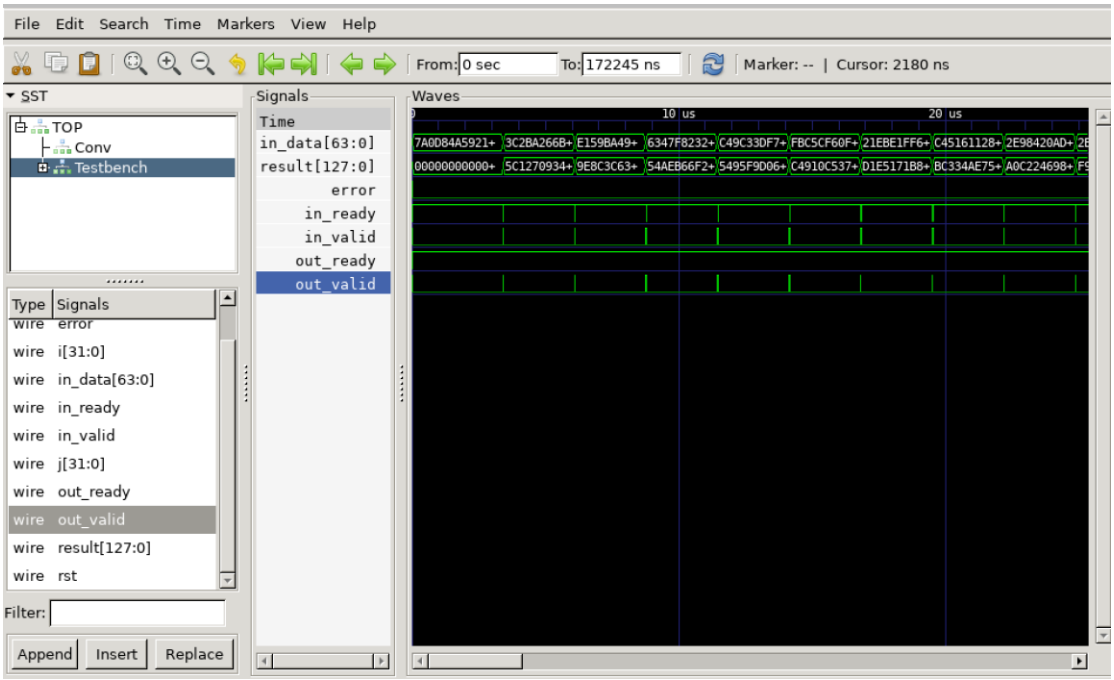
Lab4-1 Report

1.仿真输出 success

```
simulate result: 8682f5bc72938a10b3aafddac30f7a1c
simulate result: 90adcd831d131ec968a409a00ea11896
simulate result: 16178ac9e559f755273580475fd07c96
simulate result: 8e01629c952bec69fc0b64152cc935b0
simulate result: 85c4790a3b384d966f8c055b2f6e7f32
simulate result: a8622b88882ea3af2a7668c61084f0f8
simulate result: faf5206a3248848b4a7d408eaa9578d0
simulate result: 6a39b05b312ca34e5f426042824548c2
simulate result: 58a12ed48558225e410d869a79f5200a
simulate result: bdc5bb29c5e898c87cae6cd523c35ee0
simulate result: 4dbe7bc723c2bcffc7a264f7b3f610f8
simulate result: ffa42c06967ba7a06e9ab3b84e748570
simulate result: 98166c34cba4ebf0a95f36c3607de872
simulate result: b4f549b05d44c52852a3c8776ef02b3a
simulate result: 70b763d2966b8b021ed5ca380260bb9a
simulate result: 178093ff2bffd56db862c7ce45403806
simulate result: a3585bfdf71370a1ebbc033d95dd2f2c
simulate result: 82701fb21bedc1fcb73528efa7d64122
simulate result: 180b7f45ddef4b7651d7811f2daaa44e
simulate result: 37492f3d85469b95c89bb9ed73873a6e
simulate result: fd58f688df4f1e121166438d8fb54e64
simulate result: fb57743dbfa894861e9adde3efdea966
simulate result: 9da529558805752aafe8511b95a5c02a
simulate result: a5d84e2e18bbdd581ef2bd5b055d694a
simulate result: b70eef6b291d82a131d23a1ce7814ab8
simulate result: c3b246d1ec245aa64eac0e23b299c882
simulate result: 3f698634fd0b2ccfb117713fae29a9f2
success!!!
- /mnt/d/zju/system/sys1-sp24/src/lab4-1/sim/testbench.sv:40: Verilog $finish
```

仿真波形



移位部分代码解释：

```
    case (state_reg)
        RDATA: begin
            out_valid <= 0;//此时停止输出数据
            if (in_valid && in_ready) begin//如果准备好接受
                in_ready <= 0;//停止接受其他数据

                start_multiply <= 1;//开始计算乘法
                for (j = 0; j < Conv::LEN; j++) begin
                    vector_stage1[j] = 0;
                end
                vector_stage2.data = 0;
                vector_stage2.valid = 0;
                state_reg <= WORK;//切换至工作状态
            end
            else begin
                state_reg <= RDATA;
            end
        end
```

```
TDATA: begin
    reg [Conv::WIDTH*2-1:0] sum/* verilator split_var */;
    sum =vector_stage1[0] + vector_stage1[1]+vector_stage1[2]+vect
    //此处并未运用所给的并行树算法
    //$display("sum = %h", sum);
    vector_stage2.data = sum;
    vector_stage2.valid = 1;
    if (out_ready && vector_stage2.valid) begin
        result = vector_stage2.data;//赋值给最终结果
        out_valid <= 1;
        in_ready <= 1;
        state_reg <= RDATA;

    end
    else begin
        out_valid <= 0;
    end
```

解释仿真测试样例和下板的顶层结构为什么满足 valid-ready 握手协议。

在仿真测试程序中，Testbench 模块通过 in_valid 信号指示输入数据的有效性，通过 in_ready 信号指示 ConvUnit 模块是否准备好接收数据。当 Testbench 模块将 in_valid 置为 1 时，表示输入数据有效，ConvUnit 模块接收到有效数据后

将 in_ready 置为 1，表示已准备好接收下一组数据。这样，通过 in_valid 和 in_ready 信号的配合，有效地进行了数据传输和处理。

在 top.v 中，DataGenerator 模块负责生成输入数据，并通过 in_valid 信号指示数据的有效性，ConvUnit 模块根据 in_valid 信号判断是否有数据输入，并通过 out_valid 信号指示处理结果的有效性，同时通过 out_ready 信号指示 ConvUnit 模块是否准备好接收下一组数据。这样，通过 in_valid、out_valid 和对应的 ready 信号的配合，也实现了有效的数据传输和处理。

ConvUnit 模块被划分为 Shift 模块和 ConvOperator 模块，模块间用 valid-ready 协议传递数据。请思考能否对 ConvOperator 作类似上述的模块分割和数据交换，并给出这样分割后可能带来的性能提升。（bonus）

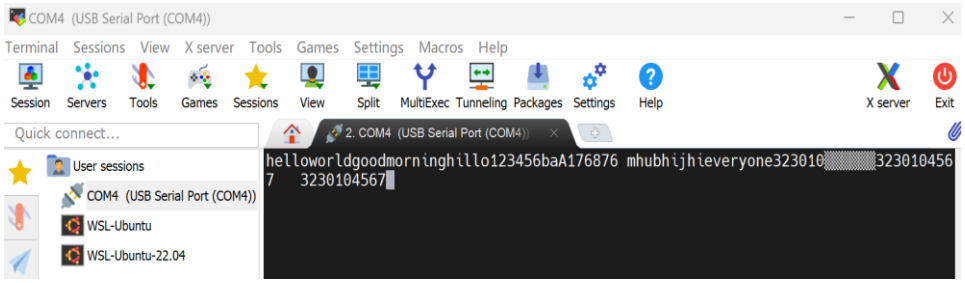
可以，convoperator 还可以划分为进行乘法的模块和进行卷积的模块，并且二者之间可以采用 valid-ready 协议进行数据传输。这样子二者可以同时计算，有利于提升效率。

Lab4-2 Report

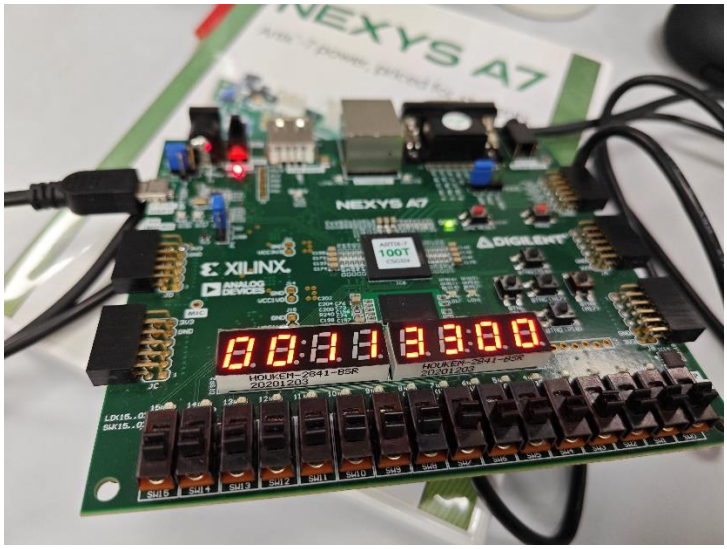
1. 仿真输出 success

```
receive data c4
receive data 9c
receive data 02
transmit data c4
receive data e4
transmit data 9c
receive data 78
transmit data 02
receive data bc
transmit data e4
receive data b6
transmit data 78
receive data e4
transmit data bc
receive data b7
transmit data b6
receive data 53
transmit data e4
success!!!
```

Mobaxterm 中输出学号



上板结果：输出输入字符的 ASCII 码



```
ays_ff @(posedge clk or negedge rstn) begin
 if (!rstn) begin
     rdata_valid <= 0;
     send_debug_data <= 0;
 end else begin
     if (debug_send) begin//如果有debug数据
         rdata <= debug_data;//将debug数据赋值给rdata
         rdata_valid <= 1;
         send_debug_data <= 1;
     end

     else if (uart_rdata.valid && !send_debug_data) begin//如果有uart
         rdata <= uart_rdata.data;//将uart数据赋值给rdata
         rdata_valid <= 1;
     end

     if (uart_tdata.ready && rdata_valid) begin//如果uart_tdata准备好
         rdata_valid <= 0;
         send_debug_data <= 0;
     end
```

代码解释

2. 设计 async_transmitter 的有限状态机，并描述 async_transmitter 的大致工作流程

等待状态（IDLE）：在这个状态下，模块等待 TxD_start 信号的触发。当 TxD_start 信号为 1 时，状态机转移到发送开始位状态。

发送开始位状态（START）：当 TxD_start 信号为 1 时，状态机进入此状态。在这个状态下，TxDBusy 输出被置为 1，表示开始发送数据。然后状态机转移到发送数据位状态。

发送数据位状态（TRANSMIT）：在这个状态下，模块依次发送 8 位数据。每个数据位持续的时间可以通过计时器来控制。状态机在发送完 8 位数据后转移到发送停止位状态。

发送停止位状态（END）：在这个状态下，模块发送两个停止位。每个停止位的持续时间同样通过计时器来控制。发送完停止位后，状态机返回到等

待状态。

3. uart 数据线不可避免存在毛刺和电平扰动，思考 async_receiver 可以用什么办法来规避接受数据的毛刺

1.滤波器：使用滤波器来消除电平扰动和毛刺。

2.时钟同步：确保接收端的时钟与发送端的时钟同步，这样可以最大程度地减少由于时钟不同步引起的毛刺和扰动。

3.数据校验：在接收数据时，使用校验来验证接收到的数据的完整性。如果接收到的数据包含错误，可以丢弃或者请求重新发送。

4.输入缓冲区：在接收数据时使用输入缓冲区来存储接收到的数据，然后在缓冲区中进行数据处理和校验。这样可以减少由于电平扰动和毛刺导致的数据丢失或错误。