Input Coprocessor (IO\_MPU)

Table of Contents

[Overview 2](#_Toc331414523)

[SPI Communications 2](#_Toc331414524)

# Overview

This document describes the Input Coprocessor (IO\_MPU) design of the RAZEC10.

The objective of this design is to offload the majority of the workload to a co-processor, so as to minimize the amount of implementation within the FPGA chip, as well as to minimize the number of FPGA pins required to implement Keyboard Mouse, and Joystick input.

The FPGA chip contains some very high speed dual ported BRAM memory. One page of of this memory can be set aside to act as a communications medium between the Z80 and the IO\_MPU.

The memory is split into two regions. Bytes to be sent to the IO\_MPU, and bytes received from the IO\_MPU.

The memory region that is received from the IO\_MPU contains the current state of the buttons (including keyboard, joystick, and mouse buttons), the current mouse position, and the current analog pin states.

The memory region that is sent to the IO\_MPU contains information about reconfiguring the mouse, and the keyboard lights.

All memory is read-write for the Z80. However, the content of the receive buffer memory will be destroyed as the new packet is received (which only happens when requested by the Z80).

# SPI Communications

The AudioChip implementation in the FPGA contains a master SPI controller. This SPI controller grants priority-controlled access to both the IO\_MPU and the SD-CARD, to the Z80 computer.

The IO\_MPU contains a slave SPI implementation that allows the current state of all input to be pulled into the FPGA’s BRAM.

The FPGA begins repeatedly transmitting ‘REQUEST\_EXCHANGE’ (0xCE) to the IO\_MPU. The IO\_MPU is pre-configured to automatically reply with BUSY’s (0x00) until the ISR and mainline have decided that a new capture has been completed and is ready for exchange. The ISR then replies with ‘OK’ (0xFF), followed by a stream of bytes which are in-fact the entire content of the packet to be sent to the FPGA. The packet’s last few bytes are a counter (for seeing the packet changing, when copied on-screen, or whatever other use you wish to make of it), and the final byte (0xA5) which is just a visually easy to identify value marking the end of the packet, when looking at memory dumps. Note that A5 may appear elsewhere in the packet as well. It’s not an end-marker.

From the FPGA’s perspective, it is continually sending 0xCE’s until it receives an oxFF. It then begins sending the outgoing memory from the BRAM, for the number of bytes that is mutually agreed-upon (at compile-time) of the length of the packet.

The FPGA will not begin this sequence until the CPU requests that it do so. Once complete, the FPGA leaves a “Done” flag set for the CPU to see.

NOTE: If the FPGA exchanges more bytes than the IO\_MPU has been compiled for, then the additional packet bytes will be viewed by the IO\_MPU as more commands. Assuming 0xEC is the only valid command, then the IO\_MPU will not do anything unless an 0xEC is sent to it. At that point, it will begin another transfer, and in all likelyhood, things will be pretty messed up pretty quickly.

NOTE: If the FPGA exchanges too few bytes, then the IIO\_MPU will be stuck in exchange mode at the end, and will not respond correctly to the beginning of another packet.