azure-rtos / usbx Public

Code Issues 20 Pull requests Discussions Actions Security 5 ...

master

usbx / common / usbx_host_classes / src / ux_host_class_cdc_ecm_mac_address_get.c

TiejunMS Release 6.2.0 ✕ History

3 contributors

355 lines (279 sloc) | 19.8 KB

```c
/**************************************************************************/
/*                                                                        */
/*       Copyright (c) Microsoft Corporation. All rights reserved.        */
/*                                                                        */
/*       This software is licensed under the Microsoft Software License   */
/*       Terms for Microsoft Azure RTOS. Full text of the license can be  */
/*       found in the LICENSE file at https://aka.ms/AzureRTOS_EULA       */
/*       and in the root directory of this software.                      */
/*                                                                        */
/**************************************************************************/


/**************************************************************************/
/**************************************************************************/
/**                                                                       */
/** USBX Component                                                        */
/**                                                                       */
/**   CDC ECM Class                                                       */
/**                                                                       */
/**************************************************************************/
/**************************************************************************/


/* Include necessary system files.  */

#define UX_SOURCE_CODE

#include "ux_api.h"
#include "ux_host_class_cdc_ecm.h"
```

```c
#include "ux_host_stack.h"


/**************************************************************************/
/*                                                                        */
/*  FUNCTION                                               RELEASE        */
/*                                                                        */
/*    _ux_host_class_cdc_ecm_mac_address_get              PORTABLE C      */
/*                                                           6.2.0        */
/*  AUTHOR                                                                */
/*                                                                        */
/*    Chaoqiong Xiao, Microsoft Corporation                               */
/*                                                                        */
/*  DESCRIPTION                                                           */
/*                                                                        */
/*    This function calls the USBX stack to retrieve the MAC address from */
/*    the configuration descriptor.                                       */
/*                                                                        */
/*  INPUT                                                                 */
/*                                                                        */
/*    cdc_ecm                               Pointer to cdc_ecm class     */
/*                                                                        */
/*  OUTPUT                                                                */
/*                                                                        */
/*    Completion Status                                                   */
/*                                                                        */
/*  CALLS                                                                 */
/*                                                                        */
/*    _ux_host_stack_transfer_request        Transfer request             */
/*    _ux_utility_memory_allocate            Allocate memory              */
/*    _ux_utility_memory_free                Free memory                  */
/*    _ux_utility_descriptor_parse           Parse descriptors            */
/*                                                                        */
/*  CALLED BY                                                             */
/*                                                                        */
/*    _ux_host_class_cdc_ecm_activate            CDC ECM class activate   */
/*                                                                        */
/*  RELEASE HISTORY                                                       */
/*                                                                        */
/*    DATE              NAME                      DESCRIPTION             */
/*                                                                        */
/*  05-19-2020     Chaoqiong Xiao           Initial Version 6.0           */
/*  09-30-2020     Chaoqiong Xiao           Modified comment(s),          */
/*                                            resulting in version 6.1    */
/*  07-29-2022     Chaoqiong Xiao           Modified comment(s),          */
/*                                            checked MAC string length,  */
/*                                            resulting in version 6.1.12 */
/*  10-31-2022     Chaoqiong Xiao           Modified comment(s),          */
/*                                            checked descriptor length,  */
/*                                            resulting in version 6.2.0  */
/*                                                                        */
/**************************************************************************/
UINT  _ux_host_class_cdc_ecm_mac_address_get(UX_HOST_CLASS_CDC_ECM *cdc_ecm)
{

UINT                                            status;
UX_ENDPOINT                                     *control_endpoint;
UX_TRANSFER                                     *transfer_request;
UX_CONFIGURATION_DESCRIPTOR                     configuration_descriptor;
UCHAR                                           *descriptor;
UCHAR                                           *start_descriptor = UX_NULL;
ULONG                                           configuration_index;
ULONG                                           total_configuration_length;
UINT                                            descriptor_length;
UINT                                            descriptor_type;
UINT                                            descriptor_subtype;
UX_HOST_CLASS_ECM_INTERFACE_DESCRIPTOR          ecm_interface_descriptor;
UCHAR                                           *mac_address_string;
ULONG                                           string_index;
ULONG                                           string_length;
UCHAR                                           element_content;
UCHAR                                           element_hexa_upper;
UCHAR                                           element_hexa_lower;

    /* We now need to retrieve the MAC address of the node which is embedded in the ECM descriptor
       We will parse the entire configuration descriptor of the device and look for the ECM Ethern
    configuration_index = cdc_ecm -> ux_host_class_cdc_ecm_interface_data -> ux_interface_configur

    /* We need to get the default control endpoint transfer request pointer.  */
    control_endpoint =  &cdc_ecm -> ux_host_class_cdc_ecm_device -> ux_device_control_endpoint;
    transfer_request =  &control_endpoint -> ux_endpoint_transfer_request;

    /* Need to allocate memory for the descriptor. Since we do not know the size of the
       descriptor, we first read the first bytes.  */
    descriptor =  _ux_utility_memory_allocate(UX_SAFE_ALIGN, UX_CACHE_SAFE_MEMORY, UX_CONFIGURATIO
    if (descriptor == UX_NULL)
        return(UX_MEMORY_INSUFFICIENT);

    /* Memorize the descriptor start address.  */
    start_descriptor =  descriptor;

    /* Create a transfer request for the GET_DESCRIPTOR request.  */
    transfer_request -> ux_transfer_request_data_pointer =      descriptor;
    transfer_request -> ux_transfer_request_requested_length =  UX_CONFIGURATION_DESCRIPTOR_LENGTH
    transfer_request -> ux_transfer_request_function =          UX_GET_DESCRIPTOR;
    transfer_request -> ux_transfer_request_type =              UX_REQUEST_IN | UX_REQUEST_TYPE_ST
    transfer_request -> ux_transfer_request_value =             (UX_CONFIGURATION_DESCRIPTOR_ITEM
    transfer_request -> ux_transfer_request_index =             0;
```

```c

    /* Send request to HCD layer.  */
    status =  _ux_host_stack_transfer_request(transfer_request);

    /* Check for correct transfer and entire descriptor returned.  */
    if ((status == UX_SUCCESS) && (transfer_request -> ux_transfer_request_actual_length == UX_CON
    {

        /* Parse the descriptor so that we can read the total length.  */
        _ux_utility_descriptor_parse(descriptor, _ux_system_configuration_descriptor_structure,
                                                                            UX_CONFIGURATION_DESCRIPTOR_ENTRIE

        /* We don't need this descriptor now.  */
        _ux_utility_memory_free(descriptor);

        /* Reallocate the memory necessary for the reading the entire descriptor.  */
        total_configuration_length =  configuration_descriptor.wTotalLength;
        descriptor =  _ux_utility_memory_allocate(UX_SAFE_ALIGN, UX_CACHE_SAFE_MEMORY, total_confi
        if (descriptor == UX_NULL)
            return(UX_MEMORY_INSUFFICIENT);

        /* Save this descriptor address.  */
        start_descriptor =  descriptor;

        /* Read the descriptor again with the correct length this time.  */
        transfer_request -> ux_transfer_request_requested_length =  total_configuration_length;

        /* Since the address of the descriptor may have changed, reprogram it.  */
        transfer_request -> ux_transfer_request_data_pointer =  descriptor;

        /* Send request to HCD layer.  */
        status =  _ux_host_stack_transfer_request(transfer_request);

        /* Check for correct transfer and entire descriptor returned.  */
        if ((status == UX_SUCCESS) && (transfer_request -> ux_transfer_request_actual_length == co
        {

            /* The ECM descriptor is embedded within the configuration descriptor. We parse the
               entire descriptor to locate the ECM functional descriptor portion.  */
            while (total_configuration_length)
            {

                /* Gather the length and type of the descriptor.   */
                descriptor_length  =  *descriptor;
                descriptor_type    =  *(descriptor + 1);
                descriptor_subtype =  *(descriptor + 2);

                /* Descriptor length validation.  */
                if (descriptor_length < 3 || descriptor_length > total_configuration_length)
                {

                    /* Error trap.  */
                    _ux_system_error_handler(UX_SYSTEM_LEVEL_THREAD, UX_SYSTEM_CONTEXT_CLASS, UX_D

                    /* Free descriptor memory.  */
                    _ux_utility_memory_free(start_descriptor);

                    /* Return error.  */
                    return(UX_DESCRIPTOR_CORRUPTED);
                }

                /* Check the type for an interface descriptor and the subtype for a ECM functional
                if ((descriptor_type == UX_HOST_CLASS_CDC_ECM_CS_INTERFACE) && (descriptor_subtype
                {

                    /* Parse the interface descriptor and make it machine independent.  */
                    _ux_utility_descriptor_parse(descriptor,
                                _ux_system_ecm_interface_descriptor_structure,
                                UX_HOST_CLASS_CDC_ECM_INTERFACE_DESCRIPTOR_ENTRIES,
                                (UCHAR *) &ecm_interface_descriptor);


                    /* Release the memory.  */
                    _ux_utility_memory_free(start_descriptor);

                    /* We now have the ECM functional descriptor in memory. We can retrieve the in
                       which we need for NetX.  */

                    /* Allocate memory for the MAC address.  */
                    mac_address_string =  _ux_utility_memory_allocate(UX_SAFE_ALIGN, UX_CACHE_SAFE

                    /* Check memory allocation.  */
                    if (mac_address_string == UX_NULL)
                        return(UX_MEMORY_INSUFFICIENT);

                    /* Create a transfer request for the GET_DESCRIPTOR request.  */
                    transfer_request -> ux_transfer_request_data_pointer =      mac_address_string
                    transfer_request -> ux_transfer_request_requested_length =  UX_HOST_CLASS_CDC_
                    transfer_request -> ux_transfer_request_function =          UX_GET_DESCRIPTOR;
                    transfer_request -> ux_transfer_request_type =              UX_REQUEST_IN | UX
                    transfer_request -> ux_transfer_request_value =             (UX_STRING_DESCRIP
                    transfer_request -> ux_transfer_request_index =             0x0409;

                    /* Send request to HCD layer.  */
                    status =  _ux_host_stack_transfer_request(transfer_request);

                    /* Check for correct transfer. */
                    if (status == UX_SUCCESS)
                    {

                        /* Translate from Unicode to string. Length is in the first byte followed 
                           We must take away 2 from it and divide by 2 to find the right ascii len
                        string_length = (ULONG) *mac_address_string;

                        /* Check the length of the MAC address Unicode string
                           (length or 1B + type of 1B + string or 12*2B).  */
                        if (string_length != 26)
                        {

                            /* Error trap. */
                            _ux_system_error_handler(UX_SYSTEM_LEVEL_THREAD, UX_SYSTEM_CONTEXT_CLA

                            /* Return error.  */
                            status =  UX_DESCRIPTOR_CORRUPTED;
                        }
                        else
                        {

                            /* No error in length, decode the string.  */
                            string_length -=2;
                            string_length = string_length / 2;

                            /* Now we have a string of 12 hex ASCII digits to be translated into 6
                               and copy into the node ID.  */
                            for (string_index = 0; string_index < string_length; string_index++)
                            {

                                /* Get the upper element from the ASCII string.  */
                                element_content = *(mac_address_string + (string_index * 2) + 2);

                                /* We have a valid element content.  Turn it into a hex decimal va
                                   that only hex digits are allowed.  */
                                if (element_content <= '9')

                                    /* We have a digit.  */
                                    element_hexa_upper = (UCHAR)(element_content - '0');

                                else
                                {
                                    /* We have a 'A' to 'F' or 'a' to 'f' value.  */
                                    if (element_content >= 'a')

                                        /* We have a 'a' to 'f' char.  */
                                        element_hexa_upper = (UCHAR)(element_content - 'a' + 10);

                                    else

```

```
                                        /* We have a 'A' to 'F' char.  */
                                        element_hexa_upper = (UCHAR)(element_content - 'A' + 10);

                                }

                                /* Get the lower element from the ASCII string.  */
                                element_content = *(mac_address_string + ((string_index + 1) * 2)

                                /* We have a valid element content.  Turn it into a hexa decimal v
                                   that only hex digits are allowed.  */
                                if (element_content <= '9')

                                    /* We have a digit.  */
                                    element_hexa_lower = (UCHAR)(element_content - '0');

                                else
                                {
                                    /* We have a 'A' to 'F' or 'a' to 'f' value.  */
                                    if (element_content >= 'a')

                                        /* We have a 'a' to 'f' char.  */
                                        element_hexa_lower = (UCHAR)(element_content - 'a' + 10);

                                    else

                                        /* We have a 'A' to 'F' char.  */
                                        element_hexa_lower = (UCHAR)(element_content - 'A' + 10);

                                }

                                /* Assemble the byte from the 2 nibbles and store it into the node
                                *(cdc_ecm -> ux_host_class_cdc_ecm_node_id + string_index / 2) = (

                                /* Skip the lower nibble. */
                                string_index ++;

                            }

                            /* Operation was successful ! */
                            status = UX_SUCCESS;
                        }
                    }
                    else
                    {

                        /* We have a bad MAC address string.  Do not proceed.  */
                        status = UX_ERROR;
                    }

```

```
                        /* Free the MAC address string.  */
                        _ux_utility_memory_free(mac_address_string);

                        /* Return completion status.  */
                        return(status);
                    }
                    else
                    {

                        /* Jump to the next descriptor if we have not reached the end.  */
                        descriptor +=  descriptor_length;

                        /* And adjust the length left to parse in the descriptor.  */
                        total_configuration_length -=  descriptor_length;
                    }
                }
            }
        }

        /* Error trap. */
        _ux_system_error_handler(UX_SYSTEM_LEVEL_THREAD, UX_SYSTEM_CONTEXT_CLASS, UX_DESCRIPTOR_CORRUP

        /* If trace is enabled, insert this event into the trace buffer.  */
        UX_TRACE_IN_LINE_INSERT(UX_TRACE_ERROR, UX_DESCRIPTOR_CORRUPTED, &configuration_descriptor, 0,

        /* Release the memory.  */
        _ux_utility_memory_free(start_descriptor);

        /* Return an error.  */
        return(UX_DESCRIPTOR_CORRUPTED);

    }
```