



Programming the Ready Bus for the Intel[®] IXP1200 Network Processor

Application Note

December 2004



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel® IXP1200 Network Processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com>.

AlertVIEW, AnyPoint, AppChoice, BoardWatch, BunnyPeople, CablePort, Celeron, Chips, CT Connect, CT Media, Dialogic, DM3, EtherExpress, ETOX, FlashFile, i386, i486, i960, iCOMP, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Create & Share, Intel GigaBlade, Intel InBusiness, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel Play, Intel Play logo, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel TeamStation, Intel Xeon, Intel XScale, IPLink, Itanium, LANDesk, LanRover, MCS, MMX, MMX logo, Optimizer logo, OverDrive, Paragon, PC Dads, PC Parents, PDCharm, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your Command, RemoteExpress, Shiva, SmartDie, Solutions960, Sound Mark, StorageExpress, The Computer Inside., The Journey Inside, TokenExpress, Trillium, VoiceBrick, Vtune, and Xircom are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © Intel Corporation, 2004

Contents

- 1.0 Introduction** 5
- 2.0 IX Bus Programming** 5
 - 2.1 Introduction 5
 - 2.1.1 Ready Bus 5
 - 2.1.2 Ready Bus Sequencer 5
 - 2.2 Programming the Ready Bus 6
 - 2.2.1 Ready Bus Sync Count 7
 - 2.2.2 Ready Bus PROG 8
 - 2.3 Programming the Microengine 8
 - 2.3.1 Microcode Processing 8
 - 2.3.2 Workaround for Reading the Stale Flag 8
- 3.0 Summary** 9

Figures

- 1 Programmable Sequencer for Ready Bus 6
- 2 Ready Bus Program Cycle 7

Revision History

Date	Revision	Description
December 2004	001	Initial document.

This page intentionally left blank.

1.0 Introduction

This document is designed to accelerate the learning curve of software engineers who are new in developing microcode for the Intel® IXP1200 Network Processor.

This document lists a number of best known methods (BKMs) on programming the microengine to receive a packet from the MAC. The methods and suggestions may or may not be what a programmer is used to, and they are not by any means the hard rules on programming with the microengines. Programmers have their own style of programming, and this document only provides suggested methods.

Note: Incorrect programming of the ready bus can cause the microengine to stop transmitting data.

2.0 IX Bus Programming

2.1 Introduction

2.1.1 Ready Bus

The Ready Bus, which is a separate bus from the IX Bus, is used for the following purposes:

- Retrieving the MAC Receive and Transmit FIFO Ready flags from MAC devices.
- Asserting Flow Control to MAC devices.
- Inter-Intel IXP1200 Network Processor communications.

The Ready Bus is controlled by a programmable sequencer and is comprised of an 8-bit data bus (RDYBUS[7:0]) and five control pins (RDYCTL#[4:0]). The control pins select which of the three functions is being performed on the bus. In 1-2 MAC mode, separate control signals are provided by the Intel IXP1200 Network Processor for each function. In this mode, the GPIO[0] pin is also used by the Ready Bus as a control signal, and Inter-Intel IXP1200 Network Processor communications are not supported. 3+ MAC mode requires an external decode of the RDYCTL#[4:0] signals to provide the individual select signals for each function. All of the Ready Bus functions are supported in 3+ MAC mode.

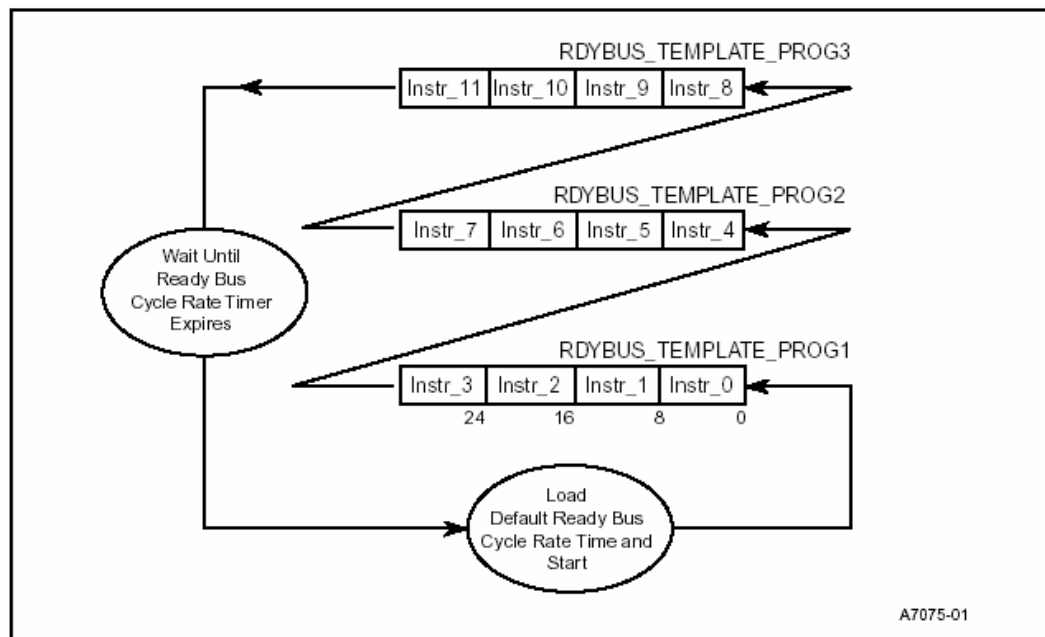
2.1.2 Ready Bus Sequencer

The Ready Bus is controlled by a programmable sequencer. The sequencer is configured and programmed using FBI registers. The sequencer is configured and enabled after reset and then runs freely.

The Ready Bus Sequencer instructions are loaded into the three RDYBUS_TEMPLATE_PROG_x registers. These registers hold a total of twelve 8-bit instructions. A sequence rate count can be programmed into the RDYBUS_SYNC_COUNT_DEFAULT register to determine the maximum rate at which the program repeats the instructions. The Ready Bus Sequencer is enabled via the RDYBUS_TEMPLATE_CTL register. When it is enabled, the sequence rate counter is started.

Instruction execution begins with **instr_0** and completes after **instr_11**. The sequencer then waits until the Ready Bus sequence rate timer expires and repeats execution of instructions 0 through 11. The sequence rate count has no effect if it is less than the time required to execute the instructions.

Figure 1. Programmable Sequencer for Ready Bus



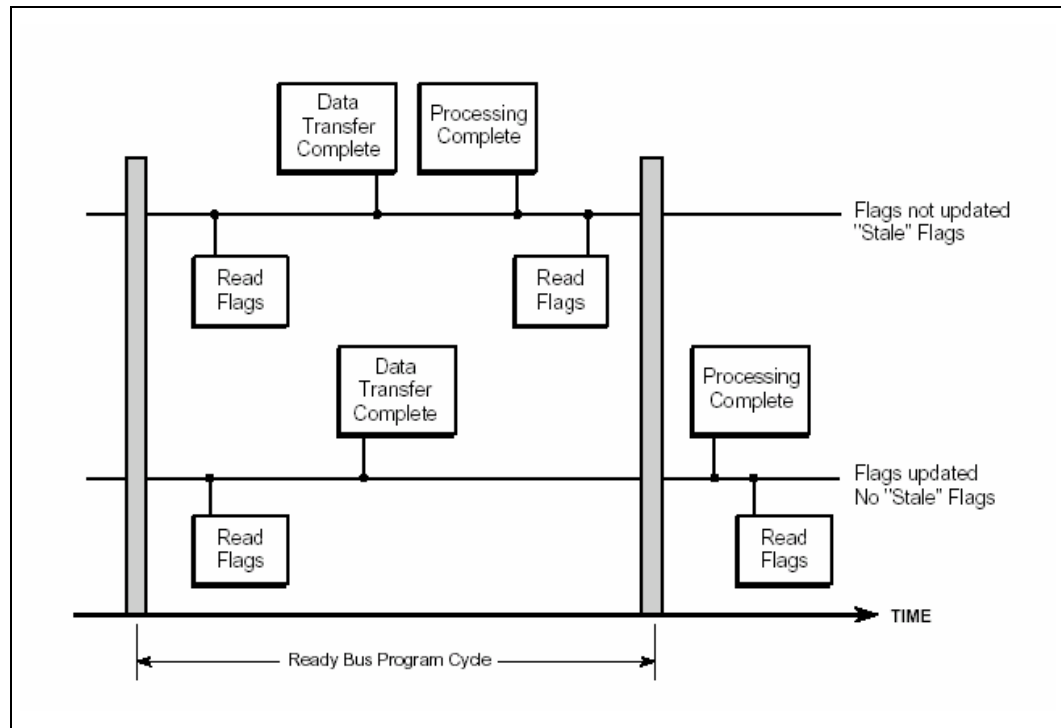
The complete list of instructions that could be used for programming the ready bus template is available in the *Intel® IXP1200 Hardware Reference Manual*.

2.2 Programming the Ready Bus

To have the ready bus run as quickly as possible, make sure that the ready flag in the microengine is refreshed every time the microengine reads it. Some of the things that could minimize the instructions to complete a ready bus cycle are discussed below.

If the speed of the ready bus sequencer is not maximized, you might have an issue of reading a stale ready flag. This is because the microengine and the IX BUS is running at a different speed. To avoid this, make sure that every time the microengine reads the ready flag from the RCV_RDY_LO/HI that it is the latest flag. The rate at which this flag is updated in the RCV_RDY_LO/HI is determined by how quickly the ready bus runs.

Figure 2. Ready Bus Program Cycle



In Figure 2, notice from the top example that before the ready bus completes its cycle, the microengine has finished its processing and also has read the flag for the second time. This will cause an issue since the flag is not the 'latest'. As an example of why this is an issue, consider that there might be a last mpacket in the MAC, and therefore the ready flag initially shows a '1'. When the microengine receives this packet, there will be no more packets in the MAC, and therefore the rxrdy in the MAC is '0'. However, since the ready bus hasn't finished its cycle yet, it will not poll the '0' status from the MAC. Therefore, if the microengine polls the RCV_RDY_LO/HI before it is updated, it will still show a '1' from the previous packet and will issue a receive request.

When the MAC doesn't have a packet and the Intel IXP1200 Network Processor issues a receive request, both the MAC and the IXP1200 will go into an unknown state. This will cause the microengine to hang. When this happens, the receive microengine will have a thread that is waiting for a receive start signal that will never arrive, while another microengine will get a receive start signal that it is not waiting for.

To make sure this never happens, follow the instructions below to program the ready bus, and always program the microcode to avoid reading a stale flag. Also, a BKM is available that explains a workaround in case a stale flag is read. See Section 2.3.2.

2.2.1 Ready Bus Sync Count

The ready Bus Sync Count should be kept as low as possible. A value of 0 should be programmed so that you don't have to wait for the sync count to count to 0 to begin the ready bus sequencer again.

2.2.2 Ready Bus PROG

To program the ready bus PROG, first determine if auto push is used. In the Intel reference design, the autopush is not used, and therefore the following instructions show only how to program the rdybus to update the receive ready flag into the Intel IXP1200 Network Processor.

```

; running 8 10/100 + 1 gigabit port
#define RDYBUS_PROG1 0x00fb0000 ; rx rdy mac 0 == 0xfb. flow control == 0x00
#define RDYBUS_PROG2 0x00d70000 ; tx rdy mac 0 and 1 == 0xd7, flow control
                                (executes faster than nop)
#define RDYBUS_PROG3 0x00000000 ; flow control (executes faster than nop)

```

Assuming you only have one IXF440, read the ready flag from MAC0 and program 0xfb into RDYBUS_PROG1. To poll the transmit ready flag from both MAC0 and MAC1, you then issue a 0xd7 in RDYBUS_PROG2, and the rest is set to 0x00.

Note: In the above example, the IXF440 is polled only for the receive ready flag. Speculative receive requests are used for the IXF1002. For additional detail, see the *Intel® IXP1200 Network Processor Hardware Reference Manual*.

- Flow control is used to fill the RDYBUS_PROGx as it only requires 1 cycle to run flow control while NOP takes 3 cycles.
- No more than one rxrdy/txrdy is used for every ready bus cycle because the Ready Bus only execute these instructions once. Having more than one rxrdy/txrdy doesn't help in the polling speed of these flags.

2.3 Programming the Microengine

The microengine code can be optimized so stale flags are not read. The RCV_RDY_LO/HI flag should not be read too quickly, which might cause a stale flag to be read.

2.3.1 Microcode Processing

In the Intel reference design and also in most of the code written by customers, there is always a path in the code for minimal processing. An example of this is an exception packet. In the Intel reference design, when an exception packet is presented, that packet is immediately dropped and the code execution returns to the beginning of the packet processing routine.

Another example of minimal processing is during an mpacket. In the case of an mpacket, the microengine simply moves the packet into the SDRAM and loops back into beginning of the code, checking for the ready flag again. This is another example where there might be a chance to read the stale flag.

Once a short path has been identified, a delay (with ctx_arb and nops) should be added to make sure the microengine doesn't loop back to read the ready flag immediately.

2.3.2 Workaround for Reading the Stale Flag

When a receive request is issued and there is no packet in the MAC, the MAC will issue two receive start signals to the thread that issued this receive request. So when this happens, there will be an extra signal in the thread that has issued this receive request. When the other thread tries to

issue a receive request, it will not get a receive start as the microengine can have only one receive start at any one time. Therefore, the following microengine that issued a valid receive request will not get the receive start signal until the additional signal in the previous microengine has been consumed.

The code shown below will help in checking and consuming an extra receive start signal should it ever arise:

```
// This macro will consume a start receive signal, if present.
// It is used as a safety net to catch start_receive signals
// sent from the IXF440 that we don't expect
#macro CONSUME_START_RCV()
.local $ignored
    br_!signal[start_receive, skip#], guess_branch
    nop
    csr[read, $ignored, rcv_cntl], ctx_swap
    skip#:
.endlocal
#endm
```

This macro could be added after the receive control is read to check for an additional signal.

3.0 Summary

This document should provide a better understanding of how to program the receiving of packets with the microengine. It is not guaranteed to give the best result; however, the solutions presented above have been useful in some applications.



This page intentionally left blank.