Table 1: Cross Reference of Applicable Products

PRODUCT NAME	MANUFACTURER PART NUMBER	SMD #	DEVICE TYPE	INTERNAL PIC NUMBER
UT700 LEON	UT700	5962-13238	CAN Module	WQ03

1.0 Overview

The Controller Area Network (CAN) protocol was designed in 1983 as a vehicle standard to allow microcontrollers and devices to communicate with each other's applications without a host computer. It is a message based protocol that provides a priority driven paradigm to ensure the highest priority message is delivered on-time. Today, the CAN protocol has proliferated beyond the automotive industries.

The UT700 LEON 3FT SPARC V8 Microprocessor leverages this technology and provides its user with two identical CAN modules that are compliant to the CAN 2.0A (BasicCAN) and CAN 2.0B (PeliCAN). These two CAN modules can be configured into a combination of BasicCAN and PeliCAN allowing its users' applications the flexibility to communicate with other CAN applications both in CAN2.0A and CAN2.0B protocols within the same network.

The UT700 LEON 3FT CAN core is a derivative of the Philips SJA1000 and has a compatible register map with a few exceptions. These exceptions are described in the UT700 LEON 3FT Functional Manual, CAN chapter. Besides the UT700 LEON 3FT Functional Manual, the Philips SJA1000 data sheet can be used as an additional reference.

This application note is not intended to be a CAN protocol tutorial; it will elaborate the CAN features to enhance readers understanding of this application note. In the end, readers should be able to apply this newly acquired knowledge to develop a successful CAN project using the UT700 LEON 3FT SPARC V8 microprocessor.

Note: The description in this application note describes how to directly use the memory mapped interface of a specific hardware peripheral. If you are using an operating system such as RTEMS, Linux, and VxWorks or an environment such as BCC then it is recommended to use the infrastructure provided by those environments instead of accessing the peripheral directly as described in this application note.

2.0 Application Note Layout

This application note starts by providing a brief description of the CAN-associated hardware and communication requirements. In the CAN-associated hardware sections, this application note talks about the PHY, how it is connected to the UT700 LEON and the type of connectors used for CAN interface. The communication requirements section provides a connection diagram and a table showing the relative distance how the CAN nodes should be connected to ensure interoperability. Next, the CAN features such as priority, bitwise arbitration, baud rate and acceptance filter will be reviewed followed by the software programming of the CAN modules. The sections are appended as follows:

- CAN Associated Hardware
- CAN Connection Paradigm
- CAN Technologies
- CAN Programming

3.0 CAN Associated Hardware

Since this is a software centric Application Note, the discussion of hardware will be restricted to connectivity verification and testing purposes.

3.1 CAN PHY

The UT700 LEON 3FT CAN's modules can transmit and receive data up to 1 Mbits/s; it can also transmit at a lower bit rate. All the 1 Mbits/s CAN's PHYs are also capable of transmitting data at a lower bit rate; hence, it is preferred to use a 1 Mbits/s PHY for your design.

CAN modules provide simple connection to the CAN PHY via CAN_TXD and CAN_RXD pins as shown in **Table 2**. If a transmission error occurs, these are the pins to check first. Also, refer to the UT700 LEON 3FT Functional Manual for the type of package you are using for its pin numbers.

Table 2: CAN Module TX/RX Signal Pin

UT700 CAN Module 0	UT700 CAN Module 1	Description
CAN_TXD[0]	CAN_TXD[1]	Transmit Signal Pin
CAN_RXD[0]	CAN_RXD[1]	Receive Signal Pin

3.2 CAN Interface Connectors

There are several types of connectors used for CAN interface such as the 9-pin male D-sub, 10-pin header, RJ-Style, 7-pin open style, etc. Among these connectors, the 9-pin male D-sub connector is most widely used and **Table 3** shows the pinout. The LEAP controller boards provided by Cobham also used the 9-pin male D-sub connector.

Table 3: 9-Pin Male DSUB Connector Pinout for CAN Bus

Pin #	Signal names	Signal Description
1	Reserved	Upgrade Path
2	CAN_L	Dominant Low
3	CAN_GND	Ground
4	Reserved	Upgrade Path
5	CAN_SHLD	Shield, Optional
6	GND	Ground, Optional
7	CAN_H	Dominant High
8	Reserved	Upgrade Path
9	CAN_V+	Power, Optional

3.3 CAN Connection Paradigm

The CAN network is a party-line connection that allows many CAN nodes to be connected. The logical number of CAN nodes on a CAN bus is limited to the size of the CAN ID. In CAN 2.0B extended mode, more nodes can be connected. In order for the CAN bus to function properly, termination resistors are used to impede reflections on the bus. To determine the correct value of the termination resistors, check the impedance of the cable and match the resistor to it. For a CAN bus cable with a 120-ohm line impedance, you should use a 120-ohm resistor. In a high-speed bus like this, it will require termination at the two ends of the bus.

Commonly, you would place the Master at one end of the bus. It is also possible to have the master connected in the middle with no termination, but with the termination at the nodes at the end of the bus line. **Figure 1** shows the CAN nodes connection paradigm and the relationship between nodes and their respective maximum distance.

Table 4: CAN Bus Speed and Cable Length

Bus Speed	Bus Length (L)	Cable Stub Length (l)	Node Distance (d)
1 Mbit/s	40 meters/131 feet	0.3 meters/1 foot	40 meters/131 feet
500Kbits/s	100 meters/328 feet	0.3 meters/1 foot	100 meters/328 feet
100Kbits/s	500 meters/1640 feet	0.3 meters/1 foot	500 meters/1640 feet
50 Kbits/s	1000 meters/3280 feet	0.3 meters/1 foot	1000 meters/3280 feet

Legend:

L: Maximum Bus Length
l: Maximum Cable Stub Length
d: Maximum Node Distance

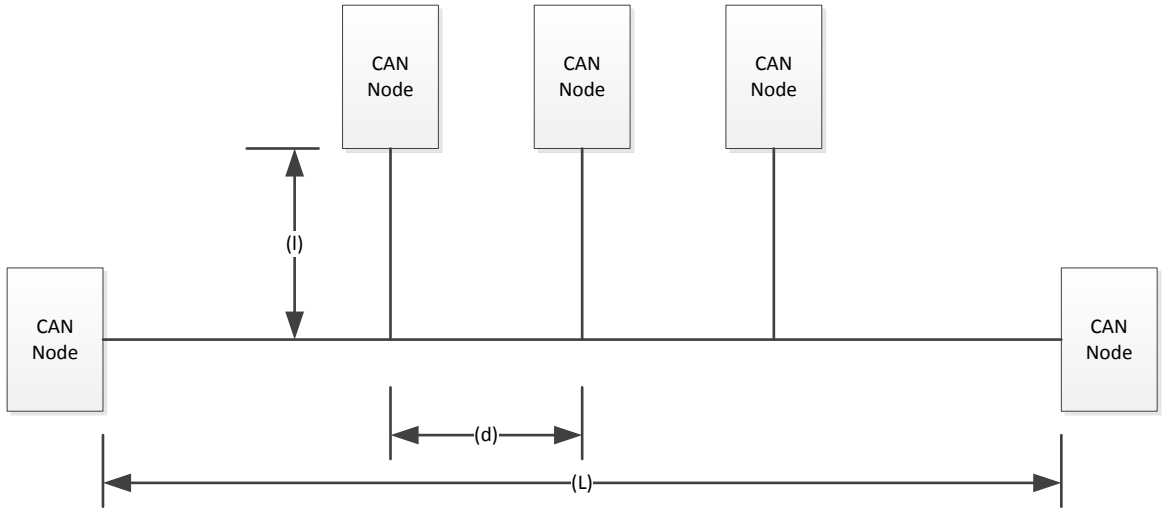

Figure 1: CAN Nodes Connection Paradigm

The distances between the nodes are subjected to multiple variables such as the end nodes, the sub-nodes, and the bus speeds. The CAN bus maximum length is as shown in **Table 4**.

4.0 CAN Technologies

In this section, we will explain the CAN's priority and bitwise arbitration work, how to calculate CAN baud rate and how acceptance filters work.

4.1 CAN Priority and Bitwise Arbitration

CAN protocol does not have a priority field. It uses the message ID and non-destructive bitwise arbitration to determine which message has the highest priority. When the CAN ID bits are cleared, dominant bits, it has the highest priority and when the bits are set, recessive bits, it has the lowest parity.

If all challenging message ID signals on the bus are synchronized and connected in a Wire-AND circuit then the low ID's bit signal, the dominant bit, overwrites the high ID's bit signal, the recessive bit. The high priority message ID overwrites the low priority message ID causing all low priority nodes to terminate its transmission. This is how non-destructive bitwise arbitration works with the cooperation of every node on the bus.

4.2 CAN Baud Rate

Before we calculate the CAN baud rate, let us familiarized ourselves with the two registers that are used for this purpose, the Bus Timing Register 0 and 1, BTR0 (**Table 5**) and BTR1 (**Table 6**) respectively. These fields collectively are used to calculate the CAN baud rate as shown in **Equation 1** and **Equation 2**. More information on these registers can be found in the UT700 LEON Function Manual and **Table 7**.

Table 5: Bus Timing Register 0 (BTR0), Offset 6

Bit#	7	6	5	4	3	2	1	0
R W	SJW		BRP					
Reset	--		--					

Table 6: Bus Timing Register 1 (BTR1), Offset 7

Bit#	7	6	5	4	3	2	1	0
R W	SAM	TSEG2			TSEG1			
Reset	--	--			--			

In order to calculate the CAN baud rate, know the system clock provided to the UT700 LEON 3FT. For this example, the system clock (tclk) is 50MHz. To divide the system clock down, the BRP field in BTR0 register is used as shown in **Equation 1** as follows:

Equation 1: CAN baud rate Equation

$$t_{scl} = 2 * t_{clk} * (BRP + 1)$$

Equation 2: CAN baud rate Equation

$$CAN_{BR} = [(TSEG1 + 1) + (TSEG1 + 1) + 1] * t_{scl}$$

Example: The required baud rate is 250Kbits/s and the system clock is 50MHz. **Table 7** provides a brief description of the registers fields.

By using the **Equation 2** and let BRP = 3, TSEG1 = 15 and TSEG2 = 7, the baud rate is equal to 250Kbits/s.

$$CAN_{BR} = [16 + 8 + 1] * t_{scl} = 250\ Kbits/s$$

Table 7: Bus Timing Register Description

BIT	NAME	DESCRIPTION
BTR0[7:6]	SJW	Synchronous jump width
BTR0[5:0]	BRP	Baud rate pre-scaler
BTR1[7]	SAM	0: Single sample point, 1: Sample three times
BTR1[6:4]	TSEG2	Time segment 2
BTR1[3:0]	TSEG1	Time segment 1

4.3 CAN Acceptance Filters

The Acceptance filters effectively filter off messages or a group of messages not valid for a certain node to prevent from being stored in the receiving FIFO; hence, reducing the processing loads of the host controller.

Both BasicCAN and PeliCAN modes support acceptance filtering. In BasicCAN mode, the acceptance filter consists of an 8-bit Acceptance Code Register (ACR) and Acceptance Mask Register (AMR). The PeliCAN, there are four 8-bit Acceptance Code Registers and Acceptance Mask Registers. Their usages are elaborated in the UT700 LEON Functional Manual. In order to access the Acceptance Filters, the CAN module must enter into Reset Mode.

Note: Enter Reset Mode (Mode register at offset 0) to access Acceptance Filters

5.0 CAN Programming

In the previous sections, we learned which signals on the UT700 LEON 3FT are used and the pins on the 9-pin male D-sub socket to probe to verify communication issues. We learned the physical constraint of the CAN's network connection and the CAN's priority. We also learned how to calculate the CAN transmission bit rate. By now, we have enough essential information to proceed to our next subject, the programming of the CAN modules.

The following sections of this application note highlights and provides programming examples of the CAN's module configuration and setting to enable the CAN module to perform the following functions. The order is as follows:

- How to select BasicCAN or PeliCAN
- How to enter Reset Mode
- How to select Acceptance Filter
- How to set the baud rate
- How to enable interrupt
- How to read/write CAN messages

5.1 Select BasicCAN or PeliCAN

Selection of BasicCAN or PeliCAN operational mode can be done in both CAN's operational or reset mode. The mode operation selection is performed through CM bit in the Clock Divider Register. Clear the CM bit to select BasicCAN and set the CM bit to select PeliCAN. **Table 8** shows the Clock Divider Register format and how BasicCAN and PeliCAN are selected as follows:

- BasicCAN mode, Clock Divider Register Offset 31

```
CANX.CD.B.CM = 0;  // enter BasicCAN Mode
```

- PeliCAN mode, Clock Divider Register Offset 31

```
CANX.CD.B.CM = 1;  // enter PeliCAN mode
```

Please see header file in Appendix A.

Table 8: CAN Clock Divider Register at Offset 31

Legend:

CM: CAN Mode

	7	6	5	4	3	2	1	0
R	CM							
W								
Reset	0	--						

5.2 Enter Reset Mode

There are some registers in the CAN module that can only be accessed in the Reset Mode. Appended are the codes to switch into CAN's Reset mode as follows:

- BasicCAN Mode, Control Register Offset 0

```
CANX.CR.B.RR = 1;       // enter Reset Mode (Reset Request)
```

- PeliCAN Mode, Mode Register Offset 0

```
CANX.MOD.B.RM = 1;      // enter Reset Mode
```

Table 9 and **Table 10** show the Control Register and Mode format.

Table 9: BasicCAN Control Register at Offset 0

Legend:

OIE: Overrun Interrupt Enable
EIE: Error Interrupt Enable
TIE: Transmit Interrupt Enable
RIE: Receive Interrupt Enable
RR: Reset Request

	7	6	5	4	3	2	1	0
R				OIE	EIE	TIE	RIE	RR
W								
Reset	--			0	0	0	0	0

Table 10: PeliCAN Mode Register at Offset 0

Legend:

AFM: Acceptance Filter Mode
STM: Self-Test Mode
LOM: Listen-Only Mode
RM: Reset Mode

	7	6	5	4	3	2	1	0
R					AFM	STM	LOM	RM
W								
Reset	--				0	0	0	0

5.3 Select Acceptance Filter

Since the Acceptance Filter can only be accessed in Reset mode, switch the CAN into Reset Mode as shown in Section **5.2**.

After the CAN modules have entered Reset Mode, the Acceptance Filters are accessible. This is how ACR and AMR can be programmed.

- In BasicCAN Mode, Acceptance Filter Register Offset 4 and 5

```
CANX.CR.R.ACR = 0x01;   // Acceptance Code Register
CANX.CR.R.AMR = 0x00;   // Acceptance Mask Register
```

- In PeliCAN Mode, Acceptance Filter Register Offset 16 to 23

```
CANX.FIFO.RM.ACR0 = 0x00;     // Acceptance Code 0 Register
CANX.FIFO.RM.ACR1 = 0x00;     // Acceptance Code 1 Register
CANX.FIFO.RM.ACR2 = 0x00;     // Acceptance Code 2 Register
CANX.FIFO.RM.ACR3 = 0x10;     // Acceptance Code 3 Register
CANX.FIFO.RM.AMR0 = 0x00;     // Acceptance Mask 0 Register
CANX.FIFO.RM.AMR1 = 0x00;     // Acceptance Mask 1 Register
CANX.FIFO.RM.AMR2 = 0x00;     // Acceptance Mask 2 Register
CANX.FIFO.RM.AMR3 = 0x10;     // Acceptance Mask 3 Register
```

Note: Must be in Reset Mode to access Acceptance Filters.

Also, refer to UT700 LEON 3FT Functional Manual CAN chapter for more information about ACR and AMR.

5.4 Set the Baud Rate

In section **4.2**, we explained how to calculate CAN baud rate. Then program those parameters into their respective registers as shown in **Table 5** and **Table 6**.

```
CANX.BTR0.B.SJW   = 0;  // Sync Jump width + 1
CANX.BTR0.B.BRP   = 3;  // Baud Rate Prescaler
CANX.BTR1.B.SAM   = 0;  // Number of bus samples + 1
CANX.BTR1.B.TSEG2 = 8;  // Time Segment 2
CANX.BTR1.B.TSEG1 = 15; // Time Segment 1
```

Note: Must be in Reset Mode to access the BasicCAN BTR0 and BTR1.

5.5 Enable Interrupt

The CAN's interrupts are enabled through the BasicCAN Control Register and PeliCAN Interrupt Enable Register as shown in **Table 9**. The respective interrupts can be enabled as follows:

- BasicCAN, Control Register Offset 0

```
CANX.CR.B.OIE = 1;      // Overrun Interrupt Enable
CANX.CR.B.EIE = 1;      // Error Interrupt Enable
CANX.CR.B.TIE = 1;      // Transmit Interrupt Enable
CANX.CR.B.RIE = 1;      // Receive Interrupt Enable
```

- PeliCAN, Interrupt Enable Register Offset 4

```
CANX.IER.B.BEI = 1;     // Bus Error Interrupt
CANX.IER.B.ALI = 1;     // Arbitration Lost Interrupt
CANX.IER.B.EPI = 1;     // Error Passive Interrupt
CANX.IER.B.DOI = 1;     // Overrun Interrupt Enable
CANX.IER.B.EWI = 1;     // Error Interrupt Enable
CANX.IER.B.TI = 1;      // Transmit Interrupt Enable
CANX.IER.B.RI = 1;      // Receive Interrupt Enable
```

Note: Make sure you have setup the respective Interrupt Routine before enable these interrupt bits.

5.6 Read/Write CAN Messages

- Read CAN Message

```
while (CANX.SR.B.RBS==0);     // wait for data
CAN_ID1 = CANX.TID1.R;        // read the ID
CAN_ID2 = CANX.TID2.R;        //

for (x=0; x<8; x++)           // read data from Read FIFO
{
      message[x] = CANX.RX_FIFO[x];
}
```

- Write CAN Message

```
CANX.TID1.R = 0x01;     // write the ID
```

```
CANX.TID2.R = 0x00;      //

for (x=0; x<8; x++)      // write data to Write FIFO
{
      CANX.TX_FIFO[x] = message[x];
}

CANX.CMR.B.TR = 1;       // Transmission Request
```

6.0 Summary and Conclusion

For more information about our UT700 LEON 3FT/SPARC™ V8 Microprocessor and other products please visit our website, www.cobham.com/HiRel or contact your area sales representative.

7.0 Appendix A

The header files are designed for this application note purpose only.

```
/*********************************************************************************************\
* MODULE : CAN-2.0 Interface, BasicCAN mode(Address 0xFFF20000 and 0xFFF20100)          *
\*********************************************************************************************/

#define CAN_BAS1_ADDR                   0xFFF20000
#define CAN_BAS2_ADDR                   0xFFF20100

        struct CAN_BAS_TAG {                                    // Table 13.1 Basic CAN Address Allocation
                union {
                        vuint8_t R;                             // 13.3.2 Control register - R/W
                        struct {
                                vuint8_t RES75          :3;     // Reserved
                                vuint8_t OIE            :1;     // Overrun Interrupt Enable
                                vuint8_t EIE            :1;     // Error Interrupt Enable
                                vuint8_t TIE            :1;     // Transmit Interrupt Enable
                                vuint8_t RIE            :1;     // Receive Interrupt Enable
                                vuint8_t RR             :1;     // Reset request
                        } B;
                } CR;
                union {
                        vuint8_t R;                             // 13.3.3 Command register - WO
                        struct {
                                vuint8_t RES74          :4;     // Reserved
                                vuint8_t CDO            :1;     // Clear Data Overrun
                                vuint8_t RRB            :1;     // Release Receive Buffer
                                vuint8_t AT             :1;     // Abort Transmission
                                vuint8_t TR             :1;     // Transmission Request
                        } B;
                } CMR;
                union {
                        vuint8_t R;                             // 13.3.4 Status register - RO
                        struct {
                                vuint8_t BS             :1;     // Bus Status
                                vuint8_t ES             :1;     // Error Status
                                vuint8_t TS             :1;     // Transmit Status
                                vuint8_t RS             :1;     // Receive Status
                                vuint8_t TC             :1;     // Transmission Complete
                                vuint8_t TBS            :1;     // Transmit Buffer Status
                                vuint8_t DOS            :1;     // Data Overrun Status
                                vuint8_t RBS            :1;     // Receive Buffer Status
                        } B;
                } SR;
                union {
                        vuint8_t R;                             // 13.3.5 Interrupt register - RO
                        struct {
                                vuint8_t RES74          :4;     // Reserved
                                vuint8_t DOI            :1;     // Data Overrun Interrupt
```

```
                vuint8_t EI             :1;     // Error Interrupt
                vuint8_t TI             :1;     // Transmit Interrupt
                vuint8_t RI             :1;     // Receive Interrupt
        } B;
} IR;

union {
        vuint8_t R;                             // Acceptance Code - RW
        struct {
                vuint8_t D              :8;     // OM:0xFF  RM: Acceptance Code
        } B;
} ACR;
union {
        vuint8_t R;                             // Acceptance Mask - RW
        struct {
                vuint8_t D              :8;     // OM:0xFF  RM: Acceptance Mask
        } B;
} AMR;

union {
        vuint8_t R;                             // 13.5.2 Bus timing 0 - RW
        struct {
                vuint8_t SJW            :2;     // Synchronization jump width
                vuint8_t BRP            :6;     // Baud rate prescaler
        } B;
} BTR0;
union {
        vuint8_t R;                             // 13.5.3 Bus timing 1 - RW
        struct {
                vuint8_t SAM            :1;     // 1 - bus is sampled three times, 0 - single sample point
                vuint8_t TSEG2          :3;     // Time segment 2
                vuint8_t TSEG1          :4;     // Time segment 1
        } B;
} BTR1;

vuint8_t padding[2];

union {
        vuint8_t R;                             // OM:TX ID1  RM:0xFF
        struct {
                vuint8_t ID             :8;     // CAN ID
        } B;
} TID1;
union {
        vuint8_t R;                             // OM:TX ID2, rtr, dlc  RM:0xFF
        struct {
                vuint8_t ID             :3;     // CAN ID
                vuint8_t RTR            :1;     // RTR
                vuint8_t DLC            :4;     // DLC
        } B;
} TID2;

vuint8_t TX_FIFO[8];                            // TX Data

union {
        vuint8_t R;                             // OM:RX ID1  RM:0xFF
```

```
                struct {
                        vuint8_t ID             :8;     // CAN ID
                } B;
        } RID1;
        union {
                vuint8_t R;                             // OM:RX ID2,  rtr,  dlc  RM:0xFF
                struct {
                        vuint8_t ID             :3;     // CAN ID
                        vuint8_t RTR            :1;     // RTR
                        vuint8_t DLC            :4;     // DLC
                } B;
        } RID2;

        vuint8_t RX_FIFO[8];                            // RX Data
        vuint8_t padding30;

        union {
                vuint8_t R;                             // OM: Clock Divider RM: Clock Divider
                struct {
                        vuint8_t CM             :1;     // 1 - PeliCAN, 0 - BasicCAN
                        vuint8_t RES60          :7;     // Reserved
                } B;
        } CD;
} PACKED;


/*****************************************************************************************\
* MODULE : CAN-2.0 Interface, PeliCAN mode(Address 0xFFF20000 and 0xFFF20100)           *
\*****************************************************************************************/
#define CAN_PELI1_ADDR                  0xFFF20000
#define CAN_PELI2_ADDR                  0xFFF20100
struct CAN_PELI_TAG {                                   // Table 13.7 PeliCAN Address Allocation
        union {
                vuint8_t R;                             // 13.4.2 Mode Register
                struct {
                        vuint8_t RES74          :4;     // Reserved
                        vuint8_t AFM            :1;     // Acceptance Filter Mode
                        vuint8_t STM            :1;     // Self-Test Mode
                        vuint8_t LOM            :1;     // Listen-Only Mode
                        vuint8_t RM             :1;     // Reset Mode
                } B;
        } MOD;
        union {
                vuint8_t R;                             // 13.4.3 Command register
                struct {
                        vuint8_t RES75          :3;     // Reserved
                        vuint8_t SRR            :1;     // Self Reception Request
                        vuint8_t CDO            :1;     // Clear Data Overrun
                        vuint8_t RRB            :1;     // Release Receive Buffer
                        vuint8_t AT             :1;     // Abort Transmission
                        vuint8_t TR             :1;     // Transmission Request
                } B;
        } CMR;
        union {
                vuint8_t R;                             // 13.4.4 Status register
                struct {
```

```
                vuint8_t BS             :1;     // Bus Status
                vuint8_t ES             :1;     // Error Status
                vuint8_t TS             :1;     // Transmit Status
                vuint8_t RS             :1;     // Receive Status
                vuint8_t TC             :1;     // Transmission Complete
                vuint8_t TBS            :1;     // Transmit Buffer Status
                vuint8_t DOS            :1;     // Data Overrun Status
                vuint8_t RBS            :1;     // Receive Buffer Status
        } B;
} SR;
union {
        vuint8_t R;                             // 13.3.5 Interrupt register
        struct {
                vuint8_t BEI            :4;     // Bus Error Interrupt
                vuint8_t ALI            :1;     // Arbitration Lost Interrupt
                vuint8_t EPI            :1;     // Error Passive Interrupt
                vuint8_t RES4           :4;     // Reserved
                vuint8_t DOI            :1;     // Data Overrun Interrupt
                vuint8_t EWI            :1;     // Error Warning Interrupt
                vuint8_t TI             :1;     // Transmit Interrupt
                vuint8_t RI             :1;     // Receive Interrupt
        } B;
} IR;
union {
        vuint8_t R;                             // 13.4.6 Interrupt enable register
        struct {
                vuint8_t BEI            :4;     // Bus Error Interrupt
                vuint8_t ALI            :1;     // Arbitration Lost Interrupt
                vuint8_t EPI            :1;     // Error Passive Interrupt
                vuint8_t RES4           :4;     // Reserved
                vuint8_t DOI            :1;     // Data Overrun Interrupt
                vuint8_t EWI            :1;     // Error Warning Interrupt
                vuint8_t TI             :1;     // Transmit Interrupt
                vuint8_t RI             :1;     // Receive Interrupt
        } B;
} IER;

vuint8_t padding5;                              // Reserved
union {
        vuint8_t R;                             // 13.5.2 Bus timing 0
        struct {
                vuint8_t SJW            :2;     // Synchronization jump width
                vuint8_t BRP            :6;     // Baud rate prescaler
        } B;
} BTR0;
union {
        vuint8_t R;                             // 13.5.3 Bus timing 1
        struct {
                vuint8_t SAM            :1;     // 1 - bus is sampled three times, 0 - single sample point
                vuint8_t TSEG2          :3;     // Time segment 2
                vuint8_t TSEG1          :4;     // Time segment 1
        } B;
} BTR1;

vuint8_t padding0810[3];                        // 0x00
```

```
union {
        vuint8_t R;                             // 13.4.7 Arbitration lost capture register
        struct {
                vuint8_t RES75          :3;     // Reserved
                vuint8_t BN             :5;     // Bit number
        } B;
} ALC;
union {
        vuint8_t R;                             // 13.4.8 Error code capture register
        struct {
                vuint8_t EC             :2;     // Error code
                vuint8_t DIR            :1;     // Direction
                vuint8_t SEG            :5;     // Segment
        } B;
} ECC;

vuint8_t ERR_WARNING_LIMIT_REG;     // 13.4.9 Error warning limit register (96)
vuint8_t RX_ERR_CNT_REG;            // 13.4.10 RX error counter register (address 14)
vuint8_t TX_ERR_CNT_REG;            // 13.4.11 TX error counter register (address 15)

union {
        vuint8_t R;                             // Table 13.18 Description of TX Frame Information,
Address 16
        struct {
                vuint8_t FF             :1;     // frame format,1 = EFF, 0 = SFF
                vuint8_t RTR            :1;     // Remote Transmission Request
                vuint8_t RES54          :2;     // Reserved
                vuint8_t DCL            :4;     // Data Length Code
        } B;
} TRFI;
union {
        vuint8_t R;                             // Table 13.19 Description of TX Identifier 1, Address 17
        struct {
                vuint8_t ID             :8;     // CAN ID
        } B;
} TRID1;
union {
        vuint8_t R;                             // Table 13.20 Description of TX Identifier 2, Address 18
        struct {
                vuint8_t ID             :8;     // CAN ID
        } B;
} TRID2;

union {
        struct {
                vuint8_t DATA[8];               // read/write FIFO
                vuint8_t res[2];                // not used
        } OM;                                   // Operating Mode
        struct {
                vuint8_t TRID3;
                vuint8_t TRID4;
                vuint8_t DATA[8];               // read/write FIFO
        } EM;                                   // Operating Mode, Extended Mode
        struct {
                vuint8_t ACR0;                  // Acceptance Code Register
                vuint8_t ACR1;                  //
```

```
                        vuint8_t ACR2;          //
                        vuint8_t ACR3;          //
                        vuint8_t AMR0;          // Acceptance Mask Register
                        vuint8_t AMR1;          //
                        vuint8_t AMR2;          //
                        vuint8_t AMR3;          //
                        vuint8_t res[2];        // not used
                } RM;                           // Reset Mode
        } FIFO;

        vuint8_t RX_MESS_CNT;           // 13.4.15 RX Message Counter
        vuint8_t padding30;
        union {
                vuint8_t R;                     // OM: Clock Divider RM: Clock Divider
                struct {
                        vuint8_t MODE   :1; // 1 - PeliCAN, 0 - BasicCAN
                        vuint8_t RES60  :7; // Reserved
                } B;
        } CD;
} PACKED;
```

8.0 REVISION HISTORY

Date	Rev. #	Author	Change Description
08/18/2016	1.0.0	MTS	Initial Release
09/29/2017	1.0.1	MTS	Add note, page 1

Cobham Semiconductor Solutions

This product is controlled for export under the U.S. Department of Commerce (DoC). A license may be required prior to the export of this product from the United States.

Cobham Semiconductor Solutions
4350 Centennial Blvd
Colorado Springs, CO 80907

E: info-ams@aeroflex.com
T: 800 645 8862