

4ビット リレー加算器と RISC-V をつなぐ

田中 二郎¹

概要：80年前にツェーゼが作ったリレーによる計算機で使われた全加算器は、まったく無駄のない設計であった。その回路から論理式を再構築し、ツェーゼがいかにこの回路に至ったかを考察したかったが、ツェーゼの発想までは理解できていない。リレーによる論理回路は、とくに OR 回路をいかに Wired OR で組めるか重要であることがわかった。

また、2進数で動く回路を人にわかりやすい10進で表示するためのロジックは複雑であり、リレーのみで構成することが難しいことがわかった。そのため、最新のCPUであるRISC-Vをユーザインターフェース用に使用することとした。

キーワード：Zuse,Z3, リレー, 加算器,RISC-V

1. リレー式計算機

コンラート・ツェーゼ (Konrad Zuse, 1910-1995) はドイツの発明家で、プログラム制御式リレー・コンピュータ Zuse Z3 を1941年に稼働させた。Z3は、およそ2600個のリレーを使い(うち演算用には約600個を使用)[1]、さん孔テープでプログラム可能な計算機である。

1988年には、Z3が任意のチューリングマシンをシミュレート可能であることが示された。[2]

Z3は1945年に戦争で破壊されたが、1963年にレプリカが作成され、ミュンヘンのドイツ博物館で展示されている。[3]

1.1 リレーによる論理回路

(メカニカル)リレーとは、継電器ともいい、電磁石でスイッチを動かす機構である。電磁石に電気が流れた時にONになるのをa接点もしくはメーク接点,NO(Normally Open)といい、電気が流れていないときにONな接点をb接点もしくはブレイ

ク接点,NC(Normally Close)という。a,b両接点をもつ場合、切り替えの中心となる接点をc接点という。また、その他にも様々な呼び方があり、回路図の書き方も様々である。

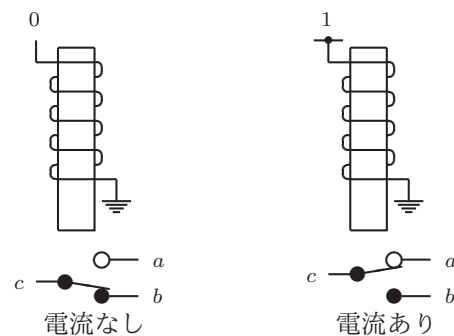


図1 リレー

リレーは、その性質上、電磁石に電気が流れたり、流れなくなってから接点がON/OFFするまでの動作に時間がかかる。さらに、動作してから接点の接続が安定するまでON/OFFが繰り返されることがある(チャタリング)。

¹ jiro@bimyo.jp



図 2 DC12V 4 回路 2 接点リレー

また,リレーの接点は,通常,a接点が入 ON になる前に b 接点が入 OFF になり (Break before Make),c 接点が入 a,b どちらにも接続されていない瞬間が存在する. 逆に,b 接点が入 OFF になる前に a 接点が入 ON になる (Make before Break) ももあり,この場合は c 接点が入 a,b 両方と接続される瞬間がある. この場合は,a,b 間も接続されることとなり,回路設計で留意する必要がある.

電気が流れている (電圧がかかっている) 状態を 1,電流が流れていない (電圧がかかっていない) 状態を 0 と考え,論理回路を作成することができる. この時に注意することは,電流が流れていない (電圧がかかっていない) 状態というのが,接地状態ではなく,どこにも接続されていない状態であることである. この定義に従ったリレー回路では,電源装置の一端はリレーの電磁石端子にのみ接続され,接点側には接続されない. そのため,回路の異常動作等により電源が短絡することはない.

1.2 Buffer, NOT, Selector

リレーをひとつ使うことで,a 接点として絶縁した別回路を作成することができる. また b 接点により NOT(\bar{A}) を作成することができる. この時,c 接点は電源装置のうち,リレーの電磁石端子に接続するのは反対側を接続する.

また,a,b 接点を入力,c 接点を出力とすると,Selector($Ax + \bar{A}y$) になる.

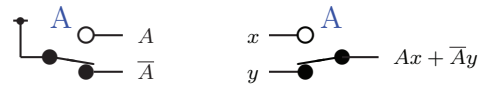


図 3 Buffer, NOT, Selector

1.3 AND, OR

接点を直列に接続することで,AND(Ax) を作成することができる. b 接点を使うことで,NOT AND($\bar{A}x$) にすることもできる. OR は単純に並列にすると入力側の値にまで影響が及ぶので,Selector を使い $A \cdot 1 + \bar{A}x$ とする. a,b 接点を逆にすると,NOT OR($\bar{A} \cdot 1 + Ax$) となる.

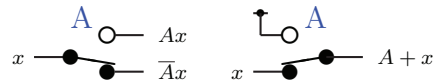


図 4 AND,OR

1.4 Wired OR

$xy + \bar{x}z$ という論理回路は,Selector を使わなくても,線を直接接続することで OR となる.

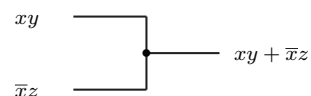


図 5 Wired OR

1.5 XOR の実現

通常の XOR($A \oplus B = A\bar{B} + \bar{A}B$) の論理回路を,上述の NOT,AND,OR で作成すると複雑になるが,c 接点を選択回路として使う ($c = xa + \bar{x}b$) ことにより,XOR が簡単に実現できる.

どちらかの回路の a,b 接点を入れ替えると,XNOR ($\overline{A \oplus B} = \overline{A\bar{B} + \bar{A}B}$) となる.

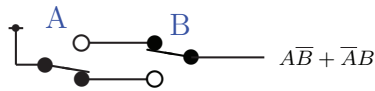


図 6 Buffer, NOT と Selector による XOR

1.6 加算器

半加算器は、結果 $S = A \oplus B$, キャリー $Cout = AB$ であり、簡単に実現できる。

全加算器は半加算器ふたつと OR で作成できるが、多段にした場合、前段のキャリー出力を次段の入力とすると、リレーの伝播時間分の遅れが生じる。これを回避するためには、キャリーを電磁石に流さない回路にしなければならない。論理式で書く場合、 $x \oplus y$ の x や y にキャリーを使えないということである。しかし、全加算器の場合、結果 $S = A \oplus B \oplus Cin$ となり、 \overline{Cin} が必要となる。

そこで、キャリー入力 Cin だけでなく \overline{Cin} 入力を用意する ('dual rail') ことで、 $S = A \oplus B \oplus Cin = A \cdot (\overline{B \oplus Cin}) + \overline{A} \cdot (B \oplus Cin) = A \cdot (\overline{B \cdot Cin} + B \cdot \overline{Cin}) + \overline{A} \cdot (B \cdot \overline{Cin} + \overline{B} \cdot Cin)$ とし、 A による Selection と B, \overline{B} による AND/NOT AND, Wired OR だけの回路で構成することができ、キャリーを電磁石に流す必要がなくなる。このとき必要となる $B \cdot Cin, \overline{B} \cdot Cin$ は AND/NOT AND の 1 回路で構成でき、同様に $B \cdot \overline{Cin}, \overline{B} \cdot \overline{Cin}$ も 1 回路で構成できる。よって、加算の結果は 3 回路で構成できる。

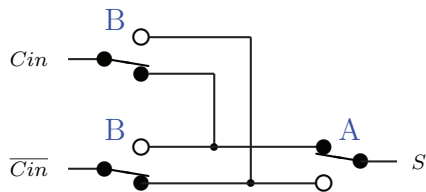


図 7 加算の結果

キャリー出力 $Cout = A \cdot B + (A \oplus B) \cdot Cin = A \cdot B + A \cdot \overline{B} \cdot Cin + \overline{A} \cdot B \cdot Cin = B \cdot (A \cdot 1 + \overline{A} \cdot Cin) + \overline{B} \cdot (A \cdot Cin)$ であり、 B による Selector と、 A による Wired OR と AND

みっつで構成できる。しかし、このうち $\overline{A} \cdot Cin$ と $A \cdot Cin$ は 1 回路で構成できる。また、必要となった \overline{Cout} は、 $\overline{Cout} = \overline{A \cdot B + (A \oplus B) \cdot Cin} = \overline{B \cdot (A \cdot Cin) + \overline{B} \cdot (A \cdot 1 + A \cdot \overline{Cin})}$ となる。 $Cin, Cout$ と同様に $\overline{A} \cdot \overline{Cin}$ と $A \cdot \overline{Cin}$ は 1 回路で構成できる。さらに $A \cdot 1$ と $\overline{A} \cdot 1$ も 1 回路で生成できるので、全加算器に必要な回路数は A, B 各 4 回路の合計 8 回路となる。

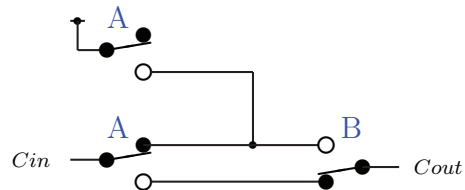


図 8 キャリー

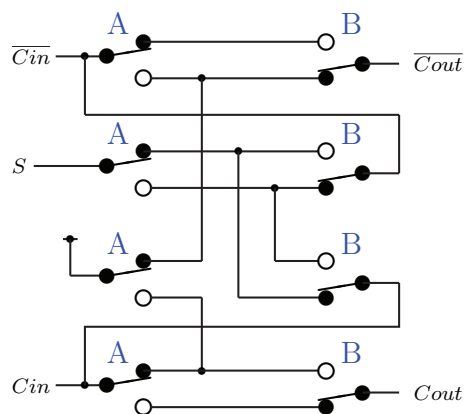


図 9 全加算器

表 1 全加算器

Cin	\overline{Cin}	A	B	$Cout$	\overline{Cout}	S
0	1	0	0	0	1	0
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	1
1	0	0	1	1	0	0
1	0	1	0	1	0	0
1	0	1	1	1	0	1

1.7 4ビット加算器

全加算器が,A用の4回路リレーと,B用の4回路リレーで構成できた.これを4段接続することにより,4ビットの加算器となる.最下位のキャリ-は $C_{in} = 0$ (どこにも接続しない), $\overline{C_{in}} = 1$ (電源に接続)に固定する.

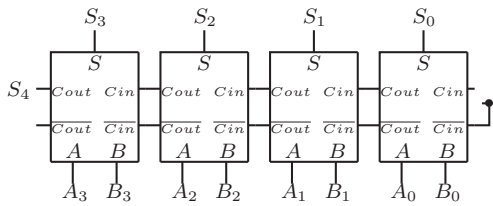


図 10 4ビット加算器

2. 入出力

入力には,ロータリ DIP スイッチ (16進リアルコード) を使用し,0~15までの値を入力できる.

表示には7セグメントLEDを使用することとした.そのため,5bit($S_0 \sim S_4$)⇒10進7セグメント2桁($a_0 \sim g_1$)への変換が必要である.

これを論理式で記述すると,

$$a_0 = \overline{S_4}S_2S_0 + \overline{S_3}S_1\overline{S_0} + S_4S_1\overline{S_0} + S_4\overline{S_2}S_0 + S_3S_2\overline{S_1} + S_4\overline{S_3}S_0 + \overline{S_4}S_3\overline{S_1} + \overline{S_4}S_2\overline{S_0}$$

$$b_0 = S_3\overline{S_1}S_0 + \overline{S_3}S_1S_0 + \overline{S_4}S_2 + S_4S_1S_0 + S_4\overline{S_3}S_1 + S_4\overline{S_3}S_0 + S_3S_2\overline{S_0} + S_3S_2\overline{S_1} + S_2\overline{S_1}S_0$$

$$c_0 = S_4\overline{S_2} + S_0 + S_3S_1 + S_4\overline{S_1} + \overline{S_2}S_1 + \overline{S_4}S_3S_2$$

$$d_0 = \overline{S_4}S_2\overline{S_1}S_0 + \overline{S_4}S_3S_2S_0 + \overline{S_3}S_2S_1 + \overline{S_3}S_1\overline{S_0} + S_3\overline{S_1}S_0 + S_4\overline{S_3}S_1 + S_4S_1\overline{S_0} + S_3S_2\overline{S_1} + S_4\overline{S_3}S_0 + \overline{S_4}S_2\overline{S_0}$$

$$e_0 = \overline{S_3}S_1\overline{S_0} + S_4S_1\overline{S_0} + S_4\overline{S_3}S_0 + \overline{S_4}S_2\overline{S_0} + S_3S_2\overline{S_1}S_0$$

$$f_0 = \overline{S_4}S_3S_2\overline{S_1} + S_4\overline{S_3}S_2S_1 + \overline{S_4}S_3S_2S_1 + S_3\overline{S_2}S_1 + S_4S_3\overline{S_1} + S_3S_1\overline{S_0} + \overline{S_3}S_1\overline{S_0} + \overline{S_4}S_3S_2\overline{S_0}$$

$$g_0 = \overline{S_3}S_2S_1 + \overline{S_4}S_2\overline{S_1} + S_4\overline{S_2}S_0 + \overline{S_4}S_3S_2 + S_4\overline{S_3}S_1 + S_3\overline{S_1} + \overline{S_3}S_1\overline{S_0}$$

$$a_1 = S_4S_2 + S_4S_3$$

$$b_1 = S_3S_1 + S_3S_2 + S_4$$

$$c_1 = S_4\overline{S_3}S_2 + \overline{S_4}S_3S_1 + \overline{S_4}S_3S_2 + S_3S_2S_1$$

$$d_1 = S_4S_2 + S_4S_3$$

$$e_1 = S_4\overline{S_3}S_2 + S_4S_3\overline{S_2} + S_4S_2\overline{S_1}$$

$$f_1 = 0$$

$$g_1 = S_4S_2 + S_4S_3$$

であり,これをリレーのみの回路で作成するのは,回路数が増え,困難である.

2.1 RISC-V

5bit($S_0 \sim S_4$)⇒10進7セグメント2桁($a_0 \sim g_1$)の変換のために最新のRISCであるRISC-Vの評価用ボードを使用した.

RISC-Vは,2010年より開始されたオープンアーキテクチャのISAである.2017年にはArduino-compatibleな評価用ボードHiFive1が発売され,安価に使用することができるようになった.開発環境も整備されており,HiFive1の開発にはUSBで接続するだけで,特別なハードウェアは必要ない.

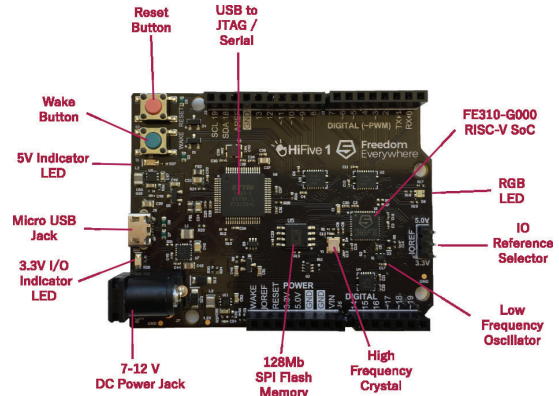


図 11 HiFive[4]

RISC-Vの入出力用のGPIOは0~23まで定義されているが,HiFive1ではそのうち19本が使用可能である.

ただし,UARTの2本はデバッグ用に使われており,LEDの3本も使用が難しい.つまり使えるのは14本であり,5入力14出力の変換にばピン数が足りない.

そこで,出力はBCD2桁とし,BCD⇒7セグメントの変換には専用IC(74HC4511)を使うこととした.これにより,出力は6bitとなり,GPIOの0~5までを使用することとした.

また,入力には電圧変換のためにフォトカプラ

表 2 HiFive1 の GPIO

GPIO	IOF0	IOF1	LED
0		PWM0_0	
1		PWM0_1	
2	SPI1:SS0	PWM0_2	
3	PI1:SD0/MOSI	PWM0_3	
4	SPI1:SD1/MISO		
5	SPI1:SCK		
9	SPI1:SS2		
10	SPI1:SS3	PWM2_0	
11		PWM2_1	
12		PWM2_2	
13		PWM2_3	
16	UART0:RX		
17	UART0:TX		
18			
19		PWM1_1	GREEN
20		PWM1_0	
21		PWM1_2	BLUE
22		PWM1_3	RED
23			

(TLC621) を使用し,GPIO の 9~13 に接続した。

2.2 UI プログラム

プログラムは $10\mu\text{s}$ ごとにカウントされるタイマの値をみて,10ms 毎に入力値を読み込んで 2 進 10 進変換をおこない, 出力する。入力に変化があれば, デバッグ用コンソールに GPIO の値を出力する。

ついでに,16 回 (160ms) 毎に, ボード上の LED の色を変化させている。

質疑・応答

Q 7セグLEDをダイナミック点灯にしないのは?

A 2桁なので,配線の手間があまり変わらない。

Q HiFive1の価格は?

A HiFive1は1万円弱です。

Q 最新のボードの値段は?

A HiFive Unleashedは999ドルとのこと。

Q HiFive1は,Arduino互換といえる?

A ほぼ,Arduino互換です。

Q I/Oの電圧は?

A 3.3Vと5Vをジャンパで切り替えられます。

Q 電源は?

A リレーは12Vで駆動しています。

HiFive1は7~12V OKなので,12Vを供給しています。

HiFive1搭載のレギュレーターの5V出力を,ロジックIC用に使っています。

参考文献

- [1] <http://ed-thelen.org/comp-hist/Reckoners-ch-2.html>
- [2] <https://ieeexplore.ieee.org/document/707574>
- [3] <https://www.deutsches-museum.de/sammlungen/meisterwerke/meisterwerke-iii/z3-und-z4/>
- [4] https://sifive.cdn.prismic.io/sifive%2F9c57065b-6d28-465b-b67d-f416894123a9_hifive1-getting-started-v1.0.2.pdf

```

#include <stdint.h>
#include "platform.h"

void _putc(char c) {
    while ((int32_t) UART0_REG(UART_REG_TXFIFO) < 0);
    UART0_REG(UART_REG_TXFIFO) = c;
}

int _getc(char * c){
    int32_t val = (int32_t) UART0_REG(UART_REG_RXFIFO);
    if (val > 0) {
        *c = val & 0xFF;
        return 1;
    }
    return 0;
}

void _puts(const char * s) {
    while (*s != '\0'){
        _putc(*s++);
    }
}

void _hex1(int32_t v) {
    char * hex = "0123456789ABCDEF";
    _putc(*(hex + (v & 0xf)));
}

void _hex2(int32_t v) {
    _hex1(v >> 4);
    _hex1(v);
}

void _hex4(int32_t v) {
    _hex2(v >> 8);
    _hex2(v);
}

void _hex8(int32_t v) {
    _hex4(v >> 16);
    _hex4(v);
}

int main (void)
{
    PRCI_REG(PRCI_HFROSCCFG) |= ROSC_EN(1);
    PRCI_REG(PRCI_PLLCFG) = (PLL_REFSEL(1) | PLL_BYPASS(1)); _puts(" ");
    PRCI_REG(PRCI_PLLCFG) |= (PLL_SEL(1));
    PRCI_REG(PRCI_HFROSCCFG) &= ~(ROSC_EN(1));
    GPIO_REG(GPIO_OUTPUT_VAL) |= IOFO_UART0_MASK;
    GPIO_REG(GPIO_OUTPUT_EN) |= IOFO_UART0_MASK;
    GPIO_REG(GPIO_IOF_SEL) &= ~IOFO_UART0_MASK;
    GPIO_REG(GPIO_IOF_EN) |= IOFO_UART0_MASK;

    UART0_REG(UART_REG_DIV) = 138;
    UART0_REG(UART_REG_TXCTRL) = UART_TXEN;
    UART0_REG(UART_REG_RXCTRL) = UART_RXEN;
    volatile int i=0;
    while(i < 10000){i++;}
    _puts("SiFive START\r\n");

    GPIO_REG(GPIO_INPUT_EN) |= 0x3e00;
    GPIO_REG(GPIO_PULLUP_EN) &= ~0x3e00;
    GPIO_REG(GPIO_OUTPUT_EN) = 0x3f;
    GPIO_REG(GPIO_INPUT_EN) &= ~(0x1 << RED_LED_OFFSET
        | 0x1 << GREEN_LED_OFFSET
        | 0x1 << BLUE_LED_OFFSET));
    GPIO_REG(GPIO_OUTPUT_EN) |= (0x1 << RED_LED_OFFSET
        | 0x1 << GREEN_LED_OFFSET
        | 0x1 << BLUE_LED_OFFSET));

    int relay = 0;
    int last = 0;
    int led = 0;
    volatile uint64_t * now
        = (volatile uint64_t*)(CLINT_CTRL_ADDR + CLINT_MTIME);
    volatile uint64_t then = *now + 1000;
    while(1){
        // wait 10mS
        while (*now < then) { }
        then += 1000;
        // input from Relay
        relay = (~GPIO_REG(GPIO_INPUT_VAL) >> 9) & 0x1f;
        // binary to BCD
        relay = int(relay / 10) * 0x10 + (relay % 10);
        // output to LED
        GPIO_REG(GPIO_OUTPUT_VAL) = relay | 0x6b0000;
        // RGB-LED blink
        if(led & 0x10) {
            GPIO_REG(GPIO_OUTPUT_VAL) &= ~(0x1 << BLUE_LED_OFFSET);
        }
        if(led & 0x20) {
            GPIO_REG(GPIO_OUTPUT_VAL) &= ~(0x1 << RED_LED_OFFSET);
        }
        if(led & 0x40) {
            GPIO_REG(GPIO_OUTPUT_VAL) &= ~(0x1 << GREEN_LED_OFFSET);
        }
        led = (led + 1) & 0x7f;
        // console log
        if(last != relay) {
            _hex8(GPIO_REG(GPIO_INPUT_VAL));
            _puts(" ");
            _hex8(GPIO_REG(GPIO_OUTPUT_VAL));
            _puts("\r\n");
            last = relay;
        }
    } // while(1)
}

```