TechNews: Ethernet Real-Time Core

Understanding the Core Buffer Management

The idea of further abstraction (real-time communication for different transport systems, such as serial communications, Ethernet (TCP / IP) and CANBUS (...) is realized with the SYBERA transportation libraries, the so-called Real-Time cores. Each Real-Time core is based on the SYBERA X-Realtime Engine. The core allow both, real-time Level1 (collecting and buffering of data at real-time, without loss of data), and Real-Time Level 2 (the cyclic functional operation at Real-Time, with one fixed buffer location). Additionally Level1 and Level2 may be combined. The buffer management of the ethernet real-time core is controlled by the flag lev2_flag of the core stack.

```
//Set core control elements
ETH_STACK_CONTROL(__pNicUserStack, 0, 0);      //Reset RX/TX buffer indexes
ETH_LEVEL2_CONTROL(__pNicUserStack, FALSE);    //Disable fixed buffer location
ETH_TASK_CONTROL(__pNicUserStack, TRUE);       //Enable RX and TX tasks
```

Core Tasks

The buffer management of the ethernet real-time core is connected to several programming interfaces. One is for cyclic frame handling inside the real-time application task, one is for acyclic frame transmission of WinSocket applications and another one is for acyclic frame transmission by a proprietary interface.
The basic of all these interfaces is a priority controlled ring buffer system, for TX (sending frames) and RX (receiving frames. Therefore 3 real-time tasks are involved:

RX task

- receiving ethernet frames into buffer
- placed inside ethernet core
- high priority

APP task

- logical data operation
- placed inside rea-time application
- low priority - 2

TX task

- transmitting ethernet frames from buffer
- placed inside ethernet core
- low priority - 1

TechNews: Ethernet Real-Time Core

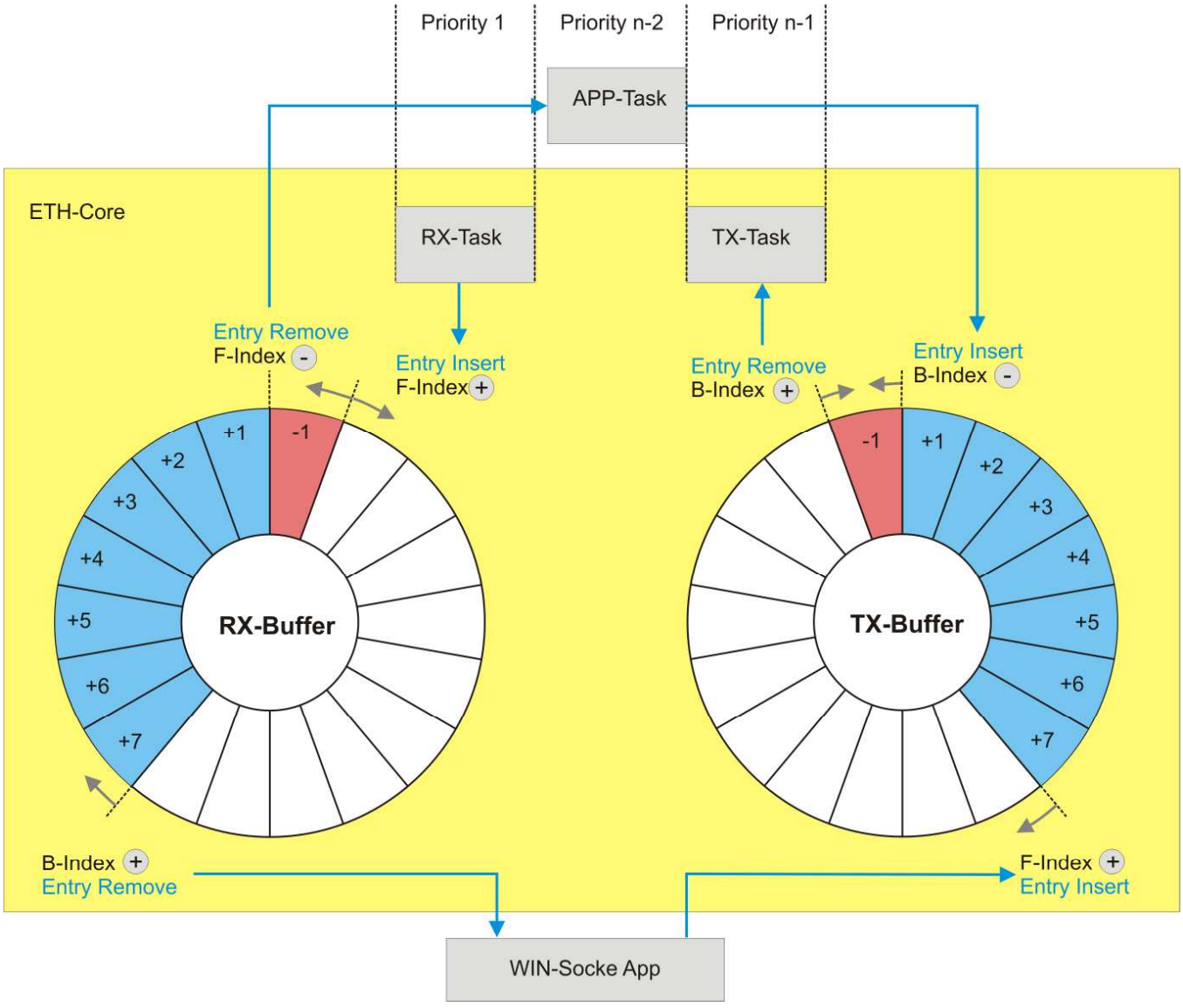
Understanding the Core Buffer Management

Buffer Tables

The ring buffer management consists of 2 table structures (RX and TX), containing elements of type of ETH_ENTRY. The ring buffer itself is controlled by a forward index (findex) and a backward index (bindex).

On receiving frames (RX) the integrated RX task first puts a new frame at the current forward index position, then increments the index. The following real-time APP task removes the entry by the forward index and then decrements the index. Asynchronously the WinSocket (or proprietary) interface removes an pending entry (due to the filter settings) by the current backward index, and increments the index.

On sending frames (TX) the real-time APP task inserts a new entry and then decrements the backward index. Afterwards the integrated TX task removes the entry by the backward index, then increments the index. Asynchronously the WinSocket (or proprietary) interface inserts new frames (due to the filter settings) by the forward index, then increments the index.



TechNews: Ethernet Real-Time Core

Understanding the Core Buffer Management

Tables Structures

```
typedef struct _ETH_TABLE
{
      UCHAR buffer[ETH_BUFFER_SIZE];      //Ethernet frame buffer
      ULONG findex;                       //Forward index
      ULONG bindex;                       //Back Index

      struct
      {
            ETH_ENTRY entry;

      } list[MAX_ETH_ENTRIES];

} ETH_TABLE, *PETH_TABLE;



typedef struct _ETH_ENTRY
{
      PETH_FRAME  pFrame;           //Pointer to the raw frame
      USHORT      FrameSize;        //Frame Size
      BOOLEAN     bOccupied;        //Occupied flag
      __int64     TscCnt;           //Time Scale Counter

} ETH_ENTRY, *PETH_ENTRY;
```

All structures are fully accessible inside the real-time application task, as well as inside the windows application. The stack pointer is returned on Sha32/64EthCreate(), which includes all elements for accessing the TX table inside the real-time application task (the include file ETH32/64Macros.h provides additional macros for table handling)

```
//Get current TX table entry
PETH_TABLE pTable = (PETH_TABLE)&__pStack->tx_table;
PULONG     bindex = (pTable->bindex == 0) ? (MAX_ETH_ENTRIES - 1)
                                          : (pTable->bindex - 1);

//Check for free TX entry
if (pNicTxEntry->bOccupied == FALSE)
{
      PETH_ENTRY pEntry = (PETH_ENTRY)&_pTable->list[bindex];

      //Save Information
      memcpy(pEntry->pFrame->s.pProtocol, ProtocolData, FRAME_SIZE);
      pNicTxEntry->FrameSize = FRAME_SIZE;

      //Update entry location and set handshake flage
      pTable->bindex = bindex;
}
```

TechNews: Ethernet Real-Time Core

Understanding the Core Buffer Management

Proprietary Interface

The proprietary interface allows easy sending and receiving of raw ethernet frames in Windows threads. The Frames will be led through by the core buffer management.

```
ETH_PARAMS  EthParams;

//Reset parameters
memset(&EthParams, 0, sizeof(ETH_PARAMS));
EthParams.dev_num = 0;              //Device number
EthParams.period = 100;             //Realtime period
EthParams.eth_type = ETH_TYPE_IP;   //Set ethernet frame type filter for
                                    //the selected interface
EthParams.eth_if = ETH_IF_CORE_RX | //Send all other ethernet frames to CORE
                   ETH_IF_CORE_TX;  //interface
EthParams.fpAppTask = AppTask;      //Real-Time application task

//Transmit ethernet frame
Sha64EthTransmitFrame(&EthParams);
```

Buffer Filter

The Ethernet realtime core contains an Y-Multiplexer between acyclic frames (typically Win Socket or proprietary Interface) and cyclic frames (logic frame operation inside real-time task) with a Cross Filter for Ethernet frames. The elements eth_type and eth_if allow to define a filter condition for the selected interface. Following values are allowed:

ETH_IF_SOCKET_RX	All RX frames of specified filter type will be directed to the socket interface
ETH_IF_SOCKET_TX	All TX frames of specified filter type will be directed to the socket interface
ETH_IF_CORE_RX	All RX frames of specified filter type will be directed to the core interface
ETH_IF_CORE_TX	All TX frames of specified filter type will be directed to the core interface

The ethernet interface filter is a bit mask, combining desired interface pathes. The following sample describes all frames (RX/TX) of specified filter type will be directed to the socket interface:

FilterIF = ETH_IF_SOCKET_RX | ETH_IF_SOCKET_TX

FilterType = 0	//No pass through to the selected interface
FilterType = ETH_TYPE_XXX	//Access only frames of type XXX for the selected interface. All other //frames are directed to the other interface

FilterIF = 0 FilterType = 0 No filter is set. All frames will be directed to all interfaces

Filter Logic:

```
if ((eth_type == filter_type) && ((filter_if & eth_if) == eth_if)) ||
    (eth_type != filter_type) && ((filter_if & eth_if) != eth_if)))
{ /*Handle Frame*/ … }
```