DSD2 Project Documentation

Medinceanu Paul-Catalin

2024

1 Introduction

The objective of this semester's digital project was to implement a module capable of reading from and writing to a processor's memory blocks, as well as controlling its activity. The complete system schematic is shown in Fig. 1.

2 Memory Interface Protocol

The module receives data via the UART protocol. The data is organized into multiple frames, each containing information about the action to be performed by the system. The instruction set of the module is detailed in Table 1.

Table 1: Protocol Message Structure

UART Debugger Message

Cmd Frame		**Add Frame 1**	**Add Frame 2**	**Len Frame 1**	**Len Frame 2**	**Data Frame**
Name	**Code**	**Code**	**Code**	**Code**	**Code**	
Reset	1000_1110					
Start	1000_1100					
Halt	1000_1010					
Read	1000_1000	xxxx_xxAA	AAAA_AAAA	xxxx_xxLL	LLLL_LLLL	8bit frames, packed into 32bit data, for Read
Write	1000_0110	xxxx_xMAA	AAAA_AAAA	xxxx_xxLL	LLLL_LLLL	32bit frames, unpacked into 8bit data, for Write

The first 8-bit frame received through the UART RX line is the command. The commands can be split into two categories: those that control the core (RESET, START, and HALT) and those that interact with the memory (READ and WRITE). The messages for the instructions that interact with the CPU contain only the command frame. Upon reception, they assert a signal that controls the CPU, and the interface module will return to a state where it waits for other commands.

The READ and WRITE commands have a message structure that includes the start address from which the data will be read or written, the length of the transfer, and, in the case of the WRITE command, the data itself. It is important to note that the length is specified in terms of the number of words. For example, a length of 4 for data written in the program memory corresponds to 4 words of 16 bits each, not 4 bytes.

For the WRITE command, the bit in position 2 of the first address frame determines whether the data will be written to the data memory (DMEM) or the program memory (PMEM). If this bit is set to 0, the destination is the data memory. If it is set to 1, the destination is the program memory.

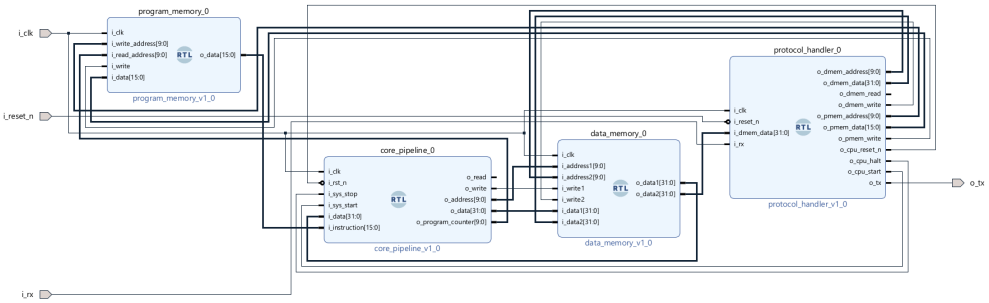

Figure 1: System Schematic

3 AXI UART State Machine

The project involved familiarizing with the AXI protocol, so an AXI UARTLite IP was used to facilitate data communication. To simplify the interaction with this circuit, a state machine that controls it through the AXI interface was developed. Its schematic is presented in Fig. 2, and its state flowchart is shown in Fig. 3.

To transfer data through the UART lines, the key signals used are write, read, $write\_data$, $read\_data$, $interface\_idle$, $read\_data\_ready$, and $write\_data\_full$. When write is set high, the circuit checks if the TX FIFO is full. If the TX FIFO is full, the $write\_data\_full$ flag will be set high, indicating that no more data can be written until space becomes available. Otherwise, the circuit will transmit the $write\_data$. For reading data received on the RX line, the block first checks if there is data in the RX FIFO. When data is available, it copies the data from the RX FIFO to the local data register and sets the $read\_data\_ready$ signal high, indicating that new data is available to be read. The $interface\_idlesignal$ is high when there is no activity in the block, meaning that neither reading nor writing operations are currently in progress.

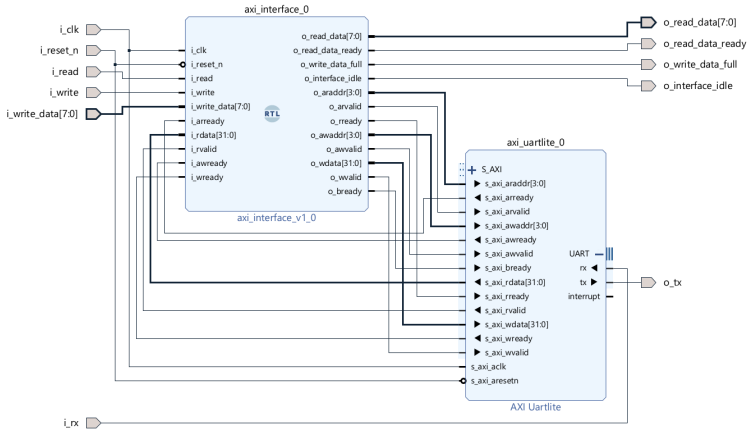

Figure 2: UART Schematic

4 Memory Interface State Machine

The memory interface state machine incorporates the UART state machine. It implements the custom communication protocol detailed in Table 1. The module is always waiting to receive a command. depending on the received instruction different actions can take place. The state diagram of the memory interface is presented in 4. In teh $REQUEST\_COMMAND$ state, the UART module is always receiving and reading data. When the *read_data_ready* signal is high the interface enters the $CHECK\_COMMAND$ state and decodes the instruction and chooses the following path of states. At the end of one of these paths the interface returns to the $REQUEST\_COMMAND$ state.

The write and read command paths are the most complex, and they follow similar processes up to the $WRITE_LEN2$ and $READ_LEN2$ states. Until these states, the received data is concatenated to deduce the start address of the transfer and the number of words that will be moved. The write path has the particularity that the third bit from the first address frame determines whether the data will be written in the PMEM or DMEM.

The write and read command paths involve two counters each. One counter counts the bytes to form the 32-bit or 16-bit word, depending on whether it's a write or read command. In the case of the write command, this counter assembles the bytes into 32-bit words for DMEM and 16-bit words for the PMEM. For the read command, this counter splits the 32-bit word into bytes. The other counter increments the address, starting with the address value sent in the initial message. The transfer stops when this counter reaches the starting address plus the specified length.

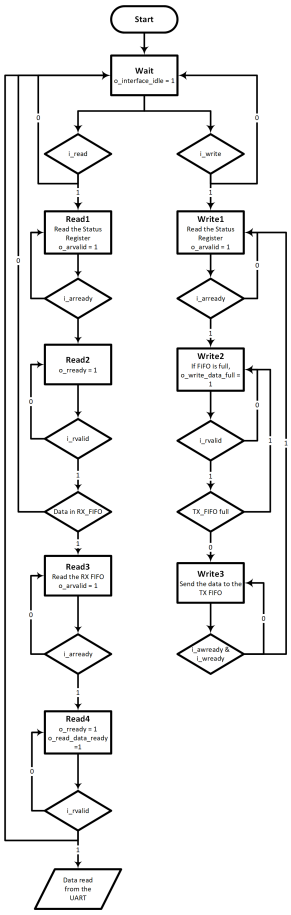

Figure 3: UART State Diagram

In the case of the read command, the data is read from the DMEM and needs to be written into the TX FIFO. The TX FIFO has only 16 memory locations of 8 bits each, so there is a high chance that it will overflow if the data is not read out fast enough. To prevent overflow, the UART state machine uses the $interface\_idle$ signal, which is considered by the read command path. If this signal is asserted low, it means that the interface is busy and cannot accept more data to be sent. In this case, the read command path loops back to the $READ\_DMEM2$ state until the TX FIFO has enough space to add more data to be sent.

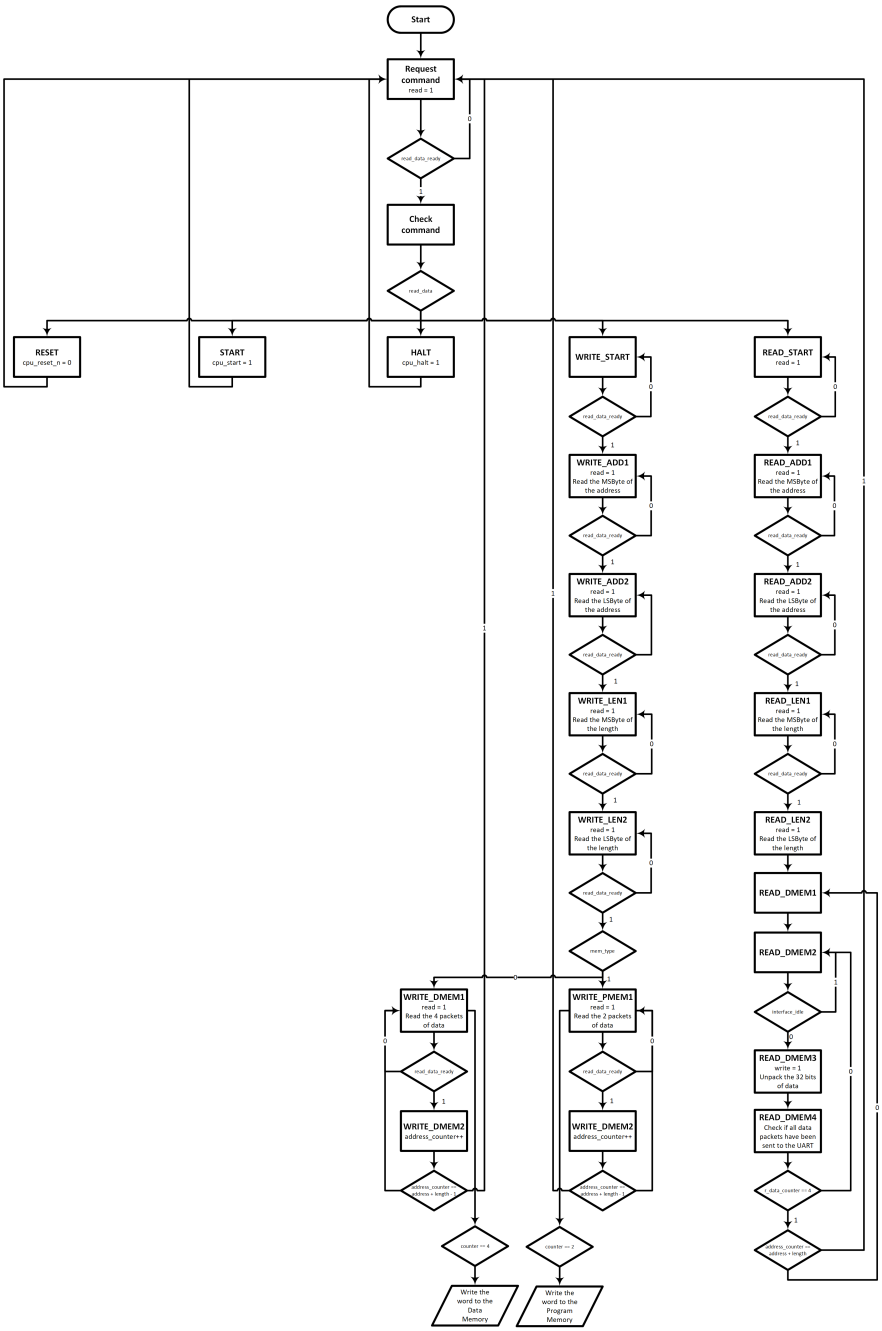

Figure 4: Memory Interface State Diagram