



プロジェクト研究A VerilogとDE10-nanoを用いた CPU制作

1W192324

藤田 智也

目次



- 研究背景と研究目的
- 中間報告までに行ったこと
- 制作したCPUについて
- 動作確認について
- 工夫・苦勞した点
- まとめ
- 参考文献

研究背景と研究目的



□研究背景

- 個人でデスクトップPCを組み立てた際に、CPU内部がどのようなになっているか興味がわいた
- 授業での実習を通して、CPUの構造を学んだため、制作してみたいと考えた

□研究目的

- 実際にCPUを制作することで、CPUの内部構造について理解を深める

中間報告までに行ったこと

□ 環境構築

- QuartusのUbuntuへのインストール
- シリアル通信に用いるminicomの導入

□ Verilog-HDLの書き方の習得

□ DE-10nanoの使い方の習得

- LEDを光らせる回路の作成

□ Laboratory Exercise

- SRラッチ、Dラッチ、Dフリップフロップの作成
- 全加算器の作成

制作した単一サイクルCPU



□ 実装した命令

- 加減算やシフト演算などの単一サイクル算術命令
- 4つの分岐命令
- ロード・ストア命令

□ 作成したモジュール

- 8本の16bitレジスタ
- ALU
- プログラムカウンタ
- 命令メモリ、データメモリ
- 各モジュールをつなぐデータバス
- トップレベルモジュールなど周辺回路

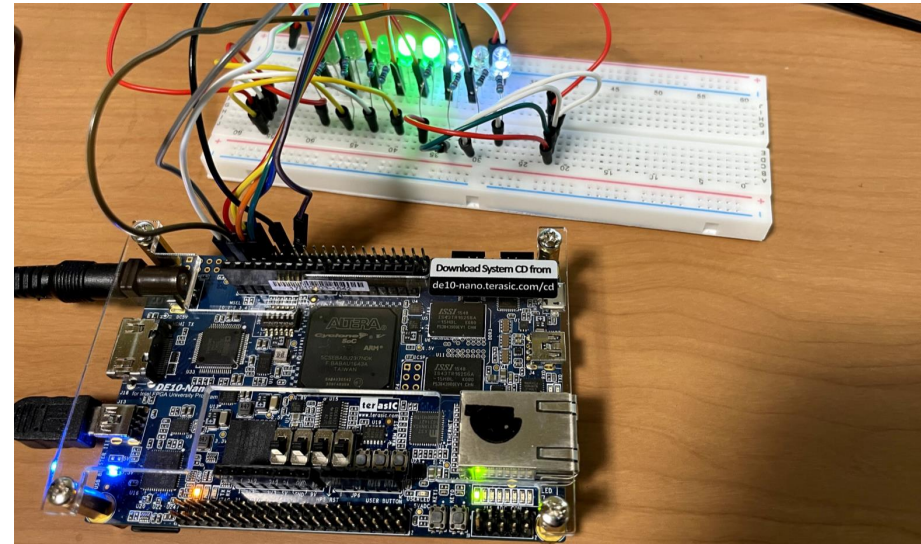
動作確認

□ 動作確認環境

- FPGA: DE10-nano
- 周波数: 50MHz

□ 動作確認方法

- メモリ上に保存されたデータを、スライドスイッチとLEDを用いて表示
- FPGA上にある8つのLEDに加えて、ピンから8つのLEDにつなげ、動作確認を行った



工夫した点



□ ブロック図を用いた制作物の整理

- ブロック図を書くことで、必要なモジュールや制作物の配線などが整理でき、具体的な方針が立てやすかった

□ Moduleごとの動作確認

- 全体での動作確認時に、各moduleに不具合があった場合、修正が難しくなる
- Moduleごとに動作確認を行うことで、全体での動作確認時に不具合が発生した場合でも、修正しやすくなった

苦勞した点



- ハードウェア記述言語の仕様の理解
- 作成したmodule全体の動作確認での動作しない原因の分析
- 実機上での動作確認時、光らないはずのLEDが暗く光ってしまう原因の究明

ハードウェア記述言語の仕様の理解



□ wireやregの使い分け

□ ハードウェア記述言語の同時実行仕様

- C/C++などの手続き型言語と異なり、複数の行が並列に同時実行される
- 手続き型言語のように考えてしまうと、意図しない動きになってしまう

作成したmodule全体の動作確認での動作しない原因の分析

- 初めて全体での動作確認をmodel simにより行った際には、ほとんどの出力が不定やハイインピーダンスなどであった
- 1つ1つ修正し、動作するものを完成させた。
修正箇所については、以下のような箇所であった
 - プログラムカウンタが不定になっていたため、プログラムカウンタを0で初期化した
 - UnClockプロセスをClockプロセスに修正した
 - レジスタ宣言時に、
“reg [15:0]rgstr[2:0]”
と宣言しており、8個ではなく3個しか宣言できていなかったため、8個に修正した


実機上での動作確認時、光らないはずのLEDが暗く光ってしまう原因の究明

- ○ ○ has unsafe behavior という Warning について修正したところ、LED が正しく出力された
 - ALUの出力である、outを決めるcase文の default文が記述されていなかった
→default文を追加した
 - RAMや動作確認用メモリで、Unclockプロセスを用いていた
→Clockプロセスに修正した

まとめ

- ■ ■ ■ ■
- 16bitCPUを制作し、FPGAやLEDを用いて動作確認した
- 工夫したこと
 - ブロック図の作成を行い、制作物について整理した
 - moduleごとに動作確認し、全体の動作確認での修正の負担を軽減した
- 制作したCPUの今後の展望
 - 32bit化
 - 実装命令を増やす
 - マルチサイクル命令の実装
 - パイプライン化

参考文献

- 
- 渡波 郁, CPUの創りかた, マイナビ出版, 2003
 - コンピュータアーキテクチャA, 実習用簡易CPU, RISC-V

詳しい報告内容 (以下レポートです)



- 制作したCPUの仕様
- 各種モジュールの説明
- 実機での動作確認

制作したCPUの仕様



- 実装した命令
- エンコード方法について
- 命令列の割り当てについて

実装した命令(R命令)

命令	演算内容
add	$rs0 = rs1 + rs2$
sub	$rs0 = rs1 - rs2$
xor	$rs0 = rs1 \wedge rs2$
or	$rs0 = rs1 \vee rs2$
and	$rs0 = rs1 \& rs2$
sll	$rs0 = rs1 \ll rs2$
srl	$rs0 = rs1 \gg rs2$
slt	$rs0 = (rs1 < rs2) ? 1 : 0$

実装した命令(I命令)



命令	演算内容
addi	$rs0 = rs1 + imm$
xori	$rs0 = rs1 \wedge imm$
ori	$rs0 = rs1 \vee imm$
andi	$rs0 = rs1 \& imm$

実装した命令(B命令)

命令	演算内容
beq	if(rs1 == rs2) PC = imm
bneq	if(rs1 != rs2) PC = imm
bl	if(rs1 < rs2) PC = imm
ble	if(rs1 <= rs2) PC = imm

実装した命令(L命令)



命令	演算内容
lw	$rs0 = M[rs1 + imm]$
sw	$M[rs1 + imm] = rs2$

エンコード方法について

□ R命令

15 14	13 11	10 8	7 5	4 2	1 0
0 0	rs2	rs1	rs0	fct	op

□ I命令

15 11	10 8	7 5	4 2	1 0
Imm	rs1	rs0	fct	op

□ B命令

15 14	13 11	10 8	7 5	4 2	1 0
Imm	rs2	rs1	Imm	fct	op

□ L命令

15 11	10 8	7 5	4 2	1 0
Imm	rs1	rs0/rs2(sw命令)	fct	op

命令列の割り当て

□ OP

OPコードを以下のように割り当てた

命令タイプ	op
R	00
I	01
B	10
L	11

命令列の割り当て

□ fct

fctを以下のように各命令ごとに割り当てた

■ R命令

命令タイプ	fct
add	000
sub	001
xor	010
or	011
and	100
sll	101
srl	110
slt	111

命令列の割り当て

■ I命令

命令	fct
addi	000
xori	001
ori	010
andi	011

■ L命令

命令	fct
lw	000
sw	001

命令列の割り当て



■ B命令

命令	fct
beq	000
bneq	001
bl	010
ble	011

各種moduleの説明



□制作したmodule

- レジスタ
- alu
- プログラムカウンタ
- デコーダ
- データパス
- CPUモジュール
- メモリ
- トップレベルモジュール

レジスタ



□register

CPUの演算結果を一時的に保存しておくモジュールである。

今回制作したCPUには8つのレジスタを搭載した。

x0レジスタが呼び出されたときは、必ず0を渡す仕様になっている。

alu



□ alu

CPUの演算を行うモジュールである。

レジスタ1、レジスタ2、aluctlの入力から、結果であるoutとB命令の真偽を出力する、btakenを出力する。

プログラムカウンタ



□pc

取り出す機械語命令列の、
命令メモリの番地の管理をする。

B命令の出力が真であれば、
入力された即値の番地へ、
そうでなければ前のプログラムカウンタの
値に1を足す。

デコーダ



□decoder

機械語命令列を受け取り、
他のmoduleに適切な入力となるよう、
デコードし出力する。

データパス



□ datapath

CPU内のモジュールが正しく動作するように、
wireなどでデータの受け渡しをする
モジュールである。

CPUモジュール



□T16

CPUの各モジュールの中で、最も上にあるモジュールである。

トップレベルモジュールと、データパスやプログラムカウンタをつないでいる。

メモリ



□ instmem

機械語命令列を格納しておくメモリモジュールである。
命令列はここから参照される。

□ datamem

演算結果のデータを格納しておくメモリモジュールである。
データはこのモジュールに保存され、このモジュールから読みだされる。

トップレベルモジュール



□ mycpu.v

クロックと動作確認用のスイッチ、LEDを入出力に持つ、トップレベルモジュールである。

メモリとCPUとのデータのやり取りや、クロック入力などの管理を行う。

動作確認用のデータメモリのコピーがとられる仕様になっている。

ソースコード






□ ソースコードのリンク

<https://drive.google.com/drive/folders/1epcARoRbP0xg9vLIMI3Li7SBg9S98Lex?usp=sharing>

PIN割り当て

□ ピンを以下のように割り当てた

All Pins

Named: *  Edit:   LED[8]

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Di
out LED[15]	Output	PIN_AH14	4A	B4A_NO	PIN_AH14	2.5 V		12mA (default)	1 (default)	
out LED[14]	Output	PIN_AF7	3B	B3B_NO	PIN_AF7	2.5 V		12mA (default)	1 (default)	
out LED[13]	Output	PIN_AH13	4A	B4A_NO	PIN_AH13	2.5 V		12mA (default)	1 (default)	
out LED[12]	Output	PIN_D8	8A	B8A_NO	PIN_D8	2.5 V		12mA (default)	1 (default)	
out LED[11]	Output	PIN_D11	8A	B8A_NO	PIN_D11	2.5 V		12mA (default)	1 (default)	
out LED[10]	Output	PIN_W12	3B	B3B_NO	PIN_W12	2.5 V		12mA (default)	1 (default)	
out LED[9]	Output	PIN_E8	8A	B8A_NO	PIN_E8	2.5 V		12mA (default)	1 (default)	
out LED[8]	Output	PIN_V12	3B	B3B_NO	PIN_V12	2.5 V		12mA (default)	1 (default)	
out LED[7]	Output	PIN_AA23	5A	B5A_NO	PIN_AA23	2.5 V		12mA (default)	1 (default)	
out LED[6]	Output	PIN_Y16	5A	B5A_NO	PIN_Y16	2.5 V		12mA (default)	1 (default)	
out LED[5]	Output	PIN_AE26	5A	B5A_NO	PIN_AE26	2.5 V		12mA (default)	1 (default)	
out LED[4]	Output	PIN_AF26	5A	B5A_NO	PIN_AF26	2.5 V		12mA (default)	1 (default)	
out LED[3]	Output	PIN_V15	5A	B5A_NO	PIN_V15	2.5 V		12mA (default)	1 (default)	
out LED[2]	Output	PIN_V16	5A	B5A_NO	PIN_V16	2.5 V		12mA (default)	1 (default)	
out LED[1]	Output	PIN_AA24	5A	B5A_NO	PIN_AA24	2.5 V		12mA (default)	1 (default)	
out LED[0]	Output	PIN_W15	5A	B5A_NO	PIN_W15	2.5 V		12mA (default)	1 (default)	
in clk	Input	PIN_V11	3B	B3B_NO	PIN_V11	2.5 V		12mA (default)		
in sw[3]	Input	PIN_W20	5B	B5B_NO	PIN_W20	2.5 V		12mA (default)		
in sw[2]	Input	PIN_W21	5B	B5B_NO	PIN_W21	2.5 V		12mA (default)		
in sw[1]	Input	PIN_W24	5B	B5B_NO	PIN_W24	2.5 V		12mA (default)		
in sw[0]	Input	PIN_Y24	5B	B5B_NO	PIN_Y24	2.5 V		12mA (default)		
<<new node>>										

Model Simによる動作確認

- 実機での動作確認をする前に、Model Simによる動作確認を行った。
- Model Simで確認できる部分で、正しく動作していないと思われる部分は修正した。

実機での動作確認

- 16bitCPUの動作確認を行った。
- DE10-nanoでは、LEDの数が8個と足りないため、LED8個とブレッドボードを用いて、動作確認した。

動作確認1のテストコード

□テストコード1

```
addi x1 x0 1
addi x2 x0 2
add x3 x1 x2
sub x3 x1 x2
xor x3 x1 x2
or x3 x1 x2
and x3 x1 x2
sll x3 x1 x2
srl x3 x1 x2
slt x3 x1 x2
xori x4 x1 1
ori x4 x1 1
sw x3 x0 2

andi x5 x1 1
addi x1 x1 1
beq x1 x2 13
sw x1 x0 1
lw x6 x0 1
addi x3 x6 1
addi x2 x2 1
bne x2 x3 18
addi x3 x0 4
addi x2 x2 1
bl x3 x2 21
ble x3 x2 21
sw x3 x0 2
```

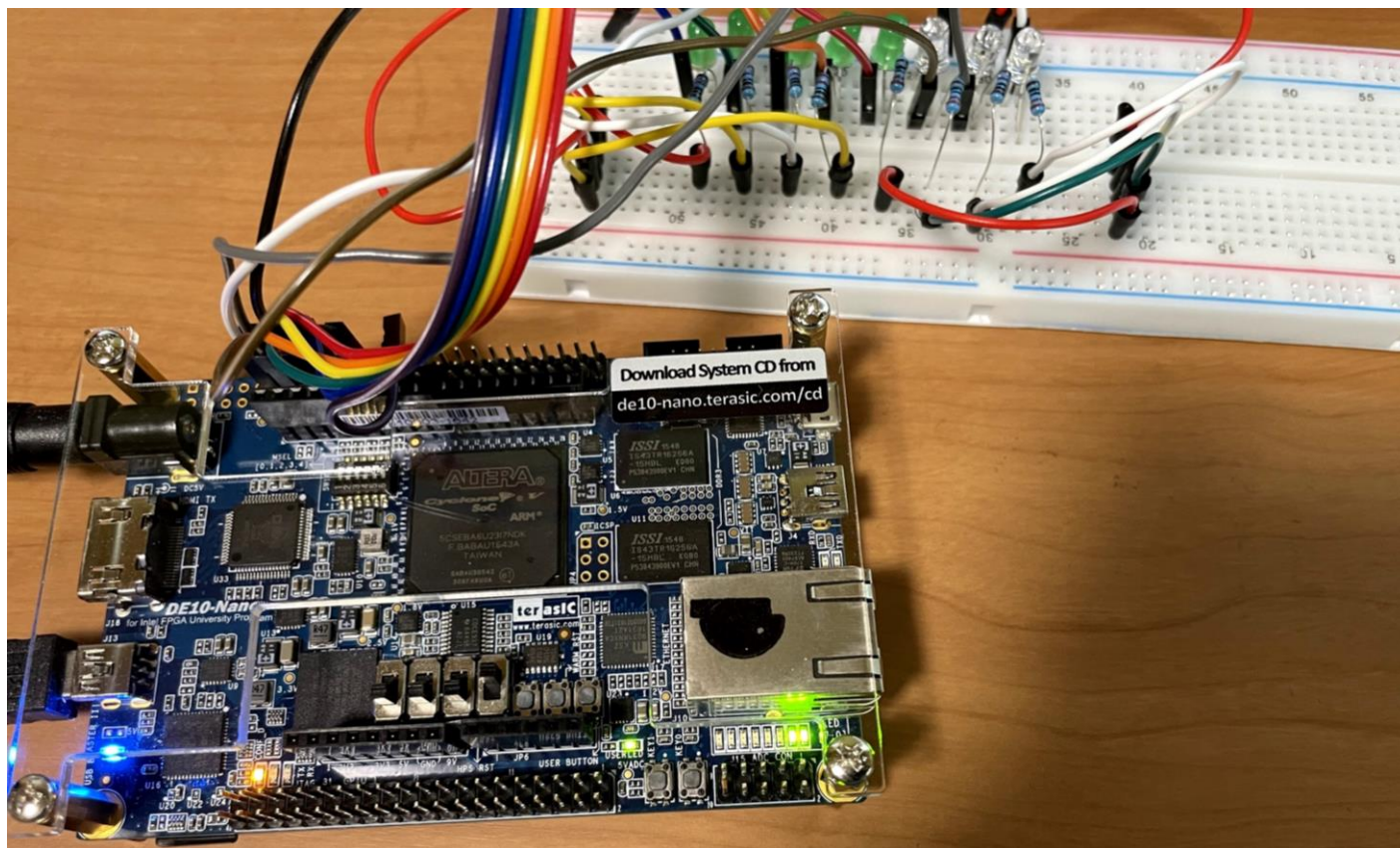
想定される出力

1番地:000000000000000011

2番地:0000000000000000111

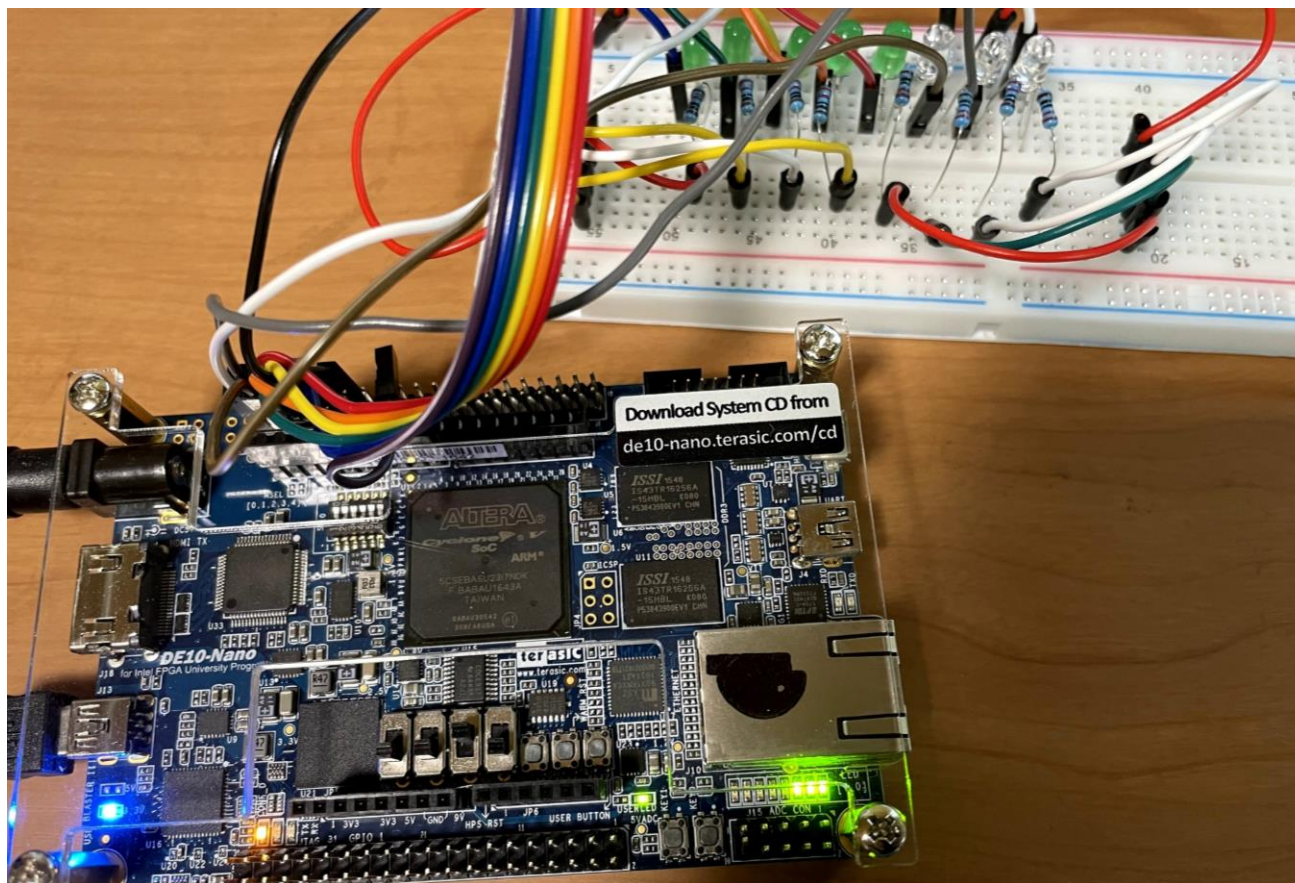
テストコード1の動作確認結果

□1番地



テストコード1の動作確認結果

□2番地



テストコード1の動作確認結果



1番地:000000000000000011

2番地:0000000000000000111

となり、想定される出力と一致した。

動作確認2のテストコード

□ テストコード2

```
addi x1 x0 11110
addi x2 x0 11111
add x3 x1 x2
addi x4 x0 3
sll x3 x3 x4
sw x3 x0 0
lw x1 x0 0
addi x3 x0 2
sub x2 x1 x3
addi x2 x2 1
bl x1 x2 9
ble x1 x2 9
addi x3 x0 1
srl x2 x2 x3
```

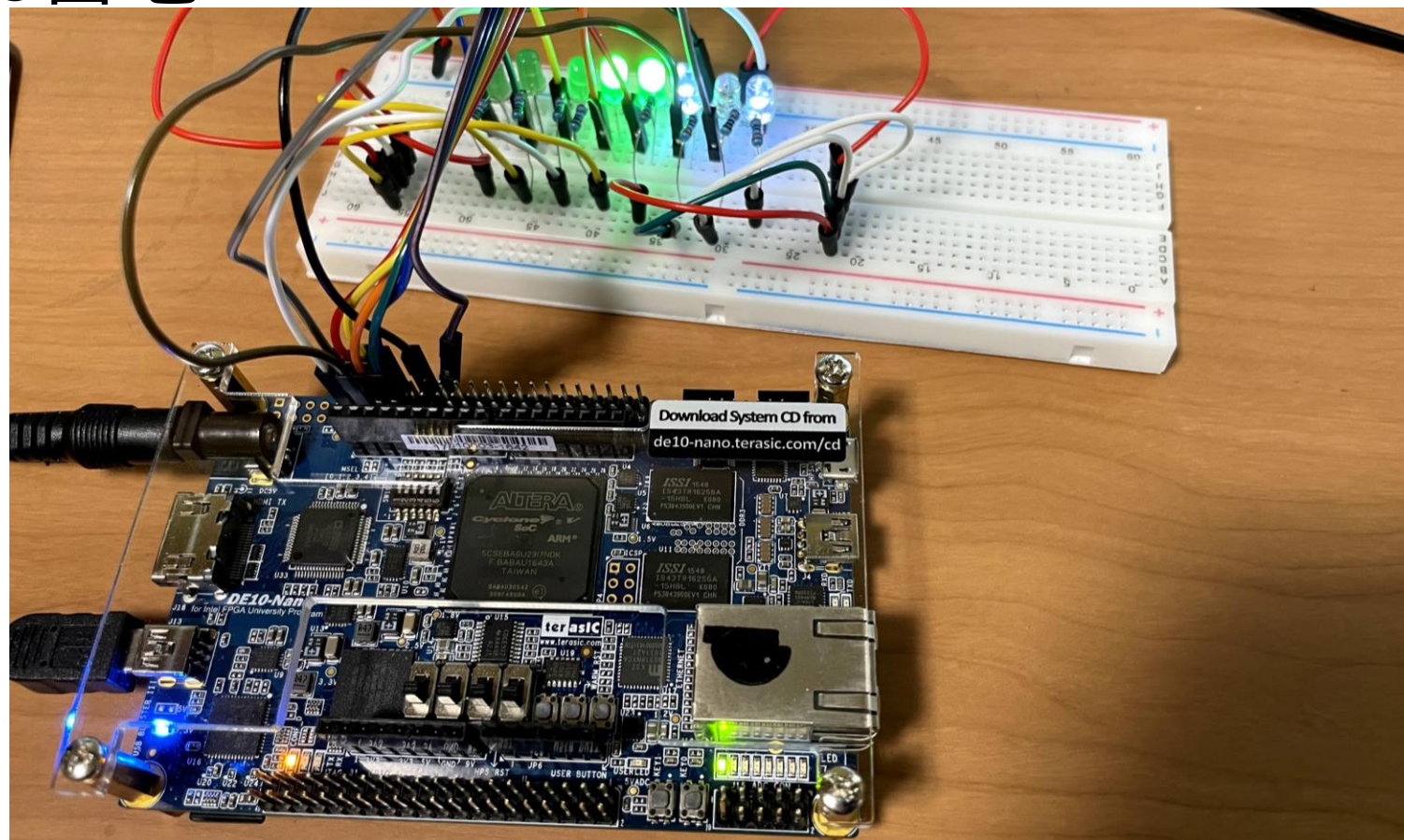
```
and x3 x1 x2
sw x3 x0 x1
or x3 x1 x2
sw x3 x0 2
xor x3 x1 x2
sw x3 x0 3
xori x3 x1 3
sw x3 x0 4
ori x3 x1 3
sw x3 x0 5
andi x3 x1 3
sw x3 x0 6
```

想定される出力

```
0番地:00011101 10000000
1番地:00001100 10000000
2番地:00011111 11000000
3番地:00010011 01000000
4番地:00011101 10000011
5番地:00011101 10000011
6番地:00000000 00000000
```

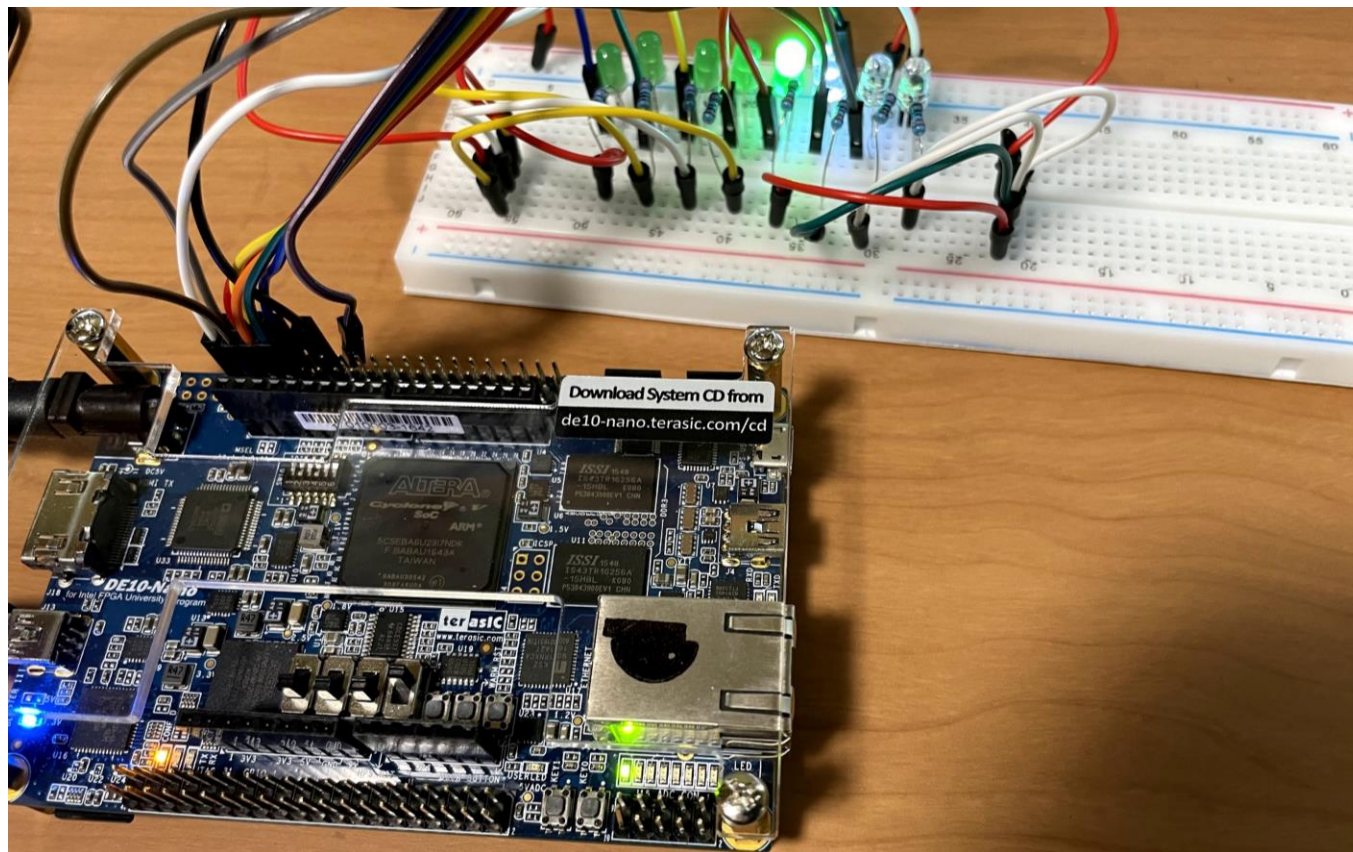
テストコード2の動作確認結果

□0番地



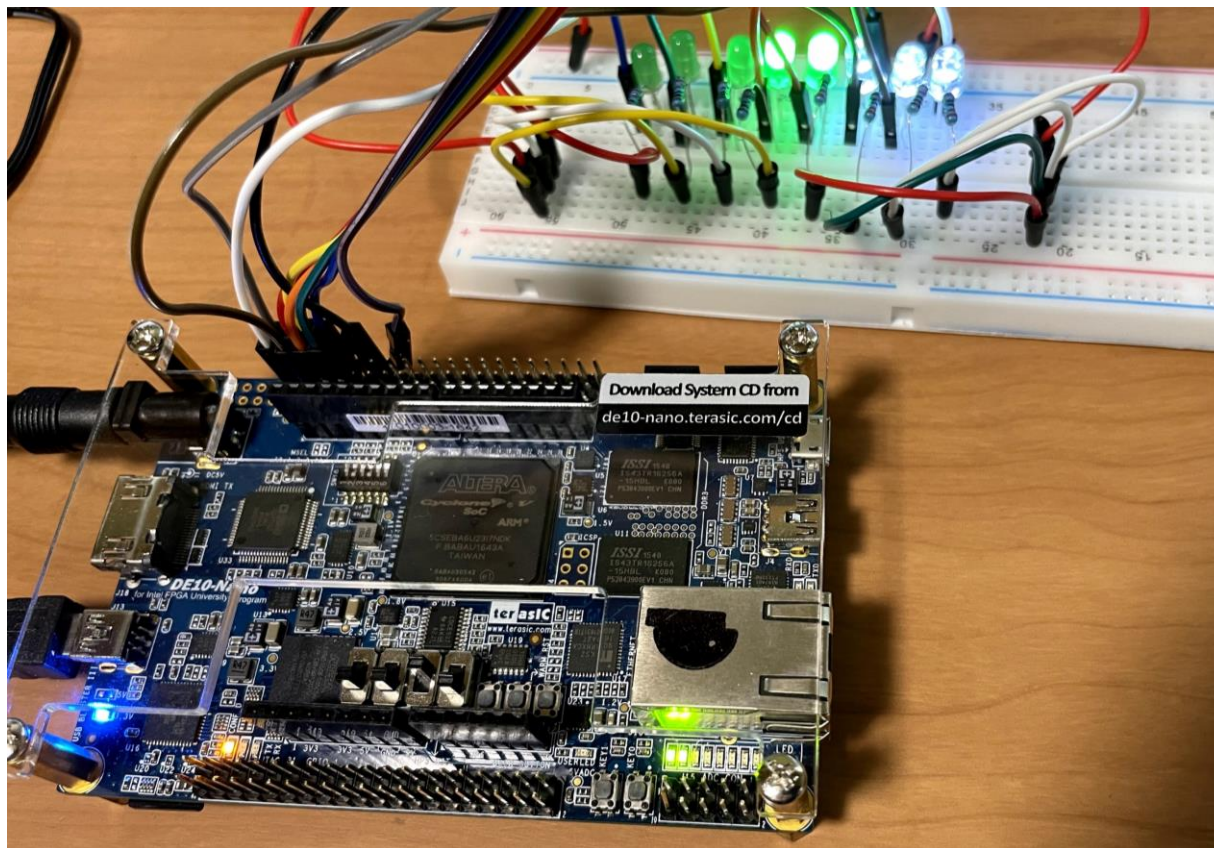
テストコード2の動作確認結果

□ 1番地



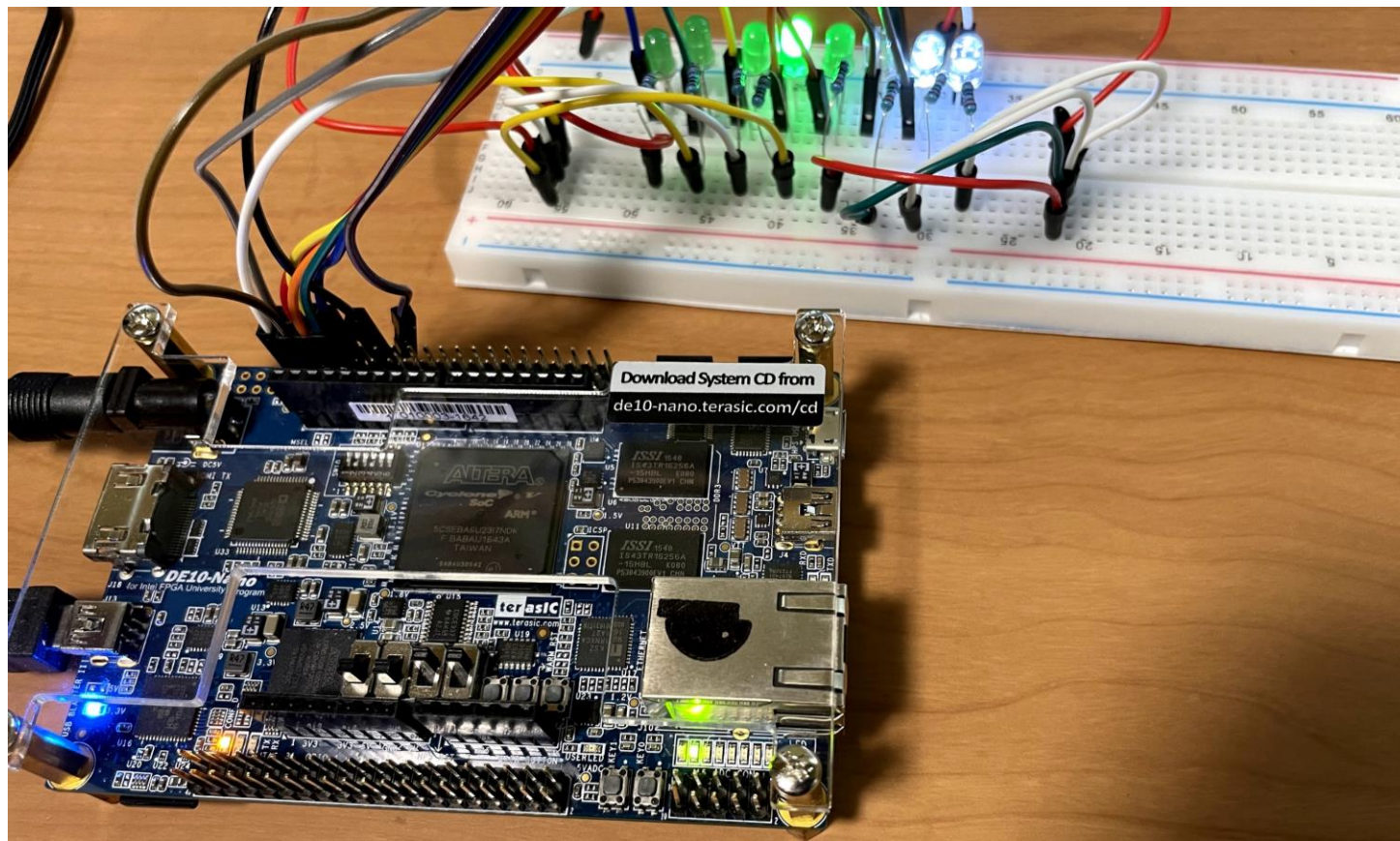
テストコード2の動作確認結果

□2番地



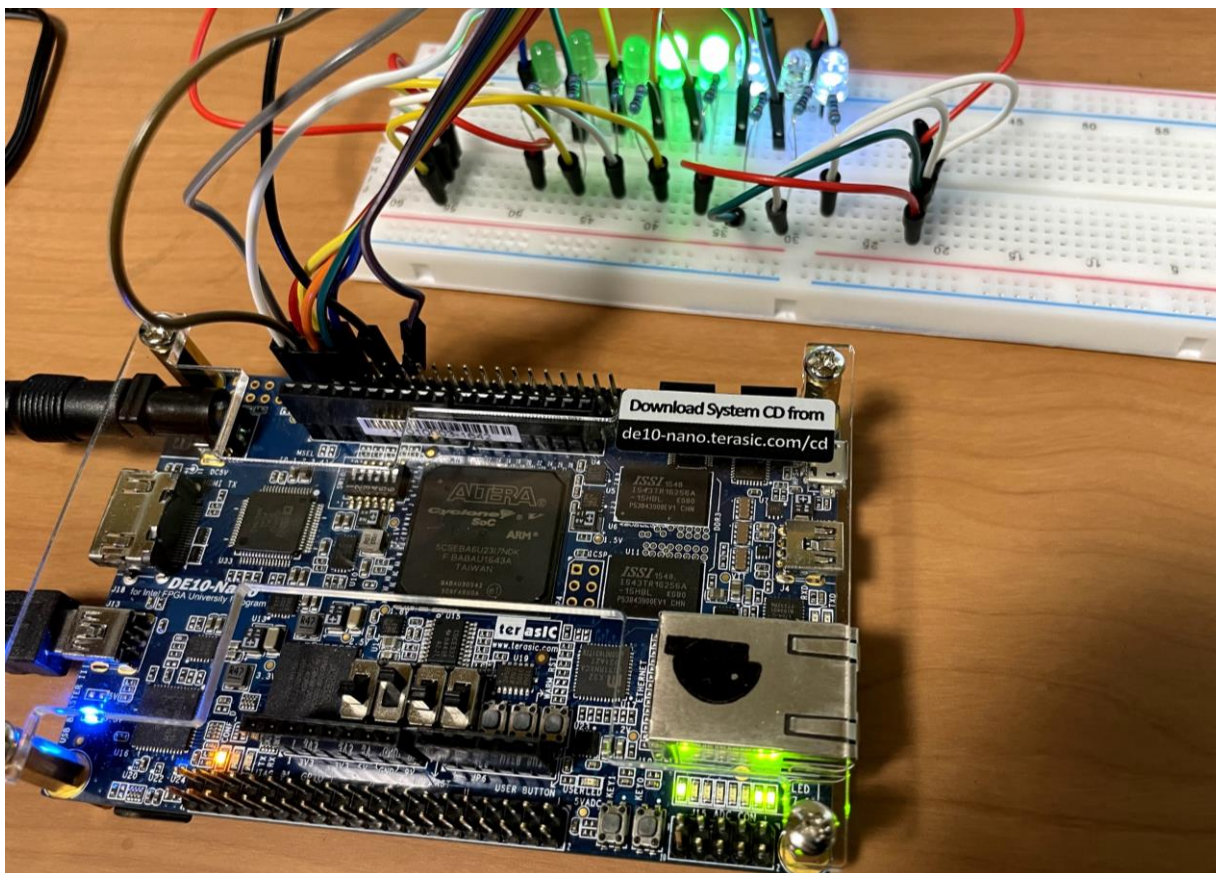
テストコード2の動作確認結果

□3番地



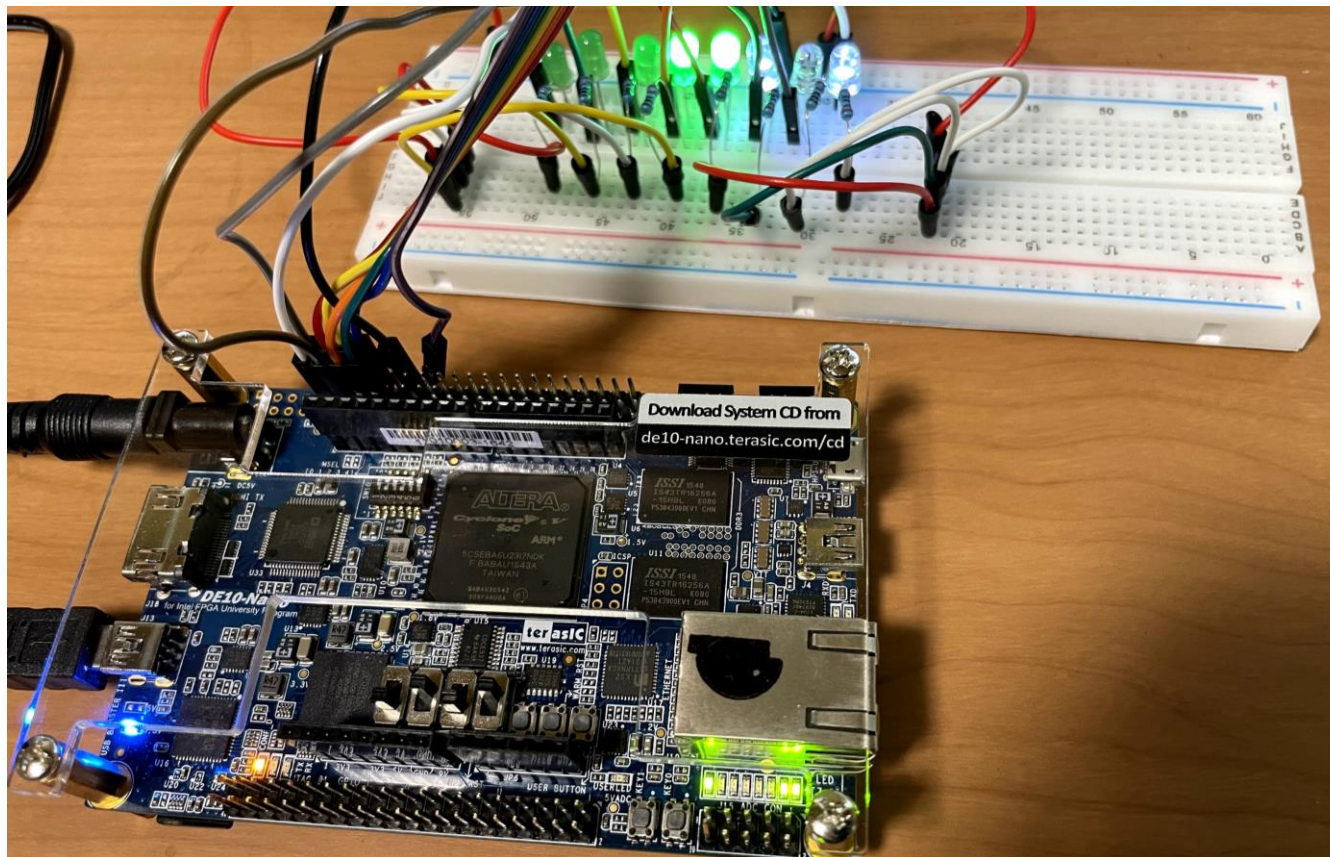
テストコード2の動作確認結果

□4番地



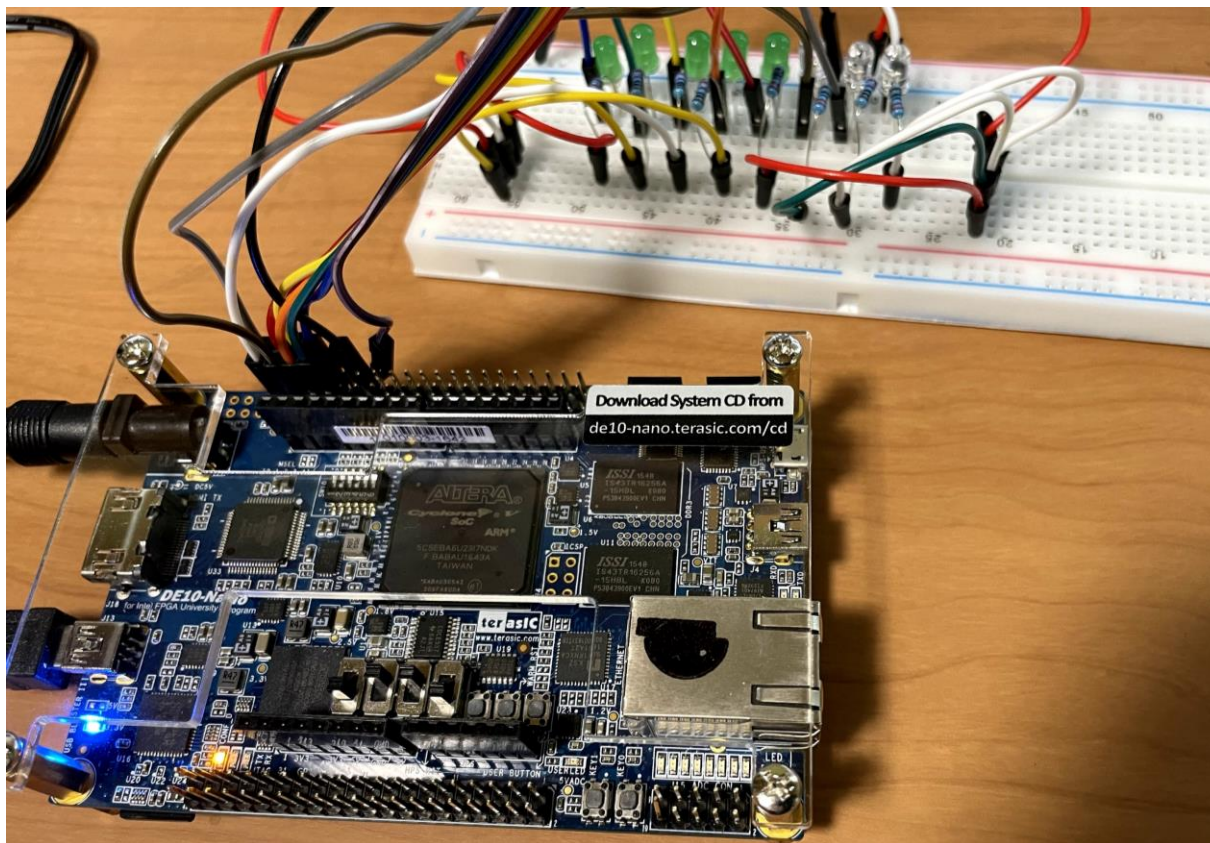
テストコード2の動作確認結果

□5番地



テストコード2の動作確認結果

□6番地



テストコード2の動作確認結果



0番地:00011101 100000000

1番地:00001100 100000000

2番地:00011111 110000000

3番地:00010011 010000000

4番地:00011101 10000011

5番地:00011101 10000011

6番地:00000000 000000000

となり、想定される出力と一致した。