Google Summer of Code 2018: MIMO basics for GNU Radio

Moritz Luca Schmid

Karlsruhe Institute of Technology
luca.moritz.schmid@gmail.com

March 24, 2018

1 Introduction

The project MIMO aims to introduce a basic MIMO capability to the GNU Radio core module gr-digital. Basic encoding and decoding algorithms are implemented and bring GNU Radio users the possibility to enjoy the many benefits of MIMO. A special focus lies on the inclusion of MIMO into the existing OFDM physical layer of GNU Radio, forming a very powerful and popular combination, MIMO-OFDM.

2 Multiple Input Multiple Output

The use of multiple antennas at the transmitter and receiver (MIMO) is an essential method in wireless communications. It uses space as an additional degree of freedom in form of spatially separated antennas at the transmitter and receiver to significantly improve the performance of a communications system. **Spatial diversity** can be used to mitigate the effect of fading by combining independently fading signal paths at the receiver [1], thereby achieving a increased signal to noise ratio (SNR). **Spatial multiplexing** obtains independent spatial paths at the same frequency and time which can be used to transmit data parallel on these paths and therefore enormously increase the channel capacity.
In the last few decades was put a lot of effort into the development and research of algorithms which could exploit the mentioned effects as efficient and effective as possible. The goal is, to converge to the theoretical Shannon channel capacity without disregarding the complexity and feasibility for practical use cases.
The easiest way to make use of spatial diversity is the reception with multiple antennas and combining the received signals in such a way, that the SNR is improved and fading effects are mitigated. Selection combining selects the antenna with the strongest signal while Maximum-ratio combining uses a phase corrected superposition of the signals of all antennas, weighted according to their SNR.
Using the two dimensions time and space in a constant frequency spectrum, the transmit data has to be distributed among the spaced antennas and across the different time slots, hence speaking of space-time codes (STCs). **Space-time block-codes** (STBCs) were introduced as a less complex alternative to common STCs, having a linear receiver complexity while still extracting excellent diversity gain [1]. The **Alamouti code** is a very popular STBC, because it is the only orthogonal STBC with a code rate of r=1 [2]. It uses a fixed number of two transmit antennas and needs no channel state information at the transmitter which makes it a very simple and elegant code

to achieve full diversity.
A simple method to achieve spatial multiplexing by parallel encoding of data is introduced in the Bell Labs Layered Space Time (BLAST) architecture [1]. Especially vertical BLAST (V-BLAST) offers an elegant and non complex way for spatial multiplexing: The encoding process is simply a demultiplexing operation followed by an independent symbol mapping of each substream without the necessarity of any inter-substream coding [3].

2.1 MIMO-OFDM

Especially in applications with high data rates, the channel bandwidth is often large relative to the multipath delay spread of the channel, causing the fading to be frequency-selective. An equalization of the channel is, especially for MIMO systems, rather complex, caused to the need of an equalization in time and space[1]. Orthogonal Frequency Division Multiplexing (OFDM) is a multicarrier system which converts the wideband frequency-selective channel into a parallel set of narrowband and therefore flat channels. The combination of frequency and space diversity can be achieved by an OFDM system with MIMO on each subcarrier. MIMO-OFDM has shown very promising performance results and is considered in a number of wireless standards, for example 4G (LTE), 5G and 802.11n (WLAN).

3 Deliverable features

MIMO is not a standard application in GNU Radio yet. Although there are several projects based on MIMO with GNU Radio, they are all rather specialized on their application purpose. This project aims to fill this lack by introducing a unified interface for basic MIMO applications in the GNU Radio core. Therefore, several MIMO algorithms are implemented in GNU Radio blocks. At first, these blocks stand for themselves and can be used as basic MIMO blocks in many projects. As a demonstration of how to integrate the MIMO capability into a transmission system, the second part of this project includes the realized MIMO algorithms into the existing OFDM transceiver of GNU Radio.

- **Implement diversity combining block**: The diversity combining block includes the easiest way of gaining spacial diversity and only applies for the reception side. The planned C++ block works in a synchronous way, it combines N input streams of complex items to one output stream. I am planning to implement the following combining schemes:

  1. Selection combining
  2. Maximum-ratio combining
  3. Switched combining (optional)
  4. Equal gain combining (optional)

  Selection combining and maximum-ratio combining are the most popular combining methods and will be implemented at first. If there is spare time at the end of GSoC, I am planning to add other interesting combining schemes.

- **Implement MIMO channel model**: The use of a channel model for simulation purposes, before transmitting over-the-air, has many advantages for the developing of MIMO algorithms. The degrees of freedom are limited and can adjusted manually. Respectively this project, it is sufficient to implement a static and frequency flat channel, simply representing the constant channel matrix $\mathbf{H}$ which combines the symbol vectors of transmitter $\mathbf{x}$

and receiver $\mathbf{y}$ in a linear way. With M transmitting and N receiving antennas, $\mathbf{H}$ has the dimensions $M \times N$. With an additive white Gaussian noise vector $\mathbf{z}$, the transfer function is $\mathbf{y} = \mathbf{Hx} + \mathbf{z}$ [4]. The channel can be assumed as flat due to the use of OFDM in case of a wideband system (in this case, each subcarrier would have a different channel matrix) and it can presumed as slow due to a sufficient rate of channel estimations via training symbols. The realization of such a MIMO channel block allows us to separate the development of the MIMO algorithms from the channel estimation.

- **Implement space-time coding blocks**: The realization of STBCs in a transmitting system includes an encoder as well as a decoder. Both are going to be implemented as GNU Radio blocks in C++.

    1. **Alamouti code**: The Alamouti encoder can be implemented in a simple block which produces two output streams that are calculated after the rules of the Alamouti code from the input stream. A touple of two input items produces a touple of two output items at each output port respectively. No channel state information (CSI) is required for the encoding. The decoder block consumes the received stream and estimates the symbols by combining the touples in such a way, that the resulting signal only comprises the signal parts of one of the two orthogonal encoded symbols which form a touple. This separation of symbols requires CSI at the receiver which has to be provided with a channel estimation.
    2. **V-BLAST**: The V-BLAST encoder does not apply any coding but only demultiplexes the data stream into M parallel streams. This technique is also called direct-mapped MIMO. Initially, the number of transmitting antennas will be limited to M = 2 and 4. The decoder block consumes M input streams and produces the same amount of output streams that contain the separated signals. The detection for V-BLAST will be realized with a linear approach after [3], using linear combinatorial nulling. The idea is to linearly weighting the received signal to maximize the performance regarding an optimization criterion, for example the complete reduction of inter-symbol interference (zero-forcing) or a minimum-mean-square error (MMSE) approach [3].

        (a) **Zero Forcing (ZF)**: The ZF equalizer simply solves the linear system $\mathbf{y} = \mathbf{Hx} + \mathbf{z}$. The resulting estimate of the transmitted symbol vector can be calculated with the pseudo inverse of $\mathbf{H}$ to $\tilde{\mathbf{r}} = \left((\mathbf{H}^*\mathbf{H})^{-1} \mathbf{H}^* \right) \mathbf{y}$.

        (b) **Minimum Mean Square Error (MMSE)**: The MMSE equalizer strikes a balance between completely canceling interfering streams (like ZF) and amplifying noise [5]. The resulting estimate is $\tilde{\mathbf{r}} = \left((\mathbf{H}^*\mathbf{H} + \frac{1}{SNR}\mathbf{I})^{-1} \mathbf{H}^* \right) \mathbf{y}$.

        I am planning on implementing the both mentioned linear approaches in one MIMO equalization block. The block consumes N input streams and estimates the transmission vector. The block therefore produces a vector of N items for each input item. An optional milestone is the additional approach of symbol cancellation as an attempt of decision feedback detection.

- **Integrate MIMO capability to OFDM** The first task of this project as described above is the general implementation of the MIMO algorithms. But especially the STBCs don't stand for themselves. They require channel state information at the receiver which they usually gain over a channel estimation based on training sequences. The second goal for this project is therefore the embedding of the generally written spatial multiplexing blocks into one particular system: The OFDM transceiver of the GNU Radio module gr-digital.

The OFDM system provides frequency flat sub-channels. To preserve this quality, the MIMO algorithms must be applied to each parallel sub-carrier stream independently. To avoid the use of one separate MIMO encoder for each sub-carrier data stream, one vector-wise working MIMO encoder is placed before the OFDM modulator [6]. Figure 1 shows the structure of the OFDM transmitter after the insertion of the MIMO encoder block. Green blocks are already existing in the GNU Radio module gr-digital. The only structural changes are the replacement of the preamble insertion block with the MIMO encoder and the multiple OFDM modulators instead of only one. For a detailed description of the current OFDM structure in GNU Radio see [7]. Note that some of the shown blocks are working vector wise. The creation of the preamble, both for OFDM synchronization and MIMO detection, is included in the MIMO encoding block, which is a hierarchical python block that includes MIMO encoding and preamble appending.

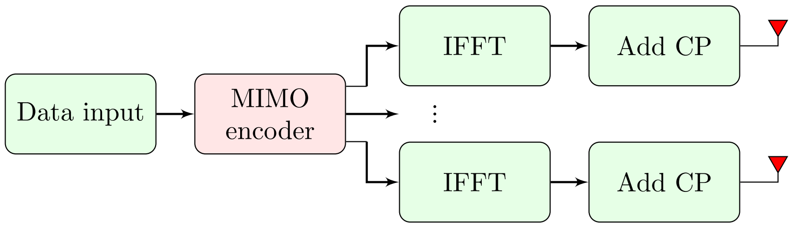

Figure 1: OFDM TX structure after MIMO inclusion

The structure of the MIMO-OFDM receiver is shown in figure 2. For the synchronization in time and frequency, the already implemented Schmidl & Cox algorithm can be applied on a superposition of the received signals. The OFDM demodulation is subsequently applied before the signals enter the MIMO demodulator. The OFDM demodulation block includes the channel estimation and the MIMO decoding algorithm.

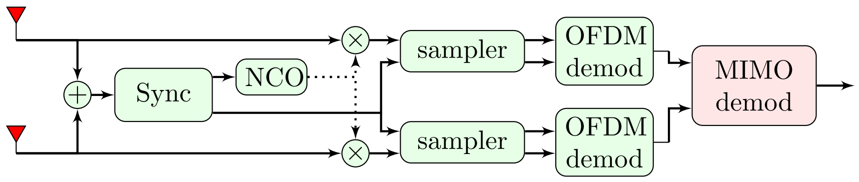

Figure 2: OFDM RX structure after MIMO inclusion

- **Quality assurance tests**: Well tested code is a very important and necessary quality proof. I am going to write a qa test for every block I introduce to the module. Matching encoding and decoding blocks are additionally tested in a loopback, using the channel model block as a connection between encoder and decoder.
- **Documentation**: A detailed documentation is an essential requirement especially for a project that is likely to be merged in the GNU Radio core module gr-digital. Gr-digital is documented with Doxygen [9], which I am planning to continue by adding extensive documentation to every written block I write. I am also going to make the code itself readable and comprehensive by commenting in-line.

I think about the proposed MIMO feature not just as a functioning tool but as well as an instructive and attractive feature for GNU Radio beginners to understand MIMO. I am therefore planning to additionally write a little in-tree top-level documentation in the style of [10].

4 Timeline

I am a graduate student at Karlsruhe Institute of Technology (KIT), Germany. I wrote my Bachelor thesis in fall 2017 and I am going to start the Master's program for electronics and information technology with a focus on communication technologies this spring. Currently I am working as an assistant researcher for the Communications Engineering Lab (CEL) in Karslruhe. I have got access to SDRs at CEL. Especially the USRP B210 devices which I am planning to use for over-the-air tests are sufficiently available. The whole project is going to be open-source available on GitHub, included the GPLv3 licensed code and documentation. After the completion of GSoC, the produced code could be merged into the GNU Radio repository with the latest upcoming GNU Radio release. My weekly progress is going to be published in an internet blog in order to keep my work transparent.

According to the timeline of GSoC 2018 there are 12 weeks of coding period including three evaluation periods. I scheduled the deliverables into 1- to 2-week periods, planning to work full-time (40 hours a week) from Monday to Friday and using the weekends as additional time buffers. The timeline is shown in table 1. I will not have any exams this summer and I am planning to skip my job at CEL for the time during GSoC. Therefore I can focus perfectly on GSoC.

Table 1: Timeline GSoC 2018

April 23 - May 13	**Community Bonding** - coordinate the project with the mentor. Research about MIMO algorithms and implementations. Investigate the setup of gr-digital's OFDM implementation and plan details for the integration of the MIMO feature. Discuss questions with the mentors and the GNU Radio community.
May 14 - May 20	Coding, 1 week - implement diversity combining block in C++ and QA tests in Python and write documentation.
May 21 - May 27	Coding, 1 week - implement MIMO channel model block and QA test in Python.
May 28 - June 10	Coding, 2 weeks - develop a MIMO training sequence and implement a MIMO channel estimation..
June 11 - June 13	**Phase 1 evaluation**, 3 days - prepare results for evaluation including a wrap up of the written blocks and QA tests.
14 June - July 8	Coding, 3.5 weeks - implement STBC encoder and decoder in C++ and QA tests in Python and write documentation.
July 9 - July 11	**Phase 2 evaluation**, 3 days - prepare results for evaluation.
July 12 - July 29	Coding, 2.5 weeks - integrate the MIMO capability to OFDM.
July 30 - Aug 5	Coding, 1 week - final bug fixes and wrap up.
Aug 6 - Aug 14	**Final evaluation** - 1 week, submission of the final code and evaluation.

I have read the rules of conduct for GSoC of GNU Radio and acknowledge the three strikes rule. Therefore I am going to intensively communicate with my mentor and keep the work transparent and my working progress up to date for everyone.

5 About myself

I firstly came in touch with programming in 2013, when I collected geodata from roads and trails for hiking and biking with a gps device and the geographic information system GeoMedia working for the department of environment of Leinfelden-Echterdingen. In order to grant the outdoor community public access of the data, I managed to read the geodata together with metadata like road conditions out of geoMedia. I additionally implemented a front end that displays the tracks on a georeferenced aerial photograph and offers the user the possibility to create a route by selecting certain track sections and automatically generating a gpx file which can be used to navigate through the route with a gps device [12]. I used the open-source environment Processing (based on java) in order to easily deal with graphical issues.

Since 2014 I have been studying electronics and information technology at KIT in Karlsruhe. I have improved my coding skills attending lectures in information technology (based on C/C++) and successfully completed a practical training. The objective was to implement the hardware abstraction layer for the control software of a segway called TivSeg. This included the control of a microcontroller, excluding the feedback control algorithm for the control of the segway. The code was written in C/C++ containing hardware-near programming.

After visiting several lectures about radio communication I got a job at CEL in 2016 in order to get more in touch with practical telecommunication engineering. I got familiar with GNU Radio and improved my coding skills in C++ and Python.

During the summer of 2017, I successfully participated in GSoC with GNU Radio, building a DAB Transceiver application [11]. I could complete all my proposed milestones, including the realization of a standard compliant DAB/DAB+ transmitter and receiver together with a graphical user interface, making the transceiver a full fleshed application. The project included a lot of GNU Radio related developing, especially the building of signal processing blocks in C/C++ but also the implementation of the graphical user interface in python. In September 2017, I could proudly present the results of my project as a student presenter at the GNU Radio Conference 2017 in San Diego. The conference gave me the great opportunity of further connecting with the GNU Radio community and actually meet a lot of developers in person.
After GSoC 2017, I have continued working on the DAB transceiver module, improving the existing code and introducing new features. As some examples, I added RTL-SDR support as a possible signal source and currently I am implementing dynamic label and Multimedia Object Transfer (MOT) support.

I wrote my Bachelor thesis in fall 2017 at the CEL, continuing the work with DAB on a more theoretical level. I rewrote the existing OFDM physical layer of the DAB/DAB+ transceiver and developed a more efficient and robust synchronization method. The OFDM system was then evaluated on its performance in different propagation channels regarding bit and packet error rates. The extensive research and experience with OFDM gives me a very solid basis to approach the MIMO-OFDM system that I am proposing.

6 Conclusion

A MIMO capability for GNU Radio would be a great addition to the enormous amount of basic signal processing blocks in the GNU Radio tree. It would be a solid foundation for upcoming MIMO projects than could build on the existing code base. The inclusion of the MIMO feature to the existing OFDM transceiver is finally a great example for a complete MIMO setup.
Through my projects I have come to realize the importance of open-source and how everyone can achieve more by sharing their ideas and thoughts. Therefore I would like to take this opportunity and participate in a GNU Radio project to contribute to the open-source idea.

Finally I want to emphasize: Cyberspectrum is the best spectrum.

References

[1] Andrea Goldsmith. 2005. Wireless Communications. Cambridge University Press, New York, NY, USA.

[2] V. Tarokh, N. Seshadri and A. R. Calderbank, "Space-time codes for high data rate wireless communication: performance criterion and code construction," in IEEE Transactions on Information Theory, vol. 44, no. 2, pp. 744-765, Mar 1998.

[3] P. W. Wolniansky, G. J. Foschini, G. D. Golden and R. A. Valenzuela, "V-BLAST: an architecture for realizing very high data rates over the rich-scattering wireless channel," 1998 URSI International Symposium on Signals, Systems, and Electronics. Conference Proceedings (Cat. No.98EX167), Pisa, 1998, pp. 295-300.

[4] Y. Jiang, M. K. Varanasi and J. Li, "Performance Analysis of ZF and MMSE Equalizers for MIMO Systems: An In-Depth Study of the High SNR Regime," in IEEE Transactions on Information Theory, vol. 57, no. 4, pp. 2008-2026, April 2011.

[5] Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall. 2010. 802.11 with multiple antennas for dummies. SIGCOMM Comput.

[6] G. L. Stuber, J. R. Barry, S. W. McLaughlin, Ye Li, M. A. Ingram and T. G. Pratt, "Broadband MIMO-OFDM wireless communications," in Proceedings of the IEEE, vol. 92, no. 2, pp. 271-294, Feb. 2004.

[7] https://github.com/gnuradio/gnuradio/blob/master/gr-digital/examples/ofdm/tx_ofdm.grc Visited March 24, 2018

[8] IEEE Std. 802.11n-2009: Enhancements for higher throughput. http://www.ieee802.org, 2009.

[9] https://gnuradio.org/doc/doxygen/namespacegr_1_1digital.html Visited March 23, 2018

[10] https://gnuradio.org/doc/doxygen/page_components.html Visited March 23, 2018

[11] https://dabtransceiver.wordpress.com/ Visited March 22, 2018

[12] https://github.com/MoritzLucaSchmid/gpx-outdoor_planning

[13] https://github.com/MoritzLucaSchmid

[14] GNU Radio Documentation http://gnuradio.org/doc/doxygen/modules.html Visited