

**EXPERIMENT 1
FUNDAMENTALS**

1. GOALS :

Learn how to develop a car alarm digital circuit during which the following are introduced :

- > *CS2204 lab fundamentals*, and
- > *Digital Circuit Design fundamentals*

2. Lab Work :

2.1. Lab Fundamentals :

The Lab will take place in CIS Lab 227RH. Each section will have three hours per week in the lab. The lab is structured to develop a term project by using the *Xilinx Foundation 4.2i software package* and the *Digilent XLA5 FPGA (Field Programmable Gate Array) board*. An FPGA chip is a “*hardware programmable*” chip that behaves like the circuit designed on the computer.

The Lab will introduce current digital design techniques and tools. It will emulate an environment where engineers grouped together as a team, design projects block by block under the guidance of a project manager. In CS2204, **two** students will work as a team. Teams are formed alphabetically.

Starting with the third lab session, the focus will be on digital circuit design. A succession of **six** design experiments will lead to the completion of the term project. Each experiment will make use of earlier experiments. Experiment 1 and 2 will introduce Xilinx software, Digilent hardware and digital circuit. Experiments 3, 4, 5 and 6 are term project phases. Experiment 6 completes the term project. Experiments will be increasingly more complex. Students are strongly suggested that they keep their experiment files and directories well organized.

The Lab will not affect the term grade directly, but “lab attention” grades that mean student’s motivation (Lab attendance, arrival/departure to/from the Lab), concentration on the experiments, how much they are able to work with their teammates and whether their circuits work, will be given. The last three design experiments will be reviewed and comments will be made.

2.2. Software Package and the Hardware Board :

In order to develop circuits, a software package from Xilinx, the Xilinx Foundation 4.2i will be used. This software is different from the one which comes with the textbook. The textbook software will **not** be used. The design will be tested on a prototyping board which is the hardware students will use in the lab. The board, the Digilent XLA5 FPGA board, makes the design process and verifying that the design works more concrete. The Digilent board contains an 84-pin *SPARTAN* FPGA chip. The circuit developed by using the software on the computer is “downloaded” to the FPGA on the Digilent board. Downloading means the FPGA chip is hardware programmed. The programmed FPGA chip characteristics are often close to those of the goal : a **custom chip**. Thus, if the circuit works correctly on the FPGA chip, one would quickly develop the custom chip from that.

2.3. Getting Started in the Lab :

Xilinx Foundation projects make use of many files as it is an industry software package. In order to help students keep track of their files, it is decided that students keep their files in one place accessible on campus : the **LABS domain** which corresponds to the **S drive** on lab PCs. The S drive is on a server accessible from laptops and any PC at Poly. In order to log on to 227RH PCs, students need to have an account for the LABS domain.

Students who do not have a LABS domain account can get it at the Information Systems (IS) help desk. **Students have to have the account by the third lab session.** Having the LABS domain account is not enough to access the S drive. This drive must be activated as well. If the drive is not active, students need to request it from the IS help desk too. The information Systems (IS) help desk :

- Room : 337 RH
- help@duke.poly.edu
- (718) 260 - 3123

In case students have 227RH lab related problems or would like to have the lab open, they contact the CS lab supervisor Mr. Keni Yip at (718) 260-3023, keni@poly.edu. His office is 225RH.

The Lab software and hardware, coupled with 3-hour lab sessions and a widely accessible project storage area are intended to provide students with a seamless digital circuit development environment.

3. Developing a Car Seat-Belt Alarm Circuit :

In order to accomplish the goals mentioned on page 1, we will develop a small digital circuit. The circuit is a car alarm circuit which sounds the alarm if the driver turns on the car engine before fastening the seat belt.

The first development cycle is the **development cycle on computers** which is the schematic (determination of components and their wiring) based on design factors. If the circuit is simple, one can completely design the circuit, the schematic, on paper. Then, the schematic is copied from the paper to the computer. If the circuit is complex, as many real-life circuits are, then the high-level design with blocks and subblocks is determined on paper. Then, the schematic (component determination and wiring) is carried out on computers. Once the schematic is complete, it is tested (**simulated**) on the computer to see if it works and satisfies product requirements, otherwise it is modified. Thus, the development cycle on computers consists of three steps :

- **logic design**
- **test**
- **modify**

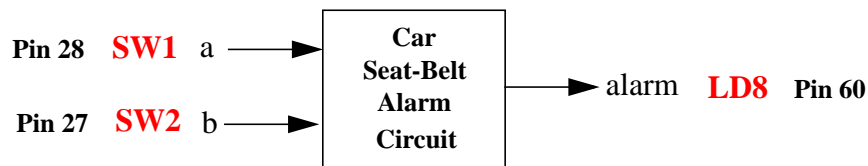
The first step of the development cycle on computers/paper is called logic design since logical concepts are applied to obtain the schematic. Logic design consists of obtaining the *precise input-output relationship* of the circuit and *implementation* of the circuit.

3.1. Development Cycle on Paper/Computers :

3.1.1. Logic Design :

3.1.1.1. The Input-Output Relationship :

The input-output relationship means that the circuit is viewed as a black box with only its inputs and outputs considered. Then, the outputs are related to the inputs. That is, we try to determine when (for which *input combinations*) an output is 1 and when it is 0. As a black box, the car seat-belt alarm circuit is shown below with two inputs and one output :



- Input “a” is connected to the engine. It is normally 0. It is 1 when “the driver turns on the engine.” The input is emulated by switch 1, SW1, which is connected to pin 28 of the FPGA chip.
- Input “b” is connected to the seat belt. It is normally 0, meaning the belt is **not** fastened. It becomes 1 if the seat belt is fastened. The input is emulated by switch 2, SW2, which is connected to pin 27 of the FPGA chip.
- Output “alarm” is normally 0, meaning no alarm. When it is 1 the alarm sounds, i.e. the driver has turned on the engine before fastening the seat belt. The output is emulated by LED light 8, LD8, which is connected to pin 60 of the FPGA chip.

Note that the pin assignment of the FPGA chip is given on page 8 of the *Development Cycle on Breadboards with FPGAs* handout.

Textually, the **operation** of the digital circuit, or the **input-output relationship** of the circuit can be specified as follows :

“The alarm sounds when the engine is turned on **AND** the seat belt is **NOT** fastened.”

An equal way to describe the input-output relationship by concentrating on the output = 1 case and without ignoring the output = 0 case is :

“The output is 1 when input a is 1 **AND** input b is 0.”

Soon we will see that often we prefer to specify all input and output values when they are 1. Then, the new input/output relationship is :

“The output is 1 when input a is 1 **AND** input b is **NOT** 1.”

Note that this is the **formal way** the textual relationship is given in digital circuit design. Another way to describe the input-output relationship is by means of a **truth table** that shows the value of the output for each input combination :

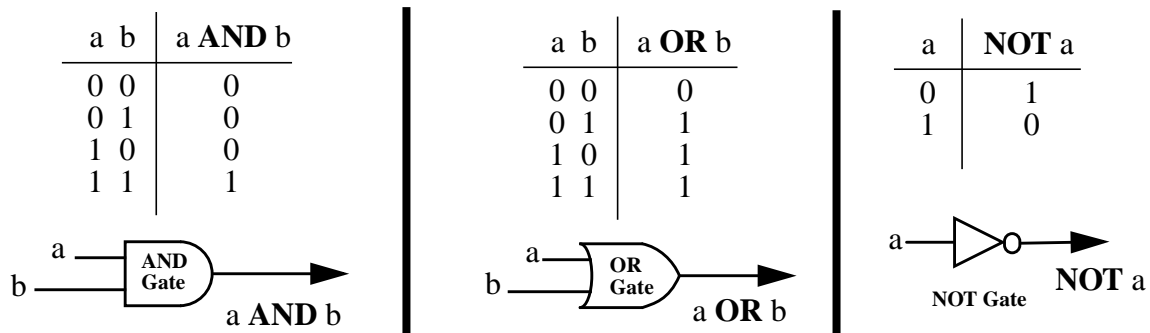
a	b	alarm
0	0	0
0	1	0
1	0	1
1	1	0

3.1.1.2. The implementation :

The implementation means the schematic (digital circuit components and their wiring) is determined based on the given product goals (factors) : **speed, cost, power consumption, reliability, size, weight**, etc. For Experiment 1, we will not focus on these goals to be able to concentrate on the components, wiring, the software, and the hardware.

Let’s start with the determination of the components : the input-output relationship indicates that we need a digital circuit that outputs 1 when it detects that input a is 1 and input b is 0.

What we are given are components (digital electronic circuits) named **gates**. We are given AND, OR and NOT gates to use. We will implement the alarm circuit by using these gates. Truth tables (input-output relationships) and symbols of these gates are as follows :



An AND gate outputs 1 if both outputs are 1 simultaneously. An OR gate outputs 1, if at least one of the inputs is 1. A NOT gate flips the input value. *Note that a gate is the simplest digital electronic circuit one has.*

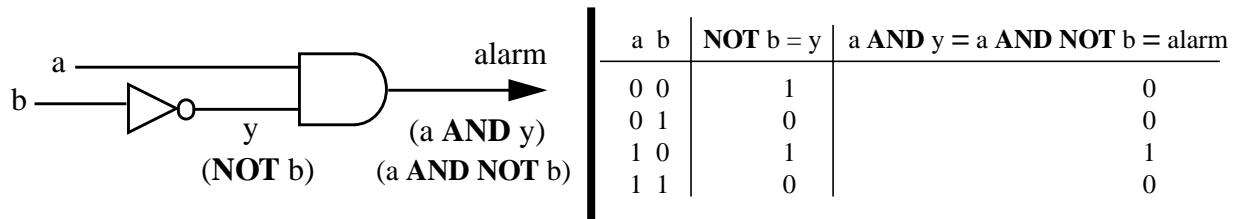
Which gates can be used and how they are connected to each other are based on the textual description given above :

“The alarm sounds when the engine is turned on **AND** the seat belt is **NOT** fastened.”

We see that in the textual description there are two sentences forming a **compound statement** by means of the word AND : one sentence is “the engine is turned on” and the other one is “the seat belt is **NOT** fastened.” We also see that the second sentence is in the negative form. Each sentence specifies what value each input should have to sound the alarm. The word that combines the two sentences, AND, indicates if the input values should be 1 simulta-

neously or not. In this case, they should be 1 simultaneously. The second sentence indicates we need to have the inverse value of the input to sound the alarm. Thus, we come to the conclusion that the two circuit inputs are ANDed **after** the second input is negated (NOT). That is, all we have to do is to use the gates implied by the words AND and NOT in the compound sentence.

We can determine the implementation (which gates, how many of each and their wiring) in a different way ! Implementing the circuit also means implementing the truth table of the digital circuit by using the truth tables of the gates. The alarm circuit truth table is similar to the AND gate truth table, except the bottom two rows. An AND gate, coupled with a NOT gate on the b input would make the bottom two rows of the two tables identical. That is, the alarm circuit truth table is implemented by the truth tables of the AND and NOT gates :



Since the input-output relationship of the circuit, (its truth table) is identical to the truth table of the car seat-belt alarm circuit, we conclude that the above circuit implements the alarm circuit. A digital circuit like the one above, consisting of gates, is called **gate network**.

There is a field in Mathematics that deals with textual description of digital circuits : **Truth-Functional Calculus**. It works on *declarative sentences* connected by *legal connectives*, forming a *truth-functional compound* (a compound statement). A declarative sentence can be either true or false. The legal connectives can be AND, OR, NOT, among others. Each sentence is assigned a variable and the connectives are represented by their symbols. For example, AND is \wedge , OR is \vee and NOT is \neg . One can then convert a long text (truth-functional compound) to a truth functional calculus *expression* with variables and symbols only. For example, the above alarm circuit expression is $(a \wedge \neg b)$.

Truth-functional calculus is concerned about when the compound (the text) is true (1, one) and false (0, zero). It does not care about the meaning of the sentences as long as they are true or false. Also, a truth functional expression concentrates on the input and output values that are equal 1 as it is clear in the above truth functional expression. Converting declarative sentences and connectives is straightforward, only for simple circuits, since the textual input-output relationship is short. Real-life problems are complex and their textual description is very difficult to convert to an expression by means of truth-functional calculus.

Now that we know what is in the black box on page 2, we end the gate network design of the alarm circuit on paper. We are now ready to move the design to the computer : drawing the schematic circuit on computer, using Xilinx Foundation software. For this, we will copy the circuit diagram from paper to computer, then simulate it and modify it if necessary. The simulation of the circuit is done on computer. Modification is the *change of the design* based on test results if errors are discovered. After the modification the circuit is simulated again to verify the circuit modification is correct.

Often logic design on paper is interleaved with or done concurrently with logic design on computers, especially for complex circuits. Therefore, starting with Experiment 4, we will stop designing them sequentially. Design on paper will be concurrent with design on computer.

3.1.1.3. Circuit Design on the Computer (Xilinx Software) :

Follow the steps given below to develop the circuit. Note that throughout the semester, steps to go through will be given where a “bullet” symbol such as “➤” that shows a new step. Key presses and mouse selections are shown in bold.

Also, throughout the semester, the handouts will be prepared by assuming that students use their **S drive** as their project storage space. If students are not able to access the S drive, for example, in a place where no network connection is possible, they should use the default Xilinx project directory which has the path “...\\Fndtn\\active\\projects...” Later, students should move their project to the S drive when they establish the connection.

Another convention to remember is that when we say “click the mouse,” we mean clicking the **left** mouse button. The right mouse click will be explicitly mentioned.

First task before starting a new project :

In order to keep our designs organized, we need to have them stored with respect to their experiment number. For each experiment, we will have a directory on the S drive. Therefore, for the current experiment, we will do the following :

- Create a new directory named “**exp1**” under cs2204.

We will have our alarm circuit project under this exp1 directory.

Starting the Xilinx software :

- On your PC, start the Xilinx Project Manager by doing one of the two below :
 - Double click on the Xilinx “**Project Manager**” icon,
 - Go through menu selections : **Start** (on the lower left corner) -> **Programs** -> **Xilinx Foundation 4.2** -> **Project Manager**.

You will see a dialogue box titled “**Getting Started**” in the foreground while the “**Project Manager**” window is in the background.

Creating a New Project :

Since we have not designed any project yet, we will start with creating a new project.

- In the “**Getting Started**” dialogue box, click on the radio button with label “**Create a New project.**”
- Click “**OK**”

A dialogue box with the name “**New Project**” will pop up.

- In the dialogue box, enter “**alarm**” as the name of your design.

We will change the directory of the project to the “exp1” directory on the S drive :

- Click on the “**Browse...**” button and select the project directory (your working directory) as the exp1 directory. All your alarm circuit files will be saved there.
- Click on the radio button with label “**Schematic**” since we will have a schematic design, not an HDL design.

Finally, there are three list boxes on the last row. They are about the FPGA chip specifically. We have to make sure that the choices are “**Spartan**,” “**S10PC84**,” and “**3**.”

- Click on the arrow in the “**Flow**” area and scroll until you see “**Spartan**.” Select it.
- Click on the arrow in the area to the right and scroll down until you see “**S10PC84**.” Select it.
- Click on the arrow in the area to the right and select speed “**3**” for the FPGA.
- Press “**OK**.”

The computer shows the “**Project Manager**” window with three panels. The upper right one which we will constantly use, shows the “flow” of the project until it completes. We are now ready to open the Schematic editor to begin the design of the alarm circuit.


The schematic design

- Click on the “**Schematic Editor**” button on the upper right panel to start the schematic editor.

A blank “design sheet” (window) will be shown. At this time, when the schematic editor is started, it is always in the “**Select and Drag**” mode.

In order to place components (such as an AND gate) on the design sheet, we need to change the mode to the “**Sym-**

bols” mode. This can be done in three different ways :

- Click on the “**Symbols toolbox**” icon on the left side of the screen : 
- Pull down the menu “**Mode**” then select “**Symbols,**” or
- Press key “**F3.**”

The “**SC Symbols**” window appears on the right side. This is the library of components we will use this semester. Scroll down the list to become familiar with it :

Let’s place one of the two components, the 2-input AND gate on the sheet :

- Click on the “**AND2**” component inside the window on the library list. Now an “AND2” gate is “attached” to your mouse and you can place it on the design sheet wherever you want.
- Drag the mouse to the left and click it in the middle of the sheet.

From this point on follow the steps shown by the professor to complete the implementation, testing and downloading.

CONTACTS :

1) Students can see the professor and teaching adjuncts (TAs), about the lectures, homework, and lab experiments.

2) Professor’s contact information :

Room : 114 LC (718) 260-3101 Fax : (718) 260-3609 haldun@photon.poly.edu

Open-door policy to see the professor. If the door is closed, he might be in the lab.

Present in the lab : Mondays (4-6), Wednesdays (3-5) and Fridays (2-4)

3) The TAs of the course are **Nikhil Joshi, Sapan Shenoy, Jeff Tao, Bo Yang** and **Peng Yao**.

4) All handout and lab files are at the course web site : <http://cis.poly.edu/cs2204>

5) When short-term problems are encountered in PC labs, students are advised to contact : help@duke.poly.edu or (718) 260- 3123 or go to Room : 337 RH.

For CS2204 lab related issues and to have the lab open, students need to contact the CIS lab supervisor Mr. Keni Yip at (718) 260-3023, keni@poly.edu. His office is 225RH

For longer-term problems in PC labs and **any other matter**, students should not hesitate to contact the professor and TAs.