A GNU Radio-based Full Duplex Radio System

Adam Parower
The Aerospace Corporation

September 13, 2017

Agenda

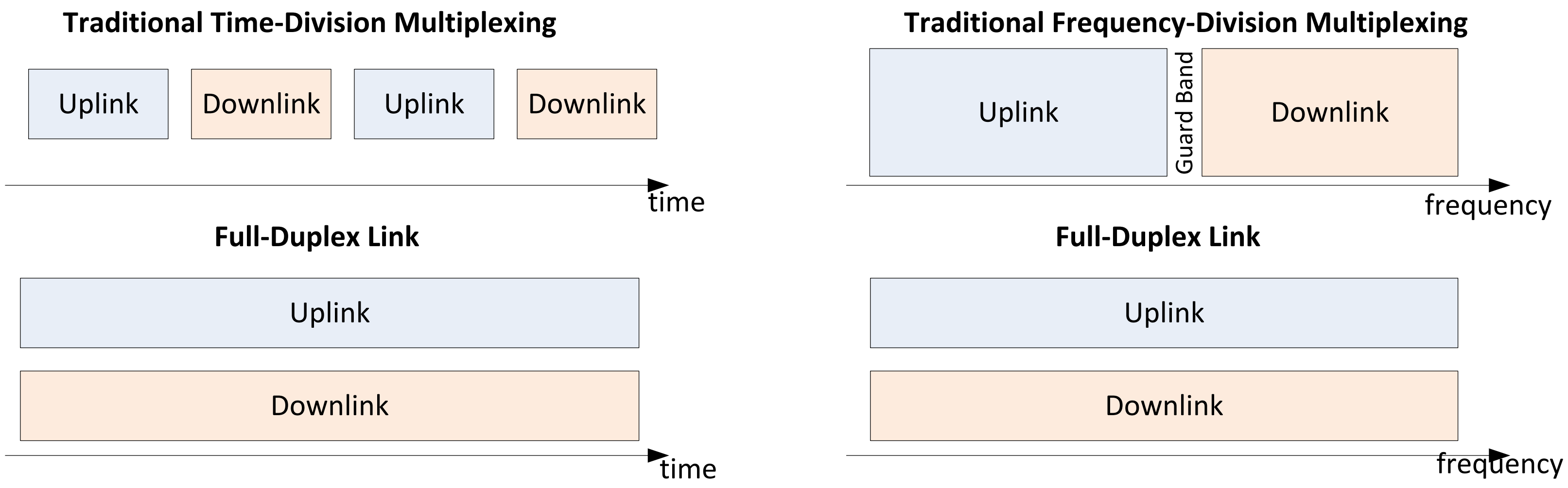
- Theory: Full-Duplex
  - *What is Full Duplex?*
  - *The Problem*
  - *The Solution*
- Theory: Digital Cancellation
  - *Black Box Model*
  - *Modeling Nonlinearity*
  - *Least-squares Problem*
  - *Algorithms for Solving*
- Custom GNU Radio Blocks
  - *Implementation*
  - *Cancellation Performance*
  - *Throughput Results*
- Using USRP Products
  - *Timing Synchronization*
  - *Built-in vs. External Mixers*
- Conclusions

What is Full-Duplex Communications?

… it's not the 'usual definition'

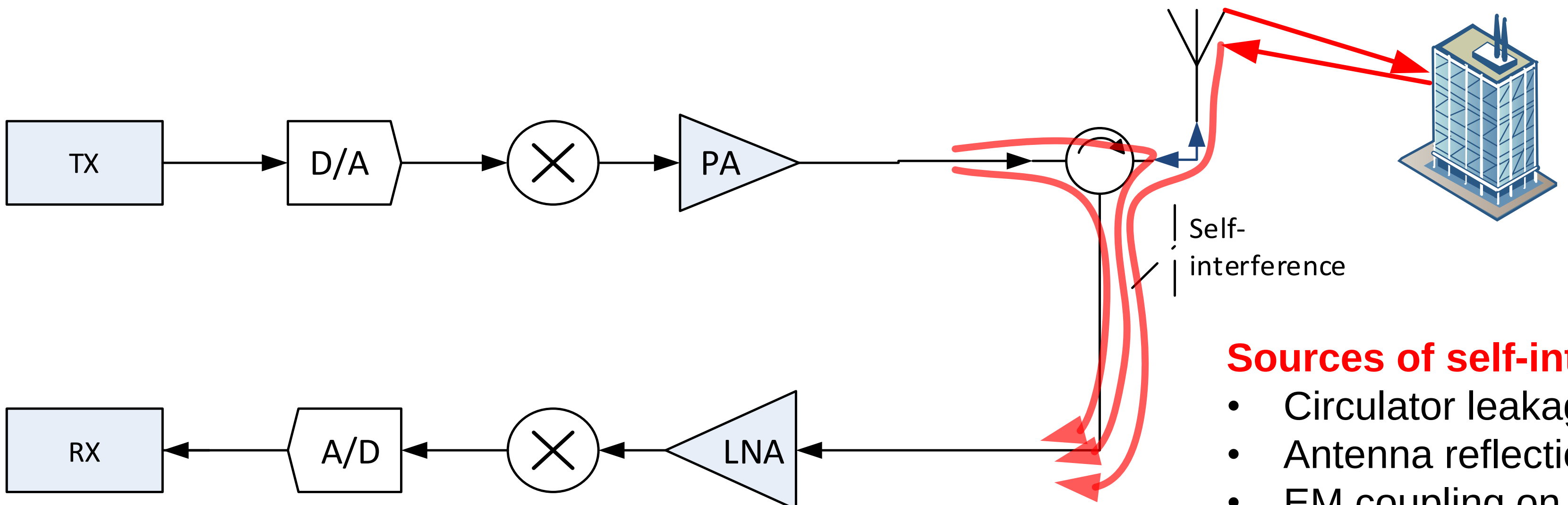
- Transmit and receive on the same antenna on the same frequency at the same time
  - *Potentially double (or more) throughput in the same spectrum*
  - *Considered impossible for typical communications links*
  - *Multiple classical textbooks explicitly state it can't be done*

- Can replace both time- and frequency-multiplexed links



The Problem

- The problem is self-interference
  - *Transmit power swamps the receive power, making it very difficult to detect the desired signal*
  - *Power difference $10\log_{10}(P_{Tx}/P_{Rx})$ depends on the distance between Tx and Rx*



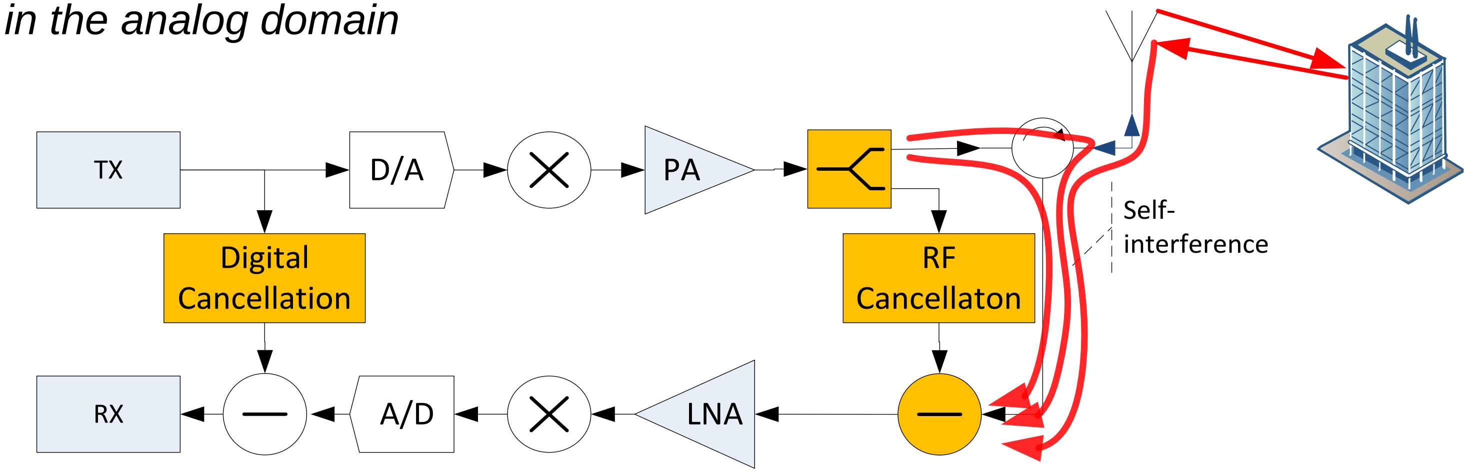
Sources of self-interference

- Circulator leakage
- Antenna reflection
- EM coupling on cables
- Reflections off objects

The Solution

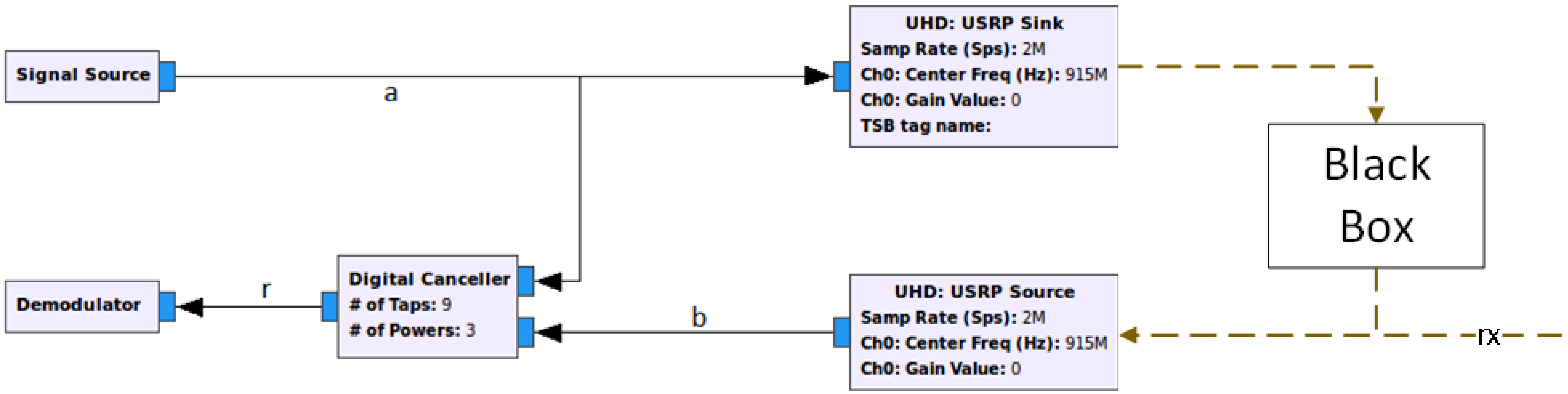
Very active area of research in the past several years

- The transmitted signal is known at the receiver
  - *To some degree, need to account for delay and distortion(s)*
- Subtract the transmitted signal from the received signal to remove self-interference
- The devil is in the implementation
  - *Transmitted signal power may be > 100 dB above received signal*
  - *Need very high linearity, very accurate matching and model of distortions*
- Two major approaches:
  1. *Cancel at baseband in the digital domain*
  2. *Cancel at RF in the analog domain*



All reported full-duplex systems use both analog and digital cancellation

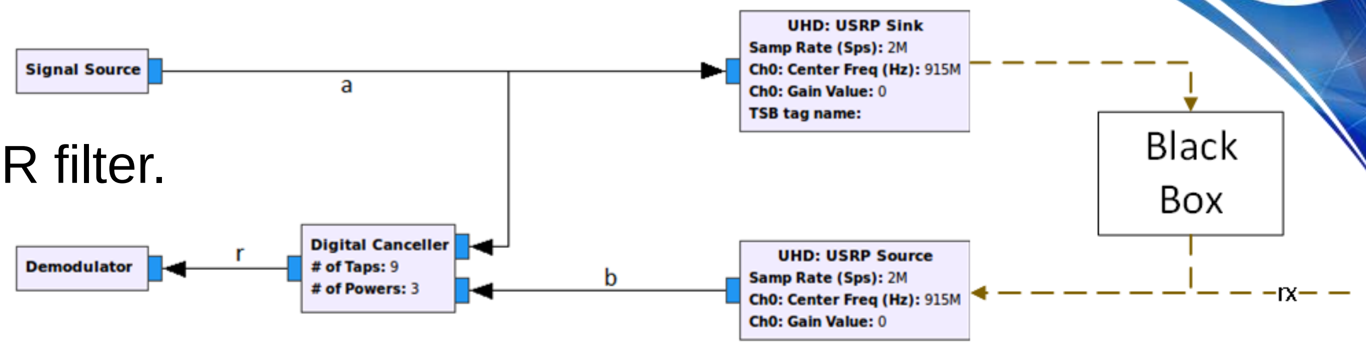
Digital Cancellation



- Find a model for the black box to minimize r.
  - *I.e. find* $\mathcal{F}$ *to minimize* $|r| = |b - \mathcal{F}(a)|$
- The model must account for:
  - *Gain*
  - *Phase shift*
  - *Time delay*
  - *Nonlinear distortion*
  - *Multipath*

Modeling the Black Box

Linear Model



- We model the black box as a non-causal FIR filter.
- Let:
  - a_i = transmitted signal, sample i
  - b_i = received signal, sample i
  - x_k = FIR filter coefficient k
- Our goal is to find the filter coefficients that produce outputs (r_i) most similar to the actual received signal.
- Example for a 5-tap filter, for the 2nd received sample:

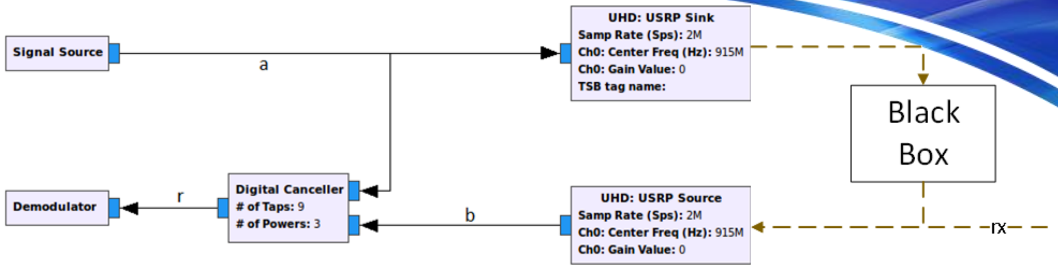
$$b_2 = x_0 a_0 + x_1 a_1 + x_2 a_2 + x_3 a_3 + x_4 a_4$$

- Or, in vector form:

$$[a_0\ a_1\ a_2\ a_3\ a_4]\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = b_2$$

Modeling the Black Box

Nonlinear Distortion



- We model the black box as a ***summation*** of FIR filters, each operating on a different (odd) power of the transmitted signal.
- Let:
  - a_i = transmitted signal, sample i
  - b_i = received signal, sample i
  - x_{jk} = FIR filter coefficient k for power j
- Example for 3 powers (1st, 3rd, and 5th), each with 5 taps:

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_0^3 & a_1^3 & a_2^3 & a_3^3 & a_4^3 & a_0^5 & a_1^5 & a_2^5 & a_3^5 & a_4^5 \end{bmatrix} \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{30} \\ x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{50} \\ x_{51} \\ x_{52} \\ x_{53} \\ x_{54} \end{bmatrix} = b_2$$

Modeling the Black Box

Matrix Representation

- If we wish to work with multiple samples of the received signal at a time, we can express the model in matrix form as follows:

$$
\begin{bmatrix}
a_0 & a_1 & a_2 & a_3 & a_4 & a_0^3 & a_1^3 & a_2^3 & a_3^3 & a_4^3 & a_0^5 & a_1^5 & a_2^5 & a_3^5 & a_4^5 \\
a_1 & a_2 & a_3 & a_4 & a_5 & a_1^3 & a_2^3 & a_3^3 & a_4^3 & a_5^3 & a_1^5 & a_2^5 & a_3^5 & a_4^5 & a_5^5 \\
a_2 & a_3 & a_4 & a_5 & a_6 & a_2^3 & a_3^3 & a_4^3 & a_5^3 & a_6^3 & a_2^5 & a_3^5 & a_4^5 & a_5^5 & a_6^5 \\
a_3 & a_4 & a_5 & a_6 & a_7 & a_3^3 & a_4^3 & a_5^3 & a_6^3 & a_7^3 & a_3^5 & a_4^5 & a_5^5 & a_6^5 & a_7^5 \\
a_4 & a_5 & a_6 & a_7 & a_8 & a_4^3 & a_5^3 & a_6^3 & a_7^3 & a_8^3 & a_4^5 & a_5^5 & a_6^5 & a_7^5 & a_8^5 \\
a_5 & a_6 & a_7 & a_8 & a_9 & a_5^3 & a_6^3 & a_7^3 & a_8^3 & a_9^3 & a_5^5 & a_6^5 & a_7^5 & a_8^5 & a_9^5 \\
 & & & & & & & \vdots & & & & & & &
\end{bmatrix}
\begin{bmatrix}
x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{30} \\ x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{50} \\ x_{51} \\ x_{52} \\ x_{53} \\ x_{54}
\end{bmatrix}
=
\begin{bmatrix}
b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ \vdots
\end{bmatrix}
$$

$$A \quad \vec{x} \quad = \quad \vec{b}$$

We are looking for the value of $\vec{x}$ ***that minimizes*** $\|A\vec{x} - \vec{b}\|$

Algorithms

For Solving $A\vec{x} = \vec{b}$

Block-based algorithms operate on the entire matrix A, effectively computing $\vec{x} = A^{+}\vec{b}$.

- Singular value decomposition (SVD)
- QR decomposition
- Cholesky decomposition
- Conjugate gradient method

Adaptive algorithms operate on one row of A at a time, adjusting the value of $\vec{x}$ each iteration.

- Least mean squares (LMS)
- Recursive least squares (RLS)

Algorithms

For Solving $A\vec{x} = \vec{b}$

Block-based algorithms operate on the entire matrix A, effectively computing $\vec{x} = A^{+}\vec{b}$.

- Singular value decomposition (SVD)
- QR decomposition
- Cholesky decomposition
- Conjugate gradient method

Adaptive algorithms operate on one row of A at a time, adjusting the value of $\vec{x}$ each iteration.

- Least mean squares (LMS)
- Recursive least squares (RLS)

We implemented a GNU Radio block for each algorithm in green.

Note

- These GNU Radio blocks are not just for full duplex!
- They can be used to solve any problem where it is desirable to cancel a known signal that may have undergone linear and/or nonlinear distortion.

Performance Optimization

- For high throughput performance, we leverage the Intel libraries:

  - ***IPP****: Intel Performance Primitives:*
    - Contains signal processing routines
    - Optimized using Streaming SIMD Extensions

  - ***MKL****: Math Kernel Library*
    - Contains optimized functions for vector and matrix math
    - API is compatible with BLAS and LAPACK functions

- We further increase throughput by dividing computationally-intensive work between multiple threads.

Implementation (Pseudocode)

SVD Block

```
int svd_canceller_cc_impl::general_work(
    int noutput_items, gr_vector_int &ninput_items,
    gr_vector_const_void_star &input_items, gr_vector_void_star &output_items)
{
    gr_complex* ref =  input_signals[0];    // ref = transmitted signal
    gr_complex* b   =  input_signals[1];    // b = received signal
    gr_complex* out = output_signals[0];

    construct_A_matrix(ref, A);
    cgelsd(A, b, x, ...);                   // solve Ax = b for x (using SVD)
    cgemv(A, x, b, out, ...);               // residual = b – Ax



    consume_each(block_size);
    return block_size;
}
```

Implementation (Pseudocode)

QR Decomposition Block

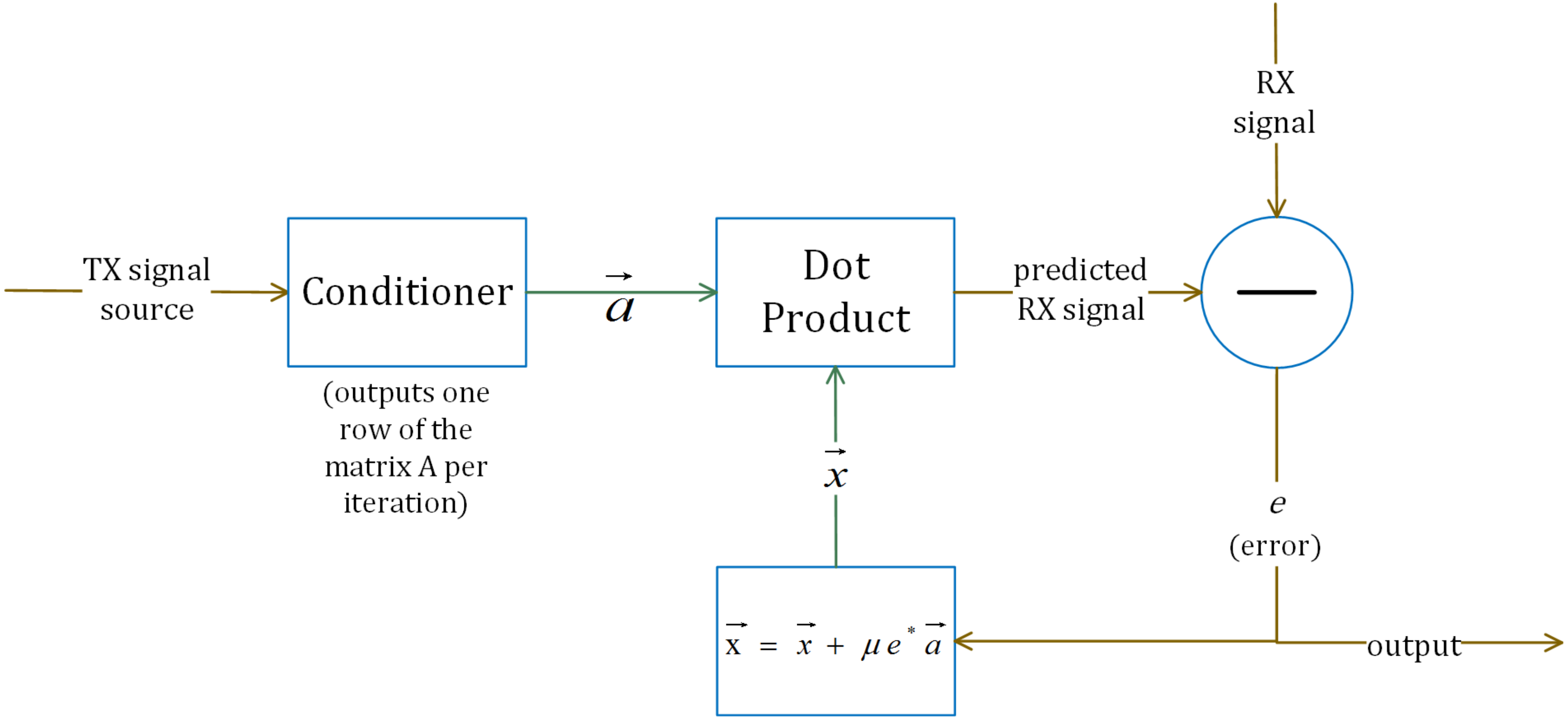
```
int qr_canceller_cc_impl::general_work(
    int noutput_items, gr_vector_int &ninput_items,
    gr_vector_const_void_star &input_items, gr_vector_void_star &output_items)
{
    gr_complex* ref =  input_signals[0];    // ref = transmitted signal
    gr_complex* b   =  input_signals[1];    // b = received signal
    gr_complex* out = output_signals[0];

    construct_A_matrix(ref, A);
    cgeqrf(A, temp, ...);                   // perform QR factorization on A
    cungqr(temp, Q, ...);                   // compute Q explicitly
    cgemv(Q, b, Qb, ...);                   // compute Q^T b
    cgemv(Q, Qb, b, out, ...);              // residual = b − Q Q^T b

    consume_each(block_size);
    return block_size;
}
```

Implementation (Block Diagram)

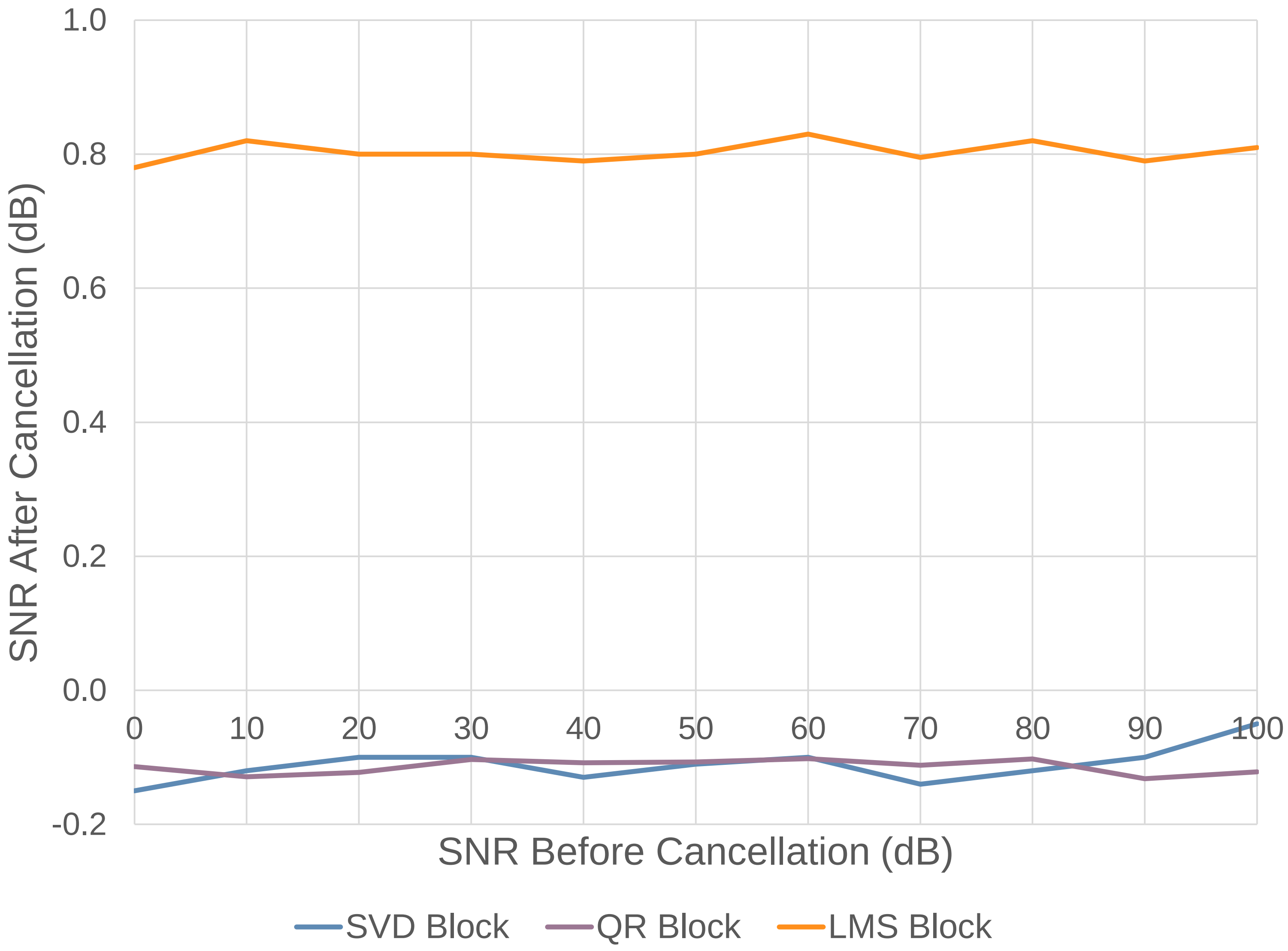
LMS Block



- We optimize this block for throughput by:
  - *Using multiple threads, each of which processes a chunk of the input*
  - *Calculating updates to the coefficients $\vec{x}$ every N samples (instead of every sample)*
  - *Averaging the threads' values of $\vec{x}$ when they synchronize*

Results and Performance

Cancellation

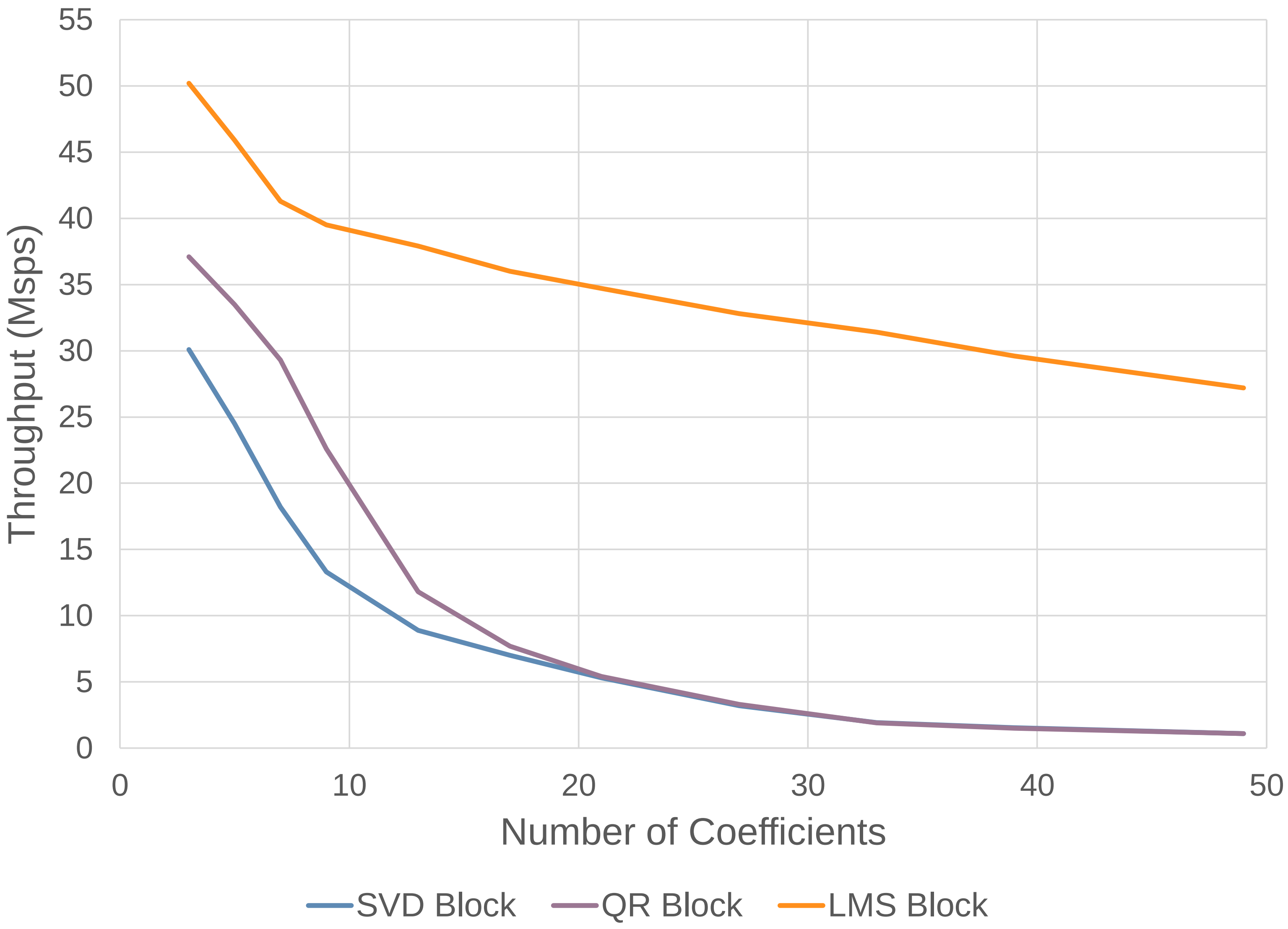


<u>Test conditions:</u>

- Digital simulation
- 13-tap FIR filter
- 1 power (linear only)
- SVD/QR block size: 512
- LMS step size: 0.01

Results and Performance

Throughput



<u>Test conditions:</u>

- Intel Xeon X5660 (2.80GHz)
- 8 threads

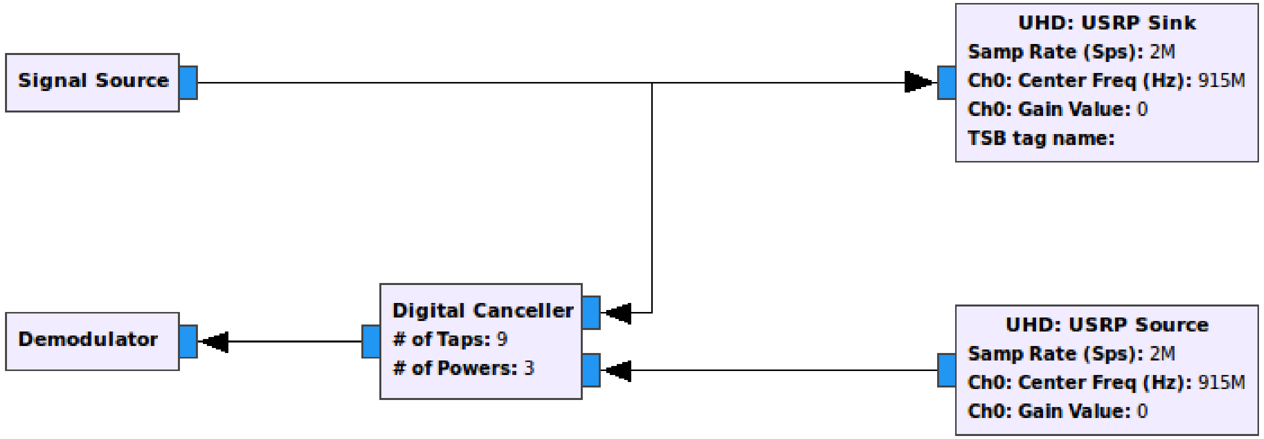
Using USRP Products

- If you decide to use USRPs in a full-duplex radio system, here are some tips ...

Source: https://www.ettus.com/content/images/prod_un210-kit_01_lg.jpg

Using USRP Products

Timing Synchronization

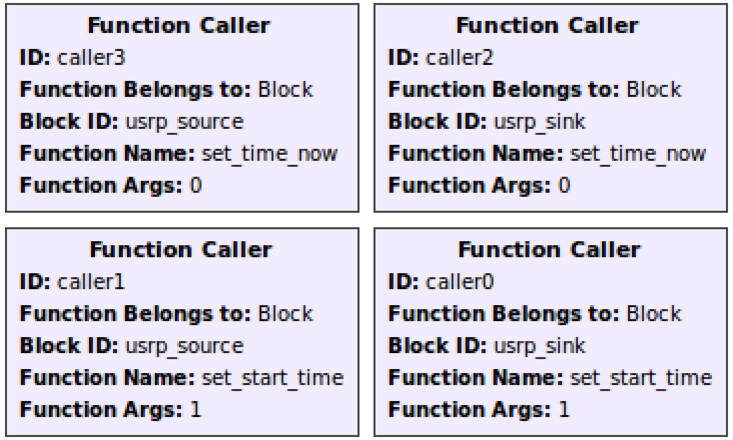


- For this flowgraph to work, the USRP sink and source blocks must start streaming at the same time.
- These blocks provide functions for timing synchronization:
  - *set_time_now()*
  - *set_time_next_pps()*
  - *set_start_time()*
- We can edit the GRC-generated Python code to call these functions ...
  - *But if we make a change to the flowgraph and regenerate, we have to do it again*
  - *There must be a better way ...*

Function Caller Block

- Wouldn't it be nice if there was a GRC block that allowed us to embed arbitrary function calls in the generated code, that execute before the flowgraph starts?

- We created one:



Generated Python code:

```
val = self.usrp_source.set_time_now(0)
...
val = self.usrp_sink.set_time_now(0)
...
val = self.usrp_source.set_start_time(1)
...
val = self.usrp_sink.set_start_time(1)
```

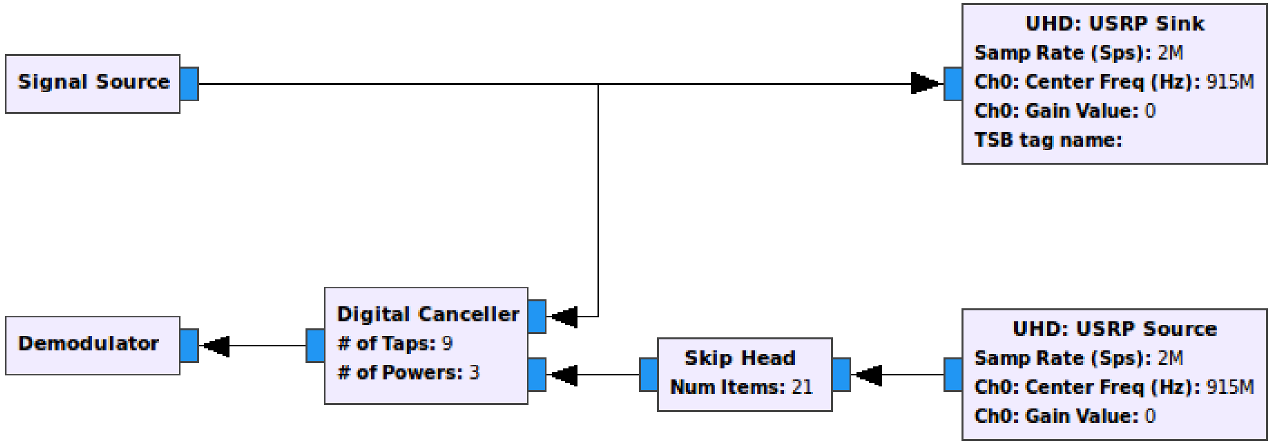
- Largely based on the built-in Function Probe block

Using USRP Products

Timing Synchronization

- Even after synchronizing the USRP source and sink, there is still a timing offset:
  - *Deterministic and repeatable*
  - *Appears to be sample-rate dependent*
  - *On the order of 10-50 samples*

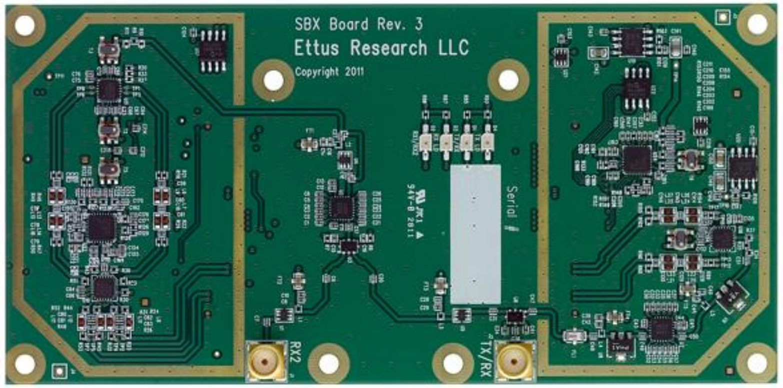
- This can be remedied with a skip head block:



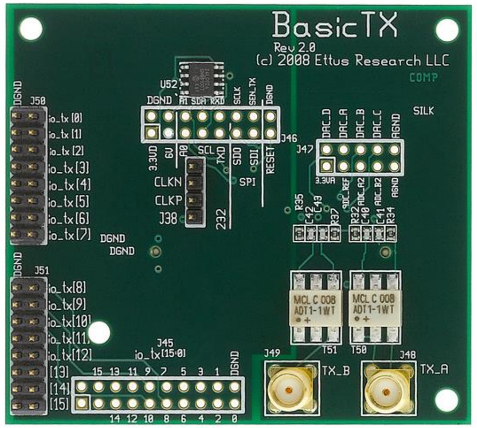
Using USRP Products

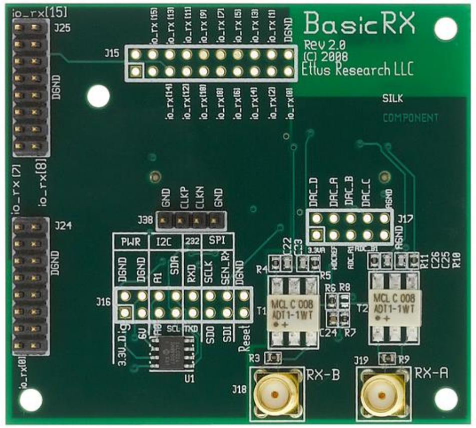
Built-in vs. External Mixers

- Most USRP daughterboards contain LO generators and mixers to convert between baseband/IF and RF.
- e.g. SBX, UBX



- Some daughterboards contain no LO or mixer but operate at baseband/IF.
- e.g. BasicTX, BasicRX





Source: https://www.ettus.com/product/category/Daughterboards

- These boards can be used with external mixers to produce RF.

Using USRP Products

Built-in vs. External Mixers

- For applications like full duplex where high SNR is essential, use external mixers for optimal performance:

Digital Cancellation (dB)		**Transmitter Setup**	
		BasicTX + External Up-converter	SBX
Receiver Setup	BasicRX + External Down-converter	54.4	44.6
	SBX	39.2	38.0

<u>Test conditions:</u>

- USRP X310
- Analog loopback

Summary and Conclusions

- It is feasible to implement a full-duplex radio system using GNU Radio.
- By using Intel libraries and multi-threading, we can support bandwidths in the tens of MHz.
- If parallelized, adaptive algorithms like LMS provide higher throughput with minimal cost to cancellation.
- Digital cancellation blocks can be applied to any scenario involving suppression of a known signal.

Questions?