Quest Technical Solutions Inc.
P.O. Box 71
Heidelberg Ontario Canada
(519) 699-5886 Fax (519) 699-5321

Author: Jim Stewart
Revision: 1.02

AN-X PBS Driver Development Document

1. Introduction

The AN-X Profibus Multislave (PBM) platform consists of several different pieces of hardware, firmware and software. Additionally, the AN-X platform provides several different utilities to aid in system construction.

This document is provided to supply the information on the PBM functionality; the common AN-X functions are documented elsewhere.

The AN-X Profibus Multislave Development software contains an example application that configures and monitors a Profibus DP network. The reader should fine the source code for this application (pbmslv) to be a very valuable companion to this documentation – reading C source code required.

1.1 PBM Platform Overview

The PBM platform consists of the following components:

- Hardware Configuration: FPGA configuration file that provides the Profibus slave hardware services. This is loaded using the /etc/rc.d/initanx.20anxapp start-up script.
- PBM Kernel Module: pbsdrv.o interfaces directly to the Profibus slave hardware and provides a basic DP multislave firmware module.
- User Software Application: This is not currently provided in the development platform. An example application is provided (pbmslv), but it is only a monitoring application. The developer will have to provide a useful user application.

The basic idea of the PBM platform is to provide a system that can have up to a total of 124 monitored and/or virtual slaves. A “monitored slave” is used to eavesdrop on a real physical slave and therefore does not transmit any traffic on the network. A “virtual slave” is used to provide an emulated DP slave to the Profibus Master. Virtual slaves look like real physical DP slaves on the network and respond (transmit) packets back to the master.

1.2 Multislave Hardware Memory

The FPGA configuration for the AN-X device is comprised of two memory areas – PGA registers and shared DRAM. The PGA registers are held in memory inside the FPGA and are used to control the operation of the multislave hardware services. The shared DRAM memory is a reserved 1 MB memory area at the top of physical RAM. It is not used by the OS and is shared between the multislave hardware (FPGA) and the kernel mode driver and/or user applications.

The AN-X platform has a PGA kernel mode driver that can provide a user mode application with a “memory mapped” pointer to the shared DRAM and PGA register memory (more details are provided later in this document).

The multislave hardware memory layout is defined in the tables to follow:

PGA Register Offset	Register Description
0x0000	LedCtl – controls the network LED
0x0004	PbsCtl – start or stop network and clear counters
0x0008	RxTnsHead – points to transaction buffer head
0x000C-0x000F	Reserved
0x0010	TxGood – number of good Tx packets (16 bit)
0x0012	RxGood – number of good Rx packets (16 bit)
0x0014	RxStpErr – stop bit error in received octet.
0x0015	RxParErr – parity error in received octet
0x0016	RxSdErr – start delimiter error in received packet
0x0017	RxRptErr
0x0018	RxUndErr – received packet too short
0x0019	RxFcsErr – checksum error in received packet
0x001a	RxEdErr – stop delimiter error in received packet
0x001b	RxOvrErr – received packet too long
0x001c	RxFitNde
0x001d	RxFitSap
0x001e	RxDupFrm
0x001f	RxTnsOvr – transaction buffer over flow
0x0020-0x03ff	Reserved
0x0400-0x05ff	NodeCtl – set monitored, virtual or ignored state of slaves (one byte register for each slave address)
0x0600-0x07ff	Reserved
0x0800-0x0bff	WdogTime – set the DP watchdog timeout (in ms) for each slave (one 32 bit register for each slave address)

Table 1 PGA Register Memory Layout

The PGA register space is a 64 kB address space and the register offsets show in the table above are addressed from the base memory pointer returned by the PGA kernel module.

Shared DRAM Offset	Area Description
0x00000-0x3ffff	PBS_CTL structures – arranged as a 3 dimensional array of PBS_CTL structures ([node][SAP][req_res])
0x40000-0x4ffff	PBS_RX_TNS structures – transaction based receive buffers.
0x50000-0x50fff	PBS_CTL default SAP structures – area used for the DP data exchange's default SAP. Two dimensional array of structures ([node] [req_res])
0x51000-0x7ffff	Reserved
0x80000-0xbffff	RxBuf – received data buffers. All data (not the layer 2 frame information) from received packets are put into this area.
0xc0000-0xfffff	TxBuf – transmit data buffers. All data (not the layer 2 frame information) to be sent out on the network comes from this area.

Table 2 Multislave Shared DRAM Memory Layout

The detailed definition of all the multislave hardware structures is contained in the anxpbslave.h header file.

The shared DRAM memory layout can be complex, in large part, this is due to the fact that both monitored and virtual slave devices are supported. The complexity of this memory area is handled by services provided by the PBM kernel module (see section, below).

The most import part of the shared DRAM memory layout is the RxBuf and TxBuf. These areas contain the DP diagnostics, configuration, parameter, input and output data areas.

2. PBM Kernel Module

The PBM kernel module interfaces with the AN-X multislave FPGA by handling hardware interrupts, processing the virtual slave DP state machine and offers configuration routines for the user space program. This section is a detailed discussion of the various services provided by the PBM kernel module.

2.1 DP Slave Configuration Services

As discussed earlier, the multislave hardware's shared memory interface can be somewhat complex. The key to understanding how the memory interface works is to remember how monitored slaves and virtual slaves work and combine this knowledge with the requirements of the Profibus DP protocol.

Profibus DP requires the following data areas:

- Diagnostics data
- Parameter data
- Check configuration data
- Inputs

- Outputs

The multislave shared memory interface (MSMI) is an OSI layer 2 implementation and therefore knows nothing of the DP protocol layers. The multislave kernel module helps to organize, configure and handle the layer 2 services to produce a maximum of 124 (1 to 125) DP slave devices. The tables to follow give a basic overview of how the MSMI is configured by the multislave kernel module:

RxBuf Allocation (0x80000 MSMI offset)	
Buffer Offset	**Area Description**
0x00000-0x077ff	Outputs – 30 kB block
0x07800-0x0efff	Chk. Configuration – 30 kB block
0x0f000-0x0167ff	Parameter data – 30 kB block
0x16800-0x1dfff	Monitored Inputs - 30 kB block (from monitored slaves)
0x1e000-0x257ff	Monitored Diagnostic data - 30 kB block (from monitored slaves)

Table 3 Rx Data Buffer Allocation for DP

TxBuf Allocation (0xc0000 MSMI offset)	
Buffer Offset	**Area Description**
0x00000-0x077ff	Inputs – 30 kB block *only for virtual slaves
0x07800-0x0efff	Chk. Configuration – 30 kB block *only for virtual slaves
0x0f000-0x0167ff	Diagnostic data – 30 kB block *only for virtual slaves

Table 4 Tx Buffer Allocation for DP

Any given user application must execute the following configuration steps:

1. Clear the current configuration (if any).
2. Set the basic network options (baud rate, tsdr, etc.).
3. For each required slave -- set monitored or virtual mode, size and offset of all DP areas (diagnostic, parameters,configuration, ip and/or op).
4. Start the bus (go on-line).
5. When leaving the application, stop the bus (go off-line).

The pbmslv application gives an example of the required configuration steps. The pbmslv_api.h header file provides all the structures and ioctl defines required to perform a configuration cycle.

In Step 3 above, the user application defines the maximum size and most importantly, the offset for each slave's data area in the predefined DP areas. The specific placement of any slave's data areas inside the allocated buffer spaces is left to the user application to allow the user to arrange data according to the user application's requirements (may be grouped contiguously or not...).

2.2 Runtime Operation

Again, the pbmslv application's source code is the best example of how to use the PBM kernel module to interact with the MSMI. There are two basic ways to access the data areas of the MSMI – using ioctl calls or a set of memory mapped pointers.

The pbmslv application uses ioctl calls for most of the monitoring functions. However, the pbmslv application does show code that initializes two memory mapped pointers – one for the PGA registers and one for the shared DRAM memory.

2.2.1. Memory Mapped Pointers

The AN-X platform provides a generic PGA kernel module (anxpga.o - /dev/anxpga) that is used to retrieve information about the PGA's available memory areas and memory mapping services. Note, the user must remember that access to the MSMI via memory mapped pointers is not atomic. Multiple threads or processes that access the MSMI can cause race conditions and unpredictable data values - only one “writer” process or thread is supported.

The user application can get a memory pointer to the PBS_REG structure and the PBS_MEM structure (anxpbslave.h). These pointers are used to access the MSMI memory areas directly from the user application.

The PBS_REG structure is very straight forward. For example, the following C code will access the NodeCtl register for the slave at station address 10:

```
PgaRegPtr->NodeCtl[10]
```

The PBS_MEM structure is much more complex. A user application would probably be best served by building a cache of data pointers at configuration time. Specifically, when the slave DP data areas are configured with the user defined offsets, a pointer to that slave's DP data area can be stored:

```
offset = RxBufAreaOfs + UserOfs

Ptr = &pPbsMem->RxBuf[offset]
```

where;

RxBufAreaOfs – is the predefined offset for the area of interest (0x16800 for monitored input data). See Table 3 Rx Data Buffer Allocation for DP for details.

UserOfs – is the user defined offset into the area of interest.

2.2.2. Monitoring Slave Status

The source code for the pbmslv application contains a function DisplayNodeList(). This function gives an example of how to get the status of any slave:

```
if(pPbsMem->DefCtl[node_cfg.slv_node][PBS_REQUEST].StrtDelim) printf (" OK");
```

The slave status indicates whether the slave is in DATA_EXCHANGE mode with the DP master. The value in StrtDelim will be 0x68 when in data exchange mode and zero when not. A test for a non-zero value is sufficient to indicate data exchange mode. This method is the same for virtual and monitored slaves.