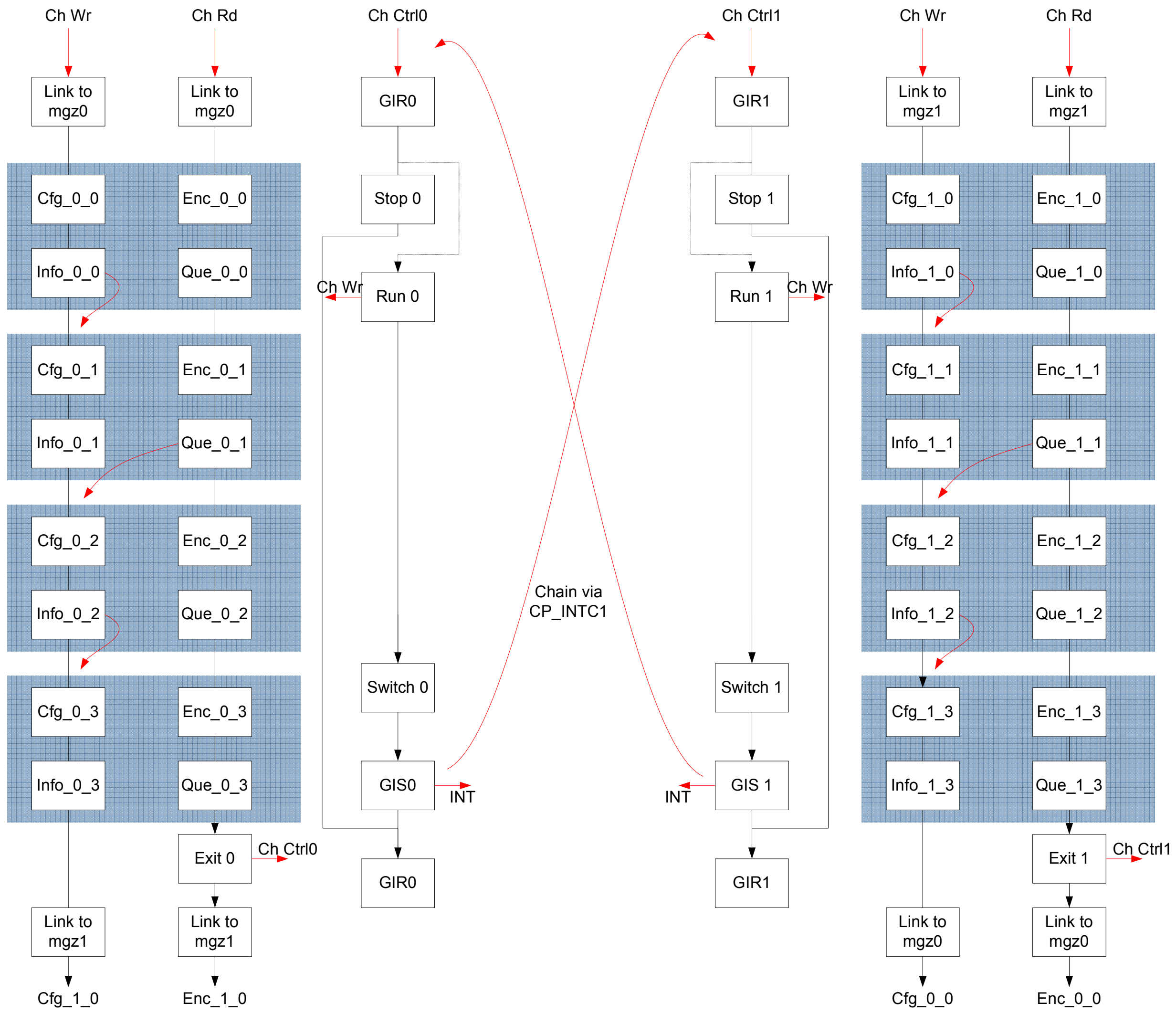

Figure 15 - Double Magazine with pair-wise chaining

2.3.8 Double Magazine Driver - Chaining without sync buffer

In this approach only one sync channel is used and the synchronization is performed with two PaRAM entries linked to each other that perform no copy, (dummy entries). Only one of them chains to the write channel. Basically the sync channel passes through every other input event. The notification to the destination queue is executed on the channel Rd, the same channel as the encoded data transfer. Both “Info_m_n” and “Que_m_n” entries trigger channel S, except when MCOUNT>1 only “Que_m_n” triggers channel S.

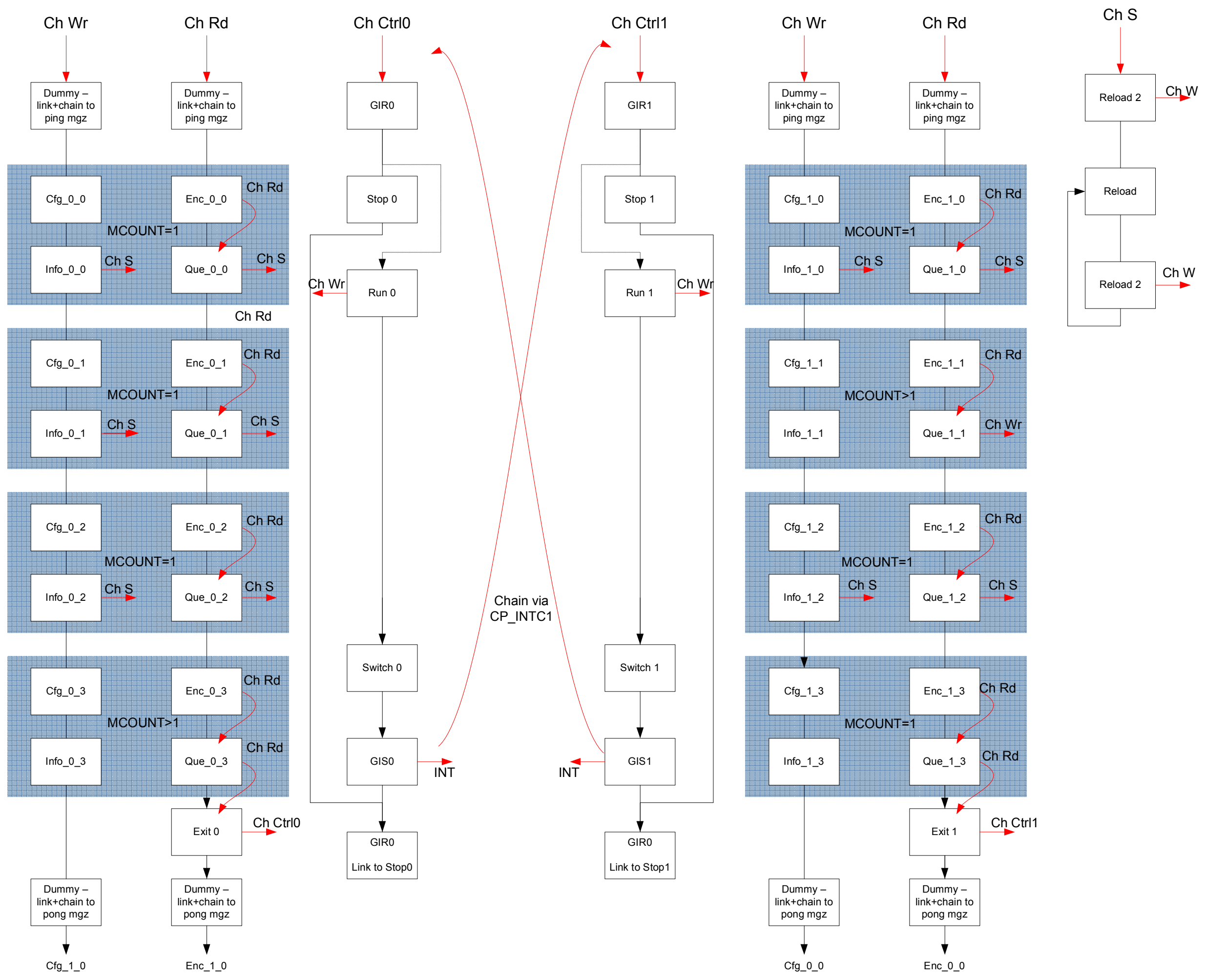

Figure 16 - Double Magazine - Chaining without Sync Buffer

2.3.9 Handling the "Sold Out" case

If the pre-processing task is loading transport blocks at the faster pace than the TCP3 encoder is encoding them at some point the free magazine, (one that is not running), will become full. This is a "sold out" case. The pre-processing task has to wait until the running magazine completes its execution, and then continue loading free magazine. It can handle this case in two ways: a) it continues to run, periodically polls the status of the driver, stores the new TB requests to the temporary queue, and when the magazine becomes free it loads TBs from the temporary queue and then the new ones, or b) it performs the following steps: clears the pending interrupts associated with the magazine switching, checks for free space in the magazine again, if it become free, loads the remaining TBs, otherwise enables interrupt and goes to sleep state. The driver will generate interrupt when one of the magazines becomes free. The sold out case is illustrated in Figure 17. The preprocessing task, after loading 8 TBs to two magazines, continued to process incoming TBs, (8th and 9th), and to poll the status of the driver. While the 10th block has been pre-processed, one magazine became free. After loading 8th, 9th, 10th and 11th block, the task enqueued the last 4 blocks to the temporary queue and performed the steps described in b) above. The last 4 blocks are loaded from the ISR.

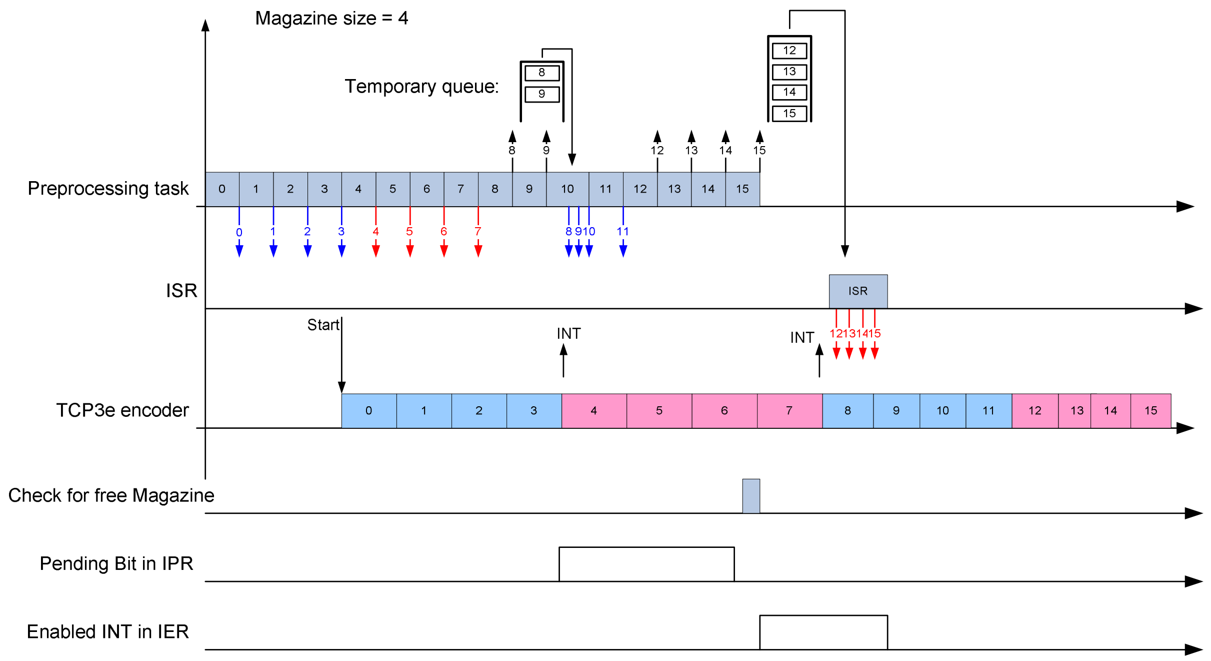

Figure 17 - Handling the "sold out" case

2.3.10 EDMA channel requirements

The number of required physical and linked channels for the three chaining approaches is summarized in Table 1. The maximum number of TB blocks per magazine is denoted as M, maximum number of CB's in both magazines is denoted as L.

	Chaining Approach		
EDMA Channels	**With Sync Buffer**	**Pair-wise**	**Without Sync Buffer**
Ch Rd = 2	Used	Used	Used
Ch Wr =3	Used	Used	Used
Ch Que	Used	Not used	Not used
Ch S	Used	Not used	Used
CH Ts	Used	Not used	Not used
Ch Ctrl 0	Used	Used	Used
Ch Ctrl 1	Used	Used	Used
# of linked Channels	2(4M +9) + 2	2(4M + 8)	2(4M + 8) + 2
Sync Buffer Size (Bytes)	12×L	0	0
Sync word transfers per CB	2	0	0

Table 1 - Required number of physical and linked channels for three chaining approaches

2.3.11 Initialization Sequence

Before using the driver the application performs several initialization steps:

1) Initialize the EDMA low level driver
2) Setup chain path within CP_INTC1
3) Init QMSS and allocate descriptor region for the TB descriptors

4) Open EDMA physical and linked channels required by driver
5) Register call back function associated with magazine control channels
6) Allocate memory for the driver instance
7) Call driver initialization function

This is illustrated in Figure 18.

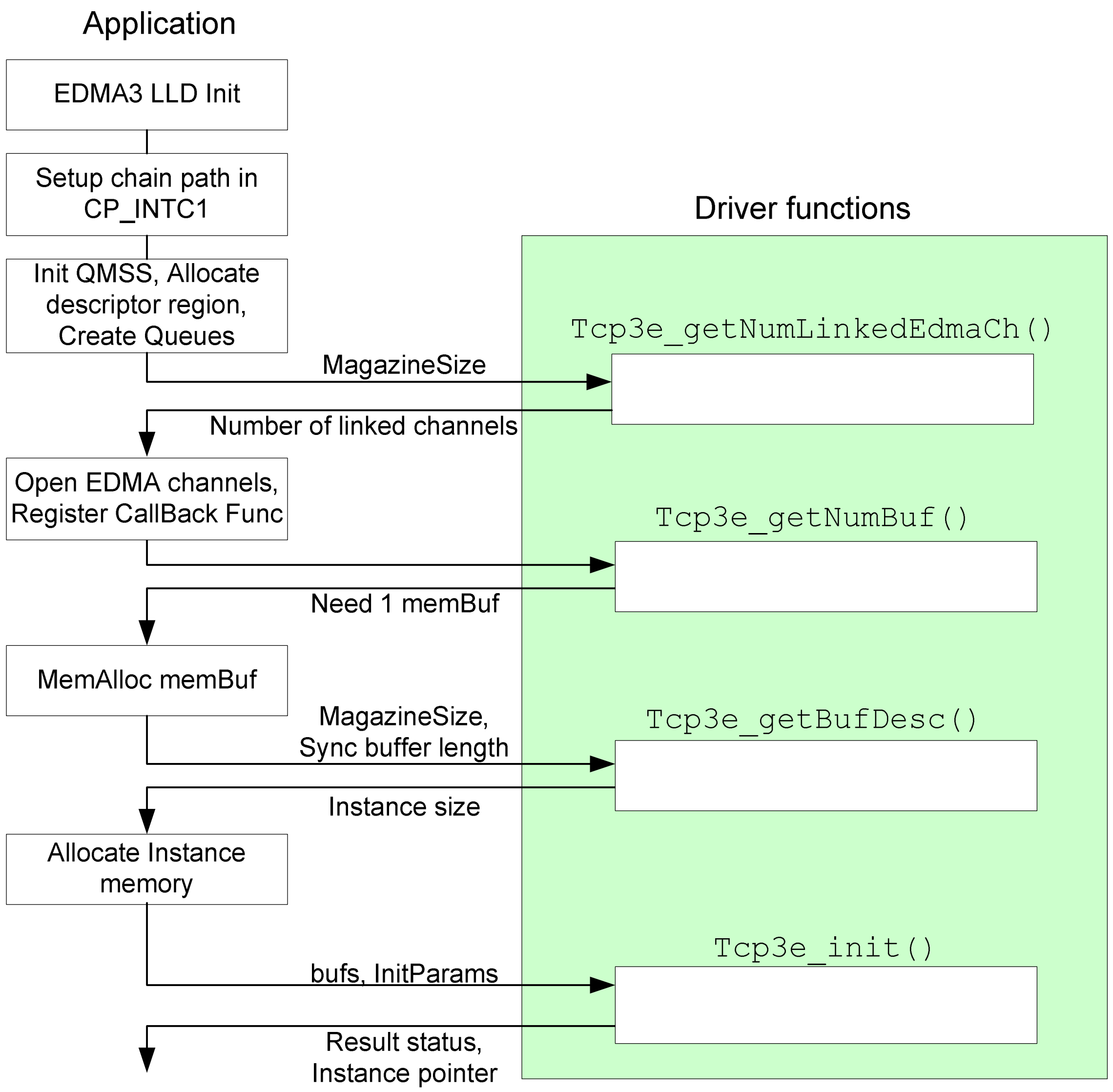
Application
EDMA3 LLD Init
Setup chain path in CP_INTC1
Init QMSS, Allocate descriptor region, Create Queues
Driver functions
Tcp3e_getNumLinkedEdmaCh()
MagazineSize
Number of linked channels
Open EDMA channels, Register CallBack Func
Tcp3e_getNumBuf()
Need 1 memBuf
MemAlloc memBuf
MagazineSize, Sync buffer length
Tcp3e_getBufDesc()
Instance size
Allocate Instance memory
Tcp3e_init()
bufs, InitParams
Result status, Instance pointer

Figure 18 - TCP3e driver initialization sequence

2.3.12 Driver Instance memory

The TCP3e driver instance memory includes a) driver instance data structure, b) array of magazine TB element descriptors, and c) trigger sync buffer and sync buffer (only for chaining with sync buffer). This is illustrated in Figure 19. Each element of the magazine TB array has pointers to four PaRAM sets. These PaRAM sets are requested as linked channels by the application using EDMA LLD and passed through the initialization structure to the driver initialization function. The driver initialization function uses EDMA LLD function to obtain the absolute address of PaRAM sets and populates these pointers. It also calculates the fixed fields of the PaRAM sets.

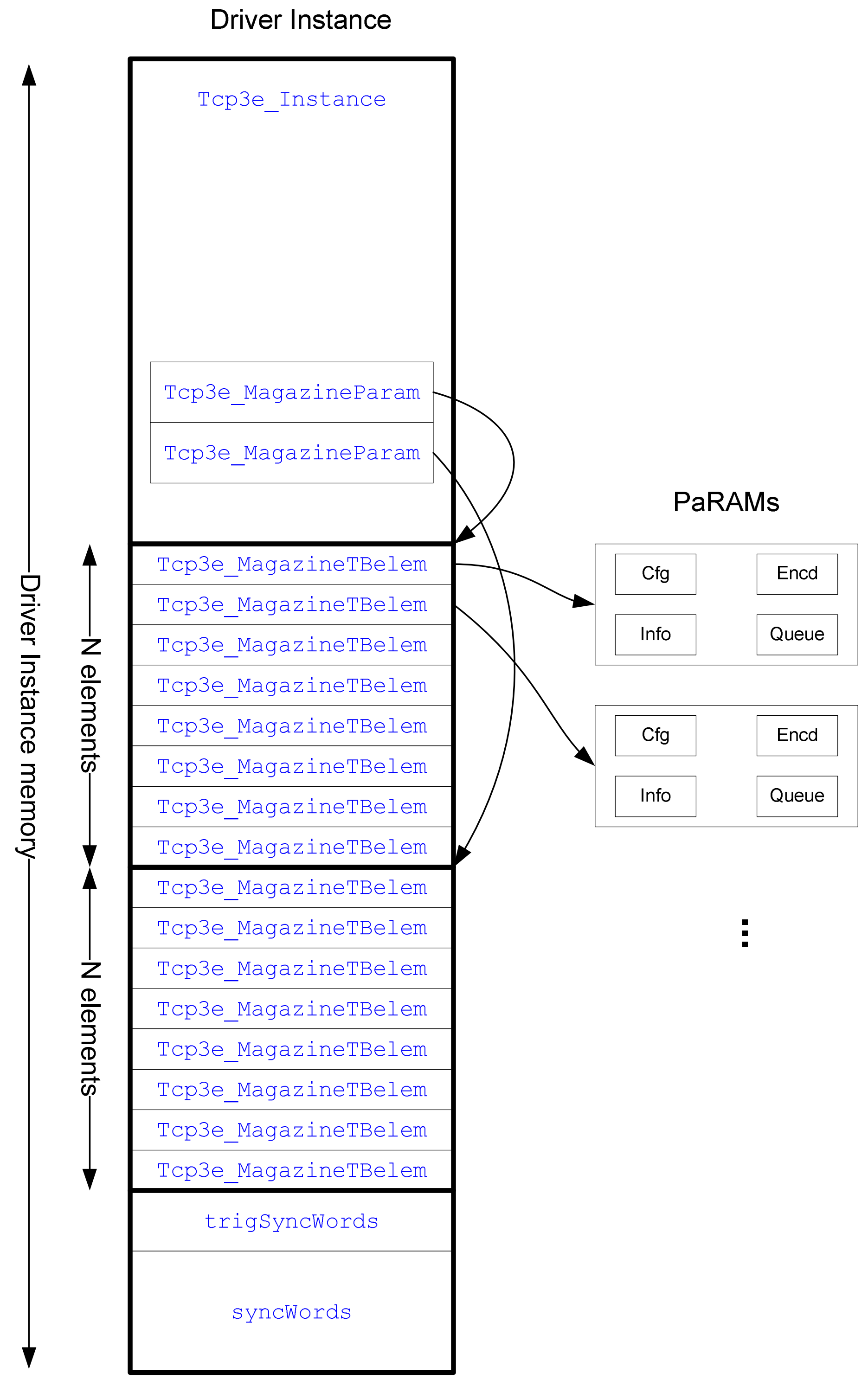

Figure 19 - Driver instance memory

2.3.13 Enqueuing Procedure

The application uses the driver enqueue function to add transport blocks to free magazine. If the driver is in the switching state the function will return status indicating that enqueuing failed. Since the switching state lasts very short time the application can keep trying in a while loop as long as the return status indicate that the magazine is in a switching state. The function input arguments are the list of TB descriptors and the number of TBs to be enqueued. The function will add TBs from the input list until the magazine becomes full and will return the number of successfully enqueued TBs. The flow diagram of the enqueue function for chaining with sync buffer is shown in Figure 20, while for chaining without sync buffer or pair-wise chaining is shown in Figure 21.

The input TB list is an array of pointers to data structure Tcp3QueueElem, (see Figure 22). This structure has parameters that describe transport block, such as the addresses of the tcp3e configuration parameters, input and output bits, block length, offsets, etc. The application may use as a TB descriptor a single Tcp3eQueueElem structure (shown in green), or it may add some other structures (as shown in yellow). The driver enqueue function uses only the array of pointers to Tcp3eQueueElem. The field in Tcp3eQueueElem, “queue Descriptor Pointer” has the address of the TB descriptor, which may be just the pointer to Tcp3eQueueElem itself. The other field shown in figure, “Destination queue address”, is the address of the destination hardware queue D Register. If the application does not use hardware queue, this field can be set to NULL, the driver in this case still performs one word transfer, but to its own dummy location.

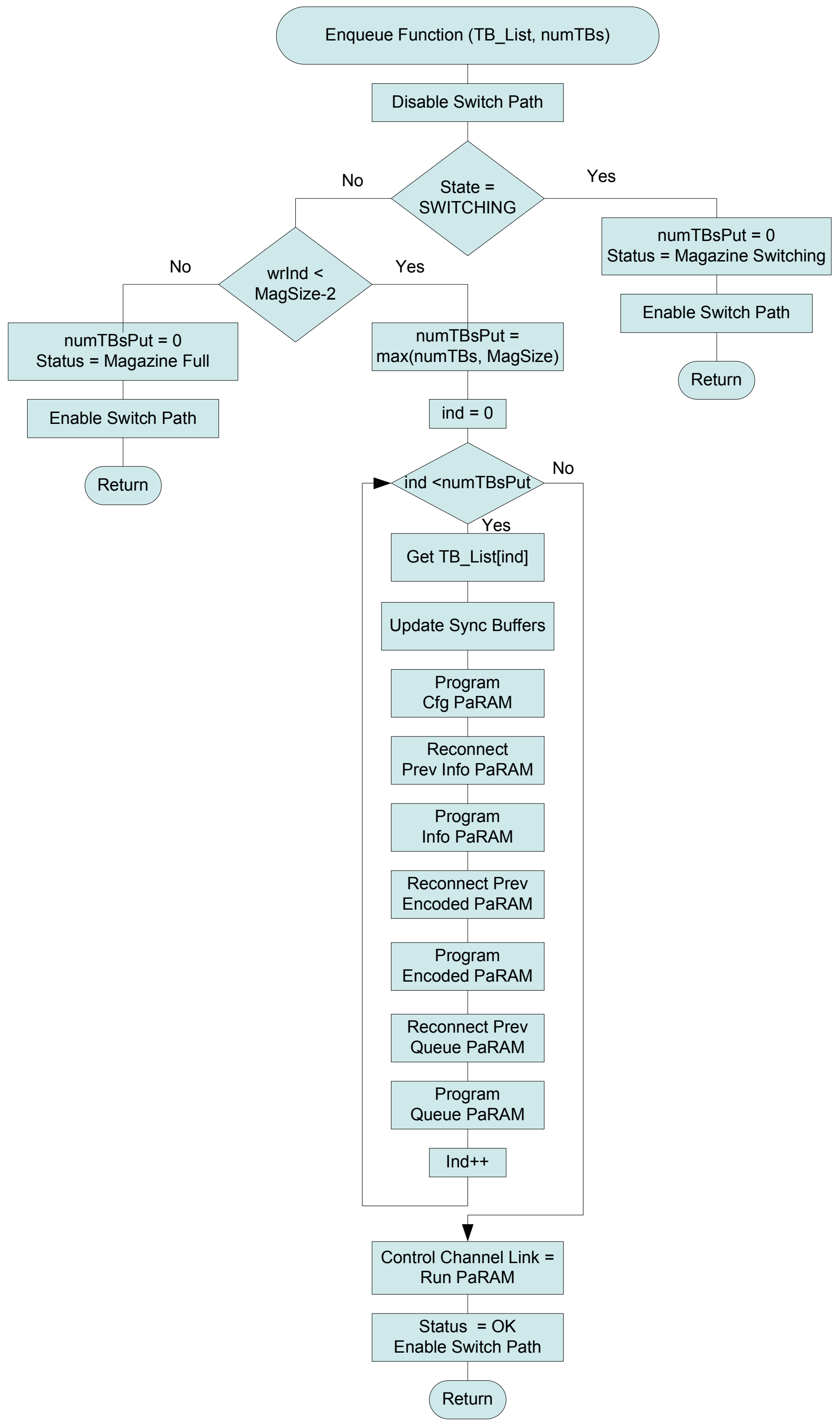

Figure 20 - Enqueue function flow diagram - chaining with sync buffer

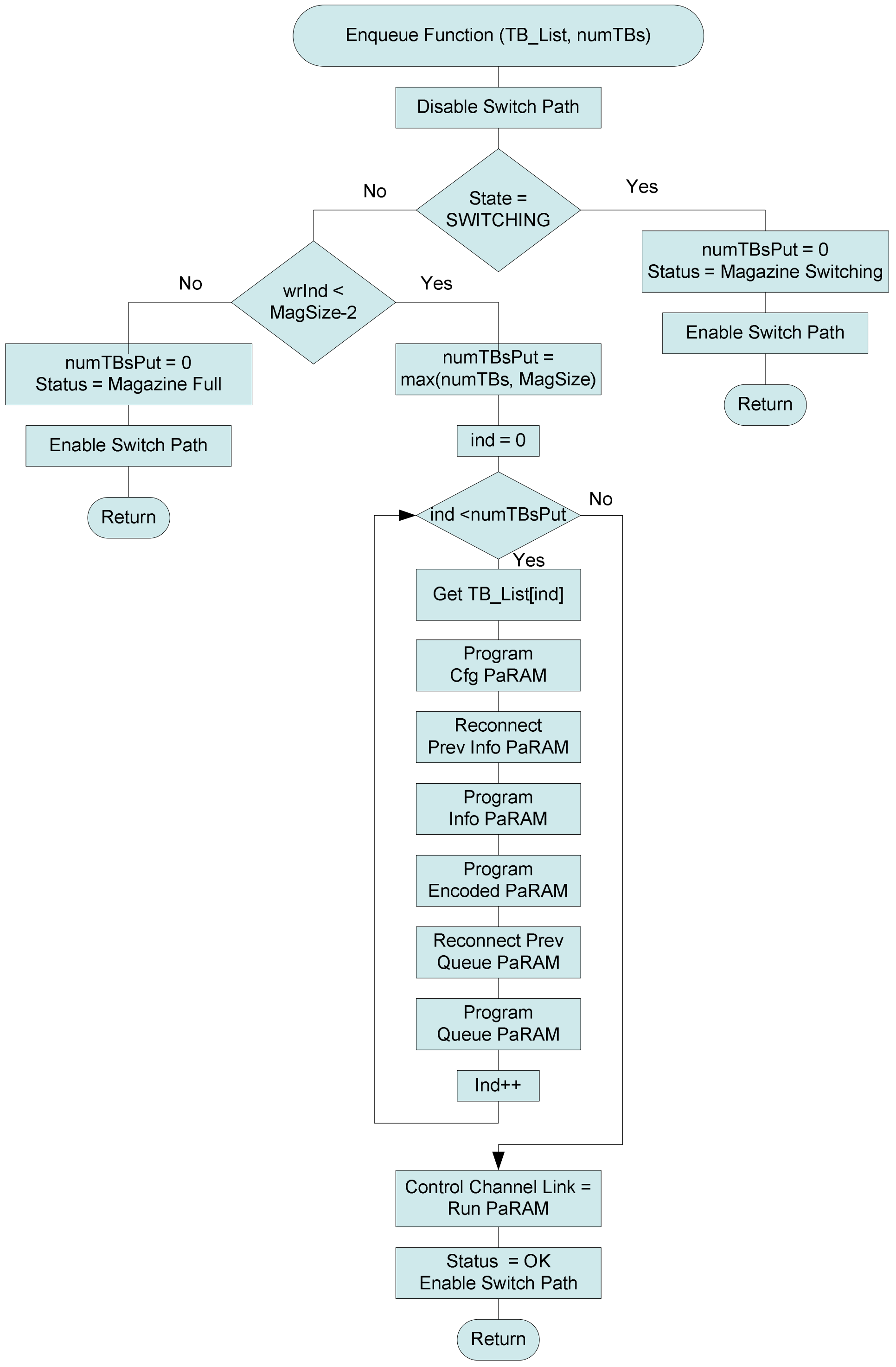

Figure 21 - Enqueue function flow diagram - chaining without sync buffer and pair-wise chaining

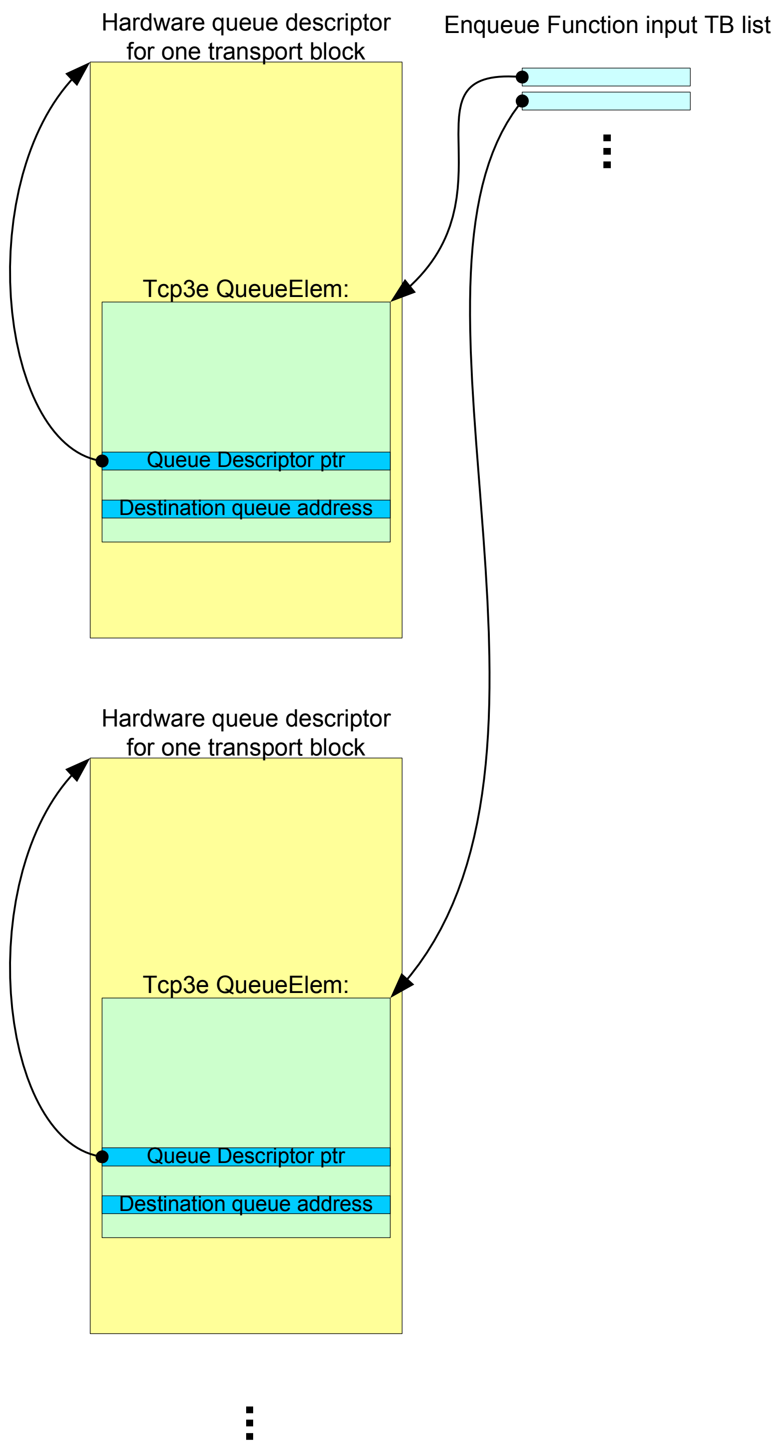
Hardware queue descriptor for one transport block
Enqueue Function input TB list
Tcp3e QueueElem:
Queue Descriptor ptr
Destination queue address
Hardware queue descriptor for one transport block
Tcp3e QueueElem:
Queue Descriptor ptr
Destination queue address

Figure 22 - Tcp3e driver enqueue function input TB list

2.3.14 Start Function

After the driver initialization the driver state is in IDLE. After loading the magazine first time, it has to be started using start function. After this function the driver enters the running state and it will never go back to the IDLE state as shown in Figure 24. If the encoding process is running at the faster pace than the pre-processing, the driver will go to STOPPED state, and it will have to be restarted using the start function. The flow diagram of the start function is shown in Figure 23.

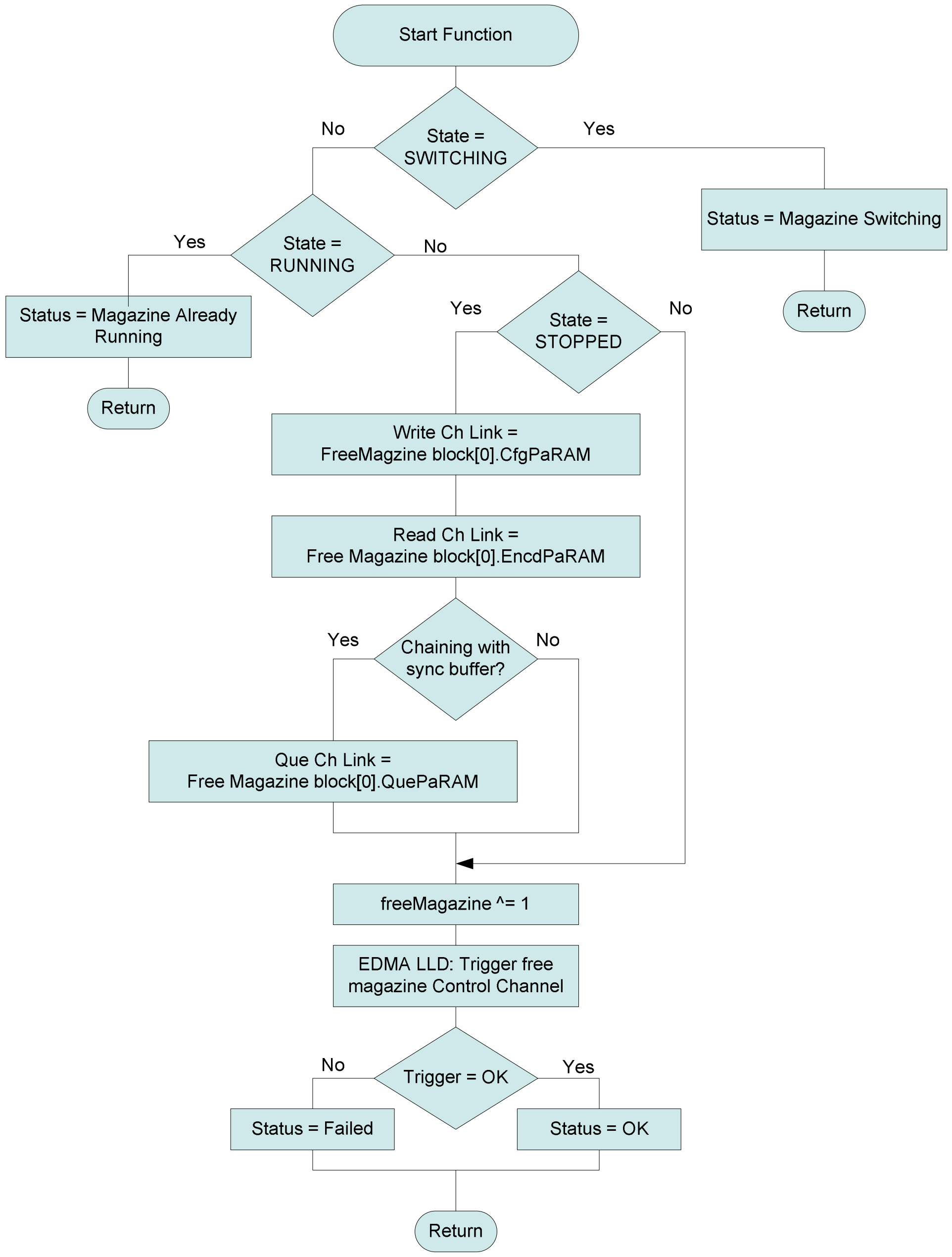
Start Function
State = SWITCHING
No
Yes
Status = Magazine Switching
Return
State = RUNNING
Yes
No
Status = Magazine Already Running
Return
State = STOPPED
Yes
No
Write Ch Link = FreeMagzine block[0].CfgPaRAM
Read Ch Link = Free Magazine block[0].EncdPaRAM
Chaining with sync buffer?
Yes
No
Que Ch Link = Free Magazine block[0].QuePaRAM
freeMagazine ^= 1
EDMA LLD: Trigger free magazine Control Channel
Trigger = OK
No
Yes
Status = Failed
Status = OK
Return

Figure 23 - TCP3e driver start function

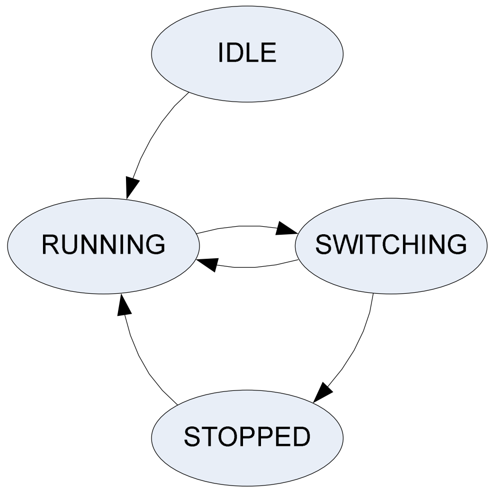

Figure 24 - Driver state diagram