

システム設計演習(前期分)

第4回

秋田純一

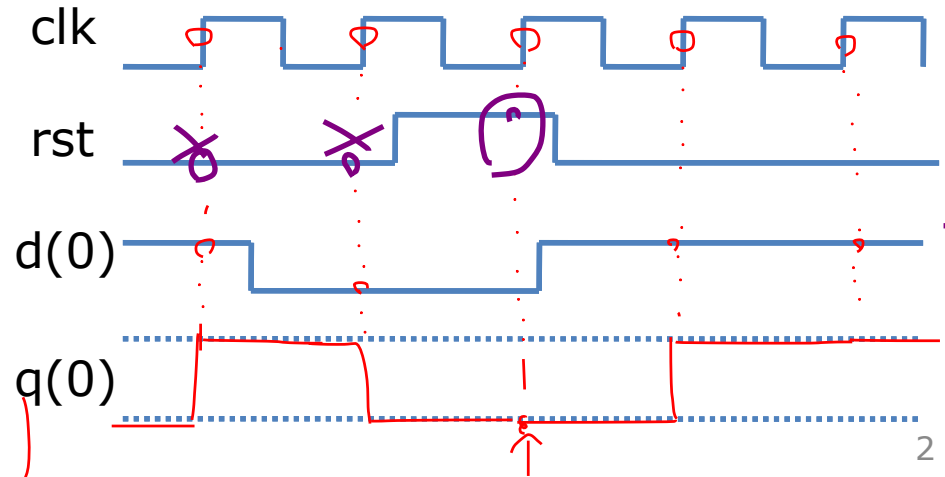
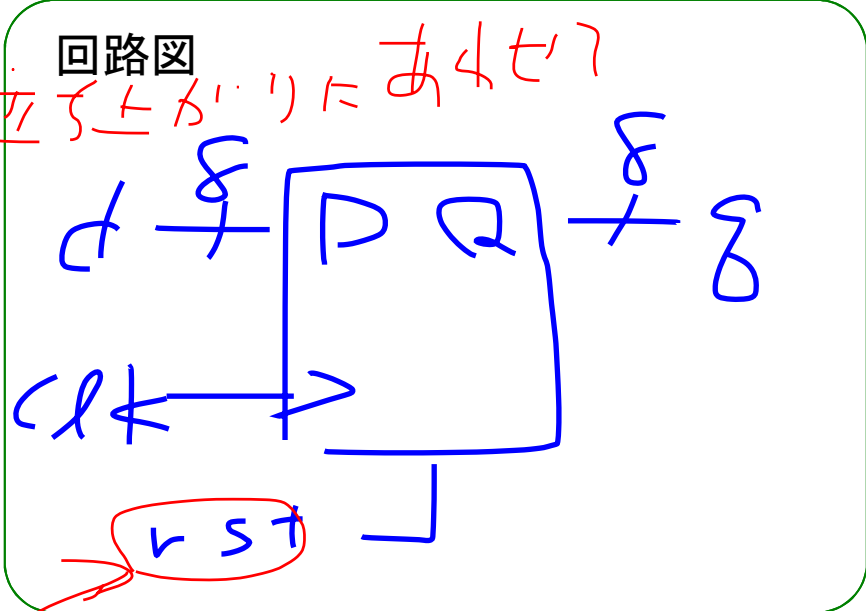
<http://j.mp/akita-class>

akita@ifdl.jp (@akita11)

HDL記述の例:同期リセットつきレジスタ(p.79)

```
entity sync_reg8 is
  port (
    clk, rst: in std_logic;
    d: in std_logic_vector(7 downto 0);
    q: out std_logic_vector(7 downto 0)
  );
end sync_reg8;
```

```
architecture arch of sync_reg8 is
begin
  process (clk) begin
    if (clk'event and clk = '1') then
      if (rst = '1') then
        q <= "00000000";
      else
        q <= d;
      end if;
    end if;
  end process;
end arch;
```



リセット (0 = 0)
 (0 = 0, 1 = 1)

実際には 0 状態

リセット (0 = 0)

同期リセットつきレジスタのポイント

- ☑ 順序回路では、基本的に
「クロック(clk)の立ち上がり」で動作
 - ☑ そのタイミングで出力や変数が変化する
 - ☑ 「if (clk'event and clk = '1')」でそのタイミングをつかまえる
 - ☑ この例では、出力qが入力dの値になる
=Dフリップフロップの動作(の8bit分)
- ☑ リセット(rst)のタイミングに注意
 - ☑ この例では、clkの立ち上がり時にrst=1ならばリセット(出力qが0になる) = 同期リセット

HDL記述の例: 非同期リセットつきレジスタ(p.80)

```
entity async_reg8 is
  port (
    clk, rst: in std_logic;
    d: in std_logic_vector(7 downto 0);
    q: out std_logic_vector(7 downto 0)
  );
end async_reg8;
```

```
architecture arch of async_reg8 is
begin
  process (clk, rst) begin
    if (rst = '1') then
      q <= "0000000";
    elsif (clk'event and clk = '1') then
      q <= d;
    end if;
  end process;
end arch;
```

セリ・リ・セ・ト・リ・ス・ト

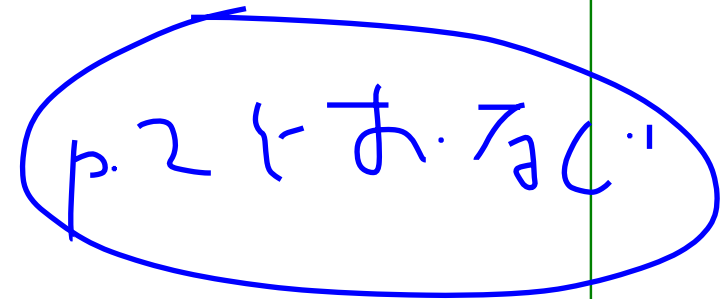
← 00000000 ← '1' のとき

← 00000000 ← '1' のとき

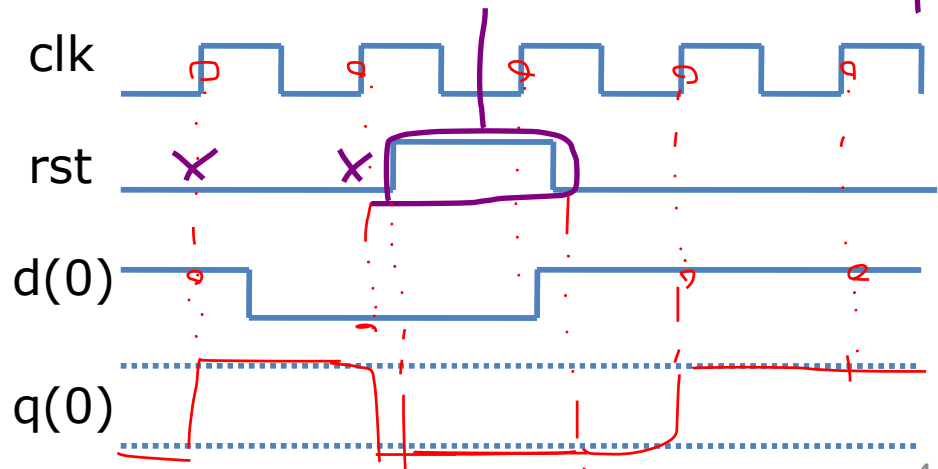
← 00000000 ← '1' のとき

'1' のとき、リセット

回路図



rst が '1' のときは、リセット



'1' のとき

非同期リセットつきレジスタのポイント

✓動作は「同期リセットつきレジスタ」と似ている

✓唯一の違いはリセットのタイミング

✓センシティブティ・リストがclkとrst

=rstが変化してもprocess内の動作を行う

✓rst=1ならば、出力qが0になる(リセット)

✓「clkの立ち上がり」でなくともリセットが起こる

④に $rst = 1$ をかかると

$rst = 1$ になったときに

if (rst = 1) q ← 0 とする

のでリセットもあつた

HDL記述の例: カウンタ

```
entity count8 is
  port (
    clk, rst: in std_logic;
    q: out std_logic_vector(7 downto 0);
  );
end count8;
```

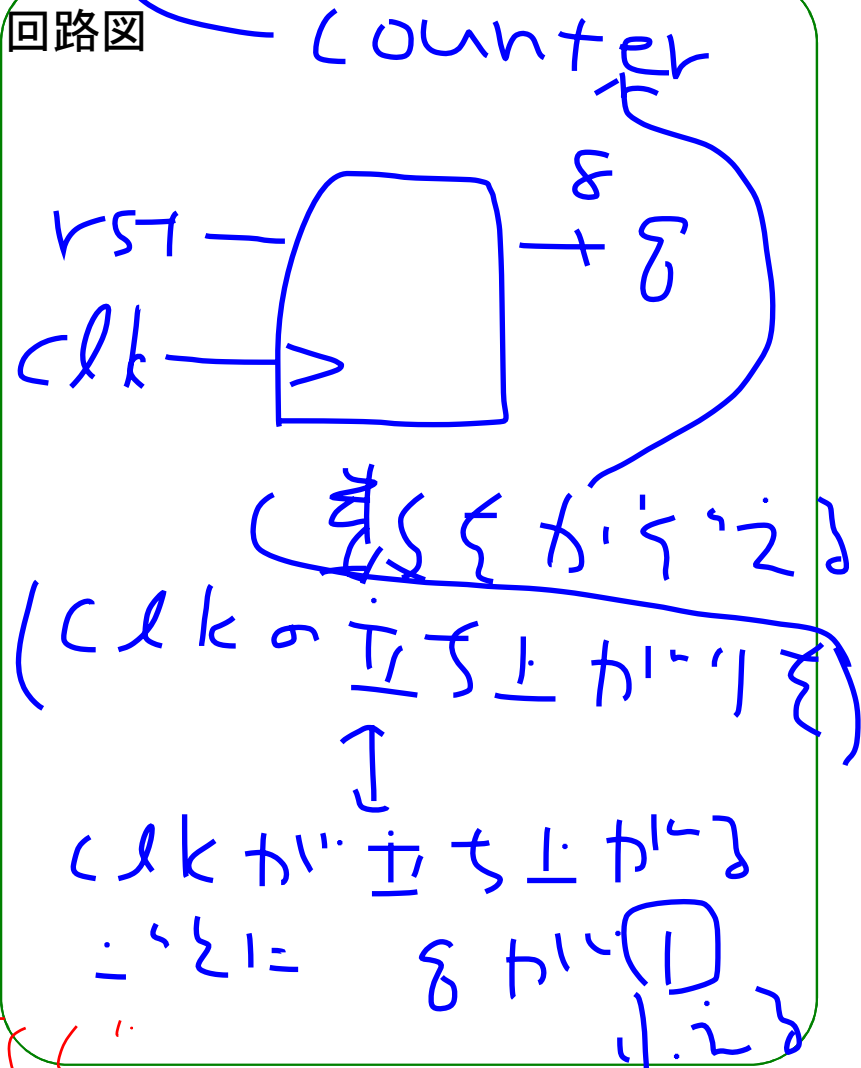
リセット

```
architecture arch of count8 is
  signal q_reg: std_logic_vector(7 downto 0);
begin
  process (clk) begin
    if (clk'event and clk = '1') then
      if (rst = '1') then
        q_reg <= "00000000";
      else
        q_reg <= q_reg + 1;
      end if;
    end if;
  end process;
  q <= q_reg;
end arch;
```

リセット時に
00000000にリセット
同期
1.1.2

q-regを出力する

回路図



カウンタのHDL記述のポイント

- ✓ signal(変数)のq_regを、clkの立ち上がりのタイミングで+1している
=q_regの値は「clk立ち上がりの数」
=カウンタの動作
- ✓ q_regの値を、別途出力qに与えている
- ✓ ※カルノー図を使ってカウンタ回路を順序回路として設計するより格段に楽

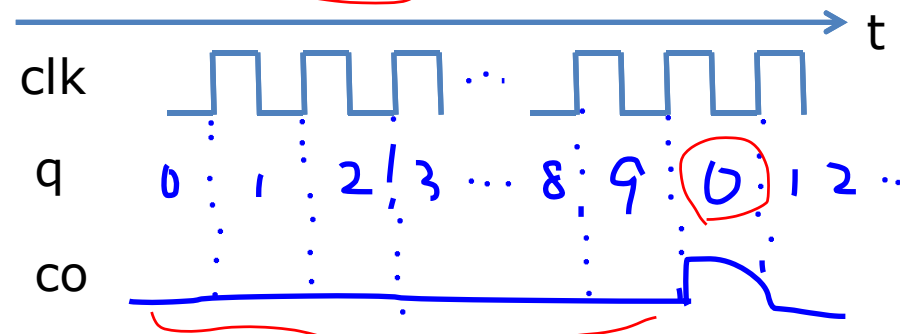
カウンタの応用: 10進カウンタ

```
entity count10 is
  port (
    clk, rst: in std_logic;
    q: out std_logic_vector(3 downto 0);
    co: out std_logic
  );
end count10;
```

```
architecture arch of count10 is
  signal q_reg: std_logic_vector(3 downto 0);
begin
  process (clk) begin
    if (clk'event and clk = '1') then
      if (rst = '1') then
        q_reg <= "0000"; co <= '0';
      elsif (q_reg = 9) then
        q_reg <= "0000"; co <= '1';
      else
        q_reg <= q_reg + 1; co <= '0';
      end if;
    end if;
  end process;
  q <= q_reg;
end arch;
```

- ✓ 基本的にはカウンタ (同期リセット)
- ✓ 値(q_reg)が9なら、次は0にする
- ✓ それ以外は+1
- ✓ co (繰り上がり)信号
- ✓ 9→0に繰り上がる時だけ'1'にする

4ビット 0000
1111
← 10進
a 16 (繰り)



10進には 4bit
61 = 24 は 7 (繰り上がり) (clk の 1/10 分)

HDL記述の例 : レジスタファイル(p.82)

```
entity regfile is
  port (
    clk, rst: in std_logic;
    d: in std_logic_vector(7 downto 0);
    n: in std_logic_vector(2 downto 0);
    w: in std_logic;
    q: out std_logic_vector(7 downto 0);
  );
end regfile;

architecture arch of regfile is
  type t_reg_f is array (integer range 0 to 7)
    of std_logic_vector(7 downto 0);
  signal regf: t_reg_f;
begin
  process (clk) begin
    if (clk'event and clk = '1') then
      if (rst = '1') then
        for i in 0 to 7 loop
          regf(i) <= "00000000";
        end loop;
      elsif (w = '1') then
        regf(to_integer(n)) <= d;
      end if;
    end if;
  end process;
  q <= regf(to_integer(n));
end arch;
```

回路図

レジスタファイルのHDL記述のポイント

- ☑ 一種の「メモリ」
 - ☑ アドレス(n)で指定された変数(レジスタ)を読み書きできる
 - ☑ $w=1$ ならば入力dの値を指定した場所に書き込み
 - ☑ 出力qは指定した場所の値が現れる
- ☑ (細かいテクニック: VHDLの文法)
 - ☑ 配列変数の添え字は整数だが、アドレスnは `std_logic_vector`なので、`to_integer()`で型変換
 - ☑ `regf`がレジスタファイル(メモリ)の実体配列だが、`type`文で8ビット×8個の配列を`t_reg`という型宣言してそれを使っている

ステートマシンのHDL記述のポイント

- ☑ ステートマシン(状態遷移回路)
 - ☑ 現状態(state)と入力の値から、次に移る状態(次状態)と出力の値が決まる
 - ☑ 次状態への移動(遷移)は、clkの立ち上がりで起こる
 - ☑ 古典的な設計法:
 - ☑ 次状態 ← 現状態と入力の関係式
 - ☑ 出力 ← 現状態と入力の関係式
 - ☑ 現状態 = Dフリップフロップ(レジスタ)の出力
 - ☑ 次状態 = Dフリップフロップ(レジスタ)の入力