# Final Project

*Due Midnight at the end of Wednesday, March 13, 2002*

## The Big Picture

Your last laboratory assignment of the quarter will be to design, document, and demonstrate a digital design project that exercises the skills you have been developing this quarter in EE121.

This handout describes four possible projects: the BreakOut video game, the Worm video game, a MIDI sound player, and a RC4 encryption cipher cracking program. You may choose a project other than the ones listed here, but you do so at your own risk. The TAs will be better able to help you if you choose one of the listed designs, and they will probably not be able to devote the same amount of effort in supporting individually designed projects. The projects are not necessarily of equal difficulty. All of the project descriptions include some basic functionality as well as many optional features that you can add as time permits. The difficulty will vary with the amount of features you choose to implement. We will keep this in mind when assigning grades. You should choose a project that you will enjoy designing and that you feel confident you will finish.

You will achieve success through three easy steps.

☺ Before you design anything in Xilinx Foundation, think carefully about your design, what modules it will have, and how these modules will interact with one another. Drawing block diagrams, state diagrams, and transition/output tables will be especially useful here. Feel free to describe your plans to the TAs and ask for feedback.
☺ Before you come to the lab to test your hardware, simulate your design thoroughly. Debugging is much easier with a simulator.
☺ Break the design process up into steps, and test each step in hardware before you proceed to the next step. Do ***not*** even think of adding advanced features before all your basic features work.

A simple but deadly mistake is to design everything in Foundation, then come in to the lab near the end of the quarter and expect your project to work right away. It won't! We will be using several external interfaces in this project (VGA monitor, SDRAM, audio codec, game controller, etc.) and you do not know whether your design works with these interfaces until you've seen it work with the real hardware. Debugging and working with real hardware are important components of digital design, and a design that works "in theory" but not in practice is simply a design that does not work. A simple design without advanced features that works well is much better than a fancy design that does not work at all. If you don't think you'll be able to finish your project by the deadline, you should talk to the TAs in advance. They can usually suggest ways to reduce the scope of the design so you'll have something to show.

The final project is due Midnight at the end of Wednesday, March 13. You will demo the project the following morning to your classmates and the teaching staff. We will not accept any late projects; you must demonstrate whatever you have working by Midnight the end of March 13th. Think of the project a "proof-of-concept" demonstration to some venture capitalists you want to fund your company, or some wealthy donors you want to fund your research group. You don't have infinite time or resources, so your visitors (the teaching staff) do not expect elaborate features or a flawless, commercial product. They simply want tangible proof that you understand the design problem and can build a system to solve the problem. Functionality, a good demonstration, and clear explanations are important.

## Bureaucratic Details

### *Legal Issues*

You should work in groups of two on this project. You may also work on the project individually, but you may not work in groups of more than two students (unless this is pre-authorized by the instructor). You should submit only one copy of everything *except* the documentation. Each team member must submit his/her own project documentation.

### *Open Lab Hours*

You should come to the lab during your regularly scheduled lab sessions to test the designs you create and simulate in Foundation. You may also work on your project during the other group's lab session, but students in that session will have priority in the use of equipment. The TAs will also open the lab at other times (to be posted on the project web page) for project work. Anyone can come in during any of these open lab hours. However, since we have many more groups than lab stations we might start a reservation closer to the due date. These open lab hours replace all previous TA office hours.

### *Web Page*

Your faithful companion in this project will be http://www.stanford.edu/class/ee121/project.html. (A link to this page is on the class web site.) We'll post helpful documentation on the interfaces, answers to frequently asked questions (FAQs), software tools, and useful links as the project progresses, so be sure to visit this page regularly.

### *Demos*

Unless specially arranged, all the demos will be all conducted on the same set of hardware in lab. All the boards are the same in the lab and there will be 10 identical gamepads for you to prototype with during the final project development phase. The instructor will download the *.bit file from your submit directory and any necessary SDRAM files (for the MIDI and RC4 projects). If you need to do anything special to setup your demo, please talk to the instructor as soon as possible.

*Documentation*

The project documentation requirements are similar to those given in the final assignment of E102E. You may use the same report you write in E102E for your EE121 documentation. Your documentation should have two distinct parts (taken from the E102E assignment):

(1) A circuit description and "theory of operation." This part should include a description of all of the inputs, outputs, functional blocks, and algorithms in your particular design, written for experts, engineers who will want to understand what's special about what you've designed.
(2) A "user manual," which enables non-specialist users to play the game. Together, both parts of your document should total about 2-3 pages, typed, single-spaced, exclusive of diagrams.

You should write your report clearly and concisely. In addition to the hardcopy you can turn in to the computer cluster in Packard 128, place a PDF copy in your submit directory or emailed in the usual submission procedure.

*Deadlines*

We will hold mid-project and final demonstrations in Packard 127. You may submit all supporting files electronically (via Z:\submit or e-mail) or in the drop box in Packard 128. No late demonstrations or submissions are accepted for any of these deadlines; you should show us what you have by the due date. The purpose of the two "milestone" deadlines is to ensure that you are on schedule to finish your project on time – and if, you are not, to provide a time for you to discuss your problems with the TAs.

*Friday, February 22nd at 6pm:* By this time, you should have chosen a project and e-mailed your choice to your regular lab section TA. Be sure to tell us whom you are working with on the project. If you wish to do your own project, you must make an appointment with the Instructor to gain approval of your project and establish appropriate milestones. After this date, you may not change your project choice.

*Thursday, February 28th, 2002, 22:00:00:* Milestone # 1, described separately for each project.

*Thursday, March 7th,, 2002, 22:00:00:* Milestone # 2, described separately for each project.

*Wednesday, end of day March 13th, 2002, midnight:* Final project submission. Archive and submit your final project.

*Thursday, March 14th, starting at 9am*: Final Project Demonstrations. Each team will have approximately 7 minutes to demonstrate their project to the class and teaching staff.

*Tuesday, end of the day March 19th, 2002, midnight:* Documentation due, both electronically (in PDF form) via e-mail or Z:\submit, and in hardcopy in the drop box in Packard 128.

**Breakout!**

http://www.stanford.edu/class/ee121/breakout.html

*Project Description*

In this project, you will design a device that allows a user to play the video game Breakout using the XSA-100 board, Xstend board, gamepad and VGA monitor. Breakout involves a paddle on the bottom of the screen, which can scroll back and forth to hit a ball toward a block of bricks. As the ball strikes a brick, the brick will disappear. When the player has eliminated all of the bricks, the game is over. The player will control the motion of the paddle using the gamepad. The video game's array of grid pieces will be displayed on the VGA monitor. By using characters stored in memory for each array element on the screen, we can dramatically simplify control of the game output on the monitor. For an example of Breakout, go to the link http://www.geocities.com/SiliconValley/Bay/6879/Breakout.html.

**Specifications**

*Required:*

(1) Set the Breakout game array to grid pieces that are 16 bits by 16 bits in size. The top fraction of the array will be bricks. The rest will be blank, and the bottom row of the array should be reserved for the movement of the paddle.
(2) Initialize the game with the paddle in the center of the bottom row. The ball can start out in the same position each time.
(3) There should be a RESET signal that can stop any currently running game, and reset the screen to the initialization state.
(4) If a player misses the ball, it should disappear and the game should be reset.
(5) The paddle should be 3 grid pieces wide. When the ball strikes the center piece in the paddle, it reflects straight back (90 degrees). If the ball is to hit either grid piece to the right or left of the center piece, then it the ball must reflect at 45 degrees in the direction opposite from where it came.
(6) The sidewalls of the game area must reflect the ball toward the middle at a 45 degree angle (this is the simplest boundary condition we can implement). Similarly, as the ball strikes a brick, the ball must be reflected at the opposite angle from its approach. If a ball goes vertically, it should reflect vertically.
(7) The paddle movement can be at a resolution of "paddle" sized blocks. This means that if the paddle is three blocks wide, it only has to move in three block increments.

*Recommended:*

(1) Implement a score in the top row of the game array. Assign some number of points for each brick that is hit, and keep a running total.
(2) Use multiple colors for the blocks. It could look nice to make each row of bricks a different color.
(3) Increase the resolution of the paddle movement to allow finer adjustment of the paddle position (i.e. have the paddle move one game array piece at a time).
(4) Allow the player to have multiple balls per game play. So the state of the block of bricks (which ones have been eliminated and which are still present) must stay the same until all of the balls have been used. It would be a good idea to display the number of balls left on the top row of the screen (you can even use the ball character that you create).

*Optional:*
The following is a partial list of extensions that would make your design more impressive. Feel free to implement any other ideas that you come up with. *These are optional, and we do not expect you to implement all (or even most) of them.* You should finish the required functionality before attempting to add extra features.

(1) Set up different levels of the video game so that after you "beat round one" (knock out all the bricks) you go on to the next level which is not identical to the first. The next round also can have a different configuration of bricks and possibly an increase in ball speed.
(2) Implement a more complex algorithm for the reflection of the ball. Add different angles (other than 45, 90) at which the ball can be reflected. These angles could be a function of current angle of the ball and the location of the piece in the paddle that the ball comes in contact with. For example, if the ball is currently at 45 degrees and it hits the outer edge of the paddle, it could be reflected now at 30 degrees.
(3) Randomize the position of the ball upon initialization and reset of the game.
(4) Add sound effects. There could be a sound for the reflection of the ball as it hits the paddle or the sides of the screen. When the ball comes in contact with a brick, there could be a different sound. If a player misses the ball, there could also be a sound effect.

**External Interfaces**

Your design will interact with the VGA monitor and the gamepad. You can find datasheets for both these devices on the project web page.

**Milestones**

*Milestone # 1:* You should submit a block diagram that includes all the major components of your design and a fully labeled state diagram or transition table. The diagram must specify precisely the outputs and next-state behavior for every state and input combination. In the lab, you should demonstrate the ability to initialize the VGA monitor and display the game board.

*Milestone # 2:* You should be able to present the video game's grid with blocks and paddle. You do not have to show the ball being reflected back and forth across the screen yet, but you should at least get the paddle to move. It would be good to have the ball move vertically between the paddle and bricks.

## Worm (against computer)

http://www.stanford.edu/class/ee121/worm.html

*Project Description*

In this project, you will design a device that allows a user to play the video game Worm using the XSA-100 board, Xstend board, gamepad and VGA monitor. Worm involves two players (worms) that move in all four directions in a grid. The players will control the direction of the worms' heads and their bodies will follow. The two worms get longer and longer as the game progresses. The object of the game is to outlast the other worm. The game ends when one of the worms "crashes" into one of the four sides, into the other worm, or into it's own tail. The two players will control the motion of their respective worms each using a gamepad. The video game's array of grid pieces will be displayed on the VGA monitor. By using characters stored in memory for each array element on the screen, we can dramatically simplify control of the game output on the monitor.

### Specifications

*Required:*

(1) Set the Worm game array to grid pieces that are 16 bits by 16 bits in size. The array will have a boarder, but the rest will be blank except for the two worms. They can start out in the same position each time.
(2) There should be a RESET signal that can stop any currently running game, and reset the screen to the initialization state.
(3) If a player "crashes", that worm should disappear and the winning worm should flash.
(4) If both players "crash" simultaneously, it is a draw and neither player wins. Both worms should disappear at this time.
(5) Initialize the worms to be 3 grid pieces in length. Once the game begins, the worms should gradually get longer until the game ends.
(6) Once again, a "crash" occurs when one of the worms' heads attempts to occupy the same bit as a wall, a part of the other worm, or it's own tail.
(7) Use different colors for the two worms.

*Recommended:*

(1) Implement a counter in the top row of the game array. The counter will simply be the grid length of the worms. Initialize this counter to 3 because the worm is initialized to three bits in length.

(2) Initialize the screen with random blocks where the worms cannot go. If a worm crashes into one of these blocks, that worm looses and the other worm flashes.
(3) Allow the game to be played against the computer. One player will control one worm, while the other worm moves randomly throughout the grid.

*Optional:*
The following is a partial list of extensions that would make your design more impressive. Feel free to implement any other ideas that you come up with. *These are optional, and we do not expect you to implement all (or even most) of them.* You should finish the required functionality before attempting to add extra features.

(1) Set up the game so you play best two out of three and keep score of how many games each player has won.
(2) If you decide to implement option #1, try making games 2 and 3 faster. Not only do the worms move faster, but their size increases faster as well.
(3) When you play one player, instead of having the computer's worm totally random, have some sort of algorithm where it tries to trap the head of the other worm.
(4) Allow the game to be played by three players using three controllers.
(5) Add sound effects. There could be a sound for each time the worms grow and a sound when one worm wins.

**External Interfaces**

Your design will interact with the VGA monitor and the gamepad. You can find datasheets for both these devices on the project web page.

**Milestones**

*Milestone # 1:* You should submit a block diagram that includes all the major components of your design and a fully labeled state diagram or transition table. The diagram must specify precisely the outputs and next-state behavior for every state and input combination. In the lab, you should demonstrate the ability to initialize the VGA monitor and display the game board.

*Milestone # 2:* You should be able to present the video game's grid with the worms initialized to their correct lengths and different colors. You do not have to show the worms growing in size yet or even what happens when they crash, but you should at least get the worms to move.

**Worm (one player)**

http://www.stanford.edu/class/ee121/worm1.html

*Project Description*

In this project, you will design a device that allows a user to play the video game Worm using the XSA-100 board, Xstend board, gamepad and VGA monitor. Worm involves one player (worm) that moves in all four directions in a grid. The players will control the direction of the worms' heads and their bodies will follow. The worm will get longer each time the worm eats a piece of "food." Their will always be one and only one piece of food that the worm is trying to acquire. Once the worm eats that piece of food and new piece will randomly appear somewhere else on the game board. The object of the game is to eat as many pieces of food as possible. The game ends when the worm "crashes" into one of the four sides or into it's own tail. The player will control the motion of the worm using a gamepad. The video game's array of grid pieces will be displayed on the VGA monitor. By using characters stored in memory for each array element on the screen, we can dramatically simplify control of the game output on the monitor.

**Specifications**

*Required:*

(1) Set the Worm game array to grid pieces that are 16 bits by 16 bits in size. The array will have a boarder, but the rest will be blank except for the worm and a piece of food. The worm can start out in the same position each time.
(2) There should be a RESET signal that can stop any currently running game, and reset the screen to the initialization state.
(3) If the player "crashes", that worm should disappear and the score should flash.
(4) Each time the worm eats a piece of food, another piece should randomly appear somewhere else in the game array.
(5) Initialize the worms to be 3 grid pieces in length. Each time the worm eats a piece of food it should get longer.
(6) Once again, a "crash" occurs when one of the worms' heads attempts to occupy the same bit as a wall or it's own tail.
(7) Use different colors for the worm and the food.

*Recommended:*

(1) Implement a score in the top row of the game array. The score will simply be a multiple of how many pieces of food the worm has eaten.
(2) Initialize the screen with random blocks where the worms cannot go. If a worm crashes into one of these blocks, the game is over.
(3) You could keep track of the high score.

*Optional:*
The following is a partial list of extensions that would make your design more impressive. Feel free to implement any other ideas that you come up with. *These are optional, and we do not expect you to implement all (or even most) of them.* You should finish the required functionality before attempting to add extra features.

(1) When the worm eats a piece of food have the worm speed up as well as get longer.
(2) Have the worm not only move faster, but it's size increase faster as well.
(3) Add sound effects. There could be a sound for each time the worms grow and a sound when the game is over.

**External Interfaces**

Your design will interact with the VGA monitor and the gamepad. You can find datasheets for both these devices on the project web page.

**Milestones**

*Milestone # 1:* You should submit a block diagram that includes all the major components of your design and a fully labeled state diagram or transition table. The diagram must specify precisely the outputs and next-state behavior for every state and input combination. In the lab, you should demonstrate the ability to initialize the VGA monitor and display the game board.

*Milestone # 2:* You should be able to present the video game's grid with the worm initialized to it's correct length and it should be a different color than the pebble of food. You do not have to show the worm growing in size yet or even what happens when it crashes, but you should at least get the worm to move.

## MIDI Player

http://www.stanford.edu/class/ee121/labs/project/midi/index.html

*Project Description*

In this project, you will design a device that plays MIDI files using the XSA-100 board, XStend board, and an auxiliary pair of speakers. After you download one or more MIDI files to the SDRAM on the XSA-100 board, your player will read the file sequentially from memory and play back the music according to the file's specifications. You will generate notes by storing sinusoid samples in the FPGA's BlockRAM, sampling them at a rate proportional to the pitch of each note, and sending them to the XStend board's stereo audio codec.

*Background Information*

The *Musical Instrument Digital Interface (MIDI)* is a protocol for connecting electronic musical instruments. It is both an asynchronous serial communication protocol and a data format. Like other serial communication protocols, MIDI describes how bytes of data should be assembled together and transmitted over a cable between two devices. But whereas the bytes traveling over most serial communication protocols might mean anything, bytes that travel over MIDI channels encode a description of a musical performance.

An auxiliary part of the MIDI specification is a file format called the Standard MIDI format (SMF) that describes a musical performance using a format that is very similar, and in many cases identical, to the format of the data that passes over MIDI communication channels. You will use this file format in the project.

MIDI is *not* an audio format like WAV, MP3, or the raw audio samples you used in Lab 6; it describes how music is to be played, not what the music sounds like. It contains information like what notes to play, the time at which the notes occur, the pressure with which the notes are struck, the tempo of the musical score, and the type of instrument to use. Many implementation details are left to the playback device, and the same MIDI data might sound differently on different devices.

The MIDI file describes much of the same information as a musical score. It is not identical to a score, but the analogy will suffice for this project. Underlying the MIDI file's description of a musical score are two fundamental time periods, a *division* period and a *quarter note* period. The MIDI file sets the tempo by specifying the number of microseconds per quarter note period. It then defines a division period as a fraction of a quarter note period.

The body of the MIDI file is a series of *event* descriptions, each of which is associated with a *timestamp*. The timestamp specifies how many division periods should elapse before the event occurs. The two most common events are *Note On* and *Note Off*, which behave exactly the way you would imagine. The Note On event turns on one of 128 specific notes, and the Note Off turns a note off. Of course, more than one note might be on at the same time.

The MIDI file also might specify many other properties of the score, like what instrument should be playing and how fast the notes should be pressed. It might include multiple voices. We will either ignore these features or use MIDI files that do not include them. The project web page will contain more detailed information on the MIDI file format.

*Rumors*

The MIDI project is challenging. Some of you might have heard troubling rumors from last quarter's students about the difficulty of creating a working MIDI player. Rest assured that we have significantly simplified your task this quarter by providing you with

a codec interface, giving you experience with reading files from memory in Lab 6, and switching to a chip that has block RAM components. But building a MIDI player is not easy, and you should make sure that you construct the basic functionality before attempting to add advanced features.

**Specifications**

*Required:*

1. Download and play single-track, single-program (i.e., single instrument), chord-less MIDI files of at least five minutes in length. We will provide sample MIDI files of varying degrees of complexity that you should be able to play.
2. The MIDI player should have a basic user interface. You should have at least a play and a stop button, either from the on-board pushbuttons we have used in the labs or from a joypad. The MIDI player should play in response to one pushbutton, and reset in response to the other. It should display a "P" on the XSA-100 board seven-segment display when playing, an "S" when stopped, and an "E" if it encounters an unsupported feature in a MIDI file. Feel free to develop an alternative user interface if you wish.
3. Play sinusoidal notes with a range of at least two octaves. You may choose the range, but one that includes middle C would make sense.

*Recommended:*

Your design would be much more impressive with these features and you should try to include them. However, they are not easy to implement and you should be sure to complete the required specifications first.

1. Set the tempo as directed by the MIDI file before starting to play. Set the tempo to a constant value while you work on the required features, then make the tempo programmable after you make the other components work.
2. Play chords of at least three notes. You should make your design work with single notes before you try to implement chords.

*Optional*

The following is a partial list of extensions that would make your design more impressive. Feel free to implement any other ideas that you come up with. *These are optional, and we do not expect you to implement all (or even most) of them.* You should finish the required and recommended functionality before attempting to add extra features.

1. Set the tempo several times while playing a score, as directed by the MIDI file.
2. The Note On and Note Off events include information on the pressure with which the note is hit and the speed with which it is released. Incorporate these into your design by adjusting the note's volume as a function of its pressure.
3. Real instruments do not generate purely sinusoidal notes. Make the note sound better by selectively adding harmonics or an attack and decay.
4. Add support for two voices. An extra voice could be something as plain as a square or sawtooth wave vs. a sine wave, or you might use some better waveforms from extension 3. Play one voice on the left stereo channel and the other voice on the right stereo channel.

5. Provide helpful information on a monitor with the VGA interface. You might display the status information ("P," "S," or "E") or you might display more complicated information like which notes are being played.
6. Allow the user to pause, advance, or rewind play.
7. Add support for multiple scores, and allow the user to select which one s/he wants to play.
8. Gracefully ignore standard MIDI features that your design does not support. This will allow you to download a wider range of MIDI files.

**External Interfaces**

To build the MIDI player, you will need to use some external interfaces.

1. The SDRAM, using the same interface macro you used in Lab 6.
2. The stereo codec on the XStend board, using a slightly modified interface macro that we will provide for you. It will behave in exactly the same way as the one in Lab 6, but its sample rate will be 97656.25 Hz rather than 8138 Hz. If you wish to change the sample rate to some other value, we can show you how to do this.
3. *(optional)* The joypad
4. *(optional)* The VGA interface macro that you used in Lab 5 to display user-friendly information to a monitor.

**Milestones**

*Milestone # 1:* You should submit a block diagram that includes all the major components of your design and fully labeled state diagrams or transition tables. The diagrams should specify precisely the outputs and next-state behavior for every state and input combination. Be sure to do this carefully, since a little planning can save you hours of debugging time. In the lab, you should demonstrate the ability to play notes independently of the contents of the MIDI file.

*Milestone # 2:* You should be able to download and play CMajor.mid (a C Major scale, not surprisingly) from the project web page. If you're a melancholy person, you may demonstrate CSharpMinor.mid instead. You do not need to demonstrate chords, the ability to set the tempo, or the ability to play more complicated files.

## RC4 Encryption Cracker

http://www.stanford.edu/class/ee121/rc4.html

*Project Description*

In cryptographic systems, a key (K) is used by an encryption algorithm (in this case, RC4) to encrypt a plaintext message (PT) thereby creating a ciphertext (CT). Algebraically this is expressed as CT = RC4(K, PT). The aim of this project is to determine the inverse. Namely, given a ciphertext (CT) and an encryption algorithm (RC4), determine the Key (K) and the plaintext (PT). This is in effect "cracking" the

encryption algorithm. This project is modeled after the Electronic Frontier Foundation's DES Cracker project (http://www.eff.org/descracker/), but using RC4 instead of DES.

*Background Information*

RC4 is an important algorithm underlying the SSL protocol that protects information transmission on the Internet. Using Internet Explorer, go to https://banking.wellsfargo.com/. Click on File→Properties and you will see that the connection is protected with 128bit RC4 encryption.

RC4 is an algorithm that is specified in many places. A very good one is at http://www.cs.berkeley.edu/~iang/isaac/hardware/main.html. In fact, the authors of this paper created a RC4 cracker very similar to this project.

The basic outline of the project is the following. A cipher text will be given to you in the SDRAM. You will load the CT into the FPGA and then successively try keys until you get a plaintext that "looks" like a message. That is the message only has ASCII characters in it—the letters, numbers, and spaces. "How to Recognize Plaintext" http://www.counterpane.com/crypto-gram-9812.html#plaintext has more info but this is the basic concept.

In order to demo the project, you must successfully decrypt a CT provided and display the resulting PT and K on the VGA display. The K used will only be 16 bits in key material (2 ASCII characters) but will expand internally to the full 256 bytes for the algorithm. The CT will be generated by either of the following equivalent means ("12" is the 16 bit key for this example):
"cat rc4.in | ./rc4.pl 12 > ! rc4_perl.txt"
"openssl rc4 -K 12121212121212121212121212121212 -iv 0 -out rc4.txt -nosalt -p -in rc4.in"

The rc4.pl script can be found on many webpages and is pretty cool ☺:
```
#!/usr/local/bin/perl -- -export-a-crypto-system-sig -RC4-in-3-lines-of-PERL
@k=unpack('C*',pack('H*',shift));sub S{@s[$x,$y]=@s[$y,$x];}for(@t=@s=0..255)
{$y=($k[$i++]+$s[$_]+$y)%256;$x=$_;&S;$i%=@k;}$/=$x=$y=0;for(unpack('C*',<>))
{$x++;$y=($s[$x%=256]+$y)%256;&S;print pack(C,$_^=$s[($s[$x]+$s[$y])%256]);}
```

**Specifications**

*Required:*

1. Read a 128 Byte Cipher Text message from the SDRAM that was encrypted with a 16 bit RC4 key and display the Plain
2. Display the final results when they are available on the VGA monitor.
3. Play a sound via the codec when the cipher text is cracked.

*Recommended:*

Your design would be much more impressive with these features and you should try to include them. However, they are not easy to implement and you should be sure to complete the required specifications first.

1. Create a faster RC4 implementation to speedup the cracking (Consider using a dual-port memory).
2. Indicate progress (number of keys searched and wall clock time) on the VGA monitor as the algorithm proceeds.
3. Play audio cues that indicate the progress of the algorithm.
4. Indicate on the VGA monitor the number of keys tried and the time to complete the search.

*Optional*
The following is a partial list of extensions that would make your design more impressive. Feel free to implement any other ideas that you come up with. *These are optional, and we do not expect you to implement all (or even most) of them.* You should finish the required and recommended functionality before attempting to add extra features.

1. Implement multiple RC4 crackers in the FPGA and coordinate their operation so as to crack the CT faster. The FPGA is much, much bigger than a single RC4 module.
2. Implement a user interface with the gamepad.

**External Interfaces**

To build the MIDI player, you will need to use some external interfaces.

1. The SDRAM, using the same interface macro you used in Lab 6.
2. The stereo codec on the XStend board
3. The VGA interface macro that you used in Lab 5.
4. (optional) the gamepad.

**Milestones**

*Milestone # 1:* You should submit a block diagram that includes all the major components of your design and fully labeled state diagrams or transition tables. The diagrams should specify precisely the outputs and next-state behavior for every state and input combination. Be sure to do this carefully, since a little planning can save you hours of debugging time. In the lab, you should demonstrate the ability of your core RC4 algorithm with a known PT, CT, and K.

*Milestone # 2:* You should be able to encrypt a hard-coded (ie, not read from the SDRAM) CT and display it to the VGA monitor.