２０１１年度 卒 業 論 文

高エネルギー衝突実験に向けた
プログラマブルゲートアレイ技術の考察

広島大学理学部物理科学科
クォーク物理学研究室

B081732 佐藤　大地

2012 年 2 月 10 日

指導教官　杉立 徹 教授

主査　三好 隆博 助教

副査　鬼丸 孝博 准教授

概 要

プログラマブルゲートアレイ (FPGA: Field Programmable Gate Array) とは、ハードウェア記述言語によるプログラミングによって回路の再構成が可能なセミカスタム集積回路です。FPGA はその汎用性に加え、近年の急速なテクノロジーの進歩に伴い高集積化や低価格化などが著しく、極めて将来性の高い集積回路の設計・構築技術です。一方、高エネルギー衝突実験においては、検出器のトリガー回路のさらなる高処理能力化が切望されています。トリガー回路の開発に FPGA を用いることによって、高処理速度・高精度のアルゴリズムを効率的に探索することが可能になると考えらます。それ故、FPGA 技術の有効性と適用条件を明確にすることは極めて重要な課題です。本研究では、ハードウェア記述言語として幅広く利用され汎用性の高い Verilog HDL を採用し、FPGA の技術試験を行いました。動作確認として、LED の点灯・スイッチを押した回数を数える・時間を数えるなどの比較的簡単な回路を作成しました。これらの技術を発展させ、ナノ秒スケールの処理速度を要するトリガー回路を用いて、宇宙線ミューオンの寿命を測定しました。その結果、寿命を見積もることができ、ミューオンの崩壊時間を測定するシグナルを FPGA で制御することに成功しました。これらの技術試験を通し、FPGA 技術の有用性を立証すると共に、FPGA によって外部信号を正確に処理する条件を明らかにしました。

目 次

1 導入 **5**
1.1 高エネルギー衝突実験 5
1.2 PHOS 6
1.3 プログラマブルゲートアレイ (FPGA) 7
1.4 Velilog HDL 10
1.5 本研究の目的 11

2 FPGA 動作確認 **12**
2.1 開発ツール 12
2.2 XILINX SPARTAN-3A Starterkit 17
2.2.1 信号を扱う 18
2.2.2 回数をカウントする 20
2.2.3 クロックを扱う 22
2.3 KEK Seminer board 24
2.3.1 NIM 信号を出力する 25
2.3.2 NIM 信号を入力する 25
2.3.3 NIM 信号を入力・出力する 26
2.3.4 コインシデンスをとる 28

3 FPGA による実証実験 **29**
3.1 NIM,CAMAC 規格 29
3.2 ミューオンの平均寿命 30
3.3 宇宙線の観測 31
3.4 セットアップ 32
3.5 擬似ミューオン崩壊シグナル測定 34
3.6 シンチレータを用いたミューオン寿命測定 36

4 結果・考察 **37**
4.1 性能評価 37
4.2 検証実験 40
4.3 ミューオン寿命測定 42
4.4 高エネルギー衝突実験に向けて 44

5 まとめ **45**

図 目 次

1 LHC 全体像と ALICE 全体像 [1][2] . . . 5
2 PHOS 全体像 [3] . . . 6
3 PHOS 検出器から CTP までの回路図と TRU ボード内 . . . 7
4 FPGA . . . 7
5 FPGA 内部構造 . . . 8
6 ISE 手順１ . . . 13
7 ISE 手順２ . . . 14
8 ISE 手順３ . . . 15
9 ISE 手順４ . . . 16
10 SPARTAN-3A Starter kit Board . . . 17
11 LED 点灯 HDL・動作 . . . 18
12 切り替え HDL・動作 . . . 19
13 押しカウント HDL・動作 . . . 21
14 ２進数押しカウント HDL・動作 . . . 22
15 クロック HDL・動作 . . . 23
16 KEK Seminer Board . . . 24
17 SW to NIMOUT HDL . . . 25
18 NIMIN to LED HDL . . . 25
19 NIMIN to NIMOUT 回路概略 . . . 26
20 NIMIN to NIMOUT HDL . . . 27
21 コインシデンス HDL とオシロスコープ波形 . . . 28
22 アナログシグナルの波形・シンチレータの写真 . . . 31
23 実験に使用したシンチレータ . . . 32
24 実験シグナル概略・HDL . . . 33
25 擬似実験回路概略・NIMIN シグナル . . . 35
26 ミューオン寿命測定回路概略・NIMIN 信号 . . . 36
27 NIMOUT 立ち上がり立ち下り . . . 37
28 NIMIN to NIMOUT シグナルの比較 . . . 38
29 CLK コインシデンス回路・シグナル概略 . . . 39
30 擬似実験 NIMOUT オシロスコープ . . . 40
31 擬似実験 NIMIN 調整 . . . 41
32 擬似実験ヒストグラム . . . 41
33 実験信号の幅 . . . 42
34 寿命測定ヒストグラム . . . 43

表 目 次

1 ハードウェア記述言語の種類と特徴 . . . 10
2 LED の状態 (○が点灯・×で消灯) . . . 39

1 導入

この章では本研究の背景を明らかにし、目的を明確にします。

1.1 高エネルギー衝突実験

高エネルギー衝突実験の目的は、高温・高エネルギー密度状態をつくり、その状態で起こる物理現象を観測することです。スイス・ジュネーブの郊外にフランスとスイスの国境にまたがって建設された LHC(Large Hadron Collider) が作り出す高温・高エネルギー密度場は宇宙がビックバンにより形成されてから、100 万分の 1 秒の状態と同じであると考えられています。この状態を観測することで、宇宙誕生のプロセスを解き明かすことができます。

広島大学クォーク物理学研究室では、LHC の ALICE(A Large Ion Collider Experiment) という実験をしています。

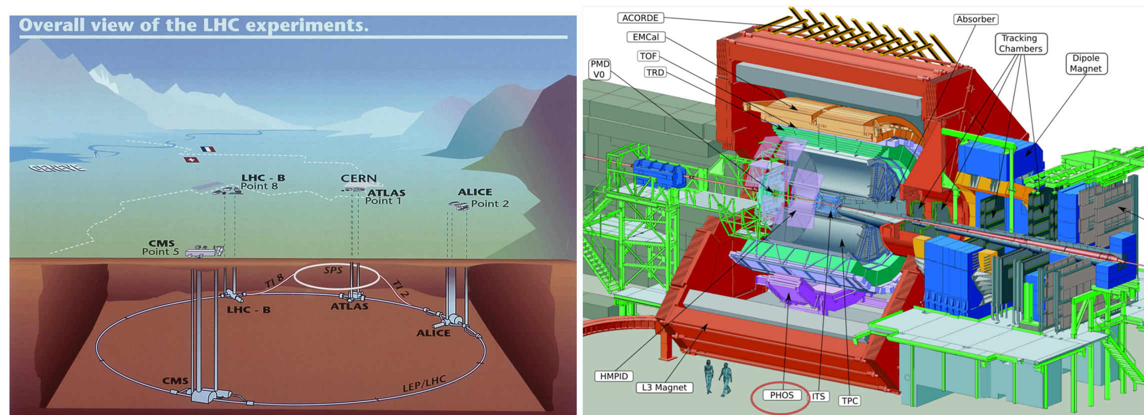


図 1: LHC 全体像と ALICE 全体像 [1][2]

1.2 PHOS

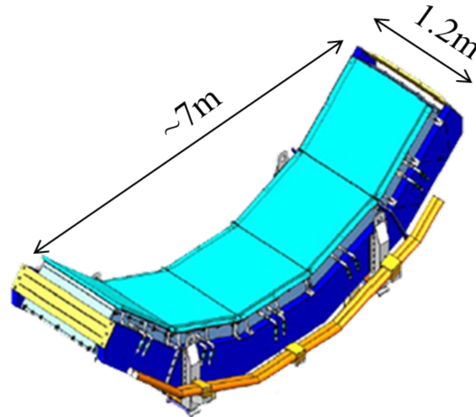


図 2: PHOS 全体像 [3]

PHOS は PHOtonSpectrometer の略で、広島大学グループが開発した光子検出器です。衝突により出てくる終状態の光子は高温・高エネルギー密度場中を通ることにより、衝突の初期やその時間発展の貴重な情報源となります。これを観測することで、極初期宇宙状態を探ることが、PHOS 検出器の役目です。

PHOS は位置分解能・エネルギー分解能が優れています。原子核衝突により生成された多くの粒子ひとつひとつの位置・エネルギーを高い精度で測定できます。入力信号が閾値を超えたかどうかを判別する回路をトリガー回路といいます。仕組みとしては、図 3 のように、PHOS からの信号が FEC ボード (Front End Card:アナログ信号を OR でサンプリング) を介し TRU(Trigger Resion Unit) で入力信号が閾値を超えたかどうかを判別しています。TOR(トリガー・オア) でトリガーからの信号を集め、CTP(Control Trigger Processor.) で ALICE 全体の検出器のトリガーを管理し、イベントの選別をしています。この結果から、目的に合ったイベントのデータを選び、記録します。

衝突してから CTP へ信号がいくまでの時間を 800ns 以内にしないと、CTP でトリガー信号を管理してくれません。すなわち、アナログ信号の時間幅 350ns と衝突からの到達時間・ ケーブル長などによるシステムの遅延時間 350ns、さらにトリガー回路での処理時間を合わせて 800ns 以内になることを 要求されています。それ故、トリガー回路でシグナルの処理に使える時間は 100ns 以下です。

現在、PHOS では FPGA と呼ばれる新しい集積回路技術を用いてトリガー回路が構築されています。しかし、トリガー回路の処理速度の制約から、検出効率が十分ではありません。

さらに、ALICE は 2018 年のアップグレードでさらなる単位時間当たりの処理すべきデータ量を増やす予定です。それ故、PHOS 検出器にとって読み出し処理速度の向上は必須と言えます。

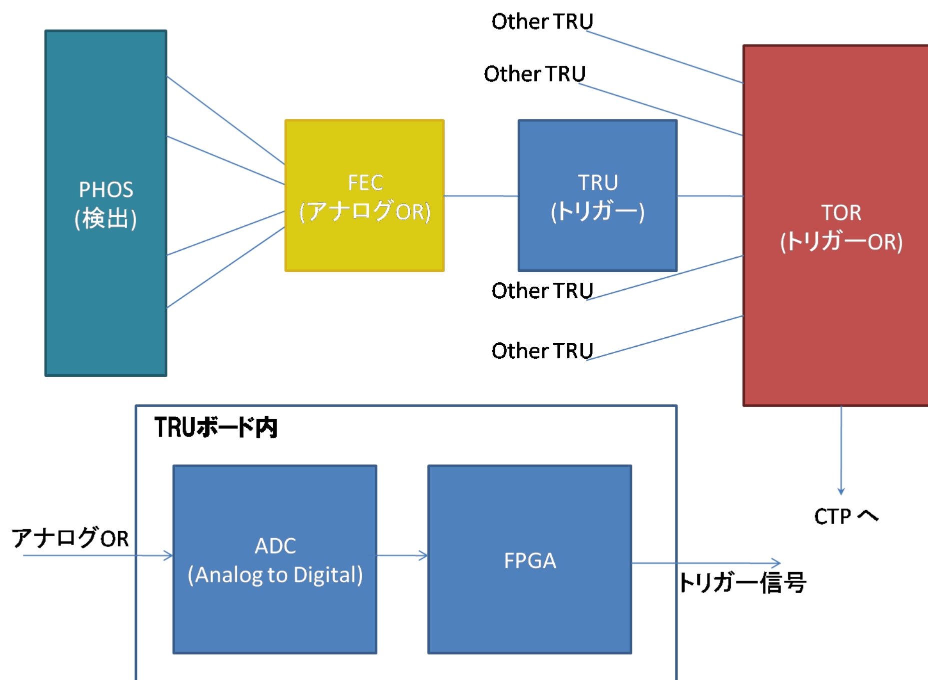


図 3: PHOS 検出器から CTP までの回路図と TRU ボード内

1.3 プログラマブルゲートアレイ (FPGA)

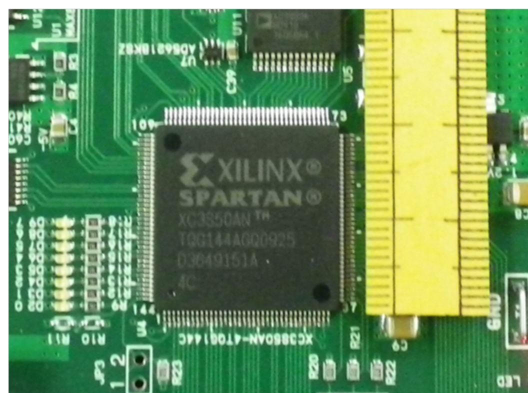


図 4: FPGA

FPGA とは Field Programmable Gate Array の略称です。ゲートアレイとはユーザーの設計によって半導体メーカーが製造するカスタム LSI(Large Scale Integration:大規模集積回路) の一種です。それと同様の機能をフィールド (Field:LSI を使用する現場) でプログラム可能 (Programmable) にしたことから、FPGA と呼ばれています。FPGA はその汎用性に加え、近年の急速なテクノロジー進化に伴い高集積化や低価格化が著しく、極めて将来性の高い集積回路の設計・構築技術です。図 5 のように FPGA は多数の小規模な基本論理ブロックを、縦横無尽に張り巡らされた配線で相互接続した構造をしています。I/O セルとは Input/Output セルのことで、信号の入力と出力を処

理しています。設計した回路情報を所定の方法で書き込むことによって、FPGA 内部の回路が確定し、LSI として必要な機能で動作させます。FPGA は論理回路であれば、搭載されている内部素子や配線の量と性能が許す限りどのような回路でも作ることが可能です。
FPGA を用いるメリットとして以下のようなことが挙げられます。

- 別注しなくても用途によってハードウェア構成を変化出来る。
- 高額な開発費が不要。
- 手元で動作させられる。

試行錯誤をしながら高処理速度・高精度のアルゴリズムを効率的に探索するには最適です。

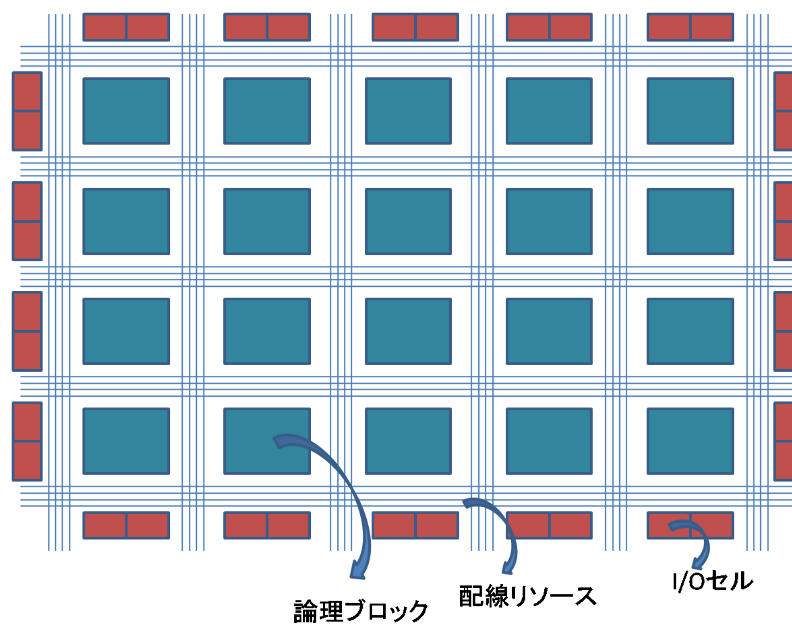


図 5: FPGA 内部構造

FPGA がよく用いられるものとして、

- ネットワーク・ルータ (データ伝送処理)
- 携帯電話基地局 (音声・画像処理・エラー訂正処理)
- 液晶テレビ・プロジェクター・カーナビゲーション (画像処理)
- 信号機 (信号制御)
- バーコードリーダー (画像処理・データ伝送)
- 複合機 (画像処理・信号処理)

が挙げられます。画像処理やデータ伝送の機能によく使われていることが分かります。これはFPGAが「高速かつ大量に流れるデータを、多チャンネル並列でリアルタイムに処理できる」からです。検出器からくる大量のデータをリアルタイムに処理する読み出し回路に最適の技術と言えます。

HDL 種類	特徴
Verilog HDL	C 言語に似た文法体系。ライブラリの充実、採用実績多数などの理由で、実質的な業界標準。
VHDL	米国国防省を中心に、いち早く標準化された HDL。言語使用が豊富でかつ厳格。
UDL/I	日本電子工業振興会の標準化委員会で採用された純国産の標準 HDL。実用的な処理系 (シミュレーター, 論理合成ツール) がないため、利用されていない。
SFL	論理合成ツール PARTHENON(NTT) 用の HDL。同期回路に用途を限定したため、記述や処理系が簡潔。

表 1: ハードウェア記述言語の種類と特徴

1.4 Velilog HDL

HDL(ハードウェア記述言語:hardware description language) は文字通りハードウェアを記述するための言語です。ここでいうハードウェアとは、主に論理回路のことです。HDL 自体は、特に新しい概念ではありません。しかし近年の急速なテクノロジー進歩に伴った高集積化や低価格化の実現により、汎用性が高まりました。また一方で、回路図による論理回路設計に限界が見えはじめ、より効率的な設計手法を必要とされていました。そんな回路図に代わる設計手法が HDL 設計なのです。

従来の回路図を使った設計では

- 難しい論理式
- 回路図入力に手間がかかる
- 回路変更が大変
- 設計者以外内容を理解しずらい

というデメリットがありましたが、HDL を用いることで、回路図入力もテキストで簡単に入力できるようになりました。テキスト形式なので変更も容易で、内容も理解しやすくなりました。これにより設計の再利用が容易になりました。

HDL の種類として表 1 に示すような多くの種類があります。これらの言語の中で現在よく用いられているのが、”Verilog HDL”と”VHDL”です。

Verilog の VHDL に対する優位な点は、

その 1 C 言語をベースにした文法体系であり、記述が簡潔。

その 2 シミュレーション向けの記述が充実している。

その 3 言語体系が簡素なためシミュレータが高速。さらに、VHDL のようなパッケージ・ファイル (データ・タイプや演算子をすべて定義したファイル) が不要なため、シミュレータの負担が軽い。

その 4 シミューレーター用のライブラリやツール類が充実している。

が挙げられます。
　VHDL は記述能力が高いが、その反面文法が難しい言語です。

1.5　本研究の目的

　PHOS 検出器のトリガー回路のさらなる高処理能力化が切望されています。トリガー回路の開発に FPGA を用いることによって、高処理速度・高精度のアルゴリズムを効率的に模索することを長期的な目標としています。それ故、本研究では、技術試験を通し培った FPGA の基礎知識を使い、ナノ秒スケールの処理速度を必要とするミューオン寿命測定実験を行うことで、FPGA 技術の有効性と適用条件を明確にすることを目的としています。

2　FPGA 動作確認

この章では FPGA 技術習得過程を明らかにし、FPGA の動作を確認をします。

2.1　開発ツール

開発ツールとは、HDL で記述した回路を FPGA に書き込むためのものです。今回 XILINX 社が提供する FPGA を使うので、開発ツールも XILINX 社が提供している”ISE Project NavigatorVer.12.4 (M.81d)”を使用します。

ISE が行う作業は、

HDL ソースコードの作成

-今回の HDL として使用する VerilogHDL ソースコードの拡張子は”.v”。

ピン配置の指定

-HDL で設定した input と output に対応する FPGA ボード上のモジュール (LED やクロック、スイッチなど) の配置を指定すること。制約 (コンストレイン) ファイルを用いる。拡張子は”.usf”。

論理合成:Synthesis

-HDL ソースコードを論理回路に変換する論理合成を行う。

実装 (テクノロジ・マッピングと配置配線):Implementation

-論理合成により作成した回路情報を FPGA の内部構造に割り当てる必要がある。これをテクノロジマッピング (mapping) と配線配置 (place and routing) と言う。

ビットストリームの作成 (Generate Programming File)

-実装処理によって生成された FPGA の回路情報を、FPGA に書き込みを行うデータに変換する。このデータがビットストリーム。拡張子は”.bit”。

FPGA へのダウンロード

-パソコンと FPGA を繋ぎ、ビットストリームを FPGA にダウンロードする。

です。手順を図 6〜図 9 に示します。

1. ISE Project Navigatorでは、設計を「プロジェクト」単位で管理します。
ISEを立ち上げます。プロジェクトの新規作成を行うため、左上の"File"の"New project"を選択してください。

2. まず、プロジェクト名と、保存するフォルダを指定します。

3. 開発対象のFPGAの型番を選択します。

4. 確認画面です。よければ"Finish"を選択してください。

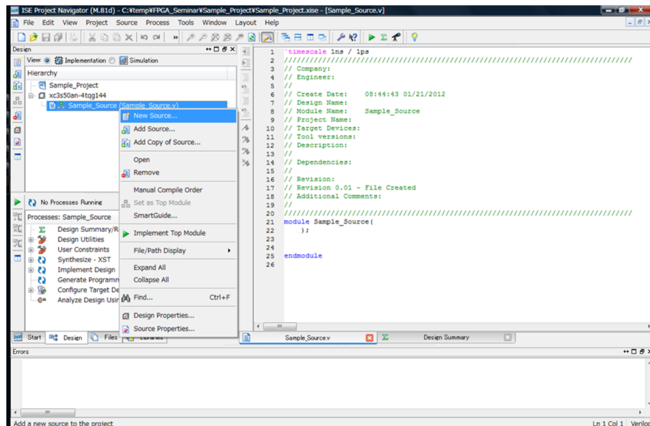
5. 新しいプロジェクトが作成されました。
次にVerilig HDL のソースコードを作成するため、"project"の"New Source"を選択してください。

6. 作成するソースコードの種類を選択します。今回はVerilog HDLによる設計なのでVerilog HDL を選択し、ファイル名と保存場所を指定します。

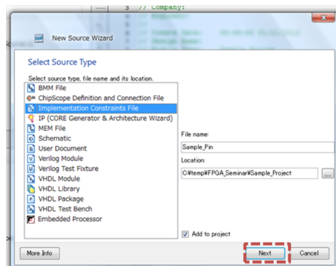
7. 開発対象のFPGAの型番を選択します。

8. 確認画面です。よければ"Finish"を選択してください。

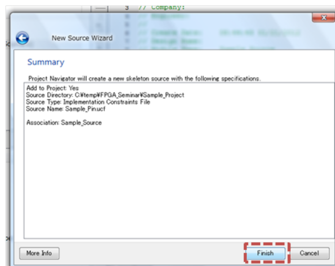
図 6: ISE 手順 1



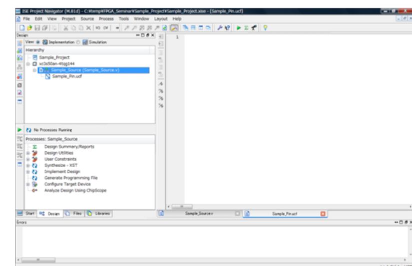
9. 新しいVファイルが作成されました。Vファイルとは Verilig HDLを書き込め為の形式の総称とします。図の右側にあるものがそれです。
　つぎにピンの配置設定を行うソースコードを作成するため、ピン配置を設定したいVファイルを右クリックし、"New Source"を選択してください。
　ピンの配置とは、論理合成で生成された回路を指定のFPGAで動作させるために、入出力信号をFPGAのI/O(Input/Output)ピンに割り当てることです。



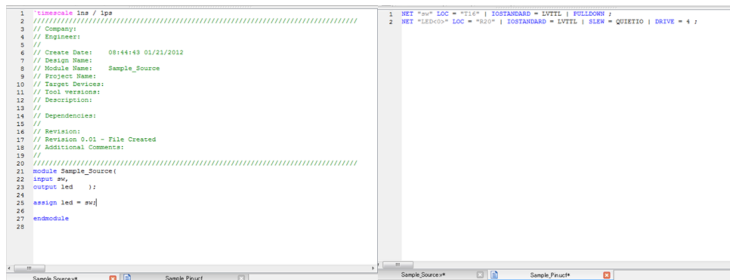
10. 作成するソースコードの種類を選択します。今度はピン配置を行う制約ファイルの作成のため、"Implementation Constraint File" を選択し、ファイル名と保存場所を指定します。



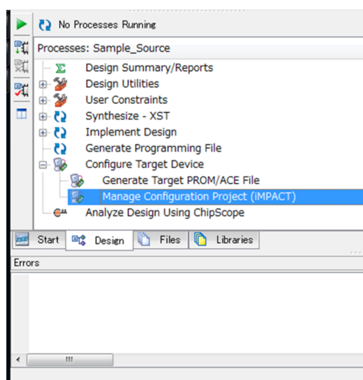
11. 確認画面です。よければ"Finish"を選択してください。



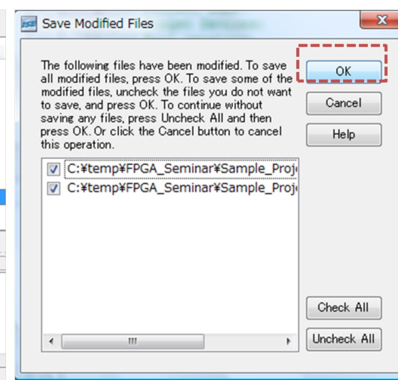
12. 新しい制約(コンストレイン)ファイルが作成されました。制約ファイルの拡張子は.ucfです。



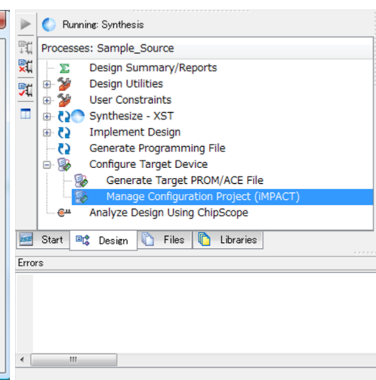
13. 左図のようにVファイルと制約ファイルを作成します。Vファイルの書き方については以降の節で説明します。制約ファイルの書き方は、各FPGAボードの説明書等を参考にしてください。ピンの配置はボードによって異なります。



14. "Manage Configuration Project(iMPACT)"をダブルクリックします。

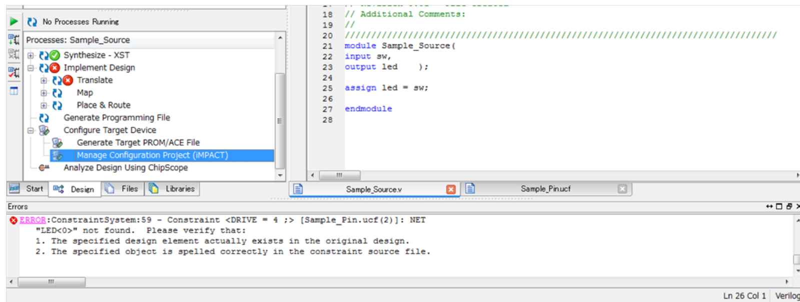


15. 作成したVファイルと制約ファイルを保存するか聞かれます。よければ"OK"を選択します。

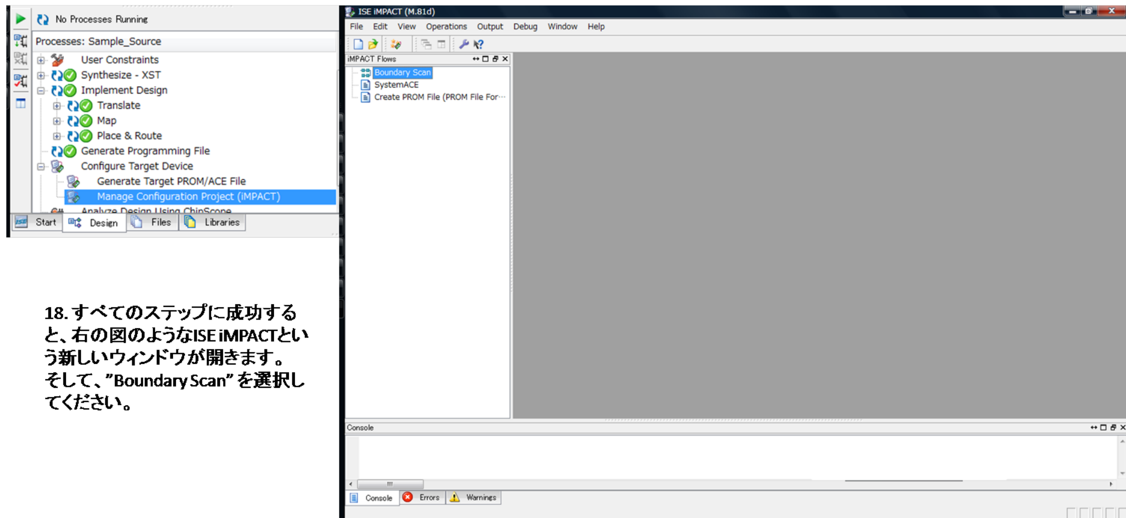


16. Ⅰ:論理合成(Synthesize)
　Ⅱ:実装(Implement)
　Ⅲ:bitファイル作成
　　(Generate Programming File)
　の順番で行います。

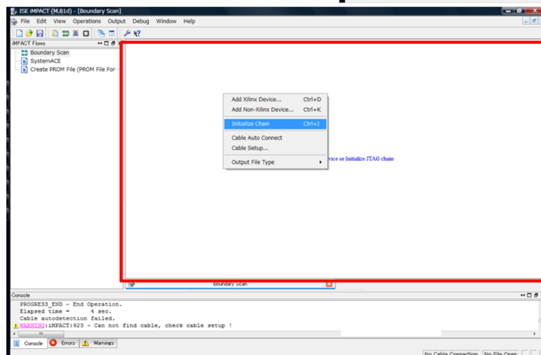
図 7: ISE 手順 2



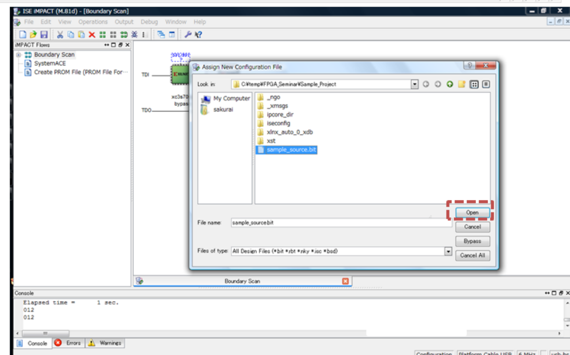
17. 作成したファイルに不備があると、左の図のように赤の×印が表示され、下にはエラーであると認識された過程が表示されます。



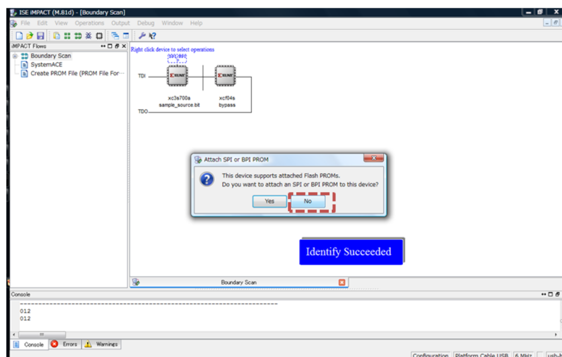
18. すべてのステップに成功すると、右の図のようなISE iMPACTという新しいウィンドウが開きます。そして、"Boundary Scan" を選択してください。



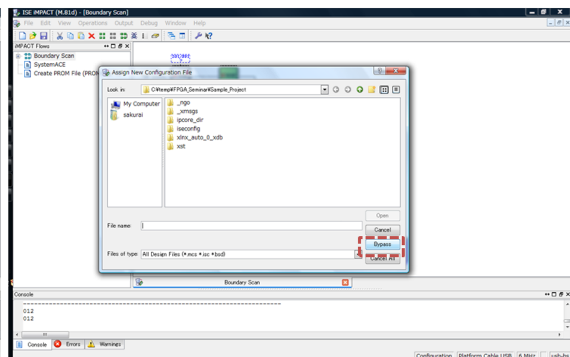
19. 赤枠内で右クリックして、Initialize Chain を選択します



20. 使用するビットファイルを指定します。

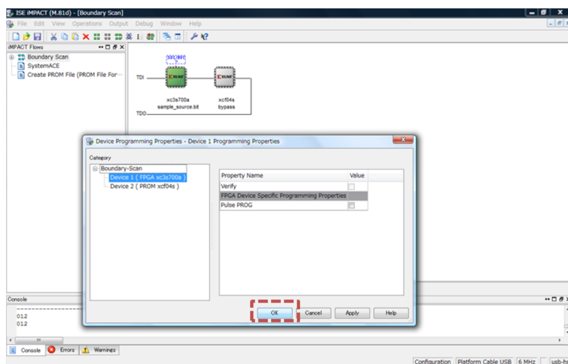


21. 書きこむ場所の確認です。今回の手順では"NO"を選択します。

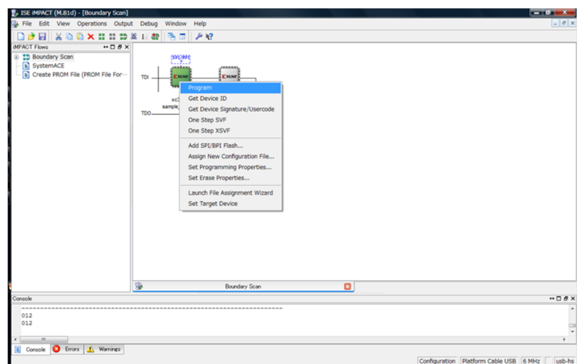


22. 未使用デバイスの設定です。"Bypass"を選択します。

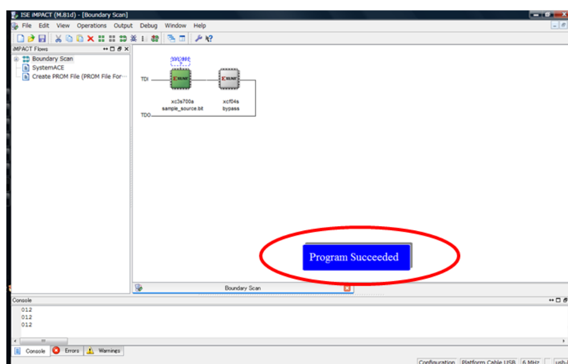
図 8: ISE 手順 3



23. 書き込みパラメ―タ―を変更できます。”OK”を選択してください。



24. プログラムするチップの図上で右クリックをする。”Program”を選択します。



25. Program Succeeded と表示されれば成功です。動作確認を行ってください。

図 9: ISE 手順 4

2.2 XILINX SPARTAN-3A Starterkit

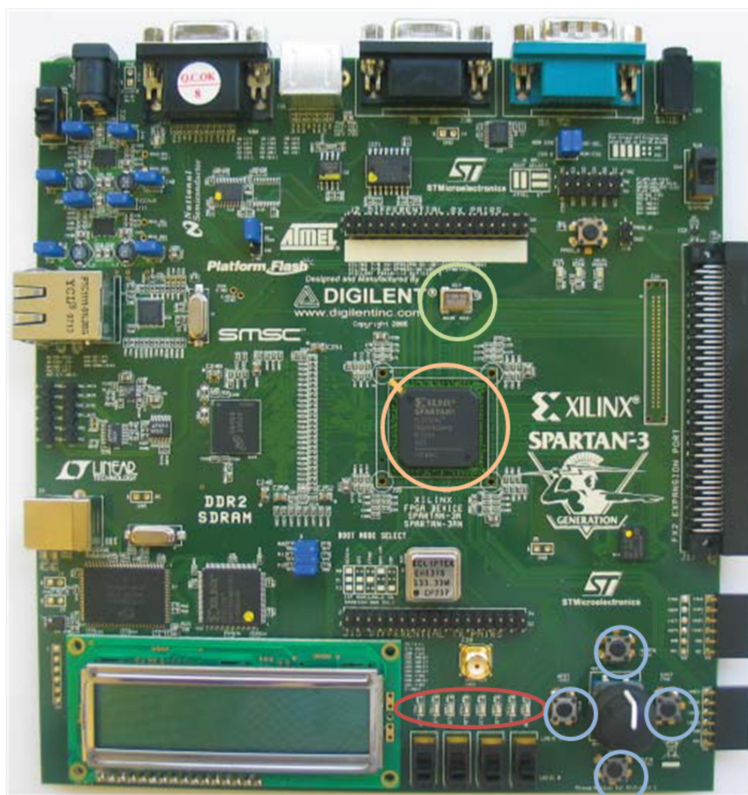


図 10: SPARTAN-3A Starter kit Board

FPGA は XILINX 社の Spartan-3A (XC3S700A-FG484) を使用しています。クロック周波数は 50MHz です。今回使う機能は図 10 に印をつけた、ボタンスイッチ・LED・クロックです。(赤:LED, 青:スイッチ. 緑:クロック, 橙:FPGA) 以下の節では、XILINX SPARTAN-3A Starterkit を用いた FPGA の基本的な動作を確認します。

2.2.1 信号を扱う

はじめに、FPGA ボード上に配置されているスイッチの on と off で、LED が点灯・消灯する回路をつくりました。
Velrilog HDL では、ひとまとまりの機能を「モジュール」として扱います。図 11 上の HDL1 行目の

module モジュール名 (入出力に使われる信号名のリスト);

で FPGA に入出力する信号を宣言します。Verilog の規則で、英字またはアンダ・スコア (__) で始まり、英数字やアンダ・スコアまたはドル ($) による文字列を使用できます。(入出力に使われる信号名のリスト) 内の信号の区切りは、コンマ (,) を使います。Verilog HDL では文の終りにセミコロン (;) を付けます。

入出力に使われる信号名のリストで、信号名は記述しましたが、その信号が FPGA に対して入力信号か出力信号かが分かりません。そこで、input(入力)、output(出力)、inout(双方向) を使って信号の方向を宣言します。

assign 文は、信号の代入を行う記述です。右辺の信号を、左辺の信号に代入します。つまり、今回の回路では、SW 信号が on のとき LED 信号が on に、off のとき off が代入されています。

結果、スイッチを押したとき LED が点灯、スイッチを離すと LED が消灯しました。入力信号に応じて出力を変化させれることに成功しました。

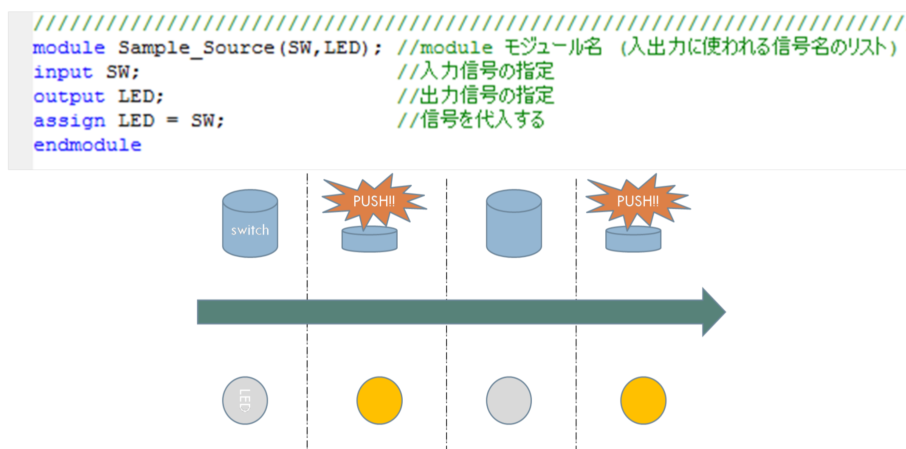


図 11: LED 点灯 HDL・動作

次は、スイッチを押すたびに LED の点灯・消灯が切り替わる回路を設計します。LED が点灯している時にスイッチを押すと消灯し、消灯している時にスイッチを押すと点灯するという動作を実現します。目指す動作を図 12 下に示します。論理回路は AND ゲートや OR ゲートの組み合わせで表現され、入力信号の変化とともに即座に出力が変化するもの (組み合わせ回路) と、クロック信号などのタイミングで変化するもの (順序回路) の 2 種類に大別できます。そこで、各出力信号がどちらのタイプになるのかを明示する必要があります。その時使われるのが、組み合わせ回路であることを示す"ネット型"と、順序回路であることを示す"レジスタ型"の 2 種類です。Verilog HDL ではネット型は"wire"、レジスタ型では"reg"を使って宣言します。

今回の回路では `led_status` をレジスタ型で宣言しています。レジスタは記憶を行うモノの総称です。特定の入力信号に変化があった時にだけ出力を変化し、それ以外のときは入力信号が変化しても出力の状態を保持し続けます。

ネット型の信号は、状態を保持することはできません。入力の変化が直ちに出力に伝わる信号であり、assign 文の左辺で使われます。逆にレジスタ型で宣言した信号は、assign 文の左辺には使用できません。

ただし、レジスタ型で宣言しただけで出力を保持することはできません。レジスタ型の信号は、always 文と一緒に使うことで、状態を保持する回路になります。always 文の構文は always@("信号"名) begin "動作" end　です。"信号"が来た時に"動作"を行います。信号を指定するときに posedge と付け加えるのは、"信号"の立ち上がりエッジ、すなわち off から on に変化したときを指定しています。信号を立下りで見たい時は negedge と付けます。

今回行う動作は"信号"が来た時、レジスタ `led_status` の論理を反転するという動作になります。

結果、スイッチを押すたびに LED の点灯・消灯が切り替えれました。入力信号に応じて出力を変化・保持させることに成功しました。

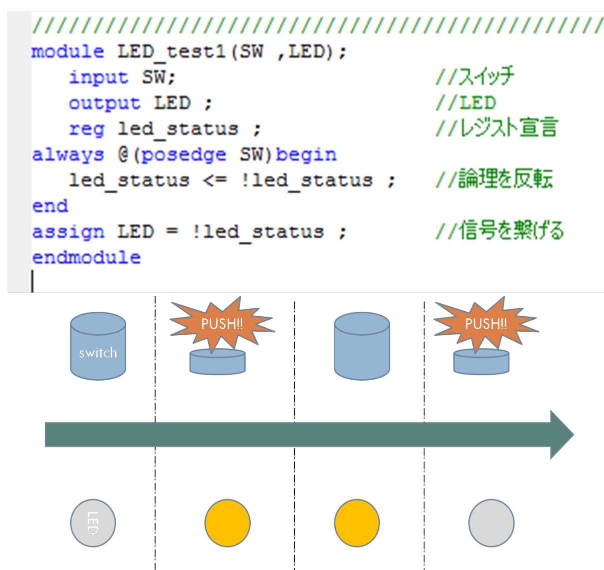


図 12: 切り替え HDL・動作

2.2.2　回数をカウントする

FPGA ボードには 8 つの LED が搭載されているので、図 13 のようにスイッチを押した回数で LED が点灯するところが変わる回路を構成します。この回路では、0〜7 回まで、スイッチを押した回数をカウントできます。

[7:0] というのは「7 から 0 までひとまとまりの信号である」という意味で、LED[0] から LED[7] までを一度に宣言したことになります。parameter とはパラメーター宣言で、ある名前の変数に対して数値を割り当てるときに使います。何度も使うような定数に変更があったとき、すべてを書き直すのはたいへんです。parameter で定義しておけば 1ヶ所の修正ですみます。ここで、`init_value` という名前に 8'b0001 という数値を割り当てています。「4'」が 4 ビット幅、「b」が 2 進数、「0001」が実際の値という意味です。つまり、定義した数値は 4 桁の 2 進数で 0001 ということになります。数値は 2 進数 (b または B) のほか、8 進数 (o または O)、10 進数 (d または D)、16 進数 (h または H) の表記があります。
always　@(〜or〜) としたときの"or"はイベント式中の 2 つのイベントを繋げています。今回の場合、「プレイ信号の立ち上がりエッジか、リセット信号の立ち上がりエッジのいずれかが検出された」に always 文の中を実行することになります。

always 文のなかで、

「リセットが押されたら初期値に戻す。」
「プレイが押されたら LED を変更。」

という動作を行う必要があります。

条件によって異なる動作をさせたい場合は if 文を使います。if 文の構文は

```
if(条件式) begin
条件が一致したときの動作 ;
end else bebin
条件が一致しないときの動作 ;
end
```

です。if 文は必ず always 文のなかで使わなければなりません。このとき、条件判断を行うタイミングを always 文の括弧内に条件式として記述する必要があります。

点灯する LED が左に移動するが、右に"シフト動作する"と記述した行です。隣の信号に値を渡しているのが分かると思います。これがシフト回路です。

結果、スイッチを押すたびに LED の位置がシフトしました。リセットスイッチも作動しました。条件付けをした動作に成功しました。

```
////////////////////////////////////////////////////////////////////////
module LED_test2(
RST_SW,
PLAY_SW,
LED );
input   RST_SW;                                          //リセット信号
input   PLAY_SW;                                         //プレイ信号
output   [7:0]   LED;                                    //LED
parameter   init_value = 8'b00000001;                    //初期値
reg   [7:0] LED_status;                                  //LEDの状態
always  @(posedge PLAY_SW or posedge RST_SW)
begin
   if (RST_SW == 1'b1) begin                             //リセット信号がonになったら
   LED_status <= init_value;                             //初期値を代入
   end else begin                                        //それ以外
   LED_status[0] <= LED_status[7] ;                      //シフト動作する
   LED_status[1] <= LED_status[0] ;                      //シフト動作する
   LED_status[2] <= LED_status[1] ;                      //シフト動作する
   LED_status[3] <= LED_status[2] ;                      //シフト動作する
   LED_status[4] <= LED_status[3] ;                      //シフト動作する
   LED_status[5] <= LED_status[4] ;                      //シフト動作する
   LED_status[6] <= LED_status[5] ;                      //シフト動作する
   LED_status[7] <= LED_status[6] ;                      //シフト動作する
   end
end
   assign LED[7:0] = LED_status[7:0] ;                   //信号を繋げる
endmodule
```

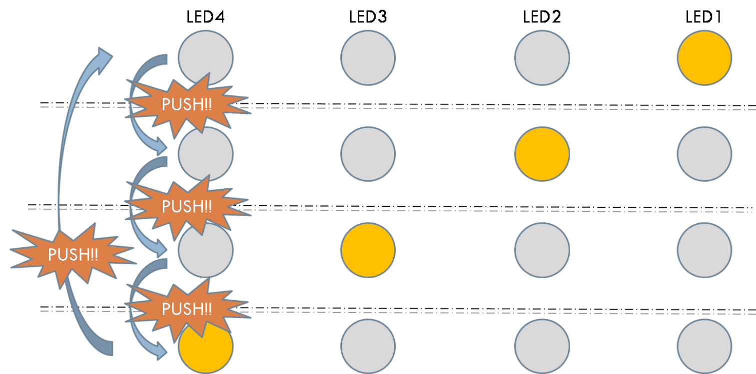


図 13: 押しカウント HDL・動作

今度は 8 つの LED を使って、図 14 のようにスイッチを押した回数を 2 進数で表示する回路です。$2^8 = 256$ まで数えることができます。つまりスイッチを押した回数を 0～255 回まで数えます。255 を超えた時には 0 に戻ります。また、リセットボタンを押した時もカウント数にかかわらず 0 に戻ります。

$$\text{counter7} <= \text{counter7} + 1\ ;$$

では、counter7 値に+1 を算術的に加算し、改めて counter7 に代入します。

結果、スイッチを押した回数を数えることができました。

```
////////////////////////////////////////////////////////////////////
module LED_test4(
RST_SW,
PLAY_SW,
LED     );
input   RST_SW;                                   //リセット信号
input   PLAY_SW;                                  //プレイ信号
output   [0:7] LED;                               //LED
reg   [7:0]   counter7 ;                          //カウンター
always @(posedge PLAY_SW or posedge RST_SW)
begin
   if ( RST_SW == 1'b1 ) begin                    //リセット信号がonになったら
        counter7 <= 8'b00000000 ;                 //リセットする
   end else begin                                 //それ以外は
      counter7 <= counter7 + 1 ;                  //カウントする
   end
end
   assign LED[7] = counter7[7];               //カウント数に応じてLEDを光らせる
   assign LED[6] = counter7[6];               //カウント数に応じてLEDを光らせる
   assign LED[5] = counter7[5];               //カウント数に応じてLEDを光らせる
   assign LED[4] = counter7[4];               //カウント数に応じてLEDを光らせる
   assign LED[3] = counter7[3];               //カウント数に応じてLEDを光らせる
   assign LED[2] = counter7[2];               //カウント数に応じてLEDを光らせる
   assign LED[1] = counter7[1];               //カウント数に応じてLEDを光らせる
   assign LED[0] = counter7[0];               //カウント数に応じてLEDを光らせる
endmodule
```

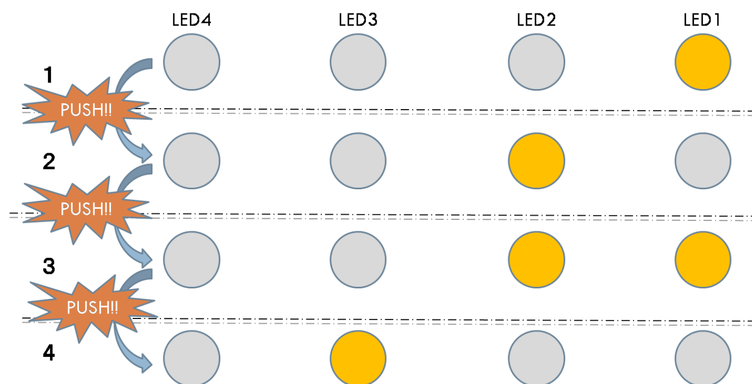


図 14: 2 進数押しカウント HDL・動作

2.2.3 クロックを扱う

クロック信号を実際に発生するのは「クロック発振器」という電子部品です。最近のデジタル機器であれば、ほとんどすべての回路で使われています。多くの FPGA にはあらかじめ搭載されています。スターターキットに搭載されている標準クロックは 50MHz です。つまり、1 秒間に 5000 万回信号の変化が繰り返していることが分かっているため、クロック信号を受信した回数を 5000 万回数えることができれば 1 秒のタイミングを知ることができます。そこでカウンター回路で 1Hz の信号を生成して、その生成された信号が来たとき、図 15 のように LED を光らせることができます。

パラメーター宣言では、今回 5000 万回を数えたいので、”49’h02FAF080” を与えます。「49’」が 49 ビット幅で、「h」が 16 進数、「02FAF080」が 16 進数による 5000 万回の値です。

初めの always 文は 50MHz のクロックから 1 秒のタイミングを作る為の記述です。49 ビット幅の信号 `sec_cnt` を 0 からクロックの立ち上がりエッジ毎に+1 ずつ加算していくと、`sec_cnt` が 5000 万 (49’h02FAF080) になったときが丁度 1 秒にとなります。そこで、`sec_cnt` が”49’h02FAF080” になったら、`sec1_flag` を on にします。同時に `sec_cnt` は再び 0 からカウントを始めます。`sec_cnt` が”49’h02FAF080” 以外の時は `sec1_flag` は off とします。

ふたつめの always 文では、sec1_flag が on のとき、toggle_flag の論理を反転しています。
LED は toggle_flag が on のときに点灯し、off のときに消灯します。
結果、1 秒毎に LED の点灯・消灯を切り替えることができました。クロックを使って、自分で信号を出すタイミングを制御することに成功しました。
また、toggle_flag という 1 秒毎に切り替わる信号を 2.2.2 のふたつめで書いた HDL の PLAY_SW 信号の代わりに入力することで、2 進数で時間をカウントすることができます。

```
//////////////////////////////////////////////////////////////////////
module LED_test3(
CLK,
LED    );
input  CLK ;                                  //クロック信号(50MHz)
output  LED ;                                 //LED
wire  LED ;                                   //
parameter  F50M000_cnt= 49'h02FAF080 ;        //1秒(=20ns*50000000)
reg  [48:0] sec_cnt ;                         //カウンター
reg          sec1_flag ;                      //1秒のフラグ
reg          toggle_flag ;                    //on,offを切り替えるフラグ
always @(posedge CLK)begin                    //クロック信号来たとき
   if ( sec_cnt == F50M000_cnt ) begin        //もしカウンターが1秒間カウントしたら
      sec_cnt  <= 49'h00000000 ;              //カウンターリセット
      sec1_flag <= 1'b1 ;                     //1秒のフラグを立てる
   end else begin                             //それ以外のとき
      sec_cnt <= sec_cnt+1 ;                  //カウントする
      sec1_flag <= 1'b0 ;                     //1秒フラグは立てない
   end
end
always @(posedge CLK )begin                   //クロック信号が来たとき
   if (sec1_flag == 1'b1) begin               //もし1秒フラグが来たら
      toggle_flag <= !toggle_flag ;           //on,offを切り替えるフラグを反転
   end
end
   assign LED = toggle_flag ;                 //信号を繋ぐ
endmodule
```

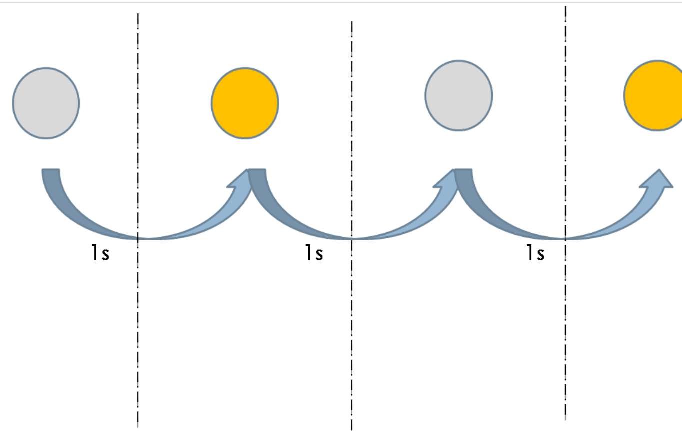


図 15: クロック HDL・動作

2.3 KEK Seminer board

KEK(高エネルギー加速器研究機構) が FPGA を学ぶために設計した FPGA ボードです。FPGA は XILINX 社の Spartan3AN(XC3S50AN-4TQG144C) を搭載しています。クロックは 50MHz です。NIM 信号の入力・出力を行えるモジュールがそれぞれ 4 つずつ載せてあります。NIM 信号とは立ち上がり・立ち下り時間がはやい深さは 800mV で規格化された信号です。(赤:LED, 青:スイッチ. 緑:クロック, 橙:FPGA, 黄:NIMIN. 紫:NIMOUT)

以下の節では、KEK Seminer board の動作を確認します。

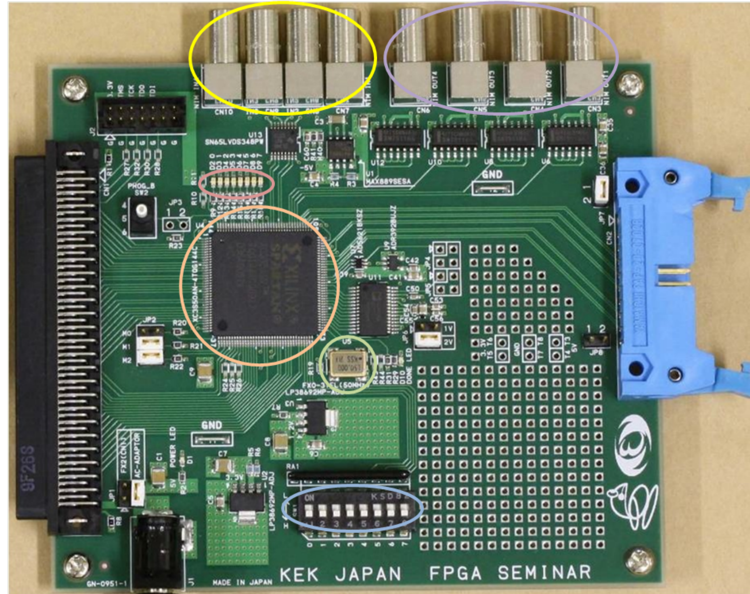


図 16: KEK Seminer Board

2.3.1 NIM 信号を出力する

スイッチを on にした時に、NIMOUT を出力し、オシロスコープで波形を観測しました。用いた HDL は図 17 です。

```
/////////////////////////////////////////
module KEK_NIM_test1_1(
 input SW,
output NIMOUT,
);
assign NIMOUT = SW;  //SWとNIMOUTを繋ぐ
endmodule
```

図 17: SW to NIMOUT HDL

2.3.2 NIM 信号を入力する

ゲートジェネレータ (任意に NIM 信号を出力する機器) から NIM 信号を入力し、NIM 信号が on のときも off のときも FPGA で認識されるか確認するため、on のときはボード上の LED 1 を光らせ、off のときは LED2 が光るようにしました。用いた HDL は図 18 です。

```
/////////////////////////////////////////
module KEK_NIM_test2(
 input NIMIN,
 output [1:2] LED );
assign LED[1] = NIMIN;     //NIM信号が来た時
assign LED[2] = !NIMIN;    //NIM信号が来ない時
endmodule
```

図 18: NIMIN to LED HDL

2.3.3 NIM 信号を入力・出力する

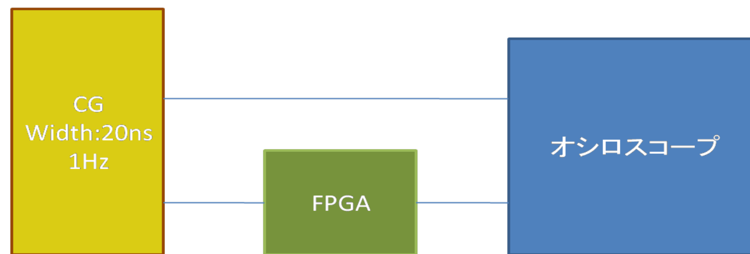


図 19: NIMIN to NIMOUT 回路概略

クロックジェネレータ (設定した時間間隔で NIM 信号を出力する機器) から FPGA へ NIM 信号が入力された時、NIM 信号を出力させました。この時、クロックジェネレータからの出力を図 19 のように 2 本に分け、片方を FPGA を介してオシロスコープへ、もう片方を直接オシロスコープへ入力することで、FPGA を通すことによる信号の遅れ (ディレイ) を調べました。FPGA 内で扱うレジスト型の信号を数回に分けて使う (回路を深くする/階層を増やす) ことによるディレイの変化を調べるために、図 20 の 3 種類の HDL を用いました。

```
//////////////////////////////////////////////////////
module NIM_test_delay(
input NIMIN,
inout CLK,
output NIMOUT
);
reg nimin;                      //1階層目のレジスト信号
always@(posedge CLK)begin
   case(NIMIN)                  //NIMIN信号の状態が
      1'b1 : nimin <= 1'b1;     //onならnimin信号をon
      1'b0 : nimin <= 1'b0;     //offならnimin信号をoff
   endcase
end
assign NIMOUT=nimin;            //nimin信号をNIMOUTに繋ぐ
endmodule
//////////////////////////////////////////////////////
module NIM_test_delay2(
input NIMIN,
inout CLK,
output NIMOUT
    );
reg nimin;                      //1階層目のレジスト信号
reg nimin2;                     //2階層目のレジスト信号
always@(posedge CLK)begin
   case(NIMIN)                  //NIMIN信号の状態が
      1'b1 : nimin <= 1'b1;     //onならnimin信号をon
      1'b0 : nimin <= 1'b0;     //offならnimin信号をoff
   endcase
end
always@(posedge CLK)begin
   case(nimin)                  //nimin信号の状態が
      1'b1 : nimin2 <= 1'b1;    //onならnimin2信号をon
      1'b0 : nimin2 <= 1'b0;    //offならnimin2信号をoff
   endcase
end
assign NIMOUT=nimin2;           //nimin2信号をNIMOUTに繋ぐ
endmodule
//////////////////////////////////////////////////////
module NIM_test_delay3(
input NIMIN,
inout CLK,
output NIMOUT
    );
reg nimin;                      //1階層目のレジスト信号
reg nimin2;                     //2階層目のレジスト信号
reg nimin3;                     //3階層目のレジスト信号
always@(posedge CLK)begin
   case(NIMIN)                  //NIMIN信号の状態が
      1'b1 : nimin <= 1'b1;     //onならnimin信号をon
      1'b0 : nimin <= 1'b0;     //offならnimin信号をoff
   endcase
end
always@(posedge CLK)begin
   case(nimin)                  //nimin信号の状態が
      1'b1 : nimin2 <= 1'b1;    //onならnimin2信号をon
      1'b0 : nimin2 <= 1'b0;    //offならnimin2信号をoff
   endcase
end
always@(posedge CLK)begin
   case(nimin2)                 //nimin2信号の状態が
      1'b1 : nimin3 <= 1'b1;    //onならnimin3信号をon
      1'b0 : nimin3 <= 1'b0;    //offならnimin3信号をoff
   endcase
end
assign NIMOUT=nimin3;           //nimin3信号をNIMOUTに繋ぐ
endmodule
```

図 20: NIMIN to NIMOUT HDL

2.3.4 コインシデンスをとる

NIMINN の 1 と 2 が同時に信号があるときのみ、NIMOUT を出すという回路です。用いた HDL とオシロスコープの波形は図 21 です。

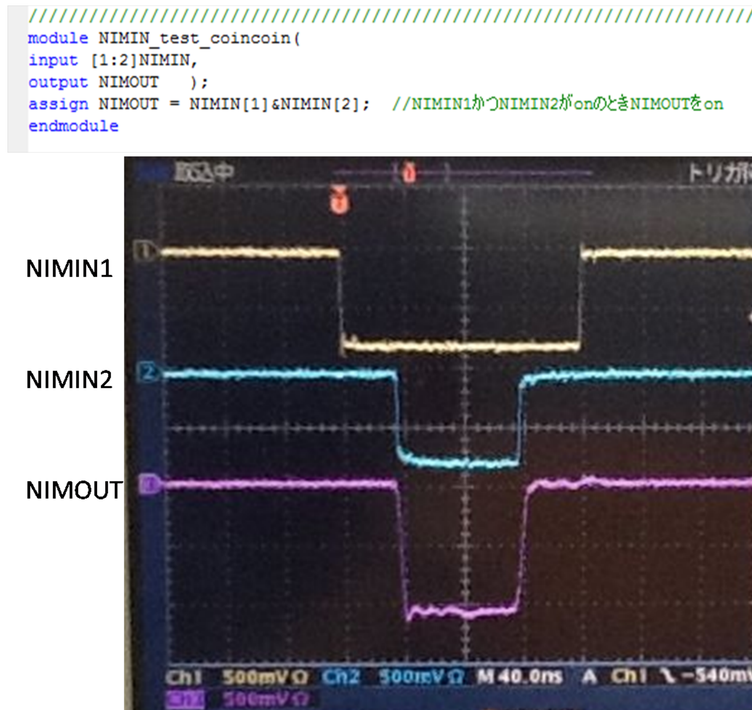


図 21: コインシデンス HDL とオシロスコープ波形

3 FPGA による実証実験

FPGA を使った信号の処理が、ナノスケールの分解能で可能か実証するため、高処理を必要とするミューオン寿命測定実験を行います。この章では、実験の方法と条件を示します。

3.1 NIM,CAMAC 規格

NIM とは Nuclear Instrument Module の略で、NIM 規格とは”放射線測定モジュール標準規格 TID-20893”に準拠した標準規格です。NIM モジュールが用いる信号を NIM 信号といいます。NIM 信号は立ち上がり・立ち下り時間がはやく、ナノ秒スケールで信号を見ても綺麗な長方形の信号です。そのため時間分解能はとてもよい規格です。深さは 800mV で統一されています。
今回の実験で使用するモジュールは、

ディスクリミネーター:Discriminator
-閾値以上の入力信号を NIM 信号に変換。閾値や出力信号の幅は調節可能

クロックジェネレーター:Clock Generator
-指定した周波数で指定した幅の NIM 信号を出力

ゲートジェネレーター:Gate Generator
-入力信号があったとき、指定した幅、指定した遅れで信号を出力

ファンイン/アウト:FAN IN/OUT
-入力信号の OR を出力

コインシデンス:Coincidenc
-入力信号の AND を出力

ディレイ:Delay
-信号をナノ秒単位で遅らすことが可能

です。ディスクリミネーターはアナログ信号など NIM 信号でないものを NIM 規格で扱い時に用います。ディレイはケーブルの長さやモジュール内を通ることによる信号のズレを調節し、信号を正確に処理するために用います。

CAMAC とは Computer Automated Measurement And Control の略で、入力信号をデジタル値に数量化するエレクトロニクスの規格です。パソコンに繋げることで、データをパソコンに送ることができます。

今回の実験ではスケイラー (Scaler) という CAMAC モジュールを使用しました。スケイラーは入力信号の数を数えることができます。

3.2　ミューオンの平均寿命

宇宙からは、主に陽子からなる一次宇宙線という高エネルギー粒子が飛来してきています。一次宇宙線が大気中に入り、大気中原子と反応を起こすことで、構成する粒子が変化し、二次宇宙線になります。二次宇宙線の多くがパイ中間子であり、荷電パイ中間子はミューオンに崩壊します。ミューオンは地上でも観測することが可能です。地上での平均 4GeV のエネルギーをもち、主に上空 15km にて生成されます。

素粒子のなかでもミューオンは電子などの仲間で、電荷を持ち、透過性に優れています。

ミューオンは

$$\mu^- \to e^- + \nu_\mu + \overline{\nu_e} \tag{1}$$

と崩壊し、電子を出します。

放射線崩壊の指数関数法則を使ってミューオンの平均寿命を求めることができます。

それぞれの粒子が単位時間に崩壊する確率が λ(ただし、λ は粒子の年齢とは無関係) である独立な粒子の集合を考えます。ある時刻 t に存在する粒子の数を $N(t)$ とすると、微小時間 dt の間に崩壊する数は次式で与えらます。

$$dN = -\lambda N(t)dt. \tag{2}$$

この式を積分して、ある時刻 t_0 に存在する粒子の数を N_0 とすると、

$$\int_{N_0}^{N} \frac{1}{N}dN = \lambda \int_{t_0}^{t} dt \tag{3}$$

$$(lnN - lnN_0) = \lambda(t - t_0) \tag{4}$$

$$ln(\frac{N_0}{N}) = -\lambda(t - t_0). \tag{5}$$

$t_0 = 0$ の場合、5 式は以下のように書き換えられます。これは放射性崩壊の指数関数法則として知られています。

$$N = N_0 e^{-\lambda t}. \tag{6}$$

時刻 t と $t + dt$ の間の無限に小さい微小時間 dt の間に崩壊する粒子の数は $N\lambda dt$ で表わされます。これらの粒子は時間 t までは存在しているので、時間 t だけ生存する粒子の生存時間の和は $tN\lambda dt$ となります。

したがって、N_0 個の粒子すべての生存時間の和 Sum は $tN\lambda dt$ を $t = 0$ から $t = \infty$ まで積分した値に等しくなり、

$$\begin{aligned} Sum &= \int_0^\infty tN\lambda dt \\ &= \int_0^\infty tN_0\lambda e^{-\lambda t}dt \\ &= N_0\lambda \cdot \frac{1}{\lambda^2}[-e^{-\lambda t}]_0^\infty \\ &= \frac{N_0}{\lambda}. \end{aligned} \tag{7}$$

よって、粒子の平均寿命 $\tau = Sum/N_0$ は、

$$\tau = \frac{1}{\lambda}. \tag{8}$$

これを式 6 に代入すると、

$$N = N_0 e^{-t/\tau}. \tag{9}$$

が得られます。

精密な測定から、ミューオンの平均寿命は $2.197109 \pm 0.000021\mu$ 秒 s であることが知られています。[8]

3.3 宇宙線の観測

勿論私たちは宇宙線の存在を肉眼で確認することはできません。しかし、シンチレーションカウンター (シンチレータ) と光電子増倍管を用いることで、宇宙線の存在を観測することが可能です。

シンチレータとは粒子が入射した際に光を発生させる物質です。シンチレーターの発光原理は、シンチレーション物質に荷電粒子が入ってくると、物質中の電子と電気的な引力や反発力が働きこの影響で電子が励起されます。そして励起された電子が元のエネルギー状態に落ち込むときに光が出るわけです。この光をシンチレーション光と言います。

この粒子が入射されたときに発生した光が光電子増倍管に入ってくると、”光電効果”と呼ばれる現象によって、金属内部の電子が飛び出します。飛び出した電子は、強い電場によって加速されて、”光電子増倍部”に衝突します。光電子増倍部に衝突した電子は、加速で得たエネルギーを使って光電子増倍部内の電子を次々と飛び出させます。この飛び出した電子を 2 次電子といいます、ここで、電子の数が増幅されているわけです。

2 次電子はまた電場によって加速され、次の段の光電子増倍部に衝突し、新たな 2 次電子群を発生させます。これをどんどんと繰り返し、始めは 1 個だった電子は最終段の光電子増倍部に達するとき 1 億個程度にまで増えます。ここまで到達した電子は、アナログシグナルとして外部に読み出されていきます。

アナログシグナルの波形、シンチレータの写真は図 22 のようになります。

図 22: アナログシグナルの波形・シンチレータの写真

3.4 セットアップ

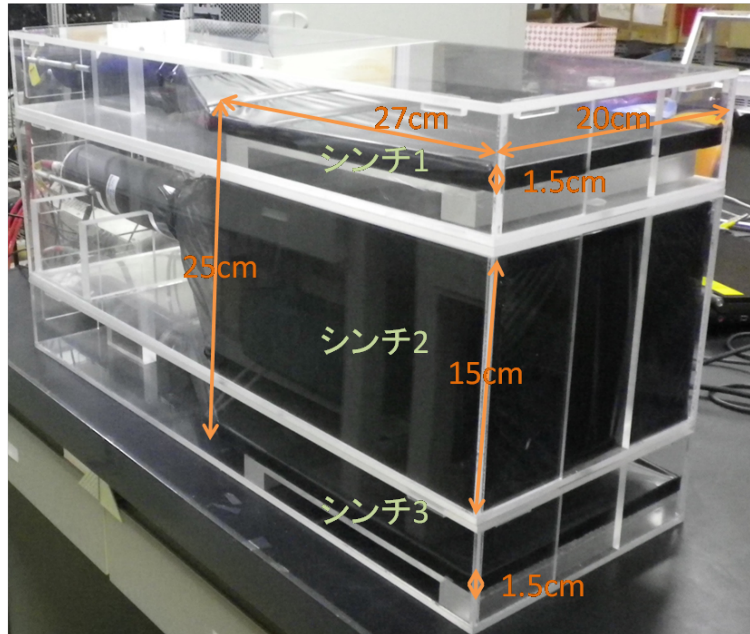


図 23: 実験に使用したシンチレータ

図 23 のシンチレータを上からシンチ 1, シンチ 2, シンチ 3 とします。

シンチ n(n=1,2,3) で粒子が検出されている状態を（シンチ n）とし、シンチ n で粒子が検出されていない状態を $(\overline{\text{シンチ } n})$ とします。

次の条件の時、

$$(\text{シンチ } 1)\bigcap(\text{シンチ } 2)\bigcap(\overline{\text{シンチ } 3}) \tag{10}$$

NIM 出力で 40ns に 1 回信号を出すようにします。

また、次の条件の時、

$$(\overline{\text{シンチ } 1})\bigcap(\text{シンチ } 2)\bigcap(\overline{\text{シンチ } 3}) \tag{11}$$

40ns に 1 回信号がでていた NIM 出力を止めます。図にすると図 24 上のようになります。

この NIM 出力をスケイラーで数えることで、ミューオンがシンチ 2 で止まって崩壊するまでの時間を計ることができます。

この回路を実践するために、FPGA には図 24 下のような記述をしました。はじめの if 文で、count シグナルを on にする条件と、off にする条件を記述しました。2 回目の if 文で、count シグナルが on のときは 1 クロック毎に toggle シグナルを反転させ、count シグナルが off のときは toggle シグナルを off にします。NIMOUT は toggle シグナルを、NIMOUT2 はスケイラーのゲートに入れて、カウントする間隔を決定します。

3種類のNIMOUTをだす。

①:シンチ1∩シンチ2∩$\overline{\text{シンチ3}}$
でシグナルを出す。

②:①が出た時に25MHzで
シグナルを出し続ける。
③が出たら
シグナルストップ。

③:$\overline{\text{シンチ1}}$∩シンチ2∩$\overline{\text{シンチ3}}$
でシグナルを出す。

①:

②:

③:

①:

②:

③:

シンチ1

シンチ2

シンチ3

シンチ1

シンチ2

シンチ3

シンチ1

シンチ2

崩壊

シンチ3

```
module KEK_scintillator2(
 input [1:3]NIMIN,
 input CLK,
output NIMOUT,
output NIMOUT2 );
reg count ;
reg toggle;
always @(posedge CLK)begin
   if (!NIMIN[1]&NIMIN[2]&!NIMIN[3] == 1'b1)      //2がon、1と3がoff
      count <= 1'b0;                                //カウントon
   end else begin
      if (NIMIN[1]&NIMIN[2]&!NIMIN[3] == 1'b1)    //1と2がon、3がoff
         count <= 1'b1;                             //カウントoff
      end
   end
end
always @(posedge CLK)begin
   if (count == 1'b1)begin                          //カウントonのとき
      toggle <= !toggle;                            //toggle反転
   end else begin
      if (count == 1'b0)begin                       //カウントoffのとき
         toggle <= 1'b0;                            //toggleストップ
      end
   end
end
assign NIMOUT = toggle;                             //NIMOUTとtoggle信号を繋げる
assign NIMOUT2 = NIMIN[1]&NIMIN[2]&!NIMIN[3];       //1と2がon、3がoffのときNIMOUT2を出す
endmodule
```

図 24: 実験シグナル概略・HDL

3.5 擬似ミューオン崩壊シグナル測定

ミューオンの寿命を測定する前に、FPGA が正確に作動をし、データを取ることができるのか、NIM モジュールを用いて擬似的にミューオンの崩壊を再現することで、検証してみました。
図 25 上に表したように、クロックジェネレーター (CG) から 1s 毎に信号を出します。ひとつはディレイ:0s、幅:100ns に設定したゲートジェネレーター１ (GG1) のスタートへ、もうひとつはディレイ:1 μ s、幅:100ns に設定したゲートジェネレーター２ (GG2) のスタートへ入力しました。GG1 の出力のうち、ひとつは 5ns のディレイをかけて NIMIN1 へ入力しました。もうひとつをファンイン/アウト (Fan in) へ入力し、GG2 の出力との OR をとりました。Fan in の出力を NIMIN2 へ入力しました。このときの NIMIN 信号は、図 25 下のようになります。黄色が NIMIN1 で、青色が NIMIN2 です。この時、NIMIN3 へは信号を送らないものとします。

NIMOUT2 信号を CAMAC モジュールのスケイラーのゲートに設定し、NIMOUT が 1 イベントごとに何個シグナルを出力するか数えました。

NIMINn(n=1,2,3) 信号が入力されている状態を（$NIMINn$）とし、NIMINn から信号を入力されていない状態を ($\overline{NIMINn}$) とします。

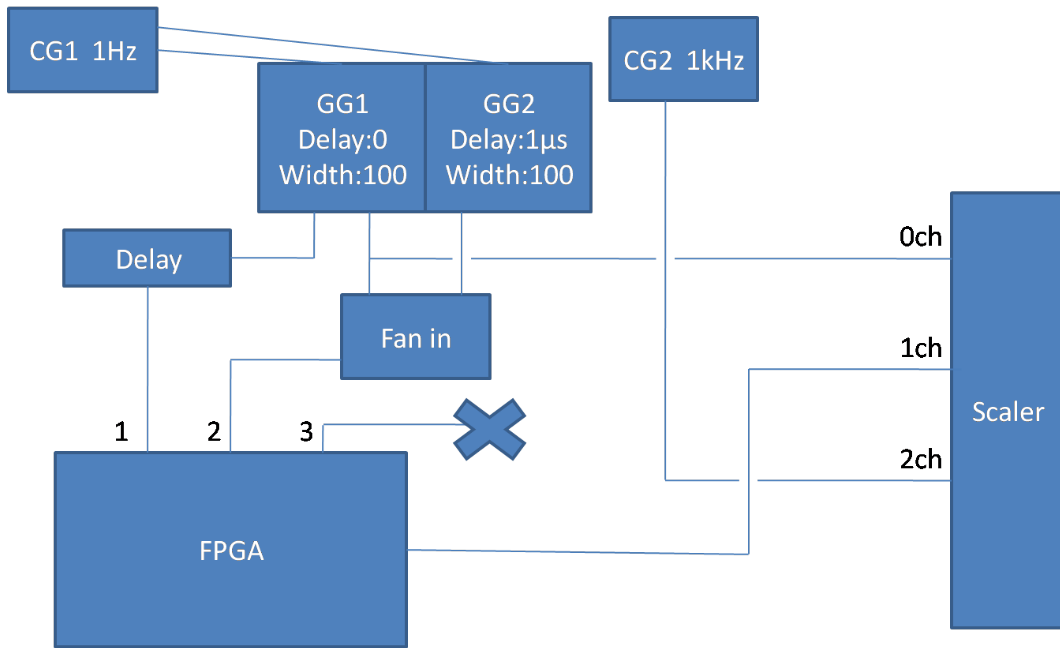
その時 1 イベントとは、

$$(NIMIN1)\bigcap(NIMIN2)\bigcap(\overline{NIMIN3}) \tag{12}$$

でカウントが始まり、

$$(\overline{NIMIN1})\bigcap(NIMIN2)\bigcap(\overline{NIMIN3}) \tag{13}$$

でカウントが終わるまでの事象のことです。1000 秒間測定しました。



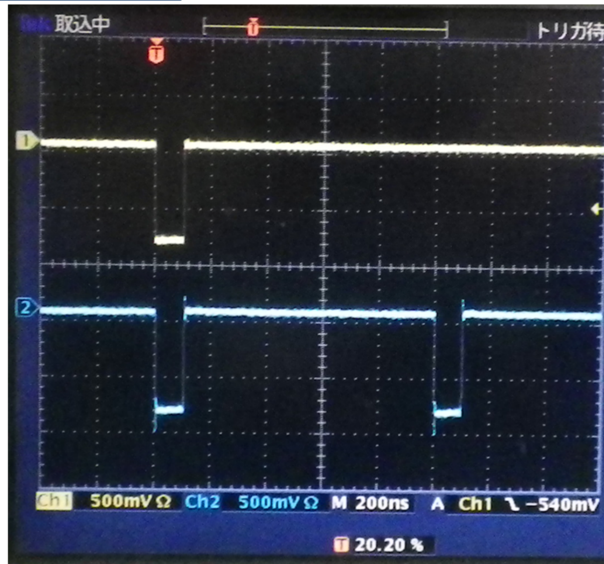


図 25: 擬似実験回路概略・NIMIN シグナル

3.6　シンチレータを用いたミューオン寿命測定

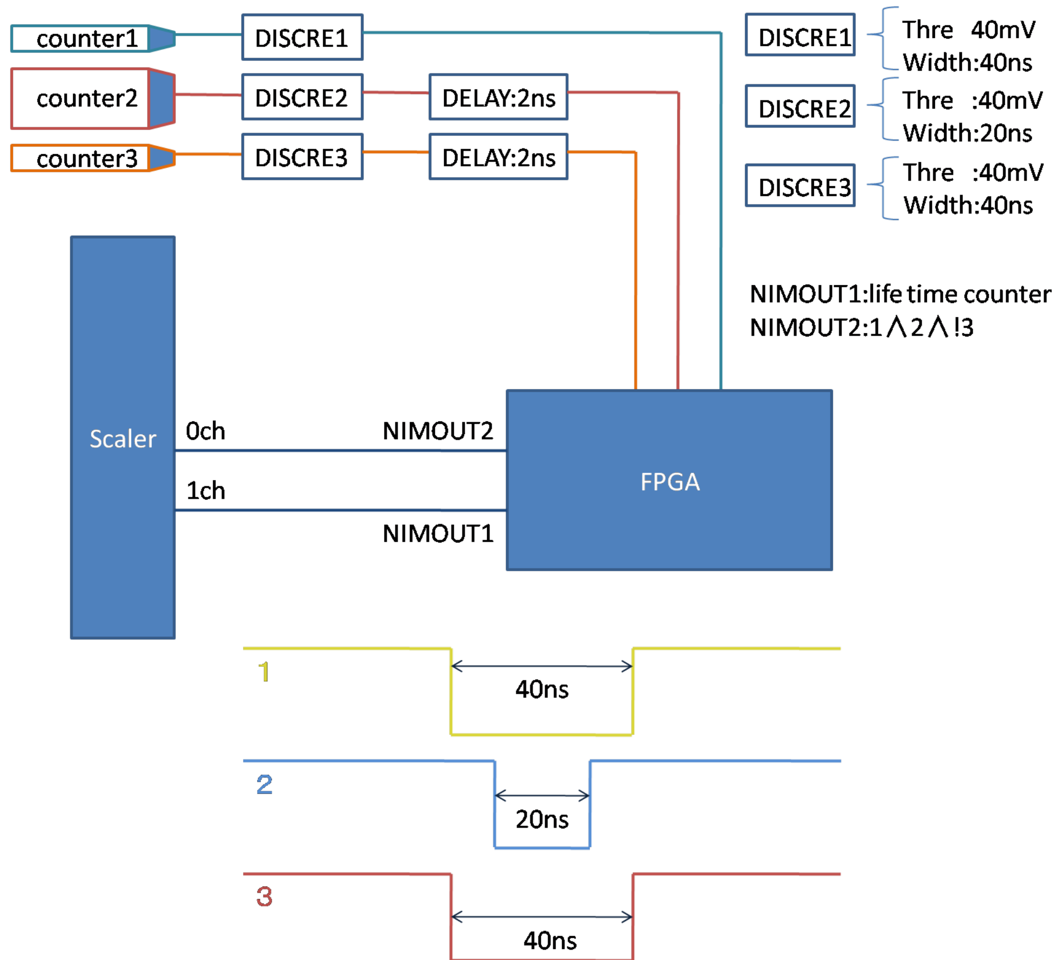


図 26: ミューオン寿命測定回路概略・NIMIN 信号

回路図は図 26 のようになります。シンチレータからの出力はアナログ信号の為、ディスクリミネーターに通して、NIM 信号に変換したのち FPGA に入力します。FPGA に入力します。ディスクリミネーターの閾値はすべて 40mV、幅は NIMIN1 が 40ns・NIMIN2 が 20ns・NIMIN3 が 40ns です。

4 結果・考察

この章では、動作確認・ミューオン寿命測定を通して、FPGA の性能を評価し、高エネルギー衝突実験への有用性を考えます。

4.1 性能評価

図 27: NIMOUT 立ち上がり立ち下り

NIMOUT の立ち上がり立ち下りは、

assign NIMOUT=CLK；

としてクロック信号と NIMOUT を繋げることで、NIMOUT をクロックの 50MHz で出力することで調べました。図 27 からもわかるように、10ns で立ち上がり、10ns で立ち下がっています。

FPGA を通ることによる信号の遅れを検証する実験は、2.3.3 の結果から、図 28 のような波形が観測されました。回路の階層を 1 つのときは 20ns のディレイ、2 つのときは 40ns のディレイ、3 つのときは 60ns のディレイとわかります。このことから、

$$(\text{ディレイ}) = (\text{階層}) \times (\text{クロックの周期}) \tag{14}$$

といえます。

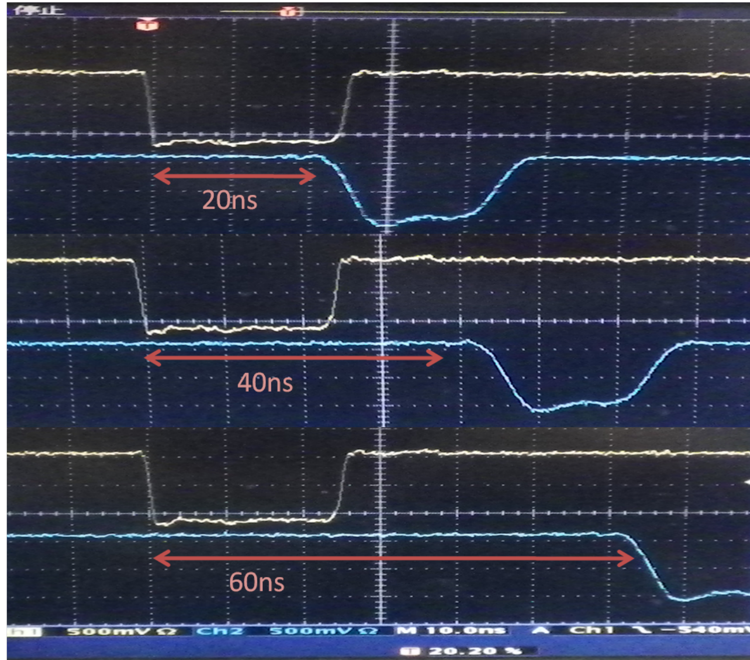


図 28: NIMIN to NIMOUT シグナルの比較

レジスト型の信号を使用するときによく用いられる、

always @(posedge CLK)　begin 〜　end

は信号が on か off かを FPGA 内で決めるのを、内部クロックが立ち上がるときと指定しています。そこで、図 29 上のように内部クロック周波数の 50MHz で幅を 4ns とした入力信号し、信号が認識された時、LED が光るようにしました。ディレイを用いて入力信号をずらしていくことで、どのくらいの精度で信号を認識しているかを調べました。

結果は表 2 のようになりました。周期 20ns 範囲 5ns で LED が光ることから、入力信号のゆらぎを考慮しても、1ns 以下の精度で信号を管理されていることがわかりました。

ディレイ (ns)	LED	ディレイ (ns)	LED	ディレイ (ns)	LED
0	◯	10	×	20	◯
1	◯	11	×	21	◯
2	◯	12	×	22	◯
3	◯	13	×	23	◯
4	◯	14	×	24	◯
5	×	15	×	25	×
6	×	16	×	26	×
7	×	17	×	27	×
8	×	18	×	28	×
9	×	19	×	29	×

表 2: LED の状態 (◯が点灯・×で消灯)

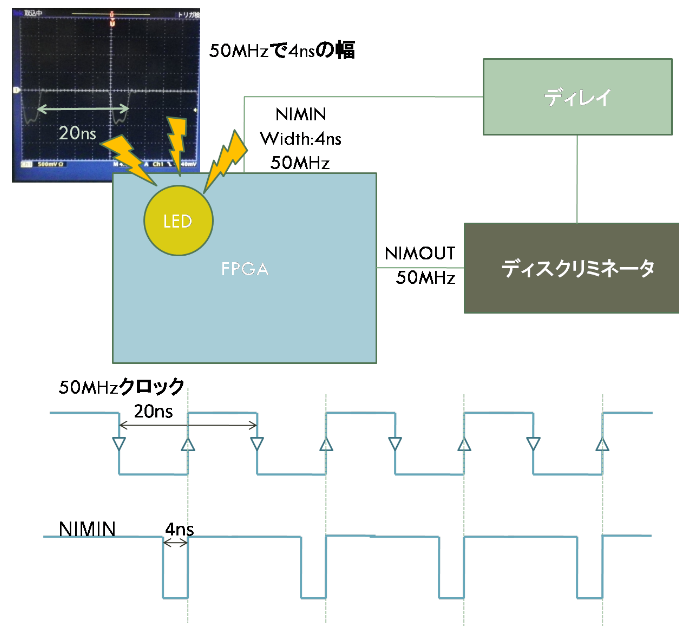


図 29: CLK コインシデンス回路・シグナル概略

4.2 検証実験

オシロスコープで NIMOUT を見たものが図 30 のシグナルです。1 周期 40ns の NIM 信号が 25 個来ていることが見てとれます。

しかし、スケイラーで 100 イベント数えてみると、3 個のシグナルを観測したイベントがランダムに見つけました。この原因を考えるため、NIMIN 信号を見直してみました。

はじめ NIMIN1 と NIMIN2 の信号をぴったり揃えて入力していたのですが、信号がゆらぐ為図 31 の上の信号ように、NIMIN1 が NIMIN2 より早く入力してしまっていたのです。

この場合カウントスタートシグナルの、

$$(NIMIN1)\bigcap(NIMIN2)\bigcap(\overline{NIMIN3}) \tag{15}$$

でカウントが始まってから約 100ns 後に来る、

$$(\overline{NIMIN1})\bigcap(NIMIN2)\bigcap(\overline{NIMIN3}) \tag{16}$$

でカウントが止まってしまいます。この現象を防ぐため、NIMIN1 にかけているディレイを長めにとり、31 の下の信号のように設定し直し、信号のゆらぎによる測定の誤差をなくしました。

図 32 はスケイラーの結果をヒストグラムにしたものです。1000 イベントすべて 25 個シグナルをカウントしています。FPGA で確かに、

$$(NIMIN1)\bigcap(NIMIN2)\bigcap(\overline{NIMIN3}) \tag{17}$$

でカウントが始まり、1μ 秒後の、

$$(\overline{NIMIN1})\bigcap(NIMIN2)\bigcap(\overline{NIMIN3}) \tag{18}$$

でカウントが終わりました。

FPGA を用いたトリガー回路を使って、信号の制御に成功し、1μ 秒を測定できました。

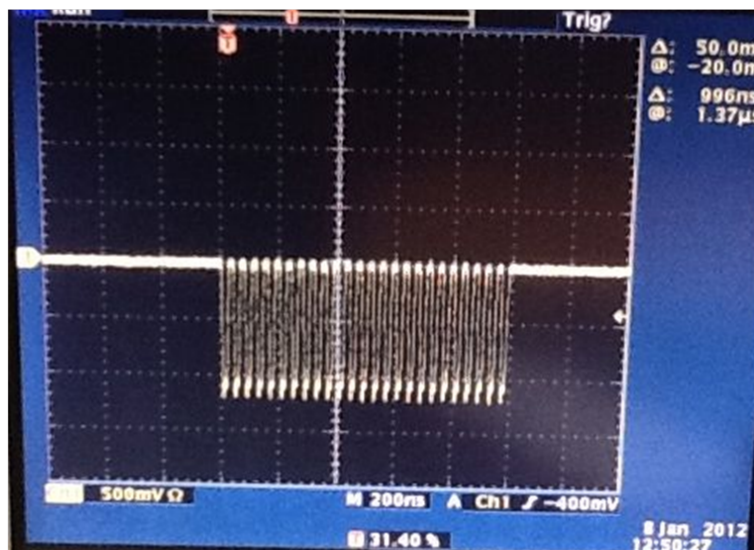


図 30: 擬似実験 NIMOUT オシロスコープ

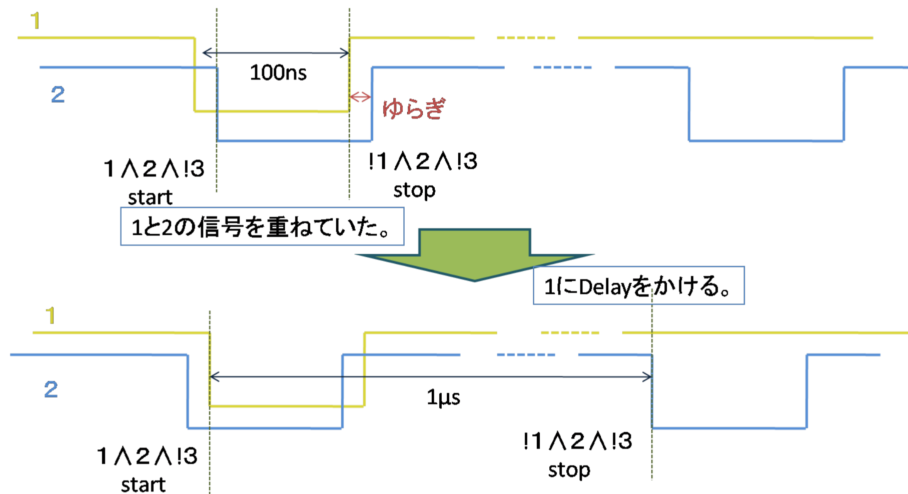


図 31: 擬似実験 NIMIN 調整

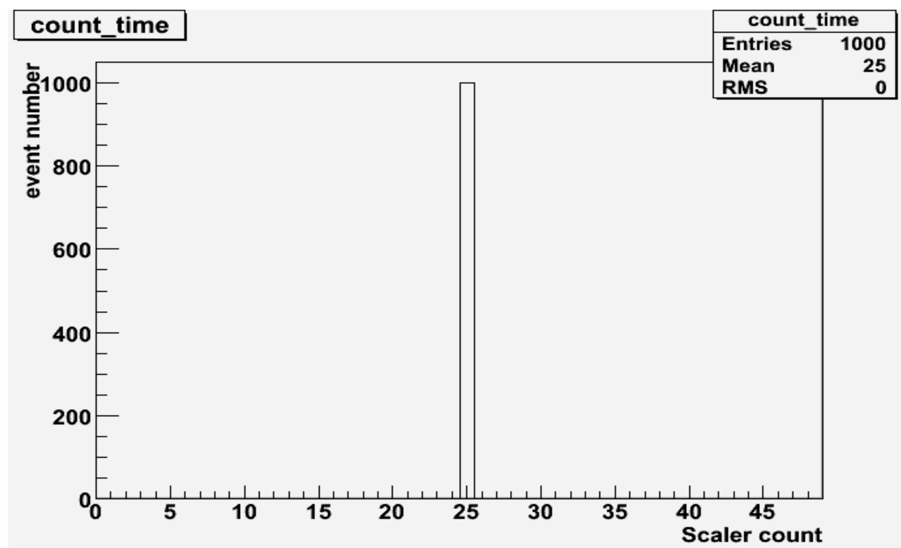


図 32: 擬似実験ヒストグラム

4.3 ミューオン寿命測定

検証実験であったような信号の読み取り間違いを防ぐ為、NIMIN の幅を図 26 下のように調整しました。

入力シグナルの幅を 20ns より小さくしたとき、スケイラーの値で 0 というイベントがランダムに発生しました。これは、スケイラーに NIMOUT2(ゲートシグナル) は入力されているが、NIMOUT1 が入力されていないことなります。しかし、NIMOUT1 のスタート信号は図 24 下を見てもわかるように、NIMOUT2 が発生する条件と同じ、

$$(NIMIN1)\bigcap(NIMIN2)\bigcap(\overline{NIMIN3}) \tag{19}$$

であるため、FPGA が NIMOUT1 を生成する過程で、

$$(NIMIN1)\bigcap(NIMIN2)\bigcap(\overline{NIMIN3}) \tag{20}$$

の信号を読み取れていないことになります。

このような現象が起こる理由は、alsays 文内では、クロックの立ち上がりの時点での信号を扱うためです。つまり、入力信号をクロックの 20ns 以内の幅にしてしまうと、図 33 のように、信号が 10n 秒間 on になっても、クロックの立ち上がりの時点では信号が off になっています。それ故、always 文内の命令が実行されなかったのです。このことが、ネット型である NIMOUT2 は信号が発生するが、レジスト型の NIMOUT1 は発生しない、という事態を招いていたのです。

そこで入力信号の幅はクロックの周期より大きくとることにしました。

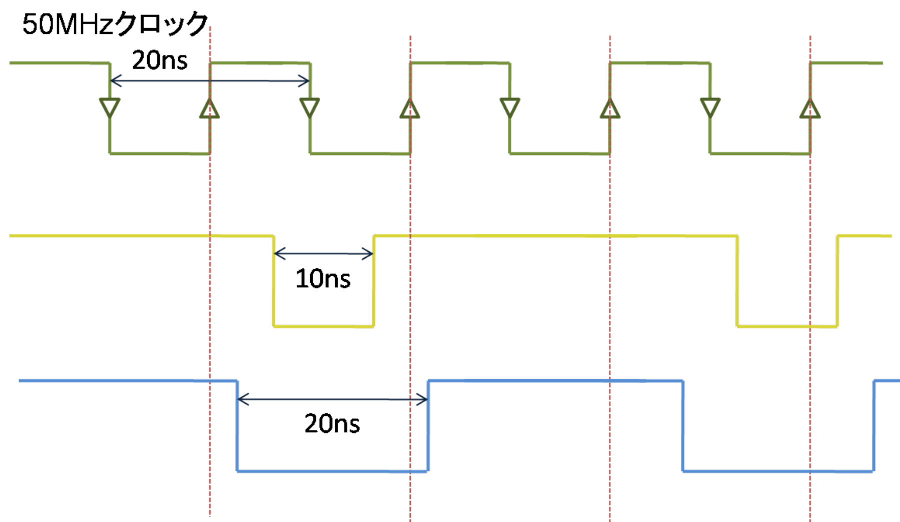


図 33: 実験信号の幅

測定結果をヒストグラムにしたものが図 34 の上のプロットです。対数グラフにプロットしたものが図 34 の下のプロットです。フィット関数は式 9 を用いて $y = p_1 exp(\frac{40ns}{p_0}x)$ であり、$p_0 = \tau$(寿命) です。

この結果より、ミューオンの寿命 (p_0) は $2.05 \pm 0.04\mu$ 秒と見積もられました。文献値の $2.197109 \pm 0.000021\mu$ 秒と比較するとほぼ一致します。少し小さく見積もられるのは、物質中のミューオンの崩壊は正確に測定された値よりは短めの寿命となることが知られています。

このことを考慮すれば、FPGA をトリガー回路としたミューオンの寿命測定は非常に精度よく測定ができていると言えます。

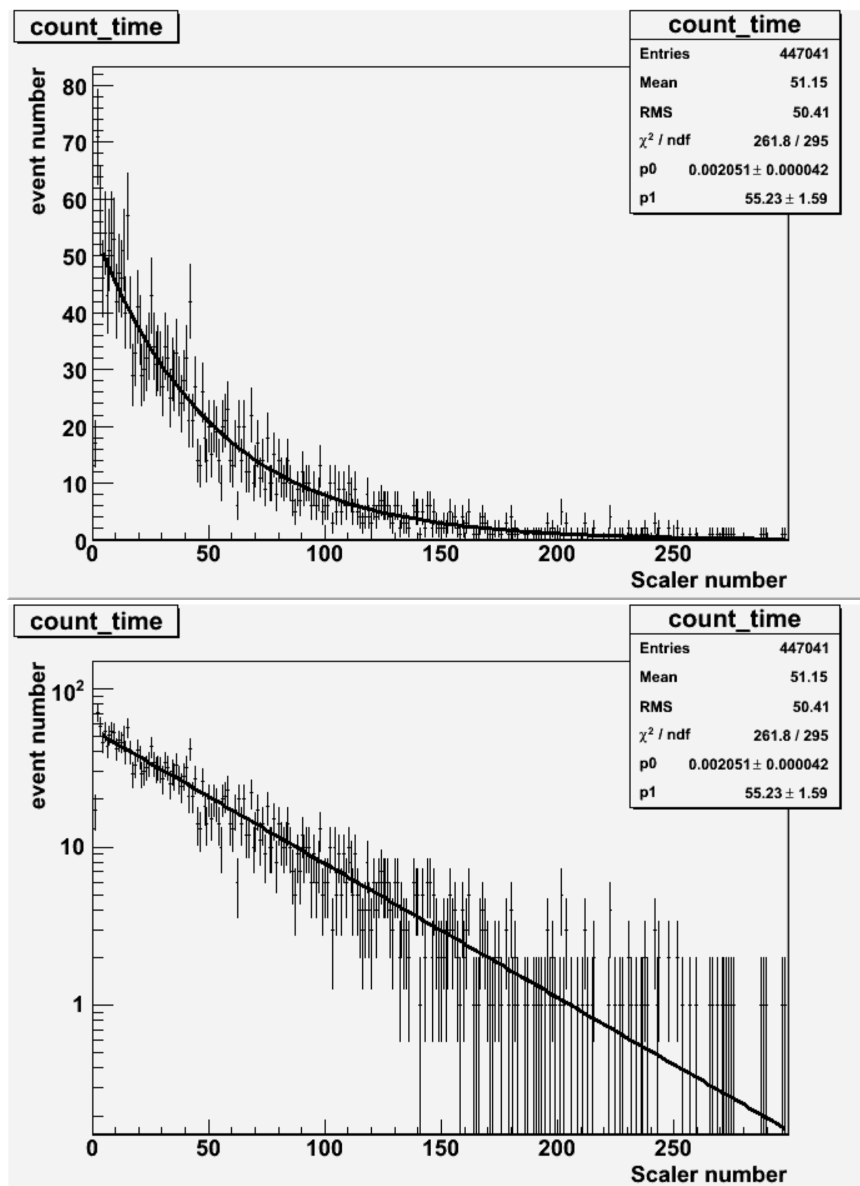


図 34: 寿命測定ヒストグラム

4.4 高エネルギー衝突実験に向けて

1.2 で述べたように、PHOS 検出器のトリガー回路に費やすことのできる時間は 100ns 以下です。この時間内に効率よく粒子を識別するために、現在より処理速度が速くするアルゴリズムを開発する必要があります。

性能評価の結果から、処理速度は内部クロックと回路の階層に大きく依存することが分かります。FPGA に使われる内部クロックをより速いものに変えるだけで、いまあるアルゴリズムの処理速度を上げることができます。

また、クロックの周波数を上げることで、入力信号を幅が狭くすることが可能であり、より精度のよい測定も同時に実現できます。

クロックに依存しない FPGA の性能である、内部クロックを使ったレジスト型信号を制御する精度は 1ns 以下でした。

これらの動作確認のもと、ミューオン寿命測定をすると、誤差 40ns で寿命を測定できていることができました。

現在 FPGA に用いられる内部クロックは 500MHz 以上のものが既にあります。この内部クロックに対応した FPGA を用いることで、1 クロック当たり 2ns 以下で処理されます。単純に計算すると、50 階層まで回路を構築することができます。

このことから、FPGA はナノスケールでの測定が可能であり、高エネルギー衝突実験に有用です。

今後の課題は、限られた階層のなかで、より効率のよいアルゴリズムを見つけ出すことです。

5 まとめ

高エネルギー衝突実験では、極初期宇宙状態を作り出すことによって、宇宙誕生のプロセスを明らかにすることを重要な目的の一つとしています。

この目的を達成するため、広島大学クウォーク物理学研究室では二粒子分解能・エネルギー分解能の優れたフォトン検出器 (PHOS 検出器) を開発しました。衝突により出てくる終状態の光子が高エネルギー場中を通ることにより、エネルギーを与えられた光子 (サーマルフォトン) が生成されます。これを観測することで、極初期宇宙状態を探ることが、PHOS 検出器の役目です。しかし、入力信号が閾値を超えたかどうかを判別するトリガー回路の処理速度が遅いため、本来期待していただけの粒子を検出できていないです。

本研究では、トリガー回路の高処理速度・高精度のアルゴリズムを効率的に探求し、粒子の検出効率を向上させることを目的としています。そこで、書き換え可能な集積回路であり、近年の急速なテクノロジー進化に伴い高集積化や低価格化が著しい FPGA 技術のトリガー回路への適用可能性について検討しました。

LED の点灯・スイッチを押した回数を数える・時間を数えるなどの比較的簡単な回路を実現することで、動作確認を行いました。FPGA の使用法を学ぶと共に、FPGA を通ることによる信号のディレイや、精度のよい測定をするための手法を調べました。それらの過程を通して、FPGA を通ることによる信号のディレイは、論理回路の階層に依存し、(階層) × (クロックの周期) で求まることがわかりました。精度のよい測定のためには、レジスト型の入力信号の幅はクロックの周期より大きくとることを要求されることがわかりました。信号のゆらぎも考慮にいれ、入力信号の幅やディレイを使い、ゆらぎやクロック周期を考慮して調節することで、信号の読み取り間違いをなくしました。回路内の信号のゆらぎは 1ns 以下の精度で制御されていることがわかるました。

また、ナノスケールでの高速処理を必要とするミューオン寿命測定を FPGA 技術を用いたトリガー回路によって成功しました。

FPGA はナノスケールで信号の処理ができ、精度も高く、高エネルギー衝突実験に有用な技術です。近年 FPGA 自体の処理速度、精度も向上していることから、最新の FPGA を導入することでトリガー回路の性能向上も期待できます。

謝辞

最後になりましたが、本卒業研究を行うにあたり、お世話になった方々へ、感謝の意を表したいと思います。

FPGA を学ぶ上で指導教員として指導をして頂いた杉立先生には、研究を行うための器材や資料などを一通り揃えて頂き FPGA 技術を習得する準備をして下さいました。研究を進める過程では、道筋を順序立てて、丁寧に示してくれました。

主査として論文の添削をして頂いた三好先生には、難解な言い回しや文の構成を的確に指摘して頂いたおかげで、読みやすい論文に仕上がりました。CERN Summer School の申込みの際には、急なお願いで時間がない中、推薦状を書いて頂き、本当に感謝しています。

スタッフの方々では、志垣先生、本間先生、には研究室会議で、実験の信憑性、再現性を指摘して頂き、物理的理解につながりました。槌本さんは、類まれなプログラミングやハードウェアの知識を使い、HDL 記述でわからない所を一緒に考え、道を拓いて下さいました。研究以外でも多くのことを学び、大変参考になりました。

研究室の先輩方では、中宮さん、来島さんには研究が順調に進んでいるかいつも気にかけて下さいました。BBC で培ったトリガー回路の知識を元に、貴重な時間を割いて熱心な指導して頂きました。中宮さんは、疑問点があると細部にわたって説明して頂き、つきっきりで教えて下さりました。来島さんには、クォーク物理についての教養や root の使い方、TeX の使い方を懇切丁寧に教えて下さいました。坂口さんには論文のアドバイスを頂きました。尾林さんには TeX の使い方、PHOS の動作のさせ方を教えて頂きました。翠さんには PHOS のトリガー回路の仕組みから、今 PHOS トリガー回路が抱える課題まで、研究の土台となる所から目指すべき目標まで教えて頂きました。論文に関しても物理的に正しい用語の使い方や、間違いを指摘して下さいました。星野さんは、後輩のことをよく気にかけて下さり、おかげで早く研究室に馴染むことができました。アブスト提出の際、多くのアドバイスを頂きました。八野さんには、プログラミング・PHOS・CERN Summer School・卒論と多方面でお世話になりました。渡辺さん、二橋さんにもアドバイスと励ましをいただきました。

研究室の同期の大久保君、久米君、辻さん、長谷部君には多くの迷惑をかけてしまいました。同期の支えがあってこそ頑張れてこれました。いつも笑顔で優しい大久保君。しっかり仕事をこなす姿に後押しされました。卒論制作では研究室で最も頑張っていた久米君。頑張っている久米君が居たから、諦めずに努力することが出来ました。女性研究者を目指し、向上心が高い辻さん。何事にも挑戦し、達成する行動力を見習いました。情熱が一番強く、常に仕事をしている長谷部君。努力を続け、自ら道を切り開く姿をみて、自分を見直すことが出来ました。

クォーク物理学研究室に FPGA 技術を扱える者が居らず、予備知識のない中での研究でした。そこで、FPGA 技術の先駆者である高エネルギー宇宙研究室の大野先生、後藤さんにもお世話になりました。FPGA について右も左もわからず、本を読んでもなかなか前に進まない時分、FPGA 技術の習得の仕方、FPGA ボードの仕組みなど、何から手をつければよいか教えて頂きました。

多くの協力があり、無事書き上げる事が出来ました。心から感謝します。

ありがとうございました。

参考文献

[1] http://atlas.kek.jp/sub/photos/Accelerator/tunnelPV.jpg

[2] http://t0.gstatic.com/images?q=tbn:ANd9GcTNpsZl3teVUCJ1otlSKMHw0Z96V2eVBtR_v0G-gvlKuUzBu4KJ

[3] https://www.hepl.hiroshima-u.ac.jp/thesis/bachelor/09sakaguchi_thesis.pdf

[4] 井倉 将実,FPGA ボードで学ぶ Verilog HDL

[5] 小林 優, 入門 Verilog HDL 記述

[6] 長谷川 祐恭,VHDL によるハードウェア設計入門

[7] 西野他編,FPGA 超入門

[8] Particle data book

[9] Lijiao Lui,L0/L1 trigger generation by the ALICE PHOS detector