Instructions:

Solve one of the following two bonus problems:

Analytical Problem or Implementation Problem.

If you solve one of these problems, you may still obtain up to 50% of points for solving the other problem.

The Implementation Problem requires knowledge of:

- VHDL or Verilog, and
- Xilinx FPGA tools.

Common Background:

The Add-Add-Multiplex (AAM) Carry Select Adder is a variant of the Carry Select Adder, optimized for implementation using FPGAs.

Its block diagram is shown in Fig. 1.

![Fig. 1 Block diagram of the Add-Add-Multiplex (AAM) Carry Select Adder.](image)

In this block diagram:

- + stands for the Ripple Carry Adder
- CCC stands for the Carry Computation Circuit
- CR stands for the Carry Recovery Circuit
- \(X_i, Y_i\) are \(w\)-bit words of operands \(X\) and \(Y\), with \(i=0..k-1\)
- \(R_i\) are \(w\)-bit words of the sum \(R\) for \(i=0..k-2\)
- \(R_{k-1}\) is a concatenation of \(c_{\text{out}}\) and the most significant \(w\)-bit word of the sum \(R\).
The Carry Computation Circuit (CCC) is implemented using a k-2 bit Ripple Carry Adder. It takes as inputs:

\[ X' = c^0\_k \cdots c^0\_2 \cdots c^0\_1 \]
\[ Y' = c^1\_k \cdots c^1\_3 \cdots c^1\_1 \]

and \( c_0 \),

and produces as outputs

\[ S = s\_k \cdots s\_3 \cdots s\_1 \]

and \( c\_k \), such that

\[ (c\_k, S) = X' + Y' + c_0. \]

Please note that according to the notation used in this diagram \( c_0 \) is a different signal than \( c_{in} \) of the entire adder.

The Carry Recovery (CR) circuit is described using the following equation:

\[ c_i = c^0_i + c_{i-1} c^1_i = c^0_i + (s_i \oplus c^0_i c^1_i) c^1_i \]

The first part of this equation is based on the truth table of each stage of CCC, shown in Table 1.

<table>
<thead>
<tr>
<th>( c^0_i )</th>
<th>( c^1_i )</th>
<th>( c_i )</th>
<th>Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Annihilate</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( c_{i-1} )</td>
<td>Propagate</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>–</td>
<td>Impossible</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Generate</td>
</tr>
</tbody>
</table>

The second part is based on the dependence of a single stage of a Ripple Carry Adder:

\[ s_i = c_{i-1} \oplus p_i = c_{i-1} \oplus \overline{c^0_i} c^1_i \]

The value of the propagate signal \( p_i \) is computed based on Table 1.

Internal structure of the Carry and Control logic used to implement one stage of a Ripple Carry adder is shown in Fig. 2.
Analytical Problem

Assumptions:

The + and CCC units from Fig. 1 are implemented using Carry and Control Logic shown in Fig. 2.

The multiplexers and CR units from Fig. 1 are implemented using LUTs from Fig. 2.

If feasible, based on the number of inputs and outputs, a 2-to-1 MUX and the CR unit may be implemented together using a single LUT.

Assume the following values of individual delays for each FPGA:

For Virtex 4:

\[ d_{LUT4} = 1.00 \text{ ns} \]
\[ d_{XOR2} = 0.25 \text{ ns} \quad \text{(applies only to XORCY shown in Fig. 2a)} \]
\[ d_{MUX2} = 0.1 \text{ ns} \quad \text{(applies only to hardwired multiplexers, shown in grey in Fig. 2a, such as MUXCY)} \]
\[ d_{FF} = 0.25 \text{ ns} \]
setup = 0.25 ns

In order to exit the carry and control logic, c_{out} of each Ripple-Carry Adder, must pass as c_{in} through XORCY and two hardwired MUXes of the next adder stage (which does not perform any computations), as shown in Fig. 2a.

Delays of all interconnects are assumed to be equal to zero.

For Virtex 5:

\[ d_{LUT6} = 0.75 \text{ ns} \]

\[ d_{XOR2} = 0.20 \text{ ns} \quad (\text{applies only to XORCY shown in Fig. 2b}) \]

\[ d_{MUX2} = 0.075 \text{ ns} \quad (\text{applies only to hardwired multiplexers, shown in grey in Fig. 2b, such as MUXCY}) \]

\[ d_{FF} = 0.20 \text{ ns} \]

\[ t_{setup} = 0.20 \text{ ns} \]

In order to exit the carry and control logic, c_{out} of each Ripple-Carry Adder, must pass as c_{in} through XORCY and two hardwired MUXes of the next adder stage (which does not perform any computations), as shown in Fig. 2b.

Delays of all interconnects are assumed to be equal to zero.

Task 1

Derive the formulas for the latency of the Add-Add-Multiplex (AAM) Carry Select Adder in

A. Virtex 4 FPGAs
B. Virtex 6 FPGAs.

as a function of the operand size n and the word size w. Note that k=n/w.

Assume that the adder is surrounded by registers, and include the delay and the setup time of a register in your formulas for latency of this adder.

Task 2

Draw a graph in MS Excel showing the latency of this adder as a function of w for the following values of n and the range of values for w.

- n=512, 1024, 2048
- w=8, 16, 32, and 64.

Task 3

Find analytically an optimal value of the word size, w, for which a 1024-bit AAM Carry Select Adder has the smallest latency. Compare the results for Virtex 4 and Virtex 5.
Implementation Problem

Task 1

Describe the Add-Add-Multiplex (AAM) Carry Select Adder in VHDL or Verilog.

Use generics (in VHDL) or parameters (in Verilog) for the operand size n, and the word size w. Use "++" for the description of ripple-carry adders.

Task 2

Surround your adder with a w-to-n SIPO (Serial-Input-Parallel-Output converter) for inputs X and Y, and an n-to-w PISO (Parallel-Input-Serial-Output converter) for output R.

Store $c_{in}$ and $c_{out}$ in D flip-flops.

Task 3

Synthesize and implement your circuit for $n=1024$, and $w=8, 16, 32, \text{ and } 64$, targeting the following two families of FPGAs:

A. Virtex 4 FPGAs
B. Virtex 6 FPGAs.

Draw diagrams showing the dependence of the latency on the value of w, separately for each of the FPGAs.

Hint: Assume that the circuit latency is equal to the minimum clock period returned by the tools.