BetterMemory Peripheral Project Summary

Rudra Goel

Submitted
December 3rd, 2024

Introduction

BetterMemory is a peripheral designed to expand SCOMP's limited memory by storing and retrieving up to 65,536 16-bit values. Throughout its creation, our team emphasized ease-of-use to a SCOMP user, leading us to develop an auto-increment feature in situations where data storage and retrieval at consecutive addresses is common. From product definitions to ideation and prototyping, we sought to minimize confusion in storing/retrieving data for SCOMP programmers. In the end, BetterMemory meets all aspects of its proposed definitions, and this document serves as a reference for understanding the functionality of our peripheral from an end-user's perspective along with a rationale for certain design choices.

Device Functionality

To use BetteryMemory, a programmer must first edit SCOMP's decoder and top-level BDF file to enable the architecture to communicate with three new registers read by the peripheral: Value Enable (0x70), Address Enable (0x71), and Auto-Increment (0x72).

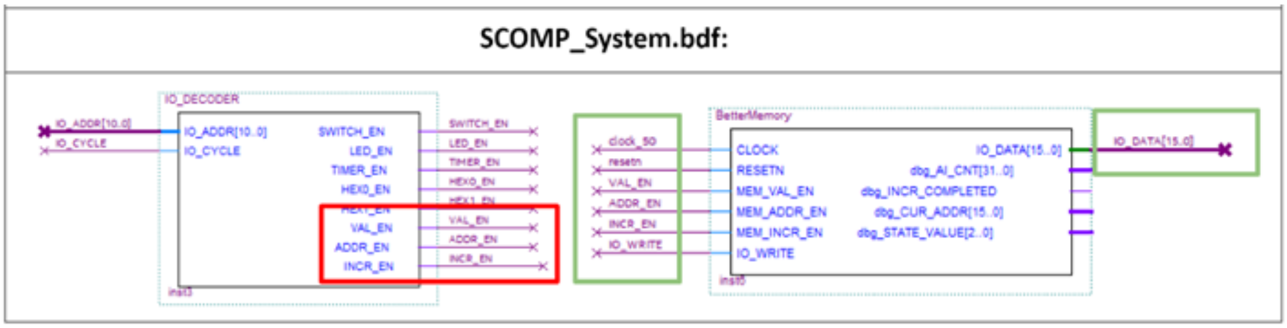

Figure 1. SCOMP modifications needed to interface with BetterMemory.

After BetterMemory is fully integrated, the user can point to any external memory address by loading in an address between 0 and 65,535 into the accumulator and subsequently writing to the address of the Address Enable register, 0x71, via the instruction "OUT." If an address is not specified at reset, the pointer will default to address 0x0000. Upon setting an address, SCOMP programmers can store a value by loading it into the accumulator and executing "OUT" to the Value Enable register address, 0x70. To load a stored value into the accumulator, they should read from the Value Enable register's address via the instruction "IN." Similarly, reading from the Address Enable register's address would load the current address into the accumulator.

Auto-Increment Feature

To enter auto-increment mode, a programmer must write to the address of the Auto-Increment register, 0x72, after which BetterMemory will point to the current address plus one, and the address will increase by one after each "IN" or "OUT" instruction to the Value Enable register. Auto-Increment is deactivated by writing a second time to the Auto-Increment register's address.

Design Decisions and Implementation

Our team first described BetterMemory as a finite state machine (FSM) with the three enable signals as inputs because it defined a concrete structure for a system that needed to be in a "reading" or "writing" state for both memory address pointers and data. Like with any FSM, an idle state is necessary especially to avoid timing issues between SCOMP and BetterMemory's distinct clock speeds. To ensure that the FSM is able to return to the idle state between SCOMP instructions, we use a 50 MHz clock for BetterMemory, instead of the 12 MHz signal clocking SCOMP.

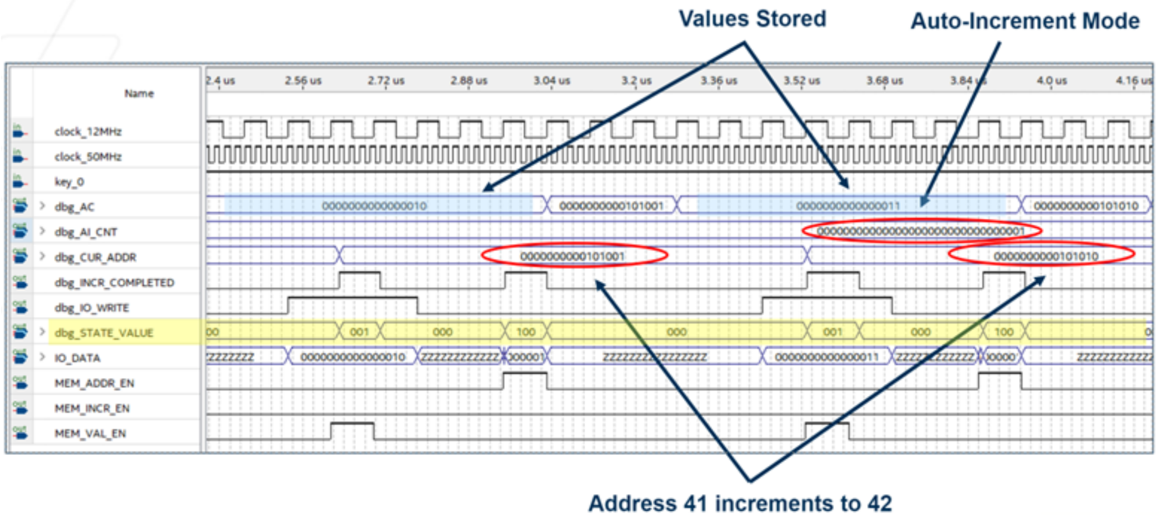

Figure 2. Waveform illustrating the process of writing values 0x0002 and 0x0003 to altsyncram memory while in auto-increment mode. Shows states Idle (000), Writing Val (001), Reading Addr (100).

While defining the auto-increment feature, we initially sought to keep the current address pointer of BetterMemory unchanged when auto-increment is first enabled. Auto-increment with this definition was error-prone, most likely because our implementation sequenced through several states per SCOMP instruction cycle. Therefore, our team decided to increment the internal address pointer by one immediately upon enabling auto-increment.

Conclusions

Our team aimed to solve SCOMP's limited memory without being invasive to its internal architecture. We thus created a peripheral accessible through SCOMP's I/O protocol allowing the computer to interface with more memory via altsyncram on the FPGA it sits on. Reflecting on our team's work, we spent significant time debugging VHDL, when in retrospect, we could have spent more time in the 'define' and 'ideate' loop to determine critical issues in our design and resolve bugs efficiently. For future engineers expanding BetterMemory, it is recommended to incorporate additional "feature" states into its FSM that transition from the idle state *only* as transitioning from another state would disrupt the baseline functionality of storage and retrieval of data. Additional features could include write protection or password-locked memory.