# FPGA を用いた様々な応用

**基礎物理学研究室**

2022 年 2 月 12 日

京都教育大学 理科領域専攻
学籍番号 181187

安澤　大輔

## 背景

近年，電子デバイスの高性能化が進み，さらに廉価版の製品も多く出てきている．特に FPGA と呼ばれる大規模なディジタル処理回路装置は演算の速度や容量が向上するとともに低価格化が進み，ここ最近では，容量が数百倍になり価格は 100 分の 1 から 10 分の 1 になっている．また，高速で動作する並列処理が必要な用途でよく使われるようになり，放射線計測分野においても研究用途で FPGA を利用する事例が多く存在している．この状況から，より低コストに放射線計測のためのデジタル処理装置を構築することができる．

しかし，FPGA も相対的に安くなっているが，研究用となればその価格は上昇し，数十万円のものまで存在している．さらに，大きな大学や研究所では一つの高価な FPGA を使いこなすことで，測定に限らず，デジタルデータ通信やレーダー装置など幅広い研究が行われている．

そこで，本研究では半導体検出器と FPGA をはじめとする安価な電子機器を用いて放射線のスペクトルを解析する装置の作成を試みる．

# 目次


1 放射線の測定について 3
1.1 放射線とは . . . 3
1.2 放射線測定器 . . . 3
1.3 放射線検出機構 . . . 3

2 半導体検出器 5
2.1 半導体のエネルギー . . . 5
2.2 n 型と p 型半導体 . . . 6
2.3 検出器としての pn 接合 . . . 8

3 FPGA について 10
3.1 Kintex ® -7 FPGA KC705 . . . 10
3.2 FPGA の利用 . . . 11

4 AD 変換 12
4.1 アナログ信号とデジタル信号 . . . 12
4.2 AD 変換 . . . 12

5 AD9467 14
5.1 AD9467 の設定 . . . 14
5.2 外部クロックの設定 . . . 15

6 実機を用いたサンプリング 16
6.1 AD9647 を用いた電圧の標本化検証 . . . 16
6.2 半導体検出器からのサンプリング . . . 18

7 AD 変換器の回路素子 20

8 まとめ 22
8.1 まとめ . . . 22
8.2 謝辞 . . . 22

9 プログラムコード 23

# 1　放射線の測定について

## 1.1　放射線とは

放射線とは 1900 年頃までは電磁波を意味する術語として用いられていたが，20 世紀に入り電子・X 線・自然放射線が発見され，これらも放射線の術語に含まれる事となった．そのため，放射線には複数の種類がある．これまで電磁波は波動として扱われていたが，新しく発見された放射線は粒子の性質を示す．これは 1920 年代に物質は二面性理論が作られてからまもなく，電子の回折実験からこの理論が確かめられ，以降粒子性と波動性の違いは重要では無くなった．

以上のように放射線には複数の種類があるが，分類する方法として電離性放射線か非電離性放射線の区別がある．ここでいう電離性とは，放射線が透過する媒質や物質中の原子または分子を電離する能力があることを意味している．約 10nm より長い波長を持つ電磁波は非電離性放射線である．非電離性放射線には電波・赤外線・可視光・紫外線の一部が含まれる．一方，電離放射線にはそれ以外の電磁波 (X 線やガンマ線など) が当てはまる．そのほか，電子線・陽電子・陽子・アルファ粒子・中性子・重イオン・中間子のような原子粒子と原子構成粒子の全ても含まれる．

## 1.2　放射線測定器

放射線測定器は動作によって，パルス型放射線測定装置と電流型放射線測定装置の二種類に分類できる．パルス型放射線検出器の一般的な機構を図に示す．本論文では実験でも使用したパルス型について述べる．

パルス型検出器にも，ガス入り・シンチレーション・半導体・放電箱・泡箔など複数の種類がある．検出器の機能は検出器に粒子が入射する度に信号を発生することである．検出器はすべて物質と粒子の相互作用を利用して動作する．ほとんどの検出器の出力は電圧パルスである．色の変化や飛跡が信号となる場合もある．

理想的なパルス型検出器には計測条件があり，

- 検出器に粒子が入射する度に利用する電子回路のノイズよりも大きなパルスを発生させる．
- パルスの幅は短く，次々に入射する粒子が別々のパスるを作る必要がある．
- 粒子のエネルギーを測定する場合，パルスの波高と粒子のエネルギーの間にある定まった関係がある．
- 検出器に同じエネルギーを付与した各粒子に対して，検出器は常に同じ波高のパルスを出力する必要がある．

と現すことができるが，すべてを満たす検出器は存在しない．実際ガンマ線と中性子の検出器についてはそのエネルギーをすべて検出器に付与することは不可能である．そのため，できるだけこれらの条件を満たす検出器を選んで測定を行い，データに適切な補正を加えるしかない．

## 1.3　放射線検出機構

放射線の検出には前章の通り複数の電子回路装置が必要となってくる．例えば，前置増幅器や主増幅器等のアンプと呼ばれる装置がある．検出器から出る信号は数 mV 程度の非常に低い信号であるため，信号を測定可能な大きさにするには 1000 倍以上もの増幅が必要となる場合もある．しかし，増幅器に接続するためには伝送線が必要で，これを通る間にも信号の減衰はおきている．検出器の出力が小さい場合は特に電子回路ノイ

ズに必要信号が埋没してしまう可能性もある．そのため，検出器のできるだけ近くに前増幅器を置くことでこれを避ける働きがある．さらに，前増幅器は信号を成形し，検出器のインピーダンスと主増幅器のインピーダンスを整合させて減衰を防ぐ働きもある．

増幅器を通過した信号に対して，不必要なパルスを取り除くための装置もある．これは波高弁別器やシングルチャネル波高分析器と呼ばれ，電子回路の雑音を除去することで必要な信号を集中して計測することができる．ある波高以上のパルスだけ，つまりある閾エネルギー値を持つ粒子を計測したい場合があるとする．この場合，その波高に満たないパルスは計数してほしくない．波高弁別器やシングルチャネル波高分析器は必要なパルスを選択する装置とも言える．これに対して，パルスの波高に応じてパルスを計測するマルチチャネル波高分析器もある．

また，半導体検出器から出力される信号のすべてはアナログ信号であり，これを解析するためにはコンピュータを要するが，間にデジタル信号に変換する機器も必要となる．アナログデジタル変換器 (ADC) にも複数の形態があり，理科教育の実験でよく見るのはディスプレイと一体型になったオシロスコープがある．

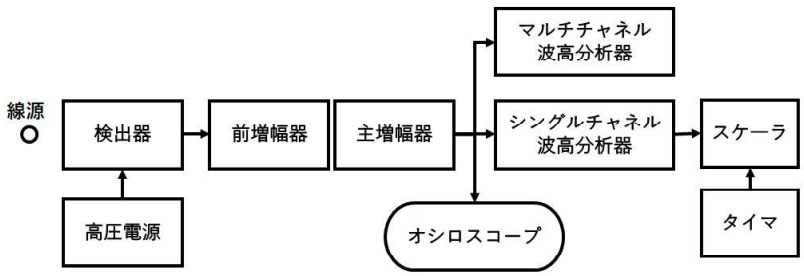


図 1　測定機構一般概略

# 2　半導体検出器

半導体を用いた放射線検出器のことで，本研究において検出器として用いたので説明する．動作としては半導体内に電離放射線が通過するときエネルギーを消費して正孔と電子の対が生成され，この電子が半導体内を移動することで外部に信号が出力される．半導体検出器に用いられる金属としては Si や Li の他 CdTe や HgI2 などがある．

他の放射線検出器と比べた時，半導体検出器の優れている点はエネルギー分解能である．エネルギー分解能とは多数のエネルギーを識別する能力である．その他の利点は以下にあげられる．

- 広い範囲のエネルギーに対して，パルス波高対放射線エネルギーの応答性がよい．
- ガス検出器に比べて密度の高い物質を利用しているので，検出器寸法に対する検出効率がよい．
- 特殊な形状の検出器を作ることができる．
- ガス検出器に比べてパルスの立ち上がり時間が早い．
- 真空中で動作させることができる．
- 磁場の影響を受けにくい．

半導体検出器の特性は仕様金属に依存する他，検出器の形状や表面の処理方法にも依存する．

## 2.1　半導体のエネルギー

半導体中では価電子帯は満たされており伝導帯は空であるが，この二つの帯の間の禁止帯は非常に小さい．0K に近い温度では半導体の電気伝導度は 0 であり，エネルギー帯図は絶縁物の場合と同じようになる．しかし，温度を上昇させるとフェルミ分布のすそが伸びて伝導帯中に電子のいくつかを上げるので電気伝導度が増す．電子がエネルギーを受け取ると一つ上の伝導帯に上がる．このとき，価電子帯には電子の抜けた状態が残る．これが正孔 (ホール，hole) と呼ばれるものである．正孔は-(-e)=+e であるので正の電荷を持つ粒子として取り扱われる．正孔は電子と同様に電気伝導に寄与する．

電子にエネルギーを与えるのは熱に限らない．放射線の吸収やあるいは高エネルギーの荷電粒子との衝突でも同様の事が起きる．

高いエネルギーを持った入射荷電粒子が半導体中の電子と衝突して，価電子帯から伝導体に電子をあげるに止まらず，もっと深い充満帯からも電子を伝導帯にあげる．すなわち，荷電粒子が入射する前までは空であった伝導帯中にも電子が現れ，電子が充満していた充満帯には正孔が現れる事になる．しかし，この状況は長く続かない．$10^{-12}$ 秒程度の時間内に電子と正孔の相互作用によって電子は一番低いバンドの底に集まる．逆に，正孔はもっと高い充満帯の頂部に集まる．この遷移過程の間に，ずっと多数の電子と正孔が作られる．この多段過程があるので一個の電子正孔対をつくるのに必要なエネルギーは禁止帯の幅のエネルギーよりもおおきくなる．例えば，室温でのシリコンの禁止体エネルギー $E_g$ は 1.106eV であるのに対して，電子正孔対をつくる平均エネルギーは 3.36eV である．

電界が印加されていない場合は，遷移過程の最終段階では電子と正孔が再結合して結晶が放射線入射前の状態に戻ることになる．

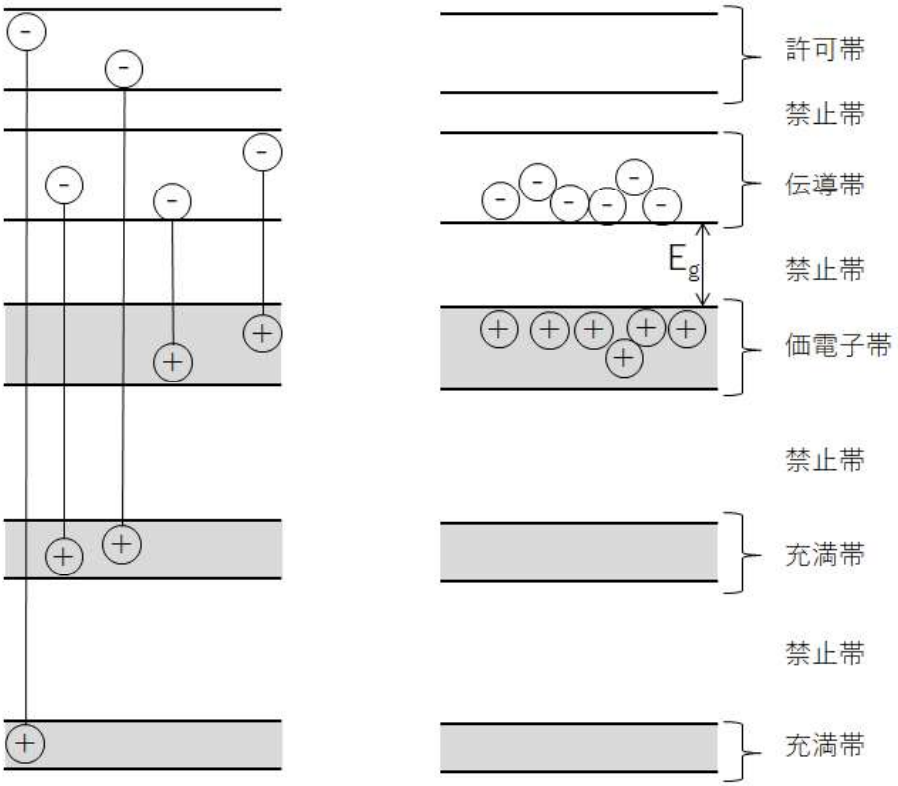


図 2　電子正孔対の遷移とエネルギー帯

## 2.2　n 型と p 型半導体

不純物を添加すると高純度半導体の性質は変化する．不純物が存在する場合，新しい状態が作られ，半導体は余分の電子/正孔を持つようになり，物質の電気伝導度は増す．

実際不純物が一切入ってない半導体は存在せず，純度の高い半導体を真正半導体と呼び，不純物を意図的に含んでいる半導体を不純物半導体と呼ぶ．ほとんどの場合，ドーピングと言われる方法によって制御された量の不純物を添加して物質の伝導度を数桁増やしている．

例えば，Si からなる真正半導体は価電子はすべて隣り合った原子と共有結合している．これに対して 5 個の価電子を持つ As を添加したとする．Si にある 4 個の価電子と結合できるのは 4 個までなので 1 個の電子はどの化学結合にも属さない．この電子は非常に弱く束縛されており，この電子を伝導帯にあげて自由電子にする為の必要エネルギーはほんの少しだけでよい．エネルギー帯理論の術語では，この第 5 の電子は伝導体の極めて近くに位置するエネルギー状態に属している事になる．このような状態はドナー状態と言われ，この状態を作る不純物原子 (今回であれば As) はドナー原子と呼ばれている．ドナー原子を持つ半導体は多数の電子と少数の正孔を持っている．その電気伝導度は主に電子によってもたらされているので，n 型半導体と呼ばれ

ている．この n は負を意味する negative の頭文字である．

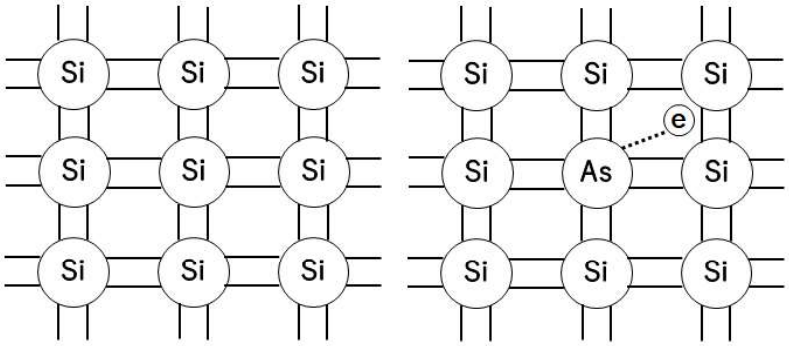


図 3　n 型半導体模式図

対して，不純物が Ga の場合を考える．Ga は価電子が 3 個しかないので，周囲の Si と 3 個の共有結合しか結ぶことができない．他の Si 原子からの電子が第 4 番目の電子として Ga 原子に配位して，のちに正孔を残すことも起こる．この場合は第 4 番目の原子を受け取った Ga 原子は陰イオンのような挙動をすることになる．エネルギー帯理論では Ga 原子の存在は価電子帯のきわめて近くに新しい状態を作る．これはアクセプタ状態と呼ばれ，この不純物 (今回であれば Ga) はアクセプタ原子と呼ばれる．アクセプタに移動した電子は跡に正孔を残すのでアクセプタは正孔を作ることとなる．この場合，電荷キャリアは正であるので半導体は p 型半導体と呼ばれる．

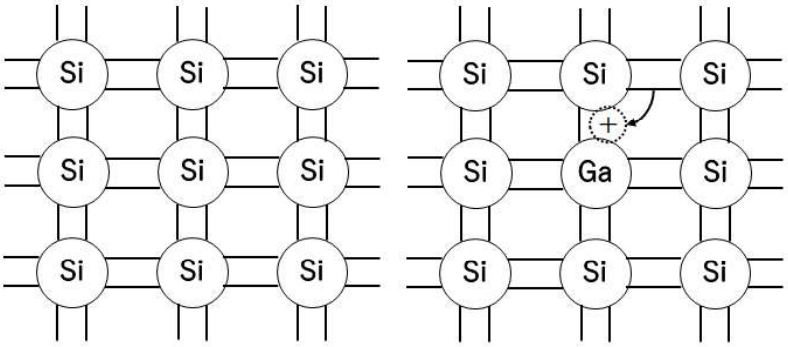


図 4　p 型半導体模式図

格子間原子もドナーまたはアクセプタとして動作し得る．ＳｉあるいはＧｅの中でリチウムは格子間原子となるが，このリチウムは伝導体の極めて近くにドナー状態をつくる．銅とニッケルは価電子帯と伝導帯の中間にドナー状態を作る．金はその格子中の位置に依存してアクセプタとしてもドナーとしても動作する．

n 型あるいは p 型不純物の原子 1 個ごとに電子または正孔が配置されることになる．いずれの場合も物質は中性であるが，電気伝導度が現れる場合には n 型半導体では電子が主なキャリアになり，p 型半導体では正孔が主なキャリアになる．不純物を添加すると新しい状態が作られてキャリアが移動できるようになるので，半導体の電気伝導度は不純物濃度が増すと大きくなる．

禁止帯エネルギーは温度に依存し，また不純物濃度・結晶の欠陥濃度にも依存している．ゲルマニウムのように禁止帯エネルギーが小さい場合には温度を上げるにつれて，不純物原子の存在によらず，熱励起によってつくられる電子正孔対のほうが電気伝導度に寄与することもある．したがって，どの半導体であっても非常に高温では真正半導体に近づく．

## 2.3　検出器としての pn 接合

半導体検出器の動作は，逆バイアス電圧を印加した pn 接合の正接に基づいている．図 5 に示すように，接合部に入射した放射線は接合部を通過する際に電子正孔対を作る．例えば，5MeV の α 粒子がシリコン製の検出器に入射して，全エネルギーが電子正孔対に付与したとすると，

$$\frac{5\times10^{6}eV}{3.65eV/対}\sim 1.37\times10^{6}個の電子正孔対 \tag{1}$$

をつくる．電子と正孔は電界によって掃引され，適切な電子回路によってパルスを出力する．

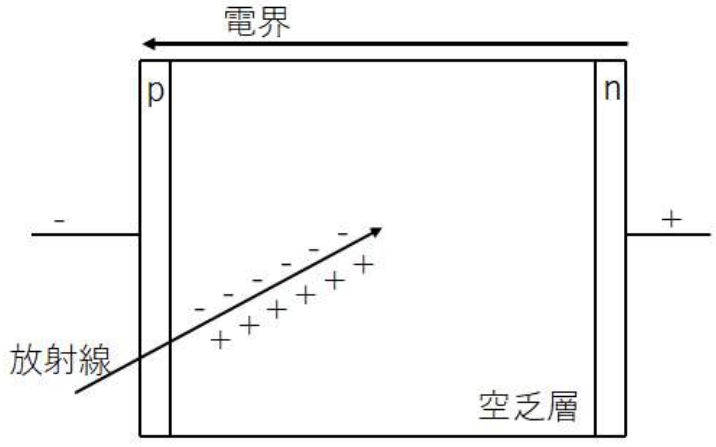


図 5　検出器としての pn 接合

半導体検出器の性能は電界がかかっている pn 接合部分に左右される．半導体検出器では有感層が数 mm し

かなく，また電子と正孔の速度が大きいので，電荷キャリアが有感層を横切って $10^{-7}$s 程度の時間に収集される．

半導体検出器において，入射粒子によって作られた電荷をすべて収集することが目標となる．この目標は電子と正孔が収集されるまでに再結合しないように電界を印加する事によって達成される．半導体検出器では再結合が全然起きなくても，格子の不完全性・空格子点・転位のような結晶の捕獲中心によって電荷キャリアの幾分かが失われる．入射放射線は結晶欠陥をつくり検出器の性能を劣化させ，その寿命を短縮させている．

これらをまとめると，検出器として用いる半導体物質はある性質を持つ必要があり，その中でも重要な性質はいかに上げられる．

- 比抵抗が高いこと．比抵抗が高くなければ電界を印加すると電流が流れてしまい，放射線によって作られた電荷パルスは定常電流に埋もれてしまう．
- キャリア移動度が大きいこと．電子と正孔は再結合あるいは捕獲される前に素早く動き電極側に収集される必要がある．
- 高い電界に耐える能力を持つこと．電界が高いほど電荷収集効率がよくなり，また電荷収集時間が短くなることからこの性質が重要になる．
- 結晶構造が完全であること．外部からの不純物を添加することを別にすると，半導体検出器材料は欠陥・空格子点・格子間原子等の無い完全な結晶である必要がある．欠陥があると電荷キャリアに対して捕獲中心として働く可能性がある．

# 3　FPGA について

FPGA とは，書き換え可能なハードウェア素子 (Field Programmable Gate Array) のことで，単なる論理回路の集合体ではなくシステムチップとしての性質・意味合いが強くなってきている．これまでの論理回路の設計では，回路の製造後設計ミスや規格変更などが起きる度に再設計・再製造・再試験と物理的な変更が必要となってくる．これに対して，必要な論理回路だけを盛り込んでおき，出荷後は利用者が配線などを組み替えられる製品が登場し，これが複雑化されて汎用なロジック IC を数百や数千個構成したデバイスが造られるようになった．FPGA はこの配線の組み換えやロジックの変更操作をプログラムの記憶要素としてデジタル的に再現し，いつでも何度でも変更できる物として登場した．

## 3.1　Kintex ® -7 FPGA KC705

高島研究室には 2018 年度の研究で用いた FPGA があった．これ自体は高価なものであるが, 新しく FPGA と ADC のセットを購入するより容易であると思われた．

Kintex ® -7 FPGA KC705 評価キットには，高性能，シリアル コネクティビティ，および最先端メモリ インターフェイスを要件とするシステムデザイン構築に必要なハードウェアのすべての基本コンポーネント，デザイン ツール，IP，およびターゲットデザインなどの検証済みリファレンスデザインが含まれている．同梱されている検証済みのリファレンスデザインや業界標準の FPGA メザニンコネクタ (FMC) によって，ドーター カードを使用した機能の拡張やカスタマイズが可能である．

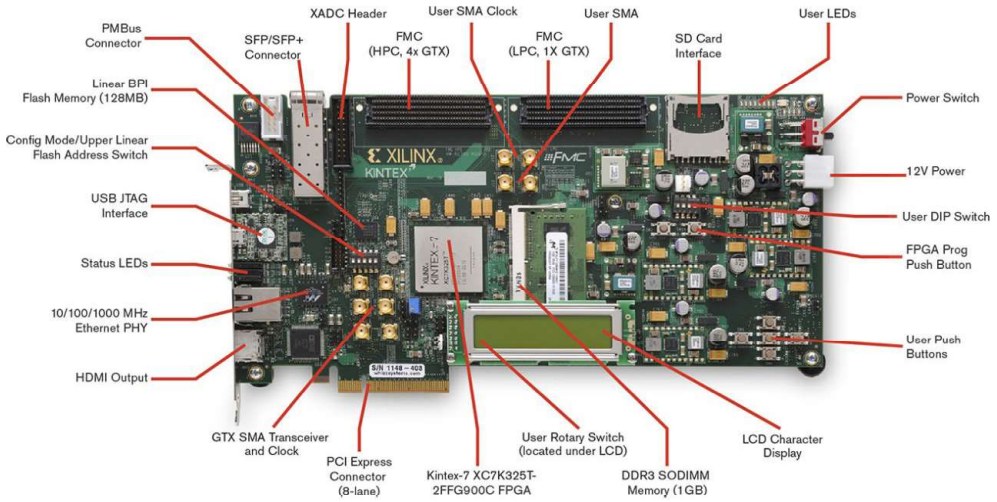


図 6　Kintex ® -7 FPGA KC705

## 3.2 FPGA の利用

Xilinx 社の FPGA は VIVADO と呼ばれる開発環境の設定が必要になる．リファレンスデザインは GitHub にまとめられており，それをインストール及び実行することで FPGA ボードと ADC のセットアップが完了する．この中には HDL や VHD といったプログラムのライブラリとコードのほか，ボードデザインを作成・構築するために必要な TcL スクリプトが含まれている．

Listing 1　KC705 と AD9467 のセットアップ

```
1 mkdir adi
2 cd adi
3 git clone https://github.com/analogdevicesinc/hdl.git
4 cd hdl
5 git checkout hdl_2018_r2
6 cd projects/ad9467_fmc/kc705
7 make
```

Vivado という Xilinx 社の FPGA 開発環境ソフトウェアのバージョンが 2018 年に使用した物であったため，パッケージの年度をそろえる必要があった．これにより，開発環境上で KC705 の作業を指定できる．このプログラムのほかに，ライセンスの電子キーが保管されているファイルまでのパスを設定すれば Vivado のコマンド一つで開発環境ソフトウェアが立ち上がる．ただし，プログラミング実行ターミナルごとに設定をしなければならないので，一度設定したターミナルは消さずに残しておく方が便利である．

Listing 2　ライセンスの設定

```
1 source /opt/Xilinx/Vivado/2018.2/settings64.sh
```

# 4　AD 変換

## 4.1　アナログ信号とデジタル信号

アナログ信号とは，信号の変化が時間的に途切れること無く連続している信号のことで，温度変化・音声・地震の揺れなど自然現象や物理現象の全てがこの信号を発している．一方，デジタル信号とは 0 と 1 のみで構成された信号のことで，中間的な 0.5 などは考えないものとしている．デジタル信号は FPGA やマイコンをはじめと知るデジタル回路の中で用いられている．デジタル信号は中間値がないことから離散信号とも呼ばれている．

アナログ信号の回路も当然存在しているが，アナログ信号には素子ごとのばらつきや信号の劣化に弱く，また必要回路素子が多いことからデジタル信号への変換をすることが一般とされている．当然デジタル回路にもデメリットはあり，データの量が多くなると桁違いのデータを伝える必要がある．

## 4.2　AD 変換

アナログ信号をデジタル信号に変換するためには 3 つの手順が必要となる．

- 標本化，アナログ信号を一定時間ごとに区切り，その値を読むこと．サンプリングともいう．
- 量子化，標本化し読み込んだ値をデジタル信号に変換できるように加工すること．
- 符号化，量子化された値を指定された 2 進数の桁数で表現すること．この最大桁数が分解能といい，大きいほど変換能力が優れている．

標本化の例を図 7 に示す．入力信号に対して，一定時間ごとに区切る作業を行う．このとき一定時間を決める信号のことをサンプリング周波数もしくはサンプリングクロックという．これに対応させて標本化そのものをサンプリングと呼ぶこともある．クロックの周期に対して信号の大きさを取り出すことが主目的となる．サンプリングクロックが短いほどより細かくデジタル信号に変換できることとなる．

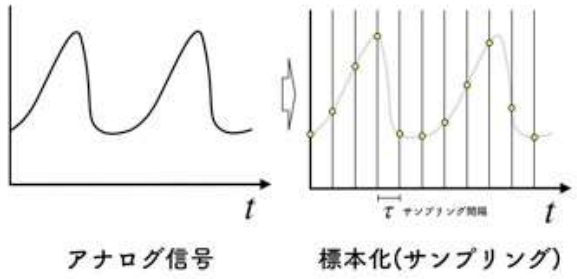


図 7　標本化 (サンプリング)

量子化の例を図 8 に示す．デジタル信号は表現できる数値がいわゆる整数値が基本となっているので，桁のまるめのようなことが起こる．この時生じる誤差を量子化誤差という．個の誤差が小さいほど変換器の信頼性が高いと言える．

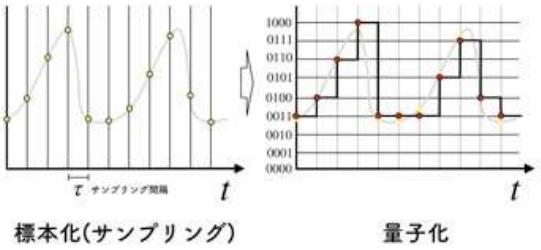


図 8　量子化

符号化の例を図 9 に示す．変換器の機能 (ビット数) に応じて量子化されたデータの値を 2 進数に変換する．ビット数が少なければ表現できる値も少なくなり，多くなればなるほど細かな値を表現できる．

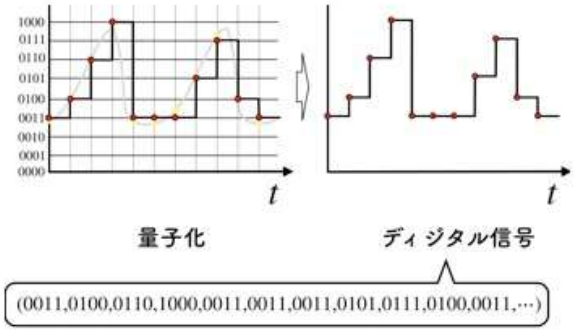


図 9　符号化

# 5　AD9467

ADC には標本化と符号化の 2 か所で性能が大きくかかわっている．標本化について，必要な周波数は信号の波形に対してどのくらい細かく観察するかできまる．例えば，信号の幅が 10$\mu$s に対して 10 回程度の標本をとりたい場合，1$\mu$s に一回サンプルする必要があるので，1MHz の性能が求められる．

符号化するさいの分解能や表現するチャネル数についても同様に，5MwV の信号を分解能 10keV の測定器で観察した場合，500 段階の測定値が得られる．これらをすべて 2 進数にするためには 9 ビット以上が必要となってくる．

近年の電子技術は発展を続けており，ADC については高速のものから高分解能のもの，さらにはその両方を兼ねそろえたものまである．本研究の趣旨はなるべく低価格で抑えたいこともあり，AnalogDegital 社の AD9467 を用いて実験をした．先述の標本化では 250MHz，符号化では 16 ビットであるためあらゆる測定が可能と考えられる．AnalogDegital 社から約 440 ドルで購入できる．

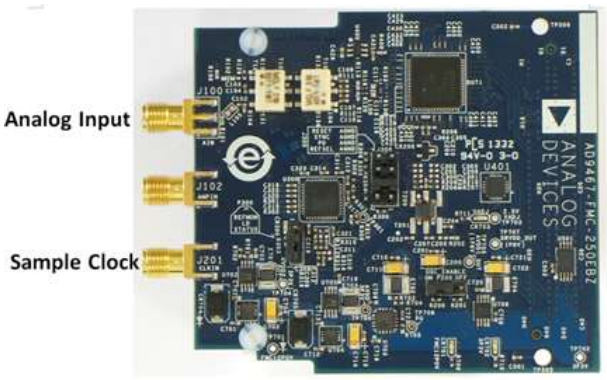


図 10　AD9467 評価ボード

## 5.1　AD9467 の設定

ここでは装置の設定やサンプリングクロックの設定などについて説明をする．また，研究に使ったパソコンは CentOS 7 のものを用いた．これは利用するプログラミングの動作が Windows に比べて容易であるためである．AD9467 の Linux 用ライブラリは no-OS としてあるものを使う．こちらも 2018 年のバージョンにそろえる必要がある．

Listing 3　AD9467 のライブラリ導入

```
1 cd adi
2 git clone https://github.com/analogdevicesinc/no-OS.git
3 cd no-OS
4 git checkout 2018_r2
```

さらに ADC 実行において参照されるプログラムのボードパスも変更することで測定準備はすべて整えることができた．リスト 4 一行目の先頭にある#記号はその行をコメントアウトして実行時に参照しないことを意味する．

Listing 4　make 参照の変更

```
#M_HDF_FILE := $(HDL-DIR)/projects/ad9467_fmc/kc705/ad9467_fmc_ebz.sdk/system_top.hdf
M_HDF_FILE := $(HDL-DIR)/projects/ad9467_fmc/kc705/ad9467_fmc_kc705.sdk/system_top.hdf
```

最後に，ライブラリ内にある KC705 のフォルダ領域で make コマンド指定して実行することで FPGA と ADC の動作を行うことができる．これらのソースコードは巻末に載せる．

Listing 5　make 参照の変更

```
make run
make capture
```

ただし，このプログラム実行自体はかなり短い時間で終わってしまい，出力される csv ファイルを参照するとサンプリングの回数としては約 8000 程で終わってしまう．

## 5.2　外部クロックの設定

ADC には外部からサンプルタイミングを決める信号を入力する必要がある．今回は Si5351 というクロックジェネレータを Arduino で制御して使う．回路図や動作プログラミングはすべて企業サイトに載っているものを使用した．プログラムは巻末のリスト 8 に載せる．これは Adafruit 社の Si5351A を流用したものであり，公称値では 8k から 150MHz までのクロック信号を出すことができる．Amazon などの通販サイトで 1000 円ほどで購入できるが，はんだ付けや配線は自ら行う必要がある．

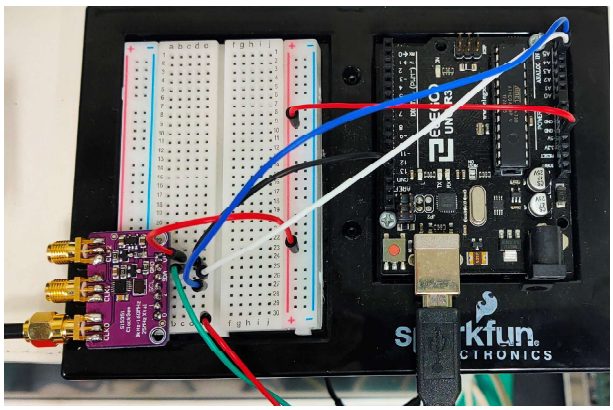

図 11　SI5351 と Arduino

# 6　実機を用いたサンプリング

KC705 と AD9467 を用いて，図 12 に示す電圧信号発生装置の波形観察を実施した．回路の外観を図に示す．また信号分配器を用いてサンプリングクロックと矩形電圧信号について，矩形の幅を変更させることでどのように違いが表れるかを示した．

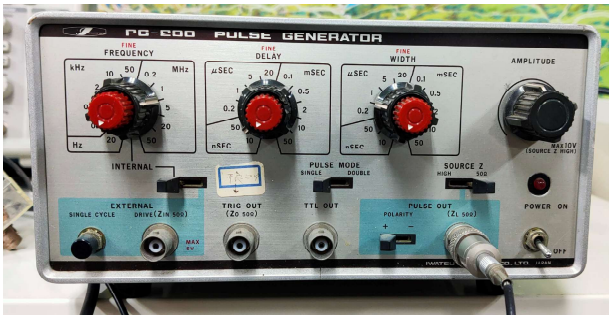


図 12　電圧信号発生装置

## 6.1　AD9647 を用いた電圧の標本化検証

研究室にあるパルスジェネレーターを用いて，標本化の動作確認を行った．オシロスコープのメモリを見ながら，矩形波の High 時間を調節して 120ns と 300ns の二通りで検証を行った．この時，標本化の信号が SI5351 の初期設定では不安定だったので，サンプルクロックを 112.5 の半分である 56.25MHz で行った．実験の様子を図 13 に示す．

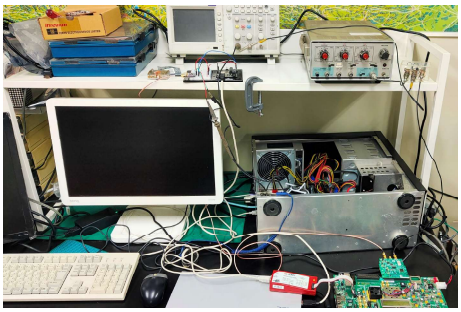

図 13　電圧サンプリングの様子

図 14 にはオシロスコープで読み取った値と，それぞれのサンプル結果の様子を示す．サンプルクロックが 56MHz であるので，120ns に対しては 6〜7 回の High 記録，300ns に対しては 16〜17 回の High 記録がとれていることを確認できた．

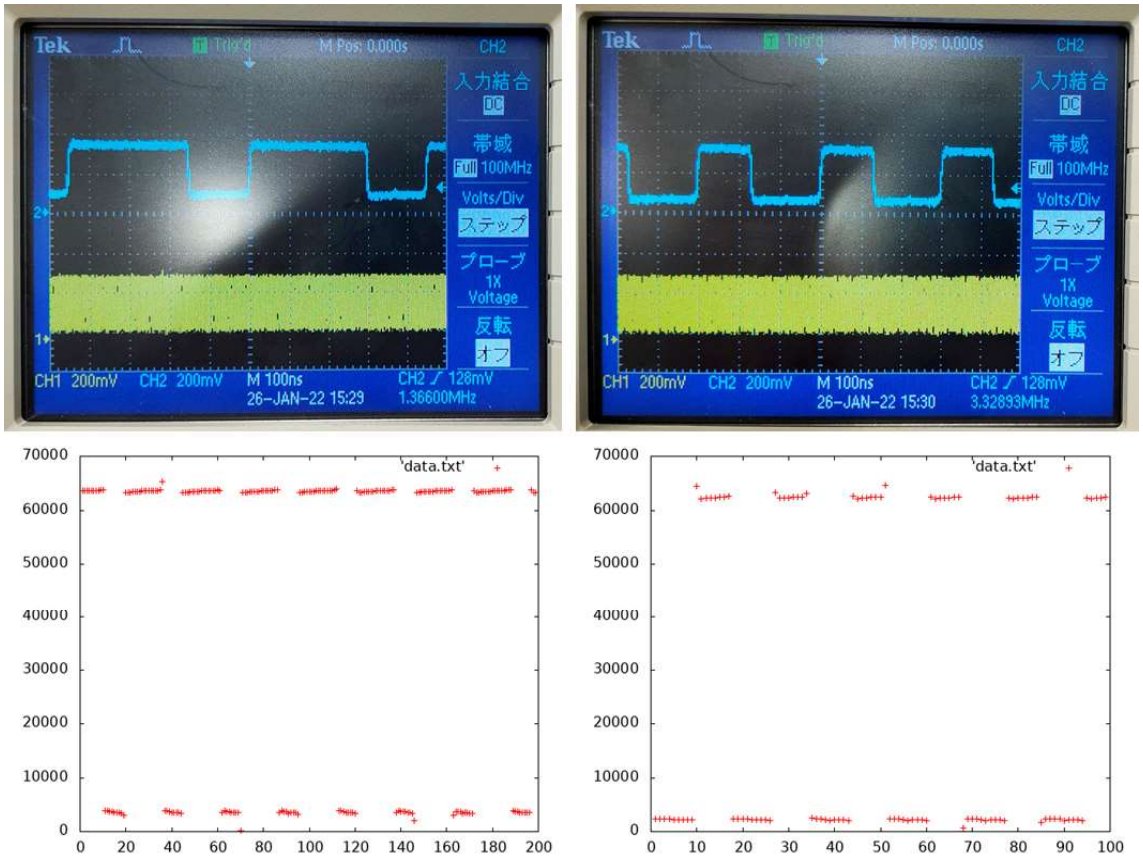


図 14　右が 120ns 左が 300ns のオシロと結果

## 6.2　半導体検出器からのサンプリング

同様の実験設定で半導体検出器の信号を入力信号に入れて，検出器からの信号波形が見えるかどうか試験を行った．オシロスコープで確認した段階では，半導体検出器からの信号は約 10ns 程度であるので，数回の測定を行いパルスの形が見えることができれば良い．線源には $^{133}$Ba を用いた．$^{133}$Ba は電子捕獲を行い約 300keV のガンマ線を出す．測定結果を図 15 に示す．

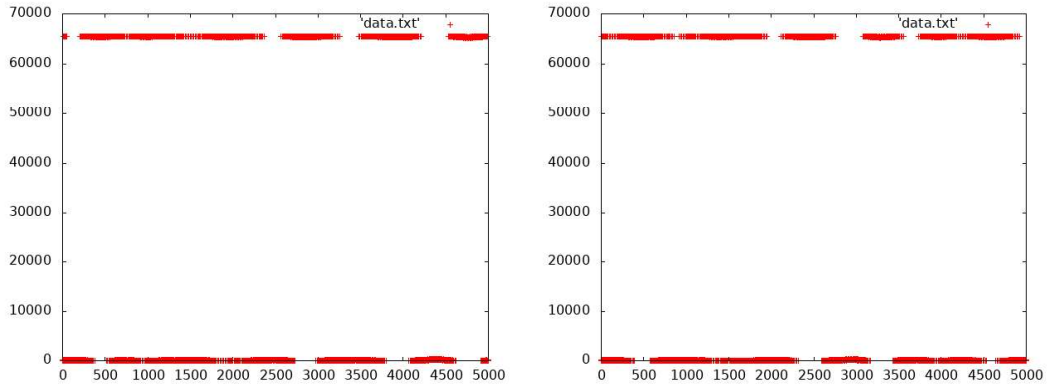


図 15　$^{133}$Ba の測定結果

図 15 は High の信号を取得できた物データを示しているが，どれも $2^{16}$ である 65536 付近の数字になっている．この現象は標本化の検証の結果とも一致しており，ADC の動作のうち量子化や符号化の段階で何か下の誤作動が起きている可能性があると思われた．

そこで，ADC の動作が正しく行えているかどうか検証するために次の検証を行った．図 12 の信号装置から電圧を 50mV・100mV・200mV の三段階に分けて再度プログラムを実行させて ADC の出力値が変化するかどうかを試した．図 16 では High の時間が 6.1 節と異なるが，これは様々な実験をしている途中でメモリを操作したまま放置していたためである．サンプリングのタイミング自体には誤作動はないので，ここでは大きな問題点ではない．

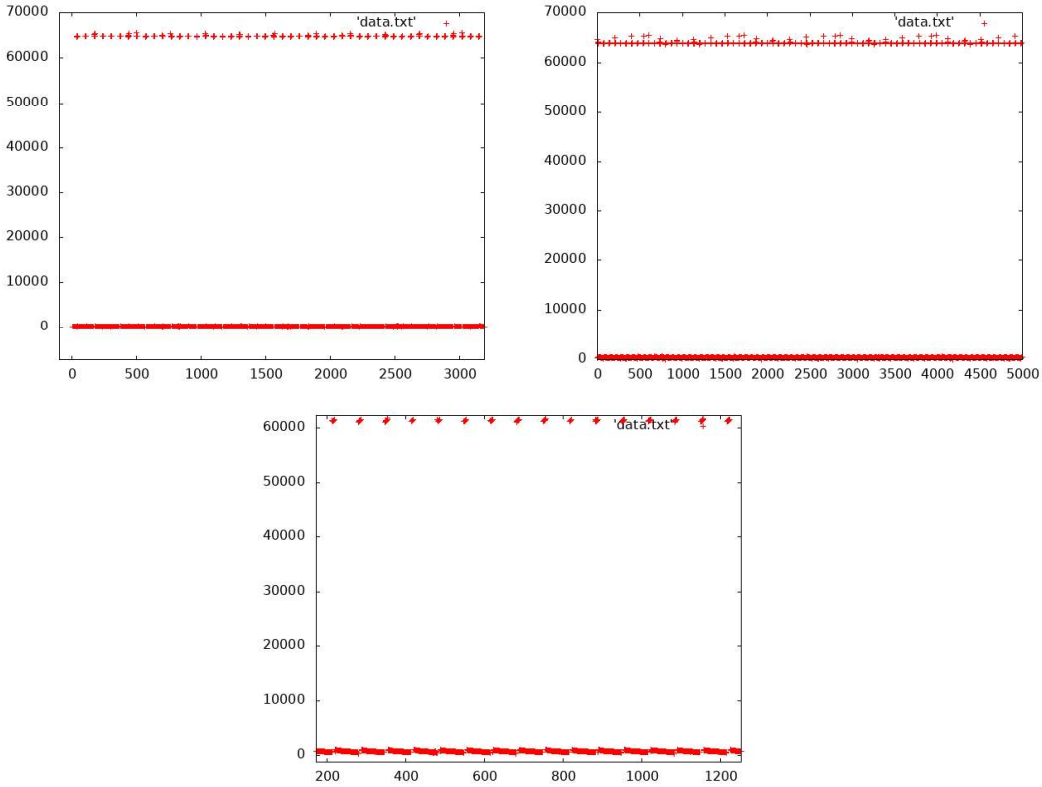


図 16　上段左から 50，100mV 下段 200mV の結果

この結果から，ADC のサンプリングは正しく行えているが，それ以外の動作が全く参考にできないことが分かった.

# 7　AD 変換器の回路素子

ADC の動作について，動作のプログラムコードを読んでみたが特に値を大きく乗算していると思われる箇所は見つからなかった．もう一度 AD9467 の商品説明ページを参照して見たとろ，AD9467 の構成回路の中に，入力端子のすぐそばの箇所にバッファ回路が入っていることが分かった．

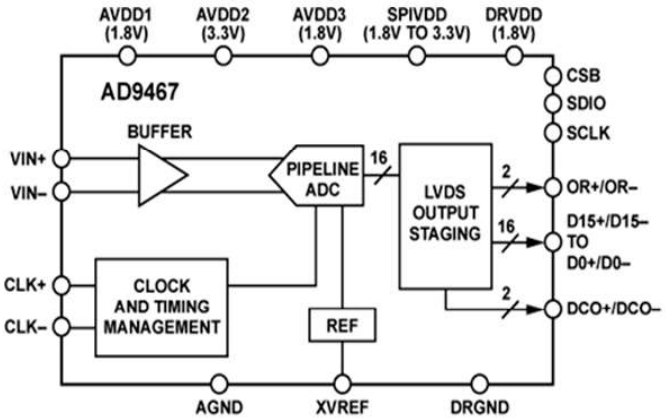


図 17　AD9467 構成回路図

バッファ回路とは，オペアンプで活用される回路の一つで主に入力信号を増幅させる働きをもつ．論理回路も電気回路や電子回路であり，それぞれの回路には電気抵抗も存在する．そのため，回路が複雑になり長くなればなるほど，流れる電気の電圧は下がり，信号強度も悪化してしまう．そこで必要となるのがバッファ回路である．バッファ回路には入力された電気の電圧，及び信号強度を補正する機能がある．そのため，バッファ回路を流れた電気は電圧，及び信号強度が補正され，あらためてその後の回路を流れていくことがでる．機能が「電圧，及び信号強度の補正」であることから，論理回路で見ると何も起こらず，見た目にはただ流れただけとなってしまう．

そのような働きを持つ回路素子がついているため，50 m V の入力信号であっても 200mV の入力信号であっても $2^{16}$ に量子化されるような信号に増幅されている可能性がある事がわかった．当初 FPGA ごと購入して実験をするか迷っていた段階で購入予定にあた EclipseZ7 という FPGA にセットで売られている ADC ボードには AD9648 のチップが搭載されており，こちらの回路図を見ると入力端子の後に何も回路素子がなくそのまま ADC 素子に向かっていることが分かる．

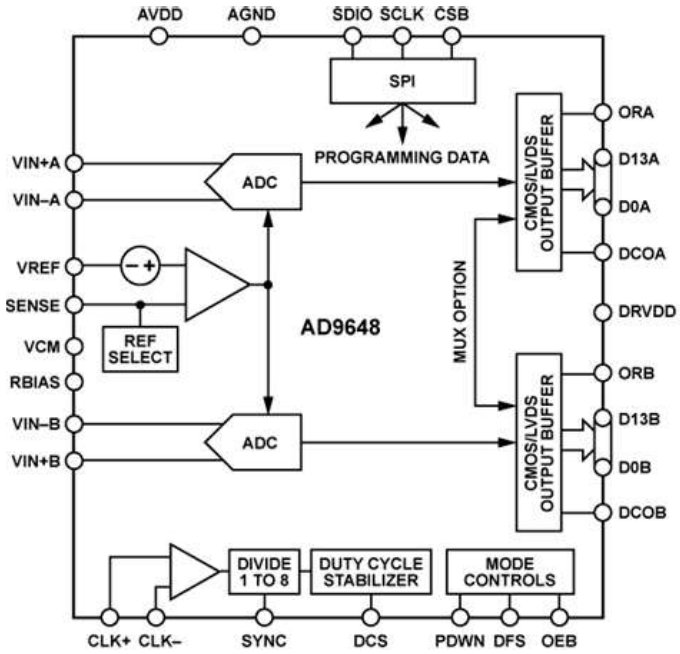


図 18　AD9648 構成回路図

# 8　まとめ

## 8.1　まとめ

研究室内にある FPGA と新規購入した ADC を用いて放射線の波高分析を行いたかったが，電子回路やプログラミングの知識が乏しく結果として標本化は成功しているが，読み取っている値に内部増幅がかけられているため解析に必要な波高分析の役目を達成することはできなかった．ADC や FPGA のプログラミングをしている研究はあるが，そのソースコードや使用している機材は高価なものが多く，あまり参考にならない現状もある．

また，ADC をかけた後のデータ処理についても課題が残っている．特に，放射線測定は数十分から数時間ずっとデータコンバータをかけながら線源の放射線エネルギーを取得し，その後 ADC の分解能から線形近似しチャネル番号とエネルギー値を照らし合わせる作業がある．今回は ADC の実動作の段階で詰まってしまい，データをチャネル分けする方法や，チャネルからヒストグラムを生成する為の方法は調査しきれていない結果となった．

ADC や FPGA をはじめとする電子機器は企業や測定専門家が自由に扱える反面，高校以下での授業資料や教材としての利用はきわめて壁が高い存在である．論理回路や電子回路，さらには VHDL や C 言語などのプログラミング知識，I2C・TCL などの通信技術，すべてが集まった研究分野である．今回も，ADC 選定時にバッファ回路の存在を知っていれば欲しいデータが得られて，次の段階に進めただろう．手元にある機械と連携があるからといって容易に購入して進めることは失敗になることを反省として活かしていきたい．

## 8.2　謝辞

本研究を取り組むにあたって，高嶋隆一先生にはお忙しい時間をさいていただき，FPGA の提供やクロックジェネレーターの選定など，様々な支援・指導をしてくださったことに深く感謝しております．研究室では放射線測定や電子回路について，豊富な知識をご享受していただき，電子回路や電子機器に対する知見もいくらか広がったと思われます．深く感謝すると共に御礼申し上げます．

# 9　プログラムコード

Listing 6　ADC 実行プログラム (make run)

```
# 7series zynq

ifeq ($(HDF-FILE),)
  HDF-FILE := $(M_HDF_FILE)
endif

ifeq ($(OS), Windows_NT)
  XSCT_CMD := xsct.bat
  XSDB_CMD := xsdb.bat
else
  XSCT_CMD := xsct
  XSDB_CMD := xsdb
endif

ifeq ($(CAPTURE_BADDR),)
  CAPTURE_BADDR := 0x80800000
endif

ifeq ($(CAPTURE_SIZE),)
  CAPTURE_SIZE := 32768
endif

ifeq ($(NR_OF_CHAN),)
  NR_OF_CHAN := 1
endif

ifeq ($(BITS_PER_SAMPLE),)
  BITS_PER_SAMPLE := 16
endif

XSCT_LOG := xsct.log
XSCT_SCRIPT := $(NOOS-DIR)/scripts/xsct.tcl
XSDB_SCRIPT := $(NOOS-DIR)/scripts/xsdb.tcl
XSDB_CAPTURE := $(NOOS-DIR)/scripts/xilinx_capture.tcl

COMPILER_DEFINES := XILINX
COMPILER_DEFINES += MICROBLAZE
COMPILER_DEFINES += $(M_COMPILER_DEFINES)

P_HDR_FILES := xilsw/src/platform_config.h
P_HDR_FILES += xilsw/src/platform.h
```

```
42 P_SRC_FILES := xilsw/src/platform.c
43
44 ELF_FILE := sw/Release/sw.elf
45
46 HDR_FILES := $(P_HDR_FILES)
47 HDR_FILES += $(M_HDR_FILES)
48 HDR_FILES += $(foreach i_dir, $(M_INC_DIRS), $(wildcard $(i_dir)/*.h))
49
50 SRC_FILES := $(P_SRC_FILES)
51 SRC_FILES += $(M_SRC_FILES)
52 SRC_FILES += $(foreach i_dir, $(M_INC_DIRS), $(wildcard $(i_dir)/*.c))
53
54 LIB_FILES := $(M_LIB_FILES)
55 LIB_NAME := $(M_LIB_NAME)
56
57 .PHONY: all
58 all: $(ELF_FILE)
59
60
61 $(ELF_FILE): $(HDR_FILES) $(SRC_FILES) $(LIB_FILES)
62         $(XSCT_CMD) $(XSCT_SCRIPT) sources $(HDR_FILES) $(SRC_FILES) $(LIB_FILES) > $(
                XSCT_LOG) 2>&1
63         $(XSCT_CMD) $(XSCT_SCRIPT) library $(LIB_NAME) >> $(XSCT_LOG) 2>&1
64         $(XSCT_CMD) $(XSCT_SCRIPT) build
65
66
67 $(P_HDR_FILES): hw/system_top.bit
68 $(P_SRC_FILES): hw/system_top.bit
69
70
71 hw/system_top.bit: $(HDF-FILE)
72         rm -fr .metadata .Xil hw bsp xilsw sw xsct.log
73         $(XSCT_CMD) $(XSCT_SCRIPT) init $(HDF-FILE) > $(XSCT_LOG) 2>&1
74         $(XSCT_CMD) $(XSCT_SCRIPT) defines $(COMPILER_DEFINES) >> $(XSCT_LOG) 2>&1
75         $(XSCT_CMD) $(XSCT_SCRIPT) make-bsp-xilsw >> $(XSCT_LOG) 2>&1
76
77
78 .PHONY: run
79 run: $(ELF_FILE)
80         $(XSDB_CMD) $(XSDB_SCRIPT) MICROBLAZE
81
82 .PHONY: clean
83 clean:
84         rm -rf hw bsp sw .metadata .Xil xilsw xsct.log SDK.log
85
86 .PHONY: capture
```

```
87 capture: $(ELF_FILE)
88         $(XSDB_CMD) $(XSDB_CAPTURE) MICROBLAZE $(CAPTURE_BADDR) $(CAPTURE_SIZE) $(
               NR_OF_CHAN) $(BITS_PER_SAMPLE)
```

Listing 7　ADC 書き出しプログラム (make capture)

```
# Data acquisition trough xsdb

# This script has two modes of geting info of the data to read from memory and store
# in local files:
# 1. Receive al info from the command line
# 2. Receive the start address for the captured data in memory, the rest
# of the info will be automatically read out of the memory (NOT YET IMPLEMENTED);

set m_type [lindex $argv 0]
set start_addr [lindex $argv 1] ;# hex (0x...)
set num_of_samples [lindex $argv 2] ;# decinal
set num_of_channels [lindex $argv 3] ;# for RF projects consider 2x nr of channels(I
    and Q)
set storage_bits [lindex $argv 4] ;# for 8-16bit rezolution data is stored on 16bit (
    max 32)

connect

if {$m_type == "ZYNQ_PSU"} {
  targets -set -filter {name =~ "ARM*#0*"}
}

if {$m_type == "ZYNQ_PS7"} {
  targets -set -filter {name =~ "ARM*#0*"}
}

if {$m_type == "MICROBLAZE"} {
  targets -set -filter {name =~ "*MicroBlaze #0*"}
}

after 1000

if { $storage_bits == "" } {
        puts "Read parameters from memory"
        set format_address [expr $start_addr - 0x02 1]
        set real_bits_address [expr $start_addr - 0x04 1]
        set storage_bits_address [expr $start_addr - 0x06 1] ;# bytes_per_sample
        set shift_address [expr $start_addr - 0x08 1]
        set endianness_address [expr $start_addr - 0x10 1]
        set num_of_samples_address [expr $start_addr - 0x12 1]
        set num_of_channels_address [expr $start_addr - 0x20 1]

        # Used information
        set storage_bits [mrd -force $storage_bits_address]
        set num_of_samples [mrd -force $num_of_samples_address]
```

```
44          set num_of_channels [mrd -force $num_of_channels_address]
45  }
46
47  if { $storage_bits == "" } {
48          puts "WRONG PARAMETER VALUE storage_bits = $storage_bits "
49          exit
50  }
51  if { $num_of_samples == ""} {
52          puts "WRONG PARAMETER VALUE num_of_samples = $num_of_samples"
53          exit
54  }
55  if { $num_of_channels == ""} {
56          puts "WRONG PARAMETER VALUE num_of_channels = $num_of_channels"
57          exit
58  }
59
60  puts "Script parameters:"
61  puts "Start address = $start_addr"
62  puts "num of samples = $num_of_samples"
63  puts "num of channels = $num_of_channels"
64  puts "bytes per sample = $storage_bits\n"
65
66  puts "Moving data into .csv files..."
67
68  set start_addr [expr $start_addr]
69  set num_of_word [expr 2 * $num_of_samples]
70  set readData [mrd -force $start_addr $num_of_word]
71
72  set f_name "capture_ch"
73  set f_type ".csv"
74
75  for {set index 1} {$index <= $num_of_channels} {incr index 1} {
76          set file_name $f_name$index$f_type
77          file delete -force $file_name
78          set f($index) [ open $file_name a ]
79  }
80
81  # mrd returns:
82  # index x x+1
83  # info address: data(32b)
84
85  # Arrange channel data into individual files (considering hdl system has a cpack core)
86  set ch_x_index 1
87  for {set index 1} {$index < $num_of_samples} {incr index 2} {
88          set data [lindex $readData $index] ; # 32 bit, 16 bit/channal
89          set data [expr 0x$data]
```

```
90          if { $storage_bits <= 16 } {
91                  set sample1 [expr {$data & 0xFFFF}]
92                  set sample2 [expr {($data >> 16) & 0xFFFF}]
93                  if { $num_of_channels == 1 } {
94                          puts $f($ch_x_index) $sample1
95                          puts $f($ch_x_index) $sample2
96                  } else {
97                          for {set n 1} {$n < $num_of_channels } {incr n 2} {
98                                  puts $f($ch_x_index) $sample1
99                                  if {$ch_x_index < $num_of_channels} {
100                                         set ch_x_index [expr $ch_x_index + 1]
101                                 } else {
102                                         set ch_x_index 1
103                                 }
104
105                                 puts $f($ch_x_index) $sample2
106                                 if {$ch_x_index < $num_of_channels} {
107                                         set ch_x_index [expr $ch_x_index + 1]
108                                 } else {
109                                         set ch_x_index 1
110                                 }
111                                 if { [expr $n + 2] < $num_of_channels} {
112                                         set index [expr $index + 2]
113                                         set data [lindex $readData $index]
114                                         set data [expr 0x$data]
115                                         set sample1 [expr {$data & 0xFFFF}]
116                                         set sample2 [expr {($data >> 16) & 0xFFFF}]
117                                 }
118                         }
119                 }
120         } elseif { $storage_bits <= 32 } {
121                 set sample [expr {$data & 0xFFFFFF}]
122                 if { $num_of_channels == 1 } {
123                         puts $f($ch_x_index) $sample
124                 } else
125                         for {set n 1} {$n < $num_of_channels } {incr n 1} {
126                                 if {$ch_x_index < $num_of_channels} {
127                                         set ch_x_index [expr $ch_x_index + 1]
128                                 } else {
129                                         set ch_x_index 1
130                                 }
131                                 puts $f($ch_x_index) $sample
132                                 if { [expr $n + 1] < $num_of_channels} {
133                                         set index [expr $index + 2]
134                                         set data [lindex $readData $index]
135                                         set data [expr 0x$data]
```

```
136                                         set sample [expr {$data & 0xFFFFFF}]
137                                 }
138                         }
139         } else {
140                 puts "ERROR: Unsupported number of bits/sample"
141                 exit
142         }
143 }
144
145 for {set index 1} {$index <= $num_of_channels} {incr index 1} {
146         close $f($index)
147 }
148
149 puts "Done."
150
151 disconnect
152
153 exit
```

Listing 8　SI5351 制御プログラム

```
#include <Adafruit_SI5351.h>

Adafruit_SI5351 clockgen = Adafruit_SI5351();

/**************************************************************************/
/*
    Arduino setup function (automatically called at startup)
*/
/**************************************************************************/
void setup(void)
{
  Serial.begin(9600);
  Serial.println("Si5351 Clockgen Test"); Serial.println("");

  /* Initialise the sensor */
  if (clockgen.begin() != ERROR_NONE)
  {
    /* There was a problem detecting the IC ... check your connections */
    Serial.print("Ooops, no Si5351 detected ... Check your wiring or I2C ADDR!");
    while(1);
  }

  Serial.println("OK!");

  /* INTEGER ONLY MODE --> most accurate output */
  /* Setup PLLA to integer only mode @ 900MHz (must be 600..900MHz) */
  /* Set Multisynth 0 to 112.5MHz using integer only mode (div by 4/6/8) */
  /* 25MHz * 36 = 900 MHz, then 900 MHz / 8 = 112.5 MHz */
  Serial.println("Set PLLA to 900MHz");
  clockgen.setupPLLInt(SI5351_PLL_A, 36);
  Serial.println("Set Output #0 to 112.5MHz");
  clockgen.setupMultisynthInt(0, SI5351_PLL_A, SI5351_MULTISYNTH_DIV_8);

  /* FRACTIONAL MODE --> More flexible but introduce clock jitter */
  /* Setup PLLB to fractional mode @616.66667MHz (XTAL * 24 + 2/3) */
  /* Setup Multisynth 1 to 13.55311MHz (PLLB/45.5) */
  clockgen.setupPLL(SI5351_PLL_B, 24, 2, 3);
  Serial.println("Set Output #1 to 13.553115MHz");
  clockgen.setupMultisynth(1, SI5351_PLL_B, 45, 1, 2);

  /* Multisynth 2 is not yet used and won't be enabled, but can be */
  /* Use PLLB @ 616.66667MHz, then divide by 900 -> 685.185 KHz */
  /* then divide by 64 for 10.706 KHz */
  /* configured using either PLL in either integer or fractional mode */

```

```
46   Serial.println("Set Output #2 to 10.706 KHz");
47   clockgen.setupMultisynth(2, SI5351_PLL_B, 900, 0, 1);
48   clockgen.setupRdiv(2, SI5351_R_DIV_64);
49
50   /* Enable the clocks */
51   clockgen.enableOutputs(true);
52 }
53
54 /**************************************************************************/
55 /*
56     Arduino loop function, called once 'setup' is complete (your own code
57     should go here)
58 */
59 /**************************************************************************/
60 void loop(void)
61 {
62 }
```

## 参考文献


[1] ニコラス ツルファニディス (坂井 英二 訳) 『放射線計測の理論と演習　上巻・下巻』 現代工学社

[2] AnalogDevices アナログデジタルコンバータ 高速 A/D コンバータ AD9467
(URL https://www.analog.com/jp/products/ad9467.html) 2016/2/09

[3] AnalogDevices アナログデジタルコンバータ 高速 A/D コンバータ AD9648
(URL https://www.analog.com/jp/products/ad9648.html) 2016/2/09

[4] Adafruit Si5351 Clock Generator Breakout
(URL https://learn.adafruit.com/adafruit-si5351-clock-generator-breakout/wiring-and-test)

[5] ウォルト ユング (藤森 弘己 訳) 『OP アンプの歴史と回路技術の基礎知識 OP アンプ大全 第一巻・第二巻』 CQ 出版社

[6] 小林 優 『FPGA 大全 Xilinx 編 第二版』 秀和システム

[7] 井原 洋平ほか 『FPGA を用いたディジタル放射線計測』 日本放射線安全管理学会誌 第十巻二号

[8] 芳原 新也 『教育用 FPGA ボードを用いた安価な多重波高分析器 (MCA) の構築』 近畿大学原子力研究所年報 2010Vol.47