

Digital Design Laboratory

Carry-Look-ahead Adder

Circuit Delays and Timing Simulation

Purpose

1. To Understand delays in digital circuits
2. To Design a 16-bit carry-look-ahead adder
3. To Use timing simulation to measure its delay

Background: Circuit Delays

In this lab, we focus on the speed at which a circuit operates. The speed of a digital circuit is very important, as it will determine the maximum frequency at which it can work. Let us consider a PC that has a clock frequency of 800 MHz. That means that each 1.25 ns (i.e. period $T=1/\text{frequency}$) the PC will perform a computation! As we will see, this will require clever circuit design.

You may ask yourself what determines the speed or the maximum frequency of a digital system. The answer is the **delays** of the circuits. There are several factors that contribute to the delay. One is the **propagation** delay due to the internal structure of the gates, another factor is the loading of the output buffers (due to **fanout** and **net delays**), and a third factor is the logic **circuit** itself.

1. Propagation delay

When the input signal of a gate changes, the output signal will not change instantaneously as is shown in Figure 1 below.

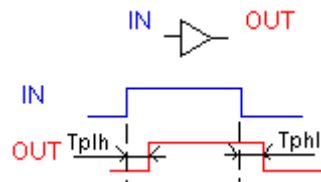


Figure 1: Propagation delay of gates

The propagation delay (or gate delay) of a gate is the time difference between the change of the input and output signals. There are two types of gate delays, T_{PHL} and T_{PLH} , as indicated in Figure 1. The value of the propagation delay varies from gate to gate and from logic family to family. In general the more you are willing to pay for a device (or chip), the faster it will be. The actual delay depends on the way the logic gates have been mapped into the LUTs (Look up table) of a CLB (Configurable Logic Block).

2. Fanout and net delays

The propagation delay described above is caused by parasitic capacitors inside the gates and the physical limitations of the devices used to build these gates. Another cause of delay is the capacitor associated with the loads seen by a gate. These capacitors are the result of the wiring (net delays) between gates (e.g. a long metal line connecting two gates on a chip) and the input capacitor of the gates as is shown in Figure 2a.

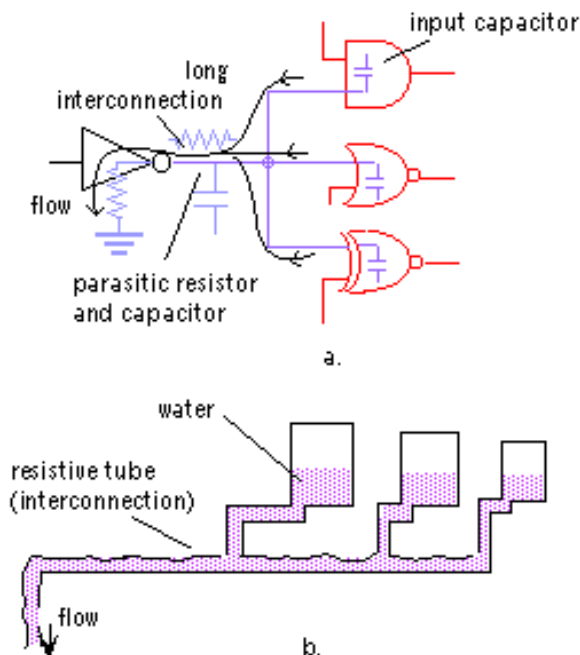


Figure 2: (a) Parasitic interconnection capacitors and fanout of a gate; (b) hydraulic equivalent.

These capacitors need to be charged or discharged through the gate that drives them (e.g. gate 1 in Figure 2a). The more capacitors that need to be charged or discharged the longer it will take for the output to change. Also, the longer the interconnection, the more resistance the nets will have. The easiest way to visualize this is to use a hydraulic equivalent of a capacitor and a resistor: a bucket filled with water and a

narrow pipe, respectively, as shown in Figure 2b. The more buckets connected to the drain (i.e. the input inverter), the longer it will take to empty them. This delay is the result of the fanout of the inverter.

3. Delay as a result of circuit topography.

Circuits that perform the same function can vary significantly in their speeds. A good example is an adder circuit. The one you designed in the previous lab is called a *ripple-adder* and is considerably slower than a *carry-look-ahead adder*, or *CLA* [1,3,4].

Measuring Circuit Delays

The overall speed of a digital system can be measured on an oscilloscope by comparing the input to the output signals. However, during the design phase, the circuit has not yet been fabricated and therefore, cannot be measured. In that case it is possible to determine the delay of circuits by doing a *Timing Simulation* (also called a *post-place & route* simulation). The advantage of a simulation is that one can also determine the delay of internal nodes of a circuit. This can be very helpful to understand which nodes or paths are the slowest and thus limit the overall speed of the circuits. These paths are called "Critical path". It is important to understand which paths are critical in a circuit so that one can reduce their delay.

The actual delay will depend on how the gates have been implemented in the various LUTs and CLBs. Also, the routing of the signals between the different CLBs determines the overall speed. Thus, one needs first to **implement** the design so that one can provide the Timing Simulator with the block and routing delay information. The Timing Simulator and the Implementation Tools are described in the [tutorials](#).

Ripple-carry vs. Carry-look-ahead Adders

One type of circuit where the effect of gate delays is particularly clear, is an ADDER. In this lab you will be measuring the delay of different types of adder circuits. The 4-bit adder you designed and implemented in the lab 2 is called a ripple-carry adder because the result of an addition of two bits depends on the carry generated by the addition of the previous two bits. Thus, the sum of the most significant bit is only available after the carry signal has rippled through the adder from the least significant stage to the most significant stage. This can be easily understood if one considers the addition of the two 4-bit words: $1\ 1\ 1\ 1_2 + 0\ 0\ 0\ 1_2$, as shown in Figure 3.

$$\begin{array}{r}
 1\ 1\ 1\ 1 \text{ --- carry bits} \\
 1\ 1\ 1\ 1 \\
 + 0\ 0\ 0\ 1 \\
 \hline
 1\ 0\ 0\ 0\ 0 \\
 \text{C}_0\ \text{S}_3\ \text{S}_2\ \text{S}_1\ \text{S}_0 \text{ --- sum bits} \\
 \text{carry-out}
 \end{array}$$

Figure 3: Addition of two 4-bit numbers illustrating the generation of the carry-out bit

In this case, the addition of $(1+1 = 10_2)$ in the least significant stage causes a carry bit to be generated. This carry bit will consequently generate another carry bit in the next stage, and so on, until the final carry-out bit appears at the output. This requires the signal to travel (ripple) through all the stages of the adder as illustrated in Figure 4 below. As a result, the final Sum and Carry bits will be valid after a considerable delay. The carry-out bit of the first stage will be valid after 4 gate delays (2 associated with the XOR gate and 1 each associated with the AND and OR gates). From the schematic of Figure 4, one finds that the next carry-out (C2) will be valid after an additional 2 gate delays (associated with the AND and OR gates) for a total of 6 gate delays. In general the carry-out of a N-bit adder will be valid after $(2N+2)$ gate delays. The Sum bit will be valid an additional 2 gate delays after the carry-in signal. Thus the sum of the most significant bit S_{N-1} will be valid after $2(N-1) + 2 + 2 = (2N + 2)$ gate delays. This delay may be in addition to any delays associated with interconnections. It should be mentioned that in case one implements the circuit in a FPGA, the delays may be different from the above expression depending on how the logic has been placed in the look up tables and how it has been divided among different CLBs.

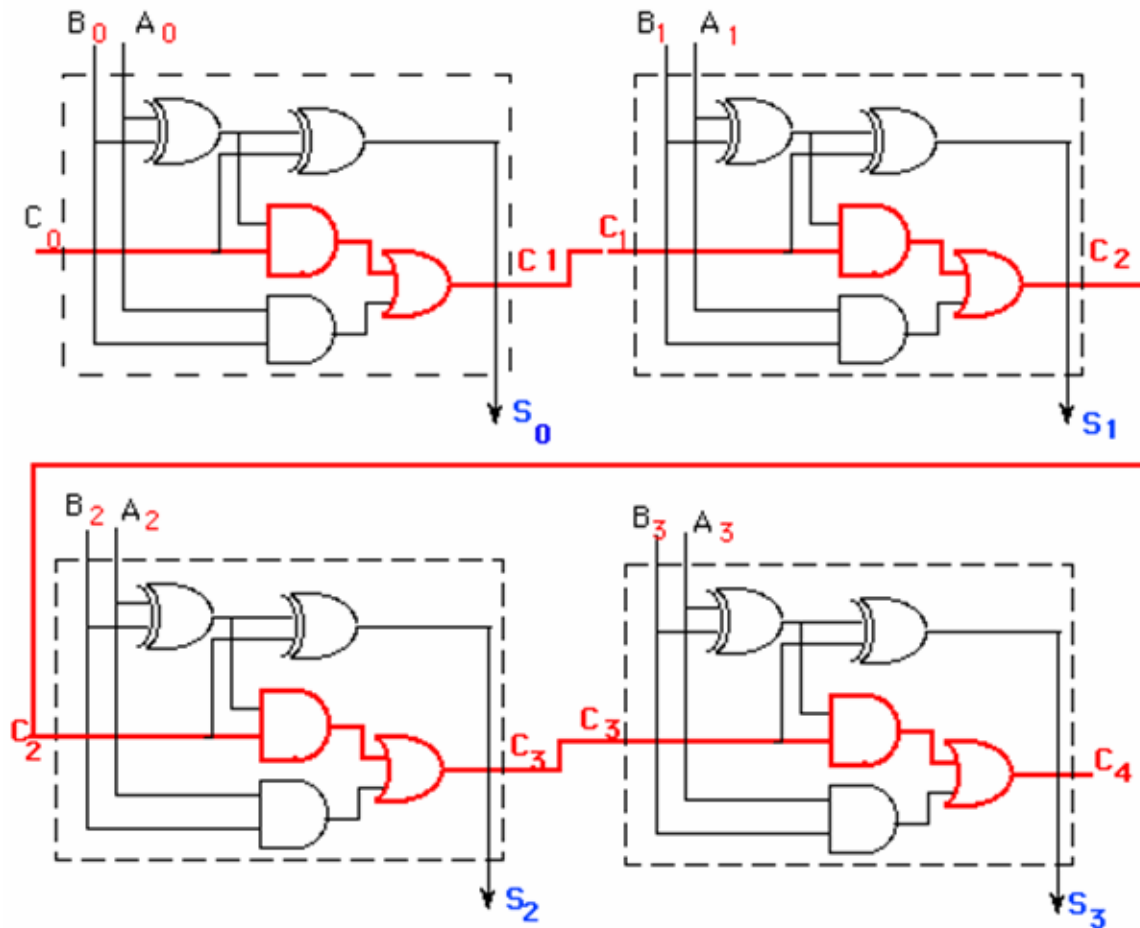


Figure 4: Ripple-carry adder, illustrating the delay of the carry bit.

The disadvantage of the ripple-carry adder is that it can get very slow when one needs to add many bits. For instance, for a 32-bit adder, the delay would be about 66 ns if one assumes a gate delay of 1 ns. That would imply that the maximum frequency one can operate this adder would be only 15 MHz! For fast applications, a better design is required. The carry-look-ahead adder solves this problem by calculating the carry signals in advance, based on the input signals. It is based on the fact that a carry signal will be generated in two cases: (1) when both bits A_i and B_i are 1, or (2) when one of the two bits is 1 and the carry-in (carry of the previous stage) is 1. Thus, one can write,

$$C_{OUT} = C_{i+1} = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_i \quad (1)$$

The " \oplus " stands for exclusive OR or XOR. One can write this expression also, as

$$C_{i+1} = G_i + P_i \cdot C_i \quad (2)$$

in which

$$G_i = A_i \cdot B_i \quad (3)$$

$$P_i = (A_i \oplus B_i) \quad (4)$$

are called the Generate and Propagate term, respectively.

Lets assume that the delay through an AND gate is one gate delay and through an XOR gate is two gate delays. Notice that the Propagate and Generate terms only depend on the input bits and thus will be valid after two and one gate delay, respectively. If one uses the above expression to calculate the carry signals, one does not need to wait for the carry to ripple through all the previous stages to find its proper value. Let's apply this to a 4-bit adder to make it clear.

$$C_1 = G_0 + P_0.C_0 \quad (5)$$

$$C_2 = G_1 + P_1.C_1 = G_1 + P_1.G_0 + P_1.P_0.C_0 \quad (6)$$

$$C_3 = G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.C_0 \quad (7)$$

$$C_4 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.C_0 \quad (8)$$

Notice that the carry-out bit, C_{i+1} , of the last stage will be available after **four** delays (two gate delays to calculate the Propagate signal and two delays as a result of the AND and OR gate). The Sum signal can be calculated as follows,

$$S_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i. \quad (9)$$

The Sum bit will thus be available after two additional gate delays (due to the XOR gate) or a total of six gate delays after the input signals A_i and B_i have been applied. The advantage is that these delays will be the same independent of the number of bits one needs to add, in contrast to the ripple counter.

The carry-look-ahead adder can be broken up in two modules: (1) the Partial Full Adder, PFA, which generates S_i , P_i and G_i as defined by equations 3, 4 and 9 above; and (2) the Carry Look-ahead Logic, which generates the carry-out bits according to equations 5 to 8. The 4-bit adder can then be built by using 4 PFAs and the Carry Look-ahead logic block as shown in Figure 5.

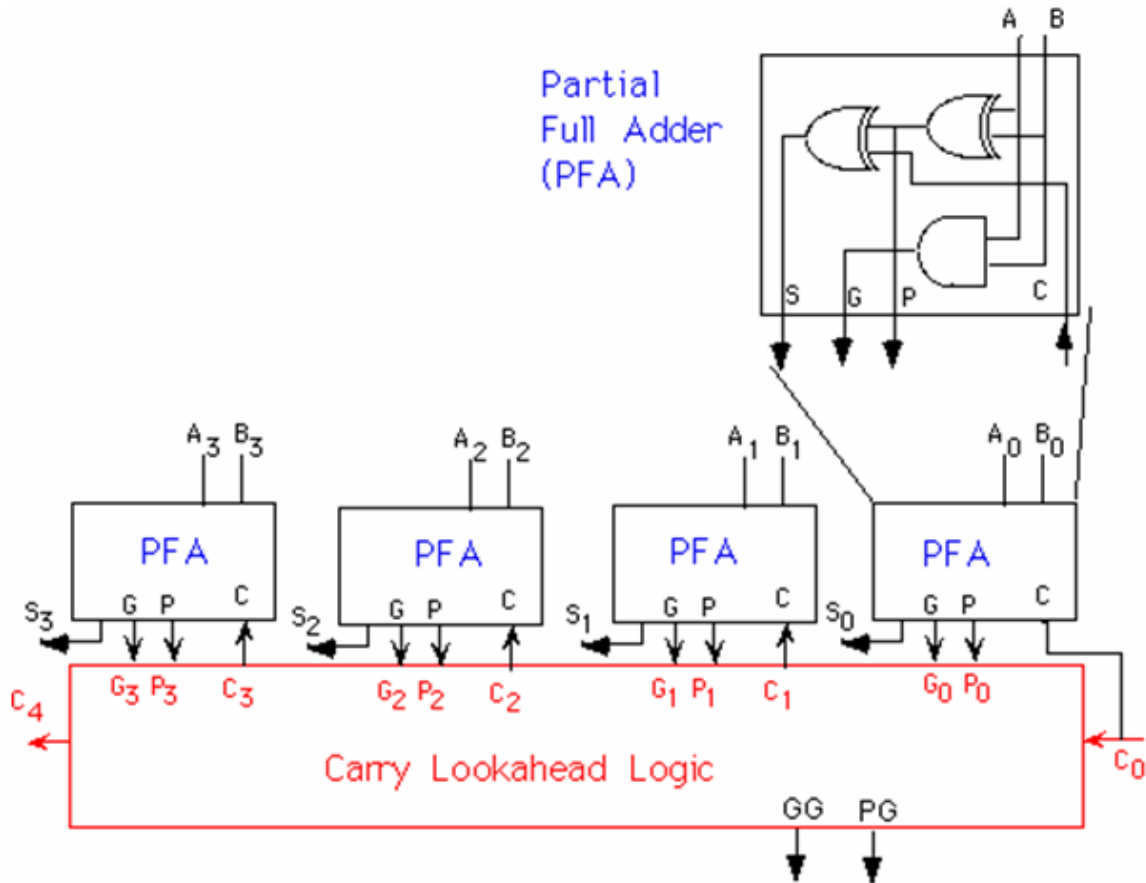


Figure 5: Block diagram of a 4-bit CLA.

The disadvantage of the carry-look-ahead adder is that the carry logic is getting quite complicated for more than 4 bits. For that reason, carry-look-ahead adders are usually implemented as 4-bit modules and are used in a hierarchical structure to realize adders that have multiples of 4 bits. Figure 6 shows the block diagram for a 16-bit CLA adder. The circuit makes use of the same CLA Logic block as the one used in the 4-bit adder. Notice that each 4-bit adder provides a group Propagate and Generate Signal, which is used by the CLA Logic block. The group Propagate P_G of a 4-bit adder will have the following expressions,

$$P_G = P_3 \cdot P_2 \cdot P_1 \cdot P_0 \quad ; \quad (10)$$

$$G_G = G_3 + P_3 G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 \quad (11)$$

The group Propagate P_G and Generate G_G will be available after 3 and 4 gate delays, respectively (one or two additional delays than the P_i and G_i signals, respectively).

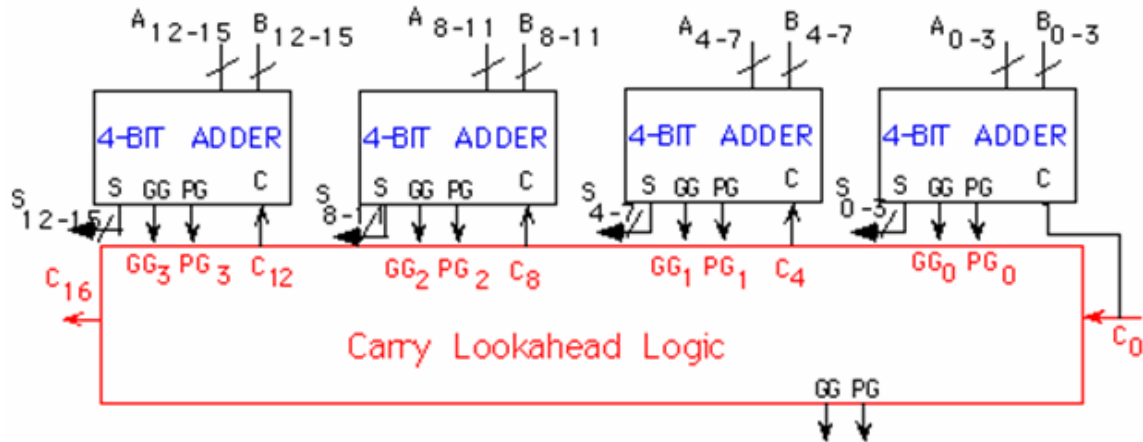


Figure 6: Block diagram of a 16-bit CLA Adder

Pre-lab assignment:

1. Describe briefly which factors determine the maximum frequency of a digital circuit.
2. What is a critical path in a digital circuit?
3. **Read** the [Tutorial](#) on Timing Simulation.
4. Assuming that one delay in a AND and OR gate is 1 gate delay and in an XOR gate 2 gate delays, determine:
 - a. What is the worst-case delay of the Carry-out of the one-bit Full Adder circuit of Lab 1
 - b. Consider the 4-bit adder you designed in the previous lab. Under what conditions of input signals does the worst case delay occur for the carry-out signal? What is the worst case delay for the carry-out (of the last stage)?
 - c. What is the worst case delay of signal S2 of a 4-bit ripple carry adder?
 - d. What is the worst case delay of signal C2 of a 4-bit ripple carry adder?
 - e. Assume you add two 16-bit words up using a ripple-carry adder. Considering worst case conditions, after how long a delay will the sum (S) be valid?
 - f. What is the delay of the Sum bits S0 and S3 of the 4-bit carry-look-ahead adder in figure 5 above?
5. **Submit your answers online using [Blackboard](#).**

In-lab assignment:

A. Parts and Equipment:

1. PC
2. Xilinx Foundation Tools 8.1i

B. Experiments:

- You will implement a **16-bit carry-look-ahead adder**. You will do this by creating a Partial Full Adder (PFA) macro and a carry-look-ahead logic macro as shown in Figures 5 and 6.
 - Open a new project in your folder (c:\users\your_name\) and call it MY16CLA. This lab will be entirely in [VHDL](#), so configure it appropriately (you may use schematics for the designs, however it is strongly recommended to use VHDL). We will be using a very hierarchical design for the CLA, so to demonstrate how to do this in VHDL an example of using a macro which contains previously defined macros is shown in the “structural description” of section 3 (Basic Structure of a VHDL File) of the VHDL tutorial.
 - Create a macro of the Partial Full Adder using VHDL. Call the macro MYPFA. When the PFA has been synthesized, do a behavioral simulation to verify that the circuit works.
 - Create a macro for the Carry-Look-Ahead Logic block. You make create this by simply adding more VHDL code to the same file you used above. Call this MYCLALOG. You should also generate the Groups Generate and Propagate signals, P_G and G_G , respectively. You do not need to create the carry C_4 , though you may if so inclined. Once again do a simulation on this circuit.
 - Now you can create a 4-bit CLA adder macro. This will contain multiple partial full adders and one logic block -- as these have been defined just now, you simply need to hook them up here.
 - When finished, synthesize the macro and simulate the 4-bit adder. Verify its operation and make a screen caption of the functional simulation waveforms (only one page).
 - You are now ready to create the 16-bit carry-lookahead adder. You may either use the top level schematic or a new macro named MY16CLA. Use the block diagram of Figure 6.
 - In order to keep the design's interconnections managable, use buses as much as possible if using schematics, and `std_logic_vector`'s as much as possible if using VHDL.
 - In VHDL, you may address multiple values in a `std_logic_vector` in a similar way to how you address a single value, for example `A(3 downto 0)` would refer to the 3..0 bits of A, even if A is defined as a 16-bit vector.

- When done, do a behavioral simulation and verify that the circuit works properly. When the simulation gives the right result, take a screen capture of the waveforms.
- When the circuit works properly, you need to **implement** it so that you can do a timing simulation later on.
 - For the implementation, see previous labs or check the [Tutorial](#) on Design Implementation. Make sure that you have your 16 bit carry-look-ahead adder selected as the top project.
 - View reports and take notes in your lab notebook:
 - Check the Map Report for the device utilization and the equivalent gate count. Check if logic has been removed during the mapping step.
 - Check the Post Layout Timing Report. Find the maximum combinational path delay and net delay. Between which input pin and output pin does the maximum delay occur? What is the maximum frequency at which the input signals can be applied without causing erroneous operation of the 16 bit adder?
- **Timing simulation (Post-Place & Route):** In the sources subwindow, change the drop-down selection to Post-Route Simulation, and simulate your test waveform, making sure that you have included possible worst-case scenarios in the waveform.
 - Apply input signals A[15:0] and B[15:0] which give a worst case scenario for the delays of the sum and carry-out signals. Measure the delays between the time the input signal is applied and the sum and carry-out signal changes (use the measurement option in the simulator). Write these numbers down in your lab notebook. Do they correspond to the ones reported in the Post Layout Timing report?
 - When doing the timing simulation, be careful that you do not change the input values before the output values have stabilized; this will mean that you can not derive any useful timing information from the scenario.
 - Take a screen capture of the simulation data, including the measurement data for the Sum and Carry-out delays.

Hand-in at the start of next lab:

Lab Report including

1. Course title, Lab no, Lab section, your name and date.
2. Section on the lab experiments:
 - A. Brief description of the lab including the goals
 - B. Copy of all VHDL source code and screen captures of all schematics created
 - C. Screen capture of the functional simulation of the 4-bit CLA adder
 - D. Screen capture of the timing simulation with measurements on the 16-bit carry-look-ahead adders.

- E. Summaries of the Map (device utilization) and Timing Reports (delays)
- F. Discussion of the measured delays using the Timing simulator.
- G. Maximum frequency at which the 16 bit CLA adder can be used.

4. Discussion and conclusions.

References:

1. R. Katz, "Contemporary Logic Design", Chapter 3, sections 3.3.1 and 3.3.2; Chapter 5, section 5.2
2. D. Van den Bout, "The Practical Xilinx Designer Lab Book", Prentice Hall, Upper Saddle River, NJ, 1998.
3. J. Wakerly, "Digital Design," 3rd edition, Prentice Hall, Upper Saddle River, NJ, 2000.
4. M. M. Mano and C. R. Kime, "Logic and Computer Design Fundamentals", 3rd Ed., Prentice Hall, Upper Saddle River, NJ, 2004.

Copyright 2006, Created by Jan Van der Spiegel Oct. 14, 1998; Updated by Carl MacKey, September 18, 2006; Sept 20., 2007.