**Lie Detector**

**ECE 287 Final Project**

Dr. Yamuna Rajasekhar

Ashley Youngwirth & Zane Shreve

**PROBLEM STATEMENT:**

In many situations, such as an interrogation being instigated in a criminal investigation, being able to identify if a person is telling the truth is invaluable. A lie detector is a device that aims to determine whether or not a person is being truthful by measuring and analyzing physical changes of the body. The most challenging part of this process, especially when working with digital equipment, is the actual measurement of the physical signals. Signals from the body must be digitized then analyzed, with the end result being a lie detector that can tell the difference between a person who is lying and one that is telling the truth.

**BACKGROUND:**

When a person lies, their body undergoes physical changes which can be monitored to distinguish false statements from those that are true. A lie detector test uses these changes to determine the validity of a test subject’s statements. A common application of such a test is during an interrogation. In such a situation, the psychological stress associated with guilt amplifies these physical changes, making it very easy to detect the truth from a lie. The most noticeable change is an increased heart rate, which can be monitored by measuring a person’s pulse at any number of points on the skin’s surface. Another giveaway that a person is not telling the truth is increased perspiration. Typically, this occurs on the subject’s hands in the form of sweaty palms and fingers. This perspiration increases the conductivity of the skin of the test subject, another easily measured change.

A lie detector, then, uses the results of these measured changes to predict if the test subject is lying. If the heart rate increases and skin conductivity does as well, then the subject is almost surely lying and a lie detector would notify the user that a lie was told. If just one of the two changes occurs, then there is a likelihood that a lie was told, but the certainty is not as high as when both changes are present.

**DESIGN:**

MEASURING ANALOG SIGNALS (ARDUINO):

To create a lie detector that is dependent upon both heart rate and skin conductivity, each of these analog signals needs to be independently read and digitized. To accomplish this, two Arduino microprocessors are used to accept the analog input signals and send out the appropriate digital responses. To measure skin conductivity, a circuit was designed with two open wires. The Arduino code reads the resistance between the wires and outputs a high signal when resistance is below a threshold value. (Because resistance is the inverse of conductance, conductance increases when resistance decreases.) The threshold was determined by comparing resistance values of dry versus moist skin between the wires. The signal sent by the Arduino is sent to a GPIO (general purpose input/output) pin on the FPGA board. In the case of measuring heart rate, a piezo sensor can be used to detect small knocks, such as in the case of a pulse. A simple Arduino circuit uses the piezo sensor as an analog input and outputs a high signal when a pulse occurs, designated by when the analog value of the sensor is greater than a resting threshold. This threshold was determined by trial and error, as the sensor can be sporadically overly-sensitive. The output signal from the Arduino is fed directly into the FPGA in the same way as the skin conductivity signal.

READING DIGITAL SIGNALS (ALTERA DE2-115 FPGA):

The digital signal that corresponds to a change in skin conductivity needs no manipulation to be used in a simple combinational logic circuit. The heart rate signal, however, due to the nature of its measurement, requires additional sequential logic before it can be used as an indicator of change. Firstly, a baseline heart rate must be determined. A digital timer and an up-counter are implemented to allow the FPGA to record the number of input signals (pulses or beats) that occur in six seconds. This value is stored as a 4-bit binary number and used as the baseline. When questions are asked and test subject begins answering, their heart rate is measured again via the same circuit for six seconds. This value is also stored as a 4-bit binary number in a register. This measured value and the previously-measured baseline are both fed to a magnitude comparator. If the measured heart rate is greater than the baseline, then an increase has been detected. This output from the magnitude comparator is now on the same functional level as the skin conductivity signal.

DETECTING A LIE (ALTERA DE2-115 FPGA):

With the signals we have now acquired, one from the skin conductivity circuit that goes high when skin conductivity increases, and one from the heart rate logic that goes high when pulse increases, it is easy to use these two parameters to detect a lie. If only one of these two signals is high, then there is a possibility of the person telling a lie. A “maybe lying” LED turns on, showing that the likelihood of the person telling the truth has decreased. To detect if either of these signals is high, an XOR gate is used. XOR is used rather than OR because of the next part of the logic: If both signals are high, which is detected by a separate AND gate, then a different LED turns on, this time to signify that the person is indeed lying. If both signals are low, no LED turns which signifies that the subject is telling the truth.

**RESULTS:**

Through the design process, we were able to stay fairly close to our original design. The skin conductivity circuit works like a charm, using the Arduino to send a high signal to the FPGA when fingers are wettened. Because it is difficult to simulate a real interrogation where a lie detector would be most effective, fingers moistened by water work to tell the difference between low and high skin conductivity. The sequential logic behind the heart rate sensor sticks to the plan as well. A “baseline” switch is thrown to collect the original heart rate reading based on the timer and up-counter described above, then this value is stored in a register. This switch is turned off, and the “measured” pulses (current heart rate) can begin to be counted on the next cycle of the timer. This measurement works just like the baseline, storing the finished value in a register when the timer goes low. The two values from the registers are then compared using a magnitude comparator, with the output going to the XOR and AND gates as described above.

The changes that we had to make to the design, for the purpose of using the best equipment we had available to us, came in the form of recording heart rate. Rather than using a finicky piezo sensor whose behavior we could never get a grasp on, we decided to simply simulate heart rate through the use of switch on the FPGA. This got rid of the need for one of the Arduino code blocks as well. Along with this, our logic requires a “reset” switch, as part of the SR Latch we used for switch-debouncing purposes. This switch must be thrown manually after every simulated heartbeat.

We also went through a number of changes with implementing the FPGA using the design software, Quartus II. Originally, we planned on describing our logic in VHDL, and came up with early attempts for the counter and timer. However, implementing the switch debouncer in VHDL seemed cumbersome, so we decided to convert our VHDL code to user-defined blocks to be used on the Block Diagram/Schematic option on Quartus. This worked alright, but we quickly decided that the counter we designed had some issues, which were fixed when we used a pre-built counter block from the Quartus library. Keeping with the Block Diagram, we were able to quickly and easily implement the two required registers and the magnitude comparator from library blocks as well. Throughout all of this, though, we retained the VHDL code block for the timer system in the final Block Diagram.  **CONCLUSIONS:**

While our finished version ended up being a much more simplified and stripped down representation of what we had in mind, it was the best attempt we could make with the limited resources and equipment that we had. We would have liked to have been able to use a pre-bought heart rate monitor that would have gotten rid of the need for Arduino and the switch debouncing circuit. However, due to price, we initially decided on the piezo and found that to be not nearly sophisticated enough for our needs.

It would have also been a nice touch to have been able to use the serial monitor for some sort of visual representation of the measurements we took, but due to time constraints, this became our lowest priority and we were not able to get around to it. We also would have liked to have taken the output from the XOR and AND gates and done something more flashy with them, but settled for simple LEDs on the FPGA board instead. Despite this, we believe that the finished product is an accurate simulation of a lie detector, with logic that is sound and sensors that are the best that we could muster.