Applieddata.net

Embedded Computer Systems

Software Specification

For

J1708 Driver

11025-10041
Version 1.0
Last Revision: December 8, 2004

Document History:

Version	**Date**	**By**	**Notes**
1.0	12/9/2004	MSS	Initial Draft

Introduction

The J1708 driver was written to support an interface to this bus on the AGX platform via a third party adapter. The adapter connects to serial port 1 and is powered by the platform. This adapter performs the required RS232 to RS485 signal conversions and it also provides a signal for gating write access to the bus. This signal is also used as a packet delimiter for bus read operations and therefore this implementation only supports P1 priority adapters. The user can replace the standard serial driver for port 1 with the J1708 driver by inserting the supported registry keys into the platform ADSLOAD.REG file.

Supported Registry Keys

`[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Serial]`

`"Dll"="J1708.dll"`

This is the name of J1708 driver file and must not be changed.

`"FriendlyName"="SAE J1708 Interface"`

This key is optional and has no effect on driver operation.

`"Prefix"="JSP"`
`"Index"=dword:1`

These two keys define the driver device name. The prefix cannot be changed. The index may be any number from 0-9.

`"XmtMID"=dword:120`

This key defines message ID for this platform. Refer to the J1708 specification for acceptable values.

`"ISTPriority"=dword:32`

This key sets the priority for the driver's interrupt service thread. This directly affects the performance of the driver and indirectly the system.

`"RcvPktEventName"="J1708RcvPacketEvent"`

This key defines a named event used by an application for determining when data is available from the bus.

Supported Public Functions

The following standard Win32 file I/O functions are supported by this driver. If a call to any of these functions returns a failure indication, call `GetLastError()` immediately to obtain a code to determine the reason.

```
hFile =    CreateFile(
              TEXT("JSP1:"),                 // Device name
              GENERIC_READ | GENERIC_WRITE,  // Access mode
              0,                             // Not used
              NULL,                          // Not used
              OPEN_EXISTING,                 // Always open existing
              0,                             // Not used
              NULL                           // Not used
           );
```

The return value from this function is a handle that is used to refer to the open J1708 device in subsequent file I/O function calls. Only one handle to this driver may be open at one time. This call will fail if the driver cannot synchronize with the J1708 bus. It does this by waiting for the bus to become idle before returning.

```
bResult =  DeviceIoControl(
              hFile,                         // Device handle from CreateFile()
              dwIoControlCode,               // See J1708.H for valid codes
              lpInBuffer,                    // Use of the following parameters
              nInBufferSize,                 // depends on the particular code
              lpOutBuffer,
              nOutBufferSize,
              lpBytesReturned,
              NULL                           // Not used
           );
```

An application can use this function to set driver behavior and retrieve driver information. The following codes can be used, which are defined in the J1708.H header file.

- IOCTL_J1708_DIS_RTL_MSG – Disable driver messages out serial port 3.
- IOCTL_J1708_ENA_RTL_MSG – Enable driver messages out serial port 3.
- IOCTL_J1708_GET_RCV_PKT_EVT_NAME – Named event that indicates when a packet is received.
- IOCTL_J1708_GET_DRIVER_STATUS – Internal use only.

Note that driver messages are off by default. The named event is to be used in conjunction with `ReadFile()` in order to prevent degradation of system performance. Instead of continually polling `ReadFile()` for any available data, the application can wait on the packet receive event before calling `ReadFile()`.

```
bResult =  PurgeComm(
              hFile,                         // Device handle from CreateFile()
              PURGE_RXCLEAR                  // Only this flag is supported
           );
```

This call is used to clear the receive buffer and discard any unwanted packets.

```
bResult =  ReadFile(
              hFile,                         // Device handle from CreateFile()
              lpBuffer,                      // Pointer to incoming data area
              nNumberOfBytesToRead,
              lpNumberOfBytesRead,
              NULL                           // Not used
           );
```

This call will copy the next available packet into the designated buffer area. This area should be large enough to hold the packet MID and the packet data. The maximum data size for a J1708 packet is 19 bytes and the MID size is 1 byte, therefore a 20 byte buffer should be sufficient for all J1708 packets. This value is constant and is always passed to the function using the `nNumberOfBytesToRead` parameter. The `lpNumberOfBytesRead` parameter will be zero after the function call if no data is available. Otherwise, it indicates the size of the retrieved packet. This function will indicate failure if a bad packet checksum is detected, the receive buffer is too small to contain the requested packet, or the size of the packet received exceeded the maximum size allowed by the J1708 standard.

```
bResult =  WriteFile(
              hFile,                         // Device handle from CreateFile()
              lpBuffer,                      // Pointer to outgoing data
              nNumberOfBytesToWrite,
              lpNumberOfBytesWritten,
              NULL                           // Not used
           );
```

A call to this function will form a J1708 packet and transmit it on the bus. The call will not return until the packet is transmitted or an error occurs. The amount of data to send is indicated in the `nNumberOfBytesToWrite` parameter and it cannot exceed 19 bytes as defined by `J1708_MAX_DATA_SIZE` in the J1708.H header file, otherwise the function fails. A packet is formed by pre-pending the data in `lpBuffer` with the transmitter MID character obtained from the registry and appending a one byte twos-complement checksum computed using the MID and the given data. This forms a maximum allowable packet on the J1708 bus of 21 bytes. Packets may be smaller if so desired. A successful will return `TRUE` in `bResult` and `lpNumberOfBytesWritten` will be equal to `nNumberOfBytesToWrite`.

```
bResult =  CloseHandle(
              hFile                          // Device handle from CreateFile()
           );
```

This call closes the driver by releasing the handle and other acquired resources allocated when the driver was opened.

Supported Constants

The constants below are defined in the J1708.H header file.

`J1708_MAX_DATA_SIZE`	- Maximum number of data bytes per J1708 packet
`J1708_MID_CHAR_SIZE`	- Number of bytes per MID character
`IOCTL_J1708_DIS_RTL_MSG`	- Turns off driver messages
`IOCTL_J1708_ENA_RTL_MSG`	- Turns on driver messages
`IOCTL_J1708_GET_RCV_PKT_EVT_NAME`	- Returns the name of the packet receive event in a string
`IOCTL_J1708_GET_DRIVER_STATUS`	- Internal use only
`J1708_ERROR_BAD_CHECKSUM`	- Received a bad packet
`J1708_ERROR_BUFFER_OVERFLOW`	- Packet size exceeded amount allowed by J1708 standard
`J1708_ERROR_PACKET_OVERRUN`	- A newly received packet over-wrote an old one
`J1708_ERROR_DRIVER_ALREADY_OPEN`	- Only one handle per driver
`J1708_ERROR_CANNOT_INIT_INTRPT`	- The UART interrupt could not be allocated to this driver
`J1708_ERROR_TX_PACKET_TOO_BIG`	- The application tried to send a packet that was too big
`J1708_ERROR_CANNOT_ENABLE_UART`	- The driver cannot synchronize with the J1708 bus
`J1708_ERROR_RX_BUFFER_TOO_SMALL`	- The application has not allocated enough space to hold a packet