The circuit below multiplies two unsigned numbers stored in the A and B registers, leaving the result in the LOW and HI registers. The four control signals, *shiftA*, *shiftB*, *shiftHI*, and *shiftLow*, are shift-enable signals for the registers. Each register is n-bits wide, and right-shifts on the rising-edge of the clock when its respective enable signal is asserted. The fifth control signal *reset*, when asserted, clears the flip-flop on the rising edge of the clock.

Assuming that the A and B registers are pre-loaded with the multiplicand and multiplier, respectively. Also, the HI and LOW registers, and flip-flop is initially reset to 0. Fill in the table, with 1’s and 0’s to indicate the necessary control algorithm. Each line in the table corresponds to one clock cycle (think of them as states in a FSM). When you assert a control signal in a row in the table, it will be available on the rising edge of the clock for that cycle. You may repeat a state or range of states by using the space at the end of each row as follows: “repeat $s_i,s_j,k$”, which means repeat from state $i$ to state $j$, $k$ times. Any “repeat” statement within a repeat range will also be repeated.

<table>
<thead>
<tr>
<th>reset</th>
<th>shiftA</th>
<th>shiftB</th>
<th>shiftHI</th>
<th>shiftLOW</th>
<th>selectSum</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bit-serial Multiplier

The easy way to look it is as follows (let n = 4):

```
1011  // registerA
x 0101  // registerB
-------
1011   0000
  1011  0000
-------
0110111
```

// We know that for each bit in B register, we need to shift A register n times, and choose carry at the end of each n
// iterations. But each bit for the LOW register only available after each n-shift of HI register; otherwise, the bits
// for the LOW register would be off. So we have:

```c
for (i = 0; i < n, i++) {
    For (j = 0; j < n; j++) {
        reset = 0,
        shiftA = 1,
        shiftB = 0,
        shiftHI = 1,
        shiftLOW = 0,
        selectSUM = 1;
    }
    reset = 1,
    shiftA = 0,
    shiftB = 1,
    shiftHI = 1,
    shiftLOW = 1,
    selectSUM = 0;
}
```

// The main part is to correctly setup the inner and outer loops. Note that either register A or
// register B can be the multiplier. So this designation has to be consistent for both inner and
// outer loops.