Stage 8

What is required to be done?

Memory partitioning and input port

Suppose you have implemented a 64x32 memory with byte adresses going from 0 to 255, that is from 0x00000000 to 0x000000FF. This can be partitioned into system are from 0x00000000 to 0x0000007F and user area from 0x00000080 to 0x000000FF. This means that if the current mode is user mode, access to addresses in range 0x00000000 to 0x0000007F is prevented (in case of read access return 0, in case of write access, do not write). Let the address of the input port be 0x00000070. Data going from memory to PMconnect should be multiplexed with data coming from input port. Input port data is selected when address = 0x00000070, otherwise memory data is selected.

ISRs

These need to be placed somewhere in system area. For example, ISR of reset can be placed at 0x00000020 and ISR for SWI can be placed at 0x00000040. Instructions to branch to these addresses need to be placed at 0x00000000 and 0x00000008. Remember that a branch instruction contains a word offset from PC+8. In this case the byte offsets are (0x20-0x08) and (0x40 - 0x08 - 0x08), i.e., 0x18 and 0x30. Corresponding byte offsets are 0x06 and 0x18. ISR for reset loads 0x80 in LR (starting address of user program) followed by RFE instruction. ISR of SWI loads from input port into register R0 followed by RFE. A more complex ISR may require saving/restoring of LR and other registers.

Extend the hardware design to implement BL, SWI, RET and RTE instructions

BL saves PC+4 in LR and adds offset to PC. No change in mode.
SWI saves PC+4 in LR and makes PC = 0x00000008. Mode changes to supervisor.
RET copies LR into PC. No change in mode.
RTE copies LR into PC. Mode changes to user.

Since RET and RTE are instructions defined by us, their codes can't be generated using ARMSim#. These would need to be manually coded in user programs/ISRs.

Test bench

The test bench should simulate a slow input peripheral which provides a sequence of data bytes at intervals which are much larger than the CPU clock period. After changing the input data, the status signal (9th bit) should become 1 for several clock cycles and then become 0 till the next data change.

Full predication and S-bit

Full predication means that all instructions are affected by the condition specified by bits 31-28 of the instruction, not only branch instructions. At this stage full predication is to be implemented with complete set of conditions. Condition checking logic is already there in the current design. Output of this logic is to be used for enabling/disabling control signal for RF write, Memory write and flags update. Also s-bit for DP and multiply group of instructions is to be used to control flags update.

There are two parts of the setting the flags - (a) determining the values to be POSSIBLY assigned to flags and (b) actually making these assignments. There are 3 sources of the values to be assigned to flags - ALU, Shifter and Multiply-accumulate unit. Thus logic for (a) would be distributed at 3 places, but (b) should be done at one place - either in a separate module designed for this purpose or in the main module/glue logic.

User Program

A program placed in user area should demonstrate reading a sequence of bytes from the input port and storing in memory. For each byte it would iterate in a loop where the input port is polled. Polling simply means reading the input port using SWI instruction and checking the status bit. When the status is "ready" the data byte is saved in memory. The user program should also demonstrate that input port can't be read in user mode.

Additional programs are required to demonstrate (a) BL and RET instructions, (b) full predication capability and (c) S-bit capability.