Problem description

The goal of this learning activity is to update the address decoder to include the UART address component. The address decoder will take a 32-bit input, Addr, and a 1-bit input, MemWrite, to determine which component is used for address. The outputs will include RAM_CS, which selects the RAM memory; RAM_WE, which writes to RAM memory; ROM_CS, which selects ROM memory; UART_WR, which writes to the UART register; UART_RD, which reads from the UART register; CE_UART, which enables access to the UART; and CE_SR, which reads from the UART status register. Finally, the design will be verified using an assembly program that transmits and receives two bytes. The results will be simulated by the HDL testbench..

Verilog implementation

The address decoder implementation is used to map addresses to selected memorys and trigger corresponding signals. Initially, all output registers are set to 0. Here are the specific mappings and behaviors:

- When the Addr input is within 0x000 to 0x3FF, the address will be written into ROM memory, and ROM_CS will be set to 1.
- When the Addr input is within 0x400 to 0x4FF, the address will be written into RAM memory. The value of RAM_WE will be determined by MemWrite, and RAM_CS will be set to 1.
- When the Addr input is 0x500, the address will interact with the UART register for transmission/reception. CE_UART will be set to 1. If MemWrite is 1, UART_WR will be set to 1 and UART_RD will be set to 0; otherwise, UART_WR will be set to 0 and UART_RD will be set to 1.
- When the Addr input is 0x504, the address will read from the UART status register. UART_RD and CE_SR will both be set to 1."

```
always @* begin
    RAM_CS = 1'b0;
    RAM_WE = 1'b0;
    ROM_CS = 1'b0;
    UART_WR = 1'b0;
    UART_RD = 1'b0;
    CE_SR = 1'b0;
    CE_UART = 1'b0;

    if (Addr >=  32'h0 && Addr <= 32'h3FF)
    begin
        ROM_CS <= 1;
    end

    else if(Addr >=  32'h400 && Addr <= 32'h4FF)
    begin
        RAM_CS <= 1;
        RAM_WE <= MemWrite;
    end

    else if(Addr == 32'h500)
    begin
        CE_UART <= 1;
        if(MemWrite)begin
            UART_WR <= 1;
        end
        else begin
            UART_RD <= 1;
        end
    end

    else if(Addr == 32'h504)
    begin
        UART_RD <= 1;
        CE_SR <= 1;
    end
end
```

Figure 1: address decoder implementation

Results Verification

The result is tested by the assembly code shown in Figure 2. The assembly starts by setting up the ASCII values and address values that will be used later. The two bytes to be transmitted, H and i, are stored in registers x1 and x2, respectively. x3 stores the address for UART data transmission and reception, and x4 stores the address for the UART status.

To begin transmitting the first byte H, the UART status is read and stored in register x10. The value in x10 is then used for bitmasking with 0x01, and the result is stored in x11. If x11 is 0, indicating that the UART status register shows it is not ready, it will loop back to transmit_H and start again until x11 is 1. Once the status changes to ready, the value from x1 will be stored at the address in x3, which is the address for UART transmission and reception.

For receiving the byte H, the UART status is first read and stored in register x12. The checking step is the same as transmission, except the bitmask used is 0x02. Once the status indicates ready, the value is loaded from the UART transmission and reception address and stored into register x5.

The same process is repeated for transmitting the byte i.

Once all transmissions and receptions are completed, the code will end with an infinite loop.

```
# Data setup
    li x1, 0x48                 # store hex value of ASCII 'H' in x1
    li x2, 0x69                 # store hex value of ASCII 'i' in x2
    li x3, 0x00000500           # store the address for UART data transmission and reception in x3
    li x4, 0x00000504           # store the address for UART status in x4

# Transmit 'H'
transmit_H:
    lw x10, 0(x4)               # Load UART status value to x10
    andi x11, x10, 0x01         # Check if TxRDY is set, store the value into x11
    beqz x11, transmit_H        # If not ready (x11 == 0), loop and check again
    sw x1, 0(x3)                # If ready, store the x1 value ('H') into the x3 (address for UART data transmission and reception)

# Receive 'H'
receive_H:
    lw x12, 0(x4)               # Load UART status value to x12
    andi x13, x12, 0x02         # Check if RxRDY is set, store the value into x13
    beqz x13, receive_H         # If not ready (x13 == 0), loop and check again
    lw x5, 0(x3)                # If ready, load the value ('H') from x3 (address for UART data transmission and reception), store the value into x5

# Transmit 'i'
transmit_i:
    lw x10, 0(x4)               # Load UART status value to x10
    andi x11, x10, 0x01         # Check if TxRDY is set, store the value into x11
    beqz x11, transmit_i        # If not ready (x11 == 0), loop and check again
    sw x2, 0(x3)                # If ready, store the x2 value ('i') into the x3 (address for UART data transmission and reception)

# Receive 'i'
receive_i:
    lw x12, 0(x4)               # Load UART status value to x12
    andi x13, x12, 0x02         # Check if RxRDY is set, store the value into x13
    beqz x13, receive_i         # If not ready (x13 == 0), loop and check again
    lw x5, 0(x3)                # If ready, load the value ('i') from x3 (address for UART data transmission and reception), store the value into x5

end:
    j end                       # infinite loop
```

Figure 2: assembly code for testing UART transmission and reception

The results from the HDL testbench are shown in Figure 3 and Figure 4. Figure 3 demonstrates that the register x5 successfully stores the value of H from the UART transmission/reception address, while Figure 4 shows that it also successfully stores the value of i from the same address. Since the expected values are stored and read correctly, we can conclude that the transmission and reception of two bytes via UART are successful.

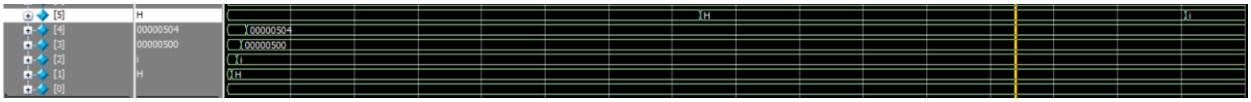

Figure 3: successful transmission/reception of byte “H”

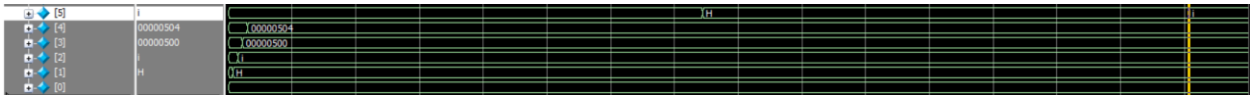

Figure 4: successful transmission/reception of byte “i”