Ingenieurbüro für Echtzeitprogrammierung

RT-LAN Funktions

Netzwerkstackerweiterungen unter RTOS-UH

Ab Software-Rev. 13.8 vom 13.11.2012

Inhaltsverzeichnis

Revisionsliste:

Rev.	Datum	Na.	Änderung
13.4	01.02.2012	Kr	Erweiterung der Funktionstabelle möglich
13.6	28.08.2012	Kr	Delete einer Funktion
13.8	13.11.2012	Kr	Callback um VOID* data erweitert

1 Urheberrecht und Haftung

Alle Rechte an diesen Unterlagen liegen bei der IEP GmbH, Langenhagen.

Die Vervielfältigung, auch auszugsweise, ist nur mit unserer ausdrücklichen schriftlichen Genehmigung zulässig.

In Verbindung mit dem Kauf von Software erwirbt der Käufer einfaches, nicht übertragbares Nutzungsrecht. Dieses Recht zur Nutzung bezieht sich ausschließlich darauf, daß dieses Produkt auf oder in Zusammenhang mit jeweils **einem** Computer zu benutzen ist. Das Erstellen einer Kopie ist ausschließlich zu Archivierungszwecken unter Aufsicht des Käufers oder seines Beauftragten zulässig. Der Käufer haftet für Schäden, die sich aus der Verletzung seiner Sorgfaltspflicht ergeben, z.B. bei unautorisiertem Kopieren, unberechtigter Weitergabe der Software usw.. Der Käufer gibt mit dem Erwerb der Software seine Zustimmung zu den genannten Bedingungen. Bei unlizensiertem Kopieren muß vorbehaltlich einer endgültigen juristischen Klärung von Diebstahl ausgegangen werden. Dies gilt ebenso für Dokumentation und Software, die durch Modifikation aus Unterlagen und Programmen von IEP hervorgegangen ist, gleichgültig, ob die Änderungen als geringfügig oder erheblich anzusehen sind.

Eine Haftung seitens IEP für Schäden, die auf den Gebrauch von Software, Hardware oder Benutzung dieses Manuskriptes zurückzuführen sind, wird ausdrücklich ausgeschlossen, auch für den Fall fehlerhafter Software oder irrtümlicher Angaben.

Das Einverständnis des Käufers oder Nutzers für den Haftungsausschluß gilt mit dem Kauf und der Nutzung der Software und dieser Unterlagen als erteilt.

2 Netzwerkerweiterung RTOS-UH

Damit andere Protokollarten vom Netzwerkstack erkannt und ausgewertet werden können, ist der Netzwerkstack auf der Empfangsseite um eine Funktionstabelle erweitert worden.

Die Funktionstabelle umfaßt maximal 16 verschiedene Eingangsfunktionen zur Interpretation des eingehenden Paketes. Sie hat folgenden Aufbau:

```
typedef int(*fktptr)(void *ip_in, Ce**work_ce, void * data);

#pragma MEMBER_PADDING_68K
#pragma STRUCT_SIZE_WORD
struct fkt_extend
{
   unsigned short type ; /* Ethernet Typ Field           */
   fktptr         fkt  ; /* the external function to call */
   void  * data ; /* userData für Datenaustausch */
} ;
#pragma MEMBER_PADDING_OLD
#pragma STRUCT_SIZE_OLD

fkt_extend ETH_IN_fkt[16]  ;
```

Für den normalen Ethernetverkehr sind die ersten 3 Einträge fest vergeben mit folgenden Funktionen.

```
ETH_IN_fkt[0].type = IP_PROTOCOL ;     /* für IP-Verkehr  */
ETH_IN_fkt[1].type = ARP_PROTOCOL ;    /* für ARP-Verkehr */
ETH_IN_fkt[2].type = IEEE801_2Q ;      /* für VLAN-ID     */
```

alle weiteren Plätze sind mit `.type = 0` vorbesetzt und somit als ungültig makiert.

Die aktuelle Tabelle der besetzten Protocolarten kann mit *snetfkt* ausgegeben werden.

3 Kodierungsvorschrift der Funktionen

Die einzuklinkende Funktion muss folgender Kodierungsvorschrift genügen. Falls sie das eingehende Paket vollständig bearbeitet, muss sie den RETURN-Wert = 0 zurückliefern, da dieser Wert vom Netzwerkstack genutzt wird, um eine optionale weitere Intepretation ab dem Offset des Return-Wert in dem eingelaufenen Paket zu veranlassen.

```
int my_protocol(void *ip_in, Ce**work_ce, void * u_data)
{
    ….
    return( 0 ) ; /* keine weitere Verarbeitung */
}
```

Die Parameter *ip_in, work_ce und u_data* werden vom Netzwerkstack an die Funktion übergeben, wobei *ip_in* auf die Position nach den Ethernetheader zeigt und *work_ce* auf das komplette CE.

u_data wird bei der Installation der Funktion besetzt, um einen Datenaustausch der von Netzwerkstack aufgerufenen Funktion mit der eigenen Anwendung zu ermöglichen.

Man beachte, dass diese externe Funktion direkt vom Netzwerkstack aufgerufen wird und somit keine static, local Variablen innerhalb der Funktion erlaubt sind.

4 Einklinken einer Funktion

Um dem Netzwerkstack eine neue Funktion einzuklinken, ist ein Paket mit folgendem Aufbau an den Netzwerkstack zu schicken:

```
#pragma MEMBER_PADDING_68K
#pragma STRUCT_SIZE_WORD
typedef struct fkt_linked
{
   unsigned short code ; /* must be 0x0020 = 32           */
   unsigned short type ; /* Ethernet Typ Field            */
   fktptr         fkt  ; /* the external function to call */
   void * data ; /* for data exchange */
} ;
#pragma MEMBER_PADDING_OLD
#pragma STRUCT_SIZE_OLD

/* Es soll die Funktion lldp_in eingeklinked werden */
int lldp_in( void * ip_ptr , Ce ** work_ce , void * data )
{
   ...
   return( 0 ) ;
}

    void
set_my_function( void )
{
   fkt_linked m_fkt ;
   int success ;
   Ce * m_ce = rt_fetch_ce( sizeof( m_fkt ) ) ;
   m_fkt.code = 32        ; /* besetzen des Dateninhalt */
   m_fkt.type = LLDP_TYPE ;
   m_fkt.fkt  = lldp_in   ;
   m_fkt.data = (void*)&meine_daten ;
   /* Besetzen der Ce Parameter */
   m_ce->ldn   = 16 ;  /* NET_LDN */
   m_ce->drive =  3 ;  /* ETH_DRV */
   m_ce->mode  = MODMWA | MODMOU | IOCRW ;
```

```
        m_ce->reclen = sizeof( m_fkt ) ;
        m_ce->buffer = ( char*) &m_fkt ;
        rt_transfer_ce( m_ce ) ;
        success = m_ce->reclen ; /* 1=erfolgreich, 0= nicht erfolgreich ) */
        rt_release_ce( m_ce ) ;
    }
```

Nun wird für alle Pakete, die den EthernetType = LLDP_TYPE haben, die Funktion `lldp_in` vom Netzwerkstack aufgerufen. Der Erfolg des Einklinken kann über .reclen abgefragt werden.

5 Ausklinken einer Funktion

Um aus dem Netzwerkstack eine Funktion auszuklinken, ist ein Paket mit folgendem Aufbau an den Netzwerkstack zu schicken:

```
#pragma MEMBER_PADDING_68K
#pragma STRUCT_SIZE_WORD
typedef struct fkt_unlinked
{
   unsigned short code ; /* must be 0x0021 = 33 */
   unsigned short type ; /* Ethernet Typ Field  */
} ;
#pragma MEMBER_PADDING_OLD
#pragma STRUCT_SIZE_OLD

    void
delete_my_function( void )
{
   fkt_unlinked m_fkt ;
   int success ;
   Ce * m_ce = rt_fetch_ce( sizeof( m_fkt ) ) ;

   /* besetzen des Dateninhalt */
   m_fkt.code = 33 ;
   m_fkt.type = LLDP_TYPE ;

   /* Besetzen der Ce Parameter */
   m_ce->ldn    = 16 ;  /* NET_LDN */
   m_ce->drive  =  3 ;  /* ETH_DRV */
   m_ce->mode   =  MODMWA | MODMOU | IOCRW ;
   m_ce->reclen = sizeof( m_fkt ) ;
   m_ce->buffer = ( char*) &m_fkt ;
   rt_transfer_ce( m_ce ) ;
   success = m_ce->reclen ; /* 1=erfolgreich, 0= nicht erfolgreich ) */
   rt_release_ce( m_ce ) ;
}
```

6 ARP Callback einrichten

Um im Netzwerkstack einen ARP Callback einzurichten, ist ein Paket mit dem folgendem Aufbau an den Netzwerkstack zu schicken. Die eingeklinkte Funktion wird vom Netzwerkstack aufgerufen, wenn ein gültiger ARP-Response vom Netzwerk eintrifft.

```
#pragma MEMBER_PADDING_68K
#pragma STRUCT_SIZE_WORD
typedef struct arp_linked
{
   unsigned short code ; /* must be 0x0022 = 34           */
   fktptr         fkt  ; /* the external function to call */
   void*          data ; /* for data exchange             */
} ;
#pragma MEMBER_PADDING_OLD
#pragma STRUCT_SIZE_OLD

int arp_callback( unsigned long ip_adr, char* eth_addr, void* data )
{
}

   void
install_arp_callback( void )
{
   arp_linked m_fkt ;
   Ce * m_ce = rt_fetch_ce( sizeof( m_fkt ) ) ;
   m_fkt.code = 34 ; /* besetzen des Dateninhalt */
   m_fkt.fkt  = arp_callback ;
   m_fkt.data = (void*)&meine_daten ;
   m_ce->ldn   = 16 ;  /* NET_LDN */
   m_ce->drive =  3 ;  /* ETH_DRV */
   m_ce->mode  = MODMWA | MODMOU | IOCRW ;
   m_ce->reclen = sizeof( m_fkt ) ;
   m_ce->buffer = ( char *) &m_fkt ;
   rt_transfer_ce( m_ce ) ;
   rt_release_ce( m_ce ) ;
}
```

7 ARP Senden

Ein ARP für eine freiwählbare IP-Adresse innerhalb der Routing-Tabellen kann folgend gesendet werden:

```
#pragma MEMBER_PADDING_68K
#pragma STRUCT_SIZE_WORD
typedef struct arp_packet
{
   unsigned long ip_adr     ;
   char          eth_adr[6] ; /* optionale Rückgabe */
} arp_packet ;
#pragma MEMBER_PADDING_OLD
#pragma STRUCT_SIZE_OLD

   void
send_arp( unsigned long ip_adr )
{
   arp_packet m_arp ;
   Ce * m_ce = rt_fetch_ce( sizeof( m_arp ) ) ;
   /* besetzen des Dateninhalt */
   m_arp.ip_adr  = ip_adr ;

   /* Besetzen der Ce Parameter */
   m_ce->ldn    = 16 ;  /* NET_LDN */
   m_ce->drive  =  5 ;  /* ARP_DRV */
   m_ce->mode   =  MODMWA | MODMOU | IOCRW ;
   m_ce->reclen = sizeof( m_arp ) ;
   m_ce->buffer = ( char*) &m_arp ;
   rt_transfer_ce( m_ce ) ;
   rt_release_ce( m_ce ) ;
}
```

8 Setzen der VLAN-ID

Um dem Netzwerkstack eine VLAN-ID zu setzen, ist ein Paket mit folgendem Aufbau an den Netzwerkstack zu schicken:

```
#pragma MEMBER_PADDING_68K
#pragma STRUCT_SIZE_WORD
typedef struct fkt_linked
{
   unsigned short code   ; /* must be 0x001F = 31 */
   unsigned short vlan_id; /* VLAN-ID             */
} ;
#pragma MEMBER_PADDING_OLD
#pragma STRUCT_SIZE_OLD
   void
set_my_vlan_id( void )
{
   fkt_linked m_fkt ;
   Ce * m_ce = rt_fetch_ce( sizeof( m_fkt ) ) ;
   /* besetzen des Dateninhalt */
   m_fkt.code = 31 ;
   m_fkt.type = 0x1234 ;  /* nur auf diese VLAN-ID reagieren */
   /* Besetzen der Ce Parameter */
   m_ce->ldn    = 16 ;  /* NET_LDN */
   m_ce->drive  =  3 ;  /* ETH_DRV */
   m_ce->mode   =  MODMWA | MODMOU | IOCRW ;
   m_ce->reclen = sizeof( m_fkt ) ;
   m_ce->buffer = ( char*) &m_fkt ;
   rt_transfer_ce( m_ce ) ;
   rt_release_ce( m_ce ) ;
}
```

Solange die gesetzte VLAN-ID <> 0xFFFF ist, werden alle Netzwerkpakete gefiltert und nur Pakete mit der gesetzten VLAN-ID empfangen.