ディジタルシステム設計
最終レポート

035743A：比嘉雅樹

提出日：2005/08/10

1　課題１

32 ビットの 2 つの数 AIN、BIN を加算して結果 YOUT を出力する回路を以下の 2 つの方針で VHDL にて設計し、それぞれに最速の加算時間を実現するように回路合成を行い、結果を比較せよ。結果には回路規模、クリティカルパスの遅延時間を含む。

1. $YOUT <= AIN + BIN$; という単純な記述を用いて実現し、デザインアナライザーの機能に頼って高速な回路を実現する。
2. 教科書 PP.219-227 にあるキャリー先見方式を VHDL で実現し、さらにデザインアナライザーの最適化機能を使って高速な回路を実現する。

1.1　VHDL 記述

1.1.1　単純な記述の加算器

２つの数の計算を、$YOUT <= AIN + BIN$ と簡単に記述する事で実現した。その際にデザインアナライザーで合成した回路と回路規模、遅延時間を以下に記す。 回

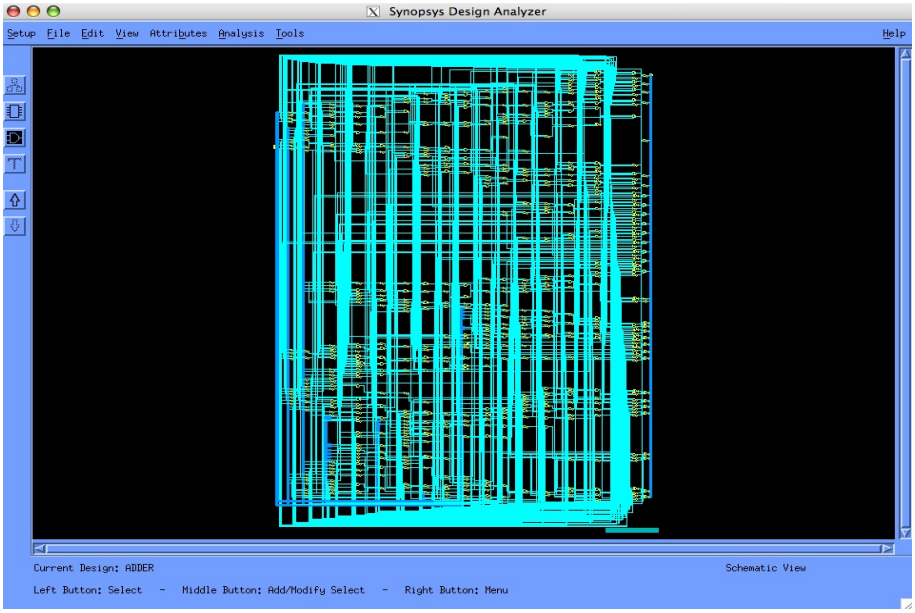

図 1: 合成した回路

路規模：853
遅延時間：4.57

1.1.2 キャリー先見方式を用いた加算器

次に、加算器の計算にキャリー先見方式を用いて設計した。その際の vhdl 記述の一部を以下に記す。

```
begin
gi <= AIN and BIN;
pi <= AIN or BIN;

P0 <= pi(3) and pi(2) and pi(1) and pi(0);
P1 <= pi(7) and pi(6) and pi(5) and pi(4);
P2 <= pi(11) and pi(10) and pi(9) and pi(8);
P3 <= pi(15) and pi(14) and pi(13) and pi(12);
P4 <= pi(19) and pi(18) and pi(17) and pi(16);
P5 <= pi(23) and pi(22) and pi(21) and pi(20);
P6 <= pi(27) and pi(26) and pi(25) and pi(24);
P7 <= pi(31) and pi(30) and pi(29) and pi(28);

G0 <= gi(3) or ( pi(3) and gi(2) ) or
  ( pi(3) and pi(2) and gi(1) ) or
  ( pi(3) and pi(2) and pi(1) and gi(0) );
G1 <= gi(7) or ( pi(7) and gi(6) ) or
  ( pi(7) and pi(6) and gi(5) ) or
  ( pi(7) and pi(6) and pi(5) and gi(4) );
G2 <= gi(11) or ( pi(11) and gi(10) ) or
  ( pi(11) and pi(10) and gi(9) ) or
  ( pi(11) and pi(10) and pi(9) and gi(8) );
G3 <= gi(15) or ( pi(15) and gi(14) ) or
  ( pi(15) and pi(14) and gi(13) ) or
  ( pi(15) and pi(14) and pi(13) and gi(12) );
G4 <= gi(19) or ( pi(19) and gi(18) ) or
  ( pi(19) and pi(18) and gi(17) ) or
  ( pi(19) and pi(18) and pi(17) and gi(16) );
G5 <= gi(23) or ( pi(23) and gi(22) ) or
  ( pi(23) and pi(22) and gi(21) ) or
  ( pi(23) and pi(22) and pi(21) and gi(20) );
G6 <= gi(27) or ( pi(27) and gi(26) ) or
  ( pi(27) and pi(26) and gi(25) ) or
  ( pi(27) and pi(26) and pi(25) and gi(24) );
G7 <= gi(31) or ( pi(31) and gi(30) ) or
  ( pi(31) and pi(30) and gi(29) ) or
  ( pi(31) and pi(30) and pi(29) and gi(28) );

C1 <= G0;
C2 <= G1 or ( P1 and G0 );
C3 <= G2 or ( P2 and G1 ) or ( P2 and P1 and G0 );
C4 <= G3 or ( P3 and G2 ) or ( P3 and P2 and G1 ) or
          ( P3 and P2 and P1 and G0 );
C5 <= G4 or ( P4 and G3 ) or ( P4 and P3 and G2 ) or
  ( P4 and P3 and P2 and G1 ) or ( P4 and P3 and P2 and P1 and G0 );
C6 <= G5 or ( P5 and G4 ) or ( P5 and P4 and G3 ) or
  ( P5 and P4 and P3 and G2 ) or ( P5 and P4 and P3 and P2 and G1 ) or
  ( P5 and P4 and P3 and P2 and P1 and G0 );
C7 <= G6 or ( P6 and G5 ) or ( P6 and P5 and G4 ) or
```

```
  ( P6 and P5 and P4 and G3 ) or ( P6 and P5 and P4 and P3 and G2 ) or
  ( P6 and P5 and P4 and P3 and P2 and G1 ) or
  ( P6 and P5 and P4 and P3 and P2 and P1 and G0 );
C8 <= G7 or ( P7 and G6 ) or ( P7 and P6 and G5 ) or
  ( P7 and P6 and P5 and G4 ) or ( P7 and P6 and P5 and P4 and G3 ) or
  ( P7 and P6 and P5 and P4 and P3 and G2 ) or
  ( P7 and P6 and P5 and P4 and P3 and P2 and G1 ) or
  ( P7 and P6 and P5 and P4 and P3 and P2 and P1 and G0 );

YOUT(3 downto 0) <= ("0000") + AIN(3 downto 0) + BIN(3 downto 0);
YOUT(7 downto 4) <= ("000" & C1) + AIN(7 downto 4) + BIN(7 downto 4);
YOUT(11 downto 8) <= ("000" & C2) + AIN(11 downto 8) + BIN(11 downto 8);
YOUT(15 downto 12) <= ("000" & C3) + AIN(15 downto 12) + BIN(15 downto 12);
YOUT(19 downto 16) <= ("000" & C4) + AIN(19 downto 16) + BIN(19 downto 16);
YOUT(23 downto 20) <= ("000" & C5) + AIN(23 downto 20) + BIN(23 downto 20);
YOUT(27 downto 24) <= ("000" & C6) + AIN(27 downto 24) + BIN(27 downto 24);
YOUT(31 downto 28) <= ("000" & C7) + AIN(31 downto 28) + BIN(31 downto 28);
YOUT(32) <= C8;
```

その結果、合成した回路と回路規模、遅延時間は以下のようになった。
回路規模：775
遅延時間：4.90

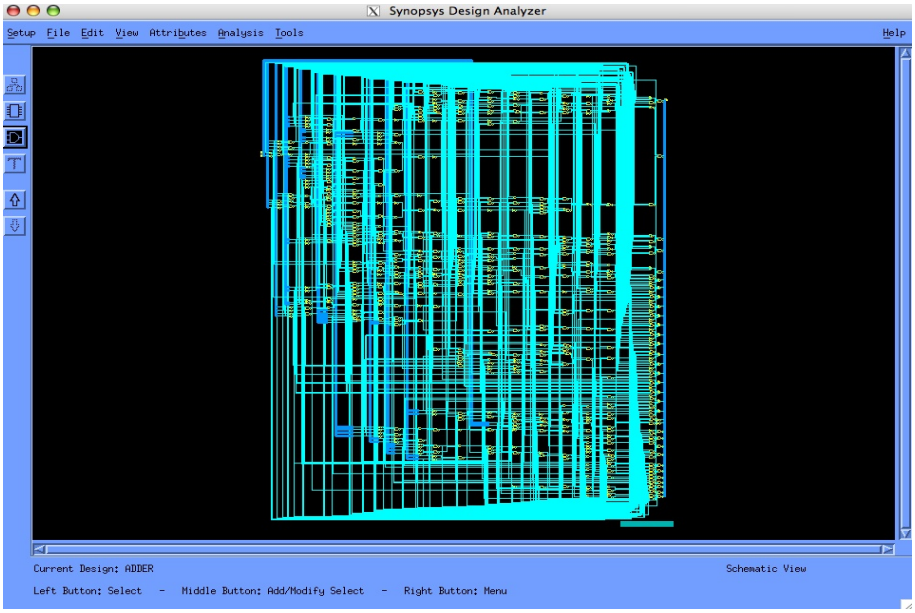

図 2: 合成した回路

両方の結果見てみると、予想に反しキャリー先見方式を用いた方が遅延時間が大きく、回路規模も小さくなっていた。

2　課題２

MINIMIPS 課題のバブルソートプログラムでは、データ RAM の 384、388、392 番地に演算に必要な数が格納されている。命令の 8、12、16 番地のロード命令によりこれらの数値がレジスタにコピーされ実際のソーティングが実行されている。教科書の P133 の例題に説明にあるように、lui 命令と addi 命令を使用すれば、データ RAM 内に数値を用意しなくても命令中に数値を用意してレジスタに適切な値を設定可能である。

1 minimips.vhd を改造して、lui、addi の２つの命令も実行できるようにせよ。

2 データ RAM の 384、388、392 番地の値を使用せずに同様のバブルソートを実行せよ。

3 改造後のプログラム及びプロセッサを用いて、32 ワードのバブルソートを実行し、総実行サイクル数を報告せよ。

4 レポートには、改造後のアセンブラプログラム、改造後の minimips 記述、総実行サイクル数を含むこと。

2.1　命令の追加

まず、alu_pkg.vhd に以下のように書き加えた。

```
constant OP_LUI  : std_logic_vector (5 downto 0) := "001111";
constant OP_ADDI  : std_logic_vector (5 downto 0) := "001000";
```

次に、minimips.vhd 内にも以下のように書き加えた

```
---------------------------------
-- RFILE
---------------------------------
    RFILE_WT: process (Clock)
    begin
if rising_edge (Clock) then
    if (opcode = OP_ALU) and (rd /= R0) then
    reg(conv_integer(rd)) <= alu_rst;
            elsif (opcode = OP_LW) and (rt /= R0) then
    reg(conv_integer(rt)) <= Rddata;
elsif ( ( (opcode = OP_ADDI) or (opcode = OP_LUI) )
and (rt /= R0) ) then
  reg(conv_integer(rt)) <= alu_rst;
            else   null;

～～略～～
---------------------------------
-- ALU
---------------------------------
    ALU: process (Inst,regout1,regout2,sgnexd)
    begin

～～略～～

end case;
elsif ((opcode = OP_LW) or (opcode = OP_SW)) then
alu_rst  <= regout1 + sgnexd;
```

```
elsif (opcode = OP_BEQ) then
    alu_rst  <= regout1 - regout2;
elsif (opcode = OP_LUI) then
alu_rst <= Inst(15 downto 0) & "0000000000000000";
elsif (opcode = OP_ADDI) then
alu_rst <= regout1 + sgnexd;
        else
    alu_rst  <= (others => 'X');
end if;
    end process ALU;
```

2.2 バブルソートプログラムの実行

irom.vhd 及び dram.vhd 内の記述を修正してソーティングプログラムを改造し、バブルソートプログラムを実行した。以下に、修正した箇所を記す。

- dram.vhd

```
32 => conv_std_logic_vector (0,32),   -- start byte address of the array
33 => conv_std_logic_vector (0 ,32),   -- size of array in byte, 8 word
34 => conv_std_logic_vector (0,  32),   -- size of word in byte
```

- irom.vhd

```
------------------------------
-- READ ONLY MEMORY
------------------------------
constant ROM  : MemVecArr  :=
(0 => OP_ALU & R0  & R0  & R0  & "00000" & FN_ADD, -- NO operation
 1 => OP_ALU & R0  & R0  & R0  & "00000" & FN_ADD, -- NO operation
 2 => OP_LUI & R0 & R1 & "0000000000000000",  -- LUI R1, 0
 3 => OP_ADDI  & R0  & R1  & "0000000010000000",     -- ADDI  R1, R0, 128
 4 => OP_ADDI  & R0  & R2  & "0000000100000000",     -- ADDI  R2, R0, 256
 5 => OP_ADDI  & R0  & R3  & "0000000000000100",     -- ADDI  R3, R0, 4
 6 => OP_ALU & R1  & R2  & R5  & "00000" & FN_ADD, -- ADD R5,R1,R2
 7 => OP_ALU & R5  & R3  & R5  & "00000" & FN_SUB, -- SUB R5,R5,R3
 8 => OP_ALU & R2  & R0  & R6  & "00000" & FN_ADD, -- ADD R6,R2,R0
 9 => OP_ALU & R6  & R0  & R7  & "00000" & FN_ADD, -- ADD R7,R6,R0
10 => OP_ALU & R7  & R3  & R8  & "00000" & FN_ADD, -- ADD R8,R7,R3
11 => OP_LW  & R7  & R10 & "0000000000000000",     -- LW  R10, 0(R7)
12 => OP_LW  & R8  & R11 & "0000000000000000",     -- LW  R11, 0(R8)
13 => OP_ALU & R10 & R11 & R9  & "00000" & FN_SLT, -- SLT R9,R10,R11
14 => OP_BEQ & R9  & R0  & "0000000000000010",     -- BEQ R9,R0, +2
15 => OP_SW  & R8  & R10 & "0000000000000000",     -- SW  R10, 0(R8)
16 => OP_SW  & R7  & R11 & "0000000000000000",     -- SW  R11, 0(R7)
17 => OP_ALU & R7  & R3  & R7  & "00000" & FN_ADD, -- ADD R7,R7,R3
18 => OP_BEQ & R7  & R5  & "0000000000000001",     -- BEQ R7,R5, +1
19 => OP_J   & "00000000000000000000001010",       -- J   10
20 => OP_ALU & R5  & R3  & R5  & "00000" & FN_SUB, -- SUB R5,R5,R3
21 => OP_BEQ & R5  & R2  & "0000000000000001",     -- BEQ R5,R2, +1
22 => OP_J   & "00000000000000000000001001",       -- J   9
others => OP_ALU & R0 & R0 & R0 & "00000" & FN_ADD); -- NOoperation
```

2.3　実行結果

修正したソーティングプログラムを実行した結果を下図に示す。

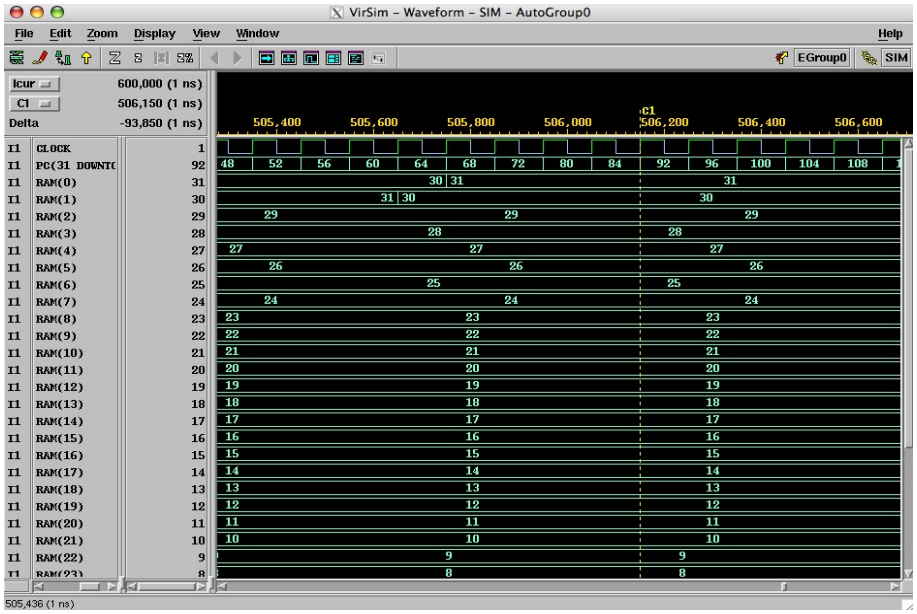

図 3: 実行結果

次に、総実行サイクル数を出す。

- 開始時間：250ns
- 終了時間：506200ns（PC=92 になった時）
- クロック：100ns

従って総実行サイクル数は、5059.5 となる。

3　ソースの場所

課題 1 と課題 2 で設計した VHDL 記述は下記の場所参照。
`naha:/net/home/y03/j03043/digital/`

- 単純な加算器：adder.vhd、adder_test.vhd
- キャーリー先見：adder2.vhd、adder2_test.vhd
- MINIMIPS:alu_pkg.vhd、irom.vhd、dram.vhd、minimips.vhd、test_minimips.vhd