Walkie Talkie FPGA

Keawe Mann
Department of EECS
Massachusetts Institute of Technology
Cambridge, MA, U.S.
keawem@mit.edu

Khang Le
Department of EECS
Massachusetts Institute of Technology
Cambridge, MA, U.S.
khangle@mit.edu

Khoa Huynh
Department of EECS
Massachusetts Institute of Technology
Cambridge, MA, U.S.
khoah_24@mit.edu

***Abstract*—We present a design for our Walkie-Talkie on FPGA (WTF). A WTF is implemented using the following key components: an FPGA, microphone, audio output device (i.e., headphones, earbuds, or a speaker), laser (or wire), and phototransistor, among others. Our device encrypts all transmissions using AES, the industry standard encryption method for encrypted radios, including those used by the U.S. government.**

I. System Overview

A WTF's laser is used to transmit audio, while the phototransistor is used to receive audio. A WTF is capable of both sending and receiving audio, given the necessary hardware. No additional hardware is necessary for audio input, as each FPGA comes with a microphone of sufficient quality. Like audio input, audio output is quite simple–each FPGA has a headphone jack. A speaker is hooked up to this headphone jack.

The laser and phototransistor are on an external circuit connected to the FPGA. This circuit uses comparators as necessary to ensure a clean digital signal. This laser and phototransistor circuit can also be replaced by a simple wire connecting the two FPGAs.

FIR and decimation modules are used to perform low pass filtering on the microphone audio. The FIR is a 30-tap filter, and the decimation module decimates the audio samples by a factor of 4. Having four stages of FIR and decimation allows us to achieve a clean audio signal devoid of high frequency content.

The whole system connects as follows: audio is received through the microphone, processed, filtered, and encrypted by the FPGA, which then transmits the data through the laser. The receiving FPGA's phototransistor receives the data, which is processed, decrypted, then sent to the audio output port.

II. Signal Processing

A. Low-Pass Filter and Decimation

The incoming audio information is a 3 Msps stream at one bit depth. Since human speech has frequency content ranging up to approximately 6000 Hz, we process information up to that frequency. The main motivation for this is to reduce resource usage, cost of computation, and aliasing. To do this, we need to put the audio data through a low pass filter and down sample. There are various approaches to achieve this—as a starting point, we decided to put this audio information through a multistage FIR low pass filter and decimation. For reliable low pass filtering and down sampling, we perform four rounds of FIR filtering and decimation. The FIR module (fir.sv) processes a history of a weighted sum of the 30 most recent data points and carefully picked coefficients, which can be tuned for higher filtering fidelity. The output is then passed into decimate.sv, where it will be down sampled by a factor of 4. The resulting signal is a data stream of 12k samples per second at 8-bit depth, which will be used in the encryption module. The FIR filter is implemented through a finite state machine, consisting of 3 states (IDLE, SUM_ADD, and DONE). The module begins in the IDLE state. It waits for a single cycle indicator that a sample of valid data is ready for processing. Upon receiving this signal, the module proceeds to the SUM_ADD stage. In this stage, a series multiply and addition is performed on the signal with its corresponding coefficients over 30 clock cycles, a trivial number of cycles compared to the number of clock cycles in 12 KHz (~8000 cycles). Once the calculation is finished, the module proceeds to the DONE state. In this stage, the module produces a single cycle valid out signal and outputs the valid data. The module then transitions back to the IDLE state and repeats. The FIR outputs will later be used by the decimation module for down sampling.

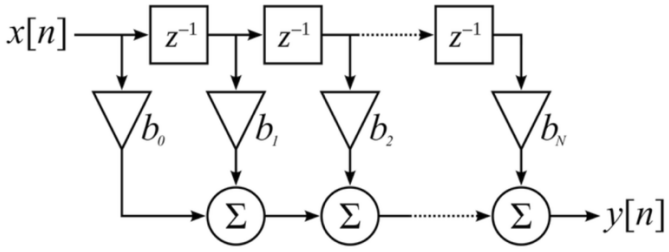

Figure 1: Basic structure of FIR filter [1]

We use the test bench (fir_tb.sv) for testing/evaluating the efficacy of the low pass filter and decimation. The test bench generates sinusoidal signal(s) and feeds it through the low pass filter and decimator. We developed a system for generating a reliable sine wave `slow_sine`. The generated sine wave is then converted to a PDM signal, which will be used as the input to the four stages of low pass filtering and decimation. An example of the simulation is shown in figure 3 below.

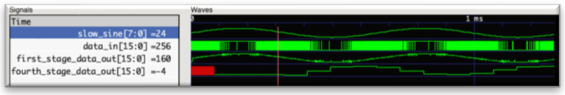

Figure 2: New test bench result for lowpass filtering.

Upon observing the test bench after trying different types of inputs, we can visually confirm that the LPF is working as expected. We further verify this fact by running this system on the FPGA. We were able to hear decent audio quality from the output of our multistage filters.

Some common issues that occurred while implementing the filtering and decimation module are bit growth and the high level of amplification to the output. The FIR module involves many rounds multiplication and addition. Because of this, the intermediate calculations may be subject to bit growth. We addressed this issue by using a larger buffer to store the intermediate result. In addition, we noticed that the output of each FIR and decimation adds additional gain to the signal. We addressed this issue by scaling down the output signal before feeding it into the next stage.

B. *Delta-Sigma Modulator*

At this stage of the pipeline, high-frequency noise is removed from the signal through the low-pass filter. However, there is still some residual quantization noise resulting from the digitization process upstream. The WTF employs a first-order delta-sigma modulator to reduce the effects of such noise.

The delta-sigma modulator samples the signal at a high sampling rate, much higher than the 12,000 Hz dictated by the Nyquist-Shannon sampling theorem, which ensures a more accurate signal. In a traditional delta-sigma modulator, the signal goes through a process called noise shaping, which corrects any quantization errors from the signal and shifts quantization noise to a higher frequency range outside of the targeted range. The quantization noise, in practice, will be removed downstream during the demodulation process where the digital signal is converted back to analog.

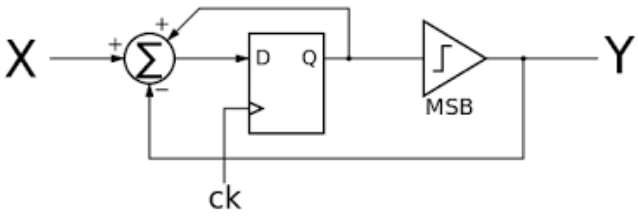

Figure 3: First-order delta-sigma modulator diagram

III. Encryption

The WTFs use the Advanced Encryption Standard, or AES, to encrypt communications. AES is widely used. iMessage uses it to encrypt text messages, and the military uses it in encrypted radios.

A. *Advanced Encryption Standard (AES)*

There are multiple versions of AES, which differ based upon the size of their input keys. We chose AES-128, where the key is 128 bits.

To encrypt, a series of transformations are applied to the provided block. There are four main transformations: substitute bytes, mix rows, mix columns, and add round key.

Byte substitution, much like the name implies, replaces each byte in the block with another. The replacement block is determined by indexing into a two-dimensional 16x16 table with the replacement blocks. The most significant nibble of the byte being replaced determines the row, while the least significant nibble determines the column. For example, a byte of value 0x6F would be replaced by the byte stored in the table at row 6, column F. The total table size is 2048 bits (256 entries of 8 bits each). Given the size of the table and to allow for future flexibility, it is stored as a read-only table in BRAM (effectively BROM). Accessing BRAM introduces an additional two clock cycle delay.

The second transformation—mix rows—shifts each row leftwards by its row number. A row is a collection of four bytes. The topmost row (row 0) is shifted left by 0 bytes, the next row (row 1) is shifted by 1 byte, row 2 is shifted by 2 bytes, and so on.

The mix columns transformation is effectively a matrix multiplication, where the block is multiplied by a given matrix. However, addition and multiplication are carried out according to the rules of finite field (Galois Field) arithmetic. This is accomplished on the FPGA using various bit operations.

The final transformation, add round key, puts the provided key to use. Ten additional keys are generated from the initial 128-bit key, resulting in a total of 11 keys. Each add round key transformation uses one of these keys, which are generated according to the key expansion protocol. The add round key transformation itself is just a bitwise XOR between the block and key.

Decryption is extremely similar to encryption, but the order of the transformations is changed. Additionally, the substitute bytes, mix rows, and mix columns transformations are the inverses of the encryption transformations. For example, while mix rows (used in encryption) shifts the rows to the left, the inverse mix rows operation (used in decryption) shifts the rows to the right.

AES protocol specifies that ten rounds of transformations be applied for both encryption and decryption, with each round using some combination of substitute bytes, mix rows, mix columns, and add round key (or the corresponding inverse transformation, in the case of decryption). Carrying out all of these transformations requires a fair amount of clock cycles, but this is no cause for concern.

Each transformation is implemented in its own module. These transformation modules are connected together in a module that carries out a round of several transformations—we call this the round module. Finally, the highest-level modules (one for encryption and one for decryption) carry out multiple

rounds using the relevant round module, plus some additional Verilog implementing the rest of the AES protocol. Ultimately, this allows us to encrypt a block with a single call to the encryption module and to decrypt a block with a single call to the decryption module. Both the encryption and decryption modules take in a block and a key. In order for decryption to reverse the results of encryption, both the encryption and decryption modules must be passed the same key.

The figure below is a simplified representation of how the modules fit together. At the bottom, we have the basic transformations. In the middle, the encryption round module combines these basic transformations together to form an AES encryption round. At the top, the encryption module uses the encryption round module, plus some additional logic to carry out the entire encryption. The decryption hierarchy is similar.

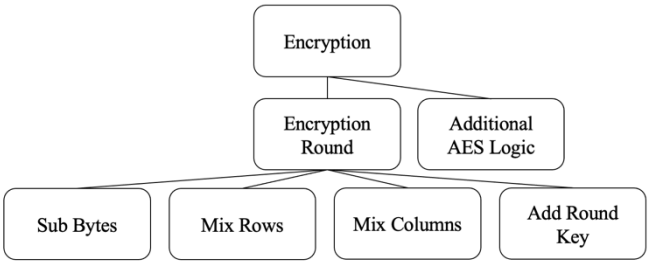

Figure 4: A simplified representation of how the various modules fit together to form the encryption module.

The output of the signal processing module/low pass filter is 8 bits at 12 kHz (size conversion between modules is discussed in the next section). The blocking module input is, correspondingly, of width 8 with an output of 128 bits. This implies that the output frequency of the blocking module is approximately 16 times (128 / 8) lower than the throughput of the filter. This means the blocking module output frequency is 0.75kHz (12 kHz / 16). The FPGA's clock runs at approximately 98 MHz. With this in mind, the encryption and decryption module scan take up to ~130,000 clock cycles (98 MHz / 0.75 kHz) without creating a bottleneck. The encryption and decryption modules are far faster than this. Each module finishes in under 8000 nanoseconds, or approximately 800 clock cycles. It's worth mentioning that these operations could be sped up, but this was not necessary for our project.

B. *Blocking*

As previously mentioned, our implementation of AES has an input size of 128 bits; output is also 128 bits. With this in mind, we have two blocking modules, creation and destruction, to handle input and output size conversion. The block creation module takes the 8-bit/1 byte output from the previous module (which handles signal processing and filtering) and gathers 16 of them to create a single 128-bit block. Similarly, the block destruction module takes the 128-bit output of the decryption module and breaks it up into individual bytes. Block creation precedes encryption, while block destruction follows decryption.

C. *Timing*

Timing is, perhaps surprisingly, not too much of an issue. Our system effectively outputs a byte of data 12,000 times a second. In other words, it moves bytes at 12 kHz. All of our modules can easily finish within a 12 kHz clock cycle. This allows us to simply wire all of our modules together and retrieve data at a 12 kHz frequency.

IV. Transmission and Reception

A. *Transmission*

Once the audio information has been filtered and processed, it will be transmitted out to other WTF devices. We have decided to transmit this information ideally through a laser, but during testing, the laser-phototransistor combination did not yield enough response time to accommodate our bit encoding scheme, which meant that we had to transition back into transmission through wire. To achieve a robust transmission and reception procedure, we needed to establish a communication standard between the WTF devices. We decided to take inspiration from the IR transmission protocol of lab 7, which contains a sync signal, followed by a series of patterns for sending a 1 or 0. We have established a finite state machine with well-defined transitions and parameterized timing and thresholds that associate to the patterns specific to the protocol. The following figure demonstrates our determined timings for the bit encoding scheme.

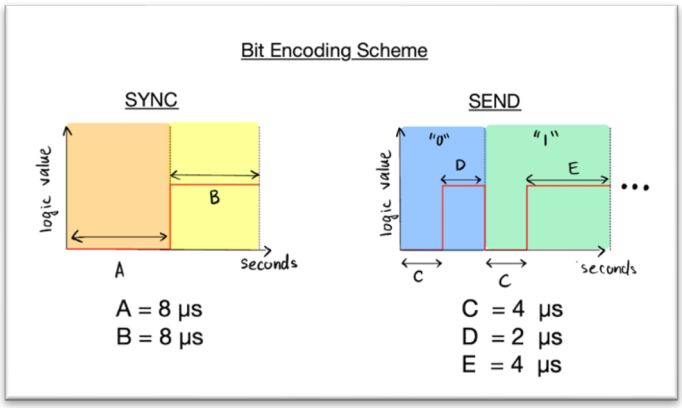

Figure 5: Bit encoding scheme

As seen in figure 4, before a WTF transmits, it must send a sync frame, which begins with a 8µs of logic low followed by 8µs of logic high. Once that sync message has been established, the WTF starts sending out the pattern from the bit encoding scheme. This scheme was designed to be fast enough to convey at least 12,000 samples at 8-bit depth originally. Considering we need to convey 8 bits at 12khz, we need to transmit 96,000 bits per second. Effectively, this specification requires us to send at least 8 bits in 1/12KHz = 83µs. Let's do some quick evaluation of this system. Suppose the transmit module needs to send out a value of 8'hFF, using the revised bit encoding scheme. The sync frame requires 12µs. For the sending procedure, sending a registered 1 requires 8µs. Sending a series of eight 1's will

accumulate 64µs. The accumulated latency would be 76µs, which satisfies our timing constraint.

During development, however, the 8-bit, 12 kHz transmission scheme was abandoned due to high error susceptibility. For example, if one bit is not transmitted, the 8-bit block will be discarded, and since AES encryption requires all 128-bits to be sent correctly, one dropped block will result in corrupted data. We ran into such error during the original development, which made us transition into transmitting all 128 bits during one cycle instead, preventing corrupted blocks as now if one bit is dropped, the whole entire block of 128 bits is dropped, removing the chance of corrupted audio. The cycle will be dictated by the completion of the AES encryption module.

We evaluated the performance of the transmission module through two means: correctness and whether timing is met or not. We used a custom testbench (tx_tb.sv) to do so, checking whether the bit pulses are correspondent to the determined bit encoding scheme and whether all bits are sent before the next cycle starts.

B. Reception

The reception module of the WTF is similar in structure to the transmission module, but in the reverse direction. The reception module must run through a synchronization module to allow for the clock rates of separate FPGAs to be equal.

The reception module will look first for the sync frame, and when that frame is detected, the module will transition into the capturing stage, receiving any bits transmitted by the transmission module, decoding it based on the bit encoding scheme, and build on a size-128 block before passing onto AES decryption.

C. Hardware

The WTF will have a laser circuit attached to its transmission output. This laser, however, only operates effectively using 5V, which means that there must be a method to translate the 3.3V output from the FPGA's PMOD headers to 5V. To achieve this, a non-inverting gain operational amplifier circuit is used to increase the signal voltage to the rail voltage of 5V. This would ensure the output of the FPGA meets the voltage requirements for the laser while simultaneously allowing its voltage to reach 5V.

To capture the laser, a phototransistor is used. The phototransistor, however, does not output a strong enough signal to pass back into the FPGA. A transimpedance amplifier is used to amplify the signal received to a relative level, which then gets passed through a comparator to translate it to a 3.3V-peak square wave to input back to the FPGA.

The laser and phototransistor determined how fast each bit pulse can be. The phototransistor only performed at an acceptable level at below 200 kHz, which was why we determined the timings for the bit encoding, with the fastest signal being 166 kHz.

The laser circuit works in theory and in testing, however, under certain conditions: the phototransistor must be positioned at the right position to output a stable waveform, which proved to be difficult especially with transportation. Since the waveform resulting from the phototransistor is not stable, the comparator voltage must be adjusted as well, which means that the performance will not be equal across test runs. At the end, it was better in practice to simulate the effects of the laser-photodiode combination with a wire to avoid any disturbances resulting from the photodiode. In testing, the laser circuit does work, but in practice, it is better to use a wire.

V. System Diagrams

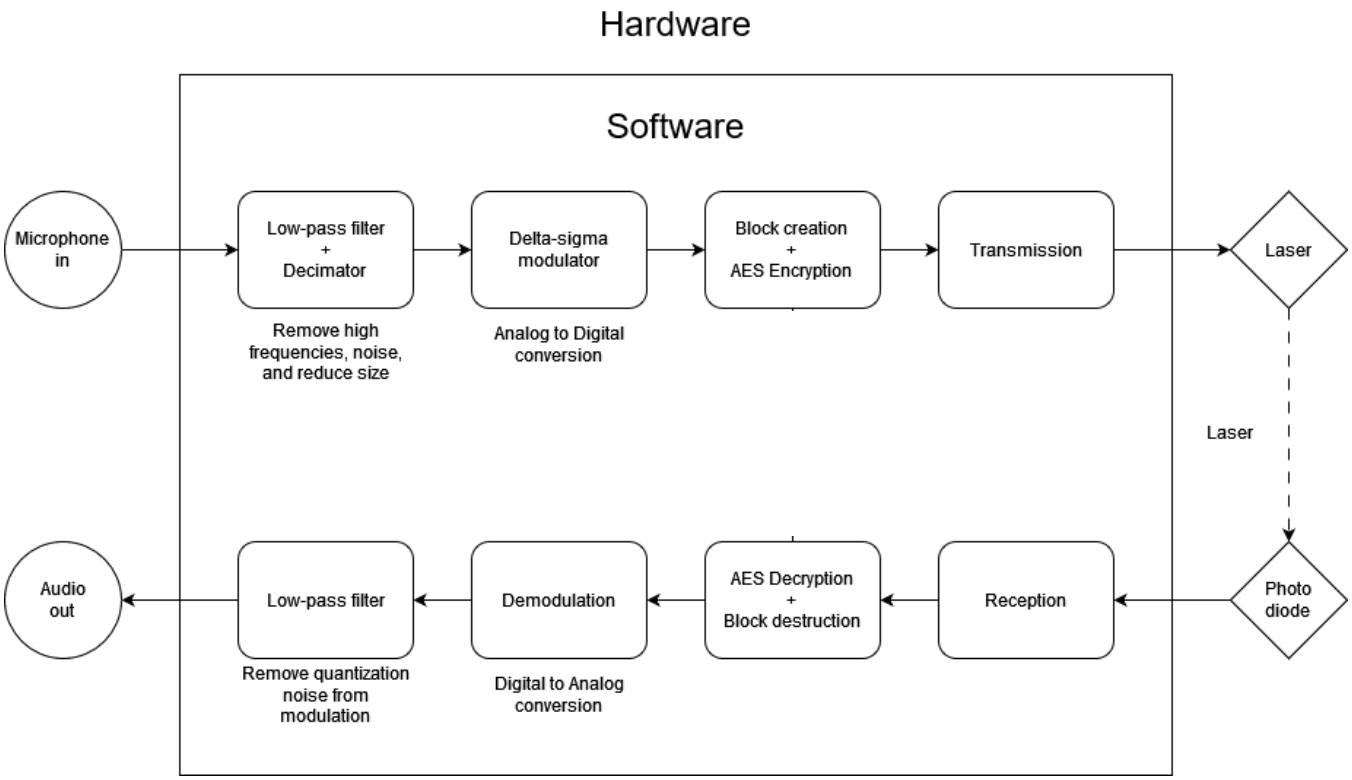

Figure 6: System block diagram

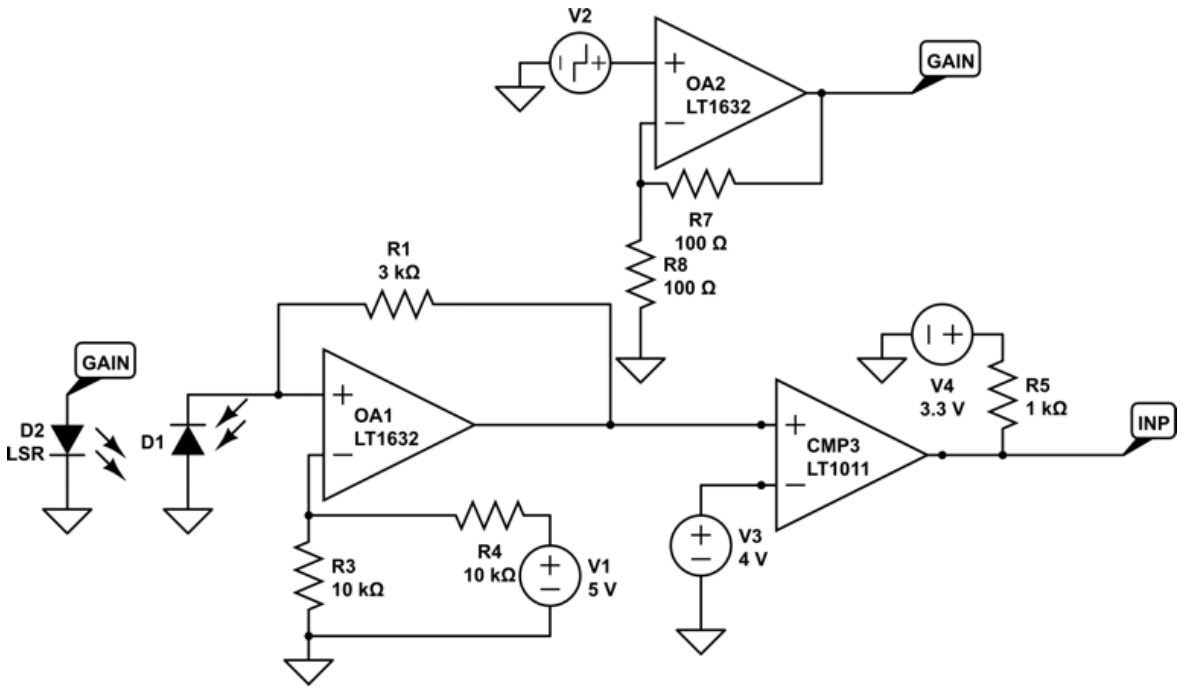

Figure 7: Laser hardware

VI. Author Contributions

Keawe Mann handled all of AES, along with block creation. He, like the rest of the group, also worked on testing, the preliminary report, the presentation, the final report, and the project video.

Khoa Huynh was responsible for designing the FIR and Decimation module, including tuning parameters related to the modules. He also worked alongside Khang to design the bit encoding scheme for the transmission and reception module. Likewise, he also worked on the testing, preliminary report, the presentation, the final report, and the project video.

Khang Le was responsible for developing the transmission and reception modules, both hardware and software-wise, which includes the laser capturing circuitry. As part of the group, he worked on testing and integrating the modules and drafting the project deliverables (preliminary report, presentation, final report, and final video).

References

[1] W. Knitter, “DSP for FPGA: Simple FIR Filter in Verilog”. Hackster.io, March 2021.

[2] “An Introduction to Delta Sigma Converters”. Beis.de, August 2007.

[3] Source code: https://github.com/KhoaHuynh02/WTF-repo