

**$\pi/4$  Quadrature Phase  
Shift Keying  
Greg Galloway  
ESE: 433  
February 17, 2007**

## **Table of Contents**

Table of Contents.....	2
Introduction .....	3
Procedure.....	3
Design.....	4
Terminology .....	4
Design Walkthrough.....	5
Result Table.....	6
Example Matlab Output.....	7
Modulation Example .....	8
Improvements .....	9
Noise Considerations.....	9

## **Introduction:**

For some, modulation schemes can be a tricky concept to grasp. They understand the individual components of the process, but find it hard to keep track of the relationships between the various steps. I was one of these people. I decided to not only design the process and plan it out, but to carry through with the design on simulation tools. This helped me reach two primary outcomes. For one, it seemed that in order to implement the process and have it behave as it should, a complete understanding of the system would be necessary. This proved to be true. Also, this allowed questions to be easily answered with manipulation of the variables, the signal flow, and various graphic displays.

I feel this information should be freely shared. Although I found the entire process of design and creation enlightening, enthralling, and entertaining, there might not be many who share my enthusiasm. If someone wants to learn this material, I would like to assist in any way I can. For this reason I have attempted to make the 'final' design as straightforward and clear as possible, for future use. Additionally, I have written this report to discuss the process and details that might not be clearly observable through the code and simulation.

## **Procedure:**

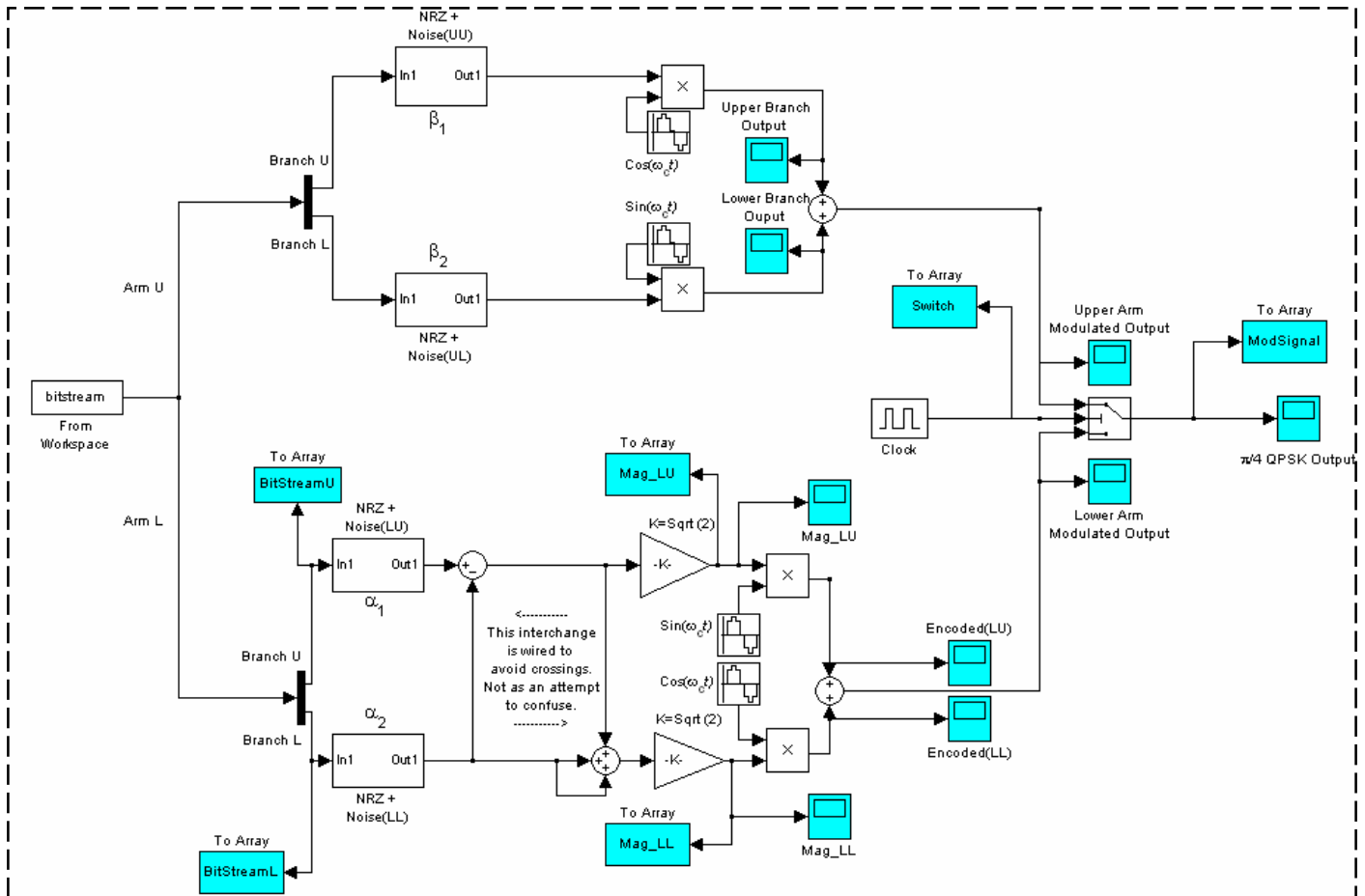
Pi/4 quadrature phase shift keying is actually a relatively basic concept, but like all modulation schemes it requires a certain level of understanding with the processes in general. Knowledge of phase and frequency of sinusoids is obviously vital. However, it takes not only intimate knowledge but artistic talent to draw graphs clearly, particularly when you're dealing with specific ranges, addition of signals, and frequent phase shifts.

Computer analysis is a clear choice for accurate graphs. I chose to use Matlab and Simulink. These programs are relatively straightforward to use, are installed on nearly every computer on campus, are applicable to many classes, and have help files on every command.

I started by putting the basic blocks down and testing the model frequently to see if it was doing what I thought it should be. Each time I added new blocks I would set the values as Matlab variables. This allowed for easy customization and testing, as well as the ability to automate tests at a later point.

The design process involved many comparisons between what I was designing and the basic configuration outlined in class. The resources on the Internet were very limited, so I was frequently forced to figure out on my own that I was incorrect about an assumption I had made.

## Design:

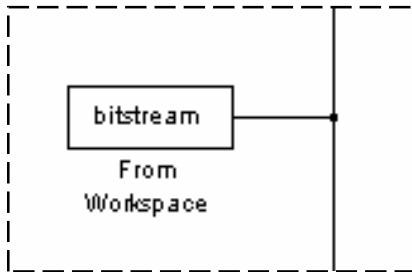


## Terminology:

A *symbol* is a set of bits from the incoming stream. The symbols I am modulating in this procedure were two bits long. I will refer to an arm as the paths after the initial split, as there are two of them. The top arm in this design is the  $\beta$  arm, and the lower arm is the  $\alpha$  arm. I will refer to the demultiplexer output paths as branches; there are four of these. The top branch of each arm will handle the first bit of the symbol, or bit X of the pair XO.

## Design Walkthrough:

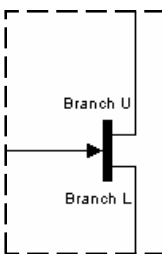
### Bitstream:



The incoming data was to be modeled as a single stream of multiple bit symbol length. I had to choose how I would represent this. I decided to make it as basic as possible, and do the manipulations through Simulink and Matlab, in order to properly demonstrate my understanding of the incoming signal. I decided to use an array of bit pairs as the starting point for the data

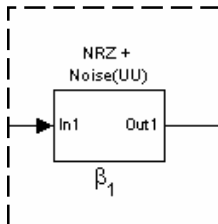
stream. I then added a line of code to give them a time index. Once I met the formatting rules that Simulink requires, the stream was represented in Simulink as a single signal.

### Demultiplexer:



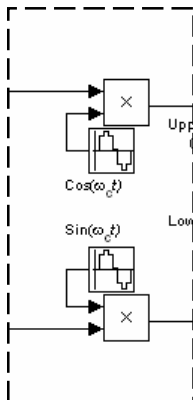
This block separates the two-bit signal and sends one bit to each branch of the model. This step is also known as serial to parallel conversion.

### Non-Return to Zero (NRZ):



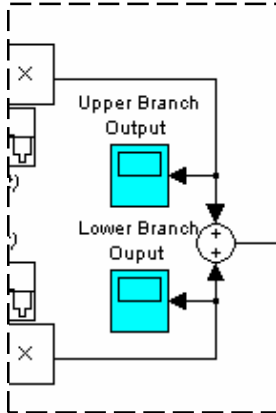
The next step involves changing the zero and one values to positive and negative one. This can be implemented in a few ways. In reality it might be implemented with a negative half volt offset, and then putting the result through an amplifier with a gain of two. In Simulink I simply used a function block that did a similar procedure. It takes the input,  $u$ , multiplies it by two and subtracts one. I also chose this step to add noise to the signal, which I will discuss later.

### Multiplier:



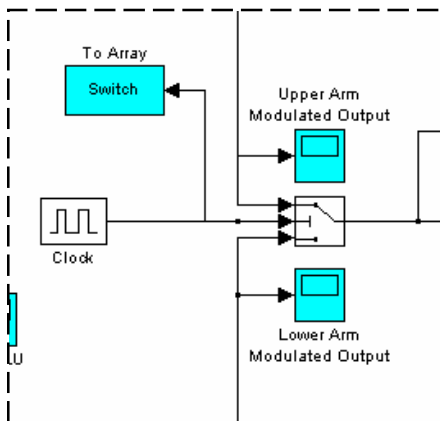
This is where the data is combined with the carrier through a simple multiplication. In the upper arm this results in a  $\pm$  sine wave and a  $\pm$  cosine wave. In the lower arm this results in a sine or cosine wave with an amplitude of  $\pm\sqrt{2}$ .

**Adder:**



This component combines the two waves together, which combines the upper bit and lower bit into one modulated signal. The blue objects are scopes that allow the user of the Simulink model to view the output at those locations.

**Switch:**



This section of the model alternates between the upper arm and the lower arm once per symbol. This allows the two signals to be combined to avoid phase shifts of 180 degrees.

The clock determines which arm the switch passes forward. The *to array* block sends the data to Matlab for manipulation and viewing.

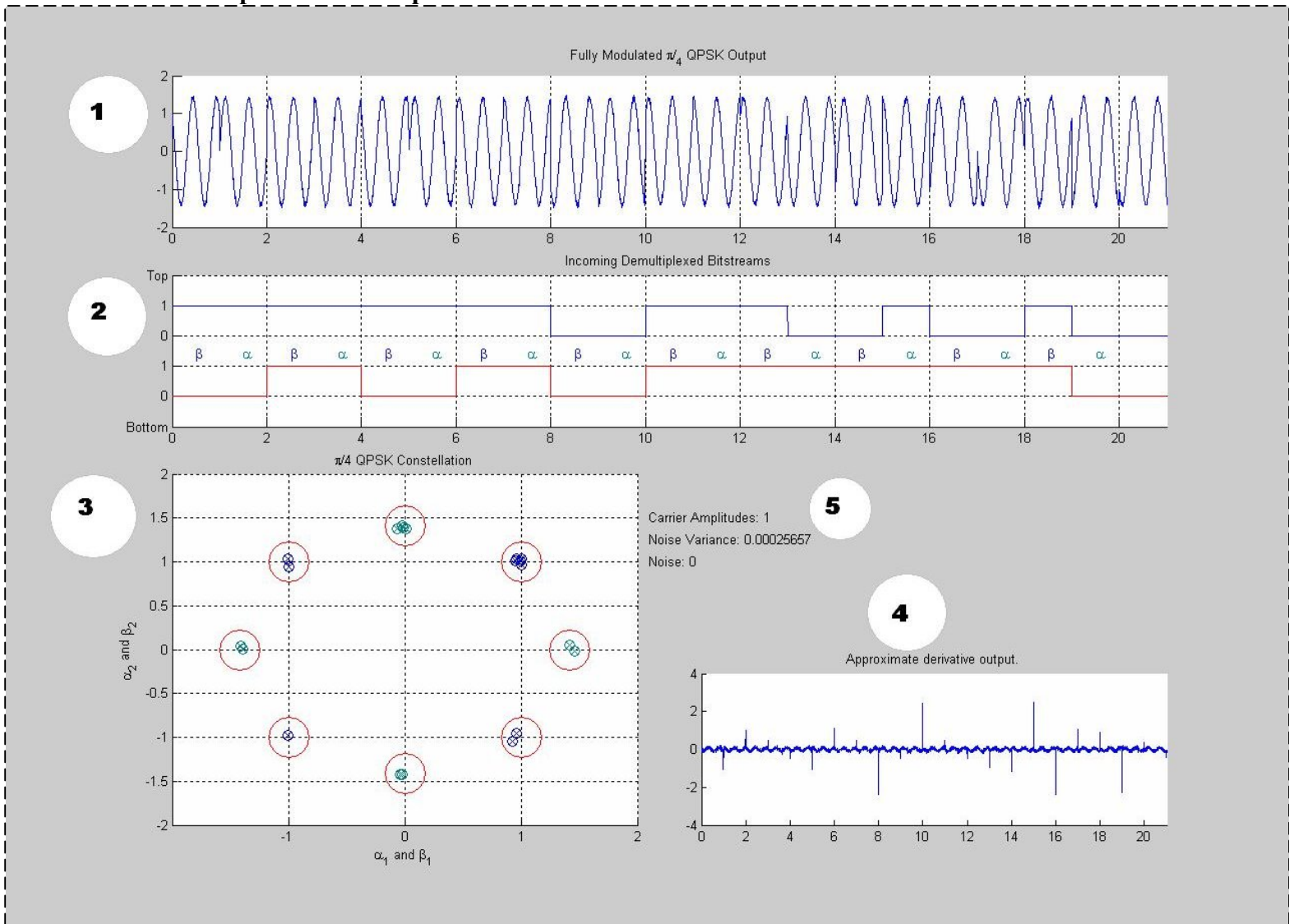
**Lower Arm:**

The lower arm follows a similar procedure, but instead generates amplitudes of 0 and  $\pm\sqrt{2}$ .

**Result Table for all possible input symbols:**

Bit Pair	$\beta_1$	$\beta_2$	$\alpha_1$	$\alpha_2$
00	-1	-1	0	$-\sqrt{2}$
01	-1	+1	$-\sqrt{2}$	0
10	+1	-1	$+\sqrt{2}$	0
11	+1	+1	0	$+\sqrt{2}$

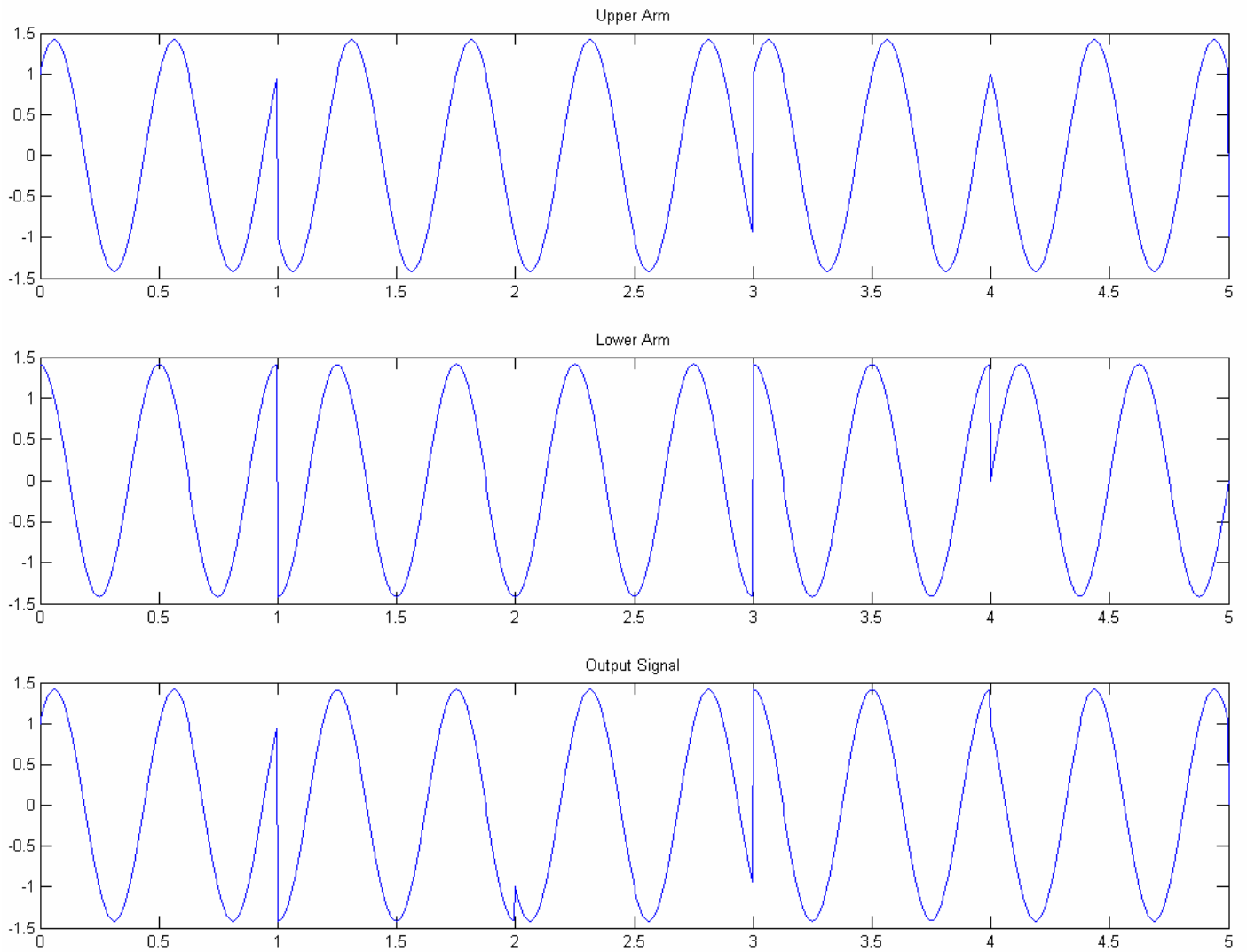
## Example Matlab Output:



### Briefly:

1. The fully modulated output generated from combining the two carrier waves.
2. The incoming bitstream after being separated into top and bottom bit. Also shows which arm it is assigned to as it displays the 'hit' on the constellation. Blue is upper bit, or bit 1. Red is lower bit, or bit 0.
3. Constellation that displays the magnitudes of the alpha and beta that were generated by the incoming bits. Simple noise has been added to prevent the hits from overlapping. The alpha hits are 45 degrees counter clockwise from their beta counterparts.
4. An approximate derivative of the output signal. This helps demonstrate what a component might experience at the transitions.
5. Displays the random parameters so their effect can be taken into consideration.

**Modulation example (For an input signal of 11 00 00 11 10 01):**



**As can be seen, the phase shifts of 180 degrees are avoided by alternating between the two modulation schemes.**

**Improvements:**

If time were not an issue, many improvements would be made to the Matlab code, however the model itself needs little more modification other than clarifying certain sections and allowing more of the system variables to be modified through the Matlab script. Noise, which makes the model substantially more interesting, would need to be worked through with both the model and the code.

**Noise:**

The noise was only added to make the constellation more interesting. If there were no noise, every constellation point would overlap with every other point that had the same destination. Different noise types could also help improve the understanding of how phase affects the constellation. Phase noise would cause the spread to be more circular. Additionally, the noise is not added evenly between the upper and lower arms due to the way the bits are mapped. This could be remedied with a little thought, or by changing the lower arm to adjust phase in the sine and cosine rather than through the amplitude.