XAPP1056 (v1.0) April 25, 2008

Reference System: CAN Using the XA Automotive ECU Development Kit

Abstract

This application note describes a reference system to test the operation of Xilinx Platform Studio (XPS) Controller Area Network (CAN) cores that are connected to each other using CAN PHYs on the XA3S1600E board, which is part of the XA Automotive Development Kit. The reference system contains two XPS CAN cores and other peripherals. This document describes how to build a system that contains two XPS CAN cores, the port connections between cores, and the clocking for the XPS CAN and other XPS peripherals. A basic description of the software application that is provided with the reference system is also given.

This reference system is targeted for the Xilinx XA3S1600E board.

Included Systems

The reference system for the Xilinx XA3S1600E Evaluation Board is included with this application note. The reference system is available at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=109471

Introduction

The Controller Area Network (CAN) is a serial communications protocol used to efficiently support distributed real-time control. In the normal operation mode of operation, a CAN controller transmits data onto the bus and other CAN nodes connected to the bus can receive it. This application note describes how to set up a system with two XPS CAN cores that communicate with each other using the CAN PHYs on the board. XPS CAN 0 node transmits a message (Standard Data Frame) and XPS CAN 1 node receives this message. The received message is compared with the transmitted message to confirm the basic transmit/receive functionality of the XPS CAN core. You can also monitor the CAN bus and transmit/receive CAN messages.

Hardware and Software Requirements

The hardware and software requirements are:

- XA Automotive ECU Development Kit
  - Xilinx XA3S1600E Evaluation Board
  - Xilinx Platform USB cable or Parallel IV programming cable
  - RS232 serial cable and serial communication utility (HyperTerminal)
- Xilinx Platform Studio 10.1.01 or above
- Xilinx Integrated Software Environment (ISE®) v10.1 with Service Pack 1 or above
- 100 Ohm Resistor for CAN bus termination

Note: The reference system has been built with EDK v10.1.01 and ISE v10.1. The latest Xilinx tool versions can be obtained from the Xilinx download page:

http://www.xilinx.com/support/download/index.htm

Reference System Specifics

This reference system is built on a Xilinx Spartan®-3E XA3S1600E board. The components of this system are listed below.

- A MicroBlaze™ processor

- An on-chip hardware debug module
- A 8Kb block RAM
- The XPS MCH External Memory controller (xps_mch_emc)
- A DCM (which generates the 50 MHz PLB CLK, 24 MHz CAN CLK)
- The XPS UART16550 core
- The XPS Interrupt Controller core
- Two XPS CAN cores

The system uses an external crystal oscillator to generate a clock that acts as an input to the DCM. The Block Diagram of this system is shown in Figure 1. The address map for this system is provided in Table 1.

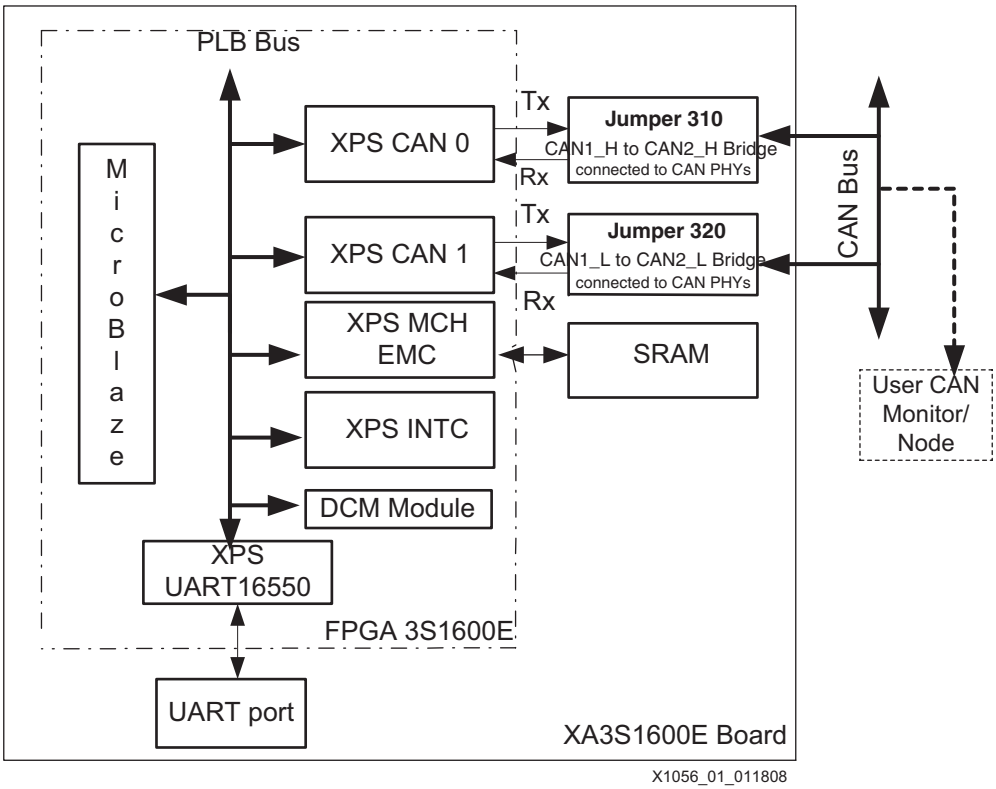

Figure 1: **Reference System Block Diagram**

Note: Jumper connections to the CAN PHYs can be found in XA3S1600E board schematic.

Table 1: **Reference System Address Map**

Peripheral	Instance	Base Address	High Address
lmb_bram_if_cntlr	dlmb_cntlr	0x00000000	0x00001FFF
lmb_bram_if_cntlr	ilmb_cntlr	0x00000000	0x00001FFF
xps_uart16550	xps_uart16550_0	0x00010000	0x0001FFFF
mdm	debug_module	0x00020000	0x0002FFFF
xps_mch_emc	xps_mch_emc_0	0x30000000	0x3003FFFF
xps_intc	xps_intc_0	0x80020000	0x8002FFFF

Table 1: **Reference System Address Map *(Continued)***

Peripheral	Instance	Base Address	High Address
xps_can	xps_can_0	0x80030000	0x800300FF
xps_can	xps_can_1	0x80040000	0x800400FF

Configuring a System with Two XPS Can Cores

Each of the XPS CAN cores is configured for a FIFO depth of 2 and uses a single acceptance filter. A DCM is connected as an input clock to the output of a crystal oscillator. The SRAM on the board stores the software application. The interrupts of the XPS CAN cores are connected to the interrupt line of the XPS INTC. The XPS UART16550 displays HyperTerminal messages. This section describes how to set up the EDK system with these cores.

Ports and Parameters for the Cores

All the cores of the system (except MicroBlaze) are slaves on the PLBv46 bus. The IP2Bus_IntrEvent ports of XPS CAN 0 node (CAN_Interrupt_0 signal) and XPS CAN 1 node (CAN_Interrupt_1 signal) are connected to the *Intr* port of the interrupt controller. The interrupt request port of the interrupt controller, *Irq*, is connected to the interrupt port of the MicroBlaze (microblaze_0_INTERRUPT).

Figure 2 and Figure 3 show the parameter settings for the XPS CAN cores. These parameters correspond to the smallest FIFO depth, and the minimum number of acceptance filters for the XPS CAN core. The RX and TX FIFO depths should always have a value of 2^n, where n is between 1 and 6. The maximum configurable FIFO depth is 64. The parameters for FIFO depth cannot be set to 0. There are four acceptance filters for the XPS CAN core; therefore, the parameter for the number of acceptance filters is assigned a value between zero (no acceptance filters used) and four (all acceptance filters used).

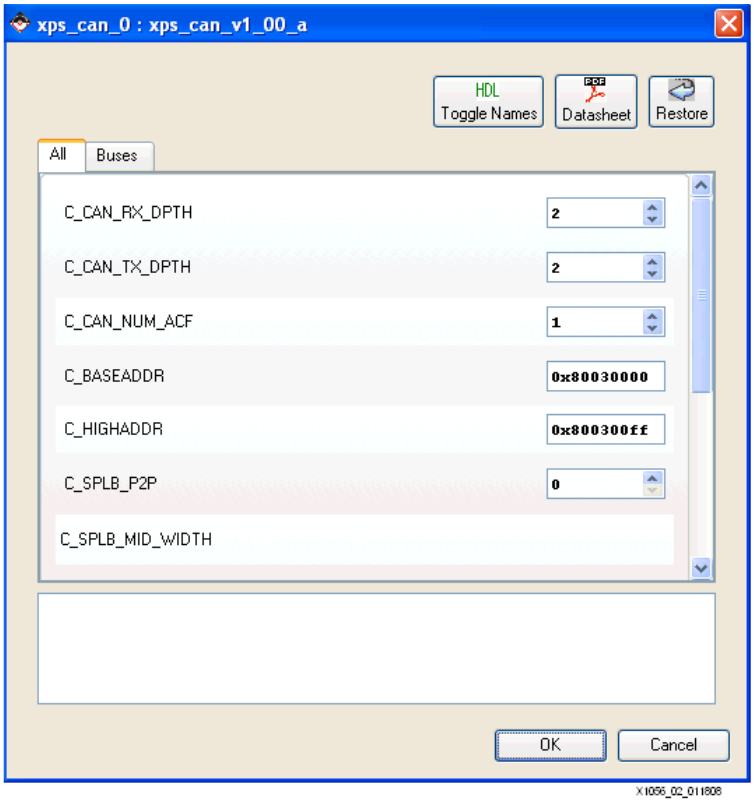

Figure 2: **Parameter Settings for XPS CAN 0 node**

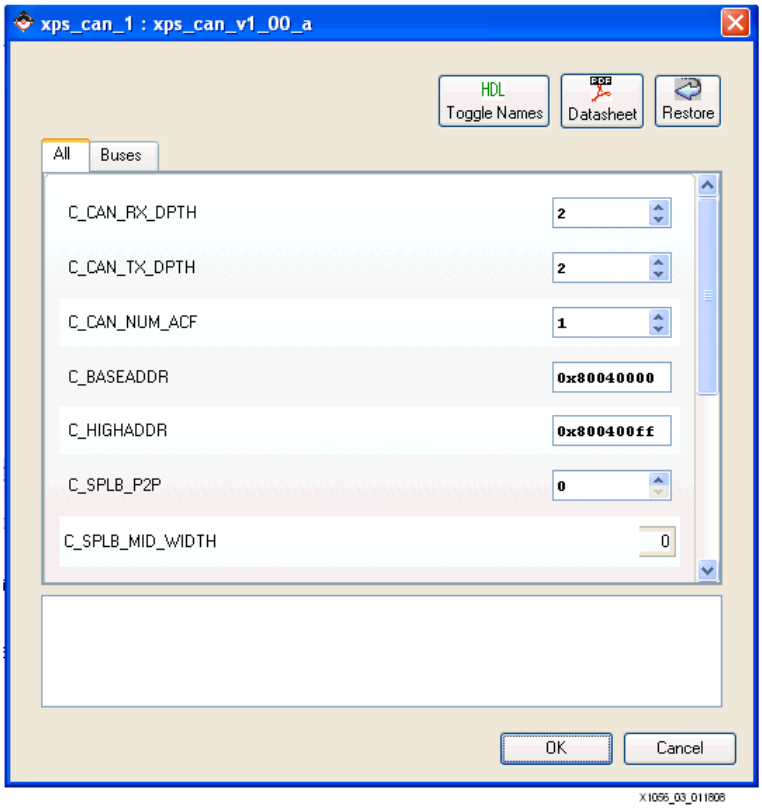

Figure 3: **Parameter Settings for XPS CAN 1 node**

Clocking for Reference System

The XA3S1600E board provides a 24 MHz clock. This clock is fed into a DCM, which generates the 50 MHz and 24 MHz clocks. The 50 MHz clock (which is the PLB clock) is fed into the processor and the other cores in the system.

The 24 MHz output of the DCM is fed to the CAN_CLK pins of the two XPS CAN cores (signal xps_can_0_CAN_CLK). The CAN CLK can have a frequency range of 8–24 MHz.

CAN PHY Connections

The CAN1_H (high line output of the CAN PHY connected to XPS CAN 0 node), CAN2_H (high line output of the CAN PHY connected to XPS CAN 1 node) lines coming out of the CAN PHYs at Jumper JP310 on the XA3S1600E Board must be connected to each other.

The CAN1_L (low line output of the CAN PHY connected to XPS CAN 0 node), CAN2_L (low line output of the CAN PHY connected to XPS CAN 1 node) lines coming out of the CAN PHYs at Jumper JP320 on the XA3S1600E Board must be connected to each other.

This ensures that the two XPS CAN cores on the XA3S1600E Board are connected to each other over PHYs. Place a termination resistor (100 Ohms) between the CAN2_L and CAN2_H lines that come out of the D-Sub 44-pin male connector (X300) on the XA3S1600E board at pin numbers 5 and 21, respectively. You can monitor the CAN bus by connecting the High and Low lines of the external CAN node to the CAN2_H and CAN2_L lines that come out of the D-Sub 44-pin male connector (X300) on the XA3S1600E board at pin numbers 5 and 21, as shown in Figure 4. Table 2 lists the CAN PHY jumper pin connections.

Table 2: **CAN PHY Jumper Pin Connections**

Connect From	Connect To
JP310 pin 1	JP310 pin 2
JP320 pin 1	JP320 pin 2

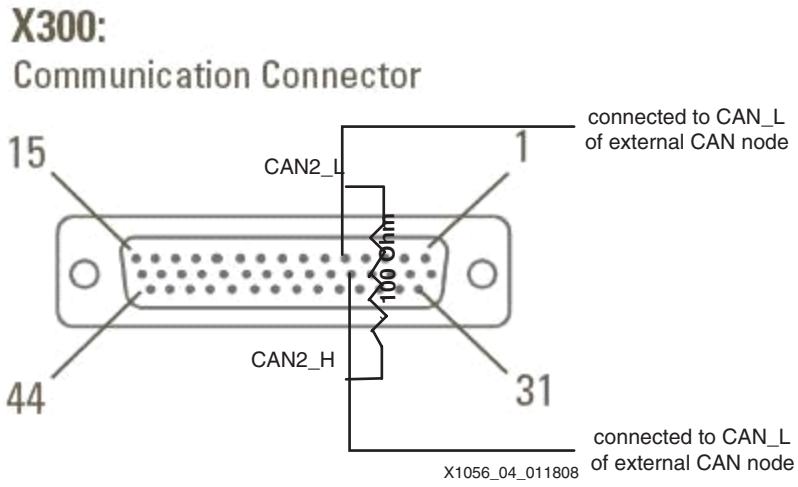

Figure 4: **CAN2_H and CAN2_L lines on X300 Connector**

The Software Application

The two XPS CAN cores are set to the Configuration mode and the Bit Timing Register (BTR) and Baud Rate Prescaler Register (BRPR) are written to so that the Baud Rate can be set to 250 Kbps. Then the interrupt handlers are set, the interrupt system is set up, and all the interrupts of the two XPS CAN cores are enabled. The two XPS CAN cores are then set to Normal mode. Frames transmitted by XPS CAN 0 node are received by XPS CAN 1 node and are verified good. The application calls the XPS CAN driver functions to set the two XPS CAN cores and the interrupt system for a transmit-receive application.

A frame is transmitted by XPS CAN 0 node and calls the Transmit Interrupt Handler (SendHandler0). The SendDone0 flag is set to TRUE when SendHandler is called. The flag TransmissionError0 is set to TRUE if transmission fails. Following the transmission by XPS CAN 0 node, the RXNEMP interrupt of XPS CAN 1 node is set and calls the Receive Interrupt Handler (RecvHandler1). The RecvDone1 flag is set to TRUE within the RecvHandler1. The flag ReceiveError1 is set to TRUE if reception fails or the frames received do not match the frames transmitted.

The application waits until Reception by XPS CAN 1 node is completed (RecvHandler1 is set to TRUE) and then checks if the ReceiveError1 flag has been set to TRUE. If yes, it either prints the message that the transmission/reception was a failure, or that it was a success. Then it loops back and waits for the Tx and Rx interrupts. The application prints status messages at different points in the program through the XPS UART16550 core.

Figure 5 illustrates the implementation flow of the software application for the XPS CAN cores operating in normal mode.

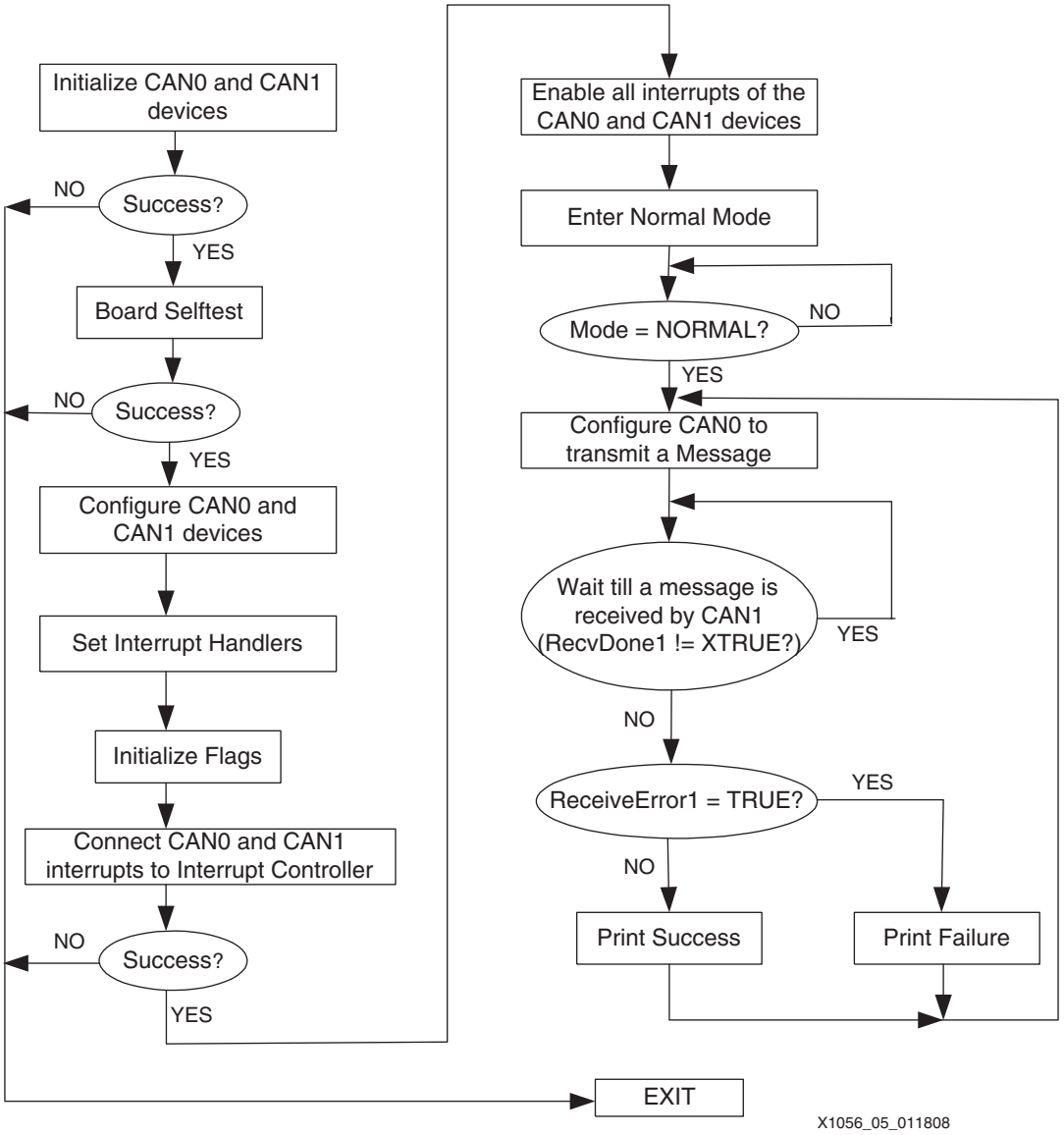

Figure 5: **Software Implementation Block Diagram**

Executing the Reference System

The XA3S1600E board must be connected to the PC via a JTAG cable to download the bitstream. The UART port (X300) of the XA3S1600E board must be connected to the UART port on the PC using a UART cable and the board must be powered-on. A pre-built bitstream, *system.bit* and the compiled software application, *executable.elf*, is available in the ready_for_download directory under the project root directory. The reference system can be executed using the pre-built bitstream together with the compiled software applications, or by generating the bitstream and the software executable in the EDK. Instructions for both methods are presented in this section.

Note: XPS CAN is a licensed core and the bitstream generation will fail without a valid license. To obtain a license key for this core, go to: http://www.xilinx.com/ipcenter/can/can_registration.htm

Primary Set-up for Executing the Reference System

1. Using HyperTerminal or a similar serial communications utility, map the utility operation to the physical COM port to be used.

2. Connect the board UART port to this COM port. T
3. Make sure the terminal settings are set to:
   - Baud rate = 9600
   - data bits = 8
   - parity = None
   - Flow control = None

See Figure 6 for the HyperTerminal settings.

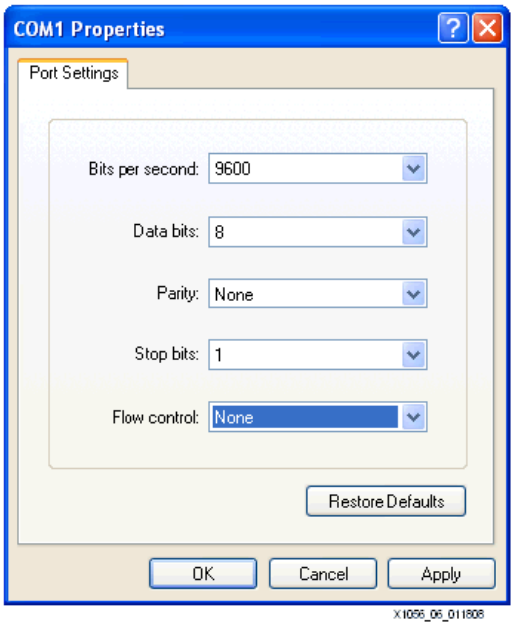

Figure 6: **HyperTerminal Settings**

Using the Pre-Built Bitstream and the Compiled Software Applications

To execute the system using files in the ready_for_download directory in the project root directory:

1. Change directories to the ready_for_download directory.
2. Use IMPACT to download the bitstream by using the command:

```
impact -batch xapp1056.cmd
```

3. Invoke XMD and connect to the MicroBlaze processor by using the command:

```
xmd -opt xapp1056.opt
```

4. Download the executable by using the following command

```
dow executable.elf
```

5. Run the software application using the **`run`** command.

Using EDK

To execute the reference system using EDK:

1. Open the desired XMP file (**`system.xmp`**) inside EDK.
2. Choose **Hardware** → **Clean Hardware** to remove previously generated hardware files.
3. Choose **Hardware** → **Generate Bitstream** to generate a bitstream for the system.

4. Choose **Software** → **Clean Software** to remove previously generated software libraries/applications.
5. Choose **Software** → **Build All User Applications** to build the software application.
6. Choose **Device Configuration** → **Download Bitstream** to download the bitstream to the board.
7. Launch XMD with **Debug** → **Launch XMD**
8. Download the executable by using the following command:

```
dow executable.elf
```

9. Run the software application using the **run** command

The status of the software application is displayed in the HyperTerminal data screen. When the software application completes, the output at the HyperTerminal should be as follows:

```
Entering Main
CAN0 Initialized
Reset and Configuration Done
Interrupt Handlers Set for First CAN
Flags Initialized for CAN0
CAN0 Connected to Interrupt Controller
Interrupts Enabled for CAN0
CAN0 in normal mode
CAN1 Initialized
Reset and Configuration Done
Interrupt Handlers Set for CAN1
Flags Initialized for CAN1
CAN1 Connected to Interrupt Controller
Interrupts Enabled for Second CAN device
CAN1 in normal mode
Frame successfully transmitted by CAN0
Frame sent by CAN0 received successfully by CAN1
Frame successfully transmitted by CAN0
Frame sent by CAN0 received successfully by CAN1
Frame successfully transmitted by CAN0
Frame sent by CAN0 received successfully by CAN1
:
:
:
```

References

1. *XPS Controller Area Network (CAN)* (v1.00a) Xilinx Product Specification, DS649
2. XA1600E Reference Guide 1.5

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/25/08	1.0	Initial Xilinx release.

Legal Disclaimer

THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.