卒業研究報告書

空気シャワー観測用望遠鏡の
トリガー回路製作

甲南大学理工学部物理学科　宇宙粒子研究室
学籍番号　10861049
氏名　松本光平

指導教授　梶野文義
山本常夏

2012 年 2 月 3 日

目次

§1 本実験の目的

1-1 目的

本研究では空気シャワー観測用望遠鏡の望遠鏡のトリガー回路の一部を開発する。この回路では①クロックのパルス幅による蛍光紫外線の光量計測や、②ノイズの除去など分担して制御するシステムを組むことを目標とする。

本研究では、VHDL（ハードウェア記述言語）言語によるプログラミングで FPGA ボードを制御し、観測対象である宇宙線をファンクションジェネレータを用い擬似信号を発生させ実験を行う。①と②の動作を行うようなプログラムを製作し、さらに PC でそれらの結果を出力及び分析できるようにする。

§2 極限エネルギー宇宙線とその観測方法

2-1 宇宙線の観測について

1960 年代中頃、ビッグバン宇宙の証拠とされる宇宙背景放射が発見された。この宇宙線は 10^{20}eV 付近にエネルギーの上限があると考えられていたが、発見から約 5 年後、10^{20}eV の上限を超えるエネルギーをもった宇宙線（以下、極限エネルギー宇宙線）の観測が報告された。このことから、宇宙線の起源と上限の有無を探ることで、宇宙や粒子の誕生などといった疑問を解消する足掛かりとなるのではないかと注目されている。

観測対象である陽子などの極限エネルギー宇宙線は地球の大気に突入すると、大気中の原子核と衝突し、電子やγ線などからなる空気シャワー（1-4）を発生させる。このとき、粒子によって励起された窒素分子から放射される蛍光紫外線を測定することにより、粒子の到来方向とエネルギーを求める。それにより、粒子の発生源や加速機構の手がかりが得られると期待される。

このような、蛍光紫外線を観測する望遠鏡の例の 1 つとして JEM-EUSO 望遠鏡があり、図 3 のような構造になっている。

これを用いて、極限エネルギー宇宙線によって発生した空気シャワーから生じる蛍光紫外線を空間分解能約 0.75km×0.75km 及び、時間分解能 2.5μs の三次元で撮影記録する。

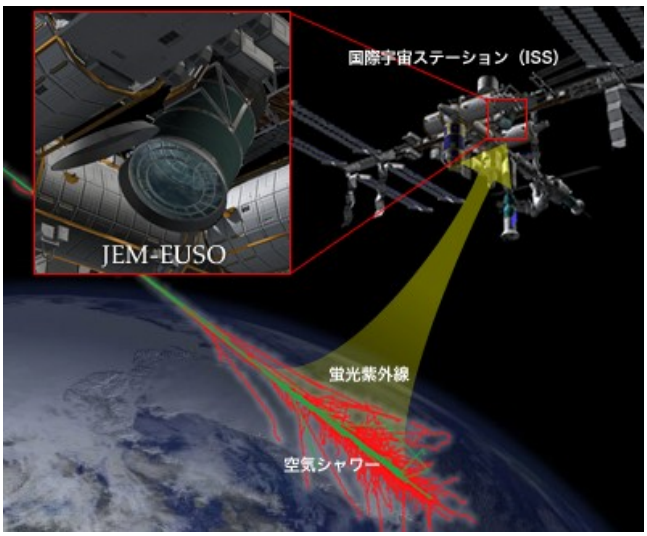

図 1　JEM-EUSO 計画のイメージ

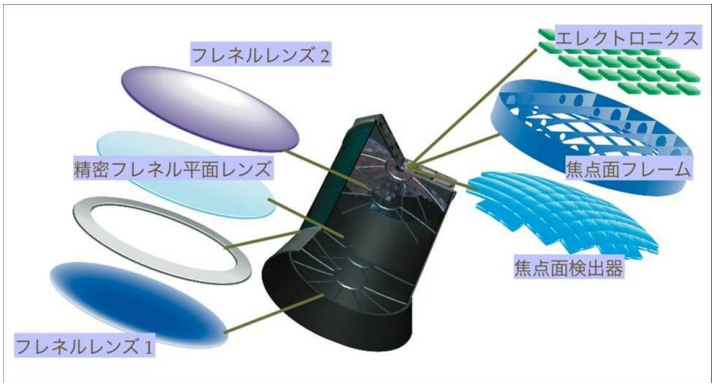

図 2　JEM-EUSO 望遠鏡の内部構造

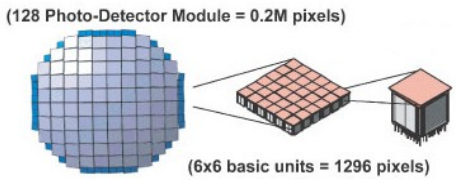

図 3　焦点面検出器の構造（右は光電子増倍管1本を示す）

2-2 極限エネルギー宇宙線

宇宙線とは、超新星爆発や恒星表面の爆発などで発生した、宇宙空間を飛び交う高エネルギーの放射線である。その中でも、10^{19}<E<10^{21}eV のエネルギーを持った粒子を極限エネルギー宇宙線と呼んでいる。また、宇宙線は宇宙空間から飛来する一次宇宙線と、一次宇宙線が大気中に突入して生じる二次宇宙線（1-4 参照）に分類することができる。

図 5 は宇宙線のエネルギースペクトルをグラフ化したものであり、横軸は一次宇宙線のエネルギー、縦軸は観測された頻度を示している。グラフからも分かるように、極限エネルギー宇宙線に分類される粒子のデータは少ない。

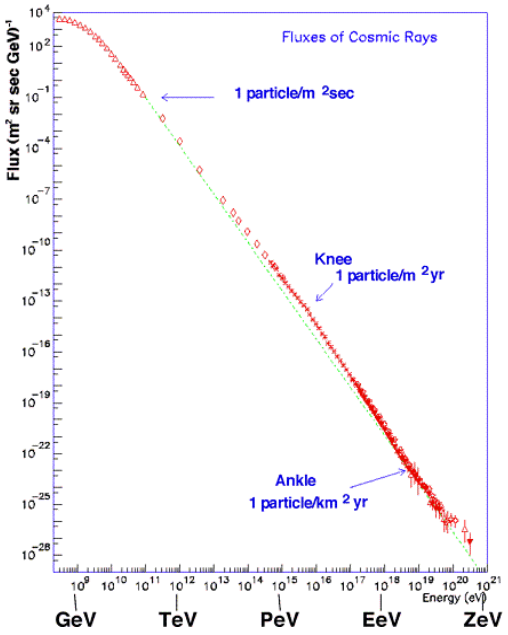

図 4　一次宇宙線の到来頻度 Flux-エネルギーのグラフ

この極限エネルギー宇宙線は、発生源から宇宙空間の磁場による影響をほとんど受けず直線的に飛んでくることが知られており、地球がある銀河を上から見て、粒子の運動を図示したものが図 6 である。その精度は平均して 0.2～0.3°程であり、磁場の強度を 1nG と考えた場合でも曲げられるのは 2～4°という精度で観測することができる。よって、観測による粒子の到来方向をより正確に特定することができると期待できる。

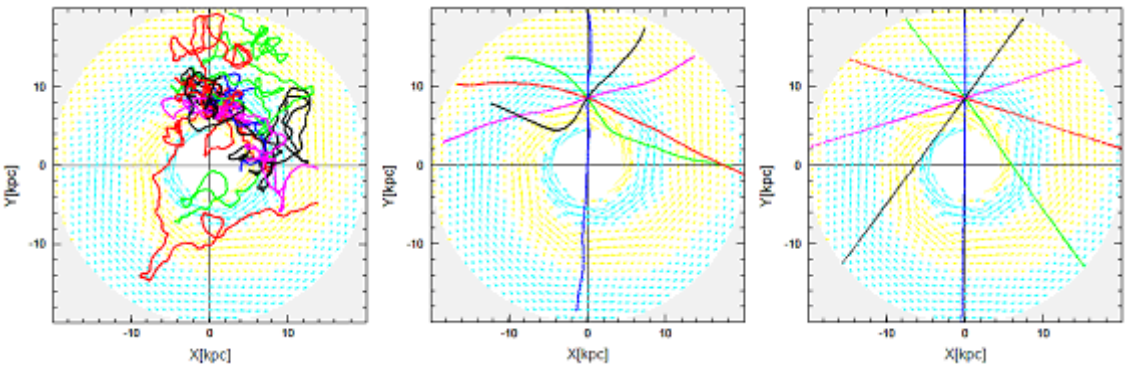

図 5　宇宙空間での粒子の動き（左から、10^{18}・10^{19}・10^{20}eV の一次宇宙線）

極限エネルギー宇宙線の起源モデルには大きく分けて2種類あり、それぞれボトムアップシナリオとトップダウンシナリオと呼んでいる。

ボトムアップシナリオとは、エネルギーの低い粒子が天体の中で徐々にエネルギーを獲得していくという過程です。これは地上の人口加速器とほぼ同じ原理ですが、極限エネルギー宇宙線のようなエネルギーを持つ粒子を発生させるには、地上のものより8桁近く上回っていなければなりません。天体における加速の理論的限界は天体の大きさ×磁場の強さで決まります。

図7を見ると、超強磁場中性子星、活動銀河中心核ジェット、ガンマ線バースト、電波銀河、銀河団などがぎりぎり10^{20}eVの極限エネルギー粒子を加速できることが分かる。つまり、極限エネルギー宇宙線がボトムアップシナリオによって発生したものであれば、10^{20}eV付近にエネルギーの上限があると予測できる。このため、このエネルギー領域を超えても加速限界が見られない場合、図7右上に未知の天体が存在する。あるいは、トップダウンシナリオが正しいことになる。

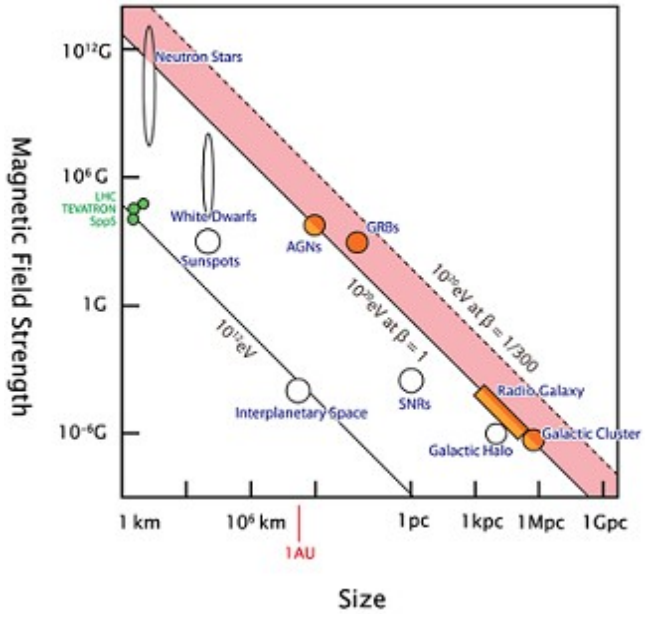

図6　Hillas Diagram

トップダウンシナリオとは、超重粒子の崩壊または対消滅で極限エネルギー粒子が発生するという考え方である。これらが作る最終生成物はニュートリノとγ線が大部分で、γ線の核子は2〜3倍ほど作られる。

2-3 空気シャワーと蛍光紫外線とは

宇宙線には大きく分けて宇宙空間から飛来する一次宇宙線と、一次宇宙線が大気中に突入して生じる二次宇宙線がある。

極限エネルギー宇宙線(陽子やα粒子)が地球の大気に突入したとき、大気中の原子核と衝突を起こす。このとき高エネルギーの二次粒子が発生し、それがまた別の原子核に衝突する。このような反応が連鎖的に起こり、二次粒子が大量に発生する。この現象を空気シャワー(図 8)と呼んでいる。

粒子の反応が進むにつれ、その数は増加するが、1 粒子あたりのエネルギーは低くなる。生成された粒子のうち、寿命が短いものは崩壊し、残った電子、γ線、π中間子などは地表に到来する。その数は、10^5～10^9個ほど観測されている。

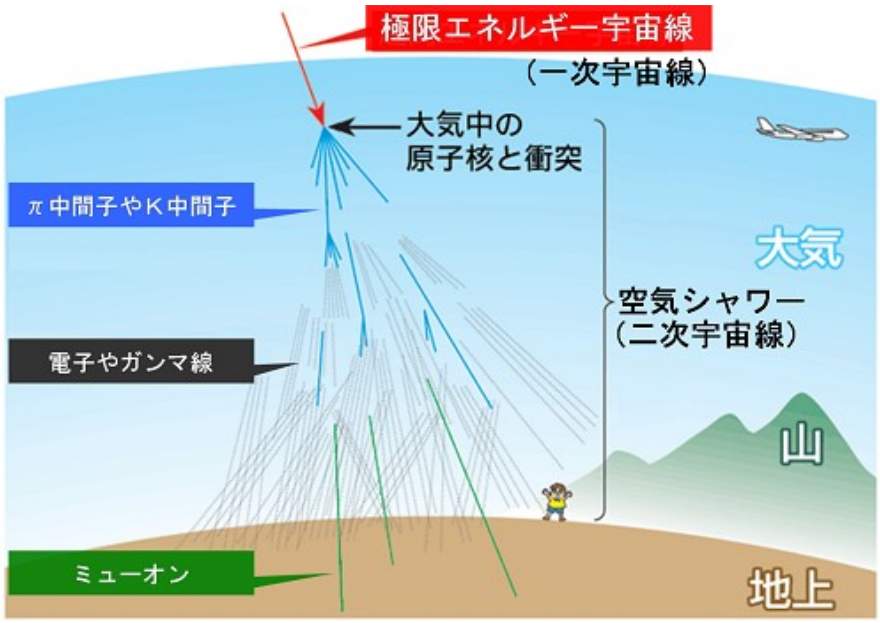

図 7　空気シャワー発生のイメージ

蛍光紫外線とは、極限エネルギー宇宙線が大気中に突入し、空気シャワーを形成する過程の反応で発光する現象である。

空気中の約 80%に含まれる窒素分子のうち、系の固有状態の中で最低のエネルギー状態(基底状態)にある粒子に、空気シャワーで発生した粒子が衝突することで励起されると、光を発する。

§3　実験方法

3-1 使用した実験装置

本実験では、以下の実験装置を使用した。

・FPGA ボード(『SPARTAN-3E』Xilinx 社製)

・I/O ボード(『PCI-2702C』interface 社製)

・オシロスコープ

・ファンクションジェネレーター

これらの装置について、以下で説明を加えることにする。

3-2 FPGA ボードと VHDL 言語

FPGA ボードとは、図 9 で示すように、FPGA（図 9 赤枠内）を主体とした LED やコネクタなどの周辺機器を含む装置を指す。FPGA は、任意のプログラムをパソコンからインストールすることで作動する LSI である。また、プログラムは何度でも上書きすることができる。ソフトウェア ISE8.を使用し、VHDL 言語でプログラミングを行う。VHDL とは『Hardware Discription Language』（ハードウェア記述言語）の略であり、接頭語の V は、1980 年代にアメリカの国防総省の超高速 IC（VHSIC：Very High Speed Integrated Circuit）開発プロジェクトによってまとめられた際のプロジェクト名に由来している。

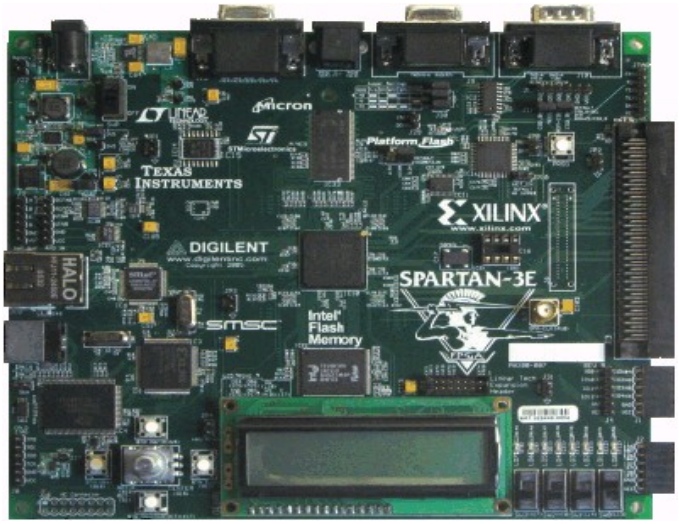

図 8　Xilinx 社製、FPGA ボード

3-3 ファンクションジェネレーター

ファンクションジェネレーターは、任意の波形を外部に出力することのできる機器である。

本研究では、光電子増倍管が出力する電気信号（パルス）を擬似的に出力させるため、パルスジェネレータを直接 FPGA ボードに接続して使用した。

設定は、最大電圧 3.3V、最小電圧 0V で、波形は Square である。

今回実験では、Trigger を 1 回押すごとに 1 波形を FPGA ボードに出力したいため、Burst の Cycles 設定を 1Cyc（N Cyde、#Cydes）Trigger Setup（Manual、立ち上がり）にした。

また、パルスジェネレータの出力抵抗は 50Ω である。

図 9　ファンクションジェネレーター

3-4 オシロスコープ

オシロスコープは、入力された信号を波形として表示することのできる計測器である。本研究ではパルスジェネレータからの信号を 1ch に、FPGA からのクロック信号を 2ch に表示し、視覚的に評価できるようにした。本実験での設定は、1ch(1.00VΩ)、2ch(2.00VΩ)、M(100ns)、立ち上がり 1.56V である。

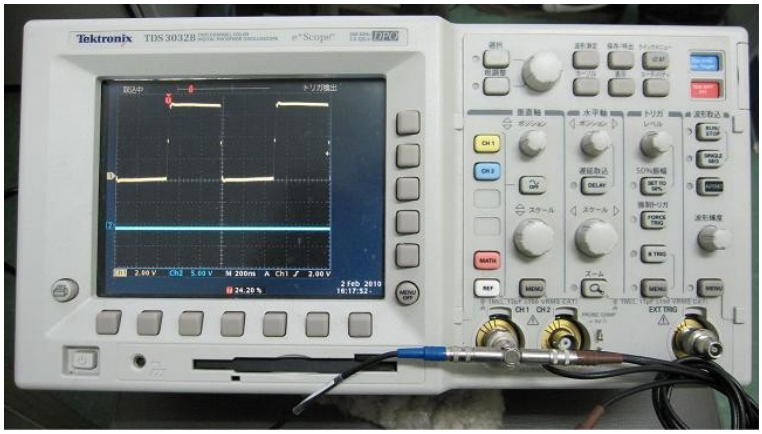

図 10　オシロスコープ

3-5 I/O ボード

I/O ボードとは PC に設置し、外部に接続するモジュールと情報の入出力を行うためのインターフェースである。今回使用するものは Interface 社製 PCI-2702C という規格の I/O ボードである。

本研究では、FPGA から出力される 8bit のカウントデータを PC へ入力するために使用した。I/O ボードは GPC-2000 というドライバソフトウェアをダウンロードすることで、DIO 診断プログラム及び入・出力ユーティリティの 3 種類を取得し制御を行う。

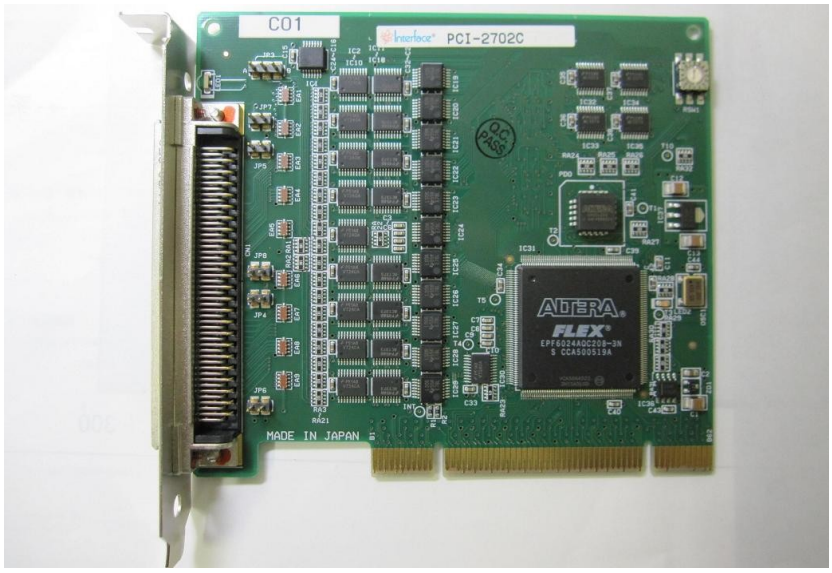

図 11　I/O ボード

3-6 ソフトウェア ISE8.2i について

ISE8.2i とは、FPGA を制御するために必要なプログラムを VHDL 言語で作成する Xilinx 社製のソフトウェアである。

インストールは Xilinx 社のホームページからアカウントを取得し、指定の情報を入力することでダウンロードできる。アカウント取得後の手順は、My Account からログイン、サポート、ダウンロードと順に選択し、必要なソフトウェアのダウンロードタイプ、ISE の Version、OS を指定することでダウンロードページを表示できる。

今回は ISE8.2i をインストールディスク(No.OBW282944895)からセットアップする際に必要となるレジストレイション ID を、XILINX 社からメールで取得する必要があるが、すでに ID が取得されているインストールディスク(No.NIV616343247)が PC にインストール済みであったため、そのままこれを用いた。

3-7 ISE10.1 クイックスタートチュートリアル

本研究を進める前に、使用する ISE8.2i と VHDL 言語の基礎を覚える必要がある。このために、ISE10.1 クイックスタートチュートリアルを参照しながらプログラムを作成した。

このプログラムはクロックの波形を 1 つ認識するたびに、LED の点滅でカウント表示させるものであり、開発中の設定は以下の通りである。

1. プロジェクトの新規作成、New Project Wizard – Create New Project。
    - Project Name： tutorial
    - 保存場所： H:Xilinx￥tutorial
    - 形式： HDL
2. FPGA ボードの指定、New Project Wizard – Device Properties の変更箇所
    - Family： Spartan3E
    - Device： XC3S500E
    - Package： FG320
    - Speed： -4
3. ソースタイプ選択、New Source Wizard – Select Source Type
    - File name： counter
    - モデル選択 ： VHDL Module
4. モジュールの定義、New Source Wizard – Define Module
    - CLOCK： in
    - DIRECTION： in
    - COUNT_OUT： out： Bus にチェック、MSB3 - LSB0

FPGA ボードに付属しているクロックは 50MHz の信号を一定間隔で出力することができる。また、COUNT_OUT の Bus とは出力したい LED の個数のことで、3-0 の場合 4 個の LED を使用できる。

プログラミング完了後、Synthesize – XST でコンパイルを行う。問題がなければ、Synthesize – XST と View Synthesis Report 、Check Syntax の文字横に  の表示が出る。

次に、プログラムを FPGA にインストールする前に PC 上でシミュレーションを行うために、テストベンチを作成する。New Source Wizard – Select Source Type を表示し、設定を以下のようにする。

- File name： counter_tbw
- モデル選択 ： Test Bench WaveForm

最初にデザインシミュレーション（図 12）を行う。これはタイミングを初期化し、シミュレーションの時間やクロック信号の間隔などを指定するウィザードである。

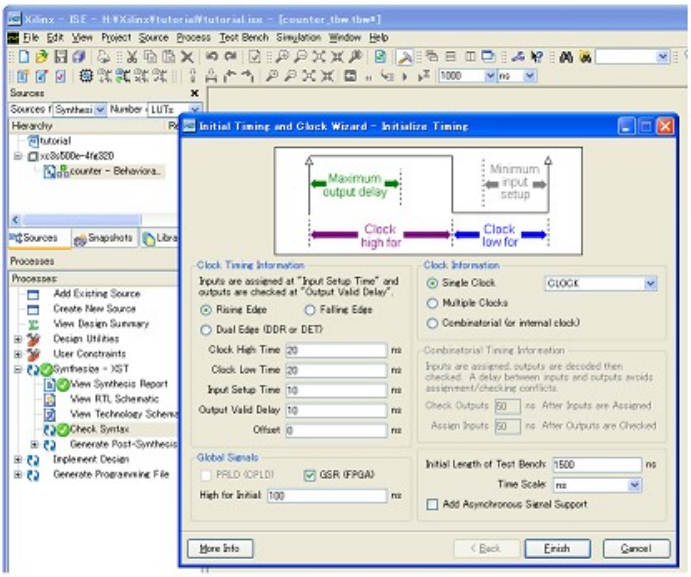

図 12　デザインシミュレーション設定画面

Finish をクリックし、テストベンチ波形を作成するウィザードで任意の波形を設定し、出力する。出力画面で波形が図 13 で設定した位置で変化していれば成功である。

プログラムを FPGA にダウンロードするためには、プログラムで使用すると宣言した LED やクロックに、FPGA ボードを見ながらピン番号を割り振る必要がある。ピン番号は FPGA ボード上の、使用したい回路の横に括弧で記してある（クロックなら C9 のように）。

User Constraints の Assign Package Pins から図 13 を立ち上げ、Design Object List の Loc にピン番号の入力を行う。入力が正しければ、図 13 の Package Pins for nc3s500e-4-fg320 に表示されるピン（丸及び六角形の駒）が青くなる。また、すでに赤や黄などで埋まっているピンには指定することができない。

ピンロケーションの選択が終わったら、TOP の View Design Summary で正しく配置されているか確認する。ISE10.1 では All Constraints Met というリンクを押せという支持があるが、ISE8.2i では選択できなかった。

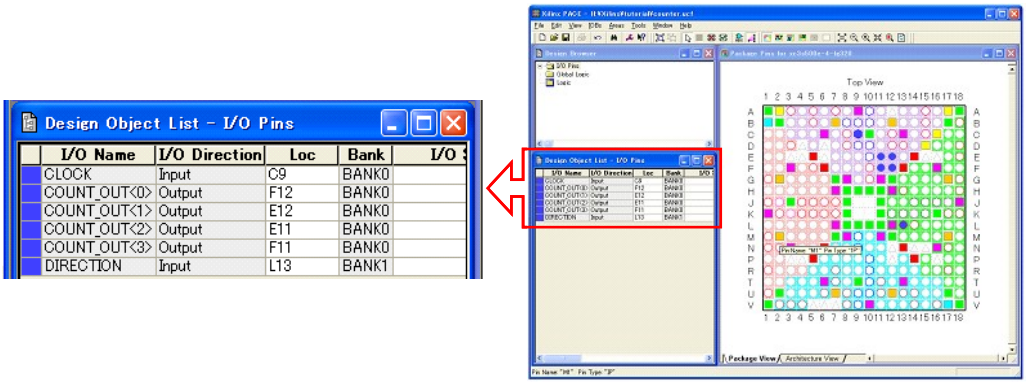

図 13　ピンロケーションの割り当て画面

View Design Summary でピンが正しく配置されたことを確認した後、FPGA ボード（図 8）の USB コネクタを PC に接続し、電源を入れる。TOP の Generate Programming file の Configure Device（iMPACT）をクリックするとダウンロードウィザードが開始される。

表示された画面（図 14）の xc3s500e file に counter.bit を適応し、次の xcf04s と xc2c64a は Bypass を選択する。

xc3s500e を右クリックし、Program→OK を選択。画面に「Program Succeeded」が表示され、FPGA ボード右上にオレンジ色の LED が点灯すればダウンロードが完了したことを示す。

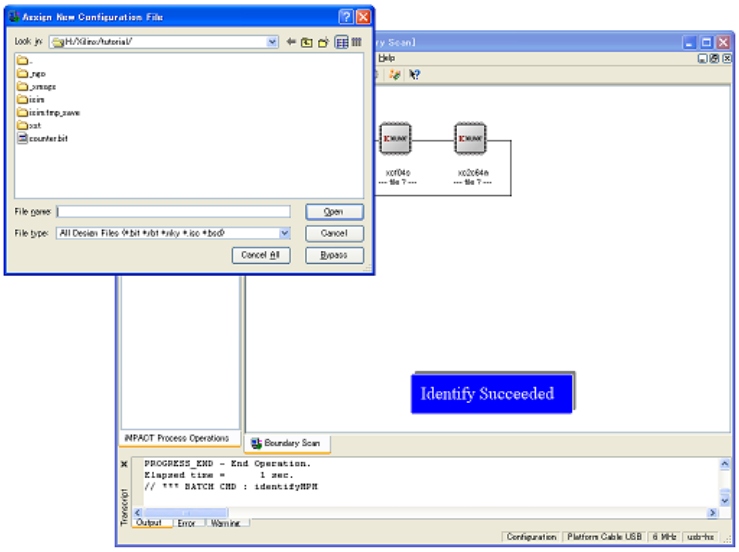

図 14　プログラムダウンロード画面

ダウンロードが終了するとすぐに動作が開始されるが、Tutorial counter プログラムでは 50MHz という非常に大きなクロックを使用したため、LED0 から 3 の点滅が早すぎて目視できず、連続的に点灯しているように見えた。

目視するためには、パルスジェネレータなど外部からクロックの代わりとなる信号を入力する。もしくは、FPGA ボード内のデジタルクロックマネージャ（DCM）を使用し、クロックを小さく調整する必要がある。

2-4 では上記 2 種類の方法を試した。

3-8 クロック信号の周波数調整

2-3 で課題となった 2 種類のクロック信号を調整する方法を検証した。それに伴い、LED 出力を 0～7 に、スイッチが 0 のときクロックのカウントを停止及びリセットし、スイッチが 1 のときカウントを+1 ずつ開始するプログラムに変更した。

まず、クロック部分のピンロケーションをピン番号 D7 に変更し、パルスジェネレータを用いて外部からクロック信号を入力した。パルスジェネレータからの出力を 1.0Hz に設定すると、予想通り LED0 から 1s 毎に+1 ずつカウントを行った。

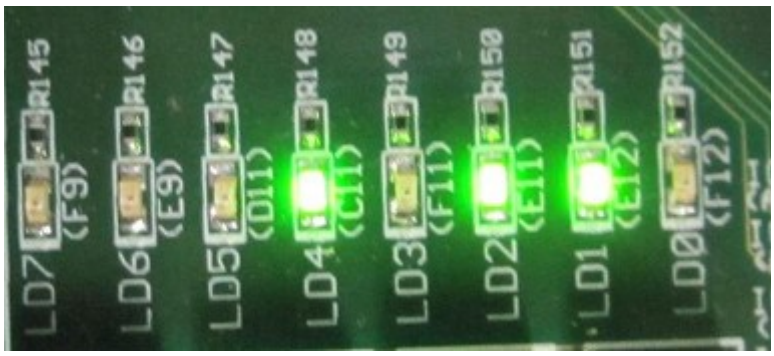

図 15　LED にカウント結果を 2 進数で出力した様子

次に、デジタルクロックマネージャ（DCM）を用いたクロックの調整方法を説明する。DCM とは FPGA ボード内に初めから設置されている回路で、付属の 50MHz クロックの周波数を任意で変更できる。

まず、図 16 の GUI 画面を表示する。手順は以下の通りである。

1. Project の New Source から、New Source Wizard – Select Source Type ダイアログボックスを開き、IP(CoreGen & Architecture Wizard)を選択し、File name に dcm1 と入力する。
2. New Source Wizard – Select IP ダイアログボックスで、FPGA Features and Design、Clocking、Virtex-Ⅱ Pro,Virtex-Ⅱ,Spartan-3、single DCM と選択する。
3. ダイアログボックスを閉じ、図 17 右の画面で RST、CLK0、LOCKED、CLKFX が ON になっていることを確認する。
4. 任意の値をボックスに入力する。例えば Input Clock Frequency に 50 と入力し、MHz を ON にすると 50MHz のクロックを出力する。
5. 設定完了後、Advanced をクリックし、Wait for DCM lock before DONE signal goes high を ON にする。
6. NEXT をクリックしていき、最後に Finish を押すと図 16 のようにプログラム設計画面左上に dcm1.xaw が追加される。

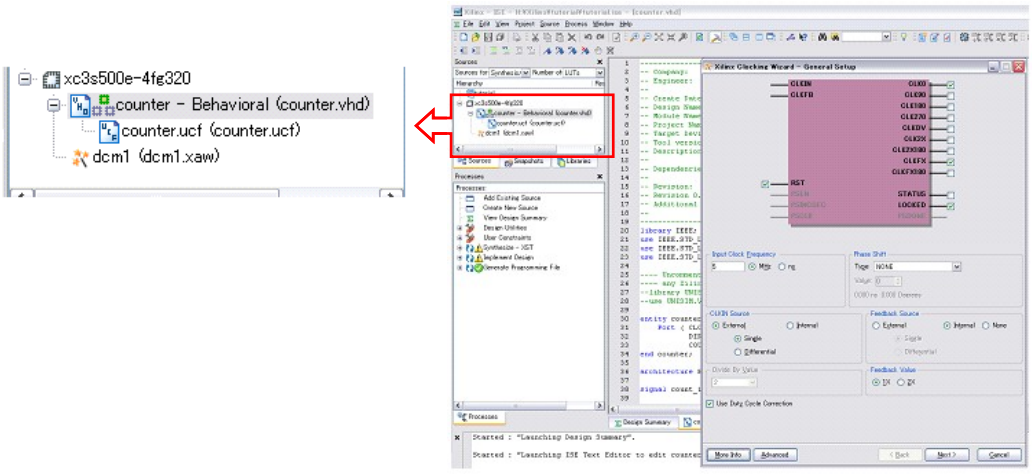

図 16　デジタルクロックマネージャ GUI 設定画面

操作手順は、スイッチ 1 と 2 を ON にし、パルスジェネレータの出力信号を設定して Trigger を押す。スイッチ 1 を OFF にして LED に出力された信号を記録する。そして、スイッチ 2 を OFF にて初期化。この繰り返しでカウントデータを取得する。

また、2 つのプロジェクトを統合するには、最初に Counter プログラムを作成し、次に TOP プログラムを作成する手順で New Project を選択したとき、以下の操作を行う。

1. New Project Wizard - Create New Source 画面で Next をクリック。
2. New Project Wizard - Add Existing Sources 画面が表示される。
3. Add Source をクリックし、counter を選択。
4. counter 内の counter.vhd を開く。
5. Next を押し、Finish を押す。

以上でプログラムを統合することができる。

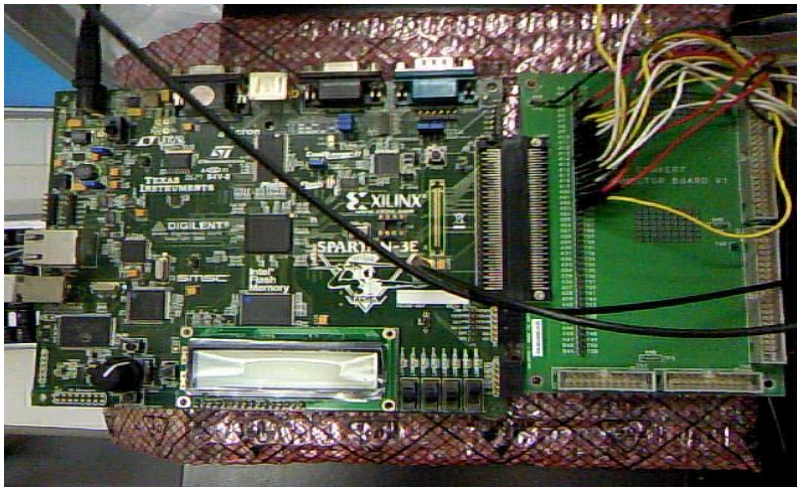

図 17　FPGA 接続図

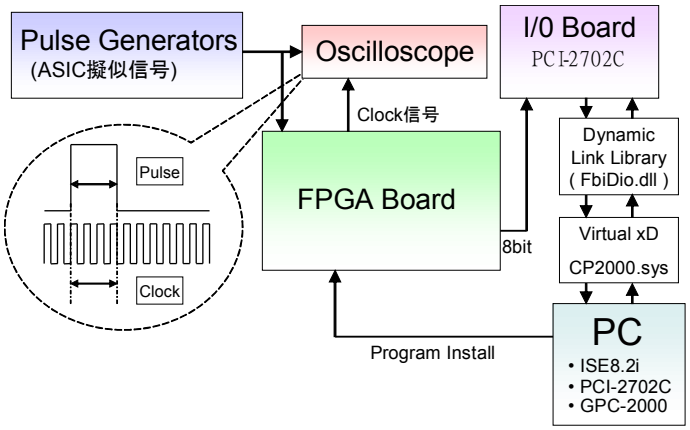

図 18　FPGA データ入力処理アプリケーション実行時の装置の接続図

§4 行なった実験とその結果

以下にこれまで行なってきた実験の説明と結果を載せていく。
尚、各々の実験に用いたプログラムのソース・コードは巻末に記載する。

4-1 クロック周波数調整プログラムの出力結果

50MHz のクロック信号のカウントを即時出力して点滅しているはずの LED が常時光って見えてしまう問題に対して、クロックの代わりにファンクションジェネレーターで 1.0Hz の信号を入力した。結果図 15 で示したように、1 秒毎に 1 ずつカウントが行われ、LED が右から順に点滅加算される様子を目視で確認することができた。

4-2 カウンタープログラムを LED に出力した結果

図 18 のように装置を接続し、実験を開始した。ファンクションジェネレーターで出力した宇宙線の擬似信号の入力信号幅を FPGA に入力し、パルス幅のカウントプログラム(トリガープログラム)で処理及び出力する。(図 19 参照)

本実験では入力信号のパルス幅を 25ns から 500ns まで変化させ、そのパルス幅を 50MHz のクロックで個数をカウントした。図 20 は 240ns の信号を出力した時のオシロスコープ(左)LED(右)の出力結果である。オシロスコープは黄色の線が ASIC 擬似信号で、青色がクロック信号であり、LED の出力は 00001100 であるため、クロックの個数は 12 個だと分かる。また図なんとかは入力信号のパルス幅とカウント数の関係を示したグラフである。

また、カウンタープログラムでは 2 進数 8bit で出力するため、最大 255 個までしかカウントできない。

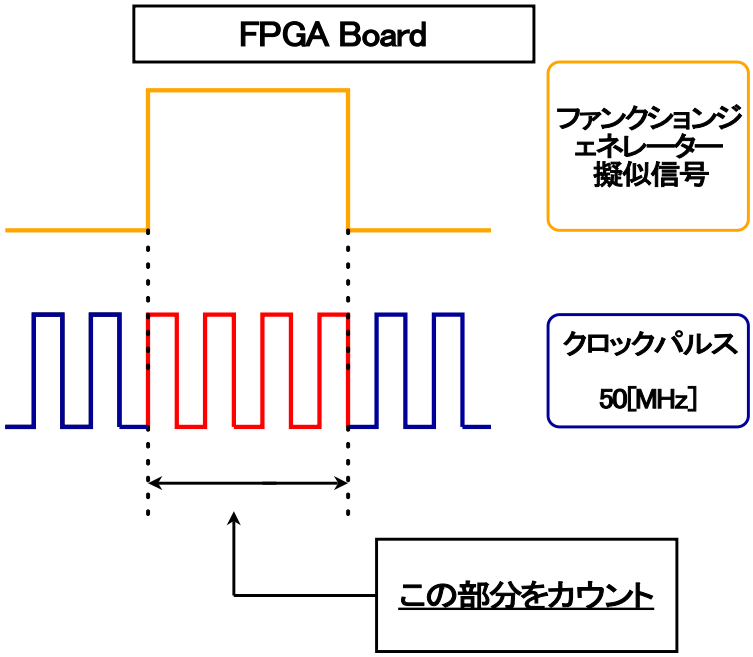

図 19　パルス幅のカウント方法

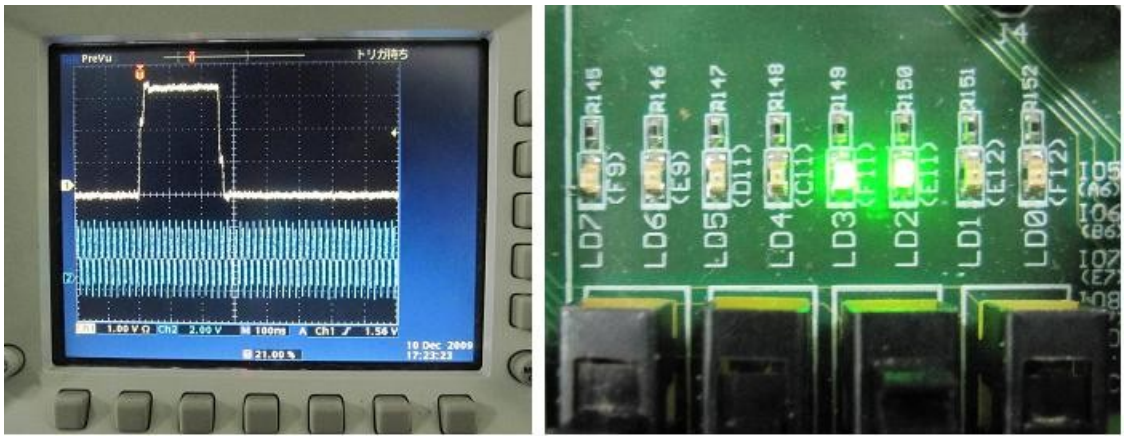

図 20　FPGA に 240ns の信号を入力したときの出力

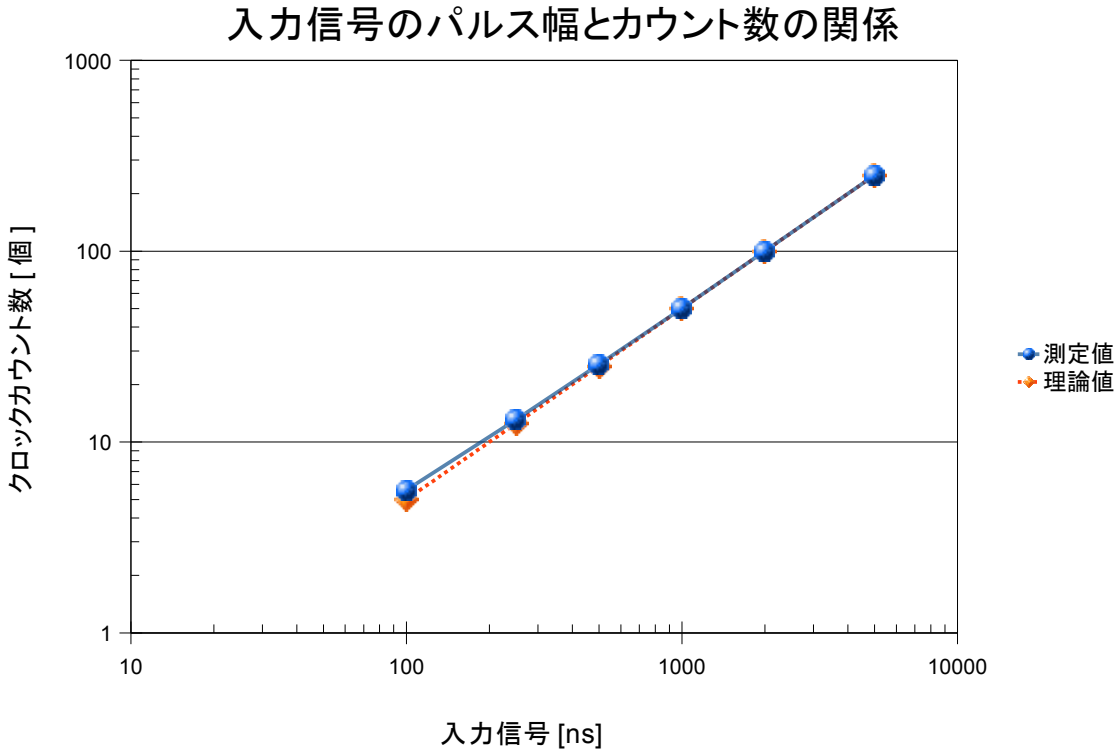

図 21　パルス幅とカウント数の関係グラフ

4-3 カウンタープログラムを PC に出力した結果

FPGA から PC の I/O ボードにデータ「11111110」を出力し、DI ユーティリティで表示した。その結果を図 22 に示す。

カウントの表示であるが、数字と黒丸のセットが 1 つの出力を表しており、信号が入り、High レベルになると、その番号の黒丸が赤色に点灯する。特に断りがない限り番号 1～8 をカウント数のデータ出力として用い、番号が小さい方から下位 bit を割り当てる。

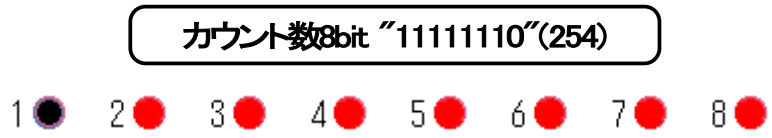

図 22　PC 出力画面

4-4 カウント数に応じてフラグ信号を出力する実験

ラッチのスイッチを切り替えることによってデータを 1 つずつ出力し、出力されたカウント数により固有のフラグ信号（2bit）を出力する実験を行なった。その結果を図 23、24、25 に示す。作成したプログラムはカウント数が 253 以下の時フラグ信号”01”、カウント数が 254 の時フラグ信号”10”、カウント数が 255 の時フラグ信号”11”、を出力するというものである。

尚、この実験では、番号 17、18 をフラグ信号の出力として用いており、これも番号が小さい方から下位 bit を割り当てている。

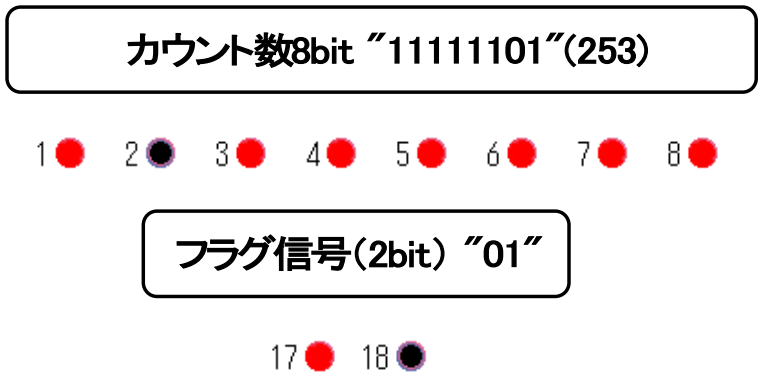

図 23　フラグ信号発行”01”(253)

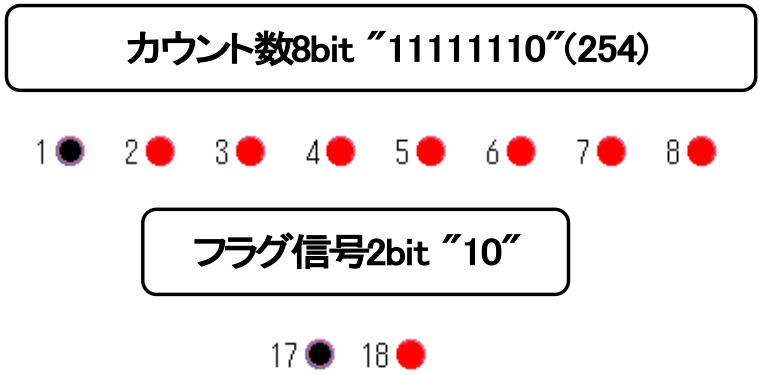

図 24　フラグ信号発行”10”(254)

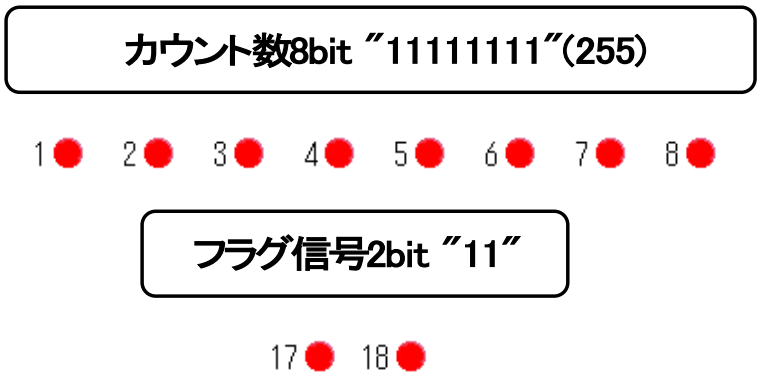

図 25　フラグ信号発行”11”(255)

4-5 トリガー範囲の拡大

3-6 の実験を経て、更にカウント数の範囲によってフラグ信号を出力する実験を行なった。作成したプログラムはカウント数が 0～99 の時フラグ信号”01”、カウント数が 100～199 の時フラグ信号”10”、カウント数が 200～255 の時フラグ信号”11”、を出力するというものである。今回の実験では、動作の閾値である、カウント数が 99、100、199、200 の時に、正常にフラグ信号が発行されるかを確認した。

図 26、27、28、29 に実験の結果を示す。

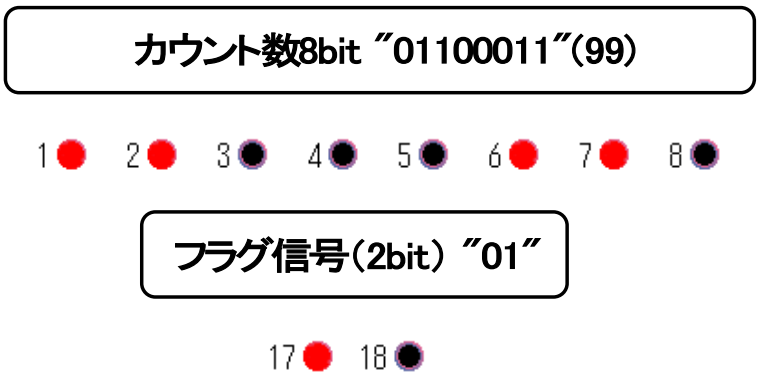

図 26　範囲によるフラグ信号発行”01”(99)

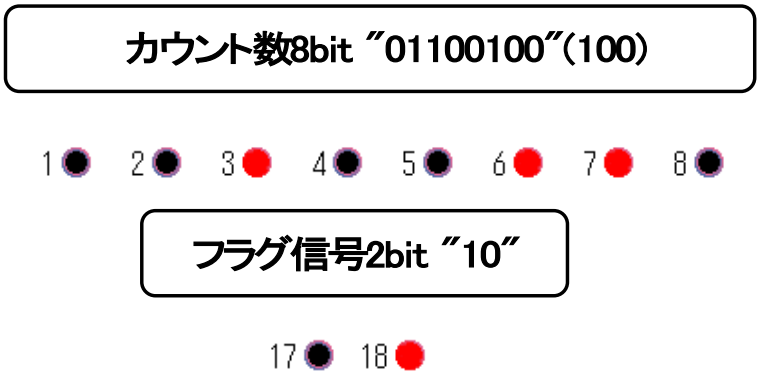

図 27　範囲によるフラグ信号発行”10”(100)

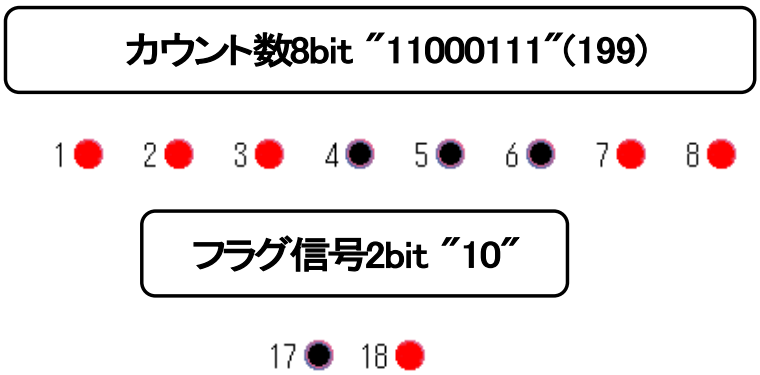

図 28　範囲によるフラグ信号発行”10”(199)

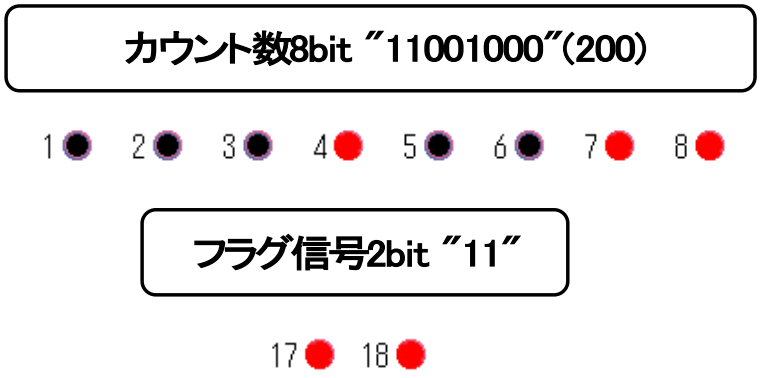

図 29　範囲によるフラグ信号発行”11”(200)

図 26～29 から分かるように、期待通りに動作している事がわかる。これにより、閾値を範囲で指定することが可能となった。
この実験で、閾値を範囲として設定することによりトリガー条件の自由度が広がったと言える。

4-6 2ch 分の信号の処理

ここまでの実験を経て、1ch 分の入出力とカウント数の範囲によって処理を分けることが可能になった。そのため次は入出力を 2ch に増やすことを考えた。
VHDL による記述では 2ch 分の動作の記述をする際、細かい部分でのタイミングの遅れを考慮しなければ、カウンタープログラムや、フラグ信号発行のプログアムの箇所は変更せず、階層構造に加えることでチャンネル数を機械的に増やすことができる。
これにより、比較的簡単に複数の入出力のチャンネル数を増やすことができた。

§5 考察

5-1 結論

今回行ってきた実験の結果として、FPGA ボードに入力した信号のパルス幅をクロックパルスを用いて計数することが可能となった。これにより、目的の①をプログラムで製作するという当初の目標は達成できた。更に、計数した結果のカウント数と、その数に応じたフラグ信号を同時に出力することにも成功した。また、範囲を設定して処理を分けることに成功したことで、トリガー回路製作の自由度が広がった。これにより、実際に望遠鏡で観測する際、空気シャワーと別のイベント（バックグラウンドやノイズとなる夜光等）をより正確に区別して、トリガー効率を上げることが期待できることから、目的の②のプログラムの製作も達成できたと言える。

ただ、カウント数を出力する際、入力信号のパルス幅が小さいときに、測定値に 1 カウント程度の誤差が出ることがあった。これは、最終的な出力には整数のみを用いているために、クロックパルスの 20[ns]では割り切れない少数を含む結果が生じ 、カウント数に誤差が出たものと考えられる。
この問題の最も簡単な解決策としては、クロックパルスのパルス幅を 20[ns]より小さくすることであるが、これにより、より精度の高い結果が得られると考えられる。尚、クロックのパルス幅は 3-8 のクロック信号の周波数調整によって可能である。

§6 巻末・実験に用いたプログラム

各々の実験に用いたプログラムのソース・コードを以下に記載する。

6-1 チュートリアルプログラム（実験方法 3-7）

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
   Port ( CLOCK : in  STD_LOGIC;
             DIRECTION : in  STD_LOGIC;
          COUNT_OUT : out  STD_LOGIC_VECTOR (3 downto 0));
end counter;

architecture Behavioral of counter is
signal count_int : std_logic_vector(3 downto 0) ;

begin

process (CLOCK)
begin
   if CLOCK='1' and CLOCK'event then
      if DIRECTION='1' then
                                        count_int <= count_int +1;
           elsif
                                        count_int <= count_int -1;
                    end if;
            end if;
end process;
 COUNT_OUT ＜= count_int;

end Behavioral;
```

6-2 クロックパルスのカウント（実験方法 3-8、実験 4-1）

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
    Port ( CLOCK : in  STD_LOGIC;
```

```
            DIRECTION : in  STD_LOGIC;
          COUNT_OUT : out  STD_LOGIC_VECTOR (7 downto 0));
end counter;


architecture Behavioral of counter is
signal count_int : std_logic_vector(7 downto 0) ;

begin

process (CLOCK)
begin
   if CLOCK='1' and CLOCK'event then
      if DIRECTION='0' then
                              count_int <= (others=>'0');
          elsif
         DIRECTION='1' then
                              count_int <= count_int +1;
                    end if;
           end if;
end process;
 COUNT_OUT ＜= count_int;

end Behavioral;
```

6-3 カウンタープログラム（実験 4-2、4-3）

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity counter2 is
          port( gCLK : in std_logic;
                              RST : in std_logic;
                              S_PMT : in std_logic;
                              LATCH : in std_logic;
                              OUT_CNT : out std_logic_vector(7 downto 0));
end counter2;

architecture Behavioral of counter2 is
          signal PW_count : std_logic_vector(7 downto 0);
          signal PW_data : std_logic_vector(7 downto 0);

begin

          process(gCLK,RST)
```

```
	begin
		if (RST='1') then
			PW_count <= (others =>'0');
		elsif (gCLK'event and gCLK='1') then
			if (S_PMT='1') then
				PW_count <= PW_count + '1';
			end if;
		end if;
	end process;

	process(RST,LATCH)
	begin
		if (RST='1') then
			PW_data <= (others =>'0');
		elsif (LATCH'event and LATCH='1') then
			PW_data <= PW_data + '1';
		end if;
	end process;
	OUT_CNT <= PW_data;

end Behavioral;
```

6-4 フラグ信号発行プログラム（実験 4-4）

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FLG is
	port( gCLK : in std_logic;
			RST : in std_logic;
			S_PMT : in std_logic;
			DATA_IN : in std_logic_vector(7 downto 0);
			FLAG : out std_logic_vector(1 downto 0);
			DATA_OUT : out std_logic_vector(7 downto 0);
			TP1 : out std_logic_vector(7 downto 0));
end FLG;

architecture Behavioral of FLG is

	component counter2
		port( gCLK : in std_logic;
				RST : in std_logic;
				S_PMT : in std_logic;
				OUT_CNT : out std_logic_vector(7 downto 0);
				TP1 : out std_logic_vector(7 downto 0));
	end component;
```

```
	signal F_data : std_logic_vector(1 downto 0);
	signal D : std_logic_vector(7 downto 0);
begin

	process(RST,gCLK)
	begin
		if (RST='1') then
			F_data <= (others =>'0');
		elsif (gCLK'event and gCLK='1') then
			if (D=X"FF") then
				F_data <= "11";
			elsif (D=X"FE") then
				F_data <= "10";
			elsif (D=X"FD") then
				F_data <= "01";
			end if;
		end if;
	end process;


	FLAG <= F_data;
	DATA_OUT <= D;

	Counter00:counter2
	port map(gCLK=>gCLK,RST=>RST,S_PMT=>S_PMT,OUT_CNT=>D,TP1=>TP1);
end Behavioral;
```

6-5 トリガー範囲拡大プログラム（実験 4-5）

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FLG is
	port( gCLK : in std_logic;
			RST : in std_logic;
			S_PMT : in std_logic;
			DATA_IN : in std_logic_vector(7 downto 0);
			FLAG : out std_logic_vector(1 downto 0);
			DATA_OUT : out std_logic_vector(7 downto 0);
			TP1 : out std_logic_vector(7 downto 0));
end FLG;

architecture Behavioral of FLG is

	component counter2
		port( gCLK : in std_logic;
				RST : in std_logic;
```

```
                                S_PMT : in std_logic;
                                OUT_CNT : out std_logic_vector(7 downto 0);
                                TP1 : out std_logic_vector(7 downto 0));
        end component;

        signal F_data : std_logic_vector(1 downto 0);
        signal D : std_logic_vector(7 downto 0);
begin

        process(RST,gCLK)
        begin
                if (RST='1') then
                        F_data <= (others =>'0');
                else
                        case(D) is
                                when( X"C9" to X"FF") => F_data <= "11";
                                when( X"64" to X"C8" )=> F_data <= "10";
                                when (X"00" to X"63" )=> F_data <= "01";
                                when others => F_data <= "00";
                        end case;
                end if;
        end process;


        FLAG <= F_data;
        DATA_OUT <= D;

        Counter00:counter2
        port map(gCLK=>gCLK,RST=>RST,S_PMT=>S_PMT,OUT_CNT=>D,TP1=>TP1);
end Behavioral;
```

6-6 2ch 分のフラグ発行プログラム（実験 4-6）

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity TOP1 is
        port( gCLK : in std_logic;
                        RST : in std_logic;
                        PMT : in std_logic_vector(1 downto 0);
                        LATCH : in std_logic;
                        LATCH1 : in std_logic;
                        LATCH2 : in std_logic;
                        FLAG0 : out std_logic_vector(1 downto 0);
                        FLAG1 : out std_logic_vector(1 downto 0);
                        DATA_OUT : out std_logic_vector(7 downto 0));
end TOP1;
```

```
architecture Behavioral of TOP1 is
	signal flag0_data,flag1_data : std_logic_vector(1 downto 0);
	signal DD00,DD01 : std_logic_vector(7 downto 0);
	signal PMT_SYN : std_logic_vector(1 downto 0);

	component counter2
	port( gCLK :in std_logic;
			RST : in std_logic;
			LATCH : in std_logic;
			S_PMT : in std_logic;
			OUT_CNT : out std_logic_vector(7 downto 0));
	end component;
begin

	DATA_OUT <= DD00 when LATCH1='1' else
					DD01 when LATCH2='1';

	process(RST,LATCH1)
	begin
		if (RST='1') then
			flag0_data <= (others =>'0');
		else
			case(DD00) is
				when( X"C9" to X"FF") => flag0_data <= "11";
				when (X"64" to X"C8" )=> flag0_data <= "10";
				when (X"00" to X"63" )=> flag0_data <= "01";
				when others => F_data <= "00";
			end case;
		end if;
	end process;
	FLAG0 <= flag0_data;


process(RST,LATCH2)
	begin
		if (RST='1') then
			flag0_data <= (others =>'0');
		else
			case(DD01) is
				when( X"C9" to X"FF") => flag0_data <= "11";
				when (X"64" to X"C8" )=> flag0_data <= "10";
				when (X"00" to X"63" )=> flag0_data <= "01";
				when others => F_data <= "00";
			end case;
		end if;
	end process;
	FLAG1 <= flag1_data;

	counter00:counter2
	port map(gCLK=>gCLK,RST=>RST,LATCH=>LATCH,S_PMT=>PMT_SYN(0),OUT_CNT=>DD00);

	counter01:counter2
	port map(gCLK=>gCLK,RST=>RST,LATCH=>LATCH,S_PMT=>PMT_SYN(1),OUT_CNT=>DD01);

end Behavioral;
```

謝辞

本研究を進めるにあたり、ご指導を頂いた指導教員の梶野文義教授並びに山本常夏准教授に感謝致します。また、多くの知識や示唆を頂いた宇宙粒子研究室の皆様に感謝致します。そして、大学で学ぶ機会を与えてくれた両親にお礼申し上げます。

ありがとうございました。

参考文献

- JEM-EUSO　望遠鏡のトリガー回路製作試験（篠田　幸秀　甲南大学学士論文）
- JEM-EUSO　望遠鏡のトリガー回路製作試験（吉田　賢司　甲南大学学士論文）
- 図解 VHDL 実習（堀　桂太郎　著）
- FPGA ISE アドバンスチュートリアル
- ISE10.1 クイックスタートチュートリアル
- JEM-EUSO 地球を見て宇宙を知る「地文台」
  http://jemeuso.riken.jp/
- Xilinx 社ホームページ
  http://www.xilinx.com/