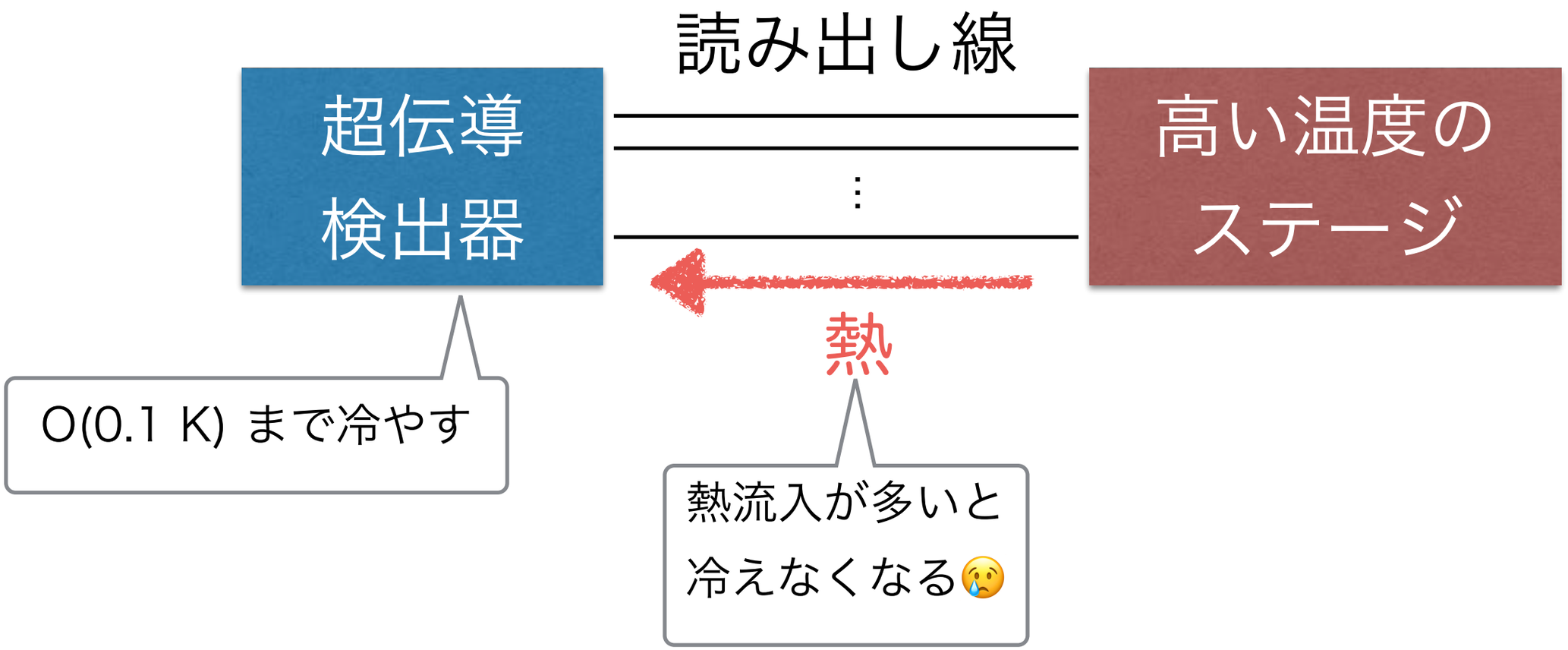
超伝導検出器の
マイクロ波読み出し技術開発

京都大学
鈴木惇也

計測システム研究会 2021

マイクロ波読み出しに
ついて

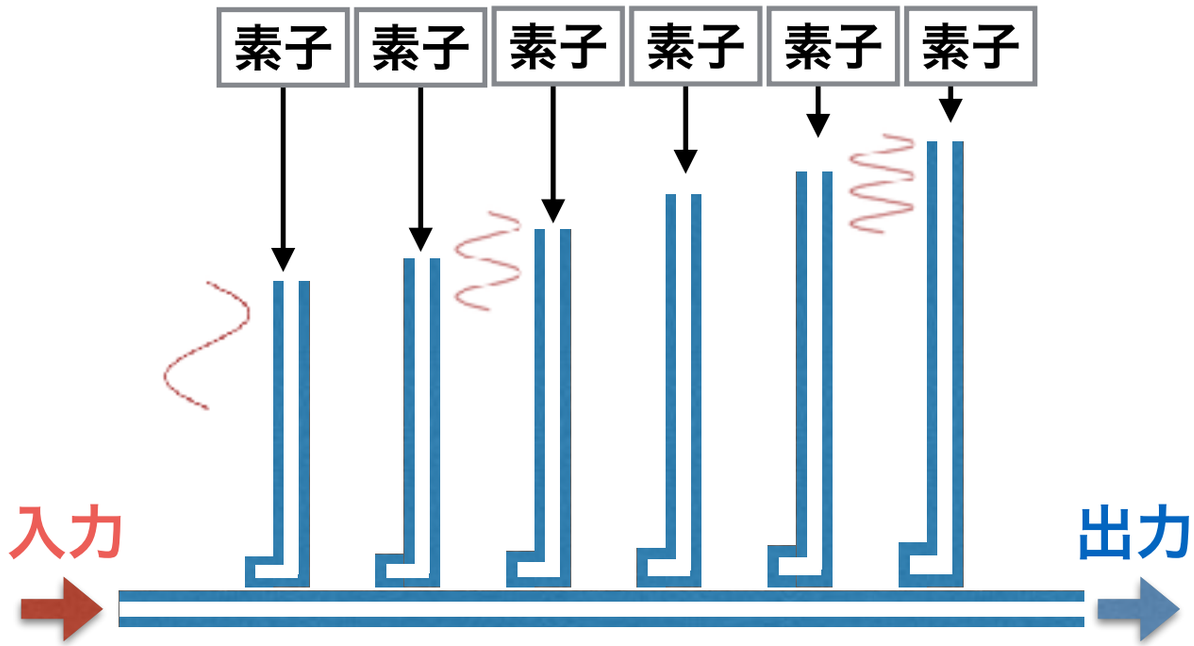
超伝導検出器と読み出し



配線の本数を減らしたい！

読み出しの多重化

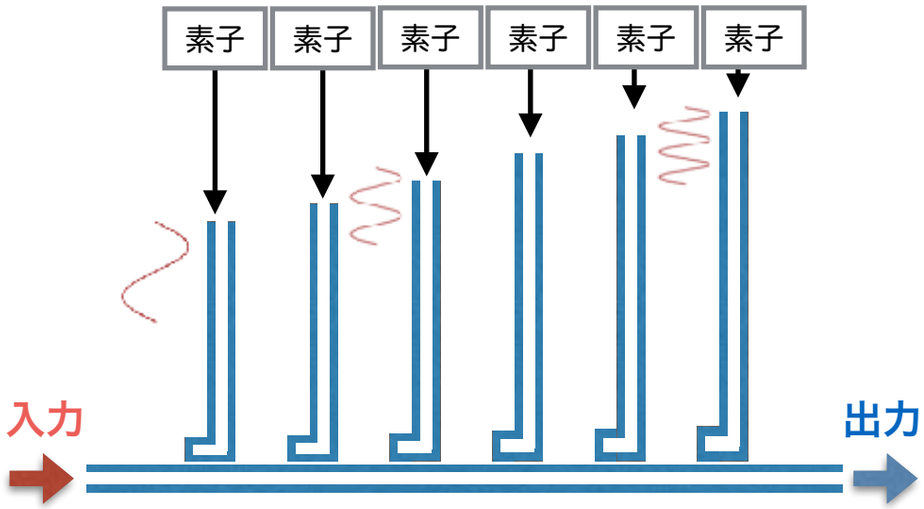
1本の線で複数の素子を読み出す　→　**多重化**

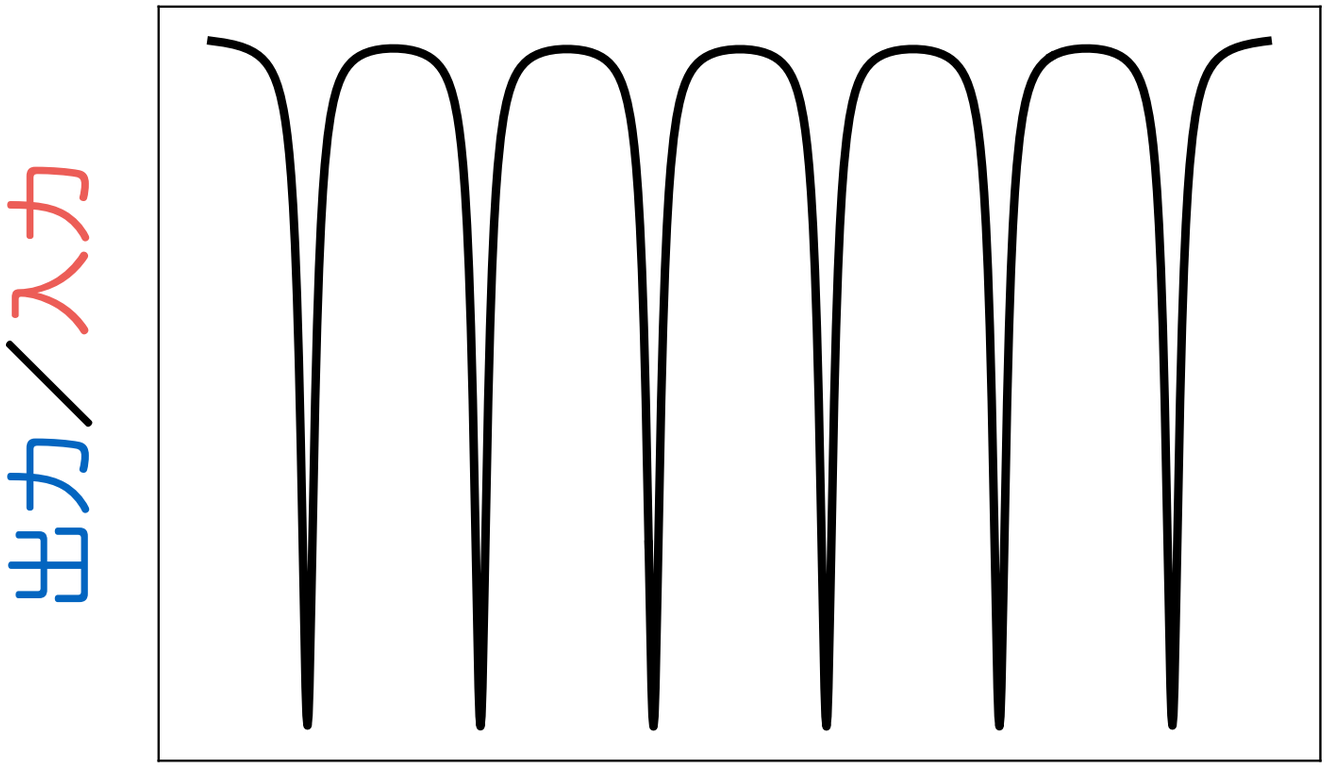


超伝導共振器を使って

周波数空間で多重化

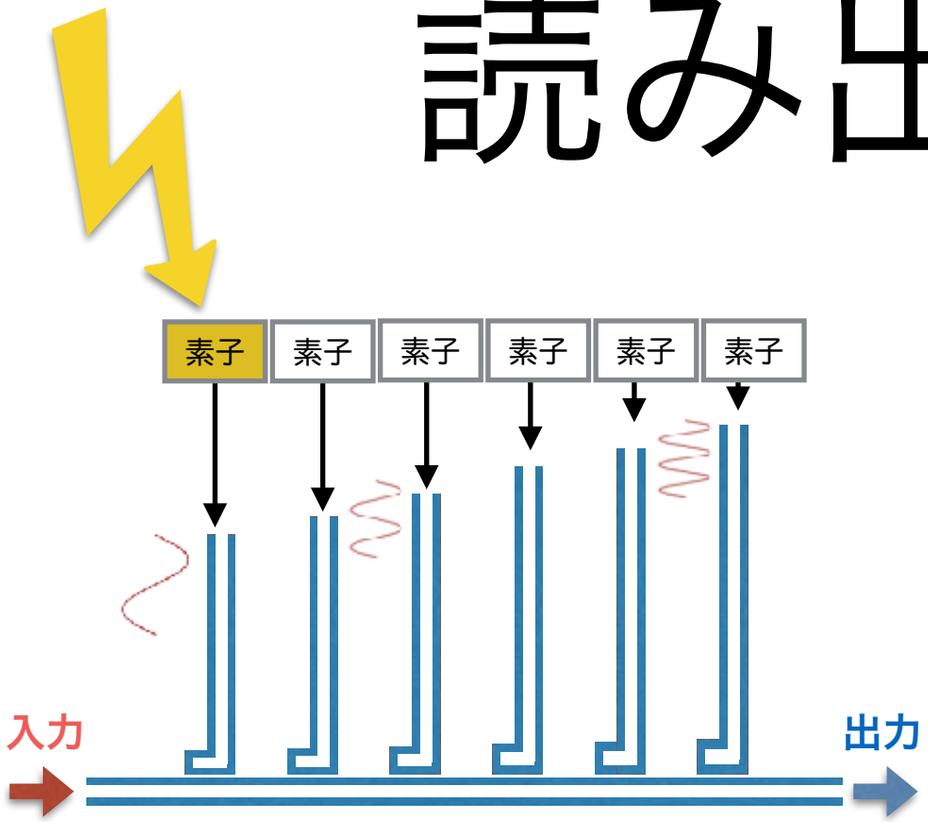
読み出しの多重化

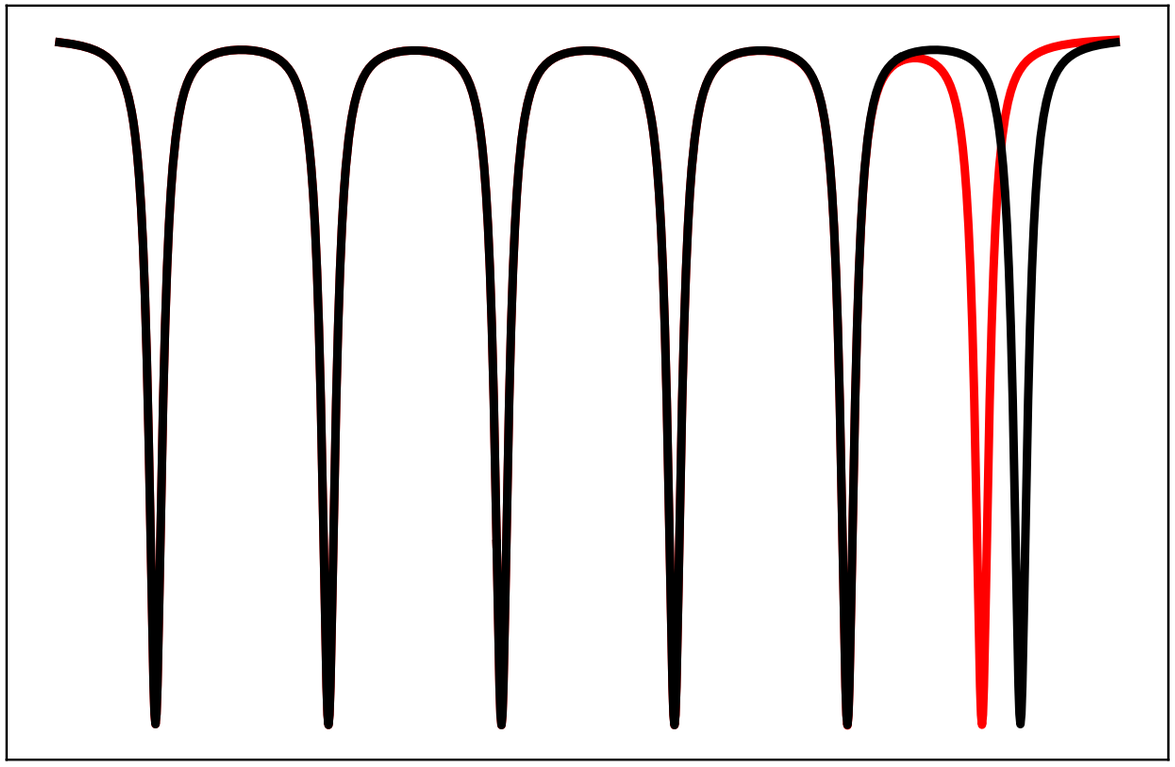




・ 共振周波数で、透過率（出力／入力）が低くなる

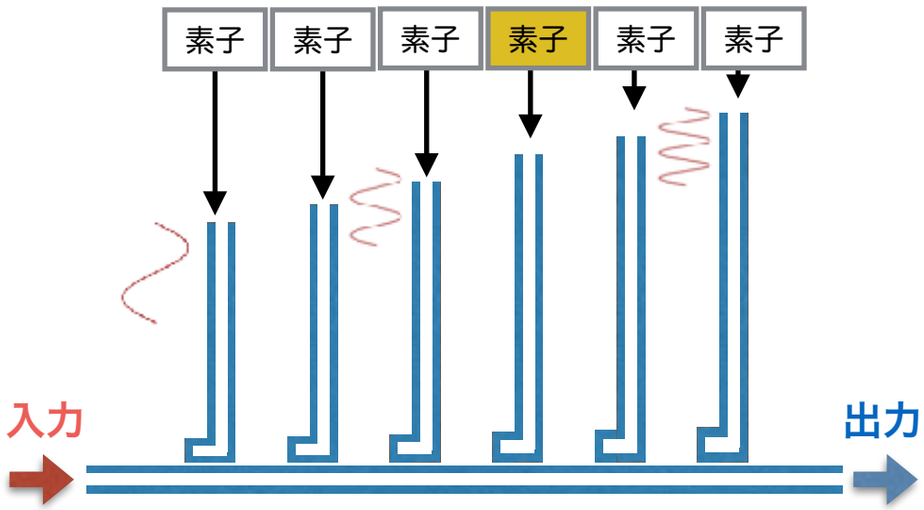
読み出しの多重化

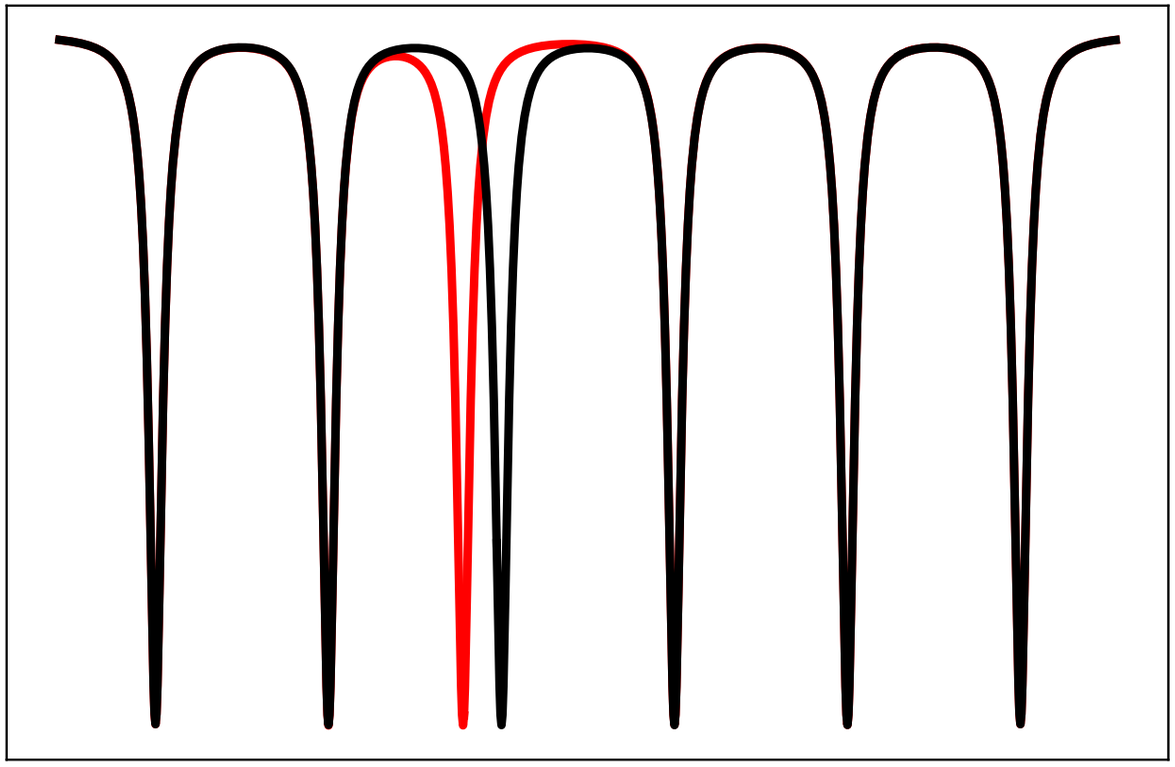




・エネルギー　→　共振周波数が変化

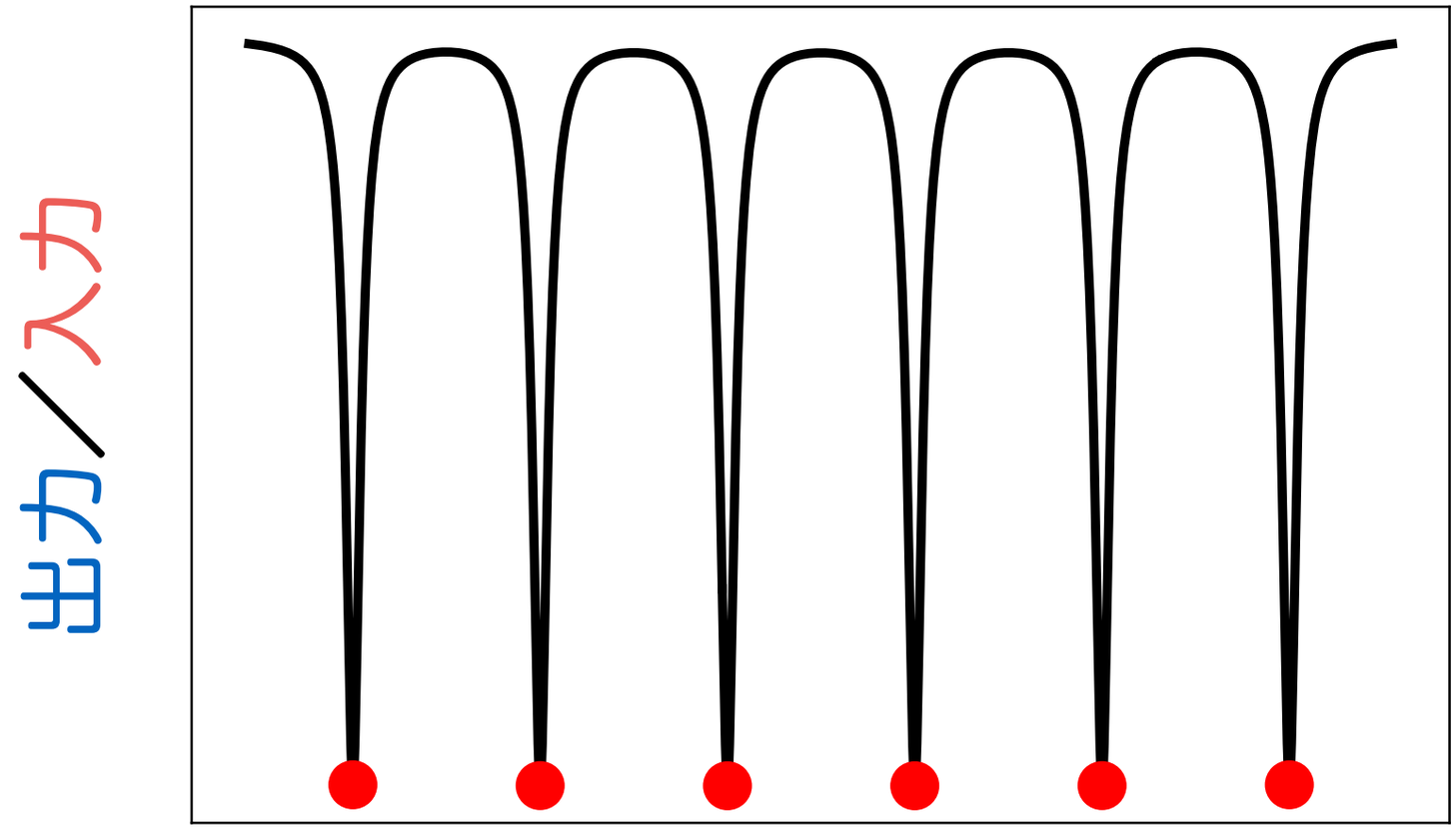
読み出しの多重化





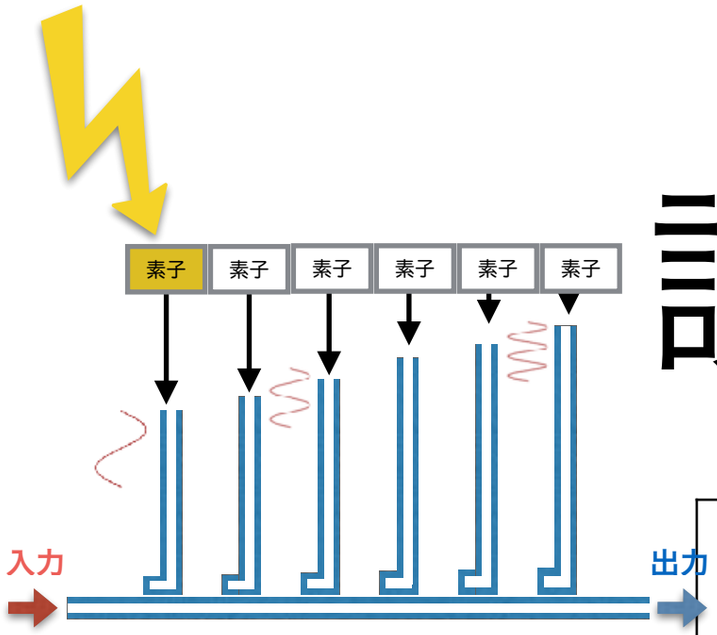
・エネルギー　→　共振周波数が変化

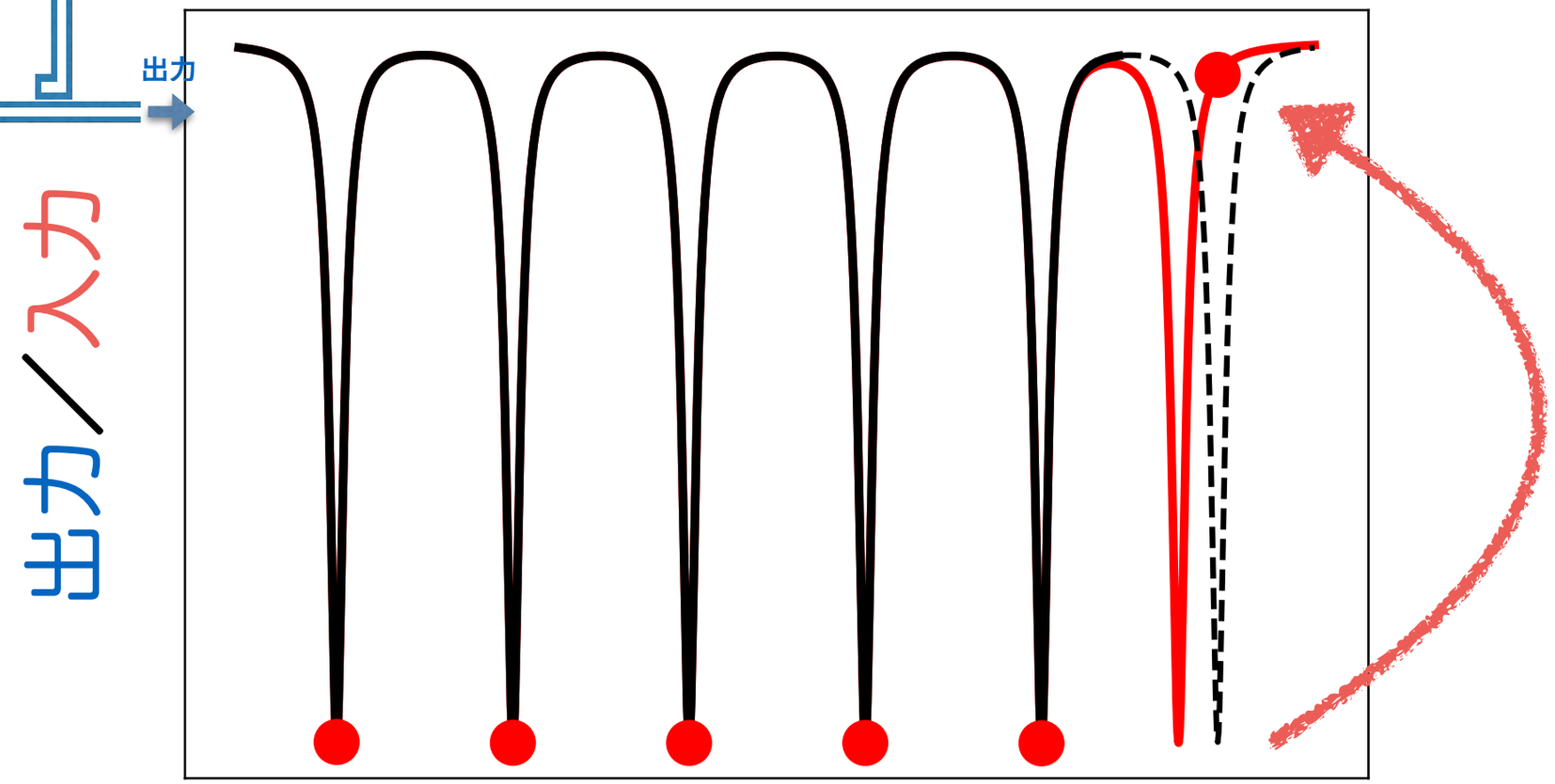
読み出しかた



・ それぞれの共振周波数で　出力／入力　をはかる

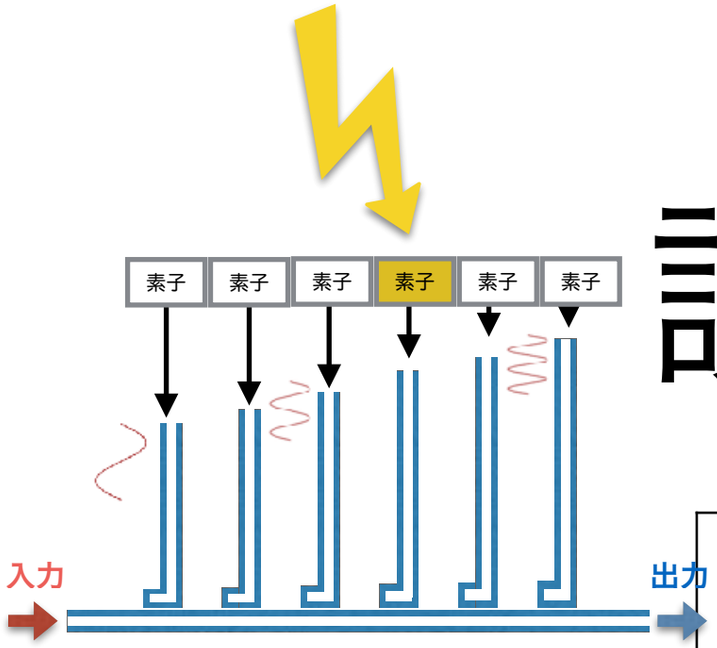
読み出しかた

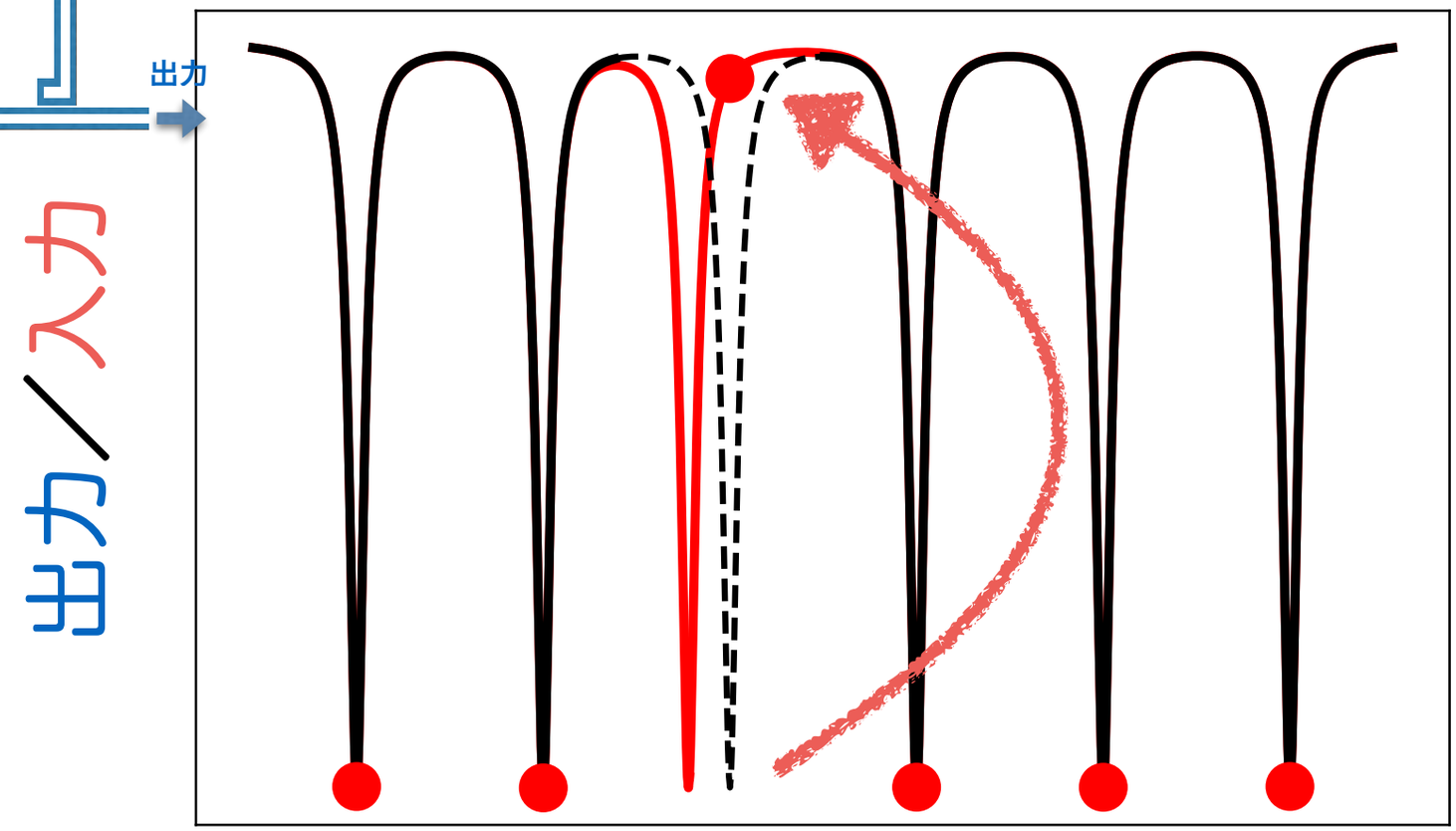




・出力／入力　が変化

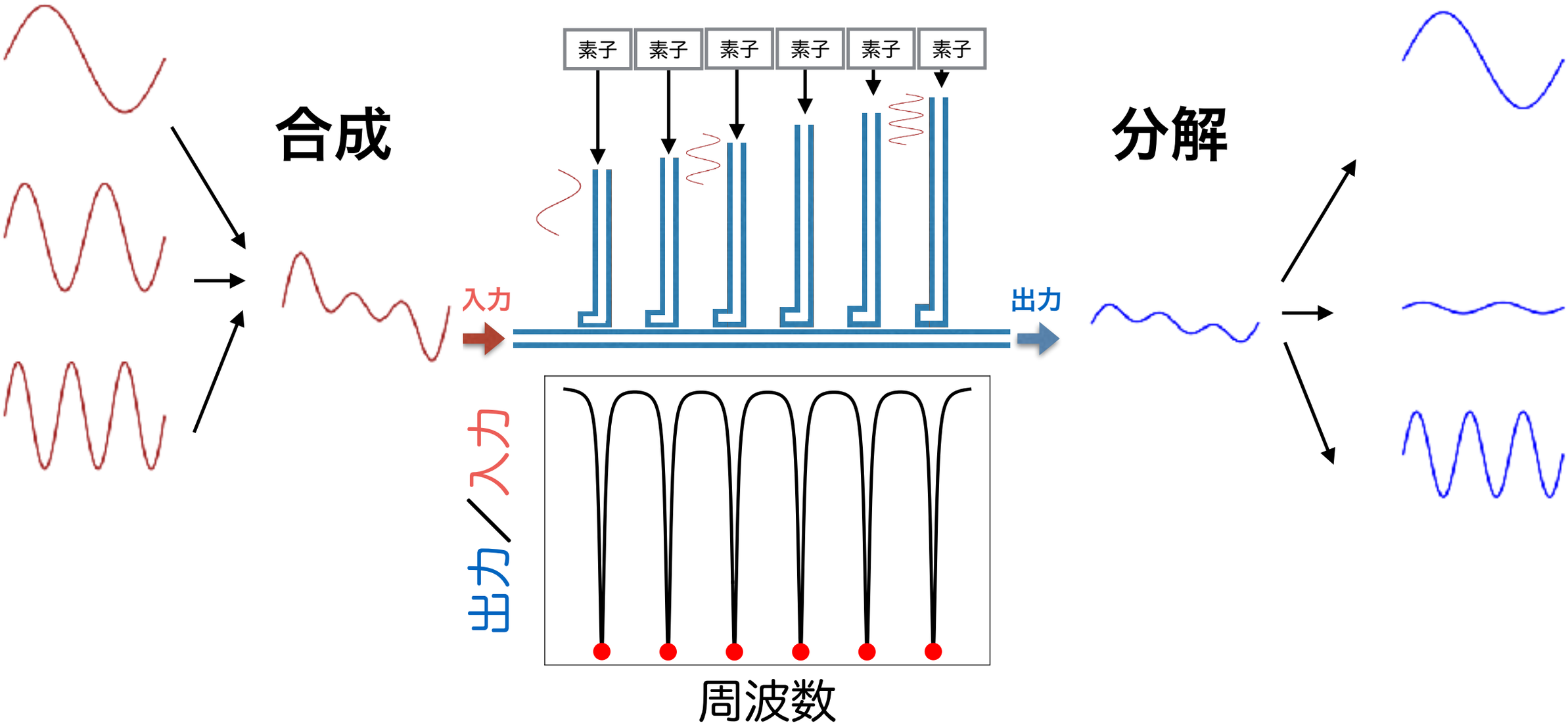
読み出しかた





・出力／入力　が変化

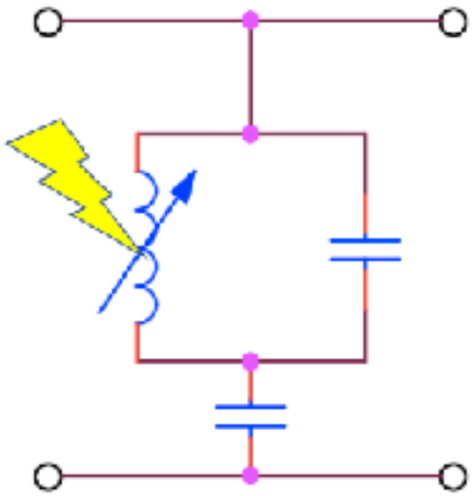
読み出しかた



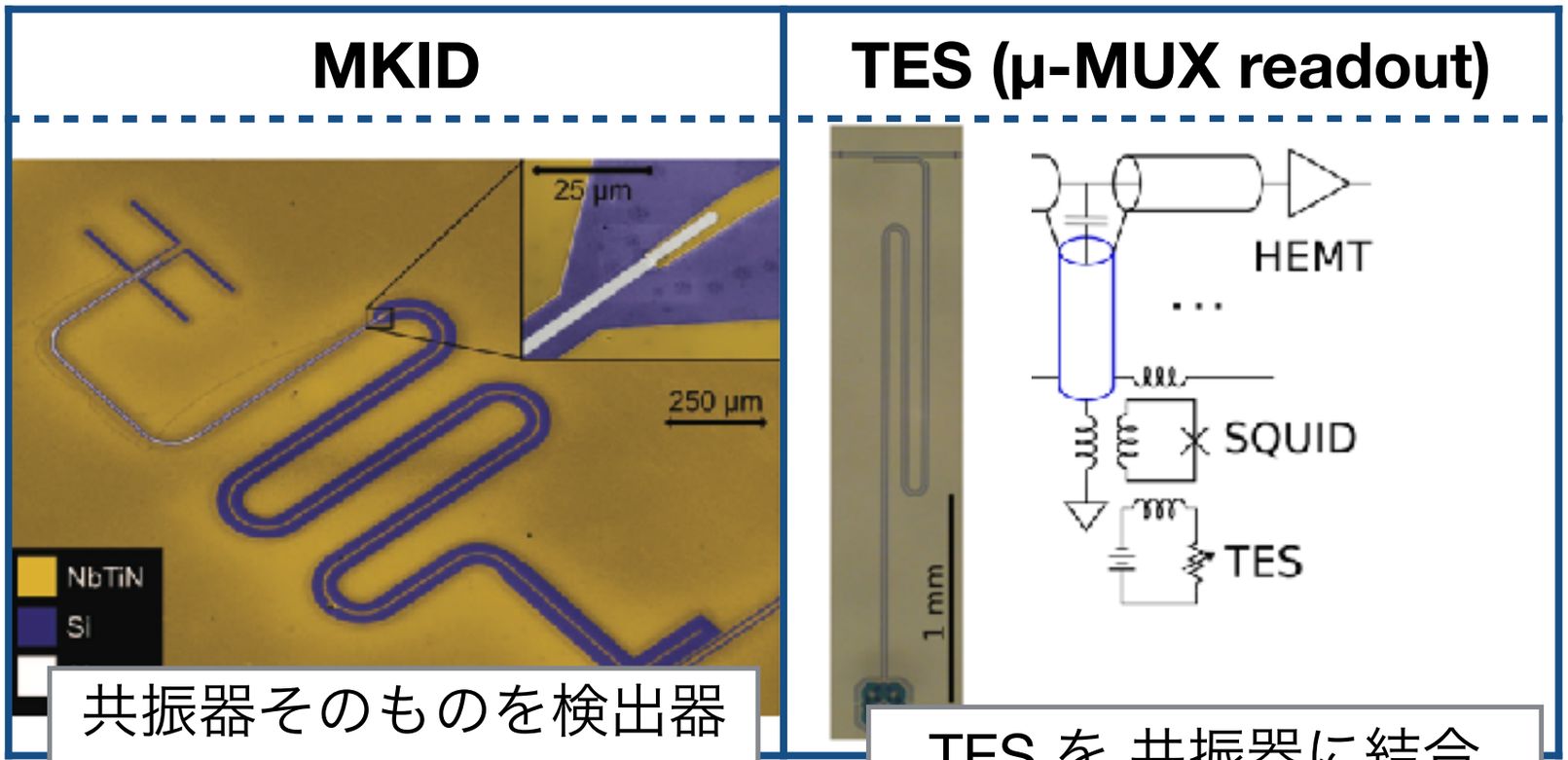
読み出し…合成と分解

→　**<u>FPGA で実装！</u>**

超伝導共振器

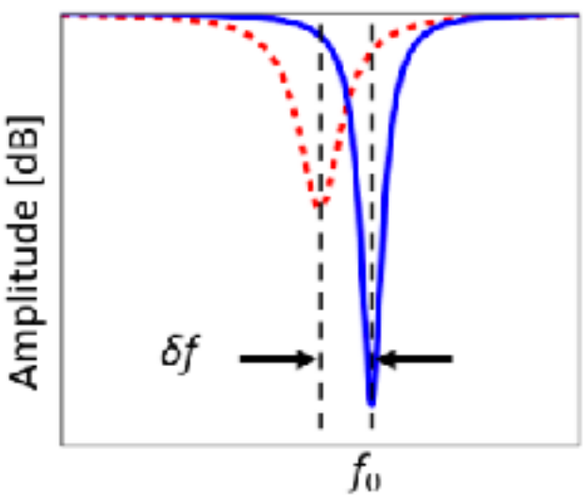


・ 入力 → インダクタンス変化 → 周波数変化



共振器そのものを検出器として利用

TES を 共振器に結合



・ ~ GHz の共振器での多重化が主流

私が関わっている研究

- チリ・アタカマの Simons Observatory / Simons Array
- スペイン・カナリアで行う GroundBIRD　　に参加

GroundBIRD

- テネリフェ島テイデ天文台
- 2019 年 9 月 first light !

Simons Observatory

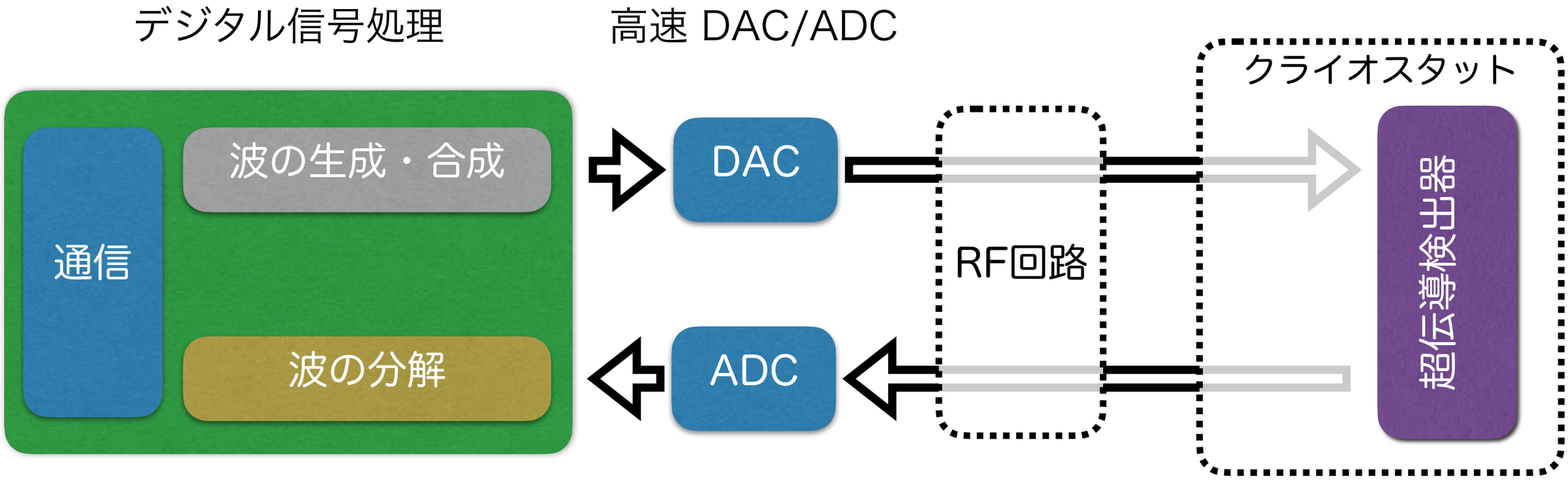
- 大口径＋小口径の組み合わせ
- 鋭意開発中

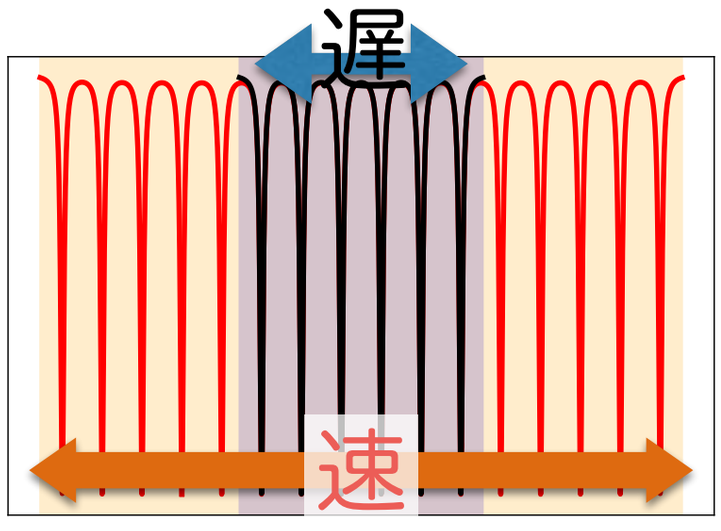
大口径望遠鏡 (LAT)

小口径望遠鏡 (SAT)

マイクロ波読み出しの
ための技術開発

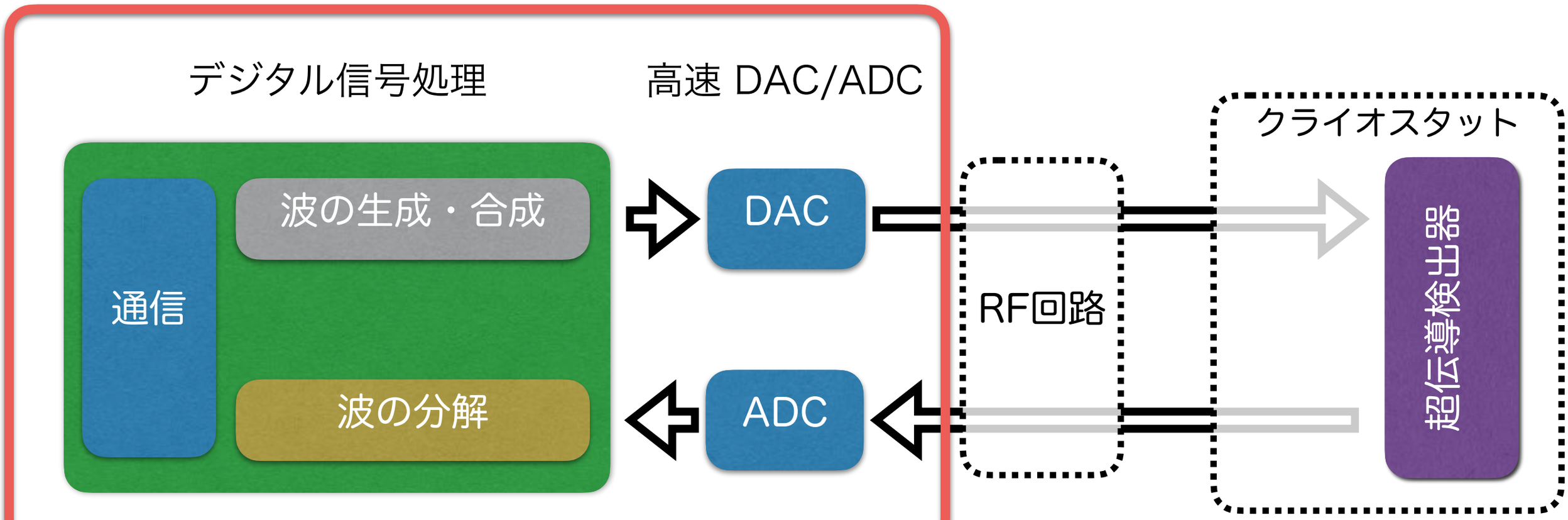
読み出しシステム

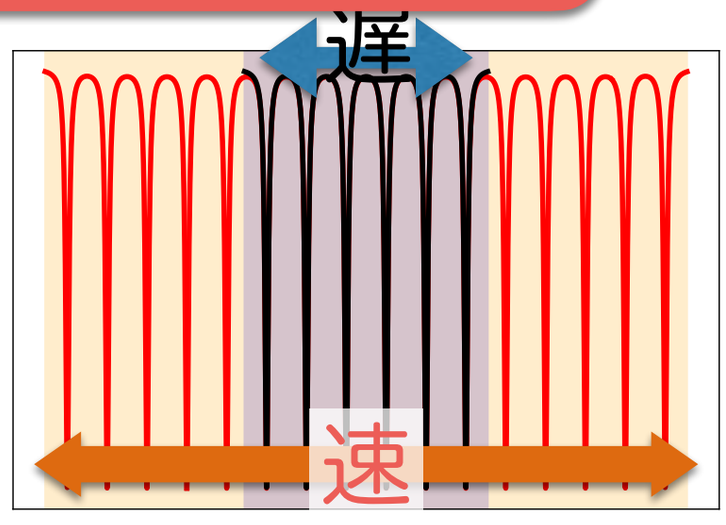




サンプリングがはやい方が有利

読み出しシステム

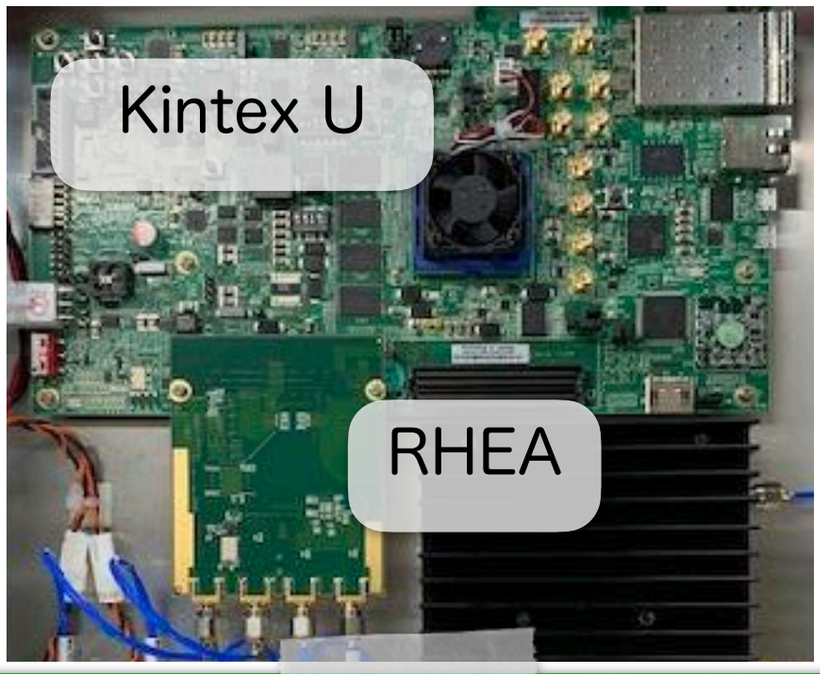




サンプリングがはやい方が有利

開発中のシステム

第１世代：
RHEAシステム



＊GroundBIRD用に開発した
アナログボードRHEA使用
＊200 MS/S
＊GroundBIRDで現状稼働

第２世代：
DAQ2システム

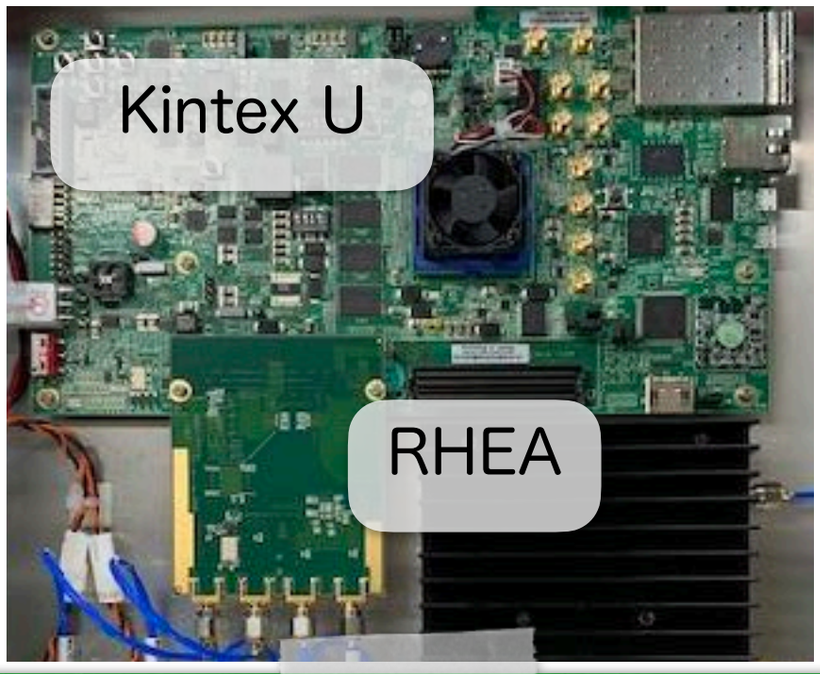
＊アナログ・デバイセズ社の
DAQ2ボードを使用
＊1000 MS/S
＊GB システムを置換え予定

第３世代：
RFSoC？

＊Xilinx の RFSoC
＊~ 4000 MS/S
＊次世代読み出しとして
期待

開発中のシステム

第１世代：
RHEAシステム



＊GroundBIRD用に開発した
アナログボードRHEA使用
＊200 MS/S
＊GroundBIRDで現状稼働

第２世代：
DAQ2システム

＊アナログ・デバイセズ社の
DAQ2ボードを使用
＊1000 MS/S
＊GB システムを置換え予定

第３世代：
RFSoC？

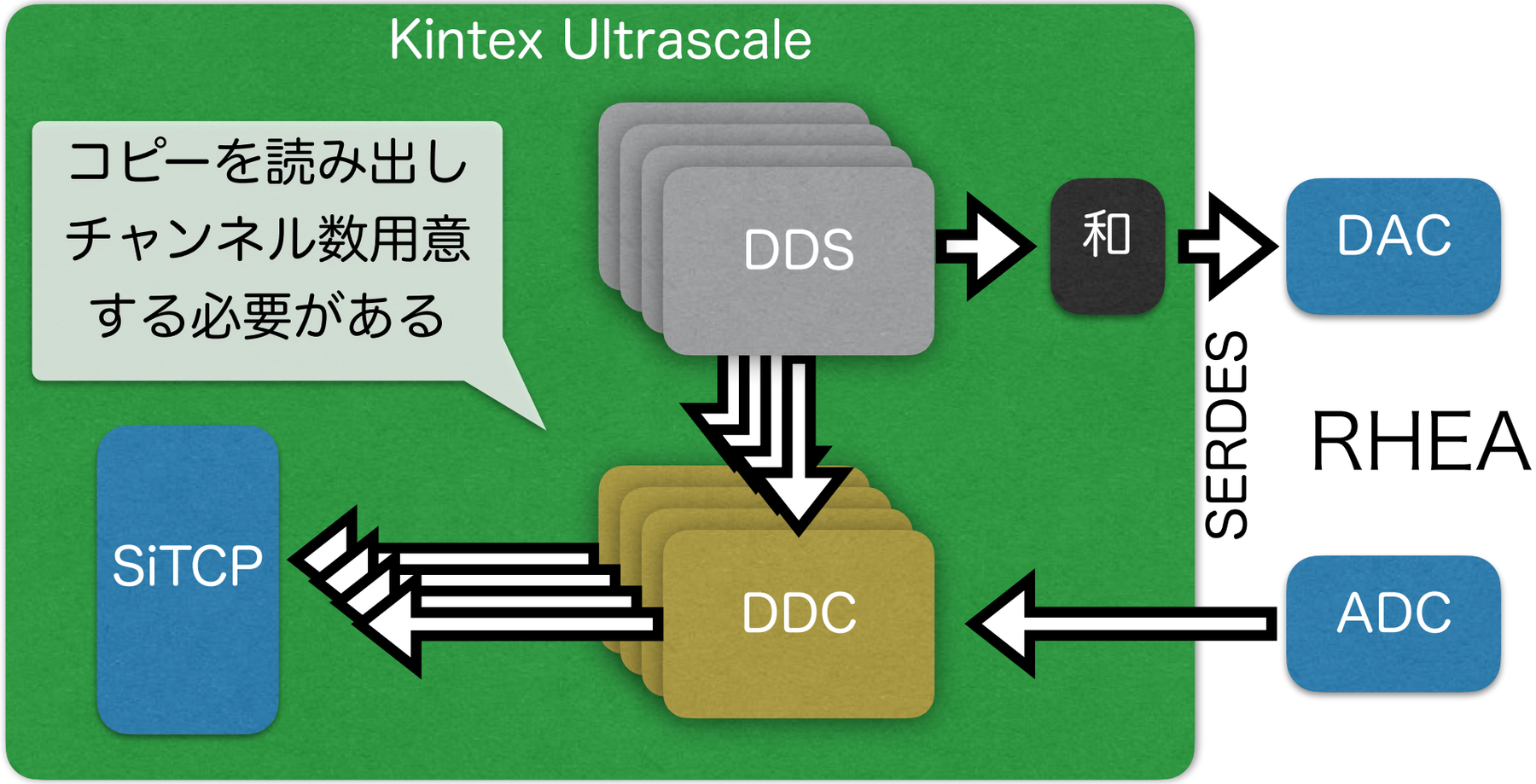
＊Xilinx の RFSoC
＊~ 4000 MS/S
＊次世代読み出しとして
期待

RHEA システム

- 独自開発の“RHEA”ボード
  + 市販 FPGAボード
- 200 MS/S
- 偏光観測望遠鏡 GroundBIRD で絶賛稼働中！

RHEA と FPGA ボードの接触不良(FMC)をデバッグ中の様子

RHEA システムでの処理



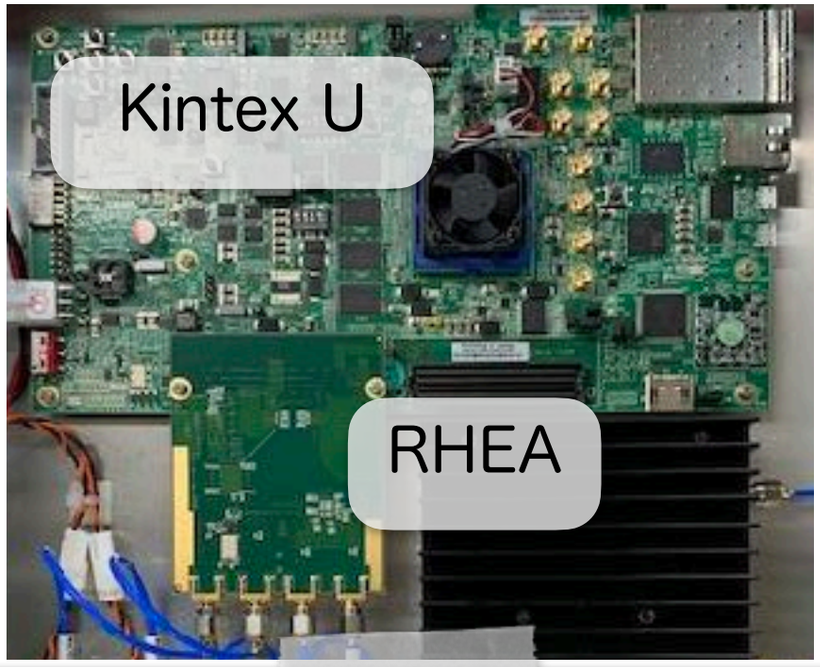
DDS: 波の生成　DDC: 特定周波数の取出

チャンネル数の増大

→容量(BRAM など)不足・タイミングエラー

開発中のシステム

第１世代：
RHEAシステム



＊GroundBIRD用に開発した
アナログボードRHEA使用
＊200 MS/S
＊GroundBIRDで現状稼働

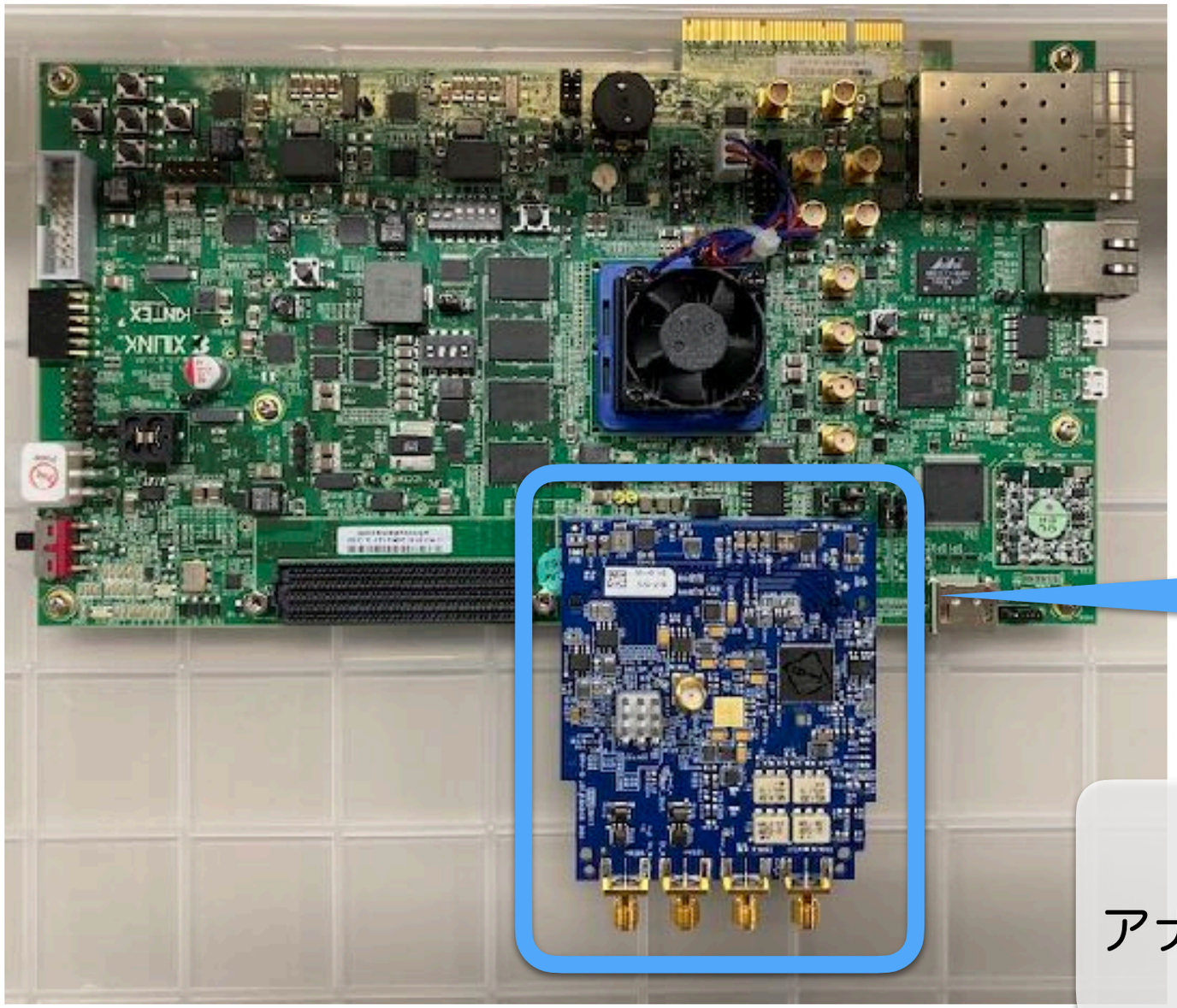
第２世代：
DAQ2システム

＊アナログ・デバイセズ社の
DAQ2ボードを使用
＊1000 MS/S
＊GB システムを置換え予定

第３世代：
RFSoC？

＊Xilinx の RFSoC
＊~ 4000 MS/S
＊次世代読み出しとして
期待

DAQ2 システム



FPGAボード

KCU 105 (Xilinx Kintex U)

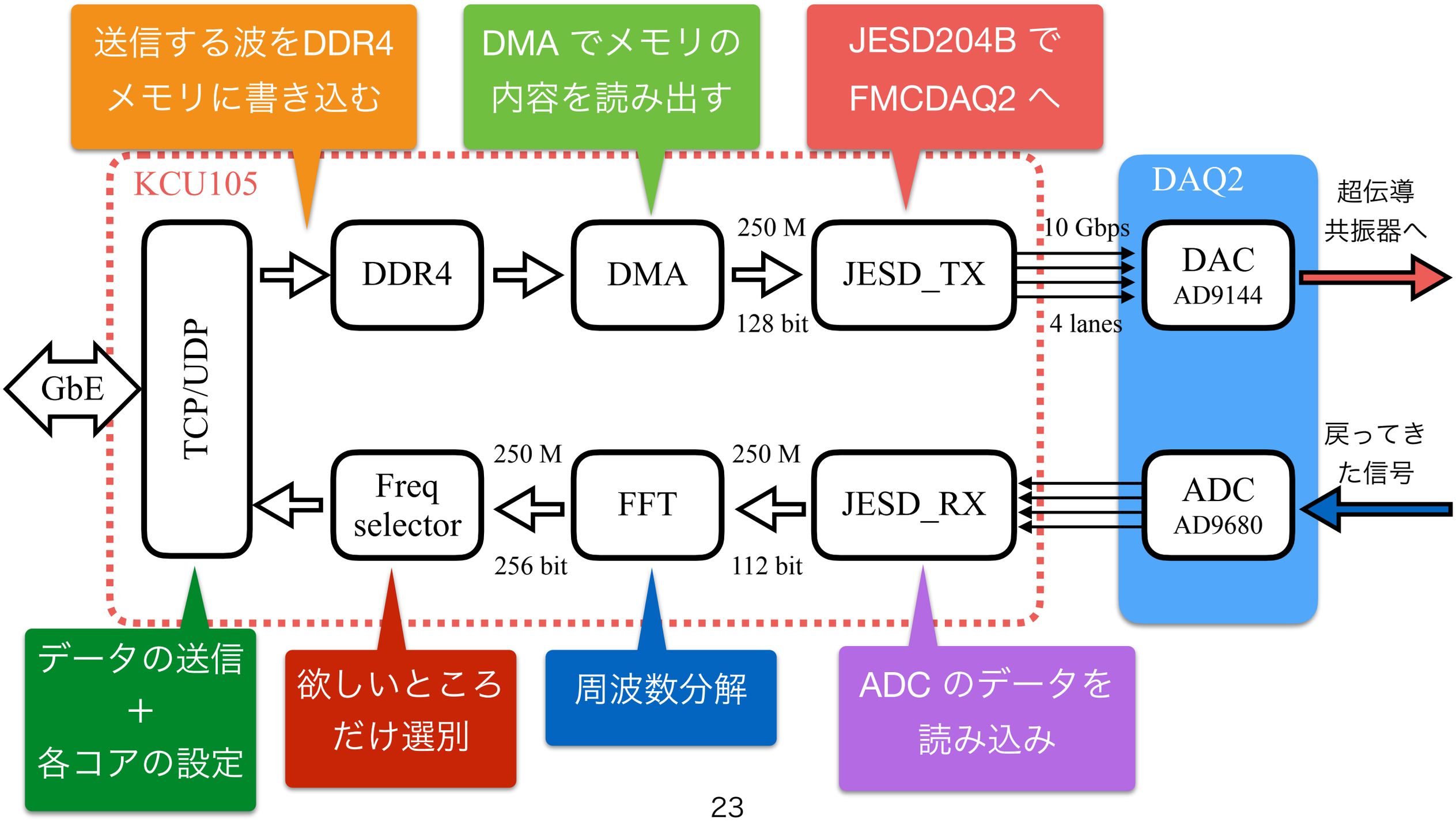
AD/DAボード:

FMCDAQ2

- **1000 MS/S**
  → RHEAの **5 倍**！

デジタルボードはそのまま、
アナログボードだけ高速のものに置き換え

ファームウェア


送信する波をDDR4
メモリに書き込む
DMA でメモリの
内容を読み出す
JESD204B で
FMCDAQ2 へ
KCU105
DAQ2
超伝導
共振器へ
GbE
TCP/UDP
DDR4
DMA
250 M
128 bit
JESD_TX
10 Gbps
4 lanes
DAC
AD9144
戻ってき
た信号
ADC
AD9680
JESD_RX
250 M
112 bit
FFT
250 M
256 bit
Freq
selector
データの送信
＋
各コアの設定
欲しいところ
だけ選別
周波数分解
ADC のデータを
読み込み

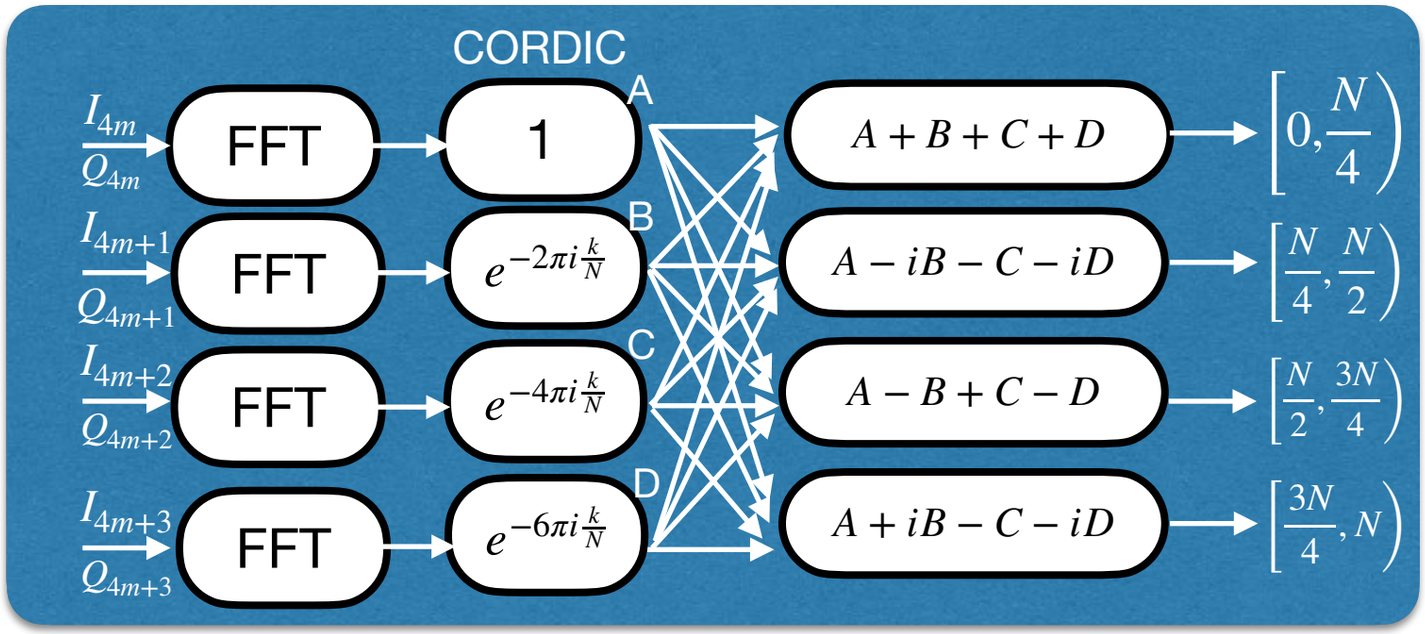
ファームウェア開発のポイント

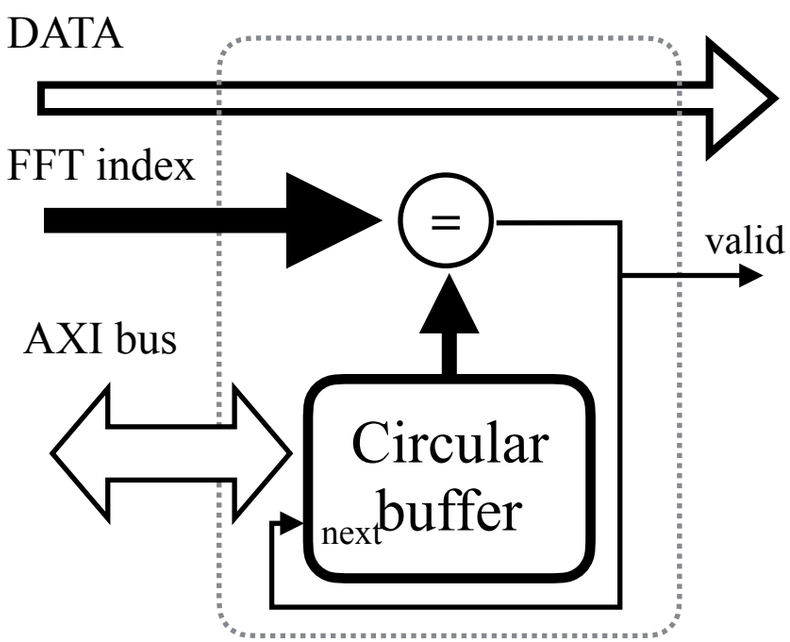
- 1 GHz の信号
  …そのままは扱えない
  → 4 レーンに分割処理

- DDC → FFT
  必要リソース量削減

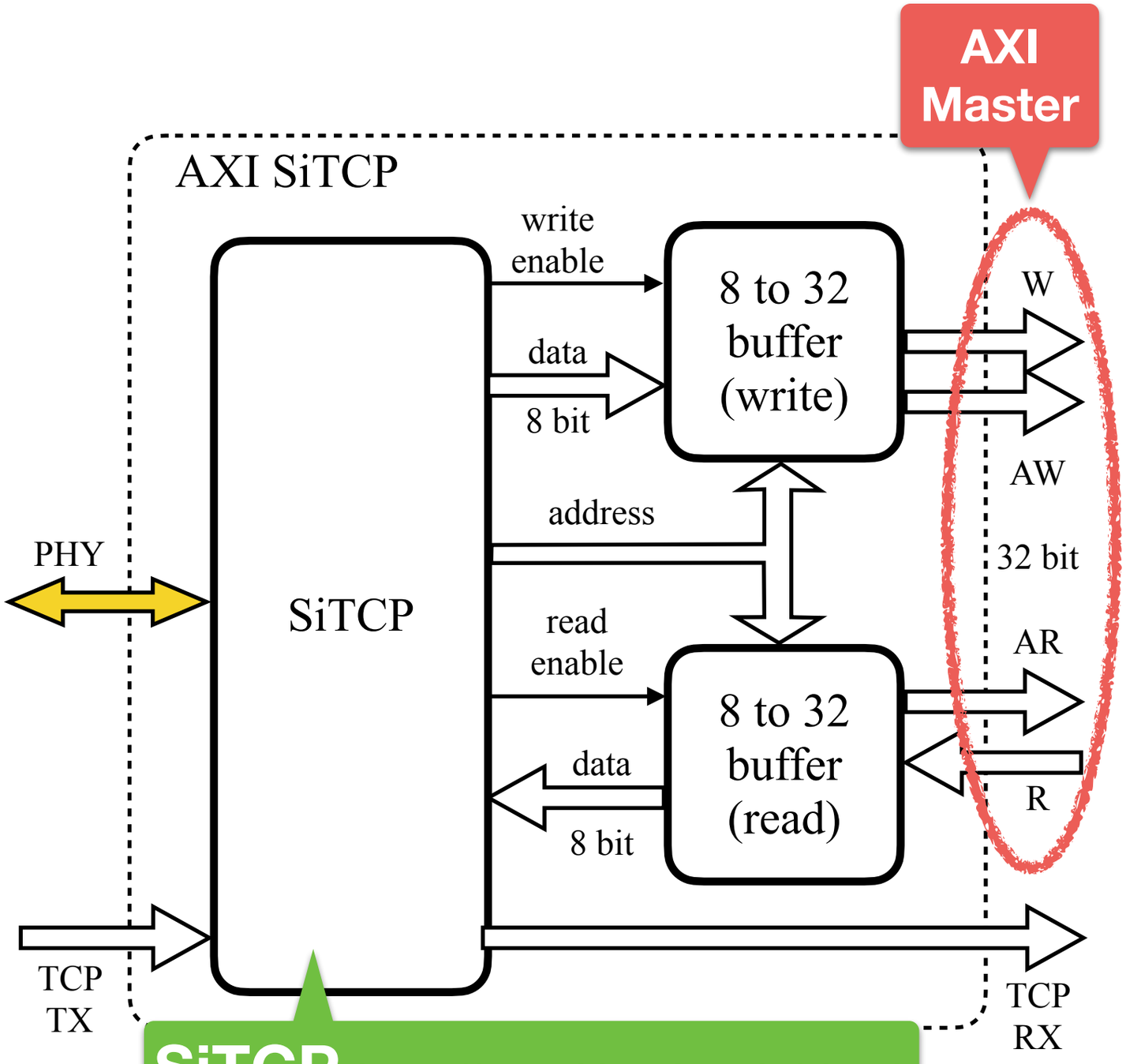
- 必要な情報を選択
  → 転送量を落とす

FFT のコア





AXI SiTCP



https://github.com/dixilo/axi_sitcp

- 各 IP コア (モジュール)設定 … AXI バスを利用
- SiTCP のスローコントロール用のバス (RBCP) を、 AXI 化する wrapper を作成した

できること

- AXI バスを持つ IP コアの設定など
- メモリへの書き込み

SiTCP https://www.bbtech.co.jp/sitcp/

- FPGA をネットワークに接続
- TCP で 1 Gbps までのデータ転送が可能
- UDP でスローコントロール

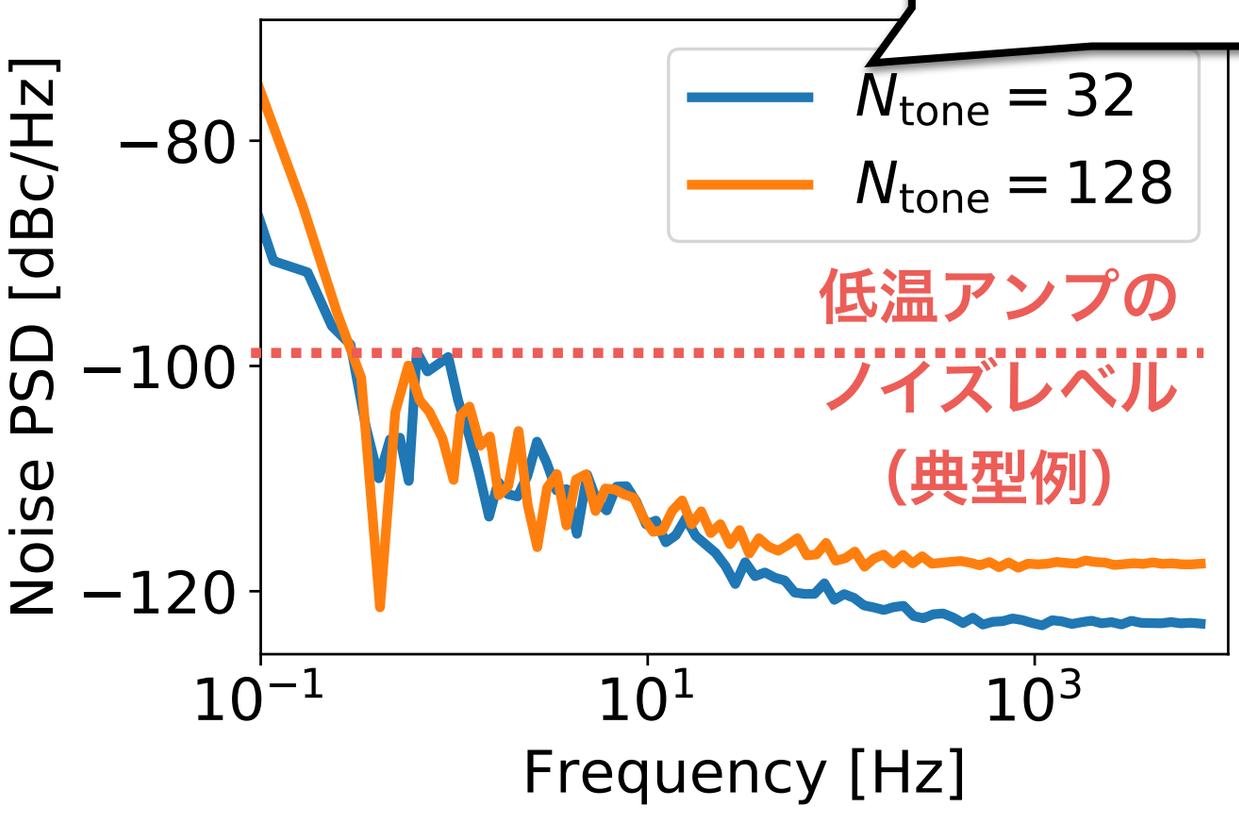
AXI DMA など

ごりやく

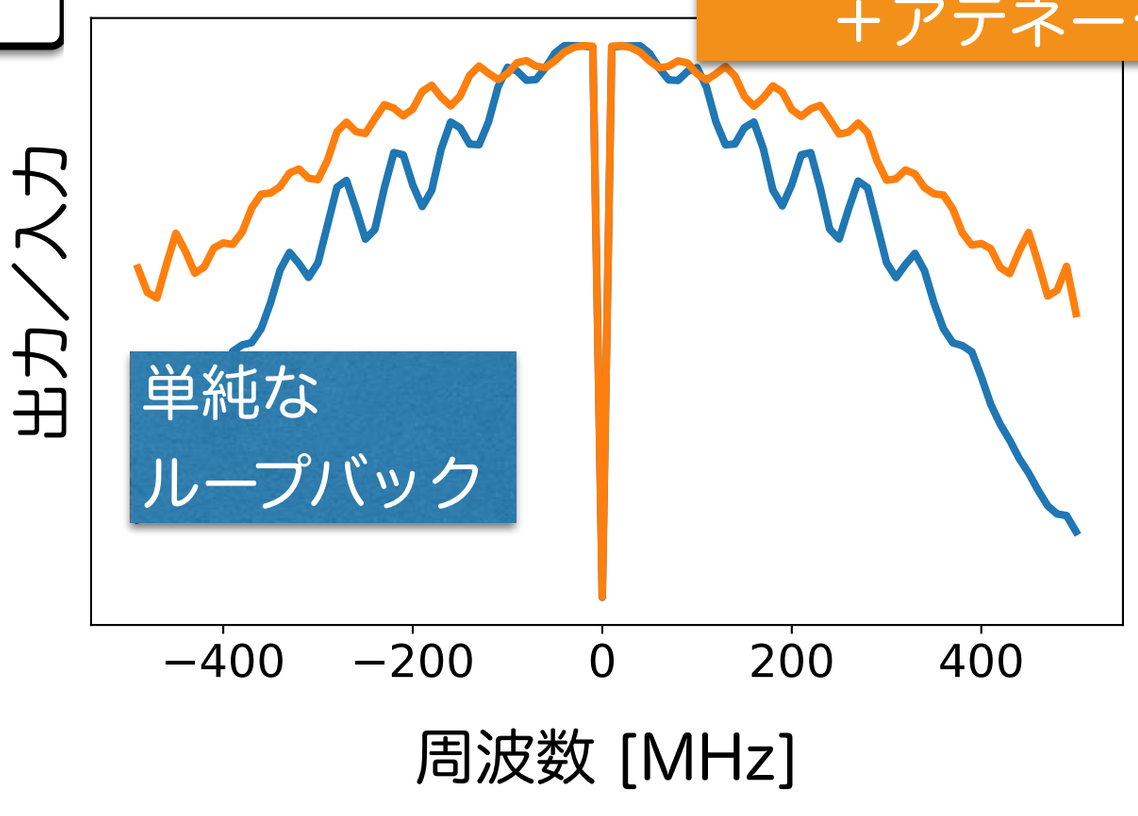
- Xilinx が提供しているコアの AXI インタフェースがそのまま使える

DAQ2 の評価

ノイズ特性



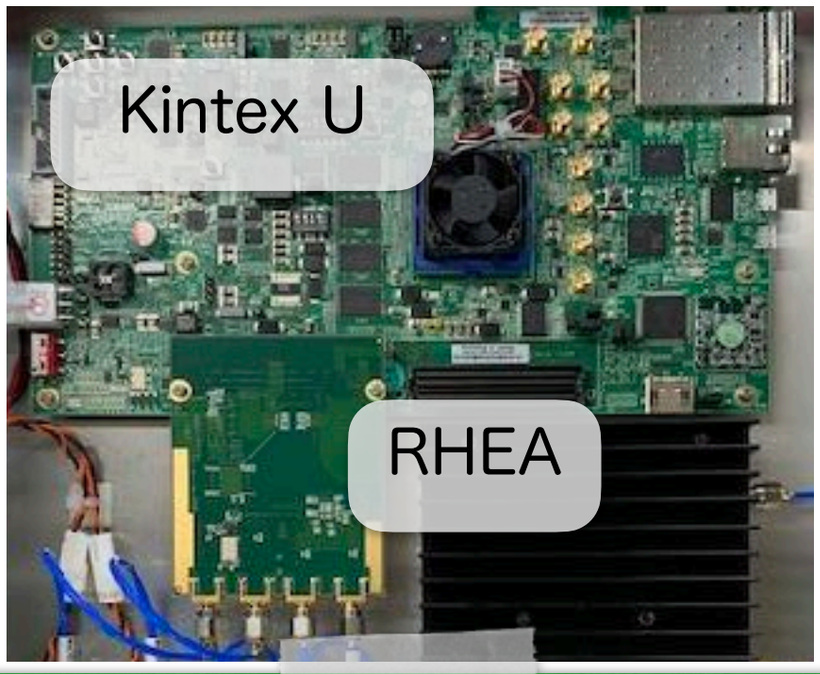
周波数特性



- ノイズや周波数特性を評価
- 論文執筆に向けて準備中

開発中のシステム

第１世代：
RHEAシステム



＊GroundBIRD用に開発した
アナログボードRHEA使用
＊200 MS/S
＊GroundBIRDで現状稼働

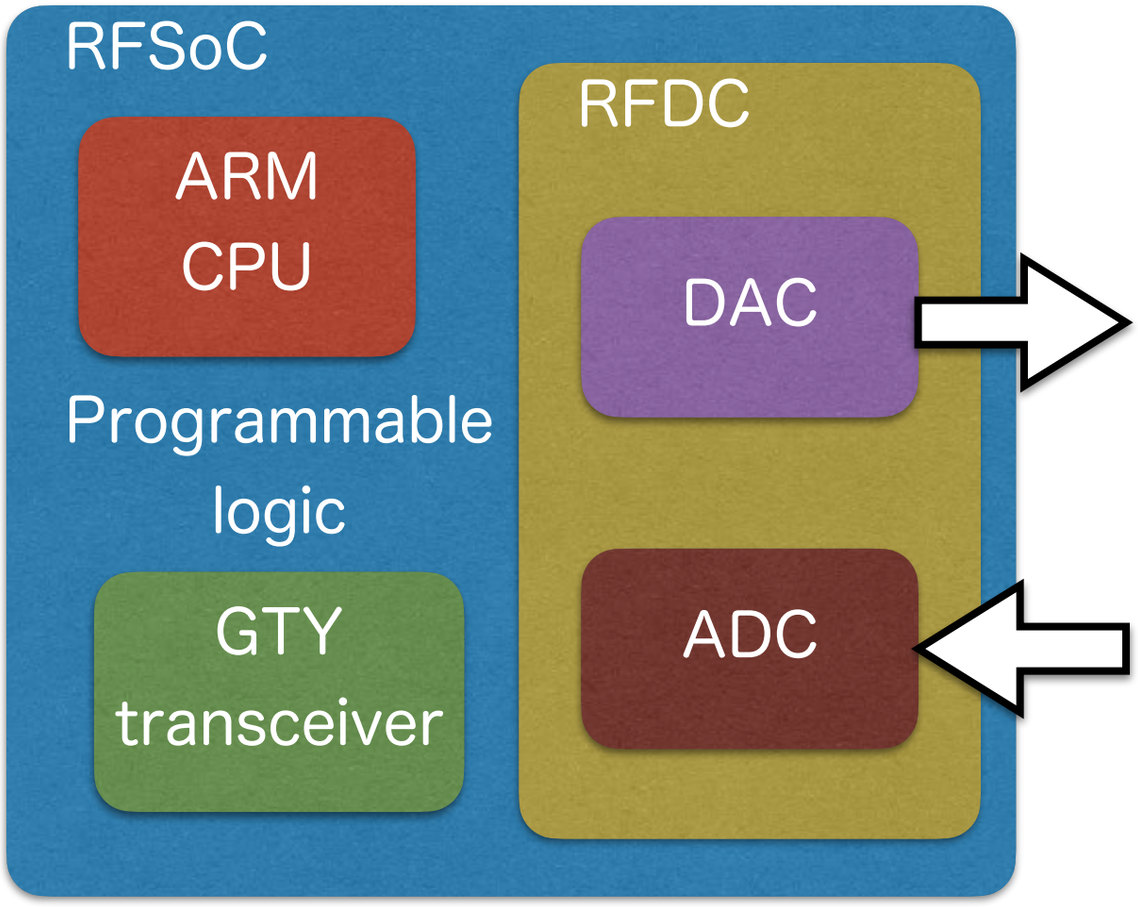
第２世代：
DAQ2システム

＊アナログ・デバイセズ社の
DAQ2ボードを使用
＊1000 MS/S
＊GB システムを置換え予定

第３世代：
RFSoC？

＊Xilinx の RFSoC
＊~ 4000 MS/S
＊次世代読み出しとして
期待

Zynq Ultrascale+ RFSoC



- ひとつのチップに FPGA データコンバータ、CPU が統合

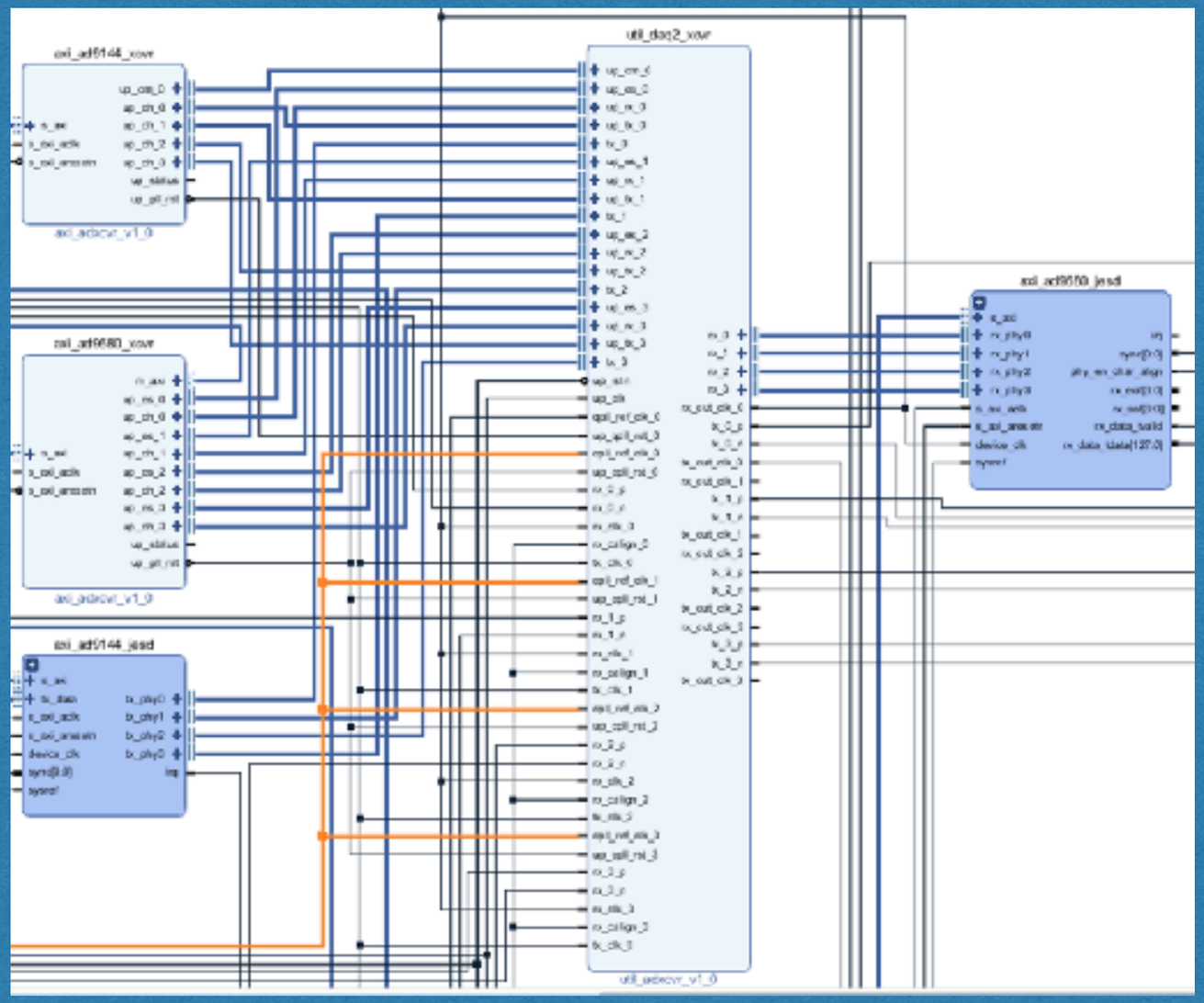
- 利点

① あつかいやすい

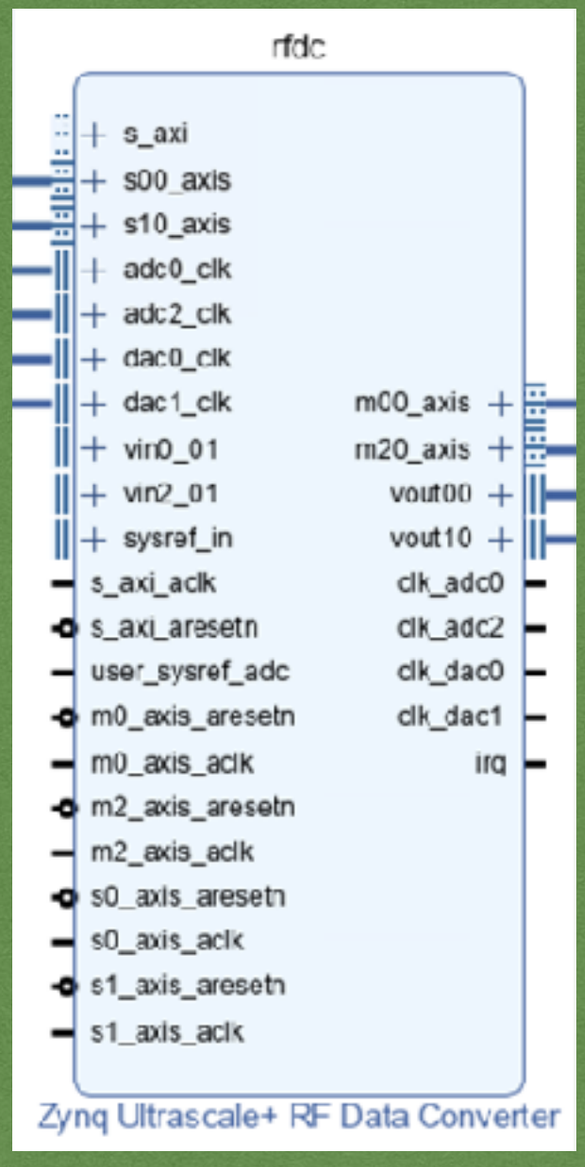
② 広帯域

例）Gen3
DAC 10 GSPS
アナログ帯域 6GHz

従来方式 (JESD204B)

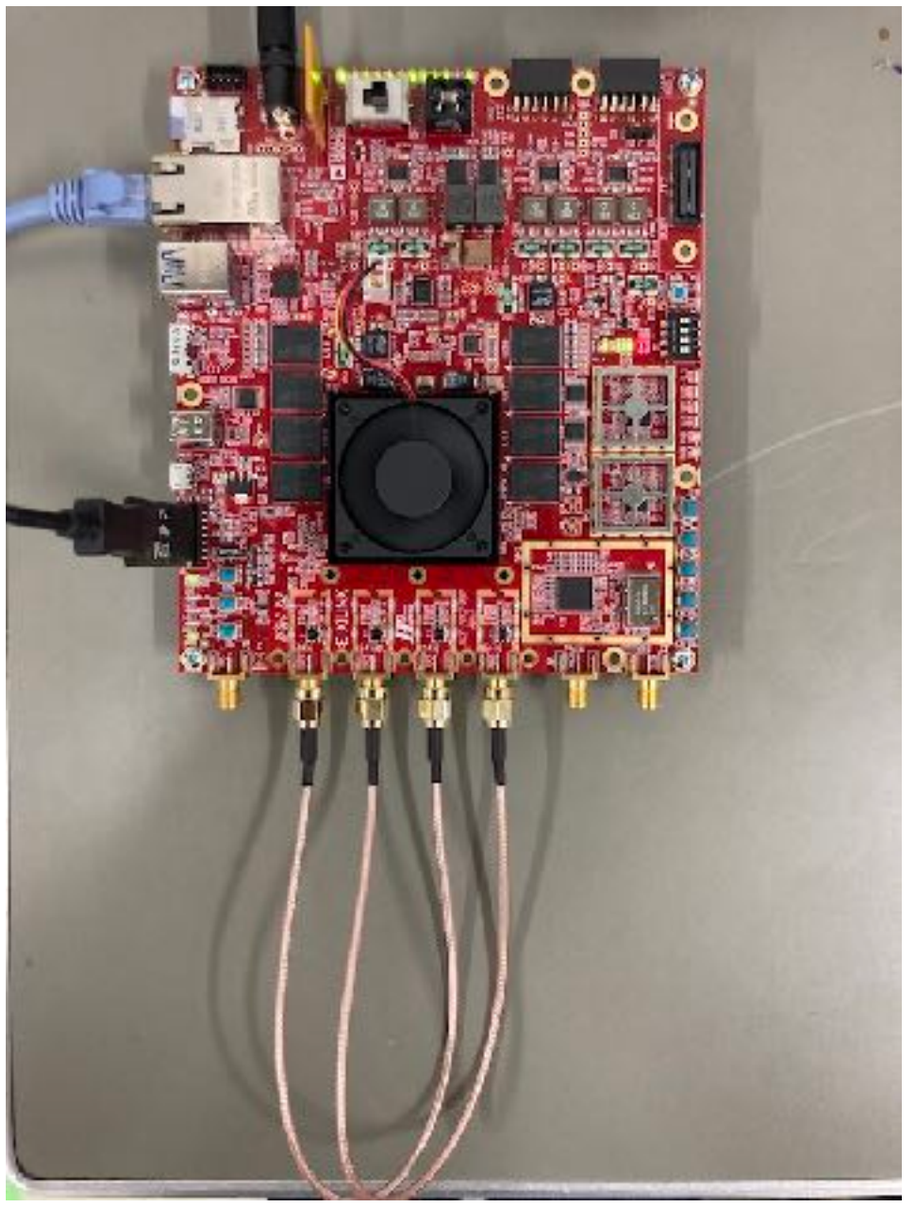


RFSoC



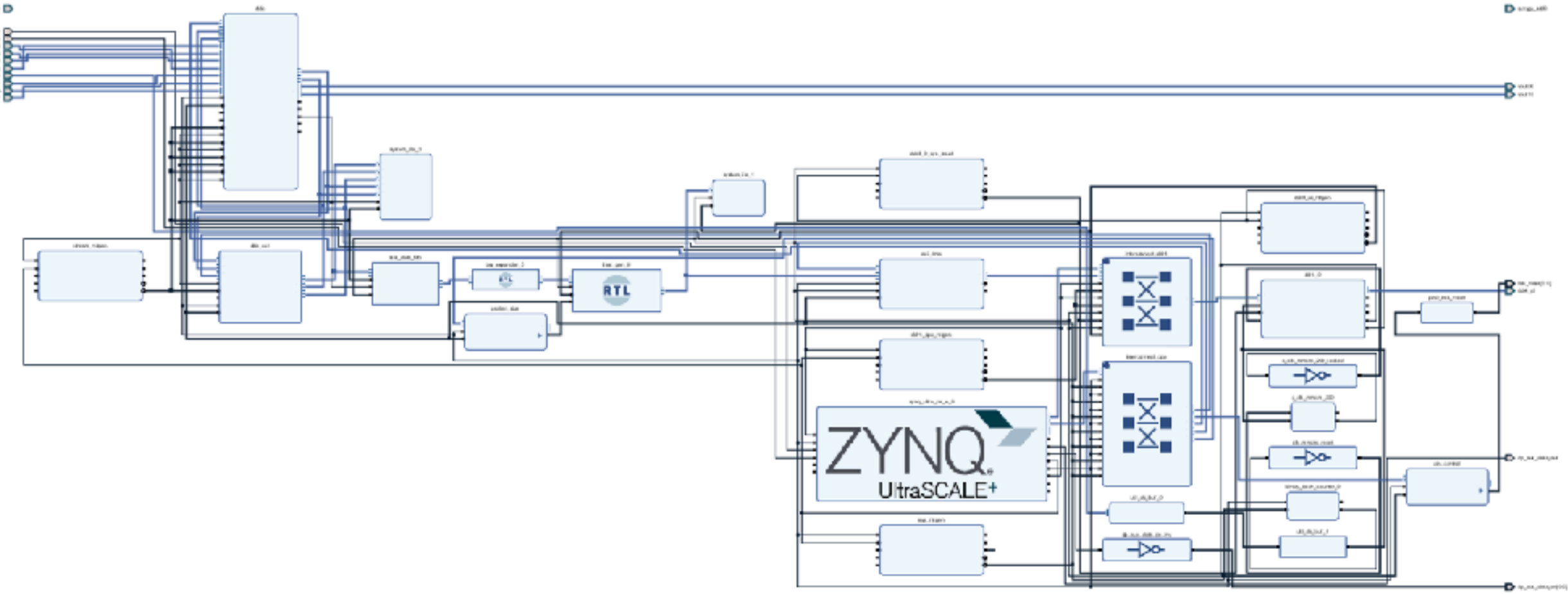
大幅な簡略化

RFSoC 2x2



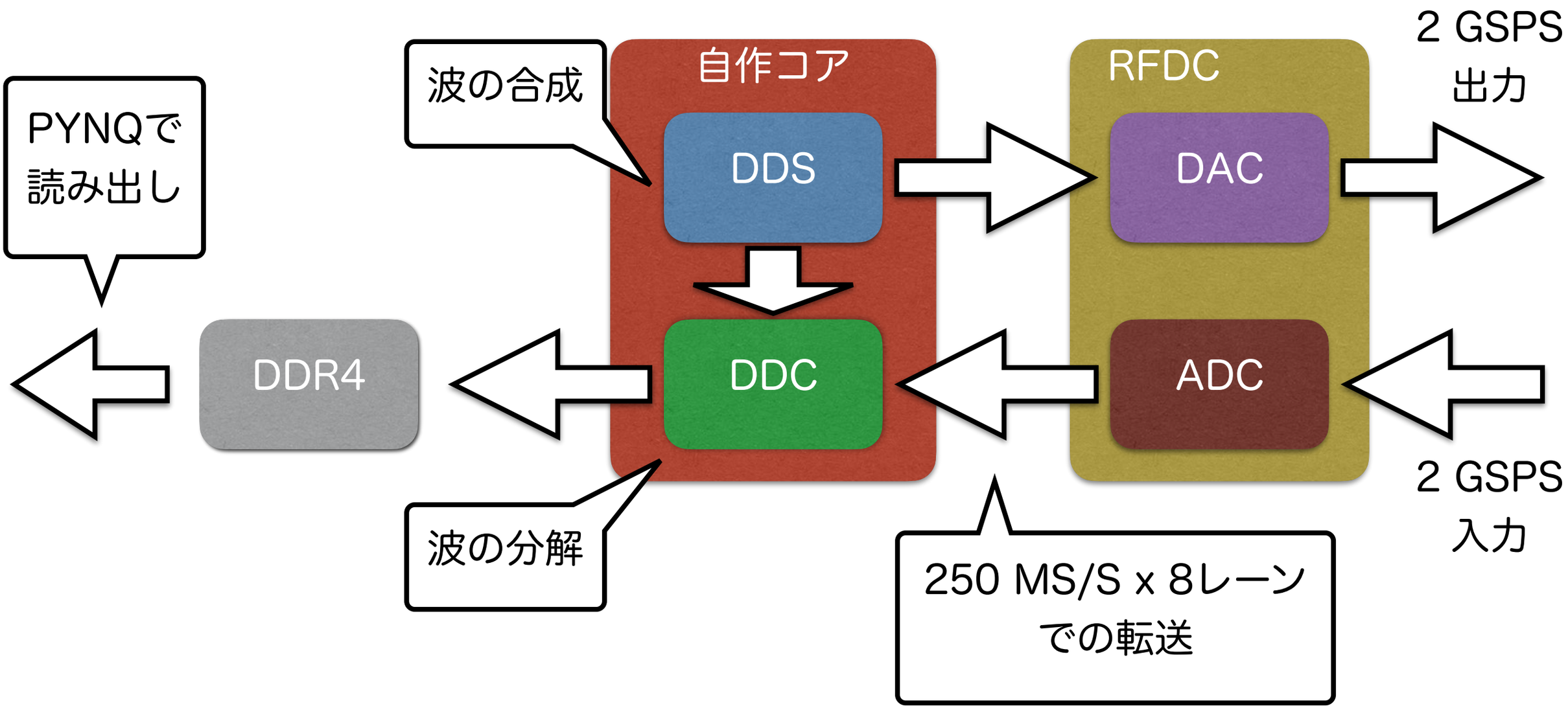
- RFSoC 搭載の評価ボード
- アカデミック用途なら格安
  （＜３０万円）
- まずはこれで試してみる

RFSoC テストのファームウェア



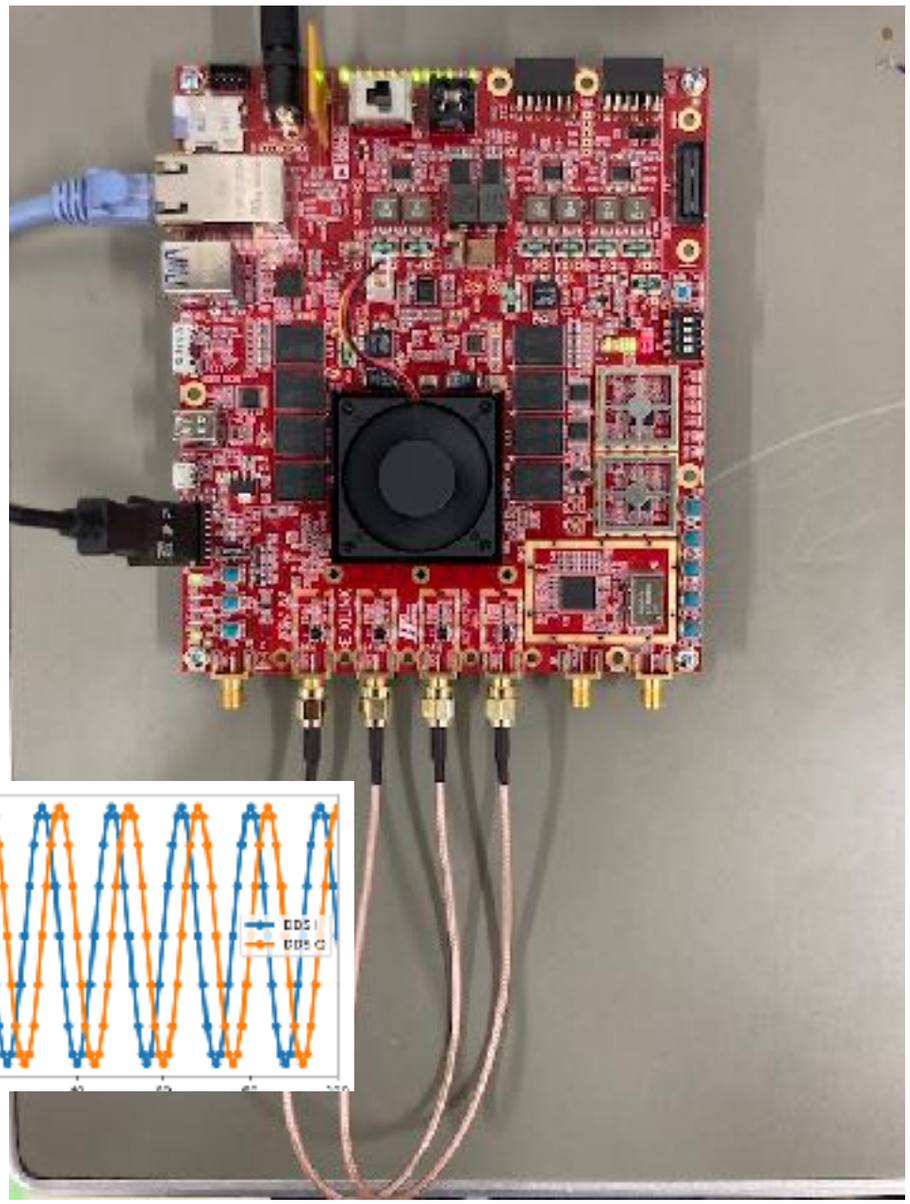
2 GSPS に対応可能なファームウェアを作成した

RFSoC テストのファームウェア

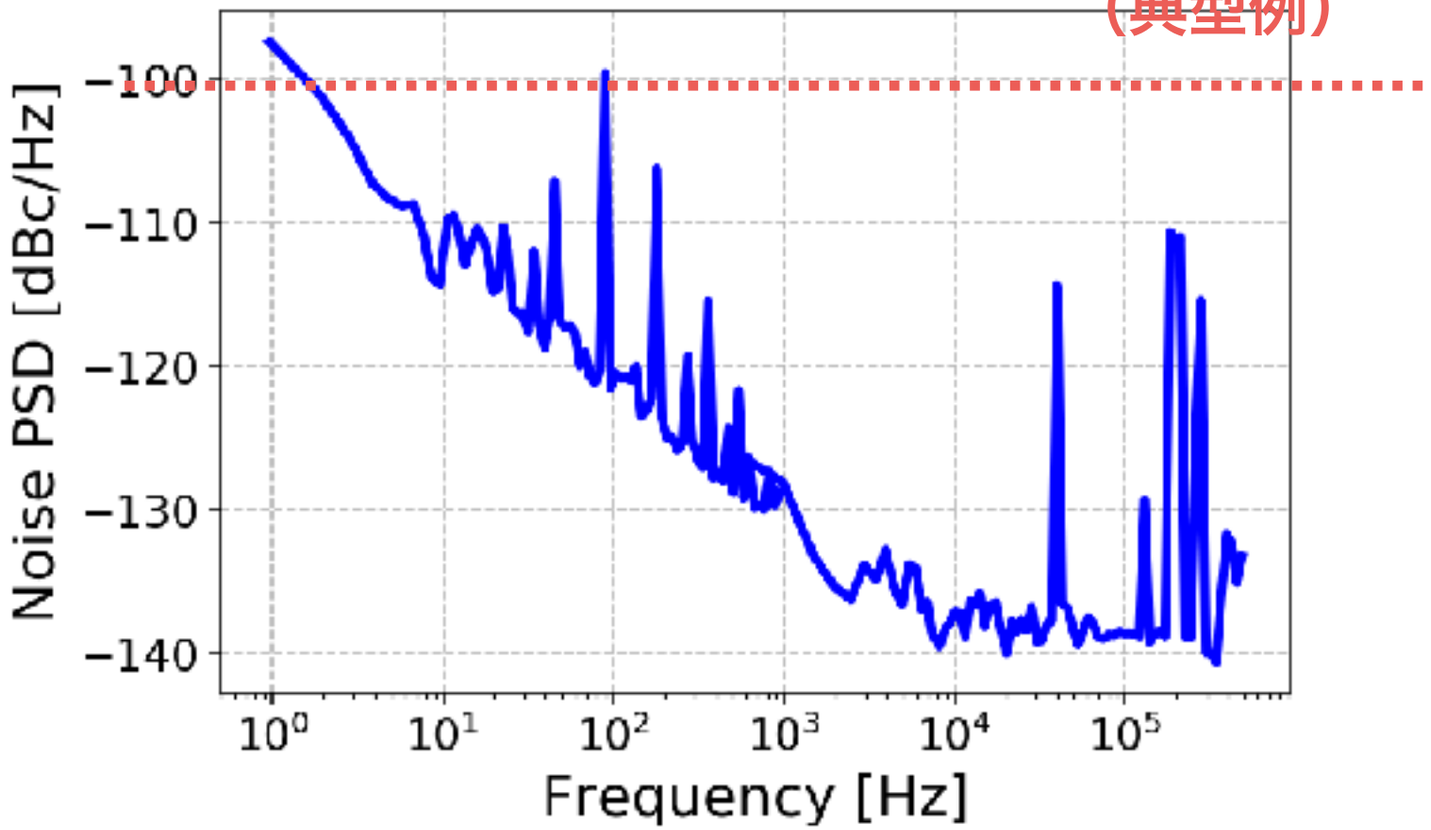


試験結果

ループバックでの試験

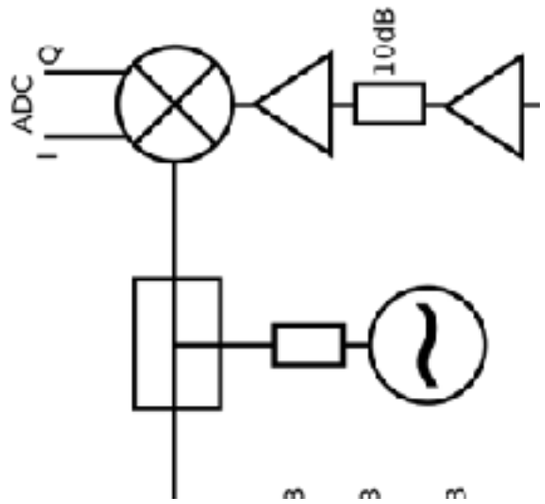


低温アンプのノイズレベル（典型例）

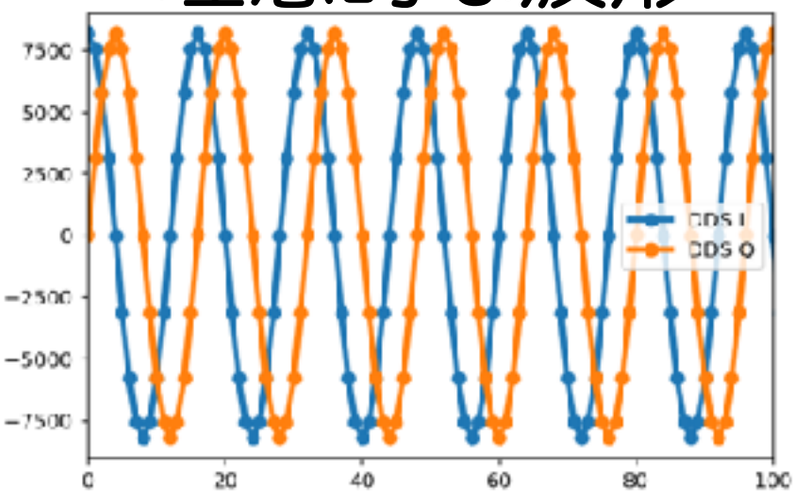


- $N_{tone} = 1$
- 目標ノイズレベルよりも十分低い

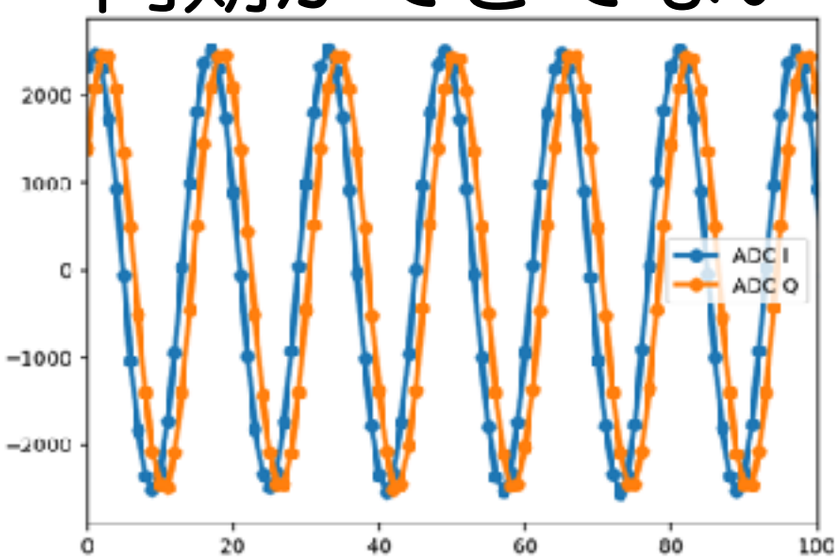
同期の問題



理想的な波形



同期ができてない



- ADC/DAC がそれぞれ 2 つ必要
  （IQミキサを使う。同期必須）
- RFSoC: MTS (Multi Tile Sync) 機能
- PYNQ (ソフトウェア)に未実装
  → C のコードを自分で移植 (~2000 行)
- …が、動かない。デバッグ中
  - そもそも同期信号(SYSREF)の設定が違う？

まとめ

- マイクロ波読み出しについて
  - 配線の本数が劇的に減らせる→熱流入削減
- 実際に作っているもの
  - １：RHEAシステム　→　GBで稼働中！
  - ２：DAQ２システム　→　詰めの段階
  - ３：RFSoC 2x2　→　使えるかどうか検証中
- 人員不足（実質、私ひとりでやっている）
  - 興味のあるポスドク・学生の方を募集

Acknowledgement

- This work is supported by Grants-in-Aid for Scientific Research from The Ministry of Education, Culture, Sports, Science and Technology, Japan (KAKENHI Grant No. 18K13568), U.S.-Japan Science and Technology Cooperation Program in High Energy Physics, and JSPS Core-to-Core program (JPJSCCA20200003).

- The authors would like to thank Josef C. Frisch (SLAC), Akito Kusaka (Univ. of Tokyo, LBNL) and Koji Ishidoshiro (Tohoku U.) for their technical advice.

バックアップ

先行研究

- “RHEA” を用いたシステム
  - CMB偏光望遠鏡 GroundBIRD などで使用
  - 独自開発の“RHEA”ボード + 市販 FPGAボード
  - 帯域は 200 MHz

ちょっと狭い…

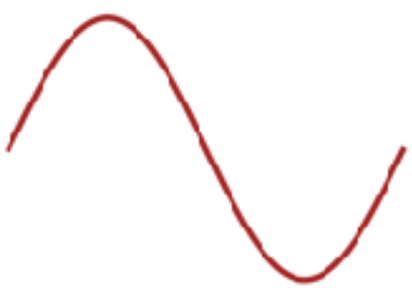
ADC:	AD9680
分解能	14 bit
速度	1000 MSPS
DAC:	AD9144
分解能	16 bit
速度	2500 MSPS

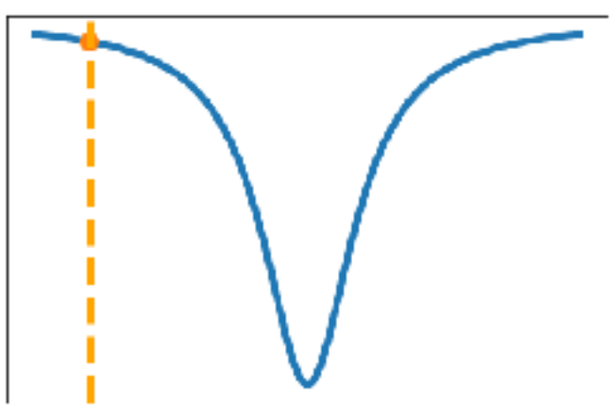
表 1: RF データ コンバーター サブシステムの機能

		ZU21DR	ZU25DR ZU27DR ZU28DR	ZU29DR	ZU39DR	ZU42DR	ZU43DR	ZU46DR	ZU47DR ZU48DR	ZU49DR
		Gen 1			Gen 2	Gen 3				
12 ビット RF-ADC、DDC あり	ADC 数	0	8	16	16	–	–	–	–	–
	最大レート (GSPS)	0	4.096	2.058	2.220	–	–	–	–	–
14 ビット RF-ADC、DDC あり	ADC 数	–	–	–	–	8 2	4	8 4	8	16
	最大レート (GSPS)	–	–	–	–	2.5 5.0	5.0	2.5 5.0	5.0	2.5
14 ビット RF-DAC、DUC あり	DAC 数	0	8	16	16	8	4	12	8	16
	最大レート (GSPS)	0	6.554	6.554	6.554	10.0	10.0	10.0	10.0	10.0
RF-ADC あたりの DDC 数[1]		0	1	1	1	1	2	1	1	1
RF 入力周波数最大 (GHz)		4			5	6				
間引き/補間		1x、2x、4x、8x			1x、2x、4x、8x	1x、2x、3x、4x、5x、6x、8x、10x、12x、16x、20x、24x、40x				

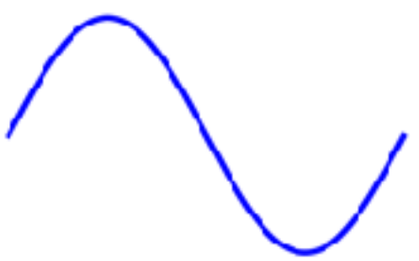
読み出し原理

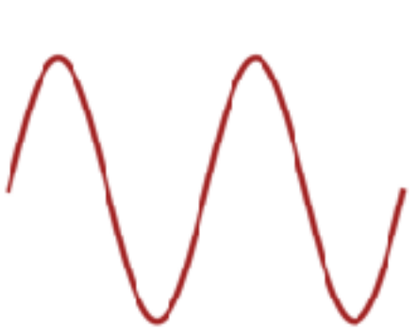
Input

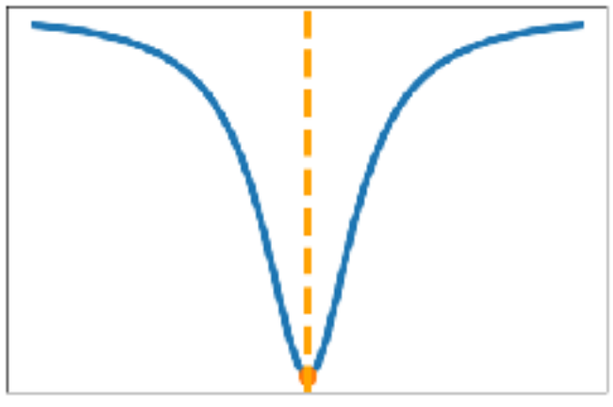


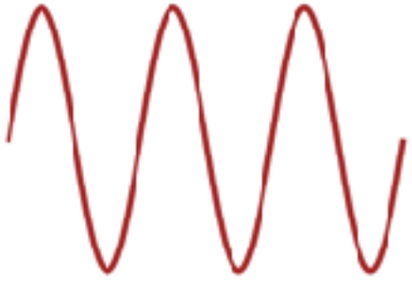


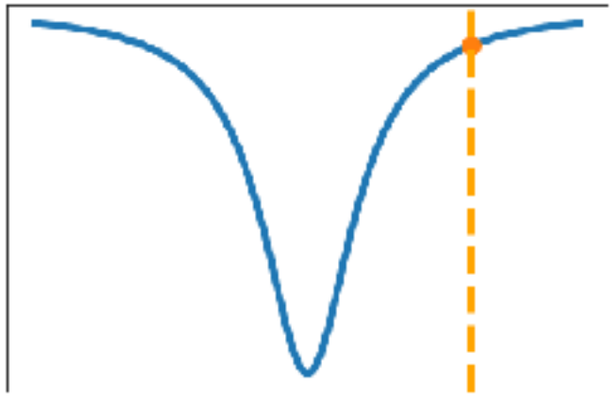
Output

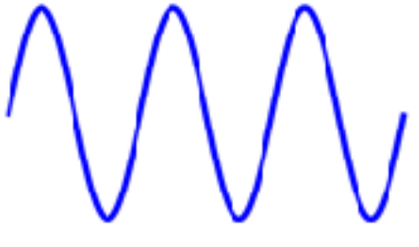








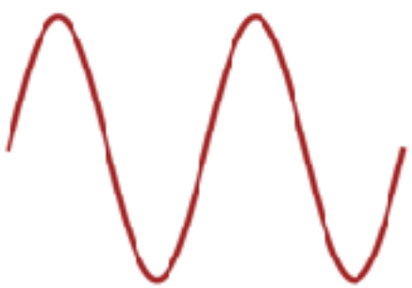


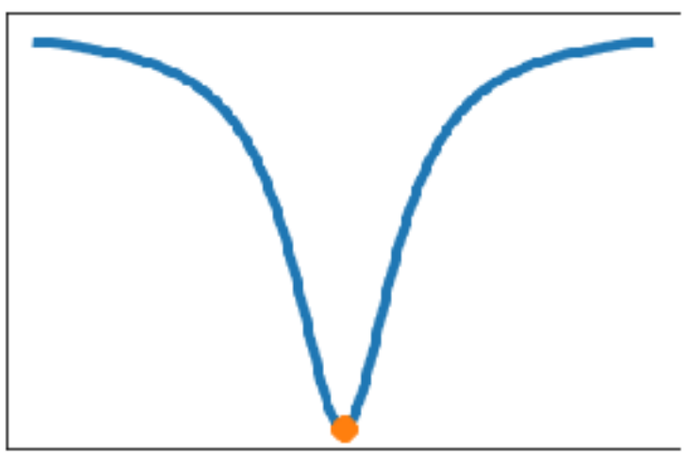


Frequency

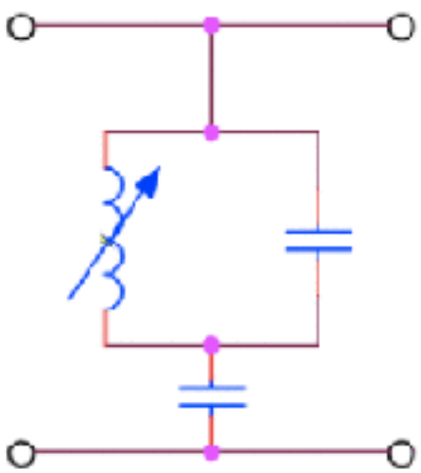
読み出し原理

Input





Output

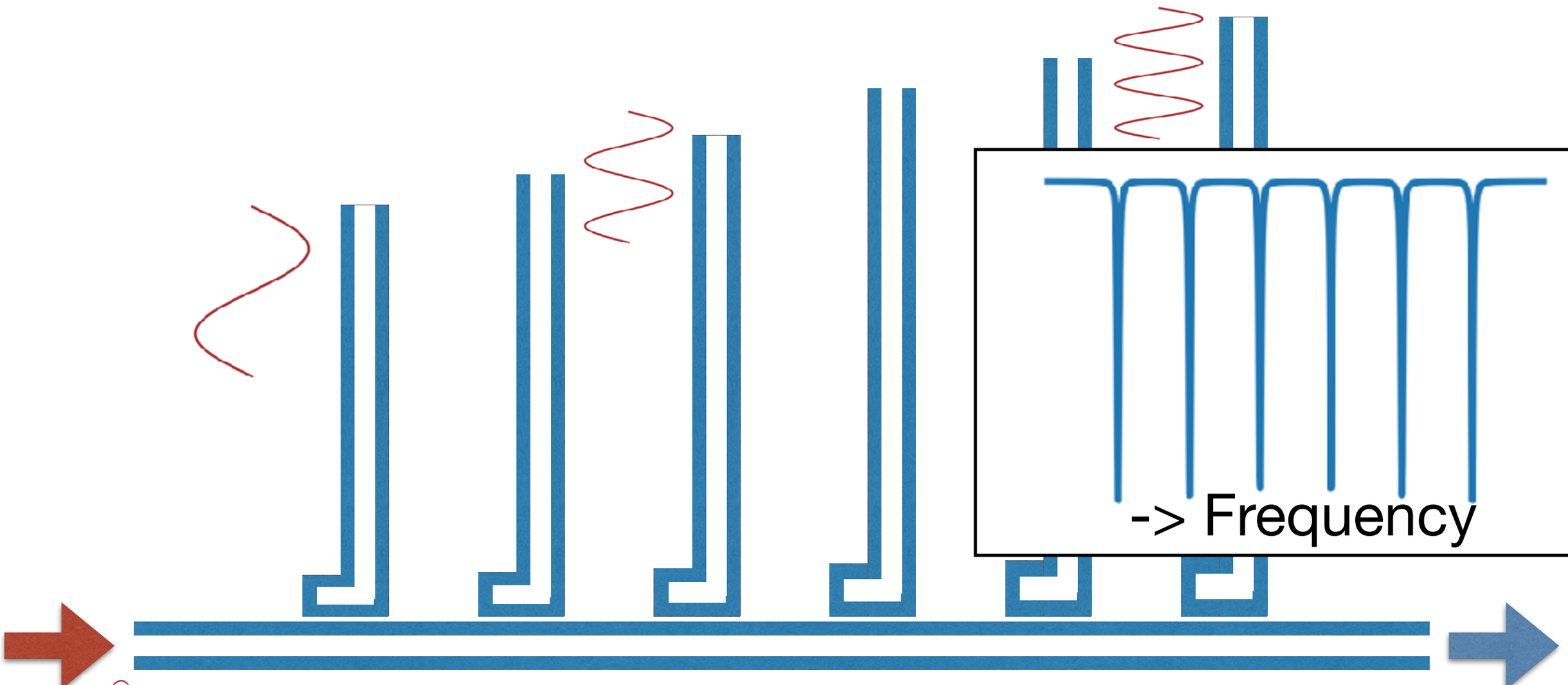


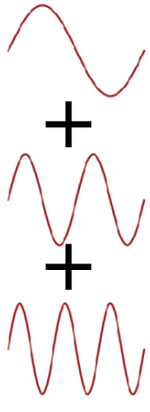
Output

Input

Time →

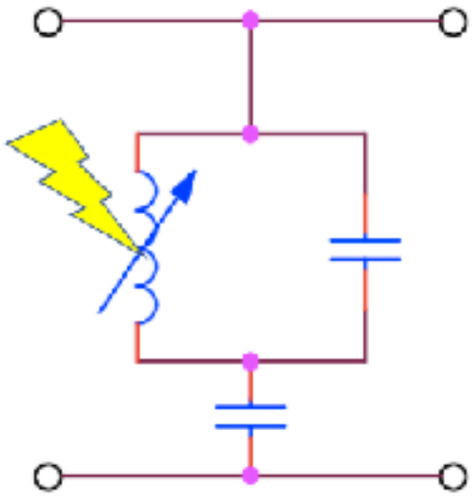
多重化



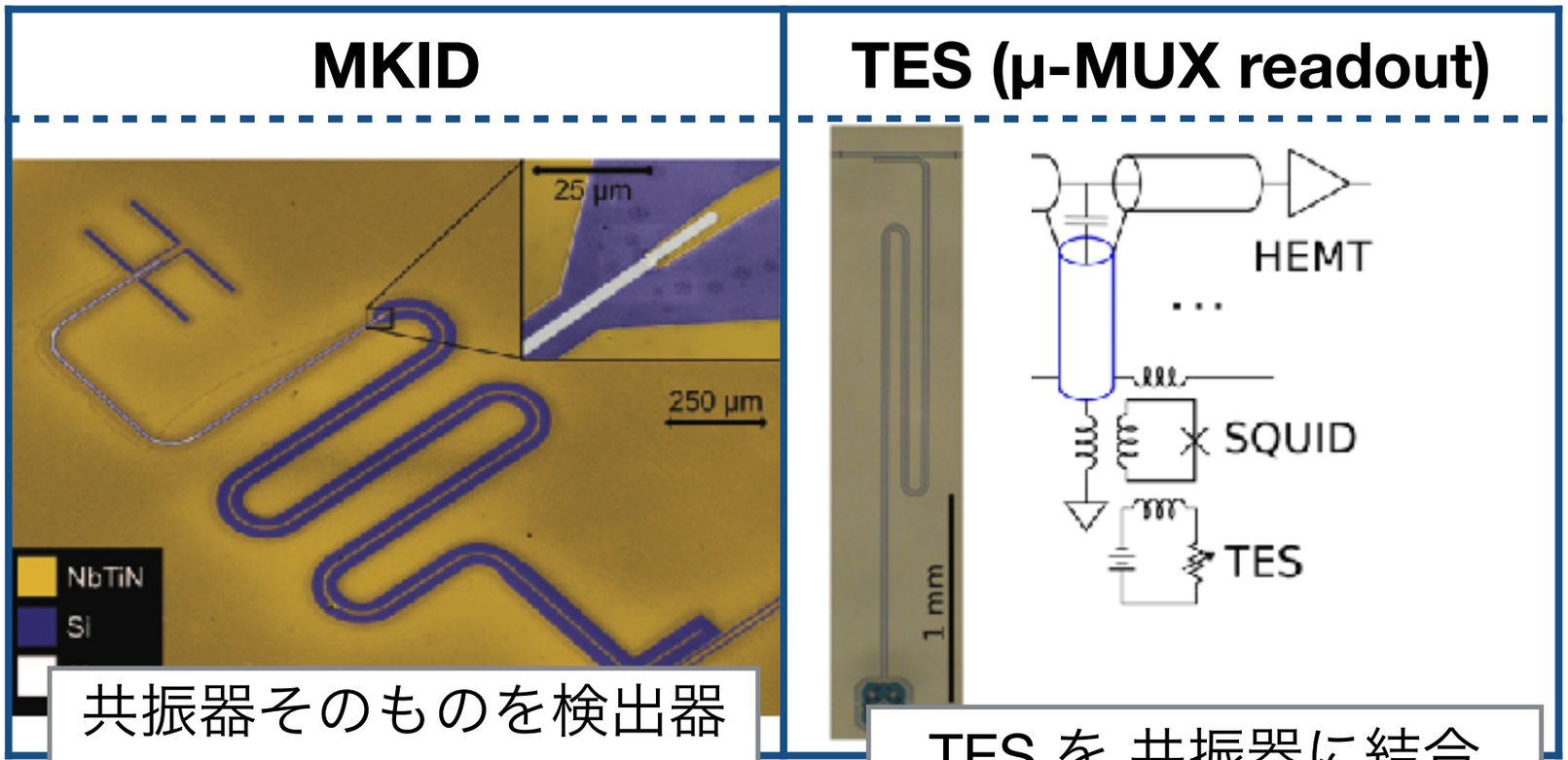


長さの違う（＝共振周波数の違う）
共振器を複数設置

超伝導共振器

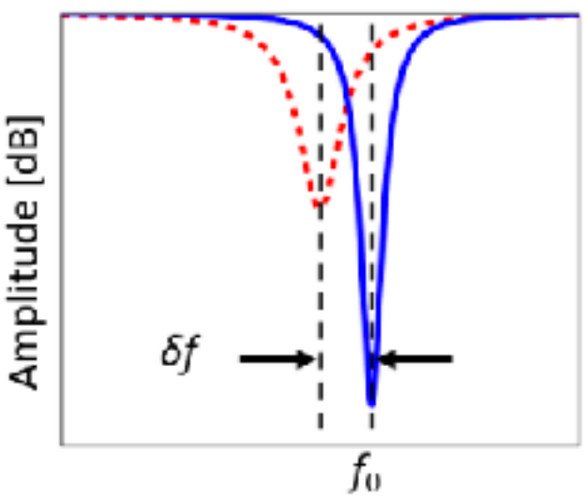


・ 入力 → インダクタンス変化 → 周波数変化



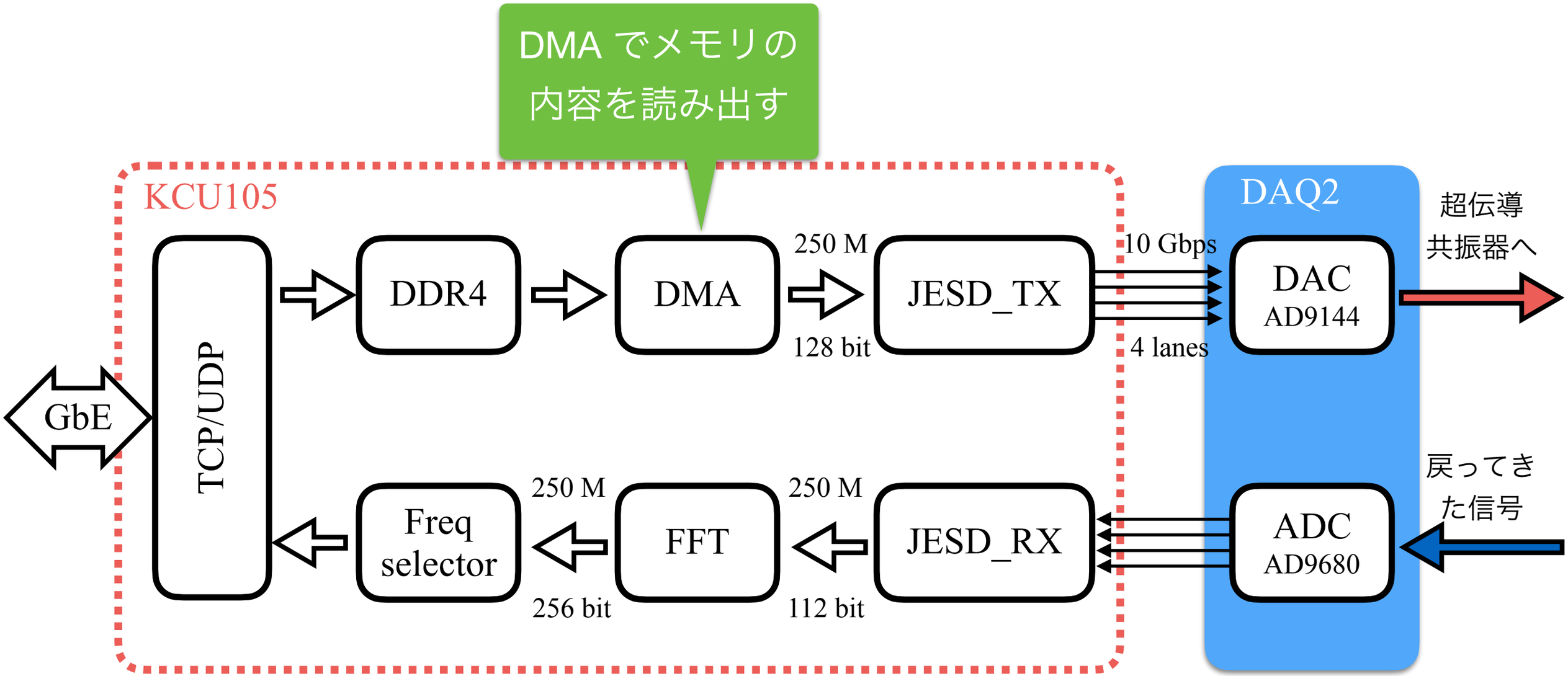
共振器そのものを検出器として利用

TES を 共振器に結合

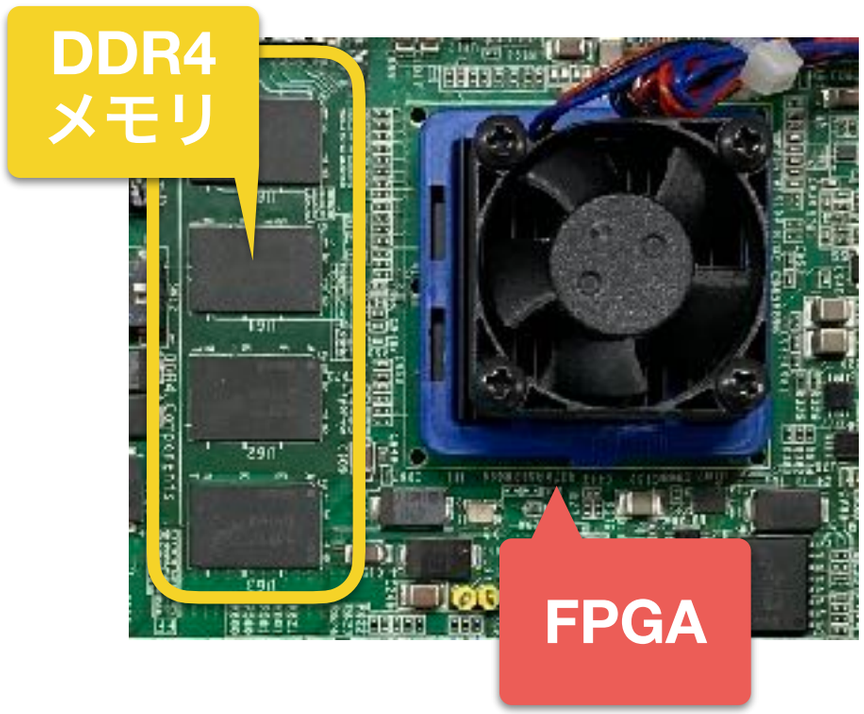


・ ~ GHz の共振器での多重化が主流

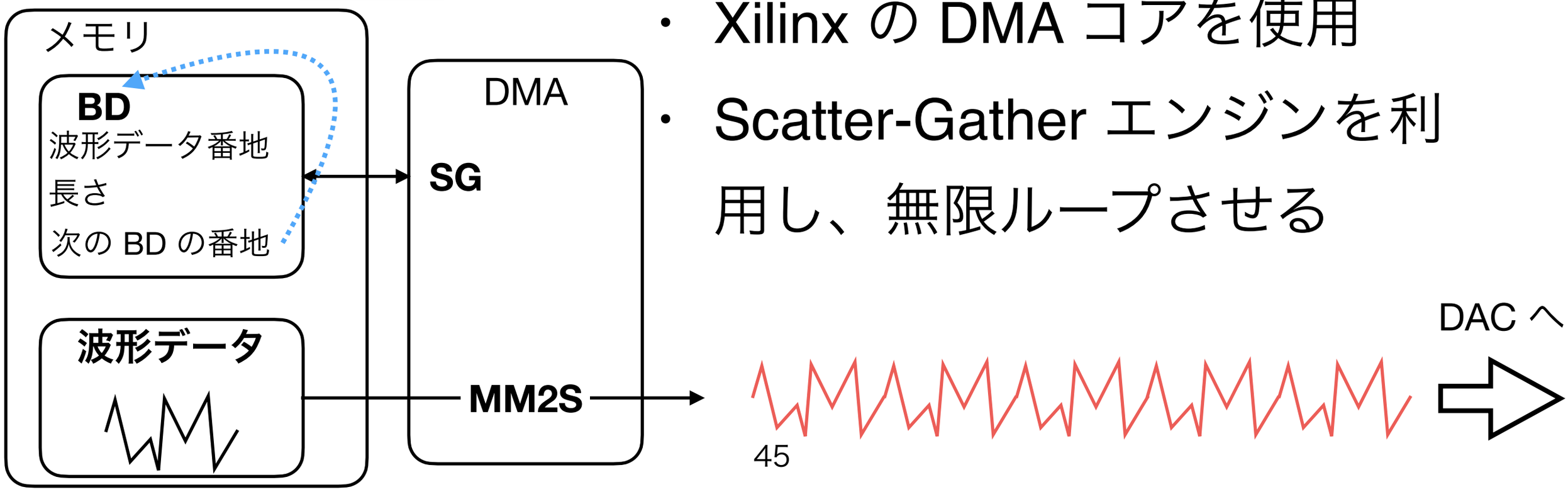
DMA


DMA でメモリの
内容を読み出す
KCU105
DAQ2
超伝導
共振器へ
GbE
TCP/UDP
DDR4
DMA
250 M
128 bit
JESD_TX
10 Gbps
4 lanes
DAC
AD9144
戻ってき
た信号
ADC
AD9680
JESD_RX
250 M
112 bit
FFT
250 M
256 bit
Freq
selector

DMA (Direct Memory Access)

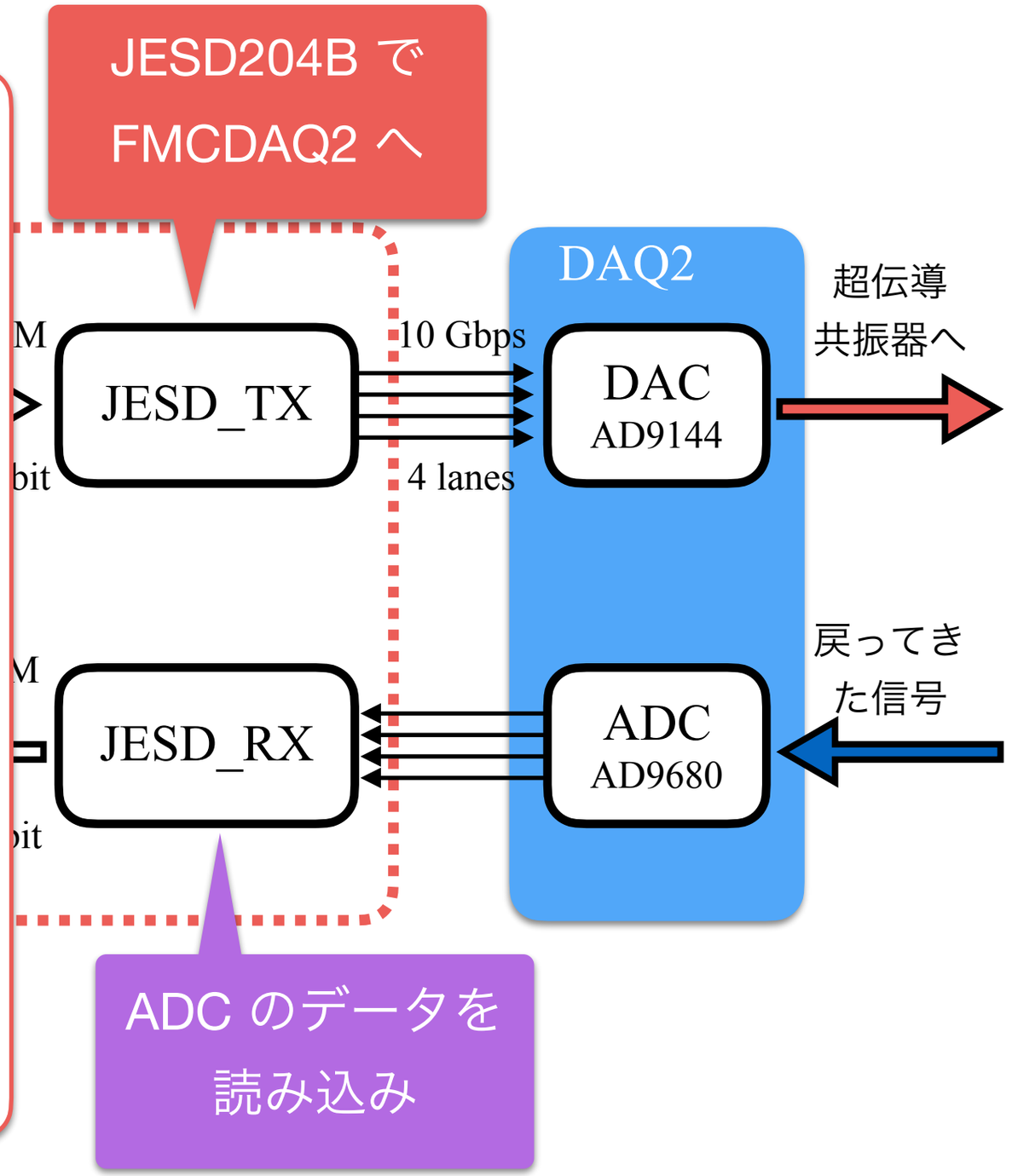


- FPGA ボード内の DDR4 メモリを使用
- 波形をメモリに書き込み、DMA コアで読み出す
- Xilinx の DMA コアを使用
- Scatter-Gather エンジンを利用し、無限ループさせる

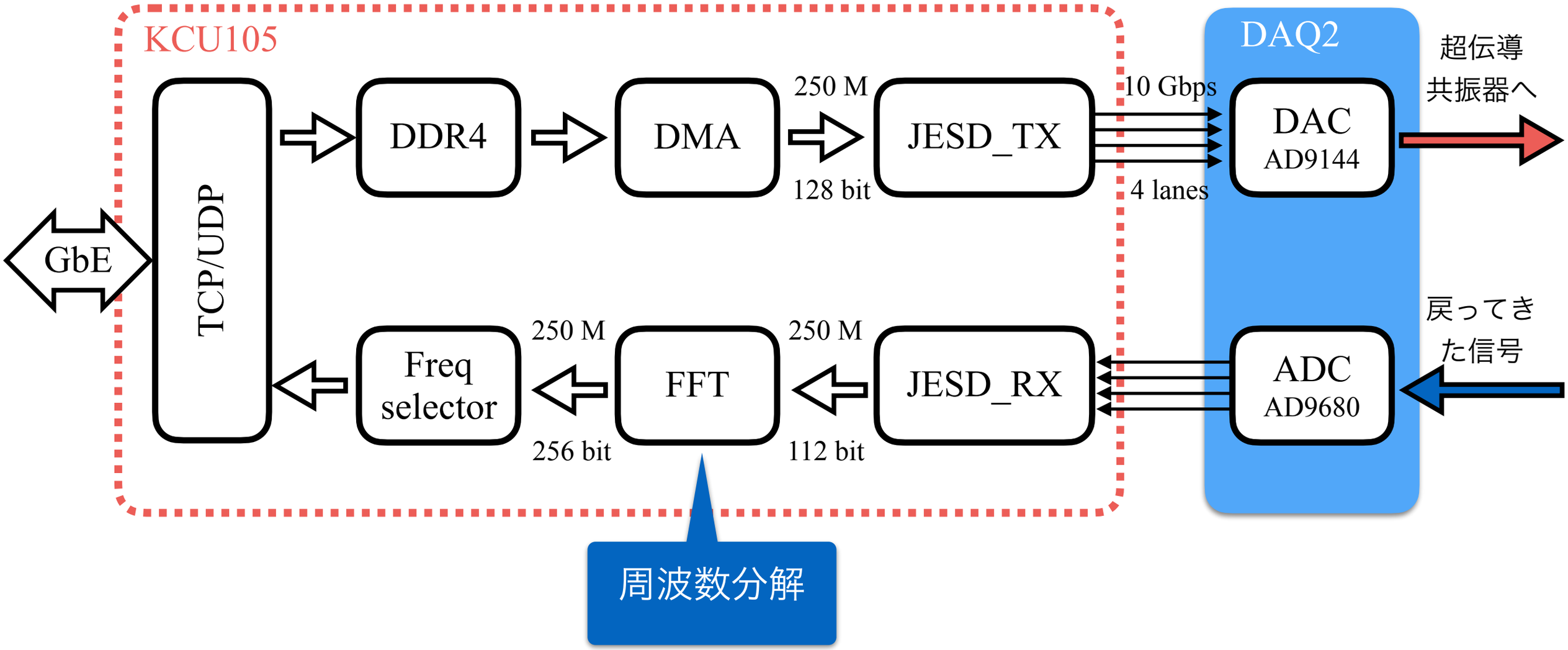


アナログボードとの通信

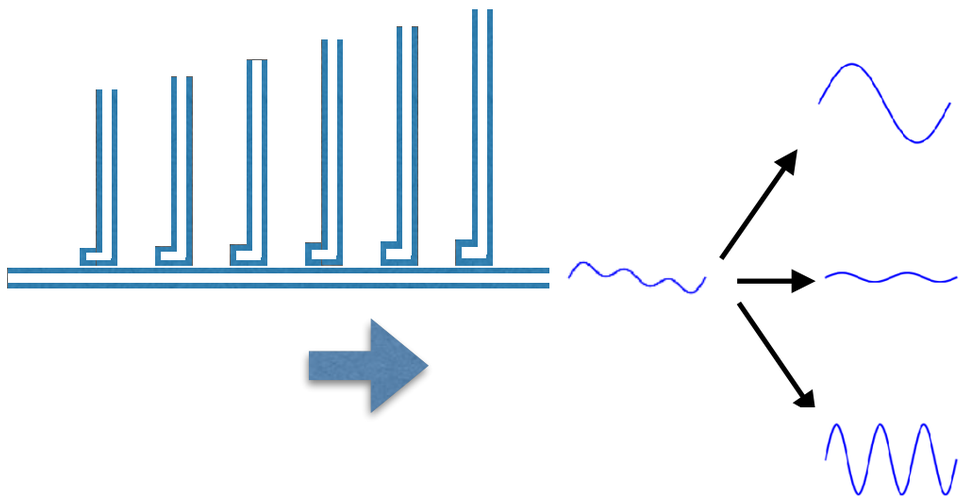
- JESD204: 高速シリアル通信
  - ADI の提供するコア (GPL) を用いた
  - FPGA 上では、GTH トランシーバを用いる
- チップを設定するための SPI 通信も



FFT


KCU105
GbE
TCP/UDP
DDR4
DMA
250 M
128 bit
JESD_TX
10 Gbps
4 lanes
DAQ2
DAC
AD9144
超伝導
共振器へ
戻ってき
た信号
ADC
AD9680
JESD_RX
250 M
112 bit
FFT
250 M
256 bit
Freq
selector
周波数分解

Fast Fourier transform (FFT)



- 合成波を周波数ごとに分解

  → **フーリエ変換**

- データレート: 1 GS/s

  … 4 レーン x 250 MHz

  $x_{4m}, x_{4m+1}, x_{4m+2}, x_{4m+3}$

直には扱えない

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i \frac{kn}{N}}$$

$$= \sum_{m=0}^{\frac{N}{4}-1} x_{4m} \cdot e^{-2\pi i \frac{k}{N} \cdot 4m} + \sum_{m=0}^{\frac{N}{4}-1} x_{4m+1} \cdot e^{-2\pi i \frac{k}{N} \cdot (4m+1)}$$

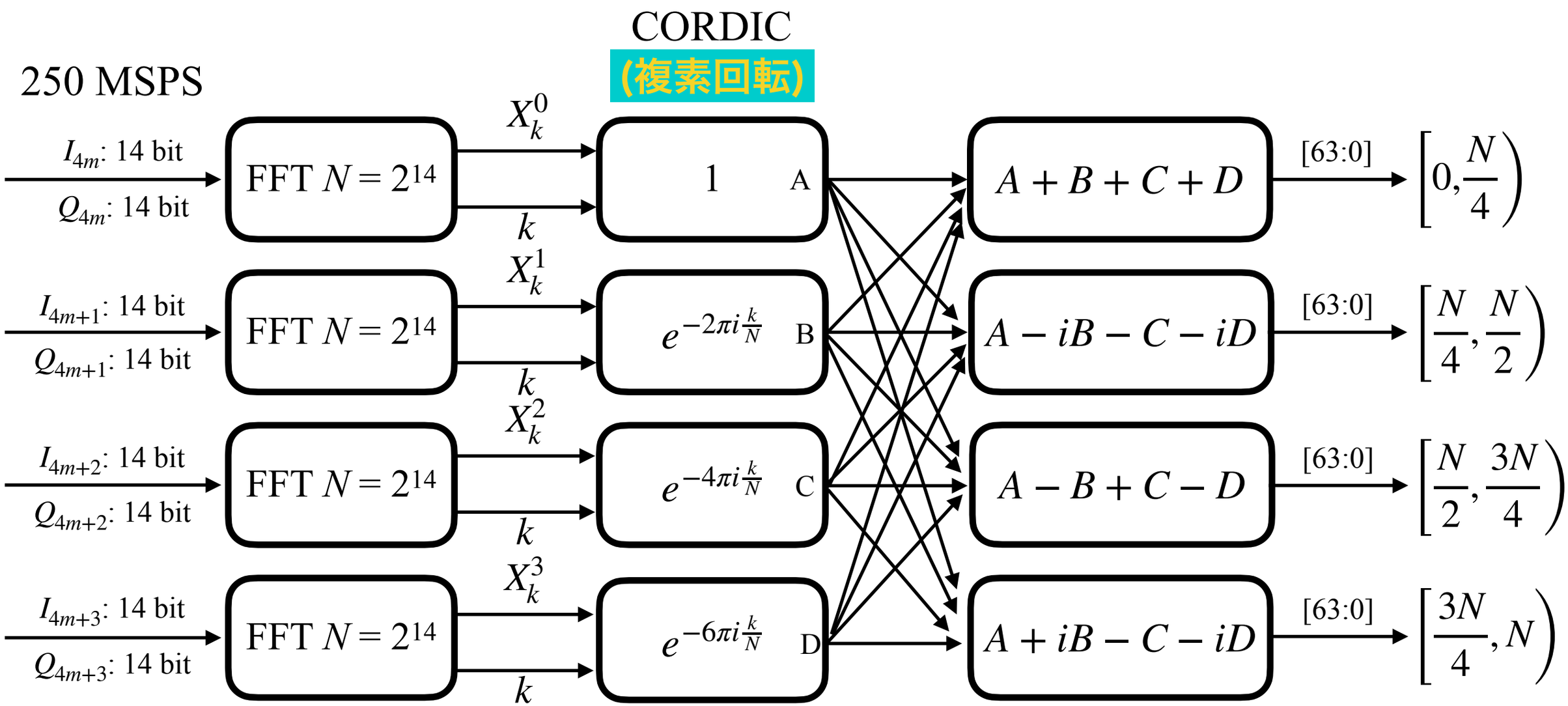
$$+ \sum_{m=0}^{\frac{N}{4}-1} x_{4m+2} \cdot e^{-2\pi i \frac{k}{N} \cdot (4m+2)} + \sum_{m=0}^{\frac{N}{4}-1} x_{4m+3} \cdot e^{-2\pi i \frac{k}{N} \cdot (4m+3)}$$

4 レーンに分割

$$e^{-2\pi i \frac{k}{N}} \sum_{m=0}^{\frac{N}{4}-1} x_{4m+1} \cdot e^{-2\pi i \frac{k}{N/4} \cdot m}$$

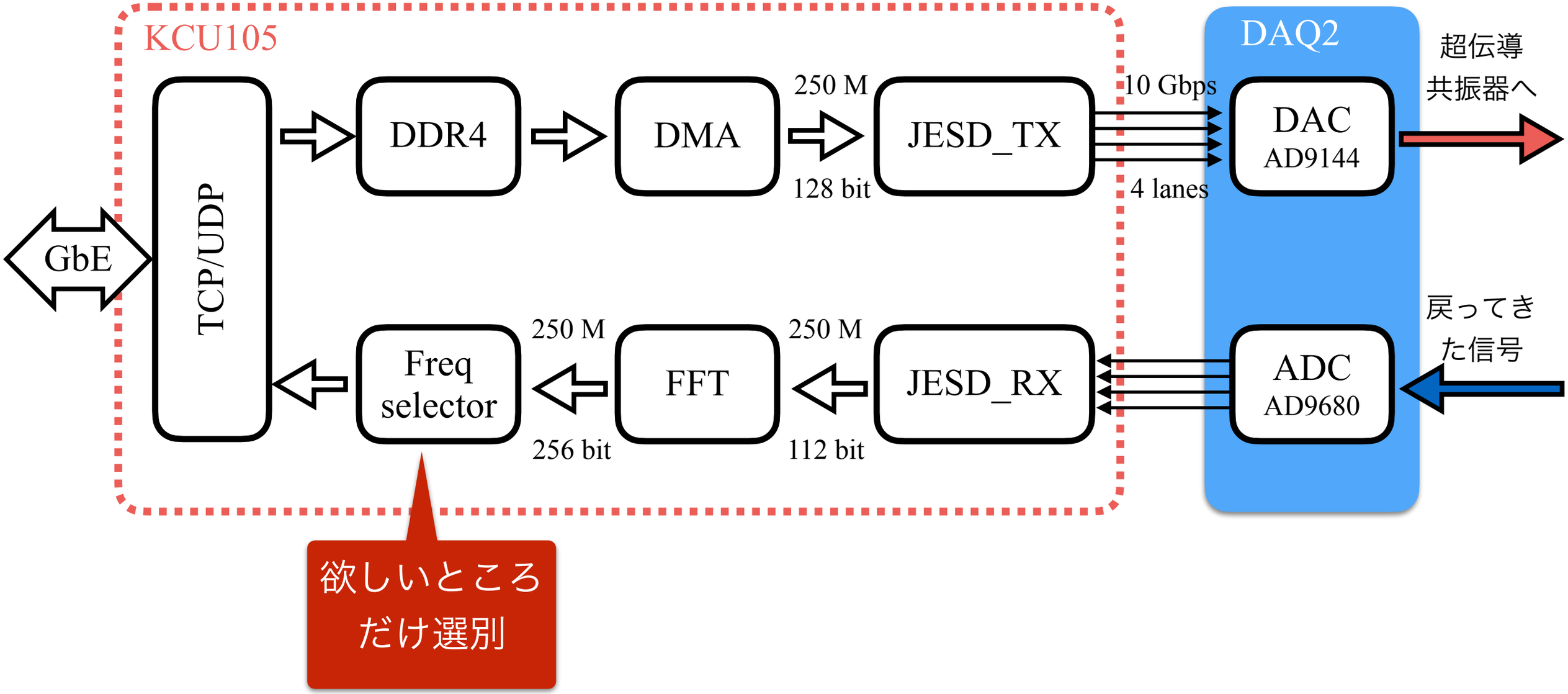
複素回転　　N/4 の長さの FFT

Fast Fourier transform (FFT)

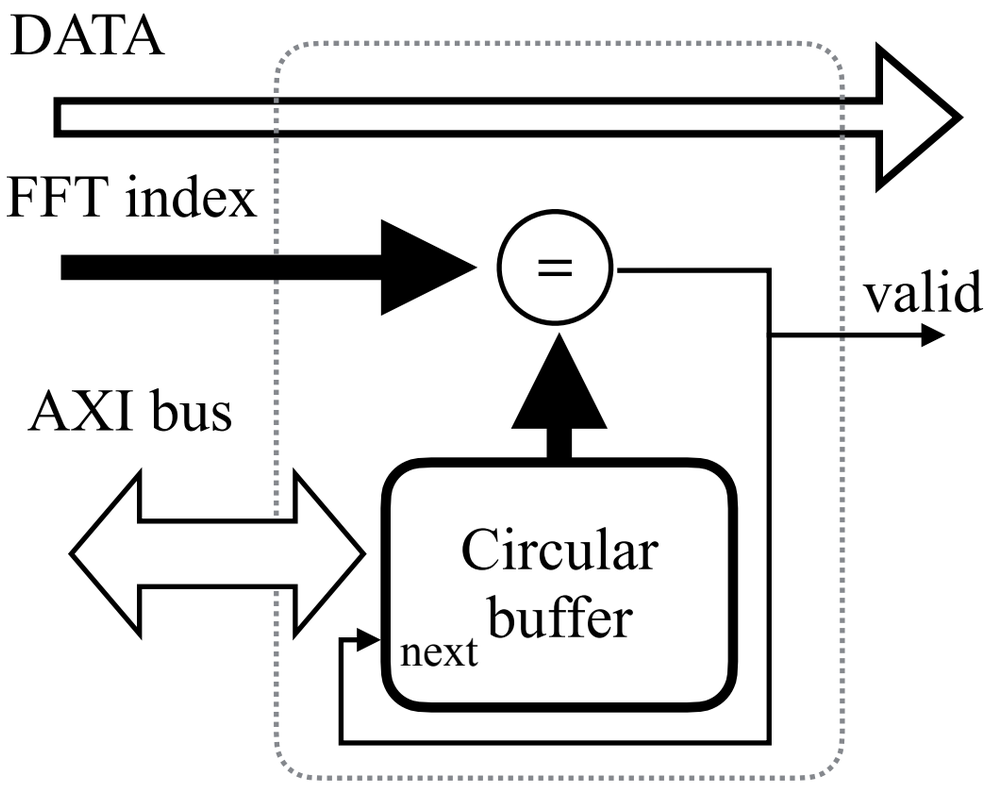


- 1 GSPS の信号は 250 MHz x 4 レーンに分割
- Xilinx の FFT と CORDIC を組み合わせて 2^{16} の 長さのFFT を実現

周波数セレクタ

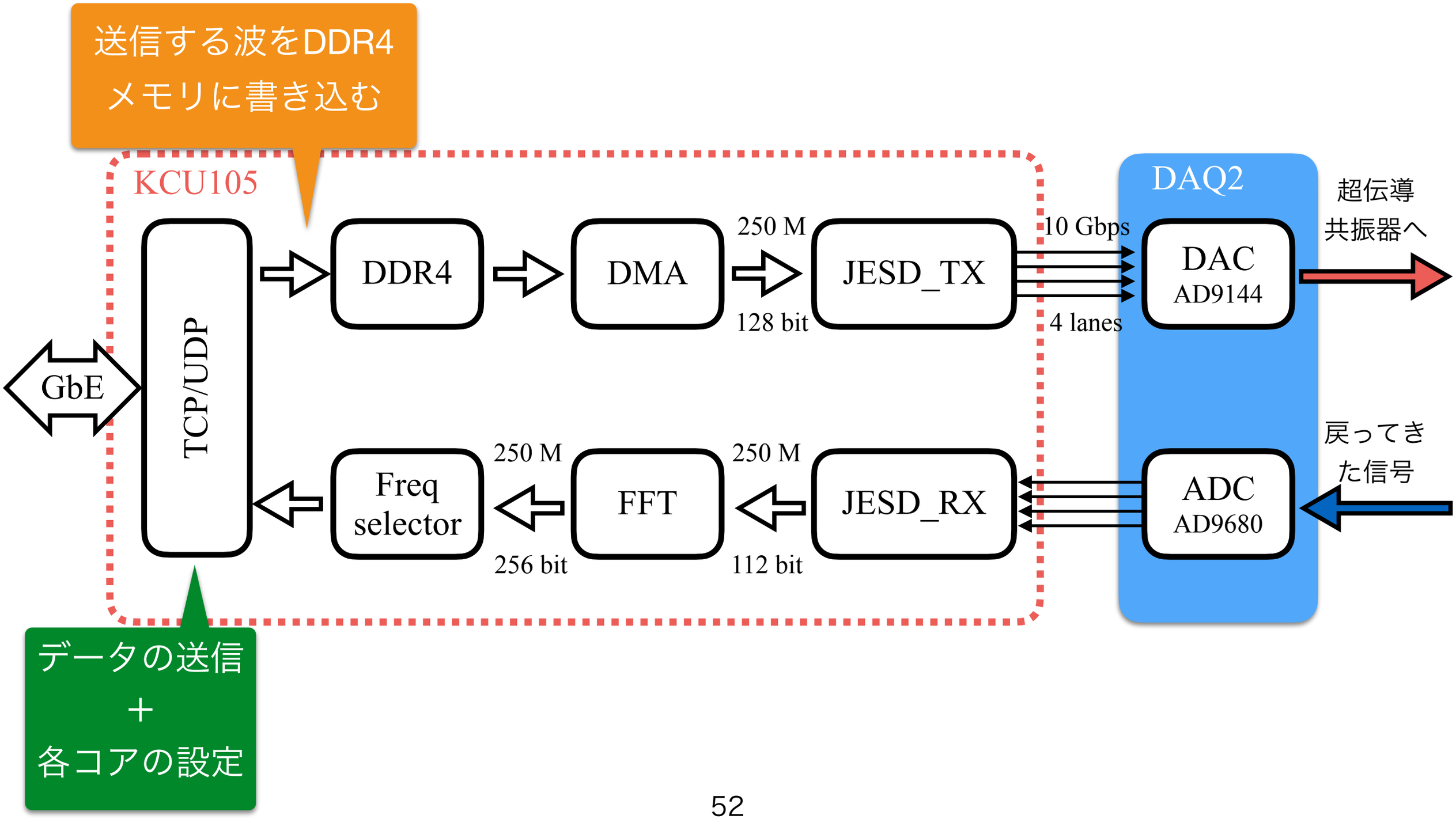

KCU105
GbE
TCP/UDP
DDR4
DMA
250 M
128 bit
JESD_TX
10 Gbps
4 lanes
DAQ2
DAC
AD9144
超伝導
共振器へ
戻ってき
た信号
ADC
AD9680
JESD_RX
250 M
112 bit
FFT
250 M
256 bit
Freq
selector
欲しいところ
だけ選別

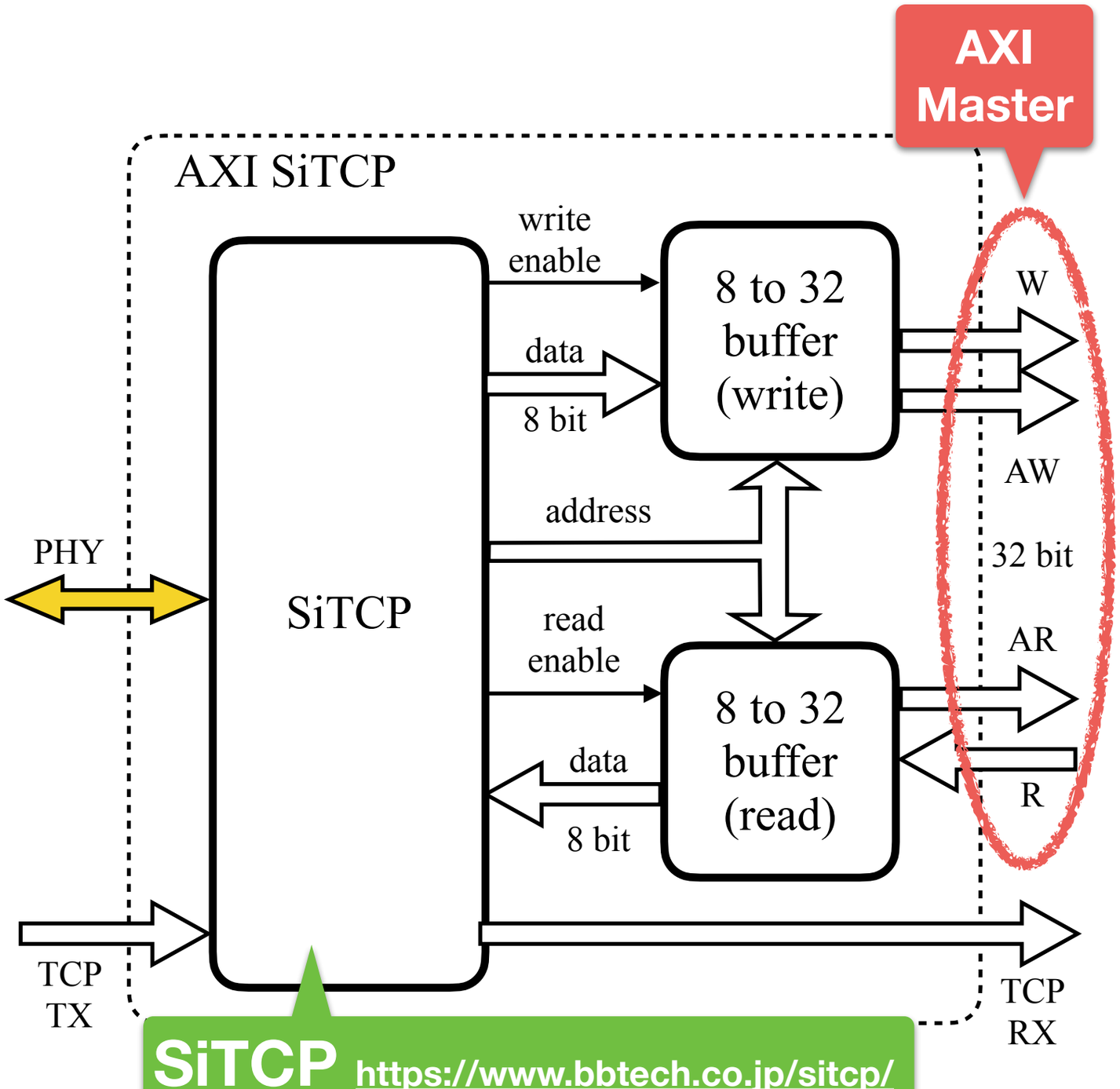
周波数セレクタ



- 取得したい周波数に対応する FFT の index を循環バッファに詰めておく
- 流れてくる FFT の index と比較して、一致していたら valid を出す
- AXI (バスの規格) で設定可能なモジュールを自作した

通信部分


送信する波をDDR4
メモリに書き込む
KCU105
DDR4
DMA
250 M
128 bit
JESD_TX
10 Gbps
4 lanes
DAQ2
DAC
AD9144
超伝導
共振器へ
GbE
TCP/UDP
Freq
selector
250 M
256 bit
FFT
250 M
112 bit
JESD_RX
ADC
AD9680
戻ってき
た信号
データの送信
＋
各コアの設定



AXI SiTCP

https://github.com/dixilo/axi_sitcp

- 各 IP コア (モジュール)設定 … AXI バスを利用
- SiTCP のスローコントロール用のバス (RBCP) を、 AXI 化する wrapper を作成した

できること

- AXI バスを持つ IP コアの設定など
- メモリへの書き込み

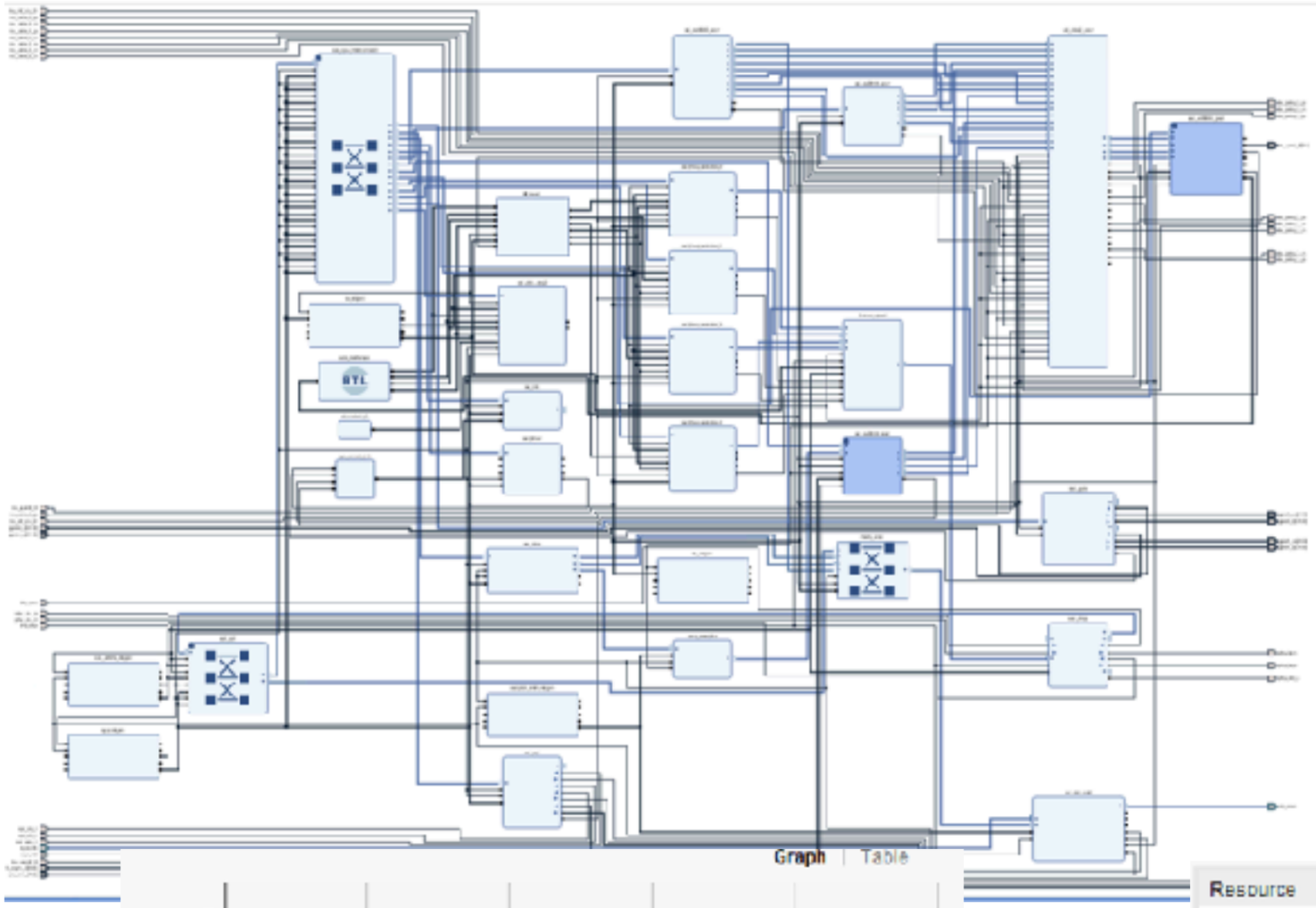
ごりやく

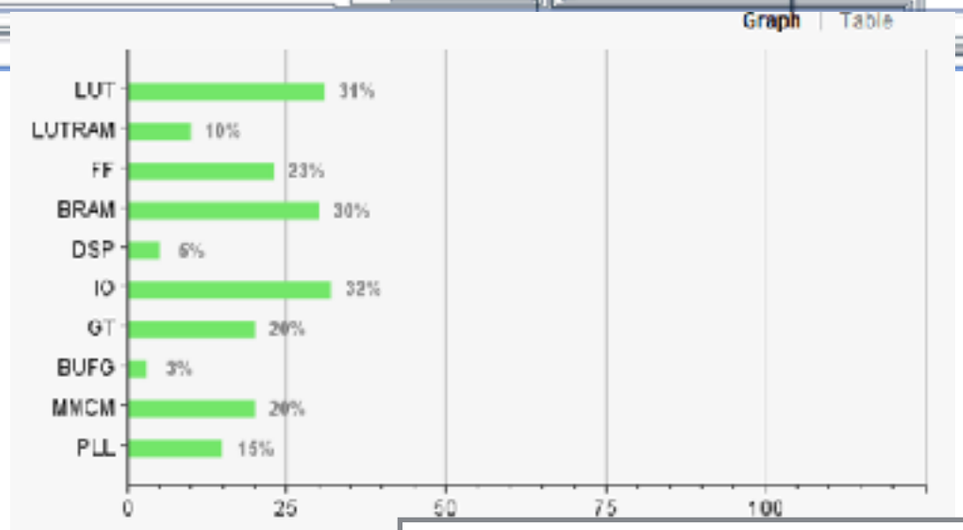
- Xilinx が提供している AXI バスのインターコネクトがそのまま使える

SiTCP https://www.bbtech.co.jp/sitcp/

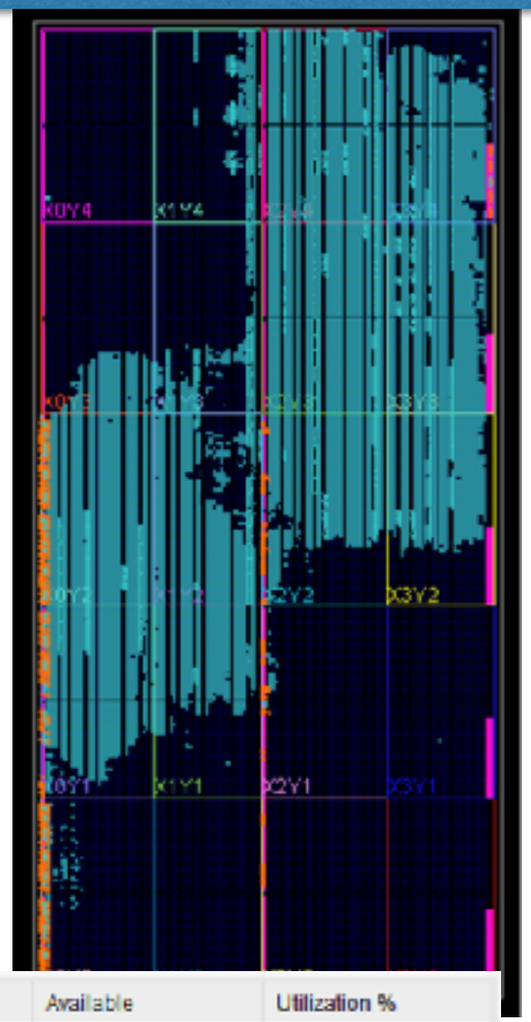
- FPGA をネットワークに接続
- TCP で 1 Gbps までのデータ転送が可能
- UDP でスローコントロール

Vivado のブロックデザイン





ロジックセルの利用状況



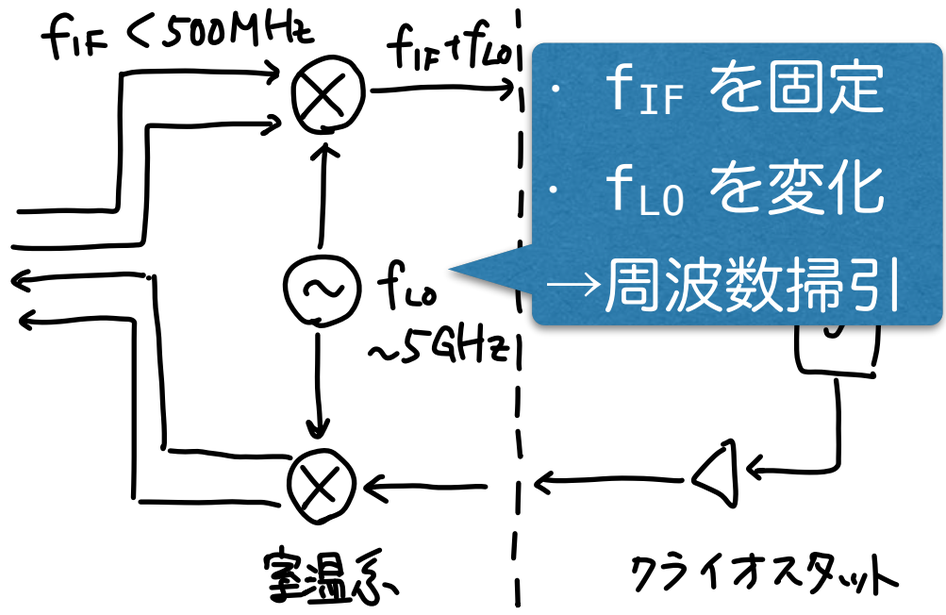
Resource	Utilization	Available	Utilization %
LUT	74271	242400	30.64
LUTRAM	11599	112800	10.28
FF	110427	484800	22.78
BRAM	182.50	600	30.42
DSP	103	1920	5.36
IO	168	520	32.31
GT	4	20	20.00
BUFG	14	480	2.92
MMCM	2	10	20.00
PLL	3	20	15.00

ひととおり完成。
まだリソースに余裕があり、パワーアップできそう

テスト

- 実際に共振器を見てみた

東北大からお借りしている冷凍機

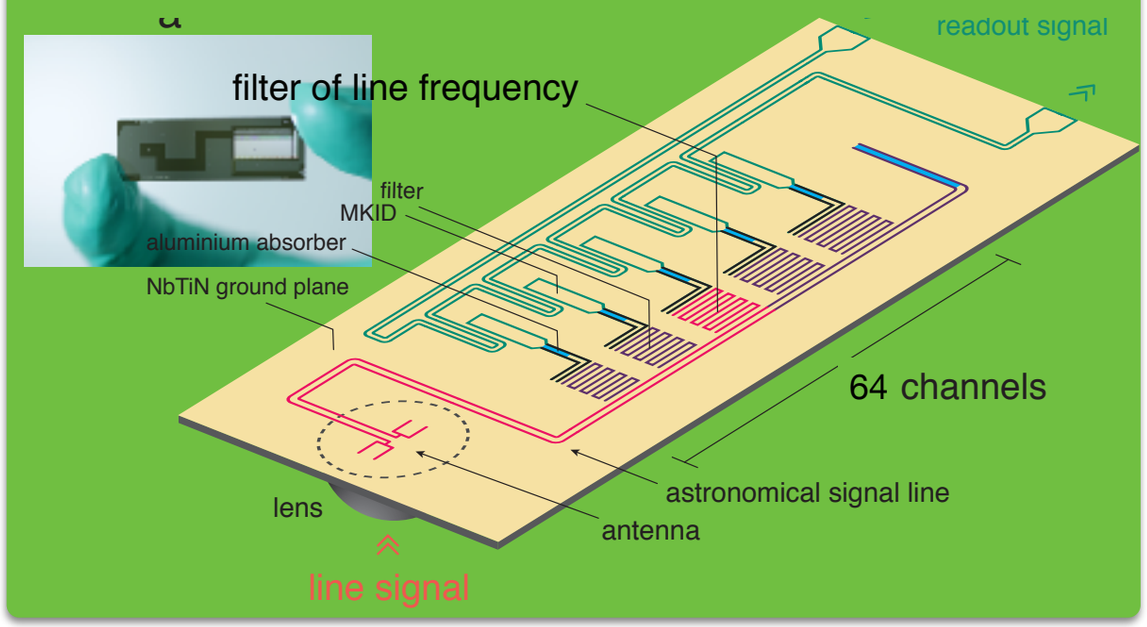


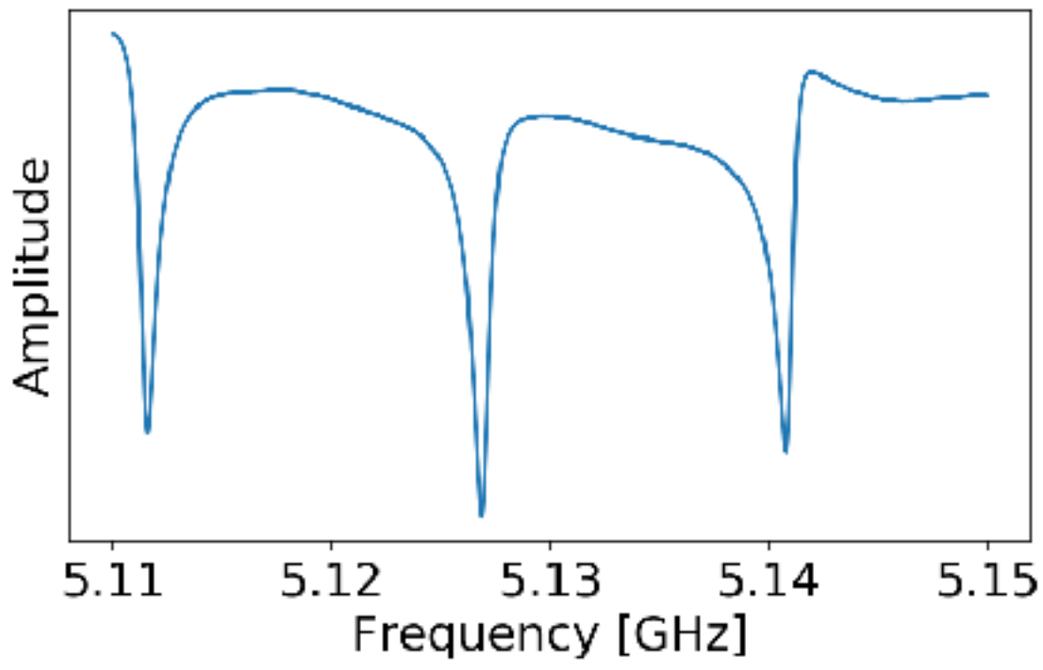
- f_{IF} を固定
- f_{LO} を変化

→周波数掃引

中に入っているチップ:

64ch filterbank by TUDelft, SRON





ちゃんと共振器が見えた！

AXI SiTCP

- 動作
  - 書き込み: 32 bit まとめて送る
    - 32 bitまとめる前にアドレスがワード区切りを超えた場合、その時点で送る
  - 読み込み: 32 bit まとめて読む
    - 読むアドレスが 32 bit 以内の場合、バッファの中から SiTCP にデータを渡す
    - 4 バイト読み終わったら、次はバッファを使わずスレーブにアクセスしてバッファを更新する
    - 4 バイト読み終わる前にワード境界を超えた場合も、スレーブにアクセスしてバッファを更新する
- DDR4 メモリへの read/write スピード: ~ 1MB/s

```
(base) PS C:\Users\kucmb\jsuzuki\test\pytest> python .\speed_test.py
====== Test start ======
==== DDR4 ====
== Write start ==
Time:  0:00:04.211201
== Write end ==
== Read start ==
Time:  0:00:04.194610
== Read end ==
== Consistency ==
check: True
====== Test end ======
```

FFT

- FPGA のロジックは 1 GHz では動かない

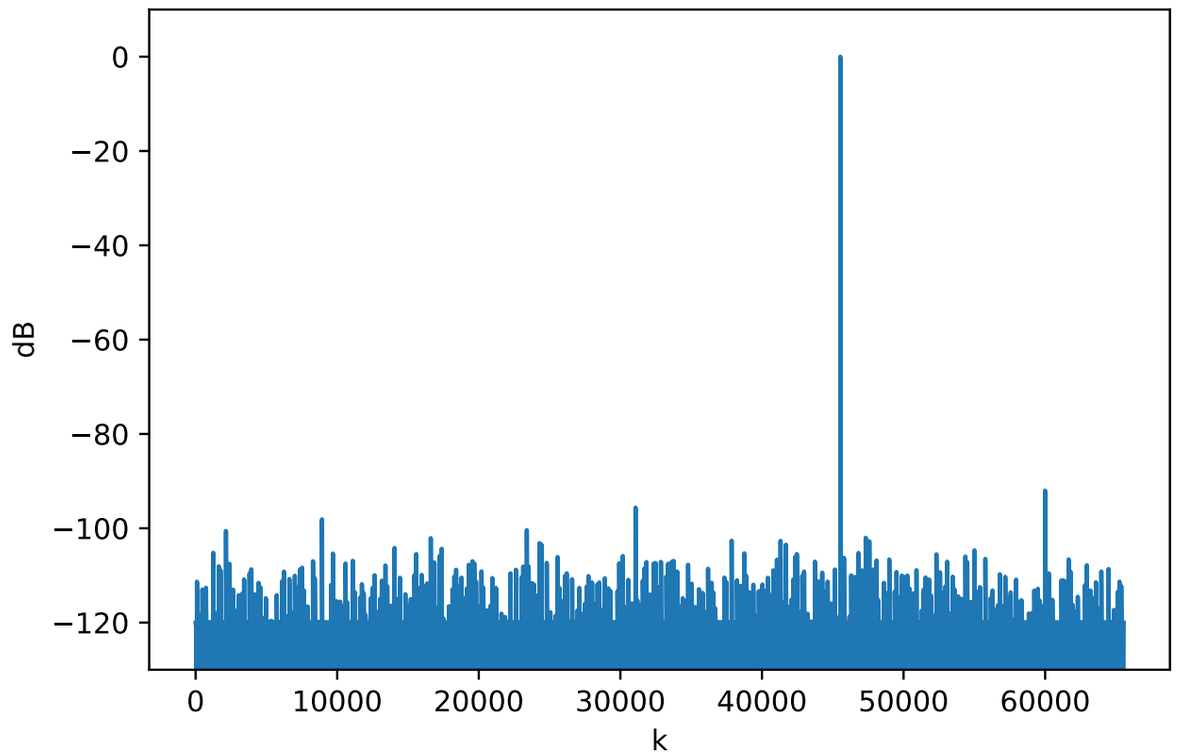
  →分割

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i \frac{kn}{N}} = \sum_{m=0}^{\frac{N}{2}-1} x_{2m} \cdot e^{-2\pi i \frac{k}{N} \cdot 2m} + \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} \cdot e^{-2\pi i \frac{k}{N} \cdot (2m+1)}$$

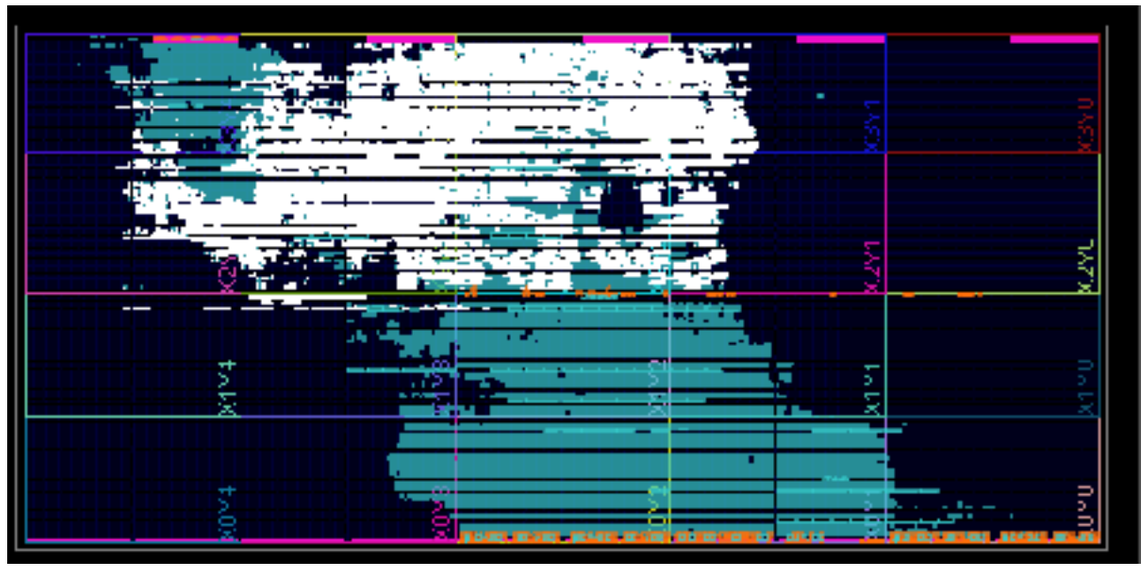
$$= \sum_{m=0}^{\frac{N}{2}-1} x_{2m} \cdot e^{-2\pi i \frac{k}{N} \cdot 2m} + e^{-2\pi i \cdot \frac{k}{N}} \sum_{m}^{\frac{N}{2}-1} x_{2m+1} \cdot e^{-2\pi i \cdot \frac{k}{N} \cdot 2m}$$

FFT

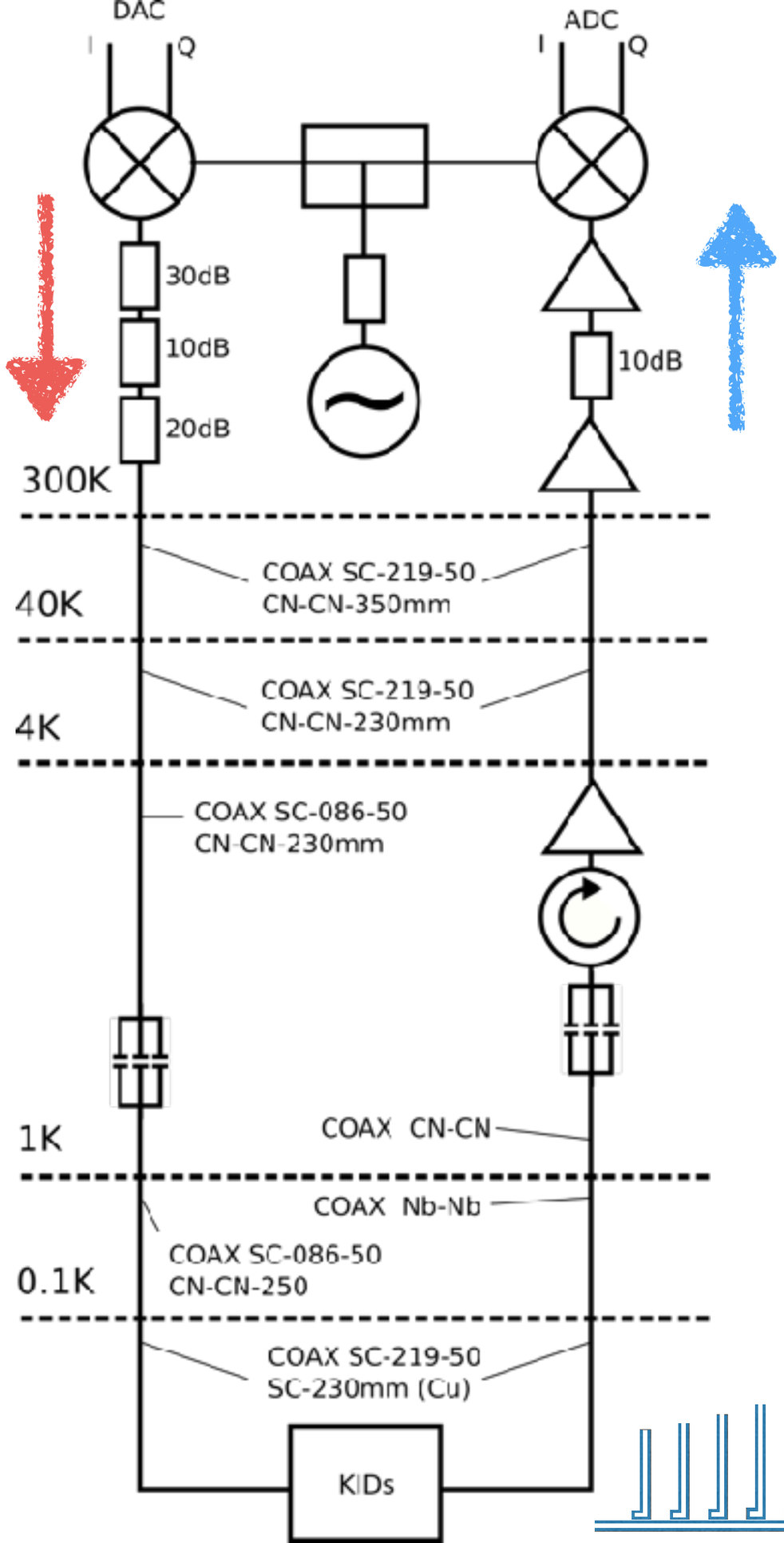
- (Vivado の)シミュレーション上で動くことを確認



- KCU105 に実装できた



FPGA 上で使われているロジックの分布。色が塗られている箇所は使われている。白色が FFT に該当する部分



Up/down conversion

- MKID (or rf-SQUID) usually has 4 — 8 GHz resonance frequencies
- Ordinary DAC or ADC cannot deal with RF frequency directly
  => Mixing

ソフトウェア

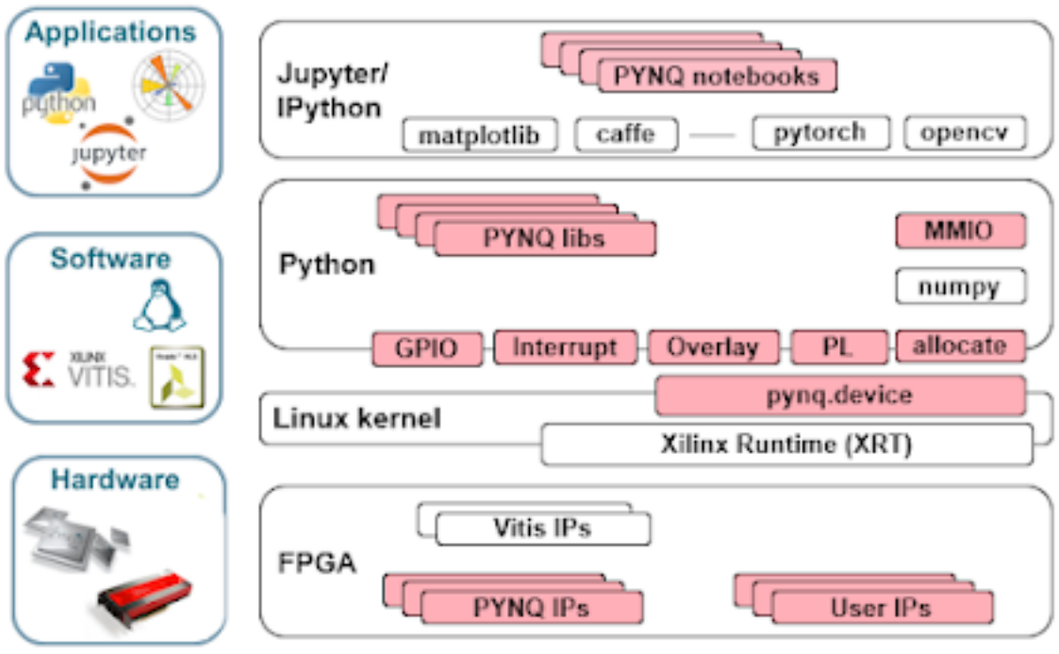
- Python を利用した
  - SiTCP を動かすところには、 sitcpy を利用

    https://github.com/BeeBeansTechnologies/sitcpy

  - 元々 Microblaze や Zynq 用に書かれていた C のソフトウェアを、 python に移植した

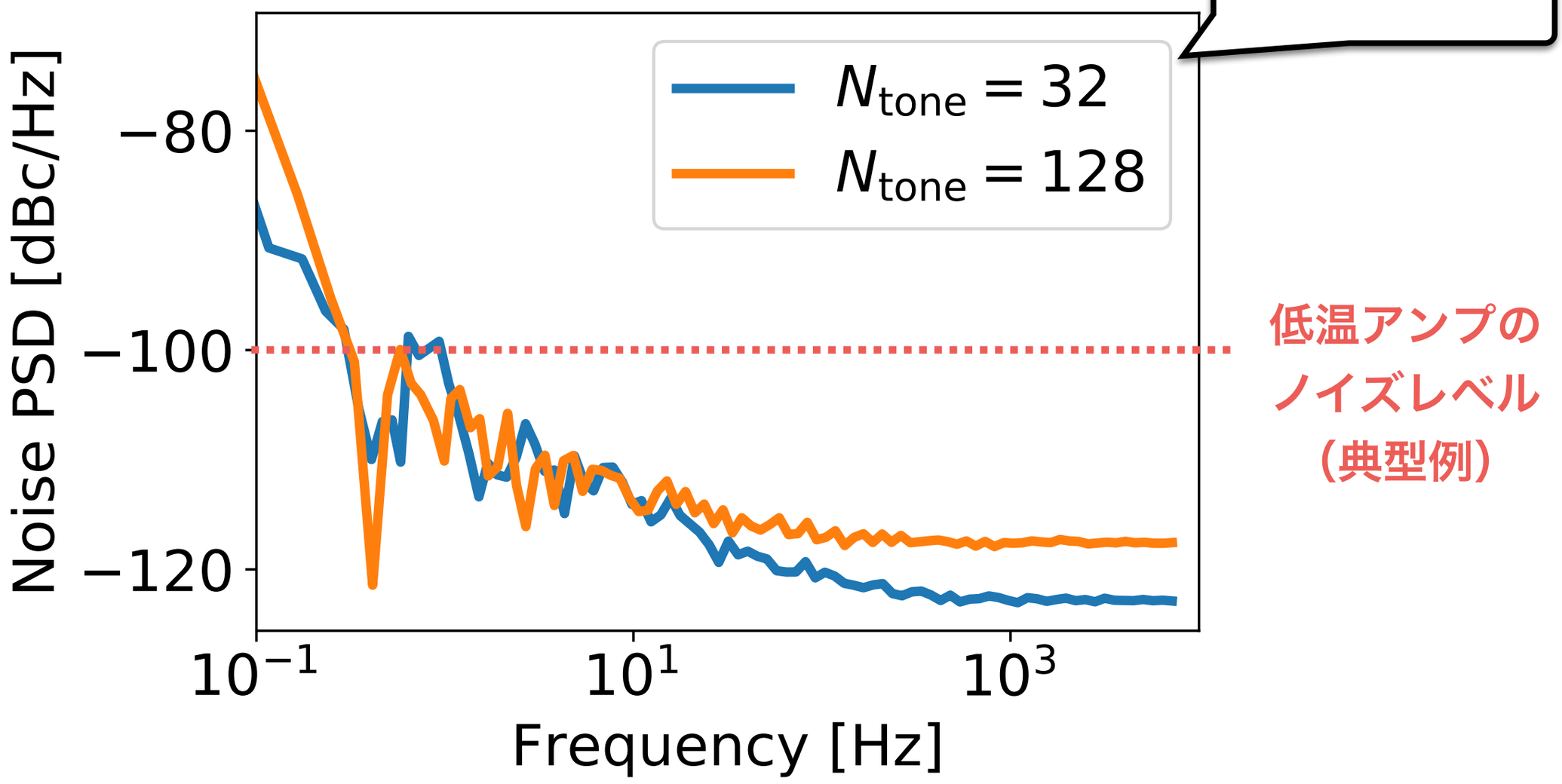
PYNQ

- Xilinx によるオープンソースプロジェクト
- ZYNQ の中で Linux を動かし、Jupyter lab などで FPGA が動かせる



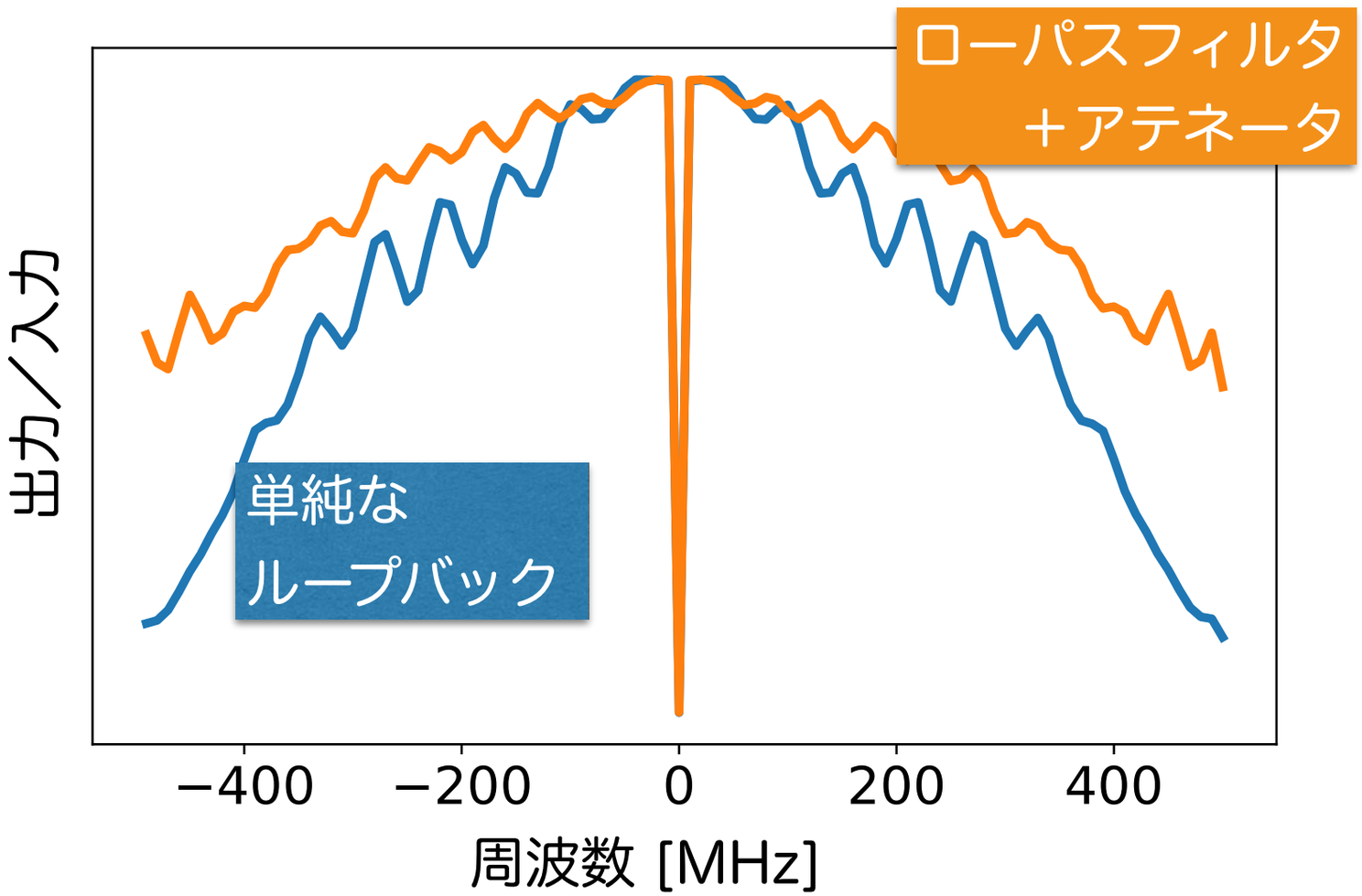
- RFSoC 2x2 用の PYNQ image が用意されており、今回はそれに自分で作った bitstream を読み込ませて動作させた

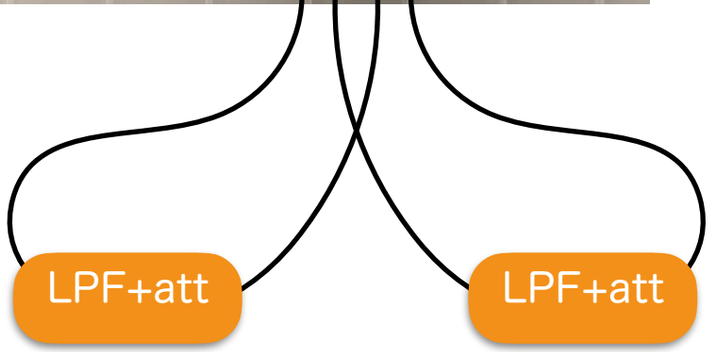
ノイズ特性



- アンプのノイズレベルよりも十分低く抑えられている
- N_{tone} 依存性も予想どおり

周波数特性





- 高周波での低下　→　ローパスフィルタ
- 波うち　→　アテネータ、アンプなど　で対応可能

＝ボードやファームウェアは問題なさそう😄