ARM – Embedded coding assignment

Alan Mujumdar

The attached code package contains software necessary to communicate with the target device and a simulation environment. File testSendMsg.c contains several test options for evaluating the sendMsg function (sendMsgFunction.c). The code contains lots of comments which should explain the behaviour of each function and test option. The build options are listed in testSendMsg.c.

Software Details

Assumptions

- The target device STATUS register is updated immediately after the DATA register receives a transaction. My sendMsg function evaluates the STATUS register after performing a write to DATA. If the target device fails to ACK the burst, the data is retransmitted.
- The target device supports little-endian data format. For transactions containing more than one byte, every consecutive data byte is bit shifted into a 32 bit variable. First byte extracted from the input array is written into the least significant byte of the variable. The fourth byte is written into the most significant byte of this variable. When a burst packet is prepared, it is written to the DATA register.
- The CTRL register is written when a new transaction begins. All the fields of this register are set as: SEND_ADDR = 1, DATA_LEN = (0, 1, 2, or 3), ADDR = desired address.
  If the transaction contains more than 4 bytes, the CTRL register is set with the maximum DATA_LEN of 3 (which is 4 bytes). Every consecutive burst does not require a change to the CTRL reg so the SEND_ADDR flag is set to zero.
  If the last burst in a transaction contains less than 4 bytes, the CTRL register must be changed accordingly. The address is sent again, but the DATA_LEN field is set to the number of bytes being transmitted (1, 2, or 3 bytes).

Approach

The target hardware can not be directly tested, however, the produced code must communicate correctly. This issue is resolved by conditionally compiling the code (#ifdef TEST_SENDMSG) to work with the specified memory mapped registers or a simulation environment.
The simulation environment provides control over the target registers and allows emulation of inputs and outputs. For instance, the STATUS register is read-only but in the simulation environment, its contents can be edited.
In some of the testing modes, the STATUS register can be set to a random response which simulated chaotic ACK and BUSY responses. This has allowed me to observe correct retransmission and back-of procedures from the software.

I use bit masks to set the correct flags in the CTRL register and read the STATUS register. Two functions, generateCtrl() and generateData() are used for packaging the correct 32 bit register values. C language has libraries for uint32_t and uint8_t, as well as functions for concatenating bytes. I have written my own functions instead of using the libraries.

The code works according to the specification and my own assumptions listed above. However, the code has not been exhaustively tested, so errors and unanticipated behaviour might still be observable. Both builds of this software produce no compiler warnings or errors. The code has also been checked with valgrind and no errors or warning have been detected.

Development Time

It is difficult to specify the exact time I spent on this exercise. I carefully read the details of this assignment when I received the email but I did not attempt it for a few days due to other commitments. When I actually sat down to write the code, it took me about an hour to outline the requirements, initial approach, and set-up all the code files.

It took me another 2 hours to write most of the code and the simulation. This includes mini tests for individual pieces of code and resolving compiler warnings/errors. I then spent another 2 hours going through the debugging and testing routine. I tried to evaluate most of the corner cases; ensuring correct behaviour when the function was called with no data, odd/even values, varying STATUS reg responses, etc.
I also spent some time aligning the print messages, code comments, etc.