Hypersonic Security System
Lab Report

Christian Deonier & Rich Lean
May 12, 2005

TA: Jenny Lee
6.111 Introductory Digital Systems Laboratory

The Hypersonic Security System (HSS) is a multi-feature security system designed to protect medium to small sized objects. HSS uses a combination of security features to protect the item. This system uses a stationary laser net to surround the object on all sides; if the laser net is broken, then the alarm is triggered. HSS also uses a pressure switch to detect differences in weight of the protected object. Additionally, HSS utilizes a motion sensor to detect if any unwarranted motion around the object occurs. This system also uses an audio sensor to detect noises around the system. Finally, a combinational numeric pad is in place, where one can enter a code and turn off the security system or disable different components of the system.

Introduction

Security is necessary in today's day in age to protect the things that we cherish the most. In order to protect ourselves, we need safe and fundamentally sound digital security systems. This is where the Hypersonic Security System (HSS) comes into play. The system we propose is intended to protect small to medium sized objects, namely laptops, briefcases, or Jenny's coffee. The best way to do this is to provide defense in depth. HSS makes use of a combination of security features. We use a two foot cube as our "digital safe" in this prototype, and a laser net is implemented to prevent penetration on any exposed face. A pressure switch is used to detect differentials in weight. Also, a motion sensor is used to prevent any unnecessary movement around the protected object. Sound can also be detected by our system, so this adds an additional layer of safely. This entire system is controlled by a simple keypad interface, allowing the user to turn on and off any system he desires.

System Overview

The higher level system, or control module (Figure 1), coordinates all of the different security modules to act together to protect against a threat. The control module coordinates the motion detector and video module, the sound detection module, and the inputs coming from the secondary labkit. The control module is abstracted away from the modules, and uses the modules' alert signals to detect if there is an alarm.

The control module has a number of components to control the system. It has several alert registers that the modules pulse to if there is an alarm. The major FSM (Figure 2) of the control module cycles through each register to see if a module sent an

alarm. If the FSM determines an alarm was triggered, the FSM uses the alert modules to sound an alarm. The control module also has a synchronizer for the lasers and pressure switch pulses. Because the signals from the lasers and pressure switch are coming from the secondary labkit which is not running on the same clock as the system, the signals are asynchronous and must be synchronized.

A nice feature of the control module is the robustness and easy expandability. It is very easy to add in more security features; one needs merely to add an alert register and a state to the major FSM to add the feature. This is why it was very quick to incorporate the sound detector in such a short time. The control modules extensive use of abstraction and modularity makes the system very extendable.

Module Description/Implementation

Motion Detector

The motion detector is designed to detect significant motion in the proximity of the object. The motion detector uses an IR camera that sends an NTSC signal to the labkit. The motion detector module (Figure 3) analyzes the signal from the camera to determine motion. If motion is detected, the alarm is triggered.

The motion detector algorithm is fairly simple. The idea is that each image is composed of many pixels, and each pixel has a certain luminance value. The image analyzer parses through the decoded video stream from the ADV7185 and extracts out each pixel's luminance value. The image analyzer keeps a running sum of all the luminance values in an image. Once the image is complete, the image analyzer compares the total luminance value with that of the next image's total luminance value to determine

if there is a significant change. The assumption is that any motion produces a change in the total luminance. If there is a significant change in the total luminance, motion has been detected, and the alarm is triggered, and the module pulses motion_pulse.

The key to the process was producing a correct FSM (Figure 4) to parse the data and do comparisons. The decoded NTSC signal has a standard to follow, where data is transmitted one value at a time. A line of the image is marked by a timing reference signal (TRS), as is the end of the active video. The FSM runs off of the clock from the decoded signal; the specification is that data comes every clock cycle of the decoded signal, so it makes sense to run asynchronously from the rest of the system. In particular, after the TRS, the luminance values start coming in every other clock cycle.

The FSM parses through the decoded camera signal to extract out the luminance values. The FSM uses the TRS to determine where exactly the FSM is in the decoded signal. The FSM looks for the TRS which signals the start of the new line, then uses the TRS to determine what line it is, an even or an odd line. Right after the TRS, the FSM starts adding all of the pixel luminance values in the running sum for the whole image. The FSM keeps track of the amount of lines processed for even and odd lines. When the FSM determines it has all the even and odd lines of an image have been written, the FSM compares its final total luminance value to the previous images total luminance value. If the difference is beyond a certain threshold, the motion detector module sends an alarm pulse to the system.

The motion detection scheme is simple, but there are certain tradeoffs. The most obvious problem to the algorithm is that it is very sensitive to light levels, and may go off when there is really not a threat. For instance, if one flipped off the light while the

system is on, the luminance values would all drop, and that change is detected as "motion". Additionally, the motion is not very specific, the motion detector determines if there is a net change in the image that it sees, but not if there is a very specific, local change, say in the corner of an image. A more sophisticated system would probably use edge detection, or something else that determines localized changes in the image. Additionally, the present algorithm is not effective for detecting very slow movements.

Keypad

The external keypad interfaces with the labkit, allowing the user to access the security system. The user needs to correctly enter a code to gain access to the system, and once she has access, she has complete control of the system. The user can enable and disable different components of the system, while leaving other components unaffected. The user can also reprogram a different four-digit code from the default code.

When the user hits a key on the keypad, the keypad generates a clock signal and a serial data signal. The keypad sends a start bit, eight data bits, a parity bit, and a stop bit. The keypad module (Figure 5) takes this asynchronous data and translates the key into an action. The keypad interface takes the asynchronous data and generates a parallelized signal, eight bits wide. This is accomplished through a trivial FSM that steps through all of the bits to get the data bits we are interested in. The FSM operates off of the keypad clock, thus is asynchronous to the rest of the system. The clock signal and data signal from the keypad are registered to make sure no glitches cause problems.

Once the keypad interface has the complete eight-bit keycode signal, it pulses a ready signal and the keycode to the synchronizer, which synchronizes the still

asynchronous signal to the rest of the system. The keypad decoder takes the synchronized keycode and keycode ready signals and determines what number the user hit, and sends the value and a ready signal to the keypad FSM (Figure 6), which keeps track of the state of the system. The FSM outputs several signals the rest of the system uses to determine what components are activated: system, motion, lasers, scale, and sounds.

Sound Detector

The sound detector triggers an alarm if there is too much noise around the object. The sound data comes from a microphone on the camera. The sound detector essentially uses two signals: the sound data and a ready signal which are generated from an external source. The sound data is an effective way to determine the relative sound level of the environment. The sound data is a twenty-bit signal. The sound detector samples the sound data when the ready signal is high. If the sound is above a certain threshold, the sound detector triggers an alarm by pulsing sounds_pulse.

Laser Net

The lasers are powered by the FPGA and are propagated along every side of the cube. A phototransistor is located on each face to collect the laser signal. During normal operation, the FSM for this module checks to see if every laser has made it to the each sensor. If one or more lasers are blocked, then an alarm signal is sent. Additionally, the user can use switches to tell the system to ignore input on any side. This allows for the user to easily access the object being protected.

This module has five inputs and two outputs (Figure 8). The inputs are Clk, Reset, Select, Side, and Sensors. The outputs are Lasers and L_alarm. On the positive clock edge, the Laser FSM cycles through its states. It starts at IDLE, and if Select is low, then the FSM goes into CHECK. If the sensors are all receiving data, then the system moves back into IDLE. Now if Select is high, then the FSM transitions into SIDE. The system stays in this state until Select is low. While in SIDE, the user can change the three bit number corresponding to the side he chooses to turn off.

When designing this system, we had to figure out how to optimize the performance without sacrificing integrity. We were inspired for this part from several different movies, including Ocean's 12, Mission: Impossible, Entrapment, and The Saint. For this part of the security system, we decided that a stationary laser net would be most suitable. A moving laser net would be ideal, but more difficult to implement. It is best to design something simple, and then build up from there. We spent one afternoon in the MIT Hobby Shop constructing the box frame. Being Course 6 students, we were unfamiliar with the use of power tools, but we learned how to use them very quickly. Figuring out how to power the lasers was another problem, but it was solved by experimenting with lasers of varying power. We mounted the lasers on each face of the cube by using some 3M clips, zip ties, and Fun-Tak. This combination proved to be successful.

A problem we ran into was during debugging. Everything was working well during simulation, but the output was correct when programmed on the FPGA. After a day of debugging, we found that the power from the phototransistors wasn't high enough due the wiring on the system. After a little but of tuning, the power problem was

corrected and the system worked flawlessly. As for the digital design, it was fairly simple to implement. The code is very straightforward, and performs without any problems.

Pressure Switch

The pressure switch makes use of a modified digital scale. The data coming out from the scale is a periodic waveform with a period of 30ms. In this block, we take six samples, one at every five milliseconds. This data is then fed into the AD670, where it is converted into eight bits. These bits are summed for every cycle, and then the sum is checked. If the sum is not equal, then an alarm pulse is sent out.

This module (Figure 9) is comprised of several blocks: a Synchronizer, Divider, Adder, and FSM. The Synchronizer takes signals and outputs them on the rising edge of the clock. This is to make sure that all inputs to the system are synchronized. If an input were not synchronized to the clock, then data going into the Synchronizer could be sent out before the next block is ready, thus resulting in invalid data. The purpose of synchronizing is to reduce the possibility of metastable inputs and remove possible glitches.

The Divider takes in the clock signal and then outputs an Enable signal when it counts up to the right number. The number we set determines the frequency of the sampling. After a little bit of experimenting, we were able to determine an adequate sampling rate in accordance with the period of the digital scale data. The Adder is used to add up the digital data from the AD670. When the adder_enable is high, the adder adds the input to an internal register. The format of the data is two's complement, so

adding the data is simple arithmetic. When the adder_reset is high, then the internal register is reset to zero.

The FSM for this module (Figure 10) is made of many states. First, we start in the INITIAL state where all values are initialized. Then we move into IDLE and wait for the Enable. After this the AD conversion begins and then the data is stored in a register in the FSM. The FSM checks to see if Store is high. If so, then we move to COMPARE and the new sum is checked against the old sum. If Store is low, then we store the data from the Adder in a register and wait until the next cycle before we compare the values.

When dealing with this part, we had to figure out how to analyze the data coming out from the digital scale. After some hacking, we found an adequate waveform. As for reliability, we found that the output was not consistent. Sometimes we would get a correct response from the system, and sometimes we wouldn't. This part was not fully resolved. To make sure that the presentation went smoothly, we created a simpler algorithm that uses a simple FSM and some analog bump sensors. The code for this section is included in the appendix.

Alarm

The Alarm module (Figure 7) makes a noise every time the system detects an intruder. We used a buzzer from a telephone to create the sound. The sound is generated by sending it a square wave set at the right frequency. Before the alarm is sounded, data is processed by the Alarm Register. When the Alarm Register is pulsed, it transmits a signal to the Alarm block, allowing it to trigger the alarm. During regular operation, the

Alarm Reset is high, so no sound is produced. This part of the system was fairly simple to design.

Simulation and Testing

The design used several testing and simulation methods for its development. My partner and I used ModelSim and Max+plus II to simulate our modules before actually programming. For the primary labkit, ModelSim was the simulator of choice. ModelSim is a quick and easy way to program and simulate code. In particular, the environment was also programmed, meaning external modules we did not have to program were coded. The idea was to try to iron out as many possible bugs in the simulation environment. For instance, to prepare the keypad, we implemented a small keypad module that would output the signals we expected from the keypad. Not only is it easier to have a module to do the external signals for you, it makes one less likely to make errors.

On the primary labkit end, although there was extensive simulation testing, sometimes the code did not work as we intended it to. The problem is that the simulation environment is the perfect environment, but our coded modules are subjected to less-than-perfect conditions. For instance, the keypad FSM did not initially work, though it worked perfectly in simulation. The states would keep changing at inappropriate times; the reason was the keypad clock had glitches which caused erratic behavior. The simulations keypad clock did not generate those glitches, so there were bugs that were latent.

The solution to the unexpected behavior was to use the logic analyzer. The logic analyzer is very effective in debugging the system. With the logic analyzer, there is no doubt as to what the signal is. When debugging, we would see an expected change in the signal that we thought wasn't possible, but we knew that that was the signal the system was getting. This led us to look more carefully at our code, and helped us target the correct locations for editing code.

After we coded the laser and pressure blocks in Verilog, We used Max+plus II to simulate. This program was very reliable for analyzing the response of individual blocks. When dealing with the laser system, we had to make sure we simulated with the correct data being fed into the Sensors. Max+plus takes in a hexadecimal number when you try to write in a number in the simulator. We used the Calculator program to convert the expected digital binary signal (11111) into hex (1F). When dealing with the scale, we tried to simulate every possible state, and made sure that the code was behaving properly. In order to simulate faster, we decreased the enable signal count, thus increasing the frequency. Also, we set my FSM to check the sum after every sample instead of six. By doing this, we were able to simulate everything properly.

Another important tool used for debugging was the oscilloscope. We used the scope to determine the aforementioned power problem when dealing with the lasers and sensors. We also used this to check signals consistently throughout our FPGA testing process. One part where the scope was very useful was when we were interfacing the two labkits. When we first interfaced the two kits, we used long pieces of wire for the laser and pressure alarm signals as well as the ground. Everything was working fine. Then we decided to move the labkits closer together in order to mount the camera on the

frame of the box. When we did this, the alarm kept signaling. When we inspected the signals between the kits using the scope, we noticed that we were getting some interference. We believe this was due to the wires being excessively long, as well as having many wires close together. After shortening the wires and optimizing the wiring of the secondary labkit, the signal was clean and everything worked smoothly.

Conclusion

Overall, the system was a success. All the components worked as planned, with some minor modifications. We realized that some of our initial designs were not complicated enough, so we made them more complicated. Planning our time was useful as well. We tried to keep a strict schedule so we would be able to finish on time. The project was finished two hours before the deadline, and we feel that it was a success. All of our proposed modules interfaced properly, and no one was able to successfully penetrate our system. Through the process of developing this project, we have seen that defense in depth is the answer to problems with security. In a dangerous world, your possessions are at risk. However, with the Hypersonic Security System, there will be safety. In conclusion, the Hypersonic Security System leaves thieves hanging!

Acknowledgements

We would like to acknowledge Nathan for his code, Keith for his help with video and analog circuitry, Charlie for his quick lesson on operation amplifiers, Yun for her advice on the sound processing, and most importantly, Jenny for all of her guidance and support. We <3 you Jenny.