Agilent Technologies

Arbitrary Waveform Generator

Client: Agilent Technologies
Client Representatives: Art Lizotte, John-Michael O’Brien
Team: Matt Buland, Luke Dunekacke, Drew Koelling

Client Description:

Agilent Technologies is a firm that designs equipment for accurate measurements and instrument testing. Some of the more common devices made by Agilent are oscilloscopes and spectrometers. Agilent primarily uses these technologies to test other firms equipment, and ensure that they adhere to the standards set forth by that firm's respective field. This ensures that engineers are confident about their measurements. Electrical circuit ground loops can be difficult to spot and can potentially ruin electronic equipment. Good measurements can help solidify the reliability of products everywhere. This in turn allows engineers to work on making technological breakthroughs, rather than worry about whether or not their measurements are reliable. Agilent primarily works with firms that relate to electronic manufacturing, or medical fields.

Project Description:

The project entails producing a normalized ATSC standard waveform of an MPEG2 stream using an Agilent 33520 series waveform generator. The MPEG2 stream will be created from a JPEG or PNG image. The specific waveform generator is a 33522A, but any waveform generator that can hold arbitrary waveform data and produce this wave is acceptable. The output from the waveform generator can then be transmitted to a T.V. that accepts a digital signal, which will decode the signal, and show the MPEG2 video stream, but the transmission to a T.V. is outside the scope of the project.

Project Vision:

Many of the devices provided by Agilent and their competitors are quite capable in a generic sense. Having a very good demonstration of these devices capabilities is necessary for Agilent to maintain a competitive edge over the rest of the industry. Showing the waveform generators capabilities on technologies that most people have access to, such as T.V., is a very good way of showing the devices capabilities and opening a large potential of future uses for the waveform generator. It would also show that the waveform generator can adhere to the many standards in producing ATSC waves, further ensuring the reliability of the products Agilent provides.

Project Requirements:

The Project description can be broken into 4 high level steps:

1. Convert a JPEG or PNG image into an MPEG2 frame.
2. Encapsulate the MPEG2 frame into an MPEG transport stream.
3. Convert MPEG transport stream into an 8VSB wave.
4. Upload the 8VSB wave to the waveform generator.

While the initial intention was to have the 8VSB wave transmitted to a T.V., this step involves knowledge of RF modulation, which is beyond the scope of the project.

Figure 1:

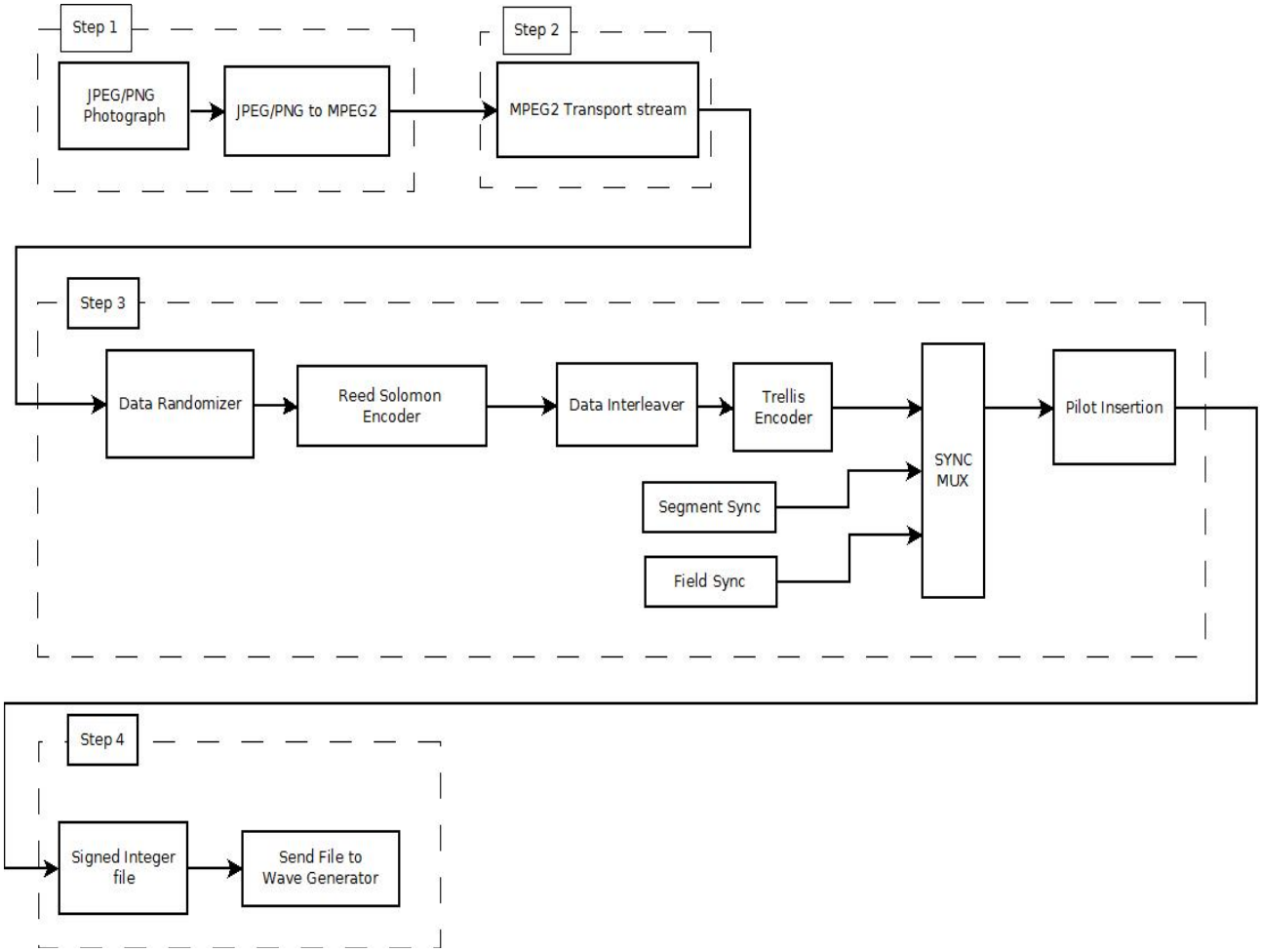


Image into MPEG2 Transport Stream

Converting an image into an MPEG2 transport stream will be accomplished using the FFmpeg library, which will complete the conversion in two steps: First, convert the image into an MPEG2 video, then convert it into a raw MPEG2-Transport Stream.

MPEG2 Transport Stream to 8VSB

In this phase of the process an MPEG2 Transport Stream is turned into an 8VSB wave. This process has the following steps.

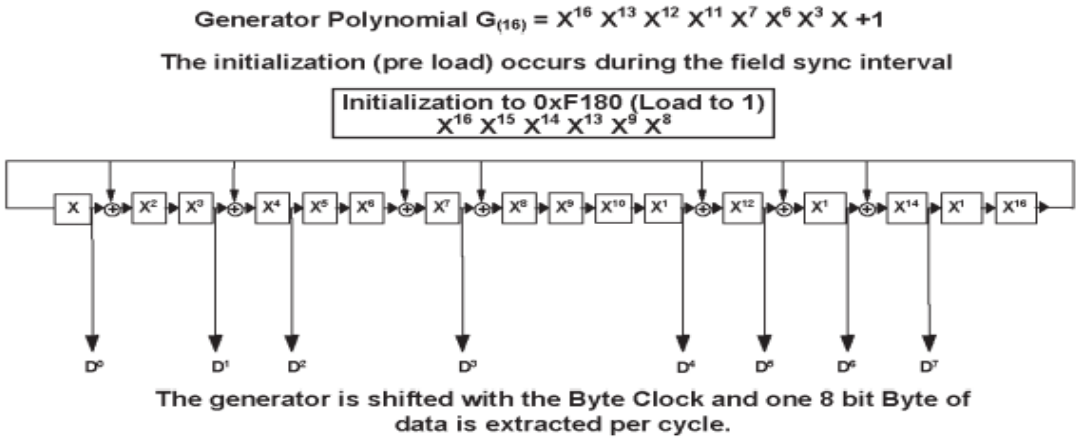
1. Remove sync byte from MPEG2 Transport Stream packets
2. Randomize the input data
3. Generate parity bytes using a Reed Solomon Encoder
4. Interleave the data
5. Run the data through 12 Trellis encoders

6. Apply the segment syncs and the field syncs
7. Do pilot insertion

Data Randomizer: The data randomizer's purpose is to randomize the data payload. This randomizing step is to make the ending wave more noise like. Without the data randomizer, if multiple patterns of bits in the input stream are similar, those bits will cause peaks in the RF spectrum. This would make certain parts of the 6MHz channel overused and other part underused. By equalizing the data spread, more space of the channel is used effectively. The data randomizer is a 16-bit pseudo random binary sequence (PRBS) represented by the polynomial $x^{16} + x^{13} + x^{12} + x^{11} + x^7 + x^6 + x^3 + x + 1$, where x^{16} is the MSB and x^1 is the LSB. The PRBS has 9 feedback traps and 8 shift register outputs that are used for the randomization of the input stream. The 16-bit PRBS is initialized to 0xF180 and is reinitialized after every field sync.

Each of the 8 shift register outputs is shown as a D in the figure below where D0 is the LSB and D7 is the MSB. One byte is processed each iteration. Each bit in the byte is XORed with the 8 shift register output, MSB to MSB ... LSB to LSB. After each byte is XORed, each bit in the polynomial is shifted as shown in the Figure 2.

Figure 2:

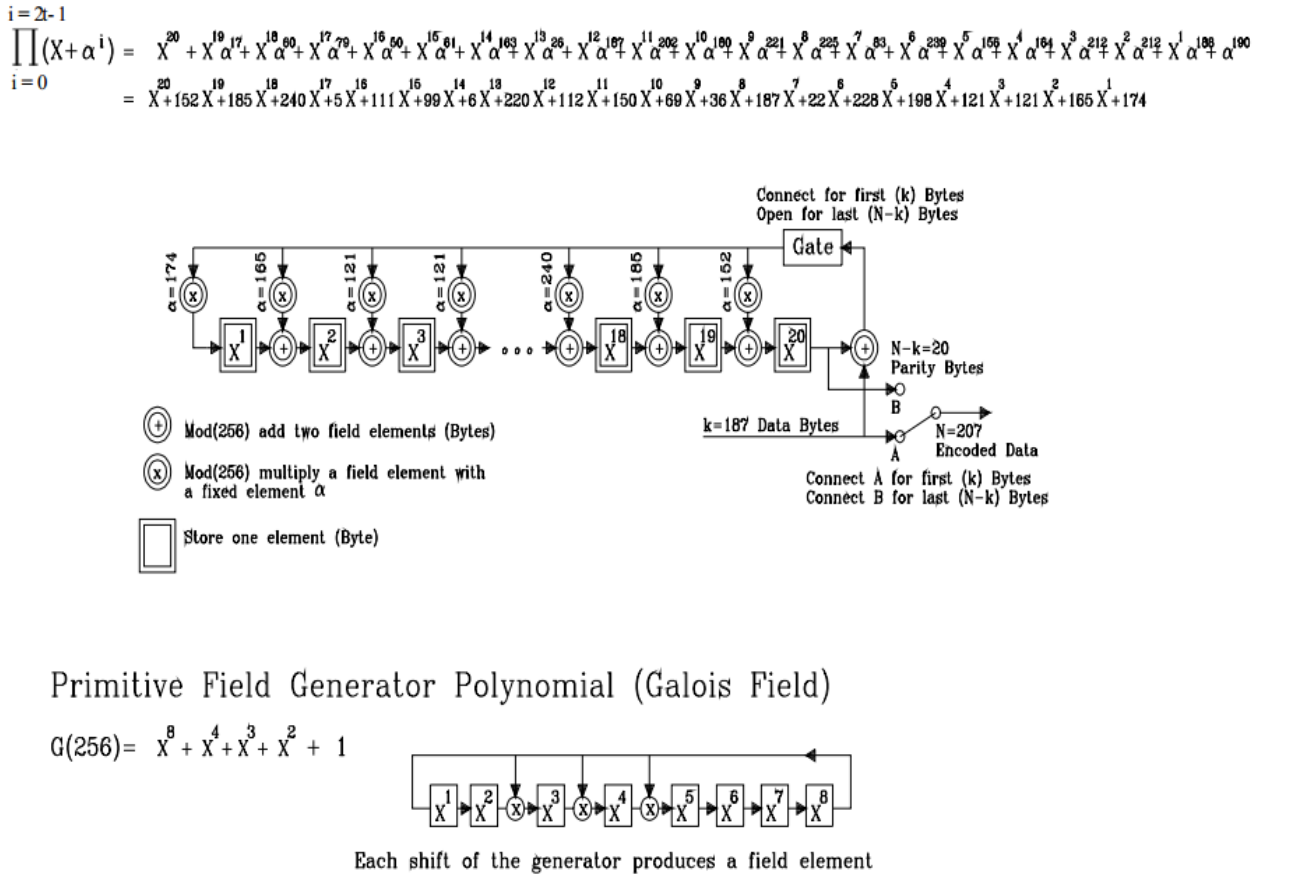


ATSC Digital Television Standard (a_53) – Part 2, page 12

Reed Solomon Encoder: The Reed Solomon Encoder is a well known form of error correction. It creates parity bytes from all the bytes of data after they go through a galois field function. This adds 20 bytes of data to our 187 byte stream, resulting in a 207 byte segment. The addition of the Reed Solomon parity bytes will allow error correction up to 10 bytes per packet. If there are more than 10 bytes of errors, the entire validity of the packet cannot be determined,

and is thus discarded. The galois field used by the ATSC standard is in Figure 3.

Figure 3:



The operations for a Reed Solomon encoding all happen inside of a galois field, or a finite field. An addition inside a galois field is implemented as doing a binary XOR. Multiplication turns out to be a binary long division. The galois field polynomial describes the way the galois field arithmetic behaves. The galois field polynomial for 256 values (8 bits) of $x^8 + x^4 + x^3 + x^2 + 1$ describes Rijndael's finite field, which is a popularly used field for polynomial coding. Thankfully, arithmetic implementation was provided on the wikipedia page on "Finite field arithmetic." This was extremely important for the implementation of Reed Solomon parity generation.

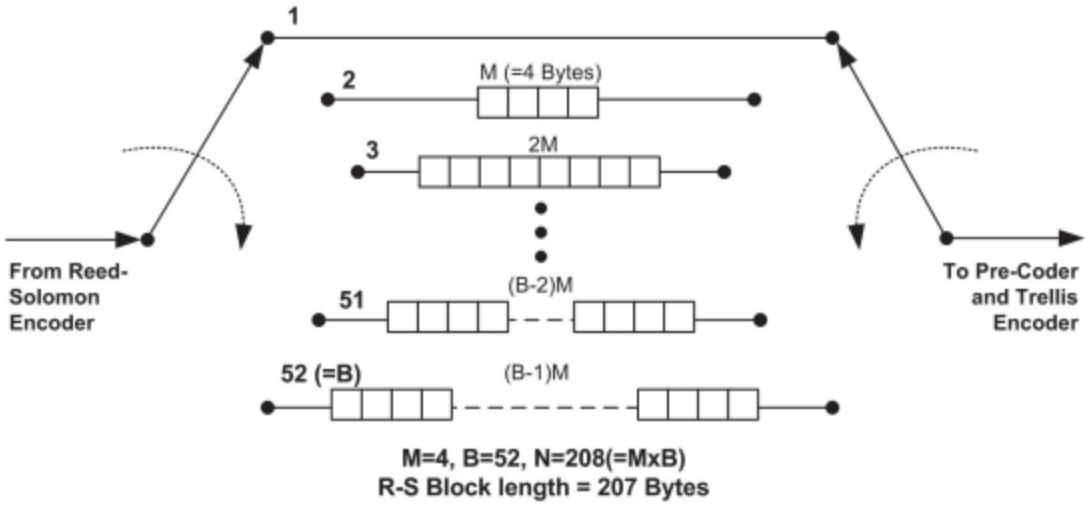
RS encoding happens on the byte level, and is reset for each data segment. The initial value for all of the states is the byte 0 (all zeroes). For each byte in the segment, the byte is propagated through the RS system, recalculating the output bytes ($X^{1 \text{ through } 20}$). The input bytes are added to the last output byte, sent through the gate, then multiplied by the coefficient for the position (alphas), and the output byte becomes the sum of that product and the previous output byte. The

gate can be thought of as a single-byte storage unit, which helps to understand what happens when the gate closes. While the input bytes are being read into the system, the output is the same as the input byte. This solely states that RS parity bytes will be appended the end of the input bytes. Once all the input bytes are read in, the state of all the output bytes is the parity bytes, with X^{20} as the first byte, and X^{1} as the last byte. In a true implementation, the iterations would continue, and the 'output' of the gate would be 0, so the bytes would transition from one place to the next, reading them out in order.

Data Interleaver: The data interleaver reorganizes the data so consecutive bytes of data are not stored beside each other. This is done so that if there is a burst of noise in the transmission and a group of bytes get lost, when the data is reorganized into the proper order only some parts of several code blocks are missing rather than one large chunk of a single code block. Reed Solomon error correction can reconstruct the smaller missing segments of each code block but may not be able to reconstruct one large missing code block. Data blocks will be interleaved at the maximum time difference of 4.5 ms.

The data is interleaved by storing different bytes of data into different queues. Where each queue will have a (B*M) delay where B is an increasing integer for each queue and M is 4 bytes. Starting with a 0 byte delay, bytes are added into the queues in each step. In each step, B is increased by 1 and returns to 0 after it reaches 52. In each iteration a byte is added onto the current queue and if a byte is pushed off the queue it gets outputted, which creates a (B*M) delay queue.

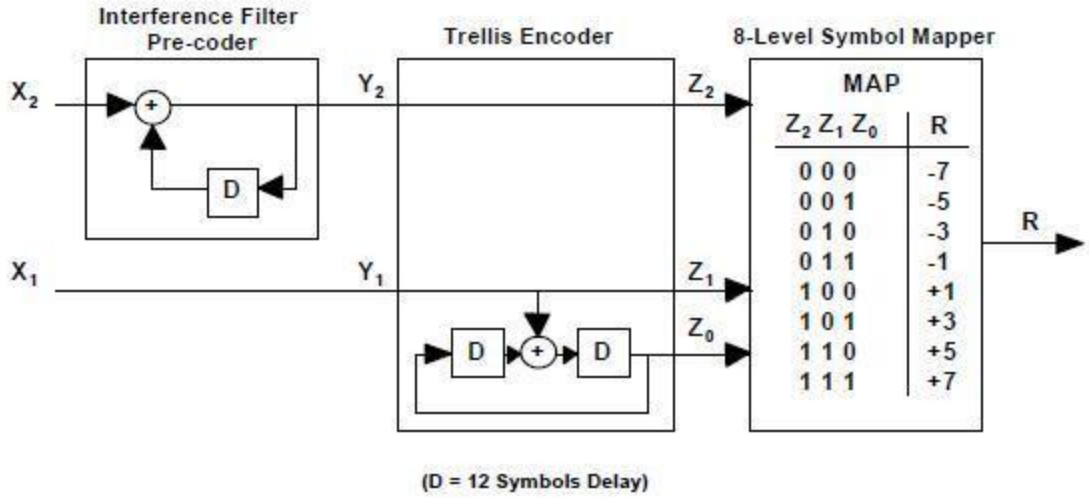
Figure 4:



Trellis Encoder: The Trellis Encoder takes the concept of encoding and modulation and combines them into a single step. By following a very specific convolutional code, we can modulate our code and make it easier to correct for errors by being able to detect invalid state transitions. If the decoder gets an invalid state transition, it can detect it and possibly correct it. A good analogy of this process is following a set of footprints in the sand. When the footprints enter an area with a bunch of other footprints or this case errors, you can follow many paths of footprints out. You can then determine which footprint paths don't make sense, and get back on the correct trail. By adding a bit, we make the transmission longer, allowing for more redundancy, but without affecting the bandwidth or power requirements for sending and receiving the wave. Think of this as carrying a 10 Kg mass 10 times as opposed to a 100 Kg mass 1 time. It takes more time to carry the mass 10 times, but it also puts much less strain on the carrier.

The D's in the image are bit buffers. They are initialized to be all 0's, and they store 12 bits before outputting the symbol. If at t=5, a 1 is inserted into a bit buffer, at t=17, the buffer outputs that 1. This process allows the signal to be longer, making it easier to transmit and at the same time having a very specific set of state transitions, making it possible to detect errors on the receiving end. The trellis encoder will output voltage levels defined by the 8 level symbol mapper (Figure 5), rather than bits.

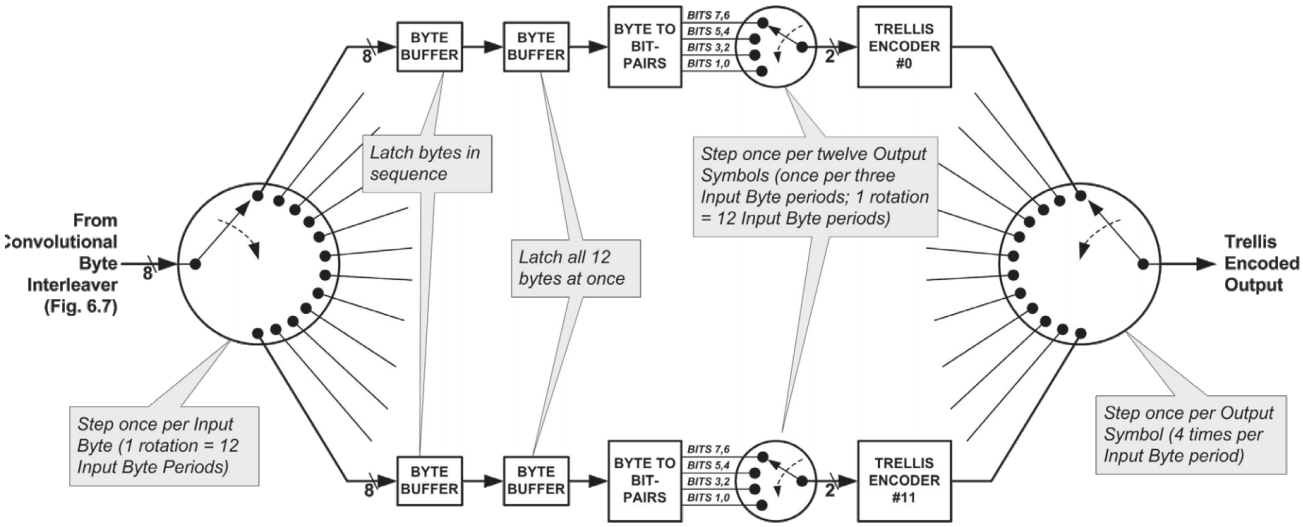
Figure 5:



Fundamentals of 8VSB, page 2

Figure 5 provides the implementation of a single trellis encoder. The actual implementation of the trellis encoder states there are 12 encoders and their use is determined by segment number and particular byte number. This will interleave the history of symbols by intra-segment only. The history symbol that influences the next segment is actually redundant information. Figure 6 shows this concept.

Figure 6.



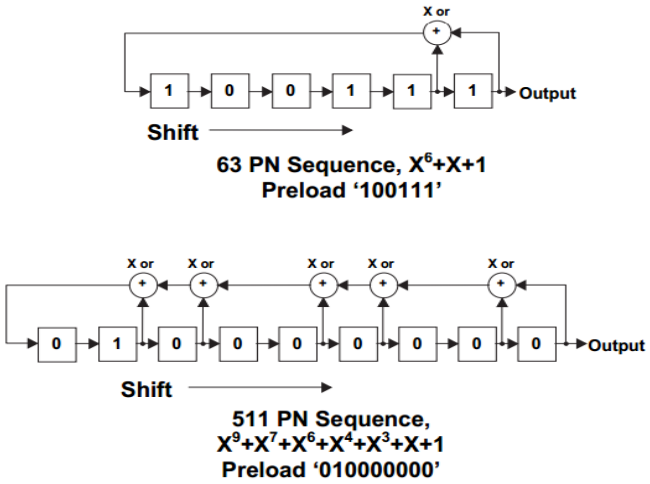
The first segment of each field will start at trellis encoder 0, send byte 0 to it, then send byte 1 to encoder 1, and so on until it sends the 11th byte to encoder 11, and then restarts back at encoder 0, sending it byte 12.

The second segment would do the same thing, except it starts at encoder 4, sending it the 0th byte. This continues with the 11th encoder receiving the 7th byte, and the the 11'th byte being sent to encoder 3. The third segment would continue this pattern, but start at encoder 8. The fourth segment would then start at encoder 0, and continue until all segments have been sent through.

Sync Mux: During the NTSC days, the need for "sync pulse" arose. Receivers had a hard time homing in on a signal if there was a lot of noise or errors. With a sync pulse before every section of data, the receiver could stay homed in on the frame, even if the packet contained errors or heavy ghosting. Tuning in the receiver clock is also important because it can let the receiver know where it is in the transmission. The "sync pulse" is achieved by the field sync and the tuning of the receiver clock is accomplished by the segment sync.The SYNC MUX adds the segment sync at the beginning of each segment and the field sync after every 312 segments. A segment consists of 828 data and FEC symbols plus 4 segment sync symbols creating a 832 symbol segment. The segment sync consists of 4 symbols. The 4 symbols will always translate to +5, -5, -5, +5. This makes the starting of each segment noticeable. The field sync is inserted every 312 segments and contains various symbols to excite the receiver, as well as correctly home in the signal. This field sync is broken up into many blocks, the exact order is a 511 symbol pn511 block, 3 pn63 blocks each consisting of 63 symbols, a 24 symbol mode flag, a 93 symbol pn63 block, and finally, the last 12 symbols of the last segment. By having a sync mux,

the receiver clock can be properly aligned. By having a field sync, you can eliminate any signal ghosting or common noise interference. Think of this as looking north again to get a better idea of where southwest is from your current orientation.

Figure 7:
The pn511 block sequence and pn63 block sequence as defined by the ATSC standard



Pilot Insertion: Every symbol will be increased by 1.25 volts. This makes it so the resulting average signal requires -11.3 dB, and making a residual carrier appear at the 0 frequency point so the receiver to lock onto something other than the data.

Uploading 8VSB data points onto the Agilent 33522A/B wave generator

This step will include the process of taking the 8 voltage levels and turning them into a normalized set of 16 bit integers. The waveform generator will take these numbers and output an arbitrary wave. Interfacing with the generator can be done through a telnet connection or through an ethernet cable; it also supports usb mass storage, so files can be uploaded to the generator through usb as well. Our method of upload will work through creating a TCP socket and uploading the data through a LAN connection.

Displaying the MPEG2 Transport Stream to 8VSB image on the TV

As of now, getting the image to the T.V. involves RF modulation, which is outside the scope of the project. Currently however, work is being done into taking the 8VSB wave from the generator, sending it to an NTSC RF transmitter through a 50 ohm BNC male to 75 ohm RCA female adapter, and feeding that RF output over coaxial cable into an ATSC compatible T.V. tuner. Additional data must also be added to the stream to describe the channel being broadcast. The Program and System Information Protocol (PSIP) is the standard for this information. It's transmitted at regular intervals, and offers a reference for the ATSC tuner to decode, and interpret as a channel.

Design issues:

Currently due to the nature of the project, we can only tell if we're doing everything correctly when we get the image on the T.V. A lot of the modulation is at the byte level, so until those bytes are decoded, we can only tell if we are in the general ballpark. Testing the project code is a very big issue. The most we can do for testing is checking the size of byte streams and validating known sequences. Most of the data is and should be random, making the only real way to test the code is by going through every step and showing the image on the screen.

Technical Decisions:

The project document described that any programing language could be used, but we decided to go with C++ due to the built in optimization that vectors provide when dealing with booleans. C++ will group together a vector of booleans into consecutive bits, so a boolean value will only take up one bit, as opposed to the standard byte it takes normally. C++ also makes it really easy to perform bitwise operations, which a major part of the project involves, as well as more control over memory allocation. It was also one of the preferred languages by the client.

Most of the implementation is defined by the ATSC A/53 standard, which means the language chosen is irrelevant in terms of implementation, because the implementation must be the same for every step as defined by the ATSC A/53 standard.

For JPEG/PNG to MPEG2 frame conversion, we decided to go to an open source library called FFmpeg to perform the conversion. There is no reason the project should involve re-implementing the conversion of MPEG2 frames. FFmpeg will also perform the process of encapsulating the MPEG2 frame into an MPEG2 transport stream.

Each step of the program flow is very imperative. This made it logical to go with a C++, but with a C paradigm. Making class abstractions was unnecessary, due to the nature of the standards set forth by the ATSC. Almost every step in the conversion is its own file and function.

Though pilot insertion is essentially negated when we upload to the waveform generator, we decided to keep it in because the normalization of our voltages is specific to this project and pilot insertion is a vital part of the 8VSB concept.

We decided to upload the data to the Waveform Generator through TCP sockets because TCP sockets can be easily scripted in comparison to telnet and USB mass storage. The USB driver controller on the back was also a possibility, but due to the ability to create sockets on both a Windows and Linux platform (winsock and sys/socket.h), we went with TCP. The Agilent website also provides C code to create sockets to interface with their devices, so the TCP setup was relatively straightforward on the Linux end, resulting in more time to test the wave on the oscilloscope.

Results:

Inside the scope of the project, we were able to create an 8-VSB wave using the Agilent waveform generator, and view the 8 separate voltage levels in the wave, as well as the recurring segment and field syncs. This means that the trellis encoding, pilot insertion, segment sync, and field sync are correct. The parts that cannot be fully tested are the data randomizer, reed-solomon encoder, and data-interleaver. These components can only be tested by verifying the signal with an ATSC decoder.

Outside the scope of the project, we have been unable to produce an RF modulated channel that can be recognized by an ATSC T.V. tuner. This would have made for a great demonstration, but we are missing a key component: Namely, the Program System Information Protocol, which is a set of tables that describe the content of the channel.

Glossary:

8VSB: 8 Vestigial sideband modulation, it is defined as the modulation method for the broadcasting of digital T.V.. It is characterized by its 8 distinct voltage levels.

Arbitrary Wave: A wave that does not follow a specific function, rather a set of points. It is often described with amplitude and symbol rate, rather than amplitude and frequency.

ATSC: "American Television Systems Committee", it is the organization that defined the standards for the transmission of digital television.

Carrier Frequency: The frequency the carrier wave is sent at.

Carrier Wave: A wave that is modulated to convey specific information, it is generally at a much higher frequency than the rest of the input signal. In the 8VSB case, it allows multiple frequencies(channels) to share the same transportation medium.

Data Field: 313 total segments, consisting of 312 data segments and 1 field sync segment.

Data Segment: an 832 symbol packet, with 4 symbols being the segment sync and 828 being the payload data.

Forward Error Correction (FEC): A technique used in data transmission where the receiver corrects errors in the transmission rather than having the sender send the information a second time.

LSB: Stands for Least Significant Bit.

MPEG Transport Stream: The system layer of MPEG formats, it is the way data is organized during transmission. They contain a series of packets containing a 4 byte header with 184 byte payload, resulting in a 188 byte packet. All Transport Streams are a multiple of 188. In 8VSB modulation, the first byte of the header is removed. leaving a size of 187 bytes during the first step of modulation.

MSB: Stands for Most Significant Bit.

NTSC: "National Television Systems Committee", it is the organization that defined the standards for the transmission of analog television.

PN: Pseudo Noise. It is usually followed by a number, indicating how long the block of Pseudo Noise is.

RF: Radio Frequency

Symbol: a pair of bits or an integer representing 3 bits. Before trellis encoding, a symbol is 2 bits. After trellis encoding it is an integer that represents 3 bits. The integer to bit map is {7, 5, 3, 1, -1, -3, -5, -7} = {111, 110, 101, 100, 011, 010, 001, 000} respectively.

Symbol rate: The number of points on a wave transmitted per second. Mathematically, it is just #symbols*Frequency_of_wave. In the 8VSB case, the symbol rate is always 10.7 MHz.

References:

ATSC A/53 Part 2: http://www.atsc.org/cms/standards/a53/a_53-Part-2-2011.pdf

Agilent Waveform Generator 33500 Series Manual and API: http://cp.literature.agilent.com/litweb/pdf/33500-90901.pdf

ATSC A/69(PSIP): http://www.atsc.org/cms/standards/a_69-2009.pdf

Agilent TCP socket code: http://www.home.agilent.com/upload/cmc_upload/All/sockets.c?&cc=US&lc=eng